

### Chapter – 1 Introduction

#### 1.1 Introduction

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

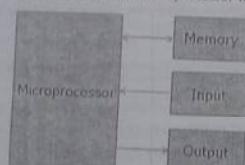


Fig 1.1: A Programmable Machine

#### Terms used

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

#### 1.2 Basic Block Diagram of a Computer

Traditionally, the computer is represented with four components such as memory, input, output and central processing unit (CPU) which consists of arithmetic logic unit (ALU) and control unit (CU).

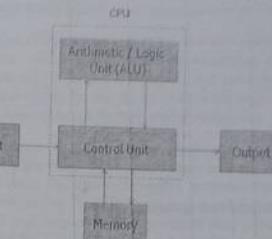


Fig 1.2 (a): Traditional Block diagram of a computer

The CPU contains various registers to store data, the ALU to perform arithmetic and logical operations, instruction decoders, counters and control lines.

The CPU reads instructions from memory and performs the tasks specified. It communicates with input/output (I/O) devices either to accept or to send data, the I/O devices is known as peripherals.

Later on around late 1960's, traditional block diagram can be replaced with computer having microprocessor as CPU which is known as microcomputer. Here CPU was designed using integrated circuit technology (IC's) which provided the possibility to build the CPU on a single chip.

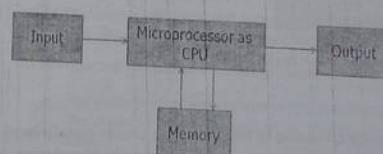


Fig 1.2 (b): Block Diagram of a computer with the Microprocessor as CPU

Later on semiconductor fabrication technology became more advanced, manufacturers were able to place not only MPU but also memory and I/O interfacing circuits on a single chip known as microcontroller, which also includes additional devices such as A/D converter, serial I/O, timer etc.

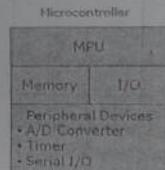


Fig 1.2 (c): Block Diagram of a Microcontroller

### 1.3 Organization of a microprocessor based system

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

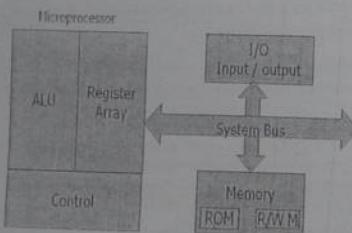


Fig 1.3: Microprocessor Based System with Bus Architecture

#### Microprocessor:

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

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

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

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

#### Memory:

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

- A. Read only Memory (ROM): Used to store programs that do not need alterations and can only read.
- B. Read/Write Memory (RAM): Also known as user memory which is used to store user programs and data. The information stored in this memory can be easily read and altered.

#### Input/Output:

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

#### System Bus:

It is a communication path between the microprocessor and peripherals; it is nothing but a group of wires to carry bits.

### 1.4 Bus organization

Bus is a common channel through which bits from any sources can be transferred to the destination. A typical digital computer has many registers and paths must be provided to transfer instructions from one register to another. The number of wires will be excessive if separate lines are used between each register and all other registers in the system. A more

efficient scheme for transferring information between registers in a multiple register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

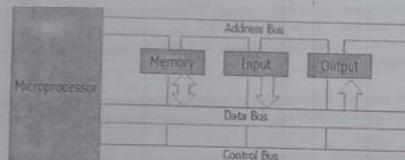


Fig: Bus Organization

A very easy way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then pleased on the bus.

A system bus consists of about 50 to 100 of separate lines each assigned a particular meaning or function. Although there are many different bus designers, on any bus, the lines can be classified into three functional groups; data, address and control lines. In addition, there may be power distribution lines as well.

- The data lines provide a path for moving data between system modules. These lines are collectively called data bus.
- The address lines are used to designate the source/destination of data on data bus.
- The control lines are used to control the access to and the use of the data and address lines. Because data and address lines are shared by all components, there must be a means of controlling their use. Control signals transmit both command and timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Control lines include memory read/write, I/O read/write, bus request/grant, clock, reset, interrupt request/acknowledge etc.

#### Historical Background of the Development of Computers:

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

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

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

#### Automated calculator:

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

Some important features of automated calculations:

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

### 1.5 Stored Program Concept and Von-Neumann Machine:

The simplest way to organize a computer is to have one processor, register and instruction code format with two parts op-code and address/operand. The memory address tells the control where to find an operand in memory. This operand is read from memory and used as data to be operated on together with the data stored in the processor register. Instructions are stored in one section of same memory. It is called stored program concept.

The task of entering and altering the programs for ENIAC was tedious. It could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data. So the computer could get its instructions by reading from the memory and program could be set or altered by setting the values of a portion of memory. This approach is known as 'stored-program concept' was first adopted by John Von Neumann and such architecture is named as von-Neumann architecture and shown in figure below.

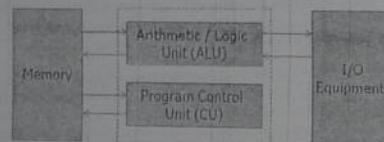


Fig: Von-Neumann Architecture

The main memory is used to store both data and instructions. The arithmetic and logic unit is capable of performing arithmetic and logical operation on binary data. The program control unit interprets the instruction in memory and causes them to be executed. The I/O unit gets operated from the control unit.

The Von-Neumann architecture is the fundamental basis for the architecture of modern digital computers. It consisted of 1000 storage locations which can hold words of 40 binary digits and both instructions as well as data are stored in it. The storage location of control unit and ALU are called registers and the various models of registers are:

MAR – memory address register – contains the address in memory of the word to be written into or read from MBR.

MBR – memory buffer register – consists of a word to be stored in or received from memory.

IR – instruction register – contains the 8-bit op code instruction to be executed.

IBR – instruction buffer register – used to temporarily hold the instruction from a word in memory.

PC – program counter - contains the address of the next instruction to be fetched from memory.

AC & MQ (Accumulator and Multiplier Quotient) - holds the operands and results of ALU after processing.

### Harvard Architecture

In von-Neumann architecture, the same memory is used for storing instructions and data. Similarly, a single bus called data bus or address bus is used for reading data and instructions from or writing to memory. It also had limited the processing speed for computers.

The Harvard architecture based computer consists of separate memory spaces for the programs (instructions) and data. Each space has its own address and data buses. So instructions and data can be fetched from memory concurrently and provides significant processing speed improvement.

In figure below, there are two data and two address buses multiplexed for data bus and address bus. Hence, there are two blocks of RAM chips one for program memory and another for data memory addresses.

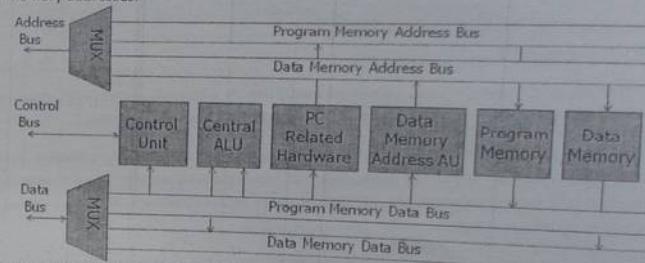


Fig: Harvard Architecture Based Microprocessor

The control unit controls the sequence of operations. Central ALU consists of ALU, multiplier, accumulator and scaling chief register. The PC used to address program memory and always contains the address of next instruction to be executed. Here data and control buses are bidirectional and address bus is unidirectional.

### Evolution of Microprocessors (Intel series)

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

## Microprocessors

## Chapter 1 : Introduction

| Date | Microprocessor  | Data bus   | Address Bus | Memory    |
|------|---|------------|-------------|-----------|
| 1971 | 4004  | 4-bit      | 10-bit      | 640 Bytes |
| 1972 | 8008  | 8-bit      | 14-bit      | 16k       |
| 1974 | 8080  | 8bit       | 16bit       | 64k       |
| 1976 | 8085  | 8bit       | 16bit       | 64k       |
| 1978 | 8086  | 16bit      | 20bit       | 1M        |
| 1979 | 8088  | 8bit       | 20bit       | 1M        |
| 1982 | 30286   | 16bit      | 24bit       | 16M       |
| 1985 | 80386   | 32bit      | 32bit       | 4G        |
| 1989 | 80486   | 32bit      | 32bit       | 4G        |
| 1993 | Pentium   | 32 / 64bit | 32bit       | 4G        |
| 1995 | Pentium pro   | 32 / 64bit | 36bit       | 64G       |
| 1997 | Pentium II  | 64bit      | 36bit       | 64G       |
| 1998 | Celeron   | 64bit      | 36bit       | 64G       |
| 1999 | Pentium III   | 64bit      | 36bit       | 64G       |
| 2000 | Pentium IV  | 64bit      | 36bit       | 64G       |
| 2001 | Itanium   | 128 bit    | 64bit       | 64G       |
| 2002 | Itanium 2   | 128 bit    | 64bit       | 64G       |
| 2003 | Pentium M/Centrino (wireless capability) for Mobile version e.g. Laptop |            |             |           |
|      | Core 2: X86 - 64 Architecture   |            |             |           |

### 1.6 Processing Cycle of a Stored Program Computer

- Fetch
- Identify
- Fetch Data
- Process
- Write Back

### 1.7 Microinstructions and Hardwired/Micropogrammed Control Unit

#### Micro-Operations

- A computer executes a program consisting instructions. Each instruction is made up of shorter sub-cycles as fetch, indirect, execute cycle, and interrupt.
- Performance of each cycle has a number of shorter operations called micro-operations.
- Called so because each step is very simple and does very little.
- Thus micro-operations are functional atomic operation of CPU.
- Hence events of any instruction cycle can be described as a sequence of micro-operations.

## Microprocessors

## Chapter 1 : Introduction

#### Microinstructions

Each instruction is characterized with many machine cycles and each cycle is characterized with many T-states. The lower instruction level patterns which are the numerous sequences for a single instruction are known as microinstructions. Suppose we can visualize the microinstruction with the help of fetch cycle, or read cycle or write cycle.

#### Fetch – Registers

- Memory Address Register (MAR)
  - Connected to address bus
  - Specifies address for read or write op
- Memory Buffer Register (MBR)
  - Connected to data bus
  - Holds data to write or last data read
- Program Counter (PC)
  - Holds address of next instruction to be fetched
- Instruction Register (IR)
  - Holds last instruction fetched

#### Fetch Sequence

- Address of next instruction is in PC
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory)
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches.

#### Fetch Sequence (symbolic)

(tx = time unit/clock cycle)

- t1: MAR <- PC
- t2: MBR <- (memory or MAR)
- t3: PC <- PC +1  
IR <- MBR

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar | 

s and data.  
Instructions  
aces for the  
ta buses. So  
s significance

us and address  
other for data

Data  
Memory

sts of ALU, multiplier,  
n memory and always  
and control buses are

ese components can be  
chnology, then such chip  
e and execute them,  
es and send results to the  
on the development of  
scale integration (SSI).

OR

- t1: MAR  $\leftarrow$  PC
- t2: MBR  $\leftarrow$  [memory or MAR]  
PC  $\leftarrow$  PC + 1
- t3: IR  $\leftarrow$  MBR

#### Control Unit

The control unit is the heart of CPU. It gets instruction from memory. The control unit decides what the instructions mean and directs the necessary data to be moved from memory to ALU. It must communicate with both ALU and main memory. It coordinates all activities of processor unit, peripheral devices and storage devices. Two types of control unit can be implemented in computing systems.

##### 1. Hardwired Control Unit

- This CU is essentially a combinatorial circuit. Its i/p logic signals are transformed into set of o/p logic signals which are control signals.
- The CU performs different operations on the basis of op codes.
- We have to derive the Boolean expression for each control signal as a function of input.
- Since modern processor needs a Boolean equation, it is very difficult to build a combinational circuit that satisfies all these operations.
- It has faster mode of operation.
- A hardwired control unit needs rewiring if design has to be modified.

##### 2. Micro-programmed Control Unit

- An alternative to hardwired CU.
- In micro-programmed control unit, the control information is stored in control memory.
- The control memory is programmed to initiate required sequence of operations.
- Use sequences of instructions to perform control operations performed by micro operations.
- Control address register contains the address of the next microinstruction to be read
- As it is read, it is transferred to control buffer register.
- Sequencing unit loads the control address register and issues a read command.
- It is cheaper and simple than hardwired CU.
- It is slower than hardwired CU.

#### 1.8 Introduction to Register Transfer Language (RTL)

The symbolic notation used to describe the micro operation transfers among register is called register transfer language. It is one of the forms of hardware description language (HDL). The

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar |



term 'register transfer' implies the availability of hardware logic circuits that can perform a stated instruction and transfer the data. It also transfers result of the operation to the same or another register. The term 'language' is borrowed from programmers, who apply this term to programming language.

RTL is the convenient tool for describing the internal organization of digital computers in concise and precise manner. It can also be used to facilitate the design process of digital systems such as microprocessors.

#### Fetch and execute cycle of MOV A, B in terms of RTL specification:

Within the fetch cycle, the operations performed during execution of instruction MOV A, B are:

- i) The program counter contains the address of the next instruction to be executed. If the next instruction to be executed is MOV A, B; the program counter contains the address of the memory location where the instruction code for MOV A, B resides. In the first operation of fetch cycle, the contents of program counter will be transferred to the memory address register (MAR). The memory address register then uses the address bus to transmit its contents that specifies the address of memory location from where that instruction code of MOV A, B is to be fetched.

Let t1 indicates the period of first operation

t1 : MAR  $\leftarrow$  PC

- ii) When the control unit issues the memory read signal, the contents of the address memory location specified by MAR will be transferred to the memory buffer register (MBR).

Suppose t2 is the time period for this operation.

t2 : MBR  $\leftarrow$  Memory or [MAR]

- iii) Finally the contents of MBR will be transferred to the instruction register and then the program counter gets incremented.  
Let t3 be the time required by the CPU to complete these operations.

t3 : IR  $\leftarrow$  (MBR)

PC  $\leftarrow$  PC + 1

Reference: Ramesh S. Gaonkar |



After the fetch cycle completed, the execution starts. The execute cycle steps:

- i) At the start of execution cycle, the instruction register (IR) consists of instruction code for instruction MOV A, B. The address field of instructions specifies the addresses of the two memory locations A & B. The first step needed is to obtain the data from the location B. For this the address field of IR indicating the address of memory location will be transferred to address bus through the MAR.

Let t1 be the time taken

t1 : MAR ← {IR(Address of B)}

- ii) When the control unit issues a memory read signal, the contents of location B will be output [written] to the memory buffer register (MBR). Now the content of B which is to be written to memory location A is contained in MBR.

Let t2 be the time taken for that operation.

t2 : MBR ← {B}

- iii) Now, we need the memory location of A because it is being written with the data of location B. For this the address field of IR indicating the address of memory location A. A will be transferred to MAR in time t3.

t3 : MAR ← {IR(Address of A)}

- iv) When the control unit issues the memory write signal, the contents of MBR will be written to the memory location indicated by the contents of MAR in time t4.

t4 : A ← MBR      or      t4 : [MAR] ← MBR

Note: [MAR] = A

Program consists of instructions which contains different cycles like fetch and execute. These cycles in turn are made up of the smaller operation called micro operations.

#### Some RTL Examples

- 1) MVI A, 02H

Fetch:

T1: MAR ← PC

T2: MBR ← [MAR]

T3: IR ← MBR

PC ← PC + 1



#### Execute:

T4: MBR ← IR [address of immediate data]

T5: MAR ← IR [address of A]

T6: A ← MBR

- 2) LXI B, 0210H

#### Execute:

T4: MBR ← IR [address of immediate data]

T5: MAR ← IR [address of C]

T6: C ← MBR

T7: MBR ← IR [address of immediate data (MSB)]

T8: MAR ← IR [address of (B)]

T9: B ← MBR

- 3) LDA 2030H

#### Execute:

T4: MAR ← IR [address of immediate data]

T5: MBR ← IR [address of C]

T6: MAR ← IR [address of A]

T7: A ← MBR

- 4) STA 2030H

#### Execute:

T4: MAR ← IR [address of immediate A]

T5: MBR ← A

T6: MAR ← IR [address of immediate data]

T7: [MAR] ← MBR

#### Advantages of Microprocessor:

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

## Applications of Microprocessors:

- **Microcomputer:** Microprocessor is the CPU of the microcomputer.
- **Embedded system:** Used in microcontrollers.
- **Measurements and testing equipment:** used in signal generators, oscilloscopes, counters, digital voltmeters, x-ray analyzers, 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 applications

Hd  
09



## Chapter-2

### Programming with 8085 microprocessor

#### Internal Architecture of 8 bit microprocessor and its registers:

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

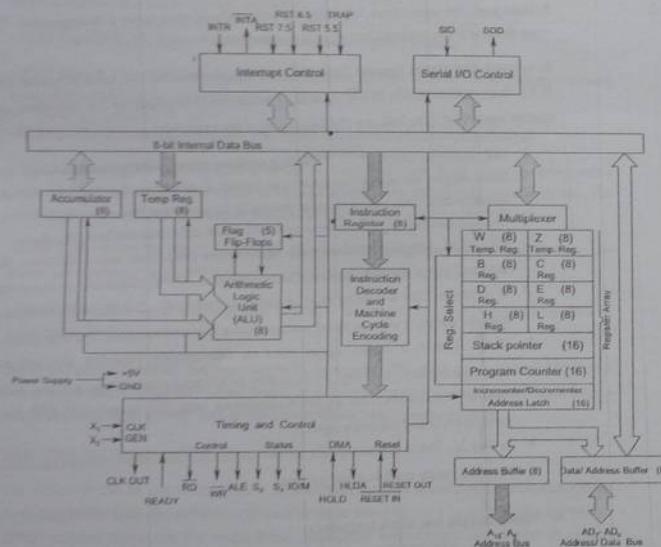


Fig: The 8085A microprocessor Functional Block Diagram

Source: Intel Corporation. *Embedded Microprocessors* (Santa Clara, Calif: Author, 1994) pp 1-11

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

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

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

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

**5. Register Array:** (Scratch pad registers B, C, D, E): It is a 8 bit register accessible to the programmers. Data can be stored upon it during program execution. These can be used individually as 8-bit registers or in pair BC, DE as 16 bit registers. The data can be directly added or transferred from one to another. Their contents may be incremented or decremented and combined logically with the content of the accumulator.

**Register H & L:** - They are 8 bit registers that can be used in same manner as scratch pad registers.

**Stack Pointer (SP):** - It is a 16 bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer.

**Program Counter (PC):** - Microprocessor uses the PC register to sequence the execution of the instructions. The function of PC is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the PC is incremented by one to point to the next memory location.

#### 6. Flags:

| D <sub>7</sub> | D <sub>6</sub> | D <sub>5</sub> | D <sub>4</sub> | D <sub>3</sub> | D <sub>2</sub> | D <sub>1</sub> | D <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| S              | Z              | X              | AC             | X              | P              | X              | CY             |

Register consists of five flip flops, each holding the status of different states separately known as flag register and each flip flop are called flags. 8085A can set or reset one or more of the flags and are sign(S), Zero (Z), Auxiliary Carry (AC) and Parity (P) and Carry (CY). The state of flags indicates the result of arithmetic and logical operations, which in turn can be used for decision making processes. The different flags are described as:

- **Carry:** - If the last operation generates a carry its status will 1 otherwise 0. It can handle the carry or borrow from one word to another.
- **Zero:** - If the result of last operation is zero, its status will be 1 otherwise 0. It is often used in loop control and in searching for particular data value.
- **Sign:** - If the most significant bit (MSB) of the result of the last operation is 1 (negative), then its status will be 1 otherwise 0.
- **Parity:** - If the result of the last operation has even number of 1's (even parity), its status will be 1 otherwise 0.
- **Auxiliary carry:** - If the last operation generates a carry from the lower half word (lower nibble), its status will be 1 otherwise 0. Used for performing BCD arithmetic.

#### 7. Timing and Control Unit:

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals. The control signals are similar to the sync pulse in an oscilloscope. The RD and WR signals are sync pulses indicating the availability of data on the data bus.

#### 8. Interrupt controls:

The various interrupt controls signals (INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP) are used to interrupt a microprocessor.

#### 9. Serial I/O controls:

Two serial I/O control signals (SID and SOO) are used to implement the serial data transmission.

#### Characteristics (features) of 8085A microprocessor and its signals

The 8085A (commonly known as 8085) is a 8-bit general purpose microprocessor capable of addressing 64K of memory. The device has 40 pins, require a +5V single power supply and can operate with a 3-MHZ, single phase clock.

The all the signals associated with 8085 can be classified into 6 groups:

**1: Address bus:** The 8085 has 16 signal lines that are used as the address bus; however, these lines are split into two segments A<sub>15</sub>-A<sub>8</sub> and A<sub>7</sub>-A<sub>0</sub>. The eight signals A<sub>15</sub>-A<sub>8</sub> are unidirectional and used as high order bus.

**2. Data bus:** The signal lines A<sub>7</sub>- A<sub>0</sub> are bidirectional, they serve a dual purpose. They are used the low order address bus as well as data bus.

**3. Control and status signals:** This group of signals includes two control signals ( $\overline{RD}$  and  $\overline{WR}$ ), three status signals ( $\overline{IO/M}$ ,  $S_1$  and  $S_0$ ) to identify the nature of the operation, and one special signals (ALE) to indicate the beginning of the operation.

➤ **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 A<sub>7</sub>-A<sub>0</sub> 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<sub>7</sub>-A<sub>0</sub>.

➤ **RD- Read:** this is a read control signal(active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.

➤ **WR- Write:** This is a write control signal (active low) . This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.

➤  **$\overline{IO/M}$ :** 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 indicates a memory operation. This signal is combined with  $\overline{RD}$  (Read) and  $\overline{WR}$  (Write) to generate I/O and memory signals.

➤  **$S_1$  and  $S_0$ :** These status signals, similar to  $\overline{IO/M}$ , can identify various operations, but they are rarely used in small systems.

#### **4. Power Supply and Clock frequency:**

- VCC: +5V power supply
- VSS: Ground reference
- X1 and X2: A crystal (RC or LC network) is connected at these two pins for frequency.

- CLK OUT: It can be used as the system clock for other devices.

#### S. Externally Initiated signals:

- INTR [input]: interrupt request, used as a general purpose interrupt.

- INTA [Output]: This is used to acknowledge an interrupt.

- RST 7.5, 6.5, 5.5 (inputs): These are vectored interrupts that transfer the program control to specific memory locations. They have higher priorities than INTR interrupt. Among these three, the priority order is 7.5, 6.5, and 5.5.

- TRAP [input]: This is a non-maskable interrupt with highest priority.

- HOLD [input]: This signal indicates that a peripheral such as a DMA (Direct Memory Access) controller is requesting use of Address and data bus.

- HLDA [output]: Hold Acknowledge: This signal acknowledges the HOLD request

- READY [Input] : This signal is used to delay the microprocessor Read or Write cycles until a slow- responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high.

- RESET IN : When the signal on this pin goes low, the program counter is set to zero, the buses are tri-stated, and MPU is reset.

- RESET OUT: This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

**Serial I/O ports:** The 8085 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, such as the transmission over telephone lines.

#### Instruction description and Format:

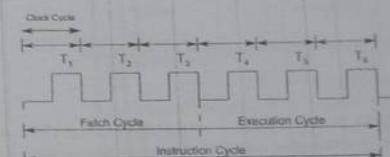
The computer can be used to perform a specific task, only by specifying the necessary steps to complete the task. The collection of such ordered steps forms a 'program' of a computer. These ordered steps are the instructions. Computer instructions are stored in central memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it until the completion of the program.



#### Instruction cycle:

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

In the fetch cycle the central processing unit obtains the instruction code the memory for its execution. Once the instruction code is fetched from memory, it is then executed. The execution cycle consists the calculating the address of the operands, fetching them, performing operations on them and finally outputting the result to a specified location.



#### Instruction description and format:

An instruction manipulates the data and a sequence of instructions constitutes a program. Generally each instruction has two parts: one is the task to be performed, called the **operation code (Op-Code)** field, and the second is the data to be operated on, called the **operand** or **address field**. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or an 8-bit (or 16-bit) address. The Op-Code field specifies how data is to be manipulated and address field indicates the address of a data item. For example:

ADD R<sub>1</sub>, R<sub>0</sub>

Op-code address

Here R<sub>0</sub> is the source register and R<sub>1</sub> is the destination register. The instruction adds the contents of R<sub>0</sub> with the content of R<sub>1</sub> and stores result in R<sub>1</sub>.

8085 A can handle at the maximum of 256 instructions ( $2^8$ )(246 instructions are used). The sheet which contains all these instructions with their hex code, mnemonics, descriptions

and function is called an instruction sheet. Depending on the number of address specified in instruction sheet, the instruction format can be classified into the categories.

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

E.g. LXI B, 4050 H

Micro operation specifies the transfer of data into or out of a register.

#### Classification of an instruction

An instruction is a binary pattern designed inside a microprocessor to perform a specific function (task). The entire group of instructions called the instruction set. The 8085 instruction set can be classified into 5 different groups:

- Data transfer group: The instructions which are used to transfer data from one register to another register or register to memory.
- Arithmetic group: The instructions which perform arithmetic operations such as addition, subtraction, increment, decrement etc.
- Logical group: The instructions which perform logical operations such as AND, OR, XOR, COMPARE etc.
- Branching group: The instructions which are used for looping and branching are called branching instructions like jump, call etc.
- Miscellaneous group: The instructions relating to stack operation, controlling purposes such as interrupt operations are fall under miscellaneous group including machine control like HLT, NOP.

#### Data transfer group instructions:

It is the longest group of instructions in 8085. This group of instruction copy data from a source location to destination location without modifying the contents of the source. The transfer of data may be between the registers or between register and memory or between an I/O device and accumulator. None of these instructions changes the flag. The instructions of this group are:

- 1) **MOV R<sub>d</sub>, R<sub>s</sub>**       $R_d \leftarrow R_s$  (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. MOV A, B
- 2) **MVI R, 8 bit data** (move immediate instruction)
  - 2 byte instruction
  - Loads the second byte ( 8 bit immediate data) into the register specified.
  - R may be A, B, C, D, E, 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 the location 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. MOV B, M

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

Sol<sup>n</sup>:

```

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

## Microprocessors

### Chapter 2 : Programming with 8085 Microprocessor

MOV M, C  
MVI L, 90H  
MOV M, D  
HLT

#### 5) LXI R<sub>P</sub>, 2 bytes data (load register pair)

- 3-byte instruction
- Load immediate data to register pair
- Register pair may be BC, DE, HL & SP(Stack pointer)
- 1<sup>st</sup> byte- Op-code
- 2<sup>nd</sup> byte – lower order data
- 3<sup>rd</sup> byte- higher order data
- E.g. LXI B, 4532H; B ← 45, C ← 32H

#### 6) MVI M, data (load memory immediate)

- 2 byte instruction.
- Loads the 8-bit data to the memory location whose address is specified by the contents of HL pair. E.g. MVI M, 35H; [HL] ← 35H

#### 7) LDA 4035H (Load accumulator direct)

- 3-byte instruction
- Loads the accumulator with the contents of memory location whose address is specified by 16 bit address.
- A ← [4035H]

#### 8) LDAX R<sub>P</sub> (Load accumulator indirect)

- 1 byte instruction.
- Loads the contents of memory location pointed by the contents of register pair to accumulator.
- E.g. LDAX B [A] ← [[BC]]  
LXI B, 9000H → B=90, C=00  
LDAX B → A= [9000]

#### 9) STA 16-bit address (store accumulator contents direct)

- 3-byte instruction.
- Stores the contents of accumulator to specified address
- E.g. STA FA00H [FA00] ← [A]

#### 10) STAX RP [RP] ← A

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

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar |



## Microprocessors

### Chapter 2 : Programming with 8085 Microprocessor

- E.g. STAX B  
LXI B, 9500H  
LXI D, 9501H  
MVI A, 32H  
STAX B  
MVI A, 7AH  
STAX D [DE] ← A

#### 11) IN 8-bit address

- 2-byte instruction
- Read data from the input port address specified in the second byte and loads data into the accumulator i.e. input port content to accumulator.
- E.g. IN 40H A ← [40H]

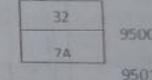
#### 12) OUT 8-bit address

- 2-byte instruction
- Copies the contents of the accumulator to the output port address specified in the 2<sup>nd</sup> byte. That means accumulator to output port: P ← A
- E.g. OUT 40H [40] ← A

#### 13) LHLD 16-bit address ( 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. LXI H, 9500H



MVI L, 01H

MVI m, 7AH

LHLD 9500H H=7A, L=32

#### 14) SHLD 16-bit address ( store HL 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

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar |

15) XCHG (Exchange)

- Exchanges DE pair with HL pair.
- E. g. LXI H, 7500H      H=75, L=00  
LXI D, 9532H      D=95, E=32  
XCHG      H=95, L=32  
                D=75, E=00

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 formats (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  $\leftarrow$ [9500]  
              IN 80H      A  $\leftarrow$ [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]  $\leftarrow$ 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

Compiled by: Er. Hari Aryal

Email: harya4@gmail.com

Reference: Ramesh S. Gaonkar |

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

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

Arithmetic group Instructions:

The 8085 microprocessor performs various arithmetic operations such as addition, subtraction, increment and decrement. These arithmetic operations have the following mnemonics.

1) ADD R/M

- 1 byte add instruction.
- Adds the contents of register/memory to the contents of accumulator and stores the result in accumulator.
- E. g. Add B; A  $\leftarrow$ [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 98H ; A  $\leftarrow$ A+98H

3) SUB R/M

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

4) SUI 8 bit 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  $\leftarrow$ A-D3H

5) INR R/M, DCR R/M

- 1 byte increment and decrement instructions.

## Microprocessors

### Chapter 2 : Programming with 8085 Microprocessor

- 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.
- 6) 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) ADC R/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 the Previous carry.
  - E.g. ADC B ; A ← A+B+CY  
ACI 70H ; A ← A + 70+CY
- 8) SBB B/M
- 1 byte instruction.
  - Subtracts the contents of register or memory from the contents of accumulator and stores 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 stores the 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  
DAD B; HL=HL+BC

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar | 13

## Microprocessors

### Chapter 2 : Programming with 8085 Microprocessor

$$7320+4220=8540$$

#### 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 | ↓ B3 |
|      | ADD C      | 1 0 1 0 0 1 0 1 0 | ↓ 4A |
|      |            | CY                | CY   |

#### Subtraction operation in 8085:

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.

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar | 13

Microprocessors

Chapter 2 : Programming with 8085 Microprocessor

E.g.

MVI A, 97H

MVI B, 65H

SUB B

$$\begin{array}{r}
 65H: 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1 \\
 1's complement of 65H: 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\
 \hline
 2's Complement of 65H: 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\
 97H: +1\ 0\ 0\ 1\ 0\ 1\ 1 \\
 \hline
 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\
 \boxed{32} \\
 \text{CY}
 \end{array}$$

Complement carry = 0 ∴ A=32 CY=0

B=97H, A=65H

97H: 1 0 0 1 0 1 1 1

MVI B, 65H

SUB B

$$\begin{array}{r}
 1's complement of 97H: 0 1 1 0 1 0 0 0 \\
 \hline
 2's Complement of 97H: 0 1 1 0 1 0 0 1 \\
 65H: +0 1 1 0 0 1 0 1 \\
 \hline
 0 1 1 0 0 1 1 0 \\
 \text{(Result in 2's complement form)} \\
 \text{CY}
 \end{array}$$

$$\begin{array}{l}
 \text{CY=1, } A= \text{CE: } 1 1 0 0 1 1 1 1 0 \\
 \text{1's complement: } 0 0 1 1 0 0 0 0 1 \\
 \text{2's complement: } 0 0 1 1 0 0 0 1 0 \\
 \boxed{32}
 \end{array}$$

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      MOVA, B

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

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar |

15

Microprocessors

Chapter 2 : Programming with 8085 Microprocessor

STA 8000H

LXI H, 8000H

MVI M, F2H

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

LHLD 2050H

MVI A, 30H

SHLD 3000H

STA 2051H

HLT

5. Pair B contains 1122H and pair D contains 3344H. WAP to exchange the contents of B and D pair using XCHG instruction.

LXI B, 1122H

B=11, C=22

LXI D, 3344H

D=33, E=44

MOV H, B

MOV L, C

XCHG

MOV B, H

MOV C, L

HLT

(Exchange DE pair with HL pair)

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

LXI H, 7342H

LXI B, 1989H

MOVA, L

ADD C

DAA

MOVC, A

MOV A, H

ADC B

DAA

MOV B, A

7. Register BC contain 2793H and register DE contain 3182H. Write instruction to add these two 16 bit numbers and place the sum in memory locations 2050H and 2051H. What is DAA instruction? Explain its purpose with an example.(Back Paper 2002)

$$\begin{array}{r}
 \text{MOV A, C} \quad 93H: \quad 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1 \\
 \text{ADD E} \quad +82H: \quad 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0 \\
 \text{MOV L, A} \quad L=15 \quad 15H: \quad \boxed{1}\ 0\ 0\ 1\ 0\ 1\ 0\ 1 \quad 15H \\
 \text{MOV A, B} \quad 27H: \quad 0\ 0\ 1\ 0\ 0\ 1\ 1
 \end{array}$$

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar |

15

|            |                               |                 |
|------------|-------------------------------|-----------------|
| ADC D      | +32H                          | 0 0 1 1 0 0 0 1 |
| MOV H, A   | H=59H                         | 0 1 0 1 1 0 0 1 |
| SHLD 2050H | : [2050] ← 15H , [2051] ← 59H |                 |

Note: SHLD stores the contents of L in specified location and contents of H in next higher location.

- E. Register BC contains 8538H and register DE contain 62ASH. Write Instructions to subtract the contents of DE from the contents of BC and Place the result in BC.

|          |             |
|----------|-------------|
| MOV A, C | 38          |
| SUB E    | - AS        |
| MOV C, A | ; C=98 1 93 |
| MOV A, B | 85 ;        |
| SBB D    | 62          |
| MOV B, A | ; B=22 -1   |
|          | 22          |

#### 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.

$$\begin{array}{r} \text{E.g. } \quad \begin{array}{r} \text{A: } 0000 \quad 1010 \\ + 0000 \quad 0110 \\ \hline 0001 \quad 0000 \rightarrow 10_{\text{bcd}} \end{array} \end{array}$$

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

$$\begin{array}{r} 77 = 0111 \quad 0111 \\ + 48 = 0100 \quad 1000 \\ \hline 125 \quad 1011 \quad 1111 \\ \quad + 0110 \\ \hline \text{CY} = 1 \quad 0010 \quad 0101 \rightarrow 25_{\text{bcd}} \end{array}$$

Compiled by: Er. Hari Aryal Email: hariyal4@gmail.com Reference: Ramesh S. Gaonkar | 17

#### Logical Group Instructions:

A microprocessor is basically a programmable logic chip. It can perform all the logic functions of the hardwired logic through its instruction set. The 8085 instruction set includes such logic functions as AND, OR, XOR and NOT (Complement).

The following features hold true for all logic instructions:

- 1) The instructions implicitly assume that the accumulator is one of the operands.
- 2) All instructions reset (clear) carry flag except for complement where flag remains unchanged.
- 3) They modify Z, P & S flags according to the data conditions of the result.
- 4) Place the result in the accumulator.
- 5) They do not affect the contents of the operand registers.

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/ANL     | 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 |

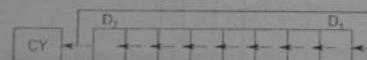
Used to indicate end of data.

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<sub>7</sub> becomes D<sub>0</sub>.
- The carry flag is modified according to D<sub>7</sub>.



$$CY = D_7, D_7 = D_6, D_6 = D_5, \dots, D_1 = D_0, D_0 = D_7$$

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar |



2) RAL: Rotate accumulator left through carry

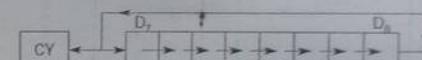
- Each bit is shifted to the adjacent left position. Bit D<sub>7</sub> becomes the carry bit and the carry bit is shifted into D<sub>0</sub>.
- The carry flag is modified according to D<sub>7</sub>.



$$CY = D_7, D_7 = D_6, D_6 = D_5, \dots, D_1 = D_0, D_0 = CY$$

3) RRC: rotate accumulator right

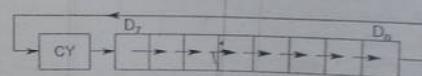
- Each bit is shifted right to the adjacent position. Bit D<sub>0</sub> becomes D<sub>7</sub>.
- The carry flag is modified according to D<sub>0</sub>.
- The carry flag is modified according to D<sub>9</sub>.



$$CY = D_0, D_7 = D_6, \dots, D_0 = D_1$$

4) RAR: Rotate accumulator right through carry

- Each bit is shifted right to the adjacent position. Bit D<sub>0</sub> becomes the carry bit and the carry bit is shifted into D<sub>7</sub>.



$$CY = D_0, D_7 = D_6, \dots, D_7 = CY$$

Others:

-CMC – Complement carry

STC- set carry flag

9. 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<sub>10</sub>)

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar |

However these procedures are invalid when logic 1 is rotated from D<sub>7</sub> to D<sub>0</sub> 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 : - [10] [2001 Ashwin [2]]

#### 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.

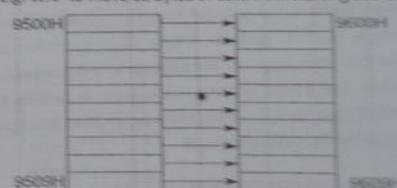
```
MVI A, 80H      START: IN 00H
OUT 43H        OUT 03H
MVI A, 00H      JMP START
L1: OUT 40H
INR A
JMP L1
HLT
```

#### 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 set 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 9500H to 9600H



```
2000  MVI B, 0AH
2002  LXI H, 9500H
2005  LXI D, 9600H
```

**Microprocessors**

Chapter 2 : Programming with 8085 Microprocessor

```

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 200B
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: MOVA, M
ANI 01H
JNZ L1 ; If data is odd then go to L1.
MVI A, 00H
JMP L3
L1: MOVA, 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.

Compiled by: Tr. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar | 23

**Microprocessors**

Chapter 2 : Programming with 8085 Microprocessor

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, RPÖ**
  - Return subroutine conditionally.
  - Same as RET except that it 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, AOH | 3001                | ADD C    |
| 2004 | CALL 3000H | 3002                | RET      |
| 2007 | MOV B, A   | PC = 2007 (i.e. sp) |          |
| 2008 | HLT        | SP = 2007 (i.e. PC) |          |

PC = 3000 (i.e. 16bit)

Q. What is the purpose of branching instruction? List out all the branching in 8085 and explain each with example. (2+8) (2046 shrawan)

Restart Instruction:

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

Compiled by: Tr. Hari Aryal

Email: haryal4@gmail.com

Reference: Ramesh S. Gaonkar | 24

| Restart instruction | Hex Code | <u>Call location in hex</u> |
|---------------------|----------|-----------------------------|
| 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.

#### Miscellaneous Group Instructions:

##### STACK

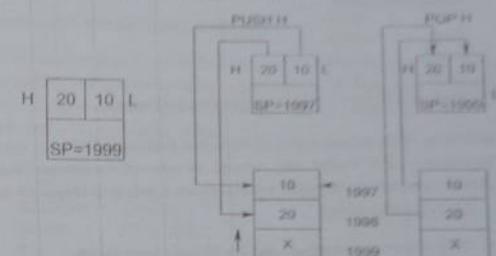
The stack is defined as a set of memory location in R/W memory, specified by a programmer in a main memory. These memory locations are used to store binary information temporarily during the execution of a program.

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 – exchanges top of stack (TOS) with HL

4. SPHL – move HL to SP

5. 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

BEFORE 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.  
 (3+2+5) [2063 kartik]

6. WAP to sort in ascending order for 10 bytes from 1120H.

START: LXI H, 1120H

MVI D, 00H

MVI C, 09H

L2: MOVA, M

INX H

CMP M

JC L1 ; if A < M

MOV R, 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

Time Delay and Counter

Counter:

It is designed simply by loading an appropriate number into one of the registers and using the INR or DCR instructions. A loop is established to update a count, and each count is checked to determine whether it has reached the final number, if not the loop is repeated.

Time Delay

When we use loop by counter, the loop causes the delay. Depending upon the clock period of the system, the time delay occurred during looping. The instructions within the loop use their own T-states so they need certain time to execute resulting delay.

Suppose we have an 8085 micro processor with 2MHZ clock frequency. Let us use the instruction MVI which takes 7 T-states. Clock frequency of system (f) = 2 mhz

$$\text{Clock period (T)} = 1/f = \frac{1}{2} \times 10^{-6} = 0.5\text{Microprocessor}$$

$$\text{Time to execute MVI} = 7 \text{ T-states} \times 0.5$$

$$= 305\text{Micro processor}$$

$$\text{Eg. MVI C, FFH} \quad 7$$

$$\text{LOOP: DCR C} \quad 4$$

$$\text{JNZ LOOP} \quad 10/7$$

Here register C is loaded with count FFH (255<sub>16</sub>) by using MVI which takes 7 T-states. Nest 2 Instructions DCR and JNZ form a loop with a total of 14 (4+10) T-states. The loop is repeated 255 times until C=0. The time delay in loop (TC) with 2 mhz frequency is

$$Tl = (T * \text{loop T-states} * \text{count})$$

Where, Tl = time delay in loop

T = system clock period

Count = decimal value for counter

$$Tl = 0.5 * 10^{-6} * 14 * 255$$

$$= 1785 \text{ ms}$$

But JNZ takes only 7 T-states when exited from loop i.e. last count = 0.50 adjusted loop delay

$$Tla = Tl - (3 \text{ T-states} - \text{clock period})$$

$$= 1785 \text{ ms} - 1.5 \text{ ms} = 1783.5 \text{ ms}$$

Total delay of program

$$Td = \text{Time to execute outside code} + Tla \text{ inside Loop}$$

$$= 7 * 0.5MS + 1783.5 \text{ ms}$$

$$= 1787 \text{ ms} \approx 1.8 \text{ ms}$$

To increase the time delay beyond 1.8 ms for 2MHZ microprocessor, we need to use counter for register pair or loop within a loop.

|                    | T-States | Clocks          |
|--------------------|----------|-----------------|
| L2: MVI B, 40H; 64 | 7        | 7*1             |
| MVI C, 80H; 128    | 7        | 7*64            |
| DCR C              | 4        | 4*128*64        |
| JNZ L1             | 10/7     | (10*127+7*1)*64 |
| DCR B              | 4        | 4*64            |
| JNZ L2             | 10/7     | 10*63+7*1       |
| RET                | 10       | 10*1            |
|                    |          | 115854          |

For 2 MHZ Microprocessor

Total time taken to execute above

$$\text{Subroutine} = 1158 * 0.5 * 10^{-6} \text{ s}$$

$$= 57.927 \text{ ms}$$

#### BCD to binary conversion:

In most microprocessor-based products, Data are entered and displayed in decimal numbers. For example, in an instrumentation laboratory, readings such as voltage and current are maintained in decimal numbers, and data are entered through decimal keyboard. The system monitor program of the instrument converts each key into an equivalent 4-bit binary number and stores two BCD numbers in an 8-bit register or a memory location. These numbers are called packed BCD.

Conversion of BCD number into binary number employs the principle of positional weighting in a given number.

$$\text{E.g. } 72_{10} = 7 \times 10 + 2$$

Converting an 8-bit BCD number into its binary equivalent requires,

- Separate an 8-bit packed BCD number into two 4-bit unpacked BCD digits i.e. BCD<sub>1</sub> and BCD<sub>2</sub>.
- Convert each digit into its binary value according to its position.
- Add both binary numbers to obtain the binary equivalent of the BCD number.

E.g.

$$72_{10} = 0111\ 0010 \text{BCD}$$

Step 1: 0111.0010 → 0000 0010 Unpacked BCD<sub>1</sub>

→ 0000 0111 Unpacked BCD<sub>2</sub>

Step 2: Multiply BCD<sub>1</sub> by 10 (7 × 10)

Step 3: Add BCD<sub>2</sub> to answer of step 2

Program: 2-digit BCD- to- Binary Conversion

A BCD number between 0 and 99 is stored in an R/W memory location called the Input Buffer (INBUF). Write a main program and a conversion subroutine (BCDBIN) to convert the BCD number into its equivalent binary number. Store the result in a memory location defined as the Output Buffer (OUTBUF).

```

LXI H, 2020H
MVI E, 0A H
MOV A, M ; 0111 0010
ANI F0H ; 0111 0000
RR
RR
RR
RR
MOV B,A
XRA A
L1: ADD B; 7×10+2
DCR E
JNZ L1
MOV C, A
MOV A, M
ANI 0FH
ADD C
STA 2030H
HLT

```

#### Binary to BCD conversion

$$1111\ 1111_2 (\text{FFH}) = 255_{10} = 0010\ 0101\ 0101_{\text{BCD}}$$

Step 1: If < 100 goto step 2

Else subtract 100 repeatedly.

Quotient is BCD<sub>1</sub> (Divide by 100)

Step 2: If < 10 goto step 3

Else subtract 10 repeatedly.

Quotient is BCD<sub>2</sub> (Divide by 10)

Step 3: Remainder is BCD<sub>1</sub>

Program to convert 8-bit binary to BCD

```
LXI SP, 1999H
LXI H, 2020H; Source
MOV A, M
CALL PWRTE
HLT
PWRTE: LDI H, 2030H; Destination
MVI B, 64H
CALL BINBCD
MVI B, 0AH
CALL BINBCD
MOV M, A
RET
BINBCD: MVI M, FFH
NEXT: INR M
SUB B
JNC NEXT
ADD B
INX H
RET
```

#### Binary to ASCII conversion

#### Alphanumeric codes:

A computer is a binary machine, to communicate with the computer in alphanumeric letters and decimal numbers, translation codes are necessary. The commonly used code is known as ASCII (American standard codes for information interchange). It is a 7-bit code with 128 combinations and each combination from 01H to 7FH is organized to a letter, decimal number, symbol or machine command. For eg. 30H to 39H represents 0 to 9, 41H to 5AH represents A to Z, 21H to 2FH represents various symbols and 61H to 7AH represents a to z.

| General Letters / Numbers | ASCII (Hex) | ASCII (Decimal) |
|---------------------------|-------------|-----------------|
| 0 - 9                     | 30 - 39     | 48 - 57         |
| A - Z                     | 41 - 5A     | 65 - 90         |
| a - z                     | 61 - 7A     | 97 - 122        |

Program to Convert 8 – bit binary number to ASCII

Step 1: If number < 10, then add 30H

Else add 37H (30H + 07H)

For example: A = A + 30H + 07H = 41H

```
LXI SP, 1999H
LXI H, 1120H; Source
LXI D, 1160H; Destination
MOV A, M
ANI F0H
RRC
RRC
RRC
CALL ASCII
STAX D
INX H
MOV A, M
ANI 0FH
CALL ASCII
STAX D
HLT
ASCII: CPI 0AH
JC BELOW
ADI 07H
BELOW: ADI 30H
RET
```

#### ASCII to Binary Conversion

Step 1: Subtract 30H

Step 2: If < 0AH, then binary as it is

Else subtract 07H

For example: If ASCII is 41H, then 41H - 30H = 11H; 11H - 07H = 0AH

Program to convert ASCII number to Binary:

```
LXI SP, 1999H
LXI H, 1040H; Source
LXI D, 1050H; Destination
MOV A, M
CALL ASCBIN
STAX D
HLT
ASCBIN: SUI 30H
```

```
CPI 0AH  
RC  
SUI 07H  
RET
```

### BCD to 7-Segment LED code Conversion:

- Table lookup technique (TLT) is used.
- In TLT, the codes of digits to be displayed are stored sequentially in memory.

| BCD Number | 7-Segment Code |
|------------|----------------|
| 0          | 3FH            |
| 1          | 06H            |
| 2          | 5BH            |
| 3          | 4FH            |
| 4          | 66H            |
| 5          | 6DH            |
| 6          | 7DH            |
| 7          | 07H            |
| 8          | 7FH            |
| 9          | 6FH            |
| Invalid    | 00H            |

### Program to convert 3 packed BCD to 7-Segment LED code

```
LXI SP, 1999H  
LXI H, 1150H  
MVI D, 03H  
LXI B, 1190H  
NEXT: MOV A, M  
ANI F0H  
RRC  
RRC  
RRC  
CALL CODE  
INX B  
MOV A, M  
ANI 0FH  
CALL CODE  
INX B  
INX H  
DCR D  
JNZ NEXT  
HLT
```

```
CODE: PUSH H  
LXI H, 1170H  
ADD L  
MOV L, A  
MOV A, M  
STAX B  
POP H  
RET
```

### 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  
RAR; Check multiplier bit is 1 or 0  
JNC BELOW  
NEXT: DAD D; Partial result in HL  
XCHG  
DAD H; Shift left multiplicand  
XCHG; Retrieve shifted multiplicand  
DCR B  
JNZ NEXT  
RET
```

Note: Try Division yourself.

### Chapter-3 Programming with 8086 microprocessor

#### Internal Architecture and Features of 8086 Microprocessor

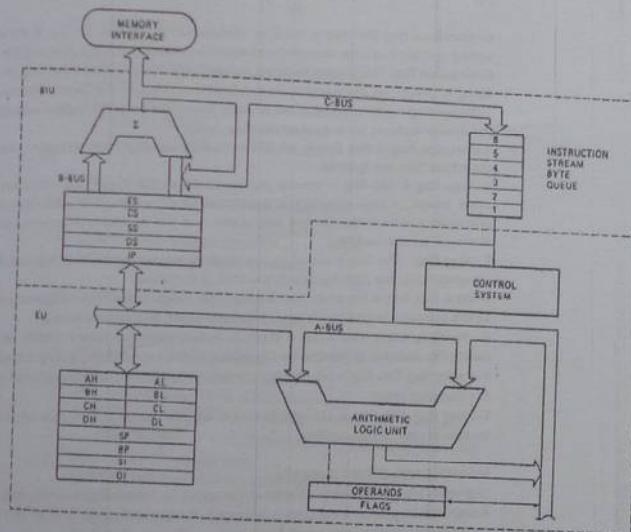


Fig: Internal Block Diagram of 8086 Microprocessor

#### Features of 8086 microprocessor

- Intel 8086 is a widely used 16 bit microprocessor.
- The 8086 can directly address 1MB of memory.
- The internal architecture of the 8086 microprocessor is an example of register based microprocessor and it uses segmented memory.
- It pre-fetches up to 6 instruction bytes from the memory and queues them in order to speed up the instruction execution.

Compiled by: Er. Hari Aryal

Email: haryala@gmail.com

Reference: Peter Abel |

- It has data bus of width 16 bits and address bus of width 20 bits. So it always accesses a 16 bit word to or from memory.
- The 8086 microprocessor is divided internally into two separate units which are Bus interface unit (BIU) and the execution unit (EU).
- The BIU fetches instructions, reads operands and write results.
- The EU executes instructions that have already been fetched by BIU so that instructions fetch overlaps with execution.
- A 16 bit ALU in the EU maintains the MP status and control flags, manipulates general register and instruction operands.

#### Bus Interface Unit(BIU) and its Components

The BIU sends out addresses, fetches instructions from memory, reads data from memory or ports and writes data to memory or ports. So it handles all transfers of data and address on the buses for EU. It has main 2 parts instruction queue and segment registers.

- The BIU can store up to 6 bytes of instructions with FIFO (First in First Out) manner in a register set called a queue. When EU is ready for next instruction, it simply reads the instruction from the queue in the BIU. This is done in order to speed up program execution by overlapping instruction fetch with execution. This mechanism is known as pipelining.

- The BIU contains a dedicated address, which is used to produce 20 bit address. Four segment registers in the BIU are used to hold the upper 16 bits of the starting address of four memory segments that the 8086 is working at a particular time. These are code segment, data segment, stack segment and extra segment. The 8086's 1 MB memory is divided into segments up to 64KB each.

- **Code segment register and instruction pointer (IP):** The CS contains the base or start of the current code segment. The IP contains the distance or offset from this address to the next instruction byte to be fetched. Code segment address plus an offset value in the IP indicates the address of an instruction to be fetched for execution.

#### Data Segment

Data segment Contains the starting address of a program's data segment. Instructions use this address to locate data. This address plus an offset value in an instruction, causes a reference to a specific byte location in the data segment.

#### Stack segment (SS) and Stack Pointer (SP)

Stack segment Contains the starting address of a program's stack segment. This segment address plus an offset value in the stack pointer indicates the current word in the stack being addressed.

Compiled by: Er. Hari Aryal

Email: haryala@gmail.com

Reference: Peter Abel |

#### **Extra Segment(s)**

It is used by some string (character data) to handle memory addressing. The string instructions always use the ES and destination index (DI) to determine 20 bit physical address.

#### **Execution Unit (EU)**

The EU decodes and executes the instructions. The EU contains arithmetic and logic (ALU), a control unit, and a number of registers. These features provide for execution of instructions and arithmetic and logical operations. It has nine 16 bit registers which are AX, BX, CX, DX, SP, BP, SI, DI and Flag. First four can be used as 8 bit registers (AH, AL, BH, BL, CH, DH, DL).

#### **AX Register**

AX register is called 16 bit accumulator and AL is called 8 bit accumulator. The I/O (IN or OUT) instructions always use the AX or AL for inputting/Outputting 16 or 8 bit data from or to I/O port.

#### **BX Register**

Bx is known as the base register since it is the only general purpose register that can be used as an index to extend addressing. The BX register is similar to the 8085's H, L register. BX can also be combined with DI or SI as C base register for special addressing.

#### **CX register**

The CX register is known as the counter register because some instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a Counter.

#### **DX register**

The DX register is known as data register. Some I/O operations require its use and multiply and divide operations that involve large values assume the use of DX and AX together as a pair. DX comprises the rightmost 16 bits of the 32-bit EDX.

#### **Stack Pointer (SP) and Base Pointer (BP)**

Both are used to access data in the stack segment. The SP is used as an offset from the current stack segment during execution of instructions. The SP's contents are automatically updated (increment/decrement) during execution of a POP and PUSH instructions.

The BP contains the offset address in the current stack segment. This offset is used by instructions utilizing the based addressing mode.

#### **Index register:**

The two index registers SI (Source Index) and DI (Destination Index) are used in indexed addressing. The instructions that process data strings use the SI and DI index register together with DS and ES respectively, in order to distinguish between the source and destination address.

#### **Flag register:**

The 8086 has nine 1 bit flags. Out of 9 six are status and three are control flags. The control bits in the flag register can be set or reset by the programmer.

|    |  |   |   |   |   |   |   |   |   |   |
|----|--|---|---|---|---|---|---|---|---|---|
|    |  | O | D | I | T | S | Z | A | P | C |
| D8 |  |   |   |   |   |   |   |   |   |   |

- **O- Overflow flag:** This flag is set if an arithmetic overflow occurs, i.e. if the result of a signed operation is large enough to be accommodated in a destination register.

- **D- 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 higher address, i.e. auto-incrementing mode otherwise the string is processed from the highest address towards the lowest address, i.e. auto-decrementing mode.

- **I-Interrupt flag:** If this flag is set the maskable interrupts are recognized by the CPU, otherwise they are ignored.

- **T- Trap flag:** If this flag is set the processor enters the single step execution mode. In other words, a trap interrupt is generated after execution of each instruction. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.

- **S- Sign flag:** This flag is set when the result of any computation is negative. For signed computations, the sign flag equals the MSB of the result.

- **Z- Zero flag:** This flag is set when the result of the computation is 0 or comparison performed by the previous instruction is zero. 1 for zero result, 0 for nonzero result.

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

- **P- Parity flag:** This flag is set to 1 if the lower byte of the result contains even number of 1s otherwise reset.

- **C-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.

#### **SEGMENT AND OFFSET ADDRESS:**

Segments are special areas defined in a program for containing the code, data and stack. A segment begins on a paragraph boundary. A segment register is of 16 bits in size and contains the starting address of a segment.

A segment begins on a paragraph boundary, which is an address divisible by decimal 16 or hex 10. Consider a DS that begins at location 038E0H. In all cases, the rightmost hex digit is zero, the computer designers decided that it would be unnecessary to store the zero digit in the segment register. Thus 038E0H is stored in register as 038EH.

The distance in bytes from the segment address to another location within the segment is expressed as an offset or displacement. Suppose the offset of 0032H for above example of data segment. Processor combines the address of the data segment with the offset as:

- SA: OA (segment address: offset address)
- 038E: 0032 H = 038E \* 10 + 0032
- = 038EO + 0032
- Physical address = 03912H

#### Instructions in 8086

##### 1) Arithmetic Instructions

- a) ADD reg<sub>8</sub>/mem<sub>8</sub>, reg<sub>8</sub>/mem<sub>8</sub>/ Immediate<sub>8</sub>  
ADD reg<sub>16</sub>/mem<sub>16</sub>, reg<sub>16</sub>/mem<sub>16</sub>/ Immediate<sub>16</sub>  
E.g. ADD AH, 15 ; It adds binary number  
      ADD AH, NUM1     ADD AX,[BX]  
      ADD [BX], CH/CX   ADD AX,[BX]
- b) ADC: Addition with Carry  
ADC reg/mem, reg/mem/Immediate data
- c) SUB: Subtract 8 bit or 16 bit binary numbers  
SUB reg/mem, reg/mem/immediate
- d) SBB: Subtract with borrow  
SBB reg/mem, reg/mem/immediate
- e) MUL: unsigned multiplication  
MUL reg<sub>8</sub>/mem<sub>8</sub> (8 bit accumulator - AL)  
MUL reg<sub>16</sub>/mem<sub>16</sub> (16 bit accumulator-Ax)  
E.g. MUL R<sub>8</sub> (multiplier) ≡ R<sub>8</sub> × AL → AX (16 bit result)  
      MUL R<sub>16</sub> (multiplier) ≡ R<sub>16</sub> × AL → DX:AX (32 bit result)  
IMUL – signed multiplication  
Same operation as MUL but takes sign into account
- f) DIV reg/mem  
E.g. DIV R<sub>8</sub> ≡ AX/R<sub>8</sub> → (Remainder → AH) & (Quotient → AL)  
      DIV R<sub>16</sub> ≡ DX:AX/R<sub>16</sub> → (Remainder → DX) & (Quotient → AX)  
IDIV- Signed division  
Same operation as DIV but takes sign into account.
- g) INC/DEC (Increment/Decrement by 1)  
INC/DEC reg./mem. (8 bit or 16bit)  
E.g. INC AL     DEC BX  
      INC NUM1

- i) NEG- Negate (2's complement)
- j) ASCII-BCD Conversion
- AAA: ASCII Adjust after addition
- AAS: ASCII Adjust after subtraction
- AAM: Adjust after multiplication
- AAD: Adjust after division
- DAA: Decimal adjust after addition
- DAS: Decimal adjust after subtraction

##### 2) Logical/shifting/comparison instructions

- a) Logical  
AND/OR/XOR reg/mem, reg/mem/immediate  
NOT reg/mem  
E.g. AND AL, AH  
      XOR [BX], CL
- b) Rotation  
ROL- rotate left, ROR-rotate right  
E.g. ROL AX, 1 ; rotated by 1  
      ROL AX, CL ; if we need to rotate more than one bit  
RCL-rotate left through carry  
RCR-rotate right through carry  
E.g. RCL AX, 1  
      RCL AX, CL ; Only CL can be used
- c) Shifting  
SHL- logical shift left  
SHR- logical shift right  
Shifts bit in true direction and fills zero in vacant place  
E.g. SHL reg/mem, 1/CL  
arithmetic shift left  
SAR- arithmetic shift right  
Shifts bit/word in true direction, in former case place zero in vacant place and in later case place previous sign in vacant place.  
E.g. 1 011010 [1 ⇢ 11011010

#### d) Comparison

CMP - compare  
CMP reg/mem, reg/mem/immediate  
E.g. CMP BH, AL

| Operand1 | Operand 2 | CF | SF | ZF |
|----------|-----------|----|----|----|
| >        |           | 0  | 0  | 0  |
| =        |           | 0  | 0  | 1  |
| <        |           | 1  | 1  | 0  |

TEST: test bits (AND operation)

TEST reg/mem, reg/mem/immediate

#### 3) Data Transfer Instructions:

MOV reg/mem, reg/mem/immediate  
LDS: Load data segment register  
LEA: Load effective address  
LES: Load extra segment register  
LSS: Load stack segment register  
E.g. LEA BX, ARR = MOV BX, OFFSET ARR  
LDS BX, NUM1

Segment address → DS  
Offset address → BX

XCHG reg/mem, reg/mem

E.g. XCHG AX, BX

XCHG AL, BL

XCHG CL,[BX]

IN AL, DX ; DX: Port address, AH also in AL  
OUT DX, AL/AH

#### 4) Flag Operation

CLC: Clear carry flag  
CLD: Clear direction flag  
CLI: Clear interrupt flag  
STC: Set Carry flag  
STD: Set direction flag  
STI: Set interrupt flag  
CMC: Complement Carry flag  
LAHF: Load AH from flags (lower byte)  
SAHF: Store AH to flags  
PUSHF: Push flags into stack  
POPF: Pop flags off stack

#### 5) STACK Operations

PUSH reg16  
POP reg16

#### 6) Looping instruction [CX is automatically used as a counter]

LOOP: loop until complete  
LOOPE: Loop while equal  
LOOPZ: loop while zero  
LOOPNE: loop while not equal  
LOOPNZ: loop while not zero

#### 7) Branching instruction

a) Conditional  
JA: Jump if Above  
JAE: Jump if above/equal  
JB: Jump if below  
JBE: Jump if below/equal  
JC: Jump if carry  
JNC: Jump if no carry  
JE: Jump if equal  
JNE: Jump if no equal  
JZ: Jump if zero  
JNZ: Jump if no zero  
JG: Jump if greater  
JNG: Jump if no greater  
JL: Jump if less  
JNL: Jump if no less  
JO: jump if overflow  
JS: Jump if sign  
JNS: Jump if no sign  
JP: jump if plus  
JPE: Jump if parity even  
JNP: Jump if no parity  
JPO: Jump if parity odd

#### b) Unconditional

CALL: call a procedure  
INT: Interrupt  
JMP: Unconditional Jump  
RET: Return  
IRET: interrupt return  
RETN/RETf: Return near/Far

#### 8) Type conversion

CBW: Convert byte to word  
CWD: Convert word to double word

AX → DX; AX

### 9) String instructions

- a) MOVS/ MOVSB/MOVSW ; Move string  
DS: SI source  
DS: DI destination  
CX: String length
- b) CMPS/ CMPSB/CMPW ; Compare string
- c) LODS/ LODSB/LODW ; Load string
- d) REP ; Repeat string

### Operators in 8086

- Operator can be applied in the operand which uses the immediate data/address.
- Being active during assembling and no machine language code is generated.
- Different types of operators are:

- 1) Arithmetic: +, -, \*, /
- 2) Logical : AND, OR, XOR, NOT
- 3) SHL and SHR: Shift during assembly
- 4) [ ]: index
- 5) HIGH: returns higher byte of an expression
- 6) LOW: returns lower byte of an expression.  
E.g. NUM EQU 1374 H  
MOV AL HIGH Num ; [ AL ] ← 13
- 7) OFFSET: returns offset address of a variable
- 8) SEG: returns segment address of a variable
- 9) PTR: used with type specifications  
BYTE, WORD, RWORD, DWORD, QWORD  
E.g. INC BYTE PTR [BX]

### 10) Segment override

MOV AH, ES: [BX]

### 11) LENGTH: returns the size of the referred variable

### 12) SIZE: returns length times type

E.g.: BYTE VAR DB?

```
WTABLE DW 10 DUP (?)  
MOV AX, TYPE BYTEVAR ; AX = 0001H  
MOV AX, TYPE WTABLE ; AX = 0002H  
MOV CX, LENGTH WTABLE ; CX = 000AH  
MOV CX, SIZE WTABLE ; CX = 0014H
```

### 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, identifiers, 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

| Directive:   | Identifiers | operation | operand | comment            |
|--------------|-------------|-----------|---------|--------------------|
| Instruction: | COUNT       | DB        | 1       | ; initialize count |
| 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]

Email: harya4@gmail.com

Reference: Peter Abel | 

Compiled by: Er. Hari Aryal

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) Dn Directive (Defining data types)

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

Compiled by: Er. Hari Aryal

Email: harya4@gmail.com

- 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.

#### Program written in Conventional full segment directive

```

Page 60,132
TITLE SUM program to add two numbers

STACK SEGMENT PARA STACK 'Stack'
DW 32 DUP(0)
STACK ENDS

DATA SEG SEGMENT PARA 'Data'
NUM1 DW 3291
NUM 2 DW 582
SUM DW?
DATA SEG ENDS

CODE SEG SEGMENT PARA 'Code'
MAIN PROC FAR
ASSUME SS: STACK, DS:DATASEG, CS:CODESEG
MOV AX, @DATA
MOV DS, AX
MOV AX, NUM1
ADD AX, NUM2
MOV AX, 4C00H
INT 21H
MAIN ENDP
CODESEG ENDS
END MAIN

```

#### Description for conventional program:

- STACK contains one entry, DW (define word), that defines 32 words initialized to zero, an adequate size for small programs.
  - DATASEG defines 3 words NUM1, NUM2 initialized with 3291 and 582 and sum uninitialized.
  - CODESEG contains the executable instructions for the program, PROC and ASSUME generate no executable code.
  - The ASSUME directive tells the assembler to perform these tasks.
  - Assign STACK to SS register so that the processor uses the address in SS for addressing STACK.
  - Assign DATASEG to DS register so that the processor uses the address in DS for addressing DATASEG.
  - Assign CODESEG to the CS register so that the processor uses the address in CS for addressing CODESEG.
- When the loading a program for disk into memory for execution, the program loader sets the correct segment addresses in SS and CS.

#### Program written using simplified segment directives:

.Model memory model  
Memory model can be  
TINY, SMALL, MEDIUM, COMPACT, LARGE, HUGE or FLAT  
TINY for .com program  
FLAT for program up to 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

#### ALP written in simplified 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  

    MOV DS, AX ; set address of data segment in DS  

    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

```

#### DOS Debug (TASM)

- 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) > tasm filename.asm      makes.obj
- 4) > link filename      makes.exe
- 5) > filename.exe      run the code
- 6) > td filename.exe      debug the code [use F7 and F8]

#### Assembling, Linking and Executing

- 1) **Assembling:**
  - Assembling converts source program into object program if syntactically correct and generates an intermediate .obj file or module.
  - It calculates the offset address for every data item in data segment and every instruction in code segment.
  - A header is created which contains the incomplete address in front of the generated obj module during the assembling.
  - Assembler complains about the syntax error if any and does not generate the object module.
  - Assembler creates .obj .lst and .crf files and last two are optional files that can be created at run time.
  - For short programs, assembling can be done manually where the programmer translates each mnemonic into the machine language using lookup table.
  - Assembler reads each assembly instruction of a program as ASCII character and translates them into respective machine code.

#### Assembler Types:

There are two types of assemblers:

##### a) One pass assembler:

This assembler scans the assembly language program once and converts to object code at the same time.

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Peter Abel | 15

- This assembler has the program of defining forward references only.
- The jump instruction uses an address that appears later in the program during scan, for that case the programmer defines such addresses after the program is assembled.

##### b) Two pass assembler

- This type of assembler scans the assembly language twice.
- First pass generates symbol table of names and labels used in the program and calculates their relative address.
- This table can be seen at the end of the list file and here user need not define anything.
- Second pass uses the table constructed in first pass and completes the object code creation.
- This assembler is more efficient and easier than earlier.

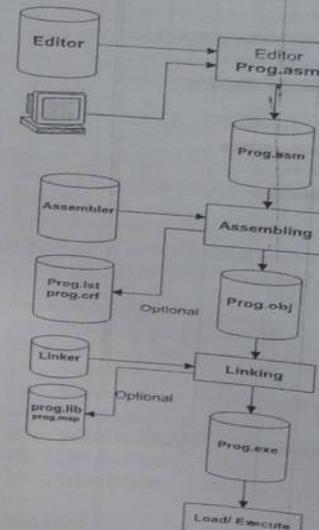


Fig: Steps in assembling, linking & Executing

##### 2) **Linking:**

- This involves the converting of .OBJ module into .EXE(executable) module i.e. executable machine code.
- It completes the address left by the assembler.

Compiled by: Er. Hari Aryal      Email: haryal4@gmail.com

Reference: Peter Abel | 16

- It combines separately assembled object files.
- Linking creates .EXE, .LIB, .MAP files among which last two are optional files.

### 3) Loading and Executing:

- It Loads the program in memory for execution.
- It resolves remaining address.
- This process creates the program segment prefix (PSP) before loading.
- It executes to generate the result.

Sample program assembling → object Program linking → executable program

### Writing .COM programs:

- It fits for memory resident programs.
- Code size limited to 64K.
- .com combines PSP, CS, DS in the same segment
- SP is kept at the end of the segment (FFFF), if 64k is not enough, DOS Places stack at the end of the memory.
- The advantage of .com program is that they are smaller than .exe program.
- A program written as .com requires ORG 100H immediately following the code segment's SEGMENT statement. The statement sets the offset address to the beginning of execution following the PSP.

```
.MODEL TINY
.CODE
ORG 100H ; start at end of PSP
BEGIN:JMP MAIN ;Jump Past data
    VAL1 DW 5491
    VAL2 DW 372
    SUM DW ?
MAIN: PROC NEAR
    MOV Ax, VALL
    ADD AX, VAL2
    MOV SUM, AX
    MOV AX, 4C00H
    INT 21H
MAIN ENDP
END BEGIN
```

### Macro Assembler:

- A macro is an instruction sequence that appears repeatedly in a program assigned with a specific name.
- The macro assembler 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 purposes:
  - o To simplify and reduce the amount of repetitive coding.
  - o To reduce errors caused by repetitive coding.
  - o To make an assembly language program more readable.
  - o Macro executes faster because there is no need to call and return.
  - o Basic format of macro definition:

|            |                        |                |
|------------|------------------------|----------------|
| Macro name | MACRO [Parameter list] | ; Define macro |
|------------|------------------------|----------------|

|                |              |
|----------------|--------------|
| [Instructions] | ; Macro body |
|----------------|--------------|

|      |                |
|------|----------------|
| ENDM | ; End of macro |
|------|----------------|

E.g. Addition

|              |  |
|--------------|--|
| MACRO        |  |
| IN AX, PORT  |  |
| ADD AX, BX   |  |
| OUT PORT, AX |  |
| ENDM         |  |

P.35

### Passing argument to MACRO:

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

```
Addition MACRO VALL1, VAL2
    MOV AX, VALL1
    ADD AX, VAL2
    MOV SUM, AX
    ENDM
```

```
.MODEL SMALL
STACK 64
.DATA
    VAL1 DW 3241
    VAL2 DW 571
    SUM DW ?
.CODE
MAIN PROC FAR
    MOV AX, @ DATA
    MOV DS, AX
```

```
Addition VAL1, VAL 2
MOV AX, 4C00H
INT 21H
MAIN ENDP
END MAIN
```

#### Addressing modes in 8086:

Addressing modes describe types of operands and the way in which they are accessed for executing an instruction. An operand address provides source of data for an instruction to process an instruction to process. An instruction may have from zero to two operands. For two operands first is destination and second is source operand. The basic modes of addressing are register, immediate and memory which are described below.

##### 1) Register Addressing:

For this mode, a register may contain source operand, destination operand or both.  
 E.g.    MOV AH, BL  
        MOV DX, CX

##### 2) Immediate Addressing

In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes. This mode contains a constant value or an expression.  
 E.g.    MOV AH, 35H  
        MOV BX, 7A25H

##### 3) Direct memory addressing:

In this type of addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it. One of the operand is the direct memory and other operand is the register.  
 E.g. ADD AX, [5000H]

Note: Here data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the Offset address and content of DS as segment address. The effective address, here, is  $10H \cdot DS + 5000H$ .

##### 4) Direct offset addressing

In this addressing, a variation of direct addressing uses arithmetic operators to modify an address.

E.g.    ARR DB 15, 17, 18, 21
   
 MOV AL, ARR[2]      ; MOV AL, 18
   
 ADD BH, ARR+3      ; ADD BH, 21

##### 5) Indirect memory addressing:

Indirect addressing takes advantage of computer's capability for segment: offset addressing. The registers used for this purpose are base register (BX and BP) and index register (DI and SI).

E.g.    MOV [BX], AL

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Peter Abel | 19 |

ADD CX, [SI]

##### 6) Base displacement addressing:

This addressing mode also uses base registers (BX and BP) and index register (SI and DI), but combined with a displacement (a number or offset value) to form an effective address.

E.g.    MOV BX, OFFSET ARR
   
        ≡ LEA BX, ARR
   
 MOV AL, [BX +2]
   
 ADD TBL [BX], CL
   
 TBL [BX] 

##### 7) Base index addressing:

This addressing mode combines a base registers (BX or BP) with an index register (SI or DI) to form an effective address.

E.g.    MOV AX, [BX +SI]
   
 ADD [BX+DI], CL

##### 8) Base index with displacement addressing

This addressing mode, a variation on base- index combines a base register, an index register, and a displacement to form an effective address.

E.g.    MOV AL, [BX+SI+2]
   
 ADD TBL [BX +SI], CH

##### 9) String addressing:

This mode uses index registers, where SI is used to point to the first byte or word of the source string and DI is used to point to the first byte or word of the destination string, when string instruction is executed. The SI or DI is automatically incremented or decremented to point to the next byte or word depending on the direction flag (DF).

##### Examples:

```
TITLE Program to add ten numbers
.MODEL SMALL
.STACK 64
.DATA
ARR DB 73, 91, 12, 15, 79, 94, 55, 89
SUM DW ?
.CODE
MAIN PROC FAR
MOV AX, @DATA
MOV DS, AX
```

Compiled by: Er. Hari Aryal      Email: haryal4@gmail.com

Reference: Peter Abel | 20 |

```

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

```

#### DOS FUNCTIONS AND INTERRUPTS (KEYBOARD AND VIDEO PROCESSING)

The Intel CPU recognizes two types of interrupts namely hardware interrupt when a peripheral devices needs attention from the CPU and software interrupt that is call to a subroutine located in the operating system. The common software interrupts used here are INT 10H for video services and INT 21H for DOS services.

##### INT 21H:

It is called the DOS function call for keyboard operations follow the function number. The service functions are listed below:

###### # 00H - It terminates the current program.

- Generally not used, function 4CH is used instead.

###### # 01H - Read a character with echo

- Wait for a character if buffer is empty
- Character read is returned in AL in ASCII value

###### # 02H - Display single character

- Sends the characters in DL to display
- MOV AH, 02H
- MOV DL, 'A' ; move DL, 65
- INT 21H

###### # 03H and 04H - Auxiliary input/output

- INT 14H is preferred.

###### # 05H - Printer service

- Sends the character in DL to printer
- # 06H - Direct keyboard and display
- Displays the character in DL.

###### # 07H - waits for a character from standard input

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Peter Abel | 21

- does not echo
- # 08H - keyboard input without echo
  - Same as function 01H but not echoed.
- # 09H - string display
  - Displays string until '\$' is reached.
  - DX should have the address of the string to be displayed.
- # 0AH - Read string
- # OBH - Check keyboard status
  - Returns FF in AL if input character is available in keyboard buffer.
  - Returns 00 if not.
- # OC - Clear keyboard buffer and invoke input functions such as 01, 06, 07, 08 or 0A.
  - AL will contain the input function.

#### INT 21H Detailed for Useful Functions

##### # 01H

- MOV AH, 01H; request keyboard input INT 21H  
  - Returns character in AL. If AL= nonzero value, operation echoes on the screen. If AL= zero means that user has pressed an extended function key such as F1 OR home.

##### # 02H

- MOV AH, 02H; request display character  
 MOV DL, CHAR; character to display  
 INT 21H  
  - Display character in DL at current cursor position. The tab, carriage return and line feed characters act normally and the operation automatically advances the cursor.

##### # 09H

- MOV AH, 09H; request display  
 LEA DX, CUST\_MSG; local address of prompt  
 INT 21H  
 CUST\_MSG DB "Hello world", '\$'  
  - Displays string in the data area, immediately followed by a dollar sign (\$) or 24H, which uses to end the display.

##### # OAH

- MOV AH, OAH ; request keyboard input  
 LEA DX, PARA\_LIST ; load address of parameter list  
 INT 21H

Parameter list for keyboard input area :  
 PARA\_LIST LABEL BYTE; start of parameter list  
 MAX\_LEN DB 20; max. no. of input character

Compiled by: Er. Hari Aryal Email: haryal4@gmail.com

Reference: Peter Abel | 22

- ACT\_LEN DB ? ; actual no of input characters  
 KB\_DATA DB 20 DUP (' '); characters entered from keyboard
- LABEL directive tells the assembler to align on a byte boundary and gives location the name PARA\_LIST.
  - PARA\_LIST & MAX\_LEN refer same memory location, MAX\_LEN defines the maximum no of defined characters.
  - ACT\_LEN provides a space for the operation to insert the actual no of characters entered.
  - KB\_DATA reserves spaces (here 20) for the characters.

**Example:**

```

TITLE to display a string
.MODEL SMALL
.STACK 64
.DATA
    STR DB 'programming is fun', '$'
.CODE
MAIN PROC FAR
    MOV AX, @DATA
    MOV DS, AX
    MOV AH, 09H      ;display string
    LEA DX, STR
    INT 21H
    MOV AX, 4C00H
    INT 21H
MAIN ENDP
END MAIN
    
```

**INT 10H**

It is called video display control. It controls the screen format, color, text style, making windows, scrolling etc. The control functions are:

```

# 00H - set video mode
    MOV AH, 00H      ; set mode
    MOV AL, 03H      ; standard color text
    INT 10H          ; call interrupt service

# 01H- set cursor size
    MOV AH, 01H
    MOV CH, 00H      ; Start scan line
    MOV CL, 14H      ; End scan line
    INT 10H          ; (Default size 13:14)
    
```

**# 02H - Set cursor position:**

```

    MOV AH, 02H
    MOV BH, 00H      ; page no
    MOV DH, 12H      ; row/y (12)
    MOV DL, 30H      ; column/x (30)
    INT 10H
    
```

**# 03H - return cursor status**

```

    MOV AH, 03H
    MOV BH, 00H
    INT 10H
    Returns: CH- starting scan line, CL-end scan line, DH- row, DL-column
    
```

**# 04H- light pen function**

```

# 05H- select active page
    MOV AH, 05H
    MOV AL,page-no.   ; page number
    INT 10H
    
```

```

# 06H- scroll up screen
    MOV AX, 060FH      ; request scroll up one line (text)
    MOV BH, 61H          ; brown, background, blue foreground
    MOV CX, 0000H          ; from 00:00 through
    MOV DX, 184F H        ; to 24:79 (full screen)
    INT 10H
    AL= number of rows (00 for full screen)
    BH= Attribute or pixel value
    CX= starting row: column
    DX= ending row: column
    
```

```

# 07H-Scroll down screen
Same as 06H except for down scroll
    
```

```

# 08H (Read character and Attribute at cursor)
    MOV AH, 08H
    MOV BH, 00H      ; page number 0(normal)
    INT 10H
    AL= character
    BH= Attribute
    
```

```

# 09H -display character and attribute at cursor
    MOV AH, 09H
    MOV AL, 01H      ; ASCII for happy face display
    
```

```
MOV BH, 00H      ; page number
MOV BL, 16H      ; Blue background, brown foreground
MOV CX, 60       ; No of repeated character
INT 10H
```

#### # 0AH- display character at cursor

```
MOV AH, 0AH
MOV AL, Char
MOV BH, page_no
MOV BL, value
MOV CX, repetition
INT 10H
```

#### # 0BH- Set color palette

- ✓ Sets the color palette in graphics mode
- ✓ Value in BH (00 or 01) determines purpose of BL
- ✓ BH= 00H, select background color, BL contains 00 to OFH (16 colors)
- ✓ BH = 01H , select palette, BL, contains palette

```
MOV AH, 0BH
```

```
MOV AH, 0BH
MOV BH, 00H; background
MOV BL, 04H; red
INT 21H
```

```
MOV BH, 01H ; select palette
MOV BL, 00H ; black
INT 21H
```

#### #0CH- write pixel Dot

- Display a selected color
- AL= color of the pixel
- CX= column
- BH= page number
- DX= row

```
MOV AH, 0CH
MOV AL, 03
MOV BH, 0
MOV CX, 200
MOV DX, 50
INT 10H
It sets pixel at column 200, row 50
```

#### #0DH- Read pixel dot

- Reads a dot to determine its color value which returns in AL

```
MOV AH, 0DH
MOV BH, 0 ; page no
MOV CX, 80 ; column
MOV DX, 110 ; row
INT 10H
```

#0EH- Display in teletype mode  
Use the monitor as a terminal for simple display

```
MOV AH, 0EH
MOV AL, char
MOV BL, color;foreground color
INT 10H
```

#OF H- Get current video mode  
Returns values from the BIOS video .  
AL= current video mode      MOV AH, 0FH  
AH= no of screen columns    INT 10H  
BH = active video page

#### TITLE To Convert letters into lower case

```
.MODEL SMALL
STACK 99H
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV SI, OFFSER STR
    M: MOV DL, [SI]
    MOV CL, DL
    CMP DL, 'S'
    JE N
    CMP DL, 60H
    JLL K:
    MOV DL, CL
    MOV AH, 02H
    INT 21H
    INC SI
    JMP M
    L: MOV DL, CL
    ADD DL, 20H
    MOV AH, 02H
    INT 21H
    INC SI
    JMP M
    N: MOV AX, 4C00H
    INT 21H
    MAIN ENDP
.DATA
STR DB "I am MR Rahul ", '$'
```

```

        END MAIN

TITLE to reverse the string
.MODEL SMALL
.STACK 100H
.DATA
    STR1 DB " My name is Rahul" , '$'
    STR2 db 50 dup ('$')
.CODE
MAIN PROC FAR
    MOV BL,00H
    MOV AX, @ DATA
    MOV DS, AX
    MOV SI, OFFSET STR1
    MOV DI, OFFSET STR2
L2:   MOV DL, [SI]
    CMP DL, '$'
    JE L1
    INC SI
    INC BL
    JMP L2
L1:   MOV CL, BL
    MOV CH, 00H
    DEC SI
L3:   MOV AL, [SI]
    MOV [DI], AL
    DEC SI
    INC DI
    LOOP L3
    MOV AH,09H
    MOV DX, OFFSET STR2
    INT 21H
    MOV AX, 4C00H
    INT 21H
MAIN ENDP
END MAIN

```

```

TITLE to input characters until 'q' and display
.MODEL SMALL
.STACK 100H
.DATA
    STR db 50 DUP ('$')
.CODE
MAIN PROC FAR

```

Compiled by: Er. Hari Aryal Email: haryal4@gmail.com

Reference: Peter Abel | 27

```

    MOV AX, @ DATA
    MOV DS, AX
    MOV SI, OFFSET STR
L2:   MOV AH,01H
    INT 21H
    CMP AL, 'q'
    JE L1
    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.

Compiled by: Er. Hari Aryal Email: haryal4@gmail.com

Reference: Peter Abel | 28

```
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 ENDP
```

#### INT 10H Video service:

#### <Video -modes>

| Text mode       | Row x column | Color      | No. of Pages | Resolution | colors    |
|-----------------|--------------|------------|--------------|------------|-----------|
| 00              | 25x40        | Color      | 8            | 360x400    | 16 colors |
| 01              | 25x40        | Color      | 8            | 360x400    | 16 colors |
| 02              | 25x80        | Color      | 4            | 720x400    | 16 colors |
| 03 (by default) | 25x80        | color<br>↓ | 4            | 720x400    | 16 colors |
| 07              | 25x80        | Mono-hrome | 0            | 720x400    | 16 colors |

| Graphic mode | Color       | Pages | Resolution | No of colors |
|--------------|-------------|-------|------------|--------------|
| 04           | Color       | 8     | 320x200    | 4            |
| 05           | Color       | 8     | 320x200    | 4            |
| 06           | Color       | 8     | 640x200    | 2            |
| 0D           | Color       | 8     | 320x200    | 16           |
| 0E           | Color       | 4     | 640x200    | 16           |
| 0F           | Mono chrome | 2     | 640x350    | 1            |
| 10           | Color       | 2     | 640x350    | 16           |
| 11           | Color       | 1     | 640x480    | 2            |
| 12           | Color       | 1     | 640x480    | 16           |
| 13           | Color       | 1     | 320x200    | 256          |

#### Attribute

Attribute: BL R G B  
 Bit number: 7 6 5 4

Background  
 I R G B  
 3 2 1 0

I – Intensity, BL - Blink

| Color         | Hex Value |
|---------------|-----------|
| Black         | 0         |
| Blue          | 1         |
| Green         | 2         |
| Cyan          | 3         |
| Red           | 4         |
| Magenta       | 5         |
| Brown         | 6         |
| White         | 7         |
| Gray          | 8         |
| Light Blue    | 9         |
| Light Green   | A         |
| Light cyan    | B         |
| Light red     | C         |
| Light magenta | D         |
| Yellow        | E         |
| Bright white  | F         |

TITLE sorting the numbers – descending order  
dosseg

```

    .MODEL SMALL
    .STACK 100H
    .CODE

MAIN PROC FAR
    MOV AX, @ DATA
    MOV DS, AX
    MOV DX, AH

DOPASS: MOV CX, AH
    MOV SI, 00H

CHECK: MOV AL, ARR [SI]
    CMP ARR [SI+1], AL
    JC NOSWAP
    MOV BL, ARR [SI + 1]
    MOV ARR [SI+1], AL
    MOV ARR [SI], BL

NOSWAP: INC SI
    LOOP CHECK
    DEC DX
    JNZ DOPASS
    MOV AX, 4000H
    INT 21H
    MAIN ENDP

    .DATA
    ARR DB 8,2,9,4,7
END MAIN

```

Note: Display if numbers are with 1 digit

```

L:    MOV CX, 05H
        MOV SI, 00H
        MOV DL, ARR[SI]
        ADD DL, 30H
        MOV AH, 02H
        INT 21H
        MOV DL, ''
        MOV AH, 02H
        INT 21H
        INC SI
        LOOP L
        MOV AX, 4000H
        INT 21H
    MAIN ENDP

```

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Peter Abel | 31

TITLE addition of 100 natural even numbers

```

    .MODEL SMALL
    .STACK 100H
    .DATA
    TEN DW 10

.MAIN PROC FAR
    MOV AX, @ DATA
    MOV DS, AX
    MOV CX, 63H
    MOV AX, 02H
    MOV DX, 04H

L1:   ADD AX, DX
    ADD DX, 02H
    LOOP L1

L2:   MOV DX, 0000H
    DIV TEN           ; DX: AX / 10
    INC CX
    ADD DX, 30H      ; remainder
    PUSH DX
    CMP AX, 00H      ; quotient
    JE L3
    JMP L2
    L3:   POP DX
    MOV AH, 02H
    INT 21H
    LOOP L3
    MOV AX, 4000H
    INT 21H
    MAIN ENDP
END MAIN

```

TITLE to display string at (10,40) with green background and red foreground  
dosseg

```

.Model small
.Stack 100H
.Code
MAIN PROC FAR
    MOV AX, @ DATA
    MOV DX, AX
    MOV SI, OFFSET VAR1
    MOV AH, 02H          ; Set cursor position
    MOV DH, ROW
    MOV DL, COL

```

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: Peter Abel | 32

```

INT 10H
MOV AL, [SI]
CMP AL, '$'
JE L1
MOV AH, 09H
MOV DH, ROW
MOV DL, COL
MOV BL, 24H
MOV BH, 00h
MOV CX, 01H
INT 10H
INC SI
INC COL
JMP L2
L1: MOV AX, 4C00H
INT 21H
MAIN ENDP
.DATA
ROW DB 10
COL DB 40
VARI DB "video model", '$'
END MAIN
    
```

**TITLE TO GENERATE MULTIPLICATION TABLE**

```

.MODEL SMALL
STACK 32
.DATA
    NUM1 DB 5
    NUM2 DB 1
    TAB DB 10 DUP (?)
.CODE
MAIN PROC FAR
    MOV AX, @ DATA
    MOV DS, AX
    MOV BX, 0
    MOV CX, 10
L1: MOV AL, NUM1
    MUL NUM2
    MOV TAB [BX], AL
    INC BX
    INC NUM2
    LOOP L1
    MOV AX, 4C00H
    
```

```

INT 21H
MAIN ENDP
END MAIN
    
```

```

TITLE to add 10 sixteen bit Numbers in memory table
.MODEL SMALL
STACK 32
.DATA
NUM DW DUP (2)
NUM DW DUP (3)
SUMH DW 0
SUML DW 0
.CODE
    
```

Note: To access the data of the  
memory i.e. table.  
We use e.g. NUM[BX]  
Increasing the BX register by 2

```

MAIN PROC FAR
    MOV AX, @ DATA
    MOV DS, AX
    MOV CX, 10
    MOV AX, 0
    MOV BX, 0
L1: ADD AX, NUM [BX]
    MOV SUML, AX
    JNC L2
    INC SUMH
L2: ADD BX, 2
    LOOP L1
    MOV AX, 4C00H
    INT 21H
MAIN ENDP
END MAIN
    
```

**SUBROUTINE TO CLEAR THE SCREEN**

```

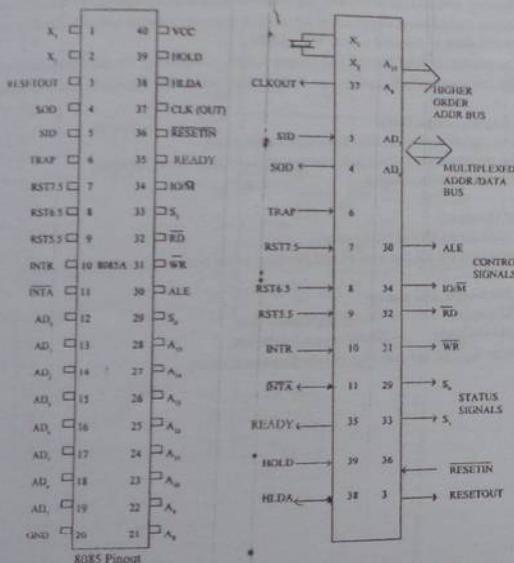
SCR_CLEAR PROC NEAR
    MOV AX, 0600H      ; Request scroll
    MOV BH, 61H ; blue on brown for attribute on pixel(generally 07H) white on black
    MOV CX, 0000      ; Full screen
    MOV DX, 184FH
    INT 10H
    RET
SCR_CLEAR ENDP
    
```

→ AH-06h: Scroll upward of lines in a specified area of the screen.  
AL-00H caused entire screen to scroll up, effectively clearing it. Setting a nonzero value in AL causes the number of lines to scroll up.

## Chapter-4 Microprocessor System

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.

### PIN Configuration of 8085



- The microprocessor is a clock-driven semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale integration (LSI) or very-large-scale integration (VLSI) technique.
- The microprocessor is capable of performing various computing functions and making decisions to change the sequence of program execution.
- In large computers, a CPU implemented on one or more circuit boards performs these computing functions.

- The microprocessor is in many ways similar to the CPU, but includes the logic circuitry, including the control unit, on one chip.
- The microprocessor can be divided into three segments for the sake clarity, arithmetic/logic unit (ALU), register array, and control unit.
- 8085 is a 40 pin IC, DIP package. The signals from the pins can be grouped as follows

1. Power supply and clock signals
2. Address bus
3. Data bus
4. Control and status signals
5. Interrupts and externally initiated signals
6. Serial I/O ports

### 1. Power supply and Clock frequency signals:

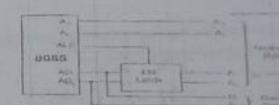
- Vcc + 5 volt power supply
- Vss Ground
- X1, X2 – Crystal or R/C network or LC network connections to set the frequency of internal clock generator
- The frequency is internally divided by two. Since the basic operating timing frequency is 3 MHz, a 6 MHz crystal is connected externally.
- CLK (output)-Clock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

### 2. Address Bus:

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

### 3. Multiplexed Address / Data Bus:

- A00 - AD7
- These multiplexed set of lines used to carry the lower order 8 bit address as well as data bus
- During the opcode fetch operation, in the first clock cycle, the lines deliver the lower order address A0 - A7
- In the subsequent IO / memory, read / write clock cycle the lines are used as data bus
- The CPU may read or write out data through these lines



#### 4. Control and Status signals:

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

- ALE (output) - Address Latch Enable
  - This signal helps to capture the lower order address presented on the multiplexed address / data bus. When it is the pulse, 8085 begins an operation. It generates  $AD_0 - AD_7$  as the separate set of address lines  $A_0 - A_7$ .
- RD (active low) - Read memory or I/O device.
  - This indicates that the selected memory location or I/O device is to be read and that the data bus is ready for accepting data from the memory or I/O device.
- WR (active low) - Write memory or I/O device.
  - This indicates that the data on the data bus is to be written into the selected memory location or I/O device.
- IO/M (output) - Select memory or an I/O device.
  - This status signal indicates that the read / write operation relates to whether the memory or I/O device
  - It goes high to indicate an I/O operation
  - It goes low for memory operations.

#### 5. Status Signals:

It is used to know the type of current operation of the microprocessor.

| Register | M1 | S2 | Data Bus Status (Output) |
|----------|----|----|--------------------------|
| 0        | 0  | 0  | Idle                     |
| 0        | 0  | 1  | Memory WRITE             |
| 0        | 1  | 0  | Memory READ              |
| 1        | 0  | 1  | WR PULSE                 |
| 1        | 1  | 0  | RD PULSE                 |
| 0        | 1  | 1  | (NO-OP)                  |
| 1        | 1  | 1  | Interrupt acknowledge    |

#### 6. Interrupts and Externally initiated operations:

- They are the signals initiated by an external device to request the microprocessor to do a particular task or work.
- There are five hardware interrupts called,
  - TRAP
  - INT 05
  - INT 08
  - INT 10
  - INT 19  
DCLI (DMA line control)
- On receipt of an interrupt, the microprocessor acknowledges the interrupt by the active low INTA (Interrupt Acknowledge) signal.



#### Hold (Input)

- This indicates peripheral controller requesting the bus.

#### HLDA (Output)

- This indicates the acknowledgement for the Hold request.

#### READY (Input)

- It is used to delay the microprocessor read and write cycles until a slow responding peripheral is ready to send or accept data.
- Memory and I/O devices will have slower response compared to microprocessors.
- Before completing the present job such a slow peripheral may not be able to handle further data or control signal from CPU.
- The processor sets the READY signal after completing the present job to access the data.
- The microprocessor enters into WAIT state while the READY pin is disabled.

#### Reset In (input, active low)

- This signal is used to reset the microprocessor.
- The program counter inside the microprocessor is set to zero.
- The buses are tri-stated.

#### Reset Out (Output)

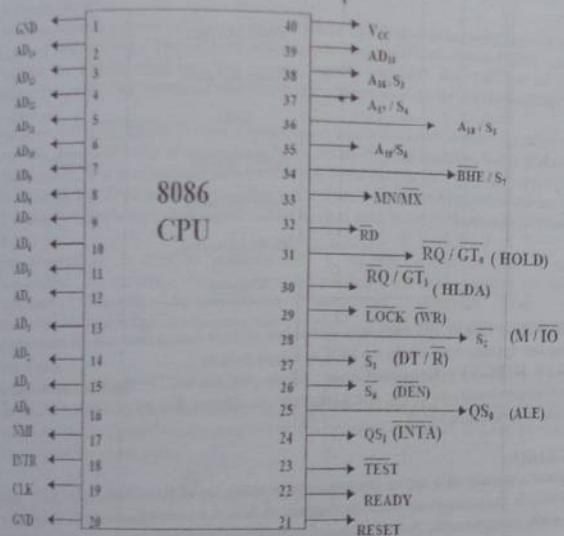
- It indicates CPU is being reset.
- Used to reset all the connected devices when the microprocessor is reset.

#### 7. Single Bit Serial I/O ports:

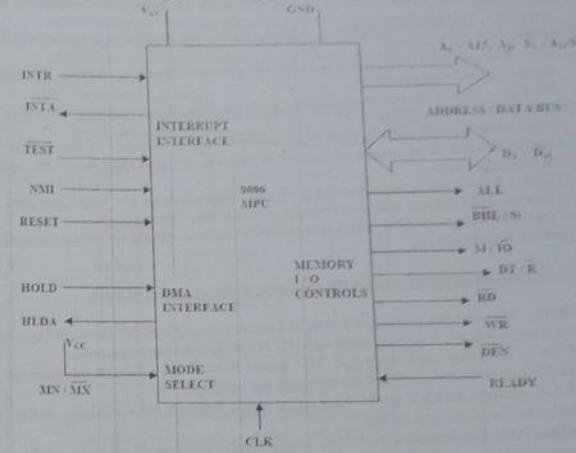
- SID (input) - Serial input data line
- SOD (output) - Serial output data line
- These signals are used for serial communication.

Pin Configuration of 8086

Pin Diagram of 8086



Signal Groups of 8086



- The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
- The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).
- The 8086 signals can be categorized in three groups
  - The first are the signal having common functions in minimum as well as maximum mode.
  - The second are the signals which have special functions for minimum mode.
  - The third are the signals having special functions for maximum mode.

The following signal descriptions are common for both modes.

- AD15-AD0 : These are the time multiplexed memory I/O address and data lines.
- A19/S6,A18/S5,A17/S4,A16/S3 : These are the time multiplexed address and status lines. The address bits are separated from the status bit using latches controlled by the ALE signal.
- BHE/S7 : The bus high enable is used to indicate the transfer of data over the higher order ( D15-D8 ) data bus.
- RD – Read : This signal on low indicates the peripheral that the processor is performing memory or I/O read operation.
- READY : This is the acknowledgement from the slow device or memory that they have completed the data transfer.
- INTR-Interrupt Request : This is to determine the availability of the request from external devices. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.
- 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.
- CLK- Clock Input : The clock input provides the basic timing for processor operation and bus control activity.

The following pin functions are for the minimum mode operation of 8086.

- M/IO – Memory/IO : When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation.
- INTA – Interrupt Acknowledge: This signal is used for interrupt acknowledge i.e. when it goes low, the processor has accepted the interrupt.
- ALE – Address Latch Enable: This output signal indicates the availability of the valid address on the address/data lines.
- DT/R – Data Transmit/Receive: This output is used to decide the direction of data flow through the trans-receivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.
- DEN – Data Enable: This signal indicates the availability of valid data over the address/data lines. It is used to enable the trans-receivers (bidirectional buffers) to separate the data from the multiplexed address/data signal.
- HOLD, HLDA- Acknowledge: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on.

The following pin functions are applicable for maximum mode operation of 8086.

- S2, S1, S0 – Status Lines: These are the status lines which reflect the type of operation being carried out by the processor.

| S2 | S1 | S0 | Indication            |
|----|----|----|-----------------------|
| 0  | 0  | 0  | Interrupt Acknowledge |
| 0  | 0  | 1  | Read I/O port         |
| 0  | 1  | 0  | Write I/O port        |
| 0  | 1  | 1  | Halt                  |
| 1  | 0  | 0  | Code Access           |
| 1  | 0  | 1  | Read Memory           |
| 1  | 1  | 0  | Write Memory          |
| 1  | 1  | 1  | Passive               |

- LOCK : This output pin indicates that other system bus master will be prevented from gaining the system bus, while the LOCK signal is low.
- RQ/GT0, RQ/GT1 – Request/Grant : These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.

**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.

**I. 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, whereas 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.

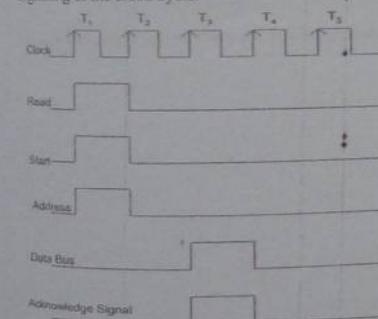


Fig: Synchronous Read Operation

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

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar & D. V. Hall |



- 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 acknowledgement 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.

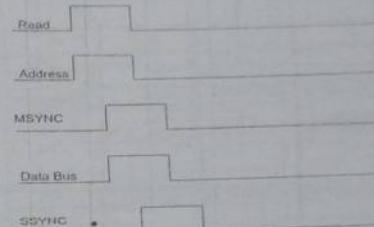


Fig: Asynchronous Read Operation

- 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.

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar & D. V. Hall |

**Instruction Cycle:** It is defined as the time required completing the execution of an instruction. The 8085 instruction cycle consists of one to six machine cycles or one to six operations.

#### 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.

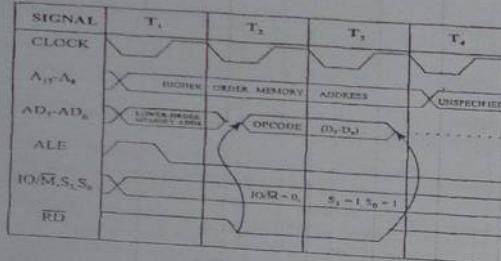


Fig - Timing Diagram for Opcode Fetch Machine Cycle

**Step 1:** Microprocessor places the 16 bit memory address from Program Counter on the address bus. At T<sub>1</sub>, high order address (20) is placed at A<sub>8</sub>-A<sub>15</sub> and lower order address (05) is placed at A<sub>7</sub>-A<sub>1</sub>. ALE signal goes high. IO/M goes low and both S<sub>0</sub> and S<sub>1</sub> 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<sub>3</sub> and T<sub>4</sub>.

**Step 3:** The byte from the memory location is placed on the data bus that is 4f into D<sub>0</sub>-D<sub>7</sub> and RD goes high impedance.

**Step 4:** The instruction 4FH is decoded and content of accumulator will be copied into register C during clock cycle T<sub>4</sub>.

#### 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 |     |    |    |   |

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar & D. V. Hall | 11

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.

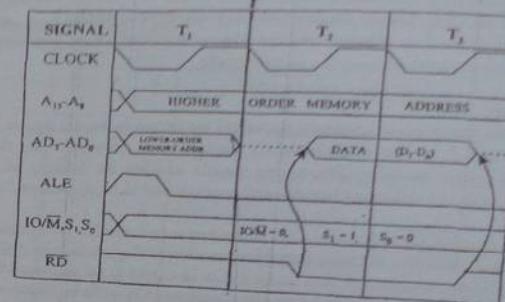


Fig - Timing Diagram for Memory Read Machine Cycle

**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<sub>1</sub>=1, S<sub>0</sub>=0 for memory read cycle. When RD =0, memory places the data byte 32H on the data bus.

#### Memory Write Machine:

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar & D. V. Hall | 12

- The processor takes, 3T states to execute this machine cycle.

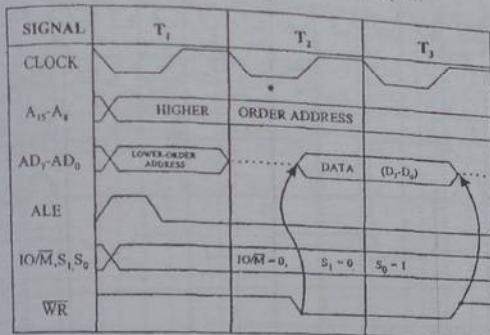


Fig - Timing Diagram for Memory Write Machine Cycle

#### 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.

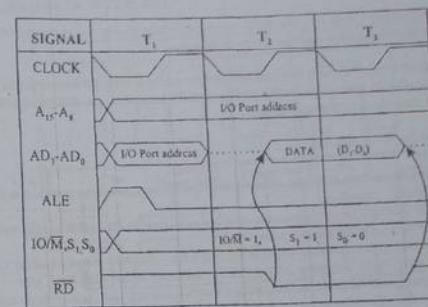


Fig - Timing Diagram for I/O Read Machine Cycle

#### 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

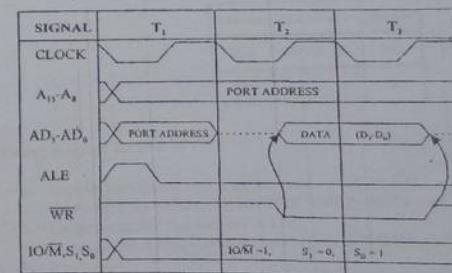


Fig - Timing Diagram for I/O Write Machine Cycle

**Step 1:** In Machine Cycle M<sub>1</sub>, the microprocessor sends  $\overline{RD}$  control signal which is combined with IO/M to generate the MEMR signal and processor fetches instruction code D3 using the data bus.

**Step 2:** In 2<sup>nd</sup> Machine Cycle M<sub>2</sub>, the 8085 microprocessor places the next address 2051 on the address bus and gets the device address 01H via data bus.

**Step 3:** In machine Cycle M<sub>3</sub>, the 8085 places device address 01H on low order as well as high order 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 12H. 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 | Op code         |
|---------|-----------|-----------------|
| 41FF    | STA 526AH | 12H             |
| 4200    |           | 6A <sub>H</sub> |
| 4201    |           | 52 <sub>H</sub> |

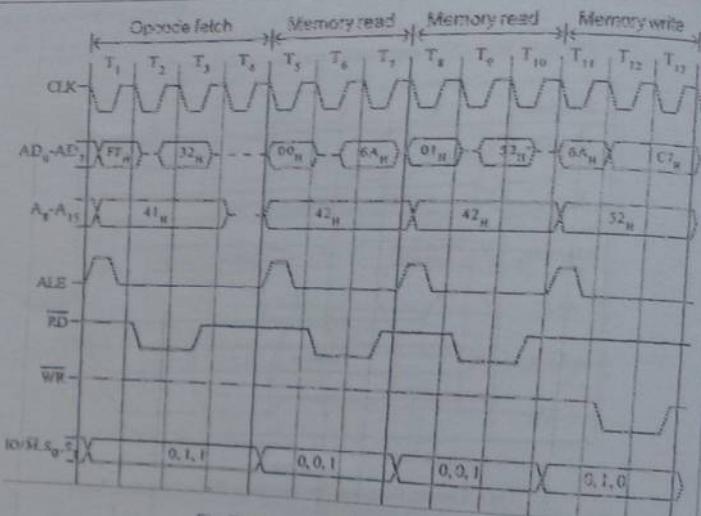


Fig. Timing diagram for STA 526AH

#### Timing diagram for IN C0H

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

| Address | Mnemonics | Op code         |
|---------|-----------|-----------------|
| 4125    | IN C0H    | DB <sub>H</sub> |
| 4126    |           | C0 <sub>H</sub> |

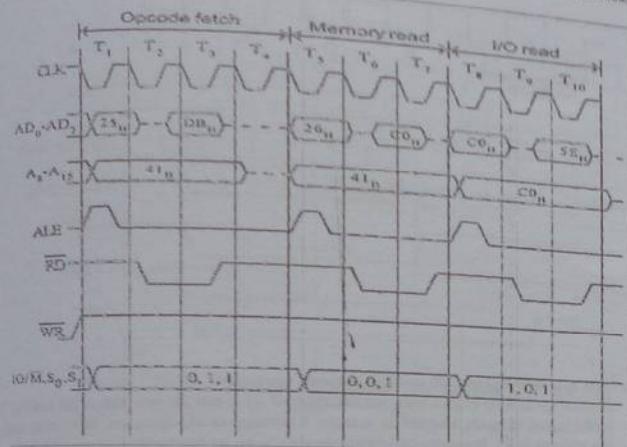
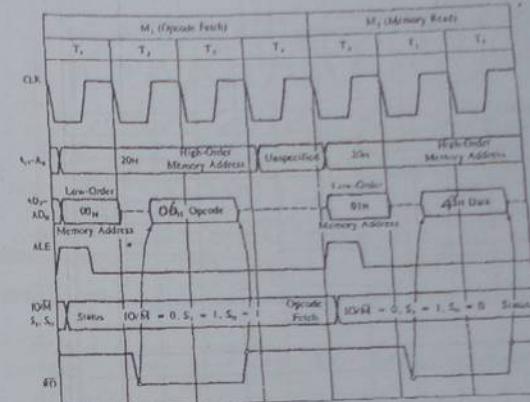


Fig. Timing diagram for IN COH.

Timing diagram for MVI B, 43H.



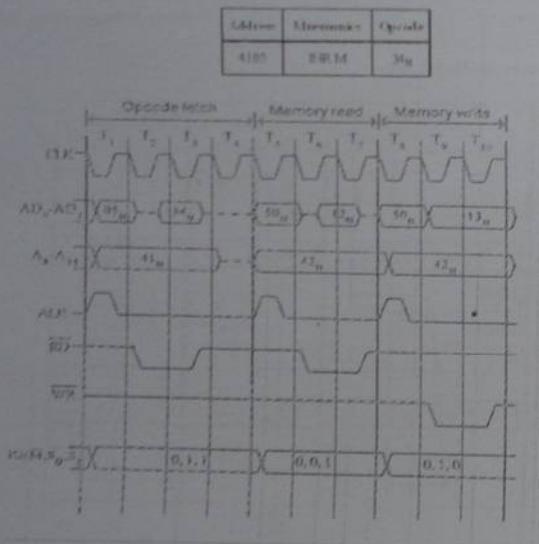
P.53

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

| Address | Machine    | Op code |
|---------|------------|---------|
| 2000    | MVI B, 43H | 06H     |
| 2001    |            | 43H     |

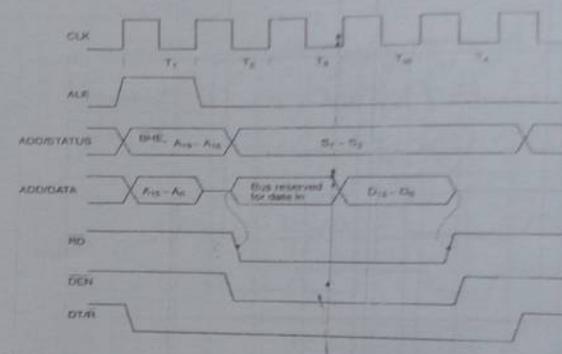
Timing diagram for INR M

- 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)



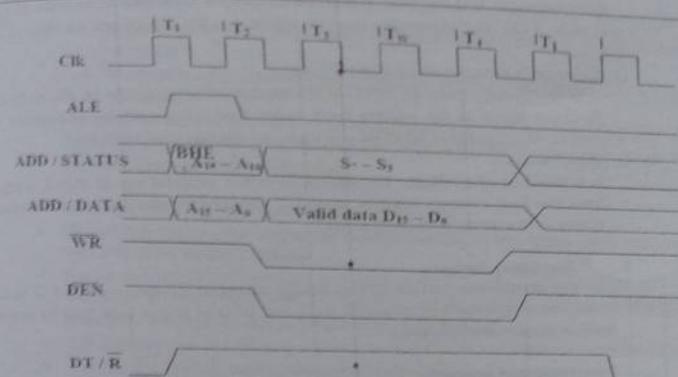
#### Read and write bus timing of 8086 microprocessor

- The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.
- The op-code fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.
- The read cycle begins at T<sub>1</sub> with the assertion of address latch enable (ALE) signal and also M/I/O signal. During the negative going edge of this signal, the valid address is latched on the local bus.
- The BHE and A<sub>0</sub> signals address low, high or both bytes. From T<sub>1</sub> to T<sub>4</sub>, the M/I/O signal indicates a memory or I/O operation.
- At T<sub>2</sub>, the address is removed from the local bus and is sent to the output. The bus is then tri-stated. The read (RD) control signal is also activated in T<sub>2</sub>.
- The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.
- The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.



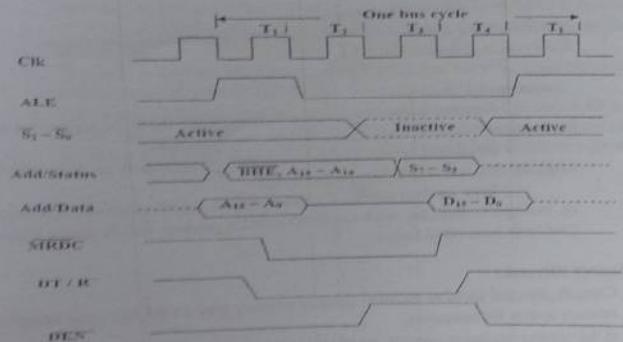
#### Write cycle timing diagram for minimum mode operation

- A write cycle also begins with the assertion of ALE and the emission of the address. The M/I/O signal is again asserted to indicate a memory or I/O operation. In T<sub>2</sub>, after sending the address in T<sub>1</sub>, the processor sends the data to be written to the addressed location.
- The data remains on the bus until middle of T<sub>4</sub> state. The WR becomes active at the beginning of T<sub>2</sub> (unlike RD is somewhat delayed in T<sub>2</sub> to provide time for floating).
- The BHE and A<sub>0</sub> signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
- The M/I/O, RD and WR signals indicate the type of data transfer as specified in table below.



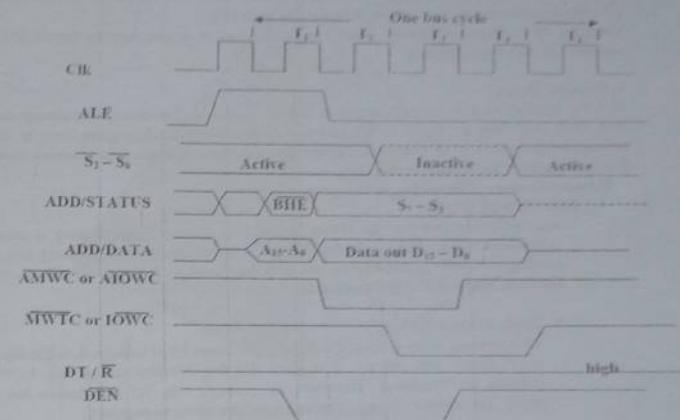
Write Cycle Timing Diagram for Minimum Mode

Memory Read Timing Diagram in Maximum Mode of 8086



Memory Read Timing in Maximum Mode

Memory Write Timing in Maximum mode of 8086



Memory Write Timing in Maximum mode.

Memory devices:

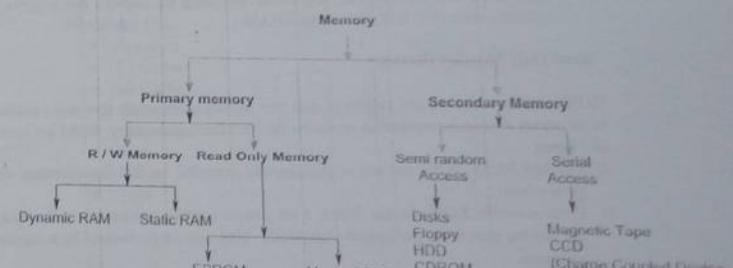


Fig: Classification of memory system

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 into two main groups i.e. Primary memory and Secondary memory.

#### 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 processor memory that is included in the microprocessor chip.

Primary memory is divided into two groups.

##### 1. RW Memory (RAM)

Microprocessor can read for and write into this memory. This memory is used for information that is 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 manufacturer 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.

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.

$$\text{Access rate } (t_a) = \frac{1}{t_a}$$

###### 2. Cycle time ( $t_c$ ):

It is the average time that lapses between two successive read operation.

$$\text{Cycle rate } (t_c) = \text{bandwidth} = 1/t_c$$

##### Access modes of memory:

Random access: In random access mode, the  $t_a$  is independent of the location from which the data is accessed like MOS memory.

Sequential access: In that mode, the  $t_a$  is dependent of the location from which the data is accessed like magnetic type.

Semi random-access: The semi random access combines these two. For eg. In magnetic disk, any track can be accessed at random. But the access within the track must be in serial fashion.

##### The Memory Hierarchy

- Capacity, cost and speed of different types of memory play a vital role while designing a memory system for computers.
- If the memory has larger capacity, more application will get space to run smoothly.
- It's better to have fastest memory as far as possible to achieve a greater performance. Moreover for the practical system, the cost should be reasonable.
- There is a tradeoff between these three characteristics cost, capacity and access time. One cannot achieve all these quantities in same memory module because

- If capacity increases, access time increases (slower) and due to which cost per bit decreases.
- If access time decreases (faster), capacity decreases and due to which cost per bit increases.
- The designer tries to increase capacity because cost per bit decreases and the more application program can be accommodated. But at the same time, access time increases and hence decreases the performance.

So the best idea will be to use memory hierarchy.

- Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system.
- Not all accumulated information is needed by the CPU at the same time.
- Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU
- The memory unit that directly communicate with CPU is called the *main memory*.
- Devices that provide backup storage are called *auxiliary memory*.
- The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory.
- The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
- A special very-high-speed memory called *cache* is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
- CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory.
- The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.
- The memory hierarchy system consists of all storage devices employed in a computer system from slow but high capacity auxiliary memory to a relatively faster cache memory accessible to high speed processing logic. The figure below illustrates memory hierarchy.

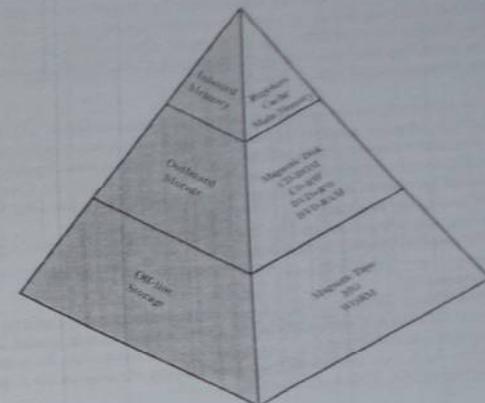


Fig. Memory Hierarchy

- As we go down in the hierarchy
  - Cost per bit decreases
  - Capacity of memory increases
  - Access time increases
  - Frequency of access of memory by processor also decreases

#### Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

### 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 8bit 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 CS signal. It is expensive and complicated but fault proof in all cases.

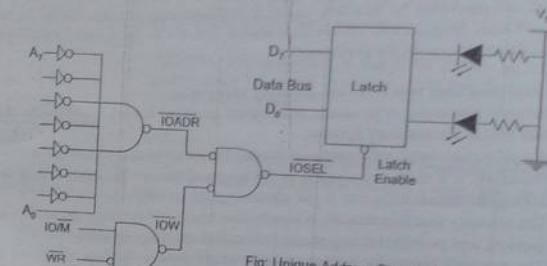


Fig: Unique Address Decoding

- If  $A_0$  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)

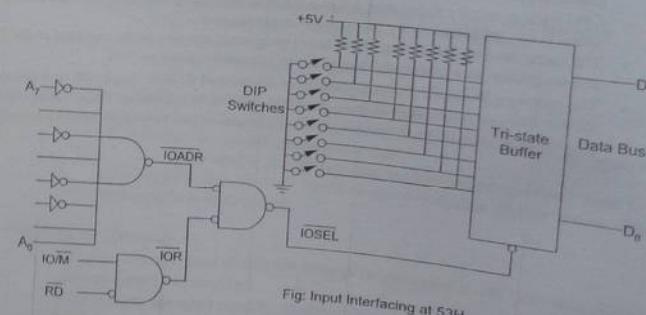


Fig: Input Interfacing at 53H

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.

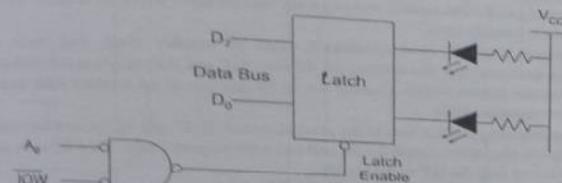


Fig: Non unique Address Decoding

- If  $A_9$  is low and  $TOW$  is low. Then latch gets enabled.
- Here  $A_1-A_7$  is neglected that is any even address can enable the latch.

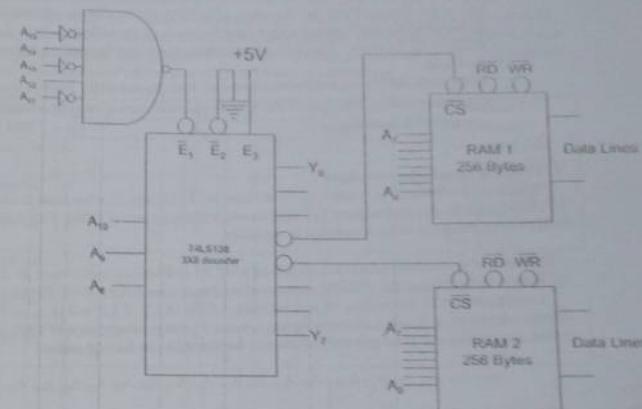
Q) Design an address decoding circuit for two RAM chips each of 256 bytes at address 5300H.

- 256 bytes requires 8 address lines.  
 $2^8=256$ ,  $x=8$

So to address one of 256 bytes in each RAM requires 8 address lines  $A_8-A_0$ .

| Block | Address | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | $A_9$ | $A_8$ | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
|-------|---------|----------|----------|----------|----------|----------|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1     | Start   | 5300H    | 0        | 1        | 0        | 1        | 0        | 0     | 1     | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
|       | End     | 53FFH    | 0        | 1        | 0        | 1        | 0        | 0     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     |
| 2     | Start   | 5400H    | 0        | 1        | 0        | 1        | 0        | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
|       | End     | 54FFH    | 0        | 1        | 0        | 1        | 0        | 0     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     |

Address Decoding Circuit



Q. Draw a circuit diagram to interface two 256 Byte memory chips at address starting at 2050H and 3050H.

Q. Two 4 KB ROM at starting address 0000H  
 $4\text{ KB} = 4 \times 1\text{ KB} = 2^2 \times 2^{10} = 2^{12}$  therefore we need 12 address lines

Address Decoding with 8086 Microprocessor

The 8086 microprocessor provides a 20 bit memory address that allows up to 1 MB main memory. Out of these several address lines are unused, but these extra lines determine the range of addresses the memory interface occupies. Address decoder circuit determines these extra address lines and enables the memory for a specific range of addresses. Depending upon number of lines used for decoder, we get

**Full Decoding (Absolute Decoding):** All the unused lines (zero lines) are used.

**Partial Decoding (Linear Decoding):** All the unused lines (zero lines) are not used.

**Block Decoding:** Same as full decoding except that in this case blocks of memory is enabled using the unused lines.

**Important Points to be considered for Memory Interfacing**

- After reset CS contains FFFFH and IP contains 0000H. Therefore, the physical address is FFFF0H. Here instruction execution starts from FFFF0H which is normally a jump from some other location where a longer program resides. This program always resides in a ROM. For example, we want to interface 4 chips of 2K memory that means 8K bytes of

- memory requires 13 address lines A<sub>0</sub> to A<sub>12</sub>. So, 8K means 01FFFH bytes, therefore EPROM address starts from FFFFFH – 01FFFH = FC000H
- Since ROMs and EPROMs are read-only devices, A<sub>0</sub> and BHE\* are not required to be part of the chip enable/select decoding. The 8086 address lines must be connected to the ROM/EPROM chip starting with A<sub>3</sub> and higher to all the address lines of the ROM/EPROM chips. The 8086 unused address lines can be used as chip enable/select decoding.
- Since static RAMs are read/write memories, both A<sub>0</sub> and BHE\* must be included in the chip select/enable decoding of the devices and write timing must be considered in the compatibility analysis.

#### EPROM Interfacing with 8086

Whenever the 8086 CPU is reset, its value is set to FFFFH and IP value is set to 0000H that corresponds to physical address FFFF0H which is always a part of ROM. This means FFFF0H to FFFF1H should be always included in the ROM.

Let us take an example to address 16 KB of EPROM to 8086 microprocessor. 16 KB means 3FFFH bytes. Hence the EPROM memory should start from FFFFFH – 03FFFH = FC000H. To address 16 KB, we require 14 address lines. Of the 16 KB, 8 KB will be at even addresses and 8 KB will be at odd addresses. Hence, we use 2 EPROM chips, each of 8 KB capacity one for storing bytes at even address and another for storing bytes for odd address.

|        | A <sub>19</sub> A <sub>18</sub> A <sub>17</sub> A <sub>16</sub> A <sub>15</sub> A <sub>14</sub> | A <sub>13</sub> A <sub>12</sub> A <sub>11</sub> A <sub>10</sub> A <sub>9</sub> A <sub>8</sub> A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub> |
|--------|---|---|
| Start: | 1 1 1 1 1 1   | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   |
| End:   | 1 1 1 1 1 1   | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   |

Used for Chip Select      Address within the 16KB

The EPROM address ranges from FC000H to FFFFH

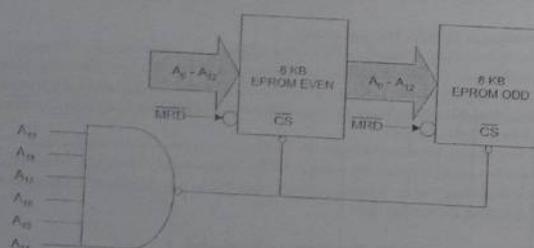


Fig: EPROM Interfacing with 8086

A<sub>0</sub> and BHE\* is not used for interfacing of EPROM. Both the 8 KB EPROM chips are selected whenever any address in the range FC000H – FFFFFH comes on the address bus.

#### Static RAM Interfacing with 8086

The general procedure of static memory interfacing with 8086 is briefly described as follows.

- Arrange the available memory chips so as to obtain 16 bits data bus with the upper 8 bit bank is called "Odd address memory bank" and the lower 8 bit bank is called "Even address memory bank".
- Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory RD\* and WR\* inputs to the corresponding processor control signals. Connect the 16 bit data bus of the memory bank was that of microprocessor 8086.
- The remaining address lines of the microprocessor, BHE\* and A<sub>0</sub> are used for loading the required chip select signals for the odd and even memory banks. The CS\* of the memory is derived from the O/P of the decoding.

Let us take an example to address 16 KB of RAM to 8086 microprocessor. We will split 16 KB into two blocks each of 8 KB one of even addressed and another of odd addressed. Depending upon the bit on A<sub>0</sub>, either the even or odd bank will be selected. If A<sub>0</sub> = 0, the even bank is selected. Now the BHE\* signal will be 0 whenever a byte or a word is being accessed at odd address. Also for an even addressed word, both the banks will have to be enabled at the same time. Hence, A<sub>0</sub> has to be given to CS\* of even bank and BHE\* has to be given to CS\* of odd bank.

- If only A<sub>0</sub> is low, memory location from even bank is accessed.
- If only BHE\* is low, memory location from odd bank is accessed.
- If both are low, 2 memory locations are accessed from each bank.

To address 16 KB, we require 14 address lines. Of the 16 KB, 8 KB will be at even addresses and 8 KB will be at odd addresses. Hence, we use 2 RAM chips, each of 8 KB capacity one for storing bytes at even address and another for storing bytes for odd address. We will start the RAM address from 80000H.

|        | A <sub>19</sub> A <sub>18</sub> A <sub>17</sub> A <sub>16</sub> A <sub>15</sub> A <sub>14</sub> | A <sub>13</sub> A <sub>12</sub> A <sub>11</sub> A <sub>10</sub> A <sub>9</sub> A <sub>8</sub> A <sub>7</sub> A <sub>6</sub> A <sub>5</sub> A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub> |
|--------|---|---|
| Start: | 1 0 0 0 0 0 0   | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   |
| End:   | 1 0 0 0 0 0 0   | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   |
|        |   | Block A   |
| Start: | 1 0 0 0 0 0 1   | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   |
| End:   | 1 0 0 0 0 0 1   | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   |
|        |   | Block B   |

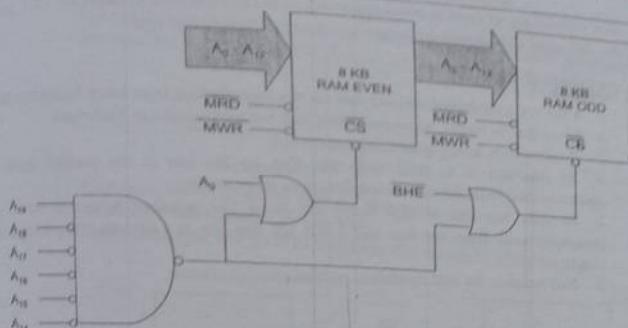


Fig: RAM Interfacing with 8086

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible i.e. these should be no windows and no feedback space should be allowed. A memory location should have a single address corresponding to it i.e. absolute decoding should be preferred.

Q) Interface two 4K X 8 EPROMs and two 4K X 8 RAM chips with 8086, select suitable maps.

We know that, after reset, the IP and CS are initiated to from address FFFF0H. Hence this address must lie in the EPROM. The address of RAM may be selected anywhere in the I/O space of 8086, but we will select the RAM address such that the address map of the system is continuous as shown in table below.

| Address      | A <sub>19</sub> | A <sub>18</sub> | A <sub>17</sub> | A <sub>16</sub> | A <sub>15</sub> | A <sub>14</sub> | A <sub>13</sub> | A <sub>12</sub> | A <sub>11</sub> | A <sub>10</sub> | A <sub>9</sub> | A <sub>8</sub> | A <sub>7</sub> | A <sub>6</sub> | A <sub>5</sub> | A <sub>4</sub> | A <sub>3</sub> | A <sub>2</sub> | A <sub>1</sub> | A <sub>0</sub> |   |
|--------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|---|
| FFFFFH       | 1               | 1               | 1               | 1               | 1               | 1               | 1               | 1               | 1               | 1               | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1 |
| EPROM 8K X 8 |                 |                 |                 |                 |                 |                 |                 |                 |                 |                 |                |                |                |                |                |                |                |                |                |                |   |
| FE000H       | 1               | 1               | 1               | 1               | 1               | 1               | 1               | 1               | 0               | 0               | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0 |
| RAM 8K X 8   |                 |                 |                 |                 |                 |                 |                 |                 |                 |                 |                |                |                |                |                |                |                |                |                |                |   |
| FDFFFH       | 1               | 1               | 1               | 1               | 1               | 1               | 0               | 1               | 1               | 1               | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1              | 1 |
| FC000H       | 1               | 1               | 1               | 1               | 1               | 1               | 1               | 0               | 0               | 0               | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0              | 0 |

Total 8K of EPROM need 13 address lines A<sub>0</sub>-A<sub>12</sub> (Since  $2^{13} = 8K$ ). Address lines A<sub>13</sub>-A<sub>19</sub> are used for decoding to generate the chip select. The BHE' signal goes low when a transfer is at odd address. If A<sub>0</sub> is 0 i.e. the address is even and is in RAM, then the lower RAM chip is selected.

The two 4K X 8 chips of RAM and EPROM are arranged in parallel to obtain 16-bit data bus width. If A<sub>0</sub> is 1 i.e. the address is odd and is in RAM, then the upper RAM chip is selected.

indicating 8-bit transfer at even address. If A<sub>0</sub> is 1 i.e. the address is odd and is in RAM, the BHE' goes low, the upper RAM chip is selected further indicating that the 8 bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A<sub>0</sub> and BHE' both are zero, both the RAM or ROM chips are selected i.e. the data transfer is of 16 bits. The selection of chips takes place as shown in table below.

| Decoder I/P                                      | A <sub>2</sub>  | A <sub>1</sub> | A <sub>0</sub> | Selection / Command         |
|--|-----------------|----------------|----------------|-----------------------------|
| Address / BHE'                                   | A <sub>12</sub> | A <sub>9</sub> | BHE'           | Even and Odd address in RAM |
| Word transfer on D <sub>8</sub> -D <sub>15</sub> | 0               | 0              | 0              | Only even address in RAM    |
| Byte transfer on D <sub>8</sub> -D <sub>7</sub>  | 0               | 0              | 1              | Only odd address in RAM     |
| Byte transfer on D <sub>8</sub> -D <sub>7</sub>  | 0               | 1              | 0              | Only odd address in RAM     |
| Word transfer on D <sub>8</sub> -D <sub>15</sub> | 1               | 0              | 0              | Even and Odd address in ROM |
| Byte transfer on D <sub>8</sub> -D <sub>7</sub>  | 1               | 0              | 1              | Only even address in ROM    |
| Byte transfer on D <sub>8</sub> -D <sub>7</sub>  | 1               | 1              | 0              | Only odd address in ROM     |

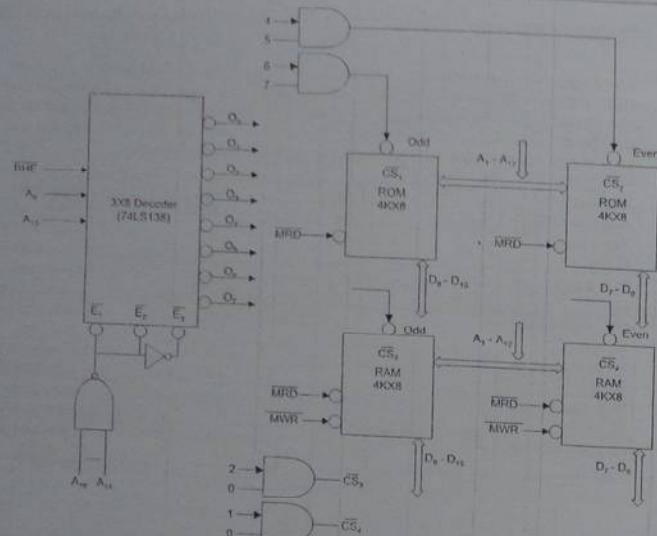


Fig: 8086 Interfacing Circuit

P.62

**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 appear 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

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar &amp; D. V. Hall |



- 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.

**1. Serial Interface**

- 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.

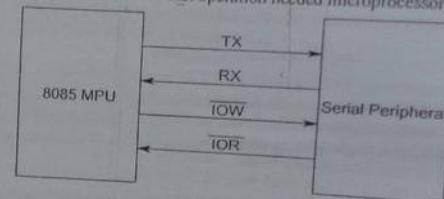


Fig: Serial Interface between microprocessor and I/O device.

**2. 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.

**Synchronizing the computer with peripherals:**

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:

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar &amp; D. V. Hall |



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.

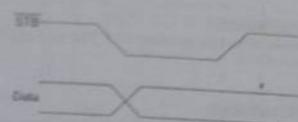


Fig: Simple Strobe I/O

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

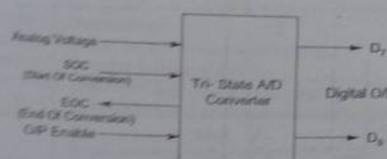


Fig: Tri-State A/D Converter

- 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) Single Handshaking:



Fig: Single Handshaking

- The peripheral outputs some data and send STB signal to MP. "here is the data for you."
- MP detects asserted 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) Double Handshaking

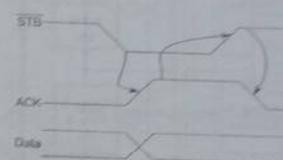


Fig: Double Handshaking

- The peripheral asserts its 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 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."

Programmable Peripheral Interface (PPI) - 8255A:

- The INTEL 8255 is a device used to parallel data transfer between processor and slave 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 number 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 handshaking signals and the remaining pins can be used as signal output lines or individually set/reset for control application.

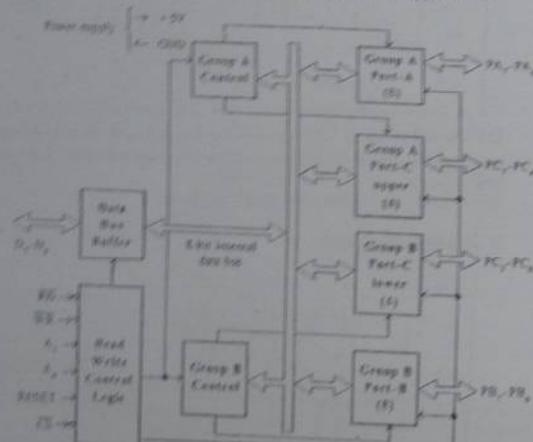


Fig: Internal Block Diagram of 8255

#### 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.

- 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.
- 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.

- RESET: This is an active high signal. It clears the control register and set all pins in the input mode.
- CS (low), A0 and A1: These are device select signals. They are,

| Device           | A0 | A1 |
|------------------|----|----|
| Port-A           | 1  | 0  |
| Port-B           | 0  | 1  |
| Port-C           | 1  | 0  |
| Control Register | 0  | 1  |

#### Serial Interface /Serial data transmission:

Serial IO transfer is more common than the parallel IO. 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.

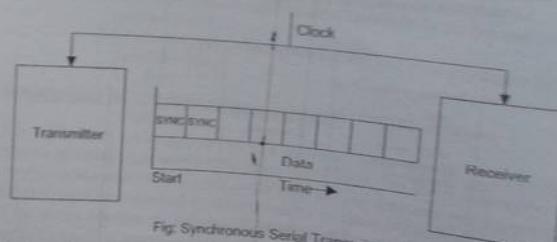


Fig: Synchronous Serial Transmission Format

##### 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

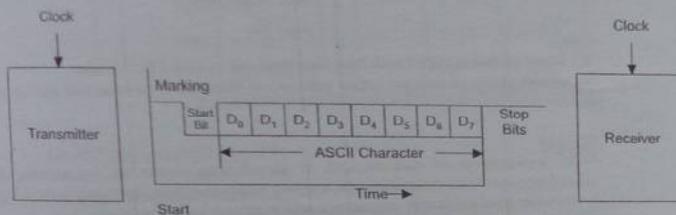


Fig: Asynchronous Serial Transmission Format

### Synchronous vs. Asynchronous Data Transmission

#### Asynchronous Communication

- Simple interface (limited data rate, typically < 64 kbps)
- Used for connecting: Printer, Terminal, Modem, home connections to the Internet
- No clock sent (Tx & Rx have own clocks)
- Requires start and stop bits which provides byte timing and increases overhead
- Parity often used to validate correct reception.
- Independent transmit & receive clocks

#### Synchronous Communication

- Clock sent with data (more configuration options).
- Synchronised transmit & receive clocks
- More complex interface (high data rates supported up to ~ 10 Gbps)
- Used for: Connections between computer and telephony networks.

### Universal Synchronous Asynchronous Receiver Transmitter (USART) – 8251A

- The 8251A is a programmable serial communication interface chip designed for synchronous and asynchronous serial data communication.
- It supports the serial transmission of data.

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

- Read/Write control logic
- Transmitter

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar & D. V. Hall | 41

- Receiver
- Data bus buffer
- Modem control

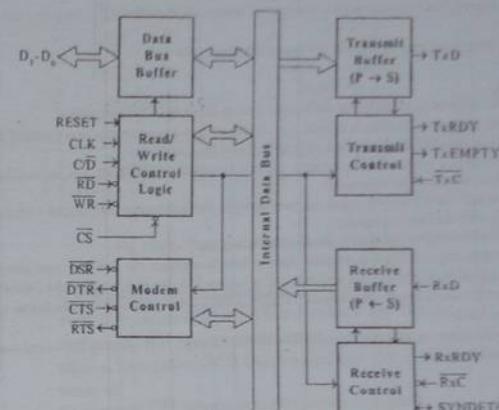


Fig: Functional block diagram of 8251A-USART

#### 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.

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar & D. V. Hall | 41

- 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 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 convert 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.

#### Band 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 band Rate is the number of times per second a signal in a communications channel changes. Bit rates measure the number of data bits (that is 0's 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:

$$\text{If signal is changing every } 1/3 \text{ ns then,} \\ \text{Baud rate} = 1/10/3\text{ns} = 3/10^9 = 3 \times 10^8 \\ = 300 \text{ mbd}$$

#### Note:

If 1 frame of data is coded with 1 bit then baud rate and bit rate are same.

Sometimes frame of data are coded with two or three bits then baud rate and bit rate are not same.

#### RS -232

Serial transmission of data is used as an efficient means for transmitting digital information across long distances, the existing communication lines usually the telephone lines can be used to transfer information which saves a lot of hardware.

RS-232C is an interface developed to standardize the interface between data terminal equipment (DTE) and data communication equipment (DCE) employing serial binary data exchange. Modem and other devices used to send serial data are called data communication equipment (DCE). The computers or terminals that are sending or receiving the data are called data terminal.

Equipment (DTE) RS- 232C is the interface standard developed by electronic industries Association (EIA) in response to the need for the signal and handshake standards between the DTE and DCE.

It uses 25 pins (DB – 25P) or 9 Pins (DE – 9P) standard where 9 pin standard does not use all signals i. e. data, control, timing and ground.

It describes the voltage levels, impedance levels, rise and fall times, maximum bit rate and maximum capacitance for all signal lines.

It also specifies that DTE connector should be male and DCE connector should be female.

It can send 20kBd for a distance of 50 ft.

The voltage level for RS-232 are:

- o A logic high or 1, -3V to -15V
- o A logic low or 0, +3V to +15V  
Normally ±12V voltage levels are used.

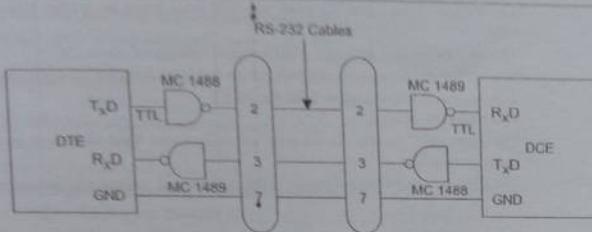


Fig: Connection of DTE and DCE through RS-232C Interface

- MC1488 converts logic 1 to -9V
- Logic 0 to +9V
- MC1485 converts RS - 232 to TTL
- Signal levels of RS-232 are not compatible with that of the DTE and DCE which are TTL signals for that line driver such as MC1488 and line receiver MC1485 are used.
- RS- 232 signals used in handshaking

| Flow       | DB-9P | DB-25P | Signal | Description         |
|------------|-------|--------|--------|---------------------|
| DTE to DCE | 3     | 2      | TxD    | Transmitted data    |
| DCE to DTE | 2     | 3      | RxD    | Received data       |
| DTE to DCE | 7     | 4      | RTS    | Request to send     |
| DCE to DTE | 8     | 5      | CTS    | Clear to send       |
| DCE to DTE | 6     | 6      | DSR    | Data set ready      |
| Common ref | 5     | 7      | GND    | Signal ground       |
| DCE to DTE | 1     | 8      | DCD    | Data carrier detect |
| DTE to DCE | 4     | 20     | DTR    | Data terminal ready |

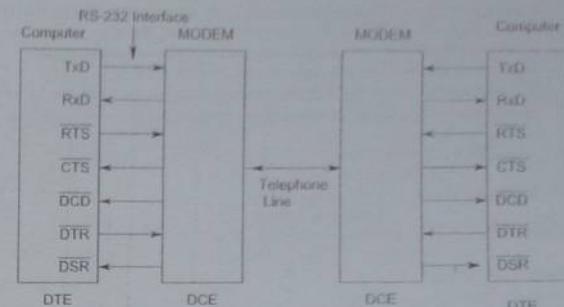


Fig: Digital Data transmission using MODEM and Telephone Line

- DTE asserts  $DTR$  to tell the modem it is ready
- Then DCE asserts  $DSR$  signal to the terminal and dials up.
- DTE asserts  $RTS$  signal to the modem.
- Modem then asserts  $CD$  signal to indicate that it has established connection with the computer.
- DCE asserts  $CTS$  signals, then DTE sends serial data.
- When sending completed, DTE asserts  $RTS$  high, this causes modem to unassert its  $CTS$  signal and stop transmitting similar handshake taken between DCE and DTE other side.
- To communicate from serial port of a computer to serial port of another computer without modem, null-modem is used.

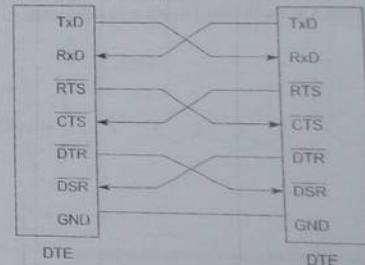


Fig: Null MODEM Connection for RS-232 Terminals

**RS 423A and RS422A serial standards**
**RS 423A**

- A major problem with RS 232C is that it can only transmit data reliably for about 50 ft at its maximum rate of 20Kbd. If longer lines are used the transmission rate has to be drastically reduced due to open signal lines with a common signal ground.
- Another EIA standard which is improvement over RS-232C is RS-423A.
- This standard specifies a low impedance single ended signal which can be sent over 50Ω coaxial cable and partially terminated at the receiving end to prevent reflection.
- Voltage levels
  - o High -4.6V negative
  - o Low +4.6V positive
- Transmission rate
  - 100 Kbd over 40 ft
  - 1 Kbd over 4000 ft

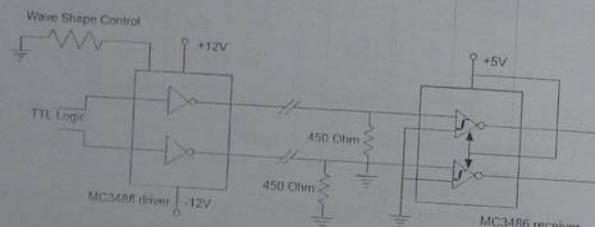


Fig. MC3488 driver and MC3486 receiver used for RS-423A Interface

**RS 422A**

- A newer standard for serial data transfer.
- It specifies that can signal will be send differentially over two adjacent wires in a ribbon cable or a twisted pair of wires uses differential amplifier to reject noise.

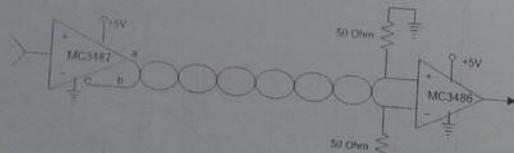


Fig. MC3487 driver and MC3486 receiver used for RS-422A Interface

Voltage level logic high making a line more positive

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar &amp; D. V. Hall | 37

- Logic low making a line lines must.
- The voltage difference between the two lines must be greater than 0.4V but less than 12V.
- The mc3487 driver provides a differential voltage of about 2V.
- The center or common mode voltage on the lines must be between -7V and +7V.
- Transmission rate is 10000 Kbd for 40 FT
- 100 KBD for 4000 ft.
- The high data transfer is because of differential line functions as a fully terminated transmission line.
- Mc 3486 receiver only responds to the differential voltage eliminating noise.

**Introduction to Direct Memory Access (DMA) & DMA Controllers**

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.

This is a process where data is transferred between two peripherals directly without the involvement of the microprocessor. This process employs the HOLD pin on the microprocessor. The external DMA controller sends a signal on the HOLD pin to the microprocessor. The microprocessor completes the current operation and sends a signal on HLDA and stops using the buses. Once the DMA controller is done, it turns off the HOLD signal and the microprocessor takes back control of the buses.

**Basic DMA operation**

- The direct memory access (DMA) technique provides direct access to the memory while the microprocessor is temporarily disabled.
- A DMA controller temporarily borrows the address bus, data bus, and control bus from the microprocessor and transfers the data bytes directly between an I/O port and a series of memory locations.
- The DMA transfer is also used to do high-speed memory-to memory transfers.
- Two control signals are used to request and acknowledge a DMA transfer in the microprocessor-based system.
- The HOLD signal is a bus request signal which asks the microprocessor to release control of the buses after the current bus cycle.

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. S. Gaonkar &amp; D. V. Hall | 38

- The HLDA signal is a bus grant signal which indicates that the microprocessor has indeed released control of its buses by placing the buses at their high-impedance states.
- The HOLD input has a higher priority than the INTR or NMI interrupt inputs.

#### DMA Data Transfer scheme

- Data transfer from I/O device to memory or vice-versa is controlled by a DMA controller.
- This scheme is employed when large amount of data is to be transferred.
- The DMA requests the control of buses through the HOLD signal and the MPU acknowledges the request through HLDA signal and releases the control of buses to DMA.
- It's a faster scheme and hence used for high speed printers.

#### Block (Burst) mode of data transfer

In this scheme the I/O device withdraws the DMA request only after all the data bytes have been transferred.

#### Cycle stealing technique

In this scheme the bytes are divided into several parts and after transferring every part the control of buses is given back to MPU and later stolen back when MPU does not need it.

#### PROGRAMMABLE DMA CONTROLLER - INTEL 8257

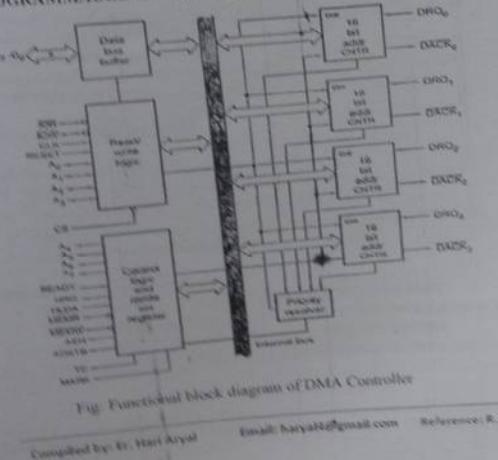


Fig: Functional block diagram of DMA Controller

Compiled by: Er. Hari Aryal

Email: hariyah@gmail.com

Reference: R. S. Gaonkar & D. V. Hall |

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 8257 is,

- The 8257 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.

The functional blocks of 8257 as shown in the above figure are data bus buffer, read/write logic, control logic, priority resolver and four numbers of DMA channels.

#### Operation of 8257 DMA Controller

- Each channel of 8257 has two programmable 16-bit registers named as address register and count register.
- Address register is used to store the starting address of memory location for DMA data transfer.
- The address in the address register is automatically incremented after every read/write/verify transfer.
- The count register is used to count the number of byte or word transferred by DMA.
- In read transfer the data is transferred from memory to I/O device.
- In write transfer the data is transferred from I/O device to memory.
- Verification operations generate the DMA addresses without generating the DMA memory and I/O control signals.
- The 8257 has two eight bit registers called mode set register and status register.

## Chapter - 5 Interrupt Operations

- 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 provide or 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.

### 5.1 Polling versus Interrupt

- Each time the device is given a command, for example "move the read head to sector 12 of the floppy disk" the device driver has a choice as to how it finds out that the command has completed. The device drivers can either poll the device or they can use interrupts.
- Polling the device usually means reading its status register every so often until the device's status changes to indicate that it has completed the request.
- Polling means the CPU keeps checking a flag to indicate if something happens.
- An interrupt driven device driver is one where the hardware device being controlled will cause a hardware interrupt to occur whenever it needs to be serviced.
- With interrupt, CPU is free to do other things, and when something happens, an interrupt is generated to notify the CPU. So it means the CPU does not need to check the flag.
- Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring.
- Interrupts win if processor has other work to do and event response time is not critical.
- Polling can be better if processor has to respond to an event ASAP, may be used in device controller that contains dedicated secondary processor.

### Advantages of interrupt over Polling

- Interrupts are used when you need the fastest response to an event. For example, you need to generate a series of pulses using a timer. The timer generates an interrupt when it overflows and within 1 or 2 sec, the interrupt service routine is called to generate the pulse. If polling were used, the delay would depend on how often the polling is done and could delay response to several msec. This is thousands times slower.
- Interrupts are used to save power consumption. In many battery powered applications, the microcontroller is put to sleep by stopping all the clocks and reducing power consumption to a few micro amps. Interrupts will awaken the controller from sleep to consume power only when needed. Applications of this are hand held devices such as TV/VCR remote controllers.
- Interrupts can be a far more efficient way to code. Interrupts are used for program debugging.

### 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.

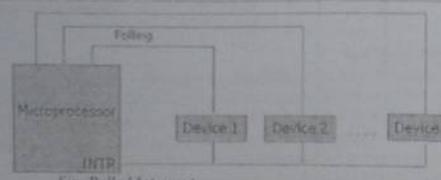


Fig. Polled Interrupt

Here several external 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 manufacturer 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

In polled interrupt, the time required to poll each device may exceed the time to service the device through software. To improve this, the faster mechanism called vectored or daisy chain interrupt is used. Here the devices are connected in chain fashion. When INTR pin goes up, the processor saves its current status and then generates INTA signal to the highest priority device. If this device has generated the interrupt, it will accept the INTA, otherwise it will push INTA to the next priority device until the INTA is accepted by the interrupting device.

When INTA is accepted, the device provides a means to the processor for finding the interrupt address vector using external hardware. The accepted device responds by placing a word on the data lines which becomes the vector address with the help of any hardware through which the processor points to appropriate device service routine. Here no general interrupt service routine need first that means appropriate ISR of the device will be called.

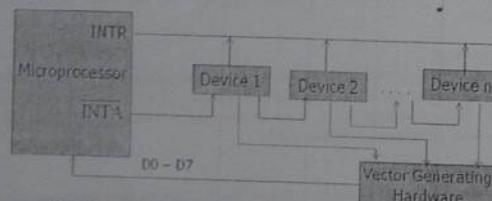


Fig. Vectored (Daisy Chain) Interrupt

## 5.2 Interrupt Processing Sequence

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 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.

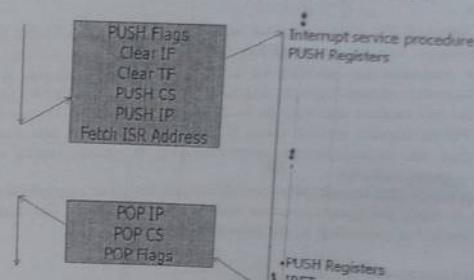


Fig. Interrupt Response for 8086 Microprocessor

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.



- 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 <sub>H</sub> |
| RST 1     | 0008 <sub>H</sub> |
| RST 2     | 0010 <sub>H</sub> |
| RST 3     | 0018 <sub>H</sub> |
| RST 4     | 0020 <sub>H</sub> |
| RST 5     | 0028 <sub>H</sub> |
| RST 6     | 0030 <sub>H</sub> |
| RST 7     | 0038 <sub>H</sub> |

**5.4.1 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 | 1 <sup>st</sup> | No       | 0024H          |
| RST 7.5        | Edge           | 2 <sup>nd</sup> | Yes      | 003CH          |
| RST 6.5        | Level          | 3 <sup>rd</sup> | Yes      | 0034H          |
| RST 5.5        | Level          | 4 <sup>th</sup> | Yes      | 002CH          |
| INTR           | Level          | 5 <sup>th</sup> | Yes      | -              |

**TRAP:**

- This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
- TRAP has the highest priority and vectored interrupt.
- TRAP interrupt is edge and level triggered. This means that 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. i.e. Input goes to high and no need to maintain high state until it is 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. i.e. Input goes to high and stay high until it is 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,
  - DI, SIM instruction
  - System or processor reset
  - After reorganization of interrupt
- Enabled by EI instruction.
- Non-vectorized interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.
- It has lowest priority.
- It is a level sensitive interrupt. i.e. Input goes to high and it is necessary to maintain high state until it is recognized.
- The following sequence of events occurs when INTR signal goes high.
  - The 8085 checks the status of INTR signal during execution of each instruction.
  - If INTR signal is high, then 8085 completes its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.
  - In response to the acknowledge signal, external logic places an instruction OPCODE on the data bus. In the case of multibyte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.
  - On receiving the instruction, the 8085 saves the address of next instruction on stack and executes received instruction.

#### 5.4.2 Using Programmable Interrupt Controller (PIC)

**Priority interrupt controller (PIC)**

The INTR pin can be used for multiple peripherals and to determine priorities among these devices when two or more peripherals request interrupt service simultaneously, PIC is used. If there are simultaneous requests, the priorities are determined by the encoder, it responds to the higher level input, ignoring the lower level input. The drawback of the scheme is that the interrupting device connected to input  $I_2$  always has the highest priority. The PIC includes a status register and a priority comparator in addition to a priority encoder.

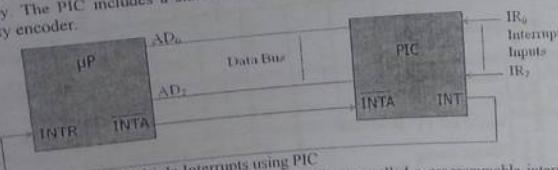


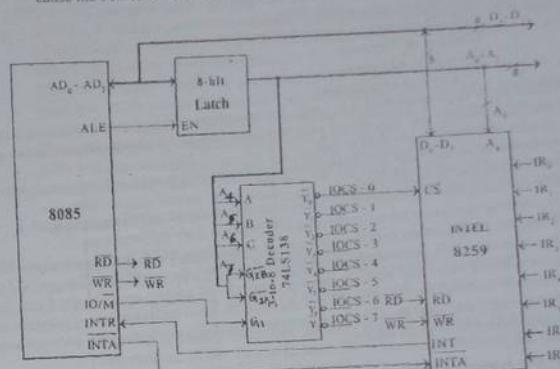
Fig: Multiple Interrupts using PIC  
Today this device is replaced by a more versatile one called a programmable interrupt controller 8259A. When an 8259A receives an interrupt signal on one of its IR inputs, it

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. Gaonkar & D.V. Hall | 9

sends an interrupt request signal to the INTR input of the μP. Then INTA pulses will cause the PIC to release vectoring information onto the data bus.



- It requires two internal address and they are  $A=0$  or  $A=1$ .
- It can be either memory mapped or I/O mapped in the system. The interfacing of 8259 to 8085 is shown in figure is I/O mapped in the system.
- The low order data bus lines D0-D7 are connected to D0-D7 of 8259.
- The address line A0 of the 8085 processor is connected to A0 of 8259 to provide the internal address.
- The 8259 require one chip select signal. Using 3-to-8 decoder generates the chip select signal for 8259.
- The address lines A4, A5 and A6 are used as input to decoder.
- The control signal IO/M (low) is used as logic high enables for decoder and the address line A7 is used as logic low enable for decoder.
- The I/O addresses of 8259 are shown in table below.

|                                       | Binary Address       |                |                              |                |                |                |                |                | Hexa address |
|---------------------------------------|----------------------|----------------|------------------------------|----------------|----------------|----------------|----------------|----------------|--------------|
|                                       | Decoder input/enable |                | Input to address pin of 8259 |                |                |                |                |                |              |
|                                       | A <sub>1</sub>       | A <sub>2</sub> | A <sub>3</sub>               | A <sub>4</sub> | A <sub>5</sub> | A <sub>6</sub> | A <sub>7</sub> | A <sub>8</sub> |              |
| For A <sub>0</sub> of 8259 to be zero | 0                    | 0              | 0                            | 0              | x              | x              | x              | 0              | 00           |
| For A <sub>0</sub> of 8259 to be one  | 0                    | 0              | 0                            | 0              | x              | x              | x              | 1              | 01           |

Note : Don't care 'x' is considered as zero.

#### Working of 8259 with 8085 processor:

- First the 8259 should be programmed by sending Initialization Command Word (ICW) and Operational Command Word (OCW). These command words will inform 8259 about the following.
  - Type of interrupt signal (Level triggered / Edge triggered).
  - Type of processor (8085/8086)
  - Call address and its interval (4 or 8)
  - Masking of interrupts.
  - Priority of interrupts.
  - Type of end of interrupts.
- Once 8259 is programmed it is ready for accepting interrupt signal. When it receives an interrupt through any one of the interrupt lines IR0-IR7 it checks for its priority and also checks whether it is masked or not.
- If the previous interrupt is completed and if the current request has highest priority and unmasked, then it is serviced.
- For servicing this interrupt the 8259 will send INT signal to INTR pin of 8085.
- In response if expects an acknowledge INTA (low) from the processor.
- In response to first, second and third INTA (low) signals, the 8259 will supply CALL opcode, low byte of call address and high byte of call address respectively. Once the processor receives the call opcode and its address, it saves the content of program counter (PC) in stack, and load the CALL address in PC and start executing the interrupt service routine stored in this call address.

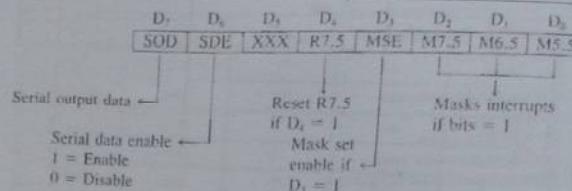
#### How INTR pin is used in 8085:

The microprocessor checks INTR one clock period before the last T-state of an instruction cycle. In the 8085, the call instructions require 18 T-states, therefore the INTR pulse should be high at least for 17.5 T-states. The INTR can remain high until the interrupt flip flop is set by the EI instruction in the service routine. If it remains high after the execution of the EI instruction, the processor will be interrupted again, as if it was a new interrupt.

#### 5.4.3 Interrupt instructions

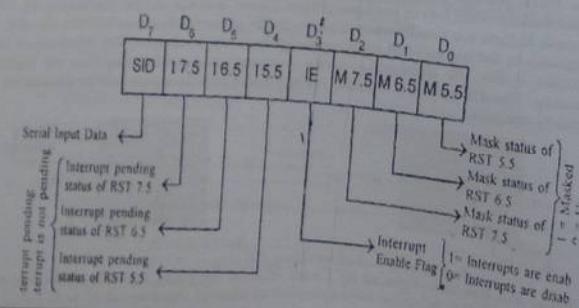
##### SIM instruction:

- The 8085 provide additional masking facility for RST 7.5, RST 6.5 and RST 5.5 using SIM instruction.
- This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output.
- The masking or unmasking of RST 7.5, RST 6.5 and RST 5.5 interrupts can be performed by moving an 8-bit data to accumulator and then executing SIM instruction.
- The format of the 8-bit data is shown below.



- SOD — Serial Output Data: Bit D<sub>7</sub> of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D<sub>6</sub> = 1.
- SDE — Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- XXX — Don't Care
- R7.5 — Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- MSE — Mask Set Enable: If this bit is high, it enables the functions of bits D<sub>2</sub>, D<sub>1</sub>, D<sub>0</sub>. This is a master control over all the interrupt masking bits. If this bit is low, bits D<sub>2</sub>, D<sub>1</sub>, and D<sub>0</sub> do not have any effect on the masks.
- M7.5 — D<sub>2</sub> = 0, RST 7.5 is enabled.  
= 1, RST 7.5 is masked or disabled.
- M6.5 — D<sub>1</sub> = 0, RST 6.5 is enabled.  
= 1, RST 6.5 is masked or disabled.
- M5.5 — D<sub>0</sub> = 0, RST 5.5 is enabled.  
= 1, RST 5.5 is masked or disabled.

##### RIM instruction



- The status of pending interrupts can be read from accumulator after executing RIM instruction.
- This is a multipurpose instruction used to read the status of RST 7.5, 6.5, 5.5 and read serial data input bit.
- When RIM instruction is executed an 8-bit data is loaded in accumulator, which can be interpreted as shown in above fig.
- Bits 0-2 show the current setting of the mask for each of RST 7.5, RST 6.5 and RST 5.5. They return the contents of the three masks flip flops. They can be used by a program to read the mask settings in order to modify only the right mask.
- Bit 3 shows whether the maskable interrupt process is enabled or not. It returns the contents of the Interrupt Enable Flip Flop. It can be used by a program to determine whether or not interrupts are enabled.
- Bits 4-6 show whether or not there are pending interrupts on RST 7.5, RST 6.5, and RST 5.5. Bits 4 and 5 return the current value of the RST5.5 and RST6.5 pins. Bit 6 returns the current value of the RST7.5 memory flip flop.
- Bit 7 is used for Serial Data Input. The RIM instruction reads the value of the SID pin on the microprocessor and returns it in this bit.

**D1**

- Disable interrupts
- The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.
- 1 byte instruction
- Example: DI

**E1**

- Enable interrupts
- The interrupt enable flip-flop is set and all interrupts are enabled.
- No flags are affected.
- After a system reset or the acknowledgement of an interrupt, the interrupt enable flip flop is reset, thus disabling the interrupts.
- This instruction is necessary to enable the interrupts (except TRAP).
- 1 byte instruction
- Example: EI

### 5.5 Interrupt Processing in 8086

The meaning of ‘interrupts’ is to break the sequence of operation. While the CPU is executing a program, on ‘interrupt’ breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR). After executing ISR, the control is transferred back again to the main program. Interrupt processing is an alternative to polling.

#### 5.5.1 Interrupt Pins INTR and NMI

- INTR is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAG5 register with the help of the POPF instruction.
- When an interrupt occurs, the processor stores FLAG5 register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 \* <interrupt type>. Interrupt processing routine should return with the IRET instruction.
- NMI is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 00008H. This interrupt has higher priority than the maskable interrupt.
- Ex: NMI, INTR.

#### 5.5.2 Interrupt Vector Table and its Organization

- An interrupt vector is a pointer to where the ISR is stored in memory.
- All interrupts (vectored or otherwise) are mapped onto a memory area called the Interrupt Vector Table (IVT).
  - The IVT is usually located in memory page 00 (0000H - 00FFH)
  - The purpose of the IVT is to hold the vectors that redirect the microprocessor to the right place when an interrupt arrives.

Interrupt Vector Table (IVT) is a 1024 bytes sized table that contains addresses of interrupts. Each address is of 4 bytes long of the form offset:segment, which represents the address of a routine to be called when the CPU receives an interrupt. IVT can hold maximum of 256 addresses (0 to 255). The interrupt number is used as an index into the table to get the address of the interrupt service routine. IVT act as pointers, unlike function call IVT need number as an argument then as a result IVT point us to interrupt service routine (ISR). ISR executes its code, when ISR finished then returns back to original statement. Interrupt vector table is a global table situated at the address 0000:0000H. The interrupt vector table is a feature of the Intel 8086/8088 family of microprocessors.

|      |                        |
|------|------------------------|
|      | 32-255 User defined    |
|      | 14-31 Reserved         |
| 040H | Coprocessor error      |
| 030H | Unassigned             |
| 020H | Page fault             |
| 010H | General protection     |
| 031H | Stack seg overflow     |
| 02CH | Segment not present    |
| 02BH | Invalid task state seg |
| 02AH | Cause seg overflow     |
| 029H | Double fault           |
| 01CH | Coprocessor not avail  |
| 018H | Undefined Opcode       |
| 014H | Bound                  |
| 010H | Overflow, INTIO        |
| 00CH | 1-byte breakpoint      |
| 008H | NMI pin                |
| 02FH | Single-step            |
| 000H | Divide error           |

The interrupt vector table is located in the first 1024 bytes of memory at addresses 00000011 through 0003FFH.

There are 256 4-byte entries (segment and offset in real mode).

| Seg high | Seg low | Offset high | Offset low |
|----------|---------|-------------|------------|
| Byte 3   | Byte 2  | Byte 1      | Byte 0     |

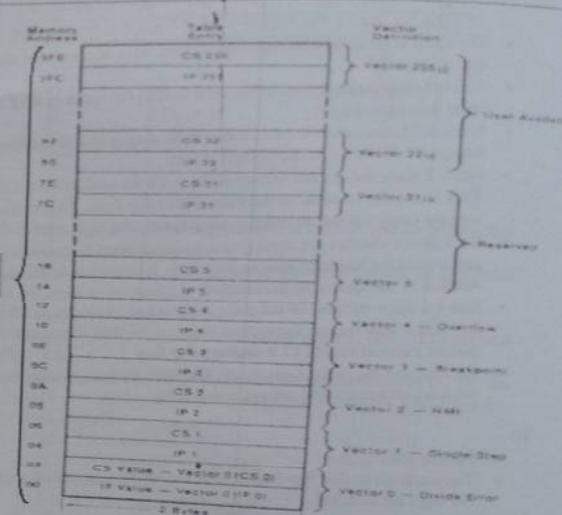


Fig: IVT Structure (Organization)

Functions associated with INT00 to INT04

| INT Number | Physical Address |
|------------|------------------|
| INT 00     | 00000            |
| INT 01     | 00004            |
| INT 02     | 00008            |
| ...        | ...              |
| INT FF     | 003FC            |

#### INT 00 (divide error)

- INT00 is invoked by the microprocessor whenever there is an attempt to divide a number by zero.
- ISR is responsible for displaying the message "Divide Error" on the screen.

#### INT 01

- For single stepping the trap flag must be 1

- After execution of each instruction, 8086 automatically jumps to 00004H to fetch 4 bytes for CS:IP of the ISR.
- The job of ISR is to dump the registers on to the screen

#### INT 02 (Non maskable Interrupt)

- When ever NMI pin of the 8086 is activated by a high signal (5V), the CPU Jumps to physical memory location 00008 to fetch CS:IP of the ISR associated with NMI.

#### INT 03 (break point)

- A break point is used to examine the CPU and memory after the execution of a group of Instructions.
- It is one byte instruction whereas other instructions of the form "INT nn" are 2 byte instructions.

#### INT 04 (Signed number overflow)

- There is an instruction associated with this INT 0 (interrupt on overflow).
- If INT 0 is placed after a signed number arithmetic as IMUL or ADD the CPU will activate INT 04 if OF = 1.
- In case where OF = 0, the INT 0 is not executed but is bypassed and acts as a NOP.

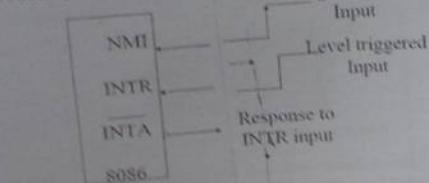
#### 5.5.3 Software and Hardware Interrupt

**Types of Interrupts:** There are two types of Interrupts in 8086. They are:

- Hardware Interrupts** (External Interrupts). The Intel microprocessors support hardware interrupts through:
  - Two pins that allow interrupt requests, INTR and NMI.
  - One pin that acknowledges, INTA, the interrupt requested on INTR.

#### Performance of Hardware Interrupts

- NMI - Non maskable interrupts - TYPE 2 Interrupt
- INTR - Interrupt request - Between 20H and FFH



- Software Interrupts** (Internal Interrupts and Instructions). Software interrupts can be caused by:
  - INT instruction - breakpoint interrupt. This is a type 3 interrupt.

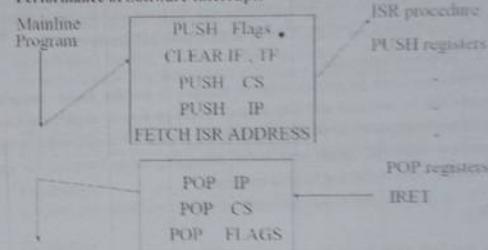
Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. Gaonkar & D.V. Hall | 37

- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- Processor exceptions: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7)
- Software interrupt processing is the same as for the hardware interrupts.
- Ex: INT n (Software Instructions)
- Control is provided through
  - IF and TF flag bits
  - IRET and IRET

#### Performance of Software Interrupts



- It decrements SP by 2 and pushes the flag register on the stack.
- Disables INTR by clearing the IF.
- It resets the TF in the flag Register.
- It decrements SP by 2 and pushes CS on the stack.
- It decrements SP by 2 and pushes IP on the stack.
- Fetch the ISR address from the interrupt vector table.

#### 5.5.4 Interrupt Priorities

| Interrupt                  | Priority |
|----------------------------|----------|
| Divide Error, INT(n), INTO | Highest  |
| NMI                        |          |
| INTR                       |          |
| Single Step                | Lowest   |

Compiled by: Er. Hari Aryal

Email: haryal4@gmail.com

Reference: R. Gaonkar & D.V. Hall | 37

## 8086 interrupts Summary

An interrupt is a signal indicating that an event needing immediate attention has occurred there are two types of interrupts; external interrupt generated outside CPU by other hardware and internal interrupt generated within CPU as a result of an instruction or operation.

In 8086, two types of interrupts occurred.

### 1. Hardware interrupt

These are external interrupts which uses NMI and INTR. When hardware interrupt is detected, the interrupt controller sends the interrupt code to the processor. When the code is finally acquired by the processor either from TNT opcode or from the interrupt controller. This is used by the processor to index the interrupt vector table to find the address of the interrupt handler. The 8086 specifies 256 different interrupts specified by type number or vector which is a pointer into IVT. The pointer is cs:ip values .

### 2. Software interrupts

Software Interrupts are used to publish internal services to outside world. These are internal interrupts like int and into and trap such as divide by zero or single step. Other software interrupts also included by 8086 processor. INT is used for breakpoint and INTO is used for overflow interrupt. Single step is the debugging mode interrupt for each instruction and divide error is dividing by zero interrupt.

### Interrupt vector table:

It is located in the first 1024 bytes of memory at address 000000-0003FFH. It contains 256 different 4-byte interrupt vectors. An interrupt vector contains the segment & offset address of the interrupt service provider.

### DOS & BIOS interrupts:

Dos interrupts services link applications with os services such as opening file, reading, writing content using certain functions of INT 4 H. BIOS interrupts control the screen disk controller and keyboard operation using INT 10H, 13H,16H etc.

## Chapter – 6

### Advanced Topics

#### 6.1 Multiprocessing Systems

Traditionally, the computer has been viewed as a sequential machine. Most computer programming languages require the programmer to specify algorithms as sequence of instructions. Processor executes programs by executing machine instructions in a sequence and one at a time. This view of the computer has never been entirely true. At the micro operation level, multiple control signals are generated at the same time. Instruction pipelining and the overlapping of fetch & execute instructions from the same program in parallel.

As computer technology has evolved and as the cost of computer hardware has dropped, computer designers have sought more and more opportunities for parallelism, usually to enhance performance and availability. Multiprocessing is an example of parallelism which uses multiple CPUs sharing the common resources such as memory, storage device etc.

#### Characteristics of Multiprocessing system

- Contains two or more similar general purpose processors of comparable capability.
- All processors share access to common memory.
- All processors share access to I/O devices either through the same channels that provide paths to the same devices.
- System is controlled by an integrated operating system that provides interaction between processors and their programs.

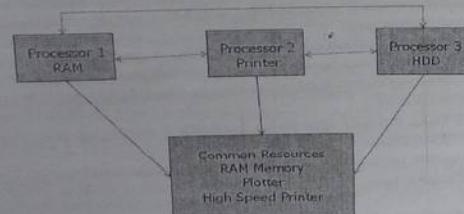


Fig: Organization of Multiprocessing System

Here the processors can communicate with each other through memory. The CPUs can directly exchange signals as indicated by dotted line. The organization of multiprocessor system can be divided into three types.

#### 1. Time shared or common bus:

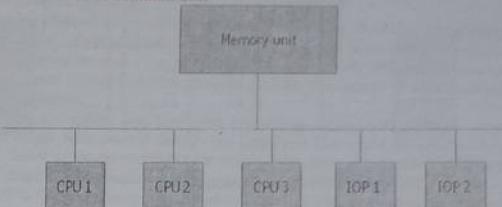


Fig: Time Shared System

There are number of CPUs, I/O modules and memory modules connected to the same bus. So the time shared system must distinguish the modules on the bus to determine the source and destination of the data. Any module in the bus can temporarily act as a master. When one module is controlling the bus, the other should be locked out. The access to each module is divided on the basis of time. The time shared multiprocessing system has the following advantages.

#### Simplicity:

The physical interface and the addressing time sharing logic of each processor remains the same as in a single processor system, so it is very simplest approach.

#### Flexibility:

It is easy to expand the system by attaching more CPUs to the bus.

#### Reliability:

The failure of any attached device should not the failure of the whole system.

#### Drawback:

The speed of the system is limited by the cycle time because all memory references must pass through the common bus.

#### 2. Multiport memory:

Each processor and I/O module has dedicated path to each memory module this system has more performance and complexity than earlier one. For this system, it is possible to configure portions of memory as private to one or more CPUs and/or I/O modules. This feature allows increasing security against unauthorized access and the storage of recovery routines in areas of memory not susceptible to modification by other processors.

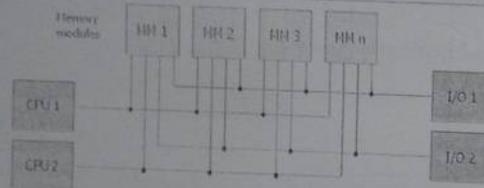


Fig: Multiport Memory System

### 3. Central control unit :

It manages the transfer of separate data streams back and forth between independent modules like CPU, memory and I/o. the controller can buffer requests and perform arbitration and timing functions. It can also pass status and control messages between CPUs. All the co-ordination is concentrated in the central control unit un-disturbing the modules. It is more flexible and complex as well.

#### 6.1.1 Real and Pseudo Parallelism

- Traditionally, software has been written for serial computation.
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions;
  - Instructions are executed one after another;
  - Only one instruction may execute at any moment in time.
- In the simplest sense, parallelism is the simultaneous use of multiple compute resources to solve a computational problem.
  - To be run using multiple CPUs;
  - A problem is broken into discrete parts that can be solved concurrently;
  - Each part is further broken down to a series of instructions;
  - Instructions from each part execute simultaneously on different CPUs.
- Real parallelism consists of the parallel modes of physical devices so that each can carry parallel operations to each other. Core parallelism consists of real parallelism. Multiple core processes which are physically different and performs their own operations in parallel.
- Pseudo parallelism consists of the same device carrying the parallel operation. We can logically manage the parallelism for system. Concurrent processing using parallelism is the pseudo parallelism which operates either in time division or using other types of parallel algorithms.

#### 6.1.2 Flynn's Classification

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's classification.
- Flynn's classification distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of **Instruction** and

Compiled by: Er. Hari Aryal Email: haryal4@gmail.com Reference: W. Stallings & A. S. Tanenbaum | 3

**Data** Each of these dimensions can have only one of two possible states: *Single* or *Multiple*.

- The matrix below defines the 4 possible classifications according to Flynn:

| S I S D<br>Single Instruction, Single Data   | S I M D<br>Single Instruction, Multiple Data   |
|--|--|
| M I S D<br>Multiple Instruction, Single Data | M I M D<br>Multiple Instruction, Multiple Data |

#### 1. Single Instruction, Single Data (SISD):

- A serial (non-parallel) computer
- Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- Single Data:** Only one data stream is being used as input during any one clock cycle
- Single instruction is performed on a single set of data in a sequential form
- Deterministic execution

Examples: older generation mainframes, minicomputers and workstations; most modern day PCs.

#### 2. Single Instruction, Multiple Data (SIMD):

- A type of parallel computer
- Single Instruction:** All processing units execute the same instruction at any given clock cycle
- Multiple Data:** Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing
- Single Instruction is performed on multiple data. A good example is the 'For' loop statement. Over here instruction is the same but the data stream is different. Examples:
  - Processor Arrays: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV
  - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10

Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

#### 3. Multiple Instruction, Single Data (MISD):

- A type of parallel computer
- Multiple Instruction:** Each processing unit operates on the data independently via separate instruction streams.
- Single Data:** A single data stream is fed into multiple processing units,
- N numbers of processors are working on different set of instruction on the same set of data.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971)

Compiled by: Er. Hari Aryal Email: haryal4@gmail.com Reference: W. Stallings & A. S. Tanenbaum | 4

- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream

Multiple cryptography algorithms attempting to crack a single coded message.

#### 4. Multiple Instruction, Multiple Data (MIMD):

- A type of parallel computer
- Multiple Instruction:** Every processor may be executing a different instruction stream
- Multiple Data:** Every processor may be working with a different data stream
- There is an interaction of N numbers of processors on a same data stream shared by all processors.
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.

Note: many MIMD architectures also include SIMD execution sub-components

#### 6.1.3 Instruction Level, Thread Level and Process Level Parallelism

##### Instruction Level Parallelism

Instruction-level parallelism (ILP) is a measure of how many of the operations in a computer program can be performed simultaneously. Consider the following program:

```

1. e = a + b
2. f = c + d
3. g = e * f
    :
```

Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in a total of two units of time, giving an ILP of 3/2.

A goal of compiler and processor designers is to identify and take advantage of as much ILP as possible. Ordinary programs are typically written under a sequential execution model where instructions execute one after the other and in the order specified by the programmer. ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed.

How much ILP exists in programs is very application specific. In certain fields, such as graphics and scientific computing the amount can be very large. However, workloads such as cryptography exhibit much less parallelism.

Micro-architectural techniques that are used to exploit ILP include:

- Instruction pipelining** where the execution of multiple instructions can be partially overlapped.
- Superscalar execution**, VLIW, and the closely related Explicitly Parallel Instruction Computing concepts, in which multiple execution units are used to execute multiple instructions in parallel.

##### Thread Level Parallelism

Thread parallelism (also known as **Task Parallelism**, **function parallelism** and **control parallelism**) is a form of parallelization of computer code across multiple processors in parallel computing environments. Thread parallelism focuses on distributing execution processes (threads) across different parallel computing nodes. It contrasts to data parallelism as another form of parallelism.

It was later recognized that finer-grain parallelism existed within a single program. A single program might have several threads (or functions) that could be executed separately or in parallel. Some of the earliest examples of this technology implemented input/output processing such as direct memory access as a separate thread from the computation thread. A more general approach to this technology was introduced in the 1970s when systems were designed to run multiple computation threads in parallel. This technology is known as multi-threading (MT). As a simple example, if we are running code on a 2-processor system (CPUs "a" & "b") in a parallel environment and we wish to do tasks "A" and "B", it is possible to tell CPU "a" to do task "A" and CPU "b" to do task "B" simultaneously, thereby reducing the run time of the execution.

Thread parallelism emphasizes the distributed (parallelized) nature of the processing (i.e. threads), as opposed to the data (data parallelism). Most real programs fall somewhere on a continuum between Thread parallelism and Data parallelism.

The pseudocode below illustrates task parallelism:  
program:

```

if CPU="a" then
    do task "A"
else if CPU="b" then
    do task "B"
end if
```

end program

The goal of the program is to do some net total task ("A+B"). If we write the code as above and launch it on a 2-processor system, then the runtime environment will execute it as follows:

- In an SPMD system, both CPUs will execute the code.
- In a parallel environment, both will have access to the same data.
- The "if" clause differentiates between the CPU's. CPU "a" will read true on the "if" and CPU "b" will read true on the "else if", thus having their own task.
- Now, both CPU's execute separate code blocks simultaneously, performing different tasks simultaneously.

Code executed by CPU "a":  
program:

```

do task "A"
end program
```

Code executed by CPU "b":  
program:

do task "B"

end program

This concept can now be generalized to any number of processors.

#### Data parallelism

**Data parallelism** is parallelism inherent in program loops, which focuses on distributing the data across different computing nodes to be processed in parallel. Parallelizing loops often leads to similar (not necessarily identical) operation sequences or functions being performed on elements of a large data structure. Many scientific and engineering applications exhibit data parallelism. A loop-carried dependency is the dependence of a loop iteration on the output of one or more previous iterations. Loop-carried dependencies prevent the parallelization of loops. For example, consider the following pseudocode that computes the first few Fibonacci numbers:

```
PREV1 = 0  
PREV2 = 1  
do:  
    CUR = PREV1 + PREV2  
    PREV1 = PREV2  
    PREV2 = CUR  
    while (CUR < 10)
```

This loop cannot be parallelized because CUR depends on itself (PREV2) and PREV1, which are computed in each loop iteration. Since each iteration depends on the result of the previous one, they cannot be performed in parallel. As the size of a problem gets bigger, the amount of data-parallelism available usually does as well.

#### Process Level Parallelism

**Process-level parallelism** is the use of one or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them. There are many variations on this basic theme, and the definition of process level parallelism can vary with context, mostly as a function of how CPUs are defined (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).

This varies, depending upon who you talk to. In the past, a CPU (Central Processing Unit) was a singular execution component for a computer. Then, multiple CPUs were incorporated into a node. Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit. CPUs with multiple cores are sometimes called "sockets". The result is a node with multiple CPUs, each containing multiple cores.

- During the past 20+ years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that **parallelism is the future of computing**.

- In this same time period, there has been a greater than 1000x increase in supercomputer performance, with no end currently in sight.

#### 6.1.4 Inter-process Communication, Resource Allocation and Deadlock

##### Inter-process communication

In computing, **Inter-process communication (IPC)** is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.

There are several reasons for providing an environment that allows process cooperation:

- Information sharing
- Speedup
- Modularity
- Convenience
- Privilege separation

IPC may also be referred to as inter-thread communication and inter-application communication. The combination of IPC with the address space concept is the foundation for address space independence/isolation.

The single operating system controls the use of system resources in a multiprocessing environment. In this system, multiple jobs or process may be active at one time. The responsibility of operating system or system software is to schedule the execution and to allocate resources. The functions of multiprocessor operating system are:

- An interface between users and machine
- Resource management
- Memory management
- Prevent deadlocks
- Abnormal program termination
- Process scheduling
- Managers security

##### Resource Allocation

In computing, **resource allocation** is necessary for any application to be run on the system. When the user opens any program this will be counted as a process, and therefore requires the computer to allocate certain resources for it to be able to run. Such resources could be access to a section of the computer's memory, data in a device interface buffer, one or more files, or the required amount of processing power.

A computer with a single processor can only perform one process at a time, regardless of the amount of programs loaded by the user (or initiated on start-up). Computers using single processors appear to be running multiple programs at once because the processor quickly alternates between programs, processing what is needed in very small amounts of time. This

process is known as multitasking or *time slicing*. The time allocation is automatic, however higher or lower priority may be given to certain processes, essentially giving high priority programs more/bigger slices of the processor's time.

On a computer with multiple processors different processes can be allocated to different processors so that the computer can truly multitask. Some programs, such as Adobe Photoshop, which can require intense processing power, have been coded so that they are able to run on more than one processor at once, thus running more quickly and efficiently.

#### Deadlock

A process requests resources, if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.

Processes need access to resources in reasonable order. Suppose a process holds resource A and requests resource B, at same time another process holds B and requests A; both are blocked and remain in deadlock.

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Usually the event is release of a currently held resource. None of the processes can run, release resources and then be awakened.

#### 6.1.5 Features of Typical Operating System

Operating system is a system s/w which control and manage the hardware through bios (basic i/o system). Bios is the part of operating system which is built in the system and it runs application s/w on a hardware. It is an interface b/w user and system/s is a complex s/w. In a simple word operating system is that which is active on a ram before switch off of the system. The operating systems key elements are

- A technical layer of software for driving software components.
- A file system for organizing and accessing files logically.
- Simple command language enabling users to run their own programs and manipulating files.

#### OS Features

##### 1. Process Management

Operating system manages the process on hardware level and user level. To create, block, terminate, request for memory, Forking, releasing of memory etc. in multi tasking operating system the multiple processes can be handle and many processes can be created, block and terminate, run etc. It allows multiple process to exist at any time and where only one process can execute at a time. The other process may be performing I/O and waiting. The process manager implements the process abstraction and creating the model to use the CPU. It is the major part of operating system and its major goal is scheduling, process synchronization mechanism and deadlock strategy.

#### 2. File Management

Is a computer program and provide interface with file system. It manages the file like creation, deletion, copy, rename etc files typically display in hierarchical and some file manager provide network connectivity like ftp, nfs, smb or webdav.

#### 3. Memory Management

Is a way to control the computer memory on the logic of data structure? It provides the way to program to allocate in main memory at their request. The main purpose of this manager is; It allocates the process to main memory, minimize the accessing time and process address allocate in a location of primary memory. The feature of memory manager on multi tasking is following:

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

#### 4. Device Management

Allow the user to view its capability and control the device through operating system. Which device may be enabling or disable or install or ignore the functionality of the device. In Microsoft windows operating system the control panel applet is the device manager. It also built on web application server model. Device Manager provides three graphical user interfaces (GUIs). Device manager manage the following:

- Device configuration
- Inventory collection
- S/W distribution
- Initial provisioning

#### 5. Resource management

Is a way to create, manage and allocate the resources? Operating system is a responsible to all activities which is done in computer. Resource manager is the major part of operating system. The main concept of operating system is managing and allocating the resources. The resources of the computer are storage devices, communication and I/O devices etc. These all resources manage and allocate or de-allocate by resource manager.

#### Services Of Operating System

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. The specific services provided, of course, differ from one operating system to another, but we can identify common classes. These operating-system services are provided for the convenience of the programmer, to make the programming task easier.

**Program execution:** The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

**I/O operations:** A running program may require I/O. This I/O may involve a file or an I/O device. For specific devices, special functions may be desired (such as to record a tape drive, or to blank a CRT screen). For efficiency and protection, users usually cannot control I/O devices directly. Therefore the operating system must provide a means to do I/O.

**File-system manipulation:** The file system is of particular interest. Obviously, programs need to read and write files. Programs also need to create and delete files by name.

**Communications:** In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of information are moved between processes by the operating system.

**Error detection:** The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer),

and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. In addition, another set of operating system functions exists not for helping the user, but for ensuring the efficient operation of the system itself. Systems with multiple users can gain efficiency by sharing the computer resources among the users.

**Resource allocation:** When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code. For instance, in determining how best to use the CPU, the operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors. There might also be routines to allocate a tape drive for use by a job. One such routine locates an unused tape drive and marks an internal table. Another routine is used to clear that table. These routines may also allocate printers, modems, and other peripheral devices.

**Accounting:** We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

**Protection:** The owners of information stored in a mult-user computer system may want to control use of that information. When several disjointed processes execute concurrently, it should not be possible for one process to interfere with the others, or with the operating system itself. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with each user having to authenticate himself to the system, usually by means of a password, to be allowed access to the resources. It extends to defending external I/O devices, including modems and network adapters, from invalid access attempts, and to recording all such connections for detection of break-ins. If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

## 6.2 Different Microprocessor Architectures

### 6.2.1 Register Based and Accumulator Based Architecture Accumulator Based Architecture

- Accumulator is a most significant register then compared to other registers and most of the arithmetic and logic operations are performed using the accumulator performed via the accumulator.
- The internal architecture of 8085 shows that the registers B,C,D,E,H and L are connected with the ALU through the accumulator and temporary register.
- Data can only enter into the ALU from accumulator and the out port of the ALU can be stored in accumulator through data bus.

### Register Based Architecture

- All the arithmetic and logical operation consists of the operands of any registers (ABCD etc). The input/output operations here register A and register B are similar.
- The internal architecture of 8086 shows that the registers A, B, C, D and others directly with the ALU.
- Data can enter into the ALU from any registers.
- For I/O operation register A only can be used.
- The advantage of register based architecture is that extensibility and flexibility in programming
- The processor will be enhanced in register based architecture.
- The disadvantage is requirement of complex circuitry.

### 6.2.2 RISC and CISC Architectures

#### CISC Based Architecture

Complex instruction set computers is the acronym for CISC and machine based on this architecture have complex instructions. Most of the personal computers which are used today are based on CISC architecture. Following are the characteristics of CISC machines.

- CISC machines have complex instructions which need a large cycles for execution. The transfer of data among the register is much faster than among memory and processor. In CISC, various instructions based on memory and processor so the processing speed gets reduced.

- Pipelining is the process of fetching one instruction when another instruction is executing in parallel. Due to complex instruction this feature cannot be heavily used in CISC machines.
- Micro-operations form the instruction and instruction form the micro-program which is written in control memory to perform timing and sequencing of the micro-operations implemented in CISC.
- CISC machines have large number of complex instructions based on multiple numbers of addressing modes.
- CISC machines processor does not consist of large number of registers due to large cost. So these machines have to perform various memory read and write operations.
- CISC Machines are preferable where the speed of processor is not the prime issue and where general applications are to be handled. Processors like 8085, 8086, 8088, 8086, 8088 etc are based on CISC processors and even today's pc.

#### RISC based architecture:

- The term RISC represents reduced instruction set computing / computers. It focused on a small set of instruction which simplifies the hardware design and improves the processor performance. Generally RISC processors include the following features:
- The number of instructions is minimized i.e. less than 100 and each can be executed in a single clock cycle the data path cycle time is the time required to fetch the operations from the registers, run them through ALU and store the result back to register which is very small in RISC.
  - the number of addressing modes is relatively low i.e. less than 3 and only few instructions are memory referencing load and store so that RISC permits heavy pipelining.
  - The processing is register intensive, meaning it includes many more registers and most of the computing is performed using registers range from 32 registers to more than 100 registers.
  - There is no micro-programs in RISC machines for interpreting the instructions. The most instructions are directly executed by the hardware.
  - Design considerations include support for high level languages through appropriate selection of instruction and optimization of compilers.
  - Usually in scientific and research oriented tasks where the speed of processors is apex issue, RISC machines are used and replacing CISC now due to reduce the price of hardware.
  - RISC based system are R4400SC from MIPS, PA7100 from HP, Power PC from Apple, IBM and Motorola, super sparc from sun, i860<sup>TM</sup> from Intel etc.

#### 6.2.3 Digital Signal Processors

The real time signals such as pressure, temperature, voice are continuous time varying signals are known as analog signals. The process of conversion of the analog signals into discrete signal means digital signal which reduces the redundancy and make more immune to noise is called digital signal processing. the digital signal processors take input the digital data for that every analog data is to be converted into digital by using A/D converter. For example microphone converts sound into electrical energy i.e. analog, it is sampled by A/D converter and converted into digital one which is fed to the DSP processor. After

performing these data DSP can transfer the discrete data to D/A converter which further to speaker to convert electrical signal to sound.

The whole function is carried out by DSP processors using hardware like microphone, transducer, A/D converter, D/A converter, speaker etc and software like C or MATLAB which carries out FFT (Fast Fourier transform). DSP processors have less processing speed due to very crucial signals need to be operated. The micro-processors and computers we used today are based on von Neumann architecture where the instruction defines both the operation and data.

So the DSP processors should be fast processing and for that we need to design best architecture which follows Harvard Architecture where separate buses for instructions and data are used. DSP chips are specially designed for particular application and they are not used for general type of processing like microprocessors do. There are very few manufacturers for DSP chip, one of them is the Texas instruments, USA. its TMS320C series is worldwide popular and can be used for implementing various types of signal processing application.

Advanced College of Engineering & Management  
Tutorial – I (ALP 8085)  
Microprocessor (BCT/BEX/BEL II / II)

- Add two numbers located at 3030H and 4040H. Display sum on Port 1. If carry is generated, display it on Port 2. Store sum on 5050H.

```
LDA 3030H  
MOV B, A  
LDA 4040H  
ADD B  
STA 5050H  
OUT PORT 1  
JNC L1  
MVI A, 01H  
OUT PORT 2  
L1: HLT
```

- Write an Assembly Language Program that retrieves a data located at 2050H and it displays, if it is even and stores FFH on that location if it is odd.

```
LDA 2050H  
ANI 01H  
JNZ L1  
LDA 2050H  
OUT PORT 1  
HLT  
L1: MVI A, FFH  
STA 2050H  
HLT
```

- Sixteen bytes of data are stored in memory location at 1050H to 105FH. Replace each data byte by FF.

```
LXI H, 1050H  
MVI C, 10H  
L1: MVI M, FFH  
INX H  
DCR C  
JNZ L1  
HLT
```

- Sixteen data are stored in memory location at 1050H to 105FH. Transfer the entire block of data to new location starting at 1070H

```
LXI H, 1050H  
MVI C, 10H  
LXLD, 1070H  
L1: MOV A, M  
STAX D  
INX H  
INX D  
DCR C  
JNZ L1  
HLT
```

- Six bytes are stored in memory locations starting at 2050H. Add all the data bytes, save any carry generated while adding the data bytes. Display entire sum at two output ports and store total carry in 2070H and sum in 2071H.

```
LXI H, 2050H  
MVI C, 06H  
MVI B, 00H  
MVI D, 00H  
L2: MOV A, M  
ADD B  
MOV B, A  
JNC L1  
INR D  
L1: INX H  
DCR C  
JNZ L2  
HLT
```

- If the content of memory location 2050H is greater than or equal to 64H, display 0FH else display FFH.

```
LDA 2050H  
CPI 64H  
JC L1  
MOV A, 0FH  
OUT PORT 1  
HLT  
L1: MOV A, FFH  
OUT PORT 1  
HLT
```

7. We have a list of data stored at memory location starting at 2050H. The end of the data array is indicated by data byte 00H. Add the set of readings. Display the sum at Port 1 and total carry at Port 2.

```
LXI H, 2050H
MVI B, 00H
MVI C, 00H
L3: MOV A, M
    CPI 00H
    JZ L1
    ADD C
    JNZ L2
    INR B
L2: MOV C, A
    INX H
    JMP L3
L1: MOV A, C
    OUT PORT 1
    MOV A, B
    OUT PORT 2
    HLT
```

8. There are two tables holding twenty data whose starting address is 3000H and 3020H respectively. WAP to add the content of first table with the content of second table having same array index. Store sum and carry into the third and fourth table indexing from 3040H and 3060H respectively.

```
LXI D, 3020H
LXI H, 3000H
MVI C, 00H
L2: LDAX D
    ADD M
    MOV B, A
    JC L1
    MVI L, 60H
    MVI I, 60H
    MOV A, L
    ADD C
    SHV L, A
    MVI M, 01H
L1: MVI I, 40H
    MOV A, L
    ADD C
    MOV L, A
    MOV M, B
```

P.90

9. For ten bytes data starting from 1120H, write a program to sort the reading in ascending and in descending order. (Note: For descending, do self)

```
START: LXI H, 1120H
        MVI D, 00H
        MVI C, 0AH
L2:    MOV A, M
        INX H
        CMP M
        JC L1
        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
```

10. A set of ten readings is stored in memory location starting at 1160H. The readings are expected to be positive (<127). WAP to

- Check each reading to determine whether it is positive or negative.
- Reject all negative readings.
- Add all positive readings & display sum in Port 1 and carry in Port 2.

```
MVI B, 00H
MVI C, 00H
MVI D, 0AH
LXI H, 1160H
L2: MOV A, M
    RAL
    JC NEGLECT
    RAR
    ADD B
    JC LI
    MOV B, A
    CPI 14H
    JC L2
    HLT
L1: NEGLECT:
```

11. A set of six data bytes is stored starting from memory location 2050H. The set includes some blank spaces (bytes with zero values). WAP to eliminate the blanks from the block.

```
MVI C, 06H
LXI H, 2050H
LXI B, 2050H
L2: MOV A, M
CPI 00H
JZ L1
STAX B
INX B
L1: INX H
DCR C
JNZ L2
HLT
```

12. A set of eight data bytes (4 Pairs) are stored in memory locations starting from 1040H. WAP to add two bytes at a time and store the sum in same memory location, sum replacing the first byte and the carry replacing the second byte. If any pair does not generate a carry, the memory location of the second byte should be cleared i.e. store 00H over there.

```
MVI C, 04H
LXI H, 1040H
L2: MOV A, M
INX H
ADD M
DCX H
MOV M, A
INX H
MVI M, 00H
JNC L1
MVI M, 01H
L1: INX H
DCR C
JNZ L2
HLT
```

13. WAP to read BCD number stored at memory location 2020H and converts it into binary equivalent and finally stores that binary pattern into memory location 2030H.  
[Note: BCD number is the combination from 0 to 9]

```
MVI C, 0AH
LXI H, 2020H
MOV A, M
ANI F0H
RCR
RCR
RCR
MOV B, A
MOV A, 00H
L1: ADD B
DCR C
JNZ L1
MOV D, A
MOV A, M
ANI 0FH
ADD D
STA 2030H
HLT
```

14. A binary number (Suppose FF<sub>16</sub>) is stored in memory location 2020H. Convert the number into BCD and store each BCD as two unpacked BCD digits in memory location from 2030H.

```
LXI SP, 2000H
LXI H, 2020H
MOV A, M
CALL PWRTE
HLT
PWRTE: LXI H, 2030H
MVI B, 64H
CALL BINBCD
MOV M, D
INX H
MVI B, 0AH
CALL BINBCD
MOV M, D
INX H
MVI H, 0A
RET
BINBCD: MVI D, 00H
NEXT: INR D
SUB B
JNC NEXT
DCR D
ADD B
RET
```

15. An 8 bit binary number is stored in memory location 1120H. WAP to store ASCII codes of these binary digits (0 to F) in location 1160H and 1161H.

|               |       |         |
|---------------|-------|---------|
| LXI SP, 2000H | CODE: | CPI 0AH |
| LXI H, 1120H  |       | JC LI   |
| LXI D, 1160H  |       | ADD 07H |
| MOV A, M      | LI:   | ADD 30H |
| ANI 0FH       |       | RET     |
| RRC           |       |         |
| RRC           |       |         |
| RRC           |       |         |
| CALL CODE     |       |         |
| STAX D        |       |         |
| INX D         |       |         |
| MOV A, M      |       |         |
| ANI 0FH       |       |         |
| CALL CODE     |       |         |
| STAX D        |       |         |
| HLT           |       |         |

16. WAP to convert ASCII at location 1040H to binary and store at location 1050H.

|               |       |         |
|---------------|-------|---------|
| LXI SP, 2000H | CODE: | CPI 40H |
| LXI H, 1040H  |       | JC LI   |
| LXI D, 1050H  |       | SUB 07H |
| MOV A, M      | LI:   | SUB 30H |
| ANI 0FH       |       | RET     |
| RRC           |       |         |
| RRC           |       |         |
| RRC           |       |         |
| CALL CODE     |       |         |
| STAX D        |       |         |
| INX D         |       |         |
| MOV A, M      |       |         |
| ANI 0FH       |       |         |
| CALL CODE     |       |         |
| STAX D        |       |         |
| HLT           |       |         |

17. A set of three packed BCD numbers are stored in memory locations starting at 1150H. The seven segment codes of digits 0 to 9 for a common cathode LED are stored in memory locations starting at 1170H and the output buffer memory is reserved at 1190H. WAP to unpack the BCD number and select an appropriate seven segment code for each digit. The codes should be stored in output buffer memory.

|                |       |              |
|----------------|-------|--------------|
| LXI SP, 2999H  | CODE: | PUSH H       |
| LXI H, 1150H   |       | LXI H, 1170H |
| MVI D, 03H     |       | ADD L        |
| LXI B, 1190H   |       | MOV L, A     |
| NEXT: MOV A, M |       | MOV A, M     |
| ANI F0H        |       | STAX B       |
| RRC            |       | POP H        |
| RRC            |       | RET          |
| RRC            |       |              |
| CALL CODE      |       |              |
| INX B          |       |              |
| MOV A, M       |       |              |
| ANI 0FH        |       |              |
| CALL CODE      |       |              |
| INX B          |       |              |
| INX H          |       |              |
| DCR D          |       |              |
| JNZ NEXT       |       |              |
| HLT            |       |              |

18. A multiplicand is stored in memory location 1150H and a multiplier is stored in location 1151H. WAP to multiply these numbers and store result from 1160H.

|              |              |              |
|--------------|--------------|--------------|
| MVI B, 08H   | L2:          | MVI B, 08H   |
| MVI D, 00H   | JNC LI       | MVI D, 00H   |
| LXI H, 1150H | LXI H, 1151H | LXI H, 1150H |
| MOV A, M     | MOV A, M     | MOV A, M     |
| MOV E, A     | RAR          | MOV A, M     |
| LXI H, 1151H | JNC LI       | MOV A, M     |
| MOV A, M     | LXI H, 0000H | RAR          |
| DAD D        | DAD D        | JNC LI       |
| L1: XCHG     | XCHG         | LXI H, 0000H |
| DAD H        | DAD H        | DAD D        |
| XCHG         | XCHG         | XCHG         |
| DCR B        | DCR B        | DCR B        |
| LNZ L2       | LNZ L2       | LNZ L2       |
| HLT          | HLT          | HLT          |

19. A set of ten packed BCD numbers is stored in the memory location starting at 1150H. WAP to add these numbers in BCD. If carry is generated save it in register B and adjust it for BCD. The final sum is less than 9999BCD.

```
LXI SP, 2000H
LXI H, 1150H
MVI C, 0AH
XRA A
MOV B, A
L1: CALL ADD
INX H
DCR C
JNZ L1
HLT
```

```
ADD: ADD M
DAA
RNC
MOV D, A
MOV A, B
ADI 01H
DAA
MOV B, A
MOV A, D
RET
```

20. A dividend is stored in memory location 2020H and a divisor is stored in 2021H. WAP to divide these numbers and store quotient and remainder from 2040H.

```
MVI C, 00H
LXI H, 2021H
MOV A, M
MOV D, A
DCX H
MOV B, M
L2: MOV A, B
SUB D
JC L1
MOV B, A
INR C
JMP L2
L1: MOV L, C
MOV H, B
SHLD 2040H
HLT
```

Compiled By: Er. Hari Aryal

email: haryal@hotmail.com

9

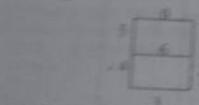
21. Write a program for 8085 to convert and copy the ten lower case ASCII codes to upper case from memory location 9050H to 90A0H if any, otherwise copy as they are. Assume there are fifty codes in the source memory. [Note: ASCII code for A-65, Z-90, a-97, z-122] [2063 Kartik]

```
LXI H, 9050H
LXI D, 90A0H
MVI C, 32H
L2: MOV A, M
CPI 60H
JC L1
SUI 20H
L1: STAX D
DCR C
JNZ L2
HLT
```

22. Write a program for 8085 to add ten 16-bit BCD numbers from location 4050H and store 24-bit BCD result at the end of the ten given numbers. [2062 Chaitra]

```
LXI B, 4050H ; Starting location of the 16-bit BCD Numbers
LXI D, 0000H
LXI H, 0000H
MVI A, 00H
L2: LDAX B
ADD L
INX B
LDAX B
ADC H
JNC L1
INR E
L1: INX B
MOV A, C
CPI 0AH
JC L2
MOV A, L
STAX B
INX B
MOV A, H
STAX B
INX B
MOV A, E
STAX B
HLT
```

23. Write an 8085 program to display the BCD digits from 0 to 9 the seven segments are as the following diagram. Use the activating data bits same as the segments number as in figure below:



[2009 Shrawan]

```

L1: SP, 2000H
MOV H, 2000H
MOV M, 3FH
MOV A, 0E
MOV B, 0E
MOV C, 0E
MOV D, 0E
MOV E, 0E
MOV F, 0E
MOV G, 0E
MOV P, 0E
MOV Q, 0E
MOV R, 0E
MOV S, 0E
MOV T, 0E
MOV U, 0E
MOV V, 0E
MOV W, 0E
MOV X, 0E
MOV Y, 0E
MOV Z, 0E

```

L1: JZ L2 ; When the BCD digit is located

```

MOV H, 0AH
MOV M, 0AH
MOV A, 0AH
MOV B, 0AH
MOV C, 0AH
MOV D, 0AH
MOV E, 0AH
MOV F, 0AH
MOV G, 0AH
MOV P, 0AH
MOV Q, 0AH
MOV R, 0AH
MOV S, 0AH
MOV T, 0AH
MOV U, 0AH
MOV V, 0AH
MOV W, 0AH
MOV X, 0AH
MOV Y, 0AH
MOV Z, 0AH

```

```

L2: SP, 2000H
MOV H, 2000H
MOV M, 3FH
MOV A, 0E
MOV B, 0E
MOV C, 0E
MOV D, 0E
MOV E, 0E
MOV F, 0E
MOV G, 0E
MOV P, 0E
MOV Q, 0E
MOV R, 0E
MOV S, 0E
MOV T, 0E
MOV U, 0E
MOV V, 0E
MOV W, 0E
MOV X, 0E
MOV Y, 0E
MOV Z, 0E

```

Compiled By: Er. Hari Aryal

[email: haryal@hotmail.com](mailto:haryal@hotmail.com)

24. Write a program for 8085 to change the bit 7 of ten numbers stored at address 2000H if the numbers are larger than or equal to 80H.

[2009 Adhikari]

```

L1: H, 2000H
MOV C, 0AH
MOV A, M
CPX B0H
JC L1
XRI 20H
MOV M, A
L1: INX H
DCR C
JNZ L2

```

25. Write a program for 8085 to find the smallest number among ten numbers stored at memory location 4500H.

[2009 Bhadra]

```

L1: H, 4500H
MOV C, 0AH
MOV A, M
INX H
CMP M
JC L1
MOV B, A
MOV A, M
MOV M, B
L1: DCR C
JNZ L2
OUT PORT 1
HLT

```

Compiled By: Er. Hari Aryal

[email: haryal@hotmail.com](mailto:haryal@hotmail.com)

26. Someone has damaged a program written at 4050H for 8085 microprocessor. The damaging is done by changing the bit D<sub>2</sub> and bit D<sub>3</sub> of each byte. The size of the program is 100 bytes. Now write a program for 8085 to correct this damaged program. [2060 Chaitra]

```
LXI H, 4050H
MVI C, 64H
L1: MOV A, M
ANI 80H , 10000000 B
RRC
RRC
MOV B, A
MOV A, M
ANI 20H , 00100000 B
RLC
RLC
MOV C, A
MOV A, M
ANI 5FH , 01011111B
ORA B
ORA C
STAX H
INX H
DCR C
JNZ L1
HLT
```

27. The temperature of two furnaces being monitored by a microprocessor based system. A set of readings of the first furnace recorded by thermal sensor is stored at memory locations starting at 4050H. Corresponding readings from the second furnace is stored at the memory location starting at 4070H. Each reading from the first furnace is expected to be higher than the corresponding reading from the second furnace. Among the eight bit data bit D<sub>1</sub> is used to test the validity of the data. Write an 8085 program to compare valid data from the two tables, if data from first table is larger than the corresponding data from the second table store 01H in the corresponding memory of the third location starting at 4090H and display 01H to indicate the normal operation else store FFH in the corresponding memory location and display FFH in the port to indicate the emergency. When emergency condition is reached stop the operation. [2060 Jethal]

```
LXI B, 4050H
LXI H, 4070H
LXI D, 4090H
L2: LDAX B
CMP M
JC L1
JZ L1
MVI A, 01H
STAX D
OUT PORT
INX B
IND H
INX D
JMP L2
L1: MVI A, FFH
STAX D
OUT PORT
HLT
```

28. Write a program to transfer eight-bit numbers from 9080H to 9090H if bit D<sub>1</sub> is 1 and D<sub>2</sub> is 0. Otherwise transfer data by changing bit D<sub>2</sub> and D<sub>3</sub> from 1 to 0 or from 0 to 1. Assume there are ten numbers. [2064 Shrawan]

```
LXI H, 9080H
LXI D, 9090H
MVI C, 0AH
L2: MOV A, M
ANI 28H
CPI 20H
JZ L1
MOV A, M
XRI 44H
MOV M, A
L1: MOV A, M
STAX D
INX H
INX D
DCR C
JNZ L2
HLT
```

29. There are two tables T1, T2 in memory having ten eight bit data in each. Write a program for RISC to find the difference of the corresponding element of these two tables. Store the result of each operation on the corresponding element of the third table. Remember that the result should not be negative . It should be  $|T1 - T2|$ . [2064 Poush]

LXISP, 2999H : TABLE T1  
LXI H, 5000H : TABLE T2  
LXI D, 6000H : COUNTER FOR 10 DATA  
MVI C, 0AH

L1: LDAX D  
MOV B, A  
MOV A, M  
CMP B  
INC L2  
MOV A, B  
MOV B, M  
SUB B

L2: PUSHD  
MVI D, 70H : TABLE T3  
STAX D  
POP D

INX H  
INX D  
DCR C  
JNZ L1  
HLT

30. Write a program for RISC to transfer data from a table to another if the number of ones in the data is greater than four else store 0F in the next table. [2065 Kartik]

LXI H, 5000H : SOURCE TABLE  
LXI D, 6000H : DESTINATION TABLE  
ST: MVI C, 0FH : NO OF BITS  
MVI B, 00H : NO OF 1'S

MVW A, M  
L1: RSC

INC L2  
INR B

L2: DCR C  
JNZ L1  
MOV A, B  
CPW 04H

MVI A, 00H  
JZ L3  
JZ L2  
MOV A, M

L3: STAX D  
INX H  
INX D  
MOV A, E  
CPW 04H  
JNZ ST  
HLT

SUPPOSE TABLE FOR 10 DATA

## section B

1. WAP to display 'WELCOME IN ACEM' using character display function.

```
.model small
.stack 32h
.data
    str DB 'WELCOME IN ACEM',0Dh,0Ah,'$'
    length DW $-str
.code
    Main proc far
    Mov ax,@data
    Mov ds,ax

    Lea dx,str
    Mov cx,length
    Mov bx,00h
    Step: Mov dl,str[bx]
    Mov ah,02h
    Int 21h
    Inc bx
    Loop step

    Mov ax,4C00h
    Int 21h
Main endp
End main
```

2. WAP to read the string using the character reading function & display the string using the character displaying function.

```
.model small
.stack 64h
.data
```

```
Prog_str level byte
Max_char db 20
Act_char db
Ent_char db 20 dup(?)

.code
Main proc far
Mov ax,@data
Mov bx,00
Lea dx,prog_str
Mov ah,0ah
Int 21h

Mov ah,02h
Mov dl,0dh
Int 21h

Mov bl,act_char
Mov ent_char[bx],'$'
Lea dx,ent_char
Mov ah,09h
Int 21h

Mov ax,4C00h
Int 21h
Main endp
End main
```

3. WAP to display same as mentioned in question no 1 using string display function.

```
.model small
.stack 64h
.data
    Str db 'WELCOME IN NATIONAL COLLEGE',0dh,0ah,'$'
```

```
.code
Main proc far
Mov ax,@data
Mov ds,ax
Lea dx,str
Mov ah,09h
Int 21h

Mov ax,4c00h
Int 21h
Main endp
End main
```

4. WAP to read the string using string reading function & display the string using string displaying function.

```
.model small
.stack 64h
.data
```

```
.code
Main proc far
Mov ax,@data
Mov ds,ax

Mov ax,4c00h
Int 21h
Main endp
End main
```

5. WAP to display the stored string in reverse order.

```
.model small
.stack 64h
```

```
.data
Str1 db "my name is dhiraj", '$'
Str2 db 50 dup(?)
```

```
.code
Main proc far
Mov ax,@data
Mov ds,ax
```

```
Mov si,offset str1
Mov di,offset str2
```

```
Cont: Mov dl,[si]
```

```
Cmp dl,'$'
```

```
Jc down
```

```
Inc si
```

```
Inc bl
```

```
Jmp cont
```

```
Down: Mov cl,bl
```

```
Mov ch,00h
```

```
Dec si
```

```
Uppe: Mov al,[si]
```

```
Mov [di],al
```

```
Dec si
```

```
Inc di
```

```
Loop uppe
```

```
Mov ah,09h
```

```
Mov dx,offset str2
```

```
Int 21h
```

```
Mov ax,4c00h
```

```
Int 21h
```

```
Main endp
```

```
End main
```

6. WAP to read the string & display each word in separate line.

```
.model sm, ff  
.stack 64h  
.data  
    Max char db 20  
    Actchar db ?  
    Enter char db 20 dup(?)  
.code  
    Main proc far  
        Mov ax,@data  
        Mov ds,ax  
        Mov dx,offset maxchar  
        Mov ah,0ah  
        Int 21h  
  
        Mov dl,0dh  
        Mov ah,02h  
        Int 21h  
  
        Mov ch,00h  
        Mov cl,actchar  
        Lea bx,entchar  
        Prod: Mov dl,[bx]  
        Cmp dl  
        Jmp down  
        Mov dl,00h  
        Int 21h  
    Down: Mov dl,0ah  
        Int 21h  
        Add bx,01h  
    Loop prod  
        Mov ax,4c00h  
  
Main endp
```

End main

7. WAP to read the string & convert it into upper case if it is in lower case & display the resulting string in next line.

```
.model small  
.stack 64h  
.data  
    Maxchar db 20  
    Actchar db ?  
    Str db 20 dup(?)  
.code  
    Main proc far  
        Mov ax,2,data  
        Mov ds,ax  
        Lea dx,maxchar  
        Mov ah,0ah  
        Int 21h  
  
        Mov ah,02h  
        Mov dl,0ah  
        Int 21h  
  
        Mov cl,actchar  
        Mov ch,00h  
        Mov bx,00h  
    Step: Mov dl,str[bx]  
        Cmp dl,97  
        Jc jump  
        Cmp dl,123  
        Jnc jump  
        Add dl 32  
        Mov ah,02h
```

Jump: Int 21h  
Inc bx  
Loop step

Mov ax,4c00h  
Int 21h  
Main endp  
End main

8. Write an ALP to read the string from user, convert it to upper case, count the no of words & display each word in each line.

.model small  
.stack 64h  
.data

.code

Main proc far  
Mov ax,@data  
Mov ds,ax

Mov ax,4c00h  
Int 21h  
Main endp  
End main

9. Write an ALP to get a string input, count no of vowels & display message 'even vowels' on the screen if the count is even otherwise display 'odd vowels'.

.model small  
.stack 64h  
.data

.code  
Main proc far  
Mov ax,@data  
Mov ds,ax

Mov ax,4c00h  
Int 21h  
Main endp  
End main

10. Write an ALP for 8085 to read string & find the no of alphabets, numerals & other characters. Display the different counts.

.model small  
.stack 64h  
.data

.code  
Main proc far  
Mov ds,ax

Mov ax,4c00h  
Int 21h  
Main endp  
End main

11. Write an ALP for 8086 to sort an array of 10 numbers stored in the memory. Display the number in the screen after sorting.

.model small  
.stack 64h

.data  
Arrnum db 3,2,1,4,5,6,7,8,9,8  
.code  
Main proc far  
Mov ax,@data  
Mov ds,ax  
Step1: mov si,offset arrnum  
Mov bl,00h  
Mov cx,0004h  
Step2: mov al,[si]  
Inc si  
Cmp al,si  
Jc down  
  
Mov dl,[si]  
Mov [si],al  
Dec si  
Mov [si],dl  
Inc si  
Mov bl,01h  
Down:Loop step2  
Dec bl  
Jz step1  
  
Mov si,offset arrnum  
Mov ch,0  
Mov cl,10  
Step3: Mov ah,0  
Mov bx,0  
Mov al,[si]  
Push cx  
Mov cx 10  
Step4: Mov dx,0

;{dx:ax/cx=ax;dx/cx}  
Div cx  
Add dx,03h  
Push dx  
Inc bx  
Cmp ax,0  
Jz step3  
Mov ah,02h  
Mov cx,bx  
Popping:Pop dx  
Int 21h  
Loop popping  
Inc si  
Pop cx  
Loop step4  
Mov ax,4c00h  
Int 21h  
Main endp  
End main

12.WAP to read string from user & make that in alphabetical order.

.model small  
.stack 64h  
.data  
.code  
Main proc far  
Mov ax,@data  
Mov ds,ax

```
Mov ax,4000h  
Int 21h  
Main endp  
End main
```