**15ECE304 - Microprocessor and Microcontroller**

MINI PROJECT REPORT

VOTING MACHINE USING AN INSTRUCTION LEVEL 8085

**Submitted by:**

Aditya Nair - CB.EN.U4ECE18003

S Amreith Venkatesh - CB.EN.U4ECE18005

A Charan - CB.EN.U4ECE18017

Maansi Prerna - CB.EN.U4ECE18038

Tayi Nithya - CB.EN.U4ECE18066

**Introduction:**

A voting machine is a machine used to record or tally votes. The first voting machines were mechanical but it is increasingly more common to use electronic voting machines. Traditionally, a voting machine has been defined by its mechanism, and whether the system tallies votes at each voting location, or centrally.

Voting machines differ in usability, security, cost, speed, accuracy, and ability of the public to oversee elections. Machines may be more or less accessible to voters with different disabilities.

Different types of voting machines used before are,

1. Buttons
2. Balls
3. Tokens
4. Analog Computers
5. Dials
6. Levers

Different types of voting machines used now

1. Optical Scan
2. Direct-recording Electronic (DRE)

In countries like India, Electronic voting systems are used widely.

**Theory:**

With candidate ID voters can choose a candidate and each candidate is allocated to a particular memory address.

List of candidates Candidate ID Memory Address

|  |  |
| --- | --- |
| 1st candidate 01H | 5001H |
| 2nd candidate 02H | 5002H |
| 3rd candidate 03H | 5003H |
| 4th candidate 04H | 5004H |
| 5th candidate(NOTA) 05H | 5005H |

We initialise the memory location as 4000 from where we need to start storing the voters’ entry.

And the vote count for every candidate is stored at their respective memory locations.

The keys send a particular hexadecimal number as input which are:

|  |  |
| --- | --- |
| 01H – | for voting for candidate 1 |
| 02H – | for voting for candidate 2 |
| 03H –  04H –  05H – | for voting for candidate 3  for voting for candidate 4  for NOTA |

These keys are used by the voters to cast votes for their preferred candidates The counting in-charge possesses a key for starting the counting process

**Initializing:**

MVI H,40H

MVI L,00H

MVI C,0FH

MVI B,00H

We will first initialize the registers and HL pair with respect to the memory locations of voters list obtained using DB instruction.

**Major looping part:**

MVI 01

CMP M

JNZ SEC

MVI D,50

MVI E,01

LDAX D

INR A

STAX D

INX H

JNZ COMP

The loops FIRST,SEC,THI,FOUR,NOTA works in a similar way.All these loops initializes with respective candidate ID and compares with the voters list if it matches, then moves to respective memory location of the candidate using MVI instruction and increments the value in that memory location(Here with the help of A,D,E registers).Increment memory location so as to change the value in voters list and jump to COMP loop.If the values in A register and value in voters list does not match after CMP instruction it directly jumps to the next candidate ID.

**Comparing:**

STA 4500

MOV A,B

CMP C

JZ END

LDA 4500

INR B

JNZ FIRST

The voting process is terminated with the help of the COMP loop which checks if the votes of all the voters have been accounted for.

In the COMP loop initially using STA instruction A register stores the value in A to a memory location 4500H.Then using MOV instruction moves the data from B register to A register.Compares A register and C register to make sure the length of the voters list is equal with A register. If so it will jump to the END loop and then HLT(As we know if both the registers are equal it’s zero flag will be 1).If not A register will load the value from memory location and increments B register and jumps back to First loop

**Pseudo Code:**

initialize values of HL pair

FIRST loop:

if value at A is not 1:

jump to SEC loop

else:

increment the value at 1st position

if no more entries:

jump to COMP loop

SEC loop:

if value at A is not 2:

jump to THI loop

else:

increment the value at 2nd position

if no more entries:

jump to COMP loop

THI loop:

if value at A is not 3:

jump to FOUR loop

else:

increment the value at 3rd position

if no more entries:

jump to COMP loop

FOUR loop:

if value at A is not 4:

jump to NOTA loop

else:

increment the value at 4th position

if no more entries:

jump to COMP loop

NOTA loop:

if value at A is not 5:

jump to COMP loop

else:

increment the value at 5th position

if no more entries:

jump to COMP loop

COMP loop:

move value of A to address 4500

move value of B to A

if value of A is same as C (which contains number of voters):

jump to END

else:

set value at 4500 back to A

increment B

jump to FIRST loop

**Source Code:**

MVI H,40

MVI L,00

MVI C,0F

MVI B,00

FIRST:

MVI A,01

CMP M

JNZ SEC

MVI D,50

MVI E,01

LDAX D

INR A

STAX D

INX H

JNZ COMP

SEC: MVI A,02

CMP M

JNZ THI

MVI D,50

MVI E,02

LDAX D

INR A

STAX D

INX H

JNZ COMP

THI: MVI A,03

CMP M

JNZ FOUR

MVI D,50

MVI E,03

LDAX D

INR A

STAX D

INX H

JNZ COMP

FOUR:

MVI A,04

CMP M

JNZ NOTA

MVI D,50

MVI E,04

LDAX D

INR A

STAX D

INX H

JNZ COMP

NOTA:

MVI A,05

CMP M

JNZ COMP

MVI D,50

MVI E,05

LDAX D

INR A

STAX D

INX H

JNZ COMP

COMP:

STA 4500

MOV A,B

CMP C

JZ END

LDA 4500

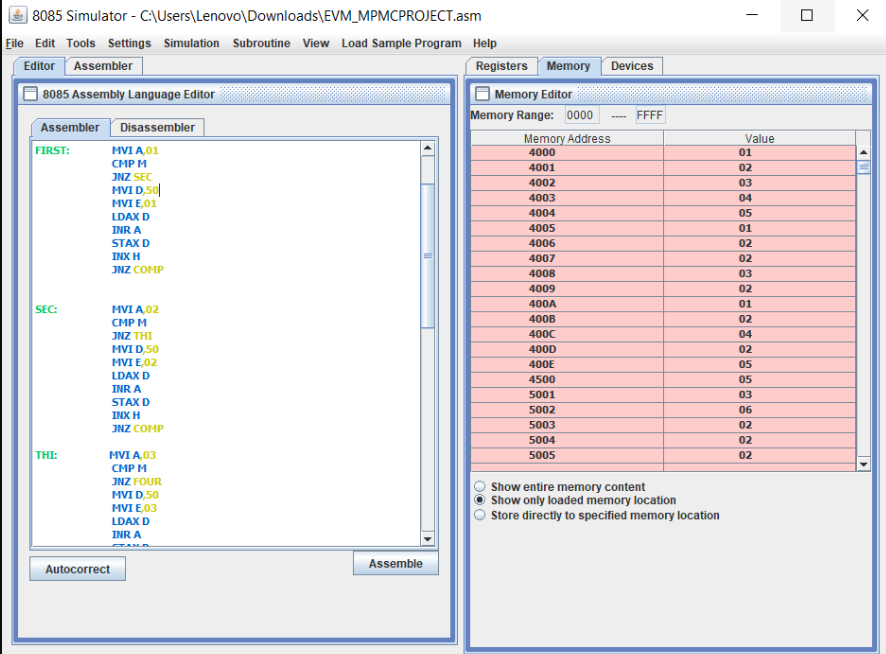
INR B

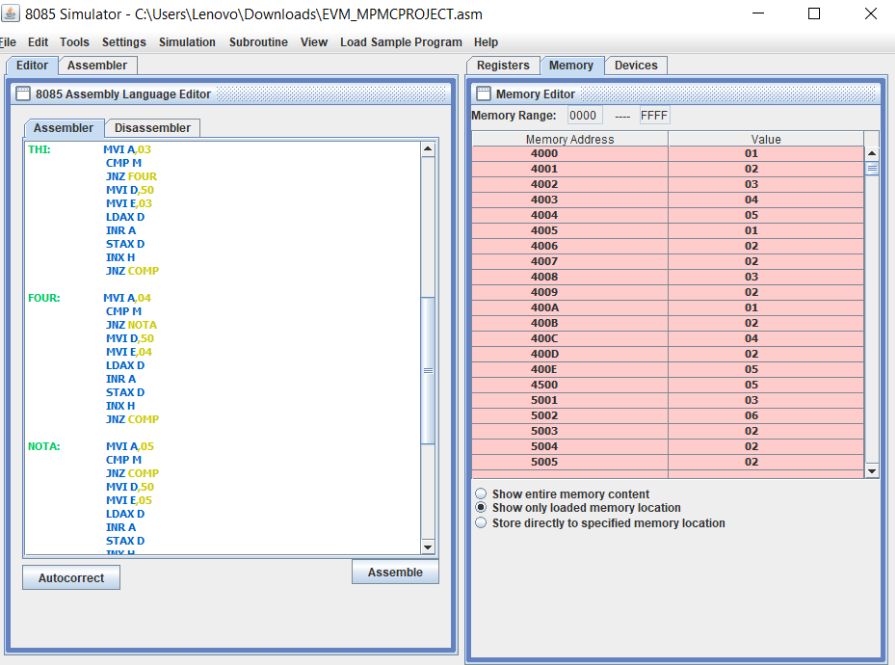
JNZ FIRST

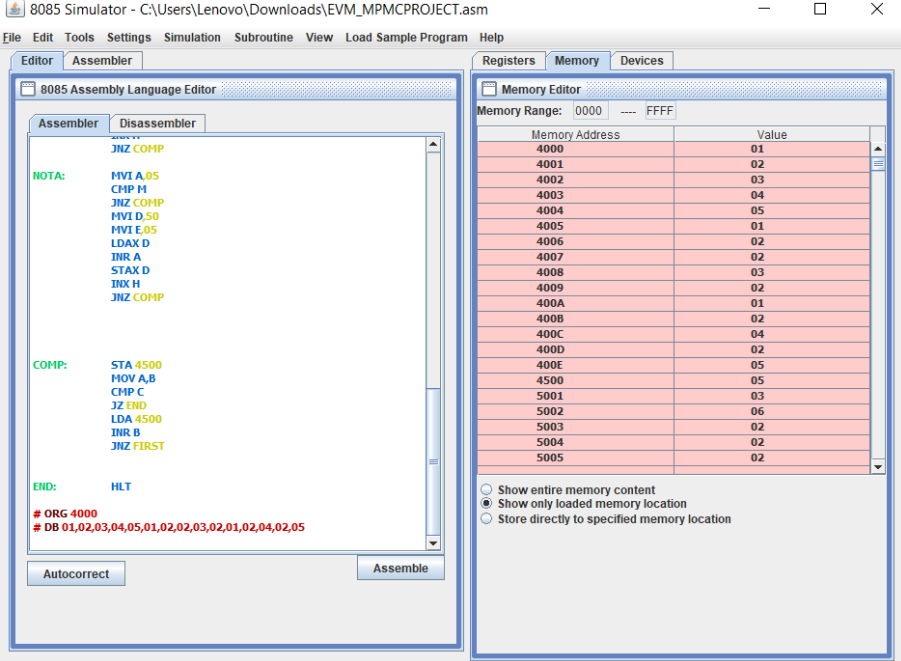
END: HLT

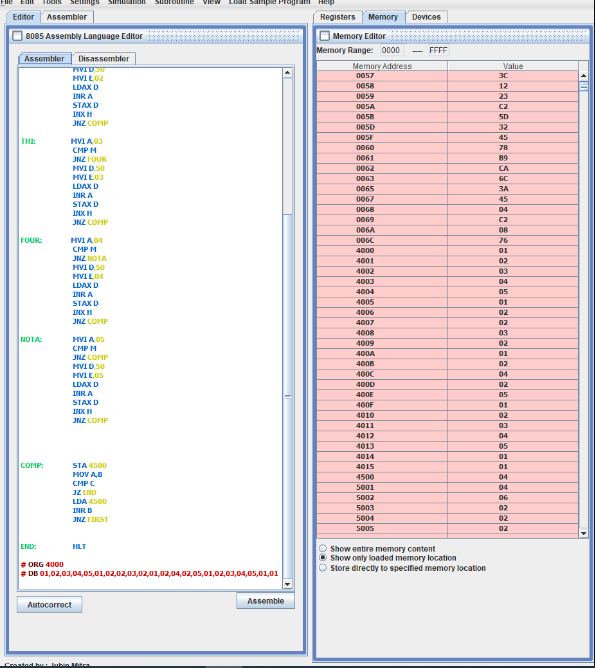
# ORG 4000

# DB 01,02,03,04,05,01,02,02,03,02,01,02,04,02,05









Instructions used

* MVI

MVI is a mnemonic, which actually means “Move Immediate”. With this instruction,we can load a register with an 8-bitsor 1-Bytevalue. This instruction supports immediate addressing mode for specifying the data in the instruction.

* CMP

In the 8085 Instruction set, CMP is a mnemonic that stands for CoMPare accumulator. This instruction is used to compare contents of the Accumulator with the given register R. The result of the compare operation will be stored in the Temp register.

* JNZ

In the 8085 Instruction set, we are having one mnemonic **JNZ** , which stands for “Jump if Not Zero”. This instruction is used to jump to the address a16 as provided in the instruction. But as it is a conditional jump so it will happen if and only if the present zero flag value is 0. If zero flag value is 1, program flow continues sequentially.

* LDAX

In 8085 Instruction set, LDAX is a mnemonic that stands for LoaD Accumulator from memory pointed by eXtended register pair denoted as “rp” in the instruction. This instruction uses register indirect addressing for specifying the data.

* INR

In the 8085 Instruction set, INR is a mnemonic that stands for ‘INcRement’ and ‘R’ stands for any of the following registers or memory location M pointed by HL pair.This instruction is used to add 1 with the contents of R. So the previous value in R will get increased by amount 1 only. The result of increment will be stored in R updating its previous content. All flags, except Cy flag, are affected depending on the result thus produced.

* STAX

In the 8085 Instruction set, STAX is a mnemonic that stands for SToreAccumulator contents in memory pointed by eXtended register denoted as “rp”.Hererp stands for register pair. This instruction uses register indirect addressing for specifying the destination. So using this instruction, the current content of Accumulator will be written to the memory location as pointed by 16-bit address as stored in the register pair.

* INX

In the 8085 Instruction set, INX is a mnemonic that stands for “INcrementeXtended register” and rp stands for register pair.This instruction will be used to add 1 to the present content of the rp. And thus the result of the incremented content will remain stored in rp itself. Though it is an arithmetic instruction, note that, flag bits are not at all affected by the execution of this instruction. A register pair is generally used to store 16-bit memory addresses. If flag bits got affected during increment of a memory address, then it may cause problems in many cases. So as per design of 8085, flag bits are not getting affected by the execution of this instruction INXrp.

* STA

In 8085, Instruction set LHLD is a mnemonic that stands for Load HL pair using Direct addressing from memory location whose 16-bit address is denoted as a16. So the previous content of the HL register pair will get updated with the new 16-bits value. The HL pair has to be updated, so data comes from two consecutive memory locations starting at the address a16 and also from the next address location. This instruction uses absolute addressing mode for specifying the data in the instruction.

* JZ

In the 8085 Instruction set, we are having one mnemonic **JZ**, which stands for “Jump Zero” .This instruction is used to jump to the address a16 as provided in the instruction. But as it is a conditional jump so it will happen if and only if the present zero flag value is 1. If the zero flag value is 0, program flow continues sequentially.

* LDA

In the 8085 Instruction set, LDA is a mnemonic that stands for LoaD Accumulator with the contents from memory. In this instructionAccumulatorwill get initialized with 8-bit content from the 16-bit memory address as indicated in the instruction as a16. This instruction uses absolute addressing for specifying the data.

* HLT

In 8085 Instruction set, **HLT** is the mnemonic which stands for ‘Halt the microprocessor’ instruction. It is having a size of 1-Byte instruction. Using these particular instructions, as 8085 enters into the halt state, so we can put the8085 from further processing of next instructions. This is indicated by S1 and S0 control signals. During the halt, S1 and S0 output signals will become 0 0.

**Software used:**

***Github 8085 simulator using Java IDE***

**Conclusion:**

Ballot paper elections give room for rigging elections especially in countries with high levels of corruption like India. Electronic voting is the hope for free and fair election. This report presents a microcontroller based electronic voting machine. The system was designed and simulated successfully. It was found working as specified by its requirements. With this system, election rigging could be reduced to a minimum.

**References:**

[www.tutorialspoint.com](http://www.tutorialspoint.com)

[www.github.com](http://www.github.com)