MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

DEPARTMENT OF ELECTRONICS & COMMUNICATION SCIENCE

SUBJECT NAME: MICROPROCESSOR (INTEL 8085)

SUBJET CODE: TAG5A

SEMESTER: V

PREPARED BY: PROF.V.SAVITHRI

CORE 9 - MICROPROCESSOR (INTEL 8085)

UNIT I

ARCHITECTURE OF 8085 MICROPROCESSOR – Demultiplexing address / data bus – Control SignalGeneration and status signals – 8085 – pin-out diagram & functions - Interrupts - Priority Concept

INSTRUCTION SET OF 8085 – Instruction classification – addressing modes

UNIT II.

MEMORY– Instruction cycle – machine cycle – T-state -Timing diagrams for Opcode Fetch Cycle Memory Read,Memory Write, I/O Read, I/O Write, – Functional explanation for RAM, ROM, EPROM, EEPROM.

UNIT III

PROGRAMMING EXERCISES – addition & subtraction(16-bit), multiplication, division, largest, smallest, blocktransfer (all 8-bit data), Binary to BCD, BCD to Binary, Binary to ASCII, ASCII to Binary, BCD to ASCII, ASCII to BCD (all 8-bit data) - Stack & Subroutines Concept – time delay using single register & calculations – Debugging a program.

UNIT IV

INTERFACING MEMORY – 2K X 8, 4K X 8 ROM, RAM to 8085, Interfacing an I/O port in Memory MappedI/O and I/O Mapped I/O – Difference between I/O mapped and Memory Mapped I/O.

UNIT V

MICROPROCESSOR APPLICATIONS – Programmable peripheral devices (8255, 8253) – Pin functions,Different Modes & Block Diagram - Keyboard and Display Interface 8279 (Architecture) - Simple temperature controller – Simple traffic light controller.

MICROPROCESSOR 8085

INTRODUCTION

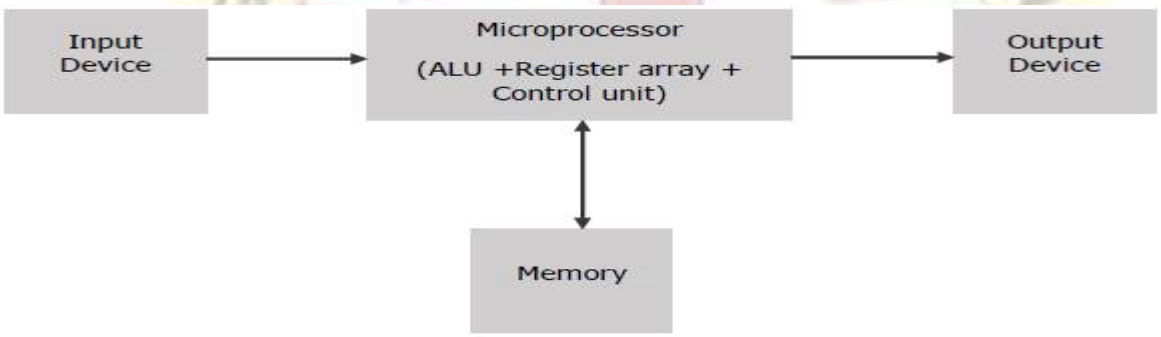
UNIT I

Microprocessor – Overview

Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.

Microprocessor consists of an ALU, register array, and a control unit. ALU performs arithmetical and logical operations on the data received from the memory or an input device. Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator. The control unit controls the flow of data and instructions within the computer.

Block Diagram of a Basic Microcomputer



How does a Microprocessor Work?

The microprocessor follows a sequence: Fetch, Decode, and then Execute.

Initially, the instructions are stored in the memory in a sequential order. The microprocessor fetches those instructions from the memory, then decodes it and executes those instructions till STOP instruction is reached. Later, it sends the result in binary to the output port. Between these processes, the register stores the temporarily data and ALU performs the computing functions.

8085 Pin Diagram | Functional Pin Diagram of 8085 Microprocessor:

The signals of 8085 Pin Diagram can be classified into seven groups according to their functions.

1. **Power supply and frequency signals.**
2. **Data bus and address bus**
3. **Control bus**
4. **Interrupt signals**
5. **Serial I/O signals**
6. **DMA signals**
7. **Reset signals**

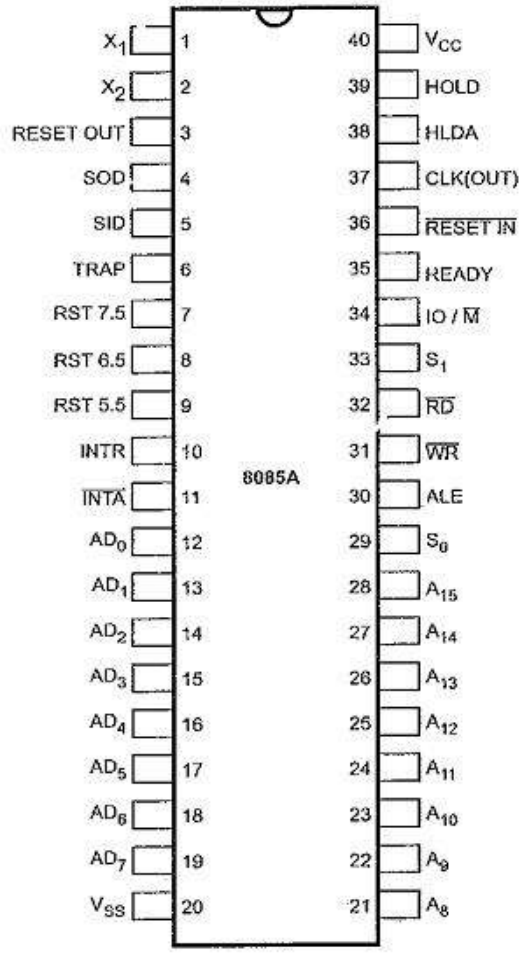


Fig. 1.3 (a) Pin configuration

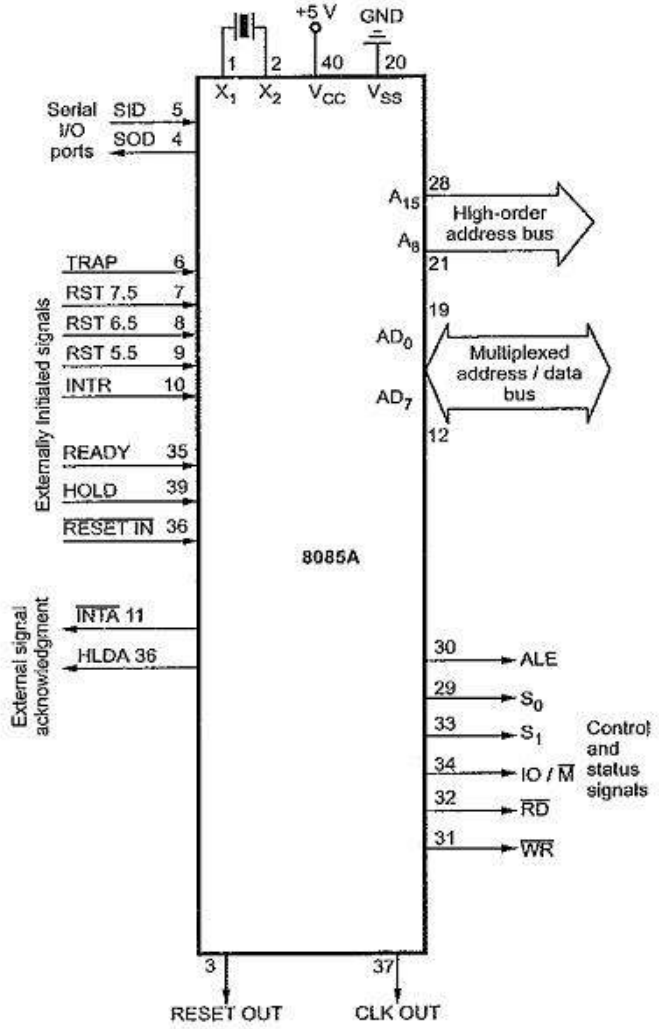


Fig. 1.3 (b) Functional pin diagram

1. Power Supply and Frequency Signals:

- **V_{cc} : It requires a single +5 V power supply.**
- **V_{ss} : Ground reference.**
- **X_1 and X_2 : A tuned circuit like LC, RC or crystal is connected at these two The internal clock generator divides oscillator frequency by 2, therefore, to operate a system at 3 MHz, the crystal of tuned circuit must have a frequency of 6 MHz.**
- **CLK OUT : This signal is used as a system clock for other devices. Its frequency is half the oscillator frequency.**

2. DATA BUS AND ADDRESS BUS:

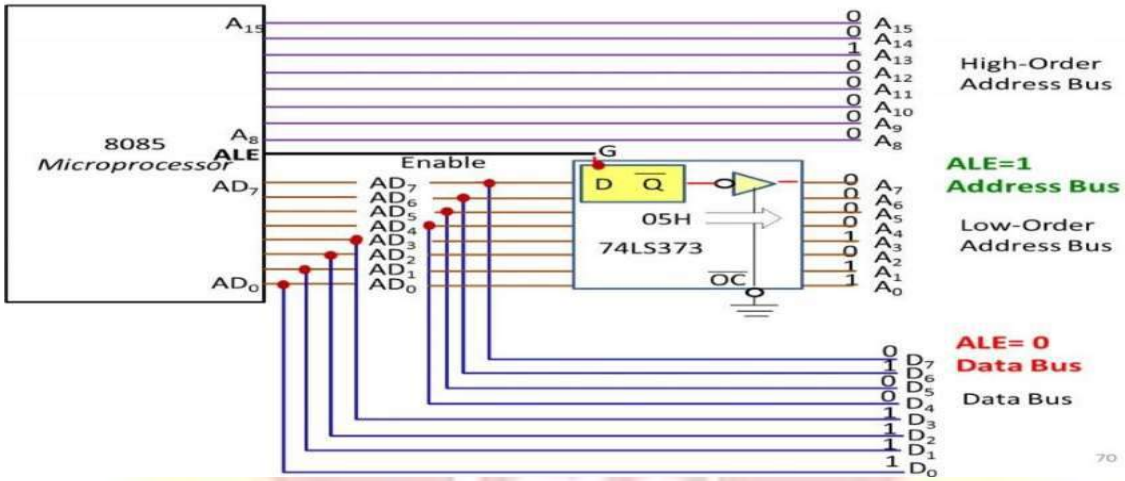
A) AD_0 to AD_7 : The 8 bit data bus ($D_0 – D_7$) is multiplexed with the lower half ($A_0 – A_7$) of the 16 bit address bus. During first part of the machine cycle (T_1), lower 8 bits of memory address or I/O address appear on the bus. During remaining part of the machine cycle (T_2 and T_3) these lines are used as a bi-directional data bus.

B) A_8 to A_{15} : The upper half of the 16 bit address appears on the address lines A_8 to A_{15}. These lines are exclusively used for the most significant 8 bits of the 16 bit address lines.

. CONTROL AND STATUS SIGNALS:

A) ALE (Address Latch Enable) : We, know that AD_0 to AD_7 lines are multiplexed and the lower half of address ($A_0 – A_7$) is available only during T_1 of the machine cycle. This lower half of address is also necessary during T_2 and T_3 of machine cycle to access specific location in memory or I/O port. This means that the lower half of an address must be latched in T_1 of the machine cycle, so that it is available throughout the machine cycle. The latching of lower half of an address bus is done by using external latch and ALE signal from 8085 Pin Diagram.

1. DemultiplexingAD0-AD7



- Thehigher-orderbusremainsonthebusforthreeclockperiods.However,thelow-orderaddressislostafterthefirstclockperiod.

Thisaddressneedtobelatchedandusedforidentifyingthememoryaddress.IfthebusAD7-AD0isusedtoidentifythememorylocation(2005H),theaddresswillchangeto204FHafterthefirstclockperiod.

- FigureshowsaschematicthatusesalatchandtheALEsignaltodemultiplexthebus.
- ThebusAD7-AD0isconnectedastheinputtothelatch.
- TheALEsignalisconnectedtotheEnablepinofthelatch,andtheoutputcontrolsignalofthelatchisgrounded.
- FigureshowsthattheALEgoeshighduringT1.AndduringT1addressoflower-orderaddressbusisstoreintothelatch.

B) RD and WR: These signals are basically used to control the direction of the data flow between processor and memory or I/O device/port. A low on RD indicates that the data must be read from the selected memory location or I/O port via data bus. A low on WR indicates that the data must be written into the selected memory location or I/O port via data bus.

C) IO/M, S_0 and S_1 : IO/M indicates whether I/O operation or memory operation is being carried out. S_1 and S_0 indicate the type of machine cycle in progress.

Truth Table of Control Signal

Operation	Status		
	IO/$\overline{M}$	S1	S0
Opcode Fetch	0	1	1
Memory Read	0	1	0
Memory Write	0	0	1
I/O Read	1	1	0
I/O Write	1	0	1
INTR Acknowledge	1	1	1
Bus Idle	0	0	0

. ***INTERRUPT SIGNALS:***

The 8085 Pin Diagram has five hardware interrupt signals : RST 5.5, RST 6.5, RST 7.5, TRAP and INTR. The microprocessor recognises interrupt requests on these lines at the end of the current instruction execution.

- The INTA (Interrupt Acknowledge) signal is used to indicate that the processor has acknowledged an INTR interrupt.
  INTR(Input) InterruptRequest.Itisusedasgeneralpurposeinterrupt
- INTA'(Output) InterruptAcknowledge.Itisusedtoacknowledgeaninterrupt.
- RST7.5,RST6.5,RST5.5(Input) RestartInterrupts.
  - Thesearevectorinterruptsthattransfertheprogramcontroltospecificmemorylocations.
  - TheyhavehigherprioritiesthanINTRinterrupt.
  - Amongthese3interrupts,thepriorityorderisRST7.5,RST6.5,RST5.5
- TRAP(Input) Thisisanon-maskableinterrupt&hasthehighestpriority.

5. SERIAL I/O SIGNALS:

A) SID (Serial I/P Data) : This input signal is used to accept serial data bit by bit from the external device.

B) SOD (Serial O/P Data) : This is an output signal which enables the transmission of serial data bit by bit to the external device.

6. DMA Signal:

A) HOLD : This signal indicates that another master is requesting for the use of address bus, data bus and control bus.

B) HLDA : This active high signal is used to acknowledge HOLD request.

7. RESET SIGNALS:

A) RESET IN: A low on this pin

- **Sets the program counter to zero (0000H).**
- **Resets the interrupt enable and HLDA flip-flops.**
- **Tri-states the data bus, address bus and control bus. (Note : Only during RESET is active).**
- **Affects the contents of processor's internal registers randomly.**

On reset, the PC sets to 0000H which causes the 8085 Pin Diagram to execute the first instruction from address 0000H. For proper reset operation reset signal must be held low for at least 3 clock cycles. The power-on reset circuit can be used to ensure execution of first instruction from address 0000H.

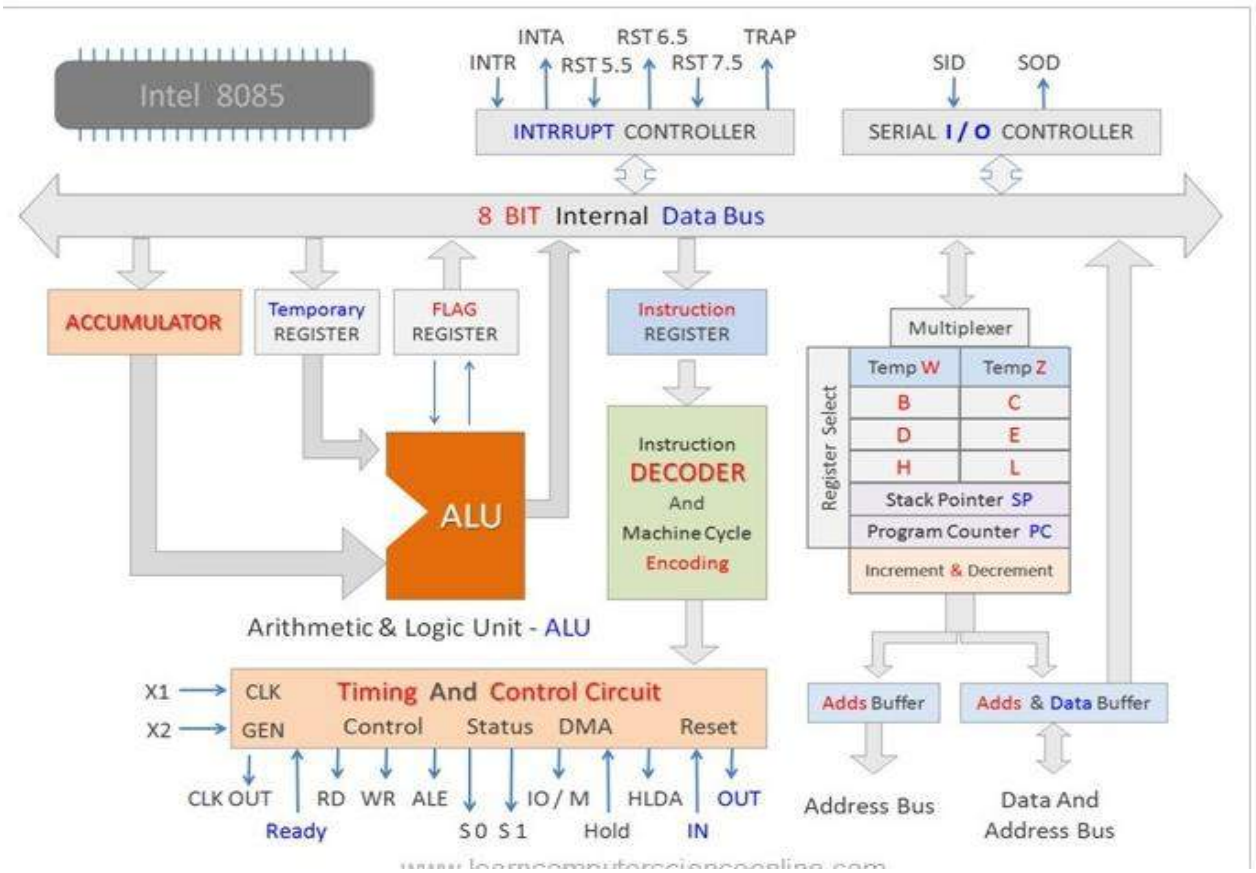
B) RESET OUT: This active high signal indicates that processor is being reset. This signal is synchronized to the processor clock and it can be used to reset other devices connected in the system.

MICROPROCESSOR - 8085 ARCHITECTURE

- It is a 40 pin I.C. package fabricated on a single LSI chip.
- The Intel 8085 uses a single +5Vd.c. supply for its operation.
- Intel 8085s clock speed is about 3 MHz; the clock cycle is of 320ns.
- 8bit data bus.
- Address bus is of 16-bit, which can address up to 64KB
- 16-bit stack pointer
- 16bit PC (Program Counter)
- Six 8-bit registers are arranged in pair: BC, DE, HL

.

The below diagram shows the **block diagram of Intel 8085**:



- Thearchitectureofmicroprocessor8085canbedividedintosevenpartsasfollows:

RegisterUnit:

GeneralPurposeDataRegister

- 8085hassixgeneralpurposedataregisterstostore8-bitdata.
- TheseregistersarenamedasB,C,D,E,HandLasshowninfig.1.
- Theusercanusetheseregisterstostoreorcopyadatatemporarilyduringtheexecutionofaprogrambyusingdatatransferinstructions.
- Theseregistersareof8bitsbutwheneverthemicroprocessorhastohandle16-bitdata,theseregisterscanbecombinedasregisterpairs–BC,DEandHL.
- Therearetwointernalregisters–WandX.TheseregistersareonlyforinternaloperationlikeexecutionofCALLandXCHGinstructionsandnotavailabletotheuser.

ProgramCounter(PC)

- 16-bitregisterdealswithsequencingtheexecutionofinstructions.
- Thisregisterisamemorypointer.

- Memorylocationshave16-bitaddresseswhicharewhythisisa16-bitregister.
- Themicroprocessorusesthisregistertosequencetheexecutionoftheinstructions.
- Thefunctionoftheprogramcounteristopointtothememoryaddressfromwhichthenextbyteistobefetched.
- Whenabyte(machinecode)isbeingfetched,theprogramcounterisincrementedbyonetopointtothenextmemorylocation.

StackPointer(SP)

- SPisalsoa16-bitregisterusedasamemorypointer.
- ItpointstoamemorylocationinR/Wmemory,calledthestack.

Thebeginningofthestackisdefinedbyloading16-bitaddressinthestackpointer

MUX/DEMUXunit

- Thisunitisusedtoselectaregisteroutofalltheavailableregisters.
- ThisunitbehavesasaMUXwhendataisgoingfromtheregistertotheinternaldatabus.
- ItbehavesasaDEMUXwhendataiscomingtoaregisterfromtheinternaldatabusofthemicroprocessor.

Theregisterselectwillbehaveasthefunctionselectionlinesofthe MUX/DEMUX

AddressBufferRegister&Data/AddressBufferRegister

- Theseregistersholdtheaddress/data,receivedfromPC/internaldatabusandthenloadtheexternaladdressanddatabuses.
- Theseregistersactuallybehaveasthebufferstagebetweenthemicroprocessorandexternalsystembuses.

ControlUnit:

- Thecontrolunitgeneratessignalswithinmicroprocessortocarryouttheinstruction,which hasbeendecoded.
- Inrealityitcausesconnectionsbetweenblocksofthemicroprocessortobeopenedorclosed ,sothatthedatagoeswhereitisrequiredandtheALUoperationsoccur.
- Thecontrolunititselfconsistsofthreeparts;theinstructionregisters(IR),instructiondecoderandmachinecycleencoderandtimingand controlunit.

InstructionRegister

- Thisregisterholdsthemachinecodeoftheinstruction.
- Whenmicroprocessorexecutesaprogramitreadstheopcodefromthememory,thisopcode isstoredintheinstructionregister.

InstructionDecoder&MachineCycleEncoder

- TheIRsendsthemachinecodetothisunit.
- Thisunit, as itsnamesuggests,decodestheopcode and findsoutwhatis to bedonein response ofthecomingopcodeandhowmanymachinecyclesarerequiredtoexecutethisinstruction.

Timing&Controlunit

- Thecontrolunitgeneratessignalswithinmicroprocessortocarryouttheinstruction,which hasbeendecoded.
- Inreality,itcausescertainconnectionsbetweenblocksofthemicroprocessortobeopenedo rclosed,sothatthedatagoeswhereitisrequiredandtheALUoperationsoccur.

Arithmetic&LogicalUnit:

- TheALUperformstheactualnumericalandlogicaloperationsuchas‘add’,‘subtract’,‘AND’,‘O R’,etc.
- ALUusesdatafrommemoryandfromaccumulatortoperformthearithmeticoperationsan dalways storestheresultoftheoperationinaccumulator.
- ALUconsistsofaccumulator,flagregisterandtemporaryregister.

Accumulator

- Theaccumulatorisan8-bitregisterthatisapartofALU.
- Thisregisterisusedtostore8-bitdataandperformarithmeticalandlogicaloperations.
- Theresultofanoperationisstoredintheaccumulator.
- ItisalsoidentifiedasregisterA.

Flagsregister

- Flagregisterincludesfiveflip-flops,whicharesetorresetafteranoperationaccordingtothedataconditionsoftheresultint heaccumulatorand otherregisters.
- Theyarecalled zero (Z),carry(CY),sign (S),parity(P)andauxiliarycarry (AC)flags;their bitpositionsintheflagregisterareshown infig.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

- Themicroprocessorusestheseflagstosetandtestdataconditions.

Figure:Flagsregistersin8085.

- Theflagsarestoredinthe8-bitregistersothattheprogrammercanexaminetheseflagsbyaccessingtheregisterthrough aninstruction.
- Theseflagshavecriticalimportanceinthedecision-makingprocessofthemicroprocessor.
- Theconditions(setorreset)oftheflagsaretestedthroughthesoftwareinstructions.
- Forinstance,JC(jumponcarry)isimplementedtochangethesequenceofaprogramwhenC Yflagisset.

Z(Zero)Flag:

- Thisflagindicateswhethertheresultofmathematicalorlogicaloperationiszeroornot.
- Iftheresultofthecurrentoperationiszero,thenthisflagwillbeset,otherwisereset.

CY(Carry)Flag:

- Thisflagindicates,whether,duringanadditionorsubtractionoperation,carryorborrowisg eneratedornot,ifgeneratedthenthisflagbitwillbeset.

AC(AuxiliaryCarry)Flag:

- ItshowscarrypropagationfromD3positiontoD4position.
- AsshowninthefIg.,acarryisgeneratedfromD3bitpositionandpropagatestotheD4positio n.Thiscarryiscalledauxiliarycarry.

S(Sign)Flag:

- Signflagindicateswhethertheresultofamathematicaloperationisnegativeorpositive.
- Iftheresultispositive,thenthisflagwillresetandiftheresultisnegativethisflagwillbeset.
- Thisbit,infact,isareplicaoftheD7bit.

P(Parity)Flag:

- Parityisthenumberof1'sinanumber.
- Ifthenumberof1'sinanumberiseventhenthatnumberisknownasevenparitynumber.
- Ifthenumberof1'sinanumberisoddthenthatnumberisknownasanoddparitynumber.
- Thisflagindicateswhetherthecurrentresultisofevenparity(set)orofoddparity(reset).

InterruptControl

- Theinterruptcontrolunithas5interruptinputsTRAP,RST7.5,RST6.5,RST5.5&INTRa ndoneacknowledgesignalINTA.
- Itcontrolstheinterruptactivityof8085microprocessor.

SerialIOcontrol

- 8085serialIOcontrolprovidestwolines,SODandSIDforserialcommunication.
- Theserialoutputdata(SOD)lineisusedtosenddataseriallyandserialinputdataline(SID)is usedtoreceivedataserially.

8085 Instructions

An **instruction** of computer is a command given to the computer to perform a specified operation on given data. Some instructions of Intel 8085 microprocessor are: MOV, MVI, LDA, STA, ADD, SUB, RAL, INR, MVI, etc.

Opcode and Operands

Each instruction contains two parts: Opcode (Operation code) and Operand.

The 1st part of an instruction which specifies the task to be performed by the computer is called Opcode.

The 2nd part of the instruction is the data to be operated on, and it is called Operand. The Operand (or data) given in the instruction may be in various forms such as 8-bit or 16-bit data, 8-bit or 16-bit address, internal registers or a register or memory location.

Instruction Word Size

A digital computer understands instruction written in binary codes (machine codes). The binary codes of all instructions are not of the same length.

According to the word size, the Intel 8085 instructions are classified into the following three types:

1. One byte instruction
2. Two byte instruction
3. Three byte instruction

1. One-byte instruction: Examples of one byte instructions are:

- **MOV A, B** - Move the content of the register B to register A.
- **ADD B** ? Add the content of register B to the content of the accumulator.

All the above two examples are only one byte long. All one-byte instructions contain information regarding operands in the opcode itself.

2. Two-byte instruction: In a two byte instruction the first byte of the instruction is its opcode and the second byte is either data or address.

Example:

MVI B, 05; 05 moved to register B.

06, 05; MVI B, 05 is in the code form.

The first byte 06 is the opcode for MVI B and second byte 05 is the data which is to be moved to register B.

3. Three-byte instruction: The first byte of the instruction is its opcode and the second and third bytes are either 16-bit data or 16-bit address.

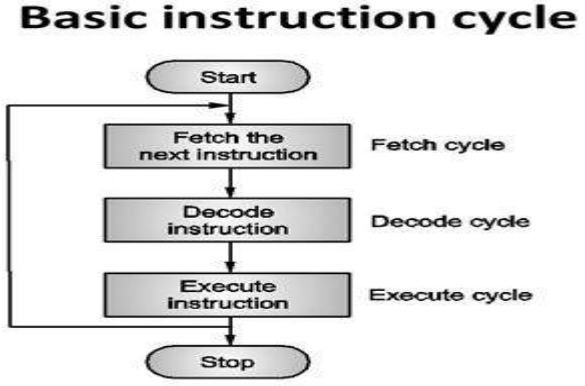
Example:

LXI H, 2400H; Load H-L Pair with 2400H

21, 00, 24; LXI H, 2400H in the code form

The first byte 21 is the opcode for the instruction LXI H. The second 00 is 8 LSBs of the data (2400H), which is loaded into register L. The third byte 24 is 8 MSBs of the data (2400H), which is loaded into register H.

Instruction Cycle



The time required to fetch an instruction and necessary data from memory and to execute it, is called an **instruction cycle**. Or the total time required to execute an instruction is given by:

IC = FC + EC

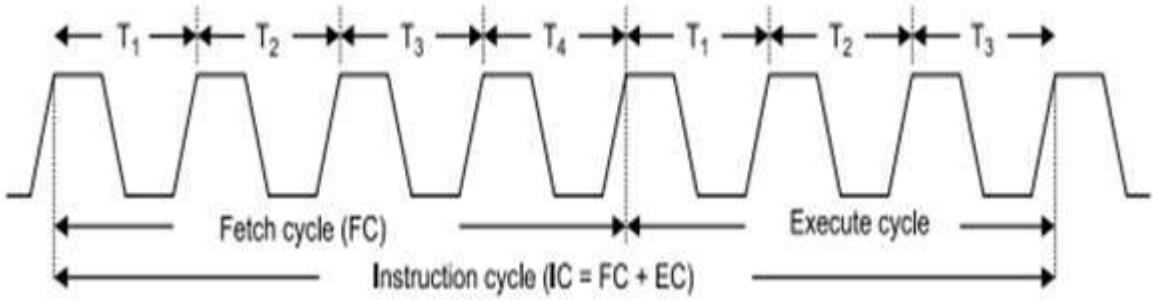
Where,

IC = Instruction Cycle

FC = Fetch Cycle

EC = Execute Cycle

Timing Diagram for Instruction Cycle



Instruction Set of 8085

Instruction and Data Formats

The various techniques to specify data for instructions are:

1. 8-bit or 16-bit data may be directly given in the instruction itself. The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.
2. In some instructions, only one register is specified. The content of the specified register is one of the operands.
3. Some instructions specify two registers. The contents of the registers are the required data.
4. In some instructions, data is implied. The most instructions of this type operate on the content of the accumulator.

Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the ame length. It may 1-byte, 2-byte or 3-byte instruction.

8085 Instructions

An **instruction** of a computer is a command given to the computer to perform a specified operation on given data. In microprocessor, the **instruction** set is the collection of the instructions that the microprocessor is designed to execute.

The programmer writes a program in assembly language using these instructions. These instructions have been classified into the following groups:

Data Transfer Group

Instructions which are used to transfer the data from a register to another register from memory to register or register to memory come under this group**...**

Instruction Set	Explanation	States	Flags	Addre-ssing	M C	Example
MOV r_1, r_2 [r1] ← [r2]	Move the content of the one register to another	4	none	Register	1	MOV A, B
MOV r, M [r]←[[H-L]]	Move the content of memory to register	7	none	Register Indirect	2	MOV B, M
MOV M,r [[H-L]]←[r]	Move the content of register to memory	7	none	Register Indirect	2	MOV M, C
MVI r, data [r] ←data	Move immediate data to register	7	None	Immediate Register	3	MVI M, 08
LXI rp, data 16 [rp] ←data 16 bits, [rh] ←8 MSBs, [rl] ←8 LSBs of data	Load Register pair immediate	10	None	Immediate	3	LXI H, 2500H
LDA addr	Load	13	None	Direct	4	LDA

[A] ←[addr]	Accumulator direct					2400 H
STA Addr [addr] ←[A]	Store accumulator direct	13	None	Direct	4	STA 2000H
LHLD addr [L] ←[addr], [H] ← [addr + 1]	Load H-L pair direct	16	None	Direct	5	LHLD 2500H
SHLD addr [addr] ←[L], [addr +1] ← [H]	Store H-L pair direct	16	None	Direct	5	SHLD 2500 H
LDAX rp [A] ←[[rp]]	Load accumulator indirect	7	None	Register Indirect	2	LDAX B
STAX rp [[rp]] ←[A]	Store accumulator indirect	7	None	Register Indirect	2	STAX D
XCHG [H-L] ↔[D-E]	Change the contents of H-L with D-E pair	4	None	Register	1	

Arithmetic Group

The instructions of this group perform arithmetic operations such as addition, subtraction, increment or decrement of the content of a register or a memory.

Instruction Set	Explanation	States	Flags	Addre-ssing	Machine Cycles	Example
ADD r [A] ←[A]+[r]	Add register to accumulator	4	All	Register	1	ADD K
ADD M [A] ← [A] + [[H-L]]	Add memory to accumulator	7	All	Register indirect	2	ADD K
ACC r [A] ← [A] + [r] + [CS]	Add register with carry to accumulator	4	All	Register	1	ACC K

ADC M [A] ← [A] + [[H-L]] [CS]	Add memory with carry to accumulator	7	All	Register indirect	2	ADC K
ADI data [A] ← [A] + data	Add immediate data to accumulator	7	All	Immediate	2	ADI 55K
ACI data [A] ← [A] + data + [CS]	Add with carry immediate data to accumulator	7	All	Immediate	2	ACI 55K
DAD rp [H-L] ←[H-L] + [rp]	Add register paid to H-L pair	10	CS	Register	3	DAD K
SUB r [A] ←[A]-[r]	Subtract register from accumulator	4	All	Register	1	SUB K
SUB M [A] ← [A] - [[H-L]]	Subtract memory from accumulator	7	ALL	Register indirect	2	SUB K
SBB r [A] ←[A]-[H-L]] - [CS]	Subtract memory from accumulator with borrow	7	All	Register indirect	2	SBB K
SUI data [A] ←[A]-data	Subtract immediate data from accumulator	7	All	Immediate	2	SUI 55K
SBI data [A] ←[A]-data-[CS]	Subtract immediate data from accumulator with borrow	7	All	Immediate	2	XCHG
INR r [r] ←[r]+1	Increment register content	4	All except carry flag	Register	1	INR K
INR M [[H-L]] ←[[H-L]]+1	Increment memory content	10	All except carry	Register indirect	3	INR K

			flag			
DCR r [r] ←[r] -1	Decrement register content	4	All except carry flag	Register	1	DCR K
DCR M [[H-L]] ← [[H-L]]-1	Decrement memory content	10	All except carry flag	Register indirect	3	DCR K
INX rp [rp] ←[rp]+1	Increment memory content	6	None	Register	1	INX K
DCX rp [rp] ←[rp]-1	Decrement register pair	6	None	Register	1	DCX K
DAA	Decimal adjust accumulator	4			1	DAA

Logical Group

The instructions in this group perform logical operation such as AND, OR, compare, rotate, etc.

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
ANA r [A] ←[A]∧[r]	AND register with accumulator	4	All	Register	1
ANA M [A] ←[A]∧[[H-]]	AND memory with accumulator	4	All	Register indirect	2
ANI data [A] ← [A] ∧ [data]	AND immediate data with accumulator	7	All	Immediate	2
ORA r [A] ←[A]∨[r]	OR-register with accumulator	4	All	Register	1
ORA M [A] ←[A]∨[[H-L]]	OR-memory with accumulator	7	All	Register indirect	2
ORI data	OR -immediate data with	7	All	Immediate	2

[A] ← [A] ∨ [data]	accumulator				
XRA r [A] ← [A]∀[r]	XOR register with accumulator	4	All	Register	1
XRA M [A] ← [A] ∀ [[H-L]]	XOR memory with accumulator	7	All	Register indirect	2
XRI data [A] ←[A] ∀ [data]	XOR immediate data with accumulator	7	All	Immediate	2
CMA [A] ←[A]	Complement the accumulator	4	None	Implicit	1
CMC [CS] ←[CS]	Complement the carry status	4	CS		1
STC [CS] ← 1	Set carry status	4	CS		1
CMP r [A]-[r]	Compare register with accumulator	4	All	Register	1
CMP M [A] - [[H-L]]	Compare memory with accumulator	7	All	Register indirect	2
CPI data [A] - data	Compare immediate data with accumulator	7	All	Immediate	2
RLC $[A_{n+1}] \leftarrow [A^n]$, $[A^0] \leftarrow [A^7]$, [CS] ←[A^7]	Rotate accumulator left	4	Cs	Implicit	1
RRC $[A^7] \leftarrow [A^0]$, [CS] ←[A^0], $[A^n] \leftarrow [A^{n+1}]$	Rotate accumulator right		CS	Implicit	1
RAL $[A^{n+1}] \leftarrow [A^n]$, [CS] ←[$A^7$], $[A^0]$ ←[CS]	Rotate accumulator left through carry		CS	Implicit	1
RAR $[A^n] \leftarrow [A^{n+1}]$, [CS] ←[$A^0$], $[A^7]$ ←[CS]	Rotate accumulator right through carry		CS	Implicit	1

Branch Control Group

This group contains the instructions for conditional and unconditional jump, subroutine call and return, and restart.

Unconditional Jump

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
JMP addr(label) [PC] ← Label	Unconditional jump: jump to the instruction specified by the address	10	None	Immediate	3

Conditional Jump

Instruction Set	Explanation	States	Machine Cycles
Jump addr (label) [PC] ← Label	Conditional jump: jump to the instruction specified by the address if the specified condition is fulfilled	10, if true and 7, if not true	3, if true and 2, if not true

Instruction Set	Explanation	Status	States	Flags	Addressing	Machine Cycles
JZ addr (label) [PC] ← address (label)	Jump, if the result is zero	Jump if Z=1	7/10	None	Immediate	2/3
JNZ addr (label) [PC] ← address (label)	Jump if the result is not zero	Jump if Z=0	7/10	None	Immediate	2/3
JC addr (label) [PC] ← address (label)	Jump if there is a carry	Jump if CS =1	7/10	None	Immediate	2/3
JNC addr (label) [PC] ← address (label)	Jump if there is no carry	Jump if CS =0	7/10	None	Immediate	2/3
JP addr (label) [PC] ← address	Jump if result is	Jump if	7/10	None	Immediate	2/3

(label)	plus	S=0				
JM addr (label) [PC] ← address (label)	Jump if result is minus	Jump if S=1	7/10	None	Immediate	2/3
JPE addr (label) [PC] ← address (label)	Jump if even parity	The parity status P =1	7/10	None	Immediate	2/3
JPO addr (label) [PC] ← address (label)	Jump if odd parity	The parity status P =0	7/10	None	Immediate	2/3

1. Stack

- Stack is a group of memory location in the R/W memory that is used for temporary storage of binaryinformationduringexecutionofa program.
- The starting memory location of the stack is defined in program and space is reserved usually at the highendofmemorymap.
- The beginning of the stack is defined in the program by using instruction **LXI SP, 16-bit memory address**.Whichloadsa 16-bitmemoryaddressin stackpointerregisterofmicroprocessor.
- Once stack location is defined storing of data bytes begins at the memory address that is one less thenaddressinstackpointerregister.LXISP,2099hthestoringofdatabytesbeginsat2098Handcontinues inreversed numericalorder.

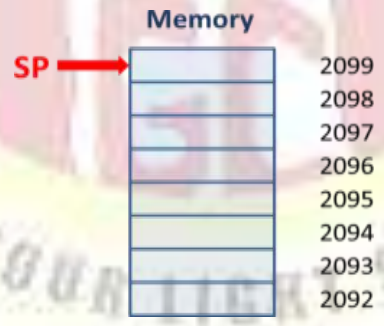


Fig.Stack

- Data bytes in register pair of microprocessor can be stored on the stack in reverse order by using thePUSHinstruction.
- PUSHB instructionsoredataofregisterpairBConsack.

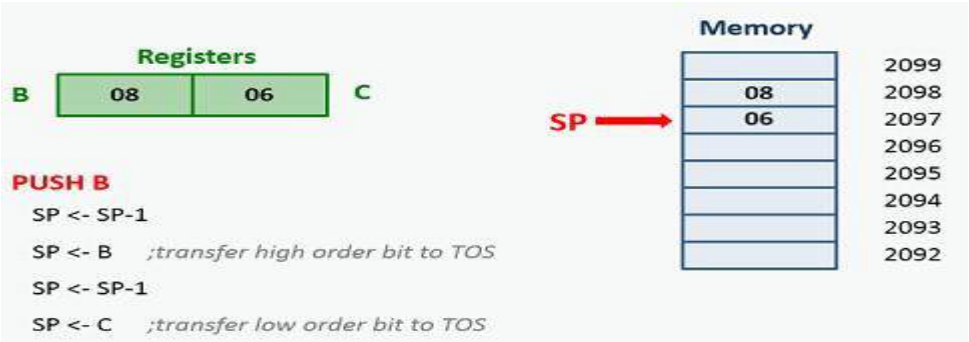


Fig.PUSHoperationonstack

- Databytescanbetransferredfromthestacktorespectiveregistersbyusinginstruction POP.

Pop operation

Instructionnecessaryforstackin8085

LXISP,2095	Loadthe stackpointerregisterwith a16-bitaddress.
PUSHB/D/H	ItcopiescontentsofB-C/D-E/H-L registerpaironthestack.
PUSHPSW	OperandPSWrepresentsProgramstatuswordmeaningcontentsofaccumulatorandflags.
POPB/D/H	Itcopies contentoftoptwomemory locationsofthestackintospecifiedregisterpair.
POPPSW	Itcopiescontentoftoptwomemorylocationsofthestackinto B-Caccumulatorandflags respectively.

2. Subroutine

- A subroutineisa groupofinstructionthat performs asubtaskof repeatedoccurrence.
- Asubroutinecanbeusedrepeatedlyindifferent locationsoftheprogram.

AdvantageofusingSubroutine

- Ratherthanrepeatthesameinstructionsseveraltimes,theycanbegroupedintoasubroutinethatiscalledfromthedifferentlocations.

WheretowriteSubroutine?

- InAssemblylanguage, asubroutinecanexistanywhere inthecode.
- However,itiscustomarytoplace subroutinesseparatelyfrom themainprogram.

Instructionsfordealingwithsubroutinesin8085.

- The**CALL**instructionisusedtoredirectprogramexecutiontothesubroutine.
  - WhenCALLinstructionisfetched,theMicroprocessorknowsthatthenexttwo**new**Memorylocationcontains16bitsubroutineaddress.
  - Microprocessor Reads the subroutine address from the next two memory location and stores thehigher order 8bit of the address in the **W** register and stores the lower order 8bit of the address inthe**Z** register.
  - Push the **Older** address of the instruction immediately following the CALL onto the stack [Returnaddress]
  - Loadstheprogramcounter(**PC**)withthe**new**16-bitaddresssupplied withtheCALL instructionfrom**WZ**register.The **RET** instructionisusedto return.

Unconditional CALL

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
CALL addr (label) [SP]-1] ← [PCH] ,[[SP-2] ← [PCL], [SP] ← [SP]-2, [PC] ← addr(label)	*Unconditional CALL: Call the subroutine identified by the address*	18	None	Immediate /register	5

Conditional CALL

Instruction Set	Explanation	States	Machine Cycles
CALL addr (label) [SP]-1] ← [PCH] , [[SP-2] ← [PCL], [PC] ← addr (label), [SP] ← [SP]-2	Unconditional CALL: Call the subroutine identified by the address if the specified condition is fulfilled	18, if true and 9, if not true	5, if true and 2, if not true

Instruction Set	Explanation	Status	States	Flags	Addressing	Machine Cycles
CC addr(label)	Call subroutine if carry status CS=1	CS =1	9/18	None	Immediate /register	2/5
CNC addr (label)	Call subroutine if carry status CS=0	CS =0	9/18	None	Immediate /register	2/5
CZ addr (label)	Call Subroutine if the result is zero	Zero status Z=1	9/18	None	Immediate /register	2/5
CNZ addr (label)	Call Subroutine if the result is not zero	Zero status Z=0	9/18	None	Immediate /register	2/5
CP addr (label)	Call Subroutine if the result is plus	Sign status S=0	9/18	None	Immediate /register	2/5
CM addr (label)	Call Subroutine if the result is minus	Sign status S= 1	9/18	None	Immediate /register	2/5
CPE addr(label)	Call subroutine if even	Parity Status P=1	9/18	None	Immediate /register	2/5

	parity					
CPO addr(label)	Call subroutine if odd parity	Parity Status P= 0	9/18	None	Immediate /register	2/5

Unconditional Return

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
RET [PCL] ← [[SP]], [PCH] ← [[SP] + 1], [SP] ← [SP] + 2	Unconditional RET: Return from subroutine	10	None	Indirect	3

Conditional Return

Instruction Set	Explanation	States	Machine Cycles
RET [PCL] ← [[SP]], [PCH] ← [[SP] + 1], [SP] ← [SP] + 2	Conditional RET: Return from subroutine	12, if true and 6, if not true	3, if true and 1, if not true

Instruction Set	Explanation	Status	States	Flags	Addressing	Machine Cycles
RC	Return from subroutine if carry status is zero.	CS =1	6/12	None	Register indirect	1/3
RNC	Return from subroutine if carry status is not zero.	CS = 0	6/12	None	Register indirect	1/3
RZ	Return from subroutine if result is zero.	Zero status Z=1	6/12	None	Register indirect	1/3
RNZ	Return from subroutine if result is not zero.	Zero status Z= 0	6/12	None	Register indirect	1/3
RP	Return from subroutine if result is not plus.	Sign Status S= 0	6/12	None	Register indirect	1/3
RM	Return from subroutine if result is not minus.	Sign Status S= 0	6/12	None	Register indirect	1/3
RPE	Return from subroutine if even parity.	Parity Status P= 1	6/12	None	Register indirect	1/3
RPO	Return from subroutine if odd parity.	Parity Status P= 1	6/12	None	Register indirect	1/3

Restart

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
RST [[SP]-1] ← [PCH], [[SP]-2] ← [PCL], [SP] ← [SP] - 2, [PC] ← 8 times n	Restart is a one word CALL instruction.	12	None	Register Indirect	3

The restart instructions and locations are as follows:

Instruction	Opcode	Restart Locations
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

PCHL

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
PCHL [PC] ← [H-L], [PCH] ←[H], **[PCL] ←[L]**	Jump address specified by H-L pair	6	None	Register	1

Stack, I/O and Machine Control Group

This group contains the instructions for input/output ports, stack and machine control.

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
IN port - address [A] ← [Port]	Input to accumulator from I/O port	10	None	Direct	3
OUT port-address [Port] ← [A]	Output from accumulator to I/O port	10	None	Direct	3
PUSH rp [[SP] - 1] ← [rh], [[SP] - 2] ← [rh], [SP] ← [SP] - 2	Push the content of register pair to stack	12	None	Register(source)/register Indirect(destination)	3
PUSH PSW [SP]-1] ← [A], [[SP] -2] ← PSW, [SP] ← [SP] - 2	Push processor word	12	None	Register(source)/register Indirect(destination)	3
POP rp [rl] ← [[SP]], [rh] ← [[SP]+1], [SP] ← [SP] + 2	Pop the content of register pair, which was saved, from the stack	10	None	Register(source)/register Indirect(destination)	3
HLT	Halt	5	None		1

XTHL [L] ↔ [[SP]], [H] ↔ [[SP] + 1]	Exchange top stack with H-L	16	None	Register indirect	5
SPHL [H-L] → [SP]	Moves the contents of H-L pair to stack pointer	6	None	Register	1
EI	Enable Interrupts	4	None		1
SIM	Set Interrupts Masks	4	None		1
RIM	Read Interrupts Masks	4	None		1
NOP	No Operation	4	None		1

8085 Addressing Modes

The way in which for specifying operands are called the addressing modes. For 8085, they are

- **Immediate Addressing:**
  - Data is provided in the instruction.
  - Load the immediate data to the destination provided.
  - Example: MVI A, 12 H
- **Register Addressing:**
  - Data is provided through the registers.
  - Example: MOV B, C
- **Direct Addressing:**
  - Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device.
  - Example: MOV A, [1000]

- **Indirect Addressing:**
  - Effective address is calculated by the processor and the contents of the address is used to form a second address. The second address is where the data is stored.
  - Example: MOV A, [[1000]]
- **Implicit addressing:**
  - In this addressing mode the data itself specifies the data to be operated upon.
  - Example: CMA ; Complement the contents of accumulator

UNIT II

Timing Diagrams of 8085

It is one of the best way to understand to process of micro-processor/controller. With the help of timing diagram we can understand the working of any system, step by step working of each instruction and its execution, etc.

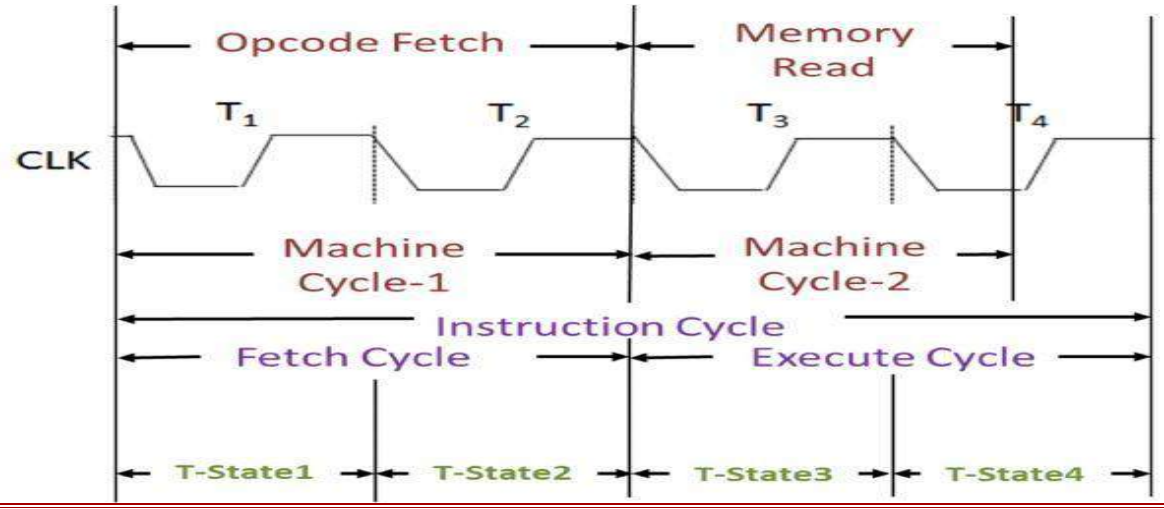
It is the graphical representation of process in steps with respect to time. The timing diagram represents the clock cycle and duration, delay, content of address bus and data bus, type of operation ie. Read/write/status signals.

Important terms related to timing diagrams:

T-state: Each clock cycle is called as T-states.

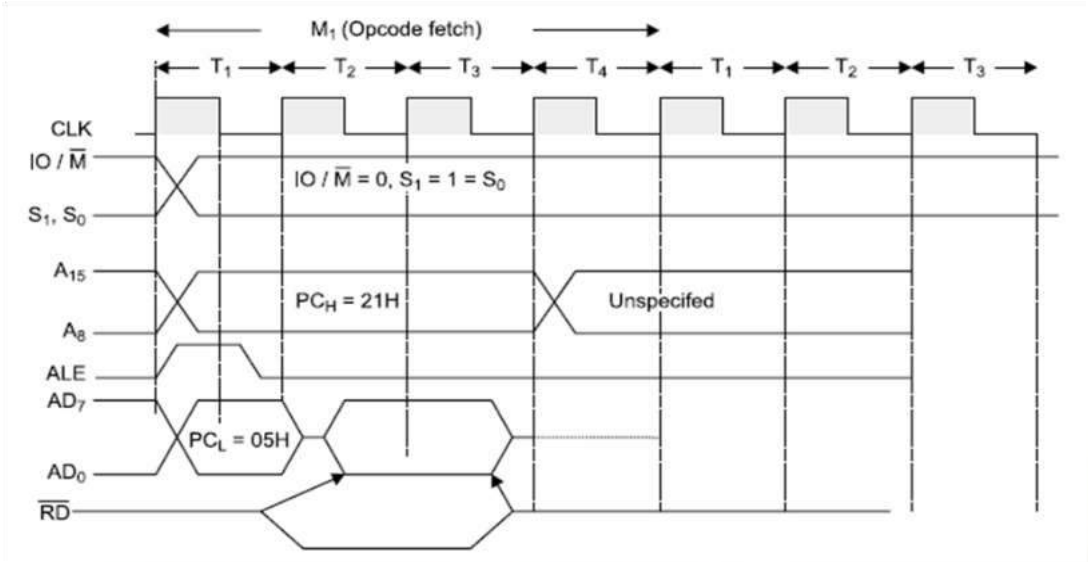
Machine cycle: It is the time required by the microprocessor to complete the one portion of an operation

Instruction cycle: this term is defined as the number of steps required by the CPU to complete the entire process i.e. Fetching and execution of one instruction. The fetch and execute cycles are carried out in synchronization with the clock.



Opcode fetch:

The microprocessor requires instructions to perform any particular action. In order to perform these actions microprocessor utilizes Opcode which is a part of an instruction which provides detail (ie. Which operation µp needs to perform) to microprocessor.



Operation:

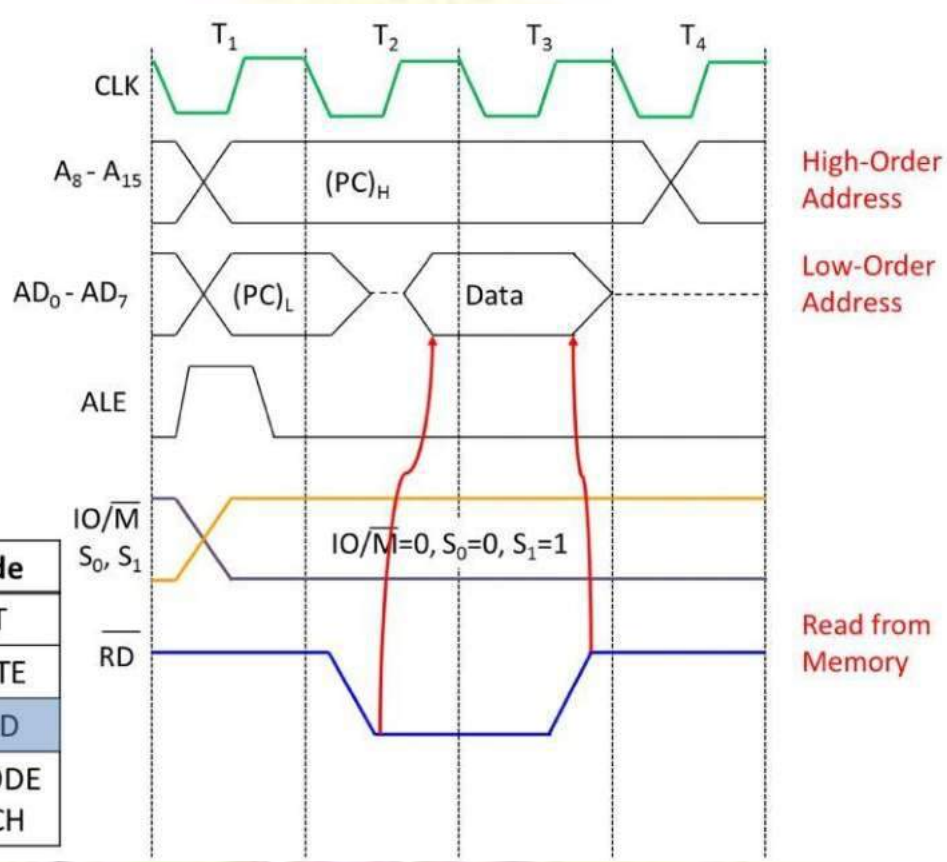
- During T1 state, microprocessor uses IO/M(bar), S0, S1 signals are used to instruct microprocessor to fetch opcode.
- Thus when IO/M(bar)=0, S0=S1= 1, it indicates opcode fetch operation.
- During this operation 8085 transmits 16-bit address and also uses ALE signal for address latching.
- At T2 state microprocessor uses read signal and make data ready from that memory location to read opcode from memory and at the same time program counter increments by 1 and points next instruction to be fetched.
- In this state microprocessor also checks READY input signal, if this pin is at low logic level ie. '0' then microprocessor adds wait state immediately between T2 and T3.
- At T3, microprocessor reads opcode and store it into instruction register to decode it further.

- During T4 microprocessor performs internal operation like decoding opcode and providing necessary actions.

- The opcode is decoded to know whether T5 or T6 states are required, if they are not required then µp performs next operation.

Read and write timing diagram for memory and I/O Operation

MemoryRead:



S_1	S_0	Mode
0	0	HLT
0	1	WRITE
1	0	READ
1	1	OPCODE FETCH

<>Figure: Memory read timing diagram

Operation:

It is used to fetch one byte from the memory.

It requires 3 T-States.

It can be used to fetch operand or data from the memory.

During T1, A8-A15 contains higher byte of address. At the same time ALE is high. Therefore Lower byte of address A0-A7 is selected from AD0-AD7.

Since it is memory ready operation, IO/M(bar) goes low.

During T2 ALE goes low, RD(bar) goes low. Address is removed from AD0-AD7 and data

D0-D7 appears on AD0-AD7.

During T3, Data remains on AD0-AD7 till RD(bar) is at low signal.

Memory Write: *Figure: Memory write timing diagram*

Operation:

It is used to send one byte into memory.

It requires 3 T-States.

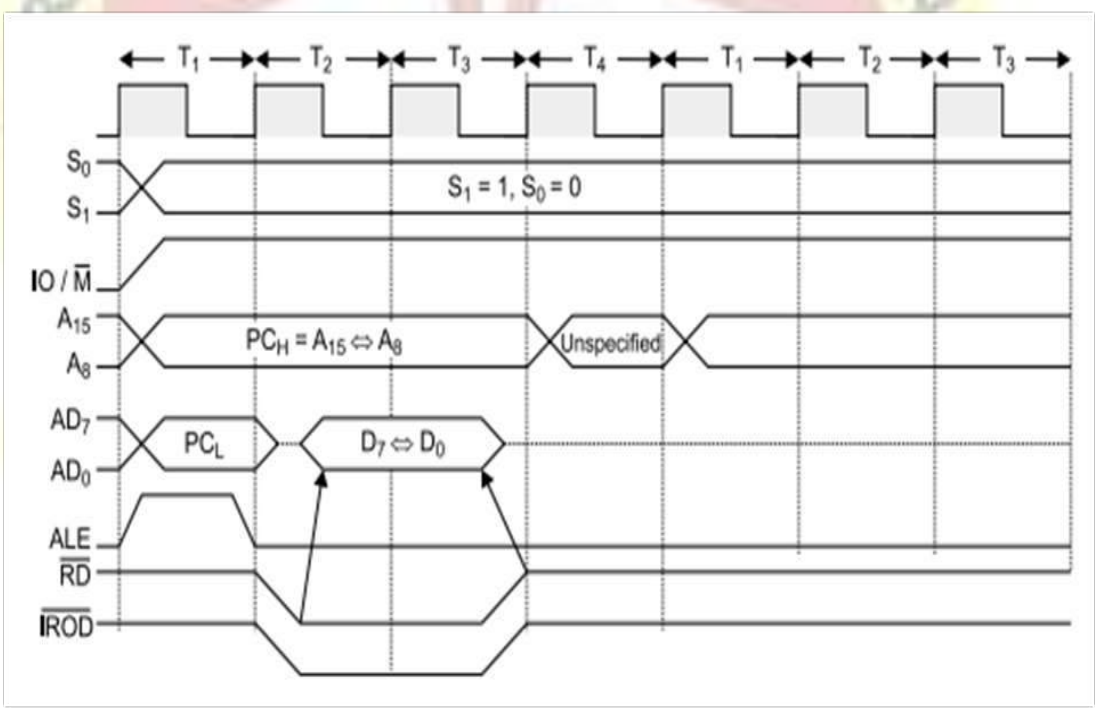
During T1, ALE is high and contains lower address A0-A7 from AD0-AD7.

A8-A15 contains higher byte of address.

As it is memory operation, IO/M(bar) goes low.

During T2, ALE goes low, WR(bar) goes low and Address is removed from AD0-AD7 and then data appears on AD0-AD7.

Data remains on AD0-AD7 till WR(bar) is low.



I/ORead:

Figure: I/O read timing diagram

Operation:

It is used to fetch one byte from an IO port.

It requires 3 T-States.

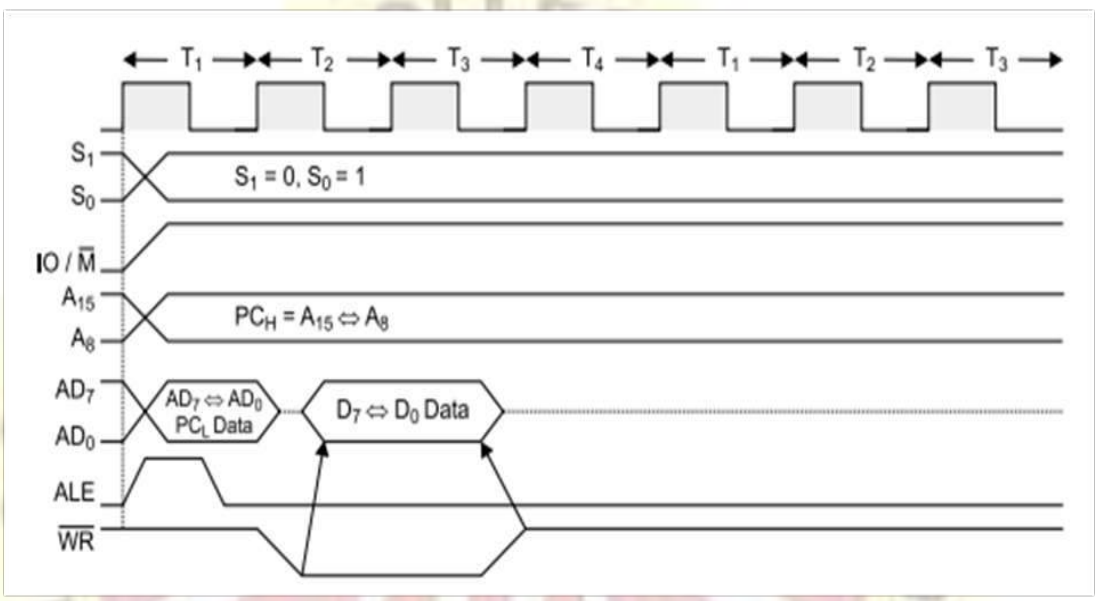
During T1, The Lower Byte of IO address is duplicated into higher order address bus A8-A15.

ALE is high and AD0-AD7 contains address of IO device.

IO/M (bar) goes high as it is an IO operation.

During T2, ALE goes low, RD (bar) goes low and data appears on AD0-AD7 as input from IO device.

During T3 Data remains on AD0-AD7 till RD(bar) is low.



I/OWrite:

Operation:

It is used to writ one byte into IO device.

It requires 3 T-States.

During T1, the lower byte of address is duplicated into higher order address bus A8-A15.

ALE is high and A0-A7 address is selected from AD0-AD7.

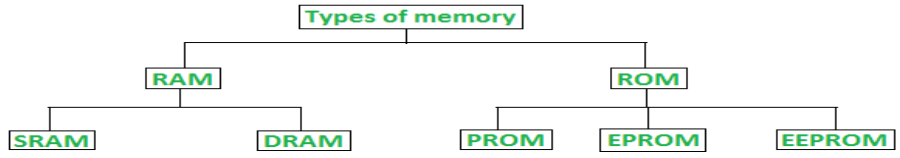
As it is an IO operation IO/M (bar) goes low.

During T2, ALE goes low, WR (bar) goes low and data appears on AD0-AD7 to write data into IO device.

During T3, Data remains on AD0-AD7 till WR(bar) is low.

ClassificationofMemory

Memory is the most essential element of a computing system because without it computer can't perform simple tasks. Computer memory is of two basic type – Primary memory(RAM and ROM) and Secondary memory(hard drive,CD,etc.). Random Access Memory (RAM) is primary-volatile memory and Read Only Memory (ROM) is primary-non-volatile memory.



Classification of computer memory

Random Access Memory (RAM) –

It is also called as read write memory or the main memory or the primary memory.

The programs and data that the CPU requires during execution of a program are stored in this memory.

It is a volatile memory as the data loses when the power is turned off.

RAM is further classified into two types- SRAM (Static Random Access Memory) and DRAM (Dynamic Random Access Memory).

DRAM	SRAM
1. Constructed of tiny capacitors that leak electricity.	1.Constructed of circuits similar to D flip-flops.
2.Requires a recharge every few milliseconds to maintain its data.	2.Holds its contents as long as power is available.
3.Inexpensive.	3.Expensive.
4. Slower than SRAM.	4. Faster than DRAM.
5. Can store many bits per chip.	5. Can not store many bits per chip.
6. Uses less power.	6.Uses more power.
7.Generates less heat.	7.Generates more heat.
8. Used for main memory.	8. Used for cache.

Difference between SRAM and DRAM

2. Read Only Memory (ROM) –

Stores crucial information essential to operate the system, like the program essential to boot the computer.

It is not volatile.

Always retains its data.

Used in embedded systems or where the programming needs no change.

Used in calculators and peripheral devices.

ROM is further classified into 4 types- ROM, PROM, EPROM, and EEPROM.

Types of Read Only Memory (ROM) –

PROM (Programmable read-only memory) – It can be programmed by user. Once programmed, the data and instructions in it cannot be changed.

EPROM (Erasable Programmable read only memory) – It can be reprogrammed. To erase data from it, expose it to ultra violet light. To reprogram it, erase all the previous data.

EEPROM (Electrically erasable programmable read only memory) – The data can be erased by applying electric field, no need of ultra violet light. We can erase only portions of the chip

RAM	ROM
1. Temporary Storage.	1. Permanent storage.
2. Store data in MBs.	2. Store data in GBs.
3. Volatile.	3. Non-volatile.
4.Used in normal operations.	4. Used for startup process of computer.
5. Writing data is faster.	5. Writing data is slower.

Difference between RAM and ROM

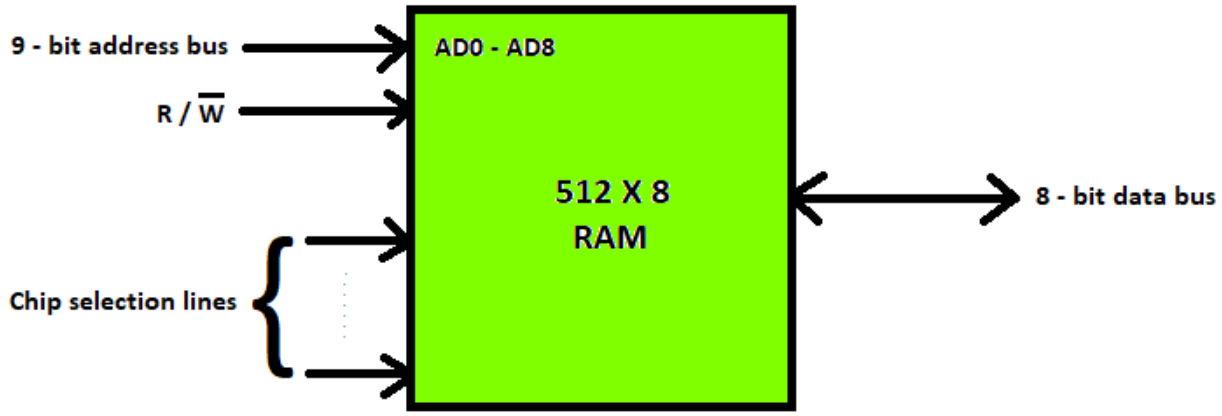
RAM(Random Access Memory) is a part of computer's Main Memory which is directly accessible by CPU. RAM is used to Read and Write data into it which is accessed by CPU randomly. RAM is volatile in nature, it means if the power goes off, the stored information is lost. RAM is used to store the data that is currently processed by the CPU. Most of the programs and data that are modifiable are stored in RAM.

Integrated RAM chips are available in two form:

SRAM(Static RAM)
DRAM(Dynamic RAM)

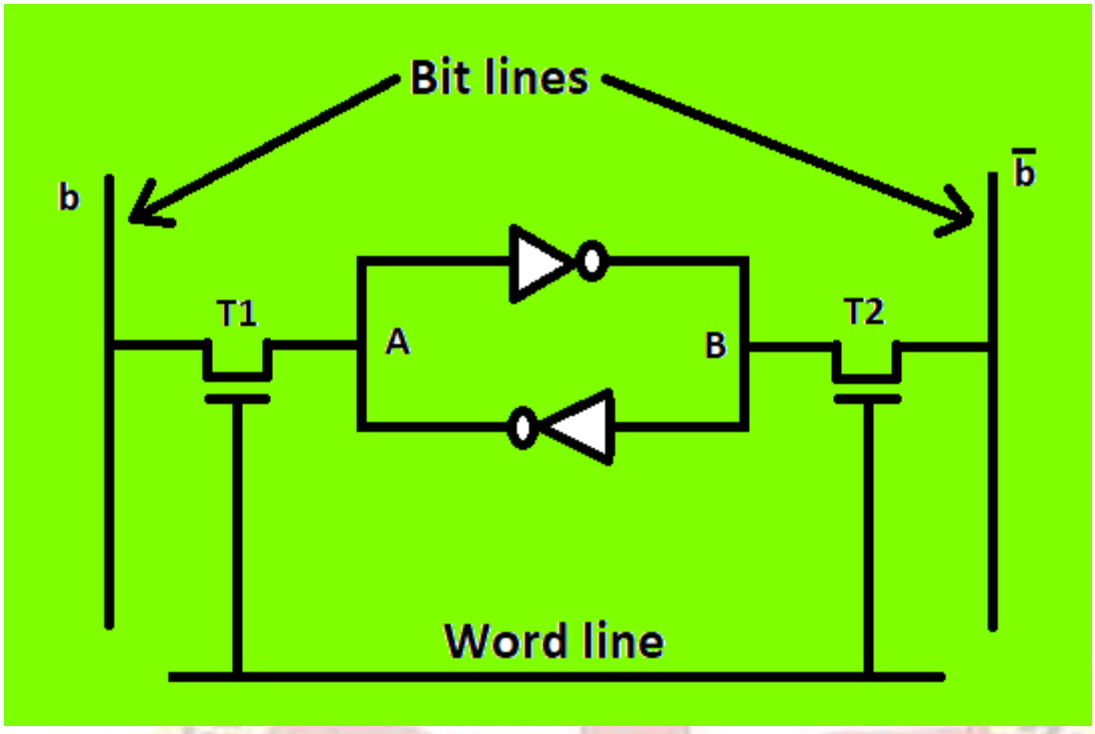
SRAM

The block diagram of SRAM chip is given below.



The SRAM memories consist of circuits capable of retaining the stored information as long as the power is applied. That means this type of memory requires constant power. SRAM memories are used to build Cache Memory.

SRAM Memory Cell: Static memories(SRAM) are memories that consist of circuits capable of retaining their state as long as power is on. Thus this type of memories is called volatile memories. The below figure shows a cell diagram of SRAM. A latch is formed by two inverters connected as shown in the figure. Two transistors T1 and T2 are used for connecting the latch with two bit lines. The purpose of these transistors is to act as switches that can be opened or closed under the control of the word line, which is controlled by the address decoder. When the word line is at 0-level, the transistors are turned off and the latch remains its information. For example, the cell is at state 1 if the logic value at point A is 1 and at point B is 0. This state is retained as long as the word line is not activated.



For **Read operation**, the word line is activated by the address input to the address decoder. The activated word line closes both the transistors (switches) T1 and T2. Then the bit values at points A and B can transmit to their respective bit lines. The sense/write circuit at the end of the bit lines sends the output to the processor.

For **Write operation**, the address provided to the decoder activates the word line to close both the switches. Then the bit value that to be written into the cell is provided through the sense/write circuit and the signals in bit lines are then stored in the cell.

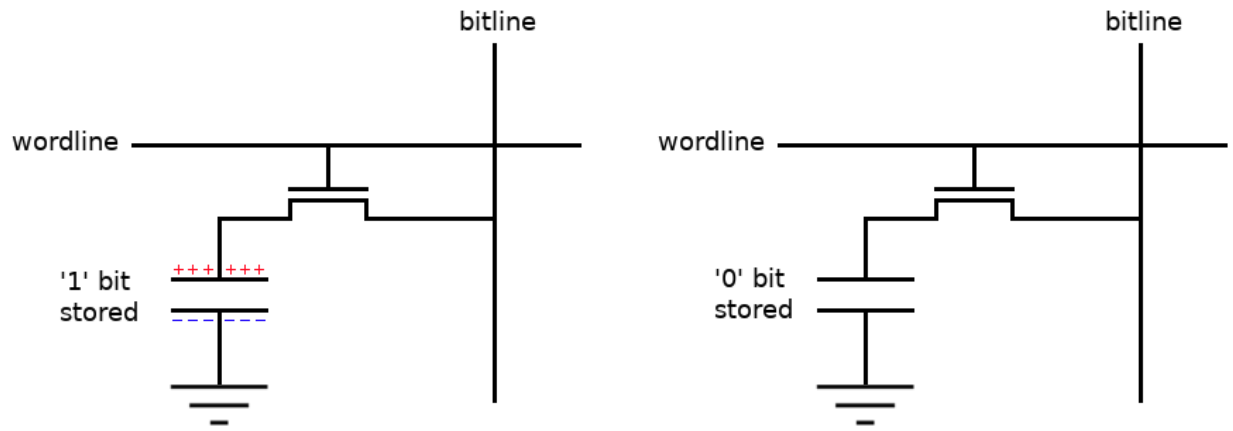
DRAM

Dynamic random-access memory, or DRAM, is a specific type of random access memory that allows for higher densities at a lower cost. The memory modules found in laptops and desktops uses DRAM.

How Does DRAM Work?

Invented by Robert Dennard in 1966 at IBM, DRAM works much differently than other types of memory. The fundamental storage cell within DRAM is composed of two elements: a transistor and a capacitor.

When a bit needs to be put in memory, the transistor is used to charge or discharge the capacitor. A charged capacitor represents a logic high, or '1', while a discharged capacitor represents a logic low, or '0'. The charging/discharging is done via the wordline and bitline, shown in Figure 1.

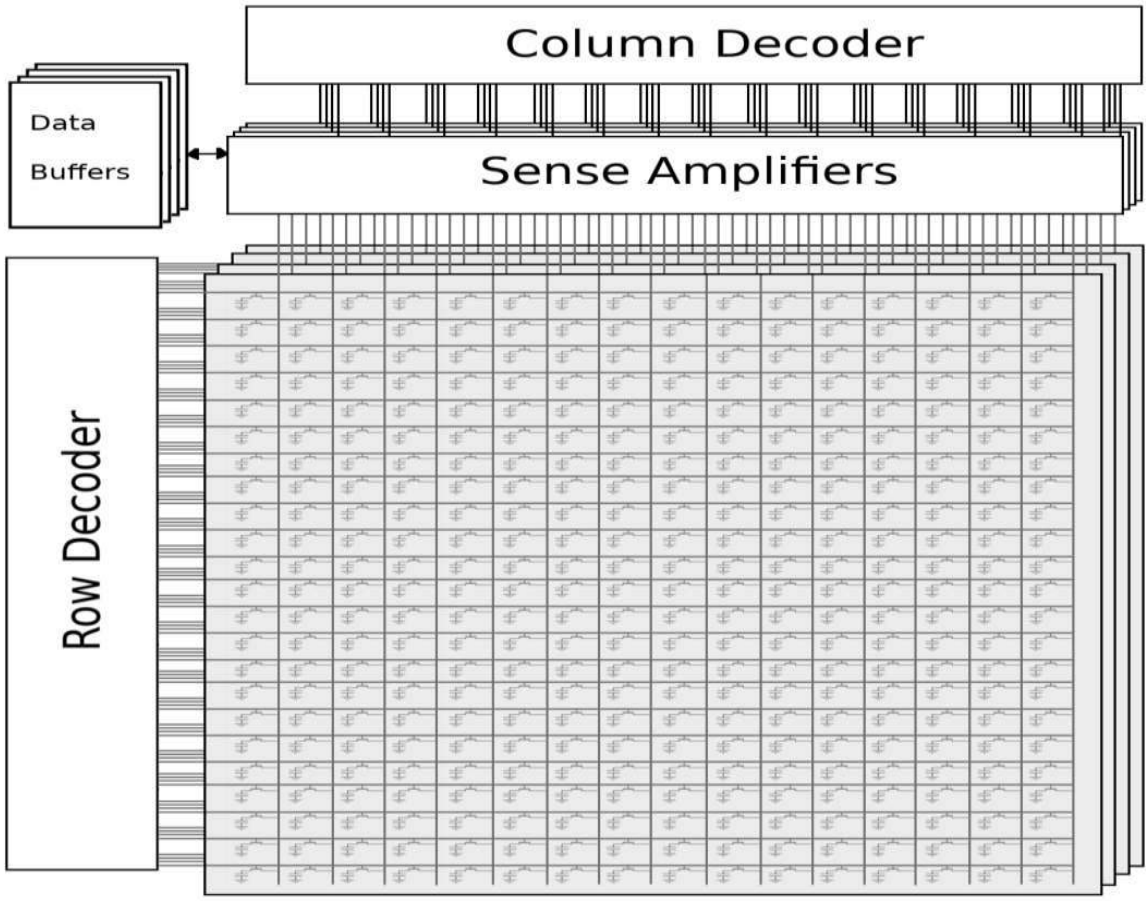


During a read or write, the wordline goes high and the transistor connects the capacitor to the bitline. Whatever value is on the bitline ('1' or '0') gets stored or retrieved from the capacitor.
The charge stored on each capacitor is too small to be read directly and is instead measured by a circuit called a sense amplifier. The sense amplifier detects the minute differences in charge and outputs the corresponding logic level. The act of reading from the bitline forces the charge to flow

out of the capacitor. Thus, in DRAM, reads are destructive. To get around this, an operation known as precharging is done to put the value read from the bitline back into the capacitor.
Equally problematic is the fact that the capacitors leak charge over time. Therefore, to maintain the data stored in memory the capacitors must be refreshed periodically. Refreshing works just like a read and ensures data is never lost. This is where DRAM gets the “Dynamic” moniker from—the charge on a DRAM cell is dynamically refreshed every so often. Contrast this with SRAM (Static RAM) which retains its state without needing to be refreshed.

Rank, Bank, Row, and Column

As mentioned earlier, the rank of a DRAM is a set of separately addressable DRAM chips. Each DRAM chip is further organized into a number of banks that contain a set of memory arrays. The number of memory arrays per bank is equal to the size of the output width. Therefore in a x4 DRAM chip, the internal banks would each have four memory arrays.



The Gray section is the memory array designed as a grid of rows and columns. A set of decoders are used to access the rows and columns, selecting a single intersection within the memory array. It is at this intersection that a small capacitor stores a charge representing the data being accessed.

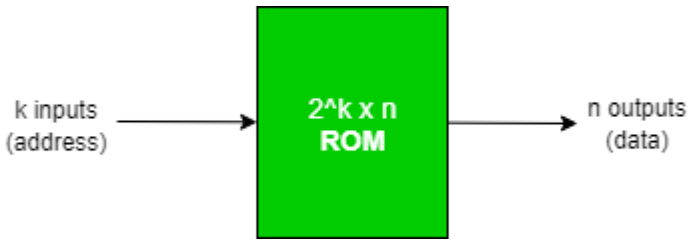
Sense amplifiers perform precharge operations on capacitors and generate logic-level outputs for a number of data buffers that store the data until it can be retrieved by a memory controller or CPU.

Classification and Programming of Read-Only Memory (ROM)

Read-Only Memory (ROM) is the primary memory unit of any computer system along with the Random Access Memory (RAM), but unlike RAM, in ROM, the binary information is stored permanently . Now, this information to be stored is provided by the designer and is then stored inside the ROM . Once, it is stored, it remains within the unit, even when power is turned off and on again .

The information is embedded in the ROM, in the form of bits, by a process known as programming the ROM . Here, programming is used to refer to the hardware procedure which

specifies the bits that are going to be inserted in the hardware configuration of the device . And this is what makes ROM a Programmable Logic Device (PLD) .



Block Structure

It consists of k input lines and n output lines .

The k input lines is used to take the input address from where we want to access the content of the ROM .

Since each of the k input lines can be either 0 or 1, so there are 2^k total addresses which can be referred to by these input lines and each of these addresses contain n bit information, which is given out as the output of the ROM.

Such a ROM is specified as 2^k x n ROM .

Internal Structure

It consists of two basic components – Decoder and OR gates .

A Decoder is a combinational circuit which is used to decode any encoded form (such as binary, BCD) to a more known form (such as decimal form) .

In ROM, the input to a decoder will be in binary form and the output will represent its decimal equivalent .

The Decoder is represented as l x 2^l, that is, it has l inputs and has 2^l outputs, which implies that it will take l-bit binary number and decode it into one of the 2^l decimal number .

All the OR gates present in the ROM will have outputs of the decoder as their input .

Classification Of ROM

PROM – It stands for Programmable Read-Only Memory . It is first prepared as blank memory, and then it is programmed to store the information . The difference between PROM and Mask ROM is that PROM is manufactured as blank memory and programmed after manufacturing, whereas a Mask ROM is programmed during the manufacturing process.

To program the PROM, a PROM programmer or PROM burner is used . The process of programming the PROM is called as burning the PROM . Also, the data stored in it cannot be modified, so it is called as one – time programmable device.

Uses – They have several different applications, including cell phones, video game consoles, RFID tags, medical devices, and other electronics.

EPROM – It stands for Erasable Programmable Read-Only Memory . It overcomes the disadvantage of PROM that once programmed, the fixed pattern is permanent and cannot be altered . If a bit pattern has been established, the PROM becomes unusable, if the bit pattern has to be changed .

This problem has been overcome by the EPROM, as when the EPROM is placed under a special ultraviolet light for a length of time, the shortwave radiation makes the EPROM return to its initial state, which then can be programmed accordingly .Again for erasing the content, PROM programmer or PROM burner is used.

EEPROM – It stands for Electrically Erasable Programmable Read-Only Memory . It is similar to EPROM, except that in this, the EEPROM is returned to its initial state by application of an electrical signal, in place of ultraviolet light . Thus, it provides the ease of erasing, as this can be done, even if the memory is positioned in the computer. It erases or writes one byte of data at a time .

Uses – It is used for storing the computer system BIOS.

Flash ROM – It is an enhanced version of EEPROM .The difference between EEPROM and Flash ROM is that in EEPROM, only 1 byte of data can be deleted or written at a particular time, whereas, in flash memory, blocks of data (usually 512 bytes) can be deleted or written at a particular time . So, Flash ROM is much faster than EEPROM .

Uses – Many modern PCs have their BIOS stored on a flash memory chip, called as flash BIOS and they are also used in modems as well.

Programming the Read-Only Memory (ROM)

To understand how to program a ROM, consider a 4 x 4 ROM, which means that it has total of 4 addresses at which information is stored, and each of those addresses has a 4-bit information, which is permanent and must be given as the output, when we access a particular address . The following steps need to be performed to program the ROM –

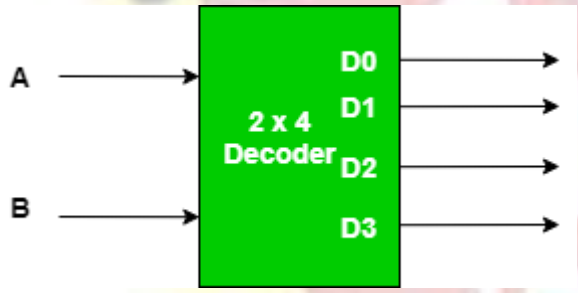
1. Construct a truth table, which would decide the content of each address of the ROM and based upon which a particular ROM will be programmed.

So, the truth table for the specification of the 4 x 4 ROM is described as below :

Inputs		Outputs			
X	Y	A	B	C	D
0	0	0	0	1	1
0	1	1	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1

1. This truth table shows that at location 00, content to be stored is 0011, at location 01, the content should be 1100, and so on, such that whenever a particular address is given as input, the content at that particular address is fetched . Since, with 2 input bits, 4 input combinations are possible and each of these combinations hold a 4-bit information, so this ROM is a 4 X 4 ROM .
2. Now, based upon the total no. of addresses in the ROM and the length of their content, decide the decoder as well as the no. of OR gates to be used . Generally, for a 2 x n ROM, a k x 2 decoder is used, and the total no. of OR gates is equal to the total no. of bits stored at each location in the ROM .

So, in this case, for a 4 x 4 ROM, the decoder to be used is a 2 x 4 decoder. The following is a 2 x 4 decoder –

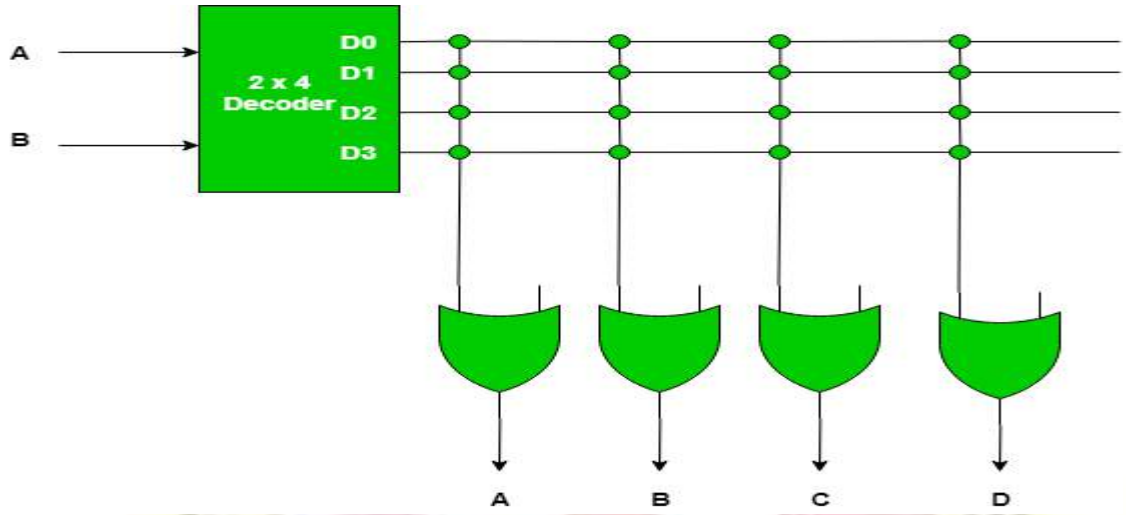


The truth table for a 2 x 4 decoder is as follows –

Inputs		Outputs			
A	B	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

When both the inputs are 0, then only D_0 is 1 and rest are 0, when input is 01, then, only D_1 is high and so on. (Just remember that if the input combination of the decoder resolves to a particular decimal number d, then at the output side the terminal which is at position d + 1 from the top will be 1 and rest will be 0).

Now, since we want each address to store 4 – bits in the 4 x 4 ROM, so, there will be 4 OR gates, with each of the 4 outputs of the decoder being input to each one of the 4 OR gates, whose output will be the output of the ROM, as follows

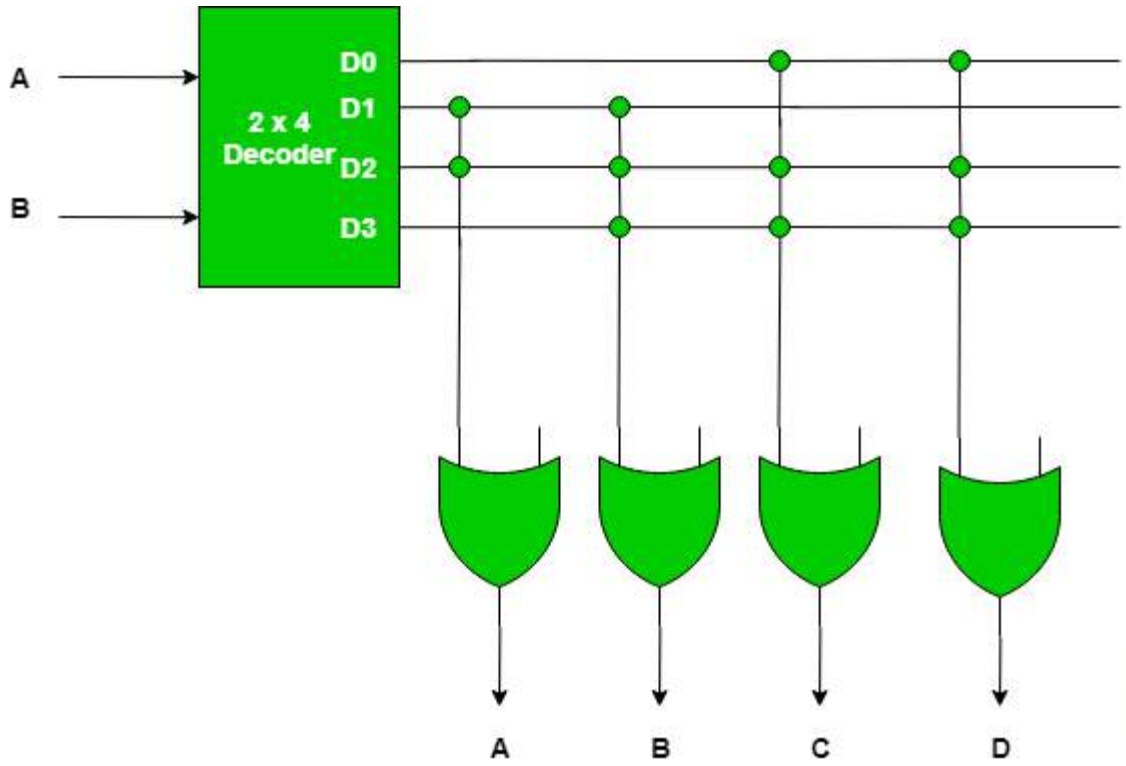


A cross sign in this figure shows connection between the two lines is intact . Now, since there are 4 OR gates and 4 output lines from the decoder, so there are total of 16 intersections, called as crosspoint .

Now, program the intersection between the two lines, as per the truth table, so that the output of the ROM (OR gates) is in accordance with the truth table .

For programming the crosspoints, initially all the crosspoints are left intact, which means that it is logically equivalent to a closed switch, but these intact connections can be blown by the application of a high – voltage pulse into these fuse, which will disconnect the two interconnected lines, and in this way the output of a ROM can be manipulated .

So, to program a ROM, just look at the truth table specifying the ROM and blow away (if required) a connection . The connections for the 4 x 4 ROM as per the truth table is as shown below –



Remember, a cross sign is used to denote that the connection is left intact and if there is no cross this means that there is no connection .

In this figure, since, as can be seen from the truth table specifying the ROM, when the input is 00, then, the output is 0011, so as we know from the truth table of a decoder, that input 00 gives output such that only D_0 is 1 and rest are 0, so to get output 0011 from the OR gates, the connections of D_0 with the first two OR gates has been blown away, to get the outputs as 0, while the last two OR gates give the output as 1, which is what is required .

Similarly, when the input is 01, then the output should be 1100, and with input 01, in decoder only D_1 is 1 and rest are 0, so to get the desired output the first two OR gates have their connection intact with D_1, while last two OR gates have their connection blown away . And for the rest also the same procedure is followed.

So, this is how a ROM is programmed and since, the output of these gates will remain constant every time, so that is how the information is stored permanently in the ROM, and does not get altered even on switching on and off .

Programming In 8085.

The memory addresses given in the program are for a particular microprocessor kit. These addresses can be changed to suit the microprocessor kit available in your system.

ADDITION OF TWO 8 BIT NUMBERS

AIM:

To perform addition of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

PROGRAM:

XRA A		Clear carry flag and Accumulator
MVI	C, 00	Initialize C register to 00
LDA	4150	Load the value to Accumulator.
MOV	B, A	Move the content of Accumulator to B register.
LDA	4151	Load the value to Accumulator.
ADD	B	Add the value of register B to A
JNC	LOOP	Jump on no carry.
INR	C	Increment value of register C
LOOP: STA	4152	Store the value of Accumulator (SUM).
MOV	A, C	Move content of register C to Acc.
STA	4153	Store the value of Accumulator (CARRY)
HLT		Halt the program.

RESULT:

Input:	FF (4150)
	03 (4151)
Output:	02 (4152)
	01 (4153)

SUBTRACTION OF TWO 8 BIT NUMBERS

AIM:To perform Subtraction of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Subtract the two register contents.
5) Check for carry.
6) Store the value of difference and borrow in memory location.
7) Terminate the program.

PROGRAM:

XRA A		Clear carry flag and Accumulator
MVI	C, 00	Initialize C register to 00
LDA	4150	Load the value to Accumulator.
MOV	B, A	Move the content of Accumulator to B register.
LDA	4151	Load the value to Accumulator.
SUB	B	Sub the value of register B to A
JNC	LOOP	Jump on no carry.
INR	C	Increment value of register C
LOOP: STA	4152	Store the value of Accumulator (difference).
MOV	A, C	Move content of register C to Acc.
STA	4153	Store the value of Accumulator (borrow)
HLT		Halt the program.

RESULT:

Input: 05(4150)
03(4151)

Output: 02 (4152)
0 (4153)

MULTIPLICATION OF TWO 8 BIT NUMBERS

AIM:

To perform the multiplication of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Increment the value of carry.
7) Check whether repeated addition is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

```
        MVI   D, 00     Initialize register D to 00
        MVI   A, 00     Initialize Accumulator content to 00
        LXI   H, 4150
        MOV   B, M      Get the first number in B - reg
        INX   H
        MOV   C, M      Get the second number in C- reg.
LOOP:   ADD   B         Add content of A - reg to register B.
        JNC   NEXT      Jump on no carry to NEXT.
        INR   D         Increment content of register D
NEXT:   DCR   C         Decrement content of register C.
        JNZ   LOOP      Jump on no zero to address
        STA   4152      Store the result in Memory
        MOV   A, D
        STA   4153      Store the MSB of result in Memory
        HLT             Terminate the program.
```

OBSERVATION:

Input:	FF (4150)
	FF (4151)
Output:	01 (4152)
	FE (4153)

RESULT:

Thus the program to multiply two 8-bit numbers was executed.

DIVISION OF TWO 8 BIT NUMBERS

AIM:

To perform the division of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register(B register).
3) Get the second data and load into Accumulator.
4) Compare the two numbers to check for carry.
5) Subtract the two numbers.
6) Increment the value of carry .
7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

	LXI	H, 4150	
	MOV	B, M	Get the dividend in B – reg.
	MVI	C, 00	Clear C – reg for qoutient
	INX	H	
	MOV	A, M	Get the divisor in A – reg.
NEXT:	CMP	B	Compare A - reg with register B.
	JC	LOOP	Jump on carry to LOOP
	SUB	B	Subtract A – reg from B- reg.
	INR	C	Increment content of register C.
	JMP	NEXT	Jump to NEXT
LOOP:	STA	4152	Store the remainder in Memory
	MOV	A, C	
	STA	4153	Store the quotient in memory
	HLT		Terminate the program.

OBSERVATION:

Input: FF (4150)
FF (4251)

Output: 01 (4152) ---- Remainder
FE (4153) ---- Quotient

RESULT:

Thus the program to divide two 8-bit numbers was executed.

LARGEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the largest number in an array of data using 8085 instruction set.

ALGORITHM:

1) Load the address of the first element of the array in HL pair
2) Move the count to B – reg.
3) Increment the pointer
4) Get the first data in A – reg.
5) Decrement the count.
6) Increment the pointer
7) Compare the content of memory addressed by HL pair with that of A - reg.
8) If Carry = 0, go to step 10 or if Carry = 1 go to step 9
9) Move the content of memory addressed by HL to A – reg.
10) Decrement the count
11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
12) Store the largest data in memory.
13) Terminate the program.

PROGRAM:

	LXI	H,4200	Set pointer for array
	MOV	B,M	Load the Count
	INX	H	
	MOV	A,M	Set 1st element as largest data
	DCR	B	Decrement the count
LOOP:	INX	H	
	CMP	M	If A- reg > M go to AHEAD
	JNC	AHEAD	
	MOV	A,M	Set the new value as largest
AHEAD:	DCR	B	
	JNZ	LOOP	Repeat comparisons till count = 0
	STA	4300	Store the largest value at 4300
	HLT		

OBSERVATION:

Input: 05 (4200) ----- Array Size
0A (4201)
F1 (4202)
1F (4203)
26 (4204)
FE (4205)

Output: FE (4300)

RESULT:

Thus the program to find the largest number in an array of data was executed

ARRANGE AN ARRAY OF DATA IN ASCENDING ORDER

AIM:

To write a program to arrange an array of data in ascending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

```
        LXI     H,4200
        MOV     C,M
        DCR     C
REPEAT: MOV     D,C
        LXI     H,4201
LOOP:   MOV     A,M
        INX     H
        CMP     M
        JC      SKIP
        MOV     B,M
        MOV     M,A
        DCX     H
        MOV     M,B
        INX     H
SKIP:   DCR     D
        JNZ     LOOP
        DCR     C
        JNZ     REPEAT
        HLT
```

OBSERVATION:

Input:	4200	05 (Array Size)
	4201	05
	4202	04
	4203	03
	4204	02
	4205	01

Output:	4200	05(Array Size)
	4201	01
	4202	02
	4203	03
	4204	04
	4205	05

RESULT:

Thus the given array of data was arranged in ascending order.

ASCII TO HEX CONVERSION

AIM:

To convert given ASCII Character into its equivalent Hexa Decimal number using 8085 instruction set.

ALGORITHM:

1. Load the given data in A- register
2. Subtract 30 H from A – register
3. Compare the content of A – register with 0A H
4. If A < 0A H, jump to step6. Else proceed to next step.
5. Subtract 07 H from A – register
6. Store the result
7. Terminate the program

PROGRAM:

```
        LDA   4500
        SUI   30
        CPI   0A
        JC    SKIP
        SUI   07
SKIP:   STA   4501
        HLT
```

OBSERVATION:

Input:	4500	31
Output:	4501	0B

RESULT:

Thus the given ASCII character was converted into its equivalent Hexa Value.

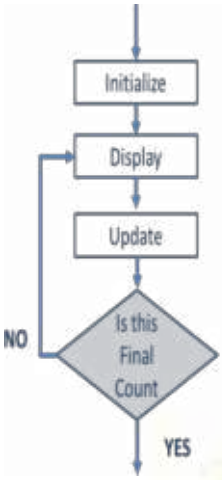
For Hex to ASCII

```
        LDA 4500
        CPI  0A
        JC  SKIP
        ADI 07
SKIP:   ADI  30
        STA 4501
        HLT
```

I/P 4500 09

O/P 4501 39

3. TimeDelay Calculation




```
1. MVI C,05
2. LOOP: MOV A,C
3.       OUT 01
4.       DCR C
5.    JNZ LOOP
6. HLT
```

4. Eachinstructionpassesthroughdifferentcombinationsof Fetch,MemoryRead,andMemoryWrite cycles.

5. Knowingthecombinationsofcycles,onecancalculatehowlongsuchaninstructionwouldrequireto complete.

6. Itis countedintermsof numberofT–statesrequired.

7. Calculatingthistimewegenerate requiresoftwaredelay.

TimeDelayUsingSingleRegister

Label	**Opcode**	**Operand**	**Comment**	**T-states**
	MVI	C,05h	;LoadCounter	**7**
LOOP:	DCR	C	;Decrement Counter	**4**
	JNZ	LOOP	;Jumpback toDecr.C	**10/7**

MVIC05 Mchine Cycle: F+ R=2 T-States:4T+3T =7T	DCR C Mchine Cycle:F= 1 T-States:4T=4T	JNZLOOP (true) Mchine Cycle: F+ R+ R=3 T-States:4T+3T +3T=10T	JNZLOOP(false)Mchine Cycle: F+R=3 T-States:4T+3T =7T

- InstructionMVIC,05hrequires7T-Statestoexecute.Assuming,8085Microprocessorwith2MHzclockfrequency.Howmuchtimeitwilltaketoexecuteabove instruction?

  Clock frequency of the system (f)

  = 2 MHzClockperiod(T)=1/f=½

  *10-6=0.5µs

Timeto executeMVI =7T-states* 0.5µs

=3.5µs

- Nowtocalculatetimedelayinloop,wemustaccountfortheT-statesrequiredforeachinstruction,andforthenumberoftimesinstructionsareexecuted intheloop.Thereforthenexttwoinstructions:

  DCR: 4T-States

  JNZ: 10T-States

  14T-States

- Here, theloopisrepeatedfor5 times.
- TimedelayinloopTLwith2MHzclockfrequencyiscalculated as:

  TL=T* LoopT-sates*N10 --------- (1)

  TL=(0.5 *10^{-6}*14*5)

  =35 s

- Ifwewanttocalculatedelaymoreaccurately,weneedtoaccuratelycalculateexecutionofJNZinstructioni.e

  If**JNZ =true**,then**T-States=10**

  Elseif**JNZ=false**,then**T-States=7**

- Delaygeneratedbylastclock cycle:

  =3T*ClockPeriod

  = 3T*(1/2*10^{-6})

  =1.5 s

- Now,the accurateloopdelayis:

  TLA=TL- Delaygenerated bylast

  clockcycleTLA=35 s - 1.5 s

  TLA=33.5 s

- Now,tocalculatetotaltimedelay

  TotalDelay= Timetaken toexecute instructionoutside loop+Timetakentoexecuteloopinstructions

  $T_D = T_O + T_{LA}$

  =(7*0.5 s)+33.5 s

  =3.5 s+33.5 s

  =37 s

- Inmostofthecasewearegiventimedelayandneedtofindvalueofthecounterregisterwhichdeciden umberof timesloopexecute.
- Forexample:writeALP togenerate 37 µsdelaygiventhatclockfrequencyif2MHz.
- Singleregisterloopcangeneratesmalldelayonlyforlarge delaywe useothertechnique.

TimeDelayUsingaRegisterPair

- Time delay can be considerably increased by setting a loop and using a register pair with a 16-bit number(FFFFh).
- A16-bitisdecremented byusingDCXinstruction.
- ProblemwithDCX instruction isDCXinstructiondoesn't set**Zero**flag.
- Withouttestflag,Jumpinstructioncan'tcheckdesiredconditions.
- Additionaltechnique mustbe usedto setZeroflag.

Label	Opcode	Operand	Comment	T-states
	LXI	B,2384h	; LoadBCwith16-bit counter	**10**
LOOP:	DCX	B	;DecrementBC by1	**6**
	MOV	A, C	; PlacecontentsofCinA	**4**
	ORA	B	;ORBwithCtosetZero flag	**4**
	JNZ	LOOP	;ifresultnotequalto0,10/7jumpback to loop	**10/7**

- Here the loop includes four instruction:TotalT-States= 6T +4T+4T+10T

  =24T-states

- Theloopisrepeatedfor2384 htimes.
- Converting(2384)16intodecimal.

  2384h=$(2*16^3)$+ $(3*16^2)$ + $(8*16^1)$ +$(4*16^0)$

  =8192 +768+128+4=**9092**

- Clockfrequencyofthesystem(f)=2MHz
- Clockperiod(T)=1/f=½*10^{-6}=0.5 s
- Now, to find delay in the loopTL=T *LoopT-sates*N10

  =0.5*24 *9092

  =109104 s=109ms(withoutadjustinglastcycle)

DEBUGGING A MACHINE LEVEL PROGRAM

Debugging is the process of identifying and removing bug from software or program. It refers to identification of errors in the program logic, machine codes, and execution. It gives step by step information about the execution of code to identify the fault in the program.

Debugging of machine code: Translating the assembly language to machine code is similar to building a circuit from a schematic diagram. Debugging can help in determining:

- Values of register.
- Flow of program.
- Entry and exit point of a function.
- Entry into *if* or *else* statement.
- Looping of code.
- Calculation check.

Common sources of error:

- Selecting a wrong code
- Forgetting second or third byte of instruction
- Specifying wrong jump locations
- Not reversing the order of high and low bytes in a Jump instruction
- Writing memory addresses in decimal instead of hexadecimal
- Failure to clear accumulator when adding two numbers
- Failure to clear carry registers

- Failure to set flag before Jump instruction
- Specifying wrong memory address on Jump instruction
- Use of improper combination of rotate instructions

The debugging process is divided into two parts:

1. **Static Debugging:** It is similar to visual inspection of circuit board, it is done by a paper and pencil to check the flowchart and machine codes. It is used to the understanding of code logic and structure of program.
2. **Dynamic Debugging:** It involves observing the contents of register or output after execution of each instruction (in single step technique) or a group of instructions (in breakpoint technique).

   In a single board microprocessor, techniques and tools commonly used in dynamic debugging are:

❖ **Single Step:** This technique allows to execute one instruction at a time and observe the results of each instruction. Generally, this is build using hard-wired logic circuit. As we press the single step run key we will be able to observe the contents of register and memory location. This helps to spot:

- incorrect addresses
- incorrect jump location in loops
- incorrect data or missing codes

However, if there is large loop then single step debugging can be very tiring and time-consuming. So instead of running the loop *n* times, we can reduce the number of iteration to check the effectiveness of the loop. The single step technique is very useful for short programs.

❖ **Breakpoint:** The breakpoint facility is usually a software routine that allows users to execute a program in sections. The breakpoints can be set using RST instruction. When we push the Execute key, the program will be executed till the breakpoint. The registers can be examined for the expected result. With the breakpoint facility, isolate the segment of program with errors. Then that segment can be debugged using the single-step facility. It is usually used to check:

- Timing loop
- I/O section
- Interrupts

❖ **Register Examine:** The register examine key allows you to examine the contents of the microprocessor register. This technique is used in conjunction with either single-step or breakpoint facility.

UNIT IV
MEMORY INTERFACING

Memory Interfacing in 8085:

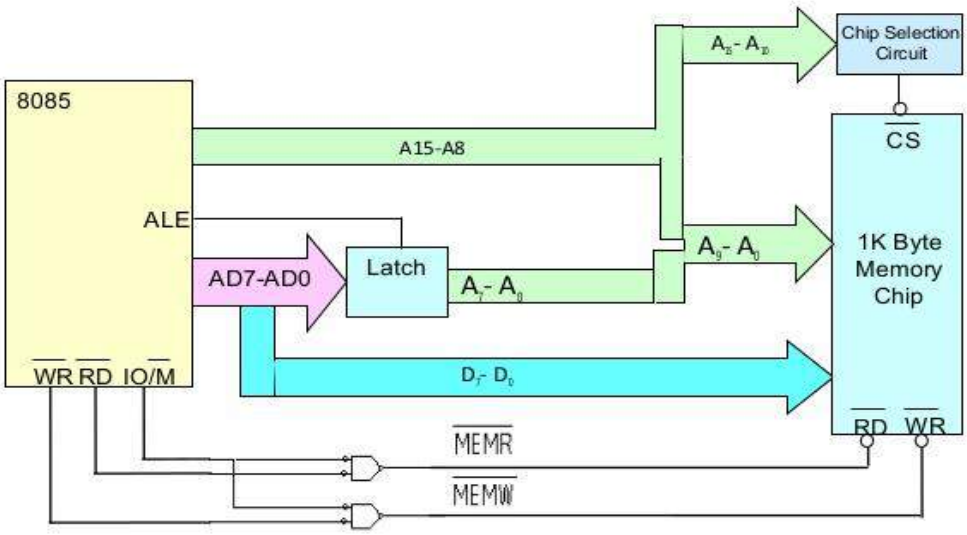
Memory is an integral part of a microprocessor system, and in this section, we will discuss how to interface a memory device with the microprocessor. The Memory Interfacing in 8085 is used to access memory quite frequently to read instruction codes and data stored in memory. This read/write operations are monitored by control signals. The microprocessor activates these signals when it wants to read from and write into memory. I

Basic Concepts in Memory Interfacing:

1. Microprocessor 8085 can access 64Kbytes memory since address bus is 16-bit. But it is not always necessary to use full 64Kbytes address space. The total memory size depends upon the application.
2. Generally, EPROM (or EPROMs) is used as a program memory and RAM (or RAMs) as a data memory. When both, EPROM and RAM are used, the total address space 64Kbytes is shared by them.
3. The capacity of program memory and data memory depends on the application.
4. It is not always necessary to select 1 EPROM and 1 RAM. We can have multiple EPROMs and multiple RAMs as per the requirement of application.
5. We can place EPROM/RAM anywhere in full 64 Kbytes address space. But program memory (EPROM) should be located from address 0000H since reset address of 8085 microprocessor is 0000H.
6. It is not always necessary to locate EPROM and RAM in consecutive memory For example : If the mapping of EPROM is from 0000H to OFFFH, it is not must to locate RAM from 1000H. We can locate it anywhere between 1000H and FFFFH. Where to locate memory component totally depends on the application.

The memory interfacing requires to :

- **Select the chip**
- **Identify the register**
- **Enable the appropriate buffer.**



25

INTERFACING MEMORY CHIPS WITH 8085

8085 has 16 address lines (A0 - A15), hence a maximum of 64 KB (= 216 bytes) of memory locations can be interfaced with it. The memory address space of the 8085 takes values from 0000H to FFFFH.

The 8085 initiates set of signals such as $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ when it wants to read from and write into memory. Similarly, each memory chip has signals such as $\overline{CE}$ or $\overline{CS}$ (chip enable or chip select), $\overline{OE}$ or $\overline{RD}$ (output enable or read) and $\overline{WE}$ or $\overline{WR}$ (write enable or write) associated with it.

Generation of Control Signals for Memory:

When the 8085 wants to read from and write into memory, it activates $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals as shown in Table 8.

Table 8 Status of $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals during memory read and write operations

$IO/\overline{M}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	1	8085 reads data from memory
0	1	0	8085 writes data into memory

Using $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals, two control signals $\overline{MEMR}$ (memory read) and $\overline{MEMW}$ (memory write) are generated. Fig. 16 shows the circuit used to generate these signals.

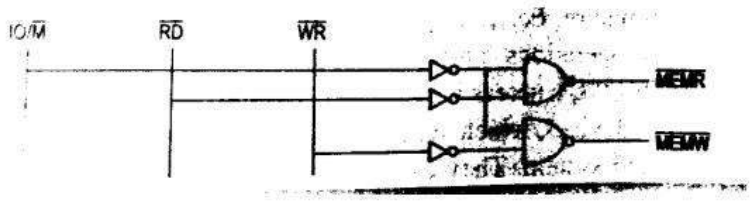


Fig. 16 Circuit used to generate $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ signals

When is $\text{IO}/\overline{\text{M}}$ high, both memory control signals are deactivated irrespective of the status of $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals.

Ex: Interface an IC 2764 with 8085 using NAND gate address decoder such that the address range allocated to the chip is 0000H – 1FFFH.

Specification of IC 2764:

- 8 KB (8×2^{10} byte) EPROM chip
- 13 address lines (2^{13} bytes = 8 KB)

Interfacing:

- 13 address lines of IC are connected to the corresponding address lines of 8085.
- Remaining address lines of 8085 are connected to address decoder formed using logic gates, the output of which is connected to the $\overline{\text{CE}}$ pin of IC.
- Address range allocated to the chip is shown in Table 9.
- Chip is enabled whenever the 8085 places an address allocated to EPROM chip in the address bus. This is shown in Fig. 17.

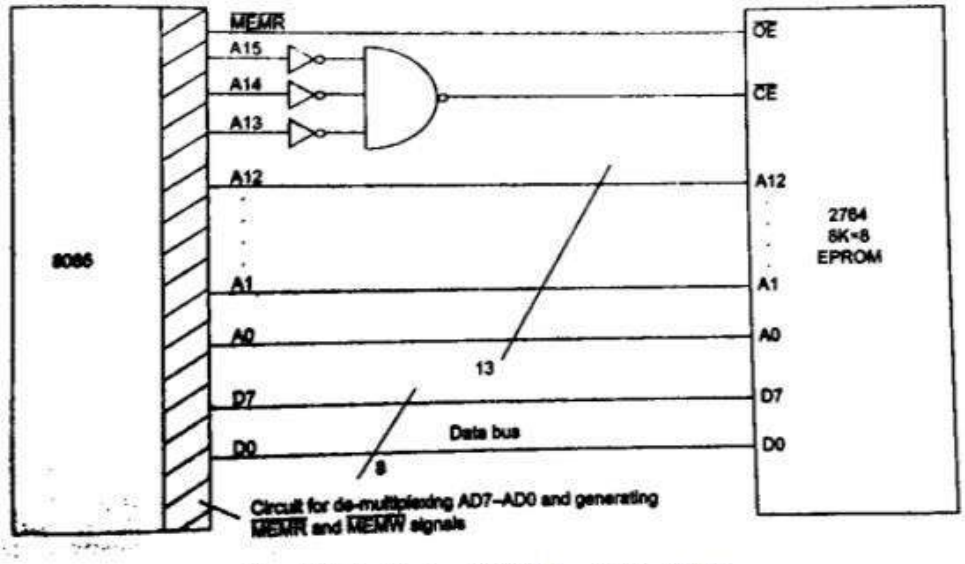


Fig. 17 Interfacing IC 2764 with the 8085

Table 9 Address allocated to IC 2764

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1FFEH
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

Ex: Interface a 6264 IC (8K x 8 RAM) with the 8085 using NAND gate decoder such that the starting address assigned to the chip is 4000H.

Specification of IC 6264:

- 8K x 8 RAM
- 8 KB = 213 bytes
- 13 address lines

The ending address of the chip is 5FFFH (since 4000H + 1FFFH = 5FFFH). When the address 4000H to 5FFFH are written in binary form, the values in the lines A15, A14, A13 are 0, 1 and 0 respectively. The NAND gate is designed such that when the lines A15 and A13 carry 0 and A14 carries 1, the output of the NAND gate is 0. The NAND gate output is in turn connected to the ^(CE1) pin of the RAM chip. A NAND output of 0 selects the RAM chip for read or write operation, since CE2 is already 1 because of its connection to +5V. Fig. 18 shows the interfacing of IC 6264 with the 8085.

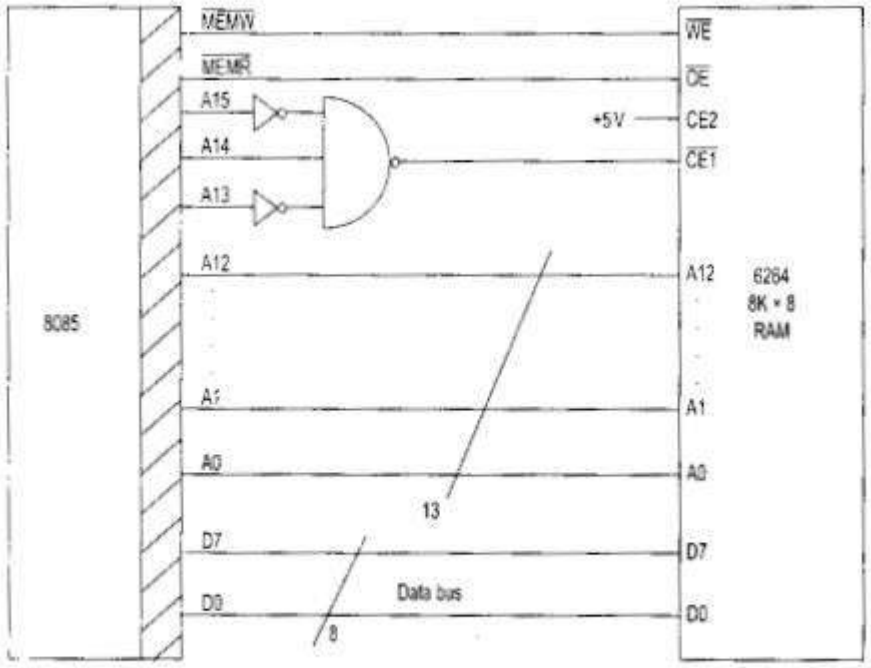


Fig. 18 Interfacing 6264 IC with the 8085

Ex: Interface two 6116 ICs with the 8085 using 74LS138 decoder such that the starting addresses assigned to them are 8000H and 9000H, respectively.

Specification of IC 6116:

- 2 K x 8 RAM
- 2 KB = 211 bytes
- 11 address lines

6116 has 11 address lines and since 2 KB, therefore ending addresses of 6116 chip 1 is and chip 2 are 87FFH and 97FFH, respectively. Table 10 shows the address range of the two chips.

Table 10 Address range for IC 6116

Table 10 Address range for IC 6116

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	87FFH (RAM chip 1)
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9000H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	97FFH (RAM chip 2)

Interfacing:

A0 – A10 lines of 8085 are connected to 11 address lines of the RAM chips.

· Three address lines of 8085 having specific value for a particular RAM are connected to the three select inputs (C, B and A) of 74LS138 decoder.

· Table 10 shows that A13=A12=A11=0 for the address assigned to RAM 1 and A13=0, A12=1 and A11=0 for the address assigned to RAM 2.

· Remaining lines of 8085 which are constant for the address range assigned to the two RAM are connected to the enable inputs of decoder.

· When 8085 places any address between 8000H and 87FFH in the address bus, the select inputs C, B and A of the decoder are all 0. The Y0 output of the decoder is also 0, selecting RAM 1.

· When 8085 places any address between 9000H and 97FFH in the address bus, the select inputs C, B and A of the decoder are 0, 1 and 0. The Y2 output of the decoder is also 0, selecting RAM 2.

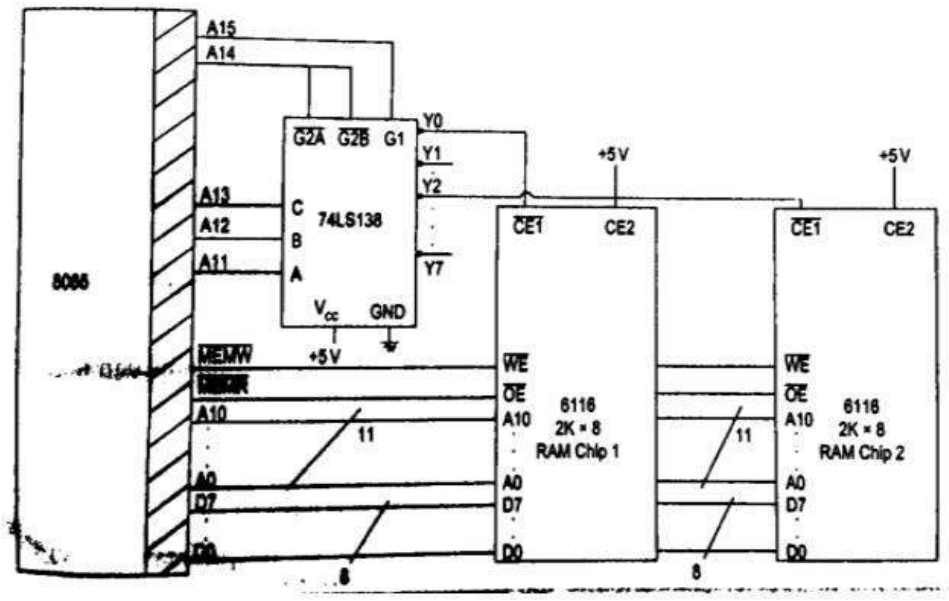


Fig. 19 Interfacing two 6116 RAM chips using 74LS138 decoder

Input Output Interfacing Techniques:

The most of the microprocessors support isolated I/O system. It partitions memory from I/O, via software, by having instructions that specifically access (address) memory, and others that specifically access I/O. When these instructions are decoded by the microprocessor, an appropriate control signal is generated to activate either memory or I/O operation. In 8085, IO/M signal is used for this purpose. The 8085 outputs a logic '1' on the IO/M line for an I/O operation and a logic '0' for memory, operation. In 8085, it is possible to connect 64 Kbyte memory and 256 I/O ports in the system since 8085 sends 16 bit address for memory and 8 bit address for I/O. I/O devices can be Input Output Interfacing Techniques to an 8085A system in two ways :

I/O Mapped I/O

Memory mapped I/O

In I/O mapped I/O, the 8085 uses IO/M signal to distinguish between I/O read/write and memory read/write operations. The 8085 has separate instructions IN and OUT for I/O data transfer. When 8085 executes IN or OUT instruction, it places device address (port number) on the demultiplexed low order address bus as well as the high order address bus. In other words, we can say that higher order address bus duplicates the contents of demultiplexed low-order address bus, when 8085 microprocessor executes an IN or OUT instruction. For example, if the device address is 60H then the contents on A_{15} to A_0 will be as follows :

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
0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0

Here, A_8 follows A_0, A_9 follows A_1 and so on, as shown below.

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Device Address
A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	
0	1	1	0	0	0	0	0	60H

The instruction IN inputs data from an input device (such as keyboard) into the accumulator and the instruction OUT sends the contents of the accumulator to an output device such as LED display. These are two byte instructions. The second byte of the instruction specifies the address or the port number of an I/O device. As it is a byte, the address or port number can be any of the 256 combinations of eight bits, from 00H to FFH. Therefore, the 8085 can communicate with 256 different I/O devices. When we want to Input Output Interfacing Techniques, it is necessary to assign a device address or a port number. Before going to see this device address logic, we will examine how the 8085 executes IN and OUT instructions.

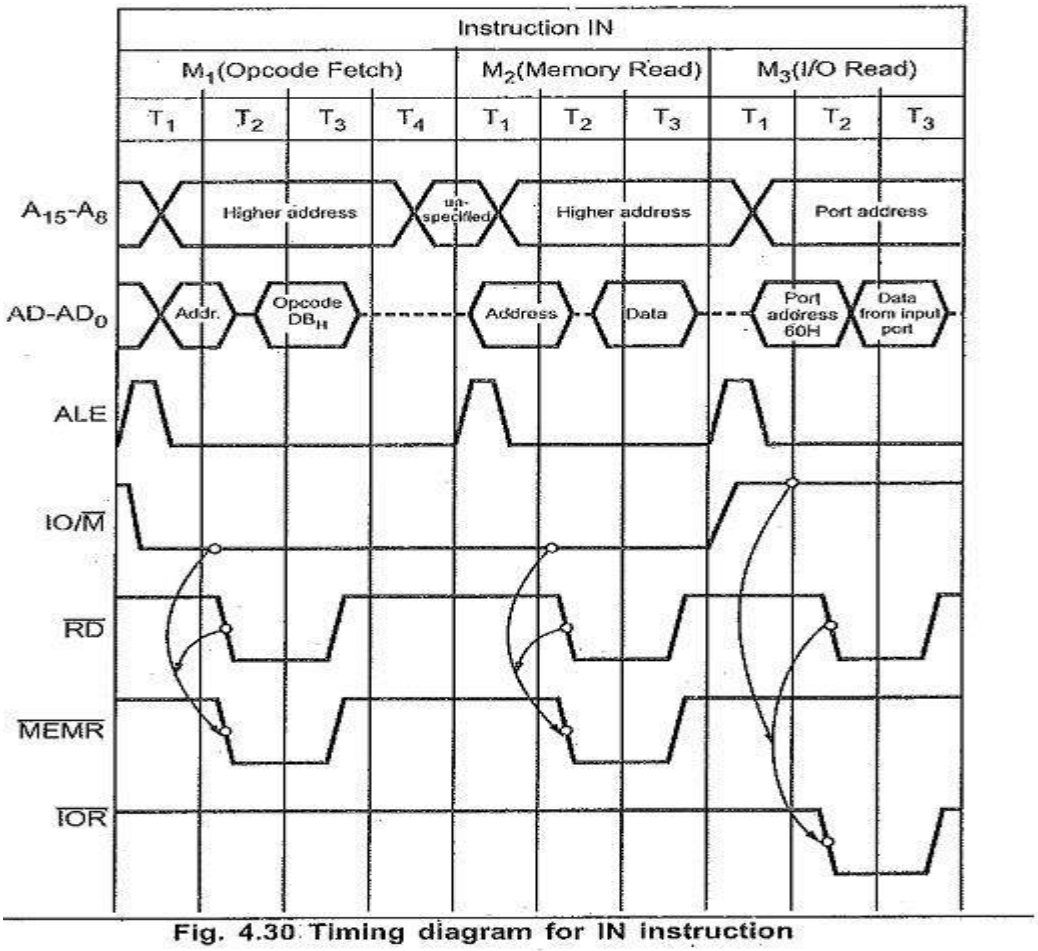


Fig. 4.30 Timing diagram for IN instruction

Fig. 4.30 shows the timing diagram of the IN instruction. It has three machine cycles. As usual, firSt cycle is opcode fetch machine cycle. The opcode fetch cycle is followed by one memory read machine cycle to read the address of port.

In the third machine cycle (I/O read) the 8085 microprocessor places the address of the input port on the low-order address bus AD_7-AD_0 as well as on the high-order address hubs A_{15}-A_8 and asserts the RD signal. During T_2 and T_3 of the machine cycle, RD and IO/M signals are 0 and 1 respectively, which activates IOR signal. The IOR signal enables the input port arid the data from the input port is placed on the data bus and transferred into the accumulator.

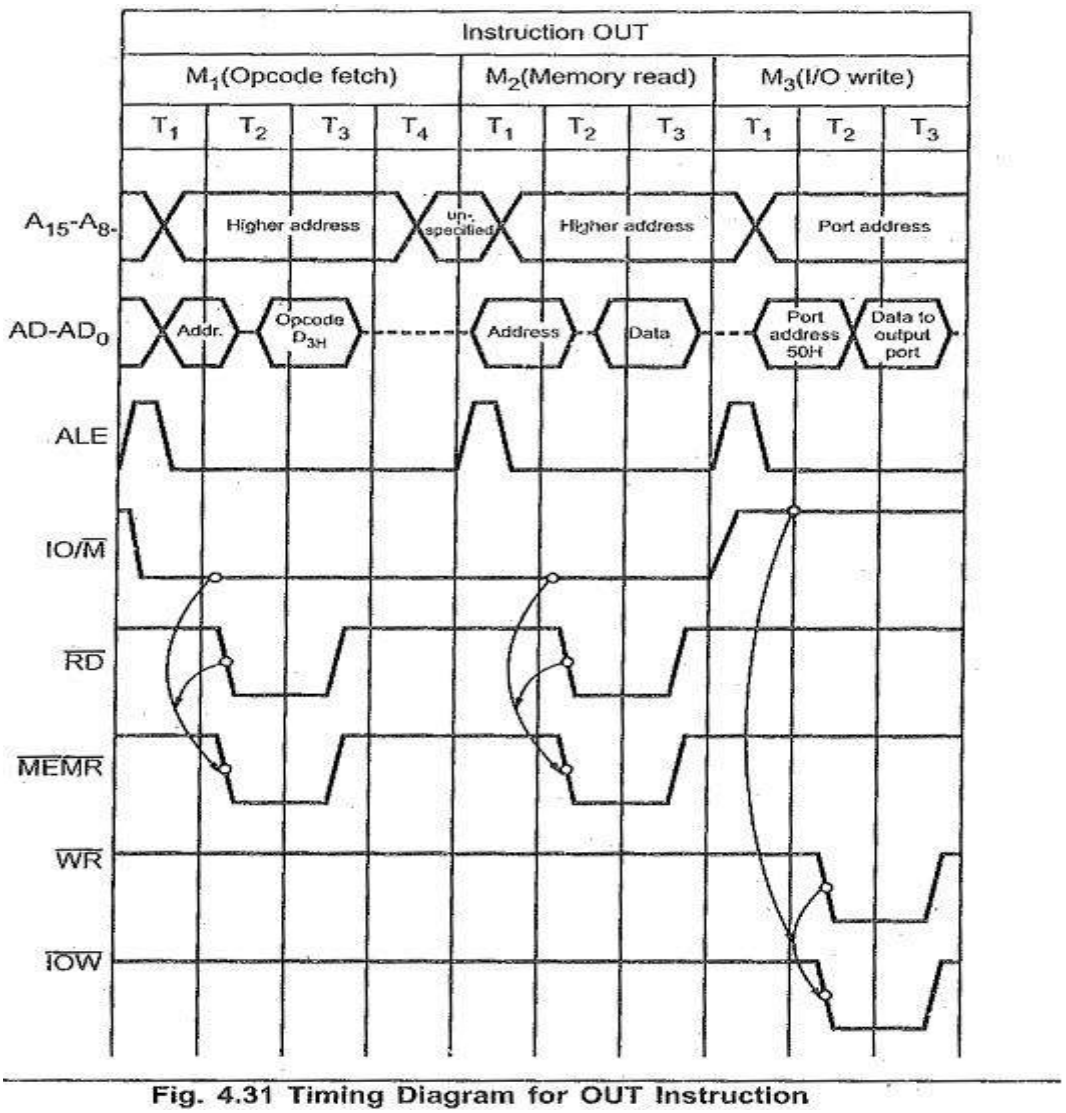


Fig. 4.31 Timing Diagram for OUT Instruction

Fig. 4.31 shows the timing diagram of OUT instruction. It has, three machine cycles. The first machine cycle is an opcode fetch machine cycle, which reads the opcode of OUT instruction from the memory. The second machine cycle is a memory read machine cycle. This machine cycle

reads the address of the port. In the third machine cycle (I/O write), the 8085 microprocessor places this address of the output port on the low-order address bus as well as on the high order address bus and asserts the WR signal. During T_2 and T_3 of the I/O write machine cycle, WR and ION signals are 0 and 1 respectively, which activates IOW signal. The IOW signal enables the output port and the data from the accumulator is sent to the output port.

Comparison Between Memory Mapped I/O and I/O Mapped I/O:

Memory mapped I/O	I/O mapped I/O
1. In this device address is 16 bit. Thus A_0 to A_{15} lines are used to generate device address.	1. In this I/O device address is 8 bit. Thus A_0 to A_7 or A_8 to A_{15} lines are used to generate device address.
2. $\overline{MEMR}$ and $\overline{MEMW}$ control signals are used to control read and write I/O operations.	2. $\overline{IOR}$ and $\overline{IOW}$ control signals are used to control read and write I/O operations.
3. Instructions available are LDA addr, STA addr, LDAX rp, STAX rp, MOV M,R, MOV R,M ADD M, CMP M etc.	3. Instructions available are IN and OUT.
4. Data transfer is between any register and I/O device.	4. Data transfer is between accumulator and I/O device.
5. Maximum number of I/O devices are 65536 (theoretically).	5. Maximum number of I/O devices are 256.
6. Execution speed using LDA addr, STA addr is 13 T-state and 7 T-states for MOV M, r and MOV r, M instructions.	6. Execution speed is 10 T-states.
7. Decoding 16 bit address may require more hardware.	7. Decoding 8 bit address will require less hardware.

UNIT V

MICROPROCESSOR APPLICATIONS

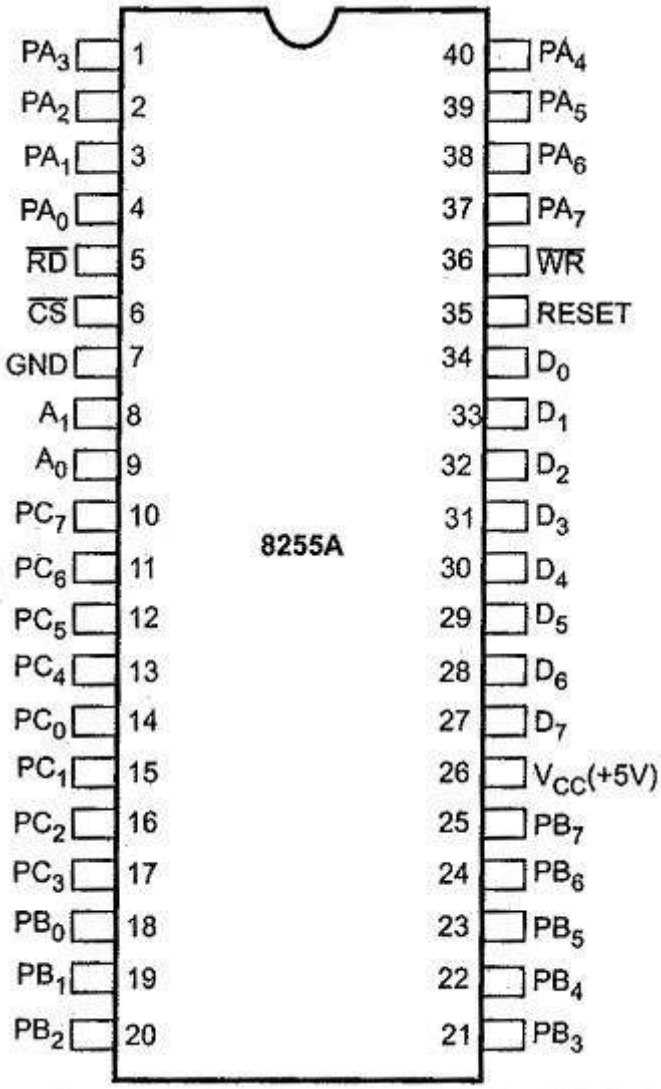
8255 Programmable Peripheral Interface

- 8255 is a Programmable Peripheral Interface, available in the form of a 40 pin IC which works on a power supply of +5 V DC.
- It is compatible with a wide range of microprocessors and microcontrollers, making it widely popular.
- It has three 8-bit I/O: Ports A, B, and C.
- Port A and port B can function as 8-bit input or output ports.
- Bits of port C are divided into two subgroups of 4 bits each – port C upper and port C lower.

- There are other control pins which are used to specify and control the flow of data and operation of the 8255..
- The port pins have the ability to source 1 mA current at 1.5 V when programmed to function as output pins. This provides the capability of driving Darlington transistors for applications such as printers and high voltage displays.
- The most important feature of 8255 is that it is 'programmable.' This means that the operation of 8255 can be controlled by programming the microprocessor appropriately. This gives us the freedom to use 8255 in a number of ways without having to change the wiring and connections.

- It has a few different modes of operation.

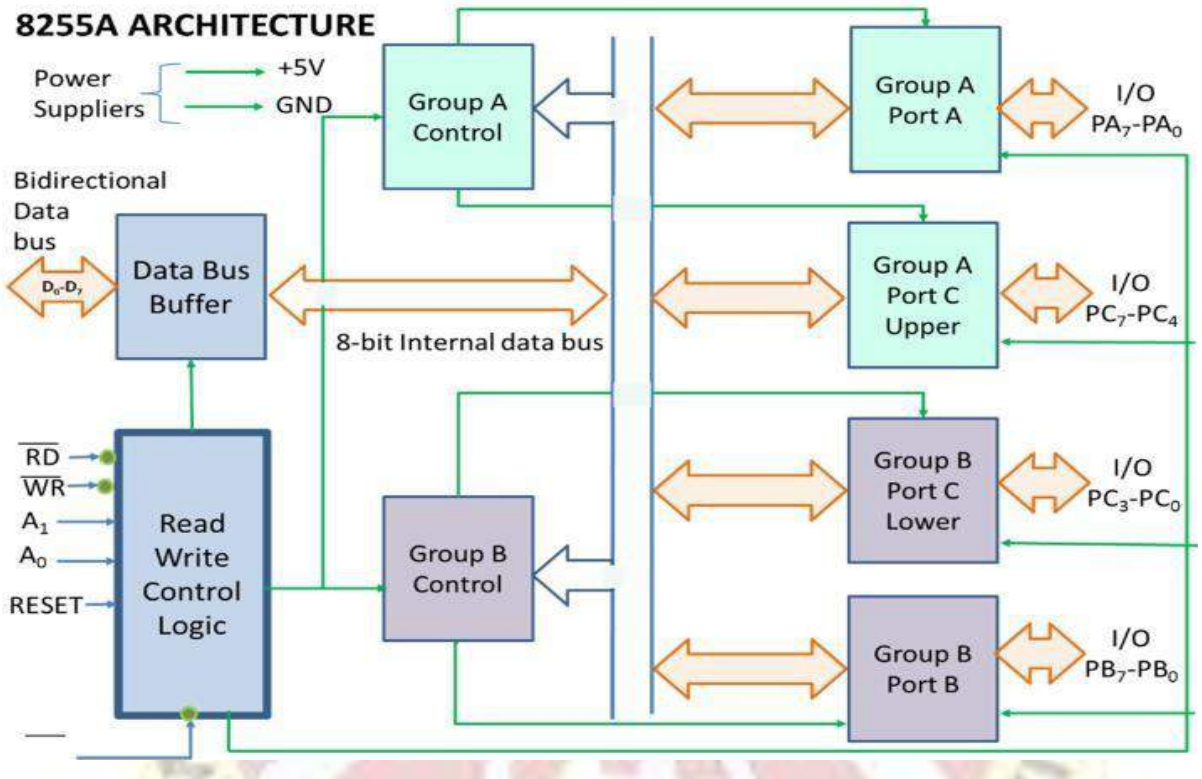
Pin diagram of 8255



Pin name	Number of pins	Description
Vcc	1	Used to supply power to the IC. Usually at +5 V dc with respect to ground.
Ground	1	Ground pin. All the voltages (signals) are measured with respect to this pin. It is connected to the common ground of the circuit.
PA0-PA7	8	These 8-bit bi-directional I/O pins are used to send data to output from a device and to receive data from an input device.
PB0-PB-7	8	These 8-bit bi-directional I/O pins are used to send data to an output device and to receive data from an input device.
PC0-PC3	4	These four pins are used by 8255 for communicating with an I/O device.
PC4-PC7	4	These four pins are used by 8255 for communicating with an I/O device.
D0-D7	8	These are the data pins used by the Master (uP/uC) to communicate with 8255. All the data to be transmitted and received and control instructions are transmitted to and from 8255 through these pins. These are connected to the data bus of the microprocessor.
CS	1	This is an active low input pin. The microprocessor uses this input to select the chip 8255. In other words, the microprocessor uses this pin to say to 8255 that "*Hey!! Now I am talking to you. And I will keep talking to you until this CS signal remains low.*"
RD	1	This is also an active low input pin used by the microprocessor to tell 8255 that it wants to read data from one of its ports.
WR	1	This is also an active low input pin used by the microprocessor to tell 8255 that it wants to write data to one of its ports.
A1, A0	2	These are the port address pins. They are used to select the port with which the microprocessor intends to communicate. For values of A1A0: 00 = Port A is selected 01 = Port B is selected 10 = Port C is selected

11 = Control port is selected.

Reset 1 An active high input. Used to reset 8255. Immediately after reset, all the three ports work as input ports in mode 0.



CS	A1	A0	Selected
0	0	0	PORT A
0	0	1	PORT B
0	1	0	PORT C
0	1	1	Control Register
1	X	X	8255A is not selected

Data Bus Buffer

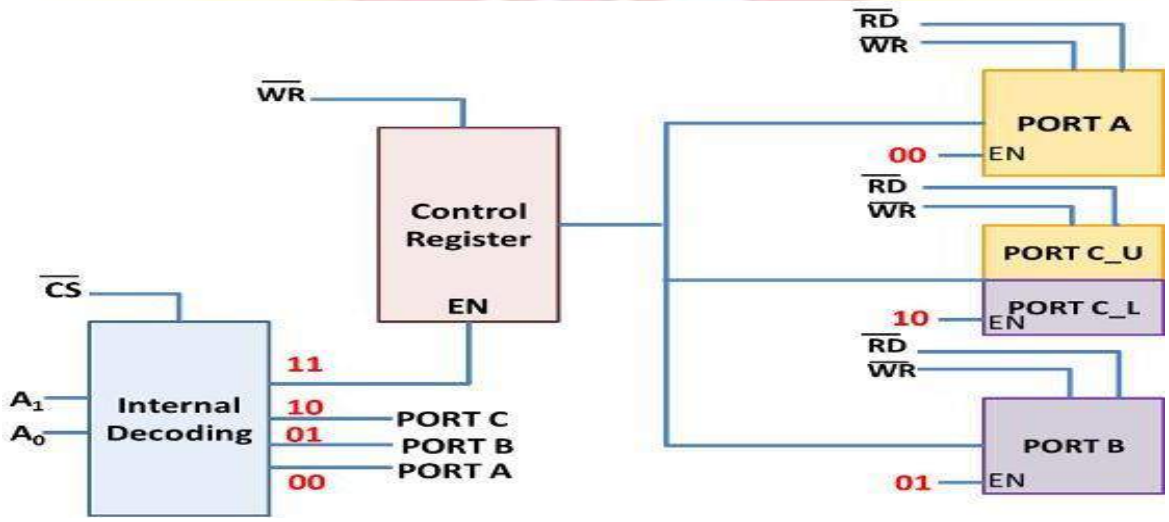
- This three-state bi-directional 8-bit buffer is used to interface the 8255 to the system data bus.
- Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU.
- Control words and status information are also transferred through the data bus buffer.

Group A and Group B Controls

- The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255.
- The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255.
- Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

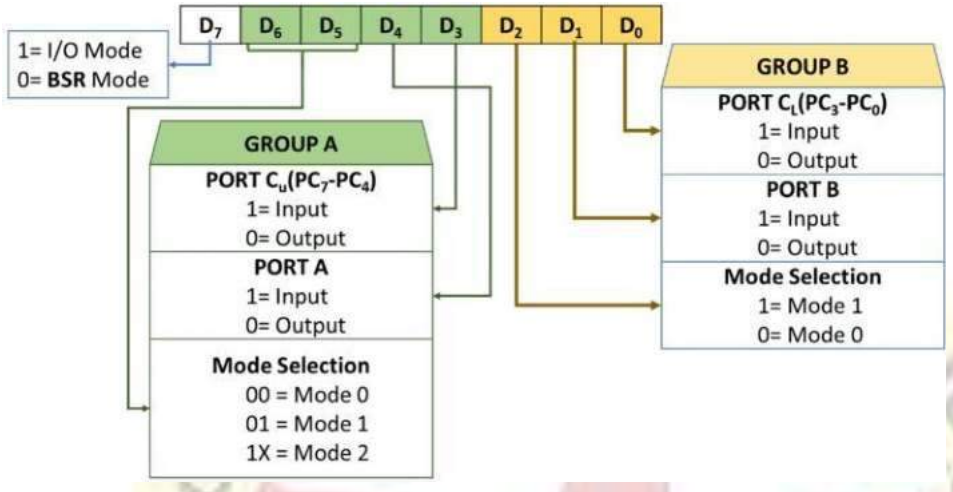
Ports A, B, and C

- The 8255 contains three 8-bit ports (A, B, and C).
- All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.
- Port A One 8-bit data output latch/buffer and one 8-bit data input latch.
- Both "pull-up" and "pull-down" bus-hold devices are present on Port A.
- Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer.
- Port C One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control.
- Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B.

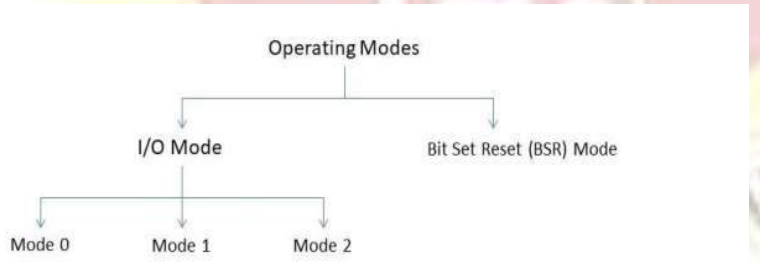


Control Word: Content of Control register is known as Control Word.

- Control word specify an I/O function for each port this register can be



- Accessed to write a control word when A0 and A1 are at logic1, the register is not accessible for a read operation.
- Bit D7 of the control register either specifies the I/O function or the bit Set/Reset function, as classified in figure 1.
- If bit D7=0, bits D6-D0 determine I/O function in various mode, as shown in figure 4.
- If bit D7=0 port C operates in the bit Set/Reset (BSR) mode.
- **MODES OF 8255**



Thesearetwobasicmodesofoperationof8255.

I/OmodeandBitSet-Resetmode(BSR).

InI/Omode,the8255portsworkasprogrammableI/Oports,whileinBSRmodeonlyportC(PC0-PC7) canbe usedtosetorresetitsindividualportbits.

UndertheI/Omodeofoperation,furtherthereare threemodesofoperationof8255,soasto

supportdifferenttypesof applications,mode0,mode1 andmode2.

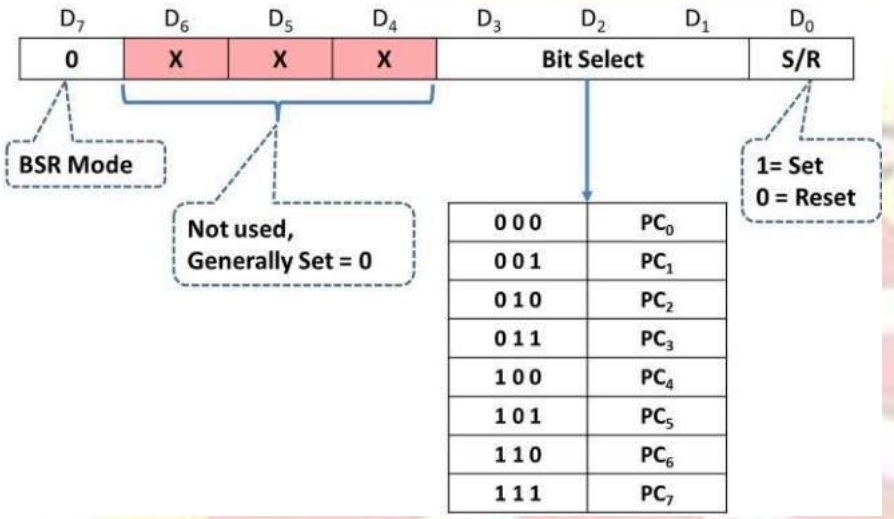
8255A:BSR(BitSet/Reset)Mode

Inthismodeanyofthe8-bitsofportCcanbesetorresetdependingonD0ofthecontrolword.

ThebittobesetorresetisselectedbybitselectflagsD3,D2andD1oftheCWR(ControlWord Register).

BSRControlWordaffectsonebitatatime

ItdoesnotaffecttheI/Omode



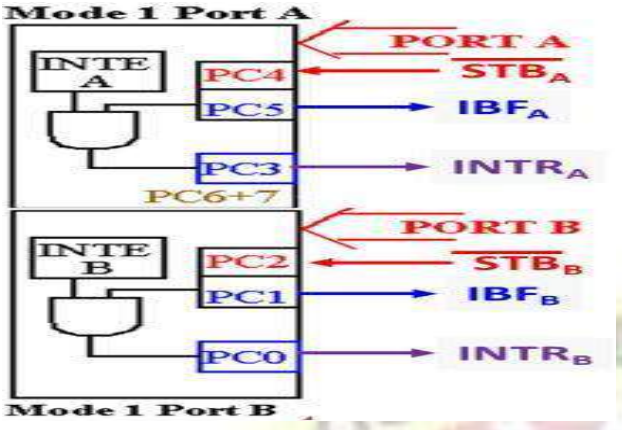
8255A has three different I/O operating modes:

1. Mode 0
2. Mode 1
3. Mode 2

MODE 0

- Simple I/O for port A,B and C
- In this mode, Port A and B is used as two 8-bit ports and Port C as two 4-bit ports.
- Each port can be programmed in either input mode or output mode where outputs are latched and inputs are not latched.
- **MODE 1: INPUT OR OUTPUT WITH HANDSHAKE**
- Handshake signal are exchanged between MPU and peripheral prior to data transfer.
- In this mode, Port A and B is used as 8-bit I/O ports.
- Mode 1 is a handshake Mode whereby ports A and/or B use bits from port C as handshake signals.
- In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt.
- Port A uses upper 3 signals of Port C: PC3, PC4, PC5

- Port B uses lower 3 signals of Port C : PC0, PC1, PC2
- PC6 and PC7 are general purpose I/O pinso not have handshake or interrupt capability.



STB (Strobe Input):

- This active low signal is generated by a peripheral device to indicate that, it has transmitted a byte of data. The 8255A, in response to STB, generates IBF and INTR.

IBF (Input Buffer Full)

This signal is acknowledged by 8255A to indicate that the input latch has received the data byte. It will get reset when the MPU reads the data.

INTR(Interrupt Request)

This is an output signal that may be used to interrupt the MPU. This signal is generated if STB, IBF and INTE (internal flip-flop) are all at logic 1. It will get reset by the falling edge of RD

INTE(Interrupt Enable)

- This signal is an internal flip-flop, used to enable or disable the generation of INTR signal.
- The interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the PC4 (port A) or PC2 (port B) bits.

MODE 2

- In this mode, Port A can be configured as the bidirectional port and Port B either in Mode 0 or Mode 1.
- Port A uses five signals from Port C as handshake signals for data transfer.
- The remaining three signals from Port C can be used either as simple I/O or as handshake for port B.

PROGRAMMABLE INTERVAL TIMER

TheIntel8254isacounter/timerdevicedesignedtosolvethecommontimingcontrolproblemsin microcomputersystemdesign.8254isthehighspeedversionofthe8253.

8254Vs8253

Characteristics	8254	8253
FrequencyRange	DCto8MHz DCto10MHz(8254-2)	DCto2MHz
SpecialCommand	Statusread-back command	Nosuchcommand

Application

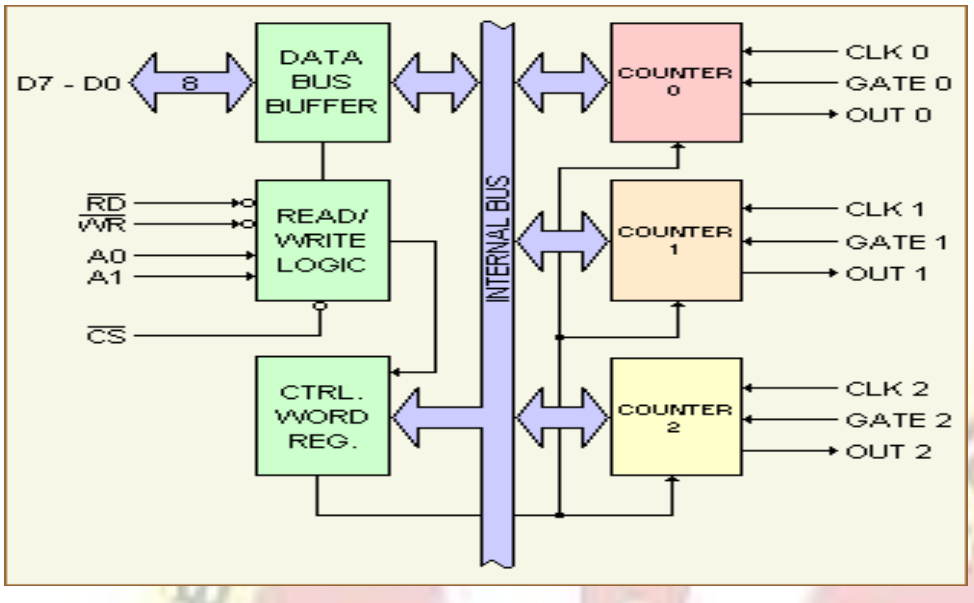
Someoftheothercounter/timerfunctionscommontomicrocomputers whichcanbeimplementedwiththe8254are:

- ✓ Realtimeclock
- ✓ Event-counter
- ✓ Digitalone-shot
- ✓ Programmableрategenerator
- ✓ Squarewavegenerator
- ✓ Binaryratemultiplier
- ✓ Complexwaveformgenerator
- ✓ Complexmotorcontroller

8254 Features

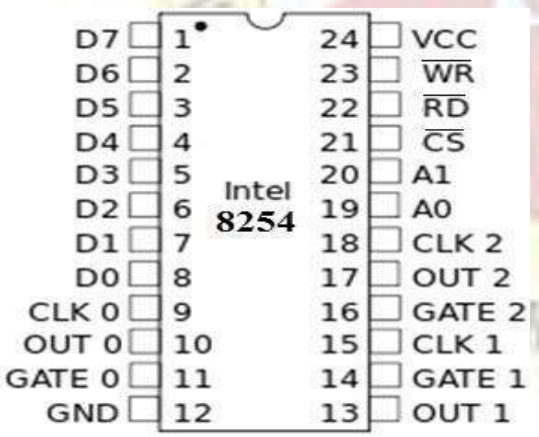
- ✓ Itincludesthree16-bitcountersthatcanworkindependentlyin6differentmodes.
- ✓ It ispackaged ina24-pinDIP(Dualin-linepackage) andrequires +5Vpowersupply.
- ✓ ItcancounteitherinbinaryorBCD.
- ✓ It'scounterscanoperateatamaximumfrequencyof10MHz.

✓ FunctionalDiagram



PIN FUNCTIONS

A0, A1: The address inputs select one of the four internal registers within the 8254.



A1	A0	Function
0	0	Counter 0
0	1	Counter 1

1 0 Counter 2

1 1 Control Word

CS: Chip select enables the 8254 for programming and for reading or writing a counter.

Vcc: Power connects to the +5V power supply.

GND: Ground connects to the system ground bus.

GATE : The gate input controls the operation of the counter in some modes of operation.

GATE 0 Gate input of counter 0

GATE 1 Gate input of counter 1

GATE 2 Gate input of counter 2

D0-D7: Bidirectional three state data bus lines connected to system data bus.

CLK : The clock input is the timing source for each of the internal counters. This input is often connected to the PCLK signal from the microprocessor system bus controller.

CLK0 Clock input of counter 0

CLK1 Clock input of counter 1

CLK2 Clock input of counter 2

OUT: A counter output is where the waveform generated by the counter is available.

OUT 0 Output of counter 0

OUT 1 Output of counter 1

OUT 2 Output of counter 2

RD: Read causes data to be read from the 8254 and often connected to the $IORC$ signal.

WR: Write causes data to be written to the 8254 and often connects to the

write strobe ($IOWC$)

Pogrammingthe8254(ControlWordFormat)

$A_1, A_0 = 11$ $\overline{CS} = 0$ $\overline{RD} = 1$ $\overline{WR} = 0$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

8254 Write operation

The programming procedure for the 8254 is very flexible. Only two conversion

need to be remember.

1) For each Counter, the Control Word must be written before the initial

count is written.

2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

With a clock and an appropriate gate signal to one of the counters, the above steps should start the counter and provide appropriate output according to the control word.

8254 Read Operations

There are three possible methods for reading the counters:

A simple read operation

The Counter Latch Command, and

The Read-Back Command.

Simple Read Operation

This operation read the counter after stopping.

To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

Two I/O read operation are performed by the MPU

1. The first I/O operation reads the low order byte.
2. The second I/O operation reads high order byte.

Counter Latch Command

This allows reading the contents of the Counters “on the fly" without affecting counting in progress.

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received.

This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to “following" the counting element (CE).

Read-Back Command

- ✓ This command is used to read several counters at a time. It eliminates the need of writing separate counter-latch commands for different counters.
- ✓ It allows the user to check the count value, programmed Mode, and current states of the OUT pin and Null Count flag of the selected counter/ counters.
- ✓ The read back command is written to the Control Word Register.
- ✓ The command is written into the Control Word Register and has the format shown in Figure.
- ✓ The read-back command may be used to latch multiple counter output latches (OL) by setting the COUNT bit D5 =0 and selecting the desired counter(s).

- ✓ A single **read back command** is functionally equivalent to several **counter latch commands.**
- ✓ Each counter's latched count is held in the OL until it is read (or the counter is reprogrammed). The counter is automatically unlatched when read, but other counters remain latched until they are read.
- ✓ Theread-backcommand mayalso beused tolatch statusinformation ofselected counter(s)by setting STATUSbit D4=0. Status must be latched toberead;statusofacounterisaccessedbyareadfromthatcounter.
- ✓ ThecounterstatusformatisshowninFigurebelow.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Output	Null Count	RW1	RW0	M2	M1	M0	BCD

D_7 1 = OUT Pin is 1
0 = OUT Pin is 0

D_6 1 = Null Count
0 = Count available for reading

D_5–D_0 Counter programmed mode

Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word.

OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system

Modes of Operation

Mode 0: Interrupt on terminal count.

Mode 1: Hardware Retriggerable One-Shot.

Mode 2: Rate Generator.

Mode 3: Square Wave Mode.

Mod 4: Software Triggered Mode.

Mode 5: Hardware Triggered Mode

Mode 0: Interrupt on terminal count.

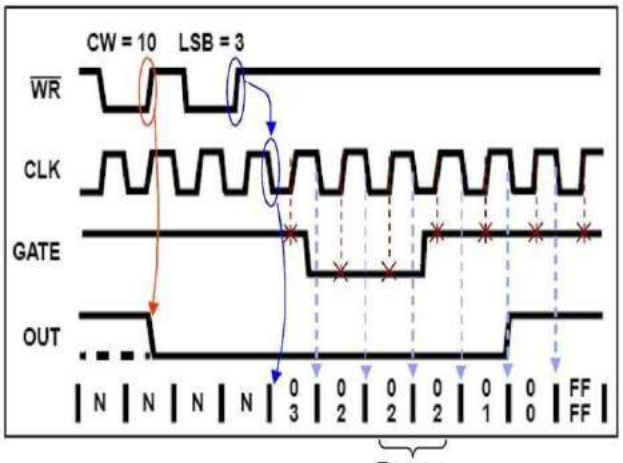
* N stands for an undefined count.

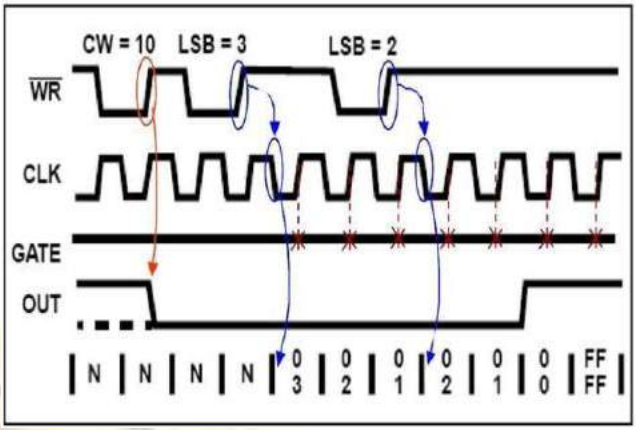
- ✓ Mode 0 is typically used for event counting.
- ✓ After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.
- ✓ After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N +1 CLK pulses after the initial count is written.

GATE =1 enables counting; GATE = 0 disables counting.

GATE has no effect on OUT. If G becomes a logic 0 in the middle of the count, the counter will remain stop until G again becomes a logic 1.

If a new count is written to the Counter, it will be loaded on the next



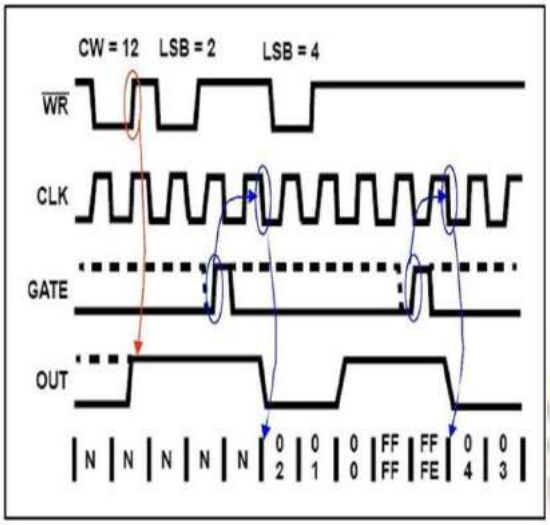


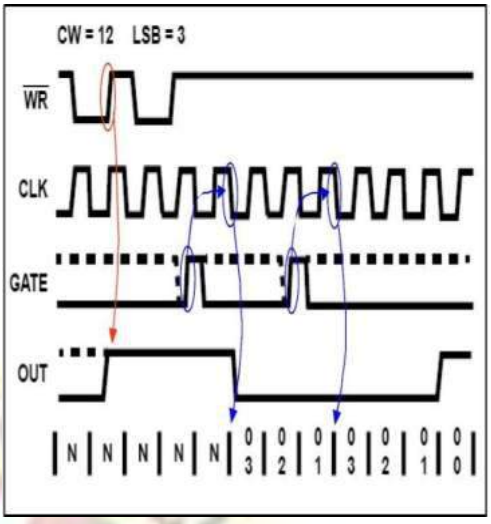
new count.

- ✓ At the rising edge of WR(CW) OUT becomes high.
- ✓ At the first falling edge of clock after first rising edge of WR(LSB), counter starts counting.

Mode 1: Hardware Retriggerable One-Shot

- **Causes the counter to function as a retriggerable, monostable multivibrator (one-shot).**
- **OUT is initially (after loading CW) high. Also remain high when count is written.**
- **When gate is triggered, OUT goes low and will remain low until the Counter reaches zero. On completion of count OUT goes high again**
- **If the GATE input occurs within the duration of counting, the counter is again reloaded with the count and start counting from the beginning.**



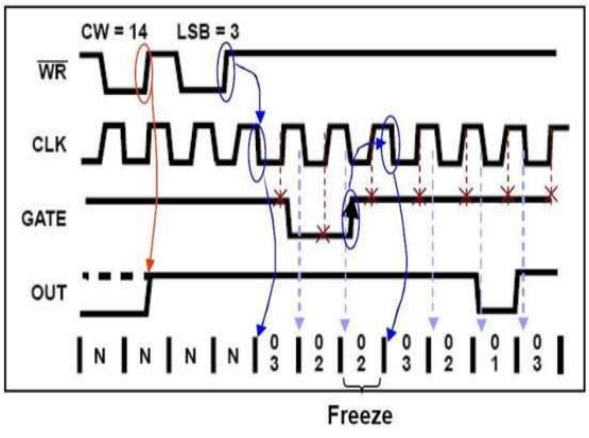


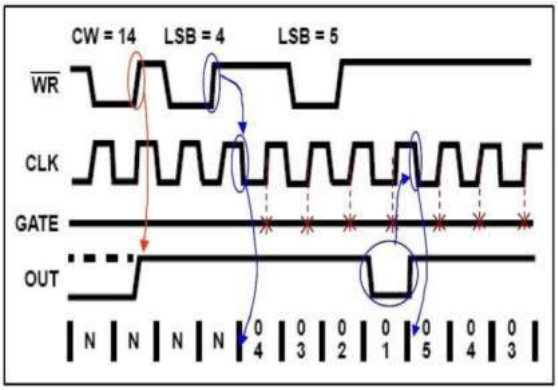
Mode2:RATEGENERATOR

- Allows the counter to generate a series of continuous pulses that are one clock pulse wide.
- The separation between pulses is determined by the count.
- If count N is loaded then, output will remain high for (N-1) clock period and low for 1 clock period. For example, for a count of 10, the output is a logic 1 for nine clock period and low for 1 clock period. This cycle is repeated until the counter is programmed with a new count or until G pin is placed at a logic 0 level.
- The G input must be logic 1 for this mode to generate a continuous series of pulses.
- In mode 2, a COUNT of 1 is illegal

  At the rising edge of WR(CW) OUT becomes high.
- At the first falling edge of clock after first rising edge of WR(LSB), counter

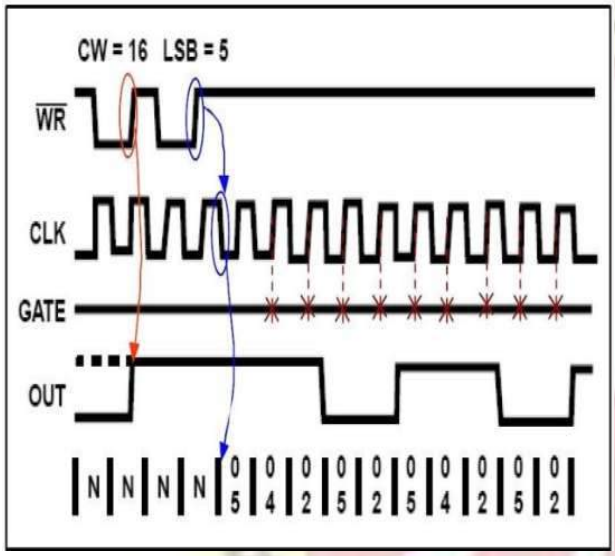
  starts counting.

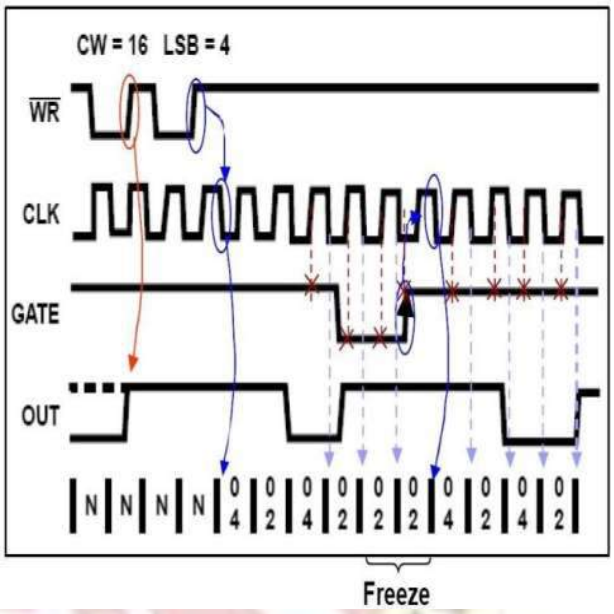




Mode3:SquareWaveMode.

- ✓ **For example, if the count is programmed for a count of 5, the output is high for three clocks and low for two clocks.**
- ✓ **Gate should be maintained at logic 1 always (GATE =1 enables counting; GATE =0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required).**





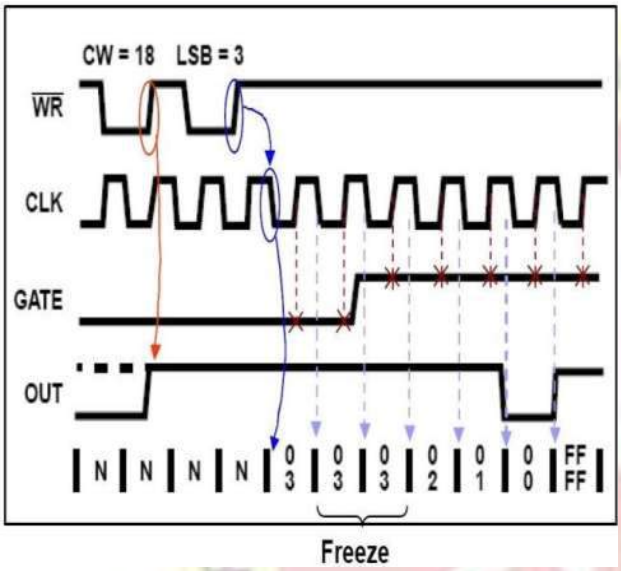
- **At the rising edge of WR(CW) OUT becomes high.**
- **At the first falling edge of clock after first rising edge of WR(LSB), counter starts counting.**

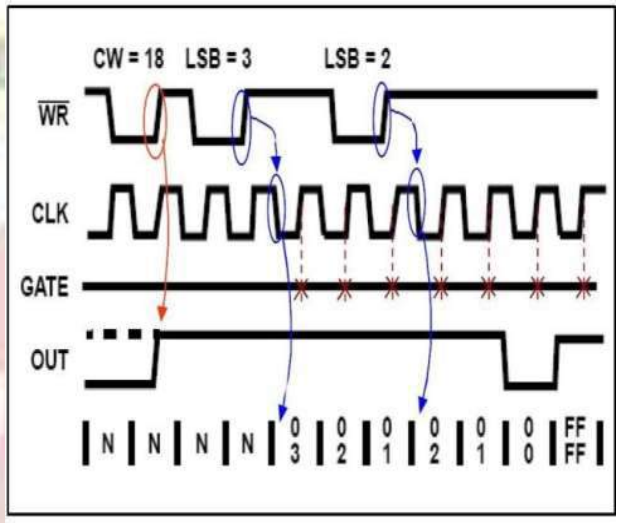
Mode 4: Software Triggered One-shot.

- ✓ **Allows the counter to produce a single pulse at the output.**
- ✓ **If count of N is loaded, then OUT will be high for N clock cycles and low for one clock cycle at the end.**
- ✓ **The cycle does not begin until the counter is loaded again.**
- **G input must be maintained at logic 1 throughout the operation.**
- **This mode operates as a software triggered one-shot.**

N.B. The G input must be a logic 1 for the counter to operate for these three modes (Mode 2, 3, 4)

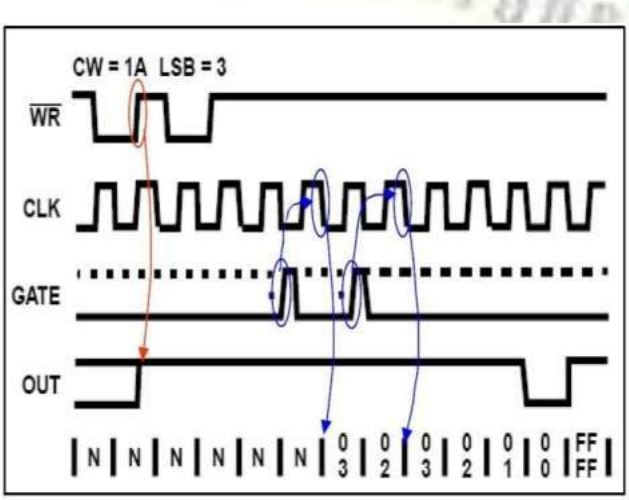
- **At the rising edge of WR(CW) OUT becomes high.**
- **At the first falling edge of clock after first rising edge of WR(LSB), counter starts counting.**

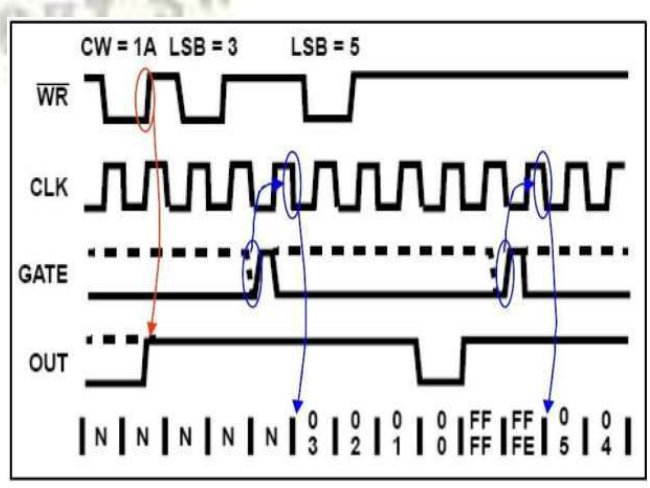




Mode5:HardwareTriggeredMode.

- **At the rising edge of WR(CW) OUT becomes high.**
- **At the first falling edge of clock after first rising edge of GATE , counter starts counting.Mode 5: Hardware Triggered Mode**





8279 - Programmable Keyboard

8279 programmable keyboard/display controller is designed by Intel that interfaces a keyboard with the CPU. The keyboard first scans the keyboard and identifies if any key has been pressed. It then sends their relative response of the pressed key to the CPU and vice-a-versa.

How Many Ways the Keyboard is Interfaced with the CPU?

The Keyboard can be interfaced either in the interrupt or the polled mode. In the Interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU will continue with its main task.

In the Polled mode, the CPU periodically reads an internal flag of 8279 to check whether any key is pressed or not with key pressure.

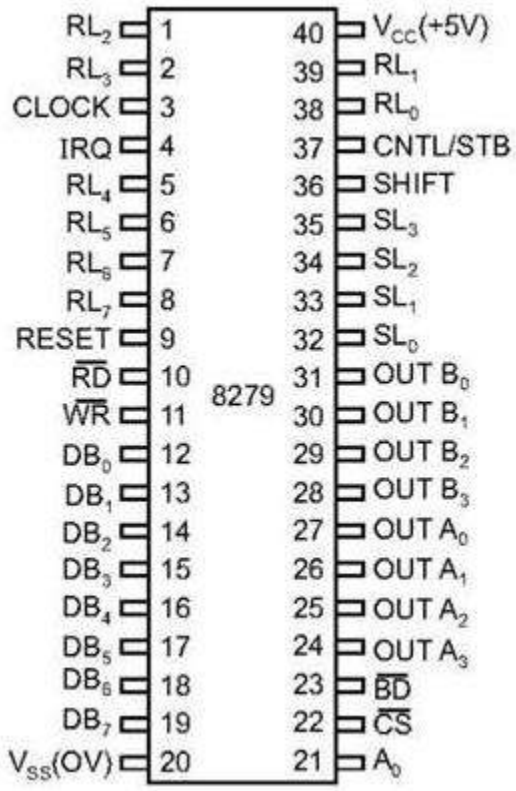
How Does 8279 Keyboard Work?

The keyboard consists of maximum 64 keys, which are interfaced with the CPU by using the key-codes. These key-codes are de-bounced and stored in an 8-byte FIFORAM, which can be accessed by the CPU. If more than 8 characters are entered in the FIFO, then it means more than eight keys are pressed at a time. This is when the overrun status is set.

If a FIFO contains a valid key entry, then the CPU is interrupted in an interrupt mode else the CPU checks the status in polling to read the entry. Once the CPU reads a key entry, then FIFO is updated, and the key entry is pushed out of the FIFO to generate space for new entries.

8279 − Pin Description

The following figure shows the pin diagram of 8279 −



Data Bu Lines, DB_0 - DB_7

These are 8 bidirectional data bus lines used to transfer the data to/from the CPU.

CLK

The clock input is used to generate internal timings required by the microprocessor.

RESET

As the name suggests this pin is used to reset the microprocessor.

CS Chip Select

When this pin is set to low, it allows read/write operations, else this pin should be set to high.

A_0

This pin indicates the transfer of command/status information. When it is low, it indicates the transfer of data.

RD, WR

This Read/Write pin enables the data buffer to send/receive data over the data bus.

IRQ

This interrupt output line goes high when there is data in the FIFO sensor RAM. The interrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

V_{ss}, V_{cc}

These are the ground and power supply lines of the microprocessor.

$SL_0 - SL_3$

These are the scan lines used to scan the keyboard matrix and display the digits. These lines can be programmed as encoded or decoded, using the mode control register.

$RL_0 - RL_7$

These are the Return Lines which are connected to one terminal of keys, while the other terminal of the keys is connected to the decoded scan lines. These lines are set to 0 when any key is pressed.

SHIFT

The Shift input line status is stored along with every key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure, it is pulled up internally to keep it high

CNTL/STB - CONTROL/STROBED I/P Mode

In the keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a key closure.

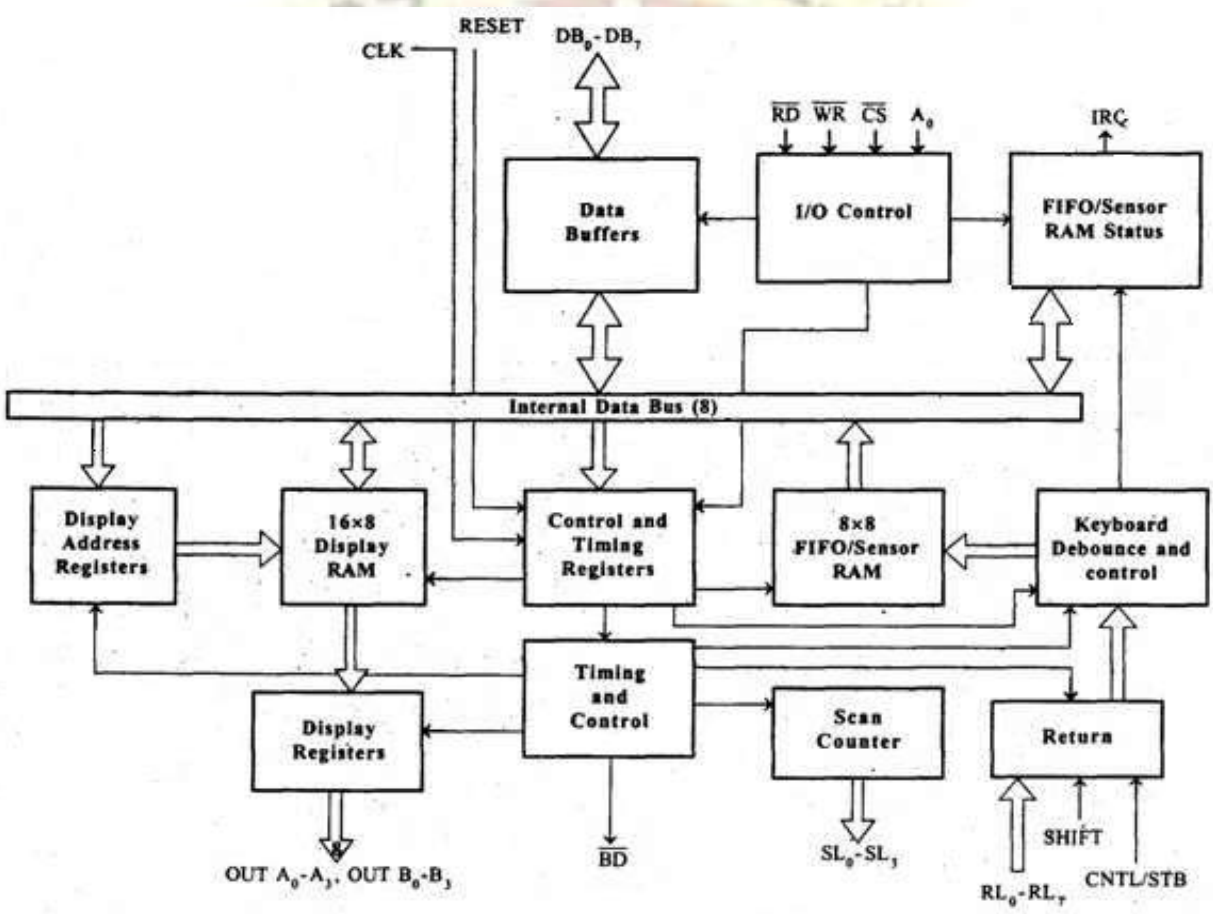
BD

It stands for blank display. It is used to blank the display during digit switching.

$OUTA_0$ – $OUTA_3$ and $OUTB_0$ – $OUTB_3$

These are the output ports for two 16x4 or one 16x8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and the keyboard.

ARCHITECTURE AND DESCRIPTION



I/O Control And Data Buffer

This unit controls the flow of data through the microprocessor. It is enabled only when D is low. Its data buffer interfaces the external bus of the system with the internal bus of the microprocessor. The pins A0, RD, and WR are used for command, status or data read/write operations.

Control And Timing Register And Timing Control

This unit contains registers to store the keyboard, display modes, and other operations as programmed by the CPU. The timing and control unit handles the timings for the operation of the circuit.

Scan Counter

It has two modes i.e. Encoded mode and Decoded mode. In the encoded mode, the counter provides the binary count that is to be externally decoded to provide the scan lines for the keyboard and display.

In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL_0-SL_3.

Return Buffers, Keyboard Debounce, And Control

This unit first scans the key closure row-wise, if found then the keyboard debounce unit debounces the key entry. In case, the same key is detected, then the code of that key is directly transferred to the sensor RAM along with SHIFT & CONTROL key status.

FIFO/Sensor RAM and Status Logic

This unit acts as 8-byte first-in-first-out (FIFO) RAM where the key code of every pressed key is entered into the RAM as per their sequence. The status logic generates an interrupt request after each FIFO read operation till the FIFO gets empty.

In the scanned sensor matrix mode, this unit acts as sensor RAM where its each row is loaded with the status of their corresponding row of sensors into the matrix. When the sensor changes its state, the IRQ line changes to high and interrupts the CPU.

Display Address Registers and Display RAM

This unit consists of display address registers which holds the addresses of the word currently read/written by the CPU to/from the display RAM.

Operational Modes of 8279

There are two modes of operation on 8279 − Input Mode and Output Mode.

Input Mode

This mode deals with the input given by the keyboard and this mode is further classified into 3 modes.

Scanned Keyboard Mode − In this mode, the key matrix can be interfaced using either encoded or decoded scans. In the encoded scan, an 8×8 keyboard or in the decoded scan, a 4×8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.

Scanned Sensor Matrix − In this mode, a sensor array can be interfaced with the processor using either encoder or decoder scans. In the encoder scan, 8×8 sensor matrix or with decoder scan 4×8 sensor matrix can be interfaced.

Strobed Input − In this mode, when the control line is set to 0, the data on the return lines is stored in the FIFO byte by byte.

Output Mode

This mode deals with display-related operations. This mode is further classified into two output modes.

Display Scan − This mode allows 8/16 character multiplexed displays to be organized as dual 4-bit/single 8-bit display units.

Display Entry − This mode allows the data to be entered for display either from the right side/left side.

KEYBOARD AND DISPLAY INTERFACE USING INTEL 8279 MICROPROCESSOR

In a microprocessor b system, when keyboard and 7-segment LED display is interfaced using ports or latches then the processor has to carry the following task.

- Keyboard scanning
- Key debouncing
- Key code generation
- Sending display code to LED
- Display refreshing

Interfacing 8279 with 8085 processor:

- A typical Hexa keyboard and 7-segment LED display interfacing circuit using 8279 is shown.

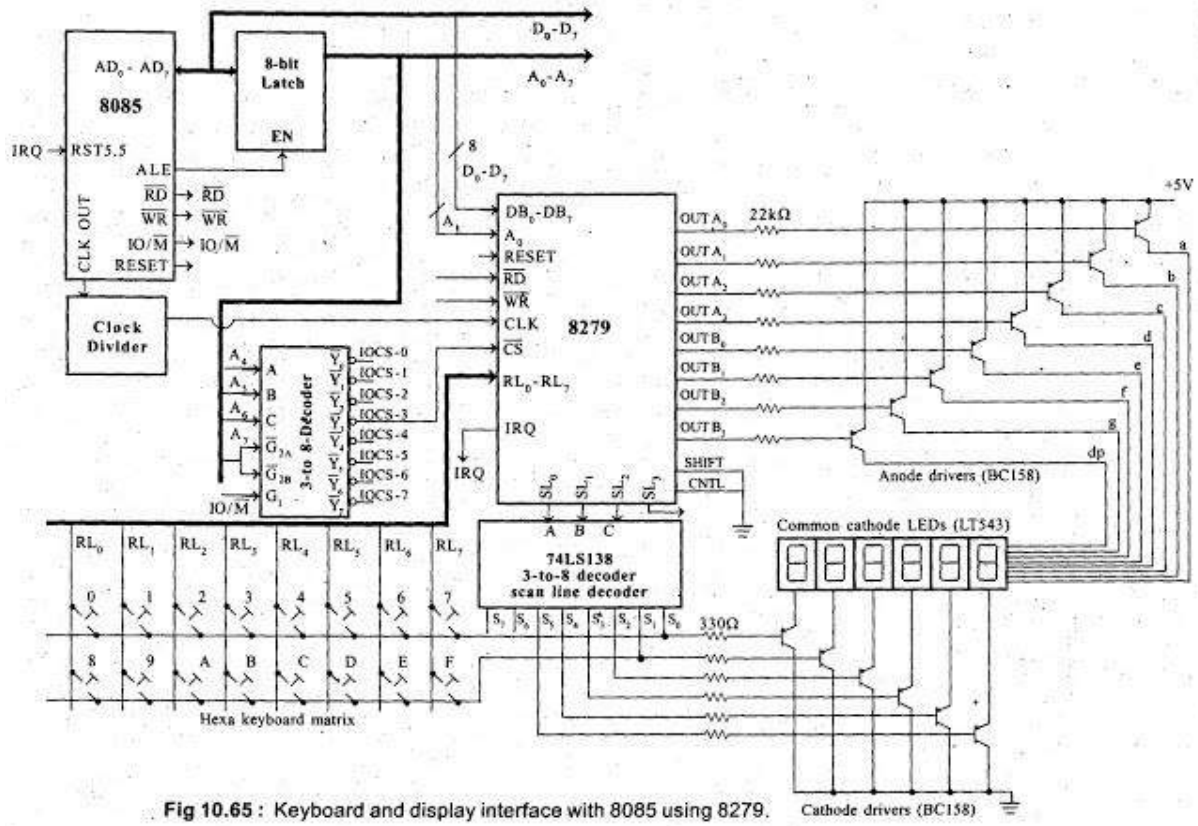


Fig 10.65 : Keyboard and display interface with 8085 using 8279.

- Th ecircuit can be used in 8085 microprocessor system and consist of 16 numbers of hexa-keys and 6 numbers of 7-segment LEDs.
- The 7-segment LED scan be used to display six digital phanumeric character.

- The 8279 can be either memory mapped or I/O mapped in the system. In the circuit shown is the 8279 is I/O mapped.
- The address line A0ofthesystem isusedasA0of8279.
- The clock signal for 8279 is obtained by dividing the output clock signal of 8085 by a clock divider circuit.
- The chip select signal is obtained from the I/O address decoder of the 8085 system. The chip select signals for I/O mapped devices are generated by using a 3-to-8 decoder.

- TheaddresslinesA4,A5 andA6are usedasinputtodecoder.
- TheaddresslineA7andthecontrolsignalIO/M(low)areusedasenablefordecoder.
- Thechipselect signalIOCS-3isusedtoselect8279.
- TheI/Oaddressofthe internaldevicesof8279 areshownintable.

Internal Device	Binary Address								Hexa Address
	Decoder input and enable				Input to address line of 8279				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data register	0	0	1	1	x	x	x	0	30
Control register	0	0	1	1	x	x	x	1	31

Note : ***Don't care "x" is considered as zero.***

- Thecircuithas6 numbersof7-segmentLEDsandso the8279hastobe programmedin encodedscan.(Because indecodedscan,only4numbersof7-segmentLEDscanbeinterfaced):
- Inencodedscantheoutputofscanlineswillbebinarycount.Thereforeanexternal,3-to-8decoderisusedtodecode thescanlinesSL0,SL1 andSL2of8279 to produceeightscanlinesS0 to S7.
- ThedecodedscanlinesS0 andS1arecommonforkeyboardanddisplay.
- The decoded scan lines S2to S5are used only for display and the decoded scan lines S6andS7arenotusedinthesystem.
- AnodeandCathodedriversareprovidedtotake careofthecurrentrequirementofLEDs.
- Thepnptransistors,BC158areusedasdrivertransistors.
- Theanodedriversarecalledsegmentdriversandcathodedriversarecalleddigitdrivers.
- The 8279 output the display code for one digit through its output lines

(OUT A0to OUTA3 andOUTB0toOUTB3)andsenda scancode through, SL0-SL3.

- Thedisplaycodeisinvertedbysegmentdriversandsenttosegmentbus.
- Thescancode isdecodedbythedecoderandturns ONthecorrespondingdigitdriver.

Now one digit of the display characteris displayed.Afterasmallinterval (10milli- second, typical), the displayis turned OFF(i.e., display is blanked) and the above process Is repeated for next digit.Thus multiplexed displa yis performed by 8279.

- The keyboard matrix is- formed using the return lines, RL0 to RL3 of 8279 as columns and decoded scanlinesS0andS1asrows.
- Ahexakey isplacedatthe crossingpointofeachrowandcolumn.Akeypresswillshortthe rowandcolumn.Normallythecolumnandrow linewill behigh.

Duringscanningthe8279willoutput binarycount onSL0toSL3,whichisdecodedby decodertomakea row as zero. When a row is zero the 8279 reads the columns. If there isakeypressthenthe correspondingcolumnwillbezero.

- If8279detectsakeypressthenitwait fordebounce timeandagainreadthecolumnsto generatekeycode.
- In encoded scan keyboard mode, the 8279 stores an 8-bit code for each valid key press. The keycodeconsist of the binary value of the column and row in which the key is found and the status of shift and controlkey.
- Afterascantime,the nextrowismadezeroandthe above processisrepeatedandsoon.Thus8279continuously scanthekeyboard.

HARDWARE FOR TRAFFIC LIGHT CONTROL

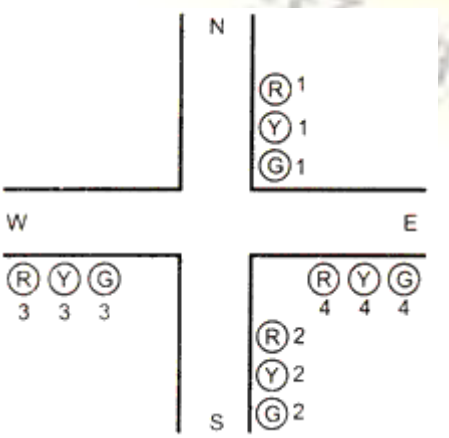
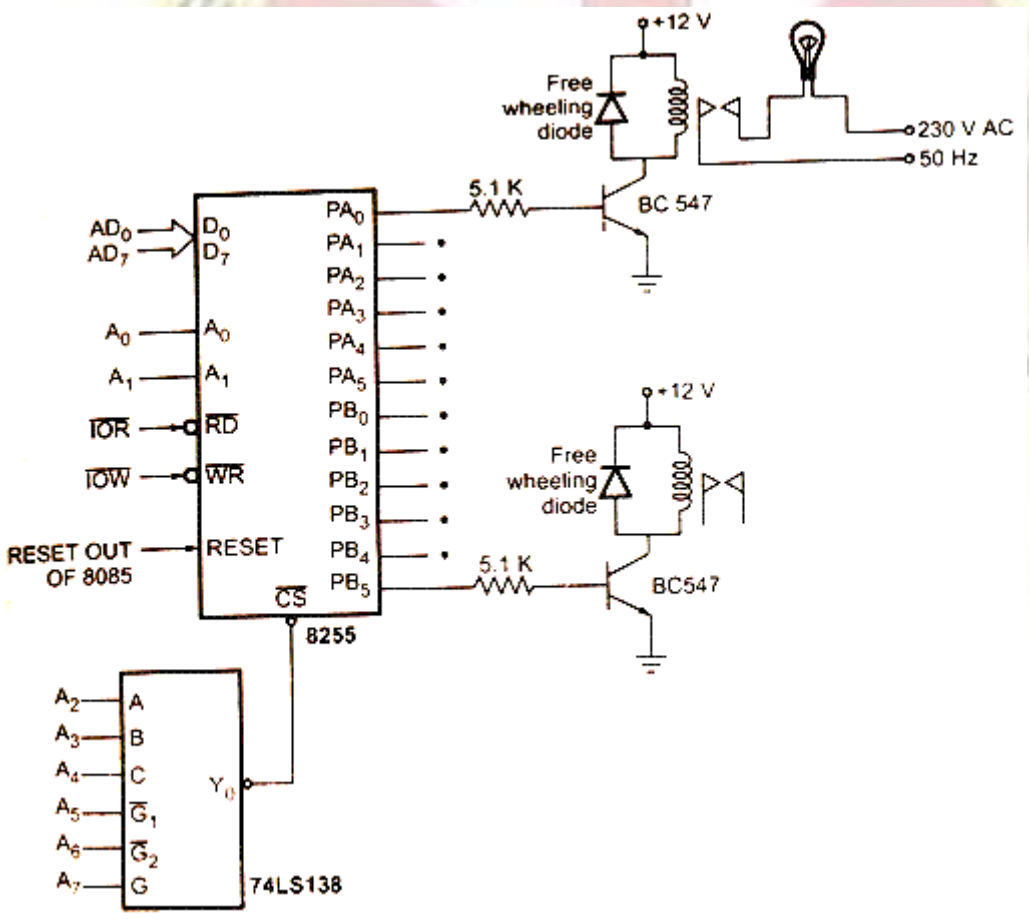


Fig. shows the interfacing diagram to control 12 electric bulbs. Port A is used to control lights on N-S road and Port B is used to control lights on W-E road. Actual pin connections are listed in Table 1 below.

Pins	Light	Pins	Light
PA_0	R_1	PB_0	R_3
PA_1	Y_1	PB_1	Y_3
PA_2	G_1	PB_2	G_3
PA_3	R_2	PB_3	R_4
PA_4	Y_2	PB_4	Y_4
PA_5	G_2	PB_5	G_4

Table 1

INTERFACING DIAGRAM



I/O MAP:

Ports / Control Register	Address lines A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0	Address
Port A	1 0 0 0 0 0 0 0	80H
Port B	1 0 0 0 0 0 0 1	81H
Port C	1 0 0 0 0 0 1 0	82H
Control Register	1 0 0 0 0 0 1 1	83H

Table 2

SOFTWARE FOR TRAFFIC LIGHT CONTROL

Control word : For initialization of 8255.

BSR/IO	$MODE_A$		P_A	PC_H	MODE B	P_B	PC_L
1	0	0	0	X	0	0	X

= 80H

Fig. Control word

Table shows the data bytes to be sent for specific combinations.

To glow	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0	Port B Output	Port A Output
R_1,R_2,G_3 and G_4	X	X	1	0	0	1	0	0	X	X	0	0	1	0	0	1	24H	09H
Y_1,Y_2,Y_3 and Y_4	X	X	0	1	0	0	1	0	X	X	0	1	0	0	1	0	12H	12H
R_3,R_4,G_1 and G_2	X	X	0	0	1	0	0	1	X	X	1	0	0	1	0	0	09H	24H

Source program:

```
        MVI A, 80H          : Initialize 8255, port A and port B
        OUT 83H (CR)        : in output mode
START:  MVI A, 09H
        OUT 80H (PA)        : Send data on PA to glow R1 and R2
        MVI A, 24H
        OUT 81H (PB)        : Send data on PB to glow G3 and G4
        MVI C, 28H          : Load multiplier count (40ıo) for delay
        CALL DELAY          : Call delay subroutine
```

```
    MVI A, 12H
    OUT (81H) PA        : Send data on Port A to glow Y1 and Y2
    OUT (81H) PB        : Send data on port B to glow Y3 and Y4
    MVI C, 0AH          : Load multiplier count (10ıo) for delay
CALL: DELAY             : Call delay subroutine
    MVI A, 24H
    OUT (80H) PA        : Send data on port A to glow G1 and G2
    MVI A, 09H
    OUT (81H) PB        : Send data on port B to glow R3 and R4
    MVI C, 28H          : Load multiplier count (40ıo) for delay
    CALL DELAY          : Call delay subroutine
    MVI A, 12H
    OUT PA              : Send data on port A to glow Y1 and Y2
    OUT PB              : Send data on port B to glow Y3 and Y4
    MVI C, 0AH          : Load multiplier count (10ıo) for delay
    CALL DELAY          : Call delay subroutine
    JMP START
```

Delay Subroutine:

```
DELAY: LXI D, Count     : Load count to give 0.5 sec delay
BACK: DCX D             : Decrement counter
    MOV A, D
    ORA E               : Check whether count is 0
    JNZ BACK            : If not zero, repeat
    DCR C               : Check if multiplier zero, otherwise repeat
    JNZ DELAY
    RET                 : Return to main program
```

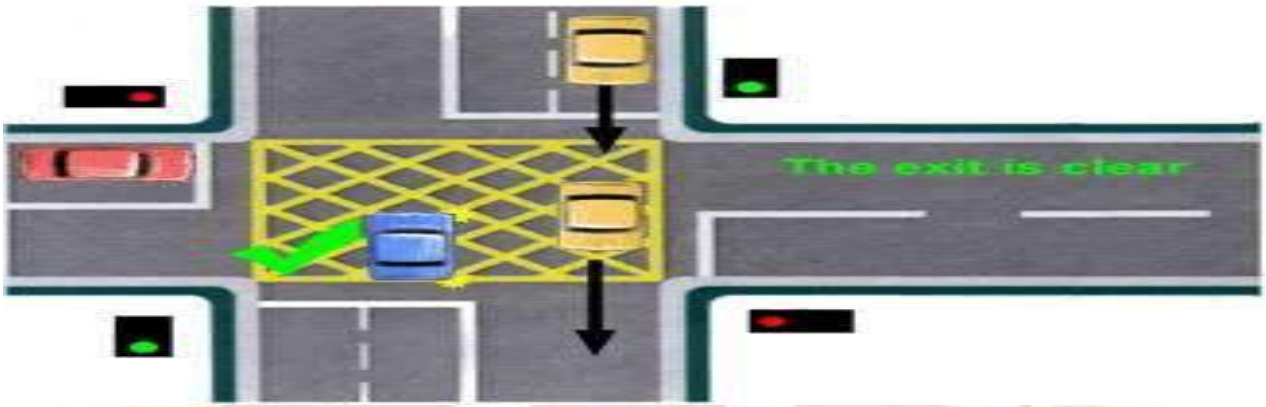
Assume Operating Frequency = 2 MHz

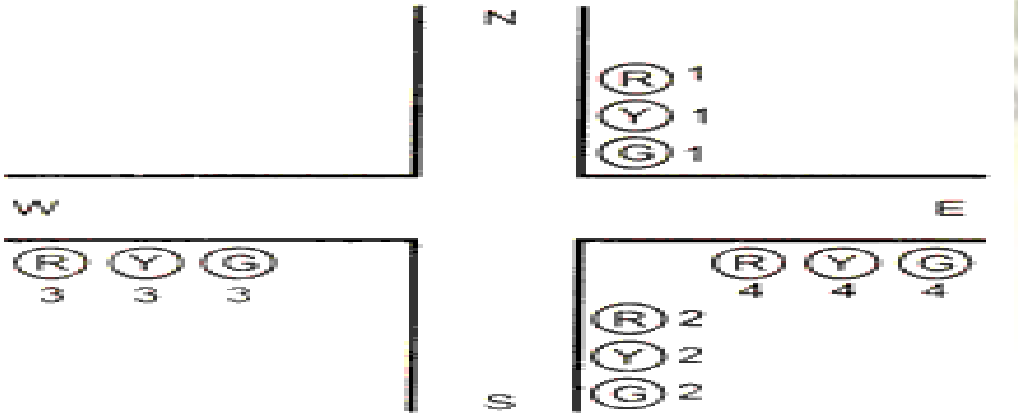
$$\text{Time for one T - state} = \frac{1}{2\text{MHz}} = 0.5\mu\text{sec}$$

$$\text{Count} = \frac{\text{Required Delay}}{\text{Time required for 1 loop}}$$

$$= \frac{0.5\,\text{sec}}{0.5\,\mu\text{sec} \times 24} \because 1 \text{ loop needs } 24 \text{ T-states}$$

$$= 41666_{10} = \text{A2C2H}$$

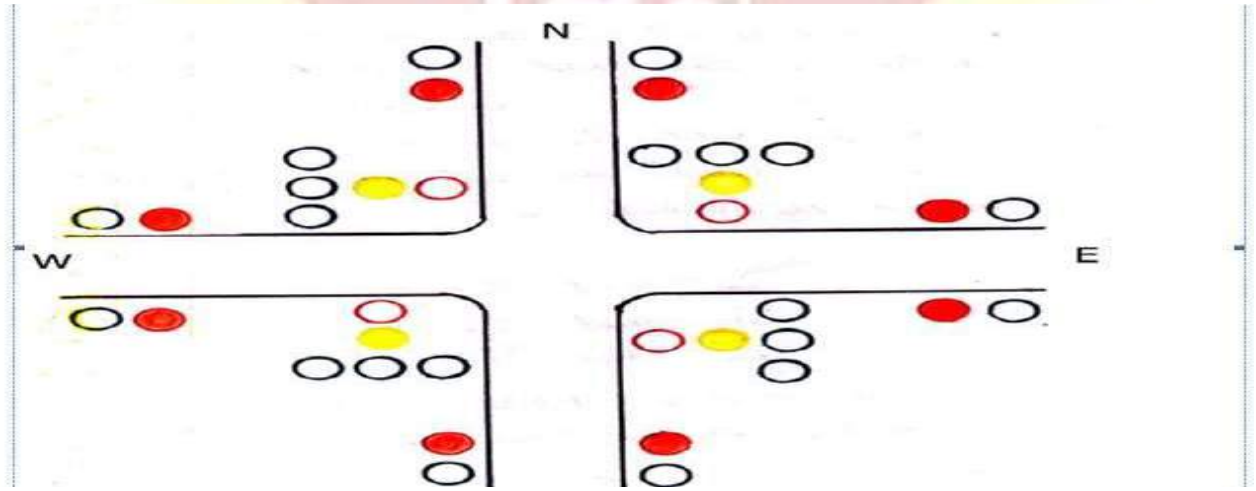




Source program:

```
    MVI A, 80H          : Initialize 8255, port A and port B
    OUT 83H (CR)        : in output mode
START: MVI A, 09H
    OUT 80H (PA)        : Send data on PA to glow R1 and R2
    MVI A, 24H
```

```
        OUT 81H (PB)        : Send data on PB to glow G3 and G4
        MVI C, 28H          : Load multiplier count (40ıo) for delay
        CALL DELAY          : Call delay subroutine
        MVI A, 12H
        OUT (81H) PA        : Send data on Port A to glow Y1 and Y2
        OUT (81H) PB        : Send data on port B to glow Y3 and Y4
        MVI C, 0AH          : Load multiplier count (10ıo) for delay
CALL: DELAY                 : Call delay subroutine
        MVI A, 24H
        OUT (80H) PA        : Send data on port A to glow G1 and G2
        MVI A, 09H
        OUT (81H) PB        : Send data on port B to glow R3 and R4
        MVI C, 28H          : Load multiplier count (40ıo) for delay
        CALL DELAY          : Call delay subroutine
        MVI A, 12H
        OUT PA              : Send data on port A to glow Y1 and Y2
        OUT PB              : Send data on port B to glow Y3 and Y4
        MVI C, 0AH          : Load multiplier count (10ıo) for delay
        CALL DELAY          : Call delay subroutine
        JMP START
```



*****