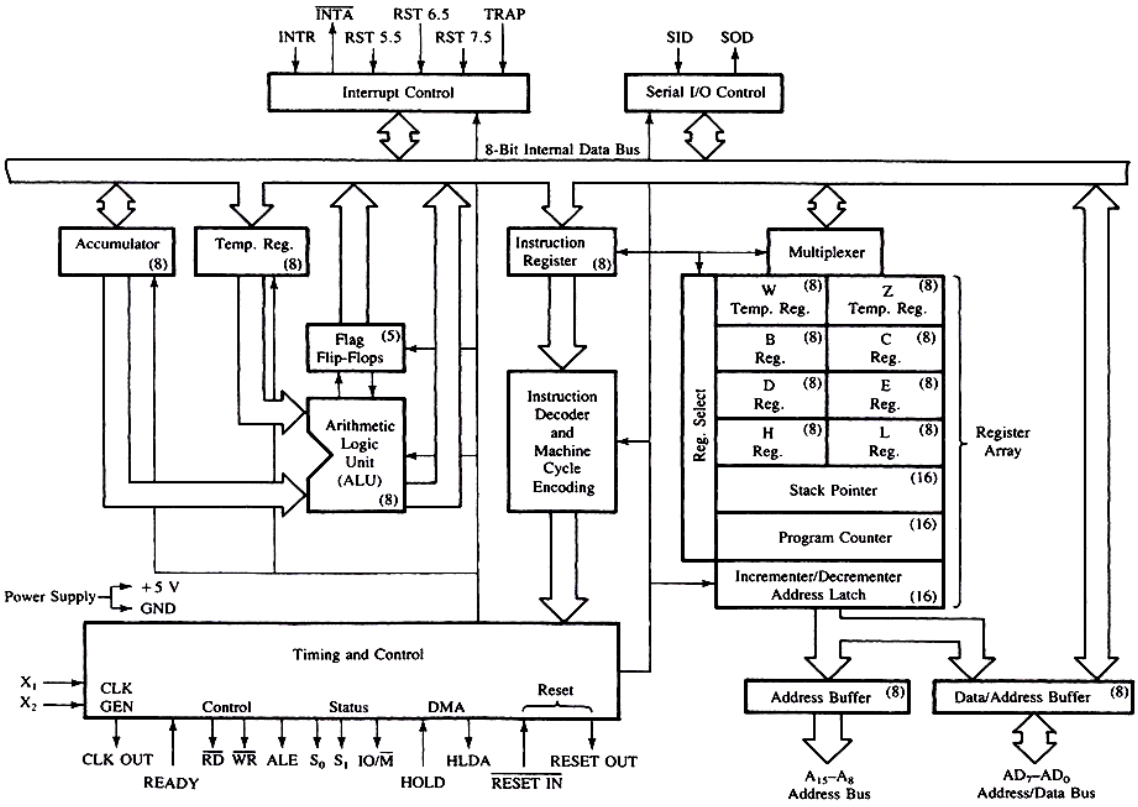
Time : 2½ Hrs.] **[Marks : 75**

Q.1 Attempt the following (any THREE) **[15]**

Q.1(a) Draw architecture of 8085. **[5]**

(A)



Q.1(b) Write a short note on flag register. **[5]**

(A) Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator. These are the set of 5 flip-flops –

1. Sign (S)
2. Zero (Z)
3. Auxiliary Carry (AC)
4. Parity (P)
5. Carry (C)

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

Fig. Bit positions of various flags in the flag register of 8085.

Its bit position is shown in the following table –

Carry flag

This flag indicates an overflow condition for arithmetic operations.

Auxiliary flag
When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.

Parity flag
This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1's, then the Parity Flag is set. For odd number of 1's, the Parity Flag is reset.

Zero flag
This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.

Sign flag
This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.

Q.1(c) What is word, nibble, assembly language and machine language. [5]

(A) **Word**:
Byte: The **byte** is a unit of digital information that most commonly consists of eight bits. Historically, the **byte** was the number of bits used to encode a single character of text in a computer and for this reason it is the smallest addressable unit of memory in many computer architectures.

Nibble: In computing, a **nibble** (often niybble or nyble to match the spelling of byte) is a four-bit aggregation, or half an octet.

Machine Language: Programs written in high-level **languages** are translated into assembly **language** or **machine language** by a compiler. Assembly **language** programs are translated into **machine language** by a program called an assembler. Every CPU has its own unique **machine language**.

Assembly Language: An **assembly** (or **assembler**) **language**, often abbreviated asm, is a low-level programming **language** for a computer, or other programmable device, in which there is a very strong (but often not one-to-one) correspondence between the **language** and the architecture's machine code instructions.

Q.1(d) What is tri- state buffer? [5]

(A) A tri–state buffer is similar to a buffer, but it adds an additional "enable" input that controls whether the primary input is passed to its output or not. If the "enable" inputs signal is **true**, the tri–state buffer behaves like a normal buffer. If the "enable" input signal is **false**, the tri–state buffer passes a high impedance (or hi–Z) signal, which effectively disconnects its output from the circuit.

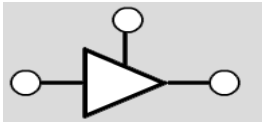
Tri state buffers are often connected to a bus which allows multiple signals to travel along the same connection.

The truth table for a tri–state buffer

Enable Input	Input A	Output
false	false	hi-Z
false	true	hi-Z
true	false	false
true	true	true

Symbols

The symbol below can be used to represent a tri–state buffer.



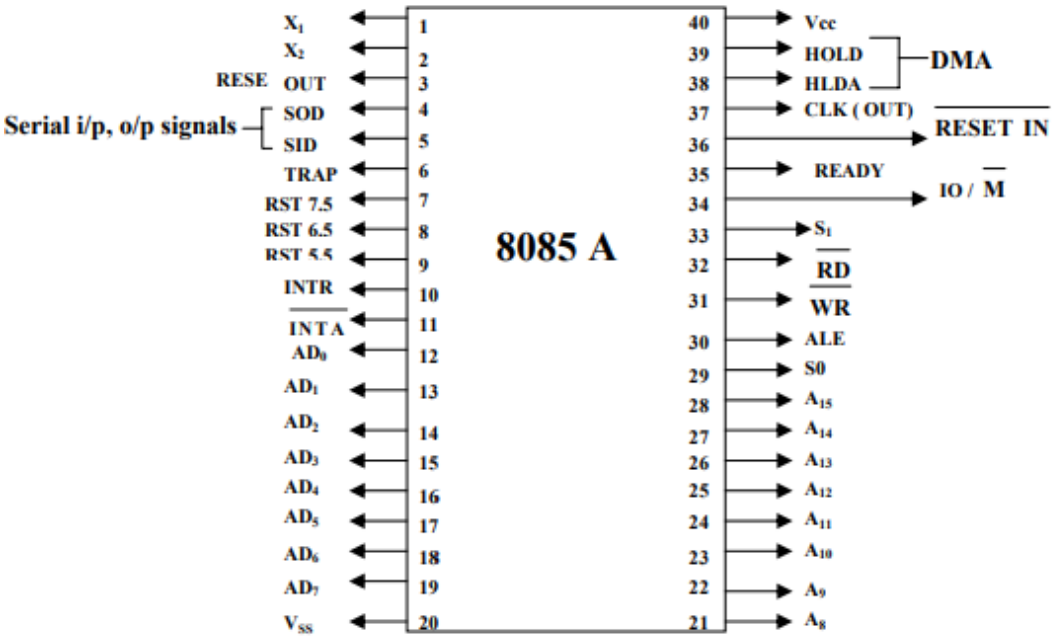
Q.1(e) With a neat diagram discuss the programming model of 8085 microprocessor. [5]
(A)

Features	High level language	Low level language
Abstraction	Strong abstraction with a computer language	Negligible abstraction with the computer language
Use interpreters	Use autocode as compilers to convert the instructions into machine language	Low level language is difficult to use since it required the elaborate technical details at each step
Flexibility	It is a readable and machine friendly language	Faster execution of programs
Execution	Slow execution of programs	Faster execution of programs
Modification	Easy modification of programs written in high level language	Modification of programs written in low level language is difficult.
Hardware	It has no correspondence with the hardware and used only to write software application program	Low level language are closely related to hardware and hence used to write to hardware programs.

Machine Language Vs Assembly Language	
Machine language is the lowest level programming language where the instructions execute directly by the CPU	Assembly language is a low–level programming language which requires an assembler to convert to machine code/object code
Compressibility	
Machine language is comprehensive only to the computers	Assembly language is comprehensible to humans.
Syntax	
A machine language consists of binary digits	Assembly language follows a syntax similar to the English language
Dependency	
Machine language varies depending on the platform	Assembly language consists of a standard set of instructions.
Applications	
Machine language is machine code	Assembly language is using for microprocessor based, real–time systems.

Q.1(f) List different addressing modes used by 8085 microprocessor. Write any one 1byte and any one 2 byte instruction to perform arithmetic operaiton using 8085 microprocessor. **[5]**

(A)



Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

- **RD** – This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** – It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals – VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** – A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** – This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

- **INTA** – It is an interrupt acknowledgment signal.
- **RESET IN** – This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT** – This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY** – This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD** – This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge)** – It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Serial I/O signals

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD** (Serial output data line) – The output SOD is set/reset as specified by the SIM instruction.
- **SID** (Serial input data line) – The data on this line is loaded into accumulator whenever a RIM instruction is executed.

Q.2 Attempt the following (any THREE) [15]

Q.2(a) Write a short note on rotate instruction. [5]

(A) **1. RLC: -** Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. Any other bit is not affected.

Eg: - RLC

2. RRC: - Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. Any other bit is not affected.
Eg: - RRC

3. **RAL: -** Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.
   Eg: - RAL
4. **RAR: -** Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.
   Eg: - RAR

Q.2(b) WAP To perform addition of two 8 bit numbers using 8085. [5]

(A) ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

PROGRAM:

MVI C, 00	Initialize C register to 00
LDA 4150	Load the value to Accumulator.
MOV B, A	Move the content of Accumulator to B register.
LDA 4151	Load the value to Accumulator.
ADD B	Add the value of register B to A
JNC LOOP	Jump on no carry.
INR C	Increment value of register C
LOOP: STA 4152	Store the value of Accumulator (SUM).
MOV A, C	Move content of register C to Acc.
STA 4153	Store the value of Accumulator (CARRY)
HLT	Halt the program.

OBSERVATION:

Input: 80 (4150)
80 (4251)
Output: 00 (4152)
01 (4153)

RESULT:

Thus the program to add two 8-bit numbers was executed.

Q.2(c) WAP To perform the subtraction of two 8 bit numbers using 8085. [5]

(A) ALGORITHM:

1. Start the program by loading the first data into Accumulator.
   Move the data to a register (B register).
2. Get the second data and load into Accumulator.
3. Subtract the two register contents.
4. Check for carry.
5. If carry is present take 2's complement of Accumulator.
6. Store the value of borrow in memory location.
7. Store the difference value (present in Accumulator) to a memory
8. Location and terminate the program.

PROGRAM:

```
MVI C, 00I Initialize C to 00
LDA 4150   Load the value to Acc.
MOV B, A   Move the content of Acc to B register.
LDA 4151   Load the value to Acc.
SUB B
JNC LOOP   Jump on no carry.
CMA        Complement Accumulator contents.
INR A      Increment value in Accumulator.
INR C      Increment value in register C
LOOP: STA 4152  Store the value of A-reg to memory address.
MOV A, C   Move contents of register C to Accumulator.
 STA 4153  Store the value of Accumulator memory address.
HLT        Terminate the program.
```

OBSERVATION:

Input: 06 (4150)
02 (4251)
Output:04 (4152)
01 (4153)

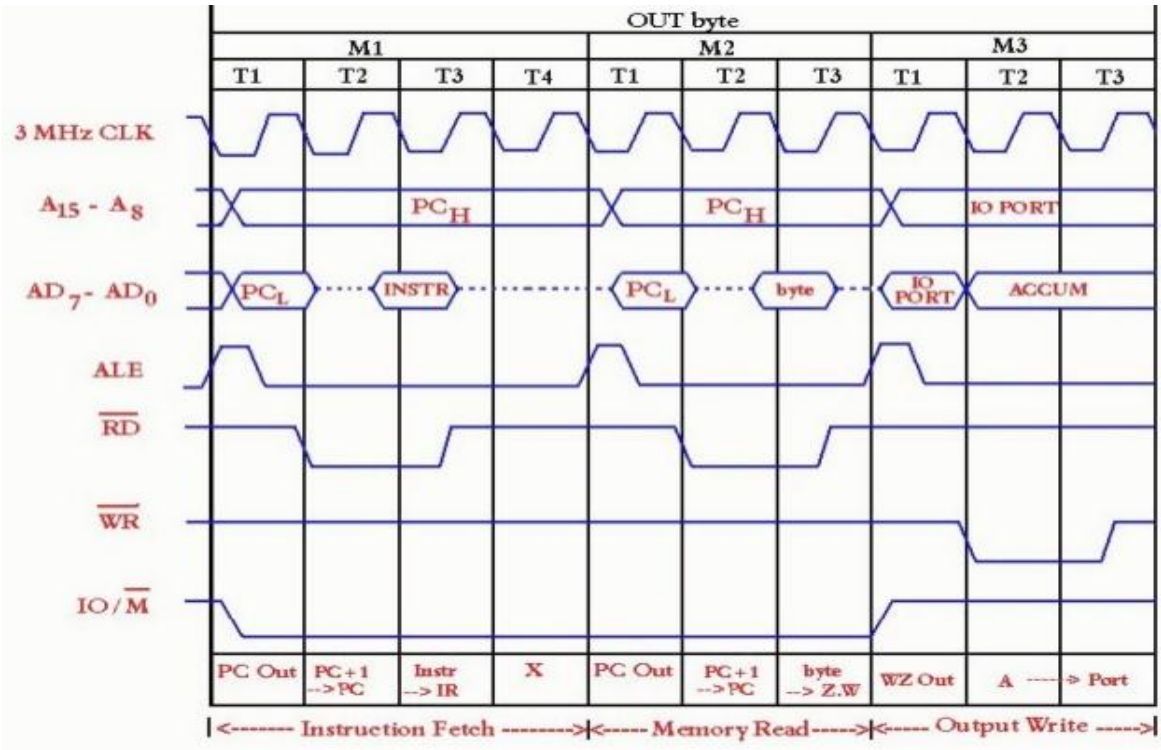
RESULT:

Thus the program to subtract two 8-bit numbers was executed.

Q.2(d) Explain the use of OUT instruction. Also explain how the instruction is executed with the help of relevant timing diagram. [5]

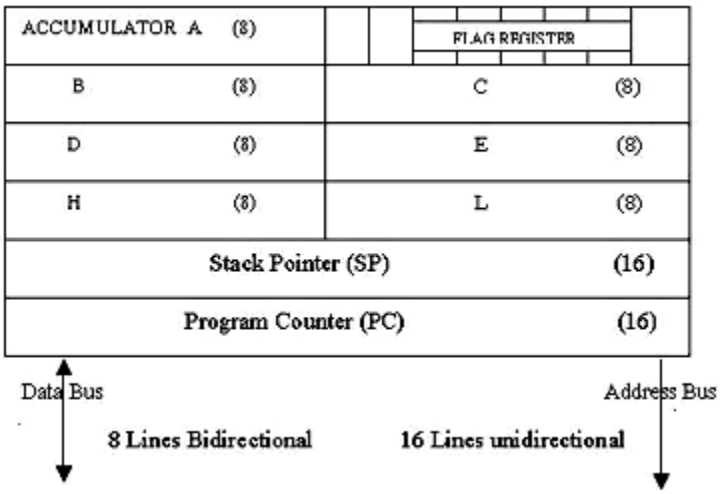
(A) **OUT -** Output data from accumulator to a port with 8-bit addresses. The contents of the accumulator are copied into the I/O port specified by the operand.

Eg: OUT F8H



Q.2(e) With a neat diagram discuss the programming model of 8085 microprocessor. [5]

(A) The 8085 programming model includes six registers, one accumulator, and one flag register, Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.



Register Diagram

Registers

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

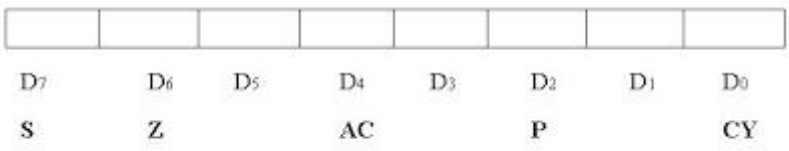
Accumulator

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

ACCUMULATOR A (8) FLAG REGISTER
B (8)
D (8)
H (8)
Stack Pointer (SP) (16)
Program Counter (PC) (16)
C (8)
E (8)
L (8)
Data Bus Address Bus
8 Lines Bidirectional 16 Lines unidirectional

Flags

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



Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.

The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP)

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

Q.2(f) List different addressing modes used by 8085 microprocessor. Write any one 1byte and any one 2 byte instruction to perform arithmetic operaiton using 8085 microprocessor. [5]

(A) Addressing Modes in 8085

These are the instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content. Addressing modes in 8085 is classified into 5 groups –

Immediate addressing mode

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand.
For example: MVI K, 20F: means 20F is copied into register K.

Register addressing mode

In this mode, the data is copied from one register to another.
For example: MOV K, B: means data in register B is copied to register K.

Direct addressing mode

In this mode, the data is directly copied from the given address to the register.
For example: LDB 5000K: means the data at address 5000K is copied to register B.

Indirect addressing mode

In this mode, the data is transferred from one register to another by using the address pointed by the register.
For example: MOV K, B: means data is transferred from the memory address pointed by the register to the register K.

Implied addressing mode

This mode doesn't require any operand; the data is specified by the opcode itself.
For example: CMP.

Q.3 Attempt the following (any THREE) : [15]

Q.3(a) Write a short note on rotate instruction. [5]

(A)

1. **RLC: -** Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. Any other bit is not affected.
   Eg: - RLC

2. **RRC: -** Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. Any other bit is not affected.
   Eg: - RRC

3. **RAL: -** Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.
   Eg: - RAL

4. **RAR: -** Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.
   Eg: - RAR

Q.3(b)Write a short note on stack . [5]

(A) In computer science, a **stack** is an abstract data type that serves as a collection of elements, with two principal operations:

- **push**, which adds an element to the collection, and
- **pop**, which removes the most recently added element that was not yet removed.

The order in which elements come off a stack gives rise to its alternative name, **LIFO** (**last in, first out**). Additionally, a peek operation may give access to the top without modifying the stack.

The name "stack" for this type of structure comes from the analogy to a set of physical items stacked on top of each other, which makes it easy to take an item off the top of the stack, while getting to an item deeper in the stack may require taking off multiple other items first.

Considered as a linear data structure, or more abstractly a sequential collection, the push and pop operations occur only at one end of the structure, referred to as the *top* of the stack. This makes it possible to implement a stack as a singly linked list and a pointer to the top element.

A stack may be implemented to have a bounded capacity. If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack.

Q.3(c)Write a procedure to execute CALL instruction. [5]

(A) The 8085 microprocessor has two instructions to implement subroutines: CALL (call a subroutine), and RET (return to main program from a subroutine). 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.

Following operations are performed by the microprocessor when CALL is executed:

1. First Saves the contents of the program counter (the address of next instruction) on the stack.
2. Then Decrements the stack pointer register by two.
3. Then Jumps unconditionally to the memory location specified by the second and third bytes. The second byte specifies a line number and third byte specifies a page number. Note that this instruction is accompanied by RET instruction in the subroutine.

Q.3(d)Explain how 8085 microprocessor performs logical operation of comparing two data. [5]

(A) 1. **CMP: -** (compare register or memory with accumulator) The contents of the operand register or memory are M compared with the contents of the accumulator.

Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:
if (A) < reg/mem: carry flag is set.if (A) = reg/mem:
zero flag is set.if (A) > reg/mem: carry and zero flags are reset.
Eg: - CMP B
CMP M

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

Q.3(e) What is time delay? Why is it needed? Explain how time delay can be generated using a register pair. **[5]**

(A) **Time delay**:- A delay used to separate the occurrence of two events, especially in a mechanical or electronic device.

Time delay can be generated using a register pair

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

Q.3(f) Explain the following concepts for subroutine program – **[5]**
(i) Nesting **(ii) Multiple Ending Subroutine**

(A) (i) **Nesting** :

In computer programming, a nested function (or nested procedure or subroutine) is a function which is defined within another function, the *enclosing function*. Due to simple recursive scope rules, a nested function is itself invisible outside of its immediately enclosing function, but can see (access) all local objects (data, functions, types, etc.) of its immediately enclosing function as well as of any function(s) which, in turn, encloses that function. The nesting is theoretically possible to unlimited depth, although only a few levels are normally used in practical programs.

(ii) **Multiple Ending Subroutine :**

"MULTIPLE ENDING SUBROUTINE" is a subroutine that because of conditions and test has more than one return point:

```
code...
...
JNC OTHER
...
RET
OTHER: ...
...
RET
```

Q.4 Attempt the following (any THREE) **[15]**

Q.4(a) Write steps to convert a binary number to BCD. Write a program to convert given 8 bit binary number to BCD. **[5]**

(A)

```
LXI SP,330AH      ;initiate stack pointer
LXI H,4350H       ;pointer to input table of BCD number
LXI B,4360H       ;pointer to output table of equivalent binary NEXTBYTE:
MOV A,M           ;get BCD number
CALL BCT_BIN      ;call BCTB_BIN conversion subroutine
STAX B            ;store binary number to output table
MOV A,L           ;copy reg. L to reg. A
CPI 59H           ;compare it with value 59H
JZ HALT           ;if it is 59H, then we have processed 10
                  ;BCD numbers hence jump to HALT
INX H             ;if it is not increment HL pair
INX B             ;and increment BC pair
JMP NEXTBYTE      ;and get next BCD
HALT :
HLT
BCT_BIN:
PUSH B            ;save BC register
PUSH D            ;save DE registers
MOV B,A           ;save BCD number
ANI OFH           ;mask most significant four bits
MOV C,A           ;save unpacked BCD1 in C
MOV A,B           ;get BCD again
ANI F0H           ;mask least significant four bits
JZ BCD1           ;if BCD2=0, the result it only BCD1
RRC               ;if not, convert most significant four
RRC               ;bits into unpacked BCD2
RRC
RRC
MOV D,A       ;save BCD2 in D
XRA A         ;clear accumulator
MVI E,0AH     ;set E as multiplier of 10
SUM:
ADD E         ;add 10 until D=0
DCR D         ;reduce BCD2 by one
JNZ SUM       ;is multiplication complete? ;if not, go back and add again
BCD1:
ADD C add BCD1
POP D         ;retrieve previous contents
POP B
RET
```

Q.4(b) . What are utility Programs? What is their use in software development systems? [5] Discuss various tools used for developing software assembly language programs.

(A) **Utility Programs:-** A program that performs a very specific task, usually related to managing system resources. Operating systems contain a number of utilities for managing disk drives, printers, and other devices.

Utilities differ from applications mostly in terms of size, complexity and function. For example, word processors, spreadsheet programs, and database applications are considered applications because they are large programs that perform a variety of functions not directly related to managing computer resources.

Utilities are sometimes installed as *memory-resident* programs. On DOS systems, such utilities are called *TSRs*.

Utility software :- is system software designed to help analyze, configure, optimize or maintain a computer. It is used to support the computer infrastructure in contrast to application software, which is aimed at directly performing tasks that benefit ordinary users.

Although a basic set of utility programs is usually distributed with an operating system (OS), utility software is not considered part of the operating system, and users often install replacements or additional utilities.

Q.4(c) What do you mean by vectored interrupts? Discuss each of 8085 vectored interrupt in brief. **[5]**

(A) **VECTORED INTERRUPT** : In vectored interrupts, the processor automatically branches to the specific address in response to an interrupt.

The vector addresses of software interrupts are given in table below.

Interrupt	Vector address
RST 0	0000_H
RST 1	0008_H
RST 2	0010_H
RST 3	0018_H
RST 4	0020_H
RST 5	0028_H
RST 6	0030_H
RST 7	0038_H

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

Q.4(d) What is cross assembler and loader **[5]**

(A) **Cross Assembler**:-

An **assembler** is a program that converts assembly language ("human readable" text - if you are a nerd) into the actual binary processor specific machine code (non-human readable binary code - unless you are a nerd). Normally the machine code generated is for the processor used in the machine it is run on. A **cross assembler** takes this conversion process a step further by allowing you to generate machine code for a *different* processor than the one the compiler is run on.

Cross assemblers are generally used to develop programs which are supposed to run on game consoles, appliances and other specialized small electronics systems which are not able to run a development environment. They can also be used to speed up development for low powered system, for example XAsm enables development on a PC based system for a Z80 powered MSX computer. Even though the MSX system is capable of running an assembler, having the additional memory, processor speed and storage capabilities like a harddisk significantly speeds up development efforts.

Loader:-

In computing, a **loader** is the part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution.

Q.4(e) Write a note on DMA. **[5]**

(A) **Direct memory access** (**DMA**) is a feature of computer systems that allows certain hardware subsystems to access main system memory(Random-access memory), independent of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller when the operation is done.

This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O.

Q.4(f) WAP to convert BCD to Binary. **[5]**

(A) Source Program:

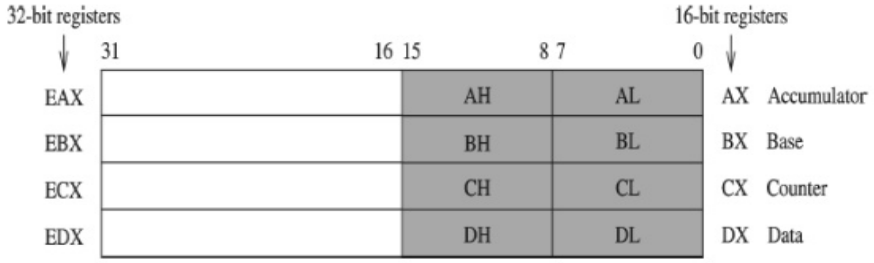
```
        LDA 2200H       : Get the BCD number
        MOV B, A        : Save it
        ANI OFH         : Mask most significant four bits
        MOV C, A        : Save unpacked BCDI in C register
        MOV A, B        : Get BCD again
        ANI FOH         : Mask least significant four bits
        RRC             : Convert most significant four bits into unpacked BCD2
        RRC
        RRC
        RRC
        MOV B, A        : Save unpacked BCD2 in B register
        XRA A           : Clear accumulator (sum = 0)
        MVI D, 0AH      : Set D as a multiplier of 10
Sum:    ADD D           : Add 10 until (B) = 0
        DCR B           : Decrement BCD2 by one
        JNZ SUM         : Is multiplication complete? i if not, go back and add again
        ADD C           : Add BCD1
      STA 2300H         : Store the result
      HLT               : Terminate program execution
```

Q.5 Attempt the following (any THREE) **[15]**

Q.5(a) What are different types of special Pentium registers? Describe them in brief. **[5]**

(A) Pentium Registers

- Four 32–bit registers can be used as
- Four 32–bit register (EAX, EBX, ECX, EDX)
- Four 16–bit register (AX, BX, CX, DX)
- Eight 8–bit register (AH, AL, BH, CH, CL, DH, DL)
- Some registers have special use
- ECX for count in loop instructions



Q.5(b) Discuss the SYSENTER and SYSEXIT instructions of Pentium II Processor. [5]

(A) The SYSENTER instruction is part of the "Fast System Call" facility introduced on the Pentium II processor. The SYSENTER instruction is optimized to provide the maximum performance for transitions to protection ring 0 (CPL = 0). The SYSENTER instruction sets the following registers according to values specified by the operating system in certain model-specific registers.

- CS register set to the value of (SYSENTER_CS_MSR)
- EIP register set to the value of (SYSENTER_EIP_MSR)
- SS register set to the sum of (8 plus the value in SYSENTER_CS_MSR)
- ESP register set to the value of (SYSENTER_ESP_MSR)

Looks like processor is trying to help us. Let's look at SYSEXIT also very quickly:

The SYSEXIT instruction is part of the "Fast System Call" facility introduced on the Pentium II processor. The SYSEXIT instruction is optimized to provide the maximum performance for transitions to protection ring 3 (CPL = 3) from protection ring 0 (CPL = 0). The SYSEXIT instruction sets the following registers according to values specified by the operating system in certain model-specific or general purpose registers.

- CS register set to the sum of (16 plus the value in SYSENTER_CS_MSR)
- EIP register set to the value contained in the EDX register
- SS register set to the sum of (24 plus the value in SYSENTER_CS_MSR)
- ESP register set to the value contained in the ECX register

Q.5(c) What are the basic categories of SPARC instructions? Discuss any two categories. [5]

(A)

1. **Arithmetic/Logical/Shift instructions**
   opcode reg1,reg2,reg3 !reg1 op reg2 -> reg3
   opcode reg1,const13,reg3 !reg1 op const13 -> reg3
   - Examples:
     add %L1,%L2,%L3 !%L1+%L2->%L3
     add %L1,1,%L1 !increment L1

2. **Load/Store Instructions**
   opcode [reg1+reg2],reg3
   opcode [reg1+const13],reg3
   - Examples:
     ld [%L1+%L2],%L3 !word at address [%L1+%L2]->%L3
     ld [%L1+8],%L2 !word at address [%L1+8]->%L2

3. **Branch Instructions**
   opcode address
   - Examples:
     call printf

Q.5(d) Write short note on CPUID instruction. **[5]**

(A) The **CPUID** opcode is a processor supplementary instruction (its name derived from CPU Identification) for the x86 architecture allowing software to discover details of the processor. It was introduced by Intel in 1993 when it introduced the Pentium and SL-enhanced 486 processors.

By using the CPUID opcode, software can determine processor type and the presence of features (like MMX/SSE). The CPUID opcode is 0Fh, A2h (as two bytes, or A20Fh as a single word) and the value in the EAX register, and in some cases the ECX register, specifies what information to return.

In assembly language the CPUID instruction takes no parameters as CPUID implicitly uses the EAX register to determine the main category of information returned. In Intel's more recent terminology, this is called the CPUID leaf. CPUID should be called with EAX = 0 first, as this will return in the EAX register the highest EAX calling parameter (leaf) that the CPU supports.

To obtain extended function information CPUID should be called with the most significant bit of EAX set. To determine the highest extended function calling parameter, call CPUID with EAX = 80000000h.

CPUID leaves greater than 3 but less than 80000000 are accessible only when the model-specific registers have IA32_MISC_ENABLE.BOOT_NT4 [bit 22] = 0 (which is so by default). As the name suggests, Windows NT4 did not boot properly unless this bit was set,[4] but later versions of Windows do not need it, so basic leaves greater than 4 can be assumed visible on current Windows systems. As of July 2014, basic valid leaves go up to 14h, but the information returned by some leaves are not disclosed in publicly available documentation, i.e. they are "reserved".

Some of the more recently added leaves also have sub-leaves, which are selected via the ECX register before calling CPUID.

Q.5(e) Compare i3, i5, i7. **[5]**

(A)

	Core i3	Core i5	Core i7
1	Entry level processor	Mid-range processor	High end processor
2	2–4 Cores	2–4 Cores	4 Cores
3	4 Threads	4 Threads	8 Threads
4	Hyper–Threading (efficient use of processor resources)	Hyper–Threading (efficient use of processor resources)	Hyper–Threading (efficient use of processor resources)
5	3–4 MB Cache	3–8 MB Cache	4–8 MB Cache
6	32 nm Silicon (less heat and energy)	32–45 nm Silicon (less heat and energy)	32–45 nm Silicon (less heat and energy)

Q.5(f) What are the features of SUN SPARC. **[5]**

(A) SPARC (Scalable Processor Architecture) is a 32-and 64-bit microprocessor architecture from Sun Microsystems that is based on reduced instruction set computing (RISC).

SPARC has become a widely-used architecture for hardware used with UNIX-based operating systems, including Sun's own Solaris systems.

Sun has made SPARC an open architecture that is available for licensing to microprocessor manufacturers.

In its most recent brand name, Ultra SPARC, microprocessors can be built for PC boards (using either Peripheral Component Interconnect or ATX) as well as for SPARC's original workstation market. As evidence of SPARC's scalability, Sun says that its Ultra SPARC III will be designed to allow up to 1,000 processors to work together.

❑ ❑ ❑ ❑ ❑ ❑