

**NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY**

School of Electrical Engineering and Computer Sciences

**Computer Organization & Assembly Language**

**ASSIGNMENT NO. 1,2,3**

**Title of the assignment: Creating an 8086 Simulator**

**SUBMITTED TO: Sir Imran Abeel**

**SUBMITTED BY: Sohaib Ahsan (383611), Mujtaba Shafqat (367935), Samama Farooq (373810) , Azaan Siddiq (371850), Habibullah Khan (372880)**

**CLASS - SECTION: BSCS-11 B**

**Last Date of Submission: 5th December, 2022**

**Description Of 8086:**

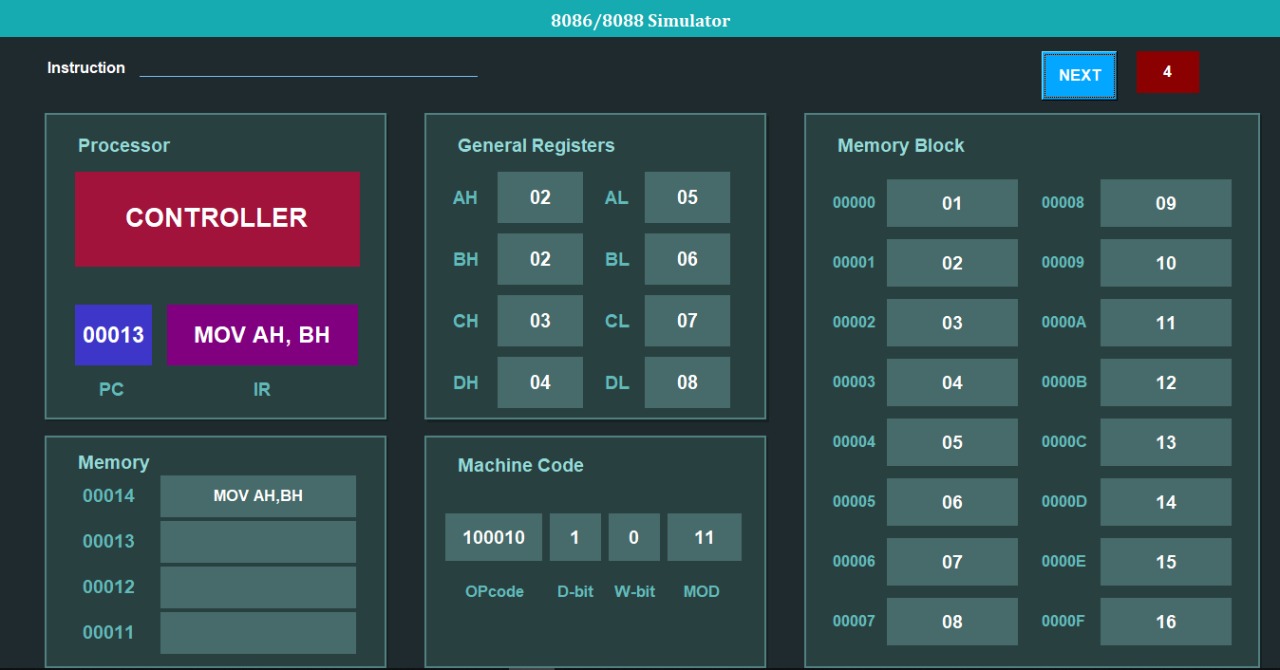
8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

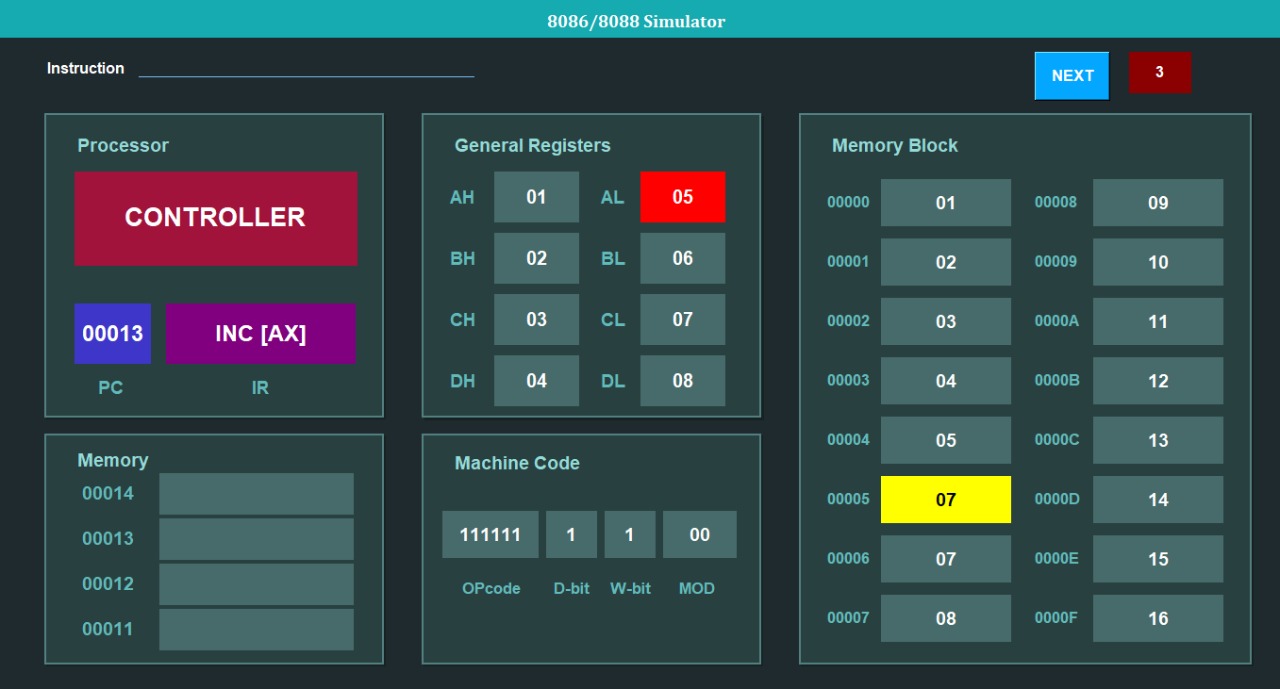
It supports two modes of operation, i.e., Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

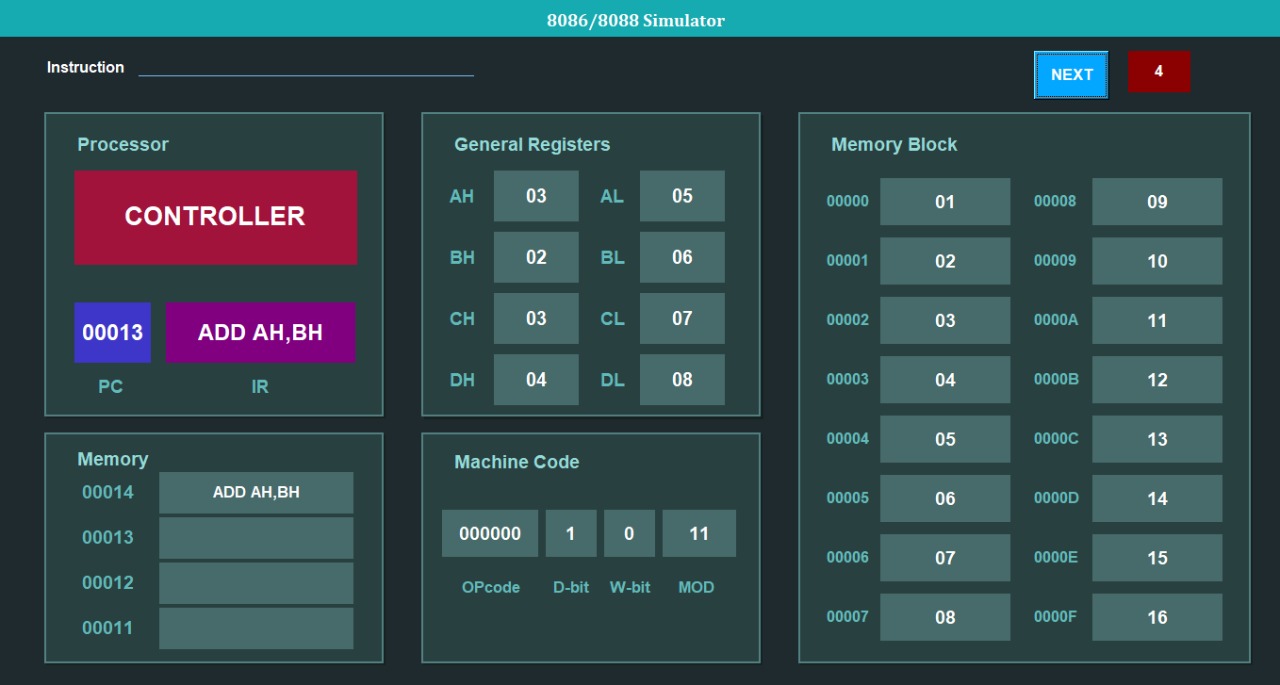
**Description:**

In this project we have used JavaScript ,CSS, html .On front end there is textbox for instruction input with box showing machine code We have used 8 registers al, ah, bl, bh, cl, ch, dl, dh with sixteen memory locations along with CPU.

**Screenshot:**

****





**Instructions:**

**Instruction Set:**

MOV REG REG

MOV REG [MEM]

MOV REG [REG]

MOV [MEM] REG

MOV [REG] MEM

MOV REG IMM

INC REG

INC [REG]

DEC REG

DEC [REG]

ADD REG REG

ADD REG [REG]

ADD [REG] REG

ADD REG [MEM]

ADD [MEM] REG

ADD REG IMM

SUB REG REG

SUB REG [REG]

SUB [REG] REG

SUB REG [MEM]

SUB [MEM] REG

SUB REG IMM

We have simulated the following instructions and their sub-instructions as following:

1. **Mov:**

**Scenarios:**

* Register to Register
* Memory to register
* Register to memory
* Immediate to register
* Immediate to memory

**Checks:**

We have restricted user only to enter valid memory locations and along with valid names of registers otherwise unable to perform instruction .We also implemented the check that 16 bit register can’t move to 8 bit registers

**Implementation:**

When user enters valid name of register while moving data from register to register it goes to that particular register address fetch its content and then updated that content to the destination register same logic is applied to memory

1. **Add:**

**Scenarios:**

* register to register
* register to memory
* memory to register
* direct to register
* direct to memory

**Checks:**

In addition along with checking correct names of instruction in addition we have applied overflow checks Let data we adding is greater than ff and register is 8 bit then overflow happens same is case with 16 bit register

Implementation**:**

Fetching data from registers mentioned in instruction then converting their data to hex to number adding there content and then converting that to hex and updating that to destination

1. **Subtract:**

**Scenarios:**

* register to register
* register to memory
* memory to register
* direct to register
* direct to memory

**Checks:**

In subtraction along with checking correct names of instruction in subtraction we have applied overflow checks Let data we adding is greater than ff and register is 8 bit then overflow happens same is case with 16 bit register

**Implementation:**

Fetching data from registers mentioned in instruction then converting their data to hex to number subtraction there content and then converting that to hex and updating that to destination

1. **Mul:**

**Scenarios:**

* register to register
* register to memory
* memory to register

**Checks:**

In multiplication along with checking correct names of instruction in subtraction we have applied overflow checks Let data we are multiplying is greater than if and register is 8 bit then overflow happens same is case with 16 bit register

**Implementation:**

Fetching data from registers mentioned in instruction then converting their data to hex to number multiplying there content and then converting that to hex and updating that to ax.

1. **Increment:**

**Scenarios:**

* register
* memory

**Checks:** We simply don’t need to apply checks on it.

**Implementation:**

Fetching data from 8 bit register, 16 bit register or memory and then updating data there after performing increment operation

1. **Decrement:**

**Scenarios:**

* Register
* memory

**Checks:** We simply don’t need to apply checks on it

**Implementation:**

Fetching data from 8 bit register, 16 bit register or memory and then updating data there after performing decrement operation

**File Names:**

**Main.py:** we have written main file for the implementation of GUI and running of all the function in different files.

**Processing.py:**

We have designed to work basically as a backend for the whole project for the implementation of different instructions such as mov, add, sub etc.

**Poriorqueue.py:**  
 we have implemented a data structure named as queue for insertion of instruction.

**GitHub:**

As we were working in team, we have used GitHub to collaborate, and every member committed code accordingly as every file is included in GitHub repository you can clone and use our project from GitHub as it is made public.