Proiect FIC

-Calculator de Buzunar-

Documentatie

Introducere

Un calculator de buzunar este un dispozitiv electronic alimentat cu baterie utilizat pentru a efectua calcule aritmetice simple la introducerea datelor. Calculatoarele de buzunar și-au primit numele datorită dimensiunii lor compacte, care este suficient de mică și la îndemână și poate fi transportată într-un buzunar.

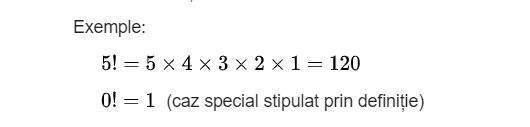
Busicom LE-120A HANDY a fost primul calculator real de mărime de buzunar, realizat în Japonia și comercializat la începutul anului 1971.

Inițial, toate calculatoarele erau calculatoare mecanice. Calculatoarele electronice au fost introduse la începutul anilor 1960, care aveau o dimensiune destul de mare datorită utilizării mai multor tranzistoare, a unei surse de energie uriașe și a altor piese voluminoase. Apoi, la începutul anilor ’70 acestea au început să-și reducă din volum.

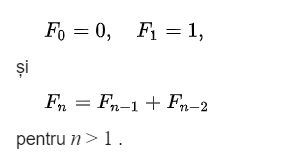
Obiective

Obiectivul nostru este realizarea unui calculator de buzunar performant capabil atât de realizarea operațiilor de baza, cât și a unor operații speciale. Am ales să includem operații elementare cum ar fi adunarea, scaderea, înmulțire, împărțire, dar și operații mai speciale cu ar fi ridicarea la putere, factorialul unui număr sau determinarea elementelor șirului lui Fibonacci.

**Factorial :** În matematică **factorialul** unui număr întreg pozitiv *n*, notat cu *n!*, este egal cu produsul numerelor naturale mai mici sau egale cu *n*. Este o funcție numerică discretă și o operație unară (cu un singur operand).



**Numerele Fibonacci:** În matematică, **numerele Fibonacci**  , denumite în mod obișnuit *F n*  , formează o secvență , **șirul Fibonacci** , în care fiecare număr este suma celor două precedente

Metodologie

**1. Verilog** este un limbaj de descriere a hardware-ului (Hardware Description Language -HDL), destinat descrierii comportamentului și/sau arhitecturii unui sistem numeric, cu alte cuvinte al unei funcții logice combinatorii sau secvențiale. Un sistem numeric poate fi descris, la niveluri diferite, in functie de aspectele care intereseaza. Astfel, un HDL poate descrie, la nivel de comutator, amplasarea traseelor de legatura (firele), a rezistoarelor si tranzistoarelor pe un circuit integrat.

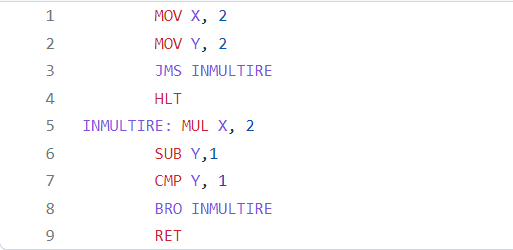
Limbajul HD poate descrie sistemul numeric avand in vedere portile logice si bistabilele componente, adica la nivel de porti. La un nivel mai ridicat, sistemul numeric poate fi descris in termenii transferurilor vectorilor de informatie intre registre.

Acesta reprezinta **Nivelul Transferurilor între Registre** (**Register Transfer Level - RTL**)/**Nivelul Fluxului de Date**. Limbajul **Verilog** suporta toate aceste niveluri.

2. Un **limbaj de asamblare** este un [limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) a [calculatoarelor](https://ro.wikipedia.org/wiki/Calculator) care, în locul codului mașină, folosește o desemnare simbolică a elementelor programului, relativ ușor de citit și interpretat de către om. Exemplu: limbajul numit Assembler de la compania [IBM](https://ro.wikipedia.org/wiki/IBM).

Fiecare tip de [procesor](https://ro.wikipedia.org/wiki/Microprocesor) deține propriul său limbaj numit "codul mașină", care reprezintă modul [binar](https://ro.wikipedia.org/wiki/Sistem_binar) de codificare a instrucțiunilor și [datelor](https://ro.wikipedia.org/wiki/Dat%C4%83) în [memorie](https://ro.wikipedia.org/wiki/Memorie), în forma direct executabilă. Instrucțiunile pot fi în general operații elementare (aritmetice, logice, transfer de date) sau și operații de comandă și control a procesorului. Codul mașină este, ca orice cod, format mai ales din cifre, fiind greu de reținut pentru un programator. De exemplu, un calculator ar putea folosi pentru instrucțiunile sale de adunare și scădere codurile mașină 5B și 5C. Însă, în limbajul de asamblare, aceste două operații ar putea fi simbolizate prin ADD și SUB, care sunt mult mai ușor de reținut și folosit.

Pentru a putea fi interpretate de procesor, programul scris de om ("[codul sursă](https://ro.wikipedia.org/wiki/Cod_surs%C4%83)", scris de exemplu în limbaj de asamblare) trebuie întâi redus prin compilare (sau asamblare sau interpretare) la "codul obiect" (în cod mașină), în acest scop fiind folosite "compilatoarele", "asambloarele" sau "interpretorii".

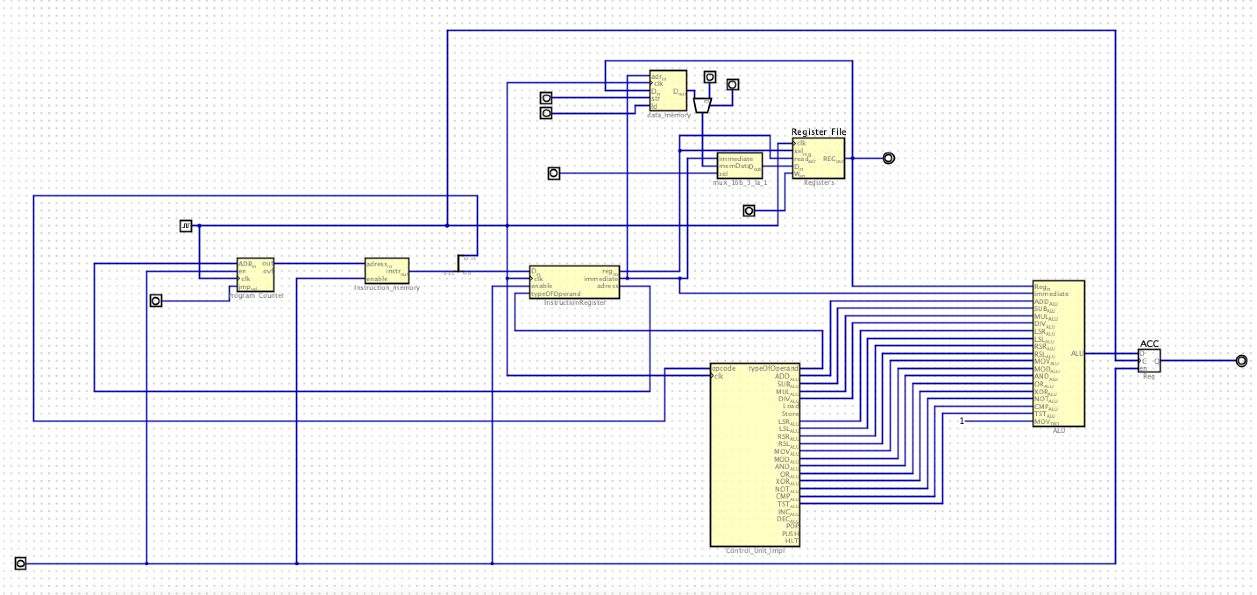


Exemplu din Proiect

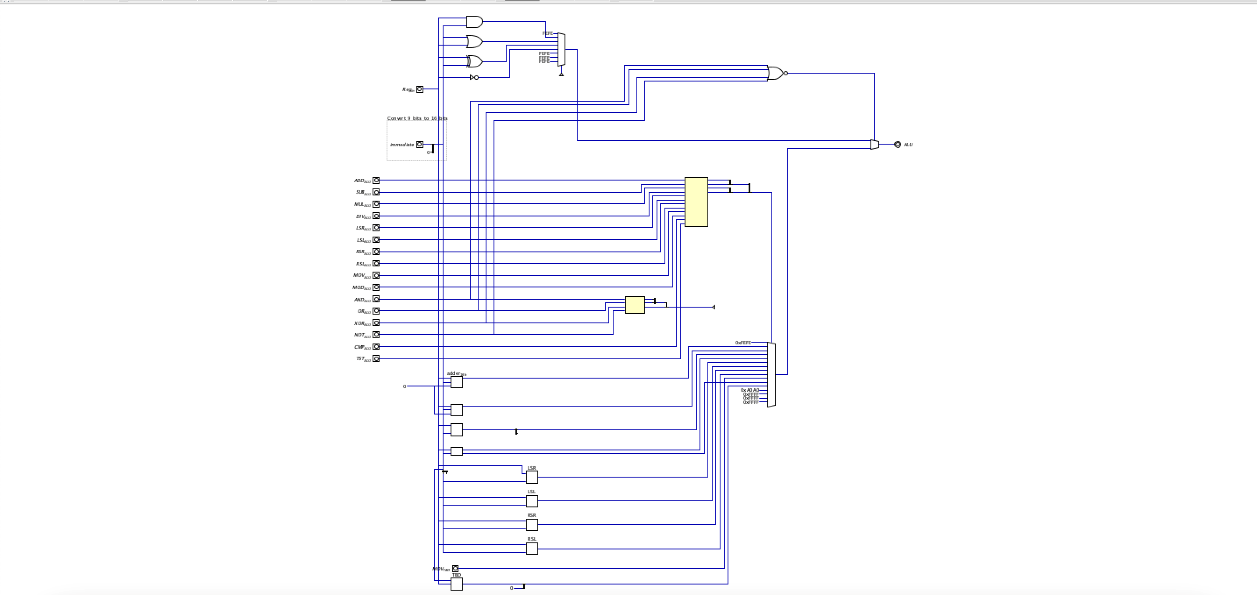
3. **Java** este un [limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) orientat-[obiect](https://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte), puternic tipizat, conceput de către James Gosling la [Sun Microsystems](https://ro.wikipedia.org/wiki/Sun_Microsystems) (acum filială [Oracle](https://ro.wikipedia.org/wiki/Oracle)) la începutul [anilor ʼ90](https://ro.wikipedia.org/wiki/Anii_1990), fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc.

Arhitectura Hardware

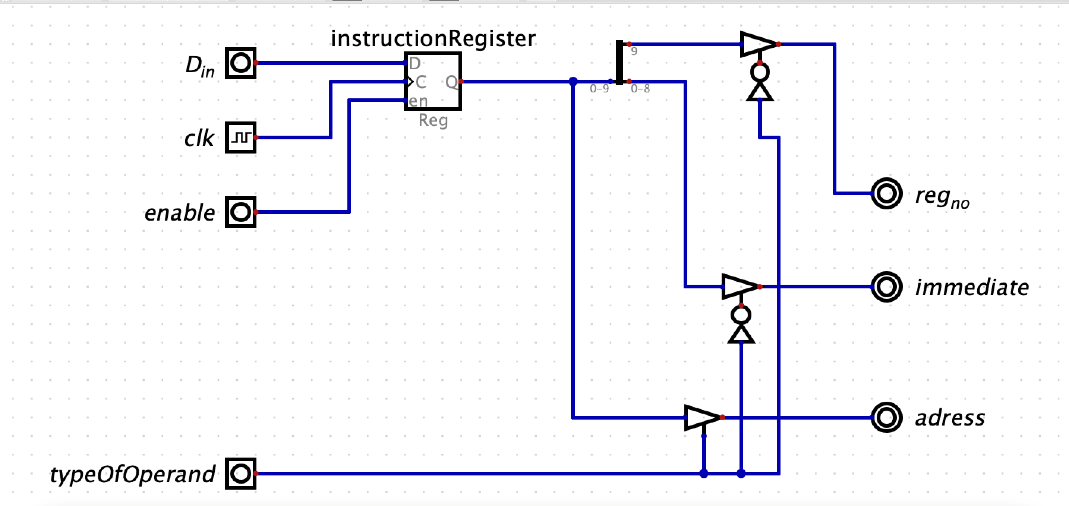
Arhitectura Generala



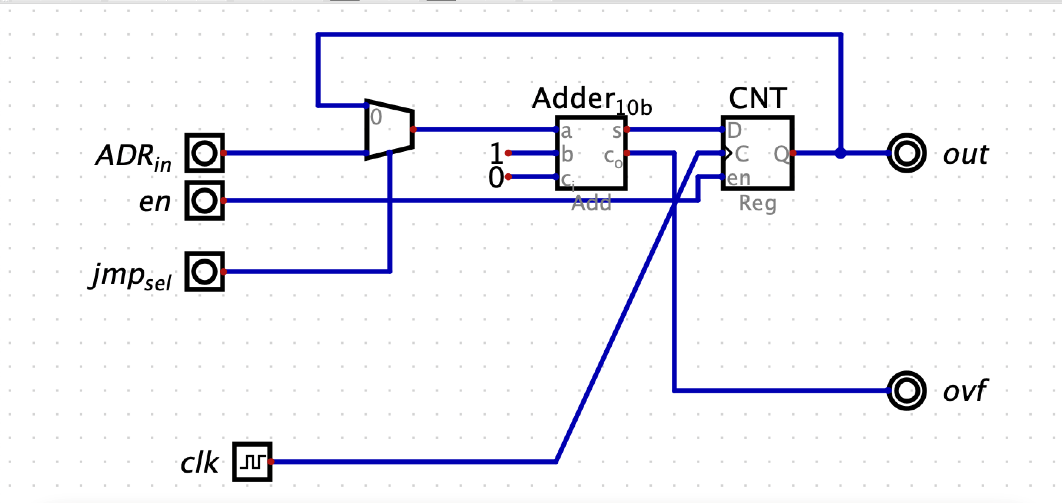
ALU



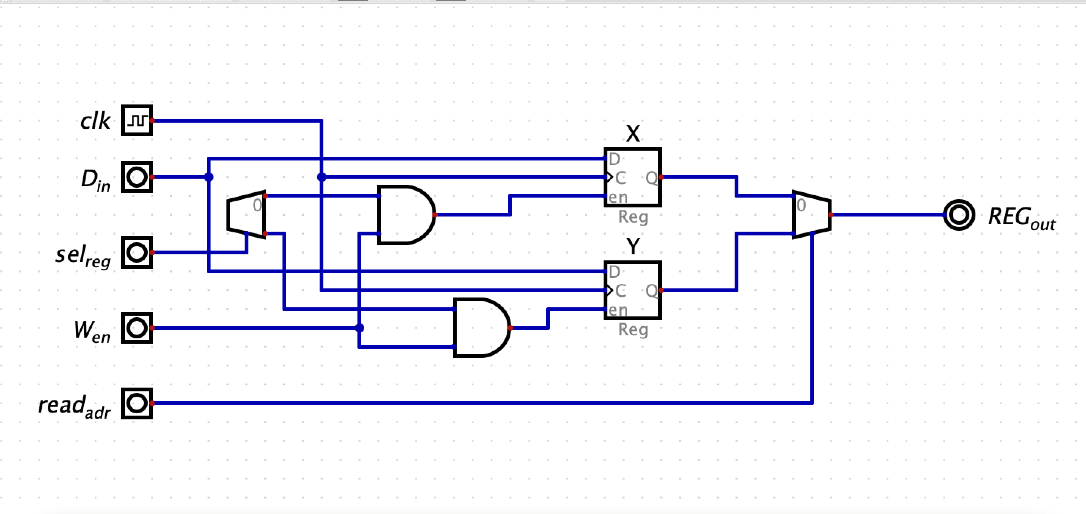
Instruction Register



Program Counter



Register File



Assembler

Un asamblator(Assembler) este un program care preia instrucțiuni de bază ale computerului și le convertește într-un model de biți pe care procesorul computerului îl poate folosi pentru a efectua operațiunile sale de bază.

//  
// Source code recreated from a .class file by IntelliJ IDEA  
// (powered by FernFlower decompiler)  
//  
  
import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Hashtable;  
  
public class Assembler {  
 private static File sourceFile;  
 private static File destinationFile;  
 private static Hashtable<String, Integer> labelList = new Hashtable();  
 private static Hashtable<String, Integer> instructionList = new Hashtable();  
 private static int currentLine = 0;  
 private static int codeLineCnt = 0;  
  
 public Assembler() {  
 }  
  
 public static void main(String[] var0) {  
 if (var0.length == 0) {  
 System.out.println("no filename specified");  
 System.exit(-1);  
 }  
  
 sourceFile = new File(var0[0]);  
 if (var0.length == 1) {  
 if (!var0[0].equalsIgnoreCase("-h") && !var0[0].equalsIgnoreCase("--help")) {  
 if (sourceFile.getParent() != null) {  
 destinationFile = new File(sourceFile.getParent().concat(File.separator).concat("assembled.prog"));  
 } else {  
 destinationFile = new File("assembled.prog");  
 }  
 } else {  
 System.out.println("Useage:");  
 System.out.println("java Aassembler [options]");  
 System.out.println("options:");  
 System.out.println("<source\_code\_path> <dest\_path>:");  
 System.out.println("\tAssemble source code to dest. For exaple .\\bin\\test1.asm .\\bin\\test1.prog");  
 System.out.println("<source\_code\_path>:");  
 System.out.println("\tAssemble source code to dest file a.prog");  
 System.out.println("-h(or --help):");  
 System.out.println("\tShow this help");  
 System.exit(0);  
 }  
 } else if (var0.length == 2) {  
 destinationFile = new File(var0[1]);  
 }  
  
 if (!destinationFile.exists()) {  
 try {  
 destinationFile.createNewFile();  
 } catch (IOException var2) {  
 System.out.println("Create machine language file error");  
 var2.printStackTrace();  
 }  
 }  
  
 initialInstructionList();  
 findLabels(sourceFile, destinationFile);  
 assembleFile(sourceFile, destinationFile);  
 }  
  
 private static void initialInstructionList() {  
 instructionList.put("HLT", 0);  
 instructionList.put("LOAD", 15);  
 instructionList.put("STORE", 1);  
 instructionList.put("BRZ", 2);  
 instructionList.put("BRN", 3);  
 instructionList.put("BRC", 4);  
 instructionList.put("BRO", 5);  
 instructionList.put("BRA", 6);  
 instructionList.put("JMP", 7);  
 instructionList.put("RET", 8);  
 instructionList.put("ADD", 41);  
 instructionList.put("SUB", 42);  
 instructionList.put("LSR", 43);  
 instructionList.put("LSL", 44);  
 instructionList.put("RSR", 45);  
 instructionList.put("RSL", 46);  
 instructionList.put("MOV", 47);  
 instructionList.put("MUL", 48);  
 instructionList.put("DIV", 49);  
 instructionList.put("MOD", 50);  
 instructionList.put("AND", 51);  
 instructionList.put("OR", 52);  
 instructionList.put("XOR", 53);  
 instructionList.put("NOT", 55);  
 instructionList.put("CMP", 56);  
 instructionList.put("TST", 57);  
 instructionList.put("INC", 58);  
 instructionList.put("DEC", 59);  
 instructionList.put("POP", 28);  
 instructionList.put("PSH", 29);  
 }  
  
 private static void findLabels(File var0, File var1) {  
 new String();  
 String[] var5 = new String[5];  
 currentLine = 0;  
 codeLineCnt = 0;  
 System.out.println("===== Step one: Finding Labels =====");  
  
 try {  
 BufferedReader var3 = new BufferedReader(new FileReader(var0));  
 BufferedWriter var4 = new BufferedWriter(new FileWriter(var1));  
 boolean var6 = false;  
  
 while(!var6) {  
 String var2 = var3.readLine();  
 if (var2 != null) {  
 ++currentLine;  
 ++codeLineCnt;  
 var2 = var2.toUpperCase();  
 var5 = parseLine(currentLine, var2);  
 if (var5[0] == null) {  
 --codeLineCnt;  
 }  
 } else {  
 var6 = true;  
 }  
 }  
  
 var3.close();  
 var4.close();  
 System.out.println("Labels found: ");  
 System.out.println(labelList);  
 } catch (FileNotFoundException var7) {  
 var7.printStackTrace();  
 } catch (IOException var8) {  
 var8.printStackTrace();  
 }  
  
 }  
  
 private static String[] parseLine(int var0, String var1) {  
 String[] var5 = new String[5];  
 String[] var2 = var1.split("[\\s,]");  
 int var4 = 0;  
 boolean var6 = false;  
  
 int var3;  
 for(var3 = 0; var3 < var2.length && !var2[var3].contains(";"); ++var3) {  
 if (var2[var3].length() > 0) {  
 if (var2[var3].contains(":")) {  
 String[] var7 = null;  
 if (!var6) {  
 var6 = true;  
 } else {  
 reportSyntaxError(var0, "found multi-labels");  
 }  
  
 var7 = var2[var3].split(":");  
 if (var7.length == 1) {  
 labelList.put(var7[0], codeLineCnt);  
 } else if (var7.length == 2) {  
 labelList.put(var7[0], codeLineCnt);  
 var5[var4++] = var7[1];  
 } else {  
 reportSyntaxError(var0, "error when parsing labels");  
 }  
 } else {  
 var5[var4++] = var2[var3];  
 }  
 }  
 }  
  
 System.out.print("parsed Line " + var0 + ": ");  
  
 for(var3 = 0; var3 < var4; ++var3) {  
 System.out.print(var5[var3] + " ");  
 }  
  
 System.out.println("");  
 return var5;  
 }  
  
 private static void reportSyntaxError(int var0, String var1) {  
 System.out.println("error at line:" + var0 + ", " + var1);  
 System.exit(-1);  
 }  
  
 private static void assembleFile(File var0, File var1) {  
 new String();  
 String[] var5 = new String[5];  
 currentLine = 0;  
 codeLineCnt = 0;  
 System.out.println("===== Step two: translate =====");  
  
 try {  
 BufferedReader var3 = new BufferedReader(new FileReader(var0));  
 BufferedWriter var4 = new BufferedWriter(new FileWriter(var1));  
 boolean var7 = false;  
  
 while(!var7) {  
 String var2 = var3.readLine();  
 if (var2 != null) {  
 ++currentLine;  
 ++codeLineCnt;  
 var2 = var2.toUpperCase();  
 var5 = parseLine(currentLine, var2);  
 if (var5[0] != null) {  
 String var6 = translateLine(currentLine, var5);  
 System.out.print("translatedLine: " + var6);  
 var4.write(var6);  
 var4.newLine();  
 System.out.println(" wrote");  
 } else {  
 --codeLineCnt;  
 System.out.println("Non-Code line, skiped");  
 }  
 } else {  
 var7 = true;  
 }  
 }  
  
 var3.close();  
 var4.close();  
 System.out.println("assemble complete ");  
 } catch (FileNotFoundException var8) {  
 var8.printStackTrace();  
 } catch (IOException var9) {  
 var9.printStackTrace();  
 }  
  
 }  
  
 private static String translateLine(int var0, String[] var1) {  
 String var2 = null;  
 String var3 = var1[0];  
 String var4 = var1[1];  
 String var5 = var1[2];  
 Integer var6 = (Integer)instructionList.get(var3);  
 if (var6 != null) {  
 String var10000;  
 String var7;  
 String var8;  
 if (verifyIfBranch(var3)) {  
 var7 = Integer.toBinaryString(var6);  
 var10000 = saturateWithZeros(var7.length(), 6);  
 var2 = var10000 + var7;  
 var8 = Integer.toBinaryString((Integer)labelList.get(var1[1]));  
 var2 = var2 + saturateWithZeros(var8.length(), 10) + var8;  
 } else {  
 var7 = Integer.toBinaryString(var6);  
 var10000 = saturateWithZeros(var7.length(), 6);  
 var2 = var10000 + var7;  
 var2 = var2 + getRegisterType(var4);  
 if (verifyImmediateValue(var5) && verifyShiftValue(var3, Integer.parseInt(var5))) {  
 var8 = Integer.toBinaryString(Integer.parseInt(var5));  
 var2 = var2 + saturateWithZeros(var8.length(), 9) + var8;  
 } else {  
 reportSyntaxError(var0, "Immediate Value out of bounds: " + var5);  
 }  
 }  
 } else {  
 reportSyntaxError(var0, "unkown Instruction:" + var3);  
 }  
  
 return var2;  
 }  
  
 private static String saturateWithZeros(Integer var0, Integer var1) {  
 String var2 = "";  
  
 for(Integer var3 = var1 - var0; var3 > 0; var3 = var3 - 1) {  
 var2 = var2 + "0";  
 }  
  
 return var2;  
 }  
  
 private static String getRegisterType(String var0) {  
 String var1 = "";  
 byte var3 = -1;  
 switch(var0.hashCode()) {  
 case 88:  
 if (var0.equals("X")) {  
 var3 = 0;  
 }  
 break;  
 case 89:  
 if (var0.equals("Y")) {  
 var3 = 1;  
 }  
 }  
  
 switch(var3) {  
 case 0:  
 var1 = "0";  
 break;  
 case 1:  
 var1 = "1";  
 }  
  
 return var1;  
 }  
  
 private static boolean verifyImmediateValue(String var0) {  
 boolean var1 = true;  
 Integer var2 = Integer.parseInt(var0);  
 if (var2 < 0 || var2 > 511) {  
 var1 = false;  
 }  
  
 return var1;  
 }  
  
 private static boolean verifyShiftValue(String var0, int var1) {  
 boolean var2 = true;  
 if ((var0.equals("LSR") || var0.equals("LSL") || var0.equals("RSR") || var0.equals("RSL")) && (var1 <= 0 || var1 >= 7)) {  
 var2 = false;  
 }  
  
 return var2;  
 }  
  
 private static boolean verifyIfBranch(String var0) {  
 boolean var1 = false;  
 if (var0.equals("BRZ") || var0.equals("BRN") || var0.equals("BRC") || var0.equals("BRO") || var0.equals("BRA") || var0.equals("JMP") || var0.equals("RET")) {  
 var1 = true;  
 }  
  
 return var1;  
 }  
}

Interfata grafica

