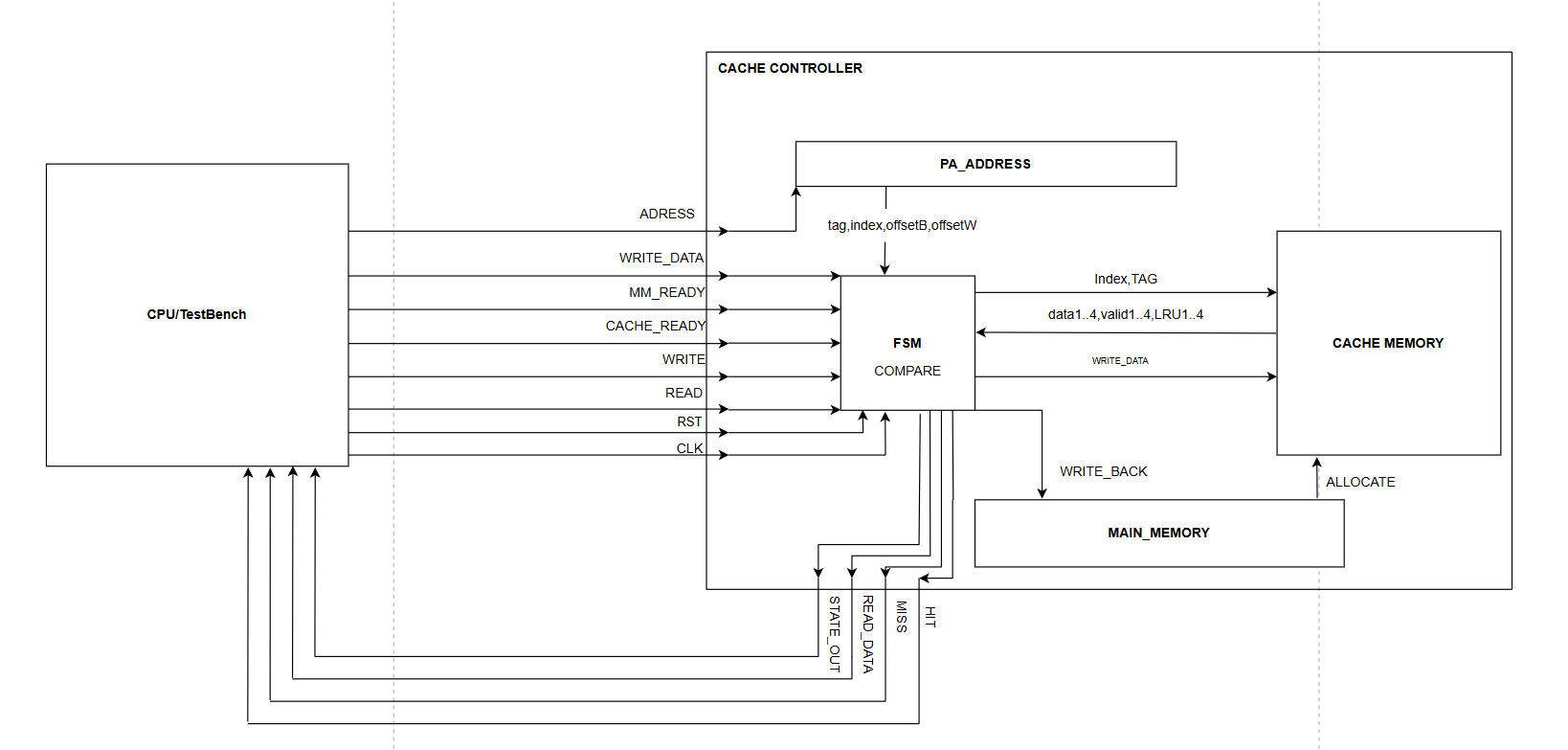


Resources:

ModelSim - simulating the ALU

Drawio - the diagrams

DIAGRAMA



În acest proiect am implementat un cache controller complet funcțional, conceput să intermedieze comunicația dintre CPU (simulat aici printr-un testbench) și memoria principală. Arhitectura urmărește optimizarea timpului de acces la date, printr-un sistem de cache performant și eficient, după cum se observă în diagrama de mai sus.

**Specificații tehnice ale cache-ului:**

* + Tip cache: 4-way set associative
  + Dimensiune totală: 32 KB
  + Dimensiune block: 64 bytes => 512 blockuri totale in cache
  + Dimensiune cuvânt: 4 bytes
  + Număr de seturi: 128
  + Politică de înlocuire: Least Recently Used (LRU)
  + Politică de scriere: Write-back cu write-allocate

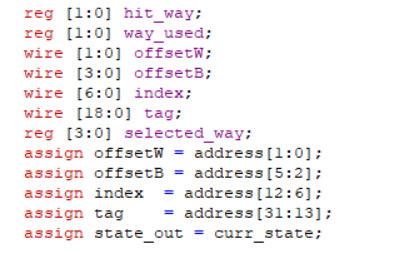
**Componente principale:**

* CPU/TestBench: trimite adrese și comenzi de citire/scriere către controller.
* FSM (Finite State Machine): controlează întregul flux de operații în funcție de semnalele de control și starea sistemului.
* PA\_ADDRESS: modulează adresa fizică în componente (tag, index, offset).
* CACHE MEMORY: stochează datele structurate pe seturi și căi (ways), folosind valid bits și tag-uri pentru comparare.
* MAIN MEMORY: este accesată doar în caz de cache miss, pentru operații de alocare (read miss) sau write-back (write miss cu dirty bit activ).
* Fluxul de operare: La o cerere de la CPU (read/write), adresa este decodificată în tag, index și offset.

A screenshot of a computer

AI-generated content may be incorrect.**Descompunerea adresei fizice (P.A.)**

Adresa fizică de 32 biți este împărțită astfel:

* Tag (19 biți): identifică blocul de memorie, folosit pentru a verifica dacă datele se află în cache (tag match).
* Index (7 biți): selectează unul dintre cele 128 de seturi din cache (2⁷ = 128).
* Block Offset (4 biți): selectează un cuvânt în interiorul unui block de 64 de bytes (16 cuvinte × 4B).
* Word Offset (2 biți): adresează exact cuvântul din cadrul unui grup de 4 bytes.

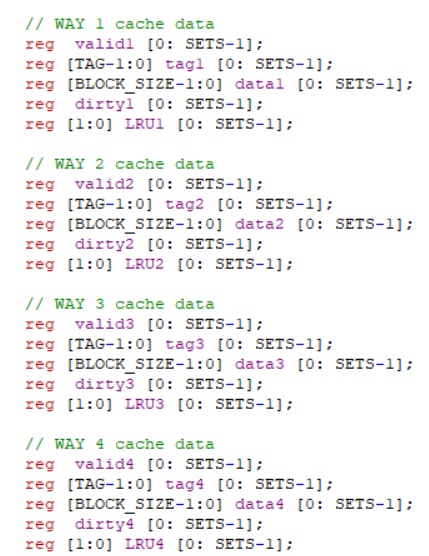
Această împărțire este esențială pentru a localiza rapid poziția unei date în cache.

A screenshot of a computer

AI-generated content may be incorrect.**Structura internă – 128 seturi, 4 way-uri**

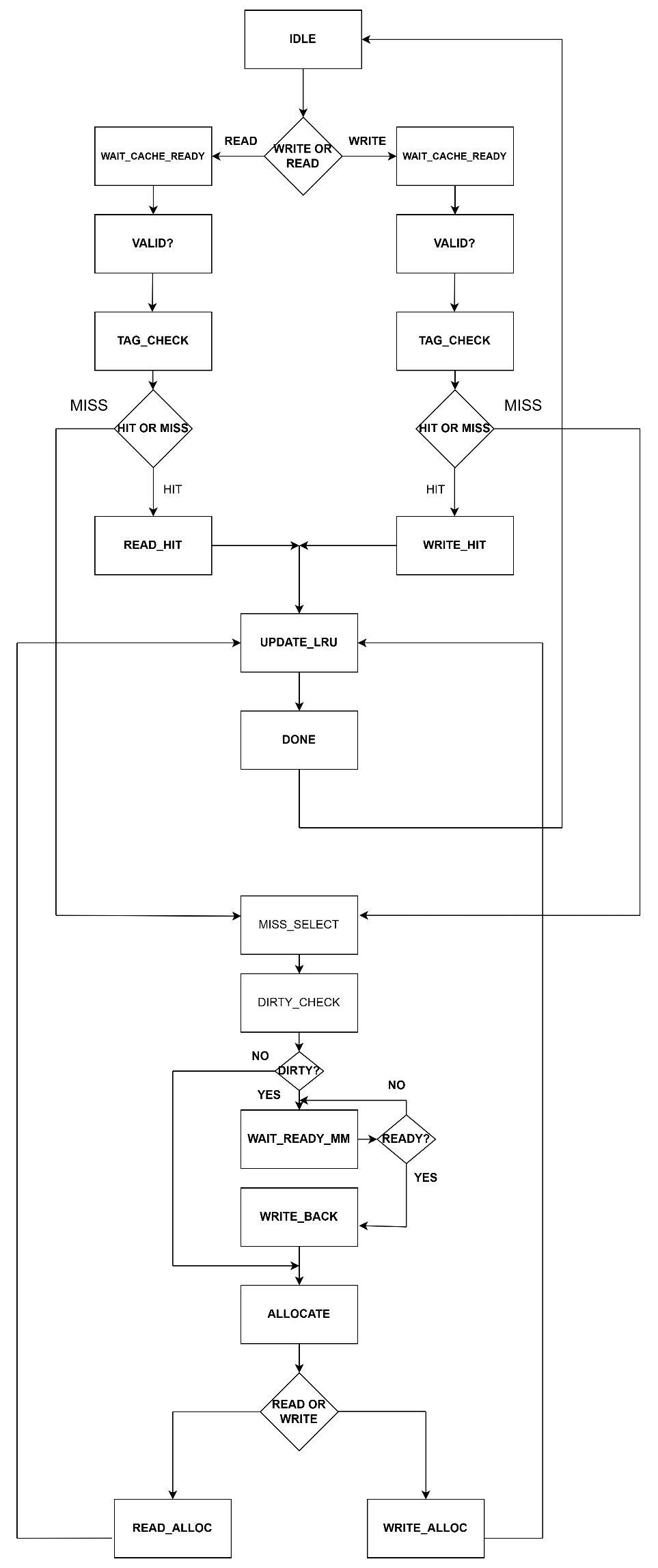
Cache-ul este organizat ca 4-way set associative, ceea ce înseamnă că:

Fiecare set are 4 poziții (way-uri) unde un bloc de date poate fi stocat.

Fiecare way conține:

* Valid (1 bit): indică dacă linia conține date valide.
* Tag (19 biți): identificatorul blocului.
* Data (64 bytes): blocul complet de date.
* Dirty (1 bit): semnalează dacă linia a fost modificată (folosit în write-back).
* LRU (2 biți): codifică poziția de utilizare a celui mai puțin recent folosit way pentru politica Least Recently Used.

Această structură este replicată pentru fiecare dintre cele 128 de seturi (SET0 până la SET127), rezultând un total de 512 linii de cache (128 seturi × 4 way-uri).

 **Fluxul principal al FSM-ului:**

* IDLE: Așteaptă comenzi read sau write. La o solicitare, trece în WAIT\_CACHE\_READY.
* WAIT\_CACHE\_READY: Așteaptă confirmarea că cache-ul este pregătit (cache\_ready == 1). Continuă în VALID.
* VALID: Verifică dacă vreuna dintre cele 4 căi este validă la indexul respectiv. Dacă da, merge în TAG\_CHECK; altfel, trece direct în MISS\_SELECT.
* TAG\_CHECK: Compară tag-ul cerut cu cele 4 tag-uri din set: Dacă este un hit, merge în READ\_HIT sau WRITE\_HIT. Altfel, continuă în MISS\_SELECT.
* READ\_HIT / WRITE\_HIT: Execută operația de citire/scriere pe calea (way) care a dat hit. Marchează hit = 1, miss = 0. Merge în UPDATE\_LRU.
* UPDATE\_LRU: Actualizează informațiile LRU pentru a reflecta ultima cale accesată. Merge în DONE.
* MISS\_SELECT: Selectează o cale disponibilă (nevalidă), sau cea mai puțin recent folosită (LRU = 3). Trecere în DIRTY\_CHECK.
* DIRTY\_CHECK: Verifică dacă linia selectată este dirty. Dacă da, merge în WAIT\_READY\_MM. Dacă nu, sare direct în ALLOCATE.
* WAIT\_READY\_MM: Așteaptă ca memoria principală (MM\_ready) să fie disponibilă. Merge în WRITE\_BACK.
* WRITE\_BACK: Curăță linia selectată (setează dirty = 0) după ce datele au fost scrise în memoria principală. Merge în ALLOCATE.
* ALLOCATE: Alocă blocul în cache: scrie tag-ul, validează linia, setează dirty pe 0, încarcă date de simulare. Merge în READ\_ALLOC sau WRITE\_ALLOC, în funcție de operație.
* READ\_ALLOC / WRITE\_ALLOC: Execută efectiv citirea/scrierea pe linia alocată, în caz de miss. Marchează hit = 0, miss = 1. Merge în UPDATE\_LRU.
* DONE: Finalizează operația curentă, revine în IDLE

TESTBENCH

Acest testbench validează funcționarea unui cache controller 4-way set associative implementat în Verilog, evaluând atât comportamentul logic al operațiilor read/write, cât și performanța sistemului prin măsurarea hit/miss rate-ului și AMAT-ului.

**Structura generală**:

* Inițializează semnale pentru CPU (clk, rst, read, write), adresă, date și control (cache\_ready, MM\_ready).
* Instanțiază modulul cache\_controller (uut) și monitorizează semnalele hit, miss, read\_data, state\_out.
* Include o sarcină (task) pentru resetarea cache-ului (restart\_cache).
* Include un bloc always @(posedge clk) care contorizează accesările, hit-urile și miss-urile.

**Testele executate**:

* + READ HIT – verifică dacă datele sunt citite corect din cache (way0).
  + WRITE HIT – scrie date într-o linie validă de cache și verifică actualizarea.
  + READ MISS (way invalid) – forțează un miss prin invalidarea unei linii (way1).
  + WRITE MISS (way invalid) – verifică scrierea în cache prin alocare.
  + WRITE MISS cu linie dirty (way3) – testează write-back corect înainte de înlocuire.
  + READ MISS cu linie dirty (way4) – verifică alocarea după scrierea în memorie.

**Măsurători de performanță**:

* La finalul simulării, se calculează:
  + Total accesses – totalul operațiilor read și write.
  + Hits, Misses – numărul de accesări corecte/respinse din cache.
  + AMAT (Average Memory Access Time) – în format fix, cu 3 zecimale (x1000).
* Se presupune:
  + Latency cache = 1 ciclu
  + Latency memorie = 100 cicluri

**Acest testbench este esențial pentru:**

* Validarea corectitudinii operațiilor din cache (hit/miss, write-back, allocate).
* Măsurarea performanței și optimizarea politicilor (LRU, write-back).

A screenshot of a computer program

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

exemplu wave-uri dupa terminarea primelor două teste de hit