20CS401

Hall Ticket Number:

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

July/August,2023 — **Computer Science and Engineering**

Fourth Semester — **Microprocessors & Microcontrollers**

Time: Three Hours — **Maximum:** 70 Marks

Answer question 1 compulsory. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56 Marks)**

			CO	BL	M
1	a)	What is pipelining operation in 8086?	CO1	L1	1M
	b)	How does 8086 generate physical address?	CO1	L3	1M
	c)	What is Editor?	CO1	L1	1M
	d)	Define accumulator?	CO1	L2	1M
	e)	Explain MOVSB instruction in 8086.	CO2	L2	1M
	f)	What is Macro and when it is used?	CO2	L2	1M
	g)	Explain REP instruction of 8086.	CO2	L2	1M
	h)	What is the function of EQU directive in 8086?	CO2	L1	1M
	i)	What is the difference between minimum mode and maximum mode configurations of 8086.	CO3	L1	1M
	j)	Define Machine cycle.	CO3	L1	1M
	k)	Give software interrupt instruction of 8086.	CO3	L2	1M
	l)	What is embedded computer system?	CO4	L1	1M
	m)	Write any two single-bit instructions of 8051?	CO4	L2	1M
	n)	What is the difference between the Microprocessor and Microcontroller	CO4	L2	1M
		Unit-I			
2	a)	Explain the register organization of 8086.	CO1	L1	7M
	b)	Discus the writing of 8086 program for use with an assembler.	CO2	L2	7M
		(OR)			
3	a)	Write an 8086 assembly language program to multiply two 16 bit numbers.	CO2	L3	7M
	b)	What are the various instruction sets of 8086? Explain them in detail.	CO1	L2	7M
		Unit-II			
4	a)	Explain various addressing modes of 8086.	CO2	L1	7M
	b)	Explain the following Assembler Directives i) SEGMENT ii) ASSUME iii) PROC iv) DW	CO1	L1	7M
		(OR)			
5	a)	Explain Jump and call instructions of 8086.	CO2	L2	7M
	b)	Build an 8086 assembly language program to transfer a block of 50H data bytes stored in memory location from 4000H onwards to a memory location starting from 5000H using String instructions.	CO2	L3	7M
		Unit-III			
6	a)	Demonstrate the memory write cycle operation of 8086 µp with a neat timing diagram.	CO3	L3	4M
	b)	Draw and explain the pin diagram of 8086 in minimum mode configuration.	CO1	L2	10M
		(OR)			
7	a)	Draw and explain the interrupt structure of 8086.	CO3	L2	7M
	b)	Explain 8259A priority interrupt controller.	CO3	L2	7M
		Unit-IV			
8	a)	Draw and explain the block diagram of 8051 microcontroller.	CO4	L2	10M
	b)	Write about addressing modes of 8051?	CO4	L3	4M
		(OR)			
9	a)	Draw the pin diagram of 8051 and explain about I/O port pins.	CO4	L1	10M
	b)	Write the assembly code for the 8051 microcontroller to toggle the pins of PORT1.	CO4	L5	4M

20CS401

Hall Ticket Number:

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

July/August,2023 — **Computer Science and Engineering**

Fourth Semester — **Microprocessors & Microcontrollers**

Time: Three Hours — **Maximum:** 70 Marks

Answer question 1 compulsory. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56 Marks)**

			CO	BL	M
1	a)	What is pipelining operation in 8086? Pipelining is the feature of fetching the next instruction while executing the current instruction. Instructions are stored in memory, therefore, it has to be fetched, decoded and then executed.	CO1	L1	1M
	b)	How does 8086 generate physical address? To access a specific memory location from any segment we need 20- bit physical address. The 8086 generates this address using the contents of segment register and the offset register associated with it. Physical address=segment address*10H+offset address	CO1	L3	1M
	c)	What is Editor? An Editor is a program which allows / provide environment for the programmer to create,edit, update an ALP program. The Editor stores the ASCII codes for letters and numbers in successive RAM locations.	CO1	L1	1M
	d)	Define accumulator? It consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Multiplication and Division instructions also use the AX or AL.	CO1	L2	1M
	e)	Explain MOVSB instruction in 8086. The MOVSB (move string, byte) instruction fetches the byte at address SI, stores it at address DI and then increments or decrements the SI and DI registers by one.	CO2	L2	1M
	f)	What is Macro and when it is used? A macro is a set of instructions that can be defined once and then used multiple times in a program. A MACRO is group of small instructions that usually performs one task. It is a reusable section of a software program.A macro can be defined anywhere in a program using directive MACRO &ENDM.	CO2	L2	1M
	g)	Explain REP instruction of 8086. REP − Used to repeat the given instruction till CX ≠ 0	CO2	L2	1M
	h)	What is the function of EQU directive in 8086? The EQU directive is used to give name to some value or symbol. Each time the assembler finds the given names in the program, it will replace the name with the value or a symbol.	CO2	L1	1M
	i)	What is the difference between minimum mode and maximum mode configurations of 8086. In **Minimum mode,** 8086 is the only processor in the system and 8086 is operated in minimum mode when MN/MX' pin to logic 1. In **Maximum mode,** we can connect more processors to 8086. 8086 max mode is basically for implementation of allocation of global resources and passing bus control to other co processor (i.e. second processor in the system), because two processors can not access system bus at same instant. 8086 is operated in minimum mode when MN/MX' pin to logic 0.	CO3	L1	1M

j)	Define Machine cycle. Machine cycle refers to a sequence of steps that a computer's central processing unit (CPU) goes through in order to execute a single machine language instruction. (or) a machine cycle is the time required to execute one instruction.	CO3	L1	1M
k)	Give software interrupt instruction of 8086. Software Interrupts – These are instructions inserted within the program to generate interrupts. The instructions are of the format INT type.	CO3	L2	1M
l)	What is embedded computer system? An embedded computer system is a combination of computer hardware and software designed for a specific function.	CO4	L1	1M
m)	Write any two single-bit instructions of 8051? 1. SETB C, 2. CLR C, 3. CPL C or any other	CO4	L2	1M
n)	What is the difference between the Microprocessor and Microcontroller Microprocessor consists of only a Central Processing Unit, whereas Micro Controller contains a CPU, Memory, I/O all integrated into one chip. The microprocessor is useful in Personal Computers whereas Micro Controller is useful in an embedded system	CO4	L2	1M

Unit-I

2	a)	Explain the register organization of 8086.	CO1	L1	7M

(Architecture-2M, Segment Registers-2.5M, General Purpose registers-2.5M)

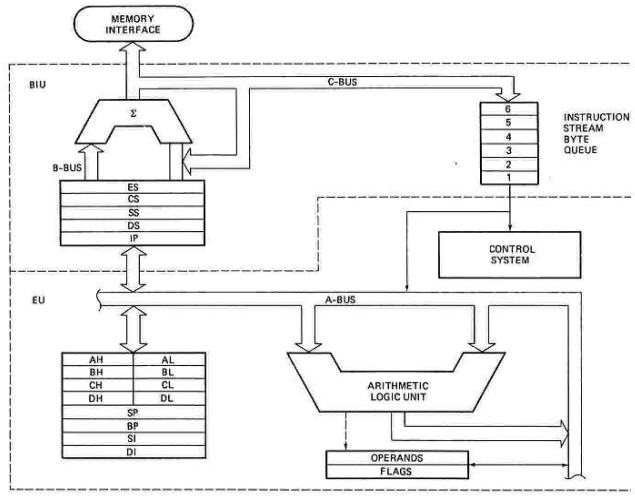


FIGURE 8086 internal block diagram. (*Intel Corp.*)

In 8086, the **BIU** has 4 segment registers. Namely CS,DS,ES,DS registers gives the starting address of the physical segments.

Code Segment Register : 16-bit CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.

BIU computes the 20-bit physical address by logically shifting the contents of CS bits to the left and then adding the 16-bit contents of IP.

That is, all instructions of a program are relative to the contents of the CS register multiplied by 16 and then offset is added provided by the IP.

Data Segment Register: 16-bit register Points to the current data segment; operands for most instructions are fetched from this segment. The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.

Stack Segment Register: 16-bit register Points to the current stack. The 20-bit physical stack address is calculated from the Stack Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP. In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).

Extra Segment Register: 16-bit register Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored. String instructions use the ES and DI to determine the 20-bit physical address for the

destination.

The **EU** has 4 general purpose registers:

Accumulator Register (AX): Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port. Multiplication and Division instructions also use the AX or AL.

Base Register (BX): Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. This is the only general purpose register whose contents can be used for addressing the 8086 memory. All memory references utilizing this register content for addressing use DS as the default segment register.

Counter Register (CX) : Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a counter.

Data Register (DX) : Word multiply, word divide, indirect I/O (Used to hold I/O address during I/O instructions. If the result is more than 16-bits, the lower order 16-bits are stored in accumulator and higher order 16-bits are stored in DX register)

b) Discuss the writing of 8086 program for use with an assembler. CO2 L2 7M
(Assembly level programming-2M, Instruction format:3M Explanation-2M)

Assembly level programming is very important to low-level embedded system design is used to access the processor instructions to manipulate hardware. It is a most primitive machine level language is used to make efficient code that consumes less number of clock cycles and takes less memory as compared to the high-level programming language. It is a complete hardware oriented programming language to write a program the programmer must be aware of embedded hardware. Here, we are providing basics of assembly level programming 8086. type of low-level computer programming language consisting mostly of symbolic equivalents of a particular computer's machine language. Computers produced by different manufacturers have different machine languages and require different assemblers and assembly languages. Used two or three mnemonics to represent each instruction type.

Assembly language program statement format :

Standard form to write statements in assembly language has four fields.

LABEL FIELD	OPCODE FIELD	OPERAND FIELD	COMMENT FIELD
Next:	ADD	AL,07H	;ADD CORRECTION FACTOR

1. A label is symbol or group of symbols used to represent an address which is not specially known at the time the statement is written. Labels are not required in a statement; they are inserted where they are needed.
2. The opcode (operation codes) contains the mnemonic for the instruction to be

performed.
3. Operand field contains the data, the memory address, the port address, or the name of the register on which the instruction is be performed.
4. Comments field starts with a semicolon. They are not part of programs but used to remind the statement purpose.
Assembly language should be converted into binary codes for execution.
Assembler is a software program which converts assembly statements into equivalent binary codes.
A sample Assembly language program:

```
mov  dx, msg      ; the address or message in dx
   mov  ah, 9       ; ah=9 - "print string" sub-function
   int  0x21        ; call dos services

   mov  ah, 0x4c     ; "terminate program" sub-function
   int  0x21        ; call dos services

   msg  db 'Hello, World!',
```

(OR)

3 a) Write an 8086 assembly language program to multiply two 16 bit numbers. CO2 L3 7M

```
            ; 8086 PROGRAM F3-14.ASM
;ABSTRACT  : This program multiplies the two 16-bit words in the memory
            ; locations called MULTIPLICAND and MULTIPLIER. The result
            ; is stored in the memory location, PRODUCT
;REGISTERS : Uses CS, DS, AX, DX
;PORTS     : None used

DATA_HERE    SEGMENT
             MULTIPLICAND  DW 204AH      ; First word here
             MULTIPLIER    DW 3B2AH      ; Second word here
             PRODUCT       DW 2 DUP(0)   ; Result of multiplication here
DATA_HERE    ENDS

CODE_HERE    SEGMENT
             ASSUME    CS:CODE_HERE, DS:DATA_HERE
START:       MOV  AX, DATA_HERE          ; Initialize DS register
             MOV  DS, AX
             MOV  AX, MULTIPLICAND       ; Get one word
             MUL  MULTIPLIER             ; Multiply by second word
             MOV  PRODUCT, AX            ; Store low word of result
             MOV  PRODUCT+2, DX          ; Store high word of result
             INT  3                      ; Wait for command from user
CODE_HERE    ENDS
             END START

;     Programs to be run using a debugger in DOS must include the START: label and the
; START after the END followed by a carriage return. Programs to be downloaded and run need
; only the END directive followed by a carriage return.
```

Assembly language source program to multiply two 16-bit binary numbers to give a 32-bit result.

Scanned with CamScanner

(Or any equivalent code -7M)

b) What are the various instruction sets of 8086? Explain them in detail. CO1 L2 7M
(Explain any two/three instructions from each set-7M)

8086 supports 6 types of instructions

1. Data Transfer Instructions
2. Arithmetic Instructions
3.Logical Instructions
4. String manipulation Instructions
5. Process Control Instructions
6. Control Transfer Instructions

Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

Instruction to transfer a word

MOV − Used to copy the byte or word from the provided source to the provided destination.
PUSH − Used to put a word at the top of the stack.
POP − Used to get a word from the top of the stack to the provided location.
XCHG − Used to exchange the data from two locations.
XLAT − Used to translate a byte in AL using a table in the memory.

Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

Following is the list of instructions under this group −

Instructions to perform addition

ADD − Used to add the provided byte to byte/word to word.
ADC − Used to add with carry.
INC − Used to increment the provided byte/word by 1.
AAA − Used to adjust ASCII after addition.
DAA − Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

SUB − Used to subtract the byte from byte/word from word.
SBB − Used to perform subtraction with borrow
DEC − Used to decrement the provided byte/word by 1.
CMP − Used to compare 2 provided byte/word.
AAS − Used to adjust ASCII codes after subtraction.
DAS − Used to adjust decimal after subtraction.

Instruction to perform multiplication

MUL − Used to multiply unsigned byte by byte/word by word.
IMUL − Used to multiply signed byte by byte/word by word.

Instructions to perform division

DIV − Used to divide the unsigned word by byte or unsigned double word by word.
IDIV − Used to divide the signed word by byte or signed double word by word.

Bit Manipulation Instructions/ Logical Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group −

Instructions to perform logical operation

NOT − Used to invert each bit of a byte or word.
AND − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
OR − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
XOR − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
TEST − Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

SHL/SAL − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
SHR − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.

Instructions to perform rotate operations

ROL − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
ROR − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
RCR − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
RCL − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group −

REP − Used to repeat the given instruction till CX ≠ 0.
REPE/REPZ − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
REPNE/REPNZ − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
MOVS/MOVSB/MOVSW − Used to move the byte/word from one string to another.
COMS/COMPSB/COMPSW − Used to compare two string bytes/words.
LODS/LODSB/LODSW − Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

CALL − Used to call a procedure and save their return address to the stack.
RET − Used to return from the procedure to the main program.
JMP − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions −

JA/JNBE − Used to jump if above/not below/equal instruction satisfies.
JAE/JNB − Used to jump if above/not below instruction satisfies.
JBE/JNA − Used to jump if below/equal/ not above instruction satisfies.
JC − Used to jump if carry flag CF = 1
JE/JZ − Used to jump if equal/zero flag ZF = 1
JG/JNLE − Used to jump if greater/not less than/equal instruction satisfies.
JGE/JNL − Used to jump if greater than/equal/not less than instruction satisfies.
JL/JNGE − Used to jump if less than/not greater than/equal instruction satisfies.
JLE/JNG − Used to jump if less than/equal/if not greater than instruction satisfies.
JNC − Used to jump if no carry flag (CF = 0)
JNE/JNZ − Used to jump if not equal/zero flag ZF = 0
JNO − Used to jump if no overflow flag OF = 0
JNP/JPO − Used to jump if not parity/parity odd PF = 0
JNS − Used to jump if not sign SF = 0
JO − Used to jump if overflow flag OF = 1

JP/JPE − Used to jump if parity/parity even PF = 1
JS − Used to jump if sign flag SF = 1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group −

STC − Used to set carry flag CF to 1
CLC − Used to clear/reset carry flag CF to 0
CMC − Used to put complement at the state of carry flag CF.
STD − Used to set the direction flag DF to 1
CLD − Used to clear/reset the direction flag DF to 0
STI − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
CLR − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Unit-II

4 a) Explain various addressing modes of 8086. CO2 L1 7M
(Any 7 addressing modes-7M)

Every instruction of a program has to operate on a data. The different ways in which a source operand is denoted in an instruction are known as addressing modes.

Immediate Addressing Mode
The operands are specified within the instructions in this type of mode.
The first operand is never an immediate value. Instead, it contains a destination field that specifies the length of the data and can be either a register or a memory address.
MOV AL, 28H ; It moves the 8-bit data 28H into AL.

Register Addressing Mode
The data is referred to using the specific register in which it is stored. Both operands in this particular addressing mode are registers.
MOV AX, BX

MOV CL, DL

Direct Addressing Mode
The instruction specifies the effective address as displacement when using this sort of addressing mechanism.
This indicates that the value is immediately fixed and immutably entered into the instruction.
MOV AX, [5000H] ;The square brackets around the 5000H imply the contents of the memory location. When it is executed, this instruction will copy the contents of the memory location into the AX register.

Register Indirect Addressing Mode
In this Addressing Mode, the offset registers indirectly derive the address of the memory region containing the data or operand.
The base register, source index, or destination index (BX, SI, or DI register, respectively) contains the offset address of the data. Either DS or ES are the default segments.

MOV BX, [AX] Suppose the register AX holds 4895H, then its data, 4895H, is moved to BX.

Based Addressing Mode
Based Addressing specifies a signed 8-bit or an unsigned 16-bit displacement and

stores the base value for the effective address in BX or BP.
When there is an 8-bit displacement, the sign is 16-bit extended before being added to the base value. When BX contains the base value of EA, DS, and BX are combined to form a 20-bit physical address. BP and SS are used when BP holds the base value of EA.
MOV AX, [BX + 04] ; Given instruction indicates that there is a value stored in the BX register and this is added with 4 bits displacement value and will store in the AX register.

Indexed Mode
The SI or DI register stores the index value for memory data, and the instruction will specify either a signed 8-bit or an unsigned 16-bit displacement.
To produce the EA, displacement is added to the index value in the SI or DI register. When there is an 8-bit displacement, the sign is 16-bit extended before being added to the base value.
MOV CX, [SI + 16] ;Given instruction indicates that there is a value stored in the SI (Source Index) register and this is added with 16 bits displacement value and will store in the BX register.

Based Indexed Mode
In the based index addressing of 8086, a base register (BX or BP), an index register (SI or DI), and a displacement are added to determine the effective address.
MOV DX, [BX + SI] ;Given instruction indicates that there is a value stored in the SI (Source Index) register and this is added to the value of the BX register and will store in the AX register.

Based Indexed Displacement Mode
In this addressing mode, the operand offset is calculated by summing the content of the base register. The index records the content and an 8- or 16-bit offset.
MOV AX, [BX+DI+08] ;Given instruction indicates that there is a value stored in the SI (Source Index) register and this is added to the value of the BX register along with the 8-bit displacement value and will store in the AX register.

String Addressing Mode
Used to manipulate string data when performing string operations.
Effective addresses (EAs) for source and destination data are maintained in SI and DI registers. DS and ES are the segment registers used to determine the base addresses of the source and destination data, respectively.

MOVSB
REP MOVSB

Input/Output Mode
Accessing data from commonly used I/O mapped devices or ports is done using these addressing modes.
Examples of Input/Output modes are given below.
IN AL, [09H]
PORTadder = 09H
(AL) ← (PORT)

The content of the port with address 09_H is moved to the AL register

Relative Addressing
An 8-bit signed displacement is used in this addressing mode to specify the effective address of a program instruction in relation to the Instruction Pointer (IP).
Examples of Relative Addressing Modes are given below.
JNC START

If CY=O, then PC is assigned with current PC contents plus 8 bit signed value of START,
otherwise, the next instruction is executed.

Implied Addressing Mode
This kind of mode's instructions has no operands, and the data that the instruction will act on will be given explicitly in the instruction.

CLC

This clears the carry flag to zero.

b) Explain the following Assembler Directives CO1 L1 7M

i) SEGMENT **(2M)** ii) ASSUME **(2M)** iii) PROC **(2M)** iv) DW **(1M)**

SEGMENT : The SEGMENT and ENDS directives are used to identify a group of data items or a group of instructions that put together in a particular segment. That segment is called logical segment.
<<name to logical segment>> SEGMENT

<<name to logical segment>> ENDS

To specify the name for logical segment :Can not use spaces to separate words but underscore. Can not use instruction mnemonics as segment names or labels.

ii) **ASSUME**: Informs the assembler the name of the program/ data segment that should be used for a specific segment
Ex: ASSUME CS:CODE, DS:DATA ;Tells the assembler that the instructions of the program are stored in the segment CODE and data are stored in the segment DATA

iii) **PROC**: Indicates the beginning of a procedure

<<name of procedure>> PROC

--- Body of procedure

<<name of procedure>> ENDP

iv) **DW** : Define Word used to initialize 16-bit value
Ex: value1 DW 5089H

(OR)

5 a) Explain Jump and call instructions of 8086. CO2 L2 7M

(JUMP INSTRUCTION- 3.5M CALL instruction-3.5M)

A **CALL instruction** is utilized to call a sub-routine. Using a CALL instruction, the program control is transferred to a location in memory that is not a part of the main program. A CALL instruction necessarily requires the initialization of a Stack Pointer (SP). It is basically a control transfer type instruction because it is used to invoke the subroutine. Once all the CALL instructions called the subroutine and the execution of these subroutine is completed, then the program control is transferred back to the caller by using the RET instruction. . There are two basic types of CALLs, 1.near and 2.far.

A near CALL is a call to a procedure which is in the same code segment as the CALL instruction. When the 8086 executes a near CALL instruction it decrements the stack pointer by two and copies the offset of the next instruction after the CALL on the stack. It loads IP with the offset of the first instruction of the procedure in same segment. The near CALL is also known as intra segment CALL

A far CALL is a call to a procedure which is in a different segment from that which contains the CALL instruction. When the 8086 executes a far CALL it decrements the stack pointer by two and copies the contents of the CS register to the stack. It then decrements the stack pointer by two again and copies the offset of the instruction ,after the CALL to the stack. Finally, it loads CS with the segment base of the segment which contains the procedure and IP with the offset of the first instruction of the procedure in that segment. The far CALL is also known as inter segment CALL.

Jump Instruction: There are 2 types of jump instructions.

1. Unconditional Jump (JMP)
2. Conditional Jump

Unconditional jump /JMP Instruction :

This instruction will always cause the Program Execution Transfer Instructions in 8086 Microprocessor to fetch its next instruction from the location specified in the instruction rather than from the next location after the JMP instruction.

There are two basic types of JMPs, near and far. A near JMP is a jump where destination location is in the same code segment. In this case only IP is changed. A near JMP is known as intrasegment JMP. A far JMP is a jump where destination location is from a different segment. In this case both IP and CS are changed as specified in the destination. A far IMP is known as Inter segment JMP

J cond – Conditional Transfer Instructions:

These instructions will cause a jump to a label given in the instruction if the desired condition(s) occurs in the program before the execution of the instruction. The destination must be in the range of – 128 bytes to + 127 bytes from the address of the instruction after the conditional transfer instruction. If the jump is not taken, Program Execution Transfer Instructions in 8086 Microprocessor simply goes on to the next instruction.

(Refer Q.No 3b. Conditional Transfer Instructions from program execution transfer instructions)

b) Build an 8086 assembly language program to transfer a block of 50H data bytes stored in memory location from 4000H onwards to a memory location starting from 5000H using String instructions. CO2 7M

Mnemonics	**Comments**
MOV SI, 4000H	SI<-4000h
MOV DI,5000H	DI<-5000h
MOV AX, 0000	AX<-0000
MOV DS,AX	DS<-AX
MOV ES,AX	ES<-AX
MOV CX,50H	CX<-50H
Cld	clears the directional flag
rep movsb	repeat until CX is not equal to zero and CX=CX-1 at every step transfer the data from source to destination memory location

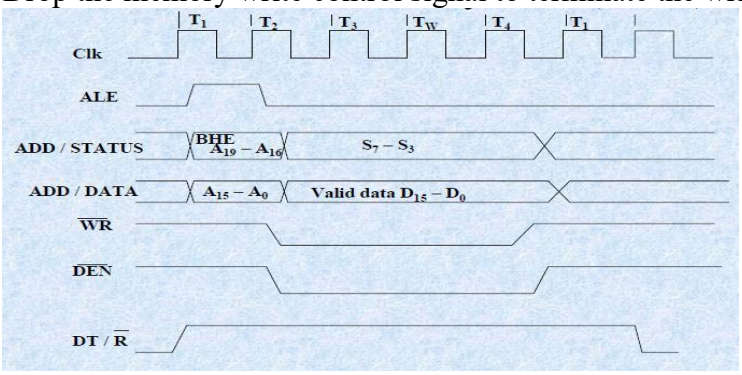
int 21h	Interrupt the program execution	
END	end of the program	

(OR) Any Equivalent Code -7M

Unit-III

6 a) Demonstrate the memory write cycle operation of 8086 µp with a neat timing diagram. **(Diagram: 2M, Explanation:2M)** CO3 L3 4M

The following steps have to be followed in a typical write cycle:
Place the address of the location to be written on the address bus.
Place the data to be written on the data bus.
Activate the memory write control signal on the control bus.
Wait for the memory to store the data at the address location..
Drop the memory write control signal to terminate the write cycle



b) Draw and explain the pin diagram of 8086 in minimum mode configuration. **(Diagram: 3M, Explanation:7M)** CO1 L2 10M

Pin Diagram and Description of 8086 Microprocessor

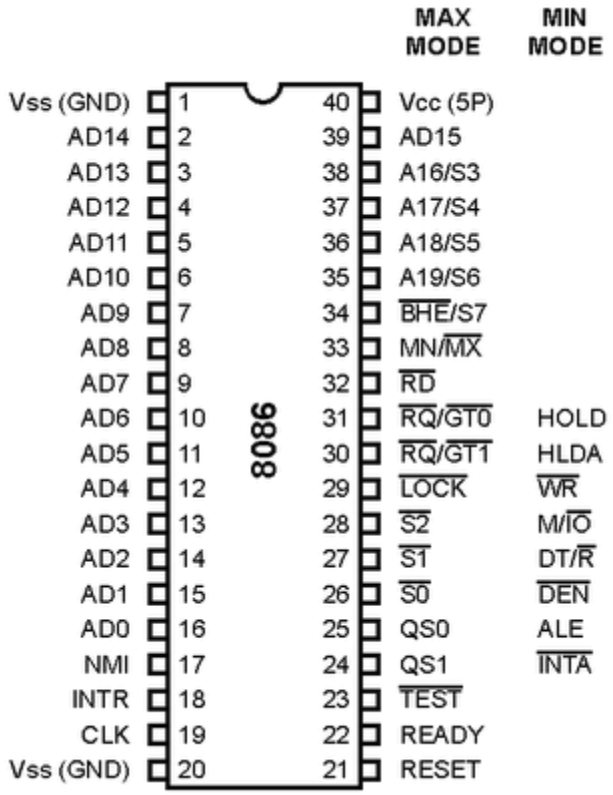
Definition: 8086 is a 16-bit microprocessor and was created by Intel in 1978. Like the pin configuration of 8085 microprocessor, the 8086 microprocessor also contains 40 pins dual in line. However, unlike the 8085 microprocessor, an 8086 to have better performance, operates in 2 modes that are minimum and maximum mode.

The minimum mode is a single processor configuration while the maximum mode is a multiple processor configuration. Due to this reason, in the 40 pin IC of 8086 microprocessor, 8 pins i.e., pin numbered from 24 to 32 are assigned different configurations separately according to the two modes.

Pin Diagram 8086 Microprocessor

An 8086 microprocessor is also a 40 pin IC but has few separate pin configuration for minimum and maximum mode

The figure below represents the pin diagram of 8086 microprocessor:



Here, from the above figure it is clear that from pin number 24 to 31, we have shown the different configuration for minimum and maximum mode. However, excluding these 8 pins, the rest 32 pins are the same for both minimum as well as maximum mode.

So, let us move further to understand the operation of each pin in the pin configuration of 8086.

Pin description of 8086 Microprocessor

V_{CC} – ***Pin number 40*** – At this pin, the external power supply of **+ 5V** is provided to the processor.

V_{SS} – ***Pin number 1 and 20*** – These two pins acts as the ground. This pin directs the extra current of the microprocessor to ground.

AD_0 – AD_{15} – ***Pin number 2 to 16 and 39*** – These are the multiplexed address and data bus.
We know that the 8086 microprocessor has 20-bit address bus and 16-bit data bus. So, the 16 lines of the address and data bus are multiplexed together so as to reduce the number of lines inside the IC.

We are aware of the fact that at a time either address or data will be transmitted by the bus. So, at a particular time only either the address or the data bus will be enabled from the multiplexed buses.

A_{16}/S_3, A_{17}/S_4, A_{18}/S_5 and $A_{19}S_6$ – ***Pin number 35 to 38*** – Out of 20 address bits, 4 are present in the multiplexed form with the status signals. In the case of memory operations, these pins act as an address bus and contain the memory address of any particular instruction or data.
However, from I/O operations these pins are low that shows the status of the processor.

Basically, the signal at S_3 and S_4 show that which segment is currently accessed by the microprocessor among the four segments present in it.

The table below will show the encoding of S_3 and S_4:

S_3	S_4	STATUS
0	0	ES
0	1	SS
1	0	CS or idle
1	1	DS

Table 1

Also, S_5, when enabled, shows the presence of an interrupts in the microprocessor. So, basically, it serves as an **interrupt flag**.
The signal at S_6 shows the status of the bus master for the current operation. More simply we can say, whether the 8086 is the bus master or any other proficient device is acting as the bus master.
When 0 is present as the signal at this pin then it indicates the 8086 is holding the access of the bus otherwise it is high i.e., 1.

BHE' / S_7 – ***Pin number 34*** – BHE is an acronym for Bus High Enable. The combination of the BHE signal and S_7 status informs about the existence of the data on the bus. Also, different combinations show whether the bus is containing overall 16 bit, upper byte or lower byte of the data.

MN/MX' – ***Pin number 33*** –The status at this particular pin shows whether the processor is operating in the minimum mode or maximum mode.
A signal 0 at this pin informs that the 8086 is operating in maximum mode i.e., multiple processors. While signal 1 shows the operation under minimum mode i.e., single processor.

RD' – ***Pin number 32*** – An active low signal at this pin shows that the microprocessor is performing read operation with either memory or I/O devices.

CLK – ***Pin number 19*** – A signal at this pin provides the timing to the internal operations that are being executed inside the microprocessor.

NMI – ***Pin number 17*** – NMI is Non-maskable interrupt. These are basically uncontrollable interrupts generated inside the processor. When an NMI occurs, then an interrupt service routine is generated by the interrupt vector table. An edge triggered input, causes a type-2 interrupt. NMI is not maskable internally by software.

TEST – ***Pin number 23*** – This pin basically shows the wait instruction. Whenever a low signal at this pin occurs then the processing inside the processor continues. As against, in case of the high signal, the processor has to wait for the disabling of this pin.

INTR – ***Pin number 18*** – INTR stands for an interrupt request. The processor after each clock cycle samples the INTR and if the signal at this pin is found to be high then the processor controls that interrupt internally.

READY – ***Pin number 22*** – This signal is used by the peripherals and memory devices in order to show the readiness for the next operation.

RESET – ***Pin number 21*** – Whenever this pin is enabled then it resets the processor and other devices connected to the system by immediately terminating the recent task. Reset causes the processor to immediately terminate its present activity. To be recognized, the signal must be active high for at least four clock

cycles. It causes the 8086 to initialize registers DS, SS, ES, IP and flags to all zeros. It also initializes CS to FFFF H. Upon removal of the RESET signal from the RESET pin, the 8086 will fetch its next instruction from the 20 bit physical address FFFF0H.

Pins in Minimum mode

INTA' – ***Pin number 24*** – It is an interrupt acknowledge pin. Whenever an INTR signal is generated, then the microprocessor generates INTA signal, as a response to that interrupt.

ALE – ***Pin number 25*** – ALE is an abbreviation for address latch enable. Whenever an address is present in the multiplexed address and data bus, then the microprocessor enables this pin.
This is done to inform the peripherals and memory devices about fetching of the data or instruction at that memory location.

DEN' – ***Pin number 26*** – DEN is used for data enable. This is an active low pin that means whenever a 0 is present at this pin then the transceiver gets enabled and it separates the data from the multiplexed address and data bus.

DT/R' – ***Pin number 27*** – This pin is used to show whether the data is getting transmitted or is received. A high signal at this pin indicates that data is being transmitted. While a low indicates reception of data.

M/IO' – ***Pin number 28*** – This pin indicates whether the processor is performing an operation with memory or I/O devices. Whenever a high is present at this pin then it shows the operation is carried out through the memory. While a low signal shows operation through I/O devices.
WR' – ***Pin number 29*** – An active low signal at this pin indicates that the processor is performing write operation from either memory or I/O devices.

HOLD – ***Pin number 31*** – When an external device enables this pin then the processor stops accessing the buses immediately after the recent task gets over.

HLDA – ***Pin number 30*** – This pin is used as a response pin for the hold request. Once request for accessing the buses is produced by an external entity. Then the microprocessor acknowledges the device that its request will be considered once it gets over by the current operation.

(OR)

7 a) Draw and explain the interrupt structure of 8086. CO3 L2 7M

(Definition of Interrupt-1M, Interrupt Sources-2M, response-2M, Types-2M)

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

An 8086 interrupt can come from any one the three sources:

1. Exernal signal: An 8086 can get interrupt from an external signal applied to the nonmaskable interrut (NMI) input pin, or the interrupt (INTR) input pin.

2. Special instruction: An execution of the Interrupt instruction (INT). This is referred as software interrupt.

3. Condition produced by Instruciton: An 8086 is interrupted by some condition produced in the program by the execution of an instruction.

For example divide by zero: program execution will automatically be interrupted if you attempt to divided an operand by zero.

At the end of each instruction cycle, 8086 checks to see if any interrupts have been requested. If an interrupt has been requested, **the 8086 responds** to interrupt by stepping through the following series of major steps:

1. It decrements the stack pointer by 2 pushes the flag register on the stack.

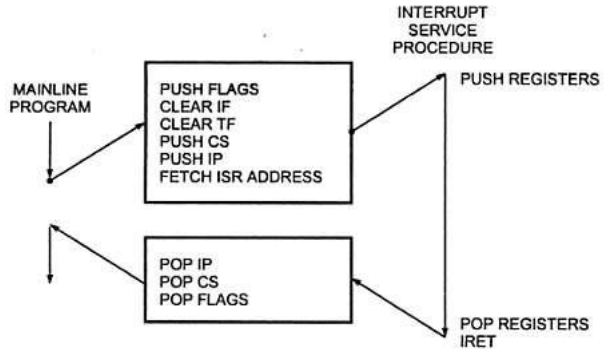
2. It disables the 8086 INTR interrupt input by clearing the interrupt flag(IF) in the flag register.

3. It resets the trap flag (TF) in the flag register.

4. It decrements the stack pointer by 2 and pushes the current code segment register contents on the stack.

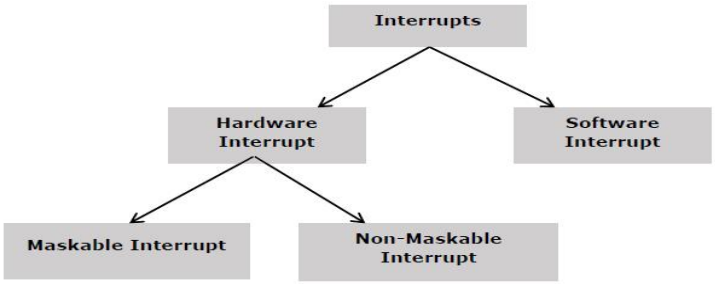
5. It decrements the stack pointer again by 2 and pushes the current instruction pointer contents on the stack.

6. It does an indirect far jump to start of the procedure by loading the CS and IP values for the start of the interrupt service routine.



8086 interrupt response

INTERRUPT TYPES:



Hardware Interrupts

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

INTR

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be

enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends '0' on INTA pin twice. The first '0' means INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

Software Interrupts

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes −

INT- Interrupt instruction with type number

It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

TYPE 0 interrupt represents division by zero situation.

TYPE 1 interrupt represents single-step execution during the debugging of a program.

TYPE 2 interrupt represents non-maskable NMI interrupt.

TYPE 3 interrupt represents break-point interrupt.

TYPE 4 interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

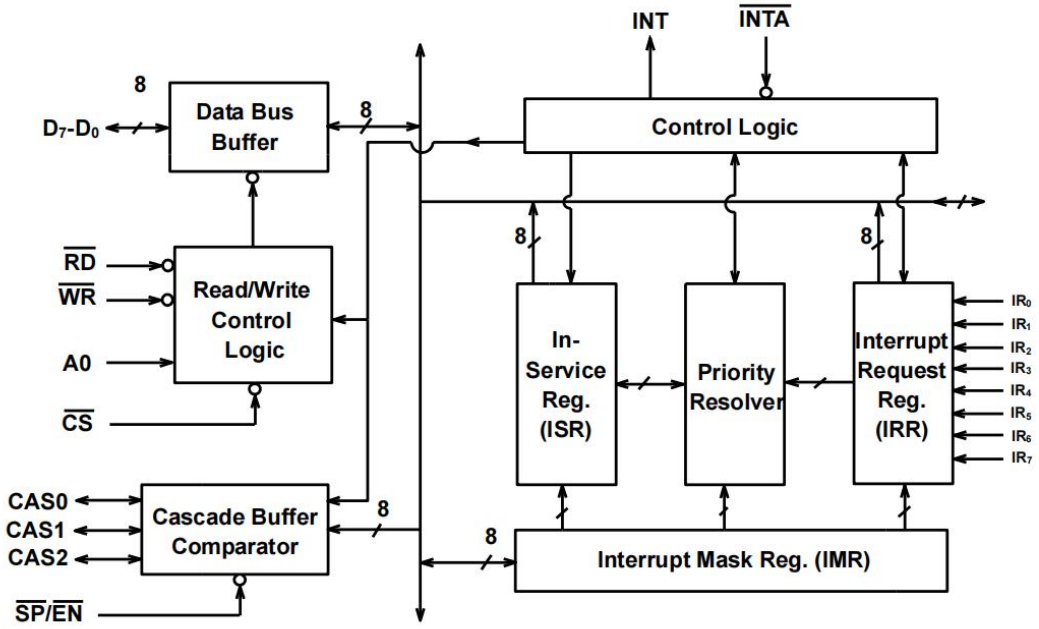
b) Explain 8259A priority interrupt controller. **(Diagram: 2M, Explanation:5M)** CO3 L2 7M

The 8259A (PIC) has eight interrupt request inputs – IR7 - IR0.

The 8259A uses its INT output to interrupt the 8085A via INTR pin. The 8259A receives interrupt acknowledge pulses from the at its INTA input. Vector address, used by the 8085A to transfer control to the service subroutine of the interrupting device, is provided by the 8259A on the data bus. The 8259A is a programmable device that must be initialized by command words sent by the microprocessor. After initialization the 8259A mode of operation can be changed by operation command words from the microprocessor.

Data bus buffer:

This 3- state, bidirectional 8-bit buffer is used to interface the 8259A to the system data bus. Control words and status information from the microprocessor to PIC and from PIC to microprocessor respectively, are transferred through the data bus buffer.



IR7Read/Write & Control Logic: The function of this block is to accept output commands sent from the CPU. It contains the initialization command word (ICW) registers and operation command word (OCW) registers which store the various control formats for device operation. This function block also allows the status of 8259A to be transferred to the data bus.

Interrupt Request Register (IRR): Interrupt request register (IRR) stores all the interrupt inputs that are requesting service. It is an 8-bit register – one bit for each interrupt request. Basically, it keeps track of which interrupt inputs are asking for service. If an interrupt input is unmasked, and has an interrupt signal on it, then the corresponding bit in the IRR will be set. The content of this register can be read to know the status of pending interrupts.

Interrupt Mask Register (IMR): The IMR is used to disable (Mask) or enable (Unmask) individual interrupt request inputs. This is also an 8-bit register. Each bit in this register corresponds to the interrupt input with the same number. The IMR operates on the IRR. Masking of higher priority input will not affect the interrupt request lines of lower priority. To unmask any interrupt the corresponding bit is set '0'.

In-service Register (ISR): The in-service register keeps track of which interrupt inputs are currently being serviced. For each input that is currently being serviced the corresponding bit of in-service register (ISR) will be set. In 8259A, during the service of an interrupt request, if another higher priority interrupt becomes active, it will be acknowledged and the control will be transferred from lower priority interrupt service subroutine (ISS) to higher priority ISS. Thus, more than one bit of ISR will be set indicating the number of interrupts being serviced. Each of these 3-registers can be read as status register.

Priority Resolver: This logic block determines the priorities of the interrupts set in the IRR. It takes the information from IRR, IMR and ISR to determine whether the new interrupt request is having highest priority or not. If the new interrupt request is having the highest priority, it is selected and processed. The corresponding bit of ISR will be set during interrupt acknowledge machine cycle.

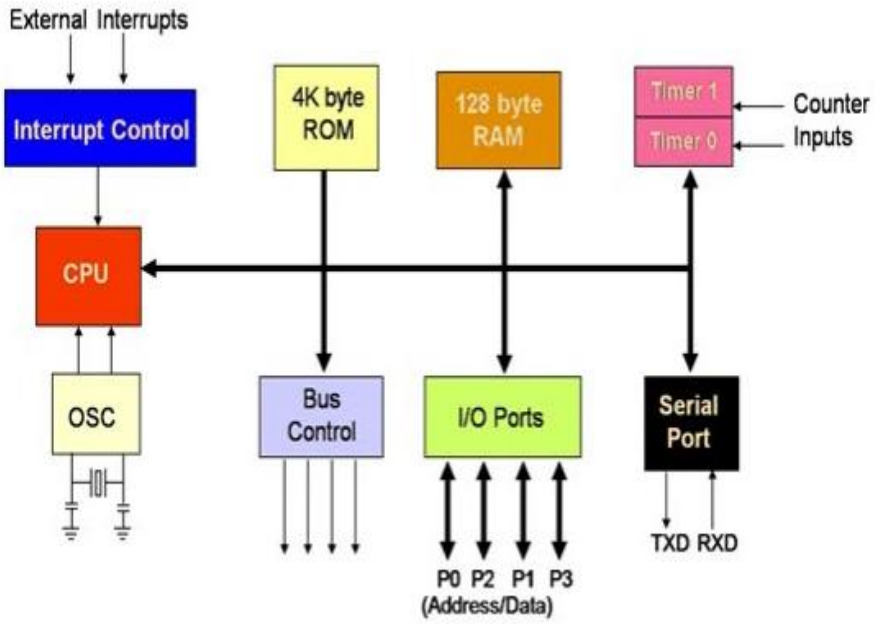
Cascade Buffer/Comparator: This function block stores and

compares the IDs of all 8259A's in the system. The associated 3-I/O lines (CAS2-CAS0) are outputs when 8259A is used as a master and are inputs when 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS2-0 lines. The slave 8259As compare this ID with their own programmed ID. Thus selected 8259A will send its pre-programmed subroutine address on to the data bus during the next one or two successive INTA' pulses.

Unit-IV

8 a) Draw and explain the block diagram of 8051 micro controller. CO4 L2 10M
(Diagram: 3M, Explanation:7M)

The internal architecture of 8051 Micro controller represented in form of block diagram as shown below:



Basic components present internally inside 8051 Microcontroller architecture are:

CPU (Central Processing Unit): CPU act as a mind of any processing machine. It synchronizes and manages all processes that are carried out in microcontroller. User has no power to control the functioning of CPU. It interprets the program stored in ROM and carries out from storage and then performs it projected duty. CPU manage the different types of registers available in 8051 microcontroller.

Interrupts: Interrupts is a sub-routine call that given by the microcontroller when some other program with high priority is request for acquiring the system buses the n interrupts occur in current running program.

Interrupts provide a method to postpone or delay the current process, performs a sub-routine task and then restart the standard program again.

Types of interrupt in 8051 Microcontroller:

Let's see the five sources of interrupts in 8051 Microcontroller:

- Timer 0 overflow interrupt - TF0
- Timer 1 overflow interrupt - TF1
- External hardware interrupt - INT0
- External hardware interrupt - INT1

- Serial communication interrupt - RI/TI

Memory: For operation Micro-controller required a program. This program guides the microcontroller to perform the specific tasks. This program installed in microcontroller required some on chip memory for the storage of the program.

Microcontroller also required memory for storage of data and operands for the short duration. In microcontroller 8051 there is code or program memory of 4 KB that is it has 4 KB ROM and it also comprise of data memory (RAM) of 128 bytes.

Bus : Bus is a group of wires which uses as a communication canal or acts as means of data transfer. The different bus configuration includes 8, 16 or more cables. Therefore, a bus can bear 8 bits, 16 bits all together.

- **Address Bus**: 8051 microcontrollers is consisting of 16 bit address bus. It is generally be used for transferring the data from Central Processing Unit to Memory.
- **Data bus**: 8051 microcontroller is consisting of 8 bits data bus. It is generally be used for transferring the data from one peripherals position to other peripherals.

Oscillator: As the microcontroller is digital circuit therefore it needs timer for their operation. To perform timer operation inside microcontroller it required externally connected or on-chip oscillator. Microcontroller is used inside an embedded system for managing the function of devices. Therefore, 8051 uses the two 16 bit counters and timers. For the operation of this timers and counters the oscillator is used inside microcontroller.

b) Write about addressing modes of 8051? CO4 L3 4M

(Any 4 addressing modes-4M)

The addressing modes of 8051 are:

Immediate AddressingMode
Register AddressingMod
Direct AddressingMode
Register IndirectAddressing Mode
Indexed AddressingMode
Implied AddressingMode

Immediate addressing mode

In this Immediate Addressing Mode, the data is provided in the instruction itself. The data is provided immediately after the opcode. These are some examples of Immediate Addressing Mode.

```
MOVA, #0AFH;
MOVR3, #45H;
MOVDPTR, #FE00H;
```

In these instructions, the # symbol is used for immediate data. In the last instruction, there is DPTR. The DPTR stands for Data Pointer. Using this, it points the external data memory location. In the first instruction, the immediate data is AFH, but one 0 is added at the beginning. So when the data is starting with A to F, the data should be preceded by 0.

Register addressing mode

In the register addressing mode the source or destination data should be present in a register (R0 to R7). These are some examples of RegisterAddressing Mode.

```
MOVA, R5;
```

```
MOVR2, #45H;
MOVR0, A;
```

In 8051, there is no instruction like **MOVR5, R7**. But we can get the same result by using this instruction **MOV R5, 07H**, or by using **MOV 05H, R7**.

Direct Addressing Mode

In the Direct Addressing Mode, the source or destination address is specified by using 8-bit data in the instruction. Only the internal data memory can be used in this mode. Here some of the examples of direct Addressing Mode.

```
MOV80H, R6;
MOVR2, 45H;
MOVR0, 05H;
```

The first instruction will send the content of registerR6 to port P0 (Address of Port 0 is 80H). The second one is forgetting content from 45H to R2. The third one is used to get data from Register R5 (When register bank RB0 is selected) to register R5.

Register indirect addressing Mode

In this mode, the source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes. Let us see some examples of this mode.

```
MOV0E5H, @R0;
MOV@R1, 80H
```

In the instructions, the @ symbol is used for register indirect addressing. In the first instruction, it is showing that theR0 register is used. If the content of R0 is 40H, then that instruction will take the data which is located at location 40H of the internal RAM. In the second one, if the content of R1 is 30H, then it indicates that the content of port P0 will be stored at location 30H in the internal RAM.

```
MOVXA, @R1;
MOV@DPTR, A;
```

In these two instructions, the X in MOVX indicates the external data memory. The external data memory can only be accessed in register indirect mode. In the first instruction if the R0 is holding 40H, then A will get the content of external RAM location40H. And in the second one, the content of A is overwritten in the location pointed by DPTR.

Indexed addressing mode

In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A. These are some examples of Indexed addressing mode.

```
MOVCA, @A+PC;
MOVCA, @A+DPTR;
```

The C in MOVC instruction refers to code byte. For the first instruction, let us consider A holds 30H. And the PC value is1125H. The contents of program memory location 1155H (30H + 1125H) are moved to register A.

Implied Addressing Mode

In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only. These types of instructions are also known as register specific instruction. Here are some examples of Implied Addressing Mode.

```
RLA;
SWAPA;
```

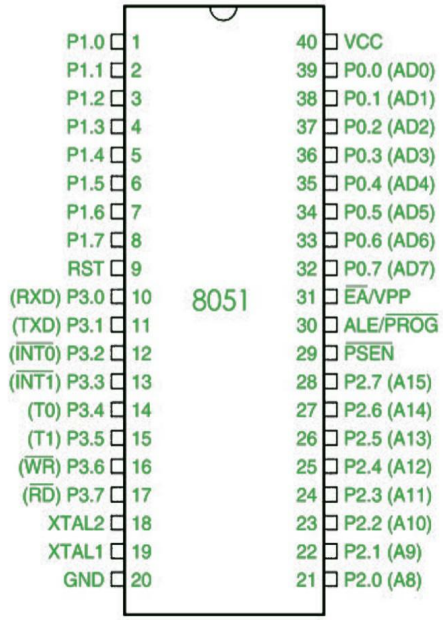
These are 1- byte instruction. The first one is used to rotate the A register content to the Left. The second one is used to swap the nibbles in A.

(OR)

9 a) Draw the pin diagram of 8051 and explain about I/O port pins. CO4 L1 10M

(Diagram: 3M, Explanation:7M)

8051 microcontroller is a 40 pin Dual Inline Package (DIP). These 40 pins serve different functions like read, write, I/O operations, interrupts etc. 8051 has four I/O ports wherein each port has 8 pins which can be configured as input or output depending upon the logic state of the pins. Therefore, 32 out of these 40 pins are dedicated to I/O ports. The rest of the pins are dedicated to VCC, GND, XTAL1, XTAL2, RST, ALE, EA' and PSEN'. Pin diagram of 8051 microprocessor is



40 - PIN DIP

as

Description of the Pins :

Pin 1 to Pin 8 (Port 1) – Pin 1 to Pin 8 are assigned to Port 1 for simple I/O operations. They can be configured as input or output pins depending on the logic control i.e. if logic zero (0) is applied to the I/O port it will act as an output pin and if logic one (1) is applied the pin will act as an input pin. These pins are also referred to as P1.0 to P1.7 (where P1 indicates that it is a pin in port 1 and the number after '.' tells the pin number i.e. 0 indicates first pin of the port. So, P1.0 means first pin of port 1, P1.1 means second pin of the port 1 and so on). These pins are bidirectional pins.

Pin 10 to Pin 17 (Port 3) – Pin 10 to pin 17 are port 3 pins which are also referred to as P3.0 to P3.7. These pins are similar to port 1 and can be used as universal input or output pins. These pins are bidirectional pins. These pins also have some additional functions which are as follows:

P3.0 (RXD) : 10th pin is RXD (serial data receive pin) which is for serial input. Through this input signal microcontroller receives data for serial communication.

P3.1 (TXD) : 11th pin is TXD (serial data transmit pin) which is serial output pin. Through this output signal microcontroller transmits data for serial communication.

P3.2 and P3.3 (INT0′, INT1′) : 12th and 13th pins are for External Hardware Interrupt 0 and Interrupt 1 respectively. When this interrupt is activated(i.e. when it is low), 8051 gets interrupted in whatever it is doing and jumps to the vector value of the interrupt (0003H for INT0 and 0013H for INT1) and starts

performing Interrupt Service Routine (ISR) from that vector location.

P3.4 and P3.5 (T0 and T1) : 14th and 15th pin are for Timer 0 and Timer 1 external input. They can be connected with 16 bit timer/counter.

P3.6 (WR') : 16th pin is for external memory write i.e. writing data to the external memory.

P3.7 (RD') : 17th pin is for external memory read i.e. reading data from external memory.

Pin 21 to Pin 28 (Port 2) – Pin 21 to pin 28 are port 2 pins also referred to as P2.0 to P2.7. When additional external memory is interfaced with the 8051 microcontroller, pins of port 2 act as higher-order address bytes. These pins are bidirectional.

Pin 32 to Pin 39 (Port 0) – Pin 32 to pin 39 are port 0 pins also referred to as P0.0 to P0.7. They are bidirectional input/output pins. They don't have any internal pull-ups. Hence, 10 K? pull-up registers are used as external pull-ups. Port 0 is also designated as AD0-AD7 because 8051 multiplexes address and data through port 0 to save pins.

b) Write the assembly code for the 8051 microcontroller to toggle the pins of PORT1. CO4 L5 4M

Port 1 can be used as input or output. Upon reset, port 1 is configured as an input port

```
;Toggle all bits of P1 continuously
          MOV   A,#55H
BACK:     MOV   P1,A
          ACALL DELAY
          CPL   A        ;complement(Invert) reg. A
          SJMP  BACK
```

(OR any equivalent code)

Scheme Prepared by: M.Karuna
Asst. Professor,
CSE Dept.

Dr. M. Rajesh Babu
HOD, CSE Dept.

Internal Faculty:

Faculty1:

Faculty2:

External Faculty: