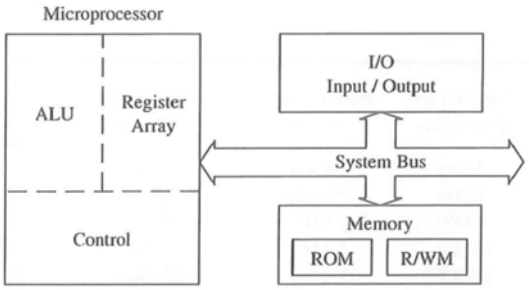
(2½ Hours) **[Total Marks: 75]**

N. B.: (1) **All** questions are **compulsory**.
(2) Make **suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labeled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is **allowed**.

1.	**Attempt *any three* of the following:**	**15**
a.	Describe a Microprocessor based system.  FIGURE 1.3 Microprocessor-Based System with Bus Architecture Describe function of each	
b.	Explain the terms:- i) Word ii) Byte iii) Nibble iv) Machine language v) Assembly language Microprocessors recognize and operate in binary numbers. However, each microproce sor has its own binary words, meanings, and language. The words are formed by con bining a number of bits for a given machine. The **word** (or word length) is defined as th number of bits the microprocessor recognizes and processes at a time. The word lengt ranges from four bits for small, microprocessor-based systems to 64 bits for high-spee large computers. Another term commonly used to express word length is byte. A **byte** defined as a group of eight bits. For example, a 16-bit microprocessor has a word lengt equal to two bytes. The term **nibble,** which stands for a group of four bits, is found als in popular computer magazines and books. A byte has two nibbles.	

Each machine has its own set of instructions based on the design of its CPU or of i microprocessor. To communicate with the computer, one must give instructions in bina language (**machine language**). Because it is difficult for most people to write programs i sets of 0s and 1s, computer manufacturers have devised English-like words to represent th binary instructions of a machine. Programmers can write programs, called **assembly lan guage** programs, using these words. Because an assembly language is specific to a give machine, programs written in assembly language are not transferable from one machine t another. To circumvent this limitation, such general-purpose languages as BASIC an

c. Explain Tristate device logic and Buffer.

Tri-state logic devices have three states: logic 1, logic 0, and high impedance. The term *Tri-State* is a trademark of National Semiconductor and is used to represent three logic states. A tri-state logic device has a third line called Enable, as shown in Figure 3.17. When this line is activated, the tri-state device functions the same way as ordinary logic devices. When the third line is disabled, the logic device goes into the high impedance state—as if it were disconnected from the system. Ordinarily, current is required to drive a device in logic 0 and logic 1 states. In the high impedance state, practically no current is drawn from the system.

In microcomputer systems, peripherals are connected in parallel between the address bus and the data bus. However, because of the tri-state interfacing devices, peripherals do not load the system buses. The microprocessor communicates with one device at a time by enabling the tri-state line of the interfacing device. Tri-state logic is critical to proper functioning of the microcomputer.

The **buffer** is a logic circuit that amplifies the current or power. It has one input line and one output line (a simple buffer is shown in Figure 3.18a). The logic level of the output

FIGURE 3.17
Tri-State Inverters with Active High and Active Low Enable Lines

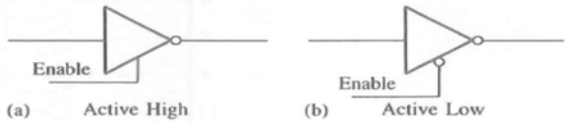
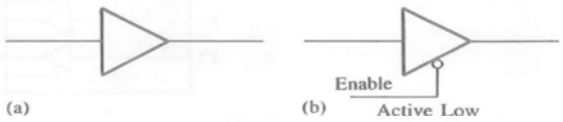


FIGURE 3.18
A Buffer and a Tri-State Buffer



is the same as that of the input; logic 1 input provides logic 1 output (the opposite of an inverter). The buffer is used primarily to increase the driving capability of a logic circuit. It is also known as a driver.

Figure 3.18b shows a tri-state buffer. When the Enable line is low, the circuit functions as a buffer; otherwise it stays in the high impedance state. The buffer is commonly used to increase the driving capability of the data bus and the address bus.

d. Write a short note on classification of memory.

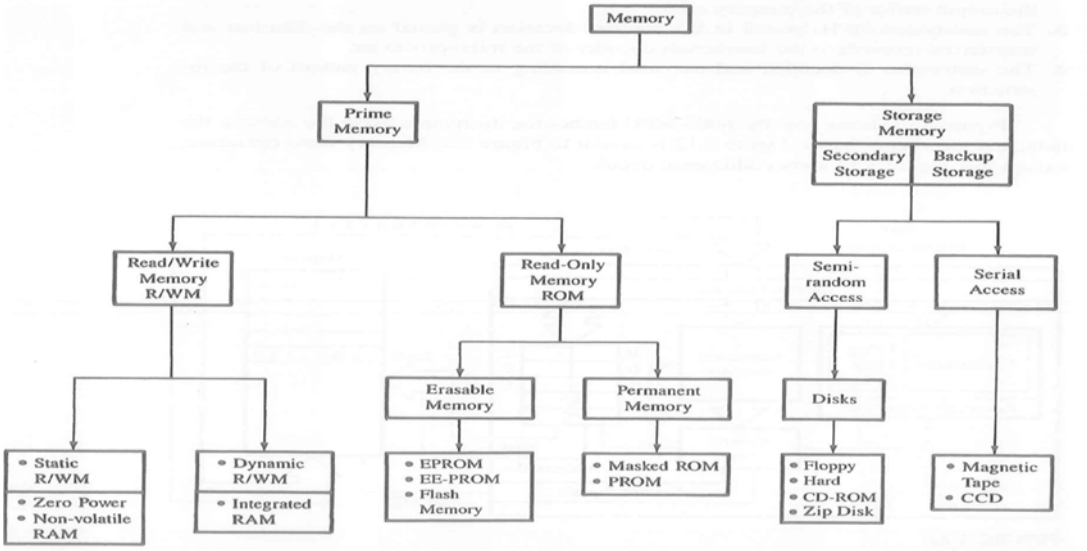
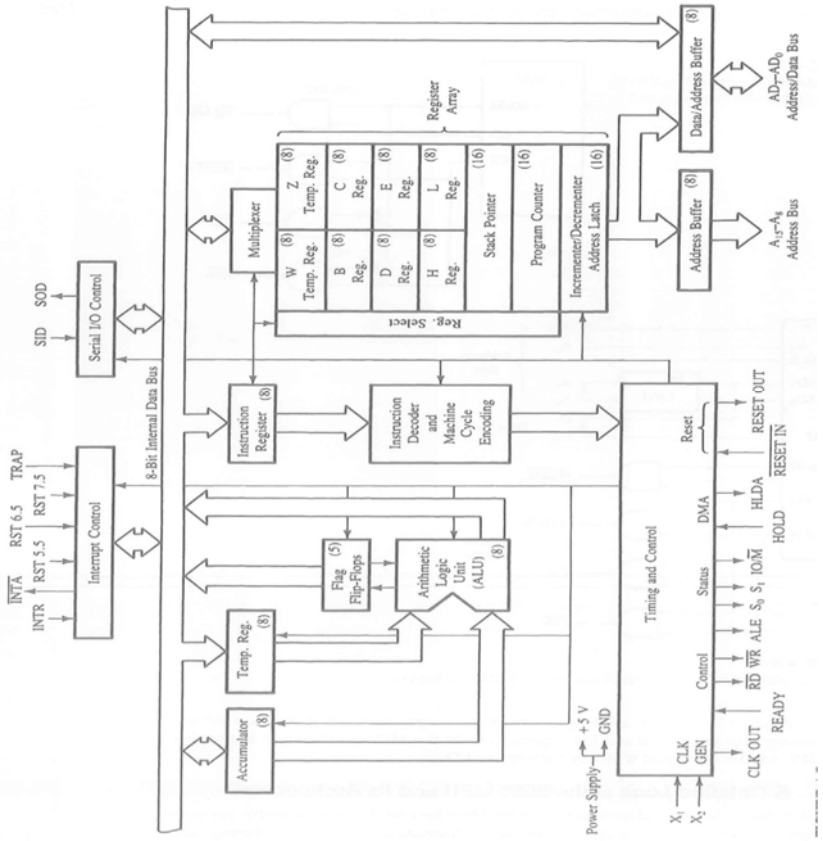


FIGURE 3.13
Memory Classification

Describe each in a single line.

e. Draw a neat label functional block diagram of 8085 microprocessor and explain the flags of the flag register.



The flags are affected by the arithmetic and logic operations in the ALU. In most of these operations, the result is stored in the accumulator. Therefore, the flags generally reflect data conditions in the accumulator—with some exceptions. The descriptions and conditions of the flags are as follows:

- **S—Sign flag:** After the execution of an arithmetic or logic operation, if bit D_7 of the result (usually in the accumulator) is 1, the Sign flag is set. This flag is used with signed numbers. In a given byte, if D_7 is 1, the number will be viewed as a negative number; if it is 0, the number will be considered positive. In arithmetic operations with signed numbers, bit D_7 is reserved for indicating the sign, and the remaining seven bits are used to represent the magnitude of a number. However, this flag is irrelevant for the operations of unsigned numbers. Therefore, for unsigned numbers, even if bit D_7 of a result is 1 and the flag is set, it does not mean the result is negative. (See Appendix A2 for a discussion of signed numbers.)
- **Z—Zero flag:** The Zero flag is set if the ALU operation results in 0, and the flag is reset if the result is not 0. This flag is modified by the results in the accumulator as well as in the other registers.
- **AC—Auxiliary Carry flag:** In an arithmetic operation, when a carry is generated by digit D_3 and passed on to digit D_4, the AC flag is set. The flag is used only internally for BCD (binary-coded decimal) operations and is not available for the programmer to change the sequence of a program with a jump instruction.
- **P—Parity flag:** After an arithmetic or logical operation, if the result has an even number of 1s, the flag is set. If it has an odd number of 1s, the flag is reset. (For example, the data byte 0000 0011 has even parity even if the magnitude of the number is odd.)
- **CY—Carry flag:** If an arithmetic operation results in a carry, the Carry flag is set; otherwise it is reset. The Carry flag also serves as a borrow flag for subtraction.

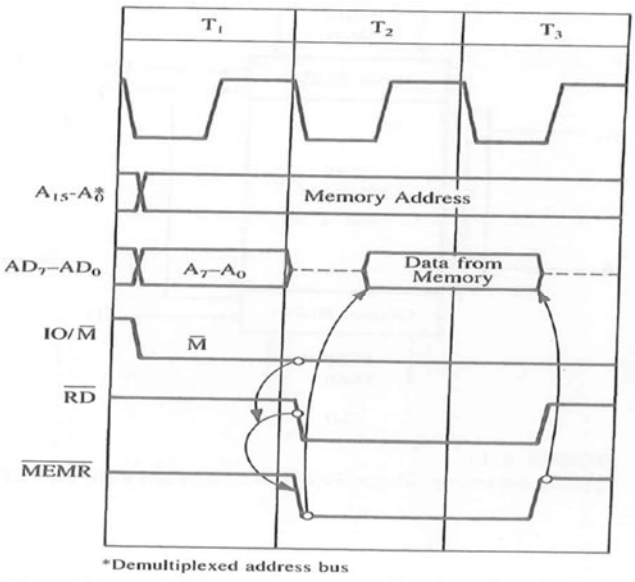
The bit positions reserved for these flags in the flag register are as follows:

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

Among the five flags, the AC flag is used internally for BCD arithmetic; the instruction set does not include any conditional jump instructions based on the AC flag. Of the remaining four flags, the Z and CY flags are those most commonly used.

f. Explain the timing diagram of The Memory Read Cycle.

FIGURE 4.12
Timing of the Memory Read Cycle



1. The microprocessor places a 16 bit address on the address bus which selects only register. The lower order is seen on AD_0 - AD_7. The higher order is seen on A_{15} - A_9

2. The remaining 8085 address lines (A_{15}–A_{11}) should be decoded to generate a Chip Select ($\overline{CS}$) signal unique to that combination of address logic (illustrated in Examples 4.3 and 4.4).
3. The 8085 provides two signals—IO/$\overline{M}$ and $\overline{RD}$—to indicate that it is a memory read operation. The IO/$\overline{M}$ and $\overline{RD}$ can be combined to generate the $\overline{MEMR}$ (Memory Read) control signal that can be used to enable the output buffer by connecting to the memory signal $\overline{RD}$.
4. Figure 4.12 also shows that memory places the data byte from the addressed register during T_2, and that is read by the microprocessor before the end of T_3.

2. Attempt *any three* of the following: **15**

a. Explain the working of the OUT instruction in 8085 microprocessor.

Opcode	Operand	Description
OUT	8-bit Port Address:	This is a two-byte instruction with the hexadecimal opcode D3, and the second byte is the port address of an output device. This instruction transfers (copies) data from the accumulator to the output device.

Typically, to display the contents of the accumulator at an output device (such as LEDs) with the address, for example, 01H, the instruction will be written and stored in memory as follows:

Memory Address	Machine Code	Mnemonics	Memory Contents
2050	D3	OUT 01H	; 2050 → 1 1 0 1 0 0 1 1 = D3H
2051	01		; 2051 → 0 0 0 0 0 0 0 1 = 01H

(Note: The memory locations 2050H and 2051H are chosen here arbitrarily for the illustration.)

If the output port with the address 01H is designed as an LED display, the instruction OUT will display the contents of the accumulator at the port. The second byte of this OUT instruction can be any of the 256 combinations of eight bits, from 00H to FFH. Therefore, the 8085 can communicate with 256 different output ports with device addresses ranging from 00H to FFH. Similarly, the instruction IN can be used to accept data from 256 different input ports. Now the question remains: How does one assign a device address or a port number to an I/O device from among 256 combinations? The decision is arbitrary and somewhat dependent on available logic chips. To understand a device address, it is necessary to examine how the microprocessor executes IN/OUT instructions.

OUT INSTRUCTION (8085)

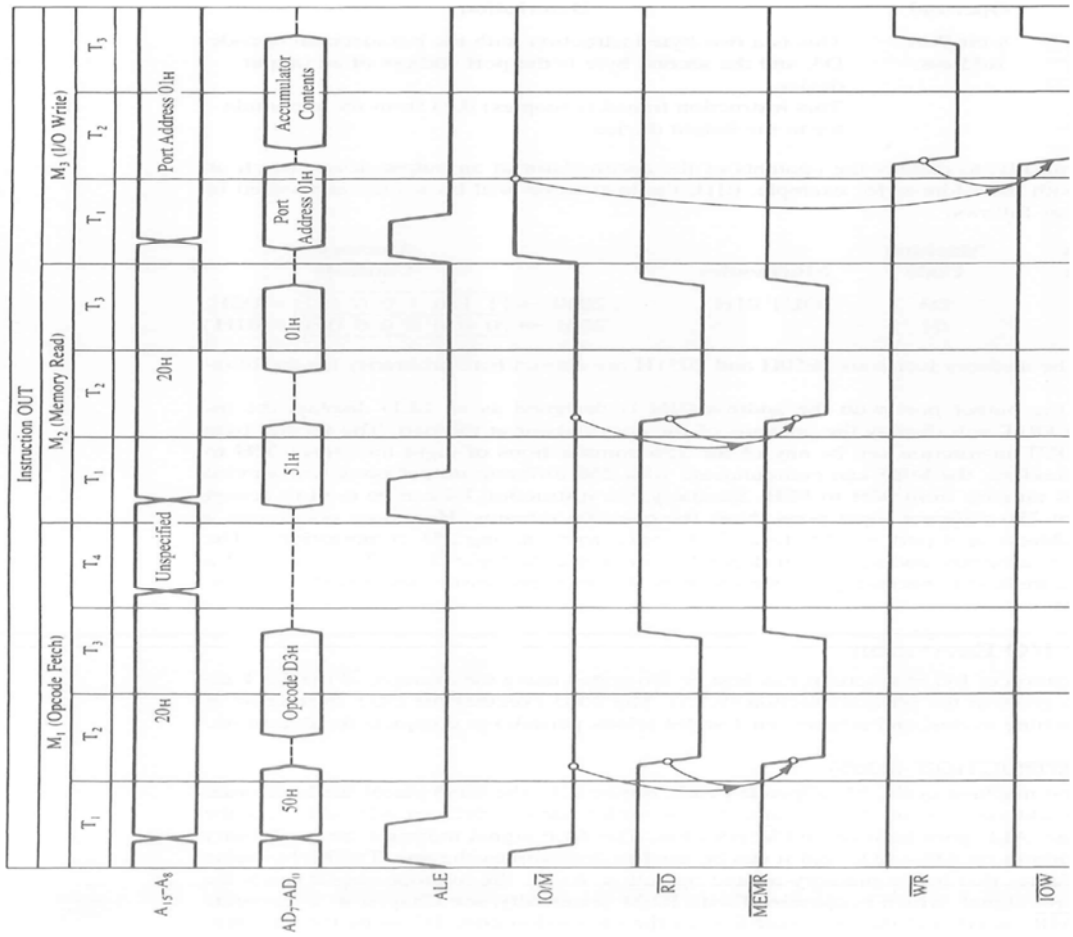
In the first machine cycle, M_1 (Opcode Fetch, Figure 5.1), the 8085 places the high-order memory address 20H on A_{15}–A_8 and the low-order address 50H on AD_7–AD_0. At the same time, ALE goes high and IO/$\overline{M}$ goes low. The ALE signal indicates the availability of the address on AD_7–AD_0, and it can be used to demultiplex the bus. The IO/$\overline{M}$, being low, indicates that it is a memory-related operation. At T_2, the microprocessor sends the $\overline{RD}$ control signal, which is combined with IO/$\overline{M}$ (externally, see Chapter 4) to generate the $\overline{MEMR}$ signal, and the processor fetches the instruction code D3 using the data bus.

When the 8085 decodes the machine code D3, it finds out that the instruction is a 2-byte instruction and that it must read the second byte.

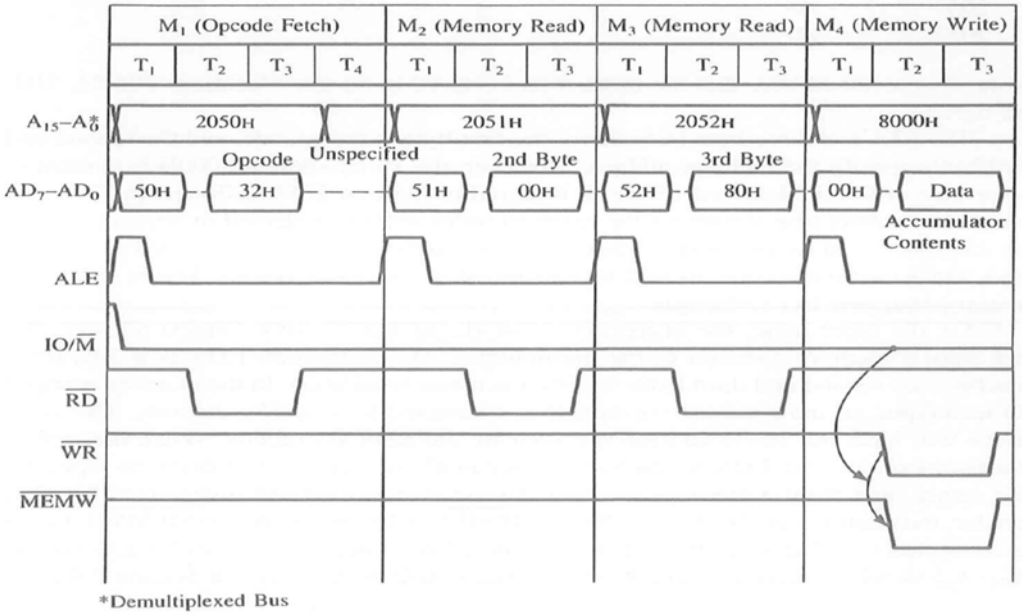
In the second machine cycle, M_2 (Memory Read), the 8085 places the next address, 2051H, on the address bus and gets the device address 01H via the data bus.

In the third machine cycle, M_3 (I/O Write), the 8085 places the device address 01H on the low-order (AD_7–AD_0) as well as the high-order (A_{15}–A_8) address bus. The IO/$\overline{M}$ signal goes high to indicate that it is an I/O operation. At T_2, the accumulator contents are placed on the data bus (AD_7–AD_0), followed by the control signal $\overline{WR}$. By ANDing the IO/$\overline{M}$ and $\overline{WR}$ signals, the $\overline{IOW}$ (see Figure 4.5) signal can be generated to enable an output device.

Figure 5.1 shows the execution timing of the OUT instruction. The information necessary for interfacing an output device is available during T_2 and T_3 of the M_3 cycle. The data byte to be displayed is on the data bus, the 8-bit device address is available on the low-order as well as high-order address bus, and availability of the data byte is indicated by the $\overline{WR}$ control signal. The availability of the device address on both segments of the address bus is redundant information; in peripheral I/O, only one segment of the address bus (low or high) is sufficient for interfacing. The data byte remains on the data bus only for two T-states, then the processor goes on to execute the next instruction. Therefore, the data byte must be latched now, before it is lost, using the device address and the control signal (Section 5.13).



b. Explain the memory mapped I/O with STA 8000H stored at memory address 2050H.



In memory-mapped I/O, the input and output devices are assigned and identified by 16-bit addresses. To transfer data between the MPU and I/O devices, memory-related instructions (such as LDA, STA, etc.)* and memory control signals (MEMR and MEMW) are used. The microprocessor communicates with an I/O device as if it were one of the memory locations. The memory-mapped I/O technique is similar in many ways to the peripheral I/O technique. To understand the similarities, it is necessary to review how a data byte is transferred from the 8085 microprocessor to a memory location or vice versa. For example, the following instruction will transfer the contents of the accumulator to the memory location 8000H.

Memory Address	Machine Code	Mnemonics	Comments
2050	32	STA 8000H	;Store contents of accumulator in mem-
			; ory location 8000H
2051	00		
2052	80		

(Note: It is assumed here that the instruction is stored in memory locations 2050H, 51H, and 52H.)

The STA is a three-byte instruction; the first byte is the opcode, and the second and third bytes specify the memory address. However, the 16-bit address 8000H is entered in the reverse order; the low-order byte 00 is stored in location 2051, followed by the high-order address 80H (the reason for the reversed order will be explained in Section 5.6). In this example, if an output device, instead of a memory register, is connected at this address, the accumulator contents will be transferred to the output device. This is called the **memory-mapped I/O technique.**

The execution of memory-related data transfer instructions is similar to the execution of IN or OUT instructions, except that the memory-related instructions have 16-bit addresses. The microprocessor requires four machine cycles (13 T-states) to execute the instruction STA (Figure 5.12). The machine cycle M_4 for the STA instruction is similar to the machine cycle M_3 for the OUT instruction.

For example, to execute the instruction STA 8000H in the fourth machine cycle (M_4), the microprocessor places memory address 8000H on the entire address bus (A_{15}–A_0). The accumulator contents are sent on the data bus, followed by the control signal Memory Write MEMW (active low).

The device selection has the following steps :-

1. Decode the address bus to generate the device address pulse.
2. AND the control signal with the device address pulse to generate the device select (I/O select) pulse.
3. Use the device select pulse to enable the I/O port.

To interface a memory-mapped input port, we can use the instruction LDA 16-bit, which reads data from an input port with the 16-bit address and places the data in the accumulator. The instruction has four machine cycles; only the fourth machine cycle differs from M_4 in Figure 5.12. The control signal will be $\overline{RD}$ rather than $\overline{WR}$, and data flow from the input port to the microprocessor.

c. List and explain the various data transfer instruction.

Opcode	Operand	Description
MOV	Rd,Rs*	Move ☐ This is a 1-byte instruction ☐ Copies data from source register Rs to destination register Rd
MVI	R,8-bit*	Move Immediate ☐ This is a 2-byte instruction ☐ Loads the 8 bits of the second byte into the register specified
OUT	8-bit port address	Output to Port ☐ This is a 2-byte instruction ☐ Sends (copies) the contents of the accumulator (A) to the output port specified in the second byte
IN	8-bit port address	Input from Port ☐ This is a 2-byte instruction ☐ Accepts (reads) data from the input port specified in the second byte, and loads into the accumulator

d. What is a instruction , instruction word size and their types based on size ?

An **instruction** is a command to the microprocessor to perform a given task on specified data. Each instruction has two parts: one is the task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or an 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

2.31 Instruction Word Size

The 8085 instruction set is classified into the following three groups according to word size or byte size.

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

1. 1-byte instructions
2. 2-byte instructions
3. 3-byte instructions

ONE-BYTE INSTRUCTIONS

A 1-byte instruction includes the opcode and the operand in the same byte. For example:

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

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

TWO-BYTE INSTRUCTIONS

In a 2-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A,32H	0011 1110	3E	First Byte
			0011 0010	32	Second Byte
Load an 8-bit data byte in register B.	MVI	B,F2H	0000 0110	06	First Byte
			1111 0010	F2	Second Byte

These instructions would require two memory locations each to store the binary codes. The data bytes 32H and F2H are selected arbitrarily as examples.

THREE-BYTE INSTRUCTIONS

In a 3-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address. For example:

Task	Opcode	Operand	Binary Code	Hex Code*	
Load contents of memory 2050H into A.	LDA	2050H	0011 1010	3A	First Byte
			0101 0000	50	Second Byte
			0010 0000	20	Third Byte
Transfer the program sequence to memory location 2085H.	JMP	2085H	1100 0011	C3	First Byte
			1000 0101	85	Second Byte
			0010 0000	20	Third Byte

e. Explain the following instruction

I)ADI : Add Immediate Data to Accumulator \

Example The accumulator contains 4AH. Add the data byte 59H to the contents of the accumulator.

Instruction: ADI 59H Hex Code: C6 59

Addition:

(A) : 4AH = 0 1 0 0 1 0 1 0

\+

(Data) : 59H = 0 1 0 1 1 0 0 1

A3H = 1 0 1 0 0 0 1 1

Flags: S = 1, Z = 0, AC = 1
P = 1, CY = 0

ii) JC : Jump on Carry

Example JC 2050 transfers control to instruction stored at 2050H when the carry flag is set to 1

iii) XRA : EX-OX the content of the Register with accumulator

Example Assume the contents of the accumulator are 77H and of register D are 56H Exclusive OR the contents of the register D with the accumulator.

Instruction: XRA D Hex Code: AA

(A): 77H = 0 1 1 1 0 1 1 1
(D): 56H = 0 1 0 1 0 1 1 0
Exclusive OR: 0 0 1 0 0 0 0 1
Flags: S = 0, Z = 0, P = 1,
CY = 0, AC = 0

iv) ORI : OR Immediate data to Accumulator

Example Assume the accumulator has data byte 03H and register C holds byte 81H. Combine the bits of register C with the accumulator bits.

Instruction: ORA C Hex Code: B1

Register contents before instruction

A	03	XX	F
B	XX	81	C

Logical OR

03H = 0 0 0 0 0 0 1 1
81H = 1 0 0 0 0 0 0 1
83H = 1 0 0 0 0 0 1 1
S = 1, Z = 0, P = 0
Flags: CY = 0, AC = 0

Register contents after instruction

		SZ AC P CY	
A	83	1 0 0 0 0 0	F
B	XX	81	C

v) JNZ : Jump on no Zero

Example JNC 2050 transfers control to instruction stored at 2050H when the ZERO flag is set to 0

f. Write an assembly program for 8085 microprocessor to add the content of C030H and C031H . Store the sum in C040H and carry at C041H.

```
        MVI A,00H
        MVI B,00H
        MVI C,00H
        LDA C030H
        MOV B,A
        LDA C031
        ADD B
        JNC  HERE
        INR C
HERE: STA C040
        MOV A,C
        STA C041
        HLT
```

3. Attempt *any three* of the following: **15**

a. Write an assembly program for 8085 microprocessor to transfer the contents of 10 memory location from C030H- C039H to C040H - C041H.

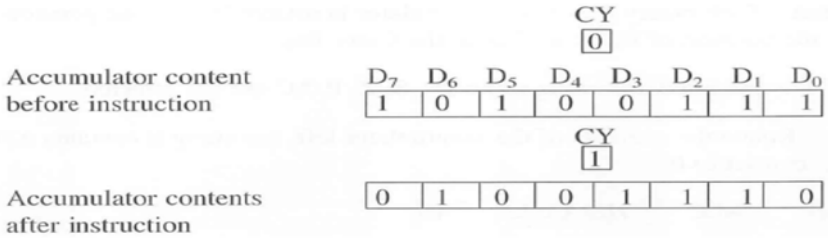
```
        MVI C, 0AH
        LXI H, CO30H
        LXI D, CO40H
HERE: MOV A,M
        STAX D
        INX H
        INX D
        DCR C
        JNZ HERE
        HLT
```

b. Explain the various Rotate Instruction for 8085 microprocessor

- RLC: Rotate Accumulator Left
- RAL: Rotate Accumulator Left Through Carry
- RRC: Rotate Accumulator Right
- RAR: Rotate Accumulator Right Through Carry

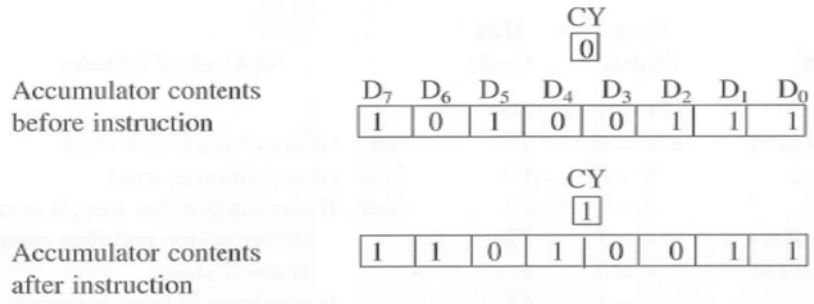
Example Rotate the contents of the accumulator through Carry, assuming the accumulator has A7H and the Carry flag is reset.

Instruction: RAL Hex Code: 17



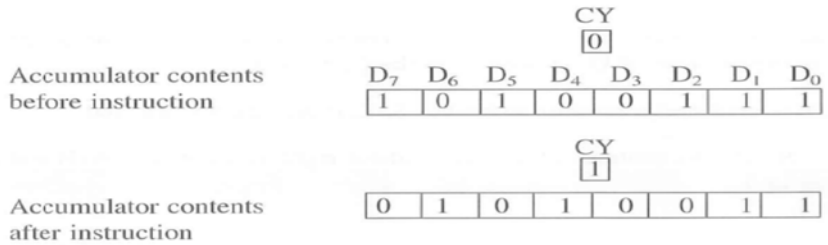
Example :rotate the contents of the accumulator right, if it contain AFH and the carry flag is reset to 0

Instruction: RRC Hex Code: 0F



Example Rotate the contents of the accumulator assuming it contains A7H and the Carry flag is reset to 0.

Instruction: RAR Hex Code: 1F



c. Calculate the time delay for the 8085-based Microcomputer with 2 MHz clock frequency.

Label	Mnemonics	Operand	T cycle
	MVI	C,FFH	7
LOOP:	DCR	C	4
	JNZ	LOOP	10/7

Clock frequency of the system f = 2 MHz

Clock period T = 1/f = $1/2 \times 10^{-6} = 0.5$ µs

Time to execute MVI = 7 T-states × 0.5

= 3.5 µs

In Figure 8.2, register C is loaded with the count FFH (255_{10}) by the instruction MVI, which is executed once and takes seven T-states. The next two instructions, DCR and JNZ, form a loop with a total of 14 (4 + 10) T-states. The loop is repeated 255 times until register C = 0.

The time delay in the loop T_L with 2 MHz clock frequency is calculated as

$$TL = (T \times \text{Loop T-states} \times N10)$$

where T_L = Time delay in the loop

T = System clock period

N_{10} = Equivalent decimal number of the hexadecimal count loaded in the delay

$$T_L = (0.5 \times 10^{-6} \times 14 \times 255)$$
$$= 1785\ \mu s$$
$$\approx 1.8\ ms$$

The T-states for JNZ instruction are shown as 10/7. This can be interpreted as follows: The 8085 microprocessor requires ten T-states to execute a conditional Jump instruction when it jumps or changes the sequence of the program and seven T-states when the program falls through the loop (goes to the instruction following the JNZ). In Figure 8.2, the loop is executed 255 times; in the last cycle, the JNZ instruction will be executed in seven T-states. This difference can be accounted for in the delay calculation by subtracting the execution time of three states. Therefore, the adjusted loop delay is

$$T_{LA} = T_L - (3\ \text{T-states} \times \text{Clock period})$$
$$= 1785.0\ \mu s - 1.5\ \mu s = 1783.5\ \mu s$$

Now the total delay must take into account the execution time of the instructions outside the loop. In the above example, we have only one instruction (MVI C) outside the loop. Therefore, the total delay is

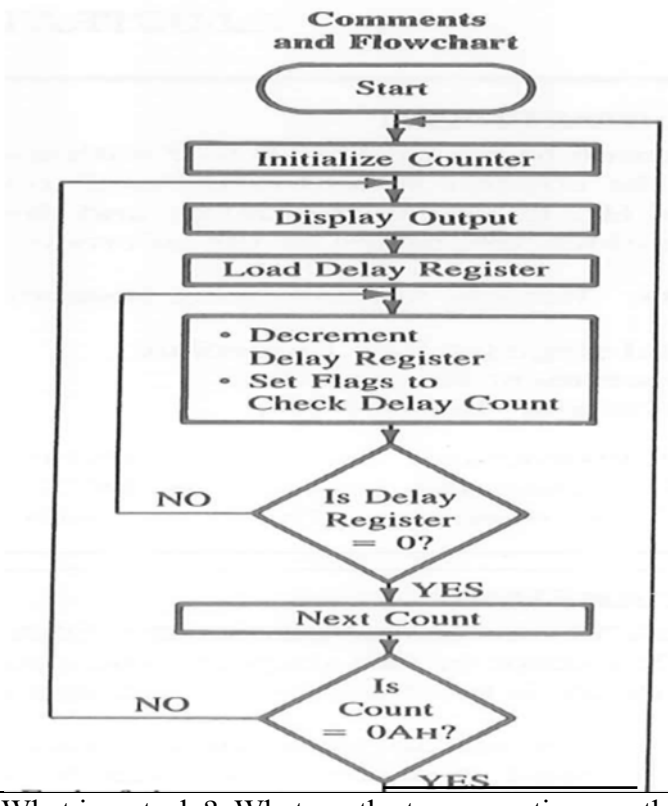
$$\text{Total Delay} = \begin{matrix}\text{Time to execute instructions} \\ \text{outside loop}\end{matrix} + \begin{matrix}\text{Time to execute} \\ \text{loop instructions}\end{matrix}$$

$$T_D = T_O + T_{LA}$$
$$= (7 \times 0.5\ \mu s) + 1783.5\ \mu s = 1787\ \mu s$$
$$\approx 1.8\ ms$$

The difference between the loop delay T_L and these calculations is only 2 μs and can be ignored in most instances.

The time delay can be varied by changing the count FFH; however, to increase the time delay beyond 1.8 ms in a 2 MHz microcomputer system, a register pair or a loop within a loop technique should be used.

d. Draw and explain a flowchart for a zero to nine counter.



e. What is a stack ? What are the two operations on the stack?

The **stack** in an 8085 microcomputer system can be described as a set of memory locations in the R/W memory, specified by a programmer in a main program. These memory locations are used to store binary information (bytes) temporarily during the execution of a program.

The beginning of the stack is defined in the program by using the instruction LXI SP, which loads a 16-bit memory address in the stack pointer register of the microprocessor. Once the stack location is defined, storing of data bytes begins at the memory address that is one less than the address in the stack pointer register. For example, if the stack pointer register is loaded with the memory address 2099H (LXI SP,2099H), the storing of data bytes begins at 2098H and continues in reversed numerical order (decreasing memory addresses such as 2098H, 2097H, etc.). Therefore, as a general practice, the stack is initialized at the highest available memory location to prevent the program from being destroyed by the stack information. The size of the stack is limited only by the available memory.

Data bytes in the register pairs of the microprocessor can be stored on the stack (two at a time) in reverse order (decreasing memory address) by using the instruction PUSH. Data bytes can be transferred from the stack to respective registers by using the instruction POP. The stack pointer register tracks the storage and retrieval of the information. Because two data bytes are being stored at a time, the 16-bit memory address in the stack pointer register is decremented by two; when data bytes are retrieved, the address is incremented by two. An address in the stack pointer register indicates that the next two memory locations (in descending numerical order) can be used for storage.

POP: Pop off Stack to Register Pair

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code	
					Reg.	**Hex**
POP	Reg. pair	1	3	10	B	C1
					D	D1
					H	E1
					PSW	F1

Description The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (such as C, E, L, and 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.

Flags No flags are modified.

Example Assume the stack pointer register contains 2090H, data byte F5 is stored in memory location 2090H, and data byte 01H is stored in location 2091H. Transfer the contents of the stack to register pair H and L.

Instruction: POP H Hex Code: E1

Register contents before instruction

H	XX	XX	L
SP	2090		

Stack contents

2090	F5
2091	01
2092	

Register contents after instruction

H	01	F5	L
SP	2092		

PUSH: Push Register Pair onto Stack

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code	
PUSH	Reg. pair	1	3	12	**Reg.**	**Hex**
					B	C5
					D	D5
					H	E5
					PSW	F5

Description The contents of the register pair designated in the operand are copied into 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.

Flags No flags are modified.

Example Assume the stack pointer register contains 2099H, register B contains 32H and register C contains 57H. Save the contents of the BC register pair on the stack.

Instruction: PUSH B Hex Code: C5

Register contents before instruction

B	32	57	C

SP 2099

Stack contents after instruction

2097	57
2098	32
2099	XX

Register contents after instruction

B	32	57	C

SP 2097

f. Explain the execution of a CALL instruction for 8085 microprocessor and its effect on the stack pointer and program counter.

The 8085 microprocessor has two instructions to implement subroutines: CALL (call a subroutine), and RET (return to main program from a subroutine). The CALL instruction is used in the main program to call a subroutine, and the RET instruction is used at the end of the subroutine to return to the main program. When a subroutine is called, the contents of the program counter, which is the address of the instruction following the CALL instruction, is stored on the stack and the program execution is transferred to the subroutine address. When the RET instruction is executed at the end of the subroutine, the memory address stored on the stack is retrieved, and the sequence of execution is resumed in the main program. This sequence of events is illustrated in Example 9.3.

INSTRUCTIONS

Opcode	Operand	
CALL	16-bit memory address of a subroutine	Call Subroutine Unconditionally ☐ This is a 3-byte instruction that transfers the program sequence to a subroutine address ☐ Saves the contents of the program counter (the address of the next instruction) on the stack ☐ Decrements the stack pointer register by two ☐ Jumps unconditionally to the memory location specified by the second and third bytes. The second byte specifies a line number and the third byte specifies a page number ☐ This instruction is accompanied by a return instruction in the subroutine

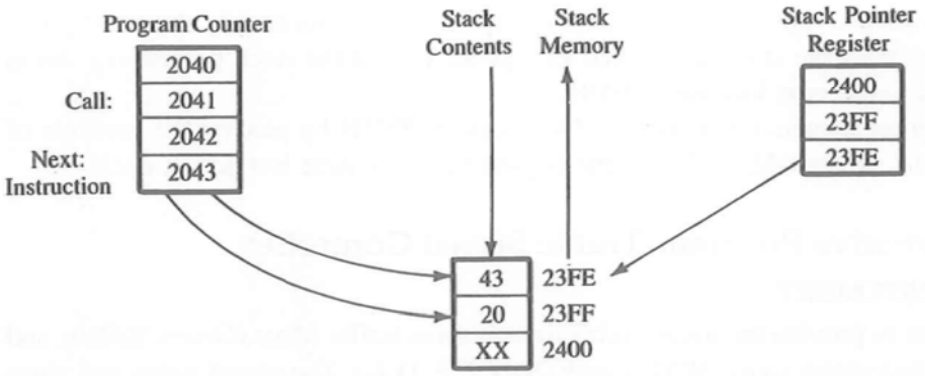
CALL EXECUTION

Memory Address	Machine Code	Mnemonics	Comments
2040	CD	CALL 2070H	;Call subroutine located at the memory
2041	70		; location 2070H
2042	20		
2043	NEXT	INSTRUCTION	

Instruction: CALL 2070H

Memory Address	Code (H)
2040	CD
2041	70
2042	20

Machine Cycles	Stack Pointer (SP) 2400	Address Bus (AB)	Program Counter (PCH) (PCL)	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FF (SP–1)	2040	20 41	CD Opcode	—
M_2 Memory Read		2041	20 42	70 Operand	→ 70
M_3 Memory Read	23FF	2042	20 43	20 Operand	→ 20
M_4 Memory Write	23FE (SP–2)	23FF	20 43	20 → (PCH)	
M_5 Memory Write	23FE	23FE	20 43	43 → (PCL)	(20) (70)
M_1 Opcode Fetch of Next Instruction		20 70 → (W)(Z)	→ 2071		(2070) → (W)(Z)



4.	**Attempt *any three* of the following:**	**15**
a	Write an assembly program for 8085 microprocessor to convert 72_{BCD} to its binary equivalent.	

```
        MVI A, 72H
        MOV B,A
        ANI 0FH
        MOV C,A
        MOV A,B
        ANI F0H
        RRC
        RRC
        RRC
        RRC
        MOV D,A
        XRA  A
        MVI E,0AH
HERE :  ADD E
        DCR D
        JNZ HERE
        ADD C
        STA C040H
        HLT
```

b Explain the following instruction :-

i) LHLD and SHLD — 02

LHLD: Load H and L Registers Direct — 02

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
LHLD	16-bit address	3	5	16	2A

Description The instruction copies the contents of the memory location pointed out by the 16-bit address in register L and copies the contents of the next memory location in register H. The contents of source memory locations are not altered.

Flags No flags are affected.

Example Assume memory location 2050H contains 90H and 2051H contains 01H. Transfer memory contents to registers HL. — 01

Instruction: LHLD 2050H Hex Code: 2A 50 20

Memory contents before instruction

2050	90
2051	01

Register contents after instruction

H		L
01	90	

SHLD: Store H and L Registers Direct

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SHLD	16-bit address	3	5	16	22

Description The contents of register L are stored in the memory location specified by the 16-bit address in the operand, and the contents of H register are stored in 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.

Flags No flags are affected.

Example Assume the H and L registers contain 01H and FFH, respectively. Store the contents at memory locations 2050H and 2051H.

Instruction: SHLD 2050H Hex Code: 22 50 20

Register contents before instruction

H		L
01	FF	

2050	FF
2051	01

Memory and register contents after instruction

H		L
01	FF	

ii) XCHG and XTHL — 02

XCHG : Exchange the content of the HL register Pair with DE Register Pair respectively

XTHL : Exchange the H and L with the top of the Stack

Example The contents of various registers and stack locations are as shown: — 01

H	A2	57	L
SP	2095		

Stacks

2095	38
2096	67

Illustrate the contents of these registers after instruction XTHL.

Register contents after XTHL

H		L
67	38	
SP	2095	

Stacks

2095	57
2096	A2

iii) SBB

SBB: Subtract Source and Borrow from Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States
SBB	Reg.	1	1	4
	Mem.	1	2	7

Hex Code	
Reg.	**Hex**
B	98
C	99
D	9A
E	9B
H	9C
L	9D
M	9E
A	9F

Description The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the results are placed in the accumulator. The contents of the operand are not altered; however, the previous Borrow flag is reset.

Example Assume the accumulator contains 37H, register B contains 3FH, and the Borrow flag is already set by the previous operation. Subtract the contents of B with the borrow from the accumulator.

Instruction: SBB B Hex Code: 98

The subtraction is performed in 2's complement; however, the borrow needs to be added first to the subtrahend:

```
(B):              3F
Borrow:         + 1
Subtrahend:     40H = 0 1 0 0  0 0 0 0
2's complement of
                40H = 1 1 0 0  0 0 0 0
(A)               = 0 0 1 1  0 1 1 1
                  0/1 1 1 1  0 1 1 1 = F7H
Complement Carry: 1/1 1 1 1  0 1 1 1
```

The Borrow flag is set to indicate the result is in 2's complement. The previous Borrow flag is reset during the subtraction.

c Explain the following :- 03

i) Cross Assembler 02

PCs and their compatibles are widely used on college campuses, and we can use PCs to develop (assemble) 8085 assembly language programs by using a program called Cross-Assembler. PCs are designed around Intel processors that have different mnemonics from those of the 8085; thus, we need a program that can translate the 8085 mnemonics, but operate under the PC microprocessor. Such a program is called a **cross-assembler.** For example, the 8085 cross-assembler from 2500 AD Software Inc. has two programs: one is an assembler named X8085 and the other is a linker with the file name LINK. After assembling a program, the Hex file (described later) can be directly transferred to R/W memory of your 8085 single-board microcomputer by using a download program. Thus, programs and/or hardware-related laboratory experiments can be easily performed.

Writing and assembling a program using a cross-assembler such as X8085 on the PC is described as follows.

1. Call an editor program and create a source file in assembly language
2. Call a Cross Assembler to Assemble the source file
3. Call a link program to generate the executable file
4. Execute the program

ii) Loader

	LOADER The Loader (or Linker) is a program that takes the Object file generated by the Assembl program and generates a file in binary code called the COM file or the EXE file. T COM (or EXE) file is the only executable file—i.e., the only file that can be executed the microcomputer. To execute the program, the COM file is called under the control the operating system and executed. In different assemblers, the COM file may be label by other names.	
d	What is the function performed by a debugger? **DEBUGGER** The Debugger is a program that allows the user to test and debug the Object file. The user can employ this program to perform the following functions: ☐ Make changes in the object code. ☐ Examine and modify the contents of memory. ☐ Set breakpoints, execute a segment of the program, and display register contents after the execution. ☐ Trace the execution of the specified segment of the program, and display the register and memory contents after the execution of each instruction. ☐ Disassemble a section of the program; i.e., convert the object code into the source code or mnemonics.	
e	Explain the steps of 8085 microprocessor interrupt process. **Step 1:** The interrupt process should be enabled by writing the instruction EI in the main program. This is similar to keeping the phone receiver on the hook. The instruction EI sets the Interrupt Enable flip-flop. The instruction DI resets the flip-flop and disables the interrupt process. **Instruction** EI (Enable Interrupt) ☐ This is a 1-byte instruction. ☐ The instruction sets the Interrupt Enable flip-flop and enables the interrupt process. ☐ System reset or an interrupt disables the interrupt process. **Instruction** DI (Disable Interrupt) ☐ This is a 1-byte instruction. ☐ The instruction resets the Interrupt Enable flip-flop and disables the interrupt. ☐ It should be included in a program segment where an interrupt from an outside source cannot be tolerated. **Step 2:** When the microprocessor is executing a program, it checks the INTR line during the execution of each instruction. **Step 3:** If the line INTR is high and the interrupt is enabled, the microprocessor completes the current instruction, disables the Interrupt Enable flip-flop and sends a signal called INTA—Interrupt Acknowledge (active low). The processor cannot accept any interrupt requests until the interrupt flip-flop is enabled again. **Step 4:** The signal INTA is used to insert a restart (RST) instruction (or a Call instruction) through *external hardware*. The RST instruction is a 1-byte call instruction (explained below) that transfers the program control to a specific memory location on page 00H and restarts the execution at that memory location after executing Step 5. **Step 5:** When the microprocessor receives an RST instruction (or a Call instruction), it saves the memory address of the next instruction on the stack. This is similar to inserting a bookmark. The program is transferred to the CALL location. **Step 6:** Assuming that the task to be performed is written as a subroutine at the specified location, the processor performs the task. This subroutine is known as a service routine. **Step 7:** The service routine should include the instruction EI to enable the interrupt again. This is similar to putting the receiver back on the hook. **Step 8:** At the end of the subroutine, the RET instruction retrieves the memory address where the program was interrupted and continues the execution. This is similar	
f	Write a short not on 8085 microprocessor vectored interrupts.	

The 8085 has five interrupt inputs (Figure 12.5). One is called INTR (discussed in the previous section), three are called RST 5.5, 6.5, and 7.5, respectively, and the fifth is called TRAP, a nonmaskable interrupt. These last four (RSTs and TRAP) are automatically vectored (transferred) to specific locations on memory page 00H without any external hardware. They do not require the INTA signal or an input port; the necessary hardware is already implemented inside the 8085. These interrupts and their call locations are as follows:

Interrupts		Call Locations
1. TRAP	⟶	0024H
2. RST 7.5	⟶	003CH
3. RST 6.5	⟶	0034H
4. RST 5.5	⟶	002CH

The TRAP has the highest priority, followed by RST 7.5, 6.5, 5.5, and INTR, in that order; however, the TRAP has a lower priority than the Hold signal used for DMA

5. Attempt *any three* of the following: **15**

a. Explain the internal structure of the Pentium Pro Processor.

The Pentium Pro is structured differently than earlier microprocessors. Early microprocessors contained an execution unit and a bus interface unit with a small cache buffering the execution unit for the bus interface unit. This structure was modified in later microprocessors, but the modifications were just additional stages within the microprocessors. The Pentium architecture is also a modification, but more significant than earlier microprocessors. Figure shows a block diagram of the internal structure of the Pentium Pro microprocessor.

The system buses, which communicate to the memory and I/O, connect to an internal level 2 cache that is often on the main board in most other microprocessor systems. The level 2 cache in the Pentium Pro is either 256K bytes or 512K bytes. The integration of the level 2 cache speeds processing and reduces the number of components in a system.

The bus interface unit (BIU) controls the access to the system buses through the level 2 cache, as it does in most other microprocessors. Again, the difference is that the level 2 cache is integrated. The BIU generates the memory address and control signals, and passes and fetches data or instructions to either a level 1 data cache or a level 1 instruction cache. Each cache is 8K bytes in size at present and may be made larger in future versions of the microprocessor. Earlier versions of the Intel microprocessor contained a unified cache that held both instructions and data. The implementation of separate caches improves performance because data-intensive programs
no longer fill the cache with data.

b. List any five Pentium instructions and explain the function of any two.

Instruction	Function
CMPXCHG8B	Compare and exchange eight bytes
CPUID	Return CPU identification code
RDTSC	Read time-stamp counter
RDMSR	Read model-specific register
WRMSR	Write model-specific register
RSM	Return from system management interrupt

The CMPXCHG8B instruction is an extension of the CMPXCHG instruction added to the

80486 instruction set. The CMPXCHG8B instruction compares the 64-bit number stored in EDX and EAX with the contents of a 64-bit memory location or register pair. For example, the CMPXCHG8B DATA2 instruction compared the eight bytes stored in memory location DATA2 with the 64-bit number in EDX and EAX. If DATA2 equals EDX:EAX, the 64-bit number stored in ECX:EBX is stored in memory location DATA2. If they are not equal, the contents of DATA2 are stored into EDX:EAX. Note that the zero flag bit indicates that the contents of EDX:EAX were equal or not equal to DATA2.

The CPUID instruction reads the CPU identification code and other information from the Pentium. To use the CPUID instruction, first load EAX with the input value and then execute CPUID. If a 0 is placed in EAX before executing the CPUID instruction, the microprocessor returns
the vendor identification in EBX, EDX, and EBX. For example, the Intel Pentium returns"GenuineIntel" in ASCII code with the "Genu" in the EBX, "ineI' in EDX, and "ntel" in ECX. TheEDX register returns information if EAX is loaded with a 1 before executing the CPUID instruction.

The RDTSC instruction reads the time-stamp counter into EDX:EAX. The time-stamp counter counts CPU clocks from the time the microprocessor is reset, where the time-stamp counter is initialized to an unknown count. Because this is a 64-bit count, a 1GHz microprocessor can accumulate a count of over 580 years before the time-stamp counter rolls over. This instruction functions only in real mode or privilege level 0 in protected mode.

The RDMSR and WRMSR instructions allow the model-specific registers to be read or written. The model-specific registers are unique to the Pentium and are used to trace, check performance, test, and check for machine errors. Both instructions use ECX to convey the register number to the microprocessor and use EDX:EAX for the 64-bit-wide read or write. Note that the register addresses are 0H–13H. See Table 18–5 for a list of the Pentium model-specific registers and their contents. As with the RDTSC instruction, these model-specific registers operate in the real or privilege level 0 of protected mode.

c. Explain the CPUID instruction in Pentium II.

CPUID Instruction

Table below lists the values passed between the Pentium II and the CPUID instruction. These are changed from earlier versions of the Pentium microprocessor. The version information returned after executing the CPUID instruction with a logic 0 in EAX is returned in EAX. The family ID is returned in bits 8 to 11; the model ID is returned in bits 4 to 7. The stepping ID is returned in bits 0 to 3. For the Pentium II, the model number is 6 and the family ID is a 3. The stepping number refers to an update number—the higher the stepping
number, the newer the version. The features are indicated in the EDX register after executing the CPUID instruction with a zero in EAX. Only two new features are returned in EDX for the Pentium II. Bit position 11
indicates whether the microprocessor supports the two new fast call instructions, SYSENTER and SYSEXIT. Bit position 23 indicates whether the microprocessor supports the MMX instruction set. Bit 16 indicates whether the microprocessor supports the page attribute table or PAT. Bit 17 indicates whether the microprocessor supports the page size

TABLE CPUID instruction for the Pentium II.

Input	EAX Output Register Contents
0	EAX Maximum allowed input to EAX for CPUID
0	EBX "uneG"
0	ECX "Inei"
0	EDX "letn"
1	EAX Version number
1	EDX Feature information
2	EAX Cache data
2	EBX Cache data
2	ECX Cache data
2	EDX Cache data

Extension found with the Pentium Pro and Pentium II microprocessors. The page size extension allows memory above 4G through 64G to be addressed. Finally, bit 24 indicates whether the fast floating-point save (FXSAVE) and restore (FXRSTOR) instructions are implemented.

d. Compare Core i3, i5 and i7 processors.

Family	Core i7	Core i5	Core i5	Core i3	Pentium
Codename	Lynnfield	Lynnfield	Clarkdale	Clarkdale	Clarkdale
Cores	4	4	2	2	2
Hyper-Threading support	Yes	No	Yes	Yes	No
Clock frequencies	2.8-2.93 GHz	2.66 GHz	3.20-3.46 GHz	2.93-3.06 GHz	2.80 GHz
L3 cache	8 MB	8 MB	4 MB	4 MB	3 MB
Graphics core	No	No	Yes	Yes	Yes
Turbo Boost	Yes	Yes	Yes	No	No
Max. memory frequency	DDR3-1600	DDR3-1333	DDR3-1333	DDR3-1333	DDR3-1067
TDP	95 W	95 W	73-87 W	73 W	73 W
Price	$284-$562	$196	$176-$284	$113-$133	$87

e. What are the features of the SPARC Architecture?

SPARC includes the following principal features:

- .A linear, 32-bit address space.
- .Few and simple instruction formats — All instructions are 32 bits wide, and are aligned on 32-bit boundaries in memory. There are only three basic instruction formats, and they feature uniform placement of opcode and register address fields. Only load and store instructions access memory and I/O.
- .Few addressing modes — A memory address is given by either "register + register" or "register+immediate."
- .Triadic register addresses— Most instructions operate on two register operands (or one register and a constant), and place the result in a third register.
- .A large "windowed" register file — At any one instant, a program sees 8 global integer registers plus a 24-register window into a larger register file.
- The windowed registers can be described as a cache of procedure arguments, local values, and return addresses.
- .A separate floating-point register file — configurable by software into 32 single-precision (32-bit), 16 double-precision (64-bit), 8 quad-precision registers (128-bit), or a mixture thereof.
- .Delayed control transfer— The processor always fetches the next instruction after a delayed control-transfer instruction. It either executes it or not, depending on the control-transfer instruction's "annul" bit.
- .Fast trap handlers— Traps are vectored through a table, and cause allocation of a fresh register window in the register file.
- .Tagged instructions — The tagged add/subtract instructions assume that the two least-significant bits of the operands are tag bits.

f. What are the various data format in the SPARC Architecture?

The SPARC architecture recognizes three fundamental data formats (or types):

.Signed Integer— 8, 16, 32, and 64 bits

.Unsigned Integer— 8, 16, 32, and 64 bits

.Floating-Point — 32, 64, and 128 bits

The format widths are defined as:

.Byte — 8 bits

.Halfword— 16 bits

.Word/Singleword — 32 bits

.Tagged Word— 32 bits (30-bit value plus 2 tag bits)

.Doubleword— 64 bits

.Quadword— 128 bits

The Signed Integer formats encode two's-complement whole numbers. The Unsigned Integer formats are general-purpose in that they do not encode any particular data type; they can represent a whole number, string, fraction, boolean value, etc. The Floating-Point formats conform to the IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985. The Tagged formats define a word in which the least-significant two bits are treated as tag bits.
