CMPE 230
Project-1
2016400297 – Hasan Basri BALABAN

In this project, I used two different java file. First one is the **COMP_Main.java** which read the data and convert the given expressions to assembly language. The second file is **Vars.java** which holds information about the given expressions.

COMP_Main.java

COMP_Main class holds four static instance, a printstream object which is used for printing info to the necessary files and three static integers which is used to avoid errors in the .asm file while performing the operations.
The main function has two important loop to perform the necessary conversions and printing the required .asm file.

First loop takes every single line from .co file and control it's form. If there is a syntax error it prints the error line to the .asm file. If there is not an error in the .co file, it parses every line and converts the expressions from infix notation to postfix notation. It adds every expression to vars array list which hold Vars objects. While checking the expressions it follows an algorithm like this :

-If the line is not empty
--If the expression is an assignment expression
---If the parenthesis in the line are in correct form
----Split the line in 2 pieces, LHS (left-hand side) and RHS (right-hand side)
----If the LHS of the expression does not contains a syntax error, set the necessary values
-----If it contains error, prints error line to .co file and end the program
----Replace "pow(a,b)" form with "a^b"
----If the RHS of the expression does not contains a syntax errors, set the necessary values
-----If it contains errors, prints error line to .co file and end the program
---If the parenthesis are not in correct form, prints error line to .co file and end the program
--If the expression is an output expression
---Checks the expression and set the necessary values if it does not contains a syntax error
----Prints error line to .co file if it contains a syntax error
-If the line is empty, just skip the line

After checking syntax errors and converting every expression from infix notation to postfix notation, the program declares all the necessary variables in .asm file.

Then the second important loop comes. This loop reads and executes every element of vars array list (which holds all the expressions). While parsing expressions to assembly code it follows an algorithm like this:

-If the expression is an assignment expression
--Get the line in postfix form and start reading it's elements
---If the element is an operand, create necessary assembly code in .asm file
---Else if the element is an operator, call the operator methods and create assembly code in .asm file
-If the expression is an output expression
–-Read every element of the current line
---If the current element is an only variable or number to print, create the assembly code and print it
---If it is a mathematical expression, calculate the final value by creating assembly code and print it

After the second loop, main method prints the ending statements to the end of the .asm file.

The main function ends. In below, there are explanations of every method that used in this project.

isNumber(String s) Method
This method checks if the given string contains a hexadecimal number or not. It follows an algorithm like this:
-If the first character in the string is an integer
--Try to parse string s to integer, if it is possible return true
--Catch NumberFormatException, return false
-If the first character is not an integer, return false

output(String s) Method
This method creates the necessary assembly code to print the given variable in hexadecimal form. It prints the same code two times and it updates the printCounter every time. printCounter is an integer used to avoid duplicating label names in assembly code.

assignment(String s) Method
This method creates the necessary assembly code to assign the value which generated in RHS of the expression to the given variable. Since I used double word (32-bit) variables in this project, it assign the values two times.

power() Method
This method creates the assembly code to perform power operation. It follows this algorithm:
-Pop all two operand to temporary variables
-Check the power number
--If it is zero, push stack 1h and end the loop
--If it is one, push the base number to the stack and end the loop
--If it is more than one
---Multiply the number with itself one time and decrease the power by one
---If the power is more than or equal to two, repeat the loop
---If it is less than two, push the final number to the stack and end the loop

multiplication() Method
This method creates the assembly code to execute the multiplication operation. It performs the operation using this logic :
-a1a0 = (a1*2^16+a0)
-b1b0 = (b1*2^16+b0)
-a1a0*b1b0 = a1b1*2^32 + a1b0*2^16 + a0b1*2^16 + a0b0

addition() Method
This method creates the necessary assembly code to execute the addition operation. It follows this algorithm:
-Pop the operands to registers
-First add two lower 16 bits
-Then add the higher 32 bits
--If there is a carry from the first addition, add one to the result of the second addition
-Push the final number to stack

declareVariables(ArrayList<String>varList) Method
This method takes the variable list as parameter and creates the assembly code to declare every variable in the beginning of the assembly file
It creates extra variables for helping the code generation during the operations.

isLineEmpty(String s)
This method returns false if the given string has a character different than a whitespace.

isOutput(String s)
This method is used to figure out it the given expression is an assignment expression or an output expression. It checks if there is a "=" sign in the string.

checkParent(String s)
This method counts the number of "(" and ")". If the numbers are not the same, return false.

varNameCheck(String s)
This method controls the RHS of the expression. If there is an error like "x y = ..." it returns false.

removePow(String s)
This method replace "pow(x,y)" form with "x^y". It counts the number of pow operations and every time it sees a pow it just get rid of the pow and replace the comma sign with power sign.

prec(char ch)
A utility method to return precedence of a given operator. Higher value means higher precedence.

InfixToPostfix(String exp)

This method converts the given infix expression to postfix expression. It first checks the whitespaces in the string. Then using a stack algorithm it converts the expression. Basically the algorithm is like this:
-Take every char in the string and read from the beginning
--If the char is an operand add it to result string
--If the char is an operator
---If the stack's first element has precedence pop from stack to result string
---Else push the operator to stack
--If there no character pop every element from stack to result string
-return result string

Vars.java

Vars.java is used to hold information about the expressions. It just has two String instance, which holds the RHS and LHS of the expression. It has two constructors and a boolean method which returns if the object is an assignment expression or not.

TO RUN THE PROGRAM
Main method takes an argument about which file will be read.
Just give the necessary input file name as an argument in the terminal.

Ex: javac COMP_Main.java (This will create COMP_Main.class)
java COMP_Main ~/path/of/the/input/file/example.co

This code will generate example.asm file