UNIT I

Intel 8085 Microprocessor: Introduction - Need for Microprocessors – Evolution – Intel 8085 Hardware - Architecture – Pin description - Internal Registers – Arithmetic and Logic Unit – Control Unit – Instruction word size - Addressing modes – Instruction Set – Assembly Language Programming - Stacks and Subroutines - Timing Diagrams. Evolution of Microprocessors – 16-bit and 32-bit microprocessors.

1. What is microprocessor? ****

A microprocessor is a multipurpose, programmable, register based; clock-driven, electronic device that reads binary instructions from a storage device called memory accepts binary data as input and processes data according to those instructions and provides result as output. The power supply of 8085 is +5V and clock frequency in 3MHz.

2. Define ROM?

A memory that stores binary information permanently. The information can be read from this memory but cannot be altered.

3. **List few applications of microprocessor-based system. ****

It is used:

- For measurements, display and control of current, voltage, temperature, pressure, etc.
- For traffic control and industrial tool control.
- For motor speed control of machines.
- For data acquisition system etc.

4. What is an Assembler?

A computer program that translate an assembly language program from mnemonics to the binary machine code of a computer.

5. What are the four primary operations of a MPU? **

1. Memory read
2. Memory write
3. I/O read
4. I/O write

6. What do you mean by address bus? ****

A group of lines that are used to send a memory address or a device address from the MPU to the memory location or a peripheral. The 8085 microprocessor has 16 address lines.

7. How many memory locations can be addressed by a microprocessor with 14 address lines?

The 8085 MPU with its 14-bit address is capable of addressing 214=16,384 (ie) 16K memory locations.

8. Why is the data bus bi-directional?

The data bus is bi-directional because the data flow in both directions between the MPU and memory and peripheral devices.

9. What is the function of the accumulator? ****

The accumulator is the register associated with the ALU operations and sometimes I/O operation s. It is an integral part of ALU. It holds one of data to be processed b y ALU. It also temporarily stores the result of the operation performed by the ALU.

10. Define control bus? ***

This is single line that is generated by the MPU to provide timing of various operations.

11. What is a flag? Write the flags of 8085. ****

The data conditions, after arithmetic or logical operations, are indicated by setting or resetting the flip-flops called flags.

The 8085 has nine flags and they are

1. Carry Flag (CF)
2. Parity Flag (PF)
3. Auxiliary carry Flag (AF)
4. Zero Flag (ZF)
5. Sign Flag (SF)

12. Why are the program counter and the stack pointer 16-bit registers? ****

Memory locations for the program counter and stack pointer have 16-dit address. So the PC and SP have 16-bit registers.

13. Define memory word?

The number of bits stored in a register is called a memory word.

14. Explain the function of ALU and IO/M signals in the 8085 architecture? ****

The ALU signal goes high at the beginning of each machine cycle indicating the availability of the address on the address bus, and the signal is used to latch the low-order address bus. The IO/M signal is a status signal indicating whether the machine cycle is I/O or memory operation. The IO/M signal is combined with the RD and WR control signals to generate IOR, IOW, MEMW, MEMR.

15. Write down the control and status signals? ****

- Two control signals and three status signals
- Control signals: RD and WR
- Status signals: IO/M, S1, and S2

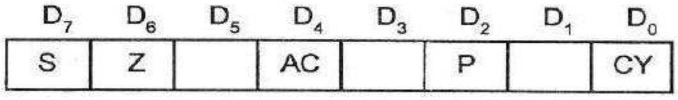
16. Define machine cycle? ****

Machine cycle is defined, as the time required completing one operation of accessing memory, I/O, or acknowledging an external request.

17. Define T-state? *

T-state is defined as one subdivision of the operation of performed in one clock period.

18. Give the bit positions reserved for the flags? *



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

Fig 1.7 : Bit positions of various flags in the flag register of 8085

19. Define instruction cycle? *

Instruction cycle is defined, as the time required completing the execution of the instruction.

20. List the allowed register pairs of 8085.

- B-C register pair
- D-E register pair
- H-L register pair

21. Mention the purpose of SID and SOD lines *

- **SID (Serial input data line):**
- It is an input line through which the microprocessor accepts serial data.
- **SOD (Serial output data line):**
- It is an output line through which the microprocessor sends output serial data.

22. What is an Opcode?

The part of the instruction that specifies the operation to be performed is called the operation code or opcode.

23. What is the function of IO/M signal in the 8085?

It is a status signal. It is used to differentiate between memory locations and I/O operations. When this signal is low (IO/M = 0) it denotes the memory related operations. When this signal is high (IO/M = 1) it denotes an I/O operation.

24. List out the five categories of the 8085 instructions. Give examples of the instructions for each group. *

- Data transfer group – MOV, MVI, LXI.
- Arithmetic group – ADD, SUB, INR.
- Logical group –ANA, XRA, CMP.

- ➢ Branch group – JMP, JNZ, CALL
- ➢ Stack I/O and Machine control group – PUSH, POP, IN, HLT.

25. Explain the difference between a JMP instruction and CALL instruction. **

A JMP instruction permanently changes the program counter. A CALL instruction leaves information on the stack so that the original program execution sequence can be resumed.

26. Explain the purpose of the I/O instructions IN and OUT.

The IN instruction is used to move data from an I/O port into the accumulator. The OUT instruction is used to move data from the accumulator to an I/O port. The IN & OUT instructions are used only on microprocessor, which use a separate address space for interfacing.

27. What is the difference between the shifts and rotate instructions?

A rotate instruction is a closed loop instruction. That is, the data moved out at one end is put back in at the other end. The shift instruction loses the data that is moved out of the last bit locations.

28. What is meant by Wait State? ***

This state is used by slow peripheral devices. The peripheral devices can transfer the data to or from the microprocessor by using READY input line. The microprocessor remains in wait state as long as READY line is low. During the wait state, the contents of the address, address/data and control buses are held constant.

29. Define instruction cycle, machine cycle and T-state ****

Instruction cycle is defined, as the time required completing the execution of an instruction. Machine cycle is defined as the time required completing one operation of accessing memory, I/O or acknowledging an external request. T-cycle is defined as one subdivision of the operation performed in one clock period.

30. What is the use of ALE ****

The ALE is used to latch the lower order address so that it can be available in T2 and T3 and used for identifying the memory address. During T1 the ALE goes high, the latch transparent ie, the output changes according to the input data, so the output of the latch is the lower order address. When ALE goes low the lower order address is latched until the next ALE.

31. How many machine cycles does 8085 have, mention them ***

The 8085 have seven machine cycles. They are

- Opcode fetch
- Memory read
- Memory write
- I/O read
- I/O write
- Interrupt acknowledge
- Bus idle

32. Explain the signals HOLD, READY and SID. *

HOLD indicates that a peripheral such as DMA controller is requesting the use of address bus, data bus and control bus. READY is used to delay the microprocessor read or write cycles until a slow responding peripheral is ready to send or accept data. SID is used to accept serial data bit by bit.

33. Explain LDA, STA and DAA instructions

LDA copies the data byte into accumulator from the memory location specified by the 16-bit address. STA copies the data byte from the accumulator in the memory location specified by 16-bit address. DAA changes the contents of the accumulator from binary to 4-bit BCD digits.

34. Explain the different instruction formats with examples. **

The instruction set is grouped into the following formats

- One byte instruction MOV C,A
- Two byte instruction MVI A,39H
- Three byte instruction JMP 2345H

35. What is the use of addressing modes, mention the different types

The various formats of specifying the operands are called addressing modes, it is used to access the operands or data. The different types are as follows

- Immediate addressing
- Register addressing
- Direct addressing
- Indirect addressing
- Implicit addressing

36. What is the use of bi-directional buffers?

It is used to increase the driving capacity of the data bus. The data bus of a microcomputer system is bi-directional, so it requires a buffer that allows the data to flow in both directions.

37. Define stack and explain stack related instructions **

The stack is a group of memory locations in the R/W memory that is used for the temporary storage of binary information during the execution of the program. The stack related instructions are PUSH & POP.

38. Compare CALL and PUSH instructions *****

CALL	PUSH
When CALL is executed the microprocessor automatically stores the 16-bit address of the instruction next to CALL on the stack	The programmer uses the instruction PUSH to save the contents of the register pair on the stack
When CALL is executed the stack pointer is decremented by two	When PUSH is executed the stack pointer register is decremented by two.

39. What is Microcontroller and Microcomputer. **

Microcontroller is a device that includes microprocessor; memory and I/O signal lines on a single chip, fabricated using VLSI technology. Microcomputer is a computer that is designed using microprocessor as its CPU. It includes microprocessor, memory and I/O.

40. Compare RET and POP. ****

RET	POP
RET transfers the contents of the top two locations of the stack to the PC	POP transfers the contents of the top two locations of the stack to the specified register pair
When RET is executed the SP is incremented by two	When POP is executed the SP is incremented by two
Has 8 conditional RETURN instructions	No conditional POP instructions

41. What is assembly language programming (ALP)? List its field.

Assembly language programming is a program written in a mnemonics or set of instruction.

ALP contain 4 fields,

- Label
- Opcode
- Operand
- Comments

Unit: I

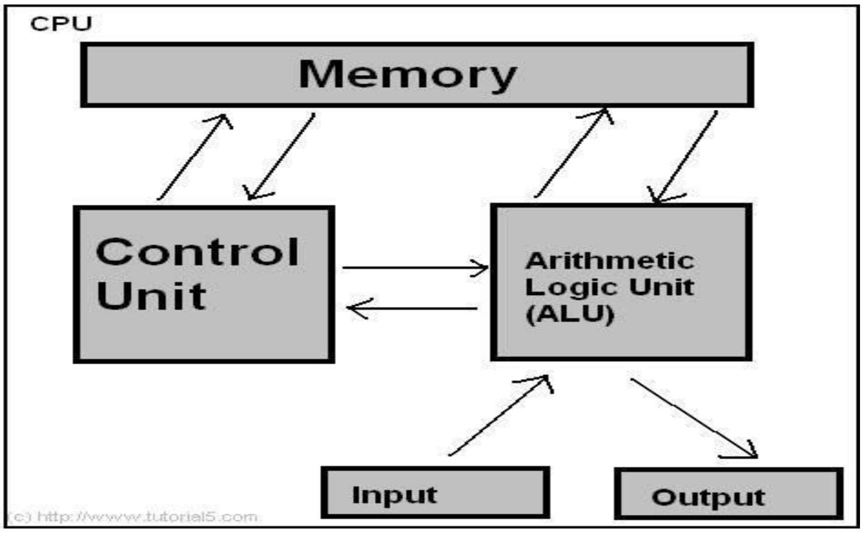
Intel 8085 Microprocessor: Introduction - Need for Microprocessors – Evolution – Intel 8085 Hardware - Architecture – Pin description - Internal Registers – Arithmetic and Logic Unit – Control Unit – Instruction word size - Addressing modes – Instruction Set – Assembly Language Programming - Stacks and Subroutines - Timing Diagrams - Evolution of Microprocessors – 16-bit and 32 - bit microprocessors.

1. Introduction

Definition of the Microprocessor:

The microprocessor is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a result.

"CPU is in the form of chip". The major component of microprocessor is CPU, memory, input and output devices.



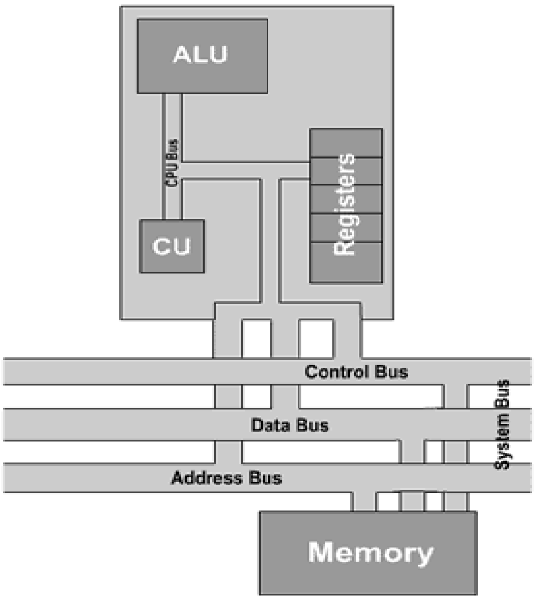
Basic Concepts of Microprocessors & Differences between:

Microcomputer –a computer with a microprocessor as its CPU. Includes memory, I/O etc.,

Microprocessor –silicon chip which includes ALU, register circuits & control circuits.

Microcontroller –silicon chip which includes microprocessor, memory & I/O in a single package.

1.1 GENERAL ARCHITECTURE OF MICROPROCESSOR



Organization of Microprocessor

The microprocessor is sometimes referred to as the 'brain' of the personal computer, and is responsible for the processing of the instructions which make up computer software. It houses the central processing unit, commonly referred to as the CPU.

CPU Structure This section, using a simplified model of a central processing unit as an example, takes you through the role of each of the major constituent parts of the CPU.

The simplified model consists of five parts, which are:

Arithmetic & Logic Unit (ALU) The part of the central processing unit that deals with operations such as addition, subtraction, and multiplication of integers and Boolean operations. It receives control signals from the control unit telling it to carry out these operations. For more, click the title above.

Control Unit (CU) This controls the movement of instructions in and out of the processor, and also controls the operation of the ALU. It consists of a decoder, control logic circuits, and a clock to ensure everything happens at the correct time. It is also responsible for performing the instruction execution cycle.

Register Array This is a small amount of internal memory that is used for the quick storage and retrieval of data and instructions. All processors include some common registers used for specific functions, namely the program counter, instruction register, accumulator, memory address register and stack pointer.

System Bus This is comprised of the control bus, data bus and address bus. It is used for connections between the processor, memory and peripherals, and transfers of data between the various parts.

Memory The memory is not an actual part of the CPU itself, and is instead housed elsewhere on the motherboard. However, it is here that the program being executed is stored, and as such is a crucial part of the overall structure involved in program execution.

2. EVOLUTION OF MICROPROCESSORS

- **4-bit Microprocessors**

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

- **8-bit Microprocessors**

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

- **16-bit Microprocessors**

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

- **32-bit Microprocessors**

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

- **Pentium Series**

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

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

We divide the years of development of microprocessors as 5 generations.

- **First generation (1971 – 73)**

Intel Corporation introduced 4004, the first microprocessor in 1971. It is evolved from the development effort while designing a calculator chip.

There were three other microprocessors in the market during the same period:

- Rockwell International's PPS-4 (4 bits)
- Intel's 8008 (8 bits)
- National Semiconductor's IMP-16 (16 bits)

They were fabricated using PMOS technology which provided low cost, slow speed and low output currents. They were not compatible with TTL.

- **Second Generation (1974 – 1978)**

Some of the popular processors were:

- Motorola's 6800 and 6809
- Intel's 8085
- Zilog's Z80

They were manufactured using NMOS technology. This technology offered faster speed and higher density than PMOS It is TTL compatible

- **Third generation microprocessors (1979 – 80)**

This age is dominated by 16 – bits microprocessor some of them were:

- Intel's 8086/80186/80286
- Motorola's 68000/68010

They were designed using HMOS technology HMOS provides some advantages over NMOS as Speed-power-product of HMOS is four times better than that of NMOS HMOS can accommodate twice the circuit density compared to NMOS Intel used HMOS technology to recreate 8085A and named it as 8085AH with a higher price tag.

- **Fourth Generation (1981 – 1995)**

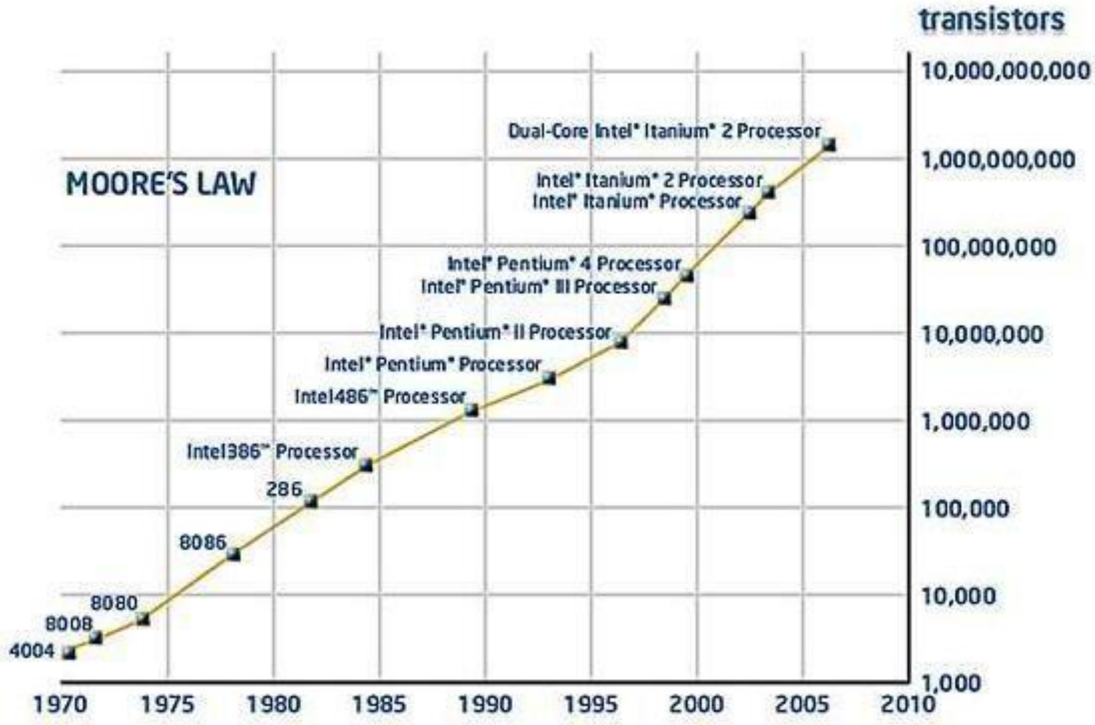
- This era marked the beginning of 32 bits microprocessors.
- Intel introduced 432, which was bit problematic
- Then a clean 80386 in launched.
- Motorola introduced 68020/68030.

They were fabricated using low-power version of the HMOS technology called HCMOS.

Motorola introduced 32-bit RISC processors called MC88100.

- **Fifth Generation (1995 – till date)**

This age the emphasis is on introducing chips that carry on-chip functionalities and improvements in the speed of memory and I/O devices along with introduction of 64-bit microprocessors. Intel leads the show here with Pentium, Celeron and very recently dual and quad core processors working with up to 3.5GHz speed.



3. HARDWARE ARCHITECTURE OF INTEL -8085

ACCUMULATOR:

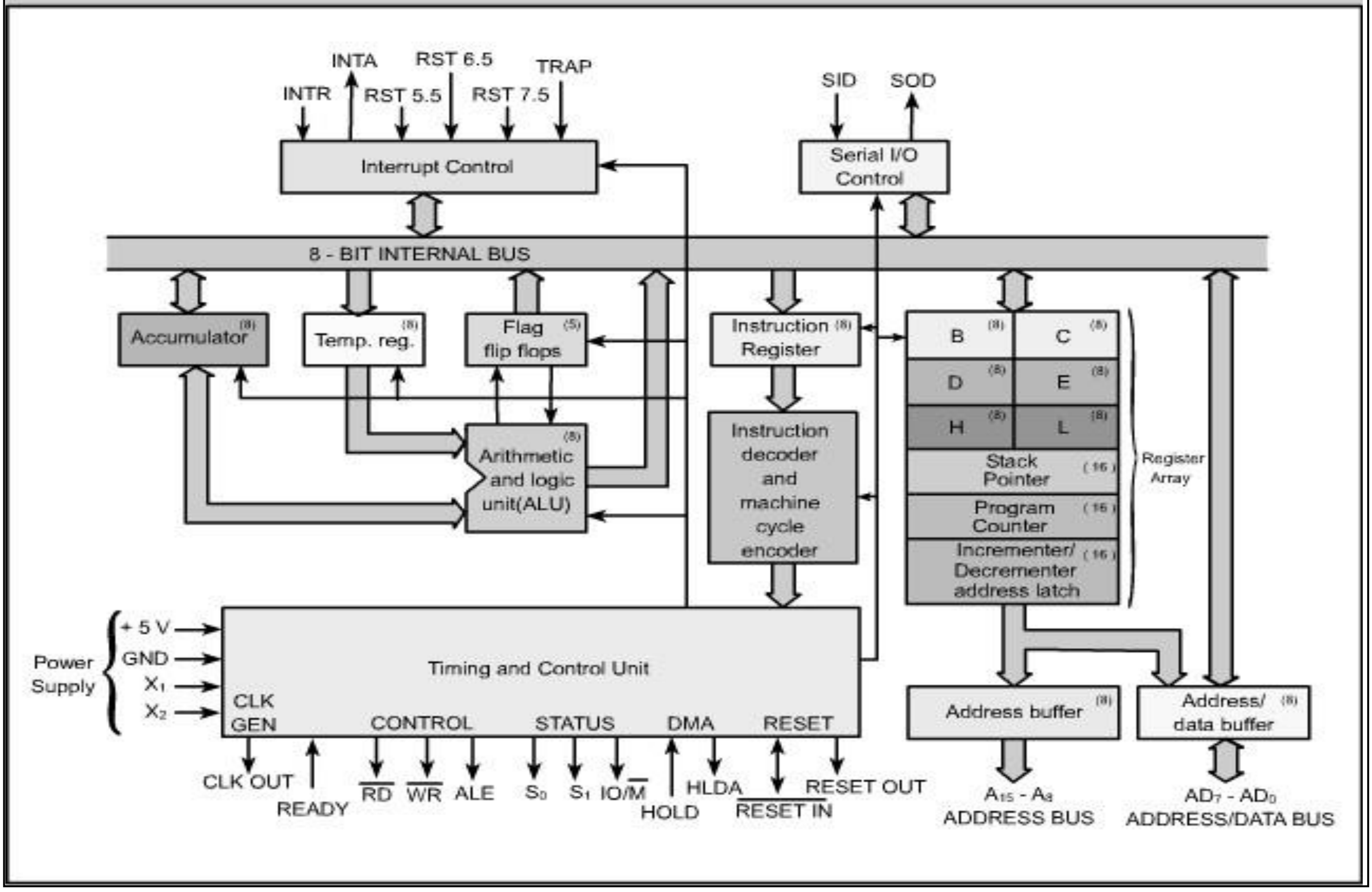
- Accumulator is an 8-bit register.
- The accumulator is also called as A-register.
- It holds one of the data to be processed by ALU.
- It stores the result of the operation.
- The accumulator is connected to the 8 – bit internal data bus.
- The two-state output of the accumulator drives the ALU.

TEMPORARY REGISTER:

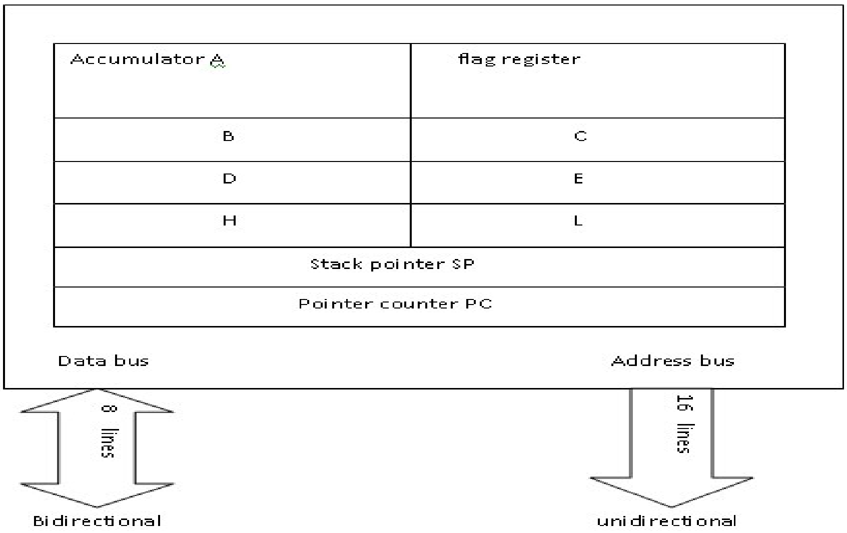
- The temporary register receives one of the data to be processed by ALU from external memory or general purpose registers.
- The other input for the ALU comes from the temporary register. This 8-bit register stores the operands of arithmetic – logic operations.

- For instance, during an ADD C the contents of the C register are copied in the temporary register during one T state and added to accumulator contents during another T state.

Block Diagram of 8085 Microprocessor



GENERAL PURPOSE REGISTERS:



- In INTEL 8085 microprocessor, there are six 8 –bit general – purpose registers.
- They are B, C, D, E, H And L.
- They may be used individually or combined as register pairs to perform some 16 – bit operations.
- The permitted combinations of register pairs are B –C, D-E and H-L. The H-L registers pair, which is normally used to form a 16-bit memory pointer.
- The register array (B, C, D, E, H and L) is like a small on-chip RAM with addressable memory locations.
- Control signals select the register for a read or write operation. This means that the CPU can wither load or read a register contents to this data bus.

STACK POINTER (SP):

- Stack Pointer is a 16-bit register used as a memory pointer.
- It maintains the address of the last byte entered into the stack.
- The stack pointer is decremented each time when data is loaded into the stack and is incremented when data is retrieved from the stack.

PROGRAM COUNTER (PC):

- This 16-bit register deals with sequencing the execution of instruction.
- The register is also a memory pointer.
- Memory locations have 16-bit address, and hence it is a 16-bit register.
- The microprocessor uses this register to sequence the execution of the instructions.
- The function of the program counter is to point to the address of the next instruction to be executed.
- At the end of the execution of an instruction, the program counter is incremented by 1, pointing to the next memory location where the next instruction is available.

INCREMENTER/DECREMENTER:

- It can add 1 or subtract 1 from the contents of the Stack Pointer or Program Counter.

ALU:

- The ALU carries out the arithmetic and logic operations on 8-bit words.
- The contents of the accumulator and the temporary register are the inputs to the ALU.
- It can perform arithmetic operations such as addition, subtraction and logical operations such as AND, OR and EX-OR. The ALU result is then stored back in the accumulator.

FLAGS:

Flag register is a group of five individual flip-flops. The content of the flag register will change (0 or 1) after the execution of arithmetic and logic operations.

1. The carry flag bit (CY) is set if a carry or borrow occurs during the arithmetic operation. The carry flag indicates that the operation resulted in overflow.

2. The parity flag bit (P) is set if the result has an even number of 1s, otherwise it will be reset (made zero).

3. The sign flag bit (S) is set if the bit D7 of the result is 1, otherwise it is reset. The sign bit indicates the sign of the number (Positive or Negative) and becomes useful in the signed binary number system.

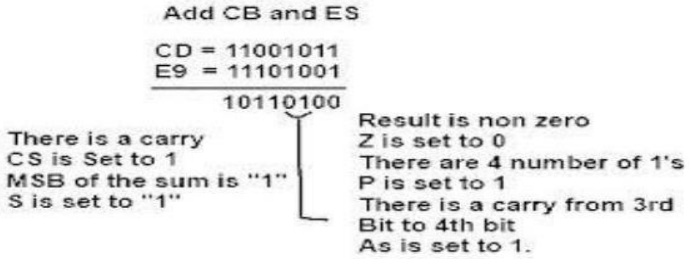
4. The zero flag bit (Z) is set if the result of the operation becomes 0. For all other values of the result the bit is reset.

5. The auxiliary carry flag bit (AC) is set, when a carry is generated at digit D3 position, and passed on to digit D4. The flag is used only internally for Binary Coded Decimal(BCD) operations.

The bit position of different flag register is shown in table.

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

Example:



Add CB and ES

CD = 11001011
E9 = 11101001

10110100

There is a carry
CS is Set to 1
MSB of the sum is "1"
S is set to "1"

Result is non zero
Z is set to 0
There are 4 number of 1's
P is set to 1
There is a carry from 3rd
Bit to 4th bit
As is set to 1.

INSTRUCTION REGISTER AND DECODER

- Instruction register and decoder is an 8-bit register.
- When an instruction is fetched from memory, it is loaded in the instruction register.
- The instruction decoder decodes the contents of the instruction register.
- It also determines the operation to be followed in executing the entire instruction and directs the timing and control unit accordingly.
- During the fetch cycle, the opcode of an instruction is stored in the instruction register. This opcode then drives the instruction decoder and machine- cycle encoder.

TIMING AND CONTROL

- The timing and control section of microprocessor includes an oscillator and controller-sequencer.
- The oscillator generates the two – phase clock signals (CLK and) that synchronize all registers. CLK
- The controller-sequencer also produces the control signals needed for internal and external control.

- The controller – sequencer is micro programmed; it has a ROM that stores all the micro routines needed for executing the instruction.
- After each instruction is fetched and stored in the instruction register, the opcode is decoded to get the starting address of the desired micro routine.
- As each microinstruction is read out of the control ROM, control signals are sent to the internal and external data buses.
- The effect is to move data between registers, to perform arithmetic and logic operations, to input or output data, etc.
- The control ROM is sometimes called the control store.

INTERRUPT CONTROL

- Sometimes it is necessary to interrupt the execution of the main program to answer a request from an I/O device.
- For instance, an I/O device may send an interrupt signal to the interrupt control unit to indicate that data is ready for input.
- The computer temporarily stops the execution of main program, inputs the data, and then returns to the main program.
- The interrupt concept is analogous to reading a book (main program), hearing the phone (interrupt), answering the phone (servicing the interrupt), then returning back to reading (main program).

SERIAL I/O CONTROL

Sometimes, I/O devices work with serial data rather than parallel. In this case, the serial data stream from an input device must convert to 8-bit parallel data before the computer can use it. Likewise, the 8-bit data out of a computer mus t be converted to serial form before a serial output device can use it.

The Serial Input Data enters 8085 through pin 5 (SID – Serial Input Data) and leaves through pin 4 (SOD – Serial Input Data) and leaves through pin 4 (SOD – Serial Output Data). Two new instructions known as SIM and RIM allows us to perform the serial- parallel conversion needed for serial I/O devices.

ADDRESS, DATA, AND CONTROL BUSES

Near the top of the Figure is an 8-bit internal data bus. This carries instructions and data between the CPU registers. The external buses are the ones we have to connect to otherchips like memory I/O and so forth. Near the bottom left of the Figure is the external control bus (, ALE). On the bottom right are the external address- data buses. WR RD,

The upper 8 address bits are on a separate bus always used for address bits; this upper section of the address bus is designated A15 - A8. The lower 8-bit are multiplexed. This means that the eight lower bus lines are used for address bits during some T states and for data bits during other T states. This is why the bus is labeled address-data bus, and designed as AD7 – AD0.

ADDRESS BUFFER AND ADDRESS DATA-BUFFER

At the bottom right in Figure are two buffer registers called the address buffer and the address – data buffer. The contents of the stack pointer or program counter can be loaded into the address buffer and address – data buffer. The put of these buffers then drives the external address bus and address –data bus. Memory and I/O chips are connected to these buses. In this way, the CPU can send the address of desired data to the memory or I/O chips.

The 8-bit internal data bus is also connected to the address- data buffer. The bi-directional arrow indicates a three-state connection that allows the address-data buffer to send or receive data from the 8-bit internal data bus.

Features Of 8085

1. 8085A is an 8-bit general – purpose microprocessor.

2. It is a 40 pin dual - in – line package single chip integrated circuit.

3. Only one +5v power supply is needed for its operation.

4. It can operate with a 3 MHZ single – phase clock.

5. The 8085A – 2 versions can operate at the maximum frequency of 5MHZ.

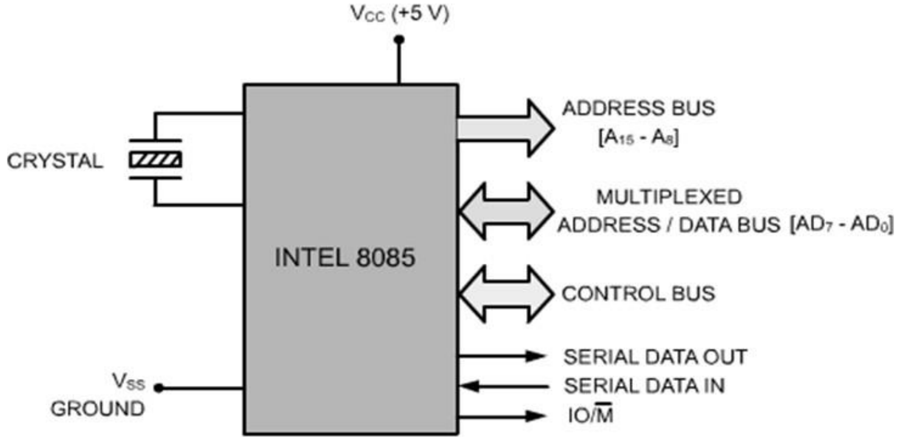
6. The width of the data bus is 8-bit. The width of the address bus is 16 –bit. Therefore maximum of 64 kilobytes of memory locations (i.e. 216=65,536=64KB) can be addressed directly by the 8085.

7. The multiplexing of address/data bus allows for extra control signals.

8. 8085 has one non- maskable (TRAP) and three maskable – vectored interrupts (RST 7.5, 6.5 & 5.5).

9. It provides Serial Input Data (SID) and Serial Output Data(SOD) lines for simple serial interface.

10. 8085 has an inbuilt clock oscillator circuit and requires externally only a crystal. The frequency of the crystal is internally divided by 2.



Bus Organization of INTEL 8085

4. PIN DIAGRAM AND PIN DESCRIPTION OF 8085

- 8085 is a 40 pin IC, DIP.
- 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.

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.

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

ADDRESS BUS

- A8 - A15 (output; 3-state)
- It carries the most significant 8 bits of the memory address or the 8 bits of the I/O address;

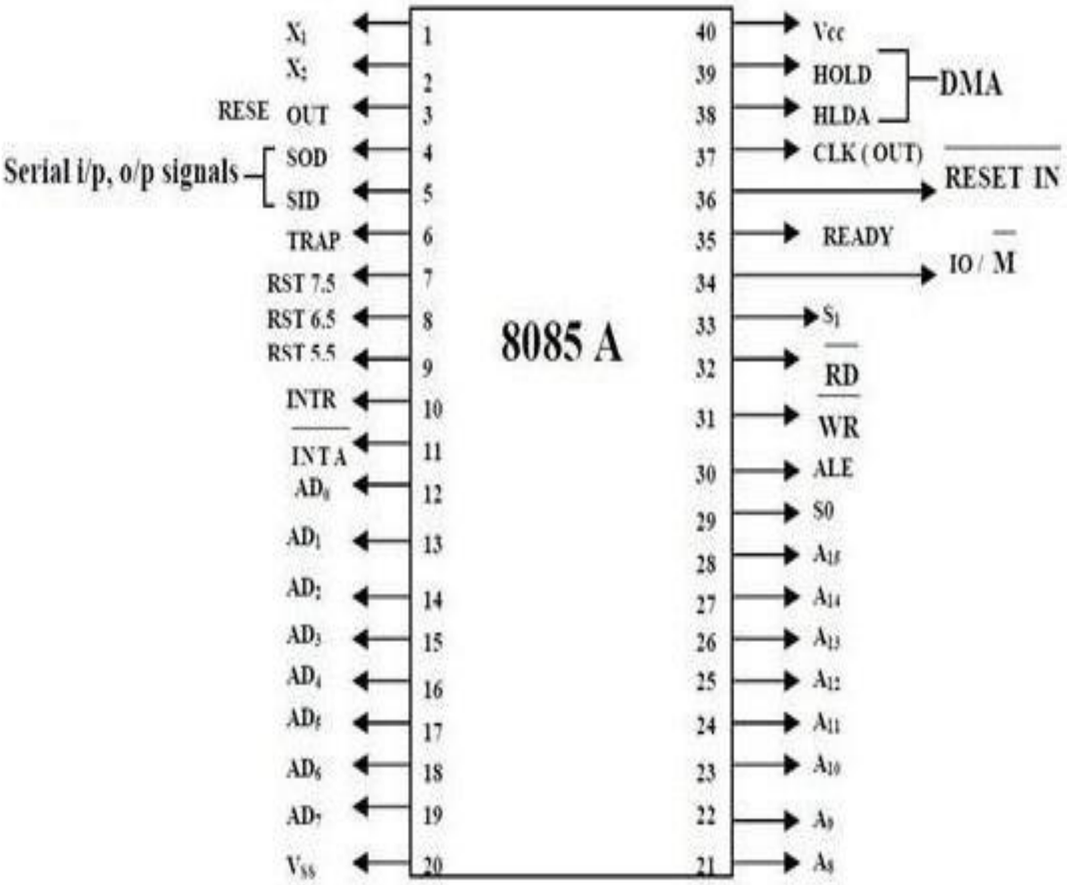


Fig - Pin Diagram of 8085

MULTIPLEXED ADDRESS / DATA BUS

- AD0 - AD7 (input/output; 3-state)
- 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.

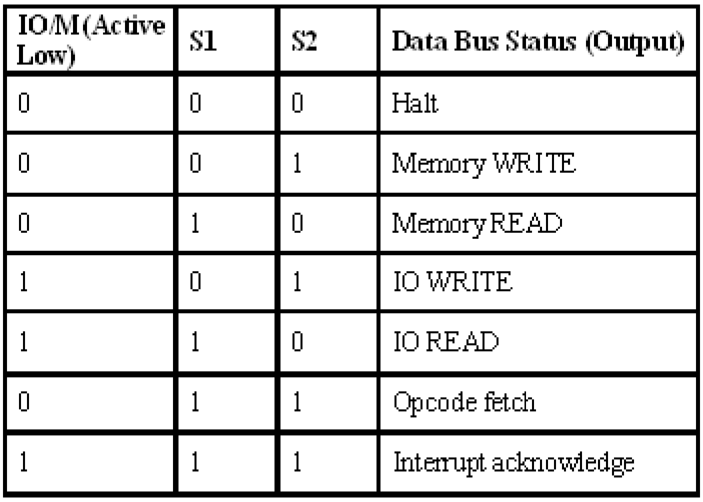
The CPU may read or write out data through these lines

CONTROL AND STATUS SIGNALS

- ALE (output) - Address Latch Enable.
- This signal helps to capture the lower order address presented on the multiplexed address / data bus.
- RD (output 3-state, active low) - Read memory or IO device.
- This indicates that the selected memory location or I/O device is to be read and that the data bus is ready for accepting data from the memory or I/O device.
- WR (output 3-state, active low) - Write memory or IO device.
- This indicates that the data on the data bus is to be written into the selected memory location or I/O device.
- IO/M (output) - Select memory or an IO device.
- This status signal indicates that the read / write operation relates to whether the memory or I/O device.
- It goes high to indicate an I/O operation.
- It goes low for memory operations.

STATUS SIGNALS

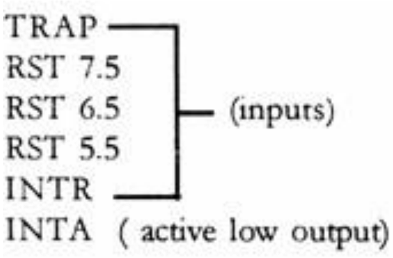
- It is used to know the type of current operation of the microprocessor.



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

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,



- On receipt of an interrupt, the microprocessor acknowledges the interrupt by the active low INTA (Interrupt Acknowledge) signal.

Reset In (input, active low)

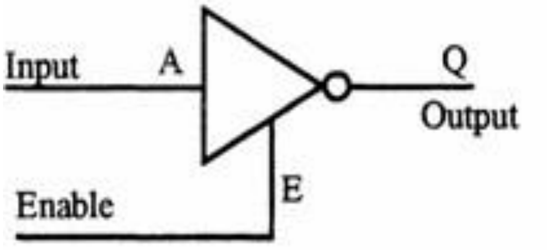
- This signal is used to reset the microprocessor.
- The program counter inside the microprocessor is set to zero.
- The buses are tri-stated.

Reset Out (Output)

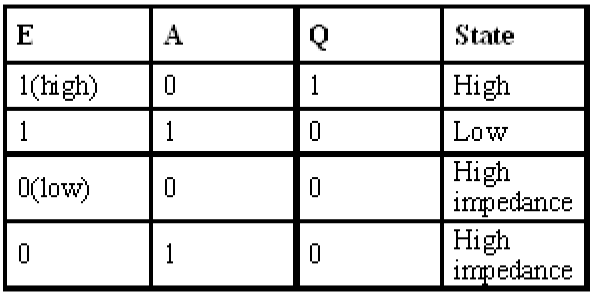
- It indicates CPU is being reset.
- Used to reset all the connected devices when the microprocessor is reset.

5. DIRECT MEMORY ACCESS (DMA)

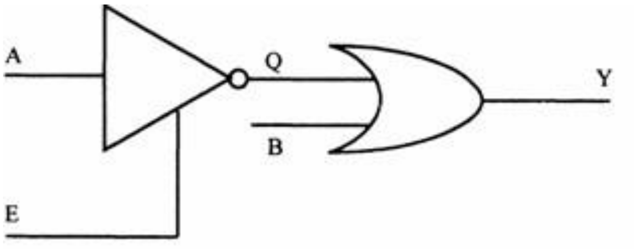
Tri state devices:



- 3 output states are high & low states and additionally a high impedance state.
- When enable E is high the gate is enabled and the output Q can be 1 or 0 (if A is 0, Q is 1, otherwise Q is 0). However, when E is low the gate is disabled and the output Q enters into a high impedance state.



E	A	Q	State
1(high)	0	1	High
1	1	0	Low
0(low)	0	0	High impedance
0	1	0	High impedance



- For both high and low states, the output Q draws a current from the input of the OR gate.
- When E is low, Q enters a high impedance state; high impedance means it is electrically isolated from the OR gate's input, though it is physically connected. Therefore, it does not draw any current from the OR gate's input.
- When 2 or more devices are connected to a common bus, to prevent the devices from interfering with each other, the tri state gates are used to disconnect all devices except the one that is communicating at a given instant.
- The CPU controls the data transfer operation between memory and I/O device. Direct Memory Access operation is used for large volume data transfer between memory and an I/O device directly.
- The CPU is disabled by tri-stating its buses and the transfer is effected directly by external control circuits.
- HOLD signal is generated by the DMA controller circuit. On receipt of this signal, the microprocessor acknowledges the request by sending out HLDA signal and leaves out the control of the buses. After the HLDA signal the DMA controller starts the direct transfer of data.

READY (input)

- 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.

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.

6. INSTRUCTION WORD SIZE

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

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

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

One-Byte Instructions

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

For example:

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

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

MOV rd, rs

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

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

Example: MOV A,B

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

ADD r

A <-- A + r

Two-Byte Instructions

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

For example

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

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

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

The instruction would require two memory locations to store in memory.

MVI r,data
r <-- data
Example: MVI A,30H coded as 3EH 30H as two contiguous bytes. This is an example of immediate addressing.
ADI data
A <-- A + data
OUT port

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

Three-Byte Instructions
In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.
opcode + data byte + data byte

For example:

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

This instruction would require three memory locations to store in memory.

Three byte instructions - opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.
rp <-- data16

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

LDA addr

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

7. ADDRESSING MODES OF 8085

- Every instruction of a program has to operate on a data.
- The method of specifying the data to be operated by the instruction is called Addressing**.**
- The various ways of specifying the operand in the operand field of an instruction are called the addressing modes.

The various addressing modes are:

1. Direct addressing mode
2. Immediate addressing mode
3. Register direct addressing mode
4. Register indirect addressing mode
5. Implicit addressing mode
6. Stack addressing mode
7. Indirect addressing mode
8. Indexed addressing mode
9. Relative addressing mode

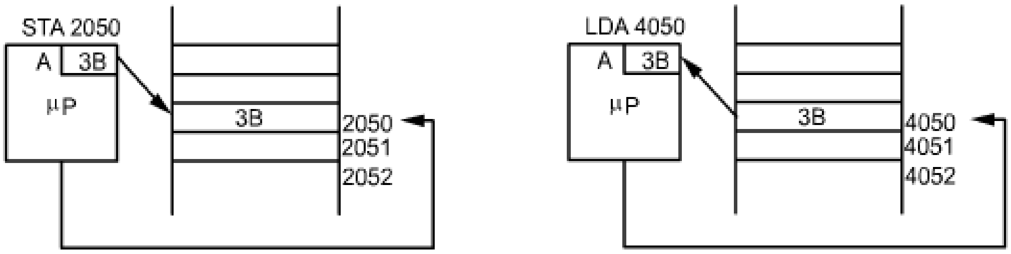
1) DIRECT ADDRESSING MODE:

In direct addressing mode, the address of the operand is directly specified in the instruction. Except IN and OUT instructions all other direct addressing modes are 3-bytes long.

Examples:

a) **STA 16-bit Address** – The contents of the accumulator are copied to a memory location whose address is specified.

b) **LDA 16-bit Address** – The contents of the memory location whose address is specified in byte 2 and byte 3 of the instructions are copied to the Accumulator. It is 3-byte instruction.



2) **IMMEDIATE ADDRESSING MODE:**

In immediate addressing mode, the actual data (8-bit or 16-bit) is part of the instruction. The length of the instruction may be two or three bytes. The first byte specifies opcode. If it is a two byte instruction, the second byte specifies an 8-bit data. If it is a three byte instruction, the second and third bytes specify 16-bit data.

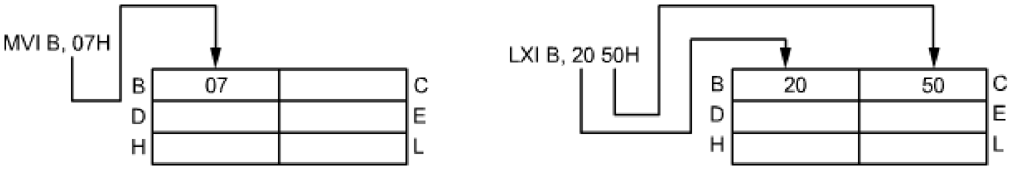
Examples:

a) **MVI R, 8-bit Data**

It is a 2 byte instruction. Byte 2 (8-bit data) of the instruction is immediately moved to register R (R may be A, B, C, D, E, H, L).

b) **LXI R_P, 16-bit Data**

It is a 3-byte instruction. Byte 2 of the instruction is immediately moved into the low-order register of the register pair R_p and byte 3 of the instruction immediately moved into the high-order register of the register pair.



3) **REGISTER DIRECT ADDRESSING MODE:**

In some instructions, general-purpose registers are specified as the address of the operands. Such instructions are called as register direct addressing mode instructions. These instructions are one byte long. Since the microprocessor need not fetch data from memory, this mode of addressing is faster than direct addressing mode. This mode of addressing is called as register addressing mode.

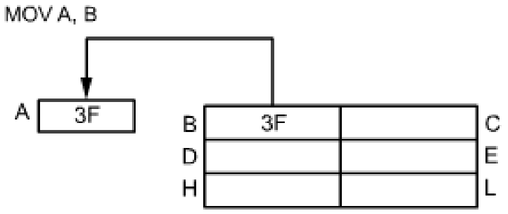
Examples:

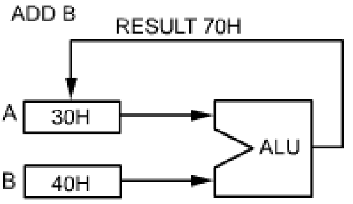
a) **MOV R_d, R_s**

The contents of the source register (R_s) are moved to the destination register (R_d). It is a single byte instruction (R_d and R_s are general purpose registers).

b) **ADD R**

It is a single byte instruction. The contents of the register pair 'R_p'. (R_p may be BC, DE, HL) are incremented by 1.





4) **REGISTER INDIRECT ADDRESSING MODE:**

In register indirect addressing mode the contents of the specified register pair is used as the address of the operand. The register pair contains the 16-bit address of the memory location where the actual operand is stored. Usually the memory pointer (HL-register pair) contains the address of the memory location.

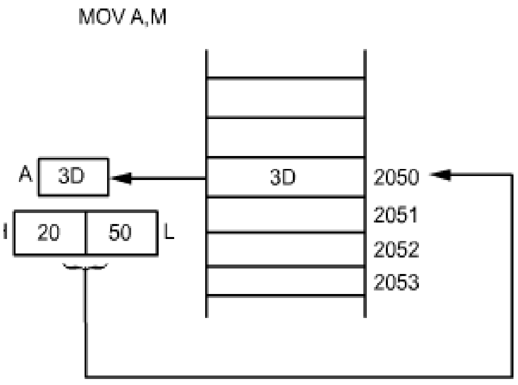
Examples

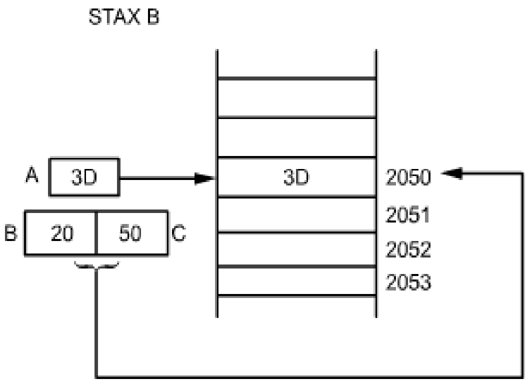
a) **MOV R_d ,M**

It is a single byte instruction. The contents of the memory location whose address is specified by the contents of the HL-register pair are moved to the destination register R_d (R_d may be any one of the general purpose register).

b) **ADD M**

It is a single byte instruction. The contents of the memory location whose address is specifies by the contents of the memory pointer (HL – register pair) are added with the contents of the accumulator and the result is placed in the accumulator.





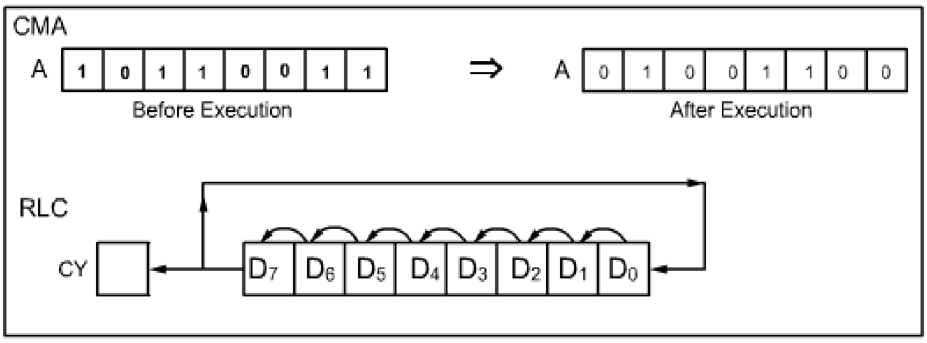
5) IMPLICIT ADDRESSING MODE

In implicit addressing mode, the address of a register (Accumulator in the case of 8085 containing the operand data) is implicitly stated in the opcode itself. In this addressing mode, the instructions are one byte long since the operand is in the accumulator.

Examples

a) **CMA** – It is a single byte instruction. The content of the accumulator is completed. There the operand (data) which is nothing but the contents of accumulator is specified within the instruction.

b) **RLC** – It is a single byte instruction. The contents of the accumulator are rotated left by one position.

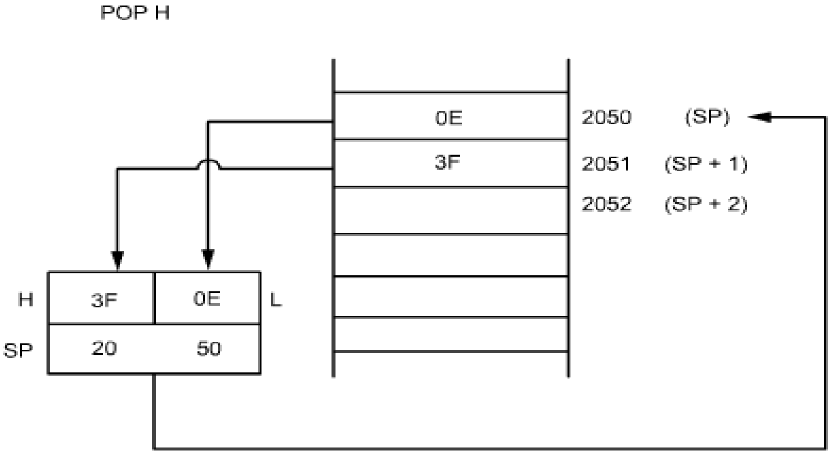


6) STACK ADDRESSING MODE

In this addressing the content of the stack pointer (16-bit register) is the address of the operand (data). It is similar to register addressing mode. Here the stack pointer content is the address of the stack memory.

Examples

POP R_p – It is a single byte instruction. The contents of the memory location pointed by the stack pointer are copied to the low-order-register. The stack pointer is incremented by 1 and contents of that memory location are copied to the high order register.

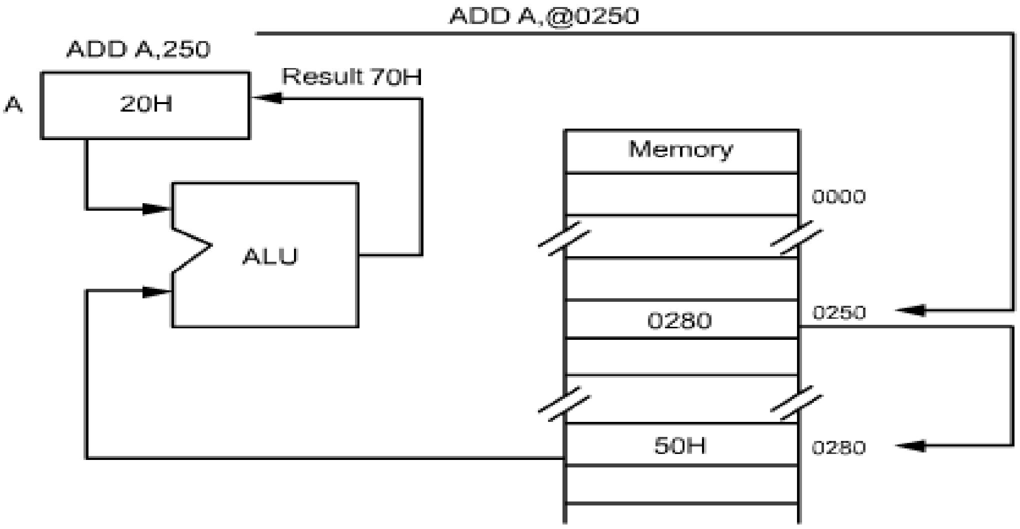


7. INDIRECT ADDRESSING MODE:

In Indirect addressing mode, the instruction points to an Address where the exact address of the operand is present.

Examples

ADD A, 2050 – In this instruction the contents of the accumulator are added with the content of memory location, whose Address is specified by the operand of the instruction.



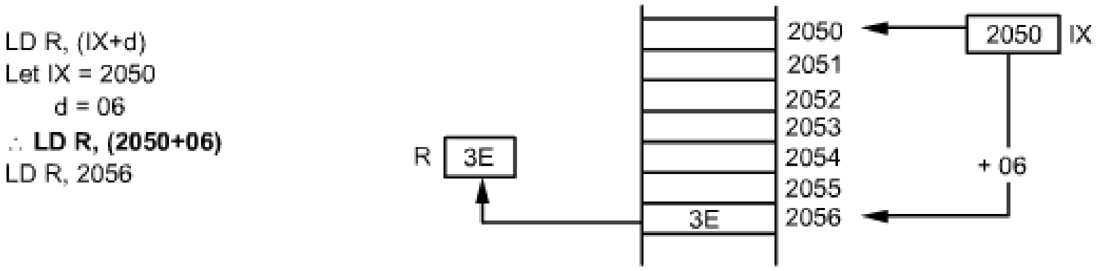
8. INDEXED ADDRESSING MODE

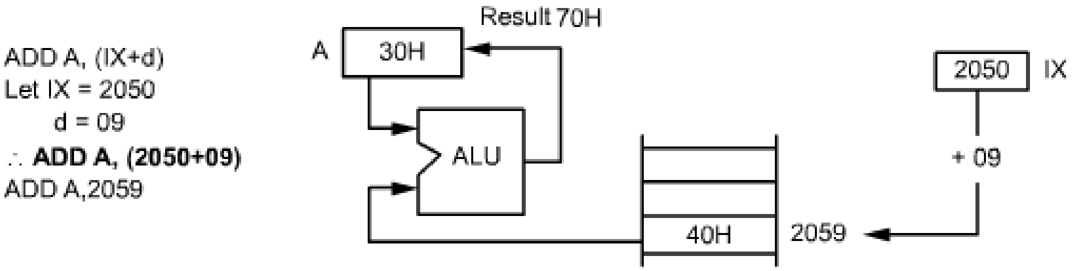
In this Addressing mode the Address of the operand is specified in relation to the contents of a 16-bit register called "Index Register". A displacement, which is to be added with the contents of the index, register is also given in the instruction itself. This type of Addressing mode is used in Z-80 microprocessor.

Examples

a) **LD R, (IX + d) –** This instruction will move the contents of the memory location specified by (IX + d) into specified register.

b) **ADD A, (IX +d) –** This instruction will add the contents of the memory location specified by (IX + d) with the contents of the accumulator.





8. RELATIVE ADDRESSING MODE

In this type of addressing mode, in order to find the effective address the address specified in the instruction (referred as offset) is added with the contents of PC. This mode of addressing is used in Motorola 6800 and Z – 80 microprocessor.

Offset is the displacement of the branching location from the Branch instruction.

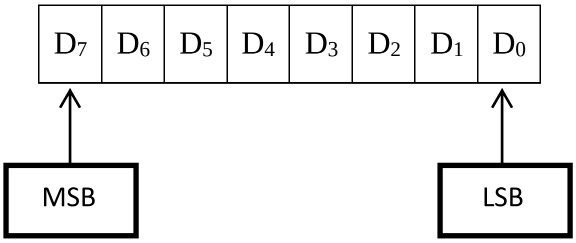
Offset = address of the loop – address of next instruction to the branch instruction

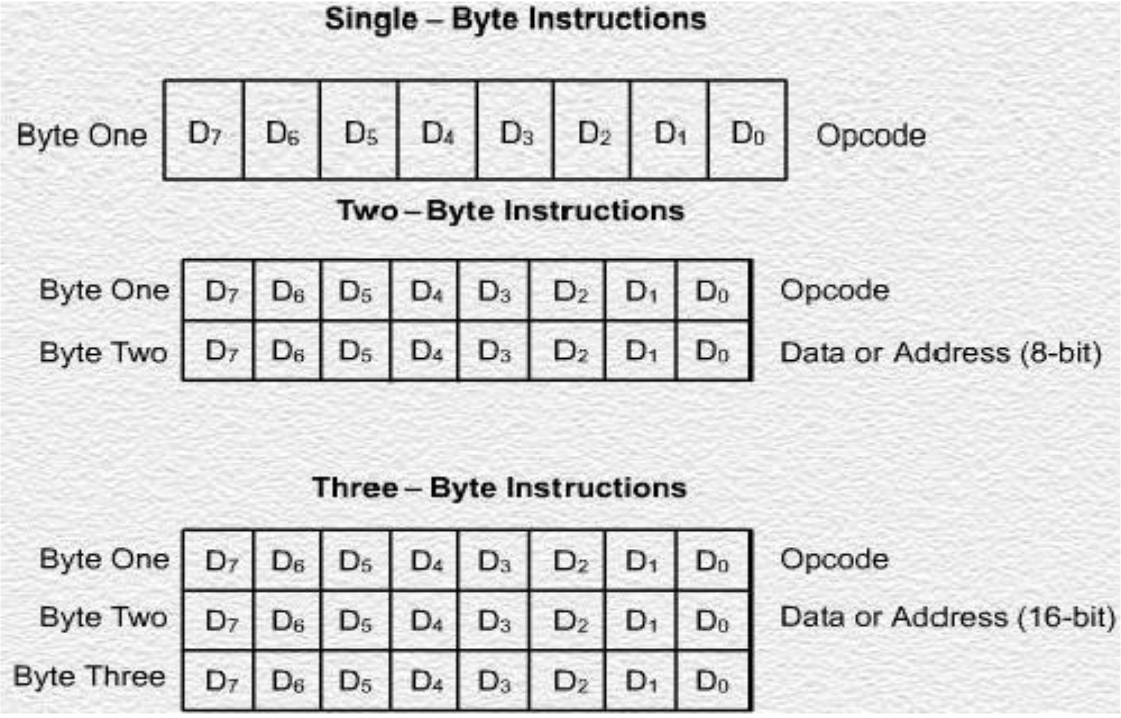
8. INSTRUCTION SET OF 8085

The 8085A implements a group of instructions that move data between registers, between a register and memory, and between registers and an I/O Port. It also has arithmetic and logic instructions, conditional branch instructions. The CPU recognizes these instructions only when they are coded in binary form.

Instruction and Data Formats

Data in the 8085A is stored in the form of 8-bit values.





Single – Byte Instructions

Byte One	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Opcode

Two – Byte Instructions

Byte One	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Opcode
Byte Two	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Data or Address (8-bit)

Three – Byte Instructions

Byte One	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Opcode
Byte Two	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Data or Address (16-bit)
Byte Three	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	

The complete 8085 instruction set is described, grouped under five different functional headings, as follows

1. DATA TRANSFER INSTRUCTIONS

It includes the instruction that moves (copies) data between memory location and register. In all data transfer operations the content of source register / memory is not altered. Hence the data transfer is copying instruction.

Copy from source to destination

Opcode	Operand	Description
MOV	**R_d, R_s**	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. **Example:** MOV B, C or MOV B, M
	M, R_s	
	R_d, M	

Load accumulator

LDA	**16-bit address**	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. **Example:** LDA 2034H

Store accumulator direct

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

Exchange H and L with D and E

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

Push register pair onto stack

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

Pop off stack to register pair

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

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

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

2. ARITHMETIC INSTRUCTIONS

It includes the instruction which performs addition, subtraction, increment, decrement operations. The flag conditions are altered after execution of an instruction in this group.

Opcode Operand Description

Add register or memory to accumulator

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

Subtract immediate from accumulator

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

Decimal adjust accumulator

DAA	none	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.
		If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.
		If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds6 to the high-order four bits.
		Example: DAA

Increment register pair by 1

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

3. BRANCHING INSTRUCTIONS

The instructions which performs the logical operations like AND, OR, EX-OR, complement, compare and rotate instructions are grouped under this heading. The flag conditions are altered after the execution of an instruction in this group.

Opcode Operand Description

Jump unconditionally

JMP	**R**	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. **Example:** JMP 2034H or JMP XYZ

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

Unconditional subroutine call

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

Opcode	**Description**	**Flag Status**
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1

CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

4. LOGICAL INSTRUCTIONS

The instructions that are used to transfer the program control from one memory location to another memory location are grouped under this heading.

Opcode Operand Description

Compare register or memory with accumulator

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

Logical AND register or memory with accumulator

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

Exclusi0ve OR register or memory with accumulator

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

5. MACHINE CONTROL INSTRUCTIONS

It includes the instructions related to interrupts and the instructions used to halt program execution.

Opcode **Operand** **Description**

Halt and enter wait state

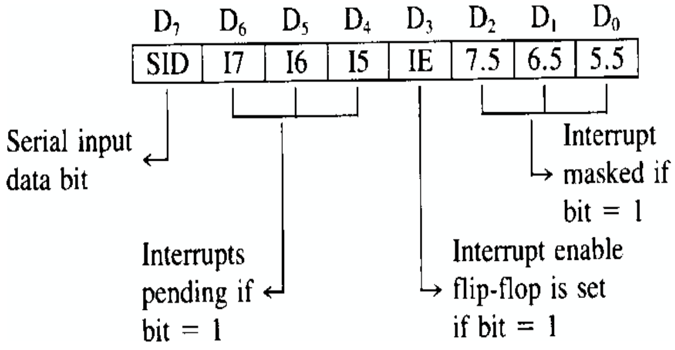
Opcode	Operand	Description
HLT	**none**	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. **Example**: HLT

Enable interrupts

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

Read interrupt mask

Opcode	Operand	Description
RIM	**none**	This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations. **Example:** RIM


D7
D6
D5
D4
D3
D2
D1
D0
SID
I7
I6
I5
IE
7.5
6.5
5.5
Serial input data bit
Interrupt masked if bit = 1
Interrupts pending if bit = 1
Interrupt enable flip-flop is set if bit = 1

9. ASSEMBLY LANGUAGE PROGRAMMING:

Assembler:

It is software that converts assembly language program code to machine language code.

Assembly language instructions have the format:

ADDRESS	LABEL	MNEMONICS	OPERAND	COMMENTS

Address

- Specify the address of an instruction.

Instruction Label (optional)

- Marks the address of an instruction, must have a colon :
- Used to transfer program execution to a labeled instruction

Mnemonic

- Identifies the operation (e.g. MOV, ADD, SUB, JMP, CALL)

Operands

- Specify the data required by the operation
- Executable instructions can have zero to three operands
- Operands can be registers, memory variables, or constants

No operands

```
stc ; set carry flag
```

One operand

```
inc eax ; increment register eax
call Clrscr ; call procedure Clrscr
jmp L1 ; jump to instruction with label L1
```

Two operands

```
add ebx, ecx ; register ebx = ebx + ecx
sub var1, 25 ; memory variable var1 = var1 – 25
```

Three operands

```
imul eax,ebx,5          ; register eax = ebx * 5
```

Identifiers

- Identifier is a programmer chosen name
- Identifies variable, constant, procedure, code label
- May contain between 1 and 247 characters
- Not case sensitive

- First character must be a letter (A..Z, a..z), underscore(_), @, ?, or $.
- Subsequent characters may also be digits.
- Cannot be same as assembler reserved word.

Comments

- Comments are very important!
  - Explain the program's purpose
  - When it was written, revised, and by whom
  - Explain data used in the program
  - Explain instruction sequences and algorithms used
  - Application-specific explanations
- Single-line comments
  - Begin with a semicolon ; and terminate at end of line
- Multi-line comments
  - Begin with COMMENT directive and a chosen character
  - End with the same chosen character

Arithmetic Operation:

Program in C:

```
void main()
{
int a=5,b=6,c;    // define the data type for  variable a,b& cand intialize the value for variable a & b respectively.
c=a+b;               // add the varible a & b and place the result in C
printf("%d",c); // display the value of variable c
}
```

Program in Microprocessor (immediate addressing):

```
MVI A, 05      // assign the value 05 to accumulator
MVI B, 06      // assign the value 06 to B register
```

```
ADD B       // add the content in B register with accumulator and place the result in
accumulator
STA 4200    // store the result to the memory location 4200. HLT // stop the program
```

Program in C:

```
Void main()
{
Int a, b, c; \\define the data type for variable
Printf("enter the value for a & b");
Scanf("%d%d"&a,&b); // get the number and place it to the memory respectively.
C=a+b; // add the variable a & b and place the result into c.
Printf("reult is %d"c); // display the value of c;
}
```

Program in Microprocessor (direct addressing):

```
LDA 4500    // load the content of memory location 4500 to accumulator. MOV B,A // move
the content from accumulator to B register.
LDA 4501    // load the content of memory location 4501 to accumulator
ADD B       // add the content in B register with accumulator and place the result in
accumulator
STA 4502    // store the result to the memory location 4200.
HLT         // stop the program.
```

Logic operation:

```
//C program for Arranging 5 Numbers in Ascending Order
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5],i,j,t; // define the variables.
clrscr();
printf("Enter 5 nos.\n\n");
for (i=0;i<5;i++) // perform loop operation.
scanf("%d",&a[i]); // get the given number to store it in the respective address .
for (i=0;i<5;i++) // perform loop operation.
{ for(j=i+1;j<5;j++) // perform inner loop operation.
{ if(a[i]>a[j]) // compare the numbers a[i] & a[j] respectively
{ t=a[i]; // use the temporary variable for making swap operation.
a[i]=a[j];
a[j]=t;
} } }
printf("Ascending Order is:");
for(j=0;j<5;j++) // perform the loop operation for display the number in ascending order.
printf("\n%d",a[j]);
getch();
}
```

ASCENDING ORDER

Aim:

To write a program to sort given 'n' numbers in ascending order

Algorithm:

1. Load the count value from memory to A-reg and save it in B-reg.

2. Decrement B-reg (B is a count for (N-1) repetitions)

3. Set HL pair as data address pointer.

4. Set C-reg as counter for(N-1) comparisons.

5. Load a data of the array in accumulator using the data address pointer.

6. Increment the HL pair(data address pointer).

7. Compare the data pointed by HL with accumulator.

8. If Carry flag is set(if the content of accumulator is smaller than memory)then goto step10,otherwise go to next step.

9. Exchange the content of memory pointed by HL and the accumulator.

10. Decrement C-register. If zero flag is reset go to step 6 otherwise go to next step.

11. Decrement B-register. If zero flag is reset go to step 3 otherwise go to next step.

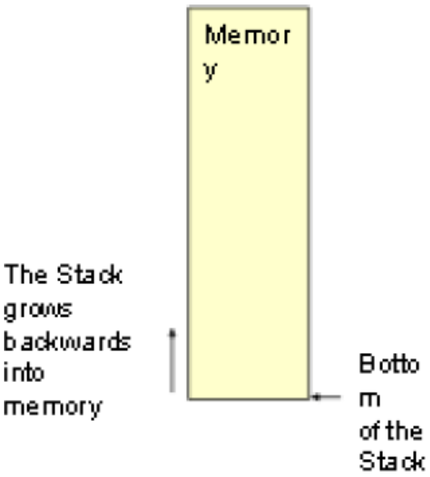
12. Stop.

Program:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		LDA 4200	3A,00,42	Load the count value in A-reg.
4103		MOV B, A	47	Set counter for (N-1) repetition of
				(N-1) comparisons.
4104		DCR B	05	
4105	LOOP2:	LXI H, 4200H	21,00,42	Set pointer for array
4108		MOV C, M	4E	Set counter for of (N-1) comparisons.
4109		DCR C	0D	Decrement the content of C-reg.
410A		INX H	23	Increment the HL-reg pair.
410B	LOOP1:	MOV A,M	7E	Move the content of memory pointer to accumulator.
410C		INX H	23	Increment the HL-reg pair.
410D		CMP M	BE	Compare the accumulator with memory.
410E		JC AHEAD	DA,16,41	If the content of A-reg is less than memory address of HL-reg pair then go to AHEAD.
4111		MOV D, M	56	If the content of A-reg is greater than the memory address of HL- reg pair, then exchange the content of memory pointed by HL and previous location.
4112		MOV M, A	77	
4113		DCX H	2B	
4114		MOV M,D	72	
4115		INX H	23	Increment the HL-reg pair.
4116	AHEAD:	DCR C	0D	Decrement the content of C-reg.
4117		JNZ LOOP1	C2,0B,41	Repeat comparisons until C count is zero.
411A		DCR B	05	
411B		JNZ LOOP2	C2,05,41	Repeat until B count is zero.
411E		HLT	76	Stop the program

10.THE STACK

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO structure.
  -Last In First Out.
- The stack normally grows backwards into memory.
  -In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.



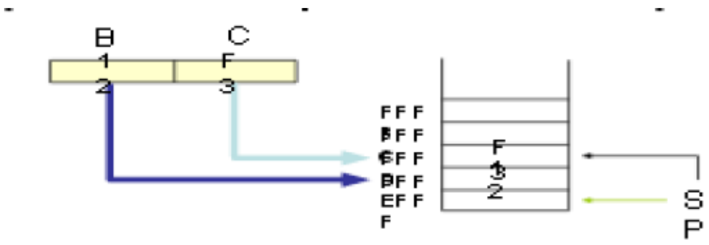
- Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.
- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.
  LXI SP, FFFFH
- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

Saving Information on the Stack

- Information is saved on the stack by PUSHing it on. It is retrieved from the stack by POPing it off.
- The 8085 provides two instructions:
  - PUSH and POP for storing information on the stack and retrieving it back.
  - Both PUSH and POP work with register pairs ONLY.

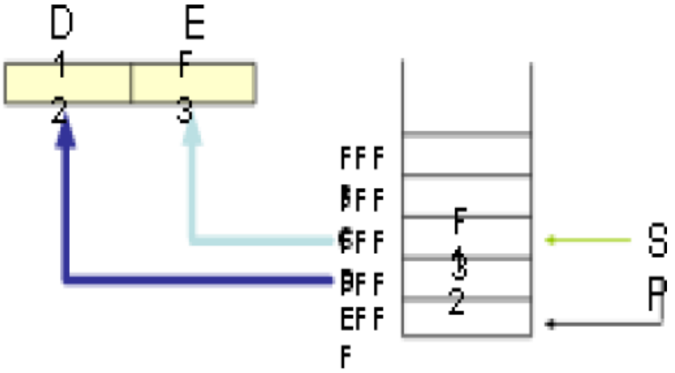
The PUSH Instruction

- PUSH B
  - Decrement SP
  - Copy the contents of register B to the memory location pointed to by SP
  - Decrement SP
  - Copy the contents of register C to the memory location pointed to by SP



The POP Instruction

- POP D
  - Copy the contents of the memory location pointed to by the SP to register E
  - Increment SP
  - Copy the contents of the memory location pointed to by the SP to register D
  - Increment SP



Operation of the Stack

- During pushing, the stack operates in a "decrement then store" style.
  - The stack pointer is decremented first, then the information is placed on the stack.
- During poping, the stack operates in a "use then increment" style.
  - The information is retrieved from the top of the the stack and then the pointer is incremented.
- The SP pointer always points to "the top of the stack".

LIFO

- The order of PUSHs and POPs must be opposite of each other in order to retrieve information back into its original location.

```
PUSH B
PUSH D
......
POP D
POP B
```

The PSW Register Pair

- The 8085 recognizes one additional register pair called the PSW (Program Status Word).
  - This register pair is made up of the Accumulator and the Flags registers.

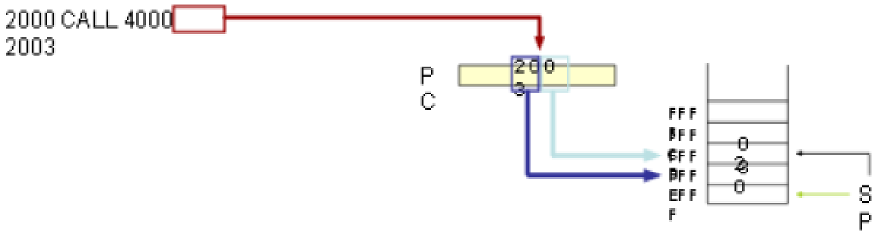
- It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.
    - The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

11. SUBROUTINES

- A subroutine is a group of instructions that will be used repeatedly in different locations of the program.
    - Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.
- In Assembly language, a subroutine can exist any where in the code.
    - However, it is customary to place subroutines separately from the main program.
- The 8085 has two instructions for dealing with subroutines.
    - The CALL instruction is used to redirect program execution to the subroutine.
    - The RTE instruction is used to return the execution to the calling routine.

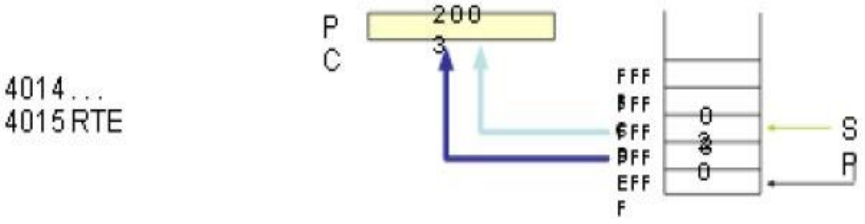
The CALL Instruction

- CALL 4000H
    - Push the address of the instruction immediately following the CALL onto the stack
    - Load the program counter with the 16-bit address supplied with the CALL instruction.



The RTE Instruction

- RTE
    - Retrieve the return address from the top of the stack
    - Load the program counter with the return address.



- The CALL instruction places the return address at the two memory locations immediately before where the Stack Pointer is pointing.
  - You must set the SP correctly BEFORE using the CALL instruction.
- The RTE instruction takes the contents of the two memory locations at the top of the stack and uses these as the return address.
  - Do not modify the stack pointer in a subroutine. You will loose the return address.

Passing Data to a Subroutine

- In Assembly Language data is passed to a subroutine through registers.
  - The data is stored in one of the registers by the calling program and the subroutine uses the value from the register.
- The other possibility is to use agreed upon memory locations.
  - The calling program stores the data in the memory location and the subroutine retrieves the data from the location and uses it.

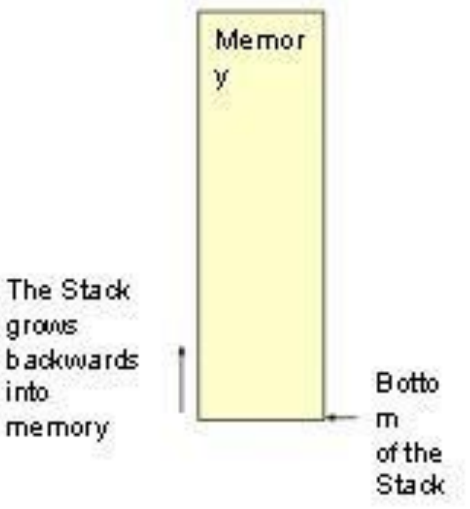
Call by Reference and Call by Value

- If the subroutine performs operations on the contents of the registers, then these modifications will be transferred back to the calling program upon returning from a subroutine.

  - Call by reference.
- If this is not desired, the subroutine should PUSH all the registers it needs on the stack on entry and POP them on return.
  - The original values are restored before execution returns to the calling program.

Cautions with PUSH and POP

- PUSH and POP should be used in opposite order.



ere are PUSH's.

l pick up the wrong information from the top of the

l.

POP inside a loop.

, and conditional RTE instructions.

h conditional JUMP instructions can be used.

Carry flag is set.

f Carry flag is not set.

utine if Carry flag is set.

routine if Carry flag is not set Etc.

12. TIMING DIAGRAM

OPCODE FETCH MACHINE CYCLE OF 8085:

- Each instruction of the processor has one byte opcode.
- The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is 4T.
- In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

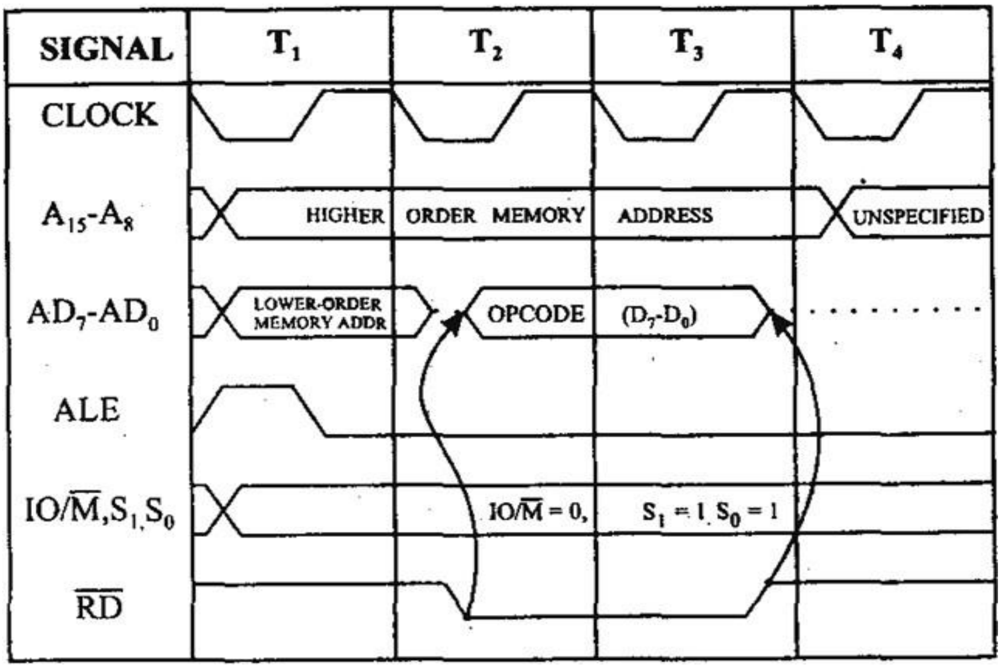


Fig - Timing Diagram for Opcode Fetch Machine Cycle

MEMORY READ MACHINE CYCLE OF 8085:

- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.
- The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.

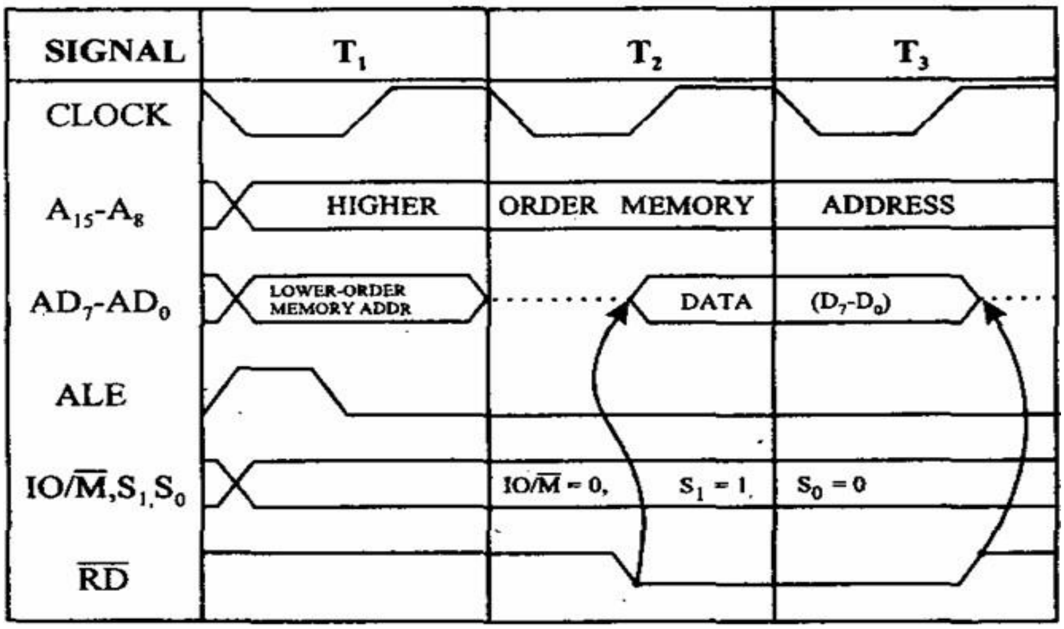


Fig - Timing Diagram for Memory Read Machine Cycle

I/O WRITE CYCLE OF 8085:

- The I/O write machine cycle is executed by the processor to write a data byte in the I/O port or to a peripheral, which is I/O, mapped in the system.
- The processor takes 3T states to execute this machine cycle.

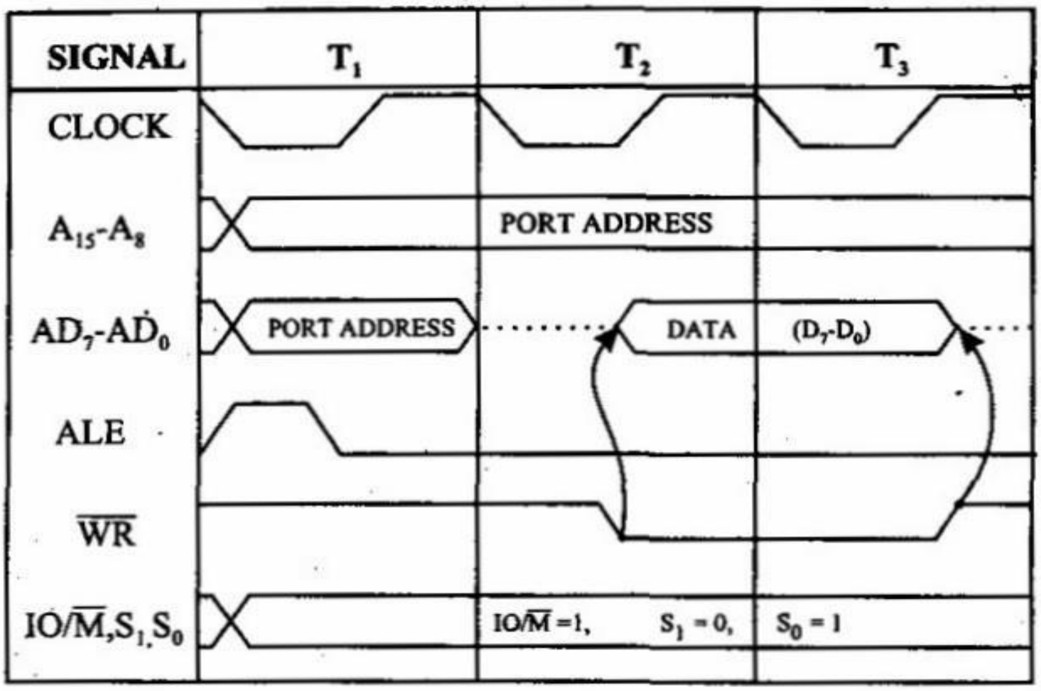


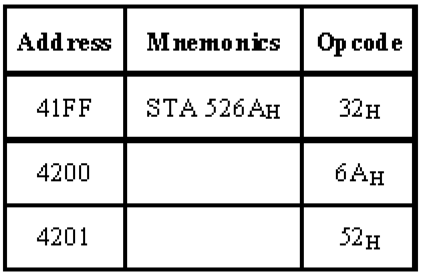
Fig - Timing Diagram for I/O Write Machine Cycle

TIMING DIAGRAM OF 8085 INSTRUCTIONS

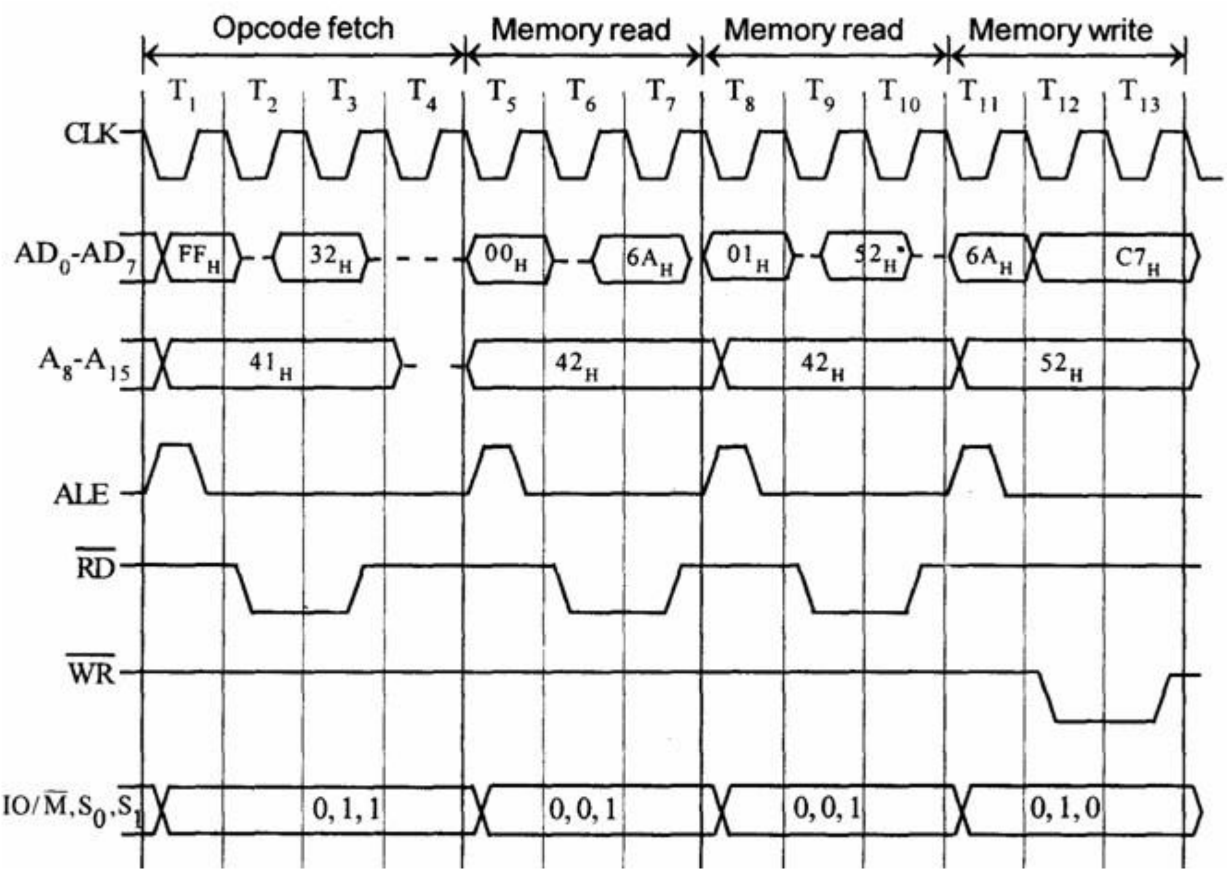
- The 8085 instructions consist of one to five machine cycles.
- Actually the execution of an instruction is the execution of the machine cycles of that instruction in the predefined order.
- The timing diagram of an instruction ate obtained by drawing the timing diagrams of the machine cycles of that instruction, one by one in the order of execution.

Timing Diagram for STA 526AH

- STA means Store Accumulator -The contents of the accumulator is stored in the specified address (526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see fig). - OF machine cycle
- Then the lower order memory address is read(6A). - Memory Read Machine Cycle
- Read the higher order memory address (52).- Memory Read Machine Cycle
- The combinations of both the addresses are considered and the content from accumulator is written in 526A. - Memory Write Machine Cycle
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

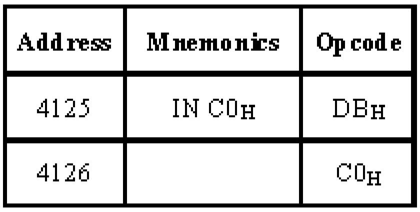


Address	Mnemonics	Opcode
41FF	STA 526A_H	32_H
4200		$6A_H$
4201		52_H

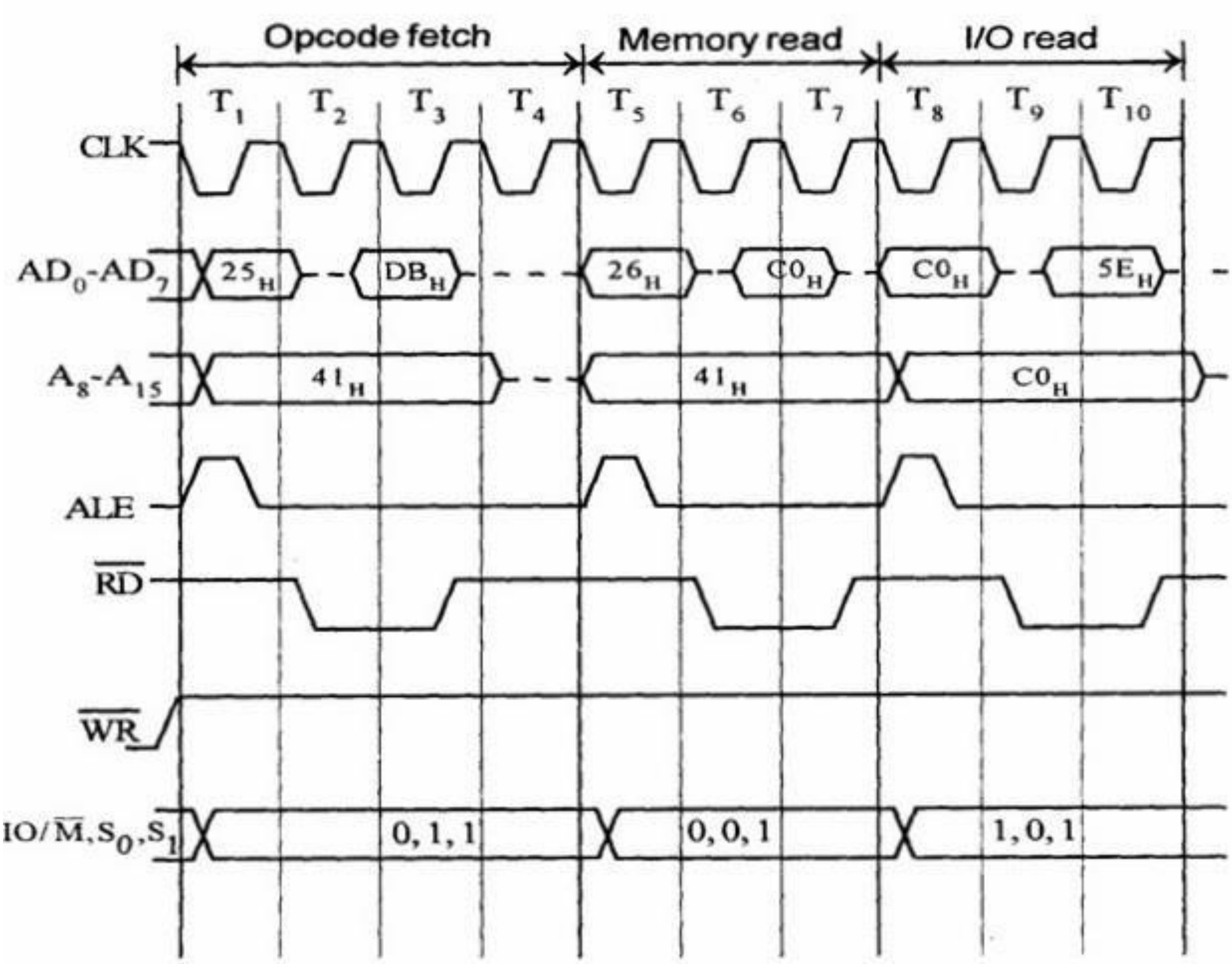


Timing Diagram for IN C0H

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

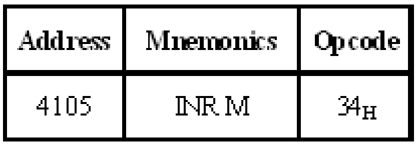


Address	Mnemonics	Opcode
4125	IN C0$_H$	DB$_H$
4126		C0$_H$

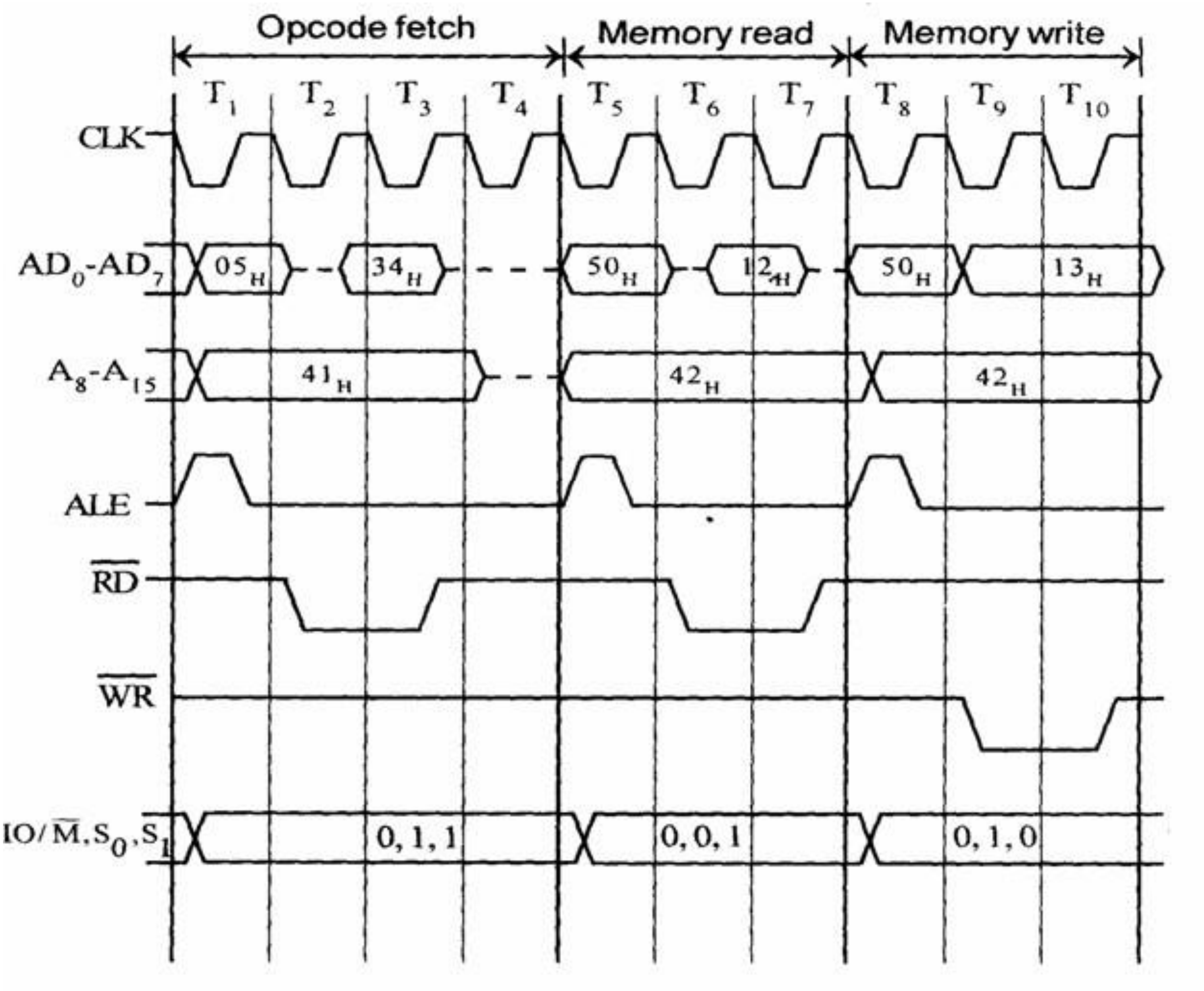


Timing diagram for INR M

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

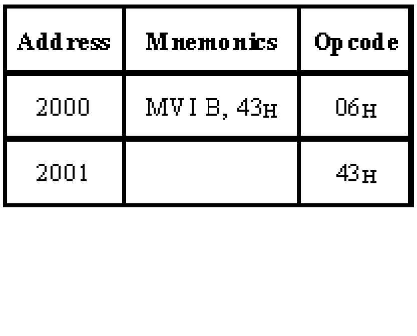


Address	Mnemonics	Opcode
4105	INR M	34_H

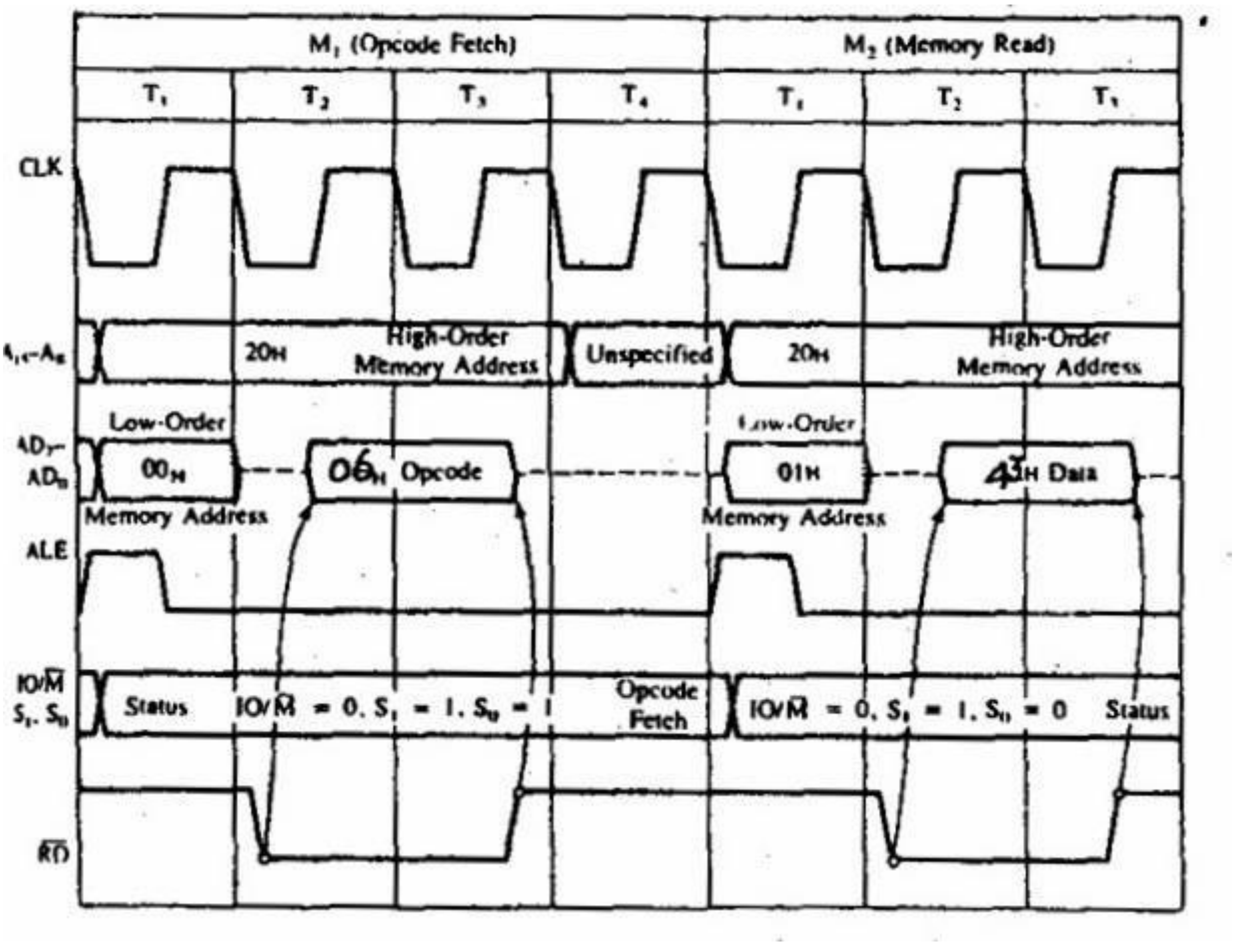


Timing diagram for MVI B, 43H

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



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



13. EVOLUTION OF 16-BIT & 32-BIT MICROPROCESSOR:

EVOLUTION OF 16-BIT MICROPROCESSORS:

The 16-bit processors: MCS-86 family

1. **8086**

Introduced June 8, 1978

Clock rates:

- 5 MHz with 0.33 MIPS
- 8 MHz with 0.66 MIPS
- 10 MHz with 0.75 MIPS

The memory is divided into odd and even banks; it accesses both banks concurrently to read 16 bits of data in one clock cycle

- Bus width 16 bits data, 20 bits address
- Number of transistors 29,000 at 3 µm

- Addressable memory 1 megabyte
- Up to 10X the performance of 8080

First used in the Compaq Deskpro IBM PC-compatible computers. Later used in portable computing, and in the IBM PS/2 Model 25 and Model 30. Also used in the AT&T PC6300 / Olivetti M24, a popular IBM PC-compatible (predating the IBM PS/2 line).

Used segment registers to access more than 64 KB of data at once, which many programmers complained made their work excessively difficult.

The first x86 CPU.

Later renamed the iAPX 86[4]

2. 8088

Introduced June 1, 1979

Clock rates:

- 4.77 MHz with 0.33 MIPS
- 8 MHz with 0.66 MIPS[3]

Internal architecture 16 bits

External bus Width 8 bits data, 20 bits address

Number of transistors 29,000 at 3 µm

Addressable memory 1 megabyte

Later renamed the iAPX 88

3. 80186

Introduced 1982

Clock rates

- 6 MHz with > 1 MIPS

Included two timers, a DMA controller, and an interrupt controller on the chip in addition to the processor (these were at fixed addresses which differed from the IBM PC, although it was used by several PC compatible vendors such as Australian company Cleveland).

Added a few opcodes and exceptions to the 8086 design; otherwise identical instruction set to 8086 and 8088

BOUND, ENTER, LEAVE

INS, OUTS

IMUL imm, PUSH imm, PUSHA, POPA

RCL/RCR/ROL/ROR/SHL/SHR/SAL/SAR reg,imm

Address calculation and shift operations are faster than 8086

Later renamed the iAPX 186

4. 80188

A version of the 80186 with an 8-bit external data bus

Later renamed the iAPX 188

5. 80286

Introduced February 2, 1982

Clock rates:

- 6 MHz with 0.9 MIPS
- 8 MHz, 10 MHz with 1.5 MIPS
- 12.5 MHz with 2.66 MIPS
- 16 MHz, 20 MHz and 25 MHz available.

Bus width: 16 bits data, 24 bits address.

EVOLUTION OF 32-BIT MICROPROCESSORS:

32-bit processors: the non-x86 microprocessors

- iAPX 432
- i960 aka 80960
- i860 aka 80860
- XScale

32-bit processors: the 80386 range

- 80386DX
- 80386SX
- 80376
- 80386SL
- 80386EX

32-bit processors: the 80486 range

- 80486DX
- 80486SX
- 80486DX2
- 80486SL
- 80486DX4

32-bit processors: P5 microarchitecture

- Original Pentium
- Pentium with MMX Technology

32-bit processors: P6/Pentium M microarchitecture

- Pentium Pro
- Pentium II
- Celeron (Pentium II-based)
- Pentium III
- Pentium II and III Xeon
- Celeron (Pentium III Coppermine-based)
- Celeron (Pentium III Tualatin-based)
- Pentium M
- Celeron M
- Intel Core
- Dual-Core Xeon LV

32-bit processors: NetBurst microarchitecture

- Pentium 4
- Xeon
- Mobile Pentium 4-M
- Pentium 4 EE
- Pentium 4E
- Pentium 4F

UNIT – 2

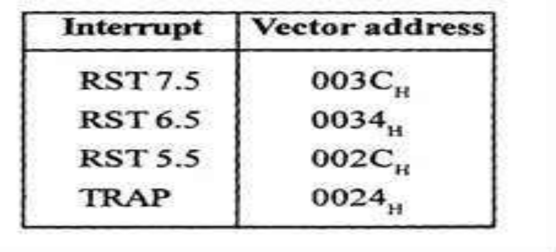
Intel 8085 Interrupts and DMA: 8085 Interrupts – Software and Hardware Interrupts – 8259 Programmable Interrupt Controller - Data Transfer Techniques – Synchronous, Asynchronous and Direct Memory Access (DMA) and 8237 DMA Controller- 8253 Programmable Interval Timer.

1. Define Hardware and Software Interrupt ***

Hardware: 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. Example: TRAP, RST 7.5,6.5& 5.5

Software: 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 is calculated by Interrupt number * 8 = vector address. Example: For RST 5, 5 * 8 = 40 = 28H

2. Give the vector interrupt in 8085 and their location ***



Interrupt	Vector address
RST 7.5	$003C_H$
RST 6.5	0034_H
RST 5.5	$002C_H$
TRAP	0024_H

3. What are the disadvantages of using 8259? ***

- All request are vectored to memory location on00_H, which is reserved for ROM or EPROM and access to this location is difficult often the system has been designed
- Process of determining the priorities are limited and the extra hardware is required to insert the restart instruction.

4. Uses of TRAP ***

- This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable
- In sudden power failure, it executes a ISR and send the data from main memory to backup memory.

5. Difference between maskable and non-maskable interrupt***

Maskable Interrupt	Non-Maskable Interrupt
1. This interrupt can be turned off by the programmer	1. It cannot be turned off by the programmer
2. RST 7.5, 6.5, 5.5 are maskable interrupt	2. TRAP is the only non-maskable interrupt.

6. Define Interrupt service routine. ***

Interrupt is an signal send by an external device to the processor so as to repeat processor to perform a particular task or work.

- ISR is used to store information about the interrupts currently being serviced
- OCWs →Operation Control Word

7. Give the priorities of 8085 interrupts

- TRAP bas the highest priority and vectored interrupt
- The RST 7.5 interrupt is a maskable interrupt.
- The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.
- INTR is the low priority interrupt

8. Difference between synchronous and asynchronous data transfer ***

Synchronous Data Transfer	Asynchronous Data Transfer
1. Simplest data transfer method and can be used when the speed of the microprocessor and the I/O devices matches	1. Data is transferred from the peripheral to the processor or from the processor to the peripheral only after getting a strobe signal from the peripheral.
2. Data can be transferred from processor to peripheral using the OUT instruction	2. Till the handshake signal is received, the processor is in a loop and cannot perform any other task. (i.e.) valuable processor time is wasted.

9. What are the primary features of 8259?

- 8259 manages 8 interrupt requests ($IR_0 – IR_7$)
- 8259 can solve eight levels of interrupt priorities in a variety of modes.
- With additional 8259 devices, the priority scheme can be expanded to 64 levels.
- 8259 is designed to operate only with 8 bit processors. 8259 A is designed to operate only with 8 bits as well as 16 bit processors.

10. What is an interrupt I/O?

The interrupt I/O 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

11. Write an instruction to enable all the interrupts in an 8085 system? ***

```
EI
MVI A,08H .
SIM
```

12. How the 8327 DMA controller transfers 64K bytes of data per channel with Address lines?

The most significant bits D15 and D14 of the count register are used to specify DMA function and the remaining fourteen bits are used to specify the number of bytes to be transferred.

13. What are the signals used by the DMA controller? ***

The Signals are:

- HLDA
- DMA request
- DMA acknowledge
- AEN – address enable
- ADSTB- address strobe

14. Give the additional features of 8259A controller?

- Input triggering
- Interrupt Status
- Poll Method

15. How the signals of the 8237 are classified? ***

The signals are classified in to two groups.

i. One group of signals are used for interfacing with the MPU

ii. Second group for communicating with the peripherals.

16. How long the INTR pulse stays high?

The INTR pulse can remain high until the interrupt flip-flop is set by the EI instruction in the service routine.

17. Give the three formats of END of Interrupt? ***

- NON-specific EOI command
- Specific EOI command
- Automatic interrupt

18. What are the two modes of DMA execution? ***

Slave Mode, Master mode

19. What do you mean by control logic?

This has two pins. INT as an output, and INTA as an input. The INT isconnected to the interrupt pin of the MPU.

20. Give the commonly used priority modes?

- Fully Nested mode
- Automatic rotation mode
- Specific rotation mode

21. What is RIM? ***

RIM: Read Interrupt Mask Used for three functions
a. To read interrupt mask
b. To identify the pending interrupt
c. To receive serial data

22. What is SIM? ***

SIM: Set interrupt Mask. It is a 1-byte instruction. Used for three functions
a. To set the Mask
b. To reset the flip flop
c. Implement the I/O

23. Give the operating modes of 8259a?

- Fully Nested Mode
- End of Interrupt (EOI)
- Automatic Rotation
- Automatic EOI Mode
- Specific Rotation
- Special Mask Mode
- Edge and level Triggered Mode
- Reading 8259 Status
- Poll command
- Special Fully Nested Mode
- Buffered mode
- Cascade mode

24. List the command words of 8259A?

- Initialization command word &
- Operation command word

25. Name the 6 modes of operations of an 8253 programmable interval timer.***

- Mode 0:interrupt on terminal count
- Mode 1:hardware re -triggerable one-shot
- Mode 2 :rate generator
- Mode3:square wave rate generator
- Mode 4:software triggered strobe
- Mode 5:hardware triggered strobe

26. What is meant by interrupt? **

Interrupt is an external signal that causes a microprocessor to jump to a specific subroutine (ie. Program counter is modified to address specified in the ISR).

Unit II

Intel 8085 Interrupts and DMA: 8085 Interrupts – Software and Hardware Interrupts – 8259 Programmable Interrupt Controller - Data Transfer Techniques – Synchronous, Asynchronous and Direct Memory Access (DMA) and 8237 DMA Controller- 8253 Programmable Interval Timer.

INTERRUPTS

Definition:

Interrupt is a signal send by an external device to the processor so as to request the processor to perform a particular task or work.

Interrupt is a process where an external device can get the attention of the microprocessor.

Analogy of interrupt process:

Assume that you are reading an interesting novel at your desk, where there is a telephone. For you to receive and respond to a telephone call, the following step should occur:

1. The telephone system should be enabled, meaning that the receiver should be on the hook.

2. You should glance at the light at certain intervals to check whether someone is calling.

3. If you see a blinking light, you should pick up the receiver, say hello, and wait for a response. Once you pick up the phone, the line is busy, and no more calls can be received until you replace the receiver.

4. Assuming that the caller is your roommate, the request may be: It is going to rain today. Will you please shut all the windows in my room?

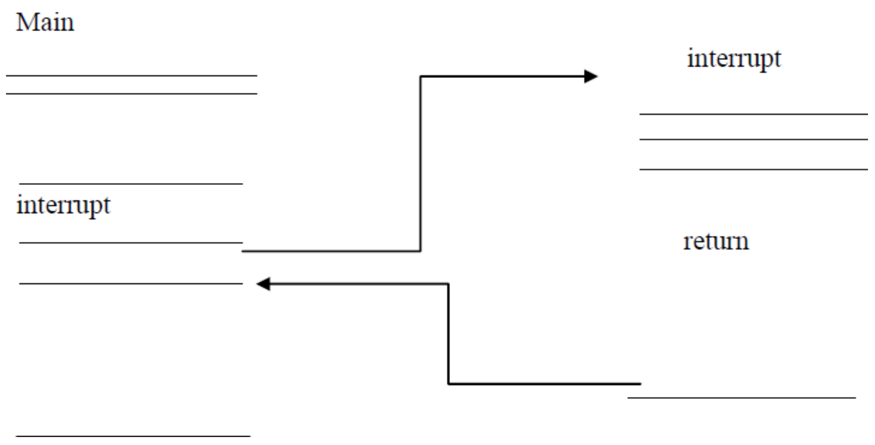
5. You insert a bookmark on the page you are reading.

6. You replace the receiver on the hook.

7. You shut your roommate’s windows.

8. You go back to your book, find your mark, and start reading again.

[Note: steps 6 and 7 may be interchanged, depending on the urgency of the request.]



Classification Of Interrupts:

Interrupts can also be classified into two types:

Maskable Interrupts (Can be delayed or rejected)

Non-Maskable Interrupts (Cannot be delayed or rejected)

Interrupts can also be classified into two types:

Vectored (the address of the service routine is hard-wired)

Non-vectored (the address of the service routine needs to be supplied externally by the device)

An interrupt is considered to be an emergency signal that may be serviced.

- The Microprocessor may respond to it as soon as possible.

What happens when MP is interrupted?

When the Microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt.

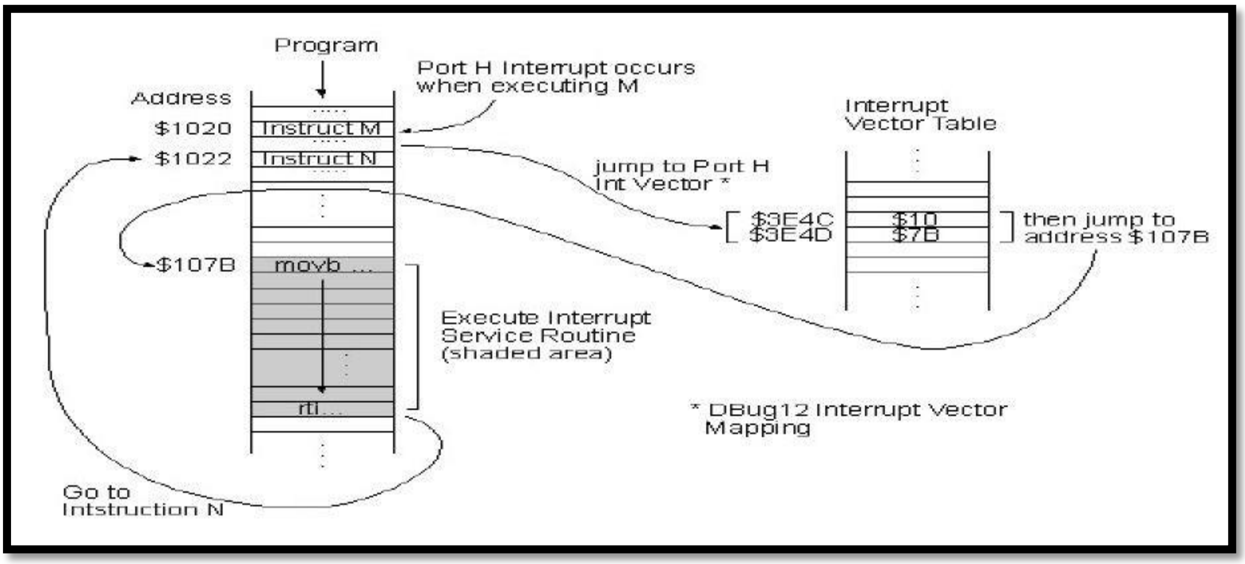
Each interrupt will most probably have its own ISR.

RESPONDING TO INTERRUPTS:

Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not.

There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.

- *Vectored:* The address of the subroutine is already known to the Microprocessor.
- *Non Vectored:* The device will have to supply the address of the subroutine to the Microprocessor.


Program
Port H Interrupt occurs when executing M
Address
$1020
$1022
Instruct M
Instruct N
Interrupt Vector Table
jump to Port H Int Vector *
$3E4C
$3E4D
$10
$7B
then jump to address $107B
$107B
movb ...
Execute Interrupt Service Routine (shaded area)
rti ...
* DBug12 Interrupt Vector Mapping
Go to Intstruction N

1. THE 8085 INTERRUPTS:

When a device interrupts, it actually wants the MP to give a service which is equivalent to asking the MP to call a subroutine. This subroutine is called ISR (Interrupt Service Routine)

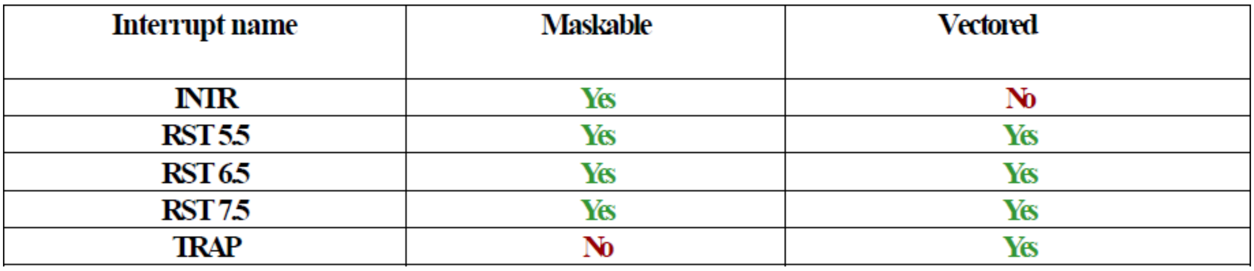
- The 'EI' instruction is a one byte instruction and is used to Enable the non-maskable interrupts.
- The 'DI' instruction is a one byte instruction and is used to Disable the non-maskable interrupts.
- The 8085 has a single Non-Maskable interrupt.
- The non-maskable interrupt is not affected by the value of the Interrupt Enable flip flop.

The 8085 Has 5 Interrupt Inputs:

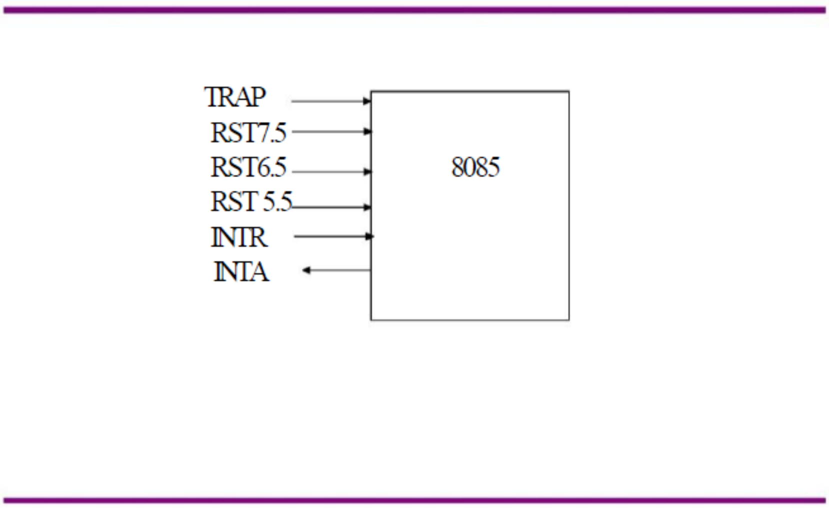
- The INTR input
  - ✓ The INTR input is the only non-vectored interrupt
  - ✓ INTR is maskable using the EI/DI instruction pair

- RST 5.5, RST 6.5, RST 7.5 are all automatically vectored
  - ✓ RST 5.5, RST 6.5, and RST 7.5 are all maskable

- TRAP is the only non-maskable interrupt in the 8085
  - ✓ TRAP is also automatically vectored



Interrupt name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes



8085 Interrupts

Interrupt Vectors And The Vector Table:

- 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

Example: Let, a device interrupts the Microprocessor using the RST 7.5 interrupt line.

Because the RST 7.5 interrupt is vectored, Microprocessor knows, in which memory location it has to go using a call instruction to get the ISR address. RST7.5 is knows as Call 003Ch to Microprocessor. Microprocessor goes to 003C location and will get a JMP instruction to the actual ISR address. The Microprocessor will then, jump to the ISR location

The 8085 Non-Vectored Interrupt Process:

1. The interrupt process should be enabled using the EI instruction.

2. The 8085 checks for an interrupt during the execution of every instruction.

3. If INTR is high, MP completes current instruction, disables the interrupt and sends INTA (Interrupt acknowledge) signal to the device that interrupted.

4. INTA allows the I/O device to send a RST instruction through data bus.

5. Upon receiving the INTA signal, MP saves the memory location of the next instruction on the stack and the program is transferred to 'call' location (ISR Call) specified by the RST instruction.

6. Microprocessor Performs the ISR.

7. ISR must include the 'EI' instruction to enable the further interrupt within the program.

8. RET instruction at the end of the ISR allows the MP to retrieve the return address from the stack and the program is transferred back to where the program was interrupted.

The 8085 Recognizes 8 Restart Instructions:

RST0 - RST7.

Each of these would send the execution to a predetermined hard-wired memory location:

Restart Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H

1.2 SOFTWARE INTERRUPTS OF 8085:

1. The software instructions are program instructions. When a software interrupt instruction is executed, the processor executes an interrupt service subroutine (ISR) stored in the vector address of that software interrupt instruction.

2. The software interrupts of 8085 are RSR 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 and RST 7. The software interrupts of 8085 are vectored interrupts. The software interrupts cannot be masked and they cannot be disabled.

3. The vector addresses of software interrupt are given in the table.

4. The software interrupt instructions are included at the appropriate (or required) place in the main program.

5. When the processor encounters the software instruction, it pushes the content of the PC (program counter) to stack.

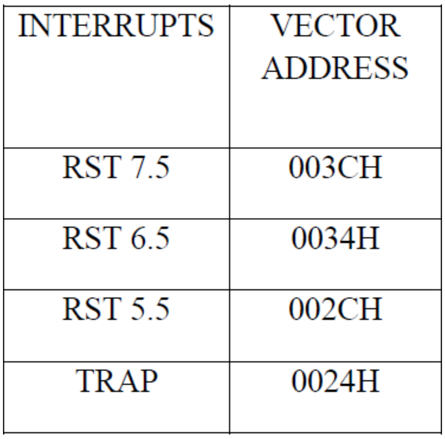
6. Then loads the vector address in PC and starts executing an ISR stored in this address.

7. The last instruction of ISR will be RET instruction. When the RET instruction is executed, the

8. Processor POP the content of the top of stack to PC.

Hence the processor control returns to the main program after receiving the interrupt.

1.3 HARDWARE INTERRUPTS OF 8085:



INTERRUPTS	VECTOR ADDRESS
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH
TRAP	0024H

The hardware interrupts of 8085 are initiated by an external device by placing an appropriate signal at the interrupt pin of the processor.

- The processor keeps on checking the interrupt pins at the second T-state of last machine cycle of every instruction.
- If the processor finds a valid interrupt signal and if the interrupt is unmasked and enabled; then the processor accepts the interrupt. The acceptance of the hardware interrupt is acknowledged by sending an INTA signal to the interrupting device.
- When the interrupt is accepted; the processor saves the content of PC into the stack.
- The hardware interrupts of 8085 are TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR. except INTR all are vectored interrupts.
- In vectored interrupts the address to which the program control is transferred is fixed by the manufacturer.
- The vector addresses of hardware interrupts are given in table.
- TRAP is edge and level sensitive. Hence to initiate TRAP, the interrupt signal has to make a low to high transition and then it has to remain high until the interrupt is recognized.
- The RST 7.5 is edge sensitive. , and in order to initiate it ,the interrupt signal has to make a low to high transition and then it need not remain high until the interrupt is recognized.

- The RST 7.5, RST 6.5, RST 5.5 and INTR are level sensitive. Hence these interrupts should remain high, until it is recognized.
- The Trap is non maskable interrupt and RST 7.5, RST 6.5, RST 5.5 are maskable interrupt using SIM instruction. The status of maskable interrupts can be read into the accumulator by executing RIM instruction.
- All the interrupts except TRAP are disabled when the processor is resettled or can also be disabled by executing DI instruction.
- In order to enable them the processor has to execute EI instruction.

1.4 MULTIPLE INTERRUPTS & PRIORITIES:

- The microprocessor can only respond to one signal on INTR at a time. Therefore, we must allow the signal from only one of the devices to reach the microprocessor.
- We must assign some priority to the different devices and allow their signals to reach the microprocessor according to the priority.

The Priority Encoder

- The solution is to use a circuit called the priority encoder (74LS148).

  – This circuit has 8 inputs and 3 outputs.

  – The inputs are assigned increasing priorities according to the increasing index of the input.

- Input 7 has highest priority and input 0 has the lowest.

  – The 3 outputs carry the index of the highest priority active input.

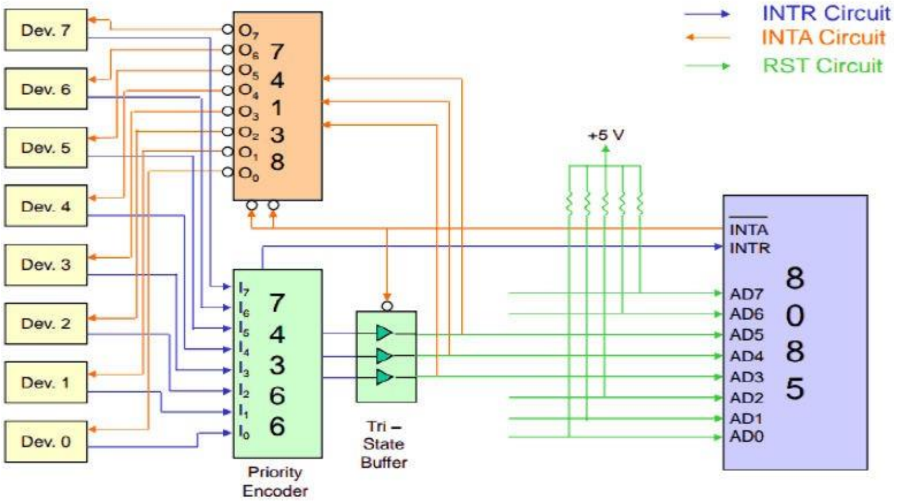
  – Below figure shows how this circuit can be used with a Tri-state buffer to implement an interrupt priority scheme.

Multiple Interrupts & Priorities:

- Note that the opcodes for the different RST instructions follow a set pattern.
  - Bit D5, D4 and D3 of the opcodes change in a binary sequence from RST 7 down to RST 0.
  - The other bits are always 1.
  - This allows the code generated by the 74366 to be used directly to choose the appropriate RST instruction.

The one drawback to this scheme is that the only way to change the priority of the devices connected to the 74366 is to reconnect the hardware.

Multiple Interrupts and Priority



The 8085 Maskable/Vectored Interrupts:

- The 8085 has 4 Masked/Vectored interrupt inputs.

  – RST 5.5, RST 6.5, RST 7.5

  - They are all maskable.
  - They are automatically vectored according to the following table:

Interrupt	Vector
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

The vectors for these interrupt fall in between the vectors for the RST instructions. That’s why they have names like RST 5.5 (RST 5 and a half).

Masking RST 5.5, RST 6.5 And RST 7.5:

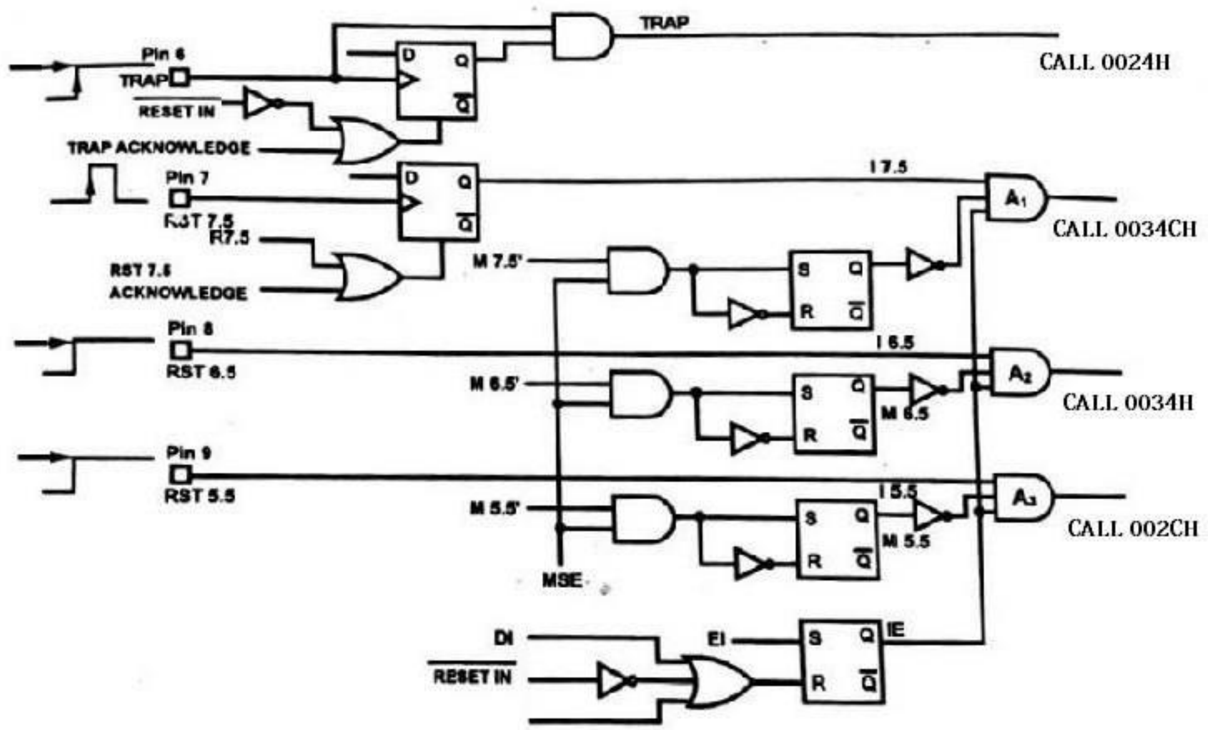
- These three interrupts are masked at two levels:

  1. Through the Interrupt Enable flip flop and the EI/DI instructions.

     - The Interrupt Enable flip flop controls the whole maskable interrupt process.

  2. Through individual mask flip flops that control the availability of the individual interrupts.

- These flip flops control the interrupts individually.



Maskable Interrupts and vector locations

The 8085 Maskable/Vectored Interrupt Process:

1. The interrupt process should be enabled using the EI instruction.

2. The 8085 checks for an interrupt during the execution of every instruction.

3. If there is an interrupt, and if the interrupt is enabled using the interrupt mask, the microprocessor will complete the executing instruction, and reset the interrupt flip flop.

4. The microprocessor then executes a call instruction that sends the execution to the appropriate location in the interrupt vector table.

5. When the microprocessor executes the call instruction, it saves the address of the next instruction on the stack.

6. The microprocessor jumps to the specific service routine.

7. The service routine must include the instruction EI to re-enable the interrupt process.

8. At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

2. 8259 PROGRAMMABLE INTERRUPT CONTROLLER

The following are the features of 8259A:

1. 8259A handles up to 8-vectored priority interrupts for the CPU
2. It is cascadable for up to 64-vectored priority interrupts without additional circuiting.
3. The priority modes can be changed or reconfigured dynamically at any time during the main program.
4. 8259 can be used with 8080/8085 or 8086/8088 microprocessors.
5. The various interrupt modes it can operate are:
    - Fully nested mode
    - Rotating priority mode
    - Special mask mode
    - Polled mode
6. 8259A supports both edge & level triggered mode of interrupted.
7. The data bus can be buffered.
8. The AEOI can be programmed.
9. The CALL address interval can be programmed to either 4 or 8.

Pin configuration:

- 8259A is packed in a 28 pin DIP, using NMOS technology and requires a single +5 V supply.
- Circulating is static, requiring no check input.
- The pin configuration of 8259A is illustrated in fig.

D0-D7 (Data Bus)

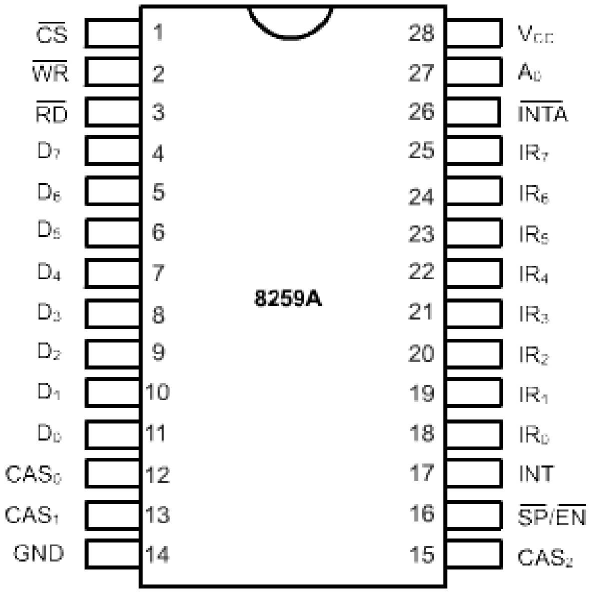
- Bidirectional data is used to transfer control, status and interrupt vector information.

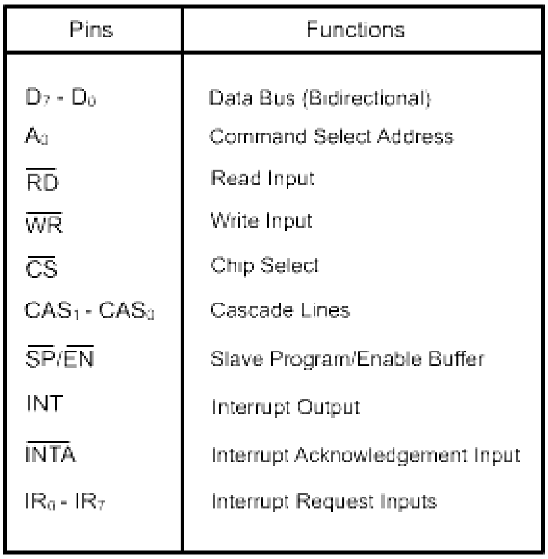
A0 (Address Line)

- This pin acts in conjunction with CS,WR and RD pins.
- It is used by 8259A to decipher various command words.
- The CPU writes to initialize command words and reads the status.
- It is typically connected to the CPU. A1 reads the address line of 8086.

CS (Chip Select)

- A low on this pin enables RD and WR communication between the CPU and the 8259A.INTA functions are independent of CS.





Pins	Functions
D_7 - D_0	Data Bus (Bidirectional)
A_0	Command Select Address
$\overline{\text{RD}}$	Read Input
$\overline{\text{WR}}$	Write Input
$\overline{\text{CS}}$	Chip Select
CAS_1 - CAS_3	Cascade Lines
$\overline{\text{SP}}/\overline{\text{EN}}$	Slave Program/Enable Buffer
INT	Interrupt Output
$\overline{\text{INTA}}$	Interrupt Acknowledgement Input
IR_0 - IR_7	Interrupt Request Inputs

BLOCK DIAGRAM

The internal block diagram of 8259 contains the 4 sections, they are:

1. Interrupts and control logic sectionData bus buffer

2. Read/Write control logic section

3. Cascade Buffer/Comparator Section

Interrupts and Control Logic Section:

This section consists of the following registers:

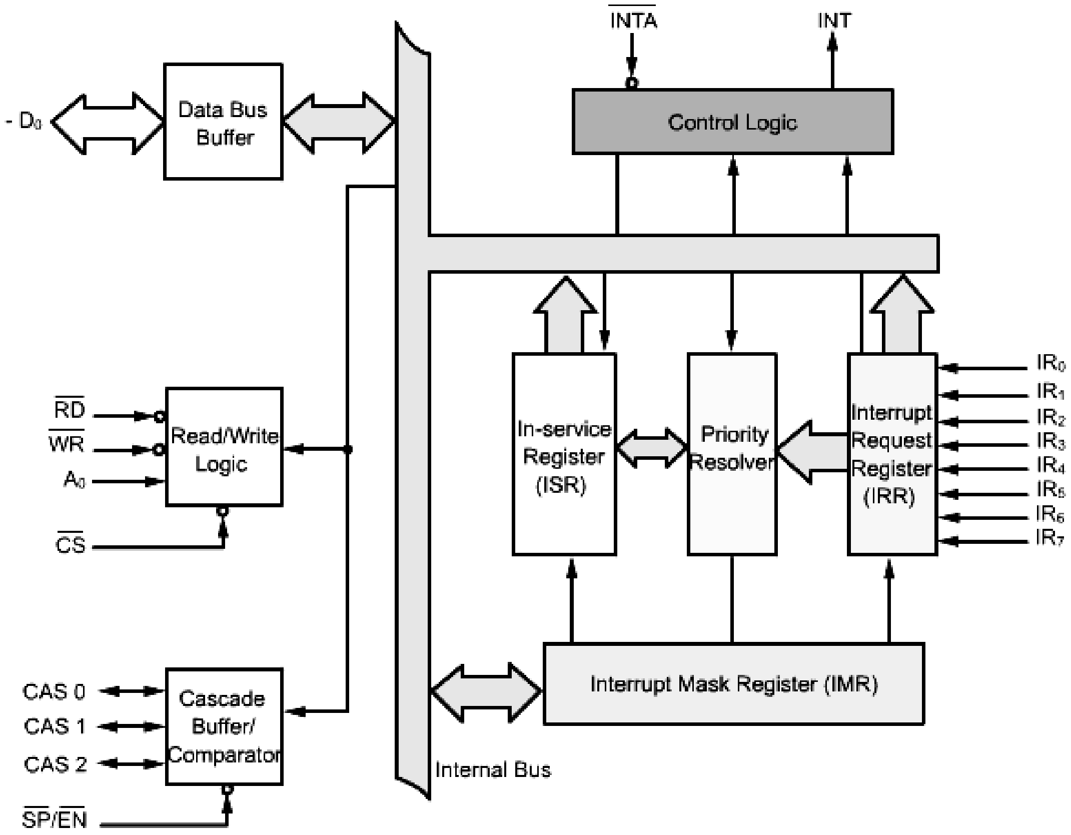
(a) Interrupt request Register (IRR)

(b) In-Service Register (ISR)

(c) Priority Resolver

(d) Interrupt Mask Register (IMR)

(e) Control logic Block



Block Diagram

Data Bus Buffer

- This 3-state, bidirectional 8-bit data buffer is used to interfere the 8259A to the system data bus.
- Control words status information are transferred through the data bus buffer.
- INTA functions are independent of CS.

Read/Write Logic

- The function of this block is to accept commands from the CPU.
- Command Word Registers (OCW registers) which are programmed by the processor to set up the 8259, and to operate This section contains the initialization Command Word Registers (ICW registers) and the Operation it in various modes.
- This section also accepts Read commands from the CPU to permit the CPU to read status words.
- When the address line Ao is at logic 0, the controller is selected to write a command or read a status.
- The chip select logic and Ao determine the port address of the controller.
- The operation command word (OCW) register which store the various control formats which define the device operation.
- This function block also allows the status of the 8359Ato be transferred onto the data bus.

The pins associated with this section are described below:

CHIP SELECT:

- This is an active low input which is used to select the device

WRITE:

- This is an active low input and is used to write OCWs and ICWs onto the 8259

READ:

- This is also an active low input.
- It is used by the CPU to read the status of the IRR, ISR, IMR, or the interrupt level.

Cascade Buffer / Comparator

- This function block stores and compares the ID (identification code) of all the 8259A's used in the system.
- The 8259 can be cascaded with other 8259s in order to expand the interrupt handling capacity to sixty-four levels.
- In such a case, the former is called a master and the latter are called slaves.
- This block is used to expand the number of interrupt levels.
- The associated 3 I/O pins (CAS0-CAS2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave.
- As a master, the 8259A sends the ID of the interrupting slave device on to the CAS0-CAS2 lines.

The slave thus selected will output the pre-programmed subroutine address onto the data bus in response to the INTA pulses from the CPU.

Control Logic

- The logic blocks controls the overall operation of the controller.
- This block has an input and output line.
- The 8259, after resolving its input interrupt request priorities, places an interrupt request to the processor on the INT output.
- This is directly connected to the cpu interrupt input.
- It generates interrupt to the microprocessor and receive INTA signal.
- It enables the data bus buffer to send the required information when ¯I¯N¯T¯A signal is obtained.

Interrupt request register (IRR)

- IRR is used to store all the interrupt levels which are requesting service.
- IRR has eight input lines for interrupts.
- The eight interrupt inputs set corresponding bits of the Interrupt request register.

- When these lines go to HIGH, the requests are stored in the register

Priority Resolver (PR)

- The logic block determines the priorities of the bits set in the IRR.
- The highest priority is selected and strobed into the corresponding bit of the ISR in response to the pulse.

Interrupt Mask Register (IMR)

- The IMR stores the interrupt mask information (i.e., whether each interrupt has to be enabled/disabled).
- This register can be programmed by an OCW to store the bits which mask interrupts.
- The 8 bits of the IMR are used to disable or enable individual interrupt inputs; writing 0 in the corresponding bit location enables the interrupt.
- An interrupt, which is masked by software (by programming the IMR), will not be recognized and serviced even if it sets the corresponding bit in the IRR.
- The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt requests lines of lower priority interrupts.

Interrupt Service Register (ISR)

- The ISR is used to store all the interrupt levels which are being serviced.

Interrupt Operations:

- First, the 8259 should be initialized by placing control words in the control register.
- It requires two types of control words:

(1)Initialization command words

(2) Operational command words

Priority Interrupt Modes:

- Many types of priority modes are available under software control in the 8259.
- They can be changed dynamically during the program by writing appropriate command words.

Commonly used priority modes are given below:

1. FULLY NESTED MODE:

- This is a general purpose mode.
- In this mode all IRs (Interrupt Requests) arranged from highest to lowest IRo has the highest priority and IR7 has the lowest priority.

2. AUTOMATIC ROTATION MODE:

- In this mode, a device after being serviced, receives the lowest priority .

3. SPECIFIC ROTATION MODE:

- This mode is similar to the automatic rotation maode, expect that the user can select any IR for the lowest priority , thus fixing all other priorities.

END OF INTERRUPT

- After the completion of the interrupt service, the corresponding ISR bit needs to be reset to update the information in the ISR.
- This is called the end-of-interrupt (EOI) command.
- It can be issued three formats.

1. Non Specific EOI command

When this command is sent to the 8259 it resets the highest priority ISR bit.

2. Specific EOI command

This command specifies, which ISR bit to reset. 3. AUTOMATIC EOI

- In this mode no command is necessary.
- The major drawback with this mode is that the ISR does not have information on which IR is being serviced .
- Thus nay IR can interrupt the service routine, irrespective of its priority , if the interrupt enable Flip-Flop is set.

VECTORING DATA FORMATS

- The eight interrupts levels generate CALLs to eight equally spaced locations in memory.
- These locations can be programmed to be spaced at an intyerval of four or eight locations.
- The vectoring table will therefore be either a page of thirty-two bytes , or a page of sixty- four bytes.

INITIALIZATION COMMAND WORD (ICW1)

- ICW 1 is used to give the information about single or multiple 8259s in the system.
- 4 or 8 bytes of interval between the interrupt vector locations .
- The address bit A7- A5 of the CALL instruction.

IC4

- This is used toi specify whether ICW4 is required or not required.
- If it is set to 0, the 8259 is set in the non-buffered mode.

SIGNAL

- This is used to inform the the 8259 if it is the only 8259 in the system, or if additional 8259s are present.

ADI

- This sets the CALL address interval to either four or eight

LTIM

- This bit determines if the interrupts are to be recognized in the level-triggered mode or in the edge - triggered mode.

A5 –A7

- A0 – A4 of the vectoring address are automatically inserted the 8259 for all the IR inputs for an interval spacing of four.
- A0 – A5 are inserted automatically for an interval spacing of eight.

INITIALIZATION COMMAND WORD2 (ICW 2)

- A write command following ICW1 with A0=1, is interpreted as ICW2.
- The format of the byte to be loaded as ICW2
- This is used to lad the high-order byte of the interrupt vector address of all the interrupts.
- This byte is common for all interrupts.

OPERATION COMMAND WORDS (OCWS)

- After initialization, the 8259 is ready to process interrupt requests.
- Operation Command Words are used to change manner of processing the interrupts during operation.
- They may be located anytime after the 8259s initialization to dynamically after the priority modes.

4. DATA TRANSFER TECHNIQUES

The data transfer technique refers to the method of data transfer between the processor and peripheral devices. In a typical microcomputer, data transfer takes place between any two devices:

- Microprocessor and memory

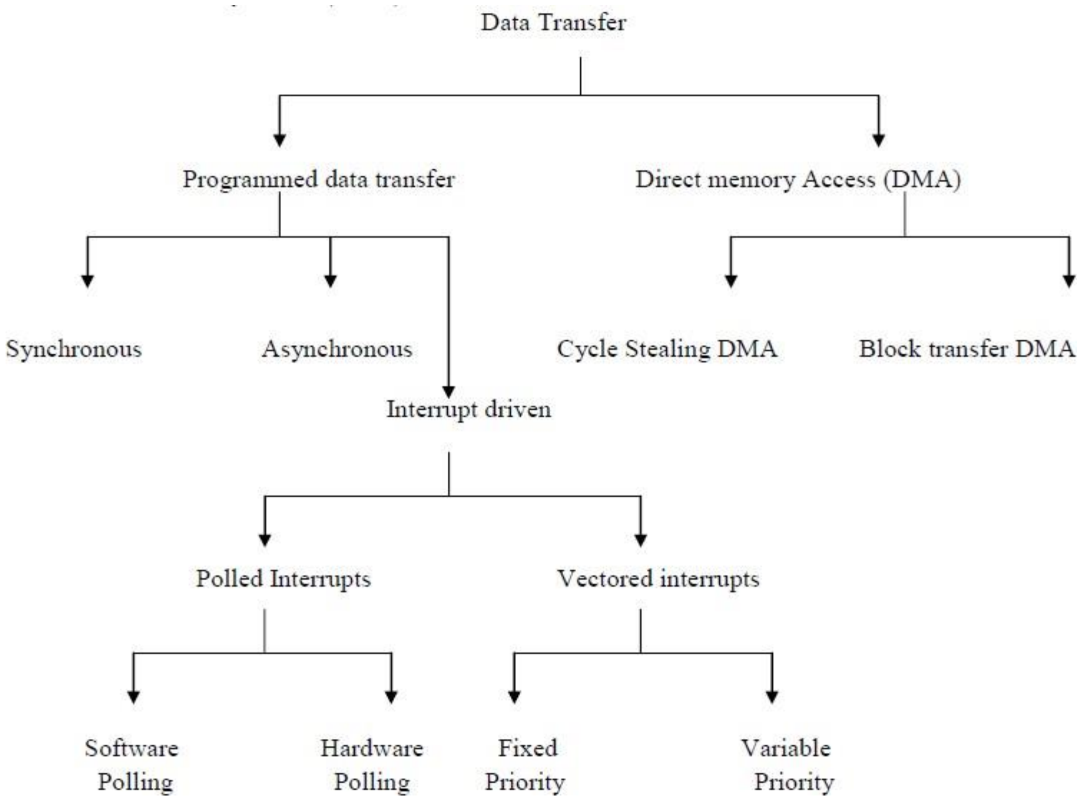
- Microprocessor and I/O devices
- Memory and I/O devices

For effective data transfer between these devices, the timing parameters of the devices should be matched.

TYPES OF DATA TRANSFER:-

The data transfer techniques have been broadly classified into the following two categories;

- Programmed data transfer
- Direct memory access (DMA) data transfer



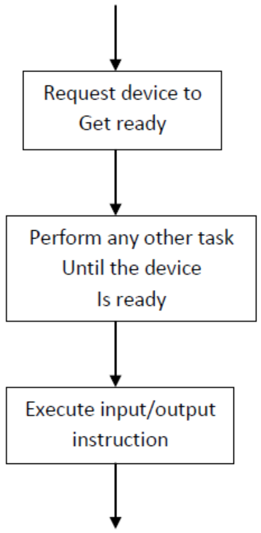
PROGRAMMED DATA TRANSFER:

In programmed data transfer, a memory resident request the device for data transfer to or from one of the processor register. It is used when relatively small amount of data are to be transferred. They can be further classified into the following three types;

- Synchronous data transfer
- Asynchronous data transfer
- Interrupt driven data transfer

SYNCHRONOUS DATA TRANSFER:

The synchronous data transfer scheme is the simplest of all data transfer schemes. In this scheme the processor does not check the readiness of the devices. The I/O device or peripheral should have matched timing parameters. Whenever data is to be obtained from the device or transferred to the device, the user program can issue a suitable instruction for the device. At the end of he execution of this instruction, the transfer would have been completed.



Synchronous data Transfer scheme

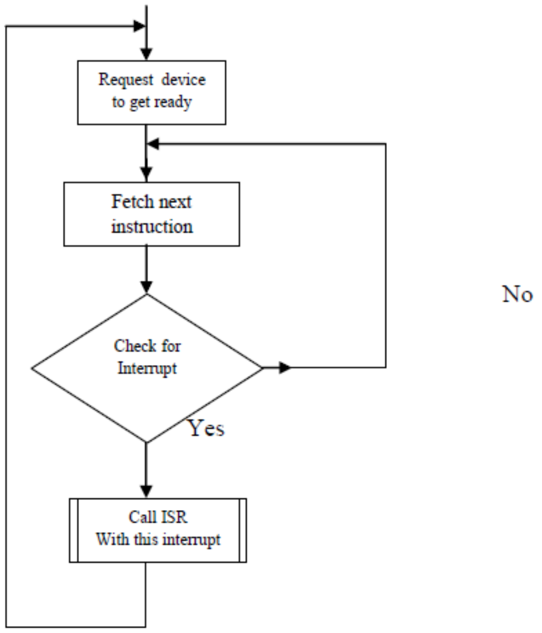
ASYNCHRONOUS DATA TRANSFER:

The asynchronous data transfer scheme is employed when the speed of the processor and I/O devices does not match. In this scheme the processor sends a request to the device for read/write operation. Then the processor keeps on polling the status of the devices. Once the device is ready, the processor executes a data transfer instruction to complete the process.

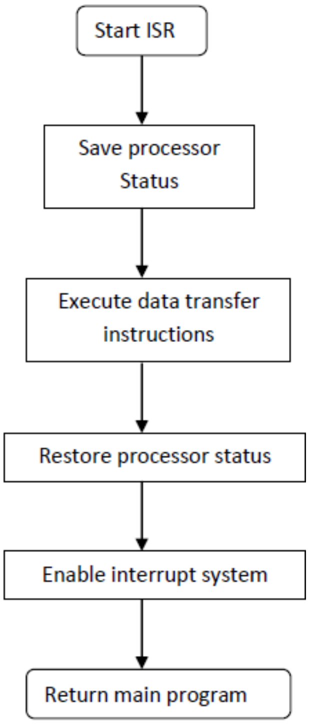
INTERRUPT DRIVEN DATA TRANSFER:

The interrupt driven data transfer scheme is the best method of data transfer for efficient utilization of processor time. In this scheme, the processor first initiates the I/O device for data transfer. After initiating the device, the processor will continue the execution of instructions in the program. Also at the end of every instruction the processor will check for a valid interrupt signal. If there is no interrupt then the processor will continue the execution.

When the I/O device is ready, it will interrupt the processor. On receiving an interrupt signal the processor will complete the current instruction execution and save the processor status in stack. Then the processor calls an Interrupt Service Routine (ISR) to service the interrupting device. At the end of ISR, the processor status is retrieved from the stack and the processor starts executing its main program.



Main program execution sequence.



ISR execution sequence

4. DIRECT MEMORY ACCESS (DMA):

Normally the data transfer from memory to I/O device or I/O device to memory can be achieved only through microprocessor. When data has to be transferred from memory to I/O device, first the processor sends address and control signals to memory to read the data from memory. Then the processor send address and control signals to I/O device to write data to I/O device. Similarly, the data can be transferred from I/O device to memory.

In the data transfer method described above, the data cannot be directly transferred between memory and I/O devices, even though they are connected to common bus. This process is inevitable, because the processor cannot simultaneously select two devices. Hence a scheme called DMA has been developed in which the I/O device can access the memory directly for data transfer.

The DMA data transfer will be useful to transfer large amount of data between memory and I/O device in a short time.

The modes of DMA operations are two types;

- Burst mode
- Cycle stealing

BURST MODE DATA TRANSFER:

As bus control is granted to the device controller, it continues with the controller till the data transfer is completed. After all the data has been transferred, the device interrupts the processor to indicate the completion to the user program. During this period the microprocessor is idling and it is in hold state.

The microprocessor exits from this state only after an interrupt is received or after the DMA request is withdrawn by the peripheral device. The duration of HOLD depends on I/O device speed, the memory speed and the number of bytes transferred. This type of DMA transfer is known as burst mode data transfer.

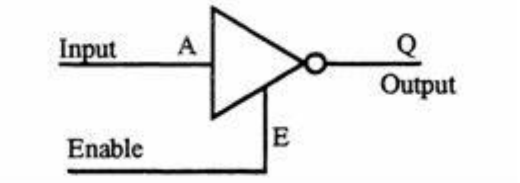
CYCLE STEALING DATA TRANSFER:

The I/O device uses the concept of cycle stealing. The I/O device requests the processor for DMA cycle. When request is granted a byte or a word is transferred and DMA request is withdrawn. After sometime, when the device is again ready for data transfer, it repeats the above process.

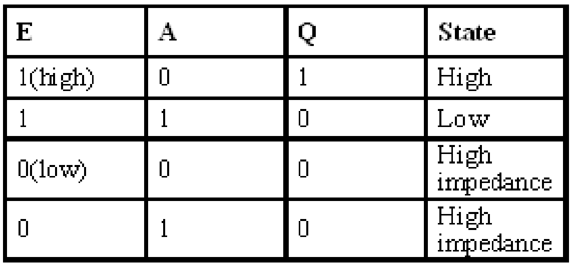
Finally when the last data byte has been transferred, the device interrupts the processor indicating the end of the requested I/O operation. This type of DMA access is cycle stealing data transfer.

Direct Memory Access (DMA)

Tri state devices:



- 3 output states are high & low states and additionally a high impedance state.
- When enable E is high the gate is enabled and the output Q can be 1 or 0 (if A is 0, Q is 1, otherwise Q is 0). However, when E is low the gate is disabled and the output Q enters into a high impedance state.



E	A	Q	State
1(high)	0	1	High
1	1	0	Low
0(low)	0	0	High impedance
0	1	0	High impedance

Fig (a) - Pin Diagram of 8085

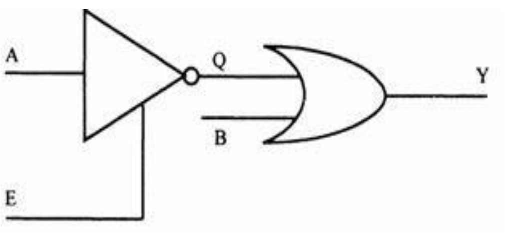


Fig (b) - logical schematic of Pin diagram.

- For both high and low states, the output Q draws a current from the input of the OR gate.
- When E is low, Q enters a high impedance state; high impedance means it is electrically isolated from the OR gate's input, though it is physically connected. Therefore, it does not draw any current from the OR gate's input.

- When 2 or more devices are connected to a common bus, to prevent the devices from interfering with each other, the tristate gates are used to disconnect all devices except the one that is communicating at a given instant.
- The CPU controls the data transfer operation between memory and I/O device. Direct Memory Access operation is used for large volume data transfer between memory and an I/O device directly.
- The CPU is disabled by tri-stating its buses and the transfer is effected directly by external control circuits.
- HOLD signal is generated by the DMA controller circuit. On receipt of this signal, the microprocessor acknowledges the request by sending out HLDA signal and leaves out the control of the buses. After the HLDA signal the DMA controller starts the direct transfer of data.

READY (input)

- 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.

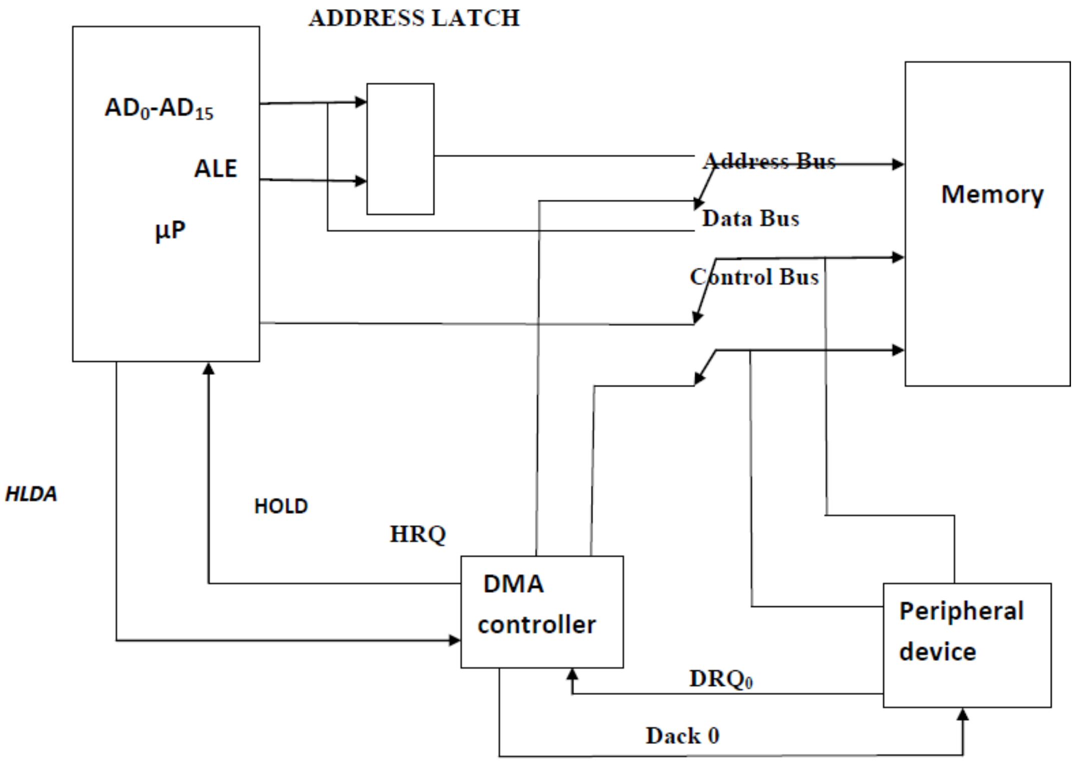
Single Bit Serial I/O ports:

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

DMA CONTROLLER

To transfer the data at a faster rate, the CPU is isolated and transfer of data is affected between the memory and the peripheral directly. This I/O technique is called Direct Memory Access (DMA).

Direct memory access operation (DMA):



- The DMA operation can be done with three switches.
- Normally, the switch positions are such that the memory and peripheral devices are connected to the CPU, i.e., the data lines, address lines and the control lines of the memory and I/O devices are connected to the CPU.
- Whenever the DMA operation is to be carried out, the CPU I totally isolated and the address lines and control lines are taken over by the DMA controller circuitry.
- To actuate the DMA operation, the following sequence is carried out
  - The device which needs data transfers the device and the memory has to send DMA request (DRQ) to the DMA controller.
  - The DMA controller raises the Hold request (HRQ) line and it is connected to the Hold signal input of the µp
  - The µp tristate all the address lines, data lines and control lines and acknowledges the Hold input signal through Hold Acknowledge (HLDA) output signal.
  - The HLDA signal is connected to the DMA controller. Once it becomes active, then DMA controller takes care of direct data transfer operation.

- It sends acknowledgement signal (DACK)
- To the device which requested for DMA operation to enable the device for data transfer.
- DMA operation is carried out by sending suitable address to the memory and suitable control signals to transfer a byte of data. Then it increments the address and send control signals and so on.
- Before actually carrying out the DMA operation, the DMA controller should know

A. The starting address of the memory location.

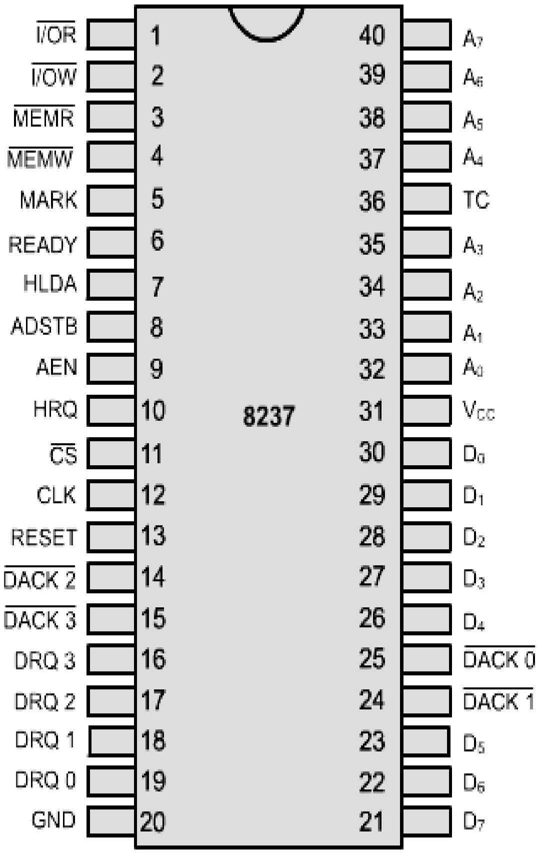
B. Number of bytes to be transferred.

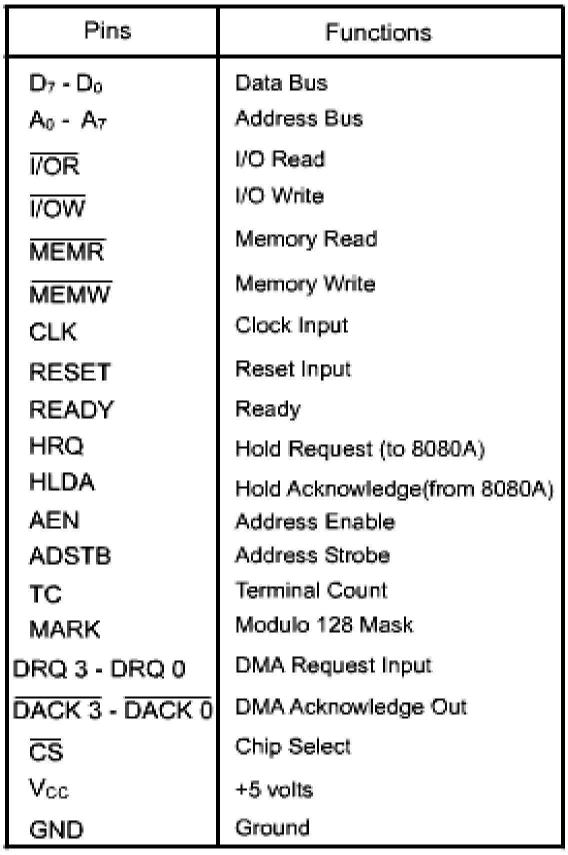
C. Whether the transfer is from memory to I/O or from I/O to memory.

6. PROGRAMMABLE 8237 DMA CONTROLLER

Features

- In DMA, data can be transferred between the memory and an external device without utilizing the microprocessor.
- DMA is typically used to transfer large volume of data between memory and an external device.
- A DMA read operation transfers data from memory to an external device.
- A DMA write operation transfers data from an external device to memory.
- The device called DMA controller is used to perform read and write operations in the same manner as the processor.
- Therefore, the DMA controller is actually a special-purpose microprocessor whose only task is to perform high-speed data transfers between memory and an external device.
- The INTEL 8237 DMA controller is a 40 pin programmable device.
- The 8237 has four independent DMA channels.
- One 8237 can provide DMA transfers to four external devices.
- Data transfers begin with DMA request lines DREQ 0 - DREQ 3.
- DREQ 0 has the highest priority and DREQ 3 has the lowest priority.
- DACK 0 – DACK 3 signals are used to acknowledge a DMA channel request.
- After being initialized by the MPU, the 8237 takes control of the bus in order to perform the DMA operation.
- Data bits are then transferred between a peripheral or external device without involving the microprocessor.
- The basic pin configuration of the 8237 is shown in the figure.


$\overline{\text{I/OR}}$ 1
$\overline{\text{I/OW}}$ 2
$\overline{\text{MEMR}}$ 3
$\overline{\text{MEMW}}$ 4
MARK 5
READY 6
HLDA 7
ADSTB 8
AEN 9
HRQ 10
$\overline{\text{CS}}$ 11
CLK 12
RESET 13
$\overline{\text{DACK 2}}$ 14
$\overline{\text{DACK 3}}$ 15
DRQ 3 16
DRQ 2 17
DRQ 1 18
DRQ 0 19
GND 20
8237
40 A_7
39 A_6
38 A_5
37 A_4
36 TC
35 A_3
34 A_2
33 A_1
32 A_0
31 V_{CC}
30 D_0
29 D_1
28 D_2
27 D_3
26 D_4
25 $\overline{\text{DACK 0}}$
24 $\overline{\text{DACK 1}}$
23 D_5
22 D_6
21 D_7



Pins	Functions
D_7 - D_0	Data Bus
A_0 - A_7	Address Bus
$\overline{\text{I/OR}}$	I/O Read
$\overline{\text{I/OW}}$	I/O Write
$\overline{\text{MEMR}}$	Memory Read
$\overline{\text{MEMW}}$	Memory Write
CLK	Clock Input
RESET	Reset Input
READY	Ready
HRQ	Hold Request (to 8080A)
HLDA	Hold Acknowledge(from 8080A)
AEN	Address Enable
ADSTB	Address Strobe
TC	Terminal Count
MARK	Modulo 128 Mask
DRQ 3 - DRQ 0	DMA Request Input
$\overline{\text{DACK 3}}$ - $\overline{\text{DACK 0}}$	DMA Acknowledge Out
$\overline{\text{CS}}$	Chip Select
V_{CC}	+5 volts
GND	Ground

Need for 8212 and signal ADSTB

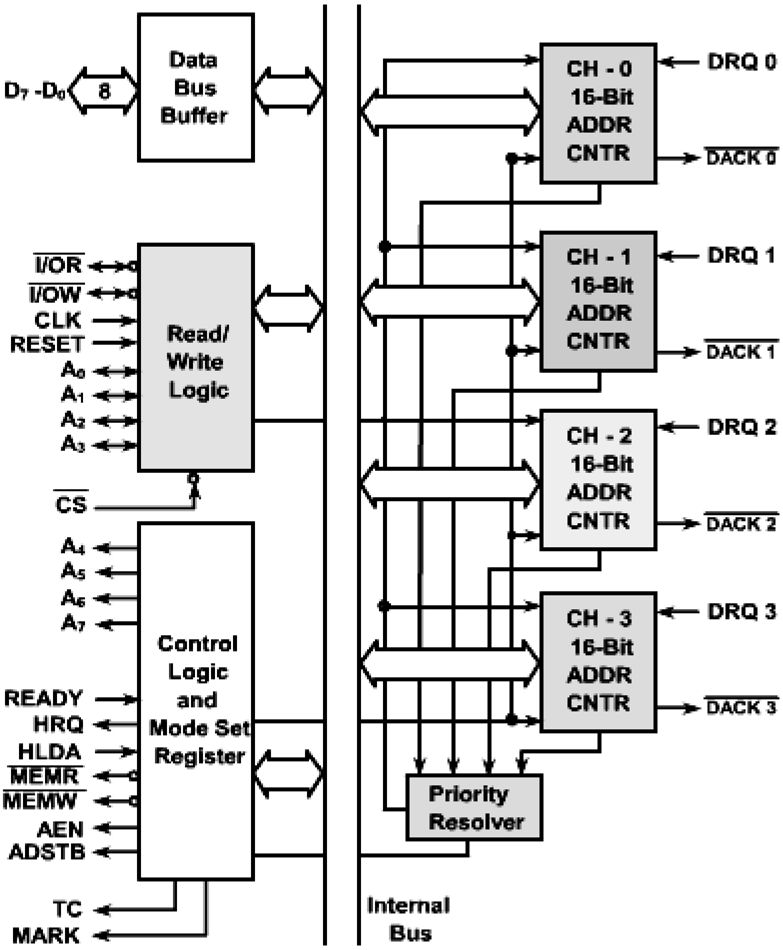
- The 8237 has eight address line, but requires 16 address lines to address a memory location.
- The additional eight lines are generated by using the signal ADSTB to strobe a high-order memory address into 8212 from the data bus.
- A latch, such as the 74LS373 can replace the 5212.

Signal AEN (address enable)

- The AEN output signal is used to disable the system data bus and control bus.
- The signal is necessary to switch the 8237 from the slave mode to the master mode.

Block diagram of INTEL IC 8237

The block diagram of Intel 8237 DMA controller is shown in figure.



- The INTEL 8237 DMA controller is a 40 pin programmable device.
- The 8237 has four independent DMA channels.
- One 8237 can provide DMA transfers to four external devices.
- Data transfers begin with DMA request lines DREQ 0 - DREQ 3.
- DREQ 0 has the highest priority and DREQ 3 has the lowest priority.
- DACK 0 – DACK 3 signals are used to acknowledge a DMA channel request.
- After being initialized by the MPU, the 8237 takes control of the bus in order to perform the DMA operation.
- Data bits are then transferred between a peripheral or external device without involving the microprocessor.
- Data bus buffer
- The data bus lines are bi-directional three state signals connected to the system data bus.
- The outputs are enabled in the program condition during the I/O read to output the contents of an address register, a status register, the temporary register or a word count register to the CPU.
- The outputs are disabled and the inputs are read during an I/O write cycle when the CPU is programming the 8237A control registers.
- During DMA cycles the most significant 8 bits of the address are output on to the data bus to be stored into an external latch by ADSTB.

- In memory to memory operations, data from memory comes into the 8237A on the data bus during the read from memory transfer.
- In the write to memory transfer, the data bus outputs place the data into the new memory location.

Read / write logic

- The 8237 has eight address lines, but requires 16 address lines to address a memory location.
- The additional eight lines are generated by using the signal ADSTB to strobe a high-order memory address into 8212 from the data bus.
- A latch, such as the 74LS373 can replace the 8212.

I/OR (I/O Read)

- I/O read is a bi-directional active low three state line.
- In the idle cycle, it is an input control signal used by the CPU to read the control registers.
- In the active cycle, it is an output control signal used by the 8237A to access data from a peripheral during DMA write transfer.

I/OW (I/O write)

- I/O write is a bi-directional active low three state line.
- In the idle cycle, it is an input control signal used by the CPU to load information into the 8237A.
- In the active cycle, it is an output control signal used by the 8237A to load data to the peripheral during a DMA read transfer.

CLK (Clock input)

- Clock input controls the internal operations of the 8237A and its rate of data transfers.
- The input may be driven at up to 3 MHZ for the standard 8237A and up to 5 MHZ for the 8237A-5.

RESET (Reset)

- Reset is an active high input which clears the command, status, request and temporary registers.
- It also clears the first/last flip/flop and sets the mask register.
- Following a reset the device is in the idle cycle.

A0-A3 (Address)

- The four least significant address lines are bi-directional three-state signals.
- In the idle cycle, they are inputs and are used by the 8237A to address the control register to be loaded or read.

- In the active cycle, they are outputs and provide the lower 4 bits of the output address.

CS (Chip select)

- The chip select is an active low input used to select the 8237A as an I/O device during the idle cycle.
- This allows CPU communication on the data bus.

Control logic and mode set registers

A4-A7 (Address)

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

READY (Ready)

- Ready is an input used to extend the memory read and write pulses from the 8237A to accommodate slow memories or I/O peripheral devices.

MEMW (Memory write)

The memory write is an active low three-state output used to write data to the selected memory location during a DMA read or memory-to-memory transfer.

AEN (Address enable)

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

ADSTB (Address strobe)

- The active high, address strobe is used to strobe the upper address byte into an external latch.

DREQ 0 – DREQ 3 (DMA request)

- The DMA request lines are individual asynchronous channel request inputs used by peripheral circuits to obtain DMA service.
- In fixed priority, DREQ 0 has the highest priority and DREQ 3 has the lowest priority.
- A request is generated by activating the DREQ line of a channel.
- DACK will acknowledge the recognition of DREQ signal.
- Polarity of DREQ is programmable.

- Reset initializes these lines to activate high.
- DREQ must be maintained until the corresponding DACK goes active.

DACK 0 – DACK 3 (DMA acknowledge)

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

Registers in 8237

The 8237 has number of internal registers. These registers are used to control the DMA cycle operation.

1. Current address register

- Each channel contains a 16 bit current address register.
- This register holds the address of the data to be transferred.
- This address can be automatically incremented or decremented after each transfer.

2. Current word register

- Each channel has a 16 bit word count register.
- This register is used to determine the number of transfers to be performed.
- It is incremented after each byte transfer.

3. Base address register

- This 16 bit register stores the starting address of the current address register.

4. Base word register

- This 16 bit register stores the starting address of the current word register.

5. Command register

- This 8 bit register is used to program and control the 8237.
- It is used to initialize the device.

6. Mode register

- Each channel has a 8 bit mode register to define its mode of operation.

7. Request register

- This 4 bit register is used to request a DMA transfer by software.

8. Mask register

- Each channel has mask register bit that is used to disable incoming DREQ signals.

9. Temporary register

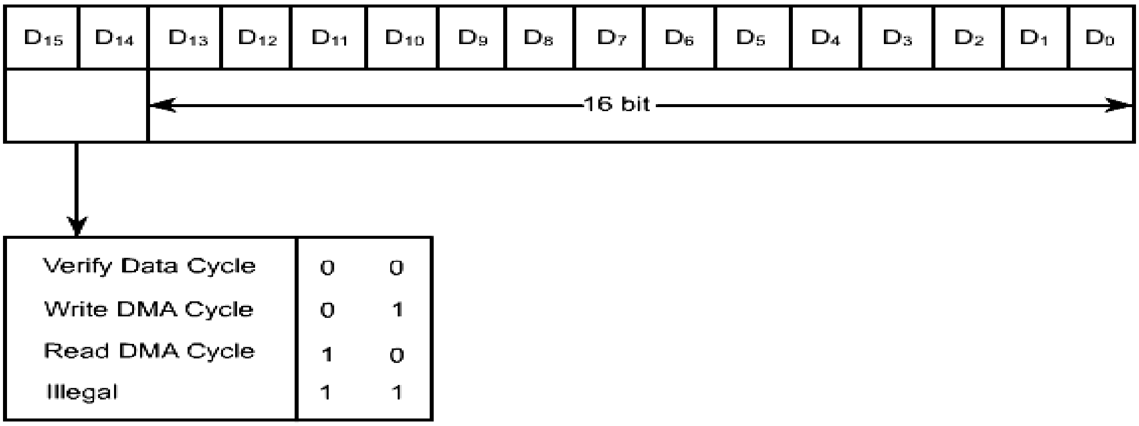
- This is an 8 bit register.
- It is used to hold the data during a memory-to-memory transfer.

10. Status register

- This is an 8 bit register.
- It is used for the microprocessor to read the present status of the 8237.

DMA channels

- The 8237 has four identical channels, each with two signals:
  - DRQ (DMA request)
  - DACK (DMA acknowledge)
- Each channel has two 16 bit register, one for the memory address where data transfer should begin, and the second for a 14 bit count.
- Bits D15 and D14 of the count register specify the DMA function such as write, read, or verify as shown in figure.
- In 8237 two additional 8 nit register are available to enable the DMA channels and to read the status of DMA channels.
- They are the control register, called the mode set register, shown in figure, and the status register shown in figure.
- The control word in the mode set register enables or disables channels and determines other functions.
- The port addresses of each register are determined by four address lines, as shown in figure and chip select logic.

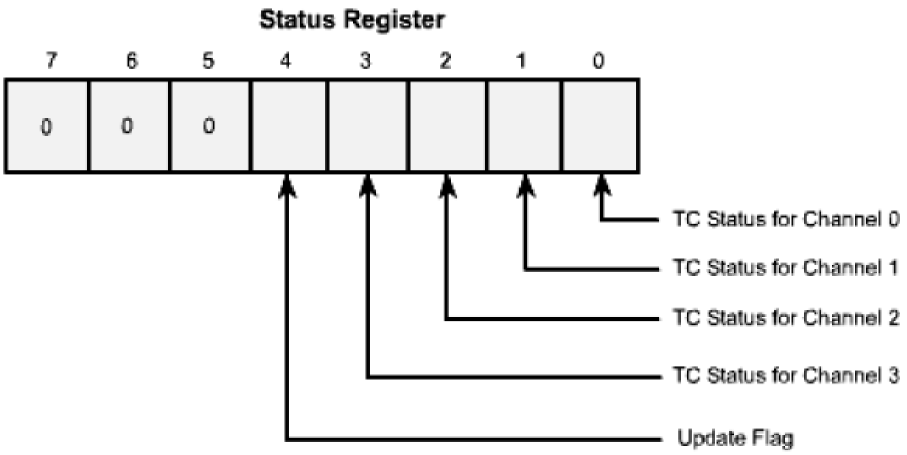


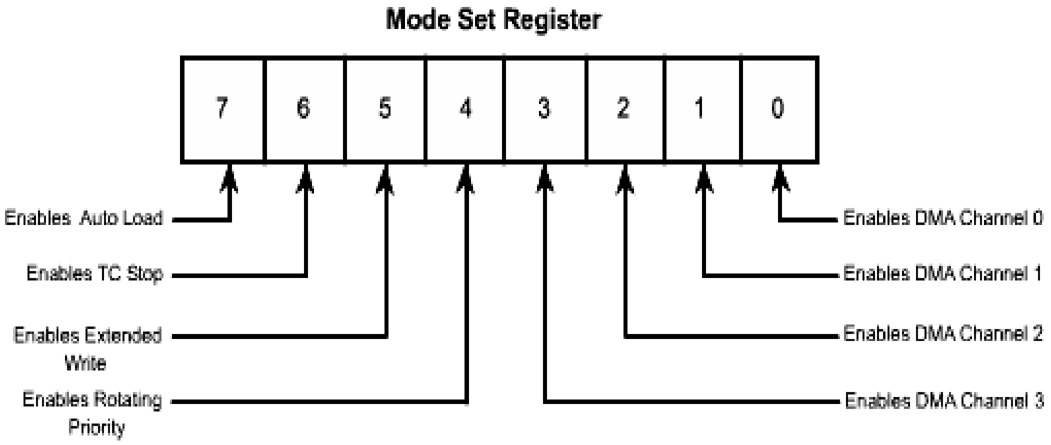
In 8237 two additional 8-bit registers are available to enable the DMA channels and to read the status of DMA channels.

They are the control register, called the mode set registers, shown in figure, and the status register shown in figure.

- The control word in the mode set register enables or disables channels and determines other functions.
- The port addresses of each register are determined by four address lines, as shown in figure and chip select logic.

A3	A2	A1	A0	
0	0	1	0	Selects channel 0 DMA address
0	0	0	1	Selects channel 0 terminate count address
1	0	0	0	Selects mode/status





Operation of DMA controller

- The 8237 DMA controller operates in two major cycles. They are

  i. The idle cycle.

  ii. The active cycle.

- When the 8237 is in the idle cycle, no external devices are requesting a DMA transfer.
- In this cycle, the 8237 samples the DREQ lines every clock cycle to determine if any of the four channels are requesting the service.
- When any one of the DMA request lines DREQ 0 – DREQ 3 becomes active, the 8237 will enter into the active cycle.
- First, the DMA controller requests a HOLD from the microprocessor in response to a DREQ (DMA request) from an external device.
- Next, when the microprocessor recognizes the request and is ready, it outputs a grant signal on the same signal line back to the DMA controller.
- The DMA controller can now use the buses to transfer data between memory and an external device
- The DMA controller now requests external device with a DACK signal.
- This indicates the start of a DMA transfer.
- After completing the data transfer by the DMA, it outputs an EOP, which indicates the completion of the DMA transfer.
- Then the control buses are returned to the microprocessor.

The 8237 in its active mode can perform a DMA transfer in any one of the following four modes:

Single transfer mode

- In this mode, 8237 transfers one byte of data each time the request is active.

Block transfer mode

- In this mode, 8237 transfer a block of data.
- The 8237 is programmed with the starting address of the data and the number of bytes to be transferred.
- The transfer is continued until the word count register in the 8237 reaches its final count or an EOP is received.

Demand transfer mode

- In this mode, the 8237 is programmed to continue making transfers until an EOP is received or until a DREQ from an external device goes inactive.
- In this mode, one byte of data is transferred for each demand (DREQ) received.
- Cascade transfer mode
- This mode is used to cascade more than one 8237 together of a system expansion.
- To use the DMA properly the programmer must first access the internal register of the 8237.
- The command register must be loaded with a command word.
- This command word defines the initial conditions, mode of operation, and the type of operation.

- Next, depending on the mode and the type of operation, the internal registers must be loaded with information defining the starting address of the data in memory, the number of bytes to be transferred, DMA channel number and so on.
- Finally, the DMA channel's request signal must be enabled.
- Placing control words does initialization and programming of the 8237.
- The OUT instruction is used to send control words to the internal registers.

DMA execution

The process of data transfer from the external devices to the system memory by the DMA controller can be classified under two modes:

- The slave mode
- The master mode

Slave mode

In the slave mode, the DMA controller is treated as a peripheral, using the following steps:

- Chip select is used to select the DMA.
- The microprocessor writes the command mode and terminal count in channel register by accessing the register through A0 – A3 and through the control signals IOR and IOW.

In this mode, the signals shown in the read / write logic block are used; the address line A7 – A4 and the control signals MEMR, MEMW from the control logic block are in tri-state. The other signals are not being used.

Master mode

After the initialization, the 8237 in master mode checks for a DMA request. The steps in data transfer can be listed as follows:

- When the peripheral is ready for data transfer, it sends a high signal to DRQ.
- When the DRQ has been received and the channel enabled, the control logic sets HRQ high.
- In the next cycle, the microprocessor releases the buses and sends the HLDA signal to the 8237.
- After receiving the HLDA signal, the control logic generates DACK (DMA acknowledge) and sends the acknowledgement to the peripheral.
- Meanwhile, the 8237 enables the signal AEN (address enable)
- AEN disables the microprocessor de-multiplexed address bus A7 – A0.
- The entire bus, A7 – A0, OF THE 8237 becomes output.
- The low-order byte of the memory location is placed on the A7-A0 of the 8237.
- When the AEN signal is high, the ADSTB (address strobe) signal goes high.

- This signal places the high-order byte of the memory location, generated by the lower 8212, on address bus A15-A8.
- Data transfer continues until the count reaches zero.

7. INTEL 8253 PROGRAMMABLE INTERVAL TIMER

- The 8253/54 solves one of the most common problem in any microcomputer system- the generation of accurate time delays under software control.
- Instead of setting up timing loops in system software, the programmer configures the 8253/54 with the desired quantity, then upon command the 8253/54 will count out the delay and interrupt the CPU when it has completed its tasks.

INTRODUCTION:

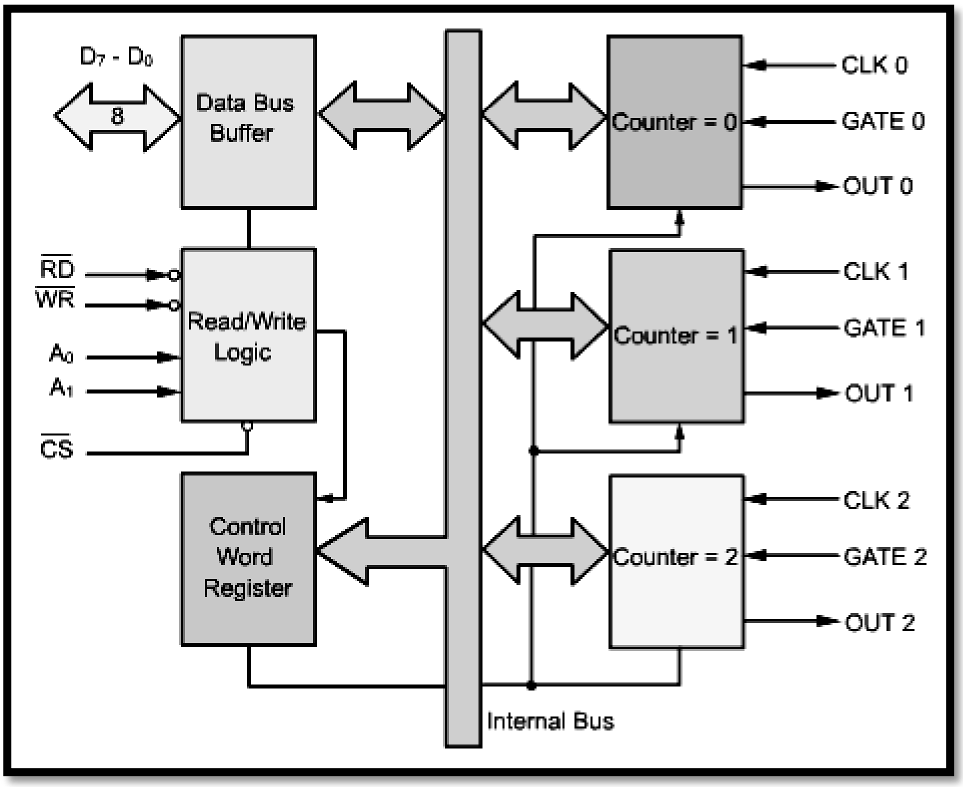
- The 8253 is a programmable interval timer/counter designed for use with intel microcomputer systems.
- It is a general purpose, multi-timing element that can be treated as an array of i/o ports in the system software.
- The 8253 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control.
- Instead of setting up timing loops in software, the programmer configures the 8253 to match his requirements and programs one of the counters for the desired delay.
- After the desired delay, the 8253 will interrupt the cpu. Software overhead is minimal and variable length delays can easily be accommodated.
- The 8253/54 includes three identical 16 bit counters that can operate independently.
- To operate a counter, a 16 bit count is loaded in its register and, on command, it begins to decrement the count until it reaches 0.
- At the end of the count, it generates a pulse that can be used to interrupt the CPU.
- The counter can count either in binary or BCD.
- In addition, a count can be read by the CPU while the counter is decrementing.
- In this section, we are going to study two timer ICs 8253 and 8254 .
- The 8254 is a superset of 8253.

FEATURES:

- Three independent 16-bit down counters.
- 8254 can handle inputs from DC to 10 MHz (5 MHz 8254-5 8 MHz 8254 10 MHz 8254-2) where as 8253 can operate up to 2.6 MHz.
- Three counters are identical, pre-settable and can be programmed for either binary or BCD count.
- Counter can be programmed in six different modes.
- Compatible with all Intel and most other microprocessors.

- 8254 has powerful command called READ BACK command which allows the user to check the count value, programmed mode and current mode and current status of the counter.

Block diagram:



DATA BUS BUFFER:

- This tri-state, bi-directional, 8-bit buffer is used to interface the 8253/54 to the system data bus.
- The Data bus buffer has three basic functions.

1. Programming the 8253/54 in various modes
2. Loading the count registers.
3. Reading the count values.

READ/WRITE LOGIC:

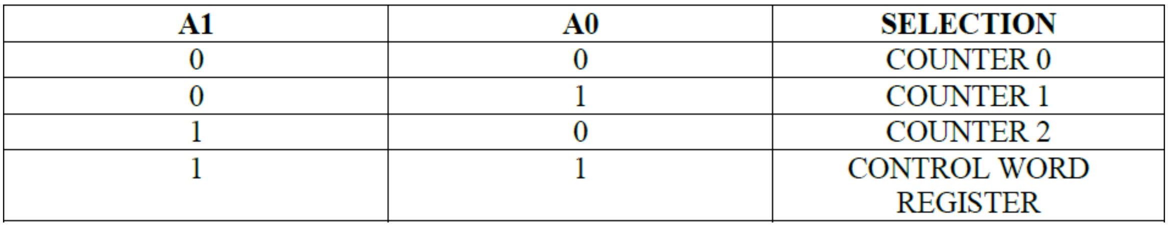
- The Read/Write logic has five signals: ¯R¯D, ¯W¯R, ¯C¯S and the address lines A0 and A1.
- In peripheral I/O mode, the RD and WR signals are connected to IOR and IOW, respectively
- In memory-mapped I/O, these are connected to MEMR and MEMW. Address lines A0 and A1 of the CPU are usually connected to lines A0 and A1 of the CPU are

usually connected to lines A0 and A1 of the 8253/54, and CS is tied to a decoded address.

- The control word register and counters are selected according to the signals on lines A0 and A1.

Control word register:

- This register is accessed when lines A0 and A1 are at logic 1.
- It is used to a write a command word which specifies the counter to be used (binary or BCD), its mode and either a read or write operation.



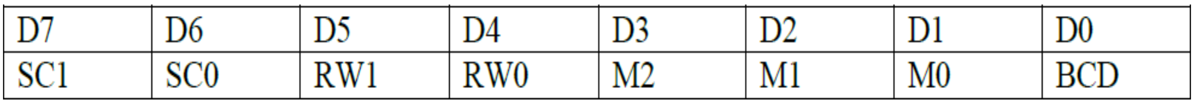
A1	A0	SELECTION
0	0	COUNTER 0
0	1	COUNTER 1
1	0	COUNTER 2
1	1	CONTROL WORD REGISTER

Counter:

- These three functional blocks are identical in operation.
- Each counter consists of a single, 16 bit, pre-settable, down counter.
- The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of modes stored in the control word register.
- The counters are fully independent.
- The programmer can read the contents of the three counters without disturbing the actual count in process.

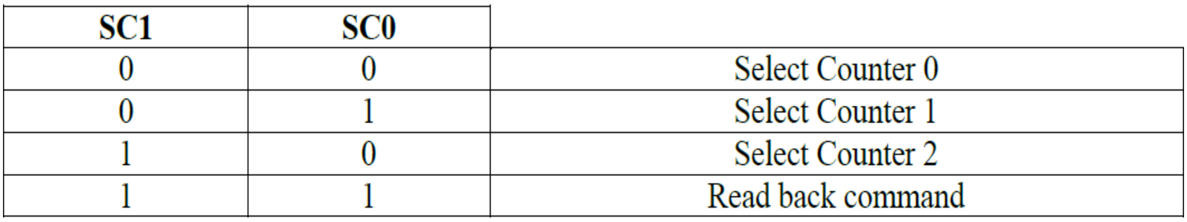
Operational description:

- The complete functional definition of the 8254 is programmed by the system software.
- Once programmed the 8254 is ready to perform whatever timing tasks it is assigned to accomplish.



D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

- The register is accessed when lines A0 and A1 are at logic 1.
- It is used to write a command word which specifies the counter to tb used.
- Its mode & either a read or write operation.

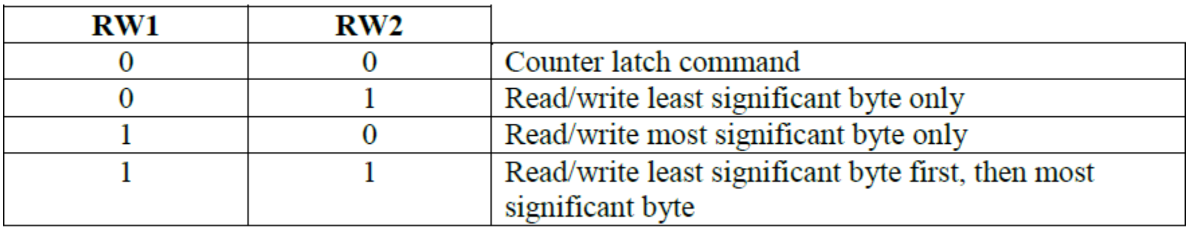


SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read back command

The bits for D7 and D6 of the control word select anyone of the three counters or the read back command.

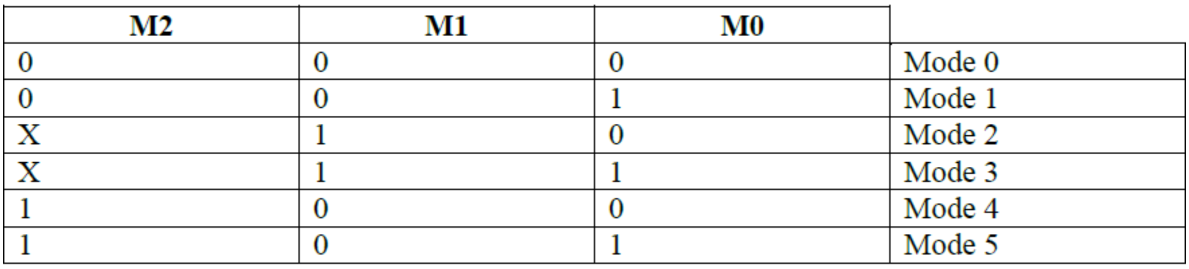
(SC-select counter D7 to D6)

The bits D5 and D4 decide the various read/write operation.



RW1	RW2	
0	0	Counter latch command
0	1	Read/write least significant byte only
1	0	Read/write most significant byte only
1	1	Read/write least significant byte first, then most significant byte

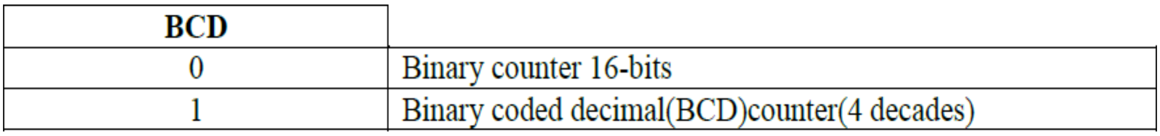
The bits D3, D2 & D1 decide the mode in which the 8254 has to operate the six modes.



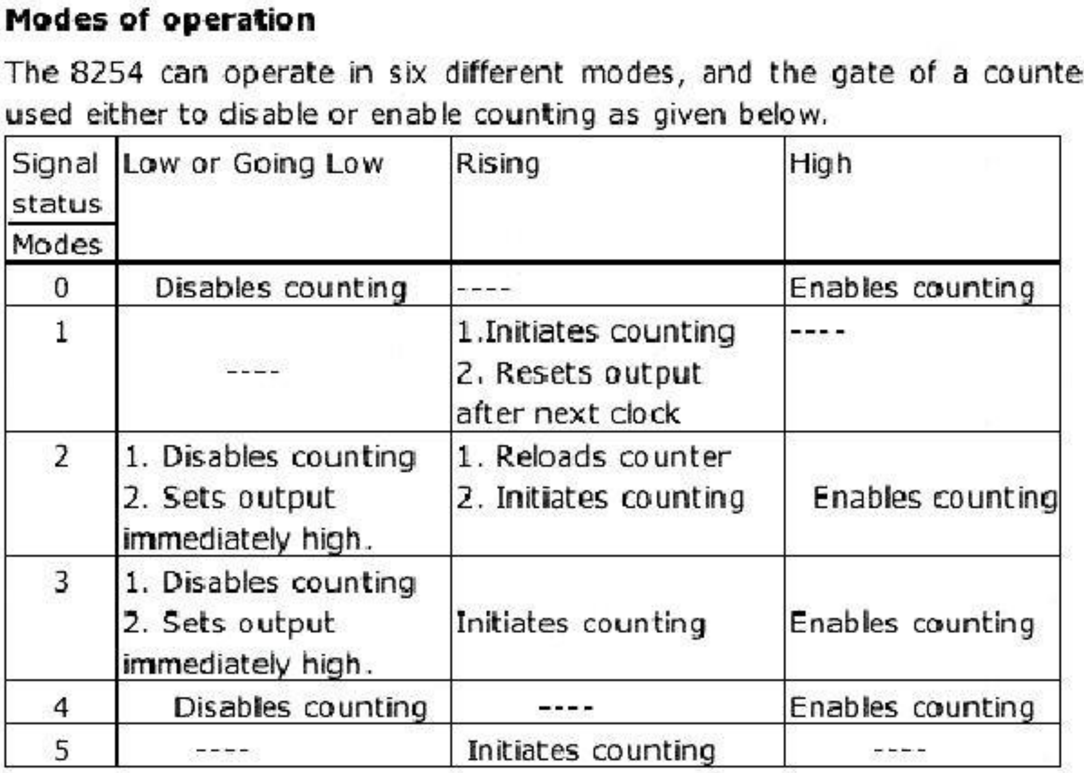
M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

M-Mode(D3,D2 &D1)

The bits D0 of the control word decides whether counter should count in BCD or Binary.



BCD	
0	Binary counter 16-bits
1	Binary coded decimal(BCD)counter(4 decades)



Modes of operation

The 8254 can operate in six different modes, and the gate of a counter is used either to disable or enable counting as given below.

Signal status Modes	Low or Going Low	Rising	High
0	Disables counting	----	Enables counting
1	----	1.Initiates counting 2. Resets output after next clock	----
2	1. Disables counting 2. Sets output immediately high.	1. Reloads counter 2. Initiates counting	Enables counting
3	1. Disables counting 2. Sets output immediately high.	Initiates counting	Enables counting
4	Disables counting	----	Enables counting
5	----	Initiates counting	----

<u>Mode Definition:</u>

READ AND WRITE OPERATIONS

The 8254 can be programmed to provide various types of output through write operations, or to check a count while counting through read operations. The details of these operations are given below.

READ OPERATIONS

In some applications especially in event counters, it is necessary to read the values of the count in progress. This can be done by either of two methods.

- One method involves reading a count after inhibiting (stopping) the counter to be read. It is known as reading by halting the count.
- The second method involves reading a count while the count is in progress (known as reading on the fly). It is known as reading while counting.

In first method, counting is stopped (or inhibited) by controlling the gate input or the clock input of the selected counter, and two I/O read operations are performed by the MPU. The first I/O operation reads the low order byte, and the second I/O operation reads the higher order byte.

In the second method, an appropriate control word is written into the control register to latch a count in the output latch, and two I/O read operations are performed by the MPU.

WRITE OPERATION:

To initialize a counter, the following steps are necessary.

- Write a control word into the control register.
- Load the low order byte of a count in the counter register.
- Load the high order byte of a count in the counter register.

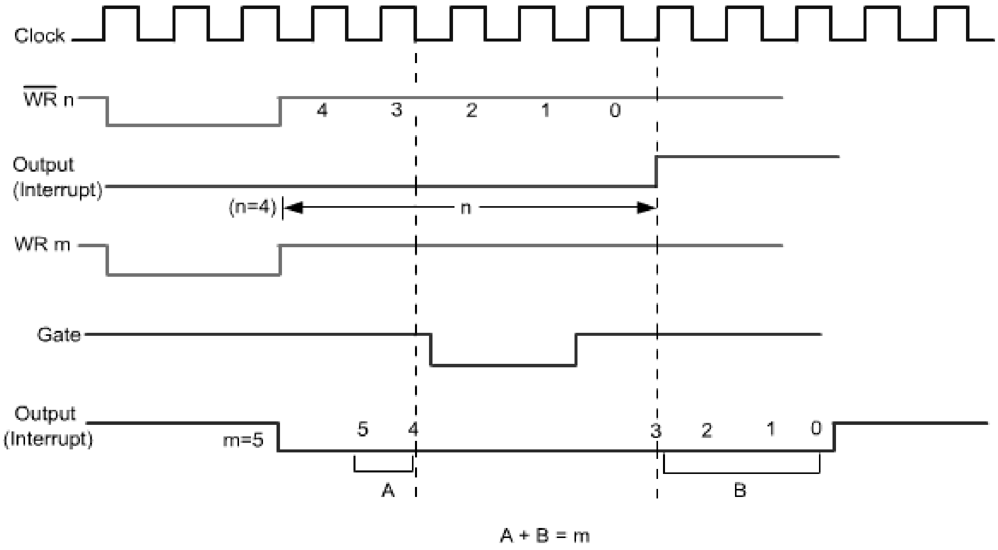
With a clock and an appropriate gate signal to one of the counters, the above steps should start the counter and provide appropriate output according to the control word.

MODES OF OPERATION:

As mentioned earlier, the 8254 can operate in six different modes; we will describe briefly various modes of the 8254.

MODE 0: Interrupt on terminal count

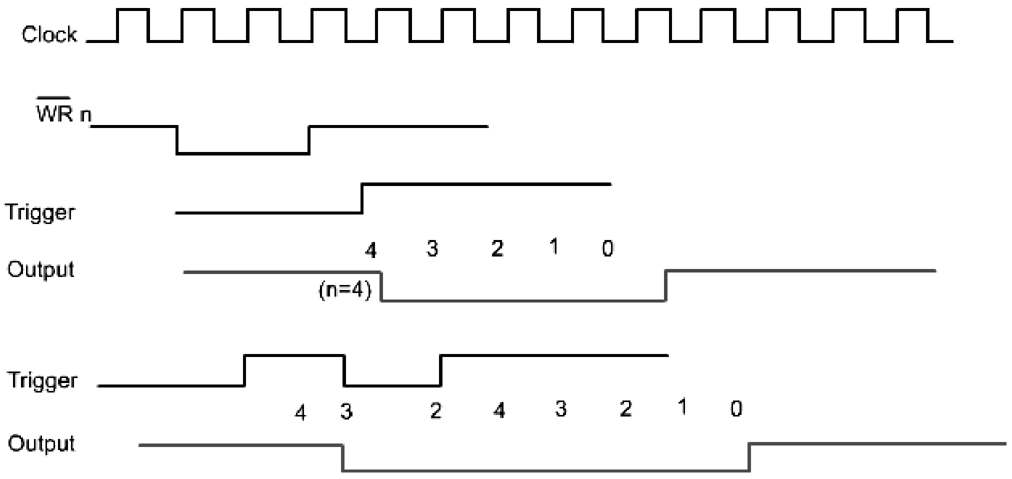
- The output goes low on setting the mode, and goes high after the desired count.
- In this mode, initially the OUT is low.
- Once count is loaded in the register, the counter is decremented every cycle, and when the count reaches zero, the OUT goes high.
- This can be used as an interrupt.



- The OUT remains high until a new count or a command word is loaded.
- Figure shows that the counting (m=s) is temporarily stopped when the gate is disabled (G=0), and continued again when the gate is at login 1.

MODE 1: Hardware-Retriggerable One-Shot

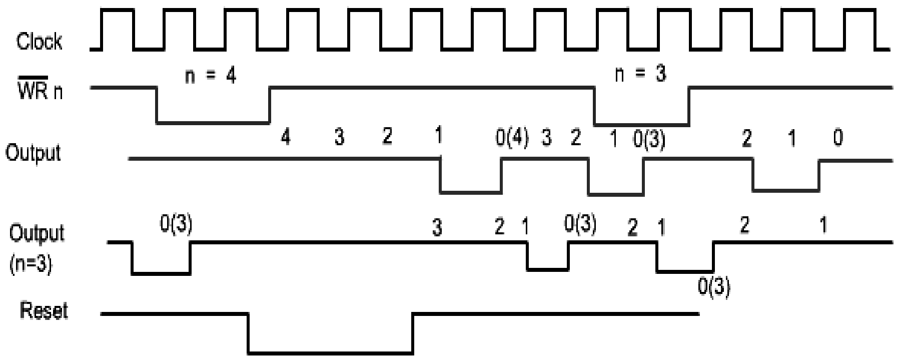
- The output goes low on a gate input, and goes high on terminal count.



- In this mode, the OUT is initially high.
- When the gate is triggered, the out goes low, and at the end of the count, the OUT goes high again, thus generating a one-shot pulse.

MODE 2: Rate generator

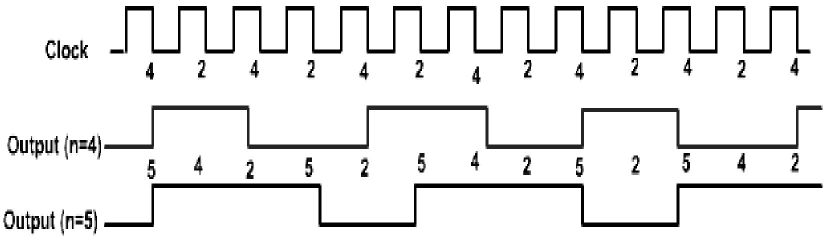
- It is equivalent to a ‘clock pulse divide by n’ counter.
- The output pulse frequency is 1/n of the input pulse frequency ,where n is the count number .
- This mode is used to generate pulse equal to the clock period at a given interval
- When a count is loaded, the OUT stays high until the count reaches 1, and then the OUT goes low for one clock Period.



- The count is reloaded automatically, and the pulse is generated continuously.
- The count = 1 is illegal in this mode.

MODE 3: Square wave generator

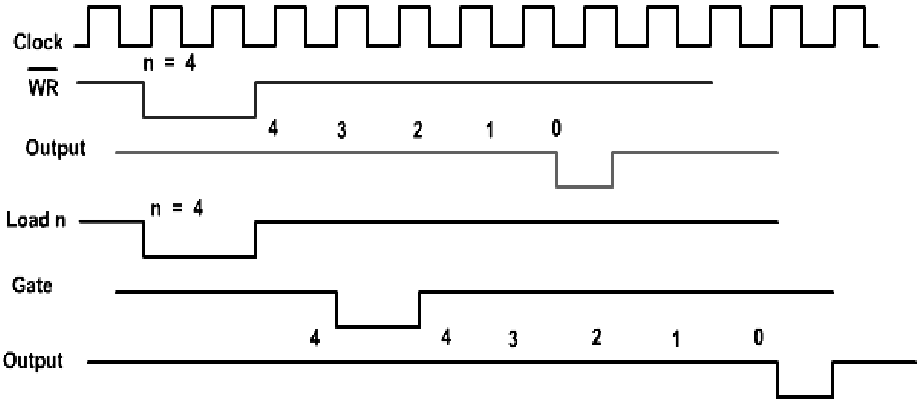
- This is similar to mode 2 with difference in minor operating details.
- In this mode, when a count is loaded, the OUT is high.
- The count is decremented by two at every clock cycle, and when it reaches zero, the OUT goes low, and the count is reloaded again.



- This is repeated continuously; thus, a continuous square wave with period equal to the period of the count is generated.
- In other word, the frequency of the square wave is equal to the frequency of the clock divided by the count.
- If the count (N) is odd the pulse stays high for (N+1)/2 clock cycles and stays low for (N-1)/2 clock cycles.

MODE 4: Software-triggered strobe

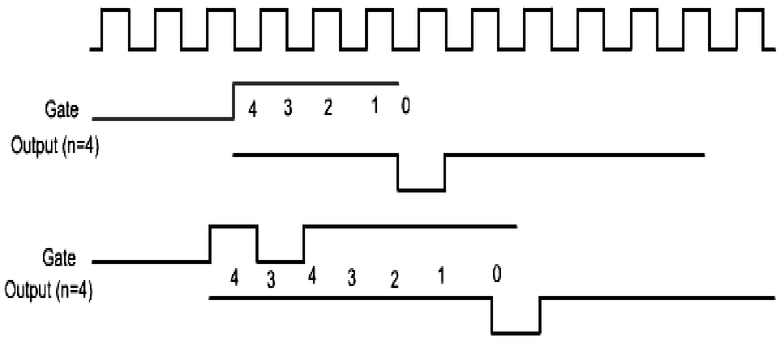
- The output goes high on setting the mode
- After terminal count, the output goes low for one clock period and then goes high again



- In this mode, the OUT is initially high; it goes low for one clock period at the end of the count.
- The count must be reloaded for subsequent outputs.

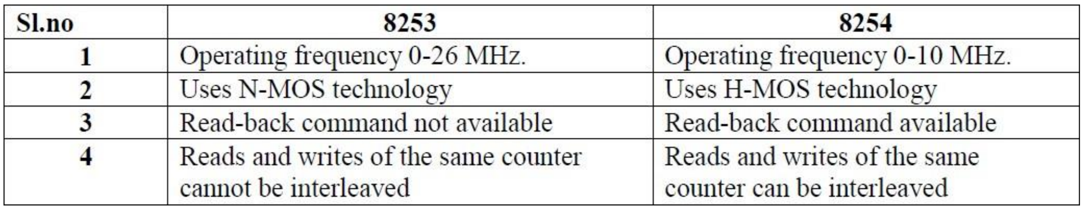
MODE 5: Hardware-triggered strobe

- This is similar to mode 4, but a trigger at the gate initiates the counting.
- This mode is similar to mode 4, except that it is triggered by the rising pulse at the gate.



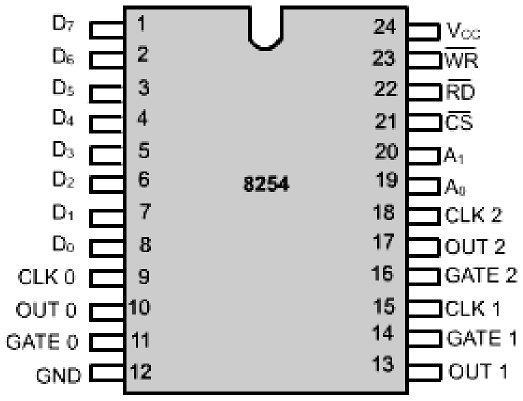
- Initially, the OUT is high, and when the gate pulse is triggered from low to high, the count begins.
- At the end of the count, the OUT goes low for one clock period.

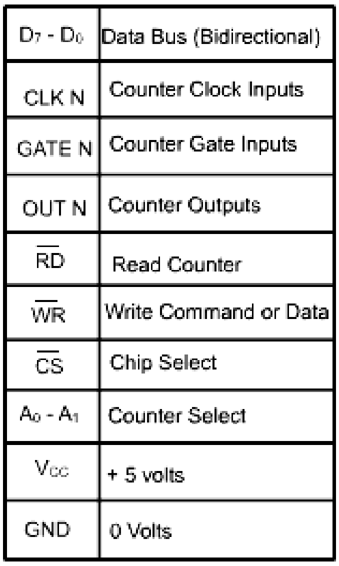
Difference between 8253A and 8254A:



Sl.no	8253	8254
1	Operating frequency 0-26 MHz.	Operating frequency 0-10 MHz.
2	Uses N-MOS technology	Uses H-MOS technology
3	Read-back command not available	Read-back command available
4	Reads and writes of the same counter cannot be interleaved	Reads and writes of the same counter can be interleaved

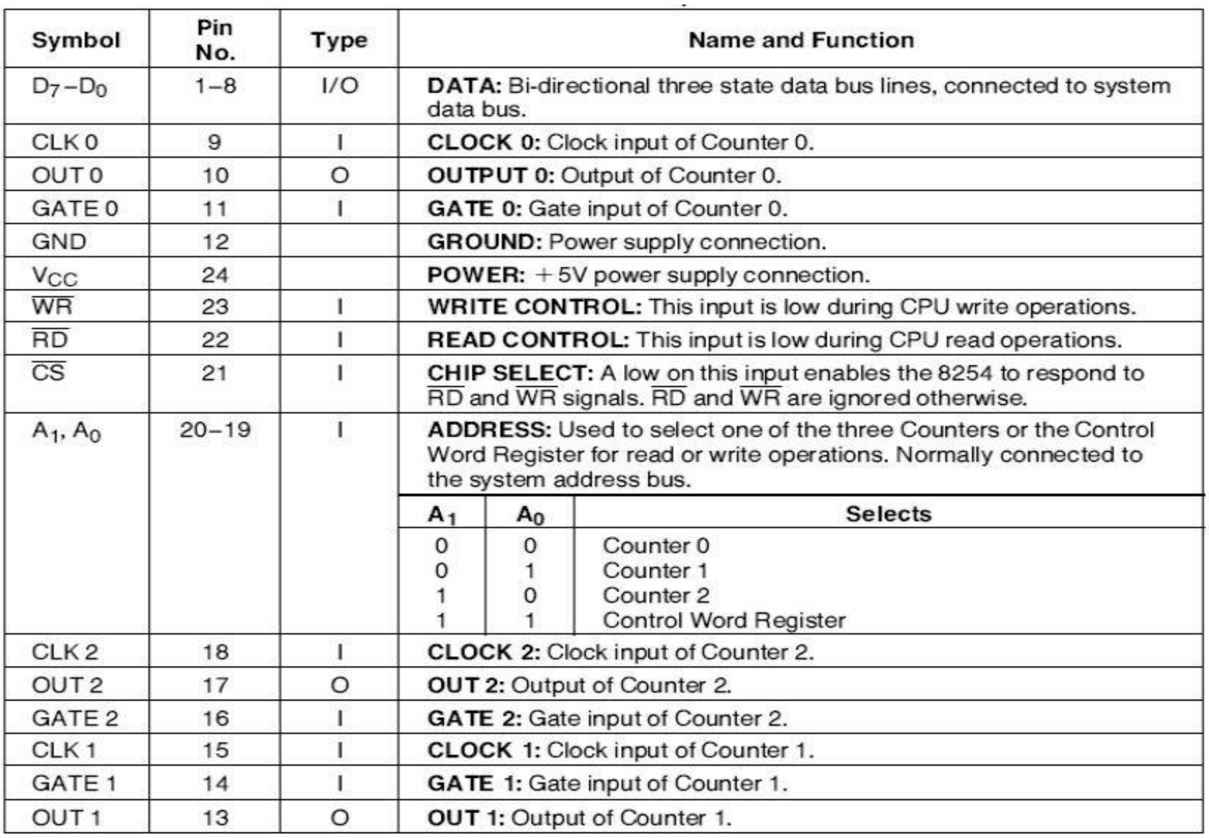
PIN DIAGRAM:





D_7 - D_0	Data Bus (Bidirectional)
CLK N	Counter Clock Inputs
GATE N	Counter Gate Inputs
OUT N	Counter Outputs
$\overline{RD}$	Read Counter
$\overline{WR}$	Write Command or Data
$\overline{CS}$	Chip Select
A_0 - A_1	Counter Select
V_{CC}	+ 5 volts
GND	0 Volts

Pin diagram details:



Symbol	Pin No.	Type	Name and Function
D_7-D_0	1–8	I/O	**DATA:** Bi-directional three state data bus lines, connected to system data bus.
CLK 0	9	I	**CLOCK 0:** Clock input of Counter 0.
OUT 0	10	O	**OUTPUT 0:** Output of Counter 0.
GATE 0	11	I	**GATE 0:** Gate input of Counter 0.
GND	12		**GROUND:** Power supply connection.
V_{CC}	24		**POWER:** +5V power supply connection.
$\overline{WR}$	23	I	**WRITE CONTROL:** This input is low during CPU write operations.
$\overline{RD}$	22	I	**READ CONTROL:** This input is low during CPU read operations.
$\overline{CS}$	21	I	**CHIP SELECT:** A low on this input enables the 8254 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
A_1, A_0	20–19	I	**ADDRESS:** Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus. **A_1 A_0 Selects** 0 0 Counter 0 0 1 Counter 1 1 0 Counter 2 1 1 Control Word Register
CLK 2	18	I	**CLOCK 2:** Clock input of Counter 2.
OUT 2	17	O	**OUT 2:** Output of Counter 2.
GATE 2	16	I	**GATE 2:** Gate input of Counter 2.
CLK 1	15	I	**CLOCK 1:** Clock input of Counter 1.
GATE 1	14	I	**GATE 1:** Gate input of Counter 1.
OUT 1	13	O	**OUT 1:** Output of Counter 1.

UNIT IV

Intel 8086 Microprocessor: Introduction-Intel 8086 Hardware – Pin description – External memory Addressing – Bus cycles – Interrupt Processing. Addressing modes - Instruction set – Assembler Directives.

1. What are the difference between 8085 & 8086?***

8085	8086
8-bit microprocessor	16-bit microprocessor
2^{16} memory locations	2^{20} memory locations
Sequential facility	Pipelined architecture available
Low speed	High speed
It does not have much operational instructions when compared to 8086	It allows to have much large set of operation and instruction.

2. What are the functional units available in 8086?

The internal functions of the 8086 processor are portioned logically into two processing units.
1. Bus Interface Unit (BIU)
2. Execution unit (EU)

- BIU & EU function independently.

The **BIU** contains	The **EU** contains
1.Segment registers 2.Instruction pointer 3. Instruction queue.	1.ALU 2.General purpose registers 3. Index registers 4. Pointers 5.Flag register

3. What **are the functions of BIU and EU?*****

- The BIU and EU function independently.

Bus Interface Unit (BIU):

- The BIU interfaces the 8086 to the outside of the world.
- The BIU fetches instruction, reads data from memory and ports and write data to the memory and I/O ports.

Execution unit (EU):

- EU receives program instruction codes and data from the BIU, executes these instructions and stores the result in the general registers or output them through the BIU.

4. How the instructions are classified in 8086?**

Instructions of 8086 are classified into **six** groups.

1. Data transfer instructions
2. Arithmetic instructions
3. Bit manipulation instructions
4. String instructions
5. Program execution transfer instructions
6. Processor control instructions

5. What are the types of addressing modes in 8086?***

The method of specifying the data to be operated by the instruction is called **Addressing.**

The 8086 has **12** addressing modes:

1. Register Addressing Mode
2. Immediate Addressing Mode]
3. Direct Addressing Mode
4. Register Indirect Addressing Mode
5. Based Addressing Mode
6. Indexed Addressing Mode
7. Based Indexed Addressing Mode
8. String Addressing Mode
9. Direct I/O port Addressing Mode
10. Indirect I/O port Addressing Mode
11. Relative Addressing Mode
12. Implied Addressing Mode.

6. Define segment register? List types of segment in 8086 memory.***

Segment registers are in Bus Interface Unit (BIU) of 8086.

Most of the registers contain data/instruction offset within 64KB memory segment.

There are **four** different 64 KB segments for instructions, stack, data & extra data.

The segment registers are:

1. **C**ode **S**egment(CS)
2. **S**tack **S**egment(SS)
3. **D**ata **S**egment(DS)
4. **E**xtra **S**egment(ES)

7. What is 8086 directives?

An assembler is a program which translates an assembly language program into machine language program.

An assembly language program consists of two types of statements:

Instruction & Directives.

The instructions are translated to machine codes by the assembler, whereas the directives are not translated to machine codes.

Some assembler directives are:

1. Borland Turbo Assembler (TASM)
2. IBM Macro Assembler (MASM)
3. Intel 8086 Macro Assembler (ASM)
4. Microsoft Macro Assembler.

8. What is the need of assembler directives?**

- ❖ An assembler directive is a statement to give direction to the assembler to perform the task of assembly process.
- ❖ The assembler directives control organization ofthe program and provide necessary information to the assembler to understand assembly language program to generate machine codes.
- ❖ They indicate how an operand or a section of a program is to be processed by the assembler.
- ❖ An assembler supports directives to define data, to organize segments, to control procedures, to define macros etc.

9. Give examples for some assembler directives that are specific to 8086 assembly language.***

The general assembler directives are: ASSUME , EXTRN, GROUP, INCLUDE, LABEL, MACRO, ORG, PTR, PROC, PUBLIC, RECORD, SEGMENT, STRUC, EVEN, EQU, END, ENDM, ENDS, ENDP, DT, DQ , DD,DW, DB.

10. Explain ASSUME.

The ASSUME directive enables error-checking for register values.

It is used to inform the assembler the names of the logical segments, which are to be assigned to the different segments used in an assembly language program.

Format:

ASSUMEsegregister:name[[,segregister:name]]…
ASSUME dataregister:type[[,dataregister:type]]…
ASSUME register:**ERROR**[[,register**:ERROR**]]…
ASSUME [[register:]] **NOTHING** [[,register:**NOTHING**]]…

After an ASSUME is put into effect, the assembler watches for changes to the values of the given registers. ERROR generates an error if the register is used. NOTHING removes register error-checking.

Examples:

```
        ASSUME CS : CODE
        ASSUME DS : DATA
```

11. Explain SEGMENT.**

SEGMENT is used to indicate the start of a logical segment. It defines a program segment called *name* having segment attributes *align* (BYTE, WORD, DWORD), *combine* (PUBLIC, STACK), *use* (USE16, USE32, FLAT), and *class.*
The ENDS statement indicates the end of the program.

Format:

```
    name        SEGMENT type (WORD or PUBLIC)
    statements
    name        ENDS
```

Examples:

```
    CODE        SEGMENT    WORD
                .
                .
                .
    CODE        ENDS
```

12. Explain EVEN.

EVEN (Align on Even memory Address):

The EVEN directive tells the assembler to increment the location counter to the next even address if it is not already at an even address.

Format:

```
                EVEN
```

Examples:

```
SALES               DB
EVEN
DATA_ARRAY          DW      100 DUP (?)
```

13. Explain DD, DQ and DT.***

DD (Define Double Word):

It can be used to define data like **DWORD** (4bytes)

Format:

```
    Name of the variable        DD      Initial values
```

Example:

```
        NUMBER                  DD      12345678
```

DQ (Define Quad Word):

It can be used to define data like **QWORD** (8 bytes).

Format:

Name of the variable DQ *Initial values*

Example:

TABLE DQ 1234567812345678

DT (Define Ten Bytes):

It can be used to define data like **TBYTE** (10 bytes).

Format:

Name of the variable DT *Initial values*

Example:

AMOUNT DT 12345678123456781234

14. What is the role of TF and IF flags in 8086?***

TF (Trap Flag)

Setting TF puts the 8086 in the single step mode. In this mode, the 8086 generates an internal interrupt after the execution of each instruction.

IF (Interrupt Flag)

Setting IF causes the 8086 to receive external maskable interrupts through INTR pin. Clearing IF disable these interrupts.

15. What is the function of T and D flags in 8086?

D Flag- String Direction Flag:

It is used to set direction in string operation.

T Flag- Single Step Trap Flag:

It is used for single stepping through a program.

16. What is meant by software interrupt in 8086?

The software interrupt are the program instructions. These instructions are inserted at desired locations in a program. While running a program, if a software interrupt is encountered then the processor executes an interrupt service routine (ISR).

17. Explain the instructions AAA ,AAS,AAM & AAD?***

AAA:

ASCII Adjust for Addition instruction adjusts the binary result of ADD or ADC instruction.If bits 0-3 of AL contain a value greater than 9, or if the Auxiliary carry flag is set, the CPU adds 06 to AL and adds 1 to AH. The bits 4-7 are set to zero.

(AL) ← (AL) + 6
(AH) ← (AH) + 1

(AF) ← 1

Example:

AAA

Before execution

AH	AL
00	0B

After execution

AH	AL
01	01

AAS:

ASCII Adjust for Subtraction instruction adjusts the binary result of a SUB or SBB instruction.

If D_3-D_0 OF AL > 9
(AL) ← (AL) - 6
(AH) ← (AH) - 1
(AF) ← 1

AAM:

ASCII Adjust for Multiplication instruction adjusts the binary results of a MUI instruction. AL is divided by 10($0A_H$) and the quotient is stored in AH. The remainder is stored in AL.

(AH) ← (AL/$0A_H$)
(AL) ← Remainder

AAD:

ASCII Adjust for Division instruction adjusts unpacked BCD dividend in AX before a division operation. AH is multiplied 10($0A_H$) and added to AL. AH is set to zero.

(AL) ← (AH X $0A_H$) + (AL)
(AH) ← 0

18. What is Intra-segment (NEARJMP) Inter segment (FARJMP)?***

A **NEAR-JMP (Intra segment)** is a jump where destination location is in the same code segment. In this case only IP (**I**nstruction **P**ointer) is changed.

IP = IP + signed displacement

A **FAR-JMP (Inter-segment)** is a jump where destination location is from a different segment. In this case both IP (**I**nstruction **P**ointer) and CS (**C**ode **S**egment) are changes as specified in the destination.

19. What is NEAR-CALL&FAR-CALL?

Intra-Segment CALL:

A NEAR-CALL is a call to a procedure which is in the same code segment as the CALL instruction. When 8086 executes a NEAR-CALL instruction, it decrements the stack pointer (SP) by 2 and copies the offset of the next instruction after the CALL on the stack. It loads IP with the offset of the first instruction of the procedure in same segment.

Inter-Segment CALL:

A FAR-CALL is a call to a procedure which is in a different segment from that which contains the CALL instruction. When 8086 executes a FAR-CALL, it decrements the SP by 2 and copies the content of the CS register to the stack. It then decrements SP by 2 again and copies the offset of the instruction after the CALL to the stack. Finally it loads CS with the segment base of the segment which contains the procedure and IP with the offset of the first instruction of the procedure in that segment.

20. How 8086 is organized to facilitate memory read/write operation for addressing external memory?***

The 16 bit members of the 8086 family can load a word from any arbitrary address. The processor fetches the lower order byte of the value from the address specified and the higher order byte from the next consecutive address.

The memory bank is selected by BHE and A_0.

The EVEN memory bank is selected by the address line A_0.

The ODD memory bank is selected by the control signal $\overline{\text{BHE}}$.

Any memory location in the memory bank is selected by the address line A_1 TO A_{19}.

$\overline{\text{BHE}}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to ODD address
1	0	Lower byte from/to EVEN address
1	1	None

Program, data and stack memories occupy the same space. The total addressable memory size is 1 MB. As the most of the processor instructions use 16-bit pointers, the processor can effectively address only 64KB of memory. To access memory outside of 64 KB the CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory.

21. Write about interrupt priority of 8086.

Interrupt	Priority
INT n, INTO, Divide Error	Highest
NMI	↓
INTR	↓
Single step	lowest

Table: the priority of interrupts of 8086.

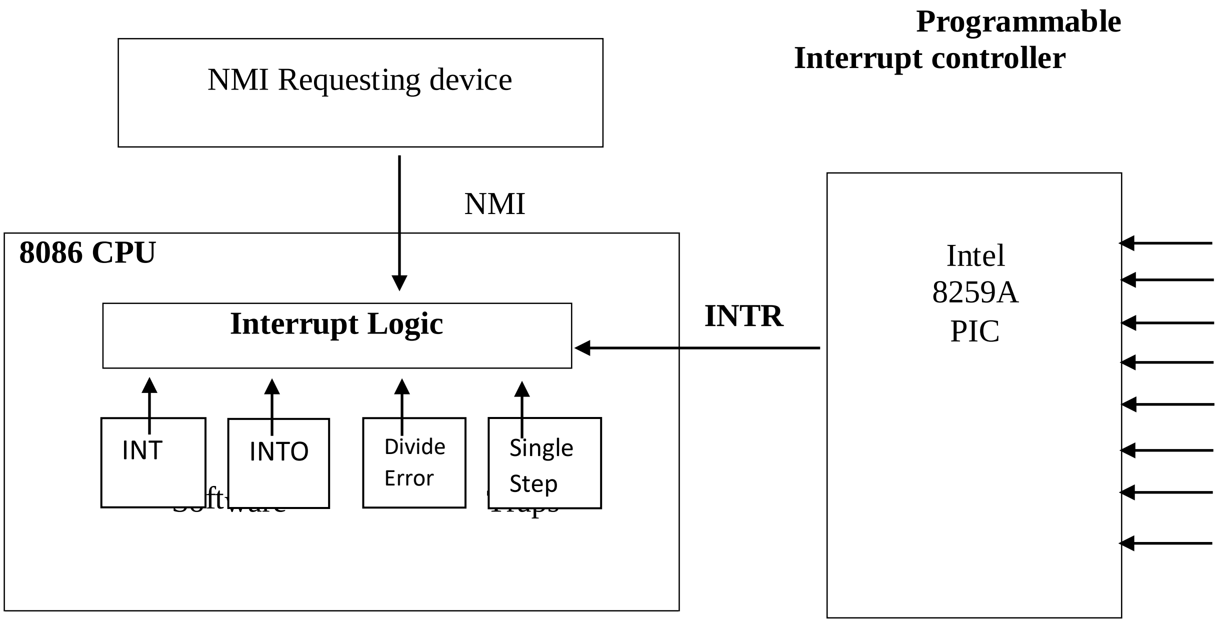
The software interrupts except Single Step interrupt have the highest priority; followed by NMI, followed by INTR. Single step interrupt has the least priority. The 8086 checks for internal interrupts before for any hardware interrupt. Therefore software interrupts have higher priority than hardware interrupt.

22. What are the interrupts in 8086***

1. Hardware Interrupt – External Uses INTR and NMI
2. Software Interrupt – Internal – from INT or INTO
3. Processor Interrupt – Traps and 10 Software Interrupts

External –generated outside the CPU by other hardware.
(INTR, NMI)

Internal –generated within CPU as of an instruction or operation.
(INT, INTO, Divide Error and Single Step)



8086 INTERRUPT CONNECTIONS

23. How the physical address for fetching the next instruction to be executed is obtained in 8086?

The physical address is obtained by appending four zeros to the content present in CS register and then adding the content of IP register with the above value.

24. What is pipelining?***

The instruction queue is a First-In-First-Out (FIFO) group of registers where 6 bytes of instruction code is pre-fetched from memory ahead of time. It is being done to speed up program execution by overlapping instruction fetch and execution. This mechanism is known as **PIPELINING.**

- Fetching the next instruction while the current instruction executes is called **PIPELINING.**

Unit IV

Intel 8086 Microprocessor: Introduction-Intel 8086 Hardware – Pin description – External memory Addressing – Bus cycles – Interrupt Processing. Addressing modes - Instruction set – Assembler Directives.

INTEL 8086 MICROPROCESSOR

1. ARCHITECTURE OF 8086:

The internal architecture of 8086 is divided into two separate units. They are

(i) Bus Interface Unit (BIU)

(ii) Execution Unit (EU)

The two units function independently.

The BIU is needed to fetch instruction, read operands, and write results. The execution unit is used to execute instructions that have already been fetched by the BIU.

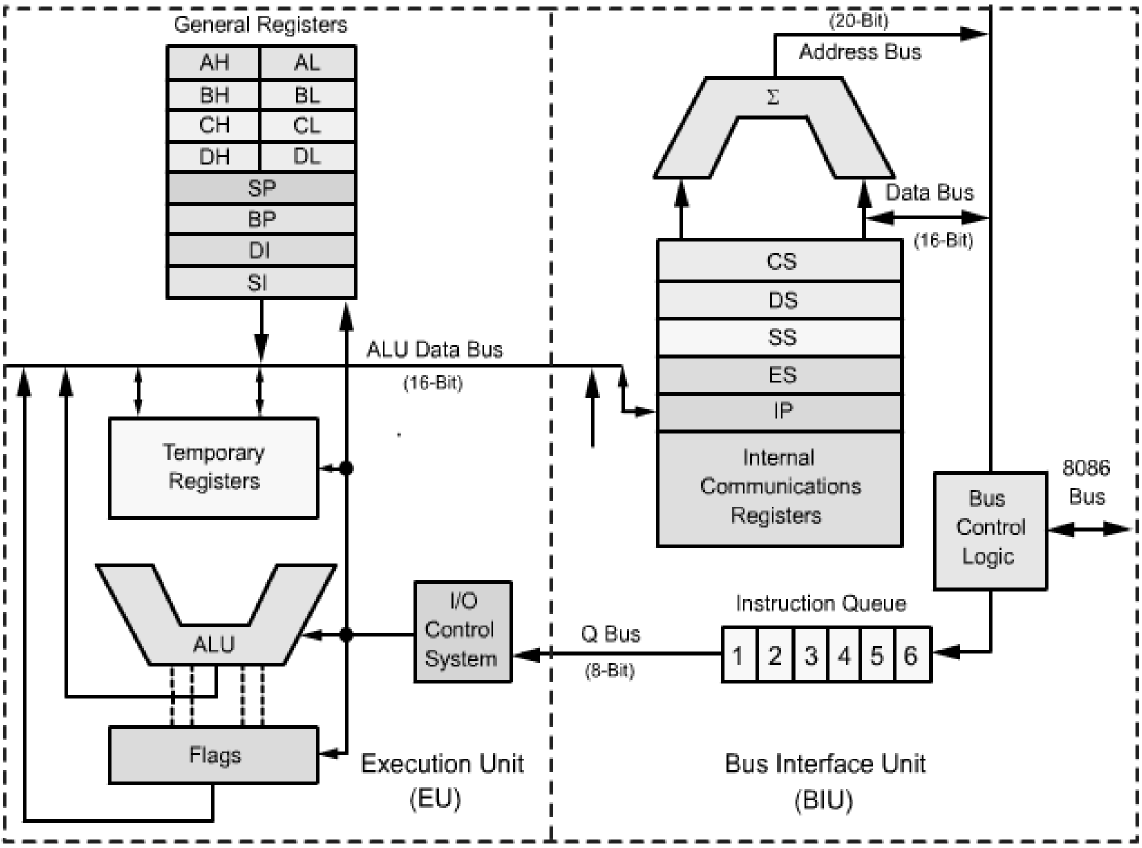
BUS INTERFACE UNIT

- The BIU is used to handle all transfers of data and addresses on the buses for the Execution unit.
- The BIU is used to send addresses, fetch instructions from the memory, read data from ports and memory and write data to ports and memory.
- The BIU contains segment registers, instruction pointer and the instruction queue.

QUEUE

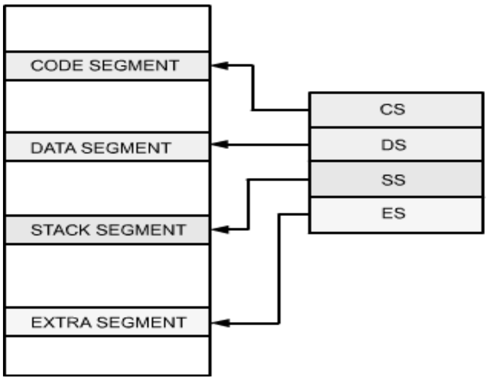
- The queue is a First – In – First – Out (FIFO) group of registers.
- 6 bytes of instruction code are fetched in advance and stored in a queue, while the EU is not using the buses.
- EU fetched instructions from this queue to execute.
- When the EU is ready for the execution of the next instruction, it reads the byte from the instruction queue.
- The time required to access the memory. So it increases the overall processing is called as “Pipelining”.

Architecture of 8086 Microprocessor



SEGMENT REGISTER

- In 8086 memory (1 MB) is divided into number of segments.
- The size of the each segment is 64K bytes.
- A segment is an area that begins with a paragraph boundary, that is, at any location divisible by 16.
- A segment may be located anywhere in the memory.
- Each of these segments can be used for a specific function.
- Code segment is used for storing the instructions.
- The stack segment is used as a stack and it is used to store the return addresses.
- The data and extra segment are used for storing data byte.
- In the assembly language programming, more than one data/code/ stack segments can be defined.
- But only one segment of each type can be accessed at any time.
- The Figure shows different segment and segment registers.



Segments and Segment Register

The BIU contains four segments registers: They are

(i) Code segment register

(ii) Stack segment register

(iii)Data segment register and

(iv) Extra segment register

- These registers are used to hold the upper 16-bits of the starting address of the logical group of memory, called the segment.
- The address of the memory bytes that need to be accessed is generated with the help of address contained in the segment registers and other registers.

CS REGISTER:

- This register contains the initial address of the code segment (CS).
- This address plus the offset value contained in the instruction pointer (IP) indicates the address of an instruction to be fetched for execution.

SS REGISTER:

- The stack segment (SS) register contains the initial address of the stack segment. This address plus the value contained in the stack pointer (SP) is used for stack operations.

DS REGISTER:

- The data segment (DS) register contains the initial address of the current data segment.
- This address plus the offset value in instruction causes a reference to a specific location in the data segment.

ES REGISTER:

- Extra segment is used by some string operations.

- The Extra segment register contains the initial address of the extra segment.
- String instructions always use the ES and DI registers to calculate the physical address for the destination.

ADVANTAGES OF SEGMENT REGISTERS:

1. The segment registers permit a program and its data to be placed in different areas of memory each time the program is executed.

2. The segment registers facilitate the use of separate memory areas for instructions, its data, and the stack.

3. The segment registers are used to allow the instruction, data, or stack portion of a program to be more than 64 K bytes long. The above can be achieved by using more than one code, data, or stack segment.

4. The segment registers are used to allow the memory capacity to be 1MB even though the address associated with the individual instructions are only 16-bits.

INSTRUCTION POINTER (IP):

- The instruction pointer register contains a 16-bit offset address of the instruction that is to be executed next.
- The value contained in the instruction pointer is called as an offset because this value must be added to the base address of the code segment, which is available in the CS register to find the 20-bit physical address.
- The value of the instruction pointer is incremented after executing every instruction.

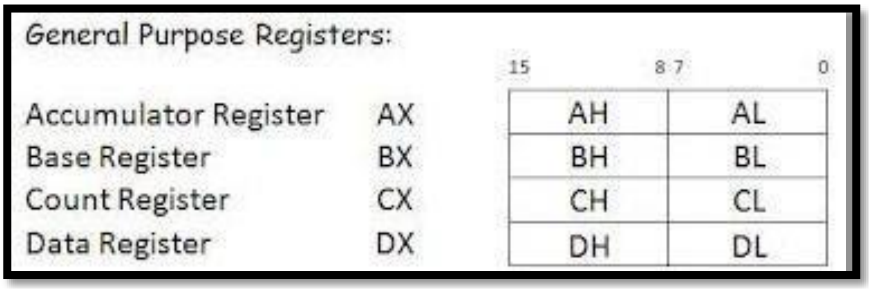
EXECUTION UNIT (EU):

- The Execution Unit is responsible for executing the instructions.
- The EU decodes and executes instructions.
- BIU provide instructions and data to the EU.

The execution unit contains the following sections:

- Control circuitry, Instruction decoder, ALU and nine 16-bit registers.
- The nine registers are AX, BX, CX, DX, SP, BP, SI and DI and flag register.

GENRAL PURPOSE REGISTERS:



General Purpose Registers:

		15 8	7 0
Accumulator Register	AX	AH	AL
Base Register	BX	BH	BL
Count Register	CX	CH	CL
Data Register	DX	DH	DL

The 8086 has four 16-bit general purpose registers: AX, BX, CX and DX. The above 16-bit registers can also be used as 8-bit registers. This BH, BL, CH, CL, DH, DL.

AX REGISTER:

- The AX register is also called as "Accumulator".
- The use of accumulator registers is assumed by some instructions like divide, rotate, shift etc.
- In such kind of instructions, the user loads the accumulator properly before executing the instruction.
- In this case, the complete 16-bit, or only its lower 8-bit AL is used.

BX REGISTER:

- The BX register is called as "Base Register".
- The contents of this register can be used to address the memory.
- All memory references use the content of this register for addressing by using the DS as the default segment register.

CX REGISTER:

- The CX register is called as "Count Register".
- Some instructions like SHIFT, ROTATE and LOOP use the contents of CX as a counter.
- CX register contains the number of times the loop is to be executed.

DX REGISTER:

- The DX register is known as "Data Register".
- Some input/output operations require the use of this register.
- The DX register is used to hold the high 16-bit result in 16 x 16 multiplications or the high 16-bit dividend in 32/16 division and the 16-bit remainder after the division.

STACK POINTER REGISTER:

- A stack is a block of memory to store address or data.
- The base address of the stack is stored in the Stack segment register.
- The special register called the stack pointer contains the address of the top of the stack.
- The stack pointer contains the offset of the data that has been stored latest on the stack.

BASE POINTER REGISTER:

- Base pointer register is also used to access the data from the stack.
- The value in Sp always represents the offset of the top of the stack.
- BP register also contains an offset relative to SS register.

- With the help of BP register, it is possible to access any location within the stack segment of the memory.

INDEX REGISTER:

- The 8086 contain two index registers Source Index (SI) register and Destination Index (DI) register.
- The main use of three register is to hold the offset of a data in one segments.

CONTROL CIRCUITARY, INSTRUCTION DECODER AND ALU:

- The Execution Unit fetches instruction from the instruction queue.
- The above instruction is stored in the decoder.
- The decoder translates each instruction into sequence of actions, which the EU carries out.
- The ALU is used to perform the arithmetic and logic operations.
- All the above actions are controlled by the control circuitry.
- Control circuitry generates appropriate signals at fixed intervals of time.

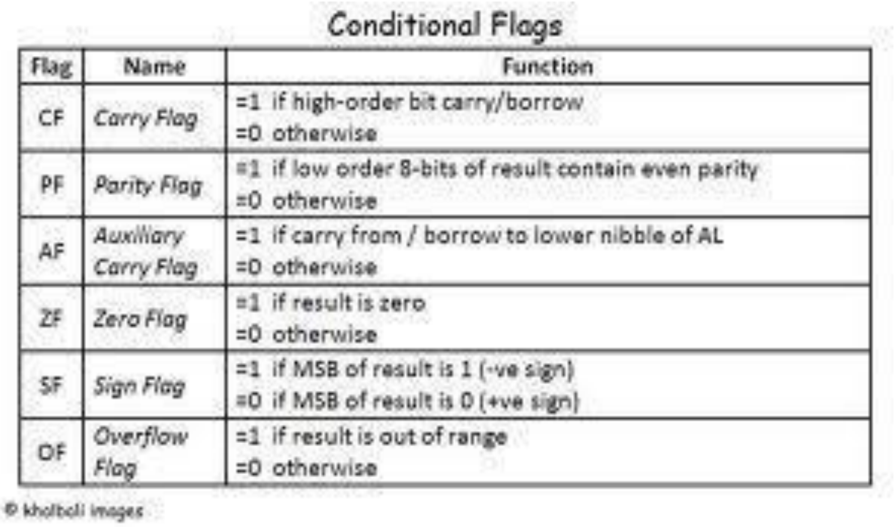
8086 FLAG REGISTER:

A flag register is a 16-bit register. A flag is a flip-flop. The flag register contains nine active flags. Out of the above nine flags, six flags are called status flags or conditional flags and the remaining three flags are called control flags.

The six conditional flags are:

1. The carry Flag (CF)
2. The Auxiliary Carry Flag (AF)
3. The Zero Flag (ZF)
4. The Overflow Flag (OF)
5. The Sign Flag (SF)
6. The Parity Flag (PF)

The status Flags are used to indicate some condition produced by an instruction. The Execution Unit sets or resets these flags at the completion of execution of the arithmetic or logical instruction.



Conditional Flags

Flag	Name	Function
CF	*Carry Flag*	=1 if high-order bit carry/borrow =0 otherwise
PF	*Parity Flag*	=1 if low order 8-bits of result contain even parity =0 otherwise
AF	*Auxiliary Carry Flag*	=1 if carry from / borrow to lower nibble of AL =0 otherwise
ZF	*Zero Flag*	=1 if result is zero =0 otherwise
SF	*Sign Flag*	=1 if MSB of result is 1 (-ve sign) =0 if MSB of result is 0 (+ve sign)
OF	*Overflow Flag*	=1 if result is out of range =0 otherwise

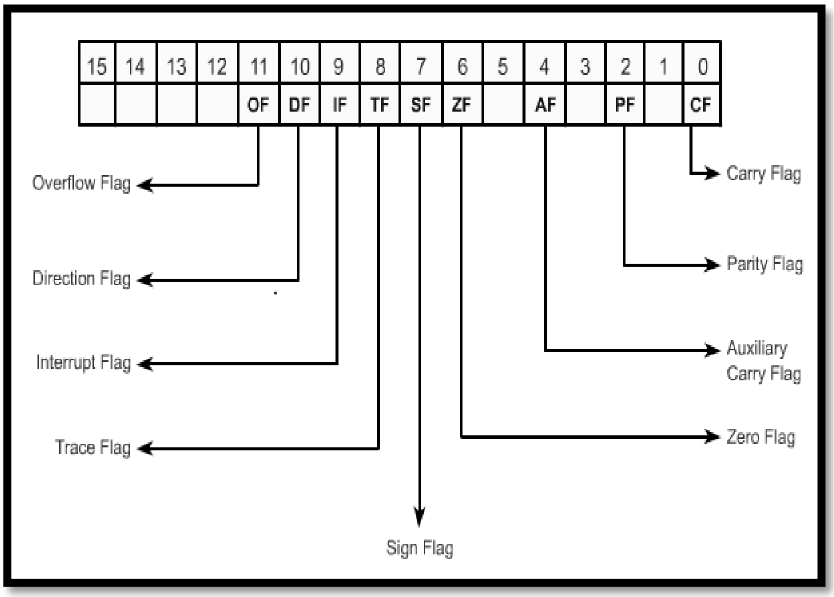
© kholboli images

The three control flags are:

1. The Directory Flag (DF)
2. The Trap Flag (TF)
3. The Interrupt Flag (IF)

These flags are used to control certain operations of the processor.

The control flags are set or reset by the specific instruction. The following figure shows the format of the 8086 flag register. The first row indicates the bit positions and the second row indicates the corresponding flags.



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

8086 flag Register

Status Flags or Conditioal Flags:

CARRY FLAG (CF)

CF is set if there is a carry or borrow for the most significant bit from addition or subtraction operation respectively.

PARITY FALG (PF)

PF is set, if the result contains an even number of 1's. PF is reset, if the result conatins an odd number of 1's.

AUXILIARY CARRY FLAG (AF)

AF is set if there is a carry from the low nibble into the high nibble during addition or a borrow from the high nibble into the low nibble during subtraction of the low order 8-bit of a 16-bit number. Otherwise, Af is reset.

ZERO FLAG (ZF)

ZF is set, if the result of the arithmetic operation is zero. Otherwise it is reset.

SIGN FLAG (SF)

SF is set if the most significant bit of the result is one; otherwise, it is zero. The value of SF =0 indicates that the sign of the result is positive. If SF =1, the result is negative number.

CONTROL FLAGS

TRAP FLAG (TF)

If TF is set, the 8086 works in the single step mode. In the single step mode, one instruction is executed at a time. This type of operation is very useful for debugging programs.

INTERRUPT FLAG (IF)

If IF is set to 1, the processor recognizes all the interrupts; if IF is cleared to zero, the processor ignores interrupts that come from the external devices.

DIRECTION FLAG (DF)

DF is used in string processing. When Df is set to 1, the string is processed backward. If DF is cleared to zero, the string is processed forward.

OVERFLOW FLAG (OF)

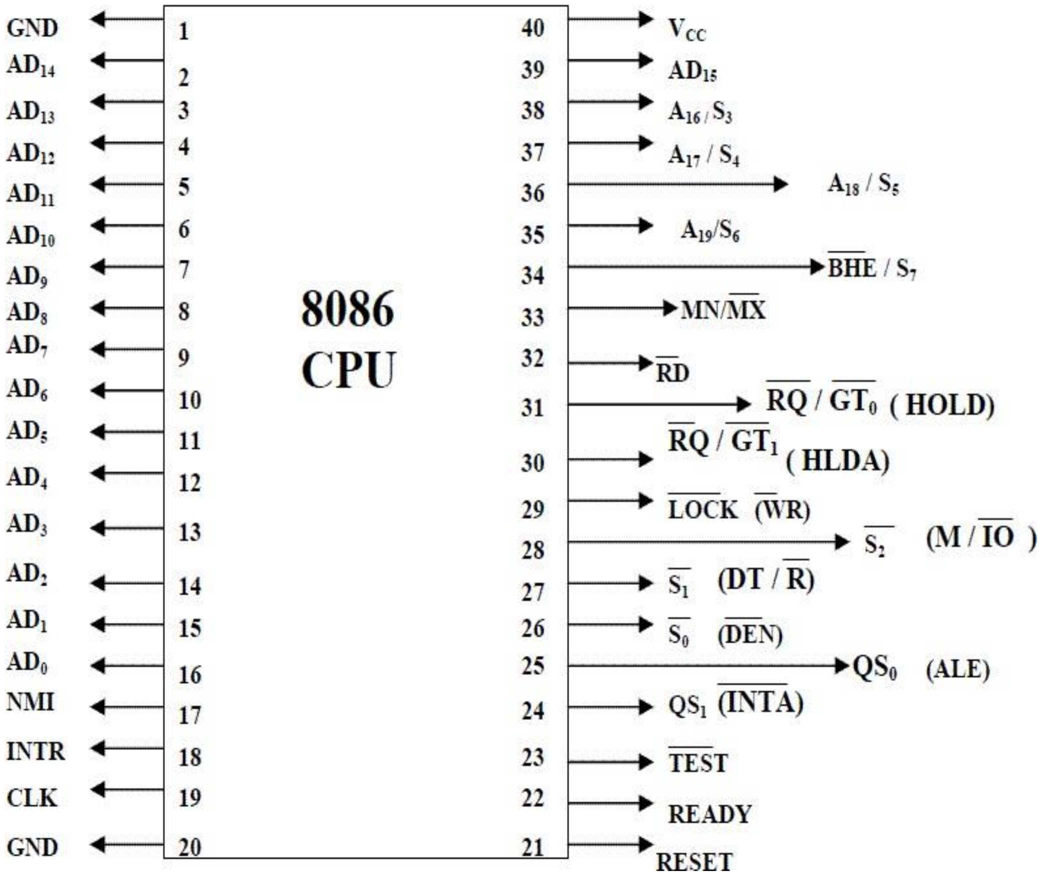
OF is set if there is an arithmetic overflow, i.e., if the size of the result exceeds the capacity of the destination location.

2. PIN DIAGRAM AND PIN DESCRIPTION OF 8086:

Figure shows the Pin diagram of 8086. The description follows it.

- 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).

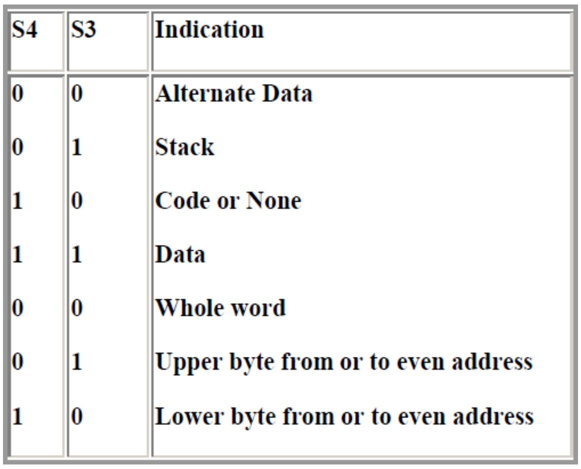
Pin Diagram of 8086



- 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.
    - Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.
- A19/S6, A18/S5, A17/S4, A16/S3 : These are the time multiplexed address and status lines.
    - During T1 these are the most significant address lines for memory operations.
    - During I/O operations, these lines are low.
    - During memory or I/O operations, status information is available on those lines for T2, T3, Tw and T4.
    - The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
    - The S4 and S3 combinely indicate which segment register is presently being used for memory accesses as in below fig.
    - These lines float to tri-state off during the local bus hold acknowledge. The status line S6 is always low.
    - The address bits are separated from the status bit using latches controlled by the ALE signal.



S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or None
1	1	Data
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address

- **BHE/S7:** The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in table. It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T2, T3 and T4. The signal is active low and tristated during hold. It is low during T1 for the first pulses of the interrupt acknowledge cycle.
- **RD – Read:** This signal on low indicates the peripheral that the processor is performing memory or I/O read operation. RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.

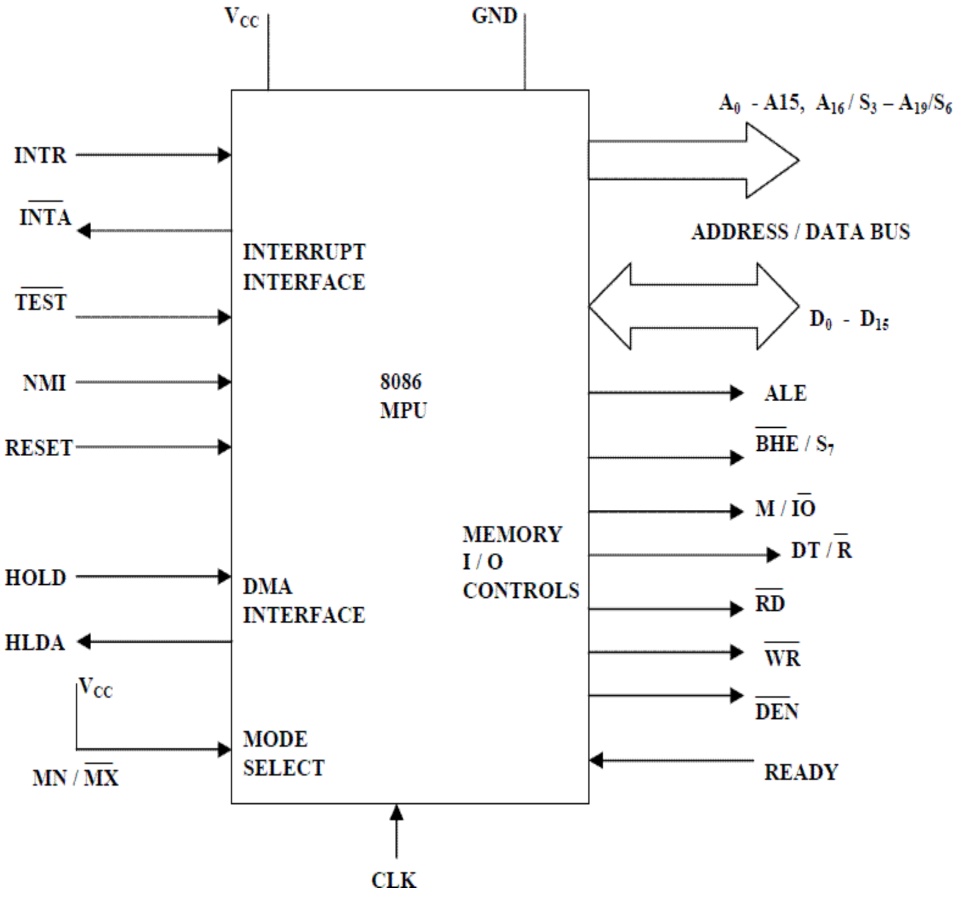
- **READY**: This is the acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. the signal is active high.

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

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

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

Signal Groups of 8086



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

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

- **INTA – Interrupt Acknowledge**: This signal is used as a read strobe for interrupt acknowledge cycles. 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, and is connected to latch enable input of latches. This signal is active high and is never tristated.

- **DT/R – Data Transmit/Receive**: This output is used to decide the direction of data flow through the transceivers (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 transceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T2 until the middle of T4. This is tristated during 'hold acknowledge' cycle.

- **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 HLDA pin, in the middle of the next clock cycle after completing the current bus cycle.

- At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and is should be externally synchronized. If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided:

  1. The request occurs on or before T2 state of the current cycle.

  2. The current cycle is not operating over the lower byte of a word.

  3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

  4. A Lock instruction is not being executed.

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. These become activity during T4 of the previous cycle and active during T1 and T2 of the current bus cycles.

- **LOCK:** This output pin indicates that other system bus master will be prevented from gaining the system bus, while the LOCK signal is low. The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus.

The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This is known as instruction pipelining.

S2	S1	S1	INDICATION
0	0	0	Interrupt Acknowledgement
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

- At the starting the CS:IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty an the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even.

- The first byte is a complete opcode in case of some instruction (one byte opcode instruction) and is a part of opcode, in case of some instructions (two byte opcode instructions), the remaining part of code lie in second byte.

- The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.

- The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program. The fetch operation of the next instruction is overlapped with the execution of the current instruction. As in the architecture, there are two separate units, namely Execution unit and Bus interface unit.

- While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

QS1	QS0	INDICATION
0	0	No Operation
0	1	First Byte of the opcode the queue
1	0	Empty Queue
1	1	Subsequent Byte from the Queue

- 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.
- Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1. RQ/GT pins have internal pull-up resistors and may be left unconnected. Request/Grant sequence is as follows:

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

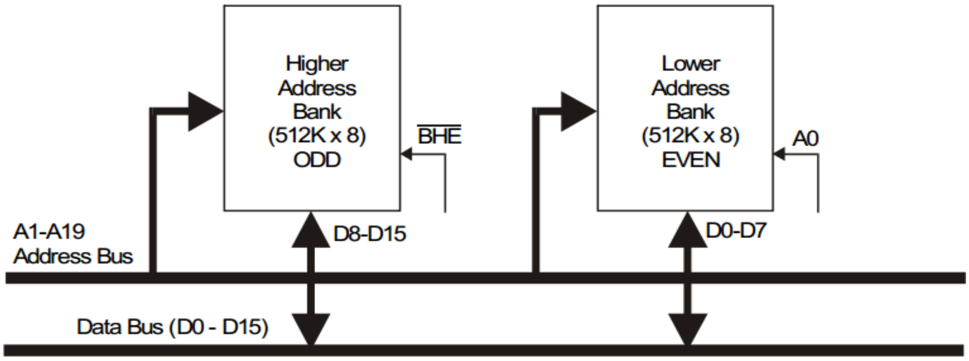
2. During T4(current) or T1(next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the ‘hold acknowledge’ state at next cycle. The CPU bus interface unit is likely to be disconnected from the local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

3. A one clock wide pulse from another master indicates to the 8086 that the hold request is about to end and the 8086 may regain control of the local bus at the next clock cycle. Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus request those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as in case of HOLD and HLDA in minimum mode.

3. 8086 EXTERNAL MEMORY ADDRESSING

The 8086 memory address space can be viewed as a sequence of one million bytes in which any byte may contain an 8-bit data element and any two consecutive bytes may contain a 16-bit data element. There is no constraint on byte or word address boundaries. The address space is physically connected to a 16-bit data bus by dividing the address space into two 8-bit banks of up to 512K bytes each.

One bank is connected to the lower half of the 16-bit data bus (D0 – D7) and contains even address bytes. i.e., when A0 bit is low, the bank is selected. The other bank is connected to the upper half of the data bus (D8 - D15) and contains odd address bytes. i.e., when A0 is high and BHE (Bus High Enable) is low, the odd bank is selected. A specific byte within each bank is selected by address lines A1-A19.


Higher Address Bank (512K x 8) ODD
$\overline{BHE}$
Lower Address Bank (512K x 8) EVEN
A0
A1-A19 Address Bus
D8-D15
D0-D7
Data Bus (D0 - D15)

Data can be accessed from the memory in four different ways. They are:

- 8 - bit data from Lower (Even) address Bank.
- 8 - bit data from Higher (Od) address Bank.
- 16 - bit data starting from Even Address.
- 16 - bit data starting from Od Address.

4. 8086 INTERRUPT PROCESSING

- The event that causes the interruption is called interrupt.
- The special routine executed to service the interrupt is called interrupt service routine.

Normal program can be interrupted by three ways:

1. By external signal
2. By a special instruction in the program or
3. By the occurrence of some condition

An interrupt caused by an external signal is referred as a hardware interrupt conditional interrupts of interrupts caused by special instructions are called software interrupts.

4.1 8086 INTERRUPT TYPES

- Divide by zero interrupt (type 0)
- Single step interrupt (type 1)

- Non maskable interrupt (type 2)
- Breakpoint interrupt (type 3)
- Overflow interrupt (type 4)

Divide by zero interrupt (type 0)

- When the quotient from either a DIV or IDIV instruction is too large to fit in the result register; 8086 will automatically execute type 0 interrupt.

Single step interrupt (type 1)

- The type 1 interrupt is the single step trap.
- In the single step mode, system will execute one instruction and wait for further direction from user.
- Then user can examine the contents of registers and memory locations and if they are correct, user can tell the system to execute the next instruction.
- This feature is useful for debugging assembly language programs.
- An 8086 system is used in the single step mode by setting the trap flag.
- If the trap flag is set, the 8086 will automatically execute a type 1 interrupt after execution if each instruction.
- But the 8086 has no such instruction to directly set or reset the trap flag.
- These operations can be performed by taking the flag register contents into memory, changing the memory contents so to set or reset trap flag and save the memory contents into flags register.
- To reset the trap flag we have to reset bit 8

Non maskable interrupt (type 2)

- As the name suggests, this interrupt cannot be disabled by any software instruction
- This interrupt is activated by low to high transition on 8086 NMI input pin
- In response 8086 will do a type 2 interrupt

Breakpoint interrupt (type 3)

- Type3 interrupt is used to implement break point function in the system.
- It is produced by execution of the INT3 instruction.
- Break point function is often used as debugging aid in case where single stepping provides more detail than wanted.
- When you insert a break point, the system executes the instruction upto the breakpoint, and then goes to the break point procedure.
- In the breakpoint procedure you can write a program to display register contents, memory contents and other information that is required to debug your program. .
- You can insert as many breakpoints as you want in your program.

Overflow interrupt (type 4):

- It is used to check overflow condition after any signed arithmetic operation in the system.
- For example, if you add the 8-bit signed number 0111 1000(+120 decimal) and the 8 bit signed number 0110 0010 (-98 decimal).
- In signed number, MSB is reserved for sign and other bit represent magnitude of the number.
- In the previous example, after addition of two 8-bit signed numbers result is negative, since it is too large to fit in 7bits.
- To delete this condition in the program you can put interrupt on overflow instruction, INTO, immediately after the arithmetic instruction in the program.
- If the overflow flag is not set when the 8086 executes the INTO instructions, the instruction will simply function as a NOP (no operation).
- However, if the overflow flag is set, indicating an overflow error, the 8086 will executes type4 interrupt after executing the INTO instruction.
- Another way to detect and respond to the overflow error in a program is to put the jump if overflow (JO) instruction immediately after the arithmetic instruction.
- If the overflow flag is set as a result of arithmetic operation, execution will jump to the address specified in the JO instruction.
- At this address you can put an error routine which response in the way you want to overflow.

Software interrupts

Type 0-255:

- The 8025 INT instruction can be used to cause the 8086 to do one of the 256 possible interrupt types.
- The interrupt type is specified by the number as a part of the instructions.
- We can use an INT2 instruction to send execution to an NMI interrupt service routine.
- With the s/w interrupts u can call the desired routines from many different programs in a system.
- The BIOS (Basic Input Output System) routines are called INT instructions. we will summarize interrupt response and how it is serviced by going through following steps.

1. 8086 pushes the flag register on the stack

2. It disables the single step & the INTR input by clearing the trap flag & interrupt flag in the flag register.

3. It saves the current CS &IP register contents by pushing them on the stack.

4. Once these values are loaded in CS & IP, 8086 will fetch the instruction from the new address which is the starting address of ISR

5. An IRET instruction at the end of ISR gets the previous values of CS &IP by popping the CS and IP from the stack.

6. At the end of flag register contents are copied back into flag register by popping the flag register from stack.

MASKABLE INTERRUPT (INTR)

- The 8086 INTR input can be used to interrupt a program execution.
- This interrupt can be enabled or disabled by STI(IF=1)or CLI(if=0)

The 8086 responds to an INTR interrupt as follows:

1. The 8086 has 2 interrupt i) interrupt acknowledge machine cycle the 8086 floats the data bus line AD0-AD15.

2. Once the 8086 receives the interrupt type, it pushes the flag register on the stack, Clears TF &IF, pushes the CS & IP values of the next instruction on the stack.

3. The 8086 then gets the new value of IP from the memory address = 4 times the interrupt type & CS value from memory address = 4 times the interrupt number plus2.

5. ADDRESSING MODES OF 8086

Definition: An instruction acts on any number of operands. The way an instruction accesses its operands is called its **Addressing modes**.

Operands may be of three types:

- Implicit
- Explicit
- Both Implicit and Explicit.

Implicit operands mean that the instruction by definition has some specific operands. The programmers do NOT select these operands.

Example: Implicit operands
XLAT; automatically takes AL and BX as operands AAM; it operates on the contents of AX.

Explicit operands mean the instruction operates on the operands specified by the programmer.

Example: Explicit operands
MOV AX, BX; it takes AX and BX as operands XCHG SI, DI; it takes SI and DI as operands

Implicit and explicit operands

Example: Implicit/Explicit operands
MUL BX; automatically multiply BX explicitly times AX

The location of an operand value in memory space is called the **Effective Address (EA)**

We can classify the addressing modes of 8086 into four groups:

- Immediate addressing
- Register addressing
- Memory addressing
- I/O port addressing

The first three Addressing modes are clearly explained.

Immediate addressing mode & Register addressing mode

Immediate Addressing Mode

In this addressing mode, the operand is stored as part of the instruction. The immediate operand, which is stored along with the instruction, resides in the code segment -- not in the data segment. This addressing mode is also faster to execute an instruction because the operand is read with the instruction from memory. Here are some examples:

Example: Immediate Operands
MOV AL, 20 ; move the constant 20 into register AL ADD AX, 5 ; add constant 5 to register EAX MOV DX, offset msg ; move the address of message to register DX

Register addressing mode

In this addressing mode, the operands may be:

- reg16: 16-bit general registers: AX, BX, CX, DX, SI, DI, SP or BP.
- reg8: 8-bit general registers: AH, BH, CH, DH, AL, BL, CL, or DL.
- Sreg: segment registers: CS, DS, ES, or SS. There is an exception: CS cannot be a destination.

For register addressing modes, there is no need to compute the effective address. The operand is in a register and to get the operand there is no memory access involved.

Example: Register Operands
MOV AX, BX ; mov reg16, reg16 ADD AX, SI ; add reg16, reg16 MOV DS, AX ; mov Sreg, reg16

Some rules in register addressing modes:

1. You may not specify CS as the destination operand.

Example: mov CS, 02h –> wrong

2. Only one of the operands can be a segment register. You cannot move data from one segment register to another with a single mov instruction. To copy the value of cs to ds, you would have to use some sequence like:

mov ds,cs -> wrong
mov ax, cs
mov ds, ax -> the way we do it

You should never use the segment registers as data registers to hold arbitrary values. They should only contain segment addresses.

Memory Addressing Modes

Memory (RAM) is the main component of a computer to store temporary data and machine instructions. In a program, programmers many times need to read from and write into memory locations.

There are different forms of memory addressing modes

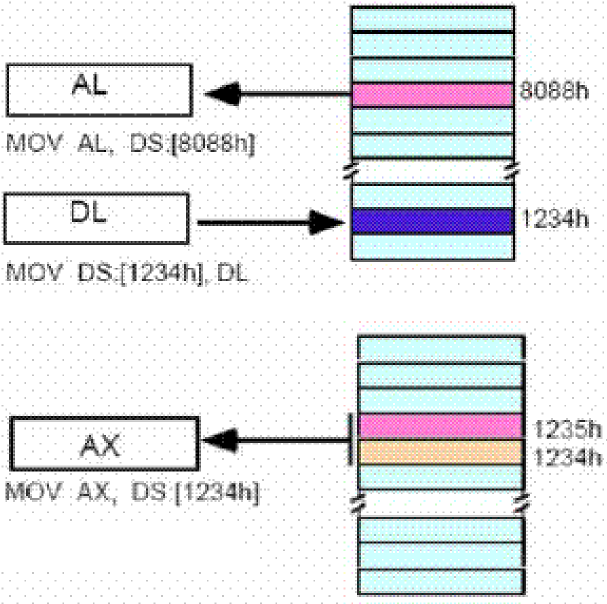
1. Direct Addressing
2. Register indirect addressing
3. Based addressing
4. Indexed addressing

5. Based indexed addressing
6. Based indexed with displacement

Direct Addressing Mode & Register Indirect Addressing Mode

Direct Addressing Mode

The instruction mov al, ds:[8088h] loads the AL register with a copy of the byte at memory location 8088h. Likewise, the instruction mov ds: [1234h],dl stores the value in the dl register to memory location 1234h. By default, all displacement-only values provide offsets into the data segment. If you want to provide an offset into a different segment, you must use a segment override prefix before your address. For example, to access location 1234h in the extra segment (es) you would use an instruction of the form mov ax, es:[1234h]. Likewise, to access this location in the code segment you would use the instruction mov ax, cs: [1234h]. The ds: prefix in the previous examples is not a segment override.



The instruction mov al, ds:[8088h] is same as mov al, [8088h]. If not mentioned DS register is taken by default.

Register Indirect Addressing Mode

The 80x86 CPUs let you access memory indirectly through a register using the register indirect addressing modes. There are four forms of this addressing mode on the 8086, best demonstrated by the following instructions:

```
mov al, [bx]
mov al, [bp]
mov al, [si]
mov al, [di]
```

Code Example

```
MOV BX, 100H
MOV AL, [BX]
```

The [bx], [si], and [di] modes use the ds segment by default. The [bp] addressing mode uses the stack segment (ss) by default. You can use the segment override prefix symbols if you wish to access data in different segments. The following instructions demonstrate the use of these overrides:

```
mov al, cs:[bx]
mov al, ds:[bp]
mov al, ss:[si]
mov al, es:[di]
```

Intel refers to [bx] and [bp] as base addressing modes and bx and bp as base registers (in fact, bp stands for base pointer). Intel refers to the [si] and [di] addressing modes as indexed addressing modes (si stands for source index, di stands for destination index). However, these addressing modes are functionally equivalent. This text will call these forms register indirect modes to be consistent.

Based Addressing Mode and Indexed Addressing Modes

Based Addressing Mode

8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.

```
Mov al, [bx],[si]
Mov bl , [bp],[di]
Mov cl , [bp],[di]
```

Code Example

```
If bx=1000h
si=0880h
Mov AL, [1000+880]
Mov AL,[1880]
```

Indexed Addressing Modes

The indexed addressing modes use the following syntax:

```
mov al, [bx+disp]
mov al, [bp+disp]
mov al, [si+disp]
mov al, [di+disp]
```

Code Example

```
MOV BX, 100H
MOV AL, [BX + 15]
MOV AL, [BX + 16]
```

If bx contains 1000h, then the instruction mov cl, [bx+20h] will load cl from memory location ds:1020h. Likewise, if bp contains 2020h, mov dh, [bp+1000h] will load dh from location ss:3020. The offsets generated by these addressing modes are the sum of the constant and the specified register. The addressing modes involving bx, si, and di all use the data segment, the [bp+disp] addressing mode uses the stack segment by default. As with the register indirect addressing modes, you can use the segment override prefixes to specify a different segment:

```
mov al, ss:[bx+disp]
mov al, es:[bp+disp]
mov al, cs:[si+disp]
mov al, ss:[di+disp]
```

Based Indexed Addressing Modes & Based Indexed Plus Displacement Addressing Mode

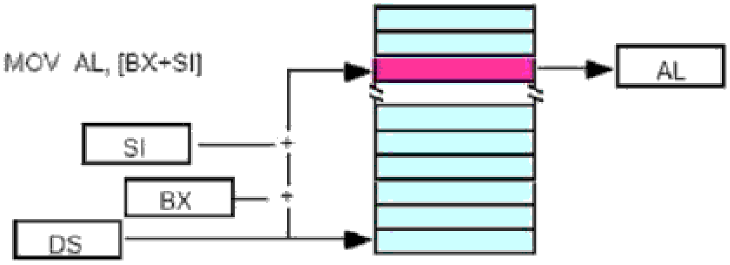
Based Indexed Addressing Modes

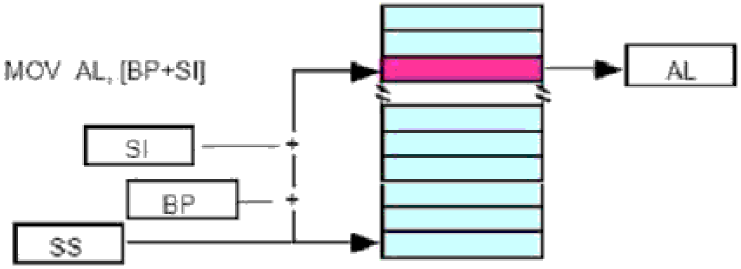
The based indexed addressing modes are simply combinations of the register indirect addressing modes. These addressing modes form the offset by adding together a base register (bx or bp) and an index register (si or di). The allowable forms for these addressing modes are:

```
mov al, [bx+si]
mov al, [bx+di]
mov al, [bp+si]
mov al, [bp+di]
```

Code Example

```
MOV BX, 100H
MOV SI, 200H
MOV AL, [BX + SI]
INC BX
INC SI
```





Suppose that bx contains 1000h and si contains 880h. Then the instruction mov al, [bx][si] would load al from location DS:1880h. Likewise, if bp contains 1598h and di contains 1004, mov ax, [bp+di] will load the 16 bits in ax from locations SS: 259C and SS: 259D. The addressing modes that do not involve bp use the data segment by default. Those that have bp as an operand use the stack segment by default.

Based Indexed Plus Displacement Addressing Mode

These addressing modes are a slight modification of the base/indexed addressing modes with the addition of an eight bit or sixteen bit constant. The following are some examples of these addressing modes

```
mov al, disp[bx][si]
mov al, disp[bx+di]
mov al, [bp+si+disp]
mov al, [bp][di][disp]
```

Code Example

```
MOV BX, 100H
MOV SI, 200H
MOV AL, [BX + SI +100H]
INC BX
INC SI
```

6. INSTRUCTION SETS OF 8086

The instruction set of a processor can defines the basic operations that a programmer can make the device to perform. The 8086 instruction contains no operand, single operand and two operand instructions. The instruction set will be divided into number of groups of functionally related instructions.

- Data transfer instructions
- Arithmetic instructions
- Bit manipulations instructions
- String manipulations instructions
- Conditional branch instructions
- Unconditional branch instructions
- Iteration control instructions
- Interrupt control instructions
- Processor control instructions

1. Data transfer instructions

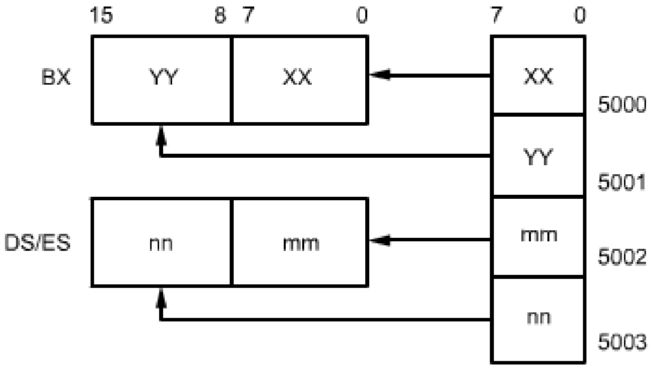
The data transfer instructions transfers' data from one register/memory locations to other register/memory locations. All the store, move, load, exchange, input and output instructions belong to this category. These instructions move single byte or word between a register and I/O ports. Some of the data transfer instructions are listed below.

- MOV d,s
- PUSH d
- POP d
- LEA reg,mem
- LDS reg,mem

The following example will explain the instructions mentioned above

MOV CX,DX copies 16 bit content of DX to CX.

LDS BX, 5000H loads register and DS from memory.



2. Arithmetic instructions

This type of instructions usually perform the arithmetic operations, like Addition, Subtraction, Multiplications, Division, Increment and Decrement operation along with respective ASCII and decimal adjust instructions. The operands are either the registers or memory locations or immediate data depending upon the addressing mode. Some of the arithmetic instructions are given below.

- ADD a,b
- ADC a,b
- AAA
- INC reg/mem
- SUB a,b
- SBB a,b
- CMP a,b
- MUL reg,mem
- DIV reg,mem

The following are the examples for some of the above mentioned instructions.

ADD AX, 0100H Adds to register

SUB 0100H Subtract byte or word, destination is AX

3. Bit manipulation instructions

These types of instructions are used for carrying out the bit by bit shift, rotate in basic logical operations. All the condition code flags are affected depending upon the result. The 8086 provides three groups of bit manipulation instructions. Some of the instructions are listed below.

- AND a, b
- OR a, b
- XOR a, b
- SHL/SAL mem / reg, CNT
- SHR/SAR mem /reg, CNT
- RCL mem/reg, CNT
- RCR mem/reg, CNT

The following are the example for the above mentioned instructions.

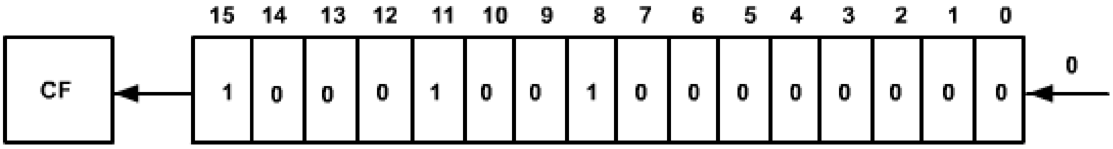
1. AND, Logical AND, Logical OR, Logical Inverter and Logical XOR

The source operand that may be available immediately, register or a memory location ANDed bit by bit to the destination operand that may be a register or a memory location.

Eg: AND AX, 008H

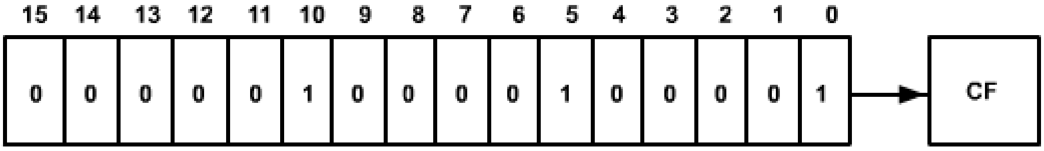
- **SHL/SAL [Shift Logical/Arithmetic Left]**

The instructions shift the operand word or byte bit by bit to the left and insert zeros in the newly introduced least significant bits.



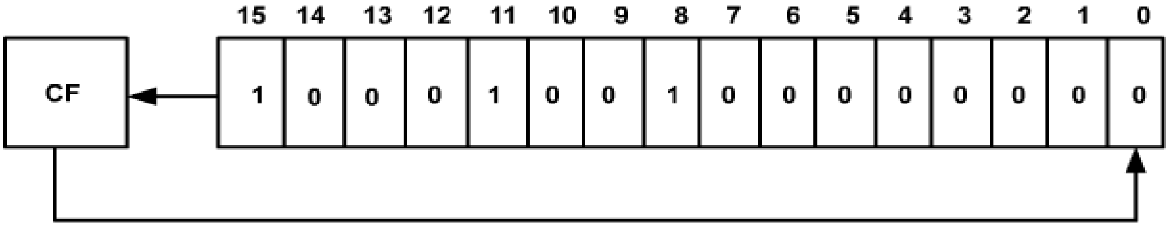
- **SHR/SAR [Shift Logical Right /Arithmetic Right]**

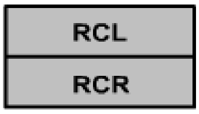
The instructions are same as SHL/SAL; the only difference is shift the operand word or byte bit by bit to the right. The result is stored in the destination operand.



- **RCR/RCR [Rotate Right through Carry/Rotate Left through Carry]**

These instructions rotate the contents (bit-wise) of the destination operand right or left respectively by the specified count through Carry flag (CF). In RCR, the Carry flag is pushed in to the MSB of the operand, and the LSB is pushed into carry flag for each operation. In RCL the carry flag for each operation.





4. String manipulation instructions

String instructions are available to MOVE, COMPARE or SCAS for a value as well as to move string elements to and from the accumulator. A series of the data bytes or words available is memory at consecutive locations, to be referred to collectively or word strings. For referring to a string two parameters are required (a) string or end address of the sting (b) length of the string. Some of the string manipulations are given below.

- MOVSB/MOVSW[Move String Byte or String Word]

These instruction moves 8 or 16 bit data from the memory l0cation addressed by SI to another set of destination location which is addressed by SI.

Eg: MOV AX, 5000H -Source segment address is 5000H

- CMPS [Compare String Byte or String Words]

When two strings of bytes or words are to be compared the CMPs can be used. The length of the strings must be stored in the CX register. If both strings are equal Z=1, other flags are affected in the same way as CMP instruction.

5. Conditional branch instructions

In these instructions, the control is transferred to the specified location provided the result of previous operation satisfies a particular condition, otherwise, the execution continuous in normal flow sequence. All the conditional branch instructions use 8-bit signed displacement these type of instructions do not affect any flag. The typical structure of the conditional branch is follows.

If condition is true,

Then PC←PC + disp 8 otherwise

PC←PC + 2 and execute next instruction.

Some of the conditional branch instructions are given below

- JZ
- JNZ
- JS
- JNS

6. Unconditional branch instructions

These instructions, transfers the execution control to the specified location independent of any status or condition. The CS and IP are unconditionary, modified to the new CS and IP. The 8086 unconditional transfers are

CALL reg/mem/disp 16 ; call subroutine

RET ; return from subroutine

JMP reg/mem/disp 8/disp 16 ; Unconditional jump

- **CALL[Unconditional call]**

This instruction is used to call a subroutine form amain program. While executing this instruction IP is incremented (i.e. address of the next instruction to be executed) and CS on to the stack with the flags and loads the CS and IP register respectively.

- **RET[Return from the Subourtine]**

At the end of the subroutine, the RET must e executed, upon executing the previously stored contents of IP and CS along with flags are retrieved into CS, IP and flag registers from the stack and the main program will be executed the types of procedure and the SP contents.

They are

a) Return with segment

b) Return within segment adding 16-bit immediate displacement to the SP contents

c) Return intersegment

d) Return intersegment adding 16-bit immediate displace ment to the SP contents

- **JMP[Unconditional JumP]**

This instruction transfers the control of execution to the specified address using an 8-bit or 16-bit displacement or CS unconditionally. No flags are affected by this instruction.

7. Iteration control instructions

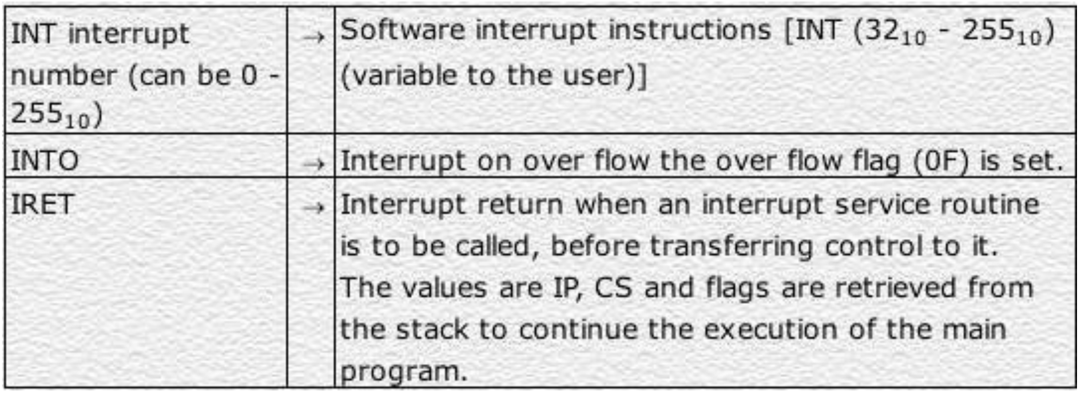
These instructions execute the part of the program from the label or address specified in the instruction up to the loop instruction, CX number of times. These instructions are given below

- LOOP disp 8 Decrement CX by 1 without affecting flags and loop if CX≠0.
- LOOP E/LOOP Z disp8 Decrement CX by 1 without affecting flags and loop if CX≠0 /not equal .
- JCX Z disp 8 JMP if register CX = 0

8. Interrupt instructions

In the 8086, there are 256 interrupts are defined corresponding to the types from 00Hto FFH. Where an Interrupt instruction executed, the TYPE byte N Is multiplied and the contents of IP and CS of the interrupt service routine will be taken from the hexadecimal multiplication

(N*4) as offset address and 0000 as segment address. The interrupt instruction in8086 is listed below.



INT interrupt number (can be 0 - 255_{10})	→	Software interrupt instructions [INT (32_{10} - 255_{10}) (variable to the user)]
INTO	→	Interrupt on over flow the over flow flag (0F) is set.
IRET	→	Interrupt return when an interrupt service routine is to be called, before transferring control to it. The values are IP, CS and flags are retrieved from the stack to continue the execution of the main program.

Process control instructions

Hardware function in the processor chip can be controlled by these instructions.

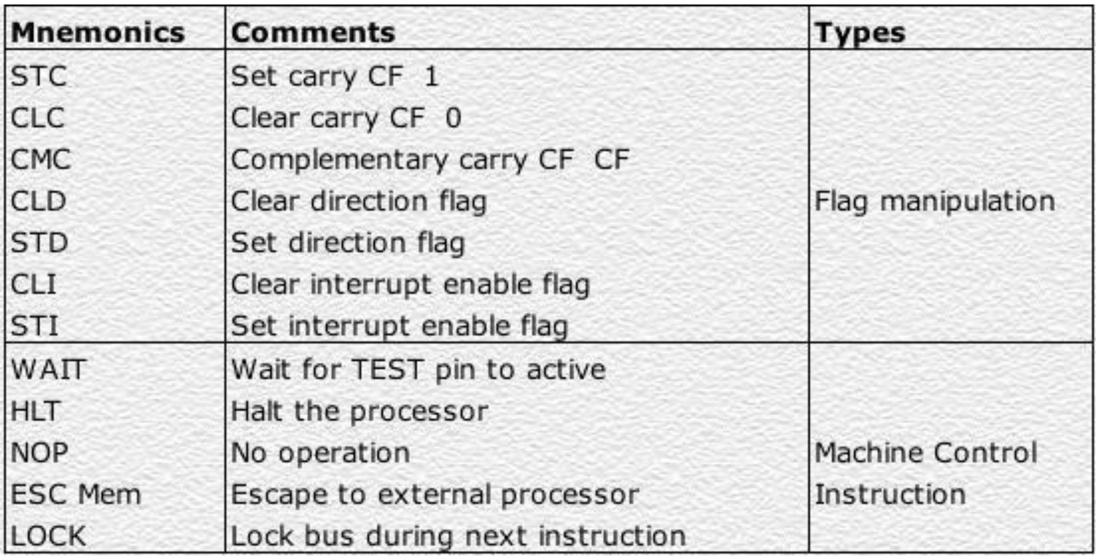
There are two types

i. Flag manipulation Instruction

ii. Machine control Instruction

First one directly modifies some of the flags of 8086, later control the bus usage and execution.

A processor control instruction available in 8086 is listed below.



Mnemonics	Comments	Types
STC	Set carry CF 1	Flag manipulation
CLC	Clear carry CF 0	
CMC	Complementary carry CF CF	
CLD	Clear direction flag	
STD	Set direction flag	
CLI	Clear interrupt enable flag	
STI	Set interrupt enable flag	
WAIT	Wait for TEST pin to active	Machine Control Instruction
HLT	Halt the processor	
NOP	No operation	
ESC Mem	Escape to external processor	
LOCK	Lock bus during next instruction	

7. ASSEMBLER DIRECTIVES

Assembly languages are low-level languages for programming computers, microprocessors, microcontrollers, and other IC. They implement a symbolic representation of the numeric machine Codes and other constants needed to program a particular CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on abbreviations that help the programmer to remember individual instructions, registers. An assembler directive is a statement to give direction to the assembler to perform task of the assembly process.

It control the organization if the program and provide necessary information to the assembler to understand the assembly language programs to generate necessary machine codes. They indicate how an operand or a section of the program is to be processed by the assembler.

An assembler supports directives to define data, to organise segments to control procedure, to define macros. It consists of two types of statements: instructions and directives. The instructions are translated to the machine code by the assembler whereas directives are not translated to the machine codes.

Assembler directive 8086 microprocessor

(a) The DB directive

(b) The DW directive

(c) The DD directive

(d) The STRUCT (or STRUC) and ENDS directives (counted as one)

(e)The EQU Directive

(f)The COMMENT directive

(g)ASSUME

(h) EXTERN

(i) GLOBAL

(j) SEGMENT

(k)OFFSET

(l) PROC

(m)GROUP

(n) INCLUDE

Data declaration directives:

1. DB - The DB directive is used to declare a BYTE -2-BYTE variable - A BYTE is made up of 8 bits.

Declaration examples:

Byte1 DB 10h

Byte2 DB 255 ; 0FFh, the max. possible for a BYTE

CRLF DB 0Dh, 0Ah, 24h ;Carriage Return, terminator BYTE

2. DW - The DW directive is used to declare a WORD type variable - A WORD occupies 16 bits or (2 BYTE).

Declaration examples:

Word DW 1234h

Word2 DW 65535; 0FFFFh, (the max. possible for a WORD)

3. DD - The DD directive is used to declare a DWORD - A DWORD double word is made up of 32 bits =2 Word's or 4 BYTE.

Declaration examples:

Dword1 DW 12345678h

Dword2 DW 4294967295 ;0FFFFFFFFh.

4. STRUCT and ENDS directives to define a structure template for grouping data items.

(1) The STRUCT directive tells the assembler that a user defined uninitialized data structure follows. The uninitialized data structure consists of a combination of the three supported data types. DB, DW, and DD. The labels serve as zero-based offsets into the structure. The first element's offset for any structure is 0. A structure element is referenced with the base "+" operator before the element's name.

A Structure ends by using the ENDS directive meaning END of Structure.

Syntax:

STRUCT

Structure_element_name element_data_type?

```
. . .
. . .
. . .
ENDS
(OR)
STRUC
Structure_element_name element_data_type?
. . .
. . .
. . .
ENDS
DECLARATION:
STRUCT
Byte1 DB?
Byte2 DB?
Word1 DW?
Word2 DW?
Dword1DW?
Dword2 DW?
ENDS
```

Use OF STRUCT:

The STRUCT directive enables us to change the order of items in the structure when, we reform a file header and shuffle the data. Shuffle the data items in the file header and reformat the sequence of data declaration in the STRUCT and off you go. No change in the code we write that processes the file header is necessary unless you inserted an extra data element.

(5) The EQU Directive

The EQU directive is used to give name to some value or symbol. Each time the assembler finds the given names in the program, it will replace the name with the value or a symbol. The value can be in the range 0 through 65535 and it can be another Equate declared anywhere above or below.

The following operators can also be used to declare an Equate:

THIS BYTE

THIS WORD

THIS DWORD

A variable - declared with a DB, DW, or DD directive - has an address and has space reserved at that address for it in the .COM file. But an Equate does not have an address or space reserved for it in the .COM file.

Example:

A - Byte EQU THIS BYTE

DB 10

A_ word EQU THIS WORD

DW 1000

A_ dword EQU THIS DWORD

DD 4294967295

Buffer Size EQU 1024

Buffer DB 1024 DUP (0)

Buffed_ ptr EQU $; actually points to the next byte after the; 1024th byte in buffer.

(6) Extern:

It is used to tell the assembler that the name or label following the directive are I some other assembly module. For example: if you call a procedure which is in program module assembled at a different time from that which contains the CALL instructions ,you must tell the assembler that the procedure is external the assembler will put information in the object code file so that the linker can connect the two module together.

Example:

PROCEDURE -HERE SEGMENT

EXTERN SMART-DIVIDE: FAR ; found in the segment; PROCEDURES-HERE

PROCEDURES-HERE ENDS

(7) GLOBAL: The GLOBAL directive can be used in place of PUBLIC directive .for a name defined in the current assembly module; the GLOBAL directive is used to make the symbol available to the other modules.

Example: GLOBAL DIVISOR:

WORD tells the assembler that DIVISOR is a variable of type of word which is in another assembly module or EXTERN.

(8) SEGMENT:

It is used to indicate the start of a logical segment. It is the name given to the the segment. Example: the code segment is used to indicate to the assembler the start of logical segment.

(9) PROC: (PROCEDURE)

It is used to identify the start of a procedure. It follows a name we give the procedure.

After the procedure the term NEAR and FAR is used to specify the procedure Example: SMART-DIVIDE PROC FAR identifies the start of procedure named SMART-DIVIDE and tells the assembler that the procedure is far.

(10) NAME:

It is used to give a specific name to each assembly module when program consists of several modules.

Example: PC-BOARD used to name an assembly module which contains the instructions for controlling a printed circuit board.

(11) INCLUDE:

It is used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source module. An alternative is use of editor block command to cop the file into the current source module.

(12) OFFSET:

It is an operator which tells the assembler to determine the offset or displacement of a named data item from the start of the segment which contains it. It is used to load the offset of a variable into a register so that variable can be accessed with one of the addressed modes. Example: when the assembler read MOV BX.OFFSET PRICES, it will determine the offset of the prices.

(13) GROUP:

It can be used to tell the assembler to group the logical segments named after the directive into one logical group. This allows the contents of all he segments to be accessed from the same group. Example: SMALL-SYSTEM GROUP CODE, DATA, STACK-SEG.