**CMPE 230 Homework 1**

**Student Names & Numbers:**

Yağmur Ceren Dardağan 2014400063

Buse Ece Güldiken 2015400087

**INTRODUCTION**

This project aims to design a program which translates \*.bc files to \*.asm files. It first tokenizes the statement from \*.bc file, places every token into two stacks regarding priorities and types of the elements. It puts variables into variable stack and defined operators into operator stack.

Then it writes respective assembly statements while emptying the variable stack and the operator stack.

The constraints about the project and the exceptions handled in the project are explained below.

**Constraints:**

* An undefined variable has value 0.
* Variable names start with $.
* A variable’s name can not be empty.
* All variables and expressions are 16 bit.
* All numbers are expressed in hexadecimal format.
* The operations can accept expressions.
* & has higher precedence than | .

**Example parsing of a statement**

1. Input statement:

$y = not(ls($y,2))

1. Concatenate each character, get rid of the blank character

$y=not(ls($y,2))

1. Tokenize the statement and make an array, while making an array add extra paranthesis to detect their priorities in a correct manner.Paranthesis increases the next statement’s priority number by 2.

[$y, =, not, (, ls ,( ,$y ,2 ,) , ) ]

1. Assign priorities to each element

[$y, =, not, (, ls ,( ,$y ,2 ,) , ) ]

[0 , 0, 2 , 2, 4, 4, 6, 6, 4, 2]

1. Push variables & numbers to variableStack, operators to operatorStack regarding priorities

Variable Stack

(tail)[$y, $y, 2

Operator Stack

(tail)[=, not, ls

1. Initiliaze Variable stack in assembly language (Use another stack put all variables into other stack and while popping write push statements)
2. Perform translation

Several exceptional cases are not explained here in sake of simplicity but

you can find the explanation in the comments.

While(operatorStack not empty)

Operator<- OperatorStack.pop

if it was the last operator, perform mov operation

else

//Take how many arguments this operator takes

popTimes<- popTimes.get(operator)

if(popTimes == 2) write pop ax, pop bx

else if (popTimes == 1) write pop ax

if it is a shift write pop cx, pop ax

if it is not a first operation of ‘|’ or ‘&’ pop 3 times

perform the operation

push back the value into variable stack

endwhile

1. Use displayedVariables list in order to perform display at the end of the file i.e. display $x

**EXCEPTIONS**

It raises InvalidSyntaxException with message “bad syntax” on the command line if;

* There is no “=” but it is an assignment statement
* There is more than 1 “=” in assignment statement
* ‘|’ or ‘&’ only have 1 argument
* There are unmatched left & right paranthesis
* The name of a variable is empty
* File to read is empty
* The statement does not start with ‘$’ sign

Empty line between statements is handled in the project

**BITC CLASS**

We only have bitc.class (name of the class is minuscule in order to meet project’s requirements.)

Every operation is handled under this class.

**Main lists & sets used in the project:**

* toRead (ArrayList)

includes all lines of the file

* variables (Set)

includes all variables used in the file

* operators (ArrayList)

all operators defined in the project

* popTimes (HashMap)

number of arguments that operators take

* arrayStr (ArrayList)

includes array version of each line

* checkPoints (ArrayList)

includes checkpoints in a line

**Methods:**

|  |
| --- |
| [**checkForErrors**](bitc.html#checkForErrors-java.util.ArrayList-)(java.util.ArrayList<java.lang.String> arrayStr)  Checks if there are equal number of "(" and ")" paranthesis and the correct number of equal sign in the  statement, if not, raises and InvalidSyntaxError with message “bad syntax” |
| [**containsEqual**](bitc.html#containsEqual-java.lang.String-)(java.lang.String line)  Checks if the line contains equal sign |
| [**createStack**](bitc.html#createStack-java.util.ArrayList-java.util.ArrayList-java.util.ArrayList-boolean-)(java.util.ArrayList<java.lang.Integer> prioties, java.util.ArrayList<java.lang.String> line, java.util.ArrayList<java.lang.String> operators, boolean check)  Creates stacks regarding the priorities determined in findPriority The elements with the lower priority are  pushed deeper. Operator stack are created with boolean value true and variable stack are created with the    boolean value false. |
| [**findNextOperator**](bitc.html#findNextOperator-java.lang.String-)(java.lang.String s)  Takes a line and returns it into an array Uses concatNumbers function to accumulate numbers,  e.g. it gets f123h as 1 token not takes it 5 tokens as f, 1, 2, 3, h |
| [**findPriority**](bitc.html#findPriority-java.util.ArrayList-)(java.util.ArrayList<java.lang.String> lines)  It assigns priority to each operator, It increments priority by 2 if it enters into paranthesis decrements by 2  if it leaves paranthesis if it is 'and' operation its priority will be higher by 1 than others |
| [**getLines**](bitc.html#getLines-java.io.File-)(java.io.File f)  Gets the file and reads its content via Scanner Put its lines into an ArrayList |
| [**main**](bitc.html#main-java.lang.String:A-)(java.lang.String[] args)  Reads a file and calls necessary methods to perform tokenization on statements. It translates each line to  assembly language by tokenizing line, creating operator and variable stacks and printing the line’s assembly  version to example.asm file. |
| [**makeArray**](bitc.html#makeArray-java.lang.String-)(java.lang.String s)  It makes array from each line, it checks for "(",",",")","=" and adds the tokens between these checkpoints  and raises an InvalidSyntaxException if the line does not meet with the constraints . |
| [**makeDisplay**](bitc.html#makeDisplay-java.io.PrintWriter-)(java.io.PrintWriter out)  It is used to display variables at the end, if there is more than one variable to display it labels its sections'  differently e.g. display1: for the first one display2: for the 2nd.. |
| [**makePush**](bitc.html#makePush-java.io.PrintWriter-)(java.io.PrintWriter out)  Makes pushes in assembly language regarding elements in the stack. e.g. push offset vx, push vx w |
| [**setPopTimes**](bitc.html#setPopTimes--)()  Map the number of arguments needed by the operators |
| [**transferFromLine**](bitc.html#transferFromLine--)()  lineStack is used to store elements in POSTFIX notation, in order to put elements into assembler's stack in  the same order we are using a second stack. |
| [**withoutSpace**](bitc.html#withoutSpace-java.lang.String-)(java.lang.String s)  Returns the same string without any space character |
| [**writeInitialization**](bitc.html#writeInitialization-java.io.PrintWriter-)(java.io.PrintWriter out)  When a new variable is defined, its initilisation is pushed into variableInits arraylist before the code ends,  this method is called in order to make inits e.g. vx dw 0 |

**HOW TO RUN THE PROJECT**

Open Terminal, go to project file

Go to src file under project file

Write “javac bitc.java” to compile

Write “java bitc example.bc” to run it with input file

You will get “example.asm” file under bitc project file

If you are using DOSBOX

Put “example.asm” under DOSBOX

Write “c:”

Write “a86 example.asm”

Write “example.com” to run

You will get the output

else

In terminal where a86 is ready write “a86 example.asm”

Write “example.com” to run

You will get the output

**CONCLUSION**

Since assembler is a very detailed program and it is a school project, the program may fail in edge and in some simple exceptional cases. In exceptions part, it is written that which expressions are handled.

It was a good project in terms of understanding the basics of the assembler & compiler and also it was a good practice to learn a86 assembly language.