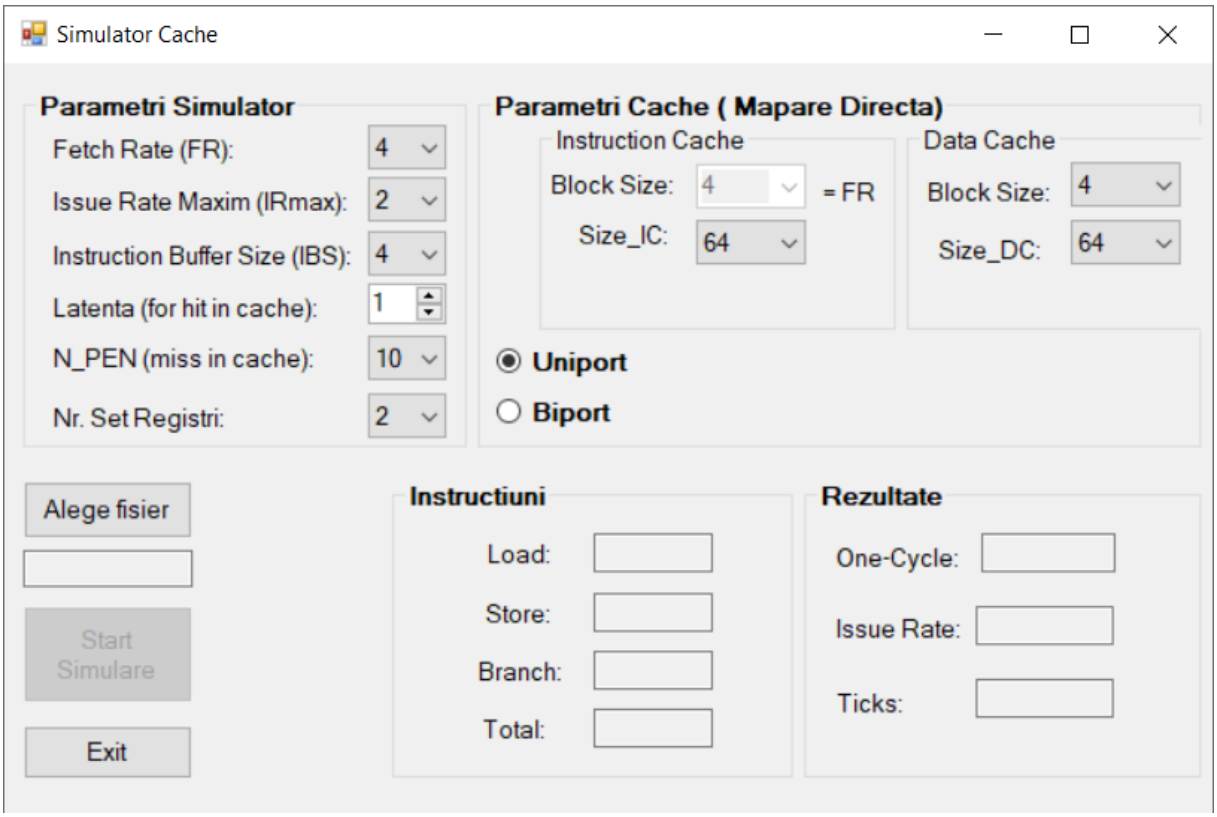

Am incercat sa va sintetizez workflow-ul aplicatiei, sub forma de pseudocod.

E doar un pseudocod din care sper sa intelegeti cam ce si cum se vrea.

Nu e obligatoriu sa respectati structura acestui pseudocod. Va puteti aranja codul cum doriti.

Ideea e doar sa intelegeti cam ce se vrea, din acest pseudocod.

Pseudocod:

//Initializari – cele puse de mine aici, plus parametrii pe care ii colectati de pe interfata

//clasa care defineste cum “arata” o instructiune

```
public class Instructiune
{
        public char TipInstructiune ;
        public int PcCurent ;
        public int Target ;
};
TICKS = 0 //ciclii totali, necesari executiei programului
nrInstrProcesate = 0
nrBranchProcesate = 0
```

```
nrStoreProcesate = 0
nrLoadProcesate = 0
nrArithProcesate = 0
if (TipCache == unificat) //asta e parametrul ala de pe interfata Uniport/Biport
{
	acceseDisponibileLaMemoriePerCiclu = 1;
}
else
{
	acceseDisponibileLaMemoriePerCiclu = 2;
}
//In vectorul urmator, introduceti intructiunile din benchmark
Instructiune[] instructiuniCititeDinBenchmark = new Instructiune[];
Citire_instructiuni_benchmark();
//in matricea urmatoare, veti procesa instructiuni pe ideea pe care v-am zis-o la laborator si
//care e descrisa sub forma de pseudocod, in continuare
//de exemplu, daca IRMaxDePeInterfata = 2, atunci vezi avea o matrice de 1000000 linii si 2
//coloane, unde practic, pe fiecare linie veti avea 2 instructiuni
Instructiune[ , ] instructiuniAduseDinMemorie = new Instructiune[1000000,
IRMaxDePeInterfata];
PCnormal = 0 // initializare
foreach(Instructiune instructiune in instructiuniCititeDinBenchmark)
{
	//completati cu instructiuni arithmetic-logice, unde si cat e necesar, vezi while-ul de
	//mai jos
	while(instructiune.PCcurent != PCnormal)
	{
		AdaugaOInstructiuneALUInMatricea("instructiuniAduseDinMemorie");
		PcNormal++;
		nrArithProcesate++;
	}
```

```
        if(instructiune.TipInstructiune == “B”)
        {
                PcNormal = instructiune.Target; //faceti saltul, practic, aici
                AdaugaOInstructiuneBInMatricea(“instructiuniAduseDinMemorie”);
                nrBranchProcesate++;
        }
        if(instructiune.TipInstructiune == “S”)
        {
                AdaugaOInstructiuneSInMatricea(“instructiuniAduseDinMemorie”);
                PcNormal++;
                nrStoreProcesate++;
        }

        if(instructiune.TipInstructiune == “L”)
        {
                AdaugaOInstructiuneLInMatricea(“instructiuniAduseDinMemorie”);
                PcNormal++;
                nrLoadProcesate++;
        }
}

//Practic, aici ati terminat de “executat” instructiunile
//Acum, mai trebuie doar calculate statisticile
TICKS = cate_linii_a_cate_2_instructiuni_ati_obtinut_in_matricea_
instructiuniAduseDinMemorie * latenta; //latenta e parametru de pe inerfata
deCateOriAFostMemoriaDejaAccesata = 0 //initializare
foreach(Instructiune instructiune in instructiuniAduseDinMemorie)
{
        if(deCateOriAFostMemoriaDejaAccesata < acceseDisponibileLaMemoriePerCiclu)
        {
```

```
            if(instructiune == “Load” SAU instructiune == “Store”)
            {
                deCateOriAFostMemoriaDejaAccesata++;
            }
        }
        Else
        {
            if(instructiune == “Load” SAU instructiune == “Store”)
            {
                deCateOriAFostMemoriaDejaAccesata = 0 //resetare
                TICK += latena // parametru de pe interfata
            }

        }
}
```

//!!! Atentie unde anume resetati “deCateOriAFostMemoriaDejaAccesata”, in codul vostru, //in for-ul de mai sus

penalizareMissCache = nrLoadProcesate * cacheMiss * N_PEN_interfata;

//cacheMiss = 0.1, il setam noi, default

TICKS += penalizareMissCache;

Acum, ar mai trebui calculate numarul total de intructiuni procesate, si IR-ul.

Teoretic, aveti toate acum toate datele pentru a le calcula (sunt doar niste adunari si impartiri).

Daca nu va e clar, sau nu va dati seama, sa ma intrebati.