18CS306

Hall Ticket Number:

II/IV B.Tech (Regular) DEGREE EXAMINATION

NOVEMBER, 2019 — **Computer Science and Engineering**

Third Semester — **Microprocessors and Microcontrollers**

Time: Three Hours **Maximum :** 50 Marks

Answer Question No.1 compulsorily. (1X10 = 10 Marks)

Answer ONE question from each unit. (4X10=40 Marks)

1. Answer all questions (1X10=10 Marks)

a What is the difference between the Microprocessors and Microcontrollers?
b What is the maximum memory size that can be addressed by 8086?
c Draw the flag register of 8086.
d Define procedure.
e What is macro?
f What is type 4 interrupt?
g Define string.
h Define DMA process.
i List the addressing modes of 8051.
j List the features of 8051.

UNIT – I

2.a Explain the internal architecture of 8086 with a neat sketch. 7M
2.b Write an ALP in 8086 to find the largest of three numbers. 3M

(OR)

3.a Explain the directives of 8086 microprocessor. 5M
3.b Demonstrate WHILE-DO structure with suitable example program. 5M

UNIT – II

4.a Explain 8086 CALL and RET instructions. 5M
4.b Explain the 8086 stack. 5M

(OR)

5.a Differentiate between procedures and macros with examples. 5M
5.b Develop a program to arrange ten bytes of data in descending order. 5M

UNIT – III

6.a What are the predefined interrupts in 8086? 5M
6.b Explain Micro Computer system architecture with a neat diagram 5M

(OR)

7.a Explain the maximum mode operation of 8086. 5M
7.b Write 8086 ALP and to add 10 non-negative data items using string instructions. 5M

UNIT – IV

8.a Explain the internal architecture of 8259. 5M
8.b Explain the working of 8237 DMA controller With a neat block diagram. 5M

(OR)

9.a Draw the architectural diagram of 8051 microcontroller and explain in detail about each block. 5M
9.b List and explain the addressing modes of 8051. 5M

18 CS 306

II/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

November, 2019 **Scheme** **Computer Science and Engineering**

Third Semester **Microprocessors and Microcontrollers**

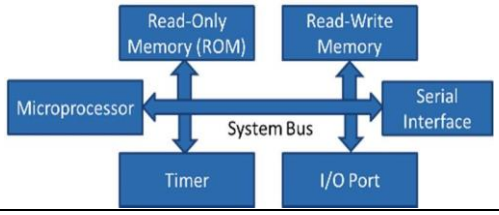
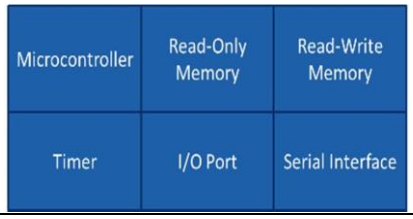
Time: Three Hours **Maximum : 50** Marks

Answer Question No.1 compulsorily. (1X10 = 10 Marks)

Answer ONE question from each unit. (4X10=40 Marks)

Answer Question No.1 compulsorily. **(1X10 = 10 Marks)**

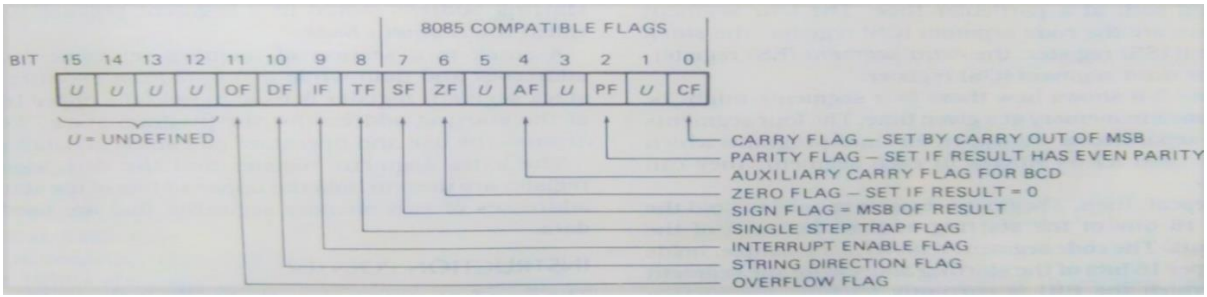
a) What is the difference between the Microprocessors and Microcontrollers? **At least 2 differences 1M**

Microprocessor	Microcontroller
	
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and I/O components

b) **What is the maximum memory size that can be addressed by 8086?**

M**ax memory size 1M** All internal registers, as well as internal and external data buses, are **16 bits** wide, which firmly established the "**16**-bit microprocessor" identity of the 8086. A 20-bit external address bus provides a **1 MB** physical address space (2^{20} = 1,048,576)

c) Draw the flag register of 8086. **1M**



d) Define procedure? **Definition 1M**

It is a subprogram, consists of set of sequence of instructions that perform a specific task.

e) **What is MACRO?** **Definition 1M**

When the repeated group of instruction is too short or not suitable to be implemented as a procedure, we use a MACRO. A macro is a group of instructions to which a name is given. Each time a macro is called in a program, the assembler will replace the macro name with the group of instructions.

f) What is type 4 interrupt? **Definition 1M**

Type 4 interrupts: This interrupt is also known as the over flow interrupt.
This interrupt used to test whether the result of addition of two signed numbers is not fit in the destination register of memory location.

g) Define string? **Definition 1M**

A string is a series of bytes or words stored in successive memory locations.
A string consists of a series of ASCII character codes.

h) Define DMA process. **Definition 1M**

Direct memory access (**DMA**) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. The **process** is managed by a chip known as a **DMA** controller (DMAC)

i) **List the addressing modes of 8051**? **List of addressing modes 1M**

The CPU can access data in various ways, which are called addressing modes
1) Immediate
2) Register
3) Direct
4) Register indirect
5) Indexed

j) **List the features of 8051 microcontroller.** **At least 2 features 1M**

- 64KB Program Memory address space
- 64KB Data Memory address space
- 4K bytes of on-chip Program Memory
- 128 bytes of on-chip Data RAM
- 32 bidirectional and individually addressable I/0 lines
- Two 16-bit timer/counters

Answer ONE question from each unit. (4X10=40 Marks)

UNIT I

2 a) Explain the internal architecture of 8086 with a neat sketch. 7M

Block diagram -2M **Explanation-5M**

The 8086 CPU is divided into two independent functional units:
Bus Interface Unit (BIU)
Execution Unit (EU)

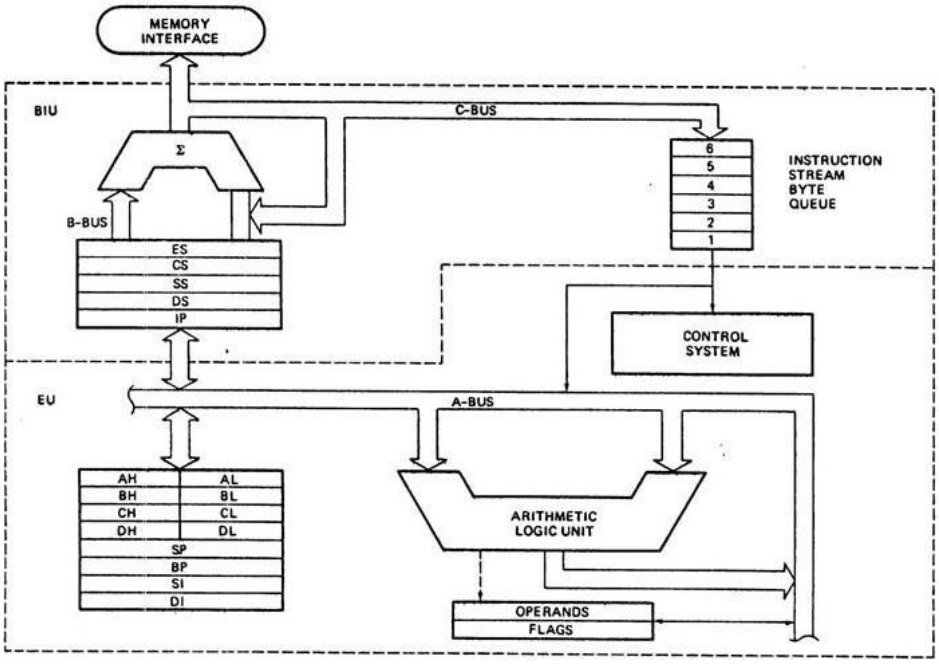


Fig. 1: Block Diagram of Intel 8086

Features of 8086

Microprocessor:

Intel 8086 was launched in 1978.
It was the first 16-bit microprocessor.
This microprocessor had major improvement over the execution speed of 8085. It is available as 40-pin Dual-Inline-Package (DIP).
It is available in three versions:
8086 (5 MHz)
8086-2 (8 MHz)
8086-1 (10 MHz)
It consists of 29,000 transistors.

The function of BIU is to:

Fetch the instruction or data from memory. Write the data to memory.
Write the data to the port. Read data from the port.

Instruction Queue

To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time from memory. All six bytes are then held in first in first out 6 byte register called instruction queue. Then all bytes have to be given to EU one by one. This pre fetching operation of BIU may be in parallel with execution operation of EU, which improves the speed execution of the instruction.

The functions of execution unit are:

To tell BIU where to fetch the instructions or data from. To decode the instructions.
To execute the instructions.

The EU contains the control circuitry to perform various internal operations. A decoder in EU decodes the instruction fetched memory to generate different internal or external control signals required to perform the operation. EU has 16-bit ALU, which can perform arithmetic and logical operations on 8-bit as well as 16-bit.

GeneralPurposeRegistersof8086:
These registers can be used as 8-bit registers individually or can be used as 16-bit in pair to have AX, BX, CX, and DX.
AX Register: AX register is also known as accumulator register that stores operands for arithmetic operation like divided, rotate.
BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.
CX Register: It is defined as a counter. It is primarily used in loop instruction to store loop counter.
DX Register: DX register is used to contain I/O port address for I/O instruction.

Segment Registers
Additional registers called segment registers generate memory address when combined with other in the microprocessor. In 8086 microprocessor, memory is divided into 4 segments as follow:

Memory Segments of 8086
Code Segment (CS): The CS register is used for addressing a memory location in the Code Segment of the memory, where the executable program is stored.
Data Segment (DS): The DS contains most data used by program. Data are accessed in the Data Segment by an offset address or the content of other register that holds the offset address.
Stack Segment (SS): SS defined the area of memory used for the stack.
Extra Segment (ES): ES is additional data segment that is used by some of the string to hold the destination data.
FlagRegistersof8086:
Flag register in EU is of 16-bit and is shown in fig. 3:

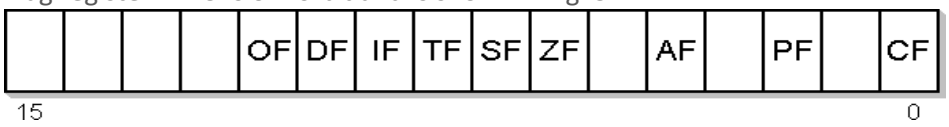


Fig. 3: Flag Register of 8086
Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. 8086 has 9 flags and they are divided into two categories:
Conditiona
l Flags
Control
Flags

Conditional Flags
Conditional flags represent result of last arithmetic or logical instruction executed.
Conditional flags are as follows:

Carry Flag (CF): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.
Auxiliary Flag (AF): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AF flag is set i.e. carry given by D3 bit to D4 is AF flag. This is not a general-purpose flag, it is used internally by the processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity Flag is reset.
Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.
Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.
Overflow Flag (OF): It occurs when signed numbers are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.

Control Flags
Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TP):
It is used for single step control.
It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.
Interrupt Flag (IF):
It is an interrupt enable/disable flag.
If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.
Direction Flag (DF):
It is used in string operation.
If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

2 b) **Write an ALP in 8086 to find the largest of three numbers.** 3M

Program 3M

```
data segment
a db 5h
b db 8h
c db 10h
max db 1 dup(0)
data ends

code segment
assume cs:code, ds:data
start:
    mov ax,data
    mov ds,ax

    mov al,a
    mov bl,b
    mov cl,c

    cmp al,bl
    ja label2
    jb label3
label2:
    cmp al,cl
    ja label1
```

```
        Jb label5
label1:
        mov max,al
        jmp ext
 label4:
        mov max,bl
        jmp ext
label3:
        cmp bl,cl
        ja label4
        jb label5
label5:
        mov max,cl

ext:   int 21h
code ends
end start
```

(OR)

3 a) Explain the directives of 8086 microprocessor. Any 5 directives 5M 5M

1.ASSUME

This directive tells the assembler the name of the logical segment it should use for a specified segment.

ASSUME CS:CODE, tells the assembler that the instructions for a program are in a logical segment named CODE.

2.DB – Define Byte

This directive is used to declare a byte type variable or to store a byte in memory location.

NAME DB 'THOMAS';

VALUES DB '0','1','2','3';

3.DD – Define Double word

This directive is used to define a variable of type double word or to reserve storage location of t double word in memory.

4.DW – Define Word

This directive is used to define a variable of type word or to reserve storage location of type word in memory.

ARRAY DW 100 DUP(0) ; defines an array of size 100 all initialized to 0.

5.ENDS

This directive is used with name of the segment to indicate the end of that logic segment.

CODE SEGMENT ; this statement starts the segment

CODE ENDS ; this statement ends the segment

6.EQU

This directive is used to give a name to some value or to a symbol. Each time the assembler finds the name in the program, it will replace the name with the value or symbol you given to that name.

7.INCLUDE

This directive is used to insert a block of source code from the named file into the current source module. ample ASSUME

Unit II

3 b) Demonstrate WHILE-DO structure with suitable example program. 5M

Definition 1M

WHILE-DO Structure 1M

WHILE-DO Structure is used in a condition when **we don't know the number of iterations**. It may be a case in which a user enters values repeatedly, and the loop doesn't end unless the user enters a particular value which satisfy the condition. This structure is implemented with conditional jump instructions.

Basic structure of a WHILE DO is given below:

WHILE some condition is present DO

action

action

any assembly language program with while do structure 3M

4 a) Explain 8086 CALL and RET instructions. 5M

- A CALL instruction in the mainline program loads the Instruction Pointer and in some cases also the code segment register with the starting address of the procedure
- At the end of the procedure, a RET instruction sends execution back to the next instruction after the CALL in the mainline program
- The 8086 CALL instruction performs 2 operations when it executes
- First, it stores the address of the instruction after the CALL instruction on the stack
- This address is called the return address because it is the address that execution will return to after the procedure executes
- If the CALL is to a procedure in the same code segment, then the call is ***near***, and only the instruction pointer contents will be saved on the stack
- If the CALL is to a procedure in another code segment, the call is ***far*** ; In this case both the instruction pointer and the code segment register contents will be saved on the stack
- Second operation of the CALL instruction is to change the contents of the instruction pointer and, in some cases, the contents of the code segment register to contain the starting address of the procedure

There are two types of calls.

- Near Call or Intra segment call.
- Far call or Inter Segment call

Operation for Near Call :

A near call refers a procedure which is in the same code segment. When 8086 executes a near CALL instruction, it decrements the stack pointer by 2 and copies the IP register contents on to the stack. Then it copies address of first instruction of called procedure.

SP □□SP-2
IP □□stores onto stack
IP □□starting address of a procedure.

Operation of FAR CALL:

A Far call refers to a procedure which is in different code segment. When 8086 executes a far call, it decrements the stack pointer by 2 and copies the contents of CS register to the stack. It the decrements the stack pointer by 2 again and copies the content of IP register to the stack. Finally it loads cs register with base address of segment having procedure and IP with address of first instruction in procedure.

SP <-sp-2
cs contents stored on stack
SP<-sp-2
IP contents stored on stack
CS Base address of segment having procedure
IP □□address of first instruction in procedure.

- **Direct within-segment NEAR CALL**

Opcode	***DispLow***	***DispHigh***

Operation:

IP ← Disp16 -- (SP) ←Return Link

- The starting address of the procedure can be anywhere in the range of **-32,768** bytes to **+32,767** bytes from the address of the instruction after the CALL

- **Indirect within-segment NEAR CALL**

Opcode	mod 010 r/m		

Operation:
IP ← Reg16 --(SP) ← Return Link
IP ← Mem16 --(SP) ← Return Link

- **Direct Inter-Segment FAR CALL**

Opcode	Offset-low	Offset-high	Seg-low	Seg-high

Operation:
CS ← SegBase
IP ← Offset

- **Indirect Inter-Segment FAR CALL**

Opcode	***Mem-low***	***Mem-high***

Operation:
CS ← SegBase
IP ← offset

- New value of IP and CS is got from memory
- The first word from memory is put in the instruction pointer and the second word from memory is put in the code segment register

The 8086 RET Instruction

- A return at the end of the procedure copies this value from the stack back to the instruction pointer to return execution to the calling program
- When the 8086 does a *far* call, it saves the contents of both the instruction pointer and the code segment register on the stack
- A RET instruction at the end of the procedure copies these values from the stack back into the IP and CS register to return execution to the mainline program
- The within segment return adding immediate to SP form is also used to return from a near procedure
- When this form executes, however, it will copy the word at the top of the stack to the instruction pointer and also add an immediate number contained in the instruction to the contents of SP

Operation for Near Procedure : For NEAR procedure ,the return is done by replacing the IP register with a address popped off from stack and then SP will be incremented by 2.

IP □□Address from top of stack
SP □□SP+2

Operation for FAR Procedure: IP register is replaced by address popped off from top of stack, then SP will be incremented by 2.The CS register is replaced with address popped off from top of stack. Again SP will be incremented by 2.

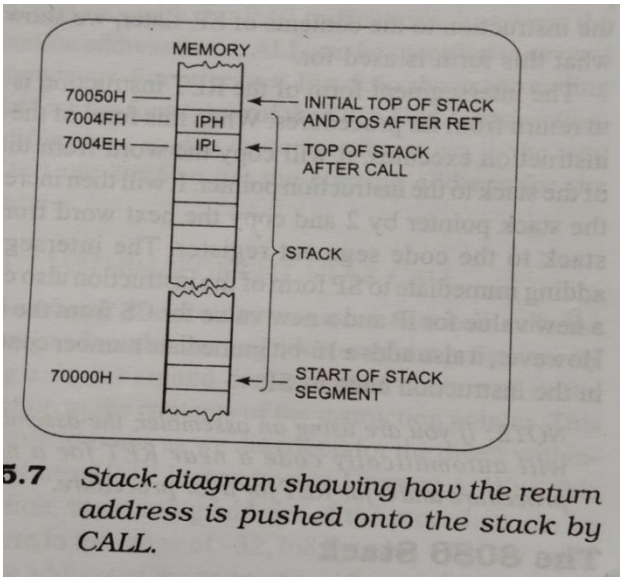
IP Address from top of stack
SP <-SP+2
CS Address from top of stack
SP <-SP+2

4 b) Explain the 8086 stack 5M

- 8086 lets us to set aside entire 64 KB segment of memory as a stack
- The stack segment register is used to hold the upper 16 bits of the starting address you give to the stack segment
- 8086 produces the physical address for a stack location by adding the offset contained in the SP register to the stack segment base address represented by the 16-bit number in the SS register
- SP register is automatically decremented by 2 before a word is written to the stack
- This means that at the start of your program you must initialize the SP register to point to the top of the memory you set aside as a stack rather than initializing it to point to the bottom location
- When a near RET instruction executes, the IP value stored in the stack will be copied back to the IP register and the SP register will be automatically incremented by 2
- After a CALL-RET sequence, then, the SP register is again pointing to the initial top-of-stack location

Stack initialization is given below in the code fragment

```
STACK_SEG  SEGMENT  STACK
                DW        40  DUP(0)
STACK_TOP  LABEL  WORD
STACK_SEG ENDS

CODE SEGMENT
                ASSUME CS:CODE, SS:STACK_SEG
START:
                MOV   AX, STACK_SEG
                MOV SS, AX
                LEA SP, STACK_TOP
                        .
                        .
                    .
CODE ENDS
END START
```



5.7 *Stack diagram showing how the return address is pushed onto the stack by CALL.*

(OR)

5 a) **Differentiate between procedures and macros with examples** **At least 6 differences 6M**

Macros	Procedures
Accessed during assembly when name given to macro is written as an instruction in the assembly program.	Accessed by CALL and RET instructions during program execution.
Machine code is generated for instructions each time a macro is called.	Machine code for instructions is put only once in the memory.
This due to repeated generation of machine code requires more memory.	This as all machine code is defined only once so less memory is required.
Parameters are passed as a part of the statement in which macro is called.	Parameters can be passed in register memory location or stack.
I don't use macros.	I do use procedures.

Defining procedures:

Assembler provides PROC and ENDP directives in order to define procedures. The directive PROC indicates beginning of a procedure. Its general form is:

Procedure_name PROC [NEAR|FAR]

NEAR | FAR is optional and gives the types of procedure. If not used, assembler assumes the proce dure as near procedure. All procedures are defined in code segment. The directive ENDPindicates end of a procedure.

Its general form is: Procedure_name ENDP

For example,

Factorial PROC NEAR

. ..

. ..

Factorial ENDP

Defining macros:

Before using macros, we have to define them. Macros are defined before the definition of segments. Assembler provides two directives for defining a macro: MACRO and ENDM.MACRO directive informs the assembler the beginning of a macro. The general form is:

Macro_name MACRO argument1, argument2, …

Arguments are optional. ENDM informs the assembler the end of the macro. Its general form is :

ENDM

5 b) **Develop a program to arrange ten bytes of data in descending order** **Program 5M** 5M

```
DATA SEGMENT
        LIST DB 98H,52H,68H,32H,72H
DATA ENDS

CODE SEGMENT
        ASSUME CS:CODE,DS:DATA
        START: MOV AX,DATA
                MOV DS,AX
                XOR AX,AX
                MOV SI,OFFSET LIST
                MOV CL,04H
        L1:     MOV DL,CL
                MOV SI,OFFSET LIST
        L2:     MOV AL,[SI]
                CMP AL,[SI+1]
                JA      L3
                XCHG AL,[SI+1]
                XCHG AL,[SI]
        L3:     INC SI
                DEC DL
                JNZ     L2
                DEC CL
                JNZ     L1
                INT 21H
CODE ENDS END START
```

Unit III

6 a) What are the predefined interrupts in 8086? 5M

Interrupt definition -1M

The meaning of '**interrupts**' is to break the sequence of operation. ¬ While the **Microprocessor** is executing a program, an '**interrupt**' breaks the normal sequence of execution of instructions, diverts its execution to some other program called **Interrupt Service Routine** (ISR)

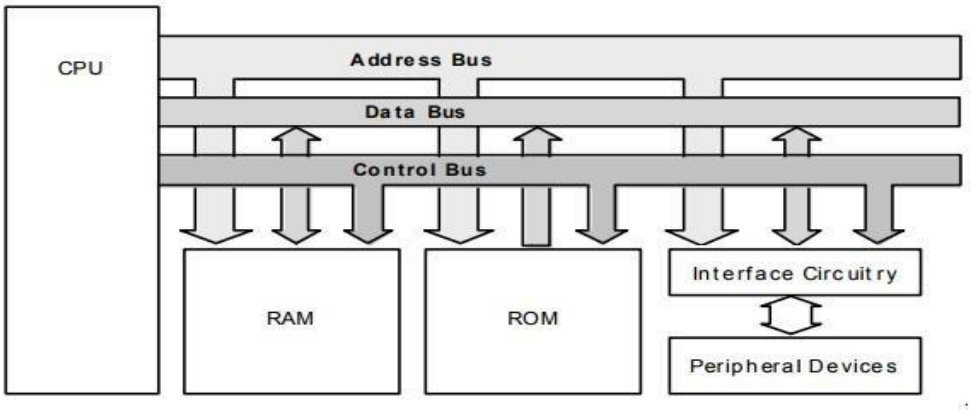
Predefined interrupts -4M

Type 0:	INT0	Divide by zero
Type 1:	INT1	Single step
Type 2:	INT2	Nonmaskable interrupt (NMI pin)
Type 3:	INT3	Breakpoint
Type 4:	INT4	Interrupt on overflow

Explanation for each interrupt with a few lines.

6 b) **Explain Micro Computer system architecture with a neat diagram.** 5M

Architecture diagram-2M



Explanation-3M

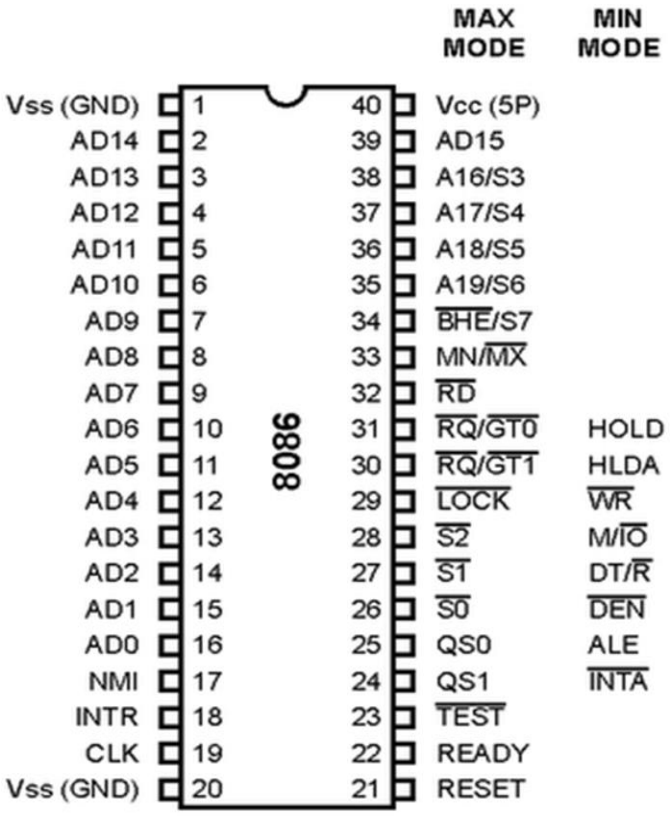
- A Microcomputer Fetches, Decodes, and Executes an instruction.
- Fetch Operation:
  - The CPU sends address of required instruction to Memory (port) through address bus.
  - Then Memory (Port) sends the instruction through data bus to the CPU.
- Decode:
  - The CPU then decodes the instruction fetched from memory.
  - Determines the set of actions.
  - Selects the sequence of microinstructions to be carried out etc.
- Execute:
  - Finally CPU performs the operation according to the instruction.
- Memory Read – Memory to CPU.
- Memory Write – CPU to Memory.
- Port Read – I/O Port to CPU

Port Write – CPU to I/O Port

(OR)

7 a) **Explain the maximum mode operation of 8086** 5M

Pin diagram-2M



Explanation-3M

The following pin function descriptions are for the microprocessor 8086 in either minimum or maximum mode.

AD0 - AD15 (I/O): Address Data Bus: These lines constitute the time multiplexed memory/IO address during the first clock cycle (T1) and data during T2, T3 and T4 clock cycles. A0 is analogous to BHE for the lower byte of the data bus, pins D0-D7. A0 bit is Low during T1 state when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. 8-bit oriented devices tied to the lower half would normally use A0 to condition chip select functions. These lines are active high and float to tri-state during interrupt acknowledge and local bus "Hold acknowledge"**.**

A19/S6, A18/S5, A17/S4, A16/S3 (0): Address/Status : During T1 state these lines are the four most significant address lines for memory operations. During I/O operations these lines are low. During memory and I/O operations, status information is available on these lines during T2, T3, and T4 states.S5: The status of the interrupt enable flag bit is updated at the beginning of each cycle. The status of the flag is indicated through this bus.

S6: When Low, it indicates that 8086 is in control of the bus. During a "Hold acknowledge" clock period, the 8086 tri-states the S6 pin and thus allows another bus master to take control of the status bus.

S3 & S4: Lines are decoded as follows:

A17/S4	A16/S3	Function
0	0	Extra segment access
0	1	Stack segment access
1	0	Code segment access
1	1	Data segment access

BHE /S7 (O): Bus High Enable/Status: During T1 state the BHE should be used to enable data onto the most significant half of the data bus, pins D15 - D8. Eight-bit oriented devices tied to the upper half of the bus would normally use BHE to control chip select functions. BHE is Low during T1 state of read, write and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus.
RD (O): READ: The Read strobe indicates that the processor is performing a memory or I/O read cycle. This signal is active low during T2 and T3 states and the Tw states of any read cycle. This signal floats to tri-state in "hold acknowledge cycle".
TEST (I): TEST pin is examined by the "WAIT" instruction. If the TEST pin is Low, execution continues. Otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
INTR (I): Interrupt Request: It is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation.
NMI (I): Non-Maskable Interrupt: An edge triggered input, causes a type-2 interrupt. A subroutine is vectored to via the interrupt vector look up table located in system memory. NMI is not maskable internally by software.
Reset (I): Reset causes the processor to immediately terminate its present activity.

Ready (I): Ready is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory or I/O is synchronized by the 8284 clock generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
CLK (I): Clock: Clock provides the basic timing for the processor and bus controller. It is asymmetric with 33% duty cycle to provide optimized internal timing. Minimum frequency of 2 MHz is required, since the design of 8086 processors incorporates dynamiccells.
MN/MX (I): Maximum / Minimum: This pin indicates what mode the processor is to operate in. In minimum mode, the 8086 itself generates all bus control signals. In maximum mode the three status signals are to be decoded to generate all the bus control signals.
Minimum Mode Pins The following 8 pins function descriptions are for the 8086 in minimum mode; MN/ MX = 1. The corresponding 8 pins function descriptions for maximum mode is explained later.
S2, S1, S0 (O): Status Pins: These pins are active during T4, T1 and T2 states and is returned to passive state (1,1,1 during T3 or Tw (when ready is inactive). These are used by the 8288 bus controller to generate all memory and I/O operation) access control signals. Any change by S2, S1, S0 during T4 is used to indicate the beginning of a bus cycle. These status lines are encoded as shown in table 3.

S0, QS1 (O): Queue – Status: Queue Status is valid during the clock cycle after which the queue operation is performed. QS0, QS1 provide status to allow external tracking of the internal 8086 instruction queue.
LOCK :It indicates to another system bus master, not to gain control of the system bus while LOCK is active Low. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the instruction. This signal is active Low and floats to tri-state OFF during 'hold acknowledge'.

RQ/GT0 and RQ/GT1 (I/O): Request/Grant

These pins are used by other processors in a multi-processor organization. Local bus masters of other processors force the processor to release the local bus at the end of the processors current bus cycle. Each pin is bi-directional and has an internal pull up resistors. Hence they may be left un-connected.

7 b) Write 8086 ALP and to add 10 non-negative data items using string instructions. 5M

Program -6M

Ans:

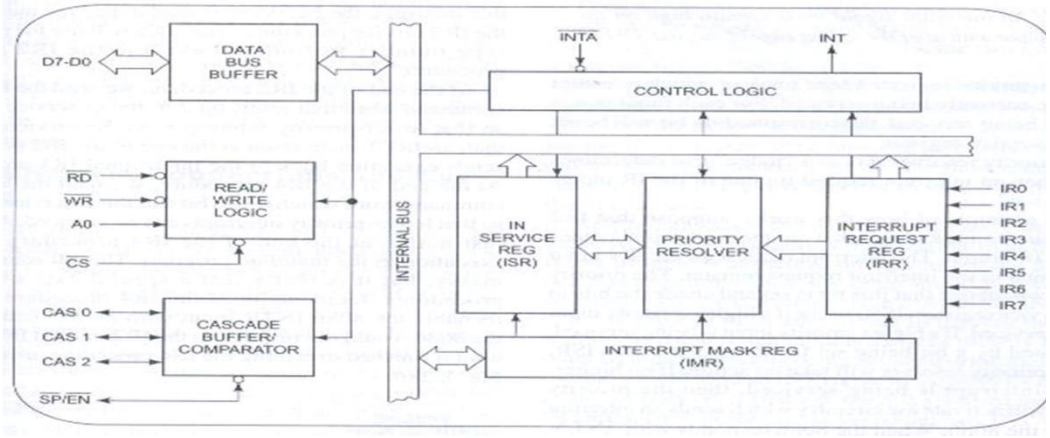
```
DATA SEGMENT
        ARRAY DB 1, 2, 3, 4, 5, 6, 7, 8, 9, 10      ; consider any 10 non -ve numbers
        SUM DB 0                                    ; initialize sum=0
DATA ENDS

CODE SEGMENT
 ASSUME CS:CODE, DS:DATA
 START:MOV AX,DATA       ; data segment initialization
        MOV DS,AX

        LEA SI, ARRAY    ; load offset address of ARRAY into SI
        MOV CX, 10       ; initialize counter=10
   B1: LODSB             ; load memory content pointed by SI into AL & increment SI
       ADD SUM, AL
       DEC CX            ; loop until CX becomes 0
       JNZ B1

       MOV AH,4CH        ; terminate program normally
       INT 21H
CODE ENDS
 END START
```

Unit IV

8 a) **Explain the internal architecture of 8259** 8M 5M

Architectural Diagram of 8259

2M



Explanation

3M

1) When 2 interrupts come at a time to 8259 Interrupt inputs (IR0-IR7), the highest priority will be for IR0 and lowest priority will be for IR7.
2) Priority of Interrupts is based on the 4 blocks in 8259. They are
3) Interrupt Request Register (IRR): Keeps track of which interrupts are asking for service.
4) Interrupt Mask Register (IMR): Disable or enable individual interrupt inputs.
5) In-Service Register (ISR): Keeps track of which interrupts are being serviced.
6) Priority Resolver: Determines when an interrupt request on one of the IR input gets serviced.

8259 Functional Description

7) CS' for enabling the device.
8) IR0-IR7 are Interrupt input pins.
9) WR' for accepting command words from 8086.
10) RD' for releasing status onto the data bus.
11) Cas2-cas0 are used to connect multiple 8259 devices.
12) INT is used to interrupt 8086 when an interrupt is generated in 8259.
13) INTA' is used to enable the interrupt pointer onto the data bus.
14) A0 is connected to 8086 Address line A1.
15) SP'/EN' is used for Master Slave or Buffer Enable.

8 b) **With a neat block diagram explain the working of 8237 DMA controller.** 5M

Block diagram of 8237 2M

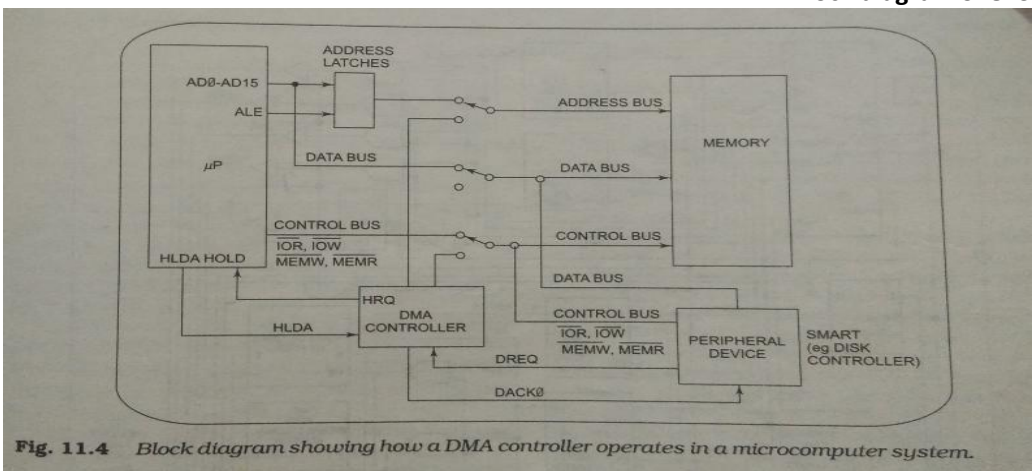


Fig. 11.4 *Block diagram showing how a DMA controller operates in a microcomputer system.*

working of 8237 DMA controller 3M

- When DMA module needs buses it sends HOLD signal to processor
- CPU responds by HLDA (hold acknowledge)

Then DMA module can use buses

- Steps for transfer data from Memory to I/O Device:

1. Device requests service of DMA by pulling DREQ (DMA request) high
2. DMA puts high on HRQ (hold request),
3. CPU finishes present bus cycle (not necessarily present instruction) and puts high on HDLA (hold acknowledge). HOLD remains active for duration of DMA
4. DMA activates DACK (DMA acknowledge), telling device to start transfer
5. DMA starts transfer by putting address of first byte on address bus and activating MEMR;
6. It then activates IOW to write to peripheral. DMA decrements counter and increments address pointer. Repeat until count reaches zero
7. DMA deactivates HRQ, giving bus back to CPU

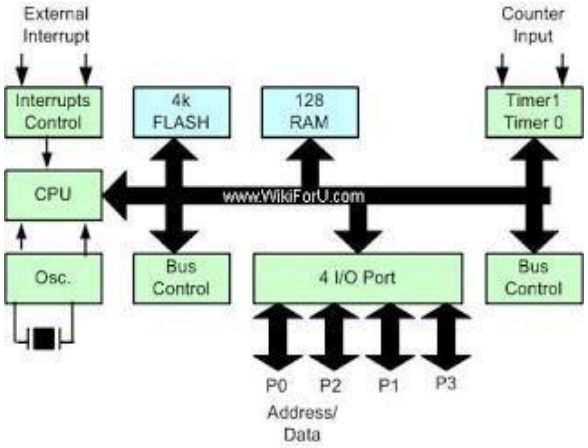
DMA uses MEMW and IOR for I/O device read.

(OR)

9 a) **Draw the architectural diagram of 8051 microcontroller and explain in detail about each block** 6M

Architectural diagram of 8051 3M

Following is the block diagram of Microcontroller 8051. Let us have a look at each part or block of this Architecture:



Microcontroller 8051 - Block Diagram

Explanation: 3M

Central Processor Unit(CPU): As you may know that CPU is the brain of any processing device. It monitors and controls all operations that are performed in the Microcontroller. User have no control over the work of CPU. It reads program written in ROM memory and executes them and do the expected task.

Interrupts: As its name suggests, Interrupt is a subroutine call that interrupts Microcontroller's main operation or work and causes it to execute some another program which is more important at that time.

Memory: Microcontroller requires a program which is a collection of instructions. This program tells Microcontroller to do specific tasks. These programs requires a memory on which these can be saved and read by Microcontroller to perform specific operation. The memory which is used to store the program of Microcontroller, is known as code memory or Program memory . It is known as '**ROM**'(**Read Only Memory**).

Bus: Basically Bus is a collection of wires which work as a communication channel or medium for transfer of Data. These buses consists of 8, 16 or more wires. Thus these can carry 8 bits, 16 bits simultaneously. Buses are of two types:

Address Bus: Microcontroller 8051 has a 16 bit address bus. It used to address memory locations. It is used to transfer the address from CPU to Memory.

Data Bus: Microcontroller 8051 has 8 bits data bus. It is used to carry data.

Oscillator: As we know Microcontroller is a digital circuit device, therefore it requires clock for its operation. For this purpose, Microcontroller 8051 has an on-chip oscillator which works as a clock source for Central Processing Unit. As the output pulses of oscillator are stable therefore it enables synchronized work of all parts of 8051 Microcontroller.

Input/Output Port: As we know that Microcontroller is used in Embedded systems to control the operation of machines. Therefore to connect it to other machines, devices or peripherals we requires I/O interfacing ports in Microcontroller.

Timers/Counters: Microcontroller 8051 has 2 16 bit timers and counters. The counters are divided into 8 bit registers. The timers are used for measurement of intervals , to determine pulse width etc.

9 b) **List and explain the addressing modes of 8051** 5M

List -1M Explanation-5M

1. Immediate addressing mode:

In this type, the operand is specified in the instruction along with the opcode. In simple way, it means data is provided in instruction itself.

Ex: MOV A,#05H -> Where MOV stands for move, # represents immediate data. 05h is the data. It means the immediate date 05h provided in instruction is moved into A register.

2.Register addressing mode:

Here the operand in contained in the specific register of microcontroller. The user must provide the name of register from where the operand/data need to be fetched. The permitted registers are A, R7-R0 of each register bank. Ex: MOV A,R0-> content of R0 register is copied into Accumulator.

3. Direct addressing mode:

In this mode the direct address of memory location is provided in instruction to fetch the operand. Only internal RAM and SFR's address can be used in this type of instruction.

Ex: MOV A, 30H => Content of RAM address 30H is copied into Accumulator.

4. Register Indirect addressing mode:

Here the address of memory location is indirectly provided by a register. The '@' sign indicates that the register holds the address of memory location i.e. fetch the content of memory location whose address is provided in register.

Ex: MOV A,@R0 => Copy the content of memory location whose address is given in R0 register.

5. Indexed Addressing mode:

This addressing mode is basically used for accessing data from look up table. Here the address of memory is indexed i.e. added to form the actual address of memory.

Ex: MOVC A,@A+DPTR => here 'C' means Code. Here the content of A register is added with content of DPTR and the resultant is the address of memory location from where the data is copied to A register.