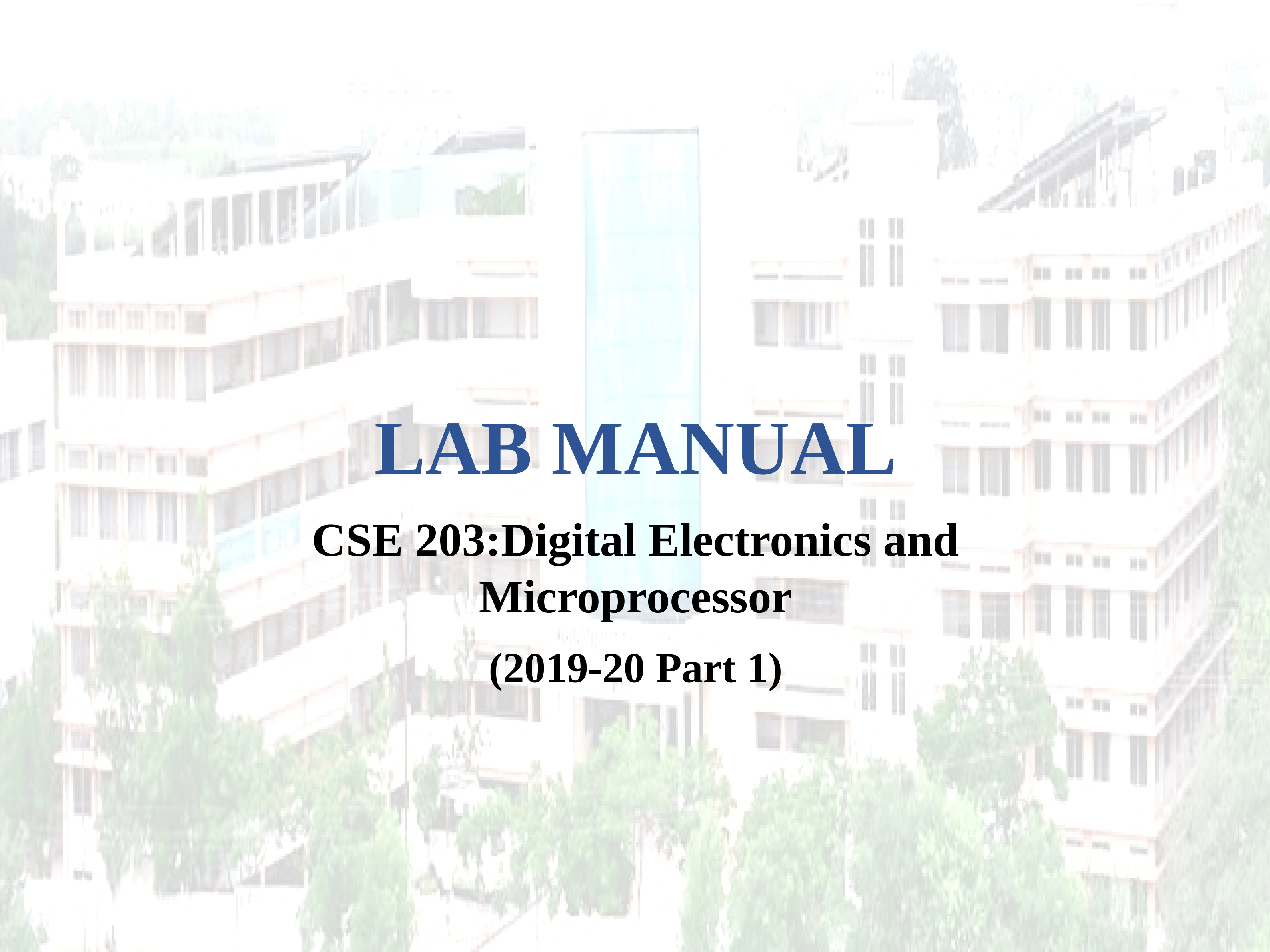
G.S. Mandal's

Maharashtra Institute of Technology, Aurangabad

Department of Computer Science and Engineering



LAB MANUAL

CSE 203:Digital Electronics and Microprocessor

(2019-20 Part 1)

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia

Department of Computer Science and Engineering

Vision

To develop the department as a center of excellence in the field of computer science and engineering by imparting knowledge & training to the students for meeting growing needs of the industry & society.

Mission

Providing quality education through a well-designed curriculum in tune with the challenging needs of software industry by providing state of the art facilities and to impart knowledge in the thrust areas of computer science and engineering.

Department of Computer Science and Engineering

Program Educational Objectives

PEO1: To prepare the students to achieve success in Computing Domain to create individual careers, innovations or to work as a key contributor to the private or Government sector and society.

PEO2: To develop the ability among the students to understand Computing and mathematical fundamentals and apply the principles of Computer Science for analyzing, designing and testing software for solving problems.

PEO3: To empower the students with ability to quickly reflect the changes in the new technologies in the area of computer software, hardware, networking and database management.

PEO4: To promote the students with awareness for lifelong learning, introduce them to professional practice, ethics and code of professionalism to remain continuous in their profession and leaders in technological society.

Program Specific Objectives

PSO1: Identify appropriate data structures and algorithms for a given contextual problem and develop programs to design and implement applications.

PSO3: Design and manage the large databases and develop their own databases to solve real world problems and to design, build, manage networks and apply wireless techniques in mobile based applications.

PSO3: Design a variety of computer-based components and systems using computer hardware, system software, systems integration process and use standard testing tools for assuring the software quality.

Department of Computer Science and Engineering

Program Outcomes

PO1: Apply knowledge of mathematics, science, and engineering fundamentals to solve problems in Computer science and Engineering.

PO2: Identify, formulate and analyze complex problems.

PO3: Design system components or processes to meet the desired needs within realistic constraints for the public health and safety, cultural, societal and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data for valid conclusions.

PO5: Select and apply modern engineering tools to solve the complex engineering problem.

PO6: Apply knowledge to assess contemporary issues.

PO7: Understand the impact of engineering solutions in a global, economic, environmental, and societal context.

PO8: Apply ethical principles and commit to professional ethics and responsibilities.

PO9: Work effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10: Communicate effectively in both verbal and written form.

PO11: Demonstrate knowledge and apply engineering and management principles to manage projects and in multi-disciplinary environment.

PO12: To engage in life-long learning to adopt to the technological changes.

Department of Computer Science and Engineering

Course: CSE 203: Digital Electronics and Microprocessor

Course Outcomes:

After completing the course students will be able to

CO1: Use the basics of Digital electronics.

CO2: Demonstrate the knowledge of combination and sequential logic circuits.

CO3: Explain the basics and architecture of 8086 microprocessor along with memory organization.

CO4: Select addressing mode and instruction set to write and execute assembly language program.

CO5: Compare and describe interfacing of basic and special purpose peripherals with 8086.

CO6: Develop and implement simple program for 8051 microcontroller.

Mapping

Experiment No.	Blooms Level	Mapping To CO	Mapping To PO
1	Apply	CO1	3
2	Apply	CO2	3
3	Apply	CO2	3
4	Apply	CO3	3
5	Apply	CO4	5
6	Apply	CO4	5
7	Apply	CO4	5
8	Apply	CO4	5
9	Apply	CO4	5
10	Apply	CO6	5

Index

	MAHARASHTRA INSTITUTE OF TECHNOLOGY AURANGABAD	LABORATORY MANUAL
QUEST FOR EXCELLENCE	PRACTICAL EXPERIMENT INSTRUCTION SHEET	
	DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING	

LABORATORY :
A.Y.: 2019-20

Class: SY CSE	PART: I	SUBJECT: DEMP

MASTER LIST OF EXPERIMENT

EXPERIMENT NO.	EXPERIMENT DESCRIPTION	Co Mapping
1	Implementation of Boolean expression using AND/OR/NOT & NAND/NOR logic.	CO1
2	Realization of Half & Full Adder using logic gates	CO2
3	Realization of Half & Full Substractor using logic gates .	CO2
4	To study TASM/MASM/emu8086.	CO3
5	Write an Assembly language program to print the string in 8086	CO4
6	Write an Assembly language program for 8-bit addition & 16-bit addition in 8086	CO4
7	Write an Assembly Language Program for 8-bit subtraction & 16-bit subtraction in 8086	CO4
8	Write an Assembly Language Program for 8-bit multiplication & 16-bit multiplication in 8086	CO4
9	Write an Assembly Language Program for finding smallest number from an array	CO4
10	Write a program to create LED pattern generation/blink LED in 8051 microcontroller.	CO6

PREPARED BY : Mrs. S. J. Nandedkar & Mr. K.R. Khandarkar	APPROVED BY : HCSED

EXPERIMENT NO.1

AIM: Verification of Boolean expression using logic gates truth tables.

APPARATUS:

Components	Quantity
1. Omega LTB 866 trainer kit	1
2. 2mm patch cords.	6

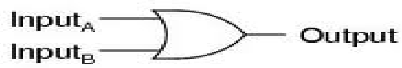
THEORY:-

1. **IC 7408 2-INPUT AND GATE**: It has an n input and output y. Mathematically AND operation for two **input** A, B is given as Y=A.B. The output of AND GATE is true only when both inputs are logic 1(true) state and in other case it is logic 0(false).

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

2. **IC 7432 2-input OR GATE**: It has n input and output Y. Mathematically OR operation for two **input** A, B is given as Y=A+B. The output of OR GATE is false only when both inputs are logic 0 state and in other case it is logic 1.

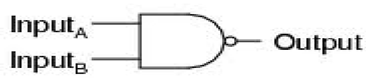
2-input OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

3. **IC 7400 input NAND GATE** : It has n input and single output Y. Mathematically NAND operation for two input A,B is given as Y=A.B. The output of NAND GATE is false only when both inputs are logic 1 and in other case it is logic 0.

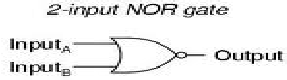
NAND gate



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

4. **IC 7402 input NOR GATE** : It has n input and Single output Y. Mathematically NOR operation for two input A,B is given as Y=A+B.The output of NOR GATE is true only when both input are logic 0 state and in other case it is logic 0.

2-input NOR gate

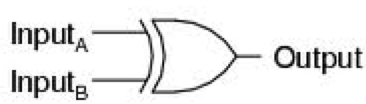


A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Equivalent gate circuit

5**. IC 7486 input EX-OR GATE**: It has n input and Single output Y. Mathematically EX-OR operation for two **input** A, B is given as Y=A+B. The output of EX-OR GATE is logic only when both input is either logic 1or logic 0 state.

Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

CONCLUSION: In this way I have verified Boolean expression using logic gates truth tables.

Rubrics for Practical Assessment:

Cognitive (3)	**Affective (3)**	**Psychomotor (3)**	**Total (9)**

Sign of Course Teacher with Date

EXPERIMENT NO.2

AIM: Realization of Half adder and Full adder using logic gates.

APPRATUS:

A. Components required for half adder.

Component	Quantity
1. Trainer kit	1
2. IC7408 2-input AND GATE	1
3. IC7486 2-input EX-OR GATE	1
4. 2mm patch cords.	

B. Component required for full adder.

Component	Quantity
1. Omega type LTB 860	1
2. IC7408 2-input AND GATE	1
3. IC7486 2-input EX-OR GATE	1
4. IC7432 2-input OR GATE	1
5. 2mm patch cords.	

THEORY :

A. HALF ADDER: Half adder is a combinational unit with 2-input and 2-output.it is a basic building block for addition of two ‘single’ bit numbers. It has two inputs A and B and output ‘sum’, ‘Carry’ namely. This circuit is not suitable for multi bit building addition where carry is propagated to next bit.

B.FULL ADDER: Full adder 3 single bit adder circuit. It can add two one-bit numbers A, B and carry Cin. The full adder in 3 inputs and 2 output combinational circuit. It can be used for multi-bit addition.

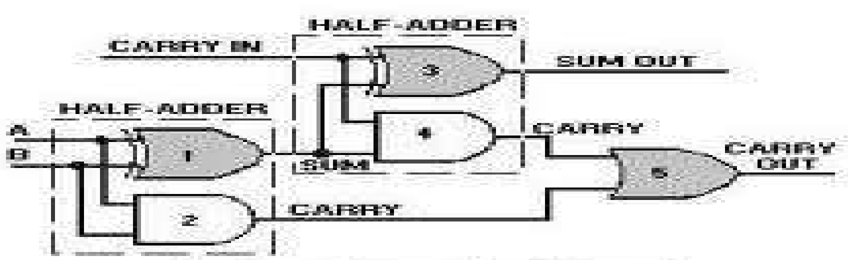
A full adder can be implemented in many different ways such as with a custom transistor level circuit or composed of other gates. One example implementation is with

S=(A+B)+Cin and Cout=(A.B)+(Cin(A B)).

Input : $\{A,B,Cin\} \longrightarrow$ Output : $\{S,Out\}$

In this implementation the final OR GATE before the carry out output may be replaced by an XOR GATE without output may be replaced by an XOR GATE without altering the resulting logic.

A full adder can be constructed from two half by connecting A and B to the input of one half adders. Connecting the sum from to an input to the second adder. Connecting Cin to the other input and or the two carry outputs equivalently. S could be made the three bit XOR of A,B,Ci and Co could be made the three bit majority function A, B and Ci.



A	B	CARRY IN	SUM OUT	CARRY OUT
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

PROCEDURE:

A. HALF ADDER:

1. Make connection as shown in fig.
2. Apply +5 volt to pin 14 and GND to pin 7.
3. Apply the data to input A, B.
4. Switch on the instrument.
5. Observe the output on S, C using LED display.
6. Repeat the steps for other combinations of inputs and verify the truth table.

B. FULL ADDER:

1. Make connection as shown in fig.
2. Apply +5 volt to pin 14 and GND to pin 7.
3. Apply the data to input A, B and Cin.
4. Switch on the instrument.
5. Observe the output on S, C using LED display.
6. Repeat the steps for other combinations of inputs and verify the truth table.

CONCLUSION: In this way I have verified truth table for half and full adder.

Rubrics for Practical Assessment:

Cognitive (3)	Affective (3)	Psychomotor (3)	Total (9)

Sign of Course Teacher with Date

EXPERIMENT NO.3

AIM: Realization of Half subtractor and Full subtractor using gates.

APPRATUS:

A. Component required for half subtractor.

Component	Quantity
1. ST2611 Digital Lab	1
2. IC7408 2-input AND GATE	1
3. IC7486 2-input EX-OR GATE	1
4. IC 7404 Hex inverter	1
5. 2mm patch cords.	

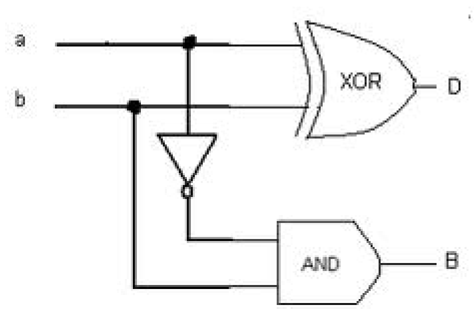
B. Component required for full subtractor

Component	Quantity
1. ST2611 Digital Lab	1
2. IC7408 2-input AND GATE	1
3. IC7486 2-input EX-OR GATE	1
4. IC7432 2-input OR GATE	1
5. IC 7404 Hex inverter	1
6. 2mm patch cords	

THEORY :

A.HALF SUBTRACTOR:

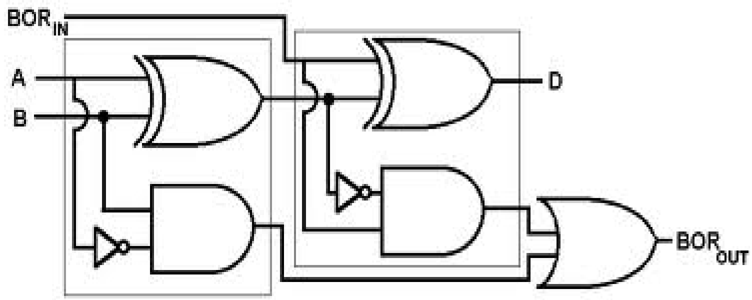
Half subtrctor is a combinational unit with 2-input X, Y and 2-output ‘difference (0)’,’borrow (B)’namely. It can perform the subtractions of two binary bits, but while performing the subtraction it does not take into account the borrow of loIr significant stage.



A	B	C	BORROW	DIFFERENCE
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

B. FULL SUBTRACTOR:

The full subtrctor is combinational with three input X,Y and borrow Bin. X is minuend Y IS SUBSTRCTED AND Bin borrow from the previous stage D is difference output and B is borrow output. It can be used in multi bit subtrctor.



Inputs			Outputs	
a	b	Bin	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

PROCEDURE:

A.HALF SUBTRCTOR:

1. Make connection as shown in fig.
2. Apply +5 volt to pin 14 and GND to pin 7.
3. Apply the data to input X, Y as shown in truth table.
4. Switch on the instrument.
5. Observe the output on D, B using LED display.
6. Repeat the steps for other combinations of inputs and verify the truth table.

B.FULL SUBTRCTOR:

1. Make connection as shown in fig.
2. Apply +5 volt to pin 14 and GND to pin 7.
3. Apply the data to input X, Y and Bor in.
4. Switch on the instrument.
5. Observe the output on D, B using LED display.
6. Repeat the steps for other combinations of inputs and verify the truth table.

CONCLUSION: In this way I have verified truth table for half and full subtractor.

Rubrics for Practical Assessment:

Cognitive (3)	**Affective (3)**	**Psychomotor (3)**	**Total (9)**

Sign of Course Teacher with Date

EXPERIMENT NO.4

Aim: Study of Architecture and register organization of 8086.

Theory:

Intel 8086 microprocessor is a first member of x86 families of processors. Advertised as a "source-code compatible" with Intel 8080 and Intel 8085 processors, the 8086 was not object code compatible with them. The 8086 has complete 16-bit architecture - 16-bit internal registers, 16-bit data bus, and 20-bit address bus (1 MB of physical memory). Because the processor has 16-bit index registers and memory pointers, it can effectively address only 64 KB of memory. To address memory beyond 64 KB the CPU uses segment registers - these registers specify memory locations for code, stack, data and extra data 64 KB segments. The segments can be positioned anywhere in memory, and, if necessary, user programs can change their position. This addressing method has one big advantage - it is very easy to write memory-independent code when the size of code, stack and data is smaller than 64 KB each.

The complexity of the code and programming increases, sometimes significantly, when the size of stack, data and/code is larger than 64 KB. To support different variations of this awkward memory addressing scheme many 8086 compilers included 6 different memory models: tiny, small, compact, medium, large and huge. 64 KB direct addressing limitation was eliminated with the introduction of the 32-bit protected mode in Intel 80386 processor.

The 8086 CPU is divided into two independent functional units:

1. Bus Interface Unit (BIU)

2. Execution Unit (EU)

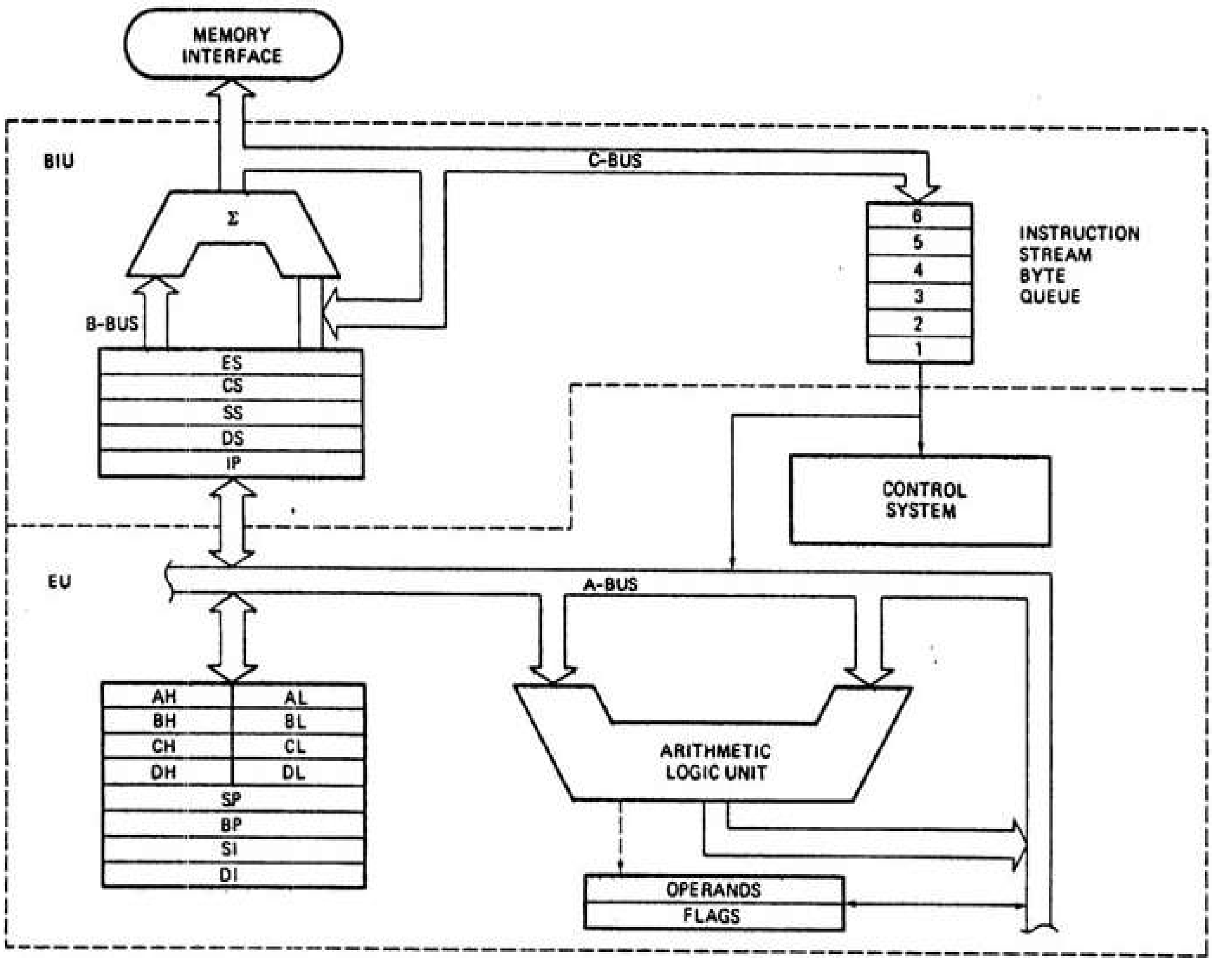


Fig. 1: Block Diagram of Intel 8086

Execution unit (EU)

The EU is where the actual processing of data takes place inside the 8086 MPU. It is here that the arithmetic and logic unit (ALU) is located, along with the registers used to manipulate data and store immediate results. The EU accepts instructions and data that have been fetched by the BIU and then processes the information. Data processed by the EU can be transmitted to the memory or peripheral devices through the BIU. EU has no direct connection with the outside world and relies solely on the BIU to feed it with instructions and data.

Bus Interface Unit (BIU)

The BIU is made up of the address generation and bus-control unit, the instruction queue, and the instruction pointer. It has the task of making sure that the bus is used to its fullest capacity in order to speed-up operations. This function is carried in two ways. First, by fetching the instructions before they are needed by the execution unit and storing them in the instruction queue, the 8086 MPU is able to increase computing speed. Second, by taking care of all bus-control functions, the EU is free to concentrate on processing data and carrying out the instructions. The instruction pointer contains the location or address of the next instruction to be executed.

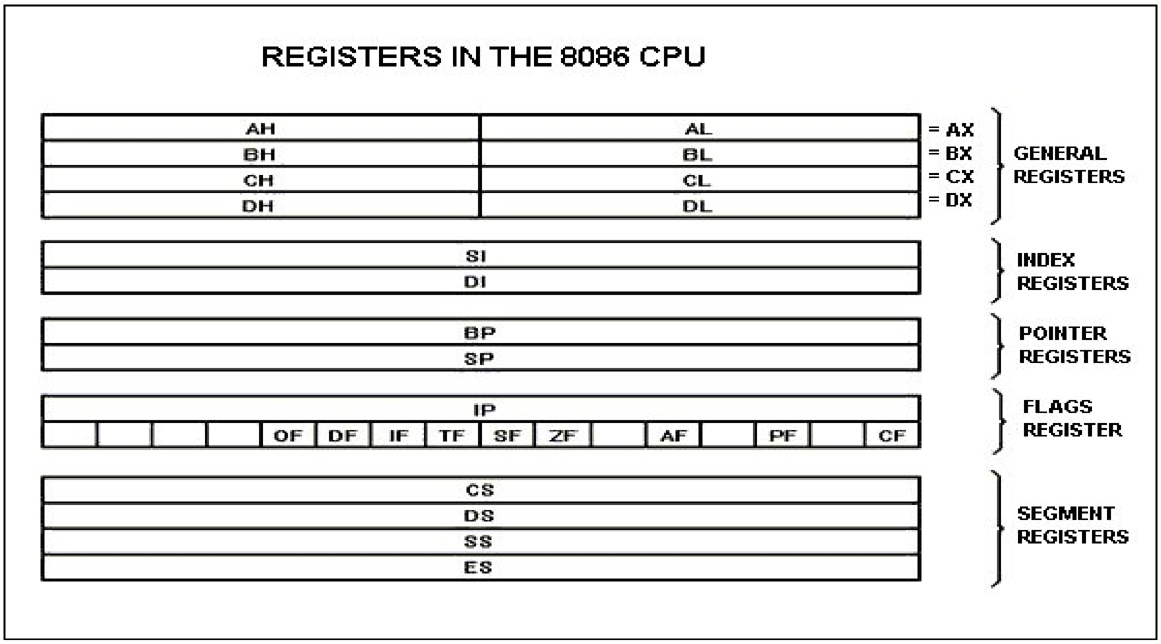
Inside the EU

The EU is made up of two parts known as the ALU and the general registers. It is here that instructions are received, decoded, and executed from the instruction queue portion of BIU. The instructions are taken from the top of the instruction queue on the first-in, first-out, or FIFO, basis.

ALU

The ALU is the calculator part of the execution unit. It consists of electronic circuitry that performs arithmetic operations or logical operations on the binary represented electrical signals. The control system for the execution unit can also be thought of as part of ALU. It provides a path for the flow of instructions into the ALU, the general registers, and the flag register.

REGISTER ORGANIZATION OF 8086:



GENERAL PURPOSE REGISTERS

8086 CPU has 8 general purpose registers, each register has its own name:

AX - the accumulator register (divided into **AH / AL**):

1. Generates shortest machine code
2. Arithmetic, logic and data transfer
3. One number must be in AL or AX
4. Multiplication & Division
5. Input & Output

BX - the base address register (divided into **BH / BL**).

CX - the count register (divided into **CH / CL**):

1. Iterative code segments using the LOOP instruction
2. Repetitive operations on strings with the REP command
3. Count (in CL) of bits to shift and rotate

DX - the data register (divided into **DH / DL**):

1. DX:AX concatenated into 32-bit register for some MUL and DIV operations
2. Specifying ports in some IN and OUT operations

SI - source index register:

1. Can be used for pointer addressing of data
2. Used as source in some string processing instructions
3. Offset address relative to DS

DI - destination index register:

1. Can be used for pointer addressing of data
2. Used as destination in some string processing instructions
3. Offset address relative to ES

BP - base pointer:

1. Primarily used to access parameters passed via the stack
2. Offset address relative to SS

SP - stack pointer:

1. Always points to top item on the stack
2. Offset address relative to SS

3. Always points to word (byte at even address)
4. An empty stack will had SP = FFFEh

SEGMENT REGISTERS

CS - points at the segment containing the current program.

DS - generally points at segment where variables are defined.

ES - extra segment register, it's up to a coder to define its usage.

SS - points at the segment containing the stack.

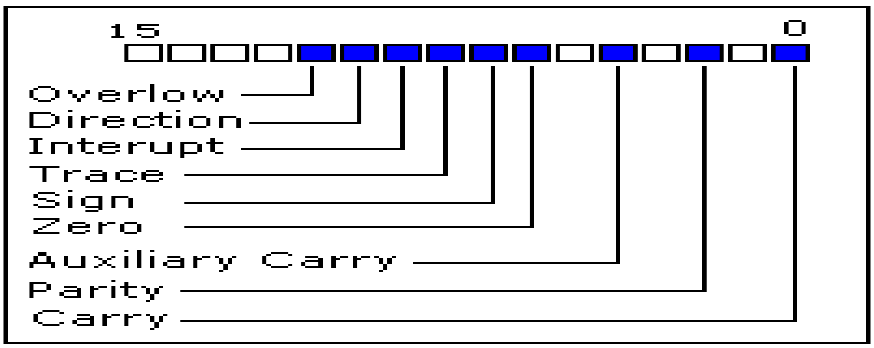
Although it is possible to store any data in the segment registers, this is never a good idea. The segment registers have a very special purpose - pointing at accessible blocks of memory.
IP - the instruction pointer:

1. Always points to next instruction to be executed
2. Offset address relative to CS

IP register always works together with **CS** segment register and it points to currently executing instruction.

FLAGS REGISTER

Flags Register - determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. Generally you cannot access these registers directly.



1. **Carry Flag (CF)** - this flag is set to **1** when there is an **unsigned overflow**. For example when you add bytes **255 + 1** (result is not in range 0...255). When there is no overflow this flag is set to **0**.
2. **Parity Flag (PF)** - this flag is set to **1** when there is even number of one bits in result, and to **0** when there is odd number of one bits.
3. **Auxiliary Flag (AF)** - set to **1** when there is an **unsigned overflow** for low nibble (4 bits).
4. **Zero Flag (ZF)** - set to **1** when result is **zero**. For non-zero result this flag is set to **0**.
5. **Sign Flag (SF)** - set to **1** when result is **negative**. When result is **positive** it is set to **0**. (This flag takes the value of the most significant bit.)
6. **Trap Flag (TF)** - Used for on-chip debugging.
7. **Interrupt enable Flag (IF)** - when this flag is set to **1** CPU reacts to interrupts from external devices.
8. **Direction Flag (DF)** - this flag is used by some instructions to process data chains, when this flag is set to **0** - the processing is done forward, when this flag is set to **1** the processing is done backward.
9. **Overflow Flag (OF)** - set to **1** when there is a **signed overflow**. For example, when you add bytes **100 + 50** (result is not in range -128...127).

Conclusion: Thus I have studied the architecture of 8086 microprocessor along with register organization of 8086 and flag register.

Rubrics for Practical Assessment:

Cognitive (3)	**Affective (3)**	**Psychomotor (3)**	**Total (9)**

Sign of Course Teacher with Date

<u>EXPERIMENT NO.5</u>

Aim: Write Assembly language program to print the string in 8086 using TASM

Requirement:

1. TASM Software

2. Pentium – 4 PC

3. Data transfer and copy instruction

Theory:

Initialization of variables:

Variables are declared and initialized in the data segment part of segment register. For string ie for data string it is initialized as data byte and string specified such as 0dh,0ah,'$' are also declared.

Code Segment:

1) Assume CS as code segment and DS as data segment.

2) Data segment is initialized using the following instructions-

   Mov ax , data

   Mov ds , ax

3) Then to print message we have use dos prompt function that is display routine that print string from DS:DX

   Mov dx, offset msg

   Mov ah,09h

Int 21h

4) Then for returning back from dos prompt we have to call clear screen routine
   Mov ah , 4ch

   Int 21h

Conclusion: Thus, we have studied & executed assembly language program using TASM to print the string

Rubrics for Practical Assessment:

Cognitive (3)	Affective (3)	Psychomotor (3)	Total (9)

Sign of Course Teacher with Date

EXPERIMENT NO.6

Aim: Write Assembly language program for 8 bit addition and 16 bit addition in 8086.

Objective:- a) For 8-bit addition-

a1=01h

a2=02h

Store the result in a3 (ASCII) =a1+a2

b) For 16-bit addition-
b1=0001h

b2=0002h

Store the result in b3 (ASCII) =b1+b2

Requirements:-1) TASM software

2) Pentium-4 PC

3) Data and arithmetic instructions

Theory:-
Initialization of variables: Variables are initialized in the data segment part of segment register. For 8-bit data, it is initialized as data byte (DB) and for 16-bit data it is initialized as data word (DW).
Code Segment:-
1) Assume CS as code segment and DS as data segment.
2) Data segment is initialized using the following instructions-
   Mov ax , data
   Mov ds , ax
3) Then we move the first variable into al(for 8-bit dat) or in ax(for 16-bit data)
   i.e. mov al , a1

or mov ax , b1

4) Add the second variable to the first.
   i.e. add al , a2

   or add ax , b2

5) Pass the result to the third variable.
   i.e. mov a3 , al

   or mov b3 , ax

6) Display Routine-

   Mov dx , a3

   Mov ah , 02h

   Int 21h

7) Clear Screen Routine-

   Mov ah , 4ch

   Int 21h

Conclusion: Thus I have studied and executed the ALP for addition of two 8 bit and two 16 bit numbers in 8086 using TASM.

Rubrics for Practical Assessment:

Cognitive (3)	Affective (3)	Psychomotor (3)	Total (9)

Sign of Course Teacher with Date

EXPERIMENT NO.7

Aim: Write Assembly language program for 8 bit subtraction and 16 bit subtraction in 8086.

Objective:-

a) For 8-bit Subtraction-
   a1=01h

   a2=02h

   Store the result in a3(ASCII)=a1-a2

b) For 16-bit Subtraction-
   b1=0002h

   b2=0001h

   Store the result in b3(ASCII)=b1-b2

Requirements:-

1) TASM software
2) Pentium-4 PC
3) Data and arithmetic instructions

Theory:-

Initialization of variables-

Variables are initialized in the data segment part of segment register. For 8-bit data, it is initialized as data byte (DB) and for 16-bit data it is initialized as data word (DW).

Code Segment:-

1) Assume cs as code segment and ds as data segment.

2) Data segment is initialized using the following instructions-

   Mov ax , data

   Mov ds , ax

3) Then we move the first variable into al(for 8-bit data) or in ax(for 16-bit data)

   mov al, a1

   or mov ax , b1

4) Subtract the second variable from the first.
   i.e. sub al , a2

   or sub ax , b2

   Mov dx , result

5) Pass the result to the third variable.
   i.e. mov result , al

   or mov result , ax

6) Display Routine-

   Mov ah , 02h

   Int 21h

7) Clear Screen Routine-
   Mov ah , 4ch

   Int 21h

Mov dx , result

Conclusion: Thus I have studied and executed the assembly language program for subtraction of two 8 bit and 16 bit numbers using TASM.

Rubrics for Practical Assessment:

Cognitive (3)	Affective (3)	Psychomotor (3)	Total (9)

Sign of Course Teacher with Date

EXPERIMENT NO.8

Aim: Write Assembly language program for 16 bit multiplication in 8086 using TASM.

Objective:-

c) For 8-bit multiplication-
   a1=01h

   a2=02h

   store the result in a3(ASCII)=a1-a2

d) For 16-bit multiplication-
   b1=0002h

   b2=0001h

   store the result in b3(ASCII)=b1-b2

Requirements:-

1) TASM software

2) Pentium-4 PC

3) Data and arithmetic instructions

Theory:-

Initialization of variables-

Variables are initialized in the data segment part of segment register. For 8-bit data, it is initialized as data byte (DB) and for 16-bit data it is initialized as data word (DW).

Code Segment:-

1) Assume cs as code segment and ds as data segment.
2) Data segment is initialized using the following instructions-
   Mov ax , data

   Mov ds , ax

3) Then we move the first variable into al(for 8-bit data) or in ax(for 16-bit data)
   mov al, num1

   or mov ax , num1

4) Multiply the second variable with the first variable.
   i.e. mul al , num2

   or mul ax, num2

5) Pass the result to the third variable.
   i.e. mov result , al

   or mov result , ax

6) Display Routine-
   Mov dx , result

   Mov ah , 02h

   Int 21h

7) Clear Screen Routine-
   Mov ah , 4ch

Int 21h

Finally the code ends

i.e. code ends

end

Conclusion: Thus I have studied and performed the assembly language program for multiplication of two 16 bit numbers using TASM.

Rubrics for Practical Assessment:

Cognitive (3)	Affective (3)	Psychomotor (3)	Total (9)

Sign of Course Teacher with Date

EXPERIMENT NO.9

Aim: Write an Assembly Language Program for finding smallest number from an array in 8086

Objective:-

To find smallest number from a list

List = 56h, 05h, 61h

Store result in variable called as small

Requirements:-

1) TASM software
2) Pentium-4 PC
3) Data and arithmetic instructions

Theory:-

Initialization of variables-

Array is initialized in the data segment part of segment register. All array elements are 8 bit so array initialized with 8 bit and initialize count by number of elements in array – 1.

Code Segment:

1) Assume CS as code segment and DS as data segment.
2) Data segment is initialized using the following instructions-
   Mov ax , data

   Mov ds , ax
3) Then store the offset of the array list into SI means SI will point to the first memory location of the array.

4) Store the content of the location which is pointed by SI that is the first element of the array into AL register.
5) Move count into CL register.
6) Use loop to find out the smallest from the array for that first compare the first element with the next element in an array then while comparing there are two cases:
   i) carry is generated: if carry is generated means first element is smaller than second, so move that second element into first element ie in AL.
   ii) carry is not generated: if carry is not generated then jump on skip routine
7) Then increment SI so that it points to next element and decrement CL that is count by 1.

8) Then check whether CL ie count become zero or not.if it is not zero repeat the loop till it become
zero.

from given unordered array using TASM

Rubrics for Practical Assessment:

Cognitive (3)	Affective (3)	Psychomotor (3)	Total (9)

Sign of Course Teacher with Date

EXPERIMENT NO.10

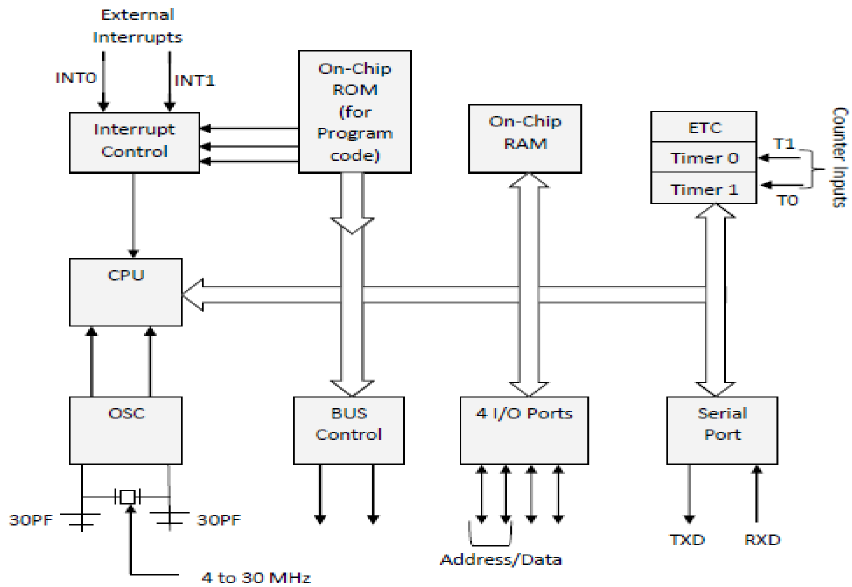
Aim: Write a program for LED blinking in 8051 microcontroller

Theory:

A Microcontroller has all the necessary components which a microprocessor possesses and invariably it poses ROM, RAM, Serial Port, timers, interrupts Input Output ports, and clock circuit. The microcontroller always focus on the chip facility and it is more prominent in the case of serial ports, analog-to-digital converters, timers, counters, read only memory, parallel input, interrupt control, random access memory and output ports.

8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package), 4kb of ROM storage and 128 bytes of RAM storage, 2 16-bit timers. It consists of are four parallel 8-bit ports, which are programmable as well as addressable as per the requirement. An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12 MHz.

Architecture of 8051 Microcontroller



What is Microcontroller 8051?

A lot has been said about the 8051 microcontroller and after coming towards the end of the article you might be aware about the various aspects of the 8051 microcontroller. This microcontroller was invented by the Intel and it works with a 8 bit family processor. When it comes to the use the 8051 microcontroller has extensive application in various industries and in domestic purpose also.

History of the 8051 Microcontroller

If we will go back to history the 8051 microcontroller was first invented in the year 1980 by the microprocessor giant Intel and gradually it has been accepted worldwide and with the every coming days the importance of the 8051 microcontroller is escalating. When it was invented by the Intel, it was developed by means of NMOS technology, but as NMOS technology but it was not very effective.

Company	Processor	Year
INTEL 4004	4-bit	1971
INTEL 8085	8-bit	1974
INTEL 8048	8-bit	1976
INTEL 8031	8-bit(ROM-LESS)	-
INTEL 8051	8 bit(MASK ROM)	1980
INTEL 8086	16-bit	1978
Atmel At89C51	8-bit(Flash Memory)	1984
Microchip PIC16C64	8-bit	1985
Motorola 68HC11	8-bit(on chip ADC)	1985
AVR	8-bit RISC	1996

8051 Microcontroller Programming

8051 microcontroller programming is certainly very interesting and to make it even interesting here we will give you some tools which will help you to understand the 8051 microcontroller programming in a better way.

Have a Look at the Tools

- Code editor -Syntax highlighting Notepad
- RIDE software – simulation
- A51-Assembler
- Proteus – Fully embedded simulation software
- Simulator-windows based Smart n Small Simulator
- Keil uVision – 8051/ARM simulation
- Baud -Timer value calculators for various baud rates

Now we will write the program as per the Keil Uvison4 simulation software and the program is

- Install software on your system
- Click Project -> New Uvision Project
- Save your project
- Select Target Device (8051 – AT89s51)
- File -> New
- New text-editor will be opened. Here you need to write your code

Algorithm:

1. Declare the pin1.0 as LED so that its easy to use it in our code in future.

2. Declare two functions. One of them is the delay function which is just adding the delay, while the second function is for initialization of Port 1 as output port.
3. In the Main function, use LED blinking code in which LED is ON and then OFF continuously and so that make it blink.
4. After writing the code in your Keil software, compile it and run.

Conclusion: Thus I have executed the LED blinking program in 8051.

Rubrics for Practical Assessment:

Cognitive (3)	**Affective (3)**	**Psychomotor (3)**	**Total (9)**

Sign of Course Teacher with Date