**A Java Based Application Framework for**

**Conversion of**

**BASIC Program into 8085 Assembly**

**Program**

**(BASICto8085)**

*A report submitted in the partial fulfillment of*

*the requirement for the degree of*

Bachelor of Technology

*In*

**Computer Science and Engineering**

*By*

**Raunak Kumar**

**Parimal Kumar**

*Under the supervision of*

**Prof. Sourav Saha**

Assistant Professor

Department of Computer Science & Engineering

Institute of Engineering & Management



Department of Computer Science & Engineering

Institute of Engineering & Management

Gurukul,EP Block, Sector-V, Salt Lake Electronics Complex

Kolkata-700091, West Bengal, India

**June, 2016**

**CERTIFICATE**

This is to certify that the project report entitled “**A** **Java Based Application Framework for Conversion of BASIC Program into 8085 Assembly Program**”, submitted by **Raunak Kumar (*Registration No. 121040410073 of 2012-2013)*** and **Parimal Kumar(*Registration No. 121040410063 of 2012-2013)*** students of **INSTITUTE OF ENGINEERING & MANAGEMENT,** in fulfillment of requirement for the degree of **Bachelor of Computer Science & Engineering** is a bona fide work carried out by them under the supervision and guidance of **Prof. Sourav Saha** during the academic session of 2012-2016. The content of this report has not been submitted to any other University or Institute for the award of any other degree.

We are glad to inform that the work is entirely original and the performance is found to be satisfactory.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Sourav Saha Prof. Dr. Debika Bhattacharya

Project Guide H.O.D

Dept. of Computer Science Engineering Dept. of Computer Science Engineering

Institute of Engineering & Management Institute of Engineering &Management

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr. A. K. Nayak

Principal

Institute of Engineering & Management

Sector- V, Saltlake Electronics Complex, Kolkata-700091

**ABSTRACT**

**BASIC** (Beginner's All-purpose Symbolic Instruction Code) is a general-purpose user friendly high-level **programming** language which is very easy to use. BASIC was designed specifically for less technical users who did not have or want the mathematical background previously expected. It is the most popular programming language among general purpose high level languages. BASIC remains popular in many dialects and in new languages influenced by BASIC, such as Microsoft's Visual Basic.

The **Intel 8085** is an 8-bit microprocessor produced by Intel and introduced in 1976.It is software-binary compatible with the more-famous Intel 8080 with only two minor instructions added to support its added interrupt and serial input/output features. However, it requires less support circuitry, allowing simpler and less expensive microcomputer systems to be built.

BASIC programmiming language is a very friendly high level language which has very basic syntax which resemble English language. We need an operating system to compile and run BASIC program. On the contrary, 8085 is a low level language not easily understandable by users and it needs 8085 microcontroller to operate. But it is has an inherent advantage that it is faster than high level languages. So , we needed some framework which can give us 8085 assembly level programs on entering BASIC program. Hence, we designed BASICto8085 which converts BASIC program to 8085 Assembly Program.



**ACKNOWLEDGEMENTS**

We would like to take this opportunity to thank everyone whose cooperation and encouragement throughout the ongoing course of this project remains invaluable to us.

We are sincerely grateful to our guide and mentor **Prof. Sourav Saha** of the Department of Computer Science & Engineering, IEM Kolkata, for his wisdom, guidance and inspiration that helped us go through with this project and take it to where it stands now.

We would also like to express our sincere gratitude to **Prof. Satyajit Chakraborty**, Director**, Prof. Dr. Amlan Kusum Nayak**, Principal and **Prof. Dr. Debika Bhattacharyya**, HOD of Computer Science & Engineering and other faculties of Institute of Engineering & Management, for their assistance and encouragement.

Last but not the least, we would like to extend our warm regards to our families and peers who have kept supporting us and always had faith in our work.

**TABLE OF CONTENTS**

**CHAPTER 1- INTRODUCTION 4-5**

1.1 Motivation 4

1.2 Objectives 5

**CHAPTER 2-BACKGROUND 6-9**

2.1 Netbeans 6

2.2 Java SE 7

2.3 Swing 8

**CHAPTER 3 – REQUIREMENT GATHERING & ANALYSIS 10-14**

3.1 Requirement Gathering 10

3.1.1 Use Case Diagram 10

3.2 Requirement Analysis 11

**CHAPTER 4- DESIGN 15-28**

4.1 Class Diagram 15

4.2 Sequence Diagram 16

4.3 Control Flow Diagram 17

**CHAPTER 5 – IMPLEMENTATION 29-40**

**CHAPTER 6- TESTING 41-43**

**CHAPTER 7- SCREENSHOTS 44-45**

**CHAPTER 8- CONCLUSION 46**

**CHAPTER 9-FUTURE SCOPE 47**

**REFRENCES 48**

**Appendix 49-60**

**LIST OF FIGURES**

**Fig. 3.1 Use Case Diagram 10**

**Fig. 4.1 Class Diagram 15**

**Fig. 4.2 Sequence Diagram 16**

**Fig. 4.3 Runtime Flowchart 18**

**Fig. 4.4 Block Flowchart 19**

**Fig. 4.5 Statement Flowchart 20**

**Fig. 4.6 Expression Flowchart 21**

**Fig. 4.7 IF Flowchart 24**

**Fig. 4.8 While Flowchart 27**

**Fig. 4.9 Array Flowchart 28**

**Fig. 7.1 First Page 44**

**Fig. 7.2 Second Page 45**

**1. INTRODUCTION**

* 1. **Motivation**

The **motivation** for doing this **project** was primarily an interest in undertaking a challenging **project**  in an interesting area. So, we have chosen to convert BASIC programming language into 8085 Assembly Level Program, which according to our knowledge and belief, has not been done yet.

BASIC programmiming language is a very friendly high level language which has very basic syntax which resemble English language. We need an operating system to compile and run BASIC program. On the contrary, 8085 is a low level language not easily understandable by users and it needs 8085 microcontroller to operate. But it is has an inherent advantage that it is faster than high level languages. So , we needed some framework which can give us 8085 assembly level programs on entering BASIC program. Hence, we designed BASICto8085 which converts BASIC program to 8085 Assembly Program.

We derive our motivation from approximately 5 lakhs students who take admission in Computer Science and Information Technology(IT) Engineering in India each year. Most universities teach 8085 as an assembly level programming. So, we have at least 4 lakhs plus potential users each year. Few embedded systems like firmware for telephones, ignition system, AC control system and sensors can use this, too.

* 1. **Objectives**

BASIC Language is a very precise, simple and easy to understand programming language. 8085 is a very low level language which is not very intuitive. Our java application will convert Basic Programs to 8085 programs which run on 8085 microprocessor.

We try to implement Basic blocks required for the Conversion. Basic blocks like statement, Expression, If, While, For, Array, etc. will be implemented in java for converting Basic Program into 8085 program. Then we do identification of Basic Structure and identification of nesting structure present in assembly code.

**2. BACKGROUND**

**2.1 NetBeans**

NetBeans is a software development platform written in Java. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. Applications based on the NetBeans Platform, including the NetBeans integrated development environment (IDE), can be extended by third party developers.

The NetBeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++and HTML5.

NetBeans is cross-platform and runs on Microsoft Windows, Mac OS X, Linux, Solaris and other platforms supporting a compatible JVM.

The NetBeans Team actively support the product and seek feature suggestions from the wider community. Every release is preceded by a time for Community testing and feedback.

**NetBeans Platform**

The NetBeans Platform is a [framework](https://en.wikipedia.org/wiki/Software_framework) for simplifying the development of [Java Swing](https://en.wikipedia.org/wiki/Java_Swing) desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing NetBeans plugins and NetBeans Platform based applications; no additional SDK is required.

Applications can install modules dynamically. Any application can include the Update Center module to allow users of the application to download [digitally signed](https://en.wikipedia.org/wiki/Digital_signature) upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again.

The platform offers reusable services common to desktop applications, allowing developers to focus on the logic specific to their application. Among the features of the platform are:

* User interface management (e.g. menus and toolbars)
* User settings management
* Storage management (saving and loading any kind of data)
* Window management
* Wizard framework (supports step-by-step dialogs)
* NetBeans Visual Library
* Integrated development tools

**NetBeans IDE**

NetBeans IDE is an open-source integrated development environment. NetBeans IDE supports development of all Java application types (Java SE (including JavaFX), Java ME, web, EJB and mobile applications) out of the box. Among other features are an Ant-based project system, Maven support, refactorings, version control  (supporting  CVS, Subversion,  Git,  Mercurial  and  Clearcase ).

Modularity: All the functions of the IDE are provided by modules. Each module provides a well-defined function, such as support for the Java language, editing, or support for the CVS versioning system, and SVN. NetBeans contains all the modules needed for Java development in a single download, allowing the user to start working immediately. Modules also allow NetBeans to be extended. New features, such as support for other programming languages, can be added by installing additional modules. For instance, Sun Studio, Sun Java Studio Enterprise, and Sun Java Studio Creator from Sun Microsystems are all based on the NetBeans IDE.

License: From July 2006 through 2007, NetBeans IDE was licensed under Sun's Common Development and Distribution License (CDDL), a license based on the Mozilla Public License (MPL). In October 2007, Sun announced that NetBeans would henceforth be offered under a dual license of the CDDL and the GPL version 2 licenses, with the GPL linking exception for GNU Classpath.

**2.2 Java SE**

Java part of the Java software-platform family. Java SE defines a wide range of general-purpose APIs – such as Java APIs for the Java Class Library – and also includes the Java Language Specification and the Java Virtual Machine Specification. One of the most well-known implementations of Java SE is Oracle Corporation's Java Development Kit (JDK).

Java SE was known as Java 2 Platform, Standard Edition or J2SE from version 1.2 until version 1.5. The "SE" is used to distinguish the base platform from the Enterprise Edition (Java EE) and Micro Edition (Java ME) platforms. The "2" was originally intended to emphasize the major changes introduced in version 1.2, but was removed in version 1.6. The naming convention has been changed several times over the Java version history. Starting with J2SE 1.4 (Merlin), Java SE has been developed under the Java Community Process, which produces descriptions of proposed and final specifications for the Java platform called Java Specification Requests (JSR). JSR 59 was the umbrella specification for J2SE 1.4 and JSR 176 specified J2SE 5.0 (Tiger). Java SE 6 (Mustang) was released under JSR 270.

**2.3 Swing**

Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

Swing is a platform-independent, Model-View-Controller GUI framework for Java, which follows a single-threaded programming model. Additionally, this framework provides a layer of abstraction between the code structure and graphic presentation of a Swing-based GUI.

Platform, Standard Edition or Java SE is a widely used platform for development and deployment of portable code for desktop and server environments. Java SE uses the object-oriented Java programming language.

### Foundations

Swing is platform-independent because it is completely written in Java. Complete documentation for all Swing classes can be found in the Java API Guide for Version 6 or the Java Platform Standard Edition 8 API Specification for Version 8.

#### Extensible

Swing is a highly modular-based architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations using Java's inheritance mechanism.

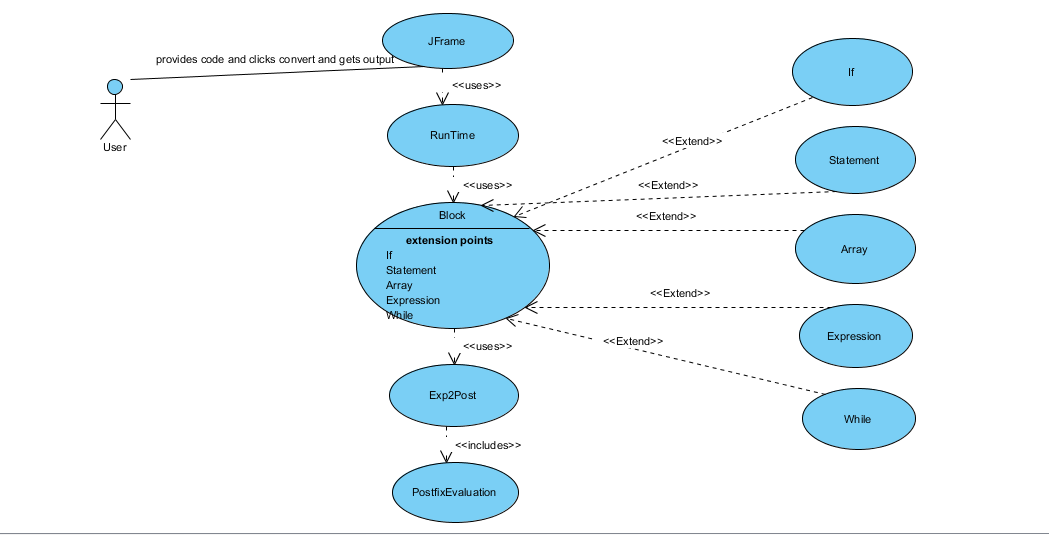
Swing is a component-based framework, whose components are all ultimately derived from the javax.swing. JComponent class. Swing objects asynchronously fire events, have bound properties, and respond to a documented set of methods specific to the component. Swing components are Java Beans components, compliant with the Java Beans Component Architecture specifications.

#### Configurable

Swing's heavy reliance on runtime mechanisms and indirect composition patterns allows it to respond at run time to fundamental changes in its settings. For example, a Swing-based application is capable of hot swapping its user-interface during runtime. Furthermore, users can provide their own look and feel implementation, which allows for uniform changes in the look and feel of existing Swing applications without any programmatic change to the application code.

**3. Requirement Gathering**

**3.1 Use Case Diagram**



**Fig. 3.1**

* 1. **REQUIREMENT ANALYSIS**

**FUNCTIONAL REQUIREMENT**

**3.2.1 JFrame**

**Description:** Actors can interact with the system with the help of the Jframe. This provides the actor with user interface, by virtue of which they can navigate/use the different functionalities of the system. The Jframe use case can be divided into two sub categories: One for input and other for output.

1. The first Jframe is for input code in Basic language written or browsed.
2. Then we can click on the convert button to convert the code into mnemonics.
3. Thus finally the system returns the converted mnemonic code to us in another Jframe window.
   * 1. **Runtime**

**Description:** Actors provide their code to system, which is then fed to this class. This class interprets each line and then creates object accordingly. Further it runs the run() method for each of the objects in order to generate the mnemonic code.

1. After user has provided the code and then clicked the run button.
2. Code is interpreted by the Runtime class
   1. For each of the line it figures out which type it is like If, While etc. And creates object for each of these.
3. Then for each of the object the run class is called to generate the mnemonic code.

.

1. **Block**

**Description:** Application creates object of different type of blocks according to the use. This helps them store information about the line which can be used in future for reference.

1. While Runtime breaks down the code it tries to create object for each line.
   1. The application tries to detect which type the line belongs to with the help of the keywords used.
   2. Thus that Blocks object is created and is added as sub block to some other block.
   3. We are also able to store extra information like variable name, values, addresses etc. with the block. .
   4. This information is used in future to generate the mnemonic code.
2. **Exp2Post**

**Description:** This is used by some of the child block classes to change a infix expression to postfix form. This uses a stack of operators and maintains operator precedence to get the postfix expression.

1. It scan the expression from left to right.
2. Whenever the scanned symbol is an open bracket push it on to the stack.
3. If the scanned symbol is an operand then push it on to the expression.
4. If the symbol is an operator push it on to the stack and go on copying the elements from the stack if and only if the scanned symbol has a lesser equal precedence then top of the stack.
5. Pop out the elements from the stack if you find the scanned symbol is a right parenthesis till you find the matching left parenthesis.
6. Thus final sent the postfix expression to PostfixEvaluation class for evaluation.
7. **PostfixEvaluation**

**Description:** This class tries to evaluate the postfix expression to get result after using the actual for the variables.

* + 1. Scan elements from right to left.
    2. If the elements are numbers or variables push them to the stack.
    3. Whenever there is a operator being scanned pop out the last two variables/numbers and then apply the operation get the result.
    4. Then this is again pushed to the stack.
    5. Repeat until whole expression is evaluated thus the last element in the stack is the output of the expression, return this value.

1. **If**

**Description:** This block is used to form object for If statement.

1. When you see the “IF” keyword then create object of If block object.
2. Also pass “IF” as its type, the condition statement along with the global label it is being assigned.
3. If keyword is “ENDIF” use “ENDIF” as its type and null as value and label.
4. **Statement**

**Description:** Used to create statement block.

**3.2.7.1** If there is a “LET” keyword try to create a statement block object.

**3.2.7.2** Also pass the variable name and the value it is assigned, it can even be an expression.

1. **Array**

**Description:** Used to create array block.

**3.2.8.1** If there is a “DIM” keyword try to create a array block object.

**3.2.8.2** Also pass the variable name, its size and the values it is assigned.

1. **Expression**

**Description:** Used to create expression block.

* + - 1. If there is a “DO” keyword try to create a expression block object.

**3.2.9.2** Also pass the variable name and the value it is assigned, it can even be an expression.

1. **While**

**Description:** This block is used to form object for While statement.

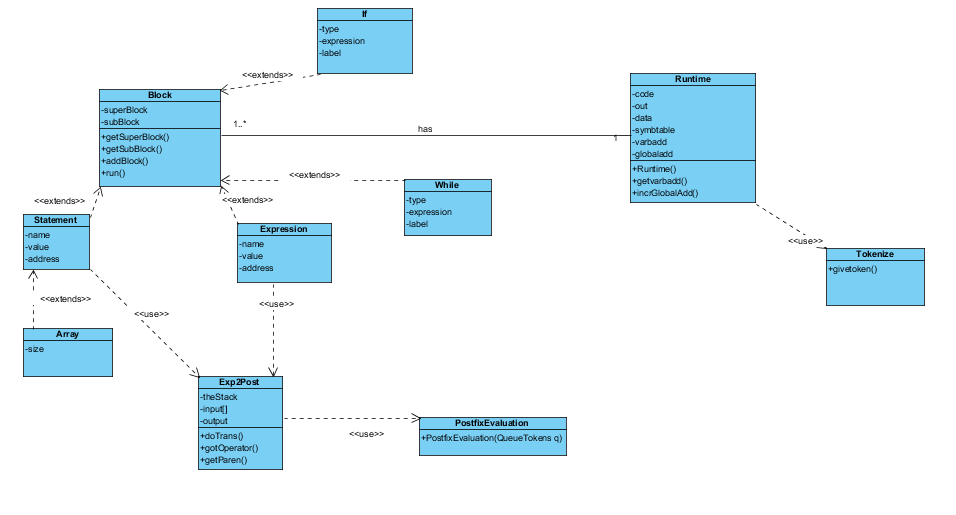
1. When you see the “WHILE” keyword then create object of While block object.
2. Also pass “WHILE” as its type, the condition statement along with the global label it is being assigned.
3. If keyword is “WEND” use “WEND” as its type and null as value and label.

**NON-FUNCTIONAL REQUIREMENT**

* **Performance Requirements**
* The system designed is expected to have a quick response time.
* **Safety Requirements**
* The system should be capable of recovering from earliercrashes and continue the process.

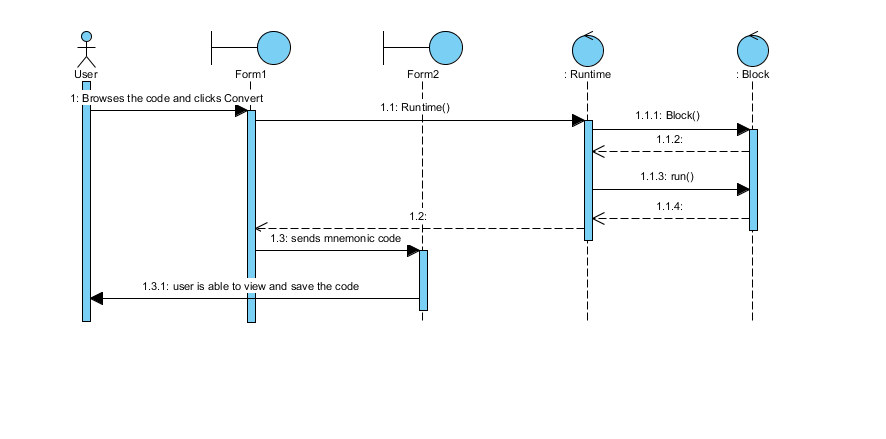
**4. DESIGN**

4.1 Class Diagram:



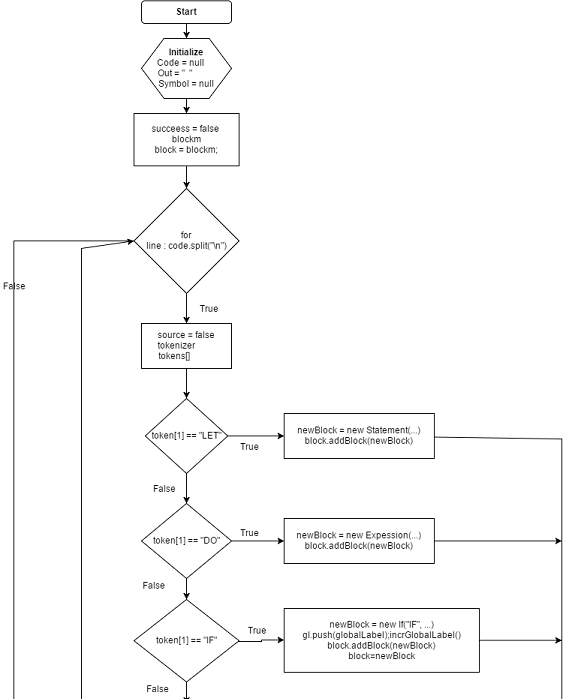
**Fig. 4.1**

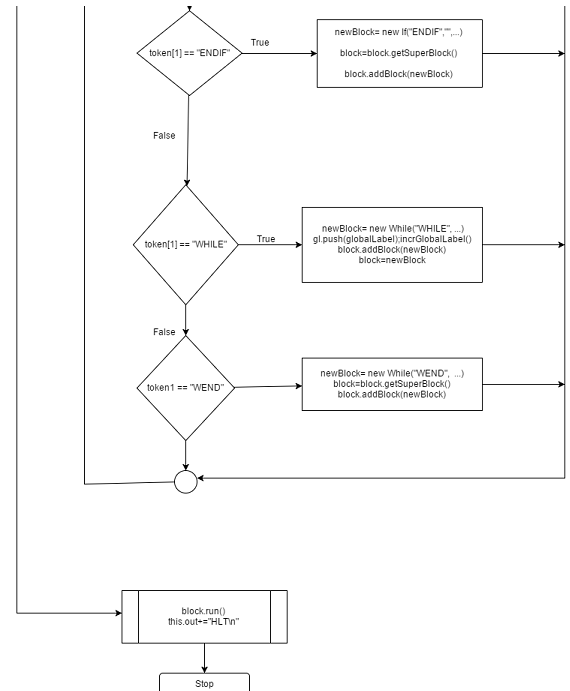
4.2 Sequence Diagram:



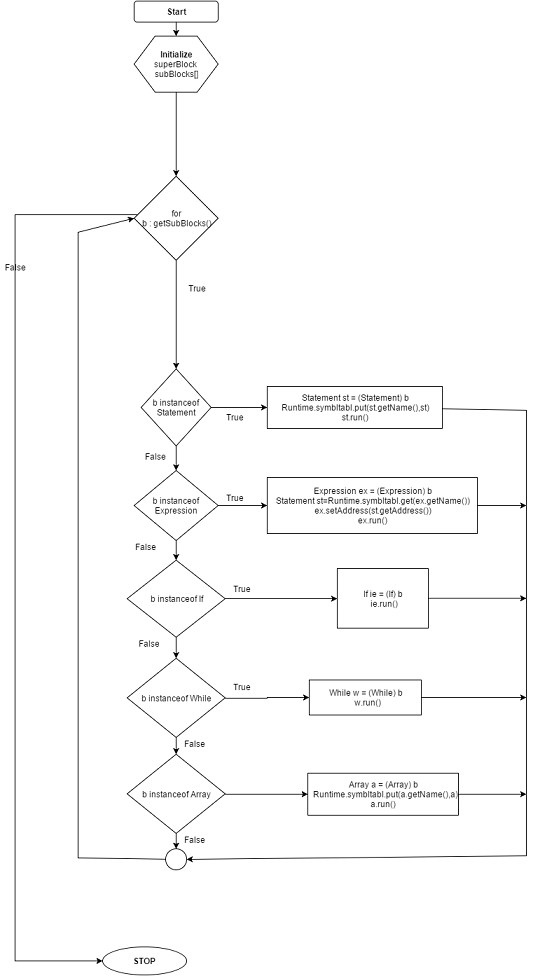
**Fig. 4.2**

* 1. **Control Flow Diagrams:**
     1. Runtime



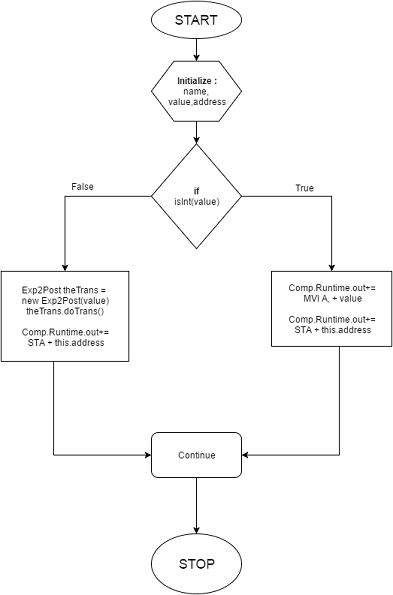


**Fig. 4.3**

4.3.2 BLOCK FLOWCHART

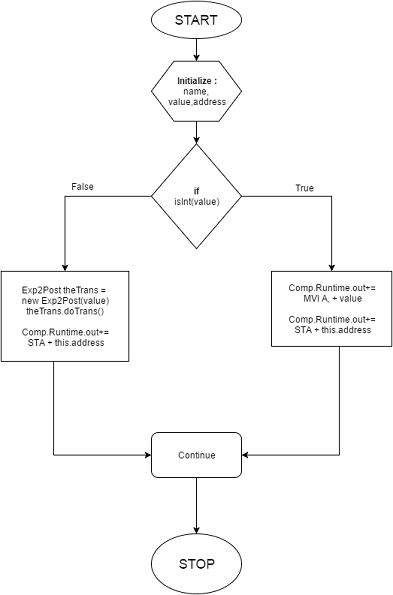
**Fig. 4.4**

4.3.4 STATEMENT FLOWCHART



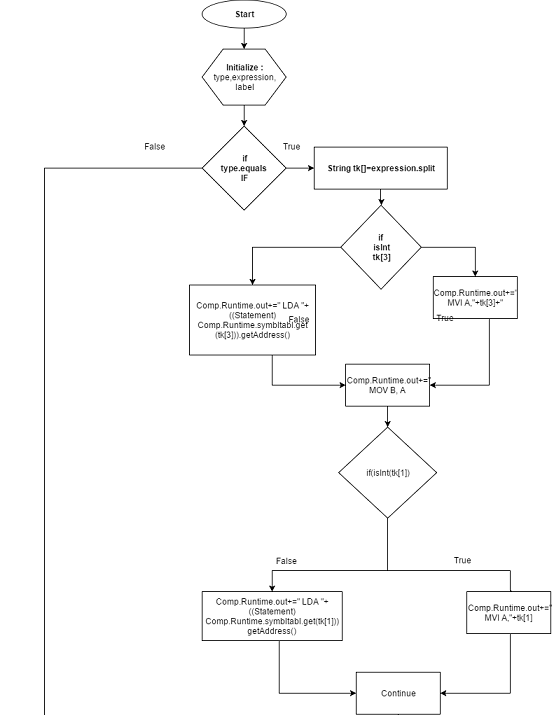
**Fig. 4.5**

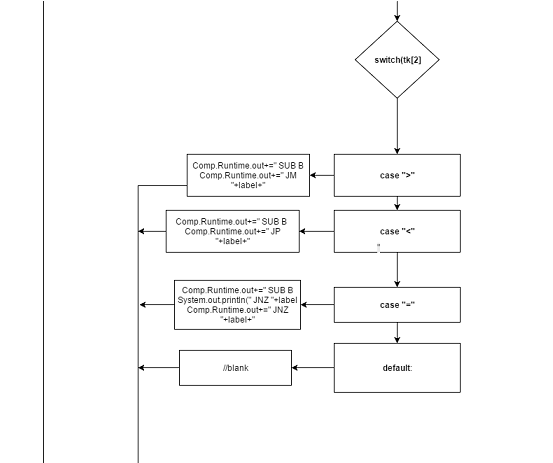
4.3.5 EXPRESSION FLOWCHART

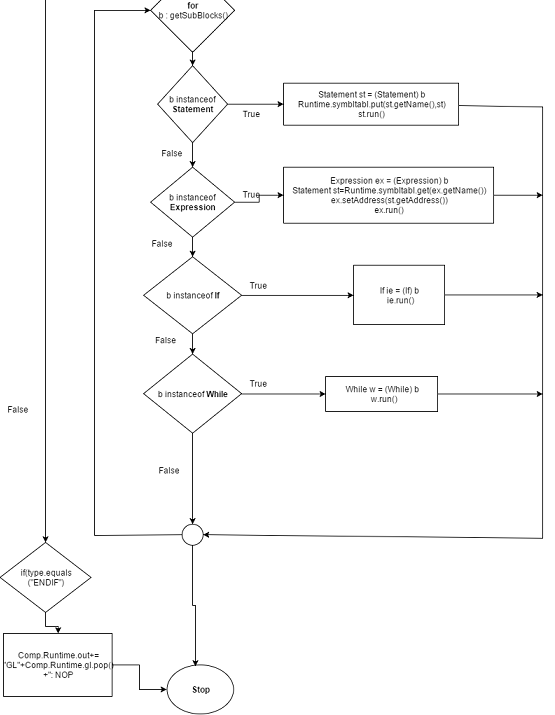


**Fig. 4.6**

4.3.6 IF FLOWCHART

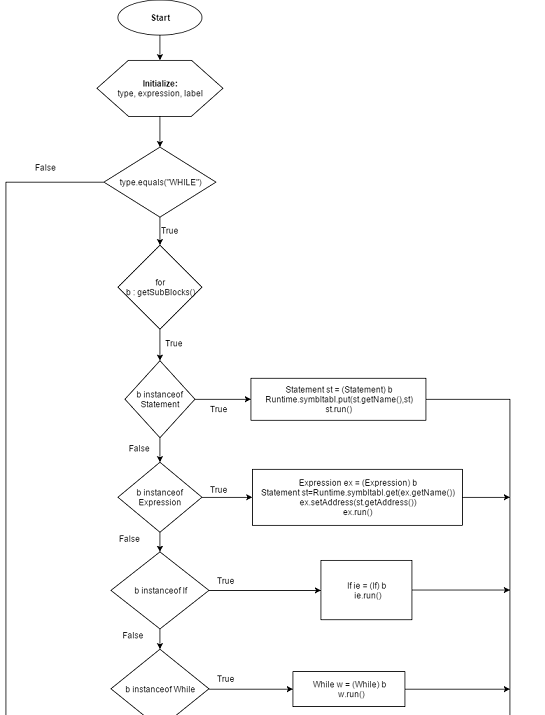


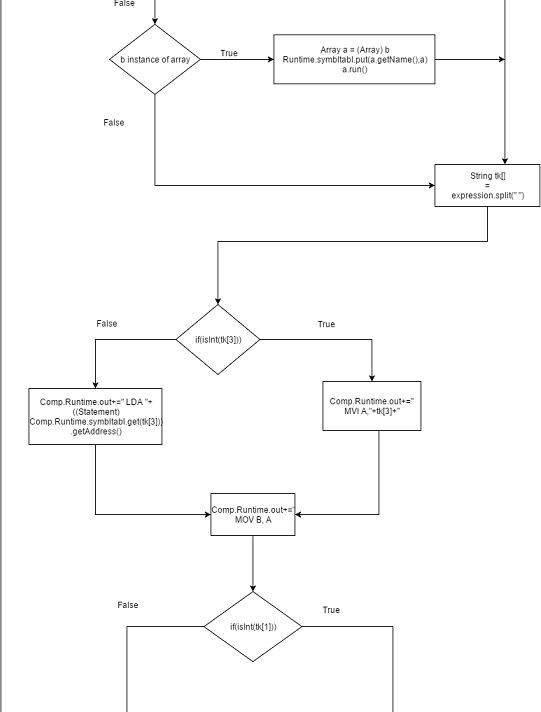


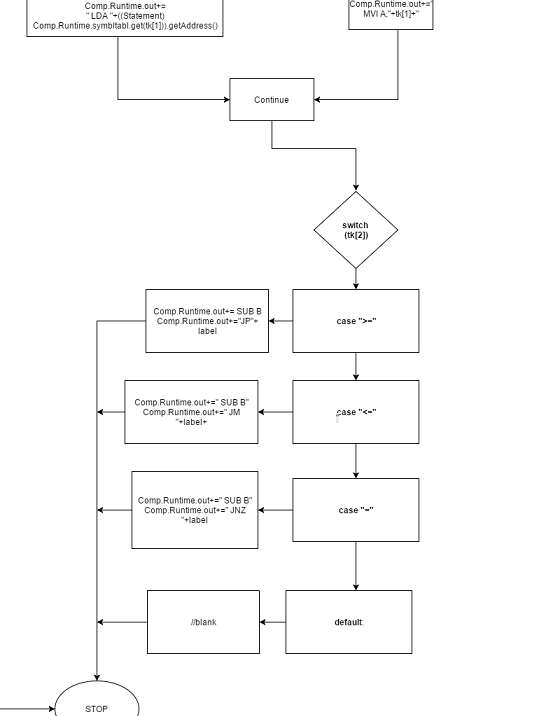


**Fig. 4.7**

4.3.7 WHILE FLOWCHART

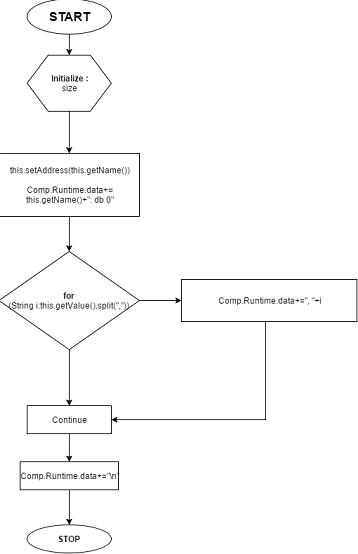






**Fig. 4.8**

**4.3.8 ARRAY FLOWCHART**



**Fig. 4.9**

**5. IMPLEMENTATION**

* 1. **Runtime**

/\*

Function: Identify each block and Generate code

Input: Basic language code

Output: code

\*/

String code=null,out="",data="";

/\* code:- for the input code

out:- for output code

data:- for data segment code \*/

HashMap<String,Statement> symbltabl; /\*symbltabl:- used to store all the variables declared in the program \*/

int varbadd=1000,globalLabel=1; /\* varbadd:- stores the address where

variables are stored, starts at 1000

globalLabel:- used to provide labels for

various jump statement \*/

Stackn gl; /\* gl:- used to keep track of all the global declared labels

\*/

int getvarbadd() /\* increments the varbadd value by one \*/

int incrGlobalLabel() /\* increments the globallabel value by one \*/

Runtime()

success = false

Block block //for the whole program

for every line in code.split("\n") do

Tokenize tokenizer = new Tokenize()

String token[]=tokenizer.givetoken(line) //tokenize each

lines

if token[1] is "LET"

newBlock= new Statement(block,token[2],line.substring(line.indexOf(token[4],4))) //create a statement block

block.addBlock(newBlock) // add the block as child to the current parent blocks

else if token[1] is "DO"

newBlock= new Expression(block,token[2],line.substring(line.indexOf(token[4],4))) //create a expression block

block.addBlock(newBlock) // add the block as child to the current parent blocks

else if token[1] is "IF"

newBlock= new If("IF",line.substring(line.indexOf('('),line.indexOf(')')+1),"GL"+globalLabel,block) //create If block

gl.push(globalLabel)

incrGlobalLabel() /\* push a global label value and

increment it \*/

block.addBlock(newBlock) // add the block as child

to the current parent blocks

block=newBlock //change the current block as the

main

else if token[1] is "ENDIF"

newBlock= new If("ENDIF","","",block.getSuperBlock()); //create If block

block=block.getSuperBlock() //make the parent

block as main bock

block.addBlock(newBlock) //add the block as child

to the current parent

blocks

else if token[1] is "WHILE"

newBlock= new While("WHILE",line.substring(line.indexOf('('),line.indexOf(')')+1),"GL"+globalLabel,block) //create While block

gl.push(globalLabel)

incrGlobalLabel()/\* push a global label value and

increment it \*/

block.addBlock(newBlock) // add the block as child

to the current parent blocks

block=newBlock //change the current block as the

main

else if token[1] is "WEND"

newBlock= new While("WEND","","",block.getSuperBlock()) //create While block

block=block.getSuperBlock() //make the parent block as main bock

block.addBlock(newBlock); // add the block as child to the current parent blocks

else if token[1] is "DIM"

newBlock= new

Array(block,token[2].substring(0,token[2].indexOf('(')),token[2].substring(token[2].indexOf('(')+1,toke[2].indexOf(')')),line.substring(line.indexOf('[')+1,line.indexOf(']'))) //create array block

block.addBlock(newBlock) // add the block as child

to the current parent blocks

block.run(); // run the block object

//arrange the code accordingly into code

* 1. **BLOCK**

/\*

Function: Store the data about code

Input: parent block

Output: returns block object

\*/

Block superBlock //stores the super block reference

ArrayList<Block> subBlocks //stores sublocks as child

Block(Block superBlock) // initialize the instance variabes

Block getSuperBlock() // to get the parent block

Block[] getSubBlocks() // get all the sub blocks

addBlock(Block block) // adds the block to subblock

run()

for every Block in getSubBlocks() do

if b instanceof Statement // for a statement type

st = (Statement) b //create statement block

Runtime.symbltabl.put(st.getName(),st)//store the

symbol into the symbol table

st.run() // runt the block

else if b instanceof Expression // for a expression

type

ex = (Expression) b//create expression block

st=Runtime.symbltabl.get(ex.getName())

ex.setAddress(st.getAddress())//get the address of the variable from its corresponding statement

ex.run()// runt hte block

else if b instanceof If // for a if type

If ie = (If) b;//create If block

ie.run()// run the block

else if b instanceof While // for a while type

While w = (While) b// create While block

w.run()// runt the block

else if b instanceof Array // for a array type

Array a = (Array) // create array block

Runtime.symbltabl.put(a.getName(),a)// store array

variable to symbol table

a.run()// runt the block

* 1. **Exp2post**

/\*

Function: Converts an expression to its postfix form

Input: input expression

Output: postfix exxpression

\*/

Stack theStack //used to store operators

String input[] //contains the input expression

QueueTokens output // to store output postfix expression

Exp2Post(String in) // initialize the variables

doTrans()

i=0

while i less then input.length do //checks for each string in the

queue

String ch = input[i];

switch (ch)

case "+":

case "-":

gotOper(ch, 1);

break;

case "\*":

case "/":

gotOper(ch, 2);

break;

case "(":

theStack.push(ch);

break;

case ")":

gotParen(ch);

break;

default:

output.push(ch);

break;

i++

while (!theStack.isEmpty())

output.push(theStack.pop()) // push all the remaining symbols

PostfixEvaluationpe=new PostfixEvaluation(output)

gotOper(String opThis, int prec1)

/\* if scanned symbl is operator push it on to the stack and go on copying elements of stack if scanned symbol is lesser equal precedence then top of the stack \*/

while theStack is not empty do

opTop = theStack.pop()

if opTop is "("

theStack.push(opTop)

break

else

int prec2;

if opTop is "+" or opTop is "-"

prec2 = 1

else

prec2 = 2

if prec2 < prec1

theStack.push(opTop)

break

else output.push(opTop)

theStack.push(opThis)

gotParen(String ch)

/\*Scanned symbol is open bracket push it on stack pop out elements if it is right parenthesis \*/

while theStack is not empty do

chx = theStack.pop()

if chx is "("

break

else output.push(chx)

* 1. **PostfixEvaluation**

/\*

Function: Evaluates the postfix expression

Input: postfix expression

Output: evaluted form

\*/

stckLabel=1 /\*stckLabel:- for declaring local labels\*/

PostfixEvaluation(QueueTokens q)

int n,stckadd=2500; //n:- size of the expression , stckadd:-

starting address for the stack used in evaluation

a=new Stack(n);//a is the stack where each numbers are pushed

Stackn lb=new Stackn(100); //to keep track of labels

Comp.Runtime.out+=" LXI H, "+stckadd+" \n";//loads the stack

address

while q is not empty

ch=q.pop()

if not (ch is "+" or ch is "-" or ch is "\*" or ch is "/")

if ch is not int

if ch.indexOf("[") is -1 //thus the variable is

not array type get the value from the address

Comp.Runtime.out+=" LDA "+Comp.Runtime.symbltabl.get(ch).getAddress()+" \n"

else // if array index is variable so retrieve

data for that array element

if((ch.substring(ch.indexOf("[")+1,ch.indexOf("]")))is not int) // if index is represented by third variable

Comp.Runtime.out+=" LDA "+Comp.Runtime.symbltabl.get((ch.substring(ch.indexOf("[")+1,ch.indexOf("]")))).getAddress()+" \n"+" MOV C, A\n"+" MVI A, "+0+"\n"+"LXI D,"+ch.substring(0, ch.indexOf("["))+"\n"+"L"+(stckLabel)+": ADI 01\n"+" INR E\n"+" DCR C\n"+" JNZ L"+lb.pop()+"\n"+" LDAX D \n"

else //if index is constant thus directly retrieve

Comp.Runtime.out+=" LDA "+Comp.Runtime.symbltabl.get(ch.substring(0, ch.indexOf("["))).getAddress()+"+"+ch.substring(ch.indexOf("[")+1,ch.indexOf("]"))+" \n"

Comp.Runtime.out+=" MOV M, A \n"+" INR L \n"

a.push(ch)

else //if it is an int type so just directly load it

Comp.Runtime.out+=" MVI A, "+ch+" \n"+" MOV M, A \n"+" INR L \n"

a.push(ch)

else // if it is operator pop out last two elements and apply the operator and store the result

x=a.pop();

y=a.pop();

Comp.Runtime.out+=" DCR L \n"+" MOV B, M\n"+" DCR L\n"+

"MOV A, M\n";

switch(ch)

case "+":Comp.Runtime.out+=" ADD B \n"//for

addition

break

case "-":Comp.Runtime.out+=" SUB B \n"//for

substraction

break

case "\*":Comp.Runtime.out+=" MOV C, A\n"+" MVI A,

"+0+"\n"+"L"+(stckLabel)+": ADD B\n"+" DCR C\n"+" JNZ L"+lb.pop()+"\n"

//for multiplication

break

case "/"://or division

Comp.Runtime.out+=" MVI C, "+0+"\n"+"L"+(stckLabel)+": SUB B\n"+" INR C\n"+" CMP B\n"+" JNC L"+lb.pop()+"\n"+" MOV A,C\n"

break

default:

Comp.Runtime.out+=" MOV M, A\n" //store the output

r=a.pop();

* 1. **IF Implementation**

/\*

Function: For if block

Input: type, expression, label for the IF block

Output: returns If block

\*/

String type,expression,label

/\* type:- to store type of the block

expression:- store the evaluating condition of if block

label:- the used by the if to jump from one place to other\*/

If(String type, String expression, String label, Block superBlock) //initializes instance variable

//to get the variables for the object

String getType()

String getExpression()

String getLabel()

//to set the variables for the object

void setExpression(String expression)

void setLabel(String label)

void setType(String type)

run()

if type is "IF" //when the block IF type

tk[]=expression.split(" ")

if tk[3] is integer // checks if the first operator is

integer loads it

Comp.Runtime.out+=" MVI A,"+tk[3]+"\n"

else //gets the value from the variable address

Comp.Runtime.out+=" LDA "+((Statement)Comp.Runtime.symbltabl.get(tk[3])).getAddress()+"\n"+" MOV B, A \n"

` if tk[1] is integer // checks if the second operator is

integer loads it

Comp.Runtime.out+=" MVI A,"+tk[1]+"\n"

else //gets the value from the variable address

Comp.Runtime.out+=" LDA "+((Statement)Comp.Runtime.symbltabl.get(tk[1])).getAddress()+"\n"

switch tk[2] //and then compares accordingly

case ">=":Comp.Runtime.out+=" SUB B \n"+" JM

"+label+" \n"

break;

case "<=":Comp.Runtime.out+=" SUB B \n"+" JP "+label+" \n"

break;

case "=":Comp.Runtime.out+=" SUB B \n"+" JNZ "+label+" \n"

break;

for every Block in getSubBlocks() do

if b instanceof Statement // for a statement type

st = (Statement) b //create statement block

Runtime.symbltabl.put(st.getName(),st)//store the symbol into the symbol table

st.run() // runt the block

else if b instanceof Expression // for a expression

type

ex = (Expression) b//create expression block

st=Runtime.symbltabl.get(ex.getName())

ex.setAddress(st.getAddress())//get the address of the variable from its corresponding statement

ex.run()// runt hte block

else if b instanceof If // for a if type

If ie = (If) b;//create If block

ie.run()// run the block

else if b instanceof While // for a while type

While w = (While) b// create While block

w.run()// runt the block

else if b instanceof Array // for a array type

Array a = (Array) // create array block

Runtime.symbltabl.put(a.getName(),a)// store array

variable to symbol table

a.run()// runt the block

else if type is "ENDIF" // if type is "ENDIF" jus place label there with no operation

Comp.Runtime.out+="GL"+Comp.Runtime.gl.pop()+": NOP \n"

* 1. **STATEMENT Implementation**

/\*

Function: for statement block

Input: type, value, address for the statement block

Output: returns statement block

\*/

String name, value,address

/\* name:- for assigning the variable name

value:- for storing either the vale or the expression

address:- to store microprocessor address for the variable\*/

Statement(Block superBlock, String name,String value) //initializes the variable

//to get values of the variables

String getName()

String getValue()

String getAddress()

//to set the values for the variables

void setName(String name)

void setValue(String value)

void setAddress(String address)

void run()

if value is integer // if the value is not an expression

this.address=""+Runtime.getvarbadd()// get an address for the variable

Comp.Runtime.out+=" MVI A, "+value+" \n"+" STA "+this.address+" \n";//store variables value to this address

else//for evaluating expression

this.address=""+Runtime.getvarbadd();// get an address for the variable

Exp2Post theTrans = new Exp2Post(value)

theTrans.doTrans()

Comp.Runtime.out+=" STA "+this.address+" \n"//store the value

after evaluating the eexpression

* 1. **ARRAY Implementation**

/\*

Function: for array block

Input: name, size, value for the array block

Output: returns array block

\*/

String size;//size:- to store size of the array

Array(Block superBlock, String name,String size,String value) // initializes the variables

public String getSize() //to get the size

public void setSize(String size) //to set the size

public void run()

this.setAddress(this.getName());//set the address

Comp.Runtime.data+=this.getName()+": db 0";

for every String in this.getValue().split(",") do

Comp.Runtime.data+=", "+i // add each element of the array

Comp.Runtime.data+="\n"

* 1. **EXPRESSION Implementation**

/\*

Function: for expression block

Input: type, value, address for the expression block

Output: returns expression block

\*/

String name, value,address;

/\* name:- for assigning the variable name

value:- for storing either the vale or the expression

address:- to store microprocessor address for the variable\*/

Expression(Block superBlock, String name,String value) //initializes value to the instance variables

//to get values of the variables

String getName()

String getValue()

String getAddress()

//to set the values for the variables

void setName(String name)

void setValue(String value)

void setAddress(String address)

void run()

if value is integer // if the value is not an expression

Comp.Runtime.out+=" MVI A, "+value+" \n"+" STA "+this.address+" \n"//store variables value to variables address

else //for evaluating expression

theTrans = new Exp2Post(value)

theTrans.doTrans()

Comp.Runtime.out+=" STA "+this.address+" \n"//store the value

after evaluating the eexpression

* 1. **WHILE Implementation**

/\*

Function: for while block

Input: type, expression, label for the while block

Output: returns while block

\*/

String type,expression,label

/\* type:- to store type of the block

expression:- store the evaluating condition of if block

label:- the used by the if to jump from one place to other\*/

public While(String type, String expression, String label, Block superBlock) //initializes instance variables

//to get the variables for the object

String getType()

String getExpression()

String getLabel()

//to set the variables for the object

void setExpression(String expression)

void setLabel(String label)

void setType(String type)

void run()

if type is "WHILE" //when it is While type block

Comp.Runtime.out+=label+": NOP\n"//place the label with no

operation

for every Block in getSubBlocks() do

if b instanceof Statement // for a statement type

st = (Statement) b //create statement block

Runtime.symbltabl.put(st.getName(),st)//store the

symbol into the symbol table

st.run() // runt the block

else if b instanceof Expression // for a expression

type

ex = (Expression) b//create expression block

st=Runtime.symbltabl.get(ex.getName())

ex.setAddress(st.getAddress())//get the address of the variable from its corresponding statement

ex.run()// runt hte block

else if b instanceof If // for a if type

If ie = (If) b;//create If block

ie.run()// run the block

else if b instanceof While // for a while type

While w = (While) b// create While block

w.run()// runt the block

else if b instanceof Array // for a array type

Array a = (Array) // create array block

Runtime.symbltabl.put(a.getName(),a)// store array

variable to symbol table

a.run()// runt the block

tk[]=expression.split(" ");

if tk[3] is integer // checks if the first operator is

integer loads it

Comp.Runtime.out+=" MVI A,"+tk[3]+"\n"

else //gets the value from the variable address

Comp.Runtime.out+=" LDA "+((Statement)Comp.Runtime.symbltabl.get(tk[3])).getAddress()+"\n"

Comp.Runtime.out+=" MOV B, A \n"

if tk[1] is integer // checks if the second operator is integer loads it

Comp.Runtime.out+=" MVI A,"+tk[1]+" \n"

else //gets the value from the variable address

Comp.Runtime.out+=" LDA "+((Statement)Comp.Runtime.symbltabl.get(tk[1])).getAddress()+" \n"

switch tk[2] //and then compares accordingly

case ">=":Comp.Runtime.out+=" SUB B \n"+" JP

"+label+" \n";

break;

case "<=":Comp.Runtime.out+=" SUB B \n"+" JM "+label+" \n";

break;

case "=":System.out.println(" SUB B")+" SUB B \n"+" JNZ "+label+" \n";

break;

Comp.Runtime.gl.pop()

else if type is "WEND"

//No Operation

**6. TESTING**

**Statement and Expression**

**INPUT**:

10 LET A = 0  
20 LET E = 0  
30 DO E = ( E + 1 )  
40 DO A = ( A + E )

**OUTPUT:**  
;<Program>

jmp start

;data

;code  
start: nop  
MVI A, 0  
STA 1000  
MVI A, 0  
STA 1001  
LXI H, 2500  
LDA 1001  
MOV M, A  
INR L  
MVI A, 1  
MOV M, A  
INR L  
DCR L  
MOV B, M  
DCR L

MOV A, M  
ADD B  
MOV M, A  
INR L  
STA 1001  
LXI H, 2500  
LDA 1000  
MOV M, A  
INR L  
LDA 1001  
MOV M, A  
INR L  
DCR L  
MOV B, M  
DCR L  
MOV A, M  
ADD B  
MOV M, A  
INR L  
STA 1000  
HLT

**IF Block**

**OUTPUT:**  
;<Program>

jmp start

;data

;code  
start: nop  
MVI A, 2  
STA 1000  
MVI A, 20  
STA 1001  
MVI A,2  
MOV B, A  
LDA 1000  
SUB B

JNZ GL1  
LXI H, 2500  
LDA 1000  
MOV M, A  
INR L  
LDA 1001  
MOV M, A  
INR L  
DCR L  
MOV B, M  
DCR L  
MOV A, M  
ADD B  
MOV M, A  
INR L  
STA 1000  
GL1: NOP  
HLT

**INPUT**:

10 LET A = 2  
20 LET B = 20  
30 IF ( A = 2 ) THEN  
40 DO A = ( A + B )  
50 ENDIF

**WHILE Block**

**OUTPUT** :  
;<Program>  
  
jmp start  
  
;data  
  
;code  
start: nop  
MVI A, 0   
STA 1000   
MVI A, 0   
STA 1001   
GL1: NOP  
LXI H, 2500   
LDA 1001   
MOV M, A   
INR L   
MVI A, 1   
MOV M, A

INR L   
DCR L   
MOV B, M  
DCR L  
MOV A, M  
ADD B   
MOV M, A  
INR L  
STA 1000   
MVI A,11  
MOV B, A   
LDA 1001   
SUB B   
JM GL1   
HLT

INR L   
DCR L   
MOV B, M  
DCR L  
MOV A, M  
ADD B   
MOV M, A  
INR L  
STA 1001   
LXI H, 2500   
LDA 1000   
MOV M, A   
INR L   
LDA 1001

MOV M, A

**INPUT:**

10 LET A = 0  
20 LET E = 0  
30 WHILE ( E<= 11 ) THEN  
40 DO E = ( E + 1 )  
50 DO A = ( A + E )  
60 WEND

**ARRAY Block**

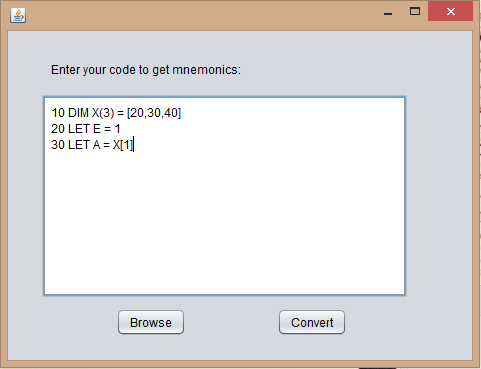
**INPUT :**  
10 DIM X(3) = [20,30,40]  
20 LET E = 1  
30 LET A = X[1]

**OUTPUT:**  
;<Program>  
  
jmp start  
  
;data  
  
X: db 0, 20, 30, 40  
;code  
start: nop

MVI A, 1   
STA 1000   
LXI H, 2500   
LDA X+1   
MOV M, A   
INR L   
STA 1001   
HLT

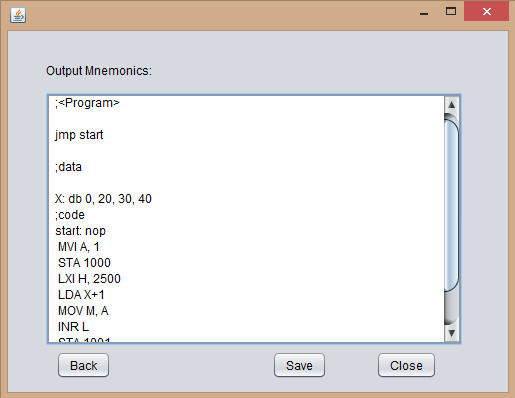
**7. SCREENSHOTS**

First page (input)

****

**Fig. 7.1**

Second Page (output)

****

**Fig. 7.2**

**8. CONCLUSION**

It was a wonderful and learning experience for us while working on this project. This project help us learn basics of both BASIC Language as well as 8085 language.

It was due to this project that we have come to know about a very basic programming language i.e. BASIC Language.

We enjoyed each and every bit of work that we had put in this project. The project is further extendable.

**9. FUTURE SCOPE**

* Future enhancements
  + New features can incorporate like function calls .
  + Automatic testing of generated code using GNUsim8085 instead of first copying and then testing.

*REFERENCES:*

1. Lecture : Youtube lecture <https://www.youtube.com/playlist?list=PLAF3anQEEkzRuJ7UW82CmSEtT6g-J4Q_p>
2. Web : “BASIC programming Language”<http://www.cs.bris.ac.uk/~dave/basic.pdf>
3. Microprocessor Architecture, Programming and Applications with the 8085Book by Ramesh S. Gaonkar
4. Youtube Video lectures on " Microprocessors and Microcontrollers " by  
   Prof.Ajit Pal
5. Java The complete reference by Herbert Schildt
6. Guide To the BASIC programming Language : <http://media.salford-systems.com/pdf/spm7/BasicProgLang.pdf>
7. Principles of Compiler Design , Textbook by Alfred Aho and Jeffrey Ullman
8. JavaDoc reference : <https://docs.oracle.com/javase/7/docs/api/>
9. JDK reference : http://www.oracle.com/technetwork/java/javase/downloads/index.html

**Appendix**

SAMPLE PROGRAM

Example 1

Input:- A program to generate factorial of a number in basic language

10 LET A = 1

20 LET E = 0

50 WHILE ( E <= 4 ) THEN

60 DO E = ( E + 1 )

70 DO A = ( A \* E )

80 WEND

Output:-After executing the code in simulator, the ouput for factorial is stored at the address 1000 of microprocessor

;<Program>  
  
jmp start  
  
;data  
  
;code  
start: nop  
MVI A, 0   
STA 1000   
MVI A, 0   
STA 1001   
GL1: NOP  
LXI H, 2500   
LDA 1001   
MOV M, A   
INR L   
MVI A, 1   
MOV M, A

INR L   
DCR L   
MOV B, M  
DCR L  
MOV A, M  
ADD B   
MOV M, A  
INR L  
STA 1001   
LXI H, 2500   
LDA 1000   
MOV M, A   
INR L   
LDA 1001

MOV M, A

INR L   
DCR L   
MOV B, M  
DCR L  
MOV A, M  
ADD B   
MOV M, A  
INR L  
STA 1000   
MVI A,11  
MOV B, A   
LDA 1001   
SUB B   
JM GL1   
HLT

Example 2

Input:- A program to generate summation of all array elements using basic language

10 LET A = 0

20 LET E = 0

30 DIM X(3) = [20,30,40]

40 WHILE ( E <= 3 ) THEN

50 DO E = ( E + 1 )

60 DO A = ( A + X[E] )

70 WEND

Output:-After executing the code in simulator, the ouput for summation is stored at the address 1000 of microprocessor, which is the corresponding address for variable A.

;<Program>

jmp start

;data

X: db 0, 20, 30, 40

;code

start: nop

MVI A, 0

STA 1000

MVI A, 0

STA 1001

GL1: NOP

LXI H, 2500

LDA 1001

MOV M, A

INR L

MVI A, 1

MOV M, A

INR L

DCR L

MOV B, M

DCR L

MOV A, M

ADD B

MOV M, A

INR L

STA 1001

LXI H, 2500

LDA 1000

MOV M, A

INR L

LDA 1001

MOV C, A

MVI A, 0

LXI D,X

L1: ADI 01

INR E

DCR C

JNZ L1

LDAX D

MOV M, A

INR L

DCR L

MOV B, M

DCR L

MOV A, M

ADD B

MOV M, A

INR L

STA 1000

MVI A,3

MOV B, A

LDA 1001

SUB B

JM GL1

HLT

Example 3

Input:- A program to find the maximum of all array elements using basic language

10 LET A = 0

15 LET E = 0

20 LET M = 0

25 DIM X(3) = [20,30,40]

30 WHILE ( E <= 3 ) THEN

35 DO E = ( E + 1 )

40 DO A = X[E]

45 IF ( M <= A ) THEN

50 DO M = A

55 ENDIF

60 WEND

Output:-After executing the code in simulator, the maximum number is stored at the address 1002 of microprocessor, which is the corresponding address for variable M.

DCR L

MOV B, M

DCR L

MOV A, M

ADD B

MOV M, A

INR L

STA 1001

LXI H, 2500

LDA 1001

MOV C, A

MVI A, 0

LXI D,X

L1: ADI 01

INR E

DCR C

JNZ L1

LDAX D

MOV M, A

INR L

STA 1000

;<Program>

jmp start

;data

X: db 0, 20, 30, 40

;code

start: nop

MVI A, 0

STA 1000

MVI A, 0

STA 1001

MVI A, 0

STA 1002

GL1: NOP

LXI H, 2500

LDA 1001

MOV M, A

INR L

MVI A, 1

MOV M, A

INR L

LDA 1000

MOV B, A

LDA 1002

SUB B

JP GL2

LXI H, 2500

LDA 1000

MOV M, A

INR L

STA 1002

GL2: NOP

MVI A,3

MOV B, A

LDA 1001

SUB B

JM GL1

HLT

USE CASE DESCRIPTION

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 1 | | |
| Use Case Name: | JFrame | | |
| Created By: | Raunak Kumar | Last Updated By: | Raunak Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Actor writes or browses code and gets mnemonics as result. |
| Preconditions: | a. Application is already running.  b. Actor is on the first page. |
| Priority: | High |
| Frequency of Use: | High |
| Flow of Events: | User:  1. Actor writes code or browses the basic code.  2. Actor clicks the convert button.  3. The application transforms this code into corresponding mnemonic code.  4. Result is available to user and can be saved. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 2 | | |
| Use Case Name: | Runtime | | |
| Created By: | Raunak Kumar | Last Updated By: | Raunak Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Actor’s code is transferred to this class for processing. |
| Preconditions: | a. Application is already running.  b. Actor is successfully transferred to this class . |
| Postconditions: |  |
| Priority: | High |
| Frequency of Use: | High |
| Flow of Events: | User:  1. Actor code is transferred to this class.  2. The cod is processed line by line.  3. The application tries to recognize which type of block the line resembles with and stores respective object accordingly.  4. At the end it calls the run method for each block. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 3 | | |
| Use Case Name: | Block | | |
| Created By: | Raunak Kumar | Last Updated By: | Raunak Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Runtime tries to create object of blocks to store data. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Postconditions: |  |
| Priority: | High |
| Frequency of Use: | High |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective blocks object is created.  3. At the end the run method of the block is called. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 4 | | |
| Use Case Name: | Exp2Post | | |
| Created By: | Raunak Kumar | Last Updated By: | Raunak Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Block uses this to evaluate its expressions. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Postconditions: |  |
| Priority: | High |
| Frequency of Use: | Less than moderate |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective blocks object is created.  3. Whenever there is expression to be evaluated block class calls this.  4. This first transforms the expression into postfix form with help of operator precedence. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 5 | | |
| Use Case Name: | PostfixEvaluation | | |
| Created By: | Raunak Kumar | Last Updated By: | Raunak Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Block uses this to evaluate its expressions. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Postconditions: |  |
| Priority: | High |
| Frequency of Use: | Less than moderate |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective blocks object is created.  3. Exp2Post has changed the expressions of a block into postfix form.  4. Now using stack we evaluate the postfix form of expression along with corresponding values for the variables to get the result. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 6 | | |
| Use Case Name: | If | | |
| Created By: | Parimal Kumar | Last Updated By: | Parimal Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Extends Block class and is used for If statement. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Priority: | High |
| Frequency of Use: | Less than moderate |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective If block object is created.  3. At the end the run method of the block is called. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 7 | | |
| Use Case Name: | Statement | | |
| Created By: | Parimal Kumar | Last Updated By: | Parimal Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Extends Block class and is used for declaration. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Priority: | High |
| Frequency of Use: | Less than moderate |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective Statement block object is created.  3. At the end the run method of the block is called. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 8 | | |
| Use Case Name: | Expression | | |
| Created By: | Parimal Kumar | Last Updated By: | Parimal Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Extends Block class and is used for Expression. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Priority: | High |
| Frequency of Use: | Less than moderate |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective Expression block object is created.  3. At the end the run method of the block is called. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 9 | | |
| Use Case Name: | While | | |
| Created By: | Parimal Kumar | Last Updated By: | Parimal Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Extends Block class and is used for While statement. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Postconditions: |  |
| Priority: | High |
| Frequency of Use: | Less than moderate |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective While block object is created.  3. At the end the run method of the block is called. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | 10 | | |
| Use Case Name: | Array | | |
| Created By: | Parimal Kumar | Last Updated By: | Parimal Kumar |
| Date Created: | 4/3/2016 | Date Last Updated: | 4/3/2016 |

|  |  |
| --- | --- |
| Actor: | User |
| Description: | Extends Block class and is used for Array statement. |
| Preconditions: | a. Application is already running.  b. Actor code is processed by Runtime. |
| Priority: | High |
| Frequency of Use: | Less than moderate |
| Flow of Events: | User:  1. Actor code is processed by Runtime.  2. Accordingly respective While block object is created.  3. At the end the run method of the block is called. |
| Alternative Flows: | 1. System finds entered code are in incorrect format. System shows a error message. 2. Actor makes proper changes and press convert. 3. System acknowledges with mnemonic code as result. |