Technická dokumentácia

Táto technická dokumentácia sa vzťahuje k bakalárskej práci vypracovanej na STU FIIT s evidenčným číslom: FIIT-5212-92162

Táto technická dokumentácia obsahuje stručný popis niektorých tried v programe, ako aj zjednodušený diagram tried zdrojových súborov. Grafická časť programu je tvorená technológiou JavaFx, čiže je tvorená XML kódmi, táto časť tu nie je popísaná keďže tieto kódy boli generované v programe SceneBuilder.

Spustenie aplikácie

Na spustenie aplikácie je potrebné mať nainštalovanú Java JDK 11+ (ideálne Java 11 corretto-11 Amazon Corretto version 11.0.14). Všetok ostatný postup je popísaný v README.txt textovom súbore.

Spustenie zdrojových kódov

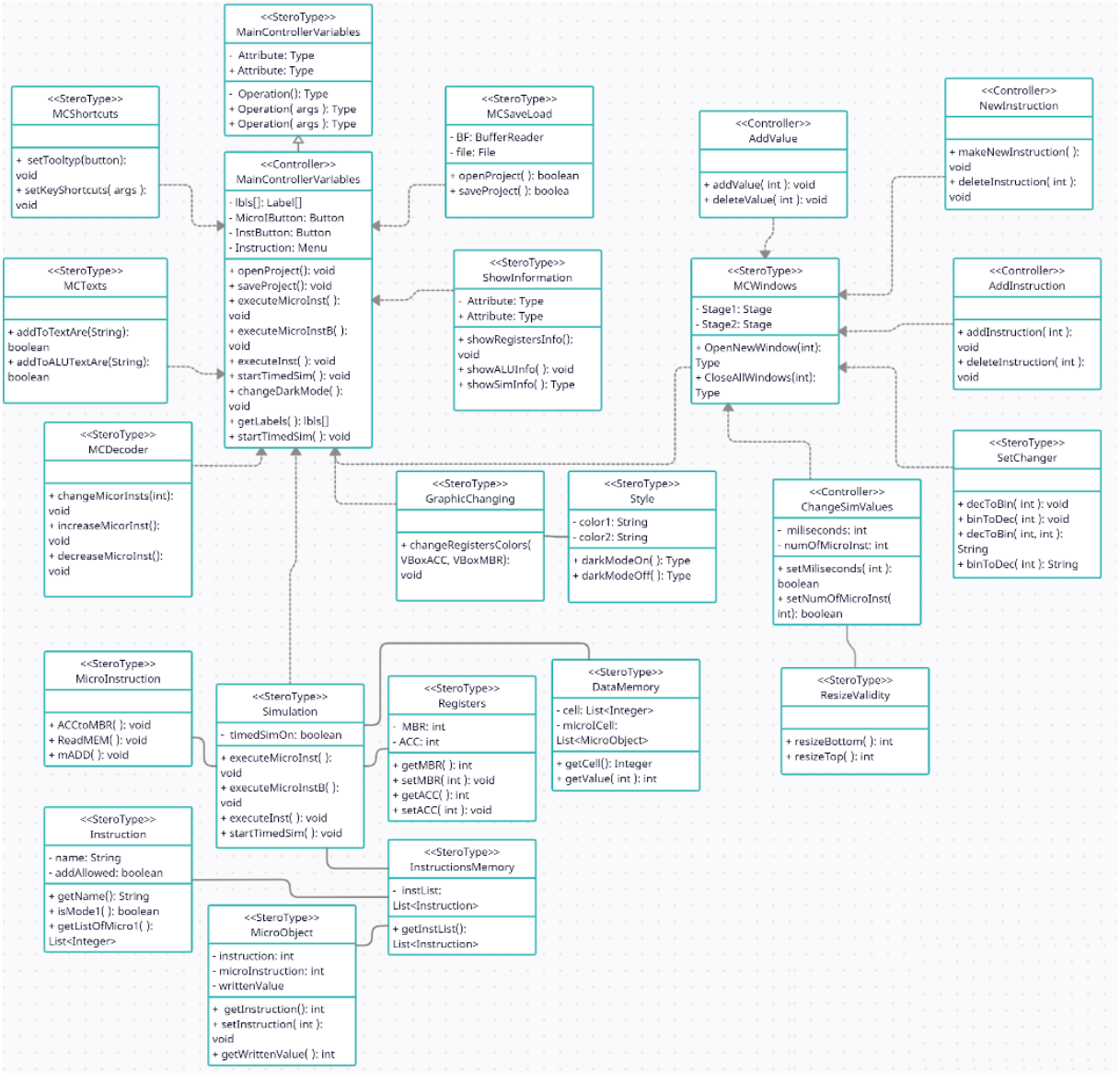
Na spustenie aplikácie je potrebné mať nainštalovanú Java JDK 11+ (ideálne Java 11 corretto-11 Amazon Corretto version 11.0.14). Všetok ostatný postup je popísaný v README.txt textovom súbore.

V tomto súbore sa nachádza tento text:

Na spustenie aplikácie je potrebné mať nainštalovanú Java JDK 11+ (ideálne Java 11 corretto-11 Amazon Corretto version 11.0.14). Taktiež je potrebné do projektu vložiť externú knižnicu “javafx-sdk-11.0.2\lib” a do VM options vložiť tento script “--module-path "..\javafx-sdk-11.0.2\lib" --add-modules javafx.controls,javafx.fxml,richtextfx.fat”. Zdrojové kódy je následne možné spustiť metódou main v triede Main.java.

Popis zdrojových kódov

Nasledujúci obrázok zobrazuje zjednodušený diagram tried zdrojových súborov RegSimu.



Nasleduje popis účelu významných tried. Popis niektorých tried obsahuje aj ukážku kódu.

Trieda: Main.java

Účel triedy je spustenie spustenie hlavného okna o velkosti 900px x 600px. Veľkosť hlavného okna sa dá ďalej meniť.

Trieda: Controller.java

Táto trieda slúži ako controller hlavného okna, je to jedna z najpodstatnejších tried. Neobsahuje takmer žiadnu funkcionalitu, len využíva funkcionalitu iných tried. Obsahuje funkcie ktoré sa spúšťajú pri kliknutí na tlačidlá v hlavnom okne a taktiež funkcie ktoré sa spúšťajú v položkách hlavného menu. Tieto funkcie však v jednom, či dvoch riadkoch len spustia metódy v iných triedach.

Ukážka kódu: Prvá metóda sa spúšťa pri vybratí položky “Inštrukcia >> Vytvoriť” z hlavného menu a druhá metóda sa spúšťa pri vybratí položky “Pamäť >>Zmena velkosti pamäte” z hlavného menu.

```
public void MakeNewInstruction(){
   mcWindows.MakeNewInstruction();
}

public void ChangeSizeMenu(){
   mcWindows.ChangeSizeMenu();
}
```

Trieda: ProjectSaveOpen.java

Táto trieda slúži na ukladanie a otváranie súborov. Obsahujú všetku funkcionalitu ohľadom otvárania a ukladania súborov.

Trieda: MicroInstruction.java

Táto trieda obsahuje funkcionalitu a názvy všetkých mikroinštrukcií. Táto trieda je veľmi rozsiahla ale je pomerne jednoduchá. Obsahuje údaje o všetkých mikroinštrukciách a preto je taká rozsiahla. O každej mikroinštrukcií uchováva jej názov, číslo a aj funkcionalitu.

Ukážka kódu: Funkcionalita prvých troch mikroinštrukcií.

```
//1
public void ACCtoMBR(){
   registers.setMBR(registers.getACC());
}

//2
public void MBRtoACC(){
   registers.setACC(registers.getMBR());
}

//3
public void PCtoMAR(){
   registers.setMAR(registers.getPC());
}
```

Trieda: Instruction.java

Trieda Instruction obsahuje premenné a zoznamy ktoré využívajú inštrukcie. Každá inštrukcia musí mať meno, povolený alebo zakázaný adresovací mód, povolený jeden až tri konkrétne adresovacie módy a taktiež musí obsahovať nejaké mikroinštrukcie aspoň v jednom z troch zoznamov mikroinštrukcií. Tieto zoznamy sú tri pretože aj adresovacie módy sú tri, takže pre každý je jeden zoznam. Objekty tejto triedy sú jednotlivé inštrukcie.

Ukážka kódu: Parametre ktoré musí obsahovať každá inštrukcia.

```
private String name;
private boolean mode1 = false;
private boolean mode2 = false;
private boolean mode3 = false;
```

```
private int instructionNumber;
private boolean addAllowed = true;
private List<Integer> listOfMicro1 = new ArrayList<>();
private List<Integer> listOfMicro2 = new ArrayList<>();
private List<Integer> listOfMicro3 = new ArrayList<>();
```

Trieda: GenerateInstructions.java

Táto trieda obsahuje zoznam všetkých vytvorených inštrukcií.

Trieda: DataMemory.java

Táto trieda obsahuje hodnoty uložené v pamäti.

Trieda: AddInstruction.java

Táto trieda zabezpečuje funkcionalitu okna na pridávanie inštrukcií do pamäte.

Trieda: AddValue.java

Táto trieda zabezpečuje funkcionalitu okna na pridávanie číselných údajov do pamäte.

Trieda ChangeSize.java

Trieda slúži na zmenu veľkosti pamäte.

Trieda: CheckValidity.java

Účelom tejto triedy je kontrola toho či sa môže zmeniť veľkosť pamäte na požadovanú veľkosť, trieda kontroluje inštrukcie v pamäti odkiaľ a pokiaľ ich akcie zasahujú v pamäti inštrukcií a pamäti údajov a podľa toho umožňuje alebo neumožňuje zmenu veľkosti pamäte. Jej metódy obsahujú pomerne zložité cykly.

Trieda: AddMiliseconds.java

Táto trieda zabezpečuje zmenu počtu milisekúnd, ktoré sa čaká medzi vykonávaním mikroinštrukcií pri časovanej simulácií. Taktiež zabezpečuje zmenu krokov vykonávaných pri rýchlej simulácií.

Ukážka kódu: Metóda na nastavovanie štýlu a textu hlásenia o korektnosti hodnoty na ktorú sa pokúša užívateľ zmeniť počet milisekúnd v časovanej simulácií.

```
private void setMillisecondsValidity(boolean b, String s){
   button1.setDisable(b);
   labelWarning1.setText(s);
   if(b) labelWarning1.setStyle(style.setRedLabel());
   else labelWarning1.setStyle(style.setGreenLabel());
}
```

Trieda: SimulationEnd.java

Táto trieda je controller okna ktoré sa zobrazí na konci simulácie. V prípade ak sa simulácia ukončí mikroinštrukciou HALT alebo užívateľom, tak sa zobrazí hlásenie o úspešnom ukončení simulácie. V prípade ukončenia simulácie z iného dôvodu sa zobrazí hlásenie o neúspešnom ukončení simulácie spolu s dôvodom neúspešného ukončenia.

Ukážka kódu: Inicializačná metóda, ktorá najskôr nastaví štýl a následne text hlásenia o úspešnom alebo neúspešnom ukončení simulácie.

```
public void initialize(){
    SimulationData simulationData = SimulationData.getSimulationData();
    int reasonOfEnd = simulationData.getReasonOfEnd();

    hasBeenInitialized = true;
    bindStatic();
    changeDarkModeOnStatic();

    Style style = new Style();

    if (reasonOfEnd == 0){
        label1.setText("Simulácia skončila\núspešne.");
        label1.setStyle(style.setGreenLabel());

        mcSimulationConsole.endSimulationSuccesfull();
    }
```

Trieda: Simulation.java

Táto trieda obsahuje funkcionality ktoré slúžia na samotné simulovanie. Napríklad na ukončenie alebo pozastavenie simulácie, či na vykonávanie jednotlivých mikroinštrukcií.

Ukážka kódu: Metóda s ktorou sa vykonáva mikroinštrukcia a metóda s ktorou sa vykonáva inštrukcia.

```
public void executeMicroInstruction(){
    step.executeMicroInstruction();
}

public void executeInstruction(){
    do{
        step.executeMicroInstruction();
    }

while(simulationMemory.getMicroICell().get(simulationMemory.getCurrentIndex()).getMicroInstructionI
ndex() != 0);
}
```

Trieda: SimulationData.java

Táto trieda obsahuje dáta ohľadom simulácie, ktoré nevyhnutné pri samotnom simulovaní. Napríklad je tu uložená hodnota ktorá uchováva počet milisekúnd ktoré sa má čakať medzi vykonávaním jednotlivých krokov v časovanej simulácií, či počet krokov ktoré sa majú vykonať pri rýchlej simulácií.

Ukážka kódu: Jednotlivé premenné ktoré obsahujú dáta ohľadom simulácie.

```
private boolean running = false;
private Timeline timelineA = new Timeline();
private int milliseconds = 400;
private int numberOfFastSimSteps = 2000;
private int instructionActive = 0;
private int valueActive = 2047;
private boolean simulationMemoryWasNotDecreased = true;
private boolean hasBeenStopped = false;
private int reasonOfEnd = 0;
private boolean aluOn = false;
```

Trieda: Step.java

Táto trieda je veľmi podstatná, pretože slúži na vykonávanie mikroinštrukcie. Z tejto triedy sa najmä spúšťajú funkcionality iných tried v ktorých sa napríklad kontrolujú, alebo nastavujú dáta ktoré sa momentálne nachádzajú v jednotlivých registroch.

Ukážka kódu: Metóda na nastavenie premenných ktoré slúžia na upresnenie nasledujúcej mikroinštrukcie ktorá sa má vykonať. Spodná metóda posiela číslo reprezentujúce mikroinštrukciu ktorá sa má vykonať na vykonanie do objektu triedy Microinstruction.java.

```
private void settingVariables(){
   currentIndex = simulationMemory.getCurrentIndex();  //currentIndex = current index in
dataMemoryField (for example 1500);
   currentMicroInst = simulationMemory.getMicroICell().get(currentIndex).getMicroInstruction();
//currentMicroInst = real number of microInstruction (for example 1 = ACCtoMBR)
}

private void execution(){  //real execution of microinstruction
   microInstruction.executeTheMicroInstruction(currentMicroInst);
}
```

Trieda: StepObject.java

Táto trieda uchováva dáta o stave simulácie po jednotlivých mikroinštrukciách. Tieto dáta sa využívajú pri spätnej simulácií.

Ukážka kódu: Premenné ktoré obsahujú dáta o hodnotách v registroch a o indexoch konkrétnej mikroinštrukcie.

```
public class StepObject {
   private int MBR;
   private int ACC;
   private int MAR;
   private int IR;
   private int PC;
   private int SR;
   private int microInstruction;
   private int microInstructionIndex;
```

Trieda: DataMemory.java

Táto trieda je veľmi dôležitá, pretože uchováva dáta o stave pamäte v simulácií. O každej bunke vie či je prázdna, či obsahuje bod prerušenia a najmä to, akú hodnotu obsahuje.

Ukážka kódu: Listové zoznamy uchovávajúce dáta o pamäti.

```
public class DataMemory {
   private static List<Integer> cell = new ArrayList<>(); //the memory
   private static List<Boolean> cellIsFull = new ArrayList<>(); //the list of cells if they are empty
   private static List<Boolean> cellBreakPoint = new ArrayList<>(); //the list of cells if they are empty
```

Trieda: TextCode.java

Táto trieda slúži na konvertovanie obsahu pamäte na text a naopak na konvertovanie textu na obsah pamäte.

Ukážka kódu: Metóda ktorá zapíše dáta z pamäte číselných údajov ako text do textového poľa ktoré reprezentuje pamäť údajov.

```
public void writeMemFromMemory() {
   codeAreaMem.clear();

   String str = new String();
   for (int i = dataMemory.getLengthOfInstructionData(); i < 2048; i++) {
      if (dataMemory.getIsFull().get(i)) {
         if (i == 2047) str = str.concat(diffSet.change(dataMemory.getCell().get(i)) + "");
         else str = str.concat(diffSet.change(dataMemory.getCell().get(i)) + "\n");
      } else {
         if (i == 2047) str = str.concat("");
         else str = str.concat("\n");
      }
   }
   codeAreaMem.replaceText(0,0,str);
}
```

Trieda: MCgraphicChanging.java

Táto trieda je grafická, pracuje s grafikou procesora. Po každej mikroinštrukcií nasáva zmena grafiky procesora, táto trieda sa stará o to aby táto zmena nastala.

Trieda: Style.java

Táto trieda je grafická, pracuje s grafikou celej aplikácie. Slúži na nastavovanie jednotlivých častí používateľského rozhrania do temného módu alebo do normálneho módu. Táto trieda obsahuje mnoho metód. Tieto metódy slúžia na nastavovanie rôznych typov grafických objektov.

Ukážka kódu: Metóda ktorá sa spustí pri zapnutí alebo vypnutí temného módu.

```
public void setDarkModeON(boolean darkModeON) {
   this.darkModeON = darkModeON;
   if(darkModeON){
```

```
        BackgroundColor = darkBackgroundColor;
        ActiveColor = darkActiveColor;
        TextColor = darkTextColor;
        B2 = DB2;
        BreakPointColor = darkBreakPointColor;
    }
    else {
        BackgroundColor = lightBackgroundColor;
        ActiveColor = lightActiveColor;
        TextColor = lightTextColor;
        B2 = LB2;
        BreakPointColor = lightBreakPointColor;
    }
}
```

Triedy ktoré sa začínajú dvoma veľkými písmenami “MC” menia grafické zobrazenie a štýly častí hlavného okna simulátora. Ako príklad je tu bližšie popísaná trieda MCRegisters.java

Trieda: MCRegisters.java

Táto trieda je grafická, pracuje s obsahom registrov zobrazených v grafickom rozhraní. Práve kvôli tejto triede je možné vidieť aktuálny obsah registrov po každom kroku simulácie.

Ukážka kódu: Metóda ktorá spúšťa zmenenie obsahu textových polí všetkých registrov.

```
public void setTextRegisters(){
    setLabelTexts(labelMBR1,labelMBR2,labelMBR3,registers.getMBR());
    setLabelTexts(labelACC1,labelACC2,labelACC3,registers.getACC());
    setLabelTexts(labelMAR1,labelMAR2,labelMAR3,registers.getMAR());
    setLabelTexts(labelIR1,labelIR2,labelIR3,registers.getIR());
    setLabelTexts(labelPC1,labelPC2,labelPC3,registers.getPC());
    setStatusLabelTexts(labelSR3,registers.getSR());
}
```