1. **INTRODUCTION**

The **Intel 8085** is an 8-bit microprocessor introduced by Intel in 1977. The processor has seven 8-bit registers accessible to the programmer, named A, B, C, D, E, H and L, where A is the 8-bit accumulator and the other six can be used as independent byte-registers or as three 16-bit register pairs, BC, DE, and HL, depending on the particular instruction. It also has a 16-bit stack pointer to memory. As in many other 8-bit processors, all instructions are encoded in a single byte (including register-numbers, but excluding immediate data), for simplicity. Some of them are followed by one or two bytes of data, which could be an immediate operand, a memory address, or a port number. The 8085 supported up to 256 input/output (I/O) ports, accessed via dedicated Input/output instructions—taking port addresses as operands.

Intel produced a series of development systems for the 8080 and 8085, known as the MDS-80 Microprocessor System. Currently several development systems are used for 8085 programming. We use 8085 trainer kits like Anshuman, STC etc. We write the mnemonic program and convert them manually into the equivalent hex code and enter into the trainer kit.

The 8085 Integrated Development Environment is software that provides the users with all the programming facilities of the 8085 microprocessor. The system translates the 8085 mnemonic program into its hex code equivalent while detecting and reporting errors. The system then runs the program and outputs the result at the corresponding address. It helps the programmers in many aspects like it reduces the human effort for the manual translation of mnemonic code, saves time, helps in correcting errors in the programs, helps in keeping track of the register values after execution of each instruction. And most importantly it provides a user friendly environment for the programmers.

**Key Features:**

* Provide a facility to enter and run the 8085 mnemonic programs without worrying about the hex code.
* Provide facility to save programs and reuse them later.
* Provides facility for the users to view the memory locations.
* Helps the users to detect and debug the errors.
* Search and view the details of instructions.

1. **SYSTEM STUDY**

**2.1. EXISTING SYSTEM**

Existing system is based on manual effort. We write the mnemonic program and convert it into its equivalent hex code manually. We enter this hex code into the 8085 trainer kit and run the program. Inputs and outputs can be entered and obtained respectively at the memory locations specified in the program.

* More human effort for conversion of mnemonic program into hex code.
* Time consuming.
* User cannot see the values what each register will going to get at the each line execution.
* No error messages. So it is difficult to find out errors in program.
* Programs cannot be saved for future references.

**2.2. PROPOSED SYSTEM**

The aim of proposed system is to develop software that translates the mnemonic program entered by the user into the equivalent hex code program and then run it, to obtain the output. It is a combination of an assembler which translates the mnemonic code into hex code and a compiler that run the program. A debugger is also included for finding and reporting the errors to the users. The 8085 IDE provides us with all the programming facilities of the 8085 microprocessor. Proposed system can overcome all the limitations of the existing system (trainer kit).

Advantages of the proposed system are,

* Reduces the human effort.
* Saves time.
* Ensures accuracy of computation.
* Minimum time needed for the various processing.
* Greater efficiency.
* Programmer is able to see the register values after the execution of one instruction.
* User friendliness and interactive.
* Chances of error reduced.
* Programs can be saved and reused in future.
* Auto-fill function allows easier typing of instructions.
* Search feature helps to find the details of mnemonic instructions.

**2.3. FEASIBILITY STUDY**

Feasibility study is made to see if the project on completion will serve the purpose of  
the organization for the amount of work, effort and the time that spend on it. Feasibility  
study lets the developer foresee the future of the project and the usefulness. A feasibility  
study of a system proposal is according to its workability, which is the impact on the  
organization, ability to meet their user needs and effective use of resources. Thus when a  
new application is proposed it normally goes through a feasibility study before it is  
approved for development.  
The document provide the feasibility of the project that is being designed and lists  
various areas that were considered very carefully during the feasibility study of this  
project such as Technical, Economic and Operational feasibilities

* Technical feasibility- do we’ have the technology’? If not, can we get it?
* Operational feasibility- do we have the resources to build the system? Will the system be acceptable? Will people use it?
* Economic feasibility, technical feasibility, schedule feasibility and operational feasibility- are the benefits greater than the costs?

**2.3.1. TECHNICAL FEASIBILITY**

The system must be evaluated from the technical point of view first. The assessment  
of this feasibility must be based on an outline design of the system requirement in the  
terms of input, output, programs and procedures. Having identified an outline system, the  
investigation must go on to suggest the type of equipment, required method developing  
the system, of running the system once it has been designed.  
Technical issues raised during the investigation are:

* Does the existing technology sufficient for the suggested one?
* Can the system expand if developed?

The project should be developed such that the necessary functions and performance  
are achieved within the constraints. The project is developed within latest technology.  
Through the technology may become obsolete after some period of time, due to the fact  
that never version of same software supports older versions, the system may still be used.

**2.3.2. OPERATIONAL FEASIBILITY**

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantages of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements identified in the requirements analysis phase of system development.

**2.3.3. ECONOMIC FEASIBILITY**

The developing system must be justified by cost and benefit. Criteria to ensure that  
effort is concentrated on project, which will give best, return at the earliest. One of the  
factors, which affect the development of a new system, is the cost it would require.  
The following are some of the important financial questions asked during preliminary  
investigation:

* The costs conduct a full system investigation.
* The cost of the hardware and software.
* The benefits in the form of reduced costs or fewer costly errors

Since the system is developed as part of project work, there is no manual cost to spend  
for the proposed system. Also all the resources are already available, it give an indication  
of the system is economically possible for development.

**2.3.4. LEGAL FEASIBILITY**

Legal feasibility is done to ensure that no violation of rules or liability that could result from development of this system.

**3. REQUIREMENT ANALYSIS**

**3.1 HARDWARE SPECIFICATION**

System : IBM-Compatible PC

Processor : Pentium IV

Speed : 2.0 GHZ

Memory : 512 MB RAM

Hard Disk Drive : 20 GB

**3.2. SOFTWARE SPECIFICATION**

Operating System : Windows XP and above

Language used : JAVA

Tools Used : NetBeans IDE 7.1.2, Eclipse.

**3.3. SELECTION OF SOFTWARE**

Java is used for the development of the software.

**JAVA Programming language:**

Java is a programming language originally developed by Sun Microsystems and released in 1995 as a component of Sun Microsystems Platform. The language derives much of its syntax from C and C++, but has a simpler object model and fewer low level facilities. Java applications are typically compiled to byte code that can run on any Java Virtual Machine (JVM) regardless of computer architecture.

The most important characteristics of Java are that it was design from the outset to be machine independent. Java programs can run unchanged on any operating system that supports Java .An application written in Java will only require a single set of source code regardless of the number of different computer platforms on which it is run-In any other programming languages the application will frequently required the source code to be tailored to accommodate different computer environments particularly if there is an extensive GUI involved .Java offers substantial savings in time and resources in developing supporting and maintaining major applications on several different hardware platforms and operating system. The next most important characteristics of Java is that it is object oriented. Object oriented programs are easier to understand and less time consuming to maintain and extend than programs that have been returned without the benefit of using object. Java is currently one of the most popular programming languages in use, and is widely used from application software to web applications

The Java programming language is a high level language that can be characterized by all the following:

* Simple
* Object-Oriented
* Distributed
* Multi-threaded
* Dynamic
* Portable
* Architecture Neutral

In Java programming language all source code is written in plain text files ending with Java extension. Those source files then compiled into .class files by the Javac compiler. A. class file does not contain code that is native to your processor: it instead contains byte codes– the machine language of Java Virtual Machine. The Java launched tool then runs on your application with an instance of JVM. Because the JVM is available on many different OS.

**3.4.SOFTWARE REQUIREMENT SPECIFICATION**

**3.4.1. GOAL OF IMPLEMENTATION:**

The primary goal of the system analyst is to improve the efficiency of the existing system. For that the study of specification of the requirements is very essential. For the development of the new system, a preliminary survey of the existing system will be conducted. Investigation done whether the upgradation of the system into an application program could solve the problems and eradicate the inefficiency of the existing system.

**3.4.2. FUNCTIONAL REQUIREMENTS:**

* The system should allow the user to enter the mnemonic program in a specified text area.
* The user should be able to edit the program if necessary.
* The user should be able to translate the mnemonic program entered into the equivalent hex code simply by clicking a button.
* The user should be able to translate the code that is entered till then without necessarily entering the complete code.
* If there are any errors present in the program regarding the instruction syntax/operand number/operand validity/operand size/label validity etc. they should be displayed in details with the line numbers on the console.
* The user should be able to correct/edit the program even after the errors are detected.
* The user should be able view the hex codes at the corresponding memory locations in the memory table after the successful assembling of program.
* He should be able to enter the input directly at the specified memory location in the memory table/IO table.
* The user should be able to run the program simply by clicking the RUN button.
* The output should be available at the specified memory location in the memory/io table.
* The user should be able to view the values of all registers after the running of instructions.
* The user should be able to save a program and he should be able to reuse them later in the future.
* The user should be able to search and view the details of the instructions of 8085 using the help and search features.

**4. SYSTEM DESIGN**

**4.1.INTRODUCTION**

System design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. Design is the first phase in development phase for any engineer’s product system. Design is the creative process. It deals with the creating ability of the programmer. A good design is the key to effective system. The term “Design” is defined as “The process of applying various techniques and principles for the purpose of defining a process or a system in sufficient details to permit its physical realization”.

**4.2.DESIGN METHODOLOGY**

The central administration system will be designed as a java application to be run primarily on windows.

**4.3.INPUT DESIGN**

The user interface design is very important for any application. The interfacedesign describes how the software communicated within itself, to system that interpreted with it and with humans who use it. The interface is a packing for computer software if the interface is easy to learn, simple to use. If the interface design is very good, the user will fall into an interactive software application.

The main input to the system is the mnemonic program and that can be entered in the

specified text area. And the other input values for the running phase can be inserted directly at the memory location in the memory table provided in the Software.

**4.4.OUTPUT DESIGN**

The system output is the most important and direct source of information to the user. So intelligible output design improves the relationship with the user and helps in decision-making. Outputs from the computer systems are required primarily to communicate the results of processing to users. They are also used to provide a permanent copy of these results for later consultation.

Outputs will be available at the memory locations specified in the program in the memory table. And the register values are also available at the corresponding field for each registers.

**4.5.MODULAR DESIGN**

A software system is always divided into several sub systems that makes it easier for the development. A software system that is structured into several subsystems makes it easy for the development and testing. The different subsystems are known as the modules and the process of dividing an entire system into subsystems is known as modularization or decomposition.

There are 3 modules in this project.

**4.5.1.TRANSLATION MODULE**

This module performs the translation function. It translates the mnemonic program into equivalent hex code. The main job of assembler is reading the mnemonic program identifying the instructions in it cross checking, comparing and matching the instructions with the previously stored 8085 instructions. Once a correct match is found the hex code associated with that instruction will be written into the hex program. Also it identifies the operands and memory locations in the mnemonic program and insert them into the hex program. The assembling process is done in two passes. Pass 1 and Pass 2. This is for the memory allocation purposes. By the end of the second pass all the mnemonic program will be translated into the hex code and the hex codes will be placed in the corresponding memory locations.

**4.5.2 RUN MODULE**

The job of compiler is to run the hex code obtained after the translation. There is a sequencing method used for this. The first hex code will be that of an instruction. Using the properties of the hex code of each instruction like instruction size, number of operands, position of immediate operands etc. we determine the following operands. And with this sequencing and according to the instructions the compiler will perform various operations and produces the results. The registers and flags also will be updated according to various operations.

**4.5.3.DEBUGGER MODULE**

This module helps in finding out various programming errors in the user input program. Various error routines are used for each type of errors and these error routines will be called from various locations in the program like pass 1 pass 2 or even from the compiling phase. The error details as well as the line numbers will be output to the console screen.

**4.5. DATA FLOW DIAGRAMS (DFD)**

**LEGENDS USED FOR DFD**

Data Flow

Data Store

Process

Source / Destination

**LEVEL 0 DFD**

files

mnemonics

USER

files

output

FILES DATABASE

**LEVEL 1 DFD**

code

USER

file name

file name

FILES DATABASE

file

errors

hex code

results

output

**LEVEL 2 DFD**

**2.1.Assembling phase (conversion to hex code)**

mnemonics

Opcodes

Negative flag

Positive flag flag

operands

Hex program

**2.2.Compiling phase (Running program)**

Hex code

opcodes

results

**5.CODING**

**5.1.PSEUDO CODE**

**5.1.1.READING PROGRAM**

1. Read the mnemonic program inputted by user
2. Split each line of program
3. Store each line of code in an array called ‘iparray’

**5.1.2.CONVERTING INTO HEX CODE**

This is done in two passes:

PASS 1 – Pass 1 creates an array of hexcodes of all opcodes and operands.

START

1. Declare an array ‘hex’ to store hexcodes of all opcodes & operands.
2. Initialize ‘hex’ array and all other variables.
3. For each remaining instruction in ‘iparray’ do,
4. Compare the opcode part of the instruction with the instructions stored in the link list ‘instructions’.
5. If it matches do the following else go to 11.
6. Insert the hexcode for the opcode of the matched instruction into ‘hex’.
7. If there are no operands OR there are only register operands, go to 4 else
8. If there are immediate operands do 9 else go to 10.
9. Check the operand size.

9.1.Call error routine to check if the operand is valid.

9.2.If operand is one byte, then insert it to the next position in ‘hex’.

9.3. If operand is two byte, then split the operand one byte each and insert them into the next consecutive positions in ‘hex’.

9.4. go to 4.

10. If the instruction is either a call or jump instruction do 10.1 else go to 11.

10.1.Put the operand (here it will be a label) into ‘hex’ array.

10.2. Leave the next position in the hex array free to add the address later.

10.3. go to 4.

11.If it is a label definition, then do

11.1. Search the label name in the symbol table.

11.2. If the label is already defined then report error “Multiple declaration”

Otherwise,

11.3. Insert the label and its address into symbol table for future reference.

11.4. go to 4.

END OF PASS 1

PASS 2 – pass 2 replaces all the labels in the hex array with the address in the symbol

table

START

1. For all labels in the hex array do the following.
2. Refer the symbol table and find the label.
3. If found do 4 else go to 5.
4. Substitute the address instead of the label in the hex array.
5. Report error “Undefined Symbol”.

END OF PASS 2

**5.1.3.RUN MODULE**

To run the program after mnemonics have been translated to hex

START

1. The first hex code in the array ‘hex’ represents an opcode.
2. Compare the opcode with all 8085 instruction opcode’s hex values.
3. When a match is found do 4 and 5.
4. Use the operand’s size, numbers, immediate position etc to find out which of the following hexcodes are the operands.
5. Once the operands are determined perform the operations specified by the instruction.
6. Take the next operand hexcode.
7. Go to 2.

END

**5.1.4.ERROR ROUTINE**

To check whether the passed hex value is valid or not

START

1. Check whether the size of the hexcode and the expected values are equal
2. If they are not equal report error “Operand size doesn’t match” otherwise
3. If any one of character in the hexcode is other than 0-9 or A-F then
4. Report error “invalid value/operand”

END

**6. TESTING**

**6.1.TYPES OF TESTING**

The aim of the testing process is to identify all the defects existing in a software product. Testing a program consists of subjecting the program to a set of test input and observing if the program behaves as expected .A test is vital to the success of the system. System test makes a logical assumption that if all parts of the system are correct, then goal will be successfully achieved. A series of tests are performed before the system is ready for user acceptance testing.

**6.1.1.UNIT TESTING**

Unit testing focuses verification effort on the smallest unit of software designs. To check whether each module in the software works properly so that it gives desired output to the given input. All validation and condition are tested in the module level in the unit test.

Here, the translation module, the run module and the debugger module are checked individually. The translation module is checked if the instructions are properly and accurately translated into their corresponding hex codes. The run module is checked for the proper execution of the instruction sequence and evaluate the results. The obtained results have been found to be equal to the expected results. The debugger module is tested by checking whether the reported errors and their location is valid or not.

**6.1.2.INTEGRATION TESTING**

Integration testing is the complete testing of the set of modules, which makes up the product. One approach is to wait all the units have passed the testing, and then combine them and then tested. Another strategy to test the product is incrementing addition of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination and so on. The advantage of this approach is that interface dispenses can be easily found and corrected.

**6.1.3.SYSTEM TESTING**

The 8085 IDE has been tested as a whole in this phase of testing. Here, the overall functioning of the program is tested as a whole.

A number of different mnemonic programs are entered into the text box of the IDE and the assemble button is clicked. It is checked if the hex code in the memory table and the opcode text box is same as the corresponding mnemonic instructions. If there are any errors in the mnemonic program, it is checked if they are reported. Then the run button is clicked. The results(eg-register values, values at memory locations) are checked if they are correct. It has been found that the program has functioned without exhibiting any errors.

**6.2. SAMPLE TEST CASES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.NO | FUNCTION | TEST CASE | INPUT | EXPECTED OUTPUT | REMARKS |
| 1 | Translation | Check if all the mnemonic codes are correctly translated. | Valid mnemonic program | Hex codes corresponding to the mnemonic instructions. | The module checks each instruction against the previously stored instructions and when a match is found, the corresponding hex code is selected. |
| 2 | Debugger | Check whether error reports are accurately reported. | Invalid instructions. | Nature and location of error. | The module reports the type and location of error if the instruction does not fit the description of a valid instruction. |
| 3 | Run module | Check whether all instructions are executed properly. | Valid mnemonic program | The results of the corresponding instructions in the correct sequence. | The module executes every instruction and display its results in the correct sequence. |

**7. CONCLUSION**

**7.1.CONCLUSION**

The 8085 IDE has be developed with the intention to provide its users with the best possible 8085 programming experience. It has been taken care to make the IDE’s interface easy to understand and user-friendly. The GUI has been made as simple as possible and at the same time, providing the users with helpful tools like the search feature and the auto-completion. The users can easily load the program from the memory or enter the program easily. The debugging function helps to reduce the errors and improves the accuracy.

The IDE can help students to practice 8085 programming at their homes. This proves to be a good alternative to the 8085 trainer kits, which are expensive and are limited in number. The students can use this IDE at their convenience, instead of being able to practice only during the college hours. The ease of use , the built-in features and the portability factor definitely makes this product very useful for students.

**BIBLIOGRABHY**

1.Java™: The Complete Reference- Seventh Edition, Herbert Schildt.

2.Java Swing, 2nd Edition. By Brian Cole, Robert Eckstein, James Elliott, Marc Loy,

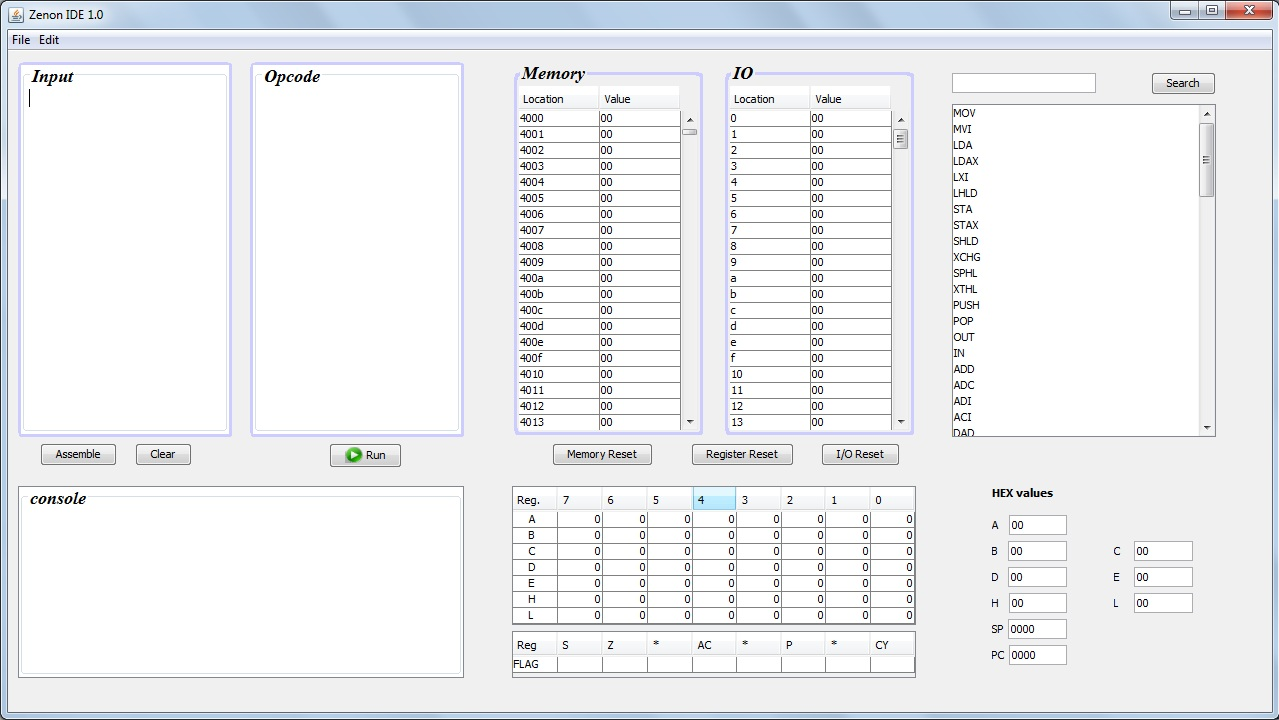
David Wood. Publisher : O'Reilly.

3.Microprocessor Architecture, Programming, and Applications with the 8085.

4. Oracle java tutorials- <http://docs.oracle.com/javase/tutorial/>

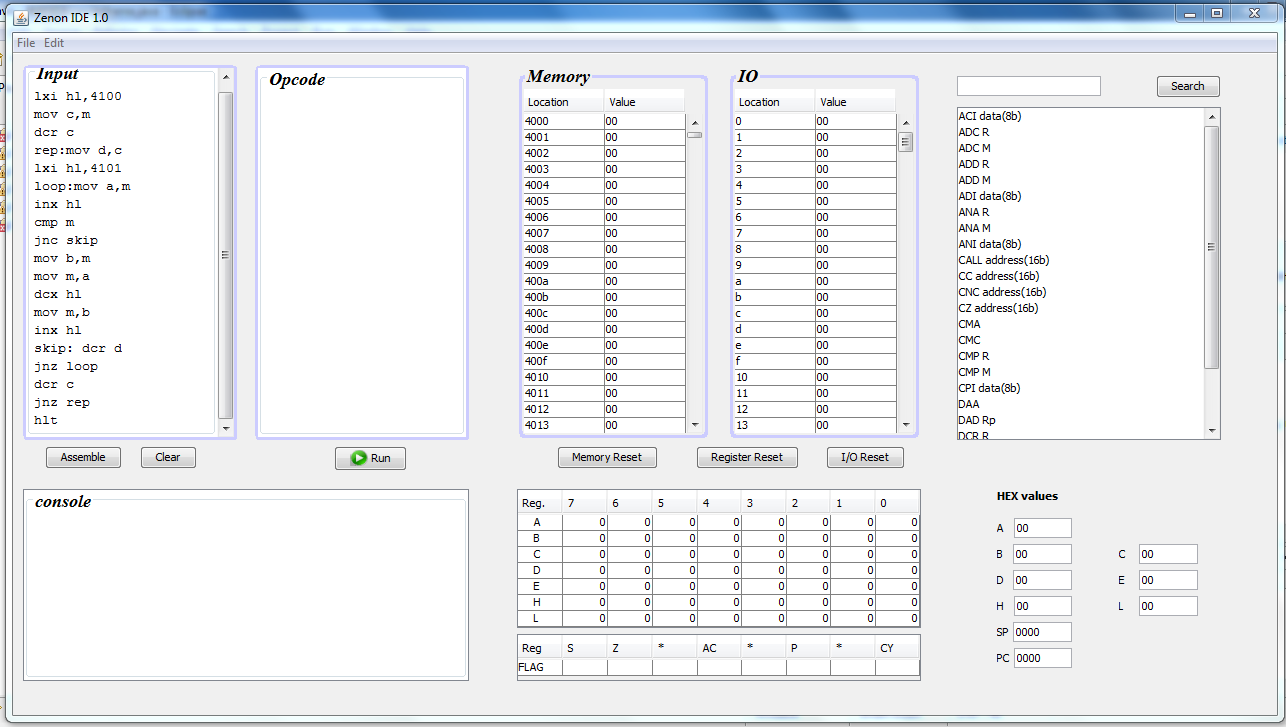
**APPENDIX - A**

**SCREENSHOTS**

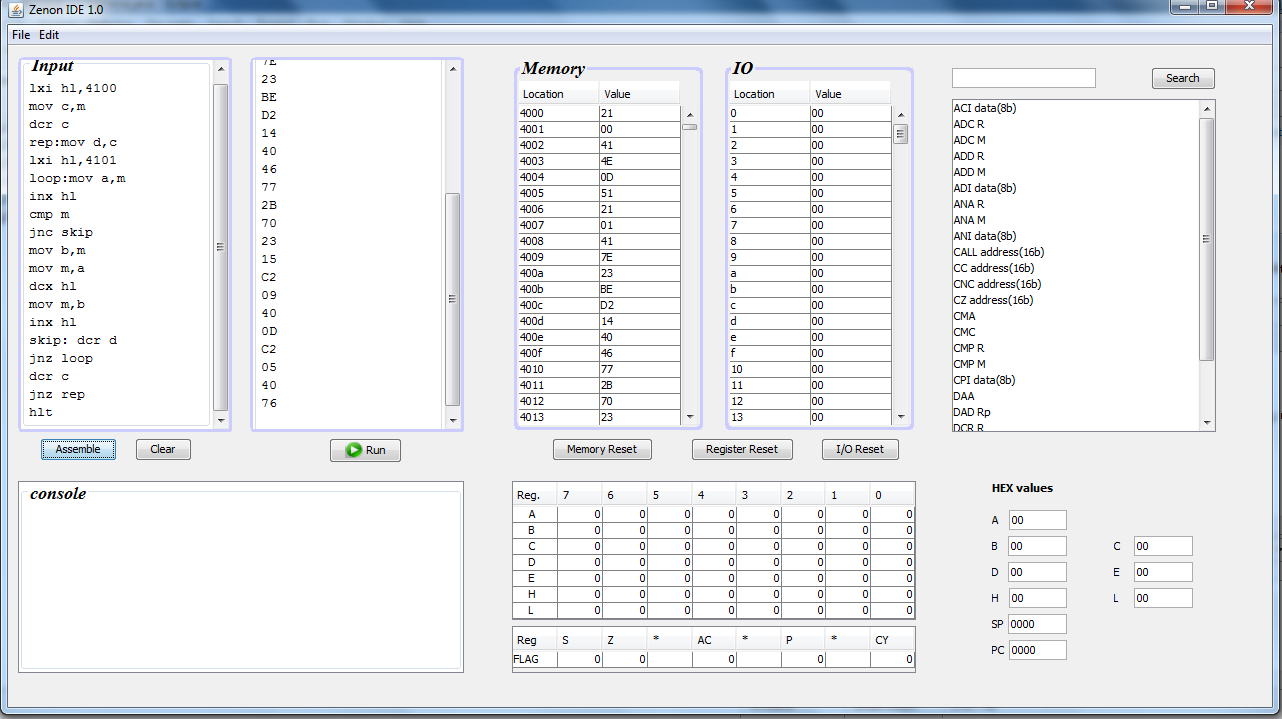
****

***8085 IDE***

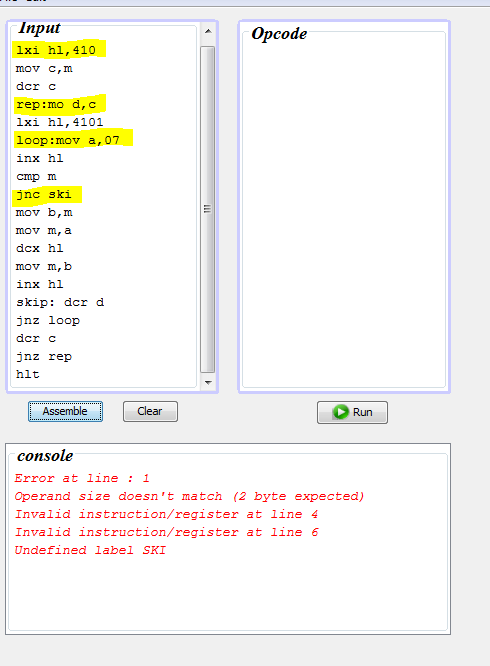
** *Entering the program***

****

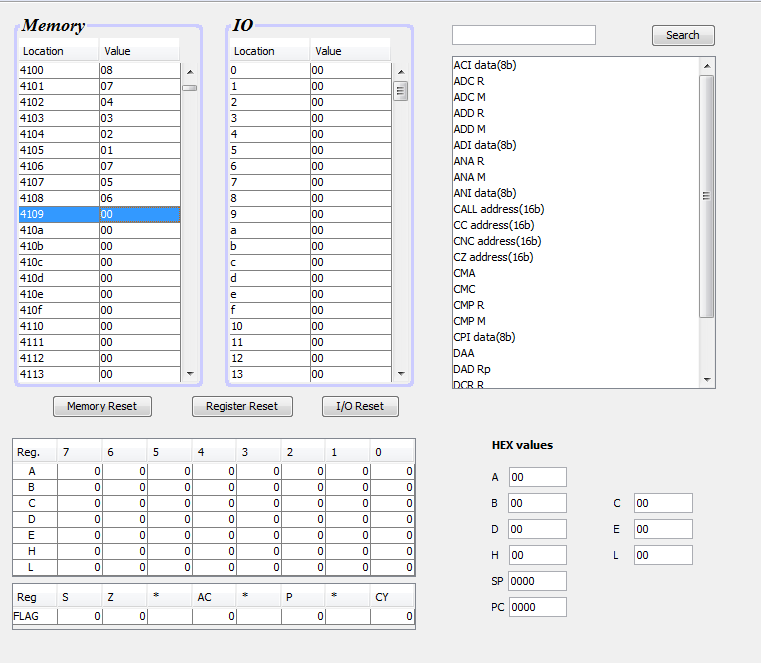
***Program***

****

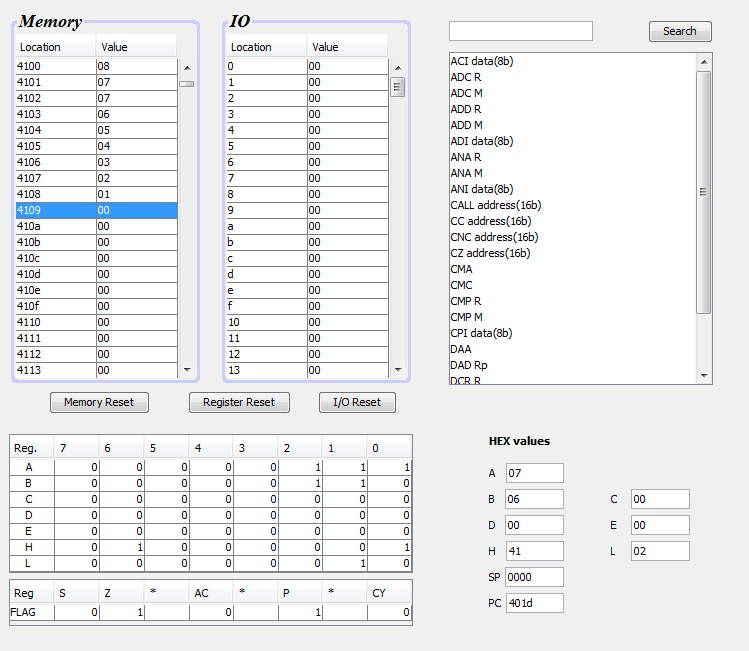
***After assembling***

****

***Error report***

****

***Input***

****

***Output***

**APPENDIX – B**

**SAMPLE CODE (TRANSLATION)**

**public** **void** toHex()

{**int** mem=0,cp=0;

**boolean** printop=**true**,match=**false**;

//reset hex

**for**(**int** hexi=0;hexi<4000;hexi++)hex[hexi]="00";

consoleTextArea.setText("");

String sub;

opcode.setText("");

symtab.first=**null**;

**for**(**int** xx=0;xx<iparraylen;xx++)//pass 1

{

link cur=instruction.first;

match=**false**;

**while**(cur!=**null**)//instr cmpr

{

**if**(iparray[xx].equalsIgnoreCase(cur.name)&&cur.ZEROOPF)

{ match=**true**;

hex[mem]=cur.hexcode;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

}

**else** **if**(iparray[xx].equalsIgnoreCase(cur.name)&&cur.REGF)

{ match=**true**;

hex[mem]=cur.hexcode;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

}

**else** **if**(iparray[xx].startsWith(cur.name)&&cur.IMDTF)

{ match=**true**;

hex[mem]=cur.hexcode;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

**if**(cur.impos==2)

{

cp=iparray[xx].indexOf(',');

cp++;

sub=iparray[xx].substring(cp);

**if**(cur.oplength==2)

{ **if**(errorimd(2,sub,xx)){printop=**false**;}

hex[mem]=sub;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

}

**else** **if**(cur.oplength==4)

{

String sub2=sub.substring(0,2);

String sub1=sub.substring(2);

**if**(errorimd(4,sub,xx)) {printop=**false**;}

hex[mem]=sub1;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

hex[mem]=sub2;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

}

}

**else** **if**(cur.impos==1)

{

cp=iparray[xx].indexOf(" ");

cp++;

sub=iparray[xx].substring(cp);

**if**(cur.oplength==2)

{

hex[mem]=sub;

**if**(errorimd(2,sub,xx)) {printop=**false**;}

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

}

**else** **if**(cur.oplength==4)

{

String sub2=sub.substring(0,2);

String sub1=sub.substring(2);

**if**(errorimd(4,sub,xx)){printop=**false**;}

hex[mem]=sub1;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

hex[mem]=sub2;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

}

}

}

**else** **if**(iparray[xx].startsWith(cur.name)&&(cur.JUMPF||cur.CALLF))

{ match=**true**;

hex[mem]=cur.hexcode;

memorytable.getModel().setValueAt(hex[mem],mem,1);

mem++;

//code for inserting the label(L) into the hex codes

hex[mem]=iparray[xx].substring(iparray[xx].indexOf(" ")+1).trim();

labeljmp.add(mem);

mem++;

mem++;

}

**else** **if**(iparray[xx].indexOf(":")!=-1)

{ match=**true**;

String lab;

**boolean** dup;

lab=iparray[xx].substring(0,iparray[xx].indexOf(":")).trim();

**if**(symtab.search(lab))

{

symtab.add(lab, mem);

}

//multiple defintion

**else**{

consoleTextArea.append("Multiple Definition for Label "+lab+" at Line "+(xx+1));

printop=**false**;

}

String temp=iparray[xx].substring(iparray[xx].indexOf(":")+1).trim();

iparray[xx]=temp;

xx--;

**break**;

}

cur=cur.next;

}

**if**(!match)

{ **if**(!iparray[xx].equals(""))

{

printop=**false**;

consoleTextArea.append("Invalid instruction/register at line "+(xx+1)+"\n");

}

}

}

labellink curlab=symtab.first;

**int** testflag=0;

**for**(**int** p2: labeljmp)

{

testflag=0;

curlab=symtab.first;

**while**(curlab!=**null**)

{

**if**(hex[p2].equalsIgnoreCase(curlab.label))

{

testflag=1;

String jadr=Integer.*toHexString*(curlab.taddr+16384);

hex[p2]=jadr.substring(2);

memorytable.getModel().setValueAt(hex[p2],p2,1);

p2++;

hex[p2]=jadr.substring(0,2);

memorytable.getModel().setValueAt(hex[p2],p2,1);

**break**;

}

curlab=curlab.next;

}

**if**(testflag==0) {printop=**false**; consoleTextArea.append("Undefined label "+hex[p2]);}

}

**for**(**int** txi=0;txi<mem;txi++)

opcode.append(hex[txi]+"\n");

**if**(!printop) {opcode.setText(""); resetmem();

resetio();

Toolkit.*getDefaultToolkit*().beep();

}

}