DEMONSTRATION

on

DIGITAL COMPUTER FLATFORMS **LAB**

(III EEE-II Sem)

By

Mrs. K.J.POORNIMA, M.Tech

Assistant Professor

DEPT.OF E.C.E,

VEMU IT CHITTOOR

Topicstobediscussed

- Vision,Mission, POs,PSOs&PEOs
- Syllabus&CourseOutcomes(COs)
- OverviewofMicroProcessorandMicro-Controller
- CodeComposerStudio

(CCS-V6.1using(MSP430Controller)

- MajorEquipmentList
- LabPhysicalView
- Dos &Don't
- SafetyPrecautions

Vision,Mission,POs,PSOs&PEOs

Visionoftheinstitute

To be a premier institute for professional education producing dynamic and vibrant force of technocrat with competent skills, innovative ideas and leadership qualities to serve the society with ethical and benevolent approach.

Missionoftheinstitute

Mission_1: To create a learning environment with state-of-the art infrastructure, well equipped laboratories, research facilities and qualified senior faculty to impart high quality technical education.

Mission_2: To facilitate the learners to foster innovative ideas, inculcate competent research and consultancy skills through Industry-Institute Interaction.

Mission_3: To develop hard work, honesty, leadership qualities and sense of direction in rural youth by providing value based education.

VisionoftheDepartment

To become a centre of excellence in the field of Electronics and Communication Engineering and produce graduates with Technical Skills, Research & Consultancy Competencies, Life-long Learning and Professional Ethics to meet the challenges of the Industry and evolving Society.

Missionof theDepartment

Mission_1: To enrich Technical Skills of students throughEffectiveTeachingandLearningpractices for exchange of ideas and dissemination of knowledge.

Mission_2: To enable the students with research and consultancy skill sets through state-of-the art laboratories, industry interaction and training on core & multidisciplinary technologies.

Mission_3: To develop and instill creative thinking, Life-long learning, leadership qualities, Professional Ethics and social responsibilities among students by providing value based education.

ProgrammeEducationalObjectives(PEOs)

PEO_1: To prepare the graduates to be able to plan, analyzeandprovideinnovativeideasto investigate complex engineering problems of industry in the field of Electronics and Communication Engineering using contemporary design and simulation tools.

PEO_2: Toprovidestudentswithsolidfundamentalsin core and multidisciplinary domain for successful implementation of engineering products and also to pursue higher studies.

PEO_3: To inculcatelearners with professionaland ethical attitude, effective communicationskills, teamworkskills,andanabilitytorelateengineering issues to broader social context at work place.

ProgrammeOutcome(POs)

PO_1:Engineeringknowledge: Applytheknowledgeof mathematics,science,engineering fundamentals,and anengineeringspecializationtothesolutionofcomplexengineeringproblems.

PO_2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO_3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health andsafety,and the cultural, societal, and environmental considerations.

PO_4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO_5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understandingof the limitations.

PO_6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO_7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO_8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO_9:Individualandteamwork: Functioneffectivelyasanindividual,andasamemberorleaderindiverse teams,andinmultidisciplinarysettings.

PO_10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO_11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles andapplythese to one's own work,as amemberandleader ina team, to manage projects and in multidisciplinary environments.

PO_12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context oftechnological change.

ProgrammeSpecificOutcome(PSOs)

PSO_1: Higher Education: Qualify in competitive examinations for pursuing higher education by applying the fundamental concepts of Electronics and Communication Engineering domains suchas Analog & Digital Electronics, Signal Processing, Communication & Networking, Embedded Systems, VLSI Design and Control Systems etc..

PSO_2: Employment: Get employed in allied industries through their proficiency in program specific domain knowledge, specialized software packages and Computer programming or become an entrepreneur.

Syllabus&CourseOutcomes(COs)

JAWAHARLALNEHRUTECHNOLOGICALUNIVERSITYANANTAPUR

IIIB.Tech.II-Sem(ECE)

(20A02602P)DIGITALCOMPUTERFLATFORMSLABORATORY

DEPTOFELECTRONICSANDCOMMUNICATIONENGINEERING

CourseOutcomes:

CO1.Write andExecute differentprograms8086,8051&ARMCortex M0assemblylevel languages using MASM.assembler.

CO2.Designandimplementsomespecificrealtimeapplications.

Listof Experiments:

Intel8086(16bitMicroProcessor)

1. Programsfor16bitarithmeticoperationsfor8086(usingvariousaddressingmodes).
2. Programforsortinganarrayfor8086.
3. Programforsearchingforanumberorcharacterinastringfor8086.
4. ProgramforStringmanipulations for8086.

1. InterfacingADCandDACto8086.
2. Parallelcommunicationbetweentwomicroprocessorsusing8255.
3. Serialcommunicationbetweentwomicroprocessorkitsusing8251.
4. Interfacingto8086andprogrammingtocontrolsteppermotor.
5. Programmingusingarithmetic,logicalandbitmanipulationinstructionsof8051
6. ProgramandverifyTimer/Counterin8051.
7. Programandverifyinterrupthandlingin8051.
8. UARToperationin8051.
9. Communicationbetween8051kitandPC.
10. InterfacingLCDto8051.
11. Interfacingmatrixorkeyboardto8051.

Note:UseMASM/8086microprocessorkit

1. IntroductiontoMASMProgramming.
2. Programsusingarithmeticandlogicaloperations.
3. ProgramsusingstringoperationsandInstructionprefix:Move Block,Reversestring,sorting, String comparison.
4. Programs forcode conversion.
5. MultiplicationandDivisionprograms.
6. Sortingandmultibytearithmetic.
7. ProgramsusingCALLandRETinstructions.

VEMUINSTITUTEOFTECHNOLOGY::P.KOTHAKOTA

DepartmentofElectronics&CommunicationEngineering)
NEARPAKALA,CHITTOOR-517112
(ApprovedbyAICTE,NewDelhi &AffiliatedtoJNTUA,Anantapuramu
(19A02601P)DIGITALCOMPUTERFLATFORMSLAB

LISTOFEXPERIMENTSTOBECONDUCTED

DCPLab

Listof Experiments:

PARTA:

LISTOFPROGRAMS USING MASAM/ALP:

1. **Programsfor16bitarithmeticoperationsfor8086(usingvariousaddressing modes).**
2. **Programforsorting anarray for8086.**
3. **Programforsearchingforanumberorcharacterinastringfor8086.**
4. **ProgramforStringmanipulationsfor8086.**

PART-B:

Listofexperimentsusing8086and8051modules:

1. **InterfacingADCandDACto 8086.**
2. **Parallelcommunication betweentwomicroprocessors using8255.**
3. **Interfacing to 8086and programming to control stepper motor.**
4. **Programmingusingarithmetic,logicalandbitmanipulationinstructionsof8051**

Additionalexperiments(BeyondCurriculum)

1. 8259-Interruptcontrolleranditsinterfacing program with 8086
2. Programming usingarithmetic,logicalandbit manipulation instructions of 8051

WHATisMicroProcessor?

- **Program controlled semiconductordevice(IC)which fetches (from memory), decodes and executes instructions.**

- **It is used as CPU(Central ProcessingUnit)incomputers.**

FifthGenerationPentium

FourthGeneration
During1980s
LowpowerversionofHMOStechnology (HCMOS)
32bitprocessors
Physical memoryspace2^{24}bytes=16Mb Virtual memory space 2^{40}bytes = 1 Tb Floating point hardware
Supportsincreasednumberofaddressing modes

Intel80386

SecondGeneration
During1973
NMOStechnology⇒Fasterspeed,Higher density, Compatible with TTL
4 / 8/ 16 bit processors ⇒40 pins
Abilitytoaddresslargememoryspaces and I/O ports
Greaternumberoflevelsofsubroutine nesting
Betterinterrupthandlingcapabilities

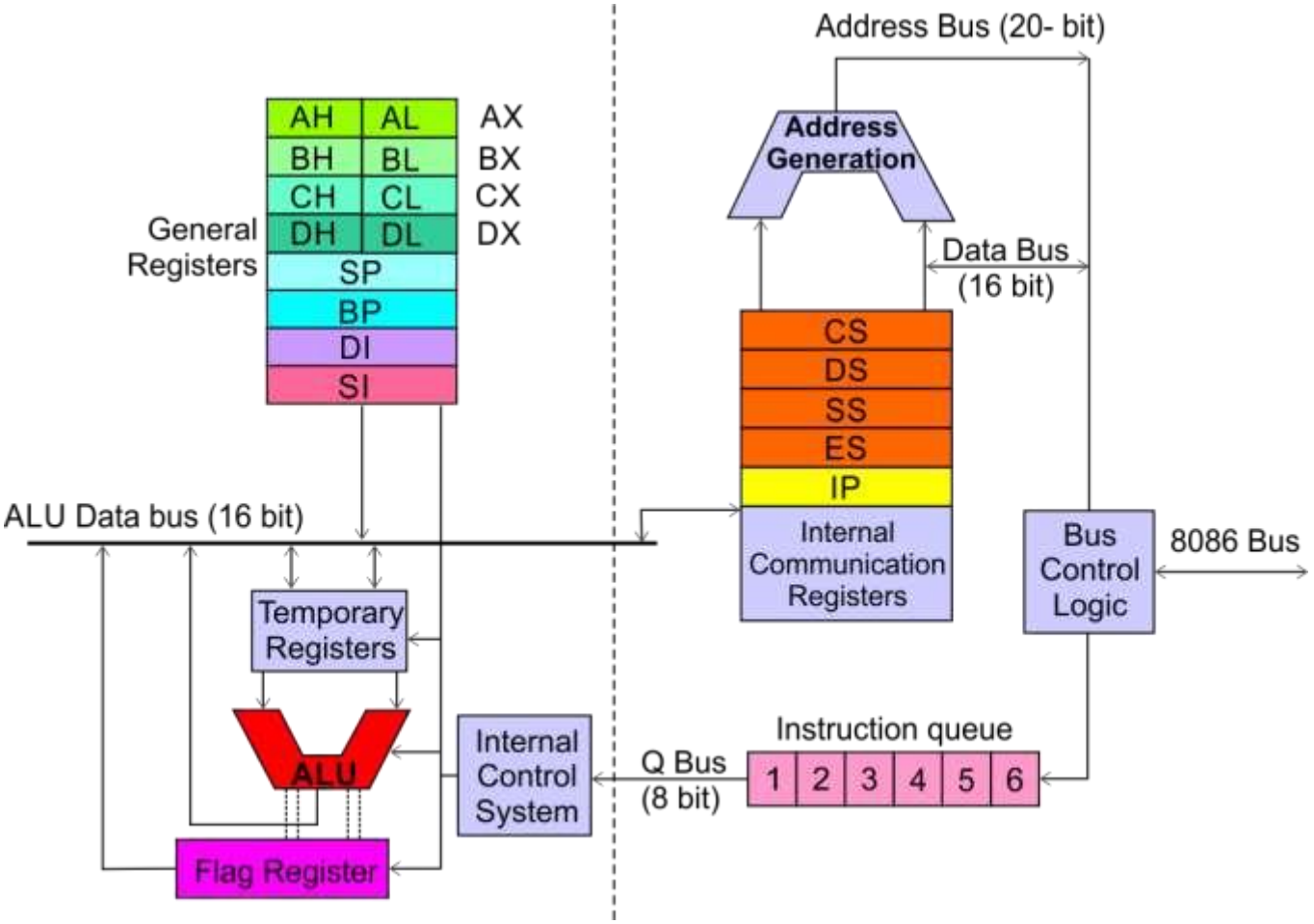
Intel8085(8bitprocessor)

ThirdGeneration
During1978
HMOStechnology⇒Fasterspeed,Higher packingdensity
16bit processors⇒40/48/64pins
Easiertoprogram
Dynamicallyrelatable programs
Processorhasmultiply/dividearithmetic hardware
Morepowerfulinterrupthandling capabilities
FlexibleI/Oportaddressing

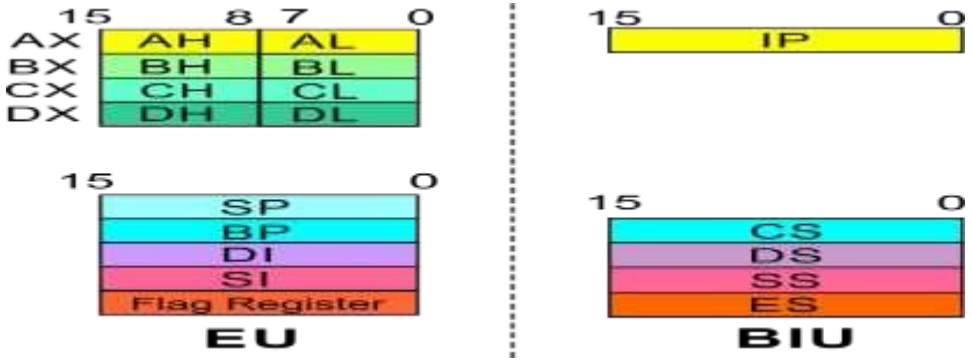
Intel8086(16bitprocessor)

FirstGeneration
Between1971–1973
PMOStechnology,noncompatiblewithTTL
4bitprocessors⇒16pins
8and16bitprocessors ⇒40pins
Duetolimitationsofpins,signals are multiplexed

ARCHITECTUREOF8086



REGISTERORGANIZATION



Sl.No.	Type	Registerwidth	Nameofregister
1	Generalpurposeregister	16bit	AX,BX,CX,DX
		8bit	AL,AH,BL,BH,CL,CH,DL,DH
2	Pointerregister	16bit	SP,BP
3	Indexregister	16bit	SI, DI
4	InstructionPointer	16bit	IP
5	Segmentregister	16bit	CS,DS,SS,ES
6	Flag(PSW)	16bit	Flagregister

INSTRUCTIONSET

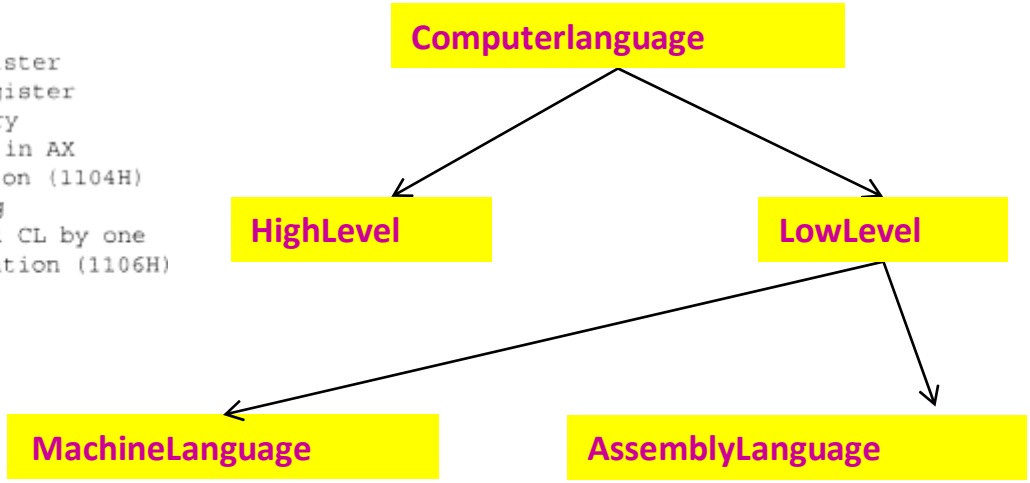
```
;PROGRAM TO ADD TWO 16-BIT DATA (METHOD-1)

DATA SEGMENT              ;Assembler directive

        ORG 1104H         ;Assembler directive
        SUM DW 0          ;Assembler directive
        CARRY DB 0        ;Assembler directive

DATA ENDS                 ;Assembler directive

CODE SEGMENT              ;Assembler directive

        ASSUME CS:CODE    ;Assembler directive
        ASSUME DS:DATA    ;Assembler directive
        ORG 1000H         ;Assembler directive

        MOV AX,205AH      ;Load the first data in AX register
        MOV BX,40EDH      ;Load the second data in BX register
        MOV CL,00H        ;Clear the CL register for carry
        ADD AX,BX         ;Add the two data, sum will be in AX
        MOV SUM,AX        ;Store the sum in memory location (1104H)
        JNC AHEAD         ;Check the status of carry flag
        INC CL            ;If carry flag is set,increment CL by one
AHEAD:  MOV CARRY,CL      ;Store the carry in memory location (1106H)
        HLT

CODE ENDS                 ;Assembler directive
END                       ;Assembler directive
```

Program

Asetofinstructionswrittentosolveaproblem.

Instruction

Directions which a microprocessor followstoexecuteataskorpartofa task.



- **Binarybits**

- **EnglishAlphabets**
- **'Mnemonics'**
- **AssemblerMnemonics→ MachineLanguage**

1.DataTransferInstructions

Instructionsthatareusedtotransferdata/addressintoregisters,memory locationsandI/Oports.

Generallyinvolvetwooperands:SourceoperandandDestinationoperandofthe samesize.

Source:Registeroramemorylocationoranimmediatedata Destination : Register or a memory location.

Thesizeshouldbeaeitherabyteoraword.

A8-bitdatacanonlybemovedto8-bitregister/memoryanda16-bitdatacanbe moved to 16-bit register/ memory.

1.DataTransferInstructions

Mnemonics:MOV,XCHG,PUSH,POP,IN,OUT…

MOVreg2/mem,reg1/mem MOV reg2, reg1 MOVmem,reg1 MOVreg2, mem	 (reg2)←(reg1) (mem)←(reg1) (reg2)←(mem)
MOVreg/mem,data MOVreg,data MOVmem,data	 (reg)←data (mem)←data

XCHGreg2/mem,reg1 XCHGreg2,reg1 XCHGmem,reg1	 (reg2)↔(reg1) (mem)↔(reg1)

1. DataTransferInstructions

Mnemonics: MOV,XCHG,PUSH,POP,IN,OUT…

PUSHreg16/mem	
PUSH reg16	**(SP)←(SP)–2** **MA $_S$= (SS) x 16_{10} +** **SP(MA$_S$;MA$_S$+1)←(reg16)**
PUSHmem	**(SP)←(SP)–2** **MA $_S$= (SS) x 16_{10} + SP** **(MA$_S$;MA$_S$+1)←(mem)**
POPreg16/ mem	
POPreg16	**MA$_S$= (SS)x16_{10}+SP** **(reg16)←(MA$_S$;MA$_S$+1)** **(SP)←(SP)+2**
POPmem	**MA$_S$= (SS)x16_{10}+SP** **(mem) ←(MA $_S$;MA$_S$+1)** **(SP)←(SP)+2**

1. DataTransferInstructions

Mnemonics:MOV,XCHG,PUSH,POP,IN,OUT…

INA,[DX]	
IN AL,[DX]	$PORT_{addr}$=(DX) (AL)←(PORT)
INAX,[DX]	$PORT_{addr}$=(DX) (AX)←(PORT)
IN A, addr8	
INAL,addr8	(AL)←(addr8)
INAX,addr8	(AX)←(addr8)

OUT [DX], A	
OUT[DX],AL	$PORT_{addr}$ =(DX) (PORT) ←(AL)
OUT[DX],AX	$PORT_{addr}$=(DX) (PORT)←(AX)
OUT addr8, A	
OUTaddr8, AL	(addr8)←(AL)
OUTaddr8,AX	(addr8)←(AX)

2. ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

ADDreg2/mem,reg1/mem ADC reg2, reg1 ADCreg2,mem ADCmem, reg1	 (reg2) ←(reg1) + (reg2) (reg2)←(reg2)+(mem) (mem)←(mem)+(reg1)
ADDreg/mem,data ADDreg, data ADDmem,data	 (reg)←(reg)+data (mem)←(mem)+data
ADDA, data ADDAL, data8 ADDAX,data16	 (AL)←(AL) + data8 (AX)←(AX)+data16

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

ADCreg2/mem,reg1/mem ADC reg2, reg1 ADCreg2,mem ADCmem, reg1	 (reg2) ←(reg1) + (reg2)+CF (reg2)←(reg2)+(mem)+CF (mem)←(mem)+(reg1)+CF
ADCreg/mem, data ADCreg,data ADCmem, data	 (reg)←(reg)+data+CF (mem)←(mem)+data+CF
ADDCA, data ADDAL, data8 ADDAX, data16	 (AL)←(AL) +data8+CF (AX)←(AX)+data16+CF

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

SUBreg2/mem,reg1/mem SUB reg2, reg1 SUBreg2,mem SUBmem,reg1	 (reg2)←(reg1)-(reg2) (reg2)←(reg2)-(mem) (mem)←(mem)-(reg1)
SUBreg/mem,data SUBreg,data SUBmem,data	 (reg)←(reg)-data (mem)←(mem)-data
SUBA,data SUBAL,data8 SUBAX,data16	 (AL)←(AL) -data8 (AX)←(AX)-data16

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

SBBreg2/mem,reg1/mem SBB reg2, reg1 SBBreg2,mem SBBmem,reg1	 (reg2)←(reg1)-(reg2)-CF (reg2)←(reg2)-(mem)-CF (mem)←(mem)-(reg1) –CF
SBBreg/mem, data SBBreg,data SBBmem,data	 (reg)←(reg)–data-CF (mem)←(mem) -data-CF
SBBA,data SBBAL, data8 SBBAX,data16	 (AL)←(AL)-data8- CF (AX)←(AX)-data16- CF

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

INCreg/mem INC	
reg8	(reg8) ←(reg8) + 1
INCreg16	(reg16)←(reg16)+1
INC mem	(mem) ←(mem) + 1
DECreg/mem	
DEC reg8	(reg8) ←(reg8) - 1
DECreg16	(reg16)←(reg16)-1
DEC mem	(mem) ←(mem) - 1

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

MULreg/mem MUL	
reg	Forbyte:(AX)←(AL)x(reg8) Forword:(DX)(AX)←(AX)x(reg16)
MULmem	Forbyte:(AX)←(AL)x(mem8) Forword:(DX)(AX)←(AX)x(mem16)
IMULreg/mem	
IMUL reg	Forbyte:(AX)←(AL)x(reg8) Forword:(DX)(AX)←(AX)x(reg16)
IMULmem	Forbyte:(AX)←(AX)x(mem8) Forword:(DX)(AX)←(AX)x(mem16)

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

DIVreg/mem DIV	
reg	For16-bit:-8-bit: (AL) ←(AX) :- (reg8)Quotient(AH)←(AX)MOD(reg 8)Remainder For32-bit:-16-bit: (AX)←(DX)(AX):-(reg16)Quotient (DX)←(DX)(AX)MOD(reg16)Remainder
DIV mem	For16-bit:-8-bit: (AL)←(AX):-(mem8)Quotient (AH)←(AX)MOD(mem8) Remainder For32-bit:-16-bit: (AX)←(DX)(AX):-(mem16) Quotient (DX)←(DX)(AX)MOD(mem16)Remainder

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

IDIVreg/mem	
IDIV reg	For16-bit:-8-bit: (AL) ←(AX) :- (reg8)Quotient(AH)←(AX)MOD(reg 8)Remainder For32-bit:-16-bit: (AX)←(DX)(AX):-(reg16)Quotient (DX)←(DX)(AX)MOD(reg16)Remainder
IDIVmem	For16-bit:-8-bit: (AL)←(AX):-(mem8)Quotient (AH)←(AX)MOD(mem8) Remainder For32-bit:-16-bit: (AX)←(DX)(AX):-(mem16) Quotient (DX)←(DX)(AX)MOD(mem16)Remainder

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

CMPreg2/mem,reg1/mem CMP	

reg2, reg1	Modifyflags←(reg2)–(reg1)
	
	If(reg2)>(reg1)thenCF=0,ZF=0,SF=0
	If(reg2)<(reg1)thenCF=1,ZF=0,SF=1
	If(reg2)=(reg1)thenCF=0, ZF=1,SF=0

CMPreg2,mem	Modifyflags←(reg2)–(mem)
	
	If(reg2)>(mem)thenCF=0,ZF=0,SF=0
	If(reg2)<(mem)thenCF=1,ZF=0,SF=1
	If(reg2)=(mem)thenCF=0,ZF=1, SF=0

CMPmem, reg1	Modifyflags←(mem)–(reg1)
	
	If(mem)>(reg1)thenCF=0,ZF=0,SF=0
	If(mem)<(reg1)thenCF=1,ZF=0,SF=1
	If(mem)=(reg1)thenCF=0,ZF=1, SF=0

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

CMPreg/mem,data CMP	

reg, data	Modifyflags←(reg)–(data)
	
	If(reg)>datathenCF=0,ZF=0,SF=0
	If(reg)<datathenCF=1,ZF=0,SF=1
	If(reg)=datathenCF=0,ZF=1,SF=0

CMPmem, data	Modifyflags←(mem)–(mem)
	
	If(mem)>datathenCF=0,ZF=0,SF=0
	If(mem)<datathenCF=1,ZF=0,SF=1
	If(mem)=datathenCF=0,ZF=1,SF=0

2.ArithmeticInstructions

Mnemonics:ADD,ADC,SUB,SBB,INC,DEC,MUL,DIV,CMP…

CMP A, data	

CMPAL,data8	Modify flags←(AL)–data8
	
	If(AL)>data8thenCF=0,ZF=0,SF=0
	If(AL)<data8thenCF=1,ZF=0,SF=1
	If(AL)=data8thenCF=0,ZF=1,SF=0

CMPAX,data16	Modifyflags←(AX)–data16
	
	If(AX)>data16 thenCF=0,ZF=0,SF=0
	If(mem)<data16thenCF=1,ZF=0, SF=1
	If(mem)=data16thenCF=0,ZF=1,SF=0

3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR,RCL…

AND A, data AND AL, data8	(AL) ← (AL) & data8
AND AX, data16	(AX) ← (AX) & data16

AND reg/mem, data AND reg, data	(reg) ← (reg) & data
AND mem, data	(mem) ← (mem) & data

3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR,RCL…

OR reg2/mem, reg1/mem OR reg2, reg1	(reg2) ← (reg2) | (reg1)
OR reg2, mem	(reg2) ← (reg2) | (mem)
OR mem, reg1	(mem) ← (mem) | (reg1)

OR reg/mem, data OR reg, data	(reg) ← (reg) | data
OR mem, data	(mem) ← (mem) | data

OR A, data OR AL, data8	(AL) ← (AL) | data8
OR AX, data16	(AX) ← (AX) | data16

3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR,RCL…

XOR reg2/mem, reg1/mem	
XOR reg2, reg1	(reg2) ← (reg2) ^ (reg1)
XOR reg2, mem	(reg2) ← (reg2) ^ (mem)
XOR mem, reg1	(mem) ← (mem) ^ (reg1)

XOR reg/mem, data	
XOR reg, data	(reg) ← (reg) ^ data
XOR mem, data	(mem) ← (mem) ^ data

XOR A, data	
XOR AL, data8	(AL) ← (AL) ^ data8
XOR AX, data16	(AX) ← (AX) ^ data16

3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR,RCL…

TEST reg2/mem, reg1/mem TEST reg2, reg1	Modify flags ← (reg2) & (reg1)
TEST reg2, mem	Modify flags ← (reg2) & (mem)
TEST mem, reg1	Modify flags ← (mem) & (reg1)

TEST reg/mem, data TEST reg, data	Modify flags ← (reg) & data
TEST mem, data	Modify flags ← (mem) & data

TEST A, data TEST AL, data8	Modify flags ← (AL) & data8
TEST AX, data16	Modify flags ← (AX) & data16

3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR,RCL…

SHR reg/mem

SHR reg

i) SHR reg, 1

ii) SHR reg, CL

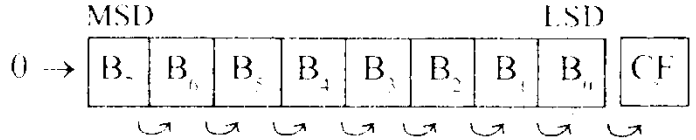
SHR mem

i) SHR mem, 1

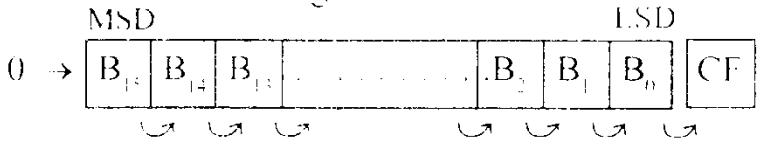
ii) SHR mem, CL

$CF \leftarrow B_{LSD} : B_{n} \leftarrow B_{n+1} : B_{MSD} \leftarrow 0$

reg 8 / mem 8



reg 16 / mem 16



3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR,RCL…

SHL reg/mem or SAL reg/mem

SHL reg or SAL reg

i) SHL reg, 1 or SAL reg, 1

ii) SHL reg, CL or SAL reg, CL

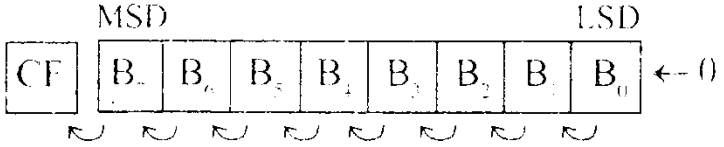
SHL mem or SAL mem

i) SHL mem, 1 or SAL mem, 1

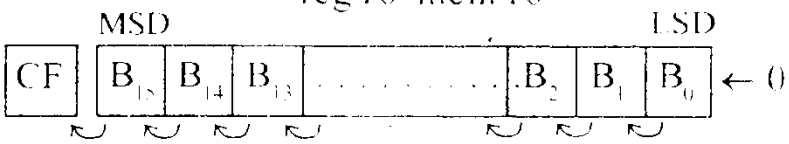
ii) SHL mem, CL or SAL mem, CL

$$CF \leftarrow B_{MSD} ; B_{n-1} \leftarrow B_n ; B_{LSD} \leftarrow 0$$

reg 8 / mem 8



reg 16 / mem 16



3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR, RCL…

RCR reg/mem

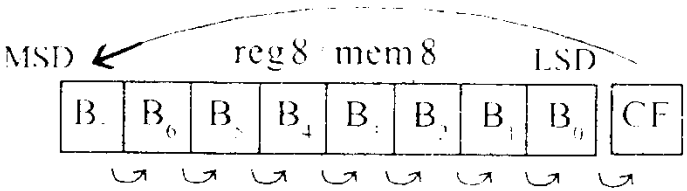
RCR reg

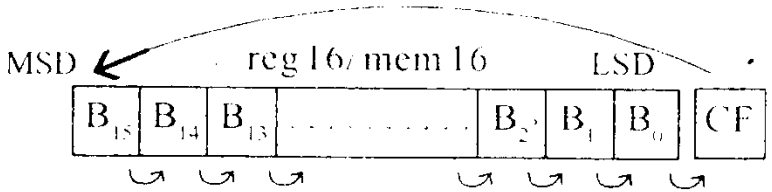
i) RCR reg, 1
ii) RCR reg, CL

RCR mem

i) RCR mem, 1
ii) RCR mem, CL

$B_n \leftarrow B_{n-1}$; $B_{MSD} \leftarrow CF$; $CF \leftarrow B_{LSD}$





3.Logical Instructions

Mnemonics:AND,OR,XOR,TEST,SHR,SHL,RCR,RCL…

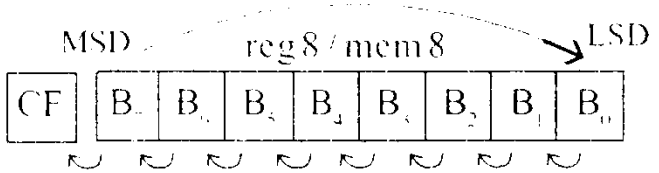
ROL reg/mem

ROL reg

i) ROL reg, 1

ii) ROL reg, CL

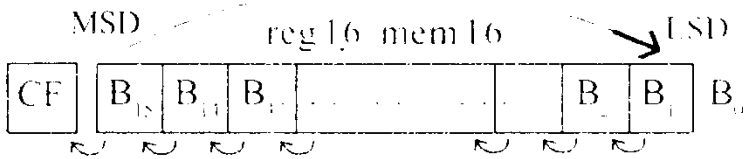
$B_{n+1} \leftarrow B_n ; CF \leftarrow B_{MSD} ; B_{LSD} \leftarrow B_{MSD}$



ROL mem

i) ROL mem, 1

ii) ROL mem, CL



4.StringManipulationInstructions

- String:Sequenceofbytesorwords
- 8086instructionsetincludesinstructionforstringmovement,comparison,scan,loadandstore.
- REPinstructionprefix:usedtorepeatexecutionofstringinstructions
- StringinstructionsendwithSorSBorSW. Srepresentsstring,SBstring byteandSWstringword.
- Offsetoreffectiveaddressofthesourceoperandisstoredin SIregisterandthatofthedestination operand is stored in DI register.
- DependingonthestatusofDF,SIandDIregistersareautomaticallyupdated.
- DF=0⇒SIandDIareincrementedby1forbyteand2forword.
- DF=1⇒SIandDIaredecrementedby1forbyteand2for word.

4.StringManipulationInstructions

Mnemonics:REP,MOVS,CMPS,SCAS,LODS,STOS

REP	
REPZ/REPE **(RepeatCMPSorSCASuntilZF=0)**	**WhileCX≠0andZF=1,repeatexecutionofstringinstruction and (CX)←(CX) –1**
REPNZ/REPNE **(RepeatCMPSorSCASuntilZF=1)**	**WhileCX≠0andZF=0,repeatexecutionofstring instruction and (CX) ←(CX) -1**

4.StringManipulationInstructions

Mnemonics:REP,MOVS,CMPS,SCAS,LODS,STOS

MOVS	
MOVSB	**MA=(DS)x16$_{10}$+(SI)** **MA$_E$=(ES)x16$_{10}$+(DI)** **(MA$_E$)←(MA)** **IfDF=0,then(DI)←(DI)+ 1;(SI)←(SI)+1** **IfDF=1,then(DI)←(DI)-1;(SI)←(SI)-1**
MOVSW	**MA=(DS)x16$_{10}$+(SI)** **MA$_E$=(ES)x16$_{10}$+(DI)** **(MA$_E$; MA$_E$+1) ←(MA;MA+ 1)** **IfDF=0,then(DI) ←(DI)+2;(SI)←(SI) +2 If DF= 1, then (DI) ←(DI) - 2;(SI) ←(SI) - 2**

4.StringManipulationInstructions

Mnemonics:REP,MOVS,CMPS,SCAS,LODS,STOS

Comparetwostringbyteorstringword

CMPS	
CMPSB	**MA=(DS)x16$_{10}$+(SI)** **MA$_E$=(ES)x16$_{10}$+(DI)** **Modifyflags←(MA)-(MA$_E$)** **If(MA)>(MA$_E$), thenCF=0; ZF=0; SF=0** **If(MA)<(MA$_E$), thenCF=1; ZF=0; SF=1**
CMPSW	**If(MA)=(MA$_E$),thenCF=0;ZF=1;SF=0** **<u>Forbyteoperation</u>** **IfDF=0,then(DI) ←(DI)+1;(SI)←(SI) +1 If DF= 1, then (DI) ←(DI) - 1;(SI) ←(SI) - 1** **<u>Forwordoperation</u>** **IfDF=0,then(DI) ←(DI)+2;(SI)←(SI) +2 If DF= 1, then (DI) ←(DI) - 2;(SI) ←(SI) - 2**

4.StringManipulationInstructions

Mnemonics:REP,MOVS,CMPS,SCAS,LODS,STOS

Scan(compare)astringbyteorwordwithaccumulator

SCAS	

SCASB	$MA_E=(ES)x16_{10}+(DI)$
	Modifyflags←(AL)-(MA_E)
	
	If(AL)>(MA_E),thenCF=0;ZF=0;SF =0
	If(AL)<(MA_E),thenCF=1;ZF=0;SF =1
	If(AL)=(MA_E),thenCF=0;ZF=1;SF=0
	
	IfDF=0,then(DI)←(DI)+1
	IfDF=1,then(DI)←(DI)–1

SCASW	$MA_E=(ES)x16_{10}+(DI)$
	Modifyflags←(AL)-(MA_E)
	
	If(AX)>$(MA_E;MA_E+1)$,thenCF=0;ZF=0;SF =0
	If(AX)<$(MA_E;MA_E+1)$,thenCF=1;ZF= 0;SF =1 If(AX) =$(MA_E;MA_E+1)$, then CF=0; ZF=1;SF=0
	
	IfDF=0,then(DI) ←(DI)+2
	IfDF=1,then(DI)←(DI)–2

4.StringManipulationInstructions

Mnemonics:REP,MOVS,CMPS,SCAS,LODS,STOS

LoadstringbyteintoALorstringwordintoAX

LODS	
LODSB	MA=(DS)x16_{10}+(SI) (AL) ←(MA) IfDF=0,then(SI) ←(SI) +1 IfDF=1,then(SI)←(SI)–1
LODSW	MA=(DS)x16_{10}+(SI) (AX)←(MA ;MA+1) IfDF=0,then(SI)←(SI)+2 IfDF=1,then(SI)←(SI)–2

4. StringManipulationInstructions

Mnemonics: REP,MOVS,CMPS,SCAS,LODS,STOS

StorebytefromALorwordfromAXintostring

STOS	
STOSB	MA_E=(ES) $x16_{10}$+(DI) (MA_E) ←(AL) IfDF=0,then(DI) ←(DI)+1 IfDF=1,then(DI)←(DI)–1
STOSW	MA_E=(ES)$x16_{10}$+(DI) (MA_E;MA_E+1)←(AX) IfDF=0,then(DI)←(DI)+2 IfDF=1,then(DI)←(DI)–2

5. ProcessorControlInstructions

Mnemonics	Explanation
STC	SetCF←1
CLC	ClearCF←0
CMC	Complement carryCF←CF/
STD	SetdirectionflagDF←1
CLD	CleardirectionflagDF←0
STI	SetinterruptenableflagIF←1
CLI	ClearinterruptenableflagIF←0
NOP	Nooperation
HLT	Haltafterinterruptisset
WAIT	WaitforTESTpinactive
ESCopcodemem/reg	Usedtopassinstruction toacoprocessor whichsharesthe address and data bus with the 8086
LOCK	Lockbusduringnextinstruction

6. ControlTransferInstructions

- Transferthecontroltoaspecificdestinationortargetinstruction
  Do not affect flags

- 8086Unconditionaltransfers

Mnemonics	Explanation
CALLreg/mem/disp16	Callsubroutine
RET	Returnfromsubroutine
JMPreg/mem/disp8/ disp16	Unconditionaljump

6.ControlTransferInstructions

- 8086signedconditionalbranch instructions
- 8086unsignedconditionalbranch instructions

- Checksflags
- If conditions are true, the program control is transferred to the new memorylocationinthesamesegmentbymodifyingthecontentofIP

6.ControlTransferInstructions

- 8086signedconditionalbranch instructions

Name	Alternatename
JE disp8 Jumpifequal	JZdisp8 Jumpifresultis0
JNEdisp8 Jumpifnotequal	JNZdisp8 Jumpifnotzero
JGdisp8 Jumpifgreater	JNLEdisp8 Jumpifnotlessorequal
JGEdisp8 Jumpifgreaterthanor equal	JNLdisp8 Jumpifnotless
JLdisp8 Jumpifless than	JNGEdisp8 Jumpifnotgreaterthan or equal
JLEdisp8 Jumpiflessthanor equal	JNGdisp8 Jumpifnotgreater

- 8086unsignedconditionalbranch instructions

Name	Alternatename
JE disp8 Jumpifequal	JZdisp8 Jumpifresultis0
JNEdisp8 Jumpifnotequal	JNZdisp8 Jumpifnotzero
JA disp8Jumpifa bove	JNBEdisp8 Jumpifnotbelowor equal
JAEdisp8 Jumpifaboveorequal	JNBdisp8 Jumpifnot below
JBdisp8 Jumpifbelow	JNAEdisp8 Jumpifnotaboveor equal
JBEdisp8 Jumpifbeloworequal	JNAdisp8 Jumpifnotabove

6.ControlTransferInstructions

- 8086conditionalbranchinstructionsaffectingindividualflags

Mnemonics	Explanation
JCdisp8	JumpifCF= 1
JNCdisp8	JumpifCF= 0
JP disp8	JumpifPF=1
JNPdisp8	JumpifPF=0
JO disp8	JumpifOF=1
JNOdisp8	JumpifOF=0
JSdisp8	JumpifSF=1
JNSdisp8	JumpifSF=0
JZdisp8	Jumpifresultiszero,i.e, Z=1
JNZdisp8	Jumpifresultisnotzero,i.e,Z=1

AssembleDirectives

- InstructionstotheAssemblerregardingtheprogrambeingexecuted.
- Controlthegenerationofmachinecodesandorganizationoftheprogram;butno machinecodesaregeneratedforassemblerdirectives. Also
- called ‘pseudo instructions’
- Usedto:
  - ›specifythestartandendofaprogram
  - ›attachvaluetovariables
  - ›allocatestoragelocationstoinput/outputdata
  - ›definestartandendofsegments,procedures,macrosetc..

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- **DefineByte**
- **Defineabytetype(8-bit)variable**
- **Reservesspecificamountofmemorylocationstoeach variable**
- **Range : 00_H– $FF_H$for unsigned value; 00_H – $7F_H$for positive value and 80_H–FF_Hfornegativevalue**
- **Generalform:variableDBvalue/values**

Example:

```
LISTDB7FH,42H, 35H
```

Three consecutive memory locations are reserved for the variable LIST and each data specified in the instruction are stored as initial value inthe reserved memory location

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- DefineWord
- Defineawordtype(16-bit)variable
- Reservestwo consecutive memorylocations to each variable
- Range:0000_H–$FFFF_H$forunsignedvalue; 0000_H–$7FFF_H$forpositivevalueand 8000_H–$FFFF_H$for negative value
- Generalform:variableDWvalue/values

Example:

```
ALISTDW6512H,0F251H,0CDE2H
```

SixconsecutivememorylocationsarereservedforthevariableALISTand each 16-bit data specified in the instruction is stored in two consecutive memory location.

DB

DW

SEGMENT
ENDS

ASSUME

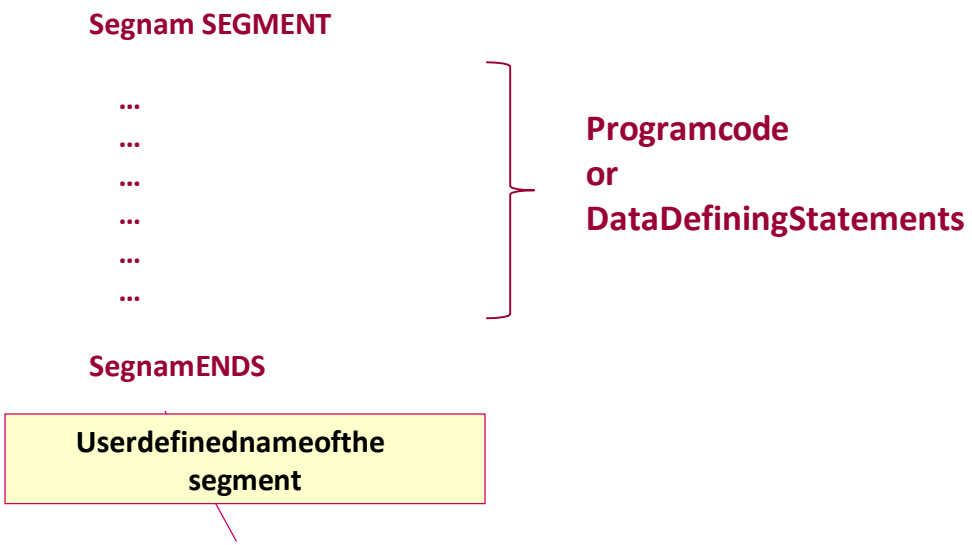
ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- **SEGMENT:Usedtoindicatethebeginningofacode/data/ stacksegment**
- **ENDS:Usedtoindicatetheendofacode/data/stack segment**
- **Generalform:**



DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

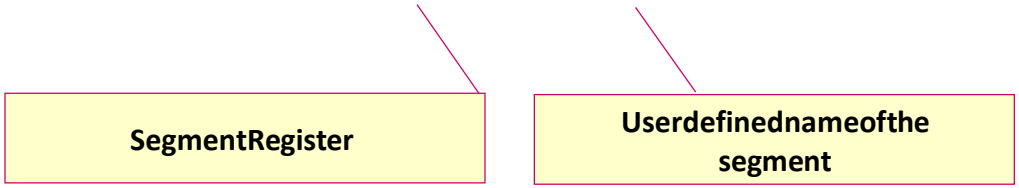
PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- **Informstheassemblerthenameoftheprogram/data segmentthatshouldbeusedforaspecificsegment.**
- **Generalform:**

ASSUMEsegreg:segnam,..,segreg:segnam



Example:

ASSUMECS:ACODE, DS:ADATA	Tells the compiler that the instructions of the program are stored in the segment ACODE and data are stored in the segment ADATA

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
FAR
NEAR
ENDP

SHORT

MACRO
ENDM

- ORG(Origin)isusedtoassignthestartingaddress(Effective address) for a program/ data segment
- ENDisusedtoterminateaprogram;statements afterENDwillbe ignored
- EVEN:Informstheassemblertostoreprogram/datasegment starting from an even address
- EQU(Equate)isusedtoattachavaluetoavariable

Examples:

ORG1000H	Informs the assembler that the statements following ORG 1000H should be stored in memory starting with effective address 1000_H
LOOPEQU10FEH	ValueofvariableLOOPis$10FE_H$
_SDATA SEGMENT ORG1200H A DB 4CH EVEN BDW1052H _SDATAENDS	In this data segment, effective address of memory location assigned to A will be $1200_H$and that of B will be $1202_H$and 1203_H.

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

- PROC Indicates the beginning of a procedure
- ENDP End of procedure
- FAR Intersegment call
  NEAR Intrasegment call
- General form

```
procnamePROC[NEAR/FAR]
        …
        …
        …
        RET

procname ENDP
```

Programstatementsoftheprocedure

Last statement of the procedure

Userdefinednameofthe procedure

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

Examples:

ADD64PROCNEAR … … … RET ADD64ENDP	The subroutine/ procedure named ADD64 is declared as NEAR and so the assembler will code the CALL and RET instructions involved in this procedure as near call andreturn
CONVERTPROCFAR … … … RET CONVERTENDP	The subroutine/ procedure named CONVERT is declared as FAR and so the assembler will code the CALL and RET instructions involved in this procedure as far call and return

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

- **Reservesone memorylocation for 8-bitsigneddisplacement in jump instructions**

Example:

JMPSHORTAHEAD	Thedirectivewillreserveonememorylocation for8-bitdisplacement namedAHEAD

DB

DW

SEGMENT
ENDS

ASSUME

ORG
END
EVEN
EQU

PROC
ENDP
FAR
NEAR

SHORT

MACRO
ENDM

- MACROIndicatethebeginningofamacro
- ENDMEndofamacro

General form:

macronameMACRO[Arg1,Arg2…]

…
…
…

Programstatements in the macro

macronameENDM

Userdefinednameofthemacro

INTRODUCTIONTOTASMPROGRAMMING

- The Turbo Assembler (TASM) mainly PC-targetedassemblerpackagewasBorland's offeringintheX86assemblerprogramming tool market.
- Asonewouldexpect,TASMworkedwellwith Borland's high-level language compilers for thePC,suchasTurboPascal,TurboBasicand Turbo C.

- The effective execution of a program in assemblylanguageweneedthefollowing

- MASMassembler
- NE(Norton’sEditor)editor(or)Edlineditor
- Linker
- DebugutilityofDOS

HowtouseTASM:

- Install the specified TASM software on PC with DOS operating system. The program implementationanditsexecutionareillustrated in four stages, they are
- Editingof program
- Assemblingtheprogram
- Linkingtheprogram
- Debuggingandexecutionoftheprogram
- C:\tasm>edit(filename).asm
- >tasm (filename).asm

Assembling

- Turboassemblerversion3.2copyright(C)1988,1992 Borland International
- Assemblingfile :(filename)
- Errormessages : None
- Warningmessages : None
- Passes : 1

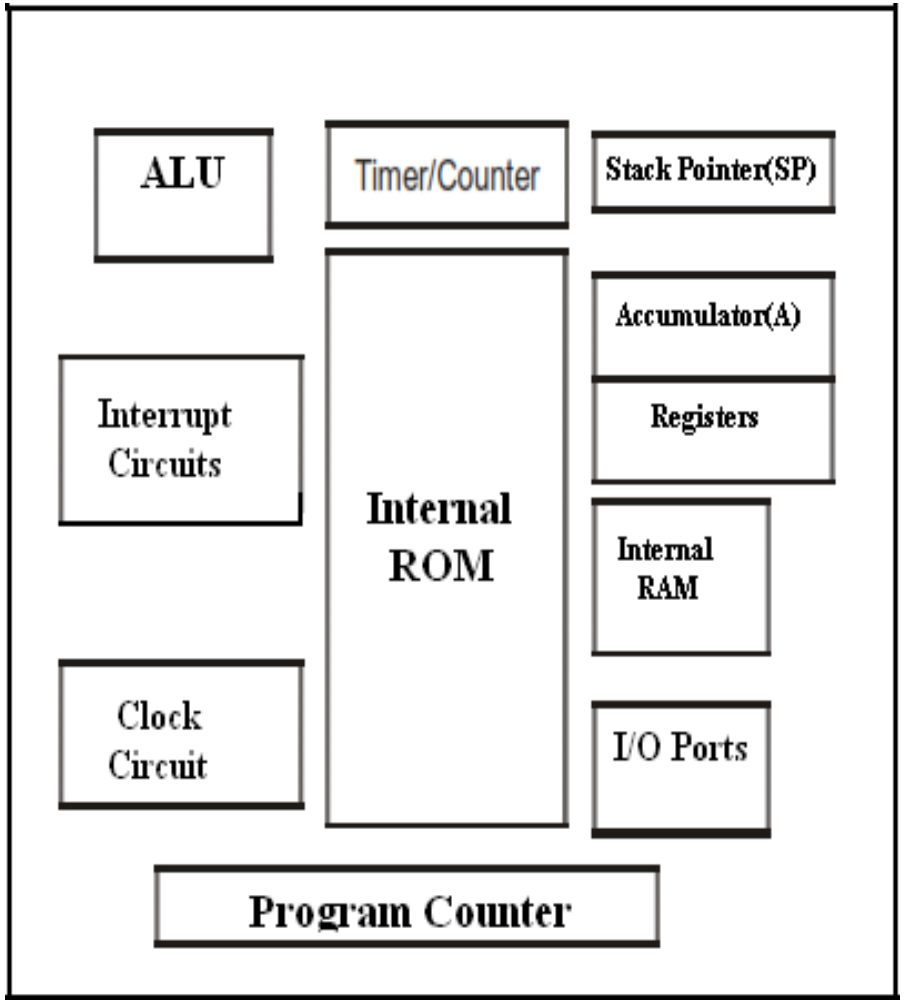
- **PROGRAM:**

- Remainingmemory : 392K

- C:\tasm>tlink(filename).obj
  TurboLinking
- Turbolinkversion5.1copyright(C)1992Borland International **the Program**
-
-
  C:\ tasm>debug (file name).exe
  Debugging&
  Operation
- -r **ofthe**
- -p **Program**
- **-**q

MICROCONTROLLER

- **WhatisaMicrocontroller?**
- A single chip computer or A CPU with all the peripherals likeRAM, ROM,I/O Ports, Timers,ADCsetc...onthesamechip.Forex: Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC 16X etc…


ALU
Timer/Counter
Stack Pointer(SP)
Accumulator(A)
Interrupt Circuits
Registers
Internal ROM
Internal RAM
Clock Circuit
I/O Ports
Program Counter

Parallel Communication Interface between two Microprocessors using 8255

The 8255 is a widely used, programmable parallel I/O device. It can be programmed to transfer data under data under various conditions, from simple I/O to interrupt I/O. The 8255 is a widely used, programmable parallel I/O device.

The 8255 has 24 I/O pins that can be grouped primarily into two8 bit parallel ports: A and B, with the remaining 8 bits as Port C. The 8 bits of port C can be used as individual bits or be grouped into two 4 bit ports: CUpper (CU) and CLower (CL). The functions ofthese ports are defined by writing a control word in the control register.

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

Mode 1 is a handshake mode whereby Port A and/or Port B use bits from Port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt. In mode 2, Port A can be set up for bidirectional data transfer using handshakesignalsfromPortC,andPortBcanbesetupeitherinmode0 or mode 1.

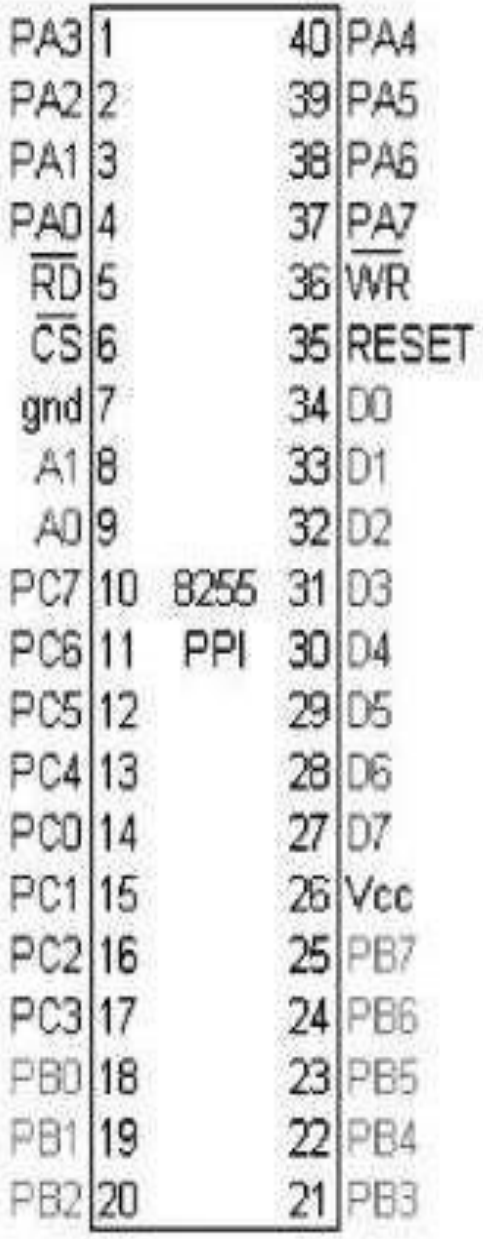


Fig. 3.8 Pins of 8255

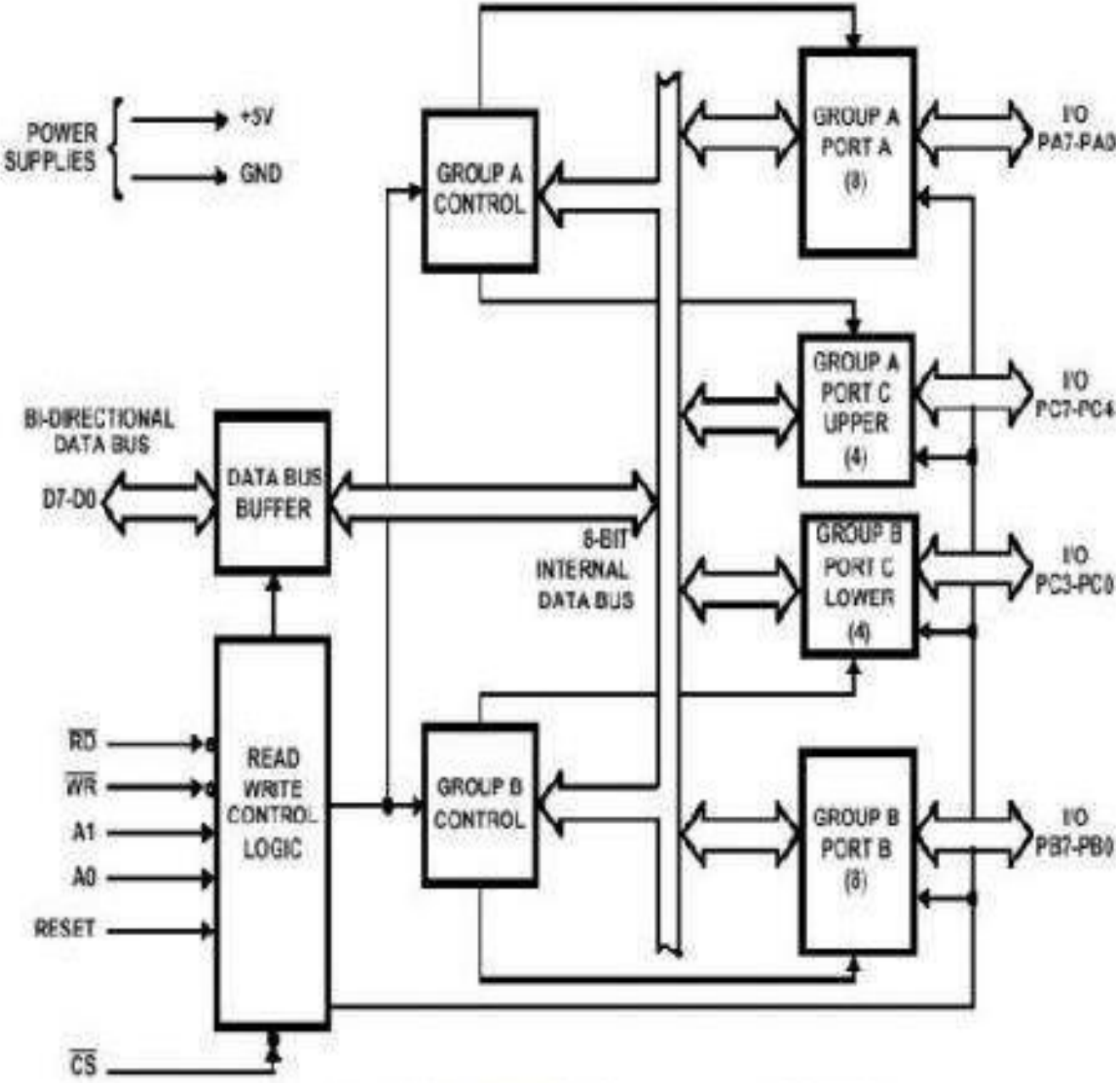


Fig.3.9 Block diagram of 8255

RD:(Read):ThissignalenablestheRead operation. When the signal is low, microprocessor reads datafrom a selected I/O port of 8255.

WR: (Write): This control signal enables the write operation.

RESET (Reset): It clears the control registers and sets all ports in input mode. CS,A0, A1: These are device select signals. is connected to a decoded address and A0, A1 are connected to A0, A1 of microprocessor.

$\overline{CS}$	A_1	A_0	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	x	x	8255 is not selected

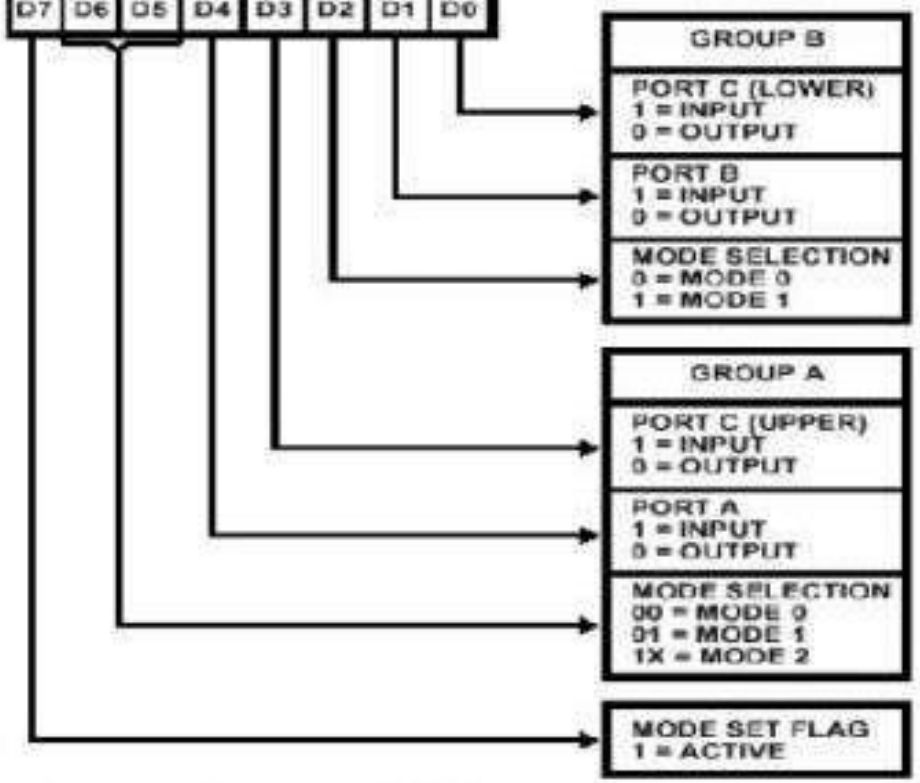


Fig 3.10 Control wordformat of8255

ü I/OModesof8255

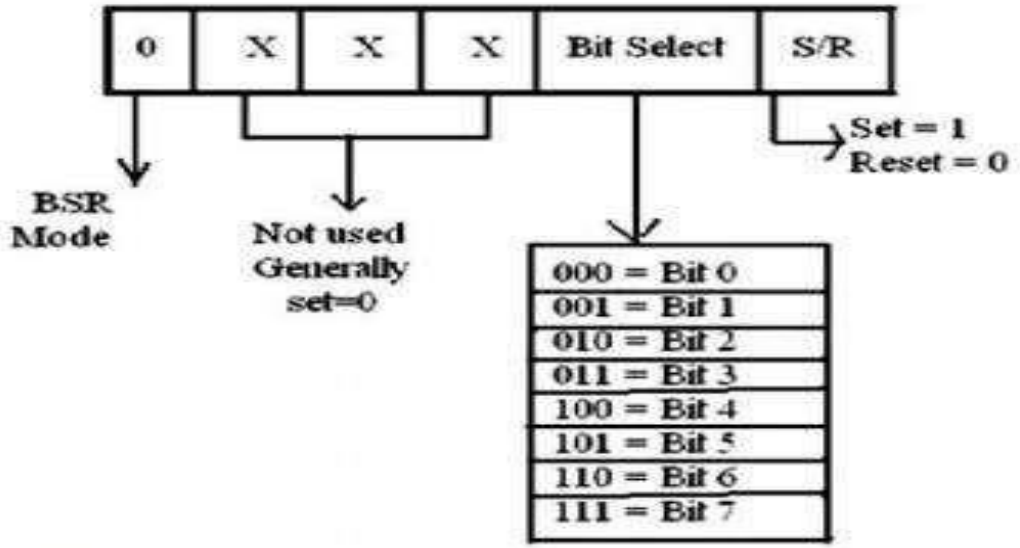


Fig. 3.11 BSR mode of 8255

Mode0: SimpleInputorOutput:

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

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

Mode1:Inputcontrolsignals

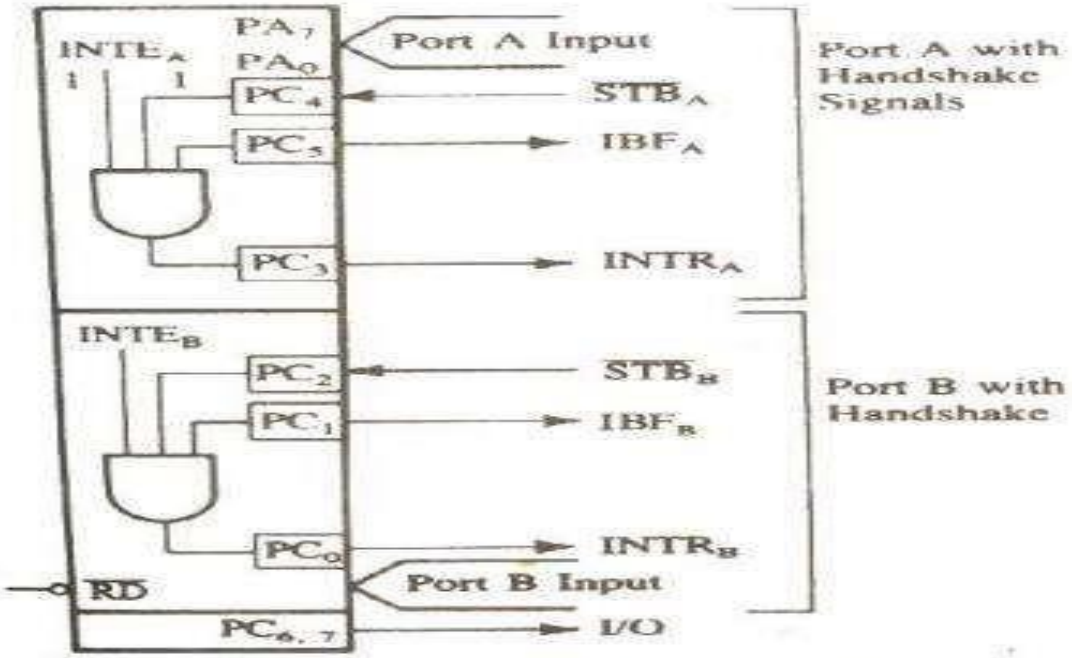


Fig.3.12 mode 1 input control signals

STBStrobeInput):Thissignal(activelow)isgeneratedbyaperipheraldevicethatit hastransmittedabyteofdata.The8255,inresponseto,generatesIBFandINTR.

IBF(Inputbufferfull): Thissignalisanacknowledgement bythe8255toindicate that the input latch has received the data byte. This is reset when the microprocessor readsthedata.INTR(InterruptRequest):Thisisanoutputsignal thatmaybeusedtointerruptthemicroprocessor.Thissignalisgenerated if,IBF and INTE are all at logic 1.

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

OutputBufferFull): Thisisanoutputsignalthatgoeslowwhenthemicroprocessor writes data into the output latch of the 8255. This signal indicates to an output peripheral that new data is ready tobe read. Itgoes high again after the 8255 receives a signal from the peripheral.

(Acknowledge): Thisisaninputsignalfromaperipheral thatmustoutputalow when the peripheral receives the data from the 8255 ports.

NTR (Interrupt Request): This is an outputsignal, and it isset by the rising edge of the signal.Thissignalcanbeusedtointerruptthemicroprocessor torequestthenextdata byte for output. The INTR is set and INTE are all one and reset bythe rising edge of

DOs&DON'TS

- Donotdisplacemonitor,keyboard,mouseetc.
- Donotusepersonalpendriveswithout permission.
- Students should not attempt torepair, open, tamper or interfere with any of the computer,cabling,orotherequipmentinthe laboratory.

SafetyPrecautions

- DatawillbepreservedusingUPSBackup.
- EquippedwithFireExtinguishers.
- StudentsandFacultyareinstructedtofollowSafetyInstructionsChartinthe Laboratories.
- BeforeinsertingUSBStick,thePendriveshavetobescannedforanymalicious content.
- TheLabis underCCCamerasurveillance.
- KeepalltheComputersUpdatedwithantivirussoftware.
- MakeSure theFirewallsareenabledoneachand everyComputer.
- MiniatureCircuitBreaker's(MCB's).
- StudentsinsertingUSBStickhavetobescannedforanymaliciouscontent.
- Studentsshouldnotattempttorepair,open,tamperorinterferewithanyof the computer,cabling,orotherequipmentinthe laboratory.
- Donotdisplacemonitor,keyboard,mouseetc.
- Donotusepersonal pendriveswithoutpermission.

	
Fire Extinguishers	First Aid Box
	
Miniature Circuit Breaker's (MCB's).	CC Camera surveillance

Table 6.4.2 Safety Measures in the Laboratories

THANK YOU