

SIMULATOR GRAFIC MEMORIE CACHE

IMPLEMENTATA PRIN TEHNICA ASOCIATIVA

Facultatea de Automatica si Calculatoare
Specialitate: Calculatoare si Tehnologia Informatiei
Student: Tincu Diana
Grupa: 30231

1. Cerinte Proiect:

Proiectul 4:

Simulator grafic pentru reprezentarea modului de functionare a unei memorii cache implementate prin tehnica asociativa.

Cerinte:

- Se va implementa un interpretor simplu de instructiuni (asamblare, 8-10 instructiuni) care va permite simularea executiei unei secvente de program
- Se vor reprezenta grafic liniile de memorie cache, memoria principala, unitatea de control si conexiunile dintre acestea
- In timpul simularii (pas-cu-pas) se vor indica elementele valorice care sunt generate pe magistrale, respectiv cele pastrate in memoria asociativa (tabela de evidenta)
- Scriere prin write-back

Limbajul folosit: Java

Voi crea o clasa pentru memoria cache, memoria principala, unitatea de control, instructionFetch.

Voi creea interpretorul de instructiuni tot in Java.

Instructiunile pentru interpretor vor fi:

add, sub, div, load, store.

Voi simula pentru fiecare instructiune si voi vedea ce se intampla in memoria cache, memoria principala si cu unitatea de control voi codifica instructiunile. Voi calcula de fiecare data miss rate-urile si hit-urile si voi vedea cum variaza acestea in functie de dimensiunea memoriei cache.

Voi avea un contor pt fiecare linie din cache adusa din memorie care o sa aiba atunci cand e adusa din memorie o sa aiba o valoare maxima (eu am pus nr de linii de cache), si se decrementeaza toate liniile cu exceptia celei folosite de fiecare data cand e utilizata. Cand memoria cache e plina scoatem/ inlocuim linia din mem cache care are contorul 0(sau cel mai mic daca este cazul).

2. Tehnici de scriere in memoria cache:

"Write-Back": se scrie si se citeste doar in/din mem. cache iar in caz de pagina negasita se realizeaza salvarea paginii de memorie cache care se inlocuieste in memoria principala (de unde a fost adusa) apoi pagina de memorie principala ceruta de procesor va fi scrisa in locul paginii de cache salvate. Daca nu se cunoaste faptul ca pagina de cache care se inlocuieste cu noua pagina a fost sau nu modificata, atunci se salveaza pagina de cache intotdeauna; daca se foloseste un bit de "modify" (sau "dirty") care este setat la prima scriere a paginii de cache, atunci se testeaza acest bit si pagina de cache este salvata in memoria principala numai daca pagina a fost modificata. Transferarea blocurilor de memorie se realizeaza ori de cate ori apare un nou acces de pagina cache negasita ("cache fault page").

- write-back: Când procesorul emite o cerere de scriere, un cache write-back actualizează datele din cache, însă nu actualizează imediat și pe cele din memorie. o Avantaj: prin acest mecanism se elimină traficul inutil dintre cache și memoria principală. o Dezavantaje: duce la creșterea complexității hardware-ului, fiind necesară reținerea (marcarea) acelor linii care au devenit inconsistente cu memoria principală. Pentru a marca aceste linii (murdare), se include un indicator suplimentar pe fiecare linie. În momentul evacuării, dacă linia este murdară va fi scrisă în memoria principală. În cazul sistemelor multicore în care avem cache-uri individuale pentru fiecare core/ procesor, care partajează memoria principală, păstrarea consistenței devine mai dificilă, necesitând mecanisme și hardware și mai complex (discutat în secțiunea 6).

3. Tehnica de mapare a memoriei cache folosita : Maparea asociativa

Pentru simplitatea si claritatea codului dar mai ales pentru eficienta implementarii am ales ca tehnica de mapare maparea asociativa.

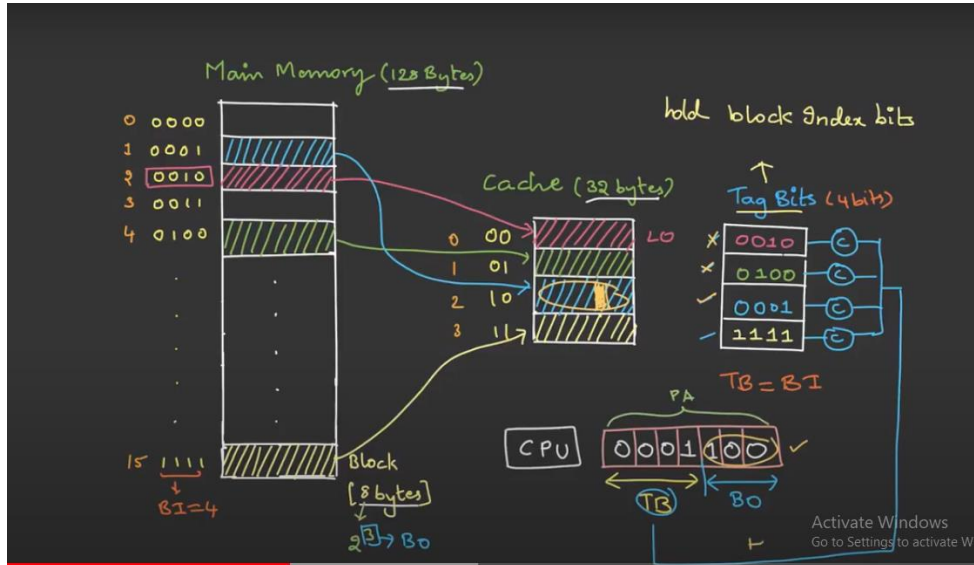
Exista o tabela asociativa TA (bazata pe o memorie cu cautare asociativa, dupa continut), asociata memoriei cache (MC), care pastreaza pentru fiecare bloc de cache o eticheta reprezentand adresa blocului din memoria principala (ABLP-ul) stocat in blocul de memorie cache. Astfel ca la fiecare acces la memorie, se va cauta in TA, ABLP-ul cerut si daca exista o eticheta cu valoarea ABLP, se va furniza adresa blocului de cache ce rezulta din pozitia etichetei respective in TA. Daca nu exista, atunci se va inlocui un bloc din MC conform unui algoritm de inlocuire. In cazul de fata este vorba de algoritmul LRU ("Least Recently Used").

Cautarea unui bloc in TA se poate face eficient in hardware avand in vedere ca putem face in paralel compararile care sunt independente.

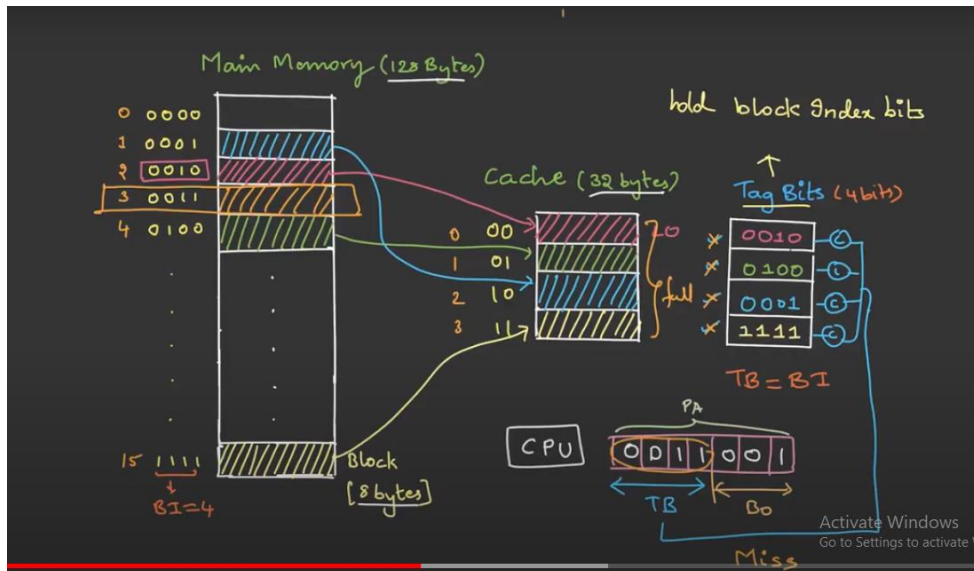
Algoritmul LRU foloseste o tabela de contori de utilizare a blocurilor de memorie cache: La fiecare acces la un bloc de memorie cache, vom seta contorul corespunzator aceluia bloc la valoarea maxima si vom decrementa ceilalti contori diferiti de 0. Avand in vedere ca valoarea maxima pe care o poate avea un contor este egala cu numarul de blocuri de memorie cache , intotdeauna vom avea cel putin un 0 pe o pozitie din tabela de contori. Pozitia pe care se va gasi primul 0 ne va furniza numarul blocului de cache care va fi inlocuit.

Implementarea algoritmului LRU se poate implementa eficient in hardware prin accesarea in paralel a locatiilor tabelului de indcsci LRU si prin folosirea unui codificator ce furnizeaza la iesire adresa primului zero.

Hit:



Miss:



Se pune 0011 (pt ca nu e in cache se cauta in main memory) pe o pozitie din mem cache (care se alege folosind un algoritm)

4. Implementare:

Pentru inceput o sa imi creez urmatoarele clase:

- **MainMemory** - pentru memoria principala
- **Cache, CacheMemory** - pentru memoria cache
- **InstructionFetch** - pentru decodificarea instructiunilor (va fi un fel de interpretor de instructiuni in assembly)
- **CacheSim** - pentru simularea modului de functionare a memoriei cache
- **Block** – un bloc va continue un set de instructiuni
- **InstructionsSet** – pentru a pune setul current de instructiuni in memorie

Clasele folosite:

MainMemory:

Aceasta clasa continue campurile:

```
String mainMemorySize;  
String tagBits;  
String indexBits;  
int nrOfLines;  
private int nbOfAddressBits;
```

Tot aici am mai creat o metoda care imi calculeaza numarul de linii din memoria principala in functie de dimensiunea memoriei pe care o setam.

Eu am calculat ca numarul de linii din memories a fie egal cu dimensiunea memoriei.

InstructionFetch:

Aceasta clasa continue campurile:

```
String opcode;  
String rs;  
String rt;  
String rd;  
String immediate;  
  
String[] setOfInstructions;
```

Aici am create o metoda pentru decodificarea instructiunilor:

Am assignat fiecarei operatii un anumit cod si am setat valorile registrilor in functie de instructiunea curenta.

Opcode-urile pentru instructiunile mele sunt:

Add: "0000";

Sub: "0000";

Addi: "0010";

Subi: "0010";

Lw: "0100"; --load

Sw: "0100"; --store

J: "0110"; --jump

InstructionSet:

In aceasta clasa avem campurile:

```
static String[] firstSetOfInstructions;  
static String[] secondSetOfInstructions;  
static String[] thirdSetOfInstructions;  
String[] currentSet;
```

Aici am creat trei seturi de instructiuni pe care le-am setat si le-am adaugat cateva instructiuni.

-> currentSet este contorul care ne arata la ce set de instructiuni suntem, asadar daca:

-currentSet=0 ne aflam la primul set de instructiuni;

-currentSet=1 ne aflam la al doilea set de instructiuni;

-currentSet=2 ne aflam la al treilea set de instructiuni;

Cache:

In aceasta clasa avem campurile:

```
String cacheSize;  
int nrLinesCache;  
int nrInstructions;  
int totalCacheAccesses=0;  
String[] cacheInstructions;  
int[] contorInstr;  
String[] instrAddresses;  
int hit=0;  
int oldHit=0;  
int hitIndex;  
int miss=0;  
int missIndex;
```

- cacheSize: reprezinta dimensiunea memoriei cache
- nrLinesCache: reprezinta numarul de linii din cache si in cazul meu este egal cu dimensiunea memoriei cache (cacheSize)
- totalCacheAccesses: reprezinta numarul total de acces al memoriei cache indiferent daca avem hit sau miss
- cacheInstructions: reprezinta instructiunile din cache
- contorInstr: este un contor pe care il folosesc atunci cand am memoria cache plina si vreau sa imi aduc in cache o instructiune din memoria principala care nu este deja in cache. Acesta se initializeaza pt fiecare instructiune cu numarul de linii din cache, iar de fiecare data cand o instructiune face hit in memoria cache decrementam contorul pentru toate celelalte instructiuni cu exceptia celei care a facut hit. Daca instructiunea face miss atunci o adaugam in memoria cache si ii setam contorul ca fiind egal cu dimensiunea memoriei cache.
- instrAddresses: reprezinta adresele instructiunilor din cache (il folosesc atunci cand imi caut pozitia din cache la care sa adaug o instructiune)
- hit: reprezinta un contor care se incrementeaza de fiecare data cand avem un HIT in memoria cache. L-am folosit pentru a calcula HitRate-ul
- miss: reprezinta un contor care se incrementeaza de fiecare data cand avem un MISS in memoria cache. L-am folosit pentru a calcula MissRate-ul
- hitIndex si missIndex: le-am folosit pentru a colora liniile de memorie in cazul in care avem hit sau miss. Pentru hit am folosit culoarea verde iar pentru miss am folosit culoarea rosie.

Tot aici am creat o metoda folosind algoritmul LRU pentru a gasi index-ul la care putem adauga instructiunea noua in cazul in care aceasta nu se gaseste in cache si cache-ul este plin.

```
int getIndexOfInstructionToSubstitute(){
    //int min =contorInstr[0];
    int indexMin= 0;
    for(int i=0; i < cacheInstructions.length; i++) {
        if (cacheInstructions[i] == null) {
            return i;
        }
    }

    for(int i =0; i<contorInstr.length;i++){
        if(contorInstr[i]==0)
        {
            indexMin=i;
            //min=contorInstr[i];
        }
    }
    return indexMin;
}
```

Prin aceasta metoda vom inlocui astfel in memorie instructiunea mai veche care a fost cel mai putin folosita.

Tot in aceasta clasa am mai creat urmatoarele metode:

```
void clearCache(){
    for (int i=0;i<cacheInstructions.length;i++){
        cacheInstructions[i]=null;
        instrAddresses[i]=null;
    }
}
```

-> pentru curatarea memoriei cache. Am folosit-o de fiecare data cand am trecut la urmatorul set de instructiuni.

```
double hitRate(Cache cache){
    return (1 - missRate(cache));
}

double missRate(Cache cache){
    return (double) (cache.miss)/(totalCacheAccesses);
}
```

->pentru a calcula HitRate-ul si missRate-ul

```
boolean inCache(Cache cache, String instructionToFind)
```

->pentru a verifica daca o instructiune se gaseste in cache

O metoda importanta tot din aceasta clasa este cea de addInstruction.

```
void addInstruction(Cache cache, String instructionToAdd, String instrAddr)
```

-> in aceasta metoda verificam daca instructiunea curenta se afla in memorie, daca da atunci incrementam variabila hit si cea de totalCacheAccesses, setam hitIndex-ul sa fie egal cu indexul instructiunii din memoria cache, si scadem contorInstr pentru toate celelalte instructiuni de pe pozitii diferite de i. Daca instructiunea curenta nu se afla in cache atunci incrementam variabila miss si cea de totalCacheAccesses, scadem contorInstr pentru toate instructiunile, calculam pozitia pe care dorim sa introducem noua instructiune folosindu-ne de metoda getIndexOfInstructionToSubstitute() descrisa mai sus si adaugam instructiunea in cache.

CacheSim:

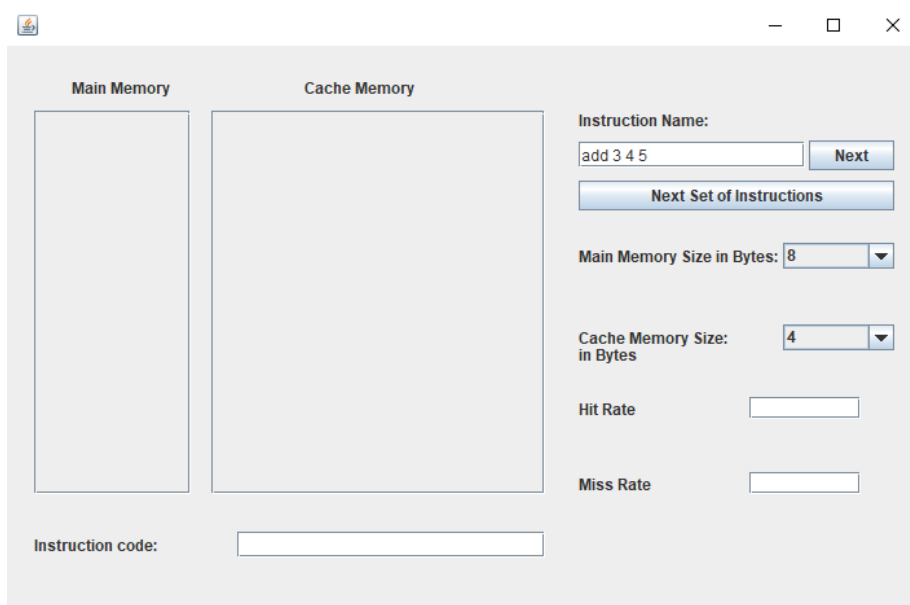
In aceasta clasa am create interfata pentru proiect si am facut-o functionabila, adica am setat actiunile pentru butoane, textField-urile, si tot ce mai aveam nevoie.

Interfata mea contine:

- un table pentru memoria principala
- un table pentru memoria cache
- un camp in care imi apare instructiunea curenta cu un buton de Next langa pentru a putea trece la instructiunea urmatoare
- un buton de Next Set of Instructions pentru a trece la urmatorul set de instructiuni
- un meniu pentru selectarea dimensiunii memoriei principale
- un meniu pentru selectarea dimensiunii memoriei cache
- un camp in care se va afisa Hit Rate-ul
- un camp in care se va afisa Miss Rate-ul
- un camp in care se va afisa instructiunea curenta codificata

Testare:

Cand rulam aplicatia va aparea urmatoarea fereastra:



Inainte de a face orice trebuie sa setam dimensiunile pentru memoria principala si pentru memoria cache. Apoi sa apasam pe butonul de nextInstruction pentru a initializa contorul pentru setul de instructiuni.

Main Memory

INDEX	DATA
0000	add 3 4 5
0001	sub 5 4 2
0010	add 1 3 2
0011	sub 5 4 2
0100	lw 3 2 1
0101	lw 3 2 1
0110	lw 3 2 1
0111	add 3 4 5
1000	lw 5 4 3

Cache Memory

Instruction Name:

Main Memory Size in Bytes:

Cache Memory Size in Bytes:

Hit Rate:

Miss Rate:

Instruction code:

Main Memory

INDEX	DATA
0000	add 3 4 5
0001	sub 5 4 2
0010	add 1 3 2
0011	sub 5 4 2
0100	lw 3 2 1
0101	lw 3 2 1
0110	lw 3 2 1
0111	add 3 4 5
1000	lw 5 4 3

Cache Memory

INDEX	ADDRESS	DATA

Instruction Name:

Main Memory Size in Bytes:

Cache Memory Size in Bytes:

Hit Rate:

Miss Rate:

Instruction code:

Dupa aceea putem trece la simularea memoriei cache apasand pe butonul de Next:

Main Memory

INDEX	DATA
0000	add 3 4 5
0001	sub 5 4 2
0010	add 1 3 2
0011	sub 5 4 2
0100	lw 3 2 1
0101	lw 3 2 1
0110	lw 3 2 1
0111	add 3 4 5
1000	lw 5 4 3

Cache Memory

INDEX	ADDRESS	DATA
0000	0000	add 3 4 5
0001	0001	sub 5 4 2
0010	1000	lw 5 4 3
0011	0100	lw 3 2 1

Instruction Name:

sub 5 4 2 Next

Next Set of Instructions

Main Memory Size in Bytes: 16

Cache Memory Size: 4 in Bytes

Hit Rate 0.6

Miss Rate 0.4

Instruction code: 0000 0101 0100 0010

In imaginea de mai sus putem observa cum de fiecare data cand gasim in cache instructiunea curenta ni se va colora in cache randul acesteia cu verde.

In tabelul cu memoria principala se va colora cu rosu instructiunea ce va urma sa fie adaugata in cache in viitorul apropiat. De exemplu ai jos se coloreaza cu rosu instructiunea "sub 5 4 2"

Main Memory

INDEX	DATA
0000	add 3 4 5
0001	sub 5 4 2
0010	add 1 3 2
0011	sub 5 4 2
0100	lw 3 2 1
0101	lw 3 2 1
0110	lw 3 2 1
0111	add 3 4 5
1000	lw 5 4 3

Cache Memory

INDEX	ADDRESS	DATA
0000	0000	add 3 4 5
0001	0010	add 1 3 2
0010	1000	lw 5 4 3
0011	0100	lw 3 2 1

Instruction Name:

add 1 3 2 Next

Next Set of Instructions

Main Memory Size in Bytes: 16

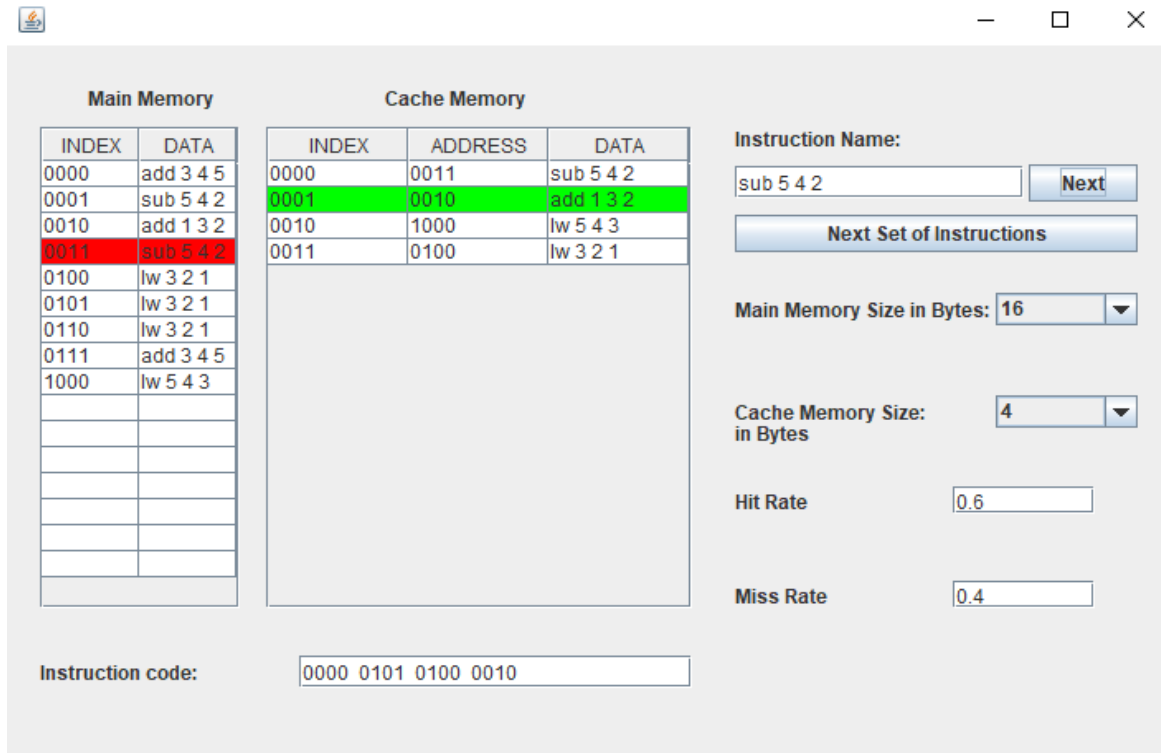
Cache Memory Size: 4 in Bytes

Hit Rate 0.6

Miss Rate 0.4

Instruction code: 0000 0001 0011 0010

Daca dam acum pe butonul de Next vom observa cum se va adauga in cache (in functie de algoritmul LRU), de data aceasta pe prima pozitie:



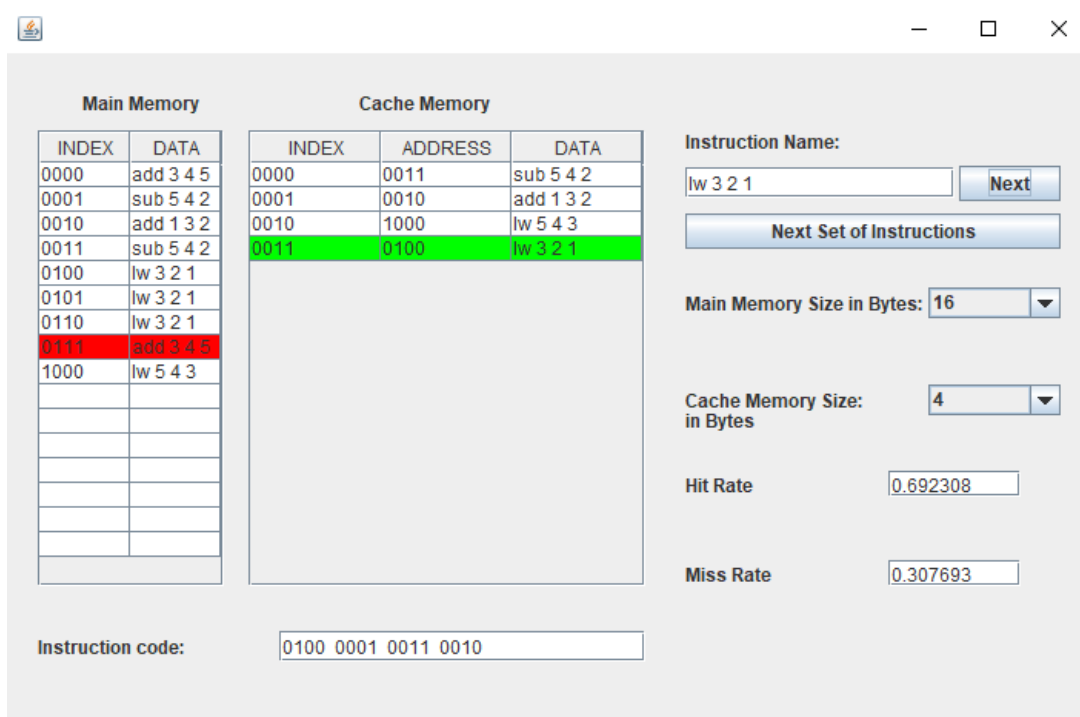
The screenshot shows a simulation window with the following components:

- Main Memory Table:**

INDEX	DATA
0000	add 3 4 5
0001	sub 5 4 2
0010	add 1 3 2
0011	sub 5 4 2
0100	lw 3 2 1
0101	lw 3 2 1
0110	lw 3 2 1
0111	add 3 4 5
1000	lw 5 4 3
- Cache Memory Table:**

INDEX	ADDRESS	DATA
0000	0011	sub 5 4 2
0001	0010	add 1 3 2
0010	1000	lw 5 4 3
0011	0100	lw 3 2 1
- Controls and Statistics:**
 - Instruction Name: sub 5 4 2 (with a Next button)
 - Next Set of Instructions (button)
 - Main Memory Size in Bytes: 16
 - Cache Memory Size: 4
 - Hit Rate: 0.6
 - Miss Rate: 0.4
 - Instruction code: 0000 0101 0100 0010

Pe parcurs ce cautam mai multe instructiuni putem observa cum se schimba Miss Rate-ul si Hit Rate-ul.

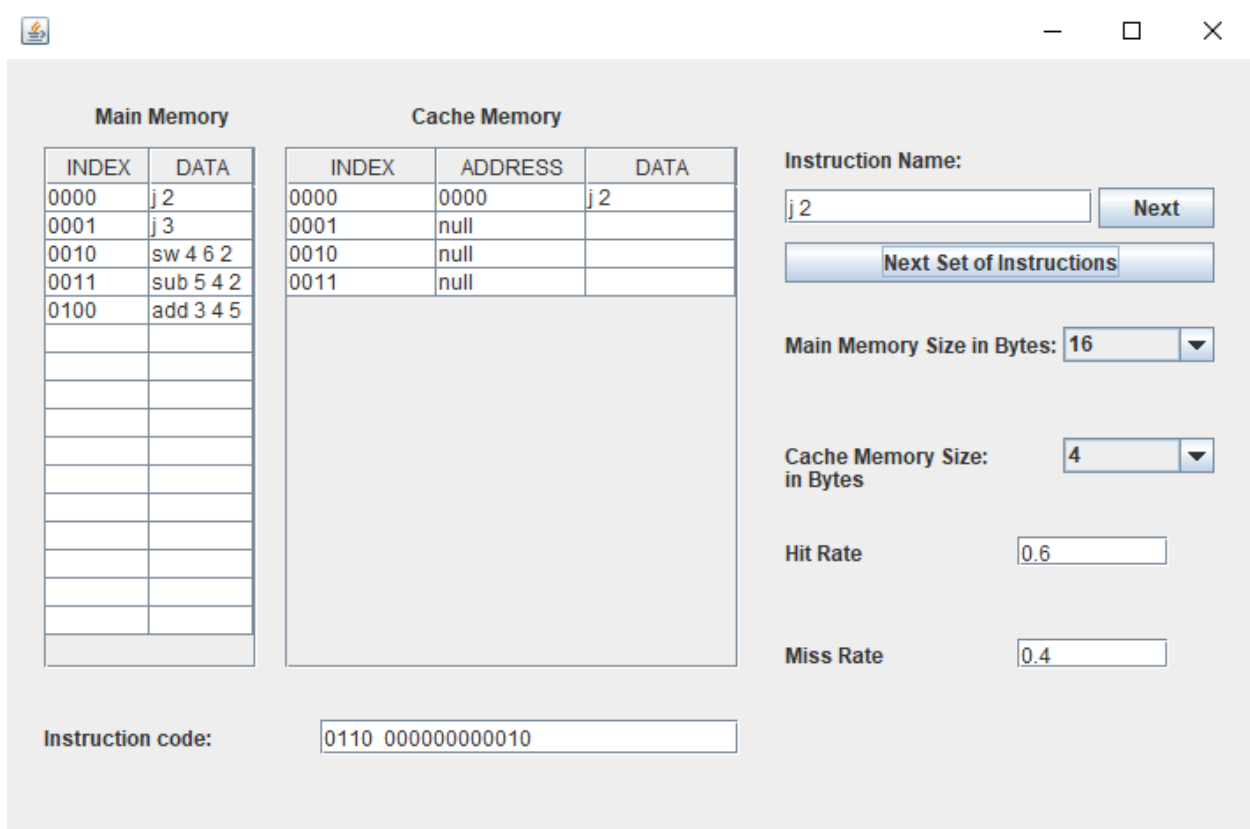


The screenshot shows the simulation window after processing more instructions. The state is as follows:

- Main Memory Table:** (Same as before)
- Cache Memory Table:**

INDEX	ADDRESS	DATA
0000	0011	sub 5 4 2
0001	0010	add 1 3 2
0010	1000	lw 5 4 3
0011	0100	lw 3 2 1
- Controls and Statistics:**
 - Instruction Name: lw 3 2 1 (with a Next button)
 - Next Set of Instructions (button)
 - Main Memory Size in Bytes: 16
 - Cache Memory Size: 4
 - Hit Rate: 0.692308
 - Miss Rate: 0.307693
 - Instruction code: 0100 0001 0011 0010

Acum daca dorim sa trecem la urmatorul set de instructiuni tot ce trebuie sa facem este sa apasam butonul de Next Set of Instructions:



The screenshot shows a software interface for simulating memory access. It features two tables for memory state, input fields for instruction details, and calculated performance metrics.

Main Memory	
INDEX	DATA
0000	j 2
0001	j 3
0010	sw 4 6 2
0011	sub 5 4 2
0100	add 3 4 5

Cache Memory		
INDEX	ADDRESS	DATA
0000	0000	j 2
0001	null	
0010	null	
0011	null	

Instruction Name:

Main Memory Size in Bytes:

Cache Memory Size: in Bytes

Hit Rate

Miss Rate

Instruction code:

Si astfel vor aparea in tabelul pentru memoria principala noile instructiuni din urmatorul set iar memoria cache se va goli.

Daca dorim putem seta alta dimensiune a memoriei principale/cache din meniul dedicate acestuia:

Schimbare dimensiune cache:

Main Memory

INDEX	DATA
0000	j 2
0001	j 3
0010	sw 4 6 2
0011	sub 5 4 2
0100	add 3 4 5

Cache Memory

INDEX	ADDRESS	DATA
0000	0010	sw 4 6 2
0001	null	
0010	null	
0011	null	
0100	null	
0101	null	
0110	null	
0111	null	

Instruction Name:

Main Memory Size in Bytes:

Cache Memory Size: in Bytes

Hit Rate:

Miss Rate:

Instruction code:

Schimbare dimensiune memorie principala:

Main Memory

INDEX	DATA
0000	j 2
0001	j 3
0010	sw 4 6 2
0011	sub 5 4 2
0100	add 3 4 5

Cache Memory

INDEX	ADDRESS	DATA
0000	0011	sub 5 4 2
0001	0100	add 3 4 5
0010	0000	j 2
0011	null	

Instruction Name:

Main Memory Size in Bytes:

Cache Memory Size: in Bytes

Hit Rate:

Miss Rate:

Instruction code:

5. Concluzii:

Astfel am implementat un simulator al memoriei cache care poate fi folosit in scop didactic pentru intelegerea si predarea modului de functionare al memoriei cache prin faptul ca:

- se pot seta dimensiunile memoriilor cache si principala
- se poate observa cum creste sau scade Hit Rate-ul sau Miss Rate-ul
- se pot testa mai multe seturi de instructiuni avand astfel o testare mai buna datorita acoperirii mai multor cazuri
- se poate observa unde este hit-ul in memoria cache (este colorat cu verde) si ce instructiune din memoria principala (este colorata cu rosu) vom adauga la pasul urmator in cache .

Bibliografie:

Associative mapping:

http://elf.cs.pub.ro/ac/wiki/_media/arhiva/2012/laborator10.pdf

http://www.csit-sun.pub.ro/~cpop/Computer_Systems_Structure/Memory/RO_Proiectare%20UCMC.html

Interpreter in assembly:

<https://discuss.codecademy.com/t/how-to-create-an-interpreter-of-assembler/425765>

<https://ruslanspivak.com/lbasi-part1/>

<https://bochs.sourceforge.io/>

Simulator UCP?

Cache memory simulator:

<https://github.com/seifhelal/Cache-Simulator>

<http://alumni.cs.ucr.edu/~snematbakhsh/code/code.html>

<https://stackoverflow.com/questions/2756068/cache-simulator-in-c>

http://www.dcs.ed.ac.uk/home/simjava/guide/examples/app_cache/