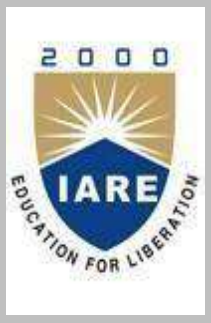
LECTURE NOTES

ON

MICROPROCESSORS AND MICROCONTROLLERS

III B. Tech II semester (JNTUH-R15)

Mr.M.LAKSHMI RAVITEJA
Mr. N.PAPARAO
Mrs.C.DEEPTHI
Mr.S.LAKSHMANA CHARI



ELECTRONICS AND COMMUNICATION ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING
(AUTONOMOUS)
Dundigal, Hyderbad-500043

UNIT-I

Microprocessor deals with hardware structure of computer system.

Microprocessor is a C.P.U fabricated on small silicon chip

(or)

C.P.U built on chip is known as Microproces

Intel introduced first MP - 4004

It is a 4-bit Microprocessor. It can operate on 4-bits at a time. Both arthematic and logical operations can be performed.

Buses:-

collection of wires is known as Bus These are of three types.

Address Bus:- (size of address bus - 4)

If there are 4 bits. It can use 16 Memory locations

Data Bus:- (Size of data bus - 4)

It can transfer only 4-bits at a time.

4004 - Its processing capability is very very less.

*** 4-bit processor:- Data bus size is 4

Later they introduced 8085.

8085 : It is most widely used. It is an 8-bit processor i.e Data bus size is 8 It can perform operations on 8-bits. Its address bus size is 16 i.e $2^{16} = 65536$ It can use 65,536 memory locations. (or) 64KB of Memory. It is having large no. of instructions.

Instruction :-

The operations understood by any C.P.U is called Instruction.

Total no. of instructions understood by any C.P.U is called instruction set.

eg:- BUN Branch shift
BSA jmp

opcode	operand

operand :- It is the data on which data opcode is to be operated.

Microprocessor use's 8-bits for opcode. If 8-bits are used for one opcode, then it uses 256 bits for two opcodes

MP releases the instruction set in hexadecimal code.

eg:- ADD - 4F - 0100 1111

8085:-

It is a 16-bit processor. It is small in size.

Micro computer:-

Micro computer is the combination of RAM, set of registers, A.L.U and C.U are fabricated on the same chip.

Micro processor:-

Micro processor is the combination of set of registers, A.L.U and C.U, In this RAM is not fabricated on IC

Most popular MP is 8085. It has built in capability. It supports interrupts.

Overview of 8085 MP:-

Why Crystal is Preferred as clock source

Be
No. of bits processed by the Processor in a sngl system
→
logical address = Segment : offset
Effective address = MOV AX [base reg + index reg + constant]

machine cycle is defined as the time required to complete one operation of accessing memory, I/O (or) acknowledging an external request. this cycle may consist of three to six states

T-state is defined as one subdivision of the operation performed in one clock Period. Those subdivisions are internal states synchronized with the system clock and each T-state is precisely equal to one clock Period

---

8259 - Programmable interrupt Controller

8 INP - Permanently changes the program Counter of Call instruction - saves information on the stack so

8085

Internal data bus contains 6 General purpose register. Whenever data is brought outside from memory into C.P.U and places data in any one of the register. Each register stores 8-bit data. If two register are combined then it is called pair of register. These are used to store (or) manupulate data containing 16-bit data.

Accumulator:

It is used to transfer one of the operands to A.L.U. It is the door for A.L.U It is the entrance register for A.L.U. Once the operation is performed, result is placed in Accumulator.

Temporary register:

It is used to transfer data to A.L.U. i.e., second operand is sent to A.L.U from temporary register.

Instruction:

opcode	operand

Whenever instructions are typed from keyboard, the opcode and operand are stored in memory (RAM).

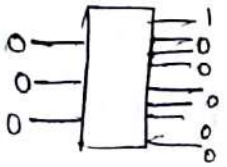
when the opcode is fetched from memory into the C.P.U, it places

the opcode in instruction register (IR)

IR is used to hold the instruction (or) code when it is fetched from memory into C.P.U

once the opcode is fetched, it must be decoded. for example, if the opcode is 8-bit, there will be 256 lines at the o/p.

consider 3x8 decoder.



The o/p enables the respective line and performs the particular operation. and remaining lines are deactivated.

Then control unit gives the respective signal to the A.L.U. then the A.L.U understands the type of operation it has to perform.

Flag register is condition (or) status register (or) condition code register (or) Processor status word (PSW).

To indicate particular condition we use flipflop. The flag register contains 8-bits. But it uses only 5 bits i.e 5 conditions.

To store the carry, processor treats it has condition and C.P.U uses flipflop or Flag register.

carry may be present (or) may not present. If carry is not present, then it resets the FlipFlop, else FlipFlop sets.

Zero Flag:-

If zero flag is 'one' then result is zero, else non-zero. it resets.

Sign Flag:-

When operation is performed, if the result is -ve then S='1'; if result is +ve then S='0'.

Parity Flag:-

It tests total no. of one's in the result, if total no. of one's is even, then it resets. else it sets.

Auxilary carry Flag:- (AC)

whenever two 8 bit no. of added, when the carry is generated from D_3 to D_4, then AC Flag is going to be set else it resets.

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

1 0 0 1 1 0 0 1

We need to check carry flag when there is repeated addition (or) Subtraction.

SUB R_1 ; $AC \leftarrow AC - R_1$

CMP R_1 ; AC ← larger - Smaller no.

Note:-

Always assume that 'AC' contains larger

Use of zero Flag:-

zero Flag is used (or) checked when there is count.

eg:- series of no. and in arrays loops are used and hence it counts in decremented manner and sets zero one when result is zero.

Use of auxilary Flag:-

It is used in BCD states.

1001 to 1111 are prohibited states and correction factor(6) is added and if carry is generated from D_3 to D_4, then auxilary flag is used.

Use of Sign Flag:-

It is used in signed and unsigned no.

Use of Parity Flag:-

Destination checks the result fohether

it is correct (or) not. it is even (or) odd. If there is error, it again sends to receiver.

Timing and control unit :-

control unit : is nerve centre of C.P.U. It generates the signals at appropriate time.

Timing unit counts clock pulses.

→ The operating clock frequency of 8085 is 3MHz

$$T = \frac{1}{3 \times 10^6}$$

$$= 0.33 \mu sec.$$

The codeword fetched from memory into C.P.U initially is opcode.

A.L.E (Address Latch Enable) :-

(1) When A.L.E = 1 then C.P.U understands that one instruction is begining and (2) multiplex buses.

(2) 8085 is 8-bit µP

Data bus size is 8-bits

Address bus size is 16-bits.

8085 is 40 pin IC (DIP Package)

In 8085, only 16-pins are used for data and address bus. by C.P.U.

Initially (C.P.U) it uses 16 pins for address, but it uses 24 pins when it comes out from C.P.U.

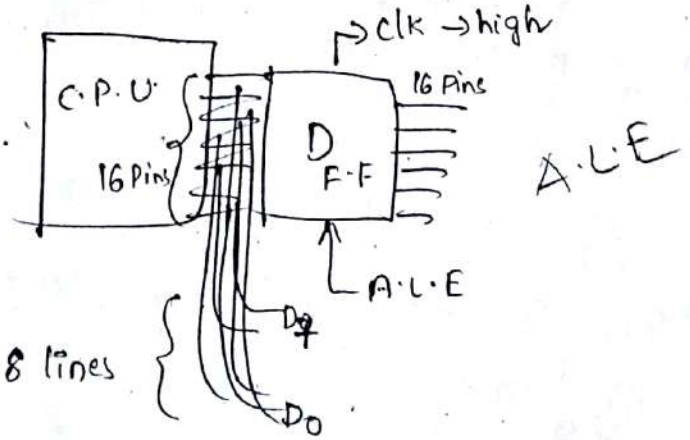
once the address is found, 8 pins are used for data bus.

Again if it want to use address bus, it uses all 24-pins outside C.P.U. of which 16 are for address and 8 for data bus

A_{15} - A_8 - high order
A_7 - A_0 - low order
} Address bus

D_7 - D_0 - Data bus.

This is called multiplexing of buses. In this two buses can be operated in time shared manner. No. of pins are reduced.



Interrupt:-

It is an asynchronous signal given to the C.P.U to bring the control of C.P.U from one program to another program.

For interupting 8085 maintains 5 pins

1st pin – INTR (Interrupt request)

2nd. It sends $\overline{INTA}$, i.e your interrupt is received.

2nd
3rd
4th
5th
RST 7.5
RST 6.5
RST 5.5
highest priority
– vector address. → user stores addres and gives '1' to the particular RST 7.5
Maskable interrupt.

6 TRAP:-

It is non – maskable, highest prior interrupt which is not usable by user. It uses the system when there is Power failure. It prevents from hanging the system. The user cannot enable (or) disable by using Software interrupt System. It is used in case of urgent applications. The vectored memory location is 0024H. The trap is also called RST 4.5 interrupt.

Serial I/O control:-

It provides facility for serial data transmission

It uses two pins

1) SID – Serial i/p Data
2) SOD – Serial o/p Data

SOD of one processor is connected to the SID of another processor and O/P is collected at SID.

8086

Architecture of 8086 Microprocessor

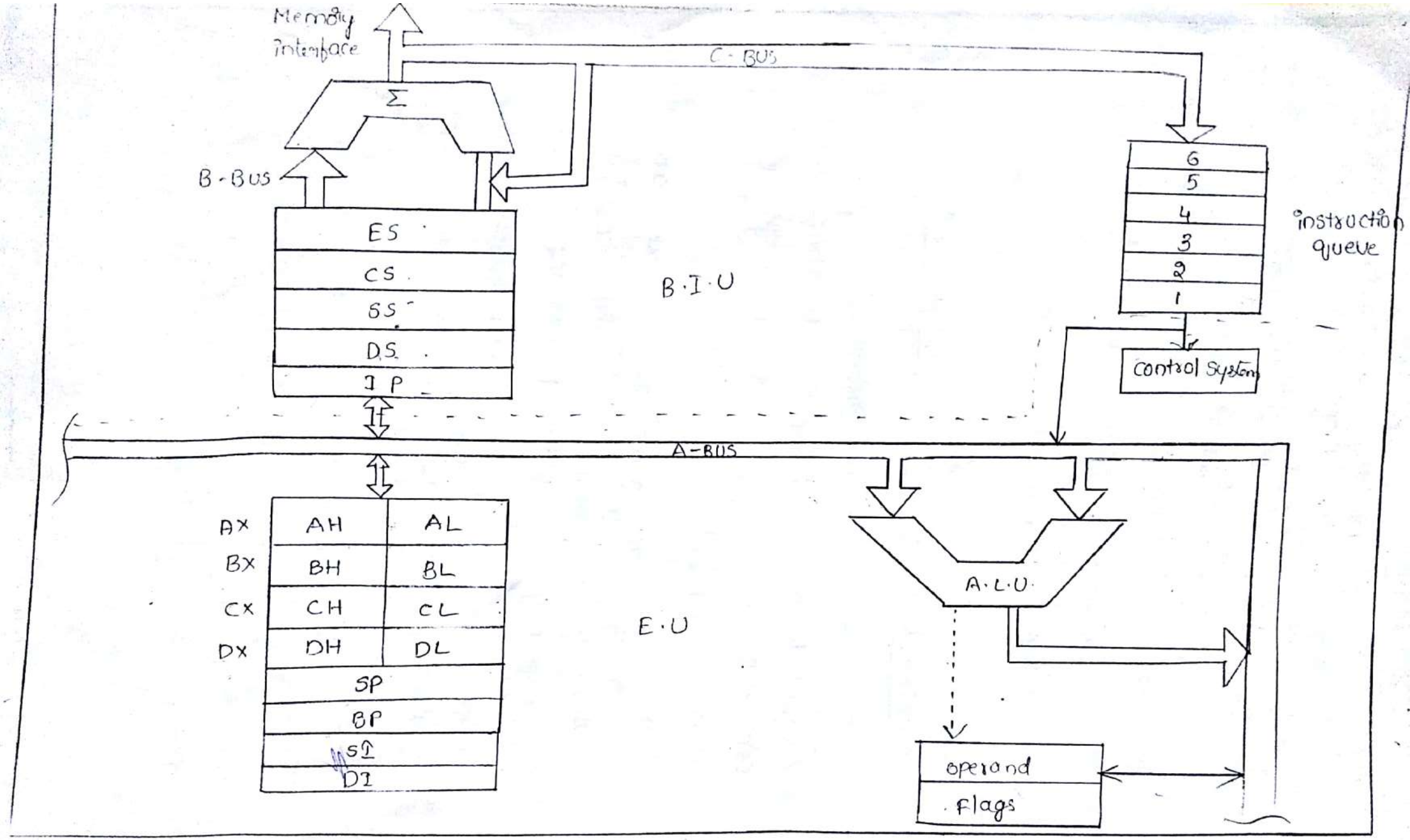
The 8086 is Intel's first 16-bit μP implemented in n-channel depletion mode silicon gate technology and packaged in 40 pin DIP package.

It has 20 bit address bus and 16 bit data bus. It is compatable with 8085 μP. The clock frequency of 8086 is 5MHz.

The internal architecture of 8086 is dividing into two separate functional units. They are :-

1) Bus interface unit (BIU)
2) Execution unit (E.U)

It's architecture is as shown.


Memory Interface
C - BUS
Σ
B - BUS
ES
CS
SS
DS
IP
B.I.U
6
5
4
3
2
1
instruction queue
control system
A-BUS
AX
AH
AL
BX
BH
BL
CX
CH
CL
DX
DH
DL
SP
BP
SI
DI
E.U
A.L.U
operand
flags

The two units B.I.U, and E.U work simultaneously for instruction execution and form two stage instruction pipelining.

B.I.U :-

The B.I.U contains bus interface logic, segment registers, memory addressing logic and a 6 byte instruction code ~~cube~~ queue.

The B.I.U performs all bus operations for the execution unit and is responsible for executing all external bus cycles.

when E.U is busy in instruction execution, the B.I.U continues fetching instructions from memory and stores them in instruction ~~cube~~ queue

If E.U executes an instruction, which transfers the control of the program to another location. then the B.I.U

1) resets the queue
2) fetches the instruction from the new address
3) passes the instruction to the E.U.
4) Begins refilling the ~~cube~~ queue from the new location.

Each time B.I.U fetches new instruction in instruction queue while E.U is in execution known as pipelining.

E.U :-

It contains A.L.U, General purpose registers, pointer and index register, flag register and control circuits, decoding circuits etc...

The E.U is responsible for:

1) The execution of all instructions.
2) Providing address to the B.I.U for fetching data (or) instruction.
3) Manipulating various registers as well as flag register.

2/12/11 General Purpose registers:-

The 8086 has 4 16-bit data registers AX, BX, CX, DX. They may be treated as 4 16-bit registers (or) 8 8-bit registers.

The AX register serves as Accumulator. I/O operations pass data through AX (or) AL In most of the instructions, one of the operand may be supplied through AX and result is placed in accumulator.

The BX register serves as general purpose data register, it can be used as base register while computing the data memory address.

The CX register serves as general purpose data register and also it can be used as count register.

The DX register serves as general purpose data register, also used in I/O instructions, multiply and divide instructions.

Segment registers:-

In 8086, the memory (R.A.M) is segmented. In this memory is divided into a no. of parts called segments. The 1MB physical memory is divided into 4 segments. They are:

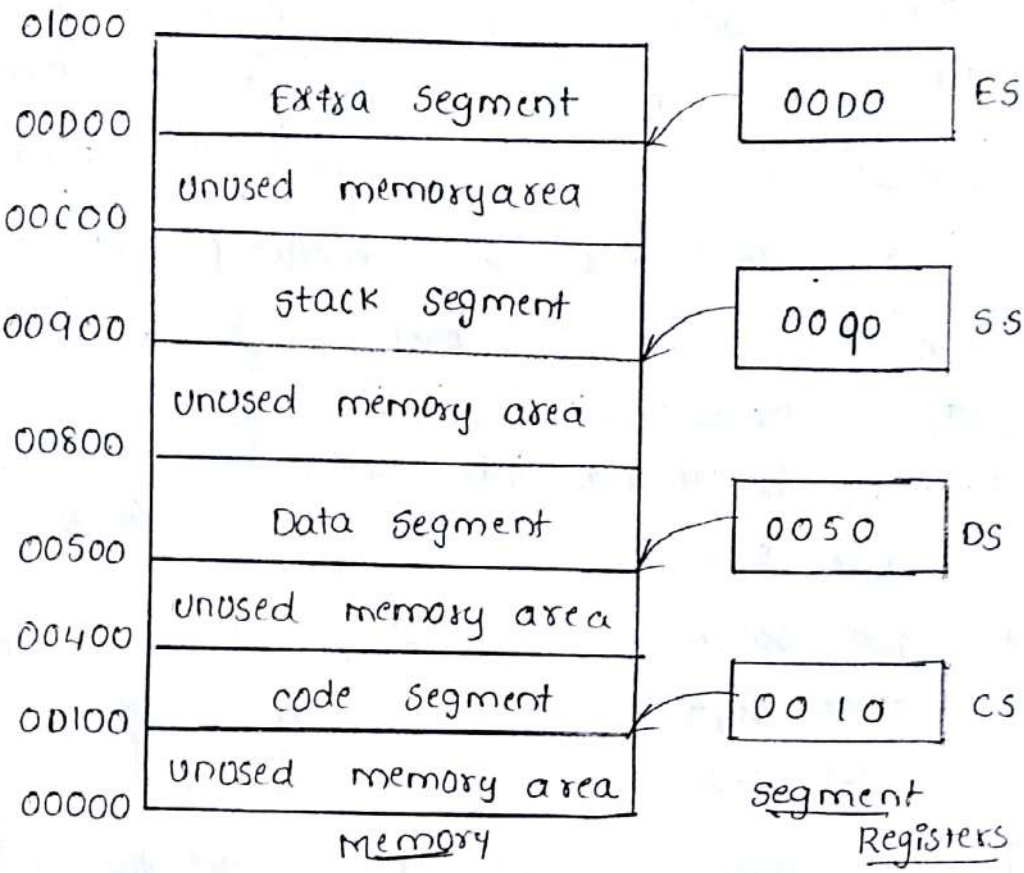
- code segment
- data segment
- stack segment
- Extra segment.

Each segment has memory space of 64KB. Each segment is addressed by a 16-bit segment register as 'CS' Register, DS register, SS register, ES register.

Each segment register holds the starting (or) begining address of the segment.

The physical address of 8086 is 20 bits. The segment register supplies the high order 16-bits of the 20 bit memory address.

The memory address of 8086 is calculated by summing the contents of the segment register shifted left by 4-bits to offset address.



The offset address (or) effective address is calculated in different ways for different addressing modes.

The selected segment register contents are left shifted by 4-bits, this now represents the starting address of the segment in memory.

The effective address i.e address basically represents the offset from the starting address of the segment.

The offset address is added to segment register contents after 4-bits left shift to get the actual memory location address.

Segment Register contents - XXXX (H)

Shift segment register - XXXX0 (H)

Offset address - XXXX (H)

Physical address - XZZZY (H)

Example :-

let content of CS is 010AH and offset address i.e the content of IP is F950H.

Shift CS 4 times

CS → 0 1 0 A 0

IP → F 9 5 0

1 0 9 F 0 H

Note :-

The segments may also be used with overlaping. Then, the common location may have two logical addresses.

Pointer and index registers :-

Stack pointer :-

The stack pointer is used in instructions which use stack. The stack pointer always points to a location in memory known as the stack top.

The complete address of stack is calculated by adding the contents of SS register after 4-bits left shift to the stack Pointer.

SS ⟶ 0090

Shift SS by 4 times

SS → 00900
SP ⟶ 16F2
01FF2

Base Pointer (BP):-

The purpose of this register is to provide indirect access to data in stack register. It may also be used for general data storage.

Source Index and Destination index :-

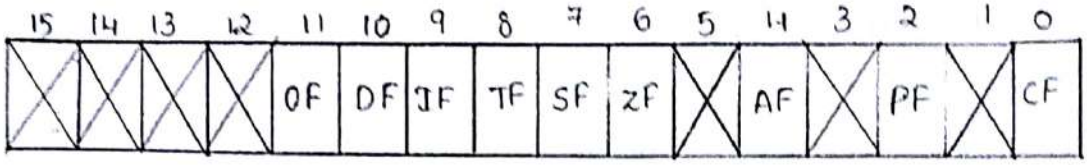
These registers may be used for general data storage, the main purpose of these registers is to store offset in case of indexed, base indexed and relative base indexed addressing modes, also used in string manupulation.

This register is also known as PC (Program counter). It is used to store the offset for the instruction.

During the instruction fetch, I.P contents are added to the code segment register after 4 bits left shift.

Flag register:-

The flag register is also known as status register (or) condition code register (or) program status word. It is as shown.



Carry Flag:-

It is set after an arthematic operation results in carry out of MSB (or) a borrow in subtraction.

It is also used in some shift and rotate instructions.

Parity Flag:-

It is set if there are even no of 1's in result, otherwise it resets.

Auxiliary carry Flag:-

This flag is set if there is a carry out from D_3 to D_4 in 8-bit operation and D_7 to D_8 in 16-bit operations. It is used for BCD operation.

Zero Flag:-

It is set whenever the result of the operation is zero, otherwise it resets.

Overflow:-

This Flag is used to detect magnitude overflow in signed arthematic. Generally, when the size of the result is not stored in a register it indicates overflow.

This flag is 'set' when there is overflow.

Direction Flag:-

It is used with string operations. when 'set' it causes the string instruction to process strings from right to left, otherwise strings are processed from left to right.

Interrupt enable:-

This flag enables the 8086 to recognise the external interrupt requests. when IF=0, all maskable interrupts are disabled. It has no effect on non-maskable interrupts (or) internally generated interrupts.

Trap flag:-

Seting the trap flag (TF) bit puts the processor into single step mode for debugging.

O.F. flag:

In signed representation, this flag is set if the result is -ve, otherwise it resets.

A.L.U:-

A.L.U performs operations like addition subtraction, multiplication, division and logical operations like AND, OR, NOT, EXOR etc.

Instruction queue:-

8086 maintains 6 byte queue in which 6 registers are present to store opcodes (or) object code.

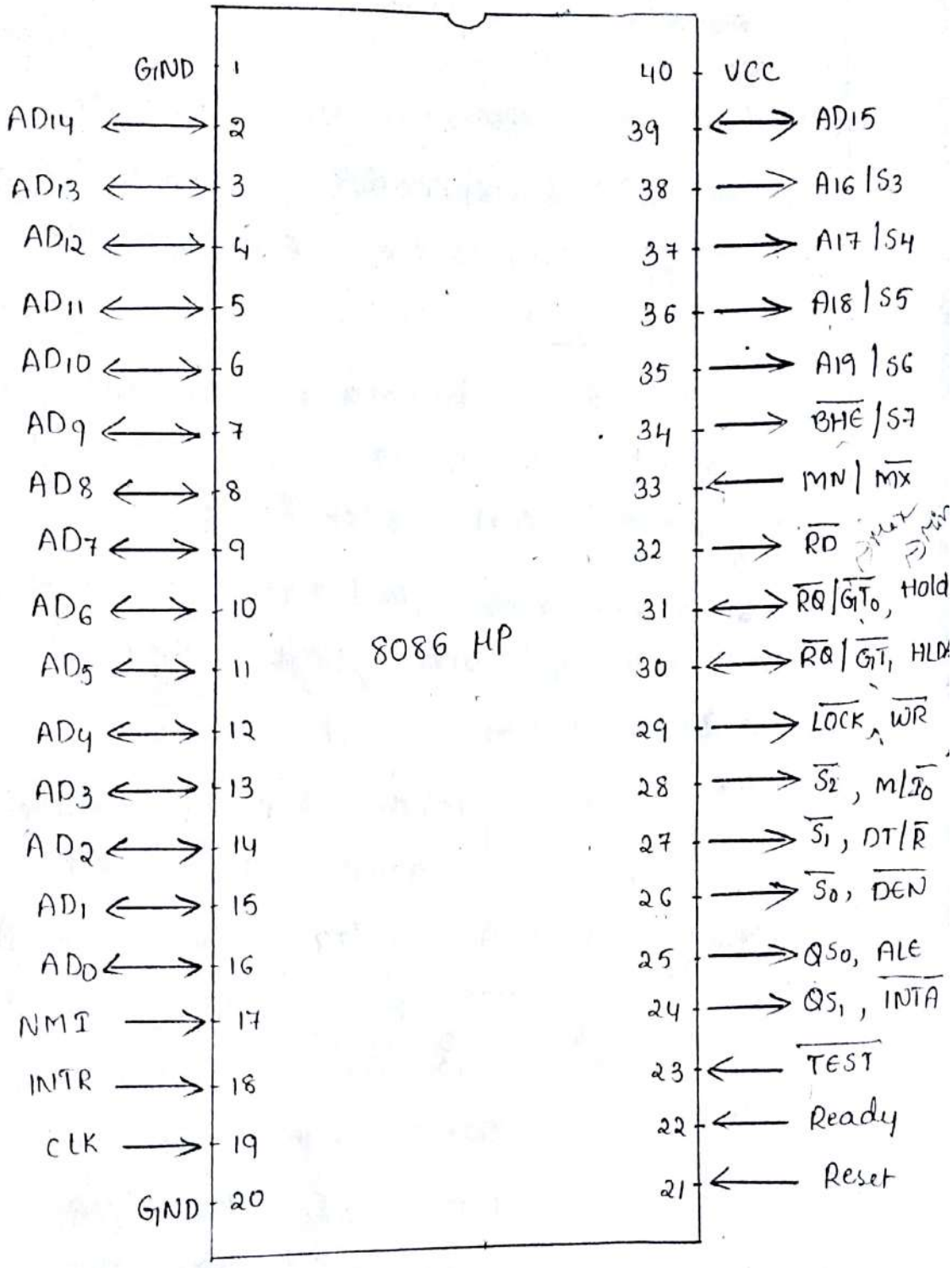
B.I.U prefetches instructions from memory and stores them in instruction queue.

E.U takes the instructions from Instruction queue i.e., 8086 supports pipelining with this concept.

Pin configuration of 8086:-

The 8086 is 40 pin IC available in DIP. 8086 can work in minimum mode (or) in maximum mode. Its pin structure (or) pin-out is as shown.

Pin configuration of 8086:-



Pin no	Name	Type	Description
39, 2-16	$AD_{15} - AD_0$	Bidirectional tristate	They act as address bus during the first part of machine cycle and as data bus in latest path.
35-38	$A_{19}/S_6 - A_{16}/S_3$	output tristate S_4, S_3 - segment pins 0 0 – CS 0 1 – SS 1 0 – CS 1 1 – DS	It contains address information in the first part and status bits in the later part The status bits when decoded indicate the type of operations. Eg: memory access, type of segment register etc...
34	$\overline{BHE}$ / S7 BUS HIGH ENABLE	output tristate	$\overline{BHE}$ is output during the first part of the machine cycle. zero on $\overline{BHE}$ pin indicates access to high order memory. AD_{15} to AD_8, otherwise, access is to $AD_7 - AD_0$. $S_7 = 1$ is o/p during the later part of ma-chine cycle. NO function is assigned to S7
32	$\overline{RD}$	Output tristate	It is low when 8086 is receiving data from memory (or) I/o device.

Pinno	Name	Type	Description.
22	Ready	Input This is ack from the slow devices or memory that they have completed the data tran	It is wait state request signal. A high on this line causes the 8086 to enter into extend the machine cycle by wait states.
23	$\overline{TEST}$	Input	It is used in conjunction with WAIT instruction. The instruction puts the 8086 in idle state which ends only when the $\overline{TEST}$ input goes low.
18	INTR	Input	It is the level-triggered interrupt signal. It is sampled during the last clock cycle of each instruction.
17	NMI	INPUT It is internally Synchronised.	Non Maskable interrupt (NMI) is positive edge triggered non-maskable interrupt request.
19	CLK	Input The 8284 clock genera for chip of Intel is	CLK is single clock signal from external crystal controlled generator. The 8086 requires clock signal with 33% duty cycle. Following are the clock frequencies for different versions of 8086 8086 5MHz 8086-2 8MHz

3/12/11

Pin no	Name	Type	Description.
			It provides the basic timing for processor operation and bus control activity. It is an asymmetric square wave with 33% duty cycle.
21	Reset	input FFFF FFFF0 0000 FFFF0	This input causes the processor to terminate the current activity and start execution from FFFF0H. This signal is active HIGH, and it is internally synchronized.
40	VCC	supply	'+5V' power supply for the operation of the internal circuit.
1, 20	Ground		Ground for the internal circuit
33	MN/$\overline{MX}$	Input	The logic level at this pin decides whether the processor is to operate in either minimum (or) maximum mode. If it is high, operates in mini-mum mode, If it is zero, operates in maximum mode.

Minimum mode pin functions:-

Pinno	Name	Type	Description.
25	A·L·E	output	It is used to indicate begining of an operation and to demultiplex address and data buses.
29	~~DEN~~, $\overline{WR}$	output tristate	write control signal:- when this is low, then data is stored in memory and I/O (or) through data bus
28	M/$\overline{I/O}$	output tristate	when it is low, it indicates the C·P·U is having an I/O operation and when it is high, it indi-cates that C·P·U is having a memory operation.
24	$\overline{INTA}$	output	This signal is used as a read strobe for interrupt acknow ledge cycle. when it goes low, it means that the Processor has accepted the interrupt.
27	DT/$\overline{R}$	output tristate	This output is used to decide the direction of dataflow through the ~~transceivers~~ transceiver

Pin no	Name	Type,	Description
			processor sends out data, this signal is high and when the processor is receiving the data, this signal is low.
26	$\overline{DEN}$	output tristate	This signal indicates the availability of valid data over address/ lines. It is used to enable, the transreceivers to seperate the data from the multiplexed address / data signal.
31	Hold	Input	when the Hold line goes high, it indicates to the Processor, that another master is requesting the buses, as in the case of D.M.A
30	HLDA	output	Hold acknowledgement:- after receiving the hold request, C.P.U issues the hold acknowledge signal on this Pin. when C.P.U detects 'zero' on hold pin, it resets 'HLDA' pin.

Maximum mode Pin functions :-

Pin no	Name	Type	Description.
28 27 26	$\overline{S}_2$ $\overline{S}_1$ $\overline{S}_0$	output tristate	These are the status lines which indicate the type of operation being carried out by the processor.

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	Indication
0	0	0	Interrupt ACK
0	0	1	Ilo Read
0	1	0	Ilo write
0	1	1	Halt
1	0	0	opcode fetch
1	0	1	memory Read
1	1	0	memory write
1	1	1	N·O·P

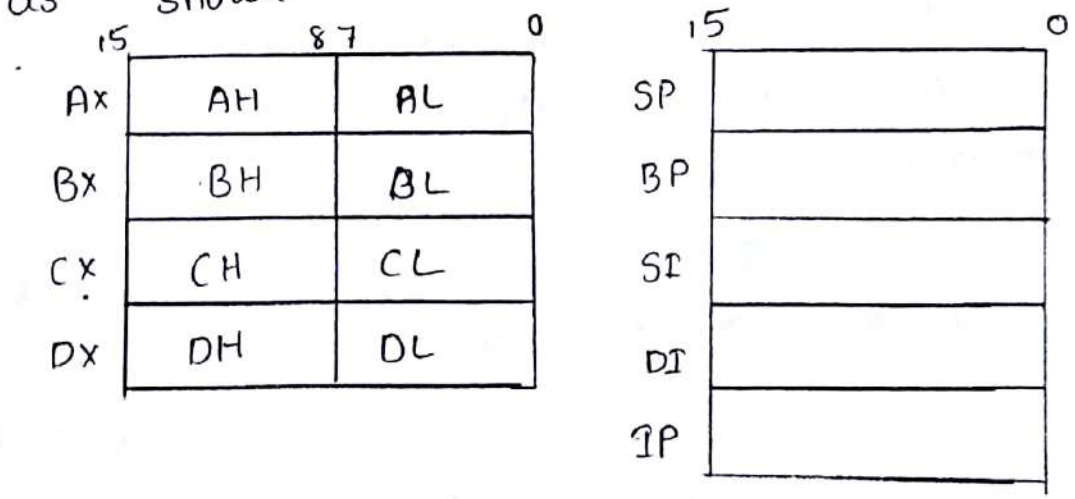
24 25	QS_1 QS_0	output	This lines give information about the status of code prefetch queue

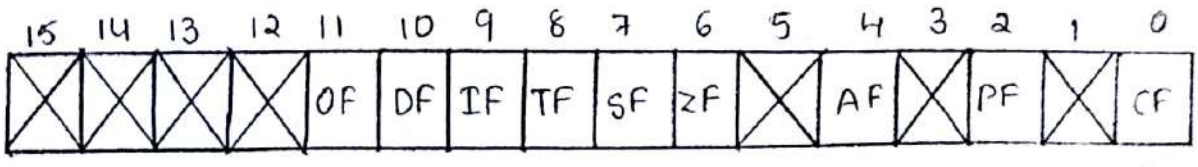
QS_1	QS_0	Indication
0	0	queue is in idle state (not operate in state)
0	1	The first byte of opcode has entered queue
1	0	queue is empty
1	1	Next byte of opcode has entered queue

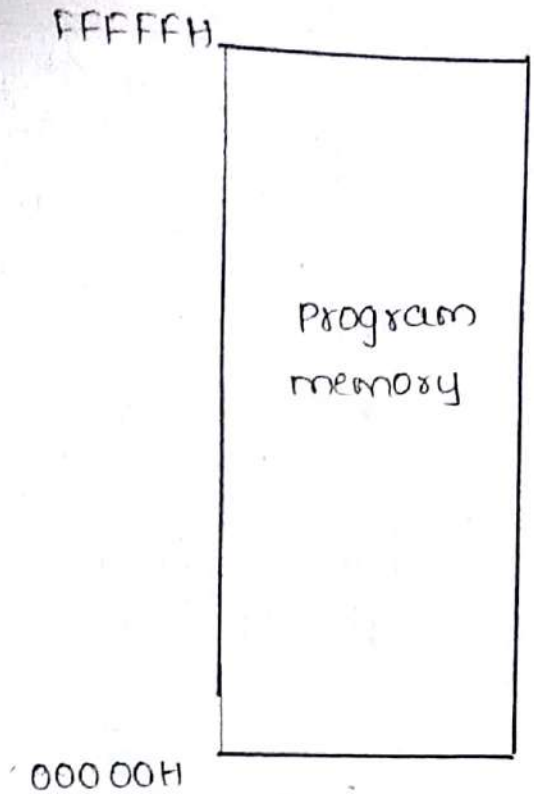
Pinno	Name	type	Description
29	$\overline{LOCK}$	output tristate	This signal indicates that an instruction with LOCK Prefix is being executed and the bus is not to be used by the other processors.
30 31	$\overline{RQ}/\overline{GT_0}$ $\overline{RQ}/\overline{GT_1}$	Bidirectional	In maximum mode HOLD, HLDA signals are converted to bidirectional signals, bus request ($\overline{RQ}$) and bus grant ($\overline{GT}$). The operation is same as HOLD, HLDA. out of $\overline{RQ}/\overline{GT_0}$ and $\overline{RQ}/\overline{GT_1}$, the $\overline{RQ}/\overline{GT_0}$ has higher priority.

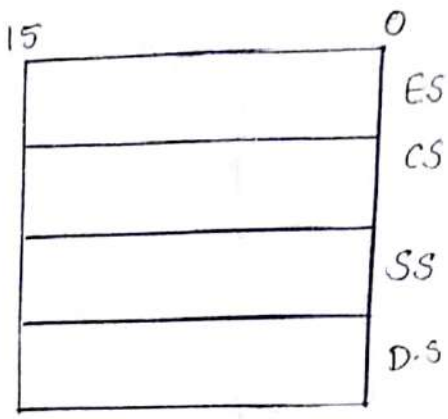
Programming Model and instruction set of 8086:-

The programming model of 8086 is as shown.






FFFFFH
Program
memory
00000H


15
0
ES
CS
SS
D.S

UNIT-II

Operand types in 8086:-

The 8086 supports the following types of operands. They are:-

1) Bytes 2) words 3) short Integers.
4) Integers 5) Double words 6) Long Integers.
7) Strings.

→ Bytes and short integers are 8-bit variables.

→ words and integers are 16-bit variables.

→ The double words and long integers are 32-bit variables.

Byte words and double words are unsigned numbers. Short integers, integers and long integers represent signed numbers. A string is a series of bytes (or) series of words stored in sequential memory locations. Normally a string is a alpha-numeric characters defined by ASCII codes.

5/12/11 Addressing modes:-

The different ways of specifying operand part of an instruction are known as "Addressing Modes". 8086 supports the following addressing modes.

(i) Register Addressing Mode

(ii) Immediate Addressing Mode

(iii) Direct Memory Addressing Mode.

(iv) Register indirect Addressing Mode.

(v) Base + Index Register Addressing Mode.

(vi) Register Relative Addressing Mode.

(vii) Base+ Index Register relative Addressing Mode

(viii) String Addressing Mode.

(i) Register Addressing Mode :-

Where both destination and source operands are present in registers. Then the addressing mode is known as register addressing mode.

Eg:-

1) MOV AX, BX

(AX) ← (BX)

Move the contents of BX register to AX register, contents of BX register is unchanged.

2) ~~AND, ALL~~

AND AL, BL

(AL) ← (AL) ∩ (BL)

AND the contents of AL register with the contents of BL register and place the result in AL.

when one of the operand indicates the data then it is known as immediate addressing mode and that data is known as immediate data.

Eg:-

1) MOV CX, 1465H

(CX) ← 1465H

copies 16 bit data 1465H into register CX.

2) SUB AL, 15H

AL ← (AL) − 15H.

Subtracts 15H from the contents of AL register and places the result in AL register.

(iii) Direct Addressing Mode :-

In this mode the 16 bit offset address is part of the instruction as displacement field. It is stored as 16 bit unsigned or 8 bit signed extended number.

Ex:

1) MOV (4625H), DX

copies the content of DX register into memory locations calculated from data segment register and offset 4625(H)

[(DS) x 10H + 4625H] ← DL

[(DS) x 10H + 4625(H)+1] ← DH

2) OR AL, (3030H)

OR's the contents of AL register with the contents of memory location calculated from DS register and offset (3030H), then the result is copied into AL register.

$$AL \leftarrow [(DS) \times 10H + 3030H] \cup (AL)$$

(iv) Register Indirect Addressing Mode :-

In this mode, the offset address is specified through pointer register or Index Register. For Index register the SI register (or) DI may be used, where as for pointer register Bx register (or) BP register may be used.

Ex :-

1) MOV AL, (BP) offset

It copies into AL register, the contents of memory location whose address is calculated using offset as contents of BP register and the contents of DS register.

$$(AL) \leftarrow [(DS) \times 10H + (BP)]$$

2) XOR (DI), CL

XOR operation is performed b/w the contents of CL register and the contents of memory location whose address is calculated by DS and DI and the result is placed in the same address.

[(DS) X 10H + (DI)] ← [(DS) X 10H + (DI)] XOR CL

(v) Base + Index Register addressing mode :-

In this mode both base register (BP or BX) index register (SI or DI) are used to indirectly address the memory location. This is useful when an array of data is to be addressed.

Ex :-

→ offset

1) MOV (BX + DI), AL

copies the contents of AL register into memory location whose address is calculated using the contents of BX register and DI registers.

[(DS) X 10H + (BX) + (DI)] ← (AL)

7/12/11

(vi) Register Relative Addressing Mode :-

This method or mode is similar to Base + Index addressing mode. The offset is calculated using either a base register (BP, BX) or an Index register

(SI, DI). The displacement is specified as an 8-bit (or) 16 bit number as part of instruction. The displacement is specified for addition or subtraction like (BX+3), (DI - 0050 H).

Ex:-

MOV AX, (DI+06H)

It copies to AL the contents of memory location whose address is calculated using DS, DI with displacement of 6 and copy to AH. The contents of Next higher Memory location.

(AL) ← [(DS) x 10H + (DI) + 06H]

(AH) ← [(DS) x 10H + (DI) + 07H]

(vii) Base + Index + Register Relative addressing mode :-

This addressing mode is the combination of base + index register addressing mode and register relative addressing mode. To find the address of the operand in memory. A base register (BP or BX), an index register (DI or SI) and the displacement which is specified in instruction is used along with the data register

Ex:-

MOV(BX + DI + 2), CL

It copies the contents of the CL register to the memory location whose address is calculated using DS, BX and DI and displacement 02.

[(DS) × 10H + (BX) + (DI) + 02] ← (CL)

(viii)

String addressing mode:-

The accessing of the string operands is different from that of other operands. There are special instruction for string operations. Apart from the segment register, the index registers SI and DI are used.

The segment register DS and the index register SI are used for source string. whereas for destination string, the segment register ES and the index register DI are used. The ~~direction flag~~ bit.

The direction flag bit indicates increment (or) decrement operations on strings.

If DF = 0, both SI and DI are incremented

If DF = 1, both SI and DI are decremented

automatically as a part of the string instruction execution. Thus, SI and DI point to the next byte (or) word of the string.

Ex:-

MOVS B

It copies the byte from the source string location determined by the DS and SI to the destination string location ES and DI.

Ex:- If (DS) = 0500H, (SI) = 0000H, (ES) = 0F00H (DI) = 0000H, DF = 0

8/12

Source location = [(DS) x 10H + SI]

= 05000H

Destination location = [(ES) x 10H + (DI)]

= 0F000H

Instruction set of 8086:-

The instructions are (1st) represented in 0's and 1's and then hexa decimal numbers and then to english like words which is known as "Mneumonics".

The program written in english like words is called assembly language (or) low level language (Ex) which is

The total number of opcodes understood by any processor is known as its instruction set.

Assembler - [language processor]:-

It is used to convert low level language into machine language.

Cross Assembler:-

It is a language processor (or) translator used to ~~operate~~ translate both the opcodes of 8085 - 8086.

8/12/11

The total no. of instructions recognised (or) understood by any microprocessor is called its "Instruction set". The 8086 has the following group of instructions.

(i) Data transfer group [copy group]
(ii) Arithematic group
(iii) Logical group
(iv) control transfer group (Branch group)
(v) machine control group.

(i) Data transfer group:-

The instructions in this group perform data movement blw registers, Register and memory, register

and Immediate data, memory and immediate data, blw two memory location blw I/O Port and register, and between stack and Memory (or) register, Both 8 and 16 bit data transfers are provided.

(i) MOV:-

It copies the byte or word from the source operand to the destination operand.

Mneumonic	Meaning	Format	operation	Flags
MOV	MOVE	MOV D,S	D ← S	None

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate

Destination	Source
segment Register	Register - 16
segment Register	Mem - 16
Register - 16	segment register
Memory	segment register

Ex:-

1) MOV DX, CS

(DX) ← (CS)

2) MOV [sum], AX

↳ address

The memory location identified by the variable sum is specified using direct addressing i.e the value of the offset sum is used in calculation of memory address.

3) MOV cx, [source * Mem]

(2) Exchange Instruction (XCHG):-

It is used to exchange data blw two operands.

NNB: Mnemonic	Meaning	Format	Operation	Flag
XCHG	Exchange	XCHG D,S	D ⇄ S	NONE

Destination	Source
Accumulator	Register - 16
Memory	Register
Register	Register
Register	Memory

Ex:-

1) XCHG AX, DX

(DX) ← (AX),

(AX) ← (DX)

(2) XCHG [SUM], DX

Here, it exchanges data between DX and memory location specified by offset value of sum and DS.

(3) XLAT (Translate)

The translate instruction is used to replace the byte in AL with the byte from LOOK UP table (or) user table addressed by BX. The original value of AL is the index into the translate table.

Mnemonic	Meaning	Format	operation	Flag
XLAT	Translate	XLAT	AL ← [(DX) + (DS)X 10H + (AL)]	NON

(4) LEA :- [load effective address]

It is used to load a specified register with a 16 bit offset address.

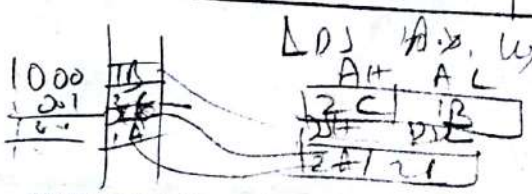
(5) LDS :- [load Data segment]

It loads 16 bit offset address into specified register and DS register.

6) LES :- [Load extra segment]

It loads a specified register as well as ES register.

Mnemonic	Meaning	Format	operation	Flag
LEA	Load effective address	LEA Reg16, EA.	Reg16 ← EA	None
LDS	(Load data segment) Load Register & DS	LDS Reg16, EA] LDS AX, [1000H] AL ← [1000H] AH ← [1001H] DSL ← [1002H] DSH ← [1003H]	(EA) ← Reg16 (Reg16) ← (EA) (DS) ← (EA+2)	None
LES	Load register and ES	LES Reg16, EA	Reg16 ← (EA) DS ← (EA+2)	None



Ex:-

LEA SI , EA

Effective address EA can be specified by any valid addressing mode.

LEA SI , (DI + BX + 5H)

Let DI = 1000H and BX = 20H

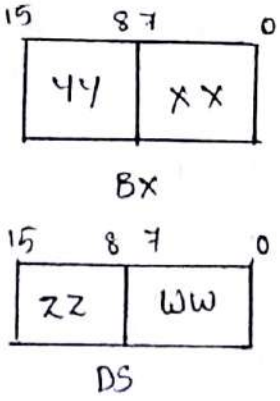
SI ← 1025H

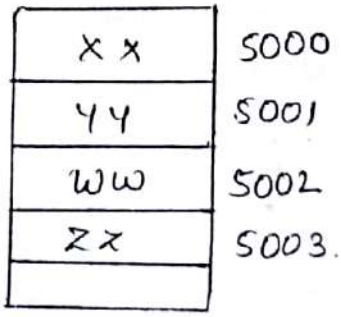
12/12/11

Note:-

LDS, LES instructions load the specified register in the instruction with the content of memory location specified and also DS (or) ES as shown below.

eg:- LDS, BX 5000H





(7) LAHF :- [Load AH from lower byte of flag register]

This instruction loads the AH register with the lower byte of the flag register. This cmd is used to observe the status of all condition flags (except

(8) SAHF :- [Store AH to lower byte of flag register].

This instruction sets (or) resets the condition code flags except overflow in the lower byte of the flag register depending upon the corresponding bit positions in AH.

(9) PUSH F :- [Push flags to stack]

This instruction pushes the flag register on to the stack, the first the upper byte and then the lower byte is pushed. The 'SP' is decremented by '2' for each push operation.

(10) POP F :- [POP flags from stack]

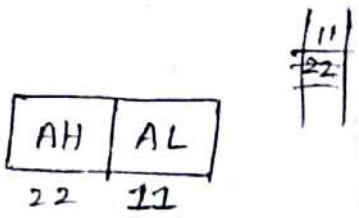
This instruction loads the flag register from the word contents of the memory location currently addressed by 'SP' and 'SS'.

The SP is incremented by '2' for each POP operation.

(11) PUSH :- [transfer the content of register to the stack top].

This instruction pushes the contents of the specified register (or) memory location onto the stack. The SP is decremented by '2'.

eg:- PUSH AX



current stack top is already occupied so decrement SP by '1' then store AH into the address pointed by 'SP', further decrement SP by '1' and store AL.

(12) POP:- [POP from stack]

This instruction loads the specified register or memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer

eg:- POP AX

(13) IN:- [Input the port]

This instruction is used to read the data from I/O device from port. The address of the i/p port may be specified in the instruction directly (or) indirectly.

AL and AX are the allowed destinations for 8 bit and 16 bit operations. Dx is the only register (implicit) which is allowed to carry the port address.

If the port address is of 16 bits, it must be in DX.

eg:-

```
IN   AL, 03H
IN   AX, DX
MOV  DX, 0100H
IN   AX, DX.
```

(14) OUT:- [output to the port]

This instruction is used for writing to an output port. The address of the output port may be specified in the instruction directly (or) implicitly in DX.

The contents of AX (or) AL are transfered.

eg:-

```
OUT  03H, AL
OUT  DX, AX
MOV  DX, 0300H
OUT  DX, AX.
```

Arithematic instructions:-

Addition instructions:-

The various addition operations are listed as shown below.

Mneumonic	Meaning	Format	operation	flag
ADD	Addition	ADD D, S	(D) ← (D) + (S)	Overflow, sign flag, Zero flag, Auxilary Carry flag, Parity flag carry flag
ADD C	Add with carry	ADC D, S	(D) ← (D) + (S) + (CF)	overflow, sign flag, Zero flag, Auxilary carry flag Parity flag carry flag
INC	increment by '1'	INC D	(D) ← (D) + 1 ;	overflow, Sign flag, zero flag, Auxilary carry flag, Parity flag
AAA	ASCII adjust for addition	AAA	PF undefined	auxilary carry flag, carry flag, Overflow, Sign flag, Zero flag.
DAA	Decimal adjust for addition	DAA	OF undefined	SF, ZF, PF CF, AF, ~~OF~~

Destination	Source
Register	Register
Register	memory
memory	Register
Register	Immediate
memory	Immediate
Accumulator	Immediate

Register 16
Register 8
memory

13/12/11

Subtract instructions :-

Mnemonic	Meaning	Format	Operation	Flag
SUB	Subtraction	SUB D, S	(D) ← (D) − (S)	OF, SF, ZF, AF, PF, CF
SUB B	Subtract with borrow	SBB D, S	(D) ← (D) − (S) − (CF)	OF, SF, ZF AF, PF, CF
DEC	Decrement by '1'	DEC D	(D) ← (D) − 1;	OF, SF, ZF AF, PF
NEG (2's Comp)	Negate	NEG D	(D) ← 0 − (D) (CF) ← 1;	OF, SF, ZF AF, PF, CF
DAS	Decimal adjust for subtraction	DAS		SF, ZF, AF, PF, CF OF − undefined
AAS	ASCII adjust for Subtraction	AAS		AF, CF, OF SF, ZF PF - undefined

For SUB, SUBB:-

Destination	Source
Register	Register
Register	Memory
Memory	register
accumulator	Immediate
Register	Immediate
Memory	Immediate

Destination (for Decrement)
Register 16
Register 8
Memory

→ It Subtracts the source operand from destination operand but the result is not stored anywhere. The Flags are modified accor-ding to the result of Subtra-ction. eg:- CMP, AX, BX TEST D, S
CMP, BX, 0100H AND operation

Destination (for Negate)
Register
Memory

CMP [0500H], BX
→ CMP [D, S] [0200H], 0200H.
(D) - (S)
But D and S are not changed according to result flags will be changed

Compare [cmp]:- [Arthematic (or) logical group]

This instruction compares the source operand which may be a register (or) Imme-diate data (or) a memory location with destination operand with a reg (or) memory

Multiplication and division instructions:-

Mnemonic	Meaning	Format	Operation	Flag
MUL	Multiply (unsigned)	MUL S	(AX) ← (AL) * S8 (DX),(AX) ← (AX) * S16	OF, CF, SF, ZF, AF, PF-undefine
DIV	Division (unsigned)	DIV S	$\frac{AX}{S8}$ → Quotient (AL) ← Q$\left(\frac{AX}{S8}\right)$ (AH) ← R$\left(\frac{AX}{S8}\right)$	OF, SF, ZF AF, PF, CF-undefine
DIV	Division (unsigned)	DIV S	AX ← Q(DX, AX)/S16 DX ← R(DX, AX)/S16	OF, SF, ZF, AF, PF, CF-undefi
IMUL	Integer Multiply (signed)	IMUL S	(AX) ← (AL) * S8 (DX)(AX) ← (AX) * S16	OF, CF, SF, ZF, AF, PF-undefined
IDIV	Integer Divide (Signed)	IDIV S	(AL) ← Q$\left(\frac{AX}{S8}\right)$ (AX) ← R$\left(\frac{AX}{S8}\right)$ (AX) ← Q(DX, AX)/S16 (DX) ← R(DX, AX)/S16	OF, SF, ZF, AF, PF CF-undefined
AAM	ASCII adjust for multiplica-tion	AAM		SF, ZF, PF, OF, AF CF-undefined

Mnemonic	Meaning	Format	Operation	Flags
AAD	ASCII adjust for division	AAD		SF, ZF. PF, OF, AF CF - undefined
CBW	convert Byte to word			None
CWD	convert word to Double word			None

Logical operations :-

The logical operations AND, OR, NOT, EX-OR are used in this group w.r.to 8-bit (or) 16-bit operands.

This group also has shift, compare instructions

AND :- [logical AND]

This instruction bit by bit ~~adds~~ 'AND's' the source operand that may be an immediate, a register (or) memory location to the destination operand that may be a register (or) a memory location,

the result is placed in destination operand

eg:-

```
AND  AX, BX
AND  AX, 0010H
AND  [4000H], AX.
```

Flags:-

Flags effected are CF, OF, PF, SF, ZF

AF - undefined.

OR:- [logical OR]

It performs logical 'OR' of the two operands replacing the destination with the result.

Flag:-

CF, ZF, PF, SF, OF.

AF - undefined.

eg:-

```
OR  AX, CX
OR  AX, 0110H
OR  AX, [SUM]
OR  [1000H], AX.
```

Ex-OR:- [logical XOR]

It performs a bitwise exclusive OR of the operands and returns the result to the destination.

Flags:-

CF, ZF, PF, OF, SF. AF - undefined.

eg:-

```
XOR    AX, DX
XOR    AX, 0101H
XOR    [1001H], AX
```

NOT :-

This instruction inverts (or) complements the contents of the operand register (or) memory location bit by bit.

eg:-

```
NOT    AX
NOT    [5000H]
```

Flags :-

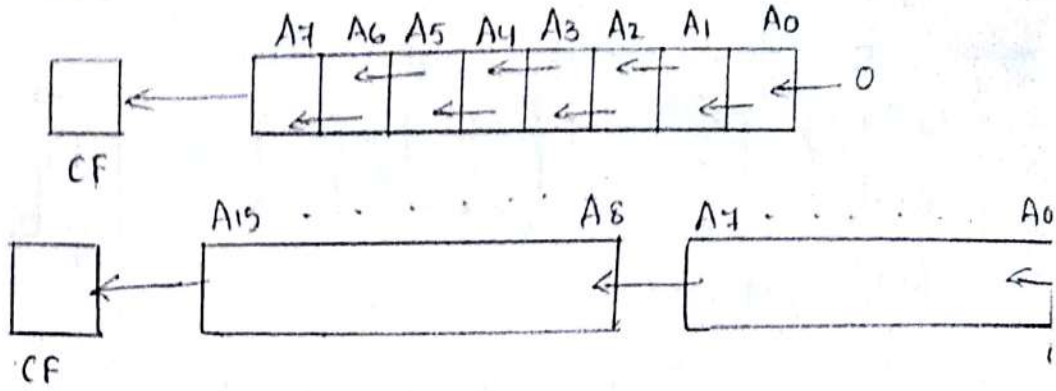
None.

Shift instruction :-

These instructions are used to shift (or) rotate the data either to left (or) right

SHL/SAL :- [Shift logical left / Shift arthematic left]

It shifts left the destination by count bits with zero's entered at the other end. The carry flag contains the last bit shifted out.



The operand may reside in a register (or) memory location. The count is either '1' (or) specified by register 'CL'

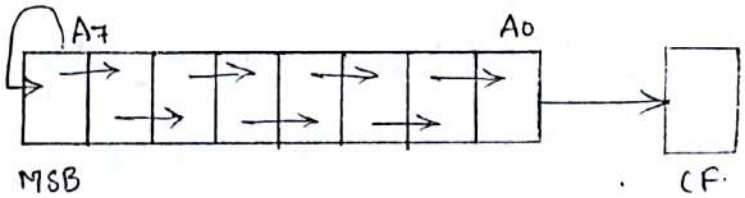
Flags effected:-

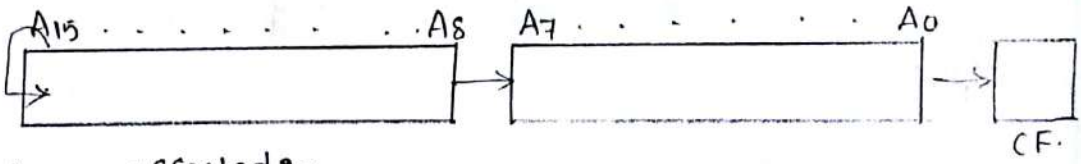
CF, OF, ZF, PF, SF

AF - undefined

SAR:- [shift arthematic right]

It Performs right shift on the operand together with sign bit





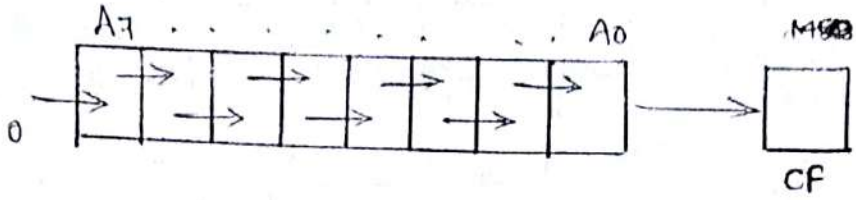
Flags effected:-

CF, OF, ZF, SF, PF.

AF - undefined.

SHR:- [shift logical right]

It shifts right the destination by count bits with '0' entered at left. The carry flag contains the last bit shifted out.



Flags effected:-

OF, CF, ZF, SF, PF

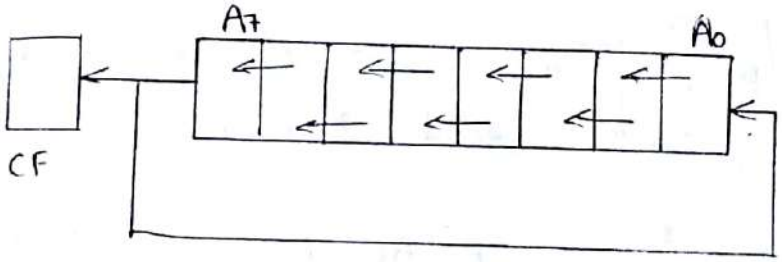
AF - undefined.

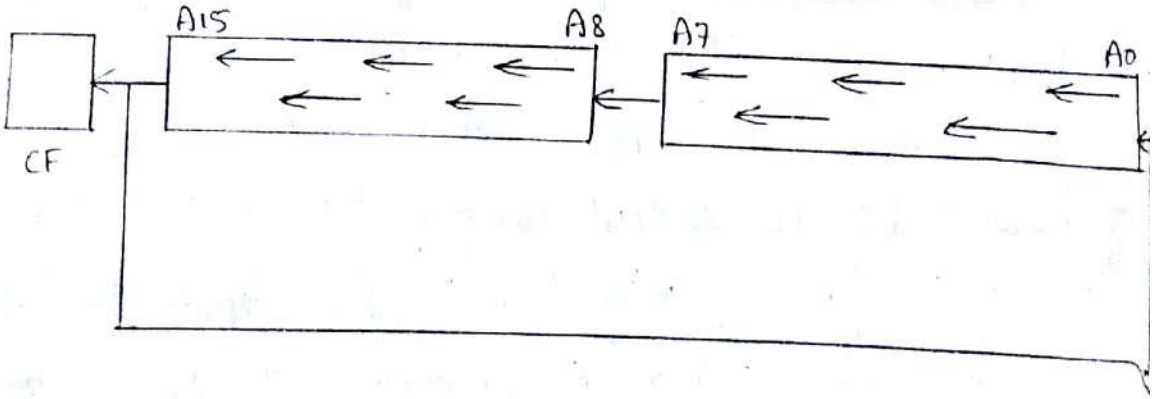
ROL :- [Rotate left (without carry)]

It Rotates the bits in the destination to the left count times with all the data shifts left by '1' bit, 'MSB' transfers to 'carry flag', as well as 'LSB'

Flags effected:-

CF, OF.



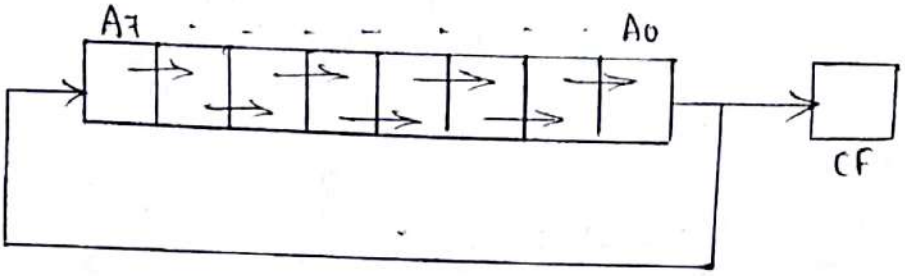


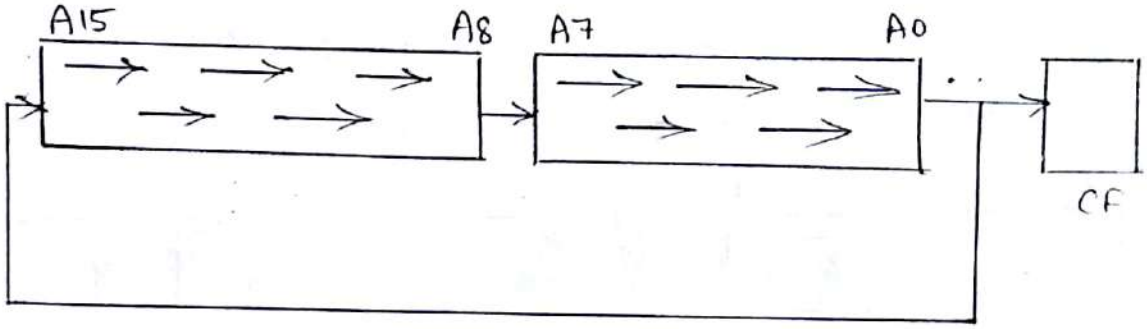
ROR :- [Rotate right (without carry)]

Rotates the bits in the destination to the right count times with each bit shifted right by '1' bit, 'LSB' transferred to 'CF' as well as 'MSB'

Flags effected :-

CF , OF



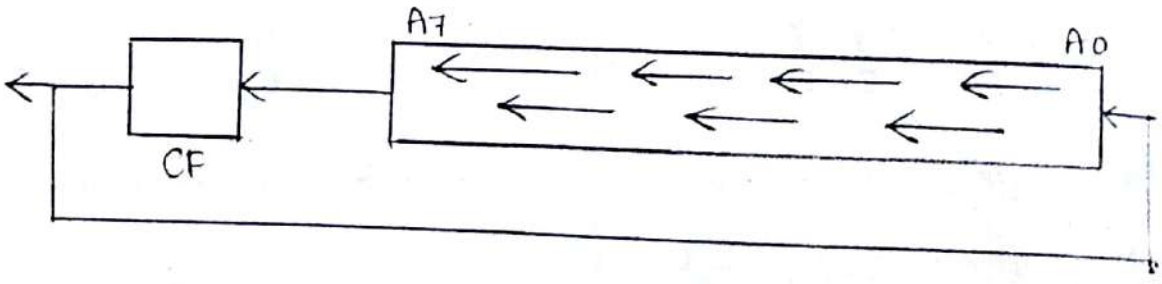


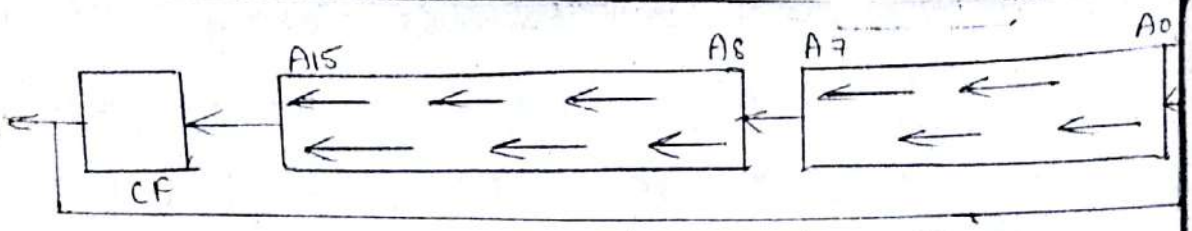
RCL :- [Rotate left through carry]

It rotates the bits in the des-tination to the left count times with each bit shifted left by '1' bit, 'MSB' transferred to 'CF' as well as 'LSB'

flags effected :-

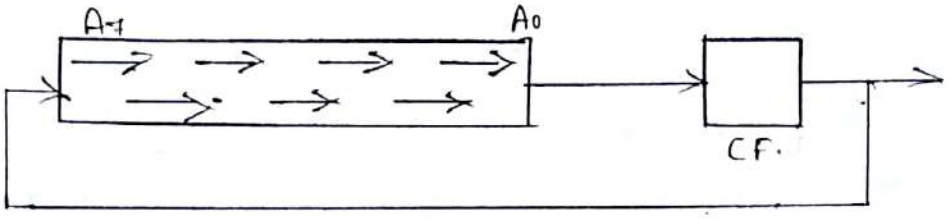
CF , OF

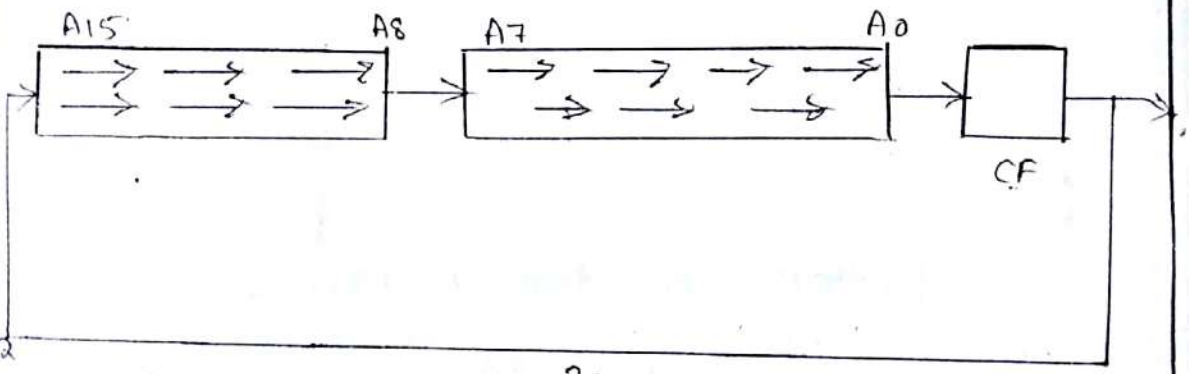




RCR;- [Rotate Right through carry]

It rotates the bits in the destination to the right count times through the carry flag with each bit shifted right by '1' bit, 'LSB' transfers to 'CF' and the content of 'CF' transfers to 'MSB'





15/12/12

TEST;- [Test opcode is Test][logical compare instruction]

It 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.

The affected flags are OF, CF, SF, ZF, PF

The operands may be Registors, memory (or) Immediate data.

eg:-

```
TEST    AX, BX
TEST    [0500H], 08H
TEST    [BX][DI], CX
```

~~String~~

Control transfer group:-

These instructions transfer the flow of execution of program to a new address specified in the instruction directly (or) indirectly. When this type of instruction is executed, the CS and IP register get loaded with new values of CS and IP corresponding to the location where the flow of execution is transfered.

(1) Unconditional Branch instructions:-

JMP:-

[unconditional Jump]

This instruction unconditionally transfers the control of execution to the specified address using 8-bit (or) 16-bit displacement.

No flags are affected by this instruction.

The Jmp may be intrasegment relative short jump. (or) Intrasegment, Direct, far Jump

eg:-

```
JMP     LABEL
JMP     DISP 8-bit
JMP     Displacement 16-bit
```

Conditional Jump Instructions :-

Mnemonic	Meaning	Condition
JA	Jump if above	CF = 0, ZF = 0
JAE	Jump if above and equal	CF = 0
JB	Jump if below	CF = 1
JBE	Jump if below (or) equal	CF = 1, ZF = 1
JC	Jump if carry	CF = 1
JCXZ	Jump if CX is zero	CX = 0
JE	Jump if equal	ZF = 1
JG	Jump if greater (signed)	ZF = 0, SF = OF
JGE	Jump if greater (or) equal (signed)	SF = OF,
JL	Jump if low (signed)	SF ≠ OF
JLE	Jump if less (or) equal (signed)	ZF = 1, SF ≠ OF
JNA	Jump if not above	CF = 1, ZF = 1
JNAE	Jump if not above (or) equal	CF = 1

Mnemonic	Meaning	condition
JNB	Jump if not below	CF = 0
JNBE	Jump if not below or equal	CF = 0, ZF = 0
JNC	Jump if no carry	CF = 0
JNE	Jump if not equal	ZF = 0
JNG	Jump if not greater (signed)	ZF = 1; SF ≠ OF
JNGE	Jump if not greater (or) equal (signed)	SF ≠ OF
JNL	Jump if not less (signed)	SF = OF
JNLE	Jump if not less and equal (signed)	ZF = 0, CF = OF
JNO	Jump if not overflow (signed)	OF = 0.
JNP	Jump if not parity	PF = 0
JNS	Jump if not signed	SF = 0.
JNZ	Jump if not zero	ZF = 0
JO	Jump if overflow (signed)	OF = 1
JP	Jump if parity	PF = 1
JPE	Jump if parity is even	PF = 1
JPO	Jump if parity is odd	PF = 0.

Mneumonic	Meaning	condition
JS	Jump if signed	SF = 1
JZ	Jump if zero	ZF = 1

The following Mneumonics perform the Same operation

JNA JNB	JNAE JC JB	JNB JNC JAE	JNBE JA	JNG JLE	JNGE JL	JNL JGE
JNLE JG	JNG JNE	JPE JP	JPO JNP	JZ JE		

19/12/12 LOOP Label ;-

LOOP LABEL:-

Decrements 'CX' by 'one' if CX ≠ 0, control transfers to label. The label operand must be within -128 (or) 127 bytes of the instruction following the loop instruction.

Flags:-

NO flags are affected.

ex:-

(CX) ← (CX) - 1;

If (CX) ≠ 0

then

(IP) ← (IP) + DISP

LOOPE/LOOPZ [loop while equal/loop while zero]

It decrements 'cx' by one, if cx and zero flag is set then control transfe to label.

ex:-

(CX) ← (CX) - 1;

If (CX) ≠ 0 and (ZF) = 1;

then

(IP) ← (IP) + DISP(8)

Flags:-

No flags are affected.

LOOPNZ/LOOPNE LABEL [loop while Not zero and loop while Not equal];

It Decrements 'cx' by 'one', if cx ≠ and zero flag is reset, the control transfers to label

ex:-

(CX) ← (CX) - 1;

If (CX) ≠ 0 and (ZF) = 0;

then

(IP) ← (IP) + DISP(8).

Flags:-

flags are affected.

CALL :-

This instruction is used to call a sub-routine from a main program. The address of the sub-routine (or) procedure may be specified directly (or) indirectly depending upon the addressing mode. If procedure (or) sub-routine is available in the same segment (±32K disp), it is known as intra segment call (or) near call.

If the procedure is outside the segment, is known as inter segment call (or) far call. This is unconditional call instruction, on execution, the incremented IP is stored on stack top (for near call) CS and IP in case of far call

Syntax:

CALL procedure NAME

((SP)) ← ((IP));

(SP) ← (SP) - 2;

(IP) ← (IP) + DISP(8);

The call instruction can be 3 bytes to 5 bytes.

[RET]:-

At each call instruction, the contents of cs and IP are pushed on to stack. At the end of the procedure, RETURN instruction must be executed to transfer control from a procedure back to the main program by getting the address stored on stack.

The size of the return instruction vary as one byte and three bytes

eg:-

(IP) ← ((SP))

(SP) ← (SP) + 2 [For near call]

(CS) ← (SP)

(SP) ← (SP) + 2 [For far call]

RET DATA 16

(IP) ← ((SP))

(SP) ← (SP) + 2 + DATA 16

INT N:- [Interrupt type N]

This instruction is used to find out the address of the interrupt service routine

INTO:- [Interrupt on overflow]

This command is executed when overflow flag is set.

IRET:- [Return from Interrupt service routine (ISR)]

It is used to transfer control back to main routine from ISR.

String Instructions:-

REP [Repeat Instruction prefix] :-

This instruction is used as a prefix to other instructions. The instruction with REP Prefix executed repeatedly until the CX register becomes zero.

When 'CX' becomes zero, the execution proceeds to the next instruction in sequence

REPE / REPZ [Repeat while operation equal (or) zero] :-

This instruction is used when the result becomes equal. When the result is equal then zero flag is going to be set.

MOVSB / MOVSW :- [Move String Byte (or) String word] → In this operand is not present

The Starting byte of the source string is located in the memory location, whose address is calculated using SI and DS.

The Starting address of the destination location is given by DI and ES. This instruction moves a string of bytes (or) words pointed to by

DS : SI Pair (source) to the memory location pointed to by

ES : DI pair (Destination).

If DF is zero (DF = 0), index registers are incremented, otherwise they are decremented

eg:- MOVSB ES : DI ← DS : SI.

SI ← SI + 1 If DF = 1

DI ← DI + 1 If DF = 0

REP MOVSB

It repeats the instruction depending on the count

LOAD SB/SW :- [load string byte (or) string word]

It transfers the string element addressed by DS : SI to the accumulator.

The size of LOAD SB/SW, MOV SB/SW } is 1 byte.

load SB

(AL) ← (DS : SI)

(SI) ← (SI) + 1; If DF = 0;

(SI) ← (SI) - 1; If DF = 1;

load SW

(AX) ← (DS : SI)

(SI) ← (SI) + 2; If DF = 0;

(SI) ← (SI) - 2; If DF = 1;

STOSB/SW :- [store string byte (or) word]

It stores value in accumulator location ES : DI

STOSB:-

(ES:DI) ← (AL)

(DI) ← (DI)+1 ; If DF = 0;

(DI) ← (DI)-1 ; If DF = 1;

STOSW:-

(ES:DI) ← (AX)

(DI) ← (DI)+2 ; If DF = 0;

(DI) ← (DI)-2; If DF = 1;

CMPS:- [compare string byte (or) string word]

[CMPSB/ CMPSW]

It subtracts the destination string value from the source, ~~the~~ without saving the results. But updates the flags based on the result of subtraction.

CMPSB:-

(DS:SI) - (ES:DI)

(DI) ← (DI)+1;

(SI) ← (SI)+1; If DF=0

(DI) ← (DI)-1;

(SI) ← (SI)-1; If DF=1

CMPSW:-

(DS:SI) - (ES:DI)

(DI) ← (DI)+2;

(SI) ← (SI)+2; If DF=0.

21/12/

(DI) ← (DI) - 2 ;

(SI) ← (SI) - 2 ; If DF = 1;

SCASB

Flags affected :-

AF, OF, CF, PF, SF, ZF

SCASB/SW :- [Scan String Byte / string word]

It compares the value at ES : DI from the accumulator and modifies the flags.

SCASB :-

(AL) - (ES : DI)

(DI) ← (DI) + 1 ; If DF = 0.

(DI) ← (DI) - 1 ; If DF = 1

SCASW :-

(AX) - (ES : DI)

(DI) ← (DI) + 2 ; If DF = 0 ;

(DI) ← (DI) + 2 ; If DF = 1 ;

21/12/12

Machine control (or) Flag manipulations group of instructions :-

CLC :-

clear carry

(CF) ← 0 ; size - 1byte.

Flags affected : CF

CMC :-

[Complement Carry]

(CF) ← (CF');

STC :-

[Set Carry]

(CF) ← 1

CLD :-

[Clear Direction Flag]

(DF) ← 0

STD :-

[Set Direction Flag]

(DF) ← 1;

CLI :-

[Clear Interrupt Flag]

(IF) ← 0;

It Disables the maskable hardware interrupts by clearing the interrupt flag. The NMI's and the software interrupts are not inhibited.

STI :-

[Set Interrupt Flag]

(IF) ← 1;

It Enables hardware interrupts.

HLT :-

[HLT] [Halt]

Halts C.P.U until Reset line is activated.

[NO operation]

This is do nothing instruction results in the occupation of both space and time

Wait (or) FWAIT :-

C.P.U enters wait state until the co-processor signals that it has finished its operation. At that instant test pin receives a high input signal.

LOCK :- [LOCK BUS]

This instruction is a prefix that causes the C.P.U assert bus lock signal during the execution of the next instruction. It is used to avoid two processors from updating the same data location.

INT num :- [Interrupt]

It initiates a software interrupt by pushing the flags, clearing the trap and interrupt flags.

FALC :- [Fills AL with carry]

AL ← 0 ; if CF=0;

AL ← FF; if CF=1;

Assembler Directives

Assembly language programming :- [A·L·P]

The language written in the form of Mneumonics is known as assembly language. It is machine dependent language. It consists of Mneumonics with different data values, additional information like assembler directives.

1) Write an A·L·P to transfer OFH and C6H to registers AL and BL.

```
MOV  AL, OFH    The data OFH is transfered to AL
MOV  BL, C6H    The data C6H is transfered to BL
HLT
```

2) Write an A·L·P to transfer the contents of Memory locations 2000H and 3000H to AH and BH, then transfer these values to memory locations 4000H and 5000H.

```
MOV  AH, [2000H]  -> It indicates address (or) Memory location
MOV  BH, [3000H]     The content of [2000H] is transfered to AH      step 1
                     The content of [3000H] is transfered to BH.    2
MOV  [4000H], AH     The Content of AH is is transfered to [4000H]  3
MOV  [5000H], BH     The content of BH is transfered to [5000H]     4
                                                                    5
```

To transfer the contents of memory locations 3000H and 4000H to AX and store them in 8000H, 8001H using register indirect addressing.

```
MOV SI, [3000H]

MOV AX, [4000H]

MOV SI, 3000H
MOV AL, [SI]
MOV SI, 4000H
MOV AH, [SI]
MOV SI, 8000H
MOV [SI], AL
MOV [SI+1], AH
HLT
```

4) Write an A.L.P to add two 8-bit numbers. The numbers are in memory locations 2000H and 2001H, then store the result in memory locations 3000H and 3001H.

Algorithm:-

Steps

1 → Clear register AH for carry.
2 → Get first data byte into AL.
3 → ADD first data byte with second data byte.
4 → check for carry. If there is no carry go to step 6.
5 → Increment AH for carry
6 → Store the result.

```
MOV   AH, 00H          [AH] ← 00H
MOV   AL, [2000H]      [AL] ← 1st byte.
ADD   AL, [2001H]      [AL] ← [AL]+[2000H]
                              [AL] + 1st byte
JNC   LAST
INC   AH
LAST: MOV  [3000H], AL
      MOV  [3001H], AH
      HLT.
```

22/12/12

5) Write an A.L.P to add a word type data located at the offset address 0800H and 0801H in segment address 3000H to another word type data located at the offset address 0700H and 0701H in the same segment. Store the result at the offset address 0900H and 0901H in the same segment. Store the carry generated in the addition in the same segment at the offset address 0902H.

opcode	operand	comment
MOV	AX, 3000H	Initialize DS with Segment address 3000H.
MOV	DS, AX	
MOV	AX, (0800H)	Move the first dataword to AX
ADD	AX, (0700H)	ADD AX with second data word.
MOV	[0900H], AX	LSB result is stored at offset addresses [0900H] & [0901H]

JC CC — If carry equal to 1 jump to address CC

MOV [0902H], 00H — If there is no carry store 00H at [0902H]

JMP END — jmp to the address END

CC: MOV [0902H], 01H — Store 01H at the offset address [0902H]

END HLT — Stop.

6) Write an A.L.P to subtract a word in memory location 0500H from a word in memory location 0600H. The segment address for these two words is 4000H. Store the result in location 0400H.

```
MOV AX, 4000H
MOV DS, AX
MOV AX, [0600H]
SUB AX, [0500H]
MOV [0400H], AX
HLT
```

7) Write an A.L.P to find the Smallest word in an array of 100 words stored sequentially in memory, starting at the offset address 1000H in the segment address 5000H. Store the result at the offset address 2000H in the same segment (or) MOV AX, FFFFH

```
MOV CX, 99      - Initialize CX with no. of comparisons (100-1) (99 comparisons)
MOV AX, 5000H
MOV DS, AX      - Data segment is initiated
MOV SI, 1000H   - source address is in reg SI
```

```
MOV AX, [SI]        - Move the first word to AX
START: INC SI       - Increment SI twice to
       INC SI         point to the next word.
       CMP AX,[SI]  - compare next word with
       JNC            word in AX.
       JC  RPT REPEAT - If AX is smaller jump
                        to the address repeat.
       MOV AX, [SI]   - Replace the word in AX
                        with the smaller word.
REPEAT: LOOP START     - Repeat the operation
                         from start.
       MOV [2000H], AX - Smallest no. is stored in
                         2000H.
       HLT.
```

7) Write an A.L.P to find smaller in two numbers

```
MOV AX, 2500H
MOV DS, AX
MOV SI, 1055H
MOV AX, [SI]
CMP AX, [1025H]
JC  BSR
MOV AX, [1025H]

BSR: MOV [1000H], AX
     HLT
```

(or)

```
MOV AX, 0500H
MOV DS, AX
MOV AX, [0200H]
CMP AX, [0300H]
JC  xyz
MOV AX, [0300H]
MOV [0400H], AX
HLT
```

9)

8) Write an A.L.P to find larger in two no. The two no. (or) words (16 bits) are in memory locations 0500H, 0502H. Store the result in 0503H, 0504H. The segment starting address is 6000H.

```
MOV AX, 6000H
MOV DS, AX.
MOV SI, [0500H]
MOV AX, [SI]
INC [SI]
INC [SI]
CMP AX, [SI]
JNC KLA
MOV AX, [SI]

KLA : INC [SI]
      INC SI
MOV [SI], AX
HLT.
```

(or)

```
MOV AX, 0500H
MOV DS, AX
MOV AX, [0200H]
CMP AX, [0300H]
JNC XYZ
MOV AX, [0300H]
MOV [0400H], AX
HLT
```

9) Write an A.L.P to find the largest number among series of 10 words. The words array is starting from the memory address 0808H. The segment is starting from 0505H.

```
MOV CX, 09
MOV AX, 0505H.
MOV DS, AX.
MOV SI, [0808H.
MOV AX, [SI]
Start : INC SI.
        INC SI
CMP AX, [SI]
```

(or)

```
MOV AX, 0000H
MOV CX, 50
MOV SI, 1000H
START : CMP AX, [SI]
        JNC CC
        MOV AX, [SI]
CC : INC SI
     INC SI
     DEC CX
     loop START
     MOV [0300H], AX
     HLT
```

```
JNC   Man

MOV [SI], AX
Mov AX, [SI]

Man: Loop start
  MOV [2200H], AX
  HLT
```

10) Write an ALP to add 50 words. The array of words is starting from memory location 0100H store the result in memory location 0200H, 0201H, 0202H The segment is starting from 7700H

11)

```
     CX, 49
MOV  SI, 0000H
     AX, [    ]
MOV  DS, AX
MOV  SI, 0100H
MOV  AX, [SI]
     INC SI
     INC SI
     AX, [SI]
JNC  PVM
     JNC BL

PVM: LOOP    ]
MOV  [0200H], AX
MOV  [    ], BL
HLT
```

Algorithm:-

1) Get count in register CX. → clear register BL to hold carry
2) Initialize data segment
3) Get 1st source address in SI.
4) Get 1st no. in array in AX.
5) ADD next no. in array with AX.
6) Check for carry.
7) If there is no carry, goto step9.
8) Increment BL.
9) Decrement count by '1'. → Increment SI
10) If count ≠0, go to step 5.
11) Store the result.

11) Write an A.L.P for the above program using addition with carry [ADC]

```
MOV CX, 49
MOV AX, 7107H
MOV DS, AX
MOV SI, 0100H
MOV AX, [SI]
Start: INC SI
       INC SI
       ADC AX, [SI]
       LOOP start
       MOV [0200H], AX
       HLT
```

12) write an A·L·P to find 1's complement of a word in location 1234H. Store the result in 2345H. segment address is 34567H.

```
MOV AX, 3456H
MOV DS, AX
MOV SI, 1234H  -> MOV AX [SI]
MOV [2345H], AX
```

13) write an A·L·P to find 2's complement of a word in location 1234H, Store the result in 2345H. segment address is in 3456H

```
MOV AX, 3456H
MOV DS, AX
MOV SI, 1234H
MOV CX, [ ]
NEG [AX]
MOV [2345H], AX
HLT
```

(or)

```
MOV BL, 00H
NOT AL
ADD AL, 01H
INC XYZ
INC BL
XYZ MOV [2000H], AL
MOV [2001H], BL
HLT
```

for byte

```
MOV AX, 5000H
MOV DS, AX
MOV AL, [1000H]
NEG AL
MOV [2000H], AL
HLT
```

14) write an A·L·P to transfer 100 bytes starting from 1000H to 2000H onwards.

```
MOV CX, 100
MOV AX, 2772H
MOV DS, AX
MOV SI, 1000H
MOV DI, 2000H
Start: MOV AL, [SI]
MOV [DI], AL
INC SI
INC DI
LOOP START
HLT.
```

15) Write an A·L·P for word manipulation. '4F' is present in memory location 1000H. '2F' is present in memory location 2000H store 'FFH' in 3000H

```
MOV AX, 3838H
MOV DS, AX.
MOV AL, [1000H]
AND AL, 0FH.
MOV BL, [2000H]
AND BL, 0F1H.
ROL
ROL
ROL
ROL                  → (AL ← F0).
ADD AL, BL
MOV [3000H], AL.
HLT
```

```
  4F
  0F  - AND
  0F.

  2F
  0F  AND
  CF

  00001111
  11110000
  F  0

  F0
  0F  - ADD
  FF
```

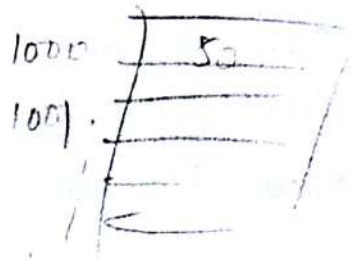
16) Write an A·L·P for word manipulation
'ED' is present in memory location 7000H
'C6' is present in memory location 5000H
Store 'BEN' 'DCH' in '66H.'

```
MOV AX, 9999H
MOV DS, AX.
MOV AL, [7000H]
AND AL, 0DH.
MOV BL, [5000H]
AND BL, C0H.
ADD AL, BL
ROL
ROL
ROL
ROL
MOV [0066H], AL
HLT
```

```
DC

ED
C6
C0  AND
C0

FD
0D  AND
0D

CD
C0
CD
ROL
DC
```

18) larger among the series of no words the no. of words are present in memory location 1000H.

```
    MOV  CX,[1000H]
    MOV  AX,[5000H]
    MOV  DS, AX
    MOV  SI, [1001H]
    MOV  AX, 0000H
START: CMP  AX,[SI]
    JNC   REPEAT
    MOV  AX,[SI]
REPEAT: INC SI
    INC SI
    LOOP START
    MOV [0300H], AX
    HLT
```



19) Find 1's comp of a number in location 1000H. Store the result in 2000H. The starting address of the segment is 5000H

```
    MOV  AX, 5000H
    MOV  DS, AX
    MOV  SI, 1000H
    MOV  AX,[SI]
    NOT [AX]
    MOV [2000H], AX
    HLT
```

```
    MOV  AX, 5000H
    MOV  DS, AX
    NOT [1000H]
    MOV  AL,[1000]
    MOV [2000H], AL
    HLT
```

20) Write an A.L.P for 1's complement of word in 1000H. Assume remaining data.

```
MOV AX, 5000H
MOV DS, AX.
MOV AX, [1000H]
MOV NOT AX
MOV [2000H], AX
HLT
```

21) Write an A.L.P for 2's complement of word

```
MOV AX, 5000H
MOV DS, AX
MOV AX, [1000H]
NEG AX
MOV [2000H], AX
HLT
```

22) Write an A.L.P for Sum of series of 16-bit numbers (or) words. The count is in memory location 2000H. The array starts from the memory location 3000H Store the result in memory location, 4000H, 4001H., 4002H The starting address of the segment is 5000H.

```
MOV BL, 00H
MOV CX, [2000H]
MOV AX, 5000H
MOV DS, AX
MOV AX, 0000H
MOV SI, 3000H
```

```
START: ADD AX, [2000H]
       JNC BSR
       INC BL

BSR    INC SI
       INC SI

       LOOP START
       MOV [4000H], AX
       MOV [4002H], BL
       HLT
```

25) Write an ALP for block data transfer.

The block of data words are stored in memory from the address 2000H onwards. Transfer all these data words to memory locations starting from 5000H onwards. The count is in memory location 1000H. Segment address is 8000H

```
    MOV CX, [1000H]
    MOV AX, 8000H
    MOV DS, AX
    MOV SI, 2000H
    MOV DI, 5000H
PVM: MOV BX, [SI]
    MOV [DI], BX
    INC SI → INC SI
    INC DI
    INC DI
    LOOP PVM
    HLT
```

24) Write an A.L.P using bubblesort technique for the given data in the specified memory locations

```
MOV CX, 04H
MOV AX, 536GH
MOV DS, AX
MOV SI, 1000H
MOV AX, [SI]
Start: INC SI
CMP AX, [SI]
JNC ELE

MOV AX, [SI]

ELE: loop start

MOV CX, 03H
MOV AX, [SI]
CMP AX, [SI]
JNC
```

```
data DB {  } 100 256 178 154 296
count DB 5
END Data segment

Assume
MOV AX, data
MOV SI, offset (Data)
MOV CX CX-1
DS, AX
AX [SI]
CMP AX, [SI]+2
JH BSR
X [SI+2], AX
BS X [SI], AX
JNC
ADD SI, 02H
Dec CX
loop start
smallest no.
Dec DX
```

I 100 256 178 154 296

256 100 178 154 296

256 178 100 154 296

256 178 154 100 296

256 178 154 296 100

II 256 178 154 296 100

256 178 154 296 100

256 178 296 154 100

256 178 296 154 100

III 256 178 296 154 100

256 296 178 154 100

256 296 178 154 100

256 296 178 154 100

IV 296 256 178 154 100

296 256 178 154 100

296 256 178 154 100

296 256 178 154 100

24

```
me CS:CODE , DS:DATA
'A SEGMENT
Array DB 1721H, 2772H, 8552H, 5366H :
                    Enter all the data items of array
COUNT EQU 04H
DATA ENDS
'.C SEGMENT
start : MOV AX, 3000H .
        MOV DS, AX
MOV CL, COUNT : load the no. of data items in CL
DEC CL : Decrements CL as the no. of passes is 1
          less than no. of data
BSC : MOV BL, CL : Initialize BL with no. of Comparisons
                    to be done in each pass
MOV SI, offset (1500H): Move the offset address into SI
LLL : MOV AX, [SI] : MOVe one of the data items into AX
      ADD SI, 02H : Adds 02 to SI to point to nxt data
      CMP AX, [SI] : Compares the nxt data item with the
                     Content of AX
      JC CC . If the 1st data item is less than 2nd
               then go to CC
      XCHG AX, [SI] ; else, exchange the data in AX
                      & (and) the memory
      SUB SI, 02H : Subtract 02 from SI to point to
                    previous memory location.
      MOV [SI], AX : store the content of Ax in the memory
      ADD SI, 02H : Increment SI by 2 to Compare the
                    next data with AX
      DEC BL : Decrement the no of Comparisons
               in BL by 1
```

JNZ ELE . If BL ≠ 0. go to compare for nxt Comparison

LOOP ABC : If BL = 0, go to ABC

MOV AX, [SI]

Code Ends

HLT

1500 → 1724
1502 - 2772
1504 - 8552
1506 - 5366.

(n-1) Passes

COUNT: 04H

CL = 04H

CL: 03H

BL: 03H

SI = 1500

ELE : AX = 1724.

SI = 1502

Compare: 1724, 2772

JC : CC

XCHG AX, [SI]

SUB SI, 02H;

MOV [SI], AX

ADD SI, 02H

CC DEC BL MOV AX, [SI]

BL : 02H

JNZ: ELE BL ≠ 0

LOOP ABC.

BL : 0

MOV [AX/SI]

I.

1724 2772 8552 5366
2772 1724 8552 5366
2772 8552 1724 5366
2772 8552 5366 1724

II.

8552 2772 5366 1724
8552 5366 2772 1724
8552 5366 2772 1724

Sorting:

Write an ALP to arrange given series of words in descending order.

```
ASSUME CS: code, DS: Data
DATA SEGMENT
NUMBS DW 1124H, 2732H, 8552H, 5366H, 9999H
COUNT EQU 05
DATA ENDS:

CODE SEGMENT

START : MOV AX, Data } Initialization of data segment
        MOV DS, AX
        MOV DX, COUNT - 1
REPEAT2 : MOV CX, DX
        MOV SI, OFFSET NUMBS.
REPEAT1 : MOV AX, [SI]
        CMP AX, [SI+2] compare two no. in array
        JH CC          If no. in AX is greater
        XCHG [SI+2], AX  then jump to the address
        XCHG SI, AX
XXX CC : ADD SI, 02 ; Increment SI to point
                      to the next number
        LOOP REPEAT1  in the array.
        DEC DX
        JNZ REPEAT2
        Code Ends
        END START.
        END
```

25) Write an A.L.P to sort an array in ascending order.

```
ASSUME CS: code, DS: data
 DATA  SEGMENT
NUMBS  DW  1724H, 2772H, 8552H, 5366H, 0026H
 COUNT  EQU 05
 DATA  ENDS
CODE  SEGMENT
START: MOV AX, data  } ; Initialization of data segment
       MOV DS, AX    }
       MOV DX, COUNT-1
REPEAT2: MOV CX, DX
         MOV SI, OFFSET NUMBS
REPEAT1: MOV AX, [SI]
         CMP AX, [SI+2] ; Compare two no. in array
         JL CC          ; If no. in AX is greater
         XCHG [SI+2], AX  than jump to the address
         XCHG SI, AX
CC :     ADD SI, 02 ; Increment SI to point to
                      the next number in the
                      array
         LOOP REPEAT1
         DEC DX
         JNZ REPEAT2
         code Ends
         END START.
```

24. Write an A.L.P to find out the no. of even and odd numbers from a series of bytes

```
ASSUME CS:code, DS:data
  DATA SEGMENT
  NUMBS DW: 0000H, 0001H, 0010H, 0011H, 0100H
   COUNT EQU 05
   DATA ENDS
CODE SEGMENT
START: MOV CX, COUNT
       MOV AX, Data
       MOV DS, AX
       MOV BX, 00H }(or)  XOR BX BX
       MOV DX, 00H }      XOR DX, DX
       MOV SI, offset NUMBS
 BSR:  MOV AX, [SI]
       RCR AX, 01
       JC ABC
       INC DX
       Jmp last
ABC :  INC BX
last : LOOP BSR > ADD SI, 02H
       MOV [2500H], DX
       MOV [7964H], BX
code     Ends
         END START
         END
```

24) Write an A.L.P to find positive & negative numbers in series of array.

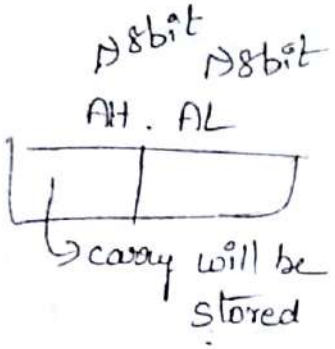
```
ASSUME CS:code, DS: data
  DATA SEGMENT
NUMBS DW : 0010H, 1011H, 1100H, D011H
   COUNT EQU 4
   DATA ENDS
CODE SEGMENT
START : MOV CX, COUNT
        MOV AX, data.
        MOV DS, AX.
        MOV BX, 00H ✓ Neg
        MOV DX, 00H ✓ Pos
        MOV SI, offset NUMBS
        MOV AX, [SI]
        RCL AX, 01
        JS Nandu
          INC DX.
          Jmp last
Nandu :   INC BX
last  :   loop CK
          mov [2564H], DX
          MOV [9876H], BX
code  :   Ends
          END START
          END.
```

MOV CX, count
LEA BX, add L
MOV DS, BX
LEA SI, Nats
MOV DX, 00
MOV AL [SI]
JNLL [SI]
ADD DX, 1
INC BX
INR SI
DEC CX
loop Skel
HLT

28) Write an A.L.P for the addition of two 8-bit without checking jump instruction & carry flag.

Ans

```
ASSUME CS: code, DS: data
  DATA SEGMENT
NUMBS DW : 003FH, 00F6H
  DATA ENDS.
 CODE SEGMENT
start: MOV Ax, data
  MOV DS, Ax
  MOV SI, offset (data)
  ADD MOV AX, [SI]
  ADD AX, [SI+2]
  MOV [2772H], AX
  code Ends
  END START
  END
```



27/12/17

29) Write an A.L.P for two 32-bit numbers addition.

```
ASSUME CS: code, DS: data
  DATA SEGMENT
NUMB1 DD: 26FD0123H
NUMB2 DD: C210EA11H
  RESULT DB 9 DUP(?)
  DATA ENDS
    CODE SEGMENT
  MOV Ax, Data      XOR BL, BL
  MOV DS, Ax  → MOV DX, 00H
  MOV SI, offset (NUMB1)
  MOV Ax, [SI+4]
  MOV DI, OFFSET NUMBER2
  MOV DX, [DI+4]
```

```
        MOV [RESULT+4], AX
        MOV AX, [SI]
        ADDC AX, [DI]
        JNC UMA
        MOV BL, 0001H
UMA:    MOV [RESULT], AX
        MOV [RESULT+8], BL
CODE    ENDS
        END start
```

29) Write an ALP

30) Write an A.L.P for 8-bit Multiplication

```
ASSUME CS:code, DS:Data
DATA SEGMENT
NUMB1 DB : F2H
NUMB2 DB : C6H
RESULT DB 2 DUP[?]
DATA ENDS
CODE SEGMENT
    MOV AX, DATA
    MOV DS, AX
    MOV AL, NUM1
    MOV BL, NUM2
    MUL BL
    MOV [RESULT], AX
CODE ENDS
    END
```

31) Write an A.L.P for the above program with-out using MUL instruction.

```
ASSUME CS:code, DS:data
    DATA SEGMENT
    NUMB1 DB F2H
    NUMB2 DB C6H
    COUNT EQU 66
    RESULT DB 2 DUP[?]
    DATA ENDS
    CODE SEGMENT
start   MOV CX, COUNT
        MOV AX, DATA
        MOV DS, AX → MOV BX, 00H
        MOV AL, 0F2H
Repeat: ADD AL, 0C6H
        DEC CX
        LOOP Repeat
        MOV BX, AX
        Code Ends
        End start.
```

33)

34)

28/12/12

32) Write an A.L.P for 16 bit multiplication. Multiplicand is in memory location 1100H, multiplier is in memory location 1200H. Store the result in memory location 1300H and 1302H. The segment address starts from 3000H.

```
MOV AX, 3000H.
MOV DS, AX
MOV AX, [1100H]
```

```
MOV  BX, [1200H]
MUL  BX
MOV  [1300H], AX
MOV  [1302H], DX
HLT.
```

33) Write an A.L.P for division where size of the dividend is 32 bits, and the size of the divider is 16 bits.

```
MOV. AX, 3000H
MOV  DS, AX
MOV  AX, [1100H]   } Dividend is in DX, AX
MOV  DX, [1102H]   }
MOV  BX, [1200H]   - divisor is in 1200
DIV  BX
MOV  [1300H], AX
MOV  [1302H], DX
HLT.
```

34) Write an A.L.P for division without using DIV Instruction.

```
MOV  AX, 3000H
MOV  DS, AX.
MOV  AX, [1100H]
MOV  DX, [1200H]
XOR  BL, BL
loc1: CMP  AX, DX
JC(OR)JL : Teja
        SUB  AX, DX
        INC  BL
        Jmp  loc1
Teja:  MOV [2772H], BL
       MOV [0011H] AX
       HLT
```

35) Write an A.L.P to compute $\sum_{i=1}^{n} x_i y_i$ where n = 100, x and y are signed 8 bit numbers The address of x_i is 1000H, The starting address of y_i is 2000H. Store the result in [3000H] and 3001H. Segment address starts from 4000H.

```
MOV  AX, 4000H
MOV  DS, AX
MOV  CX, 100
MOV  SI , 1000H
MOV  DI , 2000H
XOR  BX , BX
BACK: MOV  AL , [SI]
      IMUL [DI]
      ADD  BX, AX
      INC  SI
      INC  DI
      Loop BACK
      MOV [3000H] , BX
      HLT
```

36) Write an A.L.P for addition of two 8-bit numbers without using jump instruction (or) without checking carry flag (Back) [already written]

36) Write an A·L·P for B·C·D addition, BCD numbers are stored as bytes in two memory locations. First number is in memory location 1000H, second number is in memory location 2000H

92
59

```
MOV AX, 2772H
MOV DS, AX
XOR BL, BL
MOV AL, [1000H]
ADD AL, [2000H]
DAA
JNC L1
INC BL
L1 MOV [2500H], AL
   MOV [2501H], BL
   HLT
```

```
  1 0 0 1 0 0 1 0
  0 1 0 1 1 0 0 1
  ---------------
  1 1 1 0 1 0 1 1
  ---------------
  0 1 1 0   0 1 1 0
  ---------------
(1) 0 1 0 1   0 0 0 1
  ---------------
  1     5       1
```

37) Write an A·L·P for BCD subtraction.

```
MOV AX, 8552H
MOV DS, AX
MOV AL, [1500H]
XOR BL, BL
SUB AL, [1700H]
DAS
JNC L1
INC BL
L1: MOV [1800H], AL
    MOV [1100H], BL
    HLT
```

25 - m
13 - s
12

10's complement of 8
9's comp +1

40) Write an A.L.P for word manipulation

The byte 45H is in memory location, 0200H, and the byte FCH is in memory location 0300H. Store the byte C5H in the memory location 0400H. segment starts from 0500H.

```
MOV AX, 0500H
MOV DS, AX
MOV AL, [0200H]
AND AL, 0FH
MOV BL, [0300H]
AND BL, 0FH
ROL BL, 04
ADD AL, BL
MOV [0400H], AL
HLT
```

```
45
0F

0100 0101
0000 1111
0000 0101
   0    5

FC
0F
0C

05
0C
C5
```

41) Write an A.L.P for word manipulation

C645H
21FCH
FF52

```
MOV AX, 0500H
MOV DS, AX
MOV AX, [0200H]
AND AX, 0F0FH
MOV DX, [0300H]
AND DX, F0F0H
ROL AX, 04
ROL DX, 04
ADD AX, DX
MOV [0400H], AX
HLT
```

```
C645H
0F0F
0605

21FCH
F0F0
20F0

0605
ROL - 5060
SHL - 0506
ROL - 6050
SHL - 0605
```

12) Write the code for statement if

If (ax < bx) then x = -1;
else x = 1

```
Cmp   AX, BX
JNC(oR) JNL  loc1
mov   x, -1
   Jmp   loc2
loc1 : mov x, 1
loc2 : HLT
```

If then
else

13) Write a program code that sets the word x to zero, if (x > word xmax) then x = 0.

```
MOV  Ax, [x]
CMP  Ax, [xmAx]
JC   last
MOV [x], 0

last : HLT
```

If then

14) Write the code for statement

If (A < B) and (B >= 14) then x = A;
else x = B.

```
MOV  AX, [A]
CMP  AX, [B]
JC   loc1
   Jmp  last

loc1   MOV Ax, [B]
       MOV [x], Ax
```

If Then
else

last: CMP [B], 14

JGE CC

Jmp last1

CC: MOV [B], 14

last1: MOV [x], AX

MOV [x], B

HLT

45) Write

If (A >= 14) OR (B <= 4) (B: 17) AND (C: - 5)

then X = X+1

46) If (X > 5) and (Y < 0) OR (Z <= 13) Then A = 10
else A = 2

47) Write an A.L.P to evaluate the code

$$X = \frac{(A * B) + (C * D)}{E}$$

, where the values of variables are in bytes.

```
MOV AL, [A]
MOV BL, [B]
MUL BL
MOV DX, AX
MOV AL, [C]
MOV BL, [D]
MUL BL
ADD AX, DX
MOV BL, [E]
DIV BL
MOV X, AX
HLT
```

48) Write an A.L.P to decide whether the Parity of a given no. is even or odd. If Parity is even set DL to 00H, else set DL to 01H. The given number is a multibyte no.

Note:-

Add the multibyte no. with 00H in byte order. The result of the addition reflects the parity of that byte of the multibyte number.

Adding the parities of all the bytes of the number, overall parity can be determined.

```
ASSUME  CS: code,  DS: Data
   DATA  SEGMENT
 NUMB  DD  1F2C3B16H
   COUNT  EQU  04
     DATA ENDS
  CODE  SEGMENT

start :  MOV AX, data
         MOV DS, AX
         MOV DH, count
         XOR BL, BL
         MOV CL, 00H
         MOV SI, offset NUMB
loc2 :   ADD BL, [SI]
         JP  loc1
          INC CL
loc1 :    INC SI
          MOV BL, 00H
           DEC DH
          JNZ LOC2
```

This program is wrong

```
        MOV  DL,00
        RCR  CL,1
        JNC  LOC3
        INC  DL
LOC3:   CODE ENDS
        END start
```

49) Write a program to convert the 8-bit Packed BCD number stored in the memory location 3000H:2000H into a binary number and store it in the offset address 2001H in the same segment.

~~Binary to BCD~~

```
MOV  AX, 3000H
MOV  DS, AX
MOV  AL, [2000H]
MOV  BL, AL      - store a copy of original no. in BL
AND  AL, 0FH
AND  BL, F0
ROR  BL, 04
XCHG AL, BL
MOV  CL, 10  (or) MOV CL, 0AH
MUL  CL
ADD  AL, BL
MOV  [2001H], AL
HLT
```

BL

62

51) Write an A.L.P to do move a word string 200 bytes (100 words) long from the offset address 1000H to the offset address 3000H in the segment 5000H.

```
MOV   AX, 5000H
MOV   DS, AX
MOV   ES, AX
MOV   CX, 100
MOV   SI, 1000H
MOV   DI, 3000H
CLD
REP:  MOV SW
HLT
```

2/1/12

52) Write a Procedure named Square that Squares the content of BL and Places the result in BX. Assume that this Procedure is called from the same code segment.

```
SQUARE   PROC   NEAR
         PUSH   AX
         MOV    AL, BL
         MUL    BL
         MOV    BX, AX
         POP    AX
         RET
SQUARE   END    PROC
```

3) Write an A.L.P to find square root of a 2 digit number. Assume that the number is perfect square

```
ASSUME  CS: code,  DS: Data
    DATA SEGMENT
    NUMB  EQU 25
    RESULT  DB(?)
    Data  Ends
CODE  SEGMENT
Start:  MOV  AX, Data
        MOV  DS, AX
        MOV  CL, NUMB
        MOV  BL, 1
        MOV  AL, 00
UP:     CMP  CL, 0
        JZ, last
        SUB  CL, BL
          INC AL
        ADD  BL, 2
          Jmp UP
last:     MOV Result, AL
          Code Ends
          End start.
```

54) Write an A.L.P to find the addition of 1's complement of a no. to the 2's comp of another no.

```
MOV AX 5000H
MOV DS, AX
MOV SI, 2000H.
MOV AL, [SI]
NOT AL.
MOV BL, 00
MOV DI, 2102H.
MOV BH, [DI]
NEG BH
ADD AL, BH.
JNC BC
    INC BL
BC  MOV 4000H, AL
    MOV 4001H, BL
    HLT.
```

03/01/12.

55) Write an A.L.P for the addition of two 3×3 matrices, the matrices are stored in the forwm of lists. Then store the addition of result in 3rd list.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
ASSUME CS: code, DS: Data
Data segment
COUNT EQU 09H
```

```
MAT1  DB   01, 02, 03, 04, 05, 06, 07, 08, 09
MAT2  DB   09, 08, 07, 06, 05, 04, 03, 02, 01
MAT3  DW   09 DUP(?)
DATA  Ends
CODE  SEGMENT
Start: MOV  Ax, Data
       MOV  DS, Ax
       MOV  CX, count
       MOV  SI, offset math1
       MOV  DI, offset math2
       MOV  Bx, offsel Math3
       XOR  Ax, Ax.
NEXT : MOV  AL, [SI]
       ADD  AL, [DI]
       MOV  WORD PTR (Bx), Ax.
       INC SI
       INC DI
       ADD  Bx, 02
       loop NEXT
       code Ends
       End start
```

56) Write an A.L.P for the multiplication of two 3x3 matrices, the matrices result are stored in the form of lists and then store the result in another list.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 \\ 1 & 2 & 3 \\ 4 & 6 & 5 \end{bmatrix}$$

```
ASSUME  CS: code, DS: Data
    Data Segment
    count1 EQU 27
    count2 EQU 18

Mat1 DB  01, 02, 03, 04, 05, 06, 07, 08, 09
Mat2 DB  09, 07, 06, 05, 03, 04, 02, 01, 08
Mat3 DW  09 (?)
    Data ends
    CODE SEGMENT

start: MOV AX, Data
       MOV DS, AX
       MOV CX, count1
       MOV BX, count2

  MOV SI, offset Mat1
  MOV DI, offset Mat2
  MOV DX  Mat3.
```

Macro:- [Definition of Macro]

A macro is a group of statements under one name. This concept is used when the no. of instructions are repeating through in the main program.

A macro can be defined anywhere in a program using the directives

MACRO
&
ENDMACRO.

The label prior to the MACRO is the macro name which should be used in the actual program.

The end macro directive marks the end of the instruction (or) statement sequence assigned with the macroname.

```
DISPLAY  MACRO
         MOV AX, ADDR
         MOV DS, AX
         MOV DX, OFFSET MSG
         MOV AH, 09H
         INT 21H
ENDM
```

The above definition of macro assig
the name display to the instruction
sequence between the directives MACRO
and the ENDM.

While ascending assembling the
above sequence of instructions will repla
the label display, whenever it appears in
the program.

A macro may be defined in
another, macro (or) inother words, a mac
may be called from inside a Macro.
This type of Macro is called nested mac

—— X ——

04/01/12

Macro:- [Definition of Macro]

A macro is a group of statements under one name. This concept is used when the no. of instructions are repeating through in the main program.

A macro can be defined anywhere in a program using the directives

MACRO
&
ENDMACRO.

The label prior to the MACRO is the macro name which should be used in the actual program.

The end macro directive marks the end of the instruction (or) statement sequence assigned with the macroname.

```
DISPLAY  MACRO
         MOV AX, ADDR
         MOV DS, AX
         MOV DX, OFFSET MSG
         MOV AH, 09H
         INT 21H
ENDM
```

21/1/2013

UNIT-3

I/O Interface :- 8255 PPI, various modes of operation and Interfacing to 8086, Interfacing keyboard, display, Stepper motor interfacing, D/A and A/D converter.

→ Port is a place where data is loaded or unloaded by microprocessor.

→ The port can be In-port or out-port.

→ The µp takes the data from i/p device using In-port. for example, keyboard is the i/p device to the computer which is interfaced using In-port.

→ The µp writes to the o/p device using out-port. for example, CRT is the o/p device which is connected to the computer.

→ The In-ports should not draw more current. So Tristate buffers are used when not in use. The out-ports should be able to drive more number of devices so they should source more current.

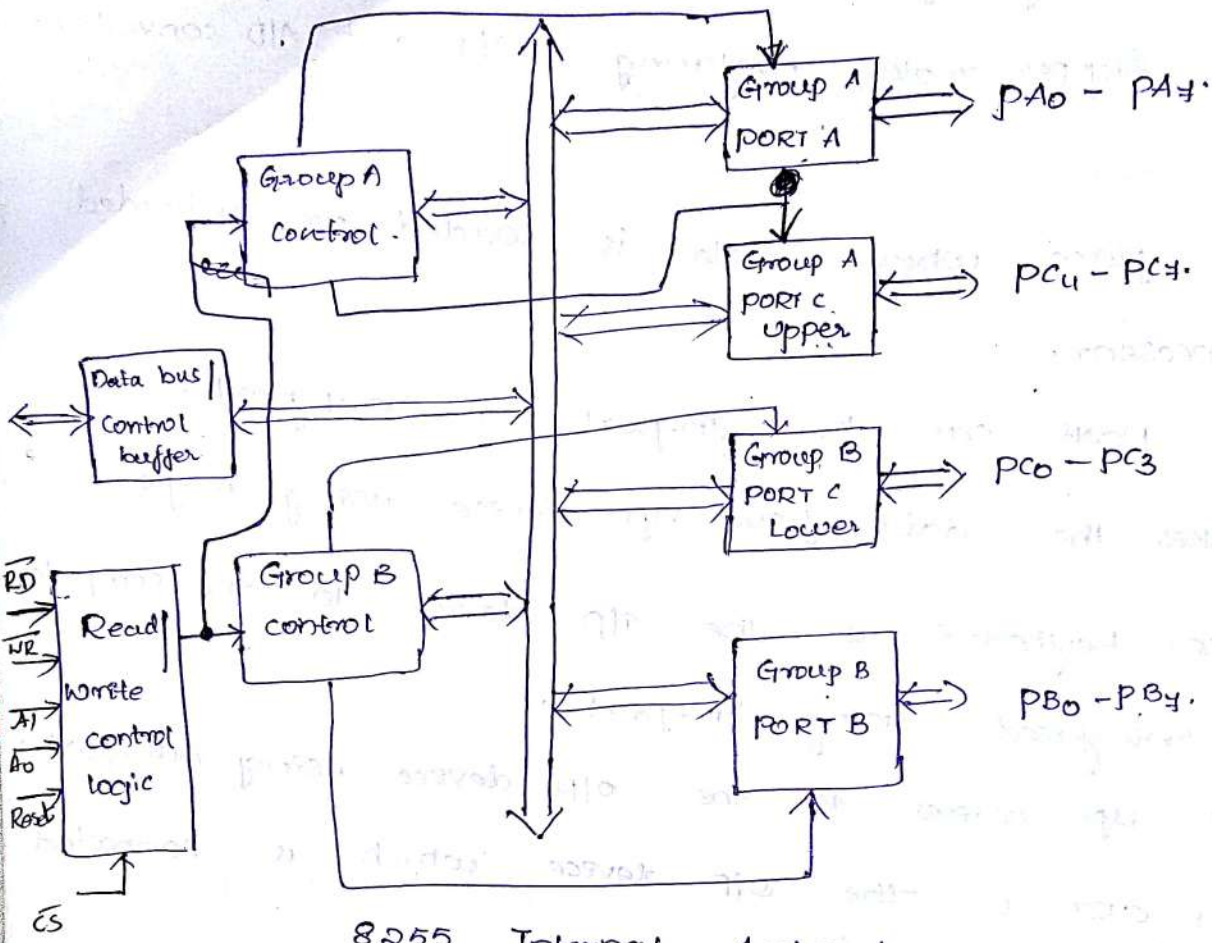
→ out-ports are latched so that the data is available for longer time.

→ ports are classified as (i) Nonprogrammable and (ii) programmable.

Eg :- Intel 8212 / 74LS 245 is an example of In-port, which is not programmable.

→ 8255 PPI (Programmable peripheral Interface) :-

8255

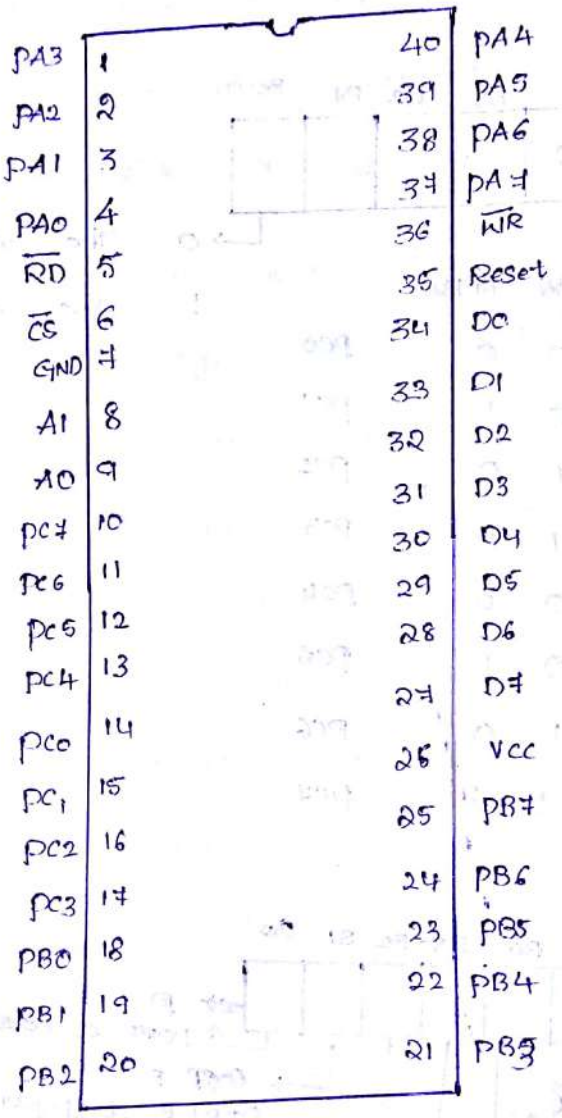


8255 Internal Architecture.

A_1	A_0	
0	0	port A.
0	1	port B
1	0	port C
1	1	CWR (command word register)

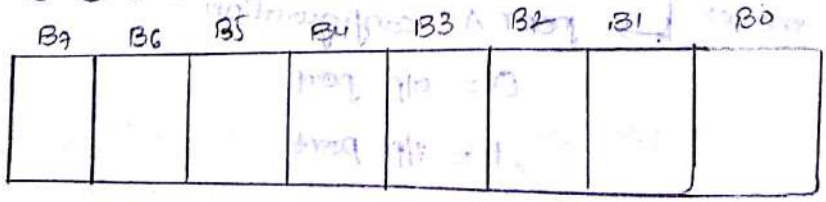
28/1/2013

8255 pin configuration :-



The 8255 is configured by the microprocessor by programming command word register CWR.

CWR (command word Register) :-

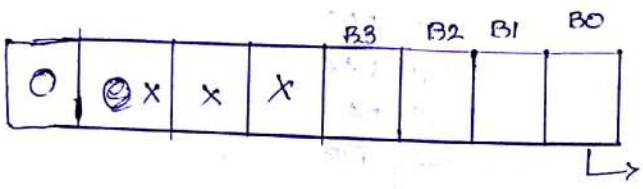


The 8255 can be made to operate in BSR (Bit Set Reset Mode) (or) I/O mode (Input output port Mode).

If B7 is zero then 8255 operation is in BSR Mode.

If B7 is one then 8255 operation is in I/O Mode.

BSR Mode :-

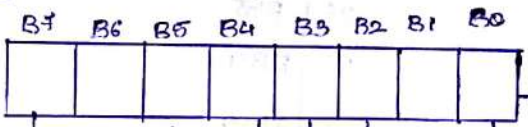


→ 0 The relevant pc bit is zero

1 The relevant pc bit is one

B_3	B2	B1	B0	
0	0	0		pc0
0	0	1		pc1
0	1	0		pc2
0	1	1		pc3
1	0	0		pc4
1	0	1		pc5
1	1	0		pc6
1	1	1		pc7

I/O Mode :-



'1' for I/O Mode

Mode of Group A.

B6 B5

0 0 mode 0

0 1 mode 1

1 X mode 2

PORT C lower pc0-pc3 configuration 0 = o/p 1 = i/p

PORT B configuration 0 = o/p 1 = i/p

Mode of Group B 0 = mode 0 1 = mode 1.

PORTC upper pc4-pc7 configuration 0 = o/p 1 = i/p.

PORT A configuration.

0 = o/p port

1 = i/p port

In Mode 0, the ports ie, port A, port B, port C works as a Simple I/o ports. In Mode 1, port A and port B can be programmed as I/O ports with handshake. The handshake signals are provided by port C. In Mode 2, which is applicable only for port A. The port A acts as a bidirectional.

→ Interface an 8255 with 8086 to work as an I/O port, initialize port A as o/p port, port B as i/p port, port C as o/p port. port A address should be 0740h. Write a program to sense switch positions SW0-SW7 connected to port B.

The sensor pattern should be displayed on port A to which 8 LEDS are connected while port C locates displays no. of ON switches out of the total 8 switches.

A, Do the Design in Mode 0.

A, The control word is formed as given below.

B7	B6	B5	B4	B3	B2	B1	B0
1	0	0	0	0	0	1	0

8 2h.

The control word for above problem is 82h

23/1/2013

8255 PORTS	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0 μp
PORT A	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0

⇒ 0740h

23/1/2013

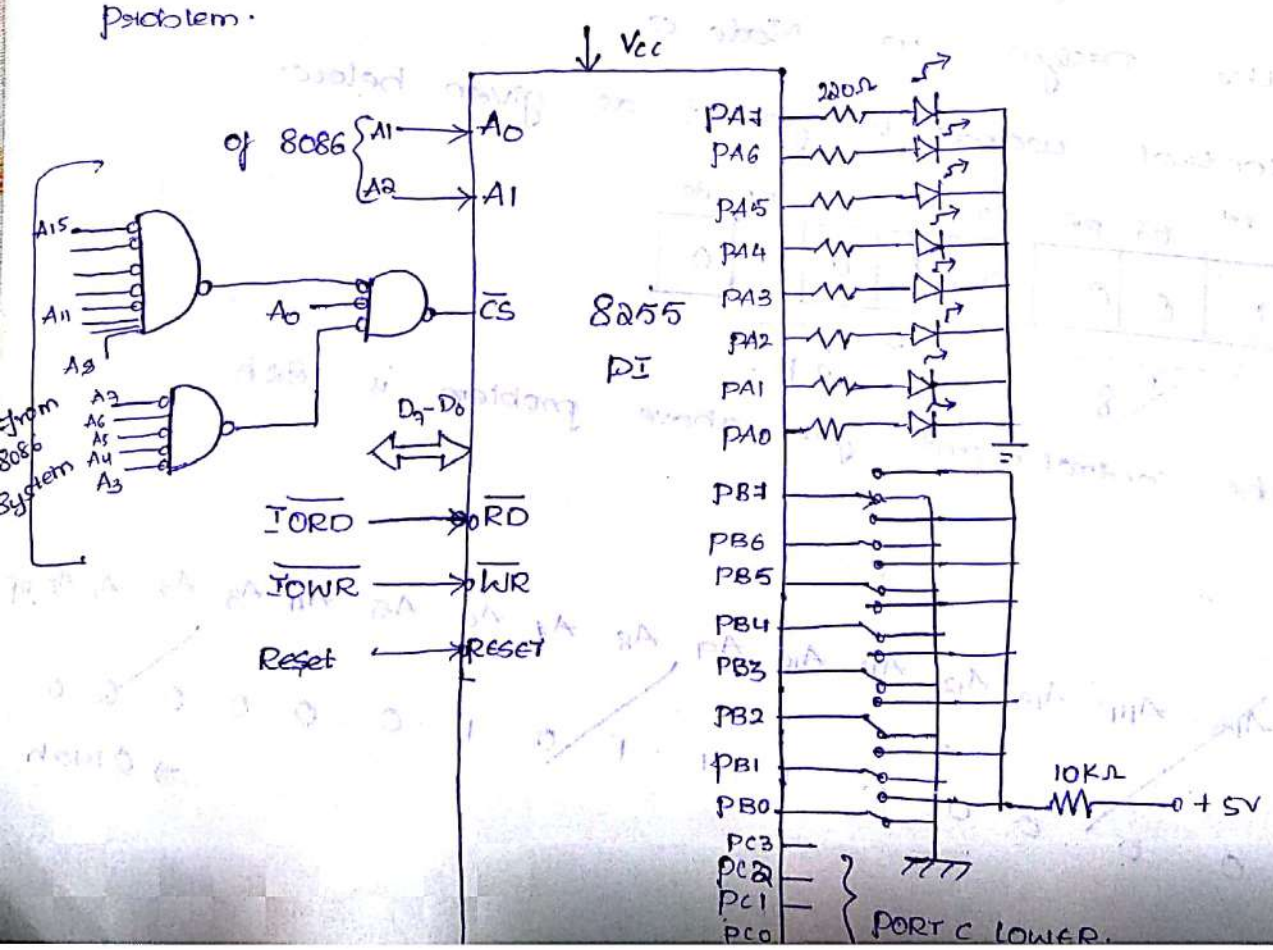
8255 A_1 A_0

0 0 PORT A — 10 — PORT C

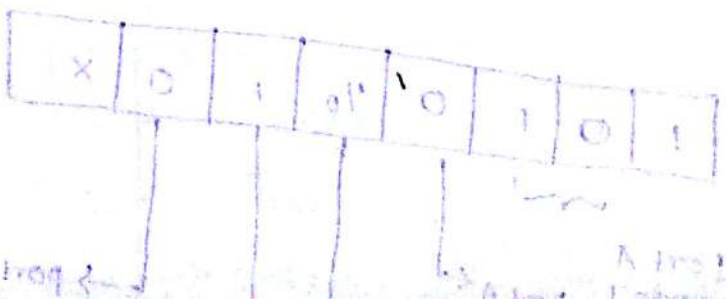
0 1 PORT B — 11 — CWR.

8255 PORTS	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Mp
PORT A	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0740h
PORT B	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	= 0742h.
PORT C	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	0	= 0744h
CWR	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	= 0746h

→ 8255 Interfacing with 8086 and I/O devices for the above problem.

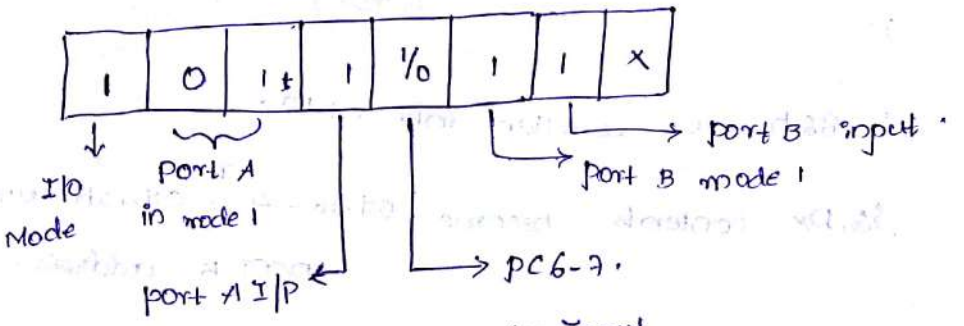


```
MOV DX, 0746h   ; address of CWR 0746h is loaded to DX.
MOV AL, 82h     ;
OUT DX, AL      ; 82h is written into CWR.
SUB DX, 04h     ; So, Dx contents become 0746 - 4 = 0742h which is
                  PORT B address.
IN AL, DX       ; Read PORT B for switch position.
SUB DX, 02h     ; Dx now points to PORTA.
OUT DX, AL      ; Display LED's as per switch position.

MOV BL, 00h
MOV CH, 08h
YY: ROL AL, 01
    JNC XX
    INC BL
XX: DEC CH
    JNZ YY
    MOV AL, BL
    ADD DX, 04
    OUT DX, AL
    HLT.
```



28/1/2013

Control word for Both port A, port B as inputs in mode 1.

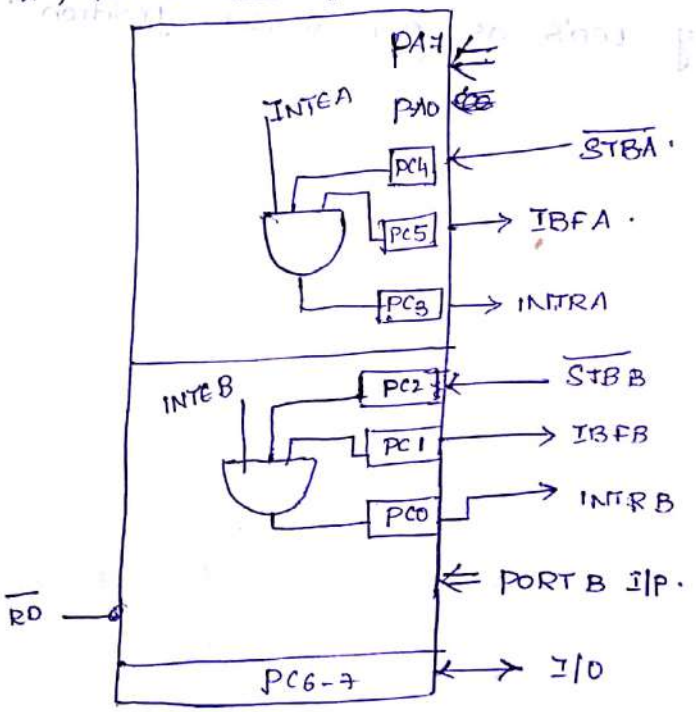


1 = Input

0 = output.

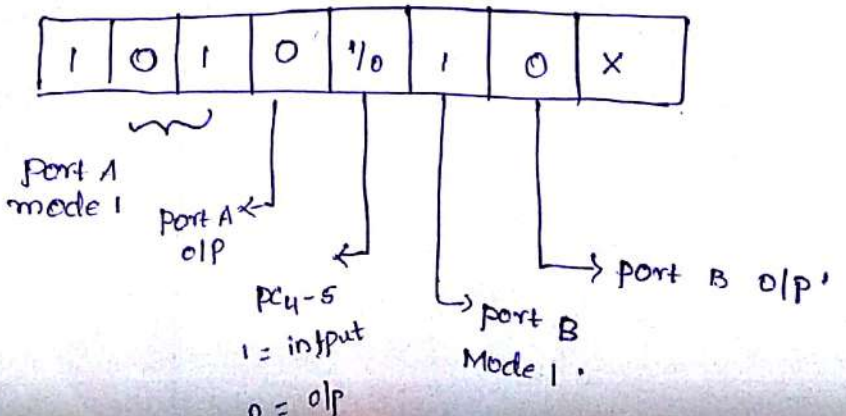
Mode 1 :-

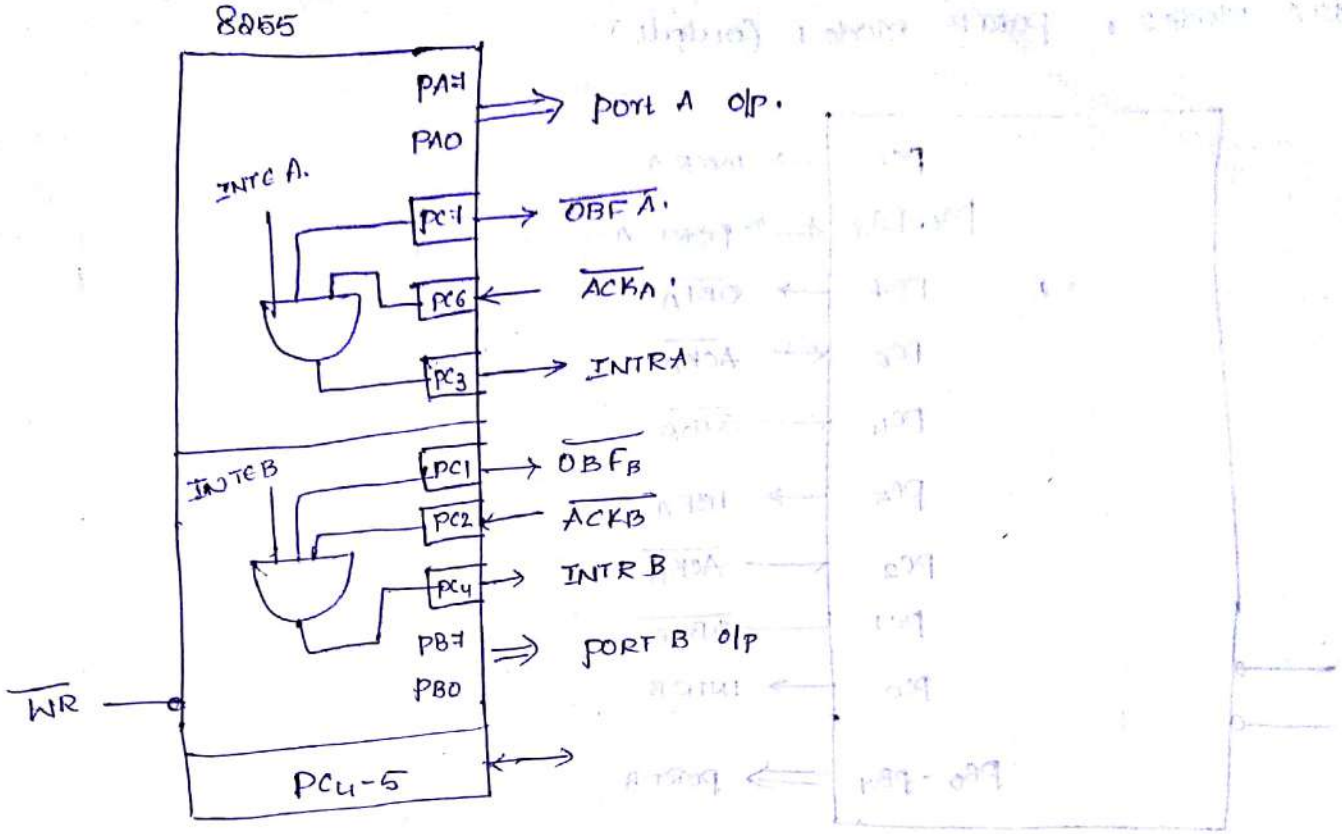
(port A, port B are I/P)



Mode 1 :-

PORT A, PORT B as output ports.





Mode 2 :-

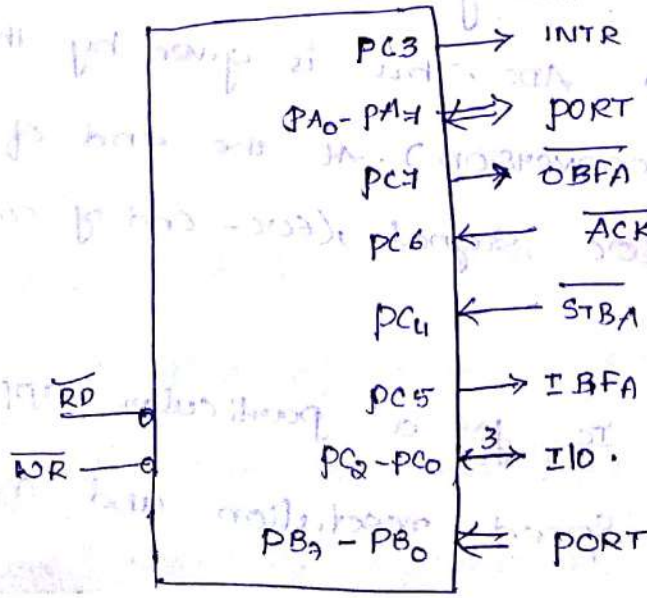
→ This option is available only for PORT A.

→ PORT A functions as Bidirectional port in Mode 2.

→ While PORT A is in mode 2, port B can be in Mode 0 or Mode 1 as i/p port or o/p port.

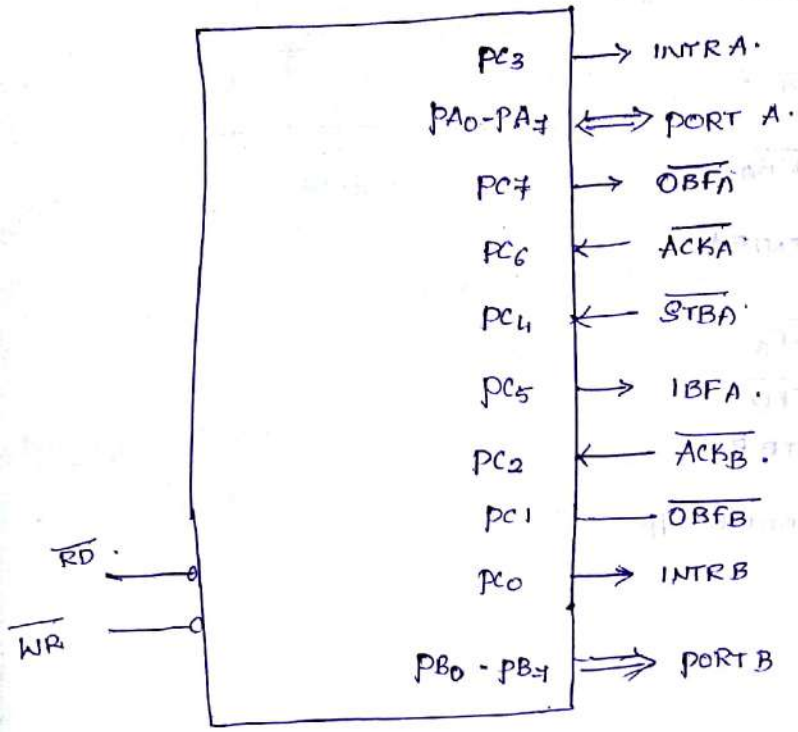
→ Mode 2 option is not available for port B.

PORT A Mode 2, PORT B Mode 0 (Input)



Mode 2:

PORTA Mode 2, PORTB Mode 1 (output)



29/1/2013

Interfacing of A/D converter :-

IC's used for A/D conversion:

a) ADC 0808/0809 8bit successive approximation converter.

b) IC LH109, Make Intersil 12bits dual slope A/D converter.

The general algorithm for ADC interfacing contains following steps

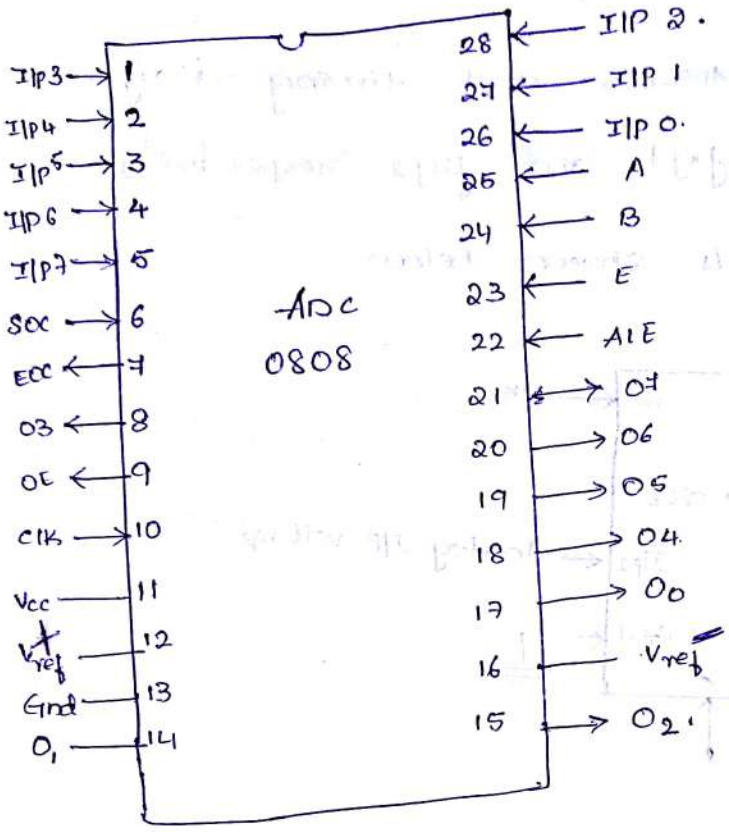
1) ~~Ensure~~ Ensure the stability of Analog input.

2) Issue SOC command to ADC. This is given by the µp. (SOC - Start of conversion). At the end of conversion, the ADC chip gives EOC signal, (EOC - End of conversion) to the µp.

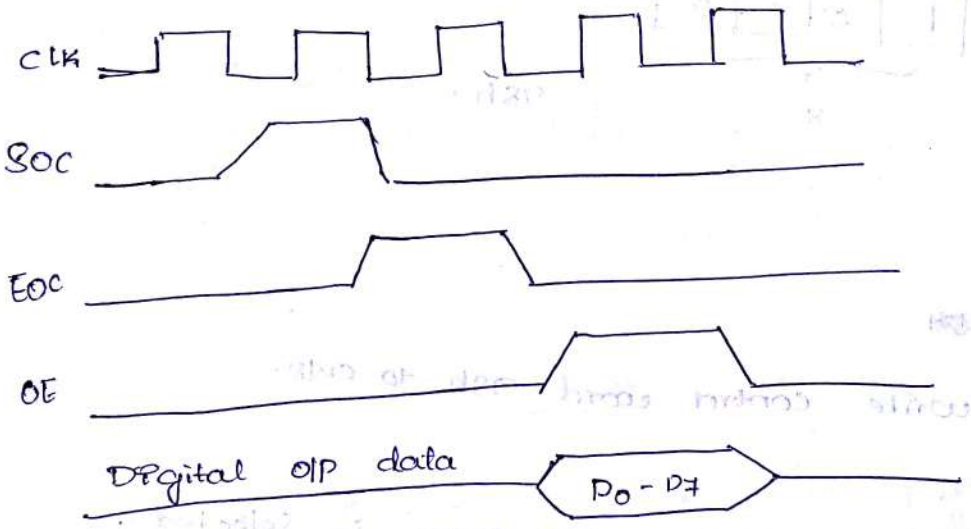
3) The selection of ADC IC for a particular application depends upon the speed, resolution and cost factor

ADC 0808/0809

Fig ①



Timing diagram :- Fig ②



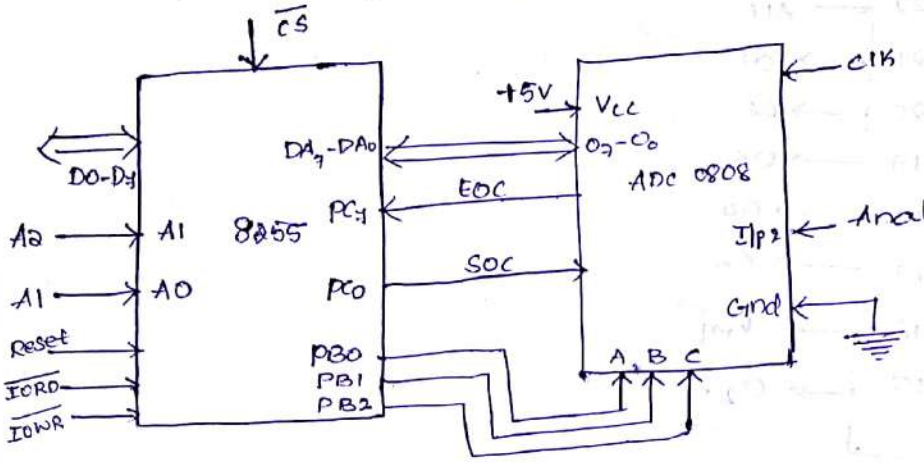
→ Interface 0808 with 8086 using 8255 ports.

use port A of 8255 for transmitting digital data o/p of ADC to cpu and port C for control signals. Assume that analog i/p is present at i/p 2 of ADC and

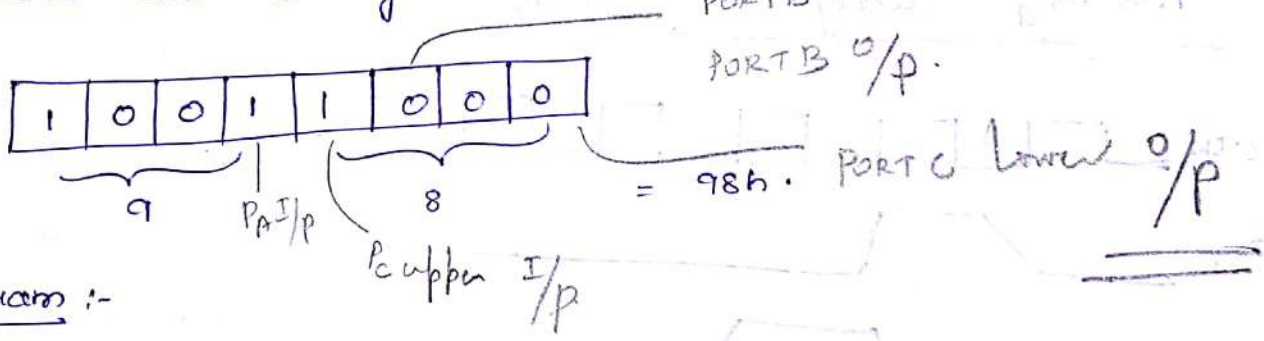
clock i/p of suitable freq. is available for ADC. Draw the schematic and write the required ALP.

Sol :- The pin configuration of ADC0808 and timing diagram of ADC0808 is shown in fig 1 and fig 2 respectively.

The Interfacing diagram is shown below.



The 8255 CWR is given below.



Program :-

```
Mov AL, 98h   ; CWR
OUT CWR, AL ; write control word 98h to CWR.
Mov AL, 02h
Out PORTB, AL ; Analog I/p 2 of ADC0808 is selected.
Mov AL, 00h
OUT PORTC, AL ; ——
Mov AL, 01h ;
OUT PORTC, AL ; _|‾
Mov AL, 00h
OUT PORTC, AL ; _|‾|_ SOC command
```

```
WAIT: IN AL, PORTC ; Read EOC pulse till it is ready.
      RCL ,
      JNC WAIT.
      IN AL, PORT A
      HLT
```

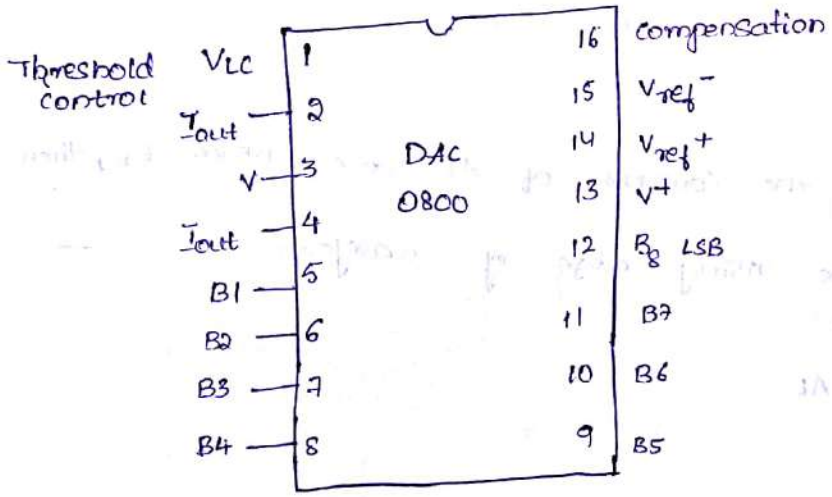
30/1/2013

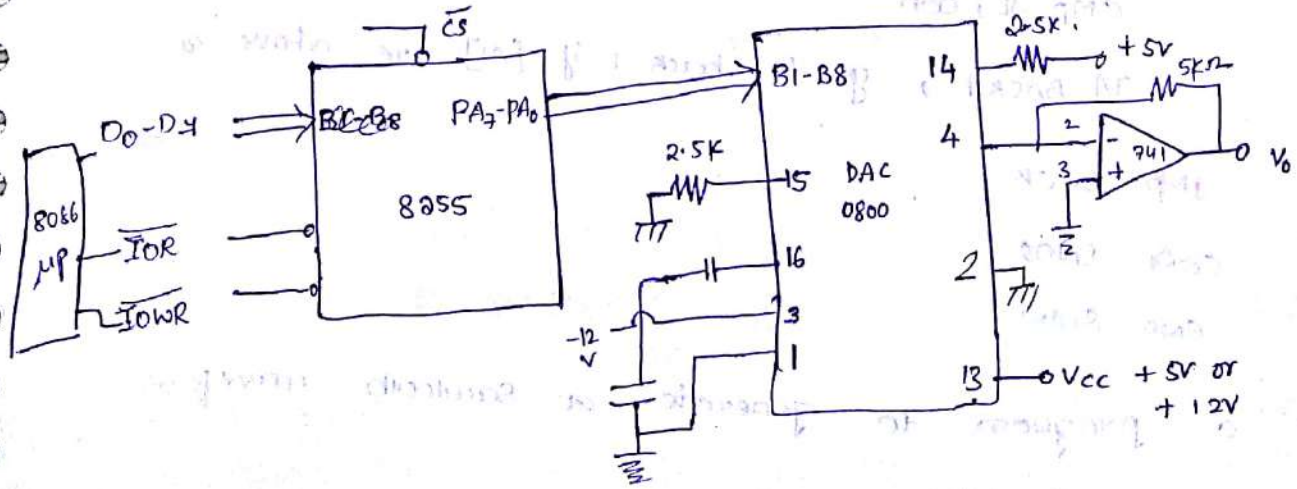
Digital to Analog converter Interfacing.

AD 7523 and

DAC 0800

The pin configuration of 0800 is given below.





The above circuit generates triangular waveform using DAC 0800 and 8255 PPI. op Amp is used to convert Analog Current to proportional analog voltage. port A is used as o/p port. The µP writes digital data i/p. port A which is transferred to DAC0800.

The Assembly language is shown below.

```
ASSUME  CS: CODE11
CODE11  SEGMENT
   START : Mov AL, 80h
           OUT CWR, AL ; control word is given to 8255 PPI
           Mov AL,00h
   BACK :  OUT PORTA, AL.
           INC AL
           CMP AL, FFh
           JB BACK ; If the contents of AL are below FF, then
   when [AL] = FF the rising edge of waveform
   BACK1 : OUT PORT A, AL
           DEC AL
           CMP AL, 00h
           JA BACK1 ; go to back 1 if [AL] are above 0
           JMP BACK
   CODE ENDS
   END START
```

CWR.

1	0	0	0	0	0	0	0

8 0h.

→ Write a program to generate a sawtooth waveform using AD7523 DAC

```
Assume  CS: CODE
CODE    SEGMENT.
START:  MOV AL, 80H     ; Initialise port A as output.
        OUT CINR, AL    ; port
AGAIN:  MOV AL, 00H     ; Start the ramp from 0V
BACK:   OUT PORT A, AL  ; Input 00H to DAC.
        INC AL          ; Increment AL to increase Ramp output
        CMP AL, 0F2H    ; Is upper limit reached ?
        JB BACK         ; If not, then increment the ramp.
        JMP AGAIN       ; Else start again from 00H.
CODE    ENDS
        END START
```

31/1/2013

Stepper motor Interfacing :-

Stator winding

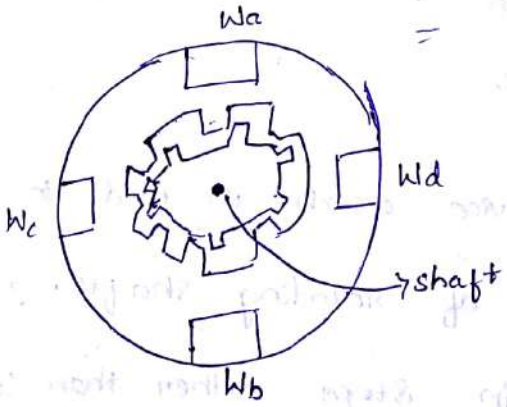
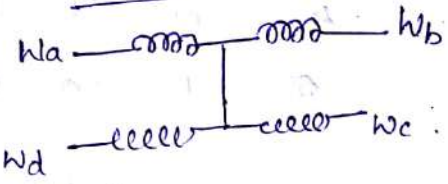
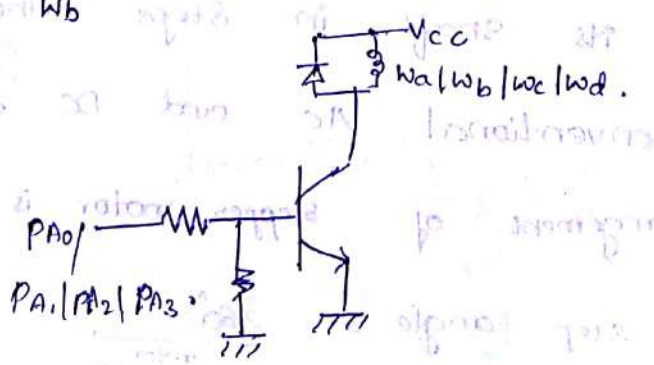
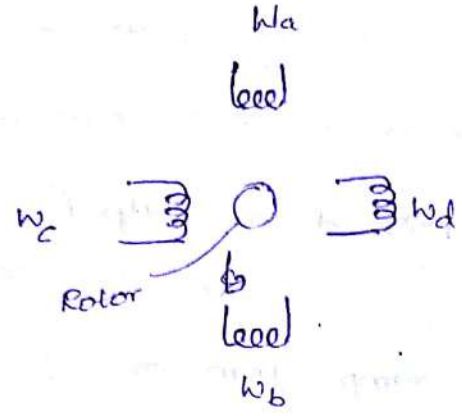


fig ②



When one





Excitation Sequence :- TABLE 1.

Motion	Step	PA_3	PA_2	PA1	PA0
CW clockwise	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
ACW	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

→ The stepper motor is a device which is used to obtain an accurate position control of rotating shafts. It employes protection of its shaft in steps rather than continuous motion in conventional AC and DC motors. The winding arrangement of stepper motor is given in fig 2.

The step angle = $\frac{360^\circ}{\text{No. of rotor teeth}}$

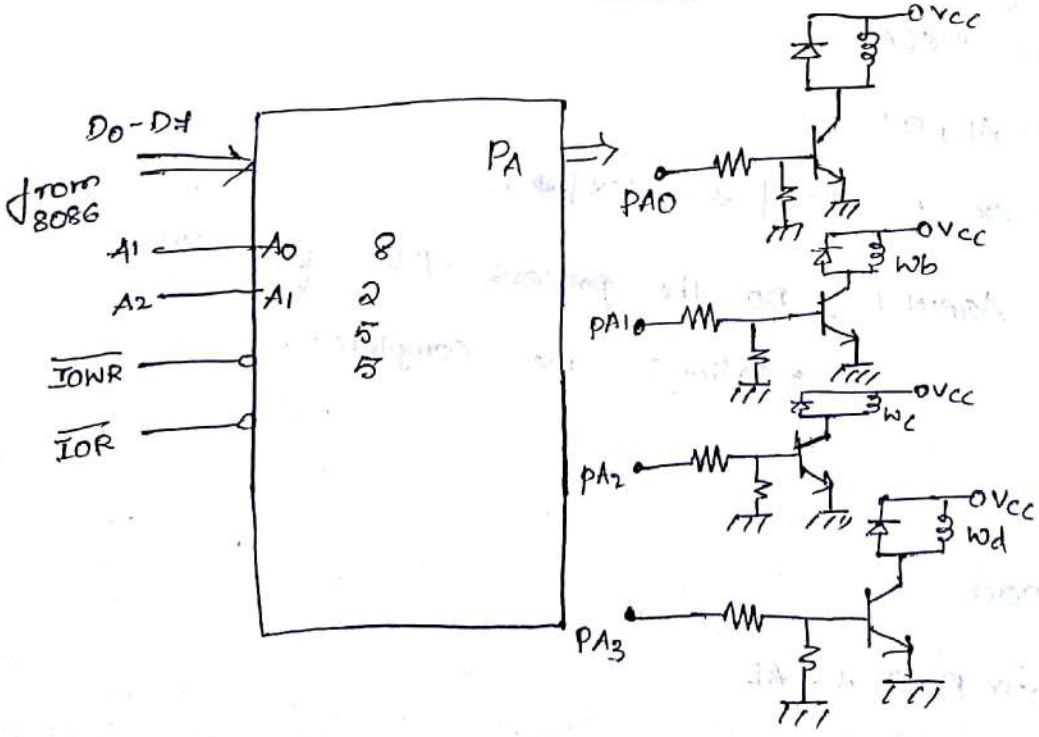
→ To rotate the shaft of stepper motor, a sequence of pulses is to applied to the windings W_a, W_b, W_c, W_d. The no. of pulses required for one revolution = No. of teeth.

↔ The Stator teeth and rotor teeth lock each other to shift in the position of the shaft.

→ The pulse applied to the microprocessor port which makes the transistor ~~teeth~~ ON and one of the winding point is grounded. As Vcc, is applied to the other end of the winding. The winding is energized and moves the rotor one step.

→ A simple scheme for rotation is a wave scheme is shown in table 1.

→ Now, we will design and write a program for a phased stepper motor having 200 teeth for 5 revolutions in clock wise and Anticlockwise.



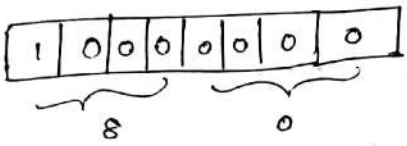
Sol :-

The counter value = No. of revolution × No. of teeth

= 5 × 200

= 1000 d.

Control word



Control word = 80 h.

```
Assume  CS : CODE12.
CODE12  SEGMENT.
START :  MOV AL, 80h
         OUT CWR, AL ; 8255 configured to make port A
                       an output port.
         Mov CX, 1000d ; The counter value is programmed
                         for 5 revolution.
         Mov AL, 88h
AGAIN :  OUT PORTA, AL
         CALL DELAY
         ROR AL, 01
         DEC CX ;  [CX] ← [CX] - 1.
         JNZ AGAIN1 ; Do the process till five CW
                      rotations are completed.
Mov AL, 88h
Mov CX, 1000d
AGAIN2 :  OUT PORT A, AL
          CALL DELAY
```

```
ROL  AL,01
DEC  CX
JNZ  AGAIN2
MOV  AH, 4Ch
INT  21h
CODE12  ENDS
    END  START
```

1/8/2013

1, Design an Interface consisting of 4x4 matrix keyboard with 8255.

1/2/2013

Interfacing with advanced devices :

Memory Interfacing to 8086, Interrupt structure of 8086. vector Interrupt lable, Interrupt Service routine, Introduction to DOS and BIOS Interrupts, Interfacing Interrupt controller 8259, DMA controller 8257 to 8086.

→ Design an Interface between 8086 CPU and 2 chips of 16K x8 EPROM and 8 chips of 32K x8 RAM. Select the starting address of EPROM suitably ~~00000h~~.

A, The last address in the map of 8086 is FFFFFh. After reseting the processor starts from FFFF0h. Hence, this address must lie in the range of EPROM. The address map for the problem is given below.

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	$A_{11} - A_8$	$A_7 - A_4$	$A_3 - A_0$
EPROM ending address FFFFFh	1	1	1	1	1	1	1	1	1111	1111	1111
EPROM starting address F8000h	1	1	1	1	1	0	0	0	0000	0000	0000
RAM ENDING Address 07FFFh	0	0	0	0	0	1	1	1	1111	1111	1111
RAM Starting address 00000h	0	0	0	0	0	0	0	0	0000	0000	0000

RAM Area

F8000h

RAM :-

Two chips of $32K \times 8$

$\Rightarrow 2 \times 32K \times 8$

$\Rightarrow 64K \times 8$

$\Rightarrow 2^6 \times 2^{10} \times 8$

$\Rightarrow 2^{16}$ bytes

$\Rightarrow$ 16 Address bits are required to address the RAM

So, we use $A_0 - A_{15}$ bits.

EPROM

2 chips of $16K \times 8$

$2 \times 16K \times 8$

$32K \times 8$

$2^5 \times 2^{10} \times 8$

2^{15} bytes.

$\Rightarrow$ 15 Address bits are required to address the EPROM

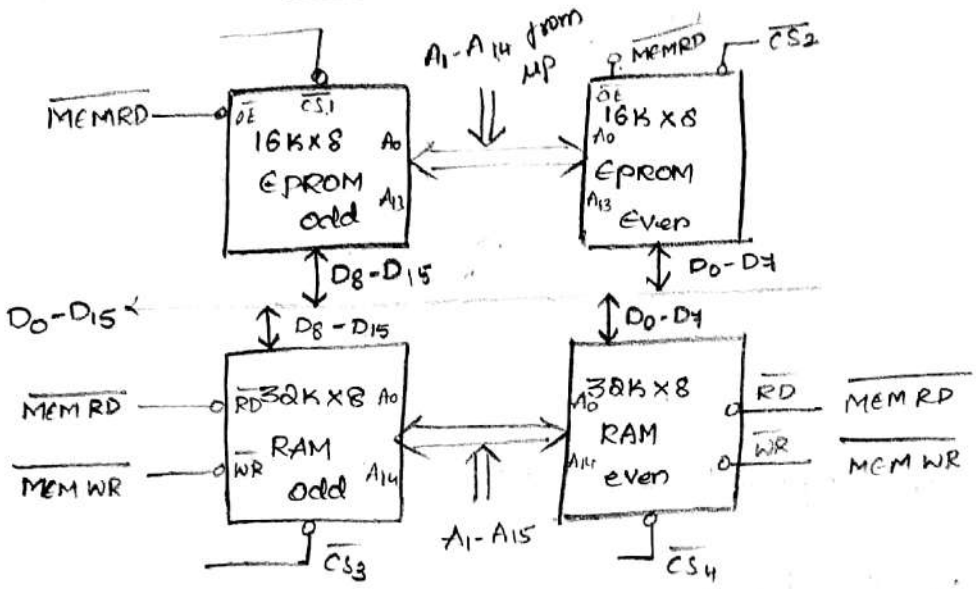
So, we use $A_0 - A_{14}$ bits.

→ The memory in 8086 is organised by odd bank (Higher byte) and even bank (lower byte). The signals A_0 and $\overline{BHE}$ are used to select these lines. As there are 2 EPROM chips of size $16K \times 8$ size, one chip is used for odd bank and another chip is used for even bank. Similarly, RAM is also arranged

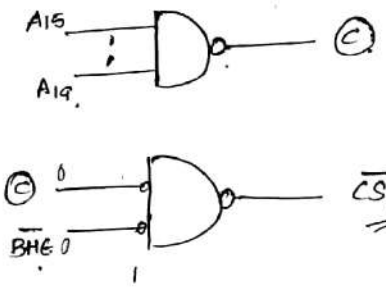
A_0	$\overline{BHE}$	
0	0	Word transfer on $D_0 - D_{15}$. Both even and odd banks are addressed.
0	1	Byte transfer on $D_0 - D_7$ and only even bank is addressed
1	0	Byte transfer on $D_8 - D_{15}$ and only odd bank is addressed

A_0 $\overline{BHE}$
1 1
- No memory operation

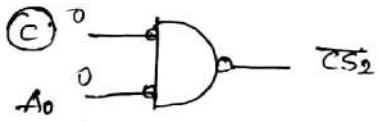
Hard ware design :-



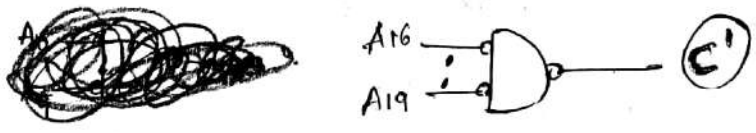
Signals for $\overline{CS1}$:-



Signals for $\overline{CS2}$:-



Signals for $\overline{CS3}$:-



Signal for $\overline{CS4}$:-

(2) Interface 2 4K x 8 EPROM and 2 4K x 8 RAM with 8086. Select suitable address lines.

RAM

$2 \times 4K \times 8$

$8K \times 8$

$2^3 \times 2^{10} \times 8$

2^{13} bytes.

13 Address bits are required to address the RAM.

So, we use $A_0 - A_{12}$ bits.

EPROM

$2 \times 4K \times 8$

$2^3 \times 2^{10} \times 8$

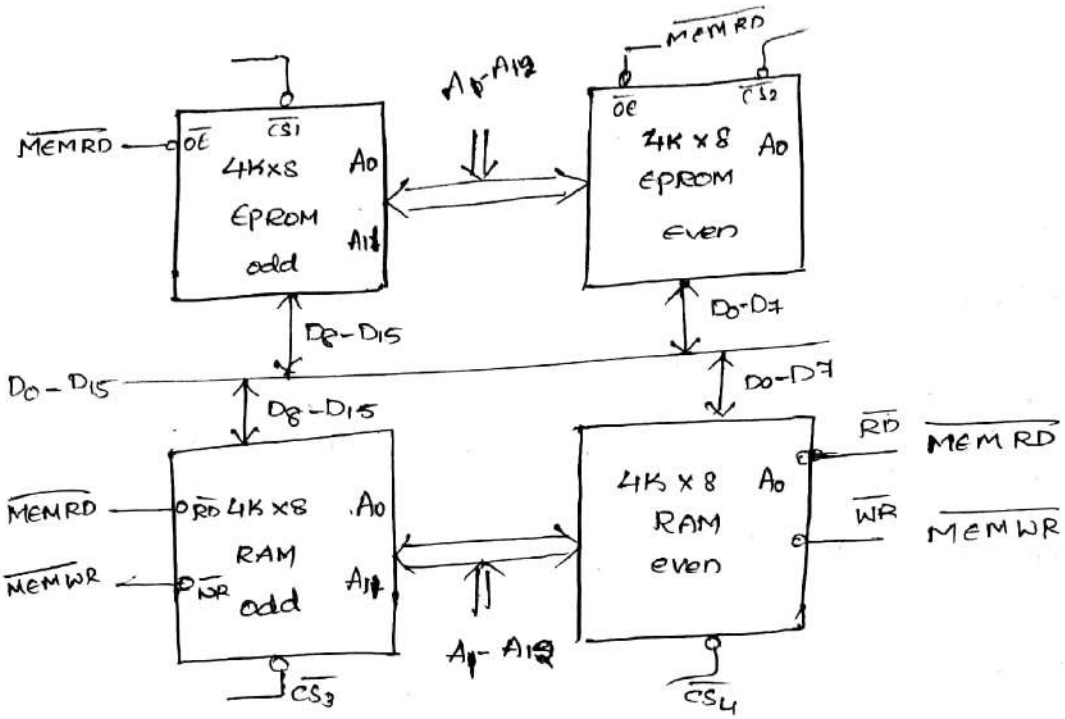
2^{13} bytes.

13 address bits are required to address the EPROM.

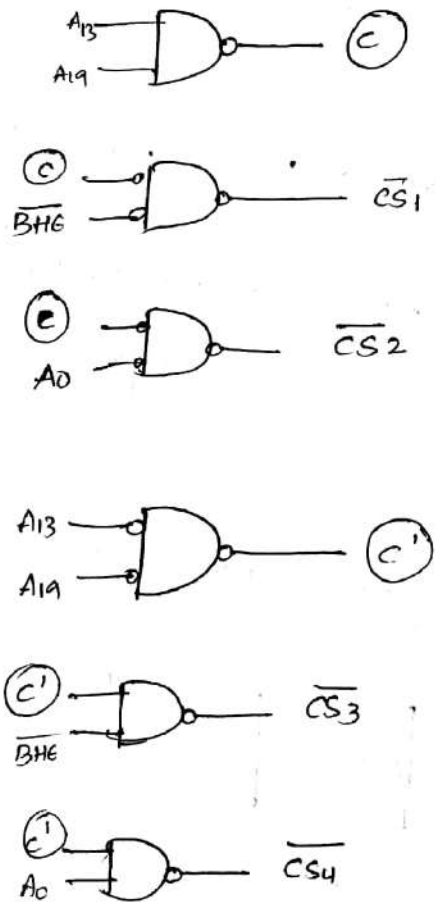
So, we use $A_0 - A_{12}$ bits

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	$A_{11} - A_8$	$A_7 - A_4$	$A_3 - A_0$
EPROM											
Ending address FFFFFh	1	1	1	1	1	1	1	1	1111	1111	1111
Starting address FE000h	1	1	1	1	1	1	1	0	0000	0000	0000
RAM											
Ending address 01FFFh	0	0	0	0	0	0	0	1	1111	1111	1111
Starting address 00000h	0	0	0	0	0	0	0	0	0000	0000	0000

Hardware design :-



Signals



1) Design an Interface consisting of 4×4 matrix keyboard with 8255.

A.

```
Assume CS: CODE
CODE SEGMENT.
PORT A  EQU 0000
PORT C  EQU 0004
CR      EQU 0006
PROC KEY NEAR.
START: MOV AL, 81H ; Initialize port CL as i/p and port CU as o/p
       MOV DX, CR ; [Initialise 8255]
       MOT DX, AL ;
       MOV AL, 00H
       MOV DX, PORTC
       OUT DX, AL ; Make all scan lines zero
BACK:  IN AL, DX
       AND AL, 0FH
       CMP AL, 0FH ; check for key release
       JNZ BACK ; If not, coait for key release.
BACK1: IN AL, DX
       AND AL, 0FH
       CMP AL, 0FH ; check for key release.
       JZ BACK1 ; If not, wait for key press.
       CALL DELAY ; wait for key debounce
       MOV BL, 00H ; Initialize key counter.
```

```
        Mov  CL, 04H
        Mov  BH, FEH ; make one column low.
NEXT COL: MOV  AL, BH
        OUT  DX, AL
        Mov  CH, 04H ; Initialize row counter.
        Mov  DX, PORT A
        IN AL, DX ; Read return line status.
NEXT ROW: RCR AL, 1 ; check for one row.
        JNC display ; If zero, go to display, Otherwise continue
        INC  BL  ; Increment key counter
        DEC  CH ; decrement row counter
        JNZ NEXT ROW ; check for next row.
        Mov AL, BH
        RCL AL, 1 ; Select the next column.
        Mov BH, AL.
        DEC CL ; decrement the next column.
        JNZ NEXT COL ; check for last column if not repeat.
        JMP start ; Go to start.
        RET
KEY     END P
        END START
```

4/2/2013

Interrupt Structure of 8086:

Whenever a no. of devices request service of the microprocessor, the µp has to handle such situations. for example, the µcomputer should be able to give response to devices like keyboard, sensors, other components when they request for service. there are two approaches to handle such situations.

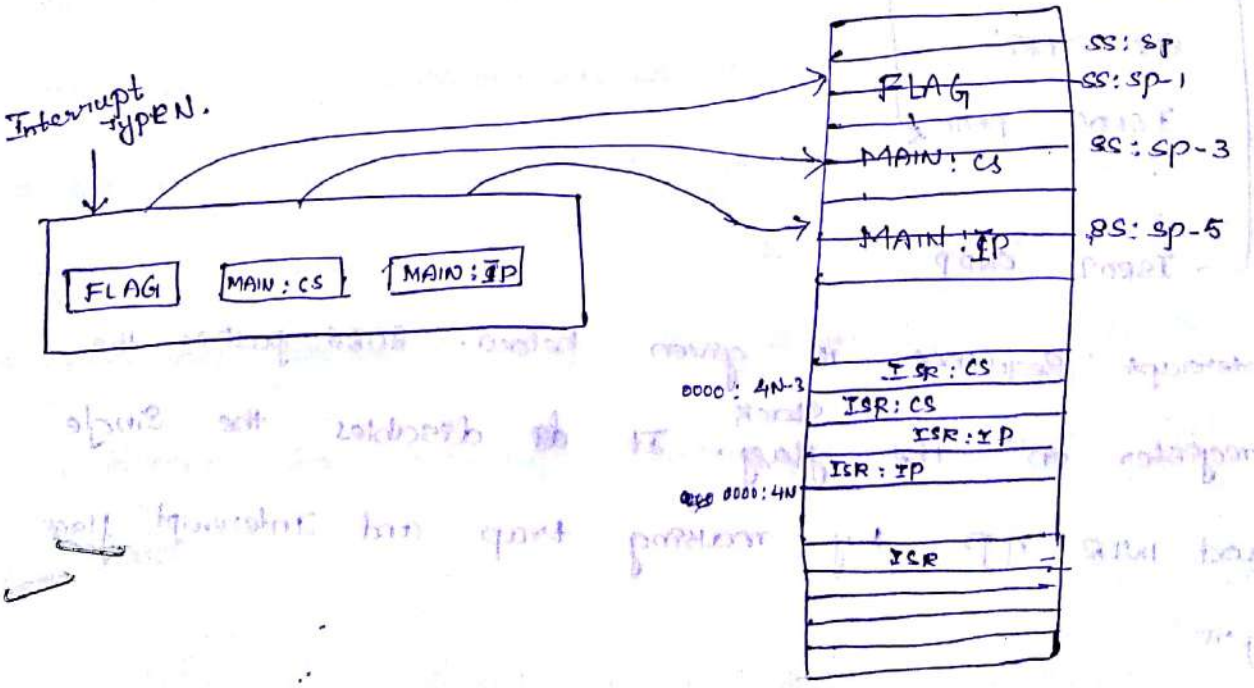
① pooled approach ② Interrupt approach.

In pooled approach, the µp periodically scan all the I/O devices for any service request and then provide service accordingly. This reduces the system speed and hence inefficien

→In the Interrupt method; the I/O deveces send signals on INTR or NMI pins. The µp receives those signals and provides the service.

In the case of 8085, there are 5 pins.

TRAP, INTR, RST 7.5, RST 6.5, RST 5.5 for external interrupts.



→ Whenever a µp is interrupted it stops the main program execution. Main program flag, CS, Ip are pushed to the Stacks.

→ The CS and Ip are loaded with Interrupt Service Routine address as shown above. When RET instruction is encountered in ISR, CS and Ip are loaded with main program data by pop operation. So, the main program is resumed appropriately.

6/2/2013

Transfer of control During ISR:-

```
ASSUME  CS: CODE.
CODE 8: SEGMENT.
START:  MOV  AL, 80H
        - - - - -
        - - - - - -
        INT 09          ; Interrupt has occurred.
        MOV BH, 90H
        - - - - - -
        - - - - - - -
CODE 8  ENDS
END START
ISR09   PROC
        - - - - - -
        - - - - - -
ISR09   ENDP
```

The Interrupt Sequence is given below. 8086 pushes the flag register on the stack. It disables the single step and INTR i/p by making trap and Interrupt flags to zero

It says a / used / put

↳ It saves the main program values (CS & IP) by pushing them into the stacks.

→ It does an indirect far jump to the start of ISR (Interrupt Service Routine) by loading new values of CS and IP of ISR

→ for ex. if Interrupt type is 4, the memory address is 4×4 = 16_{10} = 10h.

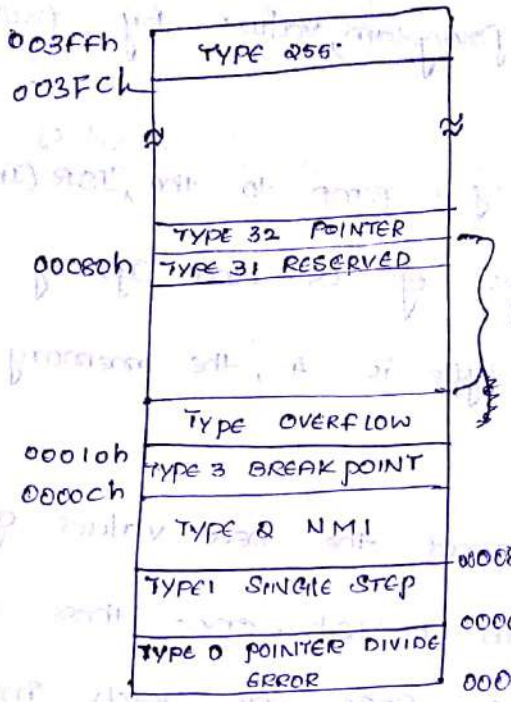
→ In 8086, we will read the new values of IP from 00010h and therefore CS from 00012h. once these values are loaded in CS and IP registers, 8086 will fetch instructions for the new address to execute ISR.

→ When IRET / ENDP instruction is encountered, the main program values are popped from the stacks and the main program is resumed.

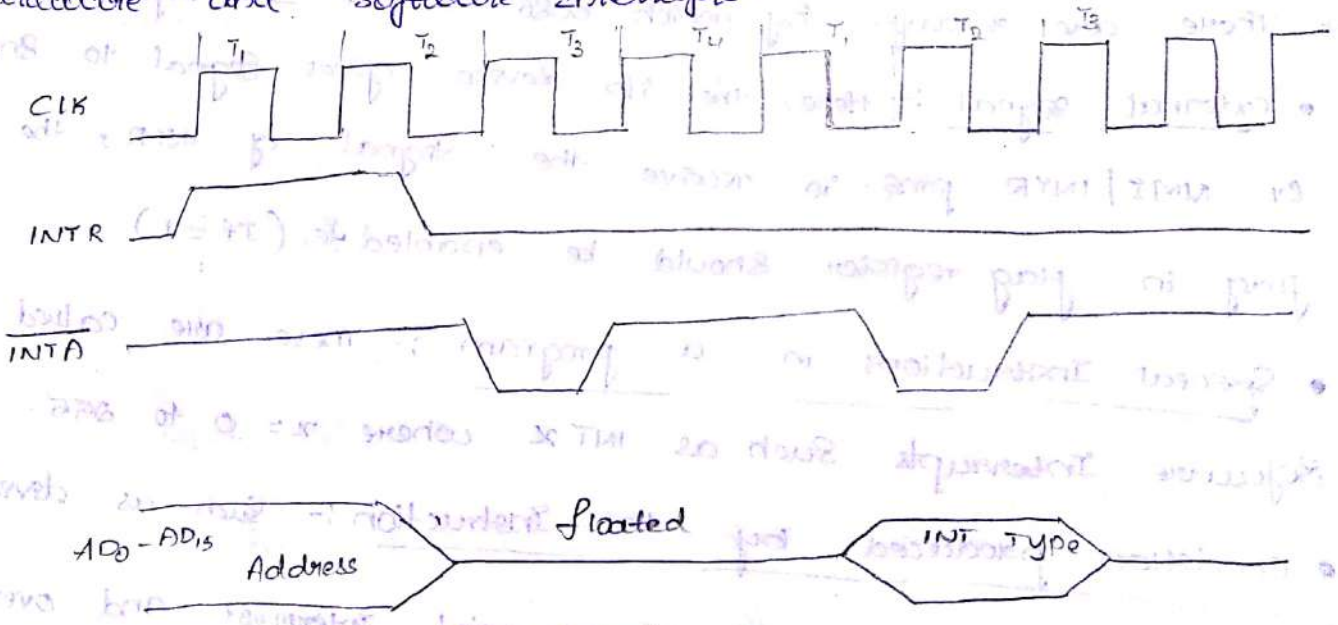
→ There are 3 ways by which 8086 can be Interrupted:

- External signal :- Here, the i/p device gives signal to 8086 on NMI / INTR pins. To receive the signal of INTR, the IF flag in flag register should be enabled ie, (IF = 1)
- Special Instructions in a program :- These are called Software Interrupts such as INT x where x = 0 to 255.
- Condition produced by the Instruction :- Such as divide by 0, Single Step Interrupt, Break point Interrupt and overflow interrupt.

8086 Interrupt Vector table :-



In 8086, the first 5 types have explicit definitions such as divide by 0, overflow etc. the next 27 interrupt type ie, type 5 to 31 are reserved for future use. the upper 224 interrupt types from type 32 to 255 are available for the user for hardware and software interrupts.



Interrupt Acknowledge Machine cycle

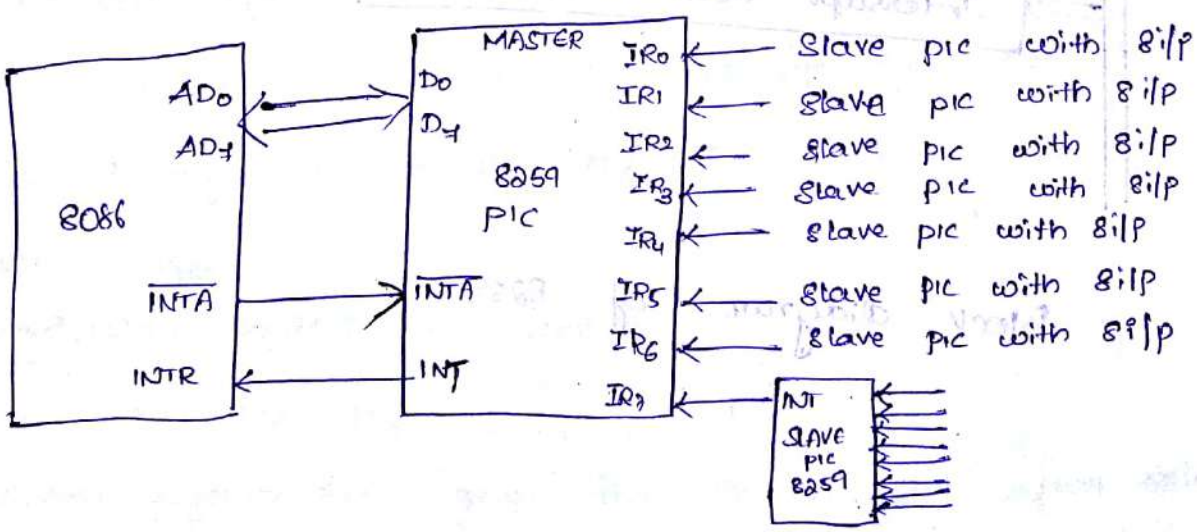
When i/p wires requires the service of the µp it sends a signal an INTR time to µp in response to this the µp gives a INTA loco (Interrupt Acknowledge) pulses as shown above. During first INTA the bus is tristated, in the second INTA pulse, the i/p device puts INTA type. The INT type information is used by µp to get new values of ISR.

The features of 8259 :-

→ It has 8 Interrupt lines so it can manage upto 8 interrupts This is equivalent to providing 8-interrupt pins on the processor instead of single INTR pin.

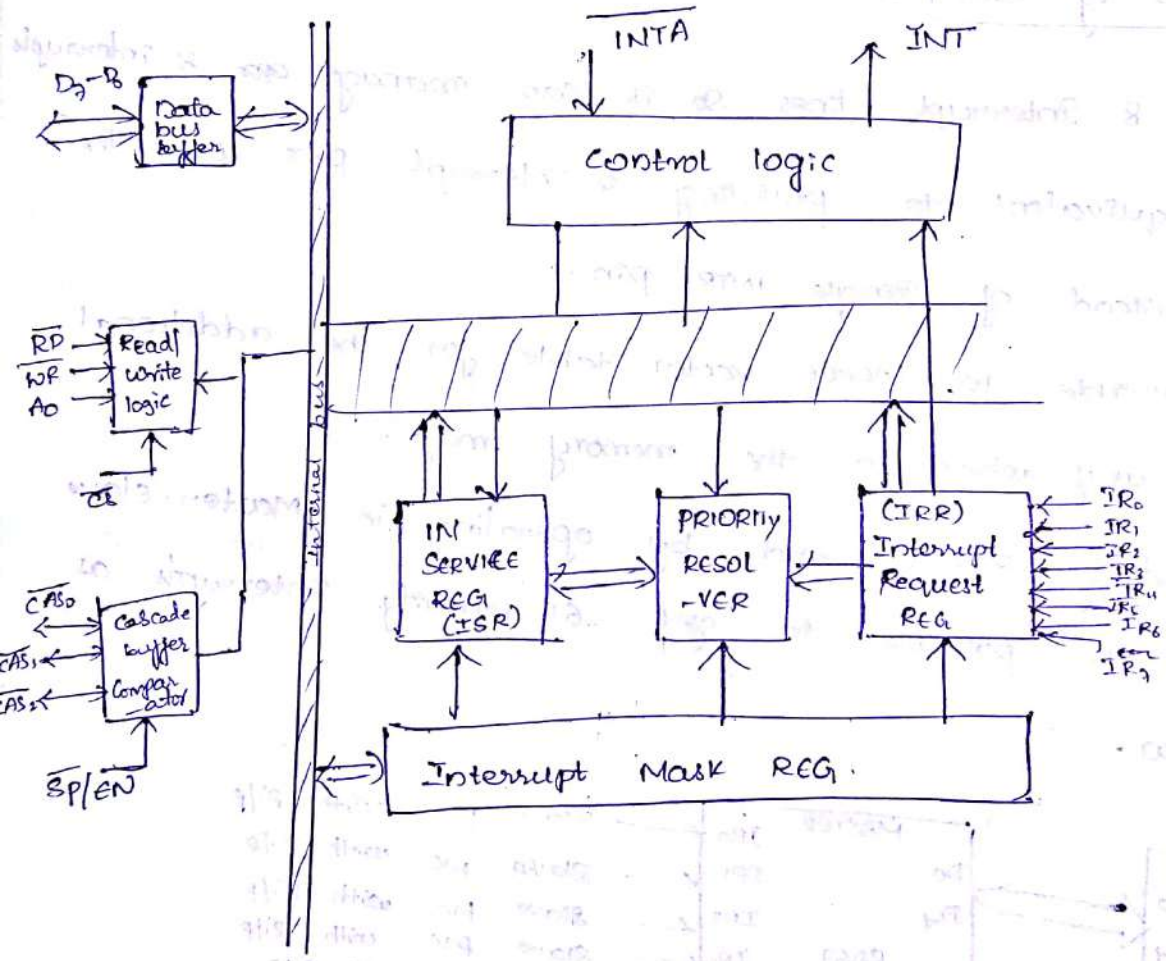
→ It is possible to locate vector table for the additional interrupts any where in the memory map.

→ By cascading 8259's and by operating in Master-slave mode, it is possible to get 64 priority interrupts as shown below.



↳ 8259 PIC has internal mask register by which the individual lines $IR_0 - IR_7$ can be enabled or disabled. 8259 PIC can be programmed to accept either level trigger or edge trigger inputs.

→ With the help of 8259, we can get information of pending interrupts, inservice Interrupts, masked Interrupts.



Block diagram of 8259

8/8/2013 8259 PIC.

(a) Data bus Buffer

(b) Read/write logic

(c) control logic

(d) IRR, ISR, IMR

(e) Priority Interrupt Resolver.

(f) Cascade buffer configuration

(g) $\overline{SP}/EN$ +Vcc MASTER

GND SLAVE.

Data bus buffer :- The cpu sends command word to PIC 8259 and obtains its status the interrupt type (8 bit data) is transferred to up by PIC using the data bus transfer. This center

Read/write logic :- This controls the direction of data flow on data bus buffer.

control Logic :- This gives the signal INT after selection of any i/p line (IR_0 to IR_{07}) by priority Interrupt Resolver. Also it receives $\overline{INTA}$ pulse from cpu.

IRR, ISR, IMR :- The IRR is used to store all the interrupt status of 8 input devices. It is 8 bit register. The Interrupt Mask register IMR enables or disables a particular I/p device. Bit 1 is used for disabling / Masking and Bit 0 is used for enabling. The Interrupt Service Register stores all the interrupts that are being processed.

Priority Interrupt Resolver :- It gets data from IRR, ISR, IMR and selects a particular input device for Interrupt processing The master uses this block for communication with slave.

f) $\overline{CAS_0}$, $\overline{CAS_1}$, $\overline{CAS_2}$ are the slave Identification bits. TO identify any one of the slaves by the master.

g) $\overline{SP}$ / EN :- For the master slave pic 8259, the pin $\overline{SP}$/EN is connected to V_{cc}. for the same pic, this pin is connected to ground.

Introduction to DOS and BIOS Interrupts :-

```
Mov AH, 4Ch    ; function code
INT 21h        ; } this is DOS Interrupt.
```

DOS Interrupts :-

Interrupts	function code	operation
INT 21h	4Ch	Terminate the programme with appearance of command prompt on the screen
INT 21h	01h	Read a character from standard input device.
INT 21h	05h	write a character to standard o/p device
INT 21h	3Dh	open file
INT 21h	3Eh	close a file

INT 21h	41h	delete a file.

BIOS Interrupts :-

INT 10h	00h	Set Video mode.
INT 10h	01h	Set cursor shape.
INT 10h	02h	Set cursor position.
INT 10h	03h	Read cursor position.
INT 10h	04h	Read light pen position.

In IBM pc, part of the operating system was located in permanent memory location (EPROM) which is referred to as BIOS (Basic I/P O/P System). This is located at the top of memory in the address range FE000h to FFFFFh or 8086 based processor. The BIOS programs provide direct and low level interactions with various devices in the system. The BIOS has programs for

(a) power on self test.

(b) Time of the day.

(c) Print screen

(d) IO Support program for asynchronous communication, keyboard, printer and display.

The DOS is stored in Harddisk. The services provided by DOS include appearance of command prompt on the screen after switch ON, file management (Create, Read, write, delete files), memory management, directory management and utility programs.

11/2/2013

DMA 8257 to 8086

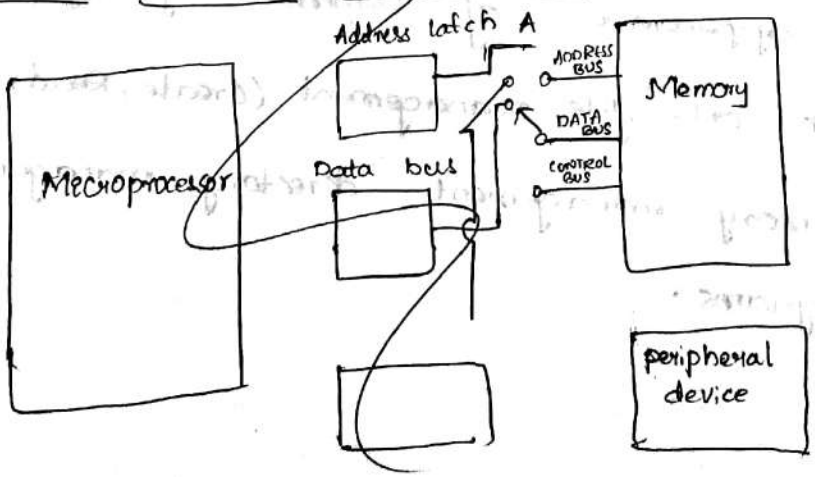
Software controlled Data transfer.

```
    MOV CX, COUNT
    MOV DX, PORT addr
BACK: MOV AL, [SI]
      OUT DX, AL
      INC DX
      INC SI
      loop BACK ;
      RET
```

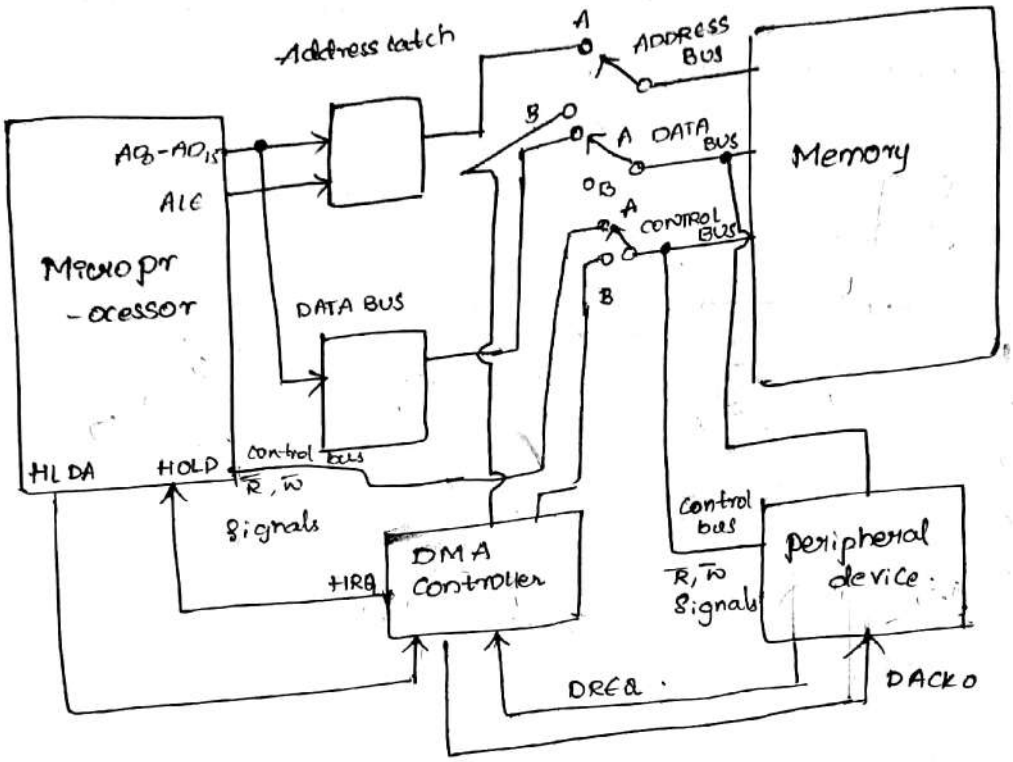
In above program, the data is transferred from memory location to I/O device. First the contents of memory location are brought inside µp ie to AL register and then to the I/O device pointed by DX.

The no. of bytes to be transferred or initialised in CX register. The above scheme is slow and suitable only for small no. of given data. for large volumes of data from disk to memory or from memory to disk, we use DMA method.

Hardware controlled Data transfer :-



Hardware controlled Data transfer :-



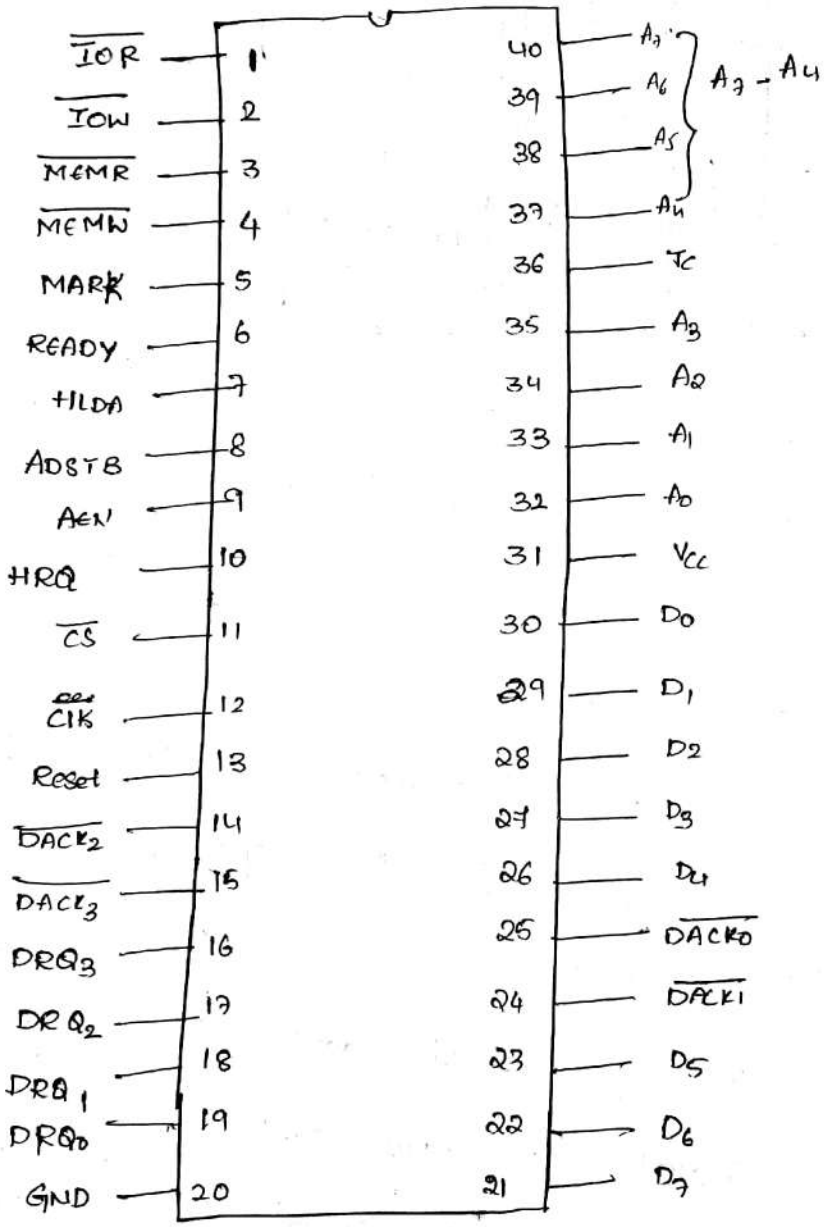
The DMA controller has two types of operations i) Slave ii) master mode

In slave mode, the cpu is in control of system bus and all switch positions are in A. This slave mode is used to configure the DMA controller and to read the status of DMA controller. During Actual progress of DMA operation, the DMA controller is in master mode.

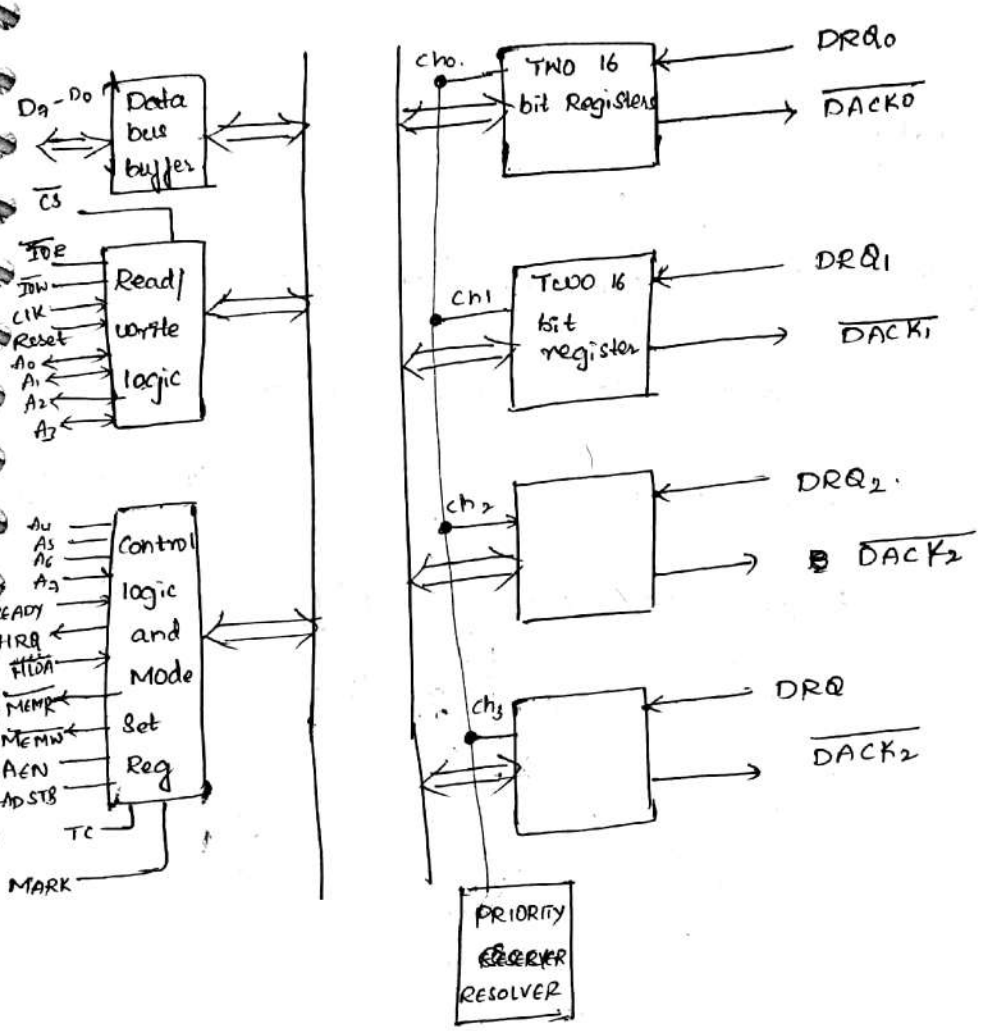
When the peripheral device is ready for DMA, it sends DREQ signal to DMA controller. Then, the DMA controller sends HOLD Request HRQ signal to up. The when the up gives HLDA signal, the switch positions are changed to B and up is completely isolated from System bus. The control signals $\overline{IOR}$, $\overline{MEMW}$ are used to copy the data from disk to memory and the signals $\overline{IOW}$, $\overline{MEMR}$ are used to transfer the

data from memory to disk. At the end of DMA, the HRQ signal is deactivated by the DMA controller and the switch positions are changed from B to A.

Pin configuration of 8257 :



Functional Block diagram of 8257 :-



The Salient features of 8257 :-

(1) This is a four channel programmable DMA controller.

(2) each channel has two 16 bit register. The DMA Address register stores the starting address of memory location for 14 bits of pc register (16 bit terminal count register) are used to store count value
Maximum 2^{14} bytes can be transferred ie, 16Kb.

(3) This has the priority resolver block which can be programmed to work in 2 modes (i) fixed priority & Rotating priority.

d) it has inhibit logic to deselect any one of the 4 channel

→ This is DTL compatible and works with +5V.

18/2/2013

Communication Interface : Serial communication standards.

Serial data transfer schemes:

8251 USART architecture and Interfacing

RS232, IEEE-488, Prototyping and

Trouble shooting.

There are 3 types of communications.

(i) Simplex : This is unidirectional.

Ex:- computer to printer, FM radio receiver.

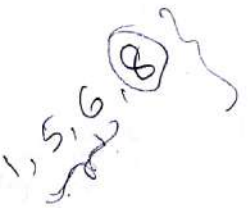
(ii) Half Duplex :- Here, the communication is both direction but simultaneous transmission and reception is not possible.

Ex:- Walkie-Talkie

(iii) Full Duplex :- Here the communication is bidirectional as well as simultaneous.

Ex:- cell phone.

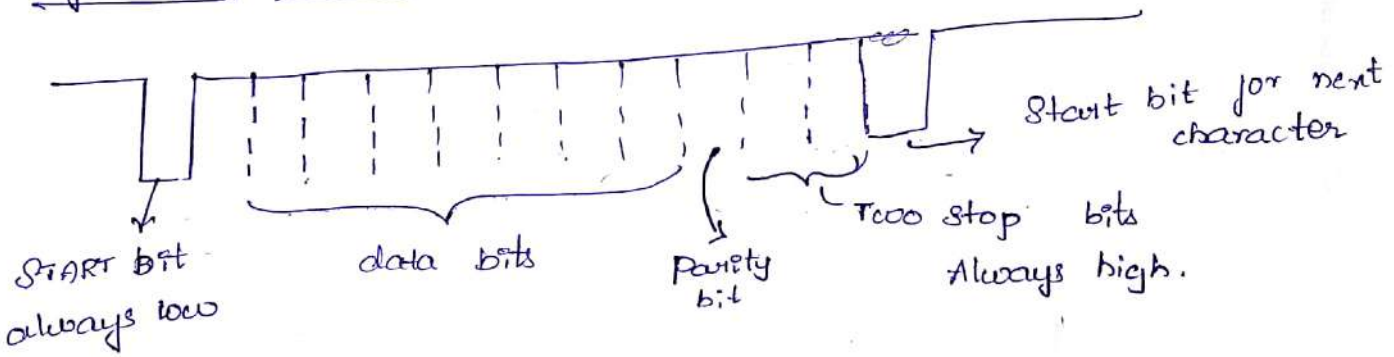
When distance involved is small, we can go for parallel communication which is faster. For long distances, Serial communication is preferred as we can save material cost for wires but the disadvantage is slow.

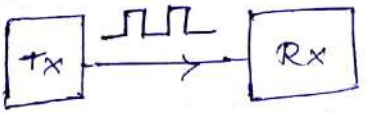


Serial communication

- Asynchronous Serial communication
- Synchronous Serial communication

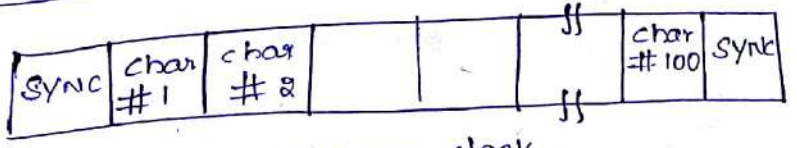
Asynchronous Mode

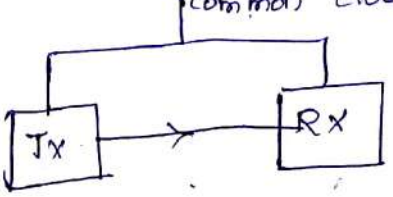




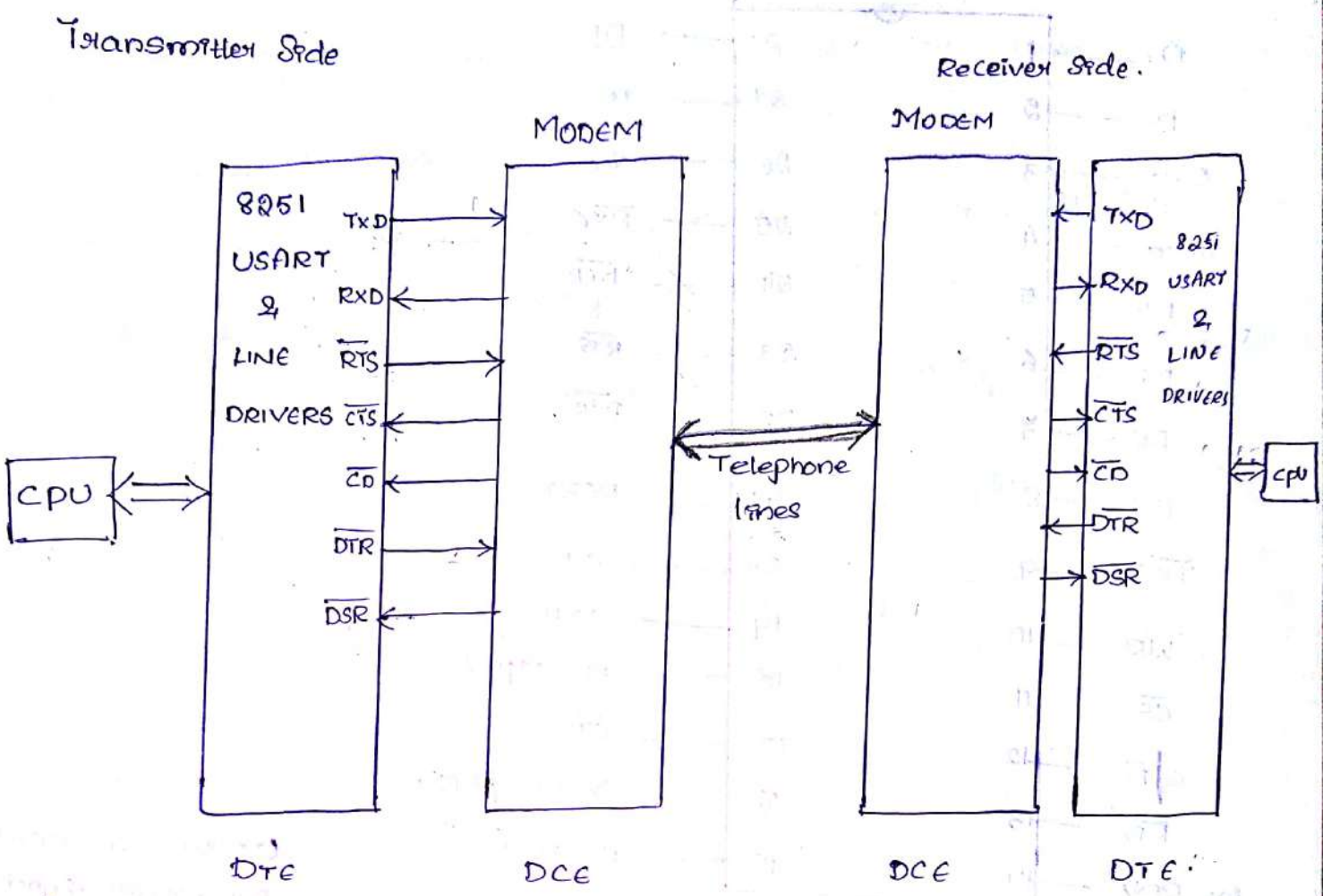
Serial data with LSB first.

Synchronous Mode :-

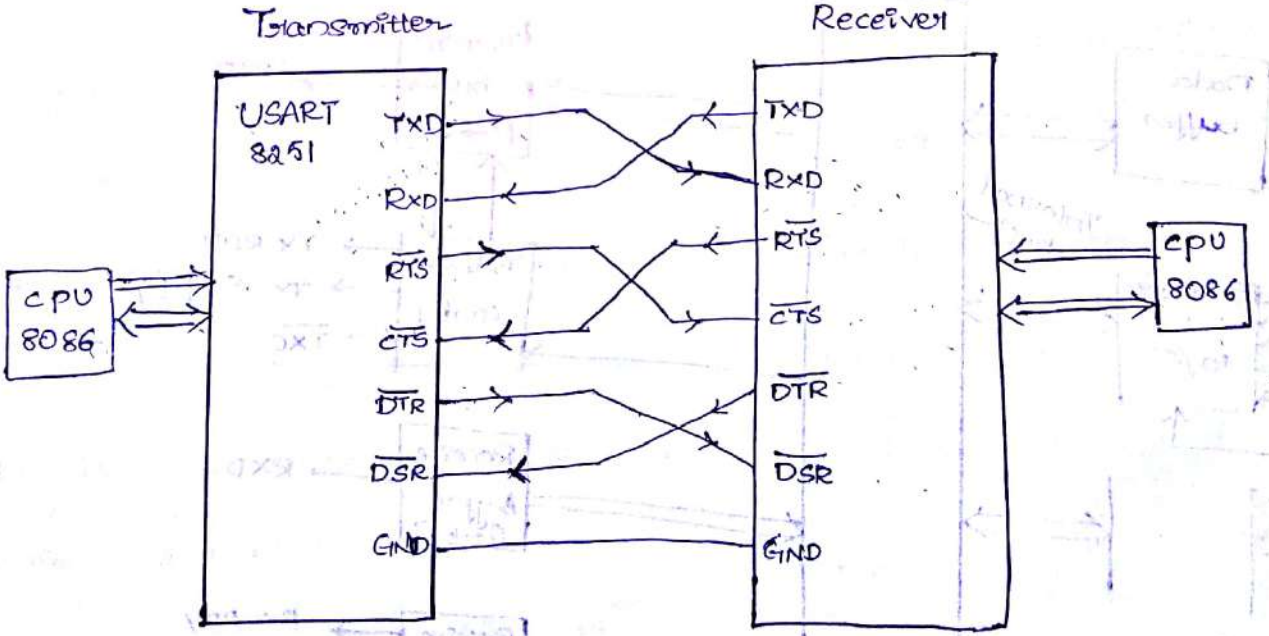




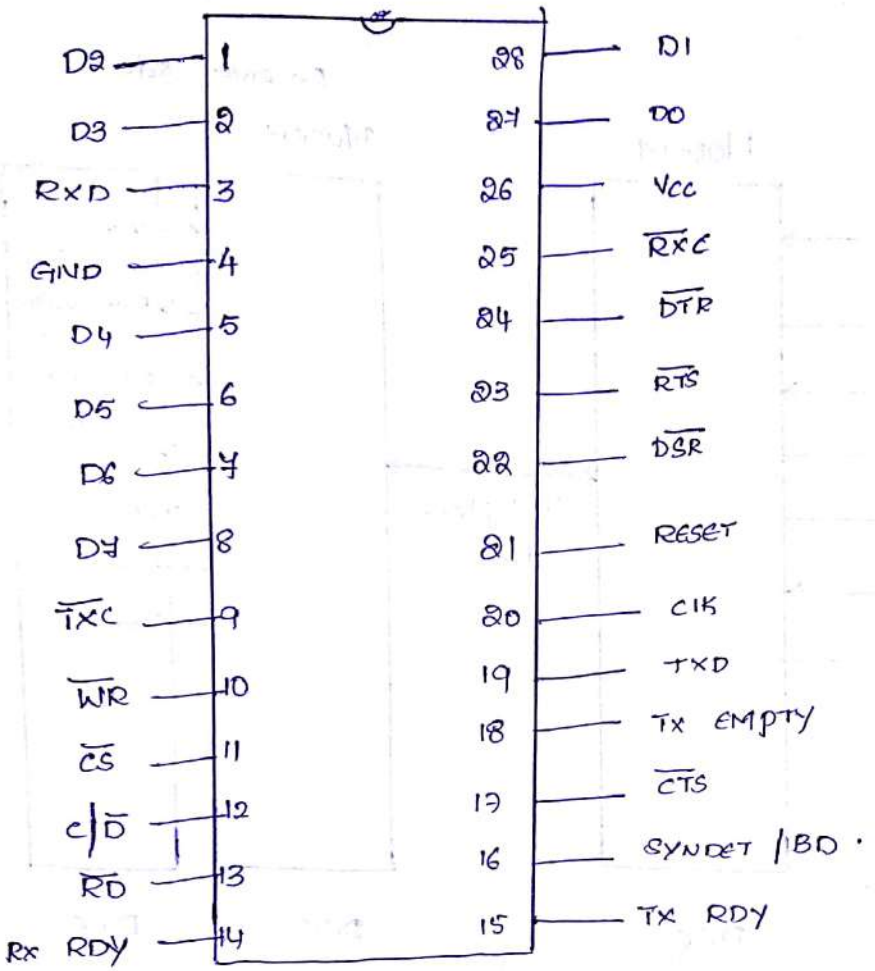
21/2/2013



→ Non Modem connection for Serial communication :-

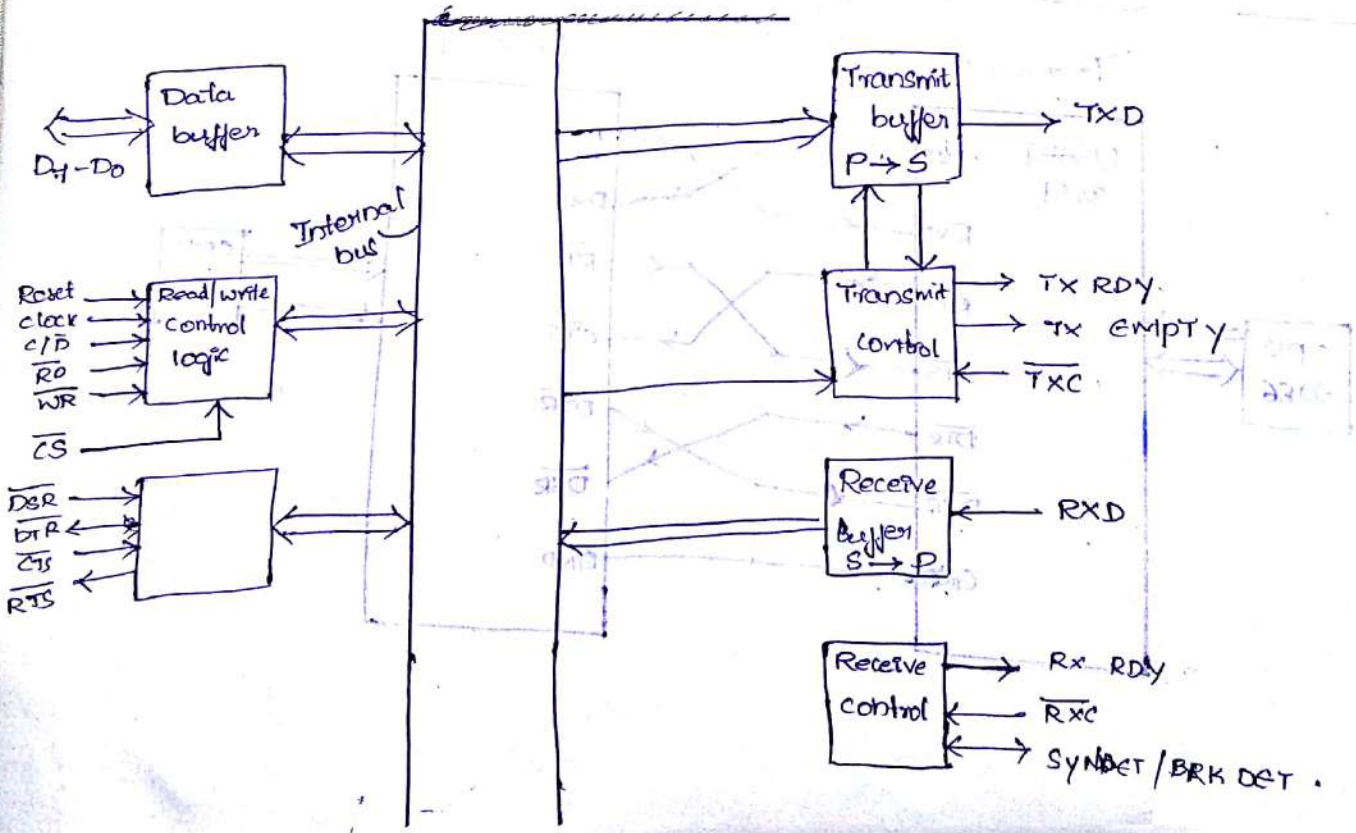


Pin configuration of 8251.



USART - Universal Synchronous Asynchronous receiver transmitter.

Functional block diagram of 8251 USART.



There are 8 data lines in 8251 and these are bidirectional. Data is transmitted or received by the cpu on these lines.

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	
0	1	1	0	8086 writes to the command Register using data bus ($D_0 - D_7$)
0	1	0	1	8086 reads the status of USART status register
0	0	1	0	8086 writes the byte data for transmission
0	0	0	1	8086 reads the byte data after reception and Assembly by USART.
1	X	X	X	USART is not selected for any operation.

Brief functional description of 8251 USART.

→ There are 8 parallel lines $D_7 - D_0$ which are connected to a system data bus, so that control or status words can be transferred between cpu and USART.

→ The chip select of USART is connected to address decoder

→ 8251 has 2 addresses as per $C/\overline{D}$ signal, when $C/\overline{D} = 1$ it is a control word address. When $C/\overline{D} = 0$, it is a data addresses.

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	
0	1	1	0	8086 writes to the command
0	1	0	1	8086 (reads) the status of Reads USART

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
0	0	1	0	8086 writes the byte data
0	0	0	1	8086 µp reads the byte data.
1	0	0	0	chip is not selected.

The clock i/p of 8251 is connected to a system clock for synchronisation.

→ Modem :- When 8251 is switched 'ON' it sends a $\overline{DTR}$ (Data terminal ready) signal to Tx modem, the Tx modem sends $\overline{DSR}$ (Data set ready) signal back to USART.
When 8251 is ready for serial transmission, it sends $\overline{RTS}$ (Request to send) signal to transmit Tx modem. The transmitting modem gives green signal for transmission by activating $\overline{CTS}$ (clear to send) signal.

→ The shift registers in transmit and receive buffer blocks require clocks for serial data-in and serial data-out.

$\overline{TxC}$ and $\overline{RxC}$ clocks are used for this purpose. Usually both are shorted and connected to a common source.

→ 8251 is double buffered. The transmitter section has holding buffer, shift registers ie, 2 buffers. When one character is loaded to holding buffer, another character can be moved out of the actual transmit shift register.
The TX RDY (Transmitter Ready) goes high, when holding buffer is empty. This is an indication to the CPU that next byte can be return to the holding buffer.

→ The TxRDY is connected to INTR of cpu.

→ The RxRDY (Receive Ready) goes high, when a character has been assembled in the receiver buffer which can be readily read by the cpu.

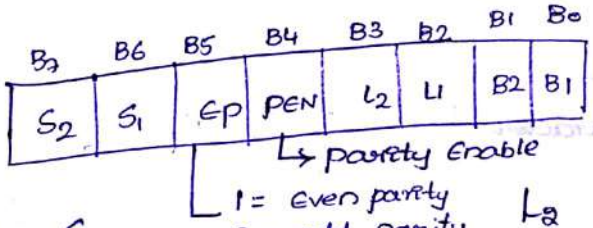
→ The TXEMPTY pin goes high (active) when both buffers in the transmitter section become empty.

→ SYNC-Detector / Break detector :- When USART is operating in Synchronous mode if RxD is low for 2 character times, this pin goes high, This is an indication for of break in communication.

for Synchronous communication, when 8251 finds a specified sync character in the incoming string of data. Then, this pin goes high.

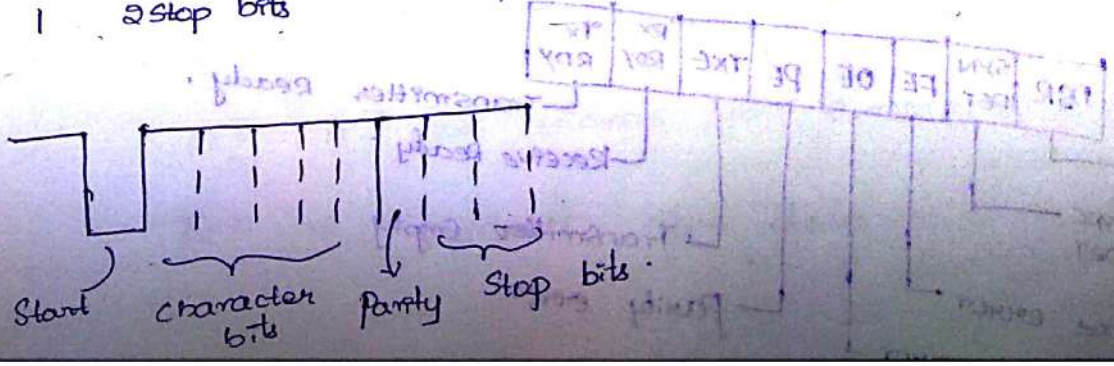
22/1/2013

8251 USART MODE WORD Register FORMAT.



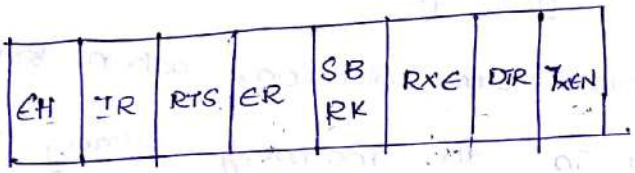
S_2	S_1	
0	0	Invalid
0	1	1 Stop bit
1	0	1½ Stop bit
1	1	2 Stop bits

L_2	L_1	
0	0	5 bits per character
0	1	6 bits per character
1	0	7 bits per character
1	1	8 bits per character



B_2	B1	
0	0	Synchronous Mode & Transmission / Reception at $\overline{TxC}$, $\overline{RxC}$
0	1	$\overline{TxC}$ // Asynchronous
1	0	$\overline{TxC}$ /16 Asynchronous
1	1	$\overline{TxC}$ /64 Asynchronous.

Command word Register format :-



EH = Enter Hunt Mode.

= Enable Search for sync character.

IR = Internal Reset.

RTS = Request to Send

ER = Error Reset

SBRK = Send Breaks character.

RxE = Receive Enable

DTR = Data Terminal Ready.

TxEN = Transmit ENable.

Trans Enable.

0000 0001

0 1 h

RR Enable

0000 0010

0 2h

Status word Register Format :-

Transmitter Ready.

Receive Ready.

Transmitter Empty.

Parity error

Frame error

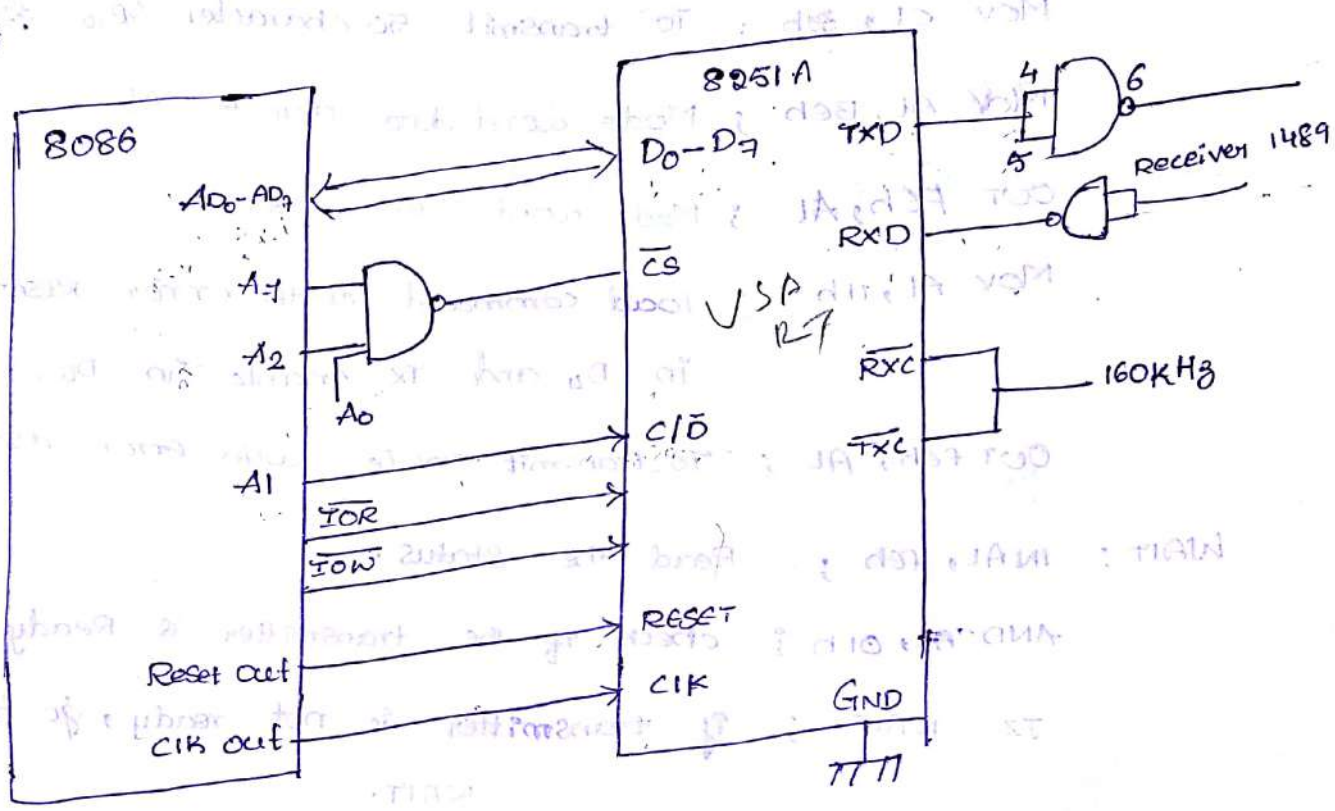
Over Run error

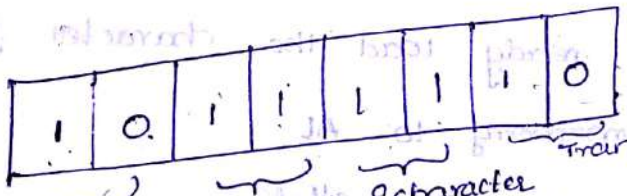
SYNC DET

ata set eady

Write an ALP in 8086 for transmitting 50 characters which are stored in the Memory location 02010h using 8251. The design may be Implemented with following conditions. (Starting from)

a) even parity Enable b) 1½ Stop bits c) 8 bit character length
d) Transmitter clock frequency 160KHz e) baudd rate 10KHz. transmitter clock freq.





is scaled ie, $\frac{160KHz}{10KHz} = 16$.

So, mode word is BEh

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	1	1	1	1	0
F				E			

Transmitter bit Enable

FEh is the address for command

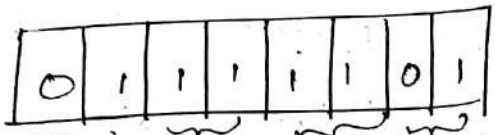
FCh is address for data

```
ASSUME CS : CODE5.

CODE 5 SEGMENT

START : MOV AX, 0200h
        MOV DS, AX
        MOV SI, 0010h
        MOV CL, 32h ; To transmit 50 character 50₁₀ = 32h
        MOV AL, BEh ; Mode word data BEH to AL
        OUT FEh, AL ; Mode word to USART.
        MOV AL, 11h ; Load command with error Reset.
                      in D4 and Tx Enable in D0.
        OUT FEh, AL ; To transmit mode with error reset.

WAIT :  IN AL, FEh ; Read the status.
        AND AL, 01h ; check if the transmitter is Ready.
        JZ WAIT ; if transmitter is not ready, go to
                  WAIT.

        MOV AL, [SI] ; if ready load the character from
                       memory to AL.
        OUT FCh, AL ; Data transmitted.
        INC SI ; point to next byte.
        DEC CL ; Decrement counter
        JNZ WAIT ; Repeat them for 50 characters.

        INT 3h ;
CODE 5 ENDS

        END START ;
```

→ Write an ALP to receive 100 bytes of data stream and storing at the memory location 3000 : 4000h using 8251 USART the format of data is given below.

(i) Even parity enable (ii) 1 Stop bit (iii) 8 bit character (iv) frequency 160KHz (v) Baud rate 160KHz.

26/2/2013



Transmitter is scaled 160KHz / 160KHz = 1

Modeword = 7Dh.

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	1	0	1	0	0

↑ Error Reset

↳ Receiver enable

} CMD

= 14h.

3000 : 4000

30000
4000
34000

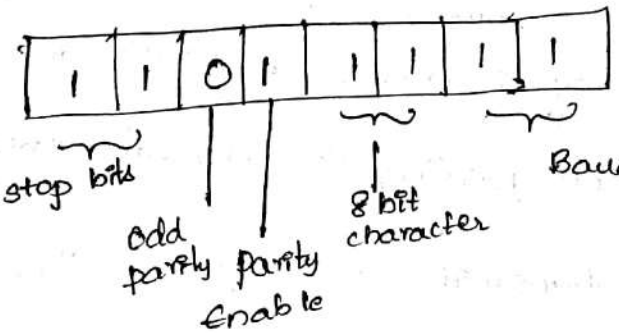
physical address

```
ASSUME CS : CODE 6.
CODE 6 SEGMENT
START : MOV AX, 3000h.
        MOV DS, AX ; the base address is loaded into segment register
        MOV SI, 4000h ; The offset address is loaded into SI. So,
                        we are pointing to physical address 34000h.
        MOV CL, 64h ; To transmit 100 bytes. (100_10 = 64h.)
        MOV AL, 7Dh ; Mode word data 7Dh to AL.
        MOV FCh
        OUT FCh, AL ; Mode word to
        MOV AL, 14h ; load command with error Reset
                      and to enable the receiver.
```

```
OUT FEh , AL ;  USART is Initialized with Command Word.
                word.
READY: IN AL, FEh ;  Read the status.
WAIT
       AND AL, 02h ;  check if receiver is ready.
       JZ READY.   ;  WAIT till the receiver is ready.
       IN AL, fCh  ;  The received character is transferred to AL.
       Mov [SI], AL ;  The received character is transferred to
                       memory.
       INC SI
       DEC CL
       JNZ READY ;  Repeat the loop for 100 bytes.
       Mov AH, 4Ch ;
       INT 21h ;
CODE G ENDS
    END START
```

→ Write an Initialization sequence to operate 8251 in Asynchronous mode with 8 bit character, size moderate with baud rate 64, 2 stop bits and odd parity enable. The 8251 is interfaced with 8086 at address 082h.

A,



Mode word = CF h

Mov AL, DFh

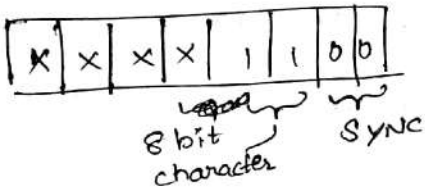
OUT 82h, AL

Mov AL, 40h // Do internal Reset for USART.

OUT 82h, AL

→ write instruction sequence to initialize 8251 in Synchronous mode with even / par Single character and 8 bit size.

A)



Assume XXXX as 0000.

So mode word is 0ch

Mov AL, 0ch

OUT 82h, AL

Mov AL, 40h // Do Internal Reset for USART.

OUT 82h, AL

28/2/2013 RS 232

RS 232 C

logic 1 -3V to -12V underload

-25V No load

logic 0 +3V to +12V underload

+25V No load.

* Give an overview of RS 232 C Serial data standard. Draw the Interface circuit for data conversion for TTL to RS 232C, RS 232C to TTL.

A, TTL is Transistor Transistor logic where bit 1 is represented in the range 2.5V to 5V and bit 0 is represented in the range less than 0.8V. The voltage levels for RS 232C is given below.

logic 1 -3V to -12V underload

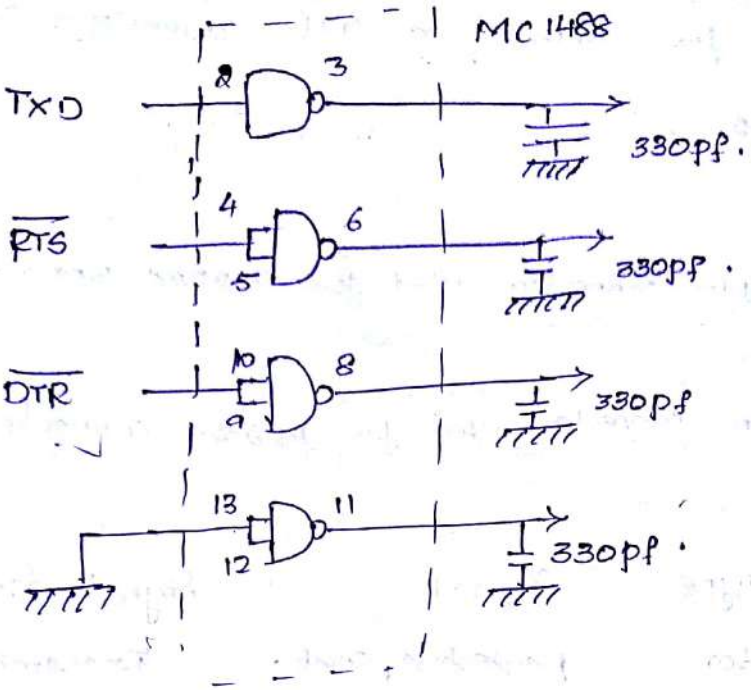
-25V No load.

logic 0 +3V to +12V underload

+25V No load.

RS 232C provides better Noise immunity compared to TTL. Voltages such as ±12V are permanently used in which case logic 1 is -12V and logic 0 is +12V

MC 1488 is a four quad · TTL RS232 converter whose configuration is given below.



MC 1488 for converting TTL to RS232C.

MC 1488 for

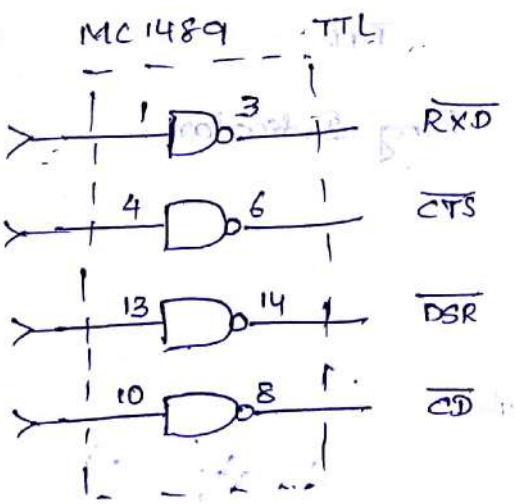
Pin 14 = +12V

Pin 1 = -12V

Pin 7 = GND.

→ for RS 232 to TTL conversion consists of Integrator circuit MC 1488 as shown above.

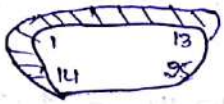
The standard Interface for TTL to RS232 conversion consists of IC's MC 1488 as shown above. This requires +12V, -12V power supply capacitor of 330pf is used to reduce the crosstalk between adjacent wires.



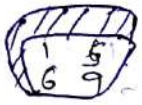
Pin 14 = +5V

7 = GND.

Ic MC 1489 is used for RS232 to TTL conversion and this requires only single 5V.



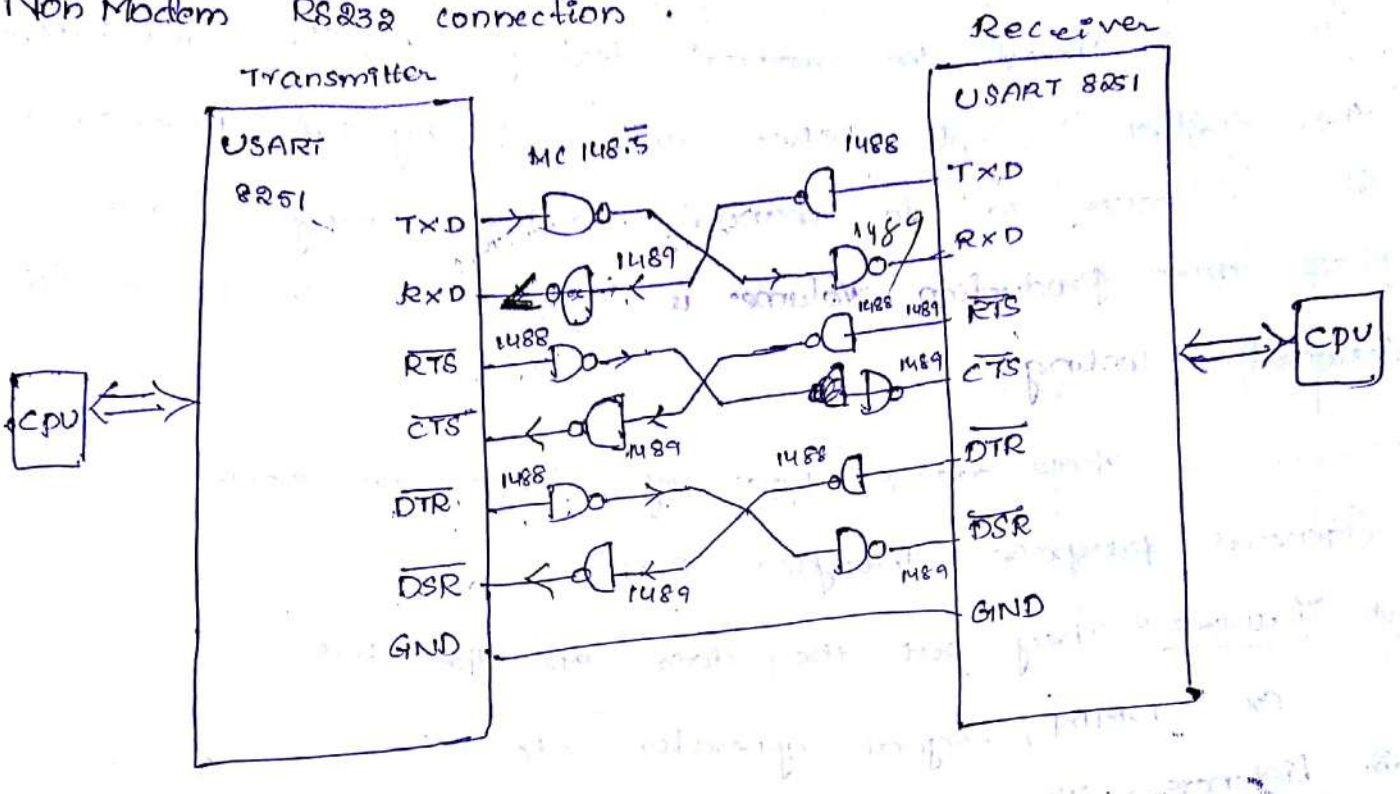
25 pin connector used for RS232C connection.



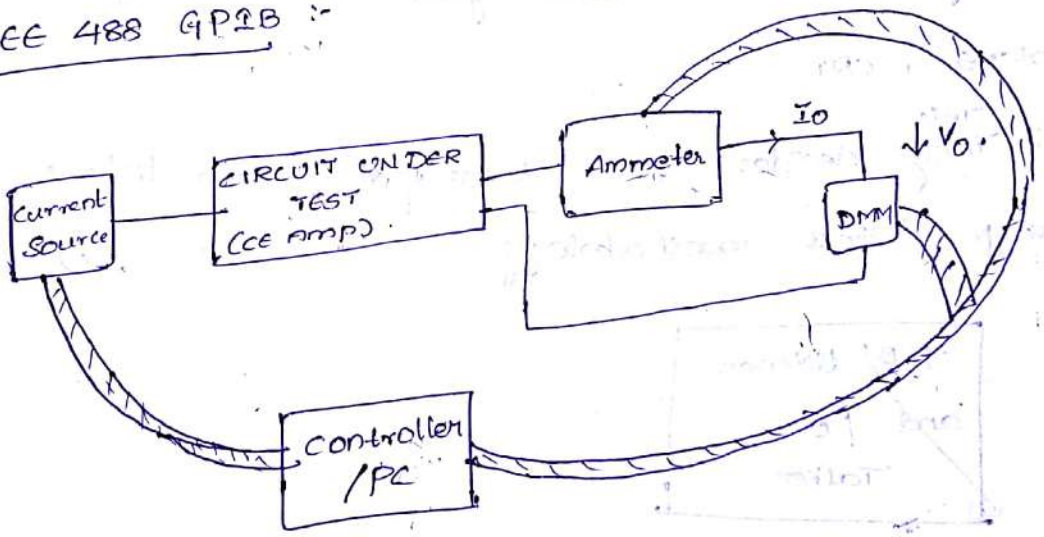
9 pin connector used for RS232C connection.

9 pin DType Connector	25 pin DType connector	Signal protective comb.	Signal direction in Transmitter
-	1	protective comb	—
3	2	TXD	OUT.
2	3	RXD	IN
7	4	$\overline{RTS}$	OUT
8	5	$\overline{CTS}$	IN
6	6	$\overline{DSR}$	IN
5	7	GND	
1	8	$\overline{CD}$ (carrier detect)	IN
-	12, 13, 14, 15, 16 17, 18, 19	Secondary channel Signals	
4	20	$\overline{DTR}$	OUT.
9	22	Ring Indicator	IN

Non Modem RS232 connection.



IEEE 488 GPIB :-



Hewlett packard (HP) has covered with the standard bus for the laboratory test equipment such as DMM's, Signal generator, freq counters, power meters etc., to facilitate Automatic test. This was adopted by IEEE in 1975. This was later revised in 1978, it is called IEEE 488 (GPIB this & the Automatic testing has 2 advantages change compared to manual testing.

The testing is faster and Accurate in ATE.

Though the manual testing is slow and inaccurate, the circuit concepts are better understood by this process. So, it is better to do manual testing during development phase when production Volume is high, we switch over to Automatic testing.

There are 3 types of devices on GPIB (General purpose Interface Bus)

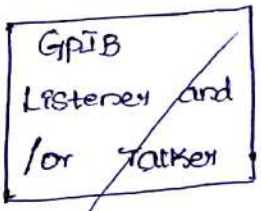
(1) Talkers :- They put the data on the bus.

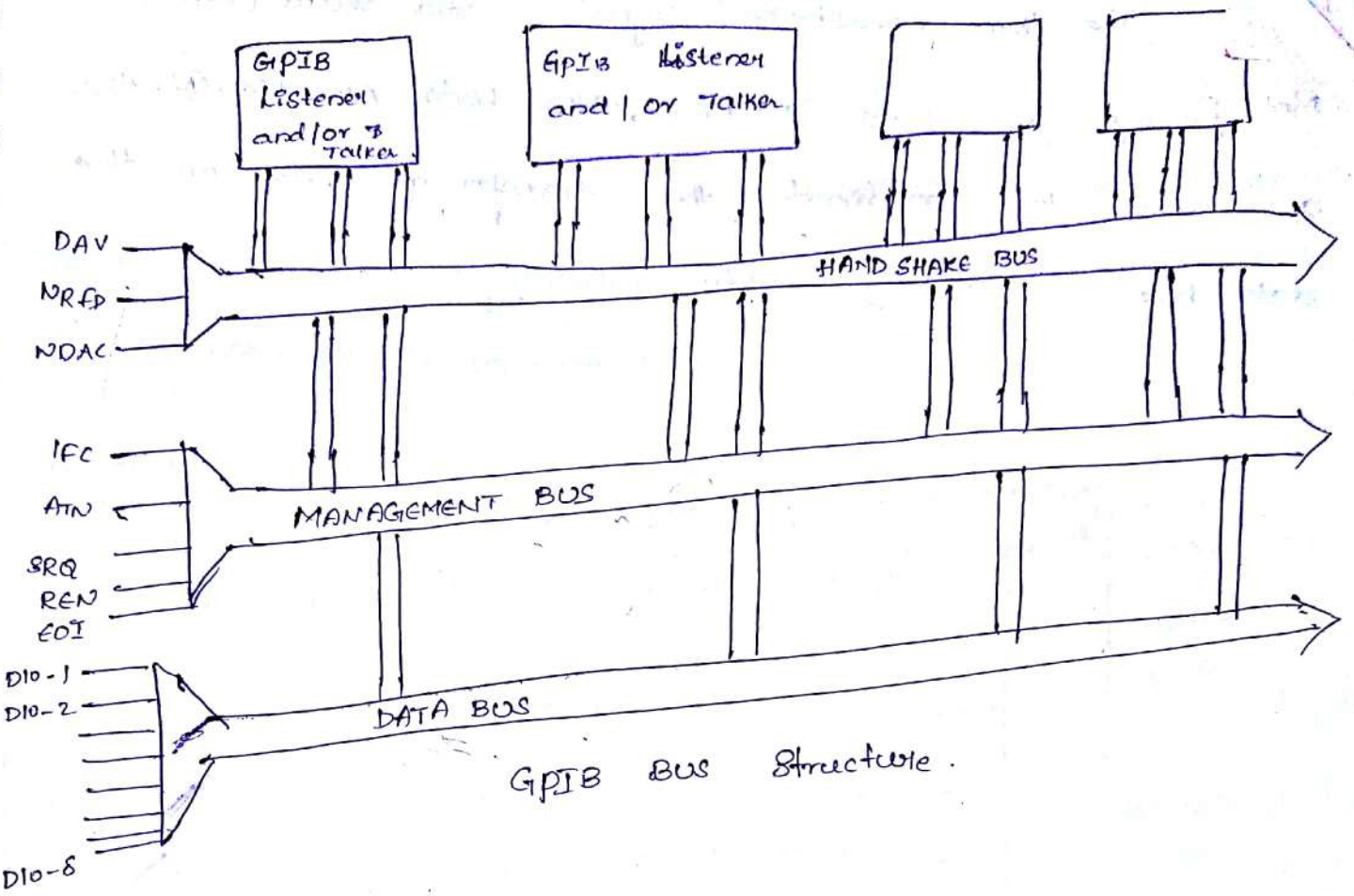
Ex. DMM, Signal generator etc.,

(2) Listeners :- They take the data from the bus for display

Ex:- printers, CRT

(3) Controllers :- This They decides who talks and who listens and also the specific test methodology.





GPIB Bus Structure.

→ The 5 management bus signals are given.

1, Interface clear (IFC) :- This resets all the devices from the bus.

2, Attention (ATN) :- The controller uses this signal to send command words to all the devices.

3, Service Request (SRQ) :- The devices are the restiquests (listeners or talkers) use this signal to interrupt the controller for a specific job

4, Remote Enable (REN) :- The controller sends this signal to the devices to disable the front panel controls.

5, End or Identify (EOI) :- This is used by the Talker to indicate that the transfer of block of data is complete.

The three handshake signals data valid (DAV), Not Ready for Data (NRFD), Not Data Accepted (NDAC) are used to coordinate the transfer of data on the data bus

21/1/2013

UNIT-3

I/O Interface :- 8255 PPI, various modes of operation and Interfacing to 8086, Interfacing keyboard, display, Stepper motor interfacing, D/A and A/D converter.

→ Port is a place where data is loaded or unloaded by microprocessor.

→ The port can be In-port or out-port.

→ The µp takes the data from i/p device using In-port. for example, keyboard is the i/p device to the computer which is interfaced using In-port.

→ The µp writes to the o/p device using out-port. for example, CRT is the o/p device which is connected to the computer.

→ The In-ports should not draw more current. So Tristate buffers are used when not in use. The out-ports should be able to drive more number of devices so they should source more current.

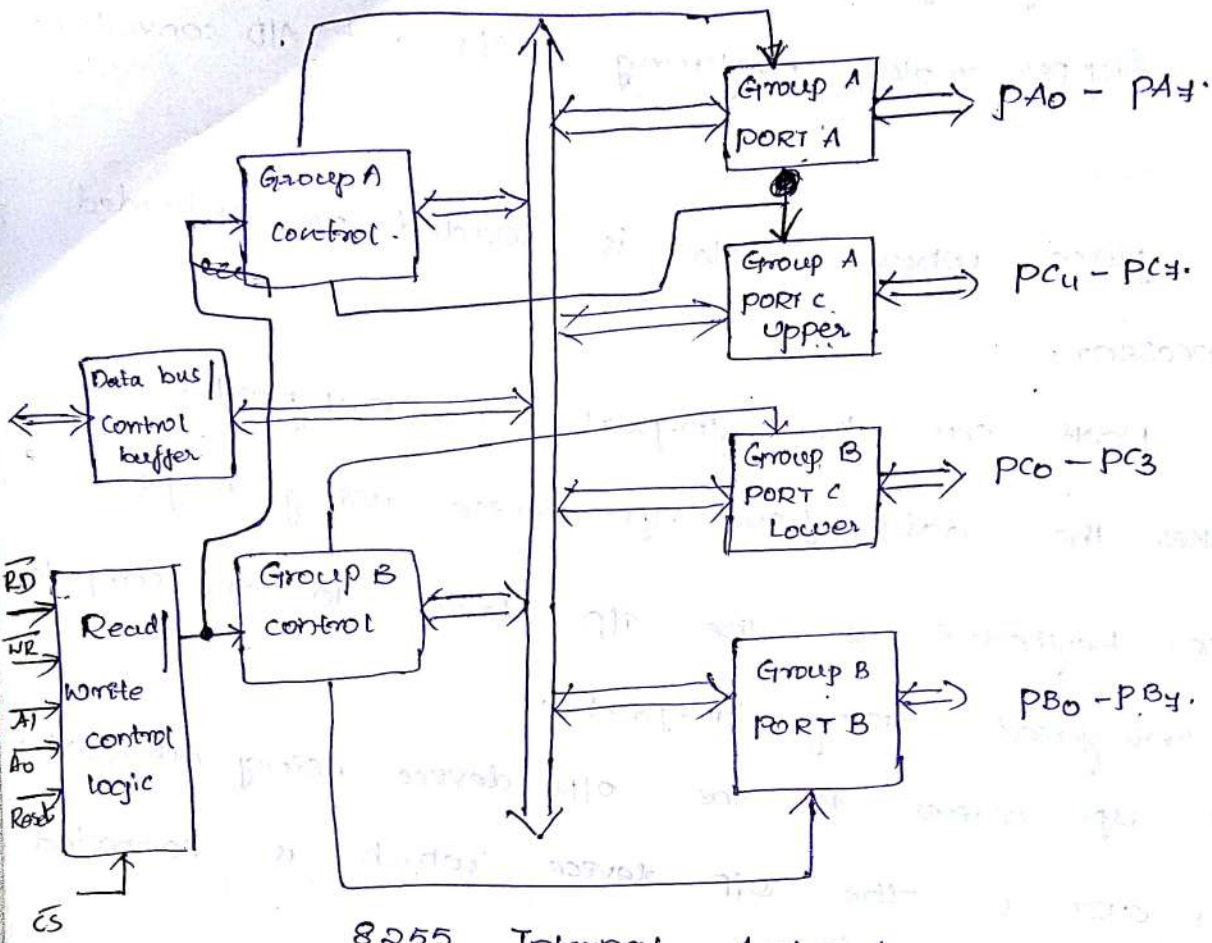
→ out-ports are latched so that the data is available for longer time.

→ ports are classified as (i) Nonprogrammable and (ii) programmable.

Eg :- Intel 8212 / 74LS 245 is an example of In-port, which is not programmable.

→ 8255 PPI (Programmable peripheral Interface) :-

8255

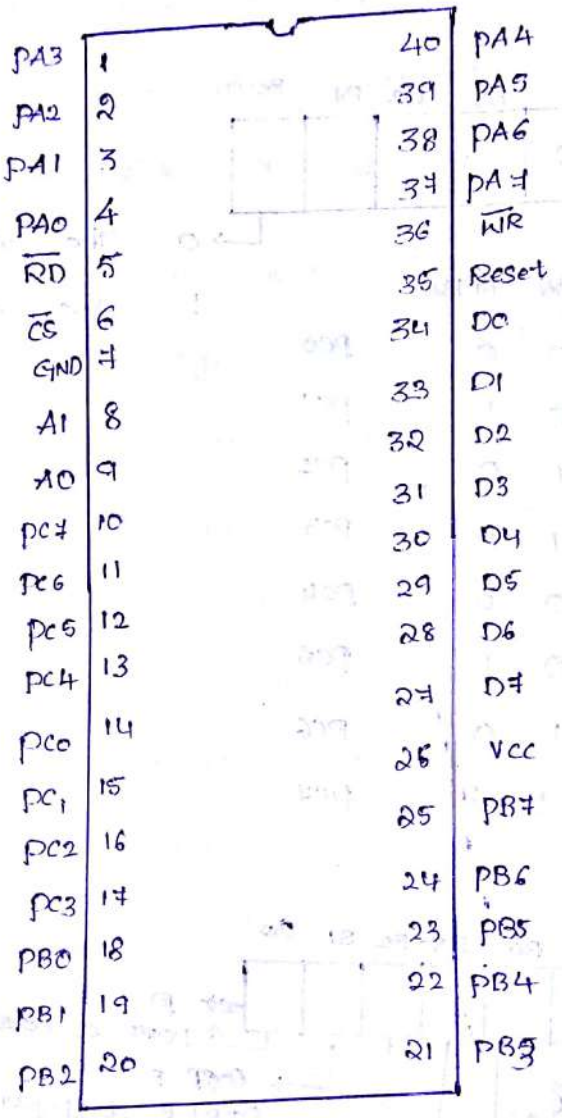


8255 Internal Architecture.

A_1	A_0	
0	0	port A.
0	1	port B
1	0	port C
1	1	CWR (command word register)

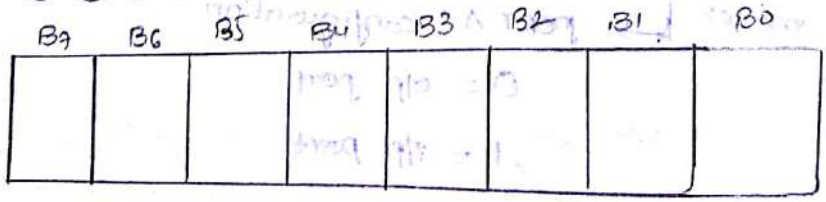
28/1/2013

8255 pin configuration :-



The 8255 is configured by the microprocessor by programming command word register CWR.

CWR (command word Register) :-

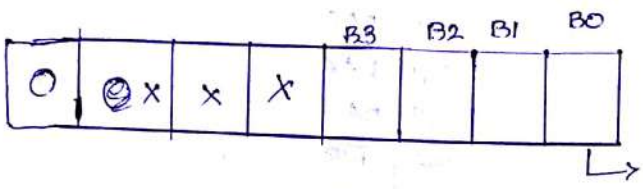


The 8255 can be made to operate in BSR (Bit Set Reset Mode) (or) I/O mode (Input output port Mode).

If B7 is zero then 8255 operation is in BSR Mode.

If B7 is one then 8255 operation is in I/O Mode.

BSR Mode :-

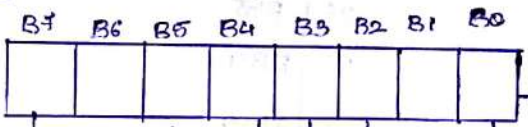


→ 0 The relevant pc bit is zero

1 The relevant pc bit is one

B_3	B2	B1	B0	
0	0	0		pc0
0	0	1		pc1
0	1	0		pc2
0	1	1		pc3
1	0	0		pc4
1	0	1		pc5
1	1	0		pc6
1	1	1		pc7

I/O Mode :-



'1' for I/O Mode

Mode of Group A.

B6 B5

0 0 mode 0

0 1 mode 1

1 X mode 2

PORT C lower pc0-pc3 configuration 0 = o/p 1 = i/p

PORT B configuration 0 = o/p 1 = i/p

Mode of Group B 0 = mode 0 1 = mode 1.

PORTC upper pc4-pc7 configuration 0 = o/p 1 = i/p.

PORT A configuration.

0 = o/p port

1 = i/p port

In Mode 0, the ports ie, port A, port B, port C works as a Simple I/o ports. In Mode 1, port A and port B can be programmed as I/O ports with handshake. The handshake signals are provided by port C. In Mode 2, which is applicable only for port A. The port A acts as a bidirectional.

→ Interface an 8255 with 8086 to work as an I/O port, initialize port A as o/p port, port B as i/p port, port C as o/p port. port A address should be 0740h. Write a program to sense switch positions SW0-SW7 connected to port B.

The sensor pattern should be displayed on port A to which 8 LEDS are connected while port C locates displays no. of ON switches out of the total 8 switches.

A, Do the Design in Mode 0.

A, The control word is formed as given below.

B7	B6	B5	B4	B3	B2	B1	B0
1	0	0	0	0	0	1	0

8 2h.

The control word for above problem is 82h

23/1/2013

8255 PORTS	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0 μp
PORT A	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0

⇒ 0740h

23/1/2013

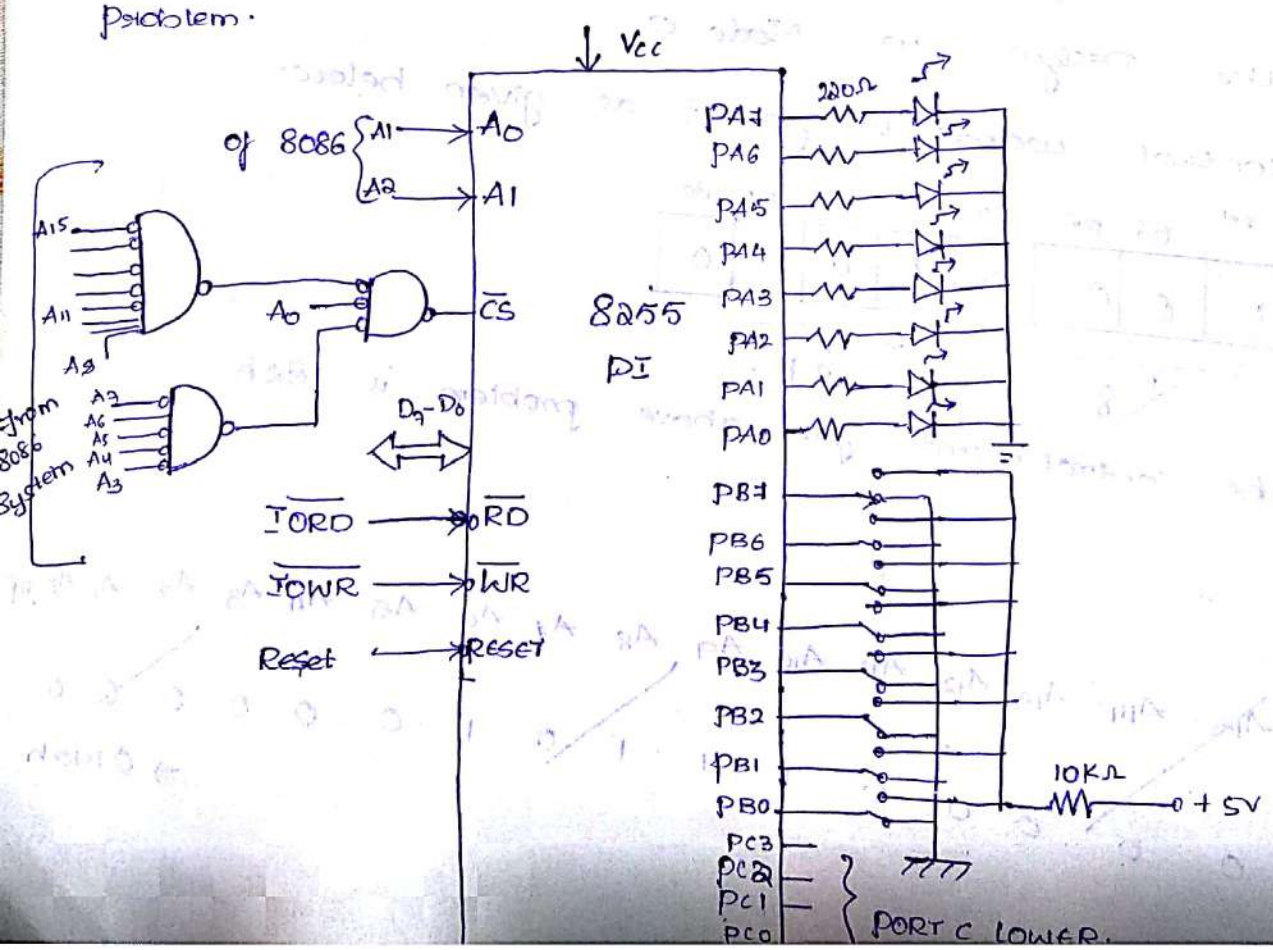
8255 A_1 A_0

0 0 PORT A — 10 — PORT C

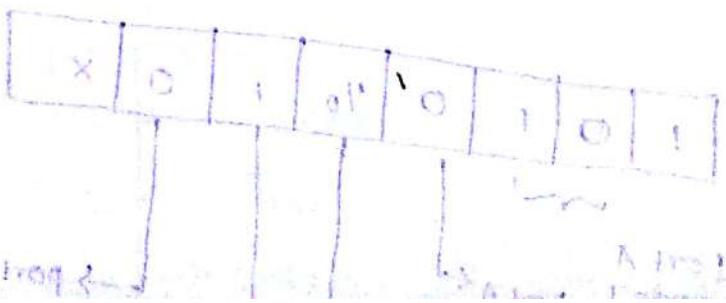
0 1 PORT B — 11 — CWR.

8255 PORTS	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Mp
PORT A	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0740h
PORT B	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	= 0742h.
PORT C	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	0	= 0744h
CWR	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	= 0746h

→ 8255 Interfacing with 8086 and I/O devices for the above problem.

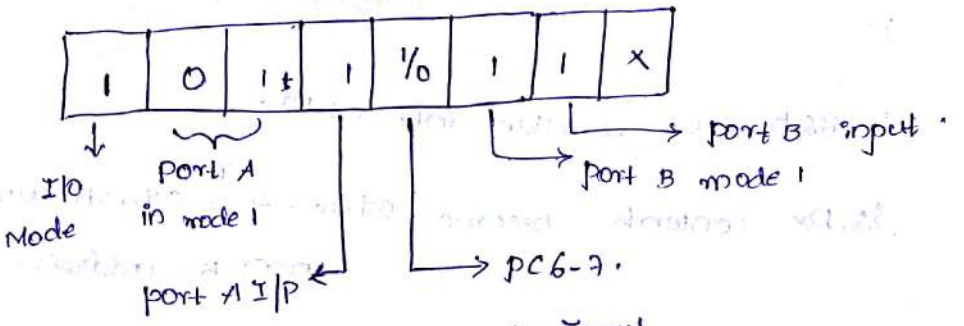


```
MOV DX, 0746h   ; address of CWR 0746h is loaded to DX.
MOV AL, 82h     ;
OUT DX, AL      ; 82h is written into CWR.
SUB DX, 04h     ; So, Dx contents become 0746 - 4 = 0742h which is
                  PORT B address.
IN AL, DX       ; Read PORT B for switch position.
SUB DX, 02h     ; Dx now points to PORTA.
OUT DX, AL      ; Display LED's as per switch position.

MOV BL, 00h
MOV CH, 08h
YY: ROL AL, 01
    JNC XX
    INC BL
XX: DEC CH
    JNZ YY
    MOV AL, BL
    ADD DX, 04
    OUT DX, AL
    HLT.
```



28/1/2013

Control word for Both port A, port B as inputs in mode 1.

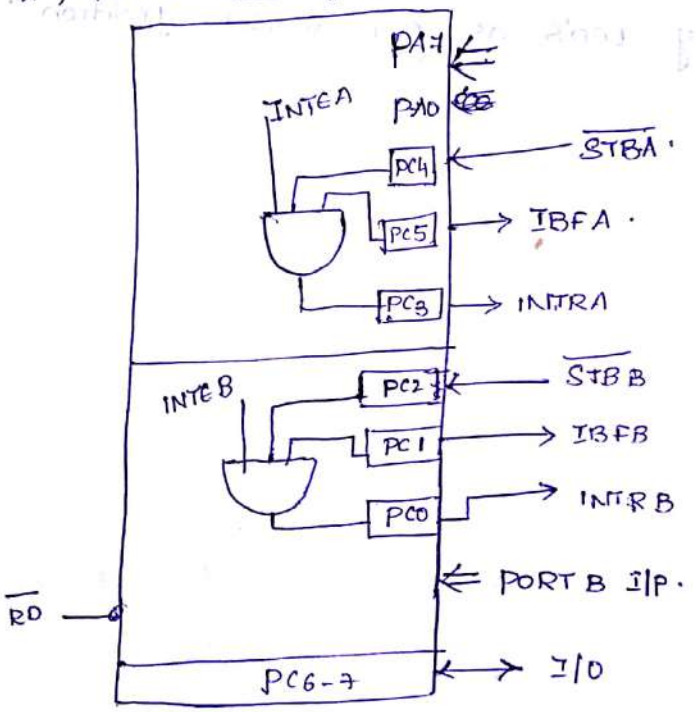


1 = Input

0 = output.

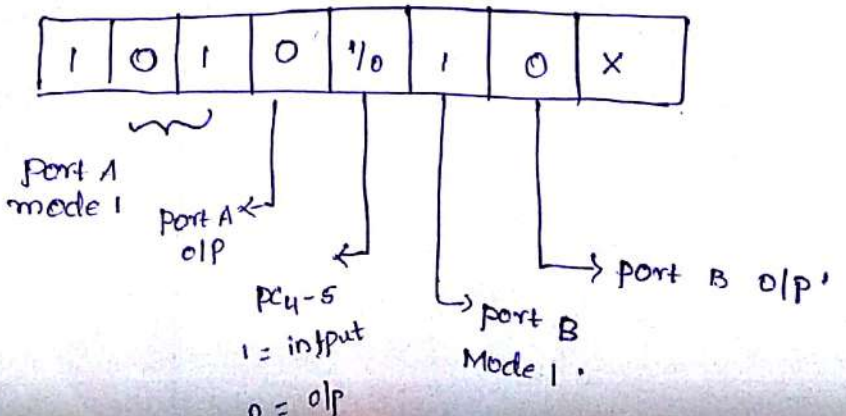
Mode 1 :-

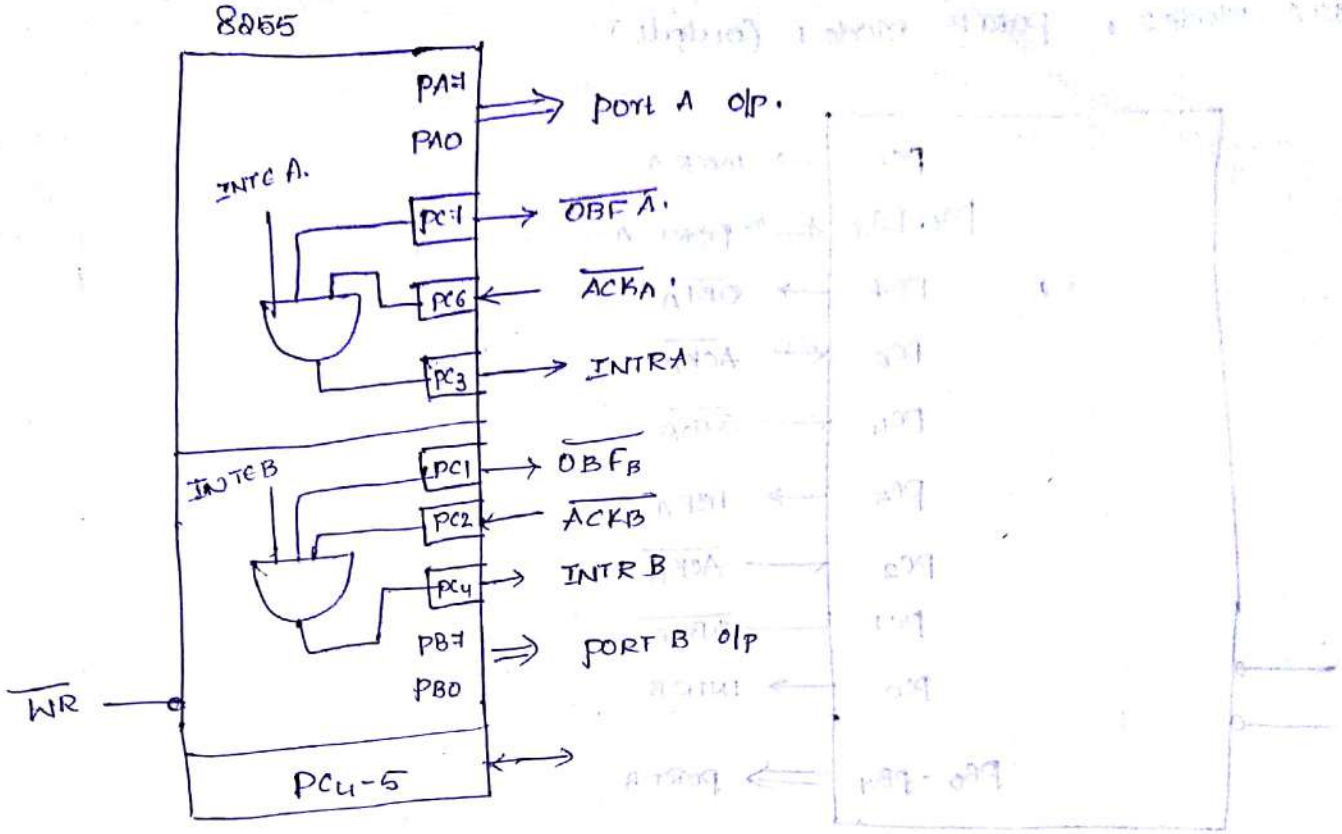
(port A, port B are I/P)



Mode 1 :-

PORT A, PORT B as output ports.





Mode 2 :-

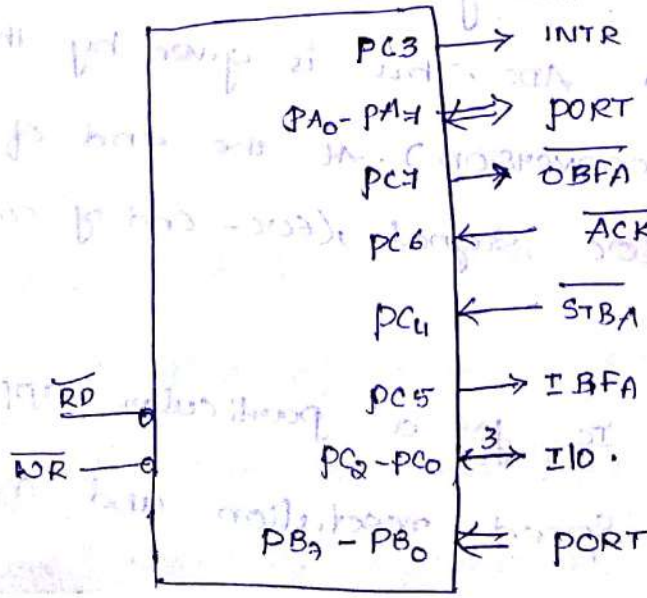
→ This option is available only for PORT A.

→ PORT A functions as Bidirectional port in Mode 2.

→ While PORT A is in mode 2, port B can be in Mode 0 or Mode 1 as i/p port or o/p port.

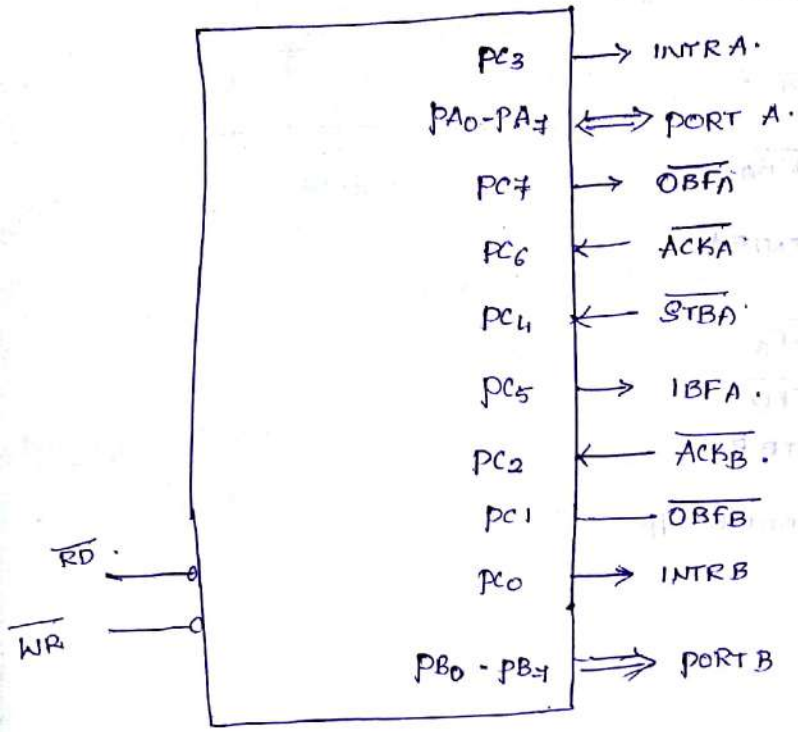
→ Mode 2 option is not available for port B.

PORT A Mode 2, PORT B Mode 0 (Input)



Mode 2:

PORTA Mode 2, PORTB Mode 1 (output)



29/1/2013

Interfacing of A/D converter :-

IC's used for A/D conversion:

a) ADC 0808/0809 8bit successive approximation converter.

b) IC LH109, Make Intersil 12bits dual slope A/D converter.

The general algorithm for ADC interfacing contains following steps

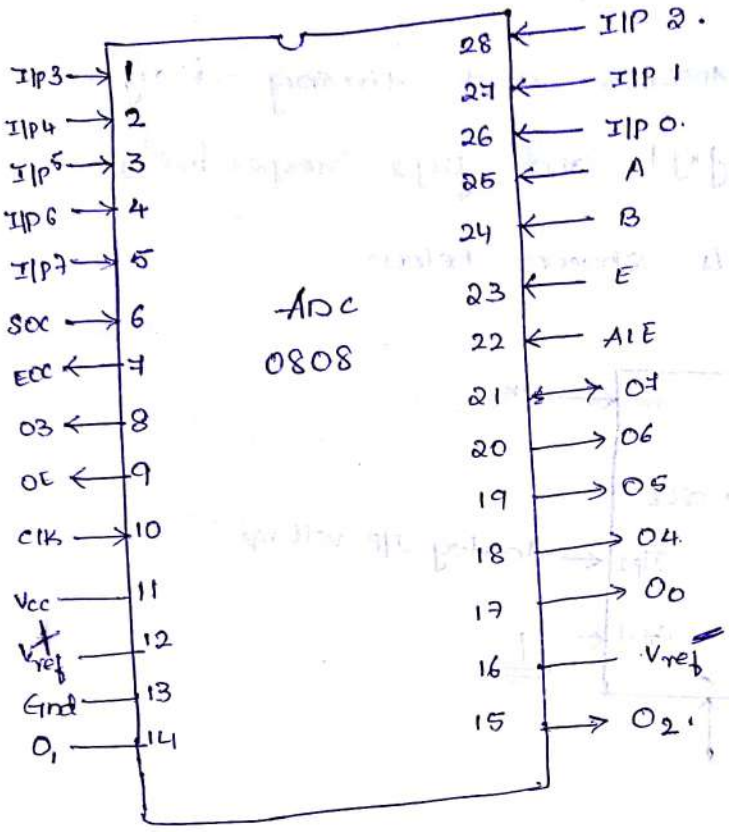
1) ~~Ensure~~ Ensure the stability of Analog input.

2) Issue SOC command to ADC. This is given by the µp. (SOC - Start of conversion). At the end of conversion, the ADC chip gives EOC signal, (EOC - End of conversion) to the µp.

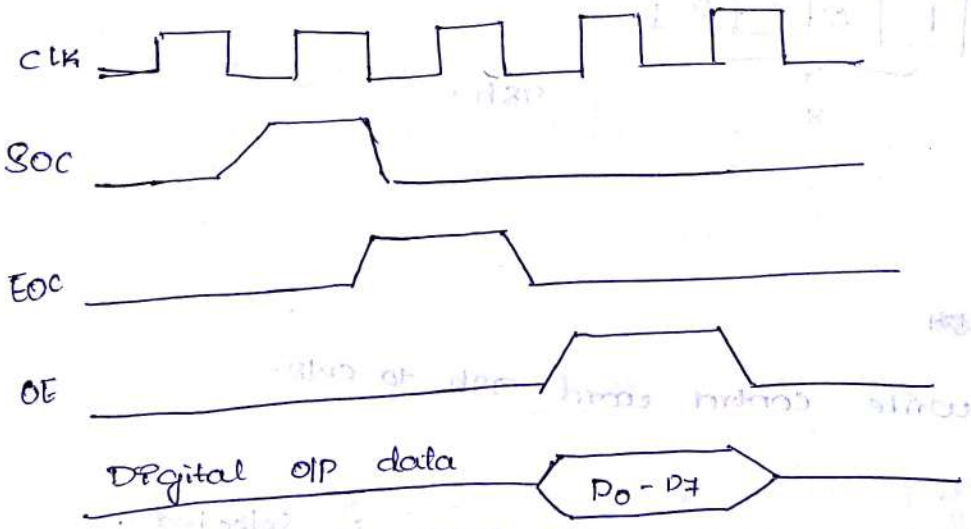
3) The selection of ADC IC for a particular application depends upon the speed, resolution and cost factor

ADC 0808/0809

Fig ①



Timing diagram :- Fig ②



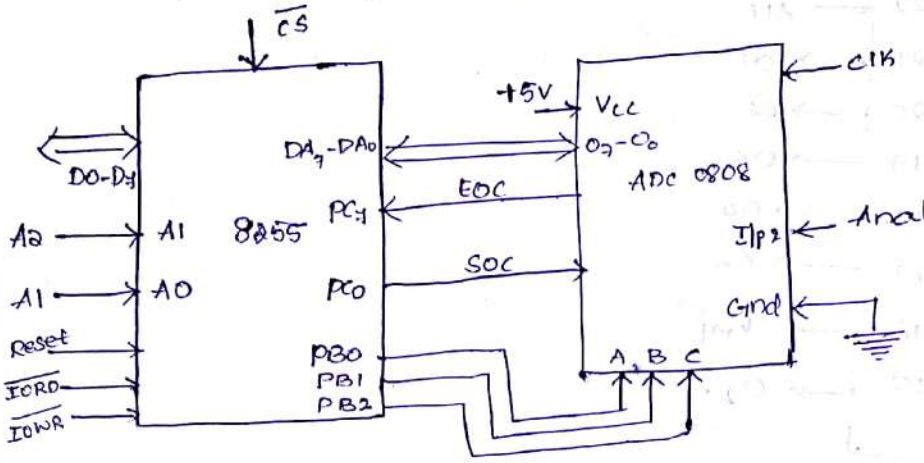
→ Interface 0808 with 8086 using 8255 ports.

use port A of 8255 for transmitting digital data o/p of ADC to cpu and port C for control signals. Assume that analog i/p is present at i/p 2 of ADC and

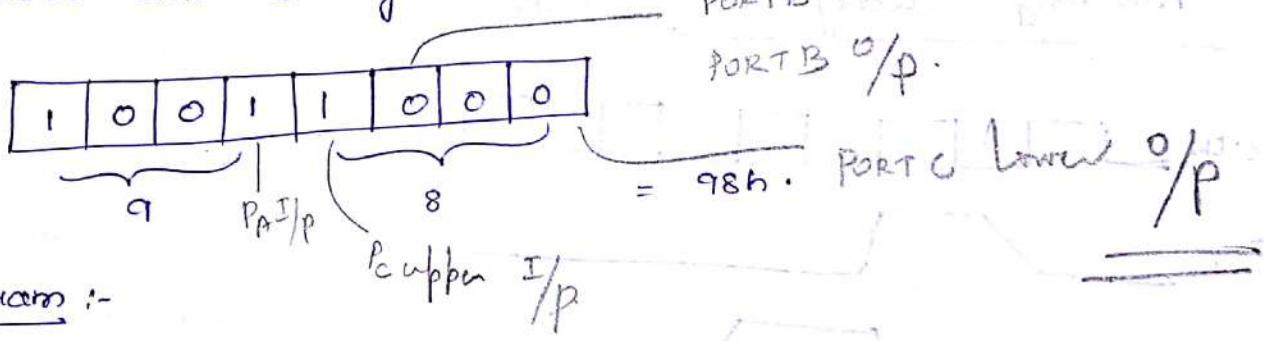
clock i/p of suitable freq. is available for ADC. Draw the schematic and write the required ALP.

Sol :- The pin configuration of ADC0808 and timing diagram of ADC0808 is shown in fig 1 and fig 2 respectively.

The Interfacing diagram is shown below.



The 8255 CWR is given below.



Program :-

```
Mov AL, 98h   ; CWR
OUT CWR, AL ; write control word 98h to CWR.
Mov AL, 02h
Out PORTB, AL ; Analog I/p 2 of ADC0808 is selected.
Mov AL, 00h
OUT PORTC, AL ; ——
Mov AL, 01h ;
OUT PORTC, AL ; _|‾
Mov AL, 00h
OUT PORTC, AL ; _|‾|_ SOC command
```

```
WAIT: IN AL, PORTC ; Read EOC pulse till it is ready.
      RCL ,
      JNC WAIT.
      IN AL, PORT A
      HLT
```

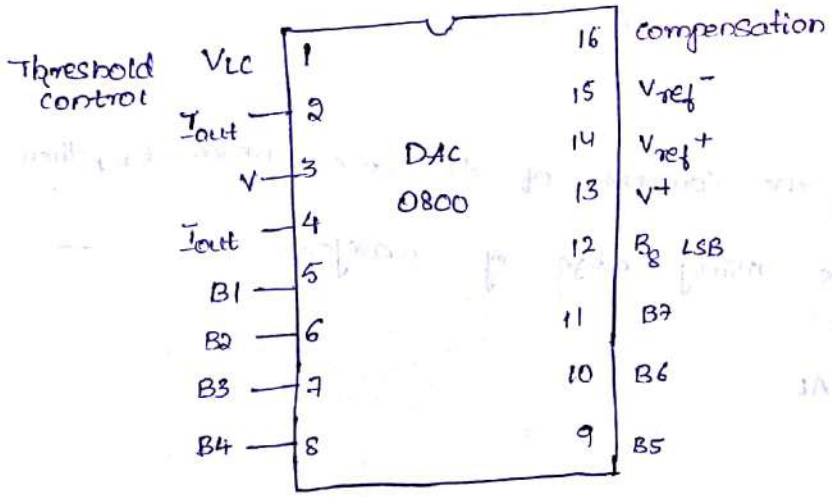
30/1/2013

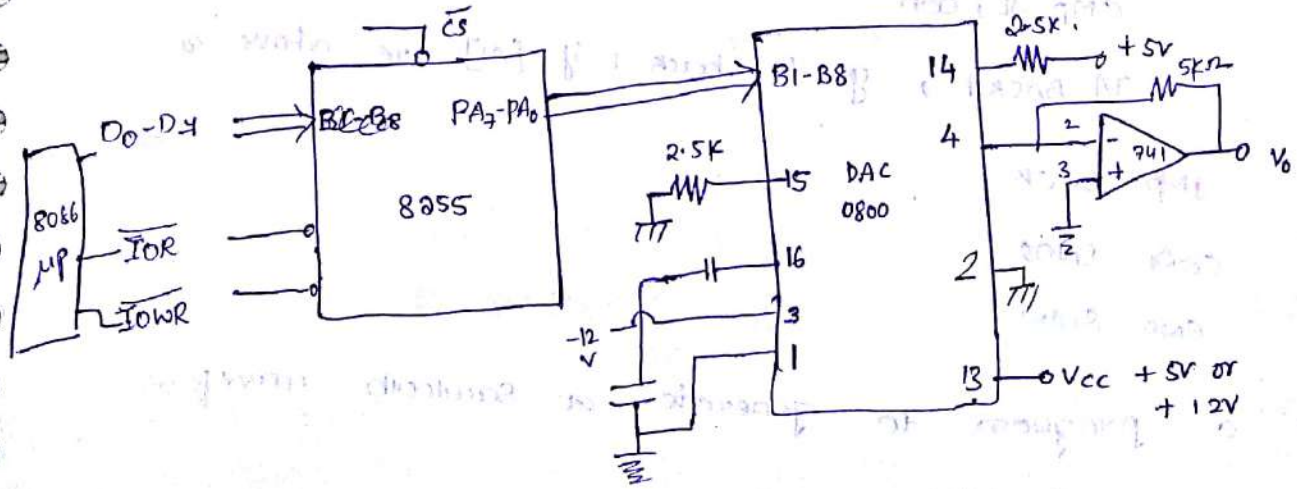
Digital to Analog converter Interfacing.

AD 7523 and

DAC 0800

The pin configuration of 0800 is given below.





The above circuit generates triangular waveform using DAC 0800 and 8255 PPI. op Amp is used to convert Analog Current to proportional analog voltage. port A is used as o/p port. The µP writes digital data i/p. port A which is transferred to DAC0800.

The Assembly language is shown below.

```
ASSUME  CS: CODE11
CODE11  SEGMENT
   START : Mov AL, 80h
           OUT CWR, AL ; control word is given to 8255 PPI
           Mov AL,00h
   BACK :  OUT PORTA, AL.
           INC AL
           CMP AL, FFh
           JB BACK ; If the contents of AL are below FF, then
   when [AL] = FF the rising edge of waveform
   BACK1 : OUT PORT A, AL
           DEC AL
           CMP AL, 00h
           JA BACK1 ; go to back 1 if [AL] are above 0
           JMP BACK
   CODE ENDS
   END START
```

CWR.

1	0	0	0	0	0	0	0

8 0h.

→ Write a program to generate a sawtooth waveform using AD7523 DAC

```
Assume  CS: CODE
CODE    SEGMENT.
START:  MOV AL, 80H     ; Initialise port A as output.
        OUT CINR, AL    ; port
AGAIN:  MOV AL, 00H     ; Start the ramp from 0V
BACK:   OUT PORT A, AL  ; Input 00H to DAC.
        INC AL          ; Increment AL to increase Ramp output
        CMP AL, 0F2H    ; Is upper limit reached ?
        JB BACK         ; If not, then increment the ramp.
        JMP AGAIN       ; Else start again from 00H.
CODE    ENDS
        END START
```

31/1/2013

Stepper motor Interfacing :-

Stator winding

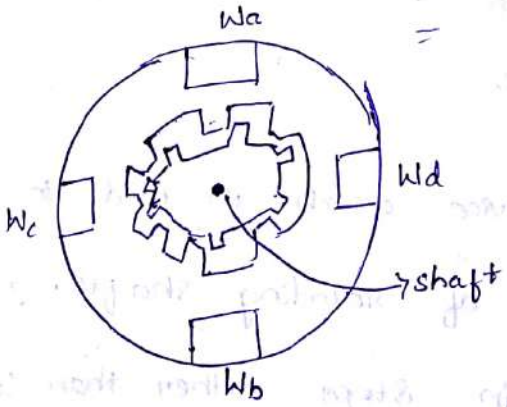
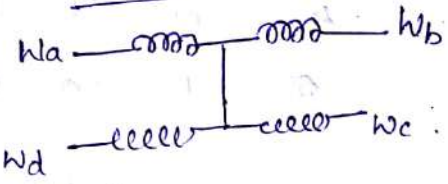
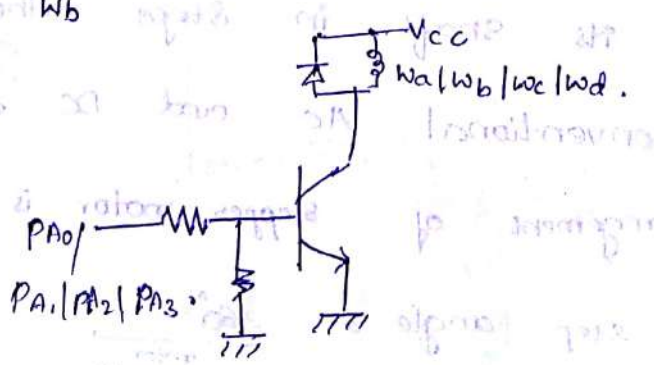
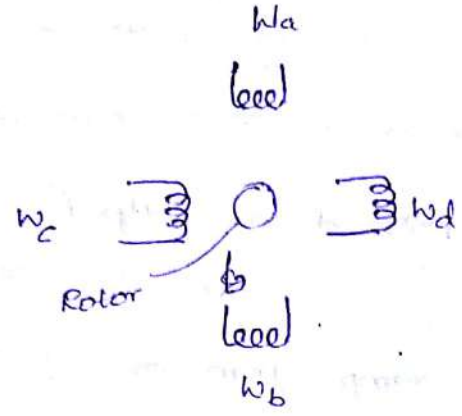


fig ②



When one





Excitation Sequence :- TABLE 1.

Motion	Step	PA_3	PA_2	PA1	PA0
CW clockwise	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
ACW	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

→ The stepper motor is a device which is used to obtain an accurate position control of rotating shafts. It employes protection of its shaft in steps rather than continuous motion in conventional AC and DC motors. The winding arrangement of stepper motor is given in fig 2.

The step angle = $\frac{360^\circ}{\text{No. of rotor teeth}}$

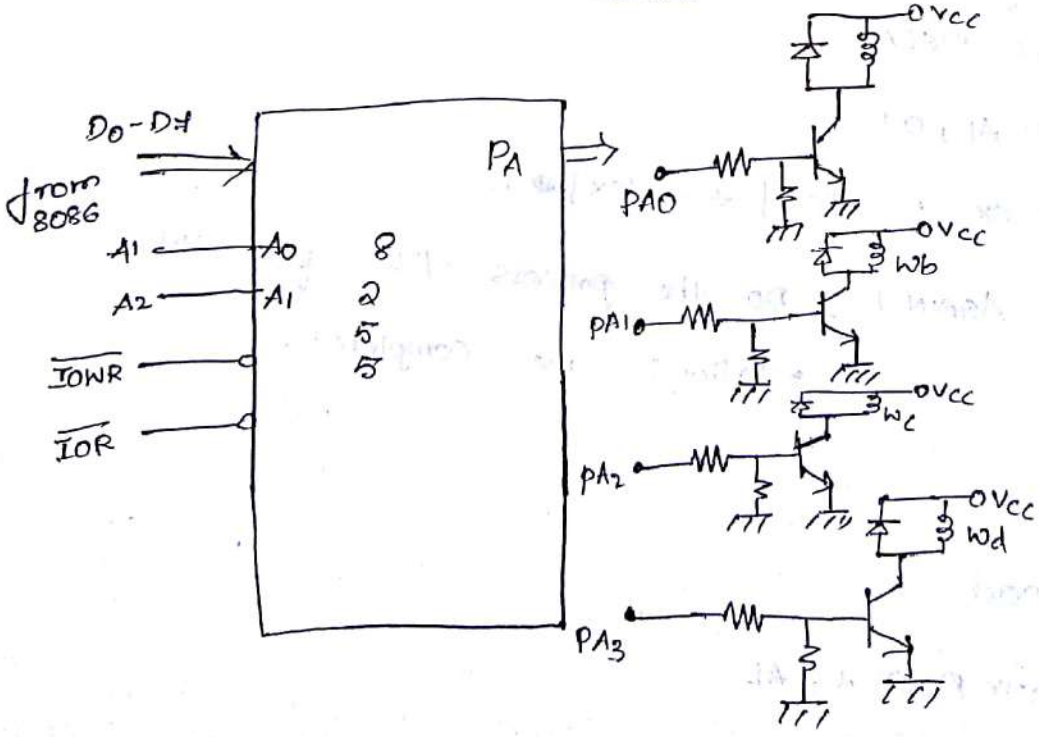
→ To rotate the shaft of stepper motor, a sequence of pulses is to applied to the windings W_a, W_b, W_c, W_d. The no. of pulses required for one revolution = No. of teeth.

↔ The Stator teeth and rotor teeth lock each other to shift in the position of the shaft.

→ The pulse applied to the microprocessor port which makes the transistor ~~teeth~~ ON and one of the winding point is grounded. As Vcc, is applied to the other end of the winding. The winding is energized and moves the rotor one step.

→ A simple scheme for rotation is a wave scheme is shown in table 1.

→ Now, we will design and write a program for a phased stepper motor having 200 teeth for 5 revolutions in clock wise and Anticlockwise.



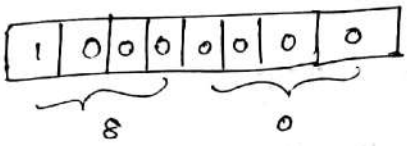
Sol :-

The counter value = No. of revolution × No. of teeth

= 5 × 200

= 1000 d.

Control word



Control word = 80 h.

```
Assume  CS : CODE12.
CODE12  SEGMENT.
START :  MOV AL, 80h
         OUT CWR, AL ; 8255 configured to make port A
                       an output port.
         Mov CX, 1000d ; The counter value is programmed
                         for 5 revolution.
         Mov AL, 88h
AGAIN :  OUT PORTA, AL
         CALL DELAY
         ROR AL, 01
         DEC CX ;  [CX] ← [CX] - 1.
         JNZ AGAIN1 ; Do the process till five CW
                      rotations are completed.
Mov AL, 88h
Mov CX, 1000d
AGAIN2 :  OUT PORT A, AL
          CALL DELAY
```

```
ROL  AL,01
DEC  CX
JNZ  AGAIN2
MOV  AH, 4Ch
INT  21h
CODE12  ENDS
    END  START
```

1/8/2013

1, Design an Interface consisting of 4x4 matrix keyboard with 8255.

1/2/2013

Interfacing with advanced devices :

Memory Interfacing to 8086, Interrupt structure of 8086. vector Interrupt lable, Interrupt Service routine, Introduction to DOS and BIOS Interrupts, Interfacing Interrupt controller 8259, DMA controller 8257 to 8086.

→ Design an Interface between 8086 CPU and 2 chips of 16K x8 EPROM and 8 chips of 32K x8 RAM. Select the starting address of EPROM suitably ~~00000h~~.

A, The last address in the map of 8086 is FFFFFh. After reseting the processor starts from FFFF0h. Hence, this address must lie in the range of EPROM. The address map for the problem is given below.

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	$A_{11} - A_8$	$A_7 - A_4$	$A_3 - A_0$
EPROM ending address FFFFFh	1	1	1	1	1	1	1	1	1111	1111	1111
EPROM starting address F8000h	1	1	1	1	1	0	0	0	0000	0000	0000
RAM ENDING Address 07FFFh	0	0	0	0	0	1	1	1	1111	1111	1111
RAM Starting address 00000h	0	0	0	0	0	0	0	0	0000	0000	0000

RAM Area

F8000h

RAM :-

Two chips of $32K \times 8$

$\Rightarrow 2 \times 32K \times 8$

$\Rightarrow 64K \times 8$

$\Rightarrow 2^6 \times 2^{10} \times 8$

$\Rightarrow 2^{16}$ bytes

$\Rightarrow$ 16 Address bits are required to address the RAM

So, we use $A_0 - A_{15}$ bits.

EPROM

2 chips of $16K \times 8$

$2 \times 16K \times 8$

$32K \times 8$

$2^5 \times 2^{10} \times 8$

2^{15} bytes.

$\Rightarrow$ 15 Address bits are required to address the EPROM

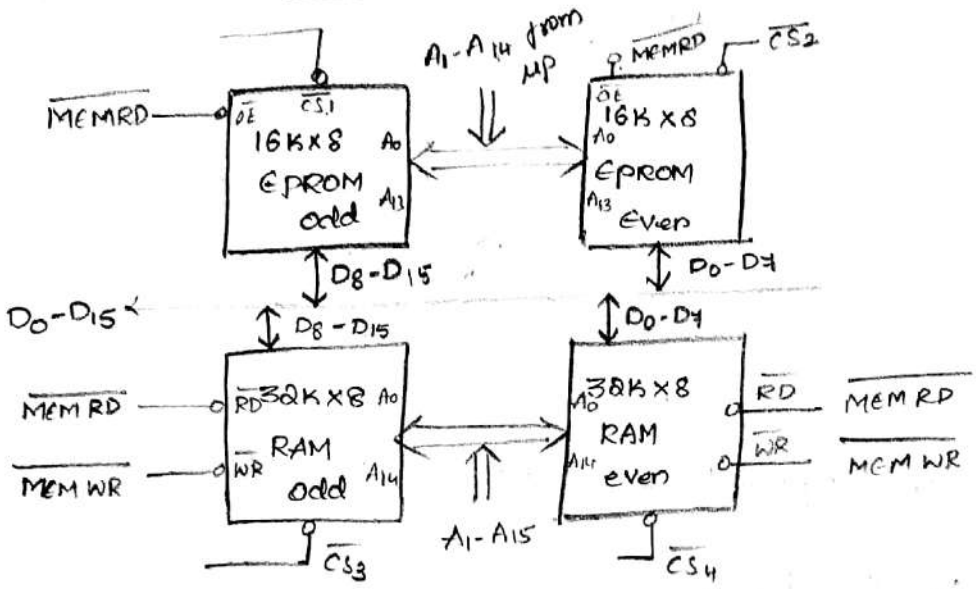
So, we use $A_0 - A_{14}$ bits.

→ The memory in 8086 is organised by odd bank (Higher byte) and even bank (lower byte). The signals A_0 and $\overline{BHE}$ are used to select these lines. As there are 2 EPROM chips of size $16K \times 8$ size, one chip is used for odd bank and another chip is used for even bank. Similarly, RAM is also arranged

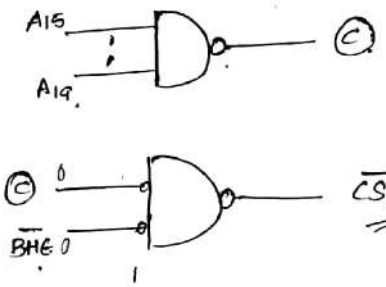
A_0	$\overline{BHE}$	
0	0	Word transfer on $D_0 - D_{15}$. Both even and odd banks are addressed.
0	1	Byte transfer on $D_0 - D_7$ and only even bank is addressed
1	0	Byte transfer on $D_8 - D_{15}$ and only odd bank is addressed

A_0 $\overline{BHE}$
1 1
- No memory operation

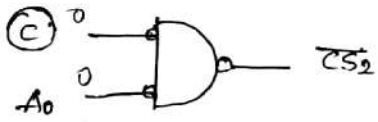
Hard ware design :-



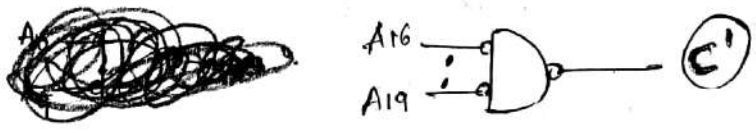
Signals for $\overline{CS1}$:-



Signals for $\overline{CS2}$:-



Signals for $\overline{CS3}$:-



Signal for $\overline{CS4}$:-

(2) Interface 2 4K x 8 EPROM and 2 4K x 8 RAM with 8086. Select suitable address lines.

RAM

$2 \times 4K \times 8$

$8K \times 8$

$2^3 \times 2^{10} \times 8$

2^{13} bytes.

13 Address bits are required to address the RAM.

So, we use $A_0 - A_{12}$ bits.

EPROM

$2 \times 4K \times 8$

$2^3 \times 2^{10} \times 8$

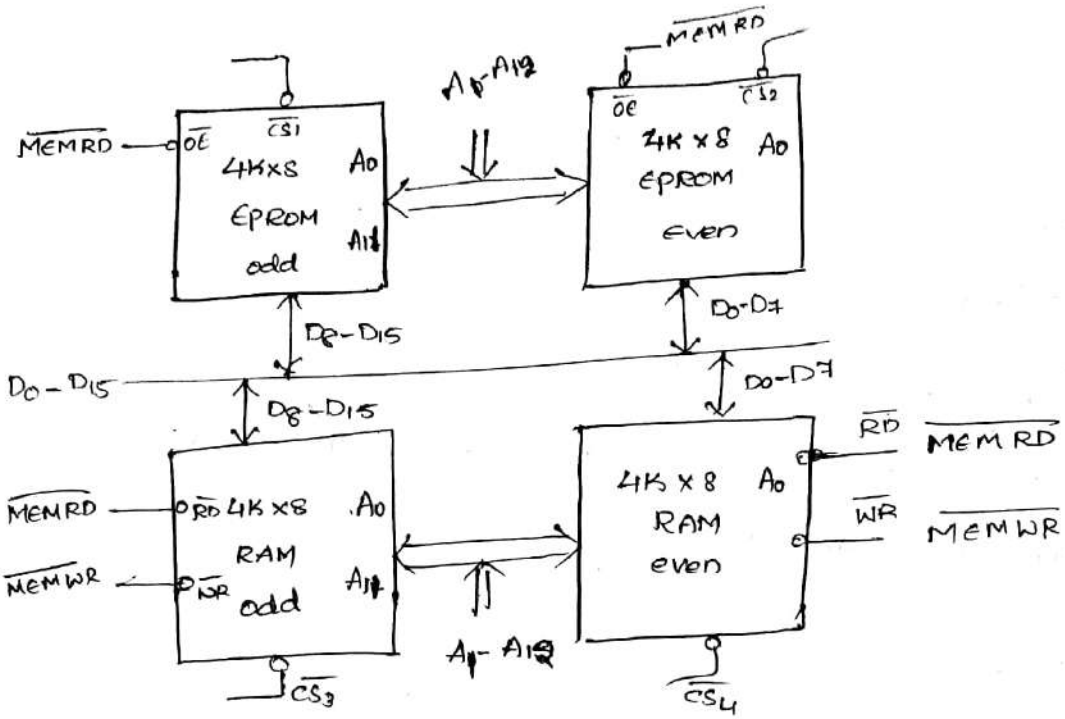
2^{13} bytes.

13 address bits are required to address the EPROM.

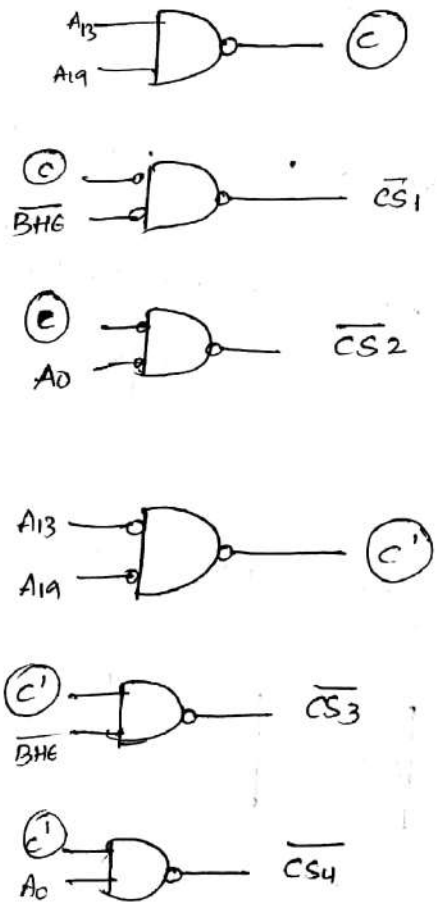
So, we use $A_0 - A_{12}$ bits

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	$A_{11} - A_8$	$A_7 - A_4$	$A_3 - A_0$
EPROM											
Ending address FFFFFh	1	1	1	1	1	1	1	1	1111	1111	1111
Starting address FE000h	1	1	1	1	1	1	1	0	0000	0000	0000
RAM											
Ending address 01FFFh	0	0	0	0	0	0	0	1	1111	1111	1111
Starting address 00000h	0	0	0	0	0	0	0	0	0000	0000	0000

Hardware design :-



Signals



1) Design an Interface consisting of 4×4 matrix keyboard with 8255.

A.

```
Assume CS: CODE
CODE SEGMENT.
PORT A  EQU 0000
PORT C  EQU 0004
CR      EQU 0006
PROC KEY NEAR.
START: MOV AL, 81H ; Initialize port CL as i/p and port CU as o/p
       MOV DX, CR ; [Initialise 8255]
       MOT DX, AL ;
       MOV AL, 00H
       MOV DX, PORTC
       OUT DX, AL ; Make all scan lines zero
BACK:  IN AL, DX
       AND AL, 0FH
       CMP AL, 0FH ; check for key release
       JNZ BACK ; If not, coait for key release.
BACK1: IN AL, DX
       AND AL, 0FH
       CMP AL, 0FH ; check for key release.
       JZ BACK1 ; If not, wait for key press.
       CALL DELAY ; wait for key debounce
       MOV BL, 00H ; Initialize key counter.
```

```
        Mov  CL, 04H
        Mov  BH, FEH ; make one column low.
NEXT COL: MOV  AL, BH
        OUT  DX, AL
        Mov  CH, 04H ; Initialize row counter.
        Mov  DX, PORT A
        IN AL, DX ; Read return line status.
NEXT ROW: RCR AL, 1 ; check for one row.
        JNC display ; If zero, go to display, Otherwise continue
        INC  BL  ; Increment key counter
        DEC  CH ; decrement row counter
        JNZ NEXT ROW ; check for next row.
        Mov AL, BH
        RCL AL, 1 ; Select the next column.
        Mov BH, AL.
        DEC CL ; decrement the next column.
        JNZ NEXT COL ; check for last column if not repeat.
        JMP start ; Go to start.
        RET
KEY     END P
        END START
```

4/2/2013

Interrupt Structure of 8086:

Whenever a no. of devices request service of the microprocessor, the µp has to handle such situations. for example, the µcomputer should be able to give response to devices like keyboard, sensors, other components when they request for service. there are two approaches to handle such situations.

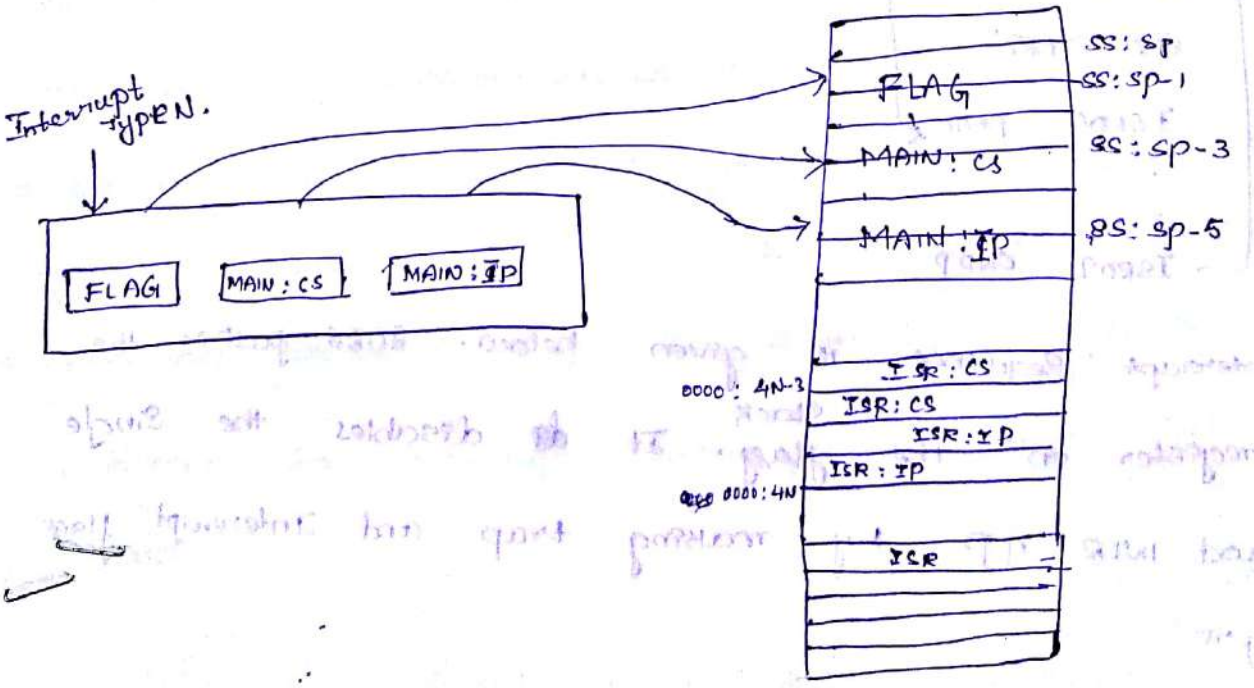
① pooled approach ② Interrupt approach.

In pooled approach, the µp periodically scan all the I/O devices for any service request and then provide service accordingly. This reduces the system speed and hence inefficien

→In the Interrupt method; the I/O deveces send signals on INTR or NMI pins. The µp receives those signals and provides the service.

In the case of 8085, there are 5 pins.

TRAP, INTR, RST 7.5, RST 6.5, RST 5.5 for external interrupts.



→ Whenever a µp is interrupted it stops the main program execution. Main program flag, CS, Ip are pushed to the Stacks.

→ The CS and Ip are loaded with Interrupt Service Routine address as shown above. When RET instruction is encountered in ISR, CS and Ip are loaded with main program data by pop operation. So, the main program is resumed appropriately.

6/2/2013

Transfer of control During ISR:-

```
ASSUME  CS: CODE.
CODE 8: SEGMENT.
START:  MOV  AL, 80H
        - - - - -
        - - - - - -
        INT 09          ; Interrupt has occurred.
        MOV BH, 90H
        - - - - - -
        - - - - - - -
CODE 8  ENDS
END START
ISR09   PROC
        - - - - - -
        - - - - - -
ISR09   ENDP
```

The Interrupt Sequence is given below. 8086 pushes the flag register on the stack. It disables the single step and INTR i/p by making trap and Interrupt flags to zero

It says a / used / put

↳ It saves the main program values (CS & IP) by pushing them into the stacks.

→ It does an indirect far jump to the start of ISR (Interrupt Service Routine) by loading new values of CS and IP of ISR

→ for ex. if Interrupt type is 4, the memory address is 4×4 = 16_{10} = 10h.

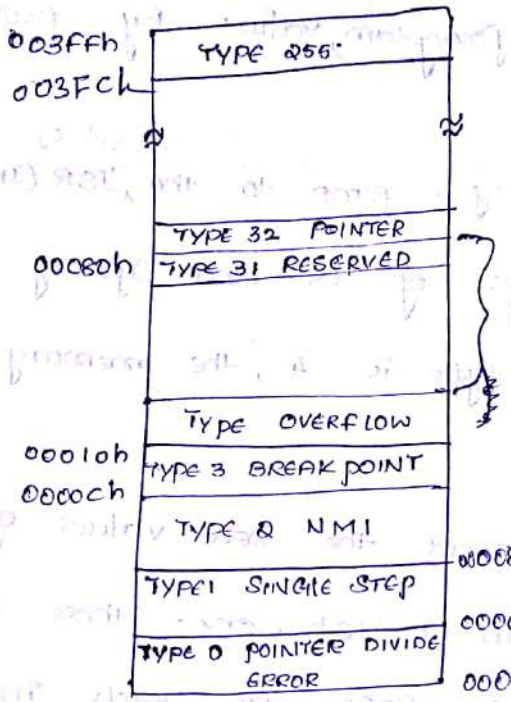
→ In 8086, we will read the new values of IP from 00010h and therefore CS from 00012h. once these values are loaded in CS and IP registers, 8086 will fetch instructions for the new address to execute ISR.

→ When IRET / ENDP instruction is encountered, the main program values are popped from the stacks and the main program is resumed.

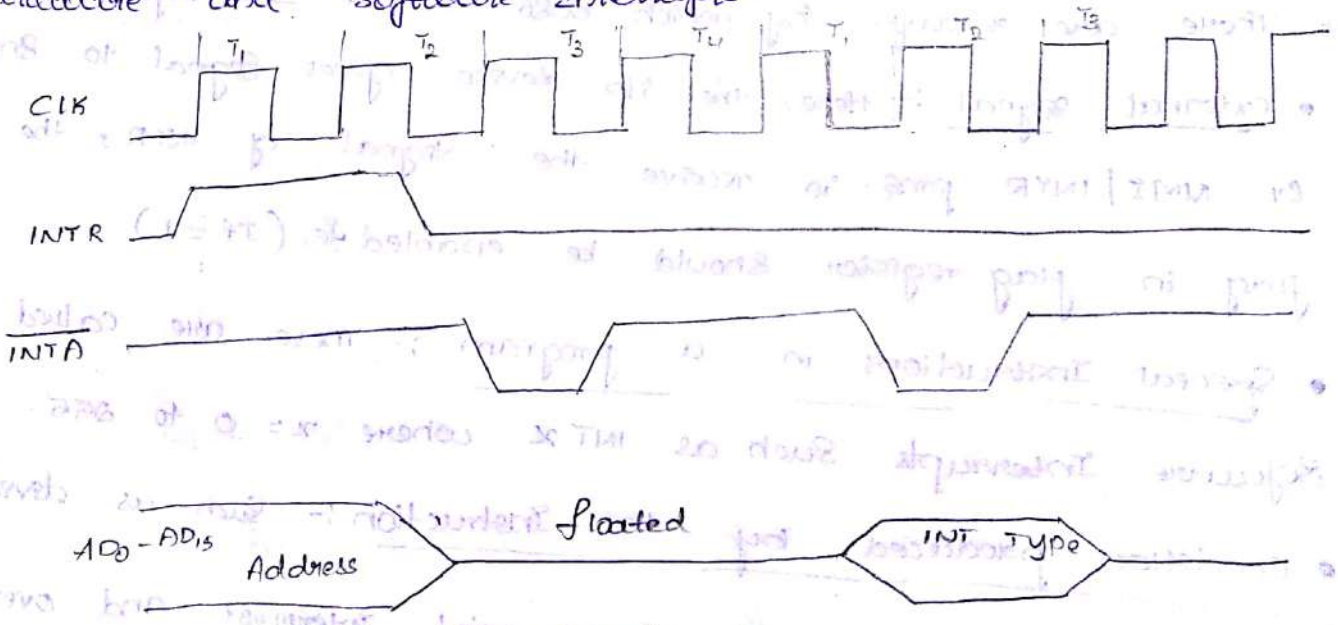
→ There are 3 ways by which 8086 can be Interrupted:

- External signal :- Here, the i/p device gives signal to 8086 on NMI / INTR pins. To receive the signal of INTR, the IF flag in flag register should be enabled ie, (IF = 1)
- Special Instructions in a program :- These are called Software Interrupts such as INT x where x = 0 to 255.
- Condition produced by the Instruction :- Such as divide by 0, Single Step Interrupt, Break point Interrupt and overflow interrupt.

8086 Interrupt Vector table :-



In 8086, the first 5 types have explicit definitions such as divide by 0, overflow etc. the next 27 interrupt type ie, type 5 to 31 are reserved for future use. the upper 224 interrupt types from type 32 to 255 are available for the user for hardware and software interrupts.



Interrupt Acknowledge Machine cycle

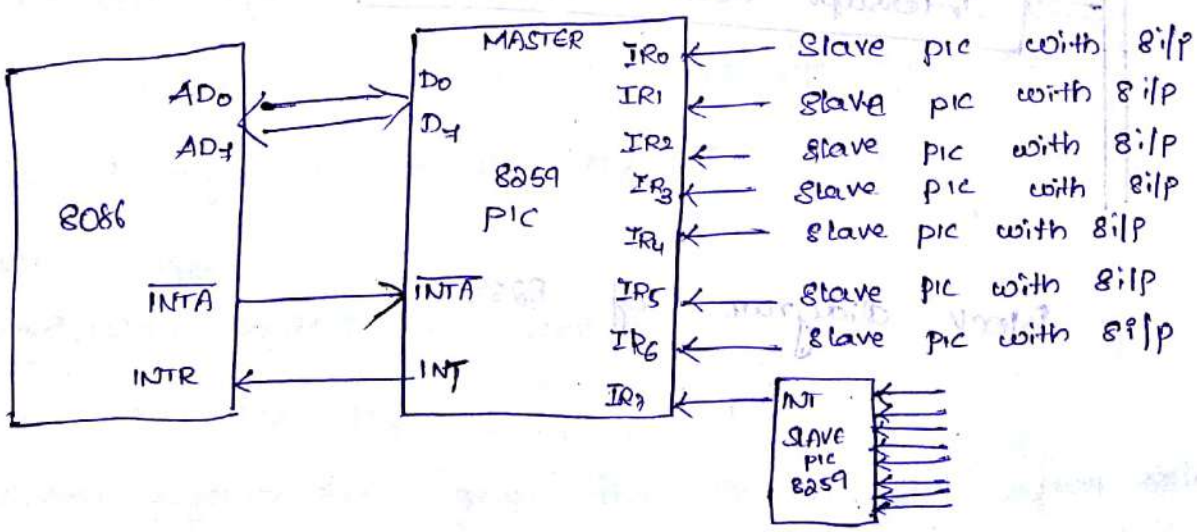
When i/p wires requires the service of the µp it sends a signal an INTR time to µp in response to this the µp gives a INTA loco (Interrupt Acknowledge) pulses as shown above. During first INTA the bus is tristated, in the second INTA pulse, the i/p device puts INTA type. The INT type information is used by µp to get new values of ISR.

The features of 8259 :-

→ It has 8 Interrupt lines so it can manage upto 8 interrupts This is equivalent to providing 8-interrupt pins on the processor instead of single INTR pin.

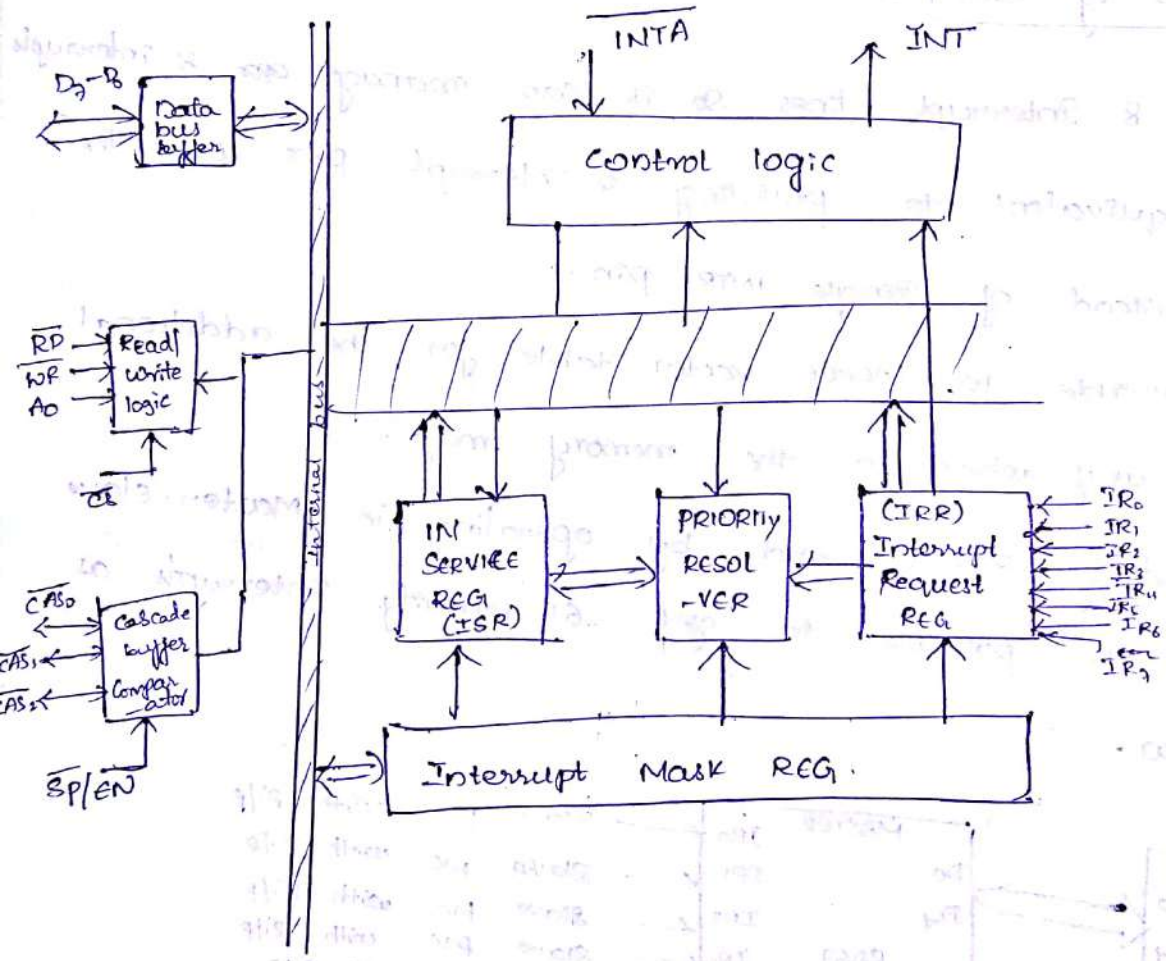
→ It is possible to locate vector table for the additional interrupts any where in the memory map.

→ By cascading 8259's and by operating in Master-slave mode, it is possible to get 64 priority interrupts as shown below.



↳ 8259 PIC has internal mask register by which the individual lines $IR_0 - IR_7$ can be enabled or disabled. 8259 PIC can be programmed to accept either level trigger or edge trigger inputs.

→ With the help of 8259, we can get information of pending interrupts, inservice Interrupts, masked Interrupts.



Block diagram of 8259

8/8/2013 8259 PIC.

(a) Data bus Buffer

(b) Read/write logic

(c) control logic

(d) IRR, ISR, IMR

(e) Priority Interrupt Resolver.

(f) Cascade buffer configuration

(g) $\overline{SP}/EN$ +Vcc MASTER

GND SLAVE.

Data bus buffer :- The cpu sends command word to PIC 8259 and obtains its status the interrupt type (8 bit data) is transferred to up by PIC using the data bus transfer. This center

Read/write logic :- This controls the direction of data flow on data bus buffer.

control Logic :- This gives the signal INT after selection of any i/p line (IR_0 to IR_{07}) by priority Interrupt Resolver. Also it receives $\overline{INTA}$ pulse from cpu.

IRR, ISR, IMR :- The IRR is used to store all the interrupt status of 8 input devices. It is 8 bit register. The Interrupt Mask register IMR enables or disables a particular I/p device. Bit 1 is used for disabling / Masking and Bit 0 is used for enabling. The Interrupt Service Register stores all the interrupts that are being processed.

Priority Interrupt Resolver :- It gets data from IRR, ISR, IMR and selects a particular input device for Interrupt processing The master uses this block for communication with slave.

f) $\overline{CAS_0}$, $\overline{CAS_1}$, $\overline{CAS_2}$ are the slave Identification bits. TO identify any one of the slaves by the master.

g) $\overline{SP}$ / EN :- For the master slave pic 8259, the pin $\overline{SP}$/EN is connected to V_{cc}. for the same pic, this pin is connected to ground.

Introduction to DOS and BIOS Interrupts :-

```
Mov AH, 4Ch    ; function code
INT 21h        ; } this is DOS Interrupt.
```

DOS Interrupts :-

Interrupts	function code	operation
INT 21h	4Ch	Terminate the programme with appearance of command prompt on the screen
INT 21h	01h	Read a character from standard input device.
INT 21h	05h	write a character to standard o/p device
INT 21h	3Dh	open file
INT 21h	3Eh	close a file

INT 21h	41h	delete a file.

BIOS Interrupts :-

INT 10h	00h	Set Video mode.
INT 10h	01h	Set cursor shape.
INT 10h	02h	Set cursor position.
INT 10h	03h	Read cursor position.
INT 10h	04h	Read light pen position.

In IBM pc, part of the operating system was located in permanent memory location (EPROM) which is referred to as BIOS (Basic I/P O/P System). This is located at the top of memory in the address range FE000h to FFFFFh or 8086 based processor. The BIOS programs provide direct and low level interactions with various devices in the system. The BIOS has programs for

(a) power on self test.

(b) Time of the day.

(c) Print screen

(d) IO Support program for asynchronous communication, keyboard, printer and display.

The DOS is stored in Harddisk. The services provided by DOS include appearance of command prompt on the screen after switch ON, file management (Create, Read, write, delete files), memory management, directory management and utility programs.

11/2/2013

DMA 8257 to 8086

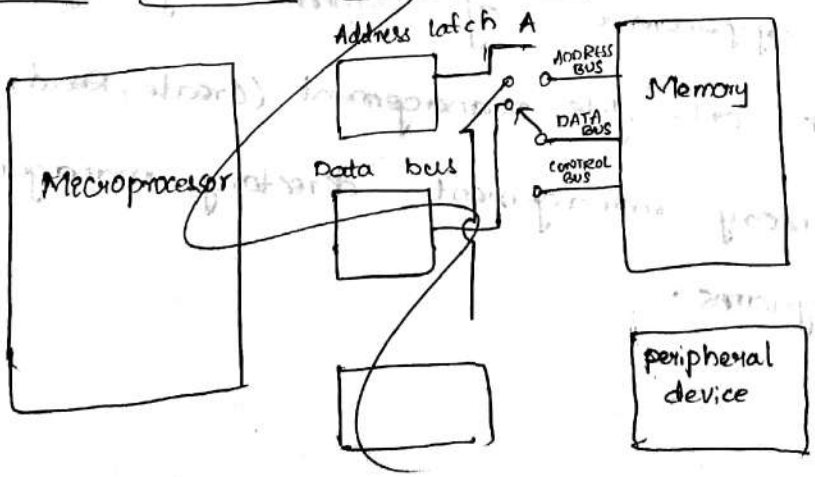
Software controlled Data transfer.

```
    MOV CX, COUNT
    MOV DX, PORT addr
BACK: MOV AL, [SI]
      OUT DX, AL
      INC DX
      INC SI
      loop BACK ;
      RET
```

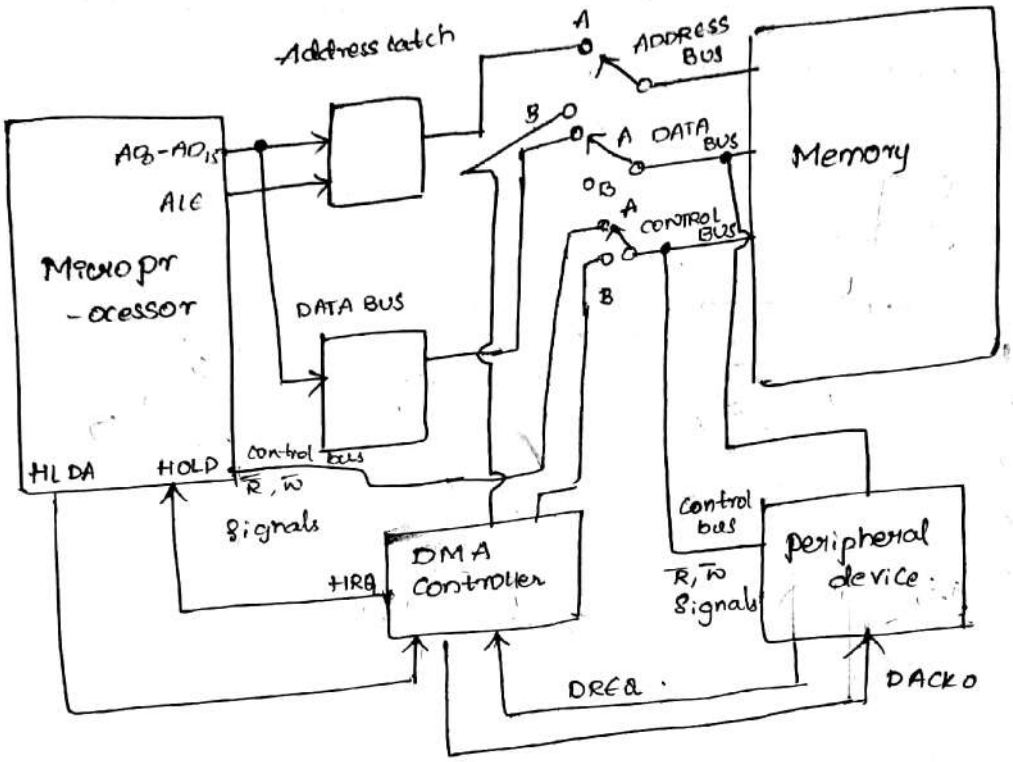
In above program, the data is transferred from memory location to I/O device. First the contents of memory location are brought inside µp ie to AL register and then to the I/O device pointed by DX.

The no. of bytes to be transferred or initialised in CX register. The above scheme is slow and suitable only for small no. of given data. for large volumes of data from disk to memory or from memory to disk, we use DMA method.

Hardware controlled Data transfer :-



Hardware controlled Data transfer :-



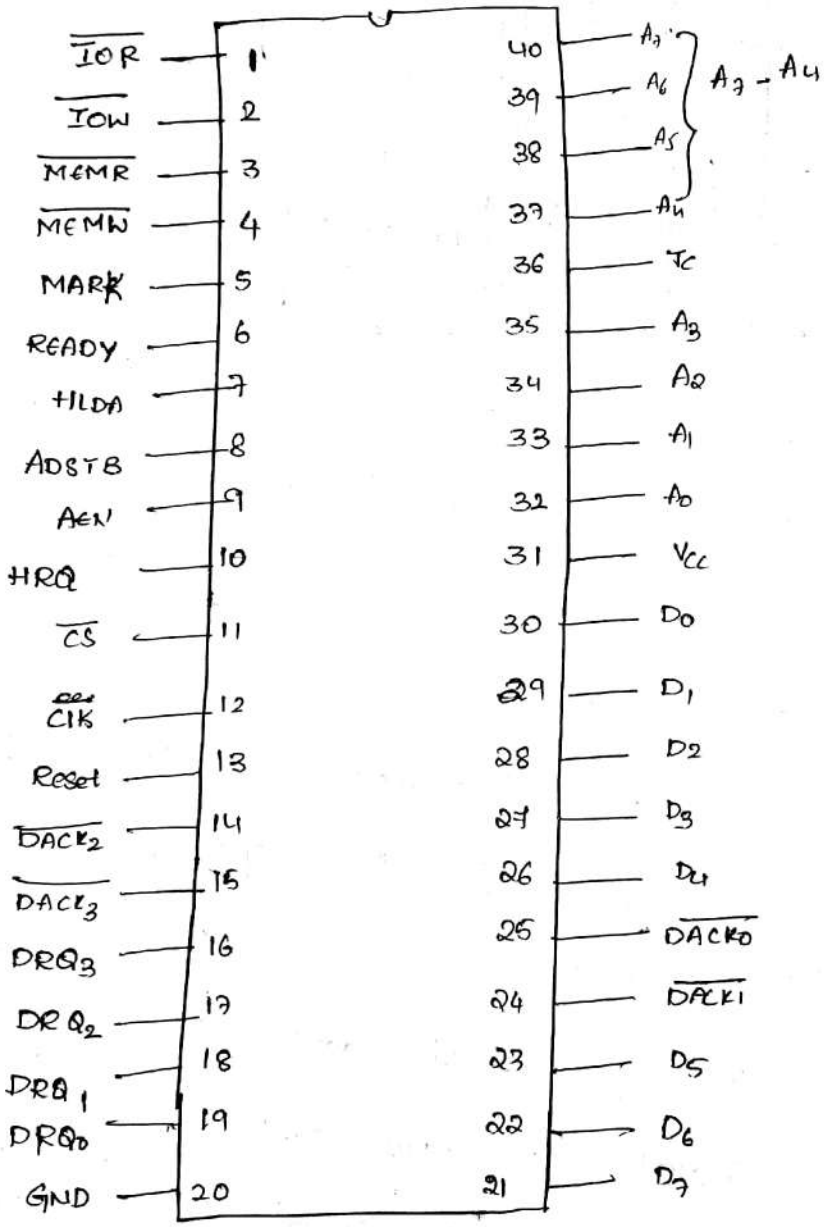
The DMA controller has two types of operations i) Slave ii) master mode

In slave mode, the cpu is in control of system bus and all switch positions are in A. This slave mode is used to configure the DMA controller and to read the status of DMA controller. During Actual progress of DMA operation, the DMA controller is in master mode.

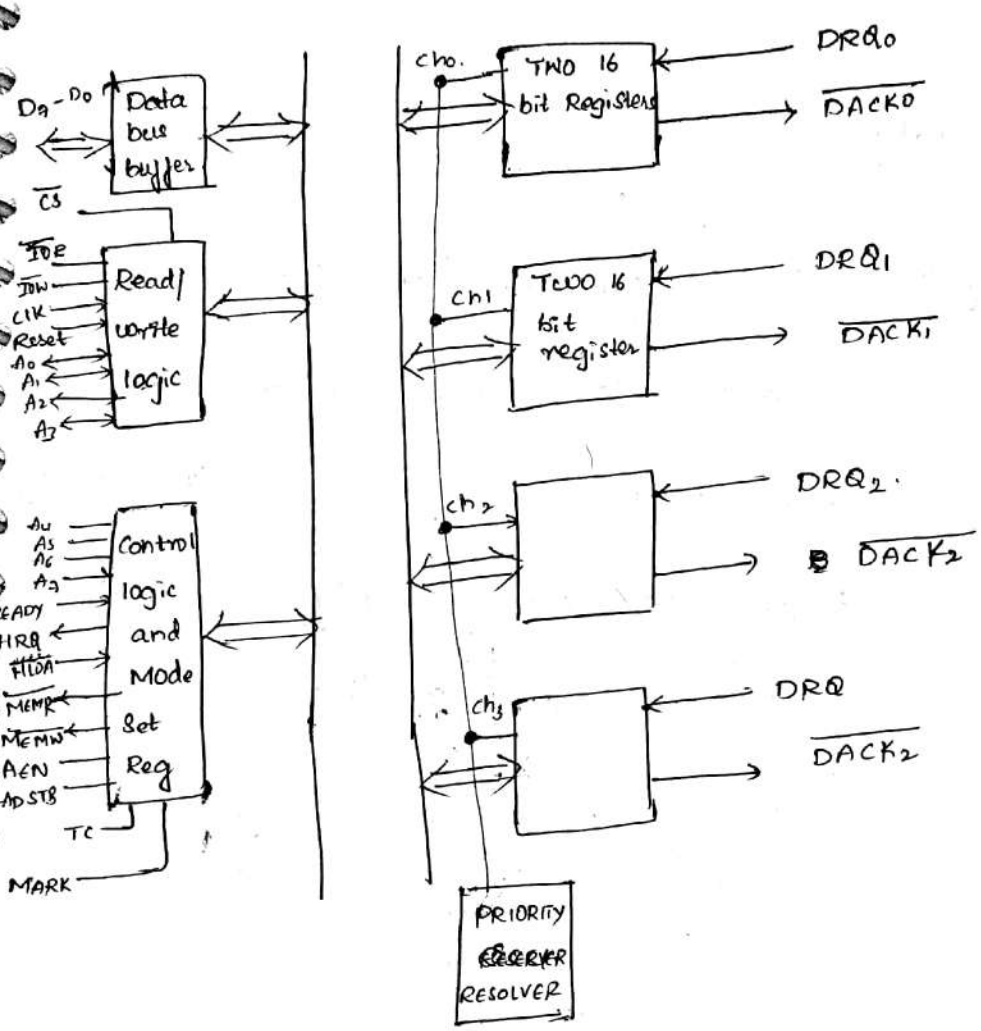
When the peripheral device is ready for DMA, it sends DREQ signal to DMA controller. Then, the DMA controller sends HOLD Request HRQ signal to up. The when the up gives HLDA signal, the switch positions are changed to B and up is completely isolated from System bus. The control signals $\overline{IOR}$, $\overline{MEMW}$ are used to copy the data from disk to memory and the signals $\overline{IOW}$, $\overline{MEMR}$ are used to transfer the

data from memory to disk. At the end of DMA, the HRQ signal is deactivated by the DMA controller and the switch positions are changed from B to A.

Pin configuration of 8257 :



Functional Block diagram of 8257 :-



The Salient features of 8257 :-

(1) This is a four channel programmable DMA controller.

(2) each channel has two 16 bit register. The DMA Address register stores the starting address of memory location for 14 bits of pc register (16 bit terminal count register) are used to store count value
Maximum 2^{14} bytes can be transferred ie, 16Kb.

(3) This has the priority resolver block which can be programmed to work in 2 modes (i) fixed priority & Rotating priority.

d) it has inhibit logic to deselect any one of the 4 channel

→ This is DTL compatible and works with +5V.

18/2/2013

Communication Interface : Serial communication standards.

Serial data transfer schemes:

8251 USART architecture and Interfacing

RS232, IEEE-488, Prototyping and

Trouble shooting.

There are 3 types of communications.

(i) Simplex : This is unidirectional.

Ex:- computer to printer, FM radio receiver.

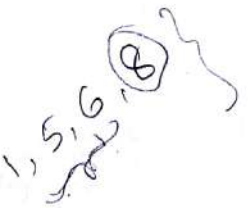
(ii) Half Duplex :- Here, the communication is both direction but simultaneous transmission and reception is not possible.

Ex:- Walkie-Talkie

(iii) Full Duplex :- Here the communication is bidirectional as well as simultaneous.

Ex:- cell phone.

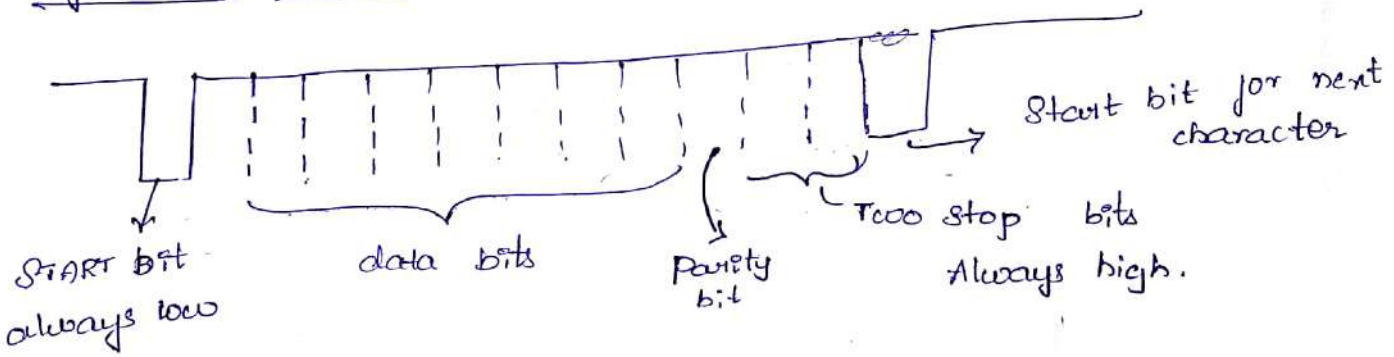
When distance involved is small, we can go for parallel communication which is faster. For long distances, Serial communication is preferred as we can save material cost for wires but the disadvantage is slow.

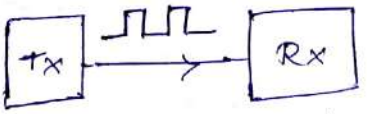


Serial communication

- Asynchronous Serial communication
- Synchronous Serial communication

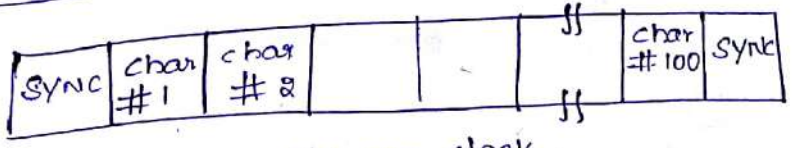
Asynchronous Mode

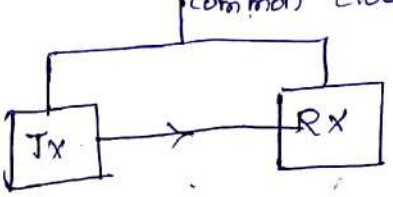




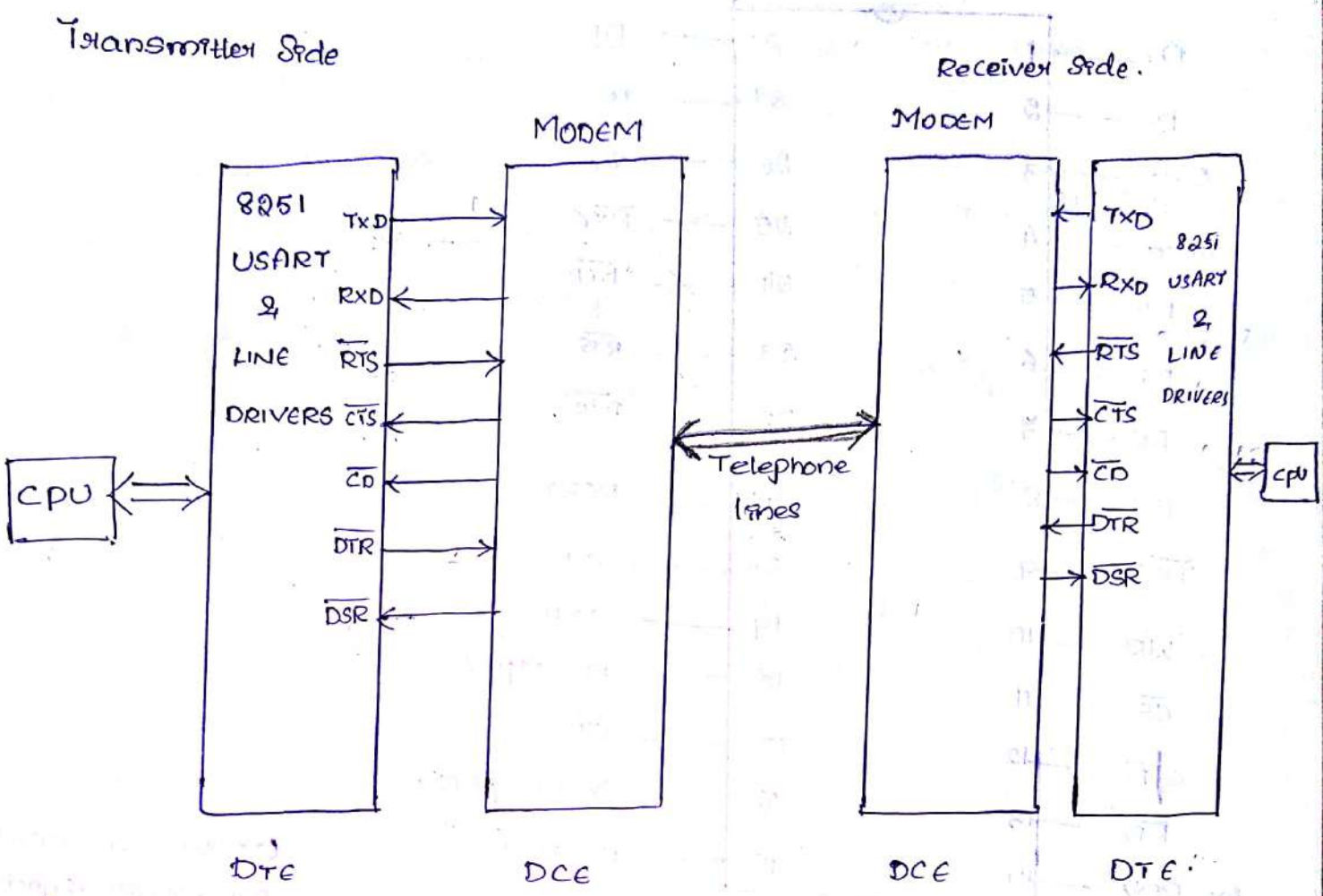
Serial data with LSB first.

Synchronous Mode :-

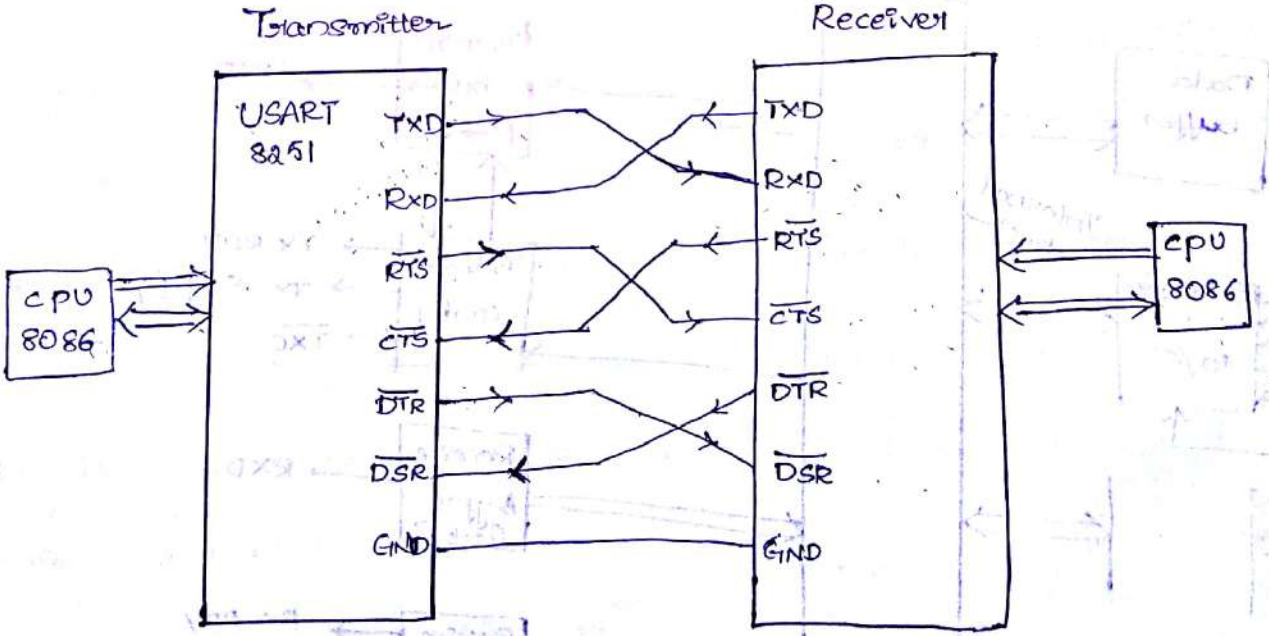




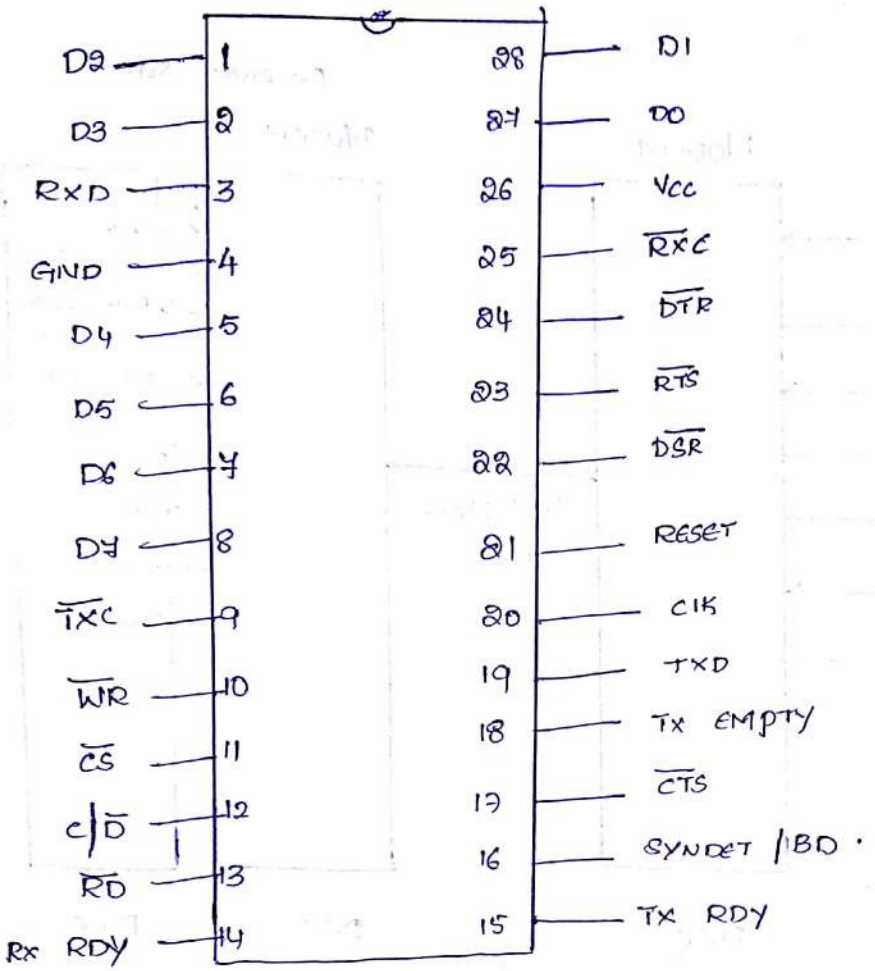
21/2/2013



→ Non Modem connection for Serial communication :-

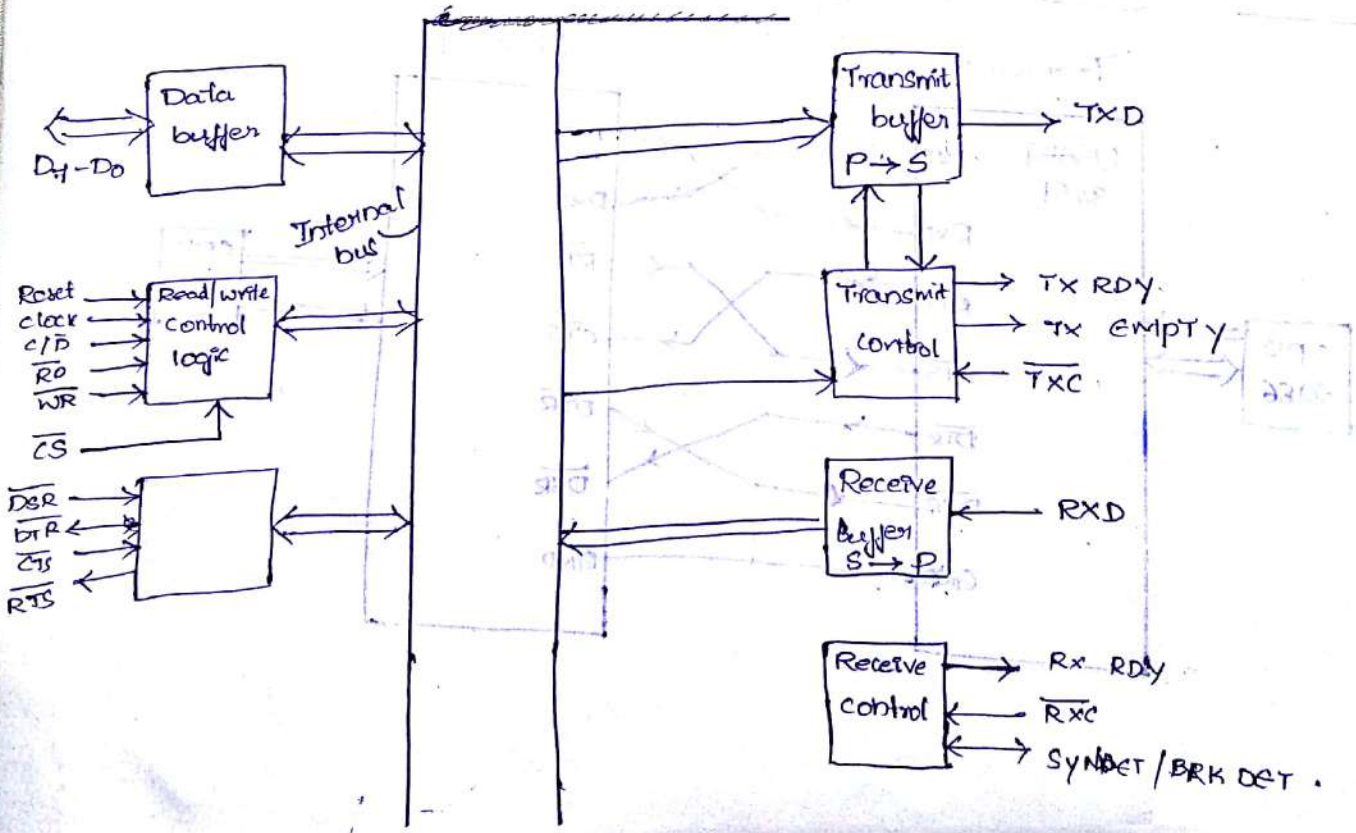


Pin configuration of 8251.



USART - Universal Synchronous Asynchronous receiver transmitter.

Functional block diagram of 8251 USART.



There are 8 data lines in 8251 and these are bidirectional. Data is transmitted or received by the cpu on these lines.

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	
0	1	1	0	8086 writes to the command Register using data bus ($D_0 - D_7$)
0	1	0	1	8086 reads the status of USART status register
0	0	1	0	8086 writes the byte data for transmission
0	0	0	1	8086 reads the byte data after reception and Assembly by USART.
1	X	X	X	USART is not selected for any operation.

Brief functional description of 8251 USART.

→ There are 8 parallel lines $D_7 - D_0$ which are connected to a system data bus, so that control or status words can be transferred between cpu and USART.

→ The chip select of USART is connected to address decoder

→ 8251 has 2 addresses as per $C/\overline{D}$ signal, when $C/\overline{D} = 1$ it is a control word address. When $C/\overline{D} = 0$, it is a data addresses.

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	
0	1	1	0	8086 writes to the command
0	1	0	1	8086 (reads) the status of Reads USART

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
0	0	1	0	8086 writes the byte data
0	0	0	1	8086 µp reads the byte data.
1	0	0	0	chip is not selected.

The clock i/p of 8251 is connected to a system clock for synchronisation.

→ Modem :- When 8251 is switched 'ON' it sends a $\overline{DTR}$ (Data terminal ready) signal to Tx modem, the Tx modem sends $\overline{DSR}$ (Data set ready) signal back to USART.
When 8251 is ready for serial transmission, it sends $\overline{RTS}$ (Request to send) signal to transmit Tx modem. The transmitting modem gives green signal for transmission by activating $\overline{CTS}$ (clear to send) signal.

→ The shift registers in transmit and receive buffer blocks require clocks for serial data-in and serial data-out.

$\overline{TxC}$ and $\overline{RxC}$ clocks are used for this purpose. Usually both are shorted and connected to a common source.

→ 8251 is double buffered. The transmitter section has holding buffer, shift registers ie, 2 buffers. When one character is loaded to holding buffer, another character can be moved out of the actual transmit shift register.
The TX RDY (Transmitter Ready) goes high, when holding buffer is empty. This is an indication to the CPU that next byte can be return to the holding buffer.

→ The TxRDY is connected to INTR of cpu.

→ The RxRDY (Receive Ready) goes high, when a character has been assembled in the receiver buffer which can be readily read by the cpu.

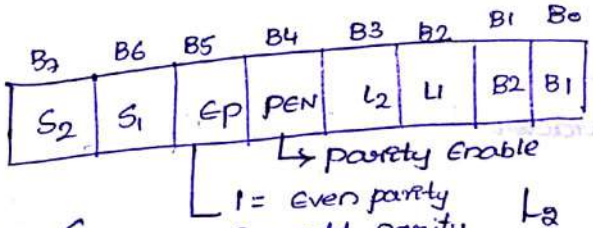
→ The TXEMPTY pin goes high (active) when both buffers in the transmitter section become empty.

→ SYNC-Detector / Break detector :- When USART is operating in Synchronous mode if RxD is low for 2 character times, this pin goes high, This is an indication for of break in communication.

for Synchronous communication, when 8251 finds a specified sync character in the incoming string of data. Then, this pin goes high.

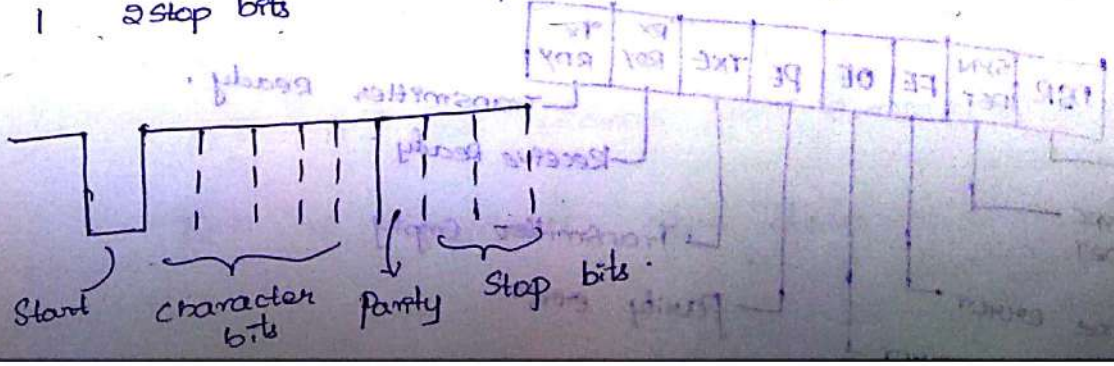
22/1/2013

8251 USART MODE WORD Register FORMAT.



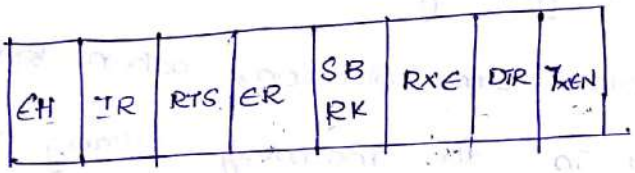
S_2	S_1	
0	0	Invalid
0	1	1 Stop bit
1	0	1½ Stop bit
1	1	2 Stop bits

L_2	L_1	
0	0	5 bits per character
0	1	6 bits per character
1	0	7 bits per character
1	1	8 bits per character



B_2	B1	
0	0	Synchronous Mode & Transmission / Reception at $\overline{TxC}$, $\overline{RxC}$
0	1	$\overline{TxC}$ // Asynchronous
1	0	$\overline{TxC}$ /16 Asynchronous
1	1	$\overline{TxC}$ /64 Asynchronous.

Command word Register format :-



EH = Enter Hunt Mode.

= Enable Search for sync character.

IR = Internal Reset.

RTS = Request to Send

ER = Error Reset

SBRK = Send Breaks character.

RxE = Receive Enable

DTR = Data Terminal Ready.

TxEN = Transmit ENable.

Trans Enable.

0000 0001

0 1 h

RR Enable

0000 0010

0 2h

Status word Register Format :-

Transmitter Ready.

Receive Ready.

Transmitter Empty.

Parity error

Frame error

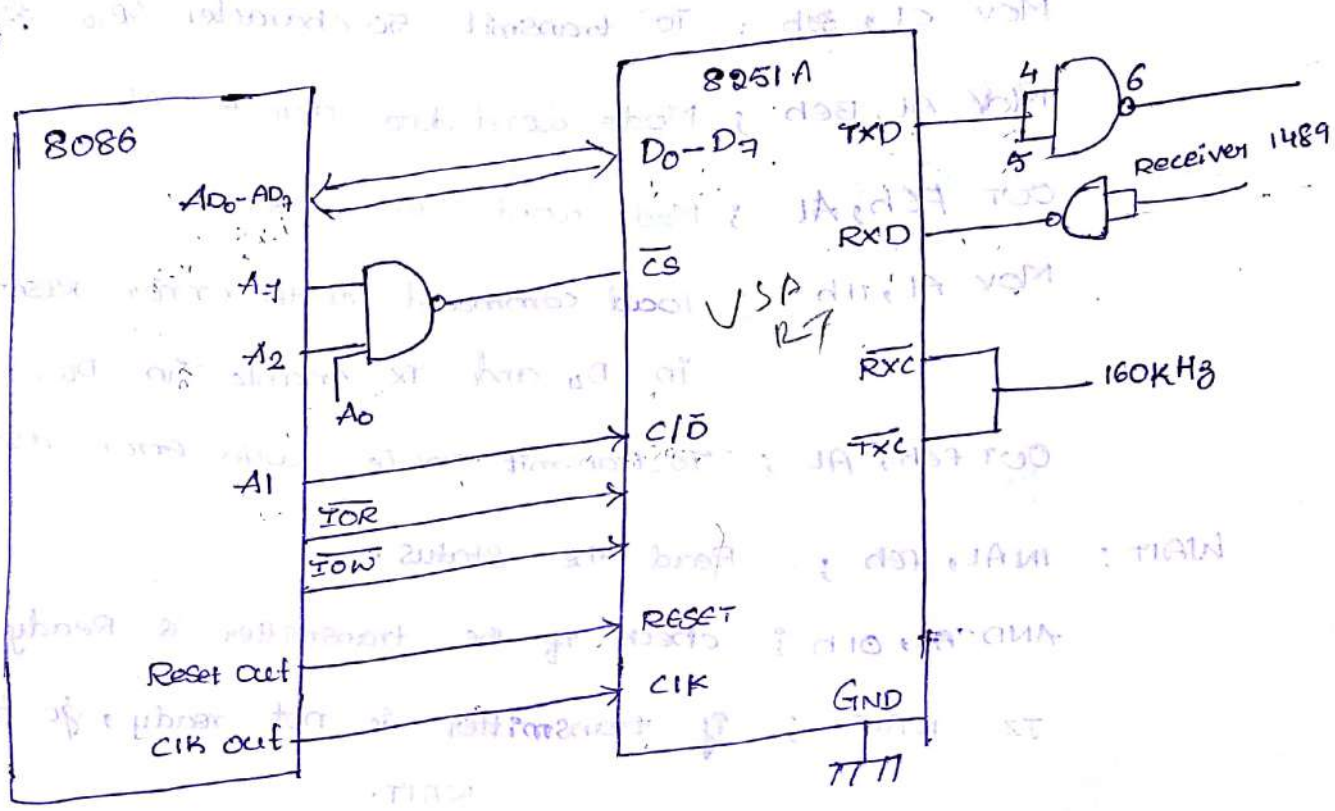
Over Run error

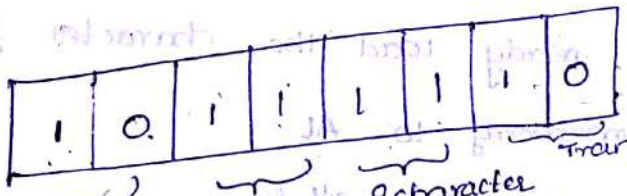
SYNC DET

ata set eady

Write an ALP in 8086 for transmitting 50 characters which are stored in the Memory location 02010h using 8251. The design may be Implemented with following conditions. (Starting from)

a) even parity Enable b) 1½ Stop bits c) 8 bit character length
d) Transmitter clock frequency 160KHz e) baudd rate 10KHz. transmitter clock freq.





is scaled ie, $\frac{160KHz}{10KHz} = 16$.

So, mode word is BEh

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	1	1	1	1	0
F				E			

Transmitter bit Enable

FEh is the address for command

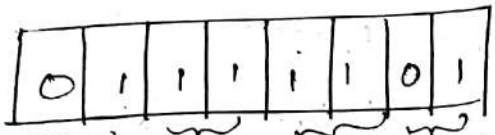
FCh is address for data

```
ASSUME CS : CODE5.

CODE 5 SEGMENT

START : MOV AX, 0200h
        MOV DS, AX
        MOV SI, 0010h
        MOV CL, 32h ; To transmit 50 character 50₁₀ = 32h
        MOV AL, BEh ; Mode word data BEH to AL
        OUT FEh, AL ; Mode word to USART.
        MOV AL, 11h ; Load command with error Reset.
                      in D4 and Tx Enable in D0.
        OUT FEh, AL ; To transmit mode with error reset.

WAIT :  IN AL, FEh ; Read the status.
        AND AL, 01h ; check if the transmitter is Ready.
        JZ WAIT ; if transmitter is not ready, go to
                  WAIT.

        MOV AL, [SI] ; if ready load the character from
                       memory to AL.
        OUT FCh, AL ; Data transmitted.
        INC SI ; point to next byte.
        DEC CL ; Decrement counter
        JNZ WAIT ; Repeat them for 50 characters.

        INT 3h ;
CODE 5 ENDS

        END START ;
```

→ Write an ALP to receive 100 bytes of data stream and storing at the memory location 3000 : 4000h using 8251 USART the format of data is given below.

(i) Even parity enable (ii) 1 Stop bit (iii) 8 bit character (iv) frequency 160KHz (v) Baud rate 160KHz.

26/2/2013



Transmitter is scaled 160KHz / 160KHz = 1

Modeword = 7Dh.

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	1	0	1	0	0

↑ Error Reset

↳ Receiver enable

} CMD

= 14h.

3000 : 4000

30000
4000
34000

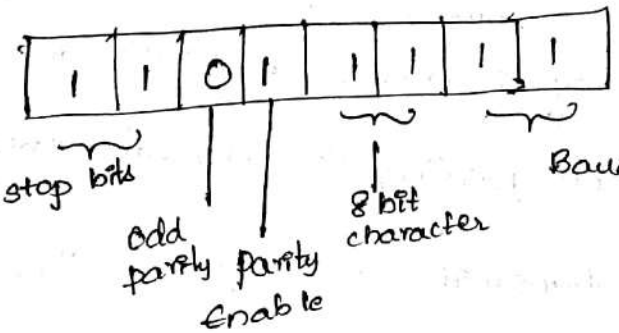
physical address

```
ASSUME CS : CODE 6.
CODE 6 SEGMENT
START : MOV AX, 3000h.
        MOV DS, AX ; the base address is loaded into segment register
        MOV SI, 4000h ; The offset address is loaded into SI. So,
                        we are pointing to physical address 34000h.
        MOV CL, 64h ; To transmit 100 bytes. (100_10 = 64h.)
        MOV AL, 7Dh ; Mode word data 7Dh to AL.
        MOV FCh
        OUT FCh, AL ; Mode word to
        MOV AL, 14h ; load command with error Reset
                      and to enable the receiver.
```

```
OUT FEh , AL ;  USART is Initialized with Command Word.
                word.
READY: IN AL, FEh ;  Read the status.
WAIT
       AND AL, 02h ;  check if receiver is ready.
       JZ READY.   ;  WAIT till the receiver is ready.
       IN AL, fCh  ;  The received character is transferred to AL.
       Mov [SI], AL ;  The received character is transferred to
                       memory.
       INC SI
       DEC CL
       JNZ READY ;  Repeat the loop for 100 bytes.
       Mov AH, 4Ch ;
       INT 21h ;
CODE G ENDS
    END START
```

→ Write an Initialization sequence to operate 8251 in Asynchronous mode with 8 bit character, size moderate with baud rate 64, 2 stop bits and odd parity enable. The 8251 is interfaced with 8086 at address 082h.

A,



Mode word = CF h

Mov AL, DFh

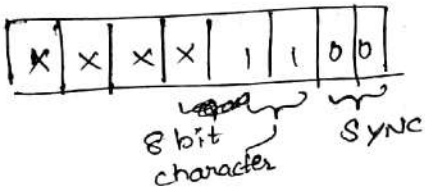
OUT 82h, AL

Mov AL, 40h // Do internal Reset for USART.

OUT 82h, AL

→ write instruction sequence to initialize 8251 in Synchronous mode with even / par Single character and 8 bit size.

A)



Assume XXXX as 0000.

So mode word is 0ch

Mov AL, 0ch

OUT 82h, AL

Mov AL, 40h // Do Internal Reset for USART.

OUT 82h, AL

28/2/2013 RS 232

RS 232 C

logic 1 -3V to -12V underload

-25V No load

logic 0 +3V to +12V underload

+25V No load.

* Give an overview of RS 232 C Serial data standard. Draw the Interface circuit for data conversion for TTL to RS 232C, RS 232C to TTL.

A, TTL is Transistor Transistor logic where bit 1 is represented in the range 2.5V to 5V and bit 0 is represented in the range less than 0.8V. The voltage levels for RS 232C is given below.

logic 1 -3V to -12V underload

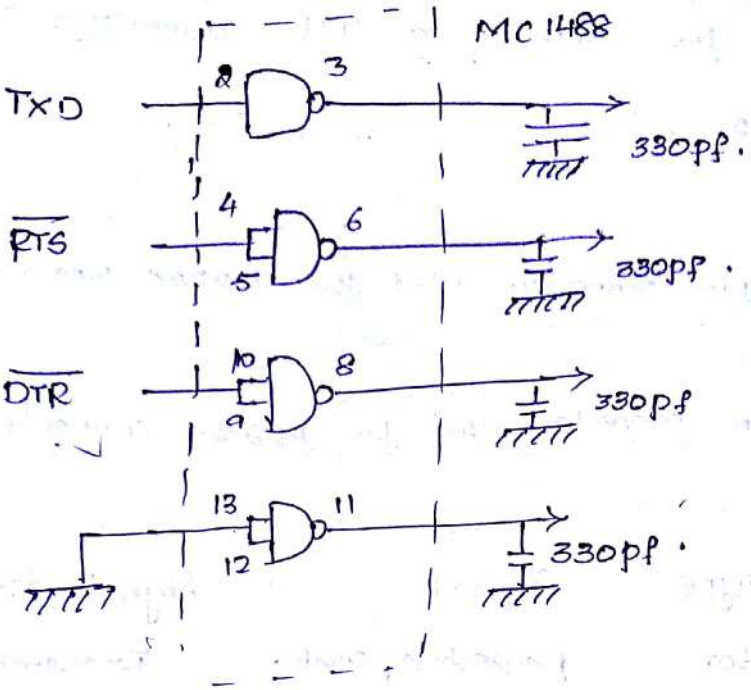
-25V No load.

logic 0 +3V to +12V underload

+25V No load.

RS 232C provides better Noise immunity compared to TTL. Voltages such as ±12V are permanently used in which case logic 1 is -12V and logic 0 is +12V

MC 1488 is a four quad · TTL RS232 converter whose configuration is given below.



MC 1488 for converting TTL to RS232C.

MC 1488 for

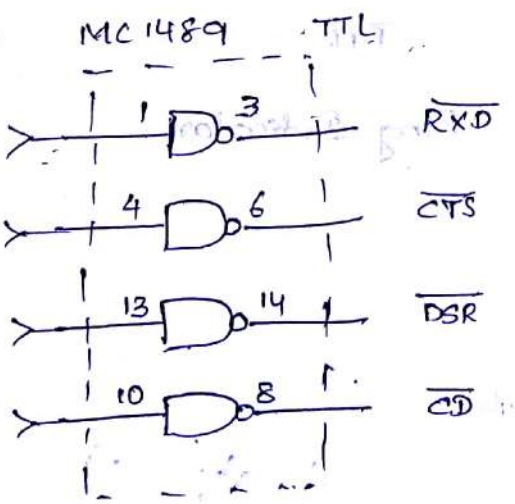
Pin 14 = +12V

Pin 1 = -12V

Pin 7 = GND.

→ for RS 232 to TTL conversion consists of Integrator circuit MC 1488 as shown above.

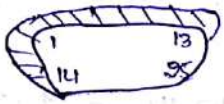
The standard Interface for TTL to RS232 conversion consists of IC's MC 1488 as shown above. This requires +12V, -12V power supply capacitor of 330pf is used to reduce the crosstalk between adjacent wires.



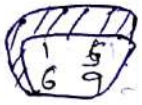
Pin 14 = +5V

7 = GND.

Ic MC 1489 is used for RS232 to TTL conversion and this requires only single 5V.



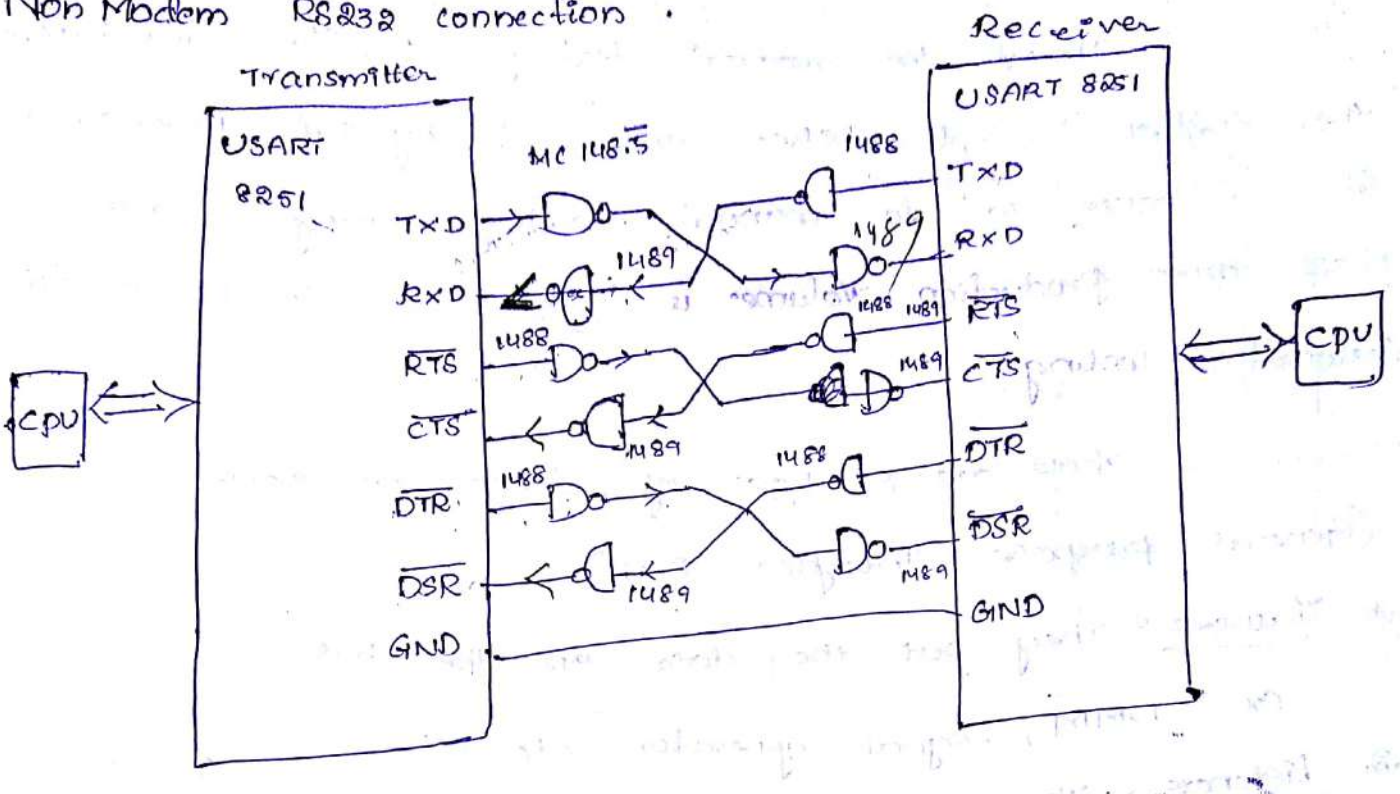
25 pin connector used for RS232C connection.



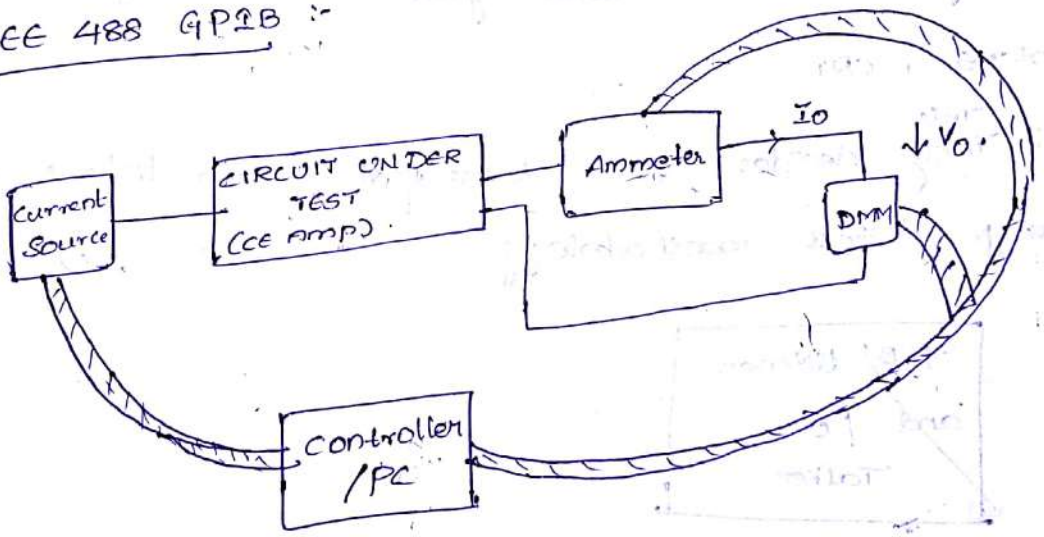
9 pin connector used for RS232C connection.

9 pin DType Connector	25 pin DType connector	Signal protective comb.	Signal direction in Transmitter
-	1	protective comb	—
3	2	TXD	OUT.
2	3	RXD	IN
7	4	$\overline{RTS}$	OUT
8	5	$\overline{CTS}$	IN
6	6	$\overline{DSR}$	IN
5	7	GND	
1	8	$\overline{CD}$ (carrier detect)	IN
-	12, 13, 14, 15, 16 17, 18, 19	Secondary channel Signals	
4	20	$\overline{DTR}$	OUT.
9	22	Ring Indicator	IN

Non Modem RS232 connection.



IEEE 488 GPIB :-



Hewlett packard (HP) has covered with the standard bus for the laboratory test equipment such as DMM's, Signal generator, freq counters, power meters etc., to facilitate Automatic test. This was adopted by IEEE in 1975. This was later revised in 1978, it is called IEEE 488 (GPIB this & the Automatic testing has 2 advantages change compared to manual testing.

The testing is faster and Accurate in ATE.

Though the manual testing is slow and inaccurate, the circuit concepts are better understood by this process. So, it is better to do manual testing during development phase when production Volume is high, we switch over to Automatic testing.

There are 3 types of devices on GPIB (General purpose Interface Bus)

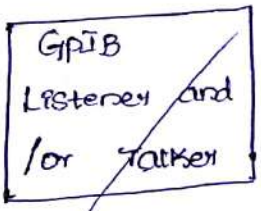
(1) Talkers :- They put the data on the bus.

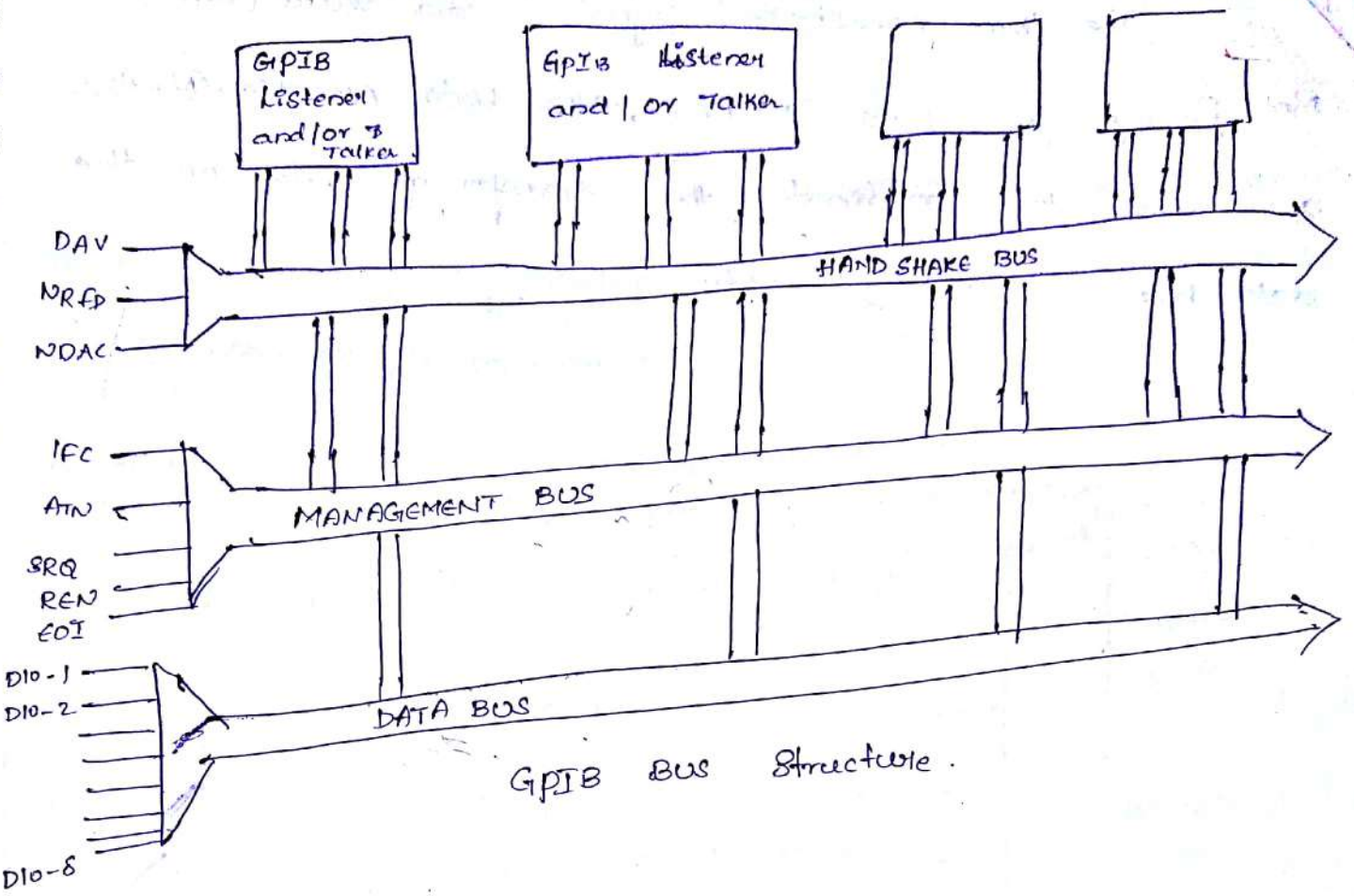
Ex. DMM, Signal generator etc.,

(2) Listeners :- They take the data from the bus for display

Ex:- printers, CRT

(3) Controllers :- This They decides who talks and who listens and also the specific test methodology.





GPIB Bus Structure.

→ The 5 management bus signals are given.

1, Interface clear (IFC) :- This resets all the devices from the bus.

2, Attention (ATN) :- The controller uses this signal to send command words to all the devices.

3, Service Request (SRQ) :- The devices are the restiquests (listeners or talkers) use this signal to interrupt the controller for a specific job

4, Remote Enable (REN) :- The controller sends this signal to the devices to disable the front panel controls.

5, End or Identify (EOI) :- This is used by the Talker to indicate that the transfer of block of data is complete.

The three handshake signals data valid (DAV), Not Ready for Data (NRFD), Not Data Accepted (NDAC) are used to coordinate the transfer of data on the data bus

19/3/2013

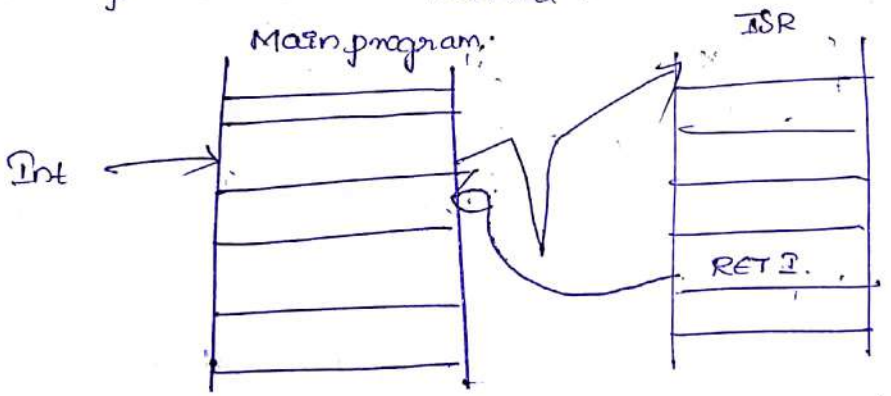
UNIT - V.

8051 : Real Time control : Interrupts, timer / counter and serial communication programming the timer interrupts, Programming external hardware interrupts, programming the Serial communication interrupts, Programming 8051.

8051 Interrupt :-

Interrupt is an input to the processor where external device informs the processor that it is ready for communication when interrupt is received, the processor executes the specific group of instructions called ISR by branching appropriately. The starting address of ISR is called interrupt vector.

The following steps are taken by the processor when interrupt vector is received.



1) It executes current instruction of main program.
2) The flag data ie, the contents of PSW register are pushed to the stack.
3) The contents of pc are pushed to the stack.
4) The program counter is loaded with starting address of ISR.
5) ISR gets executed.

6) When RET I is encountered, pop from stack top to the pc
7) Pop the flag data from the stack. So, when pc is loaded with earlier values, the main is resumed appropriately.

Classification of Interrupts :-

They can be classified as external and Internal interrupts. External IO devices send the signals to the pins of Processor incase of external interrupts.

→ In case of 8086, the external pins are NMI and INTR
→ In case of 8051 μc, there are two external interrupts and 3 internal interrupts.
→ Internal interrupts are activated inside the μc (or) by the execution of interrupt instructions.
→ Another way of classifying the interrupts is Maskable and Non Maskable interrupts.
→ Interrupts which can be disabled by the μc, by programming the configuration registers are called maskable interrupts.
→ The other type is non maskable which cannot be disabled μc.
→ In 8086, NMI is non maskable INTR is maskable.
→ RESET is also a type of interrupt which is non mask
→ There are two ways where interrupt vector can be decided

a) Vectored Interrupts :- Here starting address of ISR (Interrupt vector) is predefined at design stage.

(b) Non vectored Interrupts :- In the first phase the external device sends IntR signal, then processor executes current instruction and then sends $\overline{INTA}$ signal to the external device after receit of $\overline{INTA}$, the external device sends the starting address to the program in next phase.

→ Write delay subroutine using 8051 for 1sec.

```
A1,  Mov R3, #10d.        — 1 sec
loop1 : Mov R2, #100d     100ms
Loop 2 : Mov R1, #125d · 1ms
loop 3 : PUSH A
         POP A
         NOP
         NOP.
         DJNZ R1, loop3     — 125 x 4
         DJNZ R2, loop2     125 x 8 x 100 + 2 x 100
         DJNZ R3, loop1     + 2 x 10  = 1.00409 sec
         END                = 1msec  1sec + 4μ0  1.220
                                     ±4°
```

Write a program in 8051 to find smallest no. from array of num. The array starting address is 2000h.

```
ORG1  0030h.
TEMP  EQU 50h
  N   EQU 05h ; we have to find the small no. from
                array of 5 no's.
Mov R4, N-1
Mov DPTR, #2000.
Loop1: Movx A, @ DPTR
       Mov R1, A.
```

```
AGAIN : INC DPTR
        MOVX A, @DPTR
        MOV TEMP, A
        MOV A, R1 ; The first no. is in A, second in TEMP.
        CJNE A, TEMP, loop 2
        SJMP loop 3.
loop 2 : JC loop 3
        MOV A, TEMP
loop 3 : DJNZ R4, loop 1.
```

21/3/2013

Table 1 :-

Interrupt Source	Interrupt vector address	Associated interrup flag.
(1) External interrupt 0 $\overline{INT0}$ P3.2 pin 12	0003h	IE0 Highest
(2) Timer 0 Interrupt.	000Bh	TF0.
(3) External Interrupt 1 $\overline{INT1}$ P3.3 pin 13	0013h	IE1.
(4) Timer 1 Interrupt.	001Bh.	TF1
(5) Serial port Interrupt	0023h	TI/RI.

lowest

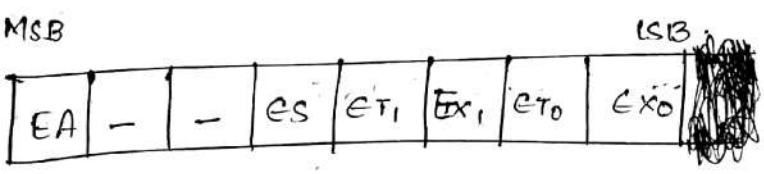
→ the 8051 has a provision for 5 vectored interrupts. Two are external interrupts and 3 are internal interrupts. When an interrupt is generated, the PSW and PC values are pushed to the stack and program counter is loaded

with ISR starting address as shown in above table. When RETI is encountered in ISR, the pc and psw registers are loaded with old values by poping down to the stack.

→ When an Interrupt is generated, the associated flag of TCON register is set as shown in above table.

As seen in the above table, the interrupt vectors are seperated in a gap of 8 bytes. So, the interrupt service routine should not exceed 8 bytes including RETI instruction. for bigger interrupt programs, JMP instruction is placed at the starting address of ISR so that, the ISR can be located somewhere else in bigger dimension.

IE (Interrupt enable) Register of 8051.



1000 1100

EA = 0 disables all Interrupts

= 1 Enables Interrupt, But the corresponding individual interrupt enable bit is also to be set.

ES = Serial port Interrupt enable bit

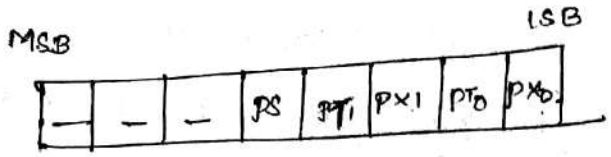
ET1 = Timer 1 overflow interrupt enable bit.

EX1 = External Interrupt 1 enable bit.

ET0 = Timer 0 overflow interrupt enable bit.

EX0 = External interrupt 0 enable bit

Interrupt priority Register in 8051 :-



PS : Serial port priority Interrupt bit

PT1 : Timer 1 interrupt priority bit

PX1 : External Interrupt 1 priority bit

PT0 : Timer 0 interrupt priority bit

PX0 = External interrupt priority bit.

Priority bit = 1 high priority

= 0 low priority.

→ If two interrupts are of same priority as per IP register are received symultaneously, then internal poling sequence as per table 1 is refered. So, within each priority level, there is a second priority structure determined by polling sequence.

Ex1: When two interrupts with different priority level occur at same time.

MOV IE, #1000 1100 B; Enable only Timer 1 overflow interrupt, Ext INT 1.

SETB PT1; Timer 1 overflow interrupt high priority as PT1 bit is Set (ie 1).

If the above two interrupts occur at the same time, Timer 1 overflow interrupt is serviced first as it has high priority.

SETB PX1 ; External interrupt 1 also has high priority.

As timer 1 interrupt is 1 and external interrupt 1 is 1 both have high priority, the µc refers to table 1 and external interrupt 1 is executed first.

TCON Register format :-

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 = Timer 1 overflow flag.

TF0 = Timer 0 overflow flag.

IE1 = Ext interrupt 1 flag

IE0 = Ext interrupt 0 flag.

IT1 = 0 low level triggered

= 1 High to low level transition triggered

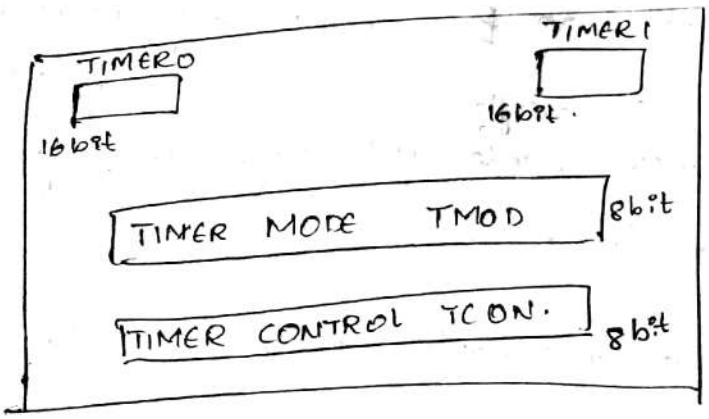
SCON Register format :-

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

TI - Transmit Interrupt flag.

RI - Receive Interrupt flag.

2013



0000h

FFFFh

fig ①

Timers are used in many applications such as time reference, time delay creation, pulse width period and frequency measurement, waveform generation and periodic interrupt generation. The various timer registers of 8051 is given in fig ①.

Format of TCON Register :-

TF1 - TIMER1 overflow flag

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

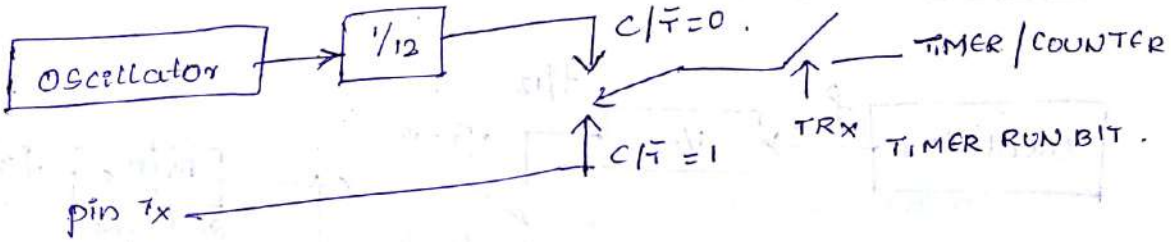
8 bit	8 bit

00 00 hex FF FF h

Format of TMOD Register :-

GATE	C/T	M1	M0	GATE	C/T	M1	M0

Timer 1 → ← TIMER0



TCON Register

TF1 = TIMER 1 overflow flag.
This is set when TIMER1 overflows.

TR1 = TIMER1 RUN CONTROL BIT.

TF0 = TIMER 0 overflow flag.

TR0 = TIMER0 RUN CONTROL BIT.

IE1 = External interrupt 1 flag.

IE0 = External interrupt 0 flag

IT1 = TIMER 1 Interrupt

IT0 = TIMER 0 INTERRUPT

TMOD Register

GATE : If 0 TIMER 0/TIMER 1 fully controlled by TR0 / TR1.

If 1 TIMER 0/TIMER 1 is controlled by TR0/TR1 and $\overline{INT0}$ / $\overline{INT1}$ and this is used for pulse width measurements.

C/$\overline{T}$ if 0 = Timer mode internal clock pulse.

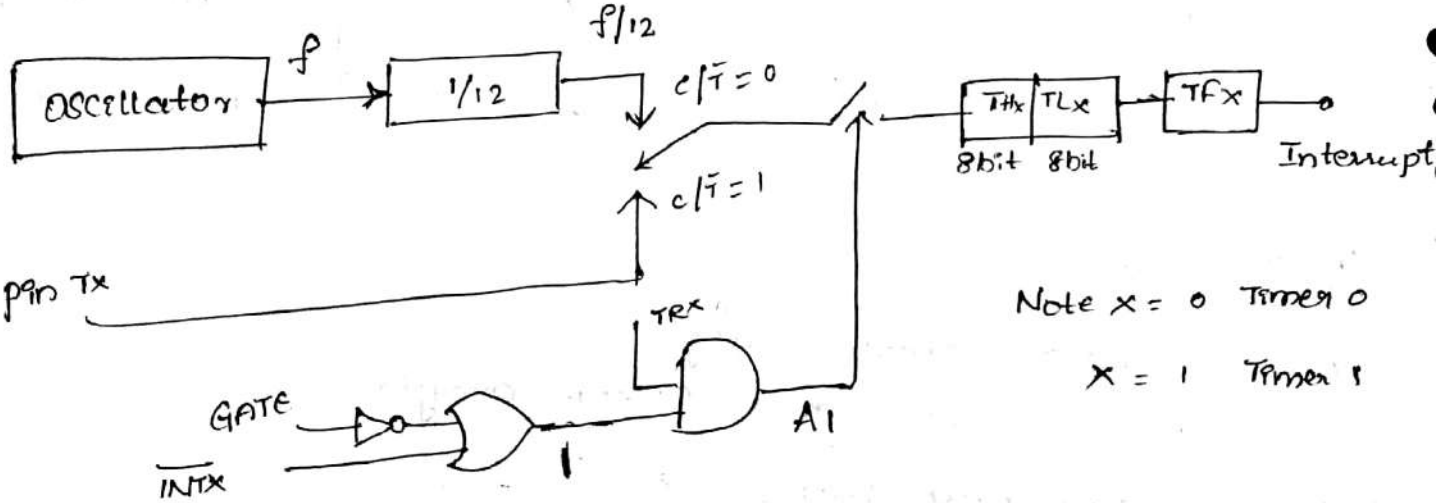
If 1 = Counter mode. The i/p is External clock pulse.

M1	M0		
0	0	Mode 0	13 bit timer.
0	1	Mode 1	16 bit timer.
1	0	Mode 2.	8 bit timer with Auto reload.
1	1	Mode 3.	used for Baud rate generation.

12013

TH0 8 TL0 = T_0 8

Mode 1 TIMER OPERATION :-



Note x = 0 Timer 0

x = 1 Timer 1

$P_{3.2}$	$\overline{INT0}$	I/p	pin 12 of 8051 µc.
$P_{3.3}$	$\overline{INT1}$	I/p	pin 13.
$P_{3.4}$	T_0	Ext. clock	for Timer 0
$P_{3.5}$	T_1	Ext. clock	for Timer 1

Time delay = (65535 - Initial count) $\times \frac{12}{f}$.

0000.

FFFF

The above fig shows timer operation.

If $C/\overline{T} = 0$, the above functional block works as a timer. If $C/\overline{T} = 1$, the block works as a counter. In either case, the external clock is applied at P3.4/P3.5. If the timer is not controlled by external interrupt, GATE = 0 i.e $g = 0$. Then, timer operation is fully controlled by TRx bit.

When TRx = 1 and when clock is applied, the timer value gradually changes from 0000h, 0001h ... FFFFh. When maximum value FFFF is reached, the timer value roles back to zero. At the instant of hold back it sets the timer flag and an internal interrupt is generated.

When Interrupt is used to control the timer, GATE = 1, TRx = 1 is and the timer starts counting. When External interrupt occurs (active low), the output of AND gate A1 becomes zero disabling the timer. The Mode 1 can be used to create various time delays as from the formula, time delay = (65535 - Initial count) $\times \frac{12}{f}$.

→ An oscillator is running at 12MHz controlling 8051 µc. Write the subroutine to create a time delay of 20msec

As clock frequency = 12MHz.

clock period = $\frac{1}{12}$ µsec.

Machine cycle period = $12 \times \frac{1}{12}$ µsec = 1µsec as each machine cycle consists of 12 clock pulses.

Required time delay = 20msec.

Time delay = (65536 - Initial count) $\times \frac{12}{f}$.

$\frac{20 \times 10^{-3}}{1\mu sec}$ = (65536 - Initial count)

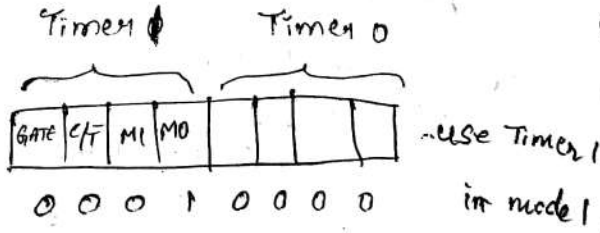
65586 - Initial count = 20000.

⇒ Initial count = 65536 - 20,000

= 45536

= B1E0h.

Subroutine for 20msec.




```
DELAY : MOV TMOD, #10h.
        CLR TF1
        CLR ET1 , disable timer 1 interrupt.
        MOV TH1, #B1h.  } ; Initial count is loaded into
        MOV TL1, #E0h.  }   TIMER 1.
        SETB TR1        ; TIMER 1 starts counting.
WAIT:   JNB TF1 WAIT    ; wait till timer flag is set.
        RET.
```

→ Assume an oscillator is running at 11.0592 MHz controlling 8051 µc. Write a program to generate 2KHz square wave at port P1.3 (pin 4) using timer 0 Interrupt.

A, Clock frequency = 11.0592 MHz.

Machine cycle period = $12 \times \frac{1}{11.0592 \times 10^6} = 1.085\ \mu sec$.

Period of square waveform = $\frac{1}{2KHz} = 500\mu sec$.

So duration of high portion = Duration of low portion

$= 250\mu sec$.

So, Initial count = $\frac{250\ \mu sec}{1.085\ \mu sec} = 230$.

Initial count = 65536 - 230

= 65306 = FFFAh

Soln

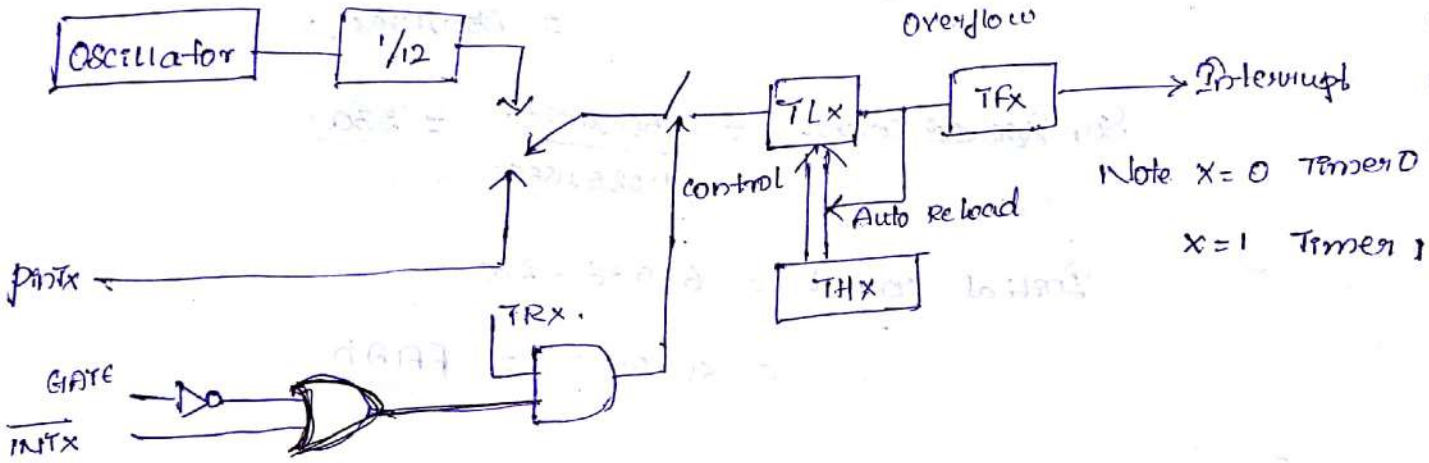
```
ORG 0030h
MOV TMOD, 01h ; Timer 0 in Mode 1.
MOV TL0, #1Ah ; Load initial count for 250 µsec.
MOV TH0, #ffh;
CLR TF0 ; Timer flag is cleared
MOV IE, #82h
LOOP1 : SETB TR0
HERE : SJMP HERE
SJMP LOOP1
```

ISR programme

```
ORG 000B
CLR TR0
CPL P1.3
MOV TL0, #1Ah.
MOV TH0, #FFh.
RETI.
```

4/4/2013

Mode 2 Timer of 8051 :-



$$\text{Time delay} = (256 - \text{Initial count}) \times \frac{12}{f}.$$

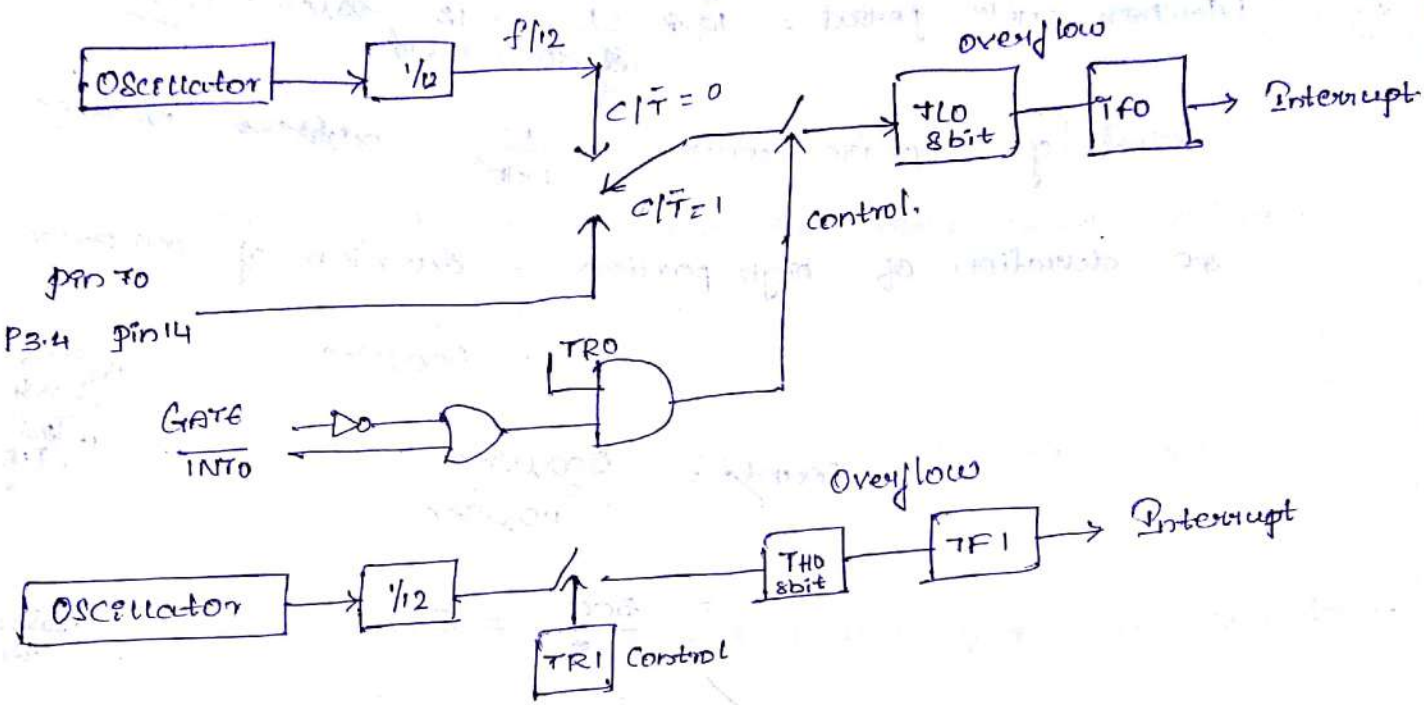
This is an 8-bit Timer and the count changes from 00h to FFh at the instant of each machine cycle clock.

When the count reaches FF, It sets the appropriate timer flag and an Internal interrupt is generated. The delay is calculated as per the formula $(256 - \text{Initial count}) \times \frac{12}{f}$.

Please the Initial count has to be loaded only once into TLx and TFx registers. Subsequently during overflow the Initial count is automatically reloaded from THx to TLx

Rest of the interrupt operation of timer and counter mode operation is same as mode 1.

Mode 3 Timer of 8051 :-

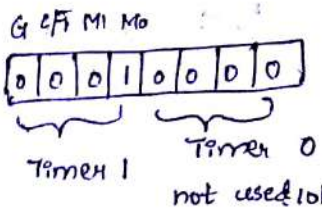


Timer 0 in Mode 3 uses TL0 and TH0 as separate 8 bit timers. TL0 uses GATE, $\overline{INT0}$ and TF0, whereas TH0 uses TR1 and TF1. The interrupt mode of operation and counter mode operation is available only for TL0 and not available for TH0. When Timer 0 is in mode 3, timer 1 can be made operate in Mode 0 or Mode 1 or Mode 2, but this is without TR and TR1 bits.

This is useful to generate baud rates for Serial communication.

The Software without ISR is given below,

```
ORG 0030h
Mov TMOD, #10h  ; Setup Timer 1 in Mode 1
CLR ETI         ; Disable Timer 1 interrupt
Loop1: CLR TF1  ; clear the timer 1 overflow flag
       Mov TH1, #fe h  ; load the initial count into Timer 1
       Mov TL1, #0c h
       SETB TR1
WAIT; JNB TF1, WAIT  ; WAIT till Timer 1 overflows.
       CLR TR1       ; Stop the timer.
       CPL P1.0      ; complement P1.0 to get high and low.
       SJMP LOOP1
       END.
```



→ Assume an oscillator is running at 12MHz with 8051 μc write a program to generate 4KHz square on port 1.2 (pin3) using timer 0 in Auto reload mode

A, Machine cycle period = $\frac{12}{12\times10^6}$ = 1μsec.

Period of square wave = $\frac{1}{4\times10^3}$ = 250 μsec.

So, duration of high portion = duration of low portion.

= 125 μsec.

Time delay = (65536 − Initial count) × 1μsec.

125 μsec = (65536 − Initial count) × 1μsec

Initial count = 65536 − 125

Time delay = $(65411)_{10}$ = FF83h.

ORG 0030h

Time delay = (256 - Initial count) × 1μsec

125μsec = (256 - Initial count) × 1μsec

Initial count = 256 - 125

= $(131)_{10}$

Initial count = 83h.

```
ORG 0030h
Mov TMOD, #02h   ; Set up Timer 0 in mode 2 Auto relaxed
CLR TF0
CLR ET0          ; Disable timer 0 Interrupt. ISR not needed.
Loop: Mov TH0, #83h.
Mov TL0, #83h.
Loop1: SETB TR0
BACK: JNB TF0 BACK.
CLR TR0
CPL p1.2.
CLR TF0
SJMP loop1
END.
```

→ Assume an oscillator running at 11.0592 MHz with 8051. Write a program to generate 1KHz square wave at port P1.1 using Timer 0.

A, Period of Square wave $= \frac{1}{1 \times 10^3} = 1000\ \mu sec$.

ON period = Off period = 500 μsec.

Time delay $= (65536 - Ic) \times \frac{12}{11.0592\ MHz}$.

$500 = (65536 - Ic) \times \frac{12}{11.0592}$

$500 = (65536 - Ic) \times 1.08$

$Ic = 65536 - 462$

$= 65075_{10} = FE33h$.

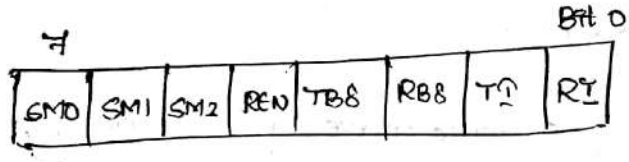
```
#include <Intel\8051.h>.
# define on 1
# define off 0.
bit Square wave pin = P1.1      // Pin 1 of Port 1.
  void delay 1KHz ( )           // Delay program for 500μsec.
main ( ) { TMOD = 0x01 ;
           conte(1) { Square wave pin = on ;
                      delay 1KHz ( ) ;
                      Square wave pin = off ;
                      delay 1KHz ( ) ;
                    }
                  }
```

```
Void delay 1KHz ( ) {
        TH0 = 0x fe ;
        TL0 = 0 x 33 ;
        TR0 = on
        while ( ! TF0 );
        TR0 = off
        TF0 = off
        }
```

Serial communication in 8051

* 8051 supports serial communication both synchronous and asynchronous.
* 3 special function registers support serial communication in 8051. They are 1) SCON register (Serial control register)
  2) PCON register (power control)
  3) SBUF (Serial buffer) register.

SCON Register :-



SM0	SM1	Mode	Description	Baud rate
0	0	0	8bit Sync. comm	fosc /12 (fixed)
0	1	1	10bits (8 bits + 1 start + 1 stop)	Variable
1	0	2	11 bits (8 bits + 1 start + 1 stop + parity)	fixed baud rate fosc/32, fosc/64
1	1	3	11 bits (8 bits + 1 start + 1 start + parity	Variable baud rate

Standard baud rates - 19,200

19,600.

SM2 :-

In Mode 2 and 3 if this bit is set. It enables multiprocessor communication.

REN :- Receiver enable.

TB8 :- This is the parity bit transmitted in mode 2 and 3.

RB8 :- This is the parity bit received in mode 2 & 3.

In case of mode 1, this is the stop bit received;

TI :- Transmit interrupt flag. This is set by hardware when all the contents of SBUF are transmitted.

RI :- Receive Interrupt flag. When SBUF is full in reception, after accumulating all 8 bits of data, this RI flag is generated by Hardware and is cleared by Software.

PCON register :- The MSB of PCON is SMOD and this is used to control baud rate.

SBUF :- This is an 8-bit register. The μc has 2 seperate SBUF register. Thus the SBUF used for transmission is write only device. The data of this is transmitted using TXD pin. The SBUF used for receiver is read only register and this is connected to RXD pin to receive the data.

Mode 0 :- Here RXD pin is used to transmit and received data. Hence it is half duplex.

TXD provides the common reference clock.

→ It is 8 bit synchronous communication.

Mode 1 :-

Here 10 bits are used with 8 bit char + 1 start + 1 stop bit. The baud rate is variable and it is determined by SMOD and Timer 1 overflow rate.

$$\text{Baud rate} = \left[\frac{2^{SMOD}}{32}\right] \quad (\text{Timer 1 overflow rate})$$

$$= \left[\frac{2^{SMOD}}{32}\right]\left[\frac{f_{osc}}{12(256 - TH1)}\right].$$

Timer 1 is normally used in Mode 2 ie, 8 bit timer with Auto reload.

Mode 2 :-

Here, we use 11 bits (8bits char + 1 Start + 1 Stop + 1 parity bits). Here baud rate is fixed and given by the formula

$$\text{Baud rate} = \left[\frac{2^{SMOD}}{64}\right] \times f_{osc}.$$

Mode 3 :-

Mode 3 is same as mode 2. but wi the baud rate formula of mode 1 is used.

$$\text{Baud rate} = \left[\frac{2^{SMOD}}{32}\right]\left[\frac{f_{osc}}{12(256 - TH1)}\right]$$

→ Calculate the baud rate Mode 1 Serial transmission if $f_{osc} = 12MHz$. $f_{osc} = 11.0592\ MHz$. Assume SMOD = 0 and TH1 = 253dec.

A, Baud rate $= \frac{1}{32} \times \frac{10^6 \times 11.0592}{(256-253)}$

$= 10416.66$ 9600 $\simeq$ 9.6 kbps.

for $f_{osc} = 11.0592 \Rightarrow \frac{1}{32} \times \frac{11.0592}{12 \times 3} \times 10^6$

$= 9600$

$= 9.6$ Kbps.

→ Write a program to initialise 8051 to operate in mode 1 for receiving a Serial byte through RxD pin. The received data is to be set to port 8. operate the timer 1 in Autoreload Assume SMOD = 0. The data is received at a baud rate of 9.6Kbps and internal osc freq $f_{osc} = 11.0592\ MHz$.

A, the Initial count is calculated as per the formula.

$$\text{Baud rate} = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12(256 - TH1)}$$

$$9.6 \times 10^3 = \frac{1}{32} \times \frac{11.0592 \times 10^6}{12(256 - TH1)}$$

$TH1 = (253)_{10}$.

$= f_D\ h$.