

LECTURE NOTES ON 8085 MICROPROCESSOR

Mr. ASHOK S PATIL

CHAPTER: 1

History of microprocessor:-

The invention of the transistor in 1947 was a significant development in the world of technology. It could perform the function of a large component used in a computer in the early years. Shockley, Brattain and Bardeen are credited with this invention and were awarded the Nobel prize for the same. Soon it was found that the function this large component was easily performed by a group of transistors arranged on a single platform. This platform, known as the integrated chip (IC), turned out to be a very crucial achievement and brought along a revolution in the use of computers. A person named Jack Kilby of Texas Instruments was honored with the Nobel Prize for the invention of IC, which laid the foundation on which microprocessors were developed. At the same time, Robert Noyce of Fairchild made a parallel development in IC technology for which he was awarded the patent.

ICs proved beyond doubt that complex functions could be integrated on a single chip with a highly developed speed and storage capacity. Both Fairchild and Texas Instruments began the manufacture of commercial ICs in 1961. Later, complex developments in the IC led to the addition of more complex functions on a single chip. The stage was set for a single controlling circuit for all the computer functions. Finally, Intel corporation's Ted Hoff and Frederico Fagin were credited with the design of the first microprocessor.

The work on this project began with an order from a Japanese calculator company Busicom to Intel, for building some chips for it. Hoff felt that the design could integrate a number of functions on a single chip making it feasible for providing the required functionality. This led to the design of Intel 4004, the world's first microprocessor. The next in line was the 8 bit 8008 microprocessor. It was developed by Intel in 1972 to perform complex functions in harmony with the 4004.

This was the beginning of a new era in computer applications. The use of mainframes and huge computers was scaled down to a much smaller device that was affordable to many. Earlier, their use was limited to large organizations and universities. With the advent of microprocessors, the use of computers trickled down to the common man. The next processor in line was Intel's 8080 with an 8 bit data bus and a 16 bit address bus. This was amongst the most popular microprocessors of all time.

Very soon, the Motorola corporation developed its own 6800 in competition with the Intel's 8080. Fagin left Intel and formed his own firm Zilog. It launched a new microprocessor Z80 in 1980 that was far superior to the previous two versions. Similarly, a break off from Motorola prompted the design of 6502, a derivative of the 6800. Such attempts continued with some modifications in the base structure.

The use of microprocessors was limited to task-based operations specifically required for company projects such as the automobile sector. The concept of a 'personal computer' was still a distant dream for the world and microprocessors were yet to come into personal use. The 16 bit microprocessors started becoming a commercial sell-out in the 1980s with the first popular one being the TMS9900 of Texas Instruments.

Intel developed the 8086 which still serves as the base model for all latest advancements in the microprocessor family. It was largely a complete processor integrating all the required features in it. 68000 by Motorola was one of the first microprocessors to develop the concept of microcoding in its instruction set. They were further developed to 32 bit architectures. Similarly, many players like Zilog, IBM and Apple were successful in getting their own products in the market. However, Intel had a commanding position in the market right through the microprocessorera.

The 1990s saw a large scale application of microprocessors in the personal computer applications developed by the newly formed Apple, IBM and Microsoft corporation. It witnessed a revolution in the use of computers, which by then was a household entity.

This growth was complemented by a highly sophisticated development in the commercial use of microprocessors. In 1993, Intel brought out its 'Pentium Processor' which is one of the most popular processors in use till date. It was followed by a series of excellent processors of the Pentium family, leading into the 21st century. The latest one in commercial use is the Pentium Dual Core technology and the Xeon processor. They have opened up a whole new world of diverse applications. Supercomputers have become common, owing to this amazing development in microprocessors.

INTRODUCTION TO MICROPROCESSOR AND MICROCOMPUTER ARCHITECTURE:

A *microprocessor* is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer. Any microprocessor-based systems having limited number of resources are called *microcomputers*. Nowadays, microprocessor can be seen in almost all types of electronics devices like mobile phones, printers, washing machines etc. Microprocessors are also used in advanced applications like radars, satellites and flights. Due to the rapid advancements in electronic industry and large scale integration of devices results in a significant cost reduction and increase application of microprocessors and their derivatives.

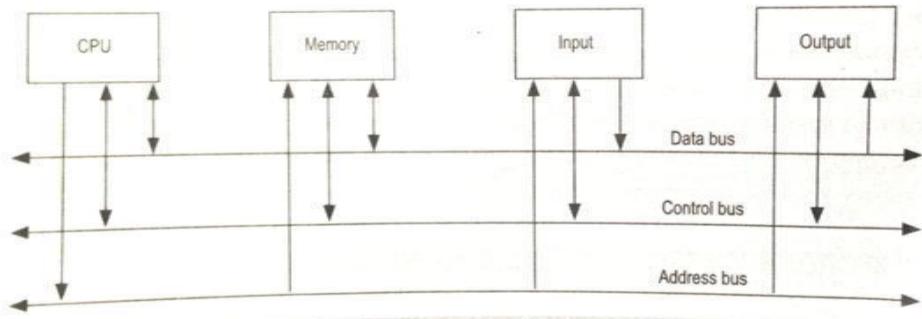


Fig.1 Microprocessor-based system

- **Bit:** A bit is a single binary digit.
- **Word:** A word refers to the basic data size or bit size that can be processed by the arithmetic and logic unit of the processor. A 16-bit binary number is called a word in a 16-bit processor.
- **Bus:** A bus is a group of wires/lines that carry similar information.
- **System Bus:** The system bus is a group of wires/lines used for communication between the microprocessor and peripherals.
- **Memory Word:** The number of bits that can be stored in a register or memory element is called a memory word.
- **Address Bus:** It carries the address, which is a unique binary pattern used to identify a memory location or an I/O port. For example, an eight bit address bus has eight lines and thus it can address $2^8 = 256$ different locations. The locations in hexadecimal format can be written as 00H – FFH.
- **Data Bus:** The data bus is used to transfer data between memory and processor or between I/O device and processor. For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.
- **Control Bus:** The control bus carry control signals, which consists of signals for selection of memory or I/O device from the given address, direction of data transfer and synchronization of data transfer in case of slow devices.

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

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

Technology Used:

The semiconductor manufacturing technologies used for chips are:

- Transistor-Transistor Logic (TTL)
- Emitter Coupled Logic (ECL)
- Complementary Metal-Oxide Semiconductor (CMOS)

Classification of Microprocessors:

Based on their specification, application and architecture microprocessors are classified.

Based on size of data bus:

- 4-bit microprocessor
- 8-bit microprocessor
- 16-bit microprocessor
- 32-bit microprocessor

Based on application:

- General-purpose microprocessor- used in general computer system and can be used by programmer for any application. Examples, 8085 to Intel Pentium.
- Microcontroller- microprocessor with built-in memory and ports and can be programmed for any generic control application. Example, 8051.
- Special-purpose processors- designed to handle special functions required for an application. Examples, digital signal processors and application-specific integrated circuit (ASIC) chips.

Based on architecture:

- Reduced Instruction Set Computer (RISC) processors
- Complex Instruction Set Computer (CISC) processors

2. 8085 MICROPROCESSOR ARCHITECTURE

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

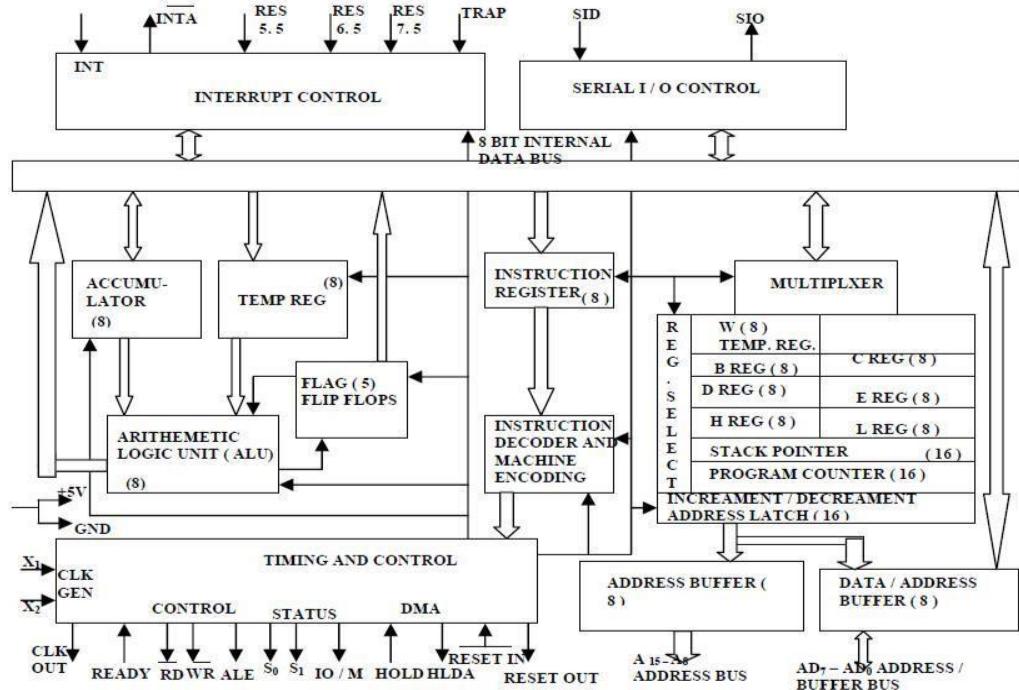


Fig. 2 Internal Architecture of 8085

Arithmetic and Logic Unit

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

Registers

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

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some

16-bit operations. The programmer can use these registers to store or copy data into the register by using data copy instructions.

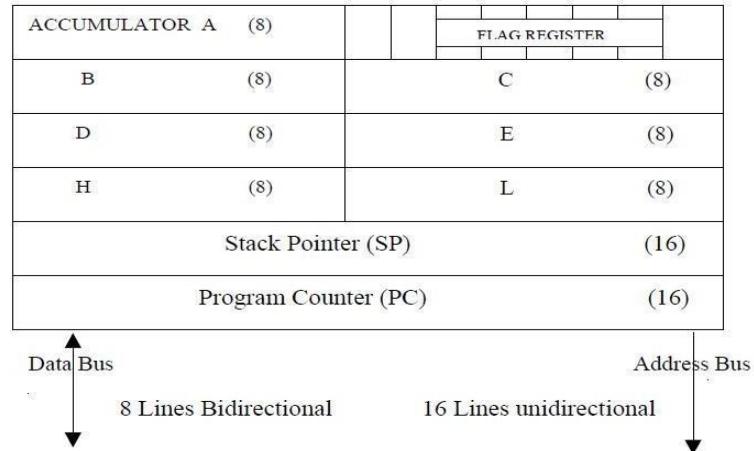


Fig. 3 Register organisation

Accumulator

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

Flag register

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

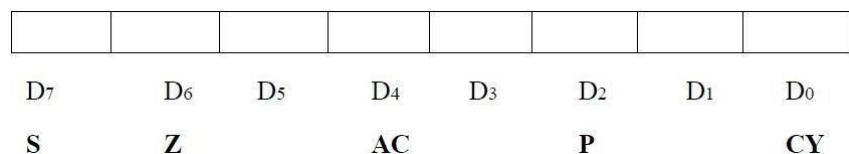


Fig. 4 Flag register

For example, after an addition of two numbers, if the result in the accumulator is larger than 8-bit, the flip-flop uses to indicate a carry by setting CY flag to 1. When an arithmetic operation results in zero, Z flag is set to 1. The S flag is just a copy of the bit D₇ of the accumulator. A negative number has a 1 in bit D₇ and a positive number has a 0 in 2's complement representation. The AC flag is set to 1, when a carry result from bit D₃ and passes to bit D₄. The P flag is set to 1, when the result in accumulator contains even number of 1s.

Program Counter (PC)

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

Stack Pointer (SP)

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

Instruction Register/Decoder

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

Control Unit

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

- *Data Bus:* Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit i.e. 2^8 combination of binary digits and are typically identified as D0 – D7. Thus size of the data bus determines what arithmetic can be done. If only 8-bit wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor.
- *Address Bus:* The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time-shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.
- *Control Bus:* Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carries control signals partly unidirectional and partly bidirectional. The following control and status signals are used by 8085 processor:
 - I. ALE (output): Address Latch Enable is a pulse that is provided when an address appears on the AD0 – AD7 lines, after which it becomes 0.

- II. RD (active low output): The Read signal indicates that data are being read from the selected I/O or memory device and that they are available on the data bus.
- III. WR (active low output): The Write signal indicates that data on the data bus are to be written into a selected memory or I/O location.
- IV. IO/M (output): It is a signal that distinguishes between a memory operation and an I/O operation. When $IO/M = 0$ it is a memory operation and $IO/M = 1$ it is an I/O operation.
- V. S1 and S0 (output): These are status signals used to specify the type of operation being performed; they are listed in Table 1.

Table 1 Status signals and associated operations

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

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

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

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

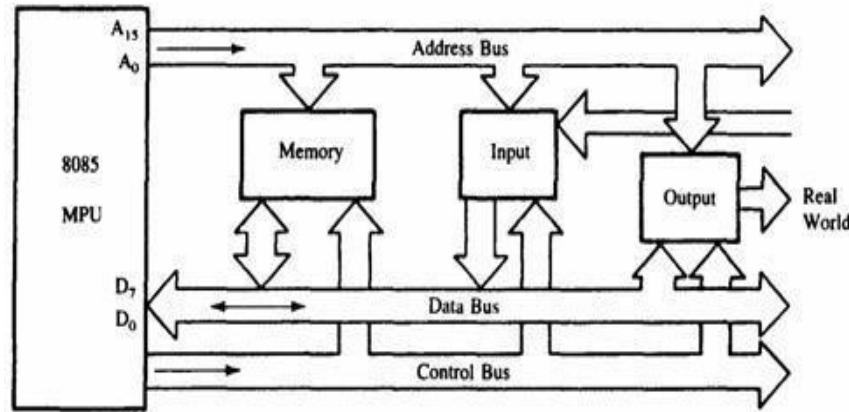


Fig. 5 The 8085 bus structure

3. 8085 PIN DESCRIPTION

Properties:

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

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

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

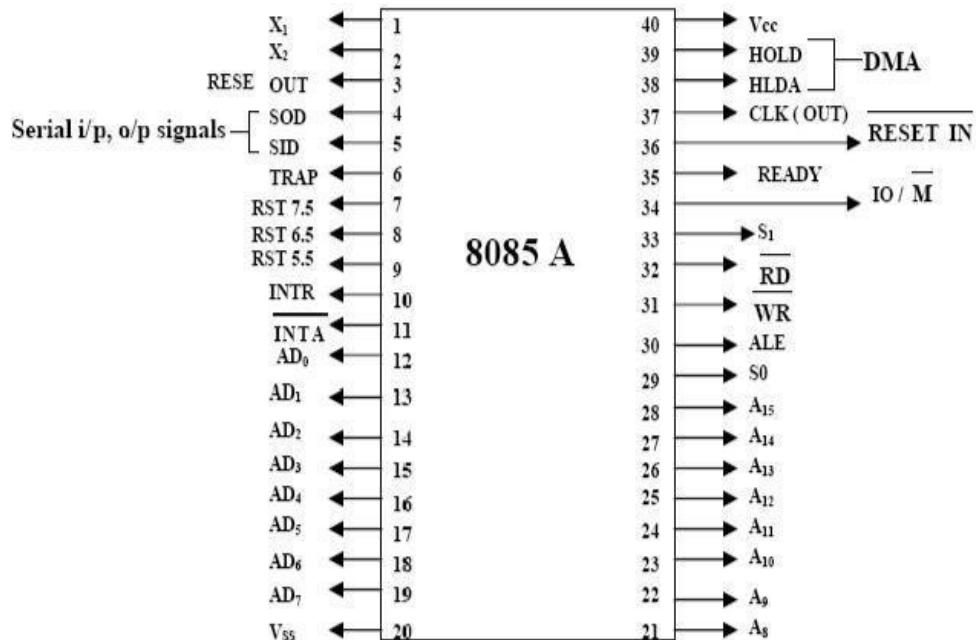


Fig. 6 8085 microprocessor pin layout and signal groups

Address and Data Buses:

- A₈ – A₁₅ (output, 3-state): Most significant eight bits of memory addresses and the eight bits of the I/O addresses. These lines enter into tri-state high impedance state during HOLD and HALT modes.
- AD₀ – AD₇ (input/output, 3-state): Lower significant bits of memory addresses and the eight bits of the I/O addresses during first clock cycle. Behaves as data bus

during third and fourth clock cycle. These lines enter into tri-state high impedance state during HOLD and HALT modes.

Control & Status Signals:

- ALE: Address latch enable
- RD : Read control signal.
- WR : Write control signal.
- IO/M , S1 and S0 : Status signals. Power

Supply & Clock Frequency:

- Vcc: +5 V power supply
- Vss: Ground reference
- X1, X2: A crystal having frequency of 6 MHz is connected at these two pins
- CLK: Clock output

Externally Initiated and Interrupt Signals:

- RESET IN : When the signal on this pin is low, the PC is set to 0, the buses are tri-stated and the processor is reset.
- RESET OUT: This signal indicates that the processor is being reset. The signal can be used to reset other devices.
- READY: When this signal is low, the processor waits for an integral number of clock cycles until it goes high.
- HOLD: This signal indicates that a peripheral like DMA (direct memory access) controller is requesting the use of address and data bus.
- HLDA: This signal acknowledges the HOLD request.
- INTR: Interrupt request is a general-purpose interrupt.
- INTA : This is used to acknowledge an interrupt.
- RST 7.5, RST 6.5, RST 5.5 – restart interrupt: These are vectored interrupts and have highest priority than INTR interrupt.
- TRAP: This is a non-maskable interrupt and has the highest priority.

Serial I/O Signals:

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

4. INSTRUCTION SET AND EXECUTION IN 8085

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

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

Classification based on functionality:

- I. Data transfer operations: This group of instructions copies data from source to destination. The content of the source is not altered.
- II. Arithmetic operations: Instructions of this group perform operations like addition, subtraction, increment & decrement. One of the data used in arithmetic operation is stored in accumulator and the result is also stored in accumulator.
- III. Logical operations: Logical operations include AND, OR, EXOR, NOT. The operations like AND, OR and EXOR uses two operands, one is stored in accumulator and other can be any register or memory location. The result is stored in accumulator. NOT operation requires single operand, which is stored in accumulator.
- IV. Branching operations: Instructions in this group can be used to transfer program sequence from one memory location to another either conditionally or unconditionally.
- V. Machine control operations: Instruction in this group control execution of other instructions and control operations like interrupt, halt etc.

Classification based on length:

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

Table 2 Examples of one byte instructions

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

Table 3 Examples of two byte instructions

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

Table 4 Examples of three byte instructions

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

Addressing Modes in Instructions:

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

1. Immediate addressing
2. Memory direct addressing
3. Register direct addressing
4. Indirect addressing
5. Implicit addressing

Immediate Addressing:

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

Ex: MVI A, 9AH

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

Memory Direct Addressing:

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

Ex: LDA 850FH

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

Register Direct Addressing:

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

Ex: MOV B, C

It copies the content of register C to register B.

Indirect Addressing:

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

Ex: MOV A, M

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

Implicit Addressing

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

Ex: CMA

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

5. INSTRUCTION SET OF 8085

Data Transfer Instructions:

MOV instruction

XCHG Instruct

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG D,S	(D) ↔ (S)	None

(a)

Destination	Source
Accumulator	Reg16
Memory	Register
Register	Register
Register	Memory

XLAT

(b)

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT	((AL) + (BX) + (DS) *10) AL	none

LEA, LDS, and LES instructions

LEA: Load effective Address, LEA Reg 16, EA

LDS: Load register and DS, LDS Reg 16, EA

LES: Load register and ES, LES Reg 16, EA

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 4350 or STA XYZ
Store accumulator indirect STAX Reg. pair	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. Example: STAX.B
Store H and L registers direct SHLD 16-bit address	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: SHLD 2470
Exchange H and L with D and E XCHG none	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example: XCHG
Copy H and L registers to the stack pointer SPHL none	The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. Example: SPHL
Exchange H and L with top of stack XTHL none	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. Example: XTHL

Push register pair onto stack
PUSH Reg. pair

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

Example: 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 87

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

The contents of the input port designated in the operand are read and loaded into the accumulator.

Example: IN 82

Arithmetic Instructions:

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

SUB R
 M

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

Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB R
 M

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

Example: SBB B or SBB M

Subtract immediate from accumulator

SUI 8-bit data

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

Example: SUI 45

Subtract immediate from accumulator with borrow

SBI 8-bit data

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

Example: SBI 45

Increment register or memory by 1

INR R
 M

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

Example: INR B or INR M

Increment register pair by 1

INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same place.

Example: INX H

Decrement register or memory by 1

DCR R
 M

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

Decrement register pair by 1

DCX R

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

Decimal adjust accumulator

DAA none

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

BRANCHING INSTRUCTIONS

Opcode Operand

Description

Jump unconditionally

JMP 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.

Example: JMP 2034 or JMP XYZ

Jump conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.

Example: JZ 2034 or JZ 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 2034 or CALL XYZ

Call conditionally

Operand: 16-bit address

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

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

Return from subroutine unconditionally

RET none

The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RET

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RZ

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

Load program counter with HL contents
PCHL none

The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.
Example: PCHL

Restart
RST 0-7

The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

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

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

LOGICAL INSTRUCTIONS

Opcode Operand

Description

Compare register or memory with accumulator

CMP R
 M

The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:
if (A) < (reg/mem): carry flag is set, s=1
if (A) = (reg/mem): zero flag is set, s=0
if (A) > (reg/mem): carry and zero flags are reset, s=0
Example: CMP B or CMP M

Compare immediate with accumulator

CPI 8-bit data

The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
if (A) < data: carry flag is set, s=1
if (A) = data: zero flag is set, s=0
if (A) > data: carry and zero flags are reset, s=0
Example: CPI 89

Logical AND register or memory with accumulator

ANA R
 M

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

Logical AND immediate with accumulator

ANI 8-bit data

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

Exclusive OR register or memory with accumulator

XRA R
 M

The contents of the accumulator are Exclusive 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

Exclusive OR immediate with accumulator

XRI 8-bit data

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

Example: XRI 86

Logical OR register or memory with accumulaotr

ORA R
 M

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

Example: ORA B or ORA M

Logical OR immediate with accumulator

ORI 8-bit data

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

Example: ORI 86

Rotate accumulator left

RLC none

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

Example: RLC

Rotate accumulator right

RRC none

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

Example: RRC

Rotate accumulator left through carry

RAL none

Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D₇ is placed in the Carry flag, and the Carry flag is placed in the least significant position D₀. CY is modified according to bit D₇. S, Z, P, AC are not affected.

Example: RAL

Rotate accumulator right through carry

RAR none

Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D₀ is placed in the Carry flag, and the Carry flag is placed in the most significant position D₇. CY is modified according to bit D₀. S, Z, P, AC are not affected.

Example: RAR

Complement accumulator

CMA none

The contents of the accumulator are complemented. No flags are affected.

Example: CMA

Complement carry

CMC none

The Carry flag is complemented. No other flags are affected.

Example: CMC

Set Carry

STC none

The Carry flag is set to 1. No other flags are affected.

Example: STC

CONTROL INSTRUCTIONS

Opcode Operand

Description

No operation

NOP none

No operation is performed. The instruction is fetched and decoded. However no operation is executed.

Example: NOP

Halt and enter wait state

HLT none

The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state.

Example: HLT

Disable interrupts

DI none

The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.

Example: DI

Enable interrupts

EI none

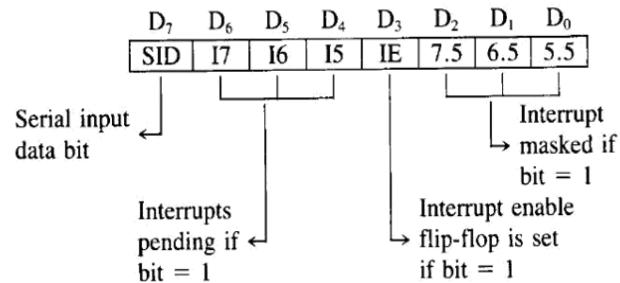
The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP).

Example: EI

Read interrupt mask
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.

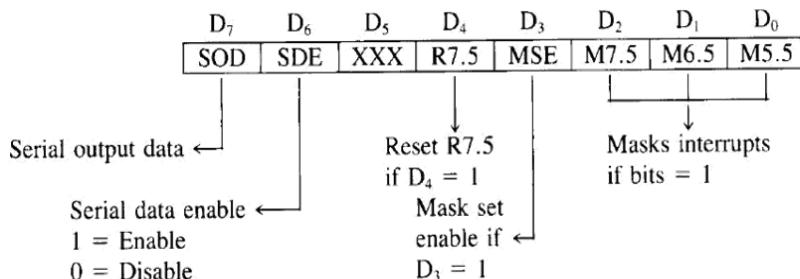
Example: RIM



Set interrupt mask
SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM



- SOD—Serial Output Data: Bit D₇ of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D₆ = 1.
- SDE—Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- XXX—Don't Care
- R7.5—Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- MSE—Mask Set Enable: If this bit is high, it enables the functions of bits D₂, D₁, D₀. This is a master control over all the interrupt masking bits. If this bit is low, bits D₂, D₁, and D₀ do not have any effect on the masks.
- M7.5—D₂ = 0, RST 7.5 is enabled.
 = 1, RST 7.5 is masked or disabled.
- M6.5—D₁ = 0, RST 6.5 is enabled.
 = 1, RST 6.5 is masked or disabled.
- M5.5—D₀ = 0, RST 5.5 is enabled.
 = 1, RST 5.5 is masked or disabled.

6. INSTRUCTION EXECUTION AND TIMING DIAGRAM:

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

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

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

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

To execute a program, 8085 performs various operations as:

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

External communication functions are:

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

Opcode Fetch Machine Cycle:

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

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

T1 clock cycle

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

T2 clock cycle

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

T3 clock cycle

- i. The opcode available on the data bus is read by the processor and moved to the instruction register.
- ii. The RD signal is deactivated by making it logic 1.

T4 clock cycle

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

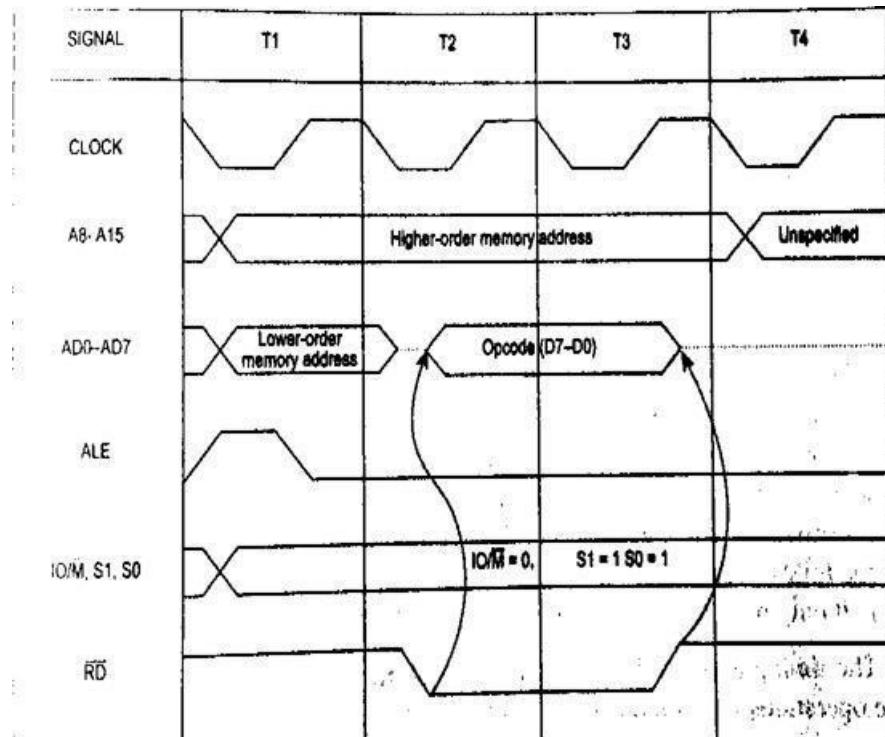


Fig. 7 Timing diagram for opcode fetch cycle

Memory Read Machine Cycle:

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

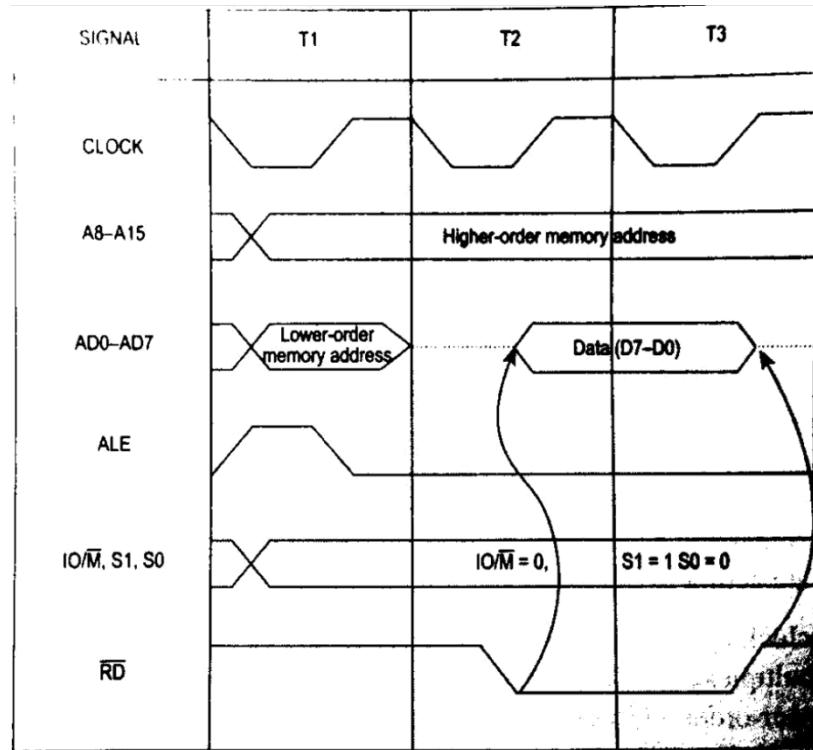


Fig. 8 Timing diagram for memory read machine cycle

Memory Write Machine Cycle:

The memory write cycle is executed by the processor to write a data byte in a memory location. The processor takes three T-states and WR signal is made low. The timing diagram of this cycle is given in Fig. 9.

I/O Read Cycle:

The I/O read cycle is executed by the processor to read a data byte from I/O port or from peripheral, which is I/O mapped in the system. The 8-bit port address is placed both in the lower and higher order address bus. The processor takes three T-states to execute this machine cycle. The timing diagram of this cycle is given in Fig. 10.

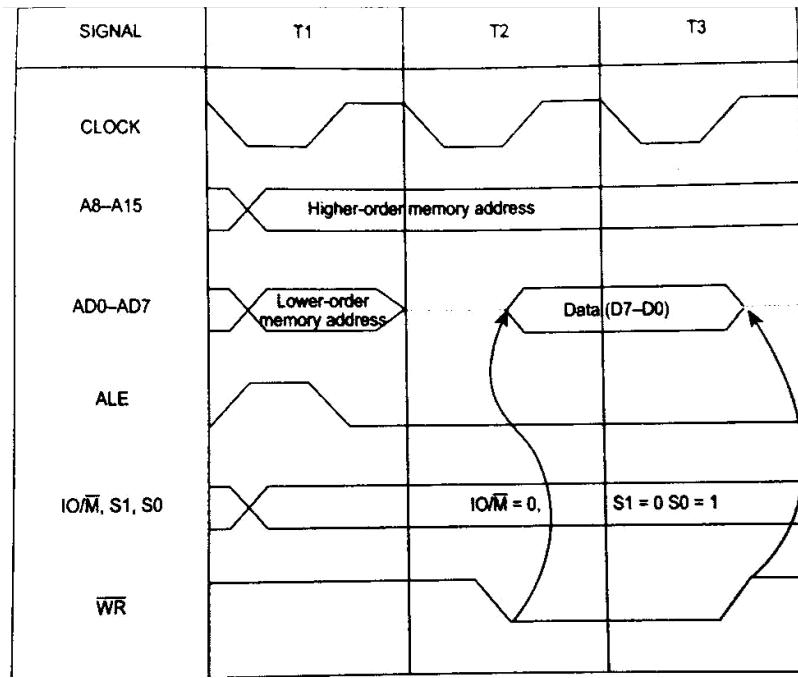


Fig. 9 Timing diagram for memory write machine cycle

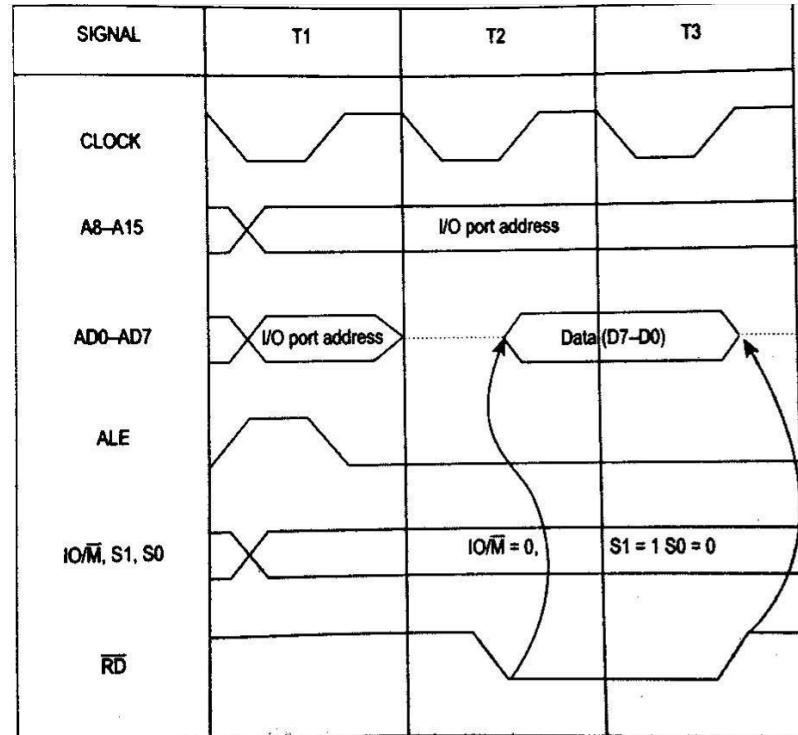


Fig. 10 Timing diagram I/O read machine cycle

I/O Write Cycle:

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

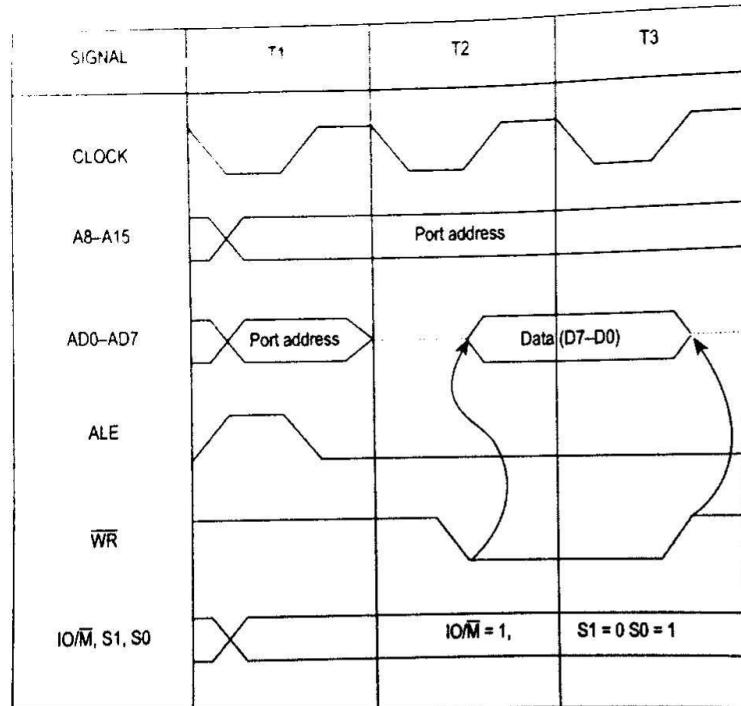


Fig. 11 Timing diagram I/O write machine cycle

Ex: Timing diagram for IN 80H.

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

Table 5 IN instruction

Address	Mnemonics	Opcode
800F	IN 80H	DB
8010		80

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

The timing diagram is shown in Fig. 12.

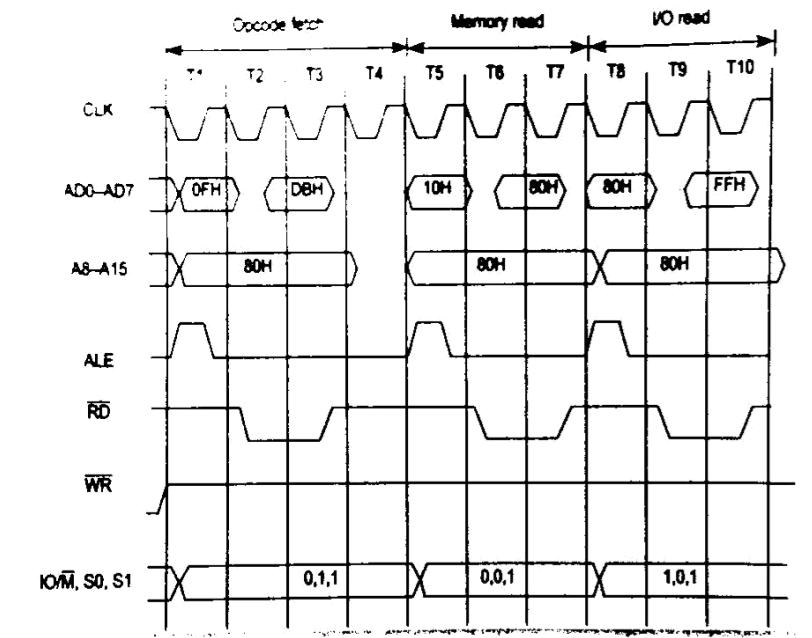


Fig. 12 Timing diagram for the IN instruction

7. 8085 INTERRUPTS

Interrupt Structure:

Interrupt is the mechanism by which the processor is made to transfer control from its current program execution to another program having higher priority. The interrupt signal may be given to the processor by any external peripheral device.

The program or the routine that is executed upon interrupt is called interrupt service routine (ISR). After execution of ISR, the processor must return to the interrupted program. Key features in the interrupt structure of any microprocessor are as follows:

- i. Number and types of interrupt signals available.
- ii. The address of the memory where the ISR is located for a particular interrupt signal. This address is called interrupt vector address (IVA).
- iii. Masking and unmasking feature of the interrupt signals.
- iv. Priority among the interrupts.
- v. Timing of the interrupt signals.
- vi. Handling and storing of information about the interrupt program (status information).

Types of Interrupts:

Interrupts are classified based on their maskability, IVA and source. They are classified as:

- i. Vectored and Non-Vectored Interrupts
 - Vectored interrupts require the IVA to be supplied by the external device that gives the interrupt signal. This technique is vectoring, is implemented in number of ways.
 - Non-vectored interrupts have fixed IVA for ISRs of different interrupt signals.
- ii. Maskable and Non-Maskable Interrupts
 - Maskable interrupts are interrupts that can be blocked. Masking can be done by software or hardware means.
 - Non-maskable interrupts are interrupts that are always recognized; the corresponding ISRs are executed.
- iii. Software and Hardware Interrupts
 - Software interrupts are special instructions, after execution transfer the control to predefined ISR.
 - Hardware interrupts are signals given to the processor, for recognition as an interrupt and execution of the corresponding ISR.

Interrupt Handling Procedure:

The following sequence of operations takes place when an interrupt signal is recognized:

- i. Save the PC content and information about current state (flags, registers etc) in the stack.
- ii. Load PC with the beginning address of an ISR and start to execute it.
- iii. Finish ISR when the return instruction is executed.
- iv. Return to the point in the interrupted program where execution was interrupted.

Interrupt Sources and Vector Addresses in 8085:

Software Interrupts:

8085 instruction set includes eight software interrupt instructions called Restart (RST) instructions. These are one byte instructions that make the processor execute a subroutine at predefined locations. Instructions and their vector addresses are given in Table 6.

Table 6 Software interrupts and their vector addresses

Instruction	Machine hex code	Interrupt Vector Address
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0032H

The software interrupts can be treated as CALL instructions with default call locations. The concept of priority does not apply to software interrupts as they are inserted into the program as instructions by the programmer and executed by the processor when the respective program lines are read.

Hardware Interrupts and Priorities:

8085 have five hardware interrupts – INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP. Their IVA and priorities are given in Table 7.

Table 7 Hardware interrupts of 8085

Interrupt	Interrupt vector address	Maskable or non-maskable	Edge or level Triggered	priority
TRAP	0024H	Non-maskable	Level	1
RST 7.5	003CH	Maskable	Rising edge	2
RST 6.5	0034H	Maskable	Level	3
RST 5.5	002CH	Maskable	Level	4
INTR	Decided by hardware	Maskable	Level	5

Masking of Interrupts:

Masking can be done for four hardware interrupts INTR, RST 5.5, RST 6.5, and RST 7.5. The masking of 8085 interrupts is done at different levels. Fig. 13 shows the organization of hardware interrupts in the 8085.

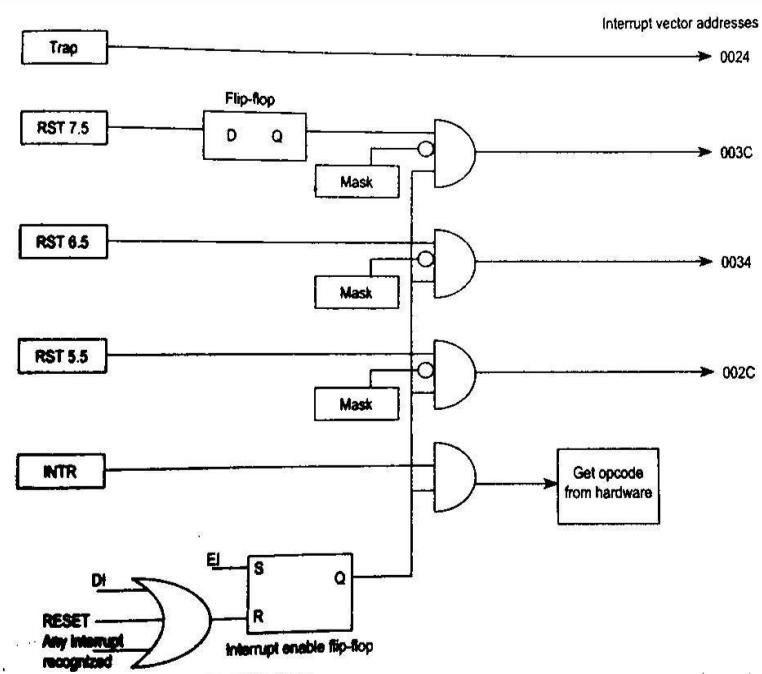


Fig. 13 Interrupt structure of 8085

The Fig. 13 is explained by the following five points:

- i. The maskable interrupts are by default masked by the Reset signal. So no interrupt is recognized by the hardware reset.
- ii. The interrupts can be enabled by the EI instruction.
- iii. The three RST interrupts can be selectively masked by loading the appropriate word in the accumulator and executing SIM instruction. This is called software masking.
- iv. All maskable interrupts are disabled whenever an interrupt is recognized.
- v. All maskable interrupts can be disabled by executing the DI instruction.

RST 7.5 alone has a flip-flop to recognize edge transition. The DI instruction reset interrupt enable flip-flop in the processor and the interrupts are disabled. To enable interrupts, EI instruction has to be executed.

SIM Instruction:

The SIM instruction is used to mask or unmask RST hardware interrupts. When executed, the SIM instruction reads the content of accumulator and accordingly mask or unmask the interrupts. The format of control word to be stored in the accumulator before executing SIM instruction is as shown in Fig. 14.

Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SOD	SDE	X	R7.5	MSE	M7.5	M6.5	M5.5
Explanation	Serial data to be sent	Serial data enable— set to 1 for sending	Not used	Reset RST 7.5 flip-flop	Mask set enable— Set to 1 to mask to mask interrupts	Set to 1 to mask RST 7.5	Set to 1 to mask RST 6.5	Set to 1 to mask RST 5.5

Fig. 14 Accumulator bit pattern for SIM instruction

In addition to masking interrupts, SIM instruction can be used to send serial data on the SOD line of the processor. The data to be send is placed in the MSB bit of the accumulator and the serial data output is enabled by making D6 bit to 1.

RIM Instruction:

RIM instruction is used to read the status of the interrupt mask bits. When RIM instruction is executed, the accumulator is loaded with the current status of the interrupt masks and the pending interrupts. The format and the meaning of the data stored in the accumulator after execution of RIM instruction is shown in Fig. 15.

In addition RIM instruction is also used to read the serial data on the SID pin of the processor. The data on the SID pin is stored in the MSB of the accumulator after the execution of the RIM instruction.

Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
Explanation	Serial input data in the SID pin	Set to 1 if RST 7.5 is pending	Set to 1 if RST 6.5 is pending	Set to 1 if RST 5.5 is pending	Set to 1 if interrupts are pending	Set to 1 if RST 7.5 is masked enabled	Set to 1 if RST 6.5 is masked	Set to 1 if RST 5.5 is masked

Fig. 15 Accumulator bit pattern after execution of RIM instruction

Ex: Write an assembly language program to enables all the interrupts in 8085 after reset.

EI : Enable interrupts

MVI A, 08H : Unmask the interrupts

SIM : Set the mask and unmask using SIM instruction

Timing of Interrupts:

The interrupts are sensed by the processor one cycle before the end of execution of each instruction. An interrupt signal must be applied long enough for it to be recognized. The longest instruction of the 8085 takes 18 clock periods. So, the interrupt signal must be applied for at least 17.5 clock periods. This decides the minimum pulse width for the interrupt signal.

The maximum pulse width for the interrupt signal is decided by the condition that the interrupt signal must not be recognized once again. This is under the control of the programmer.

CHAPTER: 2

1. INTERFACING MEMORY AND I/O DEVICES WITH 8085

The programs and data that are executed by the microprocessor have to be stored in ROM/EPROM and RAM, which are basically semiconductor memory chips. The programs and data that are stored in ROM/EPROM are not erased even when power supply to the chip is removed. Hence, they are called non-volatile memory. They can be used to store permanent programs.

In a RAM, stored programs and data are erased when the power supply to the chip is removed. Hence, RAM is called volatile memory. RAM can be used to store programs and data that include, programs written during software development for a microprocessor based system, program written when one is learning assembly language programming and data enter while testing these programs.

Input and output devices, which are interfaced with 8085, are essential in any microprocessor based system. They can be interfaced using two schemes: I/O mapped I/O and memory-mapped I/O. In the I/O mapped I/O scheme, the I/O devices are treated differently from memory. In the memory-mapped I/O scheme, each I/O device is assumed to be a memory location.

2. INTERFACING MEMORY CHIPS WITH 8085

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

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

Specification of IC 2764:

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

Interfacing:

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

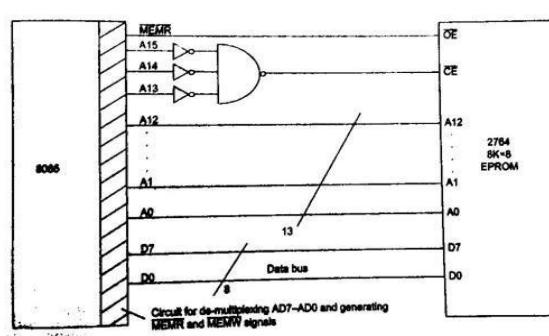


Fig. 17 Interfacing IC 2764 with the 8085

Ex: Interface a 6264 IC (8K x 8 RAM) with the 8085 using NAND gate decoder such that the starting address assigned to the chip is 4000H.

Specification of IC 6264:

- 8K x 8 RAM
- 8 KB = 2^{13} bytes
- 13 address lines

The ending address of the chip is 5FFFH (since 4000H + 1FFFH = 5FFFH). When the address 4000H to 5FFFH are written in binary form, the values in the lines A15, A14, A13 are 0, 1 and 0 respectively. The NAND gate is designed such that when the lines A15 and A13 carry 0 and A14 carries 1, the output of the NAND gate is 0. The NAND gate output is in turn connected to the CE1 pin of the RAM chip. A NAND output of 0 selects the RAM chip for read or write operation, since CE2 is already 1 because of its connection to +5V. Fig. 18 shows the interfacing of IC 6264 with the 8085.

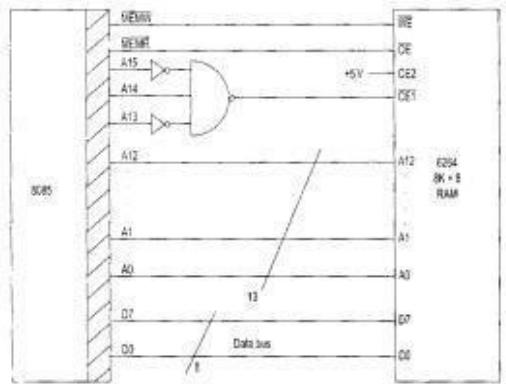


Fig. 18 Interfacing 6264 IC with the 8085

Ex: Interface two 6116 ICs with the 8085 using 74LS138 decoder such that the starting addresses assigned to them are 8000H and 9000H, respectively.

Specification of IC 6116:

- 2 K x 8 RAM
- 2 KB = 2^{11} bytes
- 11 address lines

6116 has 11 address lines and since 2 KB, therefore ending addresses of 6116 chip 1 is and chip 2 are 87FFH and 97FFH, respectively. Table 10 shows the address range of the two chips.

Table 10 Address range for IC 6116

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H
.
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	87FFH (RAM chip 1)
.
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9000H
.
1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	97FFH (RAM chip 2)
.

Interfacing:

- Fig. 19 shows the interfacing.
- A0 – A10 lines of 8085 are connected to 11 address lines of the RAM chips.
- Three address lines of 8085 having specific value for a particular RAM are connected to the three select inputs (C, B and A) of 74LS138 decoder.
- Table 10 shows that A13=A12=A11=0 for the address assigned to RAM 1 and A13=0, A12=1 and A11=0 for the address assigned to RAM 2.
- Remaining lines of 8085 which are constant for the address range assigned to the two RAM are connected to the enable inputs of decoder.
- When 8085 places any address between 8000H and 87FFH in the address bus, the select inputs C, B and A of the decoder are all 0. The Y0 output of the decoder is also 0, selecting RAM 1.
- When 8085 places any address between 9000H and 97FFH in the address bus, the select inputs C, B and A of the decoder are 0, 1 and 0. The Y2 output of the decoder is also 0, selecting RAM 2.

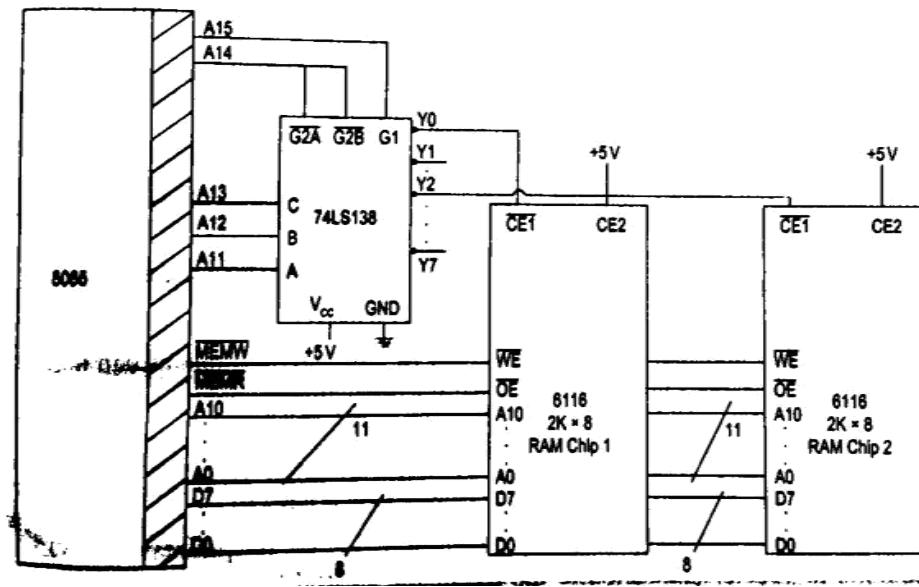


Fig. 19 Interfacing two 6116 RAM chips using 74LS138 decoder

3. PERIPHERAL MAPPED I/O INTERFACING

In this method, the I/O devices are treated differently from memory chips. The control signals I/O read (IOR) and I/O write (IOW), which are derived from the IO/M, RD and WR signals of the 8085, are used to activate input and output devices, respectively.

Generation of these control signals is shown in Fig. 20. Table 11 shows the status of IO/M, RD and WR signals during I/O read and I/O write operation.

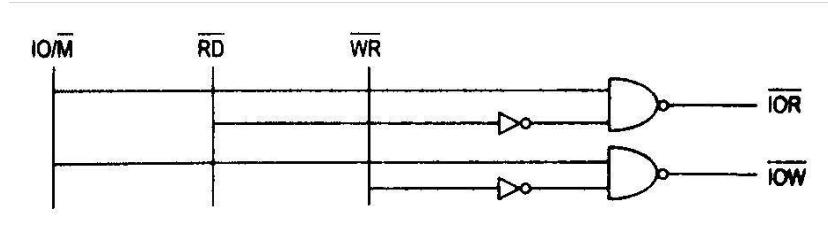


Fig. 20 Generation of \overline{IOR} and \overline{IOW} signals

IN instruction is used to access input device and OUT instruction is used to access output device. Each I/O device is identified by a unique 8-bit address assigned to it. Since the control signals used to access input and output devices are different, and all I/O device use 8-bit address, a maximum of 256 (2^8) input devices and 256 output devices can be interfaced with 8085.

Table 11 Status of \overline{IOR} and \overline{IOW} signals in 8085.

$\overline{IO/M}$	\overline{RD}	\overline{WR}	\overline{IOR}	\overline{IOW}	Operation
1	0	1	0	1	I/O read operation
1	1	0	1	0	I/O write operation
0	X	X	1	1	Memory read or write operation

Ex: Interface an 8-bit DIP switch with the 8085 such that the address assigned to the DIP switch is F0H.

IN instruction is used to get data from DIP switch and store it in accumulator. Steps involved in the execution of this instruction are:

- i. Address F0H is placed in the lines A0 – A7 and a copy of it in lines A8 – A15.
- ii. The IOR signal is activated ($\overline{\text{IOR}} = 0$), which makes the selected input device to place its data in the data bus.
- iii. The data in the data bus is read and stored in the accumulator.

Fig. 21 shows the interfacing of DIP switch.

A7	A6	A5	A4	A3	A2	A1	A0	
1	1	1	1	0	0	0	0	= F0H

A0 – A7 lines are connected to a NAND gate decoder such that the output of NAND gate is 0. The output of NAND gate is ORed with the IOR signal and the output of OR gate is connected to $\overline{1G}$ and $\overline{2G}$ of the 74LS244. When 74LS244 is enabled, data from the DIP switch is placed on the data bus of the 8085. The 8085 read data and store in the accumulator. Thus data from DIP switch is transferred to the accumulator.

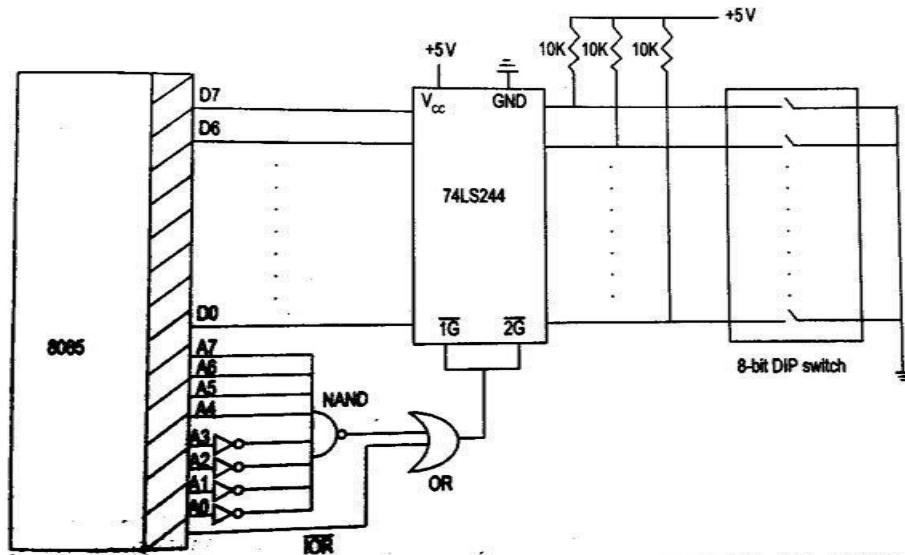


Fig. 21 interfacing of 8-bit DIP switch with 8085

4. MEMORY MAPPED I/O INTERFACING

In memory-mapped I/O, each input or output device is treated as if it is a memory location.

The $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ control signals are used to activate the devices. Each input or output device is identified by unique 16-bit address, similar to 16-bit address assigned to memory location. All memory related instruction like LDA 2000H, LDAX B, MOV A, M can be used.

Since the I/O devices use some of the memory address space of 8085, the maximum memory capacity is lesser than 64 KB in this method.

Ex: Interface an 8-bit DIP switch with the 8085 using logic gates such that the address assigned to it is F0F0H.

Since a 16-bit address has to be assigned to a DIP switch, the memory-mapped I/O technique must be used. Using LDA F0F0H instruction, the data from the 8-bit DIP switch can be transferred to the accumulator. The steps involved are:

- i. The address F0F0H is placed in the address bus A0 – A15.
- ii. The MEMR signal is made low for some time.
- iii. The data in the data bus is read and stored in the accumulator.

Fig. 22 shows the interfacing diagram.

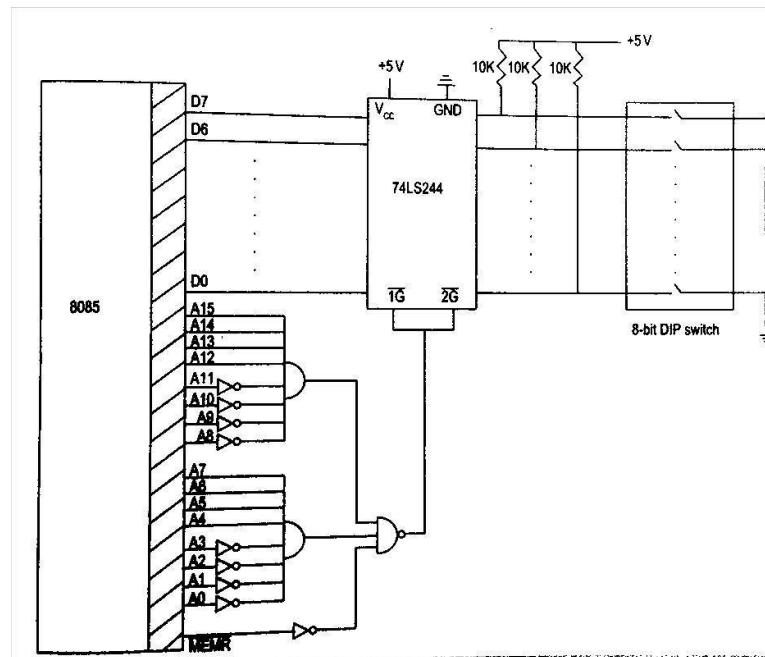


Fig. 22 Interfacing 8-bit DIP switch with 8085

When 8085 executes the instruction LDA F0F0H, it places the address F0F0H in the address lines A0 – A15 as:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0 = F0F0H

The address lines are connected to AND gates. The output of these gates along with MEMR signal are connected to a NAND gate, so that when the address F0F0H is placed in the

address bus and MEMR = 0 its output becomes 0, thereby enabling the buffer 74LS244. The data from the DIP switch is placed in the 8085 data bus. The 8085 reads the data from the data bus and stores it in the accumulator.

CHAPTER 3

8085 Assembly Language Programs & Explanations

1. Statement: Store the data byte 32H into memory location 4000H.

Program 1:

<i>MVI A, 32H</i>	: Store 32H in the accumulator
<i>STA 4000H</i>	: Copy accumulator contents at address 4000H
<i>HLT</i>	: Terminate program execution

Program 2:

<i>LXI H</i>	: Load HL with 4000H
<i>MVI M</i>	: Store 32H in memory location pointed by HL register pair
(4000H)	
<i>HLT</i>	: Terminate program execution

2. Statement: Exchange the contents of memory locations 2000H and 4000H

Program 1:

<i>LDA 2000H</i>	: Get the contents of memory location 2000H into accumulator
<i>MOV B, A</i>	: Save the contents into B register
<i>LDA 4000H</i>	: Get the contents of memory location 4000H into accumulator
<i>STA 2000H</i>	: Store the contents of accumulator at address 2000H
<i>MOVA, B</i>	: Get the saved contents back into A register
<i>STA 4000H</i>	: Store the contents of accumulator at address 4000H

Program 2:

<i>LXI H 2000H</i>	: Initialize HL register pair as a pointer to memory location 2000H.
<i>LXID 4000H</i>	: Initialize DE register pair as a pointer to memory location 4000H.
<i>MOV B, M</i>	: Get the contents of memory location 2000H into B register.
<i>LDAX D</i>	: Get the contents of memory location 4000H into A register.
<i>MOV M, A</i>	: Store the contents of A register into memory location 2000H.
<i>MOVA, B</i>	: Copy the contents of B register into accumulator.
<i>STAX D</i>	: Store the contents of A register into memory location 4000H.
<i>HLT</i>	: Terminate program execution.

3. Sample problem

$(4000H) = 14H$
 $(4001H) = 89H$
Result = $14H + 89H = 9DH$ Source program

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

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

Program - 4: Subtract two 8-bit numbers

Sample problem:

$(4000H) = 51H$
 $(4001H) = 19H$
Result = $51H - 19H = 38H$

Source program:

<i>LXI H, 4000H</i>	: HL points 4000H
<i>MOVA, M</i>	: Get first operand
<i>INX H</i>	: HL points 4001H
<i>SUB M</i>	: Subtract second operand
<i>INX H</i>	: HL points 4002H
<i>MOV M, A</i>	: Store result at 4002H.
<i>HLT</i>	: Terminate program execution

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

Program - 5.a: Add two 16-bit numbers - Source Program 1

Sample problem:

(4000H) = 15H
(4001H) = 1CH
(4002H) = B7H
(4003H) = 5AH
Result = 1C15 + 5AB7H = 76CCH (4004H) =
CCH
(4005H) = 76H

Source Program 1:

LHLD 4000H	: Get first 16-bit number in HL
XCHG	: Save first 16-bit number in DE
LHLD 4002H	: Get second 16-bit number in HL
MOV A, E	: Get lower byte of the first number
ADD L	: Add lower byte of the second number
MOV L, A	: Store result in L register
MOV A, D	: Get higher byte of the first number
ADC H	: Add higher byte of the second number with CARRY
MOV H, A	: Store result in H register
SHLD 4004H	: Store 16-bit result in memory locations 4004H and
4005H.	
HLT	: Terminate program execution

6. Statement: Add the contents of memory locations 40001H and 4001H and place the result in the memory locations 4002H and 4003H.

Sample problem:

(4000H) = 7FH
(4001H) = 89H
Result = 7FH + 89H = 108H (4002H) =
08H (4003H) = 01H

Source program:

LXI H, 4000H	: HL Points 4000H
MOVA, M	: Get first operand
INX H	: HL Points 4001H
ADD M	: Add second operand
INX H	: HL Points 4002H
MOV M, A	: Store the lower byte of result at 4002H
MVIA, 00	: Initialize higher byte result with 00H

<i>ADCA</i>	<i>:Add carry in the high byte result</i>
<i>INX H</i>	<i>:HL Points 4003H</i>
<i>MOV M, A</i>	<i>:Store the higher byte of result at 4003H</i>
<i>HLT</i>	<i>:Terminate program execution</i>

7. Statement: Subtract the 16-bit number in memory locations 4002H and 4003H from the 16-bit number in memory locations 4000H and 4001H. The most significant eight bits of the two numbers are in memory locations 4001H and 4003H. Store the result in memory locations 4004H and 4005H with the most significant byte in memory location 4005H.

Sample problem

$(4000H) = 19H$
 $(4001H) = 6AH$
 $(4004H) = 15H$ $(4003H) = 5CH$ Result = $6A19H - 5C15H = OE04H$ $(4004H) = 04H$
 $(4005H) = OEH$

Source program:

<i>LHLD 4000H</i>	<i>: Get first 16-bit number in HL</i>
<i>XCHG</i>	<i>: Save first 16-bit number in DE</i>
<i>LHLD 4002H</i>	<i>: Get second 16-bit number in HL</i>
<i>MOV A, E</i>	<i>: Get lower byte of the first number</i>
<i>SUB L</i>	<i>: Subtract lower byte of the second number</i>
<i>MOV L, A</i>	<i>: Store the result in L register</i>
<i>MOV A, D</i>	<i>: Get higher byte of the first number</i>
<i>SBB H</i>	<i>: Subtract higher byte of second number with borrow</i>
<i>MOV H, A</i>	<i>: Store 16-bit result in memory locations 4004H and 4005H.</i>
<i>SHLD 4004H</i>	<i>: Store 16-bit result in memory locations 4004H and 4005H.</i>
<i>HLT</i>	<i>: Terminate program execution</i>

8. Statement: Find the I's complement of the number stored at memory location 4400H and store the complemented number at memory location 4300H.

Sample problem:

$(4400H) = 55H$

Result = (4300B) = AAB

Source program:

<i>LDA 4400B</i>	<i>: Get the number</i>
<i>CMA</i>	<i>: Complement number</i>
<i>STA 4300H</i>	<i>: Store the result</i>
<i>HLT</i>	<i>: Terminate program execution</i>

9. Statement: Find the 2's complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.

Sample problem:

$$(4200H) = 55H$$
$$\text{Result} = (4300H) = \text{AAH} + 1 = \text{ABH}$$

Source program:

<i>LDA 4200H</i>	<i>: Get the number</i>
<i>CMA</i>	<i>: Complement the number</i>
<i>ADI, 01 H</i>	<i>: Add one in the number</i>
<i>STA 4300H</i>	<i>: Store the result</i>
<i>HLT</i>	<i>: Terminate program execution</i>

10. Statement: Pack the two unpacked BCD numbers stored in memory locations 4200H and 4201H and store result in memory location 4300H. Assume the least significant digit is stored at 4200H.

Sample problem: (4200H)

$$= 04 (4201H) = 09$$
$$\text{Result} = (4300H) = 94$$

Source program

<i>LDA 4201H</i>	<i>: Get the Most significant BCD digit</i>
<i>RLC</i>	
<i>RLC</i>	
<i>RLC</i>	
<i>RLC</i>	<i>: Adjust the position of the second digit (09 is changed to 90)</i>

<i>ANI FOH</i>	<i>: Make least significant BCD digit zero</i>
<i>MOV C, A</i>	<i>: store the partial result</i>
<i>LDA 4200H</i>	<i>: Get the lower BCD digit</i>
<i>ADD C</i>	<i>: Add lower BCD digit</i>
<i>STA 4300H</i>	<i>: Store the result</i>
<i>HLT</i>	<i>: Terminate program execution</i>

11. Statement: Two digit BCD number is stored in memory location 4200H. Unpack the BCD number and store the two digits in memory locations 4300H and 4301H such that memory location 4300H will have lower BCD digit.

Sample problem

$$(4200H) = 58 \\ \text{Result} = (4300H) = 08 \text{ and } (4301H) = 05$$

Source program

<i>LDA 4200H</i>	<i>: Get the packed BCD number</i>
<i>ANI FOH</i>	<i>: Mask lower nibble</i>
<i>RRC</i>	
<i>RRC</i>	
<i>RRC</i>	
<i>RRC</i>	<i>: Adjust higher BCD digit as a lower digit</i>
<i>STA 4301H</i>	<i>: Store the partial result</i>
<i>LDA 4200H</i>	<i>: Get the original BCD number</i>
<i>ANI OFH</i>	<i>: Mask higher nibble</i>
<i>STA 4201H</i>	<i>: Store the result</i>
<i>HLT</i>	<i>: Terminate program execution</i>

12. Statement: Read the program given below and state the contents of all registers after the execution of each instruction in sequence.

Main program:

<i>4000H</i>	<i>LXI SP, 27FFH</i>
<i>4003H</i>	<i>LXI H, 2000H</i>
<i>4006H</i>	<i>LXI B, 1020H</i>
<i>4009H</i>	<i>CALL SUB</i>
<i>400CH</i>	<i>HLT</i>

Subroutine program:

4100H	SUB: PUSH B
4101H	PUSH H
4102H	LXI B, 4080H
4105H	LXI H, 4090H
4108H	SHLD 2200H
4109H	DAD B
410CH	POP H
410DH	POP B
410EH	RET

13. Statement: Write a program to shift an eight bit data four bits right. Assume that data is in register C.

Source program:

MOVA, C
RAR
RAR
RAR
RAR
MOV C, A
HLT

14. Statement: Program to shift a 16-bit data 1 bit left. Assume data is in the HL register pair

Source program:

DAD H : Adds HL data with HL data

15. Statement: Write a set of instructions to alter the contents of flag register in 8085.

PUSH PSW	: Save flags on stack
POP H	: Retrieve flags in 'L'
MOVA, L	: Flags in accumulator
CMA	: Complement accumulator
MOV L, A	: Accumulator in 'L'

<i>PUSH H</i>	: Save on stack
<i>POP PSW</i>	: Back to flag register
<i>HLT</i>	: Terminate program execution

16. Statement: Calculate the sum of series of numbers. The length of the series is in memory location 4200H and the series begins from memory location 4201H.

1. Consider the sum to be 8 bit number. So, ignore carries. Store the sum at memory location 4300H.
2. Consider the sum to be 16 bit number. Store the sum at memory locations 4300H and 4301H

a. Sample problem

```

4200H = 04H
4201H = 10H
4202H = 45H
4203H = 33H
4204H = 22H
Result = 10 + 41 + 30 + 12 = H
4300H = H

```

Source program:

```

LDA 4200H
MOV C, A           : Initialize counter
SUB A             : sum = 0
LXI H, 4201H       : Initialize pointer
BACK:    ADD M      : SUM = SUM + data
INX H             : increment pointer
DCR C             : Decrement counter
JNZ BACK          : if counter 0 repeat
STA 4300H          : Store sum
HLT                : Terminate program execution

```

b. Sample problem

```

4200H = 04H 4201H
= 9AH 4202H = 52H
4203H = 89H 4204H
= 3EH
Result = 9AH + 52H + 89H + 3EH = H 4300H = B3H
Lower byte
4301H = 0lH Higher byte

```

Source program:

<i>LDA 4200H</i>	
<i>MOV C, A</i>	: Initialize counter
<i>LXI H, 4201H</i>	: Initialize pointer
<i>SUB A</i>	: Sum low = 0
<i>MOV B, A</i>	: Sum high = 0
<i>BACK: ADD M</i>	: Sum = sum + data
<i>JNC SKIP</i>	
<i>INR B</i>	: Add carry to MSB of SUM
<i>SKIP: INX H</i>	: Increment pointer
<i>DCR C</i>	: Decrement counter
<i>JNZ BACK</i>	: Check if counter 0 repeat
<i>STA 4300H</i>	: Store lower byte
<i>MOVA, B</i>	
<i>STA 4301H</i>	: Store higher byte
<i>HLT</i>	: Terminate program execution

17. Statement: Multiply two 8-bit numbers stored in memory locations 2200H and 2201H by repetitive addition and store the result in memory locations 2300H and 2301H.

Sample problem:

$(2200H) = 03H$
 $(2201H) = B2H$
 Result = $B2H + B2H + B2H = 216H = 216H$
 $(2300H) = 16H$
 $(2301H) = 02H$

Source program

<i>LDA 2200H</i>	
<i>MOVE, A</i>	
<i>MVI D, 00</i>	: Get the first number in DE register pair
<i>LDA 2201H</i>	
<i>MOV C, A</i>	: Initialize counter
<i>LXI H, 0000 H</i>	: Result = 0
<i>BACK: DAD D</i>	: Result = result + first number
<i>DCR C</i>	: Decrement count
<i>JNZ BACK</i>	: If count 0 repeat
<i>SHLD 2300H</i>	: Store result
<i>HLT</i>	: Terminate program execution

18. Statement: Divide 16 bit number stored in memory locations 2200H and 2201H by the 8 bit number stored at memory location 2202H. Store the quotient in memory locations 2300H and 2301H and remainder in memory locations 2302H and 2303H.

*Sample problem (2200H) =
60H (2201H) = A0H
(2202H) = 12H
Result = A060H/12H = 8E8H Quotient and 10H remainder (2300H) = E8H
(2301H) = 08H
(2302H= 10H (2303H)
00H*

Source program

<i>LHLD 2200H</i>	<i>: Get the dividend</i>
<i>LDA 2202H</i>	<i>: Get the divisor</i>
<i>MOV C, A</i>	
<i>LXI D, 0000H</i>	<i>: Quotient = 0</i>
<i>BACK: MOVA, L</i>	
<i>SUB C</i>	<i>: Subtract divisor</i>
<i>MOVL, A</i>	<i>: Save partial result</i>
<i>JNC SKIP</i>	<i>: if CY 1 jump</i>
<i>DCR H</i>	<i>: Subtract borrow of previous subtraction</i>
<i>SKIP: INX D</i>	<i>: Increment quotient</i>
<i>MOVA, H</i>	
<i>CPI, 00</i>	<i>: Check if dividend < divisor</i>
<i>JNZ BACK</i>	<i>: if no repeat</i>
<i>MOVA, L</i>	
<i>CMP C</i>	
<i>JNC BACK</i>	
<i>SHLD 2302H</i>	<i>: Store the remainder</i>
<i>XCHG</i>	
<i>SHLD 2300H</i>	<i>: Store the quotient</i>
<i>HLT</i>	<i>: Terminate program execution</i>

19. Statement: Find the number of negative elements (most significant bit 1) in a block of data. The length of the block is in memory location 2200H and the block itself begins in memory location 2201H. Store the number of negative elements in memory location 2300H

Sample problem

(2200H) = 04H

$(2201H) = 56H$
 $(2202H) = A9H$
 $(2203H) = 73H$
 $(2204H) = 82H$

Result = 02 since 2202H and 2204H contain numbers with a MSB of 1.

Source program

```
LDA 2200H          : Initialize count
MOV C, A           : Negative number = 0
MVI B, 00          : Initialize pointer
LXI H, 2201H       : Get the number
BACK: MOVA, M      : Check for MSB
ANI 80H            : If MSB = 1
JZ SKIP             : Increment negative number count
INR B              : Increment pointer
SKIP: INX H         : Decrement count
DCR C              : If count 0 repeat
JNZ BACK            : Store the result
MOVA, B             : Terminate program execution
STA 2300H
HLT
```

20. Statement: Find the largest number in a block of data. The length of the block is in memory location 2200H and the block itself starts from memory location 2201H.

Store the maximum number in memory location 2300H. Assume that the numbers in the block are all 8 bit unsigned binary numbers.

Sample problem

$(2200H) = 04$
 $(2201H) = 34H$
 $(2202H) = A9H$
 $(2203H) = 78H$
 $(2204H) = 56H$

Result = (2202H) = A9H

Source program

```
LDA 2200H          : Initialize counter
MOV C, A           : Maximum = Minimum possible value = 0
XRA A              : Initialize pointer
LXI H, 2201H       : Is number > maximum
BACK: CMP M         : Yes, replace maximum
JNC SKIP            : JNC SKIP
```

```

    MOVA, M
SKIP: INX H
    DCR C
    JNZ BACK
    STA 2300H      : Store maximum number
    HLT            : Terminate program execution

```

21. Statement: Write a program to count number of 1's in the contents of D register and store the count in the B register.

Source program:

```

MVI B, 00H
MVI C, 08H
MOVA, D
BACK: RAR
    JNC SKIP
    INR B
SKIP: DCR C
    JNZ BACK
    HLT

```

22. Statement: Write a program to sort given 10 numbers from memory location 2200H in the ascending order.

Source program:

```

MVI B, 09      : Initialize counter
START          : LXI H, 2200H: Initialize memory pointer
MVI C, 09H     : Initialize counter 2
BACK: MOVA, M
    INX H       : Get the number
    CMP M       : Increment memory pointer
    JC SKIP     : Compare number with next number
    JZ SKIP     : If less, don't interchange
    MOV D, M
    MOV M, A
    DCX H
    MOV M, D
    INX H       : If equal, don't interchange
    SKIP: DCR C : Interchange two numbers
    JNZ BACK   : Decrement counter 2
    DCR B       : If not zero, repeat
    JNZ START   : Decrement counter 1
    HLT         : Terminate program execution

```

23. Statement: Calculate the sum of series of even numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 8 bit number so you can ignore carries and store the sum at memory location 2210H. *Sample problem:*

2200H= 4H
2201H= 20H
2202H= 15H
2203H= 13H
2204H= 22H
Result 2210H= 20 + 22 = 42H = 42H

Source program:

```
LDA 2200H
MOV C, A
MVI B, 00H
LXI H, 2201H
BACK: MOVA, M
ANI      01H
JNZ SKIP
MOVA, B
ADD      M
MOV B, A
SKIP: INX H
DCR      C
JNZ      BACK
STA      2210H
HLT
```

24. Statement: Calculate the sum of series of odd numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 16-bit. Store the sum at memory locations 2300H and 2301H.

Sample problem:

2200H = 4H
2201H= 9AH
2202H= 52H
2203H= 89H
2204H= 3FH
Result = 89H + 3FH = C8H 2300H= H
Lower byte 2301H = H Higher byte

Source program

```
LDA 2200H          : Initialize counter
MOV C, A           :
LXI H, 2201H       : Initialize pointer
MVI E, 00          : Sum low = 0
MOVD, E            : Sum high = 0
BACK: MOVA, M      : Get the number
ANI 01H             : Mask Bit 1 to Bit7
JZ SKIP             : Don't add if number is even
MOVA, E             : Get the lower byte of sum
ADD M               : Sum = sum + data
MOV E, A             : Store result in E register
JNC SKIP             :
INR D               : Add carry to MSB of SUM
SKIP: INX H          : Increment pointer
DCR C               : Decrement
```

25. Statement:Find the square of the given numbers from memory location 6100H and store the result from memory location 7000H

Source Program:

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

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

Source program:

<i>LXI H, 2000H</i>	: Initialize memory pointer 52H
<i>MVI B, 52H</i>	: Initialize counter
<i>BACK: MOVA, M</i>	: Get the number
<i>CMP C</i>	: Compare with the given byte
<i>JZ LAST</i>	: Go last if match occurs
<i>INX H</i>	: Increment memory pointer
<i>DCR B</i>	: Decrement counter
<i>JNZ B</i>	: If not zero, repeat
<i>LXI H, 0000H</i>	
<i>SHLD 2200H</i>	
<i>JMP END</i>	: Store 00 at 2200H and 2201H
<i>LAST: SHLD 2200H</i>	: Store memory address
<i>END: HLT</i>	: Stop

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

Source Program:

<i>LXI H, 6000H</i>	: Initialize pointer l to first number
<i>LXI D, 6100H</i>	: Initialize pointer2 to second number
<i>LXI B, 6200H</i>	: Initialize pointer3 to result
<i>STC</i>	
<i>CMC</i>	: Carry = 0
<i>BACK: LDAX D</i>	: Get the digit
<i>ADD M</i>	: Add two digits
<i>DAA</i>	: Adjust for decimal
<i>STAX.B</i>	: Store the result
<i>INX H</i>	: Increment pointer 1
<i>INX D</i>	: Increment pointer2
<i>INX B</i>	: Increment result pointer
<i>MOVA, L</i>	
<i>CPI 06H</i>	: Check for last digit
<i>JNZ BACK</i>	: If not last digit repeat
<i>HLT</i>	: Terminate program execution

28. Statement: Add 2 arrays having ten 8-bit numbers each and generate a third array of result. It is necessary to add the first element of array 1 with the first

element of array-2 and so on. The starting addresses of array l, array2 and array3 are 2200H, 2300H and 2400H, respectively.

Source Program:

<i>LXI H, 2200H</i>	<i>: Initialize memory pointer 1</i>
<i>LXI B, 2300H</i>	<i>: Initialize memory pointer 2</i>
<i>LXI D, 2400H</i>	<i>: Initialize result pointer</i>
<i>BACK: LDAX B</i>	<i>: Get the number from array 2</i>
<i>ADD M</i>	<i>: Add it with number in array 1</i>
<i>STAX D</i>	<i>: Store the addition in array 3</i>
<i>INX H</i>	<i>: Increment pointer 1</i>
<i>INX B</i>	<i>: Increment pointer2</i>
<i>INX D</i>	<i>: Increment result pointer</i>
<i>MOVA, L</i>	
<i>CPI 0AH</i>	<i>: Check pointer 1 for last number</i>
<i>JNZ BACK</i>	<i>: If not, repeat</i>
<i>HLT</i>	<i>: Stop</i>

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

Source Program:

<i>LXI H, 2200H</i>	<i>: Initialize memory pointer 1</i>
<i>LXI D, 2300H</i>	<i>: Initialize memory pointer2</i>
<i>MVI C, 32H</i>	<i>: Initialize counter</i>
<i>BACK: MOVA, M</i>	<i>: Get the number</i>
<i>ANI 0IH</i>	<i>: Check for even number</i>
<i>JNZ SKIP</i>	<i>: If ODD, don't store</i>
<i>MOVA, M</i>	<i>: Get the number</i>
<i>STAX D</i>	<i>: Store the number in result list</i>
<i>INX D</i>	<i>: Increment pointer 2</i>
<i>SKIP: INX H</i>	<i>: Increment pointer 1</i>
<i>DCR C</i>	<i>: Decrement counter</i>
<i>JNZ BACK</i>	<i>: If not zero, repeat</i>
<i>HLT</i>	<i>: Stop</i>

30. Statement: Write assembly language program with proper comments for the following:

A block of data consisting of 256 bytes is stored in memory starting at 3000H. This block is to be shifted (relocated) in memory from 3050H onwards. Do not shift the block or part of the block anywhere else in the memory.

Source Program:

Two blocks (3000 - 30FF and 3050 - 314F) are overlapping. Therefore it is necessary to transfer last byte first and first byte last.

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

31. Statement: Add even parity to a string of 7-bit ASCII characters. The length of the string is in memory location 2040H and the string itself begins in memory location 2041H. Place even parity in the most significant bit of each character.

Source Program:

<i>LXI H, 2040H</i>	
<i>MOV C ,M</i>	: Counter for character
<i>REPEAT: INX H</i>	: Memory pointer to character
<i>MOVA,M</i>	: Character in accumulator
<i>ORA A</i>	: ORing with itself to check parity.
<i>JPO PAREVEN</i>	: If odd parity place
<i>ORI 80H</i>	even parity in D7 (80).
<i>PAREVEN: MOV M , A</i>	: Store converted even parity character.
<i>DCR C</i>	: Decrement counter.
<i>JNZ REPEAT</i>	: If not zero go for next character.
<i>HLT</i>	

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

Source Program:

```
LXI H, 6000H      : Initialize memory pointer
MVI C, 00H       : Initialize number counter
MVI B, 00H       : Initialize negative number counter
MVI E, 00H       : Initialize zero number counter
BEGIN:MOVA, M    : Get the number
    CPI 00H
    JZ ZERONUM
    ANI     80H
    JNZ NEGNUM
    INR D
    JMP LAST
ZERONUM:INR E   : Increment zero number counter
    JMP LAST
NEGNUM:INR B   : Increment negative number counter
LAST:INX H      : Increment memory pointer
    INR C
    MOVA, C
    CPI 32H      : If number counter = 5010 then
    JNZ BEGIN
    LXI H, 7000
    MOV M, B      : Store      otherwise check next number
    INX H          : Initialize memory pointer.
    MOV M, E      : Store      negative number.
    INX H          : Store      zero number.
    MOV M, D      : Store      positive number.
    HLT            : Terminate execution
```

33. Statement: Write an 8085 assembly language program to insert a string of four characters from the tenth location in the given array of 50 characters

Solution:

Step 1: Move bytes from location 10 till the end of array by four bytes downwards.
Step 2: Insert four bytes at locations 10, 11, 12 and 13.

Source Program:

```
LXI H, 2131H      : Initialize pointer at the last location of array.
LXI D, 2135H      : Initialize another pointer to point the last
location of array after insertion.
AGAIN: MOVA, M    : Get the character
```

<i>STAX D</i>	: Store at the new location
<i>DCX D</i>	: Decrement destination pointer
<i>DCX H</i>	: Decrement source pointer
<i>MOVA, L</i>	: [check whether desired bytes are shifted or not]
<i>CPI 05H</i>	: if not repeat the process
<i>JNZ AGAIN</i>	: adjust the memory pointer
<i>INX H</i>	
<i>LXI D, 2200H</i>	: Initialize the memory pointer to point the string to be inserted
<i>REPE: LDAX D</i>	: Get the character
<i>MOV M, A</i>	: Store it in the array
<i>INX D</i>	: Increment source pointer
<i>INX H</i>	: Increment destination pointer
<i>MOVA, E</i>	: [Check whether the 4 bytes are inserted]
<i>CPI 04</i>	: if not repeat the process
<i>JNZ REPE</i>	
<i>HLT</i>	: stop

34. Statement: Write an 8085 assembly language program to delete a string of 4 characters from the tenth location in the given array of 50 characters.

Solution: Shift bytes from location 14 till the end of array upwards by 4 characters i.e. from location 10 onwards.

Source Program:

<i>LXI H, 210DH</i>	: Initialize source memory pointer at the 14thlocation of the array.
<i>LXI D, 2109H</i>	: Initialize destn memory pointer at the 10th location of the array.
<i>MOVA, M</i>	: Get the character
<i>STAX D</i>	: Store character at new location
<i>INX D</i>	: Increment destination pointer
<i>INX H</i>	: Increment source pointer
<i>MOVA, L</i>	: [check whether desired bytes are shifted or not]
<i>CPI 32H</i>	: if not repeat the process
<i>JNZ REPE</i>	
<i>HLT</i>	: stop

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

Sample problem:

(2200)	= 1100 (0CH)
(2201)	= 0101 (05H)
Multiplicand	= 1100 (1210)
Multiplier	= 0101 (510)
Result	= $12 \times 5 = (6010)$

Source program

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

36. Statement: Divide the 16-bit unsigned number in memory locations 2200H and 2201H (most significant bits in 2201H) by the B-bit unsigned number in memory location 2300H store the quotient in memory location 2400H and remainder in 2401H

Assumption: The most significant bits of both the divisor and dividend are zero.

Source program

MVI E, 00	: Quotient = 0
LHLD 2200H	: Get dividend
LDA 2300	: Get divisor
MOV B, A	: Store divisor
MVI C, 08	: Count = 8
NEXT: DAD H	: Dividend = Dividend x 2
MOVA, E	
RLC	
MOV E, A	: Quotient = Quotient x 2

<i>MOVA, H</i>	
<i>SUB B</i>	: Is most significant byte of Dividend > divisor
<i>JC SKIP</i>	: No, go to Next step
<i>MOVH, A</i>	: Yes, subtract divisor
<i>INR E</i>	: and Quotient = Quotient + 1
<i>SKIP: DCR C</i>	: Count = Count - 1
<i>JNZ NEXT</i>	: Is count = 0 repeat
<i>MOVA, E</i>	
<i>STA 2401H</i>	: Store Quotient
<i>Mov A, H</i>	
<i>STA 2410H</i>	: Store remainder
<i>HLT</i>	: End of program

37. DAA instruction is not present. Write a sub routine which will perform the same task as DAA.

Sample Problem:

Execution of DAA instruction:

- If the value of the low order four bits (03-00) in the accumulator is greater than 9 or if auxiliary carry flag is set, the instruction adds 6 '(06) to the low-order four bits.
- If the value of the high-order four bits (07-04) in the accumulator is greater than 9 or if carry flag is set, the instruction adds 6(06) to the high-order four bits.

Source Program:

<i>LXI SP, 27FFH</i>	: Initialize stack pointer
<i>MOVE, A</i>	: Store the contents of accumulator
<i>ANI 0FH</i>	: Mask upper nibble
<i>CPI 0A H</i>	: Check if number is greater than 9
<i>JC SKIP</i>	: if no go to skip
<i>MOVA, E</i>	: Get the number
<i>ADI 06H</i>	: Add 6 in the number
<i>JMP SECOND</i>	: Go for second check
<i>SKIP: PUSH PSW</i>	: Store accumulator and flag contents in stack
<i>POP B</i>	: Get the contents of accumulator in B register and
<i>flag register contents in</i>	<i>C register</i>
<i>MOVA, C</i>	: Get flag register contents in accumulator
<i>ANI 10H</i>	: Check for bit 4
<i>JZ SECOND</i>	: if zero, go for second check
<i>MOVA, E</i>	: Get the number
<i>ADI 06</i>	: Add 6 in the number
<i>SECOND: MOVE, A</i>	: Store the contents of accumulator
<i>ANI FOH</i>	: Mask lower nibble
<i>RRC</i>	
<i>RRC</i>	
<i>RRC</i>	

```

RRC           : Rotate number 4 bit right
CPI 0AH       : Check if number is greater than 9
JC SKIP1      : if no go to skip 1
MOVA, E       : Get the number
ADI 60 H      : Add 60 H in the number
JMP LAST      : Go to last
SKIP1: JNC LAST : if carry flag = 0 go to last
MOVA, E       : Get the number
ADI 60 H      : Add 60 H in the number
LAST: HLT

```

38. tement: To test RAM by writing '1' and reading it back and later writing '0' (zero) and reading it back. RAM addresses to be checked are 40FFH to 40FFH. In case of any error, it is indicated by writing 01H at port 10H

Source Program:

```

LXI H, 4000H          : Initialize memory pointer
BACK: MVI M, FFH       : Writing '1' into RAM
MOVA, M               : Reading data from RAM
CPI FFH               : Check for ERROR
JNZ ERROR             : If yes go to ERROR
INX H                 : Increment memory pointer
MOVA, H               : Check for last check
CPI SOH               : If not last, repeat
JNZ BACK              : Initialize memory pointer
LXI H, 4000H          : Writing '0' into RAM
BACK1: MVI M, OOH      : Reading data from RAM
MOVA, M               : Check for ERROR
CPI OOH               : Increment memory pointer
INX H                 : Check for last check
CPI SOH               : If not last, repeat
JNZ BACK1             : Stop Execution
HLT

```

39. tement: Write an assembly language program to generate fibonacci number

Source Program:

```

MVI D, COUNT MVI
B, 00 MVI C, 01

```

Initialize counter
 Initialize variable to store previous number
 Initialize variable to store current number

```

    MOVA, B          :[Add two numbers]
BACK: ADD C        :[Add two numbers]
    MOV B, C          : Current number is now previous number
    MOV C, A          : Save result as a new current number
    DCR D            : Decrement count
    JNZ BACK         : if count 0 go to BACK
    HLT              : Stop

```

40. tement: Write a program to generate a delay of 0.4 sec if the crystal frequency is 5 MHz

Calculation: In 8085, the operating frequency is half of the crystal frequency,

ie. Operating frequency = $5/2 = 2.5 \text{ MHz}$

Time for one T-state =

*Number of T-states required =
= 1×10^6*

Source Program:

```

LXI B, count      : 16 - bit count
BACK: DCX B       : Decrement count
MOVA, C
ORA B             : Logically OR Band C
JNZ BACK          : If result is not zero repeat

```

41. tement: Arrange an array of 8 bit unsigned no in descending order

Source Program:

```

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

```

```
JZ START ; do another sort, if yes  
HLT ; If flag = 0, stop execution
```

42. tement: Transfer ten bytes of data from one memory to another memory block. Source memory block starts from memory location 2200H where as destination memory block starts from memory location 2300H

Source Program:

```
LXI H, 4150 ; Initialize memory pointer  
MVI B, 08 ; count for 8-bit  
MVI A, 54  
LOOP : RRC  
JC LOOP1  
MVI M, 00 ; store zero if no carry  
JMP COMMON  
LOOP2: MVI M, 01 ; store one if there is a carry  
COMMON: INX H  
DCR B ; check for carry  
JNZ LOOP  
HLT ; Terminate the program
```

43. tement: Program to calculate the factorial of a number between 0 to 8

Source program

```
LXI SP, 27FFH ; Initialize stack pointer  
LDA 2200H ; Get the number  
CPI 02H ; Check if number is greater than 1  
JC LAST  
MVI D, 00H ; Load number as a result  
MOVE, A  
DCR A  
MOV C,A ; Load counter one less than number  
CALL FACTO ; Call subroutine FACTO  
XCHG ; Get the result in HL  
SHLD 2201H ; Store result in the memory  
JMP END  
LAST: LXI H, 0001H ; Store result = 01  
END: SHLD 2201H  
HLT
```

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

Source Program:

<i>LDA 4200H</i>	: Get the given data(Y) in A register
<i>MOV B,A</i>	: Save the data in B register
<i>MVI C,02H</i>	: Call the divisor(02H) in C register
<i>CALL DIV</i>	: Call division subroutine to get initial value(X)
<i>in D-reg</i>	
<i>REP: MOV E,D</i>	: Save the initial value in E-reg
<i>MOV A,B</i>	: Get the dividend(Y) in A-reg
<i>MOV C,D</i>	: Get the divisor(X) in C-reg
<i>CALL DIV</i>	: Call division subroutine to get initial value(Y/X) in D-reg
<i>MOV A, D</i>	: Move Y/X in A-reg
<i>ADD E</i>	: Get the((Y/X) + X) in A-reg
<i>MVI C, 02H</i>	: Get the divisor(02H) in C-reg
<i>CALL DIV</i>	: Call division subroutine to get ((Y/X) + X)/2
<i>in D-reg. This is XNEW</i>	
<i>MOVA, E</i>	: Get X in A-reg
<i>CMP D</i>	: Compare X and XNEW
<i>JNZ REP</i>	: If XNEW is not equal to X, then repeat
<i>STA 4201H</i>	: Save the square root in memory
<i>HLT</i>	: Terminate program execution

45. tement: Write a simple program to Split a HEX data into two nibbles and store it in memory

Source Program:

<i>LXI H, 4200H</i>	: Set pointer data for array
<i>MOV B,M</i>	: Get the data in B-reg
<i>MOVA,B</i>	: Copy the data to A-reg
<i>ANI OFH</i>	: Mask the upper nibble
<i>INX H</i>	: Increment address as 4201
<i>MOV M,A</i>	: Store the lower nibble in memory
<i>MOVA,B</i>	: Get the data in A-reg
<i>ANI FOH</i>	: Bring the upper nibble to lower nibble position
<i>RRC</i>	
<i>RRC</i>	
<i>RRC</i>	
<i>INX H</i>	
<i>MOV M,A</i>	: Store the upper nibble in memory
<i>HLT</i>	: Terminate program execution

46. tement: Add two 4 digit BCD numbers in HL and DE register pairs and store result in memory locations, 2300H and 2301H. Ignore carry after 16 bit.

Sample Problem:

(HL) =3629 (DE)
=4738
Step 1 : $29 + 38 = 61$ and auxiliary carry flag = 1
.add 06
 $61 + 06 = 67$
Step 2 : $36 + 47 + 0$ (carry of LSB) = 7D

Lower nibble of addition is greater than 9, so add 6. $7D + 06 = 83$
Result = 8367

Source program

MOVA, L	: Get lower 2 digits of no. I
ADD E	: Add two lower digits
DAA	: Adjust result to valid BCD
STA 2300H	: Store partial result
MOVA, H	: Get most significant 2 digits of number
ADC D	: Add two most significant digits
DAA	: Adjust result to valid BCD
STA 2301H	: Store partial result
HLT	: Terminate program execution

47. tement: Subtract the BCD number stored in E register from the number stored in the D register.

Source Program:

MVI A,99H	
SUB E	: Find the 99's complement of subtrahend
INR A	: Find 100's complement of subtrahend
ADD D	: Add minuend to 100's complement of subtrahend
DAA	: Adjust for BCD
HLT	: Terminate program execution

48. tement: Write an assembly language program to multiply 2 BCD numbers

Source Program:

MVI C, Multiplier : Load BCD multiplier
MVI B, 00 : Initialize counter
LXI H, 0000H : Result = 0000
MVI E, multiplicand : Load multiplicand
MVI D, 00H : Extend to 16-bits
BACK: *DAD D* : Result Result + Multiplicand
MOVA, L : Get the lower byte of the result
ADI, 00H
DAA
MOV L, A
MOVA, H
ACI, 00H
DAA
MOV H, A
MOVA, B
ADI 01H
DAA
MOV B,A
CMP C
JNZ BACK : Compare if count = multiplier
HLT : if not equal repeat
 : Stop

