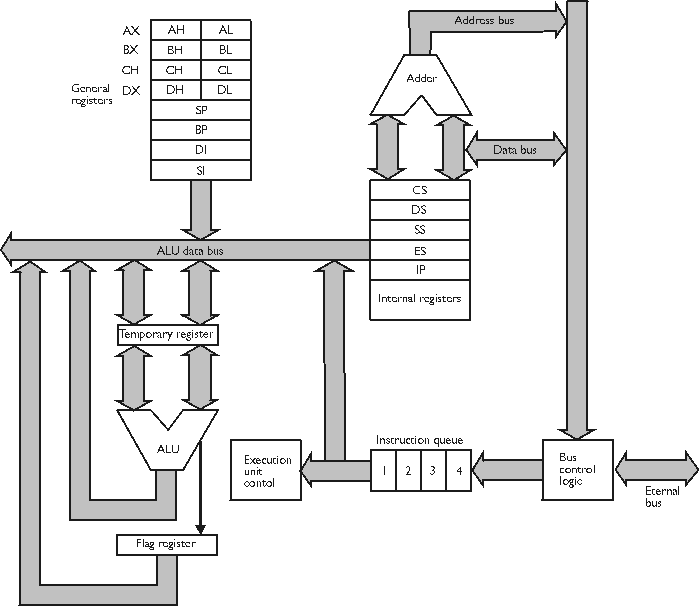
**LAB MANUAL 02**

**Implementation of 8086 based Data Movement and Arithmetic Instructions such as ADD, SUB, MOV using EMU8086**

## Difference between 8086 and 8088 microprocessor:-

The only significant difference between the 8088 microprocessor and the 8086 microprocessor is the BIU. In the 8088, the BIU data bus path is 8 bits wide Where the 8086 BIU data bus is l6-bit. Another difference is that the 8088 instruction queue is four bytes long instead of six.



**8086 Programming Model:-**

The programming model for a microprocessor shows the various internal registers that are accessible to the programmer. The Following Figure is a model for the 8086. In general, each register has a special function.



**More about Registers:**

**SI - source index register:-**

1. Used as source in some string processing instructions

2. Offset address relative to DS

**DI - destination index register:-**

1. Used as destination in some string processing instructions

2. 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 of the stack

2. Offset address relative to SS

**Segment Registers:-**

• CS (Code Segment)

**CS** - points at the segment containing the current program.

• DS (Data Segment)

DS- Points to the segment containing the data.

• SS (Stack Segment)

**SS - points at the segment containing the stack.**

• ES (Extra Segment)

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

**FLAGS Register**

• Zero flag

• Sign flag

• Parity flag

• Carry flag

• Overflow flag

.

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.

Segment registers work together with general purpose register to access any memory value. For example if we would like to access memory at the physical address **12345h**(hexadecimal), we could set the **DS = 1230h** and **SI = 0045h**. This way we can access much more memory than with a single register, which is limited to 16 bit values. The CPU makes a calculation of the physical address by multiplying the segment register by 10h and adding the general purpose register to it (1230h \* 10h + 45h = 12345h):

**Instruction Set:-**

What is an instruction set? When we talk to some person we need to talk in the language which that person is able to understand. If we talk in a language that is not understandable by that person, he will not be able to follow our words and will not be able to reply us or fulfill the task we are asking him to do. This is because we are not understandable to him, which means that we are not following his instruction set.

Same is the case with microprocessor, if the microprocessor is able to understand our words(or instructions by us), it will be able to perform our said tasks, otherwise will not be able to do it. That is only possible if we talk to microprocessor in its language i.e. language using its **INSTRUCTION SET**. Each microprocessor has its own instruction set.

“Instruction set is the complete set of all the instructions that can be recognized and executed by a central processing unit. Also called a *command set*, the basic set of commands or [instructions](http://www.webopedia.com/TERM/I/instruction.html), that a microprocessor understands.”

**8086 Instruction Set:-**

8086 Instructions can be divided into different classes, which are data movement instructions, conversions, Arithmetic Instructions, Logical Shift Rotate and Bit Instructions, I/O Instructions String Instructions, Program Flow control instructions and other Miscellaneous Instructions.

For now, we will be focusing on Data Movement and some of the Arithmetic Instructions.

**Understanding the instructions:**

**ADD:**

 It adds the source operand to destination operand and places the result in destination.

**Format: - ADD destination, source**

**SUB:**

It subtracts byte from byte or word from word.

Format: operand1 = operand1 - operand2

Example:

MOV AL, 5

SUB AL, 1 ; AL = 4

*Where AL=4 is comment in the above mentioned code*

**Lab Tasks**

**Execute the following tasks CLO [1]**

**TASK 1:**

Write and assemble a program to load register AX with value 99H. Then from register AX move it to BX, CX, and DX. Use the simulator to single-step the program and examine the registers.

**TASK 2**

Write and assemble a program to add all the single digits of your ID number and save the result in Accumulator. Pick 7 random numbers (all single digit) if you do not want to use your ID number. Then use the simulator to single-step the program and examine the registers.

**TASK 3**

Subtraction of two 8 bit numbers and place the result in accumulator register

**TASK 4**

Add sum of series of first 9 numbers and save it into one of the register. Again take the sum of first 9 numbers and save it to another register. The contents of both the register must be added finally and save the result in the third register.

**TASK 5**

Addition of first ten natural numbers by using INC and ADD instruction.

**TASK 6**

**X=(A+B) – (C+D)**

*Implement the following equation and place the result in accumulator register*