MICROPROCESSORS & MICROCONTROLLERS

LECTURE NOTES

B.TECH

Prepared by:
Mrs.G.Jyothi, Assistant Professor

Department of Electronics & Communication Engineering
MALLA REDDY ENGINEERING COLLEGE
(Autonomous)
(Approved by AICTE & Affiliated to JNTUH)
Maisammaguda, Dhulapally (Post via Kompally), Secunderabad-500 100

2018-19 Onwards (MR-18) **MALLA REDDY ENGINEERING COLLEGE (Autonomous)**

Code: 80414 **B.Tech. V Semester** **L T P**

Credits: 3 3 - -

MICROPROCESSORS AND MICROCONTROLLERS

Pre-Requisites: Digital Electronics.

Course Objectives: This course provides the students to understand operation and programming of 8085 Microprocessor, develops real time applications using 8086 processor, understand the basic concepts of 8051 Microcontroller and interfacing with I/O devices.

MODULE I: 8085 Architecture [8 Periods]

Introduction to Microprocessors, Architecture of 8085, Pin Configuration and Function, internal register & flag register, Generation of Control Signals: Bus Timings: Demultiplexing of address/ data bus; Fetch Cycle, Execute Cycle, Instruction Cycle, Machine cycles, T-states, memory interfacing.

MODULE II: Instruction Set and Programming with 8085 [10 Periods]

Instruction for Data Transfer, Arithmetic and Logical Operations, Branching Operation, Machine Cycle Concept, Addressing Modes, Instructions Format, Stacks, Subroutine and Related Instructions, Elementary Concepts of Assemblers, Assembler Directives, Looping and Counting, Software Counters with Time Delays, Simple Programs using Instruction Set of 8085, Debugging, Programs Involving Subroutines, Programs for Code Conversion e.g. BCD to Binary, Binary to BCD, Binary to Seven-Segment LED Display.Binary to ASCII, ASCII to Binary, Program for Addition Subtraction, Programs for Multiplication and Division of Unsigned Binary Numbers.

MODULE III: 8086 Architecture [09 Periods]

A:8086 Architecture-Functional diagram, Register Organization, Memory Segmentation,Programming Model, Memory addresses, Physical Memory Organization, Architecture of 8086, Signal descriptions of 8086- Common Function Signals, Timing diagrams, Interrupts of 8086.

B: Interfacing I/O Devices: Interfacing of 8086 with Memory, key board and display,A/D and D/A.

MODULE IV: Introduction to Microcontroller [10 Periods]

A brief History of Microcontrollers, Harvard Vs Von-Neumann Architecture; RISC VsCISC, Classification of MCS-51family based on their features (8051,8052, 8031, 8751,AT89C51), Pin configuration of 8051.

8051 Microcontroller Architecture and Instruction Set: Registers of 8051, Inbuilt RAM, Register banks, stack, on-chip and external program code memory ROM, power reset and clocking circuits, I/O port structure, addressing modes, Instruction set and programming.

MODULE V: 8051 Real Time Control [11 Periods]

Counter/Timer and Interrupts of 8051: Introduction, Registers of timer/counter, Different modes of timer/counter, Timer/counter programming, Interrupt *Vs* Polling, Types of interrupts and vector addresses, register used for interrupts initialization, programming of external

interrupts, Timer interrupts.
Asynchronous Serial Communication and Programming: Introduction to serial communication, Programming the Serial Communication Interrupts, RS232 standard,RS422 Standard, RS-485 standard, Max 232/233 Driver.
Interfacing with 8051: Interfacing and programming of: ADC (0804,0808/0809,0848) & DAC(0808), dc motor, stepper motor, Relays, LED and Seven segment display, LCD, 4x4 keyboard matrix.

Text Books:
1. Ramesh Gaonkar, “Microprocessor Architecture, Programing and Application with 8085” , Penram, 5th Edition, 2002.
2. A.K.Ray, “Advanced Micro processors and Peripherals” 3rd Tata McGraw-Hill,Edition.
3. Mazidi, Mazidi&McKinlay, “The 8051 Microcontroller and Embedded Systems using Assembly and C” 2nd Edition,PHI.

Reference Books:
1. D. V Hall TMH, “Microprocessors and Interfacing” 2nd Edition, 2006
2. K. Uday Kumar, B.S. Umashankar, “The 8085 Microprocessor: Architecture, programming and Interfacing” Pearson, 2008.
3. Liu and Gibson, “Micro Computer System 8086/8088 Family Architecture, Programming and Design” PHI, 2nd Edition
4. Kenneth. J. Ayala, Cengage Learning, “The 8051 Microcontroller” 3rd Edition, 2004.

E-Resources:
1. https://www.tutorialspoint.com › Microprocessor › Microprocessor – 8085 Architecture
2. http://www.cpu-world.com/CPUs/8086/
3. https://www.journals.elsevier.com/microprocessors-and-microsystems/
4. http://rtcmagazine.com/technologies/view/Microcontrollers
5. http://nptel.ac.in/courses/106108100/
6. http://nptel.ac.in/courses/108107029/
7. nptel.ac.in/courses/106108100/

Module I
8085 Architecture

INTRODUCTION TO MICROPROCESSOR AND MICROCOMPUTER ARCHITECTURE

A microprocessor is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer. Any microprocessor-based systems having limited number of resources are called microcomputers. Nowadays, microprocessor can be seen in almost all types of electronics devices like mobile phones, printers, washing machines etc.

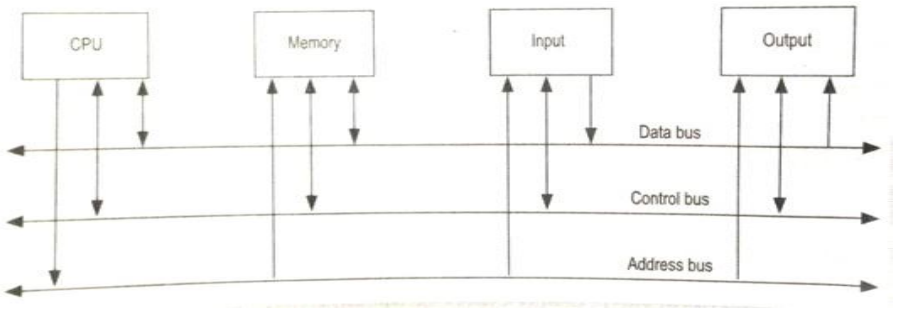


Fig.1 Microprocessor-based system

- **Address Bus**: It carries the address, which is a unique binary pattern used to identify a memory location or an I/O port. For example, an eight bit address bus has eight lines and thus it can address 2^8 = 256 different locations. The locations in hexadecimal format can be written as 00H – FFH.
- **Data Bus**: The data bus is used to transfer data between memory and processor or between I/O device and processor. For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.
- **Control Bus**: The control bus carry control signals, which consists of signals for selection of memory or I/O device from the given address, direction of data transfer and synchronization of data transfer in case of slow devices.

A typical microprocessor consists of arithmetic and logic unit (ALU) in association with control unit to process the instruction execution. Almost all the microprocessors are based on the principle of store-program concept. In store-program concept, programs or instructions are sequentially stored in the memory locations that are to be executed. To do any task using a microprocessor, it is to be programmed by the user. So the programmer must have idea about its internal resources, features and supported instructions. Each microprocessor has a set of instructions, a list which is provided by the microprocessor manufacturer. The instruction set of a microprocessor is provided in two forms: binary machine code and mnemonics.

Microprocessor communicates and operates in binary numbers 0 and 1. The set of instructions in the form of binary patterns is called a machine language and it is difficult for us to understand. Therefore, the binary patterns are given abbreviated names, called mnemonics, which forms the assembly language. The conversion of assembly-level language into binary machine-level language is done by using an application called assembler.

Evolution of Microprocessors

4-bit Microprocessors

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 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.

8-bit Microprocessors

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.

16-bit Microprocessors

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

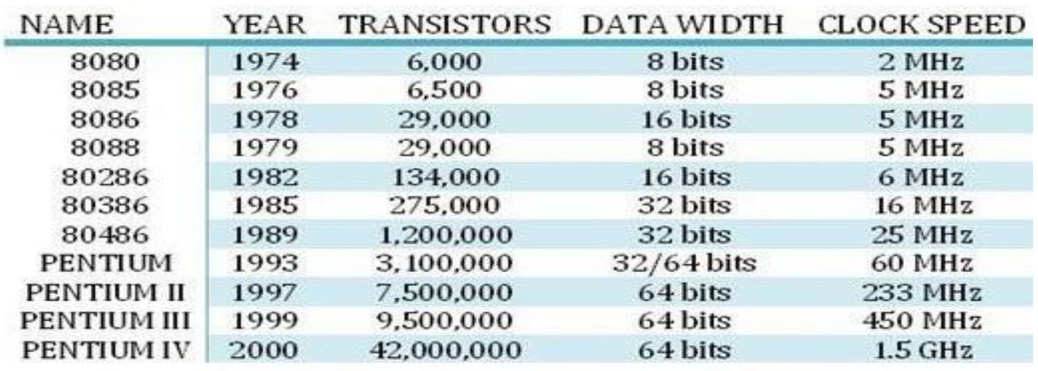
32-bit Microprocessors

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

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 multiple CPUs in a single system in order to achieve multiprocessing. The MMX extension was added to Pentium Pro and the result was Pentiuum II.

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 makes the Pentium III faster than high-end RISC CPUs.



NAME	YEAR	TRANSISTORS	DATA WIDTH	CLOCK SPEED
8080	1974	6,000	8 bits	2 MHz
8085	1976	6,500	8 bits	5 MHz
8086	1978	29,000	16 bits	5 MHz
8088	1979	29,000	8 bits	5 MHz
80286	1982	134,000	16 bits	6 MHz
80386	1985	275,000	32 bits	16 MHz
80486	1989	1,200,000	32 bits	25 MHz
PENTIUM	1993	3,100,000	32/64 bits	60 MHz
PENTIUM II	1997	7,500,000	64 bits	233 MHz
PENTIUM III	1999	9,500,000	64 bits	450 MHz
PENTIUM IV	2000	42,000,000	64 bits	1.5 GHz

ARCHITECTURE OF 8085 MICROPROCESSOR

The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power. It can run at a maximum frequency of 3 MHz. Its data bus width is 8-bit and address bus width is 16-bit, thus it can address 2^{16} = 64 KB of memory. The internal architecture of 8085 is shown is Fig. 2.

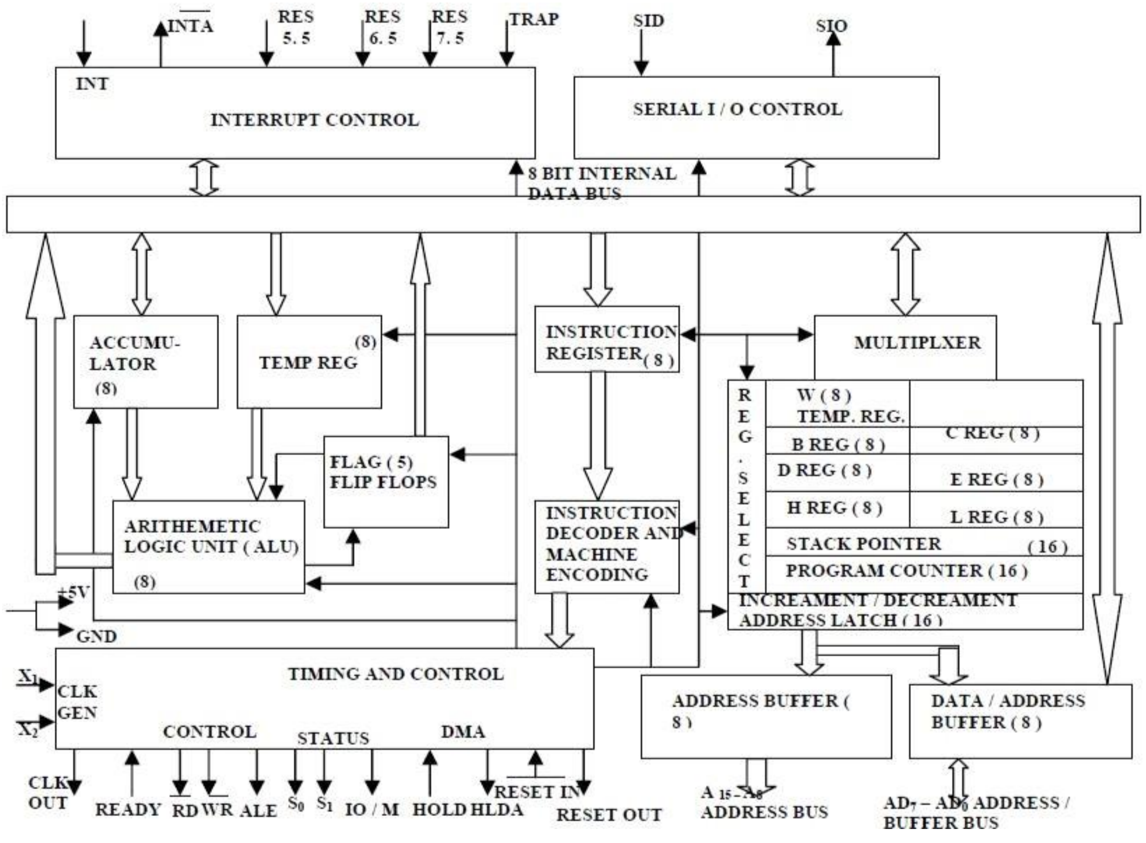


Fig. 2 Internal Architecture of 8085

Arithmetic and Logic Unit

The ALU performs the arithmetic and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc. It uses data from memory and from Accumulator to perform operations. The results of the arithmetic and logical operations are stored in the accumulator.

Registers

The 8085 includes six registers, one accumulator and one flag register, as shown in Fig. 3. In addition, it has two 16-bit registers: stack pointer and program counter. They are briefly described as follows.

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some 16-bit operations. The programmer can use these registers to store or copy data into the register by using data copy instructions.

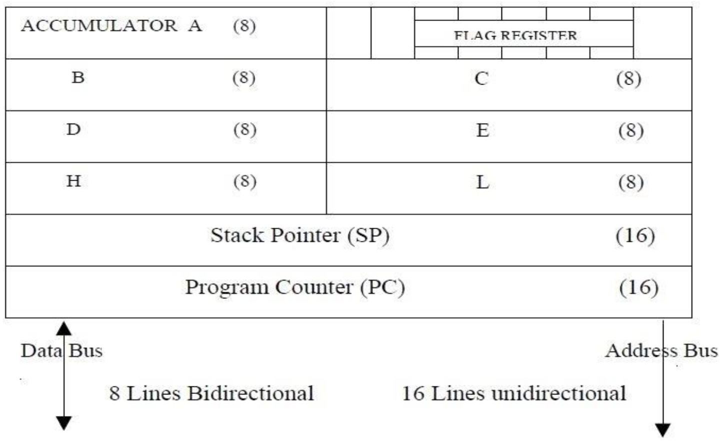


Fig. 3 Register organization

Accumulator

The accumulator is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flag register

The ALU includes five flip-flops, which are set or reset after an operation according to data condition of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. Their bit positions in the flag register are shown in Fig. 4. The microprocessor uses these flags to test data conditions.

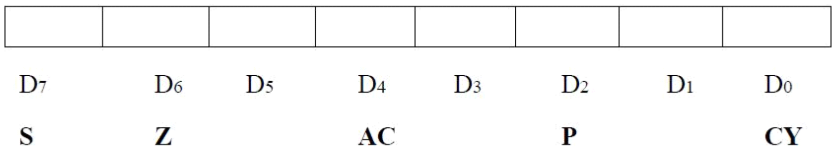


Fig. 4 Flag register

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the program counter is automatically incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register, used as a memory pointer. It points to a memory location in R/W memory, called stack. The beginning of the stack is defined by loading 16- bit address in the stack pointer.

Instruction Register/Decoder

It is an 8-bit register that temporarily stores the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.

Control Unit

Generates signals on data bus, address bus and control bus within microprocessor to carry out the instruction, which has been decoded. Typical buses and their timing are described as follows:

- **Data Bus:** Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit.
- **Address Bus:** The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time-shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.
- **Control Bus:** Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carries control signals partly unidirectional and partly bidirectional. The following control and status signals are used by 8085 processor:
    - ALE (output): Address Latch Enable is a pulse that is provided when an address appears on the AD0 – AD7 lines, after which it becomes 0.
    - RD (active low output): The Read signal indicates that data are being read from the selected I/O or memory device and that they are available on the data bus.
    - WR (active low output): The Write signal indicates that data on the data bus are to be written into a selected memory or I/O location.
    - IO/M (output): It is a signal that distinguished between a memory operation and an I/O operation. When IO/M = 0 it is a memory operation and IO/M = 1 it is an I/O operation.
    - S1 and S0 (output): These are status signals used to specify the type of operation being performed; they are listed in Table 1.

Table 1 Status signals and associated operations

S1	S0	States
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

2. Bus organization

The schematic representation of the 8085 bus structure is as shown in Fig. 5. The microprocessor performs primarily four operations:

- Memory Read: Reads data (or instruction) from memory.
- Memory Write: Writes data (or instruction) into memory.
- I/O Read: Accepts data from input device.
- I/O Write: Sends data to output device.

The 8085 processor performs these functions using address bus, data bus and control bus as shown in Fig. 5.

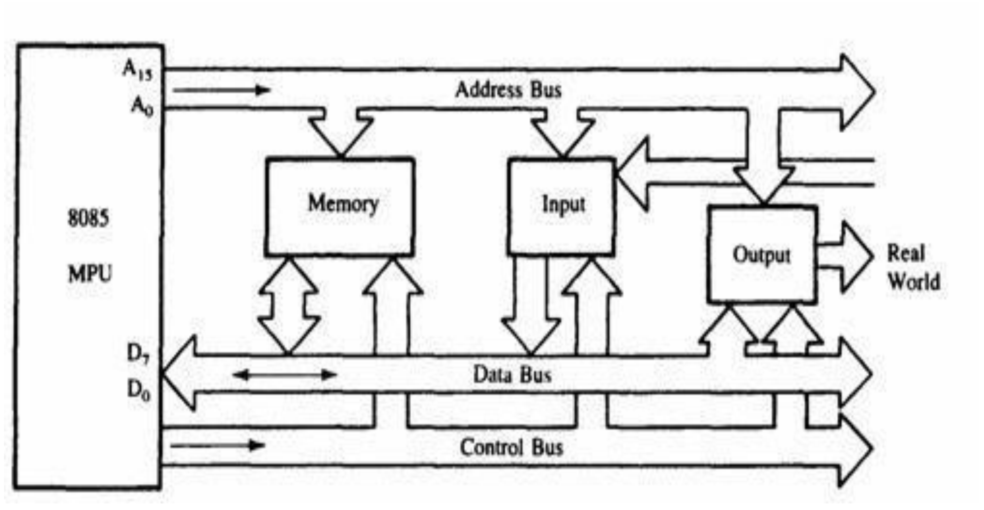


Fig. 5 The 8085 bus structure

8085 PIN DESCRIPTION

Features:

- It is an 8-bit microprocessor
- Manufactured with N-MOS technology
- 40 pin IC package
- It has 16-bit address bus and thus has 2^{16} = 64 KB addressing capability.
- Operate with 3 MHz single-phase clock
- +5 V single power supply

The logic pin layout and signal groups of the 8085nmicroprocessor are shown in Fig. 6. All the signals are classified into six groups:

- Address bus
- Data bus
- Control & status signals
- Power supply and frequency signals
- Externally initiated signals
- Serial I/O signals

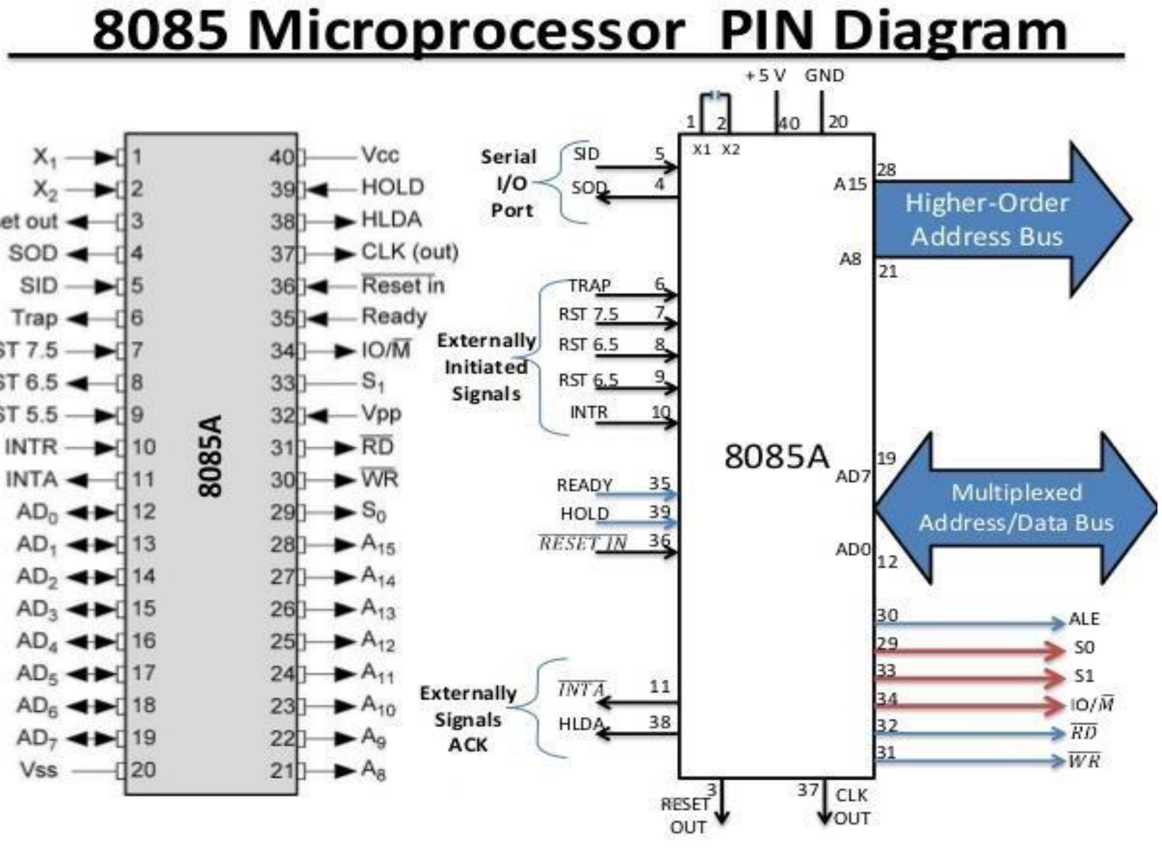

8085 Microprocessor PIN Diagram
8085A
X1, X2, Reset out, SOD, SID, Trap, RST 7.5, RST 6.5, RST 5.5, INTR, INTA, AD0–AD7, Vss
Vcc, HOLD, HLDA, CLK (out), Reset in, Ready, IO/M, S1, Vpp, RD, WR, S0, A15–A8
+5 V
GND
Serial I/O Port
Externally Initiated Signals
Externally Signals ACK
Higher-Order Address Bus
Multiplexed Address/Data Bus
ALE
S0
S1
IO/M
RD
WR
RESET OUT
CLK OUT

Fig. 6 8085 microprocessor pin layout and signal groups

Address and Data Buses:

- **A8 – A15 (output):** Most significant eight bits of memory addresses and the eight bits of the I/O addresses.
- **AD0 – AD7 (input/output):** Lower significant bits of memory addresses and the eight bits of the I/O addresses during first clock cycle. Behaves as data bus during third and fourth clock cycle.

Control & Status Signals:

- **ALE:** Address latch enable
- **RD** : Read control signal.
- **WR :** Write control signal.
- **IO/M, S1 and S0:** Status signals. Power Supply & Clock Frequency:
- **Vcc:** +5 V power supply
- **Vss:** Ground reference
- **X1, X2:** A crystal having frequency of 6 MHz is connected at these two pins
- **CLK:** Clock output

Externally Initiated and Interrupt Signals:

- **RESET IN**: When the signal on this pin is low, the PC is set to 0 and the processor is reset.
- **RESET OUT:** This signal indicates that the processor is being reset. The signal can be used to reset other devices.
- **READY:** When this signal is low, the processor waits for an integral number of clock cycles until

it goes high.

- **HOLD:** This signal indicates that a peripheral like DMA (direct memory access) controller is requesting the use of address and data bus.
- **HLDA:** This signal acknowledges the HOLD request.
- **INTR:** Interrupt request is a general-purpose interrupt.
- **INTA:** This is used to acknowledge an interrupt.
- **RST 7.5, RST 6.5, RST 5,5** – restart interrupt: These are vectored interrupts and have highest priority than INTR interrupt.
- **TRAP:** This is a non-maskable interrupt and has the highest priority.

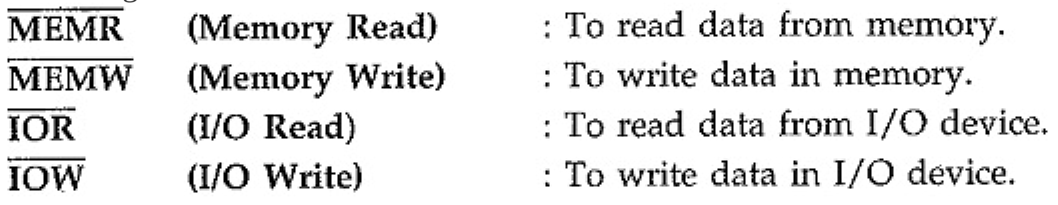
Serial I/O Signals:

- **SID:** Serial input signal. Bit on this line is loaded to D7 bit of register A using RIM instruction.
- **SOD:** Serial output signal. Output SOD is set or reset by using SIM instruction.

Generation of Control signals:

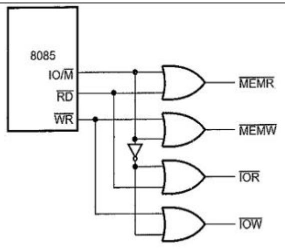
The 8085 Microprocessor provides RD and WR signals to initiate read or write cycle. Because these Control Signals of 8085 are used both for reading/writing memory and for reading/writing an input device, it is necessary to generate separate read and write signals for memory and I/O device

The 8085 provides IO/M signal to indicate whether the initiated cycle is for I/O device or for memory device. Using IO/M signal along with RD and WR, it is possible to generate separate four Control Signals of 8085 :



$\overline{MEMR}$	(Memory Read)	: To read data from memory.
$\overline{MEMW}$	(Memory Write)	: To write data in memory.
$\overline{IOR}$	(I/O Read)	: To read data from I/O device.
$\overline{IOW}$	(I/O Write)	: To write data in I/O device.

Fig. shows the circuit which generates MEMR, MEMW, IOR and IOW signals.



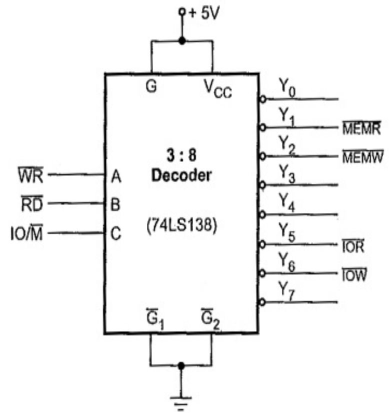
We know that for OR gate, when both the inputs are low then only output is low.

IO/$\overline{\text{M}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	$\overline{\text{MEMR}}$ $\overline{\text{RD}}$ + IO/$\overline{\text{M}}$	$\overline{\text{MEMW}}$ $\overline{\text{WR}}$ + IO/$\overline{\text{M}}$	$\overline{\text{IOR}}$ $\overline{\text{RD}}$ + $\overline{\text{IO/M}}$	$\overline{\text{IOW}}$ $\overline{\text{WR}}$ + $\overline{\text{IO/M}}$
0	0	0	Condition never exists, because $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals does not go low simultaneously			
0	0	1	0	1	1	1
0	1	0	1	0	1	1
0	1	1	1	1	1	1
1	0	0	Condition never exists, because $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals does not go low simultaneously			
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	1	1	1

The signal IO/M goes low for memory operation. This signal is logically ORed with RD and WR to get MEMR and MEMW signals. When both RD and IO/M signals go low, MEMR signal goes low.

Similarly, when both WR and IO/M Signals go low, MEMW signal goes low. To generate IOR and IOW signals for I/O operation, IO/M signal is first inverted and then logically ORed with RD and WR signals.

Same truth table can be implemented using 3:8 decoder as shown in Fig



INSTRUCTION EXECUTION AND TIMING DIAGRAM:

Each instruction in 8085 microprocessor consists of two part- operation code (opcode) and operand. The opcode is a command such as ADD and the operand is an object to be operated on, such as a byte or the content of a register.

Instruction Cycle: The time taken by the processor to complete the execution of an instruction. An instruction cycle consists of one to six machine cycles.

Machine Cycle: The time required to complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.

T-State: Time corresponding to one clock period. It is the basic unit to calculate execution of instructions or programs in a processor.

To execute a program, 8085 performs various operations as:

- Opcode fetch

- Operand fetch
- Memory read/write
- I/O read/write

External communication functions are:

- Memory read/write
- I/O read/write
- Interrupt request acknowledge

Opcode Fetch Machine Cycle:

It is the first step in the execution of any instruction. The timing diagram of this cycle is given in Fig. 7.

The following points explain the various operations that take place and the signals that are changed during the execution of opcode fetch machine cycle:

T1 clock cycle

i. The content of PC is placed in the address bus; AD0 - AD7 lines contains lower bit address and A8 – A15 contains higher bit address.
ii. IO/$\overline{\text{M}}$ signal is low indicating that a memory location is being accessed. S1 and S0 also changed to the levels as indicated in Table 1.
iii. ALE is high, indicates that multiplexed AD0 – AD7 act as lower order bus.

T2 clock cycle

i. Multiplexed address bus is now changed to data bus.
ii. The $\overline{\text{RD}}$ signal is made low by the processor. This signal makes the memory device load the data bus with the contents of the location addressed by the processor.

T3 clock cycle

i. The opcode available on the data bus is read by the processor and moved to the instruction register.
ii. The $\overline{\text{RD}}$ signal is deactivated by making it logic 1.

T4 clock cycle

i. The processor decode the instruction in the instruction register and generate the necessary control signals to execute the instruction. Based on the instruction further operations such as fetching, writing into memory etc takes place.

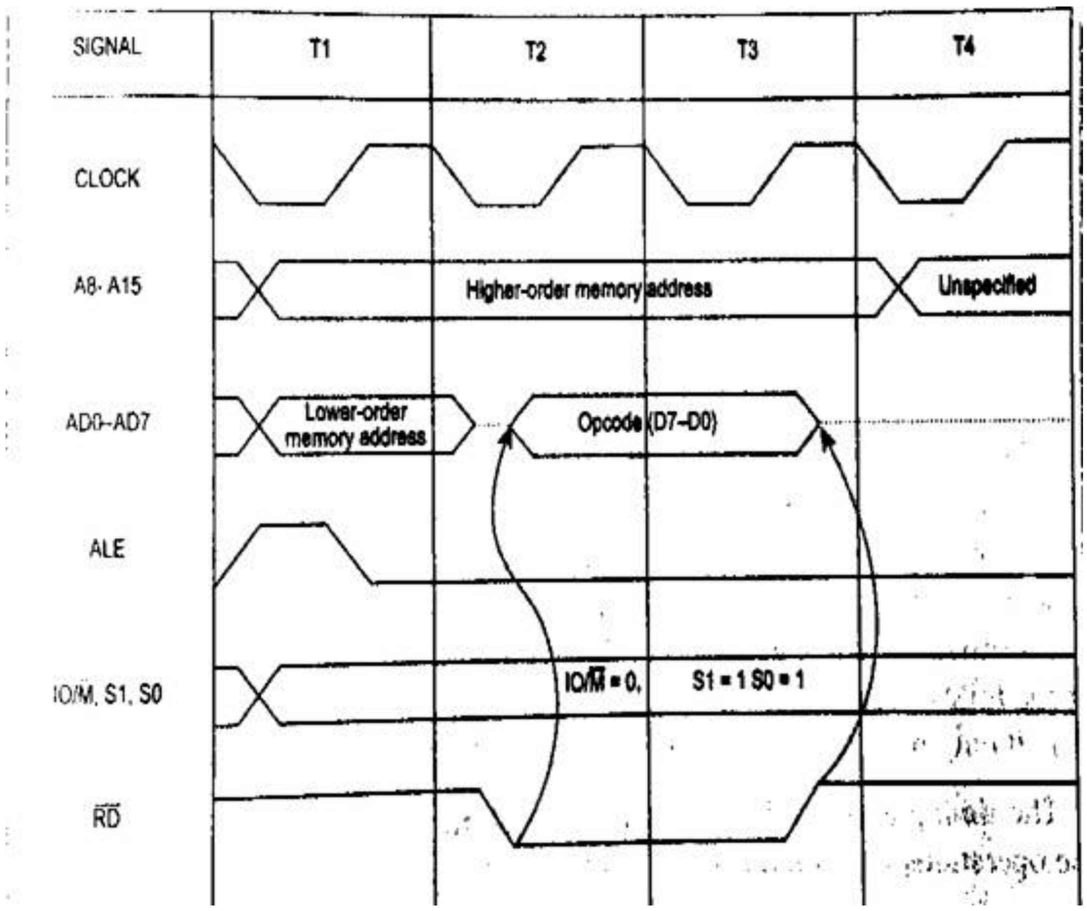


Fig. 7 Timing diagram for opcode fetch cycle

Memory Read Machine Cycle:

The memory read cycle is executed by the processor to read a data byte from memory. The machine cycle is exactly same to opcode fetch except: a) It has three T-states b) The S0 signal is set to 0. The timing diagram of this cycle is given in fig 8

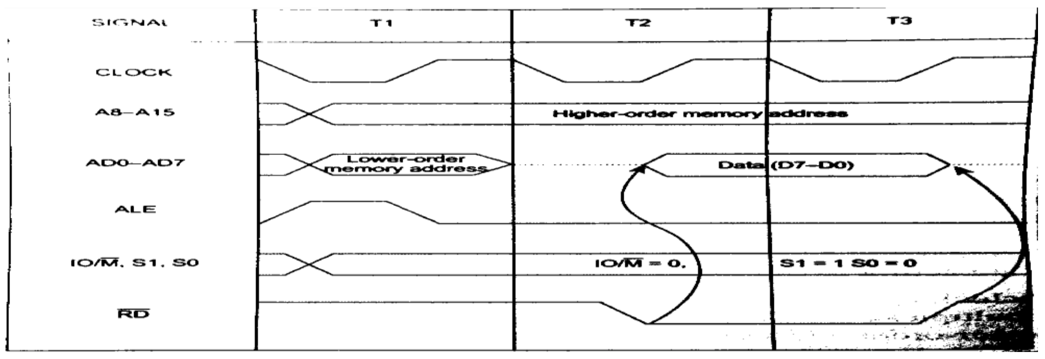


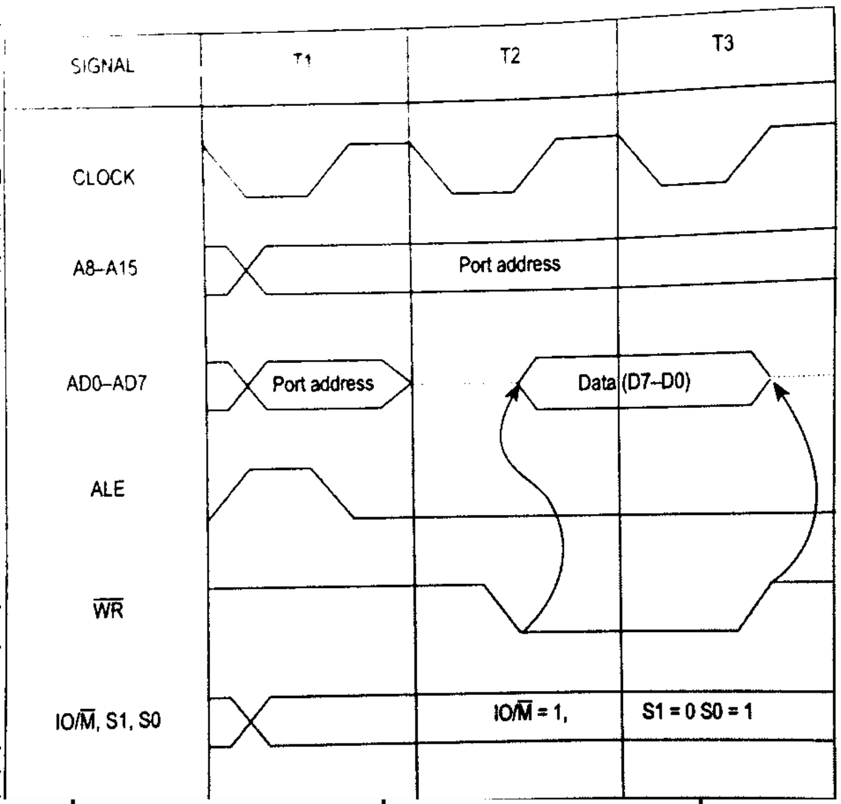
Fig. 8 Timing diagram for memory read machine cycle

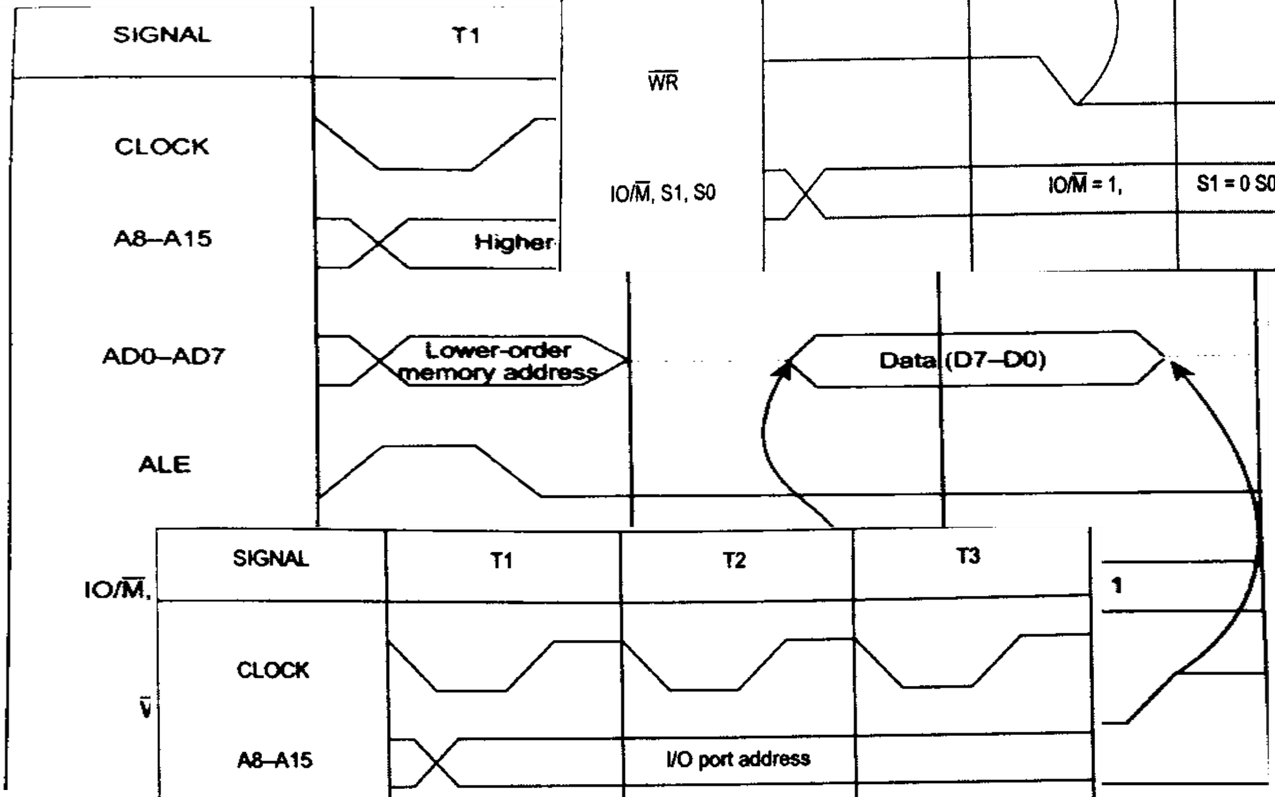
Memory Write Machine Cycle:

The memory write cycle is executed by the processor to write a data byte in a memory location. The processor takes three T-states and $\overline{WR}$ signal is made low. The timing diagram of this cycle is giv

I/O Read Cycle:

The I/O read cycle is executed by
I/O port or from peripheral, which is I/O
address is placed both in the lower and h
takes three T-states to execute this mach
cycle is given in Fig. 10.





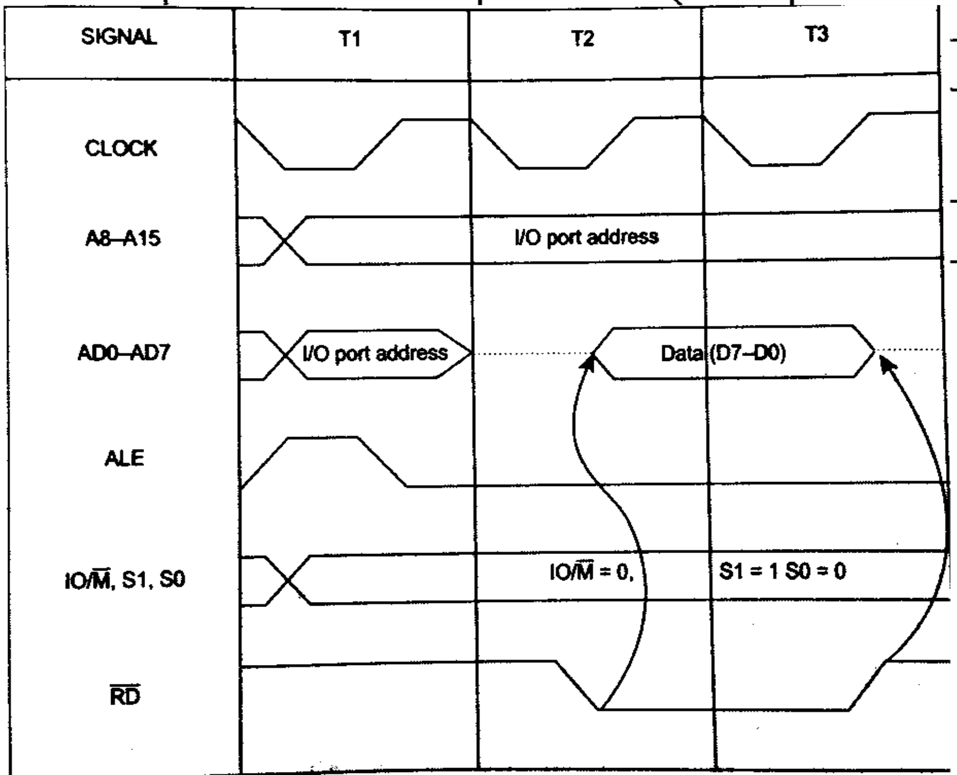


Fig. 10 Timing diagram I/O read machine cycle

I/O Write Cycle:

The I/O write cycle is executed by the processor to write a data byte to I/O port or to a peripheral, which is I/O mapped in the system. The processor takes three T-states to execute this machine cycle. The timing diagram of this cycle is given in Fig. 11.

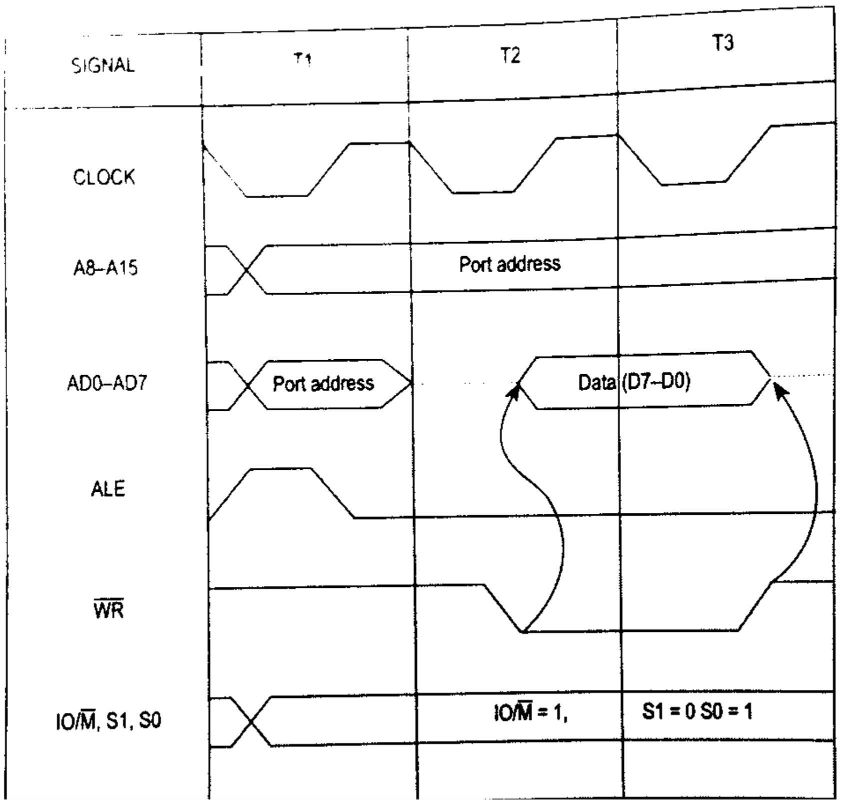


Fig. 11 Timing diagram I/O write machine cycle

Ex: Timing diagram for IN 80H.

The instruction and the corresponding codes and memory locations are given in Table 5.

Table 5 IN instruction

Address	Mnemonics	Opcode
800F	IN 80H	DB
8010		80

i. During the first machine cycle, the opcode DB is fetched from the memory, placed in the instruction register and decoded.
ii. During second machine cycle, the port address 80H is read from the next memory location.
iii. During the third machine cycle, the address 80H is placed in the address

bus and the data read from that port address is placed in the accumulator.

The timing diagram is shown in Fig. 12.

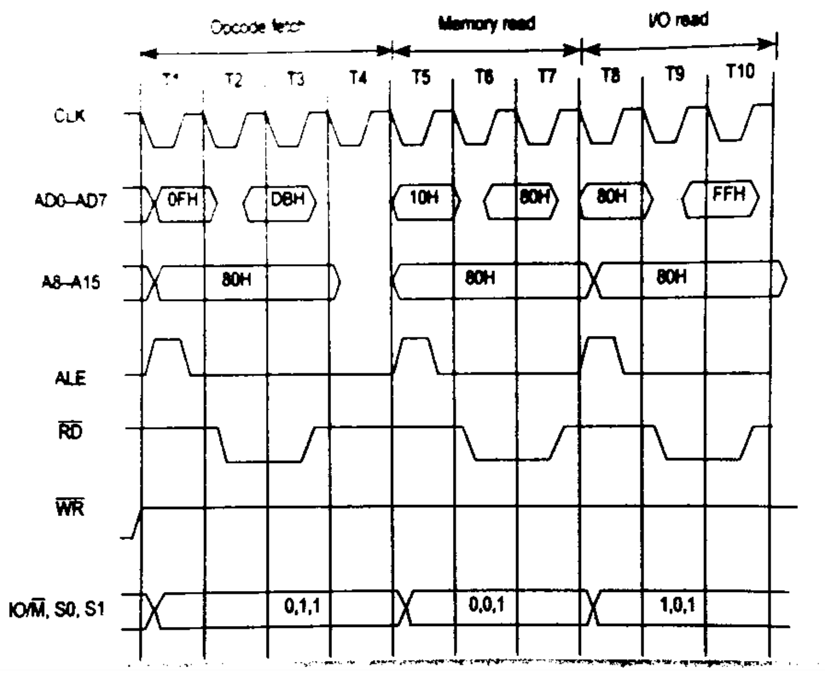


Fig. 12 Timing diagram for the IN instruction

INTERFACING MEMORY CHIPS WITH 8085

8085 has 16 address lines (A0 - A15), hence a maximum of 64 KB (= 2^{16} bytes) of memory locations can be interfaced with it. The memory address space of the 8085 takes values from 0000H to FFFFH.

The 8085 initiates set of signals such as $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ when it wants to read from and write into memory. Similarly, each memory chip has signals such as CE or CS(chip enable or chip select), $\overline{OE}$ or $\overline{RD}$ (output enable or read) and $\overline{WE}$ or $\overline{WR}$ (write enable or write) associated with it.

Generation of Control Signals for Memory:

When the 8085 wants to read from and write into memory, it activates $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals as shown in Table 8.

Table 8 Status of $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals during memory read and write operations

$IO/\overline{M}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	1	8085 reads data from memory
0	1	0	8085 writes data into memory

Using $IO/\overline{M}$, RD and $\overline{WR}$ signals, two control signals MEMR (memory read) and MEMW (memory write) are generated. Fig. shows the circuit used to generate these signals.

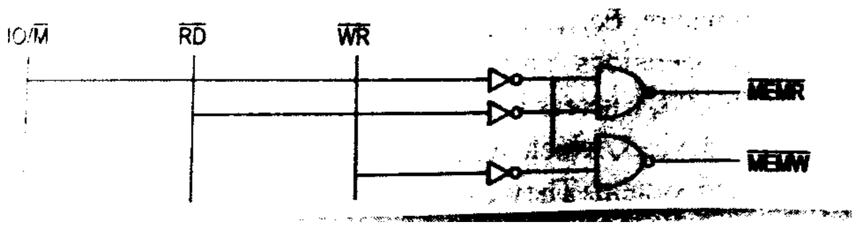


Fig. Circuit used to generate $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ signals

When is $IO/\overline{M}$ high, both memory control signals are deactivated irrespective of the status of $\overline{RD}$ and $\overline{WR}$ signals.

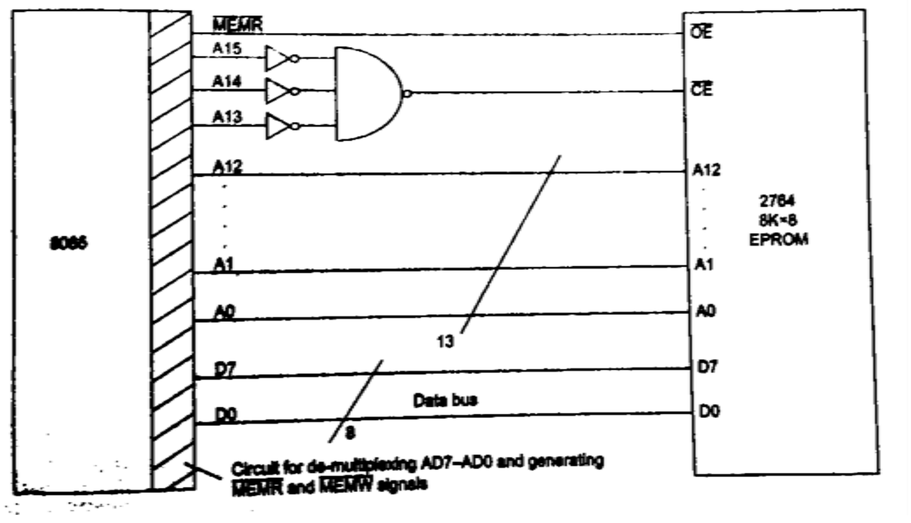
Ex: Interface an IC 2764 with 8085 using NAND gate address decoder such that the address range allocated to the chip is 0000H – 1FFFH.

Specification of IC 2764:

- 8 KB (8 x 2^{10} byte) EPROM chip
- 13 address lines (2^{13}

bytes = 8 KB) Interfacing:

- 13 address lines of IC are connected to the corresponding address lines of 8085.
- Remaining address lines of 8085 are connected to address decoder formed using logic gates, the output of which is connected to the $\overline{CE}$ pin of IC.
- Address range allocated to the chip is shown in Table 9.
- Chip is enabled whenever the 8085 places an address allocated to EPROM chip in the address bus. This is shown in Fig. 17.



MEMR
A15
A14
A13
A12
A1
A0
D7
D0
8085
13
8
Data bus
OE
CE
2764
8K×8
EPROM
Circuit for de-multiplexing AD7–AD0 and generating MEMR and MEMW signals

Fig. 17 Interfacing IC 2764 with the 8085

Table 9 Address allocated to IC 2764

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1FFEH
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

Module II

Instruction set and programming with 8085

Based on the design of the ALU and decoding unit, the microprocessor manufacturer provides instruction set for every microprocessor. The instruction set consists of both machine code and mnemonics.

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called instruction set. Microprocessor instructions can be classified based on the parameters such functionality, length and operand 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.

Classification based on length:

I. One-byte instructions: Instruction having one byte in machine code. Examples are depicted in Table 2.
I. Two-byte instructions: Instruction having two byte in machine code. Examples are depicted in Table 3
II. Three-byte instructions: Instruction having three byte in machine code. Examples are depicted in Table 4.

Table 2 Examples of one byte instructions

Opcode	Operand	Machine code/Hex code
MOV	A, B	78
ADD	M	86

Table 3 Examples of two byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
MVI	A, 7FH	3E	First byte
		7F	Second byte
ADI	0FH	C6	First byte
		0F	Second byte

Table 4 Examples of three byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
JMP	9050H	C3	First byte
		50	Second byte
		90	Third byte
LDA	8850H	3A	First byte
		50	Second byte
		88	Third byte

Data transfer instructions:

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:

1. MOV r1, r2 (Move Data; Move the content of the one register to another). [r1] <-- [r2]

2. MOV r, m (Move the content of memory register). r <-- [M]

3. MOV M, r. (Move the content of register to memory). M <-- [r]

4. MVI r, data. (Move immediate data to register). [r] <-- data.

5. MVI M, data. (Move immediate data to memory). M <-- data.

6. LXI rp, data 16. (Load register pair immediate). [rp] <-- data 16 bits, [rh] <-- 8 LSBs of data.

7. LDA addr. (Load Accumulator direct). [A] <-- [addr].

8. STA addr. (Store accumulator direct). [addr] <-- [A].

9. LHLD addr. (Load H-L pair direct). [L] <-- [addr], [H] <-- [addr+1].

10. SHLD addr. (Store H-L pair direct) [addr] <-- [L], [addr+1] <-- [H].

11. LDAX rp. (LOAD accumulator indirect) [A] <-- [[rp]]

12. STAX rp. (Store accumulator indirect) [[rp]] <-- [A].

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

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:

1. ADD r. (Add register to accumulator) [A] <-- [A] + [r].

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

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

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

5 .ADI data (Add immediate data to accumulator) [A] <-- [A] + data.

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

7.DAD rp. (Add register pair to H-L pair). [H-L] <-- [H-L] + [rp].

8.SUB r. (Subtract register from accumulator). [A] <-- [A] – [r].

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

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

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

12.SUI data. (Subtract immediate data from accumulator) [A] <-- [A] – data.

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

14. INR r (Increment register content) [r] <-- [r] +1.

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

16. DCR r. (Decrement register content). [r] <-- [r] – 1.

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

18.INX rp. (Increment register pair) [rp] <-- [rp] – 1.

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

20.DAA (Decimal adjust accumulator).

3)Logical instructions:

The Instructions under this group perform logical operation such as AND, OR, compare, rotate etc.

Examples:

1. ANA r. (AND register with accumulator) [A] <-- [A] ^ [r].

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

3.ANI data. (AND immediate data with accumulator) [A] <-- [A] ^ data.

4).ORA r. (OR register with accumulator) [A] <-- [A] ∨[r].

5.ORA M. (OR memory with accumulator) [A] <-- [A] ∨ [[H-L]]

6.ORI data. (OR immediate data with accumulator) [A] <-- [A] ∨ data.

7.XRA r. (EXCLUSIVE – OR register with accumulator) [A] <-- [A] xor [r]

8.XRA M. (EXCLUSIVE-OR memory with accumulator) [A] <-- [A] xor [[H-L]]

9.XRI data. (EXCLUSIVE-OR immediate data with accumulator) [A] <-- [A] xor data.

10.CMA. (Complement the accumulator) [A] <-- [A]'

11.CMC. (Complement the carry status) [CS] <-- [CS]'

12.STC. (Set carry status) [CS] <-- 1.

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

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

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

16.RAL (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.

17.RAR. (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.

18. RLC. (Rotate accumulator left through carry) [An+1] <-- [An], [CS] <-- [A7], [A0] <-- [CS].

19. RRC. (Rotate accumulator right through carry) [An] <-- [An+1], [CS] <-- [A0], [A7] <-- [CS].

4)Branching Instructions:

This group includes the instructions for conditional and unconditional jump, subroutine call and return, and restart.

Examples:

Unconditional jump:

MP addr (label). (Unconditional jump: jump to the instruction specified by the address). [PC] <-- Label.

Conditional Jump:

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 true. The program proceeds further in the normal sequence if the specified condition is not true.

1.JZ addr (label). (Jump if ZF=1)
2.JNZ addr (label) (Jump if ZF=0)
3.JC addr (label). (Jump if CF=1)
4.JNC addr (label). (Jump if CF=0)
5.JP addr (label). (Jump if the result is plus)
6.JM addr (label). (Jump if the result is minus)
7.JPE addr (label) (Jump if even parity)
8.JPO addr (label) (Jump if odd parity)
9.CALL addr (label) (Unconditional CALL: call the subroutine identified by the operand)
10.CALL instruction is used to call a subroutine.
11. RET (Return from subroutine).
12. 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.

5)Stack, I/O and Machine control instructions:

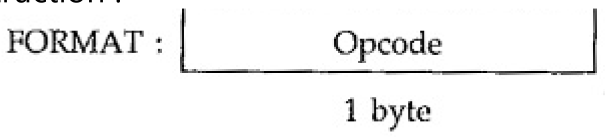
1.IN port-address. (Input to accumulator from I/O port) [A] <-- [Port]
2.OUT port-address (Output from accumulator to I/O port) [Port] <-- [A]
3.PUSH rp (Push the content of register pair to stack)
4.PUSH PSW (PUSH Processor Status Word)
5.POP rp (Pop the content of register pair, which was saved, from the stack)
6.POP PSW (Pop Processor Status Word)
7.HLT (Halt)
8.XTHL (Exchange stack-top with H-L)
9.SPHL (Move the contents of H-L pair to stack pointer)
10.EI (Enable Interrupts)
11. DI (Disable Interrupts)

12. SIM (Set Interrupt Masks)
13. RIM (Read Interrupt Masks)
14. NOP (No Operation).

Instruction Formats:

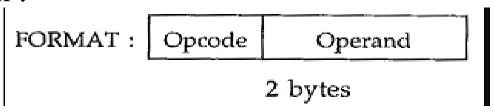
The Instruction Format of 8085 set consists of one, two and three byte instructions. The first byte is always the opcode; in two-byte instructions the second byte is usually data; in three byte instructions the last two bytes present address or 16-bit data.

1. One byte instruction :



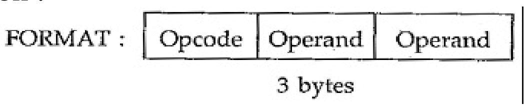
For Example : MOV A, B whose opcode is 78H which is one byte. This Instruction and Data Format of 8085 copies the contents of B register in A register.

2. Two byte instruction :



For Example : MVI B, 02H. The opcode for this instruction is 06H and is always followed by a byte data (02H in this case). This instruction is a two byte instruction which copies immediate data into B register.

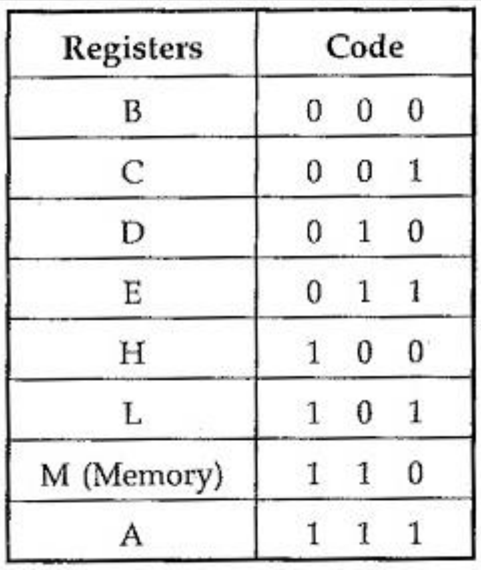
3.Three byte instruction :



For Example : JMP 6200H. The opcode for this instruction is C3H and is always followed by 16 bit address (6200H in this case). This instruction is a three byte instruction which loads 16 bit address into program counter.

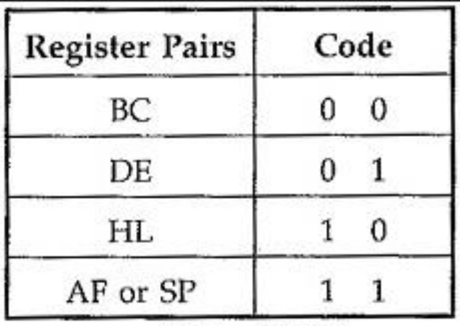
Opcode Format of 8085:

The 8085A microprocessor has 8-bit opcodes. The opcode is unique for each Instruction and Data Format of 8085 and contains the information about operation, register to be used, memory to be used etc. The 8085A identifies all operations, registers and flags with a specific code. For example, all internal registers are identified as shown in the Tables 2.1(a) and 2.2(b).



Registers	Code
B	0 0 0
C	0 0 1
D	0 1 0
E	0 1 1
H	1 0 0
L	1 0 1
M (Memory)	1 1 0
A	1 1 1

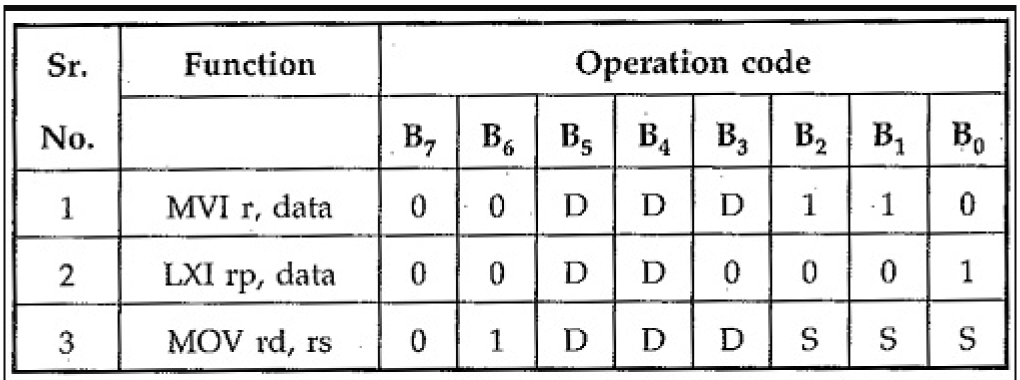
Table 2.1(a)



Register Pairs	Code
BC	0 0
DE	0 1
HL	1 0
AF or SP	1 1

Table 2.1 (b)

Similarly, there are different codes for each opera are identified as follows :



Sr.	Function	Operation code							
No.		B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
1	MVI r, data	0	0	D	D	D	1	1	0
2	LXI rp, data	0	0	D	D	0	0	0	1
3	MOV rd, rs	0	1	D	D	D	S	S	S

Table 2.2

Note : DDD defines the destination register, SSS defines the source register and DD defines the register pair.

Data Format of 8085 Microprocessor:

The operand is an another name for data. It may appear in different forms :

- **Addresses**
- **Numbers/Logical data and**
- **Characters**

Addresses : The address is a 16-bit unsigned integer ,number used to refer a memory location.

Numbers/Data : The 8085 supports following numeric data types.

- **Signed Integer :** A signed integer number is either a positive number or a negative number. In 8085, 8-bits are assigned for signed integer, in which most significant bit is used for sign and remaining seven bits are used for Sign bit 0 indicates positive number whereas sign bit 1 indicates negative number.
- **Unsigned Integer :** The 8085 microprocessor supports 8-bit unsigned integer.
- **BCD :** The term BCD number stands for binary coded decimal number. It uses ten digits from 0 through 9. The 8-bit register of 8085 can store two digit BCD

Characters : The 8085 uses ASCII code to represent characters. It is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and other special characters.

Stack and subroutine

The stack is a LIFO (last in, first out) data structure implemented in the RAM area and is used to store addresses and data when the microprocessor branches to a subroutine. Then the return address used to get pushed on this stack. Also to swap values of two registers and register pairs we use the stack as well.

In the programmer‘s view of 8085, only the general purpose registers A, B, C, D, E, H, and L, and the Flags registers were discussed so far. But in the complete programmer’s view of 8085, there are two more special purpose registers, each of 16-bit width. They are the stack pointer, SP, and the program counter, PC. The Stack Pointer register will hold the address of the top location of the stack. And the program counter is a register always it will hold the address of the memory location from where the next instruction for execution will have to be fetched. The complete programmer's view of 8085 is shown in the following figure.

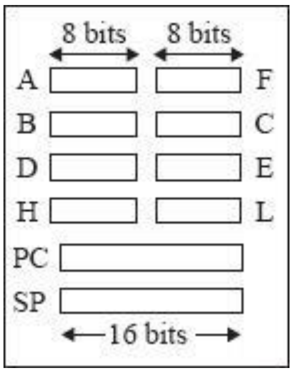


Fig. Programmer's view of 8085

SP is a special purpose 16-bit register. It contains a memory address. Suppose SP contents are FC78H, then the 8085 interprets it as follows.

Memory locations FC78H, FC79H, ..., FFFFH are having useful information. In other words, these locations are treated as filled locations. Memory locations FC77H, FC76H, ..., 0000H are not having any useful information. In other words, these locations are treated as empty locations.

On a stack, we can perform two operations. PUSH and POP. In case of PUSH operation, the SP register gets decreased by 2 and new data item used to insert on to the top of the stack. On the other hand, in case of POP operation, the data item will have to be deleted from the top of the stack and the SP register will get increased by the value of 2.

Thus, the contents of SP specify the top most useful location in the stack. In other words, it indicates the memory location with the smallest address having useful information. This is pictorially represented in the following figure –

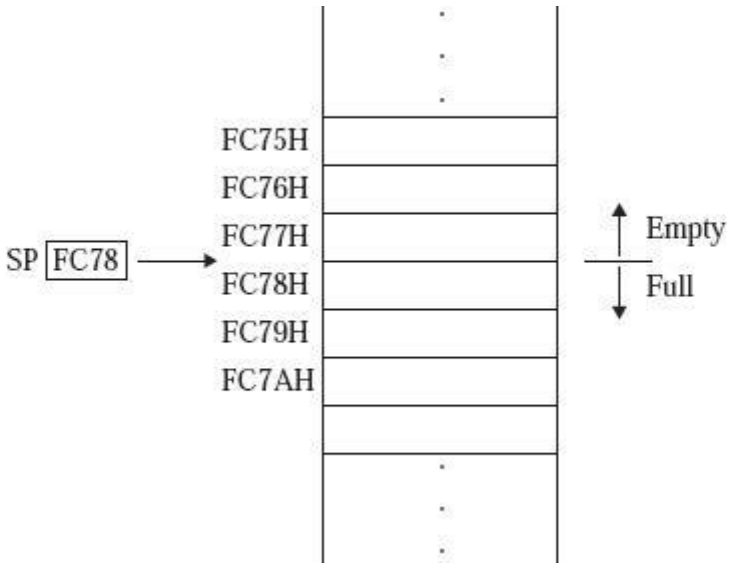


Fig. Interpretation of SP contents

Subroutine:

In computers, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task have to be performed. A subroutine is often coded so that it can be started (called) several times and from several places during one execution of the program, including from other subroutines, and then branch back (return) to the next instruction after the call, once the subroutine’s task is done. It is implemented by using Call and Return instructions. The different types of subroutine instructions are

Unconditional Call instruction –
CALL address is the format for unconditional call instruction. After execution of this instruction program control is transferred to a sub-routine whose starting address is specified in the instruction. Value of PC (Program Counter) is transferred to the memory stack and value of SP (Stack Pointer) is decremented by 2.

Conditional Call instruction –
In these instructions program control is transferred to subroutine and value of PC is pushed into stack only if condition is satisfied.

INSTRUCTION	PARAMETER	COMMENT
CC	16-bit address	Call at address if cy (carry flag) = 1
CNC	16-bit address	Call at address if cy (carry flag) = 0
CZ	16-bit address	Call at address if ZF (zero flag) = 1
CNZ	16-bit address	Call at address if ZF (zero flag) = 0
CPE	16-bit address	Call at address if PF (parity flag) = 1
CPO	16-bit address	Call at address if PF (parity flag) = 0
CN	16-bit address	Call at address if SF (signed flag) = 1
CP	16-bit address	Call at address if SF (signed flag) = 0

Unconditional Return instruction –RET is the instruction used to mark the end of sub-routine. It has no parameter. After execution of this instruction program control is transferred back to main program from where it had stopped. Value of PC (Program Counter) is retrieved from the memory stack and value of SP (Stack Pointer) is incremented by 2.
Conditional Return instruction –
By these instructions program control is transferred back to main program and value of PC is popped from stack only if condition is satisfied. There is no parameter for return instruction.

INSTRUCTION	COMMENT
RC	Return from subroutine if cy (carry flag) = 1
RNC	Return from subroutine if cy (carry flag) = 0
RZ	Return from subroutine if ZF (zero flag) = 1
RNZ	Return from subroutine if ZF (zero flag) = 0
RPE	Return from subroutine if PF (parity flag) = 1
RPO	Return from subroutine if PF (parity flag) = 0
RN	Return from subroutine if SF (signed flag) = 1
RP	Return from subroutine if SF (signed flag) = 0

Advantages of Subroutine –
1. Decomposing a complex programming task into simpler steps.
2. Reducing duplicate code within a program.

3. Enabling reuse of code across multiple programs.
4. Improving tractability or makes debugging of a program easy.

Assembler and Assembler directives

Definition: Assembler directives are the **instructions** used by the assembler at the time of assembling a source program. More specifically, we can say, assembler directives are the commands or instructions that control the operation of the assembler.
Assembler directives are the instructions provided to the assembler, not the processor as the processor has nothing to do with these instructions. These instructions are also known as **pseudo-instructions** or **pseudo-opcode**.
So, assembler directives:

- show the beginning and end of a program provided to the assembler,
- used to provide storage locations to data,
- used to give values to variables,
- define the start and end of different segments, procedures or macros etc. of a program.

Assembler:

We know that assembly language is a less complex and programmer-friendly language used to program the processors.

In assembly language programming, the instructions are specified in the form of mnemonics rather in the form of machine code i.e., 0 and 1.
But the microprocessor or microcontrollers are specifically designed in a way that they can only understand machine language.

Thus assembler is used to convert assembly language into machine code so that it can be understood and executed by the processor.

Therefore, to control the generation of machine codes from the assembly language, assembler directives are used.
However, machine codes are only generated for the program that must be provided to the processor and not for assembler directives because they do not belong to the actual program.

Assembler Directives of 8085

The assembler directives given below are used by 8085 and 8086 assemblers:

DB: *Define Byte*
This directive is used for the purpose of allocating and initializing single or multiple data bytes.
Ex:Num1 DB 30H,25H,60H
i.e.,Num1 memory has 32 consecutive where 30H,25H and 60H are stored.
DW: *Define Word*
It is used for initialising single or multiple data words (16-bit)

Ex:Num2 DW 1020H,4216H

These two 16-bit data 1020H and 4216H are stored at 4 consecutive locations in the memory Num2.

END: *End of program*
This directive is used at the time of program termination.

EQU: *Equate*

It is used to assign any numerical value or constant to the variable.

MACRO: *Represents beginning*
Shows the beginning of macro along with defining name and parameters.

ENDM: *End of macro*
ENDM indicates the termination of macro.

ORG: *Origin*
This directive is used at the time of assigning starting address for a module or segment.

Programming Techniques in Microprocessor 8085:
We have seen the instruction set of 8085 and some simple assembly language programs using it. We know that, the program is an implementation of certain logic by executing group of instructions. To implement program logic we need to take help of some common Programming Techniques in Microprocessor 8085 such as looping, counting, indexing and code conversion.

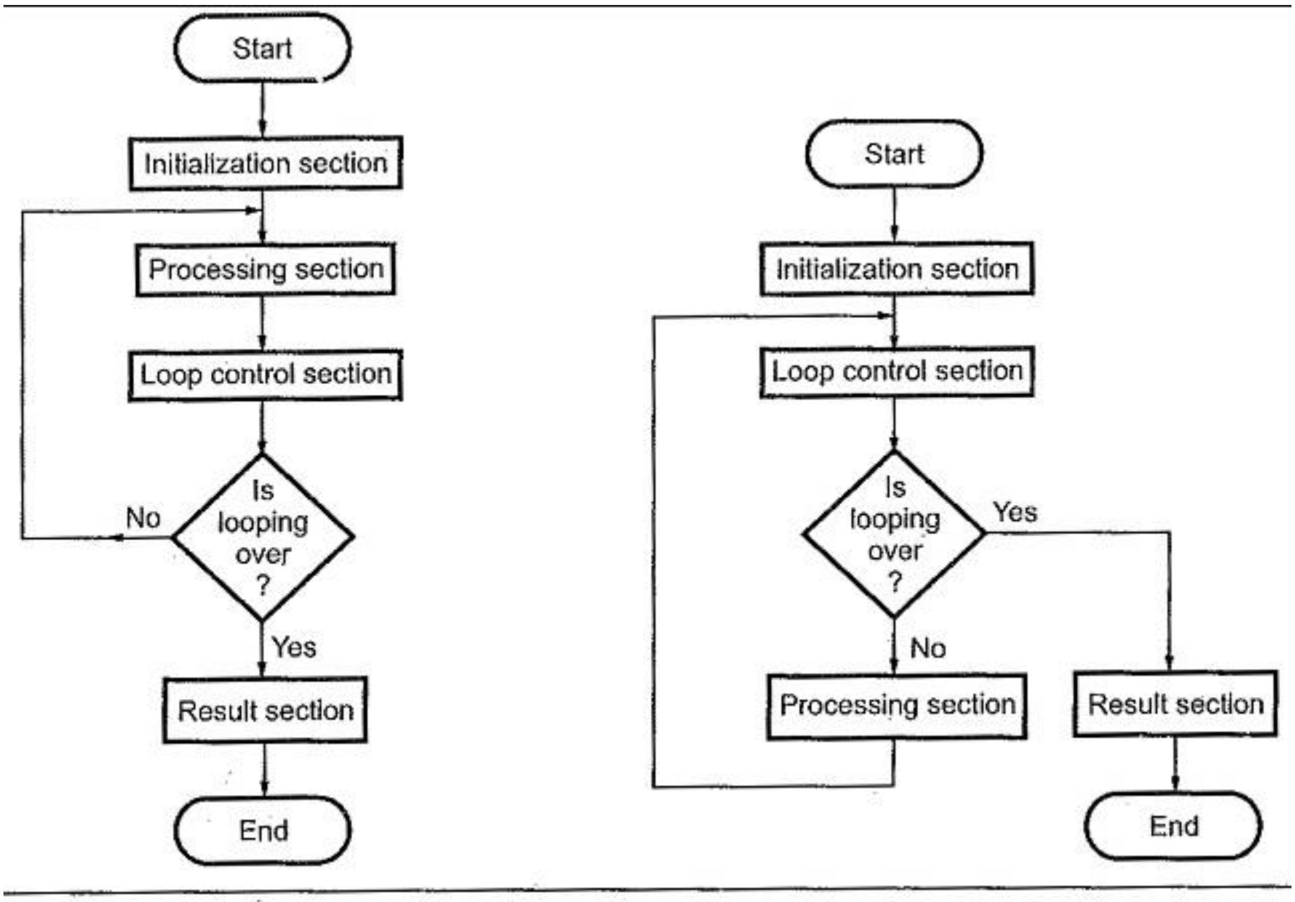
In this section, we are going to study how to implement these Programming Techniques in Microprocessor 8085 assembly language and some programming examples using them.

1. Looping, Counting and Indexing:

Before going to implement these Programming Techniques in Microprocessor 8085, we get conversant with these techniques and understand the use of them.

Looping : In this Programming Techniques in Microprocessor 8085, the Program is instructed to execute certain set of instructions repeatedly to execute a particular task number of times. For example, to add ten numbers stored in the consecutive memory locations we have to perform addition ten times.
Counting : This technique allows programmer to count how many times the instruction/set of instructions are executed.
Indexing : This Programming Techniques in Microprocessor 8085 allows programmer to point or refer the data stored in sequential memory locations one by one. Let us see the program loop to understand looping, counting and indexing.
The Program loop is the basic structure which forces the processor to repeat a sequence of instructions. Loops have four sections.

- **Initialization section.**
- **Processing section.**
- **Loop control section**
- **Result section.**



Flowchart 1 **Flowchart 2**

1. The initialization section establishes the starting values of
   - loop counters for counting how many times loop is executed,
   - address registers for indexing which give pointers to memory locations and
   - other variables
2. The actual data manipulation occurs in the processing section. This is the section which does the work.
3. The loop control section updates counters, indices (pointers) for the next iteration.
4. The result section analyzes and stores the results.

Note : The processor executes initialization section and result section only once, while it may execute processing section and loop control section many times. Thus, the execution time of the loop will be mainly dependent on the execution time of the processing section and loop control section. The flowchart 1 shows typical Program loop. The processing section in this flowchart is always executed at least once. If you interchange the position of the processing and loop control section then it is possible that the processing section may not be executed at all, if necessary. Refer flowchart 2.

2. Timers:

In the real time applications, such as traffic light control, digital clock, process control, serial communication, it is important to keep a track with time. For example in traffic light control application, it is necessary to give time delays between two transitions. These time delays are in few seconds and can be generated with the help of executing group of instructions number of times. This software timers are also called time delays or software delays. Let us see how to implement these time delays or software delays.

As you know microprocessor system consists of two basic components, Hardware and software. The software component controls and operates the hardware to get the desired output with the help of instructions. To execute these instructions, microprocessor takes fix time as per the instruction, since it is driven by constant frequency clock. This makes it possible to introduce delay for specific time between two events. In the following section we will see different delay implementation techniques.

2.1 Timer Delay Using NOP Instruction:

NOP instruction does nothing but takes 4T states of processor time to execute. So by executing NOP instruction in between two instructions we can get delay of 4 T-state

$$1\ \text{T state} = \frac{1}{\text{operating frequncy of 8085}}$$

2.2 Timer Delay Using Counters:
Counting can create time delays. Since the execution times of the instructions used in a counting routine are known, the initial value of the counter, required to get specific time delay can be determined.

Using 8 bit counter :

```
                                                 Number of T-states
         MVI C, count    ; Load count            7 T-states
BACK :   DCR C           ; Decrement count       4 T-states
         JNZ BACK        ; If count ≠ 0, repeat  10/7 T-states
```

Using 16 Bit Counter :

```
                                                       Number of T-states
         LXI B, count   ; load 16 bit count            10 T-states
BACK :   DCX B          ; Decrement count              6 T-states
         MOV A, C       ;                              4 T-states
         ORA B          ; logically OR B and C         4 T-states
         JNZ BACK       ; If result is not 0, repeat   10 T-states
```

2.3 Timer Delay Using Nested Loops:
In this, there are more than one loops. The innermost loop is same as explained above. The outer loop sets the multiplying count to the delays provided by the innermost loop.

```
                                                                Number of T states

        MVI B, Multiplier count   ; Initialize multiplier         7 T-states
START:  MVI C, Delay count        ; initialize delay count        7 T-states
BACK:   DCR C                     ; Decrement delay count         4 T-states
        JNZ BACK                  ; if not 0, repeat              10/7 T-states
        DCR B                     ; decrement multiplier count    4 T-states
        JNZ START                 ; If not 0, repeat              10/7 T-states
```

3. Code Conversion:
This Programming Techniques in Microprocessor 8085 is to translate a number represented using one coding system to another. For example,, when we accept any number from the keyboard it is in ASCII code. But for processing, we have to convert this number in its hex equivalent. The code conversion involves some basic conversions such as

- **BCD to Binary conversion**
- **Binary to BCD conversion**
- **BCD to seven segment code conversion**
- **Binary to ASCII conversion and**
- **ASCII to binary conversion**

BCD to Binary Conversion:

We are more familar with the decimal number system. But the microprocessor understands the binary/hex number system. To convert BCD number into its binary equivalent we have to use the principle of positional weighting in a given number.

Binary to BCD Conversion:

We know that microprocessor processes data in the binary form. But when it is displayed, it is in the BCD form. In this case we need binary to BCD conversion of data. The conversion of binary to BCD is performed by dividing the number by the power of ten.

BCD to Seven Segment Conversion:

Many times 7-segment LED display is used to display the results or parameters in the microprocessor system. In such cases we have to convert the result or parameter in 7-segment code. This conversion can be done using look-up technique. In the look-up table the codes of the digits (0-9) to be displayed are stored sequentially in the memory. The conversion program locates the code of a digit based on its BCD digit. Let us see the Program for BCD to comsmon cathode 7-segment code conversion.

Binary to ASCII Code Conversion:

The ASCII Code (American Standard Code for Information Interchange) is commonly used for communication. In such cases we need to convert binary number to its ASCII equivalent. It is a seven bit code. In this code number 0 through 9 are represented as 30 through 39 respectively and letters A through Z are represented as 41H through 5AH. Therefore, by adding 30H we can convert number into its ASCII equivalent and by adding 37H we can convert letter to its ASCII equivalent. Let us see the Program for binary to ASCII code conversion.

ASCII Code to Binary Conversion:

It is exactly reverse process to binary to ASCII conversion. Here, if ASCII code is less than 3AH then 30H is subtracted to get the binary equivalent and if it is in between 41H and 5AH then 37H is subtracted to get the binary equivalent of letter (A-F).

Addressing Modes in Instructions:

The process of specifying the data to be operated on by the instruction is called addressing. The various formats for specifying operands are called addressing modes. The 8085 has the following five types of addressing:

I. Immediate addressing
II. Memory direct addressing
III. Register direct addressing
IV. Indirect addressing
V. Implicit addressing

Immediate Addressing:

In this mode, the operand given in the instruction - a byte or word – transfers to the destination register or memory location.

Ex: MVI A, 9AH

- The operand is a part of the instruction.
- The operand is stored in the register mentioned in the instruction.

Memory Direct Addressing:

Memory direct addressing moves a byte or word between a memory location and register. The memory location address is given in the instruction.

Ex: LDA 850FH

This instruction is used to load the content of memory address 850FH in the accumulator.

Register Direct Addressing:

Register direct addressing transfer a copy of a byte or word from source register to destination register.

Ex: MOV B, C

It copies the content of register C to register B.

Indirect Addressing:

Indirect addressing transfers a byte or word between a register and a memory location.

Ex: MOV A, M

Here the data is in the memory location pointed to by the contents of HL pair. The data is moved to the accumulator.

Implicit Addressing

In this addressing mode the data itself specifies the data to be operated upon.

Ex:CMA

The instruction complements the content of the accumulator. No specific data or operand is mentioned in the instruction.

Examples of Assembly language program using 8085

1.BCD to Binary Conversion

```
MVI A,72
MOV B,A
ANI 0FH
MOV C,A
MOV A,B
ANI 0F0H
JZ BCD1
RRC
RRC
RRC
RRC
MOV D,A
XRA A
MVI E,0AH
SUM:ADD E
DCR D
JNZ SUM
BCD1:ADD C
MOV A,C
STA 2000H
HLT
```

RESULT: 48

Binary to BCD Conversion:

```
MVI A,8AH
MVI B,64H
MVI C,0AH
MVI D,00H
MVI E,00H
STEP1:CMP B
JC STEP2
SUB B
INR E
```

```
JMP STEP1
STEP2:CMP C
JC STEP3
SUB C
INNR D
JMP STEP2
STEP3:STA 2000H
MOV A,D
STA 2001H
MOV A,E
STA 2002H
HLT
```

Input:8AH
Output:2000h-8,2001h-3,2002h-1

Binary to ASCII

```
LDA 2050
CALL 2500
STA 3050
LDA 2050
RLC
RLC
RLC
RLC
CALL 2500
STA 3051
HLT
2500:ANI 0FH
        CPI 0AH
        JNC NEXT
        ADI 30
        RET
NEXT:ADI 37H
        RET
```

Input:2050-4AH
Output:3050-41H
        3051-34H

ASCII to Binary

```
LDA 2000H
SUI 30H
CPI 0AH
JC NEXT
SUI 07H
NEXT:STA 2001
        HLT
```

Bcd To Common Cathode Seven Segment Display

```
LXI H,2050
MOV A,M
MOV D,A
LXI B,3050
```

```
ANI 0F0H
RRC
RRC
RRC
RRC
CALL SUB
INX B
MOV A,D
ANI 0FH
CALL SUB
INX B
INX H
SUB:LXI H,4050
ADD L
MOV L,A
MOV A,M
STAX B
RET
```

```
Input:2050h-34
Output:3050h-4F
       3051h-66
```

Ascending Order

```
LXI H,2300
MVI C,03
DCR C
START:MOV D,C
CHECK:MOV A,B
      INX H
      CMP M
      JC NEXTBYTE
      MOV B,M
      MOV M,A
      DCX H
      MOV M,B
      INX H
NEXTBYTE:DCR D
         JNZ CHECK
         DCR C
         JNZ START
         HLT
```

Descending Order

```
LXI H,2300
MVI C,03
DCR C
START:MOV D,C
CHECK:MOV A,B
      INX H
      CMP M
      JNC NEXTBYTE
      MOV B,M
      MOV M,A
      DCX H
```

```
            MOV M,B
            INX H
NEXTBYTE:DCR D
                        JNZ CHECK
                        DCR C
                        JNZ START
                        HLT
```

Addition:

```
MVI C,00
LDA 4000H
MOV B,A
LDA 4001H
ADD B
JNC LOOP
INR C
LOOP:STA 4002H
            MOV A,C
            STA 4003H
            HLT
```

Subtraction

```
MVI C,00H
LDA 4000H
MOV B,A
LDA 4001H
SUB B
JNC LOOP
CMA
INR A
INR C
LOOP:STA 4002H
            MOV A,C
            STA 4003H
            HLT
```

Multiplication:

```
MVI D,00H
MVI A,00H
LXI H,4150H
MOV B,M
INX H
MOV C,M
LOOP:ADD B
            JNC NEXT
            INR D
NEXT:DCR C
            JNZ LOOP
            STA 4152H
            MOV A,D
            STA 4153H
            HLT
```

Division:

```
LXI H,4150H
MOV B,M
MVI C,00H
INX H
MOV A,M
NEXT:CMP B
        JC LOOP
        SUB B
        INR C
        JMP NEXT
LOOP:STA 4152H
        MOV A,C
        STA 4153H
        HLT
```

Logical Operations:

```
LDA 4000H
MOV B,A
LDA 4001H
ANA B
STA 4002H
LDA 4001H
ORA B
STA 4003H
LDA 4001H
CMA
 STA 4004H
LDA 4001H
XRA B
STA 4005H
HLT
```

****

UNIT-III

8086 Architecture

Introduction to Microprocessors

A microprocessor is a computer processor which incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits

The microprocessor is a multipurpose, clock driven, register based, digital-integrated circuit which accepts binary data as input, processes it according to instructions stored in its memory, and provides results as output. Microprocessors contain both combinational logic and sequential digital logic. Microprocessors operate on numbers and symbols represented in the binary numeral system.

Generation of Microprocessors:

- **INTEL 4004 (1971)**
  - 4-bit microprocessor
  - 4 KB main memory
  - 45 instructions
  - PMOS technology
  - was first programmable device which was used in calculators
- **INTEL 8008 (1972)**
  - 8-bit version of 4004
  - 16 KB main memory
  - 48 instructions
  - PMOS technology
  - Slow
- **Intel 8080** (1973)
  - 8-bit microprocessor
  - 64 KB main memory
  - 2 microseconds clock cycle time
  - 500,000 instructions/sec
  - 10X faster than 8008
  - NMOS technology
  - Drawback was that it needed three power supplies.

- Small computers (Microcomputers) were designed in mid 1970's
  Using 8080 as CPU.

➢ **INTEL 8086/8088**

Year of introduction 1978 for 8086 and 1979 for 8088

- 16-bit microprocessors
- Data bus width of 8086 is 16 bit and 8 bit for 8088
- 1 MB main memory
- 400 nanoseconds clock cycle time
- 6 byte instruction cache for 8086 and 4 byte for 8088
- Other improvements included more registers and additional instructions
- In 1981 IBM decided to use 8088 in its personal computer

➢ **INTEL 80186** (1982)

- 16-bit microprocessor-upgraded version of 8086
- 1 MB main memory
- Contained special hardware like programmable counters, interrupt controller etc.
- Never used in the PC
- But was ideal for systems that required a minimum of hardware .

➢ **INTEL 80286** (1983)

- 16-bit high performance microprocessor with memory management & protection
- 16 MB main memory
- Few additional instructions to handle extra 15 MB
- Instruction execution time is as little as 250 ns
- Concentrates on the features needed to implement MULTITASKING

➢ **Intel 80386** (1986)
➢ **Intel 80486** (1989)
➢ **Pentium** (1993)
➢ **Pentium pro**(1995)
➢ **Pentium ii** (1997)
➢ **Pentium iii** (1999)
➢ **Pentium iv** (2002)
➢ **Latest is Intel i9 processor**

General Architecture of Microprocessors

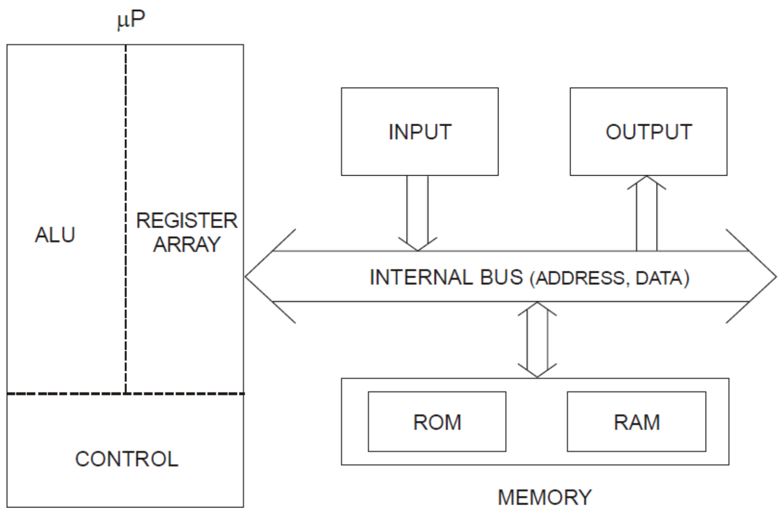


Figure 1.2 Architecture of Microprocessor

Buses

Figure shows busses interconnecting various blocks. These busses allow exchange of words between the blocks. A bus has a wire or line for each bit and thus allows exchange of all bits of a word in parallel. The processing of bits in the µP is also in parallel. The busses can thus be viewed as data highways. The width of a bus is the number of signal lines that constitute the bus.

Arithmetic-Logic Unit (ALU)

The arithmetic-logic unit is a combinational network that performs arithmetic and logical operations on the data.

Internal Registers

A number of registers are normally included in the microprocessor. These are used for temporary storage of data, instructions and addresses during execution of a program. Those in the Intel

Register Organization of 8086

8086 has a powerful set of registers containing general purpose and special purpose registers. All the registers of 8086 are 16-bit registers. The general purpose registers, can be used either 8-bit registers or 16-bit registers. The general purpose registers are either used for holding the data, variables and intermediate results temporarily or for other purpose like counter or for storing offset address for some particular addressing modes etc. The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes. Fig 1.4 shows register organization of 8086. We will categorize the register set into four groups as follows:

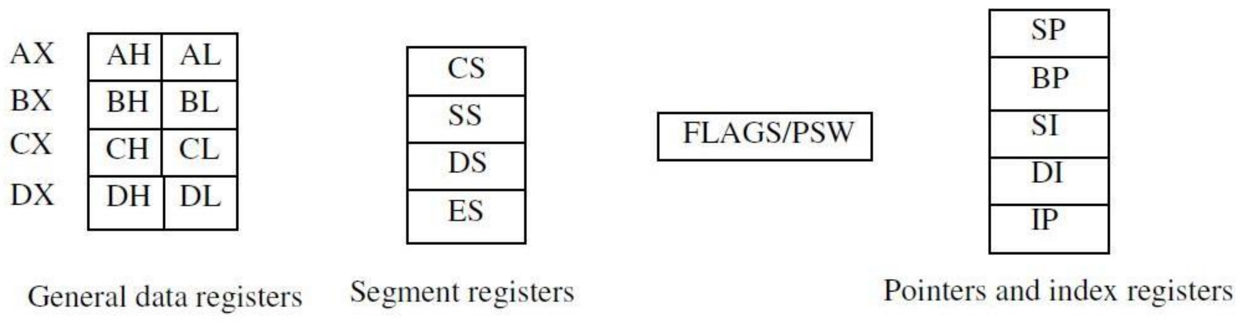


Fig.1.4 Register organization of 8086 Microprocessor

General data Registers:

The registers AX, BX, CX, and DX are the general 16-bit registers.

AX Register: Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high- order byte. Accumulator can be used for I/O operations, rotate and string manipulation.

BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.

CX Register: It is used as default counter or count register in case of string and loop instructions.

DX Register: Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Segment registers:

To complete 1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation is as shown in fig 1.5. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. It also contains data.

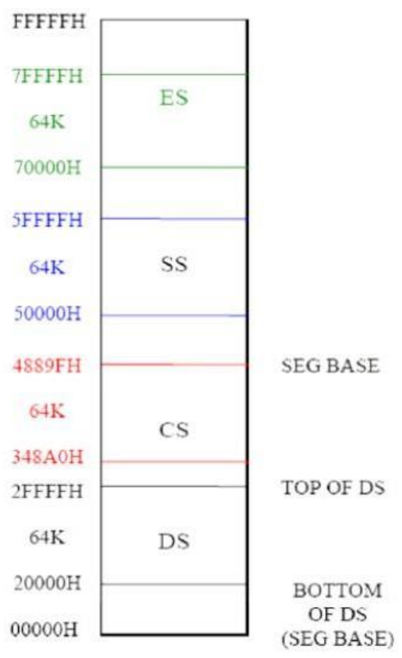


Fig1.5. Memory segmentation

Pointers and index registers.

The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Flag Register:

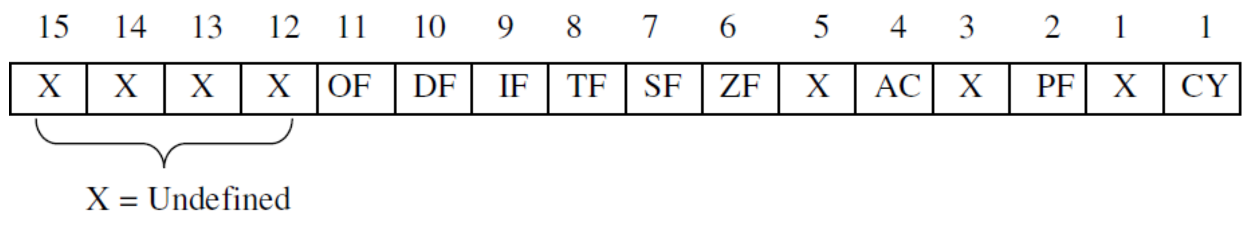


Fig1.6 . Flag Register of 8086

Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. The 8086 flag register as shown in the fig 1.6. 8086 has 9 active flags and they are divided into two categories:

1. Conditional Flags
2. Control Flags

Conditional flags are as follows:

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AC flag is set i.e. carry given by D3 bit to D4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Control Flags

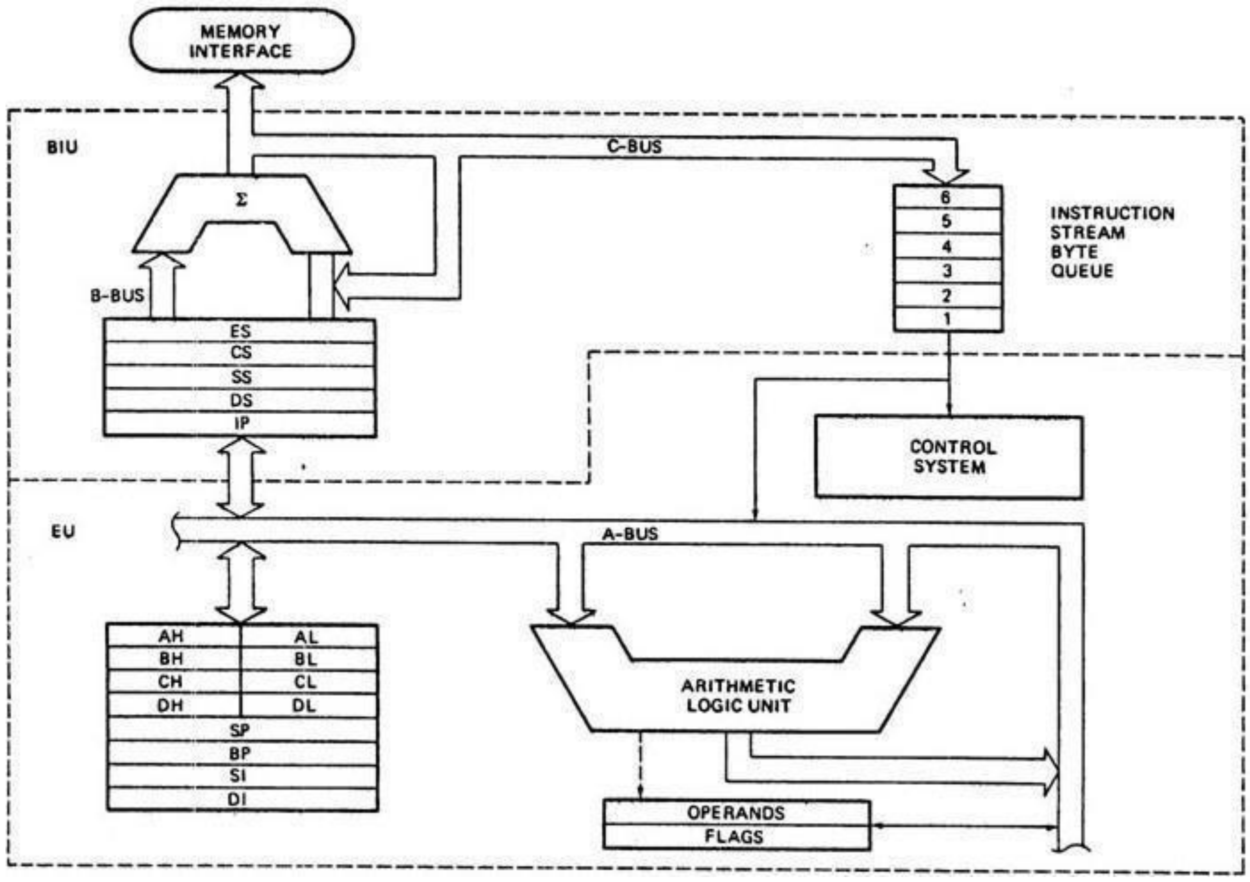
Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

8086 Architecture



The 8086 is mainly divided into mainly two blocks
1. Execution Unit (EU) 2.Bus
interface Unit (BIU)
Dividing the work between these two will speedup the processing

1) EXECUTION UNIT(EU)

The Execution unit tells the BIU where to fetch instructions or data from

- decodes instructions and
- Executes instructions

The Execution unit contains:

1) Control circuitry
2) ALU
3) FLAGS
4) General purpose Registers
5) Pointer and Index Registers

Control Circuitry:

- It directs internal operations.

- A decoder in the EU translates instructions fetched from memory into series of actions which the EU carries out

Arithmetic Logic Unit:
16 bit ALU
Used to carry the operations

- ADD
- SUBTRACT
- XOR
- INCREMENT
- DECREMENT
- COMPLEMENT
- SHIFT BINARY NUMBERS

FLAG REGISTERS:

- A flag is a flip flop that indicates some condition produced by execution of an instruction or controls certain operation of the EU.
- It is 16 bit
- It has nine active flags

Divided into two types

1. Conditional flags
2. Control flags

Conditional Flags

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AC flag is set i.e. carry given by D3 bit to D4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

General Purpose Registers:
The 8086 general purpose registers are similar to those of earlier generations 8080 and 8085 .It was designed in such a way that many programs written for 8080 and 8085 could easily be translated to run on 8086.The advantage of using internal registers for the temporary storage of data is that since data already in the EU ., it can be accessed much more quickly than it could be accessed from external memory.

General Purpose Registers

The registers AX, BX, CX, and DX are the general 16-bit registers.
AX Register: Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high- order byte. Accumulator can be used for I/O operations, rotate and string manipulation.
BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.
CX Register: It is used as default counter or count register in case of string and loop instructions.
DX Register: Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

2) BUS INTERFACE UNIT (BIU)

The BIU sends out

- Addresses
- Fetches instructions from memory
- Read data from ports and memory Or

The BIU handles all transfer of data and addresses on the buses for the Execution Unit

The Bus interface unit contains

1) Instruction Queue

2) Instruction pointer
3) Segment registers
4) Address Generator

Instruction Queue:

BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed. Fetching the next instruction while the current instruction executes is called **pipelining.**(based on FIFO) .This is much faster than sending out an addresses to the system memory and waiting for memory to send back the next instruction byte or bytes .Here the Queue will be dumped and then reloaded from the new Address.

Segment Register:

The 8086 20 bit addresses So it can address upto 2^{20} in memory (1 Mbyte) but at any instant it can address upto 4 64 KB segments. This four segments holds the upper 16 bits of the starting address of four memory segments that the 8086 is working with it at particular time .The BIU always inserts zeros for the lowest 4 bits of the 20 bit starting address

Example : If the code segment register contains 348AH then the code segment starts at 348A0H .In other words a 64Kbyte segment can be located anywhere within 1MByte address Space but the segment will always starts at an address with zeros in the lowest 4 bits

Stack: is a section of memory set aside to store addresses and data while subprogram executes is often called segment base . The stack segment register always holds the upper 16 bit starting address of program stack**.**

The extra segment register and data segment register is used to hold the upper 16 bit starting addresses of two memory segments that are used for data .

Instruction Pointer holds the 16 bit address or offset of the next code byte within the code segment. The value contained in the Instruction Pointer called as Offset because the value must be added to the segment base address in CS to produce the required 20 bit address.

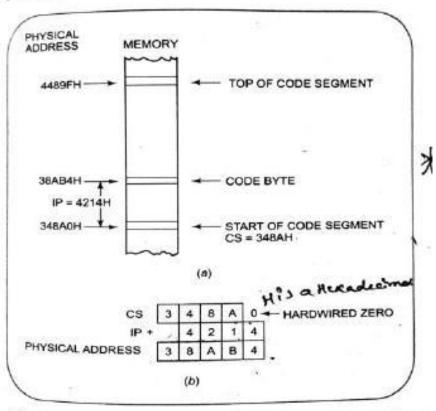


Fig. 2.10 *Addition of IP to CS to produce the physical address of the code byte. (a) Diagram, (b) Computation.*

CS register contains the Upper 16 bit of the starting address of the code segment in the 1 Mbyte address range the instruction pointer contains a 16 bit offset which tells wherein that 64 Kbyte code segment the next instruction byte has to be fetched from.

Stack Register and Stack Pointer:

Stack: is a section of memory set aside to store addresses and data while subprogram executes is often called segment base . The stack segment register always holds the upper 16 bit starting address of program stack. The Stack pointer (SP) holds the 16 bit offset from the starting of the segment to the memory location where a word was most recently stored

.The memory location where the word is stored is called as top of the stack

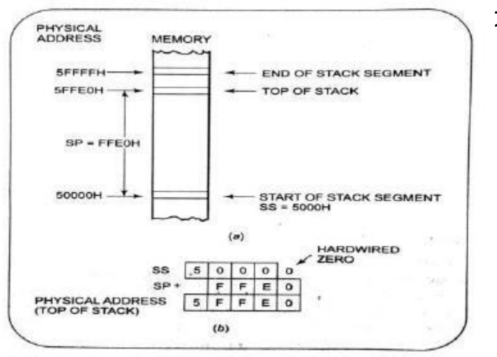


Fig. 2.11 *Addition of SS and SP to produce the physical address of the top of the stack. (a) Diagram, (b) Computation.*

Pointer and Index registers:
In addition to stack pointer register EU has Base
pointer Register (BP)
Source Pointer Register(SP) Destination
Pointer Register(DP)

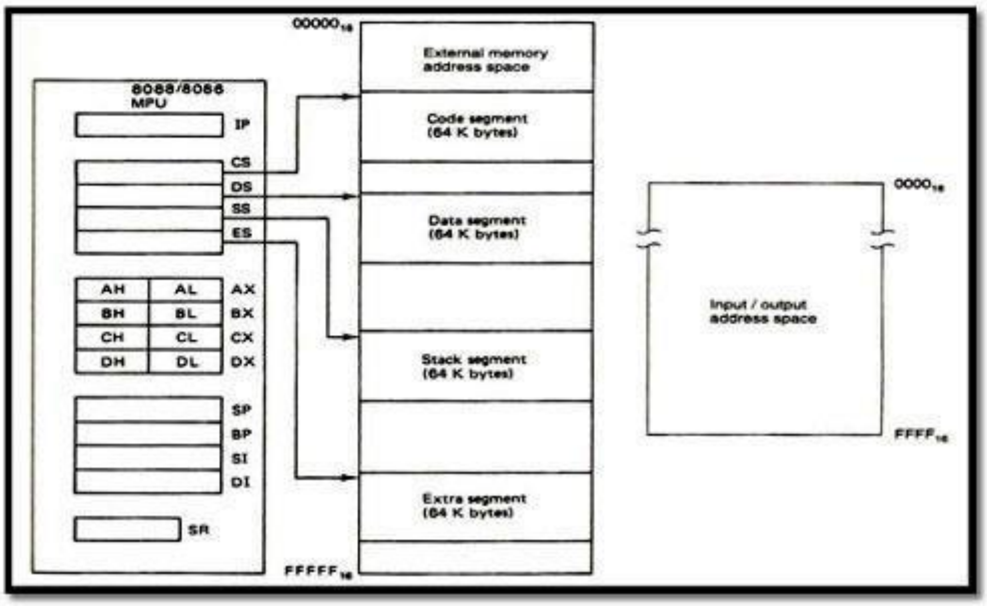
These three registers are used to store temporary storage of data like general purpose registers .They hold the 16 bit offset data of the data word in one of the segment

Programming model

How can a 20-bit address be obtained, if there are only 16-bit registers?
However, the largest register is only 16 bits (64k); so physical addresses have to be calculated. These calculations are done in hardware within the microprocessor.
The 16-bit contents of segment register gives the starting/ base address of particular segment. To address a specific memory location within a segment we need an offset address. The offset address is also 16-bit wide and it is provided by one of the associated pointer or index register.

To be able to program a microprocessor, one does not need to know all of its hardware architectural features. What is important to the programmer is being aware of the various registers within the device and to understand their purpose, functions, operating capabilities, and limitations.

The above figure illustrates the software architecture of the 8086 microprocessor. From this diagram, we see that it includes fourteenl6-bit internal registers: the instruction pointer (IP), four data registers (AX, BX, CX, and DX), two pointer registers (BP and SP), two index registers (SI and DI), four segment registers (CS, DS, SS, and ES) and status register (SR), with nine of its bits implemented as status and control flags.


00000₁₆
External memory address space
8088/8086 MPU
IP
CS
DS
SS
ES
Code segment (64 K bytes)
Data segment (64 K bytes)
Stack segment (64 K bytes)
Extra segment (64 K bytes)
AH
AL
AX
BH
BL
BX
CH
CL
CX
DH
DL
DX
SP
BP
SI
DI
SR
FFFFF₁₆
0000₁₆
Input / output address space
FFFF₁₆

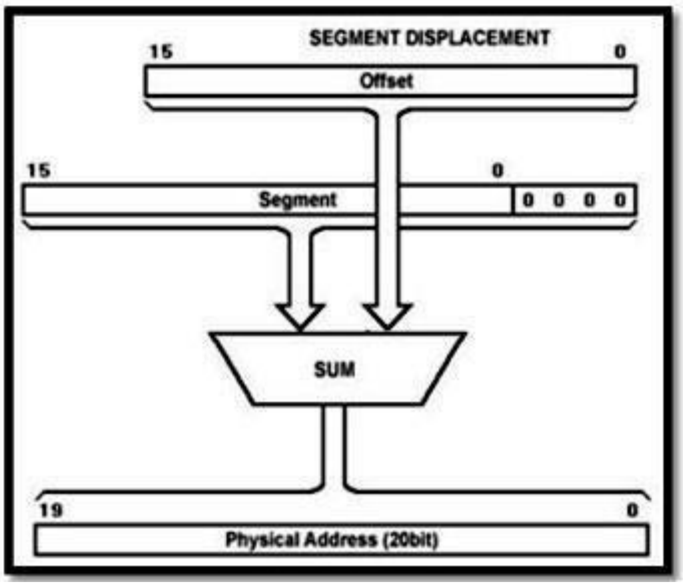
The point to note is that the beginning segment address must begin at an address divisible by 16.Also note that the four segments need not be defined separately. It is allowable for all four segments to completely overlap (CS = DS = ES = SS).

Logical and Physical Address

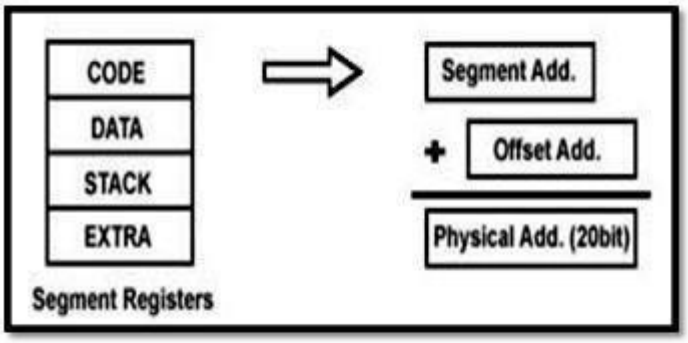
Addresses within a segment can range from address 00000h to address 0FFFFh. This corresponds to the 64K-bytelength of the segment. An address within a segment is called an offset or logical address.

A logical address gives the displacement from the base address of the segment to the desired location within it, as opposed to its "real" address, which maps directly anywhere into the 1 MByte memory space. This "real" address is called the physical address.

What is the difference between the physical and the logical address? The physical address is 20 bits long and corresponds to the actual binary code output by the BIU on the address bus lines. The logical address is an offset from location 0 of a given segment.

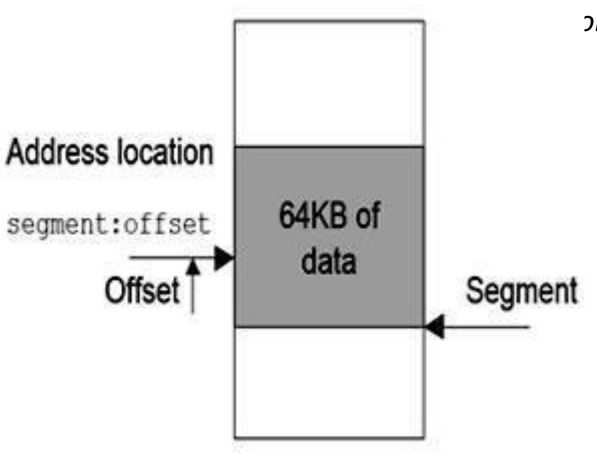


You should also be careful when writing addresses on paper to do so clearly. To specify the logical address XXXX in the stack segment, use the convention SS:XXXX, which is equal to [SS] * 16 + XXXX.



Logical address is in the form of: Base Address: Offset Offset is the displacement of the memory location from the starting location of the segment. To calculate the physical address of the memory, BIU uses the following formula:

Physical Address = Base Address of Segment * 16 + Offset



Example:

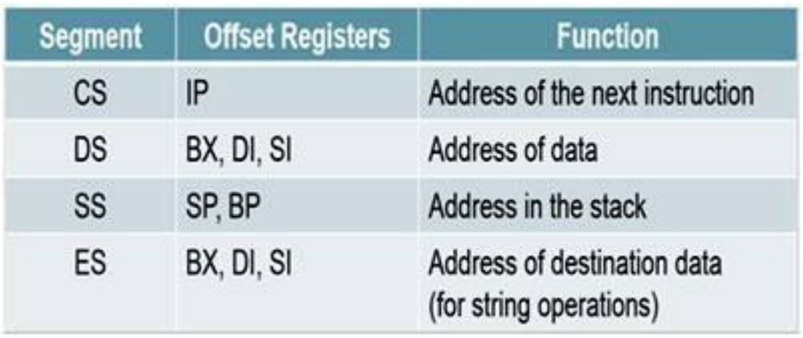
The value of Data Segment Register (DS) is 2222H.

To convert this 16-bit address into 20-bit, the BIU appends 0H to the LSB (by multiplying with 16) of the address. After appending, the starting address of the Data Segment becomes 22220H.

Data at any location has a logical address specified as:2222H: 0016H

Where 0016H is the offset, 2222 H is the value of DS Therefore the physical address:22220H + 0016H
: 22236 H

The following table describes the default offset values to the corresponding memory segments.



Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

Some of the advantages of memory segmentation in the 8086 are as follows:

- With the help of memory segmentation a user is able to work with registers having only 16-bits.
- The data and the user’s code can be stored separately allowing for more flexibility.

- Also due to segmentation the logical address range is from 0000H to FFFFH the code can be loaded at any location in the memory.

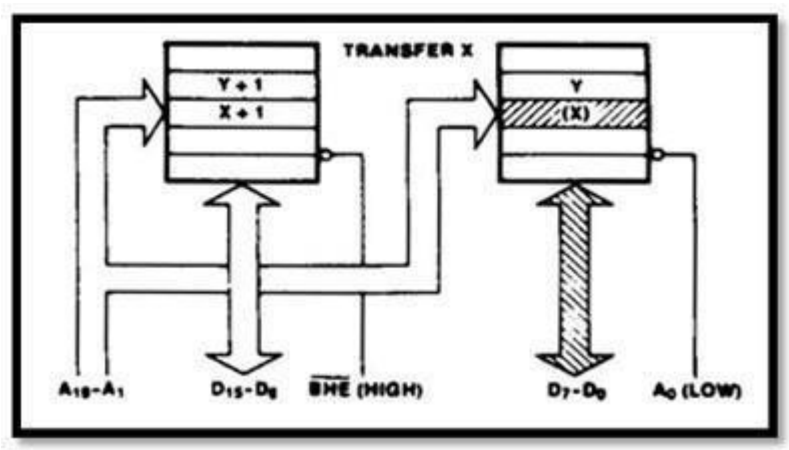
Physical memory organization:

The 8086's 1Mbyte memory address space is divided in to two independent 512Kbyte banks: the low (even) bank and the high (odd) bank. Data bytes associated with an even address (0000016, 0000216, etc.) reside in the low bank, and those with odd addresses (0000116, 0000316, etc.) reside in the high bank.

Address bits A1 through A19 select the storage location that is to be accessed. They are applied to both banks in parallel. A0and bank high enable (BHE) are used as bank-select signals.

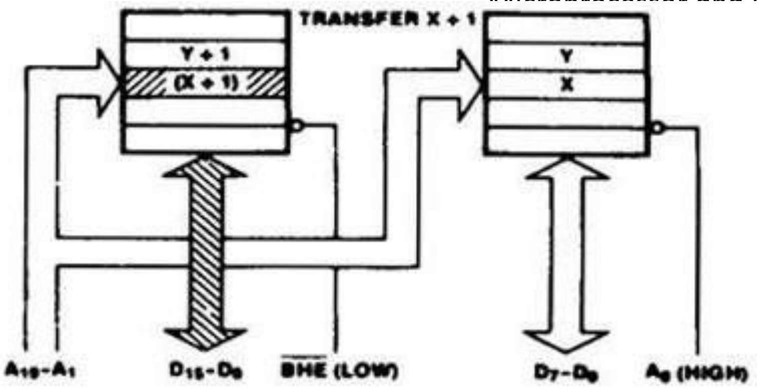
The four different cases that happen during accessing data:

Case 1: When a byte of data at an even address (such as X) is to be accessed:



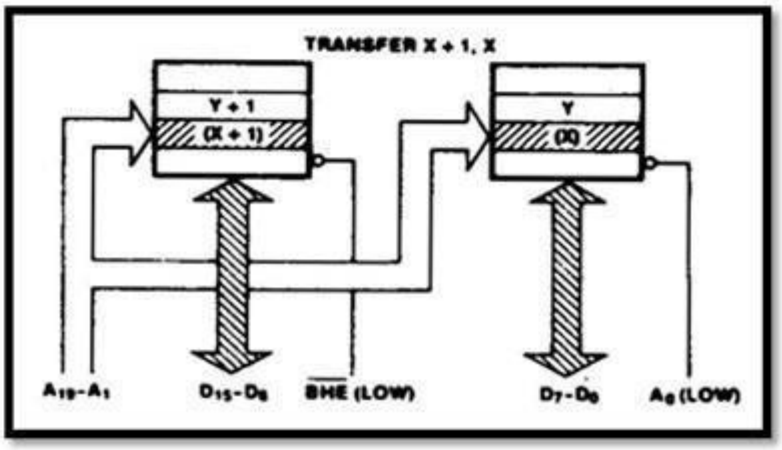
- A0 is set to logic 0 to enable the low bank of memory.
- BHE is set to logic 1 to disable the high bank.

Case 2: When a byte of data at an odd addresses (such as X+1) is to be accessed:



- A0is set to logic 1 to disable the low bank of memory.
- BHE is set to logic 0 to enable the high bank.

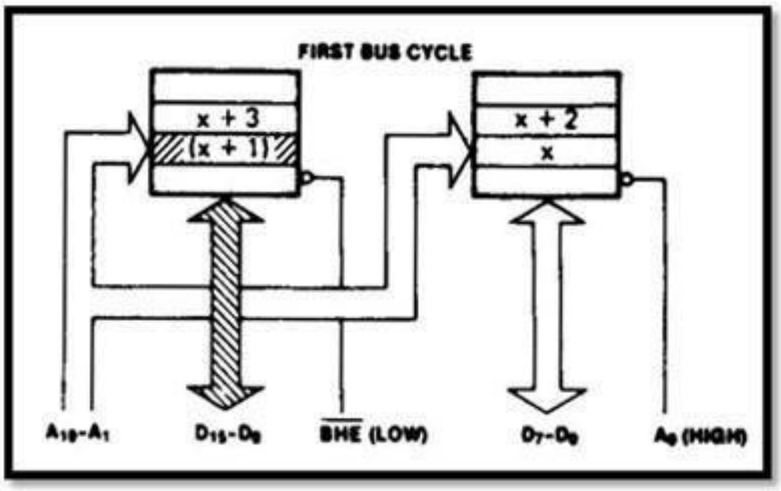
Case 3: When a word of data at an even address (aligned word) is to be accessed:



- A0 is set t
- BHE is se

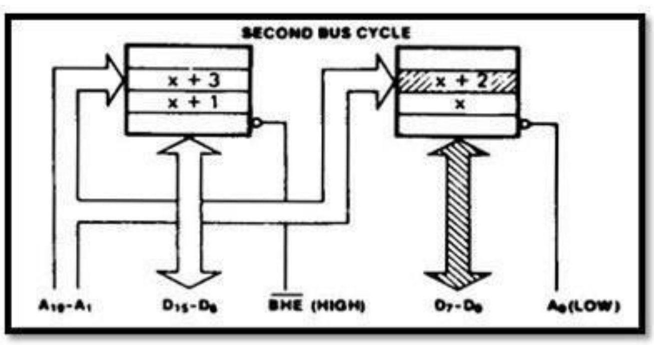
Case 4: When ccessed, then the 8086 need

a) During the first bus cycle, the odd byte of the word (in the high bank) is addressed



- A0 is set to logic 1 to disable the low bank of memory
- BHE is set to logic 0 to enable the high bank.

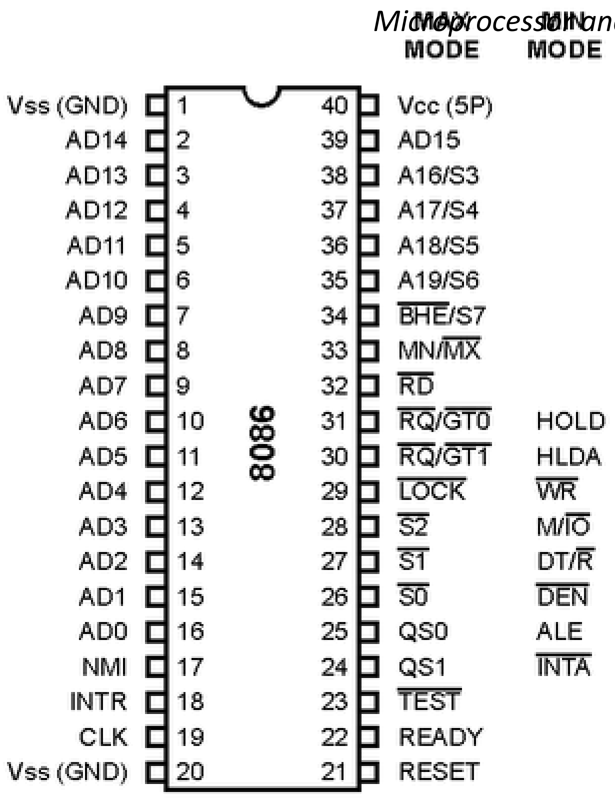
b) During the second bus cycle, the odd byte of the word (in the low bank) is addressed



- A0is set to logi
- BHE is set to lc

Signal Descriptior

The 8086 Microprocessor is a 16-bit CPU available in 3 clock rates, i.e. 5, 8 and 10MHz, packaged in a 40 pin CERDIP or plastic package. The 8086 Microprocessor operates in single processor or multiprocessor configurations to achieve high performance. The pin configuration is as shown in fig1. Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multiprocessor mode) configuration.



The 8086 signals can be categorized in three groups. The first are the signals having common functions in minimum as well as maximum mode, the second are the signals which have special functions in minimum mode and third are the signals having special functions for maximum mode.

The following signal description is common for both the minimum and maximum modes.

AD15-AD0:

These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, TW and T4. Here T1, T2, T3, T4 and TW are the clock states of a machine cycle. TW is await state. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

A19/S6, A18/S5, A17/S4, A16/S3:

These are the time multiplexed address and status lines. During T1, these are the most significant address lines or memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2, T3, TW and T4 .The status of the interrupt enable flag bit(displayed on S5) is updated at the beginning of each clock cycle. The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as shown in Table 1.1.

These lines float to tri-state off (tristated) during the local bus hold acknowledge. The status line S6 is always low(logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

Table 1.1 Bus High Enable/Status

BHE/S7 (Active Low):

The bus high enable signal is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in Table 1.2. It goes low for the data transfers over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. $\overline{BHE}$ is low during T1 for read, write and interrupt acknowledge cycles, when- ever a byte is to be transferred on the higher byte of the data bus. The status information is available during T2, T3 and T4. The signal is active low and is tristated during 'hold'. It is low during T1 for the first pulse of the interrupt acknowledge cycle.

$\overline{BHE}$	A_0	Indication
0	0	Whole Word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	None

$\overline{RD}$-Read:

Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. $\overline{RD}$ is active low and shows the state for T2, T3, TW of any read cycle. The signal remains tristated during the 'hold acknowledge'.

READY:

This is the acknowledgement from the slow devices or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. The signal is active high.

INTR-Interrupt Request:

This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

TEST:

This input is examined by a 'WAIT' instruction. If the TEST input goes low, execution will continue, else, the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

NMI-Non-maskable Interrupt:

This is an edge-triggered input which causes a Type2 interrrupt. The NMI is not maskable internally by software. A transition from low to high initiates the interrupt response at the end of the current instruction. This input is internally synchronized.

RESET:

This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronized.

CLK-Clock Input:

The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

VCC :

+5V power supply for the operation of the internal circuit. GND ground for the internal circuit.

MN/MX :

The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode. The following pin functions are for the minimum mode operation of 8086.

M/IO -Memory/IO:

This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active in the previous T4 and remains active till final T4 of the current cycle. It is tristated during local bus "hold acknowledge".

$\overline{\text{INTA}}$-Interrupt Acknowledge:

This signal is used as a read strobe for interrupt acknowledge cycles. In other words, when it goes low, it means that the processor has accepted the interrupt. It is active low during T2, T3 and TW of each interrupt acknowledge cycle.

ALE-Address latch Enable:

This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

DT /$\overline{\text{R}}$-Data Transmit/Receive:

This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low. Logically, this is equivalent to S1 in maximum mode. Its timing is the same as M/I/O. This is tristated during 'hold acknowledge'.

$\overline{\text{DEN}}$-Data Enable

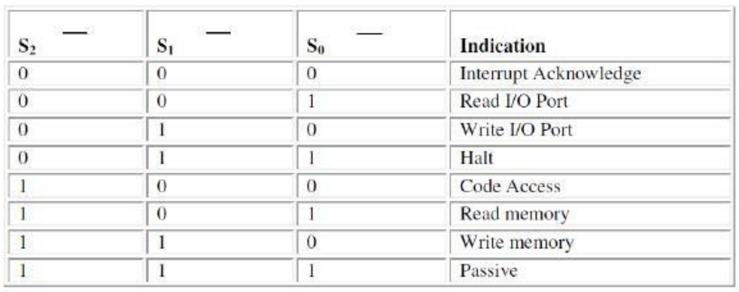
This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle ofT2 until the middle of T4 DEN is tristated during 'hold acknowledge' cycle.

HOLD, HLDA-Hold/Hold Acknowledge:

When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and it should be externally synchronized.

S2, S1, S0 -Status Lines:

These are the status lines which reflect the type of operation, being carried out by the processor. These become active during T4 of the previous cycle and remain active during T1 and T2 of the current bus cycle. The status lines return to passive state during T3 of the current bus cycle so that they may again become active for the next bus cycle during T4. Any change in these lines during T3 indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in table 1.3



$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	**Indication**
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

Table 1.3

$\overline{\text{LOCK}}$:

This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the $\overline{\text{LOCK}}$ signal is low. The $\overline{\text{LOCK}}$ signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

QS1, QS0-Queue Status:

These lines give information about the status of the codeprefetch queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in Table 1.4.

QS_1,	QS_0	Indication
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

Table 1.4

$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$- ReQuest/Grant:

These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\ \overline{GT}_1$, $\overline{RQ}/\overline{GT}$ pins have internal pull-up resistors and may be left unconnected. The request! Grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.

2. During T4 (current) or T1 (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.

3. A one clock wide pulse from the another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle.

Minimum Mode 8086 System and Timings

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX* pin to logic1. In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system. The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

Latches:

The latches are generally buffered output D-type flip-flops, like, 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

Transreceivers

Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signal. They are controlled by two signals, namely, DEN* and DT/R*. The DEN* signal indicates that the valid data is available on the data bus, while DT/R indicates the direction of data,
i.e. from or to the processor.

Memory:

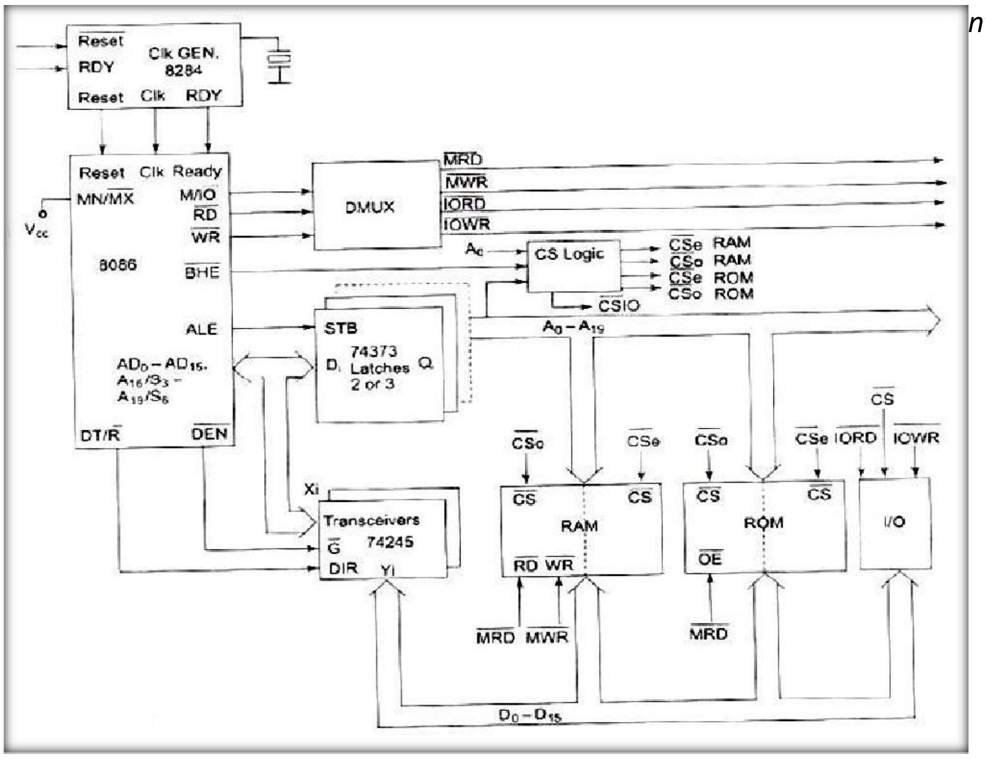
The system contains memory for the monitor and users program storage. Usually, EPROMS are used for monitor storage, while RAMs for users program storage.

IO Devices:

A system may contain I/O devices for communication with the processor as well as some special purpose I/O devices.

Clock Generator:

The clock generator generates the clock from the crystal oscillator and then shapes it and divides to make it more precise so that it can be used as an accurate timing reference for the system. The clock generator also synchronizes some external signals with the system clock.



The general system organization is shown in above fig .Since it has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.

The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations. The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts.

1) Timing diagram for read cycle
2) Timing diagram for write cycle.

Timing diagram for Read cycle :

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal and also M/IO* signal. During the negative going edge of this signal, the valid address is latched on the local bus. The BHE* and A0 signals address low, high or both bytes. From Tl to T4, the M/IO* signal indicates a memory or I/O operation. At T2 the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD*) control signal is also activated in T2 .

The read (RD) signal causes the addressed device to enable its data bus drivers. After RD* goes low, the valid data is available on the data bus.

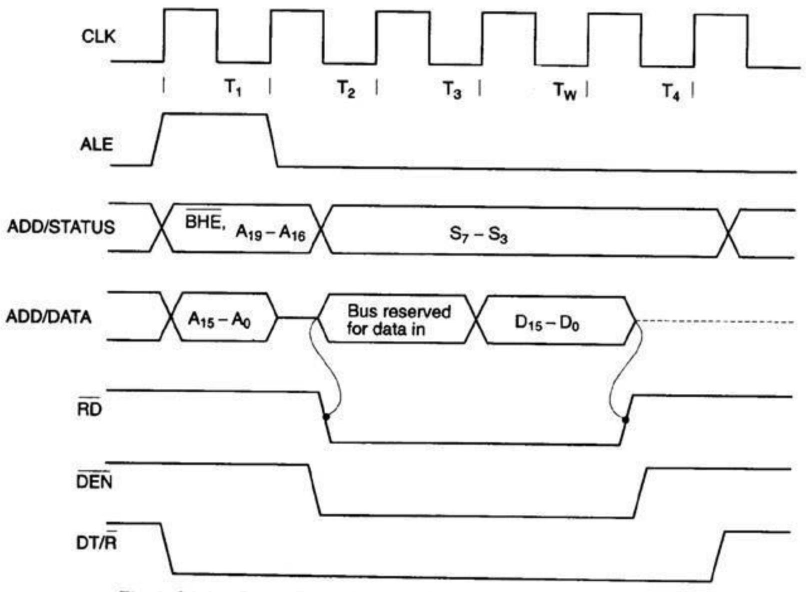
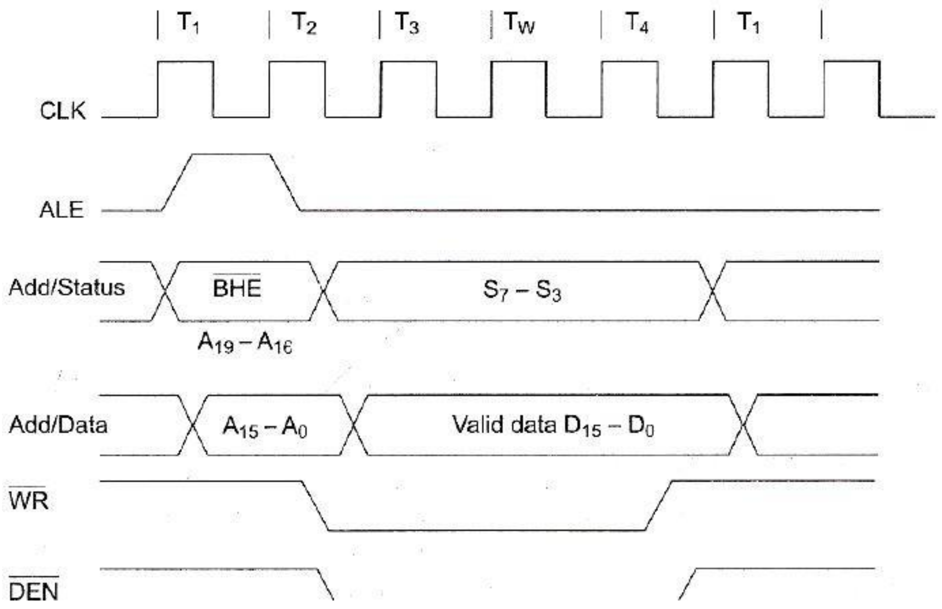


Fig.1.9(a) *Read Cycle Timing Diagram for Minimum Mode*

The addressed device will drive the READY line high, when the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.

Timing diagram for write cycle:

A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO* signal is again asserted to indicate a memory or I/O operation. In T2 after sending the address in Tl the processor sends the data to be written to the addressed location. The data remains on the bus until middle of T4 state. The WR* becomes active at the beginning of T2.

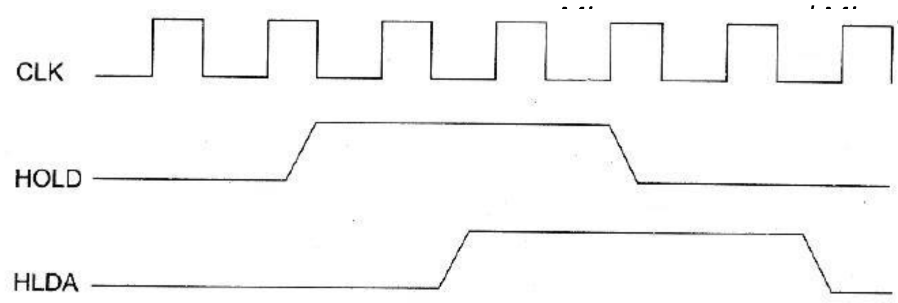


The BHE* and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written. The M/IO*, RD* and WR* signals indicate the types of data transfer as specified in Table

M/IO	RD	WR	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

HOLD Response Sequence

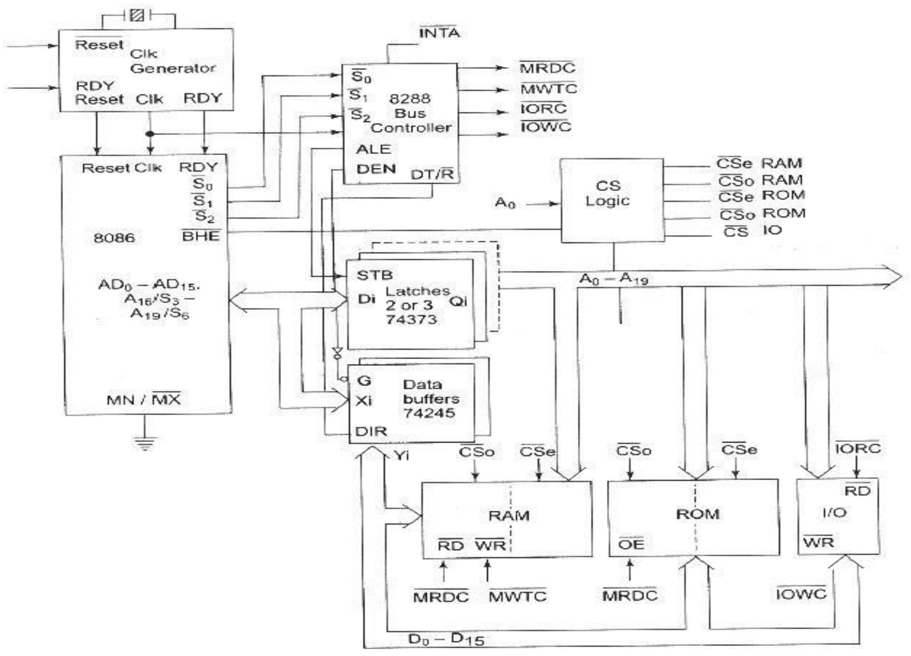
The HOLD pin is checked at the end of the each bus cycle. If it is received active by the processor before T4 of the previous cycle or during T1 state of the current cycle, the CPU activities HLDA in the next clock cycle and for the succeeding bus cycles, the bus will be given to another requesting master The control control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock as shown in fig



Maximum Mode 8086 System and Timings

In the maximum mode, the 8086 is operated by strapping the MN/MX* pin to ground. In this mode, the processor derives the status signals S2*, S1* and S0*. Another chip called bus controller derives the control signals using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration. The other components in the system are the same as in the minimum mode system. The general system organization is as shown in the fig1.1

The basic functions of the bus controller chip IC8288, is to derive control signals like RD* and WR* (for memory and I/O devices), DEN*, DT/R*, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip has input lines S2*, S1* and S0* and CLK. These inputs to 8288 are driven by the CPU. It derives the outputs ALE, DEN*, DT/R*, MWTC*, AMWC*, IORC*, IOWC* and AIOWC*. The AEN*, IOB
and CEN pins are specially useful for multiprocessor systems. AEN* and IOB are generally grounded. CEN pin is usually tied to +5V.



INTA* pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.IORC*, IOWC* are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data from or to the bus. For both of these write command signals, the advanced signals namely AIOWC* and AMWTC* are available. They also serve the same purpose, but are activated one clock cycle earlier than the IOWC* and MWTC* signals, respectively. The maximum mode system is shown in fig. 1.1.

The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are similar to the minimum mode. ALE is asserted in T1, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals. The fig. 1.2 shows the maximum mode timings for the read operation while the fig. 1.3 shows the same for the write operation.

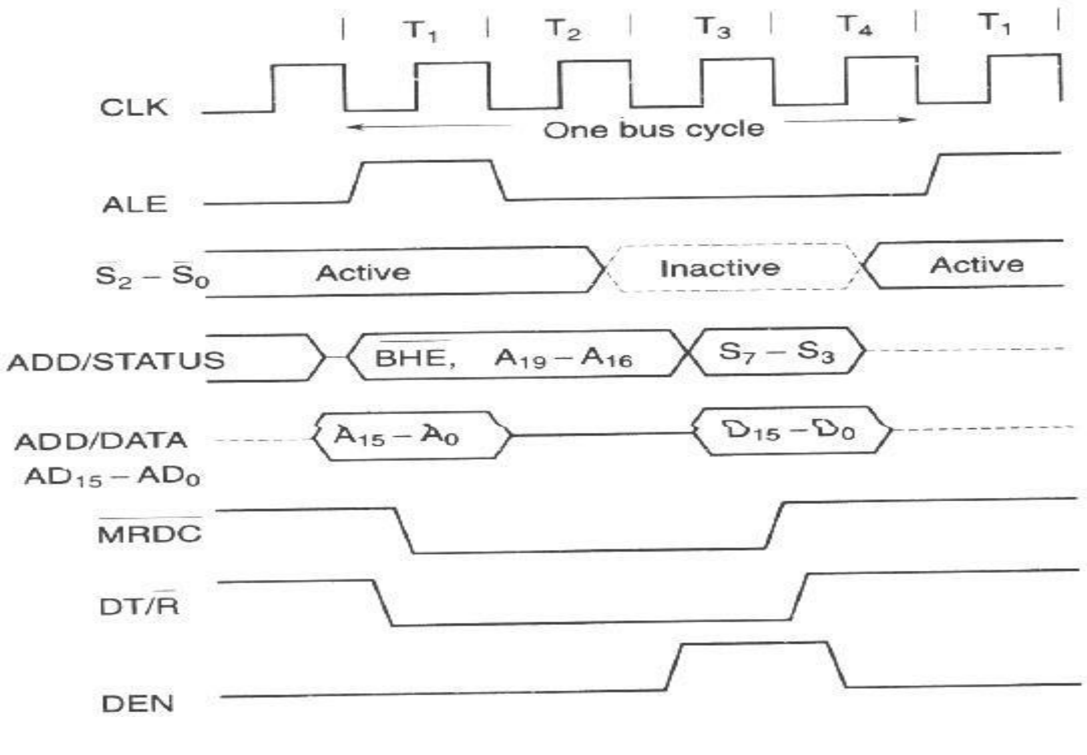


Fig. 1.2 Memory Read Timing in Maximum Mode

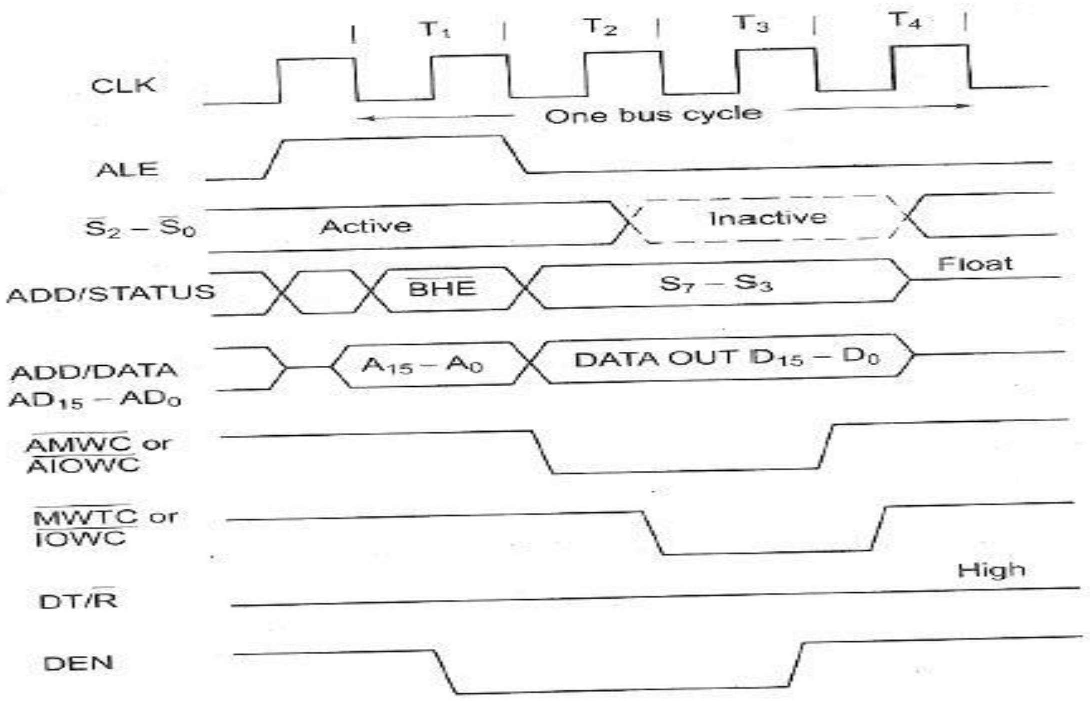


Fig. 1.3 Memory Write Timing in Maximum Mode

I/O Interface

- 8255 PPI
- Various Modes of Operation and Interfacing to 8086
- D/A and A/D Converter
- Memory Interfacing to 8086
- Interrupt Structure of 8086
- Interrupt Vector Table, Interrupt Service Routine
- architecture of 8259 keyboard and display controller.

I/O Interface

Introduction:

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. User can give information to the microprocessor based system using keyboard and user can see the result or output information from the microprocessor based system with the help of display device. The transfer of data between keyboard and microprocessor, and microprocessor and display device is called input/output data transfer or I/O data transfer. This data transfer is done with the help of I/O ports.

Input port:

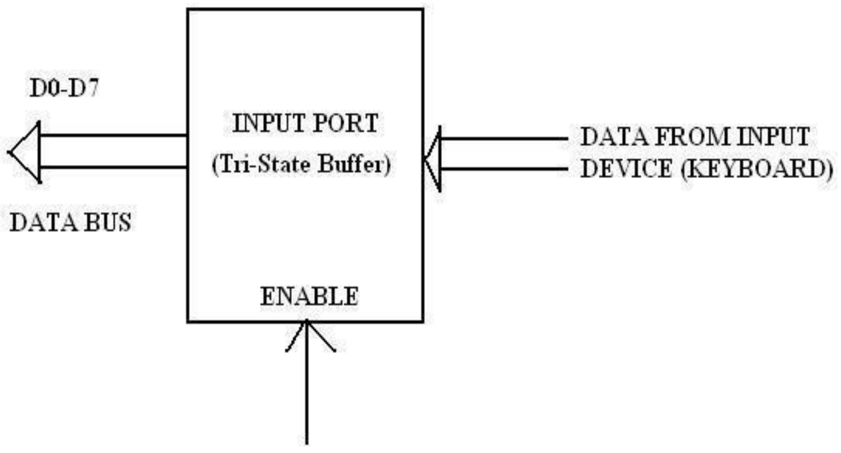


FIG.1 INPUT PORT

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer, as shown in the fig.1. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

Output port:

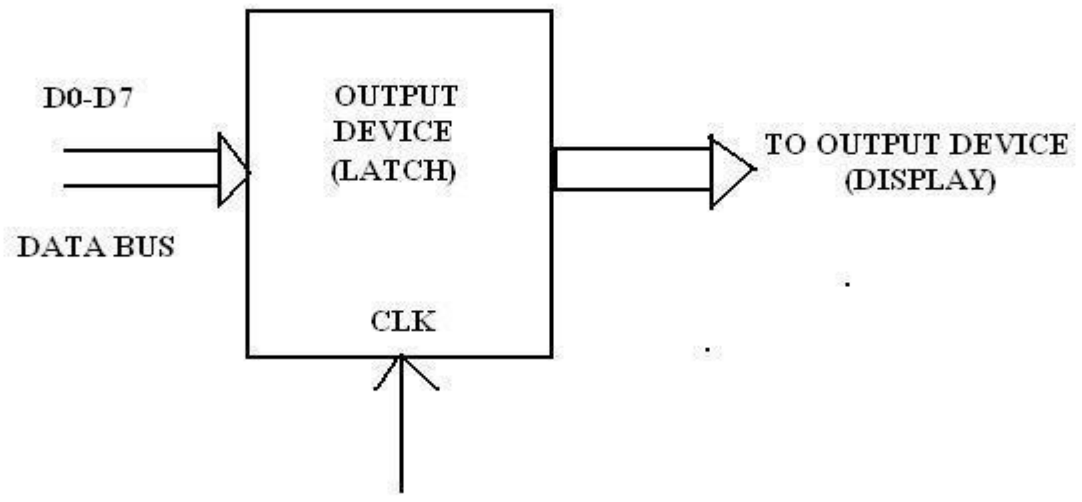


FIG.2 OUTPUT PORT

SIt is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch, as shown in the fig.2. When microprocessor wants to send data to the output device is puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

Serial and Parallel Transmission:

In telecommunications, serial transmission is the sequential transmission of signal elements of a group representing a character or other entity of data. Digital serial transmissions are bits sent over a single wire, frequency or optical path sequentially. Because it requires less signal processing and less chance for error than parallel transmission, the transfer rate of each individual path may be faster. This can be used over longer distances as a check digit or parity bit can be sent along it easily.

In telecommunications, parallel transmission is the simultaneous transmission of the signal elements of a character or other entity of data. In digital communications, parallel transmission is the simultaneous transmission of related signal elements over two or more separate paths. Multiple electrical wires are used which can transmit multiple bits simultaneously, which allows for higher data transfer rates than can be achieved with serial transmission. This method is used internally within the computer, for example the internal buses, and sometimes externally for such things as printers, The major issue with this is "skewing" because the wires in parallel data transmission have slightly different properties (not intentionally) so some bits may arrive before others, which may corrupt the message. A parity bit can help to reduce this. However, electrical wire parallel data transmission is therefore less reliable for long distances because corrupt transmissions are far more likely.

Interrupt driven I/O:

In this technique, a CPU automatically executes one of a collection of special routines whenever certain condition exists within a program or a processor system. Example CPU gives response to devices such as keyboard, sensor and other components when they request for service. When the CPU is asked to communicate with devices, it services the devices. Example each time you type a character on a keyboard, a keyboard service routine is called. It transfers the character you typed from the keyboard I/O port into the processor and then to a data buffer in memory.

The interrupt driven I/O technique allows the CPU to execute its main program and only stop to service I/O device when it is told to do so by the I/O system as shown in fig.3. This method provides an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is completed, the processor would resume exactly where it left off.

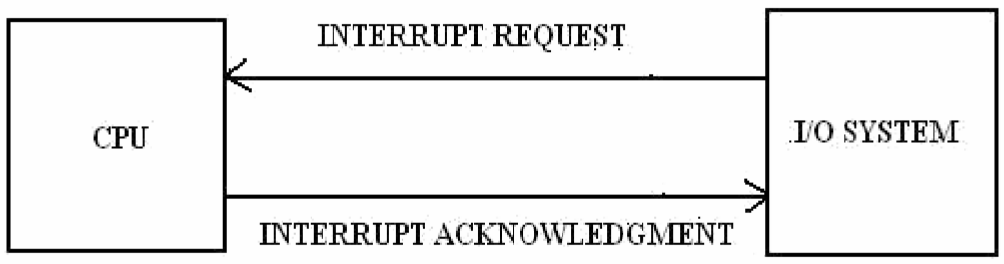


FIG.3 INTERRUPT DRIVEN I/O

An analogy to the interrupt concept is in the classroom, where the professor serves as CPU and the students as I/O ports. The classroom scenario for this interrupt analogy will be such that the professor is busy in writing on the blackboard and delivering his lecture.

The student raises his finger when he wants to ask a question (student requesting for service). The professor then completes his sentence and acknowledges student‟s request by saying "YES" (professor acknowledges the interrupt request). After acknowledgement from the professor, student asks the question and professor gives answer to the question (professor services the interrupt). After that professor continues its remaining lecture form where it was left.

PIO 8255:

The parallel input-output port chip 8255 is also called as programmable**peripheral input-output port.** The Intel‟s 8255 are designed for use with Intel‟s 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lineswhich may be individually programmed in two groups of twelve lines each, orthree groups of eight lines.

The two groups of I/O pins are named as Group A and Group B. Each of thesetwo groups contains a subgroup of eight I/O lines called as 8-bit port and anothersubgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port Aalong with a 4-bit port C upper.

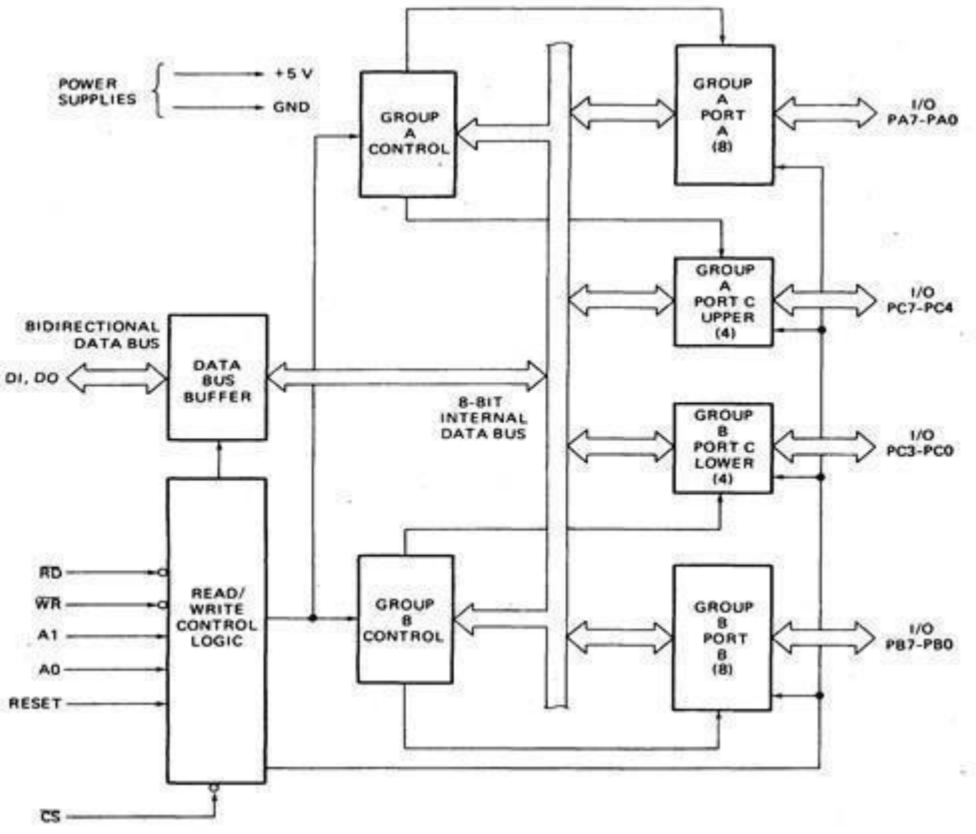
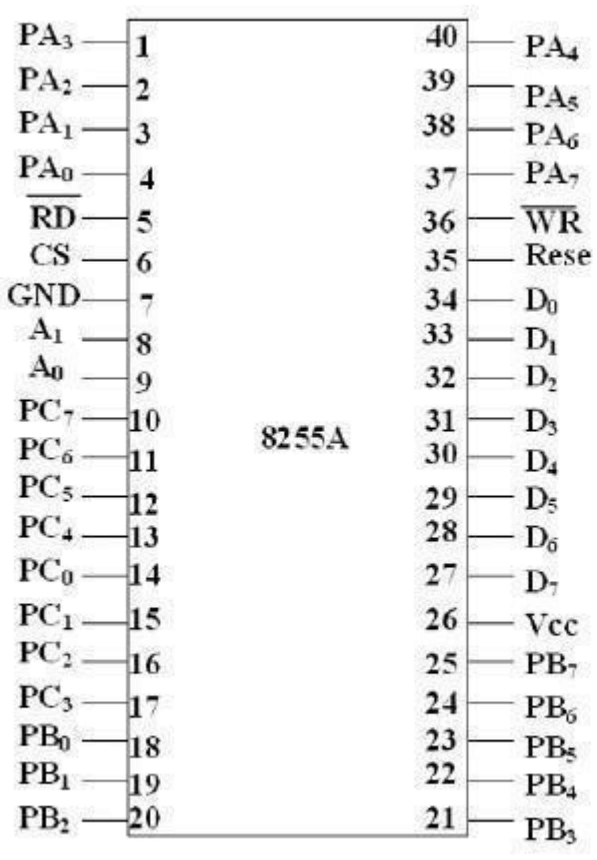


FIGURE Internal block diagram of 8255A programmable parallel port device. (*Intel Corporation*)

The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7 similarly. Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0-PC3. The port C upper and port C lower can be used in combination as an 8-bit port C. Both the port Cs is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit I/O ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR). The internal block diagram and the pin configuration of 8255 are shown in figs.

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfer of both data and control words. RD, WR, A1, A0 and RESET are the inputs, provided by the microprocessor to READ/WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus. This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

Pin Diagram of 8255A



8255A Pin Configuration

The pin configuration of 8255 is shown in fig.

- The port A lines are identified by symbols PA0-PA7 while the port C lines are
- Identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3. The port C upper and port C lower can be used in combination as an 8-bit port C.
- Both the port C is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be

achieved by programming the bits of an internal register of 8255 called as control word register (CWR).

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words.

RD,WR, A1, A0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.

This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

The signal description of 8255 is briefly presented as follows:

PA7-PA0: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

PC7-PC4: Upper nibble of port C lines. They may act as either output latches or input buffers lines.

This port also can be used for generation of handshake lines in mode1 or mode2.

PC3-PC0: These are the lower port C lines; other details are the same as PC7-PC4 lines.

PB0-PB7: These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

RD: This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

WR: This is an input line driven by the microprocessor. A low on this line indicates write operation.

CS: This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected.

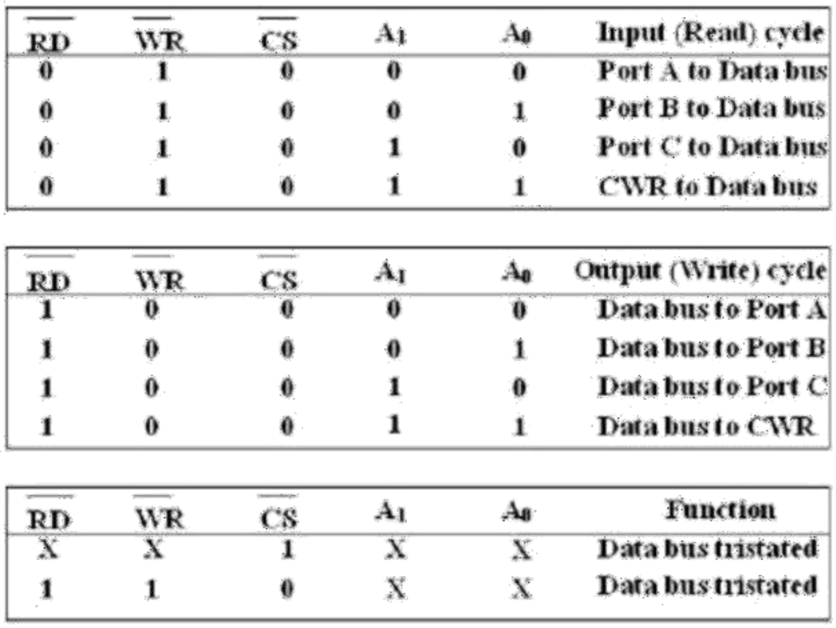
D0-D7: These are the data bus lines those carry data or control word to/from the microprocessor.

RESET: Logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

A1-A0: These are the address input lines and are driven by the microprocessor.

These lines A1-A0 with RD, WR and CS from the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.

In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.



$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Input (Read) cycle
0	1	0	0	0	Port A to Data bus
0	1	0	0	1	Port B to Data bus
0	1	0	1	0	Port C to Data bus
0	1	0	1	1	CWR to Data bus

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Output (Write) cycle
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Function
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

Control Word Register

Modes of Operation of 8255

These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).

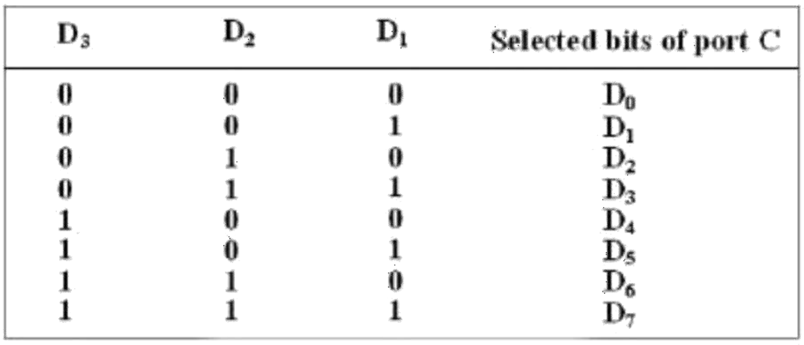
In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC0-PC7) can be used to set or reset its individual port bits.

Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.

BSR Mode: In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.

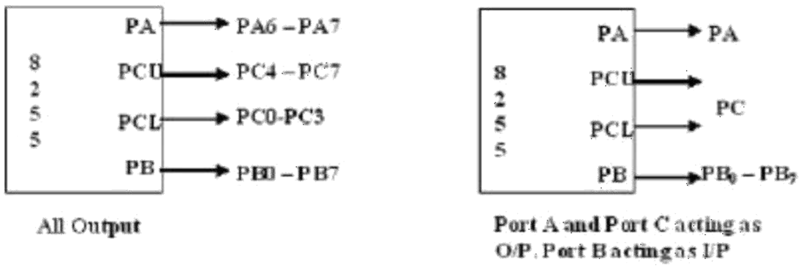
I/O Modes:

a) Mode 0 (Basic I/O mode): This mode is also called as basic input/output Mode. This mode provides simple input and output capabilities using each of the threeports. Data can be simply read from and written to the input and output portsrespectively, after appropriate initialization.



D3	D2	D1	Selected bits of port C
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

BSR Mode : CWR Format



Mode 0

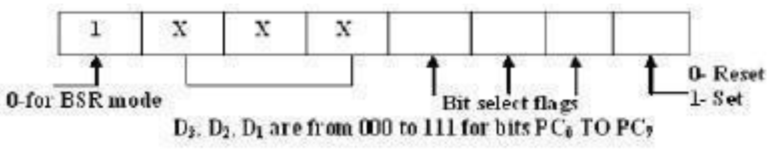
The salient features of this mode are as listed below:

1. Two 8-bit ports (port A and port B) and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combined used as a third 8-bit port.
2. Any port can be used as an input or output port.
3. Output ports are latched. Input ports are not latched.
4. A maximum of four ports are available so that overall 16 I/O configurations arepossible.

All these modes can be selected by programming a register internal to 8255known as CWR.

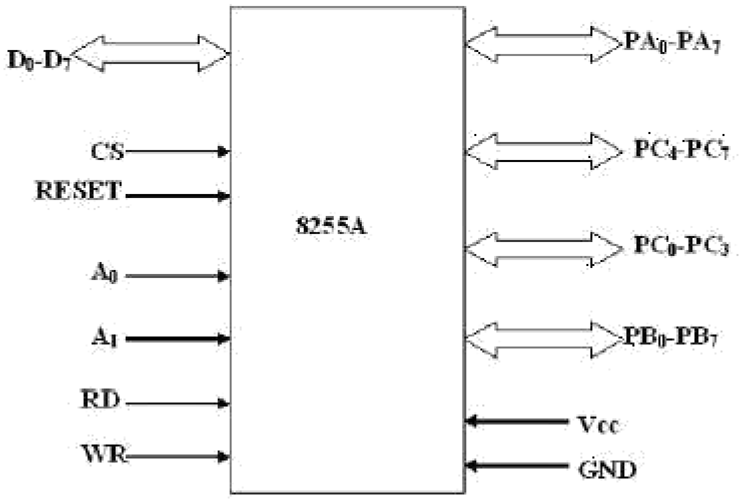
The control word register has two formats. The first format is valid for I/O modesof operation, i.e. modes 0, mode 1 and mode 2 while the second format is validfor bit set/reset (BSR) mode of operation.

These formats are shown in followingfig.

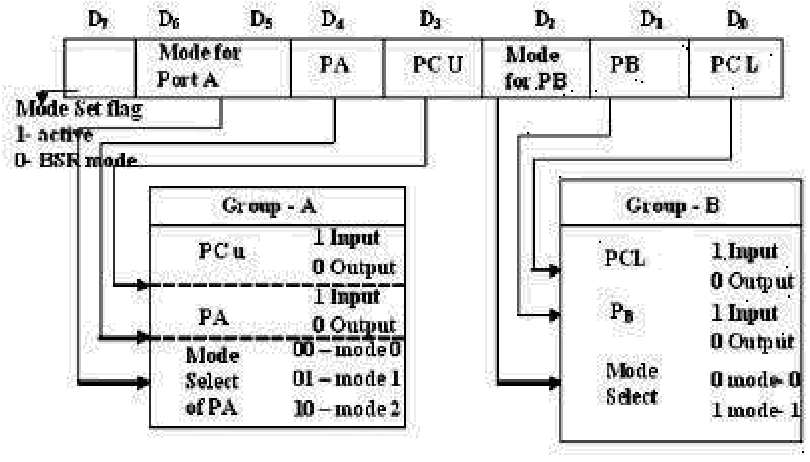


1
X
X
X
0-for BSR mode
Bit select flags
0- Reset
1- Set
D_3, D_2, D_1 are from 000 to 111 for bits PC_0 TO PC_7

I/O Mode Control Word Register Format and BSR Mode Control Word Register Format



Signals of 8255



Control Word Format of 8255

b) Mode 1: (Strobed input/output mode) In this mode the handshaking control the input and output action of the specified port. Port C lines PC0-PC2, provide strobe orhandshake lines for port B. This group which includes port B and PC0-PC2 is called asgroup B for Strobed data input/output. Port C lines PC3-PC5 provides strobe lines for portA.This group including port A and PC3-PC5 from group A. Thus port C is utilized forgenerating handshake signals.

The salient features of mode 1 are listed as follows:

1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.
3. The 8-bit data port can be either used as input and output port. The inputs andoutputs both are latched.
4. Out of 8-bit port C, PC0-PC2 are used to generate control signals for port B andPC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 may be used as independent data lines.

The control signals for both the groups in input and output modes areexplained as follows:

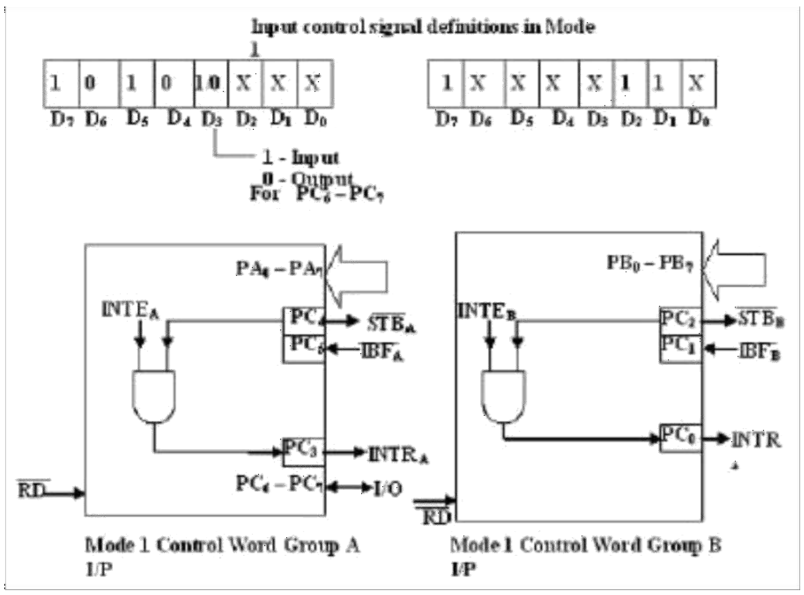
Input control signal definitions (mode 1):

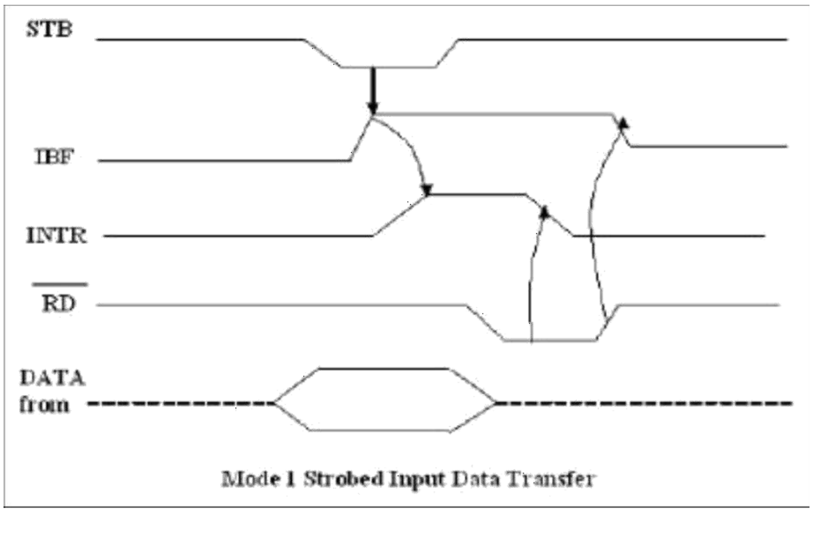
- **STB** (Strobeinput) – If this lines falls to logic low level, the data available at 8-bit input port is loaded into input latches.
- **IBF** (Input buffer full) – If this signal rises to logic 1, it indicates that data hasbeen loaded into latches, i.e. it works as an acknowledgement. IBF is set by a lowon STB and is reset by the rising edge of RD input.
- **INTR** (Interruptrequest) – This active high output signal can be used tointerrupt the CPU whenever an input device requests the service. INTR is set by ahigh STBpin and a high at IBF pin. INTE is an internal flag that can be controlledby the bit set/reset mode of either PC4 (INTEA) or PC2 (INTEB) as shown in fig.
- INTR is reset by a falling edge of RD input. Thus an external input device can berequest the service of the processor by putting the data on the bus and sending thestrobe signal.

Output control signal definitions (mode 1):

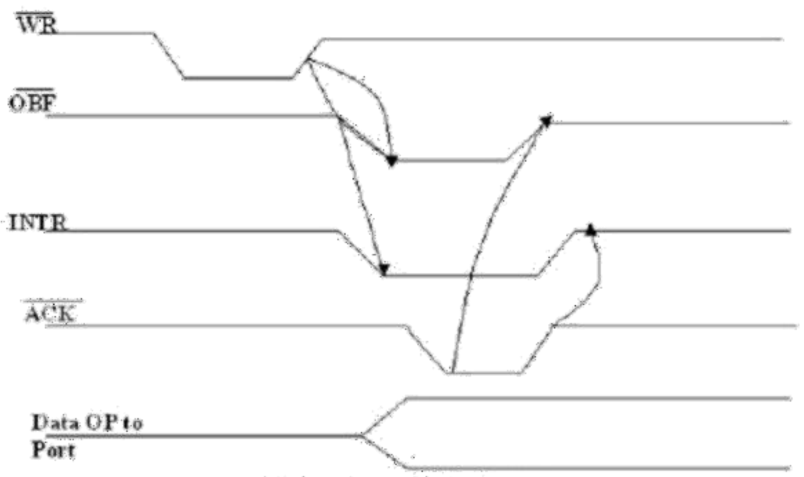
- **OBF** (Output buffer full) – This status signal, whenever falls to low, indicatesthat CPU has written data to the specified output port. The OBF flip-flop will beset by a rising edge of WR signal and reset by a low going edge at the ACKinput.
- ACK (Acknowledgeinput) – ACK signal acts as an acknowledgement to begiven by an output device. ACK signal, whenever low, informs the CPU that thedata transferred by the CPU to the output device through the port is received bythe output device.
- **INTR** (Interruptrequest) – Thus an output signal that can be used to interruptthe CPU when an output device acknowledges the data received from the CPU.INTR is set when ACK, OBF and INTE are 1. It is reset by a

fallingedge on WRinput. The INTEA and INTEB flags are controlled by the bit set-reset mode ofPC6 and PC2 respectively.



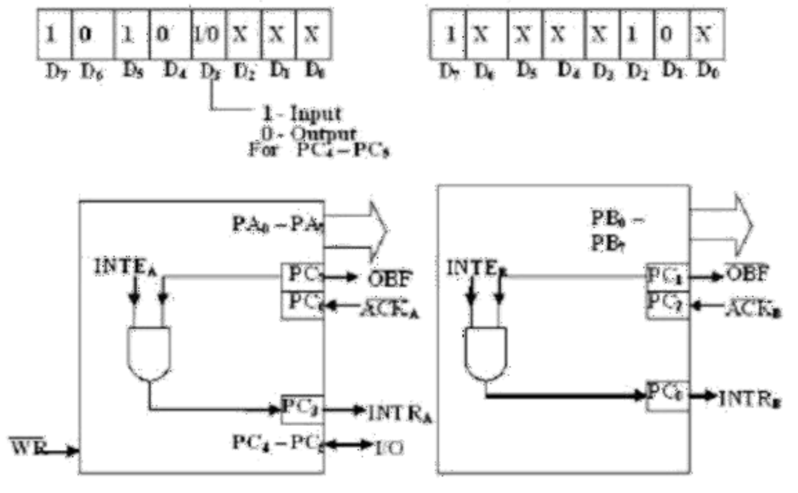


Mode 1 Strobed Input Data Transfer



Mode 1 Strobed Data Output

Output control signal definitions Mode 1



Mode 1 Control Word Group A   Mode 1 Control Word Group B

c) Mode 2 (Strobed bidirectional I/O): This mode of operation of 8255 is alsocalled as strobed bidirectional I/O. This mode of operation provides 8255 with additional features for communicating with a peripheral device on an 8-bit databus. Handshaking signals are provided to maintain proper data flow andsynchronization between the data transmitter and receiver. The interruptgeneration and other functions are similar to mode 1.

In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The Rdand WR signals decide whether the 8255 is going to operate as an input port oroutput port.

The Salient features of Mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C.(PC2 – PC0)
4. Inputs and outputs are both latched.

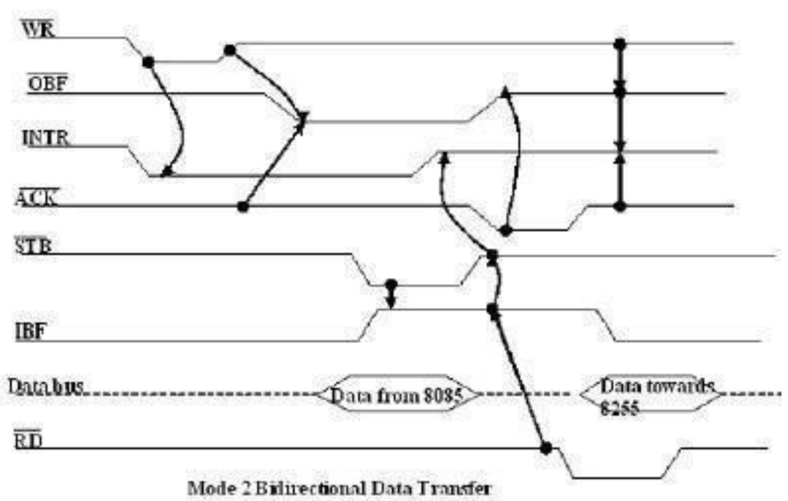
5. The 5-bit control port C (PC3-PC7) is used for generating / accepting handshakesignals for the 8-bit data transfer on port A.

Control signal definitions in mode 2:

- **INTR** – (Interrupt request) As in mode 1, this control signal is active high and isused to interrupt the microprocessor to ask for transfer of the next data byteto/from it. This signal is used for input (read) as well as output (write) operations.
- **Control Signals for Output operations**:
- OBF (Output buffer full) – This signal, when falls to low level, indicates that theCPU has written data to port A.
- ACK (Acknowledge) This control input, when falls to logic low level, Acknowledges that the previous data byte is received by the destination and nextbyte may be sent by the processor. This signal enables the internal tristate buffersto send the next data byte on port A.
- **INTE1** (A flag associated with OBF) This can be controlled by bit set/resetmode with PC6.

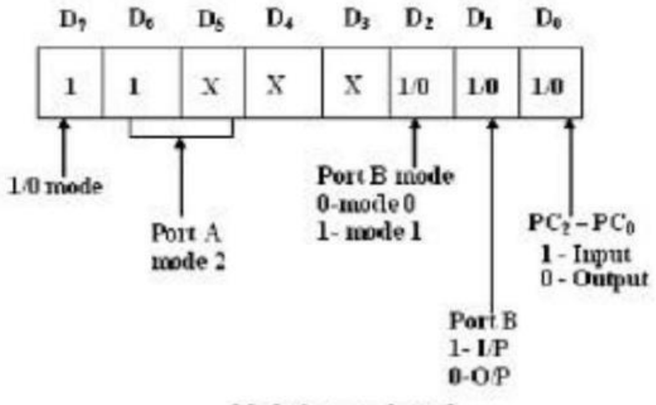
Control signals for input operations:

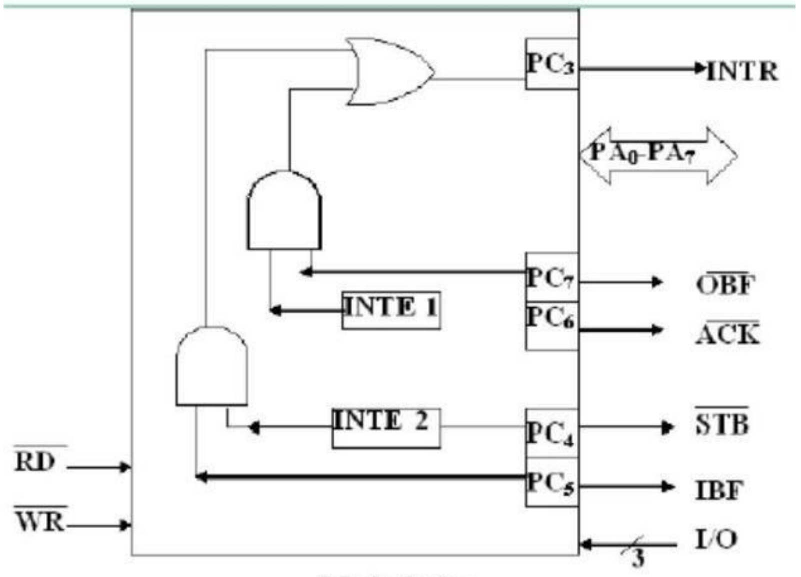
- STB (Strobe input)a low on this line is used to strobe in the data into the inputLatches of 8255.
- **IBF** (Input buffer full) when the data is loaded into input buffer, this signal risesto logic „1‟. This can be used as an acknowledge that the data has been receivedby the receiver.
- The waveforms in fig show the operation in Mode 2 for output as well as inputport.
- Note: WR must occur before ACK and STB must be activated before RD.



Mode 2 Bidirectional Data Transfer

- The following fig shows a schematic diagram containing an 8-bit bidirectionalport, 5-bit control port and the relation of INTR with the control pins. Port B caneither be set to Mode 0 or 1 with port A(Group A) is in Mode 2.
- Mode 2 is not available for port B. The following fig shows the control word.
- The INTR goes high only if IBF, INTE2, STB and RD go high or OBF,
- INTE1, ACK and WR go high. The port C can be read to know the status of theperipheral device, in terms of the control signals, using the normal I/Oinstructions.


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
1 1 X X X 1/0 1/0 1/0
1/0 mode
Port A mode 2
Port B mode 0-mode 0 1- mode 1
Port B 1- I/P 0-O/P
PC_2-PC_0 1 - Input 0 - Output
Mode 2 control word



Mode 2 pins

Interfacing Analog to Digital Data Converters:

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- The analog to digital converters is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
- The process of analog to digital conversion is a slow
- Process and the microprocessor have to wait for the digitaldata till the conversion is over. After the conversion isover, the ADC sends end of conversion EOC signal toinform themicroprocessor that the conversion is over andthe result is ready at the output buffer of the ADC. Thesetasks of issuing an SOC pulse to ADC, reading EOC signalfrom the ADC and reading the digital output of the ADCare carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOCpulse till the active edge of EOC signal is called as theconversion delay of the ADC.
- It may range anywhere from a few microseconds in caseof fast ADC to even a few hundred milliseconds in case ofslow ADCs.
- The available ADC in the market use different conversiontechniques for conversion of analog signal to digitals.Successive approximation techniques and dual slopeintegration techniques are the most popular techniquesused in the integrated ADC chip.
- General algorithm for ADC interfacing contains thefollowing steps:
- Ensure the stability of analog input, applied to the ADC.
- Issue start of conversion pulse to ADC
- Read end of conversion signal to mark the end ofconversion processes.
- Read digital data output of the ADC as equivalent digitaloutput.
- Analog input voltage must be constant at the input of theADC right from the start of conversion till the end of theconversion to get correct results. This may be ensured by asample and hold circuit which samples the analog signaland holds it constant for specific time duration. Themicroprocessor may issue a hold signal to the sample andhold circuit.
- If the applied input changes before the completeconversion process is over, the digital equivalent of theanalog input calculated by the ADC may not be correct.

ADC 0808/0809:

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.
- These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C, as shown. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.
- There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do not contain any internal sample and hold circuit.
- If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

Fig (1) and Fig (2) show the block diagrams and pin diagrams for ADC 0808/0809.

Table.1

Analog I/P selected	**Address lines**		
	C	**B**	**A**
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1

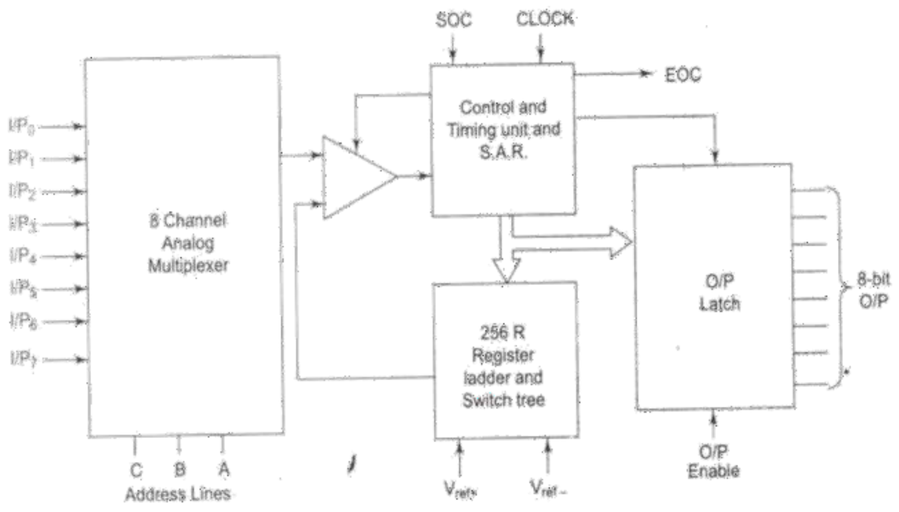
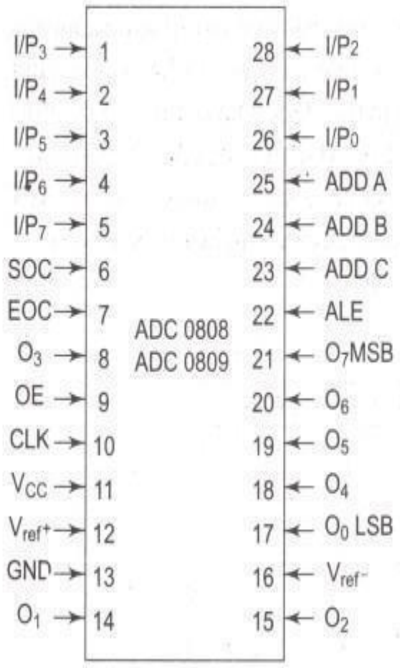

SOC
CLOCK
EOC
I/P0
I/P1
I/P2
I/P3
I/P4
I/P5
I/P6
I/P7
8 Channel Analog Multiplexer
Control and Timing unit and S.A.R.
256 R Register ladder and Switch tree
O/P Latch
8-bit O/P
C B A
Address Lines
Vref+
Vref-
O/P Enable

Fig.1 Block Diagram of ADC 0808/0809

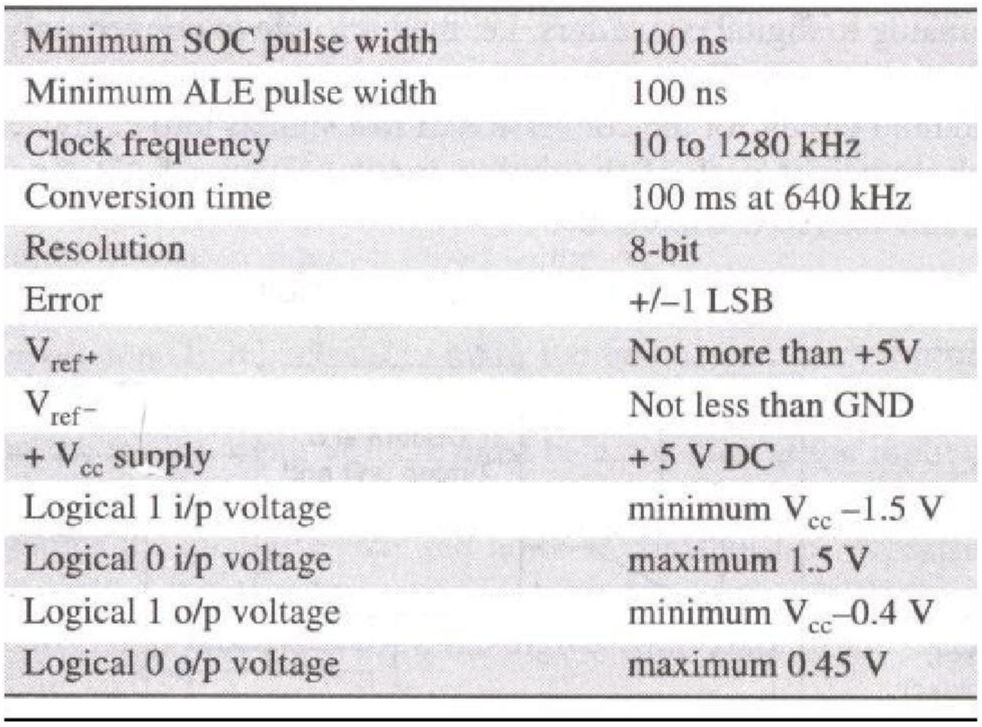


I/P_0 - I/P_7	Analog inputs
ADD A, B, C	Address lines for selecting analog inputs
O_7 - O_0	Digital 8-bit output with O_7 MSB and O_0 LSB
SOC	Start of conversion signal pin
EOC	End of conversion signal pin
OE	Output latch enable pin, if high enable output
CLK	Clock input for ADC
V_{CC}, GND	Supply pins +5V and GND
V_{ref+} and V_{ref-}	Reference voltage positive (+5 Volts maximum) and Reference voltage negative (0V minimum)

Fig.2 Pin Diagram of ADC 0808/0809

Some Electrical Specifications Of The ADC 0808/0809 Are Given In Table.2.

Table.2



Minimum SOC pulse width	100 ns
Minimum ALE pulse width	100 ns
Clock frequency	10 to 1280 kHz
Conversion time	100 ms at 640 kHz
Resolution	8-bit
Error	+/–1 LSB
V_{ref+}	Not more than +5V
V_{ref-}	Not less than GND
+ V_{cc} supply	+ 5 V DC
Logical 1 i/p voltage	minimum V_{cc} –1.5 V
Logical 0 i/p voltage	maximum 1.5 V
Logical 1 o/p voltage	minimum V_{cc}–0.4 V
Logical 0 o/p voltage	maximum 0.45 V

The Timing Diagram Of Different Signals Of Adc0808 Is Shown In Fig.3

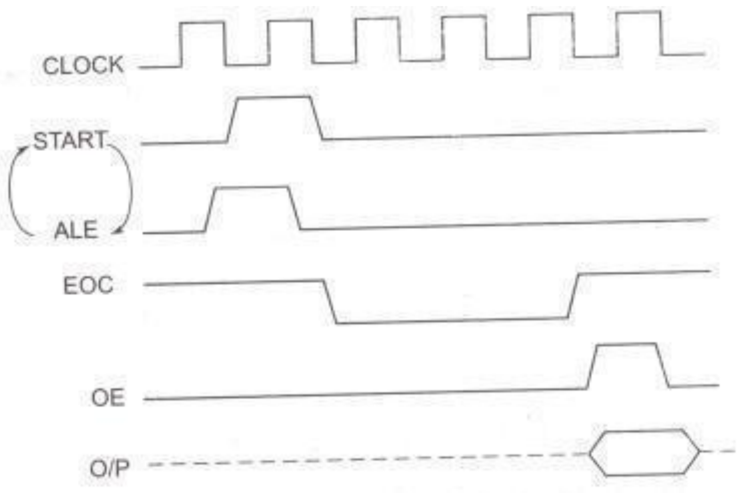
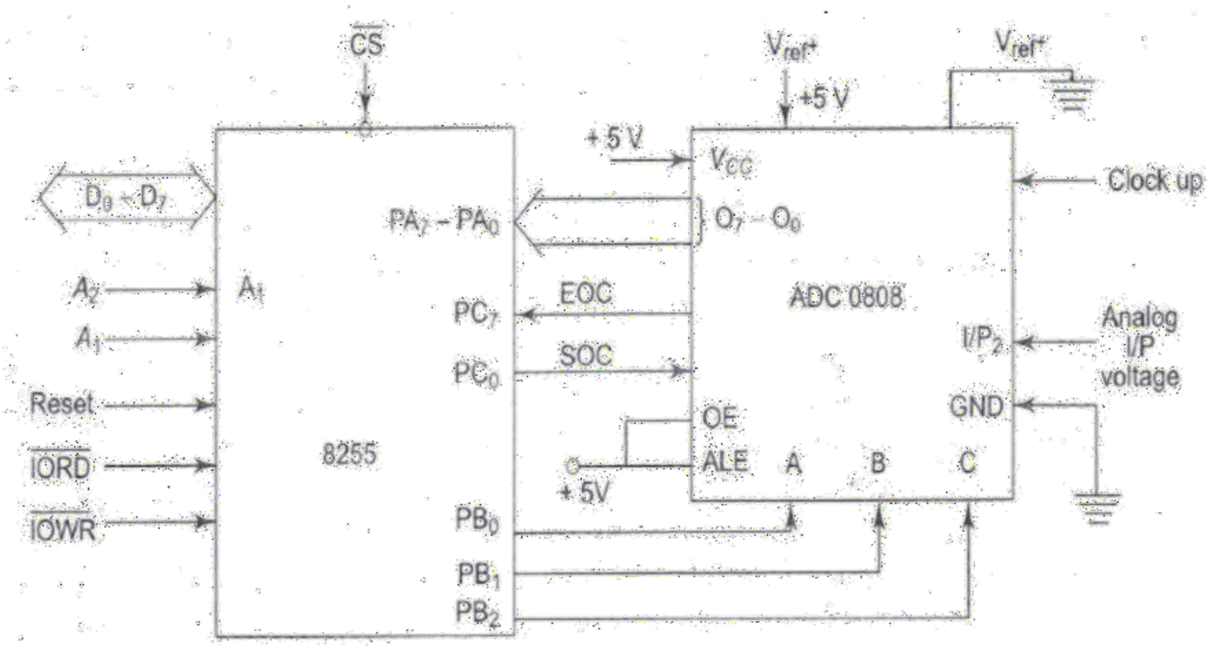


Fig.3 Timing Diagram Of ADC 0808.



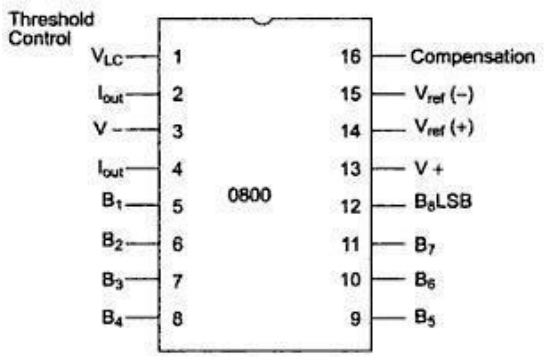
Interfacing ADC0808 with 8086

Interfacing Digital To Analog Converters:

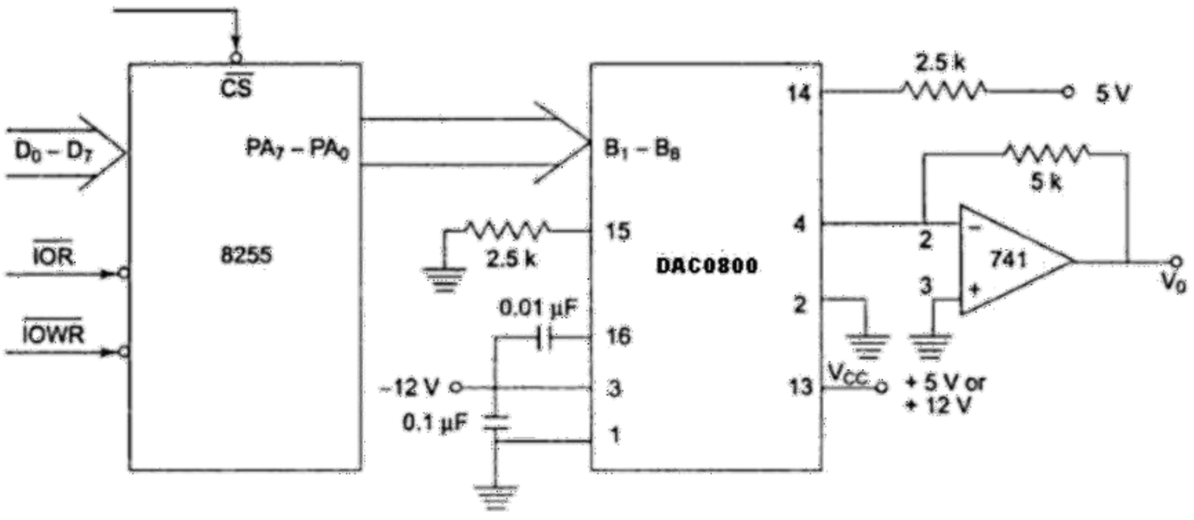
The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.

DAC0800 8-bit Digital to Analog Converter

- The DAC 0800 is a monolithic 8-bit DAC manufactured by National Semiconductor.
- It has settling time around 100ms and can operate on a range of power supply voltages i.e. from 4.5V to +18V.
- Usually the supply V+ is 5V or +12V.
- The V-pin can be kept at a minimum of -12V.



Pin Diagram of DAC 0800



Interfacing DAC0800 with 8086

<u>Keyboard Interfacing</u>

- In most keyboards, the key switches are connected in a matrix of Rows and Columns.
- Getting meaningful data from a keyboard requires three major tasks:

1. D e t e c t a k e y p r e s s
2. D e b o u n c e t h e k e y p r e s s .
3. Encode the keypress (produce a standard code for the pressed key).

Logic „0" is read by the microprocessor when the key is pressed.

Key Debounce:

Whenever a mechanical push-bottom is pressed or released once,the mechanical components of the key do not change the positionsmoothly; rather it generates a transient response. These may be interpreted as the multiple pressures and responded accordingly.

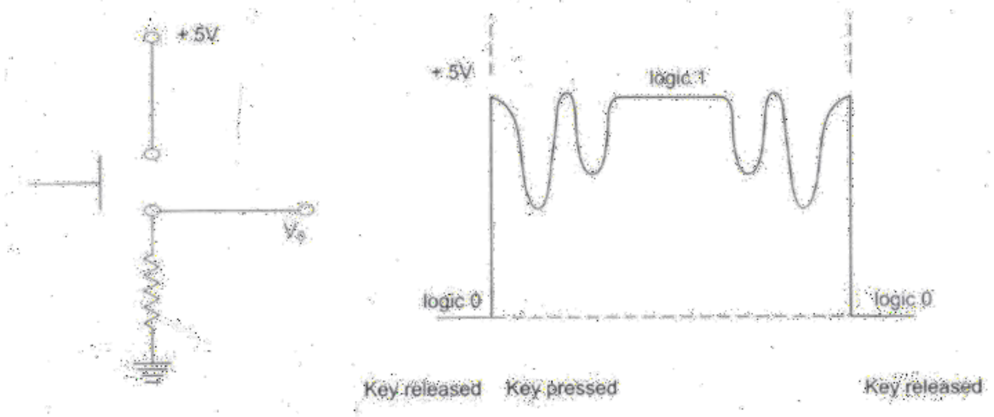


Fig. 5.23 *A Mechanical Key and Its Response*

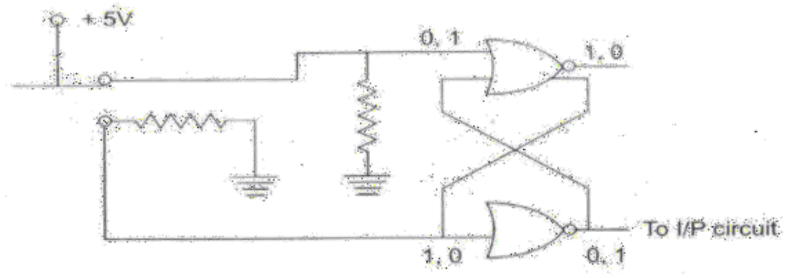
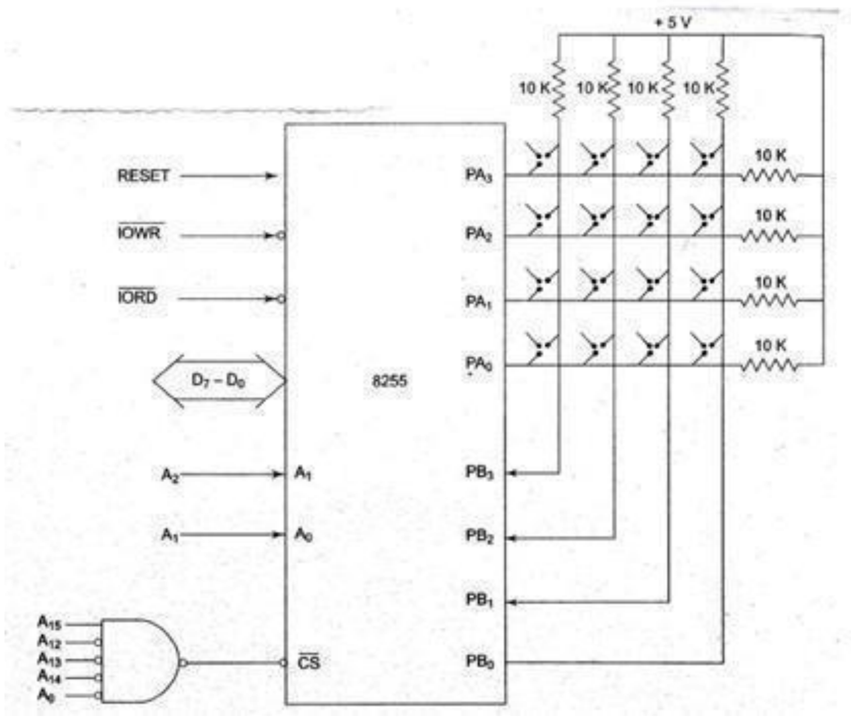


Fig. 5.24 *Hardware Debouncing Circuit*

- The rows of the matrix are connected to four output Port lines, &columns are connected to four input Port lines.
- When no keys are pressed, the column lines are held high by the pull-up resistors connected to +5v.
- Pressing a key connects a row & a column.
- To detect if any key is pressed is to output 0‟s to all rows & then check columns to see it a pressed key has connected a low (zero) to a column.
- Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns i.e indicating a key press.
- A simple 20/10 msec delay is executed to debounce task.
- After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed & the initial detection was caused by a noise pulse.
- To avoid this problem, two schemes are suggested:
  1. Use of Bistablemultivibrator at the output of the key to debounce it.
  2. The microprocessor has to wait for the transient period (at least for 10 ms), so that the transient response settles down and reaches a steady state.

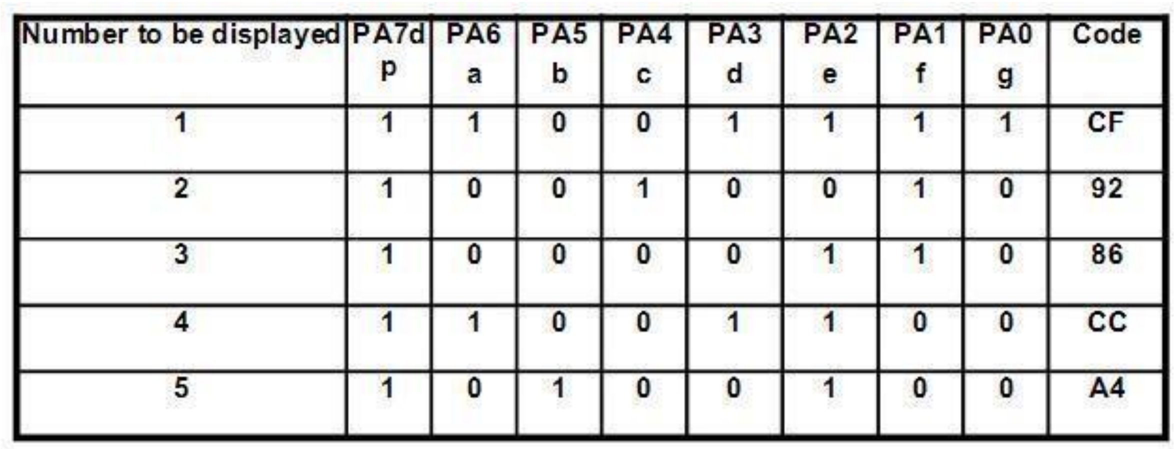
- If any of the columns are low now, then the assumption is made that it was a valid key press.

  - The final task is to determine the row & column of the pressed key &convert this information to Hex-code for the pressed key.
- The 4-bit code from I/P port & the 4-bit code from O/P port (row &column) are converted to Hex-code.

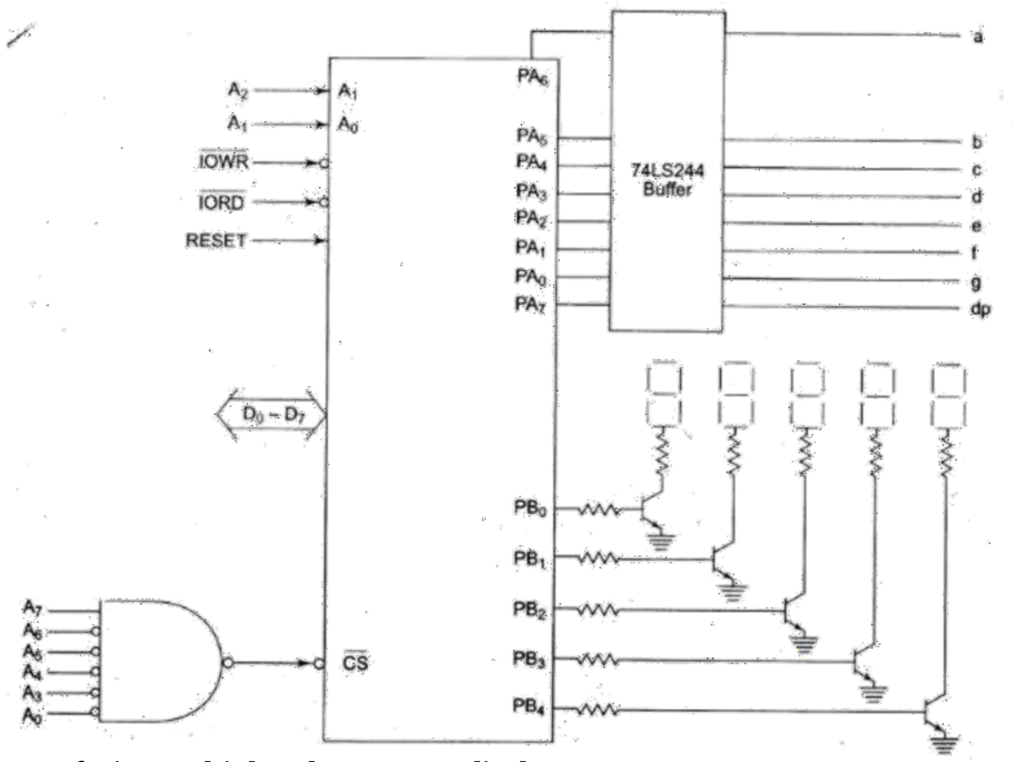


Interfacing 4x4 keyboard

<u>Display Interface</u>



Number to be displayed	PA7 dp	PA6 a	PA5 b	PA4 c	PA3 d	PA2 e	PA1 f	PA0 g	Code
1	1	1	0	0	1	1	1	1	CF
2	1	0	0	1	0	0	1	0	92
3	1	0	0	0	0	1	1	0	86
4	1	1	0	0	1	1	0	0	CC
5	1	0	1	0	0	1	0	0	A4



Interfacing multiplexed 7-segment display

Interfacing with Advanced devices

MEMORY AND I/O INTERFACING

(Ref: Interfacing through Microprocessors by K. Subba Rao, Hi-tech publishers, P. 163-166)

I/O Interface

Any application of a microprocessor system requires the transfer of data between microprocessor and external environment and also with in the microprocessor. This is known as Input/Output. There are three different ways that the data transfer can take place. They are

(1) Program controlled I/O
(2) Interrupt Program Controlled I/O
(3) Hardware controlled I/O

In program controlled I/O data transfer scheme the transfer of data is completely under the control of the microprocessor program. In this case an I/O operation takes place only when an I/O transfer instruction is executed.

In an interrupt program controlled I/O an external device indicates directly to the microprocessor its readiness to transfer data by a signal at an interrupt input of the microprocessor. When microprocessor receives this signal the control is transferred to ISS (Interrupt service subroutine) which performs the data transfer.

Hardware controlled I/O is also known as direct memory access DMA. In this case the data transfer takes place directly between an I/O device and memory but not through microprocessors. Microprocessor only initializes the process of data transfer by indicating the starting address and the number of words to be transferred.

The instruction .set of any microprocessor contains instructions that transfer information to an I/O device and to read information from an I/O device. In 8086 we have IN, OUT instructions for this purpose. OUT instruction transfers information to an I/O device where as IN instruction is used to read information from an I/O device. Both the instructions perform the data transfer using accumulator AL or AX. The I/O address is stored in register DX.

The port number is specified along with IN or OUT instruction. The external I/O interface decodes to find the address of the I/O device. The 8 bit fixed port number appears on address bus A_0 - A_7 with A_8 - A_{15} all zeros. The address connections above A_{15} are
undefined for an I/O instruction. The 16 bit variable port number appears on address connections A_0 - A_{15}. The above notation indicates that first 256 I/O port addresses 00 to FF are accessed by both the fixed and variable I/O instructions. The I/O addresses from 0000 to FFFF are accessed by the variable I/O address.

I/O devices can be interfaced to the microprocessors using two methods. They are I/O mapped I/O and memory mapped I/O. The I/O mapped I/O is also known as isolated I/O or direct I/O. In I/O mapped I/O the IN and OUT instructions transfer data between the accumulator or memory and I/O device. In memory mapped I/O the instruction that refers memory can perform the data transfer.

I/O mapped I/O is the most commonly used I/O transfer technique. In this method I/O locations are placed separately from memory. The addresses for isolated I/O devices are separate from memory. Using this method user can use the entire memory. This method allows data transfer only by using instructions IN, OUT. The pins M/ $\overline{IO}$ and W/R are used to indicate I/O read or an I/O write operations. The signals on these lines indicate that the address on the address bus is for I/O devices.

Memory mapped I/O does not use the IN, OUT instruction it uses only the instruction that transfers data between microprocessor and memory. A memory mapped I/O device is treated as memory location. The disadvantage in this system is the overall memory is reduced. The advantage of this system is that any memory transfer instruction can be used for data transfer and control signals like I/O read and I/O write are not necessary which simplify the hardware.

Memory interfacing

Memory is an integral part of a microcomputer system. There are two main types of memory.

(i) **Read only memory (ROM):** As the name indicates this memory is available only for reading purpose. The various types available under this category are PROM, EPROM, EEPROM which contain system software and permanent system data.

(ii) **Random Access memory (RAM):** This is also known as Read Write Memory. It is a volatile memory. RAM contains temporary data and software programs generally for different applications.

While executing particular task it is necessary to access memory to get instruction codes and data stored in memory. Microprocessor initiates the necessary signals when read or write operation is to be performed. Memory device also requires some signals to perform read and write operations using various registers. To do the above job it is necessary to have a device and a circuit, which performs this task is known as interfacing device and as this is involved with memory it-is known as memory interfacing device. The basic concepts of memory interfacing involve three different tasks. The microprocessor should be able to read from or write into the specified register. To do this it must be able to select the required chip, identify the required register and it must enable the appropriate buffers.

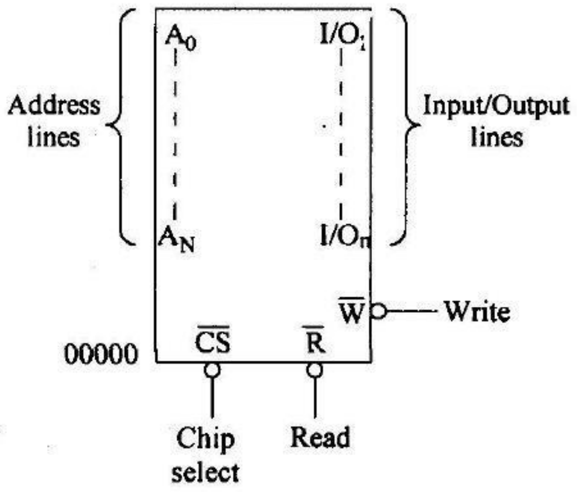


Fig. 3.13 Simple memory device

Any memory device must contain address lines and Input, output lines, selection input, control input to perform read or write operation. All memory devices have address inputs that select memory location with in the memory device. These lines are labeled as A_O .... A_N. The number of address lines indicates the total memory capacity of the memory device. A 1K memory requires 10 address lines A_0-A_9. Similarly a 1MB requires 20 lines A_0-A_{19} (in the case of 8086). The memory devices may have separate I/O lines or a common set of bidirectional I/O lines. Using these lines data can be transferred in either direction. Whenever output buffer is activated the operation is read whenever input buffers are activated the operation is write. These lines are labelled as I/O,......... I/O_n or D_O.................... D_n. The size of a memory location is dependent upon the number of data bits. If the number of data lines are eight D_0 - D_7 then 8 bits or 1 byte of data can be stored in each location. Similarly if numbers of data bits are 16 (D_0 - D_{15}) then the memory size is 2 bytes. For example 2K x 8 indicates there are 2048 memory locations and each memory location can store 8 bits of data.

Memory devices may contain one or more inputs which are used to select the memory device or to enable the memory device. This pin is denoted by CS (Chip select) or $\overline{CE}$ (Chip enable). When this pin is at logic '0' then only the memory device performs a read or a write operation. If this pin is at logic '1' the memory chip is disabled. If there are more than one $\overline{CS}$ input then all these pins must be activated to perform read or write operation.

All memory devices will have one or more control inputs. When ROM is used we find $\overline{OE}$ output enable pin which allows data to flow out of the output data pins. To perform this task both CS and $\overline{OE}$ must be active. A RAM contains one or two control inputs. They are R / W or $\overline{RD}$ and $\overline{WR}$. If there is only one input R/ W then it performs read operation when R/ $\overline{W}$ pin is at logic 1. If it is at logic 0 it performs write operation. Note that this is possible only when $\overline{CS}$ is also active.

Memory Interface using RAMS, EPROMS and EEPROMS

(Ref: Advanced Microprocessors and Peripherals by A.K. Ray & K.M. Bhurchandi, McGraw-Hill, 2nd Edition.P.158- 164)

Semiconductor Memory Interfacing:

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory).

Static RAM Interfacing:

The semiconductor RAMs are of broadly two types-static RAM and dynamic RAM. The semiconductor memories are organised as two dimensional arrays of memory locations. For example, 4K x 8 or 4K byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time. Obviously, for addressing 4K bytes of memory, twelve address lines are required. In general, to address a memory location out of N memory locations , we will require at least *n* bits of address, i.e. *n* address lines where $n = \text{Log}_2 N$. Thus if the microprocessor has *n* address lines, then it is able to address at the most N locations of memory, where $2^n = N$. However, if out of *N* locations only *P* memory locations are to be interfaced, then the least significant *p* address lines out of the available *n* lines can be directly connected from the microprocessor to the memory chip while the remaining *(n-p)* higher order address lines may be used for address decoding (as inputs to the chip selection logic). The memory address depends upon the hardware circuit used for decoding the chip select (CS). The output of the decoding circuit is connected with the $\overline{CS}$ pin of the memory chip. The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called 'odd address memory bank' and the lower 8-bit bank is called 'even address memory bank'.

2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory $\overline{RD}$ and $\overline{WR}$ inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.

3. The remaining address lines of the microprocessor, $\overline{BHE}$ and A_0 are used for decoding the required chip select signals for the odd and even memory banks. $\overline{CS}$ of memory is derived from the O/P of the decoding circuit.

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should be no windows in the map. A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred, and minimum hardware should be used for decoding. In a number of cases, linear decoding may be used to minimise the required hardware.Let us now consider a few example problems on memory interfacing with 8086.

Problem 5.1

Interface two 4K × 8 EPROMS and two 4K × 8 RAM chips with 8086. Select suitable maps.

Solution We know that, after reset, the IP and CS are initialised to form address FFFF0H. Hence, this address must lie in the EPROM. The address of RAM may be selected any where in the 1MB address space of 8086, but we will select the RAM address such that the address map of the system is continuous, as shown in Table 5.1.

Table 5.1 *Memory Map for Problem 5.1*

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							EPROM						8K × 8							
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
							RAM					8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Total 8K bytes of EPROM need 13 address lines $A_0 - A_{12}$ (since $2^{13} = 8K$). Address lines $A_{13} - A_{19}$ are used for decoding to generate the chip select. The $\overline{BHE}$ signal goes low when a transfer is at odd address or higher byte of data is to be accessed. Let us assume that the latched address, $\overline{BHE}$ and demultiplexed data lines are readily available for interfacing. Figure 5.1 shows the interfacing diagram for the memory system.

The memory system in this example contains in total four 4K × 8 memory chips.

The two 4K × 8 chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If A_0 is 0, i.e. the address is even and is in RAM, then the lower RAM chip is selected indicating 8-bit transfer at an even address. If A_0 is 1, i.e. the address is odd and is in RAM, the $\overline{BHE}$ goes low, the upper RAM chip is selected, further indicating that the 8-bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A_0 and $\overline{BHE}$ both are 0, both the RAM or ROM chips are selected, i.e. the data transfer is of 16 bits. The selection of chips here takes place as shown in Table 5.2.

Table 5.2 *Memory Chip Selection for Problem 5.1*

Decoder I/P → *Address/$\overline{BHE}$ →*	A_2 A_{13}	A_1 A_0	A_0 $\overline{BHE}$	*Selection/ Comment*
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM

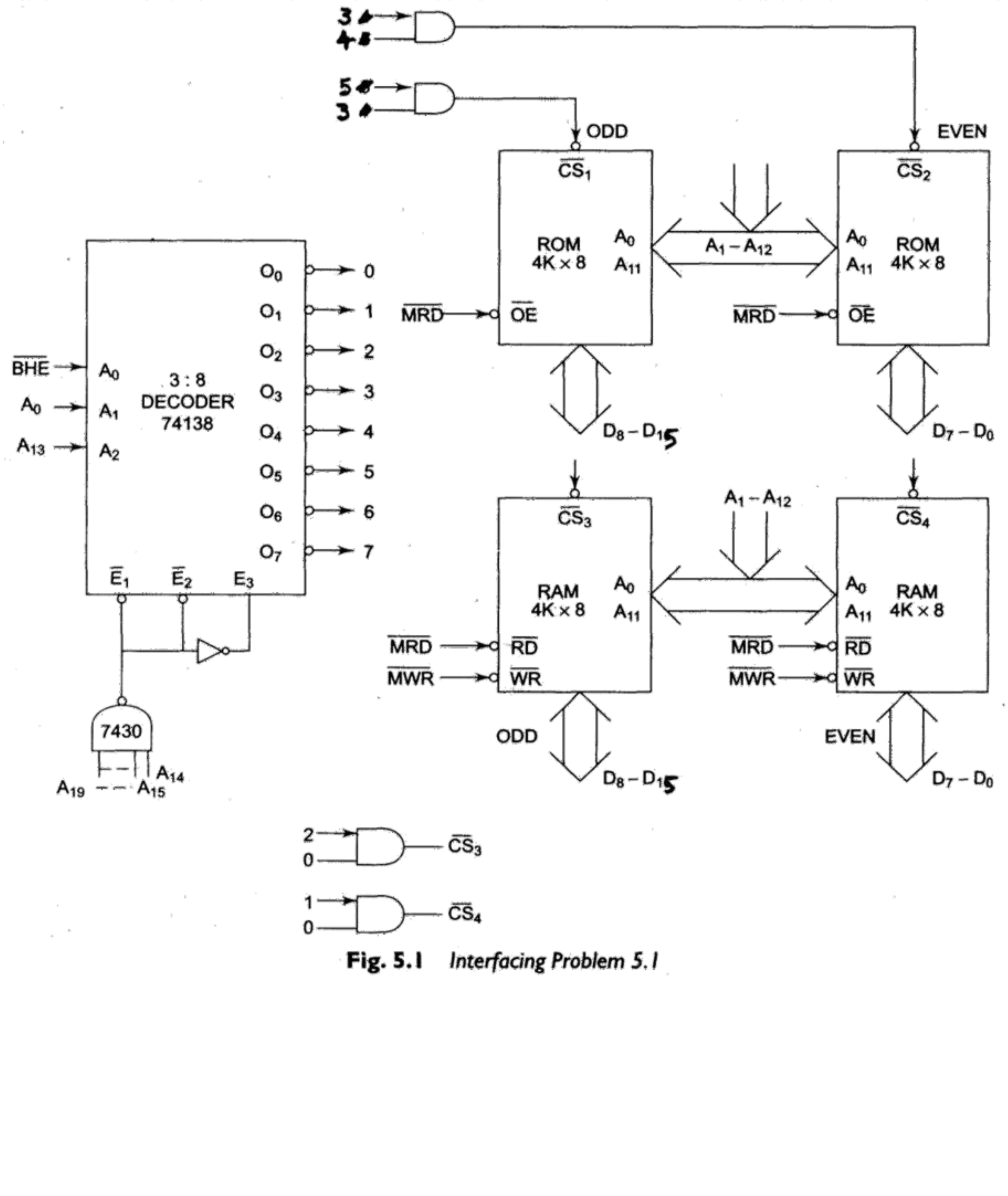


Fig. 5.1 *Interfacing Problem 5.1*

Problem 5.2

Design an interface between 8086 CPU and two chips of 16K × 8 EPROM and two chips of 32K × 8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000H.

Solution: The last address in the map of 8086 is FFFFFH. After resetting, the processor starts from FFFF0H. Hence this address must lie in the address range of EPROM. Figure 5.2 shows the interfacing diagram, and Table 5.3 shows complete map of the system.

Table 5.3 *Address Map for Problem 5.2*

Addresses	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							32KB			EPROM										
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						64KB RAM														
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFFH) and the first EPROM address F8000H). Hence the logic is implemented using logic gates, as shown in Fig. 5.2.

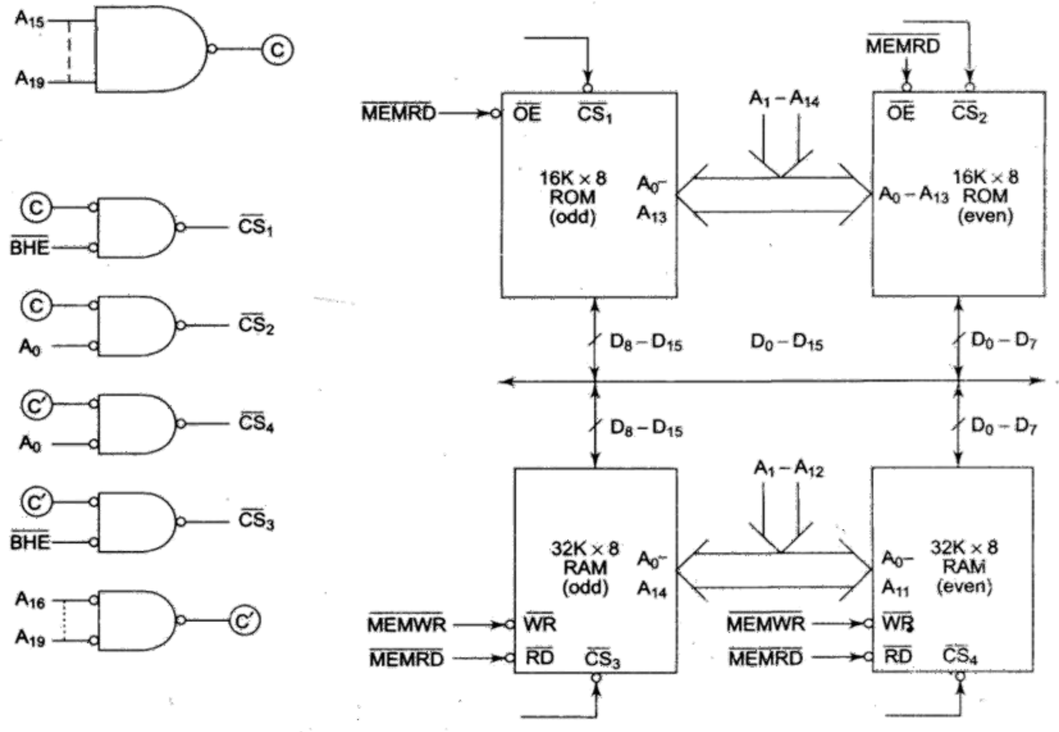


Fig. 5.2 *Interfacing Problem 5.2*

Problem 5.3

It is required to interface two chips of 32K × 8 ROM and four chips of 32K × 8 RAM with 8086, according to the following map.

```
ROM 1 and 2 F0000H - FFFFFH, RAM 1 and 2 D0000H - DFFFFH
RAM 3 and 4 E0000H - EFFFFH
```

Show the implementation of this memory system.

Solution Let us write the memory map of the system as shown in Table 5.6.

The implementation of the above map is shown in Fig. 5.3 using the same technique as in Problem 5.1 and Problem 5.2. All the address, data and control signals are assumed to be readily available.

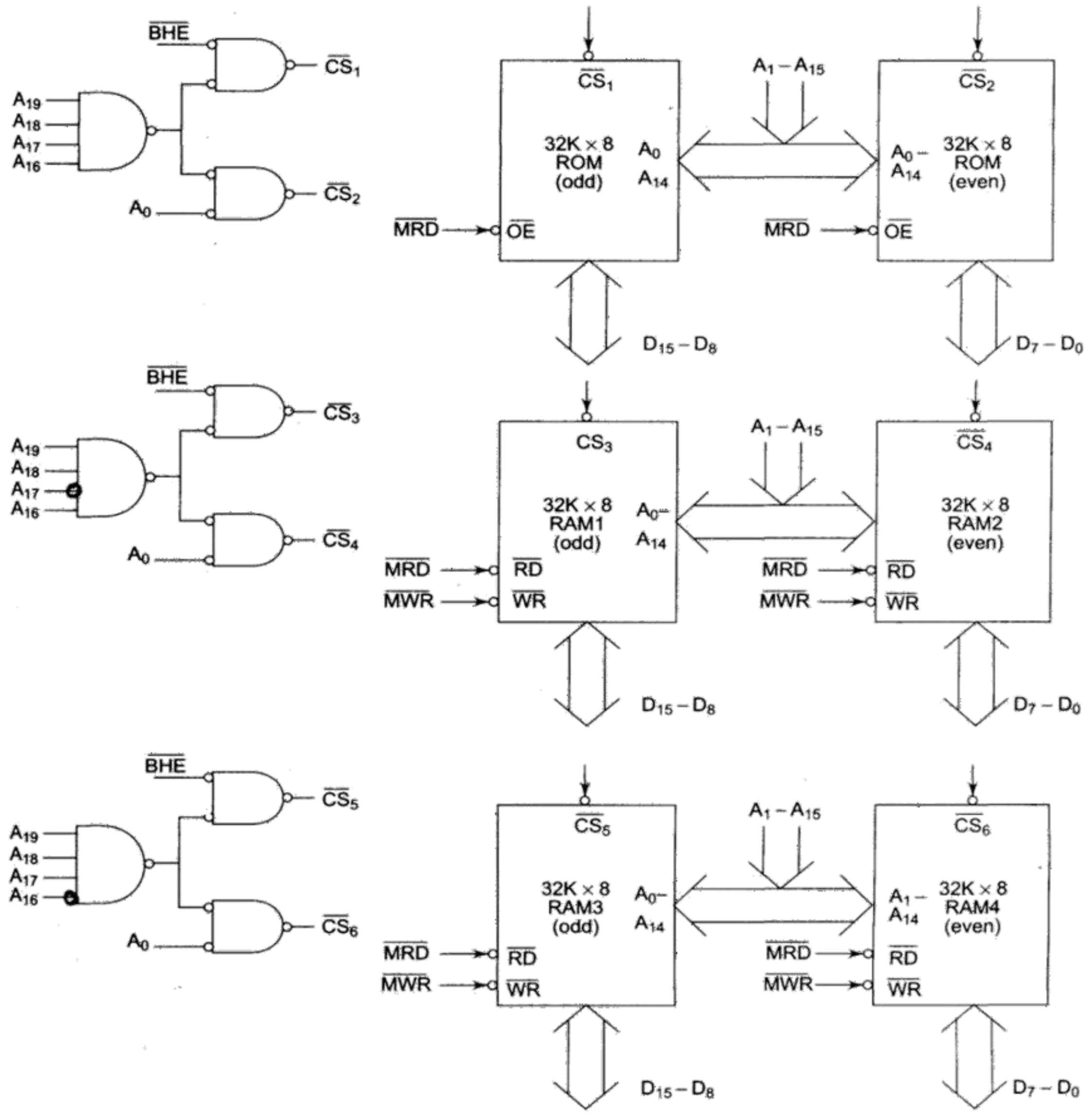
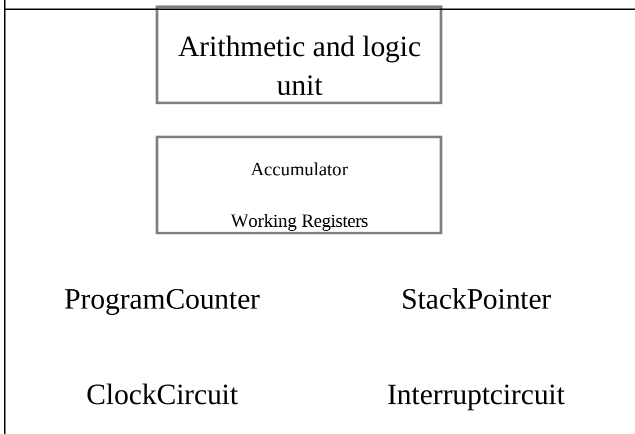
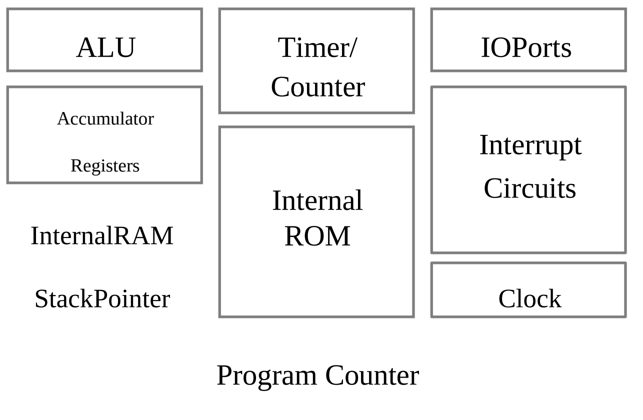


Fig. 5.3 *Interfacing Problem 5.3*

UNIT –IV

Introdution to Microcontrollers

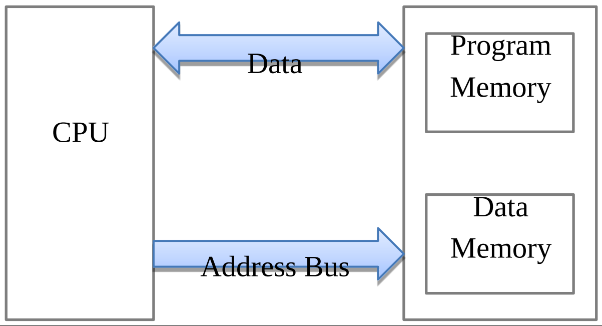
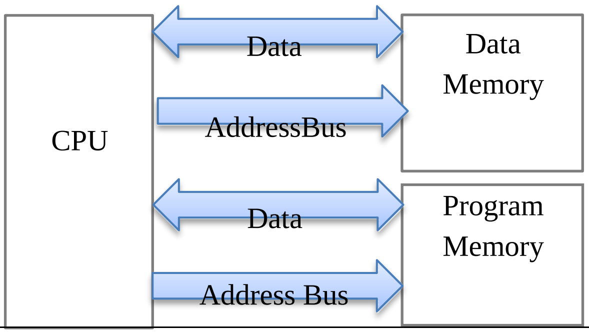
Microprocessor	***Microcontroller***
 Arithmetic and logic unit Accumulator Working Registers ProgramCounter StackPointer ClockCircuit Interruptcircuit	 ALU Timer/ Counter IOPorts Accumulator Registers Interrupt Circuits Internal ROM InternalRAM StackPointer Clock Program Counter
Block diagram of microprocessor	*Block diagram of microcontroller*
Microprocessor contains ALU, General purpose registers, stack pointer, program counter, clock timing circuit, interrupt circuit	Microcontroller contains the circuitry of microprocessor, and in addition it has built in ROM, RAM, I/O Devices, Timers/Counters etc.
It has many instructions to move data between memory and CPU	It has few instructions to move data between memory and CPU
Few bit handling instruction	It has many bit handling instructions
Less number of pins are multifunctional	More number of pins are multifunctional
Single memory map for data and code (program)	Separate memory map for data and code (program)
Access time for memory and IO are more	Less access time for built in memory and IO.
Microprocessor based system requires additionalhardware	It requires less additional hardwares
More flexible in the design point of view	Less flexible since the additional circuits which is residing inside the microcontroller is fixed for a particular microcontroller
Large number of instructions with flexible addressing modes	Limited number of instructions with few addressingmodes

RISC AND CISC CPU ARCHITECTURES

Microcontrollers with small instruction set are called reduced instruction set computer (RISC) machines and those with complex instruction set are called complex instruction set computer (CISC). Intel 8051 is an example of CISC machine whereas microchip PIC 18F87X is an example of RISC machine.

RISC	**CISC**
Instruction takes one or two cycles	Instruction takes multiple cycles
Only load/store instructions are used to access memory	In additions to load and store instructions, memory access is possible with other instructions also.
Instructions executed by hardware	Instructions executed by the micro program
Fixed format instruction	Variable format instructions
Few addressing modes	Many addressing modes
Few instructions	Complex instruction set
Most of the have multiple register banks	Single register bank
Highly pipelined	Less pipelined
Complexity is in the compiler	Complexity in the microprogram

HARVARD & VON- NEUMANN CPU ARCHITECTURE

Von-Neumann (Princeton architecture)	**Harvard architecture**
	 CPU Data AddressBus Data Address Bus Data Memory Program Memory
Von-Neumann (Princeton architecture)	**Harvard architecture**
It uses single memory space for both instructions anddata.	It has separate program memory and data memory
It is not possible to fetch instruction code and data	Instruction code and data can be Fetched simultaneously
Execution of instruction takes more machine cycle	Execution of instruction takes less machine cycle
Uses CISC architecture	Uses RISC architecture
Instruction pre-fetching is a main feature	Instruction parallelism is a main feature
Also known as control flow or control driven computers	Also known as data flow or data Driven computers
Simplifies the chip design because of single memory space	Chip design is complex due to separate memory space
Eg. 8085, 8086, MC6800	Eg. General purpose microcontrollers, special DSP chips etc.

COMPUTER SOFTWARE

A set of instructions written in a specific sequence for the computer to solve a specific task is called a program and software is a collection of such programs.

The program stored in the computer memory in the form of binary numbers is called machine instructions. The *machine language* program is called *object code*.

An *assembly language* is a mnemonic representation of machine language. Machine language and assembly language are low level languages and are processor specific.

The assembly language program the programmer enters is called *source code*. The source code (assembly language) is translated to object code (machine language) using *assembler*.

Programs can be written in *high level languages* such as C, C++ etc. High level language will be converted to machine language using *compiler or interpreter*. Compiler reads the entire program and translate into the object code and then it is executed by the processor. Interpreter takes one statement of the high level language as input and translate it into object code and then executes.

CLASSIFICATION OF MICROCONTROLLER

Microcontrollers are divided by their bits, memory architecture, memory device and instruction set

CLASSIFICATION ACCORDING TO BITS

8 bit- In this if the internal bus is 8 bit then the ALU performs arithmetic and logic operations. Examples are 8051/8031/ pic1x.

16 bit- this microcontroller performance is greater compared to 8-bit. 16 bit can use 16 bit for its operations were as in 8-bit it is only 8-bit examples are 8051 extended, pic2x

32-bit - it uses 32 bit to perform its operations these are used in medical devices and in control systems PIC3X is an example

CLASSIFICATION ACCORDING TO MEMORY DEVICES

Embedded memory microcontroller – This is a type of microcontroller in which all the functional blocks are available in a chip data memory I/O ports

External memory microcontroller – in this microcontroller all functional blocks are not available 8031 doesn't have any program memory

CLASSIFICATION ACCORDING TO INSTRUCTION SET

CISC – complex instruction set computer it allows the user to use one instruction to do the functions of many simple instructions

RISC- reduced instruction set computer in this instruction set is reduced each instruction can be operated on any register or in any addressing mode

CLASSIFICATION ACCORDING TO MEMORY ARCHITECTURE

Harvard memory architecture- in this microcontroller the program and data memory doesn't have a similar memory address space

Princeton memory architecture- in this the program and data memory have similar address space

TYPES OF MICROCONTROLLER

8051 microcontroller

It is designed by Intel in 1981 and it is an 8 bit microcontroller it has 40 pins dual inline package 128 bytes of RAM 4k byte of ROM in 8051 there are 2 busses one for programming and other for data programming in microcontroller is complicated basically we write a program in C language and then it is converted to machine language understood by microcontroller two types of memory is present the program memory and data memory program memory stores the data being executed while data memory stores the result temporarily

Renesas microcontroller

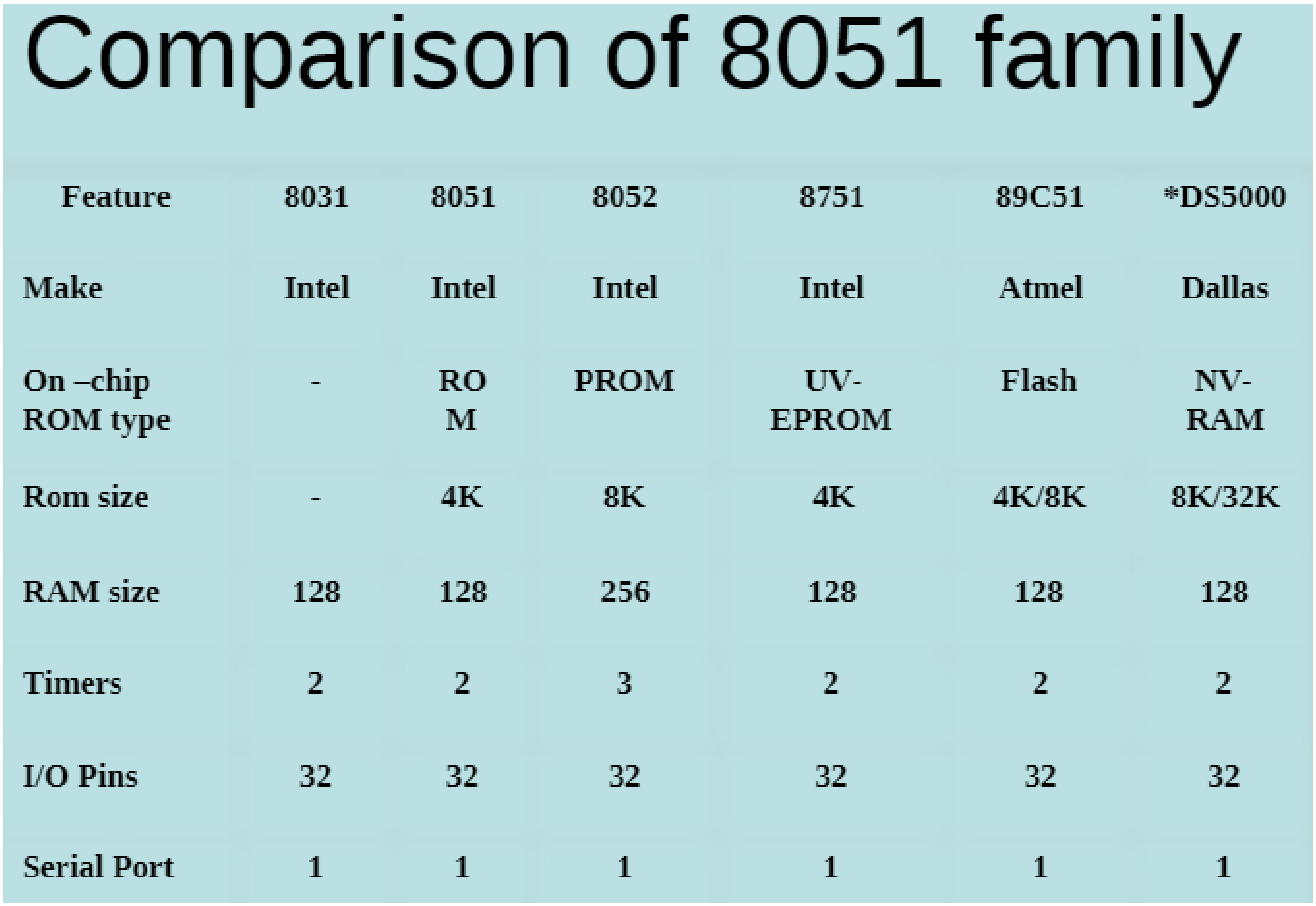
It is in the automotive microcontroller family that offers high-performance features with low power consumption this microcontroller offers high security and embedded safety character for the automotive applications RX microcontroller is an example which has 32 bit

AVR microcontrollers

The AVR microcontrollers have modified Harvard RISC architecture with separate memories for data and program and speed of AVR is really high when compared to PIC and 8051

APPLICATIONS

- Industrial automation
- Communication application
- Motor control applications
- Test and measurement
- Medical applications
- Automobiles
- Cameras
- Security alarms
- Mobile phones



Comparison of 8051 family

Feature	8031	8051	8052	8751	89C51	*DS5000
Make	Intel	Intel	Intel	Intel	Atmel	Dallas
On –chip ROM type	-	RO M	PROM	UV-EPROM	Flash	NV-RAM
Rom size	-	4K	8K	4K	4K/8K	8K/32K
RAM size	128	128	256	128	128	128
Timers	2	2	3	2	2	2
I/O Pins	32	32	32	32	32	32
Serial Port	1	1	1	1	1	1

THE 8051 ARCHITECTURE

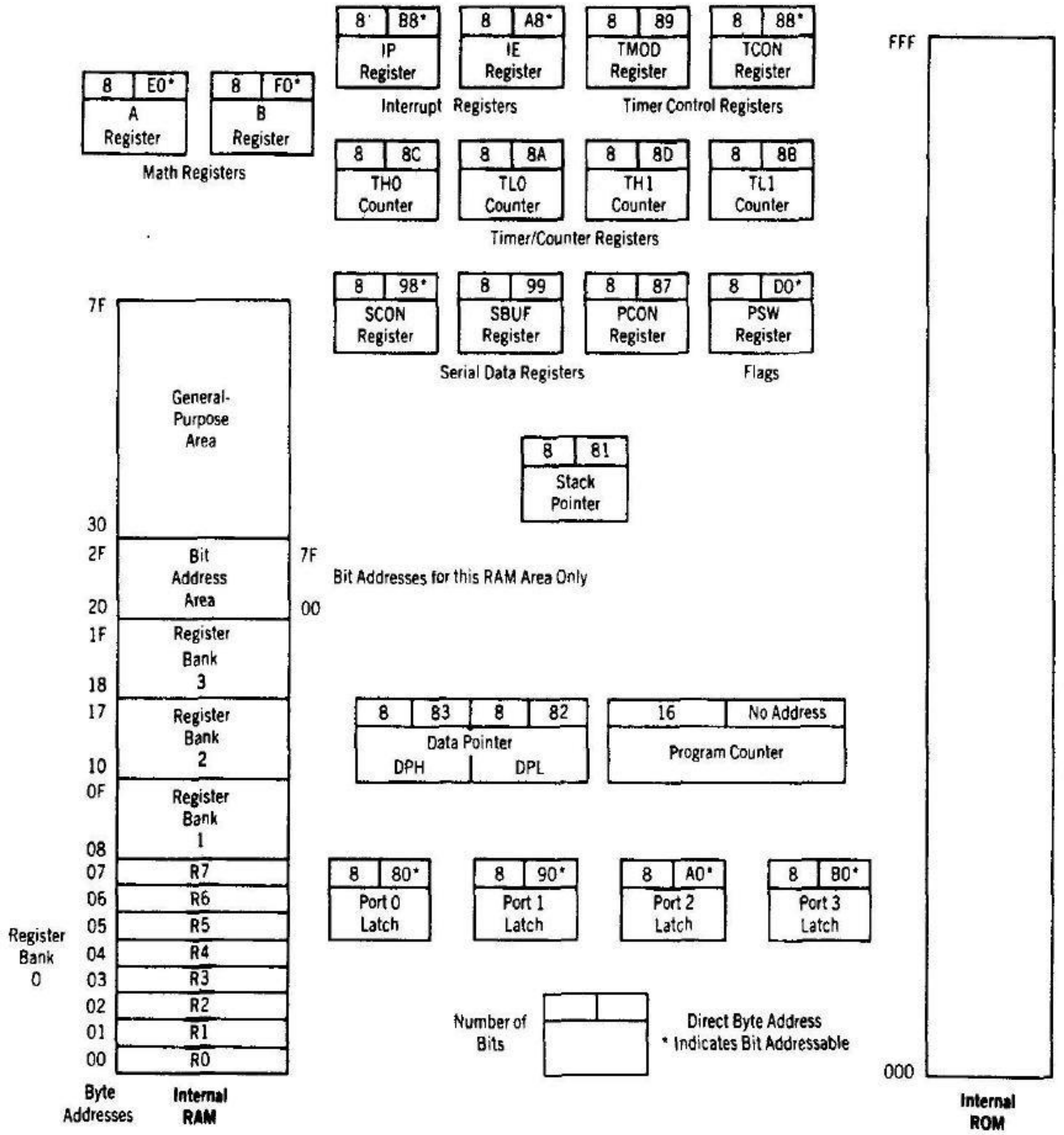
Introduction

Salient features of 8051 microcontroller are given below.

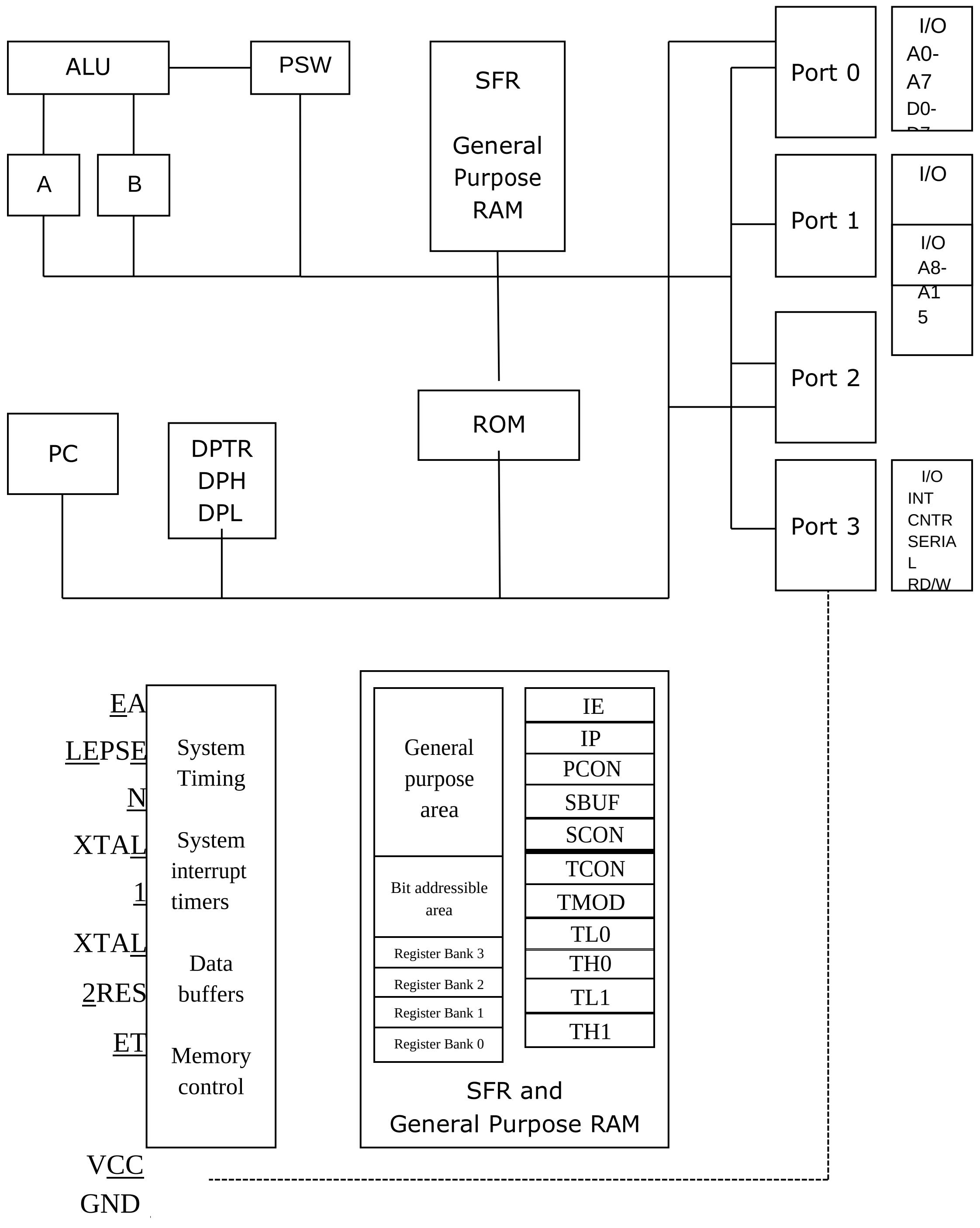
- Eight bitCPU
- On chip clockoscillator
- 4Kbytes of internal program memory (code memory)*[ROM]*
- 128 bytes of internal data memory*[RAM]*
- 64 Kbytes of external program memory addressspace.
- 64 Kbytes of external data memory addressspace.
- 32 bi directional I/O lines (can be used as four 8 bit ports or 32 individually addressable I/O lines)
- Two 16 Bit Timer/Counter :T0,T1
- Full Duplex serial datareceiver/transmitter
- Four Register banks with 8 registers in eachbank.
- Sixteen bit Program counter (PC) and a data pointer(DPTR)
- 8 Bit Program Status Word(PSW)
- 8 Bit StackPointer
- Five vector interrupt structure (RESET not considered as aninterrupt.)
- 8051 CPU consists of 8 bit ALU with associated registers like accumulator ‘A’ , B register,PSW, SP, 16 bit program counter, stackpointer.

- ALU can perform arithmetic and logic functions on 8 bitvariables.
- 8051 has 128 bytes of internal RAM which is dividedinto
  - Working registers [00 –1F]
  - Bit addressable memory area [20 –2F]
  - General purpose memory area (Scratch pad memory)[30-7F]

FIGURE 2.1b 8051 Programming Model



The 8051 architecture.



- 8051 has 4 K Bytes of internal ROM. The address space is from 0000 to 0FFFh. If the program size is more than 4 K Bytes 8051 will fetch the code automatically from external memory.
- Accumulator is an 8 bit register widely used for all arithmetic and logical operations.

Accumulator is also used to transfer data between external memory. B register is used along with Accumulator for multiplication and division. A and B registers together is also called MATHregisters.

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

CY	AC	F0	RS1	RS0	OV	-	P

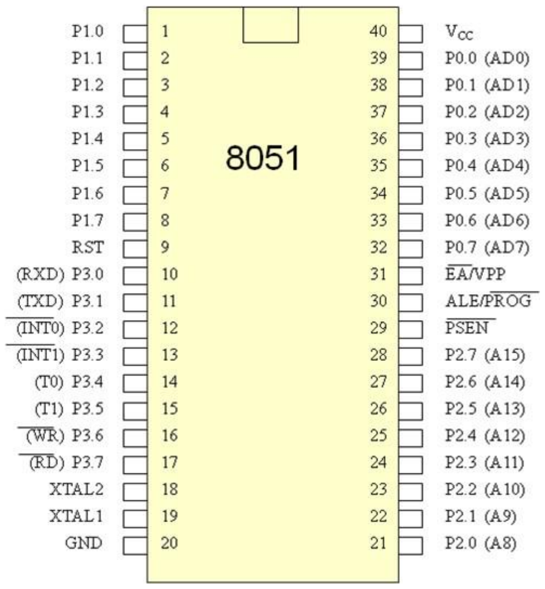
CY	-	carryflag
AC	-	auxiliary carryflag
F0	-	available to the user for general

purpose RS1,RS0- register bank selectbits

OV	-	overflow
P	-	parity

- Stack Pointer (SP) – it contains the address of the data item on the top of the stack. Stack may reside anywhere on the internal RAM. On reset, SP is initialized to 07 so that the default stack will start from address 08onwards.
- Data Pointer (DPTR) – DPH (Data pointer higher byte), DPL (Data pointer lower byte). This is a 16 bit register which is used to furnish address information for internal and externalprogram memory and for external datamemory.
- Program Counter (PC) – 16 bit PC contains the address of next instruction to be executed. On reset PC will set to 0000. After fetching every instruction PC will increment byone.

PIN DIAGRAM



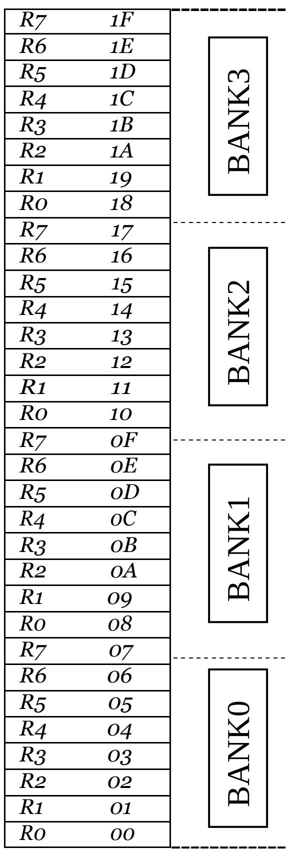
Pinout Description

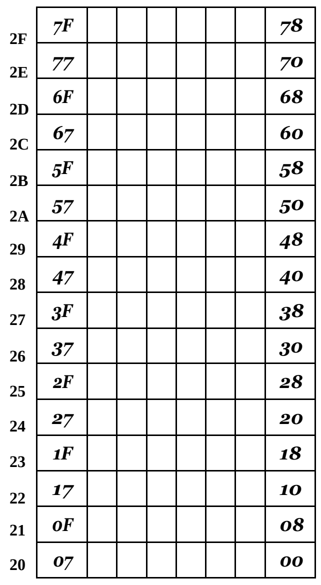
Pins 1-8	**PORT 1**. Each of these pins can be configured as an input or an output.
Pin 9	**RESET**. A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.
Pins10-17	**PORT 3**. Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions

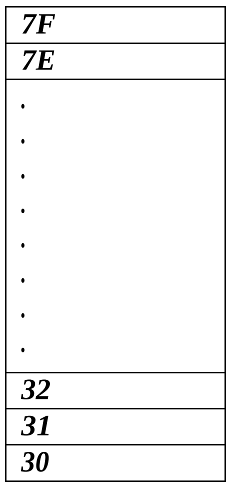
Pin 10	**RXD.** Serial asynchronous communication input or Serial synchronous communication output.
Pin 11	**TXD.** Serial asynchronous communication output or Serial synchronous communication clockoutput.
Pin 12	**INT0.**External Interrupt 0 input
Pin 13	**INT1.** External Interrupt 1 input
Pin 14	**T0.** Counter 0 clock input
Pin 15	**T1.** Counter 1 clock input
Pin 16	**WR**. Write to external (additional) RAM
Pin 17	**RD.** Read from external RAM
Pin 18, 19	**XTAL2, XTAL1.** Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins.
Pin 20	**GND**. Ground.
Pin 21-28	**Port 2**. If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available asinputs/outputs.
Pin 29	**PSEN.** If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.
Pin 30	**ALE.** Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external latch latches the state of P0 and uses it as a memory chip address. Immediately after that, the ALE pin is returned its previous logic state and P0 is now used as a Data Bus.
Pin 31	**EA**. By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).
Pin 32-39	**PORT 0**. Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).
Pin 40	**VCC**. +5V power supply.

MEMORYORGANIZATION

Internal RAM organization







General purpose memory

Bit addressable memory

Working Registers

Register Banks: 00h to 1Fh. The 8051 uses 8 general-purpose registers R0 through R7 (R0, R1,R2, R3, R4, R5, R6, and R7). There are four such register banks. Selection of register bank can be done through RS1,RS0 bits of PSW. On reset, the default Register Bank 0 will beselected.

Bit Addressable RAM: 20h to 2Fh . The 8051 supports a special feature which allows access to bit variables. This is where individual memory bits in Internal RAM can be set or cleared. In all there are 128 bits numbered 00h to 7Fh. Being bit variables any one variable can have a value 0 or 1. A bit variable can be set with a command such as SETB and cleared with a command such as CLR. Example instructions are:

SETB25h;setsthebit25h(becomes1)
CLR25h;clearsbit25h(becomes0)
Note, bit 25h is actually bit 5 of Internal RAM location 24h.
The Bit Addressable area of the RAM is just 16 bytes of Internal RAM located between 20h and 2Fh.

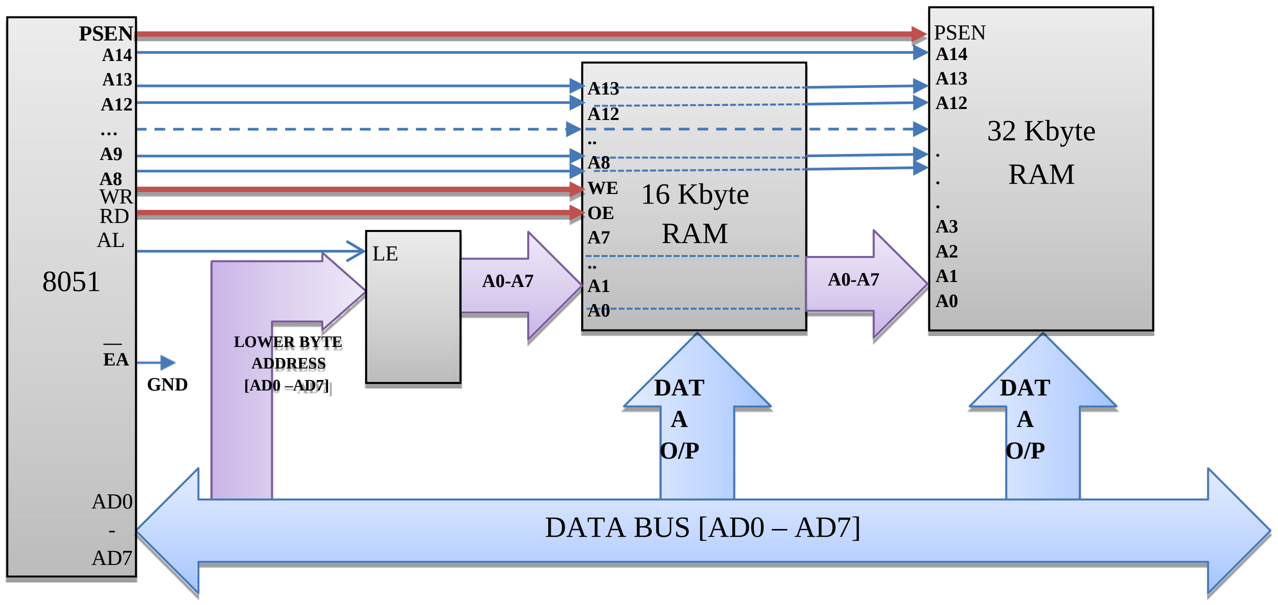
General Purpose RAM: 30h to 7Fh. Even if 80 bytes of Internal RAM memory are available for general-purpose data storage, user should take care while using the memory location from 00 -2Fh since these locations are also the default register space, stack space, and bit addressable space. It is a good practice to use general purpose memory from 30 – 7Fh. The general purpose RAM can be accessed using direct or indirect addressingmodes.

EXTERNAL MEMORYINTERFACING

Eg. Interfacing of 16 K Byte of RAM and 32 K Byte of EPROM to 8051

Number of address lines required for ***16 Kbyte memory is 14 lines*** and that ***of 32Kbytes of memory is 15 lines.***

The connections of external memory is shown below.



The lower order address and data bus are multiplexed. De-multiplexing is done by the latch. Initially the address will appear in the bus and this latched at the output of latch using ALE signal. The output of the latch is directly connected to the lower byte address lines of the memory. Later data will be available in this bus. Still the latch output is address it self. The higher byte of address bus is directly connected to the memory. The number of lines connected depends on the memory size.

The RD and WR (both active low) signals are connected to RAM for reading and writing the data.

PSEN of microcontroller is connected to the output enable of the ROM to read the data from the memory.

EA (active low) pin is always grounded if we use only external memory. Otherwise, once the program size exceeds internal memory the microcontroller will automatically switch to external memory.

STACK

A stack is a last in first out memory. In 8051 internal RAM space can be used as stack. The address of the stack is contained in a register called stack pointer. Instructions PUSH and POP are used for stack operations. When a data is to be placed on the stack, the stack pointer increments before storing the data on the stack so that the stack grows up as data is stored (pre-increment). As the data is retrieved from the stack the byte is read from the stack, and then SP decrements to point the next available byte of stored data (post decrement). The stack pointer is set to 07 when the 8051 resets. So that default stack memory starts from address location 08 onwards (to avoid overwriting the default register bank ie., bank0).

Eg; Show the stack and SP for the following.

```
                        [SP]=07         //CONTENT OF SP IS 07 (DEFAULT VALUE)
MOV R6,                 [R6]=25H        //CONTENT OF R6 IS25H
#25H MOV                [R1]=12H        //CONTENT OF R1 IS12H
R1, #12H                [R4]=F3H        //CONTENT OF R4 ISF3H
MOV R4, #0F3H

PUSH 6                  [SP]=08         [08]=[06]=25H         //CONTENT OF 08 IS 25H
PUSH 1                  [SP]=09         [09]=[01]=12H         //CONTENT OF 09 IS 12H
PUSH 4                  [SP]=0A         [0A]=[04]=F3H         //CONTENT OF 0A IS F3H

POP 6                   [06]=[0A]=F3H          [SP]=09 //CONTENT OF 06 IS F3H
POP 1                   [01]=[09]=12H          [SP]=08 //CONTENT OF 01 IS 12H
POP 4                   [04]=[08]=25H          [SP]=07 //CONTENT OF 04 IS 25H
```

I/O Ports:

8051 microcontroller have 4 I/O ports each of 8-bit, which can be configured as input or output. Hence, total 32 I/O pins allows the microcontroller to be connected with the peripheral devices.

1) PORT 0

P0 can be used as a bidirectional I/O port or it can be used for address/data connected for accessing external memory. When control is 1 the port is used for address or data interfacing. When the control is 0 then the port can be used as a bidirectional I/O port.

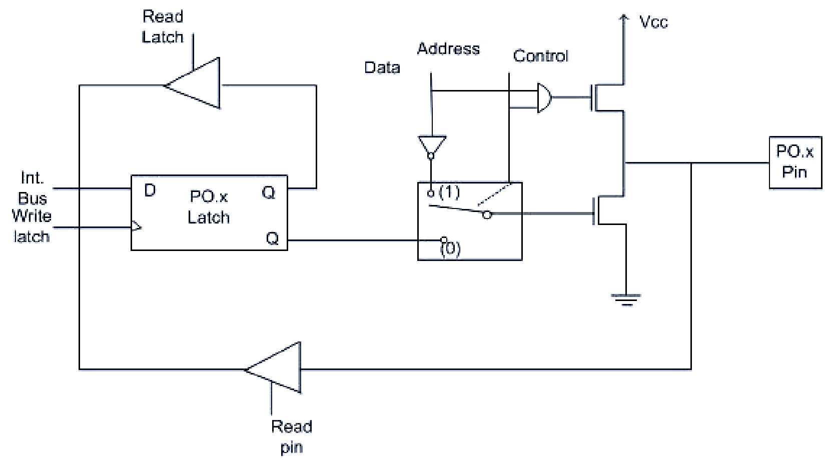


Fig: Structure of port 0 pin

PORT 0 as an Input Port

If the control is 0 then the port is used as an input port and 1 is written to the latch. In this type of situation both the output MOSFETs are off. Since the output pin has floats therefore, whatever data written on pin is directly read by read pin.

PORT 0 as an Output Port

If we want to write 1 on pin of P0, a '1' written to the latch which turns 'off' the lower FET while due to '0' control signal upper FET also turns off.

Suppose we want to write '0' on pin of port 0, when '0' is written to the latch, the pin is pulled down by the lower FET. Hence the output becomes zero.

2) PORT 1

PORT 1 is dedicated only for I/O interfacing. When used as an output port, not needed to connect additional pull-up resistor like port 0.

To use PORT 1 as an input port '1' has to be written to the latch. In this mode 1 is written to the pin by the external device then it read fine.

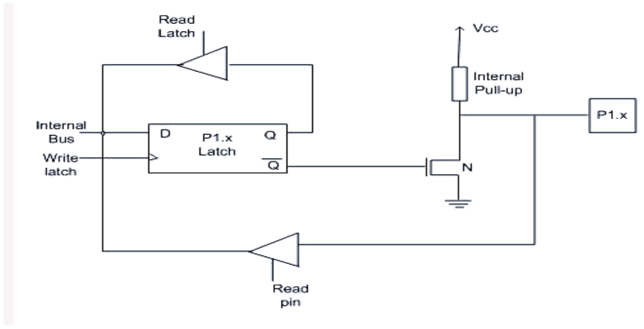


Fig: Structure of port 1 pin

3) PORT 2

PORT 2 is used for higher external address byte or a normal I/O port. Here, the I/O operation is similar to PORT 1. Latch of PORT 2 remains stable when Port 2 pin are used for external memory access.

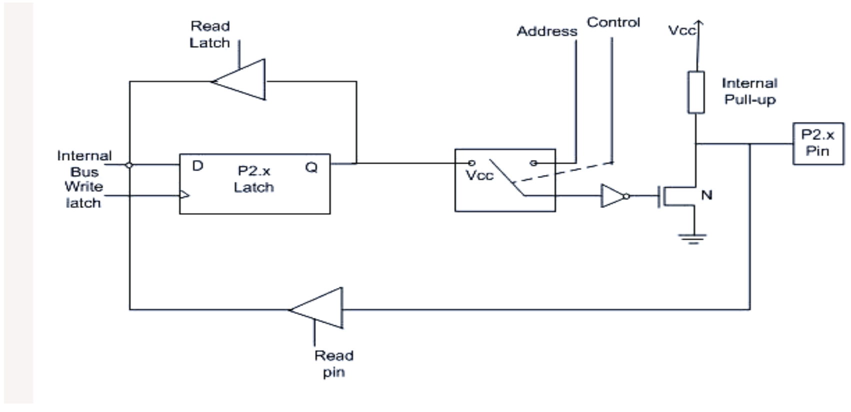


Fig: Structure of port 2 pin

4) PORT 3

Following are the alternate functions of PORT 3:

PORT 3 Pin	Function	Description
P3.0	RXD	Serial Input
P3.1	TXD	Serial Output
P3.2	$\boxed{\text{INT0}}$	External Interrupt 0
P3.3	$\boxed{\text{INT1}}$	External Interrupt 1
P3.4	T0	Timer 0
P3.5	T1	Timer 1
P3.6	$\boxed{\text{WR}}$	External Memory Write
P3.7	$\boxed{\text{RD}}$	External Memory Read

It works as an I/O port same like port 2. Alternate functions of port 3 makes its architecture different than other ports.

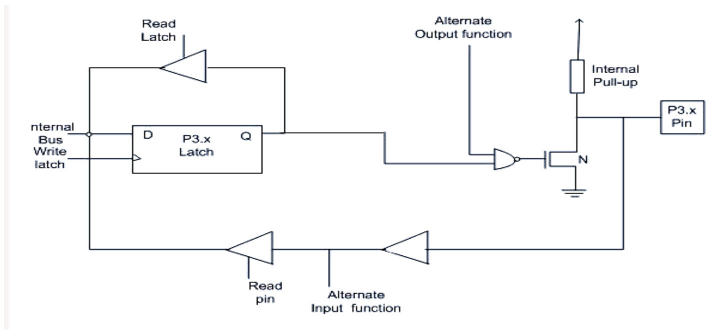


Fig: Structure of port 3

External Memory

The system designer is not limited by the amount of internal RAM and ROM available on chip. Two separate external memory spaces are made available by the 16-bit PC and DPTR and by different control pins for enabling external ROM and RAM chips. Internal control circuitry accesses the correct physical memory, depending upon the machine cycle state and the op code being executed.

There are several reasons for adding external memory, particularly program memory, when applying the 8051 in a system. When the project is in the prototype stage, the expense—in time and money—of having a masked internal ROM made for each program "try" is prohibitive.To alleviate this problem, the manufacturers make available an EPROM version, the 8751, which has 4K of on-chip EPROM that may be programmed and erased as needed as the program is developed. The resulting circuit board layout will be identical to one that uses a factory-programmed 8051. The only drawbacks to the 8751 are the specialized EPROM programmers that must be used to program the non-standard 40-pin part, and the limit of "only" 4096 bytes of program code. The 8751 solution works well if the program will fit into 4K bytes. Unfortunately, many times, particularly if the program is written in a high-level language, the program size exceeds 4K bytes, and an external program memory is needed. Again, the manufacturers provide a version for the job, the ROMIess 8031. The EA pin is grounded when using the 8031, and all program code is contained in an external EPROM that may be as large as 64K bytes and that can be programmed using standard EPROM programmers.

External RAM, which is accessed by the DPTR, may also be needed when 128 bytes of internal data storage is not sufficient. External RAM, up to 64K bytes, may also be added to any chip in the 8051 family.

Connecting External Memory

Figure 2.8 shows the connections between an 8031 and an external memory configuration consisting of I6K bytes of EPROM and 8K bytes of static RAM. The 8051 accesses external RAM whenever certain program instructions are executed. External ROM is accessed whenever the EA (external access) pin is connected to ground or when the PC contains an address higher than the last address in the internal 4K bytes ROM (OFFFh). 8051 designs can thus use internal and external ROM automatically; the 8031, having no internal ROM, must have EA grounded.

Figure 2.9 shows the timing associated with an external memory access cycle. During any memory access cycle, port 0 is time multiplexed. That is, it first provides the lower byte of the 16-bit memory address, then acts as a bidirectional data bus to write or read a byte of memory data. Port 2 provides the high byte of the memory address during the entire memory read/write cycle The lower address byte from port 0 must be latched into an external register to save the byte. Address byte save is accomplished by the ALE clock pulse that provides the correct timing for the '373 type data latch. The port 0 pins then become free to serve as a data bus. If the memory access is for a byte of program code in the ROM, the PSEN (program store enable) pin will go low to enable the ROM to place a byte of program code on the data bus. If the access is for a RAM byte, the WR (write) or RD (read) pins will go low, enabling data to flow between the RAM and the data bus.

The ROM may be expanded to 64K by using a 27512 type EPROM and connecting the remaining port 2 upper address lines AI4-A15 to the chip.

At this time the largest static RAMs available are 32K in size; RAM can be expanded to 64K by using two 32K RAMs that are connected through address A14 of port 2.

The first 32K RAM (OOOOh-7FFFh) can then be enabled when AI5 of port 2 is low, and the second 32K RAM (SOOOh-FFFFh) when A15 is high, by using an inverter.

Note that the WR and RD signals are alternate uses for port 3 pins 16 and 17. Also,port 0 is used for the lower address byte and data; port 2 is used for upper address bits. The use of external memory consumes many of the port pins, leaving only port 1 and parts of port 3 for general I/O.

FIGURE 2.8 External Memory Connections

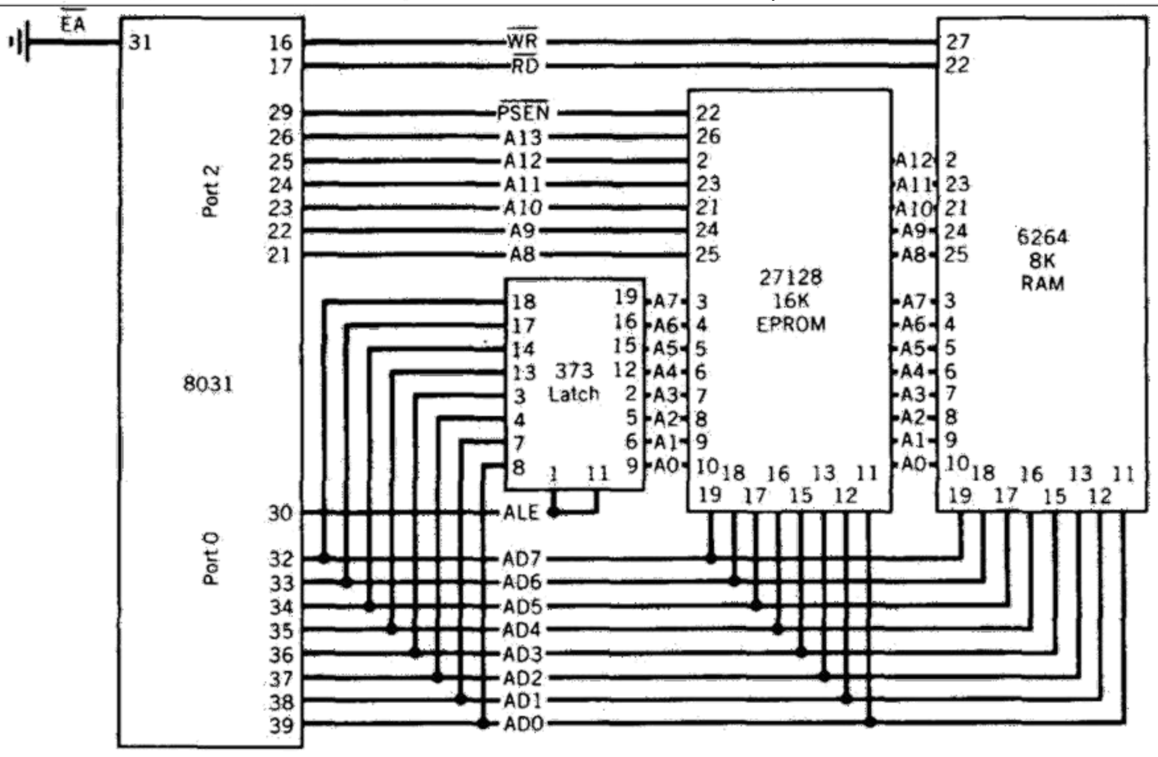
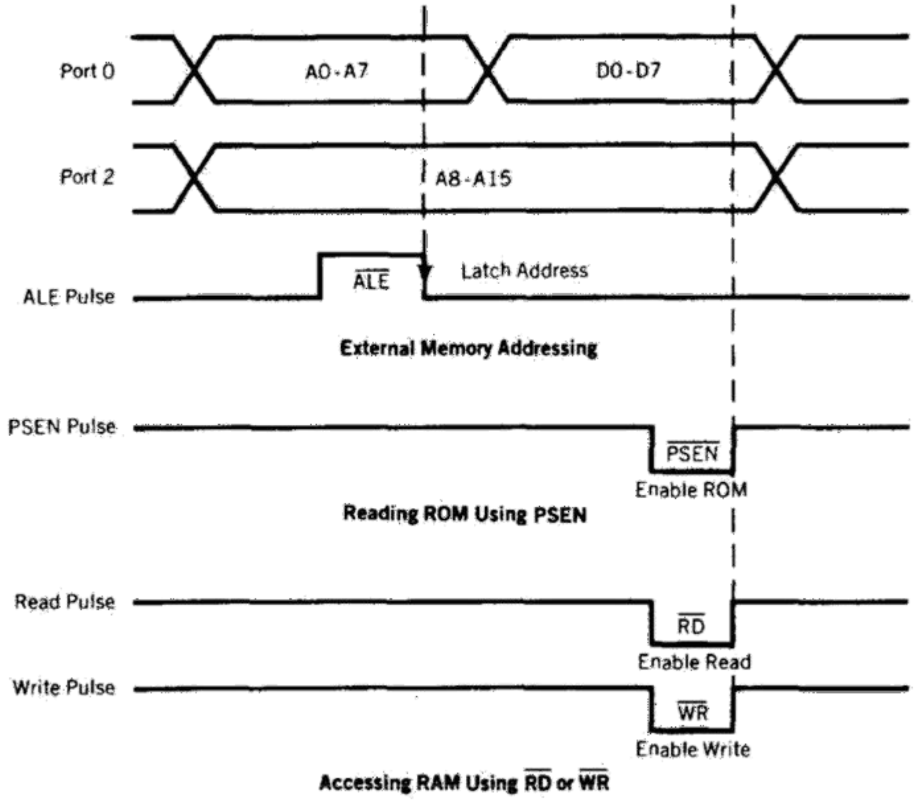


FIGURE 2.9 External Memory Timing



8051 INSTRUCTION SET

8051 has about 111 instructions. These can be grouped into the following categories

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Bit manipulation instructions

The following nomenclatures for register, data, address and variables are used while write instructions

- A: Accumulator
- B: "B" register
- C: Carry bit
- Rn: Register R0 - R7 of the currently selected register bank
- Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).
- @Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.
- #data8: Immediate 8-bit data available in the instruction.
- #data16: Immediate 16-bit data available in the instruction.
- Addr11: 11-bit destination address for short absolute jump. Used by instructions AJMP & ACALL. Jump range is 2 kbyte (one page).
- Addr16: 16-bit destination address for long call or long jump.
- Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps.
- bit: Directly addressed bit in internal RAM or SFR

Data transferinstructions.

In this group, the instructions perform data transfer operations of the following types.

a. Move the contents of a register Rn toA
   i. MOVA,R2
   ii. MOVA,R7

b. Move the contents of a register A toRn
   i. MOVR4,A
   ii. MOVR1,A

c. Move an immediate 8 bit data to register A or to Rn or to a memory location(direct or indirect)
   i. MOV A,#45H
   ii. MOV R6,#51H
   iii. MOV 30H,#44H
   iv. MOV @R0, #0E8H
   v. MOV DPTR,#0F5A2H
   vi. MOV DPTR,#5467H

d. Move the contents of a memory location to A or A to a memory location using directand indirectaddressing
   i. MOV A,65H
   ii. MOV A,@R0
   iii. MOV 45H,A
   iv. MOV @R1,A

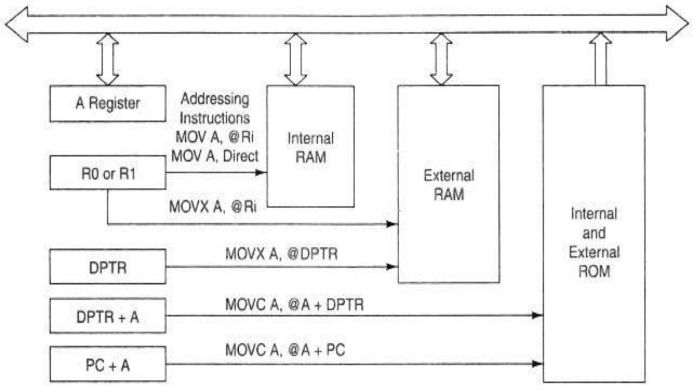
e. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing
   i. MOV R3,65H
   ii. MOV 45H,R2

f. Move the contents of memory location to anothermemory location using direct and indirectaddressing
   i. MOV 47H,65H
   ii. MOV 45H,@R0

g. Move the contents of an external memory to A or A to an externalmemory

i. MOVXA,@R1
ii. MOVX@R0,A

h. Move the contents of program memory toA
i. MOVC A,@A+PC
ii. MOVC A,@A+DPTR



MOVXA,@DPTR
iii. MOVX@DPTR,A

Arithmeticinstructions.

The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bit numbers.

Addition

In this group, we have instructions to

i. Add the contents of A with immediate data with or withoutcarry.
   i. ADD A,#45H
   ii. ADDC A,#OB4H
ii. Add the contents of A with register Rn with or withoutcarry.
   i. ADD A,R5
   ii. ADDC A,R2
iii. Add the contents of A with contents of memory with or without carry using direct and indirectaddressing
   i. ADD A, 51H
   ii. ADDC A, 75H
   iii. ADD A,@R1
   iv. ADDC A,@R0

CY AC and OV flags will be affected by this

operation. Subtraction

In this group, we have instructions to

i. Subtract the contents of A with immediate data with or withoutcarry.

   i. SUBB A,#45H
   ii. SUBB A,#OB4H

ii. Subtract the contents of A with register Rn with or withoutcarry.
   i. SUBB A,R5
   ii. SUBB A,R2

iii. Subtract the contents of A with contents of memory with or without carry using direct and indirectaddressing
   i. SUBB A,51H
   ii. SUBB A,75H
   iii. SUBB A,@R1
   iv. SUBB A,@R0

CY AC and OV flags will be affected by this

operation. Multiplication

MUL AB. This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

```
Eg.     MOVA,#45H              ;[A]=45H
        MOVB,#0F5H             ;[B]=F5H
        MULAB                  ;[A]x[B]=45xF5=4209
                               ;[A]=09H, [B]=42H
```

Division

DIV AB. This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and remainder will be stored in B register.

```
Eg.     MOVA,#45H              ;[A]=0E8H
        MOVB,#0F5H             ;[B]=1BH
        DIVAB                  ;[A]/[B]=E8/1B=08Hwithremainder10H
                               ;[A] = 08H, [B]=10H
```

DA A (Decimal Adjust After Addition).

When two BCD numbers are added, the answer is a non-BCD number. To get the result in BCD, we use DA A instruction after the addition. DA A works asfollows.

- If lower nibble is greater than 9 or auxiliary carry is 1, 6 is added to lowernibble.
- If upper nibble is greater than 9 or carry is 1, 6 is added to uppernibble.

```
Eg1:    MOV A,#23H
        MOV R1,#55H
        ADDA,R1          //[A]=78
        DA A             // [A]=78          nochangesintheaccumulatorafterdaa

Eg2:    MOV A,#53H
        MOV R1,#58H
        ADDA,R1          //[A]=ABh
         DA A            //[A]=11,C=1.ANSWERIS111.AccumulatordataischangedafterDAA
```

Increment: *increments the operand by one.*

INC A **INC Rn** **INC DIRECT** **INC @RiINCDPTR**

INC increments the value of source by 1. If the initial value of register is FFh, incrementing the value will cause it to reset to 0. The Carry Flag is not set when the value "rolls over" from 255 to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.

Decrement: *decrements the operand by one.*

DEC A **DEC RnDEC DIRECT** **DEC@Ri**

DEC decrements the value of *source* by 1. If the initial value of is 0, decrementing the value will cause it to reset to FFh. The Carry Flag is not set when the value "rolls over" from 0 to FFh.

LogicalInstructions

Logical AND

ANLdestination,source:ANL does a bitwise "AND" operation between *source*and *destination*, leaving the resulting value in *destination*. The value in source is not affected. "AND" instruction logically AND the bits of source anddestination.
ANL A,#DATA ANL A, Rn
ANL A,DIRECT ANL
A,@Ri
ANL DIRECT,A ANL DIRECT, #DATA

Logical OR

ORLdestination,source:ORLdoesabitwise"OR"operationbetween*source*and*destination*,

leaving the resulting value in *destination*. The value in source is not affected. " OR " instruction logically OR the bits of source and destination.
ORL A,#DATA ORL A, Rn
ORL A,DIRECT ORL
A,@Ri
ORL DIRECT,A ORL DIRECT, #DATA

Logical Ex-OR

XRLdestination,source: XRL does a bitwise "EX-OR" operation between *source* and *destination*,leavingtheresultingvaluein*destination*.Thevalueinsourceisnotaffected."XRL" instruction logically EX-OR the bits of source anddestination.
XRLA,#DATA XRL A,Rn
XRL A,DIRECT
XRLA,@Ri
XRL DIRECT,A XRL DIRECT, #DATA

Logical NOT

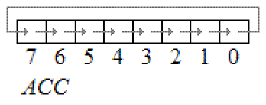
CPLcomplements*operand*,leavingtheresultin*operand*.If*operand*isasinglebitthenthestateof the bit will be reversed. If *operand* is the Accumulator then all the bits in the Accumulator will be reversed.

CPL A, CPL C, CPL bit address

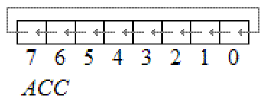
SWAP A – Swap the upper nibble and lower nibble of A.

Rotate Instructions

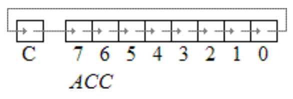
RR A
This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.



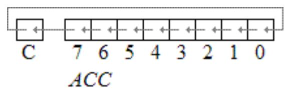
RL A
Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



RRC A
Rotate right through the carry. Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was at goes into bit 7



RLC A
Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.



Branch (JUMP)Instructions

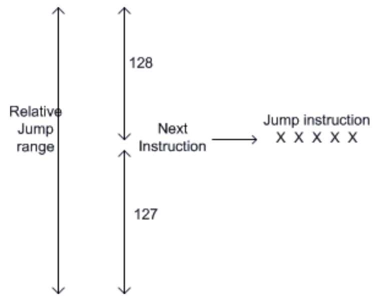
Jump and Call Program Range

There are 3 types of jump instructions. They are:-

1. RelativeJump
2. Short AbsoluteJump
3. Long AbsoluteJump

Relative Jump

Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a relative jump. Schematically, the relative jump can be shown as follows: -



The advantages of the relative jump are as follows:-

1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to-128.
2. Specifying only one byte reduces the size of the instruction and speeds up program execution.
3. The program with relative jumps can be relocated without reassembling to generate absolute jumpaddresses.

Disadvantages of the absolute jump: -

1. Short jump range (-128 to 127 from the instruction following the jump instruction)

Instructions that use Relative Jump

SJMP <relative address>; *this is unconditional jump*

The remaining relative jumps are conditional jumps

JC <relative address>
JNC <relative address>
JB bit, <relativeaddress>
JNB bit, <relative address>
JBC bit, <relative address>
CJNE <destination byte>, <source byte>, <relative address>
DJNZ <byte>, <relative address>
JZ <relative address>
JNZ <relative address>

Short Absolute Jump

In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.

In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

Page(Hex)	*Address (Hex)*
00	*0000-07FF*
01	*0800 -0FFF*
02	*1000-17FF*
03	*1800 -1FFF*
.	
.	
1E	*F000 -F7FF*
1F	*F800 -FFFF*

It can be seen that the upper 5bits of the program counter (PC) hold the page number and the lower 11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.

Advantage: The instruction length becomes 2

bytes. Example of short absolute jump: -

```
ACALL <address11>
AJMP   <address11>
```

LongAbsoluteJump/Call

Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is notre-locatable.

Example: -

```
LCALL <address 16>
LJMP <address 16>
JMP @A+DPTR
```

Another classification of jump instructions is

1. UnconditionalJump
2. ConditionalJump

1. **Theunconditionaljump** is ajumpinwhichcontrol istransferredunconditionallytothetargetlocation.
   a. **LJMP** (long jump). This is a 3-byte instruction. First byte is the op-code and second and third bytes represent the 16-bit target address which is any memory location from 0000 toFFFFH
   eg: LJMP 3000H
   b. **AJMP:** this causes unconditional branch to the indicated address, by loading the 11 bit address to 0-10bitsoftheprogramcounter.Thedestinationmustbethereforewithinthesame2Kblocks.
   c. **SJMP** (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH (forward +127 and backward -128 bytes from the current PC value). To calculate the target address of a short jump, the second byte is added to the PC value which is address of the instruction immediately below thejump.

2. Conditional Jumpinstructions.

JBC	Jump if bit ＝1 and clearbit
JNB	Jump if bit ＝0
JB	Jump if bit ＝1
JNC	Jump if CY ＝0
JC	Jump if CY ＝1
CJNEreg,#data	Jump if byte ≠ #data
CJNEA,byte	Jump if A ≠byte
DJNZ	Decrement and Jump if A ≠0
JNZ	Jump if A ≠0
JZ	Jump if A ＝0

All conditional jumps are short jumps.

Bit level jump instructions:

Bit level JUMP instructions will check the conditions of the bit and if condition is true, it jumps to the address specified in the instruction. All the bit jumps are relative jumps.

JBbit, rel	; jump if the direct bit is set to the relative address specified.
JNBbit,rel	;jumpifthedirectbitiscleartotherelativeaddressspecified.
JBCbit,rel	; jump if the direct bit is set to the relative address specified and then clear thebit.

Subroutine CALL And RETURNInstructions

Subroutines are handled by CALL and RET instructions There

are two types of CALL instructions

1. **LCALL address(16bit)**
   This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address. This is a 3 byte instruction. The LCALL instruction works as follows.
   a. DuringexecutionofLCALL,[PC]=[PC]+3;(ifaddress whereLCALLresides issay,0x3254; during execution of this instruction [PC] = 3254h + 3h =3257h
   b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and[SP]=08
   c. [[SP]] = [PC_{7-0}]; (lower byte of PC content ie., 57 will be stored in memory location08.
   d. [SP]=[SP]+1; (SP increments again and[SP]=09)
   e. [[SP]] = [PC_{15-8}]; (higher byte of PC content ie., 32 will be stored in memory location09.
   With these the address (0x3254) which was in PC is stored in stack.
   f. [PC]=address(16 bit);thenewaddressofsubroutineisloadedto PC.No flagsareaffected.

2. **ACALL address(11bit)**
   This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction. The SCALL instruction works as follows.
   a. DuringexecutionofSCALL,[PC]=[PC]+2;(ifaddress whereLCALLresides issay,0x8549; during execution of this instruction [PC] = 8549h + 2h =854Bh
   b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and[SP]=08
   c. [[SP]] = [PC_{7-0}]; (lower byte of PC content ie., 4B will be stored in memory location08.
   d. [SP]=[SP]+1; (SP increments again and[SP]=09)
   e. [[SP]] = [PC_{15-8}]; (higher byte of PC content ie., 85 will be stored in memory location09.

   With these the address (0x854B) which was in PC is stored in stack.

f. [PC_{10-0}]= address(11bit); the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

RET instruction pops top two contents from the stack and load it to PC.

g. [PC_{15-8}]=[[SP]];content of current top of the stack will be moved to higher byte ofPC.
h. [SP]=[SP]-1; (SPdecrements)
i. [PC_{7-0}] = [[SP]] ;content of bottom of the stack will be moved to lower byte ofPC.
j. [SP]=[SP]-1; (SP decrementsagain)

Bit manipulationinstructions.

8051 has 128 bit addressable memory. Bit addressable SFRs and bit addressable PORT pins. It is possible to perform following bit wise operations for these bit addressable locations.

1. LOGICAL AND
   a. ANLC,BIT(BITADDRESS) ; 'LOGICALLY AND' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT INCARRY
   b. ANLC,/BIT; ; 'LOGICALLY AND' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT INCARRY

2. LOGICALOR
   a. ORLC,BIT(BITADDRESS) ; 'LOGICALLY OR' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT INCARRY
   b. ORLC,/BIT; ; 'LOGICALLY OR' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT INCARRY
3. CLRbit
   a. CLRbit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BECLEARED.
   b. CLRC ; CONTENT OF CARRY WILL BECLEARED.
4. CPLbit
   a. CPLbit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BECOMPLEMENTED.
   b. CPLC ; CONTENT OF CARRY WILL BECOMPLEMENTEd

ASSEMBLERDIRECTIVES

Assembler directives tell the assembler to do something other than creating the machine code for an instruction. In assembly language programming, the assembler directives instruct the assembler to

1. Process subsequent assembly languageinstructions
2. Define program constants
3. Reserve space forvariables

The following are the widely used 8051 assembler directives.

ORG (origin)

The ORG directive is used to indicate the starting address. It can be used only when the program counter needs to be changed. The number that comes after ORG can be either in hex or in decimal.

Eg:ORG 0000H ;Set PC to 0000

.EQU andSET

EQU and SET directives assign numerical value or register name to the specified symbolname.

EQU is used to define a constant without storing information in the memory. The symbol defined with EQU should not be redefined.

SET directive allows redefinition of symbols at a later stage.

DB (DEFINE BYTE)

The DB directive is used to define an 8 bit data. DB directive initializes memory with 8 bit values. The numbers can be in decimal, binary, hex or in ASCII formats. For decimal, the 'D' after the decimal number is optional, but for binary and hexadecimal, 'B' and 'H' are required. For ASCII, the number is written in quotation marks ('LIKE This).

```
DATA1:  DB   40H           ; hex
DATA2:  DB   01011100B     ; b i n a r y
DATA3:  DB   48            ; decimal
DATA4:  DB   ' HELLO W'    ; ASCII
```

END

The END directive signals the end of the assembly module. It indicates the end of the program to the assembler. Any text in the assembly file that appears after the END directive is ignored. If the END statement is missing, the assembler will generate an error.

ADDRESSING MODES

Various methods of accessing the data are called addressin g modes. 8051 addressing modes are classified as follows.

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.
5. Relative addressing.
6. Absolute addressing.
7. Long addressing.
8. Indexed addressing.
9. Bit inherent addressing.
10. Bit direct addressing.

1.Immediate addressing.

In this addressing mode the data is provided as a part of instruction itself. In other words data immediately follows the instruction.

```
Eg.    MOVA,#30H
       ADDA, #83                      # Symbol indicates the data isimmediate
```

2.Register addressing

In this addressing mode the register will hold the data. One of the eight general registers (R0 to R7) can be used and specified as theoperand.

Eg. MOV A,R0
ADDA,R6

R0 – R7 will be selected from the current selection of register bank. The default register bank will be bank 0.

3.Direct addressing

There are two ways to access the internal memory. Using direct address and indirect address. Using direct addressing mode we can not only address the internal memory but SFRs also. In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory.

Eg. MOV
A,60h
ADDA,30h

4.Indirect addressing

The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0 and R1 and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold 16 bit address.

Eg. MOV A,@R0
ADD A,@R1
MOVXA,@DP
TR

5. Indexed addressing

In indexed addressing, either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

Eg. MOVCA,@A+DPTR // copies the contents of memory location pointed by the sum of the accumulatorAandtheDPTRintoaccumulatorA.
MOVCA,@A+PC //copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.

6.Relative Addressing.

Relative addressing is used only with conditional jump instructions. The relative address, (offset), is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to —128 locations. The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

```
Eg:     SJMP LOOP1
        JCBACK
```

7. Absolute addressing

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11 bit of the memory. The upper 5 bit of the destination address are the upper 5 bit of the current program counter. Hence, absolute addressing allows branching only within the current 2 Kbyte page of the program memory.

```
Eg.     AJMP LOOP1
        ACALL LOOP2
```

8.Long Addressing

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space.

```
Eg.     LJMP FINISH
        LCALLDELAY
```

9.Bit Inherent Addressing

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

Eg. CLRC *; Clears the carry flag to0*

10.Bit Direct Addressing

In this addressing mode the direct address of the bit is specified in the instruction. The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

Eg. CLR 07h *; Clears the bit 7 of 20h RAM space*

SETB 07H *; Sets the bit 7of 20H RAM space*

ASSEMBLY LANGUAGE PROGRAMS

1. **Write a program to add the values of locations 50H and 51H and store the result in locations in 52h and 53H.**

```
ORG0000H            ; Set program counter0000H
MOVA,50H            ; Load the contents of Memory location 50H into A ADD ADDA,51H
        ; Add the contents of memory 51H with CONTENTSA
MOV52H,A            ; Save the LS byte of the result in 52H
MOVA, #00           ; Load 00H intoA
ADDCA,#00           ; Add the immediate data and carry toA
MOV53H,A            ; Save the MS byte of the result in location53h
END
```

2. **Write a program to stored at a FFH into RAM memory locations 50H to 58H using direct addressing mode**

```
ORG0000H            ; Set program counter0000H
MOV A, #0FFH ; Load FFH intoA
MOV50H,A            ; Store contents of A in location50H
MOV51H,A            ; Store contents of A in location 5IH
MOV52H,A            ; Store contents of A in location52H
MOV53H,A            ; Store contents of A in location53H
MOV54H,A            ; Store contents of A in location54H
MOV55H,A            ; Store contents of A in location55H
MOV56H,A            ; Store contents of A in location56H
MOV57H,A            ; Store contents of A in location57H
MOV58H,A            ; Store contents of A in location58H
END
```

3. **Write a program to subtract a 16 bit number stored at locations 51H-52H from 55H-56H and storetheresultinlocations40Hand41H.Assumethattheleastsignificantbyteofdataorthe result is stored in low address. If the result is positive, then store 00H, else store 01H in 42H.** ORG0000H ; Set program counter0000H

```
MOVA,55H     ; Load the contents of memory location 55 intoA
CLRC         ; Clear the borrowflag
SUBBA,51H    ; Sub the contents of memory 51H from contents ofA
MOV40H,A     ; Save the LSByte of the result in location40H
MOVA,56H     ; Load the contents of memory location 56H intoA
SUBBA,52H    ; Subtract the content of memory 52H from the contentA
MOV41H,      ; Save the MSbyte of the result in location415.
MOVA, #00    ; Load 005 intoA
ADDCA,#00    ; Add the immediate data and the carry flag to A
MOV42H,A     ; If result is positive, store00H, else store 0lH in42H
END
```

4. **Write a program to add two 16 bit numbers stored at locations 51H-52H and 55H-56H and store the result in locations 40H, 41H and 42H. Assume that the least significant byte of data and the result is stored in low address and the most significant byte of data or the result is stored in highaddress.**

```
ORG0000H          ; Set program counter0000H
MOVA,51H          ; Load the contents of memory location 51H intoA
ADDA,55H          ; Add the contents of 55H with contents of A
MOV40H,A          ; Save the LS byte of the result in location 40H
MOVA,52H          ; Load the contents of 52H intoA
ADDCA,56H         ; Add the contents of 56H and CY flag withA
MOV41H,A          ; Save the second byte of the result in 41H
MOVA,#00          ; Load 00H intoA
ADDC A,#00 ; Add the immediate data 00H and CY to A
MOV42H,A          ; Save the MS byte of the result in location42H
END
```

5. **Write a program to store data FFH into RAM memory locations 50H to 58H using indirect addressingmode.**

```
      ORG0000H          ; Set program counter0000H
      MOVA,#0FFH        ; Load FFH intoA
      MOVRO,#50H        ; Load pointer, R0-50H
      MOVR5,#08H        ; Load counter,R5-08H
Start:MOV@RO,A          ; Copy contents of A to RAM pointed by R0
      INCRO             ; Increment pointer
       DJNZ R5, start ; Repeat until R5 is zero
       END
```

6. **Write a program to add two Binary Coded Decimal (BCD) numbers stored at locations 60H and 61H and store the result in BCD at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in lowaddress.**

```
ORG0000H     ; Set program counter00004
MOVA,60H     ; Load the contents of memory location 6.0.H intoA
ADDA,61H     ;Addthecontentsofmemorylocation61HwithcontentsofA DAA
             ; Decimal adjustment of the sum inA
MOV52H,A     ;Savetheleastsignificantbyteoftheresultinlocation52H
MOVA,#00     ; Load 00H into.A
ADDCA,#00H   ;AddtheimmediatedataandthecontentsofcarryflagtoA
MOV53H,A     ; Save the most significant byte of the result in location 53:,
END
```

7. **Writeaprogramtoclear10RAMlocationsstartingatRAMaddress1000H.**

```
       ORG0000H            ;Set program counter 0000H
       MOVDPTR, #1000H     ;Copy address 1000H toDPTR
       CLRA         ;ClearA
       MOVR6,#0AH          ;Load 0AH toR6
again:MOVX@DPTR,A          ;Clear RAM location pointed byDPTR
```

```
            INCDPTR              ;IncrementDPTR
            DJNZR6,again         ;Loop until counterR6=0
            END
```

8. Write a program to compute 1 + 2 + 3 + N (say N=15) and save the sumat70H

```
              ORG0000H                   ; Set program counter0000H
            N EQU 15
            MOV R0,#00                 ; ClearR0
            CLRA                               ; ClearA
again:      INC R0                     ; Increment R0
            ADDA,R0                    ; Add the contents of R0 withA
            CJNE R 0,# N, again ; Loop until counter, R0,N
            MOV70H,A                   ; Save the result in location 70HEND
```

9. Write a program to multiply two 8 bit numbers stored at locations 70H and 71H and store the resultatmemorylocations52Hand53H.Assumethattheleastsignificantbyteoftheresultis stored in lowaddress.

```
ORG 0000H ; Set program counter 00 OH
MOVA,70H;Loadthecontentsofmemorylocation70hintoA
MOVB,71H;Loadthecontentsofmemorylocation71HintoB
MULAB          ; Performmultiplication
MOV52H,A;Savetheleastsignificantbyteoftheresultinlocation52HMOV53H,B;Savethemost significant
byte of the result inlocation 53
END
```

10. Ten 8 bit numbers are stored in internal data memory from location 5oH. Write a program to increment thedata.

Assume that ten 8 bit numbers are stored in internal data memory from location 50H, hence R0 or R1 must be used as a pointer.

The program is as follows.

```
            OPT 0000H
            MOV R0,#50H
            MOV R3,#0AH
            Loopl: INC @R0
            INC RO
            DJNZ R3, loopl END
            END
```

11. Write a program to find the average of five 8 bit numbers. Store the result in H. (Assume that after adding five 8 bit numbers, the result is 8 bitonly).

```
            ORG 0000H
            MOV 40H,#05H
            MOV 41H,#55H
            MOV 42H,#06H
            MOV 43H,#1AH
            MOV 44H,#09H
            MOV R0,#40H
            MOV R5,#05H
            MOV B,R5
            CLR A
            Loop: ADD A,@RO
            INC RO
```

```
DJNZ R5,Loop
DIV AB
MOV55H,A
END
```

12. Write a program to find the cube of an 8bit number program is as follows

```
ORG 0000H
MOV
R1,#N
MOV A,R1
MOVB,R1
MULAB                  //SQUARE
ISCOMPUTED MOV R2,B
MOV B, R1
MUL AB
MOV 50,A
MOV 51,B
MOV A,R2
MOV B, R1
MUL AB
ADD A, 51H
MOV 51H,
A MOV 52H,
B
MOVA,#00H
ADDC
A,52H
MOV52H,A               //CUBE IS STORED
IN52H,51H,50H END
```

13. Write a program to exchange the lower nibble of data present in external memory 6000H and 6001H

```
ORG0000H                      ;Setprogramcounter00h MOV
DPTR, # 6000 H ; Copy address 6000 H toDPTR
MOVXA,@DPTR          ;Copycontentsof60008toA
MOVR0,#45H           ;Loadpointer,R0=45H
MOV@RO,A             ;CopycontofAtoRAMpointedby80
INCDPL               ;Increment  pointer
MOVXA,@DPTR          ;Copycontentsof60018toA
XCHDA,@R0            ; E x c h a n g e l o w e r n i b b l e o f A w i t h R A M p o i n t e d b y RO
MOVX@DPTR,A          ;CopycontentsofAto60018 DECDPL
                     ;Decrementpointer
MOVA,@R0             ; Copycont of RAMpointed by R0 to A
MOVX@DPTR,A          ; Copy cont of A to RAMpointed by DPTR
END
```

14. Write a program to count the number of and o's of 8 bit data stored in location6000H.

```
ORG00008                      ; Set program counter 00008
MOVDPTR,#6000h           ; Copy address 6000H toDPTR
MOVXA,@DPTR                   ; Copy num be r t o A
MOVR0,#08                     ;  Copy   08  inRO
MOVR2,#00                     ;  C o py 00 in R 2
MOVR3,#00                     ;  C o py 00 in R 3
```

```
                    CLRC                        ; Clear carryflag
BACK: RLCA   ;  Athroughcarryf

                    JCNEXT                      ; I f C F = 1 , b r a n c ht o n e x t
                    INCR2                  ; I f C F = 0 , i n c r e m e n t R 2AJMP
                    NEXT2 NEXT: INCR3           ; I f C F = 1 , i n c r e m e n t R3
                    NEXT2:DJNZRO,BACK           ;RepeatuntilRO iszero END
```

15. **Write a program to shift a 24 bit number stored at 57H-55H to the left logically four places. Assume that the least significant byte of data is stored in loweraddress.**

```
                    ORG0000H       ; Set program counter0000h
                    MOVR1,#04      ; Set up loop count to4
again:  MOVA,55H        ; Place the least significant byte of data inA
                    CLRC           ; Clear tne carryflag
                    RLCA           ; Rotate contents of A (55h) left throughcarry
                    MOV55H,A
                    MOVA,56H
                    RLCA           ; Rotate contents of A (56H) left throughcarry
                    MOV56H,A
                    MOVA,57H
                    RLCA           ; Rotate contentsofA       (57H) left throughcarry
                    MOV57H,A
                    DJNZ R1,again ; Repeat until R1 is zero
                    END
```

16. Two 8 bit numbers are stored in location 1000h and 1001h of external data memory.

Write a program to find the GCD of the numbers and store the result in 2000h.

ALGORITHM

- *Step1 :Initialize external data memory with data and DPTR with address*
- *Step2 :Load A and TEMP with the operands*
- *Step3 :Are the two operands equal? If yes,go to step 9*
- *Step4 :Is(A)greater than(TEMP)? If yes,go to step6*
- *Step5 :Exchange (A)with(TEMP)such that A contains the bigger number*
- *Step6 :Perform division operation(contents of A with contents of TEMP)*
- *Step7 :If the remainder is zero,go to step9*
- *Step8 :Move the remainder into A and go to step4*
- *Step9 :Save the contents 'of TEMP in memory and term in at the program*

```
                    ORG0000H                ; Set program counter 0000H
                    TEMP EQU70H
                    TEMPI EQU 71H
                    MOVDPTR, #1000H         ; Copy address 100011 toDPTR
                    MOVXA,@DPTR             ; Copy First number toA
                    MOVTEMP,A               ; Copy First number to temp INCDPTR
                    MOVXA,@DPTR             ; Copy Second number toA
LOOPS: CJNE A, TEMP, LOOP1 ; (A) /= (TEMP) branch to LOOP1
                    AJMP LOOP2                      ; (A) = (TEMP) branch toL00P2
LOOP1:              JNC LOOP3                       ; (A) > (TEMP) branch toLOOP3
                    NOV TEMPI, A            ; (A) < (TEMP) exchange (A) with (TEMP)
                    MOV A, TEMP
                    MOV TEMP, TEMPI
LOOP3:              MOV B, TEMP
                    DIV AB                  ; Divide (A) by (TEMP)
                    MOV A, B                        ; Move remainder to A
                    CJNE A,#00, LOOPS       ; (A)/=00 branch to LOOPS
LOOP2:              MOV A, TEMP
```

```
MOV DPTR, #2000H
MOVX @DPTR, A                  ; Store the result in 2000H
END
```

UNIT-V

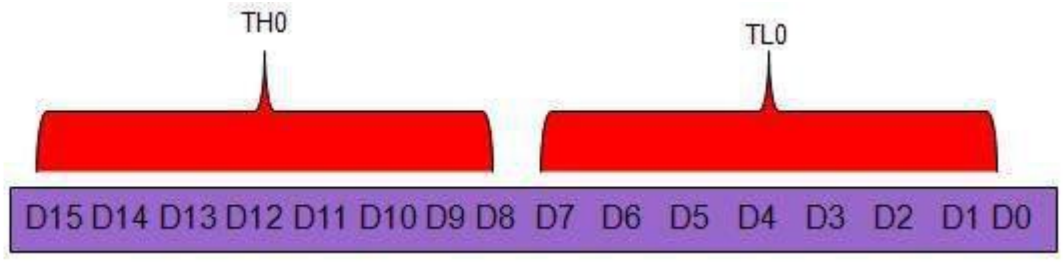
8051 Real Time Control

Timers of 8051 and their Associated Registers

The 8051 has two timers, Timer 0 and Timer 1. They can be used as timers or as event counters. Both Timer 0 and Timer 1 are 16-bit wide. Since the 8051 follows an 8-bit architecture, each 16 bit is accessed as two separate registers of low-byte and high-byte.

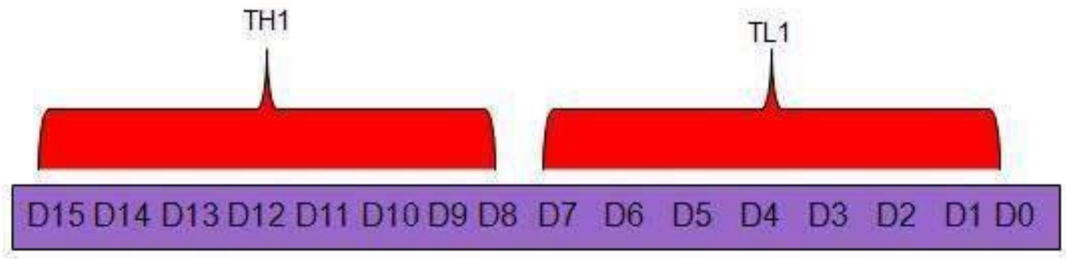
Timer 0 Register

The 16-bit register of Timer 0 is accessed as low- and high-byte. The low-byte register is called TL0 (Timer 0 low byte) and the high-byte register is called TH0 (Timer 0 high byte). These registers can be accessed like any other register. For example, the instruction **MOV TL0, #4H** moves the value into the low-byte of Timer #0.



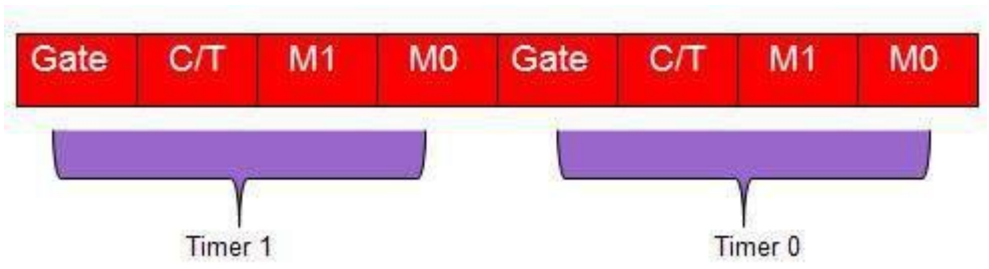
Timer 1 Register

The 16-bit register of Timer 1 is accessed as low- and high-byte. The low-byte register is called TL1 (Timer 1 low byte) and the high-byte register is called TH1 (Timer 1 high byte). These registers can be accessed like any other register. For example, the instruction **MOV TL1, #4H** moves the value into the low-byte of Timer 1.



TMOD (Timer Mode) Register

Both Timer 0 and Timer 1 use the same register to set the various timer operation modes. It is an 8-bit register in which the lower 4 bits are set aside for Timer 0 and the upper four bits for Timers. In each case, the lower 2 bits are used to set the timer mode in advance and the upper 2 bits are used to specify the location.



Gate − When set, the timer only runs while INT(0,1) is high.

C/T − Counter/Timer select bit.

M1 − Mode bit 1.

M0 − Mode bit 0.

GATE

Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. 8051 timers have both software and hardware controls. The start and stop of a timer is controlled by software using the instruction **SETB TR1** and **CLR TR1** for timer 1, and **SETB TR0** and **CLR TR0** for timer 0.

The SETB instruction is used to start it and it is stopped by the CLR instruction. These instructions start and stop the timers as long as GATE = 0 in the TMOD register. Timers can be started and stopped by an external source by making GATE = 1 in the TMOD register.

C/T (CLOCK / TIMER)

This bit in the TMOD register is used to decide whether a timer is used as a **delay generator** or an **event manager**. If C/T = 0, it is used as a timer for timer delay generation. The clock source to create the time delay is the crystal frequency of the 8051. If C/T = 0, the crystal frequency attached to the 8051 also decides the speed at which the 8051 timer ticks at a regular interval.

Timer frequency is always 1/12th of the frequency of the crystal attached to the 8051. Although various 8051 based systems have an XTAL frequency of 10 MHz to 40 MHz, we normally work with the XTAL frequency of 11.0592 MHz. It is because the baud rate for serial communication of the 8051.XTAL = 11.0592 allows the 8051 system to communicate with the PC with no errors.

M1 / M2

M1	M2	Mode
0	0	13-bit timer mode.
0	1	16-bit timer mode.
1	0	8-bit auto reload mode.
1	1	Spilt mode.

Different Modes of Timers

Mode 0 (13-Bit Timer Mode)

Both Timer 1 and Timer 0 in Mode 0 operate as 8-bit counters (with a divide-by-32 prescaler). Timer register is configured as a 13-bit register consisting of all the 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the register. The timer interrupt flag TF1 is set when the count rolls over from all 1s to all 0s. Mode 0 operation is the same for Timer 0 as it is for Timer 1.

Mode 1 (16-Bit Timer Mode)

Timer mode "1" is a 16-bit timer and is a commonly used mode. It functions in the same way as 13-bit mode except that all 16 bits are used. TLx is incremented starting from 0 to a maximum 255. Once the value 255 is reached, TLx resets to 0 and then THx is incremented by 1. As being a full 16-bit timer, the timer may contain up to 65536 distinct values and it will overflow back to 0 after 65,536 machine cycles.

If a value, say YYXXH, is loaded into the Timer bytes, then the delay produced by the Timer will be equal to the product :
[(**FFFFH – YYXXH +1**) x (period of one timer clock)].
It can also be considered as follows: convert YYXXH into decimal, say NNNNN, then delay will be equal to the product :
[(**65536-NNNNN**) x (period of one timer clock)].
The period of one timer clock is 1.085 µs for a crystal of 11.0592 MHz frequency as discussed above.
Now to produce a desired delay, divide the required delay by the Timer clock period. Assume that the division yields a number NNNNN. This is the number of times Timer must be updated before it stops. Subtract this number from 65536 (binary equivalent of FFFFH) and convert the difference into hex. This will be the initial value to be loaded into the Timer to get the desired delay.

Mode 2 (8 Bit Auto Reload)

Both the timer registers are configured as 8-bit counters (TL1 and TL0) with automatic reload. Overflow from TL1 (TL0) sets TF1 (TF0) and also reloads TL1 (TL0) with the contents of Th1 (TH0), which is preset by software. The reload leaves TH1 (TH0) unchanged.

The benefit of auto-reload mode is that you can have the timer to always contain a value from 200 to 255. If you use mode 0 or 1, you would have to check in the code to see the overflow and, in that case, reset the timer to 200. In this case, precious instructions check the value and/or get reloaded. In mode 2, the microcontroller takes care of this. Once you have configured a timer in mode 2, you don't have to worry about checking to see if the timer has overflowed, nor do you have to worry about resetting the value because the microcontroller hardware will do it all for you. The auto-reload mode is used for establishing a common baud rate.

Mode 3 (Split Timer Mode)

Timer mode "3" is known as **split-timer mode**. When Timer 0 is placed in mode 3, it becomes two separate 8-bit timers. Timer 0 is TL0 and Timer 1 is TH0. Both the timers count from 0 to 255 and in case of overflow, reset back to 0. All the bits that are of Timer 1 will now be tied to TH0.

When Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be set in modes 0, 1 or 2, but it cannot be started/stopped as the bits that do that are now linked to TH0. The real timer 1 will be incremented with every machine cycle.

Initializing a Timer

Decide the timer mode. Consider a 16-bit timer that runs continuously, and is independent of any external pins.

Initialize the TMOD SFR. Use the lowest 4 bits of TMOD and consider Timer 0. Keep the two bits, GATE 0 and C/T 0, as 0, since we want the timer to be independent of the external pins. As 16-bit mode is timer mode 1, clear T0M1 and set T0M0. Effectively, the only bit to turn on is bit 0 of TMOD. Now execute the following instruction −

```
MOV TMOD,#01h
```

Now, Timer 0 is in 16-bit timer mode, but the timer is not running. To start the timer in running mode, set the TR0 bit by executing the following instruction −

```
SETB TR0
```

Now, Timer 0 will immediately start counting, being incremented once every machine cycle.

Reading a Timer

A 16-bit timer can be read in two ways. Either read the actual value of the timer as a 16-bit number, or you detect when the timer has overflowed.

Detecting Timer Overflow

When a timer overflows from its highest value to 0, the microcontroller automatically sets the TFx bit in the TCON register. So instead of checking the exact value of the timer, the TFx bit can be checked. If TF0 is set, then Timer 0 has overflowed; if TF1 is set, then Timer 1 has overflowed.

Synchronous and Asynchronous serial communication:

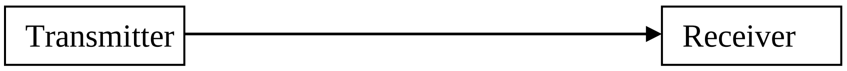
DATACOMMUNICATION

The 8051 microcontroller is parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices. Parallel data transfer over a long is very expensive. Hence, a serial communication is widely used in long distance communication. In serial data communication, 8-bit data is converted to serial bits using a parallel in serial out shift register and then it is transmitted over a single data line. The data byte is always transmitted with least significant bit first.

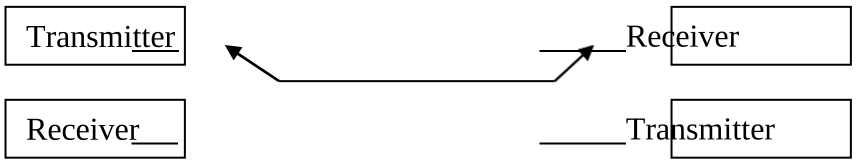
BASICS OF SERIAL DATACOMMUNICATION

Communication Links

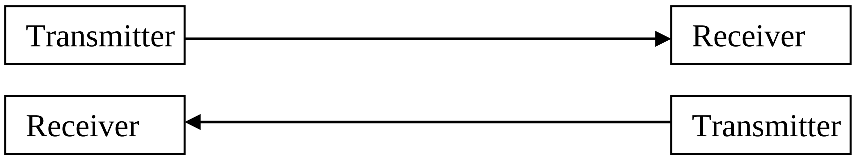
1. **Simplex communication link*:***In simplex transmission,the line is dedicated for transmission. The transmitter sends and the receiver receives the data.



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



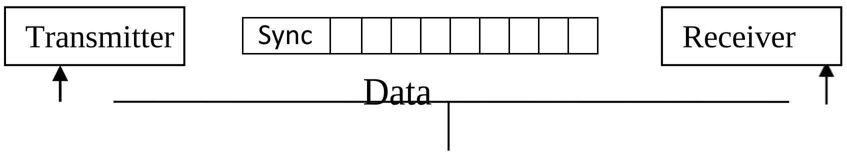
3.Full duplex communication link :If the data is transmitted in both ways at the same time,it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.



Types of Serial communication:

Serial data communication uses two types of communication.

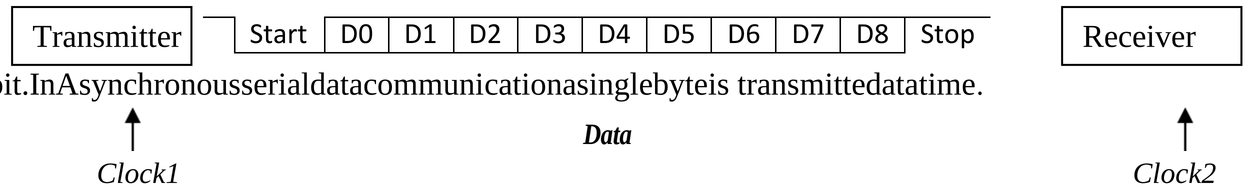
1. **Synchronous serial data communication:**In this transmitter and receiver are synchronized. It uses a common clock to synchronize the receiver and the transmitter. First the synch character is sent and then the data is transmitted. This format is generally used for high speed transmission. In



Clock

Synchronous serial data communication a block of data is transmitted at a time.

Asynchronous Serial data transmission*:*In this,different clock sources are used for transmitter and receiver. In this mode, data is transmitted with start and stop bits. A transmission begins with start bit, followed by data and then stop bit. For error checking purpose parity bit is included just priortostopbit.InAsynchronousserialdatacommunicationasinglebyteis transmittedatatime.



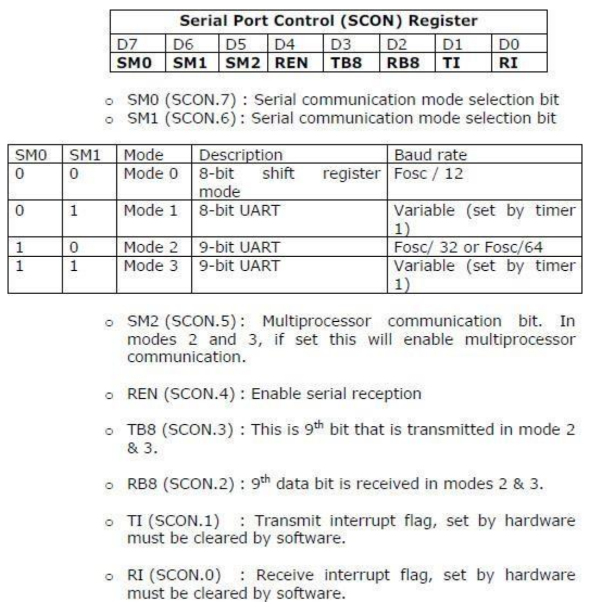
Baud rate:

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

8051 SERIALCOMMUNICATION

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

1. SBUF Register: Serial Buffer (SBUF) register is an 8-bit register. It has separate SBUF registers for data transmission and for data reception. For a byte of data to be transferred via the TXD line, it must be placed in SBUF register. Similarly, SBUF holds the 8-bit data received by the RXD pin and read to accept the receiveddata.
2. SCON register: The contents of the Serial Control (SCON) register are shown below. Thisregister contains mode selection bits, serial port interrupt bit (TI and RI) and also the ninth data bit for transmission and reception (TB8 andRB8).



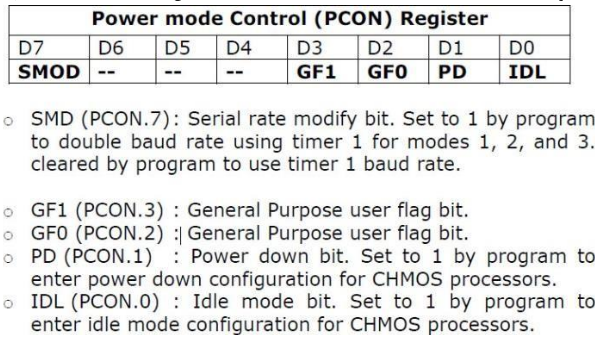
Serial Port Control (SCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

- SM0 (SCON.7) : Serial communication mode selection bit
- SM1 (SCON.6) : Serial communication mode selection bit

SM0	SM1	Mode	Description	Baud rate
0	0	Mode 0	8-bit shift register mode	Fosc / 12
0	1	Mode 1	8-bit UART	Variable (set by timer 1)
1	0	Mode 2	9-bit UART	Fosc/ 32 or Fosc/64
1	1	Mode 3	9-bit UART	Variable (set by timer 1)

- SM2 (SCON.5) : Multiprocessor communication bit. In modes 2 and 3, if set this will enable multiprocessor communication.
- REN (SCON.4) : Enable serial reception
- TB8 (SCON.3) : This is 9th bit that is transmitted in mode 2 & 3.
- RB8 (SCON.2) : 9th data bit is received in modes 2 & 3.
- TI (SCON.1) : Transmit interrupt flag, set by hardware must be cleared by software.
- RI (SCON.0) : Receive interrupt flag, set by hardware must be cleared by software.

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



Power mode Control (PCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SMOD	**--**	**--**	**--**	**GF1**	**GF0**	**PD**	**IDL**

- SMD (PCON.7): Serial rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2, and 3. cleared by program to use timer 1 baud rate.
- GF1 (PCON.3) : General Purpose user flag bit.
- GF0 (PCON.2) : General Purpose user flag bit.
- PD (PCON.1) : Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.
- IDL (PCON.0) : Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors.

SERIAL COMMUNICATION MODES

Mode0

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

Mode 1

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

$$\text{Baudrate} = [2^{smod}/32] \times \text{Timer 1 overflow Rate}$$
$$= [2^{smod}/32] \times [\text{Oscillator Clock Frequency}] / [12 \times [256 - [TH1]]]$$

Mode 2

This is similar to mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bit, a programmable 9th data bit, 1 stop bit.

$$\text{Baudrate} = [2^{smod}/64] \times \text{Oscillator Clock Frequency}$$

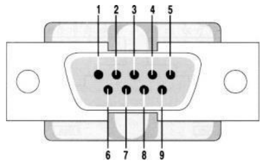
Mode 3

This is similar to mode 2 except baud rate is calculated as in mode 1

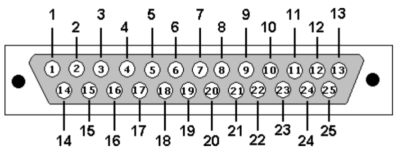
CONNECTORS-232

RS-232 standards:

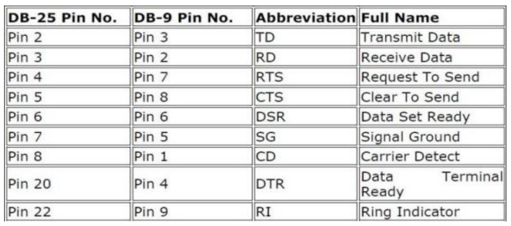
To allow compatibility among data communication equipment made by various manufactures, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. Since the standard was set long before the advent of logic family, its input and output voltage levels are not TTL compatible.
In RS232, a logic one (1) is represented by -3 to -25V and referred as MARK while logic zero
(0) is represented by +3 to +25V and referred as SPACE. For this reason to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic level to RS232 voltage levels and vice-versa. MAX232 IC chips are commonly referred as line drivers.
In RS232 standard we use two types of connectors. DB9 connector or DB25 connector.



DB9MaleConnector



DB25MaleConnector

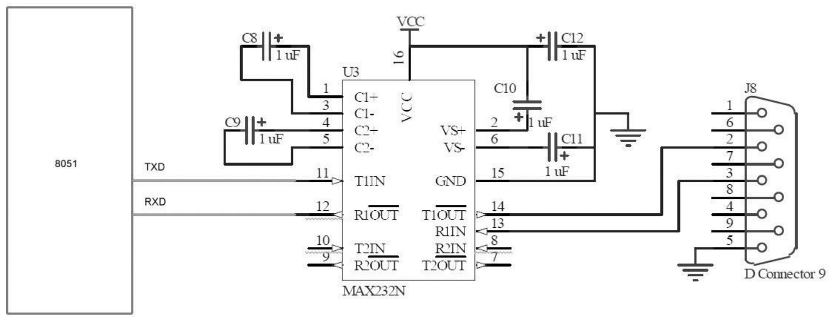


DB-25 Pin No.	DB-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

The pin description of DB9 and DB25 Connectors are as above

The 8051 connection to MAX232 is as follows.

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TXD, RXD. Pin 11 of the 8051 (P3.1) assigned to TXD and pin 10 (P3.0) is designated as RXD. These pins TTL compatible; therefore they require line driver (MAX 232) to make them RS232 compatible. MAX 232 converts RS232 voltage levels to TTL voltage levels and vice versa. One advantage of the MAX232 is that it uses a +5V power source which is the same as the source voltage for the 8051. The typical connection diagram between MAX 232 and 8051 is shown below.



SERIAL COMMUNICATION PROGRAMMING IN ASSEMBLY AND C.

Steps to programming the 8051 to transfer data serially

1.The TMOD register is loaded with the value 20H, indicating the use of the Timer 1 in mode 2 (8-bit auto reload) to set the baudrate.

2.The TH1 is loaded with one of the values in table 5.1 to set the baud rate for serial data transfer.

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

4.TR1 is set to 1 start timer1.

5.TI is cleared by the "CLR TI"instruction.

6.The character byte to be transferred serially is written into the SBUFregister.

7.The TI flag bit is monitored with the use of the instruction JNB TI, target to see if the character has been transferredcompletely.

8.To transfer the next character, go to step5.

Example1.Write a program for the 8051 to transfer letter 'A' serially at 4800-baudrate,8bit data, 1 stop bit continuously.

```
ORG 0000H LJMP
START ORG 0030H
START: MOVTMOD,#20H      ; select timer 1 mode2
MOVTH1,#0FAH             ; load count to get baud rate of 4800
MOVSCON,#50H             ; initialize UART in mode2
                                   ; 8 bit data and 1 stop bit
SETBTR1                  ; starttimer
AGAIN: MOVSBUF,#'A'    ; load char 'A' inSBUF
BACK: JNB TI, BACK ; Check for transmit interrupt flag CLRTI
                         ; Clear transmit interrupt flag SJMPAGAIN
END
```

Example2.Write a program for the 8051to transfer the message 'EARTH'serially at 9600 baud,8 bit data, 1 stop bit continuously.

```
ORG 0000H LJMP
START
ORG 0030H
START: MOVTMOD,#20H                ; select timer 1 mode2
MOVTH1,#0FDH                       ; load count to get reqd. baud rate of9600
MOVSCON,#50H                       ; initialise uart in mode2
                                   ; 8 bit data and 1 stopbit
SETBTR1                            ; starttimer
LOOP: MOVA,#'E'             ; load 1st letter 'E' in a ACALLLOAD
                                   ; call load subroutine MOVA, #'A'
                            ; load 2nd letter 'A' in a ACALLLOAD
                                   ; call load subroutine MOVA,#'R'
                            ; load 3rd letter 'R' in a ACALLLOAD
                                   ; call load subroutine MOVA,#'T'
                            ; load 4th letter 'T' in a ACALLLOAD
                                   ; call load subroutine MOVA,#'H'
                            ; load 4th letter 'H' in a ACALLLOAD
                                   ; call loadsubroutine
SJMPLOOP                           ; repeatsteps

LOAD: MOV SBUF, A
HERE: JNB TI, HERE ; Check  for transmit interrupt flag CLRTI   ; Clear
                  transmit interrupt flag RET

END
```

Interrupts:

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or **Interrupt Handler**. ISR tells the processor or controller what to do when the interrupt occurs. The interrupts can be either hardware interrupts or software interrupts.

Hardware Interrupt

A hardware interrupt is an electronic alerting signal sent to the processor from an external device, like a disk controller or an external peripheral. For example, when we press a key on the keyboard or move the mouse, they trigger hardware interrupts which cause the processor to read the keystroke or mouse position.

Software Interrupt

A software interrupt is caused either by an exceptional condition or a special instruction in the instruction set which causes an interrupt when it is executed by the processor. For example, if the processor's arithmetic logic unit runs a command to divide a number by zero, to cause a divide-by-zero exception, thus causing the computer to abandon the calculation or display an error message. Software interrupt instructions work similar to subroutine calls.

What is Polling?

The state of continuous monitoring is known as **polling**. The microcontroller keeps checking the status of other devices; and while doing so, it does no other operation and consumes all its processing time for monitoring. This problem can be addressed by using interrupts.

In the interrupt method, the controller responds only when an interruption occurs. Thus, the controller is not required to regularly monitor the status (flags, signals etc.) of interfaced and inbuilt devices.

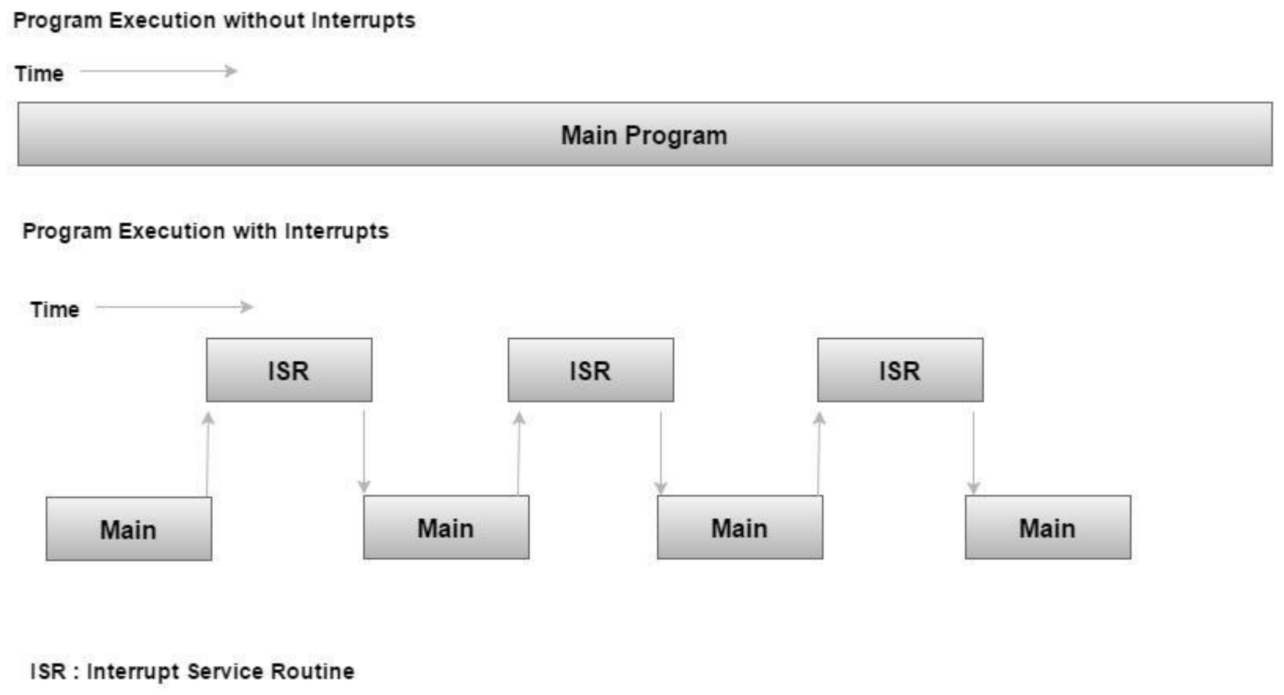
Interrupts v/s Polling

Here is an analogy that differentiates an interrupt from polling −

Interrupt	Polling
An interrupt is like a **shopkeeper**. If one needs a service or product, he goes to him and apprises him of his needs. In case of interrupts, when the flags or signals are received, they notify the controller that they need to be serviced.	The polling method is like a **salesperson**. The salesman goes from door to door while requesting to buy a product or service. Similarly, the controller keeps monitoring the flags or signals one by one for all devices and provides service to whichever component that needs its service.

Interrupt Service Routine

For every interrupt, there must be an interrupt service routine (ISR), or **interrupt handler**. When an interrupt occurs, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its interrupt service routine, ISR. The table of memory locations set aside to hold the addresses of ISRs is called as the Interrupt Vector Table.


Program Execution without Interrupts
Time
Main Program
Program Execution with Interrupts
Time
ISR
ISR
ISR
Main
Main
Main
Main
ISR : Interrupt Service Routine

Interrupt Vector Table

There are six interrupts including RESET in 8051.

Interrupts	ROM Location (Hex)	Pin
Serial COM (RI and TI)	0023	
Timer 1 interrupts(TF1)	001B	
External HW interrupt 1 (INT1)	0013	P3.3 (13)
External HW interrupt 0 (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	
Reset	0000	9

- When the reset pin is activated, the 8051 jumps to the address location 0000. This is power-up reset.
- Two interrupts are set aside for the timers: one for timer 0 and one for timer 1. Memory locations are 000BH and 001BH respectively in the interrupt vector table.
- Two interrupts are set aside for hardware external interrupts. Pin no. 12 and Pin no. 13 in Port 3 are for the external hardware interrupts INT0 and INT1, respectively. Memory locations are 0003H and 0013H respectively in the interrupt vector table.
- Serial communication has a single interrupt that belongs to both receive and transmit. Memory location 0023H belongs to this interrupt.

Steps to Execute an Interrupt

When an interrupt gets active, the microcontroller goes through the following steps −

- The microcontroller closes the currently executing instruction and saves the address of the next instruction (PC) on the stack.
- It also saves the current status of all the interrupts internally (i.e., not on the stack).
- It jumps to the memory location of the interrupt vector table that holds the address of the interrupts service routine.
- The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine, which is RETI (return from interrupt).
- Upon executing the RETI instruction, the microcontroller returns to the location where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then, it start to execute from that address.

Edge Triggering vs. Level Triggering

Interrupt modules are of two types − level-triggered or edge-triggered.

Level Triggered	Edge Triggered
A level-triggered interrupt module always generates an interrupt whenever the level of the interrupt source is asserted.	An edge-triggered interrupt module generates an interrupt only when it detects an asserting edge of the interrupt source. The edge gets detected when the interrupt source level actually changes. It can also be detected by periodic sampling and detecting an asserted level when the previous sample was de-asserted.
If the interrupt source is still asserted when the firmware interrupt handler handles the interrupt, the interrupt module will regenerate the interrupt, causing the interrupt handler to be invoked again.	Edge-triggered interrupt modules can be acted immediately, no matter how the interrupt source behaves.
Level-triggered interrupts are cumbersome for firmware.	Edge-triggered interrupts keep the firmware's code complexity low, reduce the number of conditions for firmware, and provide more flexibility when interrupts are handled.

Enabling and Disabling an Interrupt

Upon Reset, all the interrupts are disabled even if they are activated. The interrupts must be enabled using software in order for the microcontroller to respond to those interrupts.

IE (interrupt enable) register is responsible for enabling and disabling the interrupt. IE is a bitaddressable register.

Interrupt Enable Register

EA	-	ET2	ES	ET1	EX1	ET0	EX0

- **EA** − Global enable/disable.
- **-** − Undefined.
- **ET2** − Enable Timer 2 interrupt.
- **ES** − Enable Serial port interrupt.
- **ET1** − Enable Timer 1 interrupt.
- **EX1** − Enable External 1 interrupt.
- **ET0** − Enable Timer 0 interrupt.
- **EX0** − Enable External 0 interrupt.

To enable an interrupt, we take the following steps −

- Bit D7 of the IE register (EA) must be high to allow the rest of register to take effect.
- If EA = 1, interrupts will be enabled and will be responded to, if their corresponding bits in IE are high. If EA = 0, no interrupts will respond, even if their associated pins in the IE register are high.

Interrupt Priority in 8051

We can alter the interrupt priority by assigning the higher priority to any one of the interrupts. This is accomplished by programming a register called **IP** (interrupt priority).

The following figure shows the bits of IP register. Upon reset, the IP register contains all 0's. To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high.

-	-	-	-	PT1	PX1	PT0	PX0

-	IP.7	Not Implemented.
-	IP.6	Not Implemented.
-	IP.5	Not Implemented.
-	IP.4	Not Implemented.
PT1	IP.3	Defines the Timer 1 interrupt priority level.
PX1	IP.2	Defines the External Interrupt 1 priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

Interrupt inside Interrupt

What happens if the 8051 is executing an ISR that belongs to an interrupt and another one gets active? In such cases, a high-priority interrupt can interrupt a low-priority interrupt. This is known as **interrupt inside interrupt**. In 8051, a low-priority interrupt can be interrupted by a high-priority interrupt, but not by any another low-priority interrupt.

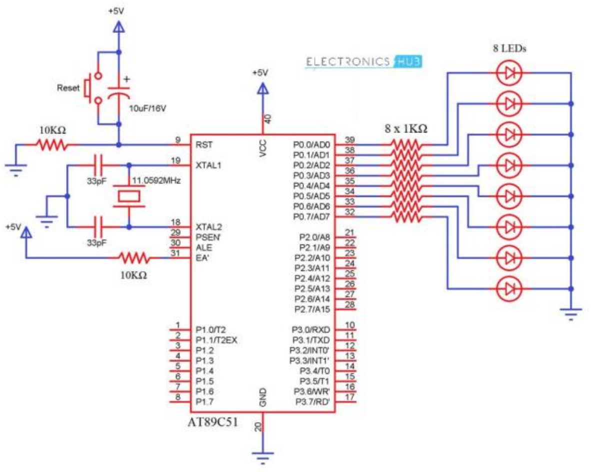
Triggering an Interrupt by Software

There are times when we need to test an ISR by way of simulation. This can be done with the simple instructions to set the interrupt high and thereby cause the 8051 to jump to the interrupt vector table. For example, set the IE bit as 1 for timer 1. An instruction **SETB TF1** will interrupt the 8051 in whatever it is doing and force it to jump to the interrupt vector table.

Interfacing with 8051

Interfacing LED with 8051 Microcontroller

Light Emitting Diodes or LEDs are the mostly commonly used components in many applications. They are made of semiconducting material. In this, It describe about basics of Interfacing LED with 8051 Microcontroller.Principle behind **Interfacing LED** with **8051.**The main principle of this circuit is to **interface LEDs** to the **8051** family micro controller. Commonly, used **LEDs** will have voltage drop of 1.7v and current of 10mA to glow at full intensity. This is applied through the output pin of the micro controller.

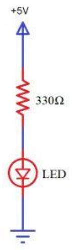


In this circuit, LEDs are connected to the port P0. The controller is connected with external crystal oscillator to pin 18 and 19 pins. Crystal pins are connected to the ground through capacitors of 33pf.

How to Control LEDs?

Light Emitting Diodes are the semi conductor light sources. Commonly used LEDs will have a cut-off voltage of 1.7V and current of 10mA. When an LED is applied with its required voltage and current it glows with full intensity.

The Light Emitting Diode is similar to the normal PN diode but it emits energy in the form of light. The color of light depends on the band gap of the semiconductor. The following figure shows "how an LED glows?"



Thus, LED is connected to the AT89C51 microcontroller with the help of a current limiting resistor. The value of this resistor is calculated using the following formula.

R= (V-1.7)/10mA, where V is the input voltage.

Generally, microcontrollers output a maximum voltage of 5V. Thus, the value of resistor calculated for this is 330 Ohms. This resistor can be connected to either the cathode or the anode of the LED.

Interfacing 16×2 LCD with 8051

We use LCD display for the messages for more interactive way to operate the system or displaying error messages etc. interfacing LCD to microcontroller is very easy if you understanding the working of LCD

LCD display is an inevitable part in almost all embedded projects and this article is about interfacing a 16×2 LCD with **8051 microcontroller**. Many guys find it hard to interface LCD module with the 8051 but the fact is that if you learn it properly, its a very easy job and by knowing it you can easily design embedded projects like digital voltmeter / ammeter, digital clock, home automation displays, status indicator display, **digital code locks**, **digital speedometer/ odometer**, display for music players etc etc. Thoroughly going through this article will make you able to display any text (including the extended characters) on any part of the 16×2 display screen. In order to understand the interfacing first you have to know about the 16×2 LCD module.

16×2 LCD module:

16×2 LCD module is a very common type of LCD module that is used in 8051 based embedded projects. It consists of 16 rows and 2 columns of 5×7 or 5×8 LCD dot matrices. The module were are talking about here is type number JHD162A which is a very popular one . It is available in a 16 pin package with back light ,contrast adjustment function and each dot matrix has 5×8 dot resolution. The pin numbers, their name and corresponding functions are shown in the table below.

Pin No:	**Name**	**Function**
1	**VSS**	This pin must be connected to the ground
2	**VCC**	Positive supply voltage pin (5V DC)
3	**VEE**	Contrast adjustment
4	**RS**	Register selection

5	**R/W**	Read or write
6	**E**	Enable
7	**DB0**	Data
8	**DB1**	Data
9	**DB2**	Data
10	**DB3**	Data
11	**DB4**	Data
12	**DB5**	Data
13	**DB6**	Data
14	**DB7**	Data
15	**LED+**	Back light LED+
16	**LED-**	Back light LED-

VEE pin is meant for adjusting the contrast of the LCD display and the contrast can be adjusted by varying the voltage at this pin. This is done by connecting one end of a POT to the Vcc (5V), other end to the Ground and connecting the center terminal (wiper) of of the POT to the VEE pin. See the circuit diagram for better understanding.

The JHD162A has two built in registers namely data register and command register. Data register is for placing the data to be displayed , and the command register is to place the commands. The 16×2 LCD module has a set of commands each meant for doing a particular job with the display. We will discuss in detail about the commands later. High logic at the RS pin will select the data register and Low logic at the RS pin will select the command register. If we make the RS pin high and the put a data in the 8 bit data line (DB0 to DB7) , the LCD module will recognize it as a data to be displayed . If we make RS pin low and put a data on the data line, the module will recognize it as a command.

R/W pin is meant for selecting between read and write modes. High level at this pin enables read mode and low level at this pin enables write mode.

E pin is for enabling the module. A high to low transition at this pin will enable the module.

DB0 to DB7 are the data pins. The data to be displayed and the command instructions are placed on these pins.

LED+ is the anode of the back light LED and this pin must be connected to Vcc through a suitable series current limiting resistor. LED- is the cathode of the back light LED and this pin must be connected to ground.

16×2 LCD module commands:

16×2 LCD module has a set of preset command instructions. Each command will make the module to do a particular task. The commonly used commands and their function are given in the table below.

Command	Function
0F	LCD ON, Cursor ON, Cursor blinking ON
01	Clear screen
02	Return home
04	Decrement cursor
06	Increment cursor
0E	Display ON ,Cursor blinking OFF
80	Force cursor to the beginning of 1st line
C0	Force cursor to the beginning of 2nd line
38	Use 2 lines and 5×7 matrix

83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, Cursor OFF
C1	Jump to second line, position1
OC	Display ON, Cursor OFF
C1	Jump to second line, position1
C2	Jump to second line, position2

LCD initialization:

The steps that has to be done for initializing the LCD display is given below and these steps are common for almost all applications.

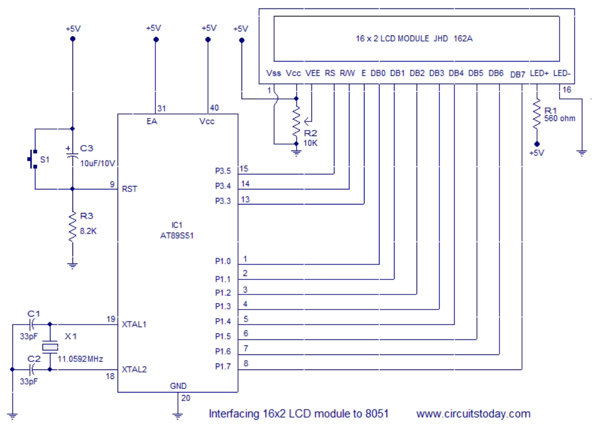
- Send 38H to the 8 bit data line for initialization
- Send 0FH for making LCD ON, cursor ON and cursor blinking ON.
- Send 06H for incrementing cursor position.
- Send 01H for clearing the display and return the cursor.

Sending data to the LCD.

The steps for sending data to the LCD module is given below. I have already said that the LCD module has pins namely RS, R/W and E. It is the logic state of these pins that make the module to determine whether a given data input is a command or data to be displayed.

- Make R/W low.
- Make RS=0 if data byte is a command and make RS=1 if the data byte is a data to be displayed.
- Place data byte on the data register.
- Pulse E from high to low.
- Repeat above steps for sending another data.

Circuit diagram:


16 x 2 LCD MODULE JHD 162A
Vss Vcc VEE RS R/W E DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7 LED+ LED-
+5V
R1 560 ohm
R2 10K
C3 10uF/10V
S1
R3 8.2K
IC1 AT89S51
EA Vcc RST XTAL1 XTAL2 GND
P3.5 P3.4 P3.3 P1.0 P1.1 P1.2 P1.3 P1.4 P1.5 P1.6 P1.7
C1 33pF
C2 33pF
X1 11.0592MHz
Interfacing 16x2 LCD module to 8051
www.circuitstoday.com

The circuit diagram given above shows how to interface a 16×2 LCD module with AT89S1 microcontroller. Capacitor C3, resistor R3 and push button switch S1 forms the reset circuitry. Ceramic capacitors C1,C2 and crystal X1 is related to the clock circuitry which produces the system clock frequency. P1.0 to P1.7 pins of the microcontroller is connected to the DB0 to DB7 pins of the module respectively and through this route the data goes to the LCD module. P3.3, P3.4 and P3.5 are connected to the E, R/W, RS pins of the microcontroller and through this route the control signals are transffered to the LCD module. Resistor R1 limits the current through the back light LED and so do the back light intensity. POT R2 is used for adjusting the contrast of the display. Program for interfacing LCD to 8051 microcontroller is shown below.

Program:

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

```
ACALL DISP
MOV A,#68D
ACALL DISP

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

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

HERE: SJMP HERE

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

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

DELY: CLR P3.3
CLR P3.5
SETB P3.4
```

```
MOV P1,#0FFh
SETB P3.3
MOV A,P1
JB ACC.7,DELY

CLR P3.3
CLR P3.4
RET

END
```

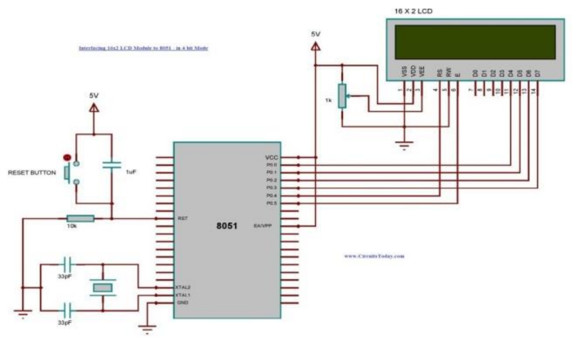
Subroutine CMND sets the logic of the RS, R/W, E pins of the LCD module so that the module recognizes the input data (given to DB0 to DB7) as a command.Subroutine DISP sets the logic of the RS, R/W, E pins of the module so that the module recognizes the input data as a data to be displayed .

Interfacing LCD Module to 8051 in 4 Bit Mode (using only 4 pins of a port)

The microcontroller like 8051 has only limited number of GPIO pins (GPIO – general purpose input output). So to design complex projects we need sufficient number of I/O pins . An LCD module can be interfaced with a microcontroller either in **8 bit mode (**as seen above**)** or in 4 bit mode. 8 bit mode is the conventional mode which uses 8 data lines and RS, R/W, E pins for functioning. However 4 bit mode uses only 4 data lines along with the control pins. This will saves the number of GPIO pins needed for other purpose.

Objectives

- Interface an LCD with 8051 in 4 bit mode
- Use a single port of the microcontroller for both data and control lines of the LCD.



LCD Module to 8051 – 4 Bit Mode

As shown in the circuit diagram, port 0 of the controller is used for interfacing it with LCD module. In 4 bit mode only 4 lines D4-D7, along with RS, R/W and E pins are used. This will save us 4 pins of our controller which we might employ it for other purpose. Here we only need to write to the LCD module. So the R/W pin can be ground it as shown in the schematic diagram. In this way the total number of pins can be reduced to 6. In 4 Bit mode the data bytes are split into two four bits and are transferred in the form of a nibble. The data transmission to a LCD is performed by assigning logic states to the control pins RS and E. The reset circuit, oscillator circuit and power supply need to be provided for the proper working of the circuit.

Program – Interface LCD Module to 8051 – 4 Bit Mode

```
RS EQU P0.4
EN EQU P0.5
PORT EQU P0
```

```
U EQU 30H
L EQU 31H
ORG 000H

MOV DPTR,#INIT_COMMANDS
ACALL LCD_CMD
MOV DPTR,#LINE1
ACALL LCD_CMD
MOV DPTR,#TEXT1
ACALL LCD_DISP
MOV DPTR,#LINE2
ACALL LCD_CMD
MOV DPTR,#TEXT2
ACALL LCD_DISP
SJMP $


SPLITER: MOV L,A
ANL L,#00FH
SWAP A
ANL A,#00FH
MOV U,A
RET

MOVE: ANL PORT,#0F0H
ORL PORT,A
SETB EN
ACALL DELAY
CLR EN
ACALL DELAY
RET


LCD_CMD: CLR A
MOVC A,@A+DPTR
JZ EXIT2
INC DPTR
CLR RS
ACALL SPLITER
MOV A,U
ACALL MOVE
MOV A,L
ACALL MOVE
SJMP LCD_CMD
EXIT2: RET

LCD_DATA: SETB RS
ACALL SPLITER
MOV A,U
ACALL MOVE
MOV A,L
ACALL MOVE
RET

LCD_DISP: CLR A
```

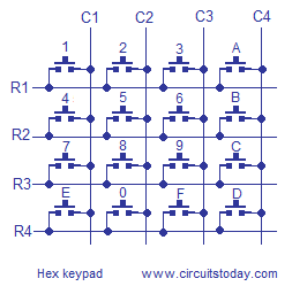
```
MOVC A,@A+DPTR
JZ EXIT1
INC DPTR
ACALL LCD_DATA
SJMP LCD_DISP
EXIT1: RET

DELAY: MOV R7, #10H
L2: MOV R6,#0FH
L1: DJNZ R6, L1
DJNZ R7, L2
RET

INIT_COMMANDS: DB 20H,28H,0CH,01H,06H,80H,0
LINE1: DB 01H,06H,06H,80H,0
LINE2: DB 0C0H,0
CLEAR: DB 01H,0

TEXT1: DB " CircuitsToday ",0
TEXT2: DB "4bit Using 1Port",0

END
```

Interfacing hex keypad to 8051

Interfacing a hex key pad to 8051 microcontroller. A clear knowledge on interfacing hex key pad to 8051 is very essential while designing embedded system projects which requires character or numeric input or both. For example projects like digital code lock, numeric calculator etc. Before going to the interfacing in detail, let's have a look at the hex keypad.

Hex keypad

Hex key pad is essentially a collection of 16 keys arranged in the form of a 4×4 matrix. Hex key pad usually have keys representing numerics 0 to 9 and characters A to F. The simplified diagram of a typical hex key pad is shown in the figure below.

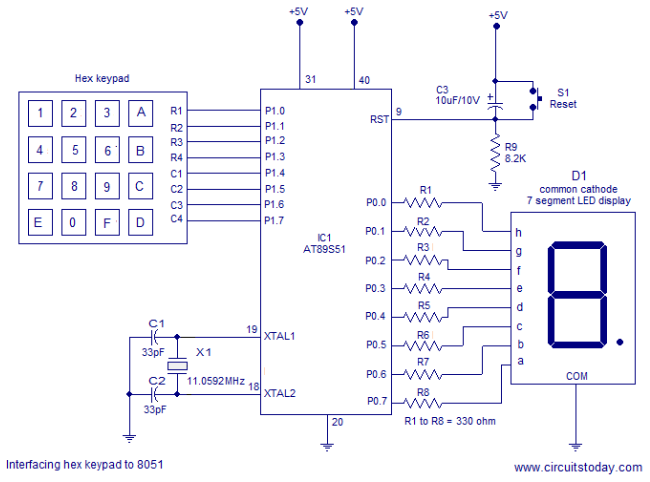


Hex keypad

The hex keypad has 8 communication lines namely R1, R2, R3, R4, C1, C2, C3 and C4. R1 to R4 represents the four rows and C1 to C4 represents the four columns. When a particular key is pressed the corresponding row and column to which the terminals of the key are connected gets shorted. For example if key 1 is pressed row R1 and column C1 gets shorted and so on. The program identifies which key is pressed by a method known as column scanning. In this method a particular row is kept low (other rows are kept high) and the columns are checked for low. If a particular column is found low then that means that the key connected between that column and the corresponding row (the row that is kept low) is been pressed. For example if row R1 is initially kept low and column C1 is found low during scanning, that means key 1 is pressed.

Interfacing hex keypad to 8051:

The circuit diagram for demonstrating interfacing hex keypad to 8051 is shown below.Like previous 8051 projects, AT89S51 is the microcontroller used here. The circuit will display the character/numeric pressed on a seven segment LED display. The circuit is very simple and it uses only two ports of the microcontroller, one for the hex keypad and the other for the seven segment LED display.



Interfacing hex keypad to 8051

The hex keypad is interfaced to port 1 and seven segment LED display is interfaced to port 0 of the microcontroller. Resistors R1 to R8 limits the current through the corresponding segments of the LED display. Capacitors C1, C2 and crystal X1 completes the clock circuitry for the microcontroller. Capacitor C3, resistor R9 and push button switch S1 forms a debouncing reset mechanism.

Program:

```
ORG 00H
MOV DPTR,#LUT // moves starting address of LUT to DPTR
MOV A,#11111111B // loads A with all 1's
MOV P0,#00000000B // initializes P0 as output port

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

```
NEXT5:JB P1.5,NEXT6
      MOV A,#5D
      ACALL DISPLAY
NEXT6:JB P1.6,NEXT7
      MOV A,#6D
      ACALL DISPLAY
NEXT7:JB P1.7,NEXT8
      MOV A,#7D
      ACALL DISPLAY
NEXT8:SETB P1.1
      CLR P1.2
      JB P1.4,NEXT9
      MOV A,#8D
      ACALL DISPLAY
NEXT9:JB P1.5,NEXT10
      MOV A,#9D
      ACALL DISPLAY
NEXT10:JB P1.6,NEXT11
       MOV A,#10D
       ACALL DISPLAY
NEXT11:JB P1.7,NEXT12
       MOV A,#11D
       ACALL DISPLAY
NEXT12:SETB P1.2
       CLR P1.3
       JB P1.4,NEXT13
       MOV A,#12D
       ACALL DISPLAY
NEXT13:JB P1.5,NEXT14
       MOV A,#13D
       ACALL DISPLAY
NEXT14:JB P1.6,NEXT15
       MOV A,#14D
       ACALL DISPLAY
NEXT15:JB P1.7,BACK
       MOV A,#15D
       ACALL DISPLAY
       LJMP BACK

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

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

```
    DB 10011110B
    DB 11111100B
    DB 10001110B
    DB 01111010B
    END
```

About the program:

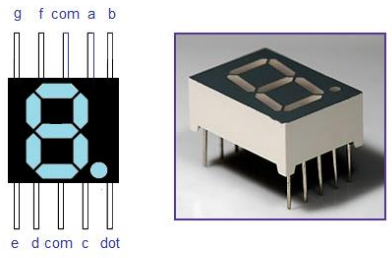
Firstly the program initializes port 0 as an output port by writing all 0's to it and port 1 as an input port by writing all 1's to it. Then the program makes row 1 low by clearing P1.0 and scans the columns one by one for low using JB instruction.If column C1 is found low, that means 1 is pressed and accumulator is loaded by zero and DISPLAY subroutine is called. The display subroutine adds the content in A with the starting address of LUT stored in DPTR and loads A with the data to which the resultant address points (using instruction MOVC A,@A+DPTR). The present data in A will be the digit drive pattern for the current key press and this pattern is put to Port 0 for display. This way the program scans for each key one by one and puts it on the display if it is found to be pressed.

Notes:

- The 5V DC power supply must be well regulated and filtered.
- Column scanning is not the only method to identify the key press. You can use row scanning also. In row scanning a particular column is kept low (other columns are kept high) and the rows are tested for low using a suitable branching instruction. If a particular row is observed low then that means that the key connected between that row and the corresponding column (the column that is kept low) is been pressed. For example if column C1 is initially kept low and row R1 is observed low during scanning, that means key 1 is pressed.
- A membrane type hex keypad was used during the testing. Push button switch type and dome switch type will also work. I haven't checked other types.
- The display used was a common cathode seven segment LED display with type number ELK5613A. This is just for information and any general purpose common cathode 7 segment LED display will work here.

Interfacing Seven segment display to 8051

Interface a seven segment LED display to an 8051 microcontroller. 7 segment LED display is very popular and it can display digits from 0 to 9 and quite a few characters like A, b, C, ., H, E, e, F, n, o,t,u,y, etc. Knowledge about how to interface a seven segment display to a micro controller is very essential in designing embedded systems. A seven segment display consists of seven LEDs arranged in the form of a squarish **'8'** slightly inclined to the right and a single LED as the dot character. Different characters can be displayed by selectively glowing the required LED segments. Seven segment displays are of two types, ***common cathode and common anode.*** In common cathode type , the cathode of all LEDs are tied together to a single terminal which is usually labeled as '**com**' and the anode of all LEDs are left alone as individual pins labeled as a, b, c, d, e, f, g & h (or dot) . In common anode type, the anode of all LEDs are tied together as a single terminal and cathodes are left alone as individual pins. The pin out scheme and picture of a typical 7 segment LED display is shown in the image below.



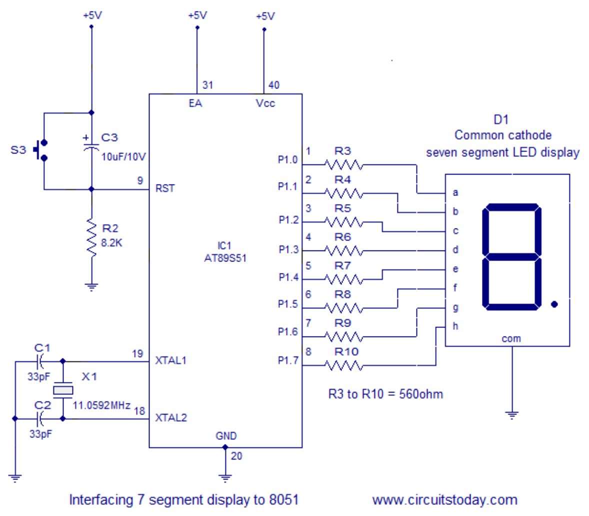
7 segment LED display

Digit drive pattern:

Digit drive pattern of a seven segment LED display is simply the different logic combinations of its terminals **'a' to 'h'** in order to display different digits and characters. The common digit drive patterns (0 to 9) of a seven segment display are shown in the table below.

Digit	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Interfacing seven segment display to 8051:



Interfacing 7 segment display to 8051

The circuit diagram shown above is of an AT89S51 microcontroller based 0 to 9 counter which has a 7 segment LED display interfaced to it in order to display the count. This simple circuit illustrates two things. How to setup simple 0 to 9 up counter using 8051 and more importantly how to interface a seven segment LED display to 8051 in order to display a particular result. The common cathode seven segment display D1 is connected to the Port 1 of the microcontroller (AT89S51) as shown in the circuit diagram. R3 to R10 are current limiting resistors. S3 is the reset switch and R2,C3 forms a debouncing circuitry. C1, C2 and X1 are related to the clock circuit. The software part of the project has to do the following tasks.

- Form a 0 to 9 counter with a predetermined delay (around 1/2 second here).
- Convert the current count into digit drive pattern.
- Put the current digit drive pattern into a port for displaying.

All the above said tasks are accomplished by the program given below.

Program:

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

About the program:

Instruction MOVC A,@A+PC is the instruction that produces the required digit drive pattern for the display. Execution of this instruction will add the value in the accumulator A with the content of the program counter(address of the next instruction) and will move the data present in the resultant address to A. After this the program resumes from the line after MOVC A,@A+PC.

In the program, initial value in A is 00001001B. Execution of MOVC A,@A+PC will add oooo1001B to the content in PC (address of next instruction). The result will be the address of label DB 3FH (line15) and the data present in this address ie 3FH (digit drive pattern for 0) gets moved into the accumulator. Moving this pattern in the accumulator to Port 1 will display 0 which is the first count.

At the next count, value in A will advance to 00001010 and after the execution of MOVC A,@+PC ,the value in A will be 06H which is the digit drive pattern for 1 and this will display 1 which is the next count and this cycle gets repeated for subsequent counts.

The reason why accumulator is loaded with 00001001B (9 in decimal) initially is that the instructions from line 9 to line 15 consumes 9 bytes in total.

The lines 15 to 24 in the program which starts with label DB can be called as a **Look Up Table (LUT)**. label DB is known as Define Byte – which defines a byte. This table defines the digit drive patterns for 7 segment display as bytes (in hex format). MOVC operator fetches the byte from this table based on the result of adding PC and contents in the accumulator.

Register B is used as a temporary storage of the initial value of the accumulator and the subsequent increments made to accumulator to fetch each digit drive pattern one by one from the look up table(LUT).

Note:- In line 6, Accumulator is incremented by 1 each time (each loop iteration) to select the next digit drive pattern. Since MOVC operator uses the value in A to fetch the digit drive pattern from LUT, value in ACC has to be incremented/manipulated accordingly. The digit drive patterns are arranged consecutively in LUT.

Register R0 is used as a counter which counts from 10 down to 0. This ensures that digits from o to 9 are continuously displayed in the 7 segment LED. You may note lines 4, 11, 12, and 13 in the above program. Line 4 initializes R0 to 10 (OAh). When the program counter reaches line 11 for the first time, 7 segment LED has already displayed 0. So we can reduce one count and that is why we have written DEC Ro. We need to continuously check if R0 has reached full count (that is 0). In order to do that lines 12 and 13 are used. We move R0 to accumulator and then use the Jump if Zero (JZ) instruction to check if accumulator has reached zero. If Acc=0, then we makes the program to jump to START (initial state) and hence we restart the 7 segment LED to display from 0 to 9 again. If Acc not equal to zero, we continue the program to display the next digit (check line 14).

Interfacing Stepper Motor with 8051

 The stepper motors coil A,B,C,D is connected to the port 1 i.e. to P1.0, P1.2, P1.2 and P1.3.

 The Microcontroller does not provide sufficient current to drive motor and to safeguard 8081 from loading effect and burn out condition, a motor driver IC ULN 2003 between 8051 and stepper motor. ULN 2003 is a stepper motor driver.

- The stepper motor is user for controlling:

  1. Position control
  2. Direction control &
  3. Speed control

➢ Calculations:

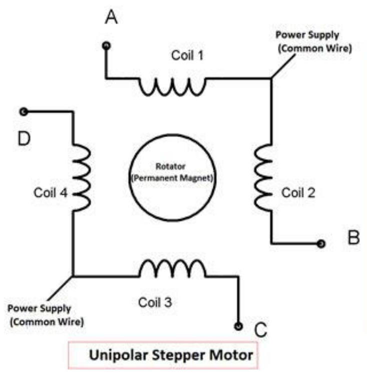
1. Total no. of steps= $\frac{Total\ rotational\ angle}{Step\ Angle}$

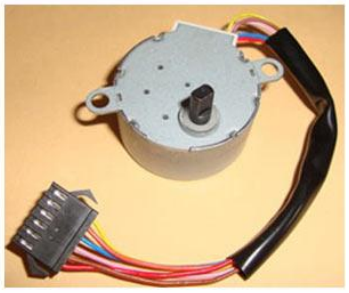
Ex: $\frac{360}{1.8}$ = 200 steps are required to complete one rotation

2. Total no. of repeated steps= $\frac{Total\ no.of\ steps}{Step\ sequence}$

Ex: $\frac{200}{4}$ = 50 repetition of sequence = (32) in Hexadecimal.

Stepper motors are basically two types: Unipolar and Bipolar. **Unipolar stepper** motor generally has five or six wire, in which four wires are one end of four stator coils, and other end of the all four coils is tied together which represents fifth wire, this is called common wire (common point). Generally there are two common wire, formed by connecting one end of the two-two coils as shown in below figure. Unipolar stepper motor is very common and popular because of its ease of use.


A
Coil 1
Power Supply (Common Wire)
D
Rotator (Permanent Magnet)
Coil 4
Coil 2
B
Coil 3
Power Supply (Common Wire)
C
Unipolar Stepper Motor

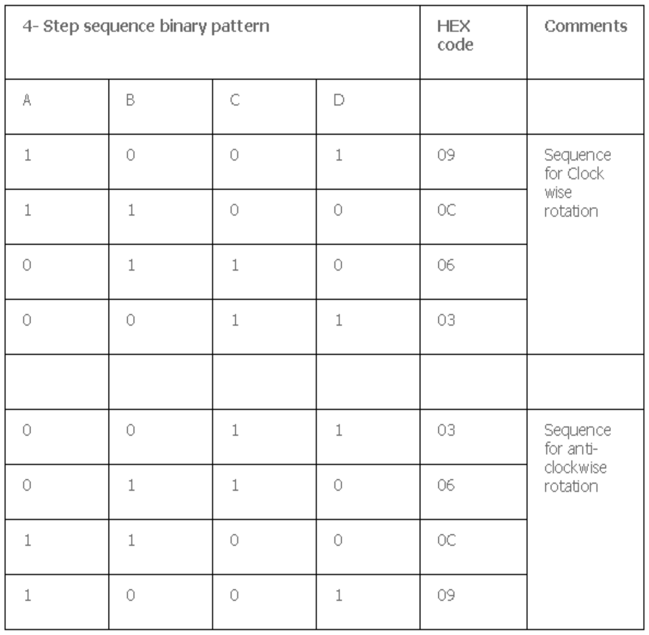


In **Bipolar stepper** motor there is just four wires coming out from two sets of coils, means there are no common wire.

Stepper motor is made up of a stator and a rotator. Stator represents the four electromagnet coils which remain stationary around the rotator, and rotator represents permanent magnet which rotates. Whenever the coils energised by applying the current, the electromagnetic field is created, resulting the rotation of rotator (permanent magnet). Coils should be energised in a particular sequence to make the rotator rotate. On the basis of this "sequence" we can divide the working method of **Unipolar stepper motor** in three modes: Wave drive mode, full step drive mode and half step drive mode.

4-Step sequence (Full Drive mode)**:**

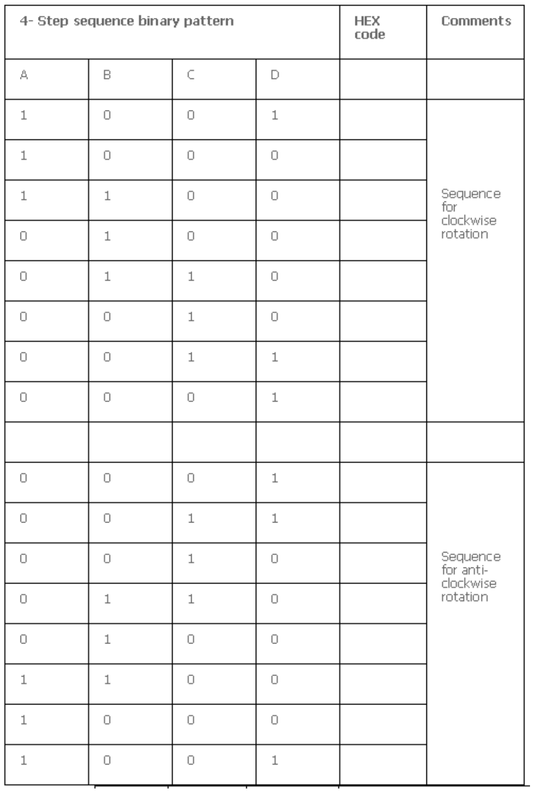
- In this type of functioning, the following 4 binary sequence/code are used for rotation: (Considering step angle= 1.8 degrees)



4- Step sequence binary pattern				HEX code	Comments
A	B	C	D		
1	0	0	1	09	Sequence for Clock wise rotation
1	1	0	0	0C	
0	1	1	0	06	
0	0	1	1	03	
0	0	1	1	03	Sequence for anti-clockwise rotation
0	1	1	0	06	
1	1	0	0	0C	
1	0	0	1	09	

8-Step Sequence(Half Drive mode:**):**

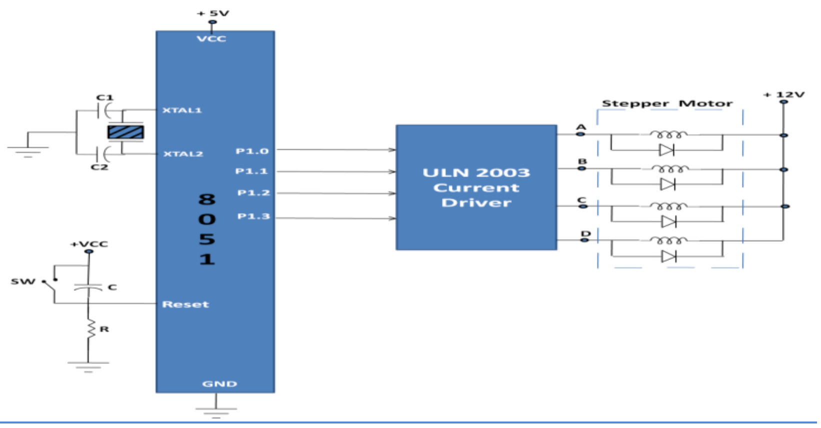
- In this type of functioning, the following 8 binary sequence/code are used for rotation: (Considering step angle= 0.9degrees)



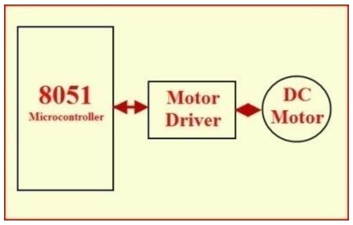
4- Step sequence binary pattern				HEX code	Comments
A	B	C	D		
1	0	0	1		Sequence for clockwise rotation
1	0	0	0		
1	1	0	0		
0	1	0	0		
0	1	1	0		
0	0	1	0		
0	0	1	1		
0	0	0	1		
0	0	0	1		Sequence for anti-clockwise rotation
0	0	1	1		
0	0	1	0		
0	1	1	0		
0	1	0	0		
1	1	0	0		
1	0	0	0		
1	0	0	1		

Program

LABEL	OPCODE	OPERAND	COMMENT
	ORG	0000H	
	MOV	A,99#	
	MOV	R0,#200	
BACK:	MOV	P1,A	
	ACALL	DELAY	
	RR	A	
	DJNZ	R0,BACK	
HERE:	SJMP	HERE	
DELAY:	MOV	R2,#225	
L2:	MOV	R3,#225	
L1:	DJNZ	R3,L1	
	DJNZ	R2,L2	
	RET		
	END		



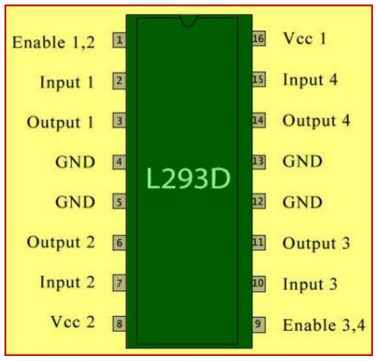
Interfacing DC Motor with 8051 Microcontroller



The main purpose of DC interfacing with 8051 microcontroller is for controlling the speed of the motor. The DC motor is an electrical machine with a rotating part termed as a rotor which has to be controlled. For example, consider the DC motor whose speed or direction of rotation of DC motor can be controlled using programming techniques which can be achieved by interfacing with 8051 microcontroller. So, in this article let us discuss about interfacing DC motor with 8051 microcontroller.

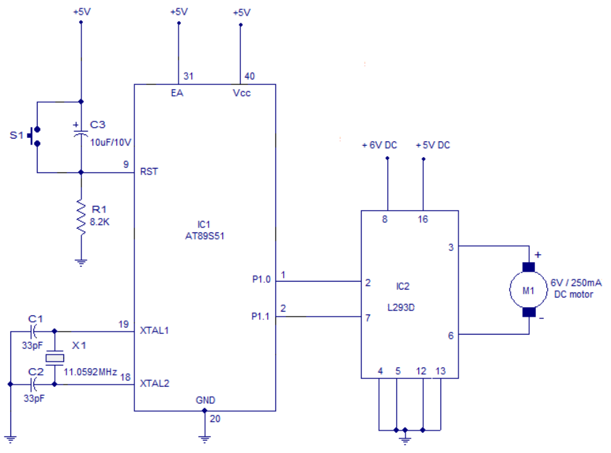
Motor Driver IC used for Interfacing DC motor with 8051

Here, interfacing 8051 with DC motor requires a motor driver. There are various types of driver ICs among which L293D is typically used for interfacing DC motor with 8051. L293 is an IC with 16 pins which are represented in the figure below.



Motor Driver IC L293D used for Interfacing DC

This L293 IC is having ratings of 600mA per channel and DC supply voltage in the range of 4.5V to 36V. These ICs can be protected from inductive spikes by connecting higher speed clamp diodes internally. This 16 pin L293D IC can be used for controlling the direction of two DC motors. The IC L293D works based on the H-bridge concept. The voltage can be made to flow in either direction using this circuit (H-bridge) such that by changing the voltage direction the motor direction can be changed.



Bi directional DC motor using 8051.

This describes a bidirectional DC motor that changes its direction automatically after a preset amount of time (around 1S). AT89S51 is the microcontroller used here and L293 forms the motor driver. Circuit diagram is shown above

In the circuit components R1, S1 and C3 forms a debouncing reset circuitry. C1, C2 and X1 are related to the oscillator. Port pins P1.0 and P1.1 are connected to the corresponding input pins of the L293 motor driver. The motor is connected across output pins 3 and 6 of the L293. The software is so written that the logic combinations of P1.0 and P1.1 controls the direction of the motor. Initially when power is switched ON, P1.0 will be high and P1.1 will be low. This condition is maintained for a preset amount of time (around 1S) and for this time the motor will be running in the clockwise direction (refer the function table of

L293). Then the logic of P1.0 and P1.1 are swapped and this condition is also maintained for the same duration . This makes the motor to run in the anti clockwise direction for the same duration and the entire cycle is repeated.

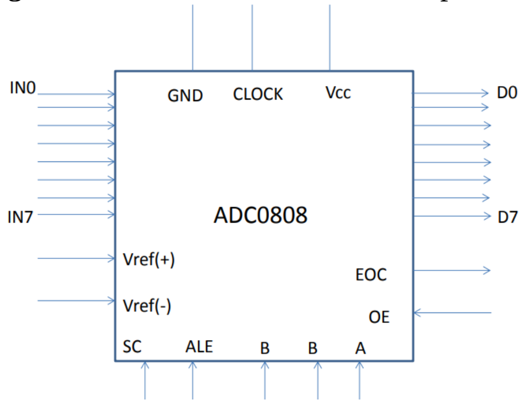
Program:

```
ORG 00H // initial starting address
MAIN: MOV P1,#00000001B // motor runs clockwise
ACALL DELAY // calls the 1S DELAY
MOV P1,#00000010B // motor runs anti clockwise
ACALL DELAY // calls the 1S DELAY
SJMP MAIN // jumps to label MAIN for repaeting the cycle
DELAY: MOV R4,#0FH
WAIT1: MOV R3,#00H
WAIT2: MOV R2,#00H
WAIT3: DJNZ R2,WAIT3
DJNZ R3,WAIT2
DJNZ R4,WAIT1
RET
END
```

Notes*:*
The maximum current capacity of L293 is 600mA/channel. So do not use a motor that consumes more than that.
The supply voltage range of L293 is between 4.5 and 36V DC. So you can use a motor falling in that range.

Interfacing ADC with 8051:

ADC 0808:
The ADC used in the interfacing is ADC 0808.It is successive approximation 8-bit ADC. It has 28 pins, and can handle upto 8 analog signals using one chip. It has got an 8-bit data output. The 8 input channels are IN0-IN7, and Vref(+)=5V; Vref(-) has been grounded. In order to select the inputs IN0-IN7; A, B and C addresses are used.



ADC receives analog signal from the source. This analog signal is received from one of the 8 input channels of ADC0808. Then this signal is processed accordingly and converted to corresponding digital signal. This signal is then sent to the microcontroller and the output is displayed using Light Emitting Diode (LED)

ADC
8051 microcontroller
LED

ALGORITHM FOR PROGRAMMING ADC
i) An analog channel is selected by giving bits to A, B, C addresses.
ii) ALE(Address Latch Enable) is activated by a low to high pulse in order to latch in the address.
iii) SC(Start Conversion) is activated by a low to high pulse in order to start the conversion.

iv) If a high to low output is obtained at EOC(End of Conversion), it indicates that the data conversion is finished and the data is ready.
v) OE(Output Enable) is activated to read output data from the ADC chip. In order to bring the digital data out of the chip a low to high pulse is is given to the OE pin.
ASSEMBLY PROGRAM TO INTERFACE ADC WITH 8051

```
ORG 000AH
SJMP MAIN
ADC_DATA EQU P1 ;Give Name To Port Pins
 ADC_SC BIT P3.0
 ADC_EOC BIT P3.1 ADC 8051 microcontroller LED 9
 ADC_ALE BIT P3.2
 ADC_OE BIT P3.3
ADD_A BIT P3.4
 ADD_B BIT P3.5
 ADD_C BIT P3.6
 MAIN: MOV ADC_DATA,#0FFH ;Port 1 is input port
 SETB ADD_A ;select channel
 SETB ADD_B
CLR ADD_C ;for channel 3 selection
ACALL DELAY1
ACALL ADC_COUNT
MOV P0,A ;
ADC Programming Start
ADC_COUNT: SETB ADC_EOC ;it is made as input Port
 CLR ADC_ALE
CLR ADC_SC
CLR ADC_OE
 BACK:  SETB ADC_ALE ;High To Low Pulse is given to ALE
ACALL DELAY1
 SETB ADC_SC ;High To Low Pulse is given to SC
 ACALL DELAY1
 CLR ADC_ALE
 CLR ADC_SC
 LOOP1: JB ADC_EOC,LOOP1 ;Wait for conversion to finish
LOOP2: JNB ADC_EOC,LOOP2 ;Output becomes high
            SETB ADC_OE ;Set OE High to covert data on controller
            ACALL DELAY1 ;For Further delay
             CLR ADC_OE ;digital converted data is saved in memory
             MOV B,#05H DIV AB ;amplify with gain in place of 05H for obtaining real digital data
             RET ;Return To Main Routine Delay ;
            App. 1.3643 Sec. Delay
             DELAY: MOV R3,#3
             LOOP3: MOV R1,#254
            LOOP4: MOV R2,#254
             LOOP5: DJNZ R2,LOOP5
                         DJNZ R1,LOOP4
                        DJNZ R3,LOOP3
                         RET ;Approximately 435 µsec
                         DELAY1: MOV R3,#1
                        LOOP6: MOV R1,#10
                         LOOP7: MOV R2,#10
```

```
LOOP8: DJNZ R2,LOOP8
DJNZ R1,LOOP7
 DJNZ R3,LOOP6
 RET
END
```

Interfacing DAC to 8051:

The digital to analog converter is a device widely used to convert digital pulses to analog signals. The two methods of creating DAC are binary weighted and R-2R ladder. DAC 0808 uses the R-2R method since it can achieve a high degree of precision. The first criterion for judging a DAC is its resolution, which is the function of the number of binary inputs. The common ones are 8, 10 and 12 bits. The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is equal to 2n, where n is the number of data inputs. DAC 0808 provides 256 discrete voltage or current levels of output. In DAC 0808, the digital inputs are converted into current Iout and by connecting a resistor to Iout pin, we convert the result to voltage. The total current provided by IOUT pin is a function of binary numbers at the D0-D7 pins inputs to DAC 0808 and reference current (Iref) is as follows: Iout=Iref (D7/2+D6/4+D5/8+D4/16+D3/32...+D0/256) Where D0 is the LSB, D7 is the MSB for the inputs and Iref is the input current that must be applied.

Algorithm for interface 8051 with DAC:

Step1: Connect the P1 of 8051 with D0-D7 pins of DAC

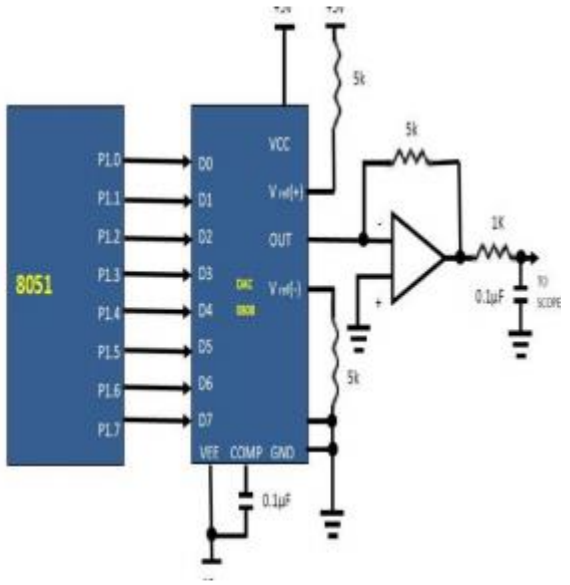
Step2: Give +5v to VCC & Vref of DAC

Step3: Connect -12v to VEE of DAC

Step4: Connect OPAMP to OUT pin of the DAC With 5K resistor

Step5: Connect the oscilloscope to the OPAMP to View the output

Digital to Analog converters are required when a digital code must be converted to analog signal. It has eight digital input lines and an output line for analog signal. The number of data bits reduces resolution of DAC. Outputting digital data 00 to FF at regular intervals to DAC, results in generation of different waveforms namely square wave, triangular wave, sine wave etc.



Interfacing diagram of DAC to 8051