4

Memorii şi ierarhii de memorii

1. Prezentare teoretică

Memoria reprezintă una din componentele esenţiale ale unui calculator numeric. Rolul său este de a memora programe şi date. La modul general, memoria se împarte în memorie principală şi memorie secundară. Memoria principală stochează informaţiile şi datele curente necesare procesorului cât timp memoria secundară memorează date care nu sunt folosite în mod curent.

Subsistemul de memorie al unui calculator numeric este văzut ca o ierarhie de module de memorie. La baza ierarhiei se află memoria auxiliară/secundară (lentă dar de capacitate foarte mare, de ordinul zecilor de GB), iar în vârful ierarhiei se află memoria **cache** (foarte rapidă dar de capacitate foarte mică, de ordinul KB).

Memoria cache este o memorie specială, utilizată pentru mărirea vitezei de prelucrare a procesorului. Aceasta este dispusă între procesor şi memoria principală în vederea compensării diferenţei de viteză dintre cele două componente. Viteza de funcţionare a memoriei cache este foarte apropiată de viteza de funcţionare a procesorului, acesta fiind un motiv pentru care, de obicei, această memorie este încapsulată împreună cu procesorul.

Memoria principală

Este o memorie cu un timp de acces de aproximativ 700ns. Capacitatea ei variază de la 32MB până la 512MB. Tehnologia folosită în construcţia acestui tip de memorie se bazează pe circuite integrate semiconductoare. Circuitele integrate RAM pot opera în două moduri:

1. ***static*** - memoria statică RAM este formată din bistabile care memorează informaţia în formă binară. Informaţia rămâne disponibilă atât timp cât circuitul este alimentat cu curent electric.

2. ***dinamic*** - memoria dinamică RAM, memorează informaţia binară în formă de sarcini electrice ce sunt aplicate unor condensatori. Condensatorii, realizaţi cu ajutorul tranzistoarelor MOS, tind să se descarce în timp şi astfel apare riscul ca informaţia să se distrugă. Pentru a elimina acest inconvenient, în aceste memorii se realizează un ciclu de reîmprospătare a informaţiei la intervale de câteva milisecunde.

Memoria ***ROM*** (Read Only Memory) este o memorie cu acces aleator şi este utilizată pentru memorarea programelor, care sunt rezidente permanent în calculator şi pentru stocarea tabelelor de constante definite de către producătorul calculatorului. Tot în această memorie se păstrează şi programul iniţial de pornire a calculatorului (bootstrap loader) responsabil cu lansarea sistemului de operare. Conţinutul acestei memorii se păstrează chiar dacă circuitul nu este alimentat. Singura metodă de a şterge informaţia dintr-un astfel de circuit constă în iradierea sa cu raze ultraviolete.

Simbolul şi tabela de funcţionare a unui circuit de memorie RAM sunt prezentate în figura 4.1.

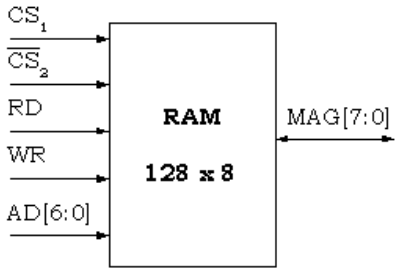

Tabela de funcţionare

CS_1	$\overline{CS_2}$	RD	WR	Funcţionare	MAG Magistrală de memorie
0	0	x	x	inhibată	Z
0	1	x	x	inhibată	Z
1	0	0	0	inhibată	Z
1	0	0	1	scriere	introducere date
1	0	1	x	citire	citire date
1	1	x	x	inhibată	Z

Figura 4.1: Memorie RAM 128 x 8.

Odată ce proiectantul unui calculator dispune de modulele de memorie fizice, el trebuie să asigneze memoriilor RAM sau ROM întreaga cantitate de memorie, determinată în prealabil, necesară pentru rularea aplicaţiilor. Adresarea memoriei de către procesor este stabilită pe baza unei tabele în care se specifică adresele de memorie asignate fiecărui dispozitiv/tip de memorie. Această tabelă poartă denumirea de "harta adreselor de memorie". Modalitatea de construire a acestei tabele este prezentată prin intermediul unui exemplu.

Exemplu Se presupune că un calculator necesită 512 octeţi de RAM şi 128 octeţi de ROM. Pentru implementare, la dispoziţia proiectantului există circuitele RAM şi ROM cu capacitate de 128 x 8 fiecare. Numărul circuitelor de memorie este limitat la 5. Harta adreselor de memorie este prezentată în tabelul 4.1.

Componenta	Adresa hexazecimală	Magistrala de adrese									
		10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
RAM_1	0000 - 007F	0	0	0	x	x	x	x	x	x	x
RAM_2	0080 - 00FF	0	0	1	x	x	x	x	x	x	x
RAM_3	0100 - 017F	0	1	0	x	x	x	x	x	x	x
RAM_4	0180 - 01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x	x

Tabelul 4.1: Harta adreselor de memorie.

Memoria totală este: 512 + 128 = 640 octeţi. Pentru a accesa o memorie de 640 octeţi este necesar un cuvânt-adresă de 10 biţi. Astfel, dacă se va considera o magistrală de adrese de 16 biţi, primii 6 biţi vor fi întotdeauna egali cu 0. Având în vedere că memoria RAM ocupă 512 octeţi rezultă necesitatea utilizării a 9 linii de adresă. Deoarece sistemul conţine si memorie RAM si memorie ROM, o linie de adresă este necesară pentru a se realiza o distincţie între ele. Deci în total vor fi 10 linii de adresă ocupate. Restul de 6 linii de adresă vor fi tot timpul 0. Pentru uşurinţa proiectării, valoarea de pe magistrala de adrese este exprimată în hexazecimal.

Memoria asociativă

Memoria asociativă se bazează pe conceptul că timpul de căutare a unei informaţii date într-o memorie este redus considerabil dacă datele memorate pot fi identificate după conţinut şi nu după adresă.

Când se scrie un cuvânt într-o memorie asociativă, nici o adresă nu este furnizată. Memoria este capabilă să determine locaţia goală şi neutilizată în vederea memorării noului cuvânt. La citirea unui cuvânt dintr-o memorie asociativă, conţinutul cuvântului sau o parte a cuvântului este specificată. Memoria localizează toate cuvintele care se potrivesc cu conţinutul specificat şi ele vor fi marcate pentru citire.

Organizarea hardware a unei memorii asociative este prezentată în figura 4.2.

În figura 4.2 registrele *A* şi *K* sunt registre de *n* biţi. Registrul M, registrul de potrivire, este de *m* biţi, unul pentru fiecare cuvânt de memorie.

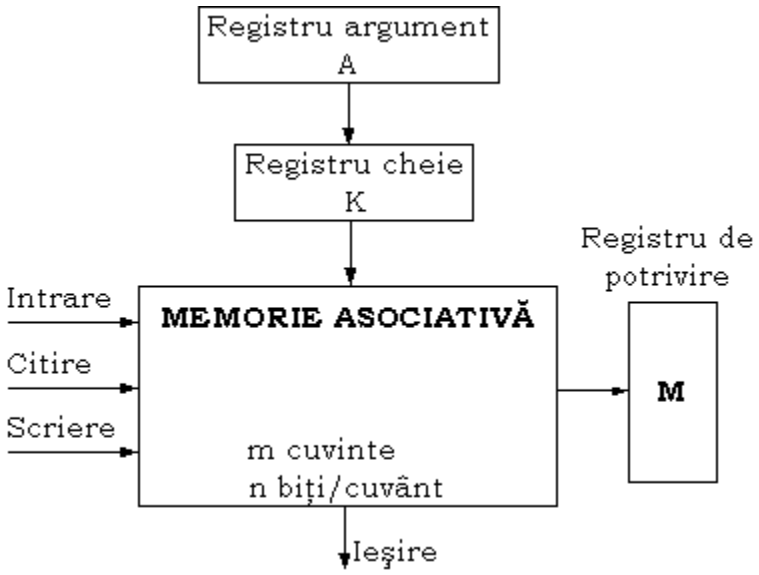

Figura 4.2: Organizarea hardware a unei memorii asociative

Fiecare cuvânt din memorie este comparat cu conţinutul registrului *A*. Cuvintele care se potrivesc setează un bit corespunzător în registrul *M* a cărui valoare este 1. În final, registrul *M* va conţine doar biţii care indică ce cuvinte s-au potrivit. Citirea se realizează printr-un acces secvenţial la memorie, doar pentru cuvintele ai căror biţi corespunzători în registrul *M* sunt 1.

Registrul *K* este un registru de mascare în vederea stabilirii unui câmp particular sau a unei chei din cuvântul memorat în registrul *A*. Comparaţia întregului cuvânt de memorie cu conţinutul registrului *A* se va realiza doar dacă registrul *K* conţine toţi biţii egali cu 1. În mod normal se compară doar cuvintele care au biţii 1 în poziţiile în care există 1 în registrul *K*.

Organizarea hardware a unei celule de memorie asociative precum şi relaţia dintre vectorul memorie şi registrele externe sunt prezentate în figura 4.3.

Aşa cum se observă în figura 4.3, fiecare bit A_j din registrul *A* este comparat cu toţi biţii coloanei j din matricea vector, dacă $k_j = 1$, $\forall\ j = \overline{1 \ldots n}$. Dacă toţi biţii registrului *K* sunt egali cu biţii din cuvântul *i,* atunci se setează $M_i = 1$, altfel $M_i = 0$.

Circuitul de potrivire logică din figura 4.3 se poate deduce matematic. Cuvântul *i* este egal cu conţinutul registrului *A* dacă:

$$A_j \quad = \quad F_{ij} \quad \forall\ j = \overline{1 \ldots n} \tag{4.1}$$

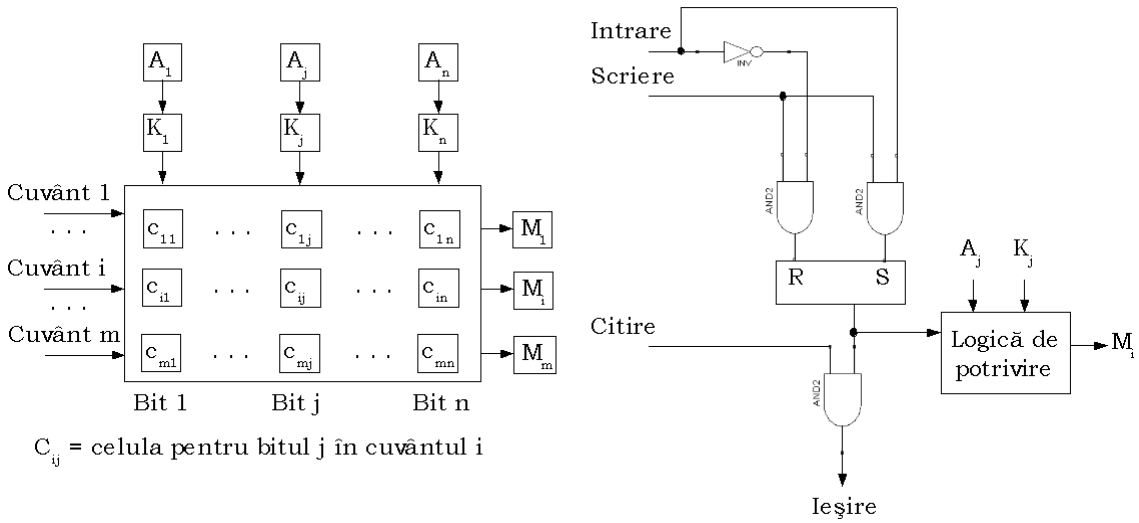

Figura 4.3: Organizarea hardware a unei celule de memorie

Egalitatea a doi biţi poate fi exprimată ca $x_j = A_j \cdot F_{ij} + \overline{A_j} \cdot \overline{F_{ij}}$, unde $x_j = 1$ dacă toţi biţii de pe poziţia *j* sunt egali. Pentru ca bitul M_i să fie l trebuie respectată următoarea egalitate:

$$M_i = x_1 \cdot x_2 \cdot x_3, ..., x_n \tag{4.2}$$

În funcţie de valoarea lui K_j, există două situaţii posibile:

1. $k_j = 0$, atunci biţii corespunzători A_j şi F_{ij} nu trebuie comparaţi;

2. $k_j = 1$, atunci biţii corespunzători A_j şi F_{ij} trebuie comparaţi.

Se poate concluziona că:

$$x_j + \overline{k_j} = \begin{cases} x_j & daca \quad k_j = 1 \\ 1 & daca \quad k_j = 0 \end{cases} \tag{4.3}$$

Ecuaţia (4.2) se rescrie ca:

$$M_i = \left(x_1 + \overline{k_1}\right) \cdot \left(x_2 + \overline{k_2}\right) \cdot \left(x_3 + \overline{k_3}\right) \cdot ... \cdot \left(x_n + \overline{k_n}\right) \tag{4.4}$$

sau într-o formă mult mai compactă:

$$M_i = \prod_{j=1}^{n} \left(A_j \cdot F_{ij} + \overline{A_j} \cdot \overline{F_{ij}} + \overline{k_j} \right) \quad \forall\, i = \overline{1, m} \tag{4.5}$$

În vederea realizării operaţiei de citire din memoria asociativă, se scanează conţinutul registrului *M* şi se citeşte doar un bit al său la un moment dat, obţinând astfel o secvenţă de cuvinte aflate în memoria asociativă şi care se potrivesc cu cuvântul de memorie.

În cazul operaţiei de scriere apar următoarele situaţii posibile:

1. *memoria este vidă* - în acest caz scrierea poate fi realizată prin adresarea fiecărei locaţii într-o anumită secvenţă. Astfel memoria devine o memorie cu acces aleator în cazul scrierii şi o memorie de tip adresabilă după conţinut în cazul operaţiei de citire. Avantajul este că adresele cuvintelor de intrare pot fi decodificate ca în cazul memoriilor cu acces aleator, rezultând astfel numai *d* linii de adrese în loc de *m* ($m = 2^d$).

2. *suprascrierea unui cuvânt* - este situaţia în care memoria este complet ocupată şi se doreşte introducerea unui nou cuvânt în memorie. În această situaţie se mai adaugă un registru special (registrul de etichete) care memorează cuvintele active şi cele inactive asignând valoarea l pentru cuvintele active şi 0 pentru cuvintele inactive. Numărul biţilor acestui registru trebuie să fie acelaşi cu numărul de cuvinte din memorie. Se înlocuiesc doar cuvintele cu eticheta 0 utilizând în acest scop un algoritm FIFO, LRU etc. După realizarea operaţiei de suprascriere, eticheta corespunzătoare va avea valoarea 1.

Memoria cache

Memoria cache este o memorie de mică capacitate dar cu o viteză de accesare foarte mare. Când CPU doreşte să acceseze memoria principală, întâi se examinează memoria cache. Dacă, cuvântul solicitat se găseşte stocat în memoria cache, atunci el este preluat de către procesor, din această memorie. În caz contrar este accesată memoria principală, se citeşte cuvântul dorit şi este stocat apoi în memoria cache.

Pentru a respecta principiul "localităţii spaţiale", se transferă din memoria principală în memoria cache nu doar cuvântul solicitat de procesor, ci un bloc de cuvinte care conţine cuvântul solicitat. Dimensiunea blocului transferat variază de la o arhitectură la alta, dar uzual, blocul transferat are o dimensiune cuprinsă între l şi 16 cuvinte.

Performanţa unei memorii cache este măsurată în termeni de "hit ratio". Dacă, cuvântul solicitat de CPU se află memorat în memoria cache, atunci este vorba de un succes ***HIT,*** iar altfel, de un insucces ***MISS***. ***HIT RATIO*** se defineşte ca fiind raportul dintre numărul de succese şi numărul total de referiri CPU la memorie.

Procesul de amplasare a datelor citite din memoria principală, în memoria cache se numeşte *mapare* . Uzual în proiectarea şi organizarea memoriilor cache se folosesc

următoarele tipuri de proceduri de mapare:

1. mapare complet asociativă;
2. mapare directă;
3. mapare asociativă pe mai multe căi.

Pentru o descriere cât mai completă a fiecărui mod de mapare se va utiliza următorul exemplu.

Exemplu Se presupunem existenţa unui calculator care posedă o memorie principală cu capacitatea de 32k × 12 şi o memorie cache cu capacitatea 512 cuvinte × 12. Procesorul calculatorului poate comunica cu ambele memorii. El trimite o adresă de 15 biţi către memoria cache. Dacă se obţine HIT, CPU acceptă 12 biţi de date de la memoria cache. În caz de MISS, CPU citeşte cuvântul din memoria principală şi cuvântul va fi apoi memorat în memoria cache.

Maparea complet asociativă

Acest tip de mapare este cea mai flexibilă şi mai rapidă organizare a memoriei cache. Organizarea acestui tip de memorie poate fi observată în figura 4.4. În memoria complet asociativă sunt memorate adresele şi conţinutul cuvintelor de memorie.

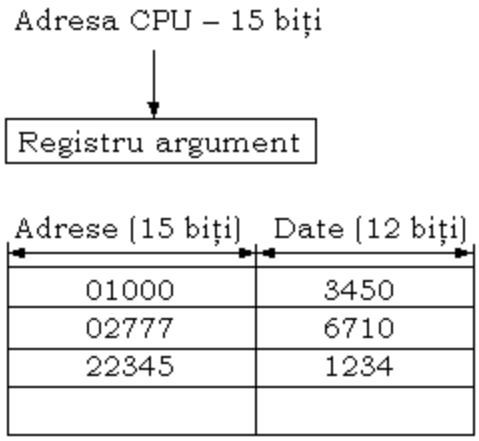

Figura 4.4: Maparea asociativă.

Adresa CPU de 15 biţi este memorată în registrul argument/descriptor şi apoi se încearcă depistarea unei potriviri de adrese dintre această adresă şi o adresă din memoria cache. Dacă procesul se încheie cu HIT, atunci datele de 12 biţi sunt citite şi transmise către CPU.

Dacă procesul se încheie cu MISS, atunci perechea adresă-dată este citită din memoria principală, trimisă la procesor şi memorată în memoria cache. În cazul cel mai

defavorabil, memoria este complet ocupată şi noua pereche adresă-dată trebuie suprascrisă în locul altei perechi stabilită de către algoritmul de replasare utilizat (uzual se folosesc algoritmii LRU şi FIFO).

Mapare directă

O versiune mai ieftină de memorie cache este prezentată în figura 4.5. În cadrul acestei implementări, adresa CPU se împarte în două câmpuri: INDEX (numărul biţilor acestui câmp este egal cu numărul biţilor necesari pentru a accesa memoria cache) şi ETICHETĂ. În concluzie, cei 15 biţi ai adresei CPU se împart după cum urmează: 6 biţi pentru câmpul ETICHETĂ şi 9 biţi pentru câmpul INDEX.

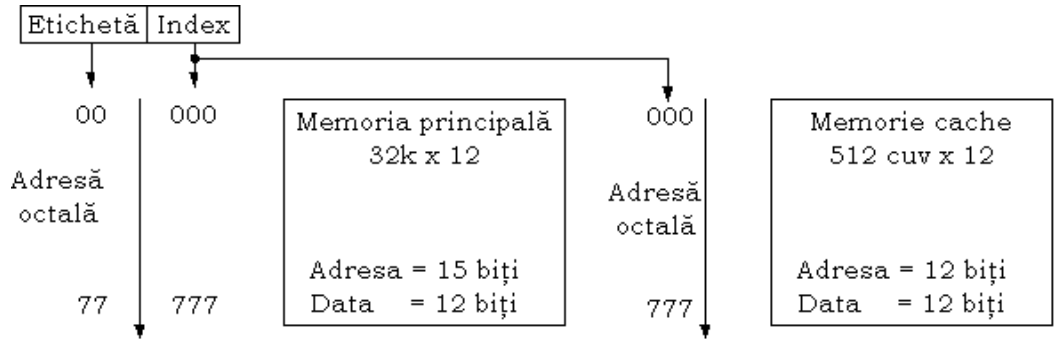

Figura 4.5: Maparea directă.

Fiecare cuvânt memorat în memoria cache este format din câmpul de date şi eticheta asociată. Atunci când o cerere este lansată, câmpul INDEX este utilizat pentru adresa de acces a memoriei cache. Câmpul ETICHETĂ al adresei CPU este comparat cu "eticheta" cuvântului citit din memoria cache.

Dacă procesul returnează MISS, cuvântul cerut se citeşte din memoria principală şi este memorat în memoria cache împreună cu o nouă etichetă, înlocuindu-se astfel valoarea anterioară.

Dezavantajul major constă în scăderea valorii HIT RATIO dacă două sau mai multe cuvinte ale căror adrese au acelaşi index dar etichete diferite sunt accesate în mod repetat.

Un exemplu numeric este prezenata în figura 4.6.

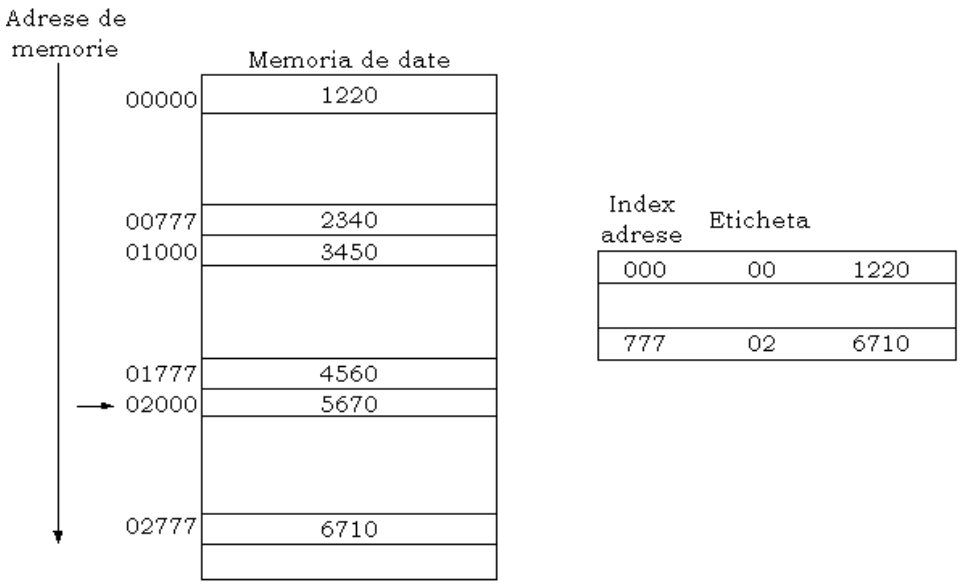

Figura 4.6: Exemplu mapare directă.

Cuvântul aflat la adresa 00000 este memorat în memoria cache. Procesorul doreşte să acceseze cuvântul aflat la adresa 02000. Indexul de adrese este 000 şi el va fi utilizat pentru accesarea memoriei cache. În urma comparaţiei dintre cele două etichete, se constată că rezultatul întors este MISS, deoarece eticheta memoriei cache este 00, iar eticheta adresă este 02.

Se accesează memoria principală şi cuvântul 5670 este transferat către CPU. Cuvântul din memoria cache de la 000 este replasat cu eticheta 02 şi data 5670.

Maparea asociativă pe mai multe căi

În cadrul acestei metode de organizare a memoriei cache, fiecare cuvânt al memoriei cache poate memora unul sau mai multe cuvinte din memorie sub acelaşi index de adresă. În figura 4.7 este prezentat un exemplu de memorie cache cu o organizare asociativă pe mai multe căi.

Index adrese	Eticheta	Date	Eticheta	Date
000	01	3450	02	5670
777	02	6710	00	2340

Figura 4.7: Maparea asociativă pe mai multe căi.

Fiecare index de adresă referă două cuvinte de date şi etichetele asociate lor. Fiecare etichetă necesită 6 biţi, iar fiecare cuvânt 12 biţi, rezultând lungimea cuvântului de

$2 \cdot (6+12) = 36$ biţi. Un index de adrese de 9 biţi poate acoperi 512 cuvinte deci dimensiunea memoriei cache este de 512 x 36. Această memorie cache poate acoperi un număr de 1024 cuvinte ale memoriei principale deoarece un cuvânt din memoria cache conţine două cuvinte dată.

Când CPU generează o cerere de memorie, valoarea index a adresei este utilizată pentru accesarea memoriei cache. Câmpul etichetă din adresa CPU este comparat cu cele două etichete din memoria cache în vederea stabilirii unei potriviri. Compararea logică este realizată printr-o căutare asociativă de etichete într-o mulţime. În caz de MISS, algoritmii de înlocuire cei mai frecvent utilizaţi sunt FIFO şi LRU.

Scrierea în memoria cache

În cazul în care CPU generează semnal de scriere în memoria principală, există două metode care se pot aplica:

1. reactualizarea informaţiei din memoria principală în paralel cu reactualizarea informaţiei din memoria cache dacă cuvântul se găseşte în memoria cache, la adresa specificată. Această procedură se numeşte WRITE-THROUGH.

2. WRITE-BACK. În cadrul acestei metode, doar locaţia memoriei cache este reactualizată în cadrul operaţiei WRITE. Locaţia reactualizată este marcată şi doar în caz de suprascriere a informaţiei se va face reactualizarea locaţiei memoriei principale.

2. Desfăşurarea lucrării

1. Se va utiliza un simulator de memorie cache pentru a simula diferite organizări ale memoriei cache pentru primele 1 milion de referinţe în monitorizarea execuţiei programului ***gcc***. Sunt disponibile (http://www.csit-sun.pub.ro/resources) atât ***dinero*** (simulatorul de memorie cache), cât şi ***gcc.*** Se presupune o memorie cache de instrucţiuni de 32kB şi o memorie cache de date de 32kB folosind aceeaşi organizare. Să se aleagă cel puţtin două tipuri de asociativităţi şi două dimensiuni de bloc. Să se deseneze o diagramă ce prezintă organizarea unei memorii cache cu cea mai bună rată de eşec.

2. Se va studia influenţa folosirii unui nivel secundar de memorie cahe asupra performanţei unui procesor. Se presupune existenţa unui procesor cu un CPI (numărul de cicluri/instrucţiune) de bază de 1.0 şi o frecvenţă de ceas de 500MHz. Se consideră că toate referinţele la memorie vor avea succes în memoria cache primară. Memoria principală are un timp de acces de 200ns, incluzând aici şi timpul necesar tratării cazurilor de eşec. Frecvenţa de eşec pe instrucţiune în memoria cache primară este de 5%. Care va fi creşterea de viteză

a maşinii dacă este adăugat un nivel suplimentar de memorie cache cu un timp de acces de 20ns, atât pentru eşec, cât şi pentru succes şi suficient de mare pentru a reduce frecvenţa de eşec la memoria principală la 2%?

3. Probleme propuse

1. Pe baza hărţii adreselor de memorie stabilită în cadrul exemplului l, să se proiecteze circuitul care realizează conexiunea memoriei cu CPU.

2. Pe baza organizării hardware a unei celule de memorie asociativă prezentată în figura 4.3, să se proiecteze o memorie asociativă cu m = 4 şi n = 3. Să se realizeze simularea operaţiilor de scriere/citire în/din memoria asociativă proiectată.

3. Se reconsideră exemplul prezentat în figura 4.6. În cadrul exemplului mărimea blocului de memorie are dimensiunea de l cuvânt. Cum arată aceeaşi proiectare dacă se utilizează un bloc de capacitate 8 cuvinte ?

4. Enunţaţi cel puţin un avantaj al folosirii metodei WRITE-THROUGH de scriere în memoria cache.

5. Folosind limbajul de asamblare pentru MIPS şi simulatorul SPIM exemplicaţi toate modurile de adresare la memorie cunoscute. Spre exemplu, pentru a exemplifica modul de adresare indirectă cu autodecrementare se poate scrie programul MIPS din figura 4.8.

```
# adresare indirectă cu autodecrementare
      .text
      la          $a1,  0x0040001c
      sub         $a1, $a1, 0x00000004
      lw          $t1, ($a1)
      lw          $t2, ($t1)
pc:   not         $t1, $t2
adr:  .word       0x00400020
      li          $v0, 1
op:   .word 7
      move $a0, $t1
      syscall
end:  mop
```

Figura 4.8: Program MIPS pentru exemplificarea modului de adresare indirectă cu autodecrementare a memoriei principale.