Cap8_1. **Organizarea memoriei in sistemele de calcul.**

1.Generalitati.

Sistemele de calcul pot fi analizate ca si sistemele de productie, indiferent de modul in care opereaza: tren de lucrari (batch processing), timp partajat (time sharing) etc.

In ambele situatii la terminalele sistemului sau la echipamentele de I/E se formeaza cozi de asteptare. Unul din scopurile urmarite in exploatarea sistemelor de calcul este acela de a maximiza numarul de lucrari, care sunt introduse in sistem si executate, in unitatea de timp.

Numarul de lucrari ponderate dupa complexitate, executate in unitatea de timp, poarta numele de productivitate (throughput).

Pentru marirea productivitatii unui sistem se pot folosi mai multe metode: cresterea vitezei procesorului central, cresterea vitezei de transfer a datelor prin sistem (prin sporirea vitezei de lucru a memeoriei interne, a echipamentelor de I/E etc.).

In general se cauta sa se realizeze un echilibru intre posibilitatile mentionate. De regula, se constata ca sistemele sunt limitate din punctul de vedere al I/E (I/O bound). In figura 1 se prezinta organizarea generala a unui sistem de calcul.

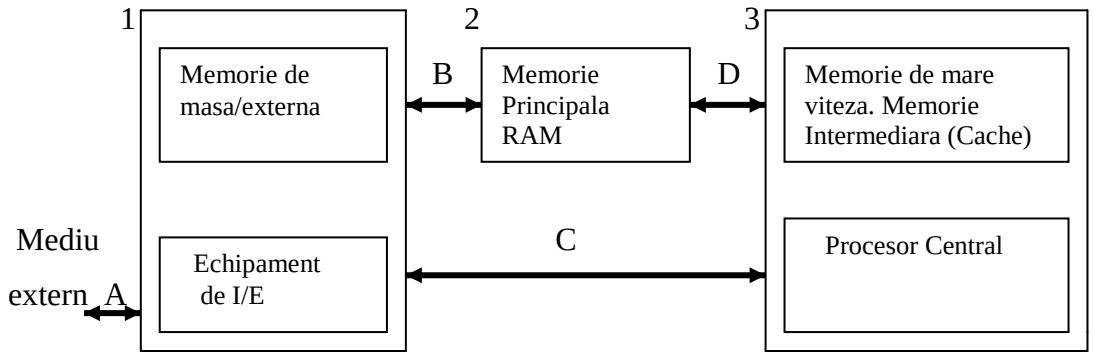

Fig. 1. Structura generala a unui sistem de calcul.

Cele trei unitati evidentiate se caracterizeaza prin intervale de timp de raspuns diferite. In fiecare unitate se afla introdusa cate o memorie. Capacitatea si viteza oricarui tip de memorie variaza invers una in raport cu cealalta.

Astfel, in unitatea 3 se afla plasata o memorie intermediara/tampon/cache, de mare viteza, capabila sa execute citiri/scrieri intr-o perioada de tact. Aceasta memorie intermediara(Cache) poate avea o capacitate care poate, de exemplu, depasi 256 ko.

Memoria interna, de tip RAM, are o viteza moderata (timpi de acces de ordinul a 60 - 350 ns) si o capacitate cuprinsa intre 0,5 - 4 Go.

Memoria secundara/externa poate fi realizata in mai multe tehnologiie care asigura timpi de acces mai mari de 20 ms si capacitati de ordinul a 30- 250Go.

In figura 1, unitatea 3 este proiectata in ideea de a minimiza costul operatiilor, de a maximiza numarul de operatii in unitatea de timp. Aceste deziderate se realizeaza atunci cand toate operatiile se efectueaza in cadrul blocului 3. Performantele sistemului vor depinde de debitul informatiilor pe magistralele B,C si D.

Magistrala C va fi utilizata mai rar, pentru a evita reducerea vitezei unitatii 3, datorita interactiunii cu unitatea 1.

In modul tren de lucrari operarea va avea loc prin transferuri pe magistralele A, B, D. Transferurile pe magistrala A pot avea loc simultan cu transferurile pe magistralele B si D.

Din figura 1 se poate constata existenta unei ierarhii in organizarea memoriei in sistemele de calcul. Schimbul de informatii intre diversele niveluri, in cadrul ierarhiei, trebuie sa aiba

un caracter transparent pentru utilizator. Aceasta se poate reliza folosind tehnici asociative pentru implementarea memoriilor tampon si virtuala.

2. Memorii Asociative.

Intr-o memorie asociativa (MA) cuvantul/data se obtine prin furnizarea unui descriptor/identificator, in locul unei adrese, care specifica locatia acelui cuvant/data in memorie.Printr-un proces de cautare se stabileste eventual identitatea intre descriptorul furnizat si descriptorul asociat unei date din memorie, care poate fi astfel citita.

Descriptorul face parte din fiecare cuvant memorat sau este stocat separat.

Intr-o memorie asociativa informatia poate fi astfel organizata incat unui descriptor sa-i corespunda mai multe cuvante. In cazul de fata, fiecarui descriptor ii va corespunde un singur cuvant.

Cautarea secventiala este extrem de lenta, in timp ce cautarea paralela implica o retea combinationala extrem de complexa, ceea ce duce la o limitare a dimensiunilor memoriei asociative

In cele ce urmeaza se va examina proiectarea la nivel de secventa AHPL, a unei memorii asociative, cu cautare combinationala avind 2^n cuvinte x m biti/cuvant. Limitarile impuse de costuri, conduc la valori n = 10 - 18.

In figura 2 se prezinta organizarea memoriei asociative, in care:

- TA reprezinta tabloul descriptorilor cu 2^n cuvinte × r biti;
- MA constituie tabloul datelor cu 2^n cuvinte x m biti.

Fiecarui descriptor ii corespunde un singur cuvant in MA, astfel ca n < r.

Intr-o aplicatie cu caracter educational, 2^n reprezinta numarul cuvintelor din memoria intermediara, iar 2^r reprezinta numarul cuvintelor din memoria principala

Spre exemplu n = 8, iar r = 18.

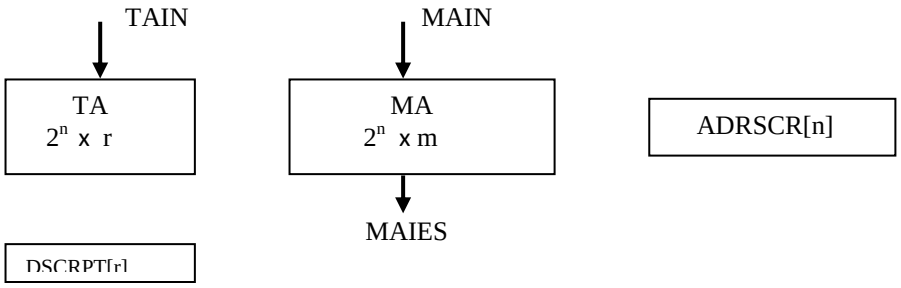

Fig. 2. Organizarea generala a unei memorii asociative.

Informatiile se pot stoca in tablourile de memorie TA si MA, de la vectorii de intrare TAIN si MAIN, la adresa specificata in registrul ADRSCR[n].

Datele se citesc asociativ din MA, adica continutul cuvantului din MA, al carui descriptor asociat (plasat in TA) coincide cu continutul registrului DSCRPT[r], daca exista vreunul, va fi fortat ca vector de iesire MAIES.

Operatiile individuale de citire/scriere pot fi exprimate ca pasi AHPL distincti. Astfel, in cadrul unei **scrieri obisnuite**, in continutul unei locatii din ansamblul tablourilor de memorare TA,MA, de la adresa specificata in ADRSCR[n], se inscrie un cuvant cu lungimea n + m.

TA,MA * DCD(ADRSCR) ← TAIN,MAIN

Citirea asociativa se reprezinta in AHPL astfel:

MAIES = BUSFN(MA;ASOC(DSCRPT;TA))

Scrierea asociativa este descrisa dupa cum urmeaza:

MA * ASOC(DSCRPT;TA) ← MAIN

Vectorul MAIES este disponibil la iesirea de date a memoriei asociative si poate fi fortat intr-un registru.

Selectia cuvintelor din MA se realizeaza cu ajutorul unei unitati logice combinationale ASOC si nu prin intermediul unui decodificaor, ca in cazul unei memorii RAM standard.

Unitatea logica combinationala ASOC are 2^n iesiri binare, cate una pentru fiecare cuvant din ansamblul TA,MA. Bitul ASOCi este 1 daca continutul cuvantului i, din tabloul TA, coincide cu continutul registrului DSCRPT, celalti biti $ASOC_j$ fiind egali cu zero ($j = 0,1,..,2^{n-1}$; $j \neq i$) Daca nu exista coincidenta toate cele 2^n ranguri ale vectorului ASOC(DSCRPT;TA) vor fi egale cu zero. Intrucat fiecare bit al vectorului ASOC(DSCRPT;TA) este independent, in descrierea AHPL a unitatii logice combinationale se va folosi o instructiune de conexiune de 2^n ori.

UNIT: ASOC(DSCRPT;TA)

INPUTS: DSCRPT[r]; TA[2^n;r]

OUTPUTS: ASOC[2^n]

1. i <= 0
2. ASOCi = $\cap/(\overline{DSCRPT \oplus TA^i})$
3. i <= i + 1
4. =>(i < 2^n)/(2)

END.

Se poate constata ca ASOCi = 1, daca si numai daca DSCRPT = TA^i .

3. Memoria Intermediara(Cache, Scratch-pad).

Memoriile intermediare sunt memorii de mare viteza si de capacitate moderata/mica, in care se stocheaza fragmente de program, date si rezultate intemediare, frecvent folosite de catre procesor.

Memoria intermediara este amplasata intre procesor si memoria principala (Fig. 3.).

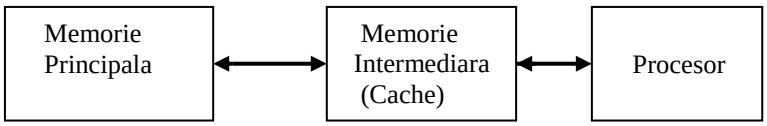

Fig. 3. Amplasarea memoriei intermediare.

Este cunoscut faptul ca procesoarele moderne dispun in mod curent de un set de registre generale, care pot fi adresate in maniera in care sunt adresate locatiile de memorie. De exemplu, unitatea centrala de prelucrare a calculatorului Felix 5000 avea un set de 16 registre generale, a cate 4 octeti fiecare. Ele erau adresate la nivel de octet, semicuvant (16 biti) sau cuvant (32 de biti).

In cazul in care o instructiune specifica o adresa efectiva de date, in zona 0 - 64 adresarea se efectua la registrele generale, in timp ce o adresa de instructiune, cuprinsa in acelasi spatiu de adresare, conducea la un acces la memorie. Se constata astfel, ca cele 16 registre generale se suprapun, ca adrese de octeti, cu primele 64 locatii de memorie.

Alte calculatoare (IBM 360/370) dispuneau de instructiuni speciale, de format scurt, pentru adresarea registrelor generale. Astfel,erau utilizate instructiuni cu lungimi diferite de tip R-R, R-M si M-M, pentru a specifica adrese de registre generale, adrese de registru general si de locatie de memorie si adrese de locatii de memorie. Numarul registrelor generale este limitat. In cazul instructiunilor cu o adresa programarea se complica.

Utilizarea unei memorii intermediare, de mare viteza, plasata intre procesor si memoria principala, in care se pot stoca atat instructiuni, cat si date, frecvent folosite, constituie una din solutiile curent intilnita in unitatile centrale moderne. Avand un timp mic de acces, memoria intermediara permite ca o operatia de citire/scriere sa se efectueze intr-o singura perioada de tact, ceea ce face ca procesorul sa nu intre in stari de asteptare (cel mult o penalizare de operioada de tact).

Prezenta memoriei intermediare va fi vizibila pentru utilizator numai prin sporirea vitezei de lucru a unitatii centrale. Transferul unor zone de cod sau date intre memoria principala si memoria intermediara se realizeaza automat, prin hardware, fara interventia utilizatorului.

Pentru ca performantele sietemelor de calcul sa nu fie afectate de viteza limitata a memoriei

principale se folosesc mai multe tehnici:

- pastrarea raportului intre numarul de adresari la memoria principala si memoria intermediara, cat mai mic posibil;
- suprapunerea adresarilor la memoria principala cu alte activitati ale procesorului central si anticiparea adresarilor la memoria principala;
- organizarea memoriei principale sub forma de blocuri, cu porturi distincte de acces si cu adrese intretesute, ceea va asigura o viteza medie de transfer mai mare decat cea corespunzatoare accesului la un bloc.

Solutiile mentionate mai sus nu sunt complet independente.

In continuare se va examina implementarea unei memorii intermediare. Se considera relativ ridicat costul memoriei intermediare, astfel incat capacitatea ei nu poate creste fara a afecta indicele cost/performanta. Daca se cunosc in avans necesitatile de adresare la memoria principala, in paralel cu unele operatii aritmetice, procesorul poate lansa operatii de transfer al continutului unor zone din memoria principala in memoria secundara si invers. Asemenea operatii nu se pot controla in cadrul unor limbaje de nivel inalt.

In general se pleaca de la proprietetea de localitate a programelor, materializata prin existenta a numeroase cicluri/bucle cu caracter repetitiv. Daca aceste constructii pot fi plasate intr-o memorie intermediara de mare viteza, procesorul va lucra continuu, fara stari de asteptare.

Pentru a asigura transferul automat, transparent pentru utilizator, al informatiei intre memoria principala si memoria intermediara, se impune utilizarea unui mecanism bazat pe o memorie asociativa.

Se considera, in scop didactic, o memorie intermediara cu capacitate de 256 cuvinte de 32 se biti, organizata asociativ, care va opera in conjunctie cu o memorie principala lenta, cu o capacitate de 256 kcuvinte × 32 de biti. Organizarea generala a unei asemenea memorii este data in figura 4, in care se folosesc urmatoarele resurse:

- AM[18], registrul de adrese al memoriei principale;
- DM[32], registrul de date al memoriei principale;
- DSCRPT[18], registrul descriptorului, folosit ca registru de adrese, de catre memoria intermediara;
- DMI[32], registrul de date al memoriei intermediare;
- CU[256;8], ansamblul contoarelor de utilizare, pentru gestiunea numarului de utilizari al

fiecarui cuvint din memoria intermediara;

- TA[256;18], ansamblu de memorare a adreselor cuvintelor prezente in memoria intermediara;
- MI[256;32] memoria intermediara propriu-zisa, in care sunt plasate codurile/datele, aduse din memoria principala.

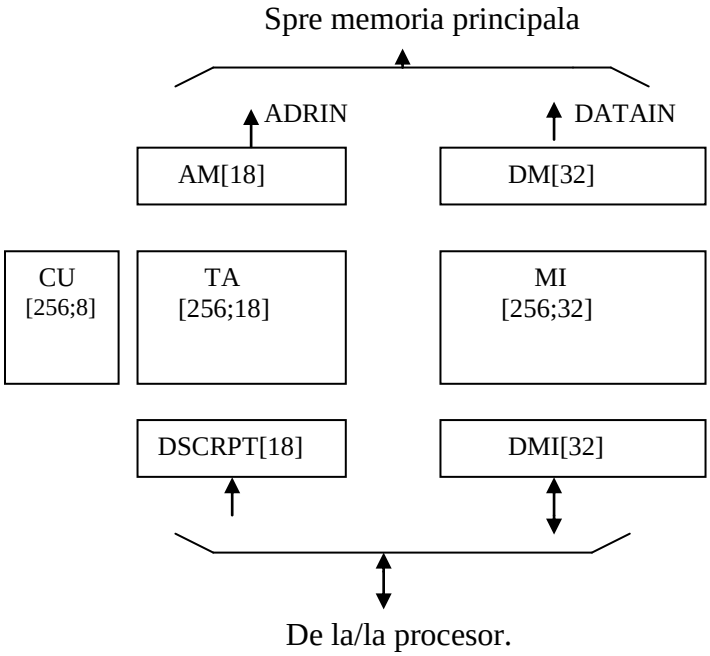

Fig. 4. Organizarea generala a memoriei intermediare.

Considerand ca procesorul furnizeaza o adresa insotita de o cerere de citire, adresa va fi plasata in registrul DSCRPT si va fi comparata, in paralel, cu continuturile tuturor locatiilor din TA. Daca intr-una din locatiile lui TA se gaseste o adresa egala cu continutul descriptorului, atunci celula de pe linia respectiva, din MT, va fi citita in DMI, iar contorul asociat, din tabloul CU, va fi fortat la valoarea 2^8 - 1, in timp ce toate celelalte contoare, din CU, vor fi decrementate cu o unitate, exceptie facand contoarele deja decrementate la valoarea zero (se foloseste in acest scop o functie logica combinationala DCZ(UC)). In mod asemanator se petrec lucrurile si la operatia de scriere in memoria intermediara. Operatiile prezentate mai sus se efectueaza sub controlul direct al procesorului. In cazul in care adresa plasata in DSCRPT nu se regaseste in nici una din celulele tabloului TA, se va ceda controlul unitatii de comanda a memoriei intermediare. Aceasta va cauta prima celula, cu cea mai putin frecventa utilizare, din memoria intermediara, folosind continutul tabloului contoarelor de utilizare, cat si o functie logica combinationala PRILIB(UC). Aceasta din urma va genera un vector cu 256 de biti, dintre care numai unul va fi egal cu 1. El va corespunde

locatiei din memoria intermediara al carei continut va fi salvat/stocat in memoria principala. In locatia data se va inscrie un nou cuvant, din memoria principala. Operatiile sunt complet transparente pentru utilizator.

Studii statistice au aratat ca probabilitatea ca un cuvant sa se gaseasca in memoria intermediara este cuprinsa intre 0,90 si 0,93 (hit ratio).

Dupa cum s-a mai aratat, deoarece accesul la memoria intermediara se efectueaza intr-o perioada de tact, procesorul va controla direct aceste operatii, in timp ce accesul la memoria principala, mai lenta, se va efectua sub controlul unitatii de comanda a memoriei intermediare.

Structura generala pentru memoria intermediara/memoria principala este data in figura 5.

Se constata ca memoria principala RAM opereaza asincron in raport cu procesorul si memoria intermediara/tampon. In acest scop se foloseste un semnal de stare: "ocupat".

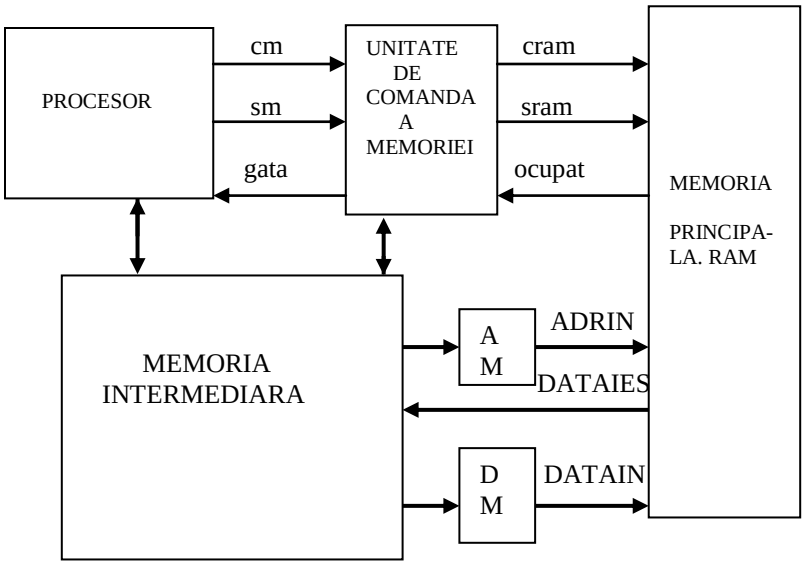

Fig. 5. Structura generala memorie intermediara/memorie principala

- cm si sm sunt semnalele de citire/scriere memorie generate de procesor, in cazul in care trebuie efectuat un acces la memoria principala;
- gata reprezinta semnalul de sfarsit de operatie de citire/scriere;
- cram si sram sunt semnalele de citire/scriere pentru RAM, elaborate de unitatea de comanda a memoriei intermediare;

- ocupat reprezinta un semnal de stare, generat de logicaRAM.

Secventa de citire din memorie, pentru procesor, se va modifica dupa cum urmeaza:

i. DSCRPT ← ADRESA

i+1.BUSTA = ASOC(TA;DSCRPT);

DMI * ($\cup$/$\overline{\text{BUSTA}}$) ← BUSFN(MI;BUSTA);

CU * $\overline{\text{BUSTA}}$← DECZ(CU);

CU * BUSTA← $\overline{8 \mid 0}$;

→($\cup$ /BUSTA)/(i+3)

i+2. $\overline{\text{cm}}$ = 1;

→($\overline{\text{gata}}$)/(i+2)

i+3. pasul urmator din secventa procesorului.

Pasul i+1. din secventa de mai sus asigura generarea unui vector logic BUSTA, cu 256 componente, de catre unitatea logica combinationala ASOC(TA;DSCRPT). In cazul in care exista o componenta egala cu 1, locatia cu acelasi indice din MI si CU va fi citita in DMI si respectiv - fortata la valoarea 2^8 - 1.

Celelalte locatii din CU vor suferi operatia de decrementare DCZ(CU), trecandu-se apoi la linia cu numarul i+3. Daca BUSTA nu are nici o componenta egala cu 1, toate contoarele de utilizare vor suferi operatia DCZ(UC) si se va trece la linia i+2 din secventa, amorsindu-se o operatie de citire din memoria principala.

Operatia de scriere in memorie capata urmatorul aspect:

j. DMI ←DATA; DSCRPT ← ADRESA;

j+1. BUSTA = ASOC(TA;DSCRPT);

MI * BUSTA← DMI;

CU * $\overline{\text{BUSTA}}$ ← DCZ(CU);

CU * BUSTA← $\overline{8|0}$;

→(∪/BUSTA)/ j+3)

j+2. sram = 1;

→$\overline{(gata)}$/(j+2)

j+3. pasul urmator din secventa procesorului.

In cazul in care cuvintul a carui adresa este plasata in DSCRPT nu se gaseste in MI, se impune lansarea unei operatii cu memoria principala, mai lenta, fapt care face ca procesorul sa astepte la pasii i + 2 , j + 2. Controlul este preluat de modulul unitatii de comanda al memoriei intermediare. Operarea modulului este descrisa de urmatoarea secventa AHPL:

MODULE: UNITATE MEMORIE INTERMEDIARA

INPUTS: cm; sm; DATAIES[32]; ocupat

OUTPUTS: cram; sram; gata; ADRESAIN[18]; DATAIN[32]

MEMORY: DMI[32]; DSCRPT[18]; AM[18]; DM[32]; CU[256;8]; TA[256;18];
MI[256;32];

BUSES: BUSTA[256}

1. →$\overline{(cm \cap sm)}$/(1)

2. BUSTA = PRILIB(CU);

AM ← BUSFN(TA;PRILIB(CU));

DM ← BUSFN(MI;PRILIB(CU))

3. scram = 1;

→$\overline{(ocupat)}$/(3)

Receptia comenzilor cm/sm arata ca trebuie gasit un spatiu in MI; un cuvant trebuie extras din MI si plasat in memoria principala. Pentru gasirea adresei acestui cuvant, se foloseste algoritmul LRU (Least Recently Used - cel mai putin recent utilizat), implementat prin unitatea logica combinationala PRILIB(CU), care examineaza continuturile contoarelor CU, oprindu-se la primul

contor cu valoarea cea mai mica (eventual zero) din intregul ansamblu.

Pe baza adresei contorului in cauza se extrag din tablourile de memorare TA si MI adresa si respectiv data, care urmeaza sa fie stocata in memoria principala. Temporar ele vor fi plasate in registrele AM si DM, generandu-se apoi comanda sram. Aceasta comanda se mentine pana cand se primeste raspunsul ocupat = 1.

4. →(ocupat, $\overline{\text{ocupat}}$ ∩ cm, $\overline{\text{ocupat}}$ ∩ sm)/(4,5,8)

In cazul cand memoria principala nu a terminat operatia de scriere (ocupat = 1) se asteapta la pasul 4. Alfel, in functie de comanda cm sau sm, se trece la pasii 5 sau 8.

5. AM ← DSCRPT

6. cram = 1;

→(ocupat)/(6)

7. DMI * $\overline{\text{ocupat}}$ ← DATAIES;

→(ocupat)/(7)

8. (TA,MI) * PRILIB(CU) ← DSCRPT,DMI;

CU * PRILIB(CU) ← $\overline{\overline{8|0}}$;

gata = 1;

9. →($\overline{\text{cm}\cup\text{sm}}$),(cm ∪sm))/(1,9)

END SEQ.

ADRESAIN = AM; DATAIN = DM

END

S-a presupus ca, la o operatie de scriere, cand cuvantul nu se gaseste in MI, se va scrie direct in RAM, dupa care, la pasul 8, cuvantul aflat in DMTI si adresa lui din DSCRPT se vor stoca in ansamblul MI,TA, la locatia specificata de PRILIB(CU). Contorul asociat se va forta la valoarea 2^8-1, ceilalti contoare fiind deja decrementati inainte de a comanda citirea din RAM.

Se poate include si cazul transferului in RAM a intregului continut al memoriei intermediare. La terminarea unui program, sistemul de operare poate asigura acest transfer in mod automat.