SKP Engineering College

Tiruvannamalai – 606611

A Course Material

on

Microprocessor and Microcontroller

By

M.Dhinakaran

Assistant Professor

Department of Electronic and Communication Engineering

Quality Certificate

This is to Certify that the Electronic Study Material

Subject Code: EC6504

Subject Name: Microprocessor and Microcontroller

Year/Sem:III/V

Being prepared by me and it meets the knowledge requirement of the University curriculum.

Signature of the Author

Name: M.Dhinakaran

Designation: Assistant Professor

This is to certify that the course material being prepared by Mr. M.Dhinakaran is of the adequate quality. He has referred more than five books and one among them is from abroad author.

Signature of HD

Name:Mr.R.Saravanakumar

Seal:

Signature of the Principal

Name: Dr.V.Subramania Bharathi

Seal:

EC6504 MICROPROCESSOR AND MICROCONTROLLER L T P C 3 0 0 3

OBJECTIVES: The student should be made to:

☐ Study the Architecture of 8086 microprocessor.
☐ Learn the design aspects of I/O and Memory Interfacing circuits.
☐ Study about communication and bus interfacing.
☐ Study the Architecture of 8051 microcontroller.

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 – IO 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.

TOTAL: 45 PERIODS

OUTCOMES: At the end of the course, the student should be able to:

☐ Design and implement programs on 8086 microprocessor.
☐ Design I/O circuits.
☐ Design Memory Interfacing 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.
2. Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay, “The 8051 Microcontroller and Embedded Systems: Using Assembly and C”, Second Edition, Pearson education, 2011.

REFERENCE:

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

CONTENTS

Unit -I

The 8086 Microprocessor

Part -A

1. **Why is the 8086 memory divided into odd and even banks?[CO1-L3-Nov/Dec 2015]**

In 8086, the 1 Mbytes memory is physically organized as an odd bank and an even bank, each of 512 Kbytes, addressed in parallel by the processor. Byte data with an even address is transferred on D_7-D_0, while the byte data with an odd address is transferred to D_{15} – D_8 bus lines.

2. **What do you mean by Segment Override Prefix? [CO1-L2]**

A segment override prefix allows any segment register (DS, ES, SS, or CS) to be used as the segment when evaluating addresses in an instruction. An override is made by adding the segment register plus a colon to the beginning of the memory reference of the instruction as in the following examples:

```
    mov     ax, [es:60126]        ; Use es as the segment
    mov     ax, [cs:bx]           ; Use cs as the segment
    mov     ax, [ss:bp+si+3]      ; Use ss as the segment
```

3. **What are the 8086 instructions used for ASCII arithmetic? [CO1-L3-Nov/Dec 2012]**

AAA: ASCII adjust after addition
AAS: ASCII adjust after subtraction
AAM: ASCII adjust after multiplication
AAD : ASCII adjust before division

4. **List the various string instructions available in 8086. [CO1-L1-May/Jun 2014]**

MOVS , CMPS, STOS, LODS and SACS. SCAS

5. **State the different data transfer schemes. [CO1-L1]**

The data transfer schemes have been broadly classified into following two categories:
(i) Programmed data transfer
(ii) Direct Memory Access (DMA) data transfer

Programmed data transfer	DMA data transfer
Synchronous data transfer Asynchronous data transfer Interrupt driven data transfer	Cycle stealing DMA Burst mode DMA Demand transfer mode DMA

6. **What are the advantages of memory mapped I/O over I/O mapped I/O. [CO1-L1]**

The advantage of memory mapped I/O over I/O mapped I/O is that all the instruction which are used for memory are also used for controlling I/O ports. It helps in performing arithmetic and logic operation on I/O data directly.

7. Identify the addressing modes in the following instructions : [CO1-L1-May/Jun 2011]

AND AL, BL	- Register Addressing Mode
SUB AL, 24H	- Immediate Addressing Mode
MOV AL, (BP)	- Register Indirect Addressing Mode
MOV CX, 1245H	- Immediate Addressing Mode

8. What is an assembler? [CO1-L2-Nov/Dec 2014]

An assembler is a program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations.

9. What is virtual addressing mode? [CO1-L2]

Virtual memory allows a large program to execute in a smaller physical memory. (Virtual memory is a feature of an operating system (OS) that allows a computer to compensate for shortages of physical memory by temporarily transferring pages of data from random access memory (RAM) to disk storage.)

10. List the addressing modes of 8086.Give examples.[CO1-L1-Nov/Dec 2013]

Register addressing mode
Immediate addressing mode
Direct memory addressing mode
Register indirect addressing mode

11. Write about the different types of interrupts supported in 8086. [CO1-L1-May/Jun 2016]

Predefined interrupt
Non-maskable interrupt
Maskable interrupt

12. What are the various Pointer and Index registers available in 8086? [CO1-L1-May/Jun 2015]

Stack pointer (SP), Base pointer (BP)
Source index (SI), Destination index (DI)

13. What are the various sources of interrupt in 8086? [CO1-L1]

The 8086 has four sources of interrupts :
Software or within program logic
Single step condition
External logic as a non-maskable interrupt
External logic as a maskable interrupt

14. State the difference between PUSH and POP instruction? [CO1-L2-Nov/Dec 2013]

PUSH instruction is used for putting something on the stack and the POP instruction is used for taking off something from the stack.Only words can be pushed or popped and the data cannot be immediate, but the PUSH and POP can use all other addressing modes.

15. Write the assembler directives of 8086. [CO1-L1-May/Jun 2014]
Directives for constant and variable definition (DUP,EQU)
Program location control directives (ORG,EVEN,ALIGN,OFFSET)
Segment declaration directives (SEGMENTS , ENDS,ASSUME)
Procedure and Macro related directives (PROC,ENDP,MACRO,ENDM)

16. Give two examples for Control Transfer instructions. [CO1-L1-May/Jun 2011]
JMP LABEL –Unconditional Jump
JMP LABEL (2 or 3 bytes) – Direct addressing

17. List the various groups of instructions in 8086.[CO1-L1]
1. Data transfer instructions
2. Arithmetic instructions
3. Bit manipulation or Logical instructions
4. String instructions
5. Control transfer instructions

18. State the importance of Macros. [CO1-L3]
A macro is a group of instructions that perform one task, just as a procedure performs a task. A macro is inserted in the program at the point of usage as a new sequence of instructions. Macro sequences execute faster than procedures.

19. Compare Macros and Procedure. [CO1-L2-May/Jun 2014]
A macro is a group of instructions that perform one task, just as a procedure performs a task. The difference is that a procedure is accessed via a CALL instruction, while a macro is inserted in the program at the point of usage as a new sequence of instructions. Macro sequences execute faster than procedures because there is no CALL and RET instructions to execute.

20. What is Interrupt Service Routine (ISR)?[CO1-L3-Nov/Dec 2015]
Interrupt means event, which invites attention of the processor on occurrence of some action at hardware or software interrupt instruction event.In response to the interrupt, the routine or program, which is running presently interrupts and an interrupt service routine(ISR) executes.

21. Name the hardware interrupts of 8086. [CO1-L1-May/Jun 2013]
The hardware interrupts of 8086 are INTR and NMI. The INTR is a general maskable interrupt and NMI is non-maskable interrupt.

22. What is modular programming? [CO1-L3]

The formulation of complex programs from numerous small sequences, called program modules each of which performs a well-defined task. Such formulation of computer code is referred to as modular programming.

23. What is a procedure?[CO1-L3-Nov/Dec 2014]

A procedure (or subroutine) is a set of code that can be branched to and returned from in such a way that the code is as if it were inserted at the point from which it is branched to. The branch to a procedure is referred to as the *call* and the corresponding branch back is known as the *return.*

24. What are the advantages of modular programming?[CO1-L1-May/Jun 2012]

Modules are easier to comprehend
Different modules can be assigned to different programmers.
Modifications may be localized
Documentation can be more easily understood.

Part -B

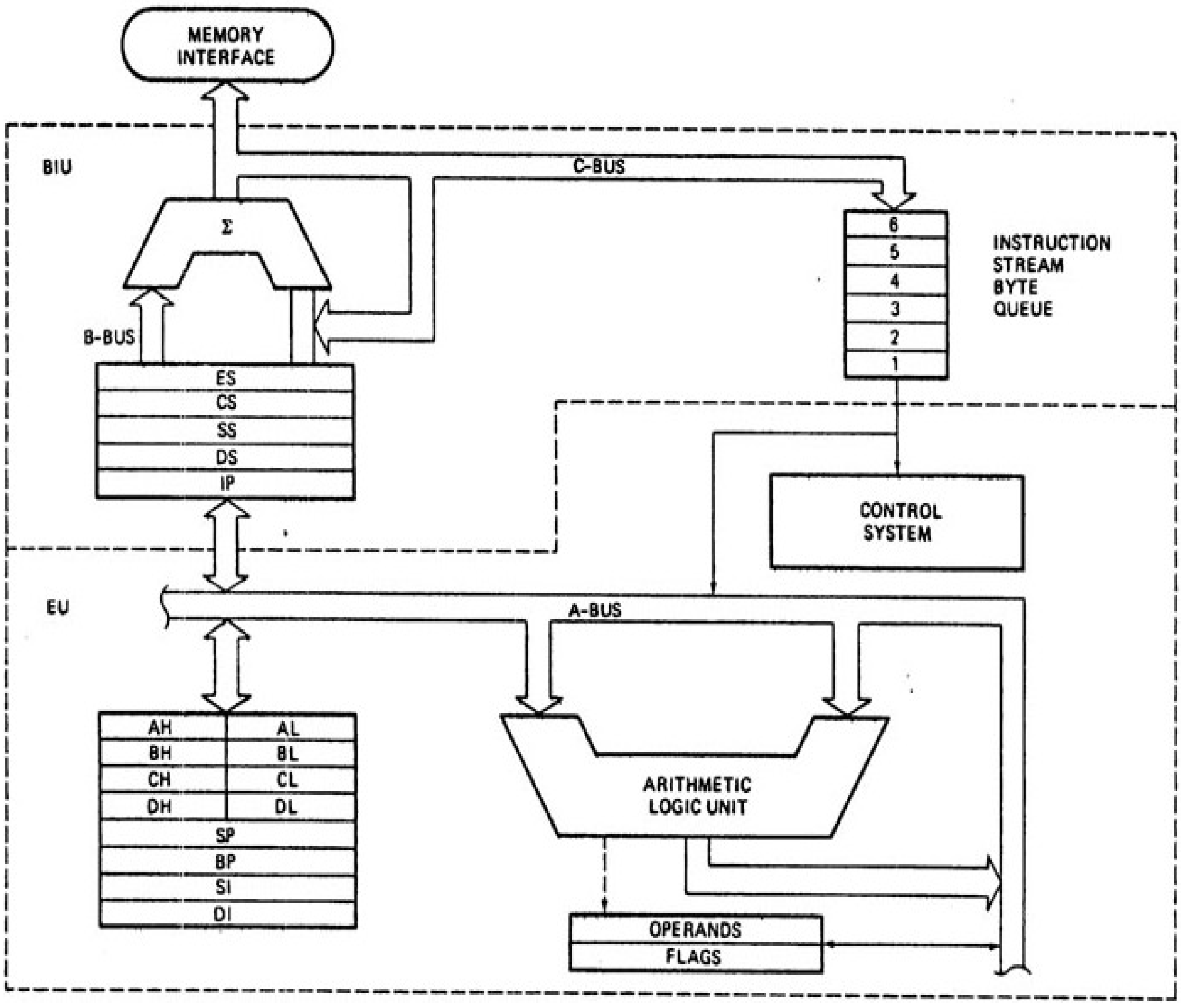
1. Explain the architecture of 8086 Microprocessor in detail? [CO1-L2-Nov/Dec 2016]

Microprocessor Architecture**:**

The 8086 CPU is divided into two independent functional parts, the Bus interface unit (BIU) and execution unit (EU).

The Bus Interface Unit contains Bus Interface Logic, Segment registers, Memory addressing logic and a Six byte instruction object code queue. The BIU sends out address, fetches the instructions from memory, read data from ports and memory, and writes the data to ports and memory.

BH = [00521H]



The execution unit: contains the Data and Address registers, the Arithmetic and Logic Unit, the Control Unit and flags. tells the BIU where to fetch instructions or data from, decodes instructions and executes instruction. The EU contains control circuitry which directs internal operations. A decoder in the EU translates instructions fetched from memory into a series of actions which the EU carries out. The EU is has a 16-bit ALU which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers. The EU is decoding an instruction or executing an instruction which does not require use of the buses. In other words the BIU handles all transfers of data and addresses on the buses for the execution unit.

The Queue: The BIU fetches up to 6 instruction bytes for the following instructions. The BIU stores these prefetched bytes in first-in-first-out register set called a queue. When the EU is ready for its next instruction it simply reads the instruction byte(s) for the instruction from the queue in the BIU. This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction byte or bytes. Except in the case of JMP and CALL instructions, where the queue must dumped and then reloaded starting from a new address, this prefetch-and-queue greatly speeds up processing. Fetching the next instruction while the current instruction executes is called pipelining.

Word Read: Each of 1 MB memory address of 8086 represents a byte wide location.16-bit words will be stored in two consecutive memory locations. If first byte of the data is stored at an even address**,** 8086 can read the entire word in one operation.

For example if the 16 bit data is stored at even address 00520H is 9634H

MOV BX, [00520H]

8086 reads the first byte and stores the data in BL and reads the 2nd byte and stores the data in BH

BL= (00520H) i.e. BL=34H

BH= (00521H) BH=96H

If the first byte of the data is stored at an odd address, 8086 needs two operations to read the 16 bit data.

For example if the 16 bit data is stored at even address 00521H is 3897H

MOV BX, [00521H]

In first operation, 8086 reads the 16 bit data from the 00520H location and stores the data of 00521H location in register BL and discards the data of 00520H location In 2nd operation, 8086 reads the 16 bit data from the 00522H location and stores the data of 00522H location in register BH and discards the data of 00523H location.

BL= (00521H) i.e. BL=97H

BH= (00522H) BH=38H

Byte Read:

MOV BH, [Addr]

For Even Address:

Ex: MOV BH, [00520H]

8086 reads the first byte from 00520 location and stores the data in BH and reads the

2ndbyte from the 00521H location and ignores it

BH =[00520H]

For Odd Address
MOV BH, [Addr]
Ex: MOV BH, [00521H]
8086 reads the first byte from 00520H location and ignores it and reads the 2nd byte
from the 00521 location and stores the data in BH

Physical address formation:

The 8086 addresses a segmented memory. The complete physical address which is 20-bits long is generated using segment and offset registers each of the size 16-bit.Thecontent of a segment register also called as segment address, and content of an offset register also called as offset address. To get total physical address, put the lower nibble 0H to segment address and add offset address. The fig 1.3 shows formation of 20-bit physical address.

Register organization of 8086:

All the registers of 8086 are 16-bit registers. The general purpose registers, can be used either 8-bit registers or 16-bit registers used for holding the data, variables and intermediate results temporarily or for other purpose like counter or for storing offset address for some particular addressing modes etc. The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.

AX Register: Accumulator register consists of two 8-bit registers AL and AH, Which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation

BX Register: This register is mainly used as a **base register**. It holds the starting Base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.

CX Register: It is used as default counter **- count register** in case of string and Loop instructions.

DX Register: Data register can be used as a port number in I/O And implicit operand or destination in case of few instructions. In integer 32- bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Segment registers:

1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation is as shown in fig 1.4. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment(CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly.

The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment(SS) is a 16-bit register containing address of 64KB segment With program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.

Data segment(DS) is a 16-bit register containing address of 64KB segmentWith program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment.DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided. Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. Pointers and index registers**.**The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BPregister 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 addresses 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.

Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of he result, and to determine conditions to transfer control to other parts of the program. The 8086 flag register as shown in the fig 1.5. 8086 has 9 active flags and they are divided into two categories:

1. Conditional Flags

2. Control Flags

Conditional Flags

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D_3) to upper nibble (i.e. D4 – D_7), the AC flag is set i.e. carry given by D3 bit to D4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.
Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.
Control Flags Control flags are set or reset deliberately to control the operations of the execution unit.
Control flags are as follows:
Trap Flag (TF): It is used for single step control. It allows user to execute oneinstruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.
Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.
Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the stringbytes are accessed from lower memory address to higher memory address.

2. What is meant by addressing mode? Explain the different addressing modes available in 8086 processor? [CO1-L2-May/Jun 2016]

Addressing Modes
The 8086 has 12 addressing modes can be classified into five groups-Addressing modes for accessing immediate and register data (register and immediate modes).
Addressing modes for accessing data in memory (memory modes)
Addressing modes for accessing I/O ports (I/O modes)
Relative addressing mode
Implied addressing mode
Immediate addressing mode:
In this mode, 8 or 16 bit data can be specified as part of the instruction - OP Code Immediate Operand
Example 1: MOV CL, 03 H:Moves the 8 bit data 03 H into CL
Example 2: MOV DX, 0525 H: Moves the 16 bit data 0525 H into DX
In the above two examples, the source operand is in immediate mode and the destination operand is in register mode.
A constant such as "VALUE" can be defined by the assembler EQUATE directive such as VALUE EQU 35H
Example: MOV BH, VALUE Used to load 35 H into BH
Register addressing mode:
The operand to be accessed is specified as residing in an internal register of 8086. internal registers, anyone can be used as a source or destination operand, however only the data registers can be accessed as either a byte or word.

Example 1: MOV DX (Destination Register) , CX (Source Register)
Which moves 16 bit content of CS into DX.
Example 2: MOV CL, DL
Moves 8 bit contents of DL into CL
MOV BX, CH is an illegal instruction.
The register sizes must be the same.

Direct addressing mode:

The instruction Opcode is followed by an affective 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. The default segment is always DS.The 20 bit physical address of the operand in memory is normally obtained as PA =DS: EA But by using a segment override prefix (SOP) in the instruction, any of the four segment registers can be referenced,The Execution Unit (EU) has direct access to all registers and data for register and immediate operands. However the EU cannot directly access the memory operands. It must use the BIU, in order to access memory operands.In the direct addressing mode, the 16 bit effective address (EA) is taken directly from the displacement field of the instruction.

Example 1: MOV CX, START

If the 16 bit value assigned to the offset START by the programmer using an assembler pseudo instruction such as DW is 0040 and [DS] = 3050. Then BIU generates the 20 bit physical address 30540 H.

The content of 30540 is moved to CL

The content of 30541 is moved to CH

Example 2: MOV CH, START

If [DS] = 3050 and START = 0040

8 bit content of memory location 30540 is moved to CH.

Example 3: MOV START, BX

With [DS] = 3050, the value of START is 0040.

Physical address: 30540

MOV instruction moves (BL) and (BH) to locations 30540 and 30541 respectively.

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 EA.

Example: MOV [DI], BX register indirect

If [DS] = 5004, [DI] = 0020, [Bx] = 2456 PA=50060.

The content of BX(2456) is moved to memory locations 50060 H and 50061 H.

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: MOV S BYTE

If [DF] = 0, [DS] = 2000 H, [SI] = 0500,

[ES] = 4000, [DI] = 0300

Source address: 20500, assume it contains 38

PA: [DS] + [SI]

Destination address: [ES] + [DI] = 40300, assume it contains 45

I/O mode (direct):

Port number is an 8 bit immediate operand.
Example: OUT 05 H, AL
Outputs [AL] to 8 bit port 05 H

I/O mode (indirect):

The port number is taken from DX.
Example 1: IN AL, DX
If [DX] = 5040
8 bit content by port 5040 is moved into AL.
Example 2: IN AX, DX
Inputs 8 bit content of ports 5040 and 5041 into AL and AH respectively.

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 have no operands.Example: CLC which clears carry flag to zero.

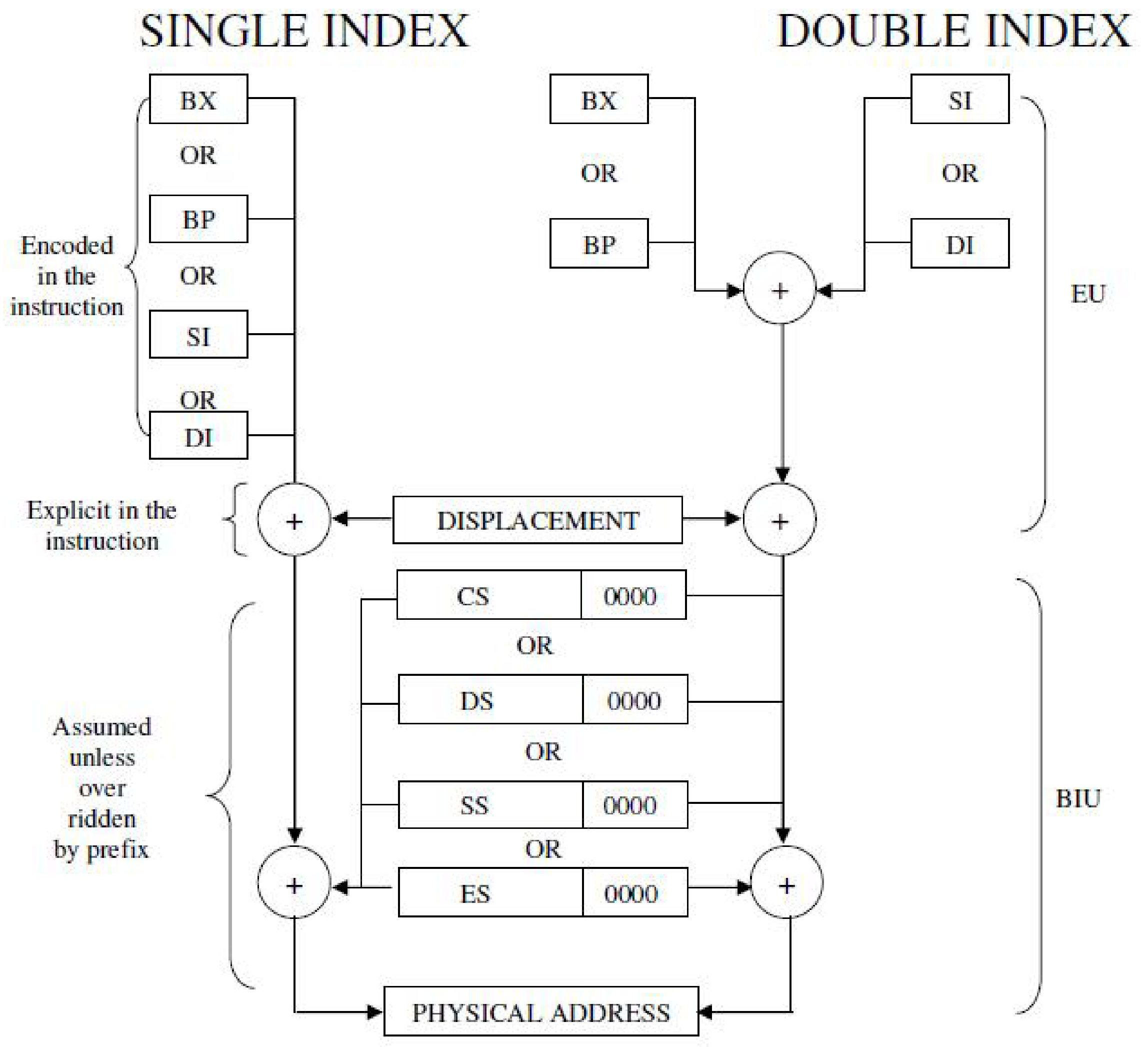


Fig.3.1 : Summary of 8086 Addressing Modes

3. Explain in detail about instruction set of 8086. [CO1-L2-May/Jun 2015]

INSTRUCTION SET OF 8086

The 8086 instructions are categorized into the following main types.

1. Data Copy / Transfer Instructions
2. Arithmetic and Logical Instructions
3. Shift and Rotate Instructions
4. Loop Instructions
5. Branch Instructions
6. String Instructions
7. Flag Manipulation Instructions
8. Machine Control Instructions

Data Copy / Transfer Instructions:

MOV:

This instruction copies a word or a byte of data from some source to a 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 AX, BX
MOV AX, 5000H
MOV AX, [SI]
MOV AX, [2000H]
MOV AX, 50H[BX]
MOV [734AH], BX
MOV DS, CX
MOV CL, [357AH]
Direct loading of the segment registers with immediate data is not permitted.

PUSH: Push to Stack

This instruction pushes the contents of the specified register/memory location on to thestack. The stack pointer is decremented by 2, after each execution of the instruction.

E.g. PUSH AX
• PUSH DS
• PUSH [5000H]

POP: Pop from Stack

This instruction when executed, loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer.
The stack pointer is incremented by 2
Eg. POP AX
POP DS
POP [5000H]

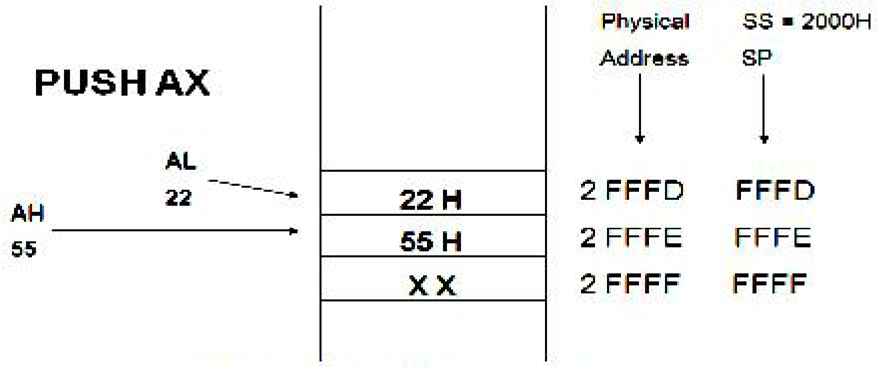


Fig. 2.2 Push Data to stack memory

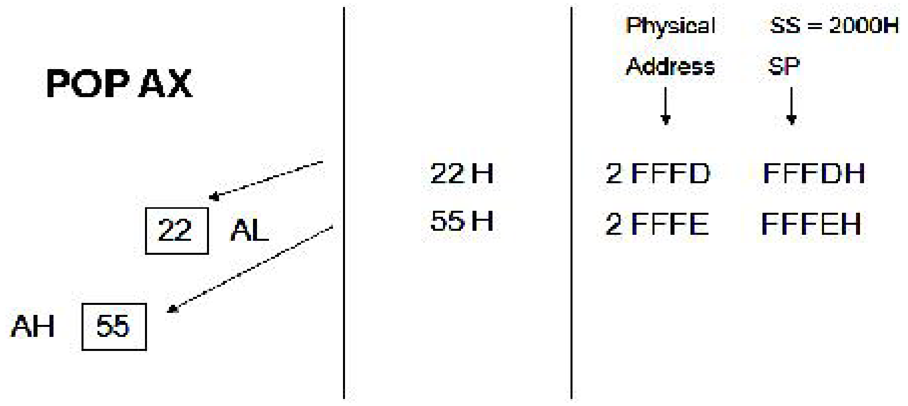


Fig 1.8 Push into and Popping Register Content from Stack Memory

XCHG: Exchange byte or word
This instruction exchange the contents of the specified source and destination operands
Eg. XCHG [5000H], AX

IN:
Copy a byte or word from specified port to accumulator.
Eg. IN AL,03H
IN AX,DX

OUT:
Copy a byte or word from accumulator specified port.
Eg. OUT 03H, AL
OUT DX, AX

LEA:
Load effective address of operand in specified register.
[reg] offset portion of address in DS
Eg. LEA reg, offset

LDS:
Load DS register and other specified register from memory.
[reg] [mem]
[DS] [mem + 2]
Eg. LDS reg, mem

LES:
Load ES register and other specified register from memory.
[reg] [mem]
[ES] [mem + 2]
Eg. LES reg, mem

Flag transfer instructions:

LAHF:
Load (copy to) AH with the low byte the flag register.
[AH] [Flags low byte]
Eg. LAHF

Flag transfer instructions:

LAHF:
Load (copy to) AH with the low byte the flag register.
[AH] [Flags low byte]
Eg. LAHF

SAHF:
Store (copy) AH register to low byte of flag register.
[Flags low byte] [AH]
Eg. SAHF

PUSHF:
Copy flag register to top of stack.
[SP] [SP] – 2
[[SP]] [Flags]
Eg. PUSHF

POPF:
Copy word at top of stack to flag register.
[Flags] [[SP]]
[SP] [SP] + 2

Arithmetic Instructions:

The 8086 provides many arithmetic operations: addition, subtraction, negation, multiplication and comparing two values.

ADD:

The add instruction adds the contents of the source operand to the destination operand.
Eg. ADD AX, 0100H
ADD AX, BX
ADD AX, [SI]
ADD AX, [5000H]
ADD [5000H], 0100H
ADD 0100H

ADC: Add with Carry

This instruction performs the same operation as ADD instruction, but adds the carry flag to the result.
Eg. ADC 0100H
ADC AX, BX
ADC AX, [SI]
ADC AX, [5000]
ADC [5000], 0100H

SUB: Subtract

The subtract instruction subtracts the source operand from the destination operand and the result is left in the destination operand.
Eg. SUB AX, 0100H
SUB AX, BX
SUB AX, [5000H]
SUB [5000H], 0100H

SBB: Subtract with Borrow
The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand
Eg. SBB AX, 0100H
SBB AX, BX
SBB AX, [5000H]
SBB [5000H], 0100H

INC: Increment

This instruction increases the contents of the specified Register or memory location by 1. Immediate data cannot be operand of this instruction.
Eg. INC AX
INC [BX]
INC [5000H]

DEC: Decrement
The decrement instruction subtracts 1 from the contents of the specified register or memory location.
Eg. DEC AX
DEC [5000H]

NEG: Negate

The negate instruction forms 2’s complement of the specified destination in the Instruction. The destination can be a register or a memory location. This instruction can be implemented by inverting each bit and adding 1 to it.
Eg. NEG AL
AL = 0011 0101 35H Replace number in AL with its 2’s complement
AL = 1100 1011 = CBH

CMP: Compare
This instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location
Eg. CMP BX, 0100H
CMP AX, 0100H
CMP [5000H], 0100H
CMP BX, [SI]
CMP BX, CX

MUL:Unsigned Multiplication Byte or Word
This instruction multiplies an unsigned byte or word by the contents of AL.
Eg. MUL BH; (AX) (AL) x (BH)
MUL CX; (DX)(AX) (AX) x (CX)
MUL WORD PTR [SI]; (DX)(AX) (AX) x ([SI])

IMUL:Signed Multiplication
This instruction multiplies a signed byte in source operand by a signed byte in AL or a signed word in source operand by a signed word in AX.
Eg. IMUL BH
IMUL CX
IMUL [SI]

CBW: Convert Signed Byte to Word
This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be sign extension of AL.
Eg. CBW
AX= 0000 0000 1001 1000 Convert signed byte in AL signed word in AX. Result in AX = 1111 1111 1001 1000
CWD: Convert Signed Word to Double Word
This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be sign extension of AL.
Eg. CWD
Convert signed word in AX to signed double word in DX: AX
DX= 1111 1111 1111 1111
Result in AX = 1111 0000 1100 0001
DIV: Unsigned division
This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word by a word.
Eg. DIV CL; Word in AX / byte in CL; Quotient in AL, remainder in AH
DIV CX; Double word in DX and AX / word; in CX, and Quotient in AX; remainder in DX
AAA: ASCII Adjust After Addition
The AAA instruction is executed after an ADD instruction that adds two ASCII coded operand to give a byte of result in AL. The AAA instruction converts the resulting contents of Al to a unpacked decimal digits.
Eg. ADD CL, DL; [CL] = 32H = ASCII for 2; [DL] = 35H = ASCII for 5; Result [CL] = 67H
MOV AL, CL; Move ASCII result into AL since; AAA adjust only [AL]
AAA; [AL]=07, unpacked BCD for 7
AAS: ASCII Adjust AL after Subtraction
This instruction corrects the result in AL register after subtracting two unpacked ASCII operands. The result is in unpacked decimal format. The procedure is similar to AAA instruction except for the subtraction of 06 from AL.
AAM: ASCII Adjust after Multiplication
This instruction, after execution, converts the product available In AL into unpacked BCD format.
Eg. MOV AL, 04; AL = 04
MOV BL ,09; BL = 09
MUL BL; AX = AL*BL; AX=24H
AAM; AH = 03, AL=06
AAD: ASCII Adjust before Division
This instruction 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. In the instruction sequence, this instruction appears Before DIV instruction.
Eg. AX 05 08
AAD result in AL 00 3A 58D = 3A H in AL
The result of AAD execution will give the hexadecimal number 3A in AL and 00 in AH where 3A is the hexadecimal Equivalent of 58 (decimal).

DAA: Decimal Adjust Accumulator

This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number. The result has to be only in AL.

Eg. AL = 53 CL = 29
ADD AL, CL; AL (AL) + (CL); AL 53 + 29;
AL 7C
DAA; AL 7C + 06 (as C>9); AL 82

DAS: Decimal Adjust after Subtraction

This instruction converts the result of the subtraction of two packed BCD numbers to a valid BCD number. The subtraction has to be in AL only.

Eg. AL = 75, BH = 46
SUB AL, BH; AL 2 F = (AL) - (BH)
; AF = 1
DAS; AL 2 9 (as F>9, F - 6 = 9)

Logical instructions

AND: Logical AND

This instruction bit by bit ANDs the source operand that may be an immediate register or a memory location to the destination operand that may a register or a memory location. The result is stored in the destination operand.

Eg. AND AX, 0008H
AND AX, BX

OR: Logical OR

This instruction bit by bit ORs the source operand that may be an immediate, register or a memory location to the destination operand that may a register or a memory location. The result is stored in the destination operand.

Eg. OR AX, 0008H
OR AX, BX

NOT: Logical Invert

This instruction complements the contents of an operand register or a memory location, bit by bit.

Eg. NOT AX
NOT [5000H]

OR: Logical Exclusive OR

This instruction bit by bit XORs the source operand that may be an immediate, register or a memory location to the destination operand that may a register or a memory location. The result is stored in the destination operand.

Eg. XOR AX, 0098H
XOR AX, BX

TEST: Logical Compare Instruction

The TEST instruction performs a bit by bit logical AND operation on the two operands.The result of this AND operation is not available for further use, but flags are affected.

Eg. TEST AX, BX
TEST [0500], 06H

Shift and Rotate Instructions

SAL/SHL: SAL / SHL destination, count.

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination to the left and 0 is stored at LSB position. The MSB is shifted into the carry flag. The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts is indicated by count.

Eg. SAL CX, 1
SAL AX, CL

SHR: SHR destination, count

This instruction shifts each bit in the specified destination to the right and 0 is stored at MSB position. The LSB is shifted into the carry flag. The destination can be a byte or a word.It can be a register or in a memory location. The number of shifts is indicated by count.

Eg. SHR CX, 1
MOV CL, 05H
SHR AX, CL

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. The LSB will be shifted into CF.

Eg. SAR BL, 1
MOV CL, 04H
SAR DX, CL

ROL Instruction: ROL destination, count

This instruction rotates all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.

Eg. ROL CX, 1
MOV CL, 03H
ROL BL, CL

ROR Instruction: ROR destination, count

This instruction rotates all bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF.

Eg. ROR CX, 1
MOV CL, 03H
ROR BL, CL

RCL Instruction: RCL destination, count

This instruction rotates all bits in a specified byte or word some number of bit positions to the left along with the carry flag. MSB is placed as a new carry and previous carry is place as new LSB.

Eg. RCL CX, 1
MOV CL, 04H

RCR Instruction: RCR destination, count

This instruction rotates all bits in a specified byte or word some number of bit positions to the right *along with the carry flag*. LSB is placed as a new carry and previous carry is place as new MSB.

Eg. RCR CX, 1
MOV CL, 04H
RCR AL, CL

ROR Instruction: ROR destination, count
This instruction rotates all bits in a specified byte or word to the *right* some number of bit positions. LSB is placed as a new MSB and a new CF.
Eg. ROR CX, 1
MOV CL, 03H
ROR BL, CL
RCL Instruction: RCL destination, count
This instruction rotates all bits in a specified byte or word some number of bit positions to the left along with the carry flag. MSB is placed as a new carry and previous carry is place as new LSB.
Eg. RCL CX, 1
MOV CL, 04H
RCL AL, CL

RCR Instruction: RCR destination, count
This instruction rotates all bits in a specified byte or word some number of bit positions to the right *along with the carry flag*. LSB is placed as a new carry and previous carry is place as new MSB.

Eg. RCR CX, 1
MOV CL, 04H
RCR AL, CL

Loop Instructions:
Unconditional LOOP Instructions
LOOP: LOOP Unconditionally
This instruction executes the part of the program from the Label or address specified in the instruction upto the LOOP instruction CX number of times. At each iteration, CX isdecremented automatically and JUMP IF NOT ZERO structure.
Example: MOV CX, 0004H

Conditional LOOP Instructions
LOOPZ / LOOPE Label
Loop through a sequence of instructions from label while ZF=1 and CX=0.
LOOPNZ / LOOPENE Label
Loop through a sequence of instructions from label while ZF=1 and CX=0.

Branch Instructions:
Branch Instructions transfers the flow of execution of the program to a new address specified n the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location to betransferred.
The Branch Instructions are classified into two types

1. Unconditional Branch Instructions.
2. Conditional Branch Instructions.

Unconditional Branch Instructions:

In Unconditional control transfer instructions, the execution control is transferred to the specified location independent of any status or condition. The CS and IP are unconditionally modified to the new CS and IP.

CALL: Unconditional Call

This instruction is used to call a Subroutine (Procedure) from a main program. Address of procedure may be specified directly or indirectly. There are two types of procedure depending upon whether it is available in the same segment or in another segment.

i. Near CALL i.e., ±32K displacement.

ii. For CALL i.e., anywhere outside the segment.

On execution this instruction stores the incremented IP & CS onto the stack and loads

the CS & IP registers with segment and offset addresses of the procedure to be called.

RET: Return from the Procedure.

At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with Flags are retrieved into the CS, IP and Flag registers from the stack and execution of the main program continues further.

INT N: Interrupt Type N.

In the interrupt structure of 8086, 256 interrupts are defined corresponding to the types from 00H to FFH. When INT N instruction is executed, the type byte N is multiplied by 4 and the contents of IP and CS of the interrupt service routine will be taken from memory block in 0000 segment.

INTO: Interrupt on Overflow

This instruction is executed, when the overflow flag OF is set. This is equivalent to a Type 4 Interrupt instruction.

JMP: Unconditional Jump

This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement. No Flags are affected by this instruction.

IRET: Return from ISR

When it is executed, the values of IP, CS and Flags are retrieved from the stack to continue the execution of the main program.

```
MOV BX, 7526H
Label 1 MOV AX, CODE
OR BX, AX
LOOP Label 1
```

Conditional Branch Instructions

When this instruction is executed, execution control is transferred to the address specified relatively in the instruction, provided the condition implicit in the Opcode is satisfied. Otherwise execution continues sequentially.

JZ/JE Label

Transfer execution control to address 'Label', if ZF=1.

JNZ/JNE Label

Transfer execution control to address 'Label', if ZF=0

JS Label

Transfer execution control to address 'Label', if SF=1.

JNS Label

Transfer execution control to address ‘Label’, if SF=0.

JO Label
Transfer execution control to address ‘Label’, if OF=1.
14
JNO Label
Transfer execution control to address ‘Label’, if OF=0.
JNP Label
Transfer execution control to address ‘Label’, if PF=0.
JP Label
Transfer execution control to address ‘Label’, if PF=1.
JB Label
Transfer execution control to address ‘Label’, if CF=1.
JNB Label
Transfer execution control to address ‘Label’, if CF=0.
JCXZ Label
Transfer execution control to address ‘Label’, if CX=0

String Manipulation Instructions
A series of data byte or word available in memory at consecutive locations, to be referred as Byte String or Word String. A String of characters may be located in consecutive memory locations, where each character may be represented by its ASCII equivalent. The 8086 supports a set of more powerful instructions for string manipulations for referring to a string, two parameters are required.

I. Starting and End Address of the String.
II. Length of the String.

The length of the string is usually stored as count in the CX register. The incrementing or decrementing of the pointer, in string instructions, depends upon the Direction Flag (DF) Status. If it is a Byte string operation, the index registers are updated by one. On the other hand, if it is a word string operation, the index registers are updated by two.

REP: Repeat Instruction Prefix
This instruction is used as a prefix to other instructions, the instruction to which the REP prefix is provided, is executed repeatedly until the CX register becomes zero (at each iteration CX is automatically decremented by one).
i. REPE / REPZ - repeat operation while equal / zero.
ii. REPNE / REPNZ - repeat operation while not equal / not zero.
These are used for CMPS, SCAS instructions only, as instruction prefixes.

MOVSB / MOVSW: Move String Byte or String Word
Suppose a string of bytes stored in a set of consecutive memory locations is to be moved to another set of destination locations. The starting byte of source string is located in the memory location whose address may be computed using SI (Source Index) and DS (Data Segment) contents. The starting address of the destination locations where this string has to be relocated is given by DI (Destination Index) and ES (Extra Segment) contents.

CMPS: Compare String Byte or String Word
The CMPS instruction can be used to compare two strings of byte or words. The length of the string must be stored in the register CX. If both the byte or word strings are equal, zero Flag is set. The REP instruction Prefix is used to repeat the operation till CX (counter) becomes zero or the condition specified by the REP Prefix is False.
SCAN: Scan String Byte or String Word
This instruction scans a string of bytes or words for an operand byte or word specified in the register AL or AX. The String is pointed to by ES: DI register pair. The length of the string sstored in CX. The DF controls the mode for scanning of the string. Whenever a match to the specified operand is found in the string, execution stops and the zero Flag is set. If no match is found, the zero flag is reset.
LODS: Load String Byte or String Word
The LODS instruction loads the AL / AX register by the content of a string pointed to by DS: SI register pair. The SI is modified automatically depending upon DF, If it is a byte transfer (LODSB), the SI is modified by one and if it is a word transfer (LODSW), the SI is modified by two. No other Flags are affected by this instruction.
STOS: Store String Byte or String Word
The STOS instruction Stores the AL / AX register contents to a location in the string pointer by ES: DI register pair. The DI is modified accordingly, No Flags are affected by this instruction. The direction Flag controls the String instruction execution, The source index SI and Destination Index DI are modified after each iteration automatically. If DF=1, then the execution follows auto decrement mode, SI and DI are decremented automatically after each iteration. If DF=0, then the execution follows auto increment mode. In this mode, SI and DI are incremented automatically after each iteration.

4. a) Write an 8086 ALP to perform matrix multiplication? [CO1-H1-May/Jun 2013]
b) Write an 8086 ALP to perform LCM and GCD of two numbers[CO1-H1-Nov/Dec 2011]
ALP for addition of two 8-bit numbers

```
DATA SEGMENT
VAR1 DB 85H
VAR2 DB 32H
RES DB?
DATA ENDS
ASSUME CS:CODE, DS:DATA
CODE SEGMENT
START: MOV AX, DATA
MOV DS, AX
MOV AL, VAR1
MOV BL, VAR2
ADD AL, BL
MOV RES, AL
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

ALP for Subtraction of two 8-bit numbers

```
DATA SEGMENT
VAR1 DB 53H
VAR2 DB 2AH
RES DB?
DATA ENDS
ASSUME CS:CODE,DS:DATA
CODE SEGMENT
START: MOV AX,DATA
MOV DS,AX
MOV AL,VAR1
MOV BL,VAR2
SUB AL,BL
MOV RES,AL
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

ALP for Multiplication of two 8-bit numbers

```
DATA SEGMENT
VAR1 DB 0EDH
VAR2 DB 99H
RES DW?
DATA ENDS
ASSUME CS: CODE, DS:DATA
CODE SEGMENT
START: MOV AX, DATA
MOV DS, AX
MOV AL, VAR1
MOV BL, VAR2
MUL BL
MOV RES, AX
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

ALP for division of 16-bit number with 8-bit number

```
DATA SEGMENT
VAR1 DW 6827H
VAR2 DB 0FEH
QUO DB?
REM DB?
```

```
DATA ENDS
ASSUME CS:CODE,DS:DATA
CODE SEGMENT
START: MOV AX, DATA
MOV DS, AX
MOV AX, VAR1
DIV VAR2
MOV QUO, AL
MOV REM, AH
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

ALP for Subtraction of two 16-bit numbers

```
DATA SEGMENT
VAR1 DW 8560H
VAR2 DW 3297H
RES DW?
DATA ENDS
ASSUME CS: CODE,DS:DATA
CODE SEGMENT
START: MOV AX, DATA
MOV DS, AX
MOV AX, VAR1
CLC
SUB AX, VAR2
MOV RES, AX
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

5. Explain about modular programming in detail? [CO1-L2-Nov/Dec 2016]

ALP for Multiplication of two 32-bit numbers

```
DATA SEGMENT
MULD DW 0FFFFH, 0FFFFH
MULR DW 0FFFFH, 0FFFFH
RES DW 6 DUP (0)
DATA ENDS
ASSUME CS: CODE,DS: DATA
CODE SEGMENT
START: MOV AX, DATA
MOV DS, AX
MOV AX, MULD
MUL MULR
MOV RES, AX
MOV RES+2, DX
MOV AX, MULD+2
MUL MULR
ADD RES+2, AX
ADC RES+4, DX
MOV AX, MULD
MUL MULR+2
ADD RES+2, AX
ADC RES+4, DX
JNC K
INC RES+6
K: MOV AX, MULD+2
MUL MULR+2
ADD RES+4, AX
ADC RES+6, DX
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

ALP to Sort a set of unsigned integer numbers in ascending/ descending order using Bubble sort algorithm.

```
DATA SEGMENT
A DW 0005H, 0ABCDH, 5678H, 1234H, 0EFCDH, 45EFH
DATA ENDS
ASSUME CS: CODE, DS: DATA
CODE SEGMENT
START: MOV AX, DATA
MOV DS, AX
MOV SI, 0000H
MOV BX, A[SI]
DEC BX
X2: MOV CX, BX
MOV SI, 02H
X1: MOV AX, A[SI]
INC SI
INC SI
CMP AX, A[SI]
XCHG AX, A[SI]
MOV A[SI-2], AX
X3: LOOP X1
DEC BX
JNZ X2
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

6**. Explain about procedures and macros in detail with examples? [CO1-L2-May/Jun 2014]**

Procedures

A procedure is a set of code that can be branched to and returned from in such a way the code is as if it were inserted at the point from which it is branched to. The branch to procedure is referred to as the *call,* and the corresponding branch back is known as the *return*. The return is always made to the instruction immediately following the call regardless of where the call is located.

Calls, Returns, and Procedure Definitions

The CALL instruction not only branches to the indicated address, but also pushes the return address onto the stack. The RET instruction simply pops the return address from the stack. The registers used by the procedure need to be stored before their contents are changed, and then restored just before their contents are changed, and then restored just before the procedure is excited. A CALL may be direct or indirect and intrasegment or intersegment. If the CALL is intersegment, the return must be intersegment. Intersegment call must push both (IP) and(CS) onto the stack. The return must correspondingly pop two words from the stack. In the case of intrasegment call, only the contents of IP will be saved and retrieved when call and return instructions are used. Procedures are used in the source code by placing a statement of the form at the beginning of the procedure Procedure name PROC Attribute and by terminating the procedure with a statement Procedure name ENDP

The attribute that can be used will be either NEAR or FAR. If the attribute is NEAR, the RET instruction will only pop a word into the IP register, but if it is FAR, it will also pop a word into the CS register.

A procedure may be in:

1. The same code segment as the statement that calls it.
2. A code segment that is different from the one containing the statement that calls it,
   but in the same source module as the calling statement.
3. A different source module and segment from the calling statement.

In the first case, the attribute could be NEAR provided that all calls are in the same codesegment as the procedure. For the latter two cases the attribute must be FAR. If the procedure is given a FAR attribute, then all calls to it must be intersegment calls even if the call is from the same code segment. For the third case, the procedure name must be declared in EXTRN and PUBLIC statements.

Saving and Restoring Registers

When both the calling program and procedure share the same set of registers, it is necessary to save the registers when entering a procedure, and restore them before returning to the
calling program.

```
MSK PROC NEAR
PUSH AX
PUSH BX
PUSH CX
POP CX
POP BX
POP AX
RET
MSK ENDP
```

Procedure Communication

There are two general types of procedures, those operate on the same set of data and those that may process a different set of data each time they are called. If a procedure is in the same source module as the calling program, then the procedure can refer to the variables directly.When the procedure is in a separate source module it can still refer to the source module directly provided that the calling program contains the directive

```
PUBLIC ARY, COUNT, SUM
EXTRN ARY: WORD, COUNT: WORD, SUM: WORD
```

Recursive Procedures

When a procedure is called within another procedure it called recursive procedure. To make sure that the procedure does not modify itself, each call must store its set of parameters, registers, and all temporary results in a different place in memory

Eg. Recursive procedure to compute the factorial

7. Explain about Interrupts and its service routine in detail? [CO1-L2-Nov/Dec 2016]

Interrupts and Interrupt Routines

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

i) by an external signal generated by a peripheral,

ii) by an internal signal generated by a special instruction in the program,

iii) by an internal signal generated due to an exceptional condition which occurs while executing an instruction. (For example, in 8086 processor, divide by zero is an exceptional condition which initiates type 0 interrupt and such an interrupt is also called execution).

The process of interrupting the normal program execution to carry out a specific task/work is referred to as interrupt. The interrupt is initiated by a signal generated by an external device or by a signal generated internal to 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 of8086) 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. The external interrupts are used to implement interrupt driven data transfer scheme. The interrupts generated by special instructions are called software interrupts and they are used to implement system services/calls (or monitor services/calls). The system/monitor services are procedures developed by system designer for various operations and stored in memory. The user can call these services through software interrupts. The interrupts generated by exceptional conditions are used to implement error conditions in the system.

Interrupt Driven Data Transfer Scheme

The interrupts are useful for efficient data transfer between processor and peripheral. When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal. Upon receiving an interrupt signal, the processor suspends the current program execution, save the status in stack and executes an ISR to perform the data transfer between the peripheral and processor. At the end of ISR the processor status is restored from stack and processor resume its normal program execution. This type of data transfer scheme is called interrupt driven data transfer scheme. The data transfer between the processor and peripheral devices can be implemented either by polling technique or by interrupt method. In polling technique, the processor has to periodically poll or check the status/readiness of the device and can perform data transfer only when the device 'is ready. In polling technique the processor time is wasted, because the processor has to suspend its work and check the status of the device in predefined intervals.

If the device interrupts the processor to initiate a data transfer whenever it is ready then the processor time is effectively utilized because the processor need not suspend its work and check the status of the device in predefined intervals. For an example, consider the data transfer from a keyboard to the processor. Normally a keyboard has to be checked by the processor once in every 10 milliseconds for a key press. Therefore once in every 10 milliseconds the processor has to suspend its work and then check the keyboard for a valid key code. Alternatively, the keyboard can interruptthe processor, whenever a key is pressed and a valid key code is generated. In this way theprocessor need not waste its time to check the keyboard once in every 10 milliseconds.

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.

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 interrupts initiated by applying appropriate signal to these pins are called hardware interrupts of 8086.

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.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).The processor has the facility for accepting or rejecting hardware interrupts. Programming the processor to reject an interrupt is referred to as masking or disabling and programming the processor to accept an interrupt is referred to as unmasking or enabling. In8086 the interrupt flag (IF) can be set to one to unmask or enable all hardware interrupts and IF is cleared to zero to mask or disable a hardware interrupts except NMI.

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 (orcannot 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 8086processor 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. When the interrupt is initiated through INTR pin, then the processor runs an interrupt acknowledge cycle to get the type number. (i.e., the interrupting device should supply the type number through D0- D7 lines when the processor requests forthe same through interrupt acknowledge cycle).

Unit –II

8086 System Bus Structure

Part-A

1. **How 8086 signals are classified?[CO2-L2-May/Jun 2011]**

The 8086 signals are classified in three groups. They are :

- Signals having common function in Minimum and Maximum mode
- Signals for Minimum mode
- Signals for Maximum mode

2. **Mention few 8086 signals having common function in minimum mode and maximum mode?[CO2-L1-Nov/Dec 2013]**

AD_{15}-AD_{0} , A_{19}/S_{6}, A_{16}/S_{3} , READY , NMI , CLK

3. **When the 8086 processor will operate in minimum and maximum mode? [CO2-L1-May/Jun 2015]**

By assigning the pin MN/MX = 5V, the 8086 will operate in minimum mode and when MN/MX = GND, the 8086 will operate in maximum mode.

4. **Differentiate minimum mode and maximum mode system configuration in 8086? [CO2-L2-Nov/Dec 2012]**

In minimum mode the 8086 works in single processor environment whereas in maximum mode the 8086 works in multiprocessor environment.

5. **What is DMA? [CO2-L1-Nov/Dec 2014]**

A direct memory access (DMA) is an operation in which data is copied (transported) from one resource to another resource in a computer system without the involvement of the CPU.

6. **Mention the types of I/O programming. [CO2-L1-May/Jun 2013]**

Programmed I/O
Interrupt I/O

7. **How priority is given to an interrupt system? [CO2-L1-Nov/Dec 2015]**

Priority to an interrupt systems is given by the following means :

- Polling
- Daisy chaining
- Interrupt priority management hardware.

8. **Define bus cycle and cycle stealing. [CO2-L1-Nov/Dec 2013]**

The activity involved in transferring a byte or word over the system bus is called a bus cycle. Taking control of the bus for a bus cycle is called cycle stealing.

9. **What is multiprogramming? [CO2-L1-May/Jun 2011]**

In the concept of multiprogramming the code for two or more processes is in memory at the same time and is executed in a time-multiplexed manner. Multiprogramming improve the system performance by overlapping the I/O processing and the CPU operations.

10. What are the three states in multiprogramming system?[CO2-L2]

In multiprogramming system there are three states that the processes can be in, with each process being in exactly one of these two states at any given time. These states are : Running Blocked and Ready.

11. What is a multiprocessing system? [CO2-L1]

If a system includes two or more components that can execute instructions simultaneously, it is called a multiprocessing system.

12. What are the types of multiprocessor configuration available in 8086? [CO2-L1-May/Jun 2013]

Multiprocessing features are provided in maximum mode to accommodate three basic configurations. They are the coprocessor, closely coupled and loosely coupled configurations.

13. Give an example for Coprocessor configuration. [CO2-L1]

The interfacing of a coprocessor to a host CPU is an example of coprocessor configuration. Both the host and the coprocessor share the same clock generator and bus control logic.

14. What are the advantages of loosely coupled configuration? [CO2-L1-May/Jun 2013]

(i) High system throughput can be achieved by having more than one CPU.
(ii) 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.

15. What is a Coprocessor? [CO2-L2-May/Jun 2015]

Most microprocessor has limited mathematical processing capabilitysuch as addition, subtraction, multiplication and division. They do not evaluatetrigonometric, logarithmic and exponential functions. Therefore, specializedprocessors have been developed to solve this problem. These are called coprocessor.

16. Mention the advantages of multiprogramming. [CO2-L1]

(i) It increases CPU utilization
(ii) It decreases total read time needed to execute a job
(iii) It maximizes the total job throughput of a computer.

17. State the various bus arbitration schemes. [CO2-L1]

(i) Daisy chaining
(ii) Polling method

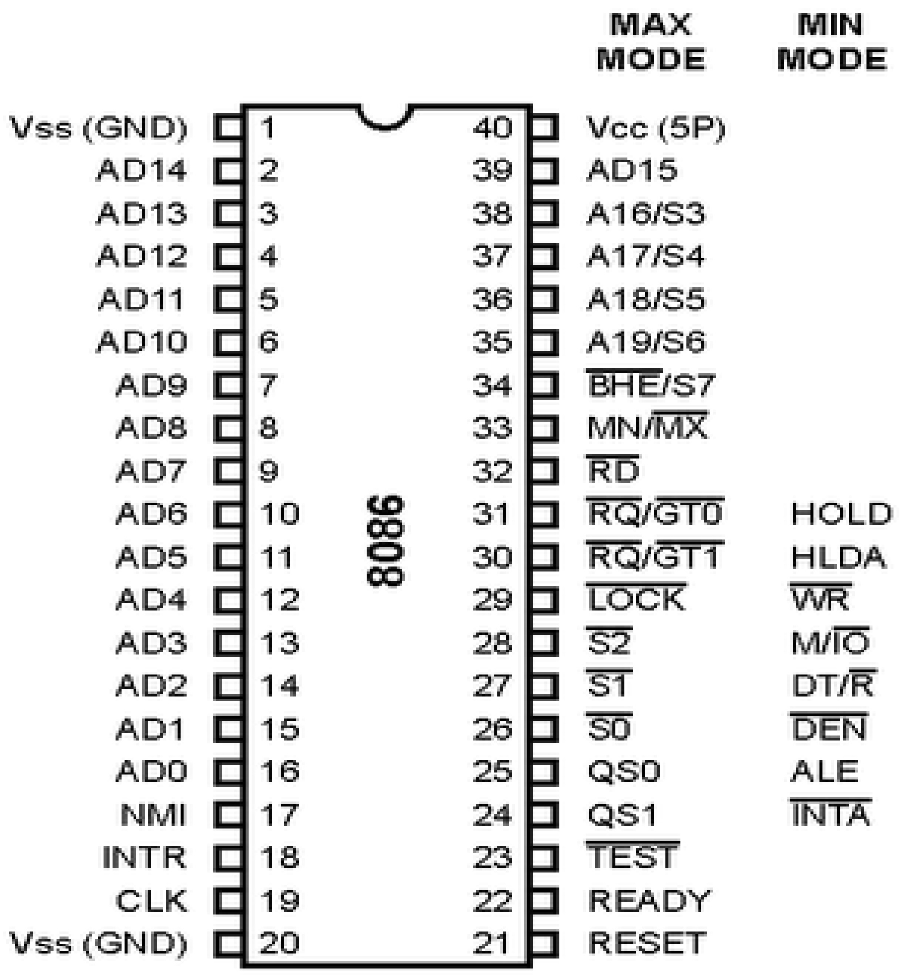
Part-B

1. **Explain the Maximum and Minimum mode of operation of 8086.[CO2-L2-May/Jun 2014]**

Minimum and Maximum Modes:

•The minimum mode is selected by applying logic 1 to the MN / MX input pin. This is a single microprocessor configuration.

•The maximum mode is selected by applying logic 0 to the MN / MX input pin. This is a multi micro processors configuration.



Signal Description of 8086•The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.

•The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

•The 8086 signals can be categorised in three groups. The first are the signal having common functions in minimum as well as maximum mode.

•The second are the signals which have special functions for minimum mode and third are the signals having special functions for maximum mode.

The following signal descriptions are common for both modes.

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

• Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.

•These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

•**A19/S6,A18/S5,A17/S4,A16/S3:** These are the time multiplexed address and status lines.

•During T1 these are the most significant address lines for memory operations.

•During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2,T3,Tw and T4.

•The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.

•The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as in below fig.

•These lines float to tri-state off during the local bus hold acknowledge. The status line S6 is always low.

•The address bit are separated from the status bit using latches controlled by the ALE signal.

BHE /S7: The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in table. It goes low for the data transfer over D15- D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T2, T3 and T4. The signal is active low and tristated during hold. It is low during T1 for the first pulse of the interrupt acknowledges cycle.

•**RDRead:** This signal on low indicates the peripheral that the processor is performing memory or I/O read operation. RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.

•**READY**: This is the acknowledgement from the slow device 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.

•**INTR-Interrupt Request**: This is a triggered input. This is sampled during the last clock cycles 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 resulting the interrupt enable flag. This signal is active high and internally synchronized.

•**TEST**This input is examined by a 'WAIT' instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

•**CLK**- Clock Input: The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.

•**MN/MX**: The logic level at this pin decides whether the processor is to operate in either

minimum or maximum mode.

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

•**M/IO – Memory/IO**: This is a status line logically equivalent to S2 in maximum mode.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. This line becomes active high inthe previous T4 and remains active till final T4 of the current cycle. It is tristated duringlocal bus "hold acknowledge "

• **INTA Interrupt Acknowledge**: This signal is used as a read strobe for interruptacknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.

•**ALE – Address Latch Enable**: This output signal indicates the availability of the validaddress on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

•**DT/ R – Data Transmit/Receive**: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

•**DEN – Data Enable**: This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T2 until the middle of T4. This is tristated during ' hold acknowledge' cycle.

•**HOLD, HLDA- 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, in the middle of the next clock cycle after completing the current buscycle

.•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 is should be externally synchronized.

•If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided:

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

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

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

4. A Lock instruction is not being executed.

•***The following pin function are applicable for maximum mode operation of 8086***.

•**S2, S1, S0 – Status Lines**: These are the status lines which reflect the type of operation,being carried out by the processor. These become activity during T4 of the previous cycle and active during T1 and T2 of the current bus cycles.

General Bus Operation:

•The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.

•The main reason behind multiplexing address and data over the same pins is the maximum utilisation of processor pins and it facilitates the use of 40 pin standard DIP package.

•The bus can be demultiplexed using a few latches and transreceivers, when ever required.

•Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, T4. The address is transmitted by the processor during T1. It is present on the bus only for one cycle.

•The negative edge of this ALE pulse is used to separate the address and the data or statusinformation. In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation.

•Status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.

2. Draw and discuss a typical minimum mode 8086 system. [CO2-H1-Nov/Dec 2016]

MINIMUM MODE 8086 SYSTEM

•In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.

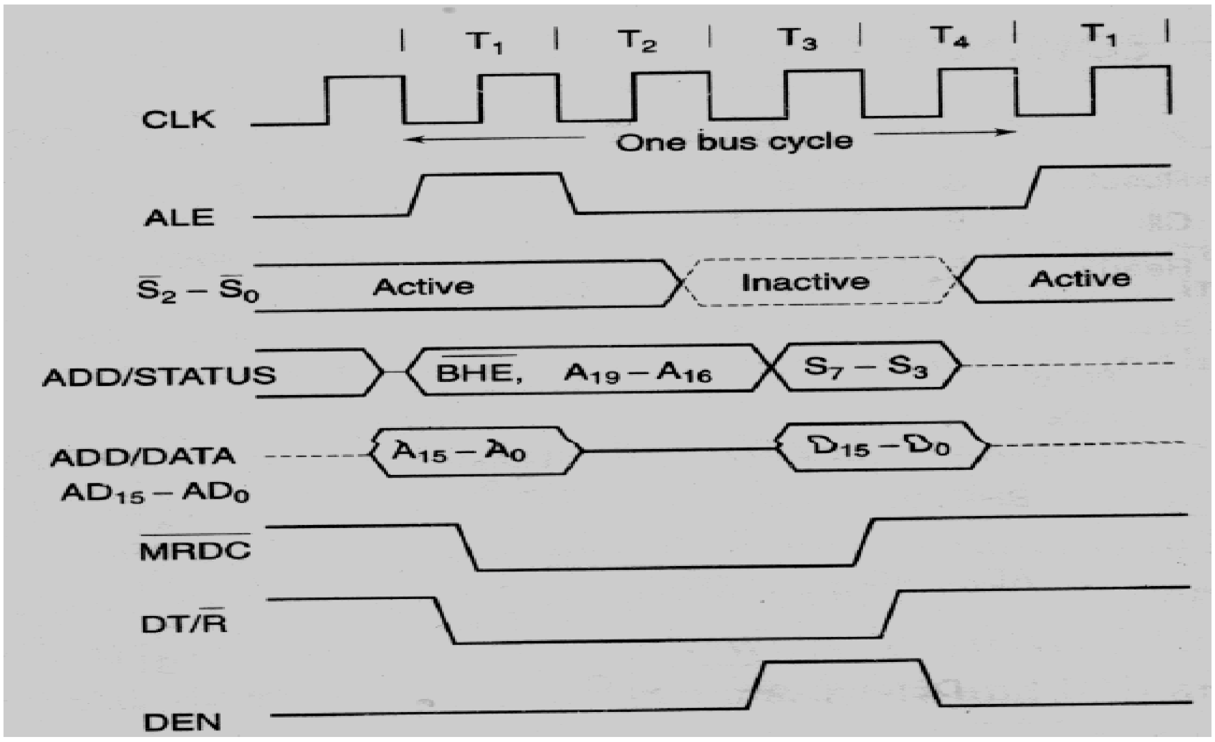
•In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system.

•The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

•Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

General Bus Operation Cycle in Maximum Mode

•Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.



•They are controlled by two signals namely, DEN and DT/R.

•The DEN signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.

•Usually, EPROM are used for monitor storage, while RAM for users program storage. A system may contain I/O devices.

•The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.

•The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.

•The read cycle begins in T1 with the assertion of 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 A0 signals address low, high or both bytes. From T1 to T4 , 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 T2.

•The read (RD) signal causes the address 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.

•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 T2, after sending the address in T1, the processor sends the data to be written to the addressed location.

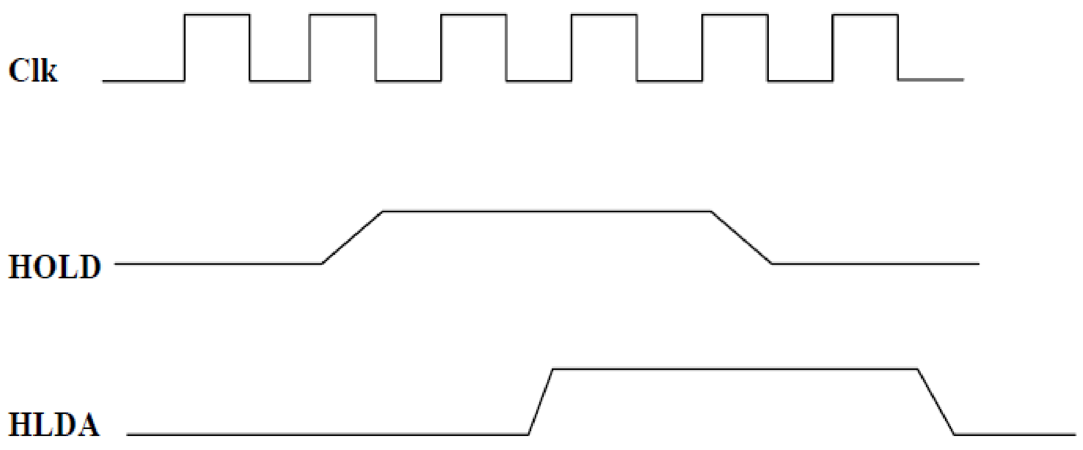
•The data remains on the bus until middle of T4 state. The WR becomes active at the beginning of T2 (unlike RD is somewhat delayed in T2 to provide time for floating).

•The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.

•The M/IO, RD and WR signals indicate the type of data transfer as specified in table below.
Write Cycle Timing Diagram for Minimum Mode

•***Hold Response sequence***: The HOLD pin is checked at leading edge of each clock pulse. If it is received active by the processor before T4 of the previous cycle or during T1 state of the current cycle, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.

•The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock.



Request and Bus Grant Timings in Minimum Mode System

Maximum Mode 8086 System •In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.

•In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.

•In the maximum mode, there may be more than one microprocessor in the system configuration.

•The components in the system are same as in the minimum mode system.

•The basic function 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 by the processor on the status lines.

•The bus controller chip has input lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU.

•It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are specially useful for multiprocessor systems.

•AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.

•If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.

•INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

•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 address port.

•The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.

•All these command signals instructs 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.

•Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.

•R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.

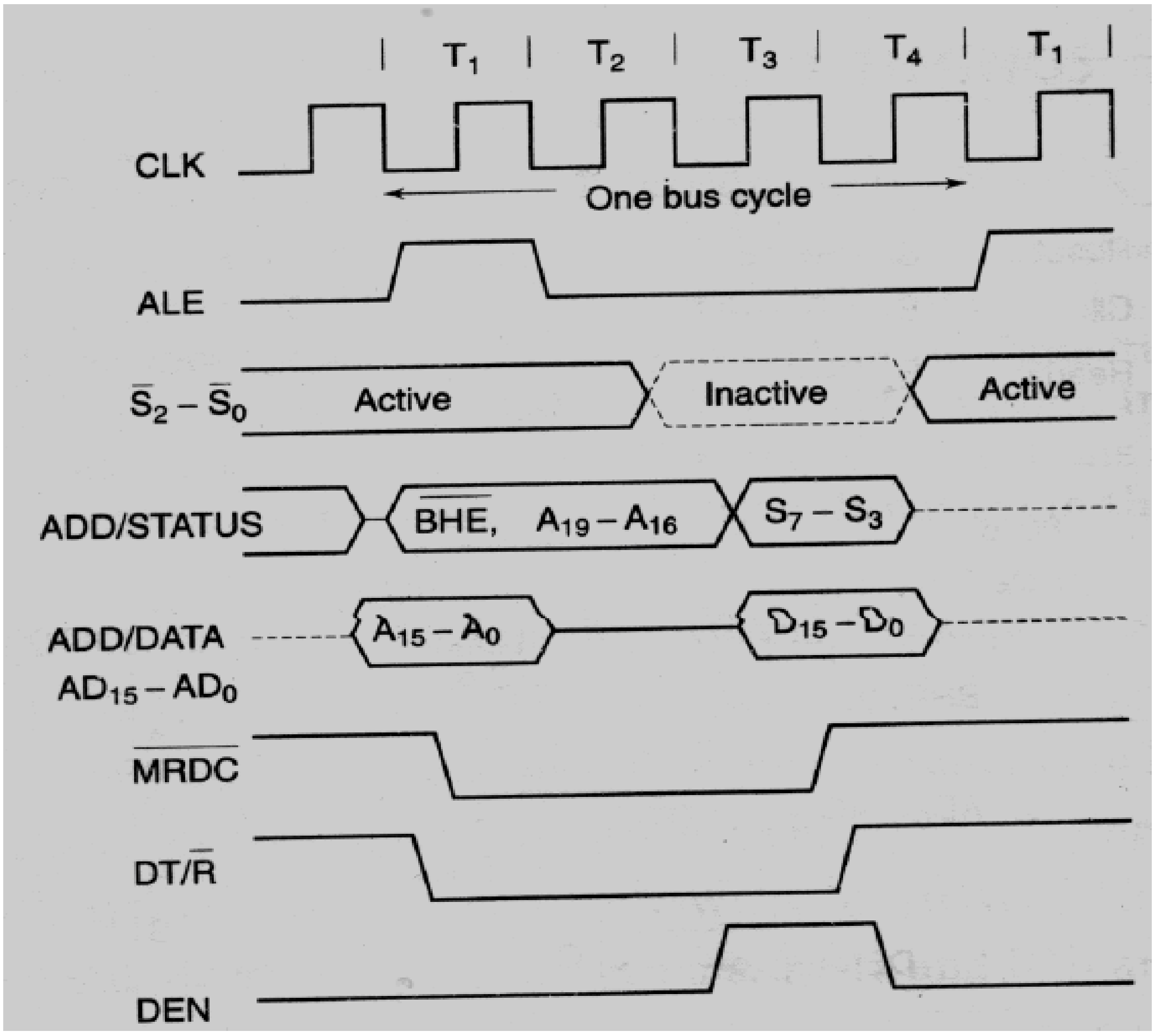
•In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.

•The status bit S0 to S2 remains active until T3 and become passive during T3 and T4**.**

•If reader input is not activated before T3, wait state will be inserted between T3 and T4.

3. Describe the Maximum Mode Signals, Bus Cycles and Maximum Mode System Configuration of 8086 Microprocessor in detail. [CO2-H1]

Memory Read Timing in Maximum Mode



MINIMUM MODE INTERFACE

•When the Minimum mode operation is selected, the 8086 provides all control signals needed to implement the memory and I/O interface. Memory Write Timing in Maximum mode.

•The minimum mode signal can be divided into the following basic groups: address/data bus, status, control, interrupt and DMA.

•**Address/Data Bus**: these lines serve two functions. As an address bus is 20 bits long and consists of signal lines A0 through A19. A19 represents the MSB and A0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. More over it has an independent I/O address space which is 64K bytes in length.

•The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D15 is the MSB and D0 LSB.

•When acting as a data bus, they carry read/write data for memory, input/output data forI/O devices, and interrupt type codes from an interrupt controller.

4. Explain how the communication between CPU and IOP processor takes place [CO2-L2-May/Jun 2015]

IO programming

On the 8086, all programmed communications with the I/O ports is done by the IN and OUT instructions.

IN and OUT instructions

Name	Mnemonic and Format	Description
Input		
Long form, byte	IN AL, PORT	(AL) <- (PORT)
Long form, word	IN AX, PORT	(AX) <- (PORT+1: PORT)
Short form, byte	IN AL, DX	(AL) <- ((DX))
Short form, word	IN AX, DX	(AX) <- ((DX) + 1: X))
Output		

Note: PORT is a constant ranging from 0 to 255

Flags: No flags are affected

Addressing modes: Operands are limited as indicated above.

If the second operand is DX, then there is only one byte in the instruction and the contents of DX are used as the port address. Unlike memory addressing, the contents of DX are not modified by any segment register. This allows variable access to I/O ports in the range 0 to 64K. The machine language code for the IN instruction is:

Although AL or AX is implied as the first operand in an IN instruction, either AL or AX must be specified so that the assembler can determine the W-bit

Similar comments apply to the OUT instruction except that for it the port address is the destination and is therefore indicated by the first operand, and the second operand is either AL or AX. Its machine code is:

Note that if the long form of the IN or OUT instruction is used the port address must be in the range 0000 to 00FF, but for the short form it can be any address in the range 0000 to FFFF (i.e. any address in the I/O address space). Neither IN nor OUT affects the flags.

The IN instruction may be used to input data from a data buffer register or the status from a status register. The instructions

```
IN  AX, 28H
MOV DATA_WORD, AX
```

would move the word in the ports whose address are 0028 and 0029 to the memory location DATA_WORD.

6. Differentiate closely coupled configuration and loosely coupled configuration [CO2-H1-Nov/Dec 2016]

Multiprocessor Systems

Multiprocessor Systems refer to the use of multiple processors that execute instructions simultaneously and communicate using mailboxes and semaphores Maximum mode of 8086 is designed to implement 3 basic multiprocessor configurations:

1. Coprocessor (8087)
2. Closely coupled (dedicated I/O processor: 8089)
3. Loosely coupled (Multi bus)

Coprocessors and closely coupled configurations are similar - both the CPU and the external processor share:

Memory
I/O system
Bus & bus control logic
Clock generator

Coprocessor configurations

Coprocessor Configuration:

WAIT instruction allows the processor to synchronize itself with external hardware, eg., waiting for 8087 math co-processor.

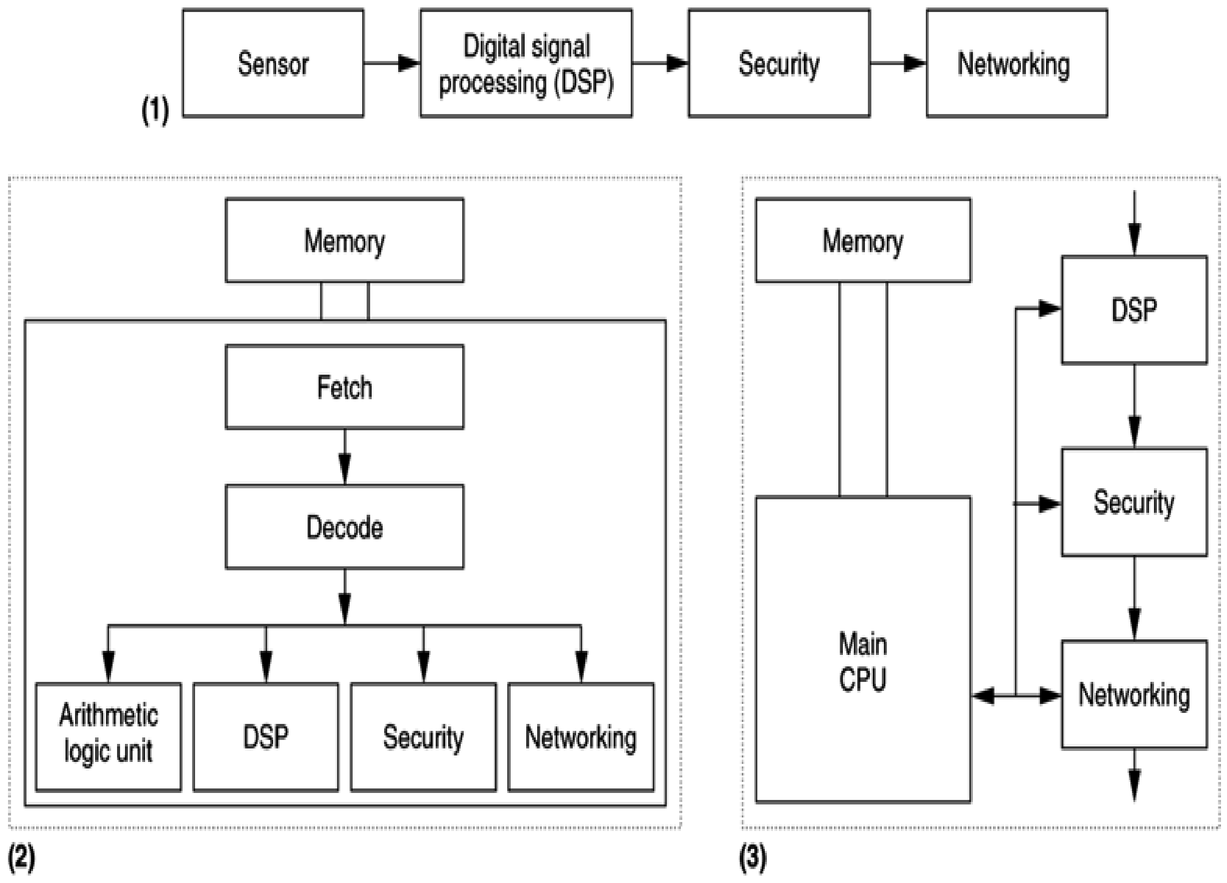
When the CPU executes WAIT waiting state.

TEST input is asserted (low), the waiting state is completed and execution will resume.

ESC instruction:

ESC opcode, operand, opcode: immediate value recognizable to a coprocessor as an instruction opcode Operand: name of a register or a memory address (in any ode)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

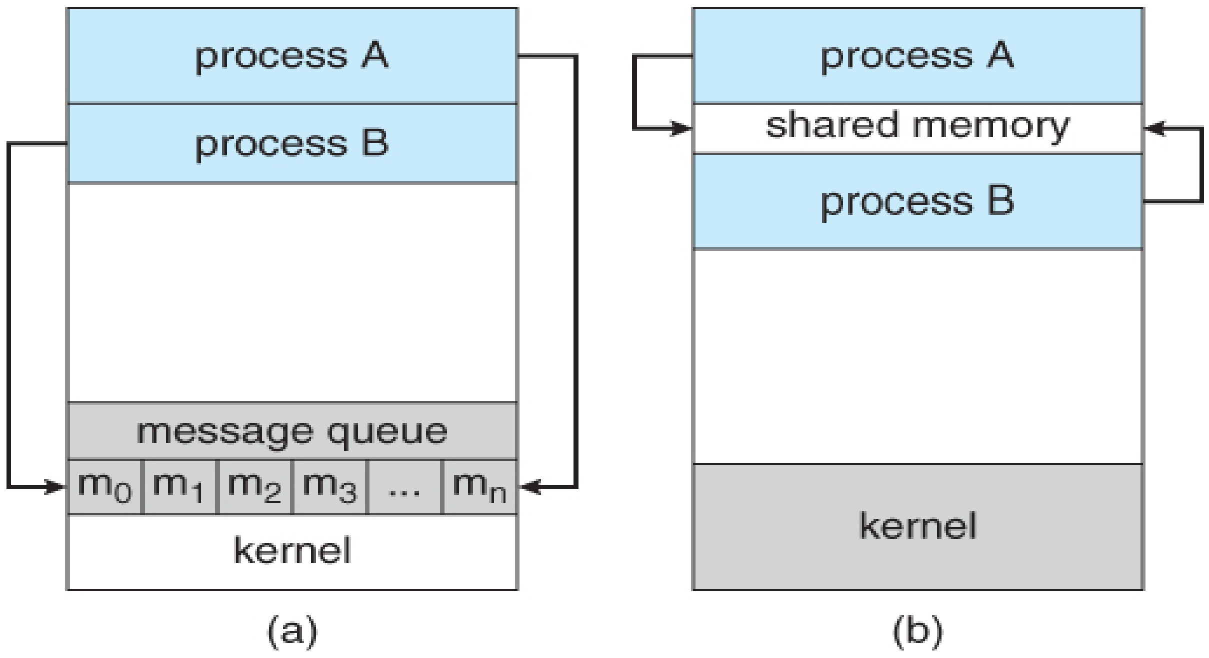
Synchronisation between the 8086 and its coprocesssor



Coprocessor configuration

1 . Coprocessor cannot take control of the bus, it does everything through the CPU.
2 . Closely Coupled processor may take control of the bus independently - 8089 shares.
3. CPU's clock and bus control logic.
4 . Communication with host CPU is by way of shared memory.
5. Host sets up a message (command) in memory.
6 . Independent processor interrupts host on completion.
7 . Two 8086's cannot be closely coupled.

Interprocessor communication through shared memory



Bus allocation schemes

Loosely Coupled Configurations:

A loosely coupled configuration provides the following advantages:

1. High system throughput can be achieved by having more than one CPU.
2. 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.
3. 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.
4. 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

Extra bus control logic must be provided to resolve the bus arbitration problem. The extra logic is called bus access logic and it is its responsibility to make sure that only one bus master at a time has control of the bus. Simultaneous bus requests are resolved on a **priority basis:** There are three schemes for establishing priority:

1. Daisy chaining.
2. Polling.
3. Independent requesting

7. Write a short note on 80286[CO2-L2]

Salient Features of 80286

The 80286 is the first member of the family of advanced microprocessors with memory management and protection abilities. The 80286 CPU, with its 24-bit address bus is able to address 16 Mbytes of physical memory. Various versions of 80286 are available that runs on 12.5 MHz, 10 MHz and 8 MHz clock frequencies. 80286 is upwardly compatible with 8086 in terms of instruction set.

1. 80286 has two operating modes namely real address mode and virtual address mode. In real address mode, the 80286 can address upto 1Mb of physical memory address like 8086. In virtual address mode, it can address up to 16 Mb of physical memory address space and 1 GB of virtual memory address space.
2. The instruction set of 80286 includes the instructions of 8086 and 80186. 80286 has some extra instructions to support operating system and memory management. In real address mode, the 80286 is object code compatible with 8086. In protected virtual address mode, it is source code compatible with 8086. The performance of 80286 is five times faster than the standard 8086.

Need for Memory Management

The part of main memory in which the operating system and other system programs are stored is not accessible to the users. It is required to ensure the smooth execution of the running process and also to ensure their protection. The memory management which is an important task of the operating system is supported by a hardware unit called memory management unit.

Swapping in of the Program

Fetching of the application program from the secondary memory and placing it in the physical memory for execution by the CPU.

Swapping out of the executable Program

Saving a portion of the program or important results required for further execution back to the secondary memory to make the program memory free for further execution of another required portion of the program.

Unit-III

I/O Interfacing

Part-A

1. What are the modes of operation supported by 8255? [CO3-L3-May/Jun 2015]

The three modes of operation supported by 8255 are :
Mode 0-Basic Input/Output
Mode 1- StrobedInput/Output
Mode 2 – Strobed Bidirectional Bus

2. How port selection is done for 8255? [CO3-L1-Nov/Dec 2015]

Control Signals			Selection
CS	A_1	A_0	
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	X	X	8255 is not selected

3. What is BSR mode in 8255? [CO3-L3-May/Jun 2014]

BSR (Bit Set/Reset) Mode is applicable for only Port C. A control word with $D_7 = 0$ is recognized as BSR control word.This control word can set or reset a single bit in Port C.

4. What are the internal devices of 8255? [CO3-L3]

Two 8-bit ports (A and B) ,Two 4-bit ports(C_U and C_L)
Data bus bufferand Control logic.

5. Specify the bit of a control word for the 8255, which differentiates between the I/O mode and the BSR mode? [CO3-L1]

In the control word format ifD7=0 then BSR mode andif D7 =1, it sets for I/O mode.

6. What are the features of 8279? [CO3-L1-Nov/Dec 2011]

The 8279 keyboard/display controller has the following features:
8279 has 3 input modes for keyboard interface

(i) Scanned keyboard mode
(ii) Scanned sensor matrix mode
(iii) Strobed input mode

8279 has 2 output modes for display interface: Left entry and Right entry
It has two depression modes : 2 Key lockout mode and N key rollover mode

7. List the major components of 8257 keyboard/display interface? [CO3-L1-May/Jun 2013]
Keyboard section,Scan section, Display section and MPU interface.

8. What are the tasks involved in keyboard interface? [CO3-L3]
The task involved in keyboard interfacing are sensing a key actuation,Debouncing the key and Generating key codes (Decoding the key).These task are performed using software if the keyboard is interfacedthrough ports and they are performed by hardware if the keyboard isinterfaced through 8279.

9. How a keyboard matrix is formed in keyboard interface using 8279? [CO3-L3Nov/Dec 2015]
The return lines, RLo to RL7 of 8279 are used to form the columns of keyboard matrix. In decoded scan mode, the scan lines SLo to SL3 of 8279 are used to form the rows of keyboard matrix. In encoded scan mode, the output lines of external decoder are used as rows of keyboard matrix.

10. What is scanning in keyboard and what is scan time?[CO3-L1-May/Jun 2012]
The process of sending a zero to each row of a keyboard matrix and reading the columns for key actuation is called scanning. The scan time is the time taken by the processor to scan all the rows one by one starting from first row and coming back to the first row again.

11. What is scanning in display and what is the scan time?[CO3-L1]
In display devices, the process of sending display codes to 7 –segment LEDs to display the LEDs one by one is called scanning (or multiplexed display). The scan time is the time taken to display all the 7-segment LEDs one by one, starting from first LED and coming back to the first LED again.

12. What is debouncing? What are the methods to detect the debouncing?[CO3-L1]
When a key is pressed it bounces for a short time. If a key code is generated immediately after sensing a key actuation, then the processor will generate the same key code a number of times. (A key typically bounces for 10 to 20 milliseconds.) Hence the processor has to wait for the key bounces to settle down before reading the keycode. This process is called keyboard debouncing.

13. List the functions performed by 8279. [CO3-L1-Nov/Dec 2014]
The various functions performed by 8279 are :
(i) Keyboard scanning (ii) Key debouncing (iii) Keycode generation
(iv) Informing the key entry to CPU (v) Storing display codes
(vi) Output display codes to LEDs (vii) Display refreshing

14. State the salient features of 8259 programmable interrupt controller. [CO3-L1]
- It can manage 8 interrupts.
- It can direct the interrupt request anywhere in the map.
- It can resolve 8 levels of interrupt priorities in various modes.
- It can mask each interrupt request individually.
- It can read the status of pending interrupts, in-service interrupts and masked interrupts.

15. What is 8259? State its functions. [CO3-L3]
The 8259 is a programmable interrupt controller can be used with the Intel Microprocessors 8085,8086 and 8088. It handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry

16. What is the use of IRR and ISR (Interrupt Request Register)?[CO3-L3]
The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

17. Name the functional blocks of 8259 . [CO3-L1]
The 8259 has eight main functional blocks.They are (1) Control logic (2) Read/Write Logic (3) Data bus buffer (4) Priority resolver (5) Cascade buffer/comparator (6) Interrupt Request Register (IRR) (7) In Service Register (ISR) and (8) Interrupt Mask Register (IMR)

18. What is the use of Cascade buffer/Comparator in 8259? [CO3-L3]
This unit is used to expand the number of interrupt levels by cascading two or more programmable interrupt controllers.

19. Explain the concept of programming the 8259A. [CO3-L2]
The 8259A accepts two types of command words generated by the CPU:

1. **Initialization Command Words (ICWs**): Before normal operation can begin, each 8259A in the system must be brought to a starting point- by a sequence of 2 to 4 bytes timed by WR pulses.
2. **Operation Command Words (OCWs**): These are the command words which command the 8259A to operate in various interrupt modes. These modes are:
   a. Fully nested mode
   b. Rotating priority mode
   c. Special mask mode
   d. Polled mode

Part-B

1. Mention the various operations performed by an I/O interface device. [CO3-L1-May/Jun 2014]

Memory Devices and Interfacing

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. Most of the peripheral devices are designed and interfaced with a CPU either to enable it to communicate with the user or an external process and to ease the circuit operations so that the microprocessor works more efficiently.

The use of peripheral integrated devices simplifies both the hardware circuits and software considerable. The following are the devices used in interfacing of Memory and General I/O devices

• 74LS138 (Decoder / Demultiplexer).
• 74LS373 / 74LS374 3-STATE Octal D-Type Transparent Latches.
• 74LS245 Octal Bus Transceiver: 3-State.

74LS138 (Decoder / Demultiplexer)

The LS138 is a high speed 1-of-8 Decoder/ Demultiplexer fabricated with the low power Schottky barrier diode process. The decoder accepts three binary weighted inputs (A0, A1, A2) and when enabled provides eight mutually exclusive active LOW Outputs (O0–O7). The LS138 can be used as an 8-output demultiplexer by using one of the active LOW Enable inputs as the data input and the other Enable inputs as strobes. The Enable inputs which are not used must be permanently tied to their appropriate active HIGH or active LOW state. These 8-bit registers feature totem-pole 3-STATE outputs designed specifically for implementing buffer registers, I/O ports, bidirectional bus drivers, and working registers. The eight latches of the 74LS373 are transparent D type latches meaning that while the enable (G) is HIGH the Q outputs will follow the data (D) inputs. The eight flip-flops of the 74LS374 are edge-triggered D-type flip flops. On the positive transition of the clock, the Q outputs will be set to the logic states that were set up at the D inputs.

Main Features

• Choice of 8 latches or 8 D-type flip-flops in a single package
• 3-STATE bus-driving outputs
• Full parallel-access for loading
• Buffered control inputs

The 74LS245 is a high-speed Si-gate CMOS device.The 74LS245 is an octal transceiver featuring non- inverting 3-state bus compatible outputs in both send and receive directions. The 74LS245 features an Output Enable (OE) input for easy cascading and a send/receive (DIR) input for direction control. OE controls the outputs so that the buses are effectively isolated. All inputs have a Schmitt-trigger action. These octal bus transceivers are designed for asynchronous two-way communication between data buses. The 74LS245 is a high-speed Si-gate CMOS device. The 74LS245 is an octal transceiver featuring non-inverting 3-state bus compatible outputs in both send and receive directions.The 74LS245 features an Output Enable (OE) input for easy cascading and a send/receive (DIR) input for direction control. OE controls the outputs so that the buses are effectively isolated. All inputs have a Schmitt-trigger action.

Memory Devices And Interfacing

The memory interfacing circuit is used to access memory quit frequently to read instruction codes and data stored in the memory. The read / write operations are monitored by control signals. Semiconductor memories are of two types. Viz. RAM (Random Access Memory) and ROM (Read Only Memory) The Semiconductor RAM's are broadly two types-static Ram and dynamic RAM

Memory structure and its requirements

The read /write memories consist of an array of registers in which each register has unique address. The size of memory is N * M.

Where N is number of register and M is the word length, in number of bits. As shown in memory chip has 12 address lines A**o–**A**11**, one chip select (CS), and two control lines, Read (RD) to enable output buffer and Write (WR) to enable the input buffer.

The internal decoder is used to decoder the address lines. Figure(b) shows the logic diagram of a typical EPROM (Erasable Programmable Read-Only Memory) with 4096 (4K) register. It has 12 address lines A**o –** A**11,** one chip select (CS)**,** one read control signal. Since EPROM does not require the (WR) signal.

EPROM (or EPROMs) is used as a program memory and RAM (or RAMs) as a data memory. When both, EPROM and RAM are used, the total address space 1 Mbytes is shared by them.

Address Decoding Techniques

- Absolute decoding
- Linear decoding
- Block decoding

Absolute Decoding:

In the absolute decoding technique the memory chip is selected only for the specified logic level on the address lines: no other logic levels can select the chip. Below figure the memory interface with absolute decoding. Two 8K EPROMs (2764) are used to provide even and odd memory banks. Control signals BHE and A**o** are use to enable output of odd and even memory banks respectively. As each memory chip has 8K memory locations, thirteen address lines are required to address each locations, independently. All remaining address lines are used to generate an unique chip select signal. This address technique is normally used in large memory systems.

Linear Decoding:

In small system hardware for the decoding logic can be eliminated by using only required number of addressing lines (not all). Other lines are simple ignored. This technique is referred as linear decoding or partial decoding. Control signals BHE and A**o** are used to enable odd and even memory banks, respectively. Figure shows the addressing of 16K RAM (6264) with linear decoding. The address line A**19** is used to select the RAM chips. When A**19** is low, chip is selected, otherwise it is disabled. The status of A**14** to A**18** does not affect the chip selection logic. This gives you multiple addresses (shadow addresses). This technique reduces the cost of decoding circuit, but it gas drawback of multiple addresses

Block Decoding:

In a microcomputer system the memory array is often consists of several blocks of memory chips. Each block of memory requires decoding circuit. To avoid separate decoding for each memory block special decoder IC is used to generate chip select signal for each block.

Static Memory Interfacing

The general procedure of static memory interfacing with 8086 as follows:

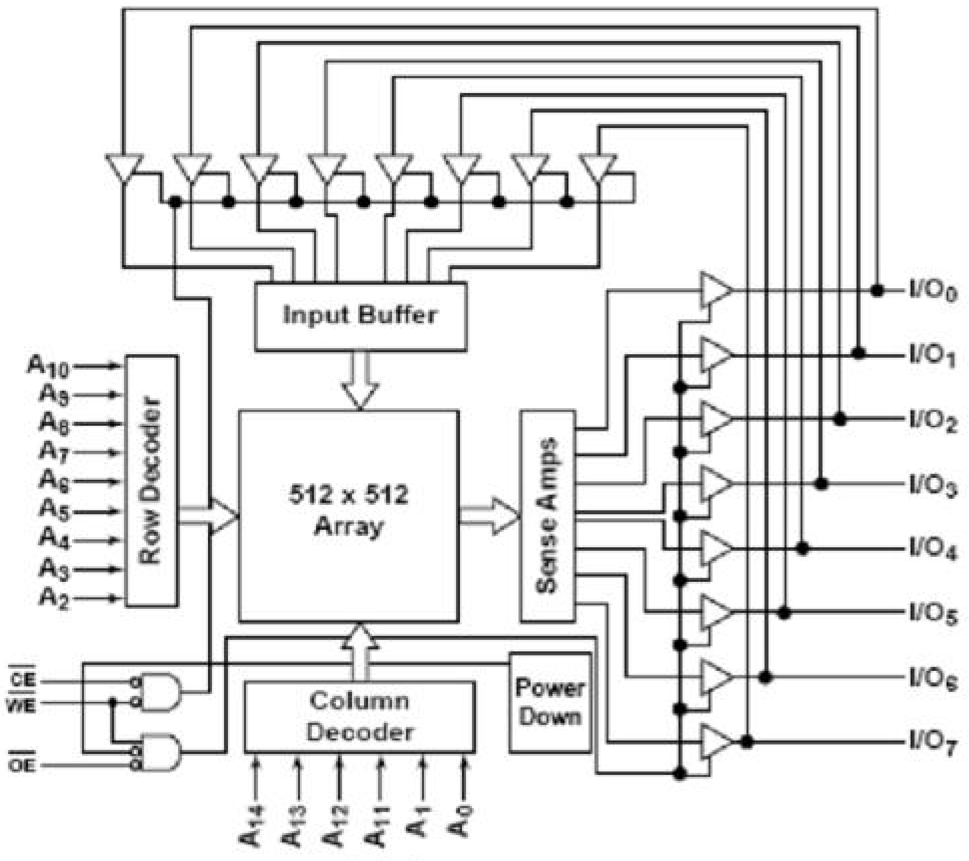
1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called ‘odd address memory bank’ and the lower 8-bit bank is called ‘even address memory bank’.
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory RD and WR inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor, BHE and Ao are used for decoding the required chip select signals for the odd and even memory banks. The CS of memory is derived from the output of the decoding circuit.
4. As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible

Dynamic RAM Interfacing

The basic Dynamic RAM cell uses a capacitor to store the charge as a representation of data. This capacitor is manufactured as a diode that is reverse-biased so that the storage capacitance comes into the picture. This storage capacitance is utilized for storing the charge representation of data but the reverse-biased diode has a leakage current that tends to discharge the capacitor giving rise to the possibility of data loss.

To avoid this possible data loss, the data stored in a dynamic RAM cell must be refreshed after a fixed time interval regularly. The process of refreshing the data in the RAM is known as refresh cycle. This activity is similar to reading the data from each cell of the memory, independent of the requirement of microprocessor, regularly. During this refresh period all other operations (accesses) related to the memory subsystem are suspended.

The advantages of dynamic RAM. Like low power consumption, higher packaging density and low cost, most of the advanced computer systems are designed using dynamic RAMs. Also the refresh mechanism and the additional hardware required makes the interfacing hardware, in case of dynamic RAM, more complicated, as compared to static RAM interfacing circuit.



Logic Block diagram

Interfacing I/O Ports

I/O ports or input/output ports are the devices through which the microprocessor communicates with other devices or external data sources/destinations. Input activity, as one may expect, is the activity that enables the microprocessor to read data from external devices, for example keyboard, joysticks, mouser etc. the devices are known as input devices as they feed data into a microprocessor system.

Output activity transfers data from the microprocessor top the external devices, for example CRT display, 7-segment displays, printer, etc, the devices that accept the data from a microprocessor system are called output devices.

Steps in Interfacing an I/O Device

The following steps are performed to interface a general I/O device with a CPU:

1. Connect the data bus of the microprocessor system with the data bus of the I/O port.
2. Derive a device address pulse by decoding the required address of the device and use it as the chip select of the device.
3. Use a suitable control signal, i.e. IORD and /or IOWR to carry out device operations, i.e. connect IORD to RD input of the device if it is an input devise, otherwise connect IOWR to WR input of the device. In some cases the RD or WR control signals are combined with the device address pulse to generate the device select pulse.

Input Port

The input device is connected to the microprocessor through buffer. The simplest form of a input port is a buffer as shown in the figure. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the device is available on the data bus. Microprocessor reads this data by initiating read command.

Output Port

It is used to send the data to the output device such as display from the microprocessor. The simplest form of the output port is a latch The output device is connected to the microprocessor through latch as shown in the figure. When microprocessor wants to send data to the output device it puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

I/O Interfacing Techniques

Input/output devices can be interfaced with microprocessor systems in two ways:

1. I/O mapped I/O
2. Memory mapped I/O

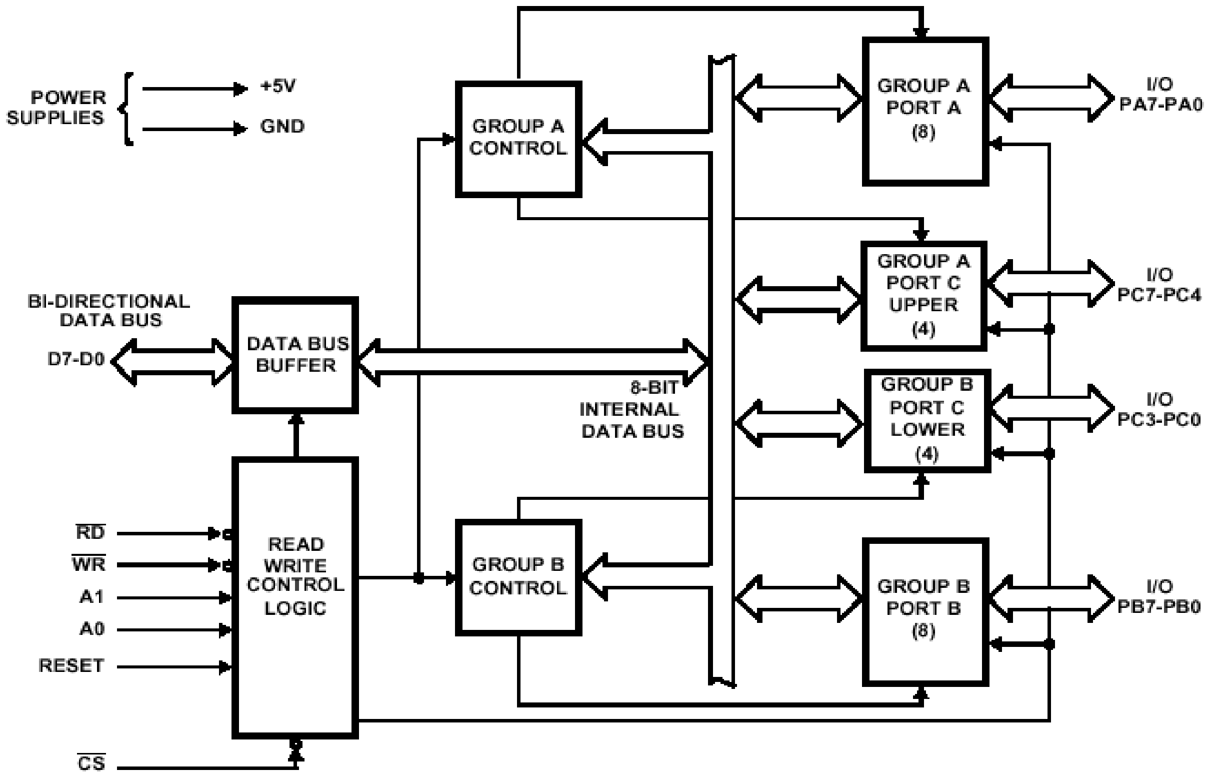
2. With neat block diagram explain the 8255 Programmable Peripheral Interface and its operating modes. [CO3-H1-Nov/Dec 2016]

Parallel Communication Interface: 8255 Programmable Peripheral Interface and Interfacing The 8255 is a widely used, programmable parallel I/O device. It can be programmed to transfer data under data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile and economical (when multiple I/O ports are required). It is an important general purpose I/O device that can be used with almost any microprocessor.

The 8255 has 24 I/O pins that can be grouped primarily into two 8 bit parallel ports: A and B, with the remaining 8 bits as Port C. The 8 bits of port C can be used as individual bits or be grouped into two 4 bit ports: CUpper (CU) and CLower (CL).

8255 can be used in two modes: Bit set/Reset (BSR) mode and I/O mode. The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into 3 modes: mode 0, mode 1 and mode 2. In mode 0, all ports function as simple I/O ports.

Mode 1 is a handshake mode whereby Port A and/or Port B use bits from Port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt. In mode 2, Port A can be set up for bidirectional data transfer using handshake signals from Port C, and Port B can be set up either in mode 0 or mode 1.



Block diagram of 8255

RD: (Read): This signal enables the Read operation. When the signal is low, microprocessor reads data from a selected I/O port of 8255.
WR: (Write): This control signal enables the write operation.
RESET (**Reset**): It clears the control registers and sets all ports in input mode. **CS,A0, A1:** These are device select signals. is connected to a decoded address and A0, A1 are connected to A0, A1 of microprocessor.

I/O Modes of 8255

Mode 0: Simple Input or Output

In this mode, Port A and Port B are used as two simple 8-bit I/O ports and Port C as two4- bit I/O ports. Each port (or half-port, in case of Port C) can be programmed to function as simply an input port or an output port. The input/output features in mode 0 are: Outputs are latched, Inputs are not latched. Ports do not have handshake or interrupt capability.

Mode 1: Input or Output with handshake

In mode 1, handshake signals are exchanged between the microprocessor and peripherals prior to data transfer. The ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports. Each port (Port A and Port B) uses 3 lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions. Input and output data are latched and Interrupt logic is supported.

STB **Strobe Input**): This signal (active low) is generated by a peripheral device that it has transmitted a byte of data. The 8255, in response to, generates IBF and INTR.

IBF (**Input buffer full):** This signal is an acknowledgement by the 8255 to indicate that the input latch has received the data byte. This is reset when the microprocessor reads the data. **INTR (Interrupt Request):** This is an output signal that may be used to interrupt the microprocessor. This signal is generated if , IBF and INTE are all at logic 1.

INTE (Interrupt Enable): This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSR mode. The INTEA is enabled or disabled through PC4, and INTEB is enabled or disabled through PC2.

(Output Buffer Full): This is an output signal that goes low when the microprocessor writes data into the output latch of the 8255. This signal indicates to an output peripheral that new data is ready to be read. It goes high again after the 8255 receives a signal from the peripheral.

(**Acknowledge):** This is an input signal from a peripheral that must output a low when the peripheral receives the data from the 8255 ports.

INTR (Interrupt Request): This is an output signal, and it is set by the rising edge of the signal. This signal can be used to interrupt the microprocessor to request the next data byte for output. The INTR is set and INTE are all one and reset by the rising edge of .

INTE (Interrupt Enable): This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSR mode. The INTEA signal can be enabled or disabled through PC6, and INTEB is enabled or disabled through PC2.

Mode 2: Bidirectional Data Transfer

OBF This mode is used primarily in applications such as data transfer between the two computers or floppy disk controller interface. Port A can be configured as the bidirectional port and Port B either in mode 0 or mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three lines from Port C can be used either as simple I/O or as handshake signals for Port B.

Serial Communication: Using 8251

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
b) Duplex
c) Half Duplex

a)Simplex mode

Data is transmitted only in one direction over a single communication channel. For example, the processor may transmit data for a CRT display unit in this mode.

b)Duplex Mode

In duplex mode, data may be transferred between two transreceivers in both directions simultaneously.

c)Half Duplex mode

In this mode, data transmission may take place in either direction, but at a time data may be transmitted only in one direction. A computer may communicate with a terminal in this mode. It is not possible to transmit data from the computer to the terminal and terminal to computer simultaneously.

Serial communication interface 8255

The data buffer interfaces the internal bus of the circuit with the system bus. The read / write control logic controls the operation of the peripheral depending upon the operations initiated by the CPU decides whether the address on internal data bus is control address / data address. The modem control unit handles the modem handshake signals to coordinate the communication between modem and USART. The transmit control unit transmits the data byte received by the data buffer from the CPU for serial communication. The transmission rate is controlled by the input frequency. Transmit control unit also derives two transmitter status signals namely TXRDY and TXEMPTY which may be used by the CPU for handshaking.

The transmit buffer is a parallel to serial converter that receives a parallel byte for conversion into a serial signal for further transmission. The receive control unit decides the receiver frequency as controlled by the RXC 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.

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,

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. This active-low input to 8251A is used to inform it that the CPU is writing data or control word to 8251A.

WR: This is an active-low chip select input of 825lA. If it is high, no read or write operation can be carried out on 8251. The data bus is tristated if this pin is high.

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. The minimum required reset pulse width is 6 clock states, for the proper reset operation.

TXC (Transmitter Clock Input): This transmitter clock input controls the rate at which the character is to be transmitted. The serial data is shifted out on the successive negative edge of the TXC.

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.

4. Explain how D/A and A/D interfacing done with 8086 with an application. [CO3-L2-Nov/Dec 2016]

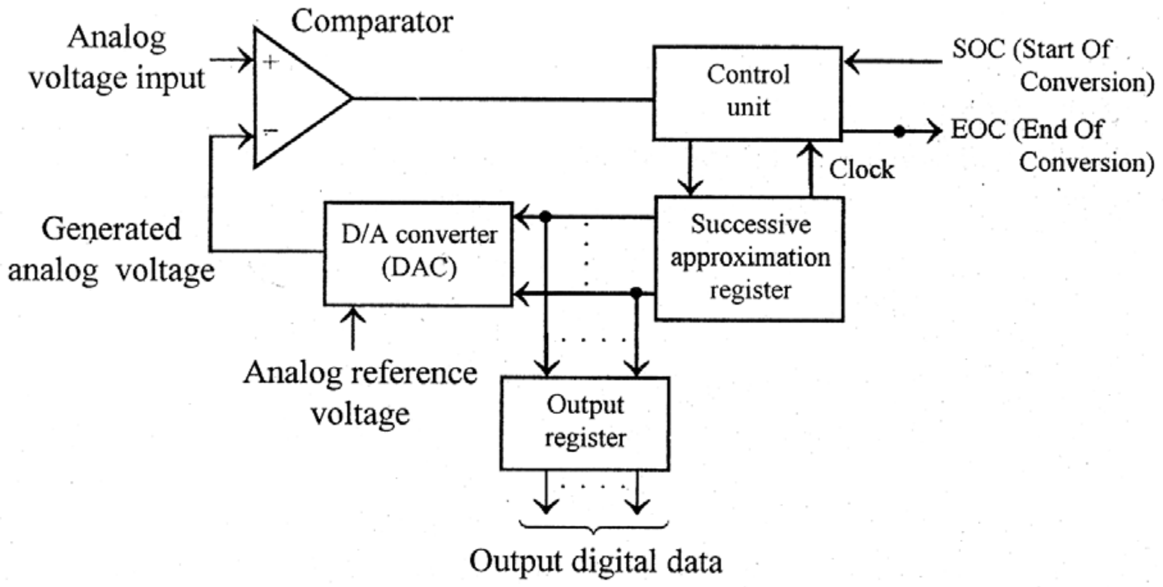


Fig 7.12 : Successive-approximation A/D converter

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. - the time it takes the converter to produce a valid output binary code for an applied input voltage. When we refer to a converter as high speed, it has a short conversion time.

The analog to digital converter is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analog to digital data conversation process.

The start of conversion signal is a pulse of a specific duration. The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over.

After the conversion is over, the ADC sends end of conversion (EOC) signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC.

These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports. The time taken by the ADC from the active edge of SOC pulse (the edge at which the conversion process actually starts) converter to calculate the equivalent digital data output from the instant of the start of conversion is called conversion delay. It may range anywhere from a few microseconds in case of fast ADCs to even a few hundred milliseconds in case of slow ADCs.

A number of ADCs are available in the market, the selection of ADC for a particular application is done, keeping in mind the required speed, resolution range of operation, power supply requirements, sample and hold device requirements and the cost factors are considered.

The available ADCs in the market use different conversion techniques for the conversion of analog signals to digital signals.

Parallel converter or flash converter, Successive approximation

Dual slope integration

A general algorithm for ADC interfacing contains the following steps.

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion (SOC) pulse to ADC.
3. Read end of conversion (EOC) signal to mark the end of conversion process.
4. Read digital data output of the ADC as equivalent digital output.

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.

This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specified time duration.

The microprocessor may issue a hold signal to the sample and Hold circuit. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

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. The conversion delay is 100 μs at a clock frequency of 640 kHz, which is quite low as compared to other converters.

These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits. These converters internally have a 3:8 analog multiplexer so that at a time eight different analog inputs can be connected to the chips. Out of these eight inputs only one can be selected for conversion by using address lines ADD A, ADD B and ADD C, as shown. Using these address inputs, multichannel data acquisition systems can be designed using a single ADC.

INTERFACING DIGITAL TO ANALOG ONVERTERS:

The digital to analog converters convert binary numbers into their analog equivalent voltages or currents. Several techniques are employed for digital to analog conversion. i. Weighted resistor network

ii. R-2R ladder network

iii. Current output D/A converter

Applications in areas like

digitally controlled gains, motor speed control, programmable gain amplifiers, digital voltmeters, panel meters, etc.

In a compact disk audio player for example a 14-or16-bit 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.

Characteristics:

1. Resolution: It is a change in analog output for one LSB change in digital input.

It is given by(1/2n)*Vref.

If n=8 (i.e.8-bit DAC) 1/256*5V=39.06mV

2. Settling time: It is the time required for the DAC to settle for a full scale code change.

DAC 0800 8-bit Digital to Analog converter

Features:

i. DAC0800 is a monolithic 8-bit DAC manufactured by National semiconductor.

ii. It has settling time around 100ms

iii. 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.

iv. Resolution of the DAC is 39.06mV

5. Explain the various modes of operation in Programmable Interval Timer 8253 [CO3-L2-Nov/Dec 2015]

Programmable timer device 8253

Intel's programmable counter/timer device (8253) facilitates the generation of accurate time delays. When 8253 is used as timing and delay generation peripheral, the microprocessor becomes free from the tasks related to the counting process and execute the programs in memory, while the timer device may perform the counting tasks. This minimizes the software overhead on the microprocessor.

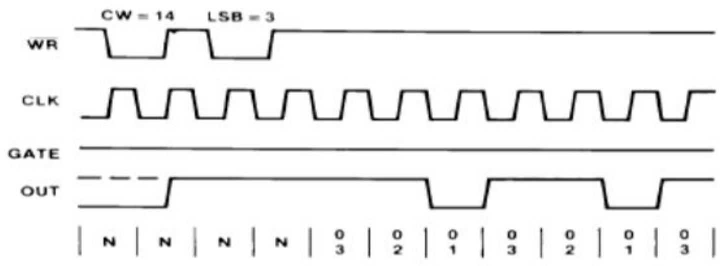
Architecture and Signal Descriptions

The programmable timer device 8253 contains three independent 16-bit counters, each with a maximum count rate of 2.6 MHz to generate three totally independent delays or maintain three independent counters simultaneously. All the three counters may be independently controlled by programming the three internal command word registers. The 8-bit, bidirectional data buffer interfaces internal circuit of 8253 to microprocessor systems bus.

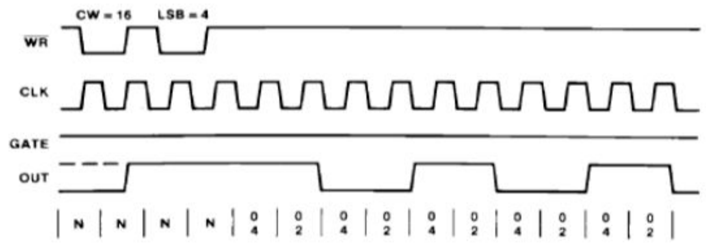
Data is transmitted or received by the buffer upon the execution of IN or OUT instruction. The read/write logic controls the direction of the data buffer depending upon whether it is a read or a write operation. It may be noted that IN instruction reads data while OUT instruction writes data to a peripheral.
The three counters all 16-bit presettable, down counters, able to operate either in BCD or in hexadecimal mode. The mode control word register contains the information that can be used for writing or reading the count value into or from the respective count register using the OUT and IN instructions. The specialty of the 8253 counters is that they can be easily read on line without disturbing the clock input to the counter. This facility is called as "on the fly" reading of counters, and is invoked using a mode control word.

Modes of Operation

- Mode 2: Continuous pulses



- Mode 3: Continuous square-wave



A0, Al pins are the address input pins and are required internally for addressing the mode control word registers and the three counter registers. A low on CS line enables the 8253.
A control word register accepts the 8-bit control word written by the microprocessor and stores it for controlling the complete operation of the specific counter. The CLK, GATE and OUT pins are available for each of the three timer channels. Their functions will be clear when we study the different operating modes of 8253.

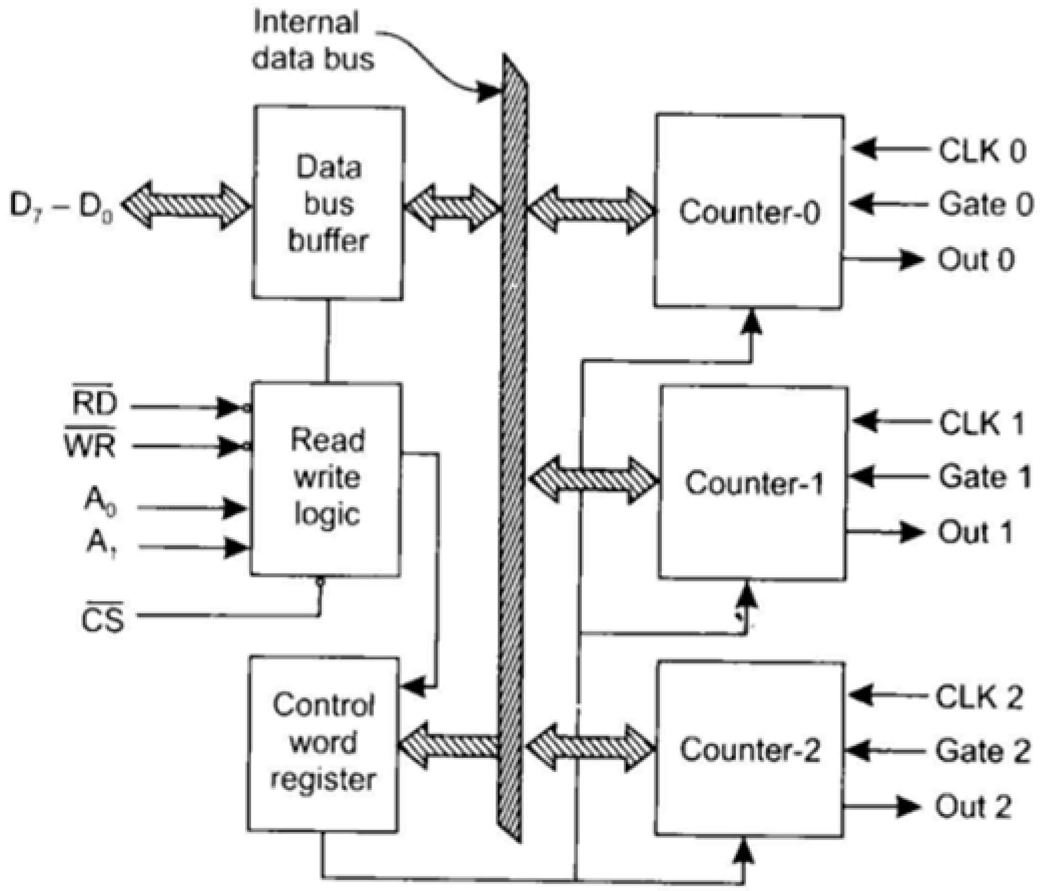


Fig. 9g.2: 8253 Functional block diagram (*Source:* Intel Corporation)

Control Word Register

The 8253 can operate in anyone of the six different modes. A control word must be written in the respective control word register by the microprocessor to initialize each of the counters of 8253 to decide its operating mode. All the counters can operate in anyone of the modes or they may be even in different modes of operation, at a time. The control word format is presented, along with the definition of each bit, while writing a count in the counter, it should be noted that, the count is written in the counter only after the data is put on the data bus and a falling edge appears at the clock pin of the peripheral thereafter. Any reading operation of the counter, before the falling edge appears may result in garbage data.

MODE 0 This mode of operation is called as interrupt on terminal count. In this mode, the output is initially low after the mode is set. The output remains low even after the count value is loaded in the counter. The counter starts decrementing the count value after the falling edge of the clock, if the GATE input is high. The process of decrementing the counter continues at each falling edge of the clock till the terminal count is reached, i.e. the count becomes zero. When the terminal count is reached, the output goes high and remains high till the selected control word register or the corresponding count register is reloaded with a new mode of operation or a new count, respectively. This high output may be used to interrupt the processor whenever required, by setting suitable terminal count. Writing a count register while the previous counting is in process, generates the following sequence of response.The first byte of the new count when loaded in the count register, stops the previous count. The second byte when written, starts the new count, terminating the previous count then and there. The GATE signal is active high and should be high for normal counting. When GATE goes low counting is terminated and the current count is latched till the GATE again goes high. Waveforms WR, OUT and GATE in Mode 0

MODE 1 This mode of operation of 8253 is called as programmable one-shot mode. the 8253 can be used as a monostable multivibrator*.* The duration of the quasistable state of the monstable multivibrator is decided by the count loaded in the count register. The gate input is used as trigger input in this mode of operation. Normally the output remains high till the suitable count is loaded in the count register and a trigger is applied. After the application of a trigger (on the positive edge), the output goes low and remains low till the count becomes zero. If another count is loaded when the output is already low, it does not disturb the previous count till a new trigger pulse is applied at the GATE input. The new counting starts after the new trigger pulse. WR, GATE and OUT Waveforms in Mode 1

MODE 2 This mode is called either rate generator or divide by N counter. In this mode, if N is loaded as the count value, then, after N pulses, the output becomes low only for one clock cycle. The count *N* is reloaded and again the output becomes high and remains high for *N* clock pulses. The output is normally high after initialisation or even a low signal on GATE input can force the output high. If GATE goes high, the counter starts counting down from the initial value. The counter generates an active low pulse at the output initially, after the count register is loaded with a count value. Then count down starts and whenever the count becomes zero another active low pulse is generated at the output. The duration of these active low pulses are equal to one clock cycle. The number of input clock pulses between the two low pulses at the output is equal to the count loaded. Figure shows the related waveforms for mode 2. Interestingly, the counting is inhibited when GATE becomes low.

MODE 3 In this mode, the 8253 can be used as a square wave rate generator. In terms of operation this mode is somewhat similar to mode 2. When, the count *N* loaded is even, then for half of the count, the output remains high and for the remaining half it remains low. If the count loaded is odd, the first clock pulse decrements it by 1 resulting in an even count value (holding the output high). Then the output remains high for half of the new count and goes low for the remaining half.

This procedure is repeated continuously resulting in the generation of a square wave. In case of odd count, the output is high for longer duration and low for shorter duration. The difference of one clock cycle duration between the two periods is due to the initial decrementing of the odd count. The waveforms for mode 3 are shown in Fig. if the loaded count value *'N* is odd, then for (N+l)/2 pulses the output remains high and for (N-l)/2 pulses it remains low.

MODE 4 This mode of operation of 8253 is named as software triggered strobe. After the mode is set, the output goes high. When a count is loaded, counting down starts. On terminal count, the output goes low for one clock cycle, and then it again goes high. This low pulse can be used as a strobe, while interfacing the microprocessor with other peripherals. The count is inhibited and the count value is latched, when the GATE signal goes low. If a new count is loaded in the count register while the previous counting is in the next clock cycle. The counting then proceeds according to the new count.

MODE 5 This mode of operation also generates a strobe in response to the rising edge at the trigger input. This mode may be used to generate a delayed strobe in response to an externally generated signal. Once this mode is programmed and the counter is loaded, the output goes high. The counter starts counting after the rising edge of the trigger input (GATE). The output goes low for one clock period, when the terminal count is reached. The output will not go low until the counter content becomes zero after the rising edge of any trigger. The GATE input in this mode is used as trigger input.

Programming and Interfacing 8253

There may be two types of write operations in 8253, viz.

(i) writing a control word into a control word register and

(ii) writing a count value into a count register.

The control word register accepts data from the data buffer and initializes the counters, as required. The control word register contents are used for (a) initialising the operating modes (mode0-mode4) (b) selection of counters (counter0-counter2) (c) choosing binary BCD counters (d) loading of the counter registers.

The mode control register is a write only register and the CPU cannot read its contents. One can directly write the mode control word for counter 2 or counter 1 prior to writing the control word for counter0. Mode control word register has a separate address, so that it can be written independently. A count register must be loaded with the count value with same byte sequence that was programmed in the mode control word of that counter, using the bits RL0 and RL1.

The loading of the count registers of different counters is again sequence independent. One can directly write the 16-bit count register for count 2 before writing count 0 and count 1, but the two bytes in a count must be written in the byte sequence programmed using RL0 and RL1 bits of the mode control word of the counter. All the counters in 8253 are down counters, hence their count values go on decrementing if the CLK input pin is applied with a valid clock signal.

A maximum count is obtained by loading all zeros into a count register, i.e. 216 for binary counting and 104 for BCD counting. The 8253 responds to the negative clock edge of the clock input. The maximum operating clock frequency of 8253 is 2.6 MHz. For higher frequencies one can use timer 8254, which operates up to 10 MHz, maintaining pin compatibility with 8253. The following Table 6.2 shows the selection of different mode control words and counter register bytes depending upon address lines Ao and A1 In 8253, the 16-bit contents of the counter can simply be read using successive 8-bit IN operations. As stated earlier, the mode control register cannot be read for any of the counters. There are two methods for reading 8253 counter registers.
In the first method, either the clock or the counting procedure (using GATE) is inhibited to ensure a stable count. Then the contents are read by selecting the suitable counter using A0, Al and executing using IN instructions. The first IN instruction reads the least significant byte and the second IN instruction reads the most significant byte. Internal logic of 8253 is designed in such a way that the programmer has to complete the reading operation as programmed by him, using RL0 and RLl bits of control word.
In the second method of reading a counter, the counter can be read while counting is in progress. This method, as already mentioned is called as reading on fly. In this method, neither clock nor the counting needs to be inhibited to read the counter. The content of a counter can be read 'on fly' using a newly defined control word register format for online reading of the count register. Writing a suitable control word, in the mode control register internally latches the contents of the counter. The control word format for 'read on fly' mode is given in Fig. 1.9 along with its bit definitions. After latching the content of a counter using this method, the programmer can read it using IN instructions, as discussed before.

6. Draw a schematic to interface keyboard and display using 8255 and explain. [CO3-H1-Nov/Dec 2014]

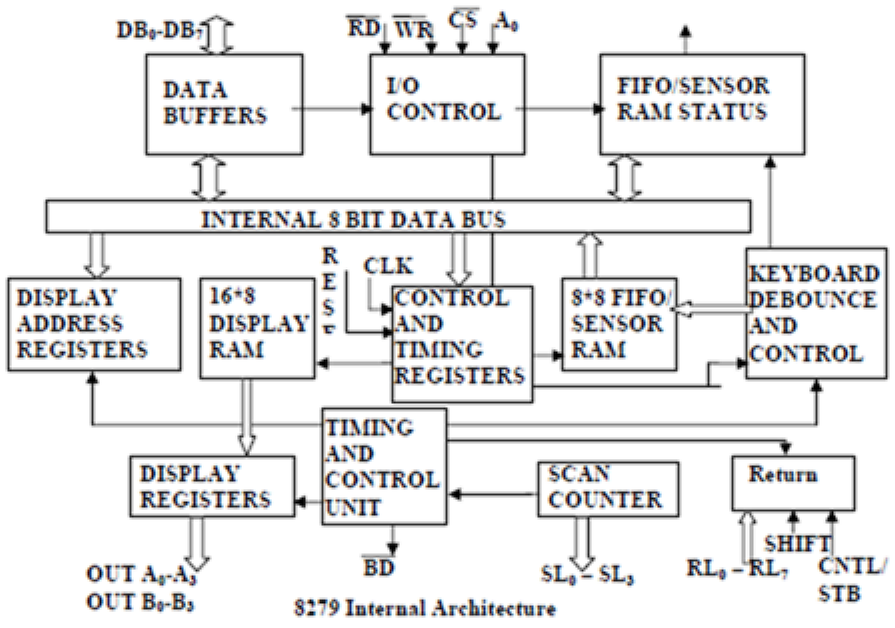
Programmable Keyboard/Display Controller

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. The Keyboard Display interface scans the Keyboard to identify if any key has been pressed and sends the code of the pressed key to the CPU. It also transmits the data received from the CPU, to the display device.

Both of these functions are performed by the controller in repetitive fashion without involving the CPU. The Keyboard is interfaced either in the interrupt or the polled mode. In the interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU can proceed with its main task.
In the polled mode, the CPU periodically reads an internal flag of 8279 to check for a key pressure. The Keyboard section can interface an array of a maximum of 64 keys with the CPU. The Keyboard entries (key codes) are de bounced and stored in an 8-byte FIFO RAM, that is further accessed by the CPU to read the key codes.

If more than eight characters are entered in the FIFO (i.e. more that eight keys are pressed), before any FIFO read operation, the overrun status is set. If a FIFO contains a valid key entry, the CPU is interrupted (in interrupt mode) or the CPU checks the status (in polling) to read the entry. Once the CPU reads a key entry, the FIFO is updated, i.e. the key entry is pushed out of the FIFO to generate space for new entries. The 8279 normally provides a maximum of sixteen 7-seg display interface with CPU It contains a 16-byte display RAM that can be used either as an integrated block of 16x8-bits or two 16x4-bit block of RAM. The data entry to RAM block is controlled by CPU using the command words of the 8279.

Architecture and Signal Descriptions of 8279



8279 Internal Architecture

The Keyboard display controller chip 8279 provides
1. A set of four scan lines and eight return lines for interfacing keyboards.
2. A set of eight output lines for interfacing display.

The pin A**o**, 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**o**=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 a binary count that is to be externally decoded to provide the scan lines for keyboard and display (four externally decoded scan lines may drive up to 16 displays). In the decoded 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

This section scans for a Key closure row-wise. If it 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 the 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 becomes empty. The status logic generates an interrupt request 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 registers hold the addresses 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. The 16-byte display RAM contains the 16-byte of data to be displayed on the sixteen 7-seg displays in the encoded scan mode.

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 timings required by 8279.

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:
8279 provides two output modes for selecting the display options.
1. Display Scan:
In this mode, 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4-bit or single 8-bit display units.
2.Display Entry:
The Display data is entered for display either from the right side or from the left side.
Details of Modes of Operation
Keyboard Modes
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 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.

5. 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 to 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. Address 0 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 Ao = 1 and CS = 0 to or from 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.

b. Programmable Clock

The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler. PPPPP is a 5-bit binary constant. The input frequency is divided by a decimal constant ranging from 2 to 31, decided by the bits of an internal prescalar, PPPPP.

c. Read FIFO/Sensor RAM

The format of this command is given as shown below

X - don't care

AI - Auto increment flag

AAA - Address pointer to 8 bit FIFO RAM

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.

d. 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 incremented 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.

e. Write Display RAM

The format of this command is given as shown below

AI - Auto increment flag.

AAAA - 4-bit address for 16-bit display RAM to be written Other details of this command are similar to the 'Read Display RAM Command.

f. Display Write Inhibit/Blanking

The IW (Inhibit write flag) bits are used to mask the individual nibble Here Do and D2 corresponds to OUTBo – OUTB3 while D**1** and D**3** corresponds to OUTAo-OUTA3 for blanking and masking respectively.

g. Clear Display RAM

The CD**2**, CD**1**, CDo is a selectable blanking code to clear all the rows of the display RAM as given below. The characters A and B represents the output nibbles. CD CD**1** CDo

1 0 x All Zeros (x don't care) AB 00

1 1 0 A**3**-Ao = 2(0010) and B**3**-Bo = 00(0000)

1 1 1 All ones (AB = FF), i.e. clear RAM

Here, CA represents clear All and CF represents Clear FIFO RAM

End Interrupt/Error Mode Set

For the sensor matrix mode, this command lowers the IRQ line and enables further writing into the RAM. Otherwise, if a charge in sensor value is detected, IRQ goes high that inhibits writing in the sensor RAM.

Key-code and status Data Formats

This briefly describes the formats of the Key-code/Sensor data in their respective modes of operation and the FIFO Status Word formats of 8279.

Key-code Data Formats:

After a valid Key closure, the key code is entered as a byte code into the FIFO RAM, in the following format, in scanned keyboard mode. The Keycode format contains 3-bit contents of the internal row counter, 3-bit contents of the column counter and status of the SHIFT and CNTL Keys The data format of the Keycode in scanned keyboard mode is given below. In the sensor matrix mode, the data from the return lines is directly entered into an appropriate row of sensor RAM, that identifies the row of the sensor that changes its status. The SHIFT and CNTL Keys are ignored in this mode. RL bits represent the return lines.

Rn represents the sensor RAM row number that is equal to the row number of the sensor array in which the status change was detected. Data Format of the sensor code in sensor matrix mode

FIFO Status Word:

The FIFO status word is used in keyboard and strobed input mode to indicate the error. Overrun error occurs, when an already full FIFO is attempted an entry, Under run error occurs when an empty FIFO read is attempted. FIFO status word also has a bit to show the unavailability of FIFO RAM because of the ongoing clearing operation.

In sensor matrix mode, a bit is reserved to show that at least one sensor closure indication is stored in the RAM, The S/E bit shows the simultaneous multiple closure error in special error mode. In sensor matrix mode, a bit is reserved to show that at least one sensor closure indication is stored in the RAM, The S/E bit shows the simultaneous multiple closure error in special error mode.

Interfacing and Programming 8279

Problem

Interface keyboard and display controller 8279 with 8086 at address 0080H. Write an ALP to set up 8279 in scanned keyboard mode with encoded scan, N-Key rollover mode. Use a 16 character display in right entry display format. Then clear the display RAM with zeros. Read the FIFO for key closure. If any key is closed, store it's code to register CL. Then write the byte 55 to all the displays, and return to DOS. The clock input to 8279 is 2MHz, operate it at 100KHz.

Solution

The 8279 is interfaced with lower byte of the data bus, i.e. Do-D**7** . Hence the Ao input of 8279 is connected with address lineA**1**.

The data register of 8279 is to be addressed as 0080H, i.e.Ao=0.

For addressing the command or status word Ao input of 8279 should be 1.

The next step is to write all the required command words for this problem.

Keyboard/Display Mode Set CW:

This command byte sets the 8279 in 16-character right entry and encoded scan N-Key rollover mode.

Program clock selection:

The clock input to 8279 is 2MHz, but the operating frequency is to be 100KHz, i.e. the clock input is to be divided by 20 (10100). Thus the prescalar value is 10100 and trhe command byte is set as given.

Clear Display RAM:

This command clears the display RAM with the programmable blanking code.

Read FIFO:

This command byte enables the programmer to read a key code from the FIFO RAM.

Write Display RAM:

This command enables the programmer to write the addressed display locations of the RAM as presented below.

7. Draw the block diagram of 8259 Programmable Interrupt Controller and explain its priority modes. [CO3-H1-May/Jun 2016]

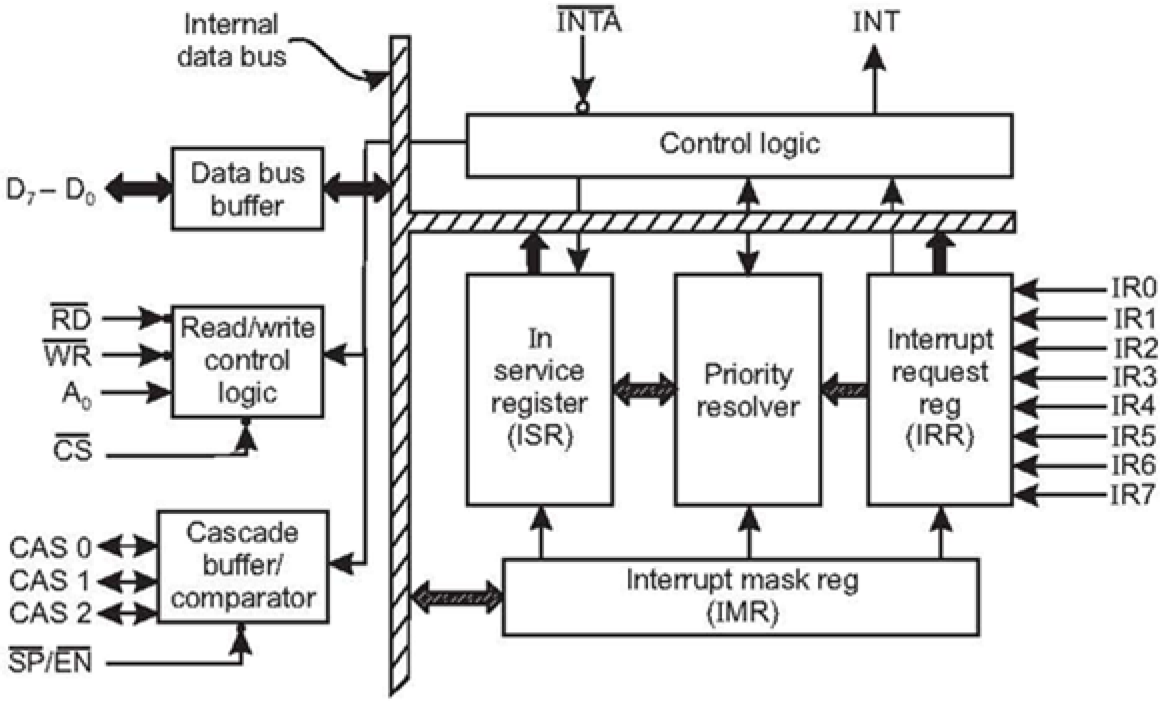


Fig. 9e.2: 8259 Functional block diagram (*Source:* Intel Corporation)

The Intel 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single a5V supply. Circuitry is static, requiring no clock input. The 8259A is designed to minimize the software and real time overhead in handling multi- level priority interrupts.

It has several modes, permitting optimization for a variety of system requirements. The 8259A is fully upward compatible with the Intel 8259. Software originally written for the259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered and Edge Triggered).

The microprocessor will be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off. This method is called Interrupt.

System throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness. The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or ``routine'' that is associated with its specific functional or operational requirements; this is referred to as a``service routine''. The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can ``point'' the Program Counter to the service routine associated with the requesting device. This ``pointer'' is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

Interrupt request register (IRR) AND in-service register (ISR):

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

Priority resolver(PR)

This logic block determines the priorites of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.

Interrupt mask register (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR.Masking of a higher priority input will not affect the interrupt request lines of lower quality.

This output goes directly to the CPU interrupt input. The VOH level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

INTA (INTERRUPT ACKNOWLEDGE)

INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (mPM) of the 8259A.

Data bus buffer:

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

CS (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless thedevice is selected.

WR (WRITE)

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

RD (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR),In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

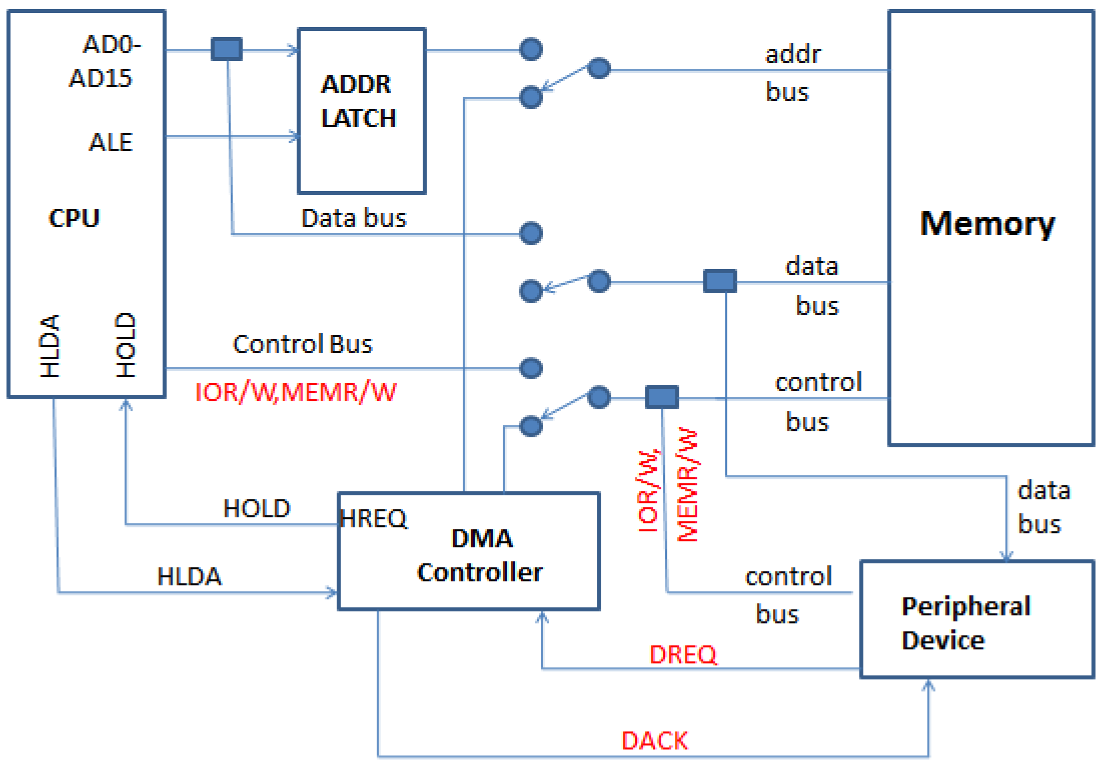
Interrupt sequence

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows:

1. One or more of the INTERRUPT REQUEST lines (IR7±0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an INTA pulse.
4. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7±0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.
8. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.

8. What is DMA? Explain the DMA based data transfer using DMA controller. [CO3-L2]



The *Direct Memory Access* or DMA mode of data transfer is the fastest amongst 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 internal architecture of 8257 is shown in figure. 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 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. Table shows how the Ao-A3 bits may be used for selecting one of these registers.

DMA Address Register

Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. 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 issued 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.

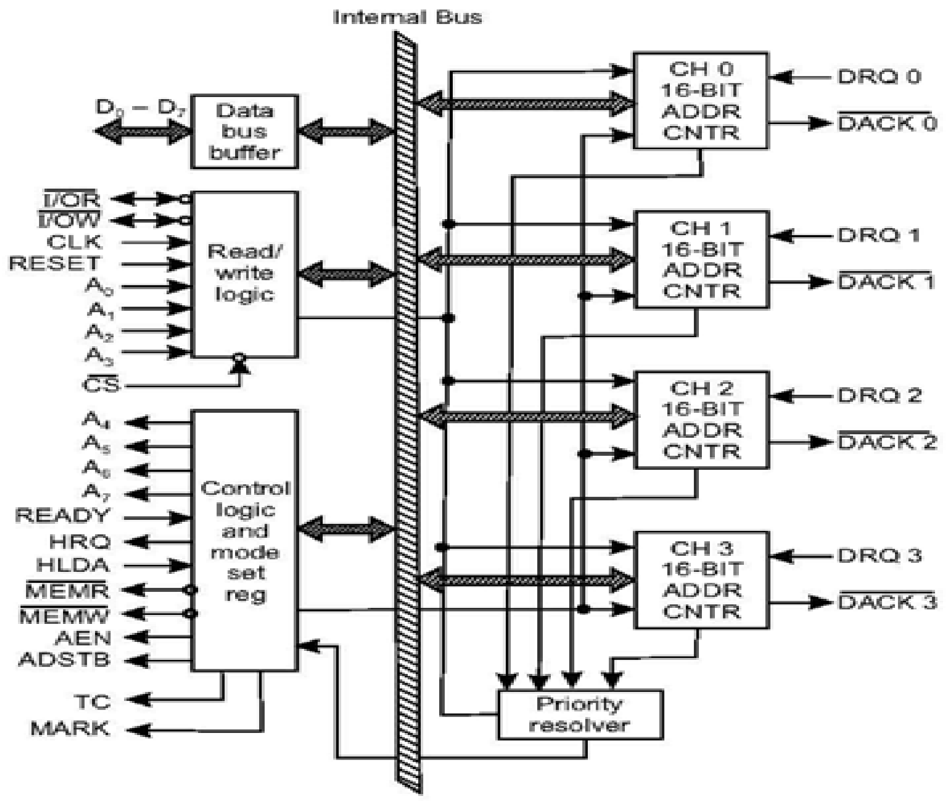


Fig. 9f.2: 8257 Functional block diagram (*Source:* Intel Corporation)

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. Bit 14 of the register in this case will be set to one and bit 15 will be set to zero.

Mode Set Register

The mode set register is used for programming the 8257 as per the requirements of the system. 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 Do-D3 enable o n e of the four DMA channels of 8257. for example, if Do is '1', channel 0 is enabled. If bit 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 different access times.

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. These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by resetting 8257 or by resetting the auto load bit of the mode set register. 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. The update flag is set every time; the channel 2 registers are loaded with contents of the channel 3 registers. It is cleared by the completion of the first DMA cycle of the new block. This register can only read.

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.

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 DRQo has the highest priority while DRQ**3** 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-bit DMA 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 select 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 he 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 non cascaded 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 read data from the addressed memory locations during DMA read cycles.
MEMW:
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.

Unit-IV

Microcontroller

Part-A

1. **What is a microcontroller? [CO4-H1-Nov/Dec2012]**

A microcontroller is a programmable semiconductor device available as IC and capable of performing arithmetic and logical operations. Microcontrollers are used for designing application specific dedicated systems.

2. **What are the basic units of a microcontroller? [CO4-L1-Nov/Dec2014]**

The basic units of a microcontroller are the ALU, a set of registers, IO ports, memory, timing and control unit. In addition some of the controllers may have timers, ADC and DAC.

3. **Mention few differences between a microprocessor and a microcontroller. [CO4-L2-Nov/Dec2015]**

A microprocessor is concerned with the rapid movement of code and data between the external memory and the processor, where as a microcontroller is concerned with the rapid movement of code and data within the controller.
A microprocessor will have few bit manipulating instructions, whereas a microcontroller will have a large number of bit manipulating instructions.
Microprocessors are generally used for designing general purpose systems, whereas microcontrollers are used for designing dedicated application specific systems.

4. **List the features of 8051 microcontroller.[CO4-L1-Nov/Dec2013]**

8 bit CPU
4KB of ROM (program memory).
128 bytes of RAM (Data memory).
Four number of 8-bit parallel ports.
64 KB address space for external data memory.
64 KB address space for program memory.
Two 16-bit timer /counter.
Five source interrupt structure.

5. **What are dedicated address pointers in an 8051?[CO4-H1-May/Jun 2012]**

The 8051 has dedicated address pointers : Program Counter (PC) and Data Pointer (DPTR). The PC is used as an address pointer for programs and DPTR is used as an address pointer for data.

6. **What are Special Function Registers (SFR)? [CO4-L2-Nov/Dec2012]**

Special Function Registers are internal registers of a microcontroller dedicated for specific functions. These registers can be used only for their specified / defined functions and cannot be used for any other function. In microcontrollers, the SFR are mapped as internal data memory and can be accessed by direct addressing.

7. What are register banks in an 8051? [CO4-L2-Nov/Dec2014]

The register banks are internal RAM locations of 8051 which can be used as general purpose registers or scratch pad registers. The first 32 bytes of internal RAM of 8051 are organized as four register banks with each bank consisting of eight locations. At any one time, the processor can work with only one register bank depending on the value of bits RS0 and RS1 in the PSW register.

8. What is PSW in an 8051? [CO4-L2-Nov/Dec2012]

The flag register of an 8051 is called PSW (Program Status Word). The PSW consists of four math flags and two register bank select bits. The math flags are carry, auxillary carry, overflow and parity flag. The register bank select bits are RS0 and RS1.

9. How is stack implemented in an 8051? [CO4-H1-May/Jun 2015]

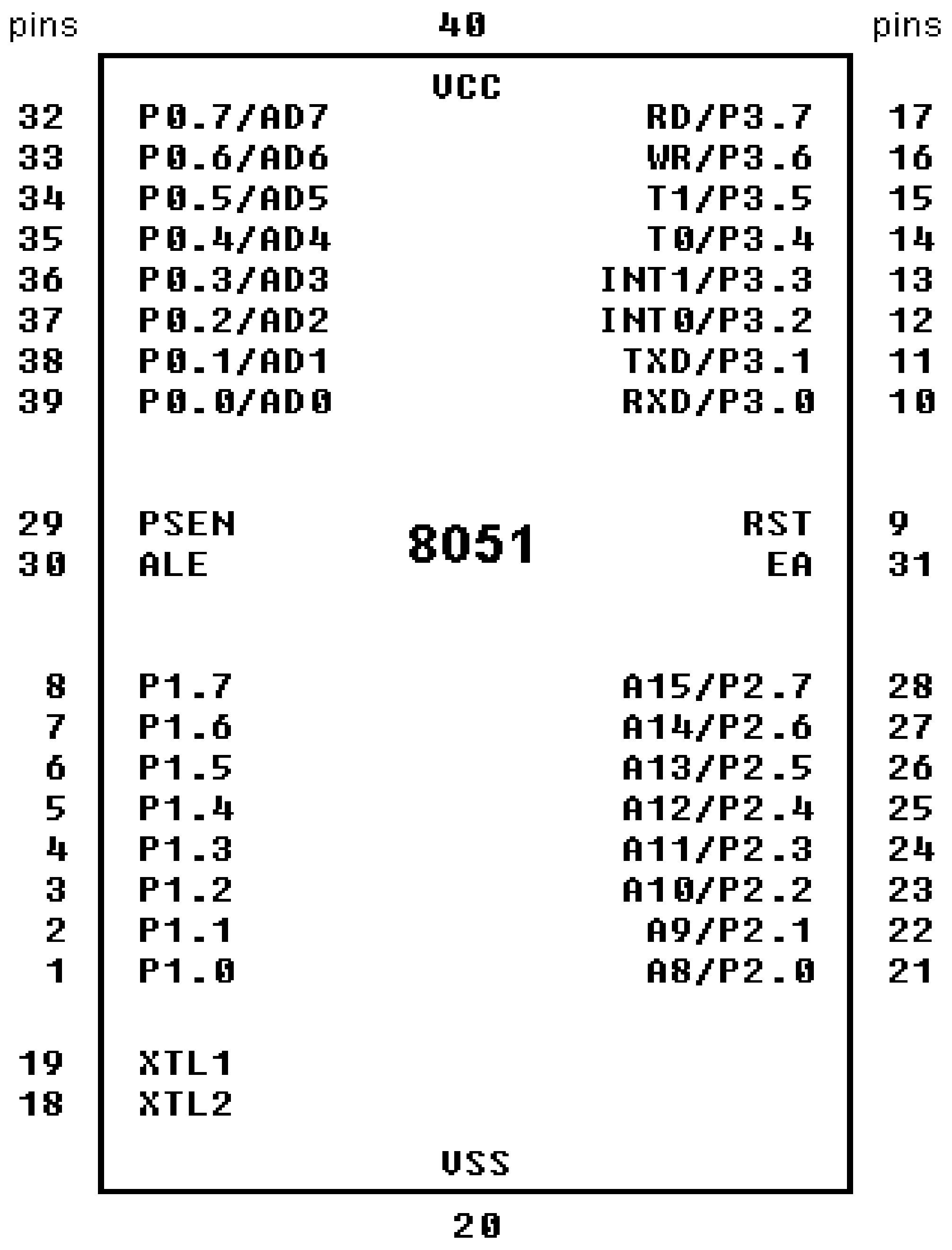
The 8051 supports a LIFO (Last-In-First-Out) stack and the stack can reside anywhere in the internal RAM. The 8051 has an 8-bit Stack Pointer (SP) to indicate the top of stack. The stack can be accessed using PUSH and POP instructions. During PUSH, the SP is automatically incremented by one and during POP, the SP is automatically decremented by one.

Mode 3 – Timer -0 can alone work in mode 3 and in this mode the TL 0 will function as 8-bit timer controlled by standard timer-0 control bits and TH0 will function as an 8-bit timer controlled by timer-1 controlled bits.

i. External data memory write cycle
ii. Port operation cycle

Part-B

1. Describe the Pin configuration of 8051 with neat diagram. [CO4-L1-Nov/Dec 2014]



The diagram above shows the 8051 pinout. The chip is a 40-pin package.

Port 0 - pins 32 to 39 make up the 8-bit I/O port 0. However, if external memory is used, these lines are used as a multiplexed address and data bus.

Port 1 - pins 1 to 8 make up the 8-bit I/O port 1.

Port 2 - pins 21 to 28 make up the 8-bit I/O port 2. However, if external memory is used, these lines make up the high-byte of the external address (A8 to A15).

Port 3 - pins 10 to 17 make up the 8-bit I/O port 3. However, each of these eight pins also has an alternate function, as detailed in the table below.

Pin	**Name**	**Bit Address**	**Function**
P3.0	RXD	B0H	Receive data for serial port
P3.1	TXD	B1H	Transmit data for serial port
P3.2	INT0-bar	B2H	External interrupt 0
P3.3	INT1-bar	B3H	External interrupt 1
P3.4	T0	B4H	Timer/counter 0 external input
P3.5	T1	B5H	Timer/counter 1 external input
P3.6	WR-bar	B6H	External data memory write strobe
P3.7	RD-bar	B7H	External data memory read strobe

RST - the reset input is on pin 9. This pin is used for resetting the 8051 (ie; loading the PC with the correct startup value).

EA-bar - the external access, on pin 31, is used for enabling or disabling the on-chip ROM. When tied high (5V), the 8051 executes instructions in internal ROM when executing in the lower 4K (8K for the 8052) of memory. If tied low the 8051 will always execute instructions in external memory. The 8031 and 8032 should always have pin 31 tied low as there is no internal code memory.

ALE - the address latch enable is on pin 30. The ALE is used for latching the low byte of the address into an external register. We will deal with this at a later date.

2. Explain in detail the various addressing modes in 8051. [CO4-L2-Nov/Dec 2016]

Addressing Modes

The eight addressing modes are:

- Immediate
- Register
- Direct
- Indirect
- Relative
- Absolute
- Long
- Indexed

Immediate Addressing

If the operand is a constant then it can be stored in memory immediately after the opcode. Remember, values in code memory (ROM) do not change once the system has been programmed and is in use in the everyday world. Therefore, immediate addressing is only of use when the data to be read is a constant. For example, if your program needed to perform some calculations based on the number of weeks in the year, you could use immediate addressing to load the number 52 (34H) into a register and then perform arithmetic operations upon this data.

MOV R0, #34

The above instruction is an example of immediate addressing. It moves the data 34H into R0. The assembler must be able to tell the difference between an address and a piece of data. The has symbol (#) is used for this purpose (whenever the assembler sees # before a number it knows this is immediate addressing).

This is a two-byte instruction.

Register Addressing

Often we need to move data from a register into the accumulator so that we can perform arithmetic operations upon it. For example, we may wish to move the contents of R5 into the accumulator.

MOV A, R5

This is an example of register addressing. It moves data from R5 (in the currently selected register bank) into the accumulator.

ADD A, R6

The above is another example of register addressing. It adds the contents of R6 to the accumulator, storing the result in the accumulator. Note that in both examples the destination comes first. This is true of all instructions.

Direct Addressing

Direct addressing is used for accessing data in the on-chip RAM. Since there are 256 bytes of RAM (128 bytes general storage for the programmer and another 128 bytes for the SFRs). That means the addresses go from 00H to FFH, any of which can be stored in an 8-bit location.

```
MOV A, 67
```

The above instruction moves the data in location 67H into the accumulator. Note the difference between this and immediate addressing. Immediate addressing uses the data, which is immediately after the instruction. With direct addressing, the operand is an address. The data to be operated upon is stored in that address. The assembler realises this is an address and not data because there is no hash symbol before it.

```
ADD A, 06
```

The above instruction adds the contents of location 06H to the accumulator and stores the result in the accumulator. If the selected register bank is bank 0 then this instruction is the same as ADD A, R6.

Indirect Addressing

Register addressing and direct addressing both restrict the programmer to manipulation of data in fixed addresses. The address the instruction reads from (MOV A, 30H) or writes to (MOV 30H, A) cannot be altered while the program is running.

There are times when it is necessary to read and write to a number of contiguous memory locations. For example, if you had an array of 8-bit numbers stored in memory, starting at address 30H, you may wish to examine the contents of each number in the array (perhaps to find the smallest number). To do so, you would need to read location 30H, then 31H, then 32H and so on.

This can be achieved using indirect addressing. R0 and R1 may be used as pointer registers. We can use either one to store the current memory location and then use the indirect addressing instruction shown below.

```
MOV A, @Ri
```

where *Ri* is either R0 or R1.

Now, we can read the contents of location 30H through indirect addressing:

```
MOVR0,#30H
MOV A, @R0
```

The first instruction is an example of immediate addressing whereby the data 30H is placed in R0. The second instruction is indirect addressing. It moves the contents of location 30H into the accumulator.

If we now wish to get the data in location 31H we use the following:

```
INC R0
MOV A, @R0
```

Once we see how to write a loop in assembly language, we will be able to read the entire contents of the array.

Relative Addressing

Relative addressing is used only with certain jump instructions. The system executes a jump by changing the contents of the PC to the address of the next instruction to be executed. For example, if we wished to jump to the instruction stored at location 4EH in code memory, the PC would be loaded with 4EH. Then, during the next execution cycle the contents of the PC (4EH) are placed on the address bus and the instruction at 4EH is retrieved.

A relative address (or offset) is an 8-bit signed value, which is added to the PC to form the address of the next instruction to be executed.

With 8-bit signed numbers, the MSB is used to determine whether the number is positive or negative. If the MSB is 0 then the number is positive, while if the MSB is 1 the number is negative.

The instruction below shows how to jump six locations ahead.

SJMP 06H

SJUMP is an unconditional jump and is a 2-byte instruction. The number following it is an offset address. If this instruction were stored in code memory at locations 100H and 101H, as shown below:

*100H***80H**
101H **06H**

The opcode for *SJMP* is 80H. The operand is the offset address. If this instruction were executed the PC would get the value 108H. This is what happens:

- The PC contains 100H, therefore the instruction 80H is read into the IR.
- The instruction is decoded as the 2-byte SJMP instruction.
- The PC is incremented so that the operand may be retrieved.
- The operand is read from code memory and the PC is incremented again
- The operand (06H) is added to the PC (102H + 06H = 108H).
- The next instruction (at 108H) is executed.

Once we deal with 2's compliment and how negative numbers are dealt with in the CPU, we will look at a backward jump.

The *S* in *SJMP* stands for *short*. The range of signed 8-bit numbers is -127 to 128. Therefore, using SJMP allows us to jump 127 locations forward or 128 locations backward. Hence the name short jumps.

Absolute Addressing

Absolute addressing is only used with the ACALL and AJMP instructions.

ACALL - subroutine call (2 byte instruction)

AJMP - unconditional jump (2 byte instruction)

These instructions allow you to move to any instruction within the same 2K of memory.

We will look at the AJMP instruction only (at a later date, when we begin dealing with subroutines we will deal with the ACALL instruction).

The operation of the AJMP instruction is detailed below:

AJMPaddress(PC)<-(PC)+2
(PC_{10}-PC_0) <- $address_{10}$ - $address_0$

Note that only the eleven least significant bits of the PC are altered. The five most significant bits remain the same. This means the AJMP will only allow you to jump to a location in the same 2K page as the instruction directly after the jump.

3. Explain the Block Diagram of 8051.[CO4-L2-Nov/Dec 2016]
Features of 8051

1. 4096 bytes on – chip program memory
2. 128 bytes on – chip data memory
3. Four register banks
4. 128 User – defined software flags.
5. 64 Kilobytes each program and external RAM addressability.
6. One microsecond instruction cycle with 12 MHz crystal.
7. 32 bidirectional I/O lines organized as four 8 – bit ports (16 lines on 8031)
8. Multiple mode, high – speed programmable serial port.
9. Two multiple mode, 16 – bit Timers / Counters.
10. Two level prioritized interrupt structure.
11. Full depth stack for subroutine return linkage and data storage.
12. Direct Byte and Bit addressability.

13. Binary or Decimal arithmetic.

14. Signed – overflow detection and parity computation.

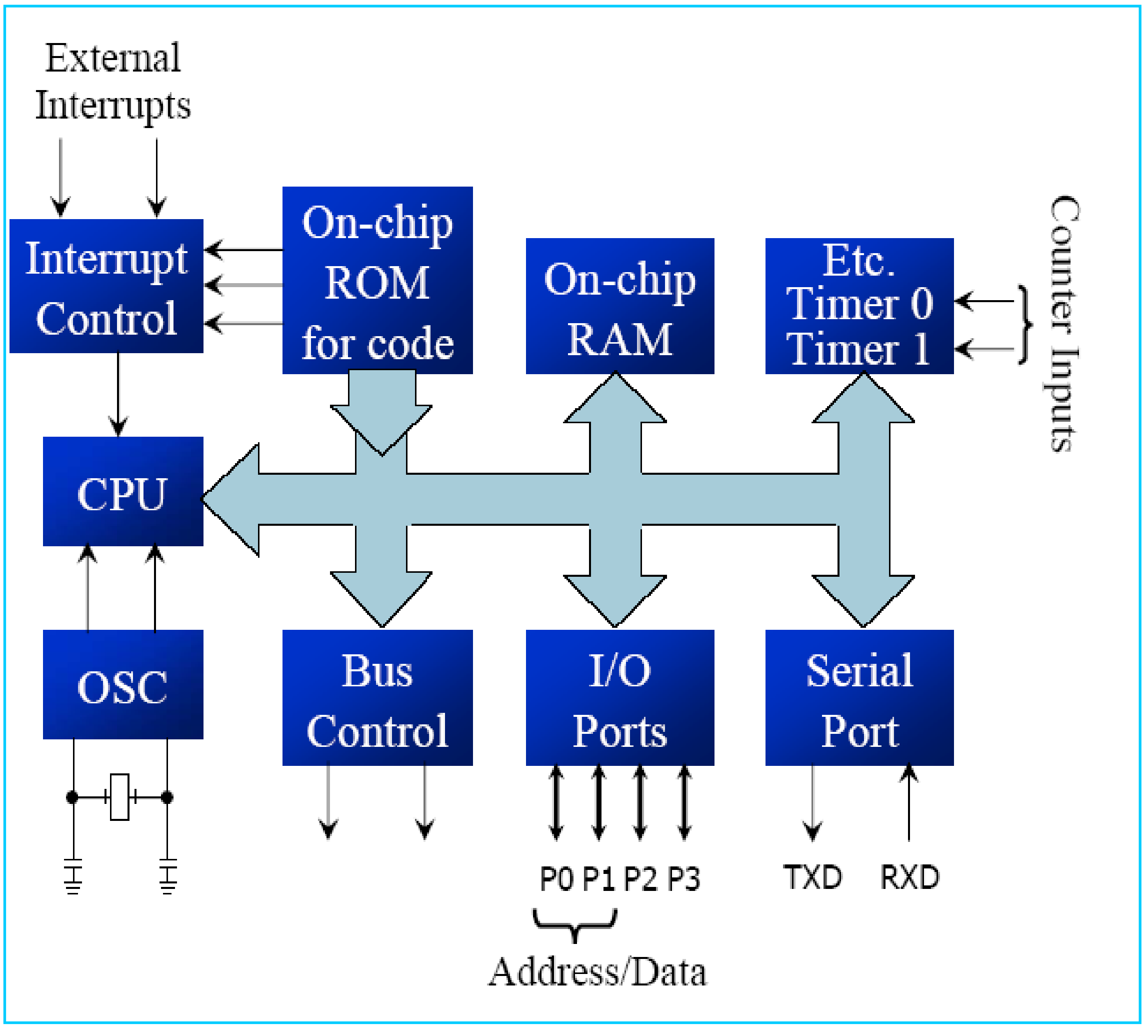
15. Hardware Multiple and Divide in 4 µsec.

16. Integrated Boolean Processor for control applications.

17. Upwardly compatible with existing 8084 software.

THE FUNCTIONAL BLOCK DIAGRAM OF 8051

The shows the internal block diagram of 8051. It consists of a CPU, two kinds of memory sections (data memory – RAM and program memory – EPROM / ROM), input / output ports, special function registers and control logic needed for a variety of peripheral functions. These elements communicate through an eight bit data bus which runs throughout the chip referred as internal data bus. This bus buffered to the outside world through an I/O port when memory or I/O expansions is desired.



Central Processing Unit (CPU)

The CPU of 8051 consists of eight-bit Arithmetic and Logic unit with associated registers like A,B, PSW , SP, the sixteen bit program counter and "Data pointer" (DPTR) registers. Along with these registers it has a set of special function registers. Along, with these registers it has a set of special function registers.

The 8051's ALU can perform arithmetic and logic functions on eight bit variables. The arithmetic unit can perform addition, subtraction, multiplication and division. The logic unit can perform logical operations such as AND, OR, and Exclusive OR, as well as rotate, clear, and complement. The ALU also looks after the branching decisions. An important and unique feature of the 8051 architecture is that the ALU can also manipulate one bit as well as eight-bit data types. Individual bits may be set, cleared, complemented, moved, tested and used in logic computation.

Internal RAM

The 8051 has 128-byte internal RAM. It is accessed using RAM address register. The figure shows the organization of internal RAM. As shown in the figure, internal RAM of 8051 is organized into three distinct areas:

Working registers

Bit Addressable

General Purpose

1. First thirty-two bytes from address 00H to 1FH of internal RAM constitute 32 working registers. They are organized into four banks of eight registers each. The four register banks are numbered 0 to 3 and are consists of eight registers named R0 to R7. Each register can be addressed by the name or by its RAM address. Only one register bank is in use at a time. Bits RS0 and RS1 in the PSW determine which bank of registers is currently in use. Register banks when not selected can be used as general purpose RAM. On reset, the Bank 0 is selected

.

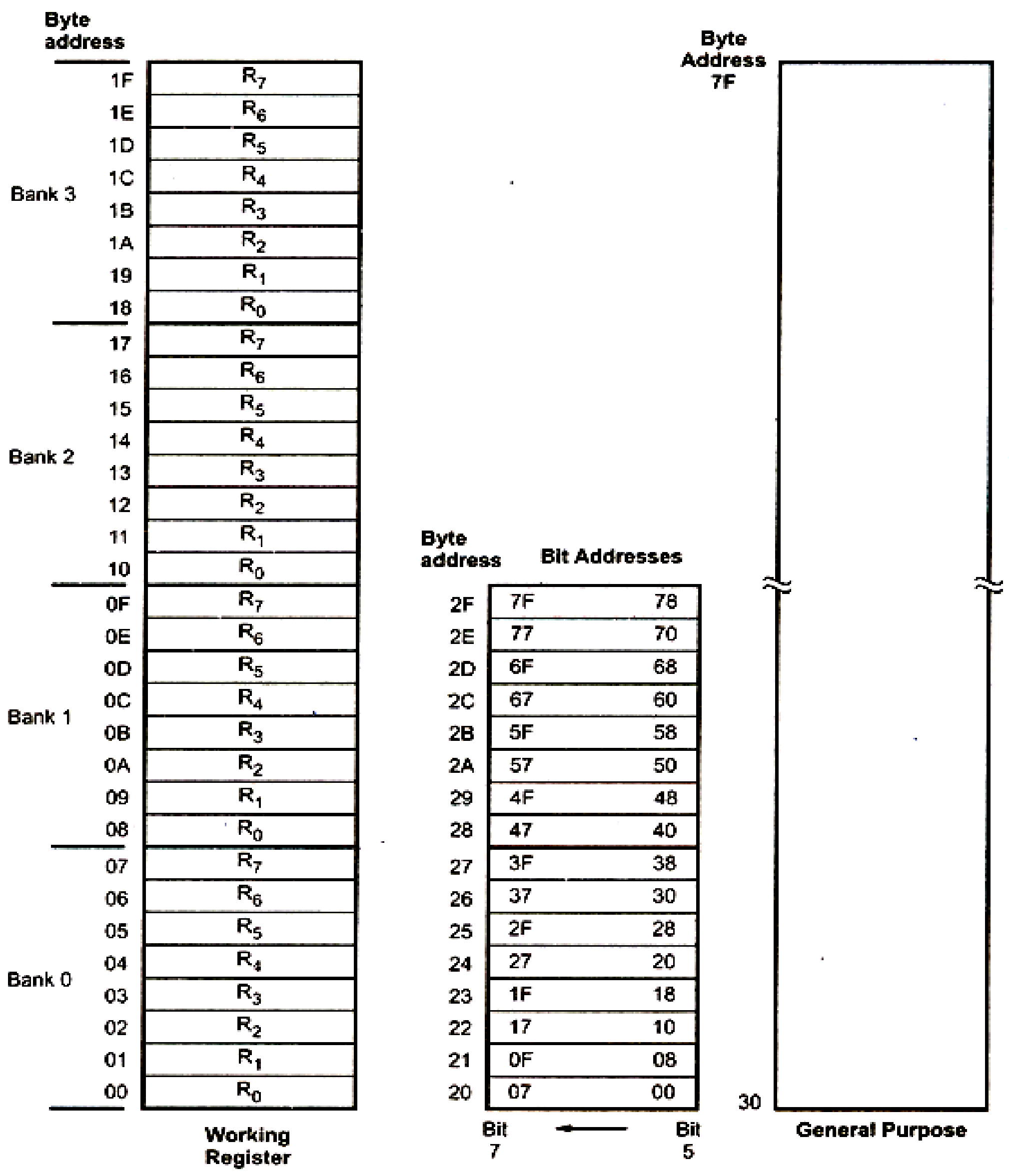


Figure: Organization of Internal RAM of 8051

2. The 8051 provides 16 bytes of a bit-addressable area. It occupies RAM byte addresses from 20H to 2FH, forming a total of 128 (16 × 8) addressable bits. An addressable bit may be specified by its bit address of 00H to 7FH, or bits may form any byte address from 20H to 2FH. For example, bit address 4EH refers bit 6 of the byte address 29H.

3. The RAM area above bit addressable area from 30H to 7FH is called general purpose RAM. It is addressable as byte

Internal ROM

The 8051 has 4K byte of internal ROM with address spaced from 0000H to 0FFFH. It is programmed by manufacturer when the chip is built. This part cannot be erased or altered after fabrication. This is used to store final version of the program.

It is accessed using program address register. The program address higher then 0FFFH, which exceed the internal ROM capacity, will cause the 8051 to automatically fetch code bytes from external program memory. However, code bytes can also be fetched exclusively from an external memory addresses 0000H to FFFFH, by connecting the external access pin (EA) to ground.

Input/Output Ports

The 8051 has 32 I/O pins configured as four eight-bit parallel ports (P0, P1,P2 and P3). All four ports are bidirectional, i.e., each pin will be configured as input or output (or both) under software control. Each port consists of a latch, an output driver, and an input buffer. The output drives of Ports 0 and 2 and the input buffers of Port 0, are used to access external memory. As mentioned earlier, Port 0 outputs the low order byte of the external memory address, time multiplexed with the data being written or read, and Port 2 outputs the high order byte of the external memory address when the address is 16 bits wide. Otherwise Port 2 gives the contents of special function register P2. All ports pins of Port 3 are multifunctional. They have special functions as shown below including two external interrupts, two counter inputs, two special data lines and two timing control strobes.

Symbol	Position	Name and Significance
$\overline{RD}$	P3.7	Active low pulse generated by hardware when external data memory is read.
$\overline{WR}$	P3.6	Active low pulse generated by hardware when external data memory is written.
T1	P3.5	Timer/counter 1 external input or test pin
T0	P3.4	Timer/counter 0 external input or test pin
$\overline{INT1}$	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.
$\overline{INT0}$	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.
TXD	P3.1	Transmit Data pin for serial port. Clock output in shift register.
RXD	P3.0	Receive Data pin for serial port. Data I/O pin in shift register mode.

Register Set of 8051

Register A (Accumulator)

It is an 8-bit register. It holds a source operand and receives the result of the arithmetic instructions (addition, substraction, multiplication and division). The accumulator can be the source or destination for logical operation and a number of special data movement instructions, including look-up tables and external RAM expansion. Several functions apply exclusively to the accumulator : rotate, parity computation, testing for zero, and so on.

Register B

In addition to accumulator, an 8-bit B-register is available as a general purpose register when it is not being used for the hardware multiply/divide operation.

4. Explain about the 8051 Instructions set. [CO4-L2-May/Jun 2015]
8051 Instructions
8051 has about 111 instructions. These can be grouped into the following categories

1. Arithmetic Instructions
2. Logical Instructions
3. Data Transfer instructions
4. Boolean Variable Instructions
5. Program Branching Instructions

The following nomenclatures for register, data, address and variables are used while write instructions.

A: Accumulator B: "B" register C: Carry bit
Rn: Register R0 - R7 of the currently selected register bank
Direct: 8-bit internal direct address for data. The data could be in lower 128bytesof RAM (00 - 7FH) or it could be in the special function register (80 - FFH).
@Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.
#data8: Immediate 8-bit data available in the instruction.
#data16: Immediate 16-bit data available in the instruction.
Addr11: 11-bit destination address for short absolute jump.
Used by instructions AJMP & ACALL. Jump range is 2 kbyte (one page).
Addr16: 16-bit destination address for long call or long jump.
Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps.
bit: Directly addressed bit in internal RAM or SFR

Arithmetic Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ADD A, Rn	A + Rn	1	1
ADD A, direct	A + (direct)	2	1
ADD A, @Ri	A + @Ri	1	1
ADD A, #data	A + data	2	1
ADDC A, Rn	A + Rn + C	1	1
ADDC A, direct	AA + (direct) + C	2	1
ADDC A, @Ri	A + @Ri + C	1	1
ADDC A, #data	A + data + C	2	1
DA A	Decimal adjust accumulator	1	1
DIV AB	Divide A by B A quotient B remainder	1	4
DEC Rn	Rn Rn - 1	1	1
DEC direct	(direct) (direct) - 1	2	1
DEC @Ri	@Ri @Ri - 1	1	1
INC A	A+1	1	1
INC Rn	Rn Rn + 1	1	1
INC direct	(direct) (direct) + 1	2	1
INC @Ri	@Ri @Ri +1	1	1
INC DPTR	DPTR DPTR +1	1	2
MUL AB	Multiply A by B A low byte (A*B) 4 B high byte (A* B)	1	
SUBB A, Rn	A A - Rn - C	1	1
SUBB A, direct	A A - (direct) - C	2	1
SUBB A, @Ri	A A - @Ri - C	1	1
SUBB A, #data	A A - data - C	2	1

☐Logical Instructions

Mnemonics	**Description**	**Bytes**	**Instruction Cycles**
ANL A, Rn	A A AND Rn	1	1
ANL A, direct	A A AND (direct)	2	1
ANL A, @Ri	A AND @Ri	1	1
ANL A, #data	A A AND data	2	1
ANL direct, A	(direct) (direct) AND A	2	1
ANL direct, #data	(direct) (direct) AND data	3	2
CLR A	A 00H	1	1
CPL A	A A	1	1
ORL A, Rn	A A OR Rn	1	1
ORL A, direct	A A OR (direct)	1	1
ORL A, @Ri	A A OR @Ri	2	1
ORL A, #data	A A OR data	1	1

ORL direct, A	(direct) (direct) OR A	2	1
ORL direct, #data	(direct) (direct) OR data	3	2
RL A	Rotat accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within	1	1
XRL A, Rn	A A EXOR Rn	1	1
XRL A, direct	A A EXOR (direct)	1	1
XRL A, @Ri	A A EXOR @Ri	2	1
XRL A, #data	A A EXOR data	1	1
XRL direct, A	(direct) (direct) EXOR A	2	1
XRL direct, #data	(direct) (direct) EXOR data	3	2

POP direct	(direct) (SP)	2	2
XCH Rn	Exchange A with Rn	1	1
XCH direct	Exchange A with direct byte	2	1
XCH @Ri	Exchange A with indirect RAM	1	1
XCHD A, @Ri	Exchange least significant nibble of A with that of indirect RAM	1	1

Boolean Variable Instructions

Mnemonics	Description	Bytes	Instruction Cycles
CLR C	C-bit 0	1	1
CLR bit	bit 0	2	1
SET C	C 1	1	1
SET bit	bit 1	2	1
CPL C	C	1	1
CPL bit	Bit	2	1
ANL C, /bit	C C	2	1
ANL C, bit	C C.	2	1
ORL C, /bit	C C	2	1
ORL C, bit	C C + bit	2	1
MOV C, bit	C bit	2	1
MOV bit, C	bit C	2	2

Program Branching Instructions

Mnemonics	Descriptio n	Byte s	Instruction Cycles
ACALL addr11	PC + 2 (SP); addr 11 PC	2	2
AJMP addr11	Addr11 PC	2	2
CJNE A, direct, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE A, #data, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE Rn, #data, rel	Compare with Rn, jump (PC + rel) if not equal	3	2
CJNE @Ri, #data, rel	Compare with @Ri A, jump (PC + rel) if not equal	3	2
DJNZ Rn, rel	Decrement Rn, jump if not zero	2	2
DJNZ direct, rel	Decrement (direct), jump if not zero	3	2
JC rel	Jump (PC + rel) if C bit = 1	2	2
JNC rel	Jump (PC + rel) if C bit = 0	2	2
JB bit, rel	Jump (PC + rel) if bit = 1	3	2
JNB bit, rel	Jump (PC + rel) if bit = 0	3	2
JBC bit, rel	Jump (PC + rel) if bit = 1	3	2
JMP @A+DPTR	A+DPTR PC	1	2
JZ rel	If A=0, jump to PC + rel	2	2
JNZ rel	If A ≠ 0 , jump to PC + rel	2	2
LCALL addr16	PC + 3 (SP), addr16 PC	3	2
LJMP addr 16	Addr16 PC	3	2
NOP	No operation	1	1
RET	(SP) PC	1	2
RETI	(SP) PC, Enable Interrupt	1	2
SJMP rel	PC + 2 + rel PC	2	2
JMP @A+DPTR	A+DPTR PC	1	2
JZ rel	If A = 0. jump PC+ rel	2	2
JNZ rel	If A ≠ 0, jump PC + rel	2	2
NOP	No operation	1	1

Unit-V

<u>Interfacing Microcontroller</u>

<u>Part-A</u>

1. What are the operation modes of the serial port of an 8051? [CO5-H1-May/Jun 2013]
The operating modes of the serial port of an 8051 are mode-0, mode-1 , mode-2 and mode-3

2. What are the operatingmodes of the timer of an 8051? [CO5-H1-Nov/Dec 2011]
The operating modes of the timers of an 8051 are mode-0, mode-1, mode-2 and mode-3.
Mode 0 – Timer will function as a 13-timer
Mode 1 – Timer will function as 16-bit timer
Mode 2 – Timers will function as 8-bit timers with auto reload feature.
Mode 3 – Timer -0 can alone work in mode 3 and in this mode the TL 0 will function as 8-bit timer controlled by standard timer-0 control bits and TH0 will function as an 8-bit timer controlled by timer-1 controlled bits.

3. How many machine cycles are needed to execute an instruction in an 8051 microcontroller? [CO5-L3]
The 8051 microcontroller executes an instruction in one to four machine cycles.

4. List the various machine cycles of 8051 controller. [CO5-L1-May/Jun 2015]
External program memory fetch cycle
External data memory ready cycle
External data memory write cycle
Port operation cycle

5. What are the addressing modes available in 8051 microcontroller? [CO5-L1-May/Jun 2014]
The addressing modes available in 8051 microcontroller are:

i. Register addressing
ii. Direct addressing
iii. Register -Indirect addressing
iv. Immediate addressing
v. Relative addressing

6. How can the 8051 instructions be classified? [CO5-L2]
The 8051 instructions can be classified into the following five groups:

i. Data transfer instructions
ii. Arithmetic instructions
iii. Logical instructions
iv. Branching instructions
v. Boolean instructions

7. List the instructions of 8051 that affect the overflow flag in 8051. [CO5-L1-Nov/Dec 2015]
The 8051 instructions that affects the overflow flag are ADD, ADDC, MUL,DIV and SUBB.

8. List the instructions of 8051 that always clear the carry flag. [CO5-L1-Nov/Dec 2016]
The 8051 instructions that always clear the carry flag are CLR C, DIV and MUL.

9. What are the operations performed by the Boolean variable instructions of an 8051? [CO5-H1]
The Boolean variable instructions can clear or complement or move a particular bit of bit-addressable RAM/SFR or carry flag. They can also transfer the program control to a new address if a particular bit set or cleared.

10. How serial communication takes place in 8051 controller? [CO5-H3-Nov/Dec 2015]
The 8051 microcontroller has an internal serial port which can be operated in four modes. The baud rates for serial communication are programmable using internal Timer-1 of the 8051 microcontroller.

11. What are the signals required to interface a memory chip to the microprocessor? [CO5-H1-May/Jun 2013]
To interface a memory chip to the microprocessor, the following signals are required to be generated :

Address bus signals
Data bus signals
Memory chip select signals
Read Control signal
Write Control signal (only in case of RAM)

12. What is the importance of TMOD and TCON?[CO5-H3-May/Jun 2011]
The special function registers TMOD and TCON are used to control the timer/counter functions.

13. What is PCON? [CO5-L2]
PCON – Power Control Regiser. It is a special function register through which certain power control functions in CMOS version of the 8051 are implemented.

14. What is the size of the on-chip program memory and on-chip data memory of 8051 microcontroller? [CO5-H1- Nov/Dec 2013]
The size of the on-chip program memory in 8051 microcontroller is 64 KB
The size of the on-chip data memory in 8051 microcontroller is 4KB

Part-B

1. How are the timers of 8051 used to produce time delay in timer mode? [CO5-L3-Nov/Dec 2012]

Programming 8051 Timers: Using Timers to Measure Time One of the primary uses of timers is to measure time.

When a timer is in interval timer mode (as opposed to event counter mode) and correctly configured, it will increment by 1 every machine cycle. A single machine cycle consists of 12 crystal pulses. Thus a running timer will be incremented:11,059,000 / 12 = 921,583 times per second.

Unlike instructions which require 1 machine cycle, others 2, and others 4--the timers are consistent: They will always be incremented once per machine cycle. Thus if a timer has counted from 0 to 50,000 you may calculate:

50,000 / 921,583 = .0542.0542 seconds have passed. To execute an event once per second you'd have to wait for the timer to count from 0 to 50,000 18.45times.

To calculate how many times the timer will be incremented in .05 seconds, a simple multiplication can be done: 0 .05 * 921,583 = 46,079.15.

This tells us that it will take .05 seconds (1/20th of a second) to count from 0 to 46.0. To work with timers is to control the timers and initialize them.

The TMOD SFR

TMOD (Timer Mode): The TMOD SFR is used to control the mode of operation of both timers. Each bit of the SFR gives the microcontroller specific information concerning how to run a timer. The high four bits (bits 4 through 7) relate to Timer 1whereas the low four bits (bits 0 through 3) perform the exact same functions, but for timer 0. The modes of operation are:

modes of Timer

TxM1	TxM0	Timer Mode	Description of Mode
0	0	0	13-bit Timer.
0	1	1	16-bit Timer
1	0	2	8-bit auto-reload
1	1	3	13-bit Time Mode (mode 0)

Timer mode "0" is a 13-bit timer. When the timer is in 13-bit mode, TLx will count from 0 to 31. When TLx is incremented from 31, it will "reset" to 0 and increment THx. Thus, effectively, only 13 bits of the two timer bytes are being used: bits 0-4 of TLx and bits 0-7 of THx. The timer can only contain 8192 values. If you set a 13-bit timer to 0, it will overflow back to zero 8192 machine cycles later.

16-bit Time Mode (mode 1)

Timer mode "1" is a 16-bit timer. TLx is incremented from 0 to 255. When TLx is incremented from 255, it resets to 0 and causes THx to be incremented by 1. Since this is a full 16-bit timer, the timer may contain up to 65536 distinct values. If you set a 16-bit timer to 0, it will overflow back to 0 after 65,536 machine cycles.

8-bit Time Mode (mode 2)

Timer mode "2" is an 8-bit auto-reload mode.

When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored in THx. For example, if TH0 holds the value FDh and TL0 holds the value FEh values of TH0 and TL0 for a few machine cycles:

The value of TH0 never changed. When we use mode 2 you almost always set THx to a known value and TLxis the SFR that is constantly incremented. The benefit of auto-reload mode is the timer always have a value from 200 to 255. If you use mode 0 or 1, you'd have to check in code to see if the timer had overflowed and, if so, reset the timer to 200. This takes precious instructions of execution time to check the value and/or to reload it. When

mode 2 operation

Machine Cycle	TH0 Value	TL0 Value
1	FDh	FEh
2	FDh	FFh
3	FDh	FDh
4	FDh	FEh
5	FDh	FFh
6	FDh	FDh
7	FDh	FEh

you use mode 2 the microcontroller takes care of this. Auto-reload mode is very commonly used for establishing a baud rate in Serial Communications.

Split Timer Mode (mode 3)

Timer mode "3" is a split-timer mode. When Timer 0 is placed in mode 3, it essentially becomes two separate 8-bit timers. Timer 0 is TL0 and Timer 1 is TH0. Both timers count from 0 to 255 and overflow back to 0. All the bits that are related to Timer 1 will now be tied to TH0. While Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be put into modes 0, 1 or 2 normally--however, you may not start or stop the real timer 1 since the bits that do that are now linked to TH0. The real timer 1,e, will be incremented every machine cycle always. The only real use in split timer mode is if you need to have two separate timers and, additionally, a baud rate generator you can use the real Timer 1 as a baud rate generator and use TH0/TL0 as two separate timers.

Reading the Timer

There are two common ways of reading the value of a 16-bit timer; which you use depends on your specific application. You may either read the actual value of the timer as a 16-bit number, or you may simply detect when the timer has overflowed.

Reading the value of a Timer

If timer is in an 8-bit mode either 8-bit Auto Reload mode or in split timer mode, you simply read the 1-byte value of the timer. With a 13-bit or16-bit timer the timer value was 14/255 (High byte 14, low byte 255) but you read 15/255.Because you read the low byte as 255. But when you executed the next instruction a small amount of time passed--but enough for the timer to increment again at which time the value rolled over from 14/255 to 15/0. But in the process you've read the timer as being 15/255. You read the high byte of the timer, then read the low byte, then read the high byte again. If the high byte read the second time is not the same as the high byte read the first time you repeat the cycle. In code, this would appear as:

```
REPEAT: MOV A, TH0
MOV R0, TL0
CJNE A, TH0, REPEAT
```

In this case, we load the accumulator with the high byte of Timer 0. We then load R0 with the low byte of Timer 0. Finally, we check to see if the high byte we read out of Timer 0--which is now stored in the Accumulator--is the same as the current Timer 0 high byte. We do by going back to REPEAT. When the loop exits we will have the low byte of the timer in R0 and the high byte in the Accumulator.

Another much simpler alternative is to simply turn off the timer run bit (i.e. CLR TR0), read the timer value, and then turn on the timer run bit (i.e. SETB TR0).

Detecting Timer Overflow

Whenever a timer *overflows* from its highest value back to 0, the microcontroller automatically sets the TFx bit in the TCON register. if TF1 is set it means that timer 1 has overflowed. We can use this approach to cause the program to execute a fixed delay. it takes the 8051 1/20thof a second to count from 0 to 46,079. However, the TFx flag is set when the timer overflows back to 0. Thus, if we want to use the TFx flag to indicate when 1/20th of a second has passed we must set the timer initially to 65536 less 46079, or 19,457. If we set the timer to 19,457, 1/20th of a second later the timer will overflow.

The following code to execute a pause of 1/20th of a second: MOV TH0,#76;High byte of 19,457 (76 * 256 = 19,456) MOV TL0,#01;Low byte of 19,457 (19,456 + 1 = 19,457) MOV TMOD,#01;Put Timer 0 in 16-bit mode

SETB TR0; Make Timer 0 start counting

JNB TF0,$;If TF0 is not set, jump back to this same instruction

In the above code the first two lines initialize the Timer 0 starting value to 19,457. The next two instructions configure timer 0 and turn it on. Finally, the last instruction **JNB TF0, $**, reads "Jump, if TF0 is not set, back to this same instruction." The "$" operand means, in most assemblers, the address of the current instruction.

2. Describe the serial interface with 8051 microcontroller. [CO5-L1-May/Jun 2013]

Serial Port Programming: 8051 Serial Communication

One of the 8051's many powerful features -integrated *UART*, known as a serial port to easily read and write values to the serial port instead of turning on and off one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits and parity bits.

Setting the Serial Port Mode configures it by specifying 8051 how many data bits we want, the baud rate we will be using and how the baud rate will be determined. First, let's present the "Serial Control" (SCON) SFR and define what each bit of the SFR represents:

Definition of SCON SFR

Bit	Name	Bit Addres	Explanation of Function
7	SM0	9Fh	Serial port mode bit 0
6	SM1	9Eh	Serial port mode bit 1.
5	SM2	9Dh	Mutli processor Communications Enable
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive Characters.
3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
2	RB8	9AH	Receive bit 8. The 9th bit received in mode 2 and 3.
1	T1	99h	Transmit Flag. Set when a byte has been completely Transmitted.
0	RI	98h	Receive Flag. Set when a byte has been completely Received.

Additionally, it is necessary to define the function of SM0 and SM1 by an additional table: Table 5.4 SCON as serial Port

Modes of SCON

SM0	SM1	Serial Mode	Explanation Baud Rate
0	0	0	0 8-bit Shift Register Oscillator / 12
0	1	1	8-bit UART Set by Timer 1 (*)
1	0	2	9-bit UART Oscillator / 32 (*)
1	1	3	9-bit UART Set by Timer 1 (*)

The SCON SFR allows us to configure the Serial Port. The first four bits (bits 4 through 7) are configuration bits:

Bits **SM0** and **SM1** is to set the *serial mode* to a value between 0 and 3, inclusive as in table above selecting the Serial Mode selects the mode of operation (8-bit/9-bit, UART or Shift Register) and also determines how the baud rate will be calculated. In modes 0 and 2 the baud rate is fixed based on the oscillator's frequency. In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows.
The next bit, **SM2**, is a flag for " Multiprocessor communication whenever a byte has been received the 8051 will set the "RI" (Receive Interrupt) flag to let the program know that a byte has been received and that it needs to be processed.
However, when SM2 is set the "RI" flag will only be triggered if the 9th bit received was a "1". if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set .You will almost always want to clear this bit so that the flag is set upon reception of *any* character.
The next bit, **REN**, is "Receiver Enable." is set indicate to data received via the serial port.
The last four bits (bits 0 through 3) are operational bits. They are used when actually sending and receiving data--they are not used to configure the serial port.
The **TB8** bit is used in modes 2 and 3. In modes 2 and 3, a total of nine data bits are transmitted. The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken from TB8. If TB8 is set and a value is written to the serial port, the data's bits will be written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear."
The **RB8** also operates in modes 2 and 3and functions essentially the same way as TB8, but on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. In this case, the first eight bits received are the data of the serial byte received and the value of the ninth bit received will be placed in RB8.**TI** means "Transmit Interrupt."
When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port. If the program were to write another byte to the serial port before the first byte was completely output, the data being sent would be garbled. Thus, the8051 lets the program know that it has "clocked out" the last byte by setting the TI bit.
When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte. Finally, the **RI** bit means "Receive Interrupt." It functions similarly to the "TI" bit, but it indicates that a byte has been received. Whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value quickly, before another byte is read.

Setting the Serial Port Baud Rate

Once the Serial Port Mode has been configured, the program must configure the serial port's baud rate. This only applies to Serial Port modes 1 and 3. 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 1.059 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 a 11.059Mhz crystal speed will yield a baud rate of172, 797.

In modes 1 and 3, the baud rate is determined by how frequently timer 1 overflows. The more frequently timer 1 overflows, the higher the baud rate. There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode2) and set a reload value (TH1) that causes Timer 1 to overflow at a frequency appropriate to generate a baud rate.

To determine the value that must be placed in TH1 to generate a given baud rate, (assuming PCON.7 is clear).

TH1 = 256 - ((Crystal / 384) / Baud)

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

TH1 = 256 - ((Crystal / 192) / Baud)

For example, if we have an 11.059 Mhz crystal and we want to configure the serial port to 19,200 baud we try plugging it in the first equation: TH1 = 256 - ((Crystal / 384) / Baud) TH1 = 256 - ((11059000 / 384) / 19200) TH1 = 256 - ((28,799) / 19200) TH1 = 256 - 1.5 = 254.5 To obtain 19,200 baud on a 11.059Mhz crystal we'd have to set TH1 to 254.5. If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud.

To achieve 19,200 baud we simply need to set PCON.7 (SMOD). When we do this we double the baud rate and utilize the second equation mentioned above. Thus we have:

TH1 = 256 - ((Crystal / 192) / Baud) TH1 = 256 - ((11059000 / 192) / 19200) TH1 = 256 - ((57699) / 19200) TH1 = 256 - 3 = 253 Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

1) Configure Serial Port mode 1 or 3.
2) Configure Timer 1 to timer mode 2 (8-bit auto reload).
3) Set TH1 to 253 to reflect the correct frequency for 19,200 baud.
4) Set PCON.7 (SMOD) to double the baud rate.

Writing to the Serial Port

Once the Serial Port has been properly configured as explained above, the serial port is ready to be used to send data and receive data. To write a byte to the serial write the value to the **SBUF** (99h) SFR. For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as:

```
MOV SBUF, #'A'
```

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Obviously transmission is not instantaneous--it takes a measureable amount of time to transmit. And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.

The 8051 lets us know when it is done transmitting a character by setting the **TI** bit in SCON. When this bit is set the last character has been transmitted and that send the next character, if any. Consider the following code segment:

```
MOV SBUF, #'A'; Send the letter 'A' to the serial port
JNB TI,$;Pause until the RI bit is set.
```

The above three instructions will successfully transmit a character and wait for the TI bit to be set before continuing. The last instruction says "Jump if the TI bit is not set to $"— $, in most assemblers, means "the same address of the current instruction." Thus the 8051 will pause on the JNB instruction until the TI bit is set by the 8051 upon successful transmission of the character.

Reading the Serial Port

Reading data received by the serial port is equally easy. To read a byte from the serial port one just needs to read the value stored in the **SBUF** (99h) SFR after the 8051 has automatically set the **RI** flag in SCON.

For example, if your program wants to wait for a character to be received and subsequently read it into the Accumulator, the following code segment may be used:

```
JNB RI,$;Wait for the 8051 to set the RI flag
MOV A,SBUF; Read the character from the serial port
```

The first line of the above code segment waits for the 8051 to set the RI flag; again, the8051 sets the RI flag automatically when it receives a character via the serial port. So as long as the bit is not set the program repeats the "JNB" instruction continuously. Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction which reads the value.

3. Explain the interrupt structure of 8051 microcontroller. [CO5-L2-May/Jun 2014]

Interrupt Programming:

The following events will cause an interrupt:

Timer 0 Overflow.

Timer 1 Overflow.

Reception/Transmission of Serial Character.

External Event 0.

External Event 1.

To distinguish between various interrupts and executing different code depending on what interrupt was triggered 8051may be jumping to a fixed address when a given interrupt occurs.

Interrupt handling

Interrupt	**Flag**	**Interrupt Handler Addres**
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001Bh
Serial	RI/TI	0023h

If Timer 0 overflows (i.e., the TF0 bit is set), the main program will be temporarily suspended and control will jump to 000BH if we have code at address 0003H that handles the situation of Timer 0 overflowing.

Setting Up Interrupts

By default at power up, all interrupts are disabled. Even if, for example, the TF0 bit is set, the 8051 will not execute the interrupt. Your program must specifically tell the 8051 that it wishes to enable interrupts and specifically which interrupts it wishes to enable. Your program may enable and disable interrupts by modifying the IE SFR (A8h):

Interrrupt and address

Bit	Name Bit	Address	Explanation of Function
7	EA	AFh	Global Interrupt Enable/Disabl
6		AEh	Undefined
5		ADh	Undefined
4	ES	ACh	Enable Serial Interrupt
3	ET1	ABh	Enable Timer 1 Interrupt
2	EX1	AAh	Enable External 1 Interrupt
1	ET0	A9h	Enable Timer 0 Interrupt
0	EX0	A8h	Enable External 0 Interrupt

Each of the 8051'sinterrupts has its own bit in the IE SFR. You enable a given interrupt by setting the corresponding bit. For example, if you wish to enable Timer 1 Interrupt, you would execute either:

MOV IE,#08h **||** SETB ET1

Both of the above instructions set bit 3 of IE, thus enabling Timer 1 Interrupt. Once Timer 1 Interrupt is enabled, whenever the TF1 bit is set, the 8051 will automatically put "on hold" the main program and execute the Timer 1 Interrupt Handler at address 001Bh. However, before Timer 1 Interrupt (or any other interrupt) is truly enabled, you must also set bit 7 of IE. Bit 7, the Global Interrupt Enable/Disable, enables or disables all interrupts simultaneously. That is to say, if bit 7 is cleared then no interrupts will occur, even if all the other bits of IE are set. Setting bit 7 will enable all the interrupts that have been selected by setting other bits in IE. This is useful in program execution if you have time-critical code that needs to execute. In this case, you may need the code to execute from start to finish without any interrupt getting in the way. To accomplish this you can simply clear bit 7 of IE (CLR EA) and then set it after your time critical code is done.

To enable the Timer 1 Interrupt execute the following two instructions: SETB ET1

SETB EA

Thereafter, the Timer 1 Interrupt Handler at 01Bh will automatically be called whenever the TF1 bit is set (upon Timer 1 overflow).

Polling Sequence

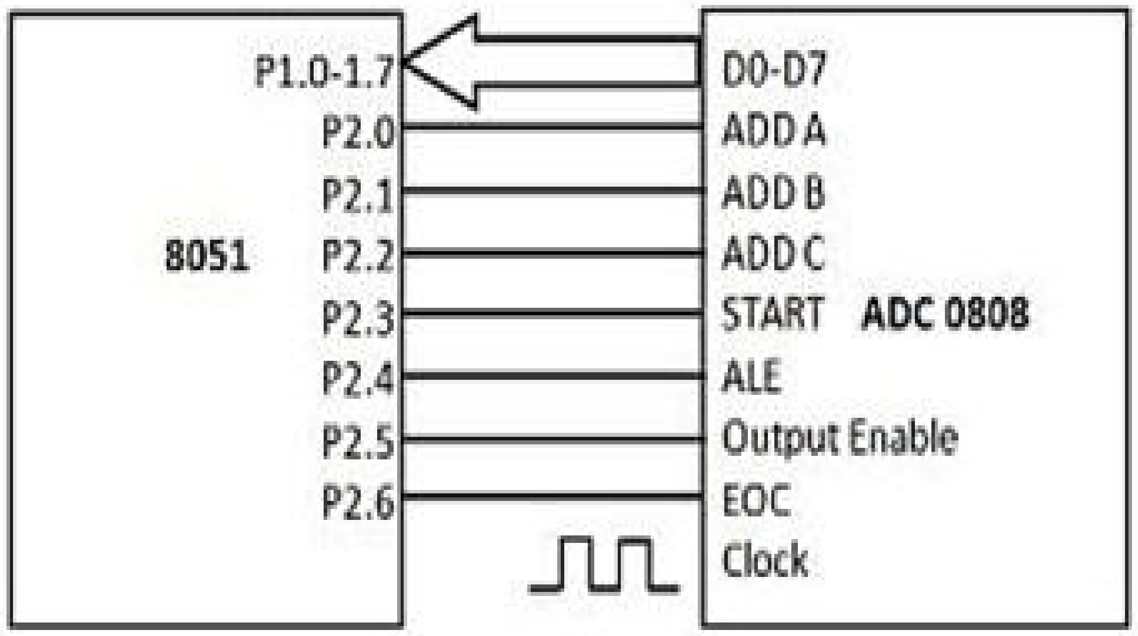
The 8051 automatically evaluates whether an interrupt should occur after every instruction. When checking for interrupt conditions, it checks them in the following order:

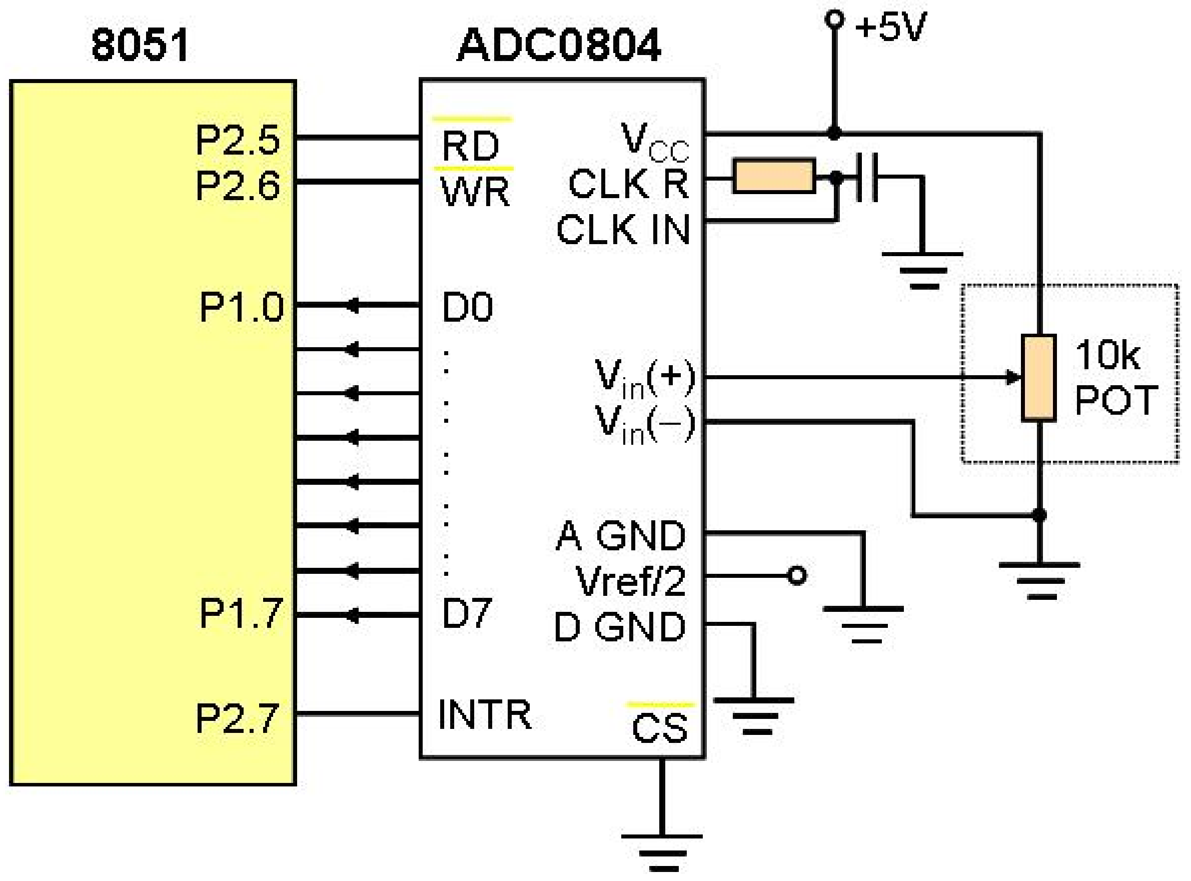
1) External 0 Interrupt
2) Timer 0 Interrupt
3) External 1 Interrupt
4) Timer 1 Interrupt
5) Serial Interrupt

Interrupt Priorities

The 8051 offers two levels of interrupt priority: high and low. By using interrupt priorities you may assign higher priority to certain interrupt conditions. For example, you may have enabled Timer 1 Interrupt which is automatically called every time Timer 1 overflows. Additionally, you may have enabled the Serial Interrupt which is called every time a character is received via the serial port. However, you may consider that receiving a character is much more important than the timer interrupt. In this case, if Timer 1 Interrupt is already executing you may wish that the serial interrupt itself interrupts the Timer 1 Interrupt. When the serial interrupt is complete, control passes back to Timer 1 Interrupt and finally back to the main program. You may accomplish this by assigning a high priority to the Serial Interrupt and a low priority to the Timer 1 Interrupt. Interrupt priorities are controlled by the **IP**SFR (B8h). PSW consists of many individual bits that are set by various 8051instructions. A*lways* protect PSW by pushing and popping it off the stack at the beginning and end of your interrupts. It will not be allow to execute the instruction: PUSH R0 Because depending on which register bank is selected, R0 may refer to either internal ram address 00h, 08h, 10h, or 18h.R0, in and of itself, is not a valid memory address that the PUSH and POP instructions can use. Thus, if you are using any "R" register in your interrupt routine, you will have to push that register's absolute address onto the stack instead of just saying **PUSH R0**. For example, instead of PUSH R0 you would execute: PUSH 00h

4. Briefly explain the method of interfacing A-to-D converter with microcontroller. [CO5-L2-Nov/Dec 2016]





Interfacing a Microprocessor to Keyboard. When you press a key on your computer, you are activating a switch. There are many different ways of making these switches. An overview of the construction and operation of some of the most common types.

Mechanical key switches: In mechanical-switch keys, two pieces of metal are pushed together when you press the key. The actual switch elements are often made of a phosphor-bronze alloy with gold platting on the contact areas. The key switch usually contains a spring to return the key to the nonpressed position and perhaps a small piece of foam to help damp out bouncing.

Some mechanical key switches now consist of a molded silicon dome with a small piece of conductive rubber foam short two trace on the printed-circuit board to produce the key pressed signal.

Mechanical switches are relatively inexpensive but they have several disadvantages. First, they suffer from contact bounce. A pressed key may make and break contact several times before it makes solid contact. Second, the contacts may become oxidized or dirty with age so they no longer make a dependable connection.

Higher- quality mechanical switches typically have a rated life time of about 1 million keystrokes. The silicone dome type typically last 25 million keystrokes.

Membrane key switches: These switches are really a special type of mechanical switches. They consist of a three-layer plastic or rubber sandwich.

The top layer has a conductive line of silver ink running under each key position. The bottom layer has a conductive line of silver ink running under each column of keys.

Mechanical key and its response to key press
The key board interfaced is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of the micro controller 8051. We normally use 8*8 matrix key board. So only two ports of 8051 can be easily connected to the rows and columns of the key board. Whenever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified.

To check the row of the pressed key in the keyboard, one of the row is made high by making one of bit in the output port of 8051 high . This is done until the row is found out. Once we get the row next out job is to find out the column of the pressed key. The column is detected by contents in the input ports with the help of a counter. The content of the input port is rotated with carry until the carry bit is set. The contents of the counter is then compared and displayed in the display. This display is designed using a seven segment display and a BCD to seven segment decoder IC 7447. The BCD equivalent number of counter is sent through output part of 8051 displays the number of pressed key.

Interfacing To Alphanumeric Displays
• To give directions or data values to users, many microprocessor-controlled instruments and machines need to display letters of the alphabet and numbers. In systems where a large amount of data needs to be displayed a CRT is used to display the data. In system where only a small amount of data needs to be displayed, simple digit-type displays are often used.
• There are several technologies used to make these digit-oriented displays but we are discussing only the two major types.
• These are *light emitting diodes* (LED) and *liquid-crystal displays*(LCD).
• LCD displays use very low power, so they are often used in portable, battery-powered instruments. They do not emit their own light, they simply change the reflection of available light. Therefore, for an instrument that is to be used in low-light conditions, you have to include a light source for LCDs or use LEDs which emit their own light.
Interfacing Analog to Digital Data Converters
• In most of the cases, the PPI 8255 is used for interfacing the analog to digital converters with microprocessor.
• The analog to digital converters is treaded as an input device by the microprocessor, that sends an initialising signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
• The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.

• The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
• It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
• The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.
• General algorithm for ADC interfacing contains the following steps:

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion pulse to ADC
3. Read end of conversion signal to mark the end of conversion processes.
4. Read digital data output of the ADC as equivalent digital output.
5. Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
6. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

ADC 0808/0809:

• The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100μs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.

Interfacing ADC with 8255 of microcontroller

These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do no contain any internal sample and hold circuit. If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

Interfacing Digital to Analog Converters: The digital to analog converters convert binary number into their equivalent voltages. The DAC find applications in areas like digitally controlled gains, motors speed controls, programmable gain amplifiers etc.

AD 7523 8-bit Multiplying DAC: This is a 16 pin DIP, multiplying digital to analog converter, containing R- 2R ladder for D-A conversion along with single pole double thrown NMOS switches to connect the digital inputs to the ladder.

5. Explain with a neat sketch how microcontrollers and microprocessors can be used for the stepper motor control application. [CO5-L2-Nov/Dec 2016]

Stepper Motor Interface:

The complete board consists of transformer, control circuit, keypad and stepper motor as shown in snap.The circuit has inbuilt 5 V power supply so when it is connected with transformer it will give the supply to circuit and motor both. The 8 Key keypad is connected with circuit through which user can give the command to control stepper motor. The control circuit includes micro controller 89C51, indicating LEDs, and current driver chip ULN2003A. One can program the controller to control the operation of stepper motor. He can give different commands through keypad like, run clockwise, run anticlockwise, increase/decrease RPM, increase/decrease revolutions, stop motor, change the mode, etc. **Unipolar stepper motor:-** unipolar stepper motor has four coils. One end of each coil is tied together and it gives common terminal which is always connected with positive terminal of supply. The other ends of each coil are given for interface. Specific color code may also be given. Like in my motor orange is first coil (L1), brown is second (L2), yellow is third (L3), black is fourth (L4) and red for common terminal.

By means of controlling a stepper motor operation we can

1. Increase or decrease the RPM (speed) of it
2. Increase or decrease number of revolutions of it
3. Change its direction means rotate it clockwise or anticlockwise

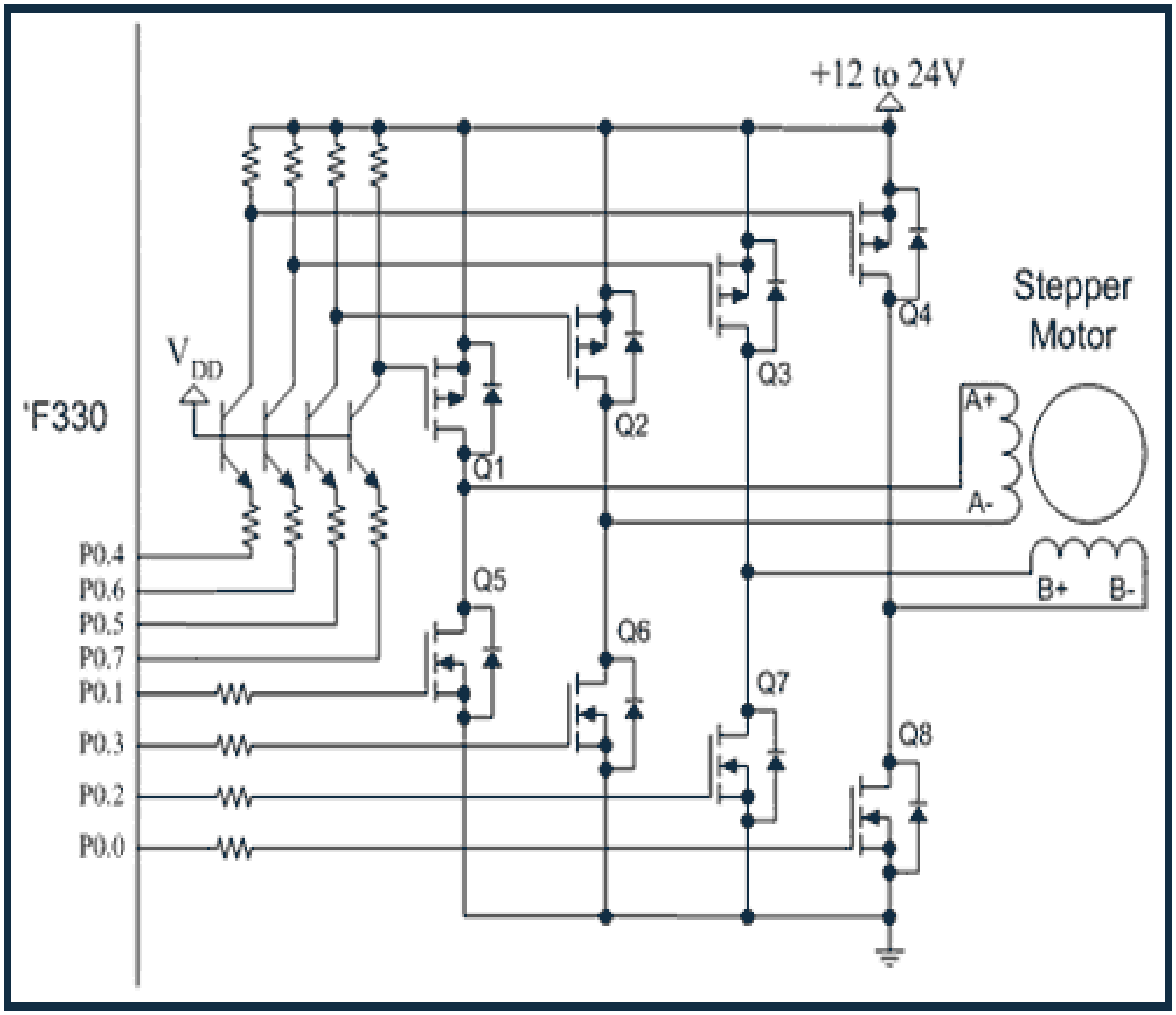
To vary the RPM of motor we have to vary the PRF (Pulse Repetition Frequency). Number of applied pulses will vary number of rotations and last to change direction we have to change pulse sequence. So all these three things just depends on applied pulses. Now there are three different modes to rotate this motor

1. Single coil excitation
2. Double coil excitation
3. Half step excitation

The table given below will give you the complete idea that how to give pulses in each mode The circuit consists of very few components. The major components are 7805, 89C51 andULN2003A.

Connections:-

1. The transformer terminals are given to bridge rectifier to generate rectified DC.
2. It is filtered and given to regulator IC 7805 to generate 5 V pure DC. LED indicates supply is ON.
3. All the push button micro switches J1 to J8 are connected with port P1 as shown to form serial keyboard.
4. 12 MHz crystal is connected to oscillator terminals of 89C51 with two biasing capacitors.
5. All the LEDs are connected to port P0 as shown
6. Port P2 drives stepper motor through current driver chip ULN2003A.
7. The common terminal of motor is connected to Vcc and rest all four terminals are connected to port P2 pins in sequence through ULN chip.



Stepper motor control board circuit