Simularea functionarii unei memorii cache

Student: Robu Raluca Ioana

Structura Sistemelor de calcul

Universitatea Tehnica Cluj Napoca

CUPRINS

1. INTRODUCERE

1.1. CONTEXT

Scopul acestui proiect este de a dezvolta un simulator de memorie cache, care sa simuleze functionalitate memoriei cache intr-un sistem de calcul. Acest simulator va oferi o imagine clara asupra modului in care functioneaza memoria cache in diverse configuratii, utilizand diverse tehnici de mapare si politici de scriere. Va permite, de asemenea, utilizatorilor sa observe indicatori de performanta, precum rata de accesari reusite , rata de erori si timpii de access la date.

1.2. OBIECTIVE

Simulatorul de memorie cache va fi dezvoltat in C++ folosind mediul de programare Visual Studio. Proiectul va include un set de functionalitati care sa simuleze comportamentul unei memorii cache si va permite utilizatorilor sa experimenteze diverse scenarii de configurare a acesteia. Un testbench va fi integrat pentru a simula si evalua performanta sistemului in diferite configuratii si setari ale cache-ului.

Simulatorul va permite utilizatorilor sa experimenteze cu urmatoarele elemente ale functionarii unei memorii cache:

- tipuri de mapare in cache (asociativa, directa, set asociativa);
- politici de inlocuire (LRU, LFU, random)
- politici de scriere (write-through, write-back)

Acest simulator va facilita intelegerea ierarhiei de memorie si va permite explorarea modului in care cache-ul influenteaza performanta generala a unui sistem.

2. STUDIU BIBLIOGRAFIC

2.1. CE ESTE MEMORIA CACHE SI CARE ESTE SCOPUL EI?

Memoria cache este o memorie mai complexa si mai eficienta decat memoria principala, care are ca scop cresterea vitezei de accesare a informatiei.

Acceseaza mai rapid informatia, deoarece este situata mai aproape de procesor decat memoria principala, de obicei fiind integrata in chip-ul CPU. Este mai mica decat memoria principala (RAM) si are mai putin spatiu de stocare. Actioneaza ca o memorie temporara, stocheaza cele mai frecvent/recent utilizate informatii.

Poate fi de 10-100 ori mai rapida decat memoria RAM. Are nevoie doar de cateva nanosecunde pentru a realizare o cerinta primita de la CPU.

Componenta hardware folosita in RAM este DRAM (**d**ynamic **r**andom **a**ccess **m**emory), iar in cache: SRAM (high speed **s**tatic **r**andom **a**ccess **m**emory).

Memoria cache este impartita pe 3 nivele, in functie de cat de aproape sunt aceste nivele de microprocessor:
-L1 (cel mai rapid, dar are putin spatiu de stocare)
-L2 (mai lent decat L1, dar are mai mult spatiu decat L1)
-L3 (implementat ca sa imbunatateasca performanta lui L1 si L2; mai lent decat L1 si L2, dar totusi dublu rapid fata de DRAM).

Atunci cand procesorul trimite o instructiune, adresa ei se cauta in memoria cache. Daca o gaseste in cache, extrage informatia de la acea adresa; se executa o operatie de scriere. Daca nu gaseste adresa, o cauta mai departe in memoria principala si o aduce de acolo in cache si se executa o operatie de scriere.
Daca informatia cautata este gasita in cache, nu mai este nevoie sa se acceseze memoria principala.

2.2. EFICIENTA SRAM

Eficienta se exprima in “rata de succes”, adica numarul de accesari finalizate cu success raportat la numarul total de accesari.
Vom numi;
->Cache **hit** : controlerul de cache a reusit sa anticipeze corect informatia.
->Cache **miss**: controlerul de cache nu a reusit sa anticipeze correct informatia.

De fiecare data cand are loc un cache miss, se asteapta ,maim ult dupa date (trebuie cautate in memoria principala si se produc intarzieri).

2.3. TEHNICI DE MAPARE A MEMORIEI CACHE

Prin tehnica de mapare, intelegem tehnica de aducere de informatii din memoria principala in memoria cache sau tehnica de identificare a liniei din cache in care se afla informatia ceruta.

Memoria prinicpala este impartita in blocuri de dimensiuni egale, iar memoria cache este impartita in linii de dimensiuni egale. De asemenea, dimensiunea unui bloc din memoria principala este egala cu dimensiunea unei linii din cache.
Exista 3 tipuri de mapare:

- Mapare asociativa
- Mapare directa
- Mapare set asociativa

Cand se incarca un program in memoria principala, memoria cache este golita. Fiecare linie din memoria cache are un “bit valid” (indica daca pastreaza un bloc din memoria principala, adica daca este ocupata) si un “dirty bit” (indica daca informatia din linia respectiva de cache s-a modificat).

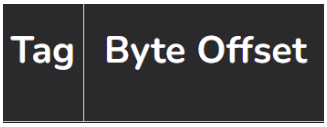
- ***Mapare asociativa***

Un bloc din memoria principala poate fi scris in orice linie din cache.

Adresa instructiunii este impartita in:

->campul Tag (indica blocul din memoria principala)

->campul Word/ Byte Offest (indica la ce byte din acel bloc se afla informatia cautata)



- ***Mapare directa***

Un bloc poate fi scris doar intr-o singura linie de cache, si aceasta se determina astfel:

linie cache = (nr blocului din memoria principala) % (nr de linii cache)

Daca un alt bloc de date care are aceeasi locatie corespunzatoare in cache trebuie incarcat in cache, iar aceasta locatie este ocupata, el va inlocui blocul existent.

Adresa se imparte in:

-> Tag (indica blocul din memoria principala in care se afla informatia cautata)

->Slot (indica linia corespunzatoare din memoria cache)

->Word/Byte Offset (indica la ce byte din acel bloc se afla informatia cautata)

Tag	Number of Cache Lines	Byte Offset

- ***Mapare set asociativa***

Este o combinatie a celor doua tehnici de mai sus.

Liniile din memoria cache sunt impartite in seturi cu numar egal de linii. Spre exemplu, daca un fiecare set continue cate 2 linii, vom spune ca avem cache “two-way”.

Fiecare bloc din memoria principala poate fi scris doar intr-un set din cache, dar in oricare linie din setul respectiv.

set cache = (nr blocului din memoria principala) % (nr de seturi cache)

Adresa se imparte in:

->Tag (indica blocul din memoria principala in care se afla informatia cautata)

->Set (indica setul corespunzator din memoria cache)

->Word/Byte Offset (indica la ce byte din acel bloc se afla informatia cautata)

Tag	Set Number	Byte Offset

2.4. POLITICI DE SCRIERE

->Write-through : data este scrisa deodata in cache si in memoria principala;

->Write-back : data este scrisa initial doar in cache; data este scrisa ulterior in memoria principala, cand linia respectiva din cache este eliminate. Se verifica “dirty bitul”; daca este 1, inseamna ca a fost modificata informatia cat timp era in cache si ca trebuie actualizata si in memoria principala.

->Write-around : data este scrisa direct in memoria principala, ignorandu-se memoria cache; se scrie in cache doar daca datele respective sunt citite din nou.

Deoarece de obicei sunt folosite doar primele doua, le vom implementa doar pe acelea in cadrul simulatorului.

2.5. STRATEGII DE INLOCUIRE

Daca nu mai exista spatiu in memoria cache, trebuie sa se inlocuiasca un bloc dintr-o linie de cache cu un bloc nou. Cum alegem ce eliminam din memoria cache? Exista mai multe tehnici de inlocuire:

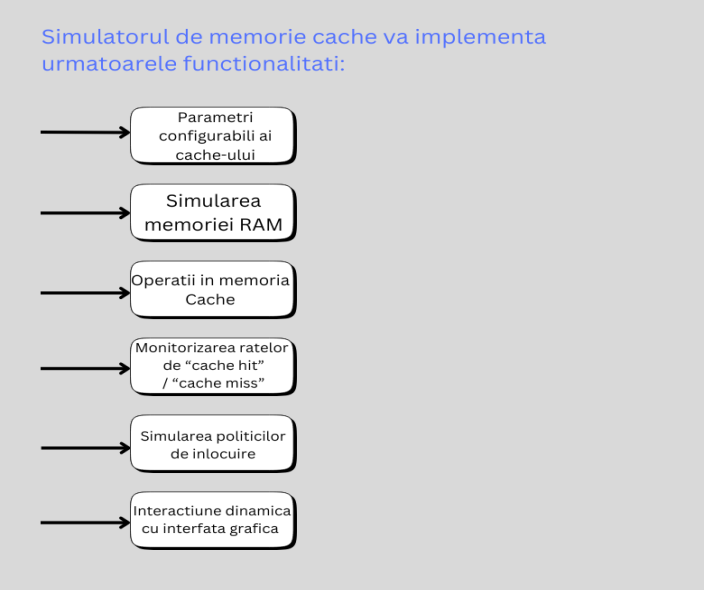
->LRU (least recently used): se elimina blocul care o fost folosit cel mai putin recent;

->LFU (least frequently used): se pastreaza un contor al numarului total de utilizari pentru fiecare bloc. Ca si avantaj, un bloc utilizat frecvent are o probabilitate mai mare de a ramane in memoria cache, insa un dezavantaj este ca blocurile incarcate recent au o valoare mica a contorului si risca sa fie eliminate.

->Inlocuire aleatoare : se allege un bloc in mod aleatory si se inlocuieste blocul cu data noua accesata; Avanatajul acestei strategii este implementarea simpla, insa exista dezavantajul ca blocurile cu probabilitate maxima de a fi utilizate din nou au aceeasi sansa de a fi eliminate ca si blocurile care nu vor fi utlizate din nou.

3. ANALIZA

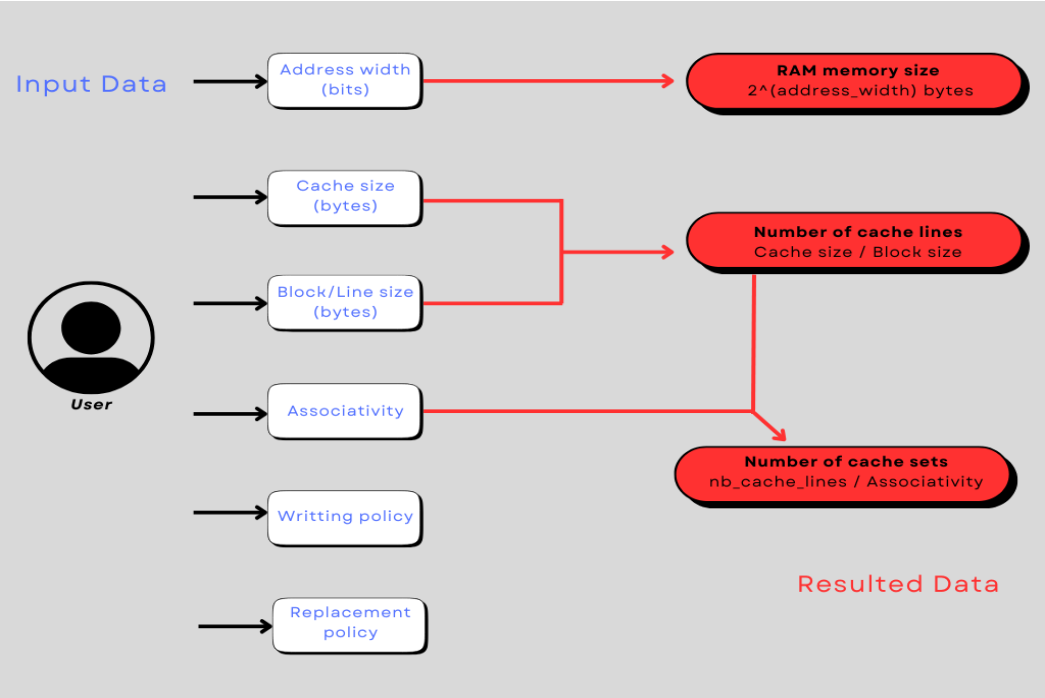
3.1. CE PROPUNE PROIECTUL?



3.2. ANALIZA PROIECTLUI

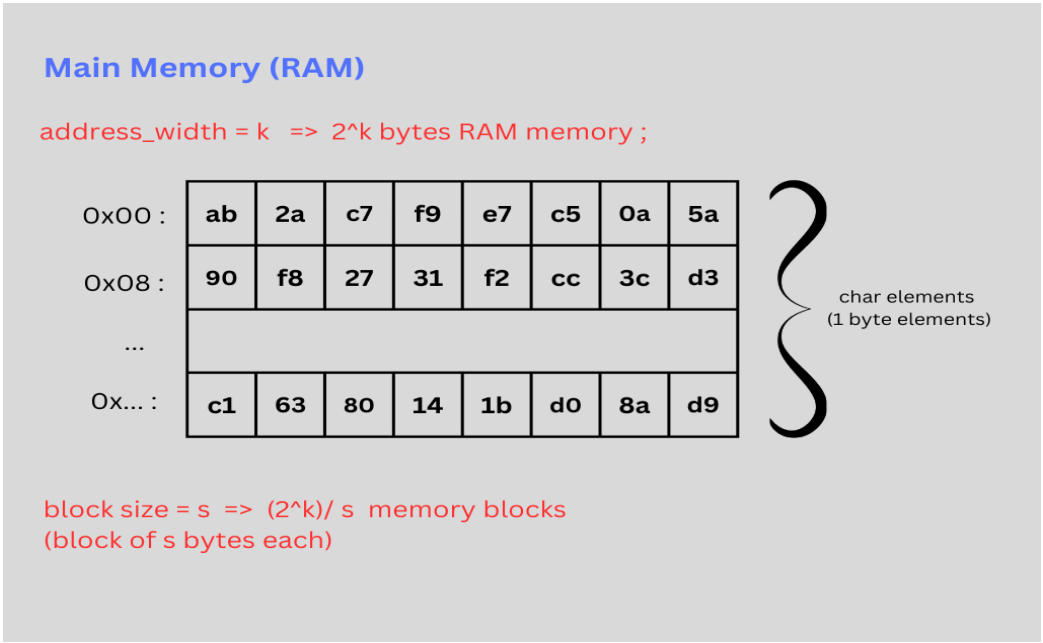
3.2.1. Introducerea datelor

Se vor introduce de catre utilizator datele prezentate in diagrama de mai jos. Cu ajutorul acestora, se va crea sistemul in care proiectul nostru se va desfasura. Pe baza valorilor introduse de utilizator, se va genera o memorie principala (RAM) si se va initializa memoria cache.

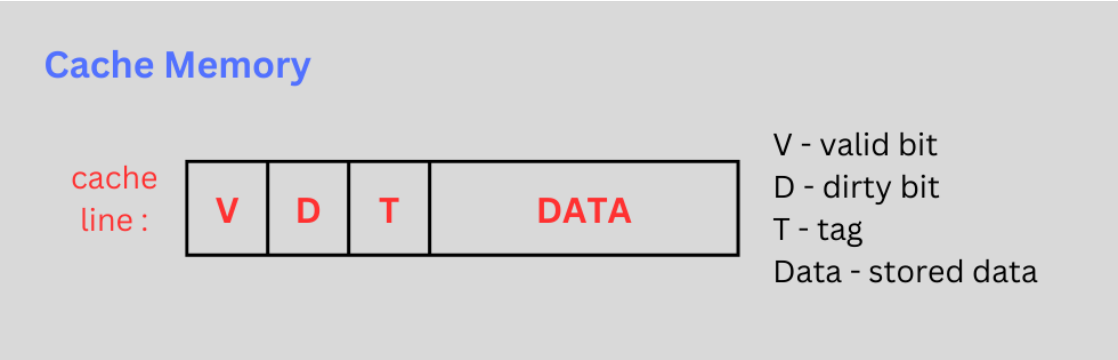


3.2.2. Generarea memoriei RAM si initializarea memoriei Cache

- In functie de inputul primit, se va genera **memoria principala** : un vector de marime 2^address_width bytes , care vor fi valori random. Apoi memoria va fi impartita in blocuri, in conformitate cu valoarea introdusa de utilizator pentru block/line size la pasul anterior.



- Structura unei linii de **memorii cache** este reprezentata in figura de mai jos:

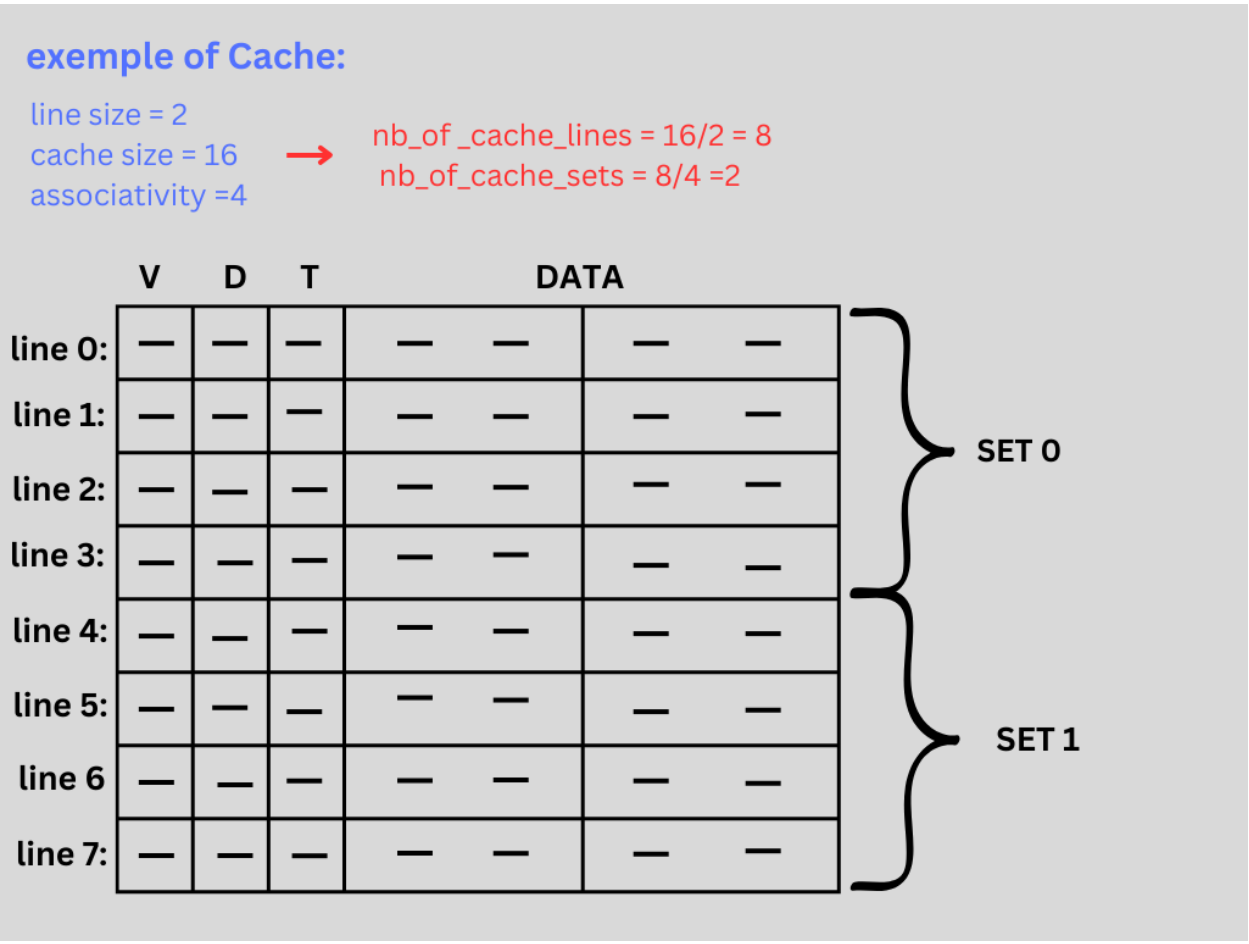


Valid bit -ul(1 bit) este setat initial pe 0 pentru toate liniile de cache. Vom vedea in continuarea implementarii proiectului ce valori ia si cand.

Dirty bit -ul(1 bit) este, de asemenea, setat initial pe 0 pentru toate liniile de cache.

Tag-ul reprezinta un identificator pentru blocurile de cache.

Data reprezinta data in sine care se scrie din blocul de memorie principala, in linia cache.



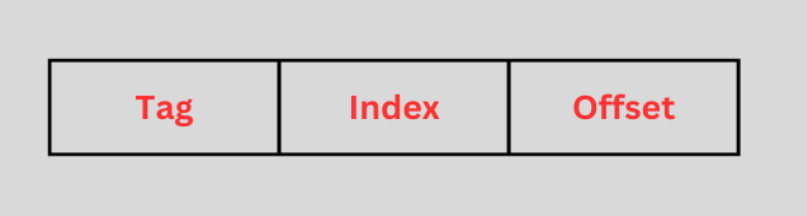
Se va initializa memoria cache conform valorilor introduse de utilizator (block/line size , cache size, associativity). Mai sus este ilustrat un exemplu al acestei initializari.

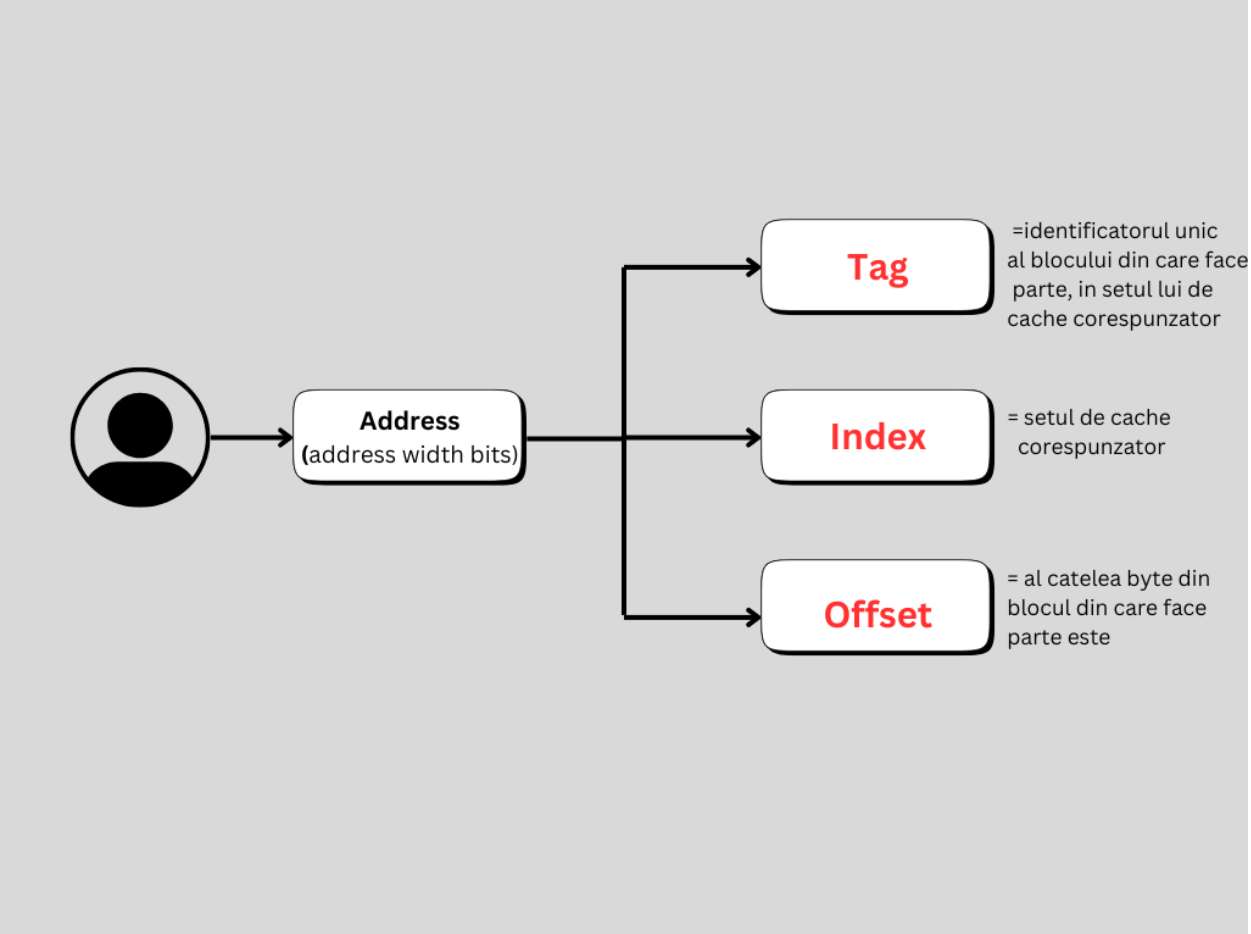
3.2.3. Cum functioneaza simularea?

Odata ce memoriile au fost initializate, incepe simularea. Vom dori sa efectuam operatii de citire/scriere de date din memoria principala, introducand adresa. De asemenea vom monitoriza rata de cache-hit si de cache-miss.

3.2.3.1. Cum impartim adresa?

Utilizatorul va introduce o adresa (de dimensiune address_width, despre care am discutat anterior). Aceasta adresa va fi impartita in 3:



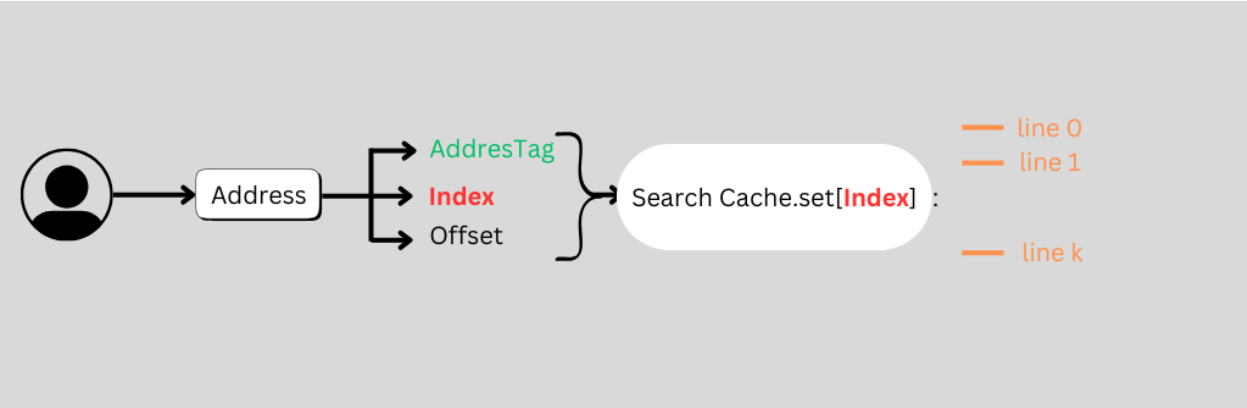


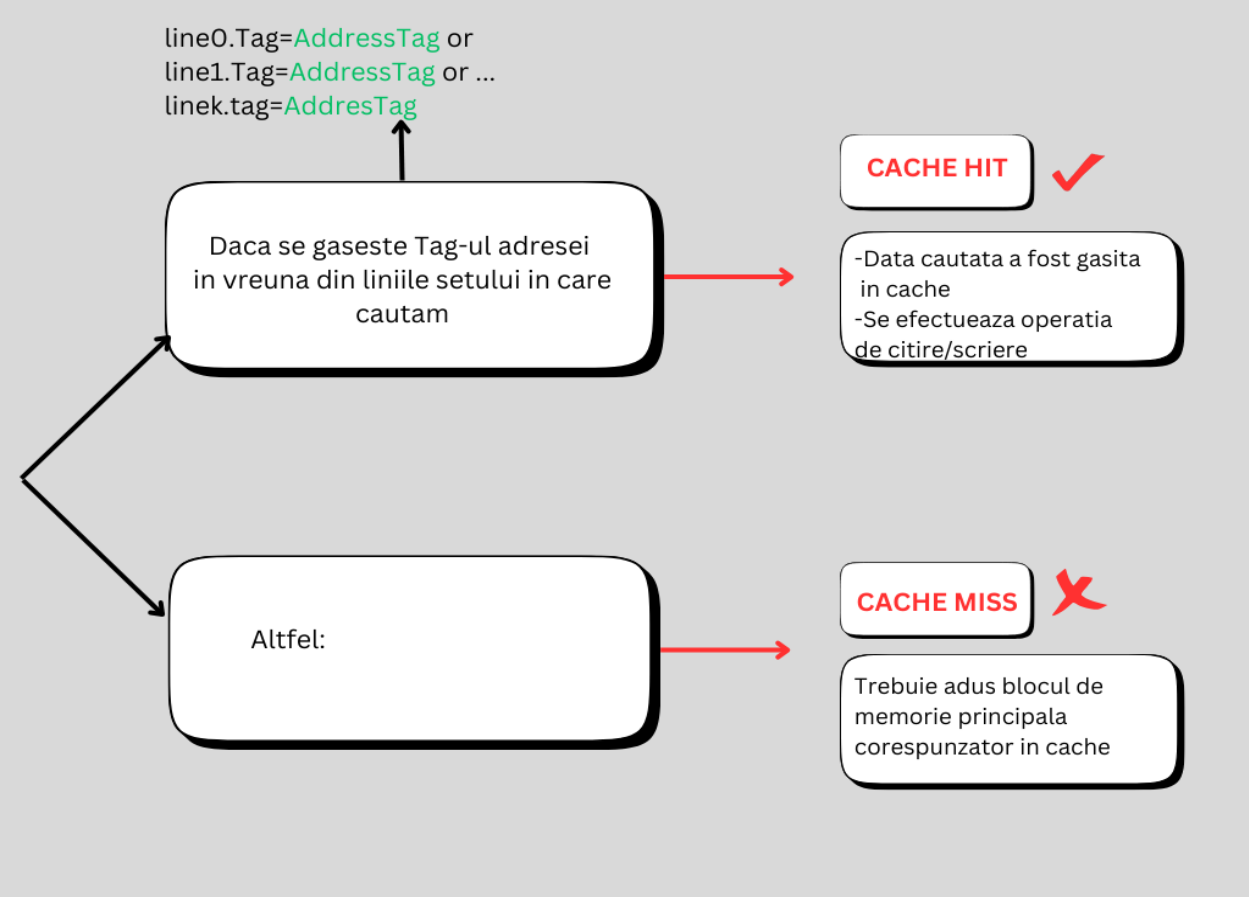
Latimea (width-ul) campurilor tag, index si offset se determina cu formulele:

- offset width = log2 (block size)
- index width = log2 (cache size / (block size * associativity))
- tag width = address width – index width – offset width

3.2.3.2. Cum accesam adresa?

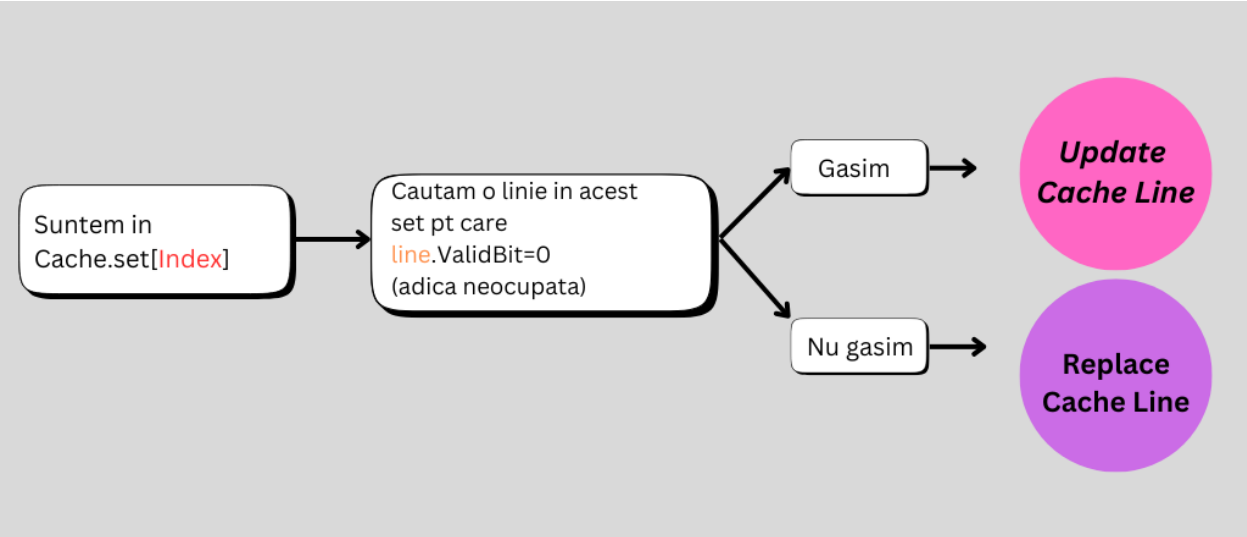
Conform explicatiilor de mai sus, odata ce utilizatorul introduce adresa datei cautate si aceasta este impartita in cele 3 campuri, trebuie cautat in setul de cache corespunzator:



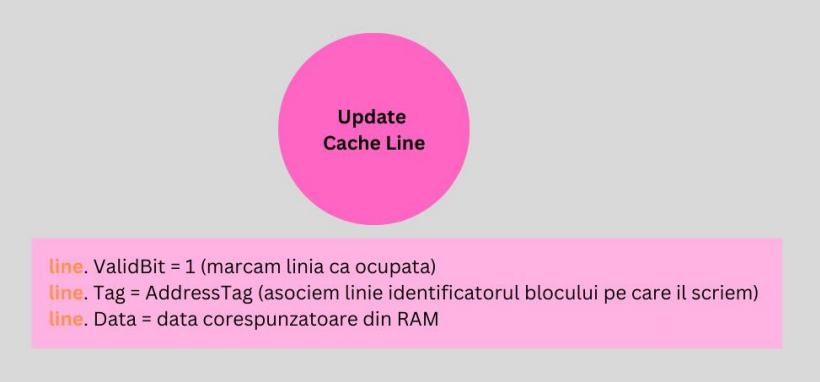


3.2.3.3. Cum gestionam cazul de CACHE MISS?

In cazul in care data de la adresa introdusa nu este gasita in memoria cache, trebuie luat din memoria principala blocul din care face parte, si apoi memorat in linia de cache corespunzatoare lui.



Daca gasim in setul corespunzator o linie de cache libera, o actualizam pe aceea astfel:

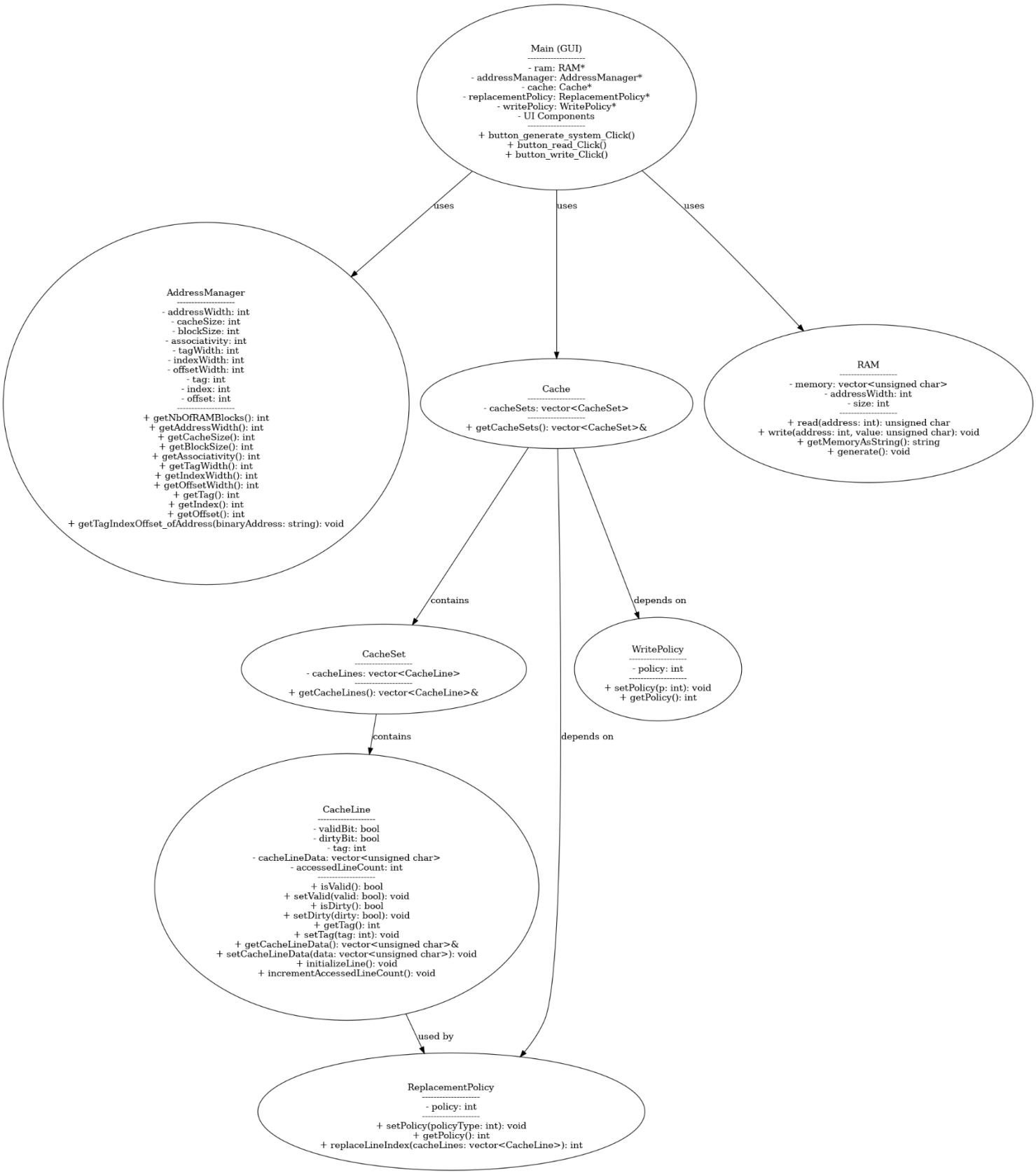


Datele corespunzatoare din RAM : vor fi acei block_size bytes din blocul in care se afla adresa introdusa. Tinand cont de offsetul adresei noastre, vedem cati bytes de dinainte si cati bytes de dupa fac parte din blocul ei.

In cazul in care nu se gaseste nicio linie de cache libera in setul in care cautam, linia pe care o actualizam (tot cu metoda de mai sus) se alege conform “Replacement policy-ului” (tehnica de inlocuire) selectat de utilizator la inceputul simularii:

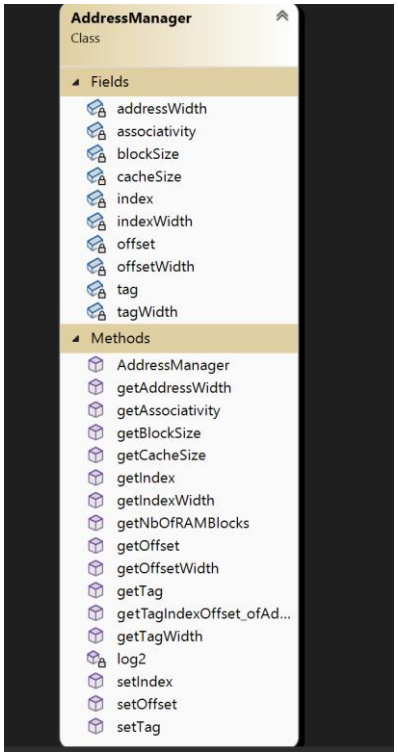
- LRU(least recently used) : se inlocuieste linia care nu a fost accesata recent(ultima);
- LFU(least frequently used) : se inlocuieste linia care a fost accesata de cele mai putine ori
- Random :se alege random linia care va fi inlocuita.

4. DESIGN



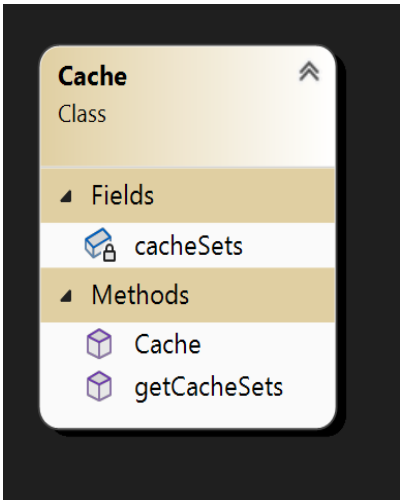
Mai sus este o diagrama a claselor ale proiectului, care ilustreaza relatiile si apartenentele dintre acestea.

AddressManager



- **Rol**: Gestioneaza adresele de memorie, impartindu-le in segmente (tag, index si offset) pentru a fi utilizate in cache.
- **Atribute importante**:
  - addressWidth, cacheSize, blockSize, associativity – determina structura si organizarea adreselor.
  - tag, index, offset – identificatori extrasi dintr-o adresa de memorie.
- **Metode principale**:
  - getTagIndexOffset_ofAddress – calculeaza segmentele (tag, index, offset) dintr-o adresa binara.
  - log2 – metoda utilizata pentru a calcula dimensiuni logaritmice, esentiala in segmentarea adreselor.
- **Rol in sistem**: Asigura transformarea adreselor binare in componentele necesare pentru accesarea datelor in cache.

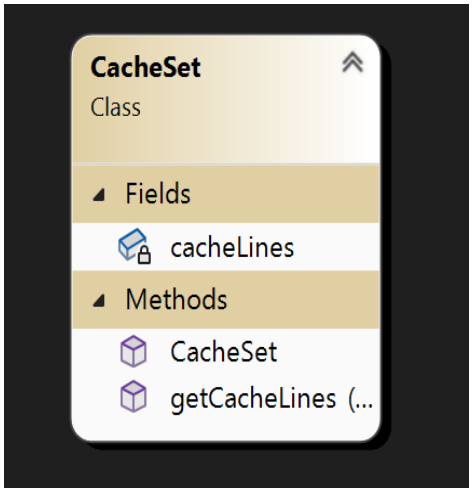
Cache



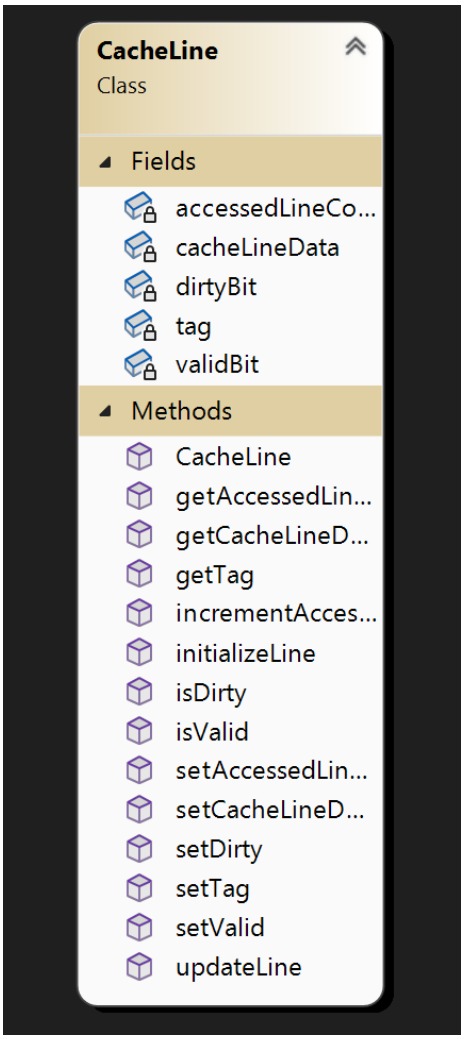
- **Rol**: Reprezinta memoria cache in ansamblu, organizata in mai multe seturi (CacheSet).
- **Atribute importante**:
  - cacheSets – o colectie de obiecte CacheSet, fiecare corespunzand unui set din cache.
- **Metode principale**:
  - getCacheSets – ofera acces la seturile din cache.
- **Rol in sistem**: Este responsabil pentru organizarea si gestionarea datelor din cache.

CacheSet

- **Rol**: Gestioneaza un set de linii de cache, conform regulii de asociativitate.
- **Atribute importante**:
  - cacheLines – o colectie de obiecte CacheLine.
- **Metode principale**:
  - getCacheLines – ofera acces la liniile din set.
- **Rol in sistem**: Este utilizat pentru a grupa mai multe linii de cache conform asociativitatii (directa, asociativa partiala sau completa).

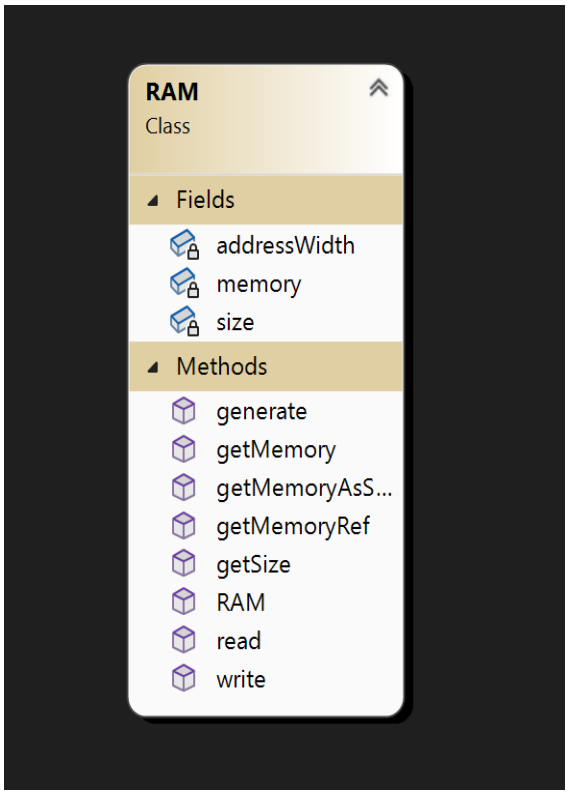


CacheLine



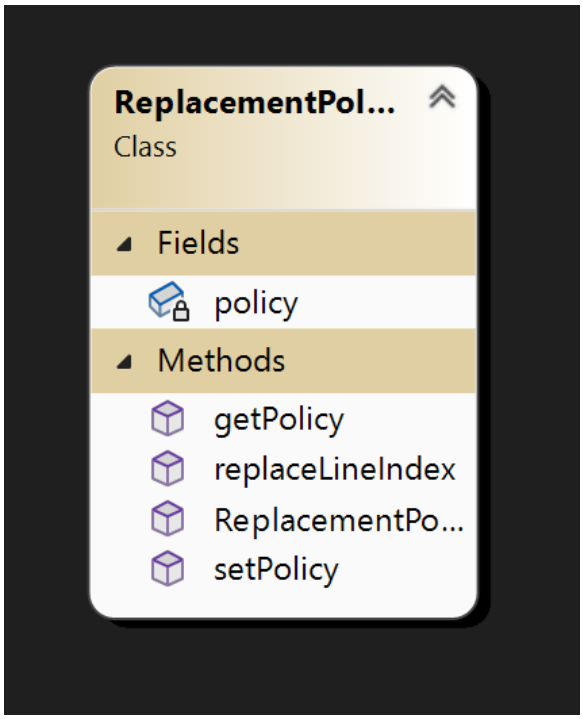
- **Rol**: Reprezinta o linie individuala din cache, care stocheaza date si stari (valid, dirty).
- **Atribute importante**:
  - validBit si dirtyBit – indica daca linia este valida sau modificata.
  - tag – identificator unic pentru linie.
  - cacheLineData – datele stocate in linia de cache.
  - accessedLineCount – contor pentru utilizare, esential pentru politica de inlocuire.
- **Metode principale**:
  - isValid, isDirty, setTag – metode pentru accesarea si modificarea starii liniei.
  - initializeLine – initializeaza linia de cache cu date noi.
  - incrementAccessedLineCount – actualizeaza contorul de acces.
- **Rol in sistem**: Asigura stocarea datelor si mentine starea lor, fiind unitatea de baza din cache.

RAM



- **Rol**: Simuleaza memoria principala (RAM) a sistemului.
- **Atribute importante**:
  - memory – vector care stocheaza datele din RAM.
  - addressWidth, size – dimensiunea adreselor si a memoriei.
- **Metode principale**:
  - read si write – permit citirea si scrierea datelor la adrese specifice.
  - generate – genereaza date aleatorii pentru RAM.
- **Rol in sistem**: Este sursa principala de date pentru cache, utilizata in cazul unui miss in cache.

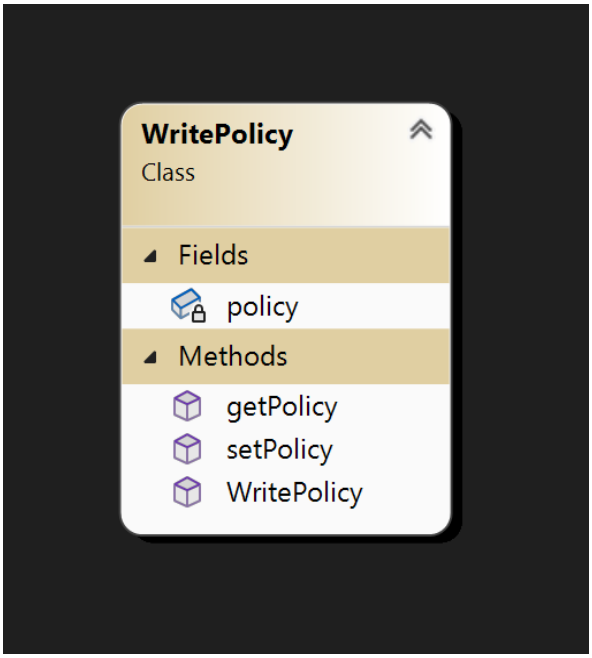
ReplacementPolicy



- **Rol**: Implementarea logicii de inlocuire a liniilor de cache.
- **Atribute importante**:
  - policy – tipul politicii utilizate (0 = LRU, 1 = LFU, 2 = Random).
- **Metode principale**:
  - setPolicy si getPolicy – configureaza si returneaza tipul politicii de inlocuire.
  - replaceLineIndex – determina linia care trebuie inlocuita pe baza politicii.
- **Rol in sistem**: Asigura gestionarea spatiului din cache, inlocuind linii conform politicii alese.

WritePolicy

- **Rol**: Defineste strategia de scriere a datelor in cache si RAM.
- **Atribute importante**:
    - policy – tipul politicii (0 = Write Back, 1 = Write Through).
- **Metode principale**:
    - setPolicy si getPolicy – configureaza si returneaza politica de scriere.
- **Rol in sistem**: Controleaza modul in care datele sunt sincronizate intre cache si RAM.



Main

Clasa Main reprezinta interfata principala a aplicatiei care permite utilizatorilor sa configureze si sa interactioneze cu sistemul de simulare al memoriei cache. Aceasta clasa gestioneaza legatura intre componentele interne ale sistemului (RAM, Cache, AddressManager, ReplacementPolicy, WritePolicy) si vizualizarea acestora in interfata grafica.

Rolul clasei Main

- Leaga toate componentele interne (RAM, cache, politici de scriere si inlocuire) si gestioneaza fluxul de date intre ele.
- Permite utilizatorului sa configureze dimensiunile cache-ului, politica de inlocuire, politica de scriere si alte setari.
- Asigura interactiunea cu utilizatorul prin evenimente (citire/scriere adrese, configurarea sistemului).

Atribute importante

1. **Obiecte interne:**
    - RAM* ram: Simuleaza memoria principala.

- Cache* cache: Reprezinta cache-ul cu seturile si liniile asociate.
- AddressManager* addressManager: Calculeaza tag-ul, index-ul si offset-ul din adresa binara.
- ReplacementPolicy* replacementPolicy: Politica de inlocuire a liniilor de cache (LRU, LFU, Random).
- WritePolicy* writePolicy: Politica de scriere a datelor in cache si RAM (Write Back, Write Through).

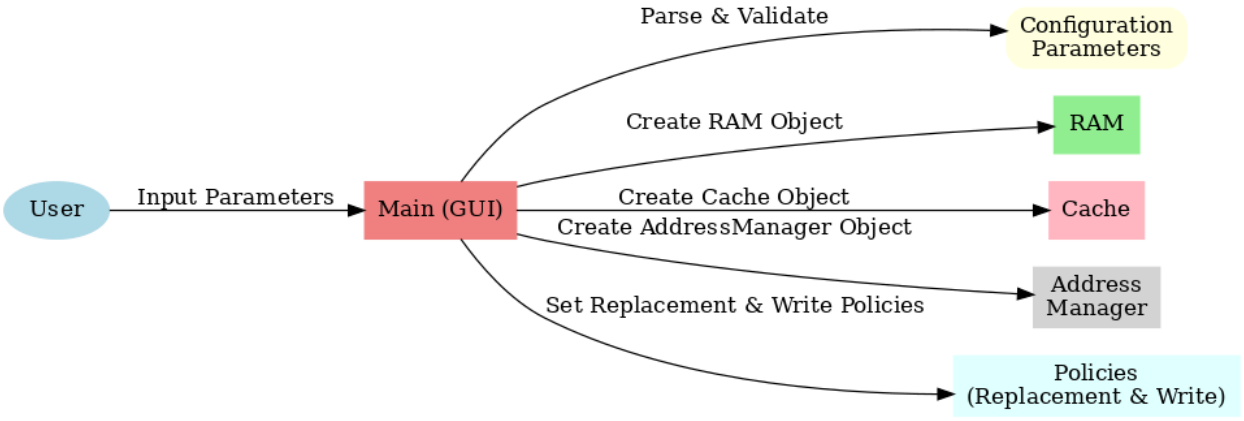
2. **Componente grafice (GUI):**
   - ComboBox-uri pentru configurarea dimensiunii cache-ului, asociativitatii si politicilor.
   - Panouri scrollabile (Panel^) pentru vizualizarea cache-ului si a memoriei RAM.
   - TextBox-uri pentru afisarea ratei hit/miss si introducerea adreselor de memorie.
3. **Alte atribute:**
   - int addressWidth: Latimea adreselor de memorie.
   - System::Collections::Generic::List<Panel^>^ cacheSetPanels: Elemente grafice pentru vizualizarea seturilor de cache.

Metode principale

1. button_generate_system_Click

- Initializeaza sistemul in functie de valorile selectate de utilizator:
  - Dimensiunea cache-ului, blocurilor si asociativitatea.
  - Politicile de inlocuire si scriere.
- Creeaza obiectele RAM, Cache, si AddressManager cu configuratia setata.
- Genereaza date aleatorii in RAM si pregateste interfata pentru a vizualiza seturile de cache si datele RAM.
- **Flux intern:**

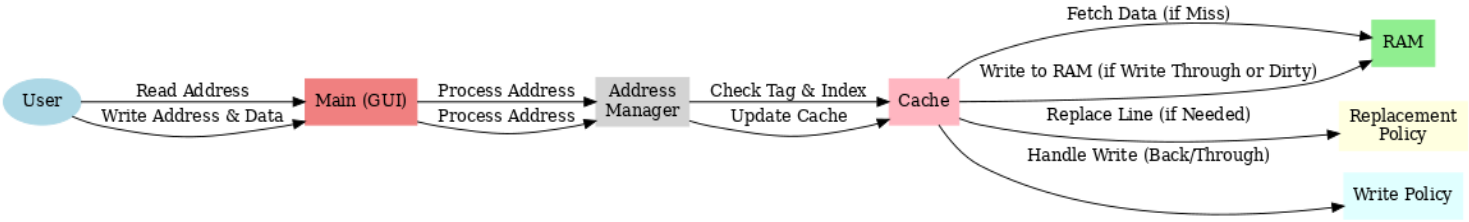
1. Citeste configuratia din interfata grafica.
2. Instanteaza si configureaza obiectele interne (RAM, Cache, etc.).
3. Genereaza si afiseaza datele in panourile de vizualizare.



2. button_read_Click

- Proceseaza o operatiune de citire a datelor dintr-o adresa specificata:
    - Adresa este transformata in tag, index si offset folosind AddressManager.
    - Se verifica daca datele sunt in cache (hit) sau trebuie incarcate din RAM (miss).
    - Daca este un **cache miss**, se incarca datele corespunzatoare din RAM si, daca este necesar, se aplica politica de inlocuire (e.g., LRU).
    - Evidentiaza datele accesate in interfata grafica.
- **Flux intern:**

1. Transformarea adresei binare in componente.
2. Cautarea datelor in cache.
3. Gestionarea unui hit/miss.
4. Actualizarea interfetei cu rezultatele operatiunii.



3. button_write_Click

- Proceseaza o operatiune de scriere a datelor intr-o adresa specificata:
    - Similar cu button_read_Click, dar include actualizarea datelor in cache si RAM in functie de politica de scriere selectata (Write Back sau Write Through).

- Daca politica este **Write Back**, datele sunt marcate ca "dirty" in cache.
- Daca politica este **Write Through**, datele sunt scrise direct in RAM.

- **Flux intern:**

1. Determinarea tag-ului, index-ului si offset-ului.
2. Verificarea cache-ului pentru un hit/miss.
3. Scrierea datelor in cache si/sau RAM in functie de politica.

Modul de legatura intre componente

1. **Configurare:** Utilizatorul seteaza parametrii (dimensiunea cache-ului, politica, etc.) prin GUI. Acesti parametri sunt utilizati pentru a crea si configura componentele interne.
2. **Adrese:** AddressManager proceseaza fiecare adresa si o transforma in segmente necesare (tag, index, offset).
3. **Cache/RAM:** Cache gestioneaza liniile si seturile, folosind ReplacementPolicy pentru inlocuire. RAM stocheaza datele in cazul unui cache miss.
4. **Interactiune:** Evenimentele de citire/scriere declanseaza operatiuni care leaga toate componentele.

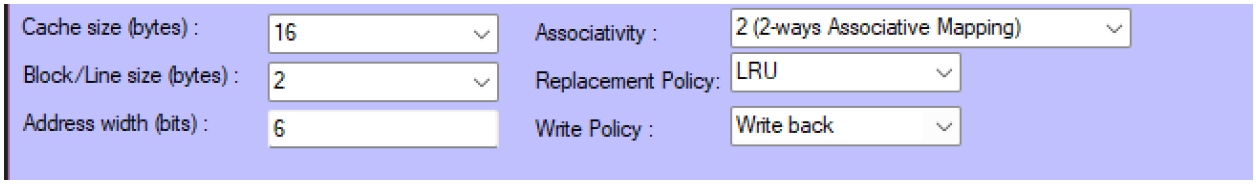
5. IMPLEMENTARE, TESTARE SI VALIDARE

Pentru a valida funcționarea sistemului, s-au definit mai multe scenarii de testare care acoperă majoritatea situațiilor posibile. Aceste scenarii evidențiază gestionarea cache hits, cache misses, scrierea datelor, și înlocuirea liniilor de cache.

5.1. Scenariul 1 **:**

cache hit, cache miss, inlocuirea unei linii (LRU), scrierea in memorie (write-back), mapare set asociativa

Configurație Cache:

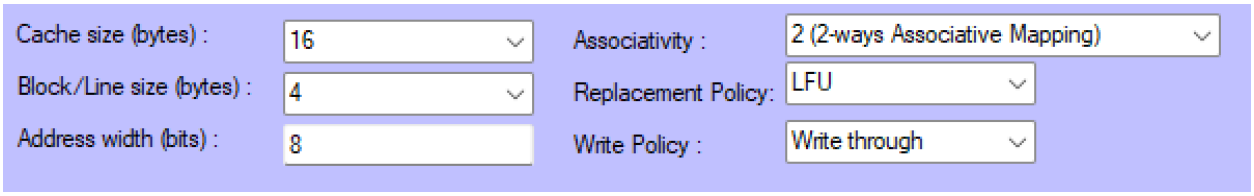


Set de instructiuni:

- Write la adresa x00 , val xAA => cache miss + actualizare dirty bit
- Read adresa x08 => umplere linie
- Read adresa x10 => LRU + write back la adresa x00 in memoria RAM (umplere set)
- Read adresa x11 => cache hit

5.2. Scenariul 2:

Write-through, LFU, mapare set asociativa

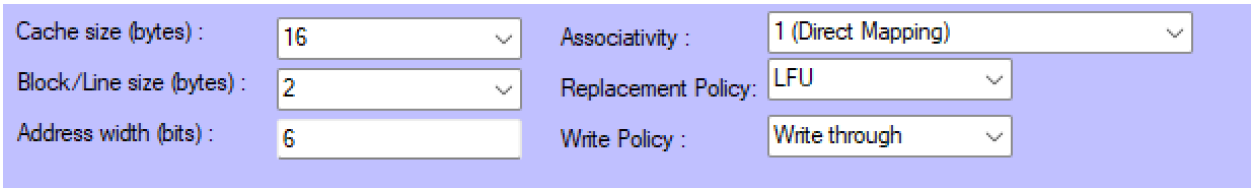


Set de instructiuni:

- Write la adresa x00 , val 22 => write through
- Read adresa x10 => umplere linie
- Write la adresa x10, val xAA
- Write la adresa x10, val xBB
- Read adresa x08 => LFU (umplere set)

5.3. Scenariul 3:

Mapare directa, umplere cache



Set de instructiuni:

- Read adresa x00
- Read adresa x02
- Read adresa x10 => inlocuire linie 0

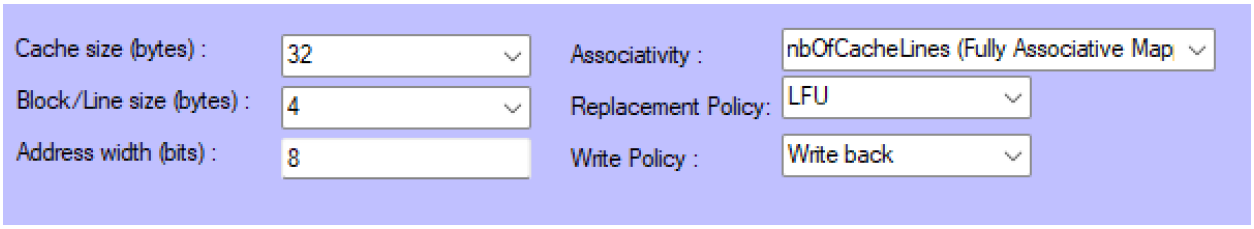
Adresa x10 se afla in blocul de ram cu index 8; sunt 8 linii de cache => linia corespunzatoare este linia

8%8=0

- Read la adresele x04, x06, x08, x0A, x0C, x0E => umplerea cache-ului

5.4. Scenariul 4:

Mapare complet asociativa



Set de instructiuni:

- Read adresa x00
- Read adresa x04
- Read adresa x0A
- Read adresa x23
- Write la adresa x01, valoarea xCC

Cand factorul de asociativitate este egal cu nr de linii de cache => Avem un singur set de cache

Orice bloc este mapat la prima linie de cache pe care o gaseste disponibila

Pentru toate scenariile de mai sus, putem de asemenea observa contorizarea cache hit-urile si cache miss-urilor si actualizarea in timp real a ratio-ului de cache hit/miss.

6. CONCLUZII

Proiectul demonstreaza modul de functionare al memoriei cache printr-un sistem modular si configurabil. Testele realizate valideaza corectitudinea mecanismelor de mapare, inlocuire si scriere, evidentiind diferentele intre politici. Solutia ofera o intelegere clara a performantelor si limitarilor memoriei cache.

7. BIBLIOGRAFIE

- https://www.techtarget.com/searchstorage/definition/cache-memory
- https://www.geeksforgeeks.org/cache-mapping-techniques/
- https://users.utcluj.ro/~apateana/Lab13_mem_cache.pdf
- https://www.youtube.com/watch?v=OxaYvJquPe0&ab_channel=NesoAcademy
- https://www.geeksforgeeks.org/difference-between-direct-mapping-associative-mapping-set-associative-mapping/
- https://www.geeksforgeeks.org/cache-write-policies-system-design/
- https://courses.cs.washington.edu/courses/cse351/cachesim/