

**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: Adnan Hassnain, Afaq Ahmad, Allah Rkha ,**

**Umer Mehmood , Hussain Hammad Naqvi**

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

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

**Description Of 8086:**

8086 Microprocessor is an enhanced version of 8085Microprocessor 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:**

We have simulated 15 instructions as following:

1. **Mov:**

Scenarios**:**

reg to reg, memory to register, reg 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 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 multiplying there content and then converting that to hex and updating that to ax

1. **And :**

Scenarios**:**

register to register, register to memory, memory to register

Checks**:** We have applied checks that user can’t perform and on 8 bit register along with 16 bit register their size must be same to perform and operation

Implementation**:**

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

1. **OR**

Scenarios**:**

register to register, register to memory, memory to register)

Checks**:** We have applied checks that user can’t perform or on 8 bit register along with 16 bit register their size must be same to perform and operation

Implementation**:**

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

1. **Not:**

Scenarios**:**

register, memory

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

Implementation**:**

Fetching data from 8 bit register, 16 bit register or memory and then updating data there after performing not operation

1. **Neg :**

Scenarios**:**

register, memory

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

Implementation**:**

Fetching data from 8 bit register, 16 bit register or memory and then updating data there after performing not operation

1. **XOR:**

Scenarios**:**

register to register, register to memory, memory to register

Checks**:** We have applied checks that user can’t perform or on 8 bit register along with 16 bit register their size must be same to perform and operation

Implementation**:**

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

1. **Left shift:**

Scenarios**:**

register, memory

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

Implementation**:**

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

1. **Right Shift :**

Scenarios**:**

register, memory

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

Implementation**:**

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

1. **Left Rotate :**

Scenarios**:**

register, memory

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

Implementation**:**

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

1. **Right Rotate :**

Scenarios**:**

register, memory

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

Implementation**:**

Fetching data from 8 bit register, 16 bit register or memory and then updating data there after performing not operation

1. **Increment:**

Scenarios**:**

register, memory

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

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 not

Implementation**:**

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

**File Name:**

Index.html**:** we have written html code

Style.CSS:

we have design and styled our project

script.js:  
 we have implemented our logic

**Major Functions:**

Checking Negative:

function negt(x, num) {

  for (var j = 0; j < x; j++) {

    if (num[j] === "0") {

      num = num.slice(0, j) + "1" + num.slice(j + 1, x);

    } else {

      num = num.slice(0, j) + "0" + num.slice(j + 1, x);

    }

  }

Checking Is register:

function isReg(opr) {

  if (reg16.includes(opr) || reg8.includes(opr)) {

    return true;

  } else {

    return false;

  }

}

Checking Type of reg:

function reg8or16(opr) {

  if (reg16.includes(opr)) {

    return 2;

  } else if (reg8.includes(opr)) {

    return 1;

  }

}

For And:

// }

function andFun(oprnd1, oprnd2) {

return oprnd1 & oprnd2;

}

For Xor:

function xorFun(oprnd1, oprnd2) {

  return oprnd1 ^ oprnd2;

}

For Or:

function orFun(oprnd1, oprnd2) {

  return oprnd1 | oprnd2;

}

Decimal to hex:

const decimalToHex = (dec) => dec.toString(16);

Hex to decimal:

const hexToDecimal = (hex) => parseInt(hex, 16);

Decimal to binary:

num =num.toString(2)

**Implantation:**

We have saved our registers in array, along with address of memory in other array we have used dictionary to save name of registers while performing some instruction these things have been used we have applied checks on instructions allowed only to enter correct instruction

For example on move user can only only enter valid name of registers and same chjeck is also appl

**Memory:**

In memory implementation we are getting address of one digit from user but while passing we are using 4 digit address

Function:

function memLoc(opr) {

  if (opr[0] === "[" && opr[opr.length - 1] === "]") {

    var mem = opr.slice(1, opr.length - 1);

    if (mem.length === 2) {

      if (reg16.includes(mem) || reg8.includes(mem)) {

        return Number(hexToDecimal(regVar[`${mem}`].textContent));

      }

    } else if (mem.length === 1 && isHexadecimal(mem)) {

      mem = hexToDecimal(mem);

      return Number(mem);

    }

  }

}

Refresh Memory:

function refreshMem() {

  for (var i = 0; i < 16; i++) {

    memArr[i].classList.remove("active1");

    memArr[i].classList.remove("active2");

  }

}

**GitHub:**

As we were working in team, we have used GitHub to collab, and every member committed code accordingly as every file is included in GitHub repository you can install and use our project from GitHub