Q.2 a. Briefly describe programmer's view of 8085 and need for accumulator. (4)

Answer:

We have seen that 8085 receives 8-bit information on AD_{7-0} from memory or an input port. When it comes inside the microprocessor, where does it reside inside the processor? For this purpose, there is what is called 'register' to store such information inside. A register is nothing but a group of flip-flops, where each flip-flop can store a bit of information. The size of such a register in 8085 has to be 8 bits, to store 8 bits of information.

It is necessary to have atleast one such register in any brand of microprocessor. The advantages of a register over a memory location are as follows.

- The contents of a register can be accessed much faster by the microprocessor, compared with the contents of a memory location (by 'access' we mean 'read or write').
- Instructions involving register operands will be shorter in length and are executed faster compared with instructions involving memory operands.

However, the disadvantages of a register over a memory location are as follows.

- During an interrupt service subroutine (ISS), the register values will have to be stored in an area of memory called stack. Upon return from the ISS, the register values have to be restored from the stack. This takes away the speed advantage that was mentioned earlier.
- If there are too many registers, they occupy lot of real estate on the chip, thus reducing the real estate available for the control unit and ALU.
- In instructions involving register operands, the number of bits that are free to specify the operation become reduced, thereby reducing the number of possible instructions.

The reader will be in a position to appreciate these advantages and disadvantages much better after a study of a few of the successive chapters.

In view of the earlier discussion, most microprocessors provide only a few registers on the chip. The registers provided on 8085 are A, B, C, D, E, H, and L. These are the general-purpose registers (GPRs). In addition to these registers there are a few special purpose registers (SPRs). These are discussed later.

2 (a). Section 4.3 Pages 34 to 35. TEXT BOOK – I.
Programmer's view of 8085 – 2m
Need for accumulator – 2m.

b. Explain the following set of instructions with the help of one example of each. (12)

(i) LDA a16
(ii) MOV r, M
(iii) MVI M, d8
(iv) XCHG
(v) LHLD a16
(vi) STAX r_p

Answer:

LDA is a mnemonic that stands for **LoaD** Accumulator contents from memory. This is an instruction to load Accumulator with the contents of a memory location whose 16-bit address is indicated in the instruction as a16. This instruction uses absolute addressing for specifying the data. It occupies 3 bytes in memory. First byte specifies the opcode, and the successive 2 bytes provide the 16-bit address.

LDA F850H is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.

	Before	*After*
(F850)	BCH	
(A)	12H	BCH

LDA F850H is stored in memory with F850 stored in byte reversal form as shown below.

Code for LDA
50
F8

There are no instructions in 8085 like LDB a16, LDC a16, etc. As was stated earlier, Accumulator is the most important 8-bit register, whose contents can be loaded in more ways than any other 8-bit register.

Summary: LDA a16 (3 bytes; LDA F850H; 1 opcode)

(i) Section 6.8 Page 57 & 58
Explanation — 1m.
Example — 1m.

This is an instruction to load register r with the 8-bit value in memory location. But from which memory location? The address of the memory location is understood to be provided in HL register pair. This instruction uses register-indirect addressing for specifying the data.

As r can have any of the seven values, there are seven opcodes for this type of instruction. It occupies only 1 byte in memory. MOV E, M is an example instruction of this type. It is a 1-byte instruction. Suppose E register content is 45H, H register content is F8H, and L register content is 50H. Let us say location F850H has the data value 8DH. When the 8085 executes this instruction, the contents of E register will change to 8DH, as shown below.

	Before	*After*
(E)	45	8D
(HL)	F850	F850
(F850)	8D	8D

Henceforth, for simplicity, only values that have changed will be shown in the 'After' column. Thus the simplified diagram for the execution of MOV E, M will be as follows.

	Before	*After*
(E)	45	8D
(HL)	F850	
(F850)	8D	

Notice that there are instructions like 'MOV H, M'. This instruction moves contents of a memory location pointed by HL register pair to H register. Generally, it is of no use to anybody! However, such useless instructions are also provided in the instruction set of 8085.

(ii) Section 6.4 Page no 55
Explanation — 1 M.
Example — 1M.

This is an instruction to load a memory location pointed by HL pair with an 8-bit value. This instruction uses immediate addressing for specifying the data. It occupies 2 bytes in memory.

MVI M, 8DH is an example instruction of this type. It is a 2-byte instruction, with opcode for MVI M using up one byte, and 8DH using up one more byte. The result of execution of this instruction is shown below with an example.

	Before	*After*
(HL)	F845	
(F845)	AC	8D

Summary: MVI M, d8 (2 bytes; MVI M, 8DH, 1 opcode)

(iii) Section 6.8 Page no 57
Explanation — 1 M.
Example — 1M.

XCHG is a mnemonic, which stands for e**XCH**anGe. This is an instruction to exchange contents of HL pair with DE pair. This instruction uses implied addressing. It occupies only 1 byte in memory. The result of execution of this instruction is shown below with an example.

	Before	*After*
(HL)	1234H	5678H
(DE)	5678H	1234H

Using XCHG instruction, only HL and DE contents can be exchanged. There are no instructions in 8085 to exchange contents of HL and BC or to exchange contents of DE and BC.

(iv) Section 6.10 Page NO 58-59.
Explanation — 1M.
Example — 1M

LHLD is a mnemonic that stands for Load HL pair using Direct addressing from memory location whose 16-bit address is denoted as a16. As HL pair has to be loaded, the data comes from two consecutive locations starting at the address a16. This instruction uses absolute addressing for specifying the data. It occupies 3 bytes in memory.

LHLD F2BCH is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.

	Before	*After*
(F2BC)	DCH	
(F2BD)	ABH	
(H)	12H	ABH
(L)	34H	DCH

LHLD F2BCH is stored in memory with F2BC stored in byte reversal form as shown below.

Code for LHLD
BC
F2

Note that there are no instructions in 8085 like LBCD a16 and LDED a16. As was stated earlier, HL pair is the most important register pair, whose contents can be loaded in more ways than any other register pair.

(v) Section 6.14 Page No 63.
Explanation — 1M.
Example — 1M.

STAX is a mnemonic that stands for STore Accumulator contents in memory pointed by eXtended register denoted as 'rp'. This instruction uses register indirect addressing for specifying the destination. It occupies only 1 byte in memory.

STAX B is an example instruction of this type. It is a 1-byte instruction. The result of execution of this instruction is shown below with an example.

	Before	*After*
(BC)	F2BCH	
(A)	12H	
(F2BC)	DCH	12H

Only other instruction of this type is STAX D. Note that STAX H is not provided in 8085. This is because, STAX H is the same as MOV M, A in its function.

Also note that there are no instructions in 8085 like STBX rp, STCX rp, etc. As was stated earlier, Accumulator is the most important 8-bit register, whose contents can be stored in memory in more ways than any other 8-bit register.

Summary: STAX rp (1 byte; STAX B; 2 opcodes)

(vi) Section 6.13 Page No 62 & 63.
Explanation — 1M
Example — 1M.

Q.3 a. Differentiate between unconditional and conditional branch instructions, mention various conditional call instructions. (4)

Answer:

10.2 UNCONDITIONAL JUMP INSTRUCTIONS

In simple programs there is a linear program flow from the first instruction to the last instruction in the program. But the real power of a computer exists in its ability to change program flow, and the programs should be written to take advantage of this ability.

Change in program flow is needed when a sequence of instructions is to be executed repeatedly. It is also needed when it is required to choose between two or more sequences of actions based on some conditions. The branch group of instructions in 8085 effects such a change in program flow. The branch instructions can be used for effecting a forward branch or a backward branch.

An important subgroup of the branch group of instructions is the jump instructions. The jump instructions are classified into:

- Unconditional jump instructions, and
- Conditional jump instructions.

10.2.1 JMP a16—Unconditional Direct Jump

JMP is a mnemonic that stands for '**JuMP**' and 'a16' stands for any 16-bit address. This instruction is used to jump to the address a16 provided in the instruction. 'JMP F950H' is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.

Memory address	Instruction
F820H	JMP F950H
F823H	MOV B, C
.	
.	
F950H	MOV C, B; This is executed next after JMP F950H

In the previous example, after 8085 fetches 'JMP F950H' the PC value would have been automatically incremented by 3 to F823H. Functionally, JMP F950H can be treated as:

JMP F950H = LXI PC, F950H

Thus, execution of JMP F950H results in loading of PC with the value F950H, overwriting the earlier value of F823H. Hence, after execution of JMP F950H, the instruction MOV C, B at memory location F950H will be fetched and executed. In this example, a forward jump was effected. JMP F805H instruction at location F820H, effects a backward jump.

In the instruction set of 8085, 'JMP' is used instead of 'LXI PC', because JMP very explicitly indicates a change in program flow. But, LXI PC just indicates that PC value is changed, but its implied change in program flow may go unnoticed by the programmer.

The result of execution of 'JMP F950H' can be indicated in terms of change in PC value as follows.

	Before	*After*
(PC)	F820H	F950H

Summary: JMP a16 (3 bytes; JMP F950H; 1 opcode)

10.2.2 PCHL—Unconditional Indirect Jump

PCHL is a mnemonic, which stands for 'Load **PC** with contents of **HL**'. This instruction is used to jump to the address provided in HL register pair. Thus it is an unconditional indirect jump instruction. Because of its indirect jump feature, it is not very commonly used.

It is a 1-byte instruction compared with the direct jump instruction, which is 3-bytes long. Because of this size advantage, it can be useful for jumping to a frequently used portion of the program.

The result of execution of this instruction is shown below with an example.

Memory address	Instruction
F820H	LXI H, F950H
F823H	PCHL
F824H	MOV B, C
.	
.	
.	
F950H	MOV C, B; This is executed next after PCHL

In the previous example, after 8085 fetches 'PCHL' the PC value would have been automatically incremented by 1 to F824H.

But execution of PCHL results in loading of PC with the value F950H, which is the content of HL, overwriting the earlier value of F824H. Hence, after execution of PCHL, the instruction 'MOV C, B' at memory location F950H will be fetched and executed.

The result of execution of 'PCHL' can be indicated in terms of change in PC value as follows.

	Before	*After*
(HL)	F950H	
(PC)	F823H	F950H

Summary: PCHL (1 byte; PCHL; 1 opcode)

10.3 CONDITIONAL JUMP INSTRUCTIONS

The conditional jump instructions of 8085 perform a jump based on the value of a single flag. The jump takes place based on the value of carry flag, zero flag, parity flag, or sign flag. There is no jump instruction based on the value of auxiliary flag. This is because, generally no one is interested in performing a jump based on this flag!

10.3.1 JNC a16—JUMP IF NOT CARRY

JNC is a mnemonic, which stands for 'Jump if Not Carry', and 'a16' stands for any 16-bit address. This instruction is used to jump to the address a16 provided in the instruction, only if carry flag value is 0. If carry flag value is 1, program flow continues sequentially. 'JNC F950H' is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.

Memory address	Instruction
F820H	JNC F950H
F823H	MOV B, C; This is executed next after JNC F950H if Cy=1
.	
.	
.	
F950H	MOV C, B; This is executed next after JNC F950H if Cy=0

If Cy=0

In the previous example, after 8085 fetches 'JNC F950H' the PC value would have been automatically incremented by 3 to F823H. But JNC F950H instruction execution results in loading of PC with the value F950H, only if Cy flag value is 0.

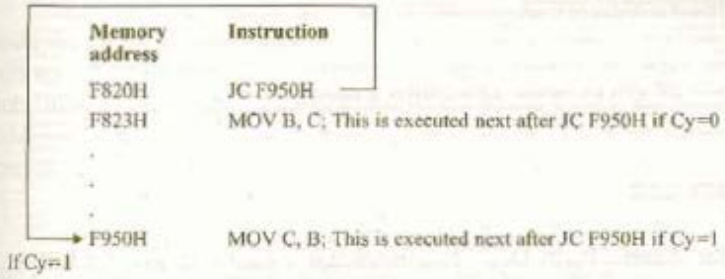
Hence, after execution of JNC F950H, the instruction 'MOV C, B' at memory location F950H will be fetched and executed only if Cy flag value is 0. If Cy flag value is 1, the PC value will remain as F823H, and so the instruction 'MOV B, C' will be executed.

Summary: JNC a16 (3 bytes; JNC F950H; 1 opcode)

10.3.2 JC a16—Jump if Carry

JC is a mnemonic, which stands for '**J**ump if **C**arry'. This instruction is used to jump to the address a16 provided in the instruction, only if carry flag value is 1. If carry flag value is 0, program flow continues sequentially.

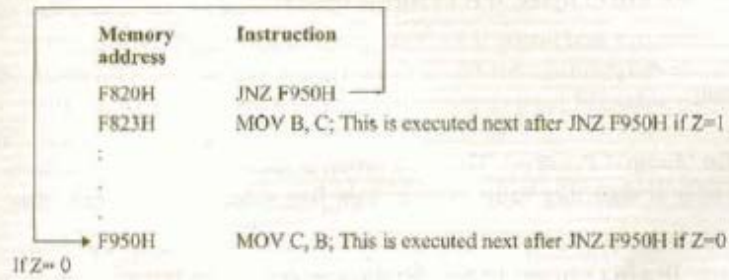
'JC F950H' is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.



Summary: JC a16 (3 bytes; JC F950H; 1 opcode)

10.3.3 JNZ a16—Jump if Not Zero Result

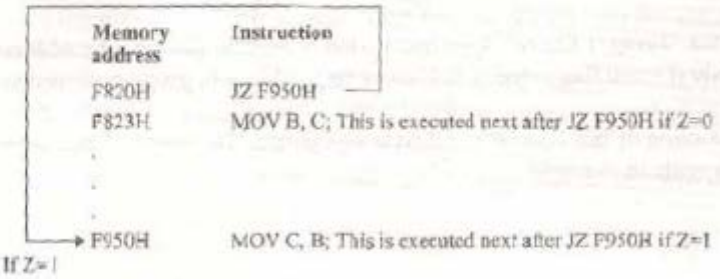
JNZ is a mnemonic, which stands for '**J**ump if **N**ot **Z**ero result'. This instruction is used to jump to the address a16 provided in the instruction, only if result is not zero, indicated by Z flag value of 0. If Z flag value is 1, program flow continues sequentially. 'JNZ F950H' is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.



Summary: JNZ a16 (3 bytes; JNZ F950H; 1 opcode)

10.3.4 JZ a16—Jump if Zero Result

JZ is a mnemonic, which stands for '**J**ump if **Z**ero result'. This instruction is used to jump to the address a16 provided in the instruction, only if result is zero, indicated by Z flag value of 1. If Z flag value is 0, program flow continues sequentially. 'JZ F950H' is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.



Summary: JZ a16 (3 bytes; JZ F950H; 1 opcode)

10.3.5 JPO a16—Jump if Parity Odd

JPO is a mnemonic, which stands for '**J**ump if **P**arity **O**dd'. This instruction is used to jump to the address a16 provided in the instruction, only if parity flag value is 0. If parity flag value is 1, program flow continues sequentially. 'JPO F950H' is an example instruction of this type. It is a 3-byte instruction.

Summary: JPO a16 (3 bytes; JPO F950H; 1 opcode)

10.3.6 JPE a16—Jump if Parity Even

JPE is a mnemonic, which stands for '**J**ump if **P**arity **E**ven'. This instruction is used to jump to the address a16 provided in the instruction, only if parity flag value is 1. If parity flag value is 0, program flow continues sequentially.

Summary: JPE a16 (3 bytes; JPE F950H; 1 opcode)

10.3.7 JP a16—Jump if Positive

JP is a mnemonic, which stands for '**J**ump if **P**ositive'. This instruction is used to jump to the address a16 provided in the instruction, only if sign flag value is 0. If sign flag value is 1, program flow continues sequentially.

Summary: JP a16 (3 bytes; JP F950H; 1 opcode)

JM is a mnemonic, which stands for '**J**ump if **M**inus'. This instruction is used to jump to the address a16 provided in the instruction, only if sign flag value is 1. If sign flag value is 0, program flow continues sequentially.

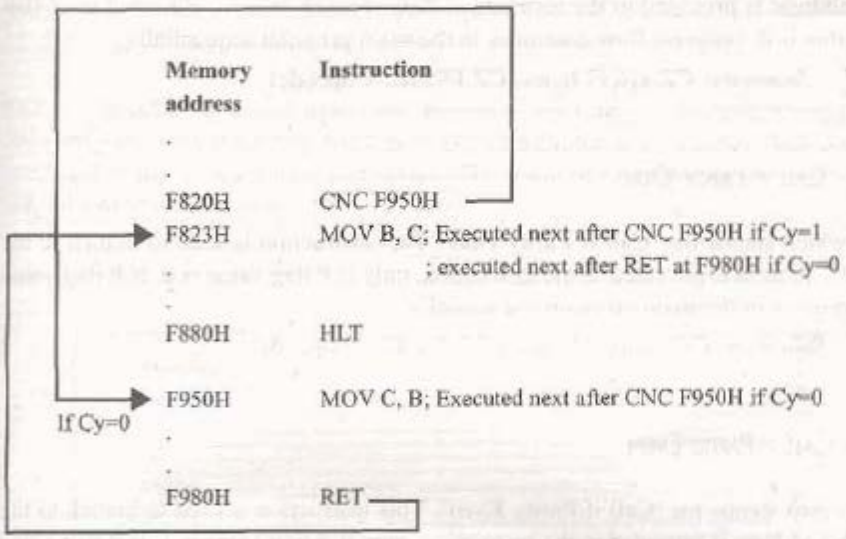
Summary: JM a16 (3 bytes; JM F950H; 1 opcode)

10.5 CONDITIONAL CALL INSTRUCTIONS

The conditional call instructions of 8085 branch to a subroutine based on the value of a single flag. The branch takes place based on the value of Cy flag, Z flag, P flag, or S flag. There is no call instruction based on the value of auxiliary flag. This is because, generally no one is interested in branching to a subroutine based on this flag! The conditional call instructions are 3 bytes in length, 1 byte for the opcode, and another 2 bytes for the subroutine address.

10.5.1 CNC a16—CALL IF NOT CARRY

CNC is a mnemonic, which stands for 'Call if Not Carry'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if Cy flag value is 0. If Cy flag value is 1, program flow continues in the main program sequentially. 'CNC F950H' is an example instruction of this type. It is a 3-byte instruction. The result of execution of this instruction is shown below with an example.



In the previous example, after 8085 fetches 'CNC F950H' the PC value would have been automatically incremented by 3 to F823H. But CNC F950H instruction execution results in saving PC value on the stack and loading of PC with the value F950H, only if Cy flag value is 0.

Hence, after execution of CNC F950H, branch to subroutine at memory location F950H will take place only if Cy flag value is 0. After executing the RET instruction in the subroutine, the program flow will continue with the instruction at F823H in the main program.

If Cy flag value is 1, the PC value will remain as F823H, and so the program flow continues with the instruction MOV B, C in the main program.

Summary: CNC a16 (3 bytes; CNC F950H; 1 opcode)

10.5.2 CC a16—Call if Carry

CC is a mnemonic, which stands for 'Call if Carry'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if Cy flag value is 1. If Cy flag value is 0, program flow continues in the main program sequentially.

Summary: CC a16 (3 bytes; CC F950H; 1 opcode)

10.5.3 CNZ a16—Call if Not Zero Result

CNZ is a mnemonic, which stands for 'Call if Not Zero result'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if result is not zero, indicated by Z flag value of 0. If Z flag value is 1, program flow continues in the main program sequentially.

Summary: CNZ a16 (3 bytes; CNZ F950H; 1 opcode)

10.5.4 CZ a16—Call if Zero Result

CZ is a mnemonic, which stands for 'Call if Zero result'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if result is zero, indicated by Z flag value of 1. If Z flag value is 0, program flow continues in the main program sequentially.

Summary: CZ a16 (3 bytes; CZ F950H; 1 opcode)

10.5.5 CPO a16—Call if Parity Odd

CPO is a mnemonic, which stands for 'Call if Parity Odd'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if P flag value is 0. If P flag value is 1, program flow continues in the main program sequentially.

Summary: CPO a16 (3 bytes; CPO F950H; 1 opcode)

10.5.6 CPE a16—Call if Parity Even

CPE is a mnemonic, which stands for 'Call if Parity Even'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if P flag value is 1. If P flag value is 0, program flow continues in the main program sequentially.

Summary: CPE a16 (3 bytes; CPE F950H; 1 opcode)

10.5.7 CP a16—Call if Positive

CP is a mnemonic, which stands for 'Call if Positive'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if S flag value is 0. If S flag value is 1, program flow continues in the main program sequentially.

Summary: CP a16 (3 bytes; CP F950H; 1 opcode)

10.5.8 CM a16—CALL IF MINUS

CM is a mnemonic, which stands for 'Call if Minus'. This instruction is used to branch to the subroutine whose 16-bit address is provided in the instruction, only if S flag value is 1. If S flag value is 0, program flow continues in the main program sequentially.

Summary: CM a16 (3 bytes; CM F950H; 1 opcode)

3 (a). Sections 10.2 & 10.3. Page no 102 to 107.
Differences — 2M.
Section 10.5 Page no 109 to 111
Mentioning of various conditional call instructions — 2M.

b. With a neat block diagram, explain the architecture of 8085. (12)

Answer:

13.1 DETAILS OF 8085 ARCHITECTURE

Shown in Fig. 13.1 is the architecture of 8085. As we are already aware, it has 8-bit ALU, control unit, the general purpose registers A, B, C, D, E, H, and L, and the special purpose SP, PC, and Flags registers.

13.1.1 ARITHMETIC LOGIC UNIT (ALU)

It basically performs 8-bit arithmetic and logical operations. Accumulator and Temp registers provide the two operands needed in the operations like addition or logical AND operations. The results will be stored in the accumulator. This is done by sending the ALU output to the accumulator via the internal bus. Also, the Flags register is affected based on the result. In an instruction like ADC B, Temp register receives B register value, and Cy flag is also input to the ALU.

In 'DAD B' instruction, it is required to add HL and BC contents, and store the result in HL. This is also performed by the 8-bit ALU, by adding the LS bytes first, and then adding the MS bytes along with any carry generated.

Even increment or decrement of 16-bit registers is done by this ALU. But in Fig. 13.1 incrementer/decrementer unit is separately shown just for convenience.

13.1.2 TIMING AND CONTROL UNIT

This unit is responsible for generating timing signals and control signals. This unit controls all the activities inside the 8085, as well as outside the 8085.

X1, X2, and Clk out pins: To facilitate timing operations in the microcomputer system, there is a clock generator in the control unit of 8085. The complete oscillator circuit, except the quartz crystal is within the chip. Two pins X1 and X2 are brought out of the chip to provide for the external crystal connection. This is shown in Fig. 13.1. Generally a capacitor of about 20pF value has to be connected between X2 and ground to ensure proper starting up of the crystal. There is a divide by 2 counter in the control unit, which divides the crystal frequency by 2. The 8085A can work at an approximate

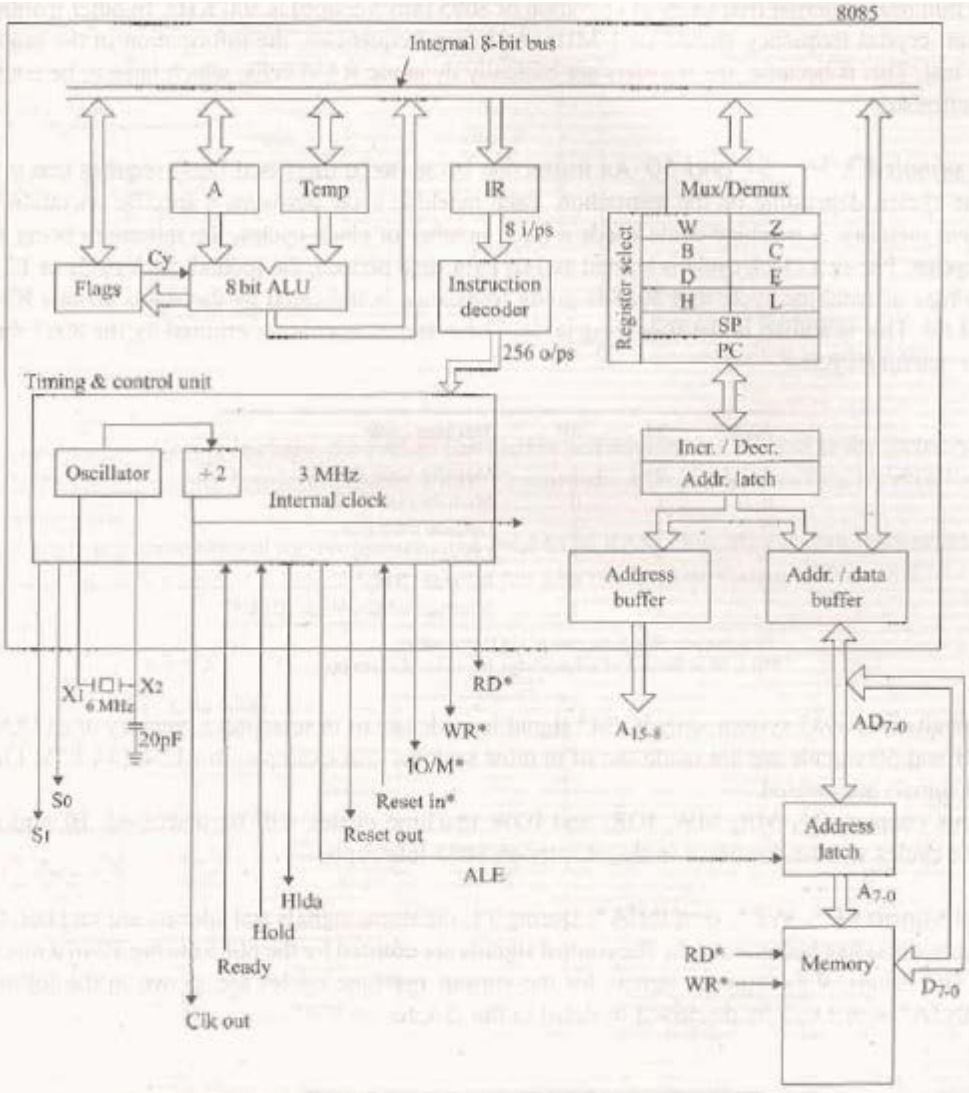


Fig. 13.1 8085 architecture (instruction execution portion only)

maximum clock frequency of 3 MHz internally. Thus typically a 6-MHz crystal is connected between X1 and X2. All operations in a 8085 system take place in synchronization with this clock. This clock signal is brought out on 'clock out' pin of 8085. Peripheral chips, like 8251 USART, which need a clock signal for their operation, use it.

It is also possible to connect a LC-tuned circuit between X1 and X2, instead of a crystal. In very low-cost systems, a resistor can be connected between X1 and X2, and a capacitor connected between X1 and ground to generate the oscillations. However, for good stability of oscillations, it is preferred to use a quartz crystal.

With internal frequency as 3 MHz, the clock period will be 333 nS. In other words, the 8085 clock ticks once every 333 nS, whereas our watch is ticking once every second! A clock cycle of 8085 is also called as a *T* state, *T* standing for 'timing'.

The minimum internal frequency of operation of 8085 (any version) is 500 KHz. In other words, the minimum crystal frequency should be 1 MHz. At lesser frequencies, the information in the registers will be lost. This is because, the registers are basically dynamic RAM cells, which have to be continuously refreshed.

Status signals IO/M, S1 and S0:* An instruction cycle (to be discussed later) requires one to five machine cycles, depending on the instruction. Each machine cycle performs a specific operation, like read from memory. A machine cycle needs a fixed number of clock cycles, the minimum being three clock cycles. The first clock cycle is termed as T1 (T for time period), the second clock cycle as T2, etc.

The type of machine cycle that 8085 is going to execute is indicated by the status signals IO/M*, S1, and S0. This is shown in the following table. These status signals are emitted by the 8085 during T1 of a machine cycle.

*IO/M**	*S1*	*S0*	*Machine cycle*
0	0	1	Memory write (MW)
0	1	0	Memory read (MR)*
0	1	1	Opcode fetch (OF)
1	0	1	I/O write (IOW)
1	1	0	I/O read (IOR)
1	1	1	Interrupt acknowledge (INA)**

*It is bus idle (BI) in the case of DAD instruction.
**It is BI in the case of acknowledge for vectored interrupts.

Generally in a 8085 system, only IO/M* signal is made use of in selecting a memory or an I/O port chip. S1 and S0 signals are not made use of in most systems. For example, in ALS-SDA-85M kit, S1 and S0 signals are unused.

In this chapter, OF, MR, MW, IOR, and IOW machine cycles will be discussed. BI and INA machine cycles will be discussed in the chapter on 8085 interrupts.

Control signals RD, WR*, and INTA*:* During T1, the status signals and address are sent out. Only after the address has become stable, the control signals are emitted by the 8085 during T2 of a machine cycle. The values of the control signals for the various machine cycles are shown in the following table. INTA* signal will be discussed in detail in the chapter on 8085 interrupts.

Machine cycle	*RD**	*WR**	*INTA**
OF, MR, IOR	0	1	1
MW, IOW	1	0	1
INA	1	1	0
BI	1	1	1

During T2 of the halt machine cycle, 8085 tristates RD* and WR*, and emits 1 on INTA.

Address latch enable signal (ALE): As discussed in the previous chapter, 8085 has multiplexed address data pins. So LS byte of address will have to be sent out for only a short time in a machine cycle. During the rest of the machine cycle, the same pins are to be used for data transmission or reception.

Some specialized chips like Intel 8155 have an address latch within the chip, as shown in Fig. 13.2.

During T1 of a machine cycle, 8085 sends out address on AD_{7-0} and sends out logic 1 on ALE. The address latch in 8155 latches on to the address using this signal. During T2, 8085 stops sending the

13.1.3 Instruction Register (IR)

This is a 8-bit register. It is used to receive the 8-bit opcode of an instruction from memory. It may be noted that opcode of an instruction is only 8 bits, even though the instruction may be 3-bytes long. IR register is not accessible to the programmer. This means there are no instructions by which a programmer can load a value of his choice into this register. There are no instructions like 'MVI IR, 35H', or 'MOV IR, C'.

Instruction decoder: The instruction decoder receives the input from the IR register. It is an 8-input to 256-output decoder. Depending on the input, only one of the 256 output lines will be activated. The instruction decoder output drives the control unit. The control unit then generates the appropriate control signals to execute the instruction.

13.1.4 W and Z Registers

These are 8-bit registers. They are not accessible to the programmer. They are used for temporarily storing inside the 8085, the 16-bit address operand of an instruction. For example, when 'LDA C234H, instruction is fetched, IR register will receive the opcode for LDA, and W and Z registers will receive C2H and 34H, respectively.

13.1.5 Temporary (Temp) Register

This is an 8-bit register. It is not accessible to the programmer. It temporarily stores inside 8085, the 8-bit operand in an instruction. For example, when 'MVI M, 34H' instruction is fetched, IR register will receive the opcode for MVI M, and Temp register will receive 34H.

In arithmetic and logical operations that involve two operands, the accumulator provides one operand, and the other is provided by the Temp register. For example, in ADD B instruction, B register contents are moved to the Temp register, and then addition of A and Temp register contents is performed by the ALU.

13.1.6 Multiplexer/Demultiplexer

Consider the execution of the instruction 'MOV A, C'. In this case, the 8-bit value in the C register has to be moved to the A register. The registers B, C, D, E, H, and L are connected to the internal bus through a multiplexer/demultiplexer. The register select unit sends the appropriate code to the multiplexer so that C register contents are sent out of the multiplexer to the internal bus. Then Accumulator receives the data from the internal bus.

Similarly, consider the execution of the instruction 'MOV D, A'. In this case, the 8-bit value in the Accumulator has to be moved to the D register. Accumulator sends out the 8-bit value to the internal bus. The registers B, C, D, E, H, and L are connected to the internal bus through a multiplexer/demultiplexer. The register select unit sends the appropriate code to the demultiplexer so that D register receives the contents of the internal bus from the demultiplexer.

13.1.7 Address/Data Buffers

These buffers are bi-directional, when used for data. When used for sending out LS byte of address, they are unidirectional. The buffers are used to increase the current driving capacity. Data comes to

the buffers from the internal bus. LS byte of address comes to the buffers from the internal address latch.

Thus the address/data sent out on AD_{7-0} can drive all the external chips, like RAM chips, EPROM chips, and other peripheral chips. Similarly, the data received by the 8085 from outside is also internally buffered. The received data on AD_{7-0} reaches the internal bus, from where it reaches the final destination.

In fact, in a practical microcomputer, the driving capacity of the data pins, after the internal buffering, may not be adequate. So there will be external buffer chips also. Practical use of such buffer chips is shown in Fig. 13.19, later in the chapter.

Address buffers: These buffers are unidirectional. They are used for sending out the MS byte of address. MS byte of address comes to the buffers from the internal address latch. Thus the address sent out on AD_{15-8} can drive all the external chips, like RAM chips, EPROM chips, and other peripheral chips.

In fact, in a practical microcomputer, the driving capacity of the address pins, after the internal buffering, may not be adequate. So there will be external buffer chips also. Details of such buffer chips are given in the appendix.

13.1.8 Internal Address Latch

The register select unit in the 8085 selects one of the register pairs (BC, DE, HL, SP, PC, or WZ) for sending it to the address latch unit. For example, let us say the contents of PC is C200H. If the register selection unit selects PC, it sends C200H from PC to the internal address latch. The latch holds on to this value, and sends it out on the address pins after buffering. The MS byte of address, C2H, is sent out on A_{15-8}, and the LS byte of address, 00H, is sent out on the pins AD_{7-0}.

A little later, the PC value may be incremented, but still the latch will continue to have the value C200H. This value of C200H is still sent out on the address pins of 8085 even if the PC content is already incremented.

13.1.9 Incrementer/Decrementer

It is actually a function perfomed by the ALU. But in Fig. 13.1, it is shown as a separate unit just for convenience.

The incrementer is used to increment the PC value, after the 8085 has fetched a byte of the instruction. This way, the PC will be pointing to the next instruction by the time the current instruction is fully fetched. It is also used for incrementing the SP value after a byte is popped out from the stack top. It is also used for incrementing an 8-bit or a 16-bit register.

The decrementer is used for decrementing the SP value before a byte is pushed above the stack top. It is also used for decrementing an 8-bit or a 16-bit register.

13.1.10 Connection of Registers to The Internal bus

Accumulator will have to receive data from internal bus in instructions like 'MOV A, C'. It has to send out data to internal bus in instructions like 'MOV D, A'. As such, the Accumulator communicates with the internal bus in a bi-directional way.

Flags register will have to receive data from internal bus in instructions like POP PSW. It has to send out data to internal bus in instructions like PUSH PSW. As such, the Flags register communicates with the internal bus in a bi-directional way.

In instructions like 'MVI M, 25H', Temp register will have to first of all temporarily receive data value of 25H from internal bus. Later this data in Temp register has to be sent out to memory via the internal bus. As such, the Temp register communicates with the internal bus in a bi-directional way.

IR register will always have to receive the opcode from memory via the internal bus for any instruction. It is never required to send out opcode to the internal bus. As such, the IR register only receives opcode via the internal bus.

Register B will have to receive data from internal bus via demultiplexer in instructions like 'MOV A, B'. It has to send out data to internal bus via multiplexer in instructions like 'MOV B, A'. As such, register B communicates with the internal bus in a bi-directional way. Similarly, registers C, D, E, H, and L also communicate with the internal bus in a bi-directional way.

In instructions like 'LDA 2345H', W and Z registers will have to first of all temporarily receive address value of 2345H from the internal bus. Later they have to send out this address in W and Z registers to memory via the internal bus. As such, W and Z registers communicate with the internal bus in a bi-directional way.

MODERATION

Section 13.1 Page no 134 to 140 TEXT BOOK-I

Figure NO 13.1 — 4M.

Explanation — 8M.

158

Q.4 a. Write an 8085 assembly language program and flow chart to search for a given byte in an array of bytes using linear search algorithm. Location X contains the size of the array and location X + 1 contains the element to be searched. The elements of the array are stored from location Y onwards. The program should display in the address field, the search element and the position when it was found. If the search element is not found, the position should be indicated as 00. (8)

Answer:

Write an 8085 assembly language program to search for a given byte in an array of bytes using linear search algorithm. Location X contains the size of the array and location X+1 contains the element to be searched. The elements of the array are stored from location Y onwards. The program should display in the address field, the search element and the position where it was found. If the search element is not found, the position should be indicated as 00.

Flowchart for solving the problem is shown in Fig. 16.1.

16.1.1 Program to Perform Linear Search

```
;FILE NAME C:\ALS\LINSRCH.ASM

  ;8085 ALP TO PERFORM LINEAR SEARCH. LOCATION X CONTAINS
  ;THE NUMBER OF BYTES TO SEARCH, LOCATION X+1 CONTAINS THE ELEMENT
  ;TO BE SEARCHED, AND LOCATION Y ONWARDS ARE THE ELEMENTS OF THE ARRAY.

  ;PROGRAM DISPLAYS IN THE ADDRESS FIELD, THE SEARCH ELEMENT AND THE
  ;POSITION WHERE IT WAS FOUND. IF THE SEARCH ELEMENT IS NOT FOUND,
  ;THE POSITION WILL BE INDICATED AS 00.

  ORG C100H
X: DB 04H,33H
```

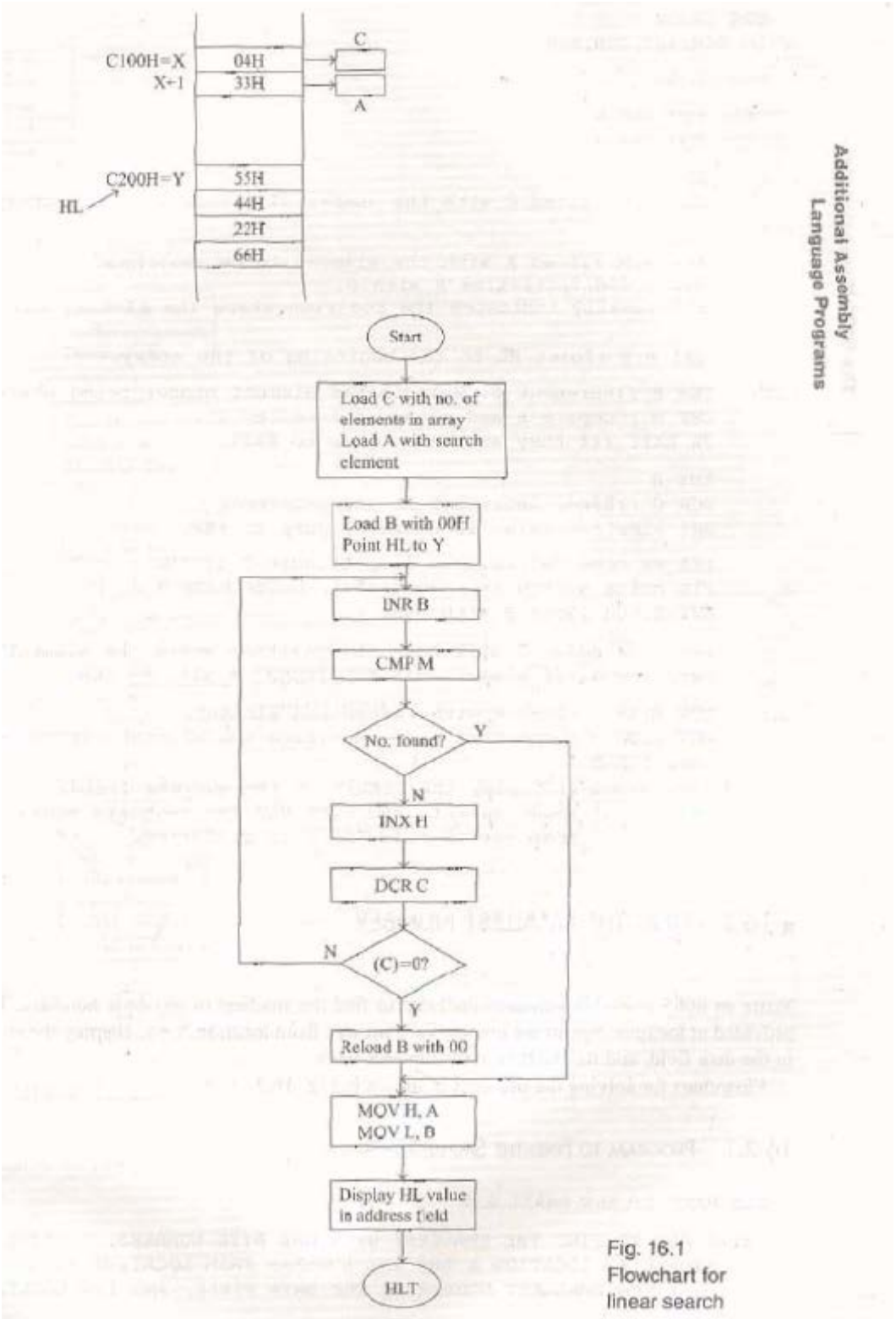


Fig. 16.1 Flowchart for linear search

The 8085 Microprocessor

```
        ORG C200H
Y: DB 55H,44H,22H,66H
        ORG C000H
CURAD:  EQU FFF7H
UPDAD:  EQU 06BCH
        LXI H,X
        MOV C,M ;;Load C with the number of elements in the array.
        INX H
        MOV A,M ;;Load A with the element to be searched.
        MVI B,00H;Initialise B with 0.
        ;It finally indicates the position where the element was found.
        LXI H,Y ;Point HL to the beginning of the array.
REP:    INR B ;Increment B. B indicates element number being checked.
        CMP M ;Compare A and memory pointed by HL.
        JZ EXIT ;If they are same, jump to EXIT.
        INX H
        DCR C ;;Else, increment HL and decrement C.
        JNZ REP;If C value is nonzero, jump to REP.
        ;If we come out of this loop because C is 00,
        ;it means search is unsuccesful. Hence make B as 00.
        MVI B,00H ;Load B with 00H.
        ;At this point B will have the position where the element
        ;was found. If element was not found, B will be 00H.
EXIT:   MOV H,A    ;Load H with the search element.
        MOV L,B    ;Load L with the position the element was found.
        SHLD CURAD
        CALL UPDAD ;;Display the result in the address field.
        HLT        ;Make sure to end with HLT for keyboard mode.
                   ;For serial mode RST 1 is preferred.
```

4 (a). Section 16.1. Page no 206 to 208. TEXT BOOK-I
Flow chart, Figure NO 16.1. — 4M.
Program, Section 16.1.1 — 4M.

b. Write an 8085 assembly language program and flowchart to exchange 10 byte of data stored from location X with 10 byte of data stored from location Y. (8)

Answer:

Write an 8085 assembly language program to exchange 10 bytes of data stored from location X with 10 bytes of data stored from location Y.

Flowchart for the program

Flowchart for solving the problem is shown in Fig. 14.1.

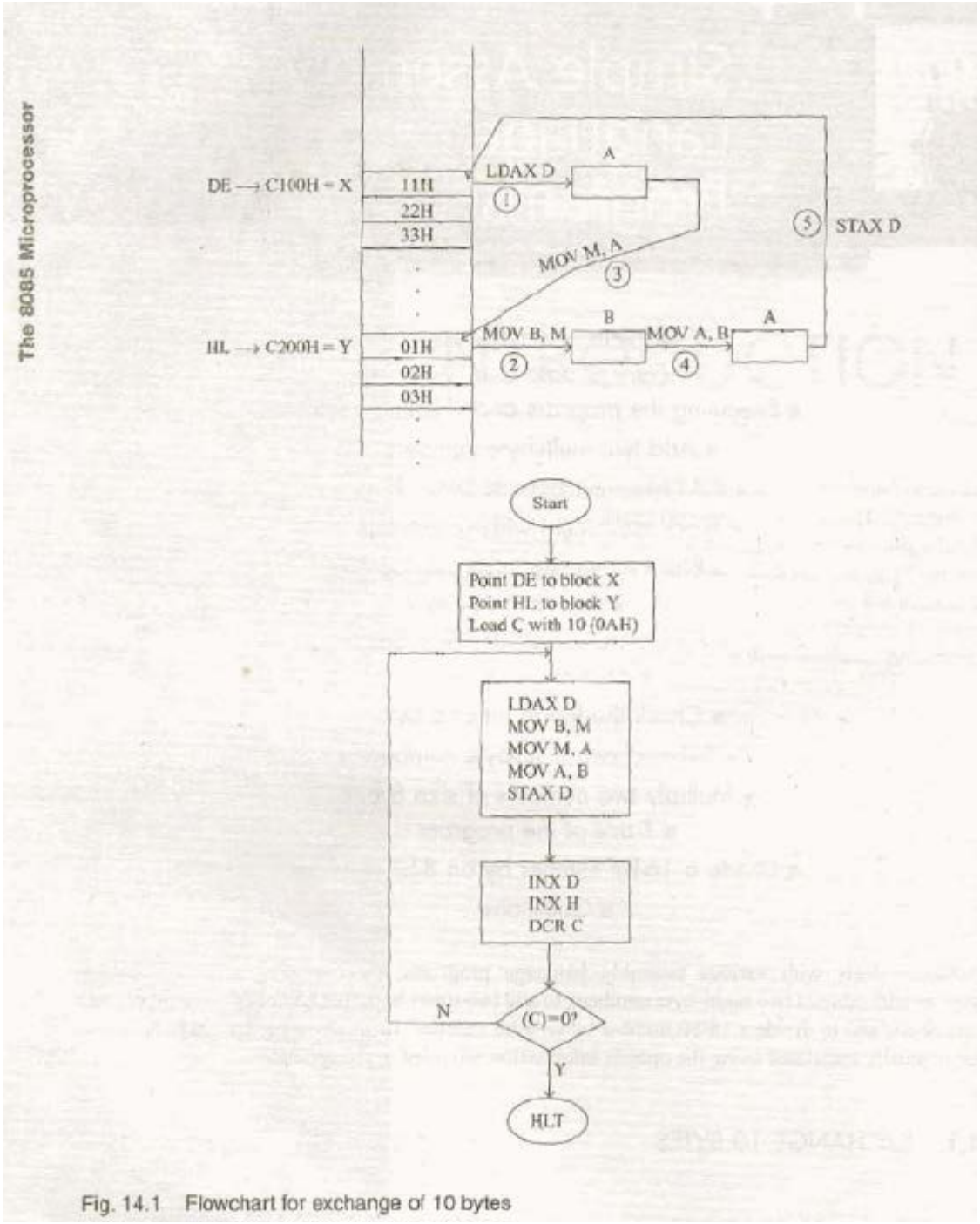


Fig. 14.1 Flowchart for exchange of 10 bytes

Program to exchange 10 bytes

```
; FILE NAME C:\AL5\XCHG.ASM
  ; Suppose we want to make some changes to the source file. If a print out
;of the program informs us about the file name for the program on the disk,
;it becomes easy to search the file and make modifications to the program.

   ;8085 ALP TO INTERCHANGE 10 BYTES OF DATA STORED FROM
   ;LOCATION X WITH 10 BYTES OF DATA STORED FROM LOCATION Y

   ORG C100H
  X: DB 11H, 22H, 33H, 44H, 55H, 66H, 77H, 88H, 99H, AAH

   ORG C200H
  Y: DB 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H, 09H, 0AH

;For manual translation store 11H,22H etc starting at location C100H.
;Similarly, the following program after manual translation should be stored
;from location C000H. ORG, DB are Assembler directives and are explained in
;detail in the next chapter.

   ORG C000H

   LXI D, X ; Load DE with C100H (address X)
   LXI H, Y ; Load HL with C200H (address Y)
   MVI C, 0AH; Load C with 0AH (C used as down counter)

   ; The instructions from here to JNZ LOOP perform exchange
   ; of bytes at memory locations pointed by DE and HL

LOOP:    LDAX D     ; Load A from memory pointed by DE
         MOV B, M   ; Load B from memory pointed by HL

         MOV M, A   ; Store A in memory pointed by HL
         MOV A, B
         STAX D     ;;Store B in memory pointed by DE

;In the convention followed in this book, a single ';' after an instruction
;provides comments for the instruction. If there are N number of ';' after an
;instruction, it provides comments for N instructions preceding the N number
;of semicolons.

      INX D
      INX H      ;;Increment address pointers DE and HL
      DCR C      ; Decrement C
      JNZ LOOP   ; If not zero jump to LOOP

      HLT        ; Stop when all the bytes are exchanged
```

Simple Assembly Language Programs

(b) Section 14.1 Page no 165 to 167 TEXT BOOK-I
Flow chart, Figure No 14.1 — 4M.
Program, Section 14.1 — 4M.

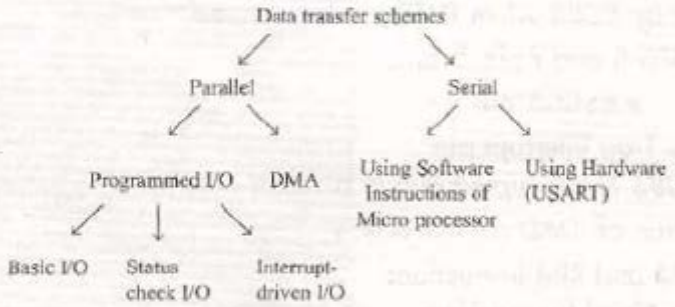
Q.5 a. Mention different data transfer schemes to communicate with a I/O device and explain any one scheme in detail. (12)

Answer:

When the 8085 is executing a program, it can get interrupted half way through the program by an I/O device. An I/O device interrupts the working of the processor, because it may want to urgently communicate with the processor. It may want to send some information to the processor, or receive some information from the processor.

A microprocessor does not directly communicate with an I/O device. It communicates with an I/O device via an I/O port. Data transfer can be in parallel or serial form. Parallel data transfer is possible using programmed I/O or Direct Memory Access (DMA). Serial data transfer and DMA data transfer are explained in later chapters. There are three different ways a microprocessor can communicate with an I/O port for parallel data transfer with programmed I/O. They are:

1. Basic or simple data transfer;
2. Status check data transfer;
3. Interrupt-driven data transfer.



18.1.1 Basic or Simple Data Transfer

This is the simplest of the data transfer schemes. This method is useful when we have accurate knowledge of the I/O device timing characteristics. When we know that the device is ready for data transfer, we execute IN or OUT instruction, depending on the required direction of data transfer. This is the case when the I/O port is connected in the system as an I/O-mapped I/O port. If the port is connected as a memory-mapped I/O port, 'MOV M, A', 'MOV A, M', or any other memory reference instruction is used depending on the direction of data transfer.

As an example, let us say we have a hypothetical multiplier chip. Registers R0 and R1 are used to load the two 8-bit numbers to be multiplied. The 16-bit product will be available in R2H and R2L when the multiplication is over. Address pins A1 and A0 select a register as shown in the following

A1	*A0*	*Selected register*
0	0	R0
0	1	R1
1	0	R2L (LS byte of product)
1	1	R2H (MS byte of product)

Let us say the chip is connected as I/O-mapped I/O, as shown in Fig. 18.1. As per the chip select circuit, the address for R0, R1, R2L, and R2H is 40H, 41H, 42H, and 43H, respectively. Let us say the data sheet for the multiplier specifies a maximum of 50 μs for completing the multiplication.

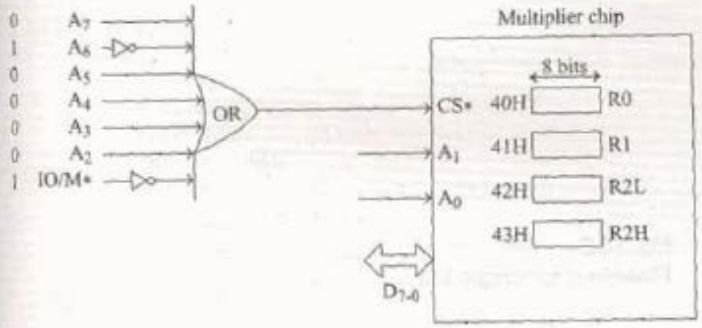


Fig. 18.1
Simple I/O data transfer

With this set up, the following program segment performs multiplication of 05H and 08H, and the result will be stored in the BC register pair.

```
MVI A, 05H
OUT 40H      ;Load R0 with 05H
MVI A, 08H
OUT 41H      ;Load R1 with 08H
CALL DELAY   ;Generate a delay of 50 micro seconds
;The subroutine is not shown for simplicity
IN 42H
MOV C, A     ;Store LS byte of product in C register
IN 43H
MOV B, A     ;Store MS byte of product in B register
```

In the previous example, even if the multiplication is over in 20 μs, we are required to wait for 50 μs, as per the data sheet. The advantage of simple I/O is its simplicity, but its disadvantage is that it is not very efficient. We come across a number of examples later in the text for this type of data transfer. The flowchart for simple I/O is illustrated in Fig. 18.2.

18.1.2 Status Check Data Transfer

This is more complex than simple data transfer. This method is used when we do not have accurate knowledge of the I/O device timing characteristics. The processor should get status information about the readiness of the I/O device for data transfer. Generally, the processor will be in a loop checking for the readiness of the device. The moment the device is ready, it comes out of the loop, and executes IN or OUT instruction depending on the requirement. In this case, the processor has simply wasted its time in a loop till the device got ready.

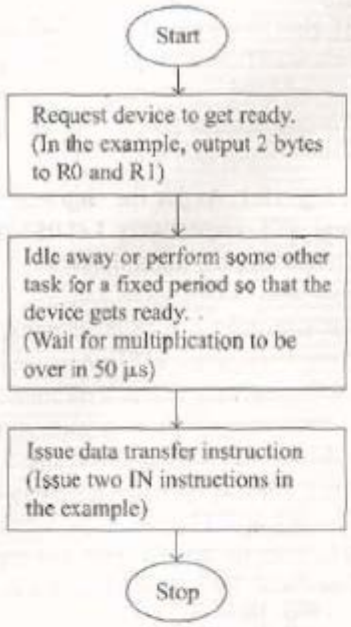


Fig. 18.2
Flowchart for simple I/O

As an example, let us once again consider the hypothetical multiplier chip. Let the chip be provided with command and status registers also. Let us say when the MS bit of the command register is set to 1, it is a command to start the multiplication of the two numbers in R0 and R1. Let us say the LS bit of the status register is set to 1 when the product is available in R2H and R2L.

Address pins A2, A1, and A0 select a register as shown in the following.

A2	*A1*	*A0*	*Selected register*
0	0	0	R0
0	0	1	R1
0	1	0	R2L (LS byte of product)
0	1	1	R2H (MS byte of product)
1	0	0	Command register
1	0	1	Status register

Let us say the chip is connected as I/O-mapped I/O, as shown in Fig. 18.3. As per the chip select circuit, the address for R0, R1, R2L, R2H, command register and status register is 40H, 41H, 42H, 43H, 44H, and 45H, respectively.

In this case, the microprocessor sends the two bytes to be multiplied to the registers R0 and R1. Then a command is issued to start the multiplication process. This is done by setting the MS bit of the command register. Then the status register is continuously monitored for the completion of the multiplication operation. This is done by checking the LS bit of status register. When the LS bit of status register becomes 1, the microprocessor reads the result.

With this set up, the following program segment performs multiplication of 05H and 08H, and the result will be stored in the BC register pair.

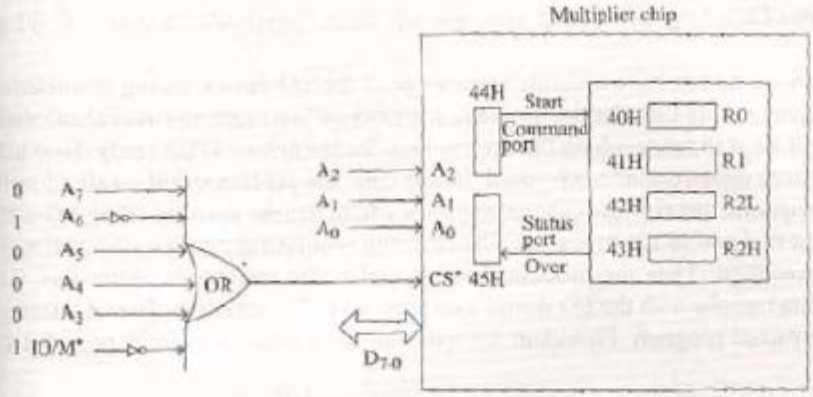


Fig. 18.3
Status check data transfer

```
        MVI A, 05H
        OUT 40H      ;Load R0 with 05H
        MVI A, 08H
        OUT 41H      ;Load R1 with 08H
        MVI A, 10000000B
        OUT 44H      ;Issue Start multiplication command
WAIT:   IN 45H
        RRC
        JNC WAIT     ;Be in wait loop till LS bit of Status
                     ;register becomes 1
        IN 42H
        MOV C, A     ;Store LS byte of product in C register
        IN 43H
        MOV B, A     ;Store MS byte of product in B register
```

In status check I/O, as soon as the device is ready, the data transfer is performed. Status check I/O is more efficient than simple I/O, but is more complex. This method of data transfer is also known as *handshake data transfer*.

We come across a number of examples later in the text for this type of data transfer. The flowchart for status check I/O is illustrated in Fig. 18.4.

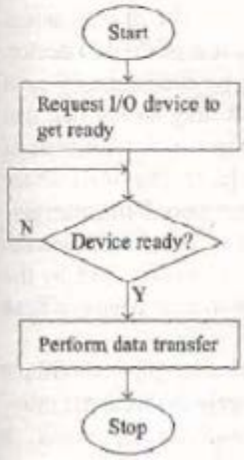


Fig. 18.4
Flowchart for status check I/O

18.1.3 Interrupt-Driven Data Transfer

This method is used when we do not have accurate knowledge of the I/O device timing characteristics, except that it takes quite a long time for the device to get ready. If we resort to status check data transfer, the processor will have to waste a long time in the loop for the device to get ready. To avoid this problem, interrupt-driven data transfer can be used. In this case, the processor will go ahead with its required work, and whenever the device gets ready for data transfer, the corresponding I/O port will send a interrupt request signal to the processor. The interrupt request may arrive even half way through an instruction execution. Then the processor will complete the instruction. After this, the processor will perform data transfer with the I/O device using IN or OUT instruction. Then it resumes the execution of the interrupted program. Flowchart for interrupt-driven data transfer is provided in Fig. 18.5a.

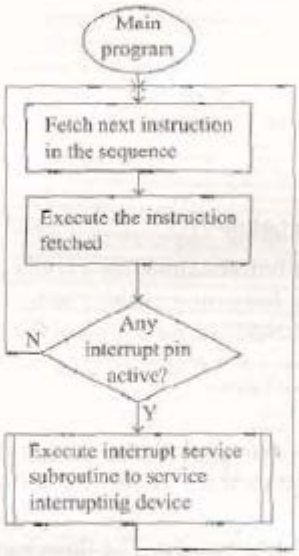


Fig. 18.5a
Flowchart for interrupt-driven data transfer

This method is obviously the most complex of the three types of data transfer. But the advantage now is that the processor is not wasting its time in a loop checking for the readiness of the I/O device.

For example, if the 8085 wants to read from a keyboard, one way is to continuously scan the keyboard looking for a pressed key. This is the status check method. It may so happen that once in a second or so, it finds that a key is pressed. The 8085 is in a loop for this amount of time just waiting for a key depression. In this time, the 8085 could have executed about 500,000 instructions, assuming an average execution time of 2 μs per instruction! In interrupt-driven data transfer, the 8085 would execute about 500,000 useful instructions, by which time the user presses a key on the keyboard. This causes an interrupt signal to be sent to the 8085 by the I/O port. Then the 8085 reads from the port attached to the keyboard, after which it goes ahead with the program.

In the remaining portion of this chapter, we mainly discuss the interrupt pins of 8085, interrupt-related instructions, and some programs that use interrupts. These programs clearly describe the interrupt-driven data transfer.

5 (a). Section 18.1 page no 278 to 282. Text Book - I
Naming of data transfer Schemes — 4 M.
Explanation of any one scheme:
- Flow chart — 4 M.
- Explanation — 4 M.

b. Write the action taken by 8085 when INTR pin is activated. (4)

Answer:

It is assumed that interrupt system is enabled using EI instruction, and higher priority internal interrupt signals are not active.

1. In the penultimate clock cycle of the last machine cycle of every instruction, the 8085 senses all the internal interrupt signals.
2. If INTR internal signal is at logic 1, the 8085 enters an interrupt acknowledge (INA) machine cycle.
3. The interrupt from the I/O port is acknowledged by the 8085 by activating INTA* pin in the T2 state of the INA machine cycle. INTA* is an active low pin. In response to INTA*, the interrupting port should send code for CALL instruction to IR register of 8085 on AD_{7-0} pins. Intel

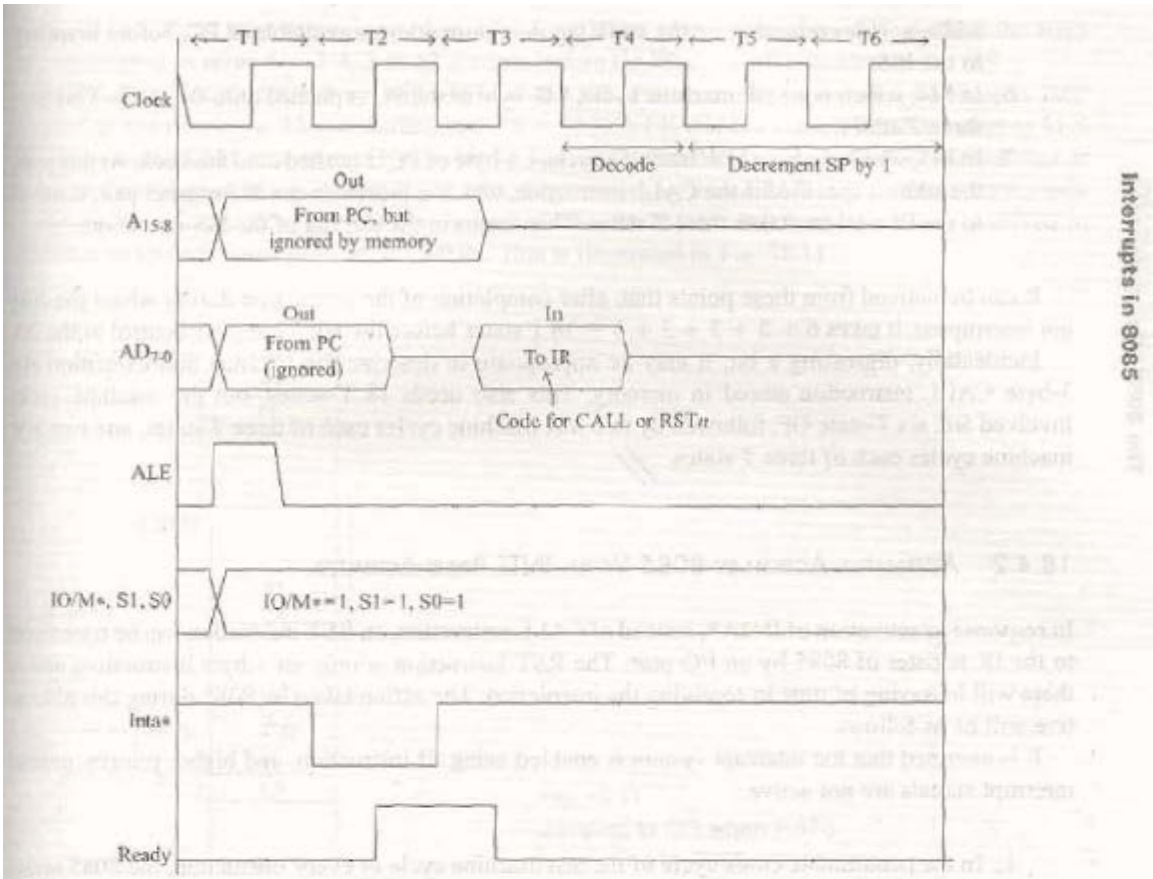
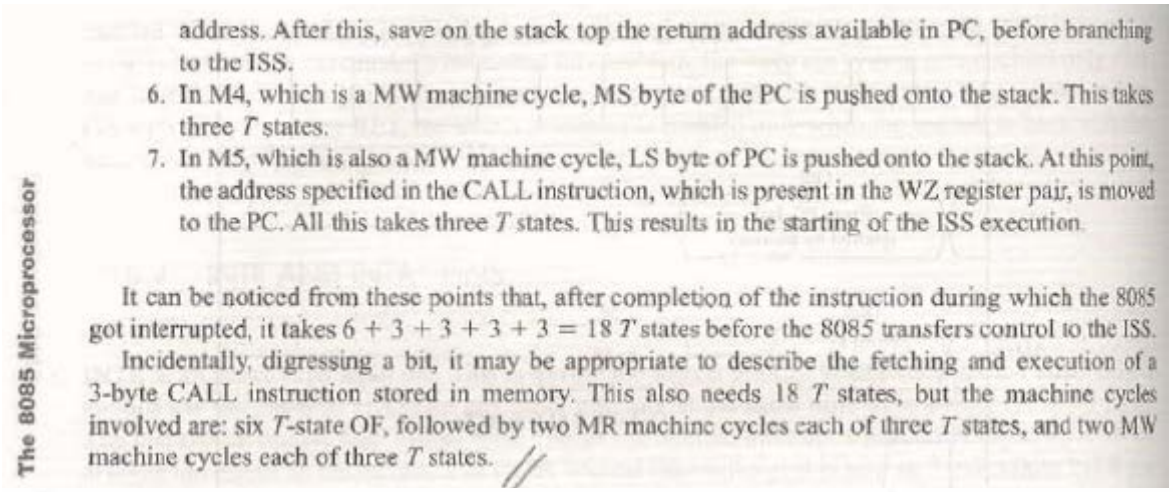


Fig. 18.10 Interrupt acknowledge (INA) machine cycle

8259 interrupt controller is capable of sending such a CALL instruction to 8085. Intel 8259 is discussed in a later chapter. Note that the CALL instruction does not come from memory in this case. So this M1 machine cycle is not an OF machine cycle from memory. Also, it is not an IOR machine cycle, because INTA* is activated, and not RD*. In this INA machine cycle, IO/M*, S1, and S0 will all be in 1 state. This INA machine cycle is shown in Fig. 18.10. INTA* signal becomes a 1 after the 8085 receives the code for CALL during this INA machine cycle. This INA machine cycle that fetches the code for CALL and decodes it, uses six *T* states.

4. In M2 machine cycle, which is also an INA machine cycle, LS byte of ISS address is transferred to the Z register of 8085 by the 8259. This will be in response to activation of INTA* during T2 of this INA machine cycle. This machine cycle uses only three *T* states, as the 8085 does not have to decode this information.
5. In M3 machine cycle, which is also an INA machine cycle, MS byte of ISS address is transferred to the W register of 8085 by the 8259. This will be in response to activation of INTA* during T2 of this INA machine cycle. This machine cycle also uses only three *T* states, as the 8085 does not have to decode this information. Now the 8085 has received the complete ISS



address. After this, save on the stack top the return address available in PC, before branching to the ISS.

6. In M4, which is a MW machine cycle, MS byte of the PC is pushed onto the stack. This takes three *T* states.
7. In M5, which is also a MW machine cycle, LS byte of PC is pushed onto the stack. At this point, the address specified in the CALL instruction, which is present in the WZ register pair, is moved to the PC. All this takes three *T* states. This results in the starting of the ISS execution.

It can be noticed from these points that, after completion of the instruction during which the 8085 got interrupted, it takes 6 + 3 + 3 + 3 + 3 = 18 *T* states before the 8085 transfers control to the ISS.

Incidentally, digressing a bit, it may be appropriate to describe the fetching and execution of a 3-byte CALL instruction stored in memory. This also needs 18 *T* states, but the machine cycles involved are: six *T*-state OF, followed by two MR machine cycles each of three *T* states, and two MW machine cycles each of three *T* states. //

The 8085 Microprocessor

(b). Section 18.4.1 Page no 288 to 290 TEXT BOOK-I

any 4 conditions should be written

Each condition carries - 1 M.

Q.6 a. Explain Port C bit set/reset control word of 8255. Write the required Port C bit set/reset control word for each of the following cases- (12)

(i) To reset to 0 bit 5 of Port C

(ii) To set to 1 bit 3 of Port C

Answer:

Suppose Port C lower has been configured as output port. If we want to send the data 0101 out on PC_{3-0} we can execute the following instructions, assuming 8255 is connected as shown in Fig. 20.4.

```
MVI A, xxxx0101B ; x = don't care (it could be 0 or 1)
OUT 22H
```

After a while if we want to send out logic 0 on PC_0 without affecting other lines of Port C, we have to execute the following instructions.

```
IN 22H
ANI 11111110B
OUT 22H
```

Intel 8255 provides an alternative way of sending out logic 1 or 0 on any pin of Port C that is configured as an output pin. It is done using the single bit set/reset feature of Port C. This feature reduces software requirement in control-based applications. This facility is provided only for Port C. Figure 20.7 explains the Port C bit set/reset control word.

7	6	5	4	3	2	1	0	← Bit number
0	x	x	x	SB2	SB1	SB0	S/R*	

Fig. 20.7 Port C bit set/reset control word

Bit 7 must be 0 to indicate that the control port contains Port C bit set/reset control word. Bits 3, 2, and 1 select a bit of Port C that is to be set or reset. Bit 0 decides whether the selected bit of Port C is to be set or reset. Bits 6, 5, and 4 are not used in this control word. They are generally loaded with 000.

Thus, the meaning for the various bits of control port when it contains Port C bit set/reset control word is as follows.

Bit 0 (S/R*): 1 = Set Port C bit selected by bits 3, 2, and 1
0 = Reset Port C bit selected by bits 3, 2, and 1

Bits 3, 2, 1 (SB2, 1, 0): 000 = Select bit 0 of Port C
001 = Select bit 1 of Port C

010 = Select bit 2 of Port C
011 = Select bit 3 of Port C
100 = Select bit 4 of Port C
101 = Select bit 5 of Port C
110 = Select bit 6 of Port C
111 = Select bit 7 of Port C

Bits 6, 5, 4: Are don't cares. Generally loaded with 000

Bit 7: 0 to indicate it is Port C bit set/reset control

Thus, to reset PC_0 using the Port C bit set/reset feature we have to execute only the following two instructions.

```
MVI A, 0 000 000 0B
OUT 23H
```

Similarly, the following two instructions set PC_2.

```
MVI A, 0 000 010 1B
OUT 23H
```

The main use of Port C bit set/reset control word is for enabling or disabling Port A and Port B interrupts when they work in strobed mode operation. This will be discussed later.

8255 Programmable Peripheral Interface Chip

Explanation — 3 4 marks.
Port C bit Requirement for i) and ii), each of 4 marks.

b. Write the functional pin diagram of Intel 8279 (4)

Answer:

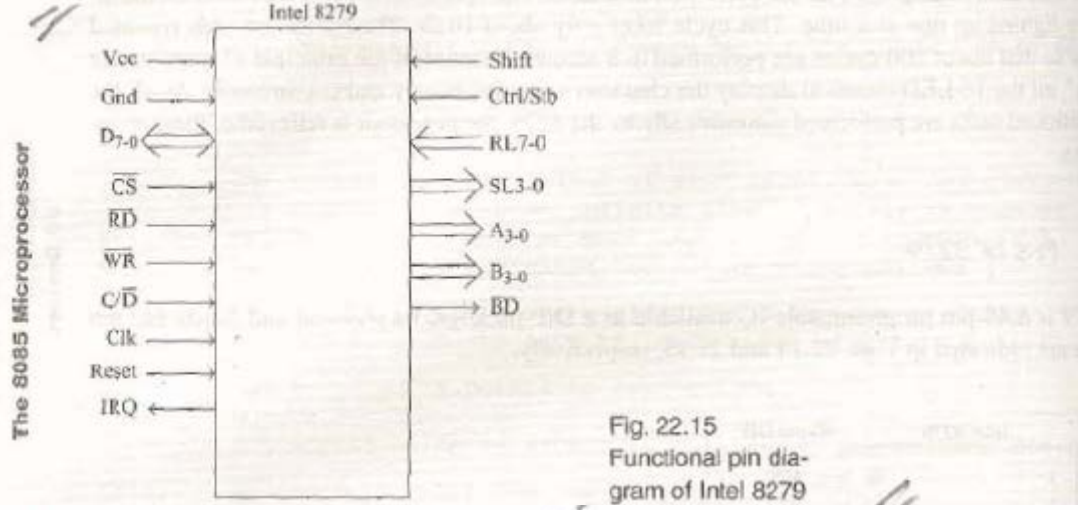


Fig. 22.15 Functional pin diagram of Intel 8279

(b) Section 22.6.1, Page no 386, Figure no 22.15 TEXT BOOK - I
Figure — 4M.

Q.7 a. Explain all the registers used in 8259 (8)

Answer:

From the point of view of a microprocessor, the 8259 is a specialized I/O port chip. It is never used for interfacing I/O devices, but only for controlling the interrupt system in a microcomputer. The 8259 has A_0 as the only address input pin. Thus only two addresses are possible for the 8259 ports as seen from a microprocessor. The two ports can be designated as low port and high port.

Low port is selected by the processor when $A_0 = 0$;
High port is selected by the processor when $A_0 = 1$.

The processor issues command words to these ports in order to configure the 8259 as per the need. There are several command words which are classified as initialization command words and operation command words. There are four initialization command words (ICW1, ICW2, ICW3, and

ICW4) and three operation command words (OCW1, OCW2, and OCW3). The processor also reads the status of 8259 by reading the low port and the high port. There are several status words to be read.

The 8259 makes use of a number of 8-bit registers shown as follows for its working.

Interrupt request register—IRR;
Interrupt mask register—IMR;
In-service register—ISR;
Slave register—SLR.

The processor writes command words, reads status words, or accesses registers using only the low port and the high port. Identification of a command word, status word, or a register is based on A_0 value, the bit values in certain bit positions, and in some cases by the context of the programming. The details about the identification of a command word, status word, or a register are provided later.

A brief description of the registers is provided as follows.

23.4.1 Interrupt Request Register

It is an 8-bit register that keeps track of active interrupt requests. Whenever an interrupt request input is activated, the corresponding bit in IRR register is set to 1. For example, if IR_4 and IR_6 inputs are activated, bits 4 and 6 of IRR are set to 1 making the contents of IRR as 01010000. The processor can only read the contents of this register but cannot write to IRR. To read the IRR contents, the processor has to issue OCW3 command to the 8259, with the LS 3 bits of the OCW3 command as 010. This results in 8259 storing the IRR status in low port of 8259. Then the processor has to read the low port of 8259.

23.4.2 Interrupt Mask Register

It is an 8-bit register that keeps track of the interrupt requests that are masked. If IR_4 and IR_6 requests should not cause an interrupt to the processor, it is easily achieved by making the contents of IMR as 01010000 that sets bits 4 and 6 of IMR to 1. Then even if IR_4 or IR_6 request is activated, the 8259 does not activate INT output and hence the processor will not be interrupted. The IMR is written by issuing the OCW1 command. The command uses high port of 8259. The processor can also read the contents of IMR register. To do this, the processor has to read the high port of 8259.

23.4.3 In-Service Register

It is an 8-bit register that keeps track of the interrupt requests that are currently being serviced. If IR_6 request is currently being serviced, then the contents of ISR will be 01000000. If IR_3 request becomes active during the service of IR_6, the 8259 sets bit 3 of ISR to 1 and activates INT output. But bit 6 of ISR remains set at 1 as IR_6 request is not fully serviced yet. Thus the contents of ISR will be 01001000. The following assumptions must hold good for this to happen.

- 8259 is operating in fully nested mode, without rotating priority, so that IR_3 has higher priority over IR_6.

- The processor has enabled interrupts in the routine for IR_6.
- IR_3 request has not been masked.

The processor can only read the contents of the ISR register but cannot write to ISR. To read the ISR contents, the processor has to issue OCW3 command to the 8259, with the LS 3 bits of the OCW3 command as 011. This results in the 8259 storing the ISR status in low port of 8259. Then the processor has to read the low port of 8259.

23.4.4 Slave Register

It is an 8-bit register. The processor writes to SLR but cannot read it. The content of this register has different meanings for Master 8259 and a Slave 8259. For Master 8259, it provides information about the IR inputs to which Slave 8259s are connected. If SLR of Master 8259 is loaded with the value 00001111, then it means that:

- There are Slave 8259s on IR_0, IR_1, IR_2, and IR_3.
- There are no Slave 8259s on IR_4, IR_5, IR_6, and IR_7.

For a Slave 8259, it provides information about the IR input of Master 8259 to which the Slave 8259 is connected. In this case, only the LS 3 bits of SLR are meaningful. If the SLR of a Slave 8259 is loaded with the value 00000101, then it means that the Slave 8259 is connected to IR_5 input of the Master 8259. The SLR is written by issuing the ICW3 command, which uses high port of 8259.

7. (a). Section 23.4 Pages 422 to 424. TEXT BOOK - I
Explanation of 4 registers, Each carries - 2M.

b. What is DMA? Explain the need for DMA data transfer. (4)

Answer:

In a microcomputer system there are basically three blocks: the microprocessor, memories such as RAM and EPROM, and I/O ports to which I/O devices are connected. Then the possible data transfers as indicated in Fig. 24.1 are as follows:

- Data transfer between microprocessor and memory (Ex. using LDA and STA instructions);
- Data transfer between microprocessor and I/O ports (Ex. using IN and OUT instructions);
- Data transfer between memory and I/O ports (DMA data transfer).

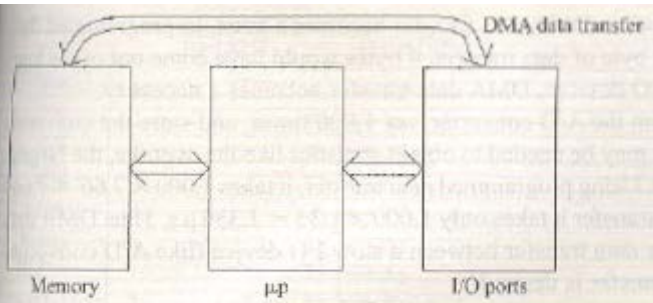


Fig. 24.1
Possible data transfers in a microcomputer system

For data transfer between microprocessor and memory or between microprocessor and I/O ports, instructions of the microprocessor are executed. As such, these data transfers are termed as programmed data transfer, which is generally used when a small amount of data transfer is involved.

Both microprocessor and memory are semiconductor chips that work at electronic speeds. Hence data transfer between microprocessor and memory is generally not a problem. In case memory is a bit slower than the microprocessor, the processor has to insert wait states. There is no problem other than this.

The I/O ports like 8255, 8212, and so on are also semiconductor chips that work at electronic speeds. However, they are used for interfacing I/O devices that are electromechanical in nature, and hence very slow compared to the speed of the microprocessor. As a result data transfer between microprocessor and I/O ports becomes more complex. In the chapter on 8255 the basic I/O, status check I/O, and the interrupt driven I/O schemes for data transfer between microprocessor and I/O ports are already discussed.

Data transfer between memory and I/O port directly (without going through the microprocessor) is called direct memory access (DMA). There are no instructions in the instruction set of the processor to perform DMA data transfer. The need for DMA data transfer and methods to perform it are discussed next.

24.2 NEED FOR DMA DATA TRANSFER

If programmed data transfer is used for reading from memory location 3456H and writing to output port number 50H, it takes 13 clocks for reading from memory location 3456H using LDA 3456H instruction and ten clocks to write to output port number 50H. Thus it takes a total of 23 clocks. If the processor is working at 3-MHz internal frequency with a clock period of 0.33 µs, it takes 7.66 µs. Similarly, for reading from input port 40H and writing to memory location 2345H, it takes 7.66 µs using programmed data transfer.

If DMA data transfer is used, reading from memory location 3456H and writing to output port number 50H requires only four clocks, which amounts to only 1.33 µs. Similarly, for reading from input port 40H and writing to memory location 2345H, it takes 1.33 µs using DMA data transfer.

Some I/O devices like the hard disk and the floppy disk are capable of performing data transfers at quite a fast rate. If we have a 1.44-MB floppy diskette rotating at 360 rpm, and has 18 sectors per track, each sector storing 512 bytes, then the data transfer rate becomes 54K bytes per second, or about 19 µs per byte. Hard disks can easily transfer data at least ten times faster. Hence it turns out to

be 1.9 μs per byte. This is the situation when DMA data transfer becomes a must. In programmed data transfer, which needs about 7.66 μs per byte of data transfer, 4 bytes would have come out of the hard disk in the same time. Thus with fast I/O devices, DMA data transfer becomes a necessity.

Sometimes we may want to read from the A/D converter, say 1,000 times, and store the converted values in 1,000 memory locations. This may be needed to obtain statistics like the average, the largest, and the smallest, of the converted values. Using programmed data transfer, it takes $1,000 \times 7.66 = 7,660$ μs of processor time. Using DMA data transfer it takes only $1,000 \times 1.33 = 1,330$ μs. Thus DMA data transfer is desirable, if not essential, for data transfer between a slow I/O device (like A/D converter) and memory, if large amount of data transfer is desired.

Thus it is to be noted that when large amount of data has to be transferred between memory and I/O port, routing each byte via microprocessor becomes a time consuming operation. If the I/O port can directly access memory for data transfer, without processor intervention, it will be more efficient. Such a scheme is known as DMA data transfer.

However, as already mentioned, there are no instructions in the instruction set of a processor to perform DMA data transfer. If DMA data transfer has to take place without processor intervention, there must be a controller circuit on the I/O port that supervises DMA data transfer. Such a controller must have the following features.

- IOR, IOW, MR, and MW control signals generation capability.
- Memory address register to generate memory address for data transfer.
- Count register to indicate the number of bytes still to be transferred.

Generally I/O ports, like 8255 PPI chip, do not possess these features. Also, if more than one I/O port needs to perform DMA data transfer, then all these I/O ports need to have such controller circuitry. To solve this problem, Intel has developed programmable DMA controller chips like Intel 8257 and Intel 8237. Intel 8257 is described in this chapter.

(b) Section 24.1 & 24.2. Pages 442 to 444 TEXT BOOK-I

Defination of DMA — 1M

Need for DMA — 3M.

c. Mention the conditions for the following modes w.r.t.8257
(i) When processor is the master & 8257 is slave.
(ii) When processor is in HOLD stack & 8257 is in master mode (4)

Answer:

D_{7-0}/A_{15-8} are used as bi-directional data lines for communication between the processor and an internal register of 8257.

A_{3-0} are input lines of 8257, to select an internal register of 8257 for communication with the processor.

IOR* and IOW* are input lines of 8257 so that the processor can read from or write to the internal registers of 8257.

MR*, MW*, and A_{7-4}, which are the output pins of 8257, are tristated by 8257.

24.3.2 Condition when Processor is in the HOLD State and 8257 is in Master Mode

D_{7-0}/A_{15-8} are used as uni-directional address output lines for sending out MS byte of address from an AR in 8257.

A_{3-0} are output lines of 8257 and are used to send out the LS 4 bits of address from an AR in 8257.

A_{7-4} are output lines of 8257 and are used to send out the MS 4 bits of LS byte of address from an AR in 8257.

IOR*, IOW*, MR*, and MW* are output pins of 8257. If the required operation is DMA read machine cycle, MR* and IOW* signals will be activated by the 8257. The IOR* and MW* signals will be in the inactive state. If the required operation is DMA write machine cycle, IOR* and MW* signals will be activated by the 8257. The MR* and IOW* signals will be in the inactive state. //

(c) Sections 24.3.1 & 24.3.2 Page no 446. Text Book-I
(i) 4 conditions — 2M.
(ii) 4 conditions — 2M.

Q.8 a. Explain the internal architecture of 8253. (8)

Answer:

Intel 8253 is a 24-pin programmable IC available as a DIP package. It has three independent counters each of 16-bits width. In addition, there is a control port to decide the mode of working of the three counters. Its physical and functional pin diagrams are indicated in Figs. 25.1 and 25.2, respectively.

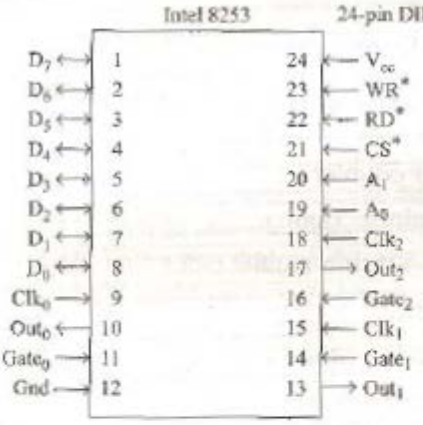


Fig. 25.1
Pin diagram of Intel 8253

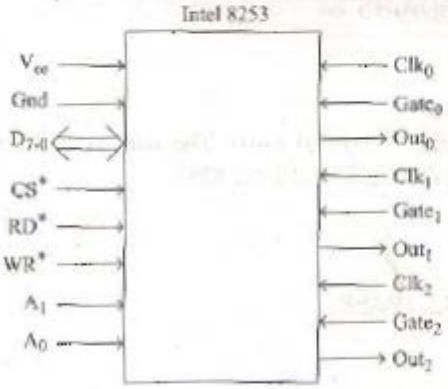


Fig. 25.2
Functional pin diagram
of Intel 8253

V_{cc} and Gnd:	Power supply and ground pins. 8253 uses +5V power supply.
D_{7-0}:	Eight bi-directional data pins for communication with the processor.
RD*:	Active low input pin that is activated by the processor to read counter information from the 8253.
WR*:	Active low input pin that is activated by the processor to write counter and control information to the 8253.
CS*:	Active low input pin used for selecting the chip.

A_1, A_0: Address input pins. They are used along with RD*, WR*, and CS* to select one of the counters or the control port. The control port can be written only from the processor. The processor cannot read it. The counters can be read or written. Table 25.1 provides the details.

Table 25.1 Selection of a counter port or control port

A_1	A_0	RD*	WR*	CS*	Operation
0	0	0	1	0	*Read Counter0
0	1	0	1	0	*Read Counter1
1	0	0	1	0	*Read Counter2
1	1	0	1	0	**No operation
0	0	1	0	0	***Write to Counter0
0	1	1	0	0	***Write to Counter1
1	0	1	0	0	***Write to Counter2
1	1	1	0	0	Write to control port
X	X	X	X	1	**No operation
X	X	1	1	0	**No operation

Notes
*Read LS byte, or MS byte, or LS and then MS byte, or LS and then MS byte on the fly.
**D_{7-0} will be tristated.
***Write to LS byte, or MS byte, or LS and then MS byte. There is no write on the fly.

CLK_0: Provides clock input for Counter0.
CLK_1: Provides clock input for Counter1.
CLK_2: Provides clock input for Counter2. Maximum frequency allowed for any of these clocks is 2 MHz.
$Gate_0$: It is an input pin that controls the function of Counter0.
$Gate_1$: It is an input pin that controls the function of Counter1.
$Gate_2$: It is an input pin that controls the function of Counter2. The gate inputs have different effects in different modes on the functioning of the corresponding counters.
Out_0: It is an output pin on which Counter0 sends its output.
Out_1: It is an output pin on which Counter1 sends its output.
Out_2: It is an output pin on which Counter2 sends its output. The output generated by the timer depends on the mode of operation.

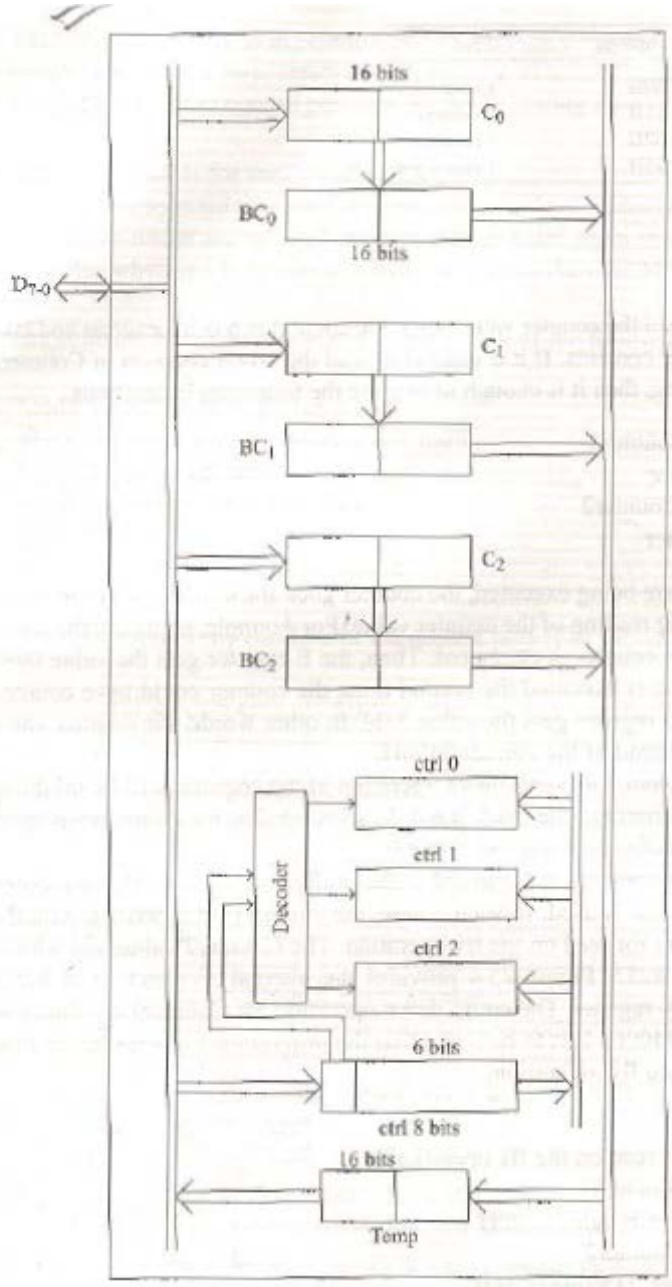


Fig. 25.4
Internal architecture of 8253

8. (a). Section 25.3. Page no 463 to 467.
Fig no 25.4. Text Book – I
Figure – 4 M. 462–463 and
Explanation – 4 M. 466

b. Discuss the following w.r.t. 8251
(i) Asynchronous Transmission
(ii) Synchronous Transmission **(8)**

Answer:

As the 8251 is used for communication purpose, it is necessary to study the various types of communications. In asynchronous transmission mode, the transmission of characters is not at regular intervals. The transmission is not synchronized with a clock.

The parallel data, to be transmitted in serial format, is sent by the processor to the transmit buffer of 8251. The transmit buffer is an 8-bit port that can only be written, but not read, by the processor. The processor writes to the transmit buffer by activating the CS* input and WR* inputs of 8251, when C/D* (control/data*) input is at 0. The information in the transmit buffer is automatically transferred to transmit shift register. This register acts like a parallel-in serial-out shift register.

When the 8251 chip is selected, the RD*, WR*, and C/D* inputs decide which register is going to be accessed by the processor, as indicated in the following.

*C/D**	*RD**	*WR**	*Action*
0	0	1	Read from receive buffer
0	1	0	Write to transmit buffer
1	0	1	Read status register
1	1	0	Write to control register

For the chip select circuit shown in Fig. 26.1, the 8251 is connected as an I/O-mapped I/O device. The transmit buffer and receive buffer have the same address 50H and the control and status registers have the same address 51H, as per this figure.

Intel 8251 is a programmable chip. It can be configured to suit our requirement by writing to the control port of 8251. The control port is an 8-bit port that can only be written, but not read, by the processor. The processor writes to the control port by activating the CS* and WR* inputs of 8251, when C/D* input is at 1. The control port of 8251 is used to supply the following types of informations to the 8251.

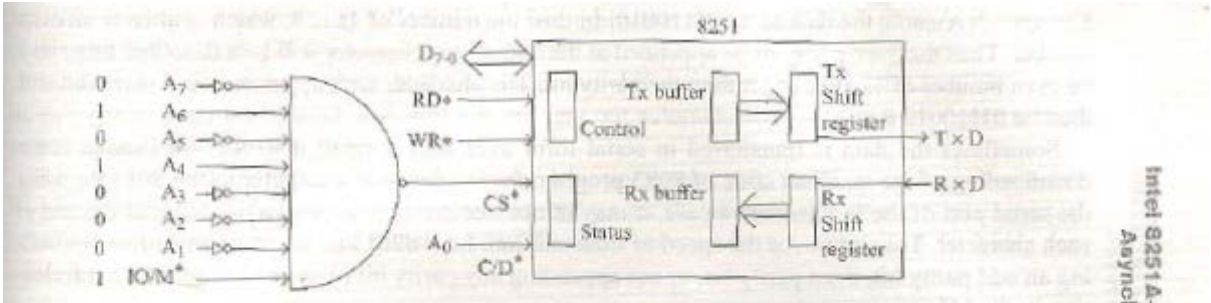


Fig. 26.1 Chip select circuit for connecting 8251 as I/O-mapped I/O

- Mode instruction (MI);
- Synchronization character or characters;
- Command instruction (CI).

Using the MI (to be described later), the 8251 can be configured as indicated in the following list in the asynchronous mode for *transmission as well as reception.*

- 5, 6, 7, or 8 bits per character;
- Even, odd, or no parity;
- 1, 1.5, or 2 stop bits;
- ×1, ×16, or ×64 mode.

Number of Bits per Character: ASCII code is normally used for representing a character. It uses a 7-bit code. Extended ASCII code that uses 8 bits are used when some special characters like graphic characters are to be represented. Intel 8251 allows the user to specify the character length as 5, 6, 7, or 8 bits by writing appropriate information in MI.

Parity Bit: Serial communication is basically used for long distance communication. During transit, the data may get corrupted because of noise on the communication medium. The receiver of the data needs to be sure that he has received the correct data. This can be achieved to a great extent by appending a parity bit at the end of the character data. There are two types of parity bits—Odd parity and Even parity.

The bit appended at the end of the character is called odd parity bit if the number of 1s in the data including the parity bit is an odd number.

Example1: Assume the data to be 01010010. In this, the number of 1s is 3, which is already an odd number. Thus the odd parity to be appended at the end of this character will be a 0. The character with appended odd parity bit will then be 01010010 **0**.

Example 2: Assume the data to be 01110010. In this, the number of 1s is 4, which is an even number. Thus the odd parity to be appended at the end of this character will be a 1, so that there may be odd number of 1s after appending the parity bit. The character with appended odd parity bit will then be 01110010 **1**.

The bit appended at the end of a character is called even parity bit if the number of 1s in the data including the parity bit is an even number.

Example1: Assume the data to be 01010010. In this, the number of 1s is 3, which is an odd number. Thus the even parity to be appended at the end of this character will be a 1, so that there may be even number of 1s after appending the parity bit. The character with appended even parity bit will then be 01010010 **1**.

Example 2: Assume the data to be 01110010. In this, the number of 1s is 4, which is already an even number. Thus the even parity to be appended at the end of this character will be a 0, so that there may be even number of 1s after appending the parity bit. The character with appended even parity bit will then be 01110010 **0**.

Sometimes the data is transferred in serial form over only a small distance. An examle is the downloading of the machine code of 8085 program from a personal computer to the 8085 kit using the serial port of the PC. In such a case, it may not be necessary to append a parity bit at the end of each character. This improves the speed of transmission. Intel 8251 can be programmed for appending an odd parity bit, even parity bit, or not appending any parity bit by writing appropriate information in the MI.

Start and Stop Bits: In the asynchronous mode of transmission, there can be any amount of time gap between the transmissions of two characters. Hence there is a need for the receiver to be informed about the beginning and end of the character. This is done using the start bit that is appended at the beginning of the character and stop bits that are appended at the end of the character.

When there is nothing to transmit, the TxD output of 8251will be in the mark state, which is a logical 1. The receiver then comes to know that the transmitter is active, but has nothing to transmit. Hence the moment the transmitter has a character to send, it sends the *start bit, which is always a logical 0*. This is followed by the data bits of the character, with LS bit transmitted first and MS bit last. The parity bit, as discussed here, is computed by the 8251 and is sent next. Finally, stop bits are sent. The number of stop bits to be transmitted at the end of a character can be programmed to be 1, 1.5, or 2 bits. The *stop bit value is always a logical 1*.

In asynchronous mode, the receiver frequency can be slightly off from the transmitting frequency without causing any problems in receiving. This is because the start bit ensures synchronization at the beginning of every character. If the number of stop bits is programmed for 1.5 or 2 bits there is that much extra time for the receiver to catch up with the transmitter frequency.

Number of Clocks for Transmitting or Receiving a Bit: Intel 8251 uses transmit clock (TxC*) input to send out the information in transmit shift register. For every falling edge of TxC*, a bit of transmit shift register is sent out on TxD output if 8251 is programmed for ×1 mode. In ×16 mode, a bit is sent out for every 16 clock transitions on TxC*. In ×64 mode, a bit is sent out for every 64 clock transitions on TxC*. This is true only for asynchronous operation. *In synchronous mode of operation, a bit is sent out for every falling edge of TxC*.*

Example: Let us say, 8251 is configured for asynchronous data transmission with character length of 5 bits, even parity, 1.5 stop bits, and ×1 mode.

Assume that the 8251 transmit buffer is loaded with 35H using the instructions

```
MVI A, 35H
OUT 50H
```

Then the waveform on TxD output pin of 8251 will be as shown in Fig. 26.2. The 8251 TxD output will be in mark state (logic 1) initially. When transmit buffer is loaded with 35H, it is automatically moved to transmit shift register. The contents of transmit shift register are sent out in serial on TxD pin only if CTS* (clear to send) input pin is activated and TxEn (transmitter enable) bit is set to 1 in the CI.

The data is shifted out on the falling edge of TxC* for every 1, 16, or 64 clock pulses depending on whether the 8251 is programmed for ×1, ×16, or ×64 modes respectively.

If conditions for transmission are met, the 8251 sends out the start bit (a logic 0) on TxD pin. Then the data bits of the character are sent out on TxD, starting with the LS bit. In the aforementioned example, only the LS 5 bits of 35H that is **10101** are sent out followed by the even parity bit which is

logic 1 in this case. This is finally followed by 1.5 stop bits that are in the logic 1 state. At the end of all this, the TxD output will again be in the mark state.

It is to be noted that the transmit buffer and transmit shift registers are only 8-bit long. The start bit, calculated parity bit (if any), and stop bits are sent out automatically from internal registers that are not accessible to the user.

The data transmission takes place at the rate of 1 bit per clock cycle, as 8251 is programmed for X1 mode. If the TxC* frequency is 1 kHz, it needs 8.5 μs for the transmission of the 5-bit character including start bit, parity bit, and the stop bits. The baud rate can be a maximum of 19.2 kHz for asynchronous transmission.

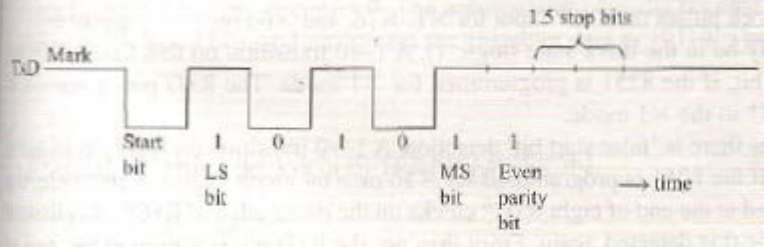


Fig. 26.2 Waveform on TxD in asynchronous transmission

The transmitter is double buffered, which means that when the data in transmit shift register is being shifted out, the transmit buffer is ready to receive another character from the processor. To indicate this situation, the TxRdy (transmitter ready) output of 8251 is activated. To be precise, TxRdy output is activated when the following are true.

- Transmit buffer is empty;
- TxEn bit is set to 1 in the CI;
- CTS* input is active.

The TxRdy output can be used to interrupt the processor. The processor can respond by sending a character to 8251 in the ISS. Assume that the processor is in disable interrupt state, or it does not send a character when it executes the ISS. A short while later the transmit shift register would send out its contents in serial form on TxD pin. Then both the transmit buffer and the transmit shift register are empty. In such a case, the TxE (transmitter empty) output is activated by the 8251.

The TxE output can also be used to interrupt the processor. It is deactivated by the 8251 when a character is written to the transmit buffer by the processor. This output indicates to the processor that the transmitter section has become completely empty and it is time to switch over from transmit mode to receive mode in the half duplex mode of operation.

26.4 SYNCHRONOUS TRANSMISSION

In synchronous transmission, characters are sent one after another without any gap, synchronized by clock pulses. Before the actual synchronous transmission, the processor writes to 8251 control port one or two synchronization characters, depending on the way the 8251 is configured. Thus the 8251 is aware of the sync (synchronization) characters to be used in the synchronous mode.

First of all, the 8251 sends out the programmed number of sync characters on the TxD pin. This is followed by the assembled data characters. The assembled data characters will not have any start or stop bits. The start and stop bits are not needed any more as one data character follows another without any time gap between them. However, the assembled data characters may have the optional parity bit. Since there are no start and stop bits for each character, the synchronous mode of transmission is faster. It is to be noted that in synchronous mode, a bit is sent out of TxD pin for every TxC* clock. It does not support ×16 and ×64 modes.

The 8251 expects a steady stream of data characters from the processor for transmission on TxD output pin. In case the transmitter becomes empty, indicated by TxE signal becoming active, the 8251 automatically sends out programmed number of sync characters on TxD pin to avoid losing synchronization. The waveform on TxD in synchronous transmission mode is shown in Fig. 26.4.

In synchronous mode, the receiver frequency must match the transmitting frequency. Else, it causes problems in receiving. This is because there is no start bit that ensures synchronization at the beginning of every character.

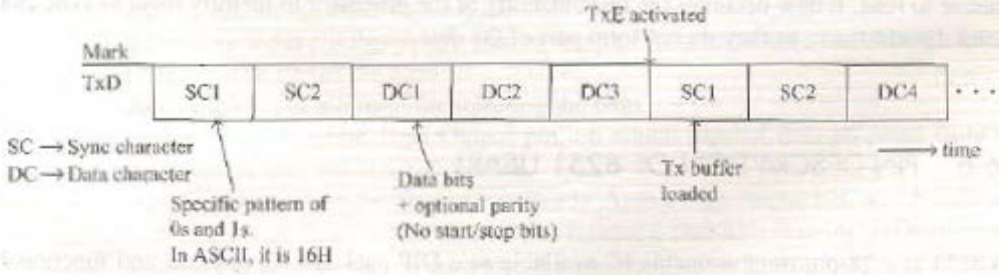


Fig. 26.4 Waveform on TxD in synchronous transmission assuming two sync characters

(b) Sections 26.2. & 26.4. Pages. 478 to 491 & 493 to 494. TEXT BOOK-I

(i) Asynchronous transmission — 4M. Explanation

(ii) Synchronous transmision Explanation — 4M.

Q.9 a. Mention the salient features and functional pin diagram of INTEL 8051. (8)

Answer:

The main features of Intel 8051 are listed as follows and details about these features are provided in the rest of this chapter and the next.

- 8-bit CPU optimized for control applications;
- Bit processing capability;
- Separate program memory and data memory spaces (Harvard architecture);
- 4 kilobytes of on-chip EPROM for program memory (range 0000–0FFFH);
- 128 bytes of on-chip data RAM (range 00–7FH);
- 64 kilobytes program memory address space. This includes 4K bytes on-chip program memory;
- 64 kilobytes data memory address space. This is excluding 128 bytes on-chip data RAM;
- 32 bi-directional and individually addressable I/O lines;
- Two numbers of 16-bit timer/counters T0 and T1;
- Full duplex UART;
- On-chip clock oscillator;
- Interrupts from six sources, two external and four internal;
- Two-level interrupt priority structure;
- Security features for EPROM parts against software piracy;
- 255 instructional opcodes, using 111 instruction types;
- 64 instruction types executed in single machine cycle of one microsecond, when crystal frequency is 12 MHz.

Intel 8051 employs Harvard type of architecture. In this type, the program memory and data memory are separated. With this feature, overlapping of program and data memory space is avoided, providing improved security for program and data.

Intel 8751 is another member of the MCS-51 family. It has 4K bytes of on-chip EPROM instead of ROM. Yet another member, Intel 8031 does not have any program memory on-chip. Thus, 8751 is an EPROM version of 8051, and 8031 is a ROM less version of 8051. Apart from the difference mentioned here, these chips have the same instruction set and internal architecture.

9 (a). Section 29.1. Page no 5.4.7 Text Book – I.

Any Four features. — 4M.

(b) Section ... Function PIN diagram — 4M

b. Explain the following instruction of 8051 (8)
(i) MOVXA, @ DPTR
(ii) CLR C
(iii) DIV A B
(iv) XCHD A, @ R1

Answer:

DPTR stands for External Data Memory
X stands for External Data Memory
This instructions moves from External Data Memory to Register A.

'CLR C' instruction results in Cy flag becoming 0. It is a 1-byte instruction, and the execution time is one machine cycle.

'DIV AB' divides the contents of A register by the contents of B register, treating them as unsigned numbers. The 8-bit quotient is stored in A and the 8-bit remainder in B register. The V flag is set to 1 only if B content is 00H before division. Also A and B values are undefined after divide operation. The Cy flag is always cleared after the divide instruction. It is a 1-byte instruction that needs four machine cycles for execution.

'XCHD A, @R1' is a typical example for the general type of instruction 'XCHD A, @Ri'. The contents of LS digit of A and LS digit of internal RAM location pointed by R1 are exchanged. If A contents are 35H, R1 contains 45H, and internal RAM location 45H contains 67H, then A contents are changed to 37H and internal RAM location 45H contents are changed to 65H. It is a 1-byte instruction, and the execution time is one machine cycle. The effect of execution of this instruction is represented as shown below.

	Before	*After*
(A)	35H	37H
(R1)	45H	
(45H)	67H	65H

b) each of 2 marks.

TEXT BOOK

I. The 8085 Microprocessor; Architecture, Programmeming and Interfacing, K. Udaya Kumar and B. S. Umashankar, Pearson Education, 2008