Prepared by :Loknath Regmi

January 1

2014

Microprocessor Manual

BSC.CSIT

Chapter: 1
Introduction to Microprocessor

Microprocessor is a multipurpose, programmable, clock-driven, register-base, electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions, and provides results as output.

A typical programmable machine can be represented with four components: microprocessor, memory, input, and output. The physical components of this system are called hardware. A set of instructions written for the microprocessor to perform a task is called a program, and a group of program is called software.

Microprocessor is programmable means it can be instructed to perform given task within its capability. A programmer can select appropriate instructions and ask the microprocessor to perform tasks on a given set of data. These instructions are entered or stored in storage, called memory, which can be read by the microprocessor.

A microprocessor incorporates the functions of a computer's central processing unit (CPU) on a single integrated chip (IC).

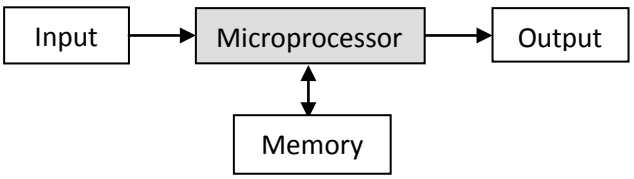


Figure: typical programmable machine

1.1 History of Microprocessor

The development of microprocessor is merely dependent on the development of integrated circuit technology (LSI, VLSI, ELSI). Intel invented the term "**microprocessor**" and in 1971 released the first 4 bit microprocessor as: **Intel 4004**. The 4004 was the first chip to contain all of components of a CPU on single chip with LSI technology. The microprocessor revolution began with this tiny chip. The features of Intel 4004 are:

- ✓ It handles four bits at a time
- ✓ It has specific instructions for reading keyboard and performing decimal and hexadecimal arithmetic.
- ✓ It uses fixed programs stored in read only memory and data stored in small read/write memory.

The next major step in the microprocessor was introduction of the **Intel 8008** in 1972. It was the first 8 bit microprocessor designed for terminals. It handles 8 bits data at a time and can access larger amount of R/W memory than Intel 4004.It can handle peripherals like printers and CRTs.

The world's first general purpose microprocessor is **Intel 8080.** This was an 8 bit machine, with 8 bit data and 16 bit address path to memory. Some other examples of 8 bit microprocessors are **Motorola 6500,**

Fair-child F-8, Signetics 2650, Toshiba-12, Texas Instrument, etc. After a couple year later **Intel 8085** microprocessor were developed as improvements over the 8080.
Intel 8086is one of the popular, powerful 16 bit microprocessor developed by Intel, with 1MB memory capacity (*20 bit address bus*).
An extension of 8086 with addressing capability of 16 MB of memory is **Intel 80286**.
While **Intel 80386** is Intel's first 32 bit microprocessor having 4 GB memory capacity.
The **Intel 80486** introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining.
With the **Pentium**, Intel introduced the use the superscalar techniques which allow multiple instructions to be executed in parallel. These days there are advancements made in the Pentium Technologies and multi core processors are available.

Microcontroller

A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

1.2 Basic Block Diagram of a Computer

The traditional block diagram of a computer Shows that the computer has four components: Memory, input, output & the central Processing unit (CPU), which consists of the Arithmetic/logic unit (ALU) and the Control unit (CU). The CPU reads instructions From the memory and performs the tasks Specified. It communicates with input/output devices either to accept or to send data.

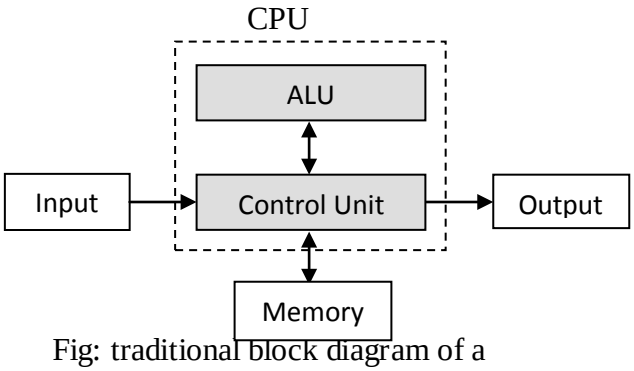


Fig: traditional block diagram of a computer

The CPU is the primary part and was designed with discrete components on various boards.

With the arrival of integrated circuit technique, it became possible to build the **CPU on a single chip**; this came to be known as a microprocessor.
The traditional block diagram of computer can be replaced by the block diagram of a computer with a microprocessor as its CPU as shown in figure.

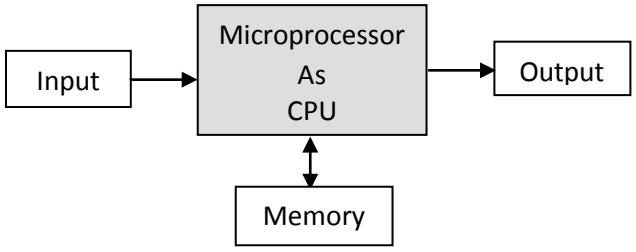


Fig: block diagram of a computer with microprocessor as CPU

The figure alongside resembles a block diagram of a digital computer. The major parts are central processing unit, memory, and input output ports. Each of this part is connected with each other through address bus, data bus, and control bus.

Memory: This consists of RAM and ROM. It may also have magnetic floppy disk, magnetic hard disk or optical disk. It's function are:

1.Store the binary codes for the sequences of instruction and then write a program from that sequence of instruction for the computer.

2.Store the binary coded data with which the computer is going to work.

I/O port: The I/P section allows the computer to take in data from the outside world or send data to the outside world. Eg:- keyboard, video display terminals, printers, modems, etc. The physical devices used to interface the computer buses to external systems are called ports. Two ports are available i/p port example keyboard, mouse. O/p port example monitor, printer.

Central processing unit "CPU": The CPU controls the operation of computer. CPU fetches binary coded instruction from memory. Decode the instruction into a series of actions and carries out these actions in a sequence of steps. It also contains the instruction pointer register which hold the address of the next instruction or data item to be fetched from memory.

1.3 Organization ofMicroprocessor Based System with Bus Architecture

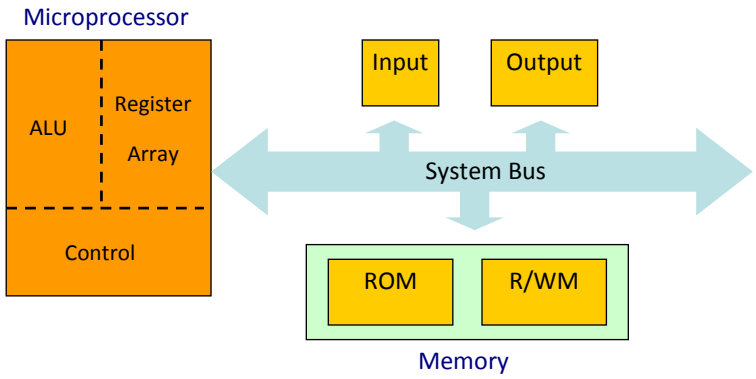


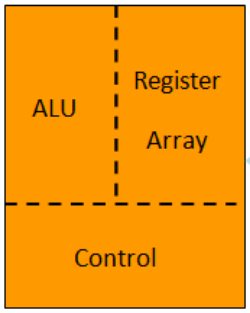
Fig: Basic structure of microprocessor based system

The basic structure of microprocessor based system includes microprocessor, I/O and memory (ROM & R/WM). These components are organized around a common communication path called bus. A microprocessor is composed of following different units.

- **Microprocessor:**

It is manufactured by IC integration technique and is capable of performing various computing functions and making decisions to change the sequence of program execution. Microprocessor can be divided into three segments:

1. Arithmetic/Logic Unit (ALU)
2. Register array
3. Control Unit (CU)



***ALU*:** This area of microprocessor performs various functions on data. The ALU unit performs arithmetic operation like addition subtraction and logical operation like And, OR, XOR.

***Register array*:** This area of microprocessor consists of various registers, which are primarily used to store the data temporarily during the execution of a program and are accessible to the user through instructions.

***Control Unit*:** This area provides the necessary timing and control signal to all the operations in the microcomputer. It contains the flow of data between the microprocessor, memory and its peripherals.

- **Memory**:

Memory stores binary information as instructions and data, and provides that information to the microprocessor whenever necessary. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operation in its ALU section. Results are either transferred to the output section for display or stored in memory for later use. It has two sections Read only memory (ROM) and Read/Write memory (R/WM), popularly known as Random Access memory (RAM).
The ROM is used to store programs that do not need alteration. Programs stored in ROM can only be read; they can not be altered.
The R/WM is used to store user programs and data. The information stored in this memory can be easily read and altered.

- **I/O (Input/Output**):

It communicates with outside world, also known as peripherals. The input devices such as a keyboard, switches, and an analog to digital converter (ADC) transfer information from the outside world to the microprocessor. Where, the output devices transfer data from the microprocessor to the outside world. They include devices such as light emitting diodes (LED's), a cathode-ray tube (CRT) or video screen, a printer, a magnetic tape etc.

- **System bus:**

It is a communication path between the microprocessor and peripherals. The microprocessor communicates with only one peripheral at a time. The timing is provided by the control unit of the microprocessor.

1.4 Bus Organization

Address bus: The address bus consists of 16, 20, 24 or 32 parallel signal lines that is used to specify a physical address. On these lines the CPU sends out the address of the memory location that is to be written to or from.The width of the address bus determines the amount of memory a system can address. For example, a system with a 32-bit address bus can address 2^{32} (4,294,967,296= 4 GB) memory locations.
Data bus: The data bus consist of 8, 16 or 32 parallel signal lines and are bidirectional that carries the actual data being processed. CPU can read data in from memory and send data out to memory on these lines.
Control bus: The control bus consists of 4-10 parallel signal lines. The CPU sends out signal on the control bus to enable the o/p of the address memory device. The control bus carries commands from the CPU and returns status signals from the devices. Control bus signal are memory read, write, i/p read, o/p write.

1.5 Stored Program concept and Von Neumann Machine:

The task of entering and altering the programs for the ENIAC (electronic numerical integrator and computer) was extremely tedious. The programming concept could be facilitated if the program could represent in a form suitable for storing in memory alongside the data. Than a computer could get its

instruction by reading them form the memory and a program could be set or altered by setting the values of a portion of memory. This approach is known "stored program concept", was first adopted by John Von Neumann and hence the architecture of computer he proposed is name as Von-Neumann's architecture. (*Note: 8085 µp uses this architecture*)

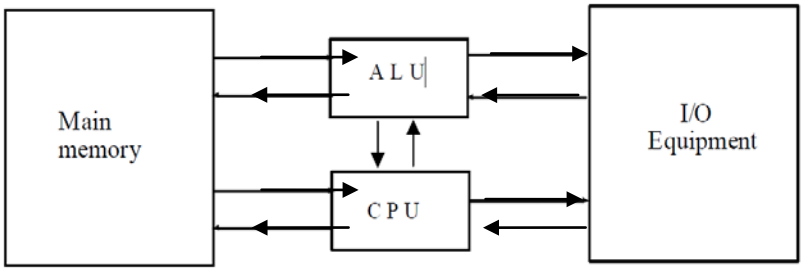


Fig: Von-Neumann Architecture

Main memory is used to store both data and instructions. The ALU is capable for performing arithmetic and logical operation binary data. The program control unit (PCU) interprets the instruction in memory and causes them to be executed. The I/O unit gets operated from the control unit. The input/output unit helps inputting data and getting results.

The Von-Neumann's Architecture is the fundamental basis for the architecture of today's digital computers. The memory of Von-Neumann machine consists of thousand storage location called words of 40 binary digits (bits). Both data and instruction are stored in it. The storage locations of control unit and ALU are called registers. The various registers of this model are MBR, MAR, IR, IBR, PC, AC.

Memory Buffer Register (MBR): It consists of a word to be stored in memory or is used to receive a memory or is used to receive a word from memory.

Memory address Register (MAR): It contains the address in memory of the world to be written from or read into the MBR.

Instruction register (IR): Contain the 8 bit op-code (operation code) instruction being executed.

Instruction buffer register (IBR): It is used to temporarily hold the instruction from a word in memory.

Program counter (PC): It contains address of next instruction to be fetched from memory.

Ac (Accumulator) and MQ (multiplier quotient): They are employed to temporarily hold operands and results of ALU operations.

Harvard Architecture:

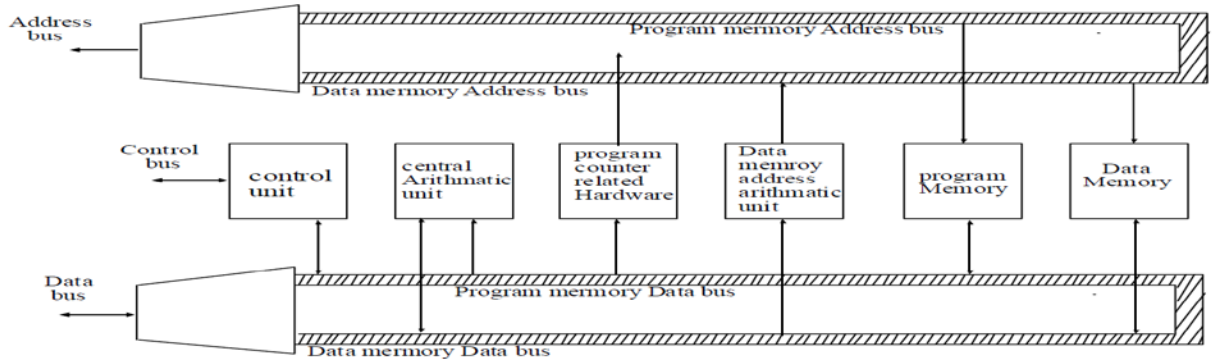


Fig. Block diagram of the Harvard architecture based µP

Harvard Architecture based computer consist so separate memory spaces for the programs (or instruction) and data. Each memory space has its own address and data bus. Thus both instruction and data can fetch from memory concurrently. From the figure it is seen that there are two data and two and address buses for the program and data memory spaces respectively. The program memory data bus and data memory data are multiplexed to form single data bus whereas program memory Address and data memory address are multiplexed to form single address bus. Hence there are two blocks of RAM chip: One for program memory and another for data memory space. Data memory address arithmetic unit generates data memory address. The data memory address bus carries the memory address of data whereas program memory address bus carries the memory address of the instruction. Central arithmetic logic unit consists of the ALU, multiplier, Accumulator, etc. The Program Counter is used to address program memory. PC always contains the address of next instruction to be fetched. Control unit control the sequence of operations to be executed. The data and control bus are bidirectional whereas address bus is unidirectional.

1.6 Applications of Microprocessor

The microprocessors applications are classified primarily in two categories: ***Reprogrammable systems*** and ***Embedded Systems.*** In reprogrammable systems, such as microcomputers, the microprocessor is used for computing and data processing, a Personal Computer (PC) is a typical illustration. In an embedded system such as photo copy machines, traffic light controller, the microprocessor is a part of a final product and is not available for reprogramming to the end user. Main applications areas are:

- 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.
- 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 application

Chapter-2
Basic Computer Architecture

SAP- 1 Architecture:

The SAP (Simple-As-Possible) computer has been designed for beginners. The main purpose of SAP is to introduce all the crucial ideas behind computer operation without burying you in unnecessary detail. SAP-1 is the first stage in the evolution toward modern computers. SAP-1 is a big step for beginners.

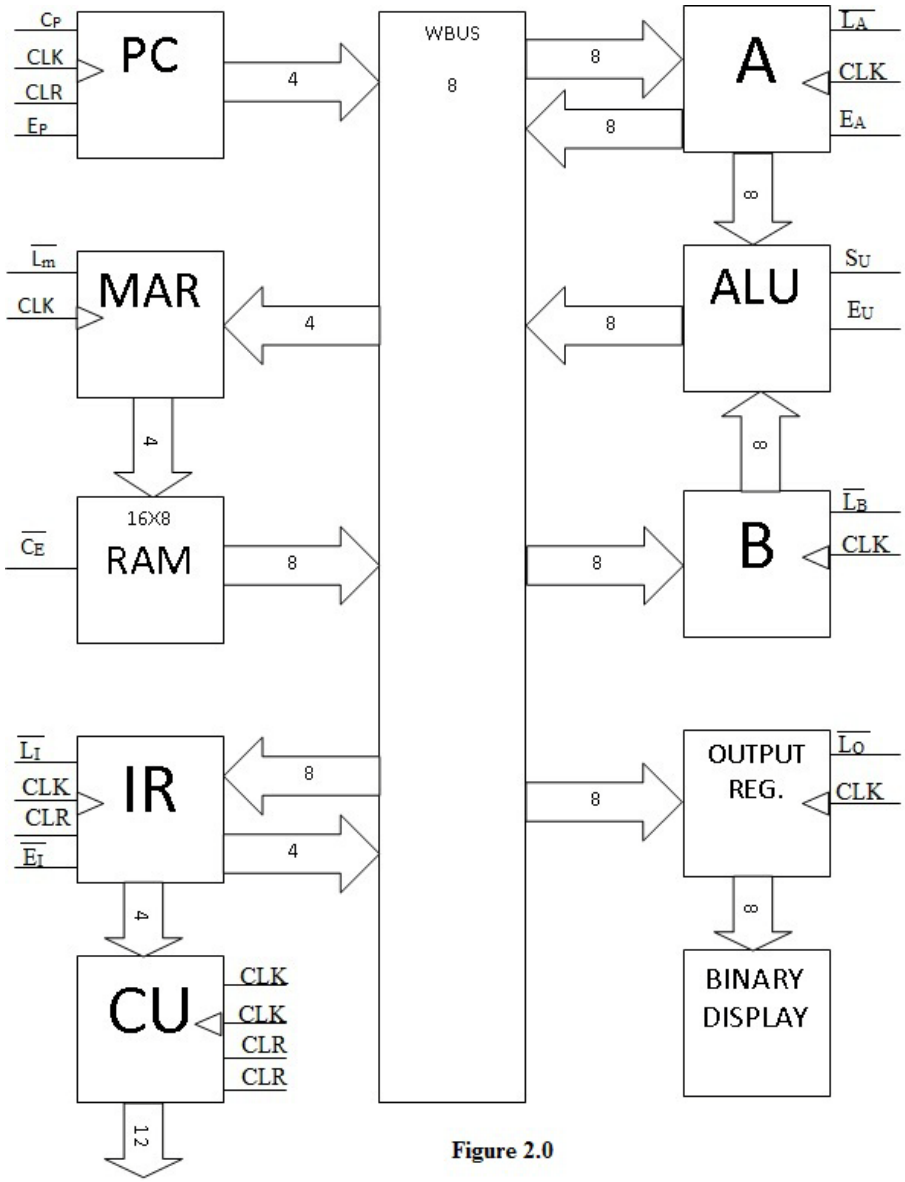


Figure 2.0

1. Program Counter

The program is stored at the beginning of the memory with the first instruction at binary address 0000, the second instruction at 0001, the third at address 0010 and so on. The program counter which is part of

the control unit, counts from 0000 to 1111. Its job is to send to the memory the address of the next instruction to be fetched and executed. It does this as mentioned in the next paragraph.

The program counter is reset to 0000 before each computer run. When the computer run begins, the program counter sends the address 0000 to the memory. The program counter is then incremented to get 0001. After the first instruction is fetched and executed, the program counter sends address 0001 to the memory. Again the program counter is incremented. After the second instruction is fetched and executed, the program counter sends address 0010 to the memory. So this way, the program counter keeps track of the next instruction to be fetched and executed.

The program counter is like someone pointing a finger at a list of instructions saying do this first, do this second, do this third, etc. This is why the program counter is called a pointer; it points to an address in memory where the instruction or data is being stored.

2. Input & MAR

The Input and MAR includes the address and data switch registers. Switch registers are part of input unit, allows us to send 4 address bits and 8 data bits to the RAM.

The memory address register (MAR) is the part of SAP-1 memory. During a computer run, the address in the program counter is latched in to the MAR. A bit later, the MAR applies this 4-bit address to the RAM where a read operation is performed.

3. The RAM

The RAM is a 16 X 8 static TTL RAM. We can program the RAM by means of the address and data switch registers. This allows you to store a program and data in the memory before a computer run.

During a computer run, the RAM receives 4-bit addresses from the MAR and a read operation is performed. In this way, the instruction or data word stored in the RAM is placed on the W bus for use in some other part of the computer.

4. Instruction Register

The instruction register is the part of the control unit. To fetch an instruction from the memory the computer does a memory read operation. This places the contents of the addressed memory location on the W bus. At the same time, the instruction register is set up for loading on the next positive clock edge. The content of the instruction register are split into two nibbles. The upper nibble goes directly to the block "Controller – Sequencer". The lower nibble is read onto the W bus when needed.

5. Controller – Sequencer

Before each computer runs, a CLR signal is sent to the program counter and CLK signal to the instruction register. This resets the program counter to 0000 and wipes out the last instruction in the instruction register.

A clock signal CLK is sent to all buffer registers; this synchronizes the operation of the computer ensuring that things happen when they are supposed to happen.

The 12 bits that come out of the controller sequencer form a word controlling the rest of the computer (like a supervisor telling others what to do). The 12 wires carrying the control word are called the control bus. The control word has the format of:

$$\mathbf{CON} = C_P E_P \bar{L}_M \overline{CE} \quad \bar{L}_I \bar{E}_I \bar{L}_A E_A \quad S_U E_U \bar{L}_B \bar{L}_O$$

This word determines how the registers will wait to the next positive CLK edge. For example, a high E_P and a low L_M means that the program counter are latched into the MAR on the next positive clock edge. As another example, a low CE and a low L_A means that the addressed RAM word will be transferred to the accumulator.

6. Accumulator

The accumulator (A) is a buffer register that stores intermediate answers during a computer run. Accumulator has two outputs, one directly goes to the adder-subtractor and the other goes to the W bus.

7. The Adder – Subtractor

SAP-1 uses a 2's complement adder-subtractor. When S_U is low, the sum out of the adder-subtractor is S = A + B. When S_U is high, the difference appears as A = A + B '.

8. B Register

The B register is another buffer register. It is used in arithmetic operations. A low LB and positive clock edge load the word on the W bus into the B register. The two state output of the B register drives the adder-subtractor, supplying the number to be added or subtracted from the content of the accumulator.

9. Output Register

At the end of the computer run, the accumulator contains the answer to the problem being solved. At this point, we need to transfer the answer to the outside world. This is where the output register is used.

When E_A is high and L_O is low, the next positive clock edge loads the accumulator content to the output register. The output register is often called an output port because the processed data can leave the computer through this register.

10. Binary Display

The binary display is a row of eight light emitting diodes (LED's). Because each LED connects to one flip-flop of the output port, the binary display shows us the content of the output port. Therefore, after we transferred an answer from the accumulator to the output port, we can see the answer in binary form.

SAP-1 Instructions:

When a user writes a program in a high-level language (HLL), the computer will not understand the program unless it is translated into a language that the computer understands. The HLL is translated into an equivalent assembly language which is then translated into the machine language where the instructions are strings of 0's and 1's which can also be easily understand by the computer. SAP-1 has five instructions which are all stored in the memory. They are explained as below.

LDA:

- ✓ LDA stands for "load the accumulator".
- ✓ A complete LDA instruction includes the hexadecimal address of the data to be loaded.

ADD:

- ✓ A complete ADD instruction includes the address of the word to be added.
- ✓ For on stance, ADD 9H means 'add the contents of memory location 9H to the accumulator contents'.
- ✓ The sum replaces the original contents of the accumulator.

SUB:

- ✓ A complete SUB instruction includes the address of the word to be subtracted.
- ✓ SUB CH means 'subtract the contents of the memory location CH from the contents of the accumulator.
- ✓ The difference out of the adder-substractor then replaces the original contents of the accumulator.

OUT:

- ✓ The instruction OUT tells the SAP-1 computer to transfer the accumulator contents to the output port.
- ✓ After OUT has been executed, we can see the answer to the problem being solved.

HLT:

- ✓ HLT stand for halt.
- ✓ This instruction tells the computer to stop processing
- ✓ HLT is complete by itself; we do not need to use RAM word using HLT because it does not involve memory.

Type of Instruction:

On the basis of involvement with memory during execution, there are two types of instructions. They are

- → Memory Reference Instruction
- → Non Memory Reference Instruction

Memory Reference Instruction

Those instructions which use the data stored on the memory are called memory reference instruction. In SAP-1 architecture, LDA, ADD, and SUB are the memory reference instructions.

Non Memory Reference Instruction

Those instructions which use the data stored on the memory are called non memory reference instruction. In SAP-1 architecture, OUT and HLT are the memory reference instructions.

Mnemonics:

Abbreviated instructions like LDA, ADD, SUB, OUT,HLT are called mnemonics.

Operation code(opcode):

The operation code specifies the operation perform by the instructions. It is the equivalent machine language of each mnemonic. Opcode identifies the instruction to be executed and each instruction has its unique opcode that the computer can handle. They take the strings of 1's and 0's. SAP-1 architecture uses the 4-bit operation code.

List of Instructions:

Mnemonics	Opcode	Syntax	Operation
LDA	**0000**	**LDA x**	M[x] –> AC
ADD	**0001**	**ADD x**	M[x] –> B AC<– [AC] + [B]
SUB	**0010**	**SUB x**	M[x] –> B AC <– [AC] - [B]
OUT	**1110**	**OUT**	[AC] –> **0**
HLT	**1111**	**HLT**	Disable

Table 1: Instruction Set of SAP-1 Architecture

AC = Accumulator
[AC] = content of accumulator
[x] = content of memory location x

Instruction cycle:

The control unit is the key to a computer's automatic operation. The CU generates the control words that fetch and execute each instruction. While each instruction is fetched and executed, the computer passes through different timing states (T states), periods during which register contents change called machine cycle. SAP-1 instruction cycle is broken into two cycles as:

- ✓ Fetch cycle
- ✓ Execute cycle

Timing and control signal for instruction is given below.

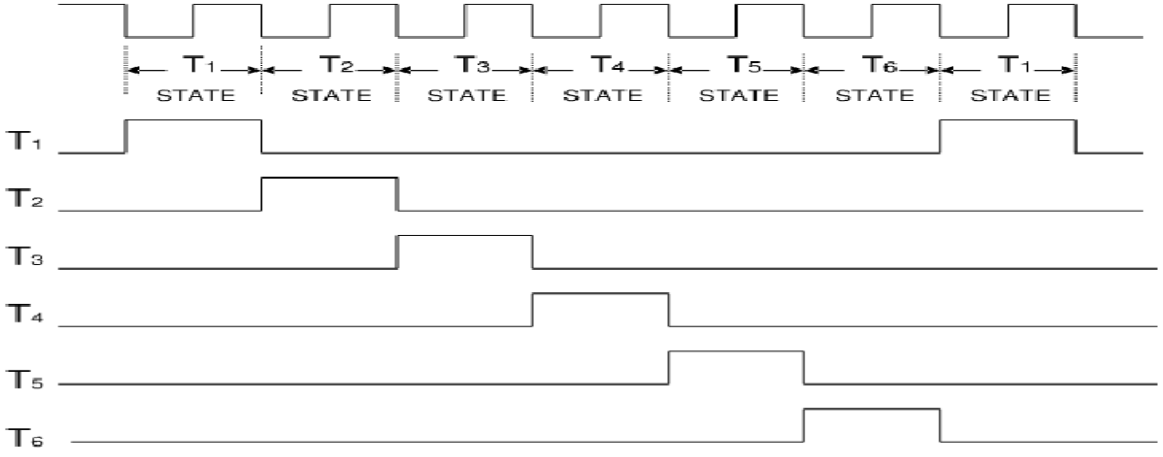


Fig: Timing and control signal

- The initial state T_1 starts with negative clock edge and end with negative clock edge. During this T-state, the T_1 bit is high.
- During next T-state, the T_2 bit is high then T_3 and so on.
- Instruction are fetched and executed on these six T- states.

Fetch Cycle:

The first cycle of instruction is the fetch cycle which makes use of the PC, MAR, IR and RAM. Fetch cycles has three timing states as, T1, T2, and T3. In fetching an instruction, it needs to have a memory to set the instruction and place to hold this instruction which is the Instruction Register (IR). Program counter which is also a part of control unit which send addresses to the memory, the address of the next instruction to be fetched are executed. These three states are

→ Address state
→ Increment state
→ Memory state

Address state:

The T1 state of the control timing signal is called the address state because the address in the program counters (PC) is transferred into the memory address register. During this state, E_P and L_M are active, this means the program counter setup the MAR via W-bus , a positive clock occur midway through the address state; this loads the MAR with content of PC.

i.e. MAR ← PC

Control word format on this stage is

$$\begin{aligned}\mathbf{CON} &= C_P E_P \bar{L}_M \overline{CE} \quad \bar{L}_I \bar{E}_I \bar{L}_A E_A \quad S_U E_U \bar{L}_B \bar{L}_O \\ &= 0\ 1\ 0\ 1 \quad 1\ 1\ 1\ 0 \quad 0\ 0\ 1\ 1\end{aligned}$$

Increment state: The T2 state is called the increment state because the value of program counter is incremented by one in this state to get the next memory location. Cp is only active during the increment state. This set up the program counter to count the positive clock edges. Half way through the increment state, a positive clock edge hits the program counter and advances the count by 1.

i.e. PC ← (PC+1)

Control word for this stage is

$$\begin{aligned}\mathbf{CON} &= C_P E_P \bar{L}_M \overline{CE} \quad \bar{L}_I \bar{E}_I \bar{L}_A E_A \quad S_U E_U \bar{L}_B \bar{L}_O \\ &= 1\ 0\ 1\ 1 \quad 1\ 1\ 1\ 0 \quad 0\ 0\ 1\ 1\end{aligned}$$

Memory state:

The state T3 is called the memory state because instruction available in RAM addressed is transferred to the instruction register (IR). During the memory state C_E and L_I are active. Therefore the addressed RAM words setup the instruction resister via W-bus. Midway through the memory state, a positive clock edge loads the instruction by the addressed RAM words.

i.e. IR ← M[MAR]

Control word for this stage is

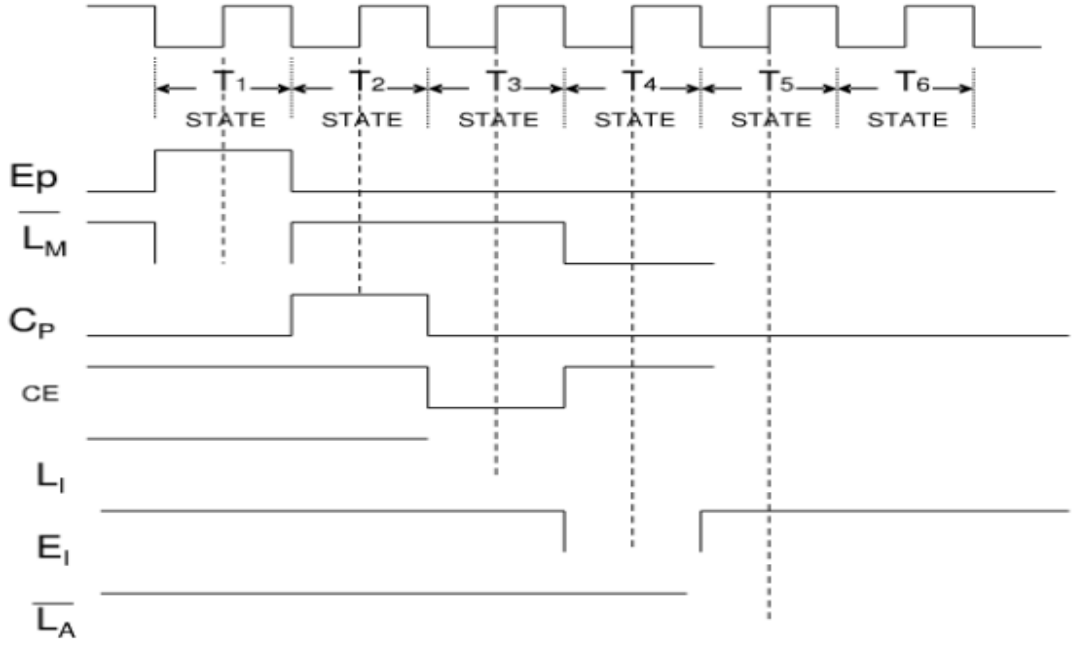
$$\begin{aligned}\mathbf{CON} &= C_P E_P \bar{L}_M \overline{CE} \quad \bar{L}_I \bar{E}_I \bar{L}_A E_A \quad S_U E_U \bar{L}_B \bar{L}_O \\ &= 0\ 0\ 1\ 0 \quad 0\ 1\ 1\ 0 \quad 0\ 0\ 1\ 1\end{aligned}$$

Hence, these three states: address, increment and memory state are called the fetch cycle of the SAP- 1 Architecture. Each component of the SAP-1 has its own control lines which need to be activated and they can be either active high or active low. Activating the control lines enable each component to communicate with a particular device and the control inputs are handled by the control unit.

It has two control signal Cp, which is high if it has to be incremented and Ep instruction store in this memory. Memory address register include the address and switch register. The switch register which are part of the input unit register allow sending an address bits to the RAM. The PC asserts an addresses to the MAR then IR take hold of the instruction to the bus. Instruction register is a part of a control unit, the contents of the instruction register are split into two nibbles; upper nibbles which has two state output that goes directly to the controller, sequencer and lower nibble which has three state output that is directed onto WBUS .Almost all the instruction in fetch cycle are the same with LDA, ADD, SUB, OUT, HLT.

Execute cycle:

Among of six timing bit only T4, T5, and T6 are used for execute cycle.



SAP 2 (SIMPLE AS POSSIBLE 2)

- Bidirectional registers
- Includes jump instructions
- All register outputs to W bus are three-state; those not connected to the bus are two-state

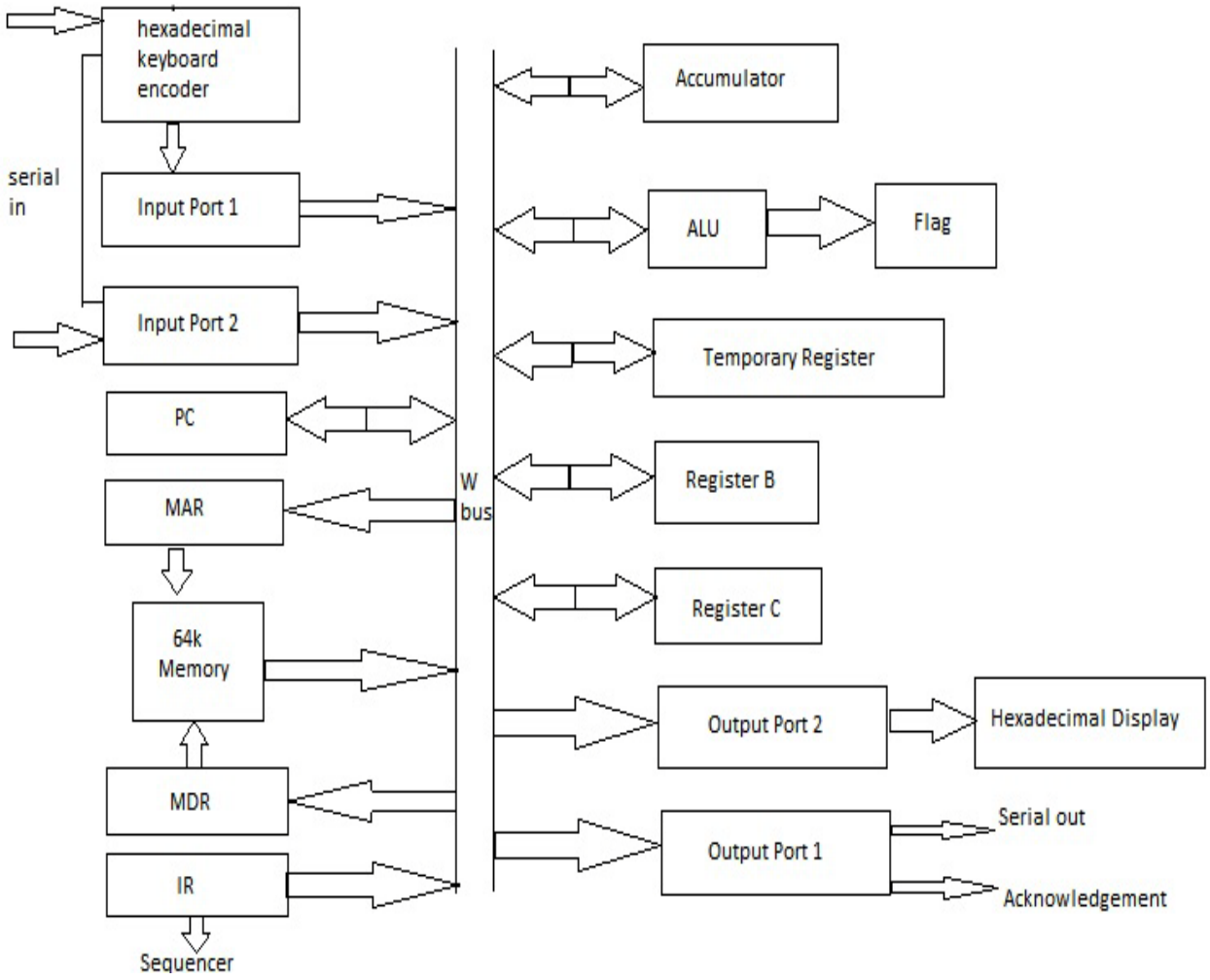


Fig: SAP-2 Architecture.

Input Ports

- 2 input ports numbered 1 and 2
- Hexadecimal keyboard encoder connected to port 1, sends READY signal to bit 0 of port 2
- This signal indicates when the data in port 1 is valid, the SERIAL IN, signal going to pin 7 or port 2.

Program Counter

- Program counter has 16 bits; therefore it can count from PC= 0000 0000 0000 000 to PC=1111 1111 1111 1111
- Its job is to send to the memory the address of the next instructions to be fetched and executed
- A low resets the PC before each computer run; so the data processing starts with the instruction stored in memory location 0000H

MAR and Memory

- During the fetch cycle, the MAR receives 16 bit addresses from PC. The two state MAR output then addresses memory location.
- The memory has 2K ROM with address 0000H too 07FFH. The ROM contains a program called monitor that initializes the computer on power-up, interprets the keyboard inputs and so on.
- The rest of memory is 64 K RAM with addresses from 0800H to FFFFH.

MDR(Memory Data Register)

- 8 bit buffer register
- Its output sets up the RAM
- Receives data from the bus before a write operation and it sends data to the bus after a read operation.

IR(Instruction Register)

- Part of control unit
- Memory read operation performed by computer to fetch an instruction from memory; this places the contents of the addressed memory location on W bus
- Contents split into 2 nibbles
- SAP 2 use 8 bits for op code which can accommodate 256 instructions

Controller Sequencer

- Produces the control words or microinstructions that coordinate and direct the rest of the computer
- Has more hardware since SAP 2 has bigger instruction set
- Microinstruction determines how the registers react to the next positive clock edge

Accumulator

- A buffer register that stores intermediate answers during the computer run
- Has two outputs two-state and three-state
- Two-state output goes to ALU and three-state to W bus
- Hence the 8 bit word in the accumulator continuously drives the ALU, but this same word appears on the bus only when EA is active

ALU and Flags

- Includes arithmetic and logic operations

- Has 4 or more control bits that determine the arithmetic or logic operation performed on words A and B
- Flag is a flip-flop that keeps track of a changing condition during a computer run
- SAP-2 has two flags; sign and zero flag

TMP, B and C Registers

- TMP(temporary) register is used instead of register B to hold data being added or subtracted; which allows us more freedom in using the B register

- Besides B and TMP SAP 2 has a register C which gives us more flexibility in moving data during the computer run

Output Ports

- Two output ports numbered 3 and 4
- Contents of the accumulator can be loaded into port 3, which drives a hexadecimal display
- The contents can also be seen through port 4

Architectural difference between SAP 1 and SAP 2 architecture

Major differences:

- Bidirectional registers
- Flags
- Large instruction set having jump, call & loop.
- Standard Input Output devices in SAP 2.
- Large no. of processor registers in SAP 2.
- High memory capacity(64K) in SAP 2.
- Address bus 16 bit in SAP 2.
- Separate MAR & MDR in SAP 2.
- Complex programming possible in SAP 2
- Introduction of ALU in SAP 2
- Serial as well as parallel I/O in SAP 2.

Flag

- Represent the status of the arithmetic and logical operation.
- flip-flop concepts are used in flag to represent the status of arithmetic and logical operation.
- Two types of flag are there sign flag and zero flag.

1)Sign flag: gets set when the accumulator content is negative.
2)Zero flag : gets set when the accumulator content is negative**.**

Instruction Set Forewords:

- The basic set of commands, or instructions, that a microprocessor understands.
- Also called the ―Command Set.
- Deals with programming, including the native data types, instructions, memory architecture and external I/O Devices.
- Step-by-step processes that are loaded into the memory before the start of a computer run.
- It is essential to understand the Instruction Set before programming a computer.
- Includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor.

The SAP-2 Instruction Set:

MEMORY REFERENCE INSTRUCTIONS

1. LDA and STA :

LDA(Load to Accumulator):

- Loads the accumulator with the content of memory location.
- More memory location can be accessed in SAP-2 than SAP-1 because the addresses are from 0000H to FFFFH.
- E.g. LDA 2000H: loads the accumulator with content of memory location 2000H

STA(Store the Accumulator):

- Stores the accumulator contents at different memory locations.
- E.g. STA 7FFFH: To store the accumulator contents at memory location 7FFFH. i.e. If A=8AH, STA 7FFFH stores 8AH at address 7FFFH.

2. MVI (Move Immediate):

- Loads a designated register with the byte that immediately follows the op code.
- E.g. MVI A,37H loads accumulator with 37H and binary content of A becomes A=0011 0111

REGISTER INSTRUCTIONS

1. MOV (Move):

Moves data from one register to another register. E.g. MOV A,B: Copies data from B to A

2. ADD and SUB:

Adds the content of assigned register to accumulator. E.g. If A=04H and B=02H, ADD B gives result in A=06H Subtract the content of the assigned register from the accumulator.

3. INR and DCR (Increment and Decrement):

Increases and decreases the register. E.g. If B=56H and C=8AH then INR B gives B=57H DCR C gives C=89H

JUMP and CALL INSTRUCTIONS

1. JMP (Jump):

Tells the computer to get the next instruction from the designated memory location.

- E.g. JMP 3000H gives next instruction from memory location 3000H.

2.JM (Jump if Minus):

- Jump to designated address if and only if the sign flag is set.
- E.g. Let JM 3000H is stored at 2005H. After this instruction is fetched, PC=3000H is executed if S=1 else (S=0) instruction is fetched from 2006H.

3. JZ (Jump if Zero):

- Tells the computer to jump to designated address only if the zero flag is set.
- E.g. JZ=3000H is stored at 2005H. Next instruction is fetched from 3000H only if Z=1 else from 2006.

4. JNZ (Jump if Not Zero):

We get a jump when the zero flag is clear and no jump when it is set. (Vice-versa of JZ)

5. CALL and RET:

CALL the Subroutine : E.g. CALL 5000H will jump to the square root subroutine and CALL 6000H produces a jump to the logarithm subroutine.

RETURN: Used at the end of every subroutine to tell the computer to go back to the original program**.**

LOGIC INSTRUCTIONS

1. CMA (Complement the Accumulator):

Inverts each bit in the accumulator, producing a 1's complement.

2. ANA (AND the Accumulator):
ANDs the content of accumulator with the content of specified register. E.g. A = 1100 1100, B = 1111 0001 ANA B results A = 1100 0000
3. ORA (OR the Accumulator):
ORs the content of accumulator with the content of specified register. E.g. A = 1111 1101, B = 1111 0001 ORA B results A = 1111 1101
4. ANI (AND Immediate):
ANDs the accumulator content with the byte that immediately follows the op code. E.g. A = 0101 1110 and ANI C7H gives A = 0100 0110 since C7= 1100 0111.
5. ORI (OR immediate):
The accumulator contents are ORed with the byte that follows the op code.
6. XRI (XOR immediate):
A = 0001 1100 XRI D4H will XOR 0001 1100 and 1101 0100 to produce A = 1100 1000 (High only if different inputs else low if same inputs)
OTHER INSTRUCTIONS

1. NOP (No Operation):
Do nothing. (Used to waste time) By repeating NOP a number of times, we can delay the data processing, which is useful in timing operations.
2. HLT (Halt):
Ends the data processing.
3. RAL (Rotate the accumulator Left):
Shift all bits to the left and move the MSB to the LSB position. E.g. A = 1011 0100 Then, RAL will produce A = 0110 1001
4. RAR (Rotate the accumulator Right):
The bits shift to the right, the LSB going to the MSB position. E.g. A = 1011 0100 Then, RAR results A = 0101 1010
5. IN (Input):
Tells a computer to transfer data from the designated port to the accumulator. E.g. IN 02H means to transfer the data in port 2 to the accumulator.
6. OUT *(Output):*
When this instruction is executed, the accumulator word is loaded into the designated output port. E.g. OUT 03H will transfer the contents of the accumulator to port 3.

Chapter:3
Intel 8085 Microprocessor

The 8085 µp is an 8 bit general purpose microprocessor having 16 bit address lines (capable of addressing 2^{16}= 65536 bytes= 64 KB of memory). The device has 40 pins, requires a +5V single power supply and can operate with 3 MHZ single phase clock.

3.1 Internal Architecture of and features of 8085 Microprocessor

The main components of 8085 µp, as shown in functional block diagram, are the arithmetic/logic unit (ALU), Register array, timing and control unit, instruction register and decoder, interrupt control and serial I/O control. These are linked by an internal data bus.

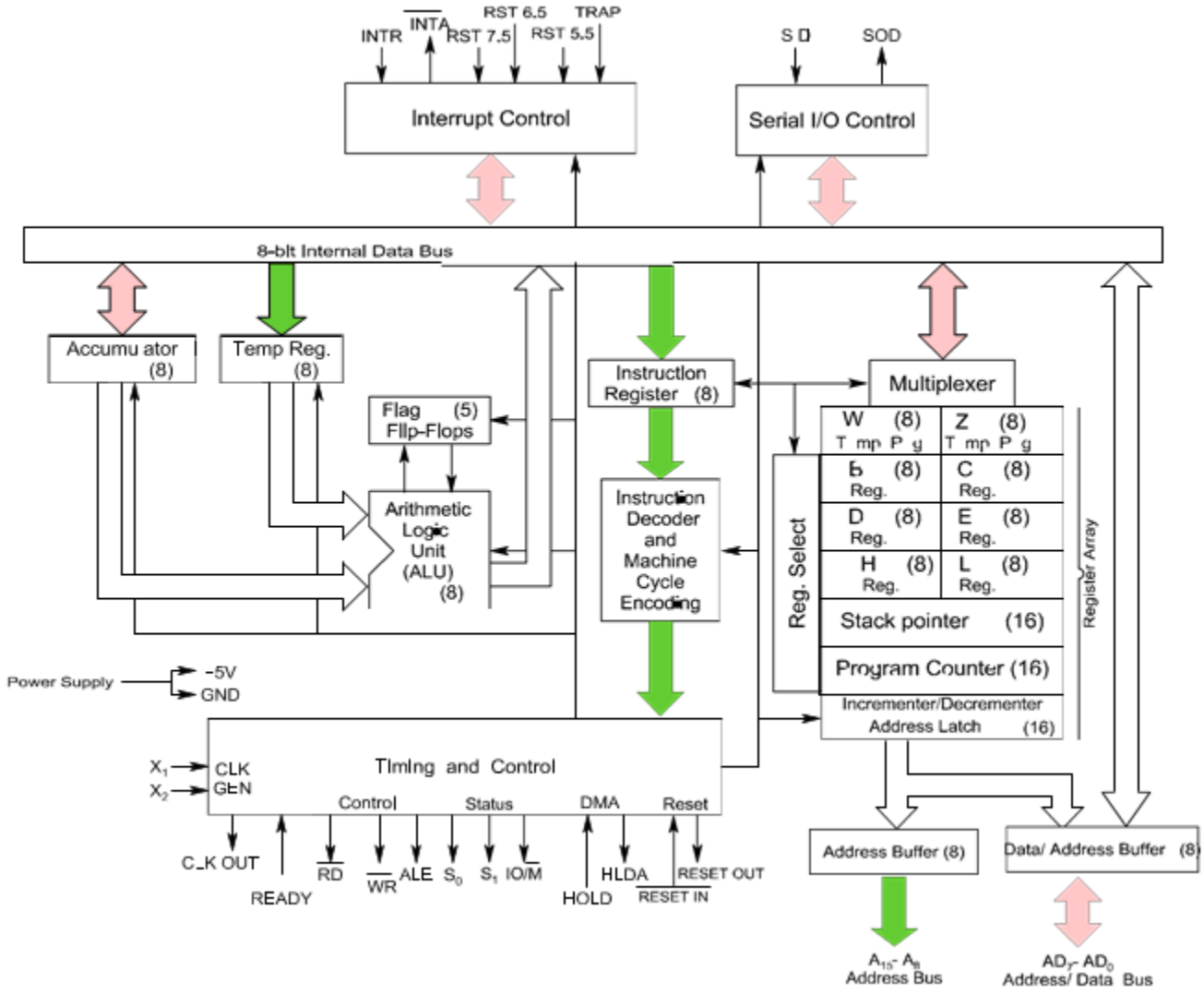


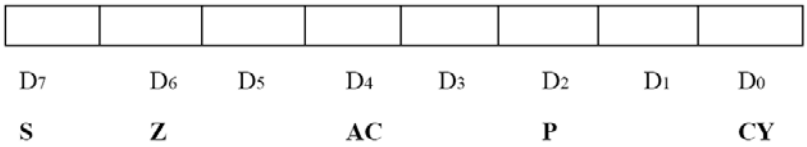
Figure: functional block diagram of 8085 microprocessor

The major components of the block diagram are described as follows.

ALU:
The ALU performs the computing functions. It includes the Accumulator, temporary register, arithmetic and logic circuits, and five flags.

- Accumulator:
  The accumulator (register A), is an 8 bit register, accessible to the programmer. Actually 8085 µp is an accumulator based µp, because one of the depends of the operations is the accumulator itself. So, it plays key role in the operation of 8085 µp. This register is used to store 8 bit data and to perform arithmetic and logical operations. The results of an operation is stored in the accumulator.
- Temporary register:
  The temporary register is an 8 bit register not accessible to the programmer. It is used to hold data during an arithmetic/logic operation for a brief period.
- Flags:
  The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers, indicating the status of different states separately. These flags are Sign(S), Zero(Z), Auxiliary Carry(AC), Parity(P) and Carry(CY) flags. They are described as follows:
  - S- Sign flag:
    After the execution of arithmetic or logic operation, if D7 bit (MSB: most significant bit) of the result (usually in accumulator) is 1, the sign flag (S) is set. Otherwise it is reset.*(For signed number operation D7 bit indicates the sign(positive or negative) of number).*
  - Z- Zero flag:
    Set i.e. 1, if the result of last operation is zero, and the flag is reset i.e. 0 if the result is not 0. This flag is often used in loop control and in searching for particular data value.
  - AC- Auxiliary Carry:
    In an arithmetic operation, when a carry is generated by digit D3 and passed on to digit D4, the AC flag is set. The flag is used only internally for BCD operations and is not available for programmer.
  - P- Parity flag:
    Set i.e. 1, if the number of 1 in the result of the last operation is even. If the result has an odd number of 1's the flag is reset i.e. 0.
  - C- Carry flag:
    Flag is set i.e. 1, if the result of the last operation generates a carry, otherwise it is reset i.e. 0.

    The bit positions reserved for these flags in flag register is as follows;



Instruction register and decoder :

The instruction register receives the operation codes of instructions from the internal data bus and passes it to the instruction decoder and machine cycle encoder circuit. The decoder decodes the instruction and establishes the sequence of events to follow. The instruction register is not accessible to the programmer.

Register array :

The 8085 µp has 6 general purpose registers to hold 8 bit data; these are B, C, D, E, H, and L. They can also be combined as register pairs BC, DE, And HL to perform 16 bit operation. These registers are accessible to programmer i.e. programmable. In addition, the register H and L are utilized in indirect addressing mode. In this mode, the memory location whose address is specified by contents of the register pair.

Register array also includes two additional register W & Z, called temporary registers, each used to hold 8 bit data during the execution of some instructions. However, they are not available to programmer.

Program counter (PC) and Stack Pointer (SP) are two 16 bit registers used to hold memory addresses.

A stack is an area of R/W memory set aside for the purpose of storing data, by an operation known as stacking. The beginning of stack is defined by loading a 16 bit address in the stack pointer register.

A computer program consists of the sequence of coded instructions. These instructions are stored sequentially in the memory location. As 8085 µp begins to execute an instruction, the memory address of the first instruction to be executed is placed in the register called program counter. Its function is to point to the memory address from which the next byte of instruction is to be fetched. Depending upon the instruction type the program counter is incremented by one, two, or three to point to the next memory location.

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 salient features of 8085 µp are:

- ✓ It is a 8 bit microprocessor.
- ✓ It is manufactured with N-MOS technology.
- ✓ It has 16-bit address bus and hence can address up to 2^{16} = 65536 bytes (64KB) memory locations through A0-A15.
- ✓ The first 8 lines of address bus and 8 lines of data bus are multiplexed AD0 – AD7.
- ✓ Data bus is a group of 8 lines D0 – D7.
- ✓ It supports external interrupt request.
- ✓ A 16 bit program counter (PC)
- ✓ A 16 bit stack pointer (SP)
- ✓ Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- ✓ It requires a signal +5V power supply and operates at 3 MHZ single phaseclock.
- ✓ It is enclosed with 40 pins DIP (Dual in line package).
- ✓ It has 74 operation codes with total 246 instructions.

3.2 Pin configuration of 8085 microprocessor

A microprocessor based system consists of a set of components or modules of three basic types: CPU, memory and I/O units, which communicate with each other. A bus is a group of wires used for communication pathway between two or more such components. The fundamental characteristic of a bus is that, it is a shared transmission medium. Number of devices is connected to the bus and a signal transmitted by a signal device is available for reception by all other devices attached. Data signal to the bus will be send by one device at a time successfully because if two devices transmit at the same period, their signals get overlapped and become garbage. The bus that connects major microcomputer components such as CPU, memory, I/O is called the system bus.

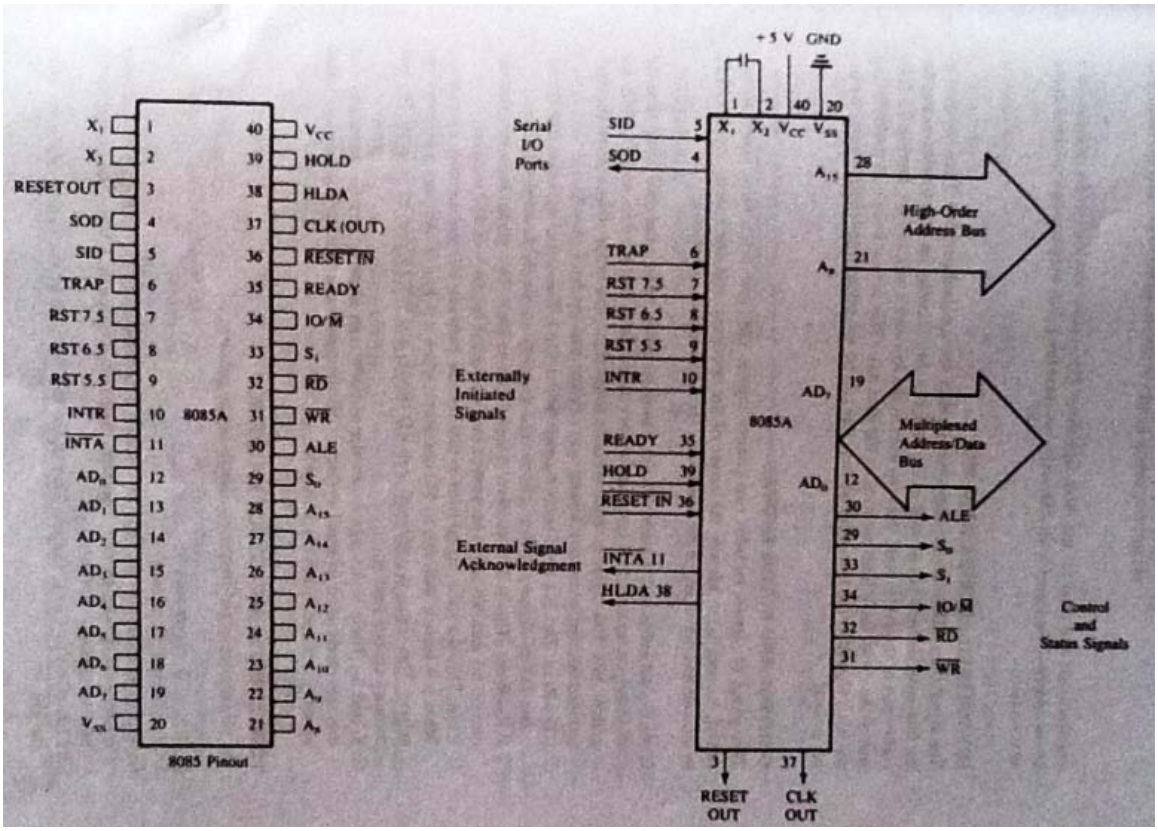


Fig: Pin diagram and signals of 8085 microprocessor

The 8085 µp is an 8 bit general purpose microprocessor capable of addressing 64 KB of memory. The device has 40 pins, requires a +5V single power supply and can operate with a 3-MHz single phase clock.

The 8085 pin signals can be classified into following groups:

1. Address signals (A_{15}-A_8)
2. Multiplexed Address/Data signals (AD_7-AD_0)
3. Control and Status signals
4. Power Supply and frequency signals

5. Externally initiated signals
6. Serial I/O signals

- **Address signals (A_{15}-A_8)**

The 8085 µp has 16 address signal lines hence can address up to 2^{16}=65536 bytes (64 KB) memory locations through A_{15} -A_0. However, these lines are split into two segments: A_{15}-A_8 and AD_7-AD_0. The 8 signal lines, A_{15}-A_8 are **unidirectional** and used for most significant bits called higher order address of a 16-bit address. Tri- stated during 'HOLD' and 'HALT' modes and during RESET.

- **Multiplexed Address/Data signals (AD_7-AD_0)**

The signal lines AD_7-AD_0 are **bidirectional**: they serve a dual purpose. They are used as the low-order memory address (or I/O address) bus as well as the data bus. In executing an instruction, during the earlier part of the cycle, these lines are used as the address bus; and during the later part of the cycle, these lines are used as the data bus. Tri- stated during 'HOLD' and 'HALT' modes and during RESET.

- **Control and status signals**

This group of signals includes two control signals (RD& WR), three status signals (IO/ M, S_1 and S_0) to identify the nature of the operation and one special signal (ALE= address latch enable) to indicate the beginning of the operation. These signals are as follows;

- **ALE**: Address Latch Enable; This is a positive going pulse generated every time the 8085 begins an operation (machine cycle); it indicates that the bits on AD_7-AD_0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines, A_7-A_0.
- **RD**: READ; This is a read *control signal* (active low) which indicates that the selected I/O or memory device is to be read and data are available on the data bus. This pin is held in the high impedance state during 'HOLD' and 'HALT' modes and during RESET.
- **WR**: Write; This is a write *control signal* (active low) which indicates that the data on the data bus are to be written into a selected memory or I/O location. This pin is held in the high impedance state during 'HOLD' and 'HALT' modes and during RESET.
- **IO/M**: This is a *status signal* used to differentiate between I/O and memory operations. When it is high, it indicates an I/O operation; when it is low, it indicates a memory operation. This signal is combined with RD(read) and WR(write) to generate I/O and memory control signals.
- **S_1** and **S_0**: These status signals, similar to IO/M, can identify various operations. These status signals go high or low according to the type of different machine cycle as following:

Machine cycle	**Status**			**Control signals**
	IO/M	**S_1**	**S_0**	
Op-code Fetch	0	1	1	RD=0
Memory read	0	1	0	RD=0
Memory write	0	0	1	WR=0
I/O read	1	1	0	RD=0
I/O write	1	0	1	WR=0
Interrupt Acknowledge	1	1	1	INTA=0

Halt	Z	0	0	WR=RD=Z, INTA=1
Hold	Z	X	X	
Reset	Z	X	X	

Z= Tri-state (High impedance)
X= unspecified

- **Power Supply and clock frequency signals:**

The power supply and frequency signals are as follows:

- **V_{cc}** → +5V power supply
- **V_{ss}** → Ground reference
- **X_1, X_2**(Clock inputs) → A crystal is connected at these two external pins to generate the clock signal. The frequency is internally divided by two; therefore to operate a system at 3 MHz, the crystal should have a frequency of 6 MHz.
- **Clock out**→ clock output: This signal can be used as the system clock for other devices.

- **Externally initiated signals and interrupts:**

External devices (or signals) can initiate the following operations, for which individual pins on the microprocessor chip are assigned: *Reset, Interrupt, Ready, Hold*.

If an I/O device needs to use the CPU for processing, it sends a service request signal to the CPU. If it is accepted, the CPU halts(interrupts) its current process, save its contents and branches to the appropriate I/O routine to service that I/O device. Once the service gets completed the CPU returns to the process it was involved before, as to the interrupt request that is next in priority. The 8085 has five interrupt signals, that can be used to interrupt a program execution.

- **INTR (pin10-input):** INTR is a maskable interrupt request pin. It uses handshaking. A 'HIGH' level on this pin causes the 8085 to complete execution of the current instruction, push the current program counter contents in the stack and issues an INTA signal. At this state either a 1 byte call (RST 0 – RST 7) or a 3 byte CALL instruction can be inserted so as to jump to the interrupt service routine. The INTR pin is monitored only during the last two clock periods of an instruction cycle and during the HOLD and HALT modes.
- **INTA (pin 11-output):** Interrupt acknowledge pin: the CPU places a low level signal on this pin to indicate that an interrupt request signal from the internal device has been acknowledged and is ready to be served. The occurrence of INTA turn off the 8085 interrupt system in order to avoid multiple interrupts from a single device.
- **RST 5.5 (pin 9-input):** Restart interrupt input is a maskable interrupt. It can be enabled or disabled using SIM instruction. RST 5.5 responds to a 'HIGH' level on its interrupt pin. This causes the 8085 to complete execution of the current instruction, push the current contents of the program counter in the stack and branch to location **002CH**.
- **RST 6.5 (pin 8-input):** Restart interrupt input is a maskable interrupt. It can be enabled or disabled using SIM instruction. RST 6.5 responds to a HIGH level on its interrupt pin. This causes the 8085

to complete execution of the current instruction, push the current contents of the program counter in the stack and branch to location **0034H**.

- **RST 7.5 (pin 7-input):** Restart interrupt input is a maskable interrupt. It can be enabled or disabled using SIM instruction. RST 7.5 responds to a HIGH level on its interrupt pin. This causes the 8085 to complete execution of the current instruction, push the current contents of the program counter in the stack and branch to location **003CH**.
- **TRAP (pin 6-input):** The trap interrupt input in non-maskable interrupt and has the highest priority. It cannot be disabled by an instruction. TTAO interrupt requires both a leading edge and HIGH level to respond to this interrupt. If the TRAP pin has a leading edge and a sustained high level, the microprocessor completes the current instruction, pushes the program in the stack, and the branches to the location **0024H**.

All of the RST interrupts have a higher priority than INTR. The priority of these 5 interrupt signals is as follows:

Interrupt signal	**Priority**	**Destination(branching) location**
TRAP	1	0024H
RST 7.5	2	003CH
RST 6.5	3	0034H
RST 5.5	4	002CH
INTR	5	XXXX (*call location given either a 1 byte call (RST 0- RST 7) or a 3 byte CALL instruction)*

In addition to the interrupts, three pins – RESET, HOLD, and READY, accept the externally initiated signals as inputs.

- **HOLD (Input):** When the HOLD pin is activated by an external signal, the microprocessor give up the control of buses and allows the external peripheral to use them. For example, the HOLD signal is used to Direct Memory Access (DMA) data transfer.
- **HLDA (Output):** To respond to the HOLD request, the 8085 has one HLDA (*hold acknowledge*) signal.

There a two RESET signal:

- **RESET IN:** When the signal on this pin goes low, the program counter is set to zero, the buses are tri-stated, and the microprocessor is reset.
- **RESET OUT:** This signal indicates that the microprocessor is being reset. This signal can be used to reset other devices.
- **READY:** It is used to find whether a peripheral device is ready for data transfer or not. If this signal goes high during a Read or Write cycles, it indicates that the memory or peripheral is ready to send or receive data. If READY is low, microprocessor will wait an integral number of clock cycles until it goes high.

- **Serial I/O signals:**

The 8085 microprocessor has two signals to implement the serial transmission: SID (Serial input data) and SOD (Serial output data). In serial transmission, data bits are sent over a single line, one bit at a time.

- **SID** : This input signal is used to accept serial data bit by bit from the external device.
- **SOD** : This output signal is used to output data serially bit by bit.

  Note: Serial Data transfer is controlled through two instructions: SIM & RIM.

3.3Instruction Format:

An instruction is a binary pattern designed inside a µp to perform a specific function. The entire group of instructions, called the instruction set. Each instruction has two parts: one is the task to be performed, called the operation code (Op-code), and the other is data to be operated on, called the operand. Operand may include 8 bit or 16 bit data, and internal register, a memory location, or an 8 (or 16 bit) address. In some instructions, the operand is implicit.

The 8085 instruction set is classified into 3 groups according to word or byte size.

1) 1- byte instructions
2) 2- byte instructions
3) 3- byte instructions

1) One byte instructions

It includes the op-code and operand in the same byte. These instructions are stored in 8 bit binary format in memory; each requires one byte memory location.

e.g.

Memory address	Op-code	Operand	Hex code	Description
3000	MOV	C,A	4F H	Copy the contents of the Accumulator inregister C
3001	ADD	B	80 H	Add the contents of register B to the contents of the Accumulator
3002	CMA		2F H	Invert (compliment) each bit in the Accumulator

2) Two byte instructions

In this type of instruction, first byte specifies the operation code and the second byte specified the operand.

e.g.

Memory address	Op-code	Operand	Hex code	Description
3100	MVI	A,32 H	3E H	Load an one byte data
3101			32 H	(32H) in the AC
8080	MVI	B, F2 H	06 H	Load an one byte data
8081			F2 H	(F2H) in reg. B

These instructions would require two memory locations each to store the binary codes.

3) Three byte instructions

In 3 byte instructions, the first byte specifies the Op-code, and the following two bytes specify the 16 bit address or data. (*Note: second byte is the low-order address and the third byte is the high-order address).*

e.g.

Memory address	Op-code	Operand	Hex code	Description
3000	LDA	2050 H	3A H	Load contents of memory
3001			50 H	2050H into accumulator
3002			20 H	
8000	JMP	2085 H	C3 H	Transfer the program
8001			85 H	sequence to memory
8002			20 H	location 2085H
9000	LXI	D,1234 H	11 H	Loads 12H in register D and
9001			34 H	34H in register E.
9002			12 H	

These instructions would require 3 memory locations each to store the binary codes.

3.4Addressing Modes of 8085

To perform any operation, we have to give the corresponding instructions to the microprocessor.
In each instruction, we have to specify the following three things:

- Operation to be performed.
- Address of source of data.
- Address of destination of result.

The method by which the address of source of data or the address of destination of result is given in the instruction is called **Addressing Mode**. The term addressing mode refers to the way in which the operand of the instruction is specified.

Types of Addressing Modes

Intel 8085 uses the following addressing modes:

1. Direct Addressing Mode
2. Register Direct Addressing Mode
3. Register Indirect Addressing Mode
4. Immediate Addressing Mode
5. Implied Addressing Mode

Direct Addressing Mode

Instructions using this mode specify the effective address as part of instruction. Instructions using this mode may contain 2 or 3 bytes, with first byte as the Op-code followed by 1 or 2 bytes or address of data. In this mode, the address of the operand is given in the instruction itself.

LDA 2500 H ➔Load the contents of memory location 2500 H in accumulator.

LDA is the operation.2500 H is the address of source.Accumulator is the destination.

IN 41 H ➔Reads the data at port 41 H and store data at the accumulator
IN is the operation. 41 H is the address of source. Accumulator is the destination.

Register Direct Addressing Mode
In this mode, the operand is in general purpose register i.e. data is provided through registers.
MOV A, B ➔Move the contents of register B to A.
MOV is the operation.B is the source of data.A is the destination.

Register Indirect Addressing Mode
In this mode, the address part of instruction specifies the memory location whose content is the address of the operand. In 8085 µp, wherever the instruction uses the HL pointer the address is called indirect addressing.
MOV A, M ➔Move data from memory location specified by H-L pair to accumulator.
MOV is the operation.M is the memory location specified by H-L register pair.A is the destination.

Immediate Addressing Mode
In this mode, the operand is specified within the instruction itself. This mode of instructions uses first byte as the Op-code and following 1 or 2 byte data itself.
MVI A, 05 H ➔Move 05 H in accumulator.
MVI is the operation.05 H is the immediate data (source).A is the destination.
LXI H, 7A21 H ➔Loads register H with 7A H and register L with 21 H
LXI is the operation. 7A21 H is the immediate data (Source). H & L registers are the destination.
So in both cases, the actual data is part of the instruction, and hence called immediate addressing.
Implied Addressing Mode
If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction. So, instructions of such mode don't have operands.
CMA ➔Complement accumulator.
CMA is the operation.A is the source.A is the destination.

3.5 Instructions Set

The entire group of Instructions can be classified into the following five functional categories.

1. Data Transfer Instructions Set
2. Arithmetic Instructions Set
3. Logical Instructions Set
4. Branching Instructions Set
5. Miscellaneous Instructions Set

1) Datatransferinstructions set:

The data transfer group is the longest group of instructions in 8085 and consists of 88 different op-codes. This group of instructions copy data from a location called a source to another location, called destination, 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 flags.

1) MOV R_d,R_S $R_d \leftarrow R_S$ (move register instruction)

- 1 byte instruction

- Copies data from source register to destination register.
- Rd&Rs may be A, B,C,D,E,H &L
- E.g. MOVA,B

2) **MVIR, 8bitdata**(move immediate instruction)
   - 2 byte instruction
   - Loads these cond byte(8 bit immediate data)into the register specified.
   - R may beA,B,C,D,E,and H& L
   - E.g. MVI C,53H C ←53H
3) **MOV M,R**(Move to memory from register)
   - Copy the contents of the specified register to memory. Here memory is thelocation specified by contents of the HL register pair..
   - E.g. MOV M,B
4) **MOV R,M** (move to register from memory)
   - Copy the contents of memory location specified by HL pair to specified register.

     .
   - E.g. MOVB,M

Write a program to load memory locations 7090 H and 7080 H with data 40H and 50H and then swap these data.

Soln:

```
MVI H,70H
MVI L,90H
MVI A,40H
MOV M,A
MOVC,M
 MVI L,80H
MVI B, 50H
MOV M,B
MOV D,M
MOV     M,C
MVI   L,90H
MOV     M,D
HLT
```

5) **LXIRp,2 bytes data**(load register pair)
   - 3-byte instruction
   - Load immediate data to register rpair
   - Register pair may be BC, DE,HL&SP (Stack pointer)

- 1^{st}byte-Op-code
- 2^{nd}byte– lower order data
- 3^{rd}byte- higher order data
- E.g. LXI B, 4532H;B←45,C←32H

6) MVIM, data(load memory immediate)

- 2 byte instruction.
- Loads the8-bit data to the memory location whose address is specified by the contents of HL pair.E.g. MVI M,35H;[HL] ←35H

7) LDA4035H(Load accumulator direct)

- 3-byte instruction
- Loads the accumulator with the contents of memory location whose address is specified by16 bit address.
- A←[4035H]

8) LDAX Rp(Load accumulator indirect)

- 1 byte instruction.
- Loads the contents of memory location pointed by the contents of register pair to accumulator.

– E.g. LDAXB[A] ⟵ [[BC]]

LXI B,9000H ⟶ B=90,C= 00

LDAXB ⟶ A=[9000]

9) STA16-bit address (store accumulator contents direct)

- 3-byte instruction.
- Stores the contents of accumulator to specified address
- E.g. STAFA00H [FA00]←[A]

10)STAX RP [RP] ←A

- Store s the contents of accumulator to memory location specified by the contents of register pair.1byte instruction

- E.g. STAXB

```
LXI B,9500H          output
LXI D,9501H          [9500]=32
MVI A , 32H          [9501]=7A
STAXB
MVI A,7AH
STAXD                [DE] ←A
```

11)IN8-bit address

– 2-byte instruction
– Read data from the input port address specified in the second byte and loads datainto the accumulator i. e. input port content to accumulator:

– E.g. IN40H A ←[40H]

12)OUT8-bitaddress

– 2-byte instruction
– Copies the contents of the accumulator to the output port address specified in the2nd byte. That means accumulator to output port: P ← A
– E.g. OUT40H [40] ←A

13)LHLD 16-bitaddress(Load HL directly)

3-byte instruction.
– Loads the contents of specified memory location to L–register and contents of next higher location to H-register.

E.g. LXIH,9500H

MVI M, 32H

MVI L,01H

MVI M,7AH

LHLD9500H H=7A,L=32

32	9500
7A	9501

14)SHLD16-bitaddress(storeHL directly)

– Opposite to LHLD.
– Stores the contents of L-register to specified memory location and contents of H-register to next higher memory location.
– E .g. LXI H, 9500H [8500]=00
SHLD 8500H [8501]=95

15)XCHG(Exchange)

– Exchanges DE pair with HL pair.

– E.g. LXIH,7500H H=75,L=00
LXID,9532H D=95. E=32
XCHG H=95,L=32
D=75 E=00

2) Arithmetic Instructions Set

Instructions that perform arithmetic operations such as addition, subtraction, increment and decrement fall under this Instruction Set. The common features of all instructions except INR & DCR are:

a) The instructions assume implicitly that the accumulator is one of the operands.

b) The flags got modified according to the data condition of result, when the instructions are executed.
c) The result is placed in the accumulator.
d) The execution of the instruction does not affect the contents of operand register.

The instructions INR and DCR affect the contents of the specified register and all the flags except carry flag.

1) ADD R/M

- 1 byte add instruction.
- Adds the contents of register/memory to the contents of the accumulator and stores the result in accumulator.
- E. g. Add B; A ←[A] + [B]

2) ADI 8 bit data

- 2 byte add immediate instruction.
- Adds the 8 bit data with the contents of accumulator and stores result in accumulator.
- E g. ADI 9BH ; A←A+9BH

3) SUB R/M

- 1 byte subtract instruction.
- Subtracts the contents of specified register / m with the contents of accumulatorand stores the result in accumulator.
- E. g. SUB D ; A ←A-D

4) SUI 8bit data

- 2 byte subtract immediate instruction.
- Subtracts the 8 bit data from the contents of accumulator stores result in accumulator.
- E. g. SUI D3H; A ←A-D3H

5) INR R/M, DCR R/M

- 1 byte increment and decrement instructions.
- Increase and decrease the contents of R(register) or M(memory) by 1 respectively.
- E. g. DCR B ; B=B-1
  DCR M ; [HL] = [HL]-1
  INR A ; A=A+1
  INR M ; [HL] +1
  For these, all flags are affected except carry.

.

5) INX Rp, DCX Rp

- Increase and decrease the register pair by 1.
- Acts as 16 bit counter made from the contents of 2 registers (1 byte instruction)
- E.g. INX B ;BC=BC+1
  DCX D ;DE=DE+1
- No flags affected

7) ADCR/M and ACI 8-bit data(addition with carry (1 byte))

- ACI 8-bit data= immediate (2 byte).

– Adds the contents of register or 8 bit data whatever used suitably with thePrevious carry.

– E.g. ADC B ; A ← A+B+**CY** ACI 70H ; A ← A + 70+**CY**

8) SBB R/M

– 1 byte instruction.

– Subtracts the contents of register or memory from the contents of accumulator andstores the result in accumulator.

– e. g. SBB D ; A←A-D-**Borrow**

SBI 8 bit data

– 2 byte instruction.

– Subtracts the 8-bit immediate data from the content of the accumulator and storesthe result in accumulator.

– E.g. SBI 70H ; A←A-70-**Borrow**

9) DAD Rp(double addition)

– 1 byte instruction.

– Adds register pair with HL pair and store the 16 bit result in HL pair.

– E. g. LXI H, 7320H
LXI B, 4220H

10)DAA(Decimaladjustmentaccumulator)

– Usedonly afteraddition.

– 1 byteinstruction.

DAD B; HL=HL+BC
7320+4220=B540

10) DAA (Decimal adjustment accumulator)

– Used only after addition.

– 1 byte instruction.

– The content of accumulator is changed from binary to two 4-bit BCD digits.

– E. g MVI A, 78H ; A=78
MVI B, 42H ; B=42
ADD B ; A=A+B = BA
DAA ; A=20, CY=1

The arithmetic operation add and subtract are performed in relation to the contents of accumulator. The features of these instructions are

1) They assume implicitly that the accumulator is one of the operands.
2) They modify all the flags according to the data conditions of the result.
3) They place the result in the accumulator.
4) 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

1) Affect the contents of specified register or memory.

2) Affect the flag except carry flag.

Addition operation in 8085:

8085 performs addition with 8-bit binary numbers and stores the result in accumulator. If the sum is greater than 8-bits (FFH), it sets the carry flag.

E.g.	MVI A,93H	1 0 1 1 0 1 1 1	B7
	MVI C,B7H	+1 0 0 1 0 0 1 1	+93
	ADD C	**101 0 0 1 0 1 0**	**1** 4A
		CY	**CY**

Subtraction operation in8085:

8085 performs subtraction operation by using 2‘s complement and the steps used are:

1) Converts the subtrahend (the number to be subtracted) into its 1‘s complement.
2) Adds 1 to 1‘s complement to obtain 2‘s complement of the subtrahend.
3) Adds 2‘s complement to the minuend (the contents of the accumulator).
4) Complements the carry flag.

E.g. MVI A,97H 65H:0 1100101

MVI B, 65H 1' s complement of 65H:1 0011010

SUB B +1

2's Complement of 65H: 1 0011011

97H: +1 00 10 1 11

1 0 01 1 0 0 1 0

32

CY

Complement carry =0 **A=32 CY=0**

B=97H,A=65H

97H: 1 00 1 0 1 1 1

MVI 1's complement of 97H :0 11 0 10 0 0

SUB B +1

2's Complement of 97H: 0 1101001

65H: +0 11 00 1 01

0 1 1 0 0 1 1 1 0

(Result in 2'scomplement form)

CY

CY=1, A= CE: 1 1 0 0 1 1 1 1 0

1's complement: 0 0 1 1 0 0 0 0 1

2's complement: 0 0 1 1 0 0 0 1 0

32

1. **The memory location 2050H holds the data byte F7H. Write instructions to transfer the data byte to accumulator using different op-codes: MOV, LDAX and LDA.**

   LXI H, 2050H LXI B, 2050H LDA 2050H

   MOV A, M LDAX B

2. **Register B contains 32H, Use MOV and STAX to copy the contents of register B in memory location 8000H.**

   LXI H, 8000H LXI D, 8000H

   MOV M, B MOV A, B

3. **The accumulator contains F2H, Copy A into memory 8000H. Also copy F2H**

directly into 8000H.

```
STA 8000H
```

4.The data 20H and 30H are stored in 2050H and 2051H. WAP to transfer the data to 3000H and 3001H using LHLD and SHLD instructions.

```
MVI A, 20H
STA 2050H        LHLD 2050H
MVI A, 30H       SHLD 3000H
STA 2051H        HLT
```

5. WAP to add two 4 digit BCD numbers equals 7342 and 1989 and store result in BC register.

```
LXI H, 7342H
LXI B, 1989H
MOV A, L
ADD C
DAA
MOV C, A MOV
A, H
ADC B
DAA
MOV B, A
```

BCD Addition:

In many applications data are presented in decimal number. In such applications, it may be convenient to perform arithmetic operations directly in BCD numbers.

The microprocessor cannot recognize BCD numbers; it adds any two numbers in binary. In BCD addition, any number larger than 9 (from A to F) is invalid and needs to be adjusted by adding 6 in binary.

E.g.

```
A: 0000  1010
+0000    0110
-------------
 0001    0000  → 10BCD
```

A special instruction called DAA performs the function of adjusting a BCD sum in 8085. It uses the AC flag to sense that the value of the least four bits is larger than 9 and adjusts the bits to BCD value. Similarly, it uses CY flag to adjust the most significant four bits.

E.g. Add BCD 77 and 48

```
77=  0111  0111
+48=  0100
------------
1000125  1011
   1111 +0110
   ---------
    1 0101
+0110 .....CY=
---------
1 0010 0101→   25BC
               D
```

Logical Instructions Set

Instructions that perform logical operations such as AND, OR, X-OR and complement fall under this Instruction Set. The common features of all logical instructions are:

a) The instructions assume implicitly that the accumulator is one of the operands.
b) The Carry Flag is reset (clear) except for CMA. It modifies Z, P, &S flags according to the data conditions of the result.
c) The result is placed in the accumulator.
d) The execution of the instruction does not affect the contents of operand register.

The logical operations have the following instructions.

1) ANA R/M (the contents of register/memory)
- Logically AND the contents of register/memory with the contents of accumulator.
- 1 byte instruction.
- CY flag is reset and AC is set.

2) ANI 8 bit data
- Logically AND 8 bit immediate data with the contents of accumulator.
- 2 byte instruction.
- CY flag is reset and AC is set. Others as per result

3) ORA R/M
- Logically OR the contents of register/memory with the contents of accumulator.
- 1 byte instruction.
- CY and AC is reset and other as per result.

4) ORI 8 bit data
- Logically OR 8 bit immediate data with the contents of the accumulator.
- 2 byte instruction.
- CY and AC is reset and other as per result

5) XRA R/M
- Logically exclusive OR the contents of register memory with the contents of accumulator.
- 1 byte instruction.
- CY and AC is reset and other as per result.

6) XRI 8 bit data
- Logically Exclusive OR 8 bit data immediate with the content of accumulator.
- 2 byte instruction.
- CY and AC is reset and other as per result.

7) CMA (Complement accumulator)
- 1 byte instruction.
- Complements the contents of the accumulator.
- No flags are affected.

Instruction	**CY**	**AC**
ANA/ANI	0	1

ORA/ORI	0	0
XRA/XRI	0	0

Logically Compare instructions

CMP R/M (1 byte instruction)

CPI 8 bit data (2 byte instruction)

- Compare the contents of register/ memory and 8 bit data with the contents of accumulator.
- Status is shown by flags & all flags are modified.

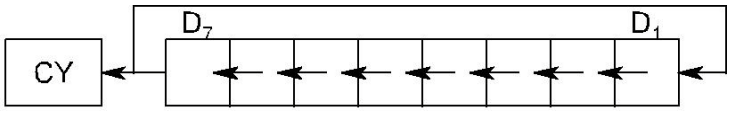
Case	**CY**	**Z**	
[A]<[R/M] or 8 bit	1	0	A-R<0
[A]=[R/M] or 8 bit	0	1	A-R=0
[A]>[R/M] or 8 bit	0	0	A-R>0

Used to indicate end of data.

Logical Rotate instructions

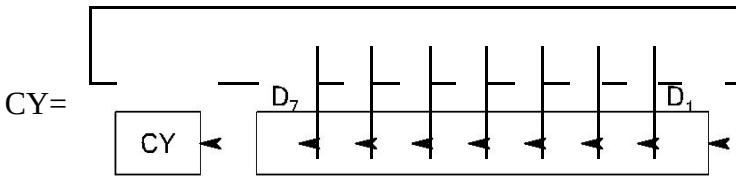
This group has four instructions, two are for rotating left and two are for rotating right. The instructions are:

1) **RLC: Rotate accumulator left**
   - Each bit is shifted to the adjacent left position. Bit D_7becomes D_0.
   - The carry flag is modified according to D_7.



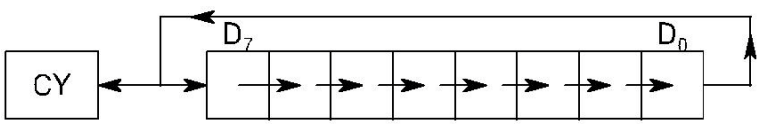
CY= D_7, D_7= D_6, D_6=D_5,……,D_1=D_0, D_0=D_7

2) **RAL: Rotate accumulator left through carry**
   - Each bit is shifted to the adjacent left position. Bit D7 becomes the carry bit andthe carry bit is shifted into D0.
   - The carry flag is modified according to D7.



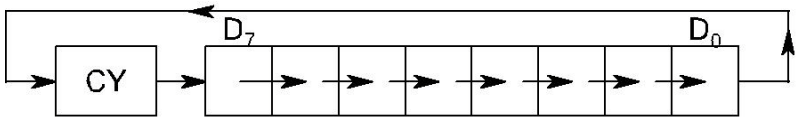
D_7, D_7= D_6, D_6=D_5,……,D_1=D_0, D_0=CY

3) **RRC: rotate accumulator right**
   - Each bit is shifted right to the adjacent position. Bit D_0 becomes D_7.
   - The carry flag is modified according to D_0.
   - The carry flag is modified according to D_0.



CY= D_0, D_7= D_0,……,D_0=D_1

4) **RAR: Rotate accumulator right through carry**
   - Each bit is shifted right to the adjacent position. Bit D_0becomes the carry bit andthe carry bit is shifted into D_7.



CY= D_0 , D_0= D_1, ……..D_7 = CY

Others:

-CMC – Complement carry STC- set carry flag

The rotate instructions are primarily used in arithmetic multiply and divide operations and for serial data transfer.

For e.g. If A=0000 1000=08H

- By rotating 08H right: A=0000 0100=04.
- This is equivalent to division by 2.
- By rotating 08H left: A= 0001 0000 =10H
- This is equivalent to multiplication by 2 (10H = 16_{10})

However these procedures are invalid when logic 1 is rotated from D_7 to D_0 or vice versa.

- E. g 80H rotates left = 01H
- 01 H rotate right = 80H

Q.Explain the instructions that fall in data transfer, arithmetic and logical groups with example: Show how the flags are affected by each instruction.

Branching Group Instructions:

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.

The branching instructions are the most powerful instructions because they allow the microprocessor to change the sequence of a program, either unconditionally or under certain test conditions. The branching instruction code categorized in following three groups:

- Jump instructions
- Call and return instruction
- Restart instruction

Jump Instructions:

The jump instructions specify the memory location explicitly. They are 3 byte instructions, one byte for the operation code followed by a 16 bit (2 byte) memory address. Jump instructions can be categorized into unconditional and conditional jump.

Unconditional Jump

8085 includes unconditional jump instruction to enable the programmer to set up continuous loops without depending only type of conditions. E.g. JMP 16 bit address: loads the program counter by 16 bit address and jumps to specified memory location.

E.g. JMP 4000H

Here, 40H is higher order address and 00H is lower order address. The lower order byte enters first and then higher order.

– The jump location can also be specified using a label or name.E.g.

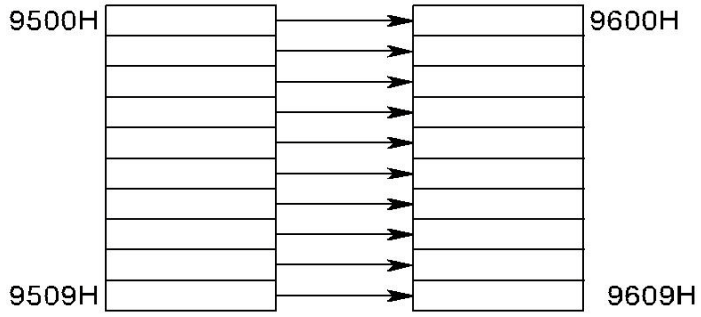
```
        MVI A, 80H          START: IN 00H
        OUT 43H                    OUT 01H
        MVI A, 00H                 JMP START
L1:     OUT 40H
        INR A
        JMP L1
        HLT
```

Conditional Jump

The conditional jump instructions allow the microprocessor to make decisions based on certain conditions indicated by the flags. After logic and arithmetic operations, flags are ser or reset to reflect the condition of data. These instructions check the flag conditions and make decisions to change or not to change the sequence of program. The four flags namely carry, zero, sign and parity used by the jump instruction.

Mnemonics	**Description**
JC 16 bit	Jump on carry (if CY=1)
JNC 16 bit	Jump on if no carry (if CY=0)
JZ 16bit	Jump on zero (if Z=1)
JNZ 16bit	jump on if no zero (if Z=0)
JP 16bit	jump on positive (if S=0)
JM 16bit	jump on negative (if S=1)
JPE 16bit	Jump on parity even (if P=1)
JPO 16bit	Jump on parity odd (if P=0)

E.g. WAP to move 10 bytes of data from starting address 9500 H to 9600H




```
2000  MVI B, 0AH
2002  LXI H, 9500H
2005  LXI D, 9600H
2008  MOV A, M
2009  STAX D          ; Store the contents of accumulator to register pair.
200A  INX H       ; Increment the register pair by 1.
200B  INX D
200C  DCR B
200D  JNZ 2008
2010  HLT
```

Q .Write to transfer 30 data starting from 8500 to 9500H if data is odd else store 00H.

```
    MVI B, 1EH
    LXI H, 8500H
    LXI D, 9500H L2:
MOV A, M
    ANI 01H
    JNZ L1 ; If data is odd then go to L1. MVI A,
    00H
    JMP L3
L1: MOV A, M L3:
STAX D
    INX D
    INX H
    DCR B
    JNZ L2 HLT
```

Call and return instructions: (Subroutine)

Call and return instructions are associated with subroutine technique. A subroutine is a group of instructions that perform a subtask. A subroutine is written as a separate unit apart from the main program and the microprocessor transfers the program execution sequence from main program to subroutine whenever it is called to perform a task. After the completion of subroutine task microprocessor returns to main program. The subroutine technique eliminates the need to write a subtask repeatedly, thus it uses memory efficiently. Before implementing the subroutine, the stack must be defined; the stack is used to store the memory address of the instruction in the main program that follows the subroutines call.

To implement subroutine there are two instructions CALL and RET.

1. **CALL 16 bit memory**
   - Call subroutine unconditionally.
   - 3 byte instruction.
   - Saves the contents of program counter on the stack pointer. Loads the PC by jump address (16 bit memory) and executes the subroutine.
2. **RET**
   - Returns from the subroutine unconditionally.
   - 1 byte instruction
   - Inserts the contents of stack pointer to program counter.
3. **CC, CNC, CZ, CNZ, CP, CM, CPE, CPO**
   - Call subroutine conditionally.
   - Same as CALL except that it executes on the basis of flag conditions.
4. **RC, RNC, RZ, RNZ, RP, RM, RPE, RPO**
   - Return subroutine conditionally.
   - Same as RET except that if executes on the basis of flag conditions.

E.g. **Write an ALP to add two numbers using subroutines.**

```
2000    MVI B, 4AH          3000    MOV A, B
2002    MVI C, A0H          3001    ADD C
2004    CALL 3000H          3002    RET
2007    MOV B, A
2008    HLT                 PC = 2007 (i. e. sp)

SP = 2007 (i.e. PC)         PC = 3000 (i.e. 16bit)
```

Restart Instruction:

8085 instruction set includes 8 restart instructions (RST). These are 1 byte instructions and transfer the program execution to a specific location.

Restart instruction	Hex Code	Call location in hex
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0038H

When RST instruction is executed, the 8085 stores the contents of PC on SP and transfers the program to the restart location. Actually these restart instructions are inserted through additional hardware. These instructions are part of interrupt process.

When RST instruction is executed, the 8085 stores the contents of PC on SP and transfers the program to the restart location. Actually these restart instructions are inserted through additional hardware. These instructions are part of interrupt process.

Miscellaneous Group Instructions:

STACK

The stack is defined as a set of memory location in R/W memory, specified by a programmer in a main memory. These memory locations are used to store binary information temporarily during the execution of a program.

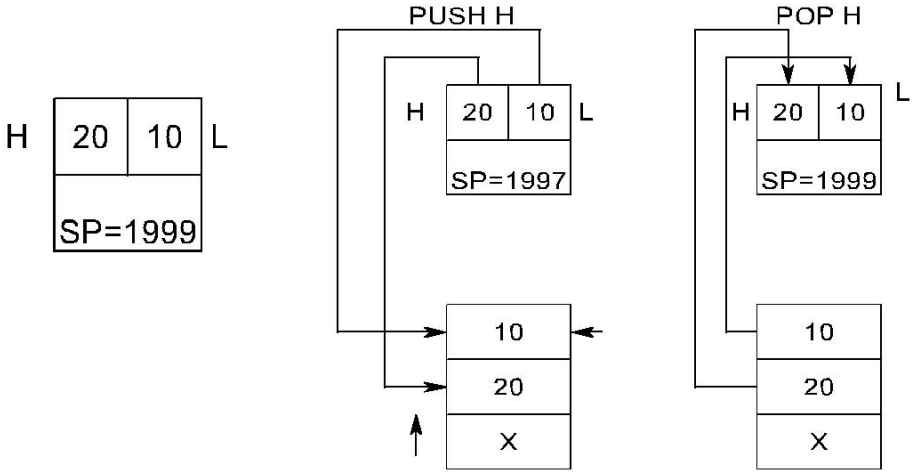
The beginning of the stack is defined in the program by using the instruction LXI SP, 16 bit address. Once the stack location is defined, it loads 16 bit address in the stack pointer register. Storing of data bytes for this operation takes place at the memory location that is one less than the address e.g. LXI SP, 2099H

Here the storing of data bytes begins at 2098H and continuous in reverse order i.e 2097H. Therefore, the stack is initialized at the highest available memory location to prevent the program from being destroyed by the stack information. The stack instructions are:

1. **PUSH Rp/PSW** (Store register pair on stack)

– 1 byte instruction.
– Copies the contents of specified register pair or program status word (accumulator and flag) on the stack.
– Stack pointer is decremented and content of high order register is copied. Then it is again decremented and content of low order register is copied.

2. **POP Rp/PSW** (retrieve register pair from stack)
   – 1 byte instruction.
   – Copies the contents of the top two memory locations of the stack into specified register pair or program status word.
   – A content of memory location indicated by SP is copied into low order register and SP is incremented by 1. Then the content of next memory location is copied into high order register and SP is incremented by 1.
3. XTHL- exchange the top of the stack (TOS) with HL.



SPHL – move HL to SP

PCHL – move HL to PC

Some instructions related to interrupt

DI – disable interrupt

EI – Enable interrupt

SIM – set interrupt mask

RIM – read interrupt mask

E.g.

```
LXI SP, 1FFFH
LXI H, 9320H
LXI B, 4732H
LXI D, ABCDH
MVI A, 34H
PUSH H
PUSH B
PUSH D
PUSH PSW
```

```
POP H
POP B
POP D
POP PSW
HLT
```

BEORE EXECUTION

H= 93 L= 20
B= 47 C=32
D= AB E=CD
A= 34 F= 10

AFTER EXECUTION

H= 34 L=10
B=AB C=CD
D= 47 E=32
A= 93 F=20

Note: **STACK Works in LIFO (Last In First Out) manner.**

Question: What do you mean by stack and subroutine? What is the purpose of stack in subroutines call? Explain the concept of subroutines call and usage along with the changes in program execution sequence with a suitable example for 8085 microprocessor.WAP to sort in ascending order for 10 bytes from 1120H.

```
START: LXI H, 1120H
       MVI D, 00H
       MVI C, 09H
L2:    MOV A, M
       INX H
       CMP M
       JC L1  ; if A<M
       MOV B, M
       MOV M, A
       DCX H
       MOV M, B
       INX H
       MVI D, 01H
L1:    DCR C
       JNZ L2
       MOV  A, D
       RRC
       JC START
       HLT
```

Program to multiply two 8 bit binary numbers.

```
       LXI SP, 1999H
       LHLD 2050H; Two numbers
       XCHG;        Multiplier on D, Multiplicand on E
       CALL MULT
```

```
            SHLD 2090H; Store the product
            HLT
MULT:       MOV A, D
            MVI D, 00H
            LXI H, 0000H
            MVI B, 08H;  Counter
NEXT:       RAR;  Check multiplier bit is 1 or 0
            JNC BELOW
            DAD D;       Partial result in HL
BELOW:      XCHG
            DAD H;       Shift left multiplicand
            XCHG;        Retrieve shifted multiplicand
            DCR B
            JNZ NEXT
            RET
```

Note: Try Division yourself.

8085 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

What do you understand by addressing modes in microprocessor? Explain all the addressing modes of 8085 up with suitable example for each.

Programming Examples

1. WAP to store the data byte 32H into memory location 4000H.
   MVI A, 32H : Store 32H in the accumulator
   STA 4000H : Copy accumulator contents at address 4000H
   HLT : Terminate program execution
   Or
   LXI H : Load HL with 4000H
   MVI M : Store 32H in memory location pointed by HL register pair (4000H)
   HLT : Terminate program execution
2. Write the program to exchange the memory content 2000H and 4000H using direct addressing or indirect addressing**.**
   LDA 2000H : Get the contents of memory location 2000H into accumulator
   MOV B, A : Save the contents into B register
   LDA 4000H : Get the contents of memory location 4000Hinto accumulator
   STA 2000H : Store the contents of accumulator at address 2000H
   MOV A, B : Get the saved contents back into A register
   STA 4000H : Store the contents of accumulator at address 4000H
   or
   LXI H 2000H : Initialize HL register pair as a pointer to memory location 2000H.
   LXI D 4000H : Initialize DE register pair as a pointer to memory location 4000H.
   MOV B, M : Get the contents of memory location 2000H into B register.
   LDAX D : Get the contents of memory location 4000H into A register.
   MOV M, A : Store the contents of A register into memory location 2000H.
   MOV A, B : Copy the contents of B register into accumulator.

STAX D : Store the contents of A register into memory location 4000H.
HLT : Terminate program execution.

3. ***Add the contents of memory locations 4000H and 4001H and place the result in memory location 4002H.***

Sample problem
(4000H) = 14H
(4001H) = 89H
Result = 14H + 89H = 9DH

Source program
LXI H 4000H : HL points 4000H
MOV A, M : Get first operand
INX H : HL points 4001H
ADD M : Add second operand
INX H : HL points 4002H
MOV M, A : Store result at 4002H
HLT : Terminate program execution

Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.

Sample problem:
(4000H) = 51H
(4001H) = 19H
Result = 51H – 19H = 38H
Source program:
LXI H, 4000H : HL points 4000H
MOV A, M : Get first operand
INX H : HL points 4001H
SUB M : Subtract second operand
INX H : HL points 4002H
MOV M, A : Store result at 4002H.
HLT : Terminate program execution

4. *Add the 16-bit number in memory locations 4000H and 4001H to the 16-bit number in memory locations 4002H and 4003H. The most significant eight bits of the two numbers to be added are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H.*

Sample problem:
(4000H) = 15H
(4001H) = 1CH
(4002H) = B7H
(4003H) = 5AH

Result = 1C15 + 5AB7H = 76CCH
(4004H) = CCH
(4005H) = 76H

Source Program 1:
LHLD 4000H : Get first I6-bit number in HL
XCHG : Save first I6-bit number in DE
LHLD 4002H : Get second I6-bit number in HL
MOV A, E : Get lower byte of the first number
ADD L : Add lower byte of the second number
MOV L, A : Store result in L register
MOV A, D : Get higher byte of the first number
ADC H : Add higher byte of the second number with CARRY
MOV H, A : Store result in H register
SHLD 4004H : Store I6-bit result in memory locations 4004H and 4005H.
HLT : Terminate program execution

5. *Add the contents of memory locations 40001H and 4001H and place the result in the memory locations 4002Hand 4003H.*
Sample problem:
(4000H) = 7FH
(400lH) = 89H
Result = 7FH + 89H = lO8H
(4002H) = 08H
(4003H) = 0lH
Source program:
LXI H, 4000H :HL Points 4000H
MOV A, M :Get first operand
INX H :HL Points 4001H
ADD M :Add second operand
INX H :HL Points 4002H
MOV M, A :Store the lower byte of result at 4002H
MVIA, 00 :Initialize higher byte result with 00H
ADC A :Add carry in the high byte result
INX H :HL Points 4003H
MOV M, A :Store the higher byte of result at 4003H
HLT :Terminate program executio

6. *Subtract the 16-bit number in memory locations 4002H and 4003H from the 16-bit number in memory locations 4000H and 4001H. The most significant eight bits of the two numbers are in*

memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H.

Sample problem:

(4000H) = 19H

(400IH) = 6AH

(4004H) = I5H (4003H) = 5CH

Result = 6A19H – 5C15H = OE04H

(4004H) = 04H

(4005H) = OEH

Source program:

LHLD 4000H : Get first 16-bit number in HL

XCHG : Save first 16-bit number in DE

LHLD 4002H : Get second 16-bit number in HL

MOV A, E : Get lower byte of the first number

SUB L : Subtract lower byte of the second number

MOV L, A : Store the result in L register

MOV A, D : Get higher byte of the first number

SBB H : Subtract higher byte of second number with borrow

MOV H, A : Store l6-bit result in memory locations 4004H and 4005H.

SHLD 4004H : Store l6-bit result in memory locations 4004H and 4005H.

HLT : Terminate program execution.

7. *find the l's complement of the number stored at memory location 4400H and store the complemented number at memory location 4300H.*

Sample problem:

(4400H) = 55H

Result = (4300B) = AAB

Source program:

LDA 4400B : Get the number

CMA : Complement number

STA 4300H : Store the result

HLT : Terminate program execution

8. *Find the '2 complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.*

Sample problem:

(4200H) = 55H

Result = (4300H) = AAH + 1 = ABH

Source program:

LDA 4200H : Get the number

CMA : Complement the number

ADI, 01 H : Add one in the number

STA 4300H : Store the result
HLT : Terminate program execution
Sample problem:
(4200H) = 04
(4201H) = 09
Result = (4300H) = 94

9. *Pack the two unpacked BCD numbers stored in memory locations 4200H and 4201H and store result in memory location 4300H. Assume the least significant digit is stored at 4200H.*
Source program:
LDA 4201H : Get the Most significant BCD digit
RLC
RLC
RLC
RLC : Adjust the position of the second digit (09 is changed to 90)
ANI FOH : Make least significant BCD digit zero
MOV C, A : store the partial result
LDA 4200H : Get the lower BCD digit
ADD C : Add lower BCD digit
STA 4300H : Store the result
HLT : Terminate program execution

10. *Two digit BCD number is stored in memory location 4200H. Unpack the BCD number and store the two digits in memory locations 4300H and 4301H such that memory location 4300H will have lower BCD digit.*
Sample problem:
(4200H) = 58
Result = (4300H) = 08 and
(4301H) = 05
Source program:
LDA 4200H : Get the packed BCD number
ANI FOH : Mask lower nibble
RRC
RRC
RRC
RRC : Adjust higher BCD digit as a lower digit
STA 4301H : Store the partial result
LDA 4200H : .Get the original BCD number
ANI OFH : Mask higher nibble
STA 4201H : Store the result
HLT : Terminate program execution

11. *Calculate the sum of series of numbers. The length of the series is in memory location 4200H and the series begins from memory location 4201H.*
    a. Consider the sum to be 8 bit number. So, ignore carries. Store the sum at memory location 4300H.
    b. Consider the sum to be 16 bit number. Store the sum at memory locations 4300H and 4301H.

    Sample problem 1:
    4200H = 04H
    4201H = 10H
    4202H = 45H
    4203H = 33H
    4204H = 22H
    Result = 10 +41 + 30 + 12 = H
    4300H = H
    Source program 1:
    LDA 4200H
    MOV C, A : Initialize counter
    SUB A : sum = 0
    LXI H, 420lH : Initialize pointer
    BACK: ADD M : SUM = SUM + data
    INX H : increment pointer
    DCR C : Decrement counter
    JNZ BACK : if counter 0 repeat
    STA 4300H : Store sum
    HLT : Terminate program execution

    Sample problem 2:
    4200H = 04H
    420lH = 9AH
    4202H = 52H
    4203H = 89H
    4204H = 3EH
    Result = 9AH + 52H + 89H + 3EH = H
    4300H = B3H Lower byte
    4301H = 0lH Higher byte
    Source program 2
    LDA 4200H
    MOV C, A : Initialize counter
    LXI H, 4201H : Initialize pointer
    SUB A :Sum low = 0

MOV B, A : Sum high = 0
BACK: ADD M : Sum = sum + data
JNC SKIP
INR B : Add carry to MSB of SUM
SKIP: INX H : Increment pointer
DCR C : Decrement counter
JNZ BACK : Check if counter 0 repeat
STA 4300H : Store lower byte
MOV A, B
STA 4301H : Store higher byte
HLT :Terminate program execution

11. *Calculate the sum of series of odd numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 16-bit. Store the sum at memory locations 2300H and 2301H.*
sample problem 1:
2200H = 4H
2201H= 9AH
2202H= 52H
2203H= 89H
2204H= 3FH
Result = 89H + 3FH = C8H
2300H= H Lower byte
2301H = H Higher byte
Source program :

LDA 2200H
MOV C, A : Initialize counter
LXI H, 2201H : Initialize pointer
MVI E, 00 : Sum low = 0
MOV D, E : Sum high = 0
BACK: MOV A, M : Get the number
ANI 0lH : Mask Bit 1 to Bit7
JZ SKIP : Don't add if number is even
MOV A, E : Get the lower byte of sum
ADD M : Sum = sum + data
MOV E, A : Store result in E register
JNC SKIP
INR D : Add carry to MSB of SUM
SKIP: INX H : Increment pointe

12. *Find the square of the given numbers from memory location 6100H and store the result from memory location 7000H.*

Sample problem 1:
2200H = 4H
2201H= 9AH
2202H= 52H
2203H= 89H
2204H= 3FH
Result = 89H + 3FH = C8H
2300H= H Lower byte
2301H = H Higher byte
Source program :

```
LXI H, 6200H : Initialize lookup table pointer
LXI D, 6100H : Initialize source memory pointer
LXI B, 7000H : Initialize destination memory pointer
BACK: LDAX D : Get the number
MOV L, A : A point to the square
MOV A, M : Get the square
STAX B : Store the result at destination memory location
INX D : Increment source memory pointer
INX B : Increment destination memory pointer
MOV A, C
CPI 05H : Check for last number
JNZ BACK : If not repeat
HLT : Terminate program execution
```

13. *Search the given byte in the list of 50 numbers stored in the consecutive memory locations and store the address of memory location in the memory locations 2200H and 2201H. Assume byte is in the C register and starting address of the list is 2000H. If byte is not found store 00 at 2200H and 2201H.*

Source program :

```
LX I H, 2000H : Initialize memory pointer 52H
MVI B, 52H : Initialize counter
BACK: MOV A, M : Get the number
CMP C : Compare with the given byte
JZ LAST : Go last if match occurs
INX H  : Increment memory pointer
DCR B : Decrement counter
JNZ B : I f not zero, repeat
LXI H, 0000H
```

SHLD 2200H
JMP END : Store 00 at 2200H and 2201H
LAST: SHLD 2200H : Store memory address
END: HLT : Stop

14. *Two decimal numbers six digits each, are stored in BCD package form. Each number occupies a sequence of byte in the memory. The starting address of first number is 6000H Write an assembly language program that adds these two numbers and stores the sum in the same format starting from memory location 6200H.*

LXI H, 6000H : Initialize pointer l to first number
LXI D, 6l00H : Initialize pointer2 to second number
LXI B, 6200H : Initialize pointer3 to result
STC
CMC : Carry = 0
BACK: LDAX D : Get the digit
ADD M : Add two digits
DAA : Adjust for decimal
STAX.B : Store the result
INX H : Increment pointer 1
INX D : Increment pointer2
INX B : Increment result pointer
MOV A, L
CPI 06H : Check for last digit
JNZ BACK : If not last digit repeat
HLT : Terminate program execution

15. *Write an assembly language program to separate even numbers from the given list of 50 numbers and store them in the another list starting from 2300H. Assume starting address of 50 number list is 2200H.*

LXI H, 2200H : Initialize memory pointer l
LXI D, 2300H : Initialize memory pointer2
MVI C, 32H : Initialize counter
BACK:MOV A, M : Get the number
ANI 0lH : Check for even number
JNZ SKIP : If ODD, don't store
MOV A, M : Get the number
STAX D : Store the number in result list
INX D : Increment pointer 2
SKIP: INX H : Increment pointer l
DCR C : Decrement counter

JNZ BACK : If not zero, repeat
HLT : Stop

16. *Write assembly language program with proper comments for the following:*
A block of data consisting of 256 bytes is stored in memory starting at 3000H. This block is to be shifted (relocated) in memory from 3050H onwards. Do not shift the block or part of the block anywhere else in the memory.
Two blocks (3000 – 30FF and 3050 – 314F) are overlapping. Therefore it is necessary to transfer last byte first and first byte last.
Source Program:

MVI C, FFH : Initialize counter
LX I H, 30FFH : Initialize source memory pointer 3l4FH
LXI D, 314FH : Initialize destination memory pointer
BACK: MOV A, M : Get byte from source memory block
STAX D : Store byte in the destination memory block
DCX H : Decrement source memory pointer
DCX : Decrement destination memory pointer
DCR C : Decrement counter
JNZ BACK : If counter 0 repeat
HLT : Stop execution

17. *A list of 50 numbers is stored in memory, starting at 6000H. Find number of negative, zero and positive numbers from this list and store these results in memory locations 7000H, 7001H, and 7002H respectively.*

LXI H, 6000H : Initialize memory pointer
MVI C, 00H : Initialize number counter
MVI B, 00H : Initialize negative number counter
MVI E, 00H : Initialize zero number counter
BEGIN:MOV A, M : Get the number
CPI 00H : If number = 0
JZ ZERONUM : Goto zeronum
ANI 80H : If MSB of number = 1i.e. if
JNZ NEGNUM number is negative goto NEGNUM
INR D : otherwise increment positive number counter
JMP LAST
ZERONUM:INR E : Increment zero number counter
JMP LAST
NEGNUM:INR B : Increment negative number counter
LAST:INX H : Increment memory pointer
INR C : Increment number counter

MOV A, C
CPI 32H : If number counter = 5010 then
JNZ BEGIN : Store otherwise check next number
LXI H, 7000 : Initialize memory pointer.
MOV M, B : Store negative number.
INX H
MOV M, E : Store zero number.
INX H
MOV M, D : Store positive number.
HLT : Terminate execution

18. *Write an assembly language program to generate fibonacci number.*

MVI D, COUNT : Initialize counter
MVI B, 00 : Initialize variable to store previous number
MVI C, 01 : Initialize variable to store current number
MOV A, B :[Add two numbers]
BACK: ADD C :[Add two numbers]
MOV B, C : Current number is now previous number
MOV C, A : Save result as a new current number
DCR D : Decrement count
JNZ BACK : if count 0 go to BACK
HLT : Stop.

19. *Arrange an array of 8 bit unsigned no in descending order*

START:MVI B, 00 ; Flag = 0
LXI H, 4150 ; Count = length of array
MOV C, M
DCR C ; No. of pair = count -1
INX H ; Point to start of array
LOOP:MOV A, M ; Get kth element
INX H
CMP M ; Compare to (K+1) th element
JNC LOOP 1 ; No interchange if kth >= (k+1) th
MOV D, M ; Interchange if out of order
MOV M, A ;
DCR H
MOV M, D
INX H
MVI B, 01H ; Flag=1
LOOP 1:DCR C ; count down

```
JNZ LOOP ;
DCR B ; is flag = 1?
JZ START ; do another sort, if yes
HLT ; If flag = 0, step execution
```

20. *Program to calculate the factorial of a number between 0 to 8*

```
LXI SP, 27FFH ; Initialize stack pointer
LDA 2200H ; Get the number
CPI 02H ; Check if number is greater than 1
JC LAST
MVI D, 00H ; Load number as a result
MOV E, A
DCR A
MOV C,A ; Load counter one less than number
CALL FACTO ; Call subroutine FACTO
XCHG ; Get the result in HL
SHLD 2201H ; Store result in the memory
JMP END
LAST: LXI H, 000lH ; Store result = 01
END: SHLD 2201H
HLT
```

Subroutine Program:

```
FACTO:LXI H, 0000H
MOV B, C ; Load counter
BACK: DAD D
DCR B
JNZ BACK ; Multiply by successive addition
XCHG ; Store result in DE
DCR C ; Decrement counter
CNZ FACTO ; Call subroutine FACTO
RET ; Return to main program
```

21. *Write a program to find the Square Root of an 8 bit binary number. The binary number is stored in memory location 4200H and store the square root in 4201H.*

Source program :

```
LDA 4200H : Get the given data(Y) in A register
MOV B,A : Save the data in B register
MVI C,02H : Call the divisor(02H) in C register
CALL DIV : Call division subroutine to get initial value(X) in D-reg
```

REP: MOV E,D : Save the initial value in E-reg
MOV A,B : Get the dividend(Y) in A-reg
MOV C,D : Get the divisor(X) in C-reg
CALL DIV : Call division subroutine to get initial value(Y/X) in D-reg
MOV A, D : Move Y/X in A-reg
ADD E : Get the((Y/X) + X) in A-reg
MVI C, 02H : Get the divisor(02H) in C-reg
CALL DIV : Call division subroutine to get ((Y/X) + X)/2 in D-reg.This is XNEW
MOV A, E : Get Xin A-reg
CMP D : Compare X and XNEW
JNZ REP : If XNEW is not equal to X, then repeat
STA 4201H : Save the square root in memory
HLT : Terminate program execution

Chapter- 4

INSTRUCTION CYCLE

INTRODUCTION

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 upcode from the memory . The necessary steps which are carried out to fetch an upcode 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 it's upcode . 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 upcode is available , is send to the memory. The memory places the upcode on the data bus so as to transfer it to CPU. The entire process takes 3 clock cycle**.**

Execute cycle/Operation: The upcode 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.**

Timing diagram: The necessary steps which are carried in a machine cycle can represented graphically. Such graphical representation is called timing diagram.

Opcode Fetch :A microprocessor either reads or writes to the memory or I/O devices. The time taken to read or write for any instruction must be known in terms of the μP clock. The 1st step in communicating between the microprocessor and memory is reading from the memory. This reading process is called opcode fetch. The process of opcode fetch operation requires minimum 4- clock cycles T1, T2, T3, and T4 and is the 1st machine cycle (M1) of every instruction. In order to differentiate between the data byte pertaining to an opcode or an address, the machine cycle takes help of the status signal IO/M, S1, and S0. The IO/M = 0 indicates memory operation and S1 = S0 = 1 indicates Opcode fetch operation. The opcode fetch machine cycle M1 consists of 4-states (T1, T2, T3, and T4). The 1st 3-states are used for fetching (transferring) the byte from the memory and the 4th-state is used to decode it. Thus, thorough understanding about the communication between memory and microprocessor can be achieved only after knowing the processes involved in reading or writing into the memory by the microprocessor and time taken w.r.t. its clock period. This can be explained by examples. The process of implementation of each instruction follows the fetch and execute cycles. In other words, first the instruction is fetched from memory and then executed. Figure 3 and 4 depict these 2-steps for implementation of the instruction ADI 05H. Let us assume that the accumulator contains the result of previous operation.

BUS STRUCTURE:

A microcomputer consists of a set of components or modules of three basic types CPU memory and I/O units which communicate with each other. A bus is a communication pathway between two or more such components. A bus actually consists of multiple communication pathway or lines. Each line is capable of transmitting signals representing binary 1 and 0. Several lines of the bus can be used to transmit binary data simultaneously. The bus that connects major microcomputer components such as CPU, memory or I/O is called the system bus. System bus consists of number of separate lines. Each line assigned a particular function. Fundamentally in any system bus the lines can be classified into three group buses.

1. Data Bus:

Data bus provides the path for monitoring data between the system modules. The bus has various numbers of separate lines like 8, 16, 32, or 64. Which referred as the width of data bus .These number represents the no. of bits they can carry because each carry 1 bit.

2. Address Bus:

Each Lines of address bus are used to designate the source or destination of the data on data bus. For example, if the CPU requires reading a word (8, 16, 32) bits of data from memory, it puts the address of desired word on address bus. The address bus is also used to address I/O ports. Bus width determines the total memory the up can handle.

3. Control Bus:

The control bus is a group of lines used to control the access to control signals and the use of the data and

address bus. The control signals transmit both command and timing information between the system modules. The timing signals indicate the validity of data and address information, where as command signals specify operations to be performed. Some of the control signals are:

Memory Write(MEMW): It causes data on the bus to be loaded in to the address location.
Memory Read (MEMR): It causes data from the addressed location to be placed on the data bus.
I/O Write (IOW): It causes the data on the bus to be output to the addressed I/O port.
I/O Read (IOR): It causes the data from the addressed I/O port to be placed on the bus.
Transfer Acknowledge: This signal indicates that data have been accepted from or placed on the bus.
Bus Request: It is used to indicate that a module wants to gain control of the bus.
Bus Grant: It indicates that a requesting module has been granted for the control of bus.
Interrupt Request: It indicates that an interrupt has been pending.
Interrupt Acknowledge: it indicates that the pending interrupt has been recognized.

Bus Types

1. Synchronous Bus:

In a synchronous bus, The Occurrence of the events on the bus is determined by a clock . The clock transmits a regular sequence of 0‘s & 1‘s of equal duration. All the events start at beginning of the clock cycle.

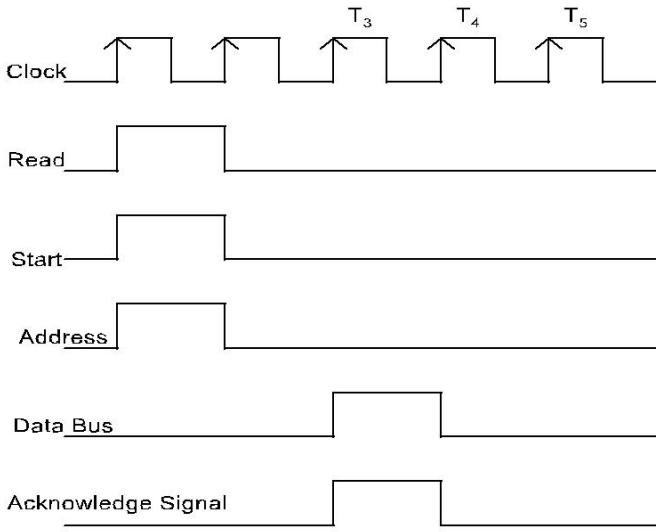


Fig: Synchronous Read Operation

Here the CPU issues a START signal to indicate the presence of address and control information on the bus.

- Then it issues the memory read signal and places the memory address on the address bus.
- The addressed memory module recognizes the address and after a delay of one clock cycle it places the data and acknowledgment signal on the buses.

In synchronous bus, all devices are tied to a fixed rate, and hence the system can not take advantage of device performance but it is easy to implement.

2. Asynchronous Bus:

In an asynchronous bus, the timing is maintained in such way that occurrence of one event on the bus follows and depends on the occurrence of previous event.

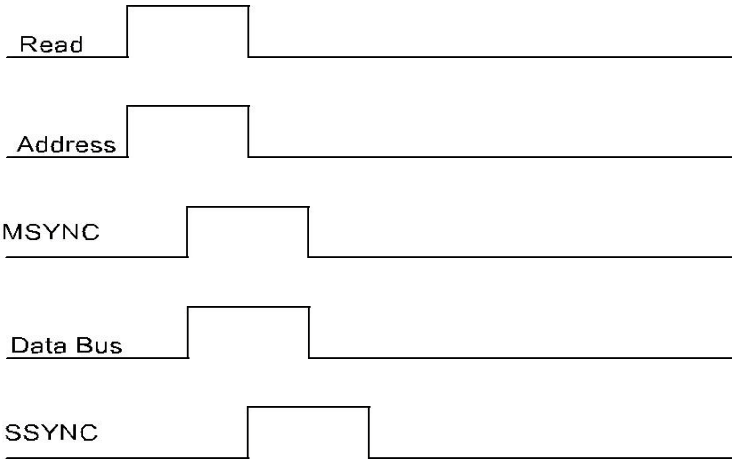


Fig: Asynchronous Read Operation

- Here the CPU places Memory Read (Control) and address signals on the bus.
- Then it issues master synchronous signal (MSYNC) to indicate the presence of valid address and control signals on the bus.
- The addressed memory module responds with the data and the slave synchronous signal (SSYNC)

Machine cycles and bus timing diagrams:

Operation of a microprocessor can be classified in to following four groups according to their nature.

- Op- Code fetch
- Memory Read /Write
- I/O Read/ Write
- Request acknowledgement

  Here Op-Code fetch is an internal operation and other three are external operations. During three operations, microprocessor generates and receives different signals. These all operations are terms as machine cycle.
- Clock Cycle (T state): It is defined as one subdivision of the operation performed in one clock period.
- Machine Cycle: It is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request. This cycle may consist of three to six T-states.

Op-Code fetch Machine Cycle

The first operation in any instruction is Op-Code fetch, The microprocessor needs to get(fetch) this machine code from the memory register where it is stored before the microprocessor can begin to execute the instruction.

Let‘s consider the instruction MOV C, A stored at memory location 2005H. The Op-Code for the instruction is 4FH and Op-Code fetch cycle is of 4 clock cycles.

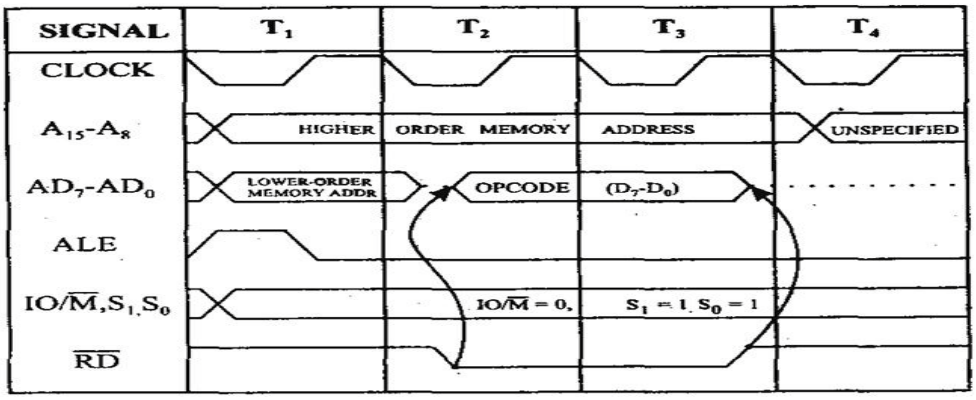


Fig - Timing Diagram for Opcode Fetch Machine Cycle

Step1: Microprocessor places the 16 bit memory address from Program Counter on the address bus. At T_1, high order address (20) is placed at A_8-A_{15} and lower order address (05) is placed at AD_0- AD_7 ALE signal goes high. IO/M goes low and both S_0 and S_1 goes high for Op-Code fetch.

Step 2: The control unit sends the control signal RD to enable the memory chip and active during $T_2$and T_3.

Step 3: The byte from the memory location is placed on the data bus .that is 4f into D_0-D_7andRD goes high impedance.

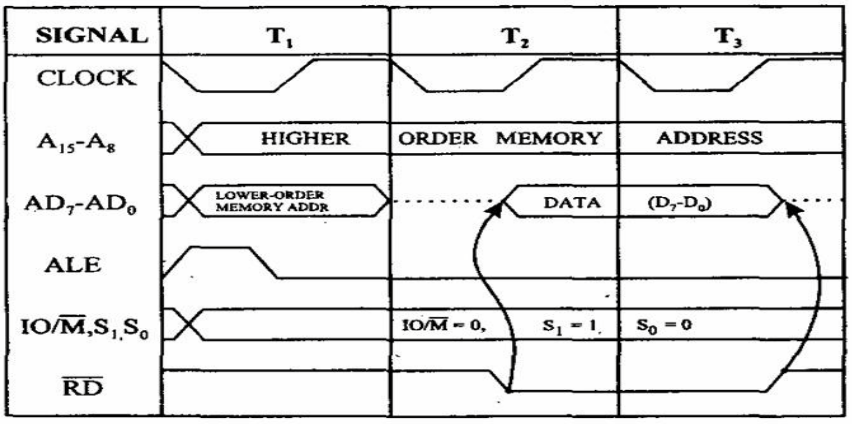
Step4: The instruction 4FH is decoded and content of accumulator will be copied into register Cduring clock cycle T_4.

Memory Read Cycle:

Let's consider the instruction MVI A, 32 H stored at memory location 2000H.

2000	3EH	MVI	A,	32	H
2001	32H				

Here two machine cycles are presented, first is Op-Code fetch which consists of 4 clock cycles and second is memory read consist of 4 clock cycle.



Step 1 : First machine cycle (Op-Code fetch) is identical for timing diagram of Op-Code fetch cycle.

Step 2: After completion of Op-Code fetch cycle, 8085 places the address 2001 on the address bus and increments PC to 2002H. ALE is asserted high IO/M =0, S_1=1, S $_0$=0 for memory read cycle. When RD =0, memory places the data byte 32H on the dada bus.

Memory Write Machine:

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes, 3T states to execute this machine cycle.

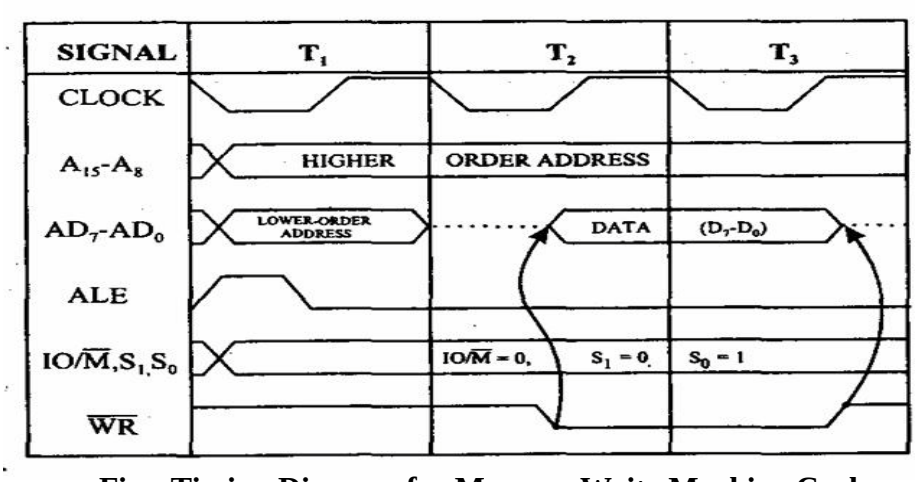


Fig - Timing Diagram for Memory Write Machine Cycle

I/O Read Cycle:

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution

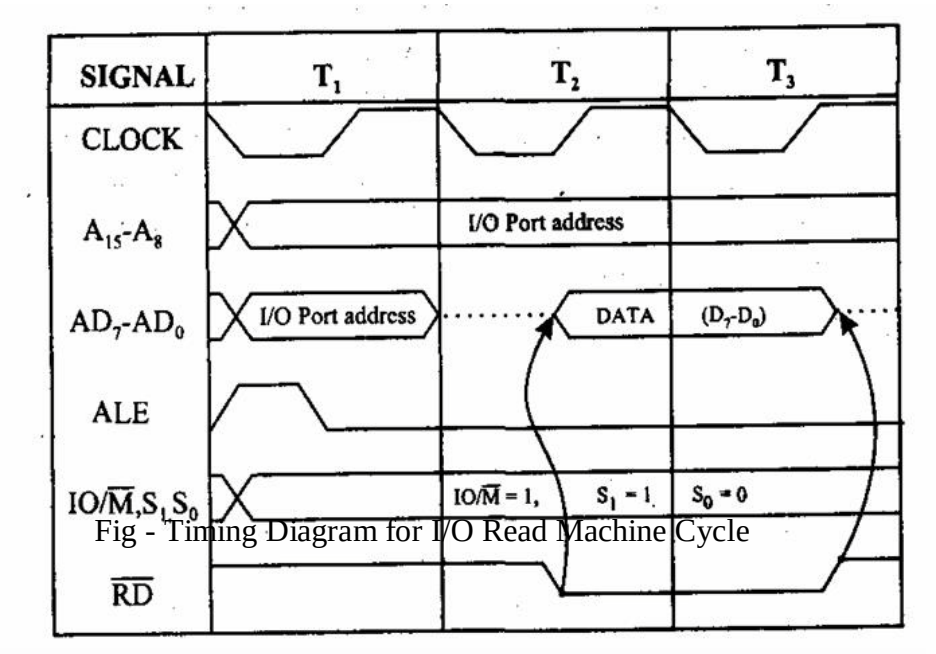


Fig - Timing Diagram for I/O Read Machine Cycle

I/O Write Cycle:

Let‘s consider the instruction OUT 01H stored at memory location 2050H.

2050 D3 OUT 01H

2051 01

Op-Code Fetch Cycle 4T
Memory read Cycle 3T
I/O Write Cycle 3T

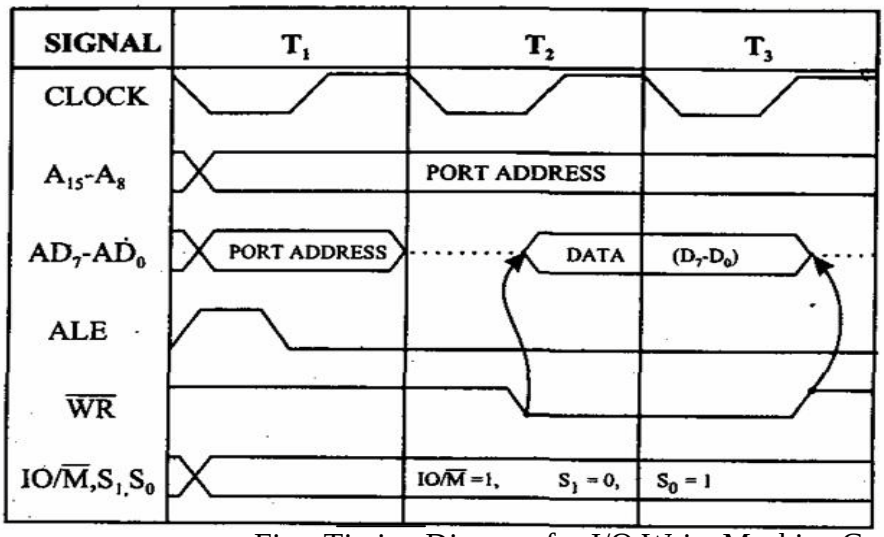


Fig - Timing Diagram for I/O Write Machine Cycle

Step 1: In Machine Cycle M_1, the microprocessor sends RD control signal which is combinedwith IO/ M to generate the MEMR signal and processor fetches instruction code D3 using the data bus.
Step 2: In 2nd Machine Cycle M_2, the 8085 microprocessor places the next address 2051 on theaddress bus and gets the device address 01H via data bus.
Step 3: In machine Cycle M_3, the 8085 places device address 01H on low order as well as highorder address bus. IO/ M goes high for IO and accumulator content are placed on Data bus which are to be written into the selected output port.

Timing diagram for 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	Mnemonics	Opcode
41FF	STA 526AH	32_H
4200		$6A_H$
4201		52_H

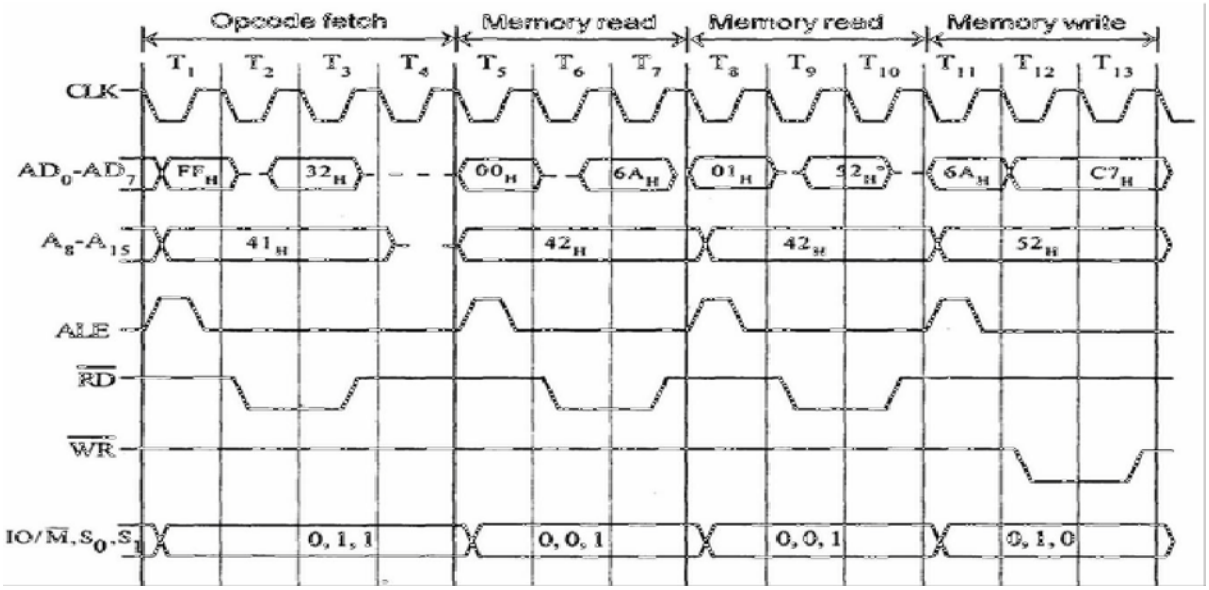


Fig. Timing diagram for STA 526AH

Timing diagram for IN C0H.

- 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	Opcode
4125	IN C0H	DB_H
4126		$C0_H$

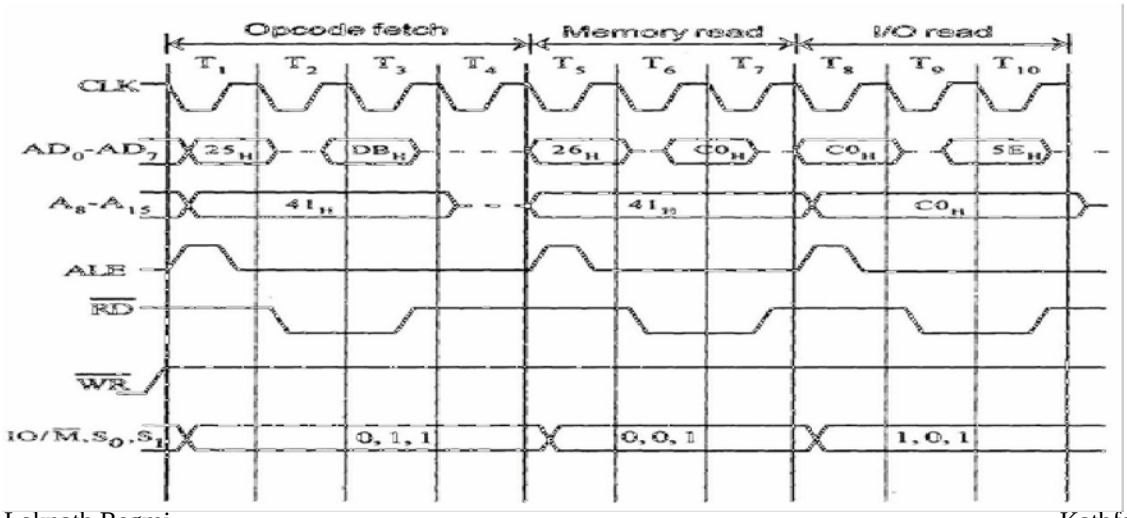
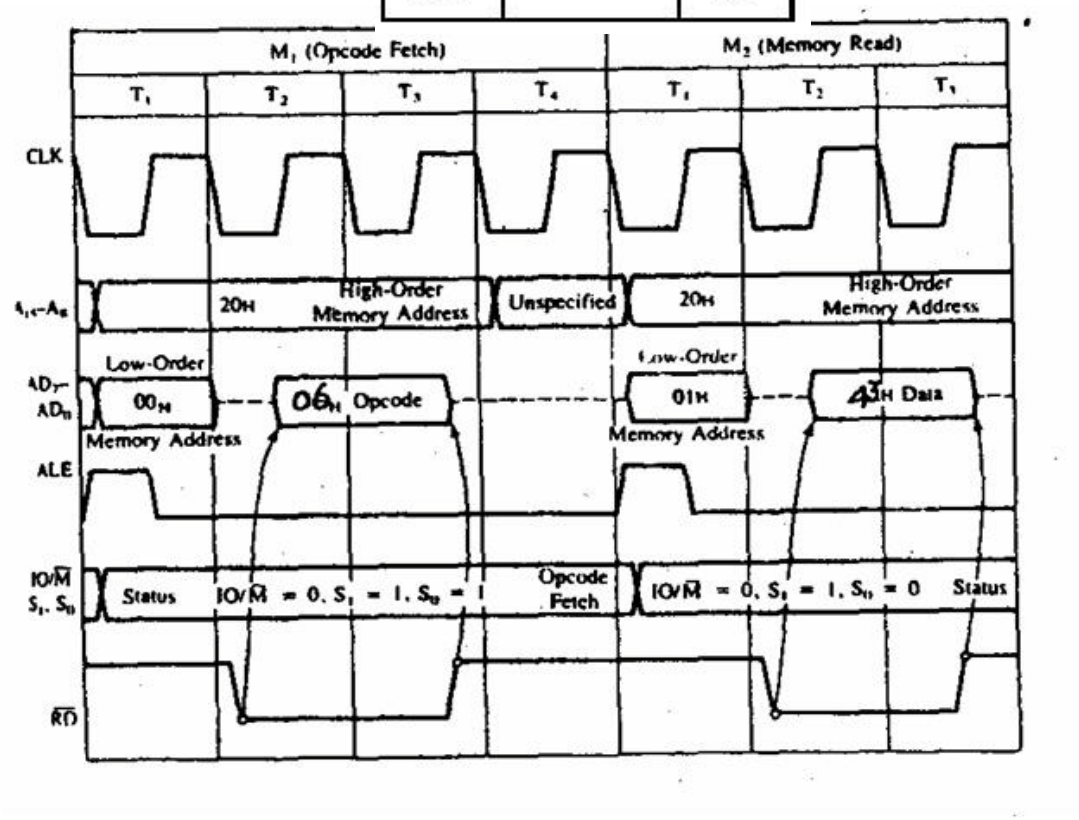


Fig.Timing diagram for IN C0H.

Timing diagram for MVI B, 43H.

- Fetching the Opcode 06H from the memory 2000H. (OF machine cycle)
- Read (move) the data 43H from memory 2001H. (memory read)

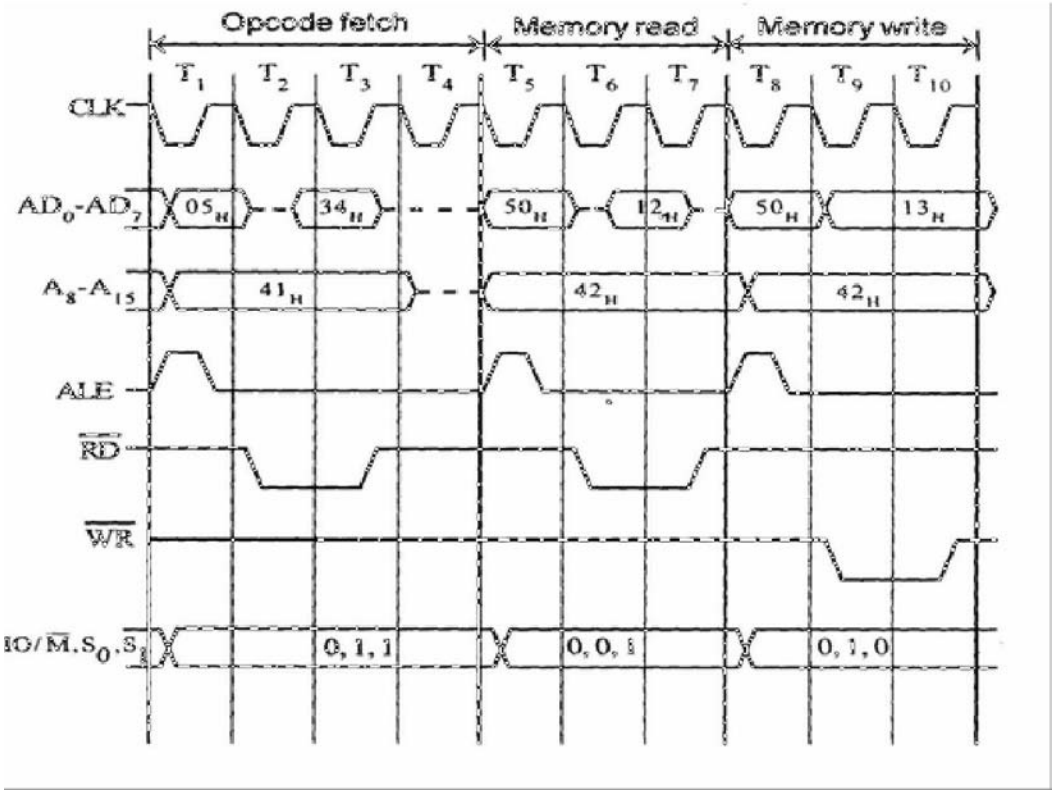
Address	Mnemonics	Opcode
2000	MVI B, 43H	06H
2001		43H



Timing diagram for INR M

- Fetching the Opcode 34H from the memory 4105H. (OF cycle)
- Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)
- Let the content of that memory is 12H.
- Increment the memory content from 12H to 13H. (MW machine cycle)

Address	Mnemonics	Opcode
4105	INR M	34_H



Chapter -5

INTEL MICROPROCESSORS 8086

SALIENT FEATURES:

8086 is the first 16-bit microprocessor from INTEL, released in the year 1978 . It is a 40 pin DIP chip based on N-channel, depletion load silicon gate technology(HMOS). The term 16 bit means that its ALU,its internal registers and most of the instructions are designed to work with 16 bit binary words.8086 is available at different clock speeds Viz, 5 M.Hz(8086);8M.Hz(8086-2) and 10(8086-1) M.Hz .8086 microprocessor has a 16-bit data bus and 20-bit address bus. So, it can address any one of 2^{20} =1048576=1 mega byte memory locations. INTEL 8088 has the same ALU ,same registers and same instruction set as the 8086.But the only difference is 8088 has only 8-bit data bus and 20-bit address bus. Hence the 8088 can only read/write/ports of only 8-bit data at a time .The 8088 was used as the CPU in the original IBM personal computers [IBMPC/XT] .The 8086 microprocessor can work in two modes of operations .They are Minimum mode and Maximum mode. In the minimum mode of operation the microprocessor do not associate with any co-processors and can not be used for multiprocessor systems. But in the maximum mode the 8086 can work in multi-processor or co-processor configuration. This minimum or maximum operations are decided by the pin MN/ MX(Active low). When this pin is high 8086 operates in minimum mode otherwise it operates in Maximium mode.

Differences between 8086 and 8088 Microprocessors :

Though the architecture and instruction set of both 8086 and 8088 processors are same, still we find certain differences between them They are

(i) 8086 has 16-bit data bus lines whereas 8088 has 8-data lines.

(ii) 8086 is available in three clock speds namely 5 M.Hz,8M.Hz(8086-2) and 10 M.Hz

(8086-1) whereas 8088 is available is only available only in two speeds namely 5M.Hz and 8M.Hz

(iii) The memory address space of 8086 is organized as two 512kB banks whereas 8088 memory space is implemented as a single 1MX 8 memory bank.

(iv) 8086 has a 6-byte instruction queue whereas 8088 has a 4 byte instruction queue . The reason for this is that 8088 can fetch only one byte at a time.

(v) In 8086 the memory control pin (M/ IO) signal is complement of the 8088 equivalent signal(IO/M)

(vi) The 8086 has BHE(Bank high enable whereas 8088 has SSO status signal .

(vii).The byte and word data operations of 8086 are different from 8088.

(viii) 8086 can read or write either 8-bit or 16-bit word at a time ,whereas 8088 can read only 8-bit data at a time.

(ix) The I/O voltage levels for 8086 are , Vol is measured at 2.5mA and for 8088 it is measured at 2.0mA.

(x) 8086 draws a maximum supply current of 360 mA and the 8088 draws a maximum of 340 mA.

ARCHTECTURE OF 8086/8088 :

To improve the performance by implementing the parallel processing concept the CPU of the 8086 /8088 is divided into two independent sections .They are Bus Interface Unit (BIU) and Execution Unit.(EI).The BIU sendsout addresses ,fetches instructions ,read data from ports and memory and writes data to ports and memory.i.e the BIU handles all transfers data and addresses on the buses required by the execution Unit . Whereas the Execution Unit decodes the instructions and executes the instructions

The Execution Unit : The Execution Unit consists of a control system , a 16-bit ALU, 16-bit Flag register and four general purpose registers(AX,BX,CX,DX), pointer registers (SP,BP) and Index registers(SI,DI) of each 16-bits .

The control circuitry controls the internal operations .The decoder in the execution unit decodes the instructions fetched from the memory into a series of actions. The ALU can add ,subtract, perform operations like logical AND,OR,XOR, increment, decrement, complement ,and shifting the binary numbers.

Bus Interface Unit : The BIU consists of a 6-byte long instruction register called Queue.

And four stack segment registers (ES,CS,SS,DS) , one Instruction Pointer(IP) and an adder circuit to calculate the 20bit physical address of a location. This bus interface unit will perform all the external bus operations. They are fetching the instructions from the memory, read/write data from/into memory or port and also supporting the instruction Queue etc. The BIU fetches up to six instruction bytes from the memory and stores these pre-fetched bytes in a first –in first out register set called Queue. When the execution unit is ready for the execution of the instruction ,instead of fetching the byte from the memory ,it reads the byte from the Queue .This will increase the overall speed of microprocessor .Fetching the next instruction while the current instruction executes is called pipelining or parallel processing.

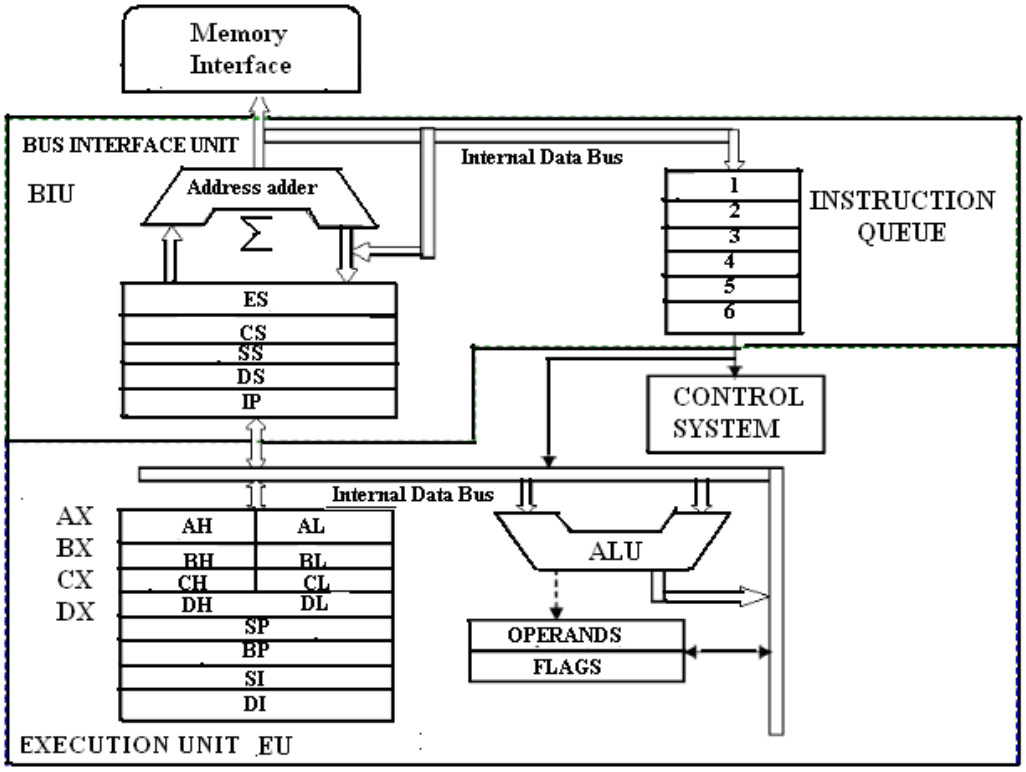


Fig.2.Architecture of 8086 Microprocessor

REGISTER ORGANISATION :

The 14 registers of 8086 microprocessor are categorized into four groups. They are general purpose data registers , Pointer & Index registers , Segment registers and Flag register as shown in the table below.

S.No	Type	Register width	Name of the Registers
1	General purpose Registers(4)	16-bit	AX,BX,CX,DX
		8-bit	AL,AH,BL,BH,CL,CH,DL,DH
2	Pointer Registers	16-bit	Stack Pointer(SP) Base Pointer(BP)

3	Index Registers	16-bit	Source Index(SI) Destination Index(DI)
4	Segment Registers	16-bit	Code Segment(CS) Data Segment(DS) Stack Segment(SS) Extra Segment(ES)
5	Instruction	16-bit	Instruction Pointer (IP)
6	Flag (PSW)	16-bit	Flag Register

8086 Microprocessor Registers.

General purpose registers: There are four 16-bit 4 general purpose registers namely (AH, AL);(BH,BL); (CH,CL); (and DH,DL) which are part of Execution unit. These registers can be used individually for storing 16-bit data temporarily .The AL register is also called the accumulator. The pairs of registers can be used together to store 16-bit data words.

It is always advantageous to store the data in these registers because the data can be accessed much more easily as these registers are already in the execution unit. Here L indicates the lower byte and H indicates the higher byte. X indicates the extended register. The general purpose data registers are used for data manipulations. The use of these registers is more dependent on the mode of addressing also.

.

The other four registers of EU are referred to as index / pointer registers. They are Stack Pointer register , Base Pointer register, Source Index register and Destination Index registers. The pointer registers contain the offset within a particular segment.

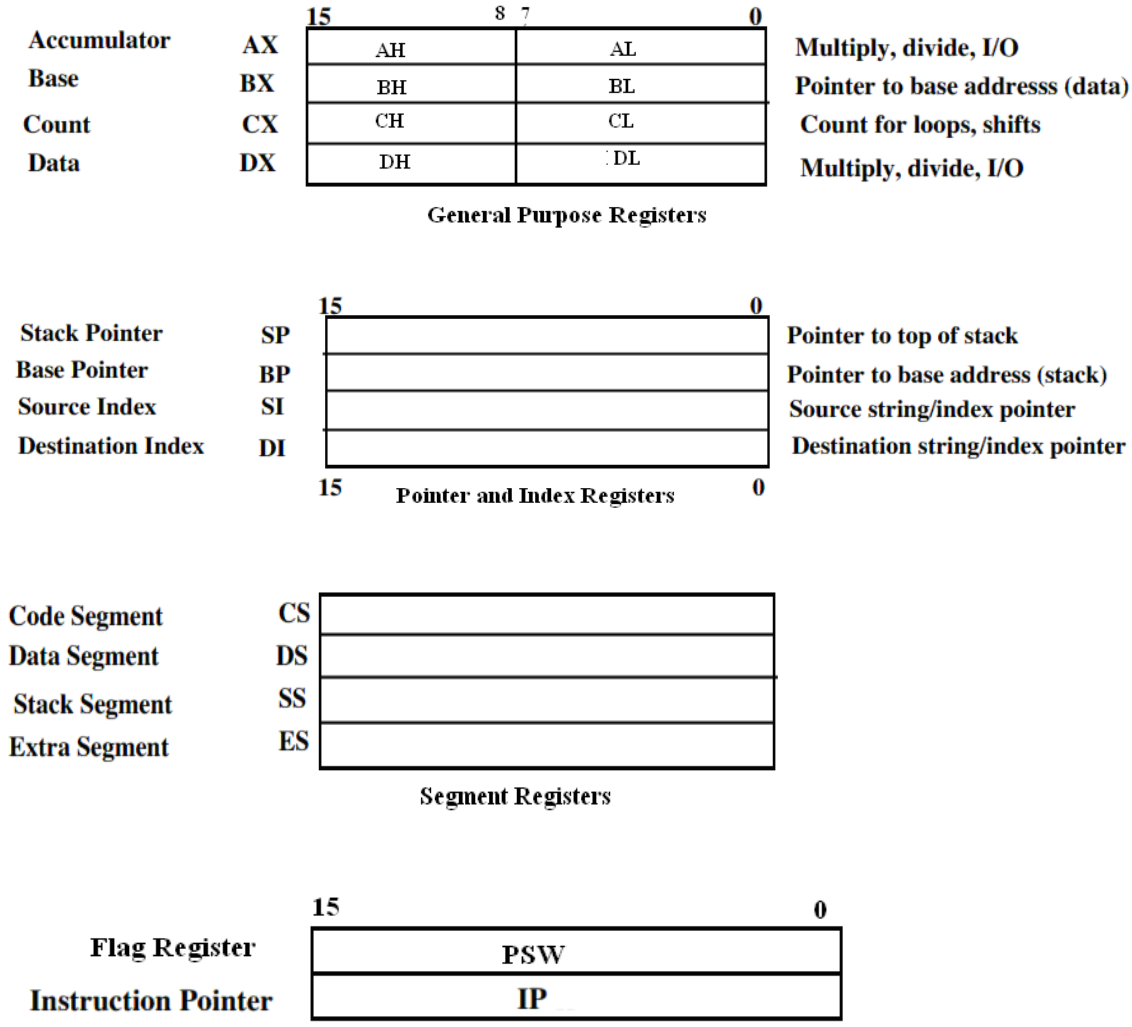


Fig 3. Register Organisation

The BP & SP registers holds the offsets within the data and stack segments respectively. The Index registers are used as general purpose registers as well as for holding the offset in case of indexed based and relative indexed addressing modes.The source Index register is generally used to store the offset of source data in data segment while the Destination Index register used to store the offset of destination in data or extra segment. These index registers are specifically used in string manipulations.

Segment Registers : There are four 16-bit segment registersnamely code segment register(CS),Stack segment register(SS),Data segment register(DS) and Extra segment register(ES).The code segment register is used for addressing the 64kB memory location in the code segment of the memory ,where the code of

the executable program is stored. Similarly the DS register points to the data segment of the 64kB memory where the data is stored. The Extra segment register also refers to essentially another data segment of the memory space. The SS register is useful for addressing stack segment of memory. So, the CS,DS,SS and ES segment registers respectively contains the segment addresses for the code, data, stack and extra segments of the memory.

Instruction Pointer Register: It is a 16-bit register which always points to the next instruction to be executed within the currently executing code segment. So, this register contains the 16-bit offset address pointing to the next instruction code within the 64kB of the code segment area. Its content is automatically incremented as the execution of the next instruction takes place.

Flag Register: This register is also called status register. It is a 16 bit register which contains six status flags and three control flags. So, only nine bits of the 16 bit register are defined and the remaining seven bits are undefined. Normally this status flag bits indicate the status of the ALU after the arithmetic or logical operations. Each bit of the status register is a flip/flop. The Flag register contains Carry flag, Parity flag, Auxiliary flag Zero flag, Sign flag ,Trap flag, Interrupt flag, Direction flag and overflow flag as shown in the diagram. The CF,PF,AF,ZF,SF,OF are the status flags and the TF,IF and CF are the control flags.

X	X	X	X	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**X**	**AF**	**X**	**PF**	**X**	**CF**

Flag Register

.**CF-** Carry Flag: This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

PF - Parity Flag : This flag is set to 1, if the lower byte of the result contains even number of 1's else (for odd number of 1s) set to zero.

AF-Auxilary Carry Flag: This is set, if there is a carry from the lowest nibble, i.e, bit three during addition, or borrow for the lowest nibble, i.e, bit three, during subtraction.

ZF- Zero Flag: This flag is set, if the result of the computation or comparison performed by the previous instruction is zero

SF- Sign Flag : This flag is set, when the result of any computation is negative

TF - Tarp Flag: If this flag is set, the processor enters the single step execution mode.

IF- Interrupt Flag: If this flag is set, the maskable interrupt INTR of 8086 is enabled and if it is zero ,the interrupt is disabled.It can be set by using the STI instruction and can be cleared by executing CLI instruction.

DF- Direction Flag: This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto incrementing mode.

OF- Over flow Flag: This flag is set, if an overflow occurs, i.e, if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, then the overflow will be set.

8086 PIN DIAGRAM – PIN DESCRIPTION

Intel 8086 is a 16-bit HMOS microprocessor. It is available in 40 pin DIP chip. It uses a 5V d.c. supply for its operation. The 8086 uses 20-line address bus. It uses a 16-line data bus. The 20 lines of the address bus operate in multiplexed mode. The 16-low order address bus lines are multiplexed with data and 4 high-order address bus lines are multiplexed with status signals. The pin diagram of Intel 8086 is shown in Fig.4.

AD_0-AD_{15} (Bidirectional): Address/Data bus. These are low order address bus. They are multiplexed with data. When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A0-A15. When data are transmitted over AD lines the symbol D is used in place of AD, for example D_0-D_7, D_8-D_{15} or D_0-D_{15}.

A_{16}-A_{19} (Output): High order address bus. These are multiplexed with status signals.

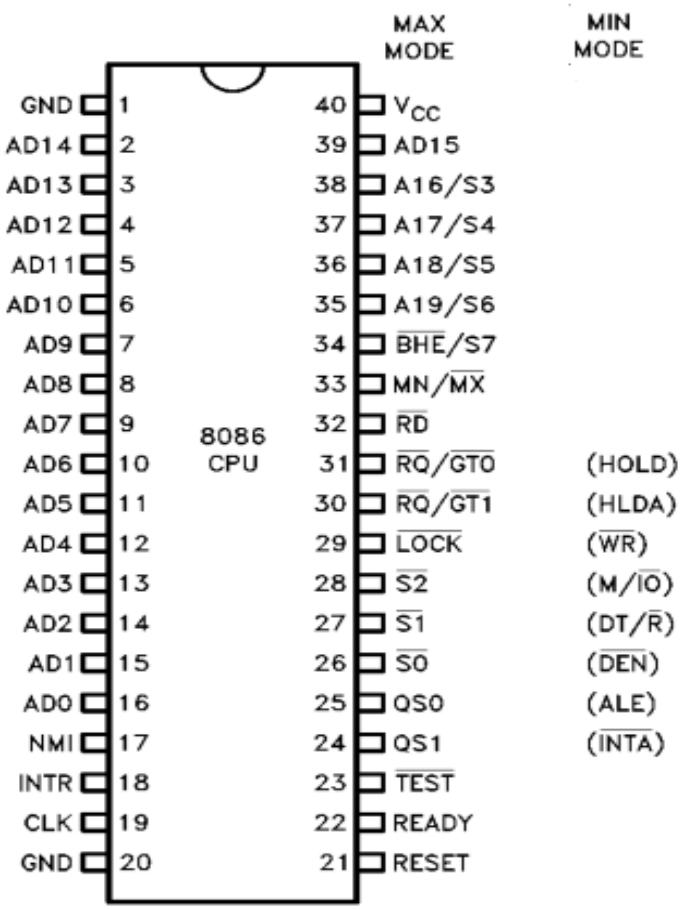


Fig.4. Pin Diagram of 8086 Processor

A_{16}/S_3, A_{17}/S_4, A_{18}/S_5, A_{19}/S_6: The specified address lines are multiplexed with corresponding status signals.

BHE (Active Low)/S_7 (Output) **:** Bus High Enable/Status. During T1 it is low. It is used to enable data onto the most significant half of data bus, D8-D15. 8-bit device connected to upper half of the data bus use BHE (Active Low) signal. It is multiplexed with status signal S7. S7 signal is available during T2, T3 and T4.

RD (Read) (Active Low) **:** The signal is used for read operation. It is an output signal. It is active when low.

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

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

NMI (Input) –NON-MASKABLE INTERRUPT : It is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.

INTA: INTA: Interrupt acknowledge. It is active LOW during T_2 ,T_3 and T_w of each interrupt acknowledge cycle.

MN/ MX MINIMUM / MAXIMUM :This pin signal indicates what mode the processor is to operate in.

RQ/GT RQ/GT0 : REQUEST/GRANT: These pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with RQ/GT having higher priority than RQ /GT_1.

LOCK: Its an active low pin. It indicates that other system bus masters are not to allowed to gain control of the system bus while LOCK is active LOW. The LOCK signal remains active until the completion of the next instruction.

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

CLK- Clock Input : The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.

RESET (Input) :RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles.

Vcc– Power Supply (+5V D.C.)

GND – Ground

QS_1,QS_0 (Queue Status) These signals indicate the status of the internal 8086 instruction queue according to the table shown below

QS_I	QS_0	Status
0 (LOW)	0	No Operation
0	1	First Byte of Op Code from Queue
1 (HIGH)	0	Empty the Queue
1	1	Subsequent Byte from Queue

DT/R : DATA TRANSMIT/RECEIVE: This pin is needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver.

DEN: DATA ENABLE .This pin is provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. DEN is active LOW during each memory and I/O access and for INTA cycles.

HOLD/HOLDA : HOLD indicates that another master is requesting a local bus .This is an active HIGH. The processor receiving the ``hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T_4 or T_1 clock cycle.

MEMORY ORGANIZATION :

The 8086 processor provides a 20-bit address toaccess any location of the 1 MB memory space. The memory is organized as a linear array of 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each . Physically, the memory is organized as a high bank (D15 - D8) and a low bank (D7 –D0) of 512 K 8-bitbytes addressed in parallel by the processor's address lines A19 -A1. Byte data with even addresses is transferred on the D7 – D0 bus lines while odd addressed byte data (A0 HIGH) is transferred on the D15-D8 bus lines. The processor provides two enable signals, BHE and A0 , to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

INTERRUPTS :

An interrupt to the microprocessor is defined as that which disturbs the normal execution of a program . Broadly the interrupts are divided into two types. They are external hardware Interrupts and internal (Software) Interrupts . The hardware interrupts are classified as non-maskable and maskable interrupts. The hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor. Whereas internal interrupts are initiated by the state of the CPU (e.g. divide by zero error) or by an instruction. So, the software interrupt is one which interrupts the normal execution of a program of the microprocessor. The 8086 has two hardware interrupt pins namely NMI and INTR.In the two ,the NMI is a non-maskable interrupt and the INTR interrupt request is a maskable interrupt which has lower proirity .The third pin associated with the hardware interrupts are the INTA called interrupt acknowledge.

NMI : The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt.

INTR: The 8086 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable FLAG status bit. The interrupt request signal is level triggered. It is internally synchronized during each clock

cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block type instruction.

Software Interrupts: Coming to the software interrupts , 8086 can generate 256 interrupt types through the instruction INT n .Any of the 256 interrupt types can be generated by specifying the interrupt type after INT instruction . For example INT 33 will cause type 33 interrupt.

I k Bytes of memory from 00000H to 003FF H is set aside to store the starting address of the Interrupt service sub-routine(ISS) programs in an 8086 based systems. To store the starting address of the each ISS , four bytes of memory space is required.Two bytes are for storing CS value and two bytes for IP value. The starting address of an ISS stored in 1kB of memory space is called Interrupt pointer or Interrupt vector.The 1kB memory space acts as a table and it is called Interrupt Vector Table(IVT).

INT Number (Type)	Physical Address	Contains
INT 00	00000 H	IP0:CS0
INT 01	00004 H	IP1:CS1
INT 02	00008H	IP2:CS2
. . .	. . .	. . .
INT FF	003FC H	IP255:CS255

The 256 interrupt pointers have been numbered from 0 to 255.The number given to an interrupt pointer denotes the type of the interrupt. For example Type0,Type1,Type2 etc…The starting address of the ISS for type0 interrupt is 000000H.For type1 interrupt is 00004H similarly for type2 is 00008H ……In the IVT the first five pointers are dedicated interrupt pointers. They are :

TYPE 0 Interrupt corresponds to divide by zero situation

TYPE 1 Interrupt corresponds to Single step execution during the debugging of a program.

TYPE 2 Interrupts to non-maskable NMI interrupt.

TYPE 3 :Interrupt corresponds to break point interrupt.

TYPE 4 Interrupt corresponds to Overflow interrupt.

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

Differences between CALL and INT : In fact both the instructions CALL and INT n

will interrupt the execution of the main program. But there are certain differences

between their functioning. They are given below in the table.

S.No	CALL Instruction	INTn instruction
1	Upon the execution ,the control will jump to any one of the 1 MB of memory locations .	Upon execution the control will jump to a fixed location in the vector table.
2	The user can insert in the sequence of instructions of a program	Can occur at any time activated by hardware
3	Once initiated it cannot be masked	Can be masked

4	When initiated ,it stores the CS:IP of the next instruction on the stack	When initiated ,it stores the CS:IP of the next instruction and also the flag register on the stack.
5	The last instruction of the subroutine will be RET	The last instruction of the ISS will be IRET

ADDRESSING MODES :

The different ways in which a source operand is denoted in an instruction are known as the addressing modes. There are 8 different addressing modes in 8086 programming. They are

1. Immediate addressing mode
2. Register addressing mode
3. Direct addressing mode
4. Register indirect addressing mode
5. Based addressing mode
6. Indexed addressing mode.
7. Based indexed addressing mode
8. Based, Indexed with displacement.

Immediate addressing mode: The addressing mode in which the data operand is a part of the instruction itself is called Immediate addressing mode.

```
For Ex: MOV CX,  4847 H
        ADD AX, 2456 H
        MOV AL, FFH
```

Register addressing mode: Register addressing mode means, a register is the source of an operand for an instruction.

For Ex : MOV AX, BX copies the contents of the 16-bit BX register into the 16-bit AX register.
EX : ADD CX,DX

Direct addressing mode: The addressing mode in which the effective address of the memory location at which the data operand is stored is given in the instruction.i.e the effective address is just a 16-bit number is written directly in the instruction.

For Ex: MOV BX, [1354H]
MOV BL,[0400H]

. The square brackets around the 1354 H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register. This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

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.

Ex: MOV AX, [BX]. Suppose the register BX contains 4675H ,the contents of the 4675 H are moved to AX.

ADD CX,{BX}

Based addressing mode: The offset address of the operand is given by the sum of contents of the BX or BP registers and an 8-bit or 16-bit displacement.

Ex: MOV DX, [BX+04]

ADD CL,[BX+08]

Indexed Addressing mode: The operands offset address is found by adding the contents of SI or DI register and 8-bit or 16-bit displacements.

Ex: MOV BX,[SI+06]
ADD AL,[DI+08]

Based -index addressing mode*:* The offset address of the operand is computed by summing the base register to the contents of an Index register.

Ex: ADD CX,[BX+SI]

MOV AX,[BX+DI]

Based Iindexd with displacement mode: The operands offset is computed by adding the base register contents, an Index registers contents and 8 or 16-bit displacement**.**

Ex : MOV AX,[BX+DI+08]
ADD CX,[BX+SI+16]

INSTRUCTION SET OF 8086/8088

The 8086 microprocessor supports 6 types of Instructions. They are

1. Data transfer instructions
2. Arithmetic instructions
3. Bit manipulation instructions
4. String instructions
5. Program Execution Transfer instructions (Branch & loop Instructions)
6. Processor control instructions

1. Data Transfer instructions : These instructions are used to transfer the data from source operand to destination operand. All the store, move, load, exchange ,input and output instructions belong to to this group

General purpose byte or word transfer instructions:

MOV **:** Copy byte or word from specified source to specified destination

PUSH **:** Push the specified word to top of the stack

POP **:** Pop the word from top of the stack to the specified location

PUSHA **:** Push all registers to the stack

POPA **:** Pop the words from stack to all registers

XCHG **:** Exchange the contents of the specified source and destination operands one of which may be a register or memory location.

XLAT : Translate a byte in AL using a table in memory

Simple input and output port transfer instructions

1. IN : Reads a byte or word from specified port to the accumulator
2. OUT : Sends out a byte or word from accumulator to a specified port

Special address transfer instructions

1. LEA : Load effective address of operand into specified register
2. LDS : Load DS register and other specified register from memory
3. LES : Load ES register and other specified register from memory.

Flag transfer registers

1. LAHF : Load AH with the low byte of the flag register
2. SAHF : Store AH register to low byte of flag register
3. PUSHF : Copy flag register to top of the stack
4. POPF : Copy word at top of the stack to flag register

2. Arithmetic instructions : These instructions are used to perform various mathematical operations like addition, subtraction, multiplication and division etc….

Addition instructions

1.ADD **:** Add specified byte to byte or word to word
2.ADC **:** Add with carry

3.INC **:** Increment specified byte or specified word by 1
4.AAA **:** ASCII adjust after addition

5.DAA **:** Decimal (BCD) adjust after addition

Subtraction instructions

1. SUB : Subtract byte from byte or word from word
2. SBB : Subtract with borrow
3. DEC : Decrement specified byte or word by 1
4. NEG : Negate or invert each bit of a specified byte or word and add 1(2's complement)
5. CMP : Compare two specified byte or two specified words
6. AAS : ASCII adjust after subtraction
7. DAS : Decimal adjust after subtraction

Multiplication instructions

1. MUL **:** Multiply unsigned byte by byte or unsigned word or word.
2. IMUL **:** Multiply signed bye by byte or signed word by word
3. AAM **:** ASCII adjust after multiplication

Division instructions

1. DIV : Divide unsigned word by byte or unsigned double word by word
2. IDIV : Divide signed word by byte or signed double word by word
3. AAD : ASCII adjust after division
4. CBW : Fill upper byte of word with copies of sign bit of lower byte
5. CWD : Fill upper word of double word with sign bit of lower word.

3. Bit Manipulation instructions : These instructions include logical , shift and rotate instructions in which a bit of the data is involved.

Logical instructions

1. NOT :Invert each bit of a byte or word.
2. AND : ANDing each bit in a byte or word with the corresponding bit in another byte or word.
3. OR : ORing each bit in a byte or word with the corresponding bit in another byte or word.
3. XOR : Exclusive OR each bit in a byte or word with the corresponding bit in another byte or word.
4. TEST :AND operands to update flags, but don't change operands.

Shift instructions

1. SHL/SAL : Shift bits of a word or byte left, put zero(S) in LSBs.
2. SHR : Shift bits of a word or byte right, put zero(S) in MSBs.
3. SAR : Shift bits of a word or byte right, copy old MSB into new MSB.

Rotate instructions

1. ROL : Rotate bits of byte or word left, MSB to LSB and to Carry Flag [CF]
2. ROR : Rotate bits of byte or word right, LSB to MSB and to Carry Flag [CF]
3. RCR :Rotate bits of byte or word right, LSB TO CF and CF to MSB
4. RCL :Rotate bits of byte or word left, MSB TO CF and CF to LSB

4. String instructions

A string is a series of bytes or a series of words in sequential memory locations. A string often consists of ASCII character codes.

1. REP : An instruction prefix. Repeat following instruction until CX=0
2. REPE/REPZ : Repeat following instruction until CX=0 or zero flag ZF=1
3. REPNE/REPNZ : Repeat following instruction until CX=0 or zero flag ZF=1
4. MOVS/MOVSB/MOVSW: Move byte or word from one string to another
5. COMS/COMPSB/COMPSW: Compare two string bytes or two string words
6. INS/INSB/INSW: Input string byte or word from port
7. OUTS/OUTSB/OUTSW : Output string byte or word to port
8. SCAS/SCASB/SCASW: Scan a string. Compare a string byte with a byte in AL or a string word with a word in AX
9. LODS/LODSB/LODSW: Load string byte in to AL or string word into AX

5. Program Execution Transfer instructions

These instructions are similar to branching or looping instructions. These instructions include conditional & unconditional jump or loop instructions.

Unconditional transfer instructions

1. CALL : Call a procedure, save return address on stack
2. RET : Return from procedure to the main program.
3. JMP : Goto specified address to get next instruction

Conditional transfer instructions

1. JA/JNBE : Jump if above / jump if not below or equal
2. JAE/JNB : Jump if above /jump if not below
3. JBE/JNA : Jump if below or equal/ Jump if not above
4. JC : jump if carry flag CF=1
5. JE/JZ : jump if equal/jump if zero flag ZF=1
6. JG/JNLE : Jump if greater/ jump if not less than or equal
7. JGE/JNL : jump if greater than or equal/ jump if not less than
8. JL/JNGE : jump if less than/ jump if not greater than or equal
9. JLE/JNG : jump if less than or equal/ jump if not greater than
10. JNC : jump if no carry (CF=0)
11. JNE/JNZ : jump if not equal/ jump if not zero(ZF=0)
12. JNO : jump if no overflow(OF=0)
13. JNP/JPO : jump if not parity/ jump if parity odd(PF=0)
14. JNS : jump if not sign(SF=0)
15. JO : jump if overflow flag(OF=1)
16. JP/JPE : jump if parity/jump if parity even(PF=1)
17. JS : jump if sign(SF=1)

6. Iteration control instructions

These instructions are used to execute a series of instructions for certain number of times.

1. LOOP :Loop through a sequence of instructions until CX=0
2. LOOPE/LOOPZ : Loop through a sequence of instructions while ZF=1 and CX = 0
3. LOOPNE/LOOPNZ : Loop through a sequence of instructions while ZF=0 and CX =0
4. JCXZ : jump to specified address if CX=0

7. Interrupt instructions

1. INT : Interrupt program execution, call service procedure
2. INTO : Interrupt program execution if OF=1
3. IRET : Return from interrupt service procedure to main program

8. High level language interface instructions

1. ENTER : enter procedure
2. LEAVE :Leave procedure
3. BOUND : Check if effective address within specified array bounds

9. Processor control instructions

Flag set/clear instructions

1. STC : Set carry flag CF to 1
2. CLC : Clear carry flag CF to 0
3. CMC : Complement the state of the carry flag CF
4. STD : Set direction flag DF to 1 (decrement string pointers)
5. CLD : Clear direction flag DF to 0
6. STI : Set interrupt enable flag to 1(enable INTR input)
7. CLI : Clear interrupt enable Flag to 0 (disable INTR input)

10. External Hardware synchronization instructions

1. HLT : Halt (do nothing) until interrupt or reset
2. WAIT : Wait (Do nothing) until signal on the test pin is low
3. ESC : Escape to external coprocessor such as 8087 or 8089
4. LOCK : An instruction prefix. Prevents another processor from taking the bus while the adjacent instruction executes.

11. No operation instruction

1. NOP : No action except fetch and decode

ASSEMBLY LANGUAGE DEVELOPMENT TOOLS:

To develop an assembly language program we need certain program development tools .The various development tools required for 8086 programming are explained below.

1. Editor: An Editor is a program which allows us to create a file containing the assembly language statements for the program. Examples of some editors are PC write WordStar. As we type the program the editor stores the ACSII codes for the letters and numbers in successive RAM locations. If any typing mistake is done editor will alert us to correct it. If we leave out a program statement an editor will let you move everything down and insert a line. After typing all the program we have to save the program for a hard disk. This we call it as source file. The next step is to process the source file with an assembler. While using TASM or MASM we should give a file name and extension .ASM.

Ex: Sample. asm

2. Assembler: An Assembler is used to translate the assembly language mnemonics into machine language (i.e. binary codes). When you run the assembler it reads the source file of your program from where you have saved it. The assembler generates two files. The first file is the Object file with the extension **.OBJ**. The object file consists of the binary codes for the instructions and information about the addresses of the instructions. After further processing, the contents of the file will be loaded in to memory and run. The second file is the assembler list file with the extension .**LST**.

3. Linker: A linker is a program used to connect several object files into one large object file. While writing large programs it is better to divide the large program into smaller modules. Each module can be individually written, tested and debugged. Then all the object modules are linked together to form one, functioning program. These object modules can also be kept in library file and linked into other programs as needed. A linker produces a link file which contains the binary codes for all the combined modules. The linker also produces a link map file which contains the address information about the linked files. The linkers which come with TASM or MASM assemblers produce link files with the **.EXE** extension.

4.Locator : A locator is a program used to assign the specific addresses of where the segments of object code are to be loaded into memory. A locator program called EXE2BIN comes with the IBM PC Disk Operating System (DOS). EXE2BIN converts a .EXE file to a .BIN file which has physical addresses.

5. Debugger: A debugger is a program which allows to load your object code program into system memory, execute the program, and troubleshoot or debug it. The debugger allows to look into the contents of registers and memory locations after the program runs. We can also change the contents of registers and memory locations and rerun the program. Some debuggers allows to stop the program after each instruction so that you can check or alter memory and register contents. This is called single step debug. A debugger also allows to set a breakpoint at any point in the program. If we insert a break point , the debugger will run the program up to the instruction where the breakpoint is put and then stop the execution.

6. Emulator: An emulator is a mixture of hard ware and software. It is usually used to test and debug the hardware and software of an external system such as the prototype of a microprocessor based instrument.

Coding in Assembly language:

Assembly language programming language has taken its place in between the machine language (low level) and the high level language.

- High level language‘s one statement may generate many machine instructions.
- Low level language consists of either binary or hexadecimal operation. One symbolic statement generates one machine level instructions.

Advantage of ALP

- They generate small and compact execution module.
- They have more control over hardware.
- They generate executable module and run faster.

Disadvantages of ALP:

- Machine dependent.
- Lengthy code

- Error prone (likely to generate errors).

Assembly language features:

The main features of ALP are program comments, reserved words, identifies, statements and directives which provide the basic rules and framework for the language.

Program comments:

- The use of comments throughout a program can improve its clarity.
- It starts with semicolon (;) and terminates with a new line.
- E.g. ADD AX, BX ; Adds AX & BX

Reserved words:

- Certain names in assembly language are reserved for their own purpose to be used only under special conditions and includes
- Instructions : Such as MOV and ADD (operations to execute)
- Directives: Such as END, SEGMENT (information to assembler)
- Operators: Such as FAR, SIZE
- Predefined symbols: such as @DATA, @ MODEL

Identifiers:

- An identifier (or symbol) is a name that applies to an item in the program that expects to reference.
- Two types of identifiers are Name and Label.
- Name refers to the address of a data item such as NUM1 DB 5, COUNT DB 0
- Label refers to the address of an instruction.
- E. g: MAIN PROC FAR
- L1: ADD BL, 73

Statements:

- ALP consists of a set of statements with two types
- Instructions, e. g. MOV, ADD
- Directives, e. g. define a data item

	Identifiers	operation	operand	comment
Directive:	COUNT	DB	1	; initialize count
Instruction:	L30:	MOV	AX, 0	; assign AX with 0

Directives:

The directives are the number of statements that enables us to control the way in which the source program assembles and lists. These statements called directives act only during the assembly of program and generate no machine-executable code. The different types of directives are:

1) **The page and title listing directives:**

   The page and title directives help to control the format of a listing of an assembled program. This is their only purpose and they have no effect on subsequent execution of the program.

   The page directive defines the maximum number of lines to list as a page and the maximum number of characters as a line.

   PAGE [Length] [Width] Default:
   Page [50][80]

TITLE gives title and place the title on second line of each page of the program. TITLE text [comment]

2) SEGMENT directive

It gives the start of a segment for stack, data and code. Seg-name Segment [align][combine][class‘]
Seg-name ENDS

- Segment name must be present, must be unique and must follow assembly language naming conventions.
- An ENDS statement indicates the end of the segment.
- Align option indicates the boundary on which the segment is to begin; PARA is used to align the segment on paragraph boundary.
- Combine option indicates whether to combine the segment with other segments when they are linked after assembly. STACK, COMMON, PUBLIC, etc are combine types.
- Class option is used to group related segments when linking. The class code for code segment, stack for stack segment and data for data segment.

3) PROC Directives

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

PROC - name PROC [FAR/NEAR]
…….
…….
…….
PROC - name ENDP

- FAR is used for the first executing procedure and rest procedures call will be NEAR.
- Procedure should be within segment.

4) END Directive

- An END directive ends the entire program and appears as the last statement.
- ENDS directive ends a segment and ENDP directive ends a procedure. END PROC-Name

5) ASSUME Directive

- An .EXE program uses the SS register to address the stack, DS to address the data segment and CS to address the code segment.
- Used in conventional full segment directives only.
- Assume directive is used to tell the assembler the purpose of each segment in the program.
- Assume SS: Stack name, DS: Data Segname CS; codesegname

6) Processor directive

- Most assemblers assume that the source program is to run on a basic 8086 level computer.
- Processor directive is used to notify the assembler that the instructions or features introduced by the other processors are used in the program.
  e.g: .386 - program for 386 protected mode.

7) <u>Dn Directive (Defining data types):</u>

Assembly language has directives to define data syntax [name] Dn expression The Dn directive can be any one of the following:

DB	Define byte	1 byte
DW	Define word	2 bytes
DD	Define double	4 bytes
DF	defined farword	6 bytes
DQ	Define quadword	8 bytes
DT	Define 10 bytes	10 bytes

VAL1 DB **25**
ARR DB 21, 23, 27, 53
MOV AL, ARR [2] or

MOV AL, ARR + 2 ; Moves 27 to AL register

8) The EQU directive

- It can be used to assign a name to constants.
- E.g. **FACTOR** EQU **12**
- MOV BX, FACTOR ; MOV BX, 12
- It is short form of equivalent.
- Do not generate any data storage; instead the assembler uses the defined value to substitute in.

9) DUP Directive:

- It can be used to initialize several locations to zero. e. g.
  SUM DW 4 DUP(0)
- Reserves four words starting at the offset sum in DS and initializes them to Zero.
- Also used to reserve several locations that need not be initialized. In this case (?) is used with DUP directives.
  E. g. **PRICE** DB 100 DUP(?)
- Reserves 100 bytes of uninitialized data space to an offset PRICE.

ALP written in simplified segment segment directives:

```
Page 60, 132
TITLE Sum program to add two numbers.
• MODEL SMALL
    • STACK 64
    • DATA
        NUM1 DW 3241
        NUM 2 DW 572 SUM  DW ?
    • CODE
    MAIN PROC FAR
            MOV AX, @ DATA          ; set address of data segment in DS
            MOV DS, AX
        MOV AX, NUM1
        ADD AX, NUM2
        MOV SUM, AX
```

```
        MOV AX, 4C00H          ; End processing
        INT 21H
MAIN ENDP                      ; End of procedure
END MAIN                       ; End of program
```

Explanation:

- .Model memory model
  Memory model can be
  TINY, SMALL, MEDIUM, COMPACT, LARGE, HUGE or FLAT
  TINY for .com program
  FLAT for program upto 4 GB
- Assume is automatically generated
  .STACK [size in bytes]
  Creates stack segment
  .DATA: start of data segment
  .CODE: start of code segment
- DS register can be initialized as
  MOV AX, @DATA
  MOV DS, AX

DOS Debug(MASM)

1) Save the code text in **.ASM** format and save it to the same folder where masm and link files are stored.
2) Open dos mode and reach within that folder.
3) \>Masm filename.asm press enter for 4 times ⇨ makes.obj
4) \>Link filename press enter for 4 times ⇨ makes.exe
5) Filename.exe press enter ⇨ run the code
   For debug : u -unassemble
6) Debug filename.exe load from disk, w-write to disk
   A- Assembly, D-Display E-Enter, G-Run
   H- Perform hex operation, N-name P-Proceed or Execute
   Q-Quit R- display reg. control, T-trace(single mode)

Example:

```
TITLE Program to add ten numbers
 oModel small
 oSTACK 64
 o DATA
        ARR DB 73,,91, 12,15, 79, 94, 55, 89
        Sum dw ?
 o CODE
 MAIN PROC FAR
 Mov Ax @ DATA
 MOV DS, AX MOV CX, 10
 MOV Ax, 0
 LEA BX, ARR
```

```
L2: ADD Al, [BX]
    JNC L1
    INC AH
L1: INC BX
    LOOP L2
    MOV SUM, AX
    MOV AX, 4c00H
    INT 21H
MAIN ENDP
END MAIN
```

System functions

The OS is a collection of routines (procedures) useful for efficiently manage system's resources. These routines are divided in 2 categories: BIOS routines and DOS routines. For microcomputers IBM-PC compatible a series of system routines are available for the user. They are written as procedures accessed through the system specific for 8086 and named as soft interruptions.BIOS interruption are written to allow access for the user to system resources (hardware) DOS interruptions are written for OS's specific purposes. DOS interruptions facilitate work with files in FAT system of files, the user doesn't need to have a full knowledge of this system of files in order to create a file. Access to BIOS and DOS functions from user's programs is done through soft interruptions (INT instruction). The main groups of BIOS functions available for the user are:

INT 10h - use of video terminal

INT 11h - determine system's configuration

INT 12h - determine RAM's capacity

INT 13h - access to HDD and FDD

INT 14h - use of serial interface

INT 15h - APM

INT 16h - use of keyboard

INT 17h - use of parallel interface

INT 19h - loader of resident system on disk

INT 1Ah - real time clock controller(RTC)

During a subroutine call (by INT) other functions can be specified. The function is specified ”by convention” , placing it’s number in AH register. The call for a certain BIOS function is possible through a generic sequence:

```
MOV  AH, function_nr        ; specify function
INT  int_nr                 ; specify interruption
```

According to a function’s complexity a series of parameters can be specified by following it’s pattern.

DOS functions are mainly referring to files but there is a large variety of functions. Every DOS function is called with INT 21h and by specifying the desired function (eventually it’s parameters) in AH register. Specially used for string opretions.

INT 21H

DOS – INT21h functions, for keyboard and monitor.

(AH)	Function	Input parameters	Output parameters
00h	End program’s execution		
01h	Read character from keyboard and send it in echo to screen. If CTRL-BREAK are pressed INT 23h executes		(AL) –inserted character
02h	Show character on screen. If CTRL-BREAK are pressed INT 23h executes	(DL) –the character	
05h	Print character	(DL) – the character	
06h	Direct read/write - from keyboard - to screen	(DL) = 0FFh (DL) – the character	(AL) – the character (AL) = 0 (no character)
07h	Read character from keyboard without		(AL) – the character

	echo and without interpretation		
08h	Read character from keyboard without echo If CTRL-BREAK are pressed INT 23h executes		(AL) – the character
09h	Show a row from memory ending with $ (24h)	(DS:DX) – row address	
0Ah	Read from keyboard and place into a memory buffer a row of characters, until <CR> is pressed	(DS:DX) – buffer's address	
0Bh	Determines keyboard's situation. If CTRL-BREAK are pressed INT 23h executes		(AL) =Off one character is available (AL) =0 no character
0Ch	Initialize keyboard's for buffer and then call a function. The system waits for a character.	(AL) – requested function (01h, 06h, 07h, 08h, 0Ah) .	

INT 10H

This interruption facilitates the use of video terminal. Within 10h interruption there are many sub-functions which allow character posting and use of graphic modes. For graphic modes with bigger resolutions this interruption it's not recommended because it's slow; it is recommended to write directly into video memory. For example posting a character on screen: calling 10h interruption implies a big amount of code to be executed (interpreting parameters transferred into registers, setting sub-function etc), while for writing directly into video memory only a MOV instruction is needed.

Still, this interruption is very practical for programs that are not posting big amount of information on screen at certain moment. By using this interruption the programmer doesn't have to calculate video memory addresses in which to write every character.

(AH)	Function	Input parameters	Output parameters
00h	Select functioning mode for graphic terminal	(AL) =0 alphanumeric 40 col. x25 lin. b/w =1 alphanumeric 40 col. x25 lin. Col =2 alphanumeric 80 col. x25lin. b/w =3 alphanumeric 80 col. x25lin. col. =4 graphic 320 colx200 lin col. =5 graphic 320 colx200 lin b/w =6 graphic 640 colx200 lin b/w	
01h	Select shape and size for cursor	(CH) 0-4 bites for cursor's start line (CH) bites 5-7=0 (CL) 0-4 bites for cursor's finish line (CL) bites 5-7=0	

02h	Cursor positioning	(DH, DL) lin., col. (0, 0) –left upper corner (BH) – page nr. =0 for graphic mode	
03h	Read cursor's coordinates	(BH) – page nr =0 for graphic mode	(DH, DL) lin. , col. (CH, CL) shape
04h	Read position for optic indicator		(AH) =0 light pen inactive =1 light pen active (DH, DL) lin., col. Cursor (CH) line pixel (0-199) (BX) col pixel (0-319636)
05h	Select active display pages	(AL) – page nr. 0-7 for mode 0 and 1 0-3 for mode 2 and 3	
06h	Execute operation "scroll up"	(AL) – nr. of lines (AL) =0 delete window (CH, CL) – lin., col. for left upper corner (DH, DL) – lin., col. for right down corner (BH) – the attribute of a white line	

07h	Execute operation "scroll down"	(AL) – nr. of lines (AL) =0 delete window (CH, CL) – lin. , col. for left upper corner (DH, DL) – lin. , col. for right down corner (BH) – the attribute of a white line	
08h	Read character from screen and determine it's attribute.(the character is read from cursor's current position)	(BH) –referred number of page(only for alphanumeric mode)	(AL) – read character (AH) – character's attribute
09h	Show character on screen.	(BH) – referred number of page (only for alphanumeric mode) (BL) –character's attribute (for alphanumeric mode) - color (for graphic mode) (CX) – nr. of characters to show (AL) – character	

0Ah	Replace characters on screen maintaining color characteristics	(BH) –the number for reference page (CX) – nr. of characters to show (AL) (AL) – character	
0Bh	Set color characteristics(only for 320x200 pixels graphic mode)	(BH) – palette's index (conf doc IBM-PC) (BL)-value of color within palette	
0Ch	Show point on screen (for graphic mode)	(DX) – line's number (CX) – column's number (AL) –color's value (0, 1, 2, 3)	
0Dh	Read point's color (for graphic mode)	(DX) – line's number (CX) – column's number	(AL) – read color
0Eh	Show character on screen and update cursor's position	(AL) – character to show (BL) –background color (for graphic mode) (BH) – page (for alphanumeric mode)	
0Fh	Read characteristics for current mode		(AL) – current module (AH) – character's number of columns (BH) – page number

Example:

TITLE to display a string

```
oMODEL SMALL
o STACK 64
o DATA
  STR DB _programming is fun‘, _$‘
o CODE
  MAIN PROC FAR
        MOV AX, @ DATA MOV
        DS, AX MOV AH, 09H
        LEA DX, STR
        INT 21H
        MOV AX, 4C00H INT
        21H
  MAIN ENDP END
  MAIN
```

TITLE to reverse the string

- MODEL SMALL
- STACK 100H
- DATA

  STR1 DB “My name is Rahul , _$”
  STR2 dbS dup (_$‘)
- CODE

```
MAIN PROC FAR
        Mov BC, 00H
        Mov Ax, @ DATA
        Mox DS, AX
        Mov SI, OFFSER STR1
        Mov DI, OFFSET STR2
L2: mov DL, [SI]
        CMP Dl, _$‘
        JE L1
        INC SI
        INC SI
INC BL
        JMP l2
L1: movCl, BL
        Mov CH, 00H
        DEL SI
L3: Mov AL, [SI]
        Mov [DI], AL
        DEC SI
        INC DI
        LOOP L3
```

```
# TITLE to input characters until _q‘ and display o
    MODEL SMALL
    o   STACK 100H
    o   DATA
            ▪ STR db 50 DUP (_$‘)
    o   CODE
        MAIN PROC FAR
                Mov ax, @ DATA
                Mov DS, AX
                Mov SI, OFFSET STR
                L2: mov AH, 01H
                INT 21H
                Cmp AL. _q‘
                JE L!
                Mov [SI] , AL
                INC SI
                JMP l2
                L1: mov AH, 09H
                Mov DX, OFFSET str
                INT 21H
                Mov Ax, 4c00H
                INT 21H
        MAIN  ENDP  END
        MAIN
```

Calling procedure/subroutine

```
Procname PROC FAR

Procname ENDP
```

Here the code segment consists only one procedure. The FAR operand in this case informs the assembler and linker that the defined procedure name is the entry point for program execution, whereas the ENDP directive defines the end of the procedure. A code segment however, may contain any number of procedures, each distinguished by its own PROC and ENDP directives.

A called procedure is a section of code that performs a clearly defined task known as subroutine which provides following benefits.

- Reduces the amount of code because a common procedure can be called from any number of places in the code segment.
- Encourage better program organization.
- Facilitates debugging of a program because defects can be more clearly isolated.
- Helps in the ongoing maintenance of programs because procedures are readily identified for modification.

  A CALL to a procedure within the same code segment is NEAR CALL& a FAR CALL calls a procedure labeled FAR, possibly in another code segment.

  Eg. For near call:

```
CALL        DISPLAY
```

```
DISPLAY PROC NEAR
        MOV AH, 09H
        MOV DX, OFFSET STR INT
        21H
        RET DISPLAY
ENDP
```

To display number contained in [Bx]

```
DISPLAY PROC NEAR
        Mov Dl, [BX]
        ADD Dl, 30
        MOV AH, 02H
        INT 21H
        RET
DISPLAY END
```

Macro Assembler:

- A macro is an instruction sequence that appears repeatedly in a program assigned with a specific name.
- The macro assemble replaces a macro name with the appropriate instruction sequence each time it encounters a macro name.
- When same instruction sequence is to be executed repeatedly, macro assemblers allow the macro name to be typed instead of all instructions, provided the macro is defined.
- Macro are useful for the following purpose:
    - To simplify and reduce the amount of repetitive coding
    - To reduce errors caused by repetitive coding
    - To make an assembly language program more readable
    - Macro executes faster because there is no need to call and return.

  Basic format of macro definition:

```
Macroname       MACRO [Parameter list] ; Define macro
                [instructions]         ; macro body
                ENDM                   ; end of macro
```

  Eg:

```
Addition  MACRO
          IN AX, PORT
          Add Ax, BX
          OUT PORT, AX
          ENDM
```

  # Passing argument to MACRO:

- To make a macro more flexible, we can define parameters as dummy argument

```
ADDITION MACRO CAL1, VAL2
          MOV  Ax,    VAL1
          ADD Ax,     VAL2
          Mov SUM,    Ax
          ENDM
```

- MODEL SMALL
- STACK 64
- DATA

```
        VAL1 DW     3241
        VAL2 DW     57
        SUM  DW     ?
```

- CODE

```
MAIN PROC FAR
        Mov Ax, @ DATA MOV DS, AX
        ADDITION Val1, VAL 2 MOV
        AX, 4C00H
        INT 21H

        MAIN ENDP

        END
```

Chapter-6

Memory Interface and Interrupt Operations

Introduction:

Memory is an essential component of the microcomputer system. It is used to store both instructions and data. It is used to store both instructions and data. Memory is made up of registers and the number of bits stored in a register is called memory word .Memory word is identified by an address .If microprocessor uses 16 bit address , then there will be maximum of 2^{16}= 65536 memory addresses ranging from 0000H to FFFFH.

There are various types of memory which can be classified in to two main groups i.e. Primary memory and Secondary memory.

Memory Devices

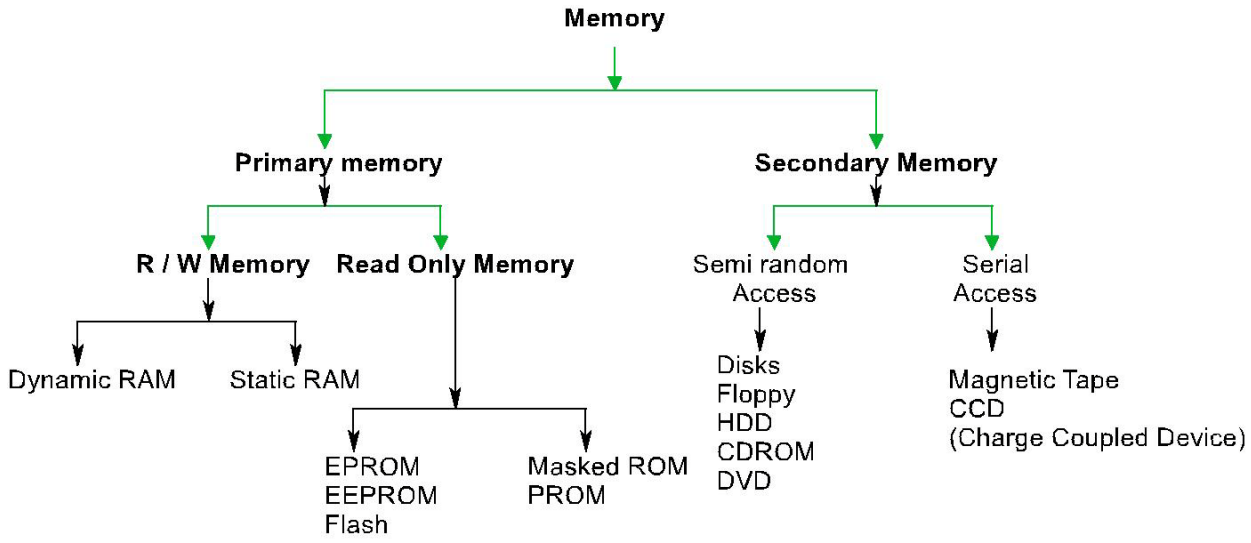


Fig: Classfication of memory system

1. Primary Memory:

It is the memory used by microprocessor to execute programs. The microprocessor can access only those items that are stored in this memory. Hence, all data and program must be within primary memory prior to its execution. Primary memory is much larger than processer memory that is included in the microprocessor chip.

Primary memory is divided in to two groups. i. R/W Memory (RAM)

Microprocessor can read for and write into this memory .This memory is used for information that are likely to be altered such as writing program or receiving data. This memory is volatile i.e. the content will be lost if the power is turned off and commonly known as RAM, RAM are basically of two types.

A. Static RAM (SRAM)

This memory is made up of flip flops and it stores bit as voltage. A single flip flop stores

binary data either 1 or 0. Each flip flop is called storage cell. Each cell requires six transistors. Therefore, the memory chip has low density but high speed. This memory is more expensive and consumes more power.

B. Dynamic RAM (DRAM)

This memory is made up of MOS transistor gates and it stores the bit as charge. The advantage of DRAM are it has high density, low power consumption and cheaper than SRAM. But the bit information leaks therefore needs to be rewritten again every few milliseconds. It is called refreshing the memory and requires extra circuitry to do this. It is slower than SRAM.

Read Only Memory (ROM):

ROM contains a permanent pattern of data that cannot be changed. It is non volatile that is no power source is required to maintain the bit values in memory. ROM are basically of 5 types.

A. Masked ROM: A bit pattern is permanently recorded by the manufactures during production.
B. Programmable ROM: In this ROM, a bit pattern may be written into only once and the writing process is performed electrically. That may be performed by a supplier or customer.
C. Erasable PROM (EPROM):

This memory stores a bit in the form of charge by using EPROM programmer which applies high voltage to charge the gate .Information can be erased by exposing ultra violet radiation. It is reusable. The disadvantages are :(i) it must be taken out off circuit to erase it (ii). The entire chip must be erased (iii) the erasing process takes 15 to 20 minutes.

D. Electrically Erasable PROM(EEPROM):

It is functionally same as EPROM except that information can be altered by using electrical signal at the register level rather than erasing all the information. It is expensive compared to EPROM and flash and can be erased in 10 ms.

E. Flash Memory:

It is variation of EPROM. The difference is that EPROM can be erased in register level but flash memory must be erased in register level but flash memory must be erased in its entirety or at block level.

2. Secondary memory

The devices that provide backup storage are called secondary memory. It includes serial access type such as magnetic disks and random access type such as magnetic disks. It is nonvolatile memory.

Performance of memory:

1. Access time (t_a):

Read access time: It is the average time required to read the unit of information from memory.

Write access time: It is the average time required to write the unit of information on memory.

Access rate (r_a) = $1/t_a$

2. Cycle time (t_c):

It is the average time that lapses between two successive read operation. Cycle rate (r_c)= bandwidth = $1/t_c$

Access modes of memory:

1. Random access: In random access mode, the t_a is independent of the location from which the data is accessed like MOS memory.
2. Sequential access: In that mode, the t_a is dependent of the location form which the data is accessed like magnetic type.
3. Semi random-access: the semi random access combines these two. foreg. In magnetic disk, any track can be accessed at random. But the access within the truck must be in serial fashion.

Address decoding:

Microprocessor is connected with memory and I/O devices via common address and data bus. Only one device can send data at a time and other devices can only receive that data. If more than one device sends data at the same time, the data gets garbled. In order to avoid this situation, ensuring that the proper device gets addressed at proper time, the technique called address decoding is used.

In address decoding method, all devices like memory blocks, I/O units etc. are assigned with a specific address. The address of the device is determined from the way in which the address lines are used to derive a
special device selection signal k/a chip select (CS). If the microprocessor has to write or to read from a device, the CS signal to that block should be enabled and the address decoding circuit must ensure that CS signal to other devices are not activated. Depending upon the no. of address lines used to generate chip select signal for the device, the address decoding is classified as:

1. **I/O mapped I/O**
   In this method, a device is identified with an 8 bit address and operated by I/O related functions IN and OUT for that IO/M =1. Since only 8 bit address is used, at most 256 bytes can be identified uniquely. Generally low order address bits A_0-A_7 are used and upper bits A_8-A_{15} are considered don't care. Usually I/O mapped I/O is used to map devices like 8255A, 8251A etc.
2. **Memory mapped I/O**
   In this method , a device is identified with 16 bit address and enabled memory related functions such as STA , LDA for which IO/M =0, here chip select signal of each device is derived from 16 bit address lines thus total addressing capability is 64K bytes . Usually memory mapped I/O is used to map memories like RAM, ROM etc.

Depending on the address that are allocated to the device the address decoding are categorized in the following two groups.

1. **Unique Address Decoding:**
   If all the address lines on that mapping mode are used for address decoding then that decoding is called unique address decoding. It means all 8-lines in I/O mapped I/O and all 16 lines in memory mapped I/O are used to derive $\overline{CS}$ signal. It is expensive and complicated but fault proof in all cases.

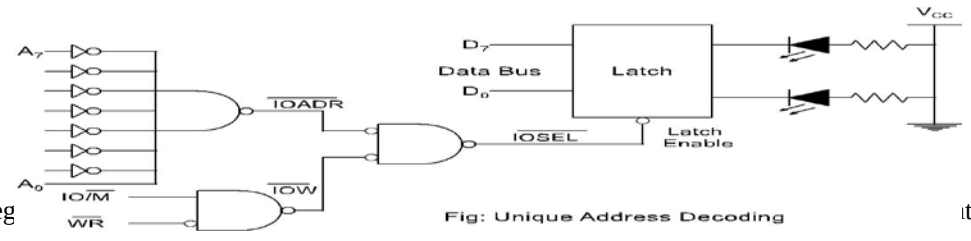


Fig: Unique Address Decoding

- f A_0 is high and A_1- $A_7$are low and if IOW becomes low, the latch gets enabled.
- The data to the LED can be transferred in only one case and hence the device has unique Address of 01H. Eight I/P switch interfacing at 53H. (01010011)

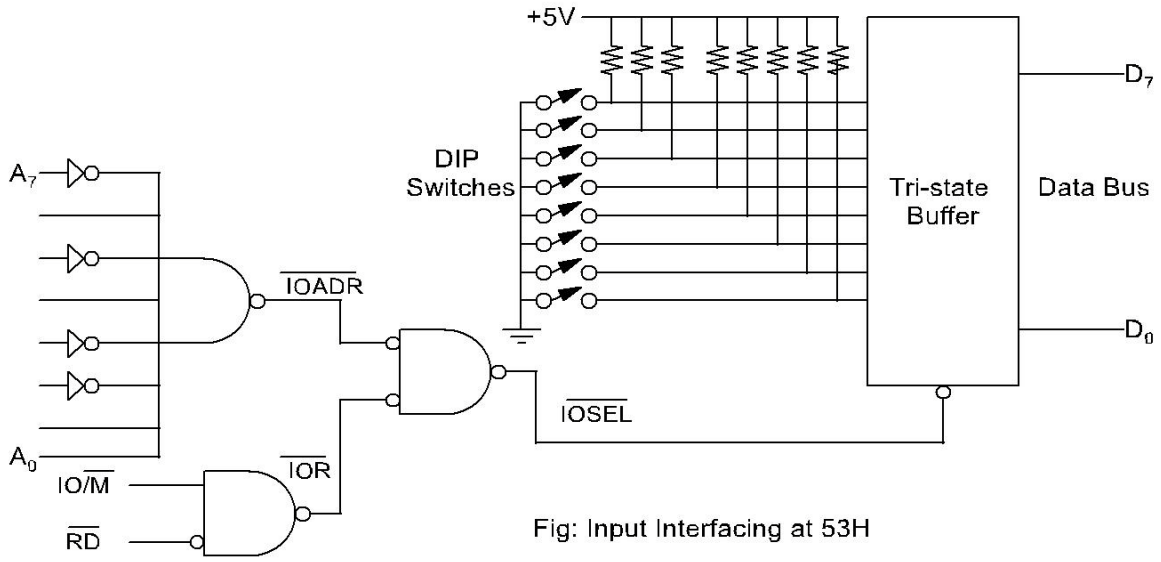


Fig: Input Interfacing at 53H

Non Unique Address decoding:

If all the address lines available on that mode are not used in address decoding then that decoding is called non unique address decoding. Though it is cheaper there may be a chance of address conflict.

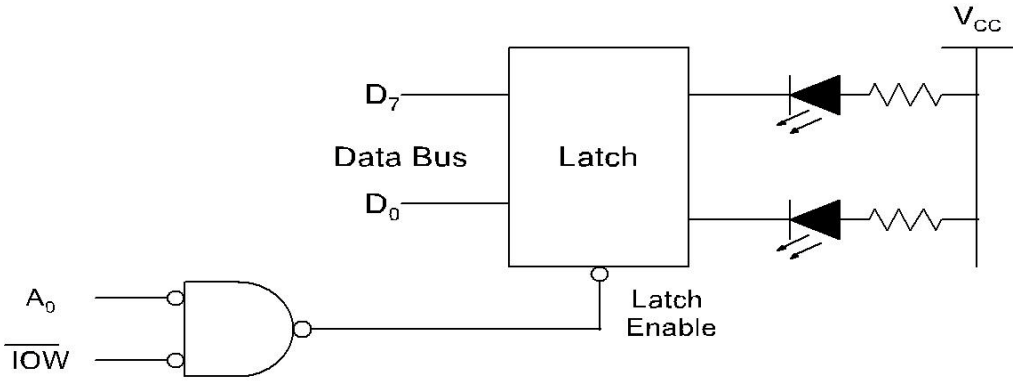


Fig: Non unique Address Decoding

- If A_0 is low and $\overline{IOW}$ is low. Then latch gets enabled. Here A_1-A_7 is neglected that is any even address can enable the latch.

DMA (Direct Memory Access)

Introduction

- An **alternative** to the basic and interrupt-driven I/O discussed previously

Direct memory access (DMA) is a feature of modern computer systems that allows certain hardware subsystems to read/write data to/from memory without microprocessor intervention, allowing the processor to do other work.

Used in disk controllers, video/sound cards etc, or between memory locations.

- DMA can transfer **large blocks of data** from memory to device or from device to memory
    - Uses same Address/Data lines on system bus
    - Controls the system bus instead of the processor ("bus master")
    - Does not require the processor for data transfer

Direct Memory Access (DMA)

During any given bus cycle, one of the system components connected to the system bus is given control of the bus. This component is said to be the master during that cycle and the component it is communicating with is said to be the slave. The CPU with its bus control logic is normally the master, but other specially designed components can gain control of the bus by sending a bus request to the CPU. After the current bus cycle is completed the CPU will return a bus grant signal and the component sending the request will become the master.

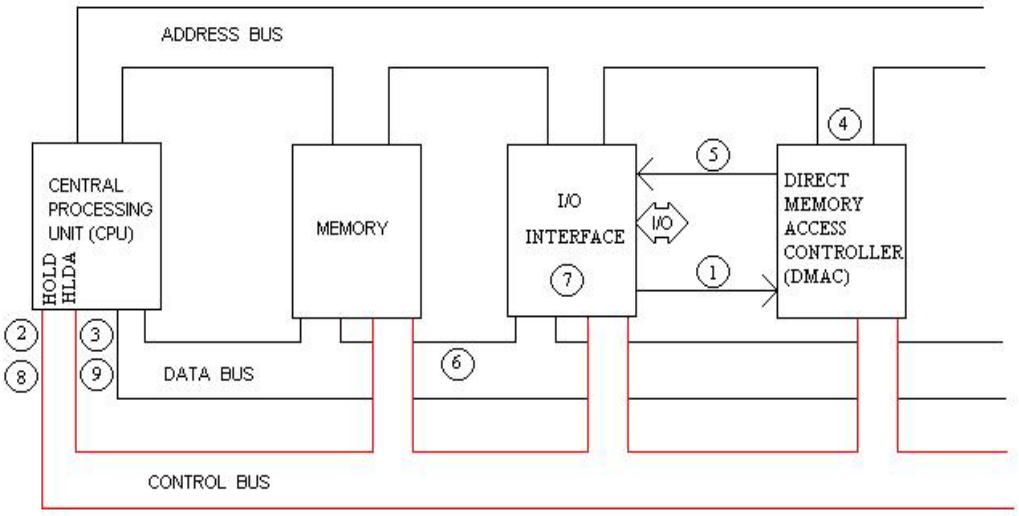
Taking control of the bus for a bus cycle is called **cycle stealing**. Just like the bus control logic, a master must be capable of placing addresses on the address bus and directing the bus activity during a bus cycle. The components capable of becoming masters are processors (and their bus control logic) and DMA controllers. Sometimes a DMA controller is associated with a single interface, but they are often designed to accommodate more than one interface.

The 8086 microprocessor receives bus requests through its HOLD pin and issues grants from the hold acknowledge (HLDA) pin. A request is made when a potential master sends a 1 to the HOLD pin. Normally, after the current bus cycle is complete the 8086 will respond by putting a 1 on the HLDA pin. When the requesting device receives this grant signal it becomes the master. It will remain master until it drops the signal to the HOLD pin, at which time the 8086 will drop the grant on the HLDA pin. One exception to the normal sequence is that if a word, which begins at an odd address is being accessed, then two bus cycles are required to complete the transfer and a grant will not be issued until after the second bus cycle.

When a DMA controller becomes master it places an address on the address bus and sends the interface

the necessary signals to cause it to put data on, or receive data from, the data bus. Since the DMA controller determines when the bus request is dropped, it can return control to the CPU after each data byte is transferred and then request control again when the next data byte is ready, or it can retain control until the entire block is moved. The former is the usual case because this allows the CPU to continue its work until the next data byte is available.

Basic DMA Operation

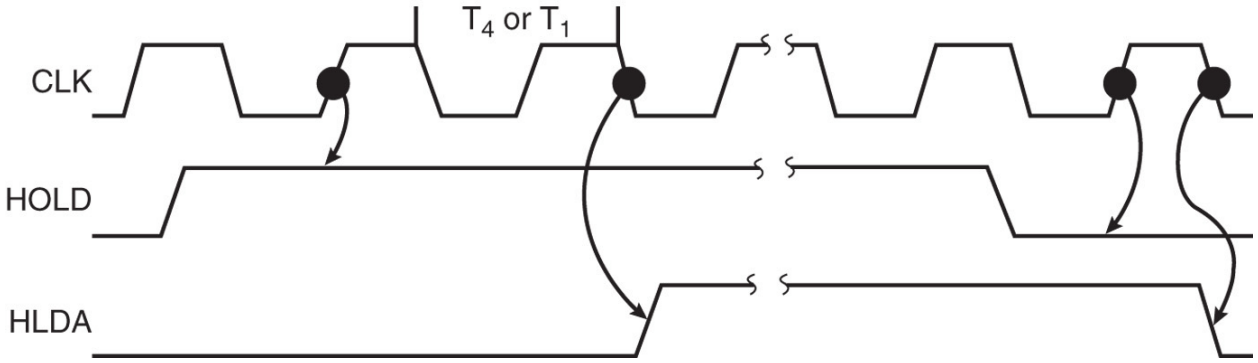


DATA TRANSFER WITH A DMA CONTROLLER

During a block input byte transfer, the following sequence occurs as the data byte is sent from the interface to the memory:

1. The interface sends the DMA controller a request for DMA service.
2. A Bus request is made to the HOLD pin (active High) on the 8086 microprocessor and the controller gains control of the bus.
3. A Bus grant is returned to the DMA controller from the Hold Acknowledge (HLDA) pin (active High) on the 8086 microprocessor.
4. The DMA controller places contents of the address register onto the address bus.
5. The controller sends the interface a DMA acknowledgment, which tells the interface to put data on the data bus. (For an output it signals the interface to latch the next data placed on the bus.)
6. The data byte is transferred to the memory location indicated by the address bus.
7. The interface latches the data.
8. The Bus request is dropped, the HOLD pin goes Low, and the controller relinquishes the bus.
9. The Bus grant from the 8086 microprocessor is dropped and the HLDA pin goes Low.
10. The address register is incremented by 1.
11. The byte count is decremented by 1.
12. If the byte count is non-zero, return to step 1, otherwise stop.

Following Figure shows HOLD and HLDA timing for the microprocessor.



  - HOLD is sampled in any clocking cycle
  - when the processor recognizes the hold, it stops executing software and enters hold cycles
  - HOLD input has higher priority than INTR or NMI
  - the only microprocessor pin that has a higher priority than a HOLD is the RESET pin
- HLDA becomes active to indicate the processor has placed its buses at high-impedance state.
  - as can be seen in the timing diagram, there are a few clock cycles between the time that HOLD changes and until HLDA changes
- HLDA output is a signal to the requesting device that the processor has relinquished control of its memory and I/O space.
  - one could call HOLD input a DMA request input and HLDA output a DMA grant signal

Direct Memory Access Controller (DMAC) options for data transfer

The DMA Controller has several options available for the transfer of data. They are:

1) Cycle Steal

A read or write signal is generated by the DMAC, and the I/O device either generates or latches the data. The DMAC effectively steals cycles from the processor in order to transfer the byte, so single byte transfer is cycle stealing.

We encounter cycle stealing in the context of Direct Memory Access (DMA). Either the DMA controller can use the data bus when the CPU does not need it, or it may force the CPU to temporarily suspend operation. The latter technique is called cycle stealing. Note that cycle stealing can be done only at specific break points in an instruction cycle.

2) Burst Transfer:

To achieve block transfers, some DMAC's incorporate an automatic sequencing of the value presented on the address bus. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.

3) Hidden DMA:

It is possible to perform hidden DMA, which is transparent to the normal operation of the CPU. In other words, the bus is grabbed by the DMAC when the processor is not using it. The DMAC monitors the execution of the processor, and when it recognizes the processor executing an instruction which has sufficient empty clock cycles to perform a byte transfer; it waits till the processor is decoding the op code, and then grabs the bus during this time. The processor is not slowed down, but continues processing normally. Naturally, the data transfer by the DMAC must be completed before the processor starts

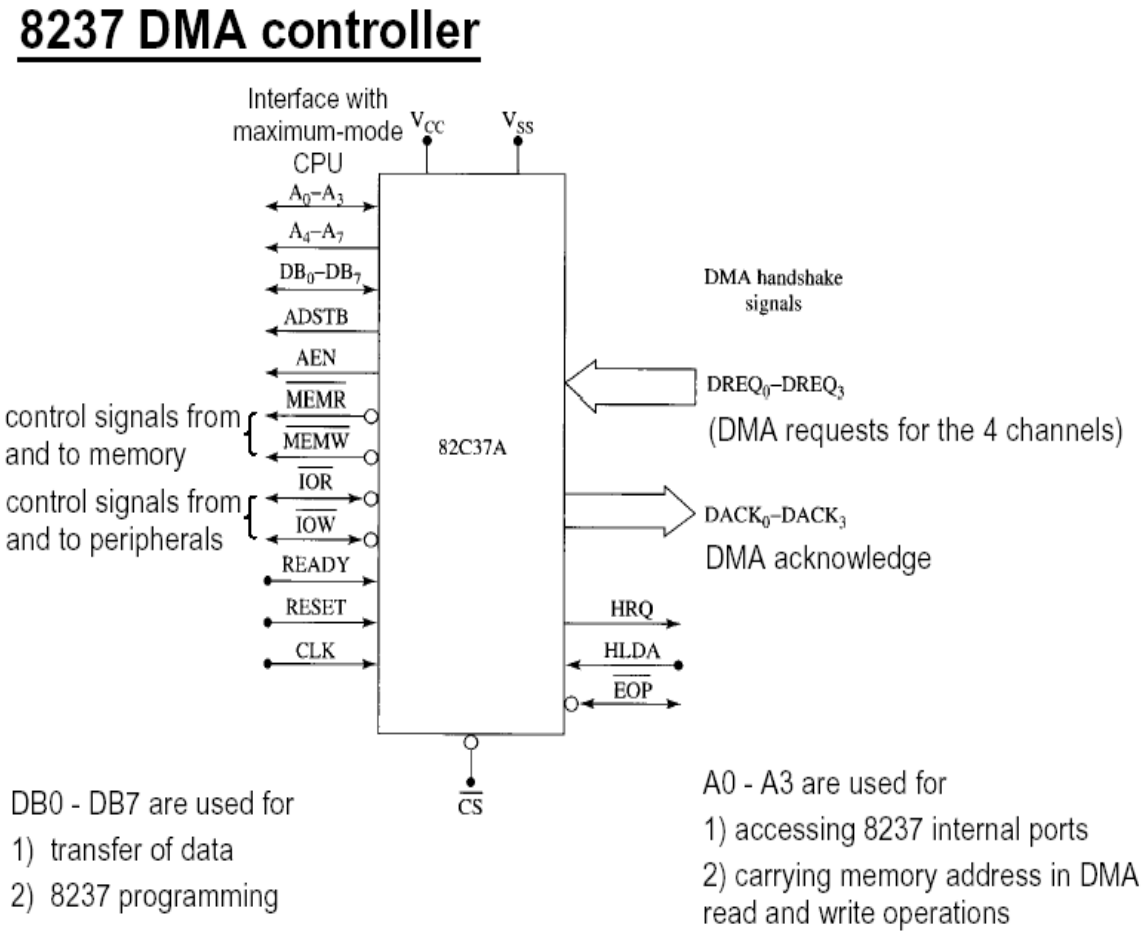
PROGRAMMABLE DMA CONTROLLER - INTEL 8237

It is a device to transfer the data directly between IO device and memory without through the CPU. So it performs a high-speed data transfer between memory and I/O device.

The features of 8237 is,

- ☐ The 8237 has four channels and so it can be used to provide DMA to four I/O devices.
- ☐ Each channel can be independently programmable to transfer up to 64kb of data by DMA.
- ☐ Each channel can be independently perform read transfer, write transfer and verify transfer.

It is a 40 pin IC and the pin diagram is,



8237 Pin Definitions

1. **CLK**
   - **Clock** input is connected to the system clock signal as long as that signal is 5 MHz or less.
     - in the 8086/8088 system, the clock must be inverted for the proper operation of the 8237
2. **CS**
   - **Chip select** enables 8237 for programming.
   - The CS pin is normally connected to the output of a decoder.
   - The decoder does not use the 8086/8088 control signal IO/M(M/IO) because it contains the new memory and I/O control signals (MEMR, MEMW, IOR and IOW).
3. **RESET**
   - The **reset** pin clears the command, status, request, and temporary registers.

- It also clears the first/last flip-flop and sets the mask register. This input primes the 8237 so it is disabled until programmed otherwise.

4. **READY**

- Logic 0 on the **ready** input causes the 8237 to enter wait states for slower memory components.

5. **HLDA**

- A **hold acknowledges** signals 8237 that the microprocessor has relinquished control of the address, data, and control buses.

6. **$DREQ_0$–$DREQ_3$**

- **DMA request inputs** are used to request a transfer for each of the four DMA channels. The polarity of these inputs is programmable, so they are either active-high or active-low inputs

7. DB0 ±DB7

- DATA BUS: The Data Bus lines are bidirectional three-state signals connected to the system data bus.
- The outputs are enabled in the Program condition during the I/O Read to output the contents of an Address register, a Status register, the Temporary register or a Word Count register to the CPU.
- The outputs are disabled and the inputs are read during an I/O Write cycle when the CPU is programming the 8237A control registers.
- During DMA cycles the most significant 8 bits of the address are output onto the data bus to be strobe into an external latch by ADSTB. In memory-to-memory operations, data from the memory comes into the 8237A on the data bus during the read-from-memory transfer. In the write-to-memory transfer, the data bus outputs place the data into the new memory location.

8. IOR

- I/O READ: I/O Read is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to read the control registers. In the Active cycle, it is an output control signal used by the 8237A to access data from a peripheral during a DMA Write transfer.

9. IOW

- I/O WRITE: I/O Write is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to load information into the 8237A. In the Active cycle, it is an output control signal used by the 8237A to load data to the peripheral during a DMA Read transfer.

10. A0 ±A3

- ADDRESS: The four least significant address lines are bidirectional three-state signals. In the Idle cycle they are inputs and are used by the CPU to address the register to be loaded or read. In the Active cycle they are outputs and provide the lower 4 bits of the output address.

11. A4 ±A7 Output

- ADDRESS: The four most significant address lines are three-state outputs and provide 4 bits of address. These lines are enabled only during the DMA service.

12. HRQ Output

- HOLD REQUEST: This is the Hold Request to the CPU and is used to request control of the system bus. If the corresponding mask bit is clear, the presence of any valid DREQ causes 8237A to issue the HRQ.

13. DACK0 ±DACK3 Output

- DMA ACKNOWLEDGE: DMA Acknowledge is used to notify the individual peripherals when one has been granted a DMA cycle. The sense of these lines is programmable. Reset initializes them to active low.

14. AEN

- ADDRESS ENABLE: Address Enable enables the 8-bit latch containing the upper 8 address bits onto the system address bus. AEN can also be used to disable other system bus drivers during DMA transfers. AEN is active HIGH.

15. ADSTB

- ADDRESS STROBE: The active high, Address Strobe is used to strobe the upper address byte into an external latch.

16. MEMR

- MEMORY READ: The Memory Read signal is an active low three-state output used to access data from the selected memory location during a DMA Read or a memory-to-memory transfer.

17. MEMW

- MEMORY WRITE: The Memory Write is an active low three-state output used to write data to the selected memory location during a DMA Write or a memory-to-memory transfer.

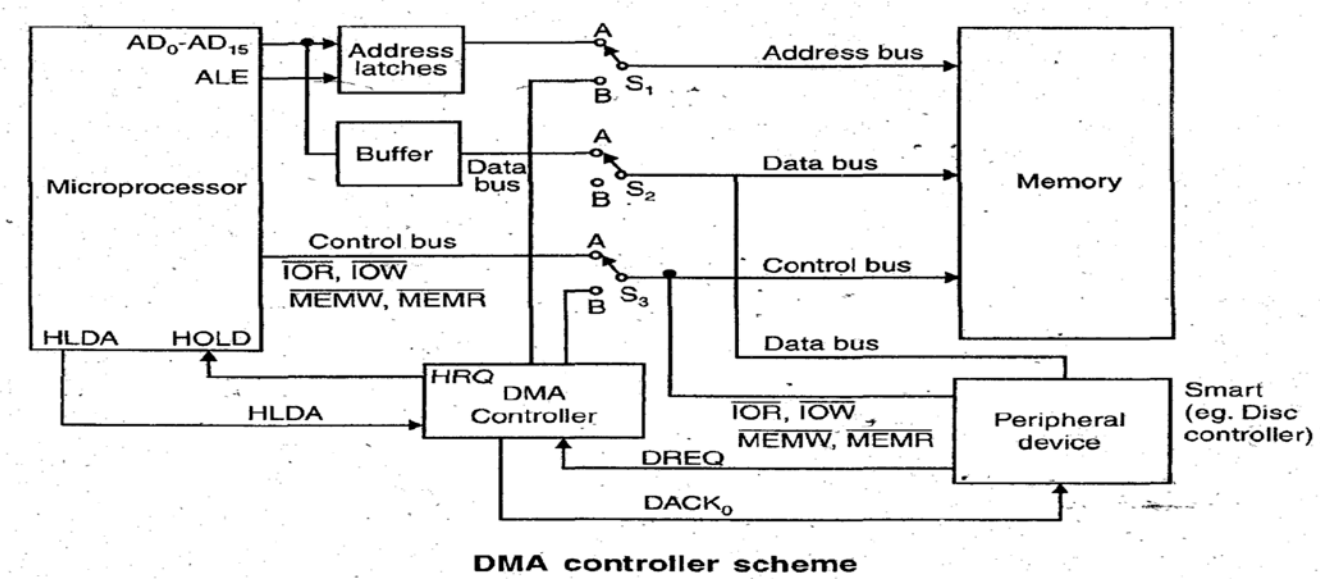
DMA Interfacing with Microprocessor

In I/O data transfer data is transferred by using microprocessor .The microprocessor will read data from I/O device and then will write data to memory

- In this case there are two operations for single data transfer.
- If the data is less, then micro process will not waste its time; transferring data from I/O to memory or back. But suppose, data is huge, then the transfer rate from I/O to memory or back will slow

down because of microprocessor intervention. In such case, to speed up the process of transferring the data, we can think, Can I/O have direct access to memory and the answer is, yes.

- It can have Direct memory access (DMA), but under Supervision. The device which supervises, data transfer is



DMA controller scheme

named as DMA controller.

- Now let's have diagrammatic representation of the scheme, which depicts microprocessor, DMA controller, memory and I/O device.

System Interface

- The DMA is used to transfer data bytes between I/O (such as floppy disk) and system memory (or from memory to memory) at high speed. It includes eight data lines, four control signals (IOR, IOW, MEMR, and MEMW), and eight address lines (A7-A0). However, it needs 16 address lines to access 64K bytes. Therefore, an additional eight lines must be generated as shown in figure 15.34.
- When a transfer begins, the DMA places the low-order byte on the address bus and the high-order byte on the data bus and asserts AEN (Address Enable) and ADSTB (Address Strobe). These two signals are used to latch the high-order byte from the data bus: thus, it places the 16-bit address on the system bus. After the transfer of first byte, the latch is updated when the lower byte generates a carry (or borrows). Figure 15.34 shows two latches: one latch (373 #1) to latch a high-order address from the data bus by using the AEN and ADSTB signals, and the second latch (373 #2) to demultiplex the 8085 bus and generate the low-order address bus by using the ALE (Address Latch Enable from the 8085) signal. The AEN signal is connected to the OE signal of the second latch to disable the low-order address bus from the 8085 when the first latch is enabled to latch the high-order byte of address.

Programming The 8237

To implement the DMA transfer, the 8237 should be initialized by writing into various control registers discussed earlier in the DMA channels and interfacing section. To initialize the 8237, the following steps are necessary:

1. Write a control word in the Mode register that selects the channel and specifies the type of transfer (Read, Write or Verify) and the DMA mode (block, single-byte, etc.).

2. Write a control word in the Command Register that specifies parameters such as priority among four channels. DREQ and DACK active levels, and timing, and enables the 8237.
3. Write the starting address of the data block to be transferred in the channel Memory Address Register (MAR).
4. Write the count (the number of the bytes in the data block) in the channel Count register.

Advantages of DMA

- Computer system performance is improved by direct transfer of data between memory and I/O devices, bypassing the CPU.
- CPU is free to perform operations that do not use system buses.
- 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.
- Minimizes latency in servicing a data acquisition device because the dedicated hardware responds more quickly than interrupts and transfer time is short.
- Minimizes latency reduces the amount of temporary storage (memory) required on an I/O device.
- Processor is not used for holding the data transfer activity and is available for other processing activity.
- Also in systems where the processor primarily operates out of its cache, data transfer actually occurring in parallel, thus increasing overall system utilization**.**

Application:

- Extensively used for computer-based data acquisition applications including streaming data to disk, real-time screen data display and continuous data acquisition applications.
- DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions.
- 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.
- Data acquisition applications have the same needs and therefore can take advantage of the technology developed for larger markets.

Disadvantages of DMA

- In case of Burst Mode data transfer, the CPU is rendered inactive for relatively long periods of time.

Interrupts

Introduction

- Interrupt is signals send by an external device to the processor, to request 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.
- The processor will check the interrupts always at the 2nd T-state of last machine cycle.
- If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral.
- The vectored address of particular interrupt is stored in program counter.
- The processor executes an interrupt service routine (ISR) addressed in program counter.
- It returned to main program by RET instruction.

Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices that provideor require data at relatively low data transfer rate.

Interrupt Operations

The transfer of data between the microprocessor and input /output devices takes place using various modes of operations like programmed I/O, interrupt I/O and direct memory access. In programmed I/O, the processor has to wait for a long time until I/O module is ready for operation. So the performance of entire system degraded. To remove this problem CPU can issue an I/O command to the I/O module and then go to do some useful works. The I/O device will then interrupt the CPU to request service when it is ready to exchange data with CPU. In response to an interrupt, the microprocessor stops executing its current program and calls a procedure which services the interrupt.The interrupt is a process of data transfer whereby an external device or a peripheral can inform the processor that it is ready for communication and it requests attention. The response to an interrupt request is directed or controlled by the microprocessor.

Process of interrupt Operation

From the point of view of I/O **unit**

- I/O device receives command from CPU
- The I/O device then processes the operation
- The I/O device signals an interrupt to the CPU over a control line.
- The I/O device waits until the request from CPU.

From the point of view of processor

- The CPU issues command and then goes off to do its work.
- When the interrupt from I/O device occurs, the processor saves its program counter & registers of the current program and processes the interrupt.
- After completion for interrupt, processor requires its initial task.

Interrupt structures:

A processor is usually provided with one or more interrupt pins on the chip. Therefore a special mechanism is necessary to handle interrupts from several devices that share one of these interrupt lines. There are mainly two ways of servicing multiple interrupts which are polled interrupts and daisy chain

(vectored) interrupts.

1. **polled interrupts:**

Polled interrupts are handled by using software which is slower than hardware interrupts. Here the processor has the general (common) interrupt service routine (ISR) for all devices. The priority of the devices is determined by the order in which the routine polls each device. The processor checks the starting with the highest priority device. Once it determines the source of the interrupt, it branches to the service routine for that device.

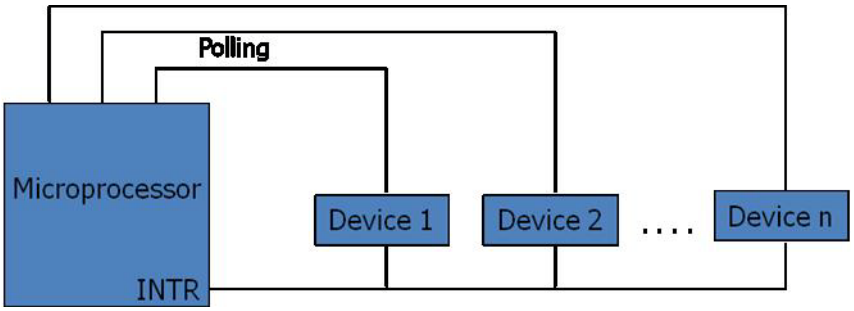


Fig: Polled Interrupt

Here several eternal devices are connected to a single interrupt line (INTR) of the microprocessor. When INTR signal goes up, the processor saves the contents of PC and other registers and then branches to an address defined by the manufactures of the processor. The user can write a program at this address to find the source of the interrupt by starting the polled from highest priority device.

2. Daisy chain (vectored) interrupt:

This is 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.

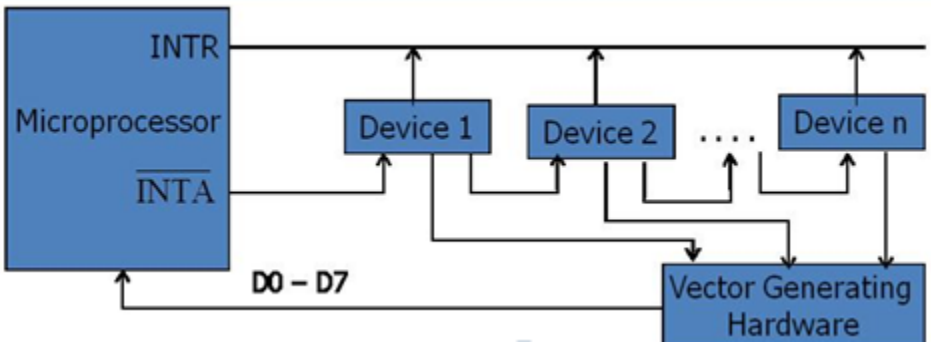


Fig: Vectored (Daisy Chain) Interrupt

Here the device with he highest priority 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 on to the next device until the INTA is accepted by the interrupting device. Once accepted, the device provides a means to the processor for finding the interrupt address vector using external hardware. Usually the 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 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.

Basic Interrupt Processing

The occurrence of interrupt triggers a number of events, both in processor hardware and in software. The interrupt driven I/O operation takes the following steps.

- The I/O unit issues an interrupt signal to the processor for exchange of data between them.
- The processor finishes execution of the current instruction before responding to the interrupt.
- The processor sends an acknowledgement signal to the device that it issued the interrupt.
- The processor transfers its control to the requested routine called ―Interrupt Service Routine (ISR) by saving the contents of program status word (PSW) and program counter (PC).
- The processor now loads the PC with the location of interrupt service routine and the fetches the instructions. The result is transferred to the interrupt handler program.
- When interrupt processing is completed, the saved register‘s value are retrieved from the stack and restored to the register.
- Finally it restores the PSW and PC values from the stack.

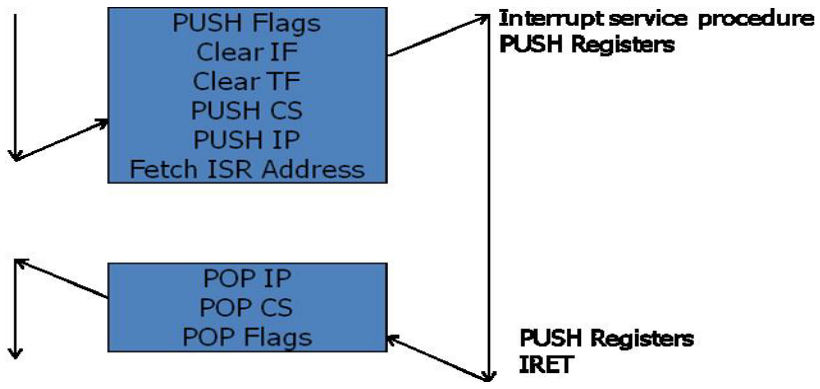


Fig: Interrupt Response for 8086 Microprocessor

The figure summarizes these steps. The processor pushes the flag register on the stack, disables the INTR input and does essentially an indirect call to the interrupt service procedure. An IRET function at the end

of interrupt service procedure returns execution to the main program

Interrupt priority:

Microcomputers can transfer data to or from an external devices using interrupt through INTR pin. When device wants to communicate with the microcomputer, it connects to INTR pin and makes it high or low depending on microcomputer. The microcomputer responds by sending signal via its pin called interrupt acknowledgement INTA. In differentiation with the occurrence of interrupts, basically following interrupts exist.

1. **External interrupts:**

These interrupts are initiated by external devices such as A/D converters and classified on following types.

- Maskable interrupt :

  It can be enabled or disabled by executing instructions such as EI and DI. In 8085, EI sets the interrupt enable flip flop and enables the interrupt process. DI resets the interrupt enable flip flop and disables the interrupt.

- Non-maskable interrupt:

  It has higher priority over maskable interrupt and cannot be enabled or disabled by the instructions.

2. Internal interrupts:

- These are indicated internally by exceptional conditions such as overflow, divide by zero, and execution of illegal op-code. The user usually writes a service routine to take correction measures and to provide an indication in order to inform the user that exceptional condition has occurred.
- There can also be activated by execution of TRAP instruction. This interrupt means TRAP is useful for operating the microprocessor in single step mode and hence important in debugging.
- These interrupts are used by using software to call the function of an operating system. Software interrupts are shorter than subroutine calls and they do not need the calling program to know the operating system‘s address in memory.

If the processor gets multiple interrupts, then we need to deal these interrupts one at a time and the dealing approaches are:

a. **Sequential processing of interrupts**

When user program is executing and an interrupt occurs interrupts are disabled immediately. After the interrupt service routine completes, interrupts are enabled before resuming the user program and the processor checks to see if additional interrupts have occurred

.

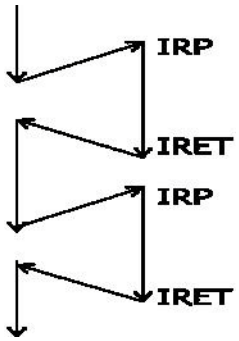


Fig: Sequential Interrupt Service

b. Priority wise processing of interrupts:

The drawback of sequential processing is that it does not take account of relative priority or time critical needs. The alternative form of this is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower priority interrupts pause until high priority interrupt completes its function.

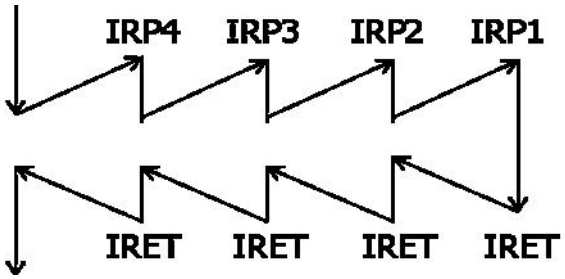


Fig: Priority wise Interrupt service

Interrupt Service Routine

- An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt.
- ISRs examine an interrupt and determine how to handle it.
- ISRs handle the interrupt, and then return a logical interrupt value.
- Its central purpose is to process the interrupt and then return control to the main program.
- An ISR must perform very fast to avoid slowing down the operation of the device and the operation of all lower priority ISRs.
- As in procedures, the last instruction in an ISR should be iret.

ISR is responsible for doing the following things:

1. Saving the processor context

   Because the ISR and main program use the same processor registers, it is the responsibility of the ISR to save the processor‘s registers before beginning any processing of the interrupt. The processor context consists of the instruction pointer, registers, and any flags. Some processors perform this step automatically.

2. Acknowledging the interrupt

   The ISR must clear the existing interrupt, which is done either in the peripheral that generated the interrupt, in the interrupt controller, or both.

3. Restoring the processor context

   After interrupt processing, in order to resume the main program, the values that were saved prior to the ISR execution must be restored. Some processors perform this step automatically.

Interrupt Processing in 8085

- Interrupt is signals send by an external device to the processor, to request 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.

- The processor will check the interrupts always at the 2nd T-state of last machine cycle.
- If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheralThe vectored address of particular interrupt is stored in program counter.
- The processor executes an interrupt service routine (ISR) addressed in program counter.
- It returned to main program by RET instruction.

Types of Interrupts:

It supports two types of interrupts.

1. Hardware
2. Software

Software interrupts:

The software interrupts are program instructions. These instructions are inserted at desired locations in a program.

The 8085 has eight software interrupts from RST 0 to RST 7. The vector address for these interrupts can be calculated as follows.

Interrupt number * 8 = vector address
For RST 5; 5 * 8 = 40 = 28H
Vector address for interrupt RST 5 is 0028H

The Table shows the vector addresses of all interrupts.

Interrupt	Vector address
RST 0	0000_H
RST 1	0008_H
RST 2	0010_H
RST 3	0018_H
RST 4	0020_H
RST 5	0028_H
RST 6	0030_H
RST 7	0038_H

Interrupt Pins and Priorities (Hardware interrupts)

An external device initiates the hardware interrupts and placing an appropriate signal at the interrupt pin of the processor.

If the interrupt is accepted then the processor executes an interrupt service routine.

The 8085 has five hardware interrupts

(1) TRAP (2) RST 7.5 (3) RST 6.5 (4) RST 5.5(5) INTR

Interrupt type	Trigger	Priority	Maskable	Vector address
TRAP	Edge and Level	1st	No	0024H
RST 7.5	Edge	2nd	Yes	003CH
RST 6.5	Level	3rd	Yes	0034H
RST 5.5	Level	4th	Yes	002CH
INTR	Level	5th	Yes	-

TRAP:

- This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
- TRAP bas the highest priority and vectored interrupt.
- TRAP interrupt is edge and level triggered. This means hat the TRAP must go high and remain high until it is acknowledged.
- In sudden power failure, it executes a ISR and send the data from main memory to backup memory.
- The signal, which overrides the TRAP, is HOLD signal. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).
- There are two ways to clear TRAP interrupt.
  1. By resetting microprocessor (External signal)
  2. By giving a high TRAP ACKNOWLEDGE (Internal signal)

RST 7.5:

- The RST 7.5 interrupt is a maskable interrupt.
- It has the second highest priority.
- It is edge sensitive. ie. Input goes to high and no need to maintain high state until it recognized.
- Maskable interrupt. It is disabled by,
  1. DI instruction
  2. System or processor reset.
  3. After reorganization of interrupt.
- Enabled by EI instruction.

RST 6.5 and 5.5:

- The RST 6.5 and RST 5.5 both are level triggered. . ie. Input goes to high and stay high until it recognized.
- Maskable interrupt. It is disabled by,
  1. DI, SIM instruction
  2. System or processor reset.
  3. After reorganization of interrupt.
- Enabled by EI instruction.
- The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.

INTR:

- INTR is a maskable interrupt.
- It is disabled by,
  1. DI, SIM instruction
  2. System or processor reset.
  3. After reorganization of interrupt.
- Enabled by EI instruction.
- Non- vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.

- It has lowest priority.
- It is a level sensitive interrupts. ie. Input goes to high and it is necessary to maintain high state until it recognized.
- The following sequence of events occurs when INTR signal goes high.
    1. The 8085 checks the status of INTR signal during execution of each instruction.
    2. If INTR signal is high, then 8085 complete its current instruction and sends active Low interrupt acknowledge signal, if the interrupt is enabled.
    3. In response to the acknowledge signal, external logic places an instruction OPCODE on the data bus. In the case of multi byte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.
    4. On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

THE 8259A 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.

BLOCK DIAGRAM OF THE 8259A

Figure 15.29 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 these blocks are given below:

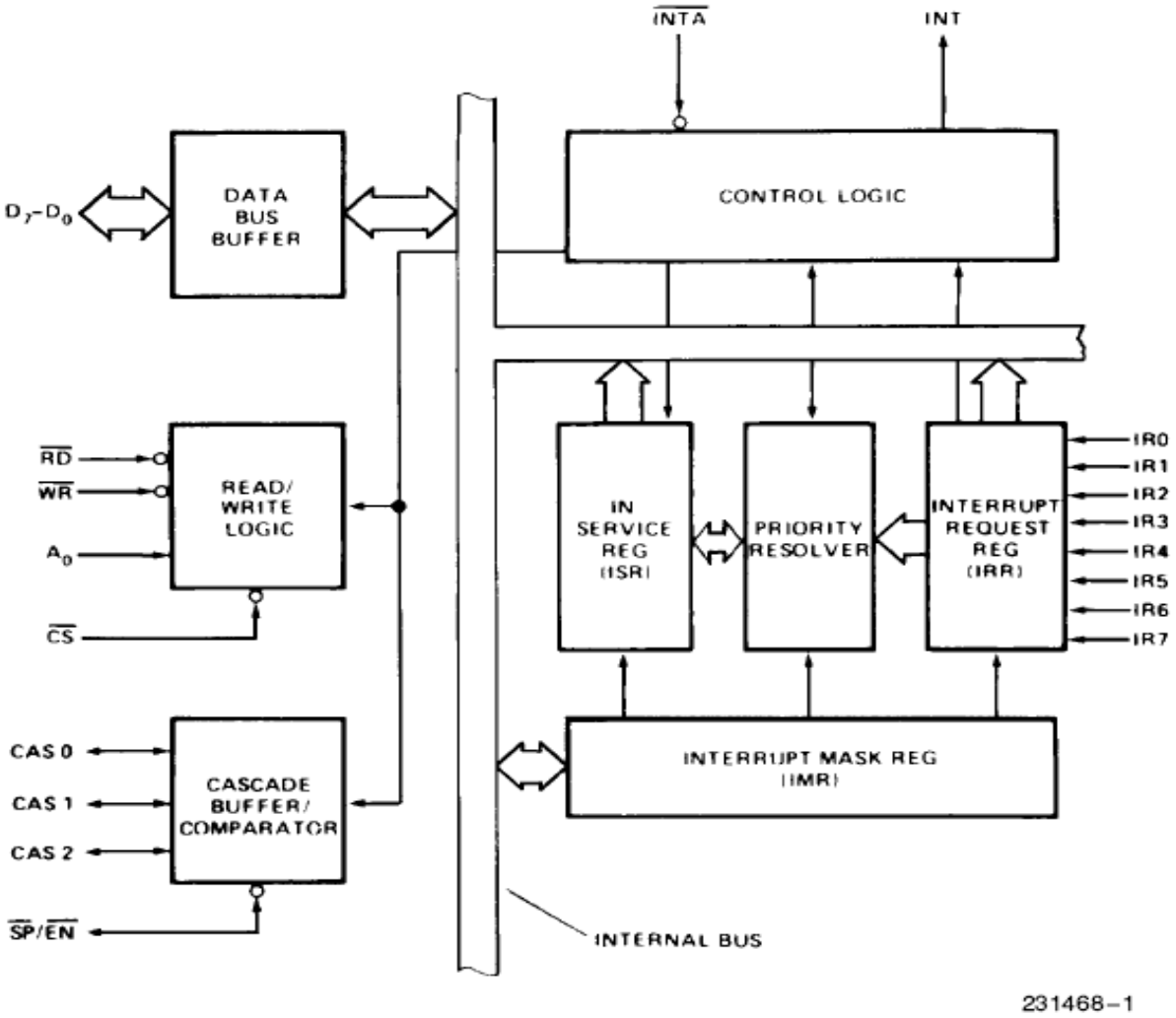


Figure 1. Block Diagram

DATA BUS BUFFER If the data format is not matched it temporarily holds the data. For eg: Printer can accept only 1 bit data but at a time more than 1 bit data may be sent. In such case data bus buffer stores the bits that printer cannot accept at the moment.

READ/WRITE LOGIC This is a typical Read/Write control logic. When the address line A0 is at logic 0, the controller is selected to write a command or read a status. The Chip Select logic and A0 determine the port address of the controller.

CONTROL LOGIC

This block has two pints: INT (Interrupt) as an output, and $\overline{INT}$ (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.

INTERRUPT REGISTERS AND PRIORITY RESOLVER

The interrupt Request Register (IRR) has eight input lines (IR0-IR7) 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) stored 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. To simplify this discussion, this block will not be mentioned again.

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 occurs 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.

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 IR0 as the highest and IR7 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, IR4 has the highest priority, and IR3 has the lowest priority:

IR0	IR1	IR2	IR3	IR4	IR2	IR3	IR4
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 IR2 has just been serviced, it will receive the seventh priority, as shown below:

IR0	IR1	IR2	IR3	IR4	IR2	IR3	IR4
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 lowe

UNIT 7

INPUT/OUTPUT INTERFACES

Input/ Output Devices

Input / Output devices are the means through which the microcomputer unit communicates with the outside world. The link between the I/O devices and the microprocessor is maintained by a circuitry known as I/O module. This circuitry includes the specific interfaces needed for I/O devices as well as control functions that implement the I/O transfers within the computer. I/O devices usually are appeared as passive devices which take action only when instructed to do. The CPU monitors the status of the I/O devices and selects them according to availability and need.
Consider the keyboard as input device and the steps when the key is pressed are

- Microprocessor detects the key change in status of keyboard i.e. the key is pressed.
- It receives the encoded information corresponding to pressed key.
- It checks the validity of required signal.

Consider the printer as output device

- Here microprocessor checks ideal condition of printer, if ideal then sends the data to be printed and required command for that.

For interfacing of typical microprocessor to I/O devices such as keyboard, CRT, printer etc. All need I/O interface circuits which are of mainly two types.

Parallel Interface

- The device which can handle data at higher speed cannot support with serial interface.
- N bits of data are handled simultaneously by the bus and the links to the device directly.
- Achieves faster communication but becomes expensive due to need of multiple wires.

Data transfer Modes of Parallel Interfacing

The information exchanged between a microprocessor and an I/O interface circuit consists of input or output data and control information. The status information enable the microprocessor monitor the device and when it is ready then send or receive data. Control information is the command by microprocessor to cause I/O device to take some action. If the device operates at different speeds, then microprocessor can be used to select a particular speed of operation of the device. The techniques used to transfer data between different speed devices and computer is called synchronizing. Different techniques under synchronizing are:

1) **Simple I/O**

For simple I/O, the buffer switch and latch switches i. e. LED are always connected to the input and output ports. The devices are always ready to send or receive data.

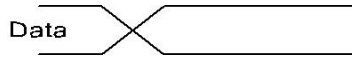


Fig: Simple I /O

Here cross line indicate the time for new valid data.

2) **Wait Interface(Simple strobe I/O)**

In this technique, MP need to wait until the device is ready for the operation.

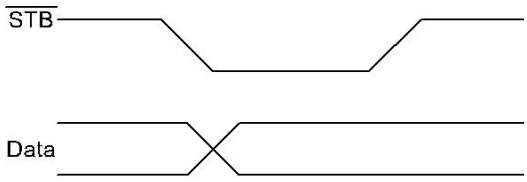


Fig: Simple Strobe I/O

Consider a simple keyboard consisting of 8 switches connected to a MP through a parallel interface CKT (Tri-state buffer). The switch is of dip switches. In order to use this keyboard as an input device the MP should be able to detect that a key has been activated. This can be done by observing that all the bits are in required order. The processor should repeatedly read the state of input port until it finds the right order of bits i.e. at least 1 bit of 8 bits should be 0.

Consider the tri-state A/D converter

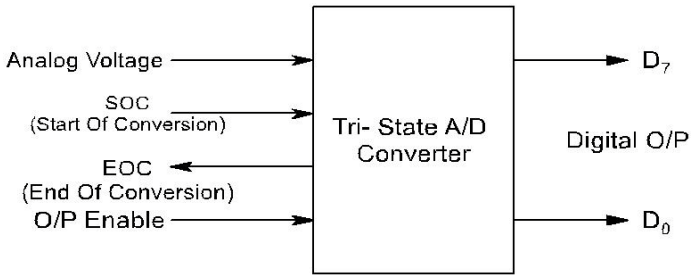


Fig: Tri-State A/D Converter

- Used to convert analog to digital data which can be read by I/O unit of MP
- When SOC appears 1, I/O unit should ready for reading binary data/digital data.
- When EOC's status is 1, then I/O unit should stop to read data.
- Strobe signal indicates the time at which data is being activated to transmit.

3) <u>Single Handshaking:</u>

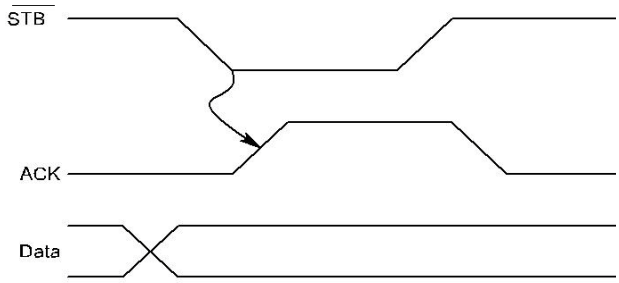


Fig: Single Handshaking

- The peripheral outputs some data and send $\overline{STB}$ signal to MP. ―here is the data for you‖
- MP detects asserted $\overline{STB}$ signal, reads the data and sends an acknowledge signal (ACK) to indicate data has been read and peripheral can send next data. ―I got that one, send me another‖
- MP sends or receives data when peripheral is ready.

4) <u>Double Handshaking</u>

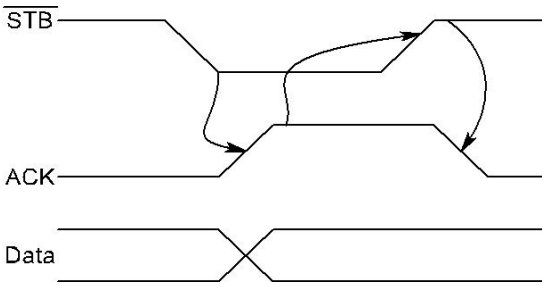


Fig: Double Handshaking

- The peripheral asserts its $\overline{STB}$ line low to ask MP ―Are you ready?‖
- The MP raises its ACK line high to say ― I am ready‖.
- Peripheral then sends data and raises its $\overline{STB}$ line low to say ―Here is some valid data for you‖
- MP then reads the data and drops its ACK line to say, ―I have the data, thank you, and I await your request to send the next byte of data.

Serial Interface /Serial data transmission:

- Data are transferred serially one bit at a time starting from Least Significant bit.
- Slow due to single communication link but inexpensive to implement.
- It uses clock to separate consecutive bits.
- Its function is to deal with the data on the bus in the parallel mode and communicate with the connected device in serial mode.
- Its data bus has n data lines, the serial I/O interface accepts n bit of data simultaneously from the bus and n bits are sent one at a time thus requiring **n** time slots.
- Not suitable for fast operation needed microprocessor.

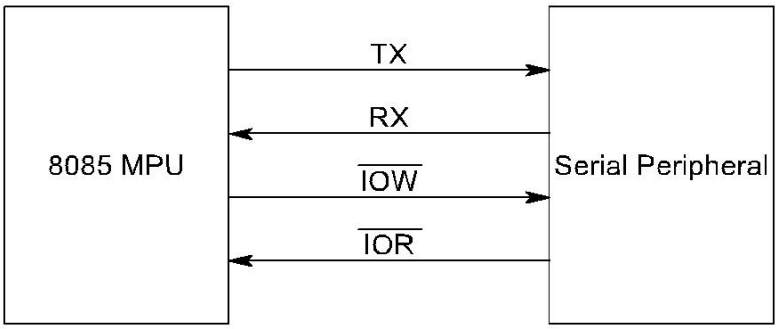


Fig: Serial Interface between microprocessor and I/O device.

Serial I/O transfer is more common than the parallel I/O. The two major forms of serial data transmission are:

1. **Synchronous serial data transmission:**

- Data is transmitted or received based on a clock signal i. e. synchronously.
- The transmitting device sends a data bit at each clock pulse.
- Usually one or more SYNC characters are used to indicate the start of each synchronous data stream.
- SYNC characters for each frame of data.
- Transmitting device sends data continuously to the receiving device. If the data is not ready to be transmitted, the transmitter will send SYNC character until the data is available.
- The receiving device waits for data, when it finds the SYNC characters then starts interpreting the data.

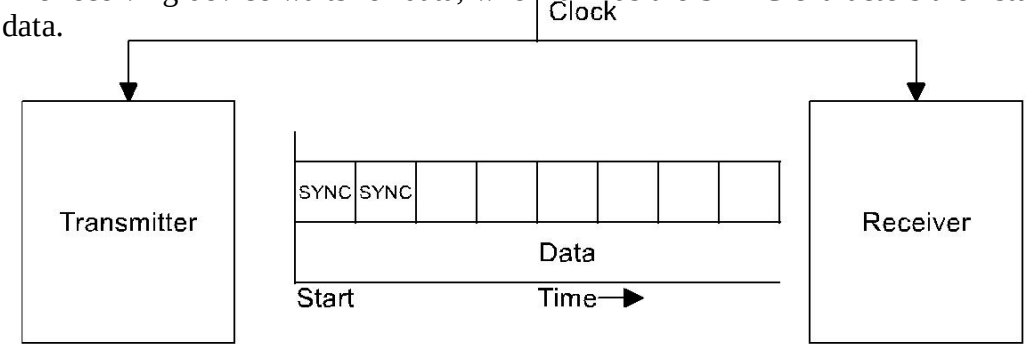


Fig: Synchronous Serial Transmission Format

2) Asynchronous serial data Transmission:

- The receiving device does not need to be synchronized with the transmitting device.
- Transmitting device send data units when it is ready to send data.
- Each data unit must contain start and stop bits for indicating beginning and the end of data unit. And also one parity bit to identify odd or even parity data.
- For e.g. To send ASCII character (7 bit)

We need: 1 start bit: beginning of data
1 stop bit: End of data
1 Parity bit: even or odd parity
7 or 8 bit character: actual data transferred

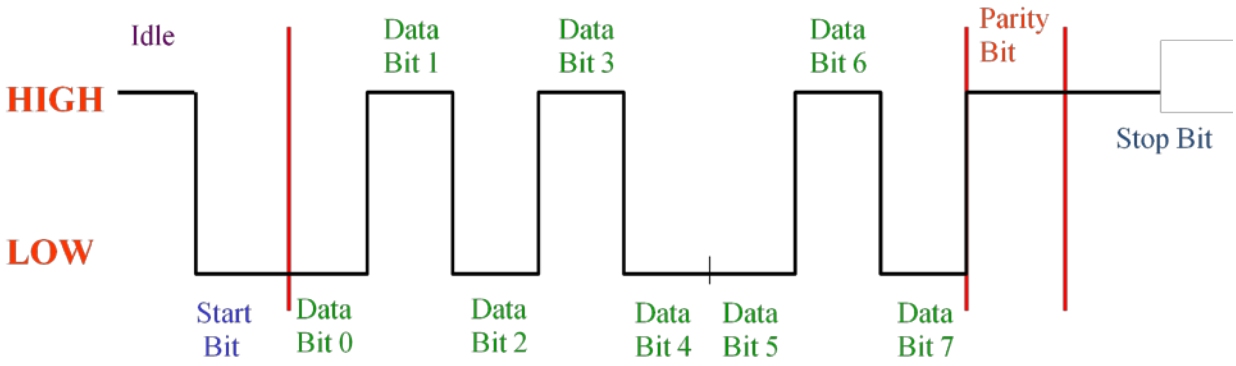


Fig : Data Format

Synchronous vs. Asynchronous Data Transmission

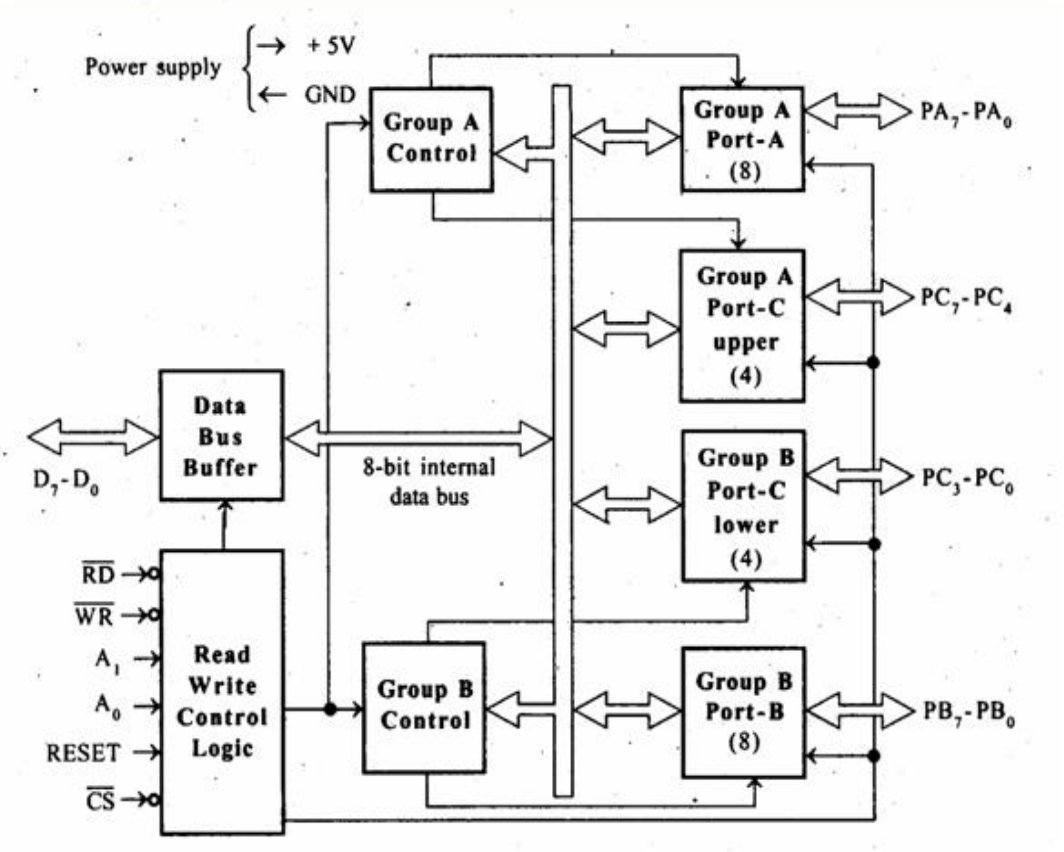
S.N.	Parameter	Asynchronous	Synchronous
1.	Fundamental	Transmission does not based on clock signal	Transmission based on clock signal
2.	Data Format	One character at a time	Group of characters i.e. a block of characters
3.	Speed	Low (< 20 kbps)	High (> 20 kbps)
4.	Framing Information	Start and stop bits are sent with each character.	SYNC characters are sent with each character.
5.	Implementation	Hardware / Software	Hardware

Programmable Peripheral Interface (PPI) - 8255A:

- The INTEL 8255 is a device used to parallel data transfer between processor and slow peripheral devices like ADC, DAC, keyboard, 7-segment display, LCD, etc.
- The 8255 has three ports: Port-A, Port-B and Port-C.

- Port-A can be programmed to work in any one of the three operating modes mode-0, mode-1 and mode-2 as input or output port.
- Port-B can be programmed to work either in mode-0 or mode-1 as input or output port.
- Port-C (8-pins) has different assignments depending on the mode of port-A and port-B.
- If port-A and B are programmed in mode-0, then the port-C can perform any one of the following functions.

    - As 8-bit parallel port in mode-0 for input or output.
    - As two numbers of 4-bit parallel ports in mode-0 for input or output.
    - The individual pins of port-C can be set or reset for various control applications.
- If port-A is programmed in mode- 1/mode-2 and port-B is programmed in mode-1 then some of the pins of port-C are used for handshake signals and the remaining pins can be used as input/ output lines or individually set/reset for control application.



Key Features of Mode-0,Mode-1 and Mode-2

- *Mode 0: Ports A and B operate as either inputs or outputs and Port C is divided into two 4-bit groups either of which can be operated as inputs or outputs*
- *Mode 1: Same as Mode 0 but Port C is used for handshaking and control*
- *Mode 2: Port A is bidirectional (both input and output) and Port C is used for handshaking. Port B is not used.*

The read/write control logic requires six control signals. These signals are given below.

1. RD (low): This control signal enables the read operation. When this signal is low, the microprocessor reads data from a selected I/O port of the 8255A.
2. WR (low): This control signal enables the write operation. When this signal goes low, the microprocessor writes into a selected I/O port or the control register.
3. RESET: This is an active high signal. It clears the control register and set all ports in the input mode.
4. CS (low), A0 and A1: These are device select signals. They are,

Internal Devices	A_1	A_0
Port A	0	0
Port B	0	1
Port C	1	0
Control Register	1	1

8255A OPERATIONAL MODES AND INITIALIZATION 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

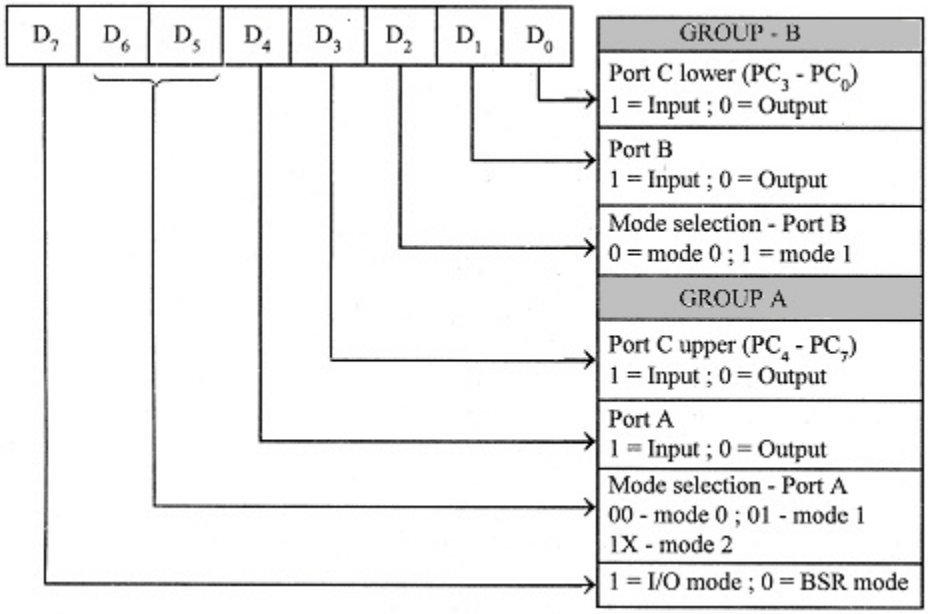
3. MODE 2

- 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.

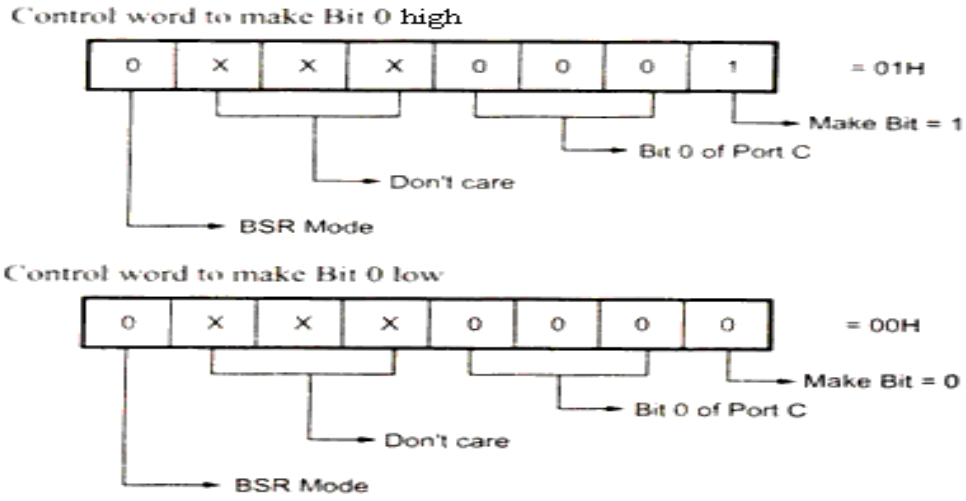
- If PORT A is initialized in MODE 2, then pins PC3 through PC7 are used as handshake lines for PORT A.
- The other three pins PC0 through PC2 can be used for I/O port if PORT B is in MODE 0.
- The same 3 pins will be used for PORT B handshake signals if PORT B is initialized in MODE 1.

CONTROL WORDS

MODE SET CONTROL WORD



BIT SET/RESET CONTROL WORD



Universal Synchronous Asynchronous Receiver Transmitter (USART) – 8251A

The 8251A is a programmable serial communication interface chip designed for synchronous and asynchronous serial data communication. As a peripheral device of a microcomputer system, it 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.

Features of 8251A

- Wide power supply voltage range from 3V to 6V
- Wide temperature range from -40^o C to 85^o C
- Synchronous communication up to 64 Kbaud
- Asynchronous communication up to 38.4 Kbaud
- Transmitting / Receiving operations under double buffered configuration.
- Error detection capability (Parity, Overrun and Framing)

The functional block diagram of 8251A consists five sections. They are:

- Read/Write control logic
- Transmitter
- Receiver
- Data bus buffer
- Modem control.

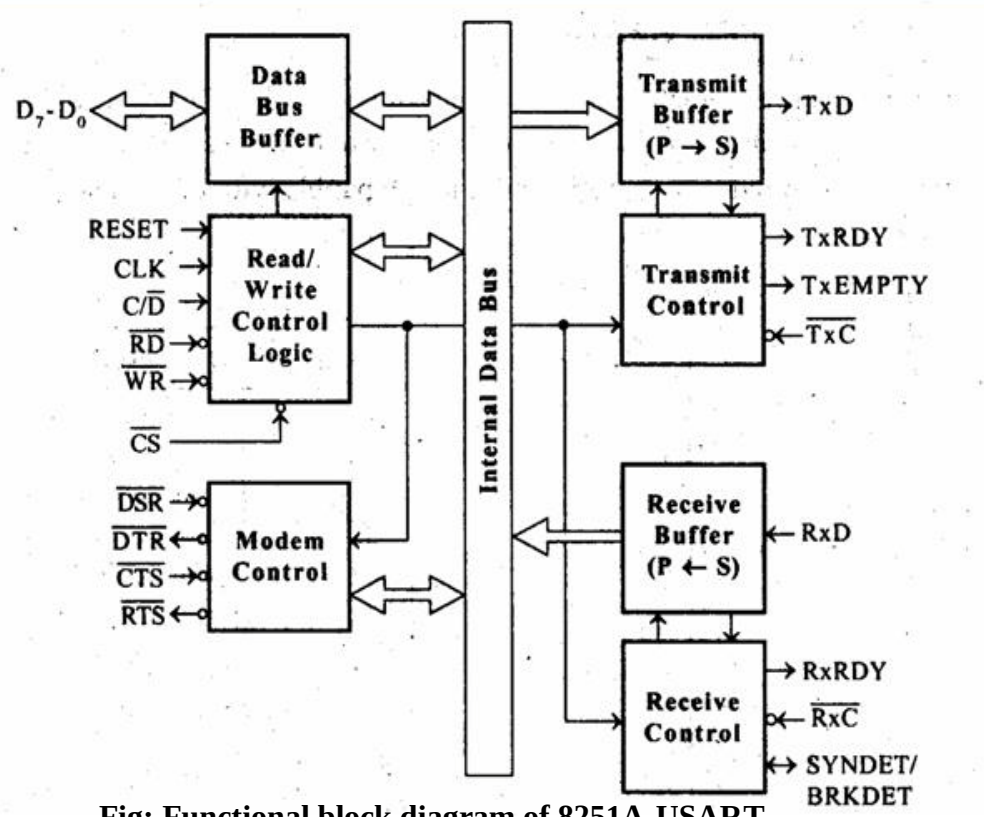


Fig: Functional block diagram of 8251A-USART

Read/Write control logic:

- The Read/Write Control logic interfaces the 8251A with CPU, determines the functions of the 8251A according to the control word written into its control register.
- It monitors the data flow.
- This section has three registers and they are control register, status register and data buffer.
- The active low signals RD, WR, CS and C/D(Low) are used for read/write

operations with these three registers.

- When C/D(low) is high, the control register is selected for writing control word or reading status word.
- When C/D(low) is low, the data buffer is selected for read/write operation.
- When the reset is high, it forces 8251A into the idle mode.
- The clock input is necessary for 8251A for communication with CPU and this clock does not control either the serial transmission or the reception rate.

Transmitter section:

- The transmitter section accepts parallel data from CPU and converts them into serial data.
- The transmitter section is double buffered, i.e., it has a buffer register to hold an 8-bit parallel data and another register called output register to convert the parallel data into serial bits.
- When output register is empty, the data is transferred from buffer to output register. Now the processor can again load another data in buffer register.
- If buffer register is empty, then TxRDY is goes to high.
- If output register is empty then TxEMPTY goes to high.
- The clock signal, TxC (low) controls the rate at which the bits are transmitted by the USART.
- The clock frequency can be 1, 16 or 64 times the baud rate.

Receiver Section:

- The receiver section accepts serial data and converts them into parallel data.
- The receiver section is double buffered, i.e., it has an input register to receive serial data and convert to parallel, and a buffer register to hold the parallel data.
- When the RxD line goes low, the control logic assumes it as a START bit, waits for half a bit time and samples the line again.
- If the line is still low, then the input register accepts the following bits, forms a character and loads it into the buffer register.
- The CPU reads the parallel data from the buffer register.
- When the input register loads a parallel data to buffer register, the RxRDY line goes high.
- The clock signal RxC (low) controls the rate at which bits are received by the USART.
- During asynchronous mode, the signal SYNDET/BRKDET will indicate the break in the data transmission.

During synchronous mode, the signal SYNDET/BRKDET will indicate the reception of synchronous character

MODEM Control:

- The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines.
- This unit takes care of handshake signals for MODEM interface.

Baud rate /Bit rate

The difference between Bit and Baud rate is complicated and intertwining. Both are

dependent and inter-related.

Bit Rate is how many data bits are transmitted per second.

A baud Rate is the number of times per second a signal in a communications channel changes.Bit rates measure the number of data bits (that's 0s and 1's) transmitted in one second in a communication channel. A figure of 2400 bits per second means 2400 zeros or ones can be transmitted in one second, hence the abbreviation —bps. Individual characters (for example letters or numbers) that are also referred to as bytes are composed of several bits.

A baud rate is the number of times a signal in a communications channel changes state or varies. For example, a 2400 baud rate means that the channel can change states up to 2400 times per second. The term —change state‖ means that it can change from 0 to 1 or from 1 to 0 up to X (in this case, 2400) times per second. It also refers to the actual state of the connection, such as voltage, frequency, or phase level).

The main difference between the two is that one change of state can transmit one bit, or slightly more or less than one bit, that depends on the modulation technique used. So the bit rate (bps) and baud rate (baud per second) have this connection:

If signal is changing every 10/3 ns then,

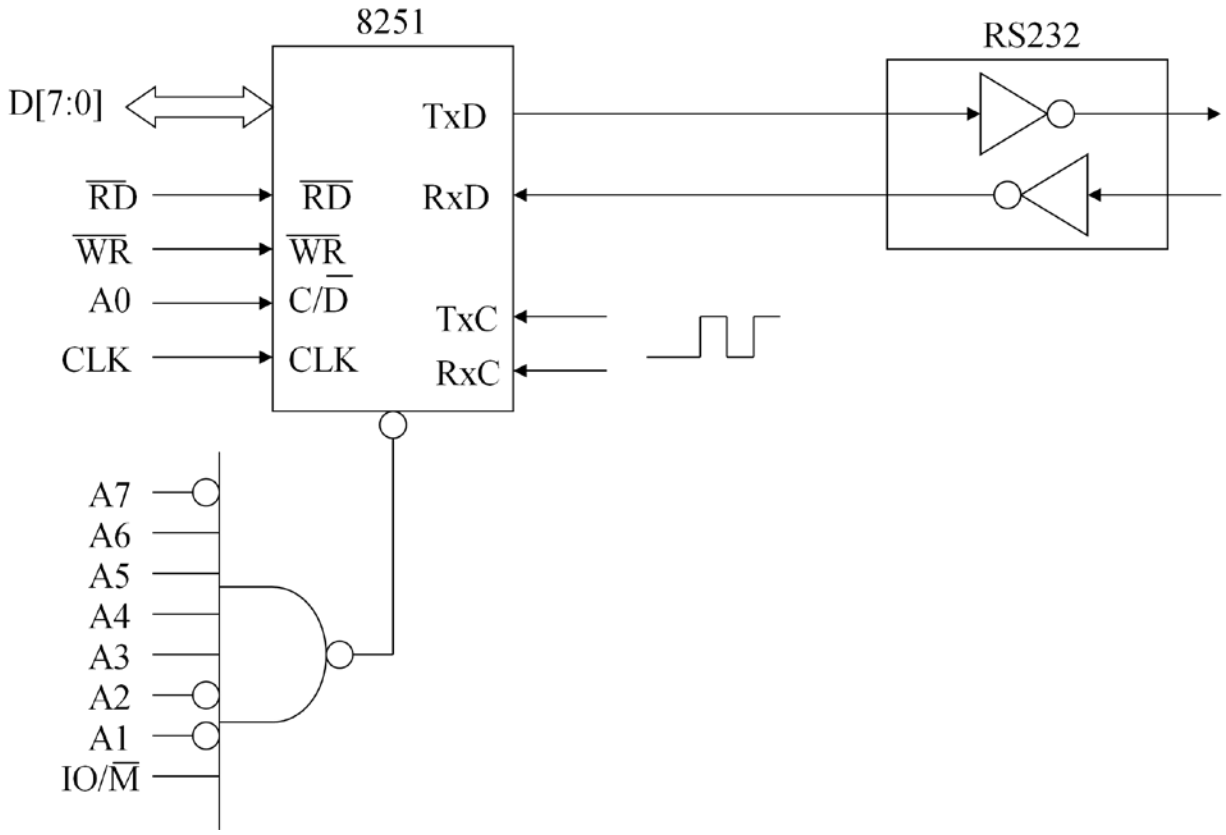
Baud rate = 1/10/3ns $= 3/10*10^9 = 3*10^8$

= 300 MBd

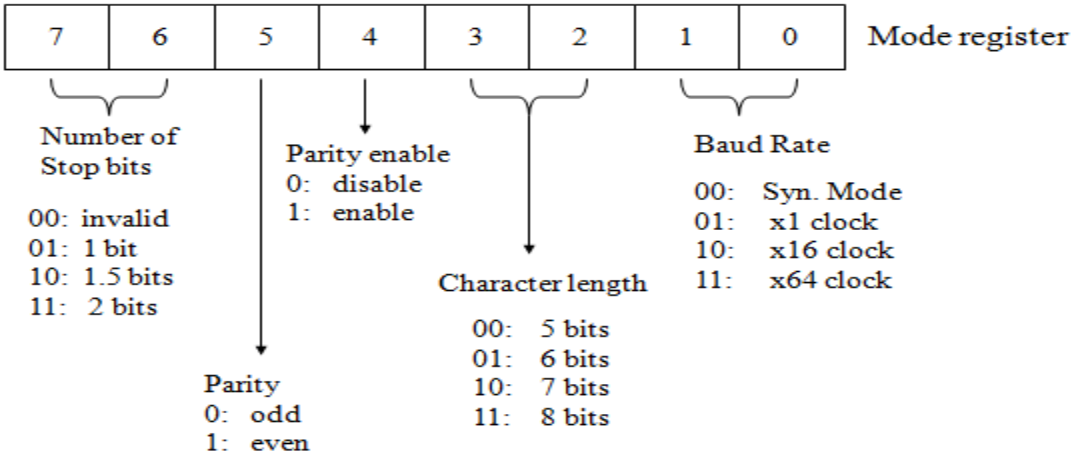
Note:

- If 1 frame of data is coded with 1 bit then band rate and bit rate are same.
- Sometimes frame of data are coded with two of three bits then baud rate and bit rate are not same.

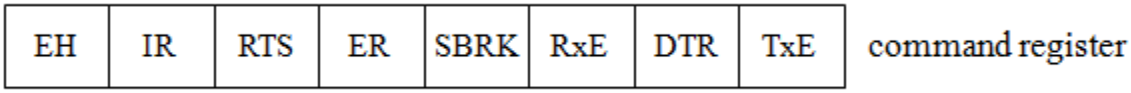
8251 USART Interface



8251 mode register



8251 command register



TxE:	transmit enable
DTR:	data terminal ready, $\overline{DTR}$ pin will be low
RxE:	receiver enable
SBPRK:	send break character, TxD pin will be low
ER:	error reset
RTS:	request to send, $\overline{CTS}$ pin will be low
IR:	internal reset
EH:	enter hunt mode

8251 status register

DSR	SYNDET	FE	OE	PE	TxEMPTY	RxRDY	TxRDY

status register

TxRDY:	transmit ready
RxRDY:	receiver ready
TxEMPTY:	transmitter empty
PE:	parity error
OE:	overrun error
FE:	framing error
SYNDET:	sync. character detected
DSR:	data set ready

RS 232C

This interface standard is most widely used standard for serial communication between microcomputers and peripheral devices. The interface, defined by EIA, relates essentially to two types of equipment. The first is known as data terminal equipment. While the second is referred as data communication equipment (DTE). The data terminal equipment (e.g a microcomputer) is capable of sending and/or receiving data via the serial interface. The data communication equipment on the other hand is generally through of as a device which can facilitate serial data communications (e.g modems).

- RS 232C works in a negative logic. The standard specifies that 'the logic one' level is a voltage between -3 and -15 v and 'the logic zero' is a voltage between +3 and +15 V. The commonly used voltages are +12v and -12V.
- The transmission line normally used a twisted pair of shielded wire with a line capacitance of more than 1200PF and no less than 300Pf. The standard specifies the line length to 50 feets only.
- The standard describes the function of 25 signal and hansake pins for serial data transfer. It also specifies that the DTE connector should be male and the DCE connector should be female. Usually 9 pin and 25 pin connectors are available.

Among the 25 pins of RS232C the independent pin numbers are:

Pin no. Signals Functions

2➔ Transmit data, TXD Output , transmit data from DTE to DCE

3➔ Receive data RXD Input, DTE receive data from DCE

4➔ Request to send, RTS General Purpose output from DTE
5➔ Clear to send, CTS Input to DTE, used as handshake
6➔ Data set ready, DSR Input to DTE, indicate that DCE is ready
7➔ Signal ground, GND Common reference bet. DTE and DCE
8➔ Data carrier detect, DCD Used by DTE to disable data reception
20 Data terminal ready, DTR Output means DTE is ready. The main problem with –RS-232C is that it can only transfer data reliably about 50ft (16.4m) at its maximum rate of 20k band. If longer lines are used the transmission rate has to be drastically reduced. For higher rate of transfers and for longer distance we have another standard defined.

Standards Speed Distances Voltage Range
RS 232C 20k band 50ft ±15V RS-422A 10Mbaud at 40ft 4000ft ±7 100 Kbaud at 4000ft RS-485A 100K baud at 30ft 1 kbaud at 4000ft 4000ft ±12V

RS 232C interface with DTE and DCE: The figure below shows a interfacing with minimum lines

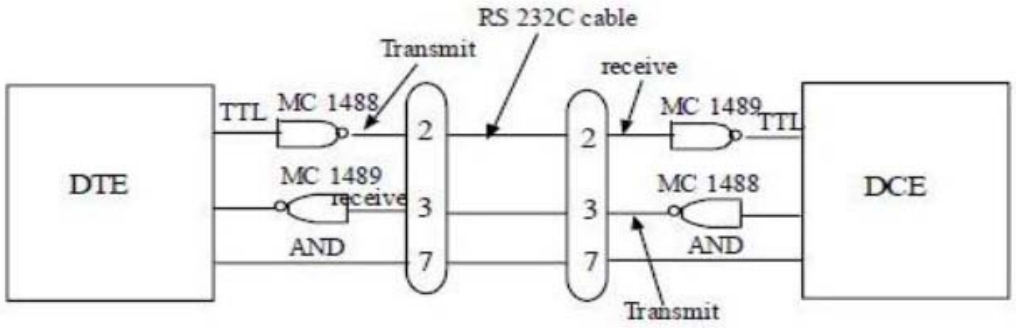
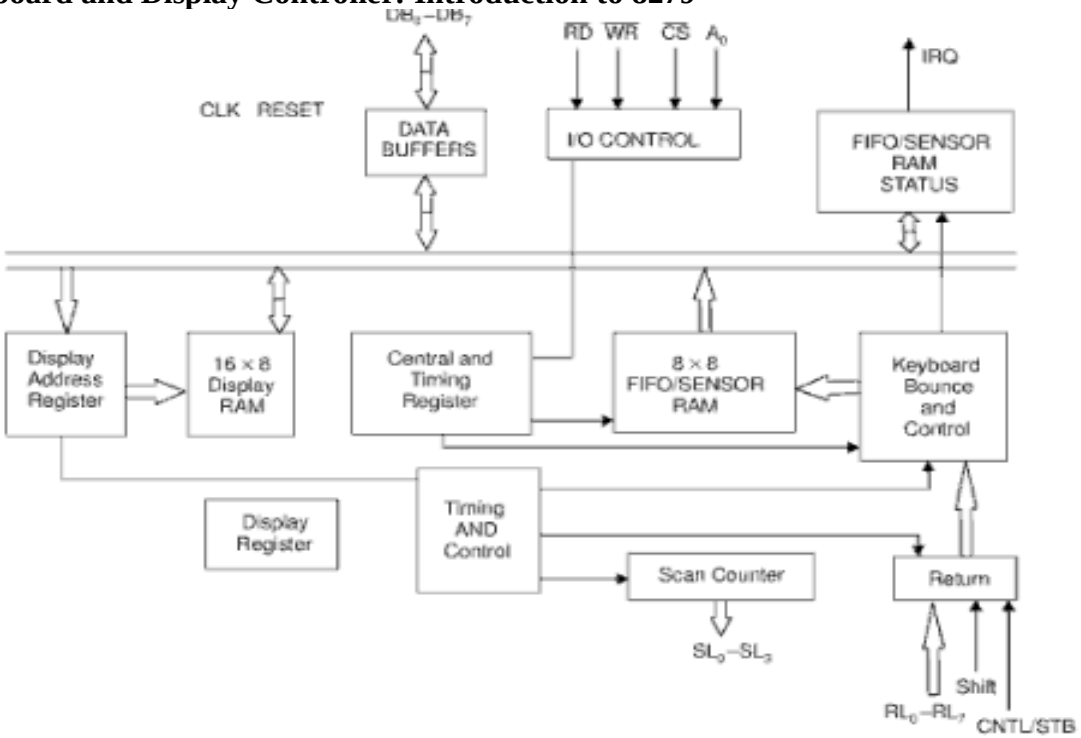


Fig. RS 232C interface

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.

Keyboard and Display Controller: Introduction to 8279



Keyboard Section: This section has eight lines (RL0 – RL7)that can be connected to 8 columns of a keyboard, plus two additional lines shift and CNTL/STB (control/strobe). The keys are automatically debouched and the keyboard can operate in two modes; two-key lockout or N-key rollover. In two-key lockout mode, if two keys are pressed almost simultaneously only the first key is recognized. In the N-key rollover mode, simultaneous keys are recognized and their codes are stored in the internal buffer. It can also be set up so that no key is recognized until only one key remains pressed. This section also includes 8 × 8 FIFO RAM, that store keyboard entries and provides IRQ (interrupt request). Signal when FIFO is not empty.

Scan Section: The scan section has scan counter and 4 scan lines (SL0 – SL3). These 4 scan lines can be decoded using a 4 to 16 decoder to generate 16 lines for scanning.

Display Section: The display section has eight output lines divided into two groups A0 – A3 and B0 –B3. These lines can be used, either as a group of eight lines or as two groups of four, in conjunction with the scan line, for a multiplexed display. The display can be blanked by using the BD line. This section includes 16 × 8 display RAM.

MPU Interface Section: This section includes eight bidirectional data lines (DB0 –DB7), one interrupt request (IRQ) line, and 6 lines for interfacing , including the buffer address line (A0) When A0 is high, signals are interpreted as control words or status; when A0 is low, signals are interpreted as data. The IRQ line goes high whenever data entries are stored in the FIFO indicating the availability of data.