Microprocessor

Course Title: Microprocessor
Course No: CSC162
Nature of the Course: Theory + Lab
Semester: II

Full Marks: 60 + 20 + 20
Pass Marks: 24 + 8 + 8
Credit Hrs: 3

Course Description: This course contains fundamental concepts of Microprocessor operations, basic I/O interfaces and Interrupts operations.

Course Objectives: The course objective is to introduce the operation, programming and application of microprocessor.

Course Details

Unit 1: Introduction **(4 Hrs.)**

- Definition of microprocessor and its application
- Evolution of microprocessor, Von Neumann and Harvard architecture
- Components of microprocessor
  - Microprocessor: Arithmetic and Logic Unit (ALU), Control Unit (CU), Registers
  - Memory
  - Input / Output
- System Bus: Data , Address and Control Bus
- Microprocessor with Bus Organization

Unit 2: Basic Computer Architecture **(7 Hrs.)**

- 8085 Microprocessor Architecture and Operations
  - Address, Data And Control Buses
  - Externally Initiated Operations
  - Memory and Memory Operations
  - 8085 Pin Diagram and Functions
  - Multiplexing and De-multiplexing of address/data bus
  - Generation Of Control Signals
  - Internal Data Operation and Registers
  - Addressing Modes
  - Flag and Flag Register
- 8086 Microprocessor
  - Logical Block Diagram
  - Memory Segmentation
  - Pipelining
  - Segment Registers,
  - Bus Interface Unit and Execution Unit

Unit 3: Instruction Cycle **(3 Hrs.)**

- Instruction Cycle, Machine Cycle and T-states
  - Machine Cycle of 8085 Microprocessor: op-code fetch, memory read, memory write, I/O read, I/O write, interrupt
  - Fetch and Execute Operation, Timing Diagram
  - Timing Diagram of MOV, MVI, IN, OUT, LDA, STA
  - Memory Interfacing and Generation of Chip Select Signal

Unit 4: Assembly Language Programming **(10 Hrs.)**

- Programming with Intel 8085 Microprocessor
  - Instruction and Data Format
  - Mnemonics and Operands
  - Instruction Sets
  - Data Transfer:- MOV, IN, OUT, STA,LDA, LXI, LDAX, STAX, XCHG
  - Arithmetic and Logic:- ADD, SUB, INR, DCR, AND, OR, XOR, CMP, RLC, RRC, RAL, RAR
  - Branching:- JMP, JNZ, JZ, JNC, JC, CALL
  - Stack:- PUSH, POP
  - Multiplication and Division
  - Simple Sequence Programs, Branching, Looping
  - Array(Sorting) and Table Processing
  - Decimal to BCD Conversion
- Programming with Intel 8086 microprocessor
  - Macro Assembler
  - Assembling and Linking
  - Assembler Directives, Comments

- Instructions: LEA, MUL, DIV, LOOP, AAA, DAA
- INT 21H Functions
- 01H, 02H, 09H, 0AH, 4CH
- INT 10H Functions (Introduction Only)
- 00H, 01H, 02H, 06H, 07H, 08H, 09H, 0AH
- Simple String and Character Manipulation Programs
- Debugging

(6 Hrs.)

Unit 5: Basic I/O, Memory R/W and Interrupt Operations

- Memory mapped I/O, I/O Mapped I/O and Hybrid I/O
- Direct Memory Access (DMA)
- Introduction, Advantage and Application
- 8237 DMA Controller and Interfacing

❑ Interrupt
- 8085 Interrupt Pins and Priority
- Maskable and Non-maskable Interrupts
- RST Instructions
- Vector and Polled Interrupt

❑ 8259 Interrupt Controller
- Block Diagram and Explanation
- Priority Modes and Additional Features

(6 Hrs.)

Unit 6: Input/ Output Interfaces

❑ Parallel Communication – Introduction and Applications

❑ Serial Communication
- Introduction and Applications
- Introduction to Programmable Communication Interface 8251
- Basic Concept of Synchronous and Asynchronous Modes

❑ Simple I/O, Strobe I/O, Single handshake I/O, Double handshake I/O

❑ 8255 A and it's Working
- Block Diagram
- Modes of Operation
- Control Word

❑ RS-232 – Introduction, Pin Configuration (9 pin and 25 pin) and function of each pin, Interconnection between DTE-DTE and DTE-DCE

(9 Hrs.)

Unit 7: Advanced Microprocessors

❑ 80286: Architecture (Block Diagram) , Registers, (Real/Protected mode), Privilege Levels, Descriptor Cache, Memory Access in GDT and LDT, Multitasking, Addressing Modes, Flag Register

❑ 80386: Architecture (Block Diagram), Register organization, Memory Access in Protected Mode, Paging (Up to LA to PA)

Laboratory Works:

The laboratory work includes Assembly language programming using 8085/8086/8088 trainer kit. The programming should include: Arithmetic operation, base conversion, conditional branching etc. The lab work list may include following concepts:

1. Assembly language program using 8085 microprocessor kit and 8085 microprocessor simulator.
2. Use of all types of instructions and addressing modes.
3. Program including basic arithmetical, logical, looping, bitwise and branching.
4. Assembly language programming using 8086 microprocessor emulator, using any types of Assembler, including the different functions of 21H.

Text Books:

1. Ramesh S.Gaonkar, Microprocessor Architecture, Programming, and Applications with 8085, Prentice Hall

Reference Books:

1. A.P.Malvino and J.A.Brown, Digital Computer Electronics, 3rd Edition, Tata McGraw Hill D.V.Hall, Microprocessors and Interfacing– Programming and Hardware, McGraw Hill
2. 8000 to 8085 Introduction to 8085 Microprocessor for Engineers and Scientists, A.K.Gosh, Prentice Hall

Contents

3 Chapter INSTRUCTION CYCLE

4 Chapter ASSEMBLY LANGUAGE PROGRAMMING

5 Chapter BASIC I/O, MEMORY R/W AND INTERRUPT OPERATIONS

6 Chapter I/O INTERFACE

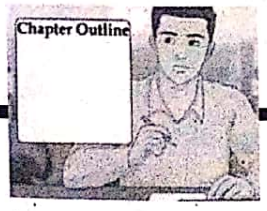
Chapter 7 ADVANCED MICROPROCESSORS

1

INTRODUCTION TO MICROPROCESSOR

CHAPTER OBJECTIVE

This chapter introduces the basic concept of microprocessor and its application areas. Furthermore, the concept of stored program system using Von Neumann Architecture & Harvard Architecture is discussed. This chapter also explains about the Intel Series evolution, block diagram of basic microprocessor based system, interconnection and intercommunication between the different modules with the help of system bus.



CHAPTER OUTLINE

- Definition of Microprocessor and its Application
- Evolution of Microprocessor, Von Neumann and Harvard Architecture
- Components of Microprocessor
  - Microprocessor: Arithmetic and Logic Unit (ALU), Control Unit (CU), Registers
  - Memory
  - Input / Output
- System Bus

1.1 Definition of Microprocessor and its Application

A Microprocessor is a multipurpose programmable, clock driven, register based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input, processes data according to those instructions and provide results as output. The microprocessor operates in binary 0 and 1 known as bits that are represented in terms of electrical voltages in the machine that means 0 represents low voltage level and 1 represents high voltage level. Each microprocessor recognizes and processes a group of bits called the word and microprocessors are classified according to their word length such as 8 bits microprocessor with 8 bit word and 32 bit microprocessor with 32 bit word, etc.

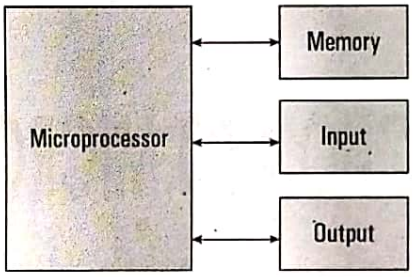


Figure 1.1: A Programmable Machine

In other words, a Microprocessor is a clock driven semiconductor device consisting of electronic circuits manufactured by using either a LSI or VLSI technique.

A typical programmable machine can be represented with three components: MPU, Memory and I/O as shown in figure above.

These three components work together or interact with each other to perform a given task; thus they comprise a system. The machine (system) represented in above figure can be programmed to turn traffic lights on and off, compute mathematical functions, or keep trace of guidance system.

This system may be simple or sophisticated, depending on its applications. The MPU applications are classified primarily in two categories: reprogrammable systems and embedded systems.

- In reprogrammable systems, such as Microcomputers, the MPU is used for computing and data processing.
- In embedded systems, the microprocessor is a part of a final product and is not available for reprogramming to end user.

Terms Used

- **CPU:** Central processing unit which consists of ALU and control unit.
- **Microprocessor:** Single chip containing all units of CPU.
- **Microcomputer:** Computer having microprocessor as CPU.
- **Microcontroller:** Single chip consisting of MPU, memory, I/O and interfacing circuits.
- **MPU:** Micro processing unit – complete processing unit with the necessary control signals.

Advantages of Microprocessor

- Computational/Processing speed is high
- Intelligence has been brought to systems
- Automation of industrial process and office automation
- Flexible

- Compact in size
- Maintenance is easier

Applications of Microprocessors

- **Microcomputer:** Microprocessor is the CPU of the microcomputer.
- **Embedded system:** Used in microcontrollers.
- **Measurements and testing equipment:** Used in signal generators, oscilloscopes, counters, digital voltmeters, x-ray analyzer, blood group analyzers baby incubator, frequency synthesizers, data acquisition systems, spectrum analyzers etc.
- Washing machine
- Microwave oven
- On Board Systems
- Scientific and Engineering research
- **Industry:** used in data monitoring system, automatic weighting, batching systems etc.
- **Security systems:** smart cameras, CCTV, smart doors, etc.
- Automatic system
- Communication system; some examples are:
  - Calculators
  - Accounting system
  - Games machine
  - Complex industrial controllers
  - Traffic light control
  - Data acquisition systems
  - Military applications

Example: A system design with MPU

- Smart fan,
- Access control system,
- Automatic water tank

Microcomputer

As the name implies, Microcomputers are small computers. They range from small controllers that work directly with 4-bit words to larger units that work directly with 32-bit words. Some of the more powerful microcomputers have all or most of the features of earlier minicomputers. Examples of Microcomputers are Intel 8051 controller-a single board computer, IBM PC and Apple Macintosh computer.

Micro Controller

Single-chip microcomputers are also known as microcontrollers. They are used primarily to perform dedicated functions. They are used primarily to perform dedicated functions or as slaves in distributed processing.

Generally they include all the essential elements of a computer on a single chip: MPU,R/W memory, ROM and I/O lines. Typical examples of the single-chip microcomputers are the Intel 8051, AT89C51, AT89C52 and Zilog Z8. Most of the micro controllers have an 8-bit word size, at least 64 bytes of R/W memory, and 1K byte of ROM. An input/output line varies from 16 to 40 depending upon the number of memory blocks.

Typical Example: AT89C51 Microcontroller

- It is low power, high performance CMOS 8 bit microcomputer with 4K bytes of Flash programmable and erasable Read Only Memory.
- 128 bytes of Internal RAM
- 32 I/O pins arranged as 4 ports (PO-P3)
- A full duplex serial port
- 6 Hardware Interrupts
- 16 bit PC and Data Pointers
- 8 bit Program Status Word
- Two 16 bits timers/counter TO and T1

Memory Calculations

- 2^{10} =1K (Kilo)
- 2^{20}=1M (Mega)
- 2^{30}=1G (Giga)
- 2^{40}=1T.(Tera)
- Specify Data and Address Bus size and calculate the size of Memory.

1.2 Evolution of Microprocessor, Von Neumann and Harvard Architecture

Historical Background of the Development of Computers

The most efficient and versatile electronic machine computer is basically a development of a calculator which leads to the development of the computer. The older computer were mechanical and newer are digital. The mechanical computer namely difference engine and analytical engine developed by Charles Babbage (the father of the computer) can be considered as the forerunners of modern digital computers.

The difference engine was a mechanical device that could add and subtract and could only run a single algorithm. Its output system was incompatible to write on punched cards and early optical disks. The 'analytical engine' provided more advanced features. It consisted mainly four components the store (memory), the mill (computation unit), input section (punched card reader) and output section (punched and printed output). The store consisted of 1000s of words of 50 decimal digits used to hold variables and results. The mill could accept operands from the store, add, subtract, multiply or divide them and return a result to the store.

The evolution of the vacuum tubes led the development of computer into a new era. The world's first general purpose electronic digital computer was ENIAC (Electronic Numerical Integrator and Calculator) built by using vacuum tubes was enormous in size and consumed very high power. However it was faster than mechanical computers. The ENIAC was decimal machine and performed only decimal numbers. Its memory consisted of 20 'accumulators' each capable of holding 10 digits decimal numbers. Each digit was represented by a ring of 10 vacuum tubes. ENIAC had to be programmed manually by setting switches and plugging and unplug a cable which was the main drawback of it.

Automated Calculator

It is a data processing device that carries out logic and arithmetic operations but has limited programming capability for the user. It accepts data from a small keyboard one digit at a time performs required arithmetic and logical calculations and stores the result on visual display like LCD or LED. The calculator's programs are stored in ROM's while the data is stored in RAM.

Some important features of automated calculations:

- ⌘ The ability to interface easily with keyboards and displays.
- ⌘ The ability to handle decimal digits, the device is able to handle more than 4 bits at a time.
- ⌘ Ability to execute the standard programs stored in Read Only Memory.
- ⌘ Extendibility, so that mathematical functions such as %, √, trigonometric, statistical etc. can be easily executed.
- ⌘ Flexibility so it can be used in engineering business or programming without a complete new design.
- ⌘ Low cost, small size and low power consumptions.

Von-Neumann and Harvard Architecture

There are two types of digital computer architectures that describe the functionality and implementation of computer systems. One is the Von Neumann architecture that was designed by the renowned physicist and mathematician John Von Neumann in the late 1940s, and the other one is the Harvard architecture which was based on the original Harvard Mark I relay-based computer which employed separate memory systems to store data and instructions.

The original Harvard architecture used to store instructions on punched tape and data in electro-mechanical counters. The Von Neumann architecture forms the basis of modern computing and is easier to implement.

Von Neumann Architecture

It's a theoretical design based on the concept of stored-program computers where program data and instruction data are stored in the same memory.

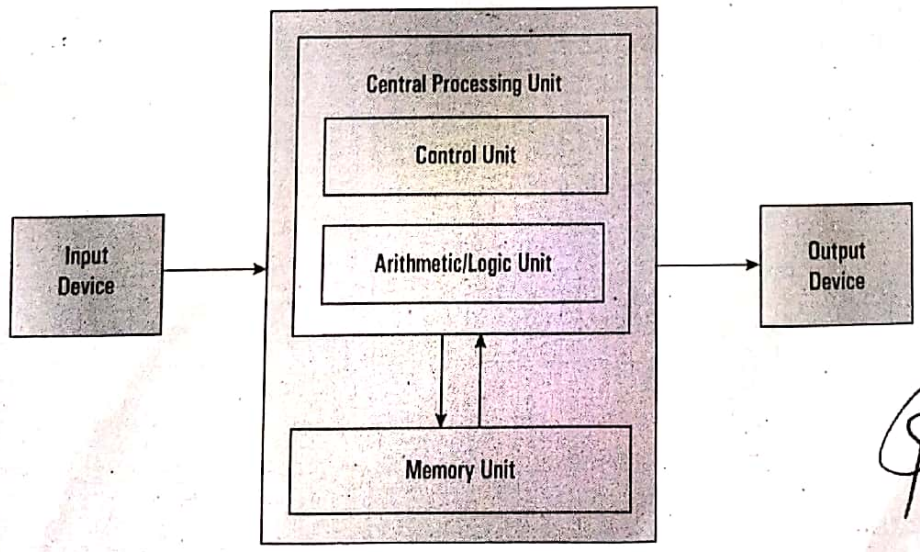


Figure 1.2: Block diagram of Von Neumann Architecture.

The architecture was designed by the renowned mathematician and physicist John Von Neumann in 1945. Until the Von Neumann concept of computer design, computing machines were designed for a single predetermined purpose that would lack sophistication because of the manual rewiring of circuitry.

The idea behind the Von Neumann architectures is the ability to store instructions in the memory along with the data on which the instructions operate. In short, the Von Neumann architecture refers to a general framework that a computer's hardware, programming, and data should follow.

The Von Neumann architecture consists of three distinct components: a central processing unit (CPU), memory unit, and input/output (I/O) interfaces. The CPU is the heart of the computer system that consists of three main components: the Arithmetic and Logic Unit (ALU), the control unit (CU), and registers.

The ALU is responsible for carrying out all arithmetic and logic operations on data, whereas the control unit determines the order of flow of instructions that need to be executed in programs by issuing control signals to the hardware.

The registers are basically temporary storage locations that store addresses of the instructions that need to be executed. The memory unit consist of RAM, which is the main memory used to store program data and instructions. The I/O interfaces allows the users to communicate with the outside world such as storage devices.

Harvard Architecture

It is a computer architecture with physically separate storage and signal pathways for program data and instructions. Unlike Von Neumann architecture which employs a single bus to both fetch instructions from memory and transfer data from one part of a computer to another, Harvard architecture has separate memory space for data and instruction.

Both the concepts are similar except the way they access memories. The idea behind the Harvard architecture is to split the memory into two parts – one for data and another for programs. The terms was based on the original Harvard Mark I relay based computer which employed a system that would allow both data and transfers and instruction fetches to be performed at the same time.

Real world computer designs are actually based on modified Harvard architecture and are commonly used in microcontrollers and DSP (Digital Signal Processing).

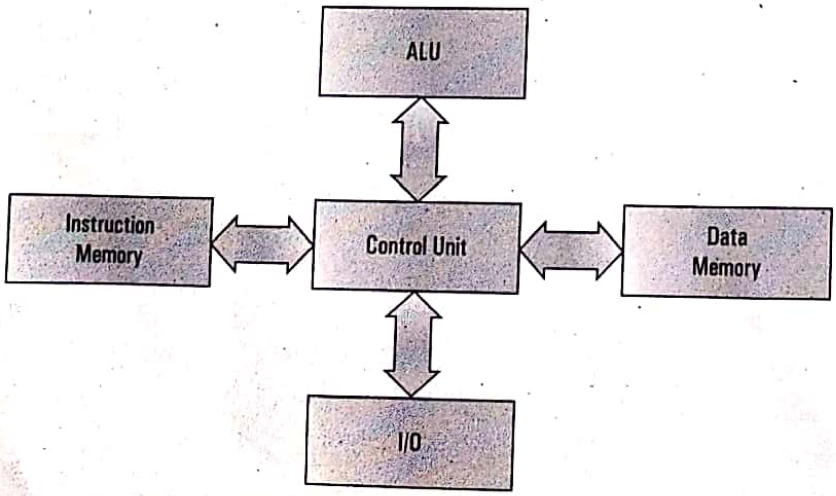


Figure 1.3: Block diagram of Harvard Architecture.

Difference between Von Neumann and Harvard Architecture

Basics of Von Neumann and Harvard Architecture

The Von Neumann architecture is a theoretical computer design based on the concept of stored-program where programs and data are stored in the same memory. The concept was designed by a mathematician John Von Neumann in 1945 and which presently serves as the basis of almost all modern computers. The Harvard architecture was based on the original Harvard Mark I relay-based computer model which employed separate buses for data and instructions.

Memory System of Von Neumann and Harvard Architecture

The Von Neumann architecture has only one bus that is used for both instruction fetches and data transfers, and the operations must be scheduled because they cannot be performed at the same time. The Harvard architecture, on the other hand, has separate memory space for instructions and data, which physically separate signals and storage for code and data memory, which in turn makes it possible to access each of the memory system simultaneously.

Instruction Processing of Von Neumann and Harvard Architecture

In Von Neumann architecture, the processing unit would need two clock cycles to complete an instruction. The processor fetches the instruction from memory in the first cycle and decodes it, and then the data is taken from memory in the second cycle. In the Harvard architecture, the processing unit can complete an instruction in one cycle if appropriate pipelining strategies are in place.

Cost of Von Neumann and Harvard Architecture

As instructions and data use the same bus system in the Von Neumann architecture, it simplifies design and development of the control unit, which eventually brings down the production cost to minimal. Development of control unit in the Harvard architecture is more expensive than the former because of the complex architecture that employs two buses for instructions and data.

Use of Von Neumann and Harvard Architecture

Von Neumann architecture is mainly used in every machine you see from desktop computers and notebooks to high performance computers and workstations. Harvard architecture is a fairly new concept used primarily in microcontrollers and digital signal processing (DSP).

Von Neumann vs. Harvard Architecture: Comparison Chart

Von Neumann Architecture	Harvard Architecture
It is a theoretical design based on the stored-program computer concept.	It is a modern computer architecture based on the Harvard Mark I computer model.
It uses same physical memory address for instructions and data.	It uses separate memory addresses for instruction and data.
Processor needs two clock cycles to execute an instruction.	Processor needs one cycle to complete an instruction.
Similar control unit design and development of one is cheaper and faster.	Control unit for two buses is more complicated which adds to the development cost.
Data transfers and instruction fetches cannot be performed simultaneously.	Data transfer and instruction fetches can be performed at the same time.
Used in personal computers, laptops, and workstations.	Used in microcontrollers and signal processing.

Table 1.1: Difference between Von Neumann and Harvard Architecture

Summary of Von Neumann vs. Harvard Architecture

Von Neumann architecture is similar to the Harvard architecture except it uses a single bus to perform both instruction fetches and data transfers, so the operations must be scheduled. The Harvard architecture, on the other hand, uses two separate memory addresses for data and instructions, which makes it possible to feed data into both the buses at the same time. However, the complex architecture only adds to the development cost of the control unit against the lower development cost of the less complex Von Neumann architecture which employs a single unified cache.

1.3 Evolution of Microprocessor: Intel Series

4004

- The first commercially available Microprocessor was the Intel 4004 produced in 1971.
- It contained 2300 PMOS transistors.
- The 4004 was a 4 bit device intended to be used with some other devices in making a calculator.
- In 1972 Intel came out with the 8008, which was capable of working with 8 bit words.

8008

- The 8008, however required 20 or more additional devices to form a functional CPU.

8080

- In 1974 Intel announced the 8080, which had a much larger instruction set than the 8008 and required only two additional devices to form a functional CPU.
- The 8080 used NMOS transistor, so it operated much faster than the 8008
- The 8080 is referred as a Second generation Microprocessor.
- It requires +5V,-5V and +12V supply.

8085

- In 1977, Intel Produced 8085, an upgrade of 8080 that required only a +5V supply
- It was a 8 bit Microprocessor.

8088

- Intel Produced 8088, which was the first Microprocessor used in Personal computer by IBM.
- It has 16 bit registers and an 8 bit data bus and can address up to 1 million bytes of internal memory.

8086

- In 1978 Intel came out with the 8086 which is a full 16 bit Microprocessor.
- It has a 16 bit data bus and runs faster.
- It can address 2^{20} or 1048576 memory locations.

80286

- Runs faster than the preceding processors, has additional capabilities and can address up to 16 million bytes.
- This processor can operate in real mode or in protected mode, which enables an operating system like windows to perform multitasking and to protect them from each other.

80386

- Has 32 bit registers and 32 bit data bus.
- It can address up to 4 billion bytes of memory.
- The processor supports virtual mode, whereby it can swap portions of memory onto disk.

80486

- ⌘ Has 32 bit registers and 32 bit data bus.
- ⌘ High speed cache memory connected to the processor bus enables the processor to store copies of the most recently used instructions and data.
- ⌘ The processor can operate faster when using the cache directly without having to access the slower memory.

PENTIUM

- ⌘ It has 32 bit registers, a 64 bit data bus and separate caches for data and for memory.
- ⌘ The Pentium has a 5 Stage pipelined structure and the Pentium II has a 12 stage super pipelined structure.
- ⌘ This feature enables them to run many operations in parallel.

The CPU of a computer consists of ALU, CU and memory. If all these components can be organized on a single chip by means of SSI, MSI, LSI, VLSI, ULSI, ELSI technology, then such chip is called microprocessor. It can fetch instructions from memory, decode and execute them, perform logical and arithmetic functions, accept data from input devices and send results to the output devices. The evolution of microprocessor is dependent on the development of integrated circuit technology from single scale integration (SSI) to giga scale integration (GSI).

Date	Microprocessor	Data bus	Address Bus	Memory
1971	4004	4 bit	10 bit	640 Bytes
1972	8008	8 bit	14 bit	16k
1974	8080	8 bit	16 bit	64k
1976	8085	8 bit	16 bit	64k
1978	8086	16 bit	20 bit	1M
1979	8088	8 bit	20 bit	1M
1982	80286	16 bit	24 bit	16M
1985	80386	32 bit	32 bit	4G
1989	80486	32 bit	32 bit	4G
1993	Pentium	32/64 bit	32 bit	4G
1995	Pentium pro	32/64 bit	36 bit	64G
1997	Pentium II	64 bit	36 bit	64G
1998	Celeron	64 bit	36 bit	64G
1999	Pentium III	64 bit	36 bit	64G
2000	Pentium IV	64 bit	36 bit	64G
2001	Itanium	128 bit	64 bit	64G
2002	Itanium 2	128 bit	64 bit	64G
2003	Pentium M/Centrino (wireless capability) for Mobile version e. g. Laptop			
	Core 2: X86 – 64 Architecture			

Table 1.2: Evolution of Intel series

1.4 Components of Microprocessor

Microprocessor based system includes three components microprocessor, input/output and memory (read only and read/write). These components are organized around a common communication path called a bus.

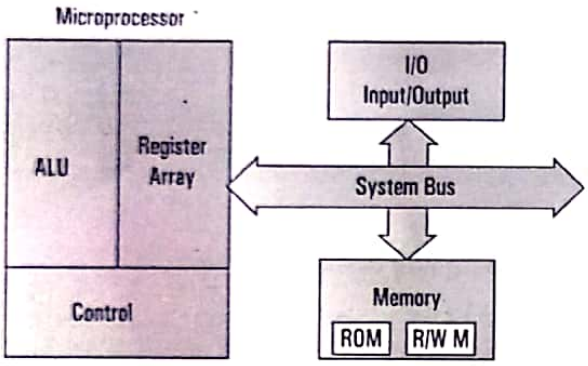


Figure 1.4: Microprocessor Based System with Bus Architecture

Microprocessor

It is clock driven semiconductor device consisting of electronic logic circuits manufactured by using either a large scale integration (LSI) or very large scale integration (VLSI) technique. It is capable of performing various computing functions and making decisions to change the sequence of program execution. It can be divided in to three segments.

A. **Arithmetic/Logic unit:** It performs arithmetic operations as addition and subtraction and logic operations as AND, OR &XOR.

B. **Register Array:** The registers are primarily used to store data temporarily during the execution of a program and are accessible to the user through instruction. The registers can be identified by letters such as B, C, D, E, H and L.

C. **Control Unit:** It provides the necessary timing and control signals to all the operations in the microcomputer. It controls the flow of data between the microprocessor and memory & peripherals.

Memory

Memory stores binary information such as instructions and data, and provides that information to the up whenever necessary. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU. Results are either transferred to the output section for display or stored in memory for later use. Memory has two sections.

A. **Read only Memory (ROM):** Used to store programs that do not need alterations and can only read.

B. **Read/Write Memory (RAM):** Also known as user memory which is used to store user programs and data. The information stored in this memory can be easily read and altered.

Input/Output

- ⌘ It communicates with the outside world using two devices input and output which are also known as peripherals.
- ⌘ The input device such as keyboard, switches, and analog to digital converter transfer binary information from outside world to the microprocessor.
- ⌘ The output devices transfer data from the microprocessor to the outside world. They include the devices such as LED, CRT, digital to analog converter, printer, etc.

1.5 System Bus

A system bus is a single computer bus that connects the major components of a computer system, combining the functions of a data bus to carry information, an address bus to determine where it should be sent, and a control bus to determine its operation. The technique was developed to reduce costs and improve modularity, and although popular in the 1970s and 1980s, more modern computers use a variety of separate buses adapted to more specific needs.

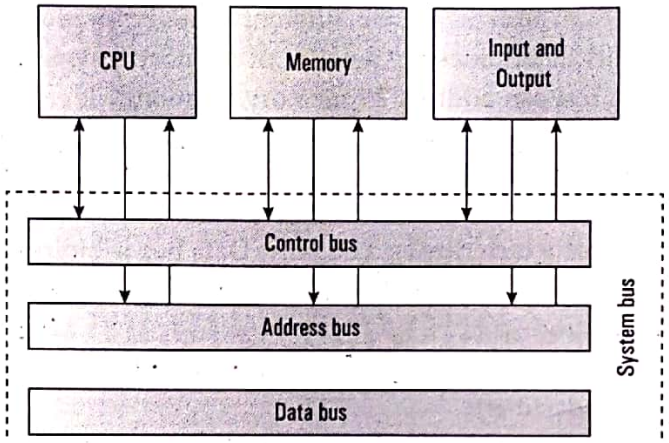


Figure 1.5: Illustration of a system bus

The design of the system bus varies from system to system and can be specific to a particular computer design or may be based on an industry standard. One advantage of using the industry standard is the ease of upgrading the computer using standard components such as the memory and IO devices from independent manufacturers.

System bus characteristics are dependent on the needs of the processor, the speed, and the word length of the data and instructions. The size of a bus, also known as its width, determines how much data can be transferred at a time and indicates the number of available wires. A 32-bit bus, for example, refers to 32 parallel wires or connectors that can simultaneously transmit 32 bits. The design and dimensions of the system bus are based on the specific processor technology of the motherboard. This, in effect, affects the speed of the motherboard, with faster system buses requiring that the other components on the system be equally fast for the best performance.

1.6 Microprocessor with Bus Organization

Bus is a group of conducting wires which carries information. All the peripherals are connected to microprocessor through bus. Diagram given below represents the bus organization system of 8085 Microprocessor.

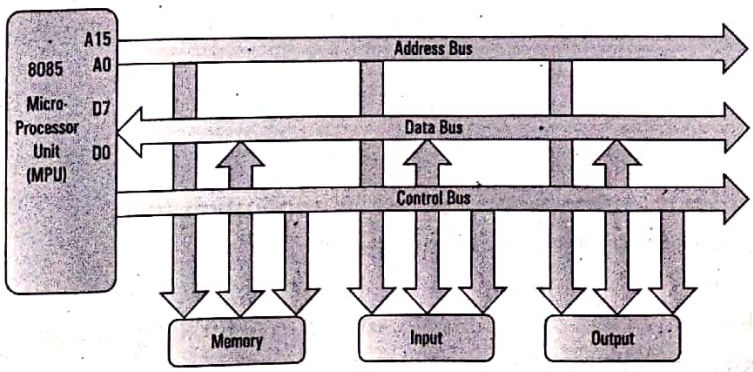


Figure 1.6: Bus organization of 8085 Microprocessor

There are three types of buses.

Address Bus

It is a group of conducting wires which carries address only. Address bus is unidirectional because data flow in one direction, from microprocessor to memory or from microprocessor to Input/output devices (That is, Out of Microprocessor).

Length of Address Bus of 8085 microprocessor is 16 Bit (That is, Four Hexadecimal Digits), ranging from 0000 H to FFFF H, (H denotes Hexadecimal). The microprocessor 8085 can transfer maximum 16 bit address which means it can address 65, 536 different memory location.

The Length of the address bus determines the amount of memory a system can address. Such as a system with a 32-bit address bus can address 2^{32} memory locations. If each memory location holds one byte, the addressable memory space is 4 GB. However, the actual amount of memory that can be accessed is usually much less than this theoretical limit due to chipset and motherboard limitations.

Data Bus

It is a group of conducting wires which carries data only. Data bus is bidirectional because data flow in both directions, from microprocessor to memory or Input/Output devices and from memory or Input/Output devices to microprocessor.

Length of Data Bus of 8085 microprocessor is 8 Bit (That is, two Hexadecimal Digits), ranging from 00 H to FF H. (H denotes Hexadecimal).

When it is write operation, the processor will put the data (to be written) on the data bus, when it is read operation, the memory controller will get the data from specific memory block and put it into the data bus.

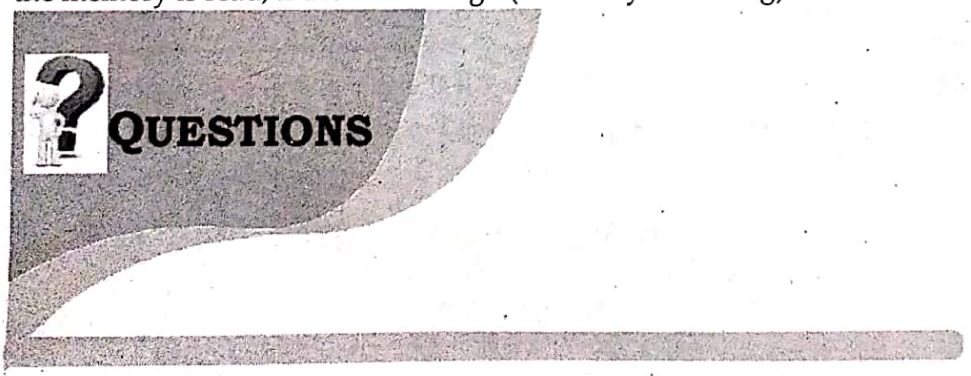
The width of the data bus is directly related to the largest number that the bus can carry, such as an 8 bit bus can represent 2^8 unique values, this equates to the number 0 to 255. A 16 bit bus can carry 0 to 65535.

Control Bus

It is a group of conducting wires, which is used to generate timing and control signals to control all the associated peripherals. Microprocessor uses control bus to process data, that is what to do with selected memory location. Some control signals are:

- ⌘ Memory read
- ⌘ I/O read
- ⌘ Opcode fetch
- ⌘ Memory write
- ⌘ I/O Write

If one line of control bus may be the read/write line. If the wire is low (no electricity flowing) then the memory is read, if the wire is high (electricity is flowing) then the memory is written.



1. What do you mean by microprocessor? List out its application areas?
2. Explain the evolution of Intel series microprocessor.
3. Differentiate between Von Neumann and Harvard Architecture.
4. Draw the block diagram of basic microprocessor and explain its operation.
5. What is system bus? Explain its types.

❑❑❑

2

BASIC COMPUTER ARCHITECTURE

CHAPTER OBJECTIVE

In the first section of this chapter, the 8085 Microprocessor Architecture is introduced. In this section the working operation of 8085 microprocessor is explained with the help of its block diagram, different registers, flags, ALU, control signal and system bus. Also, it explains pin diagram, addressing modes, memory read write operations using time multiplexed address/data bus and generation of control signals to operate the system. Furthermore, in second part the 8086 Microprocessor Architecture and organization is explained with the help of its block diagram including bus interface unit and execution unit. This section also introduces the concept of segment register, queue registers for pipelining and memory segmentation.

CHAPTER OUTLINE

- 8085 Microprocessor Architecture and Operations
  - Block Diagram
  - Memory and Memory Operations
  - 8085 Pin Diagram and Functions
  - Addressing Modes
  - Flag and Flag Register
  - Generation of Control Signals
- 8086 Microprocessor Architecture and Operations
  - Logical Block Diagram
  - Memory Segmentation
  - Pipelining
  - Segment Registers,
  - Bus Interface Unit and Execution Unit

2.1 8085 Microprocessor Architecture and Operations

The Intel 8085 A is a complete 8 bit parallel central processing unit. The main components of 8085A are array of registers, the arithmetic logic unit, the encoder/decoder, and timing and control circuits linked by an internal data bus. The block diagram is shown below:

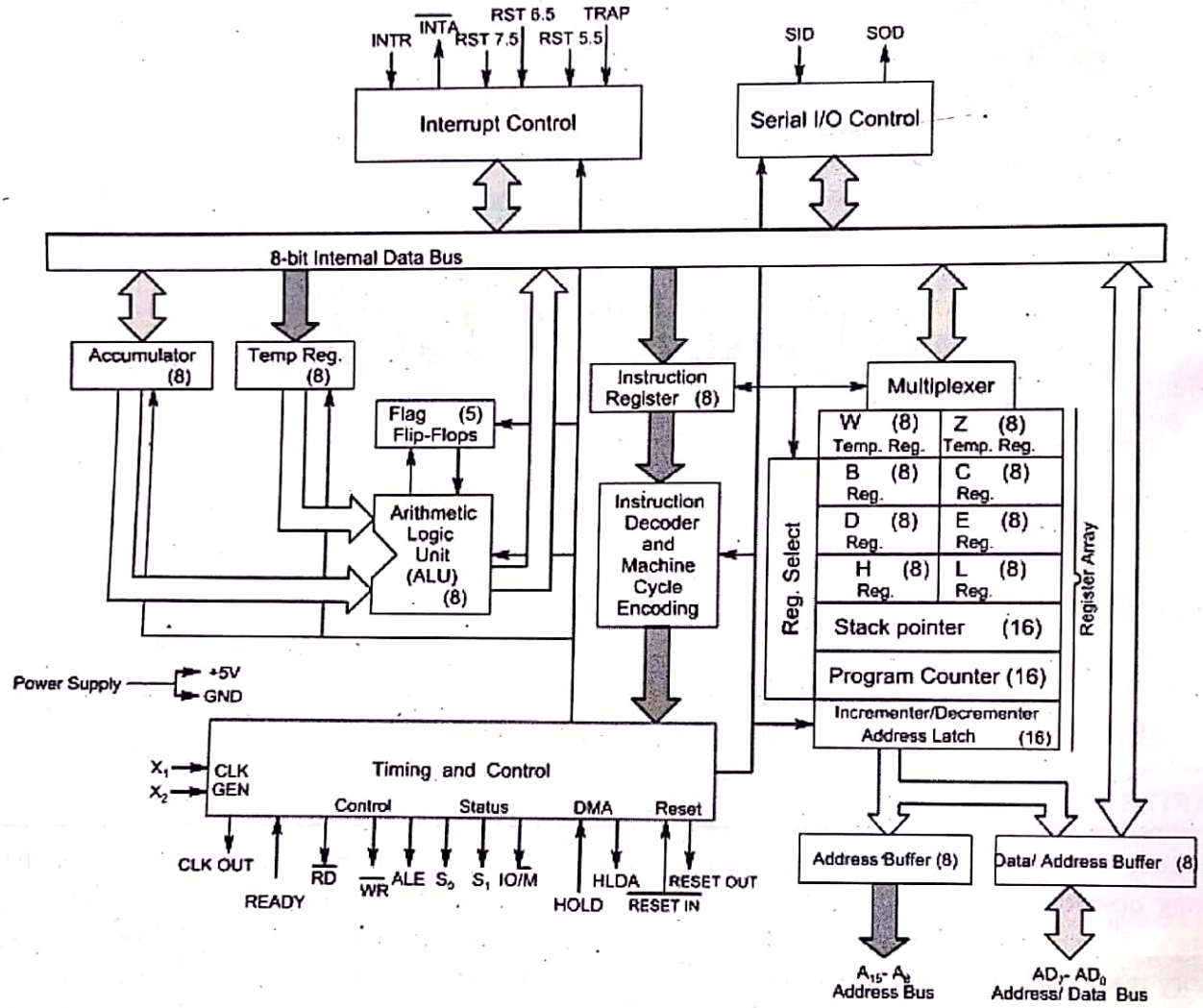


Figure: 2.1: The 8085 A microprocessor Functional Block Diagram.

1: **ALU:** The arithmetic logic unit performs the computing functions, it includes the accumulator, the temporary register, the arithmetic and logic circuits and five flags. The temporary register is used to hold data during an arithmetic/logic operation. The result is stored in the accumulator; the flags (flip-flops) are set or reset according to the result of the operation.

2. **Accumulator (register A):** It is an 8 bit register that is the part of ALU. This register is used to store the 8-bit data and to perform arithmetic and logic operations and 8085 microprocessor is called accumulator based microprocessor. When data is read from input port, it is first moved to accumulator and when data is sent to output port, it must be first placed in accumulator.

3. **Temporary registers (W & Z):** They are 8 bit registers not accessible to the programmer. During program execution, 8085A places the data into it for a brief period.

4. **Instruction register (IR):** It is a 8 bit register not accessible to the programmer. It receives the operation codes of instruction from internal data bus and passes to the instruction decoder which decodes so that microprocessor knows which type of operation is to be performed.

5. **Register Array:** (Scratch pad registers B, C, D, E): It is a 8 bit register accessible to the programmers. Data can be stored upon it during program execution. These can be used individually as 8-bit registers or in pair BC, DE as 16 bit registers. The data can be directly added or transferred from one to another. Their contents may be incremented or decremented and combined logically with the content of the accumulator.
   - ❋ **Register H & L**: They are 8 bit registers that can be used in same manner as scratch pad registers.
   - ❋ **Stack Pointer (SP):** It is a 16 bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading a 16- bit address in the stack pointer.
   - ❋ **Program Counter (PC):** Microprocessor uses the PC register to sequence the execution of the instructions. The function of PC is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the PC is incremented by one to point to the next memory location.
6. **Flag register:** Register consists of five flip-flops, each holding the status of different states separately is known as flag register and each flip-flop are called flags. 8085A can set or reset one or more of the flags and are sign(S), Zero (Z), Auxiliary Carry (AC) and Parity (P) and Carry (CY). The state of flags indicates the result of arithmetic and logical operations, which in turn can be used for decision making processes. The different flags are described as:
   - ❋ **Carry (CY)**: If the last operation generates a carry its status will 1 otherwise 0. It can handle the carry or borrow from one word to another.
   - ❋ **Zero (Z):** If the result of last operation is zero, its status will be 1 otherwise 0. It is often used in loop control and in searching for particular data value.
   - ❋ **Sign (S)**: If the most significant bit (MSB) of the result of the last operation is 1 (negative), then its status will be 1 otherwise 0.
   - ❋ **Parity (P)**: If the result of the last operation has even number of 1's (even parity), its status will be 1 otherwise 0.
   - ❋ **Auxiliary carry (AC)**: If the last operation generates a carry from the lower half word (lower nibble), its status will be 1 otherwise 0. Used for performing BCD arithmetic.

   Its bit position is shown in the following diagram:

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

7. **Timing and Control Unit:** This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals. The control signals are similar to the sync pulse in an oscilloscope. The $\overline{RD}$ and $\overline{WR}$ signals are sync pulses indicating the availability of data on the data bus.
8. **Interrupt controls:** The various interrupt controls signals (INTR, RST 5. 5, RST 6. 5, RST 7. 5 and TRAP) are used to interrupt a microprocessor.
9. **Serial I/O controls:** Two serial I/O control signals (SID and SOD) are used to implement the serial data transmission.

PIN Configuration of 8085

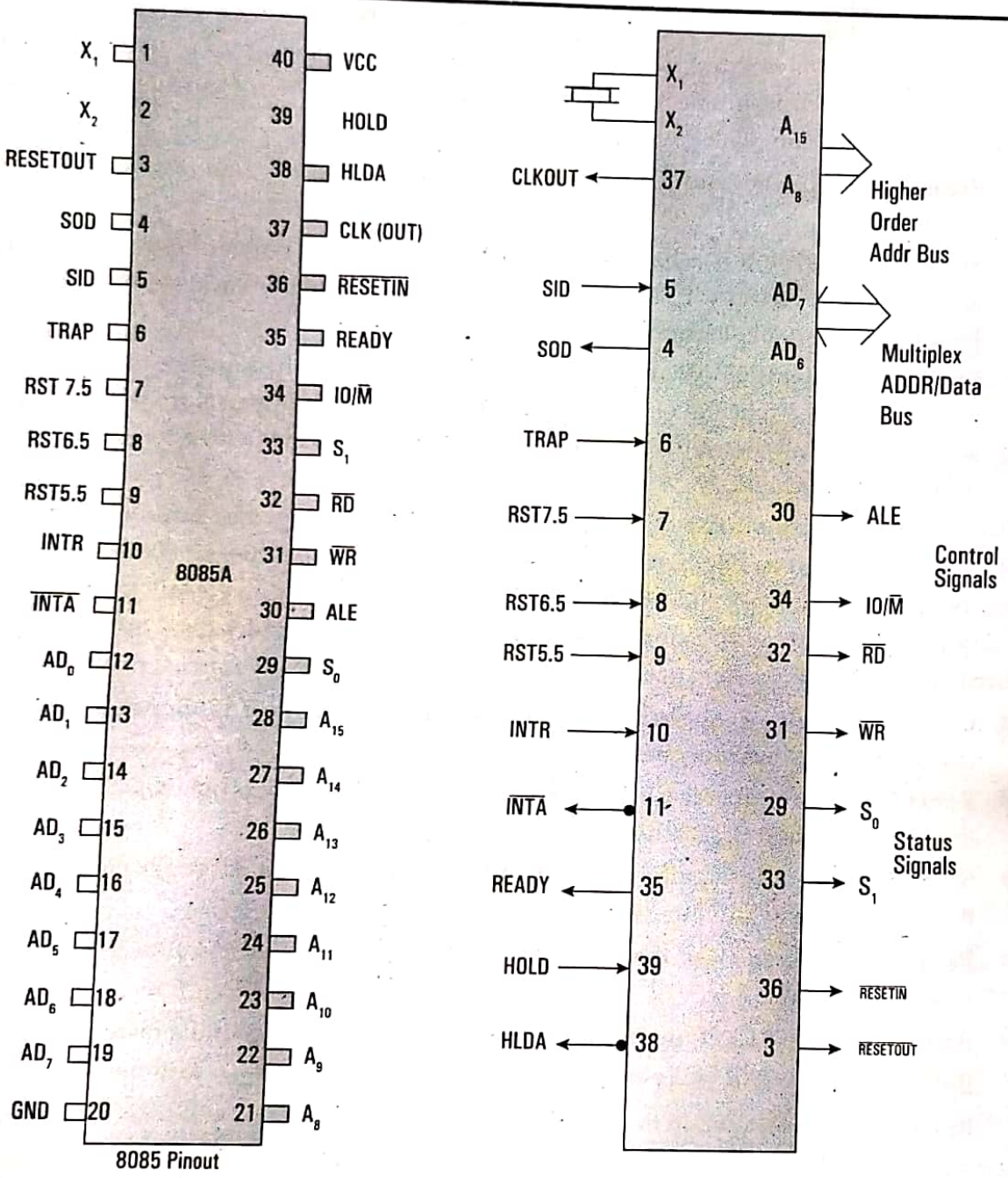


Figure: 2.2: PIN Configuration of 8085

- The microprocessor is a clock-driven semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale integration (LSI) or very-large-scale integration (VLSI) technique.
- The microprocessor is capable of performing various computing functions and making decisions to change the sequence of program execution.
- In large computers, a CPU implemented on one or more circuit boards performs these computing functions.
- The microprocessor is in many ways similar to the CPU, but includes the logic circuitry, including the control unit, on one chip.
- The microprocessor can be divided into three segments for the sake of clarity: arithmetic/logic unit (ALU), register array, and control unit.

- 8085 is a 40 pin IC, DIP package. The signals from the pins can be grouped as follows:
  1. Power supply and clock signals
  2. Address bus
  3. Data bus
  4. Control and status signals
  5. Interrupts and externally initiated signals
  6. Serial I/O ports

1. Power Supply and Clock Frequency Signals

- Vcc: + 5 volt power supply
- Vss: Ground
- X1, X2: Crystal or R/C network or LC network connections to set the frequency of internal clock generator.
- The frequency is internally divided by two. Since the basic operating timing frequency is 3 MHz, a 6 MHz crystal is connected externally.
- CLK (output)-Clock Output is used as the system clock for peripheral and devices interfaced with the microprocessor.

2. Address Bus

- A8 - A15
- It carries the most significant 8 bits of the memory address or the 8 bits of the I/O address.

3. Multiplexed Address / Data Bus

- AD0 - AD7
- These multiplexed set of lines used to carry the lower order 8 bit address as well as data.
- During the opcode fetch operation, in the first clock cycle, the lines deliver the lower order address A0 - A7.
- In the subsequent IO / memory, read / write clock cycle the lines are used as data bus.
- The CPU may read or write out data through these lines.

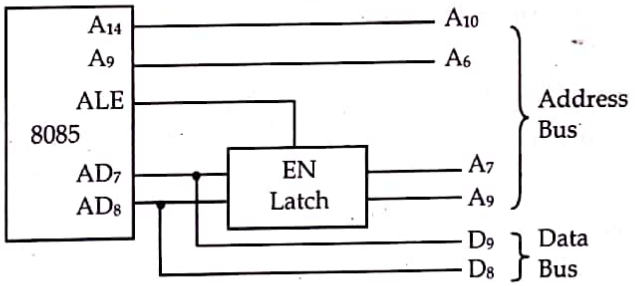


Figure 2.3: Time multiplexed address/data bus.

4. Control and Status Signals

These signals include two control signals (RD & WR) three status signals (IO/M, S1 and S0) to identify the nature of the operation and one special signal (ALE) to indicate the beginning of the operations.

- **ALE (output)** - Address Latch Enable. This signal helps to capture the lower order address presented on the multiplexed address / data bus. When it is the pulse, 8085 begins an operation. It generates AD0 - AD7 as the separate set of address lines A0 –A7.
- **RD (active low)** - Read memory or IO device. This indicates that the selected memory location or I/O device is to be read and that the data bus is ready for accepting data from the memory or I/O device.

- **WR (active low)** - Write memory or IO device. This indicates that the data on the data bus is to be written into the selected memory location or I/O device.
- **IO/M (output)** - Select memory or an IO device. This status signal indicates that the read / write operation relates to whether the memory or I/O device. It goes high to indicate an I/O operation. It goes low for memory operations.

$IO/\overline{M}$	S_1	S_0	States
0	0	1	Memory Write
0	1	0	Memory read
1	0	1	I/O write
1	1	0	I/O read
0	1	1	Opcode fetch

Table 2.1: Function table of control and status signal.

5. Interrupts & Externally Initiated Signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i. e. TRAP, RST 7. 5, RST 6. 5, RST 5. 5, and INTR. We will discuss interrupts in detail in interrupts section.

- **INTA:** It is an interrupt acknowledgment signal.
- **RESET IN:** This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT:** This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY:** This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD:** This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge):** It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

6. Serial I/O Signals

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

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

Interrupts in 8085

Interrupts are the signals generated by the external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i. e. TRAP, RST 7. 5, RST 6. 5, RST 5. 5, and INTR. Interrupt are classified into following groups based on their parameter:

- **Vector interrupt:** In this type of interrupt, the interrupt address is known to the processor. For example: RST7. 5, RST6. 5, RST5. 5, TRAP.

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

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

Interrupt Service Routine

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

- **TRAP:** It is a non-maskable interrupt, having the highest priority among all interrupts. By default, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.
- **RST 7.5:** It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.
- **RST 6. 5:** It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.
- **RST 5. 5:** It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.
- **INTR:** It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

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

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

Multiplexing and De-multiplexing of Address / Data bus

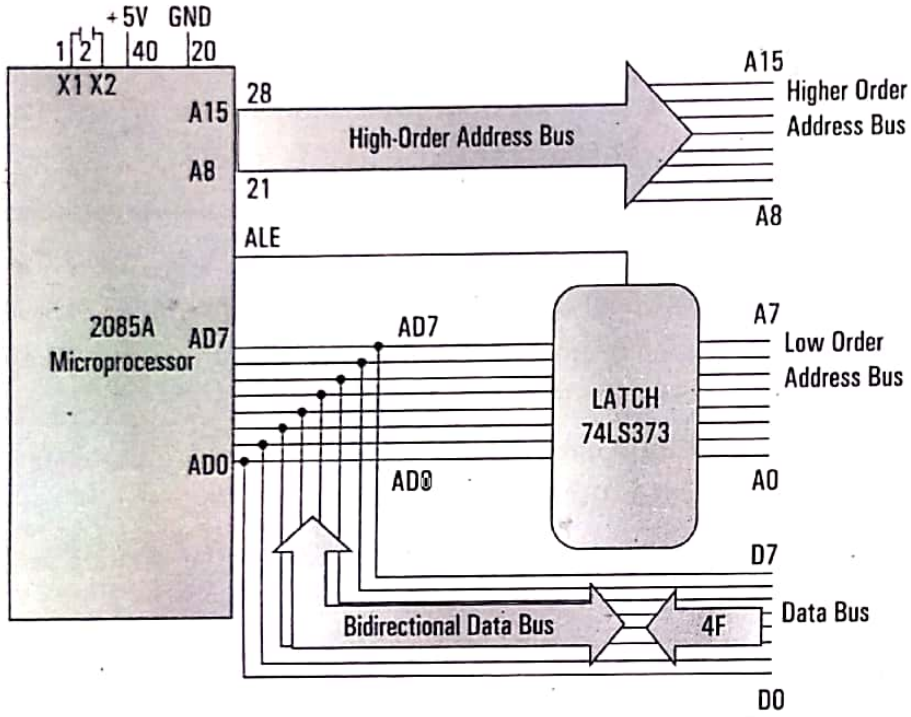


Figure 2.4: Multiplexing and De-multiplexing of Address / Data bus

- The address bus has 8 signal lines A8 -A15 which are unidirectional.
- The other 8 address bits are multiplexed (time shared) with the 8 data bits. So, the bits AD0 - AD7 are bi-directional and serve as A0-A7 and D0-D7 at the same time. During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.
- In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.
- The high order bits of the address remain on the bus for three clock periods. However, the low order bits remain for only one clock period and they would be lost if they are not saved externally. Also, notice that the low order bits of the address disappear when they are needed most.
- To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7-AD0 when it is carrying the address bits. We use the ALE (Address Latch Enable) signal to enable this latch.
- Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7-AD0 lines can be used for their purpose as the bi-directional data lines.

2.2 Addressing Modes of 8085 Microprocessor

- The different ways in which a processor can access data are referred as its addressing modes.
- In assembly language statements, the addressing mode is indicated in the instruction itself.
- The various addressing modes are
  1. Register Addressing Mode
  2. Immediate Addressing Mode

3. Direct Addressing Mode
4. Register Indirect Addressing Mode
5. Implied Addressing Mode

1. Register Addressing Mode

- ⌘ It is the most common form of data addressing.
- ⌘ Transfers a copy of a byte/word from source register to destination register.

Instruction	Source	Destination
MOV A,B	Register B	Register A

- ⌘ It is carried out with 8 bit registers A,B,C,D,E,H & L
- ⌘ It is important to use registers of same size.
- ⌘ Never mix an 8 bit register with a 16 bit register i. e. MOV A,SP

EXAMPLES : MOV A, B : Copys B into A
MOV SP, H : Copys H pair into SP

2. Immediate Addressing Mode

- ⌘ The term immediate implies that the data immediately follow the hexadecimal opcode in the memory.

Note *that immediate data are constant data.*

- ⌘ It transfers the source immediate byte/word of data in destination register or memory location.

Instruction	Source	Destination
MVI C,3AH	Data 3AH	Register C

EXAMPLES MOV A,90 : Copys 90 into A
LXI H,1234H : Copys 1234H into H

3. Direct Addressing Mode

- ⌘ In this scheme, the address of the data is defined in the instruction itself.

Instruction	Source	Destination
LDA 2000H	Memory Location 2000H	Register A

EXAMPLES LHLD 1000H : Copies the content of 1000H address memory to L and 10001H memory to H.
LDA 2000H : Copies the content of 2000H memory to accumulator
JMP 4000H
Call 5000H

4. Register Indirect Addressing Mode

- ⌘ Register indirect addressing allows data to be addressed at any memory location through an address held in any of the H pair, B pair and D pair registers.
- ⌘ It transfers byte/word between a register and a memory location addressed.

Instruction	Source	Destination
MOV C,M	Assume HL = 1000H and M is the content of 1000 H address	Register C

EXAMPLES
MOV C, M : Copies the word contents of the memory location addressed by HL pair into C
STAX B : Copies A into the memory location addressed by B pair
JMP 4000H
Call 5000H

5. Implied Addressing Mode

⌘ The addressing mode of certain instructions is implied by the instruction's function.

Instruction	Source	Destination
STC		Carry Flag

EXAMPLES
STC : Set carry flag
CMC : Complementary carry flag
DAA : Decimal adjust accumulator content

2.3 8086 Microprocessor Architecture and Operations

The 8086 Microprocessor Overview

⌘ The Intel 8086 is a 16 bit Microprocessor that is intended to be used as the CPU in a Microcomputer.

⌘ The term 16 bit means that it's ALU, its internal registers, and most of its instructions are designed to work with 16 bit binary words.

⌘ The 8086 has a 16 bit data bus, so it can read data from or writed at a to memory and ports either 16 bits or 8 bits at a time.

⌘ The 8086 has a 20 bit address bus, so it can address 2^{20} or 1,048576 memory locations.

⌘ Sixteen bit words will be stored in two consecutive memory locations.

⌘ If the first byte of a word is at an even address, the 8086 can read the entire word in one operation.

⌘ If the first byte of the word is at an odd address, the 8086 will read the first byte with one bus operation and the second byte with another bus operation

⌘ The main point here is that if the first byte of a 16 bit word is at an even address, the 8086 can read the entire word in one operation

8086 And 8088

⌘ The Intel 8088 has the same arithmetic logic unit, the same registers and the same instruction set as the 8086

⌘ The 8088 also has a 20 bit address bus, so it can address any one of 1,048,576 bytes in memory.

⌘ The 8088 has an 8 bit data bus, so it can only read data from or writed at o memory and ports 8 bits at a time.

⌘ The 8086 can read or write either 8 or 16 bits at a time.

⌘ To read a 16 bit word from two successive memory locations, the 8088 will always have to do two read operations.

- The Intel 80186 is an improved version of 8086, and 80188 is an improved version of 8088
- In addition to 16 bit CPU, the 80186 and 80188 each have programmer peripheral devices integrated in the same package

The following diagram depicts the architecture of an 8086 Microprocessor:

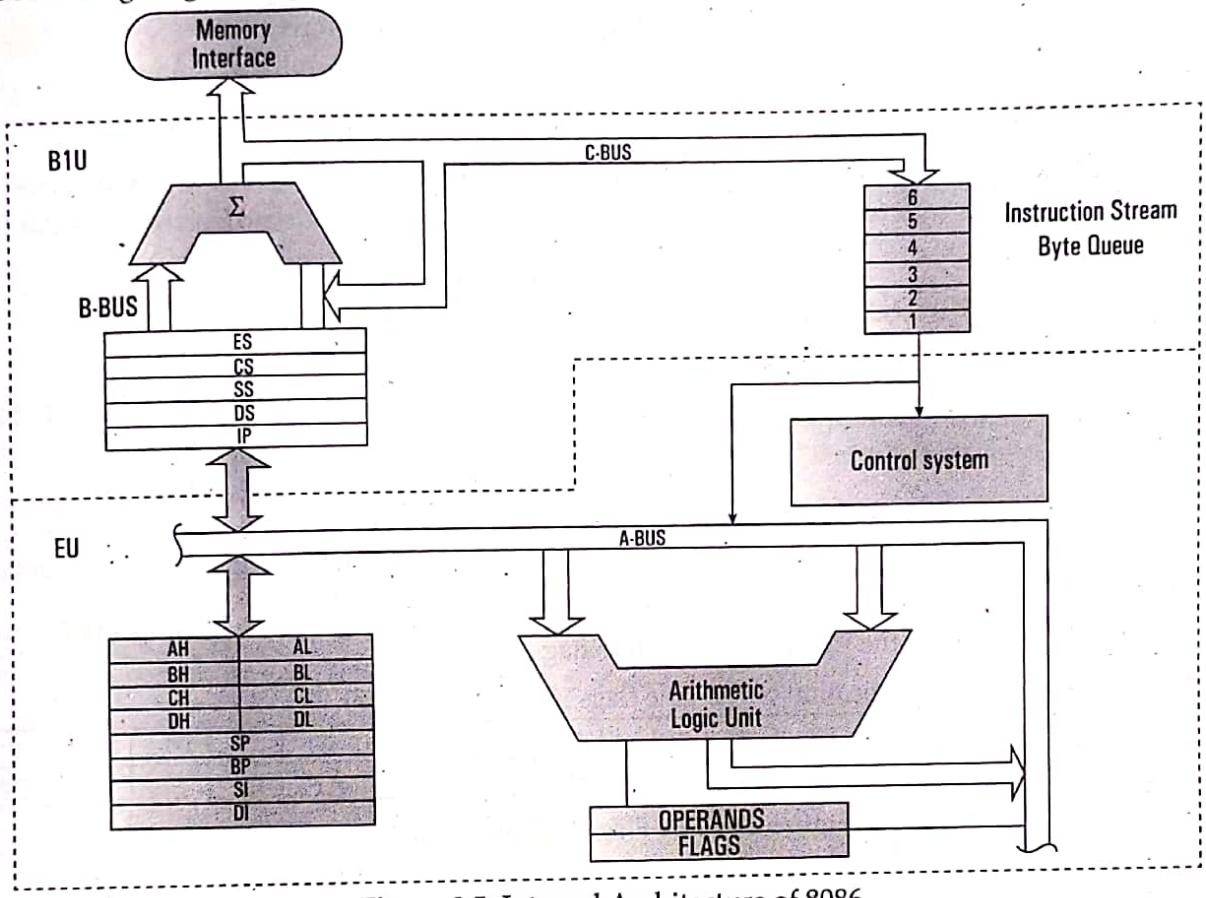


Figure 2.5: Internal Architecture of 8086

8086 Internal Architecture

- The 8086 CPU is divided in to two independent functional parts: BIU(Bus Interface Unit) and EU(Execution Unit)
- Dividing the work between these units speeds up the processing.
- The BIU send out address, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory.
- In other words BIU handles all transfers of data and addresses on the buses for the execution unit.

The EU of the 8086 tells the BIU where to fetch the instructions and data from, decodes instructions and executes instructions.

A. The Execution Unit

- The EU contains control circuitry which directs internal operations.
- Decoder in EU translates instructions fetched from memory into a series of actions which the EU carries out.
- The EU has a 16 bit ALU which can add subtract, AND, OR, increment, decrement, complement or shift binary numbers.

1. General Purpose Registers

- The EU has eight general purpose registers, labeled AH, AL, BH, BL, CH, CL, DH and DL.
- These registers can be used individually for temporary storage of 8 bit data.
- The AL register is also called accumulator
- It has some features that the other general purpose registers do not have.
- Certain pairs of these general purpose registers can be used together to store 16 bit words.
- The acceptable register pairs are AH and AL, BH and BL, CH and CL, DH and DL
- The AH-AL pair is referred to as the AX register, the BH-BL pair is referred to as the BX register, the CH-CL pair is referred to as the CX register, and the DH-DL pair is referred to as the DX register.

AX = Accumulator Register

BX = Base Register

CX = Count Register

DX = Data Register

2. Flag Register

- A Flag is a flip-flop which indicates some condition produced by the execution of an instruction or controls certain operations of the EU.
- A 16 bit flag register in the EU contains 9 active flags.
- Figure below shows the location of the nine flags in the flag register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Figure 2.6: 8086 Flag Register Format

U= Undefined

Conditional Flags

CF = Carry Flag [Set by Carry out of MSB]

PF = Parity Flag [Set if Result has even parity]

AF = Auxiliary Carry Flag for BCD

ZF = Zero Flag [Set if Result is 0]

SF = Sign Flag [MSB of Result]

OF = over flow flag

Control Flag

TF = Single Step Trap Flag

IF = Interrupt enable Flag

DF = String Direction Flag

- The six conditional flags in this group are the CF, PF, AF, ZF, SF and OF
- The three remaining flags in the Flag Register are used to control certain operations of the processor.
- The six conditional flags are set or reset by the EU on the basis of the result of some arithmetic or logic operation.

- The Control Flags are deliberately set or reset with specific instructions you put in your program.
- The three control flags are the TF, IF and DF.
- Trap Flag is used for single stepping through a program.
- The Interrupt Flag is used to allow or prohibit the interruption of a program.
- The Direction Flag is used with string instructions.

3. Pointer Registers

- The 16 bit Pointer Registers are IP ,SP and BP respectively
- SP and BP are located in EU where as IP is located in BIU

3.1 Stack pointer (SP)

- The 16 bit SP Register provides an offset value, which when associated with the SS register (SS:SP) gives the effective address of memory.

3. 2 Base Pointer (BP)

- The 16 bit BP facilitates referencing parameters, which are data and addresses that a program passes via the stack.
- The processor combines the addresses in SS with the offset in BP.
- BP can also be combined with DI and SI as a base register for special addressing.

4. Index Registers

- The 16 bit Index Registers are SI and DI

4. 1 Source Index (SI) Register

- The 16 bit Source Index Register is required for some string handling operations
- SI is associated with the DS Register.

4. 2 Destination Index (DI) Register

- The 16 bit Destination Index Register is also required for some string operations.
- In this context, DI is associated with the ES register.

B. The Bus Interface Unit

1. Segment Registers

1.1 CS Register

- It contains the starting address of a program's code segment.
- This segment address plus an offset value in the IP register indicates the address of an instruction to be fetched for execution
- For normal programming purpose, you need not directly reference this register.

1.2 DS Register

- It contains the starting address of a program's data segment
- Instruction uses this address to locate data.
- This address plus an offset value in an instruction causes a reference to a specific by the location in the data segment.

1.3 SS Register

- Permits the implementation of a stack in memory
- It stores the starting address of a program's stack segment the SS register.

- This segment address plus an offset value in the Stack Pointer (SP) register indicates the current word in the stack being addressed.

1.4 ES Register

- It is used by some string operations to handle memory addressing.
- ES Register is associated with the DI Register.

2. Instruction Pointer (IP)

- The 16 bit IP Register contains the offset address of the next instruction that is to execute.
- IP is associated with CS register as (CS:IP)
- For each instruction that executes, the processor changes the offset value in IP so that IP in effect directs each step of execution.

3. The Queue

- While the EU is decoding an instruction or executing an instruction which does not require use of the buses, the BIU fetches up to six instructions bytes for the following instructions.
- The BIU Stores prefetched bytes in First in First out register set called a queue.
- When the EU is ready for its next instruction, it simply reads the instruction bytes for the instruction from the queue in the BIU.
- This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction bytes or bytes.

Fetching the next instruction while the current instruction executes is called pipeling

Features of 8086

The most prominent features of a 8086 microprocessor are as follows:

- It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It is available in 3 versions based on the frequency of operation:
  - 8086 -> 5MHz
  - 8086-2 ->8MHz
  - 8086-1 ->10 MHz
- It uses two stages of pipelining, i. e. Fetch Stage and Execute Stage, which improves performance.
- Fetch stage can pre fetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.
- It consists of 29,000 transistors

Comparison between 8085 &8086 Microprocessor

- **Size:** 8085 is 8-bit microprocessor, whereas 8086 is 16-bit microprocessor.
- **Address Bus:** 8085 has 16-bit address bus while 8086 has 20-bit address bus.
- **Memory:** 8085 can access up to 64Kb, whereas 8086 can access up to 1 Mb of memory.
- **Instruction:** 8085 does not have an instruction queue, whereas 8086 has an instruction queue.
- **Pipelining:** 8085 does not support a pipelined architecture while 8086 supports a pipelined architecture.
- **I/O:** 8085 can address $2^8 = 256$ I/O's, whereas 8086 can access $2^{16} = 65,536$ I/O's.
- **Cost:** The cost of 8085 is low whereas that of 8086 is high.

8086 Memory Organization

1. Introduction

- The Intel 8086 is a 16 bit Microprocessor that is intended to be used as the CPU in a Microcomputer.
- The 8086 has a 20 bit address bus so it can address any one of 2^{20} or 1,048,576 memory locations.
- Each of the 1,048,576 memory address of the 8086 represents a byte-wide location.
- 16 bit word will be stored in two consecutive memory locations.
- If the first byte of a word is at an even address, the 8086 can read the entire word in one operation.
- If the first byte of a word is at an odd address, the 8086 will read the first byte with one bus cycle and the second byte with another bus cycle.

2. Accessing Data in Memory

- An important point here is that an 8086 always stores the low byte of word in lower address and stores high byte of word in higher address.
- Low Byte – Low Address : High Byte – High Address
- MOV AX,[437AH]
- Assume DS=2000H
- To Compute the physical address Add 20000 H and 437AH 20000 H + 437A H = 2437AH
- 2437A H is the physical address.

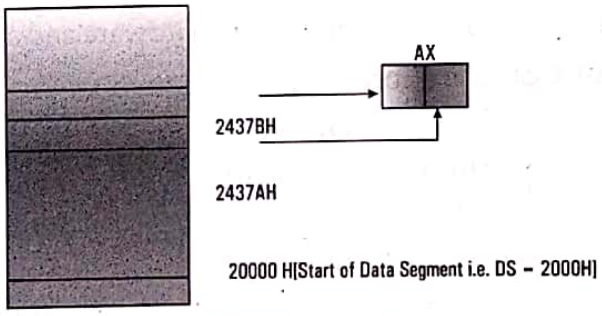


Figure 2.6: 8086 Memory Organization

3. Segmented Memory

- The 8086 BIU sends out 20-bit address so it can address any of 2^{20} or 1,048,576 bytes in memory.
- However at any given time the 8086 works with only four 65536 bytes (64 Kbyte) segment within this 1,048,576 byte (1 Mbyte) Range
- Four segments are : Code Segment, Stack Segment, Data Segment and Extra Segment

- Four segment registers in BIU are used to hold the upper 16 bits of the starting address of 4 memory segments that the 8086 is working with at a particular time.
- The 4 segment registers are code segment register (CS), stack segment register (SS), data segment register (DS) and the extra segment register (ES).
- For small programs which do not need all 64 Kbytes in each segment can overlap.
- For example, the code segment holds the upper 16 bits of the starting address for the segment from which the BIU is currently fetching instruction code bytes.
- The BIU always inserts zero for the lowest 4 bits of the 20-bit starting address.
- If the code segment register contains 348A H then the code segment will start at address 348A0 H.
- A 64 Kbytes segment can be located anywhere within the 1 Mbyte address space, but the segment will always start at an address with zeros in the lowest 4 bits

Advantages of Segmentation [Segment: Offset Scheme]

Intel designed the 8086 family devices to access memory using the segment: offset approach rather than accessing memory directly with 20 bit. The advantages are listed below.

- The segment: offset scheme requires only a 16-bit number to represent the base address for a segment and only a 16 bit offset to access any location in a segment. This means that 8086 has to manipulate and store only 16-bit quantities instead of 20-bit quantities.
- This makes easier interface with 8 and 16-bit wide memory boards and with 16 bit registers in the 8086.
- It allows programs to be relocated in memory system. A relocatable program is one that can be placed in any area of memory and executed without change.
- It allows programs written to function in the real mode to operate in protected mode.
- Segmentation also makes easy to keep user's program and data separate from one another and segmentation makes it easy to switch from one user's program to another user's program.

Disadvantage of Segment: Offset Approach

- The segment: offset scheme introduces complexity in hardware and software design.

Different Segment Offset Combination

Segment	Offset	Special Purpose
CS [Code Segment]	IP [Instruction Pointer]	Instruction
SS [Stack Segment]	SP [Stack Pointer]	Address
	BP [Base Pointer]	
DS [Data Segment]	BX [Base Register]	Stack
	DI [Destination Index]	
	SI [Source Index]	
	8 Bit Number	
	16 Bit Number	Address
ES [Extra Segment]		

Table 2.2: Different Segment Offset Combination

Memory Banking

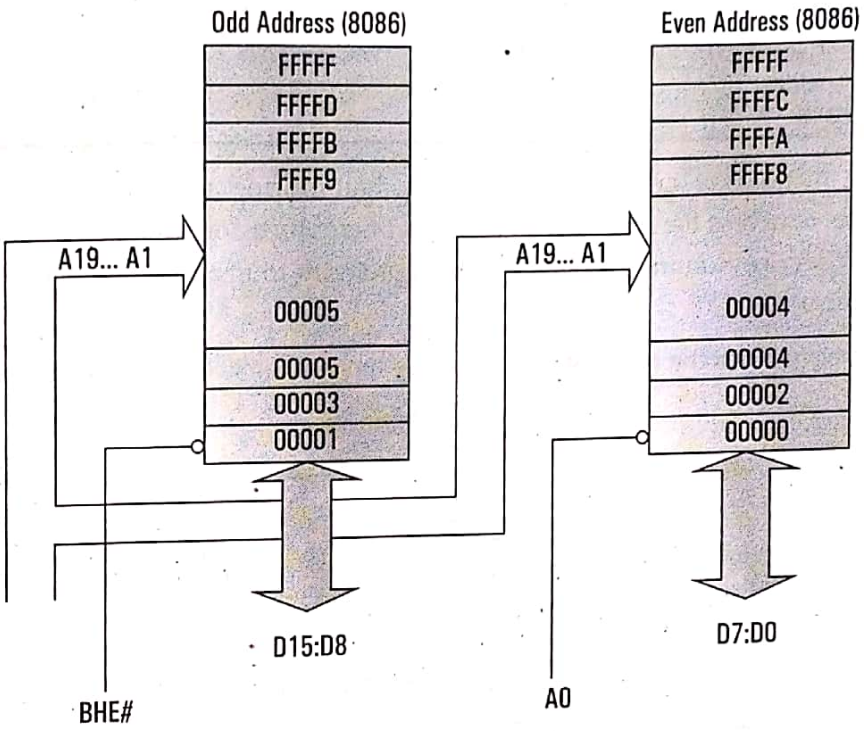


Figure 2.7: Memory Banking

- A memory bank is designated section of computer memory used for storing data.
- In 8086 there is 20 bit address bus, so it can address 1,048,576 addresses. At each address we can store 8 bit address (1-byte) but if want to write a word (16-bit) into a memory segment to store data in byte from then we write the data in two consecutive memory address which are even (low) and odd (high) memory.
- The 8086 memory address space can be viewed as a sequence of one million bytes in which any byte may contain an 8-bit data element and any two consecutive memory address which are even (low) and odd (high) memory.
- The 8086 memory address space can be viewed as a sequence of one million bytes in which any byte may contain an 8-bit data element and any two consecutive bytes may contain a 16-bit data element.
- There is no constraint on byte or word address boundaries.
- The address space is physically connected to a 16-bit data bus by dividing the address space into two 8-bit banks of up to 512K byte each.

Even Memory Bank

- One bank is connected to the lower half of the 16-bit data bus (D0-D7) and contains even address bytes, i. e. , when A0 bit is low, the bank is selected.
- To access memory bytes from even address, information is transferred over the lower half of the data bus (D0 - D7). The A0 is output Low and BHE is output high enabling only the even address bank.

Odd Memory Bank

- The other bank is connected to the upper half of the data bus (D8 - D15) and contains odd address bytes, i. e., when A0 is high and BHE (Bus High Enable) is low, the odd bank is selected.

- To access memory byte from an odd address information, is transferred over the higher half of the data bus (D8 - D15). The BHE output low enables the upper memory bank. A0 is output high to disable the lower memory bank.

8086 – Interrupts

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an ISR (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in an 8086 microprocessor:

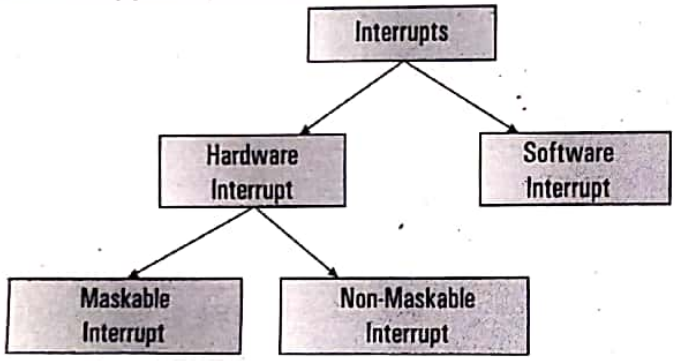


Figure 2.8: 8086 – Interrupts

Hardware Interrupts

Any peripheral device causes hardware interrupt by sending a signal through a specified pin to the microprocessor. The 8086 has two hardware interrupt pins, i.e., NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR) and it is of type 2 interrupt.

When this interrupt is activated, these actions take place:

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.

INTR:

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends '0' on INTA pin twice. The first

'0' means INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor:

- ⌘ First completes the current instruction.
- ⌘ Activates INTA output and receives the interrupt type, say X.
- ⌘ Flag register value; CS value of the return address and IP value of the return address are pushed on to the stack.
- ⌘ IP value is loaded from the contents of word location X x 4
- ⌘ CS is loaded from the contents of the next word location.
- ⌘ Interrupt flag and trap flag is reset to 0

Software Interrupts

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes:

INT- Interrupt instruction with type number. It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

Its execution includes the following steps:

- ⌘ Flag register value is pushed on to the stack.
- ⌘ CS value of the return address and IP value of the return address are pushed on to the stack.
- ⌘ IP is loaded from the contents of the word location 'type number' x 4
- ⌘ CS is loaded from the contents of the next word location.
- ⌘ Interrupt Flag and Trap Flag are reset to 0.

The starting address for type0 interrupt is 000000H, for type1, interrupt is 00004H similarly for type2 is 00008H and ... so on. The first five pointers are dedicated interrupt pointers i.e.,

- ⌘ TYPE 0 interrupt represents division by zero situation.
- ⌘ TYPE 1 interrupt represents single-step execution during the debugging of a program.
- ⌘ TYPE 2 interrupt represents non-maskable NMI interrupt.
- ⌘ TYPE 3 interrupt represents break-point interrupt.
- ⌘ TYPE 4 interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

INT 3-Break Point Interrupt Instruction

It is a 1-byte instruction having op-code is CCH. These instructions are inserted into the program so that when the processor reaches there, then it stops the normal execution of program and follows the break-point procedure.

Its execution includes the following steps:

- ⌘ Flag register value is pushed on to the stack.
- ⌘ CS value of the return address and IP value of the return address are pushed on to the stack.

- IP is loaded from the contents of the word location 3x4 = 0000CH
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0.

INTO - Interrupt on overflow instruction

It is a 1-byte instruction and their mnemonic INTO. The op-code for this instruction is CEH. As the name suggests it is a conditional interrupt instruction, i. e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

Its execution includes the following steps:

- Flag register values are pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of word location 4x4 = 00010H
- CS is loaded from the contents of the next word location.
- Interrupt flag and Trap flag are reset to 0.

2.4 8086 – Addressing Modes

The different ways in which a source operand is denoted in an instruction is known as **addressing modes.** Different addressing modes are required in programs for processing implied numbers, constants, variables, and arrays and hence the flexibility to the programmer to access the operand from memory in different ways.

There are 7 different addressing modes in 8086 programming:

- Register Addressing Mode
- Immediate Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Base plus Index Addressing Mode
- Register Relative Addressing Mode
- Base Relative Plus Index Addressing Mode

1. Register Addressing Mode

It is the most common form of data addressing.

Transfers a copy of a byte/word from source register to destination register.

Instruction	Source	Destination
Mov AX,BX	Register BX	Register AX

It is carried out with 8 bit registers AH, AL, BH, BL, CH, CL, DH & DL or with 16 bit registers AX, BX, CX, DX, SP, BP, SI and DI.

It is important to use registers of same size.

Never mix an 8 bit register with a 16 bit register i. e. MOVAX,BL

EXAMPLES

MOV AL,BL :Copys BL into AL

MOV ES,DS :Copys DS into ES

MOV AX,CX :Copys CX into AX

2. Immediate Addressing Mode

The term immediate implies that the data immediately follow the hexadecimal opcode in the memory.

Note that immediate data are constant data.

It transfers the source immediate byte/word of data in destination register or memory location.

Instruction	Source	Destination
MOV CH,3AH	DATA3AH	Register AX

EXAMPLES

MOV AL,90 : Copys 90 into AL

MOV AX, 1234H : Copys1234HintoAX

MOV CL,10000001B : Copys 100000001 binary value into CL

3. Direct Addressing Mode

In this scheme, the address of the data is defined in the instruction itself.

When a memory location is to be referenced, its offset address must be specified

Instruction	Source	Destination
MOV AL,[1234H]	Assume DS=1000h 10000h+1234h = 11234h Memory location 11234h	Register AL

EXAMPLES

MOV AL, [1234H] : Copys the byte content of data segment memory location 11234H into AL.

MOV AL, NUMBER : Copys the byte content of data segment memory location NUMBER into AL.

4. Register Indirect Addressing Mode

Register Indirect Addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI and SI.

The Index and Base registers are used to specify the address of data.

It transfers byte/word between a register and a memory location addressed by an index or base registers.

The symbol [ ] denote indirect addressing.

The data segment is used by default with register in direct addressing or any other addressing mode that uses BX, DI or SI to address memory. If BP register addresses memory, the stack segment is used by default.

Instruction	Source	Destination
MOV CL,[BX]	ASSUME DS=1000H ASSUMEBX=0300H 10000H+0300H 10300H MEMORY LOCATION 10300H	Register CL

EXAMPLES

MOV CX,[BX] : Copys the word contents of the data segment memory location addressed by BX into CX.

MOV [DI],BH : Copys BH into the data segment memory location addressed by DI.

MOV [DI],[BX] : Memory to Memory moves are not allowed except with string instructions.

5. Base plus Index Addressing Mode

Base plus index addressing is similar to indirect addressing be cause it indirectly addresses memory data

This type of addressing uses one base register (BP or BX) and one Index Register (DI or SI) to indirectly address memory.

Instruction	Source	Destination
MOV [BX+SI],CL	REGISTER CL	ASSUME DS=1000H ASSUMEBX=0300H ASSUME SI=0200H 10000H+0300H+0200H = 10500H MEMORY LOCATION 10500H

EXAMPLES

MOV CX,[BX+DI] : Copys the word contents of the data segment memory location addressed by BX plus DI into CX.

MOV CH,[BP+SI] : Copys the byte contents of the stack segment memory location addressed by BP plus SI into CH

6. Register Relative Addressing Mode

In this case, the data in a segment of memory are addressed by adding the displacement to the content of base or an index register (BP, BX, DI or SI).

Transfers a byte/word between a register and the memory location addressed by an index or base register plus a displacement.

INSTRUCTION	SOURCE	DESTINATION
MOV [BX+4],CL	REGISTERCL	ASSUME DS=1000H ASSUME BX=0300H 10000H+0300H+4H 10304H MEMORY LOCATION 10304H

EXAMPLES

MOV ARRAY[SI],BL : Copys BL into the data segment memory location addressed by ARRAY plus SI.

MOV LIST[SI+2],CL : Copys CL into the data segment memory location addressed by sum of LIST, SI and 2.

7. Base Relative plus Index Addressing Mode

The base relative plus index addressing mode is similar to the base plus index addressing mode but it adds a displacement to form a memory address.

Transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement.

Instruction	Source	Destination
MOV[BX+SI+05], CL	Register CL	Assume DS =1000H assume BX=0300H Assume SI=0200H 10000H+0300H+0200H+05H=10505H Memory Location 10505H

EXAMPLES

MOV LIST[BP+DI],CL : Copys CL into the stack segment memory location addressed by the sum of LIST, BP and DI

MOV DH,[BX+DI+20H] : Copys the byte contents of the data segment memory location addressed by the sum of BX, DI and 20 Hint oDH

Pipelining

Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in **stages.** Each stage completes a part of an instruction in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end.

Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput. The **throughput** of the instruction pipeline is determined by how often an instruction exits the pipeline.

Because the pipe stages are hooked together, all the stages must be ready to proceed at the same time. We call the time required to move an instruction one step further in the pipeline ***a machine cycle.*** The ***length*** of the machine cycle is determined by the time required for the slowest pipe stage.

The pipeline designer's *goal is to balance the length of each pipeline stage.* If the stages are perfectly balanced, then the time per instruction on the pipelined machine is equal to

Time per instruction on nonpipelined machine Number of pipe stages

Under these conditions, the speedup from pipelining equals the number of pipe stages. Usually, however, the stages will not be perfectly balanced; besides, the pipelining itself involves some overhead.

Instruction Pipeline

Any architecture can be pipelined by making each clock cycle into a pipe stage.

Clock#	1	2	3	4	5	6	7
Instruction *i*	Fetch	Decode	Execute				
Instr. *i* + 1		Fetch	Decode	Execute			
Instr. *i* + 2			Fetch	Decode	Execute		
Instr. *i* + 3				Fetch	Decode	Execute	
Instr. *i* + 4					Fetch	Decode	Execute

Table 2.1: Instruction Pipeline

Here instruction i is fetched in clock #1. After it has been fetched it is decoded in clock #2 and at the same time next instruction i. e. instr. i+1 is fetched. At Clock#3, instruction i is executed, instr. i+1 is decoded and instr. i+2 is fetched and so on.

Hence at the end of clock #3, instruction i is executed. Sometime later at the end of clock #4 instruction i+1 is executed.

If we assume that unpipelined architecture took 3 ns then this pipelined architecture will take 1 ns to finish a stage (fetch, decode and execute).

Advantages of pipelining:

- The execution unit always reads the next instruction byte from the queue in BIU. This is faster than sending out an address to the memory and waiting for the next instruction byte to come.
- In short pipelining eliminates the waiting time of EU and speeds up the processing. The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in queue. 8086 BIU normally obtains two instruction bytes per fetch.

QUESTIONS

1. Explain 8085 architecture with the help of its block diagram.
2. Why addressing modes are required in microprocessor? Explain the addressing modes of 8085 architecture with examples.
3. Why flags are required in microprocessor? Explain 8085 flags with suitable facts and figures.
4. Draw a neat pin diagram of 8085 microprocessor and explain it.
5. Explain 8086 architecture with the help of its EU and BIU.
6. What is the advantage of having more addressing modes in 8086 microprocessor? Explain the addressing modes of 8086 architecture with examples.
7. Why is flags? Explain 8086 flags with suitable facts and figures.
8. What is segmented memory? List out the advantages and disadvantages of segmentation.
9. What is pipeline? Explain instruction pipeline in brief.

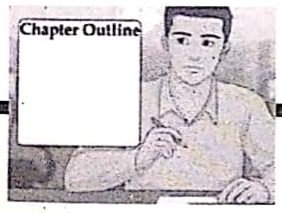
❑❑❑

3

INSTRUCTION CYCLE

CHAPTER OBJECIVE

Instruction Cycle is a process to explain instruction execution which have three steps called Fetch, Decode and Execute. This chapter explains the instruction execution process using block diagrams and timing diagrams. The timing diagrams are made of using T-states, machine cycles and instruction cycle. The last section of this chapter illustrates the concept of memory interfacing using address decoding concept.



CHAPTER OUTLINE

- Instruction cycle, Machine Cycle and T-States
  - Machine Cycle of 8085 Microprocessor: op-code fetch, memory read, memory write, I/O read, I/O write, interrupt
- Fetch and Execute Operation, Timing Diagram
  - Timing Diagram of MOV, MVI, IN, OUT, LDA, STA
- Memory Interfacing and Generation of Chip Select Signal

3.1 Instruction cycle, Machine Cycle and T-States

Instruction Cycle

The necessary steps that the CPU carries out to fetch an instruction and necessary data from the memory and to execute it constitute an instruction cycle. Moreover, it is defined as the time required to complete the execution of an instruction.

An instruction cycle consists of fetch cycle and execute cycle. In fetch cycle CPU fetches opcode from the memory. The necessary steps which are carried out to fetch an opcode from memory constitute a fetch cycle. The necessary steps which are carried out to get data if any from the memory and to perform the specific operation specified in an instruction constitute an execute cycle. The total time required to execute an instruction is given by IC = Fc + Ec. The 8085 consists of 1-6 machine cycles or operations.

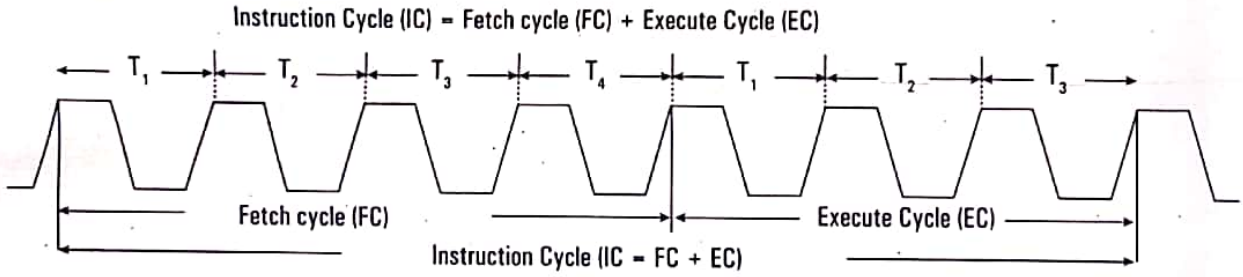


Figure 3.1: Timing diagram of instruction cycle

Fetch cycle

The first byte of an instruction is its opcode. The program counter keeps the memory address of the next instruction to be executed in the beginning of fetch cycle. The content of the program counter, which is the address of the memory location where opcode is available, is send to the memory. The memory places the opcode on the data bus so as to transfer it to CPU. The entire process takes 3 clock cycle and then the instruction is decoded in next one clock cycle.

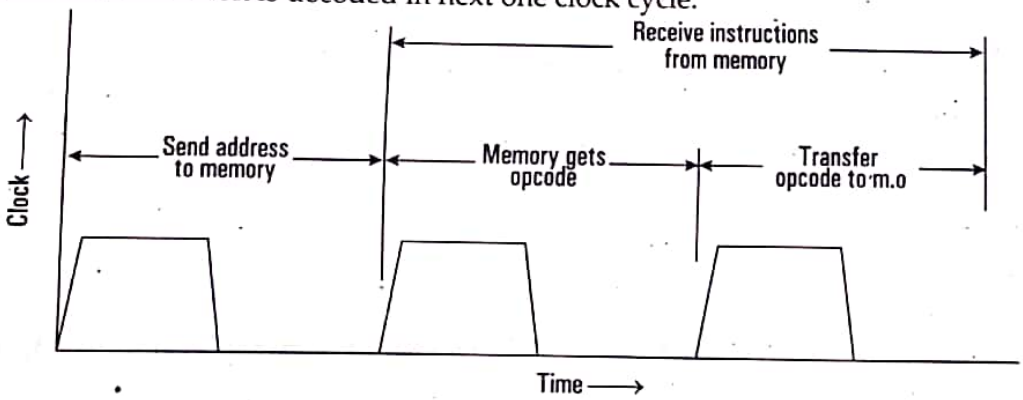


Figure 3.2: Timing diagram of fetch cycle

Execute cycle/Operation:

The opcode fetched from the memory goes to IR from the IR it goes to the decoder which decodes instruction. After the instruction is decoded execution begins.

- If the operand is in general purpose register, execution is performed immediately i,e., in one clock cycle.
- If an instruction is contains data or operand address, then CPU has to perform some read operations to get the desired data.

- In some instruction write operation is performed. In write cycle data are sent from the CPU to the memory of an o/p device.
- In some cases execute cycle may involve one or more read or write cycle or both.

Machine cycle

It is defined as the time required to complete one operation of accessing memory i/p, o/p or acknowledging and external request. This cycle may consists of 3 to 6 T states.

T-states: It is defined as one sub division of the operation performed in one clock period. These sub division are internal states synchronized with system clock and each T states precisely equal to one clock period.

Machine cycles of 8085

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

- Opcode fetch cycle (4T)
- Memory read cycle (3 T)
- Memory write cycle (3 T)
- I/O read cycle (3 T)
- I/O write cycle (3 T)
- Interrupt

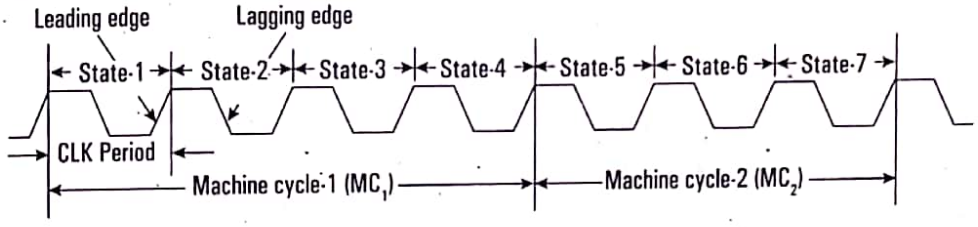


Figure 3.3: Timing diagram of machine cycle

Machine Cycle	Status			Controls		
	IO/$\overline{M}$	S_1	S_0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
Opcode Fetch (OF)	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/O Read (I/OR)	1	1	0	0	1	1
I/O Write (I/OW)	1	0	1	1	0	1
Acknowledge of INTR (INTA)	1	1	1	1	1	0
BUS Idle (BI): DAD	0	1	0	1	1	1
ACK of RST, TRAP	1	1	1	1	1	1
HALT	Z	0	0	Z	Z	1
HOLD	Z	X	X	Z	Z	1
X ⇒ Unspecified, and Z ⇒ High impedance state						

Table 3.1: Machine cycle status and control signals

1. Opcode Fetch Cycle

The first machine cycle of every instruction is opcode fetch cycle in which the 8085 finds the nature of the instruction to be executed. In this machine cycle, processor places the contents of the Program Counter on the address lines, and through the read process, reads the opcode of the instruction. Fig. 3. 4 shows flow of data (opcode) from memory to the microprocessor and Fig. 3. 5 shows the timing diagram for opcode fetch machine cycle. The length of this cycle is not fixed. It varies from 4T states to 6T states as per the instruction. The following section describes the opcode fetch cycle in step by step manner.

Step 1 : (State T_1) In T_1 state, the 8085 places the contents of program counter on the address bus. The high-order byte of the PC is placed on the A_8-A_{15} lines. The low-order byte of the PC is placed on the $AD_0 - AD_7$ lines which stays on only during T_1. Thus microprocessor activates ALE (Address Latch Enable) which is used to latch the low-order byte of the address in external latch before it disappears.

In T_1, 8085 also sends status signals IO/M, S_1, and S_0. IO/M specifies whether it is a memory or I/O operation, S_i status specifies whether it is read/write operation; S_1 and S_0 together indicates read, write, opcode fetch, machine cycle operation, or whether it is in HALT state. In opcode fetch machine cycle status signals are : IO/M = 0, S_1 = 1, S_0 = 1.

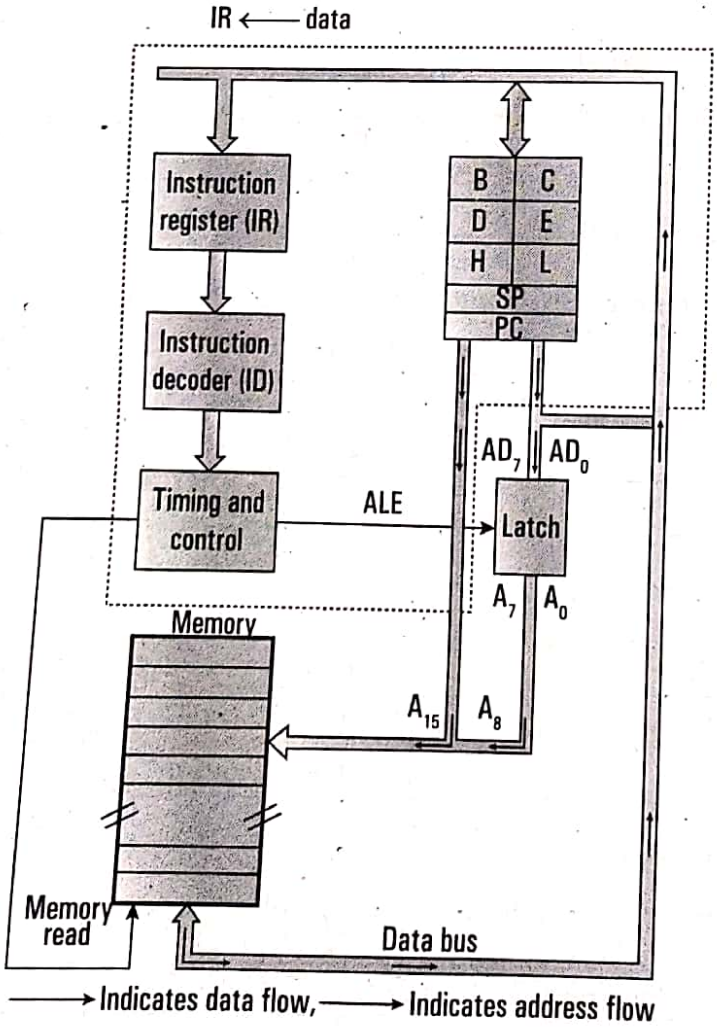


Figure 3.4: Block diagram of opcode fetch machine cycle

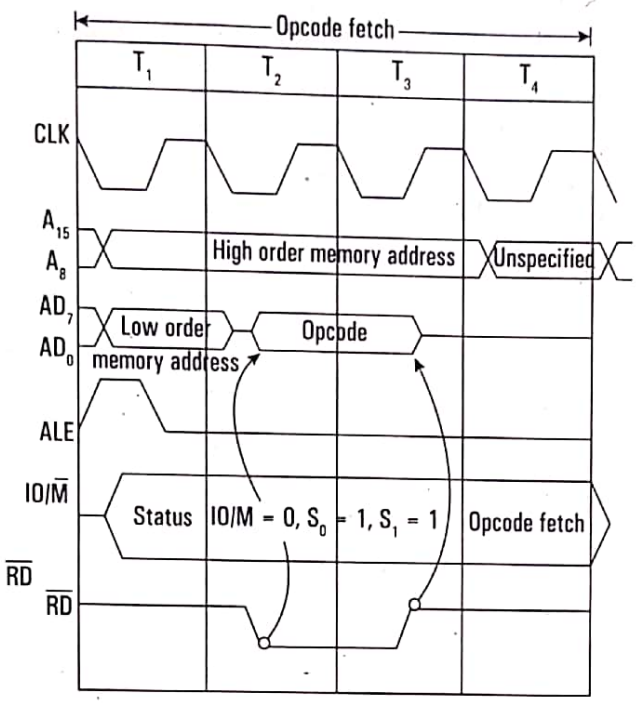


Figure 3.5: Timing diagram of opcode fetch machine cycle

Step 2 : (State T_2) In T_2, low-order address disappears from the $AD_0 - AD_7$ lines. (However $A_0 - A_7$ remain available as they were latched during T_1). In T_2, 8085 sends RD signal low to enable the addressed memory location. The memory device then places the contents of addressed memory location on the data bus ($AD_0 - AD_7$).

Step 3 : (State T_3) During T_3, 8085 loads the data from the data bus in its Instruction Register and raises RD to high which disables the memory device.

Step 4 : (State T_4) In T_4, microprocessor decodes the opcode, and on the basis of the instruction received, it decides whether to enter state T_5 or to enter state T_1 of the next machine cycle. One byte instructions those operate on eight bit data (8 bit operand) are executed in T_4.

For example : MOV A, B, ANA D, ADD B, INR L, DCR C, RAL and many more.

Note : *For one byte instructions which operate on eight bit data, data is always available in the internal memory of 8085 i. e. registers.*

Step 5 : (State T_5 and T_6) State T_5 and T_6, when entered, are used for internal microprocessor operations required by the instruction. During T_5 and T_6, 8085 performs stack write, internal 16 bit; and conditional return operations depending upon the type of instruction. One byte instructions those operate on sixteen bit data (16 bit operand) are executed in T_5 and T_6. For example DCX H, PCHL, SPHL, INX H, etc.

2. Memory Read Cycle

The 8085 executes the memory read cycle to read the contents of R/W memory or ROM. The length of this machine cycle is 3-T states ($T_1 - T_3$). In this machine cycle, processor places the address on the address lines from the stack pointer, general purpose register pair or program counter, and through the read process, reads the data from the addressed memory location. Fig. 3. 6 shows flow of data from memory to the microprocessor and Fig. 3.7 shows the timing diagram for memory read machine cycle. Memory read machine cycle is similar to the opcode fetch machine cycle. However, they use only states T_1 to T_3, and the status signal values (IO/M = 0, S_1 = 1, S_0 = 0) appropriate for memory read machine cycle are issued in T_1.

The following section describes the memory read machine cycle in step by step manner.

Step 1 : (State T_1) In T_1 state, microprocessor places the address on the address lines from stack pointer, general purpose register pair or program counter and activates ALE signal in order to latch low-order byte of address.

During T_1, 8085 sends status signals : IO/M = 0, S_1 = 1, and S_0 = 0 for memory read machine cycle.

Step 2 : (State T_2) In T_2, 8085 sends RD signal low to enable the addressed memory location. The memory device then places the contents of addressed memory location on the data bus (AD_0 -AD_7).

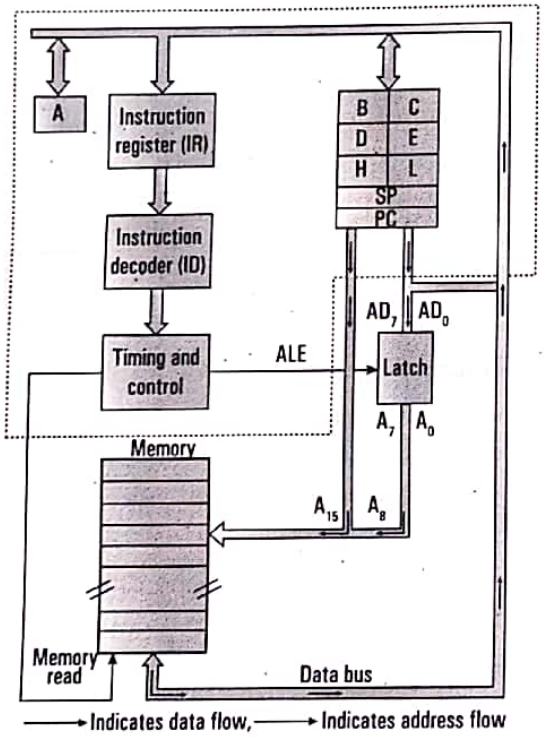


Figure 3.6: Block diagram of memory read operation

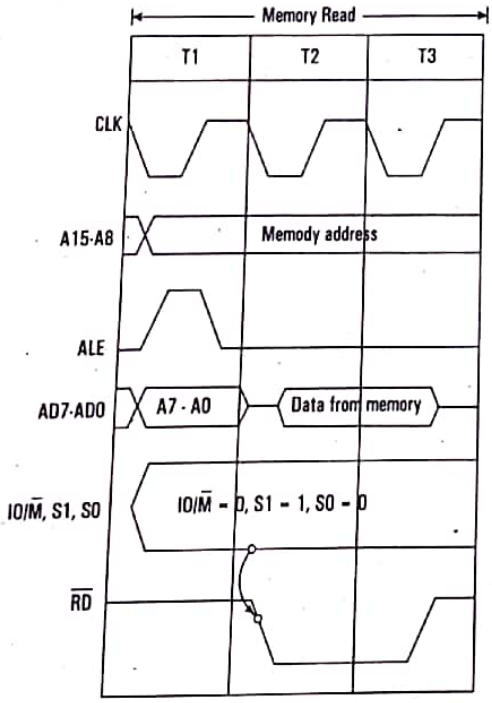


Figure 3.7: Timing diagram of memory read operation

Step 3 : (State T_3) During T_3, 8085 loads the data from the data bus into specified register (F, A, B, C, D, E, H, and L) and raises RD to high which disables the memory device.

3. Memory Write Cycle

The 8085 executes the memory write cycle to store the data into data memory or stack memory. The length of this machine cycle is 3T states ($T_1 - T_3$). In this machine cycle, processor places the address on the address lines from the stack pointer or general purpose register pair and through the write process, stores the data into the addressed memory location. Fig. 3.7 shows the timing diagram for memory write machine cycle. The memory write timing diagram is similar to the memory read timing diagram, except that instead of RD, WR signal goes low during T_2 and T_3. The status signals for memory write cycle are: IO/M = 0, S_1 = 0, S_0 = 1. The following section describes the memory write machine cycle in step by step manner.

Step 1 : (State T_1) In T_1 state, the 8085 places the address on the address lines from stack pointer or general purpose register pair and activates ALE signal in order to latch low-order byte of address. During T_1, 8085 sends status signals : IO/M = 0, S_1 = 0 and S_0 = 1 for memory write machine cycle.

Step 2 : (State T_2) In T_2, 8085 places data on the data bus and sends WR signal low for writing into the addressed memory location.

Step 3 : (State T_3) During T_3, WR signal goes high, which disables the memory device and terminates the write operation.

I/O Read and I/O Write cycles

The I/O read and I/O write machine cycles are similar to the memory read and memory write machine cycles, respectively, except that the IO/$\overline{M}$ signal is high for I/O read and I/O write machine cycles. High IO/M signal indicates that it is an I/O operation. Fig. 3.9 and Fig. 3.11 shows the timing diagrams for I/O read and I/O write cycles, respectively.

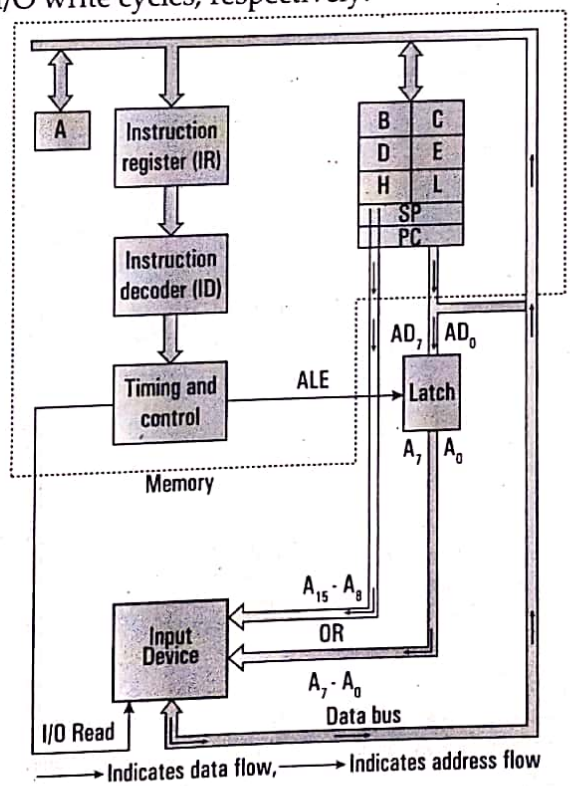


Figure 3.8: Block diagram of input output read operation.

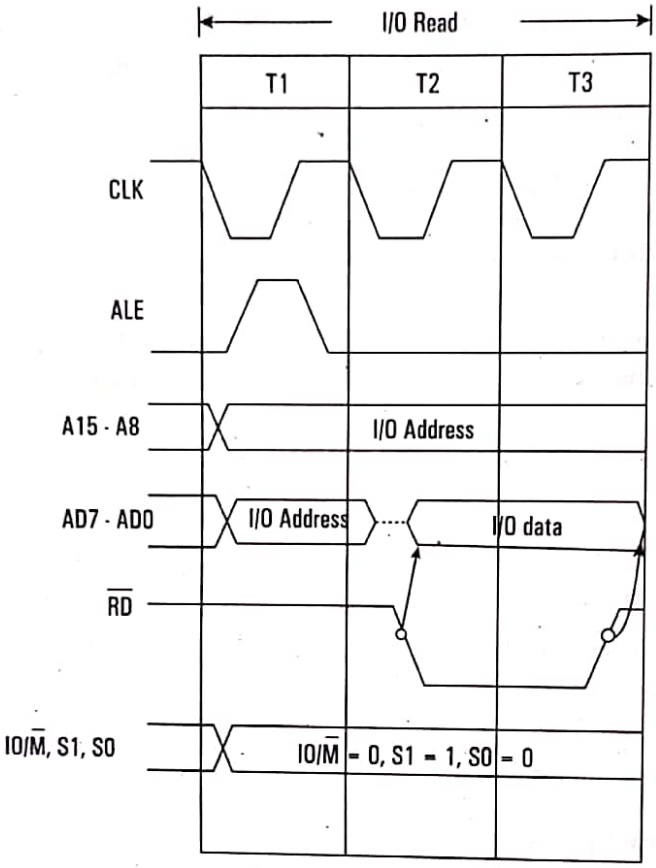


Figure 3.9: Timing diagram of input output read operation.

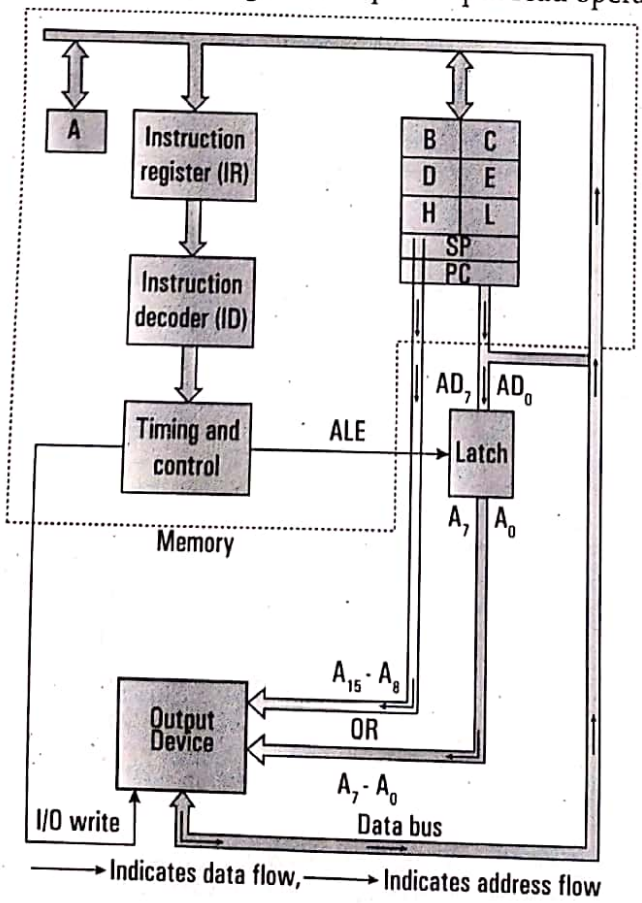


Figure 3.10: Block diagram of input output write operation

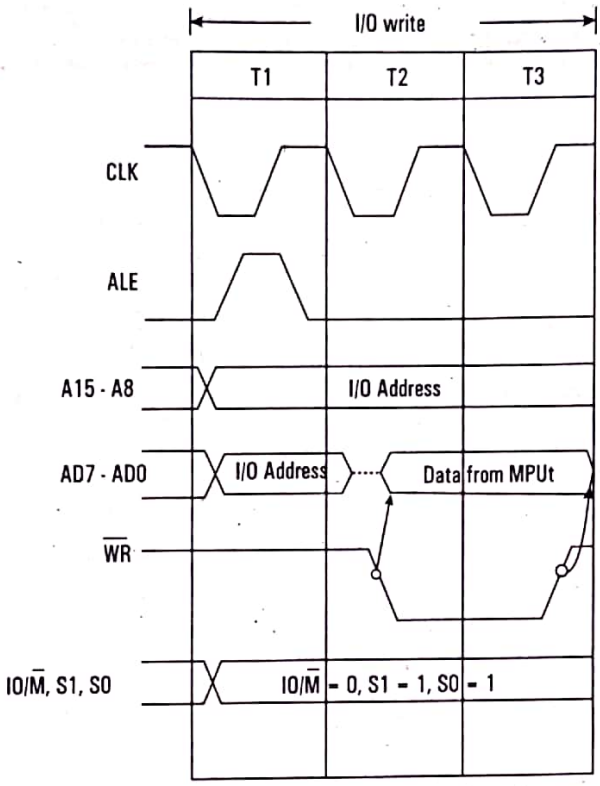


Figure 3.11: Timing diagram of input output write operation

4. Interrupt Acknowledge Cycle

In response to INTR signal, 8085 executes interrupt acknowledge machine cycle to read an instruction from the external device. Theoretically, the external device can place any instruction on the data bus in response to INTA. However, only RST and CALL, save the PC contents (return address) before transferring control to the interrupt service routine. The next sections explain interrupt acknowledge cycles for RST and CALL instructions.

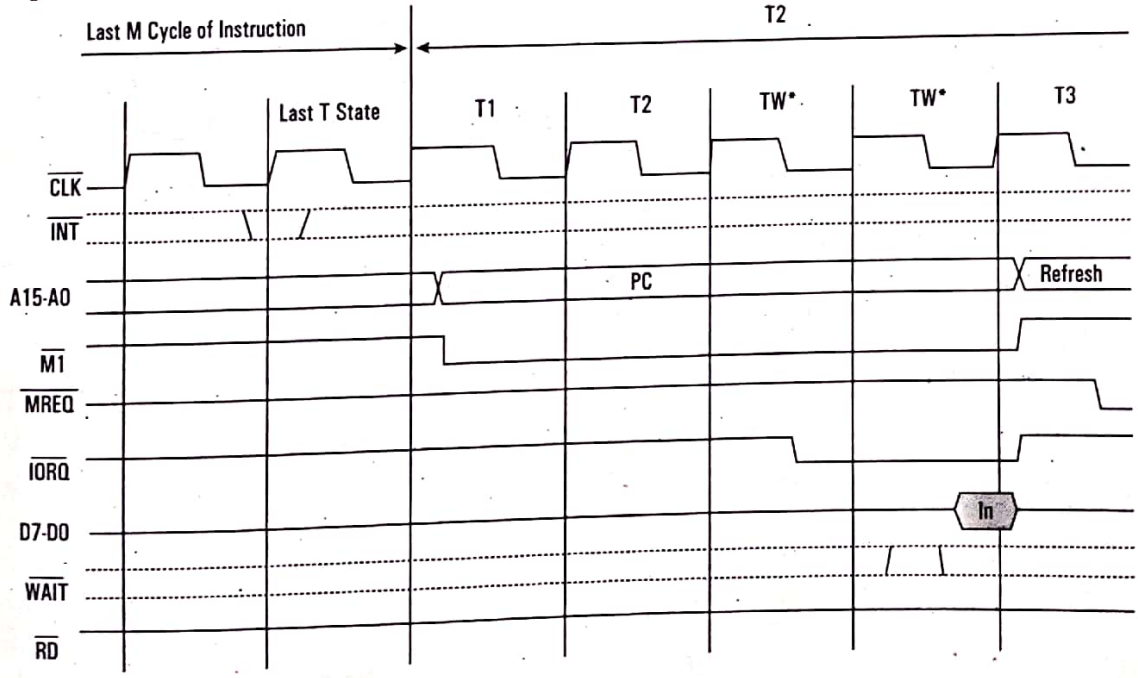


Figure 3.12: Timing diagram of interupt acknowledge cycle.

3.2 Fetch and Execute Operation Timing Diagrams

1. MOV A, B

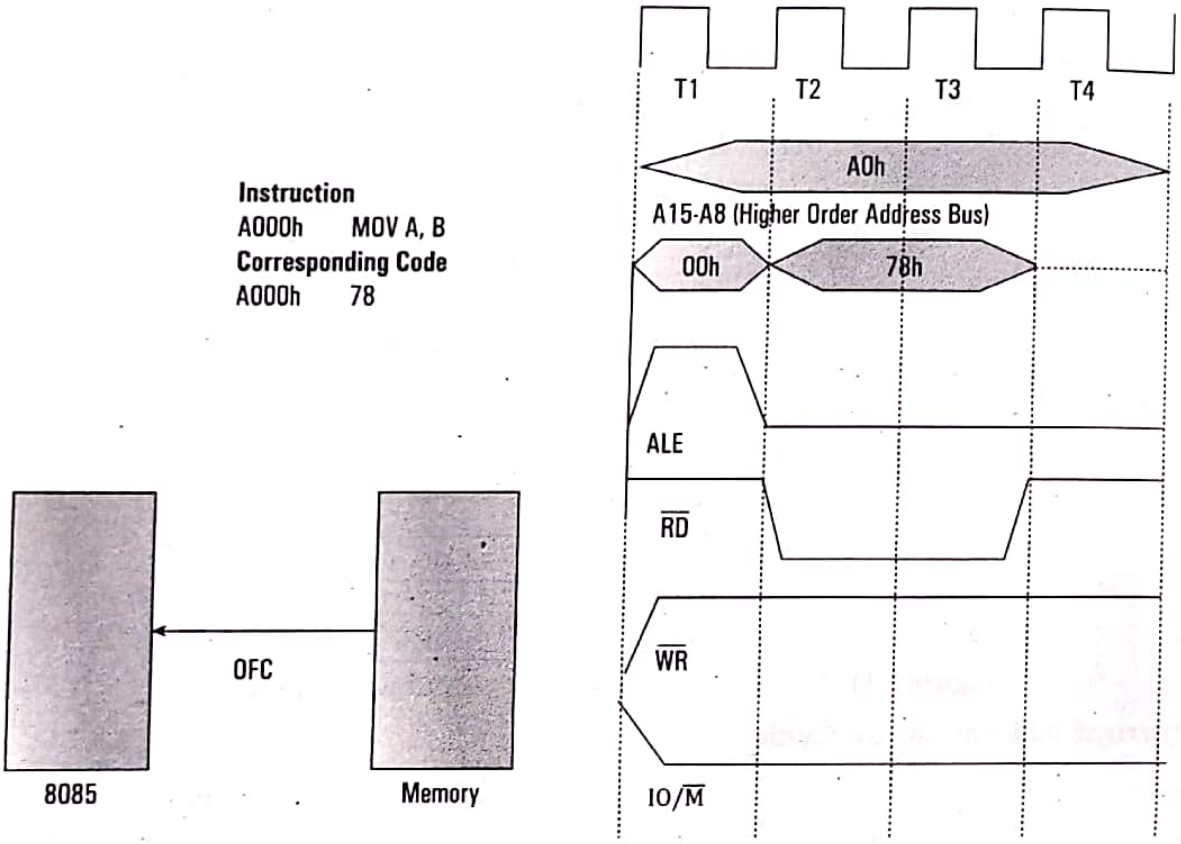


Figure 3.13: Timing diagram of instruction MOV A, B.

2. MOV A, M

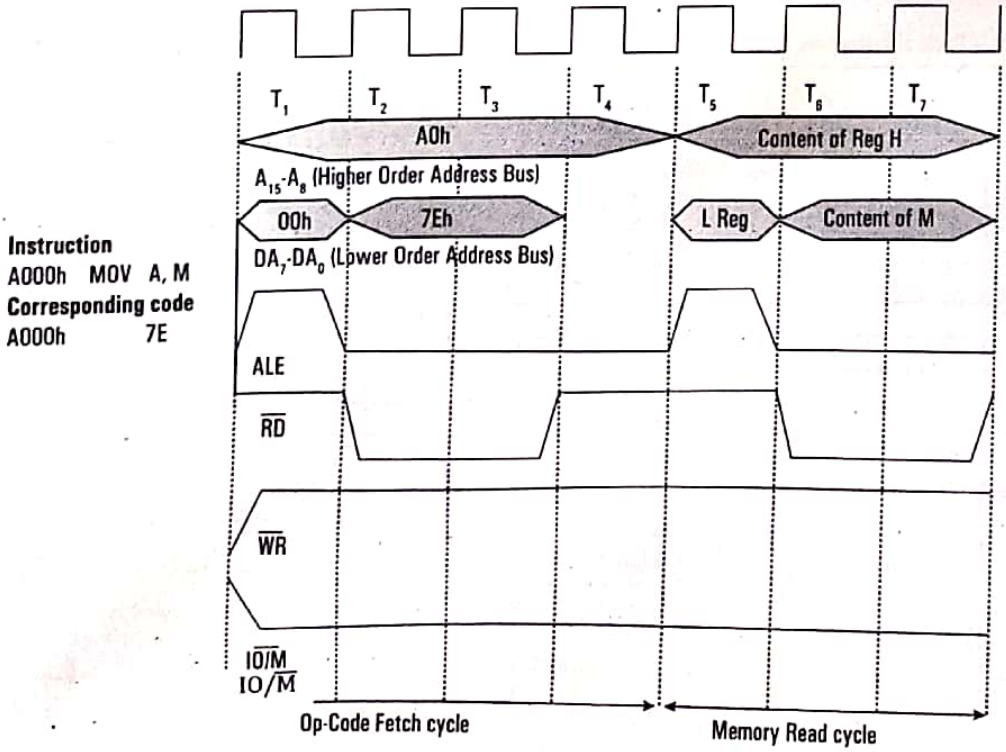


Figure 3.14: Timing diagram of instruction MOV A, M.

3. MOV M, A

Instruction
A000h MOV M, A
Corresponding code
A000h 77t

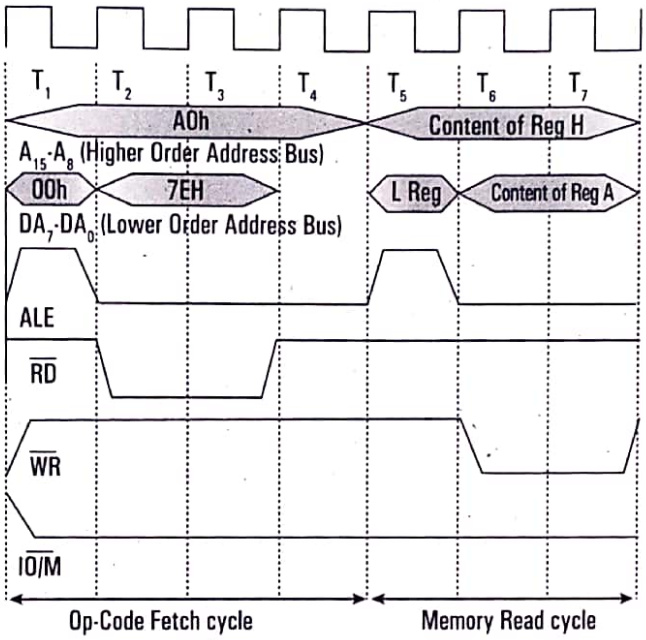


Figure 3.15: Timing diagram of instruction MOV M, A

4. MVI A, 45H

Instruction
A000h MVIA 45h
Corresponding code
A000h 3E

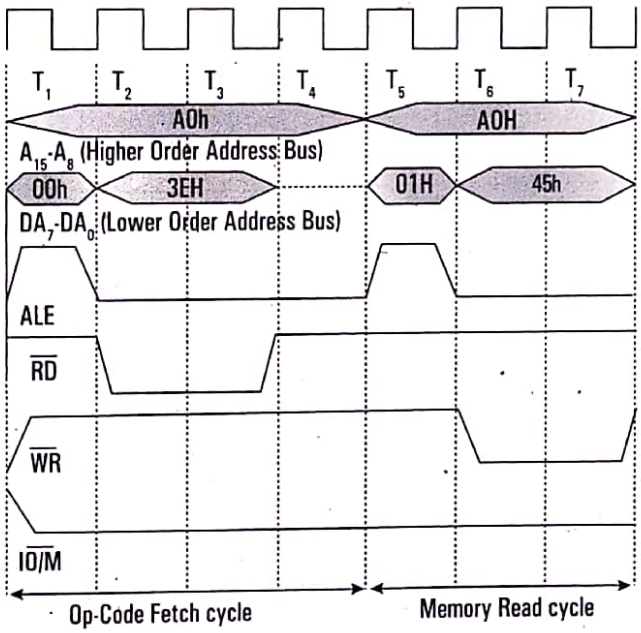


Figure 3.16: Timing diagram of instruction MOV A, 45H.

5. STA 526AH

- STA means Store Accumulator- The contents of the accumulator is stored in the specified address(526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see figure of machine cycle).
- Then the lower order memory address is read(6A). - *Memory Read Machine Cycle*
- Read the higher order memory address (52). - *Memory Read Machine Cycle*
- The combination of both the addresses are considered and the content from accumulator is written in 526A. - *Memory Write Machine Cycle*

⌘ Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

Address	Mnemonics	Op code
41FF	STA 526AH	32H
4200		6AH
4201		52H

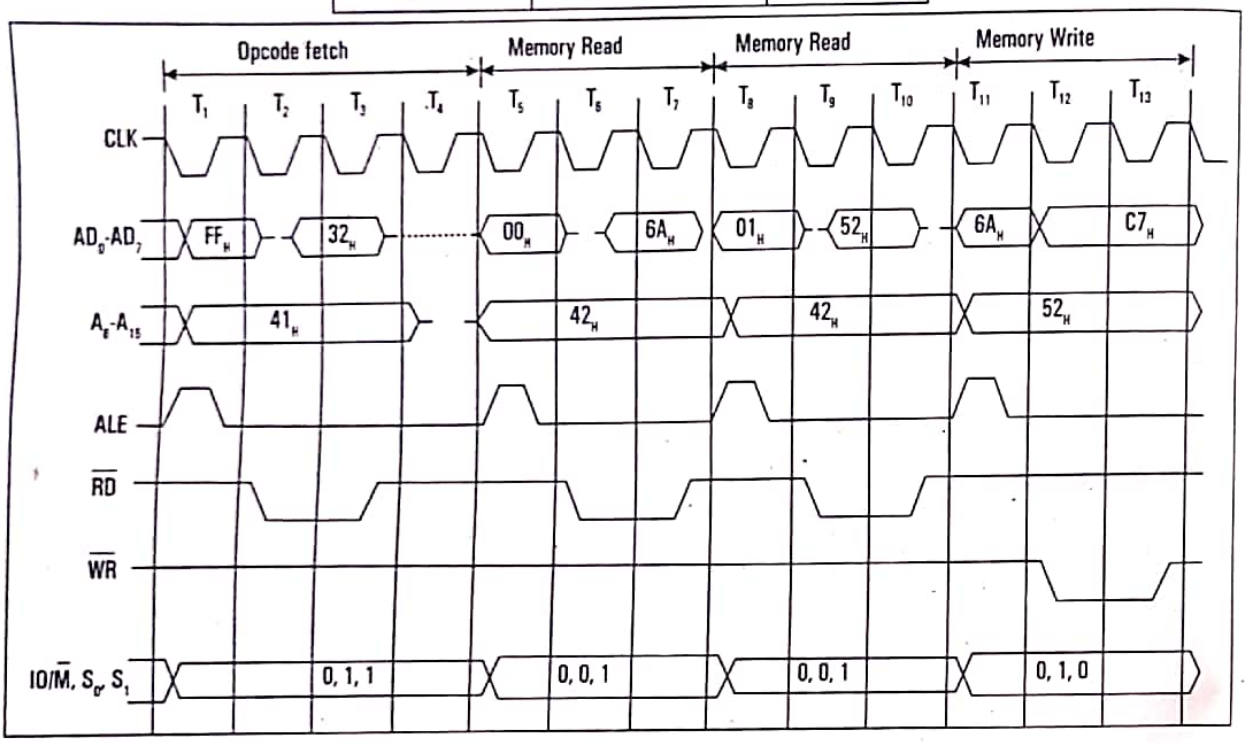


Figure 3.17: Timing diagram of instruction STA 526AH.

6. LDA XXXXH

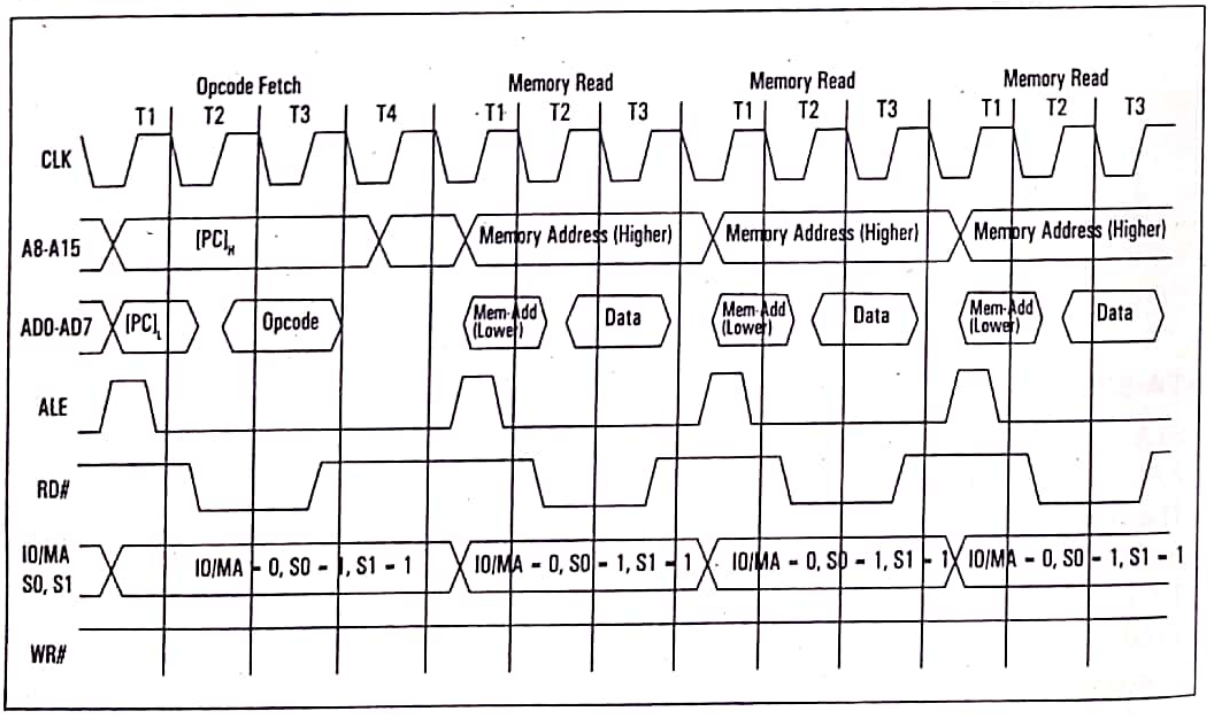


Figure 3.18: Timing diagram of instruction LDA XXXXH.

7. IN CO H

- Fetching the Opcode DBH from the memory 4125H.
- Read the port address C0H from 4126H.
- Read the content of port C0H and send it to the accumulator.
- Let the content of port is 5EH.

Address	Mnemonics	Op code
4125	IN CO_H	DB_H
4126		CO_H

Instruction:
IN COH

IN
Opcode : DBH
Address: 4125H

COH
Address: 4126H

Content of the port
COH: 5EH

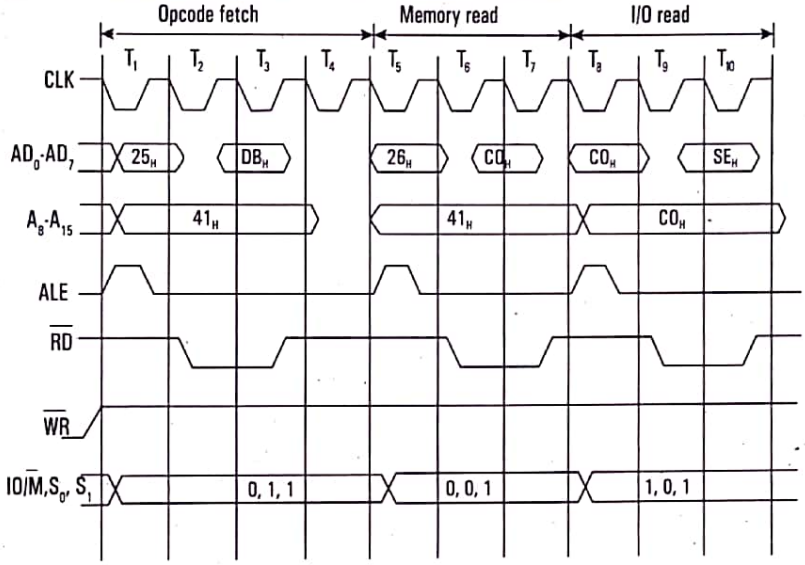


Figure 3.19: Timing diagram of IN COH

8. OUT XXH

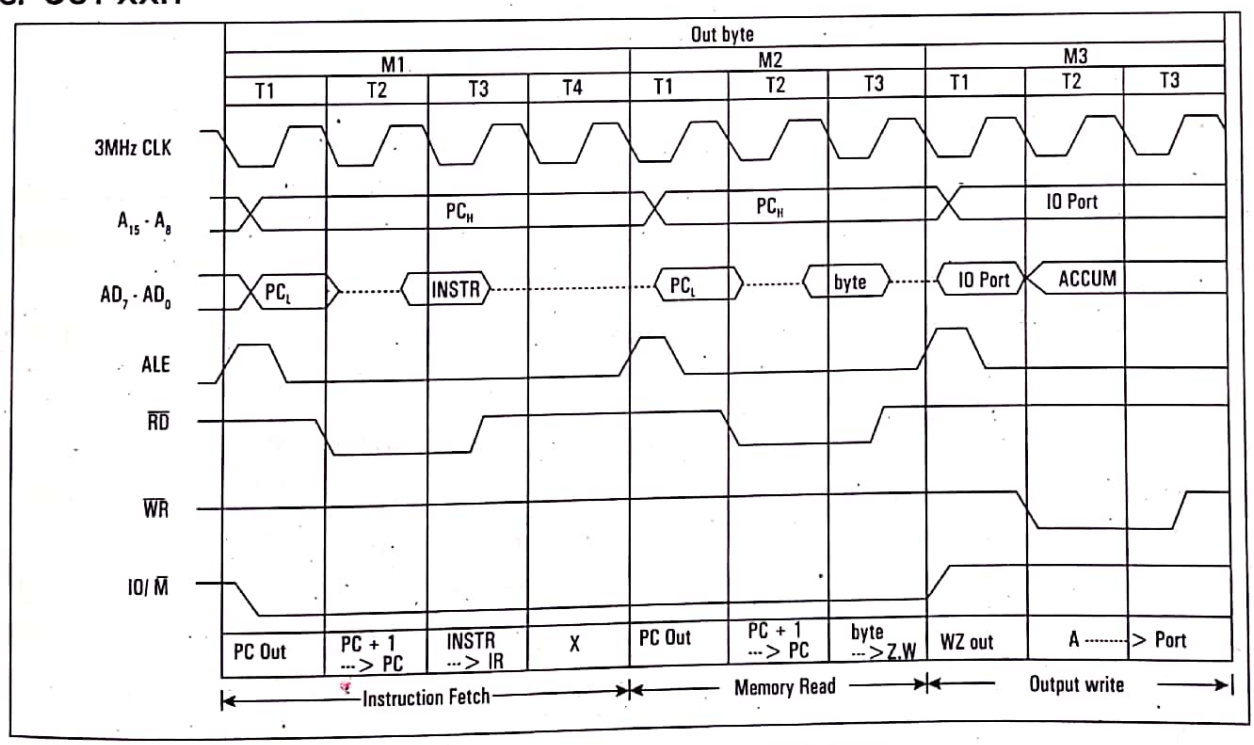


Figure 3.20: Timing diagram of OUT XXH

3.3 Memory Interfacing and Generation of Chip Select Signal

A microprocessor has to be interfaced with various peripherals to perform various functions. Let's discuss about the interfacing techniques in detail.

- ⌘ We know that a microprocessor is the CPU of a computer.
- ⌘ A microprocessor can perform some operation on a data and give the output. But to perform the operation we need an input to enter the data and an output to display the results of the operation.

Interfacing Types

There are two types of interfacing in context of the 8085 processor.

a. Memory interfacing

b. I/O interfacing

a. Memory interfacing

While executing an instruction, there is a necessity for the microprocessor to access memory frequently for reading various instruction codes and data stored in the memory. The interfacing circuit aids in accessing the memory.

But what is the purpose of interfacing circuit here?

The interfacing process involves matching the memory requirements with the microprocessor signals. The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirements with the signals of the microprocessor.

b. I/O interfacing

We know that keyboard and displays are used as communication channel with outside world. So it is necessary that we interface keyboard and displays with the microprocessor. This is called I/O interfacing. In this type of interfacing we use latches and buffers for interfacing the keyboards and displays with the microprocessor.

Basic Concepts of Memory Interfacing

The programs and data that are executed by the microprocessor have to be stored in ROM/EPROM and RAM, which are basically semiconductor memory chips.

Microprocessor need to access memory quite frequently to read instructions and data stored in memory, the interface circuit enables that access.

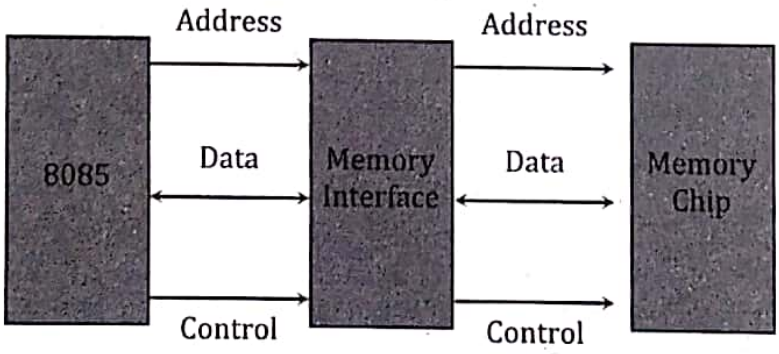


Figure 3.21: 8085 interfacing with memory chips.

The interface process involves designing a circuit that will match the memory requirements with the microprocessor signal.

Memory has certain signal requirements to read from and write into memory. Similarly Microprocessor initiates the set of signals when it wants to read from and write into memory.

- 8085 has 16 address lines (AO – A15), hence a maximum of 64 KB (= 216 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/M, RD' and 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), OE or RD' (output enable or read) and WE or 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 IOM, RD and WR signals as shown. Status of IQ/M, RD' and WR' signals during memory read and write operations

IOM'	RD'	WR'	Operation
0	0	1	8085 reads data from memory
0	1	0	8085 writes data into memory

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

Generating Control Signals

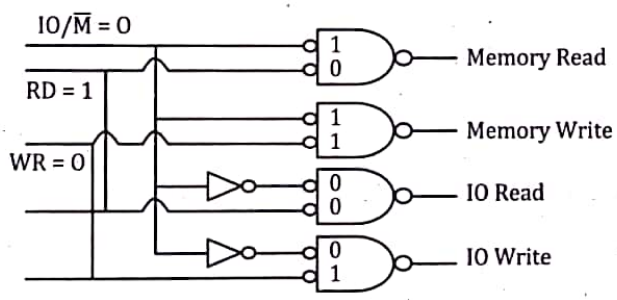


Figure 3.22: Generating Control Signals

- 8085 places 16-bit address on address bus and with this address only one register should be selected (only 11 low order address lines are required).
- Remaining 8085 address lines (A15-A11) should be decoded to generate chip select.
- 8085 provides two signal-IOM' and RD'-to indicate that is memory read operation MEMR'. (Similarly signal-IO/M' and WR'- indicates memory write operation MEMW').

Primary function of memory interfacing is that the microprocessor should be able to read from and write into a given register of a memory chip:

- Select the chip
- Identify the register
- Enable the appropriate buffer.

Memory Structure and its Requirements

Structure of R/W Memory (RAM)

- 2048 registers
- Register store 8-bits
- 8 input, 8-output lines
- 11 address lines (AD10-AD0), 1 chip select, 2 control lines to enable input and output buffer.
- Internal decoder to decode address lines.

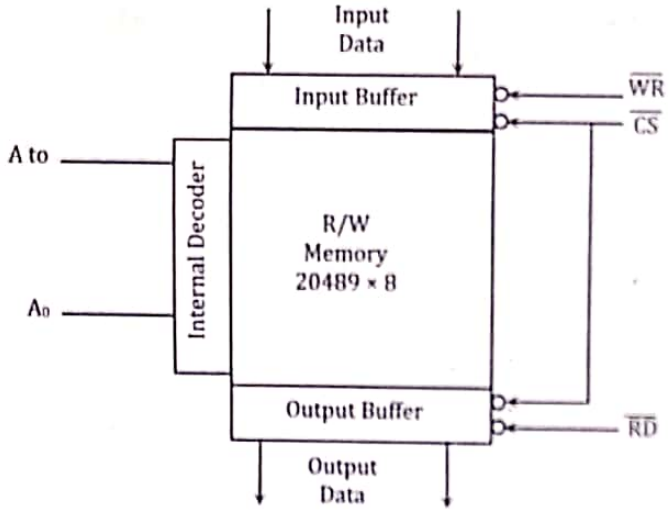


Figure 3.23: MEMR and MEMW are given to RD and WR pins of Memory chip

Structure of ROM:

Generally EPROMS are used as program memory and RAM as data memory.

- EPROM: Chip must be programmed before it can be used as ROM.
- 4096 (4K) registers
- Register store 8-bits
- 8 input lines
- Internal decoder to decode address lines.
- 12 address lines (A11-A0), 1 chip select, 1 Read control Signal lines to enable output buffer.

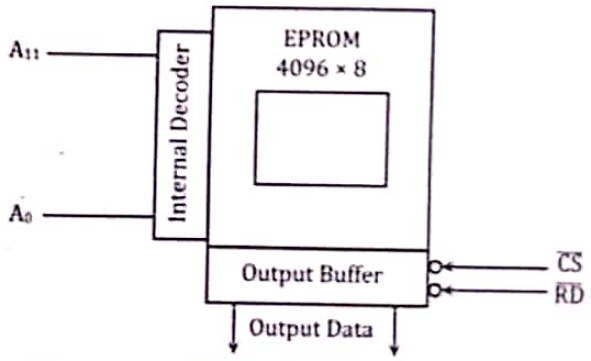


Figure 3.24: 8085 Interfacing with memory.

The following are the steps involved in interfacing memory with 8085 processor.

- First decide the size of memory requires to be interfaced. Depending on this we can say how many address lines are required for it.

For example if you want to interface 4KB memory it requires 12 address lines. Remaining 4 address lines can be used in address decoding.

- Depending on the size of memory required and given address range, construct address decoding circuity. This address decoding circuity can be implemented with NAND gates or decoders.
- Connect data bus of memory to processor data bus.
- Generate the control signals required for memory using IO/M', WR', RD' signals of 8085 processor.

Example:

- Interface 4KB memory to 8085 with starting address A000H.
- 4KB memory requires 12 address lines for addressing and 4 address lines are used for address decoding.
- Given that starting address for memory is A000H. So for 4KB memory ending address becomes A000H + 0FFFH (4KB) = AFFFH.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

A0-A11 address lines are directly connected to address bus of memory chip.

A12-A15 are used for generating chip select signal for memory chip.

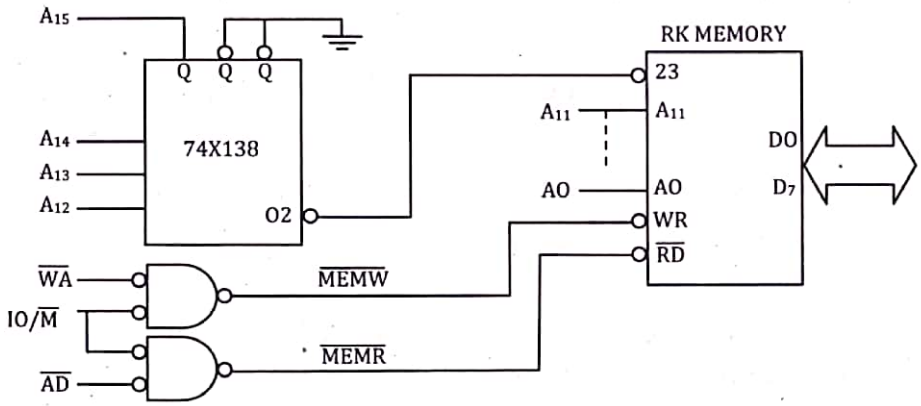


Figure 3.25: Address decoding circuit using 3×8 decoder

A15 line is use for enabling 74×138 decoder chip. A12, A13, A14 lines are connected to 74X138 chip as inputs. When theses lines are 010 output should be '0'. This is provided at O2 pin of 74X138 chip.

Address decoding circuit using only NAND gates:

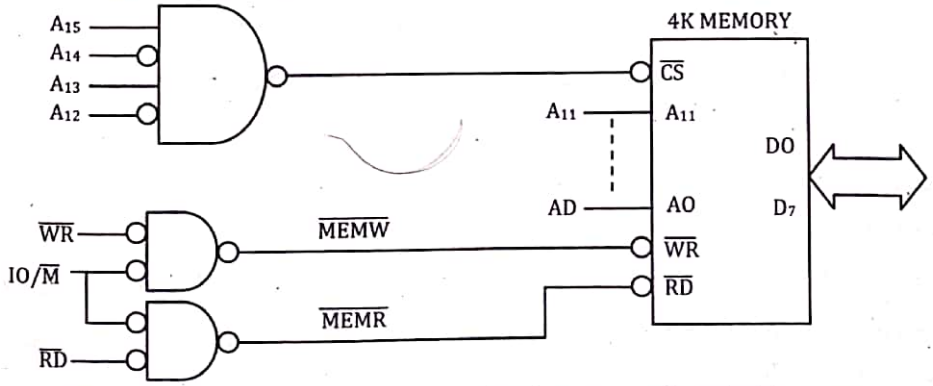


Figure 3.26: Address decoding circuit using only NAND gates

A15, A14, A13, A12 inputs should be 1010, for enabling the chip. So the circuit for this is as shown above.

8085 Interfacing Circuit to interface EPROM (using 3x8 Decoder)

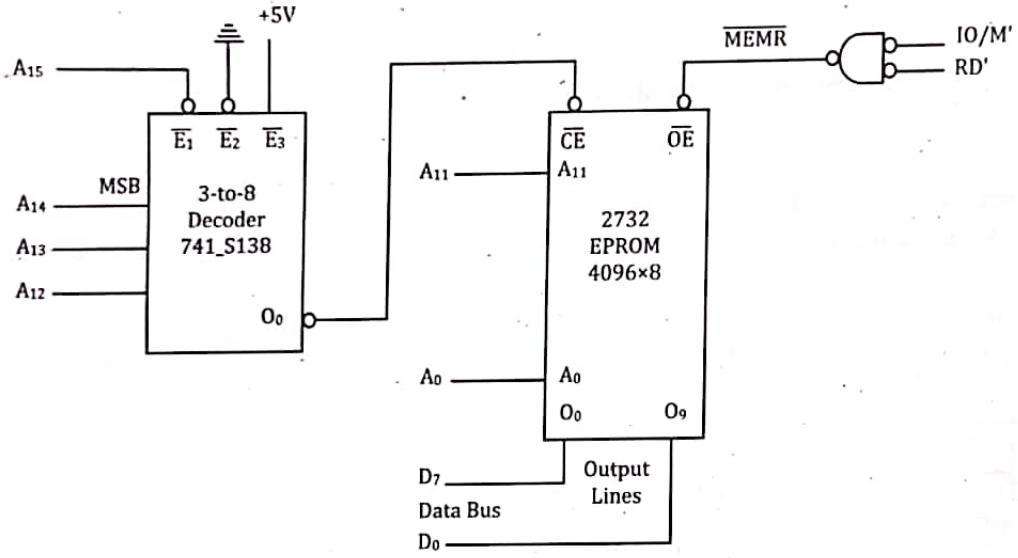


Figure 3.27: 8085 Interfacing Circuit to interface EPROM (using 3x8 Decoder)

- The 8085 address lines A11-A0 are connected to the pins A11-A0 of the memory chip.
- Decoder decode A15-A12 and output O_0 is connected to CE' which is asserted only when A15-A12 is 0000
- One control signal MEMR' is connected to OE' to enable output buffer.

An address bus: This determines the location in memory that the processor will read data from or write data to.

A data bus: This contains the contents that have been read from the memory location or are to be written into the memory location.

A control bus: This manages the information flow between components indicating whether the operation is a read or a write and ensuring that the operation happens at the right time.

Example: Interfacing 64Kb EPROM with 8085:

Consider a system in which the full memory space 64kb is utilized for EPROM memory. Interface the EPROM with 8085 processor.

- The memory capacity is 64 Kbytes. i.e. 2^n = 64 × 1000 bytes where n = address lines.
- So, n = 16.
- In this system the entire 16 address lines of the processor are connected to address inptu pins of memory IC in order to address the internal locations of memory.
- The chip select (CS) pin of EPROM is permanently tied to logic low (i.e., tied to ground).
- Since the processor is connected to EPROM, the active low RD pin is connected toa ctive low output enable pin of EPROM.

- The range of address for EPROM is 0000H to FFFFH.

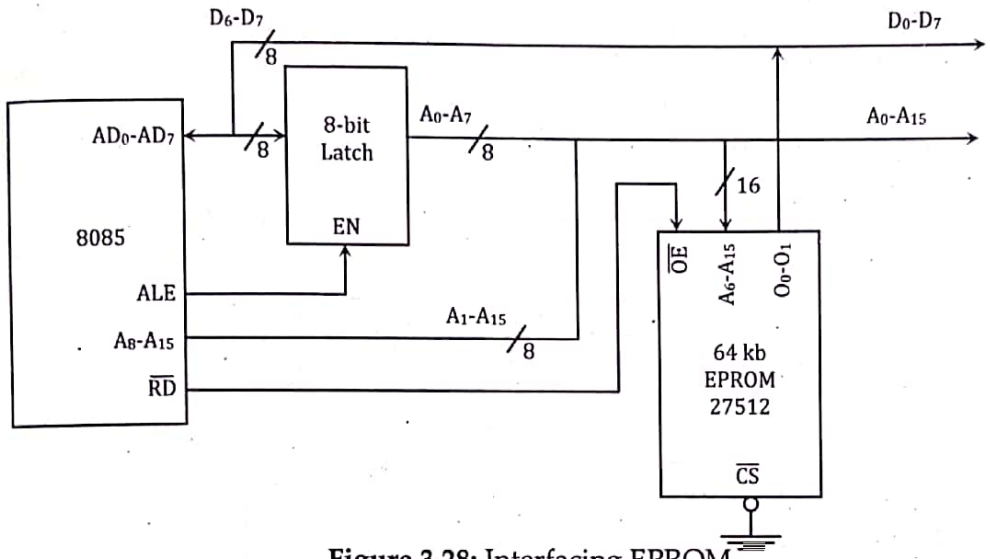


Figure 3.28: Interfacing EPROM

8085 Interfacing RAM Memory Chip

- 11 Address lines A10- A0 to decode 2048K registers.
- Address lines A15-A11 are connected to decoder (which is enabled by IO/M' signal in addition to the address lines A15 and A14).
- RD' and WR' signals are directly connected to memory chip.
- MEMR's and MEMW' need not to be generated separately (this technique save two gates).
- Memory Address Ranges from 8800H to 8FFFH.
- A 13-A11 (001) activate output O1 of decoder which is connected to CE' of memory chip and it is asserted only when IO/M' is low.

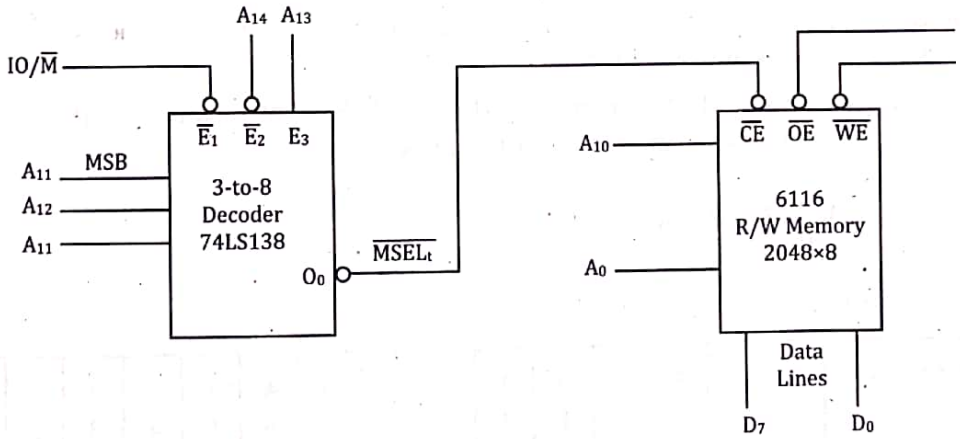


Figure 3.29: Interfacing RAM Memory

Example: Interfacing 32Kb EPROM and 32Kb RAM with 8085

- Consider a system in which the available 64kb memory space is equally divided between EPROM and RAM. Interface the EPROM and RAM with 8085 processor.
- Implement 32kb memory capacity of EPROM using single IC 27256.
- 32kb RAM capacity is implemented using singel IC 62256.
- The 32kb memory requires 15 address lines and so the address lines A0-A14 of the processor are connected to 15 address pins of both EPROM and RAM.

- The unused address line A15 is used as to chip select. If A15 is 1, it select RAM and if A15 is 0, it select EPROM.
- Inverter is used for selecting the memory.
- The memory used is both RAM and EPROM, so the low RD and WR pins of processor are connected to low WE and OE pins of memory respectively.
- The address range of EPROM will be 0000H to 7FFFH and that of RAM will be 7FFFH to FFFFH.

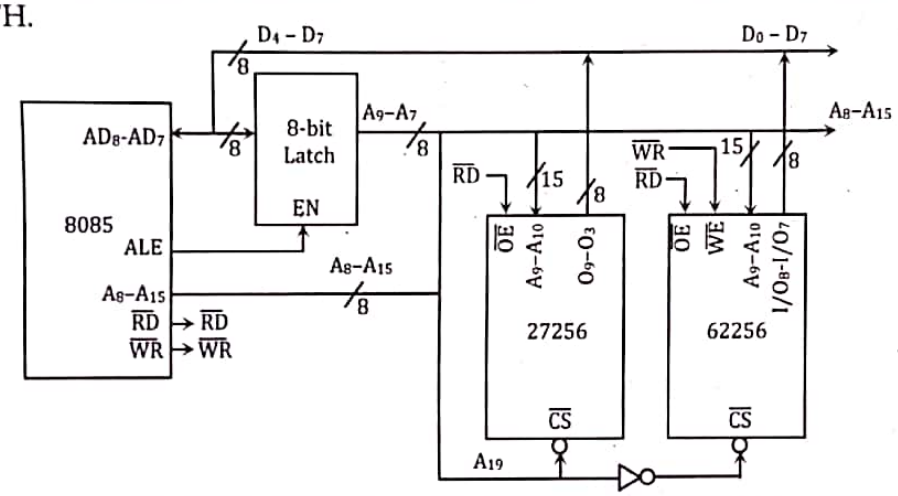


Figure 3.30: Interfacing 32Kb EPROM and 32Kb RAM with 8085

Example: Design an address decoding circuit for two RAM chips each of 4K X 8 at address2050H.

Memory Interfacing

- A memory chip requires address lines to identify a memory register. The number of address lines required is determined by the number of registers
- In a chip (2^n = number of registers where n is the number of address lines).
- A memory chip requires a chip select ($\overline{CS}$) signal to enable the chip. The remaining address lines (from above step) of the microprocessor can be connected to the CS signal through an interfacing logic.
- Thus, all address lines are responsible to select a specific register within a memory chip.

Step 1: Calculate the number of address pins

Here both memory devices are of 4K X 8 memory, which is 4KB. That means 2^n = 4KB = (4X1KB = $2^2X2^{10} = 2^{12}$). Therefore, 4KB memory requires 12 address lines.

Alternatively,

n = log (memory capacity in bytes) / log (2)

n = log (4X1024) / log (2) = 12

Step 2: Memory Mapping

Memory Block	Address	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RAM1	Start:2050H	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0
	End:304FH	0	0	1	1	0	0	0	0	0	1	0	0	1	1	1	1
RAM2	Start:3050H	0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0
	End:404FH	0	1	0	0	0	0	0	0	0	1	0	0	1	1	1	1

Here RAM1 requires 12 address lines that is 111111111111 (FFFH). The starting address of RAM1 is 2050H; we can calculate the end address of RAM1 by adding RAM1 addresses with its base address that is 2050H + FFFH = 304FH.

Similarly, RAM2 requires 12 address lines that is 111111111111 (FFFH). The next address of the RAM1's end address is the starting address of RAM2 that is 304FH + 01H = 3050H. Now we can calculate the end address of RAM2 by adding RAM2 addresses with its starting address that is 3050H + FFFH = 404FH.

Step 3: Decide decoder pins

Here, bit A12 in address lines for RAM1 and RAM2 referring to start address are different, so we require a 1X2 decoder. If we refer the end address, bits A12, A13 and A14 are different; in this case, we should use 3X8 decoder. Address lines A0 through A11 are used by RAM1 and RAM2 as both having 12 address pins. Rest of the address lines (A15 if 3X8 decoder and A13, A14 and A15 if 1X2 decoder) will be decoded to generate chip enable signals for decoder.

Step 4: Draw a decoding circuit

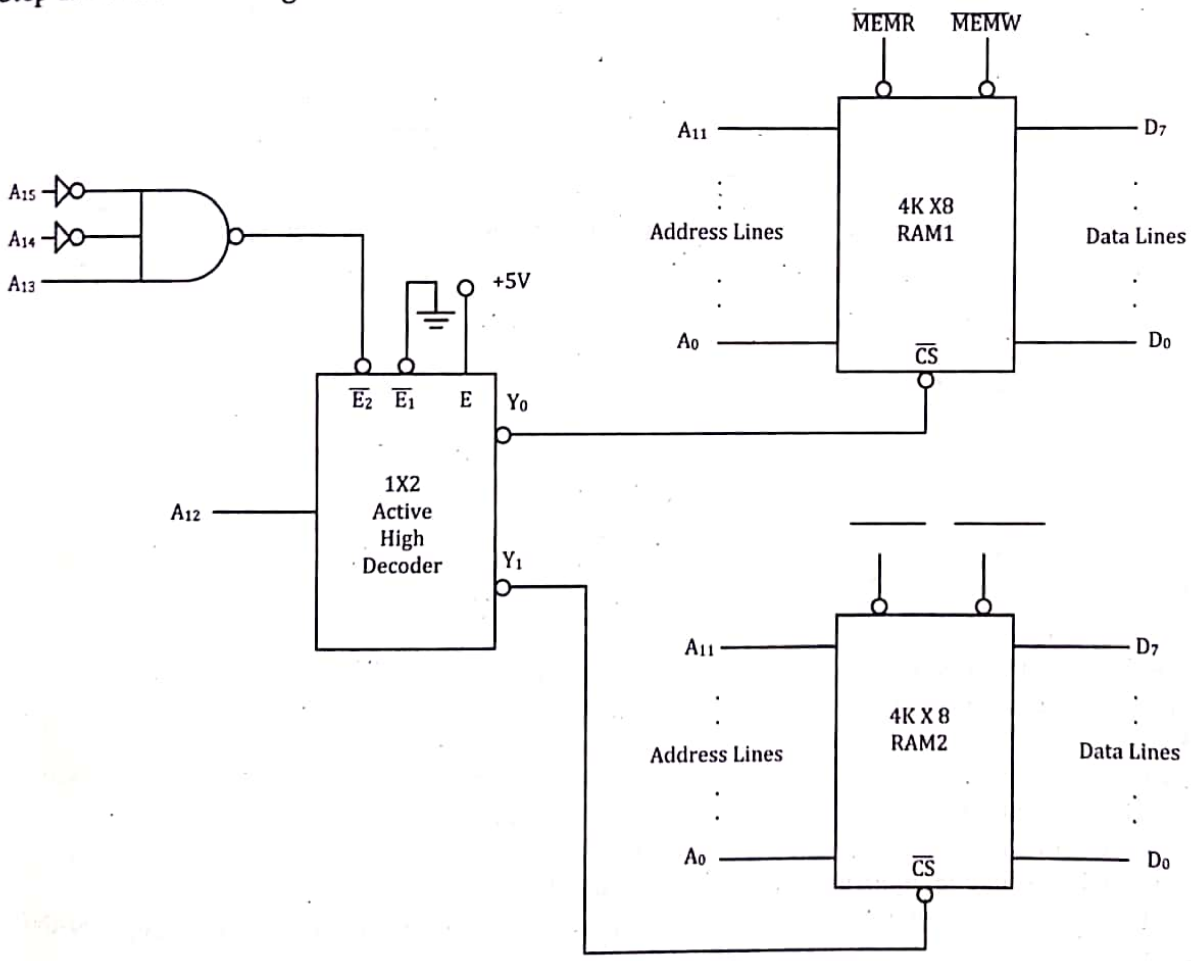


Figure 3.31: Address Decoding circuit with respect to start addresses.

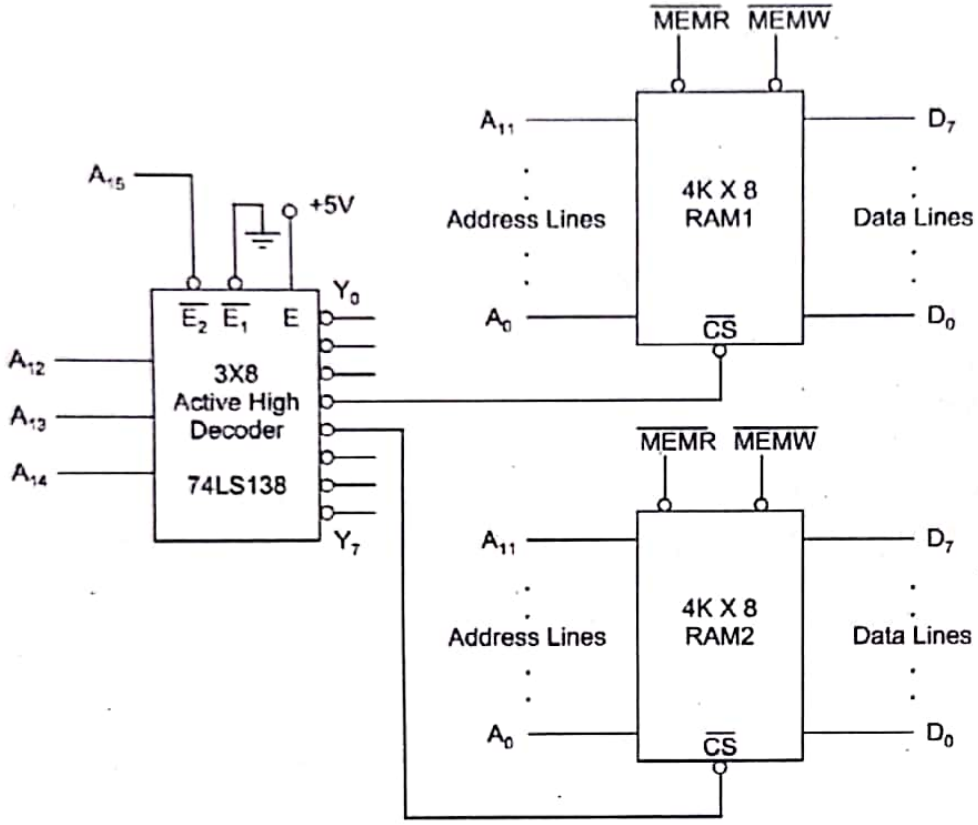


Figure 3.32: Address Decoding circuit with respect to end addresses

QUESTIONS

1. Define Instruction cycle, machine cycle and T-states.
2. Explain fetch operation with the help of its timing diagram.
3. Draw a timing diagram of instruction MVI A,32H and explain it.
4. Draw a timing diagram of instruction MOV A,32H and explain it.
5. Draw a timing diagram of instruction IN 01H and explain it.
6. Draw a timing diagram of instruction OUT 02H and explain it.
7. Draw a timing diagram of instruction STA 2000H and explain it.
8. Draw a timing diagram of instruction LDA 3000H and explain it.
9. What is address decoding in memory interfacing? Explain address decoding using simple NAND gates Decoder.
10. A 64 KB RAM is made up of four 16 KB RAM's. Show it's memory interfacing in 8085 microprocessor.
11. A 1 MB RAM is made up of four 256 KB RAM's. Show it's memory interfacing in 8086 Microprocessor.

□□□

4

ASSEMBLY LANGUAGE PROGRAMMING

CHAPTER OBJECTIVE

The core objective of this chapter is to teach assembly language programming based on 8085 and 8086 Architectures. Here, the instruction sets, instruction formats, mnemonics used in two different architectures is explained. This chapter explains the plenty of programs related to data transfer, arithmetic and logic operations, branching and looping , array processing, procedure call, conversion and table processing.

CHAPTER OUTLINE

- Programming with Intel 8085 Microprocessor
  - Instruction and Data Format
  - Instruction Sets
  - Simple Sequence Programs, Branching, Looping
  - Array (Sorting) and Table Processing
  - Decimal to BCD Conversion
  - Mnemonics and Operands
  - Multiplication and Division
- Programming with Intel 8086 microprocessor
  - Macro Assembler
  - Assembler Directives, Comments
  - INT 21H Functions
  - Simple String and Character Manipulation Programs
  - Debugging
  - Assembling and Linking
  - Instructions: LEA, MUL, DIV, LOOP, AAA, DAA
  - INT 10H Functions (Introduction Only)

4.1 Assembly Language Programming

Introduction

- Assembly Language uses two, three or 4 letter mnemonic store present each instruction type.
- Low level Assembly Language is designed for a specific family of Processors: the symbolic instruction directly relate to Machine Language instructions one for one and are assembled into machine language
- To make programming easier, many programmers write programs in assembly language
- They then translate Assembly Language program to machine language so that it can be loaded into memory and run.

Advantages of Assembly Language

- A Program written in Assembly Language requires considerably less Memory and execution time than that of High Level Language.
- Assembly Language gives a programmer the ability to perform highly technical tasks.
- Resident Programs (that resides in memory while other programs execute) and Interrupt Service Routine (that handles I/P and O/P) are almost always developed in Assembly Language.
- Provides more control over handling particular H/W requirements.
- Generates smaller and compact executable modules.
- Results in faster execution.

Typical Format of an Assembly Language Instruction

LABEL	OPCODEFIELD	OPERANDFIELD	COMMENTS
NEXT:	ADD	AL,07H	;Add correction factor

- Assembly language statements are usually written in a standard form that has 4 fields.
- A label is a symbol used to represent an address. They are followed by colon
- Labels are only inserted when they are needed so it is an optional field.
- The opcode field of the instruction contains the mnemonics for the instruction to be performed
- The instruction mnemonics are sometimes called as operation codes.
- The operand field of the statement contains the data, the memory address, the port address or the name of the register on which the instruction is to be performed.
- The final field in an assembly language statement is the comment field which starts with semi colon. It forms a well documented program.

Assembly Language Program Development Tools

1. Editor

- An Editor is a Program which allows you to create a file containing the Assembly Language statements for your Program.

2. Assembler

- An Assembler Program is used to translate the assembly language mnemonics for instruction to the corresponding binary codes.

3. Linker

- A Linker is a Program used to join several files into one large .obj file. It produces .exe file so that the program becomes executable.

4. Locator

- A Locator is a program used to assign the specific address of where the segment of object code are to be loaded into memory.
- It usually converts .exe file to .bin file.
- A Locator program EXE2BIN converts .exe file to .bin file.

5. Debugger

- A Debugger is a program which allows you to load your .obj code program into system memory, execute program and troubleshoot.
- It allows you to look at the content of registers and memory locations after your program runs.
- It allows to set the break point.

6. Emulator

- An Emulator is a mixture of hardware and software.
- It is used to test and debug the hardware and software of an external system such as the prototype of a Microprocess or based system.

Assembly Language Program Features

Program Comments

- The use of Comments throughout a program can improve its clarity, especially in Assembly Language.
- A Comment begins with Semicolon. Example: MOV AX, BX; Adds the Content of BX with AX

Reserved Words

- Instructions : MOV,ADD
- Directives : END,SEGMENT
- Operators : FAR,OFFSET
- Predefined Symbols : @DATA

Identifiers

- An Identifier (or symbol) is a name that you apply to an item in your program that you expect to reference. The two types of identifiers are NAME and LABEL.
- Name: Refers to the Address of a data item COUNTERDB0
- LABEL: Refer to the Address of an instruction, procedure, or segment.

  MAINPROCA20: MOVAL,BL

Statements

- An Assembly Program consists of a set of statements. The two types of statements are:

1. **Instruction**

- Instructions such as MOV & ADD which the Assembler translates to Object Code.

2. **Directives**

- Directives tell the Assembler to perform a specification, such as define a data item, etc.

4.2 Programming with Intel 8085 Microprocessor

Assembly Language Programming Introduction

- Assembly Language uses two, three or 4 letter mnemonics to represent each instruction type.
- Low level Assembly Language is designed for a specific family of Processors: the symbolic instruction directly relate to Machine Language instructions one for one and are assembled into machine language
- To make programming easier, many programmers write programs in assembly language
- They then translate Assembly Language program to machine language so that it can be loaded into memory and run.

Advantages of Assembly Language

- A Program written in Assembly Language requires considerably less Memory and execution time than that of High Level Language.
- Assembly Language gives a programmer the ability to perform highly technical tasks.
- Resident Programs (that resides in memory while other programs execute) and Interrupt Service Routine (that handles I/P and O/P) are almost always developed in Assembly Language.
- Provides more control over handling particular H/W requirements.
- Generates smaller and compact executable modules.
- Results in faster execution.

Typical Format of an Assembly Language Instruction

Label	Opcode field	Operand field	Comments
Next:	ADD	AL,07H	;Add correction factor

- Assembly language statements are usually written in a standard form that has 4 fields.
- A label is a symbol used to represent an address. They are followed by colon
- Labels are only inserted when they are needed so it is an optional field.
- The opcode field of the instruction contains the mnemonics for the instruction to be performed
- The instruction mnemonics are sometimes called as operation codes.
- The operand field of the statement contains the data, the memory address, the port address or the name of the register on which the instruction is to be performed.
- The final field in an assembly language statement is the comment field which starts with semicolon. It forms a well documented program.

Instruction Description and Format

The computer can be used to perform a specific task, only by specifying the necessary steps to complete the task. The collection of such ordered steps forms a 'program' of a computer. These ordered steps are the instructions. Computer instructions are stored in central memory locations and are executed sequentially one at a time.

Each instruction of a program is pointed by the program counter (PC). It is first moved to the instruction register (IR), then decoded into binary format and stored as an instruction in the memory. The computer takes a certain period to complete this task i. e. , instruction fetching, decoding and executing on the basis of clock speed. Such a time period is called 'Instruction cycle' and consists two cycles namely fetch & decode and Execute cycle.

Generally each instruction has two parts: one is the task to be performed, called the operation code (Op-Code) field, and the second is the data to be operated on, called the operand or address field. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or an 8-bit (or 16-bit) address. The Op-Code field specifies how data is to be manipulated and address field indicates the address of a data item. Though computer only understands binary language, it is complex and tedious for us to enter all the instructions in binary format. Besides, identification and correction of errors in the program is even harder. So, the instructions are written in meaningful words (relating to task they perform) as mnemonics so that it is easier for programmers to learn, understand and use easily. For example:

ADD R1, R0

Op-code address: Here R0 is the source register and R1 is the destination register.

The instruction adds the contents of R0 with the content of R1 and stores result in R1.

8085 A can handle at the maximum of 256 instructions (28) (246 instructions are used). Depending on the number of address specified in instruction sheet, the instruction format can be classified into three categories:

- **One Byte Instruction:** Here 1 byte will be Op-code and operand will be default. E. g. ADD B, MOV A,B
- **Two-Byte Instruction:** Here first byte will be Op-code and second byte will be the operand/data. E. g. IN 40H, MVI A 8-bit Data
- **Three-Byte Instruction:** Here first byte will be Op-code, second and third byte will be operands/data. That is
  - 2nd byte- lower order data.
  - 3rd byte – higher order data.
  - E. g. LXI B, 4050 H

The 8085 Instruction Sets

Instruction Details

Data Transfer Instructions

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. **Example:** MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVIB, 57H or MVIM, 57H
Load Accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H

Load Accumulator Indirect

LDAX B/D Reg. pair — The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B

Load register pair immediate

LXI Reg. pair, 16-bit data — The instruction loads 16-bit data in the register pair designated in the operand. **Example:** LXI H, 2034H or LXI H, XYZ

Load H and L registers direct

LHLD 16-bit address — The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H

Store accumulator direct

STA 16-bitaddress — The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350H

Store accumulator indirect

STAX Reg. pair — The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. Example: STAX B

Store H and L registers direct

SHLD 16-bitaddress — The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, these cond byte specifies the low-order address and the third byte specifies the high-order address. Example: SHLD2470H

Exchange H and L with D and E

XCHG None — The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example: XCHG

Copy H and L registers to the stack pointer

SPHL none — The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. Example: SPHL

Exchange H and L with top of stack

XTHL	none	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location(SP+1); however, the contents of the stack pointer register are not altered. Example: XTHL

Push register pair onto stack

PUSH	Reg. pair	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. Example: PUSHB or PUSH A.

Pop off stack to register pair

POP	Reg. pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register(C,E,L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example: POPH or POP A.

Output data from accumulator to a port with 8-bit address

OUT	8-bit port address	The contents of the accumulator are copied into the I/O port specified by the operand. Example: OUT F8H

Input data to accumulator from a port with 8-bit address

IN	8-bit port address	The contents of the input port designated in the operand are read and loaded into the accumulator. Example: IN 8CH

Arithmetic Instructions

Opcode	Operand	Description

Add register or memory to accumulator

ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. **Example:** ADD B or ADD M

Add register to accumulator with carry

ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M

Add immediate to accumulator

ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45H

Add immediate to accumulator with carry

ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H

Add register pair to H and L registers

DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. **Example:** DAD H

Subtract register or memory from accumulator

SUB	R M	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. **Example:** SUB B or SUB M

Subtract source and borrow from accumulator

SBB	R M	The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. **Example:** SBB B or SBB M

Subtract immediate from accumulator

SUI	8-bit data	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. **Example:** SUI 45H

Subtract immediate from accumulator with borrow

SBI	8-bit data	The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. **Example:** SBI 45H

Increment register or memory by 1

INR	R M	The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. **Example:** INR B or INR M

Increment register pair by 1

INX	R	The contents of the designated register pair are incremented by 1 and the result is stored in the same place. Example: INX H

Decrement register or memory by 1

DCR	R M	The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. **Example:** DCR B or DCR M

Decrement register pair by 1

DCX	R	The contents of the designated register pair are decremented by 1 and the result is stored in the same place. Example: DCX H

Decimal adjust accumulator

DAA	none	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. **Example:** DAA

Branching Instructions

Opcode	Operand	Description
Jump unconditionally		
JMP	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Example: JMP2034H or JMP XYZ
Jump conditionally		
Operand	16-bit address	The program sequence is transferred to the memory location specified by the16-bit address given in the operand based on the specified flag of the PSW as described below. Example: JZ 2034H or JZ XYZ

Opcode	Description	Flag Status
JNC	Jump on no Carry	CY= 0
JP	Jump on positive	S= 0
JM	Jump on minus	S= 1
JZ	Jump on zero	Z= 1
JNZ	Jump on no zero	Z= 0
JPE	Jump on parity even	P= 1
JPO	Jump on parity odd	P= 0

Unconditional subroutine call

CALL	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. Example: CALL 2034H or CALL XYZ

Call conditionally

Operand 16-bit address — The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack. Example: CZ 2034H or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY= 1
CNC	Call on no Carry	CY= 0
CP	Call on positive	S= 0
CM	Call on minus	S= 1
CZ	Call on zero	Z= 1
CNZ	Call on no zero	Z= 0
CPE	Call on parity even	P= 1
CPO	Call on parity odd	P= 0

Return from subroutine unconditionally

RET	none	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RET

Return from subroutine conditionally

Operand none — The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RZ

Opcode	Description	Flag Status
RC	Return on Carry	CY= 1
RNC	Return on no Carry	CY= 0
RP	Return on positive	S= 0
RM	Return on minus	S= 1
RZ	Return on zero	Z= 1
RNZ	Return on no zero	Z= 0
RPE	Return on parity even	P= 1
RPO	Return on parity odd	P= 0

Load program counter with HL contents

PCHL none — The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte. Example: PCHL

Restart

RST 0-7 — The RST instruction is equivalent to a1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However, these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Instruction	Restart Address
RST0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

Interrupt	Restart Address
TRAP	0024H
RST 5. 5	002CH
RST 6. 5	0034H
RST 7. 5	003CH

Logical Instructions

Opcode Operand — Description

Compare register or memory with accumulator

CMP R
M — The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:
if(A)< (reg/mem): carry flag is set if (A)= (reg/mem): zero flag is set
if(A)> (reg/mem): carry and zero flags are reset
Example: CMP B or CMP M

Compare immediate with accumulator

CPI 8-bit data — These cond byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
if(A)< data: carry flag is set if (A)= data: zero flag is set
if(A)> data: carry and zero flags are reset
Example: CPI 89H

Logical AND register or memory with accumulator

ANA R
M

The contents of the accumulator are logically ANDed with the contents of the operand (register or memory),and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANA B or ANA M

Logical AND immediate with accumulator

ANI 8-bitdata

The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANI 86H

Exclusive OR register or memory with accumulator

XRA R
M

The contents of the accumulator are Exclusive O Red with the contents of the operand (register or memory),and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRA B or XRA M

Exclusive OR immediate with accumulator

XRI 8-bitdata

The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRI 86H

Logical OR register or memory with accumulator

ORA R
M

The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORA B or ORA M

Logical OR immediate with accumulator

ORI 8-bitdata

The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORI 86H

Rotate accumulator left

RLC none

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D_7. S, Z, P, AC are not affected. Example: RLC

Rotate accumulator right

RRC	none	Each binary bit of the accumulator is rotated right by one position. Bit D_0 is placed in the position of D_7 as well as in the Carry flag. CY is modified according to bit D_0. S, Z, P, AC are not affected. Example: RRC

Rotate accumulator left through carry

RAL	none	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D_7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D_0. CY is modified according to bit D_7. S, Z, P, AC are not affected. Example: RAL

Rotate accumulator right through carry

RAR	none	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D_0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D_7. CY is modified according to bit D_0. S, Z, P, AC are not affected. **Example:** RAR

Complement accumulator

CMA	none	The contents of the accumulator are complemented. No flags are affected. Example: CMA

Complement carry

CMC	none	The Carry flag is complemented. No other flags are affected. Example: CMC

Set Carry

STC	none	The Carry flag is set to 1. No other flags are affected. Example: STC

Control Instructions

Opcode	Operand	Description

No operation

NOP	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP

Halt and enter wait state

HLT	none	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT

Disable interrupts

DI	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI

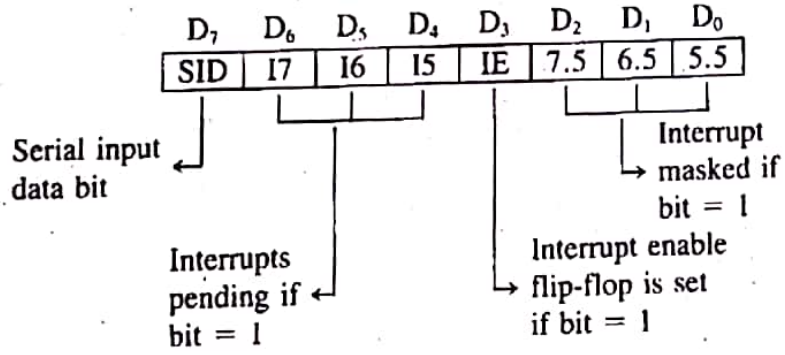
Enable interrupts

EI	none	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to re-enable the interrupts (except TRAP). Example: EI

Read interrupt mask

RIM none

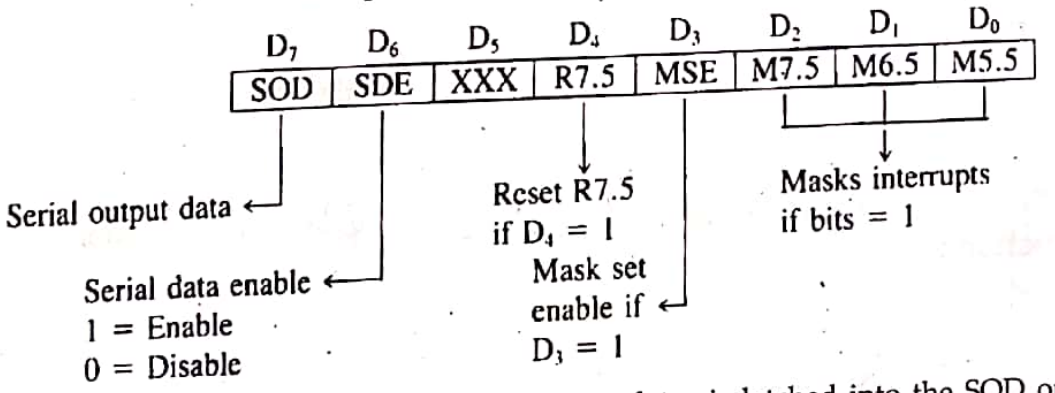
This is a multi purpose instruction used to read the status of interrupts 7. 5, 6. 5,5. 5 and read serial data in put bit. The instruction loads eight bits in the accumulator with the following interpretations. Example: RIM



Set interrupt mask

SIM none

This is a multi purpose instruction and used to implement the 8085 interrupts 7. 5, 6. 5, 5. 5, and serial data output. The instruction interprets the accumulator contents as follows. Example: SIM



- **SOD–Serial Output Data:** Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit $D_6 = 1$.
- **SDE–Serial Data Enable:** If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- **XXX–Dont's Care**
- **D7.5–Reset RST 7.5:** If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- **MSE–Mask Set Enable:** If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.
- M7.5–D_2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- M6.5–D_1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- M5.5–D_0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

Procedure for Microprocessor Programming by using ESA 8085 Kit

Intel 8085 microprocessor is used in the µp kit. The input device is the hexadecimal keyboard. The output device is six-seven segment LED display, of which the first four from left represent address field and the last two represents the data field. A +5 Volt power supply is connected to the µp kit.

Note: The steps given below are also similar for other model of 8085 microprocessor kit.

I. First write the assembly language program for the given problem using microprocessor instructions (i.e., Mnemonics form) and symbolic addresses (labels).

II. Then convert the assembly language program into Hexcode using opcode sheet (i.e. hand assemble) and assign the real addresses.

III. Procedure to enter the program or data in to the Memory
   a. Press "RESET" key; monitor program message will appear in the address field and in the data field of LED display.
   b. Press: EXAM-MEM" key a dot will appear in the led display at the right corner of the address field.
   c. Enter the starting address of the memory location i.e. the address of the memory location where we want to store the first opcode of the program or the data.
   d. Press: NEXT" key; the current data in the memory location will be displayed int ehd ata field.
   e. Enter the opcode/data in the field and press "NEXT" key. The address in the address field is automatically incremented by one, every time the :NEXT" key is pressed.
   f. Repeat the previous step until the complete program or all the data is entered. After every upcode/data entry press "NEXT" again.

IV. Procedure to execute the program and verify the result in a memory location or register.
   a. Press "RESET" key.
   b. Press "GO" key.
   c. Enter the starting address of the program where the first opcode of the program to be executed is stored.
   d. Press "EXEC" key. Either "E" or monitor message will appear in the display.
   e. If the monitor program ends with a "HLT" instruction press "RESET" key; press "EXAM-MEM" key and then enter the address of the memory location where you want to verify the result and finally press "NEXT" key to see the memory content in the data field.
   f. Otherwise press "EXAM-REG" key and then press the key for the particular register whose content is to be verified. The register name will be displayed in the address field and its content will be displayed in the data field.

8085 Microprocessor Programming-I

Aim: To write assembly language programs for

- Data transfer operations
- Block data transfer operations
- Rotate operations, hand assemble enter execute them in 8085 microprocessor kit.

Label	Instruction	Memory Address	Memory content		
			Op-code	Low A/D	High A/D
I. To transfer data/address to register(s)					
(a) To load a user given data to a register:					
	MVI C 93H	8C00	OE	93	
(b) To load a user given data/address to a register pair:					
	LXI D, 1977H	8C02	11	77	19
(c) To transfer data from one register to another:					
	MVI B, 15H	8C05	06	15	
	MOV A, B	8C07	78		
	RST 5	8C08	EF		
II. To transfer data/address to memory					
a) To load user given data to a memory location:					
	LXIH, 8D05 H	8C09	21	05	8D
	MVI M; 95H	8C0C	36	95	
(b) To move data from Accumulator to memory (direct):					
	MVI A, 65H	8C0E	3E	65	
	STA 8Do6H	8C10	32	06	8D
(c) To move data from Accumulator to memory (indirect):					
	LXIB, 8D07H	8C13	01	07	8D
	MVIA, 50H	8C16	3E	50	
	STAX B	8C18	02		
(d) To move data from a register to memory:					
	LXIH, 8D08 H	9C19	21	08	8D
	MVI D, 99H	8C1C	16	99	
	MOV M, D	8C1E	72		
(e) To move data from the register pair HL to memory:					
	LXI H, ABCE H	8C1F	21	CE	AB
	SHLD 8D09 H	8C22	22	09	8D
	HLT	8C25	76		
III. To transfer data from memory to register:					
(a) To transfer data from memory to accumulator (direct):					
	LDA 8D0B H	8C26	3A	OB	8D
	RST 5	8C29	EF		
(b) To transfer data from memory to accumulator (indirect):					
	LXI D, 8D0C H	8C2A	11	OC	8D
	LDAX D	8C2D	1A		
	RST 5	8C2E	EF		
(c) To transfer data from memory to register:					
	LXI H, 8DoDH	8C2F	21	OD	8D
	MOV C, M	8C32	4E		
	RST 5	8C33	EF		
(d) To transfer data from memory to register pair HL:					
	LHLD 8DOE H	8C34	2A	OE	8D
	RST 5	8C37	EF		

IV. Data exchange between register pairs

(a) To exchange data between HL and DE register pairs:					
	LXI H, 22EE H	8C38	21	33	22
	LXI D, 4455 H	8C3B	11	55	44
	XCHG	8C3E	EB		
	RST	8C3F	EF		
(b) To exchange data between BC and DE register pairs:					
	LXI B, 1234 H	8C40	01	34	12
	LXI D, 5678 H	8C43	11	78	56
	MOV H, B	8C46	60		
	MOV L, C	8C47	69		
	XCHG	8C48	EB		
	MOV B, H	8C49	44		
	MOV C, L	8C4A	4D		
	RST 5	8C4B	EF		

V. Block data transfer

Transfer a block of data stored in memory locations in one area starting from 8D51H onwards to another area starting from memory locations 8D61H onwards. The number of data (bytes) to be transferred is stored in memory location 8D50H.

	LXI H, 8D50	H	8C50	21	50	8D
	MOV B, M		8C53	46		
	LXI D, 8D60	H	8C54	11	60	8D
LI:	INX H		8C57	23		
	INX D		8C58	13		
	MOV A, M		8C59	7E		
	STAX D		8C5A	12		
	DCR B		8C5B	05		
	JNZ LI		8C5C	C2	57	8C
	HLT		8C5F	76		

VI. Rotate operations

To shift data in the accumulator to the left/right, without/with carry:

STC	8C60	37	
MVI A, A6	8C61	3E	A6
RLC/RRC/ RAL/RAR	8C63	07/OF/17/IF	

8085 Microprocessor Programming-II

Aim: To write assembly language programs for

- 8 bit additional (binary/decimal)
- Sum of a series of 8 bit numbers (binary/decimal)
- 8 bit subtraction (binary)
- 8 bit subtraction (decimal)
- 8 bit multiplication (binary)
- 8 bit division (binary) and
- Multi byte addition (binary/decimal) hand assembly enter and execute them in the 8085 μpkit.

(i) 8 bit addition (binary/decimal)

Add the two numbers in binary/decimal in two memory locations 8D71H and 8D72H and store the sum in the memory location 8D73H.

	LXIH, 8D71H	8C70	21	71	8D	Get address of 1st number in H-L pair.
	MOV A, M	8C73	7E			1st number in accumulator.
	INXH	8C74	23			Increment content of H-L pair.
	ADD M	8C75	86			Add 1st and 2nd number.
	NOP/DAA	8C76	00/27			No operation/decimal adjust
	STA8D73H	8C77	32	73	8D	Store sum in 8D73H.
	HLT	8C7A	76			

Note: Use NOP instruction for binary addition and DAA for decimal addition.

(ii) Sum of a series of (nine) 8 bit numbers (binary/decimal): Sum 16 bit/3 digit decimal maximum.

Add the series of nine binary/decimal numbers stored in nine memory locations starting from 8D89H. The number count is stored in the memory location 8D80H. The sum is to be stored in two memory locations 8D8AH (LAB) and 8D8BH (MAB).

	LXI H, 8D80H	8C80	21	80	8D	Address of count in H-L pair.
	MOV C, M	8C83	4E			Count in register C
	MVI A, 00	8C84	3E	00		LSBs of sum = 00 (initial value)
	MOV B, A	8C86	47			MSBs of sum = 00 (initial value)
LOOP:	INX H	8C87	23			Address of next data in H-L pair
	ADD M	8C88	86			Previous sum + next number
	NOP/DAA	8C89	00/27			No operation/decimal adjust
	JNC AHEAD	8C8A	D2	8E	8C	Is carry? No, go to AHEAD.
	INR B	8C8D	04			Yes, add carry to MSB of sum.
AHEAD:	DCR C	8C8E	0D			Decrement count.
	JNZ LOOP	8C8F	C2	87	8C	Is count = 0?
						No, jump to LOOP.
	STA 8D8AH	8C92	32	8A	8D	Store LSB of the sum in 8D8AH.
	MOV A, B	8C95	78			Get MSB of sum in accumulator.
	STA 8D8BH	8C94	32	8B	8D	Store MSB of sum in 8D8B H.
	HLT	8C99	76			Stop.

Note: For binary sum use NOP instruction and/for decimal sum use DAA instruction.

(iii) 8 bit subtraction (binary)

LXI H, 8D91H	8CA0	21	91	8D	Get address of 1st number in H-L pair.
MOV A, M	8CA3	7E			1st number in accumulator.
INX H	8CA4	23			Content of H-L pair increases from 8D91H to 8D92H.
SUB M	8CA5	96			1st number-2nd number
INX H	8CA6	23			Content of H-L pair becomes 8D93H
MOV M, A	8CA7	77			Stored result in 8D93H
MLT	8CA8	76			Halt.

(iv) 8 bit subtraction (decimal)

Subtract the 2 digit decimal number (2nd number) stored in memory location 8DA2H from the 2 digit decimal number (1st number) stored in memory location 8DA1H. The result is stored in memory location 8DA3H. If the result is negative its ten's complement will be stored.

LXI H, 8DA2H	8CB0	21	A2	8D	Get address of 2nd number in H-L pair.
MVI A, 99	8CB3	3E	99		Place 99 in accumulator.
SUB M	8CB5	96			9's complement of 2nd number.
INR A	8CB6	3C			10's complement of 2nd number.
DCX H	8CB7	2B			Get address of 1st number.
ADD M	8CB8	86			Add 1st number and 10's complement of 2nd number.
DAA	8CB9	27			Decimal adjust.
STA 8DA3H	8CBA	32	A3	8D	Store result in 8DA3H.
HLT	8CBD	76			Halt.

(v) 8 bit multiplication (binary); product 16 bit

Multiplication is stored in memory location 8DB1H. Multiplier is stored in memory location 8DB3H. In memory location 8DB2H the number 00H is stored. After the multiplication LSB of product is stored in memory location 8DB4H and MSB of product is stored in memory location 8DB5H.

	LHLD 8DB1H	8CC0	2A	B1	8D	Get multiplied in H-L pair.
	XCHG	8CC3	EB			Multiplicand in D-E pair.
	LDA 8DB3H	8CC4	3A	B3	8D	Multiplier in accumulator.
	LXI H, 0000	8CC7	21	00	00	Initial value of product = 0000 in H-L pair.
	MVI C, 08	8CCA	0E	08		Count = 08 in register C.
LOOP:	DAD H	8CCC	29			Shift partial product left by 1 bit.
	RAL	8CCD	17			Rotate multiplier left one bit. Is multiplier's bit = 1?
	JNC AHEAD	8CCE	D2	D2	8C	No, go to AHEAD.
	DAD D	8CD1	19			Product = product + Multiplicand
AHEAD:	DCR C	8CD2	0D			Decrement count
	JNZ LOOP	8CD3	C2	CC	8C	
	SHLD 8DB4H	8CD6	22	B4	8D	Store result in 8DB4H & 8DB5H.
	HLT	8CD9	76			

(vi) 8 bit division (binary)

Here dividend is 16 bit and divisor is 8 bit. LSB of dividend is stored in memory location 8DC1H and MSB of dividend is stored in memory location 8DC2H. 8 bit divisor is stored in memory location 8DC3H. After division the quotient is stored in memory location 8DC4H and the remainder is stored in memory location 8DC5H.

Label	Instruction	Memory Address	Memory content			Comments
			Op-code	Low A/D	High A/D	
	LHLD 8DC1H	8CE0	2A	C1	8D	Get dividend in H-L pair.
	LDA 8DC3H	8CE3	3A	C3	8D	Get divisor form 8DC3H.
	MOV B, A	8CE6	47			Divisor in register B.
	MVI C, 08	8CE7	0E	08		Count = 08 in register C.
LOOP:	DAD H	8CE9	29			Shift dividend and quotient left by one bit.
	MOV A, H	8CEA	7C			Most significant bits of dividend in accumulator.
	SUB B	8CEB	90			Subtract divisor from most significant bits if dividend.
	JC AHEAD	8CEC	DA	F1	8C	Is most significant part of dividend<divisor? No, go to AHEAD.
	MOV H, A	8CEF	67			Most significant bits of dividend in register H.
	INRL	8CF0	2C			Yes, add 1 to quotient.
AHEAD:	DCRC	8CF1	0D			Decrement count.
	JNZ LOOP	8CF2	C2	E9	8C	Is count = 00? No, jump to LOOP.
	SHLD 8DC4H	8CF5	22	C4	8D	Store quotient in 8DCGH and remainder in 8DC5H.
	HLT	8CF8	76			Stop.

(vii) Multibyte addition (binary/decimal)

Here there are two n byte decimal numbers. Each number has been extended to (n+1)th byte where 00H will be stored. The count is stored in memory location 8E00H. The bytes of first number are placed in memory location 8E01H onwards. The bytes of second number are placed in memory location 8F01H onwards. The sum is stored in memory locations occupied by the first number i.e., 8E01H onwards.

	LXI H, 8EOOH	8D00	21	00	8E	Address of count in H-L pair.
	MOV C, M	8D03	4E			Count in register C.
	INX H	8D04	23			Address of 1st byte of 1st number.
	LXI D, 8F01H	8D05	11	01	8F	Address of 1st byte of 2nd number.
	ORA A	8D08	B7			Clear carry.
LOOP:	LDAX D	8D09	1A			Get byte of 2nd number in accumulator.
	ADC M	8D0A	8E			Byte of 2nd number + byte of 1st number + carry.
	NOP/DAA	8D0B	00/27			No operation/decimal adjust.
	MOV M, A	8D0C	77			Store sum in memory addressed by H-L pair.
	INX D	8D0D	13			Increment D-E pair.
	INX H	8D0E	23			Increment H-L pair.
	DCR C	8D0F	OD			Decrement count.
	JNZ LOOP	8D10	C2	09	8D	Is count 00? No, got to LOOP.
	HLT	8D13	76			Stop.

Note: Use NOP instruction for binary and DAA for decimal addition.

8085 Microprocessor Programming-III

Aim: To write assembly language programs for

- finding the largest/smallest number in an array of data,
- sorting an array of data in ascending/descending order,
- converting binary to BCD,
- converting BDC to binary and
- finding square using took up table. Hand assemble enter and execute them in 8085 microprocessor kit.

(i) To find the largest/smallest number in an array of data:

The count is placed in the memory location 8D01H. The numbers in the array of data are placed in memory location 8D02H onwards. The result is to be stored in memory locations 8DOOH.

The 1st number of the series is placed in the ACC and it is compared with the 2nd number residing in memory. The larger/smaller of the two numbers is placed in the ACC. Again this number which is in the ACC is compared with the 3rd number of series and larger/smaller number is placed in the ACC. This process of comparison is repeated till all the numbers of the series are compared and the largest/smallest number is stored in the desired memory location.

	Instruction	Memory Address	Memory content			Comments
			Op-code	Low A/D	High A/D	
(i) To find the largest/smallest number in an array of data.						
	LXI H, 8DO1H	8C00	21	01	8D	Get address for count in H-L pair.
	MOV C, M	8C03	4E			Count in register C.
	INX H	8C04	23			Get address of 1st number in H-L pair
	MOV A, M	8C05	7E			1st number in ACC.
	DCR C	8C06	OD			Decrement count.
LOOP:	INX H	8C07	23			Address of next number in H-L pair.
	CMP M	8C08	BE			Compare next number with previous largest/smallest. Is next number>previous largest? Is next number< previous smallest?
						No (CY = 0), largest/Yes (CY=1) smallest number is in accumulator.
	JNC/JC AHEAD	8C09	D2/DA	OD	8C	Go to the label AHEAD
	MOV A, M		8C0C	7E		Yes, get larger number/smaller number in ACC.
AHEAD	DCR C	8C0D	OD			Decrement count.
	JNZ LOOP	8C0E	C2	07	8C	
	STA 8D00H	8C11	32	00	8D	Store result (largest/smallest) in memory location 8DOOH.
	HLT	8C14	76			Stop.

	Memory Address	Memory content	Memory content
Output	8D00	(Largest)	(Smallest)
Input	8D01	05 (length)	05 (length)
	8D02		
	8D03		
	8D04		
	8D05		
	8D06		

(ii) To sort an array of data in Ascending/Descending order

	LXIH, 8D10H	8C20	21	10	8D	Address for count
	MOV C, M	8C23	4E			Count in C register
	DCR C	8C24	OD			Count for number of passes in C register i.e. count-1
BACK	MOV D, C	8C25	51			Count for number of comparisons in D register
	LXIH, 8D11H	8C26	21	11	8D	Starting address of the array of data in H-L register pair
	MOV A, M	8C29	7E			1st number in accumulator
LOOP:	INX H	8C2A	23			Address of next number
	MOV B, M	8C2B	46			Next number in B register
	CMP B	8C2C	B8			Compare next number with the previous greatest number in ACC
	JNC AHEAD	8C2D	D2	36	8C	If previous greatest number> next number, go to AHEAD
	DCX H	8C30	2B			Address of previous memory location
	MOV M, A	8C31	77			Place smaller of the two compared numbers in memory
	MOV A, B	8C32	78			Place greater of the two numbers in accumulator
	JMP GO	8C33	C3	38	8C	Branch to step labeled GO.
AHEAD:	DCXH	8C36	2B			Address of previous memory location
	MOV M, B	8C37	70			Place smaller of the two number in memory
GO:	IN X H	8C38	23			Address of next memory location
	DCR D	8C39	15			Decrease the count for comparisons
	JNZ LOOP	8C3A	C2	2A	8C	Branch to stop labeled LOOP if the count is not equal to 0.
	MOV M, A	8C3D	77			Place the greatest number after a pass in the memory
	DCR C	8C3E	OD			Decrease the count for passes
	JNZ BACK	8C3F	C2	25	8C	If count for passes = 0, go the BACK
	HLT	8C42	76			Other wise, stop.

Note: When the instruction JNC at the memory location 8C2D is replaced by JC (code: DA), the above program becomes a program for arranging an array of data in descending order.

Memory Address	Memory content Ascending order		Memory content Descending order	
	Input	Output	Input	Output
8D10	05 (length)		05 (length)	
8D11				
8D12				
8D13				
8D14				
8D15				

In this program the microprocessor first compares the first two memory of the data array. It keeps the larger of the two numbers in the Accumulator and stores the smaller number in the memory. Then it proceeds further and takes up the third number and compares it with the number which is an accumulator. If the number in the accumulators is smaller than the 3rd number it stores the 3rd number in accumulator. If stores the smaller (which was in the accumulator) in the memory. In this way it processes all the numbers of the array. Finally, it stores the largest number in the last memory location. This is all in the 1st pass. Now the processor again goes in the loop for the 2nd pass again it selects the largest number from the remaining numbers and stores it in the last but one memory location. The processor goes through several passes and arranges numbers in ascending order. Both the number passes and the number of comparisons well be less than the number of data in t he array by one. When the processor arranges numbers in descending order it selects the smallest number from the data array and stores it in the last memory location. In this way it proceeds further.

Label	Instruction	Memory Address	Memory content Op-code	Low A/D	High A/D	Comments
(i) To convert and 8 bit binary number to 3 digit BCD						
	LXIH, 8D20H	8C50	21	20	8D	Get the address of the binary number in H-L pair.
	LXIB, 0000H	8C53	01	00	00	Initialize B (counter for 100) and C (counter for 10) registers to zero.
	MOV A, M	8C56	7E			Get the binary No in ACC.
	L1: SUI64H	8C57	D6	64		Subtract 100 (64H) from the given number
	JC L2	8C59	DA	60	8C	If the given number < 100 go to step L2
HUNDS:	INRB	8C5C	04			Otherwise increment the counter for 100
	JMP LI	8C5D	C3	57	8C	Go to step L1
I2:	ADI 64H	8C60	C6	64		Add 100 (64H) to get the original value
L3:	SUI OAH	8C62	D6	OA		Subtract 10 (OAH) from the remainder of the given number
	JC L4	8C64	DA	6B	8C	If it is <10 go to step L4
TENS:	INR C	8C67	OC			Otherwise increment the counter for 10
	JMP L3	8C68	C3	62	8C	Go to step L3
L4:	ADI OAH	8C6B	C6	OA		Add 10 (OAH) to get the original value
	INX H	8C6D	23			Store No. of hundreds in memory
STRH:	MOV M, B	8C6E	70			
	INX H	8C6F	23			Store No. of tens in memory
STRT:	MOV M, C	8C70	71			
	INX H	8C71	23			Store No. of units in memory
STRU:	MOV M, A	8C72	77			
	HLT	8C73	76			

	Memory Address		Memory content	
			Date 1	Date 2
Input	8D20	HEX BINARY		
Output	8D21	HUNDREDS		
	8D22	TENS		
	8D23	UNITS		

Label	Instruction	Memory Address	Memory content Op-code	Low A/D	High A/D	Comments
(iv) To convert a 2 digit BCD number to 8 bit binary.						
	LXIH, 8D30H	8C80	21	30	8D	Get the address of the memory where tens digit is stored into HL register.
	XRA A	8C83	AF			Clear the accumulator
	MOV B, M	8C84	46			Get the tens of BCD
	CMP B					No. into B register
		8C85	B8			Compare it with zero
	JZ L6	8C86	CA	8F	8C	If zero go to step labeled L6
L5:	ADI OAH	8C89	C6	OA		Add ten to acc. content
	DCR B	8C8B	05			Decrease the tens value by one
	JNZ L5	8C8C	C2	89	8C	If the content of B=0 go to step labelled L5
L6:	INXH	8C8F	23			Get the next memory location where units of
						BCD No is stored
	ADD M	8C90	86			Add it to the content of A register
	STA 8D32H	8C91	32	32	8D	Store the result (8 bit binary) in a memory location
	HLT	8C84	76			

	Memory Address		Memory content	
			Date 1	Date 2
Input (BCD)	8D30	TENS		
	8D31	UNITS		
Output	8D32	HEX BINARY		

(v) To find the square of the given one digit number using look up table

LDA 8D4OH	8CA0	3A	40	8D	Get data in accumulator
MOV L, A	8CA3	6F			Get data in register L
MOV H, 8E	8CA4	26	8E		Get 8e (the higher part of the address of the address of the look up table in register H)
MOV A, M	8CA6	7E			Square of data in ACC.
STA 8D41H	8CA7	32	41	8D	Store square in 8D41H
HLT	8CAA	76			Stop

The squares of data are stored in certain memory location in the tabular form. This is called look up table. The data form the index and it is transfers from memory to the accumulator and then to register L. It forms the LSBs of the memory location where square of the data is load. The MSBs of the address is moved to register. HN in the address of the desired memory location where the square of the data resides is in HL pair. The square of the data is now moved to the accumulator and then it is stored in memory location 8D41H. The number (data) for which the square is required is stored in memory location 8D40. The squares of numbers from 00 to 09 are stored in memory locations 8E00 to 8E09H. The values of squares are in decimal.

Look up Table

Memory Address (HEX)	Memory content (square in decimal)
8E00	00
8E01	01
8E02	04
8E03	09
8E04	16
8E05	25
8E06	36
8E07	49
8E08	64
8E09	81

	Memory address	Memory content
Input	8D40	
Output (square)	8D41	

OPCODES TABLE OF INTEL 8085

Opcodes of Intel 8085 in Alphabetical Order

Sr. No.	Mnemonics, Operand	Opcode	Bytes
1.	ACI Data	CE	2
2.	ADC A	8F	1
3.	ADC B	88	1
4.	ADC C	89	1
5.	ADC D	8A	1
6.	ADC E	8B	1
7.	ADC H	8C	1
8.	ADC L	8D	1
9.	ADC M	8E	1
10.	ADD A	87	1
11.	ADD B	80	1
12.	ADD C	81	1
13.	ADD D	82	1
14.	ADD E	83	1
15.	ADD H	84	1
16.	ADD L	85	1
17.	ADD M	86	1
18.	ADI Data	C6	2
19.	ANA A	A7	1

Sr. No.	Mnemonics, Operand	Opcode	Bytes
20.	ANA B	A0	1
21.	ANA C	A1	1
22.	ANA D	A2	1
23.	ANA E	A3	1
24.	ANA H	A4	1
25.	ANA L	A5	1
26.	ANA M	A6	1
27.	ANI Data	E6	2
28.	CALL Label	CD	3
29.	CC Label	DC	3
30.	CM Label	FC	3
31.	CMA	2F	1
32.	CMC	3F	1
33.	CMP A	BF	1
34.	CMP B	B8	1
35.	CMP C	B9	1
36.	CMP D	BA	1
37.	CMP E	BB	1
38.	CMP H	BC	1
39.	CMP L	BD	1
40.	CMP M	BD	1
41.	CNC Label	D4	3
42.	CNZ Label	C4	3
43.	CP Label	F4	3
44.	CPE Label	EC	3
45.	CPI Data	FE	2
46.	CPO Label	E4	3
47.	CZ Label	CC	3
48.	DAA	27	1
49.	DAD B	09	1
50.	DAD D	19	1
51.	DAD H	29	1
52.	DAD SP	39	1
53.	DCR A	3D	1
54.	DCR B	05	1
55.	DCR C	0D	1
56.	DCR D	15	1
57.	DCR E	1D	1
58.	DCR H	25	1
59.	DCR L	2D	1
60.	DCR M	35	1

Sr. No.	Mnemonics, Operand	Opcode	Bytes
61.	DCX B	0B	1
62.	DCX D	1B	1
63.	DCX H	2B	1
64.	DCX SP	3B	1
65.	DI	F3	1
66.	EI	FB	1
67.	HLT	76	1
68.	IN Port-address	DB	2
69.	INR A	3C	1
70.	INR B	04	1
71.	INR C	0C	1
72.	INR D	14	1
73.	INRE	1C	1
74.	INR H	24	1
75.	INR L	2C	1
76.	INR M	34	1
77.	INX B	03	1
78.	INX D	13	1
79.	INX H	23	1
80.	INX SP	33	1
81.	JC Label	DA	3
82.	JM Label	FA	3
83.	JMP Label	C3	3
84.	JNC Label	D2	3
85.	JNZ Label	C2	3
86.	JP Label	F2	3
87.	JPE Label	EA	3
88.	JPO Label	E2	3
89.	JZ Label	CA	3
90.	LDA Address	3A	3
91.	LDAX B	0A	1
92.	LDAX D	1A	1
93.	LHLD Address	2A	3
94.	LXI B	01	3
95.	LXI D	11	3
96.	LXI H	21	3
97.	LXI SP	31	3
98.	MOV A, A	7F	1
99.	MOV A, B	78	1
100.	MOV A, C	79	1
101.	MOV A, D	7A	1

Sr. No.	Mnemonics, Operand	Opcode	Bytes
102.	MOV A, E	7B	1
103.	MOV A, H	7C	1
104.	MOV A, L	7D	1
105.	MOV A, M	7E	1
106.	MOV B, A	47	1
107.	MOV B, B	40	1
108.	MOV B, C	41	1
109.	MOV B, D	42	1
110.	MOV B, E	43	1
111.	MOV B, H	44	1
112.	MOV B, L	45	1
113.	MOV B, M	46	1
114.	MOV C, A	4F	1
115.	MOV C, B	48	1
116.	MOV C, C	49	1
117.	MOV C, D	4A	1
118.	MOV C, E	4B	1
119.	MOV C, H	4C	1
120.	MOVC,L	4D	1
121.	MOV C, M	4E	1
122.	MOV D, A	57	1
123.	MOV D, B	50	1
124.	MOV D, C	51	1
125.	MOV D, D	52	1
126.	MOV D, E	53	1
127.	MOV D, H	54	1
128.	MOV D, L	55	1
129.	MOV D, M	56	1
130.	MOV E, A	5F	1
131.	MOV E, B	58	1
132.	MOV E, C	59	1
133.	MOV E, D	5A	1
134.	MOV E, E	5B	1
135.	MOV E, H	5C	1
136.	MOV E, L	5D	1
137.	MOV E, M	5E	1
138.	MOV H, A	67	1
139.	MOV H, B	60	1
140.	MOV H, C	61	1
141.	MOV H, D	62	1
142.	MOV H, E	63	1

Sr. No.	Mnemonics, Operand	Opcode	Bytes
143.	MOV H, H	64	1
144.	MOV H, L	65	1
145.	MOV H, M	66	1
146.	MOV L, A	6F	1
147.	MOV L, B	68	1
148.	MOV L, C	69	1
149.	MOV L, D	6A	1
150.	MOV L, E	6B	1
151.	MOV L, H	6C	1
152.	MOV L, L	6D	1
153.	MOV L, M	6E	1
154.	MOV M, A	77	1
155.	MOV M, B	70	1
156.	MOV M, C	71	1
157.	MOV M, D	72	1
158.	MOV M, E	73	1
159.	MOV M, H	74	1
160.	MOV M, L	75	1
161.	MVI A, Data	3E	2
162.	MVI B, Data	06	2
163.	MVI C, Data	0E	2
164.	MVI D, Data	16	2
165.	MVI E, Data	1E	2
166.	MVI H, Data	26	2
167.	MVIL,	2E	2
168.	MVI M, Data	36	2
169.	NOP	00	1
170.	ORA A	B7	1
171.	ORA B	B0	1
172.	ORA C	B1	1
173.	ORA D	B2	1
174.	ORA E	B3	1
175.	ORA H	B4	1
176.	ORA L	B5	1
177.	ORA M	B6	1
178.	ORI Data	F6	2
179.	OUT Port-Address	D3	2
180.	PCHL	E9	1
181.	POP B	C1	1
182.	POP D	D1	1
183.	POP H	E1	1

Sr. No.	Mnemonics, Operand	Opcode	Bytes
184.	POP PSW	F1	1
185.	PUSH B	C5	1
186.	PUSH D	D5	1
187.	PUSH H	E5	1
188.	PUSH PSW	F5	1
189.	RAL	17	1
190.	RAR	1F	1
191.	RC	D8	1
192.	RET	C9	1
193.	RIM	20	1
194.	RLC	07	1
195.	RM	F8	1
196.	RNC	D0	1
197.	RNZ	C0	1
198.	RP	F0	1
199.	RPE	E8	1
200.	RPO	E0	1
201.	RRC	0F	1
202.	RST 0	C7	1
203.	RST 1	CF	1
204.	RST 2	D7	1
205.	RST 3	DF	1
206.	RST 4	E7	1
207.	RST 5	EF	1
208.	RST 6	F7	1
209.	RST 7	FF	1
210.	RZ	C8	1
211.	SBB A	9F	1
212.	SBB B	98	1
213.	SBB C	99	1
214.	SBBD	9A	1
215.	SBB E	9B	1
216.	SBB H	9C	1
217.	SBB L	9D	1
218.	SBB M	9E	1
219.	SBI Data	DE	2
220.	SHLD Address	22	3
221.	SIM	30	1
222.	SPHL	F9	1
223.	STA Address	32	3
224.	STAX B	02	1

Sr. No.	Mnemonics, Operand	Opcode	Bytes
225.	STAX D	12	1
226.	STC	37	1
227.	SUB A	97	1
228.	SUB B	90	1
229.	SUB C	91	1
230.	SUB D	92	1
231.	SUB E	93	1
232.	SUB H	94	1
233.	SUB L	95	1
234.	SUB M	96	1
235.	SUI Data	D6	2
236.	XCHG	EB	1
237.	XRA A	AF	1
238.	XRA B	A8	1
239.	XRA C	A9	1
240.	XRA D	AA	1
241.	XRA E	AB	1
242.	XRA H	AC	1
243.	XRA L	AD	1
244.	XRA M	AE	1
245.	XRI Data	EE	2
246.	XTHL	E3	1

Few More 8085 Programming Examples

1. **Write a program to perform the following:**
   a) Load the no:1 BH in D
   b) Load the no. B5H in B
   c) Increment the content of Bby 1.
   d) Decrement the content of Dby1.
   e) Subtract the content of D from the content of B.
   f) Display the result at OUTport1.

Solution:

```
MVI D,1BH
MVI B,B5H
INR B
DCR D
MOV A,B
SUB D
OUT PORT1
HLT
```

2. **Wap to load the data byte in the register C. Mask the high-order bits (D7-D4) and display the low order bits (D3-D0) at OUT PORT1. Exclusive-OR the result with 57 Hand display at OUT PORT 2.**

Solution

```
MVI C,A8H
MOV A,C
ANI OFH
OUT PORT1
XRI 57H
OUT PORT2
HLT
```

3. **Wap to load the byte 8EH in register D and F7H in register E. Mask the higher order bits (D7-D4) from both the data bytes, EX-OR the low order bit (D3-D0) and display the answer.**

Solution

```
MVI D,8EH
MOV A,D
ANI 0FH
MVI D,8EH
MVI E,F7H
MOV A,D
ANI OFH
MOV D,A
MOV A,E
ANI OFH
XRA D
OUT PORT1
HLT
```

4. **Write a program to load two unsigned nos in register B and C respectively. Subtract C from B. If the result in 2's complement convert the result in absolute magnitude and display it port 1. Otherwise display the result.**

Solution:

```
    MVI B, byte 1
    MVI C, byte 2
    MOV A,B
    SUB C
    JNC label1
    CMA
    ADI 01H
Label 1: OUT PORT1
    HLT
```

5. ***Write an ALP to do the following:***
   a) Load A with byte 1.
   b) Load B with byte 2.
   c) Compare the equality of the contents of A and B
   d) If two nos. are equal, display 01 otherwise display 00H at port 1.

Solution:

```
    MVI A, byte1
    MVI B, byte2
    SUB B
    JNZ loop
    MVI A,01H
    OUT PORT1
    HLT
Loop: MVIA,00H
    OUT  PORT
    HLT
```

6. **The following block of data is stored in memory location from CO55 to C05AH. Transfer the entire block of data to the locations C080 to C085H in reverse order. Data:22,A5,B2,99,7F,37**

Solution:

```
        LXI H, C055H
        LXI D, C085H
        MOV B, 06H
Next:   MVI A,M
        STAXD
        INXH
        DCXD
        DCRB
        JNZ next
        HLT.
```

7. **Write a program to find larger of two nos. 1st no in C001 and 2nd no inC002 and result inC003H.**

Solution:

```
        LXI H, C001H
        MOV A, M
        INX H
        CMP M
        JNC loop
        MOVA,M
loop:   STA C003H
        HLT
```

8. **Write an ALP to find the smallest no in a data array. Data from location C000H to C005H.**

Solution:

```
        LXI H,C000H
        MVI C,06H
        MOV A,M
        DCR C
loop:   INX H
        CMP M
        JC loop1
        MOV A,M
loop1:  DCR C
        JNZ loop
        STA C0C0H
        HLT.
```

9. **Write an ALP to multiply two nos: eg05H × 08H**

Solution:

```
        MVI A,00H
        MVI B,08H
        MVI C,05H
Loop:   ADDB
        DCRC
        JNZ loop
        STA C000H
        HLT
```

10. Write an ALP for the following addition12+22+32+42+52+62+72+82+92

Solution:

```
        MVI A,00H
        MVI B,09H
Loop1:  MOV C,B
Loop2:  ADD B
        DCR C
        JNZ Loop2
        DCR B
        JNZ Loop1
        OUT PORT1
        HLT
```

11. Write an ALP to count the no of 1 in the given string

'10100110' and display the result at COCOH

Solution:

```
        MVI A, A6H
        MVI B,00H
        MVI C,08H
Loop1:  RAL
        JNC loop2
        INR B
Loop2:  DCR C
        JNZ loop1
        MOV A,B
        STA COCOH
        HLT
```

12. The following datas are stored in memory location starting from C0B0 to C0B9H. Take a test no. 48. Find out how many times the no 48 is repeated. Display the result at C0C0H.

DADA:12,23,34,45,48,56,48,67,48,89.

Solution:

```
        LXI H,C0B0H
        MVI B,00H
        MVI C,0AH
Loop1:  MOV A,M
        CPI 48H
        JNZ loop2
        INR B
Loop2:  INX H
        DCR C
        JNZ loop1
        MOV A,B
        STAC OCOH
        HLT
```

13. **8 bit multiplication, product is 16 bit. The multiplicand is loaded in the two consecutive memory locations 2501 and 2502H. The multiplier is stored in 2053H. Store the product in 2504 and 2505H.**

Solution:

```
        LHLD 2501H
        XCHG
        LDA 2503H
        LXI H,0000
        MVI C,08
Loop1:  DAD H
        RAL
        JNC loop2
        DAD D
Loop2:  DCR C
        JNZ loop1
        SHLD 2504H
        HLT
```

14. **Write an ALP to divide two nos. The dividend is in C001 and divisor is in C002. Store the quotient in C0C0H and remainder in C0C1.**

Solution:

```
        LXIH, C001H
        MOV A,M
        INX H
        MOV B,M
        MVI C,00H
Loop1:  CMPB
        JC Loop2
        INR C
        SUB B
        JNZ Loop1
Loop2:  STA COC1H
        MOV A,C
        STA COCOH
        HLT
```

15. **To arrange 54, EB, 85, A8 & 99 in descending order. These numbers are stored in the memory location 2501 to 2505H. The count = 05 is restored in 2500H. Results are to be stored in 2601 to 2605H.**

Solution:

```
        LXI D, 2601H
        LXI H, 2500H
        MOV B, M
Start:  CALL Subroutine1
        STAX D
        CALL Subroutine2
        INX D
        DCR B
        JNZ Start
        HLT
```

```
Subroutine1: LXI H,2500H
             SUB A
     Loop1:  INX H
             CMP M
             JNC Loop2
             MOV A,M
     Loop2:  DCRC
             JNZ Loop1
             RET
Subroutin2:  LXI H,2500H
             MOV C,M
     Loop1:  INX H
             CMP M
             JZ Loop2
             DCR C
             JNZ Loop1
     Loop2:  MVI A,00H
             MOV M,A
             RET
```

Note: Subroutine1 gives the largest number of array.
Subroutine2 find the largest number and replace it by 00.

Counter and delay:

Timing delay using one register:

MVIC,FFH----------7Tstate
Loop DCR C----------4Tstate
JNZ Loop--------10or7Tstate
Consideramicrocomputerwith2MHZfrequency
Clock period, T=1/f=½=0. 5µsec
Delay for inst. Outside the loop To = No of Ts state × T
= 7×0. 5 = 3. 5µsec
Delay for inst inside the loop, TL = No of T state × T*(N10)
=(14×0. 5×10-6×255) =1785µsec

Now, TLA = TL – 3 × 0. 5
= 1785 – 1. 5 = 1. 7835 ms
TD = T0 + TL
= 3. 5 + 1785 = 1788. 5 = 1. 7885 ms

Time delay for register pair:

```
        LXI B,2384H..... 10Tstate
Loop:   DCX B..... 6Tstate
        MOV A,C..... . 4Tstate
        ORAB.....4Tstate
        JNZ loop..... 10/7
```

Delay calculation:

T0 = Tstate*T
=10×0. 5×10-3 =109ms
TL =No of T state*T*(N10)
=24*0. 5*90. 92

Total Delay = T0 + TL

Time delay using a loop within a loop:

```
        MVI B,38H..... . 7T
Loop2   MVI C,FFH.....7T
Loop1   DCRC..... . 4T
        JNZ loop1------10/7T
        DCR B------4T
        JNZ loop2------10/7T
```

Delay calculation:

To = 7*0. 5 =3. 5μsec

TL1 = 14*0. 5*255–3*0. 5

TL2 = (TL1+21*0. 5)N10–3

Total delay =To+TL2

16. **Write a program to count continuously in hexadecimal from FFH to 00H in a system with a0. 5 μs clock period. Use register C to setup a 1ms delay between each count and display the number at one of the o/p ports.**

```
        MVI B,00H
Next:   DCR B
        MVI C,count
Delay:  DCRC
        JNZ Delay
        MOV A,B
        OUT port1
        JMP Next
        HLT
```

Delay calculation:

TL = Tstate*T*(T10)

= 14×0. 5×count

= 7×countμs

To = 35×T

= 35×0. 5

=17. 5μs

TD = TL + To

1ms = 7*count*10-6+17. 5*10-6

Count =140. 35 =8CH

17. **Write a program to generate a continuous square wave with the period 500 μsec. Assume the system clock period is 325 μ sec and use bit Do to output the square wave.**

Solution:

```
        MVI D,AA
ROTATE: MOVA,D
        RLC
        MOV D,A
        ANI 01H
        OUT PORT1
        MVI B,count
Delay:  DCRB
        JNZ Delay
        JMP ROTATE
        HLT
```

Delay calculation:

TL = 14*325*10-9*countor14*325*(count-1)+11T-state*325

T0 = 46*325*10-9

TD = TL+T0

250 = (52. 4)10=34H

18. Write an assembly language program in 8085 microprocessor to subtract two 16 bit numbers.

Assumption –

- Starting address of program: 2000
- Input memory location: 2050, 2051, 2052, 2053
- Output memory location: 2054, 2055

Example –

INPUT:

(2050H) = 19H
(2051H) = 6AH
(2052H) = 15H
(2053H) = 5CH

OUTPUT:

(2054H) = 04H
(2055H) = OEH

RESULT:

Hence we have subtracted two 16 bit numbers.

Algorithm

1. Get the LSB in L register and MSB in H register of 16 Bit number.
2. Exchange the content of HL register with DE register.
3. Again Get the LSB in L register and MSB in H register of 16 Bit number.
4. Subtract the content of L register from the content of E register.
5. Subtract the content of H register from the content of D register and borrow from previous step.
6. Store the result in memory location.

Program –

MEMORY ADDRESS	MNEMONICS	COMMENTS
2000	LHLD 2050	Load H-L pair with address 2050
2003	XCHG	EXCHANGE H-L PAIR WITH D-E PAIR
2004	LHLD 2052	Load H-L pair with address 2052
2007	MVI C, 00	C<-00H
2009	MOV A, E	A<-E
200A	SUB L	A<-A-L
200B	STA 2054	2054<-A
200E	MOV A, D	A<-D
200F	SBB H	SUBTRACT WITH BORROW
2010	STA 2055	2055<-A
2013	HLT	TERMINATES THE PROGRAM

Explanation –

1. **LHLD 2050:** load HL pair with address 2050.
2. **XCHG:** exchange the content of HL pair with DE.
3. **LHLD 2052:** load HL pair with address 2050.
4. **MOV A, E:** move the content of register E to A.
5. **SUB L:** subtract the content of A with the content of register L.
6. **STA 2054:** store the result from accumulator to memory address 2054.
7. **MOV A, D:** move the content of register D to A.
8. **SBB H:** subtract the content of A with the content of register H with borrow.
9. **STA 2055:** store the result from accumulator to memory address 2055.
10. **HLT: stops executing the program and halts any further execution.**

19. Write an assembly language program in 8085 microprocessor to multiply two 16 bit numbers.

Assumption –

- Starting address of program: 2000
- Input memory location: 2050, 2051, 2052, 2053
- Output memory location: 2054, 2055, 2056, 2057

Example –

INPUT:

(2050H) = 04H

(2051H) = 07H

(2052H) = 02H

(2053H) = 01H

OUTPUT:

(2054H) = 08H

(2055H) = 12H

(2056H) = 01H

(2057H) = O0H

RESULT:

Hence we have multiplied two 16 bit numbers.

Algorithm –

1. Load the first data in HL pair.
2. Move content of HL pair to stack pointer.
3. Load the second data in HL pair and move it to DE.
4. Make H register as 00H and L register as 00H.
5. ADD HL pair and stack pointer.
6. Check for carry if carry increment it by 1 else move to next step.
7. Then move E to A and perform OR operation with accumulator and register D.
8. The value of operation is zero, then store the value else go to step 3.

Program –

MEMORY ADDRESS	MNEMONICS	COMMENTS
2000	LHLD 2050	Load H-L pair with address 2050
2003	SPHL	SAVE IT IN STACK POINTER
2004	LHLD 2052	Load H-L pair with address 2052
2007	XCHG	EXCHANGE HL AND DE PAIR CONTENT
2008	LXI H,0000H	H<-00H,L<-00H
200B	LXI B,0000H	B<-00H,C<-00H
200E	DAD SP	
200F	JNC 2013	JUMP NOT CARRY
2012	INX B	INCREMENT BC BY 1
2013	DCX D	DECREMENT DE BY 1
2014	MOV A,E	A<-E
2015	ORA D	OR THE CONTENT OF ACCUMULATOR AND D REGISTER
2016	JNZ 200E	JUMP NOT ZERO
2019	SHLD 2054	L<-2053,H<-2054
201C	MOV L,C	L<-C
201D	MOV H,B	B<H
201E	SHLD 2056	L<-2055,H<-2056
2021	HLT	TERMINATES THE PROGRAM

Explanation – Registers B, C, D, E, H, L and accumulator are used for general purpose.

1. **LHLD 2050:** load HL pair with address 2050.
2. **SPHL:** save the content of HL in stack pointer.
3. **LHLD 2052:** load H-L pair with address 2052.
4. **XCHG:** exchange the content of HL pair with DE.
5. **LXI H, 0000H:** make H as 00H and L as 00H.
6. **LXI B, 0000H:** make B as 00h and C as 00H
7. **DAD SP:** ADD HL pair and stack pointer.
8. **JNC 2013:** jump to address 2013 if there will be no carry.
9. **INX B:** increments BC register with 1.
10. **DCX D:** decrements DE register pair by 1.

11. **MOV A, E:** move the content of register E to accumulator.
12. **ORA D:** or the content of accumulator and D register.
13. **JNZ 200E:** jump to address 200E if there will be no zero.
14. **SHLD 2054:** store the result to memory address 2054 and 2055 from HL pair register.
15. **MOV L, C:** move the content of register C to L.
16. **MOV H, B:** move the content of register B to H.
17. **SHLD 2056:** store the result to memory address 2056 and 2057 from HL pair register.
18. **HLT:** terminates the program.

20. Write an assembly language program in 8085 microprocessor to divide two 16 bit numbers.

Assumption –

- Starting address of program: 2000
- Input memory location: 2050, 2051, 2052, 2053
- Output memory location: 2054, 2055, 2056, 2057.

Example –

INPUT:

(2050H) = 04H

(2051H) = 00H

(2052H) = 02H

(2053H) = 00H

OUTPUT:

(2054H) = 02H

(2055H) = 00H

(2056H) = FEH

(2057H) = FFH

RESULT:

Hence we have divided two 16 bit numbers.

Algorithm –

1. Intialise register BC as 0000H for Quotient.
2. Load the divisor in HL pair and save it in DE register pair.
3. Load the dividend in HL pair.
4. Subtract the content of accumulator with E register.
5. Move the content A to C and H to A.
6. Subtract with borrow the content of A with D.
7. Move the value of accumulator to H.
8. If CY=1, goto step 10, otherwise next step.
9. Increment register B and jump to step 4.
10. ADD both contents of DE and HL.
11. Store the remainder in memory.
12. Move the content of C to L & B to H.
13. Store the quotient in memory.

Program –

MEMORY ADDRESS	MNEMONICS	COMMENTS
2000	LXI B, 0000H	INITIALISE QUOTIENT AS 0000H
2003	LHLD 2052H	LOAD THE DIVISOR IN HL
2006	XCHG	EXCHANGE HL AND DE
2007	LHLD 2050	LOAD THE DIVIDEND
200B	MOV A, L	A<-L
200C	SUB E	A<-A-E
200D	MOV L, A	L<-A
200E	MOV A, H	A<-H
200F	SBB D	A<-A-D
2010	MOV H, A	H<-A
2011	JC 2018	JUMP WHEN CARRY
2014	INX B	B<-B+1
2015	JMP 200B	
2018	DAD D	HL<-DE+HL
2019	SHLD 2056	HL IS STORED IN MEMORY
201C	MOV L, C	L<-C
201D	MOV H, B	H<-B
201E	SHLD 2054	HL IS STORED IN MEMORY
2021	HLT	TERMINATES THE PROGRAM

Explanation

1. **LXI B, 0000H:** intialise BC register as 0000H.
2. **LHLD 2052H:** load the HL pair with address 2052.
3. **XCHG:** exchange the content of HL pair with DE pair register.
4. **LHLD 2050:** load the HL pair with address 2050.
5. **MOV A, L:** move the content of register L into register A.
6. **SUB E:** subtract the contents of register E with contents of accumulator.
7. **MOV L, A:** move the content of register A into register L.
8. **MOV A, H:** move the content of register H into register A.
9. **SBB D:** subtract the contents of register D with contents of accumulator with carry.
10. **MOV H, A:** move the content of register A into register H.
11. **JC 2018:** jump to address 2018 if there is carry.
12. **INX B:** increment BC register by one.
13. **JMP 200B:** jump to address 200B.
14. **DAD D:** add the contents of DE and HL pair.
15. **SHLD 2056:** stores the content of HL pair into memory address 2056 and 2057.
16. **MOV L, C:** move the content of register C into register L.
17. **MOV H, B:** move the content of register B into register H.
18. **SHLD 2054:** stores the content of HL pair into memory address 2054 and 2055.
19. **HLT:** terminates the execution of program.

21. A program to separate even numbers from the given list of numbers and store them in the another list starting from 7000H. Assume starting address of number list is 6000H.

Label	Instruction	Memory Address	Memory Content		
			OP-Code	Low A/D	High A/D
	LXI H 6000H	8000	21	00	60
	LXI D 7000H	80003	11	00	70
	MVI C 06H	8006	0E	06	
TOP:	MOVA M	8008	7E		
	ANI 01	8009	E6	01	
	JNZ DOWN	800B	C2	11	80
	MOVA M	800E	7E		
	STAXD	800F	12		
	INXD	8010	13		
DOWN:	INXH	8011	23		
	DCRC	8012	0D		
	JNZ TOP	8013	C2	08	80
	HLT	8014	76		

This is the data entered by the user.

User Data Grid

Address	Data
6000	10
6001	11
6002	12
6003	13
6004	14
6005	15

This is the output obtained. The even numbers from the input list have been separated and stored in an address list starting from 7000h.

User Data Grid

Address	Data
6000	10
6001	11
6002	12
6003	13
6004	14
6005	15
7000	10
7001	12
7002	14
7003	00
7004	00
7005	00

22. A program to calculate the sum of series of even number from list of numbers.

Solution:

```
        LDA 8080H
        MOV C A          ; Initialize the counter
        MVI B 00H        ; Sum = 0
        LXI H 8081H      ; Initialize the pointer
BACK :  MOV A M          ; Get the number
        ANI 01H          ; Checks whether content of accumulator is odd
        JNZ SKIP         ; Don't add if odd
        MOV A B          ; Get the sum
        ADD M            ; Sum = sum * number
        MOV B A          ; Store the content of accumulator to B-Resister
SKIP:   INX H            ; Increase pointer by 1
        DCR C            ; Decrease Pointer by 1
        JNZ BACK         ; Continue LOOP until counter is 0
        STA 8090H        ; Store the sum
        HLT
```

User Data Grid	
Address	Data
8080	08
8081	10
8082	12
8283	13
8084	05
8085	15
8086	10
8087	06
8088	20
8090	58 (Output)

23. A table having starting address 3060h contains ten numbers of 8-bit data. Write a program in 8085 that transfer the data to next table having starting address 3070h if the data is greater than 90h else store it to next table at 3080h.

Solution:

```
        LXI H 3060h      ; Initialize pointer for storing data
        LXI D 3070h      ; Table to store data greater than 90
        LXI B 3080h      ; Table to store data smaller than 90
TOP:    MOV A M          ; Get the value to accumulator
        CPI 90h          ; Checks whether the value in accumulator is greater/smaller than 90
        JNC DOWN
        STAX B           ; Store to BC register if data is less than 90
        INX B            ; Increase the address of BC pair by 1
        JMP MID          ; Jump to Mid without checking condition
DOWN:   STAX D           ; Store to DE register if data is greater than 90
        INX D            ; Increase the address of DE pair by 1
MID:    INX H            ; Increase the Pointer address by 1
        CPI 01h          ; Checks next data (11th value) is zero or not
        JNC TOP
```

HLT

Input Data	
3060	22
3061	25
3062	88
3063	96
3064	99
3065	66
3066	88
3067	55
3068	44
3069	44

Output Data greater than 90	
3070	96
3071	99
3072	00
3073	00
3074	00
3075	00
3076	00
3077	00
3078	00
3079	00

Output Data smaller than 90	
3080	22
3081	25
3082	88
3083	66
3084	88
3085	55
3086	44
3087	44
3088	00
3089	00

24. A program to find the factorial of 'n' numbers.

Solution:

```
        MVI B 04H
        MOV C B
        DCR C
LOOP1:  MOV E C
        SUB A
LOOP2:  ADD B
        DCR E
        JNZ LOOP2
        MOOV B, A
        DCR C
        JNZ LOOP1
        STA 8000H
        HLT
```

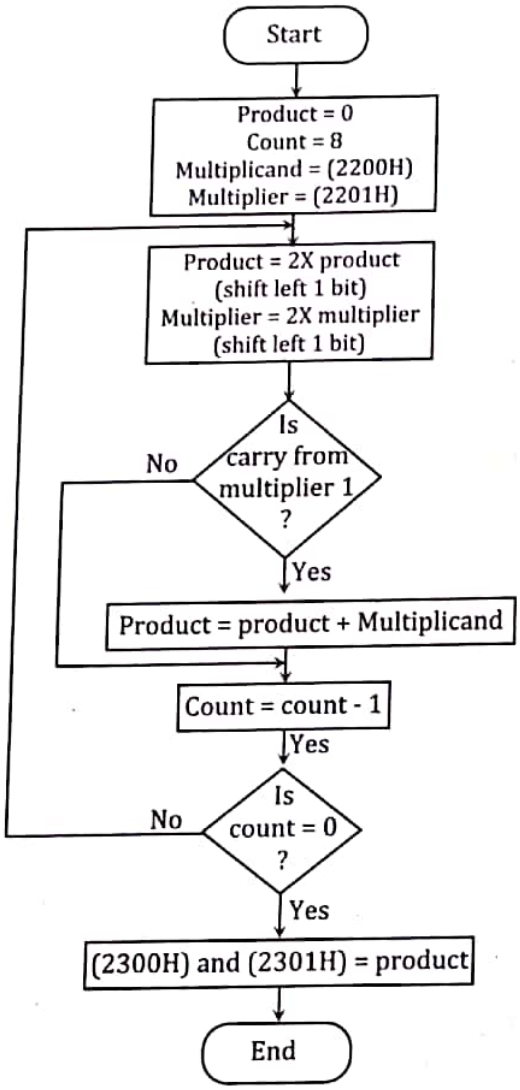
25. Multiply the 8-bit unsigned number in memory location 2200H by the 8-bit unsigned number in memory location 2201H. Store the 8 least significant bits of the result in memory location 2300H and the 8 most significant bits in memory location 2301H.

Solution:

```
        LXI H, 2200   ;  Initialize the memory pointer
        MOV E, M      ;  Get multiplicand
        MVI D, 00H    ;  Extend to 16-bits
        INX H         ;  Increment memory pointer
        MOV A, M      ;  Get multiplier
        LXI H, 0000   ;  Product = 0
        MVI B, 08H    ;  Initialize counter with count 8
MULT:   DAD H         ;  Product = product x 2
        RAL
        JNC SKIP      ;  Is carry from multiplier 1?
        DAD D         ;  Yes, Product =Product + Multiplicand
SKIP    ;  DCR B ; Is counter = zero
        JNZ MULT      ;  no, repeat
        SHLD 2300H    ;  Store the result
        HLT           ;  End of program
```

26. To perform the division of two 8 bit numbers using 8085.

Solution:



Algorithm

1. Start the program by loading HL register pair with address of memory location.
2. Move the data to a register (B register).
3. Get the second data and load into Accumulator.
4. Compare the two numbers to check for carry.
5. Subtract the two numbers.
6. Increment the value of carry.
7. Check whether repeated subtraction is over and store the value of product and carry in memory location.
8. Terminate the program.

Program

```
        LXI  H, 4150
        MOV  B,M        Get the dividend in B – reg.
        MVI  C,00       Clear C – reg for quotient
        INX  H
        MOV A,M         Get the divisor in A – reg.
```

```
NEXT:   CMP B       Compare A - reg with register B.
        JC  LOOP    Jump on carry to LOOP
        SUB  B      Subtract A – reg from B- reg.
        INR C       Increment content of register C.
        JMP NEXT    Jump to NEXT
LOOP:   STA 4152    Store the remainder in Memory
        MOV A,C
        STA 4153    Store the quotient in memory
        HLT         Terminate the program.
```

Observation:

Input:

0F (4150)

FF (4251)

Output:

01 (4152) ---- Remainder

FE (4153) ---- Quotient

27. **Suppose the size of the array is stored at memory location 2050 and the base address of the array is 2051. The sum will be stored at memory location 3050 and carry will be stored at location 3051.**

Solution:

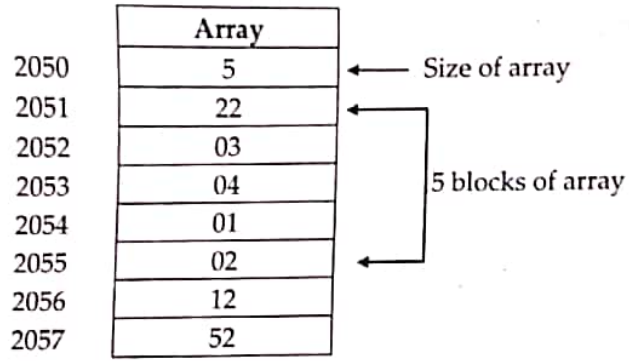
Algorithm

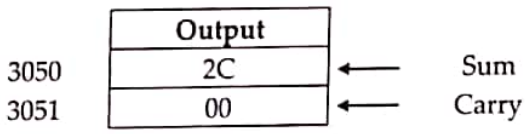
1. Load the base address of the array in HL register pair.
2. Use the size of the array as a counter.
3. Initialize accumulator to 00.
4. Add content of accumulator with the content stored at memory location given in HL pair.
5. Decrease counter on each addition.

Program

MNEMONICS	COMMENTS
LDA 2050	[A] ←[2050]
MOV B, A	[B] ←[A]
LXI H, 2051	[H]← 20 and [L]← 51
MVI A, 00	[A]← 00
MVI C, 00	[C]← 00
ADD M	[A]←[A]+[M]
INR M	[M]←[M]+1
JNC 2011	
INR C	[C]←[C]+1
DCR B	[B]←[B]-1
JNZ 200B	
STA 3050	[3050]←[A]
MOV A, C	[A]←[C]
STA 3051	[3051]←[A]
HLT	Terminates the program

Observation





28. Write an assembly language program in 8085 microprocessor to sort a given list of n numbers using Bubble Sort. Size of list is stored at 2040H and list of numbers from 2041H onwards. (8085 program for bubble sort)

Solution:

Algorithm

1. Load size of list in C register and set D register to be 0
2. Decrement C as for n elements n-1 comparisons occur
3. Load the starting element of the list in Accumulator
4. Compare Accumulator and next element
5. If accumulator is less than next element jump to step 8
6. Swap the two elements
7. Set D register to 1
8. Decrement C
9. If C>0 take next element in Accumulator and go to point 4
10. If D=0, this means in the iteration, no exchange takes place consequently we know that it won't take place in further iterations so the loop in exited and program is stopped
11. Jump to step 1 for further iteration

Program

LABEL	INSTRUCTION	COMMENT
START:	LXI H, 2040H	Load size of array
	MVI D, 00H	Clear D register to set up a flag
	MOV C, M	Set C register with number of elements in list
	DCR C	Decrement C
	INX H	Increment memory to access list
CHECK:	MOV A, M	Retrieve list element in Accumulator
	INX H	Increment memory to access next element
	CMP M	Compare Accumulator with next element
	JC NEXTBYTE	If accumulator is less then jump to NEXTBYTE
	MOV B, M	Swap the two elements
	MOV M, A	Move the content of accumulator to the memory
	DCX H	Decrement HL by 1
	MOV M, B	Move the content of B into memory
	INX H	Increment the content of HL by 1
	MVI D, 01H	If exchange occurs save 01 in D register

NEXTBYTE:	DCR C	Decrement C for next iteration
	JNZ CHECK	Jump to CHECK if C>0
	MOV A, D	Transfer contents of D to Accumulator
	CPI 01H	Compare accumulator contents with 01H
	JZ START	Jump to START if D=01H
	HLT	HALT

Observation

		List				
Size	Input	35H	10H	02H	21H	F0H
05H		List				
	Output	02H	10H	21H	35H	F0H

29. Write an assembly language program for converting a 2 digit BCD number to its binary equivalent using 8085 microprocessor.

Solution:

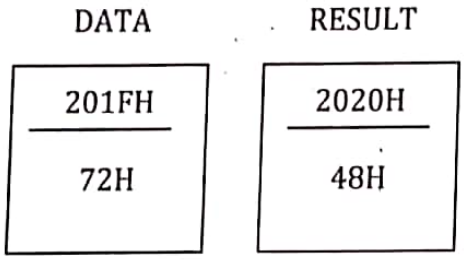
Algorithm

1. Load the BCD number in the accumulator
2. Unpack the 2 digit BCD number into two separate digits. Let the left digit be BCD1 and the right one BCD2
3. Multiply BCD1 by 10 and add BCD2 to it
4. If the 2 digit BCD number is 72, then its binary equivalent will be 7 x OAH + 2 = 46H + 2 = 48H

Program:

LABEL	MNEMONIC
	LDA 201FH
	MOV B, A
	ANI 0FH
	MOV C, A
	MOV A, B
	ANI F0H
	JZ SKIPMULTIPLY
	RRC
	RRC
	RRC
	RRC
	MOV D, A
	XRA A
	MVI E, 0AH
SUM	ADD D
	DCR E
	JNZ SUM
SKIPMULTIPLY	ADD C
	STA 2020H
	HLT

Observation



4.3 Programming with 8086 Microprocessor

Assembly Language Program Development Tools

1. Editor

- An Editor is a Program which allows you to create a file containing the Assembly Language statements for your Program.

2. Assembler

- An Assembler Program is used to translate the assembly language mnemonics for instruction to the corresponding binary codes.

3. Linker

- A Linker is a Program used to join several files into one large. obj file. It produces .exe file so that the program becomes executable.

4. Locator

- A Locator is a program used to assign the specific address of where the segment of object code are to be loaded into memory.
- It usually converts .exe file to .bin file.
- A Locator program EXE2BINconverts. exe file to. bin file.

5. Debugger

- A Debugger is a program which allows you to load your. obj code program into system memory, execute program and troubleshoot.
- It allows you to look at the content of registers and memory locations after your program runs.
- It allows to set the break point.

Emulator

- An Emulator is a mixture of hardware and software.
- It is used to test and debug the hardware and software of an external system such as the prototype of a Microprocessor based system.

Assembly Language Program Features

 # Program Comments

- The use of Comments throughout a program can improve its clarity, especially in Assembly Language.
- A Comment begins with Semicolon. Example: ADDAX, BX; Adds the Content of BX with AX

 # Reserved Words

- Instructions : MOV,ADD
- Directives : END,SEGMENT
- Operators : FAR,OFFSET
- Predefined Symbols : @DATA

 # Identifiers

- An Identifier (or symbol) is a name that you apply to an item in your program that you expect to reference. The two types of identifiers are NAME and LABEL.
- NAME: Refers to the Address of a data item COUNTER DB 0
- LABEL: Refer to the Address of an instruction, procedure, or segment.

```
MAIN PROC
A20: MOV AL, BL
```

 # Statements

- An Assembly Program consists of a set of statements. The two types of statements are:

1. **Instruction**

- Instructions such as MOV & ADD which the Assembler translates to Object Code.

2. **Directives**

- Directives tell the Assembler to perform a specific action, such as define a data item, etc.

ASSEMBLY LANGUAGE PROGRAMMING USING MASM
GENERAL PATTERN FOR WRITING ALP IN MASM
[PAGE DIRECTIVE]
[TITLE DIRECTIVE]
[MEMORY MODEL DEFINITION]
[SEGMENT DIRECTIVES]
[PROC DIRECTIVES]
.....
.....
.....
.....
[END DIRECTIVES]

BASIC FORMAT OF ALP BASED UPON THE GENERAL PATTERN

```
PAGE 60, 80
TITLE "ALP TO PRINT FACTORIAL NO"
. MODEL [MODEL NAME]
. STACK
. DATA
.....; INITIALIZE DATA VARIALBLES
. CODE

MAIN PROC
.....
.....
.....
.....; INSTRUCTION SETS
.....
.....
.....
.....
MAIN ENDP
END MAIN
```

DIRECTIVES

Assembly Language supports a number of statements that enable to control the way in which a source program assembles and lists. These Statements are called Directives.
They act only during the assembly of a program and generate no machine executable code.
The most common Directives are PAGE, TITLE, PROC, and END.

PAGE DIRECTIVE

- The PAGE Directive helps to control the format of a listing of an assembled program.
- It is optional Directive.
- At the start of program, the PAGE Directive designates the maximum number of lines to list on a page and the maximum number of characters on a line.
- Its format is: PAGE [LENGTH], [WIDTH]
- Omission of a PAGE Directive causes the assembler to set the default value to PAGE 50,80

TITLE DIRECTIVE

- The TITLE Directive is used to define the title of a program to print on line 2 of each page of the program listing.
- It is also optional Directive.
- Its format is TITLE [TEXT]
- TITLE "PROGRAM TO PRINT FACTORIAL NO"

SEGMENT DIRECTIVE

- The SEGMENT Directive defines the start of a segment.
- A Stack Segment defines stack storage, a data segment defines data items and a code segment provides executable code.
- MASM provides simplified Segment Directive.

The format (including the leading dot) for the directives that defines the stack, data and code segment are

. STACK [SIZE]
. DATA
..... Initialize Data Variables
. CODE

The Default Stack size is 1024 bytes.
To use them as above, Memory Model initialization should be carried out.

MEMORY MODEL DEFINTION

- The different models tell the assembler how to use segments to provide space and ensure optimum execution speed.
- The format of Memory Model Definition is
- . MODEL [MODEL NAME]
- The Memory Model may be TINY, SMALL, MEDIUM, COMPACT, LARGE AND HUGE.

MODEL TYPE	DESCRIPTION
TINY	All DATA, CODE & STACK Segment must fit in one Segment of Size <=64K.
SMALL	One Code Segment of Size <=64K. One Data Segment of Size <=64 K.
MEDIUM	One Data Segment of Size <=64K. Any Number of Code Segments.
COMPACT	One Code Segment of Size < =64K. Any Number of Data Segments.
LARGE	Any Number of Code and Data Segments.
HUGE	Any Number of Code and Data Segments.

#THE PROC DIRECTIVE

The Code Segment contains the executable code for a program, which consists of one or more procedures, defined initially with the PROC Directive and ended with the ENDP Directive.

Its Format is given as:

```
PROCEDURE NAME
..... .
.....
.....
PROCEDURE NAME    ENDP
```

#END DIRECTIVE

As already mentioned, the ENDP Directive indicates the end of a procedure. An END Directive ends the entire Program and appears as the last statement.

Its Format is

```
END [PROCEDURE NAME]
```

#PROCESSOR DIRECTIVE

Most Assemblers assume that the source program is to run on a basic 8086 level.
As a result, when you use instructions or features introduced by later processors, you have to notify the assemblers be means of a processor directive as . 286,. 386,. 486 or . 586
This directive may appear before the Code Segment.

#THE EQU DIRECTIVE

It is used for redefining symbolic names. EXAMPLE: DATAX DB 25 DATA EQU DATAX

#THE . STARTUP AND . EXIT DIRECTIVE

MASM 6. 0 introduced the . STARTUP and . EXIT Directive to simplify program initialization and termination.
. STARTUP generates the instruction to initialize the Segment Registers.
. EXIT generates the INT 21H function 4ch instruction for exiting the Program.

DEFINING TYPES OF DATA

The Format of Data Definition is given as [NAME] DN [EXPRESSION]

EXAMPLES

STRING DB 'HELLO WORLD' NUM1 DB 10
NUM2 DB 90

DEFINITION	DIRECTIVE
BYTE	DB
WORD	DW
DOUBLE WORD	DD
FAR WORD	DF
QUAD WORD	DQ
TEN BYTES	DT

Duplication of Constants in a Statement is also possible and is given by [NAME] DN [REPEAT-COUNT DUP (EXPRESSION)]

EXAMPLES

```
DATAX DB 5 DUP(12)          ; 5 Bytes containing hex 0c0c0c0c0c
DATA DB 10 DUP(?)           ; 10 Words Uninitialized
DATAZ DB 3 DUP(5 DUP(4))    ; 44444 44444 44444
```

CHARACTER STRINGS

Character Strings are used for descriptive data. Consequently DB is the conventional format for defining character data of any length

An Example is

```
DB 'Computer City' DB "Hello World"
DB "St. Xavier's College"
```

2. NUMERIC CONSTANTS

```
#BINARY      : VAL1 DB 10101010B
#DECIMAL     : VAL1 DB 230
#HEXADECIMAL : VAL1 DB 23H
```

8086 Microprocessor Instruction Sets

1. Data Transfer Instructions

1. 1 General Purpose Byte or Word Transfer Instructions

Instructions	Comments

MOV
MOV Destination, Source
MOV CX, 04H
	Copy byte or word from specified source to specified destination.

PUSH PUSH Source PUSH BX	Copy specified word to top of stack.
POP POP Destination POP AX	Copy word from top to stack to specified location.
XCHG XCHG Destination, Source XCHG AX, BX	Exchange word or byte.
XLAT	Translate a byte in AL using a table in memory. It first adds AL + BXt of or m memory address. It then copys the content into AL.

1.2 Simple Input and Output Port Transfer Instructions

INSTRUCTIONS	COMMENTS
IN IN AX, Port_Addr IN AX, 34H	Copy a byte or word from specified port to accumulator.
OUT OUT Port_Addr, AX OUT 2C H, AX	Copy a byte or word from accumulator to specified port.

1.3 Special Address Transfer Instructions

INSTRUCTIONS	COMMENTS
LEA LEA Register, Source LEA BX, PRICE	Load effective address of operand into specified register.

LDS
LDS Register, Source
LDS BX, [4326H]
	Load DS register and other specified register from memory.

LES	Load ES register and other specified register from memory.

1.4 Flag transfer instructions

INSTRUCTIONS	COMMENTS
LAHF	Copy to AH with the low byte of the flag register.
SAHF	Stores AH register to low byte of flag register.
PUSHF	Copy flag register to top of stack.
POPF	Copy word at top of stack to flag register.

2. Arithmetic instructions

2.1 Addition Instructions

INSTRUCTIONS	COMMENTS

ADD
ADD Destination, Source
ADD AL, 74H
	Add specified byte to byte or word to word.

ADC ADC Destination, Source ADC CL, BL	Add byte + byte + carry flag Add word + word + carry flag
INC INC Register INC CX	Increment specified byte or word by 1.
AAA	ASCII adjust after addition.
DAA	Decimal adjust after addition.

2.2 Subtraction Instructions

INSTRUCTIONS	COMMENTS

SUB
SUB Destination, Source
SUB CX, BX
	Subtract byte from byte or word from word.

SBB SBB Destination, Source SBB CH, AL	Subtract byte and carry flag from byte. Subtract word and carry flag from word.
DEC DEC Register DEC CX	Decrement specified byte or word by 1.
NEG NEG Register NEG AL	Form 2's complement.

CMP
CMP Destination, Source
CMP CX, BX
CF ZF SF
0 1 0 CX=BX
0 0 0 CX>BX
1 0 1 CX<BX
	Compare two specified bytes or words.

AAS	ASCII adjust after subtraction.
DAS	Decimal adjust after subtraction.

2.3 Multiplication Instructions

INSTRUCTIONS	COMMENTS
MUL MUL Source MUL CX	Multiply unsigned byte by byte or unsigned word by word. When a byte is multiplied by the content of AL, the result is kept into AX. When a word is multiplied by the content of AX, MS Byte in DX and LS Byte in AX.
IMUL IMUL Source IMUL CX	Multiply signed byte by byte or signed word by word.
AAM	ASCII adjust after multiplication. It converts packed BCD to unpacked BCD.

2.4 Division Instructions

INSTRUCTIONS	COMMENTS
DIV DIV Source DIV BL DIV CX	Divide unsigned word by byte Divide unsigned word double word by byte. When a word is divided by byte, the word must be in AX register and the divisor can be in a register or a memory location. After division AL(quotient) AH(remainder) When a double word is divided by word, the double word must be in DX : AX pair and the divisor can be in a register or a memory location. After division AX(quotient) DX (remainder)
AAD	ASCII adjust before division BCD to binary convert before division.
CBW	Fill upper byte of word with copies of sign bit of lower byte.
CWD	Fill upper word of double word with sign bit of lower word.

3. Bit Manipulation Instructions

3.1 Logical Instructions

INSTRUCTIONS	COMMENTS
NOT NOT Destination NOT BX	Invert each bit of a byte or word.
AND AND Destination, Source AND BH, CL	AND each bit in a byte/word with the corresponding bit in another byte or word.
OR OR Destination, Source OR AH, CL	OR each bit in a byte or word with the corresponding bit in another byte or word.
XOR XOR Destination, Source XOR CL, BH	XOR each bit in a byte or word with the corresponding bit in another byte or word.
TEST TEST Destination, Source TEST AL, BH	AND operands to update flags, but don't change the operands.

3.2 Shift Instructions

INSTRUCTIONS	COMMENTS
SHL/SAL SAL Destination, Count SHL Destination, Count CF←MSB←LSB←0	Shift Bits of Word or Byte Left, Put Zero(s) in LSB.
SHR SHR Destination, Count 0→MSB→LSB→CF	Shift Bits of Word or Byte Right, Put Zero(s) in MSB.
SAR SAR Destination, Count MSB → MSB → LSB → CF	Shift Bits of Word or Byte Right, Copy Old MSBin to New MSB.

3.3 Rotate Instructions

INSTRUCTIONS	COMMENTS
ROL	Rotate Bits of Byte or Word Left, MSB to LS and to CF.
ROR	Rotate Bits of Byte or Word Right, LSB to MSB and to CF.
RCL	Rotate Bits of Byte or Word Left, MSB to CF and CF to LSB.
RCR	Rotate Bits of Byte or Word Right, LSBTOCF and CFTOMSB.

4. Program Execution Transfer Instructions

4.1 Unconditional Transfer Instruction

INSTRUCTIONS	COMMENTS
CALL	Call a Sub program/Procedure.
RET	Return From Procedure to Calling Program.
JMP	Go to Specified Address to Get Next Instruction (Unconditional Jump to Specified Destination).

4.2 Conditional Transfer Instruction

INSTRUCTIONS	COMMENTS
JA/JNBE	Jump if Above/Jump if Not Below or Equal.
JAE/JNB	Jump if Above or Equal/Jump if Not Below.
JB/JNAE	Jump if Below/Jump if Not Above or Equal.
JBE/JNA	Jump if Below or Equal/Jump if Not Above.
JC	Jump if Carry Flag CF = 1.
JE/JZ	Jump if Equal/Jump if Zero Flag (ZF=1).
JG/JNLE	Jump if Greater/Jump if Not Less than or Equal.
JGE/JNL	Jump if Greater than or Equal/Jump if Not Less than.
JL/JNGE	Jump if Less than/Jump if Not Greater than or Equal.
JLE/JNG	Jump if Less than or Equal/Jump if Not Greater than.
JNC	Jump if No Carry i. e. CF = 0
JNE/JNZ	Jump if Not Equal/Jump if Not Zero (ZF=0)
JNO	Jump if No Overflow.
JNP/JPO	Jump if Not Parity/Jump if Parity Odd.
JNS	Jump if Not Sign (SF=0)
JP/JPE	Jump if Parity/Jump if Parity Even(PF=1)
JS	Jump if Sign (SF=1)

4.3 Iteration Control Instructions

INSTRUCTIONS	COMMENTS
LOOP	Loop Through a Sequence of Instructions Until CX = 0.
LOOPE/LOOPZ	Loop Through a Sequence of Instructions While ZF = 1 and CX! = 0.
LOOPNE/LOOPNZ	Loop Through a Sequence of Instruction While ZF = 0 & CX! = 0.
JCXZ	Jump to Specified Address if CX = 0.

4.4 Interrupt Instructions

INSTRUCTIONS	COMMENTS
INT	
INT0	Interrupt Program Execution if OF=1
IRET	Return From Interrupt Service Procedure to Main Program.

5. Process or Control Instructions

5.1 Flag Set/Clear Instruction

INSTRUCTIONS	COMMENTS
STC	Set Carry Flag CF to 1.
CLC	Clear Carry Flag to 0.
CMC	Complement the State of CF.
STD	Set Direction Flag to 1.
CLD	Clear Direction Flag to 0.
STI	Set Interrupt Flag to 1. (Enable INTR Input).
CLI	Clear Interrupt Enable to 0

5.2 No Operation Instruction

INSTRUCTIONS	COMMENTS
NOP	No Action Except Fetch and Decode.

5.3 External Hardware Synchronization INST

INSTRUCTIONS	COMMENTS
HLT	Halt (Do Nothing) Until Interrupt or Reset.
WAIT	Wait Until Signal On the TESTP in is Low.
ESC	Escape to External Coprocessor Such as 8087 or 8089.
LOCK	Prevents Another Processor From Taking the Bus While the Adjacent Instruction Executes.

6. String Instructions

INSTRUCTIONS	COMMENTS
REP	Repeat Instruction Until CX = 0.
REPE/REPZ	Repeat if Equal/Repeat if Zero
REPNE/REPNZ	Repeat if Not Equal/Repeat if Not Zero.
MOVS/MOVSB/MOVSW	Move Byte or Word From One String to Another.
COMPS/COMPSB/COMPSW	Compare Two String Bytes or Two String Words.
SCAS/SCASB/SCASW	Compares a Byte in AL or Word in AX With a Byte or Word Pointed By DI in ES.

DOS Function Codes

DOS INT 21h

AH	Description	AH	Description
01	Read character from STDIN	02	Write character to STDOUT
05	Write character to printer	06	Console Input/Output
07	Direct char read (STDIN), no echo	08	Char read from STDIN, no echo
09	Write string to STDOUT	0A	Buffered input
0B	Get STDIN status	0C	Flush buffer for STDIN
0D	Disk reset	0E	Select default drive
19	Get current default drive	25	Set interrupt vector
2A	Get system date	2B	Set system date
2C	Get system time	2D	Set system time
2E	Set verify flag	30	Get DOS version
35	Get Interrupt vector		

36	Get free disk space	39	Create subdirectory
3A	Remove subdirectory	3B	Set working directory
3C	Create file	3D	Open file
3E	Close file	3F	Read file
40	Write file	41	Delete file
42	Seek file	43	Get/Set file attributes
47	Get current directory	4C	Exit program
4D	Get return code	54	Get verify flag
56	Rename file	57	Get/Set file date

AH = 01h - READ CHARACTER FROM STANDARD INPUT, WITH ECHO

Return: AL = character read

Notes:

- ^C/^Break are checked
- ^P toggles the DOS-internal echo-to-printer flag
- ^Z is not interpreted, thus not causing an EOF if input is redirected character is echoed to standard output

See Also: AH = 06h, AH = 07h, AH = 08h, AH = 0Ah

AH = 02h -WRITE CHARACTER TO STANDARD OUTPUT

Entry: DL = character to write

Return: AL = last character output

Notes:

- ^C/^Break are checked
- the last character output will be the character in DL unless DL=09h on entry, in which case AL=20h as tabs are expanded to blanks
- if standard output is redirected to a file, no error checks (write- protected, full media, etc.) are performed

See Also: AH = 06h, AH = 09h

AH = 05h - WRITE CHARACTER TO PRINTER

Entry: DL = character to print

Notes:

- keyboard checked for ^C/^Break
- STDPRN is usually the first parallel port, but may be redirected under DOS 2+
- if the printer is busy, this function will wait

See Also: INT 17/AH = 00h

AH = 06h - DIRECT CONSOLE OUTPUT

Entry: DL = character (except FFh)

Return: AL = character output

Notes: does not check ^C/^Break

See Also: AH = 02h, AH = 09h

AH = 06h - DIRECT CONSOLE INPUT

Entry: AH = 06h DL = FFh

Return:

- ZF *set* if no character available and AL = 00h
- ZF *clear* if character available AL = character read

Notes:

- ^C/^Break are NOT checked
- if the returned character is 00h, the user pressed a key with an extended keycode, which will be returned by the next call of this function
- although the return of AL=00h when no characters are available is not documented, some programs rely on this behavior

See Also: AH = 0Bh

AH = 07h - DIRECT CHARACTER INPUT, WITHOUT ECHO

Return: AL = character read from standard input

Notes: does not check ^C/^Break

See Also: AH = 01h, AH = 06h, AH = 08h, AH = 0Ah

AH = 08h - CHARACTER INPUT WITHOUT ECHO

Return: AL = character read from standard input

Notes: ^C/^Break are checked

See Also: AH = 01h, AH = 06h, AH = 07h, AH = 0Ah, AH = 64h

AH = 09h - WRITE STRING TO STANDARD OUTPUT

Entry: DS:DX -> '$'-terminated string

Return: AL = 24h

Notes: ^C/^Break are checked

See Also: AH = 02h, AH = 06h"OUTPUT"

AH = 0Ah - BUFFERED INPUT

Entry: DS:DX -> buffer (see below)

Return: buffer filled with user input

Notes:

- ^C/^Break are checked
- reads from standard input

See Also: AH = 0Ch

Format of DOS input buffer:

Offset	Size	Description
00	1	Maximum characters buffer can hold
01	1	Number of chars from last input which may be recalled OR number of characters actually read, excluding CR
02	n	Actual characters read, including the final carriage return

AH=0Bh - GET STDIN STATUS

Return:

- AL = 00h if no character available
- AL = FFh if character is available

Notes: ^C/^Break are checked

See Also: AH = 06h"INPUT"

AH = 0Ch - FLUSH BUFFER AND READ STANDARD INPUT

Entry:

- AL = STDIN input function to execute after flushing buffer
- other registers as appropriate for the input function

Return: as appropriate for the specified input function

Note: if AL is not one of 01h,06h,07h,08h, or 0Ah, the buffer is flushed but no input is attempted
See Also: AH = 01h, AH = 06h"INPUT", AH = 07h, AH = 08h, AH = 0Ah

AH = 0Dh - DISK RESET

Notes: This function writes all modified disk buffers to disk, but does not update the directory information
See Also: AX = 5D01h

AH = 0Eh - SELECT DEFAULT DRIVE

Entry: DL = new default drive (0=A:, 1=B:, etc)
Return: AL = number of potentially valid drive letters
Notes: the return value is the highest drive present
See Also: AH = 19h, AH = 3Bh, AH = DBh

AH = 19h - GET CURRENT DEFAULT DRIVE

Return: AL = drive (0=A:, 1=B:, etc)
See Also: AH = 0Eh, AH = 47h, AH = BBh

AH = 25h - SET INTERRUPT VECTOR

Entry:

- AL = interrupt number
- DS:DX -> new interrupt handler

Notes: this function is preferred over direct modification of the interrupt vector table
See Also: AX = 2501h, AH = 35h

AH = 2Ah - GET SYSTEM DATE

Return: CX = year (1980-2099) DH = month DL = day AL = day of week (00h=Sunday)
See Also: AH = 2Bh"DOS", AH = 2Ch, AH = E7h

AH = 2Bh - SET SYSTEM DATE

Entry: CX = year (1980-2099) DH = month DL = day
Return:

- AL = 00 successful
- FFh invalid date, system date unchanged

Note: DOS 3.3+ also sets CMOS clock
See Also: AH = 2Ah, AH = 2Dh

AH = 2Ch - GET SYSTEM TIME

Return: CH = hour CL = minute DH = second DL = 1/100 seconds
Note: on most systems, the resolution of the system clock is about 5/100sec, so returned times generally do not increment by 1 on some systems, DL may always return 00h
See Also: AH = 2Ah, AH = 2Dh, AH = E7h

AH = 2Dh - SET SYSTEM TIME

Entry: CH = hour CL = minute DH = second DL = 1/100 seconds
Return:

- AL = 00h successful
- FFh if invalid time, system time unchanged

Note: DOS 3.3+ also sets CMOS clock
See Also: AH = 2Bh"DOS", AH = 2Ch

AH = 2Eh - SET VERIFY FLAG

Entry: AL = new state of verify flag (00 off, 01h o)
Notes:

- default state at system boot is OFF
- when ON, all disk writes are verified provided the device driver supports read-after-write verification

See Also: AH = 54h

AH=30h - GET DOS VERSION

Entry: AL = what to return in BH (00h OEM number, 01h version flag)

Return:

- AL = major version number (00h if DOS 1.x)
- AH = minor version number
- BL:CX = 24-bit user serial number (most versions do not use this) if DOS <5 or AL=00h
- BH = MS-DOS OEM number if DOS 5+ and AL=01h
- BH = version flag bit 3: DOS is in ROM other: reserved (0)

Notes:

- DOS 4.01 and 4.02 identify themselves as version 4.00
- MS-DOS 6.21 reports its version as 6.20; version 6.22 returns the correct value
- Windows95 returns version 7.00 (the underlying MS-DOS)

See Also: AX=3000h/ BX = 3000h, AX = 3306h, AX = 4452h

AH=35h - GET INTERRUPT VECTOR

Entry: AL = interrupt number

Return: ES:BX -> current interrupt handler

See Also: AH = 25h, AX = 2503h

AH = 36h - GET FREE DISK SPACE

Entry: DL = drive number (0=default, 1=A:, etc)

Return:

- AX = FFFFh if invalid drive
- AX = sectors per cluster BX = number of free clusters CX = bytes per sector DX = total clusters on drive

Notes:

- free space on drive in bytes is AX * BX * CX
- total space on drive in bytes is AX * CX * DX
- "lost clusters" are considered to be in use
- this function does not return proper results on CD-ROMs; use AX=4402h"CD-ROM" instead

See Also: AH = 1Bh, AH = 1Ch, AX = 4402h"CD-ROM"

AH = 39h - "MKDIR" - CREATE SUBDIRECTORY

Entry: DS:DX -> ASCIZ pathname

Return:

- CF clear if successful AX destroyed
- CF set on error AX = error code (03h,05h)

Notes:

- all directories in the given path except the last must exist
- fails if the parent directory is the root and is full
- DOS 2.x-3.3 allow the creation of a directory sufficiently deep that it is not possible to make that directory the current directory because the path would exceed 64 characters

See Also: AH = 3Ah,AH = 3Bh, AH = 6Dh

AH = 3Ah - "RMDIR" - REMOVE SUBDIRECTORY

Entry: DS:DX -> ASCIZ pathname of directory to be removed

Return:

- CF clear if successful, AX destroyed
- CF set on error AX = error code (03h,05h,06h,10h)

Notes: directory must be empty (contain only '.' and '..' entries)

See Also: AH = 39h, AH = 3Bh

AH = 3Bh - "CHDIR" - SET CURRENT DIRECTORY
Entry: DS:DX -> ASCIZ pathname to become current directory (max 64 bytes)
Return:

- CF clear if successful, AX destroyed
- CF set on error AX = error code (03h)

Notes: if new directory name includes a drive letter, the default drive is not changed, only the current directory on that drive
See Also: AH = 47h, AH = 71h, INT 2F/AX = 1105h
AH = 3Ch - "CREAT" - CREATE OR TRUNCATE FILE
Entry:

- CX = file attributes
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful, AX = file handle
- CF set on error AX = error code (03h,04h,05h)

Notes: if a file with the given name exists, it is truncated to zero length
See Also: AH = 16h, AH = 3Dh, AH = 5Ah, AH = 5Bh
AH = 3Dh - "OPEN" - OPEN EXISTING FILE
Entry:

- AL = access and sharing modes
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful, AX = file handle
- CF set on error AX = error code (01h,02h,03h,04h,05h,0Ch,56h)

Notes:

- file pointer is set to start of file
- file handles which are inherited from a parent also inherit sharing and access restrictions
- files may be opened even if given the hidden or system attributes

See Also: AH = 0Fh, AH = 3Ch, AX = 4301h, AX = 5D00h
AH = 3Eh - "CLOSE" - CLOSE FILE
Entry: BX = file handle
Return:

- CF clear if successful, AX destroyed
- CF set on error, AX = error code (06h)

Note: if the file was written to, any pending disk writes are performed, the time and date stamps are set to the current time, and the directory entry is updated
See Also: AH = 10h, AH = 3Ch, AH = 3Dh
AH = 3Fh - "READ" - READ FROM FILE OR DEVICE
Entry:

- BX = file handle
- CX = number of bytes to read
- DS:DX -> buffer for data

Return:

- CF clear if successful - AX = number of bytes actually read (0 if at EOF before call)
- CF set on error AX = error code (05h,06h)

Notes:

- data is read beginning at current file position, and the file position is updated after a successful read
- the returned AX may be smaller than the request in CX if a partial read occurred
- if reading from CON, read stops at first CR

See Also: AH = 27h, AH = 40h, AH = 93h

AH=40h - "WRITE" - WRITE TO FILE OR DEVICE

Entry:

- BX = file handle
- CX = number of bytes to write
- DS:DX -> data to write

Return:

- CF clear if successful -AX = number of bytes actually written
- CF set on error - AX = error code (05h,06h)

Notes:

- if CX is zero, no data is written, and the file is truncated or extended to the current position
- data is written beginning at the current file position, and the file position is updated after a successful write
- the usual cause for AX < CX on return is a full disk

See Also: AH = 28h, AH = 3Fh

AH = 41H - "UNLINK" - DELETE FILE

Entry:

- DS:DX -> ASCIZ filename (no wildcards, but see notes)
- CL = attribute mask for deletion (server call only, see notes)

Return:

- CF clear if successful, AX destroyed (DOS 3.3) AL seems to be drive of deleted file
- CF set on error AX = error code (02h,03h,05h)

Notes:

- (DOS 3.1+) wildcards are allowed if invoked via AX=5D00h, in which case the filespec must be canonical (as returned by AH=60h), and only files matching the attribute mask in CL are deleted
- DOS does not erase the file's data; it merely becomes inaccessible because the FAT chain for the file is cleared
- deleting a file which is currently open may lead to file system corruption.

See Also: AH = 13h, AX = 4301h, AX = 4380h, AX = 5D00h, AH=60h, AH = 71h

AH=42h - "LSEEK" - SET CURRENT FILE POSITION

Entry:

- AL = origin of move 00h start of file 01h current file position 02h end of file
- BX = file handle
- CX:DX = offset from origin of new file position

Return:

- CF clear if successful, DX:AX = new file position in bytes from start of file
- CF set on error, AX = error code (01h,06h)

Notes:

- for origins 01h and 02h, the pointer may be positioned before the start of the file; no error is returned in that case, but subsequent attempts at I/O will produce errors
- if the new position is beyond the current end of file, the file will be extended by the next write (see AH=40h)

See Also: AH = 24h

AH=43 - GET FILE ATTRIBUTES

Entry:

- AL = 00h
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful CX = file attributes
- CF set on error, AX = error code (01h,02h,03h,05h)

BUG: Windows for Workgroups returns error code 05h (access denied) instead of error code 02h (file not found) when attempting to get the attributes of a nonexistent file.

See Also: AX= 4301h, AX = 4310h, AX = 7143h, AH = B6h

AH=43 - "CHMOD" - SET FILE ATTRIBUTES

Entry:

- AL = 01h
- CX = new file attributes
- DS:DX -> ASCIZ filename

Return:

- CF clear if successful, AX destroyed
- CF set on error, AX = error code (01h,02h,03h,05h)

Notes:

- will not change volume label or directory attribute bits, but will change the other attribute bits of a directory
- MS-DOS 4.01 reportedly closes the file if it is currently open

See Also: AX = 4300h, AX = 4311h, AX = 7143h, INT 2F/AX = 110Eh

Bitfields for file attributes:

Bits	7	6	5	4	3	2	1	0
Description	shareable	-	archive	directory	vol. label	system	hidden	read-only

AH = 47h - "CWD" - GET CURRENT DIRECTORY

Entry:

- DL = drive number (00h = default, 01h = A:, etc)
- DS:SI -> 64-byte buffer for ASCIZ pathname

Return:

- CF clear if successful
- CF set on error, AX = error code (0Fh)

Notes:

- the returned path does not include a drive or the initial backslash
- many Microsoft products for Windows rely on AX being 0100h on success

See Also: AH = 19h, AH = 3Bh, AH = 71h

AH = 4Ch - "EXIT" - TERMINATE WITH RETURN CODE

Entry: AL = return code

Return: never returns

Notes: unless the process is its own parent, all open files are closed and all memory belonging to the process is freed

See Also: AH = 00h, AH = 26h, AH = 4Bh, AH = 4Dh

AH = 4Dh - GET RETURN CODE (ERRORLEVEL)

Return:

- AH = termination type (00=normal, 01h control-C abort, 02h=critical error abort, 03h terminate and stay resident)
- AL = return code

Notes:

- the word in which DOS stores the return code is cleared after being read by this function, so the return code can only be retrieved once
- COMMAND.COM stores the return code of the last external command it executed as ERRORLEVEL

See Also: AH = 4Bh, AH = 4Ch, AH = 8Ah

AH = 54h - GET VERIFY FLAG

Return: AL = verify flag (00h=off, 01h=on, i.e. all disk writes verified after writing)

See Also: AH = 2Eh

AH = 56h - "RENAME" - RENAME FILE

Entry:

- DS:DX -> ASCIZ filename of existing file (no wildcards, but see below)
- ES:DI -> ASCIZ new filename (no wildcards)
- CL = attribute mask (server call only, see below)

Return:

- CF clear if successful
- CF set on error, AX= error code (02h,03h,05h,11h)

Notes:

- allows move between directories on same logical volume
- this function does not set the archive attribute
- open files should not be renamed
- (DOS 3.0+) allows renaming of directories

AH = 57h - GET FILE'S LAST-WRITTEN DATE AND TIME

Entry:

- AL = 00h (Get attribute)
- BX = file handle

Return:

- CF clear if successful, CX = file's time DX = file's date
- CF set on error, AX = error code (01h,06h)

See Also: AX = 5701h

Bitfields for file time:

Bits	15-11	10-5	4-0
Description	hours	minutes	seconds

Bitfields for file date:

Bits	15-9	8-5	4-0
Description	year (1980-)	month	day

AH = 57h - SET FILE'S LAST-WRITTEN DATE AND TIME

Entry:

- AL =01h (Set attributes)
- BX = file handle
- CX = new time
- DX = new date

Return:

- CF clear if successful
- CF set on error AX = error code (01h,06h)

See Also: AX = 5700h

This page is maintained by **Barry Wilks.**

INT 10h, INT 10H or INT 16 is shorthand for BIOS interrupt call 10hex, the 17th interrupt vector in an x86-based computer system. The BIOS typically sets up a real mode interrupt handler at this vector that provides video services. Such services include setting the video mode, character and string output, and graphics primitives (reading and writing pixels in graphics mode).

To use this call, load AH with the number of the desired sub-function, load other required parameters in other registers, and make the call. INT 10h is fairly slow, so many programs bypass this BIOS routine and access the display hardware directly. Setting the video mode, which is done infrequently, can be accomplished by using the BIOS, while drawing graphics on the screen in a game needs to be done quickly, so direct access to video RAM is more appropriate than making a BIOS call for every pixel.

List of supported functions

Function	Function code	Parameters	Return
Set video mode	AH=00h	AL = video mode	AL = video mode flag / CRT controller mode byte
Set text-mode cursor shape	AH=01h	CH = Scan Row Start, CL = Scan Row End Normally a character cell has 8 scan lines, 0-7. So, CX=0607h is a normal underline cursor, CX=0007h is a full-block cursor. If bit 5 of CH is set, that often means "Hide cursor". So CX=2607h is an invisible cursor. Some video cards have 16 scan lines, 00h-0Fh. Some video cards don't use bit 5 of CH. With these, make Start>End (e.g. CX=0706h)	
Set cursor position	AH=02h	BH = Page Number, DH = Row, DL = Column	
Get cursor position and shape	AH=03h	BH = Page Number	AX = 0, CH = Start scan line, CL = End scan line, DH = Row, DL = Column
Read light pen position (Does not work on VGA systems)	AH=04h		AH = Status (0=not triggered, 1=triggered), BX = Pixel X, CH = Pixel Y, CX = Pixel line number for modes 0Fh-10h, DH = Character Y, DL = Character X
Select active display page	AH=05h	AL = Page Number	
Scroll up window	AH=06h	AL = lines to scroll (0 = clear, CH, CL, DH, DL are used), BH = Background Color and Foreground color. BH = 43h, means that background color is red and foreground color is cyan. Refer the BIOS color attributes CH = Upper row number, CL = Left column number, DH = Lower row number, DL = Right column number	
Scroll down window	AH=07h	like above	

Read character and attribute at cursor position	AH=08h	BH = Page Number	AH = Color, AL = Character
Write character and attribute at cursor position	AH=09h	AL = Character, BH = Page Number, BL = Color, CX = Number of times to print character	
Write character only at cursor position	AH=0Ah	AL = Character, BH = Page Number, CX = Number of times to print character	
Set background/border color	AH=0Bh, BH = 00h	BL = Background/Border color (border only in text modes)	
Set palette	AH=0Bh, BH = 01h	BL = Palette ID (was only valid in CGA, but newer cards support it in many or all graphics modes)	
Write graphics pixel	AH=0Ch	AL = Color, BH = Page Number, CX = x, DX = y	
Read graphics pixel	AH=0Dh	BH = Page Number, CX = x, DX = y	AL = Color
Teletype output	AH=0Eh	AL = Character, BH = Page Number, BL = Color (only in graphic mode)	
Get current video mode	AH=0Fh		AL = Video Mode, AH = number of character columns, BH = active page
Write string (EGA+, meaning PC AT minimum)	AH=13h	AL = Write mode, BH = Page Number, BL = Color, CX = String length, DH = Row, DL = Column, ES:BP = Offset of string	

ALP Samples Using DOS and Video BIOS Functions

```
TITLE "Program to Print Hello World in ALP"
;-----------------------------------------------------
.MODEL SMALL
.STACK
.DATA
        STRING DB 'HELLO WORLD $'
.CODE

;-----------------------------------------------------
MAIN PROC
      MOV AX,@DATA
      MOV DS,AX                 ; Initialize the DATA Segment

      MOV DX,OFFSET STRING      ; Load the Offset Address into DX
      MOV AH,09H                ; AH=09H For String Display until $
      INT 21H                   ; DOS Interrupt Function
      MOV AX,4C00H              ; End Request with AH=4CH or AX=4C00H
      INT 21H
      MAIN ENDP                 ; End Procedure
END MAIN                        ; End Program
;-----------------------------------------------------
```

TITLE "Program to Print the Sum of Two 8 Bit Numbers"

```
;-----------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
      VAL1 DB 89
      VAL2 DB 10
      MSG DB 'SUM OF 2 NUMBERS: $'
. CODE

;-----------------------------------------------------------

MAIN PROC
   MOV AX,@DATA
   MOV DS,AX

   MOV AX,0
   MOV AL,VAL1
   ADD AL,VAL2

   AAM       ;AAM Converts Binary Value to Unpacked BCD.

   ADD AX,3030H ;Ax is Added with 3030H to Obtain ASCII Value

   PUSH AX

   ;;;;;;;;;;;;;;DISPLAY MESSAGE
   LEA DX,MSG
   MOV AH,09H
   INT 21H
   ;;;;;;;;;;;;;;END DISPLAY MESSAGE

   POP AX

   MOV DL,AH
   MOV DH,AL
   MOV AH,02H
   INT 21H

   MOV DL,DH
   MOV AH,02H
   INT 21H

   MOV AX,4C00H
   INT 21H
MAIN ENDP
END MAIN
;-----------------------------------------------------------
```

TITLE "16 Bit Binary Content of Ax is Converted to 4 Digit ASCII"

```
;-----------------------------------------------------
.MODEL SMALL
.STACK
.DATA
   VAL DW 256
.CODE
;-----------------------------------------------------

MAIN PROC
   MOV AX,@DATA
   MOV DS,AX          ;DATA SEGMENT INITIALIZATION

   MOV AX,VAL
   XOR DX,DX  ;DX IS CLEARED
   MOV CX,100 ;DIVISOR IS PASSED INTO CX REGISTER

   DIV CX             ;DX:AX PAIR DIVIDED BY CX
                      ;QUOTIENT  IN AX
                      ;REMAINDER IN DX

AAM          ;QUOTIENT IS ADJUSTED TO UNPACKED BCD
  ADD AX,3030H  ;QUOTIENT IS CONVERTED TO ASCII
  XCHG AX,DX         ;DX AND AX ARE SWAPPED
  AAM                ;REMAINDER IS ADJUSTED TO UNPACKED BCD
  ADD AX,3030H  ;REMAINDER IS CONVERTED TO ASCII

      ;;;;;;;;;;;;;;;;;;;;DISPLAY OPERATION

      ;;DISPLAY QUOTIENT
      ;;DISPLAY REMAINDER


      ;;;;;;;;;;;;;;;;;;;;END DISPLAY


      MOV AX,4C00H ;END REQUEST
      INT 21H      ;DOS INTERRUPT FUNCTION

MAIN ENDP
END MAIN

;-----------------------------------------------------
```

TITLE "Program to Print the Difference of Two 8 Bit Numbers"

```
;-----------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
      VAL1 DB 89
      VAL2 DB 10
      MSG DB 'DIFFERENCE OF 2 NUMBERS: $'
. CODE

;-----------------------------------------------------------

MAIN PROC
   MOV AX,@DATA
   MOV DS,AX

   MOV AX,0
   MOV AL,VAL1
   SUB AL,VAL2

   ;;;;; AAM Converts Binary Value to Unpacked BCD.
   ;;;;; AAM Only Works with AX Register
   AAM

   ;;;;; AX is Added with 3030H to Obtain ASCII Value.
   ADD AX,3030H

   PUSH AX

   ;;;;; DISPLAY MESSAGE With AH=09H
   LEA DX,MSG
   MOV AH,09H
   INT 21H
   ;;;;; END DISPLAY MESSAGE

   POP AX

   MOV DL,AH
   MOV DH,AL
   MOV AH,02H
   INT 21H

   MOV DL,DH
   MOV AH,02H
   INT 21H

   MOV AX,4C00H
   INT 21H
MAIN ENDP
END MAIN

;-----------------------------------------------------------
```

```
TITLE "Program to Print the Sum of Two 16 Bit Numbers"
;-----------------------------------------------------
.MODEL SMALL
.STACK
.DATA
   VAL1 DW 2010
   VAL2 DW 2050
.CODE
;-----------------------------------------------------
MAIN PROC
      MOV AX,@DATA
      MOV DS,AX   ;INITIALIZE THE DATA SEGMENT

      MOV AX,VAL1
      ADD AX,VAL2 ;AX Holds 16 bit Value
      ;; 16 Bit Answer Splitting Strategy.
      ;; 16 Bit Division is Carried out
      ;; 32 Bit Divident (DX:AX) and 16 Bit Divisor is Required.

      XOR DX,DX   ;Register DX is Cleared
      MOV CX,100
      DIV CX      ;DX:Ax Divided by CX.
       ;Remainder in DX and Quotient in AX

      AAM         ;Quotient is Converted to Unpacked BCD
      ADD AX,3030H ;Ready  For Display i. e Converted to ASCII
      MOV BX,AX   ;Store the Quotient to BX

      XCHG AX,DX  ;Exchange the Contents of AX and DX
      AAM         ;Remainder is Converted to Unpacked BCD
      ADD AX,3030H ;Remainder Converted to ASCII
      PUSH AX     ;Remainder Pushed to Stack
      ;;;;;;;;;;;;;;Quotient Ready in BX and Remainder Ready in AX
      ;;;;;;;;;;;;;;Display Quotient First and then the Remainder

      ;;;;;;;;;;;;;;Display Operation Started
      MOV DL,BH
      MOV AH,02H
      INT 21H
      MOV DL,BL
      MOV AH,02H
      INT 21H
      POP AX
      MOV DL,AH
      MOV DH,AL
      MOV AH,02H
      INT 21H
      MOV DL,DH
      MOV AH,02H
      INT 21H
      ;;;;;;;;;;;;;; Display Operation End
```

```
MOV AX,4C00H ; End Request AH=4CH
        INT 21H          ; Dos Interrupt Function
MAIN ENDP
END MAIN
;-------------------------------------------------------------
```

TITLE "Program to Display Numbers From 0 To 9. [0 1 2 3 4 5 6 7 8 9]"

```
;-------------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
     VAL DB '0'
. CODE
;-------------------------------------------------------------
SPACE MACRO
     MOV DL,' '
     MOV AH,02H
     INT 21H
ENDM
;-------------------------------------------------------------

MAIN PROC
     MOV AX,@DATA
     MOV DS,AX

     MOV CX,26
     MOV DL,VAL
     TOP:
          MOV AH,02H
          INT 21H ;DISPLAY THE NUMBER
          INC DL  ;DL IS INCREMENTED BY 1
          PUSH DX ;PUSH THE CONTENT OF DX TO STACK

                    ;;;;;;;;; INVOKED SPACE MACRO
SPACE
          ;;;;;;;;; END OF SPACE MACRO

POP DX  ;POP THE CONTENT FROM STACK TO DX
          DEC CX  ;DECREMENT THE CONTENT OF COUNT BY 1.
          JZ LAST ;JUMP TO LABEL LAST IF CX=0
          JMP TOP ;UNCONDITIONAL JUMP TO LABEL TOP

     LAST:
          MOV AH,4CH
          INT 21H
MAIN ENDP
END MAIN

;-------------------------------------------------------------
```

TITLE "Program to Display Numbers From 0 To 9 With Line Feed"

```
;-----------------------------------------------------
.MODEL SMALL
.STACK
.DATA
   VAL DB 0
.CODE
;-----------------------------------------------------
LFEED MACRO
   MOV AH,06H
   MOV DL,0AH
   INT 21H
   MOV DL,0DH
   INT 21H
ENDM
;-----------------------------------------------------

MAIN PROC
   MOV AX,@DATA
   MOV DS,AX

   MOV CX,10

   MOV AH,00H
   MOV AL,VAL
   TOP:
      MOV BL,AL
      AAM
      ADD AX,3030H
      MOV DL,AL
      MOV AH,02H
      INT 21H

                  ;;;;;;;; INVOKED LFEED MACRO FOR LINE FEED
      LFEED
                  ;;;;;;;; END OF LFEED MACRO

      INC BL
      MOV AL,BL
      DEC CX
      JZ LAST
      JMP TOP
   LAST:
      MOV AH,4CH
      INT 21H
MAIN ENDP
END MAIN

;-----------------------------------------------------
```

```
TITLE "Program to Display Alphabets From A To Z. [A B C D E F ..... Z]"

;-----------------------------------------------------
. MODEL SMALL
. STACK
. DATA
     VAL DB 'A'
. CODE
;-----------------------------------------------------

SPACE MACRO
     MOV DL,' '
     MOV AH,02H
     INT 21H
ENDM

;-----------------------------------------------------

MAIN PROC
     MOV AX,@DATA
     MOV DS,AX

     MOV CX,26
     MOV DL,VAL
     TOP:
          MOV AH,02H
          INT 21H
          INC DL
          PUSH DX
                         ;;;;;;;;; INVOKED SPACE MACRO
          SPACE
                         ;;;;;;;;; END OF SPACE MACRO

          POP DX
          DEC CX
          JZ LAST
          JMP TOP
     LAST:
          MOV AH,4CH
          INT 21H
MAIN ENDP
END MAIN

;-----------------------------------------------------
```

TITLE "Program to Print the Sum of Natural Nos From 1 To 10. [1+2+3.... +10]"

```
;-------------------------------------------------
.MODEL SMALL
.STACK
.DATA
   VAL DB 1
.CODE
;-------------------------------------------------

MAIN PROC
   MOV AX,@DATA
   MOV DS,AX

   MOV CX,10
   MOV AL,VAL
   MOV DL,0

   TOP:
      ADD DL,AL ;DL=DL+AL
      INC AL   ;INCREMENT THE CONTENT OF AL
      DEC CX   ;DECREMENT THE CONTENT OF CX
      JZ LAST  ;JUMP TO LAST IF CX=0
      JMP TOP  ;UNCONDITIONAL JUMP TO LABEL TOP


   LAST:

MOV AL,DL  ;FINAL SUM IN DL IS PASSED TO AL
           ;AAM ALWAYS WORKS WITH AX REGISTER

;;;;;;;;;;;;DISPLAY OPERATION STARTED
      AAM
      ADD AX,3030H
      MOV DL,AH
      MOV DH,AL
      MOV AH,02H
      INT 21H
      MOV DL,DH
      MOV AH,02H
      INT 21H

                  ;;;;;;;;;;;;END DISPLAY

      MOV AH,4CH
      INT 21H
MAIN ENDP
END MAIN
;-------------------------------------------------
```

```
TITLE "Program to Demonstrate Multiplication Table of a Given Number"
;-----------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
    VAL DB 7

. CODE

;-----------------------------------------------------------
GODOWN MACRO
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H
ENDM
;-----------------------------------------------------------
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX ; Initialize Data Segment

        MOV AX,0  ; Make AX as 0
        MOV CX,10 ;Initialize the Counter
        MOV BL,1
        TOP:
                MOV AL,BL
                MUL VAL ; Multiply the Content of Val and AL : Answer in AL
                AAM
                ADD AX,3030H
                MOV DL,AH
                MOV DH,AL
                MOV AH,02H
                INT 21H
                MOV DL,DH
                MOV AH,02H
                INT 21H
                GODOWN ; Invoked GoDOWN Macro

                MOV AX,0
                INC BL
                DEC CX
                JZ LAST
                JMP TOP
        LAST:
        MOV AX,4C00H
        INT 21H
MAIN ENDP
END MAIN
;-----------------------------------------------------------
```

TITLE "Program to Calculate Factorial of Given Number"

```
;-----------------------------------------------
.MODEL SMALL
.STACK
.DATA
    VAL DW 7
.CODE
;-----------------------------------------------
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX  ; Initialize Data Segment
        MOV AX,VAL ; AL is Passed with the Content of DI
        MOV CX,AX  ; Counter to  no given in val
        DEC CX     ; Decrement the Content of CL to make n-1
        TOP:
                MUL CX     ; Multiply the content of AL and CL answer in AL
                DEC CX     ; Decrement the content of cl
                JZ LAST    ; Jump if Zero to Last
                JMP TOP    ; JMP to Top Unconditionally
        LAST:
                MOV CX,100
                DIV CX     ; Quotient in AX and Remainder in DX
                AAM        ; Adjust Quotient to Unpacked BCD
                ADD AX,3030H
                MOV BX,AX  ; Prserve the Content of AX to BX
                XCHG AX,DX
                AAM
                ADD AX,3030H
                PUSH AX    ; Preserve the Content of AX into Stack

                ;;;;;;;;;;;;; Display Operation Started
MOV DL,BH
                MOV AH,02H
                INT 21H
                MOV DL,BL
                MOV AH,02H
                INT 21H    ; Quotient Printing is Finished
                POP AX
                MOV DL,AH
                MOV DH,AL
                MOV AH,02H
                INT 21H
                MOV DL,DH
                MOV AH,02H
                INT 21H    ; Remainder Printing is Finished
                ;;;;;;;;;;;;; Display Operation Finished

                MOV AX,4C00H
                INT 21H    ; Normal End Request
MAIN ENDP
END MAIN
;-----------------------------------------------
```

```
TITLE "Program to Print the Fibonaci Series. [1 2 3 5 8 13 21 . . ]"
;-------------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
    VAL1 DB 0
    VAL2 DB 1
. CODE
;-------------------------------------------------------------
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

    MOV AL,VAL1 ; A = VAL1
    MOV BL,VAL2 ; B = VAL2
    MOV CX,10
    MOV AH,00H

    TOP:
        ADD AL,BL ; P= A+B
        MOV BH,BL ; TEMP1=B
        PUSH AX  ; CONTENT OF A IS PUSHED TO STACK

                    ;;;;;;;;;;; DISPLAY OPERATION STARTED
        AAM
        ADD AX,3030H
        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H
        MOV DL,DH
        MOV AH,02H
        INT 21H
        MOV DL,' '
        MOV AH,02H
        INT 21H
        ;;;;;;;;;;; DISPLY OPERATION END
        POP AX  ; CONTENT OF STACK IS POPED TO AX
        MOV BL,AL ; B=P
        MOV AL,BH ; A=B

        LOOP TOP  ; DECREMENT THE CONTENT OF CX BY1
                                    ; JUMP TO LABEL TOP UNTIL CX > 0
        MOV AX,4C00H
        INT 21H

MAIN ENDP
END MAIN

;-------------------------------------------------------------
```

```
TITLE "Program To Demonstrate File Handles For Input and Output"
;-----------------------------------------------------
.MODEL SMALL
.STACK
.DATA
    KEYDISP DB 'Enter Character/Number[10]: $'

    KEYINP  DB 20 DUP(' ') ;Additional 2 Characters For 0DH & 0CH
                ;0DH : Enter
                ;0CH : Line Feed
    KEYOUT  DB 'Output : $'
.CODE
;-----------------------------------------------------
DISPLAY MACRO A
    LEA DX,A
    MOV AH,09H
    INT 21H
ENDM
;-----------------------------------------------------
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
                    ; KEYDISP IS A PARAMETER TO MACRO
    DISPLAY KEYDISP ; Invoked DISPLAY MACRO
    CALL READ
                    ; KEYOUT IS A PARAMETER TO MACRO
  DISPLAY KEYOUT    ; Invoked DISPLAY MACRO
    MOV CX,20
    TOP :
        MOV DL,[BX]
        MOV AH,02H
        INT 21H
        INC BX
        DEC CX
        JZ LAST
        JMP TOP
     LAST:
        MOV AX,4C00H
        INT 21H
MAIN ENDP
;-----------------------------------------------------
```

```
;READ PROCEDURE FOR KEYBOARD INPUT USING FILE HANDLES
;AH=FUNCTION 3FH
;BX=00H INDICATES INPUT
;CX=MAXIMUM NO OF CHARACTERS TO ACCEPT
;DX=ADDRESS OF AREA FOR ENTERING CHARACTERS
READ PROC
    MOV AH,3FH
    MOV BX,00H
    MOV CX,20
    LEA DX,KEYINP
    INT 21H
    MOV BX,DX
    RET
READ ENDP
;-------------------------------------------------
END MAIN
```

TITLE "Program to Print the Input String In Reverse Order."

```
;-------------------------------------------------
. MODEL SMALL
. STACK
. DATA
. CODE
;-------------------------------------------------

MAIN PROC
        ;;;;;;;;;;;;;;;;;;;;Data Segment Initialization Started

MOV AX,@DATA
        MOV DS,AX

        ;;;;;;;;;;;;;;;;;;;;Data Segment Initialization Ended


        MOV CX,0                ;Initialize the Counter as 0

        READ_CHAR:
                MOV AH,01H      ;Put AH=01H to Read Character From Keyboard
                INT 21H         ;Reads Character and Puts it in AL
                CMP AL,0DH      ;0DH is the ASCII Code For Enter Key
                                ;Check if Enter Key is Pressed
```

```
    JE END_OF_LINE ;If Enter Pressed Go to END_OF_LINE
        PUSH AX    ;Push Character into Stack
        INC CX     ;Increment Counter CX
        JMP READ_CHAR  ;Unconditional Jump to READ_CHAR


    END_OF_LINE:
        POP DX     ;Pop Character From Stack into DX
        MOV AH,02H     ;Code for Displaying character on VDU
        INT 21H            ;DOS Interrupt Function
        LOOP END_OF_LINE ;Loop Until CX=0

        MOV AX,4C00H  ;End Request
        INT 21H                ;Dos Interrupt Function


MAIN ENDP
END MAIN
;-------------------------------------------------------
```

TITLE "Program To Clear the Screen With Bios Interrupt Function."

```
;-------------------------------------------------------
.MODEL SMALL
.STACK
.DATA
.CODE
;-------------------------------------------------------

MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

  ''''''''''''''''''''''''''''
    ;INT 10H Function 06H : Scroll Up Screen
    ;AH=Function 06H
    ;AL=Number of lines to scroll,or 00H for full screen
    ;BH=Attribute value(color,blinking)
    ;CX=Strating row:column
    ;DX=Ending row :column
    ''''''''''''''''''''''''''''
```

```
    MOV AX,0600H  ;AH=06,AL=00 for Full Screen
    MOV BH,71H    ;White background(7),Blue foreground(1)
    MOV CX,0000H  ;Upper left row:column
    MOV DX,184FH  ;lower right row:column
    INT 10H       ;Bios Interrupt Function

    MOV AX,4C00H
    INT 21H

MAIN ENDP
END MAIN
;-----------------------------------------------------------
TITLE "Program to Set the Cursor in Desired Location"
;-----------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
. CODE
;-----------------------------------------------------------
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

    ;;;;;;;;;;;;;;;;;;;;
    ;INT 10H Function 02H: Set Cursor Position
    ;AH=Request Cursor with AH=02H
    ;BH=Page Number [Default 00H]
    ;DH=Row
    ;DL=Column
    ;;;;;;;;;;;;;;;;;;;;

    MOV AH,02H ;Request Set Cursor
    MOV BH,00  ;Page Number to 0
    MOV DH,08  ;Row 8
    MOV DL,50  ;Column 50
    INT 10H    ;Bios Interrupt Function

    MOV DL,'C' ;Character to be Displayed in DL
    MOV AH,02H ;Character Mode
    INT 21H    ;Dos Interrupt Function

    MOV AX,4C00H
    INT 21H
MAIN ENDP
END MAIN
;-----------------------------------------------------------
```

```
TITLE "Progam to Display Character with Attributes"
;-----------------------------------------------------
.MODEL SMALL
.STACK
.DATA
.CODE
;-----------------------------------------------------

MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

    ;;;;;;;;;;;;;;;;;;;;
    ;INT 10H Function 09H: Display Character
    ;AH=09H
    ;AL=ASCII Character
    ;BH=Page Number
    ;BL=Attribute or Pixel value.
    ;CX=Count
    ;;;;;;;;;;;;;;;;;;;;

    MOV AH,09H  ;Request Display
    MOV AL,01H  ;Happy Face for display
    MOV BH,00H  ;Page Number 0
    MOV BL,0C1H ;Red background,Blue foreground
    MOV CX,79   ;No of repeated characters
    INT 10H     ;Bios Interrupt Function

    MOV AX,4C00H
    INT 21H

MAIN ENDP
END MAIN
;-----------------------------------------------------

;Program to Demonstrate Screen Scroll Down

;-----------------------------------------------------
.MODEL SMALL
.STACK
.DATA
.CODE
;-----------------------------------------------------
```

```
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;INT 10H Function 07H : Scroll Down Screen
    ;AH=Function 07H
    ;AL=Number of lines to scroll,or 00H for full screen
    ;BH=Attribute value(color,blinking)
    ;CX=Strating row:column
    ;DX=Ending row :column
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    MOV AX,0702H  ;AH=07,AL=00 for Full Screen
    MOV BH,71H    ;White background(7),Blue foreground(1)
    MOV CX,0C19H  ;Upper left row:column
    MOV DX,1236H  ;lower right row:column
    INT 10H       ;Bios Interrupt Function

   MOV AX,4C00H
   INT 21H
MAIN ENDP
END MAIN
```

```
;-------------------------------------------------------
```

TITLE "Program to Plot Pixels in Graphics Mode"

```
;-------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
. CODE
;-------------------------------------------------------
MAIN PROC
   MOV AX,@DATA
   MOV DS,AX

   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   ;
   ;INT 10H: Get Current Video Mode
   ;AH=0FH
   ;AL=Returns Current Video Mode
   ;AH=Number of Screen Columns
   ;BH=Active Video Page
```

```
MOV AH,0FH
INT 10H
PUSH AX ;PUSH the Current Mode to Stack

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;INT 10H : Set Video Mode
;AH=00H
;AL=Required Mode
MOV AH,00H
MOV AL,13H
INT 10H
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;INT 10H : Write Pixel Dot
;AH=0CH
;AL=Color of the Pixel
;BH=Page Number
;CX=Column
;DX=ROW
MOV BH,00
MOV DX,10
MOV CX,10
MOV BL,00
TOP:
    MOV AH,0CH
    MOV AL,BL
    INT 10H
    INC DX
    INC BL
    CMP DX,100
    JNE TOP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;KEYBOARD INPUT
;AH=01H
;AL=Returns Key IN ASCII
```

```
    MOV AH,01H
    INT 21H
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;
    ;Return to Previous Graphics Mode

    POP AX
    MOV AH,00H
    INT 10H
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


    MOV AX,4C00H
    INT 21H

MAIN ENDP
END MAIN
;-------------------------------------------------------------
TITLE "Program to Print the String in Reverse Order"

;-------------------------------------------------------------
. MODEL SMALL
. STACK
. DATA
    STRING DB '!EMOC-LEW'
    REVERSE DB 9 DUP(' ')
. CODE
;-------------------------------------------------------------

MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

 LEA SI,STRING  ;Load Effective Address of STRING into SI
    LEA DI,REVERSE ;Load Effective Address of REVERSE into DI
    ADD DI,9       ;Add the Address of DI With 9
    MOV CX,9       ;Initialize Counter as 9
```

```
TOP:
    MOV AL,[SI] ;Content of SI to AL
    MOV [DI],AL ;Content of AL to Content of DI
    INC SI     ;Increment SI
    DEC DI     ;Decrement DI
    LOOP TOP   ;Loop Until  CX!=0

  ADD DI,10  ;Add DI With 10 to Locate to End
  MOV AL,'$' ;Add String Termination Character
  MOV [DI],AL

  MOV AH,09H    ;AH=09 Specifies String
  LEA DX,REVERSE ;Load Effective Address of REVERSE into DX
  INT 21H       ;DOS Interrupt Function

  MOV AH,4CH
  INT 21H

MAIN ENDP
END MAIN

;-----------------------------------------------------------
;Program To Change The String Into Toggle Case
;-----------------------------------------------------------
.MODEL SMALL
.STACK
.DATA
   STRING DB 'Welcome'
   CASE DB 7 DUP(' ')
.CODE
;-----------------------------------------------------------
MAIN PROC
   MOV AX,@DATA
   MOV DS,AX
   LEA SI,STRING ;Load Effective Address of STRING into SI
   LEA DI,CASE   ;Load Effective Address of CASE with DI
   MOV CX,7      ;Load Counter with 7
   TOP:
      CMP CX,0000H ;Compare CX with 0
      JE EXIT   ;If CX=0 then Goto Exit
      MOV AH,[SI]  ;Load Content of SI into AH
CMP AH,60H  ;Compare AH with 60H i. 6 96
      JA ISSMALL
      CMP AH,5AH  ;Compare AH with 5AH i. e 90
      JB ISCAP

      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      ;TO UPPERCASE
```

```
ISSMALL:
            AND AH,11011111B ;Mask With 11011111B
            MOV [DI],AH
            INC SI
            INC DI
            DEC CX
            JMP TOP
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TO LOWERCASE
ISCAP:
            OR AH,00100000B ;Mask With 00100000B
            MOV [DI],AH
            INC SI
            INC DI
            DEC CX
            JMP TOP
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              EXIT:
              MOV AL,'$'   ;Add String Terminator.
        MOV [DI],AL  ;Pass the Content of AL to DI.
        MOV DX,OFFSET CASE
        MOV AH,09H
        INT 21H
        MOV AH,4CH
        INT 21H
MAIN ENDP
END MAIN
;-----------------------------------------------------------------
```

TITLE "TO COMPUTE THE EXPRESSION Y = MX + C"

```
.MODEL SMALL
.STACK
.DATA
.CODE
MAIN PROC
        MOV CX,50
        MOV AX,0050
TOP:    ADD AX,CX
        DEC CX
        JNZ TOP
```

```
        MOV CX,100
        XOR DX,DX
        DIV CX

        AAM
        ADD AX,3030H
        PUSH DX

        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        POP AX
        AAM AX,3030H
        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        MOV AH,4CH
        INT 21H

        MAIN ENDP
END MAIN
```

TITLE "SUM OF NATURAL NUMBERS UPTO 100"

```
.MODEL SMALL
.STACK
.DATA
.CODE
MAIN PROC
        MOV CX,100
        MOV AX,0000
TOP:    ADD AX,CX
        DEC CX
        JNZ TOP
```

```
        MOV CX,100
        XOR DX,DX
        DIV CX

        AAM
        ADD AX,3030H
        PUSH DX

        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        POP AX
        AAM AX,3030H
        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        MOV AX,4C00H
        INT 21H

        MAIN ENDP
END MAIN
```

TITLE "SUM OF ODD NATURAL NUMBER UPTO 100"

```
.MODEL SMALL
.STACK
.DATA
.CODE
MAIN PROC
        MOV CX,100
        MOV AX,0000
```

```
TOP:    DEC CX
        ADD AX,CX
        DEC CX
        JNZ TOP

        MOV CX,100
        XOR DX,DX
        DIV CX

        AAM
        ADD AX,3030H
        PUSH DX

        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        POP AX
        AAM AX,3030H
        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        MOV AX,4C00H
        INT 21H

        MAIN ENDP
END MAIN
```

TITLE "SUM OF EVEN NATURAL NUMBER UPTO 100"

```
.MODEL SMALL
.STACK
.DATA
.CODE
MAIN PROC
        MOV CX,100
        MOV AX,0000
```

```
TOP:
        ADD AX,CX
        DEC CX
        DEC CX
        JNZ TOP

        MOV CX,100
        XOR DX,DX
        DIV CX

        AAM
        ADD AX,3030H
        PUSH DX

        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        POP AX
        AAM AX,3030H
        MOV DL,AH
        MOV DH,AL
        MOV AH,02H
        INT 21H

        XCHG DH,DL
        MOV AH,02H
        INT 21H

        MOV AX,4C00H
        INT 21H

        MAIN ENDP
END MAIN
```

TITLE "PORGRAM TO FIND FACTORIAL OF A GIVEN NUMBER"

```
.MODEL SMALL
.STACK
.DATA
        X DW 6
        Y DW 100
.CODE
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX
        MOV AX,1
        MOV BX,X
```

```
TOP:    MUL BX
        DEC BX
        JNZ TOP

        DIV Y
        MOV CX,DX
        AAM
        ADD AX,3030H

        MOV DL,AH
        MOV BL,AL
        MOV AH,02H
        INT 21H

        MOV DL,B
        MOV AH,02H
        INT 21H

        MOV AX,CX
        MOV DL,AH
        MOV BL,AL
        MOV AH,02H
        INT 21H

        MOV DL,BL
        MOV AH,02H
        INT 21H

        MOV AH,4CH
        INT 21H
        MAIN ENDP
END MAIN
```

```
TITLE "TO PRINT a TO z"
.MODEL SMALL
.STACK
.DATA
.CODE
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX
        MOV DL,'a'
        L1:     MOV AH,02H
                INT 21H
                INC DL
                CMP DL,'z'
                JNZ L1
        MOV DL,'z'
        MOV AH,02H
        INT 21H
        MOV AH,4CH
        INT 21H
        MAIN ENDP
END MAIN
```

```
TITLE "TO PRINT 9 TO 0"
.MODEL SMALL
.STACK
.DATA
.CODE
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX
        MOV DL,'9'
        MOV AH,02H
        INT 21H
        L1:     DEC DL
                MOV AH,02H
                INT 21H
                CMP DL,'0'
                JNZ L1
      MOV AH,4CH
      INT 21H
      MAIN ENDP
END MAIN
```

```
TITLE "PROGRAM TO COMPUTE AVERAGE OF 10 NUMBERS"
;-----------------------------------------------
.MODEL SMALL
.STACK
.DATA
    X DB 10,5,4,3,5,10,10,3,9,1  ;DEFINE ARRAY OF NUMBERS
.CODE
;-----------------------------------------------
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX                    ;DATA SEGMENT INITIALIZATION

  MOV CL,10          ;SET COUNTER TO 10
  MOV SI,0
  MOV AL,0           ;INITIALIZE SUM TO 0
TOP:
  MOV DL,X[SI]
  ADD AL,DL          ;AX=AX+DX

 INC SI             ;INCREMENT DX BY 1
 DEC CL             ;DECREMENT CL BY 1
 JNZ TOP            ;GOTO TO TOP WHEN COUNTER IS NOT 0
    xor ah, ah
MOV BL,10
DIV BL
```

```
    MOV DH,AH

    XOR AH,AH
    AAM             ;BINARY TO UNPACK BCD
    ADD AX, 3030H   ;CONVERT TO ASCII

    MOV DL, AH      ;DL=AH
    MOV dh,AL       ;PRESERVE AL TO BL
    MOV AH,02H
    INT 21H         ;CHARACTER INTERRUPT

    MOV DL, dh
    MOV AH,02H
    INT 21H         ;CHARACTER INTERRUPT

    MOV AH,4CH
    INT 21H                 ;TERMINATION REQUEST

MAIN ENDP                   ;END PROCEDURE
END MAIN                    ;END PROGRAM
```

TITLE "The 8 data bytes are stored from memory location E000H to E007H. Write 8086 ALP to transfer the block of data to new location B001H toB008H."

```
.MODEL SMALL
.STACK 100
.DATA
.CODE
    MAIN PORC
    MOV BL, 08H
    MOV CX, E000H
    MOV EX, B001H
Loop: MOV DL, [CX]
    MOV [EX], DL
    DEC BL
    JNZ loop
    MAIN ENDP
END MAIN
```

TITLE "Write a program to multiply 2 numbers (16-bit data) for8086."

```
.MODEL SMALL
.STACK 100
.DATA
    Multiplier dw 1234H
```

```
        Multiplicant dw 3456H
        Product dw ?
.CODE
        MULT PROC
        MOV AX, @data
        MOV DS, AX
        MOV AX, Multiplicant
        MUL Multiplier
        MOV Product, AX
        MOV Product+2, DX
        MOV AH, 4CH
        INT 21H
        MULT endp
END   MULT
```

TITLE "Write a program to find Largest No. in a block of data. Length of block is 0A. Store the maximum in location result."

```
.MODEL SMALL
.STACK 100H
.DATA
list db 80, 81, 78, 65, 23, 45, 89, 90, 10, 99
result db ?
 .CODE
MAIN PROC
        MOV AX, @data
        MOV DS, AX
        MOV SI, offset List
        MOV AL, 00H
        MOV CX, 0AH
Back:  CMP AL, [SI]
        JNC Ahead
        MOV AL, [SI]
Ahead: INC SI
        LOOP Back
        MOV Result, AL
        MOV AH, 4CH
        INT 21H
        Main endp
END MAIN
```

TITLE "Write an ALP to generate square wave with period of 200μs and address of output device is 55H for 8086microprocessor."

```
        START:    MOV AX,01H
                  OUT 30H, AX
                  ; to generate loop for 200 μs using system frequency 5MHz MOVBX, Count; 7T
        Label:DECBX ;4T
                  JNZ Label   ;10T/7T
                  MOV AX, 00H
                  OUT 30H, AX
                  MOV BX, Count
        Label1: DEC BX
                  JNZ Label1
                  JMP START
```

Note: Find the value of Count using technique used in 8085 so that delay will be of 200 μs.

QUESTIONS

1. Write an Assembly Language Program to Multiply Two 8 Bit Numbers.
2. Write an Assembly Language Program to Divide Two 8 Bit Numbers.
3. Write an Assembly Language Program to Print **1000$** With Combined Parameter AH=09H and AH=02H.
4. Write an Assembly Language Program to Print the String

Microprocessor Programming .Use Only One Parameter AH=02H.

5. Write an Assembly Language Program to Display the Following Sequences
   a. a b c d e f g .....z
   b. 9 8 7 6 5 0
   c. 1 2 3 4 5 6 7 8 9 100
   d. 2 4 6 8 10 100
6. Write an Assembly Language Program to Compute the Expression for y where $y = \sum_{i=0}^{50} i + 50$.
7. Write an Assembly Language Program to Print the Sum of Natural Numbers From 1 to 100.
8. Write an Assembly Language Program to Print the Sum of Even Numbers From 1 to 100.
9. Write an Assembly Language Program to Print the Sum of Odd Numbers From 1 to 100.
10. Write an Assembly Language Program to Print the Average of 10 Numbers.
11. Write an Assembly Language Program to Calculate the Dot Product of 2 Vectors.

    $dotprod = \sum_{i=0}^{n} Ai * Bi$
12. Write an Assembly Language Program to Display 10 Happy Faces with Attributes.[Red Background and Blue Foreground]
13. Write an Assembly Language Program to Set the Cursor in Desired Location.
14. Write an Assembly Language Program to Plot Pixels in Graphics Mode.

15. Write an Assembly Language Program to Draw Double Lined Box.
16. Write an Assembly Language Program to Draw Single Lined Box.
17. Write an Assembly Language Program to Print the Sum of Two Matrices.
18. Write an Assembly Language Program to Print the Difference of Two Matrices.
19. Write an Assembly Language Program to Print and Count Total Number of Vowels from given Input String.
20. Write an Assembly Language Program to Count Total Number of Words in a given Input Sentence.
21. Write an Assembly Language Program to Print the Maximum no from the Given Array/Table.
22. Write an Assembly Language Program to Print the Minimum no from the Given Array/Table.
23. Write an Assembly Language Program to Print the Sum of only Even Numbers.
24. Write an Assembly Language Program to Display **Name of Day** from Input **Day Number.** [Use Direct Addressing Scheme]
25. Write an Assembly Language Program to Sort Numbers Stored in Array into Ascending Order.
26. Write an Assembly Language Program to Sort Numbers Stored in Array into Descending Order.

❑❑❑

5

BASIC I/O, MEMORY R/W AND INTERRUPT OPERATIONS

CHAPTER OBJECTIVE

The first section of this chapter explains the concept of memory mapped I/O and input-output mapped I/O used by microprocessor to communicate with memory and peripheral devices. In second part the requirement of direct memory access (DMA) for high speed data communication with peripheral devices is explained. Finally, the last section introduces the concept of Interrupt based input-output communication. Here the process of interrupt handling, different types of interrupt used in microprocessor is discussed. This section also introduces 8259 Peripheral Interrupt Controller with the help of its block diagram and priority modes used for I/O communication.

CHAPTER OUTLINE

- Memory Mapped I/O, I/O Mapped I/O
- Direct Memory Access (DMA)
  - Introduction, Advantage and Application
- Interrupt
  - 8085 Interrupt Pins and Priority
  - RST Instructions
- 8259 Interrupt Controller
  - Block Diagram and Explanation
  - 8237 DMA Controller and Interfacing
  - Maskable and Non-maskable Interrupts
  - Vector and Polled Interrupt
  - Priority Modes and Additional Features

5.1 Memory Mapped I/O, I/O Mapped I/O

Address space partitioning in 8085

There are two techniuqes through which devices can be interfaced to microprocessor.

1. Memory mapped I/O
2. Peripheral mapped I/O or I/O mapped I/O

Memory mapped I/O

In memory mapped I/O scheme we can use only one address space. This particular one address space is allocated to both memory and I/O devices.

In total memory addresses, some addresses are assigned to memories and some to I/O devices. But we have to assign the address for I/O devices are different from the addresses which have been assigned to memories.

In this scheme remember that I/O device is also treated as a memory location.

Now take a very good example, MOV C, M instruction would transfer one byte of data from a memory location or it can also transfer an input device to the register C,d epending on whether the address in the H-L register pair is assigned to a memory location or to an input device.

If H-L contains address of a memory location, data will be transferred from that memory location to register C, while if H-L pair contains the address of an input device, data will be transferred from that input device to register C.

I/O mapped I/O

In this method separate address space is given to I/O devices. Each I/O device is given an 8-bit address. Hence maximum 256 devices can be interfaced to the processor. The address range for the I/O devices is 00H-FFH. I/O control signals are used to perform read, write operations.

For reading data from I/O device or writing data to IO device IN, OUT instructions needs to be used.

We know that some CPUs provide one or more control lines like IO/M line for 8085, which indicates the status of operation, is memory or I/O operation.

If we get the status of IQ'M' line is high, it indicates I/O operation and when we get low, it points to memory operation. In this case the same address may be assigned to both memory and an I/O device depending on the status of IQ/M line.

The above scheme is referred as I/O mapped I/O scheme. Look in this scheme two separate address spaces exist. One space is meant exclusively for memory operations and the other for I/O operations.

The following figure shows, pictorially, both the schemes. Here it is assumed that the system has a 64 KB of memory and 256 I/O space.

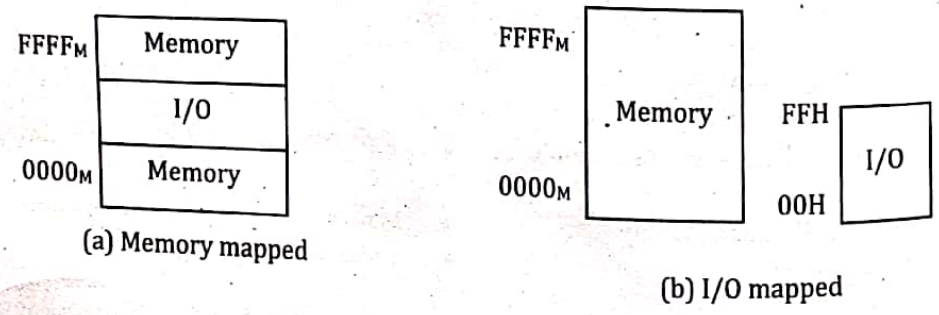


Figure 5.1: Memory map I/O and I/O mapped I/O.

Difference between Memory Mapped IO and IO Mapped IO

Definition

Memory mapped IO is a method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses one address space for memory and IO devices. IO mapped IO is a method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses two separate address spaces for memory and IO devices. Thus, this definition explains the basis of the difference between memory mapped IO and IO mapped IO.

Address Spaces

The main difference between memory mapped IO and IO mapped IO is that the memory mapped IO uses the same address space for both memory and IO devices. IO mapped IO uses two separate address spaces for memory and IO device.

Addresses for Memory

Branching from the above, there is another difference between memory mapped IO and IO mapped IO. As the memory mapped IO uses one address space for both IO and memory, the available addresses for memory are minimum due to the additional addresses for IO. In IO mapped IO, all the addresses can be used by the memory.

Instructions

While memory mapped IO uses the same instructions for both IO and memory operations, IO mapped IO uses separate instructions for read and write operations in IO and memory. We can say this as one other difference between memory mapped IO and IO mapped IO.

Efficiency

Moreover, memory mapped IO is less efficient while IO mapped IO is more efficient.

Conclusion

Memory mapped IO and IO mapped IO are two methods to perform input/output operations between the CPU and peripheral devices in the computer. The basic difference between memory mapped IO and IO mapped IO is that memory mapped IO uses the same address space for both memory and IO device while IO mapped IO uses two separate address spaces for memory and IO device.

Memory Mapped IO Versus IO Mapped IO

Memory Mapped IO	IO Mapped IO
A method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses one address space for memory and IO devices.	A method to perform input/output (I/O) operations between the central processing unit (CPU) and peripheral devices in a computer that uses two separate address space for memory and IO device.
Uses the same address space for both memory and IO devices.	Uses the two separate address space for memory and IO device.
As the memory mapped IO uses one address space for both IO and memory, the available address for memory are minimum.	All the addresses can be used by the memory.
Uses the same instructions for both IO and memory operations	Uses separate instructions for read and write operations in IO and memory.
Less efficient	More efficient

Table 5.1: Difference between memory mapped IO and IO mapped IO.

5.2 Direct Memory Access (DMA)

The data transfer technique in which peripherals manage the memory buses for direct interaction with main memory without involving the CPU is called direct memory access (DMA). Using DMA technique large amounts of data can be transferred between memory and the peripheral without severely impacting CPU performance. During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device(s) and main memory. The structure of DMA controller is described below.

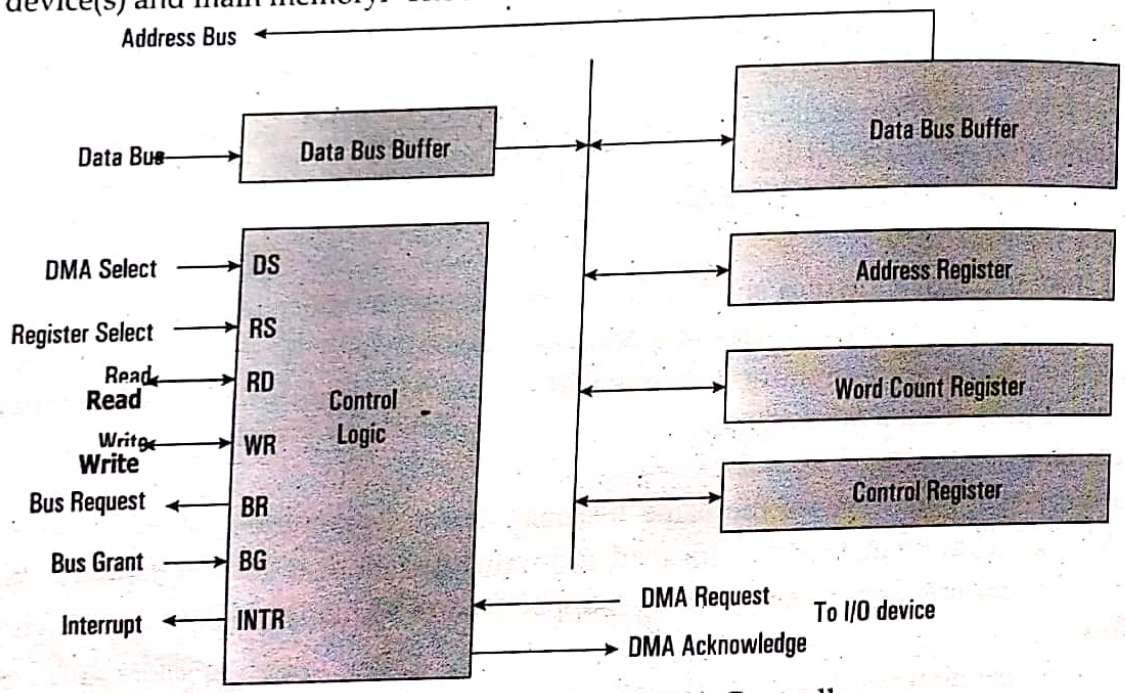


Figure 5.2: Block Diagram of DMA Controller.

The control unit communicates the CPU via data bus and control lines. The DMA controls/ relinquishes the system bus using BR (Bus Request) and BG (Bus Grant) signals. DMA operates read and write operations via RD (Read) and WR (Write) signals. DMA sends request and acknowledge to I/O devices via DMA request and DMA acknowledge signals. The registers in DMA are selected by CPU through the address bus by enabling DS (DMA Select) and RS (Register Select) inputs. All registers in the DMA appear to the CPU as I/O interface registers. The address register contains an address to specify the desired location in memory. It is incremented after each word that is transferred to the memory. The word count register holds the number of words to be transferred. It is decremented by one after each word transfer and internally tested for zero. The control register specifies the mode of transfer. The DMA transfer operation is described below:

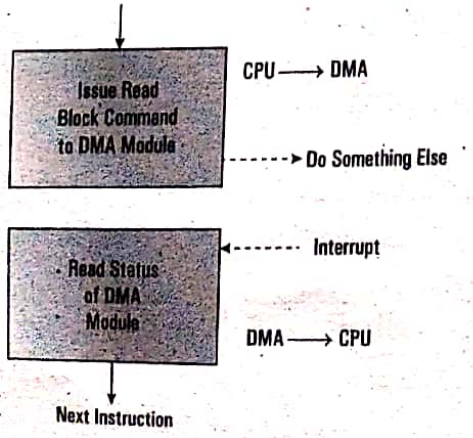


Figure 5.3: Direct Memory Access Technique

The DMA controller requests CPU to handle control of buses to the DMA using bus request (BR) signal. The CPU grants the control of buses to DMA using bus grant (BG) signal after placing the address bus, data bus and read and write lines into high impedance state (which behave like open circuit).

CPU initializes the DMA by sending following information through the data bus.

1. Starting address of memory block for read or write operation.
2. The word count which is the number of words in the memory block.
3. Control to specify the mode of transfer such as read or write.
4. A control to start the DMA transfer.

The DMA takes control over the buses and directly interacts with memory and I/O units to transfer the data without CPU intervention. When the transfer completes, DMA disables the BR line. Thus CPU disables BG line, takes control over the buses and returns to its normal operation.

The DMA transfer operation is illustrated below:

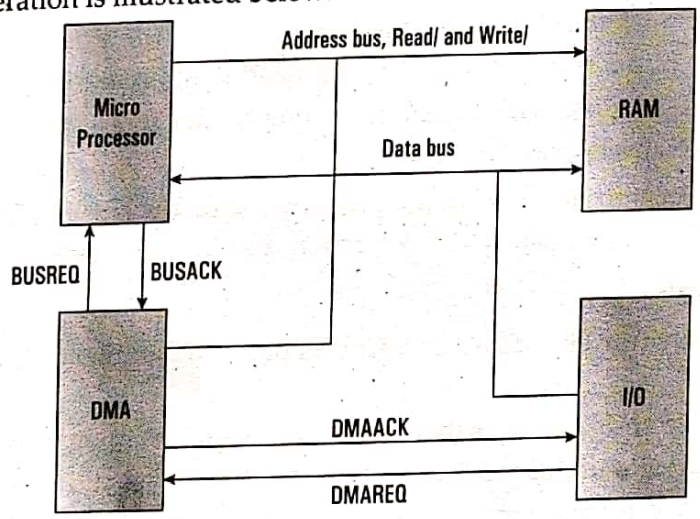


Fig 5.4: Illustration for DMA transfer in a computer system

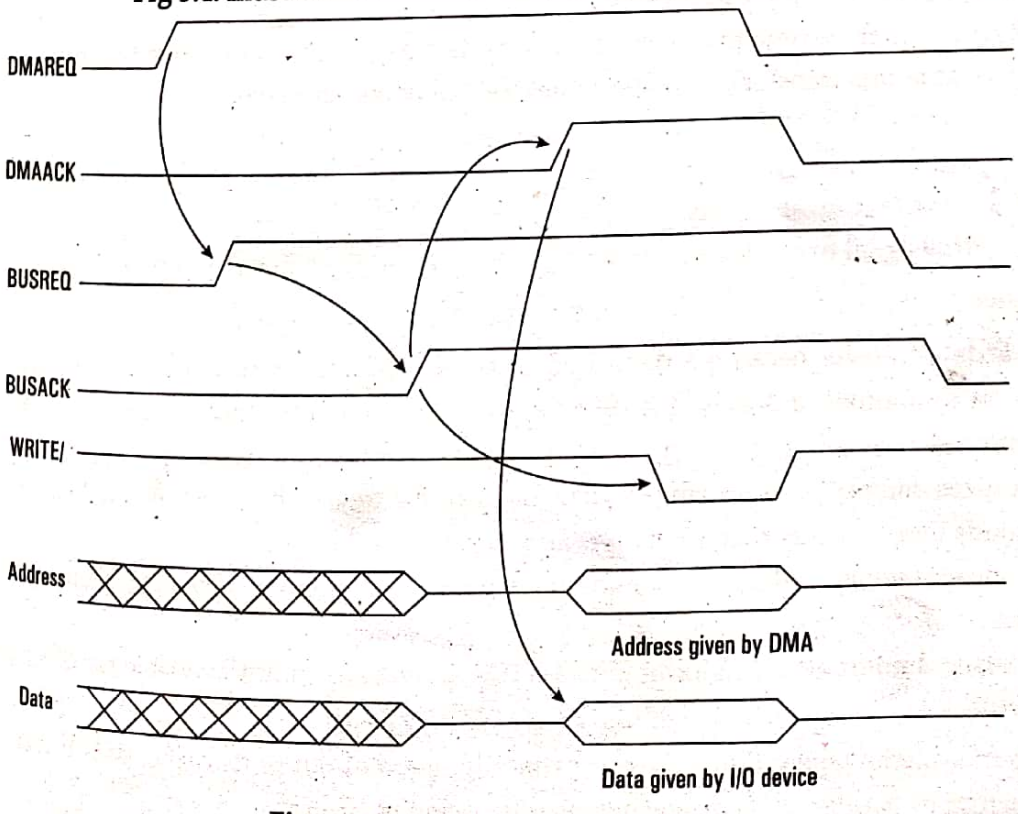


Figure 5.5: DMA Timing Diagram

DMA Transfer Modes

There are 3 different modes of DMA data transfer. They vary by how DMA controller determines when to transfer data (but actual data transfer process remains the same in all three cases).

(a) Burst Mode

- Entire block of data is transferred in one continuous sequence.
- Once the DMA controller is granted access to the system bus by CPU, it transfer all bytes of data in the data block before relinquishing control of system buses back to the CPU.
- This mode is useful for loading programs or data files into memory, but it keeps CPU idle for relatively long period of time.

(b) Cycle Stealing Mode

- DMA controller obtains access to system bus as in burst mode; transfers one byte of data and returns the control of the system bus to CPU. It continually issues requests using Bus Request (BR) signals, transferring one byte of data per request, until it has transferred its entire block of data. (steals one CPU cycle).
- The data block is not transferred as quickly as in burst mode, but the CPU is not idled for long period of time as in burst mode.

(c) Transparent Mode

- DMA controller only transfers data when CPU is performing operations that do not use system buses.
- The main advantage of this mode is that CPU never stops executing its program.
- The main disadvantages of this mode are
  - Hardware needed to determine whether the CPU is not using the system buses can be quite complex and relatively expensive.
  - Requires highest time to transfer a block of data as compared to above two modes.
- To serve as a data transfer processor, DMA controller should have:
  - A data bus
  - An address bus
  - Read/Write control signals
  - Control signal to disable its role as a peripheral and to enable it as a processor

Advantages

1. Quick data transfer because a dedicated piece of hardware transfers data from one computer location to another and only one or two bus read/write cycles are required per piece of data transferred.
2. Minimizes latency in servicing a data acquisition device because the dedicated hardware responds more quickly than interrupts and transfer time is short.
3. Minimizes latency reduces the amount of temporary storage (memory) required on an I/O device.
4. Processor is not used for holding the data transfer activity and is available for other processing activity.
5. Also in systems where the processor primarily operates out of its cache, data transfer actually occurring in parallel, thus increasing overall system utilization.

Application

1. Extensively used for computer-based data acquisition applications including streaming data to disk, real-time screen data display and continuous data acquisition applications.
2. DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions.
3. PC-based DMA technology, along with high-speed bus technology, is driven by data storage, communications and graphics needs-all of which require the highest rates of data transfer between system memory and I/O devices.
4. Data acquisition applications have the same needs and therefore can take advantage of the technology developed for larger markets.

8237 DMA Controller and Interfacing

1. Is a programmable DMA controller.
2. It is a 40 pin package.
3. It must interface with two types of devices-MPU and peripherals (e. g.. : floppy disk, pen drive, etc.)
4. Following is the logical pin diagram for 8237 DMA controller:

8237 has four independent channels, CH_0 to CH_3. 16 bit registers are associated with each channel. In the 8 registers from top, one stores starting address of byte to be copied and the next store the count. The next eight registers are accessed by MPU. The addresses of these register are determined by address lines, A_3 to A_0 and the chip select (CS). These registers are used to write command or read status.

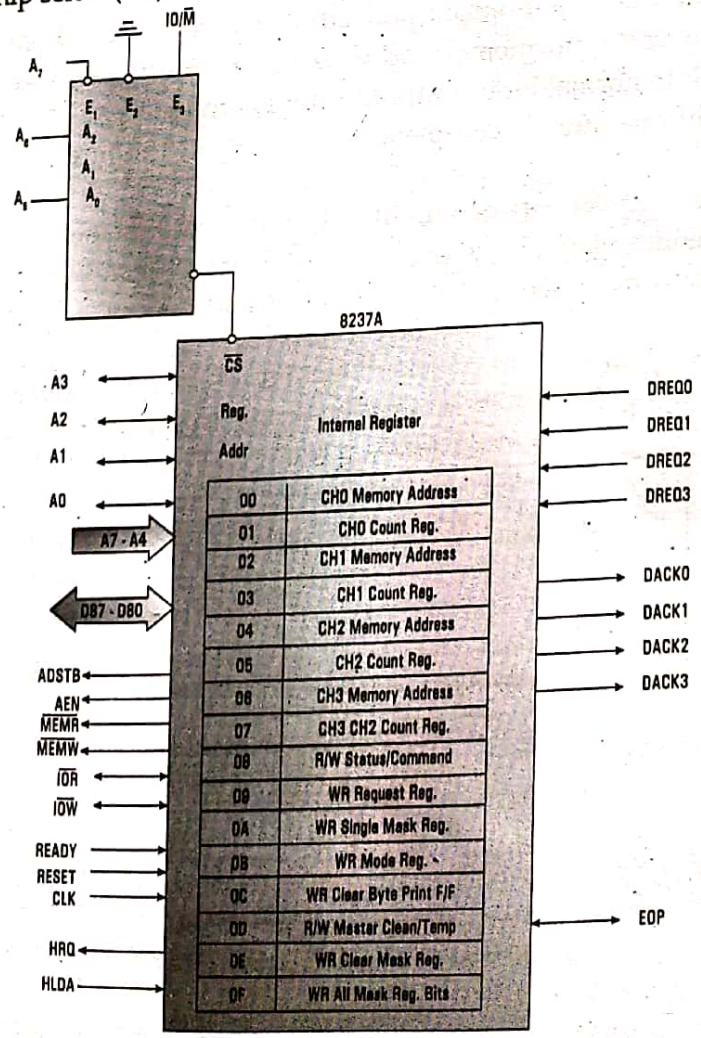


Figure 5.6: Block diagram of 8237 DMA controller and interfacing

Below are some important DMA Signals:

1. **DREQ0-DREQ3 (DMA Request):** Independent, asynchronous input signals to DMA channels from peripherals. A request is generated by activating the DREQ line to obtain DMA service
2. **DACK0-DACK3 (DMA Acknowledge):** Output lines to inform individual peripheral that DMA is granted.
3. **AEN and DSTB (Address Enable and Address Scribe):** High output signals to latch a high order address bytes to generate 16-bit address.
4. **MEMR and MEMW (Memory Read and Memory Write):** Output signal is read and write from memory.
5. **A3-A0 and A7-A4-Address:** A3-A0 are bidirectional address lines used as input to access control registers. During DMA cycle, these lines are used as output lines to generate a low order address that combines with the remaining address lines A7-A4.
6. **HRQ and HLDA (Hold Request and Hold Acknowledge):** HRQ is the output signal used to request the MPU control of the system bus. After receiving the HRQ, the MPU completes bus cycle in process and issue the HLDA signal.

Direct memory access with DMA controller 8257/8237

Suppose any device which is connected at input-output port wants to transfer data to transfer data to memory, first of all it will send input-output port address and control signal, input-output read to input-output port, then it will send memory address and memory write signal to memory where data has to be transferred. In normal input-output technique the processor becomes busy in checking whether any input-output operation is completed or not for next input-output operation, therefore this technique is slow.

This problem of slow data transfer between input-output port and memory or between two memory is avoided by implementing Direct Memory Access (DMA) technique. This is faster as the microprocessor/computer is bypassed and the control of address bus and data bus is given to the DMA controller.

- HOLD – hold signal
- DREQ – DMA request
- HLDA – hold acknowledgment
- DACK – DMA acknowledgment

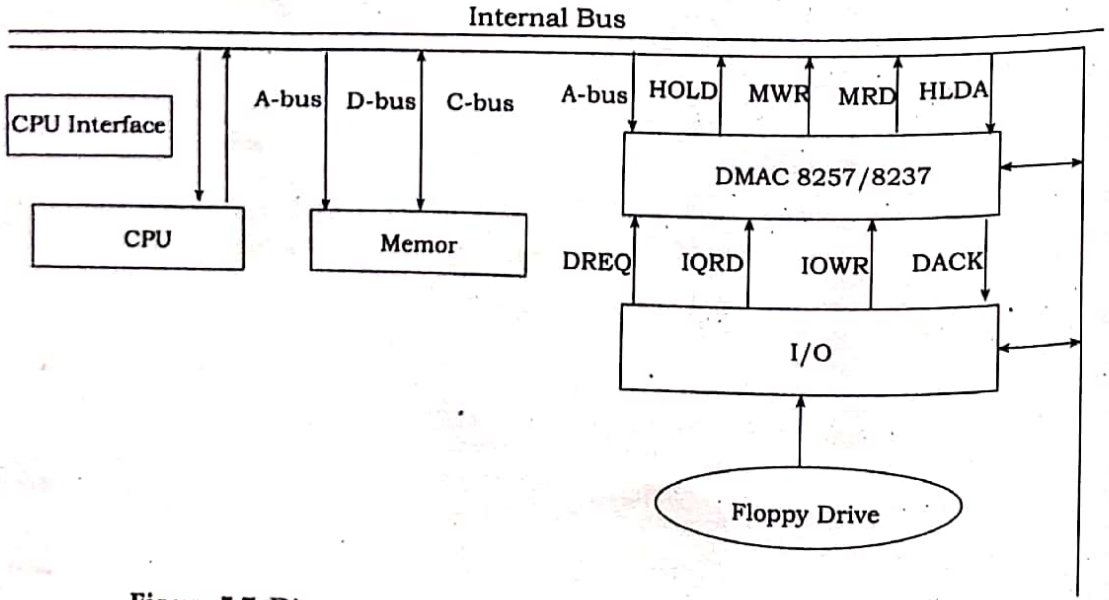


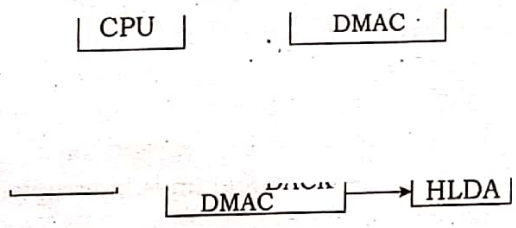
Figure 5.7: Direct memory access with DMA controller 8257/8237

Suppose a floppy drive which is connected at input-output port wants to transfer data to memory, the following steps are performed:

- **Step-1:** First of all the floppy drive will send DMA request (DREQ) to the DMAC, it means the floppy drive wants its DMA service.
- **Step-2:** Now the DMAC will send HOLD signal to the CPU.
- **Step-3:** After accepting the DMA service request from the DMAC, the CPU will send hold acknowledgement (HLDA) to the DMAC, it means the microprocessor has released control of the address bus the data bus to DMAC and the microprocessor/computer is bypassed during DMA service.
- **Step-4:** Now the DMAC will send one acknowledgement (DACL) to the floppy drive e=which is connected at the input-output port. It means the DMAC tells the floppy drive be ready for its DMA service.
- **Step-5:** Now with the help of input-output read and memory write signal the data is transferred from the floppy drive to the memory.

Modes of DMAC:

1. **Single Mode** – In this only one channel is used, means only a single DMAC is connected to the bus system.
2. **Cascade Mode** – In this multiple channels are used, we can further cascade more number of DMACs.



5.3 Interrupt

Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced. In response to an interrupt, the processor stops what it is currently doing and executes a service routine. When the execution of the service routine is terminated, the original process may resume its previous operation. The interrupt is initiated by an external device and is asynchronous, meaning that it can be initiated at any time without reference to system clock. However, the response to an interrupt request is directed or controlled by the microprocessor. Interrupts are primarily issued on:

1. Initiation of I/O operation.
2. Completion of I/O operation.
3. Occurrence of hardware or software errors.

Need for Interrupts

Interrupt is a signal sent by an external device to the processor, to the processor to perform a particular task or work. Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.

When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal to the interrupt pin of the processor. If the processor accepts the interrupt then the processor suspends its current activity and executes an interrupt service subroutine to complete the data transfer between the peripheral and the processor. After executing the interrupt service routine the processor resumes its current activity. This type of data transfer scheme is called interrupt driven data transfer scheme.

Process of Interrupt Operation

1. The I/O unit issues an interrupt signal to the processor. An interrupt signal from I/O is the request for exchange of data with the processor.
2. The processor finishes execution of the current instruction before responding to the interrupt.
3. The processor tests for an interrupt, determines that there is one, and sends an acknowledgement signal to the device that issued the interrupt. After receiving this acknowledgement, the device retains its interrupts signal.
4. The processor now begins to transfer the control to the routine which serves the interrupt request from the device. This routine is called 'Interrupt service routine' and it resides at a specified memory location.
   For this process, the CPU needs to save information needed to reassume the current program at the point of interrupt. The minimum information required is (i) the status of the processor, which is contained by the process or status word (PSW) and (ii) the location of the next instruction to be executed which is contained by the program counter (PC), these all are pushed onto the stack.
5. The processor then loads the program counter with the entry location of the interrupt service routine that will respond to this interrupt. Once the program counter has been loaded, the control is transferred to the interrupt handler program.
6. The fundamental requirement of the interrupt service routine is that it should begin by saving the contents of all the registers on the stack (as state of the main program should be safe). Suppose the user program is interrupted after the instruction at location N. The contents of all the registers plus the address of the next instruction are saved on the stack. The stack pointed is updated and the programs counter is updated to point to the beginning of the interrupt service routine.
7. The interrupt handler now proceeds to process the interrupt. This will include an examination of status information relating to the I/O operation or the other event that caused an interrupt. It may also involve sending additional commands or acknowledgement to the I/O unit.
8. When interrupt processing is complete the saved register's value (of the main program) are retrieved from the stack and restored to the register.
9. The final function is to restore the PSW and program counter values from the stack. As a result the next instruction to be executed will be from the previously interrupted main program.

Types of Interrupts

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts
2. Internal interrupts
3. Software interrupts.

1. External Interrupts

External interrupts are initiated via the microprocessor's interrupt pins by external devices I/O devices, timing device, circuit monitoring the power supply, etc. Causes of these interrupts maybe; I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event, or power failure. Time out interrupt may result from a program that is an endless loop and thus exceeded its time allocated. Power failure interrupt may have as its service routine a program that transfers the complete state of the CPU into a non-destructive memory in few milliseconds before power ceases.

An external device initiates the hardware interrupts of 8085 by placing an appropriate signal at the interrupt pin of the processor. The processor keeps on checking the interrupt pins at the second T - state of last machine cycle of every instruction. If the processor finds a valid interrupt signal and if the interrupt is unmasked and enabled, then the processor accepts the interrupt. The acceptance of the interrupt is acknowledged by sending an INTA signal to the interrupted device.

The processor saves the content of PC (program Counter) in stack and then loads the vector address of the interrupt in PC. (If the interrupt is non-vectored, then the interrupting device has to supply the address of ISR when it receives INTA signal). It starts executing ISR in this address. At the end of ISR, a return instruction, RET will be placed. When the processor executes the RET instruction, it POPs the content of the top of stack to PC. Thus the processor control returns to main program after servicing interrupt. The hardware interrupts of 8085 are TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR. External interrupts can be further divided into two types:

- Maskable interrupt.
- Non-maskable interrupt.

Maskable interrupt

A maskable interrupt is one which can be enabled or disabled by executing instructions such as EI (enable interrupts) and DI (Disable interrupt). If the microprocessor's interrupt enable flipflop' is disabled, it ignores a maskable interrupt. In 8085, the 1 byte instruction EI sets the interrupt enable flipflop and enables the interrupt process. Similarly the 1 byte instruction DI resets the interrupt enable flipflop and disables the interrupt process. No maskable interrupts are recognized by the processor when the interrupt is disabled.

Non maskable interrupt

This type of interrupt cannot be enabled or disabled by instructions. This type has higher priority over the maskable interrupt. This means that if both the maskable and nonmaskable interrupts are activated at the sametime, then the processor will service the non-maskable interrupt first. In 8085 TRAP is an example of nonmaskable interrupt.

2. Internal Interrupt

Internal interrupts arise from illegal or erroneous use of an instruction or data. Cause of this interrupt may be: register overflow, attempt to divide by zero, an invalid operation code, stack overflow etc. These error conditions usually occur as a result of premature termination of the instruction execution. These are even termed as exceptions.

The difference between internal and external interrupt is that the internal interrupt is initiated by some exceptional conditions caused by the program itself rather than by an external events. Internal interrupts are synchronous with the program, while external interrupts are asynchronous. If the program is return, the internal interrupt will occur in the same place each time. External interrupts depend on external conditions that are independent of the program being executed at the time.

3. Software Interrupt

External and internal interrupts are initiated from signal that occur in the hardware of the CPU. A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure. In 8085 the instruction like RST 0, RST 1, RST 2, RST 3..... , etc. , causes a software interrupt. The vector addresses of software interrupts are given in table below.

Interrupt	Vector address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

Interrupt	Vector address
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH
TRAP	0024H

Table 5.2: Vector address of software interupts

The software interrupt instructions are included at the appropriate (or required) place in the main program. When the processor encounters the software instruction, it pushes the content of PC (Program Counter) to stack. Then loads the Vector address in PC and starts executing the Interrupt Service Routine (ISR) stored in this vector address. At the end of ISR, a return instruction - RET will be placed. When the RET instruction is executed, the processor POP the content of stack to PC. Hence the processor control returns to the main program after servicing the interrupt. Execution of ISR is referred to as servicing of interrupt.

All software interrupts of 8085 are vectored interrupts. The software interrupts cannot be masked and they cannot be disabled. The software interrupts are RST 0, RST 1, ... RST 7 (8 Nos).

Vectored and Non-Vectored Interrupts –

A vectored interrupt is where the CPU actually knows the address of the Interrupt Service Routine in advance. All it needs is that the interrupting device sends its unique vector via a data bus and through its I/O interface to the CPU. The CPU takes this vector, checks an interrupt table in memory, and then carries out the correct ISR for that device. So the vectored interrupt allows the CPU to be able to know what ISR to carry out in software (memory).

Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address. Vector Addresses are calculated by the formula 8 * TYPE

Interrupt	Vector address
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3C H

For Software interrupts vector addresses are given by:

Interrupt	Vector address
RST 0	00 H
RST 1	08 H
RST 2	10 H
RST 3	18 H
RST 4	20 H
RST 5	28 H
RST 6	30 H
RST 7	38 H

Non-Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. INTR is the only non-vectored interrupt in 8085 microprocessor.

A non-vectored interrupt is where the interrupting device never sends an interrupt vector. An interrupt is received by the CPU, and it jumps the program counter to a fixed address in hardware. This is literally a hard coded ISR which is device agnostic. MIPS uses this via it's syscall instruction, which is the same instruction you use regardless what device, if it is an external event notifier (e.g. keyboard input), a completion signal (e.g. print completion), a clock interrupt to tell the CPU to allocate control to a different device, or an abnormal event indicator (e.g. power failure). The CPU crucially does not know which device caused the interrupt without polling each I/O interface in a loop and checking the status register of each I/O interface to find the one with status "interrupt created".

Maskable and Non-Maskable Interrupts–Maskable

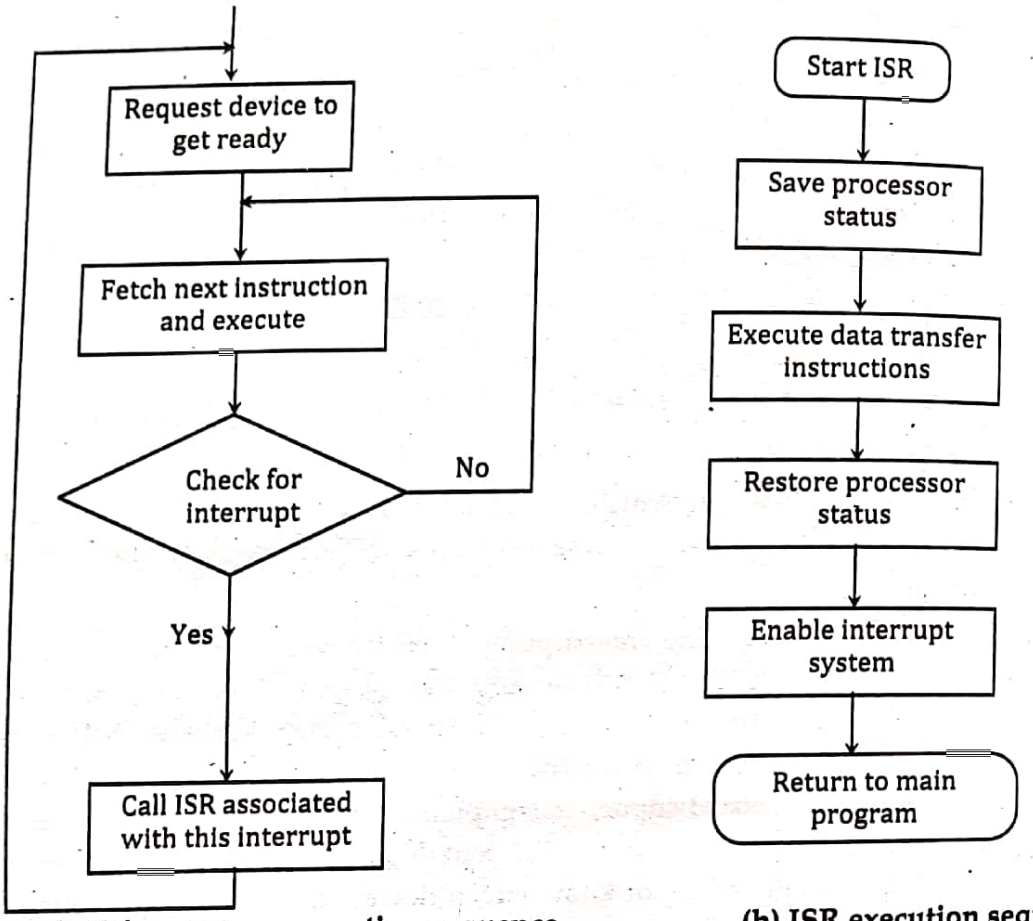
Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 microprocessor.

Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. TRAP is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

Interrupt Driven Data Transfer Scheme

The interrupt driven data transfer scheme is the best method of data transfer for effectively utilizing the processor time. In this scheme, the processor first initiates the I/O device for data transfer. After initiating the device, the processor will continue the execution of instructions in the program. Also at the end of an instruction the processor will check for a valid interrupt signal. If there is no interrupt then the processor will continue the execution.

When the I/O device is ready, it will interrupt the processor. On receiving an interrupt signal, the processor will complete the current instruction execution and saves the processor status in stack. Then the processor calls an interrupt service routine (ISR) to service the interrupted device. At the end of ISR the processor status is retrieved from stack and the processor starts executing its main program. The sequence of operations for an interrupt driven data transfer scheme is shown in figure below.



(a) Main program execution sequence **(b) ISR execution sequence.**

Figure 5.8: Interrupt driven data transfer scheme

Interrupt Priority

Data transfer between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU. The readiness of the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then the program branches to a service routine that processes the required transfer. In micro-computer a number of I/O device are attached to the processor, with each device being able to originate an interrupt request. The first task of the interrupt system is to identify the source of the interrupt. There is also the possibility that several sources will request service simultaneously. In this case the system must also decide which device to service first. An interrupt priority is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced. Higher priority interrupt levels are assigned to requests which, if delayed or interrupted, could have serious consequences. Device with higher transfers speed such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority. When two devices interrupt the processor at the same time, the processor services the device, with the higher priority first.

There are mainly two ways of servicing multiple interrupts. These are:

- Polled interrupt.
- Chained (Vectored) interrupt.

Polled Interrupt

Polled interrupts are handled using software and are therefore slower compared to vectored (hardware) interrupts. In this method there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupts sources in sequence. The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise the next lower priority source is tested, and so on. Thus, the initial servicer outline for all interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines.

Polled interrupts are very simple, but on large number of devices, the time required to poll each device may exceed the service to the device. In such case, a faster mechanism called chained interrupt is used.

Chained Interrupt

This is a hardware concept of handling the multiple interrupts. In this technique, the devices are connected in a chain fashion as shown in figure below for setting up the priority system.

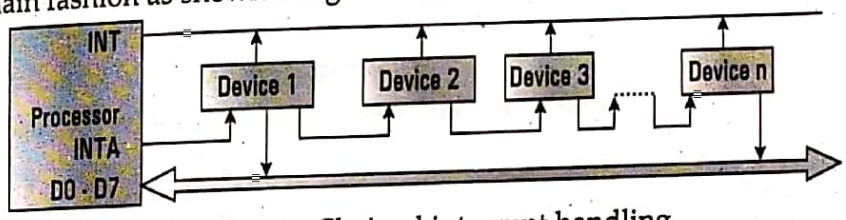


Figure 5.9: Chained interrupt handling.

Here, the device with the highest priority is placed in the first position, followed by lower priority devices. Suppose that one or more devices interrupt the processor at a time. In response, the processor saves its current status and then generates an interrupt acknowledge (INTA) signal to the highest priority device, which is device 1 in our case. If this device has generated the interrupt it will accept the INTA signal from the processor; otherwise, it will pass INTA onto the next device until the INTA is accepted by the interrupting device.

Once accepted, the device provides same answer to the processor for finding the interrupt address vector using external hardware. Usually there requesting device responds by placing a word on the data lines. With the help of hardware it generates interrupts vector address. This word is referred to as vector, which the processor uses as a pointer to the appropriate device service routine.

This avoids the need to execute a general interrupt service routine first. So this technique is also referred to as vectored interrupts.

Interrupts of 8085

The 8085 has five hardware interrupts:

- TRAP
- RST 7. 5
- RST 6. 5
- RST 5. 5
- INTR

The four interrupts TRAP, RST 7. 5, 6. 5,5. 5 are automatically vectored (transferred) tospecific locations on without any external hardware. They do not require INTA signal or an input port; the necessary hardware is already implemented inside the 8085. These interrupts and their call locations are:

INTERRUPT	CALL LOCATIONS
TRAP	0024 H
RST 7. 5	003C H
RST 6. 5	0034 H
RST 5. 5	002C H

Table 5.3: Interrupts of 8085

TRAP

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

It is an on maskable interrupt. It has the highest priority among the interrupt signal. It need not been abled and it cannot be disabled. When this interrupt is triggered the program control is transferred to the location 0024H without any external hardware or the interrupt enable instruction. TRAP is generally used for such critical events as power failure and emergency shutoff. RST7.5, 6.5, 5.5: These interrupts are maskable and are enabled by software using instructions EI and SIM (set interrupt mask). The execution of the instruction SIM enables/disables the interrupts according to the bit pattern of the accumulator.

RST 7.5

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

RST 6.5

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

RST 5.5

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

INTR

This interrupt is maskable. It can been abled by instruction EI and can be disabled by instruction DI. The INTR interrupt requires external hardware to transfer program sequence to specific CALL locations. There are 8 numbers of CALL-Locations for INTR interrupt. The hardware circuit generates RST codes for this purpose and places that on the data bus externally.

When the microprocessor is executing a program, it checks the INTR line (when interrupt enable flipflop is enabled using EI instruction) during the execution of each instruction. If the line is high and the interrupt is enabled, the microprocessor completes the currents instruction, disabled the interrupt enable flipflop and sends a INTA signal. The processor does not accept any interrupt requests until the interrupt flip flop is enabled again.

The signal INTA is used to insert a Restart (RST) instruction, (it saves the memory address of the next instruction to the stack. The program is transferred to the call location.). The RST instruction and their call locations are:

Instruction	Hex-code	Call Location
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

(i) Bit D3 is a control bit and should be 1 for bits D_0, D_1 and D_2 to be effective. Logic 0 on D_0, D_1 and D_2 will enable the corresponding interrupts and logic 1 will disable the interrupts.

(ii) The second function is to reset RST 7.5 flipflop. Bit D4 is additional control for RST 7.5 If D4 = 1, RST7.5 is reset. This is used to ignore RST 7.5 without servicing it.

(iii) The third function is to implement serial I/O. Bit D7 and D6 are used for serial I/O and do not affect the interrupts.

Assuming that the task to be performed is written as a subroutine at the specified location the processor performs a task. This service routine includes the instruction EI to enable the interrupt again and RE-instruction to retrieve the memory address where the program has interrupted. Then the execution goes to the main program again.

SET Interrupt Mask (SIM) Instruction

⌘ This is 1 byte instruction.

⌘ Can be used for three different functions:

(i) One function is set mask for RST 7.5, 6.5 and 5.5 interrupts. This instruction reads the content of the accumulator and enables or disables the interrupts.

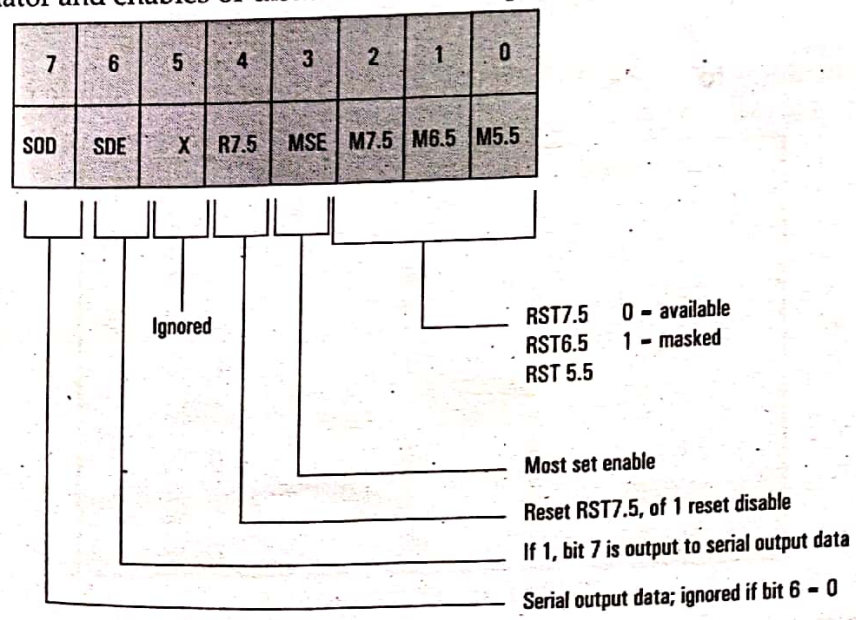


Figure 5.10: SIM Instruction.

Pending Interrupts

When one interrupt request is being served, other interrupt may occur resulting in a pending request. When more than one interrupts occur. Simultaneously the interrupts having higher priority is served and the interrupts with lower priority remain pending. The 8085 has an instruction RIM using which the programmer can know the current status of pending interrupts. This instructions gives the current status of only maskable interrupts.

Read Instruction Mask (RIM)

- ⌘ Read interrupt Mask.
- ⌘ 1 byte instruction.
- ⌘ Can be used for the followings.

a. To read interrupt mask. This instruction loads the accumulator with 8-bits indicating the current status of the interrupts.

b. To identify the pending interrupts. Bits D4, D5, and D6 identify the pending interrupts.

c. To receive serial data. Bit D7 is used to receive serial data.

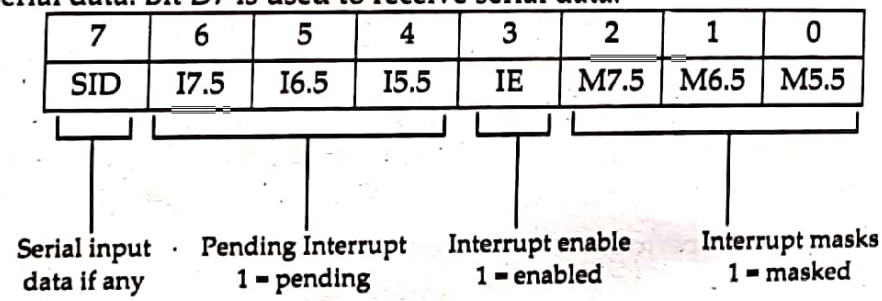


Figure 5.11: RIM Instruction

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

RST Instructions

The signal INTA is used to insert a Restart (RST) instruction, (it saves the memory address of the next instruction to the stack. The program is transferred to the call location). The RST instructions and their call locations are:

- ⌘ Bit D_3 is a control bit and should be 1 for bits D_0, D_1, and D_2 to be effective.
- ⌘ Logic 0 on D_0, D_1, and D_2 will enable the corresponding interrupts and logic 1 will disable the interrupts.
- ⌘ The second function is to reset RST 7. 5 flip flop. Bit D_4 is additional control for RST 7. 5
- ⌘ If D_4 =1, RST 7. 5 is reset. This is used to ignore RST 7. 5 without servicing it.

Instruction	Hex-code	Call location
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

Table 5.4: RST Instructions

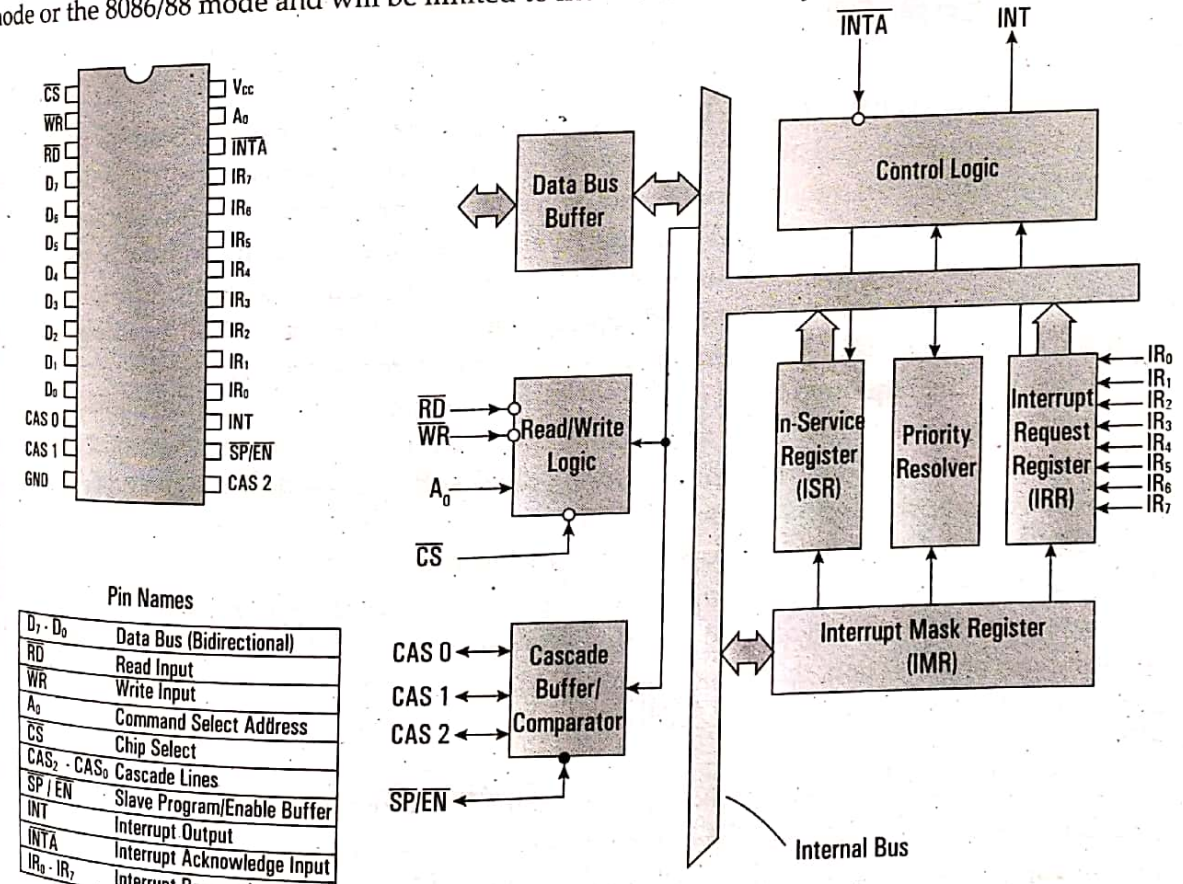
- ⌘ The third function is to implement serial I/O. Bit D_7 and D_6 are used for serial I/O and do not affect the interrupts.

5.4 The 8259 Programmable Interrupt Controller

The 8259A programmable interrupt controller designed to work with Intel microprocessors 8085, 8086 and 8088. The 8259A interrupt controller can

1. Manage eight interrupts according to the instructions written into its control registers. This is equivalent to placing eight interrupt pins on the processor in place of one INTR (8085) pin.
2. Vector can interrupt request anywhere in the memory map. However, all eight interrupts are spaced at the interval of either four or eight locations. This eliminates all the major drawback of the 8085 interrupts in which all interrupts are vectored to memory locations on page 00H.
3. Resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode.
4. Mask each interrupt request individually.
5. Read the status of pending interrupts, in-service interrupts, and masked interrupts.
6. Be set up to accept either the level-triggered or the edge-triggered interrupt request
7. Be expanded to 64 priority levels by cascading additional 8259As.
8. Be set up to work with either the 8085 microprocessor mode or the 886/8088 microprocessor mode.

The 8259A is upward-compatible with its predecessor, the 8259. The main difference between the two is that the 8259A can be used with Intel's 8086/88 16-bit microprocessor. It also includes additional features such as the level-triggered mode, buffered mode, and automatic-end-of interrupt mode. To simplify the explanation of the 8259A, illustrative examples will not include the cascade mode or the 8086/88 mode and will be limited to modes continuously used with the 8085.



Pin Names

$D_7 - D_0$	Data Bus (Bidirectional)
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A_0	Command Select Address
$\overline{CS}$	Chip Select
$CAS_2 - CAS_0$	Cascade Lines
$\overline{SP}$ / $\overline{EN}$	Slave Program/Enable Buffer
INT	Interrupt Output
$\overline{INTA}$	Interrupt Acknowledge Input
$IR_0 - IR_7$	Interrupt Request Input

Figure 5.12: Block diagram of 8259A programmable interrupt controller

The figure shows the internal block diagram of the 8259A. It includes eight blocks: control logic, Read/Write logic, data bus buffer, three registers (IRR, ISR and IMR), priority resolver, and cascade buffer. This diagram shows all the elements of a programmable device, plus additional blocks. The functions of some of these blocks need explanation, which is given below:

READ/WRITE LOGIC

This is a typical Read/Write control logic. When the address line A_0 is at logic 0, the controller is selected to write a command or read a status. The Chip Select logic and A_0 determine the port address of the controller.

CONTROL LOGIC

This block has two pints: INT (Interrupt) as an output, and INTA (Interrupt Acknowledge) as an input. The INT is connected to the interrupt pin of the MPU. Whenever a valid interrupt is asserted, this signal goes high. The INTA is the Interrupt Acknowledge signal from the MPU.

INTERRUPT REGISTERS AND PRIORITY RESOLVER

The interrupt Request Register (IRR) has eight input lines (IR_0-IR_7) for the interrupts. When these lines go high, the requests are stored in the register. The In-Service Register (ISR) stores all the levels that are currently being serviced, and the Interrupt Mask Register (IMR) stores the masking bits of the interrupt lines to be masked. The Priority Resolver (PR) examines these three registers and determines whether INT should be sent to the MPU.

CASCADE BUFFER/COMPARATOR

This block is used to expand the number of interrupt levels by cascading two or more 8259As.

Interrupt Operation

To implement interrupts, the Interrupt Enable flip-flop in the microprocessor should be enabled by writing the EI instruction, and the 8259A should be initialized by writing control words in the control register. The 8259A requires two types of control words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). The ICWs are used to set up the proper conditions and specify RST vector addresses. The OCWs are used to perform functions such as masking interrupts, setting up status-read operations, etc. After the 8259A is initialized, the following sequence of events occur when one or more interrupt request lines go high:

1. The IRR stores the requests.
2. The priority resolver checks three registers: the IRR for interrupt requests, the IMR for masking bits, and the ISR for the interrupt request being served. It resolves the priority and sets the INT high when appropriate.
3. The MPU acknowledges the interrupt by sending INTA.
4. After the INTA is received, the appropriate priority bit in the ISR is set to indicate which interrupt level is being served, and the corresponding bit in the IRR is reset to indicate that the request is accepted. Then, the opcode for the CALL instruction is placed on the data bus.
5. When the MPU decodes the CALL instruction, it places two or more INTA signals on the data bus.
6. When 8259A receives the second INTA, it places the low-order byte of the CALL address on the data bus. At the third INTA, it places the high-order byte on the data bus. The CALL address is the vector memory location for the interrupt: this address is placed in the control register during the initialization.

7. During the third INTA pulse, the ISR bit is reset either automatically (Automatic-End-of-Interrupt—AEOI) or by a command word that must be issued at the end of the service routine (End-of-Interrupt—EOI). This option is determined by the initialization command word (ICW).
8. The program sequence is transferred to the memory location specified by the CALL instruction.

Priority Modes and Other Features

Many types of priority modes are available under software control in the 8259A, and they can be changed dynamically during the program by writing appropriate command words. Commonly used priority modes are discussed below:

1. **Fully Nested Mode:** This is a general-purpose mode in which all IRS (Interrupt Requests) are arranged from highest to lowest, with IR_0 as the highest and IR_7 as the lowest.

   In addition, any IR can be assigned the highest priority in this mode; the priority sequence will then begin at that IR. In the example below, IR_4 has the highest priority, and IR_3 has the lowest priority:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_2	IR_3	IR_4
4	5	6	7	0	1	2	3

2. **Automatic Rotation Mode:** In this mode, a device, after being serviced, receives the lowest priority. Assuming that the IR_2 has just been serviced, it will receive the seventh priority, as shown below:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_2	IR_3	IR_4
1	6	7	0	1	2	3	4

3. **Specific Rotation Mode:** This mode is similar to the automatic rotation mode, except that the user can select any IR for the lowest priority, thus fixing all other priorities.

End of Interrupt

After the completion of an interrupt service, the corresponding ISR bit needs to be reset to update the information in the ISR. This is called the End-of-Interrupt (EOI) command. It can be issued in three formats.

1. **Nonspecific EOI Command:** When this command is sent to 8259A, it resets the highest priority ISR bit.
2. **Specific EOI Command:** This command specifies which ISR bit to reset.
3. **Automatic EOI:** In this mode, no command is necessary. During the third INTA, the ISR bit is reset. The major drawback with this mode is that the ISR does not have information on which IR is being serviced. Thus, any IR can interrupt the service routine irrespective of its priority, if the Interrupt Enable flip-flop is set.

Additional Features of the 8259A

The 8259A is a complex device with various modes of operation. These modes are listed below (for reference; the user should refer to the 8085 User's manual for details.)

- **Interrupt Triggering:** The 8259A can accept an interrupt request with either the edge triggered mode or the level-triggered mode. This mode is determined by the initialization instructions.
- **Interrupt Status:** The status of the three instruction registers (IRR, ISR and IMR) can be read, and this status information can be used to make the interrupt process versatile.
- **Poll Method:** The 8259A can be set up to function in a polled environment. The MPU polls the 8259A rather than each peripheral.

QUESTIONS

1. Differentiate between memory mapped I/O and Input Output mapped I/O.
2. Discuss DMA with the help of its advantages and applications.
3. What is cycle stealing? Explain DMA operation with the help of its neat block diagram and timing diagram.
4. Draw a neat diagram of 8237 DMA controller and explain it.
5. What is interrupt? Explain the steps of basic interrupt processing.
6. Why interrupt is required? Explain external interrupt.
7. Differentiate between internal interrupt and software interrupt.
8. Compare and contrast between polled interrupt and chain interrupt.
9. Explain 8085 interrupt pins.
10. Explain 8259 PIC with the help of its block diagram.

❑❑❑

6

I/O INTERFACE

CHAPTER OBJECTIVE

This chapter introduces the concept of input-output interface using serial and parallel communication technology. To explain serial communication interfacing the 8251 USART is introduced, describing its asynchronous and synchronous modes of operation. Similarly, the parallel communication interfacing is explained by using 8255 Programmable Peripheral Interface with the help of its block diagram, modes of operation and control word. Finally, the RS-232 serial interface is introduced showing the interconnection between DTE-DCE and DCE-DTE.

CHAPTER OUTLINE

- Parallel Communication – Introduction and Applications
- Serial Communication
  - Introduction and Applications
  - Introduction to Programmable Communication Interface 8251
  - Basic Concept of Synchronous and Asynchronous Modes.
- Simple I/O, Strobe I/O, Single handshake I/O, Double handshake I/O
- 8255A and it's Working
  - Block Diagram
  - Modes of Operation
  - Control Word
- RS-232 – Introduction, Pin Configuration (9 pin and 25 pin) and function of each pin, Interconnection between DTE-DTE and DTE-DCE

6.1 Parallel and Serial Communication

Data transmission refers to the process of transferring data between two or more digital devices. Data is transmitted from one device to another in analog or digital format. Basically, data transmission enables devices or components within devices to speak to each other.

Data is transferred in the form of bits between two or more digital devices. There are two methods used to transmit data between digital devices: serial transmission and parallel transmission. Serial data transmission sends data bits one after another over a single channel. Parallel data transmission sends multiple data bits at the same time over multiple channels.

Serial transmission is normally used for long-distance data transfer. It is also used in cases where the amount of data being sent is relatively small. It ensures that data integrity is maintained as it transmits the data bits in a specific order, one after another. In this way, data bits are received in-sync with one another.

Serial Communication

Introduction

Serial communication is common method of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. Serial communication transmits data one bit at a time, sequentially, over a single communication line to a receiver. Serial is also a most popular communication protocol that is used by many devices for instrumentation. This method is used when data transfer rates are very low or the data must be transferred over long distances and also where the cost of cable and synchronization difficulties makes parallel communication impractical. Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect the instrument to the computer or two computers together.

Serial and Parallel Transmission

Let us now try to have a comparative study on parallel and serial communications to understand the differences and advantages & disadvantages of both in detail.

We know that parallel ports are typically used to connect a PC to a printer and are rarely used for other connections. A parallel port sends and receives data eight bits at a time over eight separate wires or lines. This allows data to be transferred very quickly. However, the setup looks more bulky because of the number of individual wires it must contain. But, in the case of a serial communication, as stated earlier, a serial port sends and receives data, one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. Although this is slower than parallel communication, which allows the transmission of an entire byte at once, it is simpler and can be used over longer distances. So, at first sight it would seem that a serial link must be inferior to a parallel one, because it can transmit less data on each clock tick. However, it is often the case that, in modern technology, serial links can be clocked considerably faster than parallel links, and achieves a higher data rate.

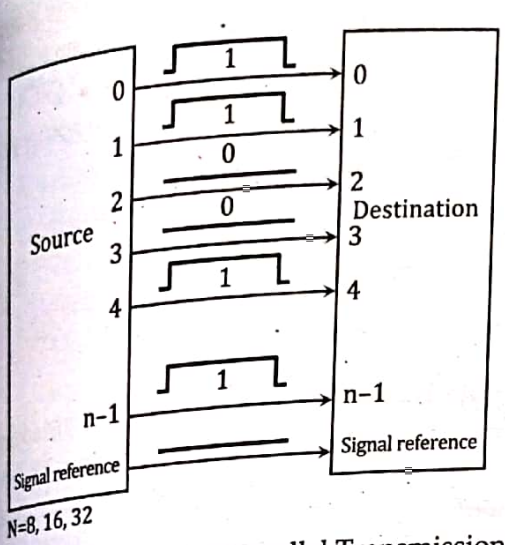


Figure 6.1: Parallel Transmission

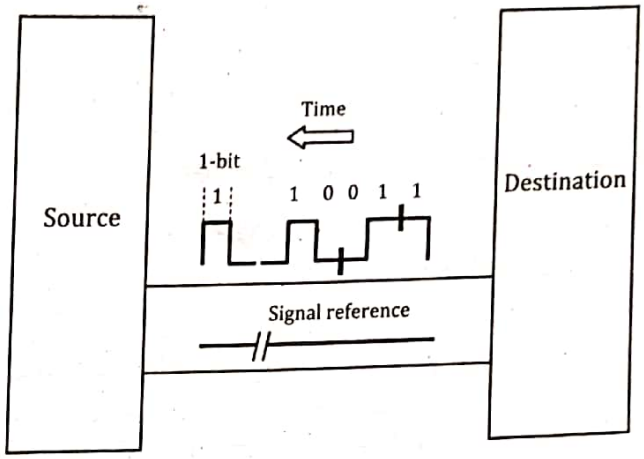


Figure 6.2: Serial Transmission

Even in shorter distance communications, serial computer buses are becoming more common because of a tipping point where the disadvantages of parallel buses (clock skew, interconnect density) outweigh their advantage of simplicity. The serial port on your PC is a full duplex device meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data.

From the above discussion we could understand that serial communications have many advantages over parallel one like:

- Requires fewer interconnecting cables and hence occupies less space.
- "Cross talk" is less of an issue, because there are fewer conductors compared to that of parallel communication cables.
- Many ICs and peripheral devices have serial interfaces.
- Clock skew between different channels is not an issue.
- Cheaper to implement.

Clock Skew

Clock skew is a phenomenon in synchronous circuits in which the clock signal sent from the clock circuit arrives at different components at different times, which can be caused by many things, like:

- Wire-interconnect length
- Temperature variations
- Variation in intermediate devices
- capacitive coupling
- Material imperfections

Serial transmission is normally used for long-distance data transfer. It is also used in cases where the amount of data being sent is relatively small. It ensures that data integrity is maintained as it transmits the data bits in a specific order, one after another. In this way, data bits are received in-sync with one another.

Advantages and Disadvantages of Using Parallel Data Transmission

The main advantages of parallel transmission over serial transmission are:

- It is easier to program;
- and data is sent faster.

Although parallel transmission can transfer data faster, it requires more transmission channels than serial transmission. This means that data bits can be out of sync, depending on transfer distance and how fast each bit loads. A simple of example of where this can be seen is with a voice over IP (VOIP) call when distortion or interference is noticeable. It can also be seen when there is skipping or interference on a video stream.

Parallel transmission is used when

- a large amount of data is being sent;
- the data being sent is time-sensitive;
- and the data needs to be sent quickly.

A scenario where parallel transmission is used to send data is video streaming. When a video is streamed to a viewer, bits need to be received quickly to prevent a video pausing or buffering. Video streaming also requires the transmission of large volumes of data. The data being sent is also time-sensitive as slow data streams result in poor viewer experience.

Serial Data Transmission Modes

When data is transmitted between two pieces of equipment, three communication modes of operation can be used.

Simplex: In a simple connection, data is transmitted in one direction only. For example, from a computer to printer that cannot send status signals back to the computer.

Half-duplex: In a half-duplex connection, two-way transfer of data is possible, but only in one direction at a time.

Full duplex: In a full-duplex configuration, both ends can send and receive data simultaneously, which technique is common in our PCs.

Serial Data Transfer Schemes

Like any data transfer methods, Serial Communication also requires coordination between the sender and receiver. For example, when to start the transmission and when to end it, when one particular bit or byte ends and another begins, when the receiver's capacity has been exceeded, and so on. Here comes the need for synchronization between the sender and the receiver. A protocol defines the specific methods of coordinating transmission between a sender and receiver. For example a serial data signal between two PCs must have individual bits and bytes that the receiving PC can distinguish. If it doesn't, then the receiving PC can't tell where one byte ends and the next one begin or where one bit ends and begins. So the signal must be synchronized in such a way that the receiver can distinguish the bits and bytes as the transmitter intends them to be distinguished.

There are two ways to synchronize the two ends of the communication.

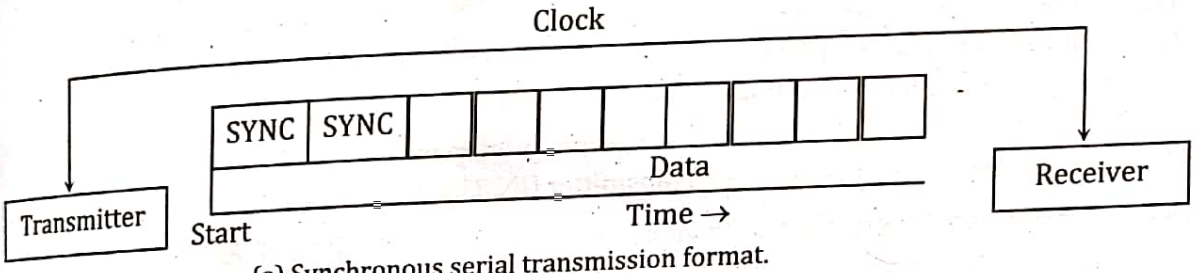
1. Synchronous data transmission
2. Asynchronous data transmission

Synchronous Data Transmission

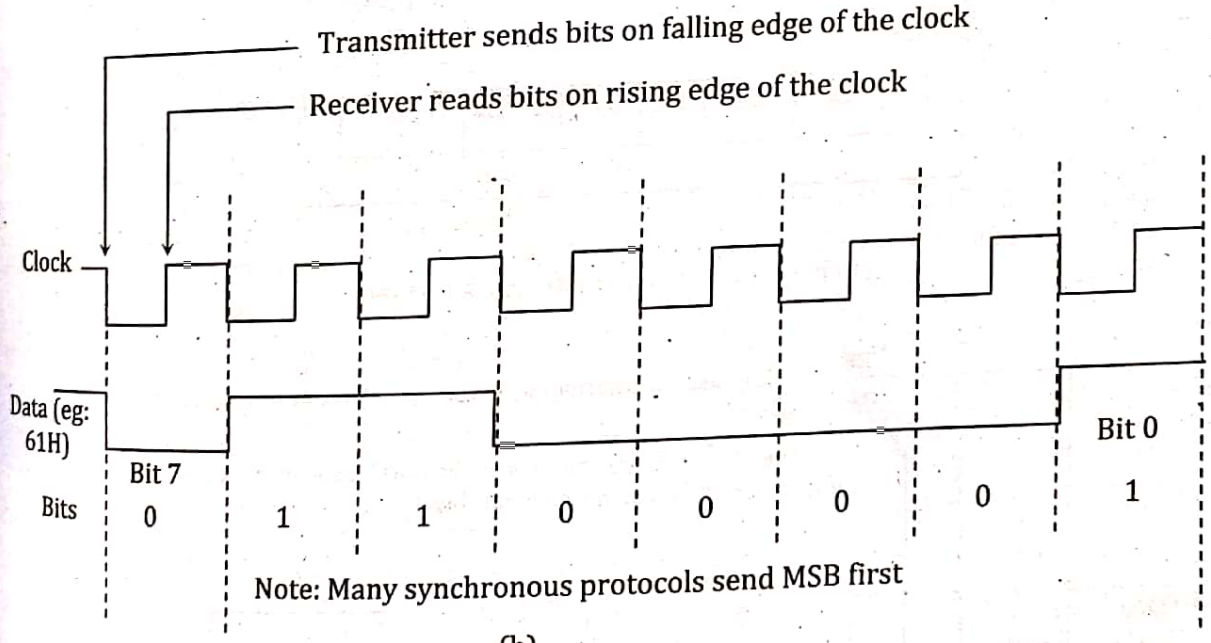
The synchronous signaling methods use two different signals. A pulse on one signal line indicates when another bit of information is ready on the other signal line.

In synchronous transmission, the stream of data to be transferred is encoded and sent on one line, and a periodic pulse of voltage which is often called the "clock" is put on another line, that tells the receiver about the beginning and the ending of each bit.

- A more efficient method of transferring serial data is to synchronize the transmitter and the receiver and then send a large block of data characters one after the other with no time between characters.
- No start or stop bits are needed with individual data characters because the receiver automatically knows that every 8 bits received after synchronization represents a data character.
- To indicate start of transmission, transmitter sends out one or more unique characters called sync. Characters.
- The receiver uses the sync characters or the flag to synchronize its internal clock with that of the receiver.
- ISDN, High Speed Modems and Digital Communication Channel use synchronous transmission.



(a) Synchronous serial transmission format.



(b)

Figure 6.3: Synchronous transmission

Advantages: The only advantage of synchronous data transfer is the Lower overhead and thus, greater throughput, compared to asynchronous one.

Disadvantages

- Slightly more complex
- Hardware is more expensive

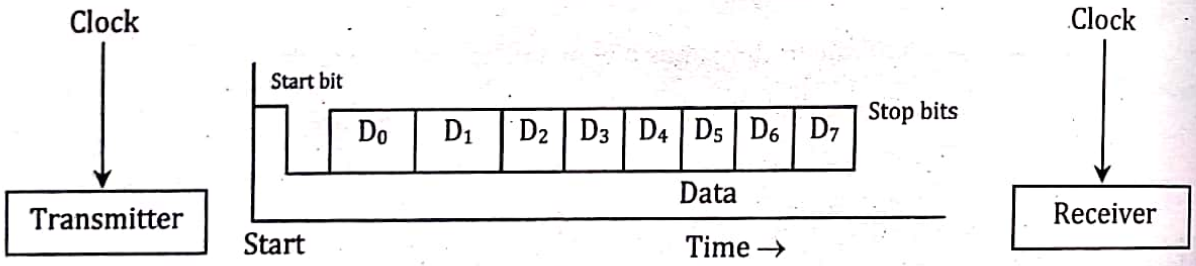
Asynchronous Data Transmission

The asynchronous signaling methods uses only one signal. The receiver uses transitions on that signal to figure out the transmitter bit rate (known as auto baud) and timing. A pulse from the local clock indicates when another bit is ready. That means synchronous transmissions use an external clock, while asynchronous transmissions uses special signals along the transmission medium. Asynchronous communication is the commonly prevailing communication method in the personal computer industry, due to the reason that it is easier to implement and has the unique advantage that bytes can be sent whenever they are ready, no need to wait for blocks of data to accumulate.

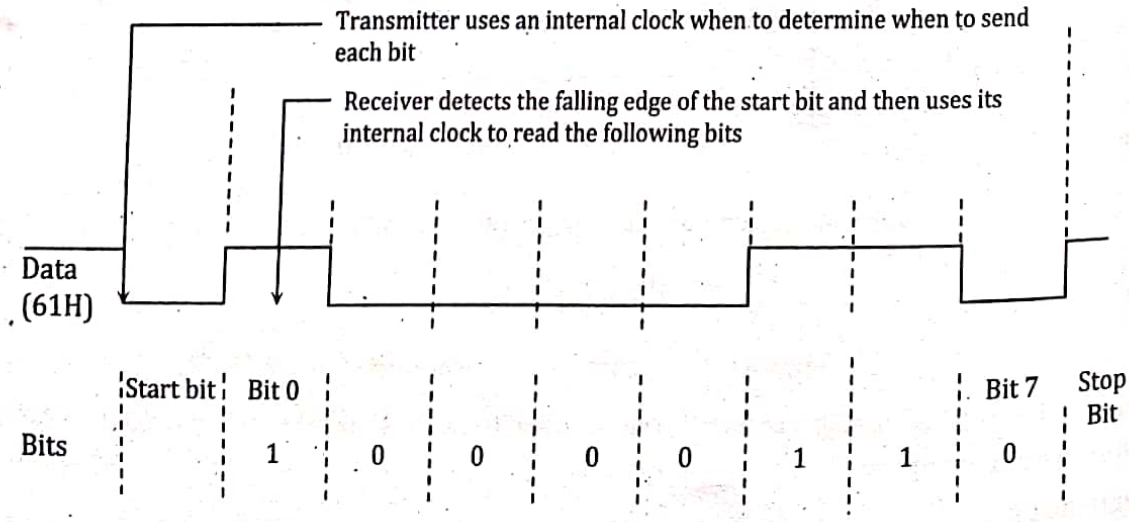
- For asynchronous transmission, each data character has a bit which identifies its start and 1 or 2 bits which identify end.
- Since each character is individually identified, character can be sent at any time asynchronously.
- The beginning of a data character is identified by the line going low for 1 bit time. This bit is called start bit.
- The data bits are then sent out on the line one after the other.
- Note that LSB is transmitted first.
  - After parity bit, the signal line is returned high for at least 1 bit time to identify the end of the character. This high bit is always referred as stop bit.

UART: Universal Asynchronous Receiver Transmitter [IN 8250]

USART: Universal Synchronous Asynchronous Receiver Transmitter [INTEL 8251A]



(a) Asynchronous serial transmission format.



Note: Asynchronous protocols send LSB first

(b)

Figure 6.4: Asyndronous Transmission

Advantages

- Simple and doesn't require much synchronization on both communication sides.
- The timing is not as critical as for synchronous transmission; therefore hardware can be made cheaper.
- Set-up is very fast, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard.

Disadvantages

One of the main disadvantages of asynchronous technique is the large relative overhead, where a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information.

Key Differences between Serial and Parallel Transmission

- Serial transmission requires a single line to communicate and transfer data whereas, parallel transmission requires multiple lines.
- Serial transmission used for long distance communication whereas, the parallel transmission used for shorter distance.
- Error and noise are least in serial as compared to parallel transmission. Since one bit follows another in Serial Transmission whereas, in Parallel Transmission multiple bits are sent together.
- Parallel transmission is faster as the data is transmitted using multiples lines whereas, in Serial transmission data flows through a single wire.
- Serial Transmission is full duplex as the sender can send as well as receive the data whereas, Parallel Transmission is half duplex since the data is either sent or received.
- Serial transmission cables are thinner, longer and economical in comparison with the Parallel Transmission cables.
- Serial Transmission is reliable and straightforward whereas, Parallel Transmission is unreliable and complicated.

Conclusion

Both Serial and Parallel Transmission have their advantages and disadvantages respectively. Parallel Transmission is used for shorter distance, provides greater speed. On the other hand, Serial Transmission is reliable for transferring data to longer distance. Hence, we conclude that both serial and parallel are individually essential for transferring data.

8251 Universal Synchronous Asynchronous Receiver Transmitter

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

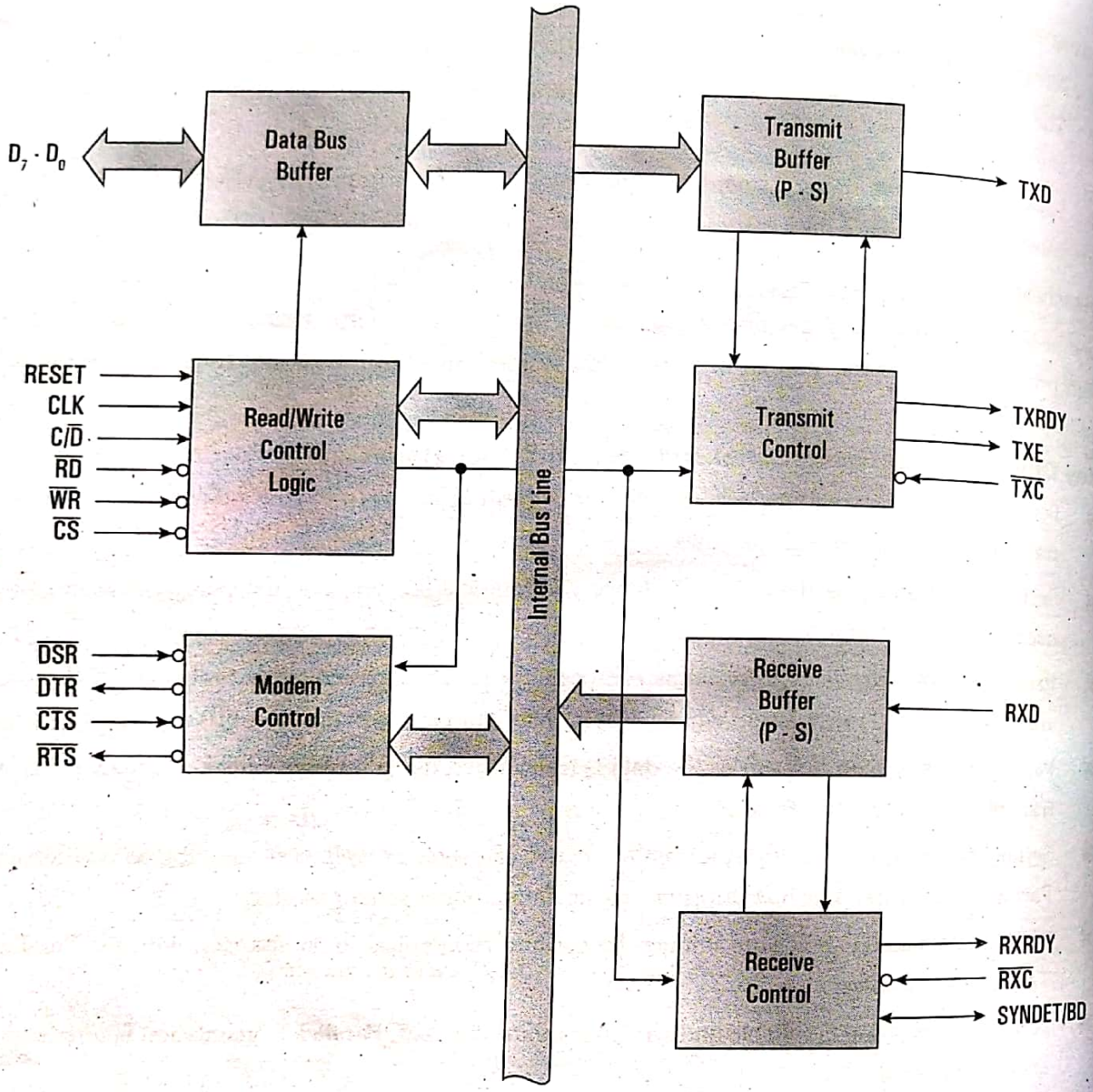


Figure 6.5: Block diagram of 8251 USART

The 8251 functional configuration is programmed by software. Operation between the 8251 and a CPU is executed by program control. Table 1 shows the operation between a CPU and the device.

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	Data Bus 3-State
0	×	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

Table 6.1: Control signal for read and write operation

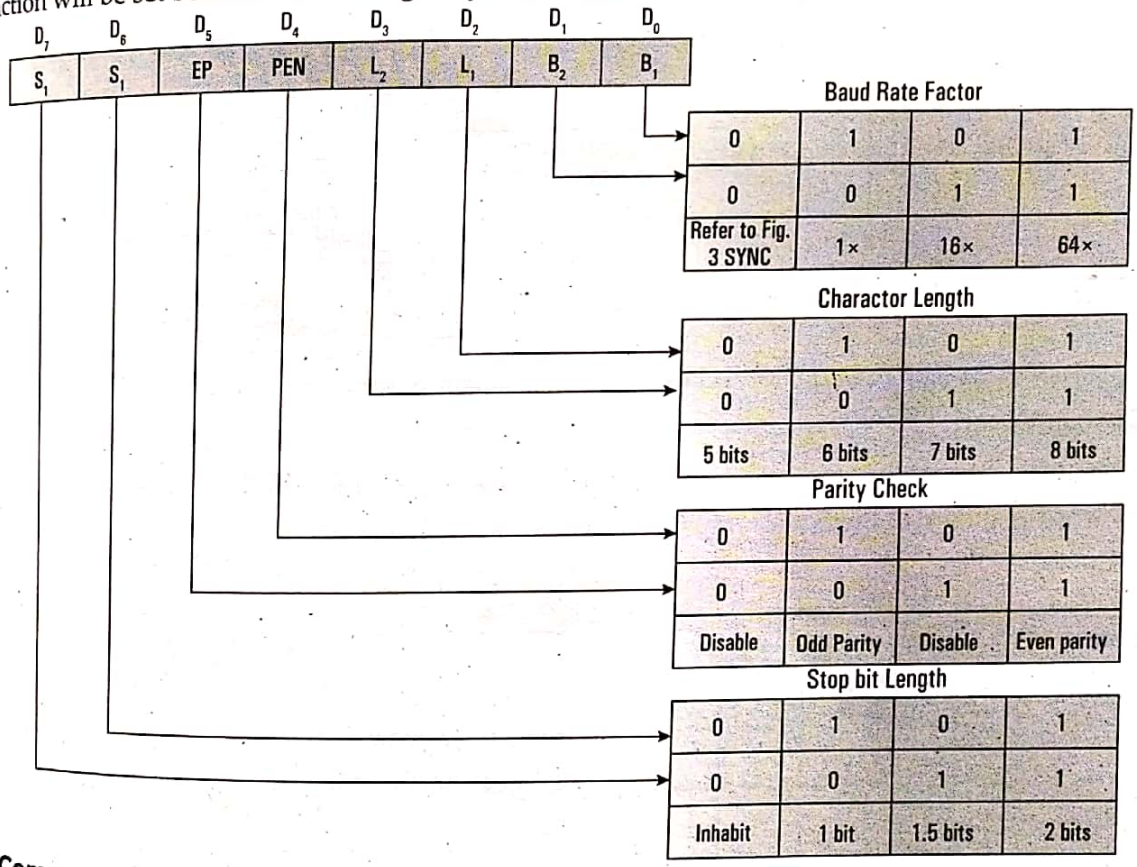
Control Words

1. Mode Instruction

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

- Synchronous/ asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/ external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction is shown in Figures 2 and 3. In the case of synchronous m ode, it is necessary to write one or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

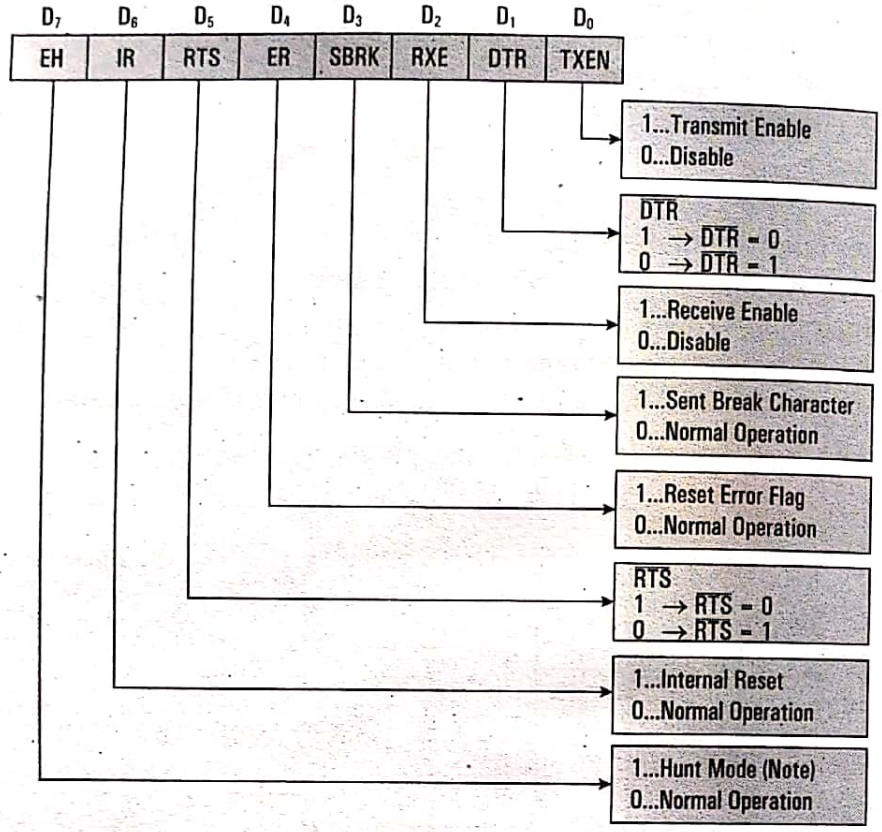


2. Command Word

Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters. Items to be set by command are as follows:

- Transmit Enable/ Disable
- Receive Enable/ Disable
- DTR, RTS Output of data
- Resetting of error flag

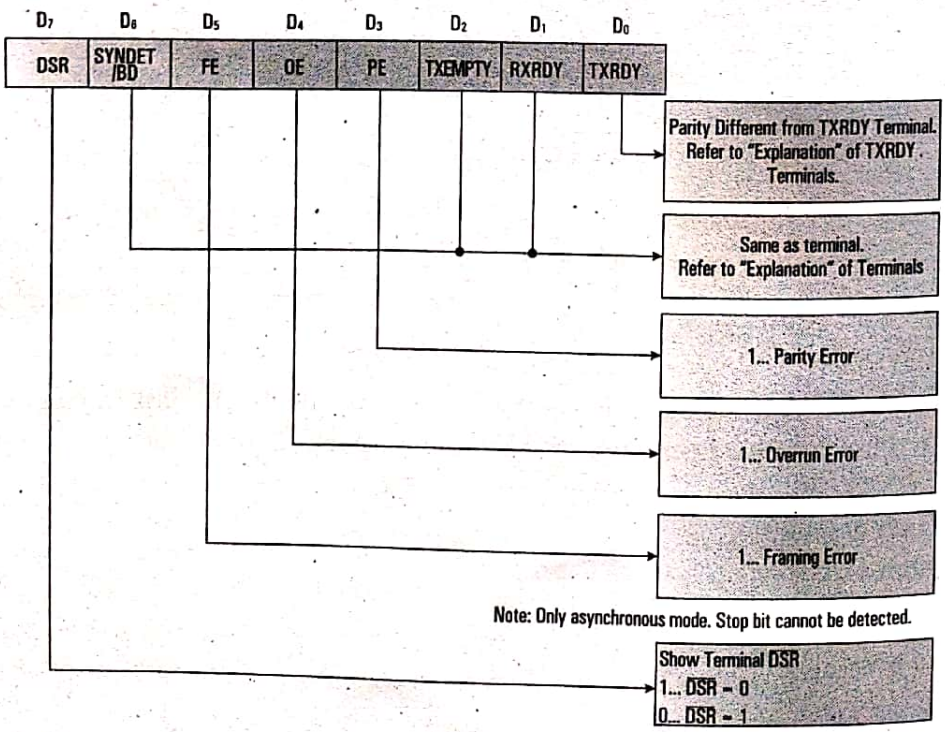
- ⌘ Sending to break characters
- ⌘ Internal resetting
- ⌘ Hunt mode (synchronous mode)



Note: Seach mode for synchronous charactors in synchronous mode.

3. Status Word

It is possible to see the internal status of the 8251 by reading a status word. The bit configuration of status word is shown in Fig 5.



Pin Description

D_0 to D_7 [I/O Terminal]

This is bidirectional data bus which receives control words and transmits its data from the CPU and sends status words and received data to CPU.

RESET [Input Terminal]

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

CLK [Input Terminal]

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

WR [Input Terminal]

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

RD [Input Terminal]

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

C/D [Input Terminal]

This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU.

If C/D=low, data will be accessed. If C/D=high, command word or status word will be accessed.

CS [Input Terminal]

This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses.

Note: *The device won't be in "standby status"; only setting CS = High.*

TXD [Output Terminal]

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

TXRDY [Output Terminal]

This is an output terminal which indicates that the 8251 is ready to accept a transmitted data character. But the terminal is always at low level if CTS=high or the device was set in "TX status" by a command

Note: *TXRDY status word indicates disable and that transmit data character is receivable of CTS or command.*

TXEMPTY [Output Terminal]

This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode" the terminal is at high level, if transmitted data are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note: As the transmitter is disabled by setting CTS "High" or com mand, data written before disable will be sent out. Then TXD and

TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High". After the transmitter is enabled, it sent out.

TXC [Input Terminal]

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode inst ruction. It can be 1, 1/ 16 or 1/ 64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD [Input Terminal]

This is a terminal which receives serial data.

RXRDY [Output Terminal]

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

RXC [Input Terminal]

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

SYNDET/ BD [Input or Output Terminal]

This is a terminal whose function changes according to mode. In "internal synchronous mode", this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode", this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates "high level" output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level.

DSR [Input Terminal]

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

DTR [Output Terminal]

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

CTS [Input Terminal]

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

RTS [Output Terminal]

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

Programming 8251A

- ⌘ According to the datasheet, the 8251A requires a worst case recovery time of 16 cycles.
- ⌘ The initialization sequence of 8251A is hence lengthy.
- ⌘ The second reason is that, the 8251A does not respond correctly to a hardware reset.

- This means that you have to delay 16 processor clock cycles on power up.
- Therefore a series of software command must be sent to the device to the 8251A distinguishes a command word from a mode word by the order in which make sure it reset properly before the desired mode and command word are sent.
- They are sent to the device
- Any words sent to the command address after the mode word will be treated as command word until the device is reset.

Techniques

A simple way to produce the required delay and a margin of safety is to load CX with 2 and count it down with the loop instruction

LOOP First > 1 Clock Cycles
LOOP Last > 5 Clock Cycles

```
MOV DX, 0FFF2H ; Command Register Address of 8251A
MOV AL,00H
OUT DX, AL
;
MOV CX, 2     ; 4 Clock Cycles
D0: LOOP DO  ; 17+ 5 Clock Cycles
OUT DX, AL   ; 8 Clock Cycles
;
MOV CX,2      ; 4 Clock Cycles
D1: LOOP DO  ; 17+ 5 Clock Cycles
OUT DX,AL    ; 8 Clock Cycles

MOV AL,40 ; Send InternalReset OUT
DX,AL

MOV CX,2
D2: LOOP D2

;

MOV AL,11001110 b ; Load Mode Word
OUT DX,AL

MOV CX,2
D2: LOOP D2

MOV AL,00110111 b ; Load Command Word
OUT DX,AL
```

6.2 Methods of Parallel Data Transfer

1. Simple I/O

- When you need to get digital data from a simple switch, all you have to do is connect the switch to an I/P port line and read the value.
- Likewise, when you need to O/P data to simple LED, all you have to do is connect the LED to an O/P port and send the value.
- The LED is always ready, so you can send data at any time.

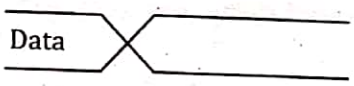


Figure 6.6: Timing diagram of simple I/O

2. Strobe I/O

- In many applications, valid data is present on an external device only at a certain time, so it must be read in at that time.
- When a key is pressed, circuitry on the keyboard sends out the ASCII code for the pressed key on eight parallel data line, and then sends out a strobe signal on another line to indicate that valid data is present on the eight data lines.
- For higher speed data transfer this method does not work.
- The sending system might send data bytes faster than the receiving system could read them. To prevent this handshake data transfer is required.

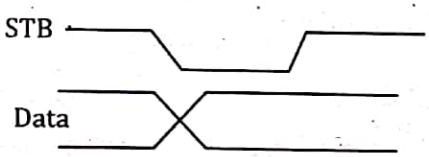


Figure 6.7: Timing diagram of strobe I/O

3. Single Handshake I/O Data Transfer

- The peripheral outputs some parallel data and sends STB signal to MPU.
- The MPU detects STB signal on a polled or interrupt basis and reads data byte.
- Then the MP sends on ACK signal to the peripheral to indicate that the data has been read and the peripheral can send to next byte of data.

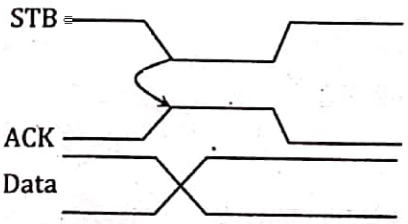


Figure 6.8: Timing diagram of single handshake I/O

4. Double Handshake I/O Data Transfer

- For data transfers where coordination is required between sending system and the receiving system, a double handshake is used.
- The sending device asserts its STB low to ask ,**Are you ready?"**

- The receiving system raises its ACK line high to say ,**I'm ready".**
- The peripheral device then sends the data byte and raises its STB signal high to say **"Here is valid data for you".**
- When the receiving system finishes to read the data, receiving system drop its ACK line low to say **"I have data than you and I await your request to send the next byte of data".**

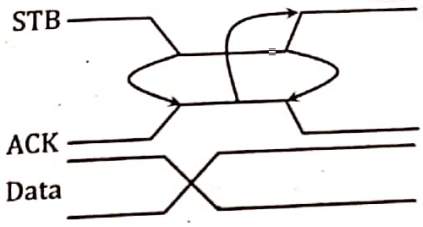


Figure 6.9: Timing diagram of double handshake I/O

6.3 8255A Programmable Peripheral Interface

Introduction

The 8255A PPI provides three 8 bit input/output ports in one 40 pin package. The chip can be interfaced directly to the data bus of the processor allowing its function to be programmed. In one application, a port may appear as an output, but in another by reprogramming it an be as input.

Description

- The 8255A is a general purpose parallel I/O interfacing device.
- It provides 24 I/O lines organized as three 8 bit input/output ports labeled A,B and C.
- Each of the ports A or B can be programmed as an 8 bit input or output port.
- Port C can be divided in half, with the topmost or bottommost four bits programmed as inputs or outputs.
- Port C can be used as an 8 bit input or output port as two 4 bit ports or to produce handshake signals for Port A and B.
- The 8255A is a very versatile device.
- It can be programmed to look like
  - Three simple I/O ports (called MODE 0)
  - Two handshaking I/O ports (called MODE 1)
  - Port A as a bidirectional I/O port with 5 handshaking signals (called MODE 2) The modes can also be intermixed.
  - For example, Port A can be programmed to operate in MODE 2 while Port B in MODE 0.

Internal Block Diagram of 8255a

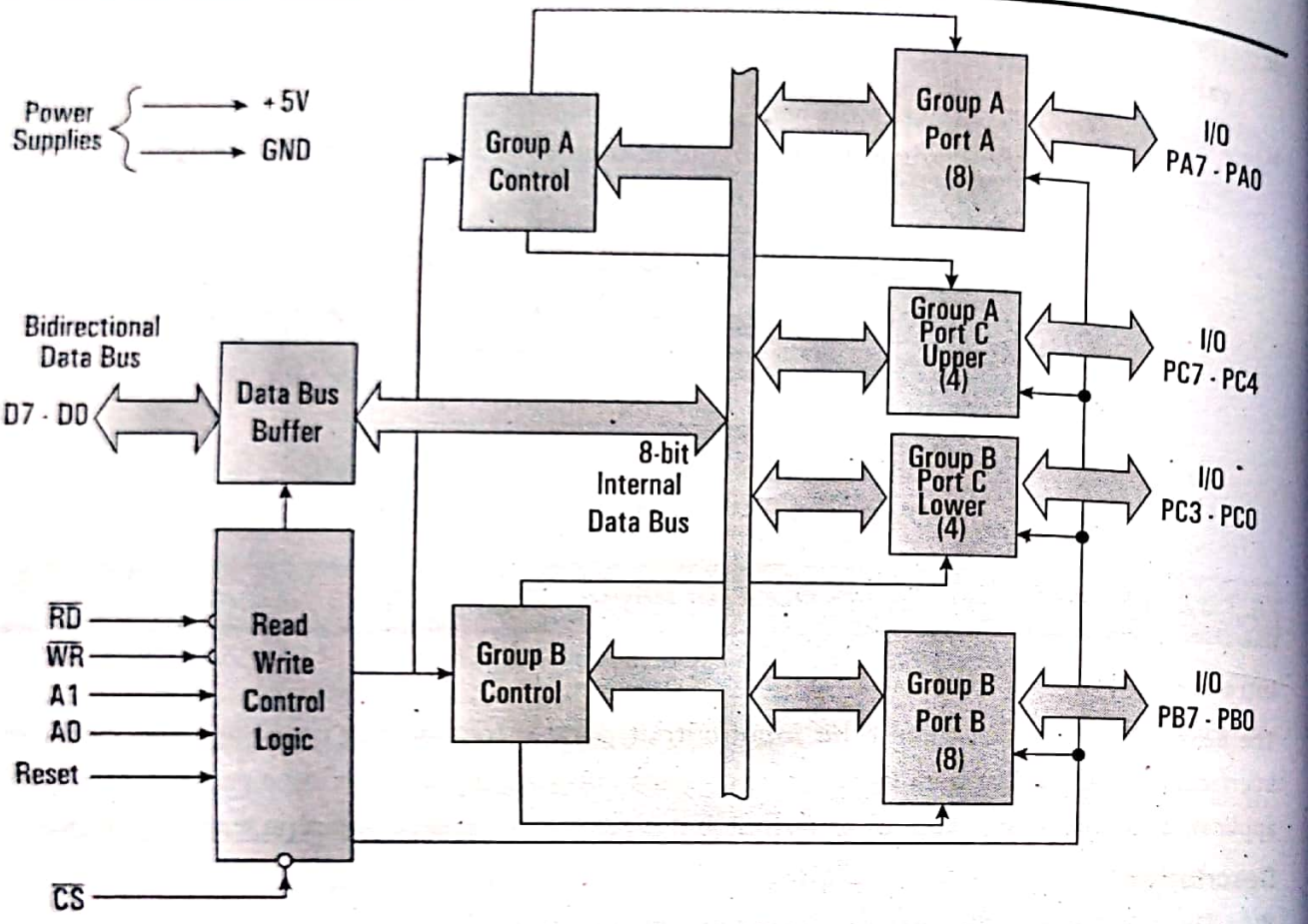


Figure 6.10: Block diagram of 8255 PPI

- The address inputs A0, A1 allow you to selectively access one of the three ports or the control register.
- The internal addresses for the devices are tabulated below

Description	A1	A0
Port A	0	0
Port B	0	1
Port C	1	0
Control	1	1

- The CS input of 8255A enables it for reading or writing
- This input is driven by an address decoder.
- The RD and WR input pins determine the direction of data flow over the chips 8 bit bidirectional data bus.
- The RESET input of 8255A is connected to the system reset line so that, when the system is reset all the port lines are initialized as input lines. This is done to prevent destruction of circuitry connected to port lines.

Pin Configuration of 8255 PPI

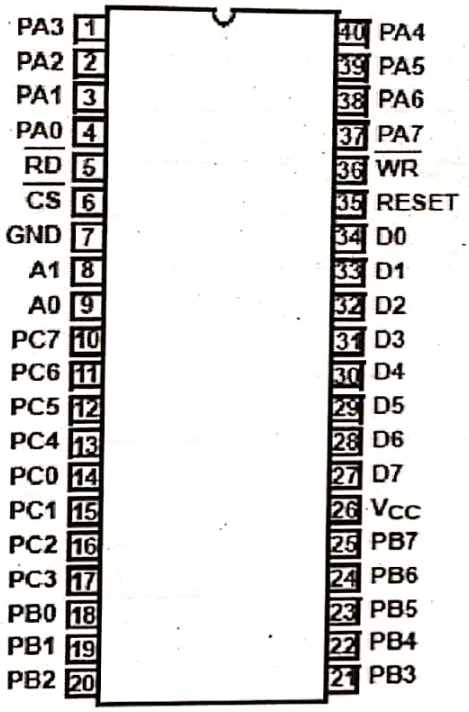


Figure 6.11: Pin Configuration of 82C55A PPI

Pin Names with Descriptions

D7-D0	DATA BUS [BIDIRECTIONAL]
RESET	RESET INPUT
CS	CHIP SELECT
RD	READ INPUT
WR	WRITE INPUT
A0,A1	PORT ADDRESS
PA7-PA0	PORT A
PB7-PB0	PORT B
PC7-PC0	PORT C
VCC	+5V
GND	GROUND

Truth Table for the 8255A PPI

1. Input Operation

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION
0	0	0	1	0	PORT A->DATA BUS
0	1	0	1	0	PORT B->DATA BUS
1	0	0	1	0	PORT C->DATA BUS

2. OUTPUT OPERATION

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION
0	0	1	0	0	DATA BUS->PORT A
0	1	1	0	0	DATA BUS->PORT B
1	0	1	0	0	DATA BUS->PORT C
1	1	1	0	0	DATA BUS ->CONTROL

3. DISABLEFUNCTION

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION
X	X	X	X	1	DATA BUS TRISTATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS TRISTATE

Operational Modes

1. Mode 0

- When programmed for MODE 0, the PPI offers three simple I/O ports with no handshaking signals.
- This mode is appropriate for I/O devices that do not need special synchronizing signals to exchange data with the processor.
- A common example is a keyboard used for data entry.
- When used as O/Ps, the PORT c lines can be individually set or reset by sending a special control word to control register address.
- Two halves of PORT C are independent so one half can be initialized as input and the other half as output.

2. Mode 1

- When programmed for MODE 1, the PPI offers PORT A or PORT B for a handshake input/output operation.
- Pins PC0,PC1 and PC2 function as handshake lines for PORT B
- Pins PC3,PC4 and PC5 function as handshake signal for PORT A (input).
- Pins PC6 and PC7 are available for use as input/output lines for PORT A
- If PORT A is initialized as handshake O/P port, then PORT C pins.
- PC3, PC6 and PC7 function as handshake signals.
- PORT C pins PC4 and PC5 are used as input/output lines for PORT A

3. Mode 2

- Only PORT A can be initialized in MODE 2.
- In MODE 2, PORT A can be used as bidirectional handshake data transfer.

- This means that data can be outputted/inputted from same eight lines.
- If PORT A is initialized in MODE 2, then pins PC3 through PC7 are used as handshake lines for PORT A.
- The other three pins PC0 through PC2 can be used for I/O port if PORT B is in MODE 0.
- The same 3 pins will be used for PORT B handshake signals if PORT B is initialized in MODE 1.

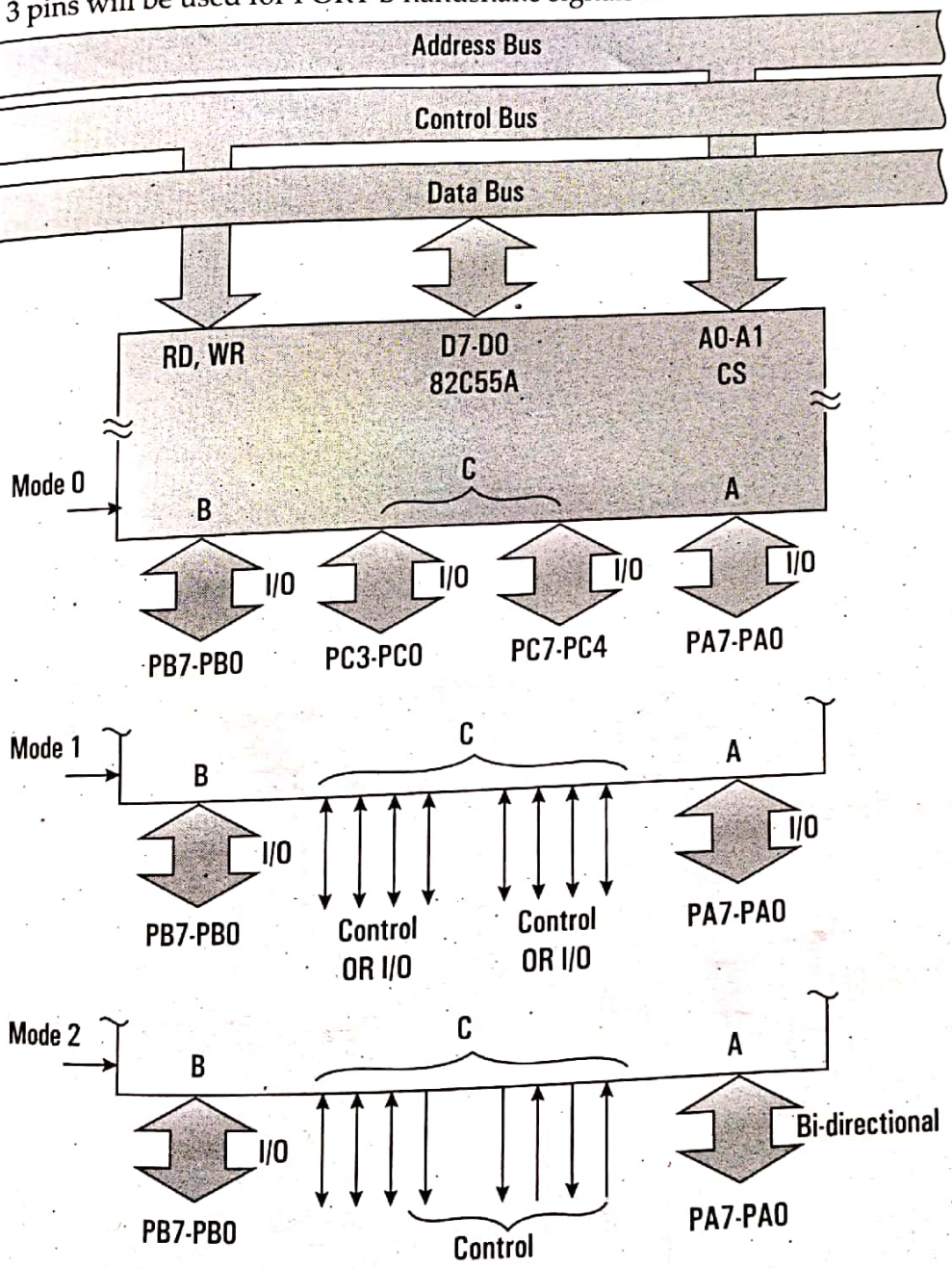


Figure 6.12: Operational modes of 8255 PPI

Control Words

Mode Set Control Word

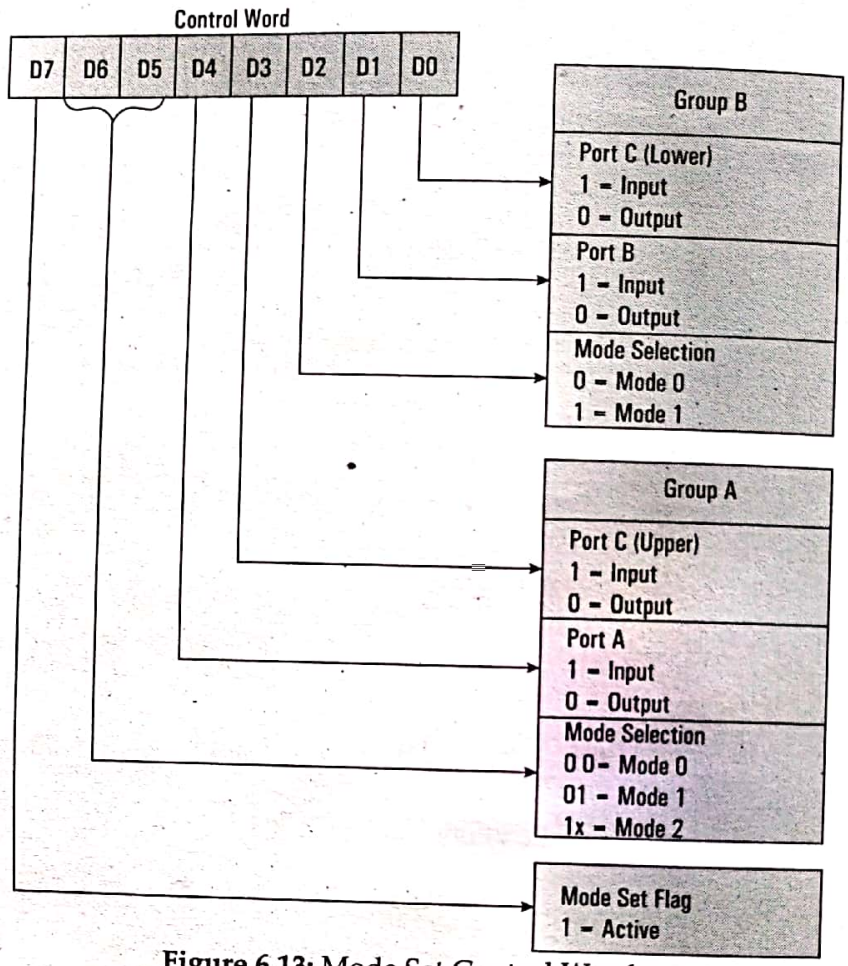


Figure 6.13: Mode Set Control Word

Port C Bit Set/Reset Control Word

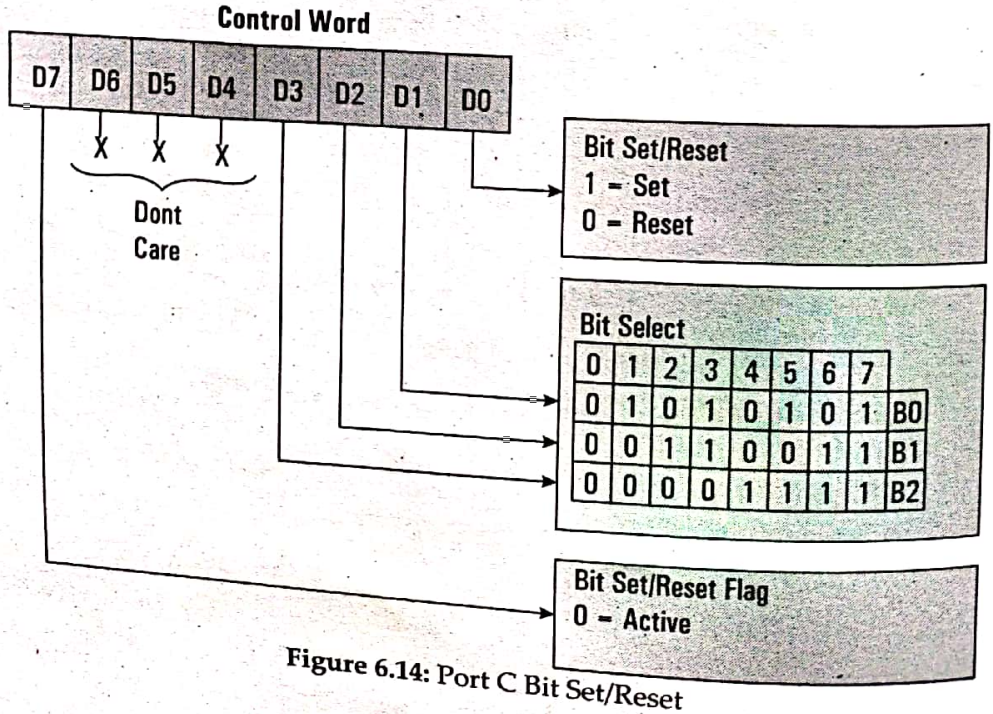


Figure 6.14: Port C Bit Set/Reset

PROBLEM 1

Write the 80x86 initialization routine required to program the 8255A for mode 0, with PORT A as an output port and PORT B and C as an input ports.

SOLUTION

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

D7 : 1-> Mode set
D6 D5 : 00 -> Mode 0
D4 : 0 -> Port A Output
D3 : 1 -> Port C Upper Input
D2 : 0 -> Mode 0
D1 : 1 -> Port B Input
D0 : 0 -> Port C Lower Input

Assume the port address of control port is 0FFH

```
MOV AL,8BH
OUT 0FFH,AL
```

PROBLEM 2

Write an 80x86 program to input a byte from PORT B of PPI chip and output this byte to PORT A of the same chip. Assume it is already initialized.

Solution

```
Assume the port address of control port is 0FDH
Assume the port address of PORT B as 0FDHH
IN AL,0FDH  ;Read from PORT B
OUT 0FCH,AL ; Write to PORT B
```

6.4 RS-232 (Recommended Standard-232) Serial Interface

The RS-232(X) is a serial communication protocol, commonly used for transferring and receiving the serial data between two devices. It supports both synchronous and asynchronous data transmissions. Many devices in the industrial environment are still using RS-232 communication cable. Rs-232 cable is used to identify the difference of two signal levels between logic 1 and logic 0. The logic 1 is represented by the -12V and logic 0 is represented the +12V. The RS-232 cable works at different baud rates like 9600 bits/s, 2400 bits/s, 4800 bits/s etc. The RS-232 cable has two terminal devices namely Data Terminal Equipment and Data communication Equipment. Both devices will send and receives the signals. The data terminal equipment is a computer terminal and data communication Equipment is modems, or controllers etc.

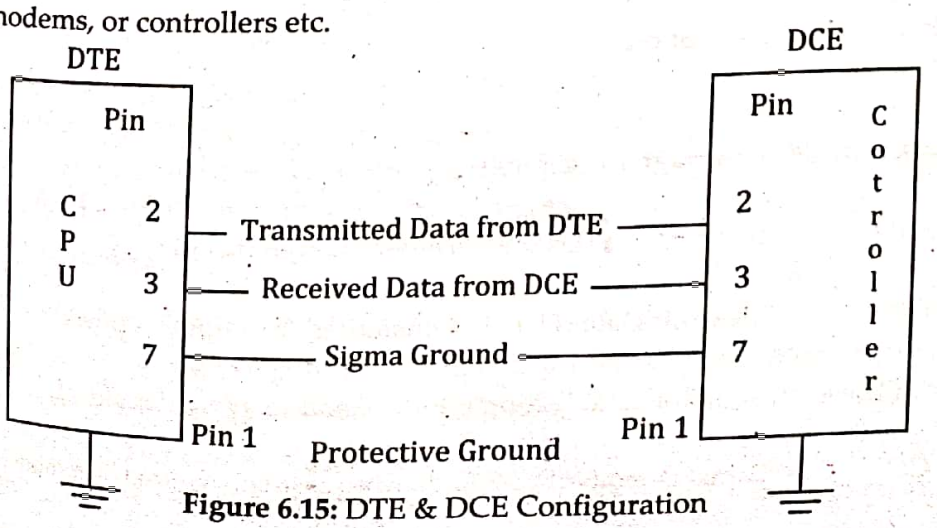


Figure 6.15: DTE & DCE Configuration

Nowadays most of the personal computers have two serial ports and one parallel port (RS232). These two types of ports are used for communicating with external devices and they work in different ways. The parallel port sends and receives the 8-bit data at a time over eight separate wires and this transfers the data very quickly, the parallel ports are typically used to connect a printer to a PC.

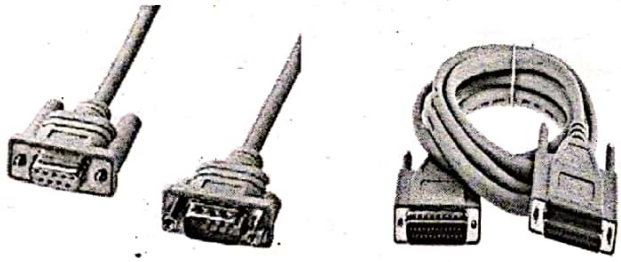


Figure 6.16: RS232 9 pin and 25 pin cables

A serial port sends and receives one-bit data at a time over one wire and it transfers data very slowly. The RS-232 stands for recommended slandered and 232 is a number X indicates the latest version like RS-232c, RS232s.

The most commonly used type of serial cable connectors are 9-pin connectors DB9 and 25-pin connector DB-25. Each of them may be a male or female type. Nowadays most of the computers use the DB9 connector for asynchronous data exchange. The maximum length of RS-232 cable is 50ft.

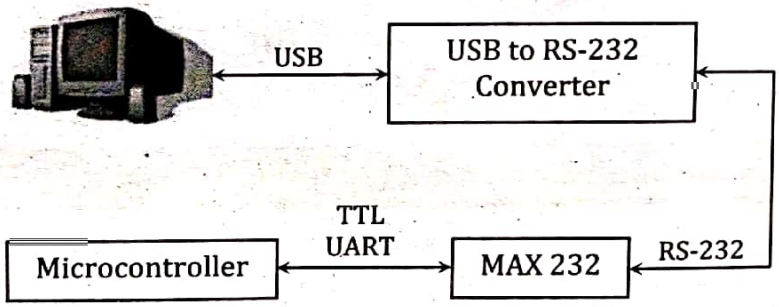


Figure 6.17: RS232 interfacing with computer

RS232 Pin Configuration for 9 Pin

New RS232 has nine pins as mentioned earlier. These nine pins are arranged in the port as shown in **RS232 Connector Pinout**. The DCE and DTE ports are exactly similar except for the direction of data flow. These nine pins are roughly divided in to three categories and we will discuss each category below.

Pin Number	Pin Name	Description
DATA pins (Data flow takes through these pins)		
2	RXD	Receive Data (Data is received though this pin)
3	TXD	Transmit Data (Data is transmitted through this pin)
CONTROL pins (These pins are for establishing interface and to avoid data loss)		
1	CD	Carrier Detect(Set by MODEM when answer is received by remote MODEM)

4	DTR	Data Terminal Ready(Set by PC to prepare MODEM to be connected to telephone circuit)
6	DSR	Data Set Ready(Set by MODEM to tell PC it is ready to receive and send data)
7	RTS	Request To Send(Set by PC to tell MODEM that MODEM can begin sending data)
8	CTS	Clear To send(Set by MODEM to tell PC that it is ready to receive data)
9	RI	Set by MODEM to tell PC a ringing condition has been detected.
REFERENCE		
5	GND	Ground (Used as reference for all pin voltage pulses)

RS232 Features and Specifications

1. RS232 uses Asynchronous communication so no clock is shared between PC and MODEM.
2. Logic '1' on pin is stated by voltage of range '-15V to -3V' and Logic '0' on pin is stated by voltage of range '+3V to +15V'. The logic has wide voltage range giving convenience for user.
3. MAX232 IC can be installed easily to establish RS232 interface with microcontrollers.
4. Full duplex interface of RS232 is very convenient.
5. Two pin simplex RS232 interface can also be established easily if required.
6. A maximum data transfer speed of 19 Kbps(Kilobits per second) is possible through RS232
7. A maximum current of 500mA can be drawn from pins of RS232
8. The interface can be established up to a distance of 50 feet.

Disadvantages of RS232

1. There is no pin dedicated for powering devices (No VCC)
2. More communication pins
3. Switching voltages between +15v and -15v is difficult at higher speeds
4. A maximum speed of 19 Kbps
5. A maximum distance of 50 feet
6. More pins lead to higher noise
7. Only a single device can be connected to RS232 connector unlike I2C
8. Need hardware to convert high voltage logic of RS232 to be compatible to TTL (controller and processor units)

Where RS232 is used?

A few examples where RS232 are suitable to use:

1. When you want a simple communication interface between two units. A two pin full duplex communication can be establishes easily on RS232 port.
2. RS232 is used in systems where clock sharing is difficult. RS232 is ASYNCHRONOUS so there will be no clock sharing between systems. All you need to do is set data bit rate for each unit.

Once baud rate is set the units will sample the data according to set baud rate.

3. RS232 is also used to control a single unit specifically without delay or errors.
4. RS232 interface also delivers data with more accuracy which is a requirement in some cases.

How to use RS232 connector?

As mentioned before we cannot connect RS232 directly to controller we need MAX232 IC to covert high voltage signals to TTL and vice versa. A typical circuit for it is shown below.

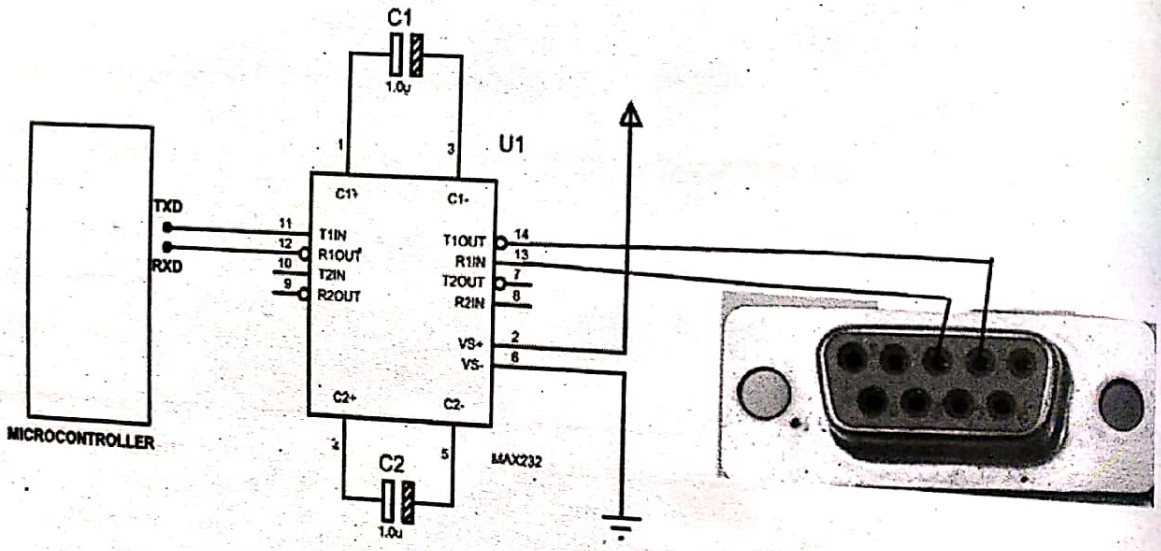


Figure 6.18: RS232 connector

In here we are connecting controller to female RS232 through MAX232 converter chip. The communication voltages reach as high as +15V and as low as -15V in RS232. There voltage levels cannot be used in sensitive electronics so we use a mediator that is MAX232.
The chip coverts TTL logic pulses of controller to RS232 voltage level pulses and vice versa. Without this chip you may damage something permanently.

Applications

1. Personal Computers
2. Modems
3. Servers
4. Memory devices
5. Motor control units
6. Printers and Scanners
7. Telephone lines
8. Used basically where serial data transfer is required

RS232 Pin Description for 25 Pin

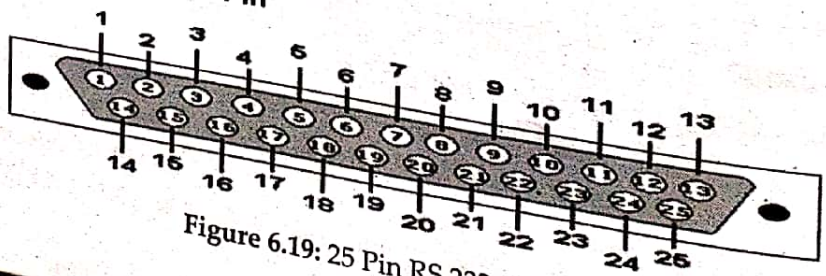


Figure 6.19: 25 Pin RS 232 Connector

It is a 25-pin connector, each pin has its own function is as follows.

PIN 1: *(Protective Ground);* It is a ground Pin.

PIN 2: Transmit Data.

PIN 3: Receive Data.

PIN 2 & PIN 3: These pins are most important pins for data transmitting and receiving. The 1 & 2-pins are used to data transmission and pin-3 used to data receiving purpose.

PIN 4: Request to send.

PIN 5: Clear to send.

PIN 6: Data Set Ready.

PIN 20: Data terminal Ready.

PIN 4, PIN 5, PIN 6, PIN 20: These pins are the handshaking pins (flow of control). Normally terminals cannot transmit the data until clear to send transmission is received from the DCE.

PIN 7: This pin is the common reference for all signals, including data, timing, and control signals. The DCE and DTE work properly across the serial interface and the pin-7 must be connected both ends without interface would not work.

PIN 8: This pin is also known as received line signal detect or carrier detect. This signal is activated when a suitable carrier is established between the local and remote DCE devices.

PIN 9: This pin is a DTE serial connector, this signal follows the incoming ring to an extent. Normally this signal is used by DCE auto answer mode.

PIN 10: Test Pin.

PIN 11: standby select.

PIN 12: Data Carrier Detect.

PIN 13: Clear to send.

PIN 14: Transmit data.

PIN 15: Transmit clock.

PIN 17: Receive clock.

PIN 24: External Clock.

PIN 15, 17, 24; Synchronous modems use the signals on these pins. These pins are controlled bit timing.

PIN 16: Receive data.

PIN 18: Test Pin.

PIN 19: Request to send.

PIN 21: (Signal Quality Detector); This pin Indicates the quality of the received carrier signal because the transmitting modem must be sent 0 or either 1 at each bit time, the modem controls the timing of the bits from the DTE.

PIN 22: (Ring Indicator): The ringing indicator means the DCE informs the DTE that the phone is ringing. All the modems designed for directly connected to the phone network equipped with auto answer.

PIN 23: Data Signal Rate Detector

QUESTIONS

1. What are the functions of I/O interface? Explain wit example.
2. Differentiate between serial communication and parallel communication.
3. Discuss synchronous and asynchronous serial data communication.
4. Explain 8251 USART with the help of its block diagram.
5. What are the different ways of parallel data communication? Explain.
6. Explain 8255 PPI with the help of a neat block diagram.
7. What is RS 232 interface? Explain with DTE and DCE.
8. Write instruction to initialize 8255 to configure PORT A as simple output port, PORT B as simple input port, PORT C upper as output and port c lower as an input port.
9. Write a program to take input from the 8 switches connected to PORT B and display the status of the switches thus red in the 8 LEDS connected to PORT A. Show how you derive the control word.
10. Write instruction to initialize 8255 to configure PORT A and display the status of the switches thus read in the 8 LEDS connected to PORT B. show how do you derive the control word.

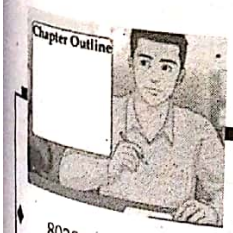
❑❑❑

7

ADVANCED MICROPROCESSORS

CHAPTER OBJECTIVE

The objective of this chapter is to introduce advanced microprocessor architectures. In first section, the 80286 architecture is explained with the help of its block diagram; register, flags and addressing modes used and real/protected modes of operation. This section also deals with the idea of privilege level, descriptor cache, memory access in GTD and LTD. The advance features of 80286 architecture are memory protection, addressing modes and cache. The last part of this chapter discuss about 80386 architecture with the help of its block diagram and register organization. In this section the concept of memory access in protection modes and paging is discussed.



CHAPTER OUTLINE

- 80286: Architecture (Block Diagram) , Registers, (Real/Protected mode), Privilege Levels, Descriptor Cache, Memory Access in GDT and LDT, Multitasking, Addressing Modes, Flag Register
- 80386: Architecture (Block Diagram), Register organization, Memory Access in Protected Mode, Paging (Up to LA to PA)

7.1 80286 Microprocessor

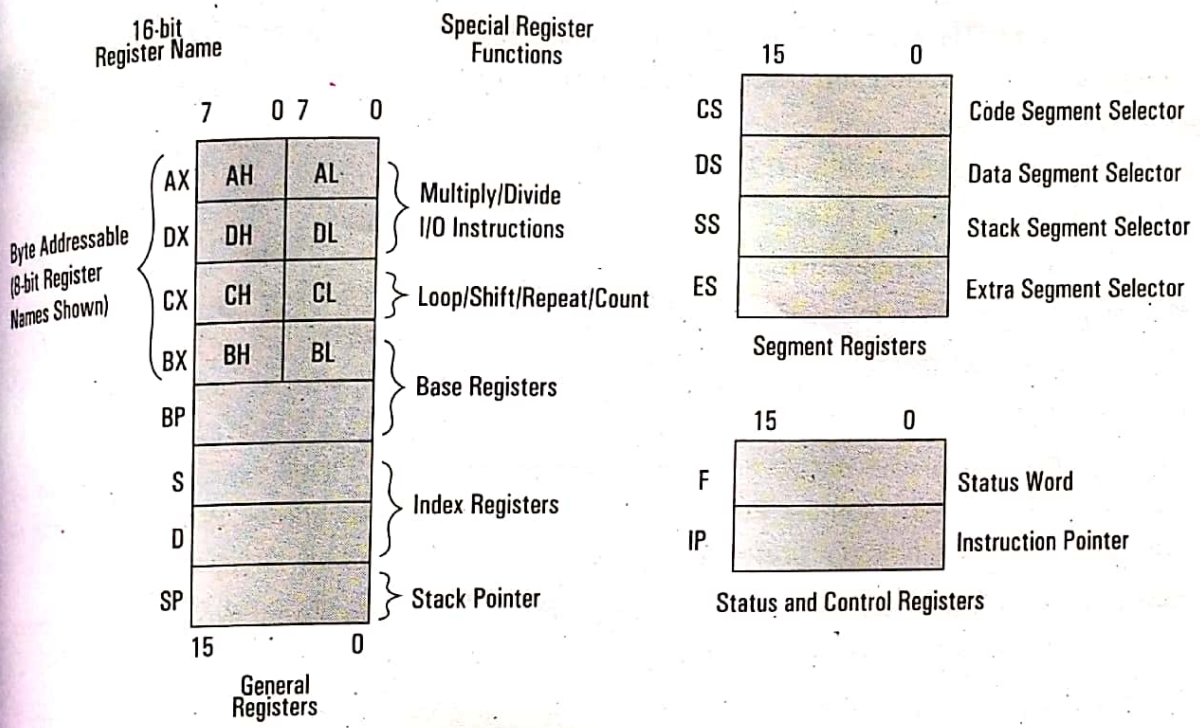
Salient Features of 80286

- **Advanced:** The 80286 is the first member of the family of advanced microprocessors with memory management and protection abilities. The 80286 CPU, with its 24-bit address bus is able to address 16 Mbytes of physical memory. Various versions of 80286 are available that runs on 12. 5 MHz, 10 MHz and 8 MHz clock frequencies. 80286 is upwardly compatible with 8086 in terms of instruction set.
- **Operating modes:** 80286 has two operating modes namely real address mode and virtual address mode. In real address mode, the 80286 can address up to 1Mb of physical memory address like 8086. In virtual address mode, it can address up to 16 Mb of physical memory address space and 1 GB of virtual memory address space.
- **Compatibility:** The instruction set of 80286 includes the instructions of 8086 and 80186. 80286 has some extra instructions to support operating system and memory management. In real address mode, the 80286 is object code compatible with 8086. In protected virtual address mode, its source code compatible with 8086. The performance of 80286 is five times faster than the standard 8086.
- **Need for Memory Management:** The part of main memory in which the operating system and other system programs are stored is not accessible to the users. It is required to ensure the smooth execution of the running process and also to ensure their protection. The memory management which is an important task of the operating system is supported by a hardware unit called memory management unit.
- **Swapping in of the Program:** Fetching of the application program from the secondary memory and placing it in the physical memory for execution by the CPU.
- **Swapping out of the executable Program:** Saving a portion of the program or important results required for further execution back to the secondary memory to make the program memory free for further execution of another required portion of the program.
- **Concept of Virtual Memory:** Large application programs requiring memory much more than the physically available 16 Mbytes of memory, may be executed by diving it into smaller segments. Thus for the user, there exists a very large logical memory space which is not actually available. Hence, there exists a virtual memory which does not exist physically in a system. This complete process of virtual memory management is taken care of by the 80286 CPU and the supporting operating system.

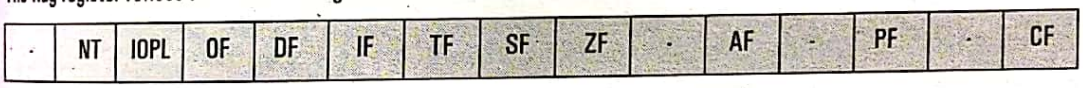
Register Organization of 80286

The 80286 CPU contains almost the same set of registers as in 8086, namely:

1. Four 16-bit general purpose registers
2. Four 16-bit segment registers
3. Status and control registers
4. Instruction Pointer



The flag register reflect the results of logical and arithmetic instructions



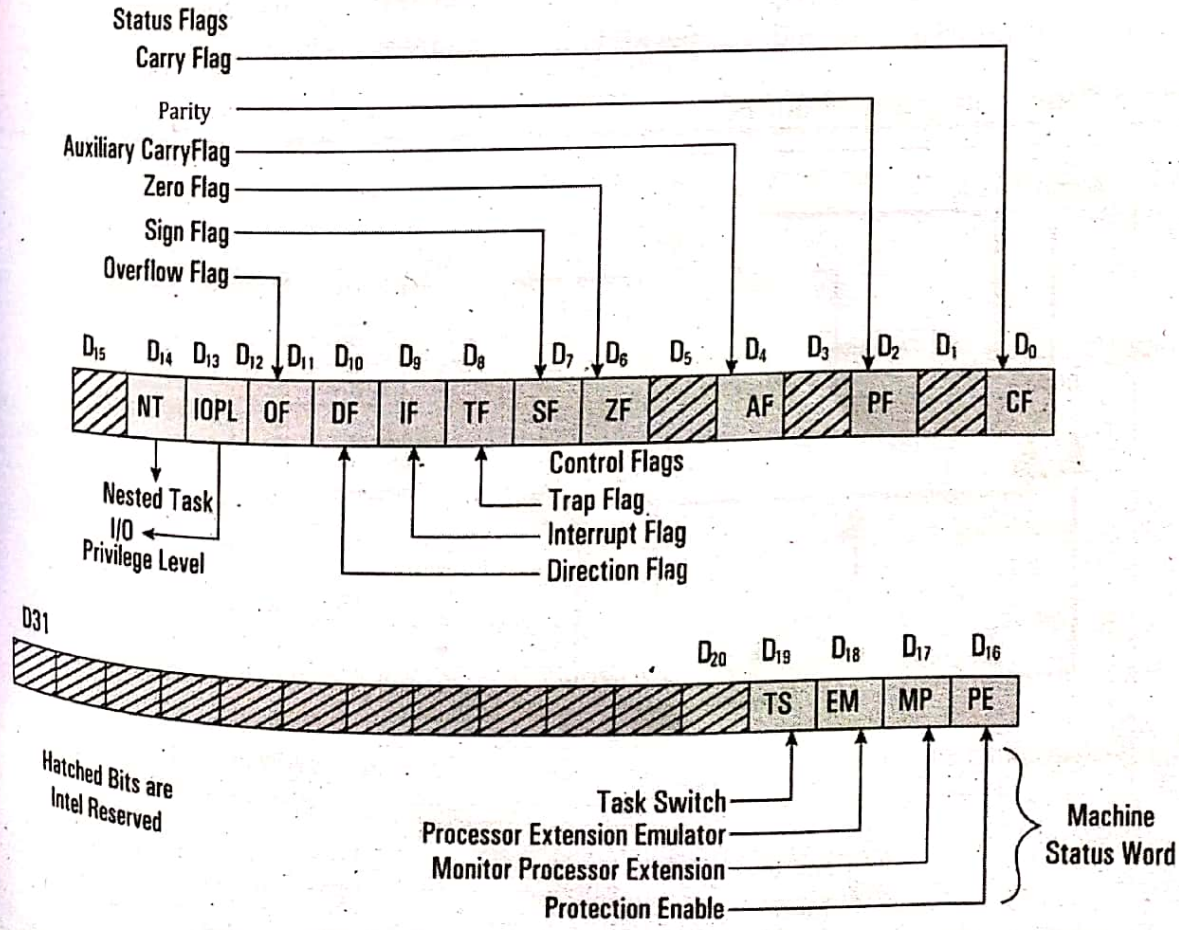


Figure 7.1: Register organization of 80286 architecture.

D2, D4, D6, D7 and D11 are called as status flag bits. The bits D8 (TF) and D9 (IF) are used for controlling machine operation and thus they are called control flags. The additional fields available in 80286 flag registers are:

1. IOPL - I/O Privilege Field (bits D12 and D13)
2. NT - Nested Task flag (bit D14)
3. PE - Protection Enable (bit D16)
4. MP - Monitor Processor Extension (bit D17)
5. EM - Processor Extension Emulator (bit D18)
6. TS – Task Switch (bit D19)

Protection Enable flag places the 80286 in protected mode, if set. This can only be cleared by resetting the CPU. If the Monitor Processor Extension flag is set, allows WAIT instruction to generate a processor extension not present exception.

Processor Extension Emulator flag if set, causes a processor extension absent exception and permits the emulation of processor extension by the CPU.

Task Switch flag if set, indicates the next instruction using extension will generate exception 7, permitting the CPU to test whether the current processor extension is for the current task.

Machine Status Word (MSW)

The machine status word consists of four flags – PE, MO, EM and TS of the four lower order bits D19 to D16 of the upper word of the flag register. The LMSW and SMSW instructions are available in the instruction set of 80286 to write and read the MSW in real address mode.

Internal Block Diagram of 80286

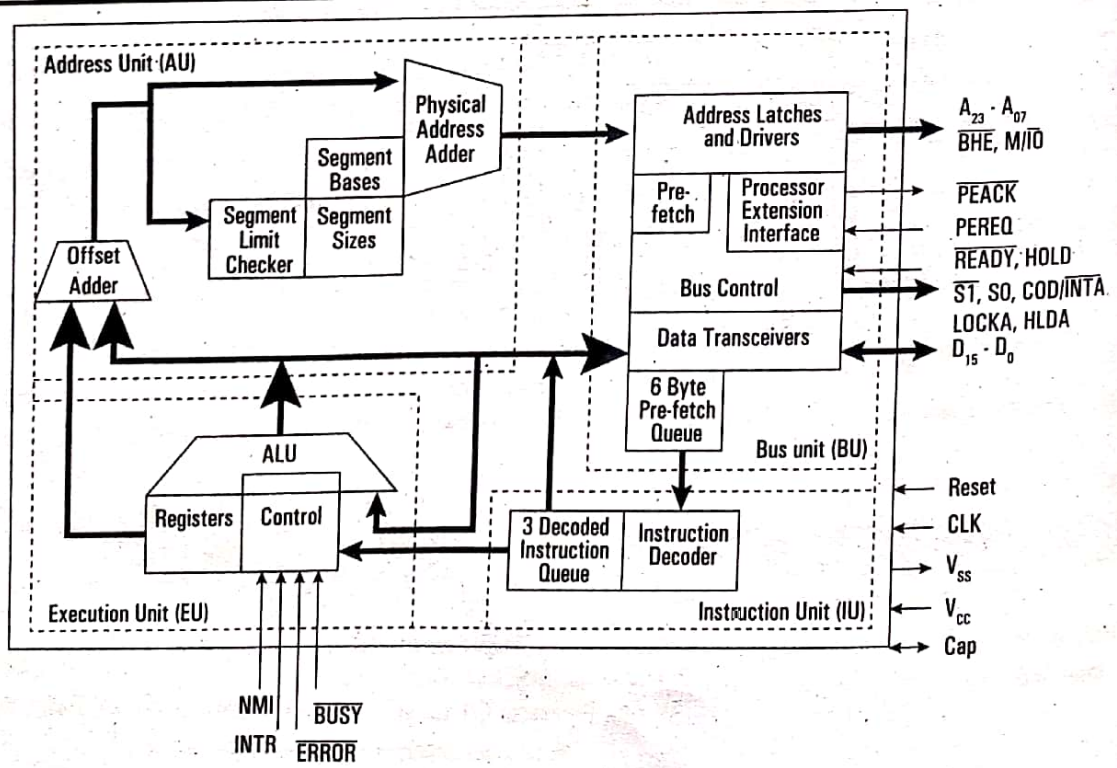


Figure 7.2: Block diagram of 80286 Architecture

The CPU contains four functional blocks. They are:

1. Address Unit (AU)
2. Bus Unit (BU)
3. Instruction Unit (IU)
4. Execution Unit (EU)

The address unit is responsible for calculating the physical address of instructions and data that the CPU wants to access. Also the address lines derived by this unit may be used to address different peripherals. The physical address computed by the address unit is handed over to the bus unit (BU) of the CPU. Major function of the bus unit is to fetch instruction bytes from the memory. Instructions are fetched in advance and stored in a queue to enable faster execution of the instructions. The bus unit also contains a bus control module that controls the prefetcher module. These prefetched instructions are arranged in a 6-byte instructions queue. The 6-byte prefetch queue forwards the instructions arranged in it to the instruction unit (IU). The instruction unit accepts instructions from the prefetch queue and an instruction decoder decodes them one by one. The decoded instructions are latched onto a decoded instruction queue. The output of the decoding circuit drives a control circuit in the execution unit, which is responsible for executing the instructions received from decoded instruction queue. The decoded instruction queue sends the data part of the instruction over the data bus. The EU contains the register bank used for storing the data as scratch pad, or used as special purpose registers. The ALU, the heart of the EU, carries out all the arithmetic and logical operations and sends the results over the data bus or back to the register bank.

Signal Description of 80286

- **CLK:** This is the system clock input pin. The clock frequency applied at this pin is divided by two internally and is used for deriving fundamental timings for basic operations of the circuit. The clock is generated using 8284 clock generator.
- **D15-D0:** These are sixteen bidirectional data bus lines.
- **A23-A0:** These are the physical address output lines used to address memory or I/O devices. The address lines A23 - A16 are zero during I/O transfers.
- **BHE:** This output signal, as in 8086, indicates that there is a transfer on the higher byte of the data bus (D15 – D8).
- **S1, S0:** These are the active-low status output signals which indicate initiation of a bus cycle and with M/IO and COD/INTA, they define the type of the bus cycle.
- **M/ IO:** This output line differentiates memory operations from I/O operations. If this signal is it "0" indicates that an I/O cycle or INTA cycle is in process and if it is "1" it indicates that a memory or a HALT cycle is in progress.
- **COD/ INTA:** This output signal, in combination with M/ IO signal and S1 , S0 distinguishes different memory, I/O and INTA cycles.
- **LOCK:** This active-low output pin is used to prevent the other masters from gaining the control of the bus for the current and the following bus cycles. This pin is activated by a "LOCK" instruction prefix, or automatically by hardware during XCHG, interrupt acknowledge or descriptor table access.

- **READY:** This active-low input pin is used to insert wait states in a bus cycle, for interfacing low speed peripherals. This signal is neglected during HLDA cycle.
- **HOLD and HLDA** This pair of pins is used by external bus masters to request for the control of the system bus (HOLD) and to check whether the main processor has granted the control (HLDA) or not, in the same way as it was in 8086.
- **INTR:** Through this active high input, an external device requests 80286 to suspend the current instruction execution and serve the interrupt request. Its function is exactly similar to that of INTR pin of 8086.
- **NMI:** The Non-Maskable Interrupt request is an active-high, edge-triggered input that is equivalent to an INTR signal of type 2. No acknowledge cycles are needed to be carried out.
- **PEREG and PEACK** (Processor Extension Request and Acknowledgement): Processor extension refers to coprocessor (80287 in case of 80286 CPU). This pair of pins extends the memory management and protection capabilities of 80286 to the processor extension 80287. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The PEACK active-low output indicates to the processor extension that the requested operand is being transferred.
- **BUSY and ERROR:** Processor extension BUSY and ERROR active-low input signals indicate the operating conditions of a processor extension to 80286. The BUSY goes low, indicating 80286 to suspend the execution and wait until the BUSY become inactive. In this duration, the processor extension is busy with its allotted job. Once the job is completed the processor extension drives the BUSY input high indicating 80286 to continue with the program execution. An active ERROR signal causes the 80286 to perform the processor extension interrupt while executing the WAIT and ESC instructions. The active ERROR signal indicates to 80286 that the processor extension has committed a mistake and hence it is reactivating the processor extension interrupt.
- **CAP:** A 0. 047 μf, 12V capacitor must be connected between this input pin and ground to filter the output of the internal substrate bias generator. For correct operation of 80286 the capacitor must be charged to its operating voltage. Till this capacitor charges to its full capacity, the 80286 may be kept stuck to reset to avoid any spurious activity.
- **Vss:** This pin is a system ground pin of 80286.
- **Vcc:** This pin is used to apply +5V power supply voltage to the internal circuit of 80286. The active-high RESET input clears the internal logic of 80286, and reinitializes whose input pulse width should be at least 16 clock cycles. The 80286 requires at least 38 clock cycles after the trailing edge of the RESET input signal, before it makes the first opcode fetch cycle.

Real Address Mode

- ⌘ Act as a fast 8086
- ⌘ Instruction set is upwardly compatible
- ⌘ It addresses only 1M byte of physical memory using A0-A19.

The 80286 addresses only 1Mbytes of physical memory using A0- A19. The lines A20-A23 are not used by the internal circuit of 80286 in this mode. In real address mode, while addressing the physical memory, the 80286 uses BHE along with A0- A19. The 20-bit physical address is again formed in the same way as that in 8086.

The contents of segment registers are used as segment base addresses. The other registers, depending upon the addressing mode, contain the offset addresses. Because of extra pipelining and other circuit level improvements, in real address mode also, the 80286 operates at a much faster rate than 8086, although functionally they work in an identical fashion. As in 8086, the physical memory is organized in terms of segments of 64Kbyte maximum size.

An exception is generated, if the segment size limit is exceeded by the instruction or the data. The overlapping of physical memory segments is allowed to minimize the memory requirements for a task. The 80286 reserves two fixed areas of physical memory for system initialization and interrupt vector table. In the real mode the first 1Kbyte of memory starting from address 0000H to 003FFH is reserved for interrupt vector table. Also the addresses from FFFF0H to FFFFFH are reserved for system initialization.

The program execution starts from FFFFH after reset and initialization. The interrupt vector table of 80286 is organized in the same way as that of 8086. Some of the interrupt types are reserved for exceptions, single-stepping and processor extension segment overrun, etc.

When the 80286 is reset, it always starts the execution in real address mode. In real address mode, it performs the following functions:

- It initializes the IP and other registers of 80286,
- It prepares for entering the protected virtual address mode.

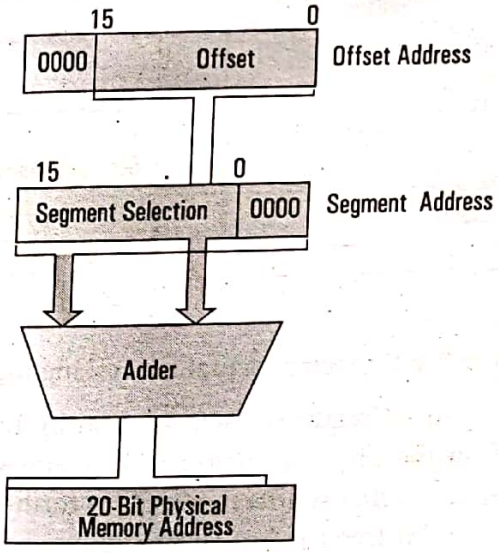


Figure 7.3: Real addressing mode

Protected Virtual Address Mode (PVAM)

80286 is the first processor to support the concepts of virtual memory and memory management. The virtual memory does not exist physically but it still appears to be available within the system. The concept of VM is implemented using physical memory that the CPU can directly access and secondary memory that is used as a storage for data and program, which are stored in secondary memory initially.

The Segment of the program or data required for actual execution at that instant is fetched from the secondary memory into physical memory. After the execution of this fetched segment, the next segment required for further execution is again fetched from the secondary memory, while the results of the executed segment are stored back into the secondary memory for further references. This continues till the complete program is executed.

During the execution the partial results of the previously executed portions are again fetched into the physical memory, if required for further execution. The procedure of fetching the chosen program segments or data from the secondary storage into physical memory is called swapping. The procedure of storing back the partial results or data back on the secondary storage is called un-swapping. The virtual memory is allotted per task.

The 80286 is able to address 1 G byte (230 bytes) of virtual memory per task. The complete virtual memory is mapped on to the 16Mbyte physical memory. If a program larger than 16Mbyte is stored on the hard disk and is to be executed, if it is fetched in terms of data or program segments of less than 16Mbyte in size into the program memory by swapping sequentially as per sequence of execution.

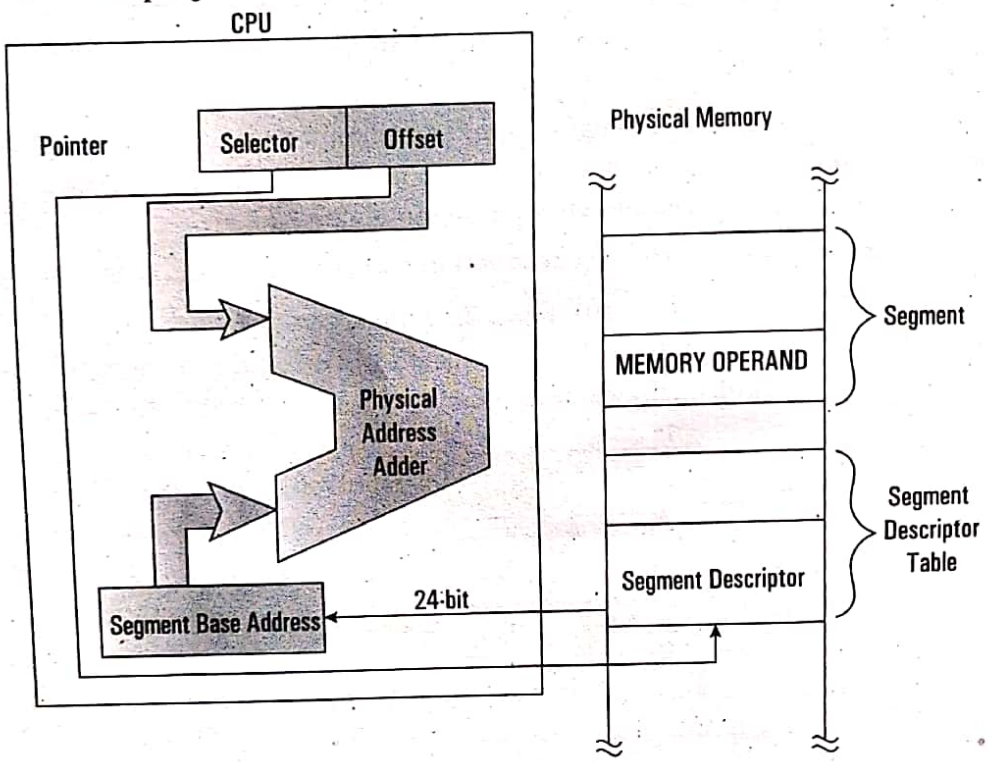


Figure 7.4: Protected virtual addressing mode

Whenever the portion of a program is required for execution by the CPU, it is fetched from the secondary memory and placed in the physical memory is called swapping in of the program. A portion of the program or important partial results required for further execution, may be saved back on secondary storage to make the PM free for further execution of another required portion of the program is called swapping out of the executable program.

80286 uses the 16-bit content of a segment register as a selector to address a descriptor stored in the physical memory. The descriptor is a block of contiguous memory locations containing information of a segment, like segment base address, segment limit, segment type, privilege level, segment availability in physical memory, descriptor type and segment use another task.

Multitasking

Multitasking has the same meaning of multiprogramming but in a more general sense, as it refers to having multiple (programs, processes, tasks, threads) running at the same time. This term is used in modern operating systems when multiple tasks share a common processing resource (e. g. , CPU and Memory). At any time the CPU is executing one task only while other tasks waiting their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task (i. e. *process* or *thread*

context switching).

There are subtle differences between multitasking and multiprogramming. A *task* in a multitasking operating system is not a whole application program but it can also refer to a "thread of execution" when one process is divided into sub-tasks. Each smaller task does not hijack the CPU until it finishes like in the older multiprogramming but rather a fair share amount of the CPU time called quantum. Just to make it easy to remember, both multiprogramming and multitasking operating systems are **(CPU) time sharing** systems. However, while in multiprogramming (older OSs) one program as a whole keeps running until it blocks, in multitasking (modern OSs) time sharing is best manifested because each running process takes only a fair quantum of the CPU time.

Privilege Level

There are four types of privilege levels: (1) 00- Kernel level (highest privilege level); (2) OS services; (3) 10 - OS extension; (4) 11 - Applications (lowest privilege level).

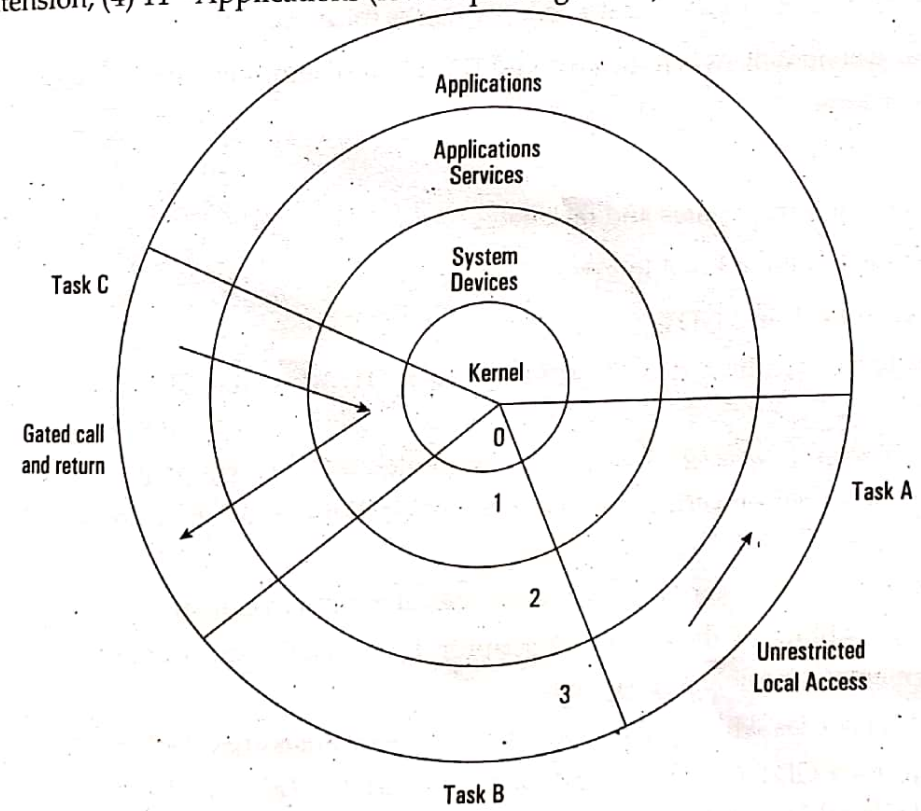


Figure 7.5: Illustration of privilege level

- Each task is assigned a privilege level, which indicates the priority or privilege of that task.
- It can only changed by transferring the control, using gate descriptors, to a new segment.
- A task executing at level 0, the most privileged level, can access all the data segment defined in GDT and LDT of the task.
- A task executing at level 3, the least privileged level, will have the most limited access to data and other descriptors.
- The use of rings allows for system software to restrict tasks from accessing data.
- In most environments, the operating system and some device drivers run in ring 0 and applications run in ring 3.

GDT & LDT

Global Descriptor Table (GDT)

- The 80286 has a single Global Descriptor Table (GDT) which is shared between all tasks and addresses up to 512MB of the virtual address space.
- The Global Descriptor Table or GDT is a data structure used by Intel x86-family processor starting with the 80286 in order to define the characteristics of the various memory areas used during program execution, including the base address, the size and access privileges like execute-ability and write-ability.

Local Descriptor Table (LTD)

- Each task will have its own Local Descriptor Table (LTD) which is a private 512MB of address space.
- LDT is essential to implement separate address spaces for multiple processes.
- The operating system will switch the current LDT when scheduling a new process, using the LDT machine instruction.

Descriptor Table (LDT)

- IDT used to store interrupt gates and task gates.
- LIDT instruction is used to Load Interrupt Descriptor table.

Difference between LDT and GDT

- LDT is actually defined by a descriptor inside the GDT, while the GDT is directly defined by a linear address.
- The lack of symmetry between both tables is underlined by the fact that the current LDT can be automatically switched on certain events, notably if TSS-based multitasking is used, while this is not possible for the GDT.
- The LDT also cannot store certain privileged types of memory segments.
- The LDT is the sibling of the Global Descriptor Table (GDT) and similarly defines up to 8191 memory segments accessible to programs.
- LDT (and GDT) entries which point to identical memory areas are called *aliases*.
- Instruction to load GDT is LGDT (Load Global Descriptor Table) and instruction to load LDT is LLDT (Load Global Descriptor Table). Both are privileged instructions.

Descriptor

- Descriptor is a identifier of a program segment or page.
- A segment cannot be accessed, if its descriptor does not exist in either LDT or GDT.
- Set of descriptor (descriptor table) arranged in a proper sequence describes the complete program.
- The descriptor is a block of contiguous memory location containing information of a segment, like
  (i) Segment base address
  (ii) Segment limit

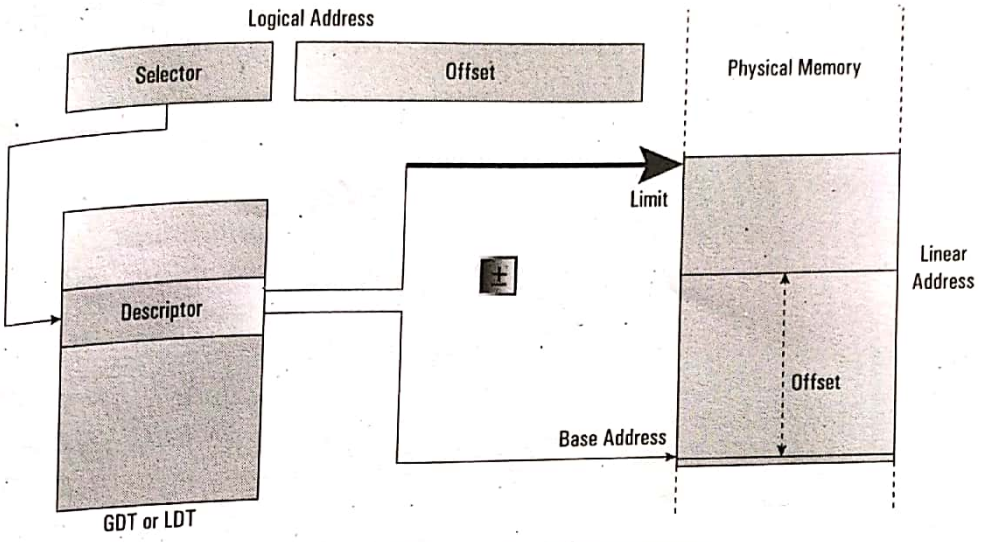


Figure 7.6: Illustration of descriptor

(iii) Segment type
(iv) Privilege level – prevents unauthorized access
(v) Segment availability in physical memory
(vi) Descriptor type
(vii) Segment use by another task

Requirement of Descriptor Table

- The descriptor describes the location, length, and access rights of the segment of memory.
- The selector, located in the segment register, selects one of descriptors from one of two tables of descriptors.

7.2 80386 Microprocessor

- The Internal Architecture of 80386 is divided into 3 sections.
  - Central processing unit
  - Memory management unit
  - Bus interface unit
- Central processing unit is further divided into Execution unit and Instruction unit Execution unit has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses.

Architecture of 80386

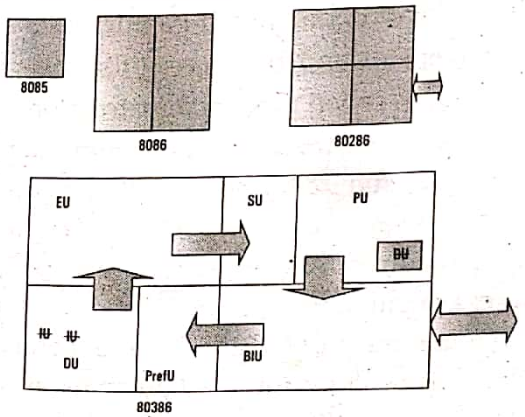


Figure 7.7: Block diagram view of evolution from 8085 to 80386 microprocessor

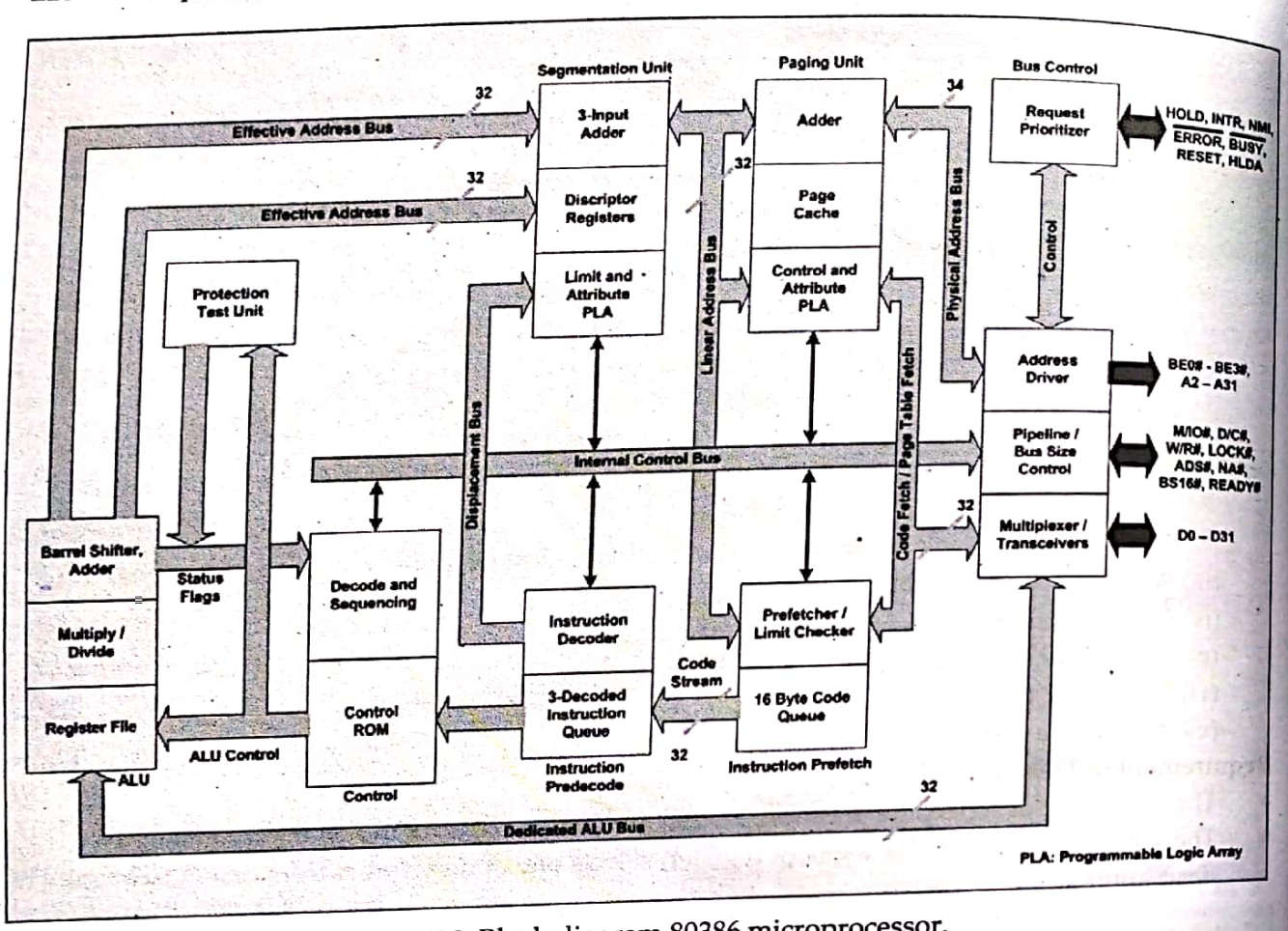


Figure 7.8: Block diagram 80386 microprocessor.

The internal architecture of 80386 is divided into following sections:

- ⌘ Central processing unit
- ⌘ Execution unit
- ⌘ Instruction decode unit
- ⌘ Memory management unit
- ⌘ Segmentation unit
- ⌘ Paging unit

Bus Control Unit

The central processing unit is further divided into **Execution unit** and **Instruction unit**. The memory management unit consists of a segmentation unit and a paging unit. These units operate in parallel. Fetching, decoding, memory management and bus access for several instructions are performed simultaneously. This parallel operation is called pipelined instruction processing.

- **Execution unit:** The execution unit read the instruction from the instruction queue and executes the instructions. It consists of three sub unit: control unit, data unit and protection test unit.
- **Control unit:** It contains microcode and special hardware. The microcode and special hardware allow 80386DX to reduce time required for execution of multiply and divide instruction. It also speeds up the effective address calculation.
- **Data unit:** The data unit contains the ALU, eight 32-bit general purpose registers and a 64-bit barrel shifter. The barrel shifter is used for multiple bit shifts in one clock. Thus it increases the speed of all shift and rotate operation. The multiply/divide logic implement the bit shift rotate

algorithm to complete the operation in minimum time. The entire data unit is responsible for data operation requested by the control unit.

- **Protection test unit**: The protection test unit checks for segmentation violations under the control of the microcode.
- **Instruction decode unit:** The instruction decode unit takes the instruction bytes from the code prefetch queue and translates them into microcode the decoded. The decoded instruction are then stored in the instruction queue. They are passed to control section for deriving the necessary control signals.
- **Segmentation unit:** The segmentation unit translates logic addresses into linear addresses at request of the execution unit. The segmentation unit compares the effective address for the length limit specified in the segment descriptor. The segment unit adds the segment base and the effective address to generate linear address. Before calculation of linear address it also checks for access rights. It provides a 4-level protection mechanism for protecting and isolating the system code and data from those of the application program.
- **Paging unit:** When the 80386DX paging mechanism is enabled, the paging unit translates linear addresses generated by the segmentation unit or the code prefetch unit into physical addresses. If paging unit is not enabled, the physical address is the same as the linear address, and no translation is necessary. The paging unit gives physical address to the bus interface unit to perform memory and I/O accesses. It organizes the physical memory in term of pages of 4 Kbytes size each.

  The control and attribute PLA check the privileges at the page level. Each of the page maintain the paging information of the task. The limit and attribute PLA checks segment limits and attributes at the segment level to avoid invalid accesses to code and data in the memory segments.
- **Bus control unit:** The bus control unit is the 80386DX's communication with the outside world. It provides a full 32-bit bi-directional data bus and 32-bit address bus. The bus control unit is responsible for the following operations:
  - It accepts internal request for code fetch and data transfer from the code fetch unit and from the execution unit. It then prioritize the request with the help of prioritize and generate signal to perform bus cycles.
  - It sends address, data and control signal to communicate with memory and I/O devices. The address driver drives the bus enable and address signal A0-A31 and the transceiver interface the internal data bus with the system bus.
  - It control the interface to the external bus masters and coprocessors.
  - It also provides the address relocation facility.

Instruction Prefetch Unit

The instruction prefetch unit sequentially fetch the instruction byte stream from the memory. It uses bus control unit to fetch instruction bytes when the bus control unit is not performing bus cycle to execute an instruction. These prefetched instruction bytes are stored in the 16-byte code queue. A 16-byte code queue holds these instructions until the decoder needs them the prefetcher always fetches instruction in the order in which they appear in the memory. In fact, the prefetcher simply reads code one double word at the time, not caring whether it's bringing in complete instruction are executed, the contents of the prefetched and decode queues are cleared out. In this case, prefetcher again starts filling its queue.

Instruction predecode unit: The instruction predecode unit takes instruction bytes from the instruction prefetch queue and translates them into microcode the decoded instruction are then stored in instruction queue.

Register Organization

Types of Registers

- Flag register
- Segment descriptor register
- Control register
- System address register
- Debug and test register

- The 80386 has eight 32 - bit general purpose registers which may be used as either 8 bit or 16 bit registers. A 32 - bit register known as an extended register, is represented by the register name with prefix E.
- Example: A 32 bit register corresponding to AX is EAX, similarly BX is EBX etc.
- The 16 bit registers BP, SP, SI and DI in 8086 are now available with their extended size of 32 bit and are names as EBP, ESP, ESI and EDI.
- AX represents the lower 16 bit of the 32 bit register EAX.
- BP, SP, SI, DI represents the lower 16 bit of their 32 bit counterparts, and can be used as independent 16 bit registers.
- The six segment registers available in 80386 are CS, SS, DS, ES, FS and GS.
- The CS and SS are the code and the stack segment registers respectively, while DS, ES, FS, GS are 4 data segment registers.
- A 16 bit instruction pointer IP is available along with 32 bit counterpart EIP.

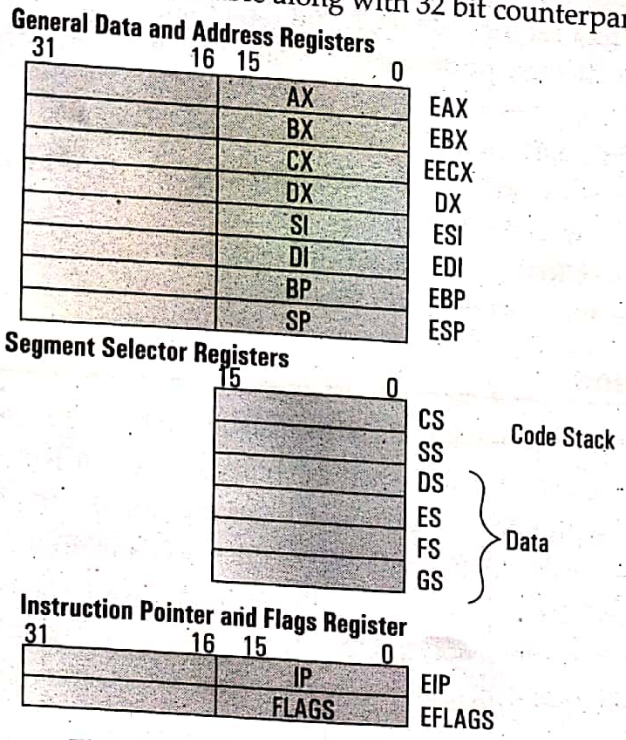


Figure 7.9: Types of registers in 80386

Control Registers

- The 80386 has three 32 bit control registers CR0, CR2 and CR3 to hold global machine status independent of the executed task.
- Load and store instructions are available to access these registers.

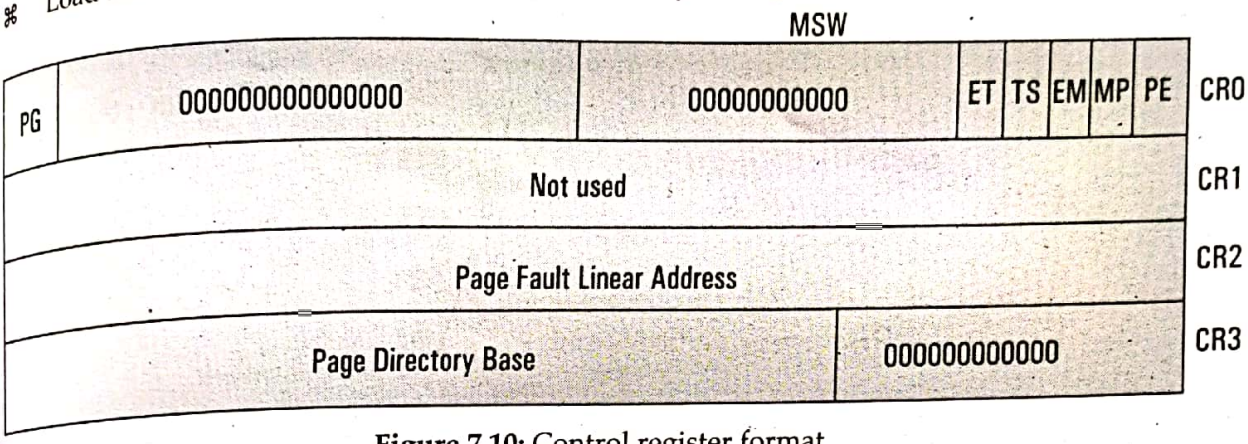


Figure 7.10: Control register format

- **CR2:** The control register CR2 is used to store the 32-bit linear address at which the previous page fault was detected (PFLA).
- **CS3:** Used when virtual addressing is enabled, hence when the PG bit is set in CR0. CR3 enables the processor to translate linear address into physical address by locating the page directory and page tables for the current task. Typically, the upper 20 bits of CR3 become the page directory base register (PDBR), which stores the physical address of the first page directory entry.
- **CR0:** Register CR0 contains a number of special control bits that are defined as follow:
  - **PG (Pagging):** If PG is set, it enables paging and use the CR3 register, else disable paging.
  - **ET (Extension type):** On the 80386, it allowed to specify whether the external math coprocessor was an 80287 or 80837. If ET = 0 than it is 80287, else 80387.
  - **TS (Task switch):** The processor sets TS with every task switch and tests TS when interpreting coprocessor instructions.
  - **EM (Emulation):** EM indicates whether coprocessor functions are to be emulated.
  - **PE (Protected mode enables):** If 1, system is in protected mode, else system is in real mode.

Debug and Test Registers

- Intel has provide a set of 8 debug registers for hardware debugging.
- Out of these eight registers DR0 to DR7, two registers DR4 and DR5 are Intel reserved.
- The initial four registers DR0 to DR3 store four program controllable breakpoint addresses
- While DR6 and DR7 respectively hold breakpoint status and breakpoint control information.
- Two more test register are provided by 80386 for page caching namely test control and test status register

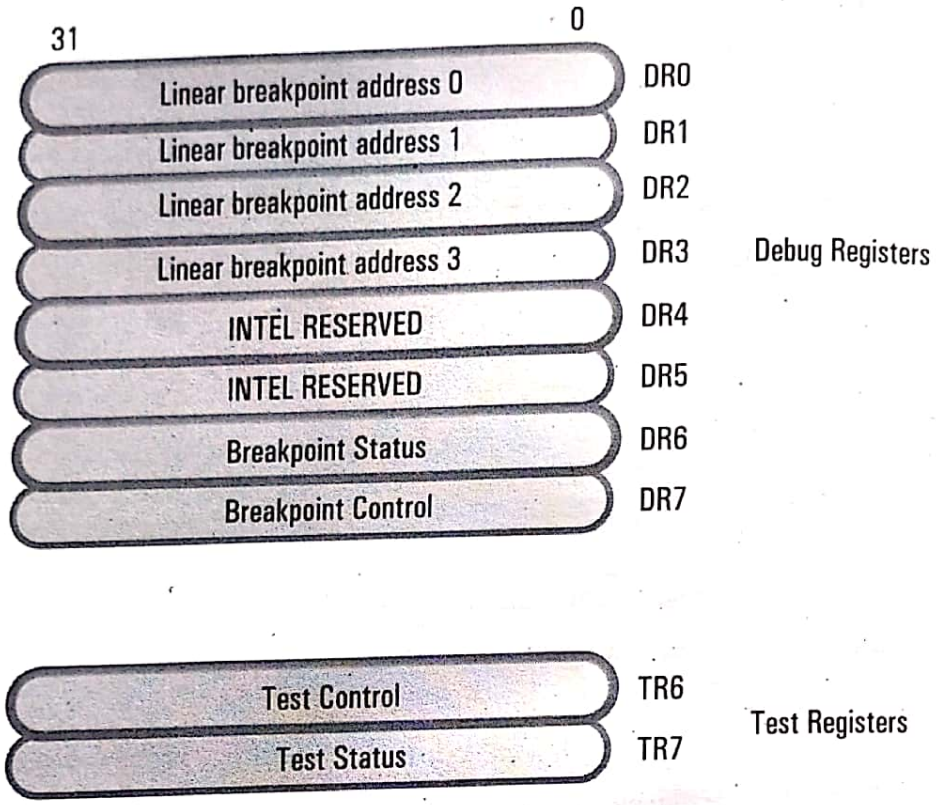


Figure 7.11: Debug and test register.

System Address Registers

Four special registers are defined to refer to the descriptor tables supported by 80386.

The 80386 supports four types of descriptor table, viz.

- ⌘ Global descriptor table (GDT),
- ⌘ Interrupt descriptor table (IDT),
- ⌘ Local descriptor table (LDT),
- ⌘ Task state segment descriptor (TSS).

Segment Descriptor Registers

- ⌘ This registers are not available for programmers, rather they are internally used to store the descriptor information, like attributes, limit and base addresses of segments.
- ⌘ The six segment registers have corresponding six 73 bit descriptor registers. Each of them contains 32 bit base address, 32 bit base limit and 9 bit attributes. These are automatically loaded when the corresponding segments are loaded with selectors.

Memory Access in Protected Mode of 80386

- ⌘ All the capabilities of 80386 are available for utilization in its protected mode of operation.
- ⌘ The 80386 in protected mode support all the software written for 80286 and 8086 to be executed under the control of memory management and protection abilities of 80386.
- ⌘ The protected mode allows the use of additional instruction, addressing modes and capabilities of 80386.

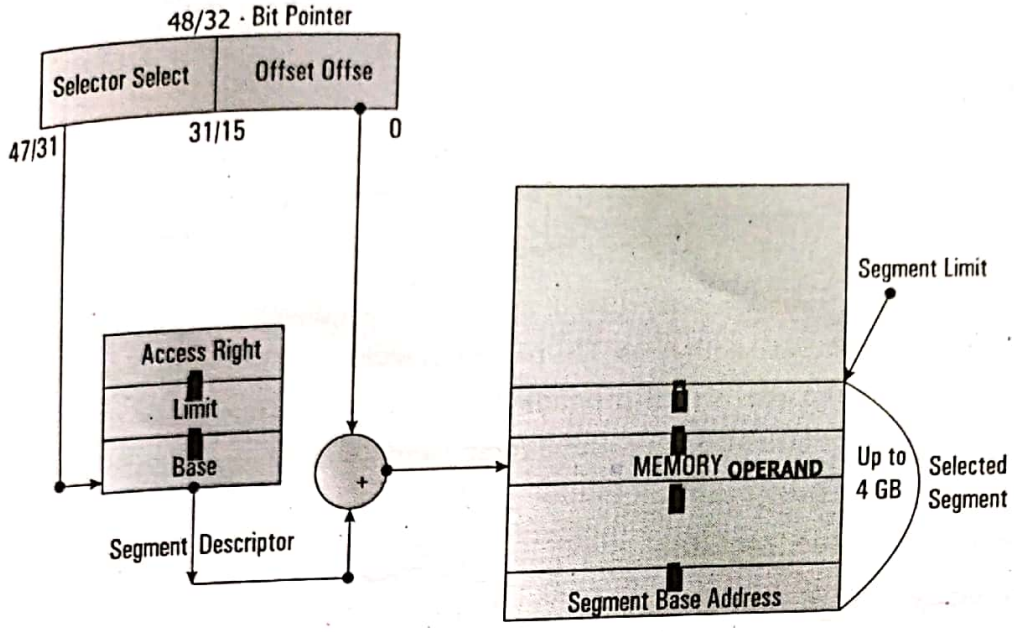


Figure 7.12: Memory access in protected mode

ADDRESSING IN PROTECTED MODE

In this mode, the contents of segment registers are used as selectors to address descriptors which contain the segment limit, base address and access rights byte of the segment.

- The effective address (offset) is added with segment base address to calculate linear address. This linear address is further used as physical address, if the paging unit is disabled, otherwise the paging unit converts the linear address into physical address.
- The paging unit is a memory management unit enabled only in protected mode. The paging mechanism allows handling of large segments of memory in terms of pages of 4Kbyte size.
- The paging unit operates under the control of segmentation unit.

PAGING

Paging is one of the memory management techniques used for virtual memory multitasking operating system. The segmentation scheme may divide the physical memory into a variable size segments but the paging divides the memory into a fixed size pages. The segments are supposed to be the logical segments of the program, but the pages do not have any logical relation with the program. The pages are just fixed size portions of the program module or data.

The advantage of paging scheme is that the complete segment of a task need not be in the physical memory at any time. Only a few pages of the segments, which are required currently for the execution need to be available in the physical memory. Thus the memory requirement of the task is substantially reduced, relinquishing the available memory for other tasks. Whenever the other pages of task are required for execution, they may be fetched from the secondary storage. The previous page which are executed, need not be available in the memory, and hence the space occupied by them may be relinquished for other tasks. Thus paging mechanism provides an effective technique to manage the physical memory for multitasking systems.

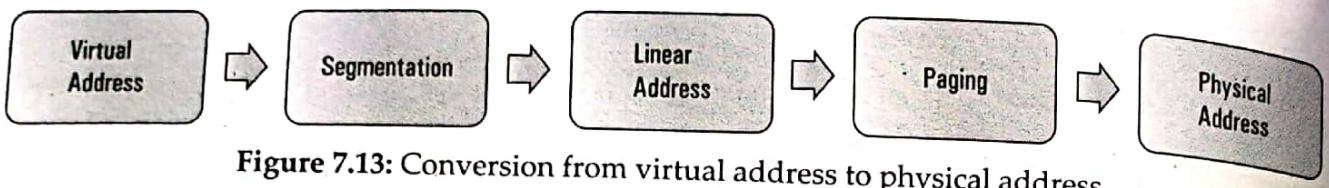


Figure 7.13: Conversion from virtual address to physical address

Paging Unit

The paging unit of 80386 uses a two level table mechanism to convert a linear address provided by segmentation unit into physical addresses. The paging unit converts the complete map of a task into pages, each of size 4K. The task is further handled in terms of its page, rather than segments. The paging unit handles every task in terms of three components namely page directory, page tables and page itself.

Paging Descriptor Base Register

The control register CR2 is used to store the 32-bit linear address at which the previous page fault was detected.

The CR3 is used as page directory physical base address register, to store the physical starting address of the page directory. The lower 12 bit of the CR3 are always zero to ensure the page size aligned directory. A move operation to CR3 automatically loads the page table entry caches and a task switch operation, to load CR0 suitably.

Page Directory

This is at the most 4Kbytes in size. Each directory entry is of 4 bytes, thus a total of 1024 entries are allowed in a directory. The upper 10 bits of the linear address are used as an index to the corresponding page directory entry. The page directory entries point to page tables.

- ⌘ It has 1024 Entries each of 4 Bytes which points towards starting of the paging table.
- ⌘ In other words, 1K Paging Directory supports 1K Paging Tables.
- ⌘ Total size of Paging Directory = 1024 * 4B = 4K.

Page Tables

- ⌘ It has 1024 Entries each of 4 Bytes which points towards the pages.
- ⌘ Every Paging table supports 1K Pages.
- ⌘ Total size of Paging Table = 1024 * 4B = 4K

Each page table is of 4Kbytes in size and many contain a maximum of 1024 entries. The page table entries contain the starting address of the page and the statistical information about the page. The upper 20 bit page frame address is combined with the lower 12 bit of the linear address. The address bits A12- A21 are used to select the 1024 page table entries. The page table can be shared between the tasks.

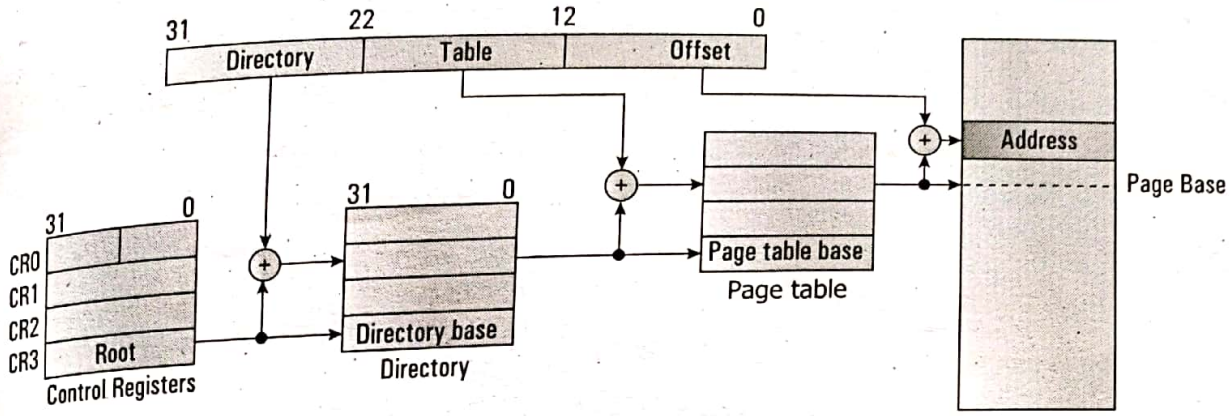


Figure 7.14: Illustration of page table

Paging Operation

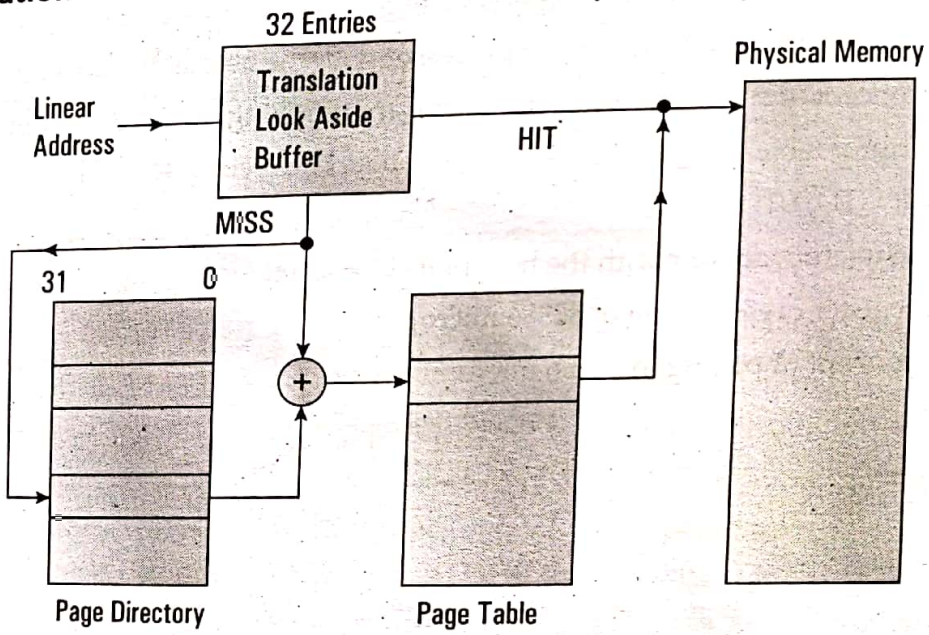


Figure 7.15: Illustration of paging operation.

- The P bit of the above entries indicate, if the entry can be used in address translation.
- If P=1, the entry can be used in address translation, otherwise it cannot be used.
- The P bit of the currently executed page is always high.
- The accessed bit A is set by 80386 before any access to the page. If A=1, the page is accessed, else unassessed.
- The D bit (Dirty bit) is set before a write operation to the page is carried out. The D-bit is undefined for page director entries.
- The OS reserved bits are defined by the operating system software.
- The User / Supervisor (U/S) bit and read/write bit are used to provide protection. These bits are decoded to provide protection under the 4 level protection model.
- The level 0 is supposed to have the highest privilege, while the level 3 is supposed to have the least privilege.
- This protection provide by the paging unit is transparent to the segmentation unit.

QUESTIONS

1. Introduce 80286 microprocessor and explain its register organization.
2. Explain the block diagram of 80286 microprocessor with the help of its functional blocks.
3. Explain the concept of PVAM, privilege level and multitasking in 80286 microprocessor.
4. What is descriptor table? What is its use? Differentiate between GTD and LTD.
5. Discuss about GTD, LTD and IDT.
6. Explain 80386 microprocessor with the help of its block diagram.
7. Explain the register organization of 80386 microprocessor.
8. Explain the concept of paging in 80386 microprocessor.

❑❑❑

Project Works using 8085 Microprocessor Programming

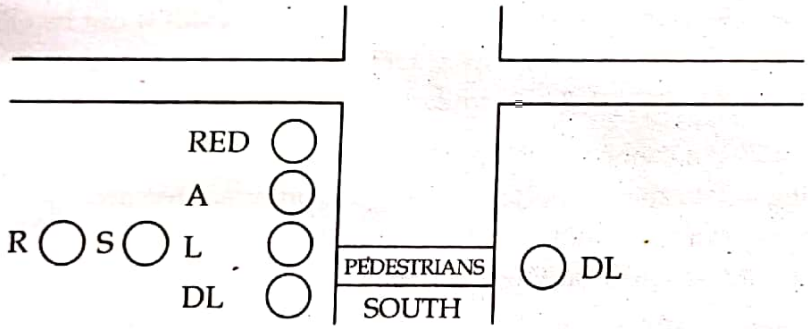
Microprocessor Interfacing Traffic Control

Aim:

- To study the hardware and software requirements of a traffic control using microprocessor kit.

Description of the circuit: The interface provides a set of 6 LEDs at each of the four corners of a four road junction. The organization of these LEDs is identical at each of the four corners. Hence, for simplicity, the organization is described below with reference to the LEDs at SOUTH-WEST corner only. The LEDs at SOUTH-WEST corner are organized as follows:

RED	:	Referred to as SOUTH RED henceforth
A	:	Referred to as SOUTH AMBER henceforth
L	:	Referred to as SOUTH LEFT henceforth
S	:	Referred to as SOUTH STRAIGHT henceforth
R	:	Referred to as SOUTH RIGHT henceforth
DL	:	Referred to as SOUTH PEDESTRAIN henceforth (DL refers to a set of two LED one on either side of the road).



Of these, the first five LEDs will be ON or OFF depending on the state of the corresponding port line (LED is ON the port line is Logic HIGH and LED is OFF if the port line is Logic LOW.) The last one marked as DL is a set of two dual-colour LEDs and they both will be either RED or GREEN depending on the state of a corresponding port line. (RED if the port line is logic HIGH and GREEN if the port line is logic LOW.)

There are four such sets of LEDs and these are controlled by 24 port lines. Each port line is inverted and buffered using 7406 (open collector inverter buffers) and is used to control an LED. Dual-colour LEDs are controlled by a port line and its complement.

The 24 LEDs and their corresponding port lines are summarized below:

	LED	PORT LINE		LED	PORT LINE
SOUTH	RED	PA3	NORTH	RED	PB3
	AMBER	PA2		AMBER	PB2
	LEFT	PA0		LEFT	PB0
	STRAIGHT	PC3		STRAIGHT	PC1
	RIGHT	PA1		RIGHT	PB1
	PEDESTRAIN	PC6		PEDESTRAIN	PC4
EAST	RED	PA7	EAST	RED	PB7
	AMBER	PA6		AMBER	PB6
	LEFT	PA4		LEFT	PB4
	STRAIGHT	PC2		STRAIGHT	PC0
	RIGHT	PA5		RIGHT	PB5
	PEDESTRAIN	PC7		PEDESTRAIN	PC5

User can assign any meaningful interpretation to these LED and then develop software accordingly. Usually the interpretation would be as follows:

Vehicles coming from one direction are controlled by LED at the opposite corner. For example, vehicles coming form NORTH are controlled by the set of LED at the SOUTH WEST corner as shown below:

Vehicles from NORTH can

go left (i.e., to EAST) if SOUTH LEFT LED is ON

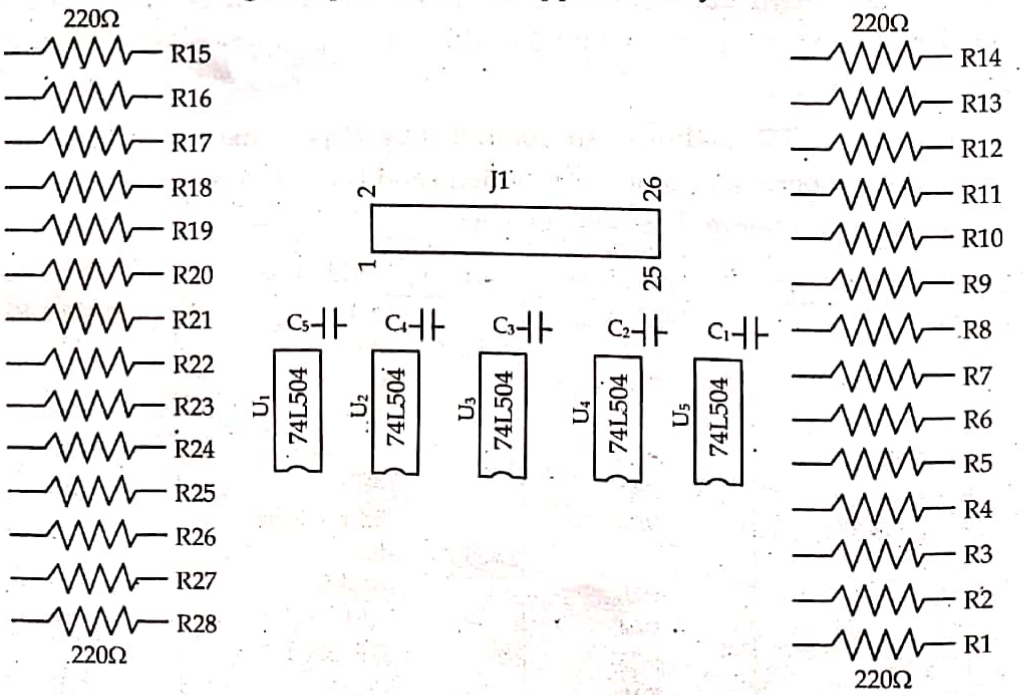
go right (i.e. to WEST) if SOUTH RIGHT LED is ON

go straight (i.e. to SOUTH) if SOUTH-STRAIGHT LED is ON.

Further the above movement are allowed only if SOUTH RED LED is OFF. If SOUTH RED LED is ON, no movement is allowed for vehicles from north. Pedestrian crossing on South is allowed when SOUTH PEDESTRAIN is green and disallowed when it is red. It is obvious that logically some combinations can not be allowed. For example SOUTH RED = OFF, SOUTH STRAIGHT = ON and SOUTH PEDESTRAIN = GREEN can not be allowed (vehicles are allowed to go from NORTH to SOUTH and pedestrians are allowed to cross on SOUTH!). SOUTH AMBER can be ON to indicate that SOUTH RED is about to change its status from OFF to ON. The movement of vehicles and pedestrians on other roads can be controlled in a similar way.

The interface module has a 26-pin connector at one edge of the road. This is used for connecting the module to the trainer using a flat cable connector. Further, this interface requires approximately 500 mA at 5V and this power is drawn from the trainer via this flat cable connector. The power supply connected to the trainer should be able to supply this extra current.

Connect the interface module to MPS-85 over the connector for J2 using a flat cable connector. The interface module can be J1 also. However the demonstration programming presented here assume that the connections is made over J2 and use the corresponding port addresses. If the connections is made over J1 user has to change the port addresses approximately.



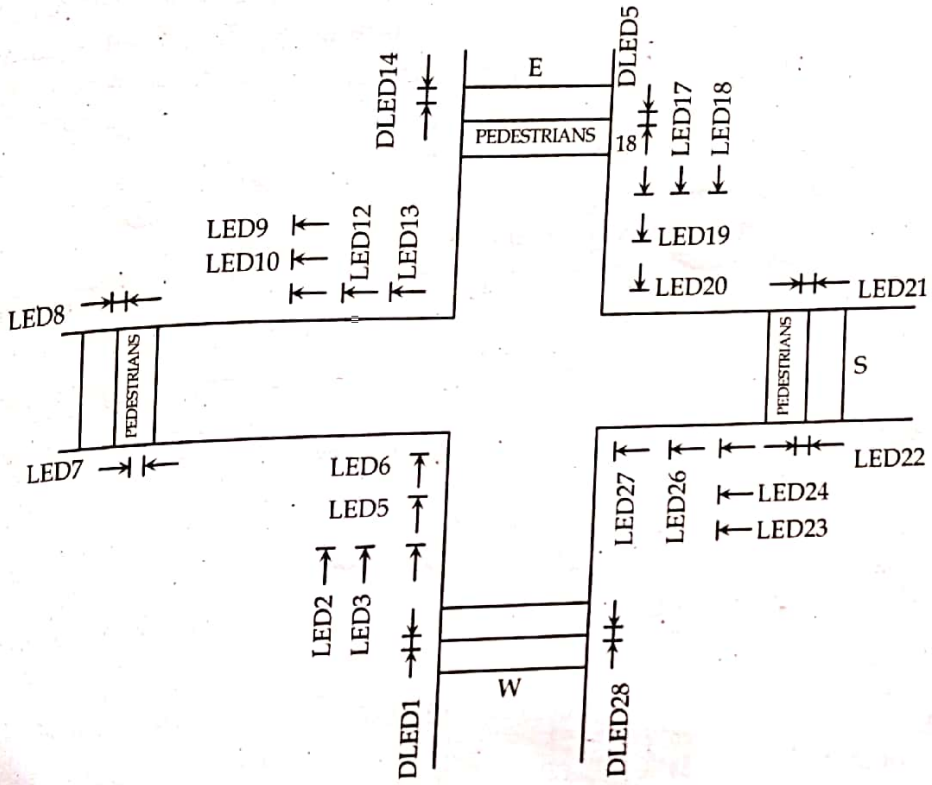


Figure:

Example:

The following sequence of simple traffic situation is simulated:

(Movements other than those listed are not allowed in a given situation)

- Vehicles from SOUTH can go North and west
  Vehicles from WEST can go NORTH
  Vehicles from NORTH can go SOUTH
  Pedestrians can cross on EAST
- Vehicles from EAST can go WEST and SOUTH
  Vehicles from WEST can go EAST
  Vehicles from SOUTH can go WEST
  Pedestrians can cross on NORTH
- Vehicles from EAST can go SOUTH
  Vehicles from NORTH can go SOUTH and EAST
  Vehicles from SOUTH can go NORTH
  Pedestrians can cross on WEST
- Vehicles from EAST can go WEST
  Vehicles from WEST can go EAST and NORTH
  Vehicles from NORTH can go EAST
  Pedestrians can cross on SOUTH
- No vehicle movement
  Pedestrians cross on all four roads

The system moves from one state to another state after a fixed time delay. The state transition is indicated by turning ON all the amber LED and all pedestrian red LED for a fixed duration. The sequence of the above states is repeated again and again. Hence, you must press the RESET key to allow the monitor program to regain control form this program.

; Simulates the traffic situations described in EXAMPLE.
; Assumes the interface is connected over J2.
; The traffic system from one state to next state after a fixed time delay.

Label	Instruction	Memory Address	Memory content Op-code	Low A/D	High A/D	Comments
	MVIA, 80H	8800		3E	80	; INITIALISE 8255.MODE0
	OUT 43H	8802	D3	43		; ALL PORTS AS OUTPUT PORTS
AGAIN:	LXIH, PORTS	8804	21	32	88	; TABLE OF PORT VALUES
NEXTST:	MOV A, M	8807	7E			; PORT A VALUE
	OUT 40H	8808	D3	40		
	INX H	880A	23			
	MOV A, M	880B	7E			
	OUT 41H	880C	D3	41		; PORT B VALUE
	INX H	880E	23			
	MOV A, M	880F	73			
	OUT 42H	8810	D3	42		; PORT C VALUE
	INXH	8812	23			
	CALL DELAY	8813	CD	IF	88	; SOME DELAY
	MOV A, L	8816	7D			
	CPI LOW TEND	8817	EF	50		; LOW=50 END OF PORT VALUES?
	JNZ NEXTST	8819	C2	07	88	; NO. NEXT STATE
	JMP AGAIN	881C	C3	04	88	; REPEAT THE WHOLE SEQUENCE
DELAY:	LXID, 0006H	881F	11	06	00	
	LY5: LXIB, FFFFH	8822	01	FF	FF	
	LY 10: DCXB	8825	0B			
	MOV A, C	8826	79			
	ORA B	8827	B0			
	JNZDLY 10	8828	C2	25	88	; INNER LOOP
	DCXD	882B	1B			
	MOV A, E	882C	7B			
	ORA D	882D	B2			
	JNZ DLY5	882E	C2	22	88	; OUTER LOOP
	RET	8831	C9			
PORTS:	DFB 10H, 81H, 7AH	8832	82	20	7A	; STATE 1
	DFB 44H, 44H, FOH	8835	44	44	F0	; ALL AMBERS ON
	DFB 08H, 11H, E5H	8838	22	08	E5	; STATE 2
	DFB 44H, 44H, F0H	883B	44	44	F0	; ALL AMBERS ON
	DFB 81H, 10H, DAH	883E	20	82	DA	; STATE 3
	DFB 44H, 44H, F0H	8841	44	44	F0	; ALL AMBERS ON
	DFB 11H, 08H, B5H	8844	08	22	B5	; STATE 4
	DFB 44H, 44H, F0H	8847	44	44	F0	; ALL AMBERS ON
	DFB 88H, 88H, 00H	884A	88	88	00	; STATES 5
	DFB 44H, 44H, F0H	884D	44	44	F0	; ALL AMBERS ON
TEND:	DFB 00H	8850	00			; DUMMY

Aim: To write assembly language programs for displaying:

- A six digit numerical data in the microprocessor kit using monitor program.
- An alphanumeric data in the address field and data field in the Microprocessor kit.
- An alphanumeric message in the display interface and assemble enter and execute them in the Microprocessor kit.

Program:

1. To display Six digit numerical data using monitor program.

Label	Instruction	Memory Address	Memory content Op-code	Low A/D	High A/D	Comments
START:	MVI E, 12H	8C00	1E	12		; Transfer the data (6 digit) to be
	MVI B, 34H	8C02	06	34		displayed to E, B and C registers.
	MVI C, 56H	8C04	0E	56	8C	; Call display sub routine program in
	CALL DISPLAY	8C06	CD	0C		8C0CH
	JMP START	8C09	C3	00	8C	; Go to the beginning
Display Subroutine						
DISPLAY:	MOV A, E	8C0C	7B			; Store the data in E register
	STA 8FF0	8C0D	32	F0	8F	; in location 8ff0h
	MOV A, B	8C10	78			Store the data in b register
	STA 8FEF	8C11	321	EF	8F	; in location 8fefh
	MOV A, C	8C14	79			Store the data in c register
	STA 8FF1	8C15	32	F1	8F	in location 8fefh
	MVIB, 00H	8C18	06	00		
	CALL 044C	8C1A	CD	4C	04	; Call monitor routine to display in the address field and data field.
	CALL 0440	8C1D	CD	40	04	
	RET	8C20	C9			

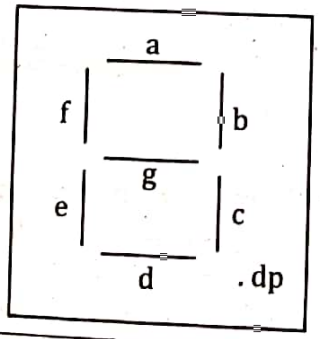
2. Program to display Alphanumeric data

One can display a message the address field and data field. The display format for the 7 segment is given below. If the bit is set '1' the particular segment will glow. If it is reset to '0' that segment will not glow.

Display format

D7	D6	D5	D4	D3	D2	D1	D0
d	c	b	a	dp	g	f	e

7 Segment LED



To display	(Binary)								(Hex)
J	1	1	1	0	0	0	0	1	→E1
O	1	1	1	1	0	0	1	1	→F3
S	1	1	0	1	0	1	1	0	→D6
E	1	0	0	1	0	1	1	1	→97
P	0	0	1	1	0	1	1	1	→37
H	0	1	1	0	0	1	1	1	→67

Store the hex code of the message along with the number of characters in memory location 8C50H onwards and use the subroutines at locations AFDY (4 characters) and DFDY (2 characters).

Label	Instruction	Memory Address	Memory content Op-code	Low A/D	High A/D	Comments
START:	MVIA, 90	8C20	3E	90		Get the control word for auto display (90H) in address field & place it in control port 31
	OUT 31	8C22	D3	31		
	CALLAFDY	8C24	CD	31	8C	Call address field display subroutine display subroutine
	MVIA, 94	8C27	3E	94		Get the control word for auto display (94H) in data field and place it in control port 31
	OUT 31	8C29	D3	31		
	CALL DFDY	8C2B	CD	41	8C	Call data field display subroutine
	JMP START	8C2E	C3	20	8C	Go to the beginnings
AFDY:	LXI H, 8C50	8C31				
	MOV B, M	8C35	21	50	8C	Place the address of the data in H-L pair and move the count value to the counter register B
LOOP 1;	INXH	8C35	46			
	MOVA, M	8C36	23			
	OUT30		7E			Display data in the address field.
	DCRB	8C37	D3	30		Decrement count
	JNZ LOOP1	8C39	05			If it is #0 Go to LOOP1
	RET	8C3A	C2	35	8C	If it is = 0 return to the main program
		8C3D	C9			
DFDY;	LXIH, 8C55	8C41	21	55	8C	Place the address of the data in H-L pair and move the count value to the counter register B
	MOV B, M	8C44	46			
LOOP2;	INXH	8c45	23			Display data in the data field.
	MOV A, M	8c46	7e			
	OUT 30	8c47	d3	30		
	DCR B	8c49	05			
	JNZ LOOP2	8c4a	c2	45	8c	If it is #0 Go to LOOP 2
	RET	8c4d	c9			If it is = 0 return to the main program
	DISDATA;		04			(No. of character to be displayed in the address field)
		8c50	e1	(J)		
		8c52	f3	(O)		
		8c53	d6	(S)		
		8c54	97	(E)		
		8c55	02			
		8c56	37	(P)		(No. of characters to be displayed in the data field)
		8c57	67	(H)		

3. Seven Segment Display Interface

Description: This interface provides a four digit 7 segment display driven by the outputs of four cascaded serial in parallel out shift registers. Data to be displayed is transmitted serially bit by bit to the interface over the port line PB0. Each bit its clocked into the shift registers by providing a common clock through the port line PC0. Thus information for all the four digits is provided by 32 bits clocked into the shift registers serially.

If can be seen from the circuit that the first bit after 32 clock pulses is available at H outputs of the shift register U5 and thus controls the H segment of the display DU0. The segments controlled by other bits can be determined similarly.

Display codes: Since the outputs of shift registers are connected to the cathode sides of LED segments low input must be given to the segments for making them glow and high inputs for making them blank. Each display has 7 bar segments and a dot (a, b, c, d, e, f, g, h) as shown in figure. For displaying any character its corresponding segments must be given low inputs.

Example: For displaying digit "3" segments a, b, c, d & g must be given two inputs (i.e. 0s) and the rest high inputs (i.e. 1s) as shown:

Segment:	h	g	f	e	d	c	b	a
Bit:	7	6	5	4	3	2	1	0
Input:	1	0	1	1	0	0	0	0
Hex code:			B					

Common Anode

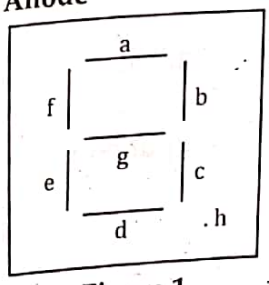


Figure 1.

Description of sample program: The sample program presented here displays the message "ELECTRO-SYS-TEMS" in a rotating fashion. In other words, it first blanks the display then displays "ELEC" then TRO-" then "SYS" and the "TEMS". These groups are displayed one after another with suitable delays. The program goes on repeating the entire display sequence. This program is written as an infinite loop, and hence user must press RESET key to recover from it. Interface is connected over J2 of microprocessor kit.

Label	Instruction	Memory Address	Memory content Op-code	Low A/D	High A/D	Comments
		8C00	3E	80		; Configure 8255 for mode 0
MVI A, 80H		8C02	D3	43		; All ports output
OUT 43H		8C04	21	40	8C	; Start of display codes
LOOP4	LXIH, STRING	8C07	16	05		; 5 groups
	MVI D, 05	8C09	06	04		; 4 characters/group
LOOP 3	MVI B, 04	8C0B	0E			; 8 segments/character
LOOP 2	MVI C, 08	8C0D	7E			; Get display code
	MOV A, M	8C0E	23			; Increment pointer
	INX H	8C1B	0D			; All bits over?
	DCR C	8C1C	C2	OF	8C	; No continue
	JNZ LOOP1	8C1F	05			; All character over
	DCR B	8C20	C2	0B	8C	; No continue
	JNZ LOOP2	8C23	CD	30	8C	; Introduce delay
	CALL DELAY	8C26	CD	30	8C	; Introduce delay
	CALL DELAY	8C29	15			; All groups over?
	DCR D	8C2A	C2	09	8C	; No, continue
	JNZ LOOP3	8C2D	C3	04.	8C	;Yes start from beginning
	JMP LOOP4	8C30	01	FF	FF	; Delay subroutine
DELAY:	LXI B, FFFFH	8C33	0B			
DLY1:	DCX B	8C34	78			
	MOV A, B	8C35	B1			
	ORA C	8C36	C2	33	8C	
	JNZ DLY1	8C39	C9			
	RET	8C40	FF	FF		; Blank, Blank
STRING:	DB FFH, FFH	8C42	FF	FF		Blank, Blank
	DB FFH, FFH	8C44	C6	86		"CE"
	DBC6H, 86H	8C46	C7	86		"LE"
	DB C7H, 86H	8C48	BF	C0		"-O"
	DB BFH, C0H	8C4A	DE	87		"RT"
	DB DEH, 87H	8C4C	BF	92		"-S"
	DB BFH, 92H	8C4E	91	92		"YS"
	DB 91H, 92H	8C50	92	C8		"SM"
	DB 92H, C8H	8C52	86	87		"ET"
	DB 86H, 87H					

Aim:

To generate square wave or pulse using microprocessor.

Description: A square wave or pulse can easily be generated by microprocessor. The microprocessor sends high and then low signals to generate square or pulse. A pulse or square wave can be generated using I/O port or SOD line or timer/counter (Intel 8253). To generate square wave or pulse using I/O port alone is discussed here.

To generate square wave connections are made as shown in Fig. 1. The pin PB0 of the port B of 8255-2 is used for taking output. This is connected to a buffer 7407. The final output is taken from the buffer terminal. The control used in the program is 98H make port B as an output port.

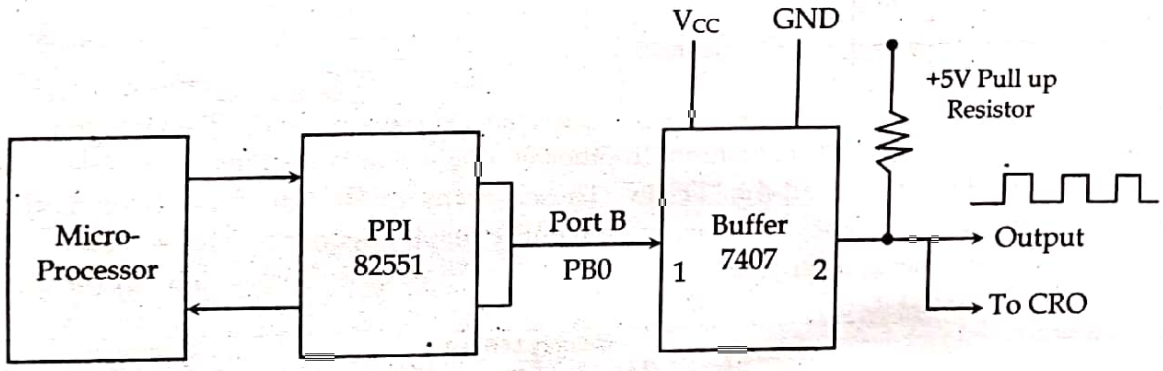


Figure 1

Label	Instruction	Memory Address	Memory content Op-code	Low A/D	High A/D	Comments
	MVI A, 98H	8E00	3E	98		; Get control word
	OUT CONTREG	8E02	D3	43		; Initialize ports
LOOP:	MVI A, 00	8E04	3E	00		
	OUTPORTB	8E06	D3	41		; Make PB0 Low
	CALL DELAY1	8E08	CD	20	8E	
	MVIA, 01	8E0B	3E	01		
	OUT PORTB	8E0D	D3	41		; Make PB0 High
	CALL DELAY2	8E0F	CD	30	8E	
	JMP LOOP	8E12	C3	04		
DELAY 1:	MVI B, 02	8E20	06	02		
GO:	DCR B	8E22	05			
	JNZ GO	8E23	C2	22	8E	
	RET	8E26	C9			
DELAY2:	MVI C, 02	8E30	OE	02		; Get count delay
BACK:	DCR C	8E32	OD			
	JNZ BACK	8E33	C2	32	8E	
	RET	8E36	C9			

Delay 1 control the time period for which the square wave remains LOW, i.e., zero. Delay2 controls the time for which the wave remains HIGH, i.e., 1. If the time period for LOW and HIGH are to be kept equal the counts in register B and register C are made equal. For such a case there is no need of two subroutines. Only one DELAY subroutine will be called at two places, i.e., at 8E08 and 8E0F memory addresses. There will be slight difference in timing of LOW and HIGH due to instruction

JMP LOOP. If accuracy is desired register C. The differences can also be minimized by interning two NOP instructions in DELAY1 subroutine. The instruction JMP LOOP has been used at the end of the program to repeat the whole process to generate square wave. It is not required when a pulse is to be generated. If generated pulse is to be observed on CRO, in that case JMP LOOP has to be sued to repeat the process. At many occasions it is required to send HIGH signals for certain duration to certain pins of IC. For such purpose the above program can also be written as:

```
MVI     A, 98H
OUT     CONTREG
MVI     A, 01H
OUT     PORT B
CALL    DELAY Duration of the pulse
MVI     A, 00
OUT     PORT B
```

Delay calculation

States	Instruction	How many times the instruction is executed	Total States
7	MVI R, 02	1	7×1
4	XX: DCR R	2	4×2
7/10	JNZ XX	2	10×1 + 7×1
10	RET	1	10×1
Total states = 7×1 + 4×2 + (10 + 7) + 10×1 (for DELAY1/DELAY2)			42

```
CALL DELAY1                 18
MVI A, 01                   7
OUT PORT B                  10
CALL DELAY2                 18
JMP LOOP                    10
MVI A, 00                   7
OUTPORT B                   10
DELAY1 + 2 = 84             10
                            Total states = 164
```

Time for one state for Intel 8085 is 320 ns

Period $T = 164 \times 320 \times 10^{-9}$ sec $= 52480 \times 10^{-9}$ sec

$T = 0.05248 \times 10^{-3}$ sec

$= 0.05248$ milli sec

$\therefore$ Frequency of the square wave $= 1/T \approx 19.05$ KHz.

❑❑❑

References

1. Ramesh S.Gaonkar, Microprocessor Architecture, Programming, and Applications with 8085, Prentice Hall
2. A.P.Malvino and J.A.Brown, Digital Computer Electronics, 3rd Edition, Tata McGraw Hill D.V.Hall, Microprocessors and Interfacing– Programming and Hardware, McGraw Hill
3. 8000 to 8085 Introduction to 8085 Microprocessor for Engineers and Scientists, A.K.Gosh, Prentice Hall
4. The Intel Microprocessors, Eighth Edition, Pearson, Barry B. Brey, Prentice Hall
5. "Microprocessor System the 8086 /8088 Family" by Liu and Gibson
6. "Advanced Microprocessor and Peripherals " by Bhurchandi K and A K Ray
7. "Microprocessors: Theory and Applications" by Rafiquzzaman

Model Question with Solution

Bachelor Level/ First Year/ Second Semester/ Science
Microprocessor (CSC 162)
Time: 3 hours.
Candidates are required to give their answers in their own words as for as practicable.
The figures in the margin indicate full marks.

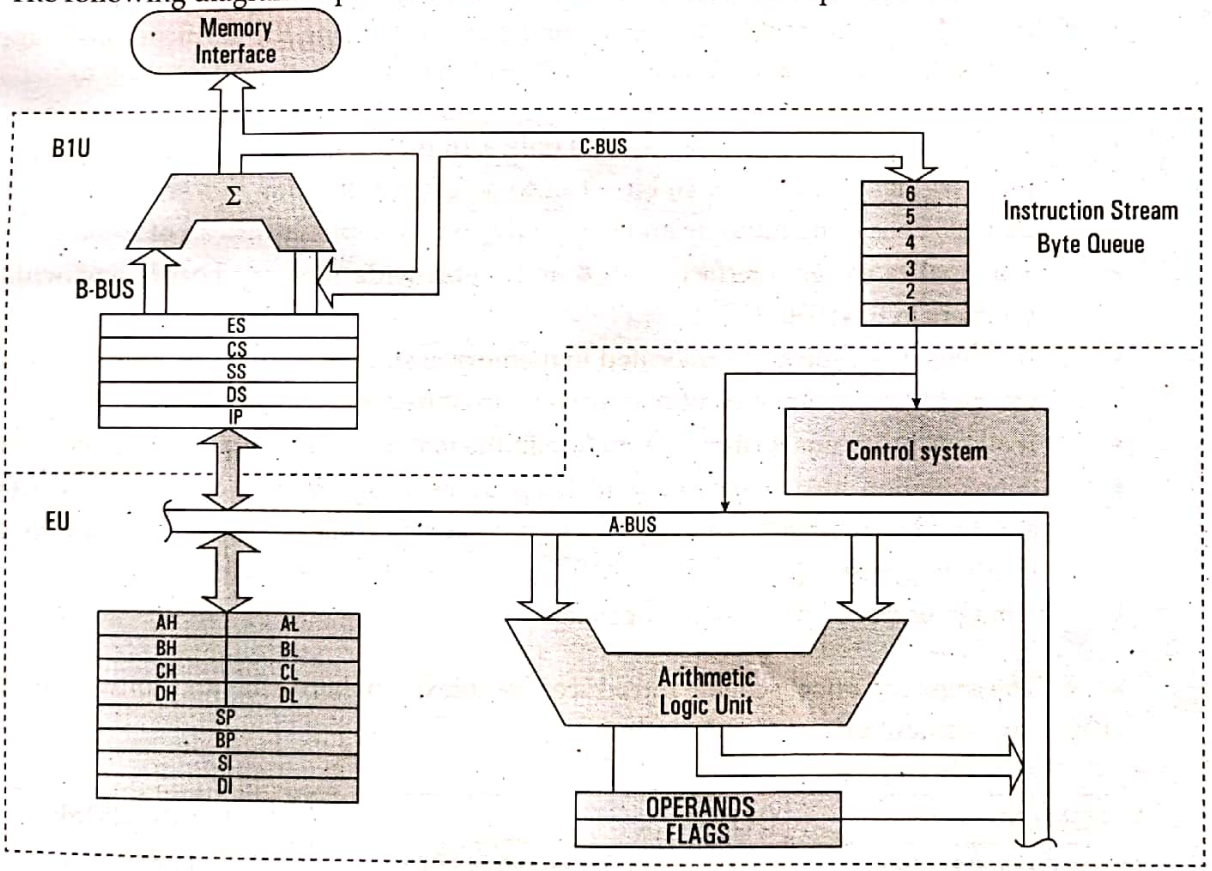
Full Marks: 60
Pass Marks: 24

Group A (Long Answer Question Section)

Attempt any TWO questions. (2x10=20)

1. Draw logical block diagram of 8086 microprocessor and explain its segmented memory structure.

Ans: The following diagram depicts the architecture of an 8086 Microprocessor:



8086 Microprocessor is divided into two functional units, i.e., EU (Execution Unit) and BIU (Bus Interface Unit).

Segmented Memory

- The 8086 BIU sends out 20-bit address so it can address any of 2^{20} or 1,048,576 bytes in memory.
- However at any given time the 8086 works with only four 65536 bytes (64 Kbyte) segment within this 1,048,576 byte (1 Mbyte) Range
- Four segments are : Code Segment, Stack Segment, Data Segment and Extra Segment

- Four segment registers in BIU are used to hold the upper 16 bits of the starting address of 4 memory segments that the 8086 is working with at a particular time.
- The 4 segment registers are code segment register (CS), stack segment register (SS), data segment register (DS) and the extra segment register (ES).
- For small programs which do not need all 64 Kbytes in each segment can overlap.
- For example, the code segment holds the upper 16 bits of the starting address for the segment from which the BIU is currently fetching instruction code bytes.
- The BIU always inserts zero for the lowest 4 bits of the 20-bit starting address.
- If the code segment register contains 348A H then the code segment will start at address 348A0 H
- A 64 Kbytes segment can be located anywhere within the 1 Mbyte address space, but the segment will always start at an address with zeros in the lowest 4 bits

Advantages of Segmentation [Segment: Offset Scheme]

Intel designed the 8086 family devices to access memory using the segment: offset approach rather than accessing memory directly with 20 bit. The advantages are listed below.

- The segment: offset scheme requires only a 16-bit number to represent the base address for a segment and only a 16 bit offset to access any location in a segment. This means that 8086 has to manipulate and store only 16-bit quantities instead of 20-bit quantities.
- This makes easier interface with 8 and 16-bit wide memory boards and with 16 bit registers in the 8086.
- It allows programs to be relocated in memory system. A relocatable program is one that can be placed in any area of memory and executed without change.
- It allows programs written to function in the real mode to operate in protected mode.
- Segmentation also makes easy to keep user's program and data separate from one another and segmentation makes it easy to switch from one user's program to another user's program.

Disadvantage of Segment: Offset Approach

- The segment: offset scheme introduces complexity in hardware and software design.

Different Segment Offset Combination

SEGMENT	OFFSET	SPECIAL PURPOSE
CS [CODE SEGMENT]	IP [INSTRUCTION POINTER]	INSTRUCTION
SS [STACK SEGMENT]	SP [STACK POINTER]	ADDRESS
	BP [BASE POINTER]	
DS [DATA SEGMENT]	BX [BASE REGISTER]	STACK
	DI [DESTINATION INDEX	
	SI [SOURCE INDEX]	
	8 BIT NUMBER	ADDRESS
	16 BIT NUMBER	
ES [EXTRA SEGMENT]		

What is machine cycle and instruction cycle? Draw a timing diagram for STA 2000h memory instruction. (Choose any memory locations for loading STA 2000h instruction)

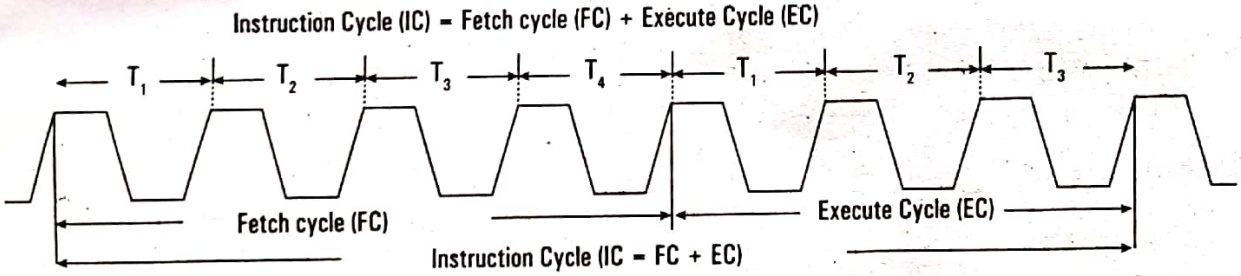
Ans: Machine cycle:

It is defined as the time required to complete one operation of accessing memory i/p, o/p or acknowledging and external request. This cycle may consists of 3 to 6 T states. T-states: It is defined as one sub division of the operation performed in one clock period. These sub division are internal states synchronized with system clock and each T states precisely equal to one clock period.

Instruction cycle:

The necessary steps that the CPU carries out to fetch an instruction and necessary data from the memory and to execute it constitute and instruction cycle it is defined as the time required to complete the execution of an instruction.

An instruction cycle consists of fetch cycle and execute cycle. In fetch cycle CPU fetches opcode from the memory. The necessary steps which are carried out to fetch an opcode from memory constitute a fetch cycle. The necessary steps which are carried out to get data if any from the memory and to perform the specific operation specified in an instruction constitute and execute cycle. The total time required to execute an instruction given by IC = Fc+ Ec the 8085 consists of 1-6 machine cycles or operations.



Timing diagram of STA 2000H

- STA means Store Accumulator- The contents of the accumulator is stored in the specified address(2000).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH.
- Then the lower order memory address is read(00). - *Memory Read Machine Cycle*
- Read the higher order memory address (20). - *Memory Read Machine Cycle*
- The combination of both the addresses are considered and the content from accumulator is written in 2000. - *Memory Write Machine Cycle*
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 2000.

Address	Mnemonics	Op code
41FF	STA 2000	32H
4200		00H
4201		20H

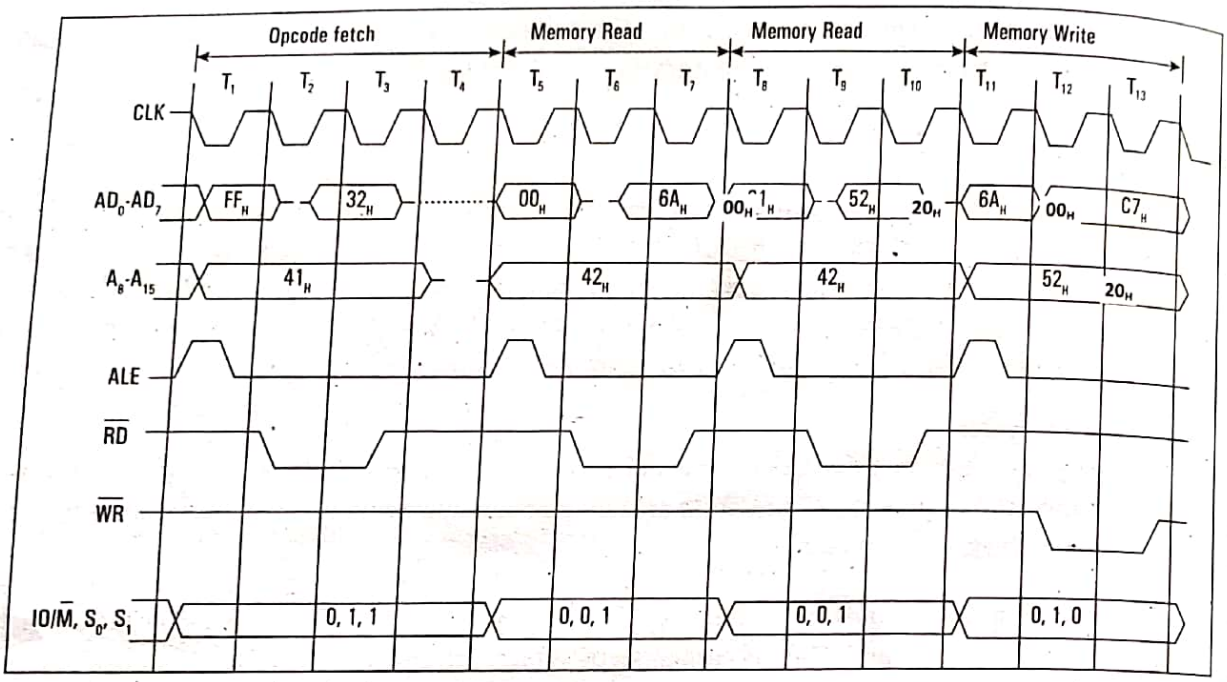


Figure: Timing diagram of instruction STA 2000H.

3. Write an assembly language program to sort an array in ascending order using 8 bit microprocessor. (Assume appropriate array data and address where minimum array size of 10 should be considered)

Ans:

LXIH, 8D10H	8C20	21	10	8D	Address for count
MOV C, M	8C23	4E			Count in C register
DCR C	8C24	OD			Count for number of passes in C register i.e. count-1
BACK MOV D, C	8C25	51			Count for number of comparisons in D register
LXIH, 8D11H	8C26	21	11	8D	Starting address of the array of data in H-L register pair
MOV A, M	8C29	7E			1st number in accumulator
LOOP: INX H	8C2A	23			Address of next number
MOV B, M	8C2B	46			Next number in B register
CMP B	8C2C	B8			Compare next number with the previous greatest number in ACC
JNC AHEAD	8C2D	D2	36	8C	If previous greatest number> next number, go to AHEAD
DCX H	8C30	2B			Address of previous memory location
MOV M, A	8C31	77			Place smaller of the two compared numbers in memory
MOV A, B	8C32	78			Place greater of the two numbers in accumulator
JMP GO	8C33	C3	38	8C	Branch to step labeled GO.
AHEAD: DCXH	8C36	2B			Address of previous memory location
MOV M, B	8C37	70			Place smaller of the two number in memory

GO:	IN X H	8C38	23			Address of next memory location
	DCR D	8C39	15			Decrease the count for comparisons
	JNZ LOOP	8C3A	C2	2A	8C	Branch to stop labeled LOOP if the count is not equal to 0.
	MOV M, A	8C3D	77			Place the greatest number after a pass in the memory
	DCR C	8C3E	OD			Decrease the count for passes
	JNZ BACK	8C3F	C2	25	8C	If count for passes = 0, go the BACK
	HLT	8C42	76			Other wise, stop.

Group B (Short Answer Question Section)

Attempt any EIGHT questions. **(8x5=40)**

4. Draw pin diagram of 8085 microprocessor with appropriate labelling.

Ans: PIN Configuration of 8085

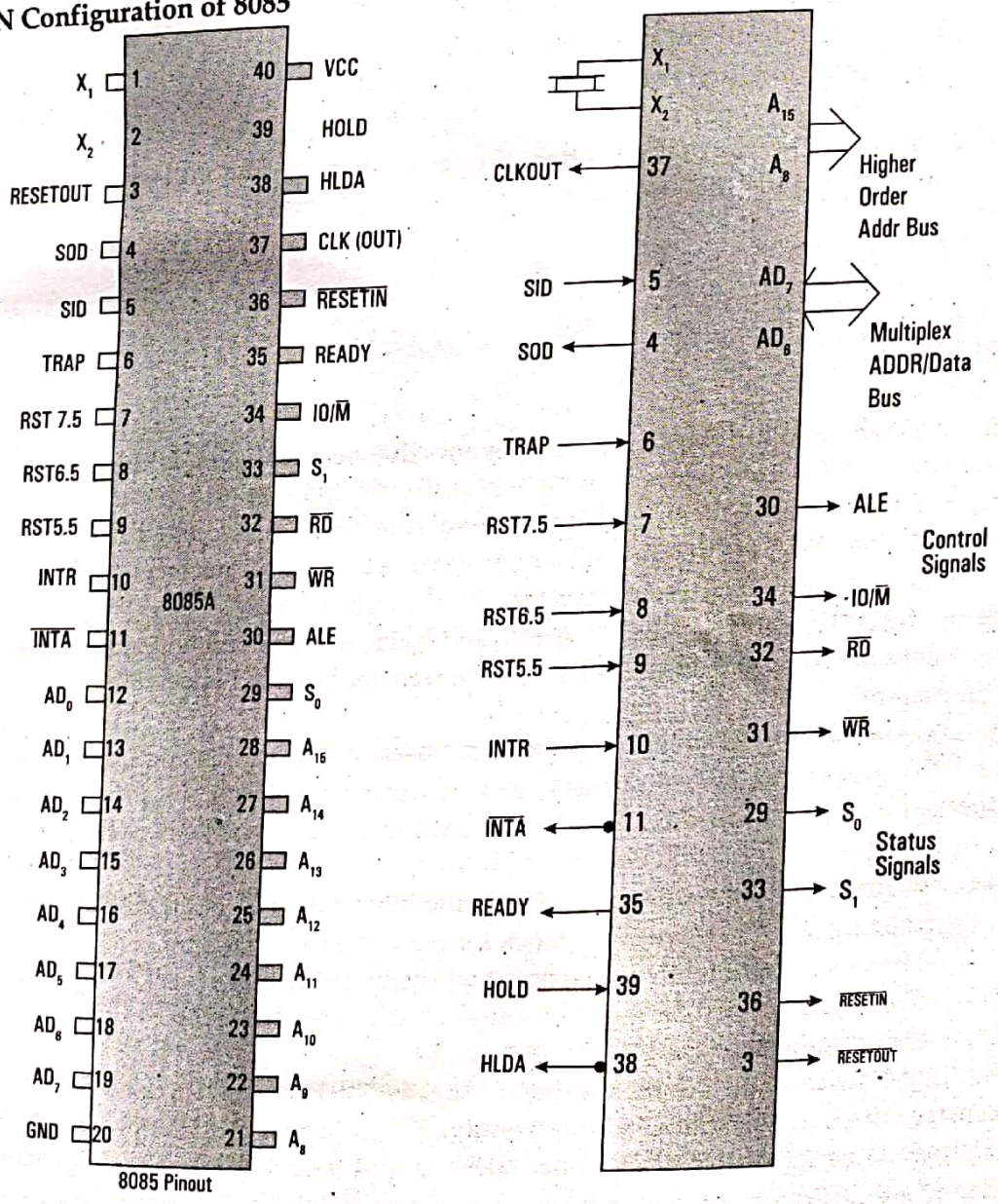


Figure: PIN Configuration of 8085

- The microprocessor is a clock-driven semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale integration (LSI) or very-large-scale integration (VLSI) technique.
- The microprocessor is capable of performing various computing functions and making decisions to change the sequence of program execution.
- In large computers, a CPU implemented on one or more circuit boards performs these computing functions.
- The microprocessor is in many ways similar to the CPU, but includes the logic circuitry, including the control unit, on one chip.
- The microprocessor can be divided into three segments for the sake clarity, arithmetic/logic unit (ALU), register array, and control unit.
- 8085 is a 40 pin IC, DIP package. The signals from the pins can be grouped as follows:
  1. Power supply and clock signals
  2. Address bus
  3. Data bus
  4. Control and status signals
  5. Interrupts and externally initiated signals
  6. Serial I/O ports

5. Specify the output in PORT1 after the execution of the following program. Write comments for each instruction.

```
MVI A, AAH
MOV B, A
RRC
XRA B
OUT PORT1
HLT
```

6. What is DMA? Explain the sequence of events that occurs during DMA operation?

Ans: The data transfer technique in which peripherals manage the memory buses for direct interaction with main memory without involving the CPU is called direct memory access (DMA). Using DMA technique large amounts of data can be transferred between memory and the peripheral without severely impacting CPU performance.

During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device(s) and main memory.

The DMA request CPU to handle control of buses to the DMA using bus request (BR) signal. The CPU grants the control of buses to DMA using bus grant (BG) signal after placing the address bus, data bus and read and write lines into high impedance state (which behave like open circuit).

CPU initializes the DMA by sending following information through the data bus.

1. Starting address of memory block for read or write operation.
2. The word count which is the no. of words in the memory block.
3. Control to specify the mode of transfer such as read or write.
4. A control to start the DMA transfer.

The DMA takes control over the buses directly interacts with memory and I/O units and transfers the data without CPU intervention. When the transfer completes, DMA disables the BR line. Thus CPU disable BG line, takes control over the buses and return to its normal operation.

The DMA transfer operation is illustrated below:

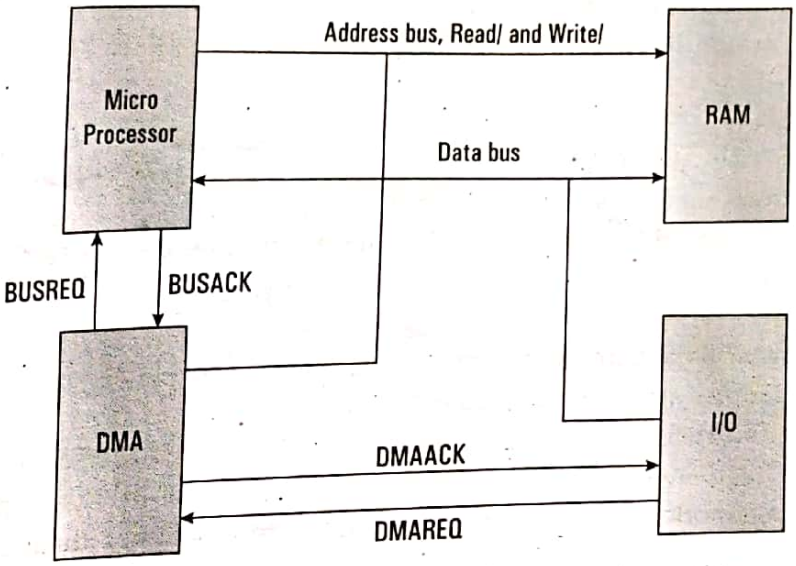


Fig: Illustration for DMA transfer in a computer system

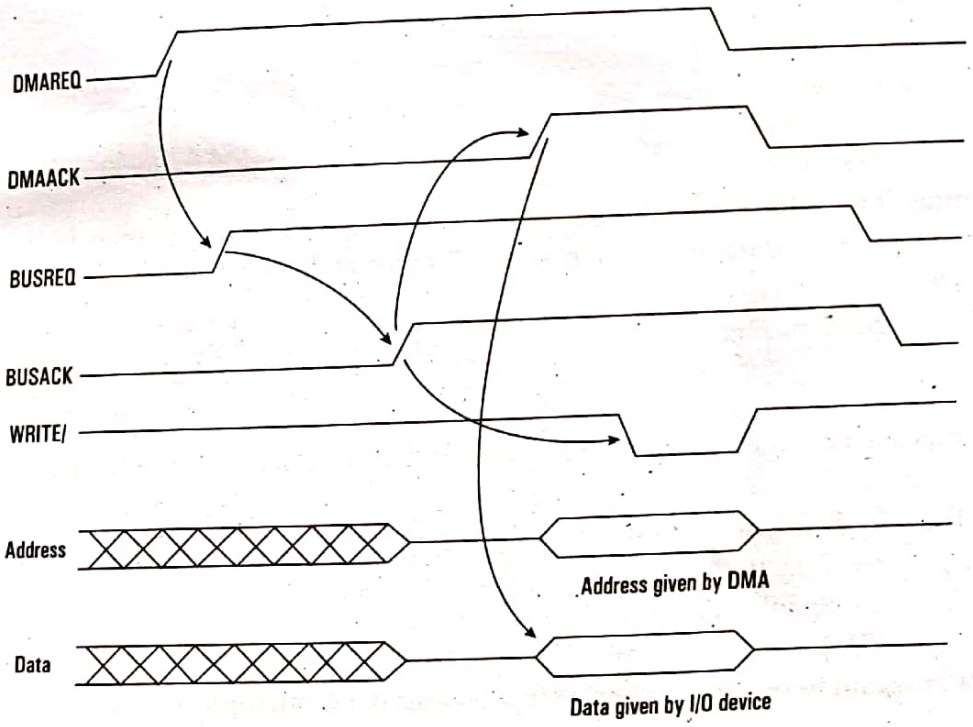


Figure: DMA Timing Diagram

7. **What is addressing mode? Explain different addressing mode in 8085 microprocessor.**

Ans: **Addressing modes:**

Instructions are command to perform a certain task in microprocessor. The instruction consists of op-code and data called operand. The operand may be the source only, destination only or both of them. In these instructions, the source can be a register, a memory or an input port. Similarly, destination can be a register, a memory location, or an output port. The various format (way) of specifying the operands are called addressing mode. So addressing mode specifies where the operands are located rather than their nature. The 8085 has 5 addressing mode:

1) **Direct addressing mode:**
The instruction using this mode specifies the effective address as part of instruction. The instruction size either 2-bytes or 3-bytes with first byte op-code followed by 1 or 2 bytes of address of data.

E. g. LDA 9500H A ← [9500]
IN 80H A ← [80]

This type of addressing is called absolute addressing.

2. **Register Direct addressing mode:** This mode specifies the register or register pair that contains the data.
E g. MOV A, B
Here register B contains data rather than address of the data.
Other examples are: ADD, XCHG etc.

3. **Register Indirect addressing mode:** In this mode the address part of the instruction specifies the memory whose contents are the address of the operand. So in this type of addressing mode, it is the address of the address rather than address itself. (One operand is register)
e. g. MOV R, M
MOV M, R
STAX, LDAX etc.
STAX B
B= 95 C =00
[9500] ← A

4. **Immediate addressing mode:** In this mode, the operand position is the immediate data. For 8-bit data, instruction size is 2 bytes and for 16 bit data, instruction size is 3 bytes.
E.g. MVI A, 32H
LXI B, 4567H

5. **Implied or Inherent addressing mode:** The instructions of this mode do not have operands.
E.g. NOP: No operation
HLT: Halt
EI: Enable interrupt
DI: Disable interrupt

8. **Write a program to reverse a given a string using 16 bit microprocessor.**

Ans:
```
.MODEL SMALL
.STACK
.DATA
STRING DB '!EMOC-LEW'
REVERSE DB 9 DUP(' ')
.CODE
;-------------------------------------------------------------
MAIN PROC
MOV AX,@DATA
MOV DS,AX
```

```
LEA SI,STRING  ;Load Effective Address of STRING into SI
   LEA DI,REVERSE ;Load Effective Address of REVERSE into DI
   ADD DI,9       ;Add the Address of DI With 9
   MOV CX,9       ;Initialize Counter as 9
   TOP:
      MOV AL,[SI] ;Content of SI to AL
      MOV [DI],AL ;Content of AL to Content of DI
      INC SI      ;Increment SI
      DEC DI      ;Decrement DI
      LOOP TOP    ;Loop Until CX!=0
   ADD DI,10  ;Add DI With 10 to Locate to End
   MOV AL,'$'         ;Add String Termination Character
   MOV [DI],AL
   MOV AH,09H    ;AH=09 Specifies String
   LEA DX,REVERSE ;Load Effective Address of REVERSE into DX
   INT 21H            ;DOS Interrupt Function
   MOV AH,4CH
   INT 21H
MAIN ENDP
END MAIN
```

9. **Explain memory interfacing in 8085 microprocessor along with appropriate diagram.**

Ans: The programs and data that are executed by the microprocessor have to be stored in ROM/EPROM and RAM, which are basically semiconductor memory chips.

Microprocessor need to access memory quite frequently to read instructions and data stored in memory, the interface circuit enables that access.

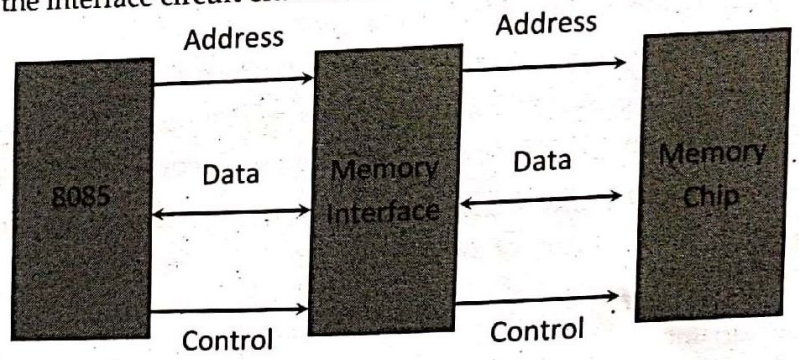


Figure: 8085 interfacing with memory chips.

The interface process involves designing a circuit that will match the memory requirements with the microprocessor signal.

Memory has certain signal requirements to read from and write into memory. Similarly Microprocessor initiates the set of signals when it wants to read from and write into memory.

- 8085 has 16 address lines (AO – A15), hence a maximum of 64 KB (= 216 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/M, RD' and 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), OE or RD' (output enable or read) and WE or 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 IOM, RD and WR signals as shown. Status of IQ/M, RD' and WR' signals during memory read and write operations

IOM'	RD'	WR'	Operation
0	0	1	8085 reads data from memory
0	1	0	8085 writes data into memory

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

Generating Control Signals

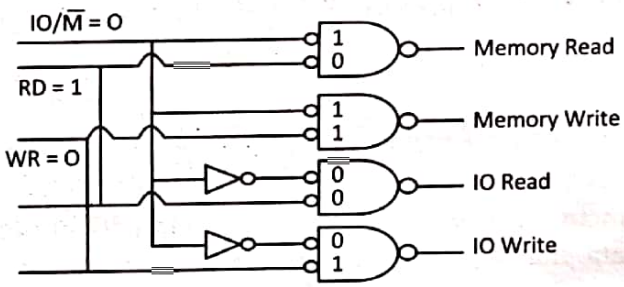


Figure: Generating Control Signals

- 8085 places 16-bit address on address bus and with this address only one register should be selected (only 11 low order address lines are required).
- Remaining 8085 address lines (A15-A11) should be decoded to generate chip select.
- 8085 provides two signal-IOM' and RD'-to indicate that is memory read operation MEMR'. (Similarly signal-IO/M' and WR'- indicates memory write operation MEMW').

Primary function of memory interfacing is that the microprocessor should be able to read from and write into a given register of a memory chip:

- Select the chip
- Identify the register
- Enable the appropriate buffer.

10. What are different operating modes in 80286 microprocessor? Explain in brief about each mode.

Ans: Real Address Mode

- Act as a fast 8086
- Instruction set is upwardly compatible
- It address only 1 M byte of physical memory using A0-A19.
- In real addressing mode of operation of 80286, it just acts as a fast 8086. The instruction set is upward compatible with that of 8086.

The 80286 addresses only 1Mbytes of physical memory using A0- A19. The lines A20-A23 are not used by the internal circuit of 80286 in this mode. In real address mode, while addressing

the physical memory, the 80286 uses BHE along with A0- A19. The 20-bit physical address is again formed in the same way as that in 8086.

The contents of segment registers are used as segment base addresses. The other registers, depending upon the addressing mode, contain the offset addresses. Because of extra pipelining and other circuit level improvements, in real address mode also, the 80286 operates at a much faster rate than 8086, although functionally they work in an identical fashion. As in 8086, the physical memory is organized in terms of segments of 64Kbyte maximum size.

An exception is generated, if the segment size limit is exceeded by the instruction or the data. The overlapping of physical memory segments is allowed to minimize the memory requirements for a task. The 80286 reserves two fixed areas of physical memory for system initialization and interrupt vector table. In the real mode the first 1Kbyte of memory starting from address 0000H to 003FFH is reserved for interrupt vector table. Also the addresses from FFFF0H to FFFFFH are reserved for system initialization.

The program execution starts from FFFFH after reset and initialization. The interrupt vector table of 80286 is organized in the same way as that of 8086. Some of the interrupt types are reserved for exceptions, single-stepping and processor extension segment overrun, etc.

When the 80286 is reset, it always starts the execution in real address mode. In real address mode, it performs the following functions: it initializes the IP and other registers of 80286, it prepares for entering the protected virtual address mode.

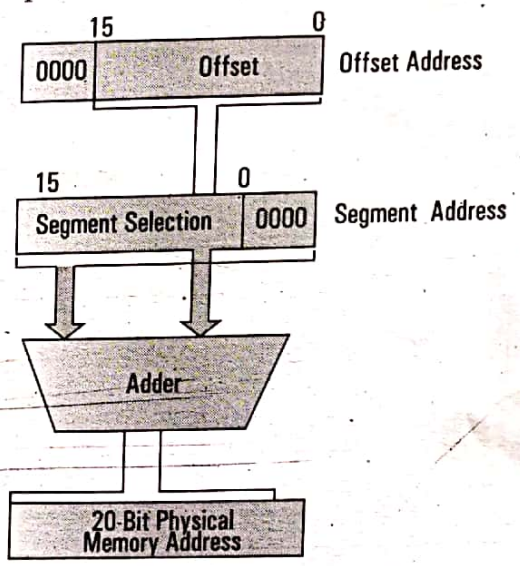


Figure: Real addressing mode

Protected Virtual Address Mode (PVAM)

80286 is the first processor to support the concepts of virtual memory and memory management. The virtual memory does not exist physically it still appears to be available within the system. The concept of VM is implemented using Physical memory that the CPU can directly access and secondary memory that is used as a storage for data and program, which are stored in secondary memory initially.

The Segment of the program or data required for actual execution at that instant is fetched from the secondary memory into physical memory. After the execution of this fetched segment, the next segment required for further execution is again fetched from the secondary

memory, while the results of the executed segment are stored back into the secondary memory for further references. This continues till the complete program is executed

During the execution the partial results of the previously executed portions are again fetched into the physical memory, if required for further execution. The procedure of fetching the chosen program segments or data from the secondary storage into physical memory is called swapping. The procedure of storing back the partial results or data back on the secondary storage is called unswapping. The virtual memory is allotted per task.

The 80286 is able to address 1 G byte (230 bytes) of virtual memory per task. The complete virtual memory is mapped on to the 16Mbyte physical memory. If a program larger than 16Mbyte is stored on the hard disk and is to be executed, if it is fetched in terms of data or program segments of less than 16Mbyte in size into the program memory by swapping sequentially as per sequence of execution.

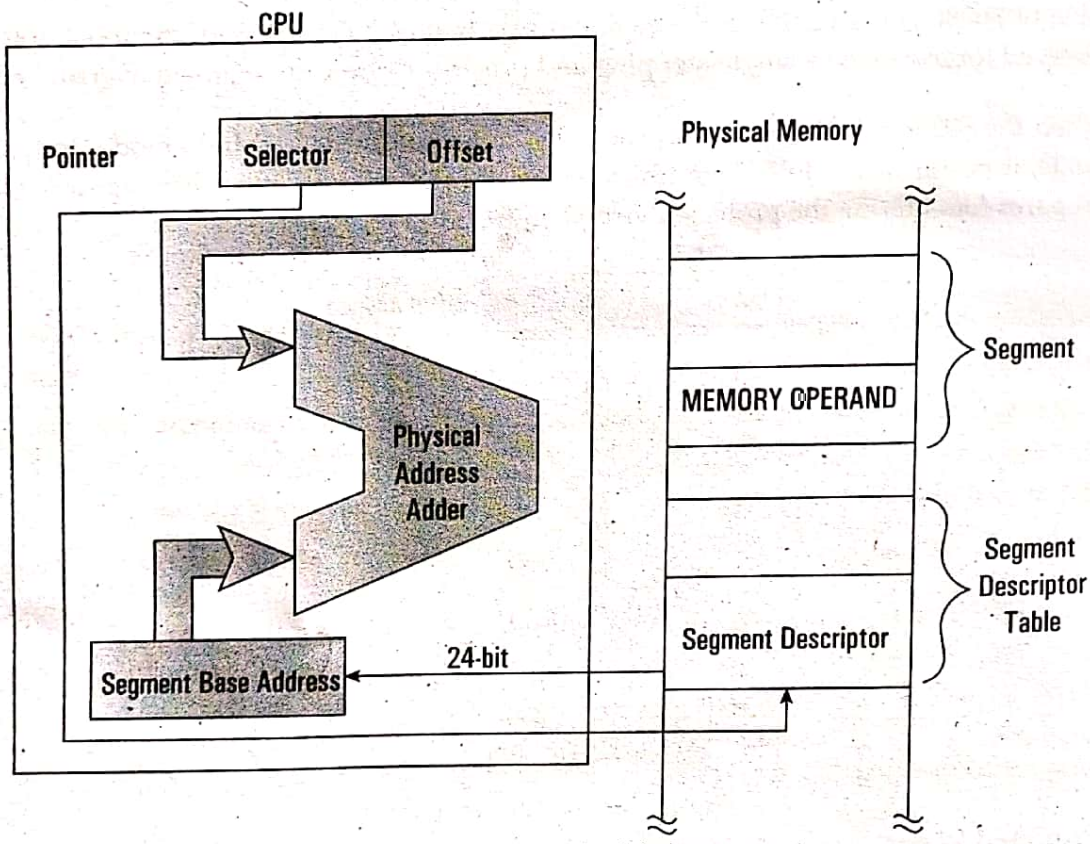


Figure: Protected virtual addressing mode

Whenever the portion of a program is required for execution by the CPU, it is fetched from the secondary memory and placed in the physical memory is called swapping in of the program. A portion of the program or important partial results required for further execution, may be saved back on secondary storage to make the PM free for further execution of another required portion of the program is called swapping out of the executable program.

80286 uses the 16-bit content of a segment register as a selector to address a descriptor stored in the physical memory. The descriptor is a block of contiguous memory locations containing information of a segment, like segment base address, segment limit, segment type, privilege level, segment availability in physical memory, descriptor type and segment use another task.

11. **"Interrupt based I/O is efficient compared to polled I/O". Justify this statement with general working mechanism in both methods.**

Ans: Polled interrupt:

Polled interrupt are handled using software and are there fores lower compared to vectored (hardware) interrupts. In this method there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupts sources in sequence. The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first, and if its interrupt signal is on, control branches to a servicer outline for this source. Otherwise the next lower priority source is tested, and so on. Thus, the initial service routine for all interrupts consists of a program that test the interrupt sources in sequence and branches to one of many possible service routines.

Polled interrupts are very simple. But or large number of devices, the time required to poll each device may exceed the service to the device. In such case, the faster mechanism called chained interruptisused.

Chained interrupt:

This is hardware concept to f handling the multiple interrupts. In this technique, the devices are connected in a chain fashion as shown in figure below for setting up the priority system.

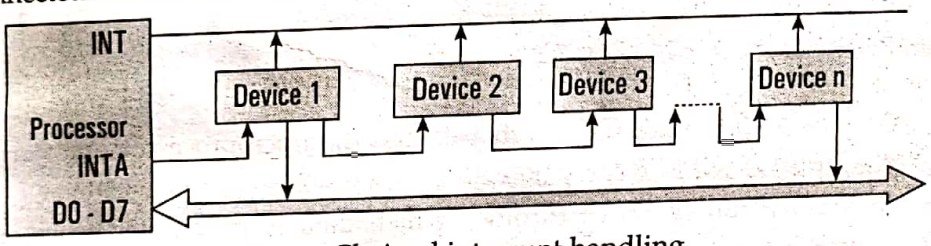


Figure: Chained interrupt handling.

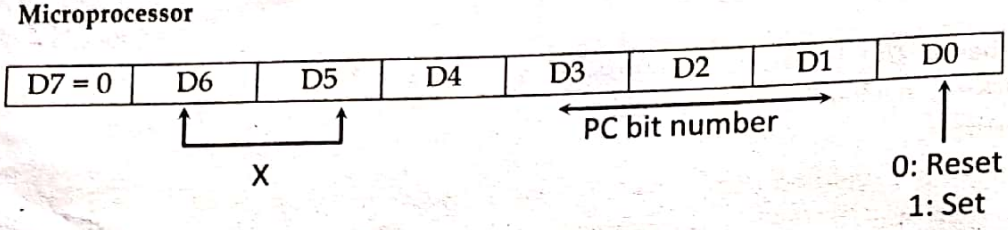
Here the device with the highest priority placed in the first position, followed by lower priority devices. Suppose that one or more devices interrupt the process or at a time. In response, the process or saves its current status and then generates an interrupt acknowledge (INTA) signal to the highest priority device, which is device 1 in our case. If this device has generated the interrupt it will accept the INTA signal from the processor; otherwise, it will pass INTA on to the next device until the INTA is accepted by the interrupting device.

Once accepted, the device provides a means to the process or for finding the interrupt address vector using external hardware. Usually there questing device responds by placing a word on the data lines. With the help of hardware it generates interrupts vector address. This word is referred to as vector, which the process or used as a pointer to the appropriate device service routine.

This avoids the need to execute a general interrupt service routine first. So this technique is also referred to as vectored interrupts.

12. **Write Short Notes (Any Two):**
a) Macro Assembler
b) BSR Mode

Ans: Bit set reset (BSR) mode – This mode is used to set or reset the bits of port C only, and selected when the most significant bit (D7) in the control register iXs 0. Control Register is as follows:



D3	D2	D1
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	0
1	1	0
1	1	1

This mode affects only one bit of port C at a time because, as user set the bit, it remains set until and unless user changes it. User needs to load the bit pattern in control register to change the bit.

c) System Bus

Ans: **System Bus:** It is a communication path between the microprocessor and peripherals; it is nothing but a group of wires to carry bits.

Data Bus : A collection of wires through which data is transmitted from one part of a computer to another is called Data Bus. Data Bus can be thought of as a highway on which data travels within a computer. This bus connects all the computer components to the CPU and main memory. The size (width) of bus determines how much data can be transmitted at one time.

E.g.:

- A 16-bit bus can transmit 16 bits of data at a time.
- 32-bit bus can transmit 32 bits at a time.

Address Bus

A collection of wires used to identify particular location in main memory is called Address Bus. In other words, the information used to describe the memory locations travels along the address bus. The size of address bus determines how many unique memory locations can be addressed.

E.g.:

- A system with 4-bit address bus can address 24 = 16 Bytes of memory.
- A system with 16-bit address bus can address 216 = 64 KB of memory.
- A system with 20-bit address bus can address 220 = 1 MB of memory.

Control Bus: The connections that carry control information between the CPU and other devices within the computer is called Control Bus. The control bus carries signals that report the status of various devices.

E.g.:

- This bus is used to indicate whether the CPU is reading from memory or writing to memory.

❑❑❑

Tribhuvan University 2075

Bachelor Level/ First Year/ Second Semester/ Science — Full Marks: 60

Microprocessor (CSC 162) — Pass Marks: 24

Time: 3 hours.

Candidates are required to give their answers in their own words as for as practicable.

The figures in the margin indicate full marks.

Group A (Long Answer Question Section)

Attempt any TWO questions. (2x10=20)

1. Draw the block diagram of 80286 microprocessor and explain its functional units.

Ans: The functional block diagram of 80286 microprocessor is as given below:

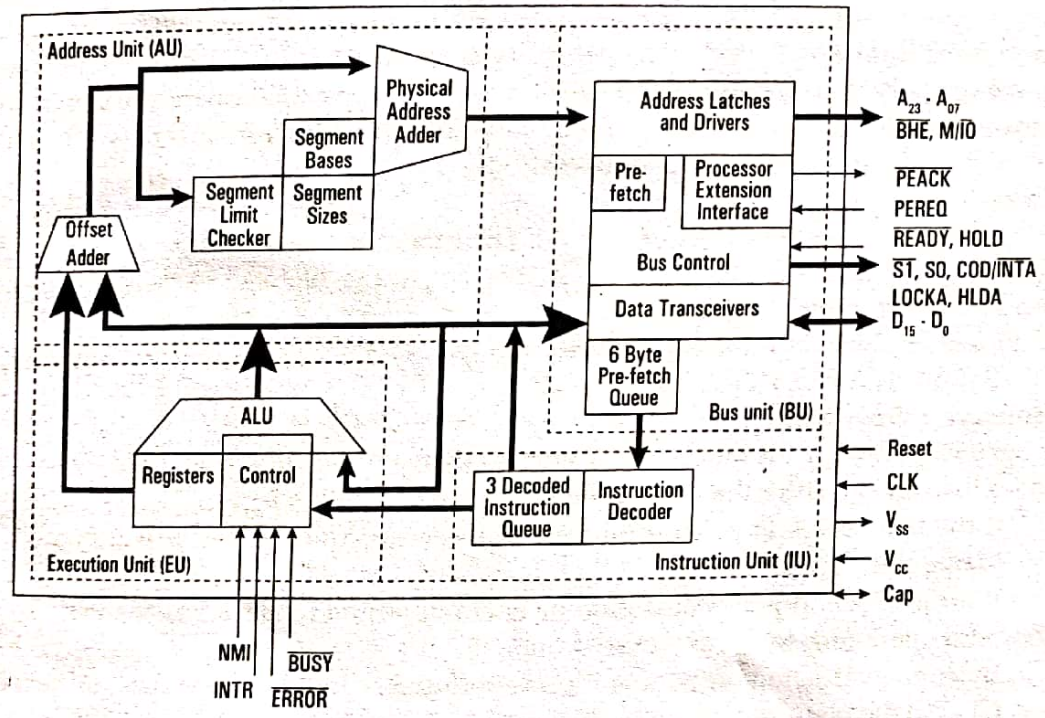


Figure: Internal Block Diagram of 80286

The CPU contain four functional blocks

1. Address Unit (AU)
2. Bus Unit (BU)
3. Instruction Unit (IU)
4. Execution Unit(EU)

The address unit is responsible for calculating the physical address of instructions and data that the CPU wants to access. Also the address lines derived by this unit may be used to address different peripherals. The physical address computed by the address unit is handed over to the bus unit (BU) of the CPU. Major function of the bus unit is to fetch instruction bytes from the memory. Instructions are fetched in advance and stored in a queue to enable faster execution of the instructions. The bus unit also contains a bus control module that controls the prefetcher module. These prefetched instructions are arranged in a 6-byte instructions queue. The 6-byte prefetch queue forwards the instructions arranged in it to

the instruction unit (IU). The instruction unit accepts instructions from the prefetch queue and an instruction decoder decodes them one by one. The decoded instructions are latched onto a decoded instruction queue. The output of the decoding circuit drives a control circuit in the execution unit, which is responsible for executing the instructions received from decoded instruction queue. The decoded instruction queue sends the data part of the instruction over the data bus. The EU contains the register bank used for storing the data as scratch pad, or used as special purpose registers. The ALU, the heart of the EU, carries out all the arithmetic and logical operations and sends the results over the data bus or back to the register bank.

2. Explain instruction cycle, machine cycle and T-states? Draw a timing diagram for STA instruction. Make necessary assumptions.

Ans: Instruction cycle

The necessary steps that the CPU carries out to fetch an instruction and necessary data from the memory and to execute it constitute and instruction cycle it is defined as the time required to complete the execution of an instruction.

An instruction cycle consists of fetch cycle and execute cycle. In fetch cycle CPU fetches opcode from the memory. The necessary steps which are carried out to fetch an opcode from memory constitute a fetch cycle. The necessary steps which are carried out to get data if any from the memory and to perform the specific operation specified in an instruction constitute and execute cycle. The total time required to execute an instruction given by IC = Fc+ Ec the 8085 consists of 1-6 machine cycles or operations.

Fetch cycle:

The first byte of an instruction is its opcode. The program counter keeps the memory address of the next instruction to be executed in the beginning of fetch cycle the content of the program counter, which is the address of the memory location where opcode is available, is send to the memory. The memory places the opcode on the data bus so as to transfer it to CPU. The entire process takes 3 clock cycle.

Execute cycle/Operation:

The opcode fetched from the memory goes to IR from the IR it goes to the decoder which decodes instruction. After the instruction is decoded execution begins.

- If the operand is in general purpose register, execution is performed immediately. I,e in one clock cycle.
- If an instruction is contains data or operand address, then CPU has to perform some read operations to get the desired data.
- In some instruction write operation is performed. In write cycle data are sent from the CPU to the memory of an o/p device.
- In some cases execute cycle may involve one or more read or write cycle or both.

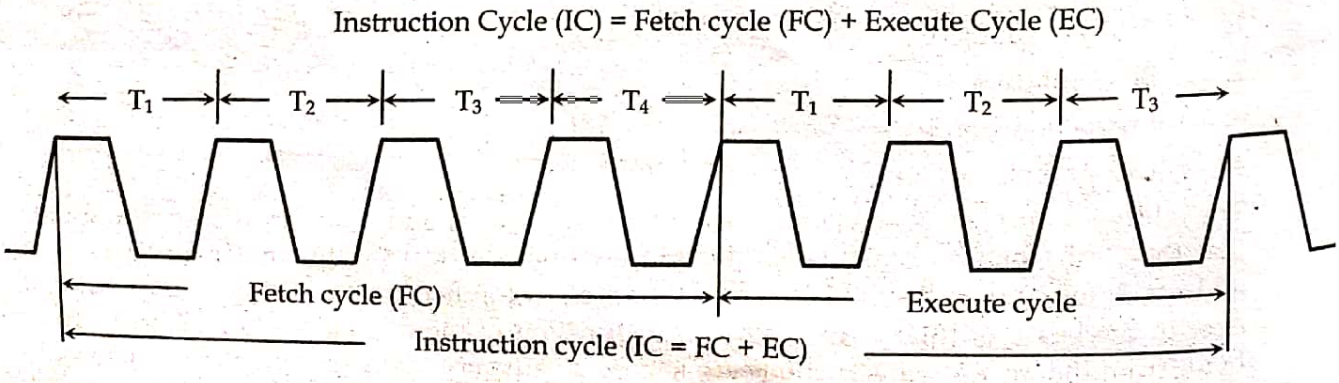


Figure: (a) Processor cycle

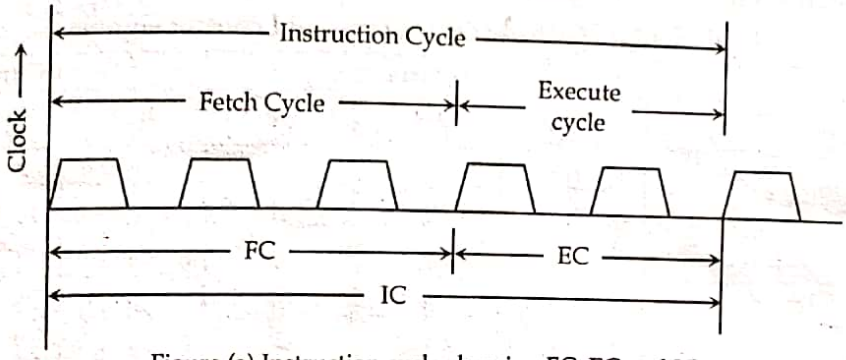


Figure (a) Instruction cycle showing FC, EC and IC

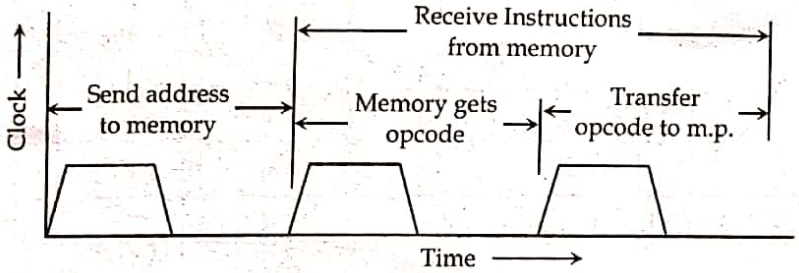


Figure (b) A Typical Fetch cycle

Machine cycle

It is defined as the time required to complete one operation of accessing memory i/p, o/p or acknowledging and external request. This cycle may consists of 3 to 6 T states. T-states: It is defined as one sub division of the operation performed in one clock period. These sub division are internal states synchronized with system clock and each T states precisely equal to one clock period.

Machine cycles of 8085

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

- ✓ Opcode fetch cycle (4T)
- ✓ Memory read cycle (3 T)
- ✓ Memory write cycle (3 T)
- ✓ I/O read cycle (3 T)
- ✓ I/O write cycle (3 T)
- ✓ Interrupt

Time period, T = 1/f; where f = Internal clock frequency

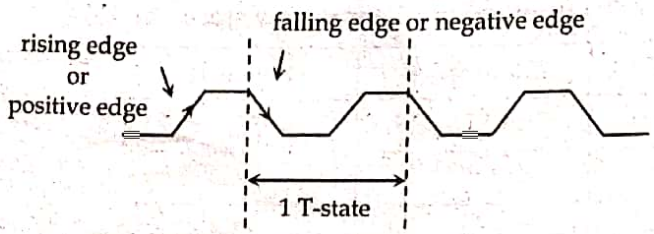


Figure: Clock Signal.

Machine Cycle Status and Control Signals

Table 5.1 (a) Machine cycle status and control signals

	Status			Controls		
Machine cycle	**IO/M**	**S1**	**S0**	**RD**	**WR**	**INTA**
Opcode Fetch (OF)	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/O Read (I/OR)	1	1	0	0	1	1
I/O Write (I/OW)	1	1	1	1	0	1
Acknowledge of INTR (INTA)	1	0	1	1	1	0
BUS Idle (BI) : DAP	0	1	0	1	1	1
ACK of RST, TRAP	1	1	1	1	1	1
HALT	Z	1	0	Z	Z	1
HOLD	Z	0	X	Z	Z	1

X ⇒ Unspecified, and Z ⇒ High impedance state

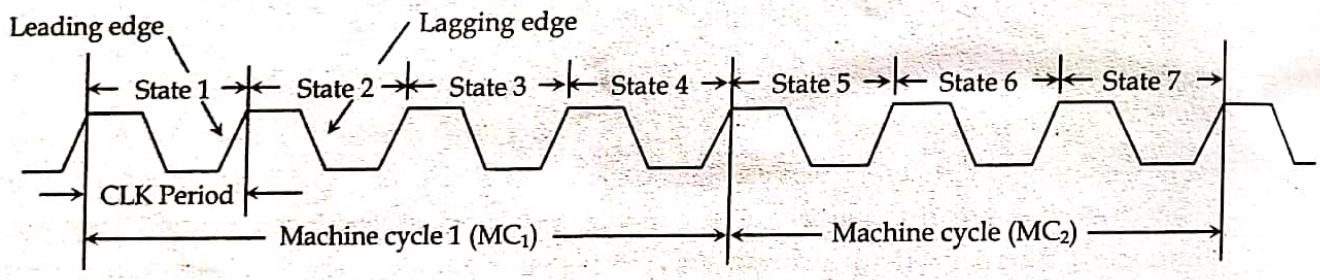
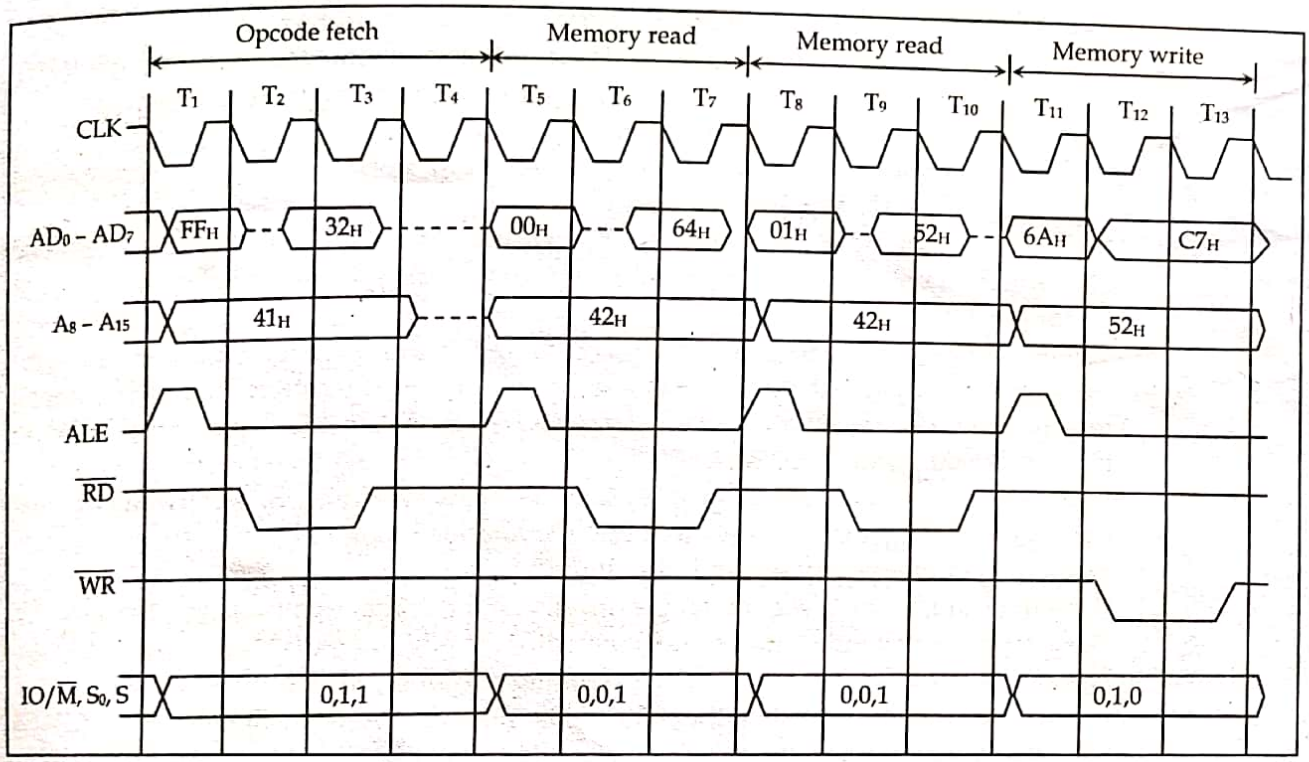


Figure 5.1 (a) Machine cycle showing clock periods.

STA 526AH

- STA means Store Accumulator -The contents of the accumulator is stored in the specified address(526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see fig). - *OF machine cycle.*
- Then the lower order memory address is read(6A). - *Memory Read Machine Cycle*
- Read the higher order memory address (52).- *Memory Read Machine Cycle*
- The combination of both the addresses are considered and the content from accumulator is written in 526A. - *Memory Write Machine Cycle*
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

Address	Memonics	Op code
41FF	STA 526 AH	32_H
4200		$6A_H$
4201		52_H



3. Write an assembly language program to find the smallest number in an array using 8 bit microprocessor. (Assume appropriate array data and address where minimum array size of 15 should be considered.)

Ans:

```
LXI H 8D01      ;  LOAD COUNTER TO MEMORY ADDRESS 8D01
MOV B M         ;  ASSIGNS B AS COUNTER
INX H           ;  POINTS WHERE DATA ARE STORED
MOV A M         ;  CONTENT TRANSFERRED TO ACCUMULATOR
DCR B           ;  COUNTER VALUE DECREASED
LOOP: INX H     ;  POINTS TO ANOTHER MEMORY ADDRESS
CMP M           ;  ADDRESSED MEMORY AND ACCUMULATOR ARE COMPARED
JNC/JC AHEAD    ;  JUMPS IF ACCUMULATOR IS GREATER
MOV A M         ;  ACCUMULATOR CARRIES GREATER VALUE
AHEAD: DCR·B    ;  DECREASE OF COUNTER
JNZ LOOP        ;  IF COUNTER IS NOT ZERO REPEAT THE PROCESS
STA 8D00 H      ;  STORES IN MEMROY ADDRESS 8D00 HLT
```

Group B (Short Answer Question Section)

Attempt any EIGHT questions. **(8x5=40)**

4. Differentiate between vectored and non-vectored interrupt. Where and how 8259 PIC can be used to handle interrupts.

Ans: **Vectored Interrupt**

In vectored interrupts, the processor automatically branches to the specific address in response to an interrupt.

Non-Vectored Interrupt

But in non-vectored interrupts the interrupted device should give the address of the interrupt service routine (ISR).

In vectored interrupts, the manufacturer fixes the address of the ISR to which the program control is to be transferred. The vector addresses of hardware interrupts are given in table above in previous page.

- The TRAP, RST 7.5, RST 6.5 and RST 5.5 are vectored interrupts.
- The INTR is a non-vectored interrupt. Hence when a device interrupts through INTR, it has to supply the address of ISR after receiving interrupt acknowledge signal.

The 8259 Programmable Interrupt Controller

The 8259A programmable interrupt controller designed to work with Intel microprocessors 8085, 8086 and 8088. The 8259A interrupt controller can

1. Manage eight interrupts according to the instructions written into its control registers. This is equivalent to proving eight interrupt pins on the processor in place of one INTR (8085) pin.
2. Vector can interrupt request anywhere in the memory map. However, all eight interrupts are spaced at the interval of either four or eight locations. This eliminates all the major drawback of the 8085 interrupts in which all interrupts are vectored to memory locations on page 00H.
3. Resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode.
4. Mask each interrupt request individually.
5. Read the status of pending interrupts, in-service interrupts, and masked interrupts.
6. Be set up to accept either the level-triggered or the edge-triggered interrupt request
7. Be expanded to 64 priority levels by cascading additional 8259As.
8. Be set up to work with either the 8085 microprocessor mode or the 886/8088 microprocessor mode.

The 8259A is upward-compatible with its predecessor, the 8259. The main difference between the two is that the 8259A can be used with Intel's 8086/88 16-bit microprocessor. It also includes additional features such as the level-triggered mode, buffered mode, and automatic-end-of interrupt mode. To simplify the explanation of the 8259A, illustrative examples will not include the cascade mode or the 8086/88 mode and will be limited to modes continuously used with the 8085.

5. Explain addressing modes of 8085 microprocessor with examples.

Ans: Addressing modes:

Instructions are command to perform a certain task in microprocessor. The instruction consists of op-code and data called operand. The operand may be the source only, destination only or both of them. In these instructions, the source can be a register, a memory or an input port. Similarly, destination can be a register, a memory location, or an output port. The various format (way) of specifying the operands are called addressing mode. So addressing mode specifies where the operands are located rather than their nature. The 8085 has 5 addressing mode:

1) Direct addressing mode:

The instruction using this mode specifies the effective address as part of instruction. The instruction size either 2-bytes or 3-bytes with first byte op-code followed by 1 or 2 bytes of address of data.

E. g. LDA 9500H A ← [9500]

IN 80H A ← [80]

This type of addressing is called absolute addressing.

2) Register Direct addressing mode:

This mode specifies the register or register pair that contains the data.

E g. MOV A, B

Here register B contains data rather than address of the data.

Other examples are: ADD, XCHG etc.

3) **Register Indirect addressing mode:**

In this mode the address part of the instruction specifies the memory whose contents are the address of the operand. So in this type of addressing mode, it is the address of the address rather than address itself. (One operand is register)

e. g. MOV R, M

MOV M, R

STAX, LDAX etc.

STAX B B= 95 C =00 [9500] ← A

4) **Immediate addressing mode:**

In this mode, the operand position is the immediate data. For 8-bit data, instruction size is 2 bytes and for 16 bit data, instruction size is 3 bytes.

E.g. MVI A, 32H

LXI B, 4567H

5) **Implied or Inherent addressing mode:**

The instructions of this mode do not have operands.

E.g. NOP: No operation

HLT: Halt

EI: Enable interrupt

DI: Disable interrupt

6. **Write an ALP to read a string and display the string in uppercase.**

```
;Program To Change The String Into Toggle Case
;------------------------------------------------------------
.MODEL SMALL
.STACK
.DATA
    STRING DB 'Welcome'
    CASE DB 7 DUP(' ')
.CODE
;------------------------------------------------------------
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    LEA SI,STRING ;Load Effective Address of STRING into SI
    LEA DI,CASE   ;Load Effective Address of CASE
    MOV CX,7
    TOP:                                          ;Load Counter with 7
        CMP CX,0000H ;Compare CX with 0
        JE EXIT
        MOV AH,[SI]  ;Load Content of SI into AH   ;If CX=0 then Goto Exit
CMP AH,60H   ;Compare AH with 60H i.6 96
        JA ISSMALL
        CMP AH,5AH   ;Compare AH with 5AH i.e 90
        JB ISCAP
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;TO UPPERCASE
```

```
ISSMALL:
        AND AH,11011111B                ;Mask With 11011111B
        MOV [DI],AH
        INC SI
        INC DI
DEC CX
        JMP TOP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TO LOWERCASE
ISCAP:
        OR AH,00100000B ;Mask With 00100000B
MOV [DI],AH
        INC SI
        INC DI
        DEC CX
        JMP TOP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
EXIT:
MOV AL,'$'      ;Add String Terminator.
  MOV [DI],AL  ;Pass the Content of AL to DI.
  MOV DX,OFFSET CASE
  MOV AH,09H
  INT 21H
  MOV AH,4CH
  INT 21H
MAIN ENDP
END MAIN
;----------------------------------------------------------------
```

7. **What is system bus? Explain different types of system bus in detail.**

Ans: System Bus:

It is a communication path between the microprocessor and peripherals; it is nothing but a group of wires to carry bits.

Data Bus: A collection of wires through which data is transmitted from one part of a computer to another is called Data Bus. Data Bus can be thought of as a highway on which data travels within a computer. This bus connects all the computer components to the CPU and main memory. The size (width) of bus determines how much data can be transmitted at one time.

E.g.:

- A 16-bit bus can transmit 16 bits of data at a time.
- 32-bit bus can transmit 32 bits at a time.

Address Bus:

A collection of wires used to identify particular location in main memory is called Address Bus. In other words, the information used to describe the memory locations travels along the address bus. The size of address bus determines how many unique memory locations can be addressed.

E.g.:

- A system with 4-bit address bus can address 24 = 16 Bytes of memory.
- A system with 16-bit address bus can address 216 = 64 KB of memory.
- A system with 20-bit address bus can address 220 = 1 MB of memory.

Control Bus

The connections that carry control information between the CPU and other devices within the computer is called Control Bus. The control bus carries signals that report the status of various devices.

E.g.: This bus is used to indicate whether the CPU is reading from memory or writing to memory.

8. **How DTE and DCE are wired using RS 232 cable? Explain the process of double handshaking I/o.**

Ans: **RS 232C interface with DTE and DCE:**

The figure below shows a interfacing with minimum lines.

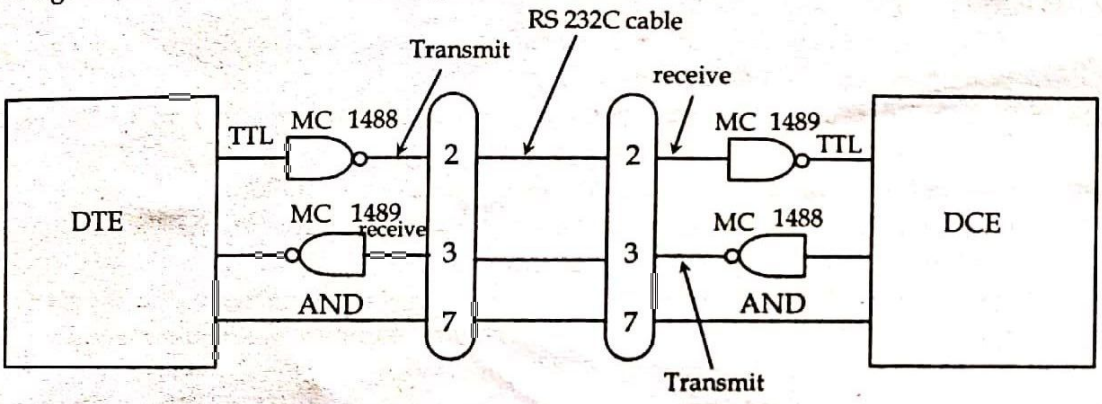


Figure: RS 232C interface

The signaling in RS-232C is not compatible with the TTL logic level. For TTL 0 v to 0.2V is considered a logic 0 and 3.4 v to 5v as logic 1. But RS-232C works in a negative logic -3 to -15v considered as logic 1 and +3 to +15v as logic 0. Because of this incompatibility of the data lines with the TTL logic, voltage translators called line drivers and line receivers are required to interface TTL logic with RS-232C signals. The line driver MC 1488 converts logic 1 into approx. -9V and logic 0 into +9v. Before it is received by the DCE it is again converted by the line receiver MC 1489 into TTL-Compatible logic.

The minimum interface required both a computer and a peripheral device requires three lines; pin 2, 3 and 7. These lines are defined in relation to the DTE; the terminal transmits on pin 2 and receives as pin 3. On the other hand the DCE transmits on pin 3 and receives on pin 1. Pin 7 is ground pin.

DOUBLE HANDSHAKE I/O DATA TRANSFER

For data transfers where coordination is required between sending system and the receiving system, a double handshake is used.

- The sending device asserts its STB low to ask ,**Are you ready?"**
- The receiving system raises its ACK line high to say ,**I'm ready".**
- The peripheral device then sends the data byte and raises its STB signal high to say **"Here is valid data for you".**
- When the receiving system finishes to read the data, receiving system drop its ACK line low to say **"I have data than you and I await your request to send the next byte of data".**

9. **What is instruction set? Explain different kinds of instruction set used in 8085 microprocessor.**

Ans: **THE 8085 INSTRUCTION SETS**

An instruction is a binary pattern designed inside a microprocessor to perform a specific function (task). The entire group of instructions called the instruction set. The 8085 instruction set can be classified into 5- different groups:

- Data Transfer Instructions

- Arithmetic Instructions
- Branching Instructions
- Logical Instructions
- Control Instructions

DATA TRANSFER INSTRUCTIONS

It is the longest group of instructions in 8085. This group of instruction copy data from a source location to destination location without modifying the contents of the source. The transfer of data may be between the registers or between register and memory or between an I/O device and accumulator. None of these instructions changes the flag.

ARITHMETIC INSTRUCTIONS

The 8085 microprocessor performs various arithmetic operations such as addition, subtraction, increment and decrement.

The arithmetic operation add and subtract are performed in relation to the contents of accumulator. The features of these instructions are

- They assume implicitly that the accumulator is one of the operands.
- They modify all the flags according to the data conditions of the result. They place the result in the accumulator.
- They do not affect the contents of operand register or memory. But the INR and DCR operations can be performed in any register or memory. These instructions

✓ Affect the contents of specified register or memory.

✓ Affect the flag except carry flag.

BRANCHINGINSTRUCTIONS

The microprocessor is a sequential machine; it executes machine codes from one memory location to the next. The branching instructions instruct the microprocessor to go to a different memory location and the microprocessor continues executing machine codes from that new location.

LOGICAL INSTRUCTIONS

A microprocessor is basically a programmable logic chip. It can perform all the logic functions of the hardwired logic through its instruction set. The 8085 instruction set includes such logic functions as AND, OR, XOR and NOT (Complement).

10. What is mean by memory interfacing? Explain the address decoding process in 8085 microprocessor with 3 X 8 decoder.

Ans: Interfacing a microprocessor is to connect it with various peripherals to perform various operations to obtain a desired output.

Memory Interfacing and I/O Interfacing are the two main types of interfacing.

Memory Interfacing is used when the microprocessor needs to access memory frequently for reading and writing data stored in the memory. It is used when reading/writing to a specific register of a memory chip.

I/O Interfacing is achieved by connecting keyboard (input) and display monitors (output) with the microprocessor.

The method use by the system to select the correct location on the correct chip is called addressing decoding. In another way, it is the process of generating a chip select (CS) or enable signals from the address bus for each device in the system. The address bus lines are split into two sections:

- The N, MSB are used to generate a chip select (CS) signal for different device.
- The M, LSB are passed in the device as address to the different memory cells or internal registers.

ADDRESS BUS

N- bits to decoder(MSB)	M- bits to memory (LSB)

Address decoding can be done using 3 to 8 decoder.

- Assume a microprocessor with 10 address lines that give total of mapping 1 KB memory.
- Let us consider that we want to implement all its memory space and we use 128*8 memory chips.
- So we have 8 memory chips with 128 bytes storage, so we need 3 address lines connected with the decoder to generate 8 different chip select signals and select one of the 8 chips.
- And 7 address lines to select a particular memory word of the selected chip.

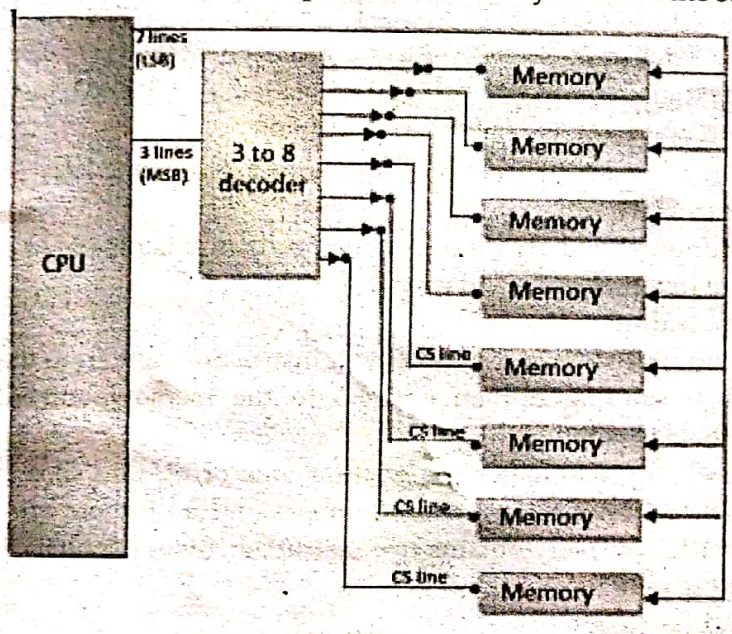


Fig: 3 to 8 address decoding

Figure: 3 to 8 address decoding.

11. Explain how pipelining is achieved in 8086 microprocessor.

Ans: Pipelining:

Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in**stages**. Each stage completes a part of an instruction in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end.

Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput. The **throughput** of the instruction pipeline is determined by how often an instruction exits the pipeline.

Because the pipe stages are hooked together, all the stages must be ready to proceed at the same time. We call the time required to move an instruction one step further in the pipeline ***a machine cycle***. The ***length*** of the machine cycle is determined by the time required for the slowest pipe stage.

The pipeline designer's *goal is to balance the length of each pipeline stage*. If the stages are perfectly balanced, then the time per instruction on the pipelined machine is equal to

Time per instruction on nonpipelined machine

Number of pipe stages

Under these conditions, the speedup from pipelining equals the number of pipe stages. Usually, however, the stages will not be perfectly balanced; besides, the pipelining itself involves some overhead.

Instruction Pipeline:

Any architecture can be pipelined by making each clock cycle into a pipe stage.

Clock#	1	2	3	4	5	6	7
Instruction i	Fetch	Decode	Execute				
Instr. i+1		Fetch	Decode	Execute			
Instr. i+2			Fetch	Decode	Execute		
Instr. i+3				Fetch	Decode	Execute	
Instr. i+4					Fetch	Decode	Execute

Here instruction i is fetched in clock #1. After it has been fetched it is decoded in clock #2 and at the same time next instruction i.e. instr. i+1 is fetched. At Clock#3, instruction i is executed, instr. i+1 is decoded and instr. i+2 is fetched and so on.

Hence at the end of clock #3, instruction i is executed. Sometime later at the end of clock #4 instruction i+1 is executed.

If we assume that unpipelined architecture took 3ns then this pipelined architecture will take 1ns to finish a stage (fetch, decode and execute).

Advantages of pipelining:

- The execution unit always reads the next instruction byte from the queue in BIU. This is faster than sending out an address to the memory and waiting for the next instruction byte to come.
- In short pipelining eliminates the waiting time of EU and speeds up the processing. The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in queue. 8086 BIU normally obtains two instruction bytes per fetch.

12. Write short notes on: (any two)

a. Von Neumann architecture

Ans: The simplest way to organize a computer is to have one processor, register and instruction code format with two parts op-code and address/operand. The memory address tells the control where to find an operand in memory. This operand is read from memory and used as data to be operated on together with the data stored in the processor register. Instructions are stored in one section of same memory. It is called stored program concept.

The task of entering and altering the programs for ENIAC was tedious. It could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data. So the computer could get its instructions by reading from the memory and program could be set or altered by setting the values of a portion of memory. This approach is known as 'stored- program concept' was first adopted by John Von Neumann and such architecture is named as von-Neumann architecture and shown in figure below.

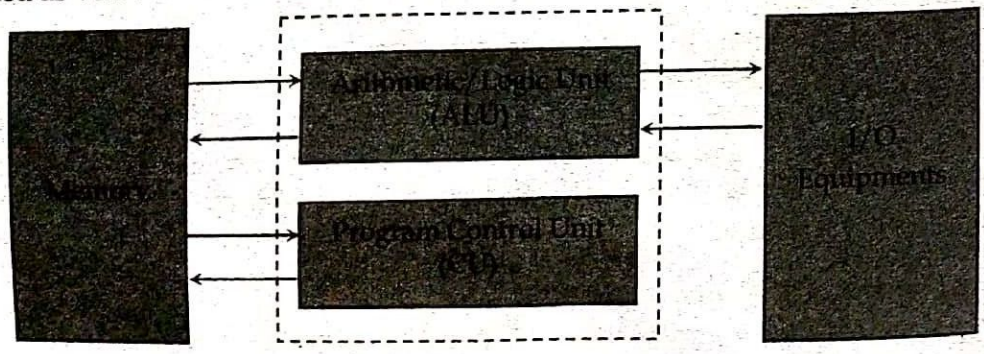


Fig: Von-Neumann Architecture.

The main memory is used to stare both data and instructions. The arithmetic and logic unit is capable of performing arithmetic and logical operation on binary data. The program control unit interprets the instruction in memory and causes them to be executed. The I/O unit gets operated from the control unit.

The Von-Neumann architecture is the fundamental basis for the architecture of modern digital computers. It consisted of 1000 storage locations which can hold words of 40 binary digits and both instructions as well as data are stored in it. The storage location of control unit and ALU are called registers and the various models of registers are:

MAR – memory address register – contains the address in memory of the word to be written into or read from MBR.

MBR – memory buffer register – consists of a word to be stored in or received from memory.

IR – instruction register – contains the 8-bit op-code instruction to be executed.

IBR – instruction buffer register – used to temporarily hold the instruction from a word in memory.

PC - program counter - contains the address of the next instruction to be fetched from memory.

AC & MQ (Accumulator and Multiplier Quotient) - holds the operands and results of ALU after processing.

b. Macro assembler

Ans: A macro assembler is able to generate a program segment, which is defined by a macro, when the name of the macro appears as an opcode in a program. The macro assembler is still capable of regular assembler functioning, generating a machine instruction for each line of assembly language code; but like a compiler, it can generate many machine instructions from one line of source code. Its instruction set can be expanded to include new mnemonics, which generate these program segments of machine code. The following discussion of how a macro works will show how this can be done.

A frequently used program segment can be written just once, in the macro definition at the beginning of a program. For example, the macro

```
LFEED MACRO
MOV AH, 06H
MOV DL, 0AH
INT 21H
MOV DL, 0DH
INT 21H
ENDM
```

Is used to put down the cursor in next line.

❑❑❑