**Batch: Roll No.:**

**Experiment / assignment / tutorial No.\_\_\_\_\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:** To To Add two 16 bit numbers, to multiply two 8 bit & to divide two 8 bit numbers |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Aim:** To expose the students to the basic concepts of Microprocessor

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 1:** Explain   the process of Compilation from Assembly language to machine language

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**1) Microprocessor architecture and applications with 8085: By Ramesh Gaonkar (Penram International Publication).**

**2) 8086/8088 family: Design Programming and Interfacing: By John Uffenbeck (Pearson Education).**

**Pre Lab/ Prior Concepts:**

**Assembler directives: These are statements that direct the assembler to do something**

**Definition:**

**Types of Assembler Directives:**

**ASSUME Directive** - The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

**Example:**

**ASUME CS:CODE** ;This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.

**ASUME DS:DATA** ;This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA

**Start:**

It is entry point of the program. without this program won’t run.

**END** - END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. Carriage return is required after the END directive.

**ENDS** - This ENDS directive is used with name ofthe segment to indicate the end of that logic segment.

**Example:**

**CODE SEGMENT** ;

Hear it Start the logic

;segment containing code

; Some instructions statements to perform the logical

;operation

**CODE ENDS** ;End of segment named as;CODE

**Arithmetic instruction set:**

**ADD instruction:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flags** |
|  |  |  |  | **Affected** |
|  |  |  |  |  |
| ADD | Addition | ADD D, S | (S) + (D)🡪(D) | All |
|  |  |  |  |  |
|  |  |  | Carry🡪(CF) |  |
|  |  |  |  |  |
|  |  |  |  |  |
| ADC | Add with | ADC D, S | (S) + (D) +(CF) | All |
|  | carry |  | 🡪 (D) |  |
|  |  |  | Carry🡪(CF) |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Syntax: ADD destination,source**

**SUB instruction:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Meaning** | **Format** | **Operation** | **Flags Affected** |
|  |  |  |  |  |
| SUB | Subtract | SUB D, S | (D) - (S)🡪(D) | All |
|  |  |  |  |  |
|  |  |  | Borrow🡪(CF) |  |
|  |  |  |  |  |
|  |  |  |  |  |
| SBB | Subtract with | SBB D, S | (D) - (S) –(CF)🡪(D) | All |
|  | borrow |  |  |  |
|  |  |  |  |  |

**MUL instruction:**

**Syntax: MUL source**

|  |  |  |  |
| --- | --- | --- | --- |
| **Multiplication** | **Multiplicand** | **Operand** | **Result** |
| **(MUL or IMUL)** |  | **(Multiplier)** |  |
|  |  |  |  |
| Byte \* Byte | AL | Register or | AX |
|  |  | Memory |  |
|  |  |  |  |
| Word \* Word | AX | Register or memory | DX :AX |

**DIV instruction:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Division** | **Dividend** | **Operand** | **Quotient : Remainder** |
| **(DIV or IDIV)** |  | **(Divisor)** |  |
|  |  |  |  |
| Word / Byte | AX | Register or memory | AL : AH |
|  |  |  |  |
| Dword / Word | DX:AX | Register or memory | AX : DX |
|  |  |  |  |

**The steps to execute a program in TASM are**

**ASSEMBLING AND EXECUTING THE ROGRAM**

1. **Writing an Assembly Language Program**

Assembly level programs generally abbreviated as ALP are written in text editor EDIT.

Type *EDIT* in front of the command prompt **(C:\TASM\BIN)** to open an untitled text file.

*EDIT<file name>*

After typing the program save the file with appropriate file name with an extension *.ASM*

Ex:Add.ASM

1. **Assembling an Assembly Language Program**

To assumble an ALP we needed executable file called MASM.EXE. Only if this file is in current working directory we can assemble the program. The command is

*TASM<filename.ASM>*

If the program is free from all syntactical errors, this command will give the **OBJEC**T file.In case of errors it list out the number of errors, warnings and kind of error.

**Note: No object file is created until all errors are rectified.**

1. **Linking**

After successful assembling of the program we have to link it to get **Executable file.**

The command is

*TLINK<File name.OBJ>*

This command results in <*Filename.exe>*which can be executed in front of the command prompt.

1. **Executing the Program**

Open the program in debugger by the command(note only exe files can be open)by the command.

*<Filename.exe>*

This will open the program in debugger screen where in you can view the assemble code with the CS and IP values at the left most side and the machine code. Register content,memory content also be viewed using ***TD***option of the debugger & to execute the program in single steps(F7)

**Algorithm for adding the two numbers:**

**Algorithm for Subtracting the two 16 bits numbers**

**Algorithm for multiplying the two numbers:**

**Algorithm for dividing the two numbers:**

**Conclusion:**

**Post Lab Descriptive Questions (Add questions from examination point view)**

**Explain instructions ADC and SBB with example**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: Roll No.:**

**Experiment / assignment / tutorial No.\_\_\_\_\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title :** To exchange blocks of data using string instructions |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective**: To understand significance of string instructions

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 1:** Explain   the process of Compilation from Assembly language to machine language

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**1) Microprocessor architecture and applications with 8085: By Ramesh Gaonkar (Penram International Publication).**

**2) 8086/8088 family: Design Programming and Interfacing: By John Uffenbeck (Pearson Education).**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts: string instructions like MOVSB, CLD, REP have to be known. Assume no of blocks to be transferred and copy block from one location to another location using string instructions**

**Instructions used:**

**Eg:**

**Algorithm:**

**Conclusion:**

**Post Lab Descriptive Questions (Add questions from examination point view)**

**Explain significance of various string instructions:**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: Roll No.:**

**Experiment / assignment / tutorial No.\_\_\_\_\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: To find Fibonacci series of N given terms** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:** To understand usage of SI.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 1:** Explain the process of Compilation from Assembly language to machine language.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: By Liu & Gibson (PHI Publication).**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

**What is the significance of index registers?**

An **index register** in a computer's [CPU](https://en.wikipedia.org/wiki/Central_processing_unit) is a [processor register](https://en.wikipedia.org/wiki/Processor_register) used for modifying [operand](https://en.wikipedia.org/wiki/Operand) addresses during the run of a program, typically for doing vector/[array](https://en.wikipedia.org/wiki/Array_data_structure) operations.

The contents of an index register is added to (in some cases subtracted from) an immediate address (one that is part of the instruction itself) to form the "effective" address of the actual data (operand). Special instructions are typically provided to test the index register and, if the test fails, increments the index register by an immediate constant and branches, typically to the start of the loop. Some [instruction sets](https://en.wikipedia.org/wiki/Instruction_set) allow more than one index register to be used; in that case additional instruction fields specify which index registers to use. While normally processors that allow an instruction to specify multiple index registers add the contents together.

**Algorithm:**

**Conclusion:**

**Post Lab Descriptive Questions (Add questions from examination point view)**

**Explain various types of instruction sets in 8087**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: Roll No.:**

**Experiment / assignment / tutorial No.\_\_\_\_\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:** Program to calculate the factorial of a given number using FAR PROCEDURE or  MACRO. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:** To understand types of procedure

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 1:** Explain the process of Compilation from Assembly language to machine language.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**1.Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: By Liu & Gibson (PHI Publication).**

2.8086/8088 family: Design Programming and Interfacing: By John Uffenbeck(Pearson Education).

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

**Theory:**

**Procedure is basically group of instructions which can be used whenever we have**

**to execute several times throughout the program**

**Syntax of procedure:**

**Endp Assembler directive:**

**Macros: When the repeated group of instruction is too short and not appropriate to be written as procedure ,we use macro**

**Syntax of Macro:**

**Instruction used:**

**1. Dec Instruction:**

**Syntax:**

**Eg:**

**2. Conditional jump instruction:**

**Syntax:**

**Eg:**

**3. Call instruction:**

**Syntax:**

**Eg:**

**Return instruction:**

**Syntax:**

**Eg:**

**Algorithm for calculating the factorial:**

**Conclusion:**

**Post Lab Descriptive Questions (Add questions from examination point view)**

**Comparison between Procedure & Macro:**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: B1 Roll No.: 1711072**

**Experiment / assignment / tutorial No. 5**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: To find the value of 2 x -1 and to find the square-root of a number using 8087 instruction set.** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:** To understand usage of the instruction set of 8087

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 3:** Analyze the techniques for faster execution of instructions and enhance performance of microprocessors

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: By Liu & Gibson (PHI Publication).**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

**What are the Transcendental Instruction 8087?**

Instructions used:

1. FINIT:

This instruction used to initialize 8087. Disables interrupt.

Syntax:

FINIT

2. FLD instruction:

This instruction is used to load stack with some value. It is like PUSH

operation. It decrement stack pointer by 1 and copies number into

stack top (ST or ST(0))

Example FLD 4.6

3. FMUL instruction:

This instruction performs multiplication between ST(0) & ST(1) and store result in ST(0)

Examples:

FMUL            ST(0) := ST(0) \*SY(1)

 FMUL i          ST(0) :=ST( 0) \* i  
 FMUL i,0        ST(i) := ST(i) \* ST(0)  
 FMUL 0,i        ST(0) := ST(0) \* ST(i)  
 FMUL mem4r     ST( 0) :=ST( 0) \* mem4r  
 FMUL mem8r      ST(0) := ST(0) \* mem8r

4. FSQRT instruction:

Calculate square root of ST & store result in ST.

Example:

Fld 4.0

Fsqrt

5. F2XM1 instruction:

It 2X -1 where 0 ≤ x ≤ 0.5. X must be in ST & result will be in ST.

**Syntax:**

**F2XM1**

**Eg:**

**F2XM1**

**Algorithm:**

DATA SEGMENT

A DD 9.0

B DD 0.25

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START:

MOV AX, DATA

MOV DS, AX

FINIT

FLD A

FSQRT

FST ST(3)

FLD B

F2XM1

FST ST(2)

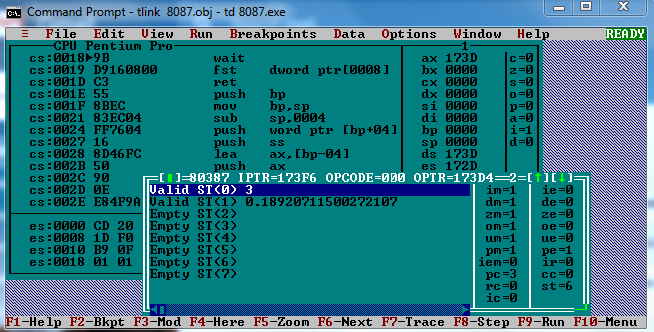
MOV AH, 4CH

INT 21H

CODE ENDS

END START

**Output Screen:**



**Conclusion:** The program was executed successfully in TASM. The program gave appropriate results.

**Post Lab Descriptive Questions (Add questions from examination point view)**

**Explain various types of instruction sets in 8087**

INSTRUCTION SET

The 8087 instruction mnemonics begins with the letter F which stands for Floating point and distinguishes from 8086. These are grouped into Four functional groups.

The 8087 detects an error conditio instruction it will set the bit in its Status register. Types:

1. DATA TRANSFER INSTRUCTIONS.
2. II. ARIT
3. COMPARE INSTRUCTIONS.
4. TRANSCENDENTAL INSTRUCT (Trigonometric and Exponential)

Data Transfers Instructions

REAL TRANS

FLD Load real

FST Store real

FSTP Store real and pop and pop

FXCH Exchange registers

INTEGER TRANSFER

FILD Load integer

FIST Store integer

FISTP Store integer

PACKED DECIMAL TRANSFER(BCD)

FBLD Load BCD

FBSTP Store BCD and pop

Example FLD Source- Decrements the stack pointer by one and copies a real element from a number of stack elements or memory location to new ST.

FLD LONG\_REAL[BX] ;Number from memo ;copied to ST.

FLD Destination- Copies ST to a specified stack position or to a specified memory location.

FST ST(2) ;Copies ST to ST(2)

**Date: 12/03/2019 Signature of faculty in-charge**

**Batch: B1 Roll No.: 1711072**

**Experiment / assignment / tutorial No. 6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Write a assembly program to find type of CPU inside the machine using CPUID instruction.** |

|  |
| --- |
|  |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Aim:** To learn instructions of Pentium.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

CO4: Identify and describe  multicore processors **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Advanced Microprocessor: By Roy & Bhurchandi (Tata McGraw Hill).**
2. **http://www.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

***Eflag Register:***

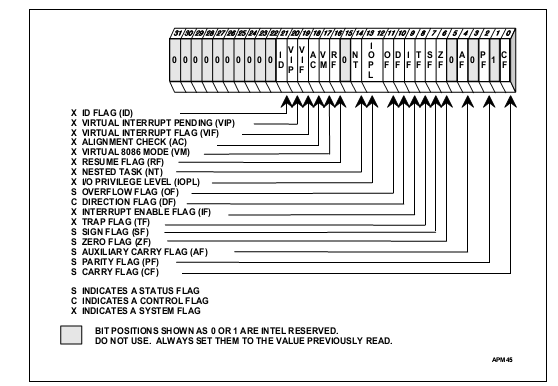
The CPUID instruction supports two sets of functions. The first set returns basic processor information. The second set returns extended processor information.

CPUID instruction provides processor identification in register EAX, EBX, ECX, EDX. This information identifies INTEL as vendor, gives the family modes and stepping of processor.

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction.

Condition codes (e.g., carry, sign, overflow) and mode bits are kept in a 32-bit register named EFLAGS. Figure 1 defines the bits within this register.

The flags control certain operations and indicate the status of the Pentium processor. Besides status and control flag bits, the flag register also contains system flags.

****

**Figure 1 EFLAG Register**

The output from the CPUID instruction is fully dependent upon the contents of the EAX register. This means, by placing different values in the EAX register and then executing CPUID, the CPUID instruction will perform a specific function dependent upon whatever value is resident in the EAX register.

1)In order to determine the highest acceptable value for the EAX register input and CPUID functions that return the basic processor information, the program should set the EAX register parameter value to “0” and then execute the CPUID instruction as follows:

MOV EAX, 00000000H

CPUID

After the execution of the CPUID instruction, a return value will be present in the EAX register.

Always use an EAX parameter value that is equal to or greater than zero and less than or equal to this highest EAX “returned” value.

So as to print numerical values, take two counters one is of 4 because each digit requires 4 bits & second is to count nos..Now move the contents of temporary to real register & rotate it to left by four places, because each digit requires four bits. Now again MOV the real register contents to temporary registers because afterwards we will lose those contents. ANDing operation is done with the contents of AL register to 0Fh, now compare the contents of al register with 0Ah if carry is generated then print that digit with adding 30h(so as to get real value not ASCII) else just add 07h to register contents with adding 30h to it. Repeat this procedure for all eight values.

A vendor identification string is retuned in EBX register for Intel processors ,the vendor identification string is ’Genuine Intel’ as shown .vendor identification string is returned in the EBX, EDX, and ECX registers. For Intel(R) processors, the vendor identification string is “GenuineIntel” as follows:

EBX ←756e6547h (\* "Genu", with G in the low nibble of BL \*)

EDX ←49656e69h (\* "ineI", with i in the low nibble of DL \*)

ECX ←6c65746eh (\* "ntel", with n in the low nibble of CL \*)

So as to print character values, take two counters one is for total no of characters that is 4 because each char requires 8 bits. Now move the contents of temporary register to real register so as to retain original values. Now first char is printed using,

MOV dl, al

MOV ah, 02h

INT 21h

Now again move the contents of temporary to real register & rotate through right the contents of EAX to CL times .Again move the contents of real register to temporary register. Now

MOV dl, al

MOV ah, 02h

INT 21h

which will display the next character ,decrement the counter & repeat the procedure till counter becomes zero.

2. When the input value is 1, the processor returns versions information in EAX register and feature information in EDX register.EBX & ECX are reserved. When the input value is 1, the processor returns version information in the EAX register (see "Version Information in the EAX Register"). The version information consists of an IA-32 processor family identifier, a model identifier, a stepping ID, and a processor type. The model, family, and processor type for the first processor in the Intel Pentium 4 processor family is as follows:

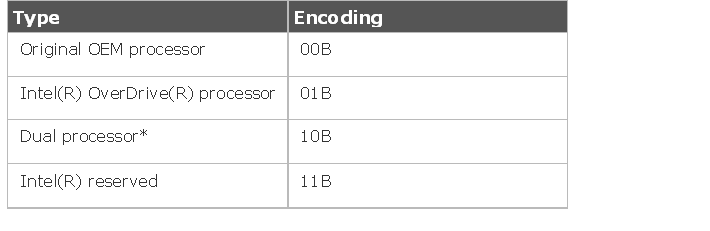
Model—0000B

Family—1111B

Processor Type—00B

The available processor types are given in the table "Processor Type Field". Intel releases information on stepping IDs as needed.

Processor Type Field



Since all the values are digits repeat the above digit display function.

3. When the input value is 2,the processor returns information about the processor, internal caches &ICB’s in the EAX, EBX, ECX & EDX

When the EAX register contains a value of 2, the CPUID instruction loads the EAX, EBX, ECX and EDX registers with descriptors that indicate the processors cache and TLB characteristics.

The lower 8 bits of the EAX register (AL) contain a value that identifies the number of times the

CPUID has to be executed to obtain a complete image of the processor’s caching systems. For example, the Pentium 4 processor returns a value of 1 in the lower 8 bits of the EAX register to indicate that the CPUID instruction need only be executed once (with EAX = 2) to obtain a complete image of the processor configuration.

The remainder of the EAX register, the EBX, ECX and EDX registers contain the cache and TLB descriptors..

When the input value is 2, the processor returns information about the processor's internal caches and TLBs in the EAX, EBX, ECX, and EDX registers. The encoding of these registers is as follows:

The least-significant byte in register EAX (register AL) indicates the number of times the CPUID instruction must be executed with an input value of 2 to get a complete description of the processor's caches and TLBs. The first member of the family of Pentium 4 processors will return a 1.

The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).

If a register contains valid information, the information is contained in 1 byte descriptors. The table "Encoding of Cache and TLB Descriptors" shows the encoding of these descriptors. Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache or TLB types. The descriptors may appear in any order.

Encoding of Cache and TLB Descriptors:

The first member of the family of Pentium 4 processors will return the following information about caches and TLBs when the CPUID instruction is executed with an input value of 2:

EAX 66 5B 50 01H

EBX 0H

ECX 0H

EDX 00 7A 70 00H

These values are interpreted as follows:The least-significant byte (byte 0) of register EAX is set to 01H, indicating that the CPUID instruction needs to be executed only once with an input value of 2 to retrieve complete information about the processor's caches and TLBs.

The most-significant bit of all four registers (EAX, EBX, ECX, and EDX) is set to 0, indicating that each register contains valid 1-byte descriptors.Bytes 1, 2, and 3 of register EAX indicate that the processor contains the following:

 50H—A 64-entry instruction TLB, for mapping 4-KByte and 2-MByte or 4-MByte pages.

5BH—A 64-entry data TLB, for mapping 4-KByte and 4-MByte pages.

66H—An 8-KByte 1st level data cache, 4-way set associative, with a 64-byte cache line size.

The descriptors in registers EBX and ECX are valid, but contain null descriptors.

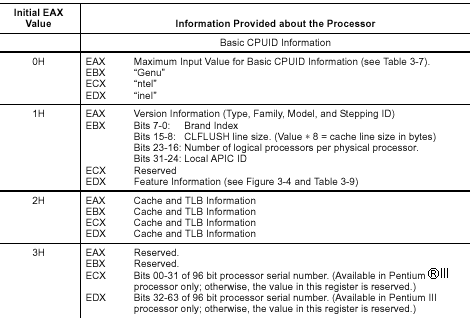
Bytes 0, 1, 2, and 3 of register EDX indicate that the processor contains the following:

00H—Null descriptor.

70H—A 12-KByte 1st level code cache, 4-way set associative, with a 64-byte cache line size.

7AH—A 256-KByte 2nd level cache, 8-way set associative, with a sectored, 64-byte cache line size.

00H—Null descripton



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Stepwise Procedure:**

**Program:**

.model small

.586p

.data

teax dd ?

tebx dd ?

tecx dd ?

tedx dd ?

str1 db '00h :$'

str2 db '01h :$'

str3 db '02h :$'

str4 db '03h :$'

.code

.startup

mov eax,00000000h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str1

mov ah,09h

int 21h

call newline

call dispd

call newline

mov eax,tebx

mov teax,eax

call dispc

mov edx,tedx

mov teax,edx

call dispc

mov ecx,tecx

mov teax,ecx

call dispc

call newline

call newline

mov eax,00000001h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str2

mov ah,09h

int 21h

call newline

call dispd

call newline

mov eax,tebx

mov teax,eax

call dispd

call newline

mov eax,tecx

mov teax,eax

call dispd

call newline

mov eax,tedx

mov teax,eax

call dispd

call newline

call newline

mov eax,00000002h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str3

mov ah,09h

int 21h

call newline

call dispd

call newline

mov eax,tebx

mov teax,eax

call dispd

call newline

mov eax,tecx

mov teax,eax

call dispd

call newline

mov eax,tedx

mov teax,eax

call dispd

call newline

call newline

mov eax,00000003h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str4

mov ah,09h

int 21h

call newline

call dispd

call newline

mov eax,tebx

mov teax,eax

call dispd

call newline

mov eax,tecx

mov teax,eax

call dispd

call newline

mov eax,tedx

mov teax,eax

call dispd

call newline

mov ah,4ch

int 21h

newline proc near

mov al,0ah

mov dl,al

mov ah,02h

int 21h

ret

endp

dispd proc near

mov ch,08h

mov cl,04h

up: mov eax,teax

rol eax,cl

mov teax,eax

and al,0fh

cmp al,0ah

jc digit

add al,06h

digit:

add al,30h

mov dl,al

mov ah,02h

int 21h

dec ch

jnz up

ret

endp

dispc proc near

mov cl,08h

mov ch,03h

mov eax,teax

mov dl,al

mov ah,02h

int 21h

up1:

mov eax,teax

ror teax,cl

mov eax,teax

mov dl,al

mov ah,02h

int 21h

dec ch

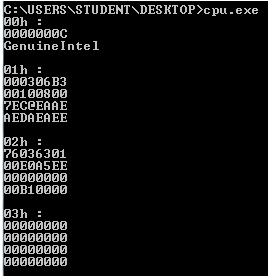
jnz up1

ret

endp

end

**Screenshot:**



**Conclusion: The CPUID program ran successfully as we were able to extract “GenuineIntel” from the CPU information.**

**Post Lab Descriptive Questions (Add questions from examination point view)**

Explain BrandID instruction of Pentium processor.

**Ans.**

.model small

.586p

.data

teax dd ?

tebx dd ?

tecx dd ?

tedx dd ?

str1 db '00h :$'

str2 db '01h :$'

str3 db '02h 03h :$'

str4 db '$'

str5 db '04 :$'

.code

.startup

mov eax,80000000h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str1

mov ah,09h

int 21h

call newline

call dispd

call newline

mov eax,tebx

mov teax,eax

call dispd

mov edx,tedx

mov teax,edx

call dispd

mov ecx,tecx

mov teax,ecx

call dispd

call newline

call newline

mov eax,80000001h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str2

mov ah,09h

int 21h

call newline

call dispd

call newline

mov eax,tebx

mov teax,eax

call dispd

call newline

mov eax,tecx

mov teax,eax

call dispd

call newline

mov eax,tedx

mov teax,eax

call dispd

call newline

call newline

mov eax,80000002h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str3

mov ah,09h

int 21h

call newline

call dispc

mov eax,tebx

mov teax,eax

call dispc

mov eax,tecx

mov teax,eax

call dispc

mov eax,tedx

mov teax,eax

call dispc

mov eax,80000003h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str4

mov ah,09h

int 21h

call dispc

mov eax,tebx

mov teax,eax

call dispc

mov eax,tecx

mov teax,eax

call dispc

mov eax,tedx

mov teax,eax

call dispc

call newline

call newline

mov eax,80000004h

cpuid

mov teax,eax

mov tebx,ebx

mov tecx,ecx

mov tedx,edx

lea dx,str5

mov ah,09h

int 21h

call dispc

mov eax,tebx

mov teax,eax

call dispc

mov eax,tecx

mov teax,eax

call dispc

mov eax,tedx

mov teax,eax

call dispc

mov ah,4ch

int 21h

newline proc near

mov al,13

mov dl,al

mov ah,02h

int 21h

mov al,10

mov dl,al

mov ah,02h

int 21h

ret

endp

dispd proc near

mov ch,08h

mov cl,04h

up: mov eax,teax

rol eax,cl

mov teax,eax

and al,0fh

cmp al,0ah

jc digit

add al,06h

digit:

add al,30h

mov dl,al

mov ah,02h

int 21h

dec ch

jnz up

ret

endp

dispc proc near

mov cl,08h

mov ch,03h

mov eax,teax

mov dl,al

mov ah,02h

int 21h

up1:

mov eax,teax

ror teax,cl

mov eax,teax

mov dl,al

mov ah,02h

int 21h

dec ch

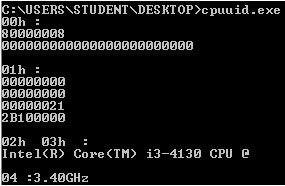
jnz up1

ret

endp

end

**Screenshot:**



**Date: 25/03/2019 Signature of faculty in-charge**

Batch: B1 Roll No.:1711072

**Experiment / assignment / tutorial No. 7**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:** Interfacing 8255 PPI with 8086 to perform different modes of 8255 i.e. basic mode  and BSR mode by using trainer kit |

**Aim:** To interface peripherals of 8086

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 2:** Build Microprocessor based system using memory chips and peripheral chips

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**1) 8086/8088 family: Design Programming and Interfacing: By John Uffenbeck (Pearson Education).**

**2) 8086 Microprocessor Programming and Interfacing the PC: By Kenneth Ayala**

**3) Microprocessor and Interfacing: By Douglas Hall (TMH Publication).**

**4) www.wikipedia.org/wiki/Intel\_8255‎**

**Pre Lab/ Prior Concepts:**

**What is PIO 8255?**

8255 is Programmable Peripheral Interface (**PPI**) chip is a peripheral chip.

The 8255 is widely used not only in many microcomputer/microcontroller systems, but also in the system board of the best known original IBM-PC. and clones, along with numerous homebuilt computers .

**The 8255 has 24 input/output pins in all These are divided into three 8-bit ports. Port A and port B can be used as 8-bit input/output ports. Port C can be used as an 8-bit input/output port or as two 4-bit input/output ports or to produce handshake signals for ports A and B.**

**The three ports are further grouped as follows:**

1. **Group A consisting of port A and upper part of port C.**
2. **Group B consisting of port B and lower part of port C.**

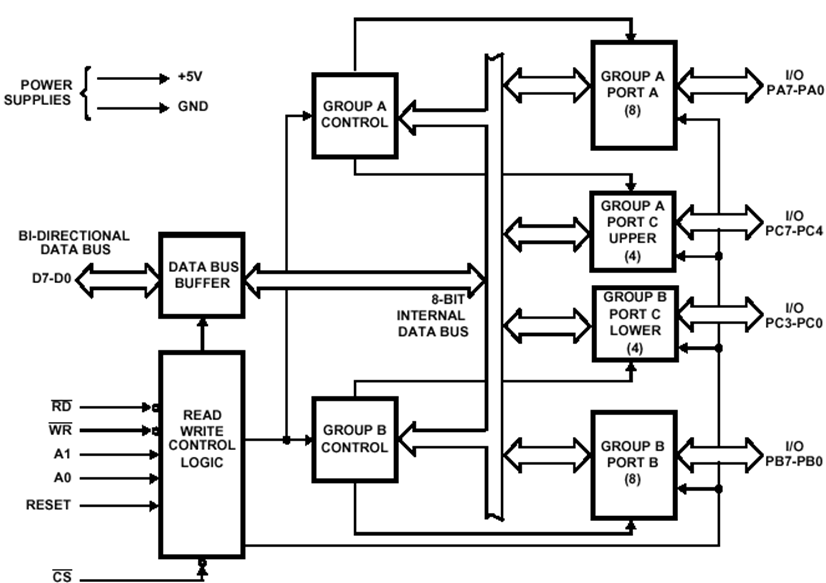
**Eight data lines (D0 - D7) are available (with an 8-bit data buffer) to read/write data into the ports or control register under the status of the {\neg}RD (pin 5) and {\neg}WR (pin 36), which are active low signals for read and write operations respectively. The address lines A1 and A0 allow to successively access any one of the ports or the control register as listed below:**

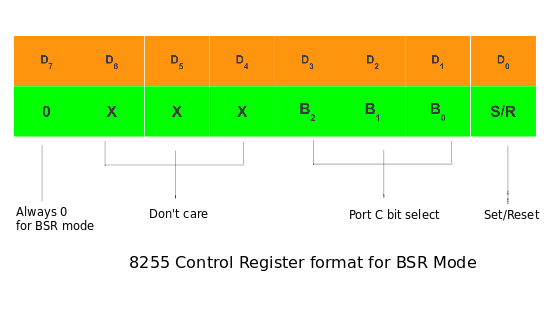
|  |  |  |
| --- | --- | --- |
| **A1** | **A0** | **Port selected** |
| **0** | **0** | **port A** |
| **0** | **1** | **port B** |
| **1** | **0** | **port C** |
| **1** | **1** | **control register** |

**The control signal {\neg}CS (pin 6) is used to enable the 8255 chip. It is an active low signal, i.e., when {\neg}CS = '0', the 8255 is enabled. The RESET input (pin 35) is connected to the RESET line of system like 8085, 8086, etc., so that when the system is reset, all the ports are initialized as input lines. This is done to prevent 8255 and/or any peripheral connected to it, from being destroyed due to mismatch of ports. As an example, consider an input device connected to 8255 at port A. If from the previous operation, port A is initialized as an output port and if 8255 is not reset before using the current configuration, then there is a possibility of damage of either the input device connected or 8255 or both since both 8255 and the device connected will be sending out data.**

**The control register or the control logic or the command word register is an 8-bit register used to select the modes of operation and input/output designation of the ports.**

**Block Diagram of 8255:**



**Modes of Operation of 8255:** BSR Mode: In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.

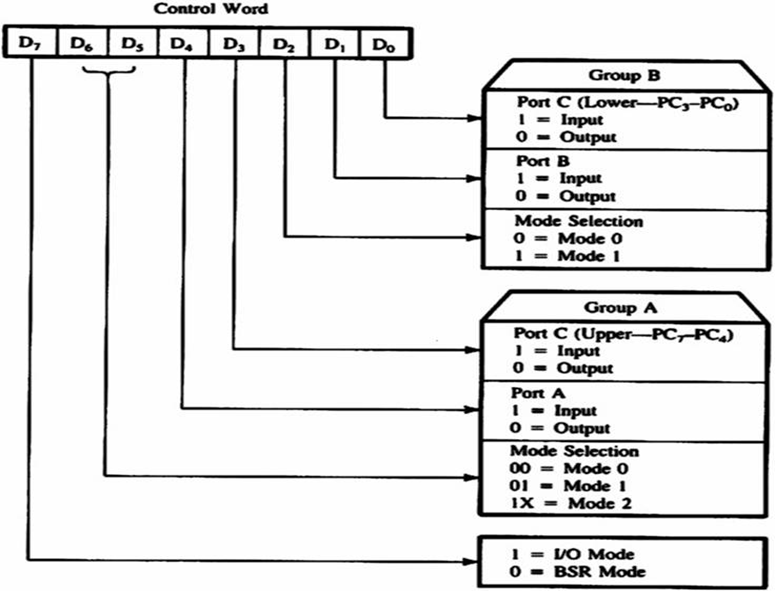
1.I/O Mode: In I/O mode, the 8255 ports work as programmable I/O ports

Under the IO mode of operation, further there are three modes of operation of 8255, So as to support different types of applications, viz. mode 0, mode 1 and mode 2.

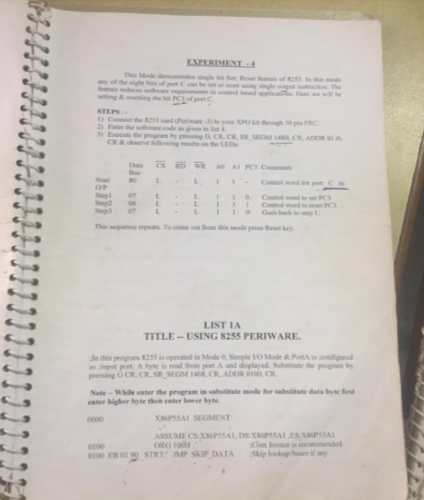
1) Mode 0 - Basic I/O mode

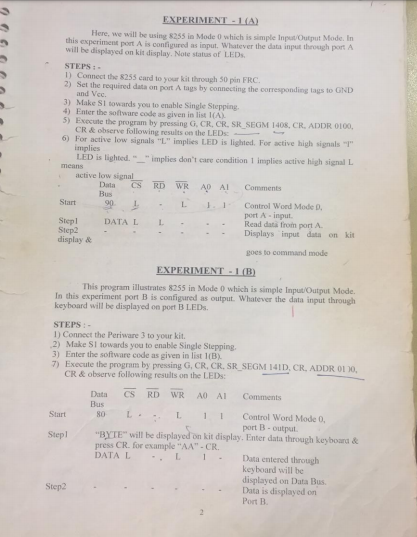
2) Mode 1 - Strobed I/O mode

3) Mode 2 - Strobed bi-directional I/O



**Steps to interface 8255 with 8086:**





**Conclusion: Thus, we learnt about interfacing 8255 along with 8086 and about it’s various modes of operation.**

**Post Lab Descriptive Questions (Add questions from examination point view)**

**Explain significance of 8255 as PIO**

**Ans.** The 8255 gives a CPU or digital system access to programmable parallel I/O.The 8255 has 24 input/output pins. These are divided into three 8-bit ports (A, B, C). Port A and port B can be used as 8-bit input/output ports. Port C can be used as an 8-bit input/output port or as two 4-bit input/output ports or to produce handshake signals for ports A and B.

The three ports are further grouped as follows:

Group A consisting of port A and upper part of port C.

Group B consisting of port B and lower part of port C.

Eight data lines (D0–D7) are available (with an 8-bit data buffer) to read/write data into the ports or control register under the status of the RD (pin 5) and WR (pin 36), which are active-low signals for read and write operations respectively. Address lines A1 and A0 allow to access a data register for each port or a control register, as listed below:

A1     A0 Port selected

0     0 port A

0     1 port B

1     0 port C

1     1 control register

The control signal chip select CS (pin 6) is used to enable the 8255 chip. It is an active-low signal, i.e., when CS = 0, the 8255 is enabled. The RESET input (pin 35) is connected to the RESET line of system like 8085, 8086, etc., so that when the system is reset, all the ports are initialized as input lines. This is done to prevent 8255 and/or any peripheral connected to it from being destroyed due to mismatch of ports. As an example, consider an input device connected to 8255 at port A. If from the previous operation, port A is initialized as an output port and if 8255 is not reset before using the current configuration, then there is a possibility of damage of either the input device connected or 8255 or both, since both 8255 and the device connected will be sending out data.

The control register (or the control logic, or the command word register) is an 8-bit register used to select the modes of operation and input/output designation of the ports.

**Date: 01/04/2019 Signature of faculty in-charge**

**Batch: B1 Roll No.: 1711072**

**Experiment / assignment / tutorial No. 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:** Interfacing 8259 PPI with 8086 to perform the ICW and OCW command words of  8259 by using trainer kit |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim:** To handle interrupts using 8259

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 2:** Build Microprocessor based system using memory chips and peripheral chips

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**1) 8086/8088 family: Design Programming and Interfacing: By John Uffenbeck (Pearson Education).**

**2) 8086 Microprocessor Programming and Interfacing the PC: By Kenneth Ayala**

**3) Microprocessor and Interfacing: By Douglas Hall (TMH Publication).**

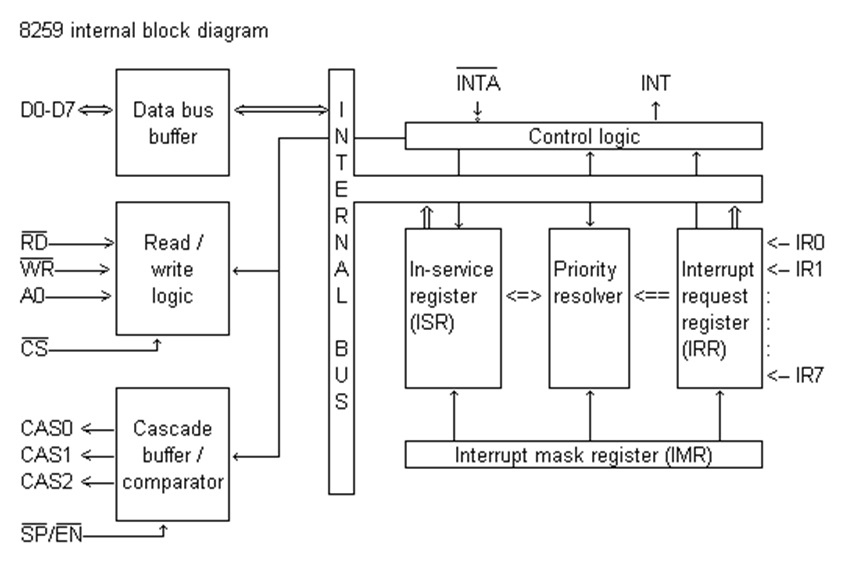
**4) www.wikipedia.org/wiki/Intel\_8259‎**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

**Need for priority interrupt controller 8259:**

**The 8259 is a Programmable Interrupt controller (PIC) designed for the Intel 8085 and Intel 8086 .The initial part was 8259, a later A suffix version was upward compatible and usable with the 8086 or 8088 processor. The 8259 combines multiple interrupt input sources into a single interupt output to the host microprocessor, extending the interrupt levels available in a system beyond the one or two levels found on the processor chip. The 8259A was the interrupt controller for the ISA bus in the original IBM PC and IBM AT.**



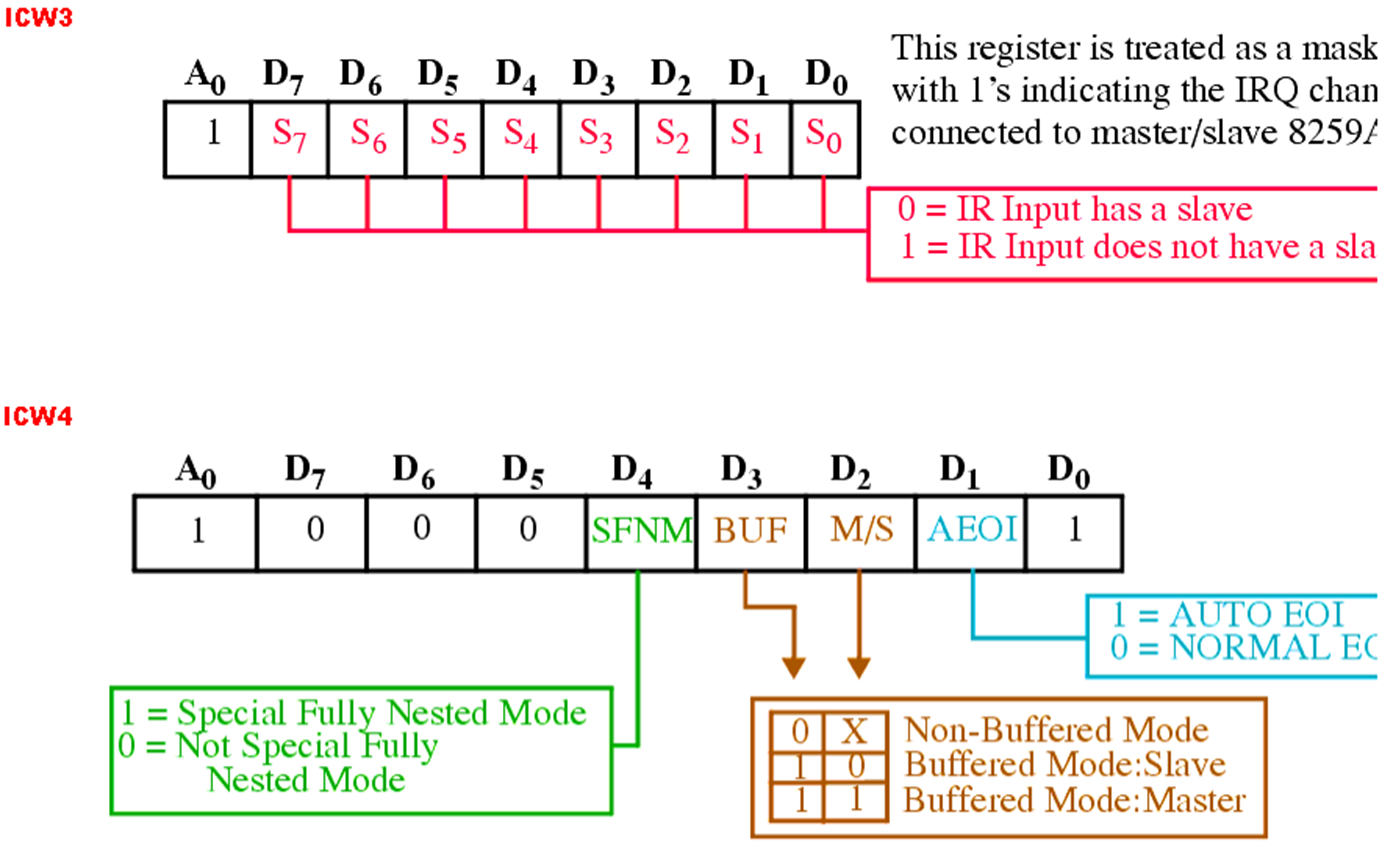
**Importance of Cascade Lines-CAS0-CAS2:**

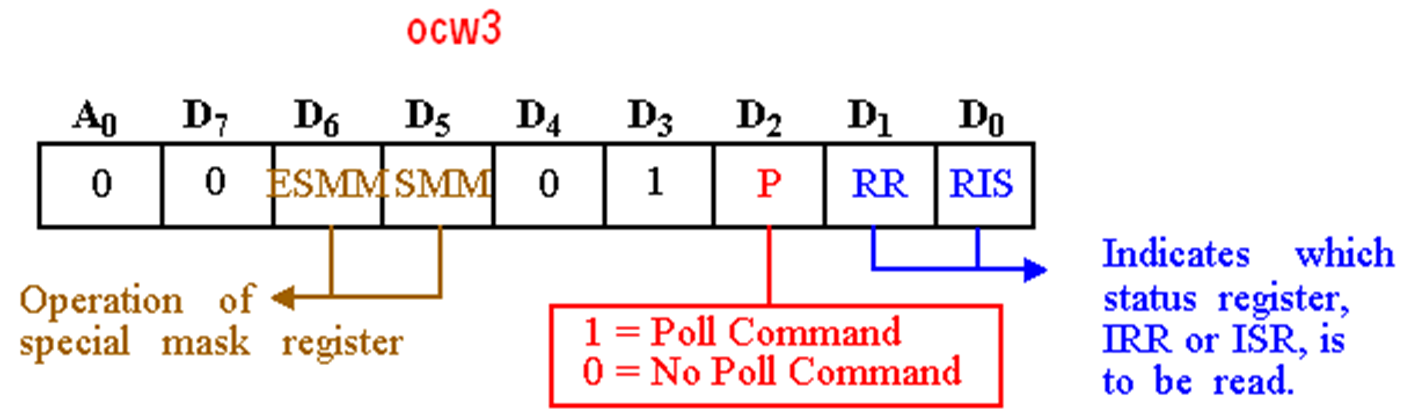
CASCADE LINES: The CAS lines form a private 8259A bus to control

a multiple 8259A structure. These pins are outputs for a master 8259A

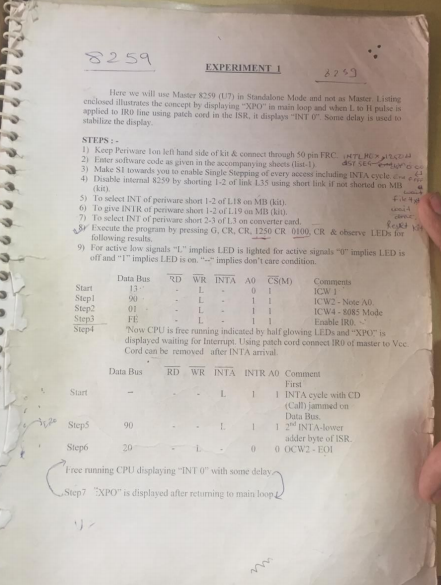
and inputs for a slave 8259A

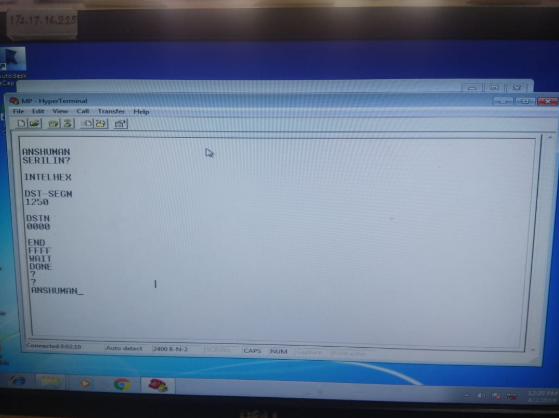
**Command Words of 8259A:**

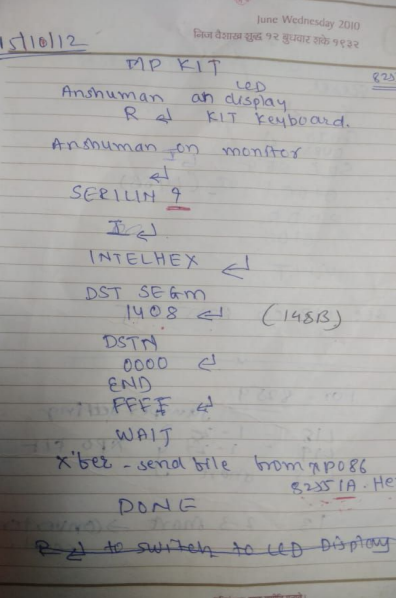
1. **(ICW)**
2. **Operation command words (OCW)**

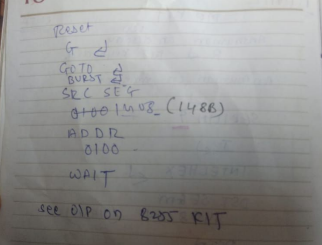


**Steps to interface 8255 with 8086:**









**Conclusion: Thus, we learnt about interfacing 8259 along with 8086 and about it’s various modes of operation.**

**Post Lab Descriptive Questions (Add questions from examination point view)**

**Explain significance of 8259 as PIC**

8259 microprocessor is defined as Programmable Interrupt Controller (PIC) microprocessor. There are 5 hardware interrupts and 2 hardware interrupts in 8085 and 8086 respectively. But by connecting 8259 with CPU, we can increase the interrupt handling capability. 8259 combines the multi interrupt input sources into a single interrupt output. Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.

Features of 8259 PIC microprocessor –

1. Intel 8259 is designed for Intel 8085 and Intel 8086 microprocessor.
2. It can be programmed either in level triggered or in edge triggered interrupt level.
3. We can masked individual bits of interrupt request register.
4. We can increase interrupt handling capability upto 64 interrupt level by cascading further 8259 PIC.
5. Clock cycle is not required.

**Date: 08/04/2019 Signature of faculty in-charge**