MICROPROCESSORS AND MICROCONTROLLERS
EC8691

2 MARKS & 13 MARKS

B.E –CSE
(III YEAR – 5th SEM)
(2021)

Prepared by:
Ms. E.SARANYA, Assistant Professor

Department of Electronics and Communication Engineering

ARUNAI ENGINEERING COLLEGE

EC8691 MICROPROCESSORS AND MICROCONTROLLERS **L T P C**
3 0 0 3

OBJECTIVES:

- To understand the Architecture of 8086 microprocessor.
- To learn the design aspects of I/O and Memory Interfacing circuits.
- To interface microprocessors with supporting chips.
- To study the Architecture of 8051 microcontroller.
- To design a microcontroller based system

UNIT I THE 8086 MICROPROCESSOR **9**

Introduction to 8086 – Microprocessor architecture – Addressing modes - Instruction set and assembler directives – Assembly language programming – Modular Programming - Linking and Relocation - Stacks - Procedures – Macros – Interrupts and interrupt service routines – Byte and String Manipulation.

UNIT II 8086 SYSTEM BUS STRUCTURE **9**

8086 signals – Basic configurations – System bus timing –System design using 8086 – I/O programming – Introduction to Multiprogramming – System Bus Structure – Multiprocessor configurations – Coprocessor, Closely coupled and loosely Coupled configurations – Introduction to advanced processors.

UNIT III I/O INTERFACING **9**

Memory Interfacing and I/O interfacing - Parallel communication interface – Serial communication interface – D/A and A/D Interface - Timer – Keyboard /display controller – Interrupt controller –DMA controller – Programming and applications Case studies: Traffic Light control, LED display , LCD display, Keyboard display interface and Alarm Controller.

UNIT IV MICROCONTROLLER **9**

Architecture of 8051 – Special Function Registers(SFRs) - I/O Pins Ports and Circuits - Instruction set - Addressing modes - Assembly language programming.

UNIT V INTERFACING MICROCONTROLLER **9**

Programming 8051 Timers - Serial Port Programming - Interrupts Programming – LCD & Keyboard Interfacing - ADC, DAC & Sensor Interfacing - External Memory Interface- Stepper Motor and Waveform generation - Comparison of Microprocessor, Microcontroller, PIC and ARM processors.

TOTAL: 45 PERIODS

OUTCOMES:

At the end of the course, the students should be able to:

- Understand and execute programs based on 8086 microprocessor.
- Design Memory Interfacing circuits.
- Design and interface I/O circuits.
- Design and implement 8051 microcontroller based systems.

TEXT BOOKS:

1. Yu-Cheng Liu, Glenn A.Gibson, “Microcomputer Systems: The 8086 / 8088 Family - Architecture, Programming and Design”, Second Edition, Prentice Hall of India, 2007. (UNIT I- III)

2. Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay, “The 8051 Microcontroller and Embedded Systems: Using Assembly and C”, Second Edition, Pearson education, 2011. (UNIT IV-V)

REFERENCES:

1. Doughlas V.Hall, “Microprocessors and Interfacing, Programming and Hardware”, TMH, 2012

2. A.K.Ray, K.M.Bhurchandi, ”Advanced Microprocessors and Peripherals” 3rd edition, Tata McGrawHill,2012

UNIT I THE 8086 MICROPROCESSOR

PART-A (2 MARKS)

1. Write about the different types of interrupts supported in 8086. [Apr/May 2015]

The following are the various types of interrupts:

Type 0 interrupts: This interrupt is also known as the divide by zero interrupt. For cases where the quotient becomes particularly large to be placed / adjusted an error might occur.

Type 1 interrupts: This is also known as the single step interrupt. This type of interrupt is primarily used for debugging purposes in assembly language.

Type 2 interrupts: also known as the non-maskable NMI interrupts. These types of interrupts are used for emergency scenarios such as power failure.

Type 3 interrupts: These types of interrupts are also known as breakpoint interrupts. When this interrupt occurs a program would execute up to its break point.

Type 4 interrupts: Also known as overflow interrupts is generally existent after an arithmetic Operation was performed.

2. Compare CALL and PUSH instructions CALL PUSH. [Nov/Dec 2011]

CALL	PUSH 22
When CALL is executed the microprocessor automatically stores the 16-bit address of the instruction next to CALL on the stack	The programmer uses the instruction PUSH to save the contents of the register pair on the stack
When CALL is executed the stack pointer is decremented by two	When PUSH is executed the stack pointer is register is decremented by two

3. What is assembler? [April/May 2008, Nov/Dec 2011,Apr/May2011]

The assembler translates the assembly language program text which is given as input to the assembler to their binary equivalents known as object code. The time required to translate the assembly code to object code is called access time. The assembler checks for syntax errors & displays them before giving the object code.

4. What is interrupt service routine? [Nov/Dec 2011]

Interrupt means to break the sequence of operation. While the CPU is executing a program an interrupt breaks the normal sequence of execution of instructions & diverts its execution to some other program. This program to which the control is transferred is called the interrupt service routine.

5. What are Macros? [Nov/Dec 2011]

Macro is a group of instruction. The macro assembler generates the code in the program each time where the macro is called. Macros are defined by MACRO & ENDM directives. Creating macro is similar to creating new opcodes that can be used in the program

```
INIT MACRO
        MOV AX, data
        MOV DS
        MOV ES, AX
ENDM
```

6. Compare Procedure & Macro [NOV/DEC 2011]

Procedure	Macro
Accessed by CALL & RET instruction during program execution	Accessed during assembly with name given to macro when defined
Machine code for instruction is put only once in the memory	Machine code is generated for instruction each time when macro is called
With procedures less memory is required	With macro more memory is required
Parameters can be passed in registers, memory locations or stack	Parameters passed as part of statement which calls macro

7. What is the purpose of segment registers in 8086? [April/May2017, April/May2008, Nov/Dec 2006, 2011]

There are 4 segment registers present in 8086. They are Code Segment (CS) register, Data Segment (DS) register, Stack Segment (SS) register, Extra Segment (ES) register. The code segment register gives the address of the current code segment. ie. It will points out where the instructions, to be executed, are stored in the memory. The data segment register points out where the operands are stored in the memory. The stack segment registers points out the address of the current stack, which is used to store the temporary results. If the amount of data used is more, the Extra segment registers points out where the large amount of data is stored in the memory.

8. Define pipelining? [Nov/Dec 2006, Nov/Dec2011]

In 8086, to speed up the execution of program, the instructions fetching and execution of instructions are overlapped each other. This technique is known as pipelining. In pipelining, when the n^{th} instruction is executed, the $n+1^{th}$ instruction is fetched and thus the processing speed is increased.

9. Discuss the function of instruction queue in 8086? [Nov/Dec 2006][Apr/May2011]

In 8086, a 6-byte instruction queue is presented at the Bus Interface Unit (BIU). It is used to prefetch and store at the maximum of 6 bytes of instruction code from the memory. Due to this, overlapping instruction fetch with instruction execution increases the processing speed.

10. Draw the Flag register format of 8086? [April/May 2011, nov/Dec 2016]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

11. What are the two modes of operations present in 8086? [May/June2007]

1. Minimum Mode (or) Uniprocessor System
2. Maximum Mode (or) Multiprocessor System

12. What are the three classifications of 8086 interrupts? [May/June-2006]

(1) Predefined Interrupts
(2) User Defined Hardware Interrupts
(3) User Defined Software Interrupts.

13. What is the processing element inside the microprocessor? What process it does? [Nov/Dec 2014]

The processing element inside the microprocessor is the ALU. It performs all computing operation such as Addition, Subtraction, Multiplication, and Division and Logical operation.

14.Calculate the physical address, when segment address is 1085H and effective address is 4537 H. [Nov/Dec 2015, April 2017]

Effective address= 4 5 3 7 +
Segment address= 1 0 8 5 0
Physical address = 1 4 D 8 7

15. What is microprocessor?

A microprocessor is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary information from a storage device called memory, accepts binary data as input and processes data according to those instructions, and provides result as output.

16. What is Accumulator?

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

17. What is stack? (EE2354April/May2013)

The stack is a group of memory locations in the R/W memory that is used for temporary storage of binary information during the execution of a program

18. What is a subroutine program?

A subroutine is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program. Thus subroutines avoid the repetition of same set of instructions in the main program.

19. Define addressing mode.

Addressing mode is used to specify the way in which the address of the operand is specified within the instruction.

20. Define instruction cycle.

It is defined as the time required to complete the execution of an instruction

21. Write a program to add a data byte located at offset 0500H in 2000H segment to another data byte available at 0600H in the same segment and store the result at 0700H in the same segment.

MOV AX, 2000H; initialize DS with value MOVDS, AX; 2000H
MOV AX, [500H]; Get first data byte from 0500H offset ADD AX, [600H]; Add this to the second byte from 0600H MOV [700H], AX; store AX in 0700H
HLT; Stop.

22. What are the different types of addressing modes of 8086 instruction set? (Nov/Dec2013) (Apr/May 2015)

The different addressing modes are:

i. Immediate
ii. Direct
iii. Register
iv. Register indirect
v. Indexed
vi. Register relative
vii. Based indexed
viii. Relative based indexed.

23. What are the different types of instructions in 8086 microprocessor? (May/jun2011)

The different types of instructions in 8086 microprocessor are:

i. Data copy / transfer instructions
ii. Arithmetic and logical instructions
iii. Branch instructions
iv. Machine control instruction
v. Flag manipulation instruction
vi. Shift and rotate instruction
vii. String instruction

24. What is assembly level programming?

A program called assembler is used to convert the mnemonics of instruction and data into their equivalent object code modules. The object code modules are further converted into executable code using linker and loader programs. This type of programming is calledassembly level programming.

25. What is a stack?

Stack is a top-down data structure, whose elements are accessed using a pointer that is implemented using the SS and SP registers. It is a LIFO data segment.

26. How is the stack top address calculated?

The stack top address is calculated using the contents of the SS and SP register. The contents of stack segment (SS) register is shifted left by four bit positions (multiplied by(0h)) and the resulted 20-bit content is added with the 16-bit offset value of the stack pointer(SP) register.

27. What are macros?

Macros are small routines that are used to replace strings in the program. They can have parameters passed to them, which enhances the functionality of the micro itself.

28. How are constants declared?

Constants are declared in the same way as variables, using the format:
Const–Label EQU 012h

When the constants label is encountered, the constant numeric value is exchanged for the string.

29. Write an assembly language program for a 16-bit increment and will not affect the contents of the accumulator.

MACRO inc16variable; Increment two bytes starting at "variable"
Local INC16 End
INC variable; Increment the low 8 bits PUSH ACC
MOV A variable; Are the incremented low 8 bits = 0? JNZ INC 16 End
INC variable + 1
Inc16 End; Yes–increment the upper 8 bits POP ACC
END MAC

30. What will happen if a label within a macro is not declared local?

If a label within a macro is not declared local, then at assembly time, there will be two types of errors:
I. The first will state that there are multiple labels in the source.
II. The second will indicate that jump instructions don't know which one to use.

31. Write an assembly language program to load the accumulator with a constant value.

MACRO invert value if (value==0)
MOV A, #1
else clr A end if
END MAC.

32. What is the difference between the microprocessor and microcontroller?

Microprocessor does not contain RAM, ROM and I/O ports on the chip. But a microcontroller contains RAM, ROM and I/O ports and a timer all on a single chip.

33. What is assembler? (NOV/DEC2014)

The assembler translates the assembly language program text which is given as input to the assembler to their binary equivalents known as object code. The time required to translate the assembly code to object code is called access time. The assembler checks for syntax errors & displays them before giving the object code.

34. What is loader?

The loader copies the program into the computer's main memory at load time and begins the program execution at execution time.

35. What is linker?

A linker is a program used to join together several object files into one large object file. For large programs it is more efficient to divide the large program modules into smaller modules. Each module is individually written, tested & debugged. When all the modules work they are linked together to form a large functioning program.

36. Explain ALIGN & ASSUME. (Nov/Dec 2010, April/may2011)

The ALIGN directive forces the assembler to align the next segment at an address divisible by specified divisor. The format is ALIGN number where number can be 2,4, 8 or 16.

Example ALIGN 8.

The ASSUME directive assigns a logical segment to a physical segment at any given time. It tells the assembler what address will be in the segment registers at execution time.

Example ASSUME CS: code, DS: data, SS: stack

37. Explain PTR & GROUP

A program may contain several segments of the same type. The GROUP directive collects them under a single name so they can reside in a single segment, usually a data segment. The format is Name GROUP Seg-name,…..Seg-name PTR is used to assign a specific type to a variable or a label. It is also used to override the declared type of a variable.

38. Explain PROC & ENDP (April/May 2010)

PROC directive defines the procedures in the program. The procedure name must be unique. After PROC the term NEAR or FAR are used to specify the type of procedure.
Example FACT PROC FAR.

ENDP is used along with PROC and defines the end of the procedure.

39. Explain SEGMENT & ENDS

An assembly program in .EXE format consists of one or more segments. The starts of these segments are defined by SEGMENT and the end of the segment is indicated by ENDS directive. Format Name SEGMENT.

40. Define SOP (Nov/Dec2010)

The segment override prefix allows the programmer to deviate from the default Segment
Eg : MOV CS: [BX] , AL

41. Define variable.

A variable is an identifier that is associated with the first byte of data item. In assembly language statement: COUNT DB 20H, COUNT is the variable.

42. What are procedures?

Procedures are a group of instructions stored as a separate program in memory and it is called from the main program whenever required. The type of procedure depends on where the procedures are stored in memory. If it is in the same code segment as that of the main program then it is a near procedure otherwise it is a far procedure.

43. Explain the linking process.

A linker is a program used to join together several object files into one large object file. The linker produces a link file which contains the binary codes for all the combined modules. It also produces a link map which contains the address information about the link files. The linker does not assign Absolute addresses but only relative address starting from zero, so the programs are relocatable& can be put anywhere in memory to be run.

44. Compare Procedure & Macro.(April/May2011)

Procedure	Macro
Accessed by CALL & RET instruction during program execution	Accessed during assembly with name to macro when defined
Machine code for instruction is put only Once in the memory	Machine code is generated for instruction each time when macro is called
With procedures less memory is required	With macro more memory is required
Parameters can be passed in registers, memory locations or stack	Parameters passed as part of statement Which calls macro

45. What is the maximum memory size that can be addressed by 8086? (April/May 2014) (Nov/Dec 2014)

In 8086, a memory location is addressed by 20 bit address and the address bus is 20 bit address and the address bus is 20 bits. So it can address up to one megabyte (220) of memory space.

46. How many data lines and address lines are available in 8086?

Address lines= 20 bit address bus Data lines= 16 bit data bus

47. What information is conveyed when Qs1 and Qs0 are 01?

Qs1 and Qs0 are output signals that reflect the status of the instruction queue. When Qs1 and Qs0 are 01, then queue has first byte of an opcode.

48. What is the addressing mode of MOV AX, 55H (BX) (SI) ?

MOV AX, 55H (BX) (SI) – Base Indexed memory addressing mode.

49. What are the 8086 interrupt types?(Apr/May 2015)

Dedicated interrupts

- Type 0: Divide by zero interrupt
- Type 1: Single step interrupt
- Type 2: Nonmaskable interrupt
- Type 3: Breakpoint
- Type 4: Overflow interrupt

Software interrupts: Type 0-255

50. What is interrupt service routine?[NOV/DEC 2011]

Interrupt means to break the sequence of operation. While the CPU is executing a program an interrupt breaks the normal sequence of execution of instructions & diverts its execution to some other program. This program to which the control is transferred is called the interrupt service routine.

51. Calculate the physical address for fetching the next instruction to be executed, in 8086?

The physical address is obtained by appending four zeros to the content present in CS register and then adding the content of IP register with the above value.

For example, assuming the content of CS = 1200 H IP = 0345 H

CS= 0001 0010 0000 0000 0000

0000 0011 0100 0101

0001 0010 0011 0100 0101 – Physical address=12345 H

52. If the execution unit generates effective address of 43A2 H and the DS register contains 4000 H. What will be the physical address generated by the BIU? What is the Maximum Size of the data segment?

Effective Address 43A2H

Physical Address 40000H Maximum size of the DS is 216

Physical Address = 40000H + 43A2H= 443A2H

53. Calculate the physical address, when segment address is 1085H and effective address is 4537H. [Nov/Dec 2015]

Segment address - 1085H Effective address - 4537H

Physical address - 14D87H

---

54. Show how the 2 byte INT instruction can be applied for debugging. [Nov/Dec2015]

INT type

The INT instruction is used as a debugging and in case where single stepping provides more detail then is wanted, by inserting INT instructions at key points called break points.

55. List the flags of 8086. [May/June 2016]

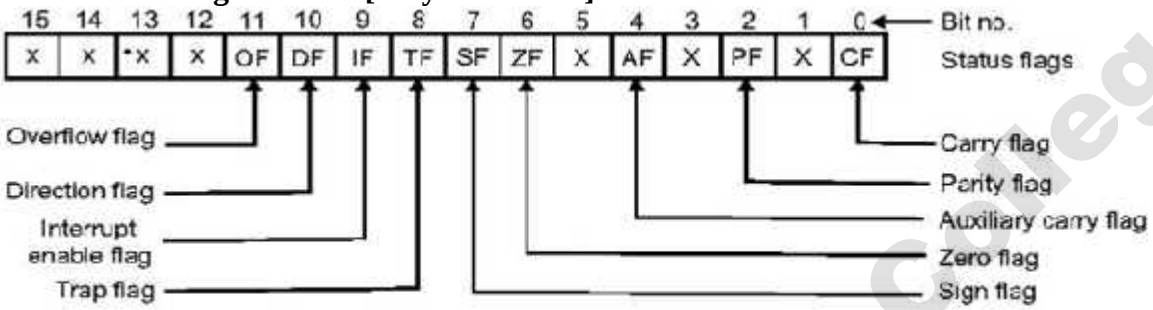


Fig.11.8: Status flags of intel 8086

- OF - Overflow Flag. Set if signed number exceeds capacity of result
- DF - Direction Flag. Set by user to indicate a direction (0=forward, 1=backward)
- IF - Interrupt Flag. Set by user to disable hardware interrupts temporarily
- TF - Trap Flag. Used by debuggers
- SF - Sign Flag
- ZF - Zero Flag
- AF – Aux
- PF - Parity Flag

56. What are the functional parts of 8086 CPU?

The two independent functional parts of the 8086 CPU are:

i. Bus Interface Unit (BIU):

BIU sends out addresses, fetches instruction from memory, reads data from ports and memory and writes data to ports and memory.

ii. Execution Unit (EU):

EU tells the BIU where to fetch instructions or data, decodes instructions and executes instructions.

57. What is the purpose of a decoder in EU?

The decoder in EU translates instructions fetched from memory into a series of actions, which the EU carries out.

58. Give the register classification of 8086.

The 8086 contains:

i. General purpose registers:

They are used for holding data, variables and intermediate results temporarily.

ii. Special purpose registers:

They are used as segment registers, pointers, index register or as offset storage registers for particular addressing modes.

59. What are general data registers?

The registers AX, BX, CX and DX are the general data registers.

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

L and H represent the lower and higher bytes of particular register.
AX register is used as 16-bit accumulator.
BX register is used as offset storage for forming physical addresses in case of certain addressing modes.
CX register is used as a default counter in case of string and loop instructions.
DX register is used as an implicit operand or destination in case of a few instructions.

60. Give the different segment registers.

The four segment registers are:
i. Code segment register:
It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
ii. Data segment register:
It points to the data segment of the memory, where data is resided.
iii. Extra segment register:
It also contains data.
iv. Stack segment register:
It is used for addressing stock segment of memory. It is used to store stack data.

CS
SS
DS
ES

Segment registers

61. What are pointers and index registers?

IP, BP and SP are the pointers and contain offsets within the code, data and stack segments respectively. SI and DI are the index registers, which are used as general purpose registers and also for offset storage in case of indexed, based indexed and relative based indexed addressing modes.

SP
BP
SI
DI
IP

62. Mention the addressing modes of the following 8086 instructions

Mov al, disp(ax)- Register Relative Addressing Mode

Mov AH, DISP [bx][si] – Relative Based Indexed Addressing Mode

63. How Will Carry And Zero Flags Reflect the Result of The Instruction Cmp Bx,Cx?

	CF	ZF
BX=CX	0	1
BX>CX	0	0
BX<CX	1	0

64. What do these instructions do?

STD, IRET

STD- Set the direction flag register(D=1)

IRET- Interrupt on return. It is used to exist any interrupt procedure, whenever activated by hardware and software

65. What is the use of HOLD and HLDA signals?

HOLD- The signal indicates the another master is requesting the host 8086 to handover the system bus

HLDA- On receiving the hold signal, 8086 outputs HLDA signal high as an acknowledgement

66. What is an assembler directive?

An assembler directive is a statement to give direction to the assembler to perform task of assembly process

67. How single stepping can be done in 8086?

By setting the Trace Flag (TF) the 8086 goes to single-step mode. In this mode, after the execution of each instruction s 8086 generates an internal interrupt and by writing some interrupt service routine we can display the content of desired registers and memory locations. So it is useful for debugging the program.

68. What are the functions of bus interface unit (BIU) in 8086?

(a) Fetch instructions from memory.
(b) Fetch data from memory and I/O ports.
(c) Write data to memory and I/O ports.
(d) To communicate with outside world.
(e) Provide external bus operations and bus control signals.

69. Explain the process of control instructions

STC – It sets the carry flag & does not affect any other flag

CLC – it resets the carry flag to zero &does not affect any other flag

CMC – It complements the carry flag & does not affect any other flag

STD – It sets the direction flag to 1 so that SI and/or DI can be decremented automatically after execution of string instruction & does not affect other flags

CLD – It resets the direction flag to 0 so that SI and/or DI can be incremented automatically after execution of string instruction & does not affect other flags

STI – Sets the interrupt flag to 1. Enables INTR of 8086.

CLI – Resets the interrupt flagto0. 8086 will not respond to INTR.

70. Discuss the function of instruction queue in 8086?

In 8086, a 6-byte instruction queue is presented at the Bus Interface Unit (BIU). It is used to prefetch and store at the maximum of 6 bytes of instruction code from the memory. Due to this, overlapping instruction fetch with instruction execution increases the processing speed.

71. What is the maximum memory size that can be addressed by 8086?

In 8086, a memory location is addressed by 20 bit address and the address bus is 20 bit address and the address bus is 20 bits. So it can address up to one mega byte (2^20) of memory space.

72. Explain DUP

The DUP directive can be used to initialize several locations & to assign values to these locations. Format Name Data_Type Num DUP (value) .Example TABLE DW 10 DUP (0). Reserves an array of 10 words of memory and initializes all 10 words with 0. array name is TABLE.

73. Explain PUBLIC

For large programs several small modules are linked together. In order that the modules link together correctly any variable name or label referred to in other modules must be declared public in the module where it is defined. The PUBLIC directive is used to tell the assembler that a specified name or label will be accessed from other modules

74. What is Microcontroller and Microcomputer

Microcontroller is a device that includes microprocessor; memory and I/O signal lines on a single chip, fabricated using VLSI technology. Microcomputer is a computer that is designed using microprocessor as its CPU. It includes microprocessor, memory and I/O.

75. What are libraries?

Library files are collection of procedures that can be used in other programs. These procedures are assembled and compiled into a library file by the LIB program. The library file is invoked when a program is linked with linker program. when a library file is linked only the required procedures are copied into the program. Use of library files increase s/w reusability & reduce s/w development time.

76. How do 8086 interrupts occur?

An 8086 interrupt can come from any of the following three sources

- External signals
- Special instructions in the program
- Condition produced by instruction

77. What is interrupt service routine

Interrupt means to break the sequence of operation. While the CPU is executing a program an interrupt breaks the normal sequence of execution of instructions & diverts its execution to some other program. This program to which the control is transferred is called the interrupt service routine.

78. What is multiple interrupt processing capability?

Whenever a number of devices interrupt a CPU at a time, and id the processor is able to handle them properly, it is said to have multiple interrupt processing capability.

79. What is hardware interrupt?

An 8086 interrupt can come from any one of three sources. One sources is an external signal applied to the nonmarkable interrupt (NMI) input in or to the interrupt (INTR) input pin. An interrupt caused by the signal applied to one of these input is referred to as a hardware interrupt.

80. What is software interrupt?

The interrupt caused due to execution of interrupt instruction is called software interrupt.

81. What is assembly level programming?

A program called „assembler" is used to convert the mnemonics of instruction and data into their equivalent object code modules. The object code modules are further converted into executable code using linker and loader programs. This type of programming is called assembly level programming.

82. How is the physical address calculated? Give an example.

The physical address, which is 20-bits long is calculated using the segment and offset registers, each 16-bits long. The segment address is shifted left bit-wise four times and offset address is added to this to produce a 20 bit physical address.

Eg: segment address - > 1005H
Offset address - > 5555H
Segment address - > 1005H - > 0001 0000 0000 0101
Shifted by 4 bit position - > 0001 0000 0000 0101 0000
Offset address - > + 0101 0101 0101 0101
Physical address - > 0001 0101 0101 1010 0101
1 5 5 A 5

83. Explain the three machine control flags.

i. Trap flag:

If this flag is set, the processor enters the single step execution.

ii. Interrupt flag:

If this flag is set, the markable interrupts are recognized by the CPU, otherwise they are ignored.

iii. Direction flag:

This is used by string manipulation instructions. If this flag bit is „0", the string is processed from the lowest to the highest address i.e., auto incrementing mode. Otherwise, the string is processed from highest address to lowest address, i.e., auto decrementing mode.

84. Define Modular Programming

It is defined as subdivision of computer program into separate programs.

PART –B (13 MARKS)

1. ARCHITECTURE OF 8086 MICROPROCESSOR

Explain briefly about the internal hardware architecture of 8086 microprocessor with a neat diagram. (10)[**Apr/May 2015, April/May2017**]

Explain the bus interface unit and execution unit of 8086 microprocessor. (8) [**Nov/Dec 2014**].

Explain The Architecture Of 8086 Microprocessor. (8) [**Nov / Dec 2012]**

Describe the hardware architecture of 8086 microprocessor with neat diagram. (10) [**Nov /Dec 2013]**

Explain the internal hardware architecture of 8086 microprocessor with neat diagram. (12) [**April/May 2011]**

Intel 8086 is a 16 bit processor. It has 16-bit data bus and 20-bit address bus. The lower 16-bit address lines and 16-bit data lines are multiplexed (AD0-AD15). Since 20-bit address lines are available, 8086 can access up to 220 or 1 Giga byte of physical memory. The architecture of the 8086 can be internally divided into two separate functional units as shown in figure 1.1

Bus Interface Unit (BIU) and Execution Unit (EU)

Bus Interface Unit (BIU)

The BIU fetches instructions, reads data from memory and IO ports, writes data to memory and IO ports. The BIU contains segment registers, instruction pointer, instruction queue, address generation unit and bus control unit. The Bus Interface Unit (BIU) generates the 20-bit physical memory address. To speed up the execution, 6-bytes of instruction are fetched in advance and kept in a 6-byte Instruction Queue called pipe-lining. In 8086 microprocessor memory are divided into four parts which is known as the segments as shown in figure 1.2. These segments are data segment, code segment, stack segment and extra segment. Each segments of 64 kilo bytes.

The BIU has four numbers of 16-bit segment registers. They are **C**ode **S**egment (CS) register, **D**ata **S**egment (DS) register, **S**tack **S**egment (SS) register and **E**xtra **S**egment (ES) register. The 4 segment registers are used to hold four segment base addresses.

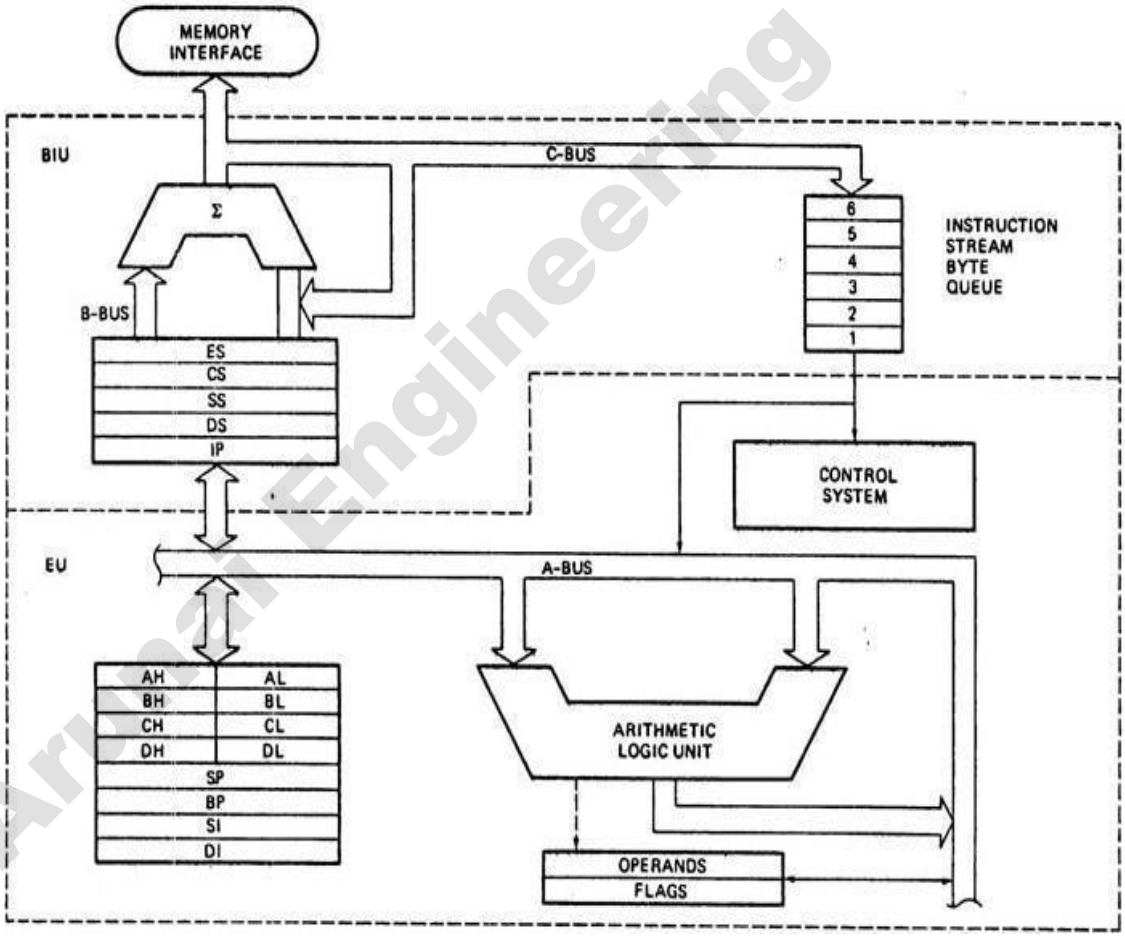


Figure: 1.1 Architecture of 8086 Microprocessor

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The programs will be stored in code segment region. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. Data related with stack operation are stored in this segment region. All data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. Data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. The DI register references the ES segment in string manipulation instructions. The address for fetching instruction codes is generated by logically shifting the content of the CS to the left four times and then adding it to the content of the IP (**I**nstruction **P**ointer). The IP holds the offset address of the program codes.

Code segment Register CS holds the segment address which is 4569 H
Instruction pointer IP holds the offset address which is 10A0 H The
physical 20-bit address is calculated as follows.

Segment address: 45690 H
Offset address :+10A0 H
Physical address : 46730 H

The data address is computed by using the content of DS or ES as base address and an offset or effective address specified by the instruction. The stack address is computed by using the content of the SS as base address and the content of the SP (**S**tack **P**ointer) as the offset address or effective address.

Execution Unit (EU)

The EU executes instructions that have already been fetched by the BIU. The BIU and EU function independently. The instruction queue is a FIFO (**F**irst-**I**n-**F**irst-**O**ut) group of registers. The size of queue is 6 bytes. The BIU fetches instruction code from the memory and stores it in the queue. The EU fetches instruction codes from the queue.

- The EU receives program instruction codes and data from the BIU, executes these instructions, and store the results in the general registers. It receives and outputs all its data through the BIU.
- A decoder in the EU translates instructions fetched from memory into a series of actions which the EU carries out.

- The EU has a 16-bit ALU which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers. The EU decodes an instruction or executes an instruction.

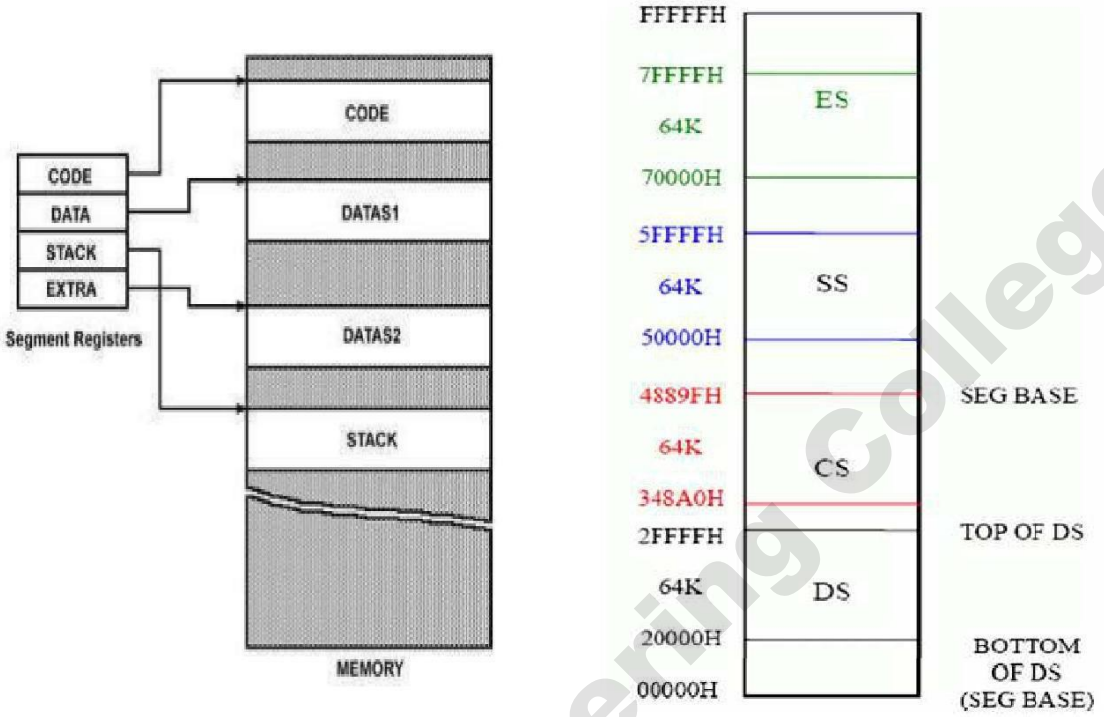


Figure: 1.2 Memory Organization of 8086 Microprocessor

Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
Base Register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BX register
Count Register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. Count register can be used as a counter in string manipulation and shift/rotate instructions.
Data Register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. Data register can be used as a port number in I/O operations

The following registers are both general and index registers:
Stack Pointer (SP) is a 16-bit register pointing to program stack.
Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.
Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.
Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Instruction Pointer (IP) is a 16-bit register which points to the instruction fetched from memory.

Flag register is a 16-bit register containing nine 1-bit flags:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

U – Undefined
OF-Overflow Flag
DF-Direction Flag
IF-Interrupt Enable Flag
TF-Single Step Trap Flag
SF-Sign Flag
ZF-Zero Flag
AF-Auxiliary Flag
PF-Parity Flag
CF-Carry Flag

Overflow Flag (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.

Direction Flag (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.

Interrupt-enable Flag (IF) - setting this bit enables maskable interrupts.

Single-step Trap Flag (TF) - if set then single-step interrupt will occur after the next instruction.

Sign Flag (SF) - set if the most significant bit of the result is
set. Zero Flag (ZF) - set if the result is zero.

Auxiliary carry Flag (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register during BCD operation.

Parity Flag (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.

Carry Flag (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

2. ADDRESSING MODES OF 8086

Explain the different addressing modes of 8086 microprocessor with examples. (8) [**Nov/Dec 2014**].

Explain the different addressing modes of 8086 microprocessor. (16)[**Apr/May 2015**]

Explain the various addressing modes of 8086 processor with suitable examples. (10) [**Nov/Dec 2011]**

The addressing modes of 8086 are divided into

- Immediate Addressing Mode
- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- String Addressing Mode
- Indexed Addressing Mode
- Base Addressing Mode

- Base Indexed Addressing Mode
- Relative Addressing Mode
- Implied Addressing Mode

Immediate Addressing Mode:

8 or 16 bit data can be specified as part of the instruction.

Example: MOV CL, 03 H Moves the 8 bit data 03 H into CL

Register Addressing Mode:

The operand to be accessed is specified as an internal register of 8086.

Example: MOV DX, CX Moves 16 bit content of CX into DX.

Direct Addressing Mode:

The instruction Opcode is followed by an effective address, this effective address is directly used as the 16 bit offset of the storage location of the operand from the location specified by the current value in the selected segment register.

Example: MOV CX, [5000] If DS = 0050. Then BIU generates the 20 bit physical address 50050 H. The content of 50050 is moved to CL. The content of 50051 is moved to CH.

Register Indirect Addressing Mode:

The EA is specified in either pointer (BX) register or an index (SI or DI) register. The 20 bit physical address is computed using DS and DI.

Example: MOV BX, [DI]If [DS] = 5000, [DI] = 0020, PA=50020.The Content of 50020 and 50021 is moved to BX Register

String Addressing Mode:

The string instructions automatically assume SI to point to the first byte or word of the source operand and DI to point to the first byte or word of the destination operand. The contents of SI and DI are automatically incremented (by clearing DF to 0 by CLD instruction) to point to the next byte or word.

Example: MOVSB

If [DF] = 0, [DS] = 2000 H, [SI] = 0500, [ES] = 4000, [DI] = 0300

Source address: 20500, [DS] + [SI] Destination address: [ES] + [DI] = 40300. The data from source address is transferred to the destination address

Indexed Addressing Mode:

PA = (CS, DS, SS, ES): (SI or DI) + 8 or 16bit displacement

Example: MOV BH, START [SI]

PA: [START] + [SI] + [DS].The content of this memory is moved into BH.

Base Addressing Mode:

PA = (CS, DS, SS, ES): (BX or BP) + displacement

Example: MOV AL, START [BX]

EA: [START] + [BX]. The content of this memory is moved into AL

Base Indexed Addressing Mode:

PA = (CS, DS, SS, ES): (SI or DI) + (BX or BP)+8 or 16 bit displacement

Example: MOV ALPHA [SI] [BX], CL

EA: ALPHA + [SI] + [BX]. The content of CL is moved this memory.

Relative Addressing Mode:

Example: JNC START

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

Implied Addressing Mode:

Instruction using this mode has no operands.

Example: CLC which clears carry flag to zero.

3. INSTRUCTION SET OF 8086

Give three examples for the following 8086 microprocessor instructions: String Instructions, Process Control Instruction, Program Execution Transfer Instructions and Bit manipulation Instructions. (12 Marks) **[April/May 2010]**

Explain the data transfer group and logical group of 8086 instructions. [Marks 10] [**April/May 2011]**

Discuss About The Different Data Transfer Schemes With Examples. (8) **[Nov / Dec 2012]**

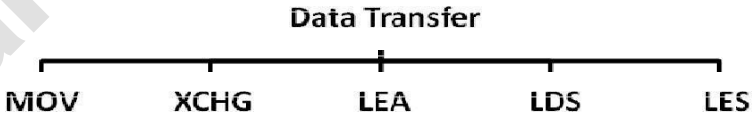
Discuss About The 8086 Instruction Used For Transferring Data Between Registers, Memory, Stacks, And I/O Devices. (8) **[Nov / Dec 2012]**

Explain the data transfer group and logical group of 8086 instruction with necessary examples. (10) [**Nov /Dec 2013]**

a. Data Transfer Instructions
b. Arithmetic Instructions
c. Logical Instructions
d. Shift and Rotate Instructions
e. Branch Instructions
f. Loop Instructions

a. String Manipulating Instructions
b. Flag Manipulation Instructions.
c. Stack Related Instructions
d. Input-Output Instructions
e. Machine Control Instructions

a. DATA TRANSFER INSTRUCTIONS



MOV – MOV Destination, Source

The MOV instruction copies a word or byte of data from a specified source to a specified destination. The destination can be a register or a memory location. The source can be a register, a memory location or an immediate number.

MOV CX, 037AH Put immediate number 037AH to CX

MOV BL, [437AH]	Copy byte in DS at offset 437AH to BL
MOV AX, BX	Copy content of register BX to AX
MOV DL, [BX]	Copy byte from memory at [BX] to DL

XCHG – XCHG Destination, Source
The XCHG instruction exchanges the content of a register with the content of another register or with the content of memory location(s).

XCHG AX, DX	Exchange word in AX with word in DX
XCHG BL, CH	Exchange byte in BL with byte in CH

LEA – LEA Register, Source
This instruction determines the offset of the variable or memory location named as the source and puts this offset in the indicated 16-bit register.

LEA CX, [BX][DI]	Load CX with EA = [BX] + [DI]

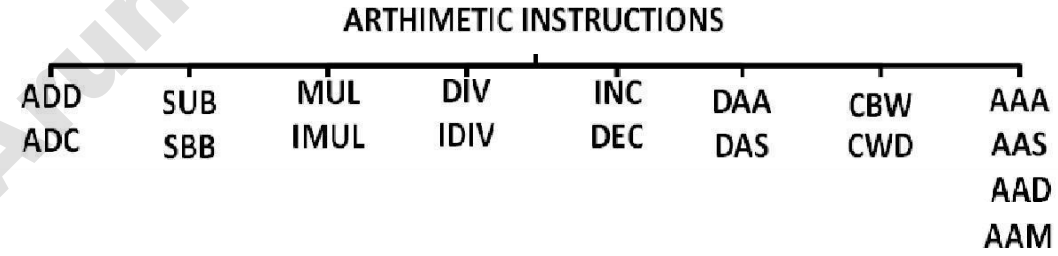
LDS – LDS Register, Memory address of the first word
The word from two memory locations is copied into the specified register and the word from the next two memory locations is copied into the DS registers.

LDS BX, [4326]	Copy content of memory at displacement 4326H in DS to BL, content of 4327H to BH. Copy content at displacement of 4328H and 4329H in DS to DS register.

LES – LES Register, Memory address of the first wor d
The word from the first two memory locations is copied into the specified register, and the word from the next two memory locations is copied into the ES register.

LES BX, [789AH]	Copy content of memory at displacement 789AH in DS to BL, content of 789BH to BH, content of memory at displacement 789CH and 789DH in DS is copied to ES register.

b. ARITHMETIC INSTRUCTIONS



ADD – **ADD Destination, Source**
ADC – **ADC Destination, Source**
These instructions add a number from some *source* to a number in some *destination* and put the result in the specified destination. The ADC also adds the status of the carry flag to the result. The source may be an immediate number, a register, or a memory location.

ADD AL, 74H	Add immediate number 74H to content of AL. Result in AL
ADC CL, BL	Add content of BL plus carry status to content of CL
ADD DX, [SI]	Add word from memory at offset [SI] in DS to content of DX

SUB – SUB Destination, Source
SBB – SBB Destination, Source
These instructions subtract the number in some *source* from the number in some *destination* and put the result in the destination. The SBB instruction also subtracts the content of carry flag from the destination. The source may be an immediate number, a register or memory location.

SUB CX, BX	CX – BX; Result in CX
SUB AX, 3427H	Subtract immediate number 3427H from AX

MUL – MUL Source
This instruction multiplies an *unsigned* byte in some *source* with an *unsigned* byte in AL register or an unsigned word in some *source* with an unsigned word in AX register. When a byte is multiplied by the content of AL, the result (product) is put in AX. When a word is multiplied by the content of AX, the result is put in DX and AX registers.

MUL BL	Multiply AL with BL; result in AX
MUL CX	Multiply AX with CX; result high word in DX, low word in AX

IMUL – IMUL Source
This instruction multiplies a *signed* byte from *source* with a *signed* byte in AL or a *signed* word from some *source* with a *signed* word in AX. When a byte from source is multiplied with content of AL, the signed result (product) will be put in AX. When a word from source is multiplied by AX, the result is put in DX and AX.

IMUL BL	Multiply signed byte in AL with signed byte in BL; result in
IMUL BX	AX. Multiply BX with AX; result in DX and AX

DIV – DIV Source

This instruction is used to divide an *unsigned* word by a byte or to divide an *unsigned* double word (32 bits) by a word. When a word is divided by a byte, the word must be in the AX register. The divisor can be in a register or a memory location. After the division, AL will contain the **8-bitquotient**, and AH will contain the **8-bit remainder**.

When a double word is divided by a word, the most significant word of the double word must be in DX, and the least significant word of the double word must be in AX. After the division, AX will contain the **16-bit quotient** and DX will contain the **16-bit remainder**.

DIV BL	Divide word in AX by byte in BL; Quotient in AL, remainder in
DIV CX	AH Divide the word in DX and AX by word in CX; Quotient in AX, and remainder in DX.

IDIV – IDIV Source

This instruction is used to divide a *signed* word by a *signed* byte or to divide a *signed* double word by a *signed* word. When dividing a signed word by a signed byte, the word must be in the AX register. The divisor can be in an 8-bit register or a memory location. After the division, AL will contain the signed quotient, and AH will contain the signed remainder.

When dividing a signed double word by a signed word, the most significant word of the dividend (numerator) must be in the DX register, and the least significant word of the dividend must be in the AX register. The divisor can be in any other 16-bit register or memory location. After the division, AX will contain a signed 16-bit quotient, and DX will contain a signed 16-bit remainder.

IDIV BL	Signed word in AX/signed byte in BL
IDIV BP	Signed double word in DX and AX/signed word in BP

INC – INC Destination

The INC instruction adds 1 to a specified register or to a memory location..

INC BL	Add 1 to content of BL register
INC CX	Add 1 to content of CX register

DEC – DEC Destination

This instruction subtracts 1 from the destination word or byte.

DEC CL Subtract 1 from content of CL register DEC BP
Subtract 1 from content of BP register

DAA (DECIMAL ADJUST AFTER BCD ADDITION)

This instruction is used to convert the result of addition of two packed BCD numbers to a valid BCD number. The result has to be in AL. After an addition if the lower nibble in AL is greater than 9 or AF is set, then the DAA instruction will add 6 to the lower nibble in AL. If the result in the upper nibble of AL is not greater than 9, then the DAA instruction will add 60H to AL.

DAA	AL = D7H; upper nibble > 9, add 60H to AL
	AL = 37 BCD, CF = 1

DAS (DECIMAL ADJUST AFTER BCD SUBTRACTION)

This instruction is used after subtracting one packed BCD number from another packed BCD number, to make sure the result is correct packed BCD. The result has to be in AL. If the lower nibble in AL after a subtraction is greater than 9 or the AF was set, then the DAS instruction will subtract 6 from the lower nibble AL. If the result in the upper nibble is now greater than 9 or if the carry flag was set, the DAS instruction will subtract 60 from AL.

Let AL = 49 BCD, and BH = 72 BCD

SUB AL, BH	AL = D7H; upper nibble > 9, subtract 60H from
DAS	AL AL = 77 BCD, CF = 1 (borrow is needed)

CBW (CONVERT SIGNED BYTE TO SIGNED WORD)

This instruction copies the sign bit of the byte in AL to all the bits in AH. AH is then said to be the sign extension of AL.

Let AX = 00000000 10011011 (–155 decimal)

OUTPUT: AX = 11111111 10011011 (–155 decimal)

CWD (CONVERT SIGNED WORD TO SIGNED DOUBLE WORD)

This instruction copies the sign bit of a word in AX to all the bits of the DX register. In other words, it extends the sign of AX into all of DX.

AX = 11110000 11000111 (–3897 decimal)

OUTPUT: DX = 11111111 11111111AX = 11110000 11000111 (–3897 decimal)

AAA (ASCII ADJUST FOR ADDITION)

This instruction is executed after an ADD instruction that adds two ASCII coded operands to give a byte of result in AL. It converts the resulting contents of AL to unpacked decimal digits. After addition AAA instruction examines the lower 4 bits of AL to check whether it contains a valid BCD number in the range 0 to 9. If it is between 0 to 9, AAA instruction sets the higher 4 bits of AL to 0. AH is cleared before addition. If it greater than 9, AAA instruction increments the AL by 06, AH is incremented by 1and sets the higher 4 bits of AL to 0.

1. AL = 07 After AAA AL = 07
2. AL = 6A, AH = 00 ie AX 006A after AAA AX = 0100

AAS (ASCII ADJUST FOR SUBTRACTION)

Corrects the result in AL register after subtracting two unpacked ASCII operands. If the lower 4 bits are greater than 9 or if AF flag is 0 the AL is decremented by 6 and AH is decremented by 1.

AAM (BCD ADJUST AFTER MULTIPLY)

Converts the product available in AL into unpacked BCD format. Before you can multiply two ASCII digits, you must first mask the upper 4 bit of each. This leaves unpacked BCD (one BCD digit per byte) in each byte. After the two unpacked BCD digits are multiplied, the AAM instruction is used to adjust the product to two unpacked BCD digits in AX. AAM works only after the multiplication of two unpacked BCD bytes, and it works only the operand in AL. AAM updates PF, SF and ZF but AF; CF and OF are left undefined.

Let AL = 00000101 (unpacked BCD 5), and BH = 00001001 (unpacked BCD 9)

MUL BH AL x BH: AX = 00000000 00101101 = 002DH
AAM AX = 00000100 00000101 = 0405H (unpacked BCD for 45)

AAD (BCD-TO-BINARY CONVERT BEFORE DIVISION)

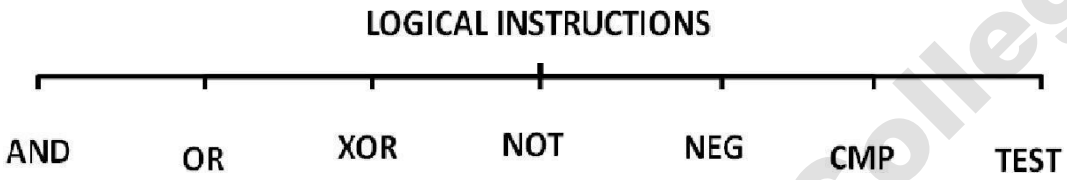
AAD converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. After the BCD division, AL will contain the unpacked BCD quotient and AH will contain the unpacked BCD remainder. AAD updates PF, SF and ZF; AF, CF and OF are left undefined.

Let AX = 0607 (unpacked BCD for 67 decimal), and CH = 09H

AAD	AX = 0043 (43H = 67 decimal)
DIV CH	AL = 07; AH = 04; Flags undefined after DIV

If an attempt is made to divide by 0, the 8086 will generate a type 0 interrupt.

c. LOGICAL INSTRUCTIONS



AND – AND Destination, Source

This instruction ANDs each bit in a source byte or word with the same numbered bit in a destination byte or word. The result is put in the specified destination.

AND BH, CL	AND byte in CL with byte in BH; Result in BH
AND BX, 00FFH	00FFH Masks upper byte, leaves lower byte unchanged.

OR – OR Destination, Source

This instruction ORs each bit in a source byte or word with the same numbered bit in a destination byte or word. The result is put in the specified destination.

OR AH, CL	CL ORed with AH, result in AH, CL not changed
OR BL, 80HBL	ORed with immediate number 80H; sets MSB of BL to 1

XOR – XOR Destination, Source

This instruction Exclusive-ORs each bit in a source byte or word with the same numbered bit in a destination byte or word. The result is put in the specified destination.

XOR CL, BH	Byte in BH exclusive-ORed with byte in CL .Result in CL.
XOR BP, DI	Word in DI exclusive-ORed with word in BP. Result in BP.

NOT – NOT Destination

The NOT instruction inverts each bit (forms the 1's complement) of a byte or word in the specified destination.

NOT BX	Complement content or BX register

NEG – NEG Destination

This instruction replaces the number in a destination with its 2's complement. It gives the same result as the *invert each bit and add one* algorithm.

NEG AL	Replace number in AL with its 2's complement

CMP – CMP Destination, Source

This instruction compares a byte / word in the specified source with a byte / word in the specified destination. The comparison is actually done by **subtracting** the source byte or word from the destination byte or word. The source and the destination are not changed, but the flags are set to indicate the results of the comparison.

	CF	ZF	SF	
CX = BX	0	1	0	Result of subtraction is 0
CX > BX	0	0	0	No borrow required, so CF = 0
CX < BX	1	0	1	Subtraction requires borrow, so CF = 1

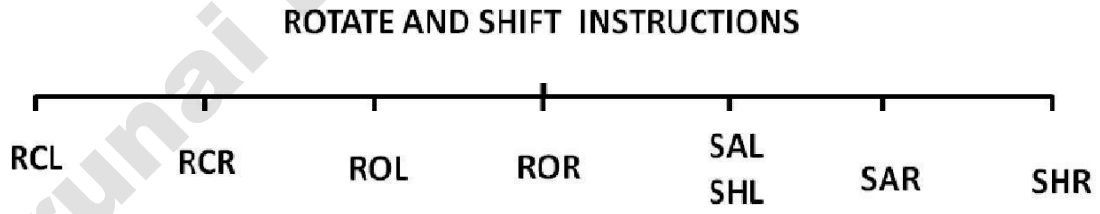
CMP AL, 01H	Compare immediate number 01H with byte in AL
CMP BH, CL	Compare byte in CL with byte in BH

TEST – TEST Destination, Source

This instruction ANDs the byte / word in the specified source with the byte / word in the specified destination. **Flags are updated, but neither operand is changed**. The test instruction is often used to set flags before a Conditional jump instruction.

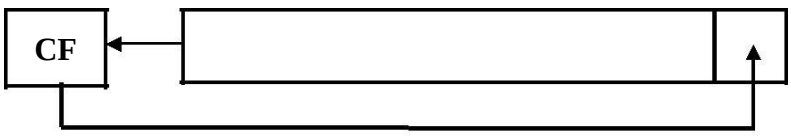
TEST AL, BH	AND BH with AL. No result stored; Update PF, SF, ZF.

d. ROTATE AND SHIFT INSTRUCTIONS



RCL – RCL Destination, Count

This instruction rotates all the bits in a specified word or byte some number of bit positions to the left. The operation is circular because the MSB of the operand is rotated into the carry flag and the bit in the carry flag is rotated around into LSB of the operand.

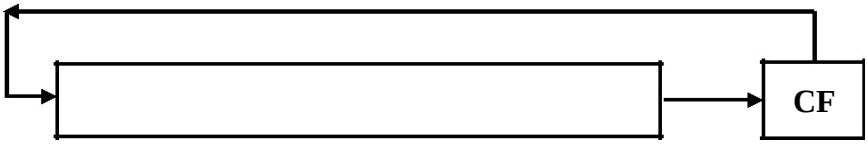


To rotate the operand by one bit position, specify this by putting a 1 in the count position of the instruction. To rotate by more than one bit position, load the desired number into the CL register and put "CL" in the count position of the instruction.

```
RCL DX, 1Word in DX 1 bit left
MOV CL, 4Load the number of bit positions to rotate into CL
    RCL DX, CL        Rotate DX register content 4 times left
```

RCR – RCR Destination, Count

This instruction rotates all the bits in a specified word or byte some number of bit positions to the right. The operation is circular because the LSB of the operand is rotated into the carry flag and the bit in the carry flag is rotated around into MSB of the operand.



If you want to rotate the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. To rotate more than one bit position, load the desired number into the CL register and put "CL" in the count position of the instruction.

```
RCR BX, 1          Word in BX right 1 bit
MOV CL, 4          Load CL for rotating 4 bit position
RCRBX, CL          Rotate BX register content 4 times right
```

ROL – ROL Destination, Count

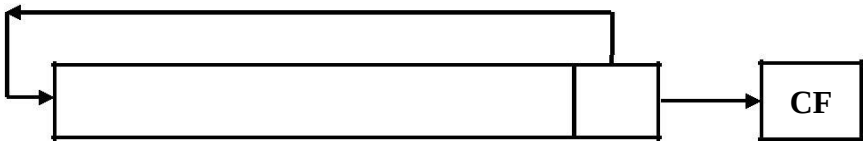
This instruction rotates all the bits in a specified word or byte to the left some number of bit positions. The data bit rotated out of MSB is circled back into the LSB. It is also copied into CF.

If you to want rotate the operand by one bit position, you can specify this by putting 1 in the count position in the instruction. To rotate more than one bit position, load the desired number into the CL register and put "CL" in the count position of the instruction.

```
ROL AX, 1          Rotate the word in AX 1 bit position left
MOV CL, 04HLoad number of bits to rotate in CL
    ROL BL, CL     Rotate BL register content 4 times left
```

ROR – ROR Destination, Count

This instruction rotates all the bits in a specified word or byte some number of bit positions to right. The operation is desired as a rotate rather than shift, because the bit moved out of the LSB is rotated around into the MSB. The data bit moved out of the LSB is also copied into CF.



To rotate the operand by one bit position, specify this by putting 1 in the count position in the

instruction. To rotate by more than one bit position, load the desired number into the CL register and put "CL" in the count position of the instruction.

```
ROR BL, 1          Rotate all bits in BL right 1 bit position
```

SAL – SAL Destination, Count
SHL – SHL Destination, Count
This instruction shifts each bit in the specified destination some number of bit positions to the left. As a bit is shifted out of the LSB operation, a 0 is put in the LSB position. The MSB will be shifted into CF.

To shift the operand by one bit position, specify this by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, load the desired number of shifts into the CL register, and put "CL" in the count position of the instruction.

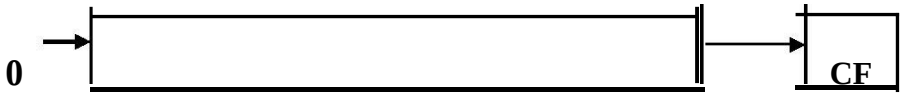
```
SAL BX, 1          Shift word in BX 1 bit position left, 0 in LSB
MOV CL, 02H        Load desired number of shifts in CL
SAL BX, CL         Shift word in BX left CL bit positions, 0 in LSBs
```

SAR – SAR Destination, Count
This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position. In other words, the sign bit is copied into the MSB. The LSB will be shifted into CF

To shift the operand by one bit position, specify this by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, load the desired number of shifts into the CL register, and put "CL" in the count position of the instruction.

```
SAR DX, 1Shift word in DX one bit position right, new MSB = old MSB
```

SHR – SHR Destination, Count
This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a 0 is put in its place. The bit shifted out of the LSB position goes to CF.



To shift the operand by one bit position, specify this by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, load the desired number of shifts into the CL register, and put "CL" in the count position of the instruction.

```
SHR BP, 1          Shift word in BP one bit position right, 0 in MSB
MOV CL, 03H        Load desired number of shifts into C
```

SHR BP, CL Shift BP register content 3 bits right; 0's in 3 MSBs

e. BRANCH INSTRUCTIONS

- **JA/JNBE**
- **JB/JC/JNAE**
- **JBE/JNA**
- **JG/JNLE**
- **JGE/JNL**
- **JL/JNGE**
- **JLE/JNG**
- **JE/JZ**
- **JNE / JNZ**
- **JS**
- **JNS**
- **JP / JPE**
- **JNP / JPO**
- **JO**
- **JNO**
- **JCXZ**

JMP (UNCONDITIONAL JUMP TO SPECIFIED DESTINATION)

This instruction will fetch the next instruction from the location specified in the instruction rather than from the next location after the JMP instruction. Two types of Jump instruction. Far Jump and Near Jump

If the destination is in the same code segment as the JMP instruction, then only the instruction pointer will be changed to get the destination location. This is referred to as a ***near jump***.
If the destination for the jump instruction is in a segment with a name different from that of the segment containing the JMP instruction, then both the instruction pointer and the code segment register content will be changed to get the destination location. This referred to as a ***far jump***. The JMP instruction does not affect any flag.

JMP CONTINUE

This instruction fetches the next instruction from address at label CONTINUE.

JA / JNBE JUMP IF ABOVE / JUMP IF NOT BELOW OR EQUAL)

If, after a compare or some other instructions which affect flags, the zero flag and the carry flag both are 0, this instruction will cause execution to jump to a label given in the instruction. If CF and ZF are not both 0, the instruction will have no effect on program execution.

```
JA NEXT          Jump to label NEXT if AX above 4371H
CMP AX, 4371H    Compare (AX – 4371H)
JNBE NEXT        Jump to label NEXT if AX not below or equal to 4371H
```

JAE / JNB / JNC
(JUMP IF ABOVE OR EQUAL / JUMP IF NOT BELOW / JUMP IF NO CARRY)

If, after a compare or some other instructions which affect flags, the carry flag is 0, this instruction will cause execution to jump to a label given in the instruction. If CF is 1, the instruction will have no effect on program execution.

```
CMP AX, 4371H    Compare (AX – 4371H)
JAE NEXT         Jump to label NEXT if AX above 4371H
```

CMP AX, 4371H	Compare (AX – 4371H)
JNB NEXT	Jump to label NEXT if AX not below 4371H

JB / JC / JNAE
(JUMP IF BELOW / JUMP IF CARRY / JUMP IF NOT ABOVE OR EQUAL)

If, after a compare or some other instructions which affect flags, the carry flag is a 1, this instruction will cause execution to jump to a label given in the instruction. If CF is 0, the instruction will have no effect on program execution.

CMP AX, 4371H	Compare (AX – 4371H)
JB NEXT	Jump to label NEXT if AX below 4371H
ADD BX, CX	Add two words
JC NEXT	Jump to label NEXT if CF = 1

JBE / JNA (JUMP IF BELOW OR EQUAL / JUMP IF NOT ABOVE)
If, after a compare or some other instructions which affect flags, either the zero flag or the carry flag is 1, this instruction will cause execution to jump to a label given in the instruction. If CF and ZF are both 0, the instruction will have no effect on program execution.

CMP AX, 4371H	Compare (AX – 4371H)
JBE NEXT	Jump to label NEXT if AX is below or equal to 4371H
CMP AX, 4371H	Compare (AX – 4371H)
JNA NEXT	Jump to label NEXT if AX not above 4371H

JG / JNLE (JUMP IF GREATER / JUMP IF NOT LESS THAN OR EQUAL)
This instruction is usually used after a Compare instruction. The instruction will cause a jump to the label given in the instruction, if the zero flag is 0 and the carry flag is the same as the overflow flag.

CMP BL, 39HCompare by subtracting 39H from BL

CMP BL, 39HCompare by subtracting 39H from BL

JNLE NEXT	Jump to label NEXT if BL is not less than or equal to 39H

JGE / JNL (JUMP IF GREATER THAN OR EQUAL / JUMP IF NOT LESS THAN)

This instruction is usually used after a Compare instruction. The instruction will cause a jump to the label given in the instruction, if the sign flag is equal to the overflow flag.

CMP BL, 39H	Compare by subtracting 39H from BL
JGE NEXT	Jump to label NEXT if BL more positive than or equal to 39H
CMP BL, 39H	Compare by subtracting 39H from BL
JNL NEXT	Jump to label NEXT if BL not less than 39H

JL / JNGE (JUMP IF LESS THAN / JUMP IF NOT GREATER THAN OR EQUAL)

This instruction is usually used after a Compare instruction. The instruction will cause a jump to the label given in the instruction if the sign flag is not equal to the overflow flag.

```
CMP BL, 39H      Compare by subtracting 39H from BL
JL AGAIN         Jump to label AGAIN if BL more negative than 39H
CMP BL, 39H      Compare by subtracting 39H from BL
JNGE AGAIN       Jump to label AGAIN if BL not more positive than or equal to 39H
```

JLE / JNG (JUMP IF LESS THAN OR EQUAL / JUMP IF NOT GREATER)

This instruction is usually used after a Compare instruction. The instruction will cause a jump to the label given in the instruction if the zero flag is set, or if the sign flag not equal to the overflow flag.

```
CMP BL, 39H      Compare by subtracting 39H from BL
JLE NEXT         Jump to label NEXT if BL more negative than or equal to 39H
CMP BL, 39H      Compare by subtracting 39H from BL
JNG NEXT         Jump to label NEXT if BL not more positive than 39H
```

JE / JZ (JUMP IF EQUAL / JUMP IF ZERO)

This instruction is usually used after a Compare instruction. If the zero flag is set, then this instruction will cause a jump to the label given in the instruction.

```
CMP BX, DX       Compare (BX-DX)
JE DONE          Jump to DONE if BX = DX
IN AL, 30H       Read data from port 8FH
SUB AL, 30H      Subtract the minimum value.
JZ START         Jump to label START if the result of subtraction is 0
```

JNE / JNZ (JUMP NOT EQUAL / JUMP IF NOT ZERO)

This instruction is usually used after a Compare instruction. If the zero flag is 0, then this instruction will cause a jump to the label given in the instruction.

```
IN AL, 0F8H      Read data value from port
CMP AL, 72       Compare (AL –72)
JNE NEXT         Jump to label NEXT if AL ≠72
```

JS (JUMP IF SIGNED / JUMP IF NEGATIVE)

This instruction will cause a jump to the specified destination address if the sign flag is set. Since a 1 in the sign flag indicates a negative signed number, you can think of this instruction as saying “jump if negative”.

```
ADD BL, DH       Add signed byte in DH to signed byte in DL
JS NEXT          Jump to label NEXT if result of addition is negative number
```

JNS (JUMP IF NOT SIGNED / JUMP IF POSITIVE)

This instruction will cause a jump to the specified destination address if the sign flag is 0. Since a 0 in the sign flag indicate a positive signed number, you can think to this instruction as saying “jump if positive”.

```
DEC AL           Decrement AL
JNS NEXT         Jump to label NEXT if AL has not decremented to FFH
```

JP / JPE (JUMP IF PARITY / JUMP IF PARITY EVEN)

If the number of 1's left in the lower 8 bits of a data word after an instruction which affects the parity flag is even, then the parity flag will be set. If the parity flag is set, the JP / JPE instruction will cause a jump to the specified destination address.

```
IN AL, 0F8H     Read ASCII character from Port F8H
 OR AL, AL      Set flags
JPE ERROR       Odd parity expected, send error message if parity found even
```

JNP / JPO (JUMP IF NO PARITY / JUMP IF PARITY ODD)

If the number of 1's left in the lower 8 bits of a data word after an instruction which affects the parity flag is odd, then the parity flag is 0. The JNP / JPO instruction will cause a jump to the specified destination address, if the parity flag is 0.

```
IN AL, 0F8H     Read ASCII character from Port F8H
OR AL, AL       Set flags
JPO ERROR       Even parity expected, send error message if parity found odd
```

JO (JUMP IF OVERFLOW)

The overflow flag will be set if the magnitude of the result produced by some signed arithmetic operation is too large to fit in the destination register or memory location. The JO instruction will cause a jump to the destination given in the instruction, if the overflow flag is set.

```
ADD AL, BL      Add signed bytes in AL and BL
 JO ERROR       Jump to label ERROR if overflow from add
```

JNO (JUMP IF NO OVERFLOW)

The overflow flag will be set if some signed arithmetic operation is too large to fit in the destination register or memory location. The JNO instruction will cause a jump to the destination given in the instruction, if the overflow flag is not set.

```
ADD AL, BL      Add signed byte in AL and BL
 JNO DONE       Process DONE if no overflow
```

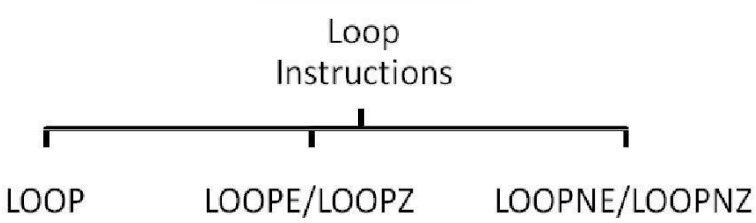
JCXZ (JUMP IF THE CX REGISTER IS ZERO)

This instruction will cause a jump to the label to a given in the instruction, if the CX register contains all 0's. The instruction does not look at the zero flag when it decides whether to jump or not.

```
        JCXZ SKIP
        SUB [BX], 07H
SKIP: ADD C
```

If CX = 0, skip the process Subtract 7 from data value Next instruction

f. LOOP INSTRUCTIONS



LOOP (JUMP TO SPECIFIED LABEL IF CX ≠0 AFTER AUTO DECREMENT)

This instruction is used to repeat a series of instructions some number of times. The number of times the instruction sequence is to be repeated is loaded into CX. Each time the LOOP instruction executes, CX is automatically decremented by 1.

```
MOV BX, [4000]          Point BX at first element in array
MOV CX, 40              Load CX with number of elements in array
NEXT: MOV AL, [BX]      Get element from array
INC AL                  Increment the content of AL
MOV [BX], AL            Put result back in array
INC BX                  Increment BX to point to next location
LOOP NEXT               Repeat until all elements adjuste
```

LOOPE / LOOPZ (LOOP WHILE CX ≠0 AND ZF = 1)

This instruction is used to repeat a group of instructions some number of times, or until the zero flag becomes 0. The number of times the instruction sequence is to be repeated is loaded into CX. Each time the LOOP instruction executes, CX is automatically decremented by 1. If CX ≠0 and ZF = 1, execution will jump to a destination specified by a label in the instruction. If CX = 0, execution simply go on the next instruction after LOOPE / LOOPZ.

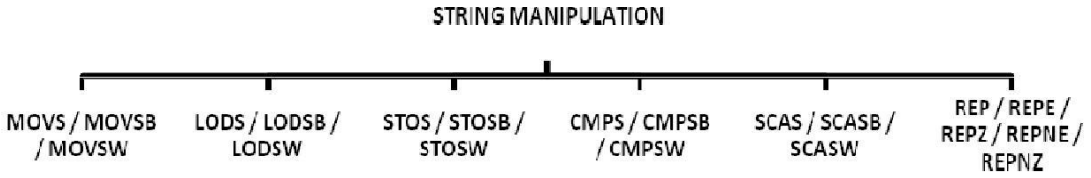
```
MOV BX, [4000]          Point BX to address before start of array
DEC BX                  Decrement BX
MOV CX, 100             Put number of array elements in CX
NEXT: INC BX            Point to next element in array
CMP [BX], OFFH          Compare array element with FFH
LOOPE NEXT
```

LOOPNE / LOOPNZ (LOOP WHILE CX ≠0 AND ZF = 0)

This instruction is used to repeat a group of instructions some number of times, or until the zero flag becomes a 1. The number of times the instruction sequence is to be repeated is loaded into the count register CX. Each time the LOOPNE / LOOPNZ instruction executes, CX is automatically decremented by 1. If CX ≠0 and ZF = 0, execution will jump to a destination specified by a label in the instruction. If CX = 0, after the auto decrement or if ZF = 1, executions simply go on the next instruction after LOOPNE / LOOPNZ.

```
MOV BX, [4000]          Point BX to adjust before start of array
DEC BX                  Decrement BX
MOV CX, 100             Put number of array in CX
NEXT: INC BX            Point to next element in array
CMP [BX], ODH           Compare array element with 0DH
LOOPNZ NEXT
```

g. STRING MANIPULATION INSTRUCTIONS



MOVS / MOVSB / MOVSW

This instruction copies a byte or a word from location in the data segment to a location in the extra segment. The offset of the source in the data segment must be in the SI register. The offset of the destination in the extra segment must be in the DI register. For multiple-byte or multiple-word moves, the number of elements to be moved is put in the CX register so that it can function as a counter. After the byte or a word is moved, SI and DI are automatically adjusted to point to the next source element and the next destination element. If DF is 0, then SI and DI will be incremented by 1 after a byte move and by 2 after a word move. If DF is 1, then SI and DI will be decremented by 1 after a byte move and by 2 after a word move.

MOV SI, 5000	Load 5000 into SI
OV DI, 6000	Load 6000 into DI
CLD	Clear DF to auto increment SI and DI after move
MOV CX, 04H	Load length of string into CX as counter
REP MOVSB	Move string byte until CX = 0

LODS / LODSB / LODSW (LOAD STRING BYTE INTO AL OR STRING WORD INTO AX)

This instruction copies a byte from a string location pointed to by SI to AL, or a word from a string location pointed to by SI to AX. If DF is 0, SI will be automatically incremented (by 1 for a byte string, and 2 for a word string) to point to the next element of the string. If DF is 1, SI will be automatically decremented (by 1 for a byte string, and 2 for a word string) to point to the previous element of the string. LODS does not affect any flag.

CLD	Clear direction flag so that SI is auto-incremented
MOV SI, OFFSET SOURCE	Point SI to start of string
LODS SOURCE	Copy a byte or a word from string to AL or AX

STOS / STOSB / STOSW (STORE STRING BYTE OR STRING WORD)

This instruction copies a byte from AL or a word from AX to a memory location in the extra segment pointed to by DI. In effect, it replaces a string element with a byte from AL or a word from AX. After the copy, DI is automatically incremented or decremented to point to next or previous element of the string. If DF is cleared, then DI will automatically incremented by 1 for a byte string and by 2 for a word string. If DI is set, DI will be automatically decremented by 1 for a byte string and by 2 for a word string.

MOV DI, OFFSET TARGET STOS TARGET

CMPS / CMPSB / CMPSW (COMPARE STRING BYTES OR STRING WORDS)

This instruction can be used to compare a byte / word in one string with a byte / word in another string. SI is used to hold the offset of the byte or word in the source string, and DI is used to hold the offset of the byte or word in the destination string.The AF, CF, OF, PF, SF, and ZF flags are affected by the comparison, but the two operands are not affected.

After the comparison, SI and DI will automatically be incremented or decremented to point to the next or previous element in the two strings. If DF is set, then SI and DI will automatically be decremented by 1 for a byte string and by 2 for a word string. If DF is reset, then SI and DI will automatically be incremented by 1 for byte strings and by 2 for word strings. The string pointed to by SI must be in the data segment. The string pointed to by DI must be in the extra segment.

The CMPS instruction can be used with a REPE or REPNE prefix to compare all the elements of a string.

```
MOV SI, 5000        Point SI to 5000
MOV DI, 6000        Point DI to 6000
CLD                 DF cleared; SI and DI will auto-increment after compare
MOV CX, 100         Put number of string elements in CX
REPE CMPSB          Repeat the comparison of string bytes until end of string
                    or until compared bytes are not equal
```

CX functions as a counter, which the REPE prefix will cause CX to be decremented after each compare. The B attached to CMPS tells the assembler that the strings are of type byte. If you want to tell the assembler that strings are of type word, write the instruction as CMPSW. The REPE CMPSW instruction will cause the pointers in SI and DI to be incremented by 2 after each compare, if the direction flag is set.

SCAS / SCASB / SCASW (SCAN A STRING BYTE OR A STRING WORD)

SCAS compares a byte in AL or a word in AX with a byte or a word in ES pointed to by DI. Therefore, the string to be scanned must be in the extra segment, and DI must contain the offset of the byte or the word to be compared. If DF is cleared, then DI will be incremented by 1 for byte strings and by 2 for word strings. If DF is set, then DI will be decremented by 1 for byte strings and by 2 for word strings. SCAS affects AF, CF, OF, PF, SF, and ZF, but it does not change either the operand in AL (AX) or the operand in the string. The following program segment scans a text string of 80 characters for a carriage return, 0DH, and puts the offset of string into DI:

```
MOV DI, OFFSET STRING
MOV AL, 0DH         Byte to be scanned for into AL
MOV CX, 80          CX used as element counter
CLD                 Clear DF, so that DI auto increments
REPNE SCAS STRING   Compare byte in string with byte in AL
```

REP / REPE / REPZ / REPNE / REPNZ (PREFIX)
(REPEAT STRING INSTRUCTION UNTIL SPECIFIED CONDITIONS EXIST)

REP is a prefix, which is written before one of the string instructions. It will cause the CX register to be decremented and the string instruction to be repeated until CX = 0. The instruction REP MOVSB, for example, will continue to copy string bytes until the number of bytes loaded into CX has been copied.

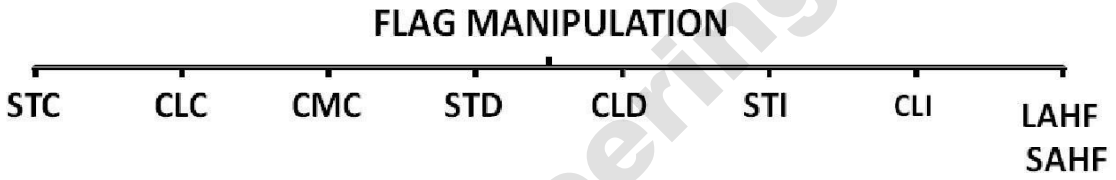
REPE and REPZ are two mnemonics for the same prefix. They stand for *repeat if equal* and *repeat if zero*, respectively. They are often used with the Compare String instruction or with the Scan String instruction. They will cause the string instruction to be repeated as long as the compared bytes or words are equal (ZF = 1) and CX is not yet counted down to zero. In other words, there are two conditions that will stop the repetition: CX = 0 or string bytes or words not equal.

REPE CMPSB Compare string bytes until end of string or until string bytes not equal

REPNE and REPNZ are also two mnemonics for the same prefix. They stand for *repeat if not equal* and *repeat if not zero*, respectively. They are often used with the Compare String instruction or with the Scan String instruction. They will cause the string instruction to be repeated as long as the compared bytes or words are not equal (ZF = 0) and CX is not yet counted down to zero.

REPNE SCASW Scan a string of word until a word in the string matches the word in AX or until all of the string has been scanned.

h. FLAG MANIPULATION INSTRUCTIONS



STC (SET CARRY FLAG) sets the carry flag to 1.
CLC (CLEAR CARRY FLAG) resets the carry flag to 0.
CMC (COMPLEMENT CARRY FLAG) complements the carry flag.
STD (SET DIRECTION FLAG) sets the direction flag to 1.
CLD (CLEAR DIRECTION FLAG) resets the direction flag to 0
STI (SET INTERRUPT FLAG) Setting the interrupt flag to a 1 enables the INTR interrupt input
CLI (CLEAR INTERRUPT FLAG) resets the interrupt flag to 0.
LAHF (COPY LOW BYTE OF FLAG REGISTER TO AH REGISTER)
The LAHF instruction copies the low-byte of the 8086 flag register to AH register.
SAHF (COPY AH REGISTER TO LOW BYTE OF FLAG REGISTER)
The SAHF instruction replaces the low-byte of the 8086 flag register with a byte from the AH register.

i. STACK RELATED INSTRUCTIONS

PUSH – PUSH Source
The PUSH instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment to which the stack pointer points.

PUSH BX Decrement SP by 2, copy BX to stack.

POP – POP Destination

The POP instruction copies a word from the stack location pointed to by the stack pointer to a destination specified in the instruction. After the word is copied to the specified destination, the stack pointer is automatically incremented by 2 to point to the next word on the stack

```
POP DX          Copy a word from top of stack to DX; increment SP by 2
```

PUSHF (PUSH FLAG REGISTER TO STACK)

The PUSHF instruction decrements the stack pointer by 2 and copies a word in the flag register to two memory locations in stack pointed to by the stack pointer.

POPF (POP WORD FROM TOP OF STACK TO FLAG REGISTER)

The POPF instruction copies a word from two memory locations at the top of the stack to the flag register and increments the stack pointer by 2.

j. INPUT-OUTPUT INSTRUCTIONS

IN – IN Accumulator, Port

The IN instruction copies data from a port to the AL or AX register. If an 8-bit port is read, the data will go to AL. If a 16-bit port is read, the data will go to AX.

```
IN AX, 34H          Input a word from port 34H to AX
```

For the variable-port form of the IN instruction, the port address is loaded into the DX register before the IN instruction. Since DX is a 16-bit register, the port address can be any number between 0000H and FFFFH. Therefore, up to 65,536 ports are addressable in this mode.

```
MOV DX, 0FF78H      Initialize DX to point to port
IN AL, DX           Input a byte from 8-bit port 0FF78H to AL
```

OUT – OUT Port, Accumulator

The OUT instruction copies a byte from AL or a word from AX to the specified port. The OUT instruction has two possible forms, fixed port and variable port. For the fixed port form, the 8-bit port address is specified directly in the instruction. With this form, any one of 256 possible ports can be addressed.

```
OUT 3BH, AL         Copy the content of AL to port 3BH
```

For variable port form of the OUT instruction, the content of AL or AX will be copied to the port at an address contained in DX. Therefore, the DX register must be loaded with the desired port address before this form of the OUT instruction is used.

```
MOV DX, 0FFF8H      Load desired port address in DX
OUT DX, AL          Copy content of AL to port FFF8H
```

k. MACHINE CONTROL INSTRUCTIONS

The Machine control instructions control the bus usage and execution

WAIT – Wait for Test input pin to go low.
HLT – Halt the process.
NOP – No operation.
ESC – Escape to external device like NDP
LOCK – Bus lock instruction prefix.

HLT (HALT PROCESSING)
The HLT instruction causes the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The different ways to get the processor out of the halt state are with an interrupt signal on the INTR pin, an interrupt signal on the NMI pin, or a reset signal on the RESET input.

NOP (PERFORM NO OPERATION)
This instruction simply uses up three clock cycles and increments the instruction pointer to point to the next instruction. The NOP instruction can be used to increase the delay of a delay loop. When hand coding, a NOP can also be used to hold a place in a program for an instruction that will be added later. NOP does not affect any flag.

ESC (ESCAPE)
This instruction is used to pass instructions to a coprocessor, such as the 8087 Math coprocessor, which shares the address and data bus with 8086. Instructions for the coprocessor are represented by a 6-bit code embedded in the ESC instruction. As the 8086 fetches instruction bytes, the coprocessor also fetches these bytes from the data bus and puts them in its queue

LOCK – ASSERT BUS LOCK SIGNAL
Many microcomputer systems contain several microprocessors. Each microprocessor has its own local buses and memory. Each microprocessor takes control of the system bus only when it needs to access some system resources. The LOCK prefix allows a microprocessor to make sure that another processor does not take control of the system bus while it is in the middle of a critical instruction, which uses the system bus

WAIT – WAIT FOR SIGNAL OR INTERRUPT SIGNAL
When this instruction is executed, the 8086 enters an idle condition in which it is doing no processing. The 8086 will stay in this idle state until the 8086 test input pin is made low or until an interrupt signal is received on the INTR or the NMI interrupt input pins. If a valid interrupt occurs while the 8086 is in this idle state, the 8086 will return to the idle state after the interrupt service procedure executes. It returns to the idle state because the address of the WAIT instruction is the address pushed on the stack when the 8086 responds to the interrupt request. WAIT does not affect any flag. The WAIT instruction is used to synchronize the 8086 with external hardware such as the 8087 Math coprocessor.

INT – INT TYPE

The term *type* in the instruction format refers to a number between 0 and 255, which identify the interrupt. When an 8086 executes an INT instruction, it will

1. Decrement the stack pointer by 2 and push the flags on to the stack.
2. Decrement the stack pointer by 2 and push the content of CS onto the stack.
3. Decrement the stack pointer by 2 and push the offset of the next instruction after the INT number instruction on the stack.
4. Get a new value for IP from an absolute memory address of 4 times the type specified in the instruction. For an INT 8 instruction, for example, the new IP will be read from address 00020H.
5. Get a new for value for CS from an absolute memory address of 4 times the type specified in the instruction plus 2, for an INT 8 instruction, for example, the new value of CS will be read from address 00022H.
6. Reset both IF and TF. Other flags are not affected.

INT 35New IP from 0008CH, new CS from 0008EH

INT 3This is a special form, which has the single-byte code of CCH; Many systems use this as a break point instruction

(Get new IP from 0000CH new CS from 0000EH).

INTO (INTERRUPT ON OVERFLOW)

If the overflow flag (OF) is set, this instruction causes the 8086 to do an indirect far call to a procedure you write to handle the overflow condition.

IRET (INTERRUPT RETURN)

When the 8086 responds to an interrupt signal or to an interrupt instruction, it pushes the flags, the current value of CS, and the current value of IP onto the stack. It then loads CS and IP with the starting address of the procedure, which you write for the response to that interrupt. The IRET instruction is used at the end of the interrupt service procedure to return execution to the interrupted program.

XLAT / XLATB – TRANSLATE A BYTE IN AL

The XLATB instruction is used to translate a byte from one code (8 bits or less) to another code (8 bits or less). The instruction replaces a byte in AL register with a byte pointed to by BX in a lookup table in the memory. Before the XLATB instruction can be executed, the lookup table containing the values for a new code must be put in memory, and the offset of the starting address of the lookup table must be loaded in BX. The code byte to be translated is put in AL. The XLATB instruction adds the byte in AL to the offset of the start of the table in BX. It then copies the byte from the address pointed to by (BX + AL) back into AL. XLATB instruction does not affect any flag.8086 routine to convert ASCII code byte to EBCDIC equivalent: ASCII code byte is in AL at the start, EBCDIC code in AL after conversion.

MOV BX, OFFSET EBCDIC Point BX to the start of EBCDIC table in DS

XLATB Replace ASCII in AL with EBCDIC from table.

Write an 8086 Assembly Language Program to Convert BCD data- Binary data.(6) [April/May 2015]

Program:

```
DATA    SEGMENT
   BCD   DB    17H
   BIN    DB     ?
DATA    ENDS
CODE    SEGMENT
ASSUME CS: CODE, DS: DATA
                MOV AX, DATA
                MOV DS, AX
                MOV AL, BCD
                MOV AH, BCD
                AND AH, 0FH
                MOV BL, AH
                AND AL, 0F0H
                MOV CL, 04H
                ROR   AL, CL
                MOV BH, 0AH
                MUL BH
                ADD AL,BL
                MOV BIN, AL
                MOV AX, 4C00H
                INT 21H
CODE            ENDS
```

RESULT: The Binary Number for the given BCD Number 17H is 11H .

Write a 8086 Assembly Language program to check whether the input string is palindrome or not.(8) [April/May 2015]

```
DATA SEGMENT
    STR1 DB "ENTER YOUR STRING HERE ->$"
    STR2 DB "YOUR STRING IS ->$"
    STR3 DB "REVERSE STRING IS ->$"
    INSTR1 DB 20 DUP("$")
    RSTR DB 20 DUP("$")
    NEWLINE DB 10,13,"$"
    N DB ?
    S DB ?
    MSG1 DB "STRING IS PALINDROME$"
    MSG2 DB "STRING IS NOT PALINDROME$"
    A DB "1"
DATA ENDS
CODE SEGMENT
    ASSUME DS:DATA,CS:CODE
START:  MOV AX,DATA
    MOV DS,AX
    LEA SI,INSTR1 ;GET STRING
```

```
    MOV AH,09H
    LEA DX,STR1
    INT  21H
    MOV AH,0AH
    MOV DX,SI
    INT  21H
    MOV AH,09H
    LEA DX,NEWLINE
    INT 21H     ;PRINT THE STRING
    MOV AH,09H
    LEA DX,STR2
    INT  21H
    MOV AH,09H
    LEA DX,INSTR1+2
    INT 21H
    MOV AH,09H
    LEA DX,NEWLINE
    INT 21H  ;PRINT THE REVERSE OF THE STRING
    MOV  AH,09H
    LEA DX,STR3
    INT 21H
    MOV CL,INSTR1+1
    ADD CL,1
    ADD SI,2
L1: INC SI
    CMP BYTE PTR[SI],"$"
    JNE L1
    DEC SI
    LEA DI,RSTR
L2: MOV  AL,BYTE  PTR[SI]
    MOV BYTE PTR[DI],AL
    DEC SI
    INC DI
    LOOP L2
    MOV AH,09H
    LEA DX,NEWLINE
    INT 21H
    MOV AH,09H
    LEA DX,RSTR
    INT 21H
    MOV AH,09H
    LEA DX,NEWLINE
    INT 21H          ;PRINT THE STRING IS PALINDROME OR NOT
    LEA SI,INSTR1
    LEA DI,RSTR
    MOV AH,09H
    LEA DX,NEWLINE
    INT 21H
    ADD SI,2
L7:  MOV BL,BYTE PTR[DI]
```

```
        CMP BYTE PTR[SI],BL
        JNE LL2
        INC SI
        INC DI
        MOV BL,BYTE PTR[DI]
        MOV AH,02H
        MOV DL,BL
        INT 21H
        MOV AH,09H
        LEA DX,NEWLINE
        INT 21H
        CMP BYTE PTR[DI],"$"
        JNE L7
        MOV AH,09H
        LEA DX,NEWLINE
        INT 21H
        MOV AH,09H
        LEA DX,MSG1
        INT 21H
        JMP L5
LL2:  MOV AH,09H
        LEA DX,NEWLINE
        INT 21H
        MOV AH,09H
        LEA DX,MSG2
        INT 21H
L5:  MOV AH,4CH
        INT 21H
CODE ENDS
END START
;OUTPUT:-
;Z:\SEM3\SS\21-30>P26
;ENTER YOUR STRING HERE ->MALAYALAM
;YOUR STRING IS ->MALAYALAM ;REVERSE
STRING IS -> ;MALAYALAM

;STRING IS PALINDROME
```

Write an 8086 assembly language program to multiply two 16-bit numbers to give 32-bit result. [Nov/Dec 2014]

```
MOV SI,1500
LODSW
MOV BX, AX
LODSW
MUL BX
MOV DI, 1520
MOV [DI], AX
INC DI
INC DI
MOV [DI], BX
```

4. ASSEMBLER DIRECTIVES AND OPERATORS

Explain any eight assembler directives of 8086 microprocessor. (8 Marks) **[April/May 2010]**

Explain the assembler directives ASSUME, EQU, DW and EVEN with suitable examples. [Marks 8] [**April/May 2011]**

Discuss About the Various Assembler Directive In 8086 Microprocessor Programming. (8) **[Nov / Dec 2012]**

What do you mean by assembler directives? Explain SEGEMENT, TYPE, OFFSET with suitable examples. (8) [**Nov /Dec 2013]**

Explain about the Assume, EQU, DD assembler directives.(8) [**Apr/May 2015**]

An assembler is a program used to convert an assembly language program into the equivalent machine code modules which may further be converted to executable codes. The assembler decides the address of each label and substitutes the values for each of the constants and variables. It then forms the machine code for the mnemonics and data in the assembly language program. While doing these things, the assembler may find out syntax errors.

The logical errors or other programming errors are not found out by the assembler. For completing all these tasks, an assembler needs some hints from the programmer, i.e. the required storage for a particular constant or a variable, logical names of the segments, types of the different routines and modules, end of file, etc. These, types of hints are given to the assembler using some predefined alphabetical strings called assembler directives. Assembler directives help the assembler to correctly understand the assembly language programs to prepare the codes.

DB: Define Byte
DW: Define Word
DQ: Define Quad Word
DT: Define Ten Bytes
EQU: Equate
ASSUME: Assume Logical Segment Name
END: END of Program
ENDP: END of Procedure
ENDS: END of Segment
EVEN: Align on Even Memory Address
EXTRN: External and PUBLIC: Public
GROUP: Group the Related segment
LABEL: Label
LENGTH: Byte Length of a Label
LOCAL
NAME: Logical Name of a Module
OFFSET: Offset of a Label
ORG: Origin
PROC: Procedure
PTR: Pointer
SEG: Segment of a Label
SEGMENT: Logical Segment
SHORT
TYPE
GLOBAL

DB: Define Byte

The DB directive is used to reserve byte or bytes of memory locations in the available memory.

LIST DB 0lH, 02H, 03H, 04H	Reserve four memory locations for a list named LIST and initialize them with the above specified four values.
MESSAGE DB 'GOOD MORNING'	Reserve the number of bytes of memory equal to the number of characters in the string named MESSAGE and initialize those locations by the ASCII equivalent of these characters.

DW: Define Word.
The DW directive makes the assembler reserve the number of memory words(16-bit).

WORDS DW 1234H, 4567H	Reserve two words in memory (4 bytes), and initialize the words with the specified values in the statements.
A DW 5 DUP (6666H)	Reserves five words, i.e. 10-bytes of memory for a word label A and initializes all the word locations with 6666H.

DQ: Define Quad word
Direct the assembler to reserve 4 words (8 bytes) of memory for the specified variable and may initialize it with the specified values.

DT: Define Ten Bytes.
Directs the assembler to define the specified variable requiring ten bytes for its storage and initialize the 10bytes with the specified values. The directive may be used in case of variables facing heavy numerical calculations, generally processed by numerical processors.

EQU: Equate
The directive EQU is used to assign a label with a value or a symbol.

```
LABEL    EQU    0500H
```

The first statement assigns the constant 500H with the label LABEL.

ASSUME: **Assume Logical Segment Name**
The ASSUME directive is used to inform the assembler, the names of the logical segments to be assumed for different segments used in the program. In the assembly language program, each segment is given a name. For example, the code segment may be given the name CODE; data segment may be given the name DATA etc.

```
ASSUME CS: CODE, DS: DATA
```

END: END of Program
The END directive marks the end of an assembly language program. When the assembler comes across this END directive, it ignores the source lines available later on. Hence, it should be ensured that the END statement should be the last statement in the file and should not appear in between.

ENDP: END of Procedure.

The ENDP directive is used to indicate the end of a procedure. A procedure is usually assigned a name, i.e. label. To mark the end of a particular procedure, the name of the procedure, i.e. label may appear as a prefix with the directive ENDP.

```
PROCEDURE STAR
.
.
STAR ENDP
```

ENDS: END of Segment

The logical segments are assigned with the names using the ASSUME directive. The names appear with the ENDS directive as prefixes to mark the end of those particular segments.

```
DATA SEGMENT
.
.
DATA ENDS
ASSUME CS: CODE, DS: DATA CODE SEGMENT.
.
.
CODE ENDS END
```

EVEN: Align on Even Memory Address

The assembler, while starting the assembling procedure of any program, initializes a location counter and goes on updating it, as the assembly proceeds. It goes on assigning the available addresses, i.e. the contents of the location counter, sequentially to the program variables, constants and modules as per their requirements, in the sequence in which they appear in the program.

The EVEN directive updates the location counter to the next even address if the current location counter contents are not even, and assigns the following routine or variable or constant to that address. The structure given below explains the directive.

```
EVEN PROCEDURE ROOT
.
ROOT ENDP
```

The above structure shows a procedure ROOT that is to be aligned at an even address.

EXTRN: External and PUBLIC: Public

The directive EXTRN informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language modules. While in the other module, where the names, procedures and labels actually appear, they must be declared public, using the PUBLIC directive.

To call a procedure FACTORIAL appearing in MODULE 1 from MODULE 2; in MODULE1, it must be declared PUBLIC using the statement PUBLIC FACTORIAL and in module 2, it must be declared external using the declaration EXTRN FACTORIAL. The

statement of declaration EXTRN must be accompanied by the SEGMENT and ENDS directives of the MODULE 1, before it is called in MOBULE 2. Thus the MODULE 1 and MODULE 2 must have the following declarations.

```
MODULEl     SEGMENT
PUBLIC      FACTORIAL FAR
MODULEl     ENDS
MODULE2     SEGMENT
EXTRN       FACTORIAL FAR
MODULE2     ENDS
```

GROUP: Group the Related segment

The directive is used to form logical groups of segments with similar purpose or type.

PROGRAM **GROUP** CODE, DATA, STACK

The above statement directs the loader/linker to prepare an EXE file such that CODE, DATA and STACK segment must lie within a 64kbyte memory segment that is named as PROGRAM. Now, for the ASSUME statement, one can use the label PROGRAM rather than CODE, DATA and STACK as shown.

ASSUME CS: PROGRAM, DS: PROGRAM, SS: PROGRAM.

LABEL: Label

The Label directive is used to assign a name to the current content of the location counter. At the start of the assembly process, the assembler initializes a location counter to keep track of memory locations assigned to the program. As the program assembly proceeds, the contents of the location counter are updated. During the assembly process, whenever the assembler comes across the LABEL directive, it assigns the declared label with the current contents of the location counter. The type of the label must be specified, i.e. whether it is a NEAR or a FAR label, BYTE or WORD label, etc.

A LABEL directive may be used to make a FAR jump as shown below. A FAR jump cannot be made at a normal label with a colon. The label CONTINUE can be used for a FAR jump, if the program contains the following statement.

CONTINUE LABEL FAR

The LABEL directive can be used to refer to the data segment along with the data type, byte or word as shown.

```
DATA SEGMENT
DATAS DB 50H DUP (?)
DATA-LAST LABEL BYTE FAR
DATA ENDS
```

After reserving 50H locations for DATAS, the next location will be assigned a label DATALAST and its type will be byte and far.

LENGTH: Byte Length of a Label

This is used to refer to the length of a data array or a string.

```
MOV CX, LENGTH ARRAY
```

This statement, when assembled, will substitute the length of the array ARRAY in bytes, in the instruction.

LOCAL

The label, variables, constants or procedures declared LOCAL in a module are to be used only by that module.

```
LOCAL a, b, DATA, ARRAY, ROUTINE
```

NAME: Logical Name of a Module

The NAME directive is used to assign a name to an assembly language program module. The module may now be referred to by its declared name. The names, if selected to be suggestive, may point out the functions of the different modules and hence may help in the documentation.

OFFSET: Offset of a Label

When the assembler comes across the OFFSET operator along with a label, it first computes the 16-bit displacement (also called as offset interchangeably) of the particular label, and replaces the string 'OFFSET LABEL' by the computed displacement.

```
CODE SEGMENT
        MOV SI, OFFSET LIST
CODE ENDS
DATA SEGMENT
        LIST DB 10H
DATA ENDS
```

ORG: Origin

The ORG directive directs the assembler to start the memory allotment for the particular segment, block or code from the declared address in the ORG statement While starting the assembly process for a module, the assembler initializes a location counter to keep track of the allotted addresses for the module. If the ORG statement is not written in the program, the location counter is initialized to 0000. If an ORG 200H statement is present at the starting of the code segment of that module, then the code will start from 200H address in code segment) In other words, the location counter will get initialized to the address 0200H instead of 0000H. Thus, the code for different modules and segments can be located in the available memory as required by the programmer. The ORG directive can even be used with data segments similarly.

PROC: Procedure

The PROC directive marks the start of a named procedure in the statement. Also, the types NEAR or FAR specify the type of the procedure, i.e whether it is to be called by the main program located within 64K of physical memory or not.

```
RESULT PROC NEAR
ROUTINE PROC FAR
```

PTR: Pointer

The pointer operator is used to declare the type of a label, variable or memory operand. The operator PTR is prefixed by either BYTE or WORD. If the prefix is BYTE, then the particular label, variable or memory operand is treated as an 8-bit quantity, while if WORD is the prefix, then it is treated as a 16-bit quantity. In other words, the PTR operator is used to specify the data type - byte or word.

MOV AL, BYTE PTR [SI]	Moves content of memory location addressed by SI (8-bit) to AL
INC BYTE PTR [BX]	Increments byte contents of memory location addressed by BX

SEG: Segment of a Label

The SEG operator is used to decide the segment address of the label, variable, or procedure and substitutes the segment base address in place of 'SEG label'. The example given below explains the use of SEG operator.

```
MOV AX, SEG ARRAY     This statement moves the segment address of ARRAY
MOV DS, AX            in which it is appearing, to register AX and then to DS.
```

SHORT

The SHORT operator indicates to the assembler that only one byte is required to code the displacement for a jump (i.e. displacement is within -128 to +127 bytes from the address of the byte next to the jump opcode).

```
JMP SHORT LABEL
```

TYPE

The TYPE operator directs the assembler to decide the data type of the specified label and replaces the 'TYPE label' by the decided data type. For the word type variable, the data type is2, for double word type, it is 4, and for byte type, it is 1. Suppose, the STRING is a word array. The instruction MOV AX, TYPE STRING moves the value 0002H in AX.

GLOBAL

The labels, variables, constants or procedures declared GLOBAL may be used by other modules of the program. Once a variable is declared GLOBAL, it can be used by any module in the program.

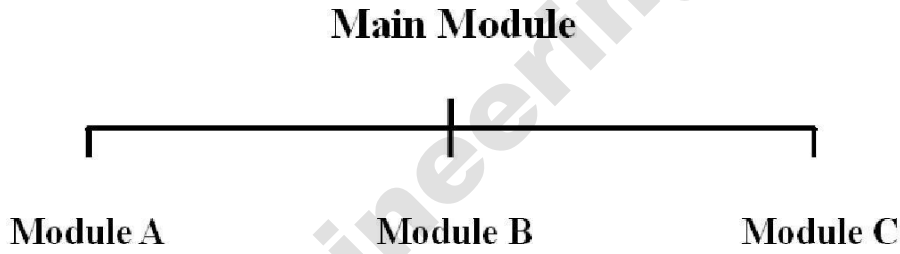
5a. MODULAR PROGRAMMING

Explain the concept of Modular Programming.

The formulation of complex programs from numerous small sequences called modules each of which performs a well-defined task. Such formulation of computer code is called **modular programming**. The various steps in development of assembly language program are,

1. Defining the overall work to be done by the program.
2. Breaking the overall program task into smaller task.
3. Determine the various communication/data exchange between tasks.
4. Writing assembly language code for each task called modules.
5. Testing each module separately.
6. Combine the modules into single program.
7. Testing and debugging the program.
8. Documenting the program.

The primary aid used in subdividing a program into modules is the hierarchical diagram which summarizes the relationships between the modules and submodules.



The main module corresponds to the president of the corporation, the Modules A, B & C corresponds to the Vice president and so on. The concept of modular programming refers to development of program codes in modules and merging the codes of various modules into single program code. When the program to be developed is too large to be developed by a single programmer, a team can be formed to develop the program. The overall task can be divided into number of smaller tasks and each smaller task can be developed as a module by a team member, and the modules can be integrated by the team leader to obtain the program for overall task.

The advantages of modular programming are,

1. Modules are easier to develop.
2. Modules can be developed independently by different programmers.
3. Debugging and testing of modules can be carried independently.
4. Any future modifications may be localized.
5. Repeated task can be developed as module and stored as subroutine/macro.
6. Common task can be developed as module and stored as library.
7. Documentation of modules can be made independently.

5b. LINKING AND RELOCATION

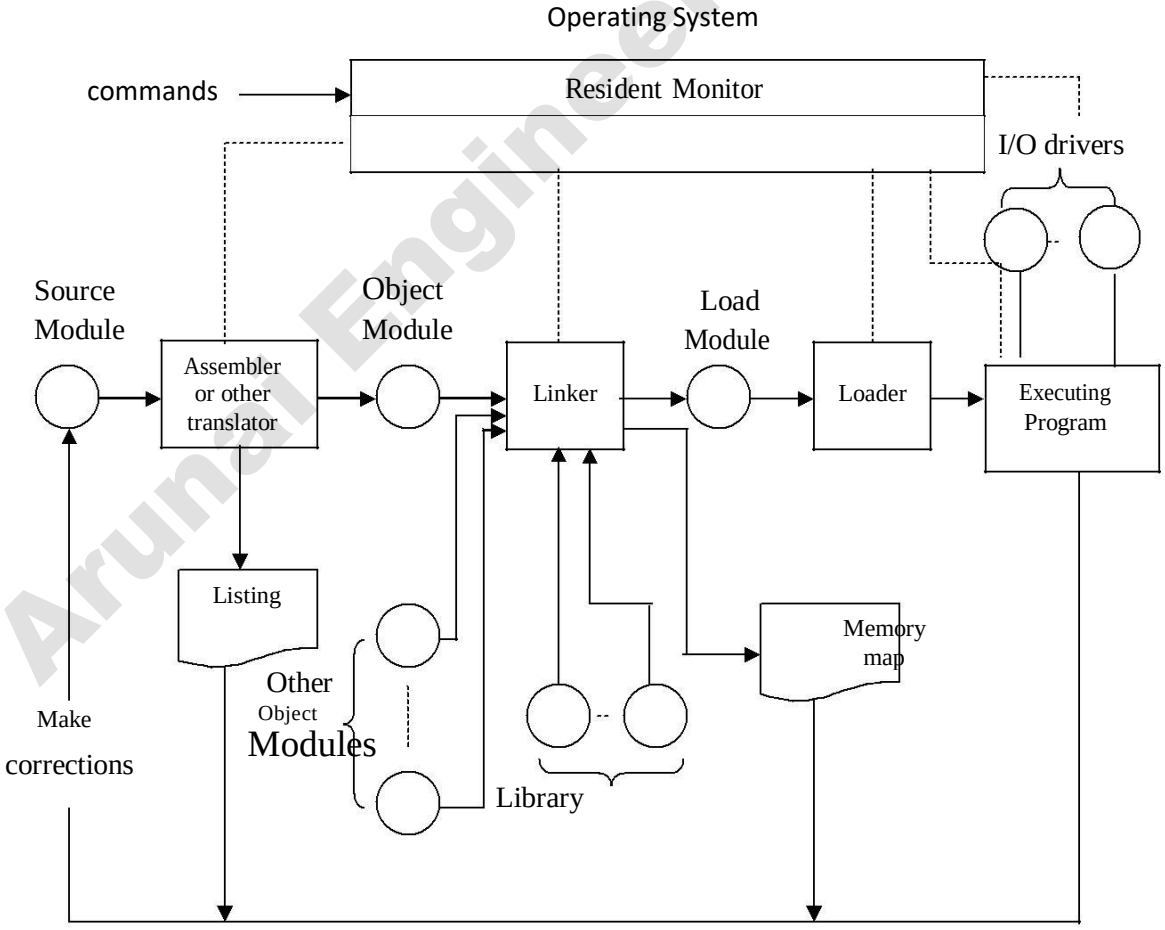
Describe the principle of linking and relocation.

The process of combining various program modules into single program is called linking and it is usually performed using a software tool called linker. The linker will generate a link file which contains the binary codes for all the combined modules.

Loader is a utility program which takes object code as input prepares it for execution and loads the executable code into the memory. Loader is actually responsible for initializing the process of execution.

Functions of Loaders

It allocates the space for program in the memory (**Allocation**)
It resolves the code between the object modules (**Linking**)
Some address dependent locations in the program, address constants must be adjusted according to allocated space (**Relocation**)
It also places all the machine instructions and data of corresponding programs and subroutines into the memory. (**Loading**)



Creation and Execution of a Program

In any event the resulting object modules, some of which are grouped into libraries must be linked together to form a load module before the program can be executed. In addition to outputting the load module normally the linker prints a memory map that indicates where the linked object modules will be loaded into memory, After the load module has been created it is loaded into the memory of the computer by the loader and execution begins. Although the I/O can be performed by modules within the program, normally the I/O is done by I/O drivers that are part of the operating system. All that appears in the user program are references to the I/O drivers that cause the operating system to execute them.

The linker/Loader must

Find the object modules to be linked

Construct the load module by assigning the positions of all the segments in all of the object modules being linked.

Fill in all offsets that could not be determined by the assembler

Fill in all segment addresses

Load the program for execution.

The object modules to be linked are determined by naming them in the command to the linker and by the operating system searching through libraries. The order in which the object modules appear in the linker command may determine the order in which they are stacked together to form the load module.

Segment combination

In addition to the linker commands, the assembler provides a means of regulating the way segments in different object modules are organized by the linker. Segments with same name are joined together by using the modifiers attached to the SEGMENT directives. SEGMENT directive may have the form

Segment name SEGMENT Combination-type

where the combine-type indicates how the segment is to be located within the load module. Segments that have different names cannot be combined and segments with the same name but no combine-type will cause a linker error. The possible combine-types are:

PUBLIC – If the segments in different modules have the same name and combine-type PUBLIC, then they are concatenated into a single element in the load module. The ordering in the concatenation is specified by the linker command.

COMMON – If the segments in different object modules have the same name and the combine-type is COMMON, then they are overlaid so that they have the same starting address. The length of the common segment is that of the longest segment being overlaid.

STACK – If segments in different object modules have the same name and the combine-type

STACK, then they becomes one segment whose length is the sum of the lengths of the individually specified segments. In effect, they are combined to form one large stack

AT – The AT combine-type is followed by an expression that evaluates to a constant which is to be the segment address. It allows the user to specify the exact location of the segment in memory.

MEMORY – This combine-type causes the segment to be placed at the last of the load module. If more than one segment with the MEMORY combine-type is being linked, only the first one will be treated as having the MEMORY combine type; the others will be overlaid as if they had COMMON combine-type.

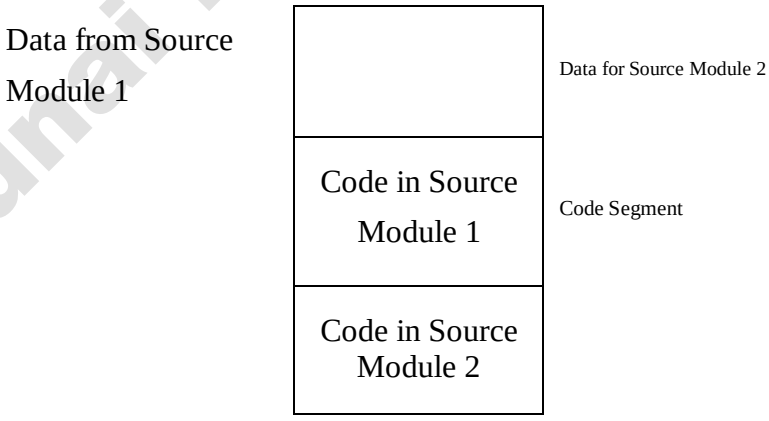
By causing two or more code segments to be put in a single segment the use of PUBLIC eliminates the need to change the contents of CS register as the program passes between sets of instructions within the code segment i.e. It allows intersegment branches to be replaced by intra segment branches. Data segments can be given the PUBLIC combine type to cause several sets of data to be combined into one larger set.

Source Module 1

```
DATA    SEGMENT   COMMON
          .
DATA    ENDS
CODE    SEGMENTPUBLIC
          .
CODE    ENDS
```

Source Module 2

```
DATA    SEGMENT   COMMON
          .
DATA    ENDS
CODE    SEGMENT   PUBLIC
          .
          .
CODE    ENDS
```



Access to External Identifiers

The variables and/or labels defined in the module itself are called **local (internal) identifiers)** relative to the module. However, if they are not defined in the module and defined in one of the other modules being linked, then they are called **external (global)**

Identifiers relative to the module.

In order for a linker to be able to access data or procedure in another assembly module correctly, use two assembly language directives: PUBLIC and EXTRN. Every address has two parts.

1. Offset address, 2. Segment address

The offset for local identifiers are inserted by the assembler. However, the offset for the external identifiers and all segment address are inserted by the linking process. Linking process determines the exact address for segment to be put in memory and then the address are assigned to segment. This process is known as **relocation.**

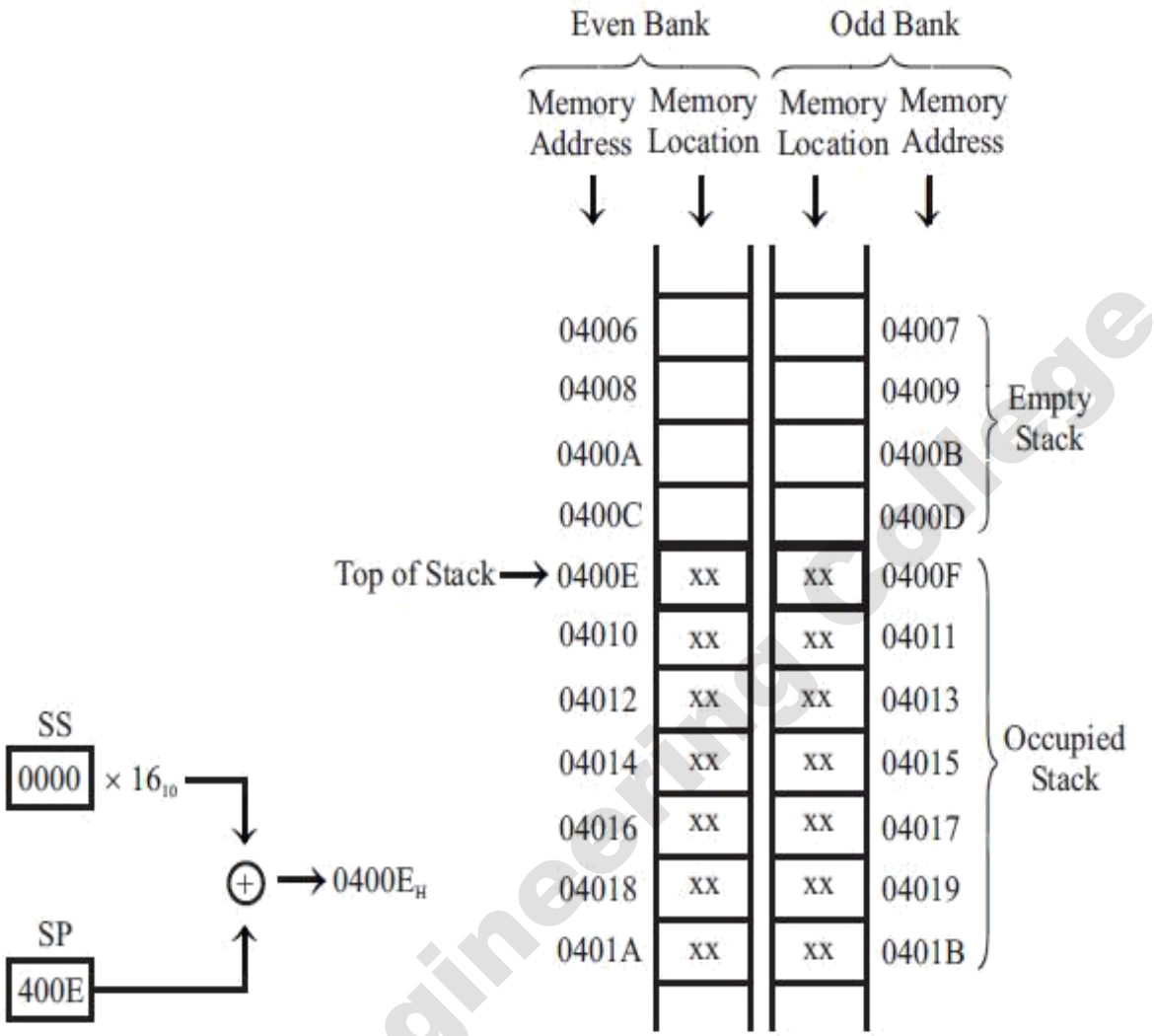
6a.STACKS

Explain Stack, procedure and macros in detail.

The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU. It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register. The stack segment, like any other segment, may have a memory block of a maximum of 64 Kbytes locations, and thus may overlap with any other segments. Stack Segment register (SS) contains the base address of the stack segment in the memory.

In 8086 microprocessor based system, the stack is created by loading a 16-bit base address in **S**tack **S**egment (SS) register and a 16-bit offset address in **S**tack **P**ointer (SP). The 20-bit physical address of the stack is computed by multiplying the contents of SS register by 16_{10} and then adding the contents of SP to this product. Here the content of SP is the offset address of the stack. Upon reset, the SS-register and SP are cleared to zero. For every write operation into stack, the SP is automatically decremented by two and for every read operation from stack; the SP is automatically incremented by two. The contents of SS register will not be altered while reading or writing into the stack.

In an 8086 processor, the content of the register can be stored in the stack using the PUSH instruction and the stored information can be retrieved back to the register using the POP instruction. when a number of registers have to be stored and retrieved in the stack, the order of retrieval should be reverse that of the order of the storageFor example, let BX be pushed to the stack first and DX next. When the stored information has to be retrieved to appropriate registers then the top of stack should be popped to DX first and then to BX next. The storage and retrieval in stack are in reverse order, because the SP is decremented for every write operation into the stack and SP is incremented for every read operation form the stack. Therefore the stack in an 8086 is called **L**ast- **I**n-**F**irst-**O**ut (LIFO) stack, i.e., the last stored information can be read first. A typical example of stack in 8086 is shown in the figure



6b. PROCEDURES

When a group of instructions are to be used several times to perform a same function in a program, then we can write them as a separate subprogram called procedure or subroutine. Whenever required the procedures can be called in a program using CALL instructions. The procedures are written and assembled as separate program modules and stored in memory. When a procedure is called in the main program, the program control is transferred to procedure and after executing the procedure the program control is transferred back to the main program. In an 8086 processor, the instruction CALL is used to call a procedure in the main program and the instruction RET is used to return the control to the main program.

The 8086 processor has two types of call instructions and they are intrasegment call or near call (call within a segment) and intersegment call or far call (call outside a segment). A procedure can be called using near call instruction if it is stored in the same segment where the main program is also stored. A procedure can be called using far call instruction if the procedure and main program are stored in different memory segments. The procedures are terminated with RET instructions. The 8086 has two types of RET instructions and they are

near return and far return. The near return instruction is used to terminate a procedure stored in the same segment. The far return instruction is used to terminate a procedure stored in a different segment.

When a procedure is called by using far call instruction, the 8086 processor will push the contents of IP and CS-register in stack and the segment base address of procedure is loaded in CS register and the effective address of procedure is loaded in IP. Now the program control is transferred to procedure stored in another segment and so the processor will start executing the instructions of the procedure. At the end of procedure, RET instruction is encountered. On executing the RET instruction, the top of stack (which is the previous stored value) is popped to CS register and IP. Thus the program control is returned to main program.

When a procedure is called by using near call instruction, the 8086 processor will push the contents of IP alone in stack and the effective address of procedure is loaded in IP. Here the content of CS register is not altered. Now the program control is transferred to procedure stored in same segment and so the processor will start executing the instructions of the procedures. At the end of procedure, RET instruction is encountered. On executing the RET instruction, the top of stack (which is the previous stored value) is popped to IP. Thus, the program control is returned to main program.

The main advantage of using a procedure is that the machine codes for the group of instructions in the procedure have to be put in memory only once. The disadvantages of using the procedure are the need for a stack, and the overhead time required to call the procedure and return to the calling program.

Disadvantages of Procedure

1. Linkage associated with them.
2. It sometimes requires more code to program the linkage than is needed to perform the task. If this is the case, a procedure may not save memory and execution time is considerably increased.

Hence a means is needed for providing the programming ease of a procedure while avoiding the linkage. This need is fulfilled by **Macros**

6c. MACROS

How does one define and call macro parameters of 8086 microprocessor? (4 Marks)
[April/May 2010]

When a group of instructions are to be used several times to perform a same function in a program and they are too small to be written as a procedure, then they can be defined as a macro. Macro is a small group of instructions enclosed by the assembler directives MACRO and ENDM. Macros are identified by their name and usually defined at the start of a program.

The macro is called by its name in the program. Whenever a macro is called in a program, the assembler will insert the defined group of instructions in place of the call. In other words, the macro call is like shorthand expression which tells the assembler, **“** Every time you see a macro name in the program, replace it with the group of instructions defined as macro**”** . Actually the assembler generates machine codes for the group of instructions

defined as macro, whenever it is called in the program. The process of replacing the macro with the instructions it represent is called expanding the macro. Hence, macros are also known as open subroutines because they get expanded at the point of macro invocation.

When macros are used, the generated machine codes are right-in-line with the rest of the program and so the processor does not have to go off to a procedure call and return. This results in avoiding the overhead time involved in calling and returning from a procedure. The disadvantage of using macro is that the program may take up more memory due to insertion of the machine codes in the program at the place of macros. Hence, the macros should be used only when its body has a few program statements.

ASM-86 Macro Facilities

The macro definition is constructed as follows:

```
%*DEFINE (Macro name (Dummy parameter
          list))(
                    Prototype code
          )
```

Macro name has to begin with a letter and can contain letters, numbers and underscore characters. Dummy parameters in the parameter list should be separated by commas. Each dummy parameter appearing in the prototype code should be preceded by a % character. Consider an example that shows the definition of macro for multiplying 2 word operands and storing the result which does not exceed 16 bit. A macro call has the form

%Macro name (Actual parameter list)

With the actual parameters being separated by commas.

%MULTIPLY (CX, VAR, XYZ [BX]

Above macro call results in following set of codes.

```
PUSH DX
PUSH AX
MOV AX,CX
IMUL VAR
MOV XYZ [BX],
AXPOP AX
POP DX
```

It is possible to define a macro with no dummy parameters, but in this case the call must not include any parameters. Consider a macro for pushing the contents at beginning of a procedure.

Macro definition consists of

```
%*DEFINE (SAVEREG)
        (PUSH  AX
         PUSH  BX
         PUSH  CX
         PUSH DX
        )
```

This macro is called using the statement

%SAVEREG

The above macro can be called at the beginning of the each procedure to save the register contents. A similar macro could be used to restore the register contents at the end of each procedure.

```
%*DEFINE (RESTORE)
    ( POP DX
      POP CX
      POP BX
      POP AX )
```

7. INTERRUPTS AND INTERRUPT ROUTINES

Discuss the interrupts types of 8086 microprocessor. [Marks 8] [**April/May 2011, April/May2017**]

How the interrupt vector is handled in 8086? (8) [**Nov /Dec 2013**]

Draw and discuss the interrupt structure of 8086. (16)[**May/Jun 2014**]

Describe the conditions which cause the 8086 to perform type 0 and type 1 interrupt.(8) [**Nov/Dec 2014**].

Explain briefly about Interrupt handling process in 8086.(8) [**Apr/May 2015**]

Interrupt and its Need

The microprocessors allow normal program execution to be interrupted in order to carry out a specific task/work. The processor can be interrupted in the following ways

1. By an external signal generated by a peripheral,
2. By an internal signal generated by a special instruction in the program,
3. By an internal signal generated due to an exceptional condition (divide by zero)

Interrupt: The process of interrupting the normal program execution to carry out a specific task/work.

The interrupt is initiated by a signal generated by an external device or by a signal generated internal by the processor. When a microprocessor receives an interrupt signal it stops executing current normal program, save the status (or content) of various registers (IP, CS and flag registers in case of 8086) in stack and then the processor executes a subroutine/procedure in order to perform the specific task/work requested by the interrupt. The subroutine/procedure that is executed in response to an interrupt is also called **Interrupt Service Subroutine**. (ISR). At the end of ISR, the stored status of registers in stack is restored to respective registers, and the processor resumes the normal program execution from the point (instruction) where it was interrupted.

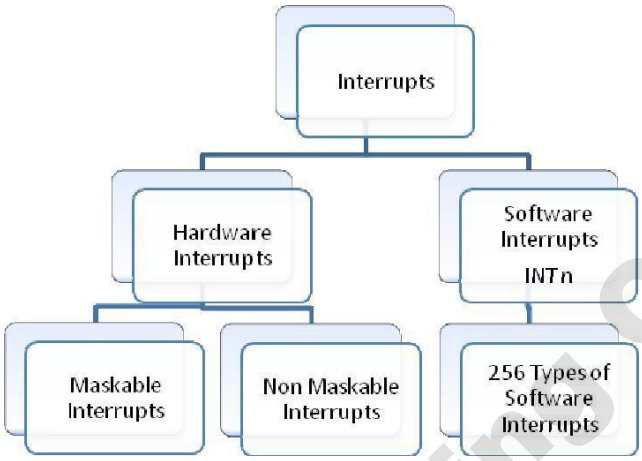
Classification of Interrupts

In general the interrupts can be classified in the following three ways :

1. Hardware and software interrupts
2. Vectored and Non Vectored interrupt
3. Maskable and Non Maskable interrupts.

Hardware and Software Interrupts

The interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor is called hardware interrupt. The 8086 processor has two interrupt pins INTR and NMI. The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if software interrupt instruction is encountered then the processor initiates an interrupt. The 8086 processor has 256 types of software interrupts. The software interrupt instruction is INT n, where n is the type number in the range 0 to 255.



Vectored and Non Vectored Interrupt

When an interrupt signal is accepted by the processor, if the program control automatically branches to a specific address (called vector address) then the interrupt is called **vectored interrupt**. The automatic branching to vector address is predefined by the manufacturer of processors. (In these vector addresses the interrupt service subroutines (ISR) are stored). In **non-vectored interrupts** the interrupting device should supply the address of the ISR to be executed in response to the interrupt. All the 8086 interrupts are vectored interrupts. The vector address for an 8086 interrupt is obtained from a vector table implemented in the first 1kb memory space (00000h to 03FFFh).

Maskable and Non Maskable Interrupts

The interrupts whose request can be either accepted or rejected by the processor are called maskable interrupts. The interrupts whose request has to be definitely accepted (or cannot be rejected) by the processor are called non-maskable interrupts. Whenever a request is made by non-maskable interrupt, the processor has to definitely accept that request and service that interrupt by suspending its current program and executing an ISR. In 8086 processor all the hardware interrupts initiated through INTR pin are maskable by clearing interrupt flag (IF). The interrupt initiated through NMI pin and all software interrupts are non-maskable.

Sources of Interrupts in 8086

An interrupt in 8086 can come from one of the following three sources.

1. One source is from an external signal applied to NMI or INTR input pin of the processor. The interrupts initiated by applying appropriate signals to these input pins are called **hardware interrupts.**
2. A second source of an interrupt is execution of the interrupt instruction "INT n", where n is the type number. The interrupts initiated by "INT n" instructions are called **software interrupts.**
3. The third source of an interrupt is from some condition produced in the 8086 by the execution of an instruction. An example of this type of interrupt is divide by zero interrupt. Program execution will be automatically interrupted if you attempt to divide an operand by zero. Such conditional interrupts are also known as exceptions.

Interrupts of 8086

The 8086 microprocessor has 256 types of interrupts. INTEL has assigned a type number to each interrupt. The type numbers are in the range of 0 to 255. The 8086 processor has dual facility of initiating these 256 interrupts. The interrupts can be initiated either by executing "INT n" instruction where n is the type number or the interrupt can be initiated by sending an appropriate signal to INTR input pin of the processor. For the interrupts initiated by software instruction" INT n ", the type number is specified by the instruction itself.

Classification of Interrupts of 8086

Predefined (Or Dedicated)
InterruptsSoftware Interrupts Of
8086 Hardware Interrupts Of 8086

Predefined (Or Dedicated) Interrupts

1. Division by zero (Type-0 interrupt).
2. Single step (Type-1 interrupt).
3. Non maskable interrupt, NMI (Type-2 interrupt).
4. Break Point interrupt (Type-3 interrupt).
5. Interrupt on overflow (Type-4 interrupt).

The predefined interrupts are only defined by INTEL and INTEL has not provided any subroutine/procedure to be executed for these interrupts. To use the predefined interrupts the user/ system designer has to write **I**nterrupt **S**ervice **S**ubroutine (ISS) for each interrupt and store them in memory. The corresponding address of the ISS should be stored in interrupt vector table. If a predefined interrupt is not used in a system then the user may assign some other functions to these interrupts.

Divide by Zero Interrupt (type-0 interrupt)

Type-0 interrupt is implemented by INTEL as a part of the execution of the divide instruction. The 8086 will automatically do a type-0 interrupt if the result of a division

operation is too large to fit in the destination register and this interrupt is nonmaskable. Since the type-0 interrupt cannot be disabled in any way, we have to account for it in the programs using divide instructions. To account for this, we have to write an ISS which takes the desired action or indicate error condition when an invalid division occurs. The ISS should be stored in memory and the address of ISS is stored in interrupt vector table.

Single Step Interrupt (type-1 interrupt)

When the **T**rap/**T**race **F**lag (TF) is set to one, the 8086 processor will automatically generate a type-1 interrupt after execution of each instruction. The user can write an ISS for type-1 interrupt to halt the processor temporarily and return the control to the user so that after execution of each instruction, the processor status (content of register/memory) can be verified. If they are correct then we can proceed to execute the next instruction. Execution of one instruction by one instruction is known as single step and this feature will be useful to debug a program.

Nonmaskable Interrupt, NMI (type-2 interrupt)

The 8086 processor will automatically generate a type-2 interrupt when it receives a **low-to-high** transition on its NMI input pin. This interrupt cannot be disabled or masked. Usually, the type-2 interrupt is used to save program data or processor status in case of system ac power failure. The ac power failure is detected by an external hardware and whenever the ac power fails, the external hardware will send an interrupt signal to the NMI input pin of the processor.

Breakpoint interrupt (type-3 interrupt)

Type-3 interrupt is used to implement a breakpoint function, which executes a program partly or up to the desired point and then return the control to the user.

The breakpoint interrupt is initiated by the execution of "INT 3" instructions. To implement the breakpoint function the system designer has to write an ISS for type-3, which takes care of displaying a message and return the control to the user whenever type-3 interrupt is initiated.

This interrupt will be useful to debug a program by executing the program part by part. The user can insert "INT 3" instruction at the desired location and execute the program. Whenever "INT 3" instruction is encountered, the processor halts the program execution and return the control to the user. Now the user can verify the processor status (contents of register/memory). If they are correct then the user can proceed to execute next part of the program.

Overflow Interrupt (type-4 interrupt)

In the 8086 processor, the **O**verflow **F**lag (OF) will be set if the signed arithmetic operation generates a result whose size is larger than the size of the destination register/memory. During such conditions, the type-4 interrupt can be used to indicate an error condition. The type-4 interrupt is initiated by "IN TO" instruction.

One way of detecting the overflow error is to put the INTO instruction immediately after the arithmetic instruction in the program. After arithmetic operation if the overflow flag

is not set then the processor will consider "INTO" instruction as NOP (**N**o **op**eration). However, if the overflow flag is set then the 8086 will generate a type-4 interrupt, which executes an ISS to indicate overflow condition.

SOFTWARE INTERRUPTS OF 8086

The "INT n" instructions are called software interrupts. The "INT n" instruction will initiate type-n interrupt, and the value of n is in the range of 0 to 255. Therefore, all the 256 type interrupts including the INTEL predefined and reserved interrupts can be initiated through "INT n" instruction. The software interrupts are non-maskable and has higher priority than hardware interrupts.

HARDWARE INTERRUPTS OF 8086

The interrupts initiated by applying appropriate signals to INTR and NMI pins of 8086 are called hardware interrupts. All the 256 types of interrupts including INTEL predefined and reserved interrupts can be initiated by applying a **high** signal to INTR pin of 8086. When a **high** signal is applied to the INTR pin and the hardware interrupt is enabled/unmasked, then the processor runs an interrupt acknowledge cycle to get the type number of the interrupt from the device which sends the interrupt signal. The interrupting device can send a type number in the range of 0 to 25510. Therefore, all the 256 types of interrupts can be initiated through INTR pin.

The hardware interrupts initiated through INTR are maskable by clearing the **I**nterrupt **F**lag (IF), i.e., the hardware interrupts are masked/disabled when IF = 0 and they are unmasked/enabled when IF = 1. The interrupts initiated through INTR has lower priority than software interrupts.

The hardware interrupt NMI is nonmaskable and has higher priority than interrupts initiated through INTR. The NMI is initiated by a rising edge (or low-to-high transition) of the signal applied to NMI pin of the processor. The processor will execute type-2 interrupt in response to interrupt initiated through NMI pin and this type number is fixed by INTEL. The external device, interrupting the processor through NMI pin, need not supply the type number for this interrupt.

PRIORITIES OF INTERRUPTS OF 8086

The 8086 processor checks for internal interrupts before it checks for any hardware interrupt. Therefore, software interrupts has higher priority than hardware interrupts. But the processor can accept the NMI interrupt request and execute a procedure for it even in between the execution of procedure for higher priority interrupt.

.

Interrupt	**Priority**
Divide error, INT n, INTO	Highest
NMI	
INTR	
SINGLE STEP	Lowest

Interrupt Vector Table

If a division by 0 is attempted, the processor will push the current contents of the PSW, CS and IP into the stack, fill the IP and CS registers from the addresses 00000 to 00003, and continue executing at the address indicated by the new contents of IP and CS.

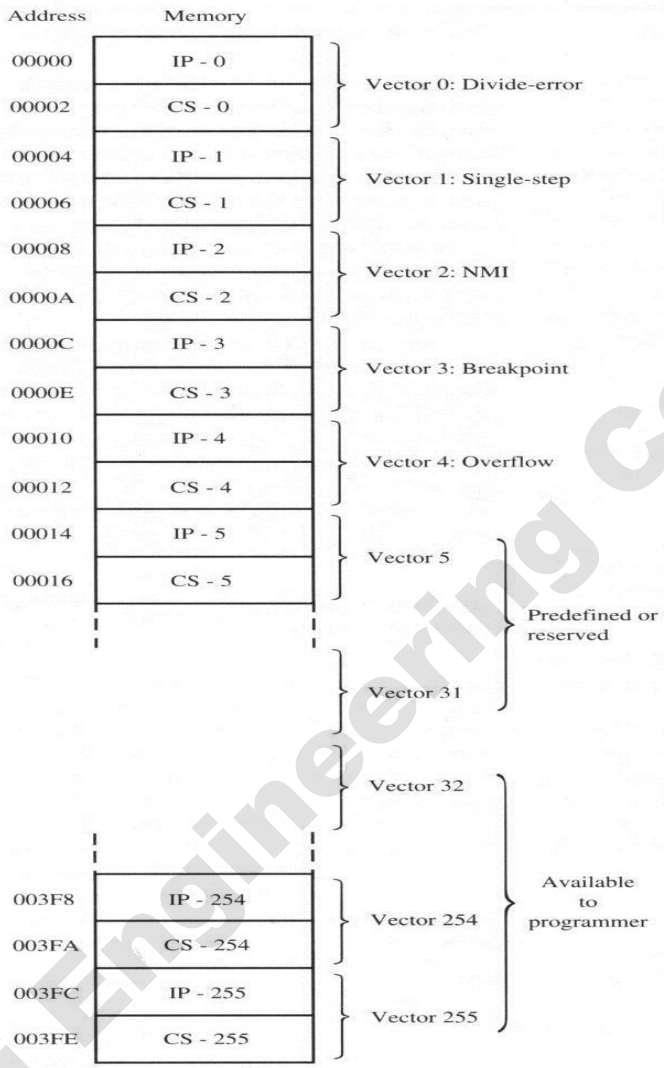


Figure 1.3 Organisation of Interrupt Vector Table in 8086

SERVICING AN INTERRUPT BY 8086

The 8086 processor checks for interrupt request at the end of each instruction cycle. If an interrupt request is deducted, then the 8086 processor responds to the interrupt by performing the following operations:

1 The SP is decremented by two and the content of flag register is pushed to stack memory.
2 The interrupt system is disabled by clearing Interrupt Flag (IF).
3 The single-step trap flag is disabled by clearing Trap Flag (TF).

4 The stack pointer is decremented by two and the content of CS-register is pushed to stack memory.

5 Again, the stack pointer is decremented by two and the content of IP is pushed to stack memory.

6 In case of hardware interrupt through INTR, the processor runs an interrupt acknowledge cycle to get the interrupt type number. For software interrupts, the type number is specified in the instruction itself. For NMI and exceptions the type number is defined by INTEL.

7 The processor generates a 20-bit memory address by multiplying the type number by four and sign extending it to 20-bit. This memory address is the address of the interrupt vector table, where the vector address of the Interrupt Service Routine (ISR) is stored by the user/system designer.

8 The first word pointed by vector table address is loaded in IP and the next word is loaded in CS-register. Now the content of the IP is the offset address and the content of the CS-register is the segment base address of the ISRS to be executed

9 The 20-bit physical memory address of ISS is calculated by multiplying the content of

10 The processor executes the ISR to service the interrupt.

11 The ISS will be terminated by the IRET instruction. When this instruction is executed, the top of stack is popped to IP, CS and flag register one word by one word. After every pop operation, the SP is incremented by two.

12 Thus, at the end of ISR, the previous status of the processor is restored and so the processor will resume the execution of normal program from the instruction where it was suspended.

UNIT II 8086 SYSTEM BUS STRUCTURE

8086 signals – Basic configurations – System bus timing –System design using 8086 – IO programming – Introduction to Multiprogramming – System Bus Structure – Multiprocessor configurations – Coprocessor, Closely coupled and loosely Coupled configurations – Introduction to advanced processors.

PART -A (2 MARKS)

1. Define bus. Why bus request and cycle stealing are required. [Apr/May 2015]

The microprocessors functions as the CPU in the stored program model of the digital computer. Its job is to generate all system timing signals and synchronize the transfer of data between memory, I/O, and itself. It accomplishes this task via the three-bus system architecture named as **address bus, data bus and control bus.**

The **cycle stealing** mode is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using BR **(Bus Request)** and BG (**Bus Grant**) signals, which are the two signals controlling the interface between the CPU and the DMA controller. However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is deserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred.

2. What are the advantages of coprocessor? [May/Jun 2014]

- A coprocessor is a special set of circuit in a microprocessor chip that is designed to manipulate numbers or perform some other specialized function more quickly than the basic microprocessor circuits could perform the same task.
- The coprocessor, also known as a math coprocessor, numeric coprocessor, or floating-point unit (FPU), became a physical part of the microprocessor chip. Some coprocessors are still available as separate chips or circuit cards. These are designed for specific applications such as high-end graphics, broadband signal processing, and encryption / decryption.
- Coprocessors of this type make it possible to customize the various models in a line of personal or business computers.

3. What are the significance of Bus High Enable Signal? [Apr/May2015]

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. The S7 status information is available during T2, T3 and T4 states. The signal is active Low and floats to 3-state during "hold" state. This pin is Low during T1 state for the first interrupt acknowledge cycle.

4. What is meant by a loosely coupled configuration? [May/Jun 2014]

A loosely coupled configuration provides the following advantages:

- High system throughput can be achieved by having more than one CPU.
- The system can be expanded in a modular form. Each bus master module is an independent unit and normally resides on a separate PC board. Therefore, a bus master module can be added or removed without affecting the other modules in the system.
- A failure in one module normally does not cause a breakdown of the entire system and the faulty module can be easily detected and replaced.
- Each bus master may have a local bus to access dedicated memory or I/O devices so that a greater degree of parallel processing can be achieved. More than one bus master module may have access to the shared system bus.

5. What is multiprogramming? [Apr/May 2017]

Two or more processes code is stored in memory at the same time and is executed in a time multiplexed manner that system is called as multiprogramming.

6. Justify the need for coprocessor. [Apr/May 2015]

- Coprocessor cannot take control of the bus, it does everything through the CPU.
- When the CPU executes the ESC instruction, the processor accesses the memory operand by placing the address on the address bus. If a coprocessor is configured to share the system bus, it will recognize the ESC instruction and therefore will get the opcode and the operand.

7. How many memory locations can be addressed by 8086 microprocessor? [Nov/Dec 2014]

It has a 20-bit address bus can access upto 2^{20} memory locations (1 MB)

8. In what ways are the standard microprocessor and coprocessor differ from each other? [Nov/Dec 2012]

The processor takes care of all the major processing and the co-processor or the auxiliary processor unit takes care of some other things like arithmetic calculations or graphics to allow the main processor to work on more difficult tasks.

A co-processor is a unique set of circuit. It is used in enhancing the functions of the primary processor. It is intended to direct the performance and the functions of the microprocessor. It has a quick performance than the primary processor.

9. How does the main processor distinguish its instructions from the co-processor instructions when it fetches the instructions from memory? [Nov/Dec 2012]

ESC instruction is used to differentiate the processor and co-processor instruction When it fetches the instruction from memory.

10. Compare Closely Coupled configuration with loosely Coupled Configuration. [Nov/Dec 2014]

Closely Coupled	Loosely Coupled
Contains multiple CPUs that are connected at the bus level. These CPU may have access to a central shared memory or may participate in a memory hierarchy with local and shared memory	These are based on multiple standalone signal or dual processor interconnected via a high speed communication system
They perform better and are physically smaller than loosely coupled system.	Opposite to closely coupled.
More expensive.	Less Expensive.
The delay experienced is short, data rate is high, and number of bits transferred per second is large.	Delay is large, data rate is low.

11. What is the floating point coprocessor? [Nov/Dec 2013]

8086 processor do not have instruction set for performing floating point arithmetic operations, but the combination of this processor with the coprocessor 8087 can perform any application which heavily involves floating point calculation, such combination is called floating point coprocessor.

12. Differentiate between minimum and maximum mode (April/May2010)

	Minimum mode	Maximum mode
i.	A processor is in minimum mode when MN/MX pin is strapped to 15V.	A processor is n maximum mode when MN/MX is grounded.
ii.	All the control signals are given out by microprocessor chip itself.	The processor derive the status signals S2 , S1 and S0. Another chip called bus controller derives control signals using this status information.
iii.	There is a single microprocessor.	There may be more than one microprocessor.

13. Give any four pin definitions for the minimum mode. (Nov/Dec2008)

Symbol		Description
i.	INTA	Indicates recognition of an interrupt request. Consists of two negative going pulses in two consecutive bus cycles.
ii.	ALE	Outputs a pulse at the beginning of the bus cycle and to indicate an address available on address pins.
iii.	HLDA	Outputs a bus grant to a requesting master.
iv.	HOLD	Receives bus requests from bus masters.

14. What are the pins that are used to indicate the type of transfer in minimum mode?

The M/IO, RD, WR lines specify the type of transfer. It is indicated in the following table:

M/IO	RD	WR	
0	0	1	I/O Read
0	1	0	I/O Write
1	0	1	Memory read
1	1	0	Memory write.

15. What are the functional parts of 8086 CPU?

The two independent functional parts of the 8086 CPU are:

i. **Bus Interface Unit (BIU):** BIU sends out addresses, fetches instruction from memory, reads data from ports and memory and writes data to ports and memory.

ii. **Execution Unit (EU):** EU tells the BIU where to fetch instructions or data, decodes instructions and executes instructions.

16. What is the operation of S0, S1 and S2 pins in maximum mode?

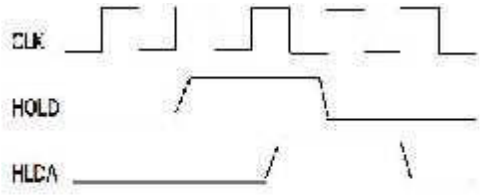
S2, S1, S0 indicates the type of transfer to take place during the current bus cycle.

S_2	S_1	S_0		
0	0	0	-	Interrupt acknowledge
0	0	1	-	Read I/O port
0	1	0	-	Write I/O port
0	1	1	-	Halt
1	0	0	-	Instruction fetch
1	0	1	-	Read Memory
1	1	0	-	Write Memory
1	1	1	-	Inactive.

17. Give any four pin definitions for maximum mode.

Symbol	Description
QS1, QS0	Reflects the status of the instruction queue. This status indicates the activity in the queue during the previous clock cycle.
$\overline{LOCK}$	Indicates that the bus is not to be relinquished to other potential bus masters.
$\overline{RQ}$/GT1	For inputting bus requests and outputting bus grants.
$\overline{RQ}$/GT0	Same as $\overline{RQ}$/GT1 except that a $\overline{request}$ on RQ/GT0 has higher priority.

18. Draw the bus request and bus grant timings in minimum mode system.



Bus request and bus grant timings in minimum

19. What is the purpose of a decoder in EU?

The decoder in EU translates instructions fetched from memory into a series of actions, which the EU carries out.

20. Give the register classification of 8086. (Nov/Dec2012)

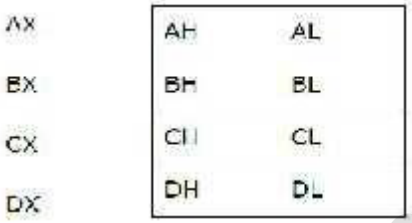
The 8086 contains:

i. General purpose registers: They are used for holding data, variables and intermediate results temporarily.

ii. Special purpose registers: They are used as segment registers, pointers, index register or as offset storage registers for particular addressing modes.

21. What are general data registers?

The registers AX, BX, CX and DX are the general data registers.



L and H represent the lower and higher bytes of particular register. AX register is used as 16-bit accumulator.

BX register is used as offset storage for forming physical addresses in case of certain addressing modes.

CX register is used as a default counter in case of string and loop instructions.

DX register is used as an implicit operand or destination in case of a few instructions.

22. Give the different segment registers. (April/May2012)

The four segment registers are:

i. Code segment register: It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

ii. Data segment register: It points to the data segment of the memory, where data is resided.

iii. Extra segment register: It also contains data.

iv. Stack segment register: It is used for addressing stock segment of memory. It is used to store stack data.

23. What are pointers and index registers?

IP, BP and SP are the pointers and contain offsets within the code, data and stack

segments respectively. SI and DI are the index registers, which are used as general purpose registers and also for offset storage in case of indexed, based indexed and relative based indexed addressing modes.

24. How is the physical address calculated? Give an example.

The physical address, which is 20-bits long is calculated using the segment and offsetregisters, each 16-bits long. The segment address is shifted left bit-wise four times and offset address is added to this to produce a 20 bit physical address.

```
Eg:   segment address - > 1005H
      Offset address  - >  5555H
      Segment address   > 1005H   >  0001 0000 0000 0101
      Shifted by 4 bit position - > 0001 0000 0000 0101 0000
      Offset address - >      +           0101 0101 0101 0101

      Physical address - >          0001 0101 0101 1010 0101
                                      1    5    5    A    5
```

25. What is meant by memory segmentation?

Memory segmentation is the process of completely dividing the physically available memory into a number of logical segments. Each segment is 64K byte in size and is addressed by one of the segment register.

26. What are the advantages of segmented memory?

The advantages of segmented memory are:

i. Allows the memory capacity to be 1Mbyte, although the actual addresses to be handled are of 16- bit size.

ii. Allows the placing of code, data and stack portions of the same program in different parts of memory for data and code protection.

iii. Permits a program and/or its data to be put into different areas of memory, each times program is executed i.e., provision for relocation may be done.

27. What is pipelining?

Fetching the next instruction while the current instruction executes is called pipelining.

28. What are the two parts of a flag register?

The two parts of the 16 bit flag register are:

i. Condition code or status flag register: It consists of six flags to indicate some condition produced by an instruction.

ii. Machine control flag register: It consists of three flags and are used to control certain operations of the processor.

29. Draw the format of 8086 flag register. (April/May2011)

8086 flag register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

U-Undefined

CF - Carry flag
PF - Parity flag
AF - Auxiliary flag
ZF - Zero flag
SF - Sign flag
TF - Single step trap flag
DF - Direction flag
IF - Interrupt enable flag
OF - Overflow flag

30. Explain the three machine control flags.

i. **Trap flag:** If this flag is set, the processor enters the single step execution.
ii. **Interrupt flag:** If this flag is set, the maskable interrupts are recognized by the CPU, otherwise they are ignored.

iii. **Direction flag:** This is used by string manipulation instructions. If this flag bit is„0", the string is processed from the lowest to the highest address i.e., auto incrementing mode. Otherwise, the string is processed from highest address to lowest address, i.e., auto decrementing mode.

31. What are the three groups of signals in 8086? (Nov/Dec2009)

The 8086 signals are categorized in three groups.
i. The signals having common functions in minimum and maximum mode.
ii. The signals having special functions for minimum mode.
iii. The signals having special functions for maximum mode.

32. What are the uses of AD15 – AD0 lines?

AD15 – AD0 are time multiplexed memory I/O address and data lines. Address remains on the lines during T1 state, while data is available on data bus during T2, T3, Tw and T4 states. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

33. What is the operation of RD signal?

RD is an active low signal. When it is low, it indicates the peripherals that the processor is performing a memory or I/O read operation.

34. Give the function of i. Ready and ii. INTR signal. (May/Jun 2013)

i**. Ready signal:** It is an acknowledgement from slow devices of memory that they have completed data transfer. The signal is synchronized by 8284 A clock generator to give ready input to 8086. The signal is active high.

ii. **INTR signal:** It is a level triggered input. This is sampled during the last cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. The signal is active high and internally synchronized.

35. What is the operation performed when TEST input is low?

When the TEST input is low, execution will continue, else, the processor remains in an idle state.

36. What is NMI (Non-Maskable Interrupt)?

NMI is an edge-triggered input, which causes a type 2 interrupt. It is not maskable internally by software and transition from low to high initiate the interrupt response at the end of the current instruction. This input is internally synchronized.

37. What is the purpose of clock input?

The clock input provides the basic timing for processor operation and bus control activity. It is an asymmetric square wave with 33% duty cycle. The range of frequency varies from5MHz to 10MHz.

38. What is the function of pin? (April/May2011)

The logic level at pin decides whether processor operates in minimum or maximum mode.

$MN/\overline{MX}$=0 Maximum Mode

$MN/\overline{MX}$=1 Minimum Mode

39. What happens when a high is applied to RESET pin?

When a high is given to RESET pin, the processor terminates the current activity and starts executing from FFFF0H. It must be active for at least four clock cycles. It is internally synchronized.

40. What will happen when a DMA request is made, while the CPU is performing a memory or I/O cycles? Nov/dec2011

When a DMA request is made, while the CPU is performing a memory or I/O cycles, it will request the local bus during T4 provided:

i. The request occurs on or before T2 state of the current cycle.

ii. The current cycle is not operating over the lower byte of a word.

iii. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

iv. A lock instruction is not being executed.

41. What is multiprogramming? [Nov/Dec 2015]

If more than one process is carried out at the same time, then it is known as multiprogramming. Another definition is the interleaving of CPU and I/O operations among several programs is called multiprogramming. To improve the utilization of CPU and I/O devices, we are designing to process a set of independent programs concurrently by a single CPU. This technique is known as multiprogramming.

42. Write the advantages of loosely coupled system over tightly coupled systems?

1. More number of CPUs can be added in a loosely coupled system to improve the system performance
2. The system structure is modular and hence easy to maintain and troubleshoot.
3. A fault in a single module does not lead to a complete system breakdown.

43. What is the different clock frequencies used in 80286?

Various versions of 80286 are available that run on 12.5MHz, 10MHz and 8MHz clock frequencies.

44. Define swapping in?

The portion of a program is required for execution by the CPU, it is fetched from the secondary memory and placed in the physical memory. This is called ‘swapping in’ of the program.

45. What are the different operating modes used in 80286?

The 80286 works in two operating modes

1. Real addressing mode
2. Protected virtual address mode.

46. What are the CPU contents used in 80286?

The 80286 CPU contains almost the same set of registers, as in 8086

- Eight 16-bit general purpose register

- Four 16-bit segment registers
- Status and control register
- Instruction pointer.

47. What are the signals used in 8086 maximum mode operation?

Qs1, Qs0, s0, s1, s2, LOCK, RQ/GT1, RQ/GT0 are the signals used in 8086 maximum mode operation.

48. Write the size of physical memory and virtual memory of 8086 microprocessor.

Physical addresses are formed when the left shifted segment base address is added to the offset address. The combination of segment register base addresses and offset address is the logical address in memory. Size of physical memory=220=1MB Size of virtual memory=216=64 KB

49. List the advantages of using segment registers in 8086.

- It allows the memory addressing capacity to be 1MB even though the address associated with individual instruction is only 16-bit.
- It facilitates use of separate memory areas for program, data and stack.
- It allows the program to be relocated which is very useful in multiprogramming.

50. Explain the BHE and LOCK signals of 8086

- *BHE (Bus High Enable):* Low on this pin during first part of the machine cycle indicates that at least one byte of the current transfer is to be made on higher byte AD15-AD8.
- *LOCK:* This signal indicates that an instruction with a LOCK prefix is being executed and the bus is not to be used by another processor.

51. What are the two modes of operations present in 8086?[may/june2007]

i. Minimum mode (or) Uniprocessor system

ii. Maximum mode (or) Multiprocessor system

52. What are the functions of status pins in 8086?

S2 S1 S0
0 0 0 ---- Interrupt acknowledge
0 0 1 ---- Read I/O
0 1 0 ---- Write I/O
0 1 1 ---- Halt

1 0 0 ---- Code access

1 0 1 ---- Read memory
1 1 0 ---- Write memory
1 1 1 ---- inactive
S4 S3
0 0 --I/O from extra segment

0 1 --I/O from Stack Segment

1 0 --I/O from Code segment

1 1 --I/O from Data segment
S5 --Status of interrupt enable flag
S6 --Hold acknowledge for system bus S7 --Address transfer.

53. What are the three classifications of 8086 interrupts? [MAY/JUNE-2006]

(1) Predefined interrupts,

(2) User defined Hardware interrupts,

(3) User defined software interrupts.

54. What are the differences between maximum mode and minimum mode? [NOV/DEC 2003]

Minimum Mode

1 A processor is in minimum mode when MN /MX pin is strapped to +5v
2. All control signals are given out by microprocessor chip it self
3. There is a single micro processor

Maximum mode

1. A processor is in maximum mode when MN /MX is grounded
2. The processor derives the status signals S2, Sl and So. Another chip called bus controller derives control signals using this status information.

3. There may be more than one microprocessor

55. What is Coprocessor? [NOV/DEC 2007] [APR/MAY2011]

The coprocessor is a processor which specially designed for processor to work under the control of the processor and support special processing capabilities. Example: 8087 which has numeric processing capability and works under 8086.

56. What are the basic multiprocessor configurations?

- Closely Coupled configuration
- Loosely coupled configuration

57. Differentiate External verses Internal Bus. [MAY/JUNE 2016]

Internal Data Bus: The internal data bus only works inside a CPU that is internally. It is able to communicate with the internal cache memories of the CPU. Since they are internally placed they are relatively quick and are now affected by the rest of the computer.

External Data bus: This type of bus is used to connect and interface the computer to its connected peripheral devices. Since they are external and do not lie within the circuitry of the cpu they are relatively slower. The 8088 processor in itself contains a 16-bit internal data bus coupled with a 20- bit address register. This allows the processor to address to a maximum of 1 MB memory.

58. Compare closely coupled and loosely coupled configurations.[NOV/DEC 2011] [May/June 2016]

Closely coupled	Loosely coupled
1. Single CPU is used	1. Multiple CPU modules are used
2. It has local bus only	2. It has local as well system bus
3. No system memory or IO	3. It has system memory and IO, shared
4. No bus arbitration logic required	4. Bus arbitration logic required among the CPU modes

PART-B (13 MARKS)

1. PINS AND SIGNALS OF 8086

Draw the pin Diagram of 8086 Processor and explain all the signals

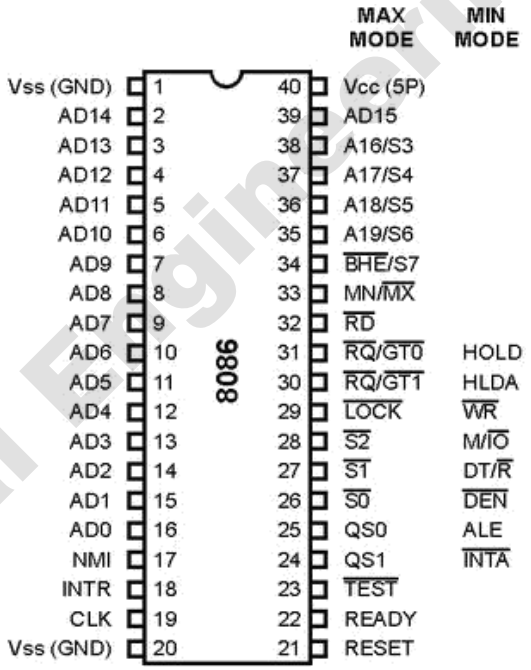
The 8086 signals can be categorized in three groups. The first are the signals having common functions in minimum as well as maximum mode, the second are the signals which have special functions in minimum mode and third are the signals having special functions for maximum mode

AD15-AD0: These are the time multiplexed memory I/O address and data lines.

When ALE = 1:**AD15-AD0 contains the address**

ALE = 0:**AD15-AD0 contains the data**

A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, and A_{16}/S_3: These are the time multiplexed address and status lines. During T_1, these are the most significant address lines or memory operations. The address bits are separated from the status bits using latches controlled by the ALE signal.



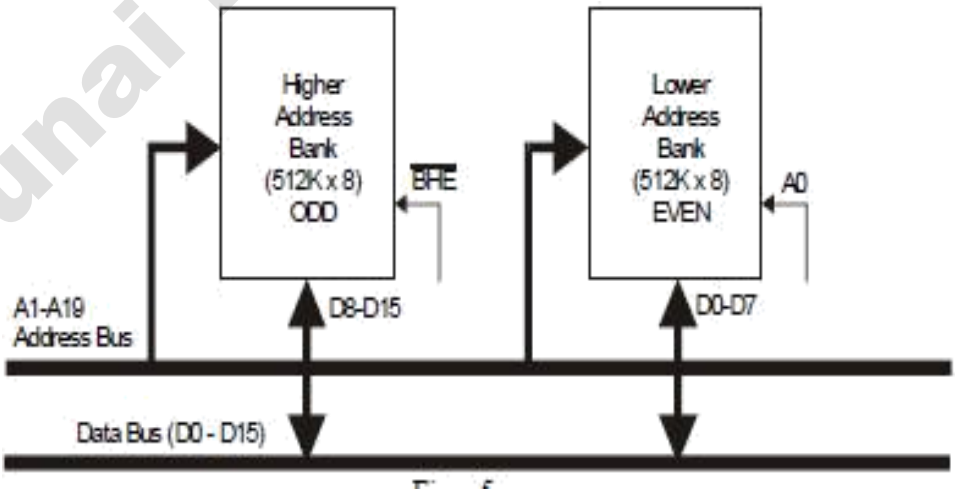
S4	S3	Function
0	0	Extra segment access
0	1	Stack segment access
1	0	Code segment access
1	1	Data segment access

BHE/S7-Bus High Enable/Status: The bus high enable signal is used to indicate the transfer of data over the higher order (D_{15}-D_8) data bus. It goes low for the data transfers over D_{15}-D_8 and is used to derive chip selects of odd address memory bank or peripherals.

One bank is connected to the lower half of the 16-bit data bus (D0 – D7) and contains even address bytes. *i.e.,* when A0 bit is low, the bank is selected. The other bank is connected to the upper half of the data bus (D8 - D15) and contains odd address bytes. *i.e.,* when A0 is high and BHE (Bus High Enable) is low, the odd bank is selected. A specific byte within each bank is selected by address lines A1-A19.

RD-Read: Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation.

READY: This is the acknowledgement from the slow devices or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. The signal is active high.



BHE	A0	Indication
0	0	Whole Word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	No Operation

INTR: Interrupt Request: If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

TEST: This input is examined by a 'WAIT' instruction. If the TEST input goes low, execution will continue, else, the processor remains in an idle state.

NMI: Non-maskable Interrupt: This is an edge-triggered input which causes a Type 2 interrupt. The NMI is not maskable internally by software.

RESET: This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronized.

CLK: The clock input provides the basic timing for processor operation and bus control activity. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

VCC: +5V power supply for the operation of the internal circuit. GND ground for the internal circuit.

MN/$\overline{\text{MX}}$: The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode.

The following pin functions are for the minimum mode operation of 8086.

M/IO: Memory/IO: When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation.

$\overline{\text{INTA}}$: Interrupt Acknowledge:. It means that the processor has accepted the interrupt.

ALE: Address latch Enable: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches.

DT/R Data Transmit/Receive: This output is used to decide the direction of data flow

through the transceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

$\overline{\text{DEN}}$: Data Enable This signal indicates the availability of valid data over the address/data lines. It is used to enable the transceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal.

HOLD, HLDA-Hold/Hold Acknowledge: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and it should be externally synchronized.

The following pin functions are applicable for maximum mode operation of 8086.

LOCK:

This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the LOCK signal is low. When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connectedto an external bus controller.

QS1, QS0-Queue Status: These lines give information about the status of the code-prefetch queue.

QS1	QS1	Characteristics
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

S2, S1, and S0: Status Lines: These are the status lines which reflect the type of operation,being carried out by the processor. These status lines are encoded in table.

S2	S1	S0	Characteristics
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive State

RQ/GT_0, RQ/GT_1: Request/Grant: These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with RQ/GT_0 having higher priority than RQ/GT_1, RQ/GT pins have internal pull-up resistors and may be left unconnected.

2. MINIMUM MODE 8086 SYSTEM AND TIMINGS

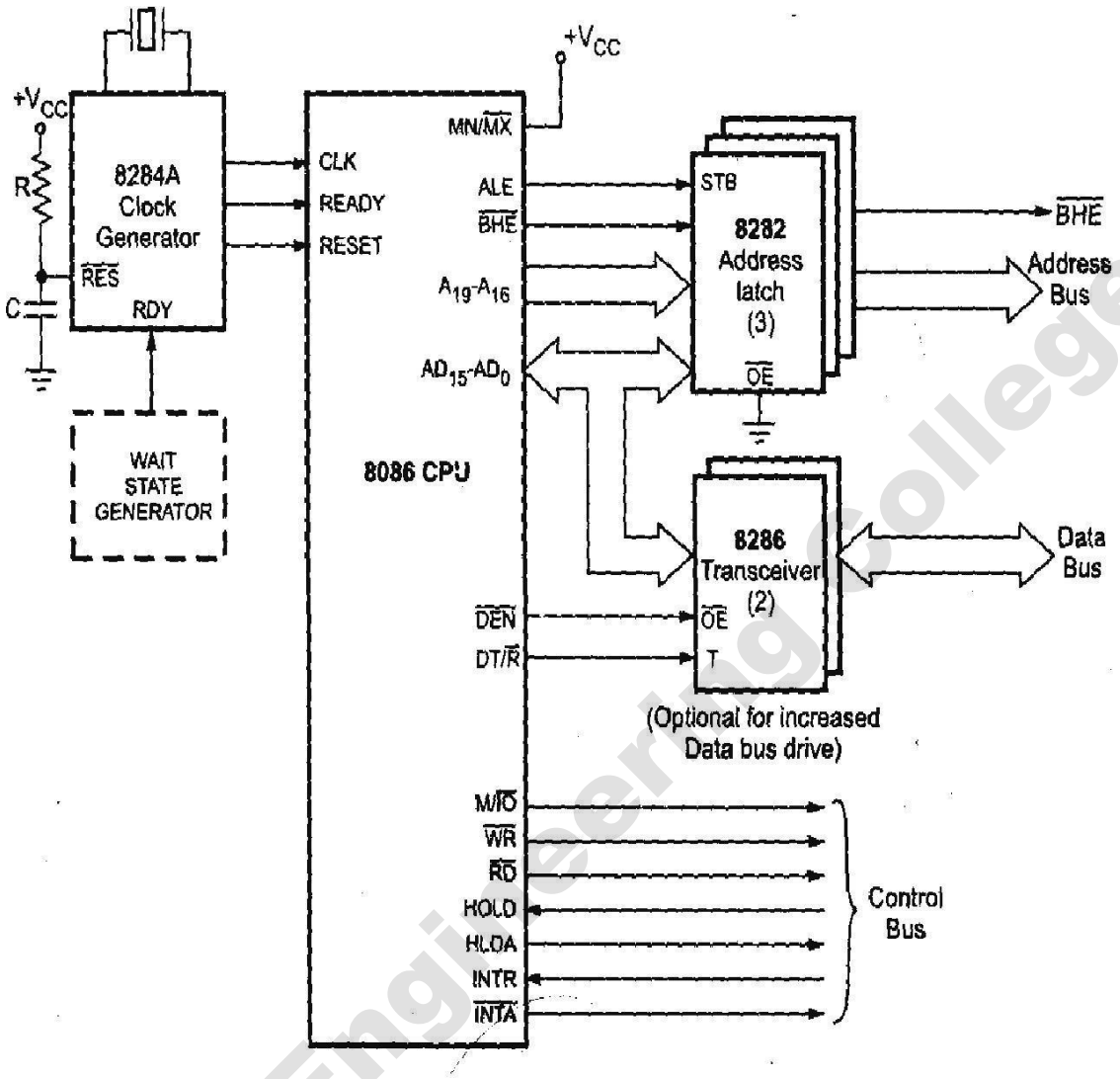
Describe the Minimum Mode Signals, Bus Cycles And Minimum Mode System Configuration Of 8086 Microprocessor In Detail. (16) [**Nov / Dec 2012]**
Draw and explain the timing diagram of write cycle in 8086 in minimum mode. (8) [**Nov /Dec 2013]**

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX* pin to logic1. In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system. The remaining components in the system are latches, transceivers, clock generator, memory and I/O devices.

The **latches** are generally buffered output D-type flip-flops, 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the 8-bit ALE signal generated by 8086.

Transceivers are the bidirectional buffers. They are required to separate the valid data from the time multiplexed address/data signal. They are controlled by two signals, namely, DEN* and DT/R*. The DEN* signal indicates that the valid data is available on the

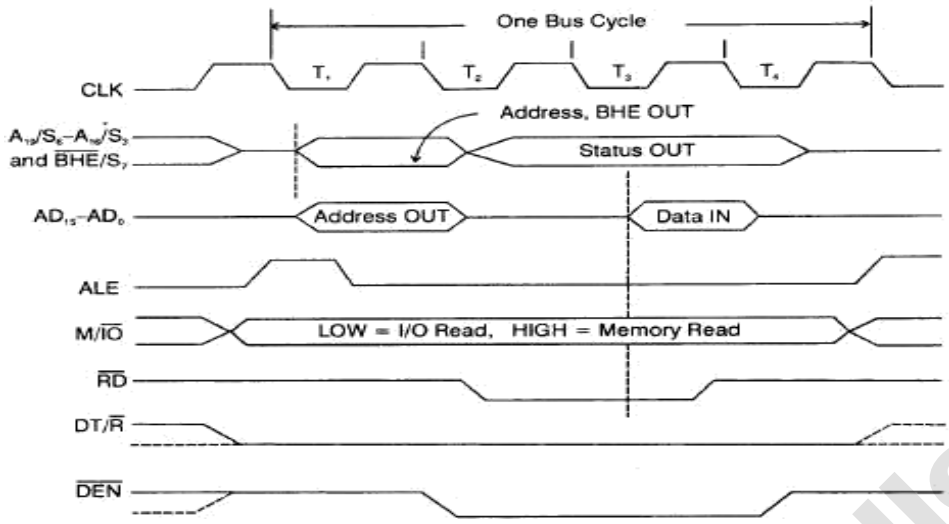
data bus, while DT/R indicates the direction of data, i.e. from or to the processor.



The system contains memory for the monitor and users program storage. Usually, EPROMS are used for monitor storage, while RAMs for users program storage. A system may contain I/O devices for communication with the processor as well as some special purpose I/O devices.

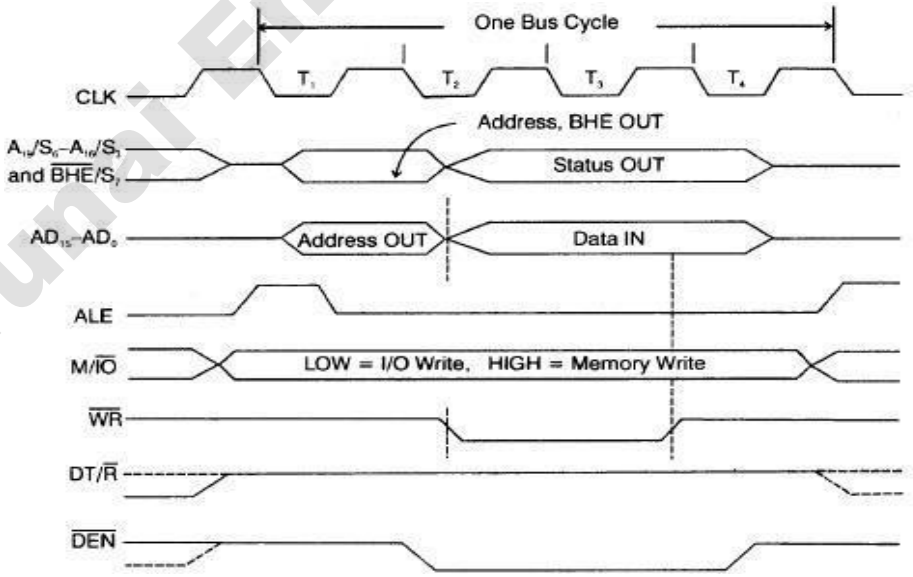
The **clock generator** generates the clock from the crystal oscillator and then shapes it and divides to make it more precise so that it can be used as an accurate timing reference for the system. The clock generator also synchronizes some external signals with the system clock. Since it has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.

The working of the minimum mode configuration system is described in terms of the timing diagrams. The opcode fetch and read cycles are similar. Hence the **timing diagram** can be categorized into two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.



Timing Diagram for Read Cycle

The above figure shows the timing diagram of read cycle. The read cycle begins in T_1 with the assertion of the address latch enable (ALE) signal and also M/IO* signal. During the negative going edge of this signal, the valid address is latched on the local bus. The BHE* and A_0 signals address low, high or both bytes. From T_1 to T_4, the M/IO* signal indicates a memory or I/O operation. At T2 the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD*) control signal is also activated in T_2 .The read (RD) signal causes the addressed device to enable its data bus drivers. After RD* goes low, the valid data is available on the data bus. The addressed device will drive the READY line high, when the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.



Timing Diagram for Write Cycle

Figure shows the timing diagram of write cycle. A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO* signal is again asserted to indicate a memory or I/O operation. In T_2 after sending the address in T_1 the processor sends the data to be written to the addressed location. The data remains on the bus until middle of T_4 state. The WR* becomes active at the beginning of T_2 (unlike RD* is somewhat delayed in T_2 to provide time for floating). The BHE* and A_0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written. The M/IO*, RD* and WR* signals indicate the types of data transfer as specified in Table

M/IO	RD	WR	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

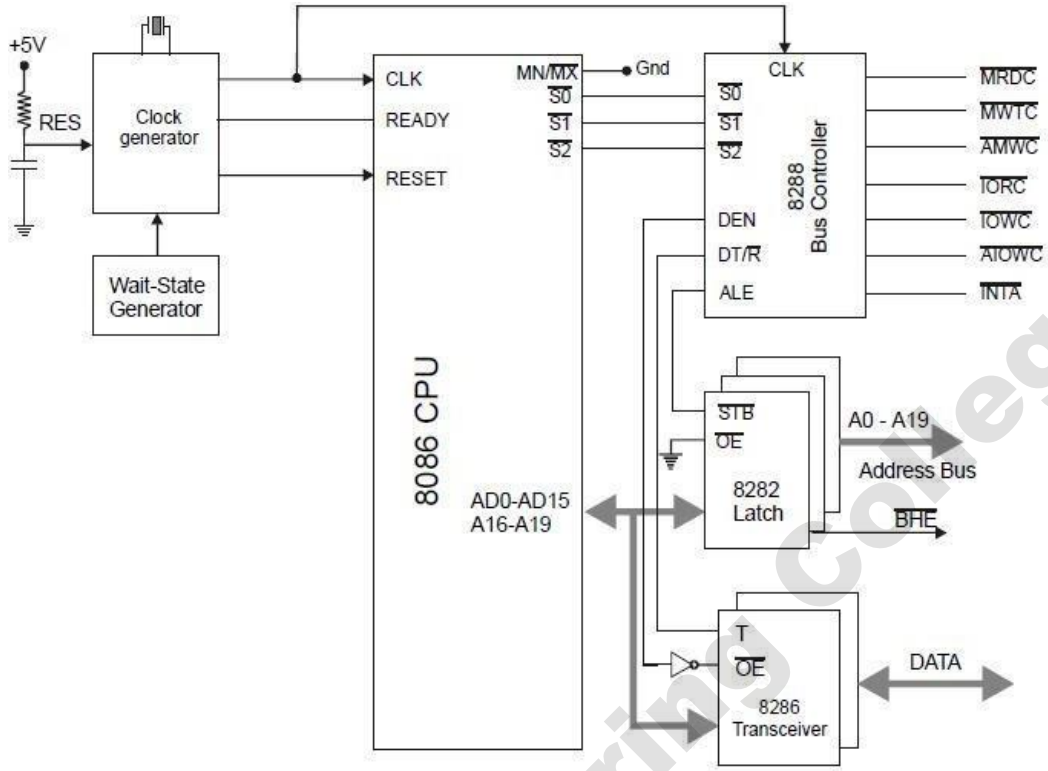
3. MAXIMUM MODE 8086 SYSTEM AND TIMINGS

Discuss the maximum mode configuration of 8086 by with a neat diagram. Mention the functions of the various signals. (16) [**Apr/May2015**]

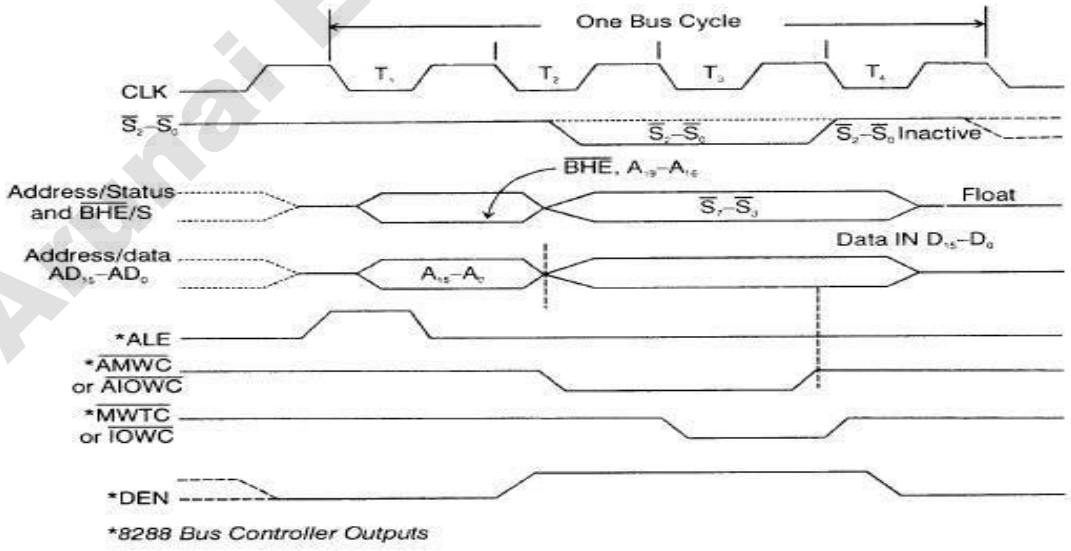
In the maximum mode, the 8086 is operated by strapping the MN/$\overline{MX}$ pin to ground. In this mode, the processor derives the status signals $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$. Another chip called **bus controller** derives the control signals using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration. The other components in the system are the same as in the minimum mode system. The general system organization is as shown in the figure

The basic functions of the bus controller chip IC8288, is to derive control signals like RD* and WR* (for memory and I/O devices), DEN*, DT/R*, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip has input lines S_2*, S_1* and S_0* and CLK. These inputs to 8288 are driven by the CPU. It derives the outputs ALE, DEN*, DT/R*, MWTC*, AMWC*, IORC*, IOWC* and AIOWC*. IORC*, IOWC* are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data from or to the bus. For both of these write command signals, the advanced signals namely AIOWC* and AMWTC* are available. They also serve the same purpose, but are activated one clock cycle earlier than the IOWC* and

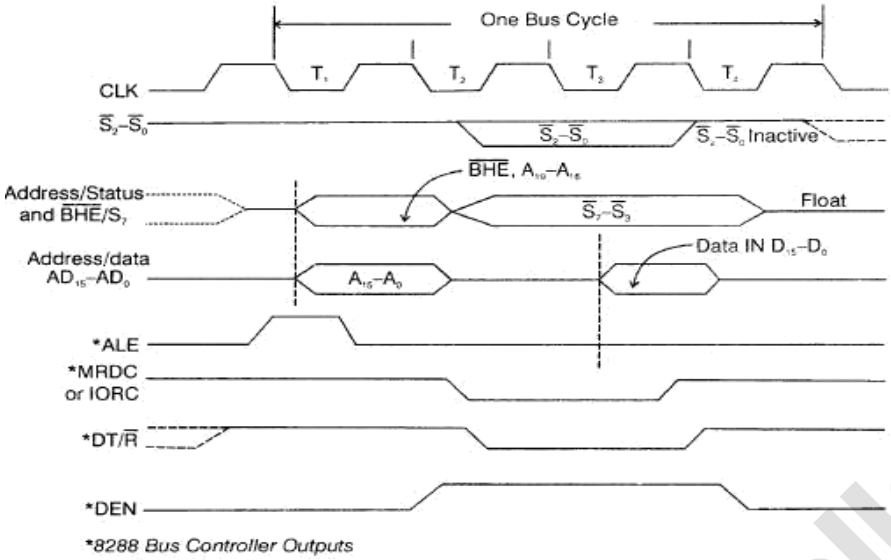
MWTC* signals, respectively.



The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are Similar to the minimum mode. ALE is asserted in T_1, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals.



Memory Write Timing in Maximum Mode



Memory Read Timing in Maximum Mode

4. MULTI PROCESSOR CONFIGURATIONS

Explain the closely coupled configuration of multiprocessor configuration with suitable diagram.(16) [**May/Jun 2014, April/May2017**]
Explain the loosely coupled configuration with suitable diagram.(16) [**Apr/May 2015**]
Compare closely coupled configuration with loosely coupled configuration.(8) [**Apr/May 2015]**

MULTIPROCESSOR SYSTEMS

A multiprocessor system will have two or more processors that can execute instructions or perform operations simultaneously.

Need for Multiprocessor Systems:

Due to limited data width and lack of floating point arithmetic instructions, 8086requires many instructions for computing even single floating point operation. For this Numeric Data Processor (8087) can help 8086 processor.

Advantages:

l. Several low cost processors may be combined to fit the needs of an application while avoiding the expense of the unneeded capabilities of a centralized system.
2. It is easy to add more processor for expansion as per requirement.
3. When a failure occurs, it is easier to replace the faulty processor.
4. In a multiprocessor system implementation of modular processing of task can be achieved

Maximum mode of 8086 is designed to implement 3 basic multiprocessor configurations:

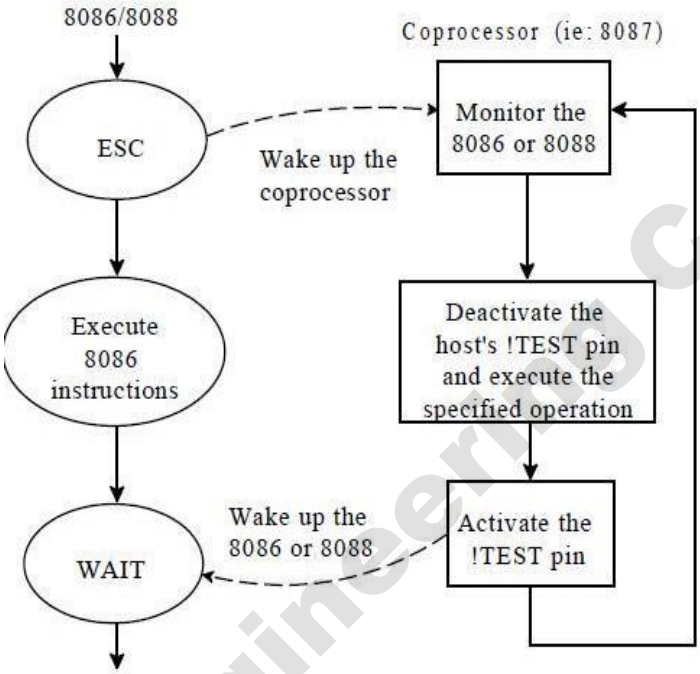
1. Coprocessor (8087)
2. Closely Coupled (8089)
3. Loosely Coupled (Multibus)

Coprocessor Configuration

In coprocessor configuration both the CPU (8086) and external processor (Math Coprocessor8087) share entire memory and I/O sub system. They also share same bus control logic and clock generator.8086 is the master and 8087 is the slave.

An instruction to be executed by the coprocessor is indicated by an escape(ESC) prefix or instruction.

1. The 8086 fetches the instructions.
2. The coprocessor monitors the instruction sequence and captures its own instructions.



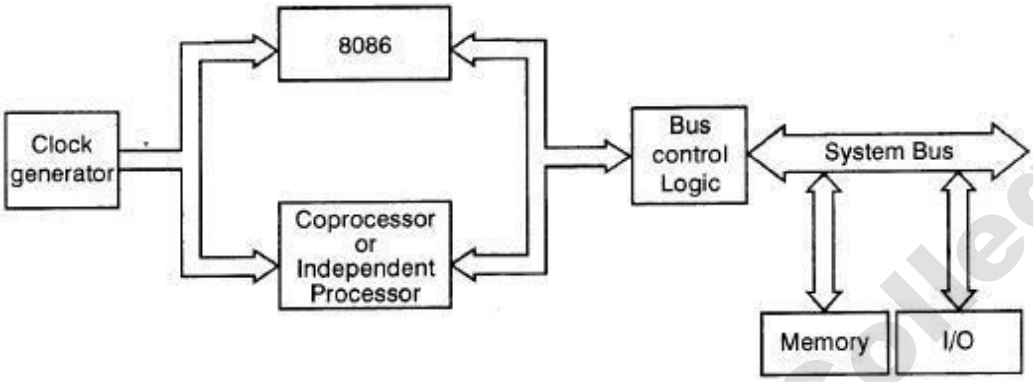
3. The ESC is decoded by the CPU and coprocessor simultaneously.
4. The CPU computes the 20 bit address of memory operand and does a dummy read. The coprocessor captures the address of the data and obtains control of the bus to load or store as needed.
5. The coprocessor sends BUSY (high) to the TEST pin.
6. The CPU goes to the next instruction and if this is an 8086 instruction, the CPU and coprocessor execute in parallel.
7. If another coprocessor instruction occurs the 8086mustwait until BUSY goes low ie, TEST pin become active. To implement this, a WAIT instruction is put in front of most 8087instructionsby the Assembler.
8. The WAIT instruction does the operations ie , wait until the TEST pin is active.
9. The coprocessor also makes use of Queue Status (QS_0-QS_1)of the 8086 Instructions queue

Closely Coupled Configuration

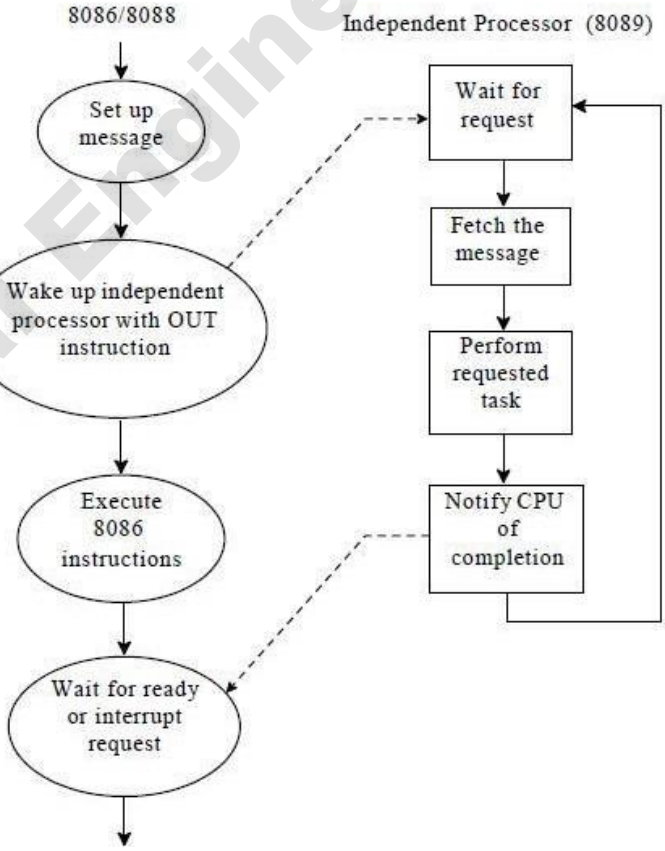
Coprocessor and closely coupled configuration are similar in that both the 8086 and the external processor (8089) share:

- Memory
- I/O system
- Bus and Bus control logic
- Clock generator

The interaction between 8086 and coprocessor or independent processor is shown below



The main difference between coprocessor and closely coupled configuration is, no special instruction WAIT or ESC is used. The communication between 8086 and independent processor is done through memory space. As shown in Figure the 8086 sets up a message in memory and wakes up independent processor by sending command to one of its ports. The independent processor then accesses the memory to execute the task in parallel with the 8086. When task is completed the external processor informs the 8086 about the completion of task by using either a status bit or an interrupt request

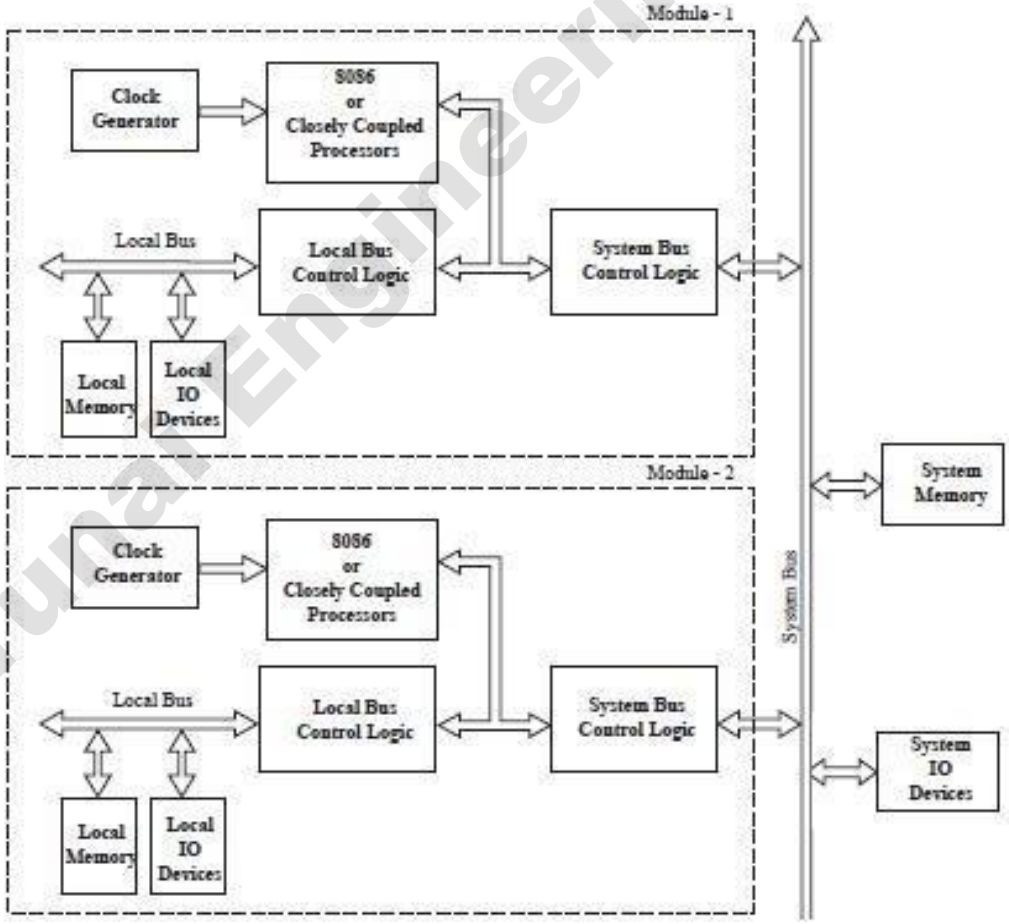


Loosely Coupled Configuration:

In loosely coupled configuration a number of modules of 8086 can be interfaced through a common system bus to work as a multiprocessor system. Each module is an independent microprocessor based system with its own clock source, and its own memory and I/O devices interfaced through a local bus. Each module can also be a closely coupled configuration of a processor or coprocessor The block diagram of a loosely coupled configuration of 8086 is shown in figure

Advantages

- High system throughput can be achieved by having more than one CPU.
- The system can be expanded in modular form. Each bus master module is an independent unit and normally resides on a separate PC board. One can be added or removed without affecting the others in the system.
- A failure in one module normally does not affect the breakdown of the entire system and the faulty module can be easily detected and replaced
- Each bus master has its own local bus to access dedicated memory or IO devices so a greater degree of parallel processing can be achieved



5. BUS ARBITRATION

Discuss the schemes used to solve the bus arbitration problem in multiprocessors. (6) [**Nov/Dec 2011, April/May2017]**

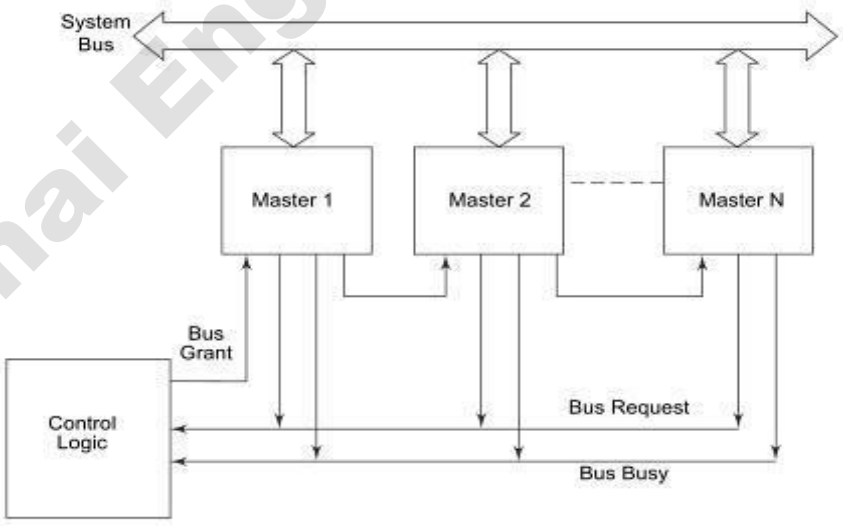
Multiple devices may need to use the bus at the same time so it must have a way to arbitrate multiple requests. Bus arbitration schemes usually try to balance:

- Bus priority – the highest priority device should b e serviced first
- Fairness – even the lowest priority device should n ever be completely locked out from the bus

Bus arbitration schemes can be divided into three classes 1.Daisy chaining.
2. Polling.
3. Independent requesting.

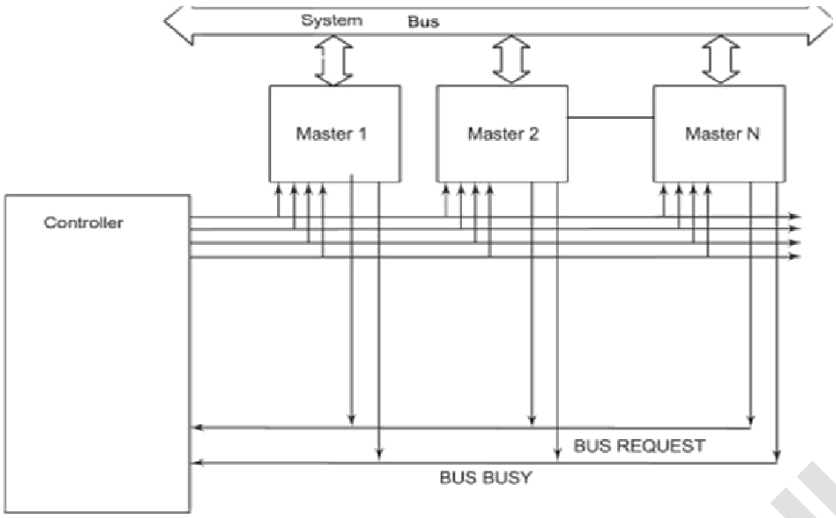
Daisy Chaining

In Daisy Chaining method all masters make use of the same line for bus request. In response to a bus request, the controller sends a bus grant if the bus is free. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activates the busy line and gains control of the bus. Therefore any other requesting module will not receive the grant signal and hence cannot get the bus access. This bus allocation scheme is simple and cheaper But failure of any one master causes the whole system to fail and arbitration is slow due to the propagation delay of bus grant signal is proportional to the number of masters



Polling

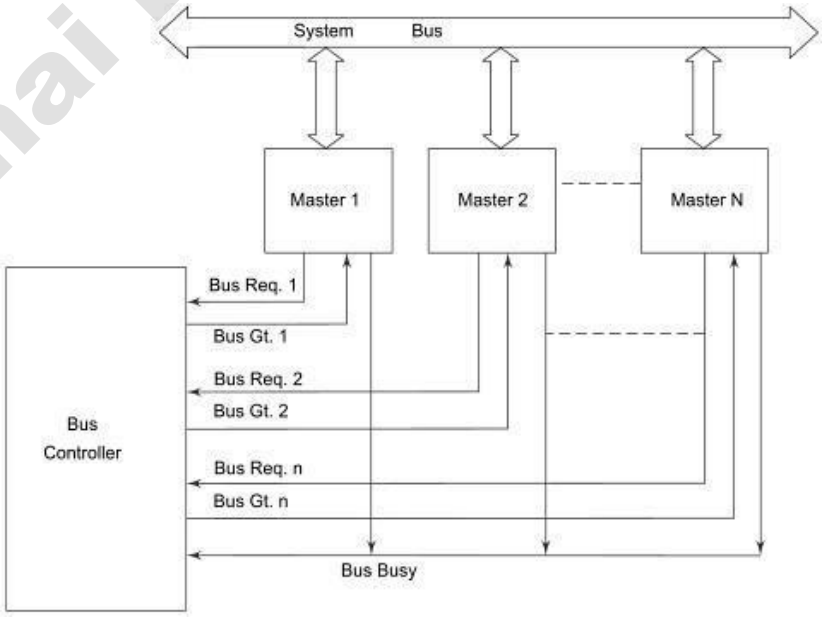
In polling method, the controller sends address of device to grant bus access. The Number of address lines required is depend on the number of masters connected in thesystem. For example, if 3



Masters are connected in the system, one address line is required. In response to a bus request, controller generates a sequence of master addresses when the requesting master recognizes the address; it activates the busy line and begins to use the bus. The priority can be changed by altering the polling sequence stored in the controller Another one advantage of this method is, if one module fails entire system does not fail.

Independent Priority

In the independent priority scheme each master has a separate pair of bus request (BRQ) and bus grant (BGR) lines and each pair has a priority assigned to it. The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal. Synchronization of clocks must be performed once a master is recognized; Master will receive a common clock from one side and pass it to the controller which will derive a clock for transfer. Due to separate pairs of bus request and bus grant signals, arbitration is fast.



UNIT III I/O INTERFACING

Memory Interfacing and I/O interfacing - Parallel communication interface – Serial communication interface – D/A and A/D Interface - Timer – Keyboard /display controller – Interrupt controller – DMA controller – Programming and applications Case studies: Traffic Light control, LED display, LCD display, Keyboard display interface and Alarm Controller

PART-A (2 MARKS)

1. List the Four Display Modes of 8279 Keyboard / Display Controller. [Nov / Dec 2012]

- Eight 8 bit Character Left Entry
- Sixteen 8 bit Character Left Entry
- Eight 8 bit Character Right Entry
- Sixteen 8 bit Character Right Entry

2. What are the enhanced features of 8254 Programmable Timer compared to 8253? [Nov / Dec 2012]

The maximum clock frequency is 8 MHZ. 8254 has a read back feature which allows to latch the count in all the counters and the status of the counter at any point.

3. How memory interfacing is differentiated from I/O interfacing? [Nov/Dec 2014]

Memory Interfacing chip select signal is needed but I/O interfacing it is not required. Also memory interfacing MEMR and MEMW control signal are used but I/O interfacing IOR and IOW control signal are used.

4. What is the need for de-bouncing the key board? [Nov/Dec 2012, Nov/Dec 2013,Nov/Dec 2014]

Debouncing the key board is used to identify the valid key. When key is depressed and released, the contact is not broken permanently. In fact the key makes and breaks the contacts several times for a few milliseconds before the contact is broken permanently

5. What is DMA? [Nov/Dec 2011]

Direct Memory Access. The device may transfer data directly to/from memory without any interference from the CPU. The device requests the CPU (through a DMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory.

6. What is the purpose of control word written to control register in 8255? [April/May2011]

The content of the control register specify an I/O function for each port. This register can be accessed to write word when A_0 and A_1 are at logic 1. This register is not accessible forread operation. Bit D_7 specifies either the I/O function or the BSR functions. If bit D_7 = 1,Bits D_6 –D_0 determine I/O functions in various modes. If Bit D_7 = 0 port C operates in BSR mode.

7. What are the advantages of Programmable Interval Timer/Counter IC? [May/Jun 2014]

- Compatible with All Intel and Most Other Microprocessors
- Handles Inputs from DC to 10 MHz
- Status Read-Back Command
- Six Programmable Counter Modes
- Three Independent 16-Bit Counters
- Binary or BCD Counting
- Single +5V Supply
- Standard Temperature Range.

8. List the features of memory mapped I/O. [May/Jun 2014]

1. Memory-mapped I/O uses a section of memory for I/O. The idea is simple. Instead of having "real" memory (i.e., RAM) at that address, place an I/O device.
2. Thus, communicating to an I/O device can be the same as reading and writing to memory addresses devoted to the I/O device. The I/O device merely has to use the same protocol to communicate with the CPU as memory uses.
3. Some ISAs use special I/O instructions. However, the signals generated by the CPU for I/O instructions and for memory-mapped I/O are nearly the same. Usually, there's just one special I/O pin that lets you know whether its a memory address or an I/O address. Other than that, they behave nearly identically.

9. Define scan counter? [Nov/Dec 2011]

The scan counter has two modes to scan the key matrix and refresh the display. In the encoded mode, the counter provides binary count that is to be externally decoded to provide the scan lines for keyboard and display. In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL0-SL3.The keyboard and display both are in the same mode at a time.

10. Give the various modes and applications of 8254 timer? [Apr/May 2015]

Mode 0 Interrupt on Terminal Count - to control parking lot Signs around electronic factory. Accurate time delay under software control

Mode 1 Programmable One Shot –To produce an interrupt signal if the ac power fails.

Mode 2 Rate Generator – to produce a 1 KHz signal for a real time clock from an 8 MHz processor clock signal. Real time Clock interrupt

Mode 3 Square Wave Generator - Programmable audio tone generator

Mode 4 Software Triggered Strobe - Parallel Data Transfer and send out a strobe signal to let the receiving system know that the data is available

Mode 5 Hardware Triggered Strobe - Parallel Data Transfer.

11. What is memory mapped I/O? (Nov/Dec 2014)

This is one of the techniques for interfacing I/O devices with μp. In memory mapped I/O, the I/O devices assigned and identified by 16-bit addresses. To transfer the data between MPU and I/O devices memory related instructions (such as LDA, STA etc.) and memory control signals (MEMR, MEMW) are used.

12. What is I/O mapped I/O? (April/May 2013)

This is one of the techniques for interfacing I/O devices with μp. In I/O mapped I/O, the I/O devices assigned and identified by 8-bit addresses. To transfer the data between MPU and I/O devices I/O related instructions (IN and OUT) and I/O control signals (IOR, IOW) are used.

13. What is simplex and duplex transmission?

Simplex transmission: data are transmitted in only one direction. Duplex transmission: data flow in both directions. If the transmission goes one way at a time, it is called half duplex; if it goes both way simultaneously, then it is called full duplex.

14. Define Baud. (EE2354May/June2012)

The rate at which the bits are transmitted, bits per second is called Baud.

15. What are the signals available for serial communication?

SID – serial input data

SOD – serial output data

16. What is USART?

It is a programmable device. Its function and specification for serial I/O can be determined by writing instructions in its internal registers. The Intel 8251A USART is a device widely used in serial I/O.

17. Write the features of 8255A. (Nov/Dec 2013)

The 8255A has 24 I/O pins that can be primarily grouped primarily in two 8-bit Parallel ports: A and B, with eight bits as port C. The 8-bits of port C can be used as two 4-bit ports: C UPPER CU and CLOWER CL.

18. What is BSR mode?

All functions of 8255 are classified according to 2 modes. In the control word, if D7 = 0, then it represents bit set reset mode operation. The BSR mode is used to set or reset the bits in port C.

19. What is mode 0 operation of 8255? (Nov/Dec2011)

In this mode, ports A and B are used as two simple 8-bit I/O ports and port C as two 4-bit ports. Each port can be programmed to function as an input port or an output port. The input/ output features in mode 0 as follows:

i. Outputs are latched

ii. Inputs are not latched

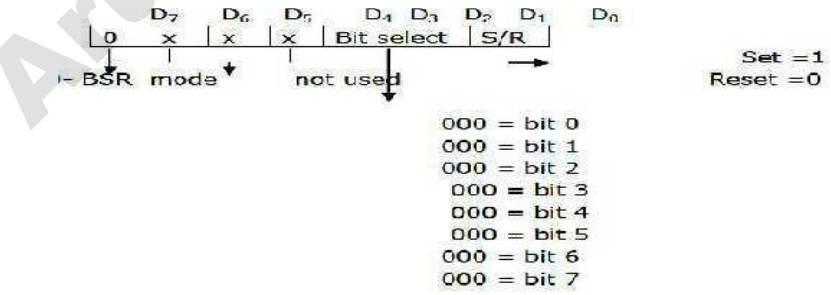
iii. Ports do not have handshake or interrupt capability.

20. What are the modes of operation supported by 8255?

i. Bit set reset mode (BSR)

ii. I/O mode Mode 0 Mode1 Mode2

21. Write the control word format for BSR mode.



22. What are ADC and DAC?

The electronic circuit that translates an analog signal into a digital signal is called analog-to- digital converter(ADC).The electronic circuit translates a digital signal into an analog signal is called Digital-to-analog Converter(DAC).

23. Define conversion time.

It is defined as the total time required to convert an analog signal into a digital output. It is determined the conversion technique used and by the propagation delay in various circuits.

24. What are the functions to be performed by µp while interfacing an ADC?

i. Send a pulse to the START pin.
ii. Wait until the end of conversion
iii. Read the digital signal at an input port

25. Write the different types of ADC.

i. Single slope ADC
ii. Dual slope ADC
iii. Successive approximation ADC
iv. Parallel comparator type ADC
v. Counter type ADC

26. What is resolution time in ADC?

It is defined as a ratio of change in value of input voltage Vi, needed to change the digital output by 1 LSB. If the full scale input voltage required to cause a digital output of all 1‟s is ViFS. Then the resolution can be given as
Resolution = ViFS / (2n-1)

27. List the functions performed by 8279. (April/May2009)

i. It has built-in hardware to provide key debounce.
ii. It provides a scanned interface to a 64 contact key matrix.
iii. It provides multiplexed display interface with blanking and inhibit options.
iv. It provides three input modes for keyboard interface.

28. What is key debounce? (Nov/Dec2014)

The push button keys when pressed, bounces a few times, closing and opening the contacts before providing a steady reading. So reading taken during bouncing may be faulty. Therefore the microprocessor must wait until the key reach to steady state. This is known as key debounce.

29. What are the operating modes in 8279? (Nov/Dec2013)

i. Scanned keyboard mode
ii. Scanned sensor matrix
iii. Strobed input

30. What is N-key rollover? Nov/Dec2013, (April/May2012)

In N-key rollover each key depression is treated independently from all others. When a key is depressed, the denounce logic is set and 8279 checks for key depress during next two scans.

31. Find the program clock command word if external clock frequency is 2MHz.

Prescalar value = (2 x106) / (100 x 103) = (10100)2 Therefore command word = (00110100)2

32. What is multiple interrupt processing capability?

Whenever a number of devices interrupt a CPU at a time, and id the processor is able to handle them properly, it is said to have multiple interrupt processing capability.

33. What is hardware interrupt?

An 8086 interrupt can come from any one of three sources. One source is an external signal applied to the nonmaskable interrupt (NMI) input in or to the interrupt (INTR) input pin. An interrupt caused by the signal applied to one of these input is referred to as hardware interrupt.

34. What is software interrupt?

The interrupt caused due to execution of interrupt instruction is called software interrupt.

35. What are the two types of interrupts in 8086?

The two types of interrupts are:

i. **External interrupts:** In this, the interrupt is generated outside the processor. Example: Keyboard interrupt.

ii. **Internal interrupts:** It is generated internally by the processor circuit or by the execution of an interrupt instruction. Example: Zero interrupt, overflow interrupt.

36. What is the purpose of control word written to control register in8255? (April/May2011)

The control words written to control register specify an I/O function for each I.O port. The bit D7 of the control word determines either the I/O function of the BSR function.

37. What is memory mapping? (Nov/Dec 2007)

The assignment of memory addresses to various registers in a memory chip is called as memory mapping.

38. What are the modes of operations used in 8254? (Apr/May 2015)

Each of the three counters of 8254 can be operated in one of the following six modes of operation.

1. Mode 0 (Interrupt on terminal count)
2. Mode 1 (Programmable mono shot)
3. Mode 2 (Rate generator)
4. Mode 3 (Square wave generator)
5. Mode 4 (Software triggered strobe)
6. Mode 5 (Hardware triggered strobe)

39. List the operating modes of 8255A and 8237A. (NOV/DEC 2015)

8255 has 2 modes.

1. I/O mode-Multiprocessor

- Mode 0
- Mode 1
- Mode 2

2. Bit Set-Reset mode (BSR) 8237 has several modes. They are,

- Single mode
- Burst mode
- Block mode
- Demand mode
- Cascade mode

40. What freq. transmit clock (Txc') is required by an 8251 in order for it to transmit data at 4800 baud with a baud rate factor of 16. (Nov/Dec 2015)

T=209us

41. What is keydebouncing? (May/June 2016)

When the key is depressed and released, the contact is not broken permanently. In fact, the key makes and breaks the contacts several times for a few milliseconds before the contact is broken permanently.

PART-B (13 MARKS)

1. 8255 PROGRAMMABLE PERIPHERAL INTERFACE (PPI)

With neat block diagram explain the 8255 Programmable Peripheral Interface and its operating modes. [Marks 16] [**April/May 2011, April/May2017]**

Explain The Programming And Operating Modes Of 8255 PPI in Detail. (16) [**Nov / Dec 2012]**

With neat block diagram explain the 8255 programmable peripheral interface and its operating modes. (16)[**Nov /Dec 2013]**

Explain the mode 0 operation of 8255 programmable Peripheral Interface.(8) [**May/Jun 2014**]

Explain in detail about the parallel communication interface. (8) [**Nov/Dec 2014**].

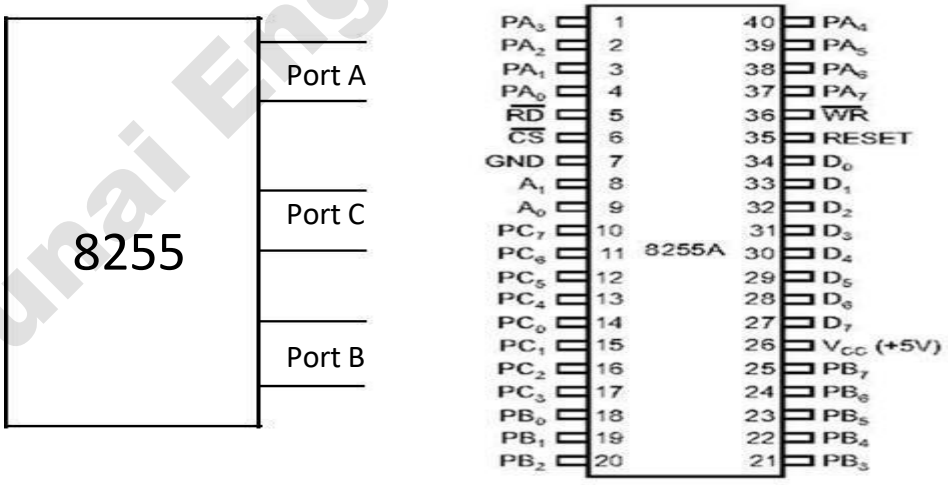
Explain the mode 1 operation of 8255 programmable Peripheral Interface.(8) [**Apr/May 2015**]

The 8255 chip is also called as Programmable Peripheral Interface. The Intel 8255A is a general purpose programmable I/O device which is designed for use with all Intel and most other microprocessors. It has 3 I/O ports, Port A , Port B and Port C each of 8 bits. The eight bits of Port C is divided into two 4 bit ports. Cupper (C_U) and C lower (C_L).8255 contains two modes of operation Bit Set/Reset Mode (BSR) and I/O mode.

BSR Mode is used to set or reset the bits in port C which is used for hand shake signals.

I/O mode is divided into three modes

Mode 0 (Simple input/output)
Mode 1 (Handshake mode)
Mode 2 (Bidirectional Data Transfer)

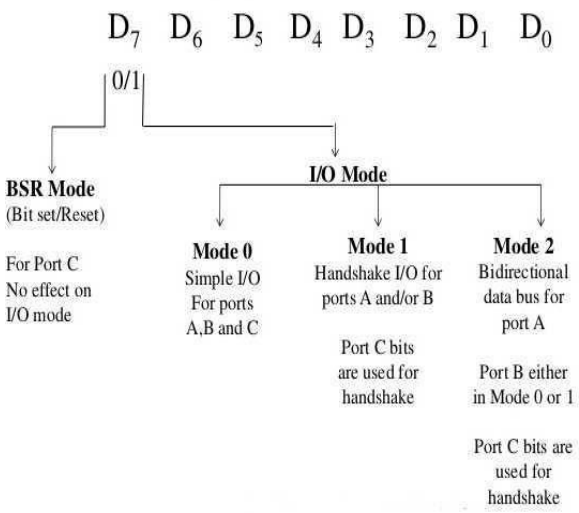


Mode 0 Operation (Simple input/output) It does not use any handshake signals. All the ports are used for simple data transfer.

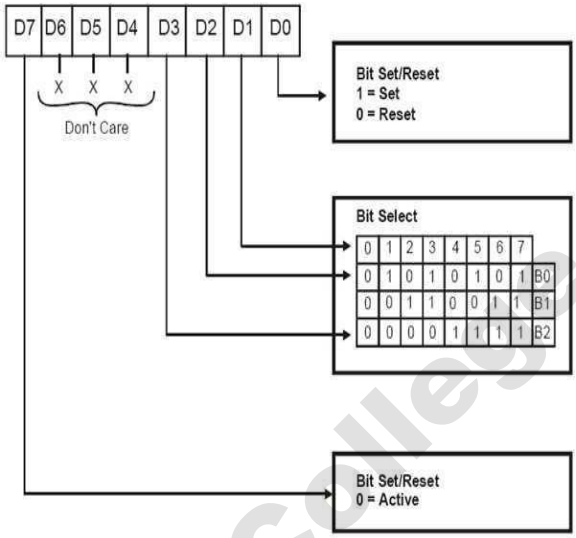
Mode 1 Operation (Handshake mode) Port A and B are used for data transfer and Port C is used for hand shake signals.

Mode 2 (Bidirectional Data Transfer) Port A is used for Bidirectional data transfer. Port Bin either in mode 0 or 1. Port C is used for Handshake signals.

MODES OF 8255



BIT SET/RESET (BSR) MODE-
Set/Reset bits in Port C



BLOCK DIAGRAM OF 8255

The block diagram contains

1. Data bus buffer
2. Read/Write control logic
3. Group A and Group B controls
4. Port A, B and C

DATA BUS BUFFER

This three-state bi-directional 8-bit buffer is used to interface the 8255 to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

READ/WRITE AND CONTROL LOGIC

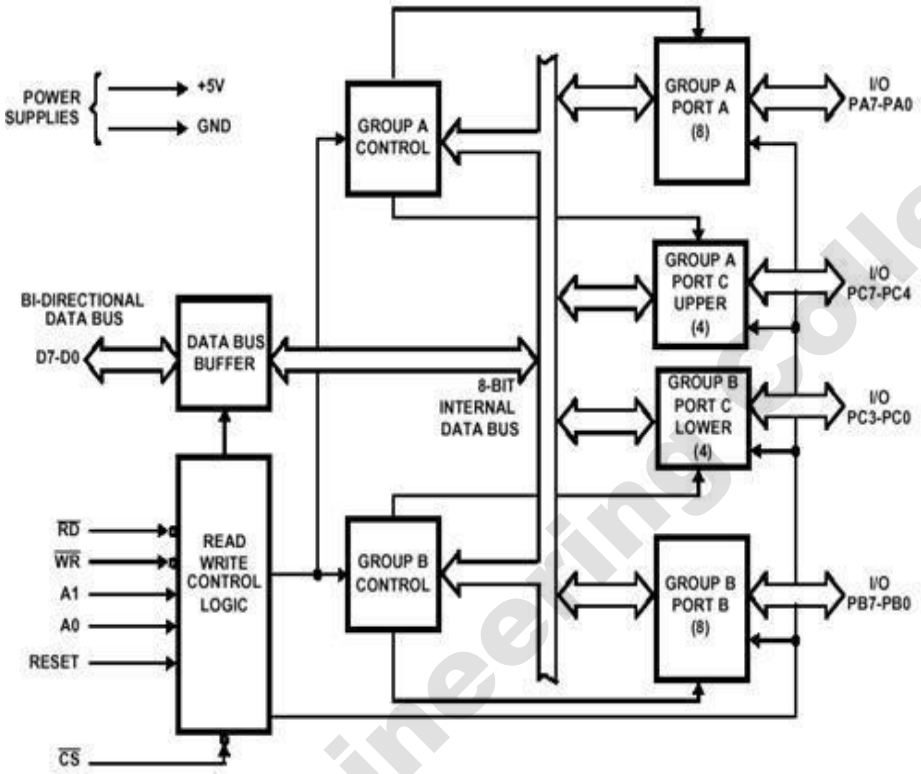
The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

CS Chip Select. A "low" on this input pin enables the communication between the 8255 and the CPU.

RD Read: This control signal enables the read operation. A "low" on this input pin enables 8255 to read data from the selected I/O.

WR Write: This control signal enables the write operation. A "low" on this input pin enables 8255 to write a data into the selected I/O or control register.

A0 and A1: These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).



A1	A0	SELECTION
0	0	PORT A
0	1	PORT B
1	0	PORT C
1	1	CONTROL

RESET: A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

GROUP A AND GROUP B CONTROLS

These blocks receive control from the CPU and issues commands to their respective ports. The two groups of I/O pins are named as Group A and Group B. Group A contains eight I/O lines of Port A (PA_0 – PA 7) and another four lines of Port Cupper (PC_0 – PC 3). Group B contains eight I/O lines of Port B (PB_0 – PB 7) and another four lines of Port C lower (PC4 – PC 7).

The functional configuration of each port is programmed by the systems software. In essence, The CPU "outputs" a control word to the 8255. The control word contains information such as"mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

PORTS A, B, AND C

The 8255 contains three 8-bit ports (A, B, and C).

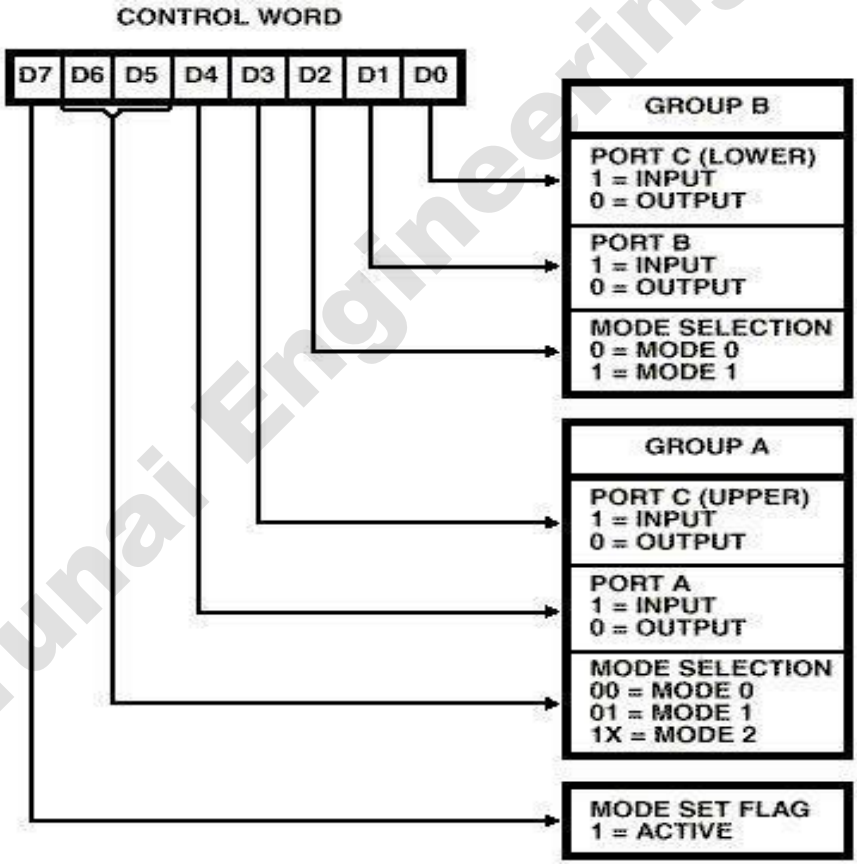
Port A: This has an 8 bit latched/buffered O/P and 8 bit input latch. It can be programmed in 3 modes – mode 0, mode 1, and mode 2

Port B: This has an 8 bit latched / buffered O/P and 8 bit input latch. It can be programmed in mode 0, mode1.

Port C: This has an 8 bit latched input buffer and 8 bit output latched/buffer. This port can be divided into two 4 bit ports and can be used as control signals for port A and port B. it can be programmed in mode 0.

Control Word Register

The content of the control register specify an I/O function for each port. This register can be accessed to write word when A_0 and A_1 are at logic 1. This register is not accessible for read operation. Bit D_7 specifies either the I/O function or the BSR functions. If bit $D_7 = 1$, bits D_6 –D_0 determine I/O functions in various modes. If Bit $D_7 = 0$ port C operates in BSR mode.



Modes of Operation

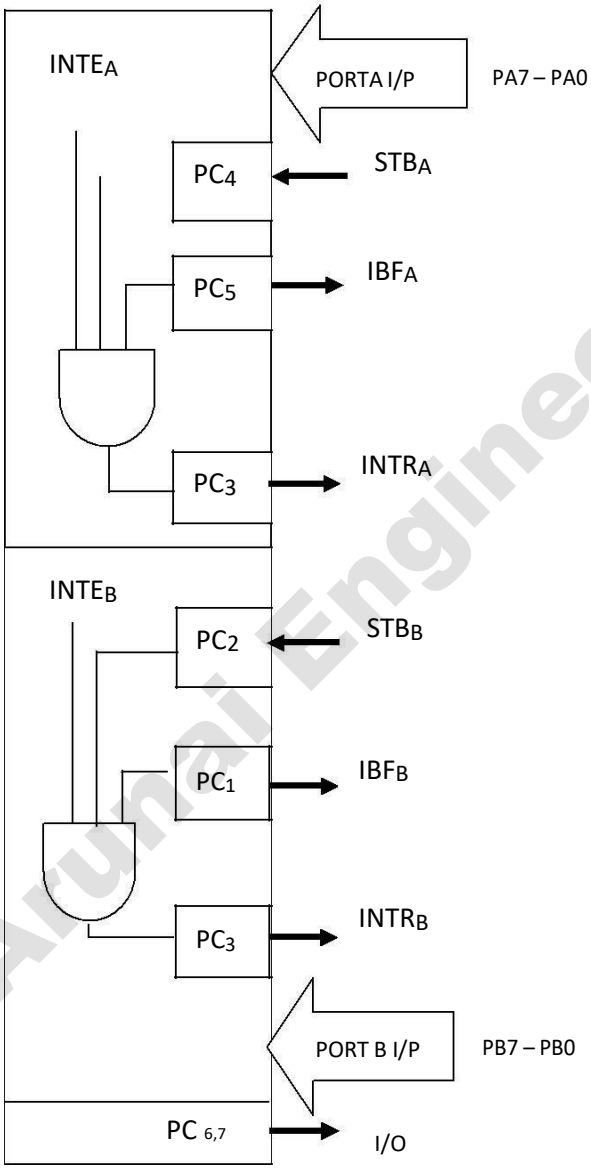
These are two basic modes of operation of 8255. Bit Set/Reset Mode (BSR) and I/O Mode

In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C can be used to set or reset its individual port bits. It is used to set or reset the bits in port C which is used for hand shake signals.

The I/O mode is divided into three modes: Mode 0 (Simple input/output) , Mode 1 (Handshake mode) , Mode 2 (Bidirectional Data Transfer)

Mode 0 Operation (Simple input/output) It does not use any handshake signals. All the ports are used for simple data transfer. It is used for interfacing an I/p device or an o/p device. It is used when timing characteristics of I/O devices is well known.

Mode 1 Operation (Handshake mode) Port A and B are used for data transfer and Port C is used for hand shake signals. 3 lines are used for handshaking. It is used for interfacing an input device or an output device. Handshake signals of the port inform the processor that the data is available, data transfer complete etc.



Mode 1 Input Control Signals

STB:

The strobe input loads data into the port latch on a 0-to-1 transition.

IBF:

Input buffer full is an output signal indicating that the input latch contains information.

INTR:

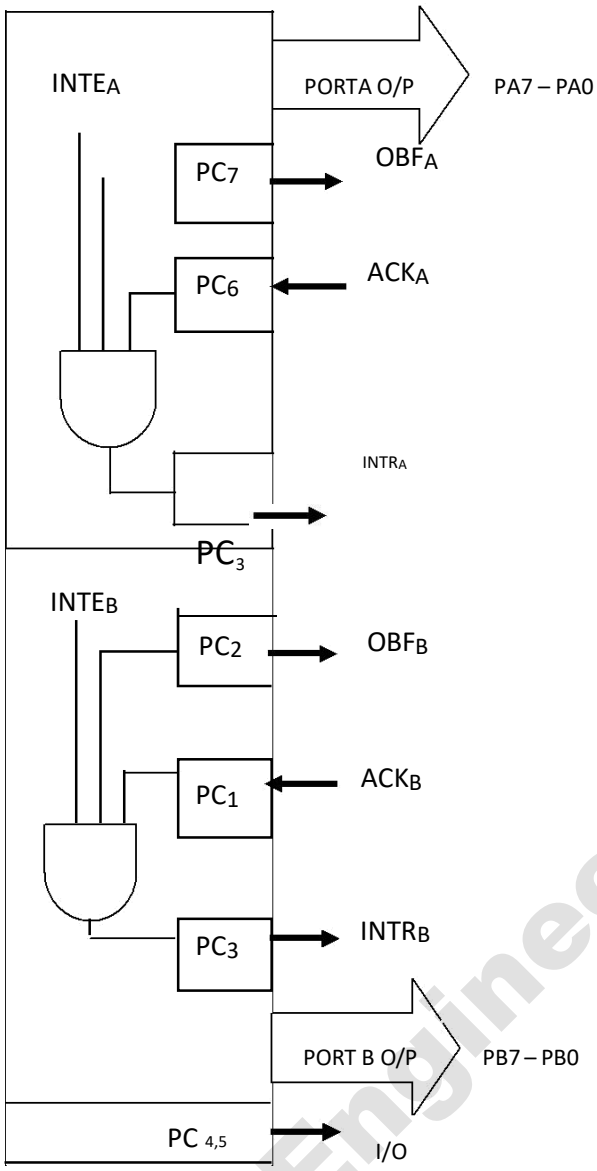
Interrupt request is an output signal that requests an interrupts.

INTE:

The interrupt enable signal is an internal flip flop used to enable or disable the generation of INTR signal.

PC7, PC6:

The port C pins 7 and 6 are general purpose I/O pins that are available for



Mode 1 Output Control Signals

OBF:
Output buffer full is an output signal that goes low when data is latched in either port A or port B. Goes low on ACK.

ACK:
An input from a peripheral device that must output a low when the peripheral receives a data.

INTR:
Interrupt request is an output signal that can be used to interrupt the MPU to request the next data byte for output.

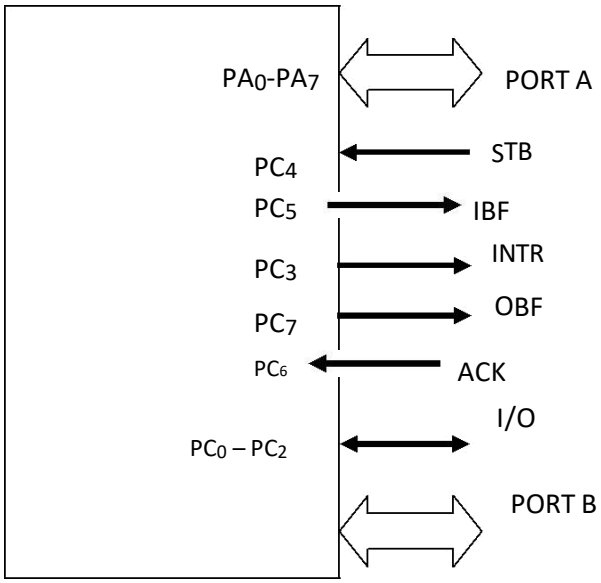
INTE:
The interrupt enable signal is an internal flip flop used to enable or disable the generation of INTR Signal.

PC5, PC4:
The port C pins 5 and 4 are general-purpose I/O pins that are available for any purpose.

Mode 2 (Bidirectional Data Transfer)

Port A is used for Bidirectional data transfer. Port B in either in mode 0 or 1. Port C is used for Handshake signals .This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow.

- **INTR:** Interrupt request is an output signal that can be used to interrupt the MPU to request the next data byte for output.
- **OBF:** Output Buffer Full is an output indicating that that output buffer contains data for the bi-directional bus.
- **ACK:** An input from a peripheral device that must output a low when the peripheral receives a data.
- **STB:** The strobe input loads data into the port A latch.

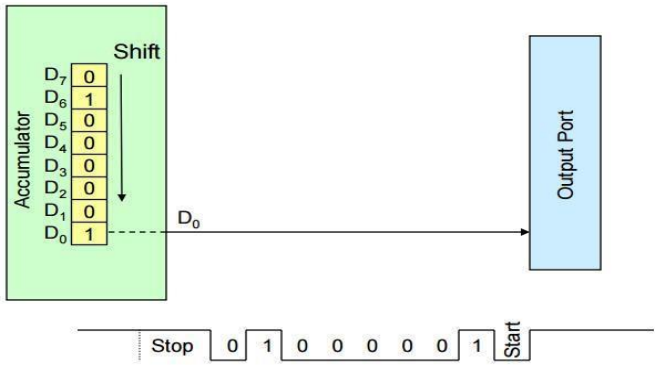


- **IBF:** Input buffer full is an output signal indicating that the input latch contains information for the external bi-directional bus.
- **INTE:** The interrupt enable signal is an internal flip flop used to enable or disable the generation of INTR signal.
- **PC2, PC1, and PC0:** These port C pins are general-purpose I/O pins that are available for anypurpose.

2. SERIAL COMMUNICATION INTERFACE 8251 (USART)

Explain the 8251 USART with neat block diagram. Also explain its mode word, command word and status word. (16) [**Nov/Dec 2011]**

A serial communications interface (SCI) is a device that enables the serial (one bit at a time) exchange of data between a microprocessor and peripherals such as printers, external drives, scanners. 8251 is a Universal Synchronous and Asynchronous Receiver and Transmitter compatible with Intel's processors. This chip converts the parallel data into a serial stream of bits suitable for serial transmission. It is also able to receive a serial stream of bits and convert it into parallel data bytes to be read by a microprocessor.



Basic Modes of data transmission

a) **Simplex Mode**: Data is transmitted only in one direction from the transmitter to the receiver over a single communication channel.

b) **Half Duplex Mode**: Data transmission may take place in either direction, but at a time data may be transmitted only in one direction.

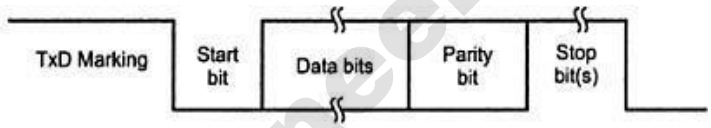
c) Full Duplex Mode: Data transmission may take place in both directions

simultaneously. Serial Communication takes place in two methods,

Asynchronous data Transfer and Synchronous data Transfer.

Asynchronous Data Transfer

It allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, special bits will be added to each word in order to synchronize the sending and receiving of the data. When a word is given for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. The stop bit will be added at the end of the data.



Synchronous Data Transfer

The receiver knows when to "read" the next bit coming from the sender. This is achieved by sharing a clock between sender and receiver. It is suitable for long distance since exchange of data is done through one cable. Once the SYNC character is detected 8251 starts receiving the data.

SYNC1	SYNC2	Data

Transmission Rate:

Bits per second: Number of bits transmitted per second.

Baud rate: It is a measurement of transmission speed in asynchronous communication; it represents the number of bits/sec that are actually being sent over the serial link.

ARCHITECTURE OF 8251A

Data Bus Buffer: This tri-state, bi-directional, 8-bit buffer is used to interface 8251 to the system data bus. Along with the data, control word, command words and status information are also transferred through the Data Bus Buffer.

Read/Write Control Logic: This functional block accepts inputs from the system control bus and generates control signals for overall device operation. It decodes control signals on the control bus into signals which controls the internal and external I/O bus. It contains the control word register and command word register that stores the various controls formats for the device functional definition.

Transmit Buffer: The transmit buffer accepts parallel data from the CPU, adds the appropriate framing information, serializes it, and transmits it on the TxD pin on the falling edge of TxC. It has two registers: A **buffer register** to hold eight bits and an **output register** to convert eight bits into a stream of serial bits. The CPU writes a byte in the buffer register, which is transferred to the output register when it is empty. The output register then transmits serial data on the TxD pin.

In the asynchronous mode the transmitter always adds START bit; depending on how the unit is programmed, it also adds an optional even or odd parity bit, and either 1,11/2, or 2 STOP bits. In synchronous mode no extra bits (other than parity, if enable) are generated by the transmitter.

Transmit Control: It manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

TxRDY (Transmit Ready): This output signal indicates CPU that buffer register is empty and the USART is ready to accept a data character. It can be used as an interrupt to the system or, for polled operation; the CPU can check TxRDY using the status read operation. This signal is reset when a data byte is loaded into the buffer register.

TxE (Transmitter Empty): This is an output signal. A high on this line indicates that the output buffer is empty. In the synchronous mode, if the CPU has failed to load a new character in time, TxE will go high momentarily as SYN characters are loaded into the transmitter to fill the gap in transmission.

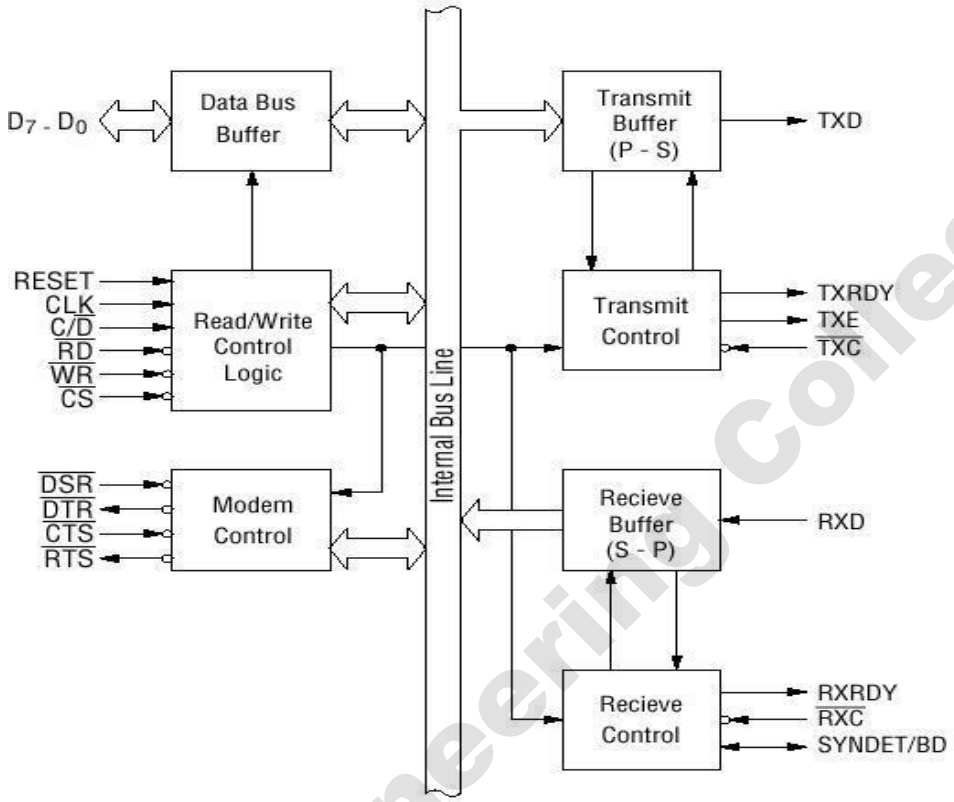
TxC (Transmitter Clock): This clock controls the rate at which characters are transmitted by USART. In the synchronous mode TxC is equivalent to the baud rate, and is supplied by the modem. In asynchronous mode TxC is 1, 16, or 64 times the baud rate. The clock division is programmable. It can be programmed by writing proper mode word in the mode set register.

Receive Buffer: The receiver accepts serial data on the RxD line converts this serial data to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU.

When 8251A is in the asynchronous mode and it is ready to accept a character, it looks for a low level on the RxD line. When it receives the low level, it assumes that it is a START bit and enables an internal counter. At a count equivalent to one-half of a bit time, the RxD line is sampled again. If the line is still low, a valid START bit is detected and the 8251A proceeds to assemble the character. After successful reception of a START bit the

8251A receives data, parity and STOP bits, and then transfers the data on the receiver input register. The data is then transferred into the receiver buffer register.

In the synchronous mode the receiver simply receives the specified number of data bits and transfers them to the receiver input register and then to the receiver buffer register.



Receive Control

It manages all receiver-related activities. Along with data reception, it does false start bit detection, parity error detection, framing error detection, sync detection and break detection.

RxRDY (Receiver Ready): This is an output signal. It goes high (active), when the USART has a character in the buffer register and is ready to transfer it to the CPU. This line can be used either to indicate the status in the status register or to interrupt the CPU. This signal is reset when a data byte from receiver buffer is read by the CPU.

RxC (Receiver Clock): This clock controls the rate at which the character is to be received by USART in the synchronous mode. RxC is equivalent to the baud rate, and is supplied by the modem. In asynchronous mod RxC is 1, 16, or 64 times the baud rate. The clock division is programmable. It can be programmed by writing proper mode word in the mode set register.

Modem Control

The 8251 has a set of control inputs and outputs that can be used to simplify the interface to almost any modem. It provides control circuitry for the generation of RTS and DTR and the reception of CTS and DSR. In addition, a general purpose inverted output and a general purpose input are provided. The output is labelled DTR and the input is labelled DSR. DTR can be asserted by setting bit 2 of the command instruction; DSR can be sensed as bit 7 of the

status register. When used as a modem control signal DTR indicates that the terminal is ready to communicate and DSR indicates that it is ready for communication.

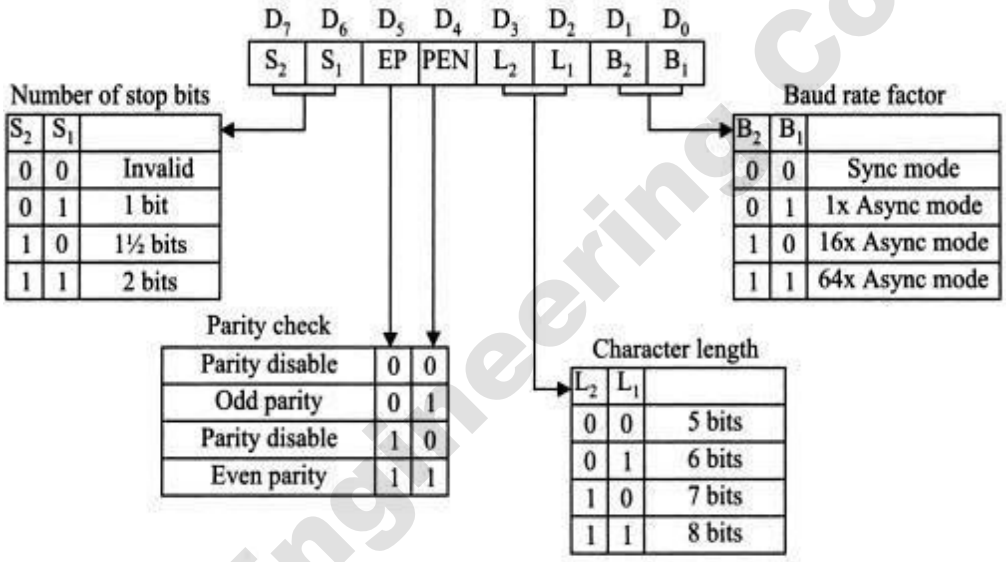
The receive control unit decides the receiver frequency as controlled by theRXC input frequency. The receive control unit generates a receiver ready (RXRDY) signal that may be used by the CPU for handshaking. This unit also detects a break in the data string while the 8251 is in asynchronous mode. In synchronous mode, the 8251 detects SYNC characters using SYNDET/BD pin.

Programming 8251

Prior to starting data transmission or reception the 8251 must be sent a set of control words. This must be done after an external or internal reset. The control words are split into two formats.

Mode Instruction Format and Command Word Format

Mode Instruction Format



The mode instruction format fixes up the baud rate, number of characters and stop bits for transmission.

D_1-D_0 determines whether the USART is to operate in the synchronous (00) or asynchronous mode. In the asynchronous mode, this field determines the division factor for clock to decide the baud rate.

D_3-D_2 determines number of data bits in one character. With this 2 bit field we can set character length from 5 bits to 8 bits.

D_5-D_4 controls the parity generation. The parity bit is added to the data bits only if parity is enabled.

D_7-D_6 has two meanings depending on whether operation is to be in synchronous mode or asynchronous mode. In asynchronous mode it controls the number of STOP bits to be transmitted. In synchronous mode it decides whether to operate with external synchronization or internal synchronization.

Command Instruction Format

It controls the operation of the USART. The command instruction controls the actual operations of the selected format like enable transmit/receive, error reset and modem control. A reset operation returns 8251 back to mode instruction format.

EH	IR	RTS	ER	SBRK	RxE	DTR	TxE

EH: Enter Hunt Mode
IR : Internal Reset
RTS: Request to Send
ER: Error Reset
SBPRK: Send Break Character
RxE : Receiver Enable
DTR: Data Terminal Ready
TxE : Transmitter Empty

Status Word

The status word enables us to read the status of the device during its operation.

DSR	SYNDET BRKDET	FE	OE	PE	TxE	RxRDY	TxRDY

DSR : Data Set Ready
TxE : Transmitter Empty
RxRDY : Receiver Ready
TxRDY: Transmitter Ready
FE: Framing Error
OE: Overrun Error
PE: Parity Error
SYNDET/BRKDET: Sync Detect/Break Error

TxRDY Transmitter Ready: This output signal indicates to the CPU that the internal circuit of the transmitter is ready to accept a new character for transmission from the CPU.

RxRDY Receiver Ready Output: This output indicates that the 8251A contains a character to be read by the CPU.

TXE Transmitter Empty: The TXE signal can be used to indicate the end of a transmission mode.

PE - Parity Error: At the time of transmission of data an even parity or odd parity is inserted in the data stream. At the receiver end, if parity of the character does not match with the predefined parity, parity error occurs.

OE - Overrun Error: In the receiver section received character is stored in the receive buffer. The CPU is supposed to read this character before reception of the next character. But if CPU fails in reading the character loaded in the receiver buffer the next received character replaces the previous one and the overrun error occurs.

FE - Framing Error: If valid stop bit is not detected at the end of each character framing error occurs.

SYNDET/BRKDET – Synchronous Detect / Break Detect :In synchronou s modes it is Synchronous Detect and it outputs a High to indicate the chip has detected the SYCN. Characters In "asynchronous mode," this is an output terminal which generates "high level" output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

DSR - Data Set Ready: This is normally used to check if data set is ready when communicating with a modem.

SIGNAL DESCRIPTION OF 8251

D0 – D7: This is an 8-bit data bus used to read or write status, command word or data from or to the 8251A.

C / D: (Control Word/Data): This input pin, together with RD and WR inputs, informs the 8251A that the word on the data bus is either a data or control word/status information. If this pin is 1, control / status is on the bus, otherwise data is on the bus.

RD: This active-low input to 8251A is used to inform it that the CPU is reading either data or status information from its internal registers.

WR: This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

CLK: This input is used to generate internal device timings and is normally connected to clock generator output. This input frequency should be at least 30 times greater than the receiver or transmitter data bit transfer rate.

RESET: A high on this input forces the 8251A into an idle state. The device will remain idle till this input signal again goes low and a new set of control word is written into it.

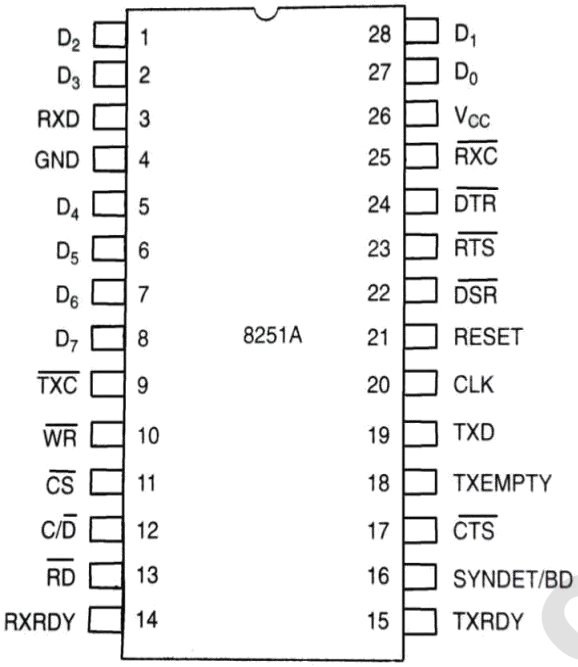
TXC (Transmitter Clock Input): This transmitter clock input controls the rate at which the character is to be transmitted.

TXD (Transmitted Data Output): This output pin carries serial stream of the transmitted data bits along with other information like start bit, stop bits and parity bit, etc.

RXC (Receiver Clock Input): This receiver clock input pin controls the rate at which the character is to be received.

RXD (Receive Data Input): This input pin of 8251A receives a composite stream of the data to be received by 8251 A.

RxRDY (Receiver Ready Output): This output indicates that the 8251A contains a character to be read by the CPU.



TxRDY - Transmitter Ready: This output signal indicates to the CPU that the internal circuit of the transmitter is ready to accept a new character for transmission from the CPU.

DSR - Data Set Ready: This is normally used to check if data set is ready when communicating with a modem.

DTR - Data Terminal Ready: This is used to indicate that the device is ready to accept data when the 8251 is communicating with a modem.

RTS - Request to Send Data: This signal is used to communicate with a modem.

TXE- Transmitter Empty: The TXE signal can be used to indicate the end of a transmission mode.

SYNDET/BRKDET – Synchronous Detect / Break Detect: In synchronous modes it is Synchronous Detect and it outputs a High to indicate the chip has detected the SYCN. Characters. In "asynchronous mode," this is an output terminal which generates "high level" output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

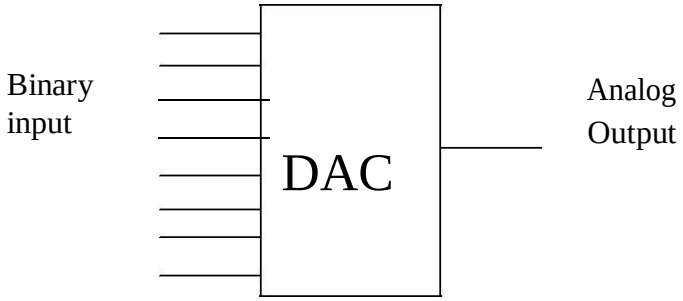
3a. INTERFACING DIGITAL TO ANALOG CONVERTERS

With Diagram, Explain The Operation Or R-2r Method Of D/A Converter. (8) [**Nov / Dec2012]**

Explain the different techniques to convert a digital quantity into its equivalent analog quantity. (8) [**Apr/May 2015**] [**May/Jun 2014**]

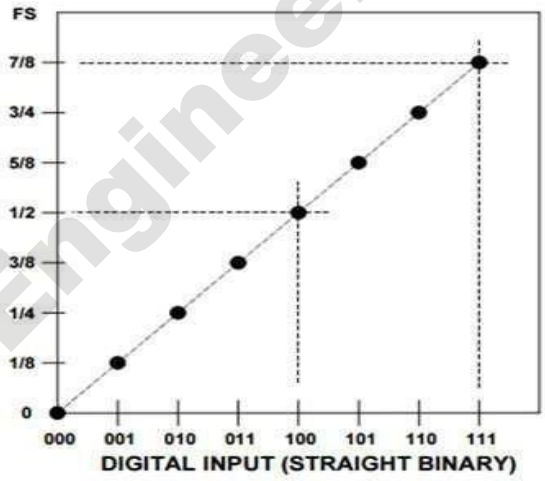
Explain how D/A and A/D interfacing done with 8086 with an application.(10) [**Apr/May 2015**]

A DAC inputs a binary number and outputs an analog voltage or current signal. The digital to analog converters converts binary numbers into their analog equivalent voltages or currents. Several techniques are employed for digital to analog conversion.



Basic Concepts

For a 3 bit D/A Converter it has 3 digital input D_2, D_1 and D_0 and one output analog signal. The three input lines can assume eight ($2^3 = 8$) input combinations from 000 to 111. D_2 is MSB and D_0 is LSB. If the input ranges from 0 to 1V it can be divided into eight equal parts(1/8 V) each successive input is 1/8 V higher than the previous combinations as shown in the graph below.



The 3 bit D/A converter have eight possible combinations. If a converter has n input lines it can have 2^n input combinations.

Characteristics:

Resolution: It is a change in analog output for one LSB change in digital input. It is given by $(1/2^n)$*Vref. If n=8 (i.e.8-bit DAC)1/256*5V=39.06mV

Settling Time: It is the time required for the DAC to settle for a full scale code change.

If the full scale analog voltage is 1 V, the smallest unit or the LSB 001 is equivalent to $1/2^n$ of 1V. This is defined as **resolution.**

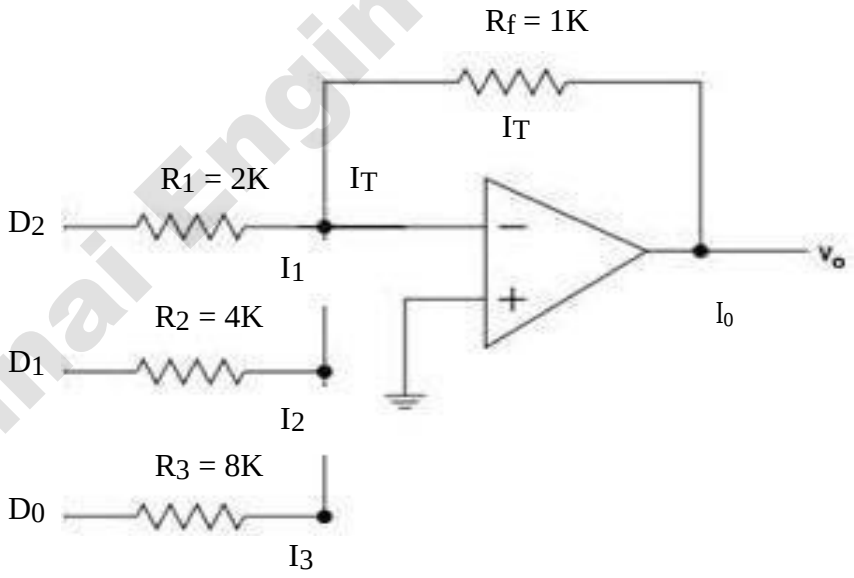
The DAC find applications in areas like digitally controlled gains, motor speed control, programmable gain amplifiers, digital voltmeters, panel meters, etc. D/A converter have many applications besides those where they are used with a microcomputer. In a compact disk audio player for example a 14or16bit D/A converter is used to convert the binary data read off the disk by a laser to an analog audio signal. Most speech synthesizer integrated circuits contain a D/A converter to convert stored binary data words into analog audio signals.

D/A Converter can be constructed by the following methods.

- Binary Weighted Resistor Network
- R-2R Ladder Network

Binary Weighted Resistor Network

The Binary Weighted DAC, which contains one resistor or current source for each bit of the DAC connected to a summing point. These precise voltages or currents sum to the correct output value. This is one of the fastest conversion methods but suffers from poor accuracy because of the high precision required for each individual voltage or current. Such high-precision resistors and current-sources are expensive, so this type of converter is usually limited to 8-bit resolution or less. The output of the DAC is current which is converted to a voltage by the operational amplifier at the output. If operational amplifier is used in a difference configuration, both positive and negative values may be obtained. The input resistors R_1, R_2 and R_3 are selected in binary weighted proportion; each has double the value of the previous resistor.



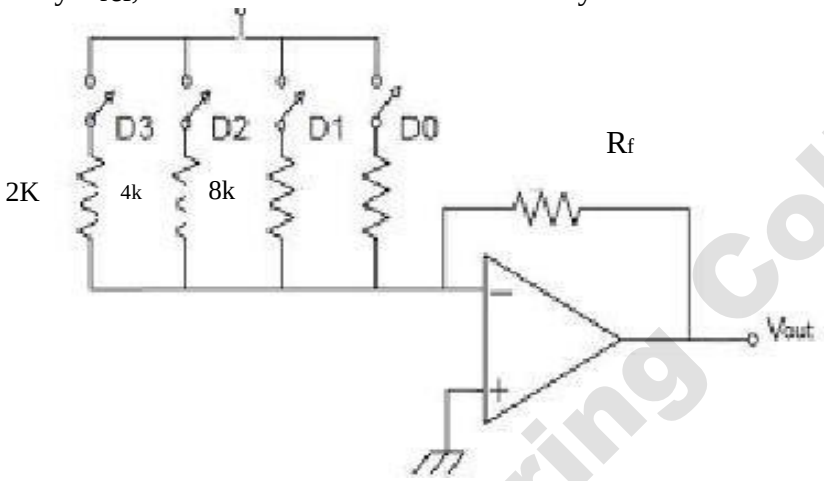
If all three inputs are 1 V the output current is

$$I_O = I_T = I_1 + I_2 + I_3$$
$$= V_{in}/R_1 + V_{in}/R_2 + V_{in}/R_3$$
$$= V_{in}/1\ K(½ + ¼ + 1/8)$$
$$= 0.875\ mA.$$

The voltage output

$$V_o = - R_f I_T$$
$$= - (1\ K\ ) (0.875)$$
$$= - 0.875\ V$$
$$= |7/8\ V|$$

It shows that for the input 111, the output is equal to either 7/8 mA or 7/7 V representing the D/A conversion process. The diagram is redrawn as shown below, where the input voltage V_{in} is replaced by V_{ref}, which can be turned On or OFF by the switches.



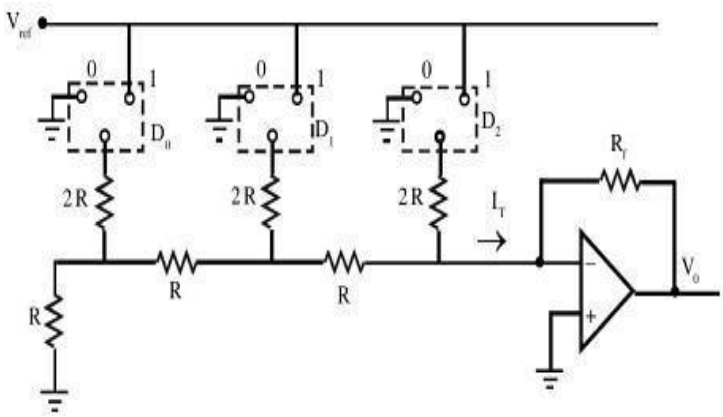
The output Current Io can be generalized for any number of bits as

$$I_o = \frac{V_{ref}}{R}\left(\frac{A}{2} + \frac{A}{4} + \frac{A}{8} + \frac{A}{16} + \frac{A}{32} + \frac{A}{64} + \frac{A}{128} + \frac{A}{256}\right)$$

where A_1 to A_8 can be 0 or 1

R-2R Ladder Network

The R-2R ladder DAC, which is a binary weighted DAC that uses a repeating cascaded structure of resistor values R and 2R. This improves the precision due to the relative ease of producing equal valued matched resistors (or current sources). However, wide converters perform slowly due to increasingly large RC-constants for each added R-2R link.

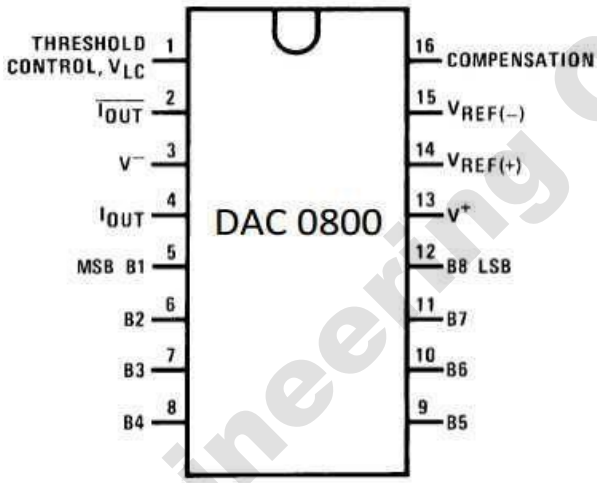


DAC 0800 8-bit Digital to Analog converter
Pin Diagram of DAC 0800:

- DAC0800 is a monolithic 8-bit DAC manufactured by National semiconductor.
- It has settling time around 100ms
- It can operate on a range of power supply voltage i.e. from 4.5V to +18V. Usually the supply V^+ is 5V or +12V. The V^- pin can be kept at a minimum of -12V.
- Resolution of the DAC is 39.06mV

DAC0800

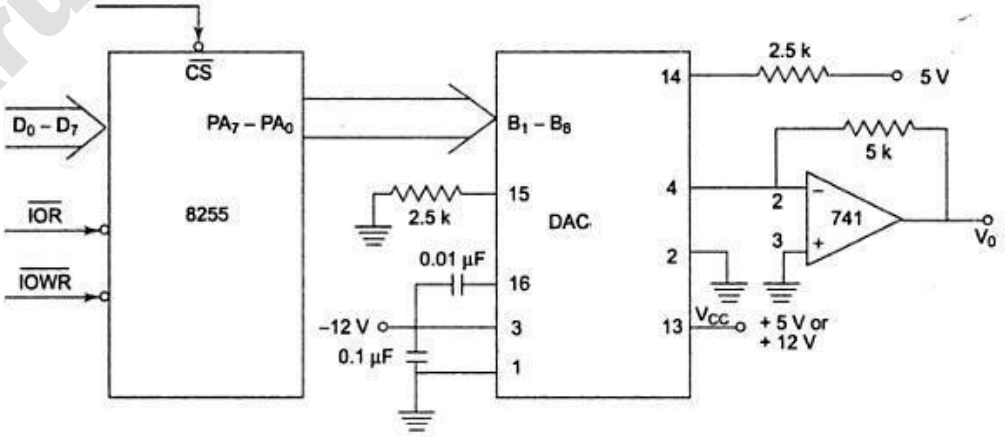
The digital inputs are converted to current I_{out}, and by connecting a resistor to the I_{out} pin, the output is converted to voltage. The total current I_{out} is a function of the binary numbers at the B0-B7



inputs of the DAC0808 and the reference current Iref , and it is given by:

$$= I_{ref} \times \frac{D_7}{2} + \frac{D_6}{4} + \frac{D_5}{8} + \frac{D_4}{16} + \frac{D_3}{32} + \frac{D_2}{64} + \frac{D_1}{128} + \frac{D_0}{256}$$

Usually reference current is 2mA. Ideally we connect the output pin to a resistor, convert this current to voltage, and monitor the output on the scope. But this can cause inaccuracy; hence an operational amplifier is used to convert the output current to voltage.



When chip select of DAC is enabled then DAC will convert digital input value given through portliness PA0-PA7 to analog value. The analog output from DAC is a current quantity. This current is converted to voltage using OPAMP based current-to-voltage converter. The Output of DAC-0800 is fed to the operational amplifier to get the final output.

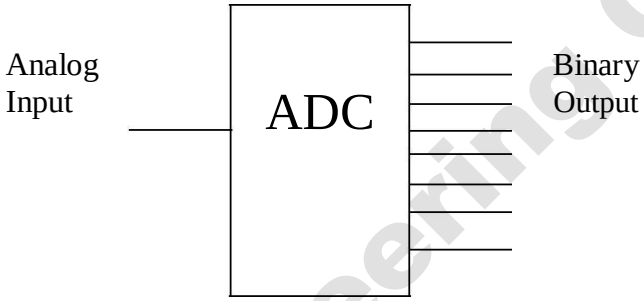
3b. INTERFACING ANALOG TO DIGITAL DATA CONVERTERS

Draw and explain the block diagram of A to D converter. (8) [**Nov /Dec 2013]**

Vin=2.25v, Vref=5v Number of data lines are 5. Convert the given analog quantity into its equivalent output digital quantity.(8) [**May/Jun 2014**]

Explain how D/A and A/D interfacing done with 8086 with an application.(10) [**Apr/May 2015**]

Vin=2.78v, Vref=5v Number of data lines are 6. Convert the given analog quantity into its equivalent **output digital quantity.(8) [Apr/May 2015]**



An ADC inputs an analog electrical signal such as voltage or current and outputs a binary number. The **function** of an A/D converter is to produce a digital word which represents the magnitude of some analog voltage or current. The **specifications** for an A/D converter are very similar to those for D/A converter. The resolution of an A/D converter refers to the number of bits in the output binary word. An 8-bit converter for example has a resolution of 1 part in 256. Accuracy and linearity specifications have the same meaning for an A/D converter as they do for a D/A converter. Another important specification for an ADC is its **conversion time**. This is defined as total time required to convert analog signal into its digital output and is determined by the conversion technique used and by the propagation delay in various circuits.

Algorithm for ADC interfacing contains the following steps.

- Ensure the stability of analog input, applied to the ADC.
- Issue start of conversion (SOC) pulse to ADC.
- Read end of conversion (EOC) signal to mark the end of conversion process.
- Read digital data output of the ADC as equivalent digital output.

Many different types of analog-to-digital converters are available. Differing ADC types offer varying resolution, accuracy and speed specifications. The most popular techniques used for

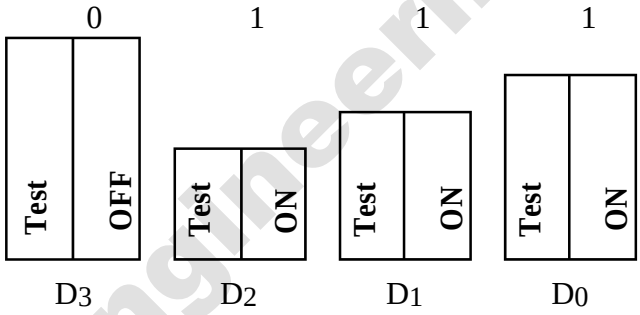
Analog-to-Digital conversions are

- Successive Approximation Method
- Dual Slope Method

SUCCESSIVE APPROXIMATION METHOD (ADC 0808/0809)

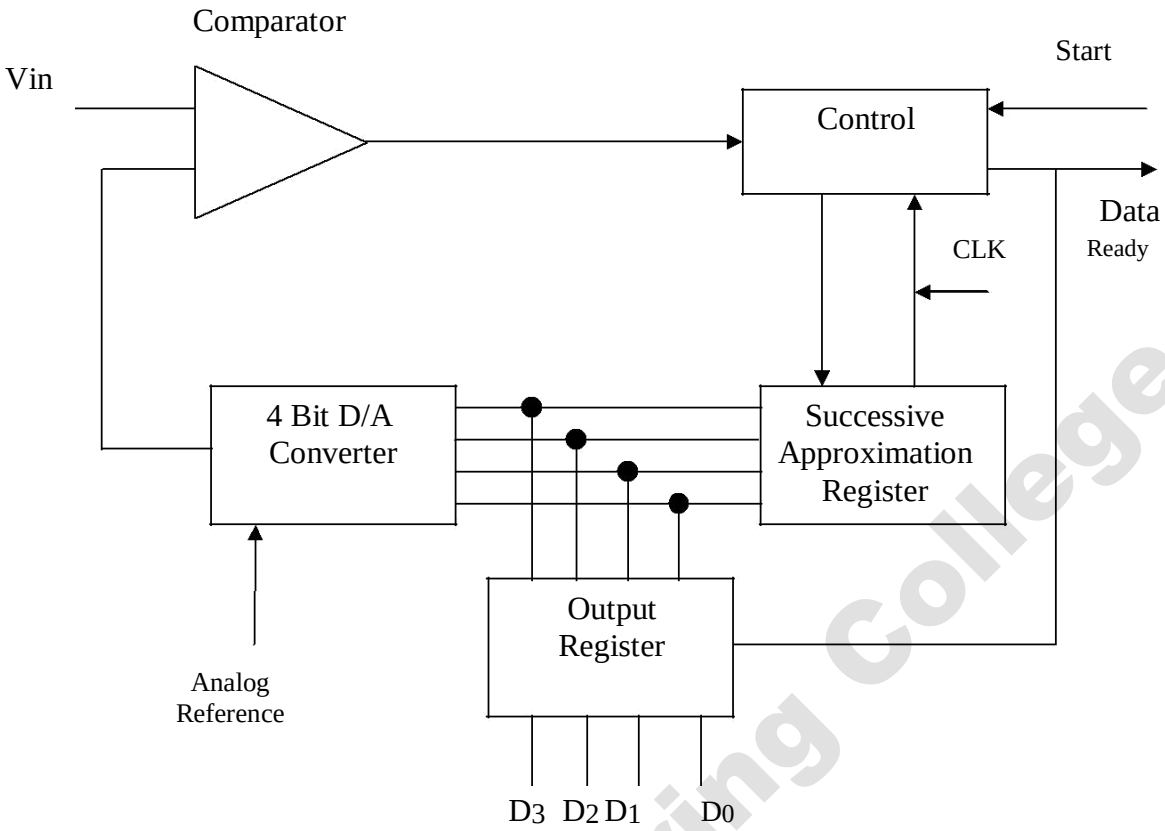
The method of generating input to the DAC is similar to weighing an unknown material on a chemical balance with a set of such fractional weights as ½ g, 1/4g, 1/8 g etc. The weighing procedure with a heaviest weight (1/2 g) and subsequent weights in decreasing order until the balance is tipped. The weight that tips the balance is removed and the process is continued until the smallest weight is used.

In the case of 4 bit A/D/ converter bit D_3bit is turned ON first and the output of the DAC is compared with an analog signal. If the comparator changes the state indicating that the output generated by d3 is larger than the analog signal bit D_3 is turned OFF in the SAR and the bit D_2 is turned ON. The process continues until the input reaches bit D_0.When bit D_3 is turned ON, the output exceeds the analog signal and therefore bit D_3 is turned OFF. When the next three successive bits are turned ON, the output becomes approximately equal to the analog signal.



The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, *successive approximation converters.* Successive approximation technique is one of the fast techniques for analog to digital conversion.

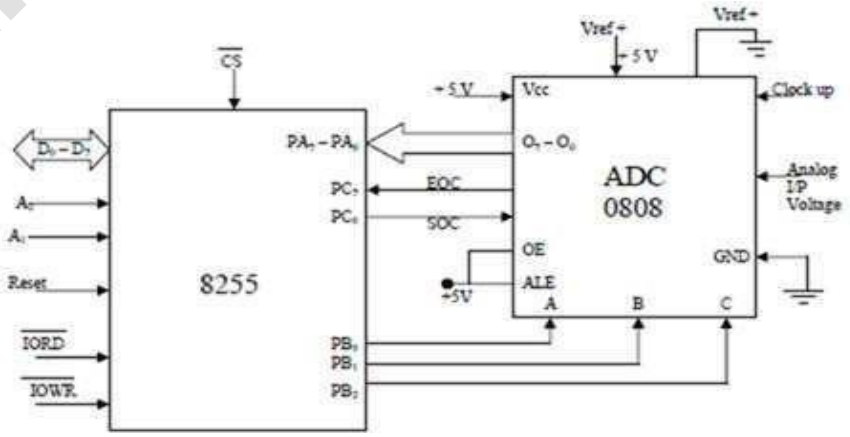
A successive approximation ADC employs a digital-to-analog converter (DAC) and a single comparator. A special shift register called a Successive Approximation Register (SAR) is used to control the DAC. It provisionally sets each bit of the DAC, beginning with the most significant bit. The search compares the output of the DAC to the voltage being measured. If setting a bit to one cause the DAC output to rise above the input voltage, that bit is set to zero. Otherwise, that bit is left unaltered. This process is continued for all the bits of the SAR. A Start Conversion (SOC) signal is provided, which when pulsed, initiates the conversion cycle. An N-bit ADC requires N clock cycles for the conversion of an analog input. When the conversion is complete, the binary result is placed on the parallel outputs of the SAR, and the SAR sends out an End-Of Conversion (EOC) signal. For continuous conversion, the EOC signal may be connected to the SOC signal.



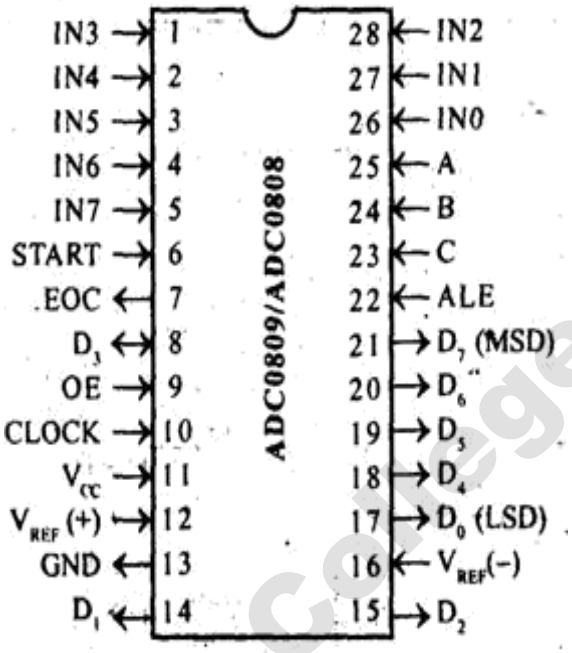
The time taken by the converter to calculate the equivalent digital data output from the instant of the start of conversion is called **conversion delay**. It may be noted that analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of conversion to get correct results.

Interfacing ADC 0808 with 8086 using 8255 ports.

Use port A of 8255 for transferring digital data output of ADC to the CPU and port C for control signals. Analog input is present at I/P 2 of the ADC and a clock input of suitable frequency is available for ADC. The analog input I/P 2 is used and therefore address pins A, B, C should be 0, 1, 0 respectively to select I/P 2. The OE and ALE pins are already kept at +5V to select the ADC and enable the outputs. Port C upper acts as the input port to receive the EOC signal while port C lower acts as the output port to send SOC to the ADC.



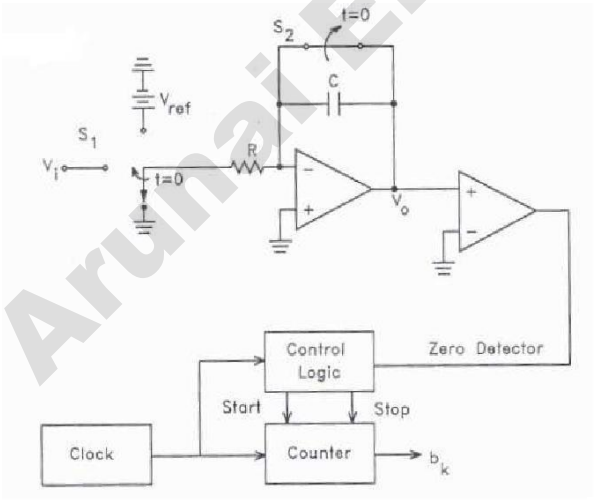
Analog I/P selected	Address lines		
	C	B	A
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1

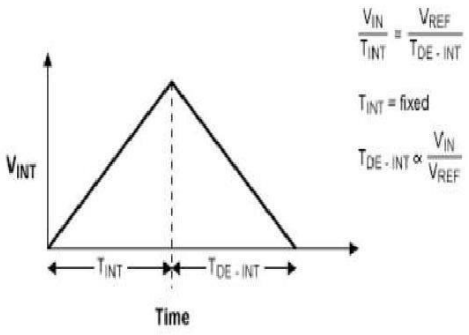


Address lines for selecting analog inputs

IN_7 - IN_0	Digital 8-bit output with O_7 MSB and O_0 LSB
SC	Start of conversion signal pin
EOC	End of conversion signal pin
OE	Output latch enable pin, if high enables output
CLK	Clock input for ADC
Vcc, GND	Supply pins +5V and GND
Vref+ and Vref-	Reference voltage positive (+5 Volts max.) and Reference voltage negative (OV minimum).

DUAL SLOPE A/D CONVERTER





A dual-slope ADC (DS-ADC) integrates an unknown input voltage (V_{IN}) for a fixed amount of time (T_{INT}), then "de-integrates" (T_{DE-INT}) using a known reference voltage (V_{REF}) for a variable amount of time.

At t<0, S_1 is set to ground, S_2 is closed, and counter=0.

At t=0 a conversion begins and $S_2$is open, and $S_1$is set so the input to the integrator is V_{in}.

$S_1$is held for T_{INT} which is a constant predetermined time interval.

When $S_1$is set the counter begins to count clock pulses, the counter resets to zero after T_{INT}

V_{out} of integrator at t = T_{INT} is V_{IN} T_{INT}/RC is linearly proportional to V_{IN}

At t = T_{INT} $S_1$is set so V_{ref} is the input to the integrator which has the voltage V_{IN} T_{INT} /RC stored in it.

The integrator voltage then drops linearly with a slope -V_{ref}/RC.

A comparator is used to determine when the output voltage of the integrator crosses zero when it is zero the digitized output value is the state of the counter.

4. 8253 PROGRAMMABLE INTERVAL TIMER

Draw the functional block diagram of 8254 timer and explain the different modes of operation. (8 Marks) **[April/May 2010]**

Draw and explain the block diagram of 8254 Programmable Interval Timer. Also explain the various modes of operation. [Marks 16] [**April/May 2011]**

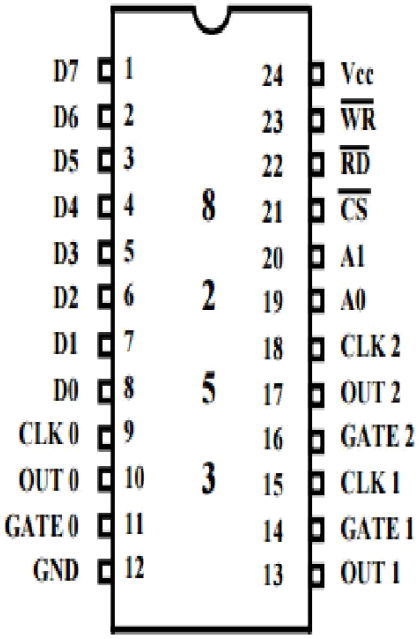
In how many modes we can use 8253/54 timer? Explain the different modes of operation of 8253/54 timer.(8) [**Nov/Dec 2014**].

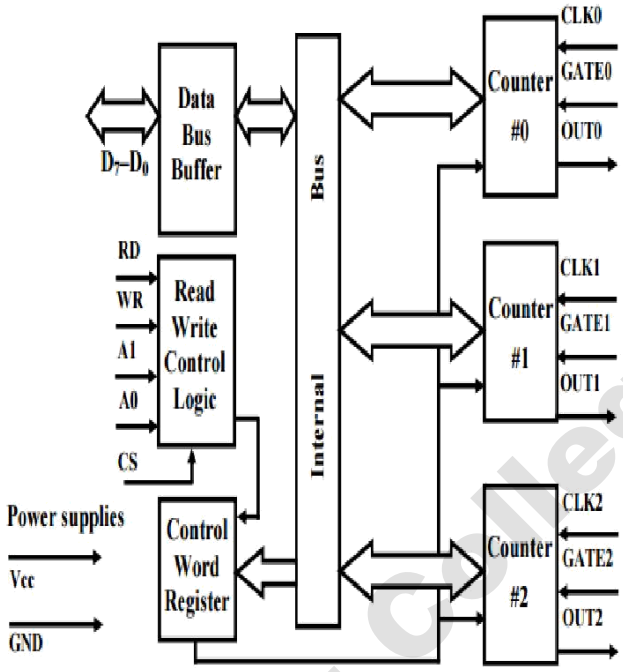
Explain the different modes of operation of a timer.(8) [**Apr/May 2015**] [**May/Jun 2014**]

INTEL 8254 programmable Timer/ counter is a specially designed chip for µC applications which require timing and counting operation. Each counter has two inputs, clock and gate and one output. The clock is signal that helps in counting by decrementing a preloaded value in the respective counter register. The gate serves as an enable input. If the gate is maintained low the counting is disabled.

Data Bus Buffer:

The data bus buffer is bidirectional, 8-bit buffer and is used to interface the 8253 to the system data bus. Data is transmitted or received by the buffer. The data bus buffer has three basic functions, (i) Programming the modes of 8253. (ii) Loading the count value in times (iii) Reading the count value from timers. The data bus buffer is connected to µ P which are also bidirectional. The data transfer is through these pins.





Read/ Write Control Logic:

It accepts inputs for the system control bus and in turn generate the control signals for overall device operation.

CS: The chip select input is used to enable the communication between 8253 and the microprocessor by means of data bus. A low on CS enables the data bus buffers, while a high disables the buffer

RD &WR: The read (RD) and write (WR) pins control the direction of data transfer on the 8-bit bus. When the RD input pin is low. The CPU is inputting data from 8253 in the form of counter value. When WR pins is low, then CPU is sending data to 8253 in the form of mode information or loading counters. The RD &WR should not both be low simultaneously. When RD & WR pins are HIGH, the data bus buffer is disabled.

A0 & A1: These two input lines are used for counter selection along with the CS pin.

A0	A1	Selected
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Register

Counters: Each counter has three pins associated with it. They are CLK (CLK) the gate (GATE) and the output (OUT).

CLK: Counters operate at the negative edge (1 to 0) of this clock input. If the signal on this pin is generated by a fixed frequency oscillator then the user has implemented a standard timer. If the input signal is a string of randomly occurring pulses, then it is an implementation of a counter.

GATE: The gate input pin is used to initiate or enable counting. The exact effect of the gate signal depends on which of the six modes of operation is chosen.

OUTPUT: The output pin provides an output from the timer. It actual use depends on the mode of operation of the timer. The counter can be read "in the fly" without inhibiting gate pulse or clock input.

Programming the Chip

All operations are decided by the control word loaded into the control register. For each counters, there is a count register which is 16 bits in size. A number is written into this register and stored in counter block. The maximum size of the number is FFFF H. The operation of the counter occurs by creating a delay by decrementing this number down to 0, the rate at which this occurs depends on the input clock frequency.

Status Register

OUT	**NULL**	**RW1**	**RW0**	**M2**	**M1**	**M0**	**BCD**

OUT: The level of the OUT Pin
NULL = 1 if counter is 0
RW1.RW0: Read/write Operation
M2, M1, M0: Counter Mode
BCD: Logic 1 for BCD counter

Control Word Register:

It accepts information from the data bus buffer and stores it in a register. The information stored in the register controls the operation mode, selection of binary or BCD counting, Selection of counter and loading the values in the count register.

SCI	SCO	RL1	RL0	M2	M1	M0	BCD

BCD	
0	Binary
1	BCD

SCI	**SCO**	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Select Counter 3

RL1	**RL0**	
0	0	Counter Latching
0	1	Read LSB
1	0	Read MSB
1	1	Read LSB,MSB

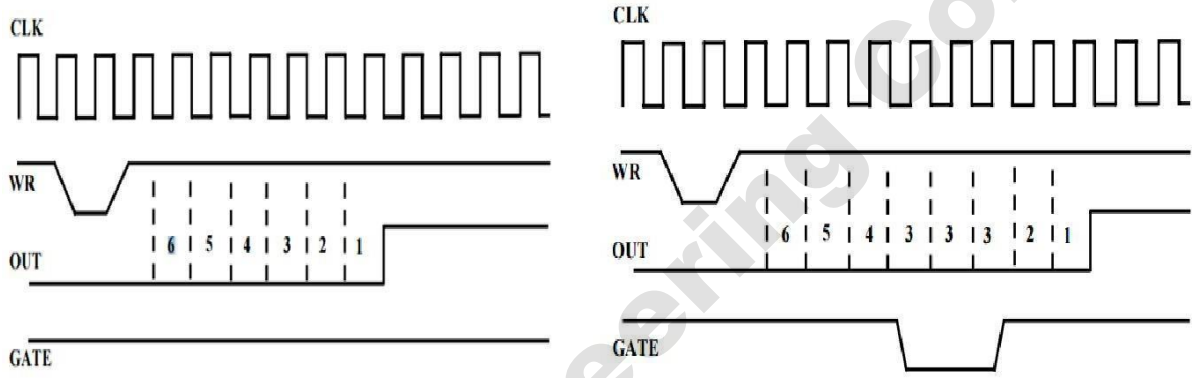
M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
x	1	0	Mode 2
x	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

8253 OPERATING MODES

Mode 0 Interrupt on Terminal Count
Mode 1 Programmable One Shot
Mode 2 Rate Generator
Mode 3 Square Wave Generator
Mode 4 Software Triggered Strobe
Mode 5 Hardware Triggered Strobe

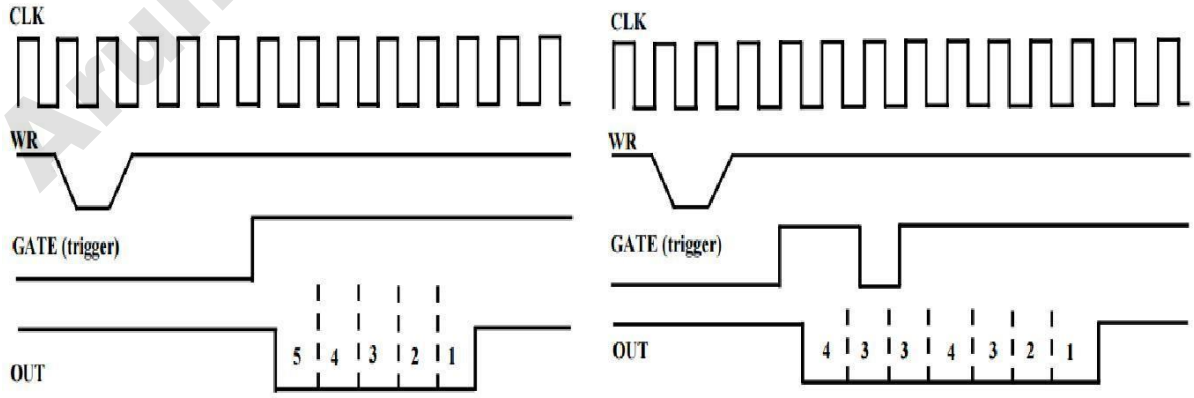
Mode 0 Interrupt on Terminal Count

The output goes high after the terminal count is reached. The counter stops if the Gate is low. The timer count register is loaded with a count (say 6) when the WR line is made low by the processor. The counter unit starts counting down with each clock pulse. The output goes high when the register value reaches zero. In the meantime if the GATE is made low the count is suspended at the value(3) till the GATE is enabled again.



Mode 1 Programmable One Shot

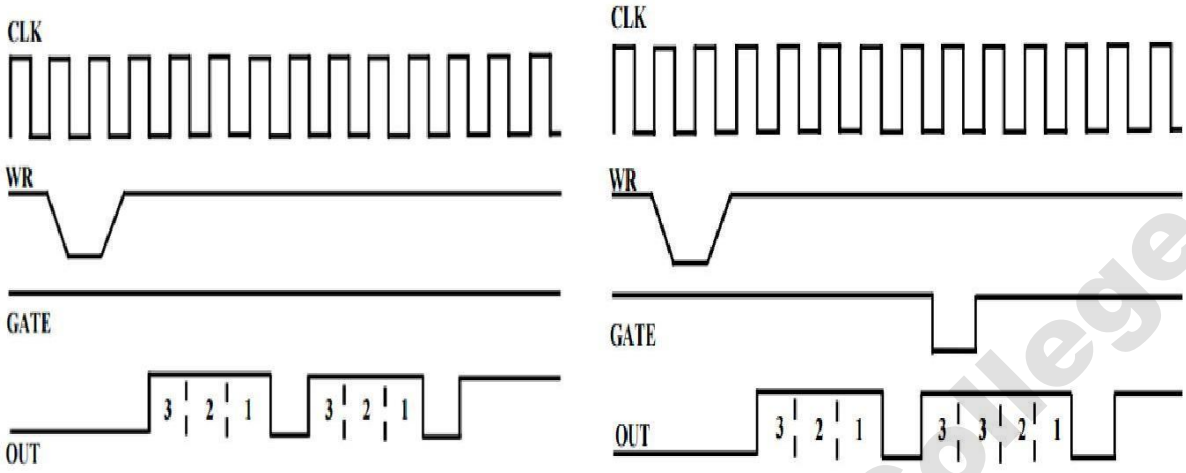
The output goes low with the Gate pulse for a predetermined period depending on the counter. The counter is disabled if the GATE pulse goes momentarily low. The counter register is loaded with a count value as in the previous case (say 5) The output responds to the GATE input and goes low for period that equals the countdown period of the register (5 clock pulses in this period). By changing the value of this count the duration of the output pulse can be changed. If the GATE becomes low before the countdown is completed then the counter will be suspended at that state as long as GATE is low. Thus it works as a mono-shot.



Mode 2 Rate Generator

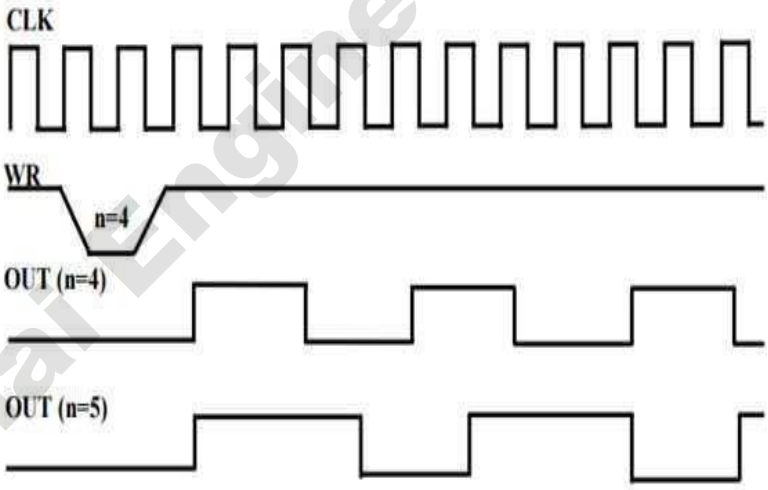
In this mode it operates as a rate generator. The output goes high for a period that equals the

time of countdown of the count register (3 in this case). The output goes low exactly for one clock period before it becomes high again. This is a periodic operation.



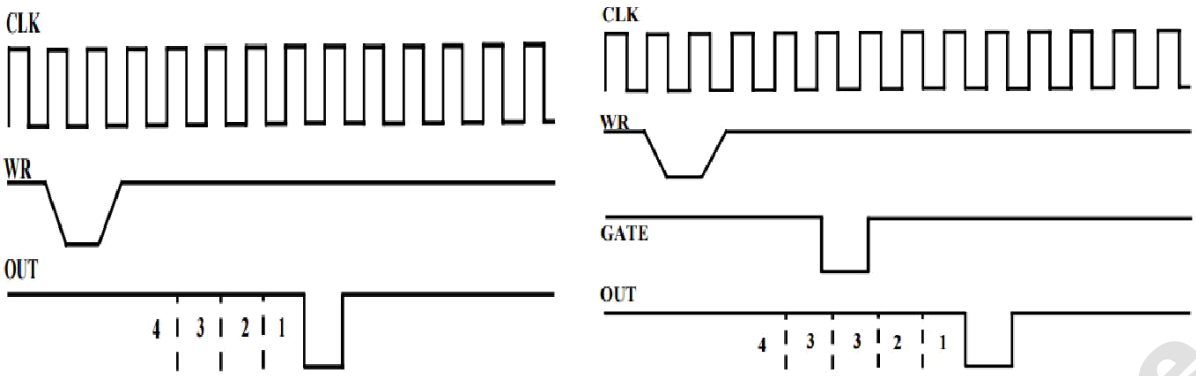
Mode 3 Square Wave Generator

It is similar to Mode 2 but the output high and low period is symmetrical. The output goes high after the count is loaded and it remains high for period which equals the countdown period of the counter register. The output subsequently goes low for an equal period and hence generates a symmetrical square wave unlike Mode 2. The GATE has no role here.



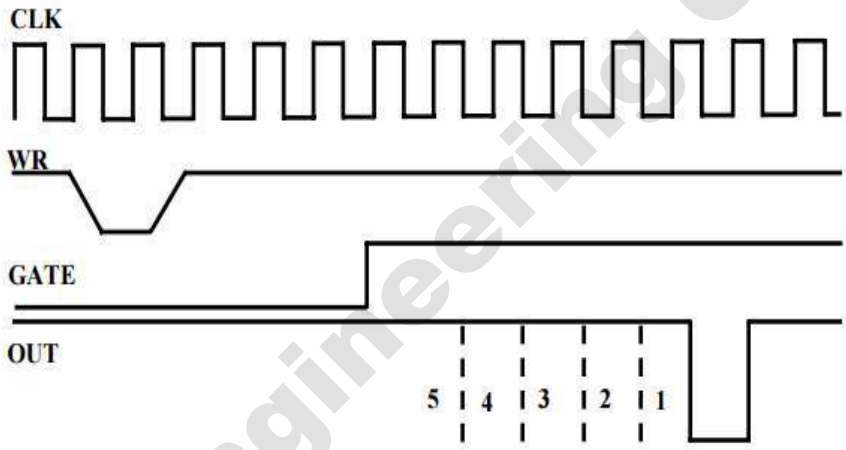
Mode 4 Software Triggered Strobe

In this mode after the count is loaded by the processor the countdown starts. The output goes low for one clock period after the countdown is complete. The countdown can be suspended by making the GATE low. This is also called a software triggered strobe as the countdown is initiated by a program.



Mode 5 Hardware Triggered Strobe

The count is loaded by the processor but the countdown is initiated by the GATE pulse. The transition from low to high of the GATE pulse enables count down. The output goes low for one clock period after the countdown is complete



Watchdog timer

A Watchdog Timer is a circuit that automatically invokes a reset unless the system being watched sends regular hold-off signals to the Watchdog.

5. KEYBOARD AND DISPLAY CONTROLLER (8279)

Draw the block diagram of 8279 keyboard/ Display controller and explain how to interface the Hex Key pad and 7- segment LEDs using 8279. (16 Marks) **[April/May 2010, April/May2017]**

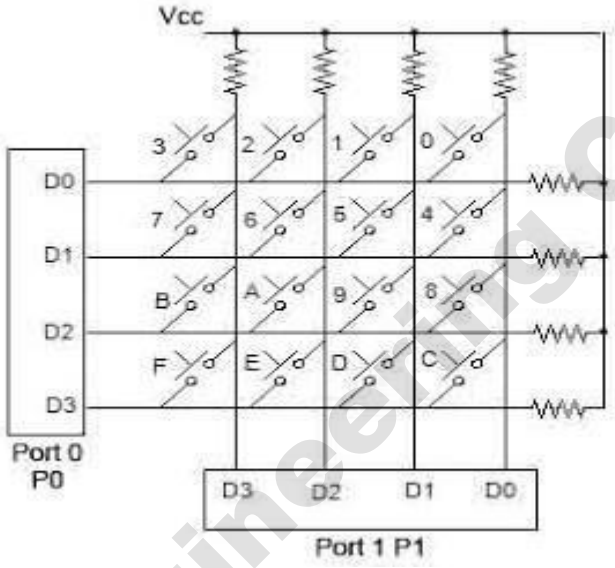
Draw the block diagram of a keyboard display controller and explain(8) [**Nov/Dec 2014**].

Intel's 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interfaces a keyboard with the CPU, leaving it free for its routine task

Basics of Keyboard Interfacing:

Matrix keyboards are connected in a series of rows and columns. The important tasks in interfacing a keyboard are 1) detecting a key press, 2) debounce the key press and 3) encode the key to some standard code. Three tasks can be done with hardware, software, or a combination of two, depending on the application.

Keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and columns through ports. Therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor. When a key is pressed, a row and a column make a contact. Otherwise, there is no connection between rows and columns. A 4x4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port.



Scanning and Identifying the Key:

It is the function of the microprocessor to scan the keyboard continuously to detect and identify the key pressed

- To detect a pressed key, grounds all rows by providing 0 to the output latch, then it reads the columns
- If the data read from columns is $D_3 – D_0$ =1111, no key has been pressed and the process continues till key press is detected
- If one of the column bits has a zero, this means that a key press has occurred For example, if $D_3 – D_0$ = 1101, this means that a key in the D1 column has been pressed After detecting a key press, microprocessor will go through the process of identifying the key
- Starting with the top row, the microprocessor grounds it by providing a low to row D_0 only. It reads the columns, if the data read is all 1s, no key in that row is activated and the process is moved to the next row
- It grounds the next row, reads the columns, and checks for any zero. This process continues until the row is identified.
- After the key press detection, it waits 20ms for the **key debounce** and then scans the

columns again

(a) It ensures that the first key press detection was not an erroneous one due a spike noise

(b) The key press. If after the 20-ms delay the key is still pressed, it goes back into the loop to detect a real key press

- Upon finding the zero, it pulls out the ASCII code for that key from the look-up table otherwise, it increments the pointer to point to the next element of the look-up table

With the interrupt method the microcomputer doesn't have to pay any attention to the keyboard until it receives an interrupt signal.

Modes of Operation

- **Two-Key Rollover**. This means that if two keys are pressed at nearly the same time, each key will be detected, debounced and converted to ASCII. The ASCII code for the first key and a strobe signal for it will be sent out then the ASCII code for the second key and a strobe signal for it will be sent out and compare this with two-key lockout.
- **2-Key Lockout Mechanism**, one key must be released before the other key is detected.
- **N-Key Rollover Mode**, if two keys are pressed almost simultaneously, both key presses are detected and are placed in a queue

ARCHITECTURE OF 8279

The keyboard display controller 8279 provides:

a) A set of four scan lines and eight return lines for interfacing keyboards

b) A set of eight output lines for interfacing display.

I/O Control and Data Buffers:

The I/O control section controls the flow of data to/from the 8279. The data buffers interface the external bus of the system with internal bus of 8279.The I/O section is enabled only if CS is low. The pins A0, RD and WR select the command, status or data read/write operations carried out by the CPU with 8279.

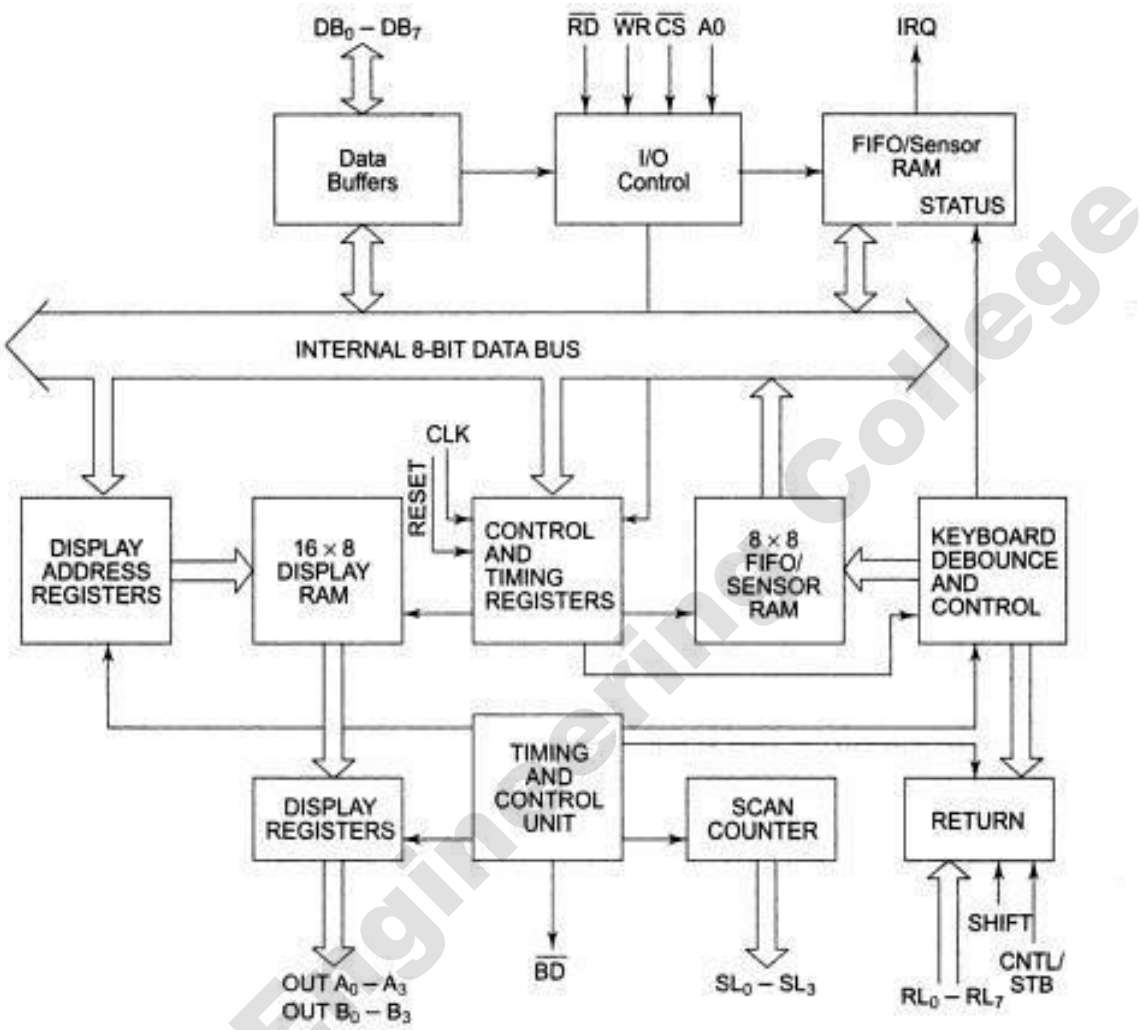
Control and Timing Register and Timing Control:

These registers store the keyboard and display modes and other operating conditions programmed by CPU. The registers are written with A 0=1 and WR=0. The Timing and control unit controls the basic timings for the operation of the circuit. Scan counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies.

Scan Counter:

The scan counter has two modes to scan the key matrix and refresh the display. In the **encoded mode**, the counter provides binary count that is to be externally decoded to provide the scan lines for keyboard and display (Four externally decoded scan lines may drive upto 16 displays). In the **decode scan** mode, the counter internally decodes the least significant 2

bits and provides a decoded 1 out of 4 scan on SL 0-SL 3(Four internally decoded scan lines may drive up to 4 displays). The keyboard and display both are in the same mode at a time.



Return Buffers and Keyboard Debounce and Control:

If a key closer is detected, the keyboard debounce unit debounces the key entry (i.e. wait for 10 ms). After the debounce period, if the key continues to be detected. The code of key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

FIFO/Sensor RAM and Status Logic:

In keyboard or strobed input mode, this block acts as 8-byte first-in-first out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry and in the meantime read by the CPU, till the RAM become empty. The status logic generates an interrupt after each FIFO read operation till the FIFO is empty. In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

Display Address Registers and Display RAM:

The display address register holds the address of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU.

MODES OF OPERATION OF 8279

Input (Keyboard) Modes and Output (Display) Modes

Input (Keyboard) Modes

1. Scanned Keyboard Mode:

This mode allows a key matrix to be interfaced using either encoded or decoded scans. In the encoded scan, an 8 x 8 keyboard or in decoded scan, a 4 x 8 Keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.

2. Scanned Sensor Matrix:

In this mode, a sensor array can be interfaced with 8279 using either encoder or decoder scans. With encoder scan 8 x 8 sensor matrix or with decoder scan 4 x 8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.

3. Strobed Input:

In this mode, if the control line goes low, the data on return lines is stored in the FIFO byte by byte.

Output (Display) Modes

Provides two output modes for selecting the display option.

Display Scan: 8279 provides 8 or 16 character multiplexed displays.

Display Scan: Options for data entry on the displays. The display data is entered for display either from right side or from the left side.

DETAILS OF MODE OF OPERATION

1. Scanned Keyboard Mode with 2 Key Lockout

In this mode of operation, when a key is pressed, a debounce logic comes into operation. The Key code of the identified key is entered into the FIFO with SHIFT and CNTL status, provided the FIFO is not full.

2. Scanned Keyboard with N-key Rollover

In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboard scans and then checks whether the key is still

depressed. If it is still depressed, the code is entered in FIFO RAM. Any number of keys can be pressed simultaneously and recognized in the order, the Keyboard scan and record them.

3. Scanned Keyboard Special Error Mode

This mode is valid only under the N-Key rollover mode. This mode is programmed using end interrupt/error mode set command. If during a single debounce period (two Keyboard scan) two keys are found pressed, this is considered a simultaneous depression and an error flag is set. This flag, if set, prevents further writing in FIFO but allows generation of further interrupts to the CPU for FIFO read.

4. Sensor Matrix Mode

In the Sensor Matrix mode, the debounce logic is inhibited the 8-byte memory matrix. The status of the sensor switch matrix is fed directly to sensor RAM matrix Thus the sensor RAM bits contains the row-wise and column-wise status of the sensors in the sensor matrix.

DISPLAY MODES

There are various options of data display The first one is known as left entry mode or type writer mode. Since in a type writer the first character typed appears at the left-most position, while the subsequent characters appears successively to the right of the first one. The other display format is known as right entry mode, or calculator mode, since the calculator the first character entered appears at the right-most position and this character is shifted one position left when the next character is entered.

1. Left Entry Mode

In the Left entry mode, the data is entered from the left side of the display unit. Address0 of the display RAM contains the leftmost display character and address 15 of the RAM contains the rightmost display character.

2. Right Entry Mode

In the right entry mode, the first entry to be displayed is entered on the rightmost display. The next entry is also placed in the right most display but after the previous display is shifted left by one display position.

Command Words of 8279

All the command words or status words are written or read with A0 = 1 and CS = 0 to or from 8279. This section describes the various command available in 8279.

a) **Keyboard Display Mode Set** – The format of the command word to select different modes of operation of 8279 is given below with its bit definitions.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	D	D	K	K	K

D	D	Display Modes
0	0	Eight 8 bit Character Left Entry
0	1	Sixteen 8 bit Character Left Entry
1	0	Eight 8 bit Character Right Entry
1	1	Sixteen 8 bit Character Right Entry

K	K	K	Keyboard Modes
0	0	0	Encoded Scan 2 Key Lockout
0	1	0	Decoded Scan 2 Key Lockout
0	1	0	Encoded Scan N Key Roll Over
0	1	1	Decoded Scan N Key Roll Over
1	0	0	Encoded Scan Sensor Matrix
1	1	0	Decoded Scan Sensor Matrix
1	1	0	Strobed input Encoded Scan
1	1	1	Strobed input Decoded Scan

b) **Read FIFO / Sensor RAM:**

The format of this command is given below. This word is written to set up 8279 for reading FIFO/ sensor RAM. In scanned keyboard mode, AI and AAA bits are of no use. The 8279 will automatically drive data bus for each subsequent read, in the same sequence, in which the data was entered. In sensor matrix mode, the bits AAA select one of the 8 rows of RAM. If AI flag is set, each successive read will be from the subsequent RAM location.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	AI	X	A	A	A

AI – Auto increment

AAA – Address pointer to 8 bit FIFO RAM

c) **Read Display RAM:**

This command enables a programmer to read the display RAM data. The CPU writes this command word to 8279 to prepare it for display RAM read operation. AI is auto increment flag and AAAA, the 4-bit address points to the 16-byte display RAM that is to be read. If AI=1, the address will be automatically, incremented after each read or write to the Display RAM. The same address counter is used for reading and writing.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	1	AI	A	A	A	A

d) **Write Display RAM:**

AI – Auto increment Flag. AAAA – 4 bit address for 16-bit display RAM to be written.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	AI	A	A	A	A

SIGNALS OF 8279

DB0-DB7: These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.

CLK: This is a clock input used to generate internal timing required by 8279.

RESET: This pin is used to reset 8279. A high on this line reset 8279. After resetting 8279, it's in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.

CS: Chip Select – A low on this line enables 8279 for normal read or write operations.

A0: A high on this line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.

RD, WR (Input/output) READ/WRITE – These input pins enable the data buffers to receive or send data over the data bus.

IRQ: This interrupt output lines goes high when there is a data in the FIFO sensor RAM. The interrupt lines goes low with each FIFO RAM read operation but if the FIFO RAM

further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

Vss, Vcc : These are the ground and power supply lines for the circuit.

SL0-SL3-Scan Lines: These lines are used to scan the key board matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.

RL0 - RL7 - Return Lines: These are the input lines which are connected to one terminal of keys, while the other terminal of the keys are connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.

SHIFT: The status of the shift input lines is stored along with each key code in FIFO, in scanned keyboard mode. It is pulled up internally to keep it high, till it is pulled low with a key closure.

BD – Blank Display: This output pin is used to blank the display during digit switching or by a blanking closure.

OUT A0 – OUT A3 and OUT B0 – OUT B3 – These are the output ports for two 16*4 or 16*8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also as one 8-bit port.

CNTL/STB- CONTROL/STROBED I/P Mode: In keyboard mode, this lines is used as a control input and stored in FIFO on a key closure. The line is a strobed lines that enters the data into FIFO RAM, in strobed input mode. It has an interrupt pull up. The lines are pulled down with a key closer.

6. PROGRAMMABLE INTERRUPT CONTROLLER 8259A

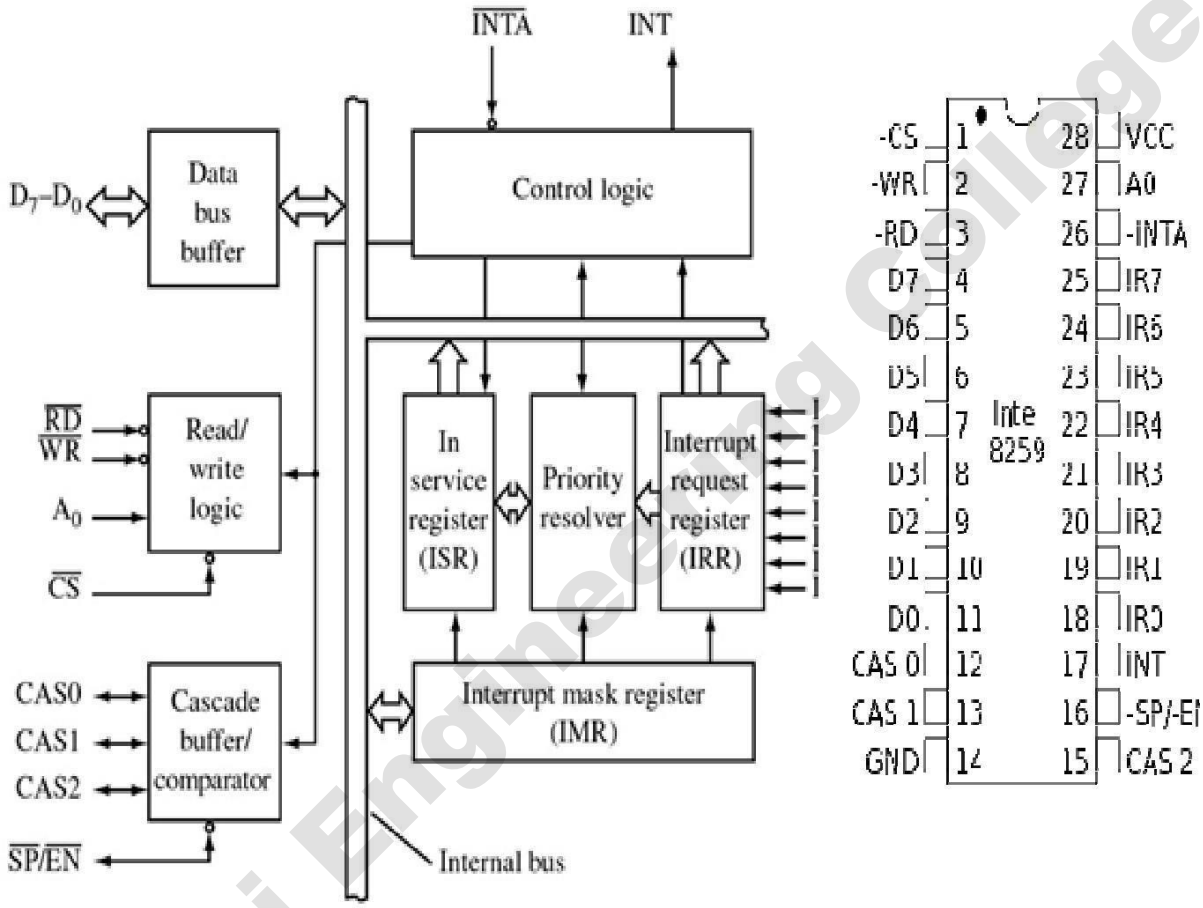
Draw the block diagram of 8259A and explain how to program 8259A. (8 Marks) **[April/May 2010]**
Describe the block diagram of 8259 Programmable Interrupt Controller and its priority modes. (16) [**Nov/Dec 2011]**

Programmable interrupt controller 8259A which is able to handle a number of interrupts at a time. This controller takes care of a number of simultaneously appearing interrupt requests along with their types and priorities. This will reduce the processor burden of handling interrupts. The 8259 A interrupt controller can

1) Handle eight interrupt inputs. This is equivalent to providing eight interrupt pins on the processor in place of one INTR/INT pin.
2) All the eight interrupt are spaced at the interval of either four or eight location.
3) Resolve eight levels of interrupt priorities in a variety of modes.

4) Mask each interrupt request individually.
5) Read the status of pending interrupts, in service interrupts, and masked interrupts.
6) Be set up to accept either the level triggered or edge triggered interrupt request.
7) Nine 8259 as can be cascaded in a master slave configuration to handle 64 interrupt inputs.

ARCHITECTURE OF 8259



Data Bus Buffer

This tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor system data bus. Control words, status and vector information pass through buffer during read or write operations.

Read /Write Control Logic

This circuit accepts and decodes commands from the CPU. This also allows the status of the 8259A to be transferred on to the data bus.

Interrupt Request Register (IRR)

The interrupts at IRQ input lines are handled by Interrupt Request Register internally. IRR stores all the interrupt requests in it in order to serve them one by one on the priority basis.

In-Service Register (ISR)
These stores all the interrupt requests those are being served, i.e ISR keeps a track of the requests being served.

Priority Resolver
This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the corresponding bit of ISR during INTA pulse. The IR0 has the highest priority while the IR7 has the lowest one

Interrupt Mask Register (IMR)
This register stores the bits required to mask the interrupt puts. IMR operates on IRR at the direction of the Priority Resolver.

Control Logic
This block manages the interrupt and interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.

Cascade Buffer/Comparator
This block stores and compares the ID's of all the 8259As used in the system. The three I/O pins CAS0-2 are outputs when the 8259A is used as a master. The same pins acts as input when 8259 is in slave mode.

The Interrupt sequence in an 8086-8259A system is described as follows:

1. One or more IR lines are raised high that set corresponding IRR bits.
2. 8259A resolves priority and sends an INT signal to CPU.
3. The CPU acknowledge with INTA pulse.
4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data during this period.
5. The 8086 will initiate a second INTA pulse. During this period 8259A releases an 8-bit pointer on to a data bus from where it is read by the CPU.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if automatic end of interrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

PIN DIAGRAM DESCRIPTION

CS: This is an active low chip select signal for enabling RD and WR operations of 8259A.

WR: This pin is an active low write enable input to 8259A. This enables it to accept command words from CPU.

RD: This is an active low read enable input to 8259A. A low on this line enables 8259A to release status onto the data bus of CPU.

D7-D0: These pins form a bidirectional data bus that carries 8-bit data either to control word or from status word registers. This also carries interrupt vector information.

CAS0-CAS2: Cascade Lines A single 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in cascade mode.

PS*/EN*: This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as buffer enable to control buffer transceivers. If this is not used in buffered mode then the pin is used as input.

INT This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.

IR0-IR7 (1nterrupt Requests) These pins act as inputs to accept interrupt requests to the CPU. In edge triggered mode, an interrupt service is requested by raising an IR pin from a low to a high state and holding it high until it is acknowledged

INTA* (Interrupt Acknowledge) This pin is an input used to strobe-in 8259A interrupt vector data on to the data bus

Command Words of 8259A

The command words of 8259A are classified in two groups,

- Initialization Command Words (ICWs)
- Operation command words (OCWs)

Initialization Command Words (ICWs)

8259A must be initialized by writing two to four command words into the respective command word registers. These are called as initialization command words (ICWs).

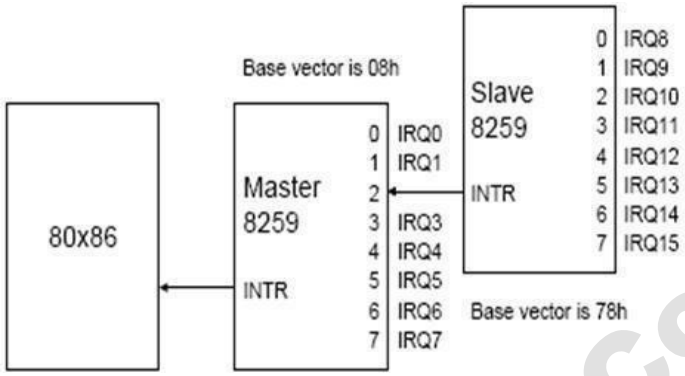
ICW1, ICW2, ICW3, ICW4 (Status Register)

ICW1 Initialization Command Word1

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC_4

LTIM 1: Level triggered
0: Edge Triggered
ADI: Call address interval
1: Interval of 4 bytes
0: Interval of 8 bytes

$A_7 – A_5$: Interrupt Vector Address
IC_4 1: ICW4 Needed
0: Not Needed
SNGL 1: Single
0: Cascaded



The SNGL bit in ICW1 indicates whether the 8259A in the cascade mode or not. ADI refers the interval of call address. LTIM refers whether it is edge triggered or level triggered.

ICW2 Initialization Command Word2

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3	A_{10}	A_9	A_8

In 8086 based system A_{15}-A_{11} of the interrupt vector address are inserted in place of $T_7 – T_3$ respectively and the remaining three bits A_8, A_9, A_{10} are selected depending upon the interrupt level, i.e. from 000 to 111 for IR_0 to IR_7.

ICW3 Initialization Command Word3

The ICW 3 loads an 8-bit slave register. It detailed functions are as follows. In master mode [SP = 1 or in buffer mode M/S = 1 in ICW 4], the 8-bit slave register will be set bit-wise to 1 for each slave in the system. The requesting slave will then release the second byte of a CALL sequence. In slave mode [SP=0 or if BUF =1 and M/S = 0 in ICW4] bits D2 to D0 identify the slave, i.e. 000 to 111 for slave 1 to slave 8. The slave compares the cascade inputs with these bits and if they are equal, the second byte of the CALL sequence is released by it on the data bus.

Master Mode of ICW3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

$S_{n:\,1}$ has a slave: 0 does not have a slave

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

ID_2, ID_1, ID_0: 000 to 111 for IR_0 to IR_7

ICW4 Initialization Command Word4

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M/S	AEOI	µPM

The bit functions of ICW4 are described as follow:

SFNM: If BUF = 1, the buffered mode is selected. In the buffered mode, SP/EN acts as enable output and the master/slave is determined using the M/S bit of ICW 4.
M/S: If M/S = 1, 8259A is a master. If M/S =0, 8259A is slave. If BUF = 0, M/S is to be neglected.
AEOI: If AEOI = 1, the automatic end of interrupt mode is selected.
µPM: If the µPM bit is 0, the Mcs-85 system operation is selected and if µPM=1, 8086/88 operation is selected.

Operation command words (OCWs)
Once 8259A is initialized it is ready for its normal function. 8259A has its own ways of handling the received interrupts called as modes of operation. These modes of operations can be selected by programming, i.e. writing three internal registers called as operation command word registers. The data written into them (bit pattern) is called as operation command words. In the three operation command words OCW1, OCW2, OCW3 every bit corresponds to some operational feature of the mode selected, except for a few bits those are either 1 or 0.

Operation Command Word 1 (OCW1)
OCW1 is used to mask the unwanted interrupt request and if it is 0 the request is enabled.

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

1: Mask Set 0: Mask Reset

Operation Command Word2 (OCW2)
In OCW2 the three bits, R, SL and EOI control the end of interrupt, the rotate mode and their combinations as shown in fig below. The three bits L_2, L_1 and L_0 in OCW2 determine the interrupt level to be selected for operation, if SL bit is active i.e. 1.

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	R	SL	EOI	0	0	L_2	L_1	L_0

	R	SL	EOI	
End of Interrupt	0	0	1	Non Specific EOI Command
	0	1	1	Specific EOI Command
Automatic Rotation	1	0	1	Rotate on Non Specific EOI Command
	1	0	0	Rotate in Automatic EOI mode(Set)
	0	0	0	Rotate in Automatic EOI mode(Clear)
Specific Rotation	1	1	1	Rotate on Specific EOI Command
	1	1	0	Set priority Command
	0	1	0	No operation

L_2 L_1 L_0 : 000 to 111 refers the Interrupt Request Numbers

Operation Command Word3 (OCW3)

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	ESMM	SMM	0	1	P	RR	RIS

P: 1 – Poll Command 0 – No Poll Command

ESMM	SMM	
0	0	No Action
0	1	
1	0	Reset Special Mask
1	1	Set Special Mask

RR	RIS	
0	0	No Action
0	1	
1	0	Read IRR on Next RD Pulse
1	1	Read ISR on Next RD Pulse

In operation command word 3 (OCW 3), if the ESMM bit, i.e. enable special mask mode bit is set to 1, the SMM bit is enabled to select or mask the special mask mode. When ESMM bit is 0 the SMM bit is neglected. If the SMM bit .i.e. special mask mode bit is 1, the 8259A will enter special mask mode provided ESMM=1. If ESMM=1 and SMM=0, the 8259A will return to the normal mask mode.

OPERATING MODES OF 8259

Fully Nested Mode: This is the default mode of operation of 8259A. IR_0 has the highest priority and IR_7 has the lowest one. When interrupt request are noticed, the highest priority request among them is determined and the vector is placed on the data bus. The corresponding bit of ISR is set and remains set till the microprocessor issues an EOI

command just before returning from the service routine or the AEOI bit is set. If the ISR (in service) bit is set, all the same or lower priority interrupts are inhibited but higher levels will generate an interrupt, that will be acknowledge only if the microprocessor interrupt enable flag IF is set. The priorities can afterwards be changed by programming the rotating priority modes.

End of Interrupt (EOI): The ISR bit can be reset either with AEOI bit of ICW1 or by EOI command, issued before returning from the interrupt service routine. There are two types of EOI commands **specific and non-specific**. When 8259A is operated in the modes that preserve fully nested structure, it can determine which ISR bit is to be reset on EOI. When non-specific EOI command is issued to 8259A it will be automatically reset the highest ISR bit out of those already set.

Automatic Rotation: This is used in the applications where all the interrupting devices are of equal priority. In this mode, an interrupt request IR level receives priority after it is served while the next device to be served gets the highest priority in sequence. Once all the devices are served like this, the first device again receives highest priority.

Automatic EOI Mode: Till AEOI=1 in ICW 4, the 8259A operates in AEOI mode. In this mode, the 8259A performs a non-specific EOI operation at the trailing edge of the last INTA pulse automatically. This mode should be used only when a nested multilevel interrupt structure is not required with a single 8259A.

Specific Rotation: In this mode a bottom priority level can be selected, using L_2, L_1 and L_0 in OCW 2 and R=1, SL=1, EOI=0. The selected bottom priority fixes other priorities. If IR 5 is selected as a bottom priority, then IR 5 will have least priority and IR4 will have a next higher priority. Thus IR 6 will have the highest priority. These priorities can be changed during an EOI command by programming the rotate on specific EOI command in OCW2.

Specific Mask Mode: In specific mask mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level and enables interrupt from other levels, which are not masked.

Edge and Level Triggered Mode: This mode decides whether the interrupt should be edge triggered or level triggered. If bit LTIM of ICW1 =0 they are edge triggered, otherwise the interrupts are level triggered.

READING 8259 STATUS

The status of the internal registers of 8259A can be read using this mode. The OCW 3 is used to read IRR and ISR while OCW1 is used to read IMR. Reading is possible only in no polled mode.

Poll Command : In polled mode of operation, the INT output of 8259A is neglected, though it functions normally, by not connecting INT output or by masking INT input of the

microprocessor. The poll mode is entered by setting P=1 in OCW3. The 8259A is polled by using software execution by microprocessor instead of the requests on INT input. The 8259A treats the next RD pulse to the 8259A as an interrupt acknowledge. An appropriate ISR bit is set, if there is a request. The priority level is read and a data word is placed on to data bus, after RD is activated. A poll command may give more than 64 priority levels.

Special Fully Nested Mode: This mode is used in more complicated system, where cascading is used and the priority has to be programmed in the master using ICW 4. This is somewhat similar to the normal nested mode. • In th is mode, when an interrupt request from a certain slave is in service, this slave can further send request to the master, if the requesting device connected to the slave has higher priority than the one being currently served. In this mode, the master interrupts the CPU only when the interrupting device has a higher or the same priority than the one current being served. In normal mode, other requests than the one being served are masked out. When entering the interrupt service routine the software has to check whether this is the only request from the slave. This is done by sending a non-specific EOI can be sent to the master, otherwise no EOI should be sent. This mode is important, since in the absence of this mode, the slave would interrupt the master only once and hence the priorities of the slave inputs would have been disturbed.

Buffered Mode: When the 83259A is used in the systems where bus driving buffers are used on data buses. The problem of enabling the buffers exists. The 8259A sends buffer enable signal on SP/ EN pin, whenever data is placed on the bus.

Cascade Mode: The 8259A can be connected in a system containing one master and eight slaves (maximum) to handle upto 64 priority levels. The master controls the slaves using CAS 0-CAS 2 which act as chip select inputs (encoded) for slaves. In this mode, the slave INT outputs are connected with master IR inputs. When a slave request line is activated and acknowledged, the master will enable the slave to release the vector address during second pulse of INTA sequence.

7a.DMA CONTROLLER 8257

Write briefly about the Direct Memory Access. (4)**[April/May 2011, April/May2017]**

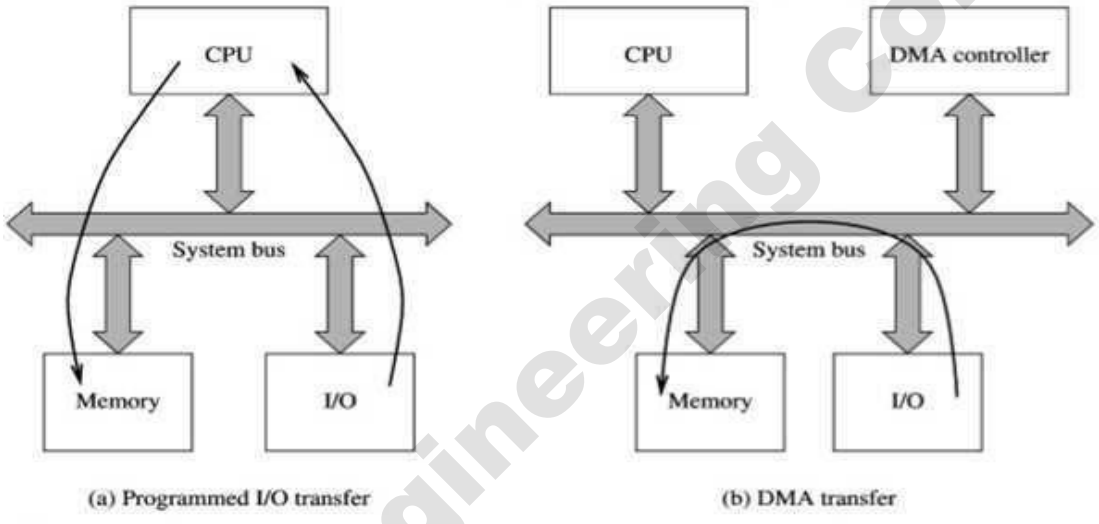
How to interface a DMA controller with a microprocessor? Explain how DMA controller transfers large amount of data from one memory locations to another memorylocation? (8) [**Nov/Dec 2014**].

Explain the internal architecture of 8257 Direct Memory Access Controller. (16) [**May/Jun 2014**]

What is DMA? Explain the DMA based data transfer using DMA controller. (6) [**Apr/May 2015**]

The Direct Memory Access or DMA mode of data transfer is the fastest among all the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU. The device requests the CPU (through a DMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

Intel's 8257 is a four channel DMA controller designed to be interfaced with their family of microprocessors. The 8257, on behalf of the devices, requests the CPU for bus access using local bus request input i.e. HOLD in minimum mode. In maximum mode of the microprocessor RQ/GT pin is used as bus request input. On receiving the HLDA signal (in minimum mode) or RQ/GT signal (in maximum mode) from the CPU, the requesting devices gets the access of the bus, and it completes the required number of DMA cycles for the data transfer and then hands over the control of the bus back to the CPU.



INTERNAL ARCHITECTURE OF 8257

The chip support four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers. The chip support four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers.

Register Organization of 8257

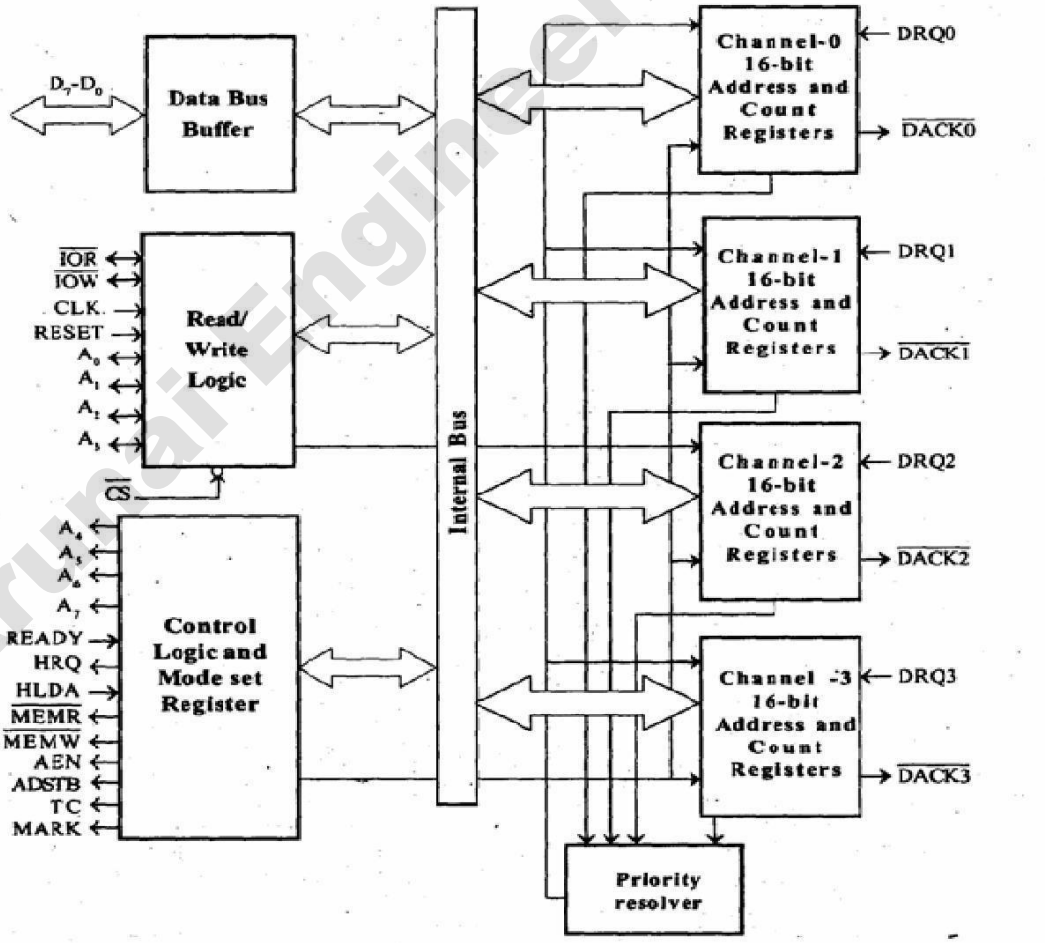
The 8257 performs the DMA operation over four independent DMA channels. Each of four channels of 8257 has a pair of two 16-bit registers, viz. DMA address register and terminal count register. There are two common registers for all the channels; namely, mode set register and status register. Thus there are a total of ten registers. The CPU selects one of these ten registers using address lines Ao-A3.

DMA Address Register

Each DMA channel has one DMA address register. The starting address of the memory block which will be accessed by the device is first loaded in the DMA address register. The device that wants to transfer data over a DMA channel, will access the block of the memory with the starting address stored in the DMA Address Register.

Terminal Count Register

Each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register is used for ascertaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. The low order 14-bits of the terminal count register are initialized with the binary equivalent of the number of required DMA cycles minus one. After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over. The bits 14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit14 of the register in this case will be set to one and bit 15 will be set to zero. Table gives detail of DMA operation selection and corresponding bit configuration of bits 14 and 15 of the TC register.



Mode Set Register

The mode set register is used for programming the 8257. The function of the mode set register is to enable the DMA channels individually and also to set the various modes of operation. The DMA channel should not be enabled till the DMA address register and the terminal count register contain valid information, otherwise, an unwanted DMA request may initiate a DMA cycle, probably destroying the valid memory data. The bits B_0- B_3 enable one of the four DMA channels of 8257. For example, if B_0 is '1', channel 0 is enabled.

Bit 15	Bit 14	Types of DMA Operation
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	Illegal

If B_4 is set, rotating priority is enabled, otherwise, the normal, i.e. fixed priority is enabled If the TC STOP bit is set, the selected channel is disabled after the terminal count condition is reached, and it further prevents any DMA cycle on the channel. To enable the channel again, this bit must be reprogrammed. If the TC STOP bit is programmed to be zero, the channel is not disabled, even after the count reaches zero and further request are allowed on the same channel. The auto load bit, if set, enables channel 2 for the repeat block chaining operations, without immediate software intervention between the two successive blocks. The channel 2 registers are used as usual, while the channel 3 registers are used to store the block reinitialisation parameters, i.e. the DMA starting address and terminal count. After the first block is transferred using DMA, the channel 2 registers are reloaded with the corresponding channel 3 registers for the next block transfer, if the update flag is set. The extended write bit, if set to '1', extends the duration of MEMW and IOW signals by activating them earlier, this is useful in interfacing the peripherals with

AL	TCS	EW	RP	EN3	EN2	EN1	EN0

AL: 1 = Enable Auto Reload
0 = Disable Auto Reload
TCS : 1 = Stop DMA on terminal Count
EW : 1 = Extended Write selection
0 = Normal write selection
RP: 1 = Rotating Priority
0 = Fixed Priority

EN0: Channel 0
EN1: Channel 1
EN2 : Channel 2
EN3 : Channel 0
1 = Enable
0 = Disable

different access times. If the peripheral is not accessed within the stipulated time, it is expected to give the 'NOT READY' indication to 8257, to request it to add one or more wait states in the DMA CYCLE. The mode set register can only be written into.

Status Register

The status register of 8257 is shown in figure. The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers of channel 2 whenever the channel 2 reaches a terminal count condition, after transferring one block and the next block is to be transferred using the auto load feature of 8257.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

D_0	TC Status Channel 0
D_1	TC Status Channel 1
D_4	Update Flag

D_2	TC Status Channel 2
D_3	TC Status Channel 3

Data Bus Buffer, Read/Write Logic, Control Unit and Priority Resolver

The 8-bit. Tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals. In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the Ao-A3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether IOW or IOR signal is activated. In master mode, the read/write logic generates the IOR and IOW signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, MEMR, MEMW, TC and MARK along with the address lines A4-A7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

MODES OF OPERATION

Single mode

In single mode only one byte is transferred per request. For every transfer, the counting register is decremented and address is incremented or decremented depending on programming.

Block transfer mode

The transfer is activated by DREQ which can be deactivated once acknowledged by DACK. The transfer continues until end of process EOP (either internal or external) is activated which will trigger terminal count TC to the card. Auto-initialization may be programmed in this mode.

Demand transfer mode

The transfer is activated by **DREQ** and acknowledged by **DACK** and continues until either **TC**, external **EOP** or **DREQ** goes inactive. Only **TC** or external **EOP** may activate auto-initialization if this is programmed.

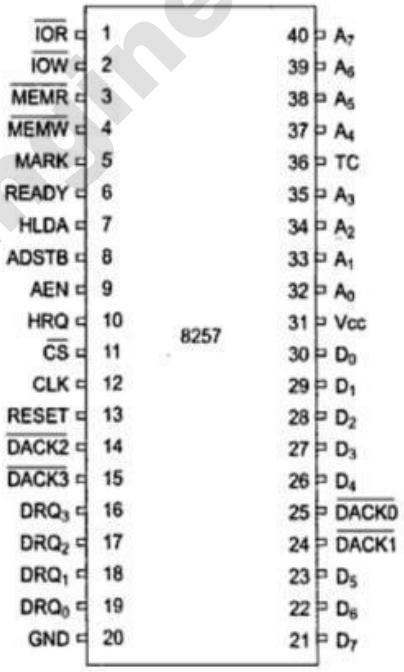
SIGNAL DESCRIPTION OF 8257

DRQo-DRQ3: These are the four individual channel DMA request inputs, used by the peripheral devices for requesting the DMA services. The DRQ_O has the highest priority while DRQ3 has the lowest one, if the fixed priority mode is selected.

DACKo-DACK3: These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. These lines may act as strobe lines for the requesting devices.

Do-D7: These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257. These lines carry command words to 8257 and status word from 8257, in slave mode, i.e. under the control of CPU. The data over these lines may be transferred in both the directions. When the 8257 is the bus master (master mode, i.e. not under CPU control), it uses Do-D7 lines to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal. the address is transferred over Do-D7 during the first clock cycle of the DMA cycle. During the rest of the period, data is available on the data bus.

IOR: This is an active-low bidirectional tristate input line that acts as an input in the slave mode. In slave mode, this input signal is used by the CPU to read internal registers of 8257.this line acts output in master mode. In master mode, this signal is used to read data from a peripheral during a memory write cycle.



IOW: This is an active low bidirection tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bitDMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK: This is a clock frequency input required to derive basic system timings for the internal operation of 8257.

RESET: This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.

Ao-A3: These are the four least significant address lines. In slave mode, they act as input which selects one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS: This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to prevent the chip from getting selected (by CPU) while performing the DMA operation.

A4-A7: This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY: This is an active-high asynchronous input used to stretch memory read and write cycles of 8257 by inserting wait states. This is used while interfacing slower peripherals..

HRQ: The hold request output requests the access of the system bus. In the noncascaded 8257 systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.

HLDA: The CPU drives this input to the DMA controller high, while granting the bus to the device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.

MEMR: This active –low memory read output is used to re ad data from the addressed memory locations during DMA read cycles.

MEMW: This active-low three state output is used to write data This active-low three state output is used to write data to the addressed memory location during DMA write operation. **ADST:** This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.

AEN: This output is used to disable the system data bus and the control the bus driven by the CPU; this may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. If the 8257 is I/O mapped, this should be used to disable the other I/O devices, when the DMA controller addresses is on the address bus.

TC: Terminal count output indicates to the currently selected peripherals that the present DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected

channel becomes equal to zero. The lower order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK: The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. The mark will be activated after each 128 cycles or integral multiples of it from the beginning if the data block (the first DMA cycle), if the total number of the required DMA cycles (n) is completely divisible by 128.

Vcc: This is a +5v supply pin required for operation of the circuit. **GND**: This is a return line for the supply (ground pin of the IC).

7b.TRAFFIC LIGHT CONTROLLER

Write the algorithm and assembly language program for traffic light control system.(8) [**Apr/May 2014**]

Draw the block diagram of traffic light control system using 8086. (8) [**Apr/May 2015**]

Vehicular traffic at intersecting streets is typically controlled by traffic control lights. The function of **traffic lights** requires sophisticated **control and coordination** to ensure that traffic moves as smoothly and safely as possible. Microprocessor is programmed in such a way to adjust their timing and phasing to meet changing traffic conditions. Traffic congestion is a severe problem in many modern cities around the world. Traffic congestion has been causing many critical problems and challenges in the major and most populated cities. To travel to different places within the city is becoming more difficult for the travellers in traffic. Due to these congestion problems, people lose time, miss opportunities, and get frustrated. Traffic congestion directly impacts the companies. Due to traffic congestions there is a loss in productivity from workers, trade opportunities are lost, delivery gets delayed, and thereby the costs goes on increasing.

Traffic lights, which may also be known as stoplights, traffic lamps, traffic signals, signal lights, robots or semaphore, are signalling devices positioned at road intersections, pedestrian crossings and other locations to control competing flows of traffic.

ABOUT THE COLORS OF TRAFFIC LIGHT CONTROL

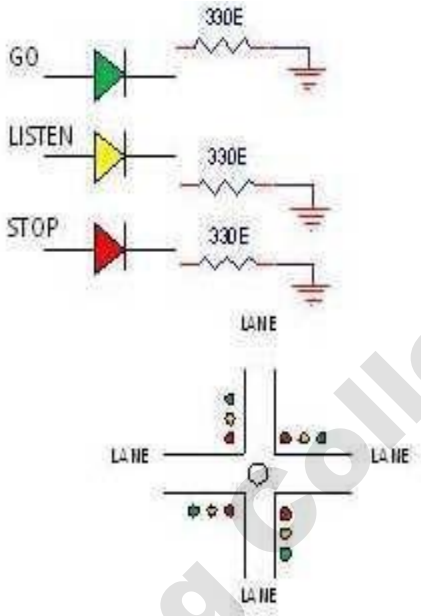
Traffic lights alternate the right of way of road users by displaying lights of a standard color red, yellow/amber, and green. Illumination of the red signal prohibits any traffic from proceeding. Usually, the red light contains some orange in its hue, and the green light contains some blue, for the benefit of people with red-green color blindness, and "green" lights in many areas are in fact blue lenses on a yellow light (which together appear green).

INTERFACING TRAFFIC LIGHT WITH 8086

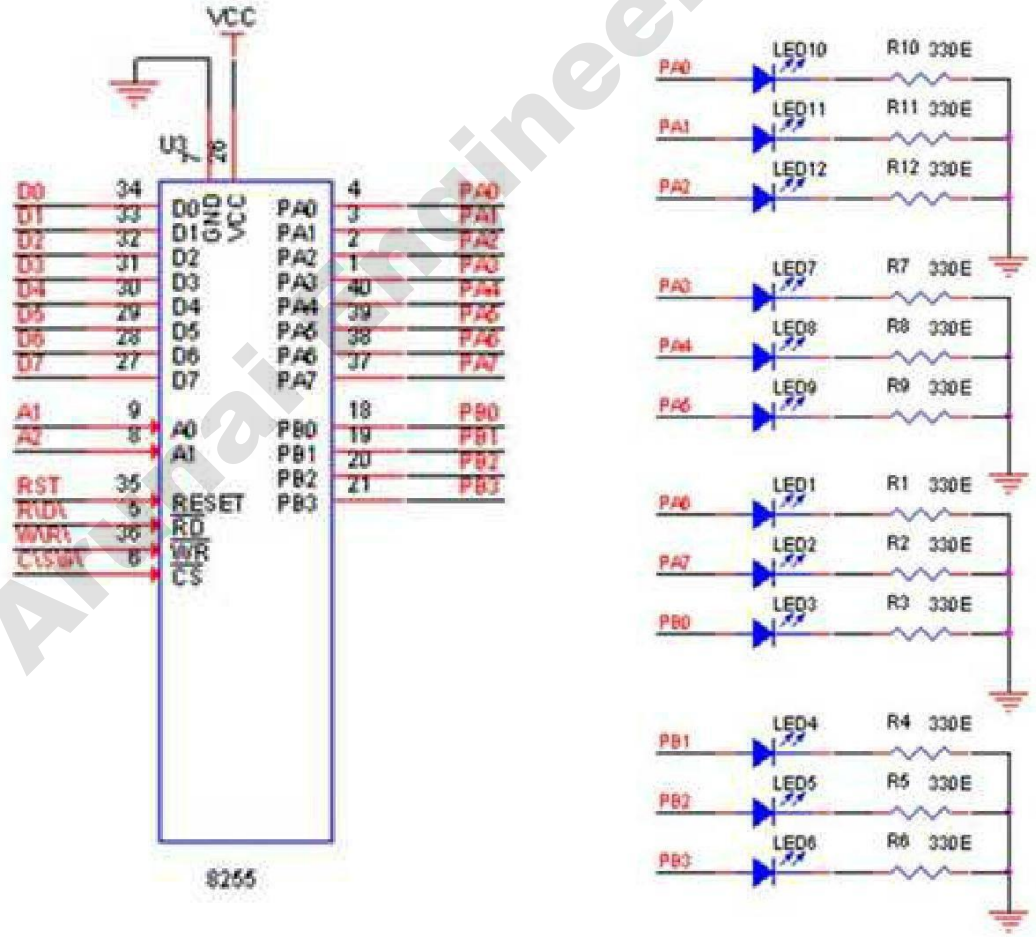
The Traffic light controller section consists of 12 Nos. of LED's arranged by 4Lanes in Traffic light interface card. Each lane has Go (Green), Listen (Yellow) and Stop (Red) LED is being placed.

PIN ASSIGNMENT WITH 8086

LAN Direction	8086 LINES	MODULES
SOUTH	PA.0	GO
	PA.1	LISTEN
	PA.2	STOP
EAST	PA.3	GO
	PA.4	LISTEN
	PA.5	STOP
NORTH	PA.6	GO
	PA.7	LISTEN
	PB.0	STOP
WEST	PB.1	GO
	PB.2	LISTEN
	PB.3	STOP
PWR	13-16	NC
	17,19	Vcc
	18,20	Gnd



Circuit Diagram To Interface Traffic Light With 8086



Assembly Program To Interface Traffic Light With 8086

```
START: MOV BX, 1200H
       MOV CX, 0008H
       MOV AL,[BX]
       MOV DX, CONTROL PORT
       OUT DX, AL
       INC BX
NEXT:  MOV AL,[BX]
       MOV DX, PORT A
       OUT DX,AL
       INC BX
       MOV AL,[BX]
       MOV DX,PORT B
       OUT DX,AL
       CALL DELAY
       INC BX
       LOOP NEXT
       JMP START

DELAY: PUSH CX
       MOV CX,0005H
REPEAT: MOV DX,0FFFFH
LOOP2: DEC DX
       JNZ LOOP2
       LOOP REPEAT
       POP CX
       RET
```

LOOKUP TABLE

1200	80H
1201	21H,09H,10H,00H (SOUTH WAY)
1205	0CH,09H,80H,00H (EAST WAY)
1209	64H,08H,00H,04H (NORTH WAY)
120D	24H,03H,02H,00H (WEST WAY)

UNIT IV MICROCONTROLLER

Architecture of 8051– Special Function Registers (S FRs) - I/O Pins Ports and Circuits - Instruction set-Addressing modes - Assembly language programming.

PART-A (2 MARKS)

1. What is mean by microcontroller? [Apr/May 2011]

A device which contains the microprocessor with integrated peripherals like memory, serial ports, parallel ports, timer/counter, interrupt controller, data acquisition interfaces like ADC, DAC is called microcontroller.

2. List the features of 8051 microcontroller? [May/June 2007] [Nov/Dec 2007, 2011]

The features are

- Single supply +5 volt operation using HMOS technology.
- 4096 bytes program memory on chip(not on 8031)
- 128 data memory on chip.
- Four register banks.
- Two multiple mode, 16-bit timer/counter.
- Extensive Boolean processing capabilities.
- 64 KB external RAM size
- 32 bi-directional individually addressable I/O lines.
- 8 bit CPU optimized for control applications.

3. What is Microcontroller and Microcomputer? [April/May 2011]

Microcontroller is a device that includes microprocessor; memory and I/O signal lines on a single chip, fabricated using VLSI technology. Microcomputer is a computer that is designed using microprocessor as its CPU. It includes microprocessor, memory and I/O.

4. Give the alternate functions for the port pins of port3? [Apr/May 2011, April/May2017]

P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
RD	WR	T1	T0	INT1	INT0	TxD	RxD

RD – Read data control output.
WR – Write data control output.
T1 – Timer / Counter1 external input or test pin.
T0 – Timer / Counter0 external input or test pin.
INT1- Interrupt 1 input pin.
INT 0 – Interrupt 0 input pin.
TXD – Transmit data pin for serial port in UART mode.
RXD - Receive data pin for serial port in UART mode.

5. What are the addressing modes supported by 8051? [April/May 2008, Nov/Dec 2011]

- Register addressing
- Direct byte addressing
- Register indirect
- Immediate
- Register specific
- Index

6. Explain the function of the SP pin of 8051. [Nov/Dec 2011]

SP: SP stands for stack pointer. SP is a 8- bit wide register. It is incremented before data is stored during PUSH and CALL instructions. The stack array can reside anywhere in on-chip RAM. The stack pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

7. State the function of RS1 and RS0 bits in the flag register of Intel 8051 microcontroller? [Nov/Dec 2011] [April/May 2010]

RS1 and RS0: Bank Selection

RS1	RS0	Bank Selection
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	0	Bank 3

8. Name the special functions registers available in 8051. [May/June 2007]

80	P0
81	SP
82	DPL
83	DPH
87	PCON
88	TCON
89	TMOD
8A	TL0
8B	TL1
8C	TH0
8D	TH1

90	P1
98	SCON
99	SBUF
A0	P2
A8	IE
B0	P3
B8	IP
D8	PSW
E0	ACC
F0	B

9. What are the differences between microprocessor and microcontroller? [May/Jun 2014]Compare Microprocessor and Microcontroller. [Nov/Dec 2006, 2011]

Microprocessor	Microcontroller
Microprocessor contains ALU, general purpose registers, stack pointer, program counter, clock timing circuit and interrupt circuit.	Microcontroller contains the circuitry of microprocessor and in addition it has built in ROM, RAM, I/O devices, timers and counters.
It has many instructions to move data between memory and CPU. It has one or two instructions to move data between memory and CPU.	It has one or two bit handling instructions. It has many bit handling instructions.
Access times for memory and I/O devices are more.	Less access time for built-in memory and I/O devices.
Microprocessor based system requires more hardware.	Microcontroller Based system requires less hardware reducing PCB size and reducing the reliability

10. Why a latch is used for an O/P port, but a tri-state buffer can be used for an input port?[May/June 2012]

Output port is to source large currents the port lines must be buffered. Hence the latch acts as a good output port. So, 74LS373 contains eight buffered latches and can be used as an 8 bit output port. An input device one must take care that much current should not be sourced or sink from the data lines to avoid loading. So, tristate buffer is used as input device.

11. What are the special function register? (EE2354 April/May2012)

The special function register are stack pointer, index pointer (DPL and DPH), I/O port addresses, status (PSW) and accumulator.

12. What are the uses of accumulator register?

The accumulator registers (A and B at addresses OEOh and OFOh, respectively) are used to store temporary values and the results of arithmetic operations.

13. What is PSW? (EE2354 Nov/Dec2011)

Program status word (PSW) is the set of flags that contains the status information and is considered as one of the special function register.

14. What is stack pointer (sp)? (EE2354 April/May2011)

Stack pointer (SP) is a 8 bit wide register and is incremented before the data is stored into the stack using PUSH or CALL instructions. It contains 8-bit stack top address. It is defined anywhere in the on-chip 128-byte RAM. After reset, the SP register is initialized to 07. After each write to stack operation, the 8-bit contents of the operand are stored onto the stack, after

incrementing the SP register by one. It is not a top-down data structure. It is allotted an address in the special function register bank.

15. What is data pointer (DTPR)? (Nov/Dec2010)

It is a 16-bit register that contains a higher byte (DPH) and lower byte (DPL) of a 16-bit external data RAM address. It is accessed as a 16-bit register or two 8-bit registers. It has been allotted two addresses in the special function register bank, for its two bytes DPH and DPL.

16. Why oscillator circuit is used?

Oscillator circuit is used to generate the basic timing clock signal for the operation of the circuit using crystal oscillator.

17. What is the purpose of using instruction register?

Instruction register is used for the purpose of decoding the opcode of an instruction to be executed and gives information to the timing and control unit generating necessary signals for the execution of the instruction.

18. Give the purpose of ALE/PROG signal. (May/June2014)

ALE/PROG is an address latch enable output pulse and indicates that valid address bits available on the respective pins. The ALE pulses are emitted at a rate of one-sixth of the oscillator frequency. The signal is valid only for external memory accesses. It may be used for external timing or clockwise purpose. One ALE pulse is skipped during each access to external data memory.

19. Explain the two power saving mode of operation. (April/May2011)

The two power saving modes of operation are:

- **Idle mode**: In this mode, the oscillator continues to run and the interrupt, serial port and timer blocks are active, but the clock to the CPU is disabled. The CPU status is preserved. This mode can be terminated with a hardware interrupt or hardware reset signal. After this, the CPU resumes program execution from where it left off.

- **Power down mode:** In this mode, the on-chip oscillator is stopped. All the functions of the controller are held maintaining the contents of RAM. The only way to terminate this mode is hardware reset. The reset redefines all the SFRs but the RAM contents are left unchanged.

20. Differentiate between program memory and data memory.

Program Memory

i. It stores the programs to be executed.

ii. It stores only program code which is to be executed and thus it need not be written, so it is implemented using EPROM It stores the data, line intermediate results, variables and constants required for the execution of the program.

Data Memory: The data memory may be read from or written to and thus it is implemented using RAM.

21. What are addressing modes?

The various ways of accessing data are called addressing modes.

22. Give the addressing modes of 8051? (April/May 2011)

There are six addressing modes in 8051.They are

- Direct addressing
- Indirect addressing

- Register instruction
- Register specific (register implicit)
- Immediate mode
- Indexed addressing

23. What is direct addressing mode?

The operands are specified using the 8-bit address field, in the instruction format. Only internal data Ram and SFRS can be directly addressed. This is known as direct addressing mode
Eg: Mov R0, 89H

24. What is indirect addressing mode?

In this mode, the 8-bit address of an operand is stored in a register and the register, instead of the 8-bitaddress, is specified in the instruction. The registers R0 and R1 of the selected bank of registers or stack pointer can be used as address registers for storing the 8-bit addresses. The address register for 16-bit addresses can only be „data pointer" (DPTR).
Eg: ADD A, @ R0.

25. What is meant by register instructions addressing mode?

The operations are stored in the registers R0 – R7 of the selected register bank. One of these eight registers(R0 – R7) is specified in the instruction using the 3-bit register specification field of the opcode format. A register bank can be selected using the two bank select bits of the PSN. This is called as register instruction addressing mode
Eg: ADD A, R7.

26. What is immediate addressing mode? (April/May2013)

An immediate data ie., a constant is specified in the instruction, after the opcode byte.
Eg: MOV A, #100 The immediate data 100 (decimal) is added to the contents of the accumulator. For specifying a hex number,it should be followed by H. These are known as immediate addressing mode.

27. What is indexed addressing? (May/June2014)

This addressing mode is used only to access the program memory. It is accomplished in 8051 for look-up table manipulations. Program counter or data pointer are the allowed 16-bit address storage registers, in this mode of addressing. These 16-bit registers point to the base of the look-up table and the ACC register contains a code to be converted using the look-up table. The look-up table data address is found out by adding the contents of register ACC with that of the program counter or data pointer. In case of jump instruction, the contents of accumulator are added with one of the specified 16-bit registers to form the jump destination address.
Eg: MOV C, A @ A + DPTP JMP @ A + DPTR

28. List the five addressing modes of 8051 microcontroller. (Nov/Dec2010)

The five addressing modes are,
I. Immediate addressing
II. Register addressing
III. Direct addressing
IV. Register indirect addressing
V. Indexed addressing.

29. MOV R4, R7 is invalid. Why?

The movement of data between the accumulator and Rn (for n = 0 to 7) is valid. But movement of data between Rn register is not allowed. That is why MOV R4, R7 is invalid.

30. WHAT IS SFR? (Nov/Dec2014)

In the 8051 microcontroller registers A, B, PSW and DPTR are part of the group of registers commonly referred to as special function registers (SFR).

31. WHAT ARE THE TWO MAIN FEATURES OF SFR ADDRESSES?

The following two points should be noted SFR addresses.

- The special function registers have addresses between 80H and FFH. These addresses are above 80H, since the addresses 00 to 7FH are addresses of RAM memory inside the 8051.

- Not all the address space of 80 to FH is used by the SFR. The unused locations 80Hto FFH are reserved and must not used by the 8051 programmer.

32. What is the difference between direct and register indirect addressing mode?

Loop is most efficient and is possible only in register indirect addressing whereas looping is not direct addressing mode.

33. List out some compare instructions. (EE2354May/June2014)

The compare instructions are:

a. CJNE

b. CLR

c. CPL

34. Write a program to save the accumulator in r7 of bank 2.

```
CLR PSW – 3
SETB PSW – 4
MOV R7, A.
```

35. What are single bit instructions? Give example.

Instructions that are used for single bit operation are called single bit instructions.

Examples: SETB bit
CLR bit
CPL bit

36. Write a program to save the status of bits p1.2 and p1.3 on ram bit locations 6 and 7 respectively.

```
MOV C, P1.2; save status of P1.2 on CY
MOV O6, C; save carry in RAM bit location 06 MOV C, p1.3; save status of p1.3 on CY
MOV 07, C; save carry in RAM bit location 07.
```

37. Write a program to see if bits 0 and 5 of register b r1. If they are not, make them so and save it in r0. (Nov/Dec2011)

```
JNB OFOH, NEXT – 1; JUMP if B.0 is low
SET BOFOH; Make bit B.0 high
NEXT – 1:JNB OF5H, NEXT – 2; JUMP if B.5 is low
SETB OF5H; Make B.5 high
NEXT – 2: MOV R0, B; Save register B.
```

38. Mention the size of DPTR and Stack Pointer in 8051 microcontroller.(April/May 2011), (May/June2014)

DPTR and SP are 16 bit register.

39. What is the operation of the given 8051 microcontroller instructions: XRL A, direct (April/May2011)

XRLA, Direct Exclusive OR operation with A register content and Direct value

40. List the features of 8051 microcontroller? (May/June2013)

The features are

- Single supply +5 volt operation using HMOS technology.
- 4096 bytes program memory on chip(not on 8031)
- 128 data memory on chip.
- Four register banks.
- Two multiple mode, 16-bit timer/counter.
- Extensive Boolean processing capabilities.
- 64 KB external RAM size
- 32 bidirectional individually addressable I/O lines.
- 8 bit CPU optimized for control applications.

41. Name the five interrupt sources of 8051? (MAY/JUNE2007) (APRIL/MAY2008)

The interrupts are:

Vector address

• External interrupt 0: IE0: 0003H
• Timer interrupt 0: TF0: 000BH
• External interrupt 1: IE1: 0013H
• Timer Interrupt 1: TF1: 001BH
• Serial Interrupt

Receive interrupt: RI: 0023H

Transmit interrupt: TI: 0023H

42. List the 8051 instructions that affect the overflow flag.

ADD, ADDC, DIV, MUL, SUBB

43. List the 8051 instructions that always clear the carry flag.

CLR C, DIV, MUL

44. List the 8051 instructions that affect all the flags. (NOV/DEC 2007)

ADD, ADDC and SUBB

45. What are the different types of ADC? (APR/MAY2008 NOV/DEC 2011)

The different types of ADC are successive approximation ADC, counter type ADC flash type ADC, integrator converters and voltage to- frequency converters.

46. What is the necessity of interfacing DAC with microcontroller? (Nov/Dec 2014)

In many applications, the microcontroller has to produce analog signals for controlling certain analog devices. Basically, the microcontroller can produce only digital signals. In order to convert the digital signal to analog signal a Digital to Analog Converter has to be employed.

47. Mention the number of register banks and their addresses in 8051? (Nov/Dec2015)

There are 4 register banks. They are Bank0, Bank1, Bank2 & Bank3.

RAM locations from 00 to 07H for bank 0
RAM locations from 08 to 0FH for bank 1
RAM locations from 10 to 17H for bank 2
RAM locations from 18 to 1FH for bank 3

48. What is the jump range? (Nov/Dec2015)

AJMP addr11 (Absolute Jump) – Within 2K bytes of program memory.
LJMP addr16 (Long Jump) -Within 64K bytes of program memory.
SJMP Rel.addr (Short Jump) –128 to +127 of program memory.

49. What are the different ways of operand addressing in 8051? (May/June 2016)

The five addressing modes are,

1. Immediate addressing
2. Register addressing
3. Direct addressing
4. Register indirect addressing
5. Indexed addressing

PART-B (13 MARKS)

1. ARCITECTURE OF 8051

Explain the memory structure of an 8051 Microcontroller. (8 Marks) **[April/May 2010]**

Discuss briefly the various registers in 8051 microcontroller. (6) [**Nov/Dec 2011]**

Explain the architecture of 8051 microcontroller with neat diagram. [Marks 12] [**April/May 2011]**

Explain the architecture of 8051 microcontroller with neat diagram. (10) [**Nov /Dec 2013]**

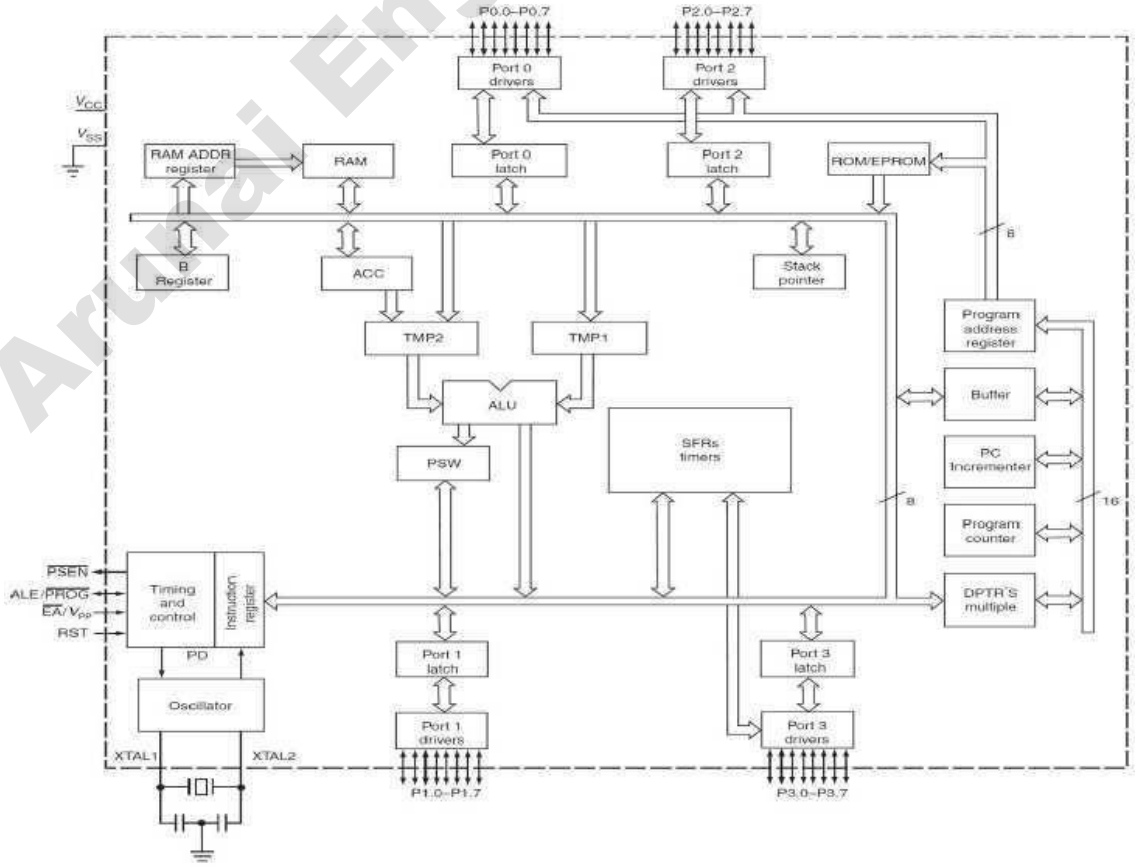
Draw the functional block diagram of 8051 microcontroller and explain each block. (8) [**Nov/Dec 2014**].

Draw the data memory structure of 8051 microcontroller and explain. (8) [**Nov/Dec 2014**].

Draw the internal architecture of 8051 Microcontroller. (16) [**Apr/May 2014**]

Explain the architecture of 8051 microcontroller with neat diagram. (8) [**Apr/May2015**]

8051 is 8-bit microcontroller; it can Read, Write and Process 8 bit data. This is mostly used microcontroller in the robotics, home appliances likemp3 player, washing machines, electronic iron and industries.



ALU

It is 8 bit unit. It performs arithmetic operation as addition, subtraction, multiplication, division, increment and decrement. It performs logical operations like AND, OR and EX-OR. It manipulates 8 bit and 16 bit data.

Accumulator

It is 8 bit register. Its address is E0H and it is bit and byte accessible. Result of arithmetic & logic operations performed by ALU is accumulated by this register.

B-register

It is used to store one of the operands for multiply and divide instructions. It is special 8 bit maths register. It is bit and byte accessible. It is used in conjunction with A register as I/P operand for ALU. It is used as general purpose register to store 8 bit data.

PSW

It is 8 bit register. Its address is D0H and it is bit and byte accessible. It has 4 conditional flags and 3 control flags

CY	AC	F0	RS1	RS0	OV	-	P

Carry Flag (CY): During addition and subtraction any carry or borrow is generated then carry flag is set otherwise carry flag resets. It is used in arithmetic, logical, jump, rotate and Boolean operations.

Auxiliary Carry Flag (AC): During addition and subtraction any carry or borrow is generated from lower 4 bit to higher 4 bit then AC sets else it resets. It is used in BCD arithmetic operations.

F0: User defined flag bit for general purpose.

Overflow Flag (OV): If signed arithmetic operations result exceeds more than 7 bit than OV flag sets else resets. It is used in signed arithmetic operations only.

Parity Flag (P): If in the result even no. of ones'1' are present than it is called even parity and parity flag sets. In the result odd no. of ones'1'are present than it is called odd parity and parity flag resets

RS1 and RS0: Register Bank Selection

RS1	RS0	Bank Selection
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	0	Bank 3

Program Counter (PC):

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory. When the 8051 is initialized PC always starts at 0000 H and is incremented each time an instruction is executed.

Data Pointer Register (DTPR):
It is a 16 bit register used to hold address of external or internal RAM where data is stored or result is to be stored. It is used to store 16 bit data. It is divided into two 8bit registers, DPH-data pointer higher order and DPL-data pointer lower order.

Stack Pointer (SP):
It is 8bit register. It is byte addressable. When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by 1.

P0, P1, P2, P3 (Port): This is input/output port0, port1, port2, port3. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit0 of port0 is pin P0.0, bit 7 is in P0.7.Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin where as a value of 0 will bring it to a low level.

Serial Data Buffer: The serial data buffer internally contains two independent registers. One of them is a transmit buffer which is necessarily a parallel in serial out register. The other is called receive buffer which is a serial in parallel out register. Loading a byte to the transmit buffer initiates serial transmission of that byte. The serial buffer is identified as SBUF. If a byte is written into SBUF it initiates a serial transmission and if the SBUF is read, it reads received serial data.

Timer Registers: These two 16 bit registers can be accessed as their lower and upper bytes. It contains two timers. TL0 represents the lower byte of the timing register, TH0 represents the higher bytes of the timer register 0. Similarly TL1 and TH1 represent lower and higher bytes of timing register 1.

Control Registers: The special function registers IP, OE, TMOD, TCON, SCON, and PCON contain control and status information for interrupts, timers/counters and serial port.

Timing and Control Unit: This unit derives all the necessary timing and control signals required for the internal operation of the circuit. It also derives the basic timing control signals required for controlling the external system bus.

Oscillator: This circuit generates the basic timing clock signal for the operation of the circuit using crystal oscillator.

Instruction Register: This register decodes the opcode of an instruction to be executed and gives information to the timing and control unit to generate necessary signals for the execution of the instruction.

EPROM and Program Address Register: These blocks provide on chip EPROM and a mechanism to internally address it.

RAM and RAM Address Register: This block provides internal 128 bytes of Ram and a mechanism to address it internally.

Power Control Register: PCON

It is 8-bit register. It is byte addressable. Its bits are used to control mode of power saving circuit, either idle or power down mode and also one bit is used to modify the baud rate of serial communication.

SMOD	-	-	-	GF1	GF0	PD	IDL

SMOD: Serial baud rate modify bit
GF1: General purpose user flag bit 1
GF0: General purpose user flag bit 0
PD: Power down Mode
IDL: Idle Mode

Idle Mode

A hardware reset exits the idle mode. The CPU starts from the instruction following the instruction that invoked the 'Idle' mode.

Power down Mode

The internal clock to the entire microcontroller is stopped (frozen). However, the program is not dead. The Power down Mode is exited (PCON.1 is cleared to 0) by Hardware Reset only. The CPU starts from the next instruction where the Power down Mode was invoked.

8051 Clock and Instruction Cycle:

The heart of 8051 is the circuitry that generates the clock pulses by which all internal operations are synchronized. Pins XTAL1 and XTAL2 are provided for connecting resonator to form an oscillator. The crystal frequency is the basic internal frequency of the microcontroller. 8051 is designed to operate between 1MHz to 16MHz and generally operates with a crystal frequency 11.04962 MHz

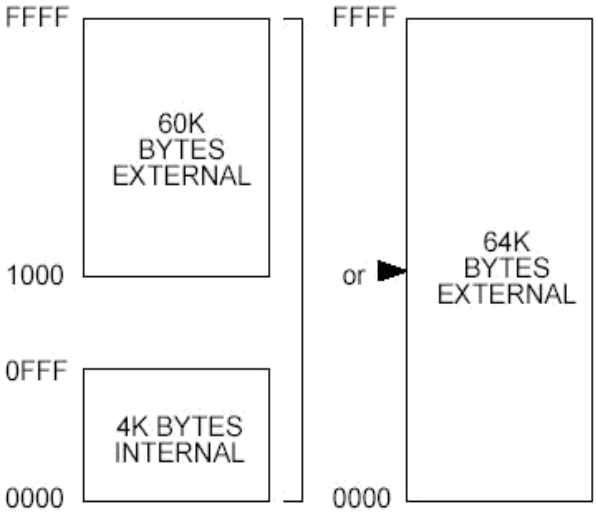
MEMORY ORGANIZATION OF 8051

The 8051 has a separate memory space for code and data. It is called as Program memory and Data memory [**April/May2017]**

Program Memory

The executable program code is stored in this code memory. The code memory size is limited to 64Kbytes. The code memory is read only in normal operation and is programmed under special conditions. e.g. it is a PROM or a Flash RAM type of memory. When EA = 0, 64 K bytes is divided as 4K bytes of Internal Memory and 60 K bytes of external Memory. When EA = 1, 64 K bytes considered as external Memory. 8051 memory is organized so that data memory and program code memory can be two entirely different physical memory entities. Each has the same address ranges.

The internal program ROM occupies code address space 0000H to 0FFFH. The PC is normally used to address program code bytes from address 0000H to FFFFH. Program addresses higher than OFFFH which exceed the internal ROM capacity will cause the 8051 to automatically fetch code bytes from external memory, addresses 1000H to FFFFH by connecting the external access pin (EA) to ground

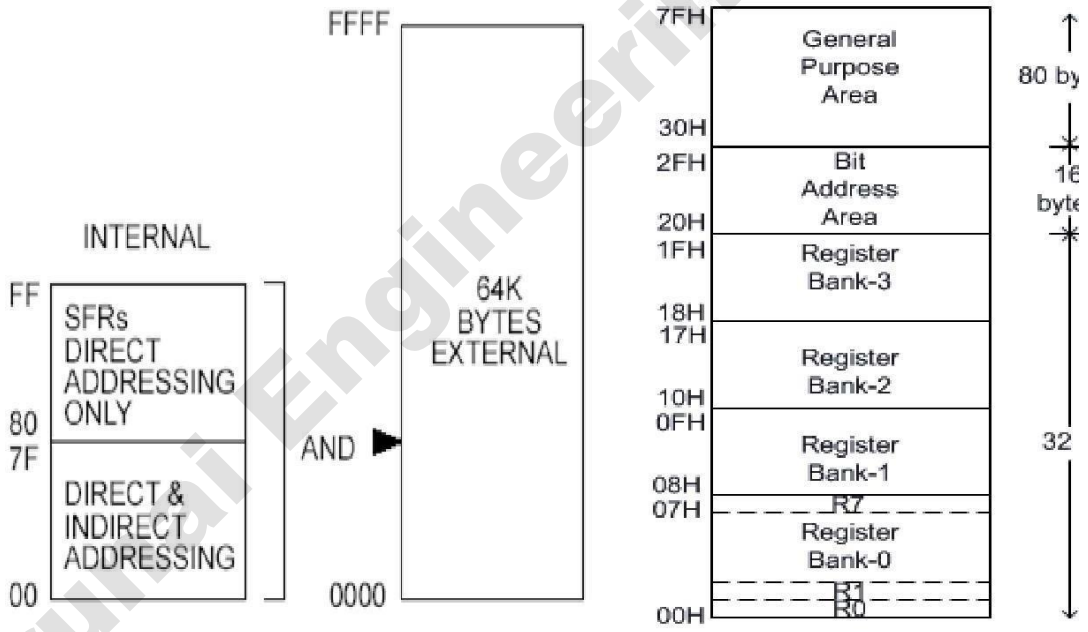


EA = 0 EA = 1

.

Data Memory

This is read write memory and is available for storage of data. Up to 64KBytes of external RAM data memory is supported in a standard 8051.



Internal Data Memory (00H to FFH)

00H to 7F H - Internal RAM

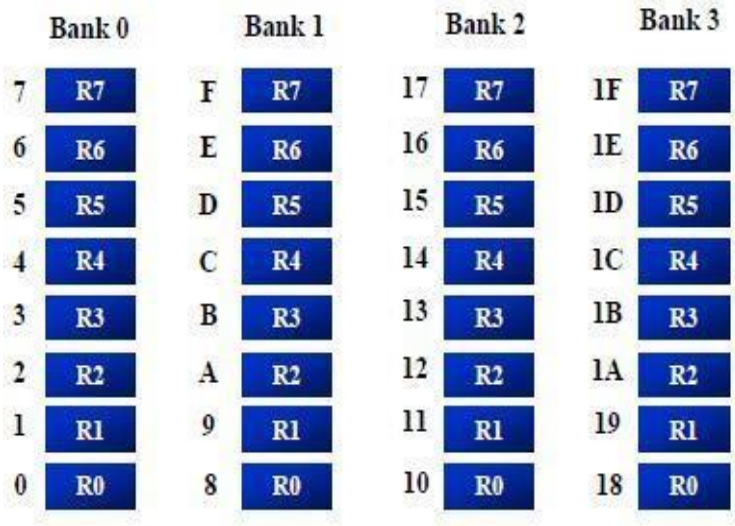
- 00H to 1FH : Register Banks
- 20H to 2FH : Bit Addressable RAM
- 30H to 7FH : General Purpose RAM

80H to FF H – Special Function Registers

Register Banks: 00H to 1FH

Four register banks (Bank0, Bank1, Bank2 and Bank3) each of 8-bits (total 32 bytes) are available. The default bank register is Bank0. The remaining Banks are selected with the help

of RS0 and RS1 bits of PSW Register. Each bank consists of 8 general-purpose registers R0 through R7. (R0, R1, R2, R3, R4, R5, R6, and R7)



Bit Addressable RAM: 20H to 2FH

The 8051 supports a special feature which allows access to bit variables. This is where individual memory bits in Internal RAM can be set or cleared. In all there are 128 bits numbered 00H to 7FH. Being bit variables any one variable can have a value 0 or 1. (20.1 mean it refers to 20^{th} address 1^{st} bit). A bit variable can be set with a command such as SETB and cleared with a command such as CLR.

General Purpose RAM: 30H to 7FH

These 80 bytes of Internal RAM memory are available for general-purpose data storage. Access to this area of memory is fast compared to access to the main memory and special instructions with single byte operands are used. The general purpose RAM can be accessed using direct or indirect addressing modes.

2. Special Function Registers (SFR)

Special Function Registers (SFR) **[April/May2017]**

The special function registers (SFRs) are mapped in the upper 128bytes of internal data memory address. The SFR registers are located within the Internal Memory in the address range 80H to FFH Each SFR has a very specific function. Each SFR has an address (within the range 80H to FFH) and a name which reflects the purpose of the SFR. The SFRs are accessed by direct addressing only. Some SFRs are also bit addressable as is the case for the bit area of RAM. This feature allows the programmer to change only what needs to be alteredleaving the remaining bits in that SFR unchanged. Not all of the addresses from 80H to FFH are used for SFRs. Only the addressed ones can be used in programming SFRs and equivalent internal RAM addresses.

80	P0	90	P1
81	SP	98	SCON
82	DPL	99	SBUF
83	DPH	A0	P2
87	PCON	A8	IE
88	TCON	B0	P3
89	TMOD	B8	IP
8A	TL0	D8	PSW
8B	TL1	E0	ACC
8C	TH0	F0	B
8D	TH1		

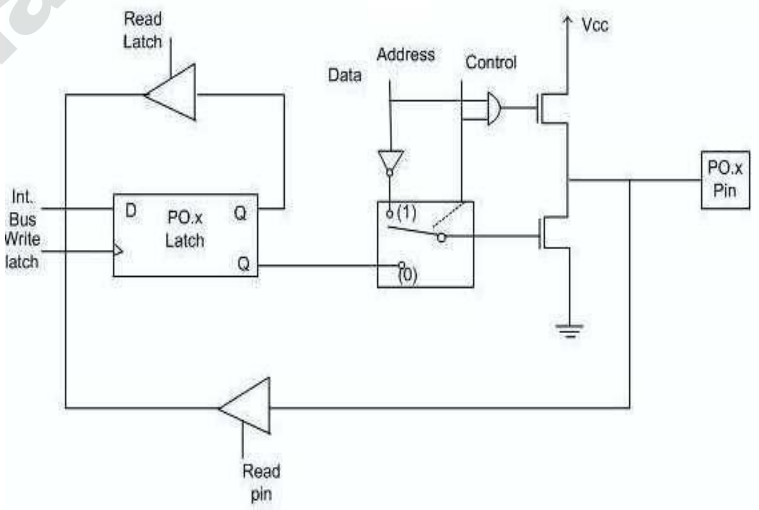
2. I/O PORT PINS, PORTS AND CIRCUITS

Draw the pin diagram of 8051 Microcontroller and explain the Input/output lines indetail. (16) [**May/Jun 2014**]

8051 microcontrollers have 4 I/O ports each comprising of 8 bits which can be configured as inputs or outputs. Accordingly, total of 32 input/output pins enabling the microcontroller to be connected to peripheral devices that are available for use. Each port of 8051 has bidirectional capability. Port1, 2, 3 are called 'quasi bidirectional port'.

Port 0 Pin Structure:

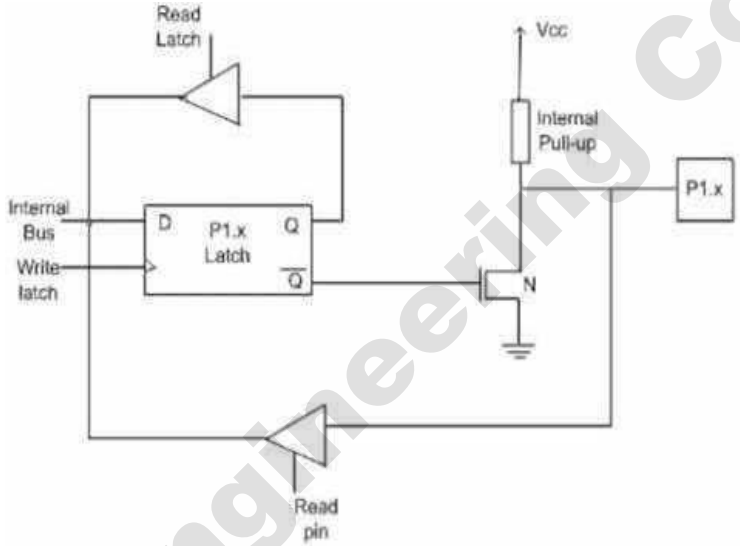
Port 0 has 8 pins (P0.0-P0.7). Port 0 is called bidirectional port as it floats (tristated) when configured as input. It can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port. If external memory is used then thelower address byte (addresses A0-A7) is applied on it. Otherwise, all bits of this port are configured as inputs/outputs.



Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch.In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is'1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required. Port-0 latch is written to with 1's when used for external memory access.

Port 1 Pin Structure:

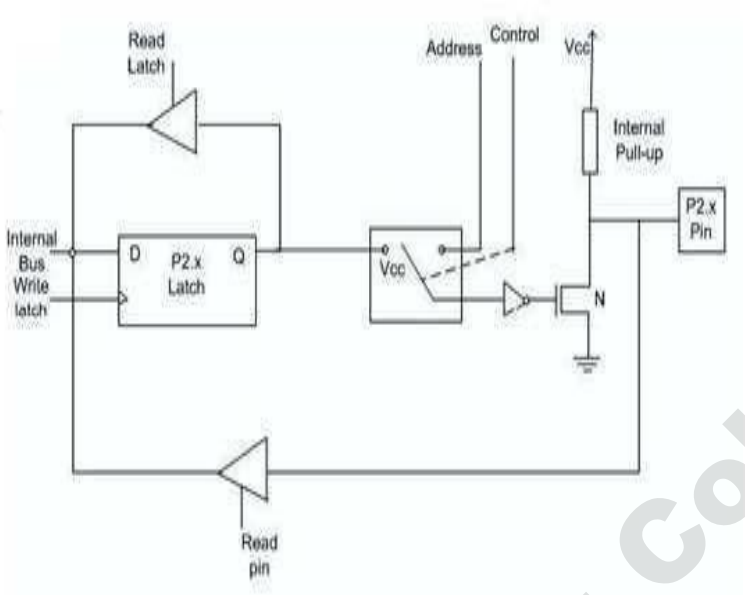


Port1 has 8 pins (P1.0 -P1.7) P1 is a true I/O port, because it doesn't have any alternative functions as is the case with P0, but can be configured as general I/O only. It has a pull-up resistor built-in. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

Port 2 Pin Structure:

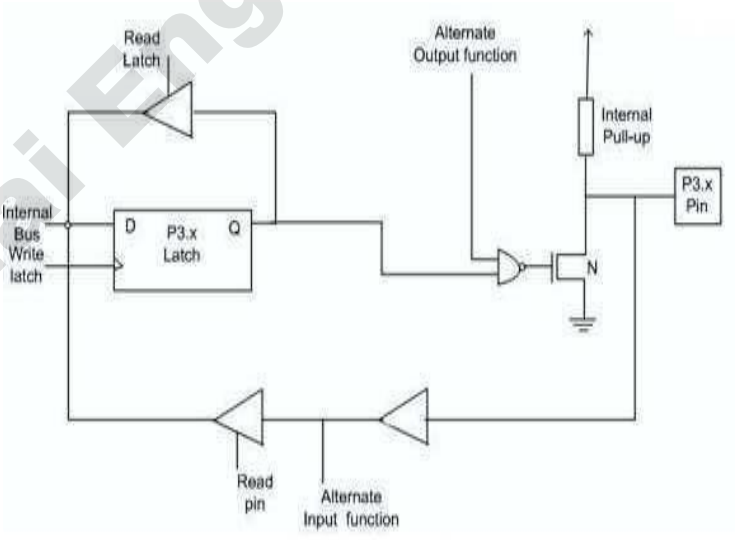
Port 2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port 1. Port 2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability. Port2 has 8-pins (P2.0-P2.7) Port 2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port 2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability. P2 acts similarly to P0 when external memory is used. Pins of this port occupy addresses intended for external memory chip. This time it is about the higher address byte with addresses A8-A15.When no memory is added, this port can be

used as a general input/output port showing features similar to P1.



Port 3 Pin Structure:

Port 3 has 8 pin (P3.0-P3.7). Port3 pins have alternate functions. All the port pins can be used as general I/O, but they also have an alternative function. In order to use these alternative functions, a logic one (1) must be applied to appropriate bit of the P3 register. In terms of hardware, this port is similar to P0, with the difference that its pins have a pull-up resistor built-in. Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'.To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.



Alternate functions of Port-3 pins

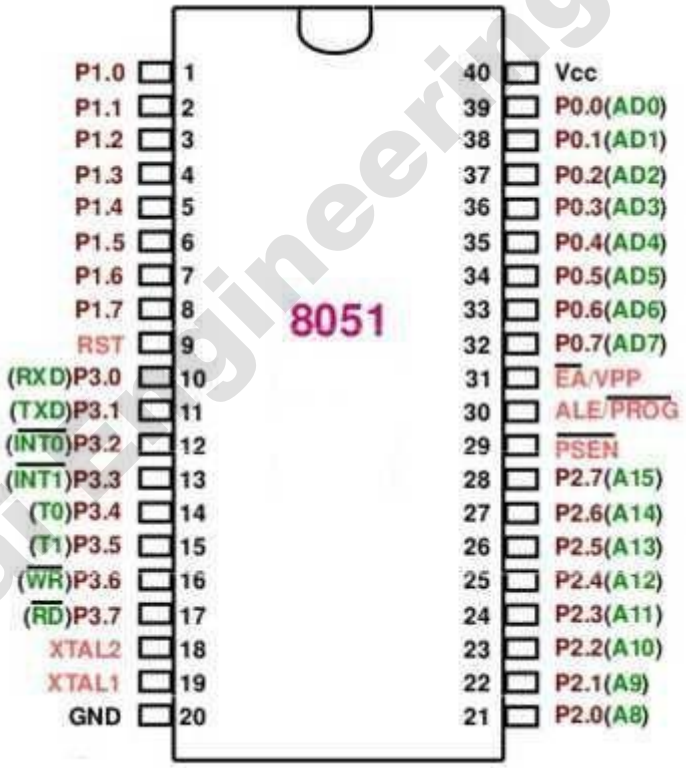
P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
RD	WR	T1	T0	INT1	INT0	TxD	RxD

RD – Read data control output.
WR – Write data control output.
T1 – Timer / Counter1 external input or test pin.
T0 – Timer / Counter0 external input or test pin.
INT1- Interrupt 1 input pin.
INT 0 – Interrupt 0 input pin.
TXD – Transmit data pin for serial port in UART mod e.
RXD - Receive data pin for serial port in UART mode.

3. SIGNALS OF 8051

Draw the pin diagram of 8051 microcontroller and explain the functions of each pin. (10) [**Nov/Dec 2011]**

Draw the pin diagram of 8051 Microcontroller and explain the Input/output lines in detail.(16) [**May/Jun 2014**]



The 8051 microcontroller is available as a 40 pin DIP chip and it works at +5 volts DC

XTAL1, XTAL2: These two pins are connected to Quartz crystal oscillator which runs theon- chip oscillator

RST: The RESET pin is an input pin and it is an active high pin. When a high pulse is applied to this pin the microcontroller will reset and terminate all activities.

EA: This pin is an active low pin. This pin is connected to ground when microcontroller is accessing the program code stored in the external memory and connected to Vcc when it is accessing the program code in the on chip memory

PSEN (Program store enable): This is an output pin which is active low. When the microcontroller is accessing the program code stored in the external ROM, this pin is connected to the OE (Output Enable) pin of the ROM.

ALE (Address latch enable): This is an output pin, which is active high**.** When connected to external memory , port 0 provides both address and data i.e address and data are multiplexed through port 0 .This ALE pin will demultiplex the address and data bus .When the pin is High , the AD bus will act as address bus otherwise the AD bus will act as Data bus.

P0.0- P0.7 (AD0-AD7): The port 0 Pins multiplexed with Address/data pins .If the microcontroller is accessing external memory these pins will act as address/data pins otherwise they are used for Port 0 pins.

P2.0- P2.7 (A8-A15) : The port2 pins are multiplexed with the higher order address pins **.**When the microcontroller is accessing external memory these pins provide the higher order address byte otherwise they act as Port 2 pins.

P1.0- P1.7: These 8-pins are dedicated for Port1 to perform input or output port operations.

P3.0- P3.7: These 8-pins are meant for Port3 operations and also for some control operations like Read, Write, Timer0, Timer1, INT0, INT1, RxD and TxD

4. INSTRUCTION SET OF 8051

With Example, Explain the Arithmetic and Branching Instruction of 8051 Microcontroller. (8) [**Nov / Dec 2012]**

Explain about Arithmetic and control instruction set in 8051. (10) [**Apr/May 2015**]

The instructions of 8051 microcontroller is divided into

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. Branch Instructions
5. Boolean Variable Instruction

Data Transfer Instructions

MOVA, Rn	A = Rn
MOVA, direct	A = (direct)

MOVA, @Ri	A = @Ri
MOVA, #data	A = data
MOV Rn, A	Rn = A
MOV Rn, direct	Rn = (direct)
MOV Rn, #data	Rn = data
MOV direct, A	(direct) = A
MOV direct, Rn	(direct) = Rn
MOV direct1, direct2	(direct1) = (direct2)
MOV direct, @Ri	(direct) = @Ri
MOV direct, #data	(direct) = #data
MOV @Ri, A	@Ri = A
MOV @Ri, direct	@Ri = (direct)
MOV @Ri, #data	@Ri = data
MOVDPTR,#data16	DPTR = data16
MOVCA,@A+DPTR	A = Code byte pointed by A+DPTR
MOVCA,@A+PC	A = Code byte pointed by A+PC
MOVCA, @Ri	A = Code byte pointed by Ri
MOVX A, @DPTR	A = External data pointed by DPTR
MOVX @Ri,A	@Ri = A (Externaldata-8bitaddress)
MOVX @DPTR,A	DPTR = A
PUSH direct	Push (direct) to the stack
POP direct	Pop (direct) from stack
XCH Rn	Exchange A with Rn

Arithmetic Instructions

ADD A, Rn	A = A+Rn
ADD A, direct	A = A +(direct)
ADD A, @Ri	A = A +@Ri
ADD A, #data	A = A+data
ADDC A, Rn	A = A+Rn+C
ADDC A, direct	A = A+(direct)+C
ADDC A, @Ri	A = A+@Ri+C
ADDC A, #data	A = A+data+C
SUB A, Rn SUB	A = A-Rn
A, direct	A = A - (direct)

SUB A, @Ri	A = A - @Ri
SUB A, #data	A = A-data
SUBBA, Rn	A = A -Rn- C
SUBB A, direct	A = A -(direct)-C
SUBB A, @Ri	A = A -@Ri-C
SUBB A, #data	A = A -data-C
DEC A	A = A- A
DEC Rn	$R_n = R_n - 1$
DEC direct	(direct) = (direct) - 1
INC A	A = A + 1
INC Rn	$R_n = R_n + 1$
INC direct	(direct) = (direct) + 1
INC @Ri	@Ri = @Ri + 1
INC DPTR	DPTR = DPTR + 1
DIV AB	A/B A = quotient B = Remainder
MULAB	A * B A = low byte (A*B) , B = high byte (A*B)
DAA	Decimal adjust accumulator

Logical Instructions

ANLA, Rn	A = A AND Rn
ANLA, direct	A = A AND (direct)
ANLA, @Ri	A = A AND @Ri
ANLA, #data	A = A AND data
ANL direct, A	A = (direct)ANDA
ANL direct, #data	A = (direct) AND data
ORLA, Rn	A = A OR Rn
ORLA, direct	A = A OR(direct)
ORLA, @Ri	A = A OR @Ri
ORLA, #data	A = A OR data
ORL direct, A	(direct) = (direct)ORA
ORL direct, #data	(direct) = (direct)OR data
XRLA, Rn	A = A EXOR Rn
XRLA, direct	A = AEXOR (direct)
XRLA, @Ri	A = A EXOR @Ri
XRLA, #data	A = A EXOR data

XRL direct, A	(direct) = (direct)EXORA
XRL direct, #data	(direct) = (direct)EXOR data
CLRA	A = 00H
CPLA	A = A’
RLA	Rotate Accumulator Left
RLCA	Rotate Accumulator left through carry
RRA	Rotate Accumulator right
RRCA	Rotate Accumulator right through carry
SWAPA	Swap nibbles within Accumulator

Branch Instructions

ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
RETI	Return from interrupt
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP Relative Address	Short jump
JMP @A+DPTR	Jump indirect
JZ Relative Address	Jump if Zero
JNZ Relative Address	Jump if Not Zero
JC Relative Address	Jump if C set
JNC Relative Address	Jump if C not set
JB bit,Relative Address	Jump if specified bit set
JNB bit,Relative Address	Jump if specified bit not set
JBC bit,Relative Address	if specified bit set, clear it and jump
CJNE A,direct,rel	Compare and Jump if Not Equal
CJNE A,#data,rel	Compare and Jump if Not Equal
CJNE Rn,#data,rel	Compare and Jump if Not Equal
CJNE @Ri,#data,rel	Compare and Jump if Not Equal
DJNZ Rn,Relative Address	Decrement and Jump if Not Zero
DJNZ direct,Relative Address	Decrement and Jump if Not Zero
NOP	No Operation

Boolean Variable Instruction

CLR	C	Clear C
CLR	bit	Clear direct bit
SETB	C	Set C
SETB	bit	Set direct bit
CPL	C	Complement c
CPL	bit	Complement direct bit
ANL	C,bit	AND bit with C
ANL	C,/bit	AND NOT bit with C
ORL	C,bit	OR bit with C
ORL	C,/bit	OR NOT bit with C
MOV	C,bit	MOV bit to C
MOV	bit,C	MOV C to bit

Write a program to bring in data in serial form and send it out in parallel form using 8051 [**April/May 2015**]

```
        MOV R0, #08          ;counter for 8 bits
        SETB P0.0            ;make P0.0 an input port
BACK: MOV C, P0.0            ;move data from p0.0 into the carry bit
        RRC A                ;rotate right, the data goes from 'cy' int o A
        DJNZ R0, BACK        ;repeat until all 8 bits are moved in
        MOV P1, A            ;the data is now transferred in parallel to P1
        END
```

5. ADDRESSING MODES

Discuss in detail about the Addressing Modes of 8051 Microcontroller. **[April/May2017]**

The 8051 instructions use eight addressing modes. These are:

1.	Register	5.	Relative
2.	Direct	6.	Absolute
3.	Indirect	7.	Long
4.	Immediate	8.	Indexed

1. Register Addressing

Data is available in the register specified in the instruction.

For example, MOV A, R0

2. **Direct Addressing**

   The address of the data is available in the instruction format.

   For example MOV A, 088H; Moves content of the address 88H to Accumulator.

3. **Indirect Addressing**

   The address of data is available in the R0 or R1 registers as specified in the instruction.
   For example- MOV A, @R0 moves content of address pointed by R0 to A.

4. **Immediate Addressing**

   Data is immediately available in the instruction

   MOV A, #77 Move the Data 77 to the Accumulator.

5. **Relative Addressing**

   Sometimes this is also called program counter relative addressing. This addressing mode is used only with certain jump instructions. The range for such a jump instruction is –128 to +127 locations.

   JZ Relative Address

6. **Absolute Addressing**

   There are only two instructions that use this addressing: **ACALL** (absolute call) and **AJMP** (absolute jump). These instructions perform branching within the current 2K page of program memory.

7. **Long Addressing**

   Only two instructions use this addressing mode. These instructions are **LCALL*addr16*** and **LJMP*addr16***. These instructions enable the program to branch to anywhere within the full 64 K-bytes of program memory address space.

8. **Indexed Addressing**

   In this mode the 16-bit address in a base register is added to a positive offset to form an effective address for the jump indirect instruction **JMP @A+DPTR**, and the two move code byte instructions **MOVC A,@A+DPTR** and **MOVC A,@A+PC**. The base register in the jump instruction is the data pointer and the positive offset is held in the accumulator. For the move instructions the base register can either be the data pointer or the program counter, and again the positive offset is in the accumulator.

UNIT V INTERFACING MICROCONTROLLER

Programming 8051 Timers - Serial Port Programming - Interrupts Programming – LCD & Keyboard Interfacing - ADC, DAC & Sensor Interfacing - External Memory Interface- Stepper Motor and Waveform generation-Comparison of Microprocessor, Microcontroller, PIC and ARM Processors.

PART-A (2 MARKS)

1. Which register is used for serial programming in 8051? Illustrate it. [Apr/May 2015]

- SBUF (serial buffer) register
  Byte of data to be transferred via the TxD line must be placed in the SBUF register. SBUF holds the byte of data when it is received by the RxD line
- SCON (Serial control) register.
  Used to program the start bit, stop bit and data bits.

2. Name the five interrupt sources of 8051? [April/May2017, May/June2007] [April/May2008]

External Interrupt 0 (INT0)
External Interrupt 1 (INT1)
Timer Interrupt 0 (TF0)
Timer Interrupt 1 (TF1)
Serial Port Interrupt.(TI or RI)

3. What is the significance of EA line of 8051 microcontroller? [May/Jun 2014]

It is an active low I/P to 8051 microcontroller. When EA = 0, then 8051 microcontroller access from external program memory (ROM) only. When EA = 1, then it access internal and external program memories (ROMS).

4. What is baud rate in 8051? [May/June 2011]

The Baud Rate is determined based on the oscillator's frequency when in mode 0 and 2. In mode 0, the baud rate is always the oscillator frequency divided by 12. This means if you're crystal is 11.059 MHz; mode 0 baud rate will always be 921,583 baud. In mode 2 the baud rate is always the oscillator frequency divided by 64, so an 11.059 MHz crystal speed will yield a baud rate of 172,797.

5. Name the sensors used in a microprocessor based temperature controller. [Apr/May 2011, April/May2017]

In order to sense the temperature either thermistor or thermocouple can be used as the transducer that converts heat energy into electrical energy.

6. Mention any two applications that use ADC and DAC. [Apr/May 2011]

a. Temperature controller b. Stepper motor

7. Differentiate between timers and counters. Draw the diagram of TCON in 8051. [Apr/May 2015]

A counter is a device that records the number of occurrences of a particular event. Modern applications, counters are based on electronic devices and the counters are sequential logic circuit designed to record the number of electric pulses fed into the counter. A timer is an application of the counters where a certain signal with a fixed frequency (hence period) is counted to record the time.

TCON is bit addressable. The address of TCON is 88H

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1: Timer1 overflow flag. It is set when timer rolls from all 1s to 0s

TR1: Timer1 run control bit. Set to 1 to start the timer / counter

TF0: Timer0 overflow flag.

TR0: Timer0 run control bit

IE1: Interrupt1 edge flag. Set by hardware when an external interrupt edge is detected.

IE0: Interrupt0 edge flag.

IT1: Interrupt1 type control bit. Set/ cleared by software to specify falling edge / low level triggered external interrupt

IT0: Interrupt0 type control bit.

8. How to change the direction of stepper motor from clockwise direction to anti clockwise direction using a program segment. [Nov/Dec 2012]

By altering and switching sequence, the motor can be made to run with incremental motionof half the full step value

9. What is the use of Vref pin in the ADC? [Nov/Dec 2012]

Vref pin in the ADC is used to compare the input signal with the reference signal.

10. List any four applications of stepper motor [Nov/Dec 2014]

a. Dot matrix printer
b. Washing machine
c. Consumer Electronics – stepper motors, In camera s for automatic digital camera focus and zoom function
d. Medical- Stepper motor are used inside medical scanners, samplers and also found inside digital dental photography, fluid pumps, respirators and blood analysis machinery

11. What is the necessity to interface DAC with microcontroller? [Nov/Dec 2014]

The digital to analog converter is a device used to convert digital pulses to analog signals. DAC is interfaced with microcontroller for many applications such as generating sine waveform.

12. What is a serial data buffer?

Serial data buffer is a special function register and it initiates serial transmission when byte is written to it and if read, it reads received serial data. It contains two independent registers internally. One of them is a transmit buffer, which is a parallel-in serial-out register. The other is a receive buffer, which is a serial-in parallel-out register

13. What are timer registers?

Timer registers are two 16-bit registers and can be accessed as their lower and upper bytes. TLO represents the lower byte of the timing register 0, while THO represents higher bytes of the timing register 0. Similarly, TLI and THI represent lower and higher bytes of timing register 1. These registers can be accessed using the addresses allotted to them, which lie in the special function registers address range, i.e., 801 H to FF.

14. What is the use of timing and control unit?

Timing and control unit is used to derive all the necessary timing and control signals required for the internal operation of the circuit. It also derives control signals that are required for controlling the external system bus.

15. When are timer overflow bits set and reset?

The timer overflow bits are set when timer rolls over and reset either by the execution of an RET instruction or by software, manually clearing the bits. The bits are located in the TCON register along with timer run control (TRn) bits.

16. Explain the mode (0 and1) operation of the timer. (April/May2012)

The operations are as follows:

• Timer mode 0 and 1 operations are similar for the 13 bit (mode) or 16 bit (mode 1) counter. When the timer reaches the limits of the count, the overflow flag is set and the counter is reset back to zero. The modes 0 and 1 can be used to time external events.They can be used as specific time delays by loading them with an initial value before allowing them to execute and overflow.

17. What are the different modes in which timer 2 can operate?

The two different modes in which Tmer 2 operates are.

i. **Capture mode**: Timer 2 operates as free running clocks, which saves the timers value on each high to low transition. It can be used for recording bit lengths when receiving Manchester-encoded data.

ii. **Auto-reload mode**: When the timer overflows, value is written into TH2/TL2 registers from RCA P2H/RCA P21 registers. This feature is used to implement a system watch dog timer.

18. What is the use of a watch dog timer?

A watching timer is used to protect an application in case the controlling microcontroller begins to run amok and execute randomly rather than the preprogrammed instructions written for the application.

19. Define interrupt.

Interrupt is defined as request that can be refused. If not refused and when an interrupt request is acknowledged, a special set of routine or events are followed to handle the interrupt.

20. What are the steps followed to service an interrupt?

The steps followed are:

I. Save the context register information.

II. Reset the hardware requesting the interrupt.

III. Reset the interrupt controller.

IV. Process the interrupt.

V. Restore the context information.

VI. Return to the previously executing code.

21. How can 8051 be interrupted?

There are five different ways to interrupt 8051. Two of these are from external electrical signals. The other three are caused by internal 8051 I/O hardware operations.

22. Give the format of the interrupt enable register. (April/ May2013)

The format of the interrupt enable register is, EA--ES ET1 EX1 ET0 EX0
The register is used to enable or disable all 8051 interrupts and to selectively enable or disable each of the five different interrupts.

EA: Disables all interrupts
Es: Enables or disable the serial port interrupt.
ET1: Enable or disable the timer 1 overflow interrupt.
EX1: Enable or disable external interrupt 1.
ET0: Enable or disable the timer 0 overflow interrupt.
EX0: Enable or disable external interrupt 0.

23. What is meant by nesting of interrupts?

Nesting of interrupts means that interrupts are re-enabled inside an interrupt handler. If another interrupt request codes in, while the first interrupt handler is executing, processor execution will acknowledge the new interrupt and jump to its vector.

24. How is the 8051 serial port different from other micro controllers? (Nov/Dec2013)

The 8051 serial port is a very complex peripheral and able to send data synchronously and asynchronously in a variety of different transmission modes.

25. Explain synchronous data transmission.

• In synchronous mode (mode 0), the instruction clock is used.
• Data transfer is initiated by writing to the serial data port address.
• Txd pin is used for clock output, while Rxd pin is for data transfer.
• When a character is received, the status of the data transfer is monitored by polling the RI-n bit in serial control register (SCON).

26. Give an application for synchronous serial communication.

An application for synchronous serial communication is RS–232.

27. When is an external memory access generated in 8051?

In 8051, during execution the data is fetched continuous. Most of the data is executed out of the 8051‟sbuilt-in control store. When an address is outside the internal control store, an external memory access is generated.

28. Give the priority level of the interrupt sources. (Nov/Dec2010)

Interrupt source Priority within a level IE0 (External INT0)
TF0 (Timer 0)
IE 1 (External INT 1)
TF 1 (Timer 1)
RI = TI (Serial port) Highest

29. What is the use of stepper motor?

A stepper motor is a device used to obtain an accurate position control of rotating shafts. A stepper motor employs rotation of its shaft in terms of steps, rather than continuous rotation as in case of AC or DC motor.

30. What is meant by key bouncing?

Microprocessor must wait until the key reach to a steady state; this is known as Key bounce.

31. Explain the operating mode0 of 8051 serial ports?

In this mode serial enters &exits through RXD, TXD outputs the shift clock.8 bits are transmitted/received:8 data bits(LSB first).The baud rate is fixed at 1/12 the oscillator frequency.

32. Explain the operating mode2 of 8051 serial ports? (April/May 2009&Nov/Dec2008)

In this mode 11 bits are transmitted(through TXD)or received(through RXD):a start bit(0), 8 data bits(LSB first),a programmable 9th data bit ,& a stop bit(1).ON transmit the 9th data bit (TB* in SCON)can be assigned the value of 0 or 1.Or for eg:, the parity bit(P, in the PSW)could be moved into TB8.On receive the 9thdatabit go in to the RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32or1/64 the oscillator frequency.

33. Explain the mode3 of 8051 serial ports? (April/May2008)

In this mode,11 bits are transmitted(through TXD)or received(through RXD):a start bit(0), 8 data bits(LSB first),a programmable9th data bit ,& a stop bit(1).In fact ,Mode3 is the same as Mode2 in all respects except the baud rate. The baud rate in Mode3 is variable. In all the four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode0 by the condition RI=0&REN=1.Reception is initiated in other modes by the incoming start bit if REN=1.

34. Write a program to mask the 0th&7thbit using8051?

MOV A,#data ANL A,#81
MOV DPTR,#4500 MOVX @DPTR,A LOOP SJMP LOOP

35. Write about CALL statement in 8051?

There are two subroutine CALL instructions. They are
*LCALL(Long CALL)
*ACALL(Absolute CALL)
Each increments the PC to the 1stbyte of the instruction & pushes them in to the stack.

36. Write a program to find the 2's complement using 8051?

MOV A,R0 CPL A INC A

37. Define baud rate. (May/June 2016)

Baud rate is used to indicate the rate at which data is being transferred. Baud rate = 1/Time for a bit cell.

38. Mention the features of serial port in mode 0. (Nov/Dec2015)

In this mode serial enters and exits through RXD, TXD outputs the shiftclock. 8 bits are transmitted
/received 8 data bits first (LSB first).The baudrate is fixed at 1/12 the oscillator frequency.

39. Which register is used for serial programming in 8051 microcontroller? Illustrate it. (Apr/May2015)

SBUF Register (Serial Buffer):

SBUF is an 8-bit register for serial communication in 8051.For a byte

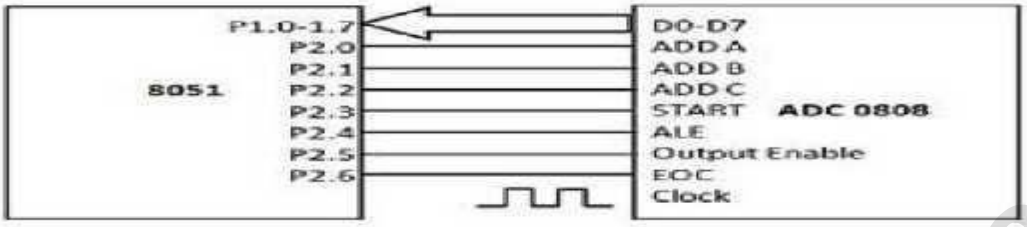
of data to be transferred via TxD line and holds the byte of data when it is received by 8051's RxD line.

SCON Register (Serial Control):

SCON is an 8 bit register used to program the start bit,stop bit and data bits of data framing among other things.

SM0	SM1	SM2	REN	TB8	RB8	T1	R1

40. How is A/D convertor interfaced with 8051? (Nov/Dec2015)



41. Compare polling and interrupt. (May/June 2016)

The 8051 microcontroller can do only one task at a time. In polling, the microcontroller continuously checks each port one by one according to the priority assigned to the ports, and if any device requires service, then it provides it. In interrupt, when the device requires service, it sends the request to microcontroller and the controller then provides service to it.

So essentially, the difference is that in polling, microcontroller has to check continuously whether any device is asking for request, while in interrupt the device itself sends the request and the controller satisfies it. And because microcontroller is freed from the task of checking each port, it can do other work.

PART-B (13 MARKS)

1. COUNTERS AND TIMERS OF 8051

Describe the different modes of operation of timers/counters in 8051 with its associated register. [Marks 10] [**April/May 2011]**

Explain the TMOD function register and its timer modes of operations. (6) [**Nov /Dec 2013]**

Describe the different modes of operation of timers/counters in 8051 with its associated registers.(10) [**Apr/May 2014**]

Explain the TMOD function register and its timer modes of operations.(8) [**Apr/May 2015, April/May2017**]

The 8051 has two timers Timer0 and Timer1. They can be used either as timer or event counter. Timer0 and Timer1 are 16 bit registers each can be accessed as two separate registers of low byte and high byte. The timer content is available in four 8-bit special function registers, viz, TL0, TH0, TL1 and TH1.respectively.

TH0	TL0
TH1	TL1
TCON	
TMOD	

The timer can act in "timer" function mode and "counter" function mode

- In the "timer" function mode, the counter is incremented in every machine cycle. Hence the clock rate is $1/12^{th}$ of the oscillator frequency.
- In the "counter" function mode, the register is incremented in response to a 1 to 0 transition at its corresponding external input pin (T0 or T1). It requires 2 machine cycles to detect a high to low transition. Hence maximum count rate is $1/24^{th}$ of oscillator frequency.

Timer0 Registers:

16 bit register of timer 0 is accessed as low byte and high byte. The low byte is called TL0 and high byte is called TH0. Minimum value is 0000 and maximum value is FFFF can be loaded in the Timer 0 Register depending on the modes of operation

Timer1 Registers:

16 bit register of timer 1 is accessed as low byte and high byte. The low byte is called TL1 and high byte is called TH1. Minimum value is 0000 and maximum value is FFFF can be loaded in the Timer 0 Register depending on the modes of operation

The operation of the timers/counters is controlled by two special function registers, TMOD and TCON.

TMOD (Timer Mode Register)

Both timers use the same register called TMOD to set the various timer operation modes. TMOD is a 8 bit register in which the lower 4 bits are set aside for Timer 0 and the upper 4 bits for Timer 1. In each case the lower 2 bits are used to set the Timer mode and upper 2 bits to specify the operation.

7	6	5	4	3	2	1	0
Gate	**C/T**	**M1**	**M0**	**Gate**	**C/T**	**M1**	**M0**
Timer 1				**Timer 0**			

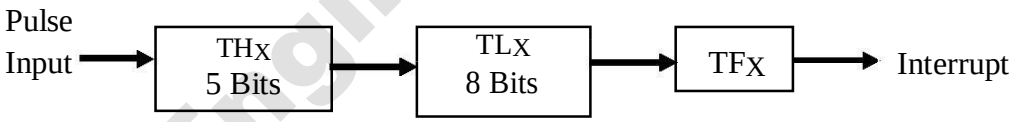
Gate: START or STOP of Timer/Counter.

C/T': It is used for the selection of Counter/Timer.

M1 & M0: Mode Select Bits

M1	M0	Mode
0	0	Mode 0 13-bit timer mode
0	1	Mode 1 16-bit timer mode
1	0	Mode 2 8-bit auto reload
1	1	Mode 3Split timer mode

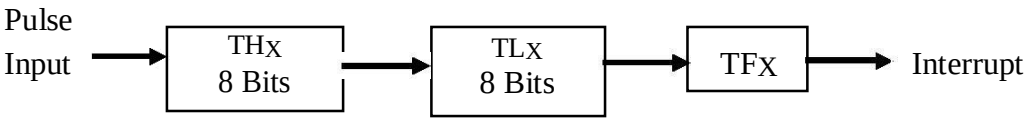
Timer Mode-0 13-bit timer mode



In this mode, the timer is used as a 13-bit UP counter. The lower 5 bits of TH_X and 8 bits of TL_X are used for the 13 bit count. Upper 3 bits of TH_X are ignored. When the counter rolls over from all 0's to all 1's, TF flag is set and an interrupt is generated. The input pulse is obtained from the previous stage. If TR bit is 1 and Gate bit is 0, the counter continues counting up. If TR bit is 1 and Gate bit is 1, then the operation of the counter is controlled by INTX input. This mode is useful to measure the width of a given pulse fed to INTX input. The ranges of values are 0000 H to 1FFF H. When the timer reaches the maximum value it rolls over to 0000H.

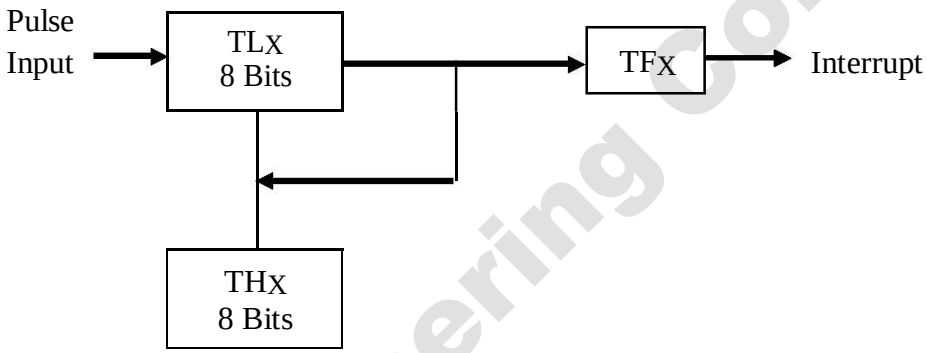
Timer Mode116-bit Timer Mode

Mode1 is similar to mode0 except TL_X is configured as a full 8-bit counter. When the mode bits are set to 01 in TMOD. The Timer operates in 16-bit mode. The ranges of values are 0000H to FFFF H. When the timer reaches the maximum value it rolls over to 0000H.



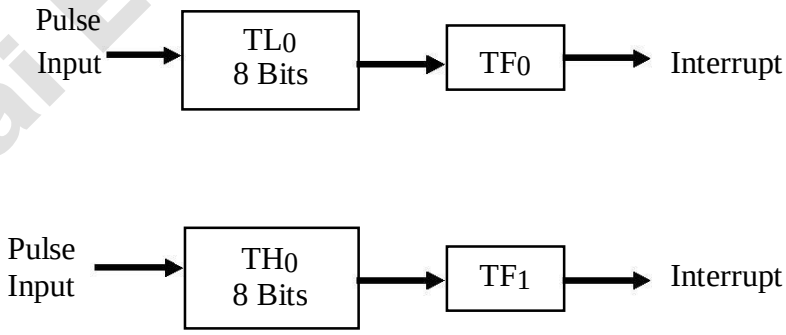
Timer Mode 2 8-bit Auto Reload

This is a 8 bit counter/timer operation. It allows the value from 00 to FF. Counting is performed in TL_X while TH_X store a constant value. In this mode when the timer overflows i.e. TL_X becomes FFH, the value in TH_X is reloaded again in the TL_X register and the Counting continues.. For example if we load TH_X with 50H then the timer in mode 2 willcount from 50H to FFH. After that 50H is again reloaded.



Timer Mode 3

Timer 1 in mode-3 simply holds its count. Timer 0 is used in mode 3. The effect is same as setting TR1=0. Timer0 in mode-3 establishes TL0 and TH0 as two separate counters



TCON Timer Control Register

TCON is 8 bit register. Upper 4 bits are used to store TF and TR bits of both timer0 and timer1. The lower four bits are set aside for controlling the interrupt bits.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1: Timer1 overflow flag. It is set when timer rolls from all 1s to 0s

TR1: Timer1 run control bit. Set to 1 to start the timer / counter

TF0: Timer0 overflow flag.

TR0: Timer0 run control bit

IE1: Interrupt1 edge flag. Set by hardware when an external interrupt edge is detected.

IE0: Interrupt0 edge flag.

IT1: Interrupt1 type control bit. Set/ cleared by software to specify falling edge / low level triggered external interrupt

IT0: Interrupt0 type control bit.

2. SERIAL PORT COMMUNICATION

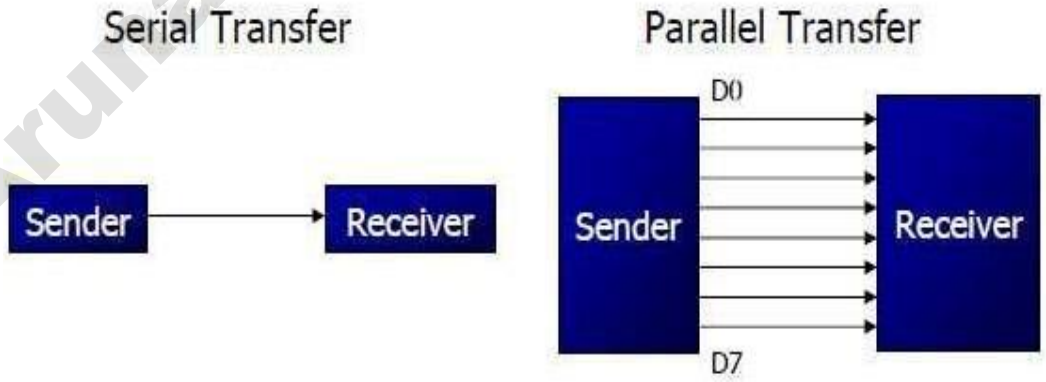
Write briefly about the operating modes for serial port of 8051 microcontroller. [Marks 4] [**April/May 2011]**

Explain the operation of Serial Port with Associated Register. (8) [**Nov / Dec 2012]**

Write a program to bring in data in serial form and send it out in parallel form using 8051. (6) [**Apr/May 2015**]

Serial Communication is used for transferring data between two systems. One of the 8051s many powerful features is its integrated *UART*, otherwise known as a serial port.

For serial data transmission, at the transmitting end, the byte of data must be converted to serial bits using parallel-in-serial-out shift register at the receiving end, there must be a serial in parallel-out shift register to receive the serial data and pack them into byte. Pins TxD (P3.1) and RxD (P3.0) are used for transmitting and receiving the data serially



Serial Communication using two methods

1. Synchronous Serial Data Communication – transfers b locks of data
2. Asynchronous Serial Data Communication – transfers single byte at a time

Basic Modes of data transmission

a) **Simplex Mode**: Data is transmitted only in one direction from the transmitter to the receiver over a single communication channel.

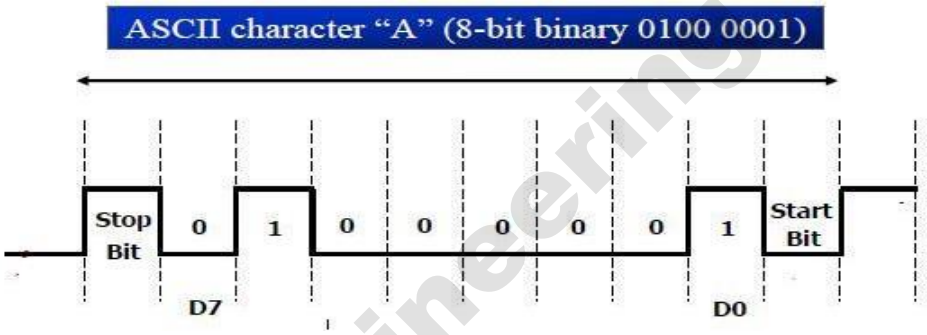
b) **Half Duplex Mode**: Data transmission may take place in either direction, but at a time data may be transmitted only in one direction.

c) **Full Duplex Mode**: Data transmission may take place in both directions simultaneously.

8051 contains built in UART

Asynchronous serial communication and data framing

It is used for character oriented transmission. Each character is placed in between the start bit and stop bit. This is called **framing.**



Start bit is always one bit and it will be low signal. Stop bit is represented by 1 or 2 bits and the stop bit must be high. Data can be 7 bits or 8 bits wide. The data is nothing but the ASCII value of the character.

8051 contains two reisters SCON and SBUF for serial transmission.

SCON Register
SBUF Register (Transmit & Receive)

Serial Interface

The serial port of 8051 is full duplex, i.e., it can transmit and receive simultaneously. The register SBUF is used to hold the data. The special function register SBUF is physically two registers. One is, write-only and is used to hold data to be transmitted out of the 8051 via TXD. The other is, read-only and holds the received data from external sources via RXD.

Data Transmission

Transmission of serial data begins at any time when data is written to SBUF. Pin P3.1 (Alternate function bit TXD) is used to transmit data to the serial data network. TI is set to 1 when data has been transmitted. This signifies that SBUF is empty so that another byte can be sent.

Data Reception

Reception of serial data begins if the receive enable bit is set to 1 for all modes. Pin P3.0 (Alternate function bit RXD) is used to receive data from the serial data network. Receive interrupt flag, RI, is set after the data has been received in all modes. The data gets stored in SBUF register from where it can be read.

Serial Port Control Register (SCON)

Register SCON controls serial data communication.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SM1	MODE	Description	Baud Rate
0	0	Mode 0	Shift Register	$f_{osc}/12$
0	1	Mode 1	8 Bit UART	Variable
1	0	Mode 2	9 Bit UART	$f_{osc}/32$, $f_{osc}/64$
1	1	Mode 3	9 Bit UART	Variable

SM2: Used for multiprocessor communication.
REN: set or cleared by software to enable/disable reception.
TB8: Transmitted bit 8, not widely used.
RB8: Received bit 8.
TI : Transmit Interrupt Flag –set by the hardware at the beginning of the stop bit in mode 1, must be cleared by software.
RI : Receive Interrupt Flag –set by the hardware ha lfway through the stop bit time in mode1, must be cleared by software.

Mode - 0 Shift Register Mode

In this mode, the serial port works like a shift register and the data transmission works synchronously with a clock frequency of f_{osc} /12. Serial data is received and transmitted through RXD and TXD. 8 bits are transmitted/ received at any time. Pin TXC outputs the shift clock pulses of frequency f_{osc} /12, which is connected to the external circuitry for synchronization. The shift frequency or baud rate is always 1/12 of the oscillator frequency.

Mode –1 8 bit UART

In mode-1, the serial port functions as a standard Universal Asynchronous Receiver Transmitter (UART) mode. 10 bits are transmitted through TXD or received through RXD. The 10 bits consist of one start bit (which is usually '0'), 8 data bits (LSB is sent first/received first), and a stop bit (which is usually '1'). Oncereceived, the stop bit goes into RB8 in the special function register SCON. The baud rate is variable.

Mode - 2 Multiprocessor Mode 9 Bit UART

11 bits are transmitted through TXD or received through RXD, a start bit (0), 8 data bits (LSB first), a programmable 9th bit and a stop bit (1).On transmission, the 9th data bit (TB8 in SCON) can be assigned the value 0 or 1. Or, for example, the parity bit (P in the PSN) could be moved into TB8. On receive; the 9th bit goes into RB8 in SFR SCON, which the stop bit is ignored. The bandwidth is programmable to either 1/32 or 1/64 of oscillator frequency.

Mode – 3 9 Bit UART

11 bits are transmitted through TXD or received through RXD: a start bit, 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). In fact, Mode 3 is same as Mode 2 in all respects except the band rate. The band rate in Mode 3 is variable.

Two ways to increase the baud rate

Use a high frequency crystal

Change a bit in PCON register. If SMOD = 1, the baud rate will be doubled.

Power Mode Control Register

Register PCON controls processor power down, sleep modes and serial data baud rate. Only one bit of PCON is used with respect to serial communication. The seventh bit (b7)(SMOD) is used to generate the baud rate of serial communication.

SMOD	-	-	-	GF1	GF0	PD	IDL

SMOD: Serial baud rate modify bit PD: Power down Mode
GF1: General purpose user flag bit 1 IDL : Idle Mode
GF0: General purpose user flag bit 0

program to bring in data in serial form and send it out in parallel form using 8051

```
        MOV R0, #08          ; counter for 8 bits
        SETB P0.0            ; make P0.0 an input port
BACK:   MOV C, P0.0          ; move data from p0.0 into the carry bit
        RRC A                ; rotate right, the data goes from ‘cy’ int o A
        DJNZ R0, BACK        ; repeat until all 8 bits are moved in

        END
```

3. INTERRUPTS

Explain the interrupt structure of 8051 microcontroller with suitable diagram.(8) [**Nov/Dec 2014**].

Briefly write about the IE and IP register in 8051 microcontroller. (6) [**Nov/Dec 2011]**

The 8051 has five sources of interrupts.

External Interrupt 0 (INT0)
External Interrupt 1 (INT1)
Timer Interrupt 0 (TF0)
Timer Interrupt 1 (TF1)
Serial Port Interrupt.(TI or RI)

These interrupts occur because of

1. Timers overflowing
2. Receiving character via the serial port
3. Transmitting character via the serial port
4. Two "external events

Timer 0 overflow: This is indicated by TF0 in TCON, being set

Timer 1 overflow: This is indicated by TF1 in TCON, being set

Serial port interrupts (RI and TI): Whenever a data byte is received, an interrupt bit, RI is set to 1 in SCON register. When a data byte is transmitted an interrupt bit TI, is set in SCON. They are ORed together to provide a single interrupt to the processor. These flags must be reset by software instruction to enable the next data communication operation.

External signal at pin INTO (P3.2): When a high-to-low edge signal is received onP3.2, the external interrupt 0 edge flag IE0 (TCON.1) is set. This flag is cleared when the processor branches to the subroutine. When the external interrupt signal control bit IT0 (TCON.0) is set to 1 (by program) then interrupt is triggered by falling edge signal. If IT0 is 0, a low-level signals in INTO triggers the interrupt.

External signal at pin INT1 (P3.3): Flags IE1 (TCON.3) and IT1 (TCON.2) are similar to IE0 and IT0 in function.

IE Register
IP Register

When an interrupt occurs, the updated PC is pushed on the stack and is loaded with the vector address corresponding to the interrupt.

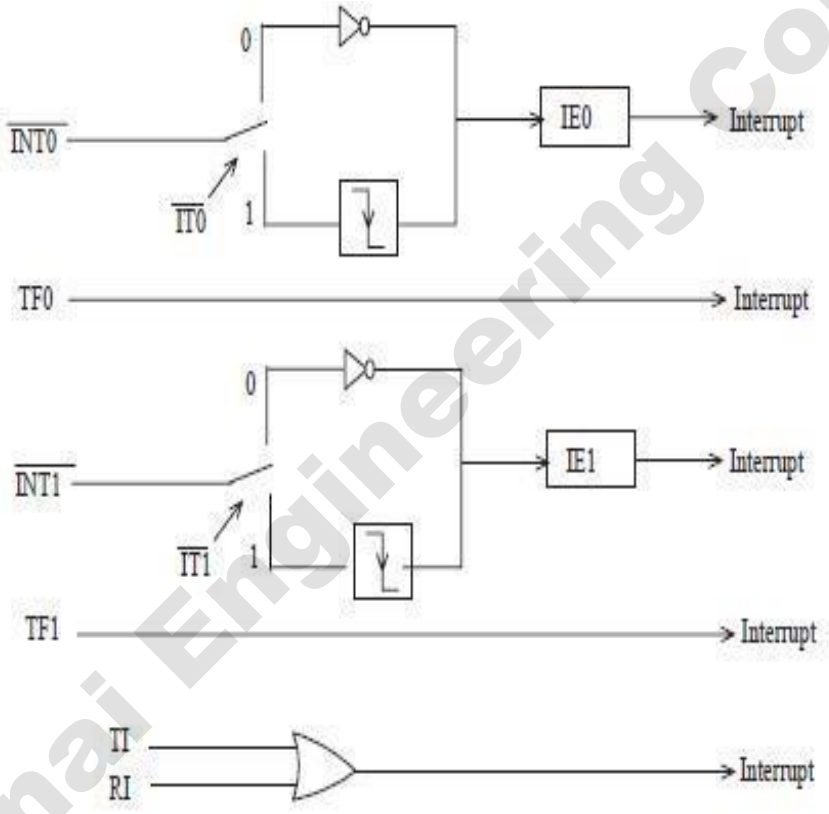
Sequence of Events after an interrupt

When an enabled interrupt occurs,

1. The PC is saved on the stack, low byte first.
2. Other interrupts of lower priority and same priority are disabled.
3. Except for the serial interrupt, the corresponding interrupt flag is cleared.
4. PC is loaded with the vector address corresponding to the interrupt.

When the handler executes 'IRET"

1. PC is restored by popping the stack.
2. Interrupt status is restored to its original value.



Interrupt Enable Register (IE)

EA	-	ET2	ES	ET1	EX1	ET0	ET1

EA: Enable all interrupts.
ET2: Reserved for future use.
ES: Enable Serial Port Interrupt.
= **1 Enable**
= **0 Disable**

ET1: Timer 1 Interrupt
EX1: External Interrupt 1
ET0: Timer 0 Interrupt
ET1: External Interrupt 0

Interrupt	Vector Address
IE0	0003
TF0	000B
IE1	0013
TF1	001B
Serial	0023

Interrupt Priority (IP):

This a bit addressable register, with byte address B8H. The priority of the interrupts is determined by the bits of IP. The bits which are set to 1, have a high priority and bits with 0 have low priority. The lower priority interrupt is serviced after higher priority interrupt is finished.

-	-	PT2	PS	PT1	PX1	PT0	PX0

PT2: Reserved for future use
PS: Serial Port Priority Interrupt
PT1: Priority of Timer 1 Interrupt
PX1: Priority of External Interrupt 1
PT0: Priority of Timer 0 Interrupt
PX0: Priority of External Interrupt 0

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in the SFR named IP (Interrupt Priority). A low-priority interrupt can be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source. If two interrupt requests of different priority levels are received simultaneously, the request of higher priority is serviced. If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence. If the flag for an enabled interrupt is found to be set (1), the interrupt system generates a CALL to the appropriate location in Program Memory; unless some other condition blocks the interrupt. Several conditions can block an interrupt, among them that an interrupt of equal or higher priority level is already in progress. The hardware-generated CALL causes the contents of the Program Counter to be pushed into the stack, and reloads the PC with the beginning address of the service routine.

Interrupt Priority Upon Reset (Highest to lowest Priority)

External Interrupt 0 (INT0)
Timer Interrupt 0 (TF0)
External Interrupt 1 (INT1)
Timer Interrupt 1 (TF1)
Serial Port Interrupt.(TI or RI)

4. LCD INTERFACING AND KEYBOARD INTERAFCING

How does one interface a 16 X 2 LCD Display using 8051 Micro-controller? (8 Marks) **[April/May 2010]**

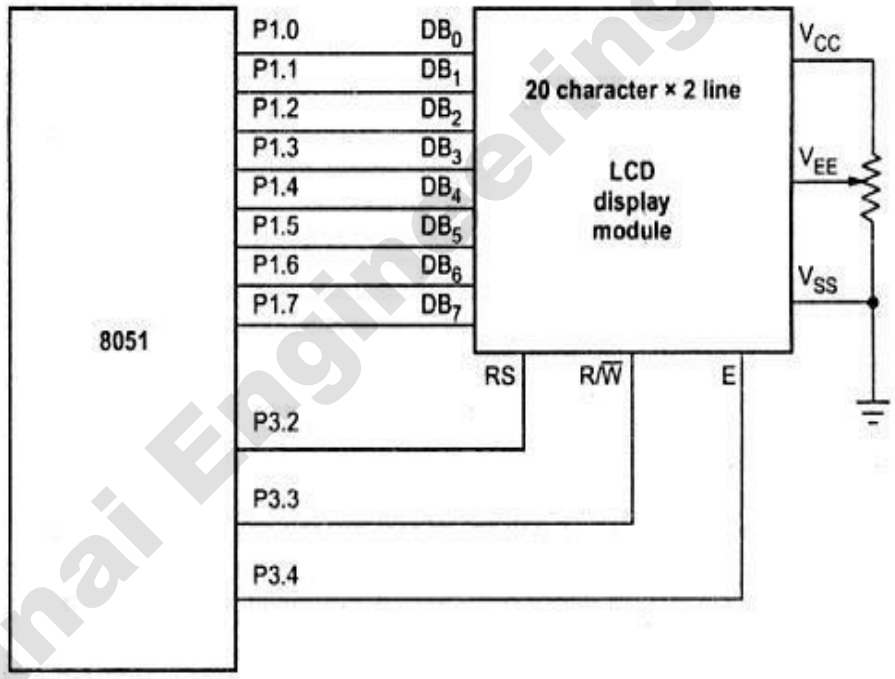
Explain the interfacing of LCD with 8051 microcontroller. (10) [**May/June 2011**]

How to interface an LCD display with microcontroller? Explain how to display a character using LCD display.(8) [**Nov/Dec 2014**].

How does one interface a 16 x 2 LCD display using 8051 Microcontroller?(6) [**Apr/May 2015**]

LCD is finding widespread use replacing LEDs for the following reasons:

- The declining prices of LCD
- The ability to display numbers, characters, and graphics
- Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD
- Ease of programming for characters and graphics



PIN DESCRIPTION

1. **VSS** - Ground
2. **VEE**- Supply Voltage
3. **VCC** - Contrast Setting
4. **RS** - Register Select
5. **R/W** - Read/Write Select
6. **E** - Chip Enable Signal

7-14 **DB0-DB7** - Data Lines

PIN SYMBOL FUNCTION

The LCD requires 3 control lines (**RS, R/W & E**) & 8 (or 4) data lines. The number on data lines depends on the mode of operation. If operated in 8-bit mode then 8 data lines + 3 control lines i.e. total 11 lines are required. And if operated in 4-bit mode then 4 data lines + 3 control lines i.e. 7 lines are required. When *RS* is low (0), the data is to be treated as a command. When RS is high (1), the data being sent is considered as text data which should be displayed on the screen.

When *R/W* is low (0), the information on the data bus is being written to the LCD. When R/W is high (1), the program is effectively reading from the LCD. Most of the times there is no need to read from the LCD so this line can directly be connected to GND thus saving one controller line. The *ENABLE* (E) pin is used to latch the data present on the data pins. A HIGH - LOW signal is required to latch the data. The LCD interprets and executes our command at the instant the E line is brought low. If you never bring E low, your instruction will never be executed.

LCD Command Codes:

LCD module has a set of preset command instructions. Each command will make the module to do a particular task. The commonly used commands and their function are given in the table below.

Command	Function
0F	LCD ON, Cursor ON, Cursor blinking ON
01	Clear screen
02	Return home
04	Decrement cursor
06	Increment cursor
08	Display OFF, Cursor OFF
OC	Display ON, Cursor OFF
0E	Display ON ,Cursor blinking OFF

Command	Function
80	Force cursor to the beginning of 1st line
C0	Force cursor to the beginning of 2nd line
38	Use 2 lines and 5×7 matrix
83	Cursor line 1 position 3
3C	Activate second line
C1	Jump to second line, position1
C2	Jump to second line, position2

LCD Initialization

The steps that has to be done for initializing the LCD display is given below and these steps are common for almost all applications.

1. Send 38H to the 8 bit data line for initialization
2. Send 0FH for making LCD ON, cursor ON and cursor blinking ON.
3. Send 06H for incrementing cursor position.
4. Send 01H for clearing the display and return the cursor.

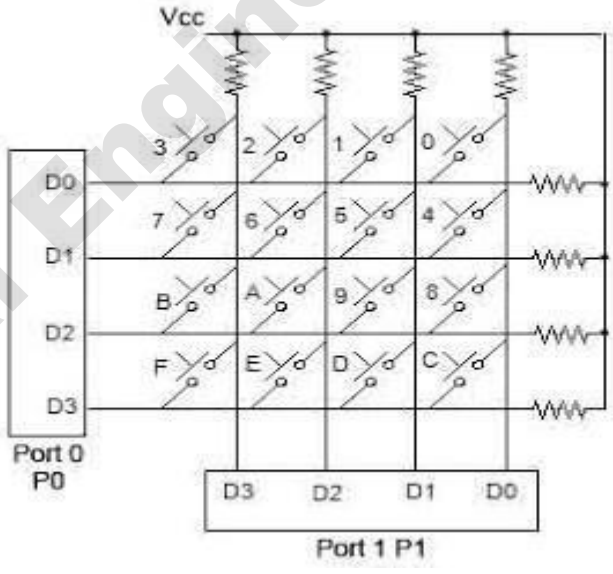
Sending data to the LCD

To send any of the commands to the LCD, make pin RS=0. For data, make RS=1. Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD. This is shown in the code below.

KEYBOARD INTERAFCING

Matrix keyboards are connected in a series of rows and columns. The important tasks in interfacing a keyboard are 1) detecting a key press, 2) debounce the key press and 3) encode the key to some standard code. Three tasks can be done with hardware, software, or a combination of two, depending on the application.

Keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and columns through ports. Therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor. When a key is pressed, a row and a column make a contact. Otherwise, there is no connection between rows and columns. A 4x4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port.



Scanning and Identifying the Key:

It is the function of the microprocessor to scan the keyboard continuously to detect and identify the key pressed

- To detect a pressed key, grounds all rows by providing 0 to the output latch, then it reads the columns

- If the data read from columns is $D_3 – D_0$ =1111, no key has been pressed and the process continues till key press is detected
- If one of the column bits has a zero, this means that a key press has occurred For example, if $D_3 – D_0$ = 1101, this means that a key in the D1 column has been pressed After detecting a key press, microprocessor will go through the process of identifying the key
- Starting with the top row, the microprocessor grounds it by providing a low to row D_0 only. It reads the columns, if the data read is all 1s, no key in that row is activated and the process is moved to the next row
- It grounds the next row, reads the columns, and checks for any zero. This process continues until the row is identified.
- After the key press detection, it waits 20ms for the **key debounce** and then scans the columns again
  (c) It ensures that the first key press detection was not an erroneous one due a spike noise
  (d) The key press. If after the 20-ms delay the key is still pressed, it goes back into the loop to detect a real key press
- Upon finding the zero, it pulls out the ASCII code for that key from the look-up table otherwise, it increments the pointer to point to the next element of the look-up table

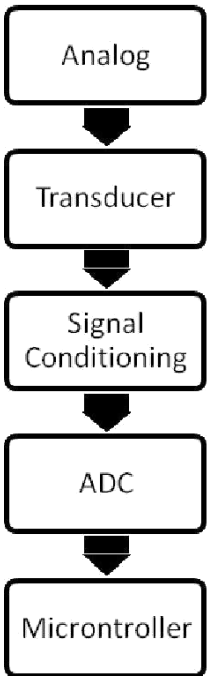
5. SENSOR INTERACING

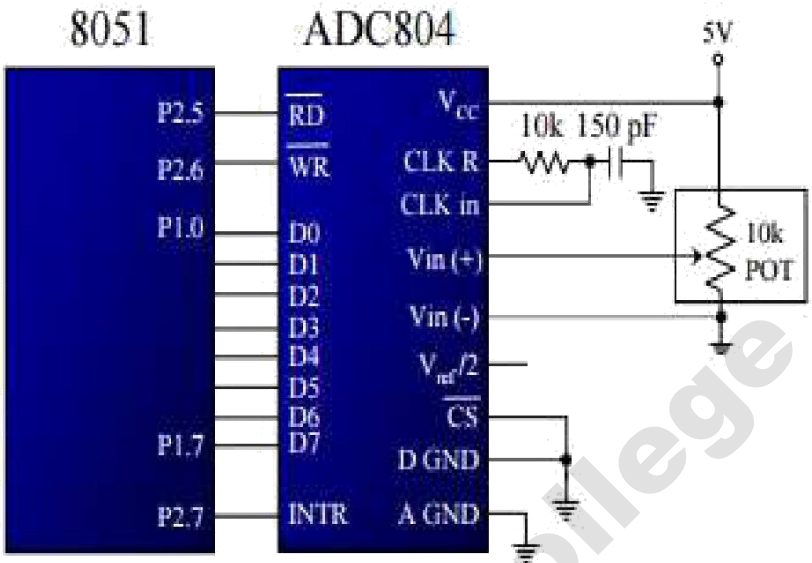
Interfacing a temperature sensor to 8051

Transducer converts physical data such as temperature, light intensity, flow and speed to electrical signals. Depending on the transducer the output produced is in the form of voltage, current, resistance or capacitance. For example temperature is converted to electrical signals using a transducer called a thermistor. A thermistor responds to temperature change by changing resistance, but its response is not linear. The complexity associated with writing software for such nonlinear devices has led many manufacturers to market the linear temperature sensor.

The sensors of the LM34/LM35 series are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Fahrenheit/Celsius temperature. The LM34/LM35 requires no external calibration since it is inherently calibrated. It outputs 10 mV for each degree of Fahrenheit/Celsius temperature

Signal conditioning is a widely used term in the world of data acquisition. It is the conversion of the signals (voltage, current, charge, capacitance, and resistance) produced by transducers to voltage, which is sent to the input of an A to-D converter.‰ Signal conditioning can be a current to voltage conversion or a signal amplification. The thermistor changes resistance with temperature, while the change of resistance must be translated into voltage in order to be of any use to an ADC





Look at the case of connecting an LM35 to an ADC804. Since the ADC804 has 8-bit resolution with a maximum of 256 steps and the LM35 (or LM34) produces 10 mV for every degree of temperature change, we can condition V_{in} of the ADC804 to produce a V_{out} of 2.56 V for full-scale output. Therefore, in order to produce the full scale V_{out} of 2.56 V for the ADC804, We need to set $V_{ref}/2 = 1.28$. This makes V_{out} of the ADC804 correspond directly to the temperature as monitored by the LM35.

Temp(C)	Vin(mV)	Vout (D7-D0)
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110

6. STEPPER MOTOR INTERFACING

Draw the diagram to interface a stepper motor with 8051 microcontroller and explain. Also write an 8051 ALP to run the stepper motor in both forward and reverse direction with delay. [16] [**April/May 2011]**

Draw the diagram to interface a stepper motor with a 8051 microcontroller and explain. Also write an 8051 ALP to run the stepper motor in both forward and reverse direction with delay. (10) [**Nov /Dec 2013]**

Draw the diagram to interface a stepper motor with 8051 microcontroller and explain. Write an 8051 assembly language program to run the stepper motor in both forward and reverse direction with delay. (16) [**Apr/May 2015, April/May2017**]

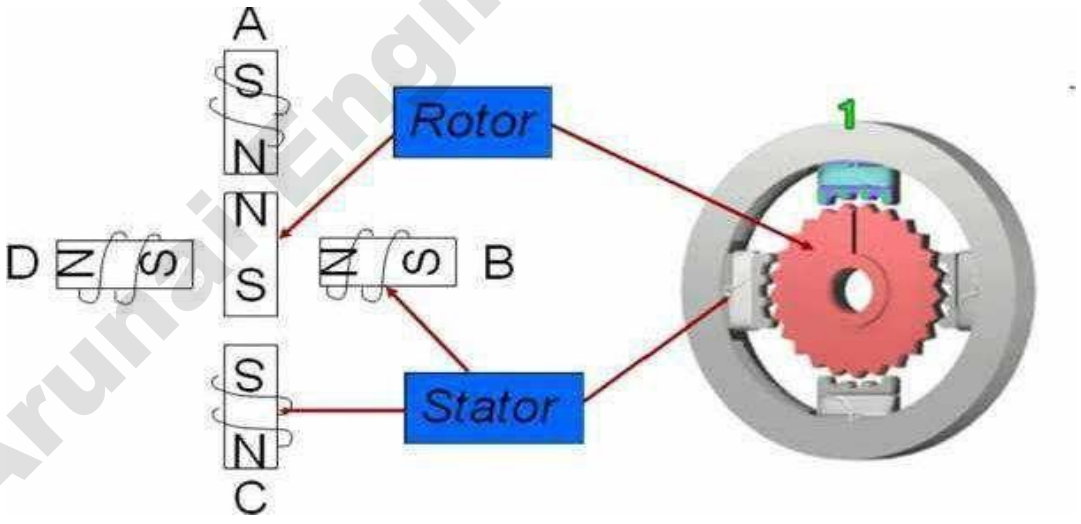
Stepper motor is a widely used device that translates electrical pulses into mechanical movement. Stepper motor is used in applications such as; disk drives, dot matrix printer, robotics etc,

It has a permanent magnet rotor called the shaft which is surrounded by a stator. Commonly used stepper motors have four stator windings that are paired with a centre tapped common. Such motors are called as four-phase or unipolar stepper motor. The stator is a magnet over which the electric coil is wound. One end of the coil are connected commonly either to ground or +5V. The other end is provided with a fixed sequence such that the motor rotates in a particular direction. Stepper motor shaft moves in a fixed repeatable increment, which allows one to move it to a precise position. Direction of the rotation is dictated by the stator poles. Stator poles are determined by the current sent through the wire coils.

Step angle:
Step angle is defined as the minimum degree of rotation with a single step.

No of steps per revolution = 360° / step angle
Steps per second = (rpm x steps per revolution) / 60

Example: step angle = 2°
No of steps per revolution = 180



Switching Sequence of Motor:

The coils need to be energized for the rotation. This can be done by sending a bits sequence to one end of the coil while the other end is commonly connected. The bit sequence sent can make either one phase ON or two phase ON for a full step sequence or it can be a combination of one and two phase ON for half step sequence. Both are tabulated below.

Full Step:

Two Phase ON

Clockwise

Step #	A	B	C	D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Counter-clockwise

One Phase ON

Clockwise

Step #	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Counter-clockwise

Half Step (8 – sequence):

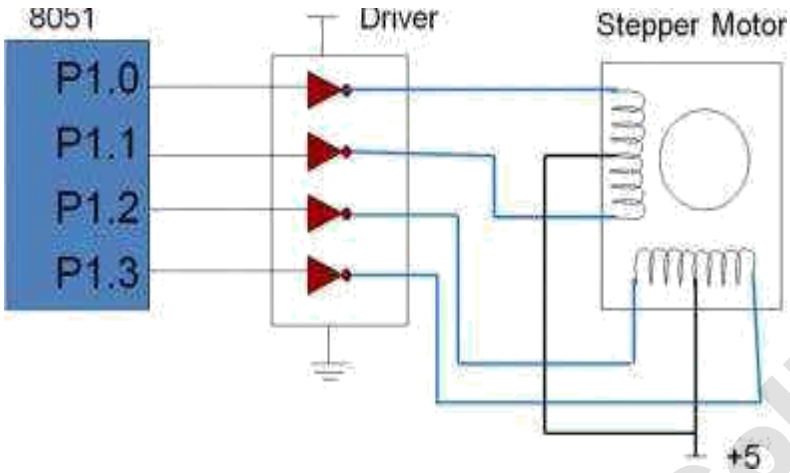
The sequence is tabulated as below:

Clk

Step #	A	B	C	D
1	1	0	0	1
2	1	0	0	0
3	1	1	0	0
4	0	1	0	0
5	0	1	1	0
6	0	0	1	0
7	0	0	1	1
8	0	0	0	1

Anti-Clk

8051 Connection to Stepper Motor: (explanation of the diagram can be done)



Interface to Stepper Motor

PROGRAM

To rotate the stepper motor clockwise / anticlockwise continuously with full step sequence.

```
        MOV A,#66H
BACK:   MOV P1,A
        RR A
        ACALL DELAY
        SJMP BACK

DELAY:  MOV R1,#100
UP1:    MOV R2,#50
UP:     DJNZ R2,UP
        DJNZ
        R1,UP1 RET
```

Note: motor to rotate in anticlockwise use instruction RL A instead of RR A

INDUSTRY CONNECTIVITY AND LATEST DEVELOPMENTS

- Microprocessors are used in the field of instrumentation and control
- Microprocessor based controllers are available in home appliances, such as microwave oven, washing machine and in controlling various parameters like speed, pressure, temperature etc.
- Microprocessors are being used in communication equipments.
- In telephone industry, these are used in digital telephone sets, Telephone exchanges and modem etc.
- The uses of microprocessor in television, satellite communication have made teleconferencing possible.
- Railway reservation and air reservation system also uses this technology.
- Microprocessor based micro computer with software packages are used in the office environment.
- Microprocessors based systems are being used for word processing, spread sheet operations, storage etc.
- The microprocessor has revolutionize the publication technology
- Microprocessors are used in:
    - Calculators
    - Accounting
    - SystemGames
    - Machine
    - Complex Industrial Controllers
    - Traffic light Control
    - Data acquisition systems
    - Multi user, multi-functionenvironments
    - Military applications
    - Communication systems

PREVIOUS YEAR
QUESTION PAPERS

Question Paper Code : 71737

B.E./B.Tech. DEGREE EXAMINATION, APRIL/MAY 2017.

Fourth/Fifth Semester

Electronics and Communication Engineering

EC 6504 – MICROPROCESSOR AND MICROCONTROLLER

(Common to Biomedical Engineering/Computer Science and Engineering/Medical Electronics Engineering/Information Technology)

(Regulations 2013)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. The offset address of a data is (341B)H and the data segment register value is (123A)H. What is the physical address of the data? Pa 6
2. Define stack register. Pa 8
3. What is meant by multiprogramming? Pa 57
4. Write some example for advanced processors. 80286,80386,80486
5. Draw the format of read back command register, of 8254. Pa 97
6. Write a 16 bit delay program in 8086. Pa 127
7. Which port used as multifunction port? List the signals. Pa 128
8. Illustrate the CJNE instruction. Pa 144
9. List the 8051 interrupts with its priority. Pa 147
10. What are the types of sensors used for interfacing? Pa 162

PART B — (5 × 13 = 65 marks)

11. (a) Draw and explain the architecture of 8086 with neat diagram. Pa 6

Or

(b) Describe the interrupts of 8086 and its types with service routine. Pa 49

12. (a) Explain the system bus structure of 8086. Draw the timing diagram for interrupt acknowledgement cycle. Pa 62,65

Or

(b) Explain the closely looped configuration with neat diagram. Pa 67

13. (a) Draw and explain the functional diagram of parallel communication interfacing chip. Pa 74

Or

(b) Explain the need of DMA controller with its functional diagram. Pa 118

14. (a) Write the available special function registers in 8051. Explain each register with its format and functions. Pa 136

Or

(b) (i) Discuss the types of addressing mode with suitable example in 8051. Pa 145 (8)

(ii) Write an 8051 assembly language program to multiply the given number 48H and 30H. (5)

LAB

15. (a) Write a program for generation of unipolar square waveform of 1 KHz frequency using Timer0 of 8051 in mode0. Consider the system frequency as 12MHz. Pa 149

Or

(b) Demonstrate the interfacing of the stepper motor with 8051 and explain its interfacing diagram and develop program to rotate the motor in clock wise direction. Pa 163

PART C — (1 × 15 = 15 marks)

16. (a) Develop a 8086 based system to display the word HELLO for every 2ms in the common cathode seven segment LED display and check how many times the word displayed for one hour. Pa 102

Or

(b) Develop 8051 based system design having 8Kbyte RAM to generate the triangular wave using DAC. DAC-pa 88

Reg. No. : | | | | | | | | | | | | | |

Question Paper Code : 80345

B.E./B.Tech. DEGREE EXAMINATION, NOVEMBER/DECEMBER 2016.

Fifth Semester

Electronics and Communication Engineering

EC 6504 — MICROPROCESSOR AND MICROCONTROLLER

(Common to Fifth Semester Biomedical Engineering and also common to Fourth Semester Information Technology and Medical Electronics/Computer Science and Engineering)

(Regulations 2013)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. List the flags of 8086 microprocessor. **Pa 6**
2. List the segment registers of 8086. **Pa 5**
3. Define machine cycle. **Pa 64**
4. Define Bus. **Pa 56**
5. How DMA is initiated? **Pa 118**
6. What is the drawback of memory mapped I/O? **Pa 73-q.no 3**
7. Draw the pin diagram of 8051. **Pa 140**
8. What is the significance of EA pin? **Pa 141**
9. List the modes of Timer in 8051. **Pa 150**
10. State how baud rate is calculated for serial data transfer in mode 1. **Pa 147**

PART B — (5 × 16 = 80 marks)

11. (a) (i) Explain the internal hardware architecture of 8086 microprocessor with neat diagrams. (12)

Pa 6

(ii) Write a short note about assembler directives. (4)

Pa 35

Or

(b) Explain the various addressing modes of 8086 microprocessor with suitable examples. (16)

Pa 10

12. (a) Discuss about the multiprocessor configurations of 8086. (16)

Pa 67

Or

(b) Explain in detail about the system bus timing of 8086/8088. Pa 65 16)

13. (a) Explain in detail about DMA controller. (16)

Pa 118

Or

Pa 88,93

(b) Explain the procedure of interfacing D/A and A/D converter circuit. (16)

14. (a) Explain in detail about the architecture of 8051 microcontroller with a neat diagram. (16)

Pa 130

Or

(b) Write an ALP using 8051 instructions to receive bytes of data serially and put them in P1. Set the baud rate at 4800, 8-bit data, and 1 stop bit. (16)

Pa 152

15. (a) Describe the different modes of operation of timers/counters in 8051 microcontroller. (16)

Pa 149

Or

(b) Draw a diagram to interface a stepper motor with 8051 microcontroller, also write an 8051 ALP to run the stepper motor in both forward and reverse direction with a delay. (16)

Pa 163

Question Paper Code: 77121

B.E./B.Tech. DEGREEE EXAMINATION, APRIL/MAY 2015

Fourth Semester

Computer Science and Engineering

EC6504 - MICROPROCESSORS AND MICROCONTROLLERS

(Regulation 2013)

Time: Three hours Maximum: 100marks

PART A- (10 x2=20 marks)

1. List the addressing modes of 8086. Give examples.
2. Write about the different types of interrupts supported in 8086.
3. Define bus. Why bus request and cycle stealing are required.
4. Draw the read cycle timing diagram for minimum mode.
5. Give the various modes and applications of 8254 timer?
6. Draw the block diagram of alarm controller with 8086 as processor
7. Draw the diagram for process status word in 8051.
8. How do you select the register bank in 8051 microcontroller?
9. Differentiate between timers and counters. Draw the diagram of TCON in 8051.
10. Which register is used for serial programming in 8051? Illustrate it.

PART B- (5 x16=80 marks)

11. (a)(i) Explain briefly about the internal hardware architecture of 8086 microprocessor with a neat diagram. (10)

(ii) Write an 8086 Assembly Language Program to Convert BCD data- Binary data. (6)

Or

(b) (i) Explain about the Assume, EQU, DD assembler directives. (8)

(ii)Explain briefly about Interrupt handling process in 8086. (8)

12. (a) Discuss the maximum mode configuration of 8086 by with a neat diagram. Mention the functions of the various signals. (16)

Or

(b) (i) Compare closely coupled configuration with loosely coupled configuration.(8)

ii) Write a 8086 Assembly Language program to check whether the input string is palindrome or not. (8)

13. (a) (i) Explain how D/A and A/D interfacing done with 8086 with an application. (10)

ii) What is DMA? Explain the DMA based data transfer using DMA controller. (6)

Or

b) (i) Draw the block diagram of traffic light control system using 8086. (8)

ii) Write the algorithm and assembly language program for traffic light control system (8)

14. (a)(i) Explain the architecture of 8051 microcontroller with neat diagram. (8)

(ii) Explain the TMOD function register and its timer modes of operations. (8)

Or

(b)(i) Explain about Arithmetic and control instruction set in 8051. (10)

(ii) Write a program to bring in data in serial form and send it out in parallel form using 8051. (6)

15.(a)(i) Describe the different modes of operation of timers/counters in 8051 with its associated register. (10)

(ii) How does one interface a 16 x 2 LCD display using 8051 Microcontroller?(6)

Or

b) Draw the diagram to interface a stepper motor with 8051 microcontroller and explain. Write a 8051 assembly language program to run the stepper motor in both forward and reverse direction with delay. (16)

Question Paper Code: 91344

B.E./B.Tech. DEGREEE EXAMINATION, November/December 2014

Fourth Semester

Computer Science and Engineering

CS2252 - MICROPROCESSORS AND MICROCONTROLLERS

(Regulation 2013)

Time: Three hours Maximum: 100marks

PART A- (10 x2=20 Marks)

1. What is the processing element inside the microprocessor? What process it does?
2. What is the need for ALE pin in 8085 microprocessor?
3. How many memory locations can be addressed by 8086 microprocessor?
4. If the stack segment register contains 3000h and the stack pointer register contains 8434h, what is the physical address of the top of the stack?
5. How does the main processor distinguish its instructions from the co-processor instructions when it fetches the instructions from memory?
6. Compare Closely Coupled configuration with Loosely Coupled Configuration.
7. List any four applications of stepper motor
8. How memory interfacing is differentiated from I/O interfacing?
9. Compare the features of microprocessor with microcontroller.
10. What is the necessity to interface DAC with microcontroller?

PART B- (5 x16=80 Marks)

11. (a) (i) Write an 8085 assembly language program to find the largest number in the given array (8)

(ii) Explain the different addressing modes of 8086 microprocessor with examples. (8)

Or

(b) (i) Write an 8085 assembly language program to divide two 8-bit numbers and store the corresponding quotient and remainder in the suitable memory locations. . (8)

Draw the functional block diagram of 8085 microprocessor and explain the functions of each block (8)

12 (a) Describe the sequence of signals that occurs on the address bus, the control bus and the data bus when a simple microcomputer fetches an instruction. (16)

Or

(b) (i) Write an 8086 assembly language program to multiply two 16-bit numbers to give 32-bit result. (8)

(ii) Describe the conditions which cause the 8086 to perform type 0 and type 1 interrupt.

(8)

13 (a) (i) Draw the control word and status word format of 8087 processor (10)

(ii) Explain how the communication between CPU and IOP processor takes place. (6)

Or

(b) (i) Draw the architecture of 8089 I/O processor and explain it. (8)

(ii) Explain the different data formats of 8087 coprocessor. (8)

14. (a) (i) In how many modes we can use 8253/54 timer? Explain the different modes of operation of 8253/54 timer. (8)

(ii) How to interface a DMA controller with a microprocessor? Explain how DMA controller transfers large amount of data from one memory locations to another memory location? (8)

Or

(b) (i) Draw the block diagram of a keyboard display controller and explain. (8)

(ii) Explain in detail about the parallel communication interface. (8)

15 (a) (i) How to interface an LCD display with microcontroller? Explain how to display a character using LCD display. (8)

(ii) Draw the data memory structure of 8051 microcontroller and Explain. Or

(b) (i) Draw the functional block diagram of 8051 microcontroller and explain each block. (8)

(ii) Explain the interrupt structure of 8051 microcontroller with suitable diagram. (8)

Question Paper Code: 51342

B.E./B.Tech. DEGREEE EXAMINATION, May/June 2014

Fourth Semester

Computer Science and Engineering

CS2252 - MICROPROCESSORS AND MICROCONTROLLERS

(Regulation 2008)

Time: Three hours Maximum: 100marks

PART A- (10 x2=20 Marks)

1. Explain the instruction PCHL of 8085 Microprocessor
2. What 8085 program to sap the content stored in two different memory addresses?
3. What do you mean by addressing modes
4. What is meant by vectored interrupts?
5. What are the advantages of coprocessor?
6. What is meant by a loosely coupled configuration?
7. What are the advantages of Programmable Interval Timer/Counter IC?
8. List the features of memory mapped I/O.
9. What are the differences between a microprocessor and microcontroller?
10. What is the significance of EA line of 8051 microcontroller?

PART B- (5 x16=80 Marks)

11 (a) Explain the internal architecture of Intel 8085 Microprocessor. (16)

Or

(b) (i) Write an 8085 assembly language program to convert a single digit BCD number into a binary number. (8)

(ii) Write an 8085 Assembly language program to add two 16-bit BCD Numbers. (8)

12. (a) Draw and discuss the interrupt structure of 8086 (16)

Or

(b) (i) Write an 8086 Assembly language program to get an input from the keyboard for 2 digits and convert that input into a hexa decimal number using BIOS interrupt. (8)

(ii) Write an 8086 Assembly language program to multiply 2 digits numbers by getting an input from the keyboard wing BIOS interrupt call. (8)

13 (a) (i) Explain the execution steps of 8087 coprocessor (10)

(ii)Explain the architecture of 8089 I/O processor (6)

Or

(b) Explain the closely coupled configuration of multiprocessor configuration with suitable diagram. (16)

14 (a) (i) Explain the mode 0 operation of 8255 programmable Peripheral Interface (8)

(ii) Explain the different modes of operation of a timer (8)

Or

(b) Explain the internal architecture of 8257 Direct Memory Access Controller. (16)

15. (a)Draw the pin diagram of 8051 Microcontroller. And explain the Input/Output lines in detail. (16)

Or

(b) (i) Vin=2.25v. Vref=5v Number of data lines are 5. Convert the given analog quantity into its equivalent output digital quantity. (8)

(ii) Explain the different techniques to convert a digital quantity into its equivalent analog quantity. (8)

Question Paper Code: 21302

B.E./B.Tech. DEGREEE EXAMINATION, MAY/JUNE 2013

Fourth Semester

Computer Science and Engineering

CS2252 - MICROPROCESSORS AND MICROCONTROLLERS

(Regulation 2013)

PART A- (10 x2=20 marks)

1. What is the effect of executing the instruction DAD B and ADD M?
2. Draw the contents of the flag register of 8085.
3. Name the hardware interrupts of 8086.
4. What is the function of LOCK and RQ/GT signals?
5. How does CPU differentiate the 8087 instructions from its own instructions?
6. How 8089 operates in loosely coupled configuration and tightly coupled configuration?
7. What are the requirements to be met while interfacing memory on I/O devices to 8085 processor?
8. What are the modes of operation of 8237?
9. What is Baud rate for mode operation of the serial port of 8051?
10. In the program status word of 8051, the bits RS) and RS! Are 1 and 0, then which register bank is selected for operation?

PART B- (5 x16=80 marks)

11. (a) (i) Write a program to find the average of ten numbers (8)
    ii).Describe the addressing modes of 8085(8)

    or

    (b) (i)Discuss the functional block diagram of 8085(12)
    (ii)Write a program to divide two eight bit numbers.(4)

12. (a) (i)Explain about the following assembler directives: END P, EQU, EVEN,EXTRN with examples (8)
    (ii) Draw and Discuss a typical minimum mode 8086 system. (8)

    or

    (b) (i)Describe the maximum mode of operation of 8086.(8)
    (ii)What are the assembler directives and pseudo ops? (8)
13. (a) Discuss the operation of 8087 numeric data processor.(16)

    or

    (b) Describe the architecture of 8089 (16)
14. (a) Explain the (i) modes of operation of timer and (ii) operation of interruptcontroller (16).

    or

    (b) Discuss briefly about Keyboard/display controller. (16)
15. (a)(i) Describe the functions of the signals present in 8051?(10)
    (ii)How a DAC is interfaced with 8051? (6)

    or

    (b)i) Explain how an LCD and keyboard is interfaced with 8051.(12)
    ii) Describe about serial port interface of 8051. (4)