Mihai ROMANCA

Microprocesoare şi microcontrolere

Universitatea Transilvania din Braşov
2015

MICROPROCESOARE ŞI MICROCONTROLERE

Mihai ROMANCA

2015

ISBN 978-606-19-0683-3

CUPRINS

Cuprins ...1
Cuvânt înainte ...3
1. INTRODUCERE ÎN ARHITECTURA MICROPROCESOARELOR...5
1.1. Definire microprocesor şi microcontroler ...6
1.2. Calculatorul digital ...7
1.3. Definirea noţiunii de arhitectură a microprocesorului ... 11
1.4. Organizarea generală a unui sistem uniprocesor... 13
1.5. Tendinţe ale tehnologiei – Legea lui Moore ... 15
1.6. Evaluarea performanţelor unităţii centrale de procesare ... 17
1.7. Legea lui Amdahl... 29
1.8. Reprezentarea funcţională a unui calculator ... 32
1.9. Reprezentarea structurală a unui calculator... 38
1.10. Organizarea ierarhică a sistemului de memorie ... 42
1.11. Organizarea magistralelor calculatorului ... 48
1.12. Exerciţii ... 54

2. ARHITECTURA SETULUI DE INSTRUCŢIUNI ... 57
2.1. Introducere în arhitectura setului de instrucţiuni... 58
2.2. Formatul instrucţiunilor... 60
2.3. Interdependenţa set de instrucţiuni - organizare internă ... 63
2.4. Scurtă privire comparativă între arhitecturile RISC şi CISC ... 68
2.5. Tipuri de instrucţiuni ... 71
2.6. Moduri de adresare ... 79
2.7. Exerciţii... 90

3. ORGANIZAREA ŞI FUNCŢIONAREA MICROPROCESORULUI DE UZ GENERAL... 93
3.1. Structura de procesor de uz general ... 94
3.2. Structura unui microprocesor elementar... 102
3.3. Alte registre interne semnificative pentru UCP ... 107
3.4. Exemple privind modul de setare al indicatorilor de condiţii... 112
3.5. Semnale la interfaţa UCP cu exteriorul ... 115
3.6. Sistemul de întreruperi... 132
3.6.1. Rolul memoriei stivă... 142
3.6.2. Utilizarea ferestrelor de registre... 146
3.7. Exerciţii... 150

4. NOŢIUNI PRIVIND REPREZENTAREA ŞI PRELUCRAREA DATELOR ... 153
4.1. Reprezentarea binară a numerelor în calculator ... 154
4.2. Coduri zecimal - binare (coduri BCD) ... 166
4.3. Reprezentarea numerelor în virgulă mobilă ... 168
4.4. Coduri alfanumerice ... 176
4.5. Adunare şi scădere pentru numere reprezentate în virgulă fixă... 177
4.6. Suma algebrică a mai multor operanzi reprezentaţi în cod complementar... 181
4.7. Adunare numerelor reprezentate în virgulă mobilă ... 183
4.8. Unitatea de prelucrare a datelor pentru virgulă fixă ... 186

4.9. Înmulţirea numerelor binare reprezentate în virgulă fixă 190
4.10. Circuite de înmulţire construite cu reţele combinaţionale 202
4.11. Împărţirea numerelor binare 204
4.12. Exerciţii 208

5. UNITATEA DE CONTROL A UCP 209
5.1. Funcţiile principale ale unităţii de control a UCP 210
5.2. Control cablat 213
5.3. Controlul microprogramat 216
5.3.1. Metode de optimizare a UC microprogramate 220
5.4. Paralelism în execuţia instrucţiunilor prin tehnici pipeline 225
5.5. Exerciţii 234

6. SISTEMUL DE INTRARE - IEŞIRE 235
6.1. Circuite de interfaţă 236
6.2. Modalităţi de transfer de intrare-ieşire 240
6.2.1. Transferul de I/O prin program 240
6.2.2. Transferul I/O prin întreruperi 243
6.2.3. Transferul prin acces direct la memorie 248
6.3. Exerciţii 250

7. SISTEMUL DE MEMORIE 251
7.1. Caracteristici principale ale dispozitivelor de memorie 252
7.2. Structura si funcţionarea memoriilor statice cu acces aleator (SRAM) 254
7.3. Structura şi funcţionarea memoriilor dinamice cu acces aleator (DRAM) 261
7.4. Memoria cache 276
7.5. Tehnici de adresare şi alocare a memoriei 281
7.6. Tehnici de translatare a adreselor 284
7.7. Exerciţii 297

8. ARHITECTURA MICROCONTROLERELOR 299
8.1. Introducere 300
8.2. Caracteristici principale ale microcontrolerelor 302
8.3. Exerciţii 318
Prescurtări utilizate 319
Bibliografie 320

Cuvânt înainte

Cartea intitulată „Microprocesoare şi Microcontrolere” cuprinde, în general informaţia furnizată la curs studenţilor de la specializarea Tehnologii şi Sisteme de Telecomunicaţii. Sper că va fi, de asemenea, extrem de utilă şi altor categorii de studenţi şi specialişti în domeniu. Materialul de faţă nu face descrierea unui microprocesor particular, ci încearcă să analizeze şi clasifice aspectele generale, relativ stabile în lumea microprocesoarelor. Motivul îl constituie continua dezvoltare a arhitecturilor şi a organizărilor specifice atât în domeniul calculului mobil, calculului paralel cât şi în domeniul specific al sistemelor embedded (cu calculator încorporat aplicaţiilor particulare). Consider că prezentarea unei arhitecturi particulare ar restrânge mult cunoştinţele transmise studenţilor, iar în plus modul de abordare sper să fie util pentru ca viitorul inginer să îşi poată asuma responsabilitatea punerii în funcţiune a diferitelor sisteme de control cu microprocesor / microcontroler, a întreţinerii sau a proiectării şi construcţiei diferitelor interfeţe (acestea fiind realizabile şi prin studiul suplimentar al documentaţiilor de firmă). Conţinutul volumului de faţă este destinată sistemelor uni-procesor, ca o cale către înţelegerea şi utilizarea sistemelor de procesare paralelă.

Cartea are ca principal obiectiv introducerea cititorului în arhitectura microprocesoarelor şi microcontrollerelor cu explicarea noţiunilor privind setul de instrucţiuni, aritmetica în procesor şi structura căii de date, calea de control, memorie de date şi memorie de program, arhitectura sistemului ierarhizat de memorie, structurile de tip pipeline, semnalele de interfaţare ale microprocesorului, sistemul de întreruperi. La multe dintre sub-sistemele descrise se face o clasificare şi exemplificare a arhitecturilor de microprocesoare/microcontrolere moderne existente pe piaţă.

Primele 7 capitole se referă la microprocesoarele de uz general şi la unităţile funcţionale principale cu care lucrează microprocesorul în cadrul unui calculator (memorie şi sistem de intrare/ieşire), iar ultimul capitol tratează aspecte particulare şi caracteristici reprezentative pentru microcontrolere. Am încercat organizarea materialului astfel încât cunoştinţele să poată fi câştigate gradual pornind de la aspectele elementare ale organizării până la extensiile actuale ale arhitecturii microprocesorului, în ceea ce priveşte paralelismul prelucrărilor.

Materialul cărţii este dedicat în special studenţilor din învăţământul ingineresc de profil electric/electronic/telecomunicaţii, de la care aştept observaţii şi comentarii privind modul de prezentare şi eventualele lipsuri ale manualului.

Prof. dr. ing. Mihai Romanca

romanca@unitbv.ro

Capitolul 1. Introducere în arhitectura microprocesoarelor

Conţinut

1.1. Definire microprocesor şi microcontroler
1.2. Calculatorul digital
1.3. Definirea noţiunii de arhitectură a microprocesorului
1.4. Organizarea generală a unui sistem uniprocesor
1.5. Tendinţe ale tehnologiei – Legea lui Moore
1.6. Evaluarea performanţelor unităţii centrale de procesare
    1.6.1. Timpul, ca măsură a performanţelor calculatorului numeric
    1.6.2 Alte unităţi de măsură
    1.6.3. Programe etalon de testare a performanţelor
1.7. Legea lui Amdahl
1.8. Reprezentarea funcţională a unui calculator
1.9. Reprezentarea structurală a unui calculator
1.10. Organizarea ierarhică a sistemului de memorie
1.11. Organizarea magistralelor calculatorului
    1.11.1. Organizarea ierarhică a magistralelor calculatorului
    1.11.2. Caracteristici principale ale magistralelor
1.12. Exerciţii

1.1. Definire microprocesor şi microcontroler

Microprocesorul este un circuit integrat ce conţine toate unităţile funcţionale specifice unei structuri de procesor de uz general. În prezent, microprocesorul constituie Unitatea Centrală de Procesare (UCP) a unui calculator numeric. UCP conţine o structură de procesare (numită „procesor”) alcătuită din:

- Unitate de control (decodifică instrucţiunile binare ale unui program, generează semnalele de control pentru celelalte unităţi funcţionale ale procesorului, generează semnalele de control pentru componentele externe ale procesorului: memorie externă, sistem de intrare ieşire, magistrale externe şi primeşte semnale de feedback de la unităţile interne şi externe microprocesorului).
- Unitate aritmetică şi logică: unitate de prelucrare a datelor, funcţia specifică executată fiind stabilită în fiecare moment prin semnalele de control primite de la UC.
- Registre de uz general folosite ca memorie locală de mare viteză, magistrale interne pentru transferul datelor, adreselor şi instrucţiunilor, interfeţe cu magistralele externe.

Microprocesorul este un dispozitiv programabil, acelaşi suport hardware oferind mai multe funcţiuni ce depind de instrucţiunile programului ce se execută. Microprocesorul nu poate lucra de unul singur. Pentru a construi un calculator numeric se ataşează memorie (de program şi date) şi un sistem de intrare / ieşire (denumit sistem I/O în continuare) care face legătura calculatorului cu lumea externă şi la care se conectează periferice. Putem spune că un calculator este o maşină de prelucrare automată a datelor ce funcţionează sub controlul unei liste de instrucţiuni (program) stocate în memoria principală a calculatorului.

Istoric, primelor calculatoare ce foloseau un microprocesor în rol de UCP li s-a spus *calculatoare pe bază de microprocesor*, sau *microcalculatoare*. Toate calculatoarele moderne sunt construite cu microprocesoare cu funcţia de UCP, unele fiind bazate pe un sigur microprocesor (calculatoare uni-procesor), altele conţinând mai multe microprocesoare în scopul creşterii puterii de calcul şi a vitezei de prelucrare (calculatoare multi-procesor).

Microprocesoarele, ca şi UCP de uz general, sunt utilizate pentru aplicaţii de prelucrare de înaltă performanţă, fiind folosite pentru construcţia calculatoarelor de uz general. Microprocesoarele sunt utilizate în PC-uri (Personal Computers), staţii de lucru, servere, laptop-uri, unde compatibilitatea software, performanţa, generalitatea şi flexibilitatea sunt importante.

Imediat după apariţia primului microprocesor (1971) firma Intel a scos pe piaţă o familie de microcontrolere numită MCS-48 (începând cu 1976) cu circuitele integrate 8048, 8035 şi 8748.

Aceste circuite nu erau destinate operaţiilor de prelucrare de uz general, ci celor specifice pentru controlul unor aplicaţii din lumea externă circuitului. La momentul apariţiei, circuitele au fost numite "*microcalculatoare pe un chip*", pentru că ele includeau în aceeaşi capsulă de circuit integrat toate componentele unui calculator: UCP, memorie şi sistem I/O.

În momentul de faţă circuitele similare sunt numite *microcontrolere*, pentru că, aşa cum sugerează şi numele sunt destinate aplicaţiilor de control (nu calculului de uz general). Microcontrolerele sunt proiectate pentru a se obţine o dimensiune redusă a chip-ului, pentru micşorarea costurilor şi includerea de spaţiu de memorie şi interfeţe IO pe chip. Microcontrolerele sunt adesea “specializate pe aplicaţii” în dauna flexibilităţii. Indiferent de ceea ce este controlat în lumea externă (de exemplu controlul turaţiei unor motoare electrice, reglarea temperaturii într-o incintă, comanda sistemului antiblocare a frânelor la automobil-ABS), sistemul de comandă digitală este alcătuit din: UCP (nucleu microprocesor), memorie de date şi program, interfeţe şi controlere de periferie. Ideea de bază la microcontrolere este integrarea într-un singur circuit a tuturor acestor componente, prin simplificarea şi micşorarea dimensiunilor diverselor unităţi funcţionale (de exemplu capacitate de stocare în memorie mult mai mică faţă de calculator, interfeţe seriale cu lumea externă etc.

De observat diferenţele semnificative dintre microprocesor şi microcontroler din punctul de vedere al structurii interne, al costurilor, vitezei de prelucrare şi flexibilităţii la rularea diverselor aplicaţii.

1.2. Calculatorul digital

Un *calculator digital* (sau numeric) este constituit dintr-un ansamblu de resurse fizice (*hardware*) şi de programe de sistem (*software de sistem*) care asigură prelucrarea automată a informaţiilor, în conformitate cu algoritmii specificaţi de utilizator prin programele de aplicaţii (*software utilizator*).

Hardware este termenul general care desemnează resursele fizice (circuitele, dispozitivele şi echipamentele componente) ale unui calculator numeric. Deoarece atât domeniul hardware cât şi cel software evoluează rapid, în foarte puţine cazuri arhitecţii unui procesor au posibilitatea să înceapă "de la zero" proiectarea unui procesor. De cele mai multe ori, proiectanţii primesc ca "date de intrare" arhitectura şi chiar setul de instrucţiuni cu care noul procesor trebuie să fie compatibil. Menţinerea compatibilităţii, deşi îngreunează proiectarea, este obligatorie datorită faptului că nimeni nu este dispus să renunţe, de exemplu din trei în trei ani, la toate datele acumulate şi toată investiţia făcută în software. Pe acest nivel se defineşte noţiunea de *compatibilitate binară*, ca fiind posibilitatea unei maşini de a rula un program scris pentru un procesor dintr-o generaţie mai veche. Menţinerea compatibilităţii binare implică, pe lângă

moştenirea setului de instrucţiuni, şi păstrarea modelului secvenţial de execuţie a instrucţiunilor. Aceste două constrângeri se dovedesc a fi foarte dure în cazul procesoarelor moderne, superscalare, care încearcă să elimine o mare parte din secvenţialitatea programului şi să îl transforme într-un program paralel, de înaltă performanţă, dar care să păstreze *aparenţa* de program secvenţial.

În ceea ce priveşte partea de program rulată pe "suportul" hardware există mai multe tipuri de programe rulate de un calculator numeric. Prin "*software*" vom înţelege totalitatea programelor de sistem şi aplicaţii. Seturile de programe dintr-un sistem de calcul pot fi denumite după utilizarea lor.

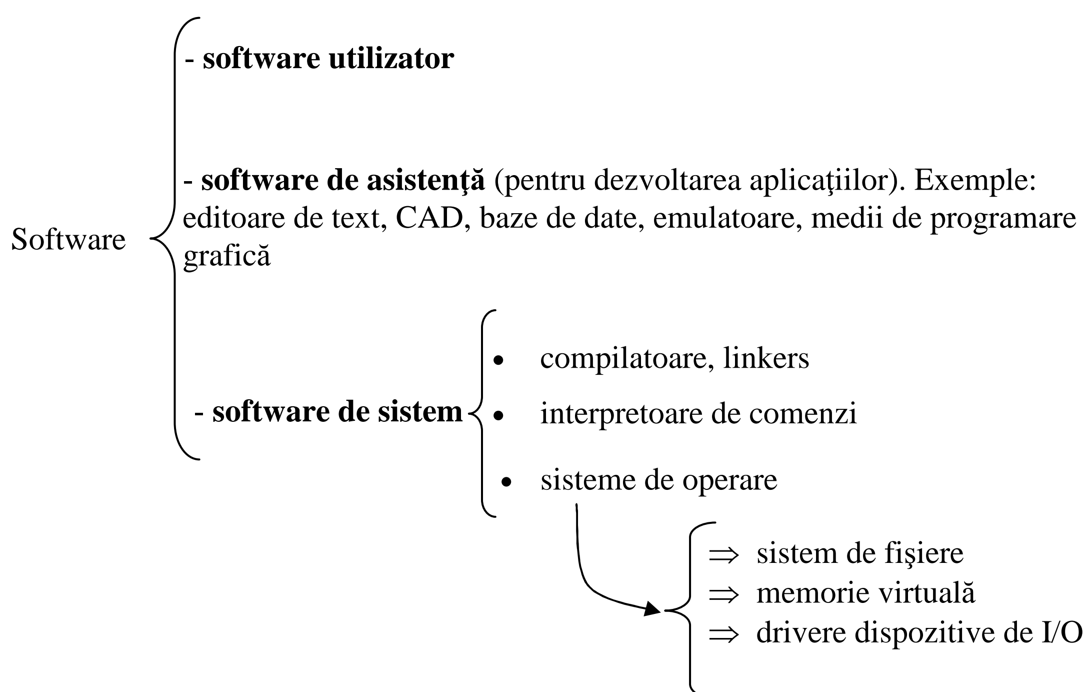


Figura 1.1. Clasificare generală a componentelor software ale unui calculator.

Astfel programele ce furnizează servicii utile tuturor proceselor şi resurselor sistemului de calcul, în mod general, sunt numite programe de sistem (*software de sistem*). Exemple de software de sistem sunt: sistemele de operare, compilatoare, asambloare etc.

Există însă şi software - destinat programatorilor - software de asistenţă pentru dezvoltarea aplicaţiilor (numit uneori şi software de aplicaţii). Exemple de software de asistenţă pentru dezvoltarea aplicaţiilor: editoare de text, programe de proiectare asistată, programe pentru baze de date etc.

O clasificare foarte generală a sub-sistemului software al calculatorului numeric, sub forma de niveluri ierarhice, este prezentată în figura 1.1.

Ultimul tip de software în clasificarea prezentată în figura 1.1. este reprezentat de programele utilizator ce rezolvă anumite probleme specifice (software utilizator sau aplicativ). De observat că la sistemul de operare s-au trecut doar funcţiile principale, "vizibile" utilizatorului:

- administrarea sistemului de fişiere (administrarea memoriei auxiliare - MA)
- gestiunea memoriei principale (MP) şi asigurarea tehnicilor software pentru implementarea memoriei virtuale,
- gestionarea programelor de control (drivere) pentru dispozitivele de intrare / ieşire (I/O).

În cadrul programelor de sistem, în afara sistemului de operare s-au specificat programele de sistem ce oferă servicii generale programatorilor (compilatoare, asambloare, editoare de legături, programe de depanare şi corectare a erorilor, bibliotecari, etc.)

Este important de observat ca aceste programe nu fac parte din sistemul de operare, chiar daca sunt furnizate în mod obişnuit de producătorii de calculatoare, împreună cu sistemul de operare. Sistemul de operare este, în general, acea porţiune din software ce rulează în "kernel mode" (mod nucleu) sau în mod supervizor. El este astfel protejat (ignorăm aici procesoarele ce nu oferă suport hardware suficient pentru rularea unui sistem de operare în mod protejat, cum ar fi procesoarele Intel din seria 80x86 lucrând în "mod real") împotriva stricăciunilor ce ar putea fi provocate de programele de aplicaţii sau de programele utilizator.

Tipurile de software incluse în categoria "software de asistenţă" se referă la resursele logice numite adesea şi programe utilitare pentru dezvoltarea de aplicaţii.

Principalele *tipuri de calculatoare* prezente pe piaţă în momentul de faţă şi caracteristicile lor principale, pot fi descrise pe scurt, conform [Hennesy 2012]:

1. Dispozitive de calcul personale şi mobile.
   Aceste dispozitive, foarte diverse în funcţie de aplicaţiile specifice pentru care au fost construite, sunt portabile (alimentate la baterii), conţin interfeţe multimedia şi de comunicaţie wireless. Includem în această categorie telefoanele mobile şi tabletele. Aceste dispozitive sunt controlate prin intermediul unui sistem de operare. La dispozitivele de calcul personal principalele caracteristici de proiectare se referă la costuri reduse, gabarit mic şi eficienţă energetică mare. Alte caracteristici cheie sunt receptivitatea şi predictibilitatea aplicaţiilor multimedia rulate.
2. Calculatoare desktop
   Includem în această categorie PC (Personal Computer), MacBook, notebook (laptop), staţii de lucru (workstations). Din punctul de vedere al sumelor încasate, domeniul desktop încă are ponderea cea mai mare pe piaţa calculatoarelor. Aceste calculatoare au memorii RAM şi memorii externe de foarte mare capacitate şi putere de calcul adecvată pentru un mare număr de aplicaţii. Principalele caracteristici de proiectare se referă la optimizarea raportului preţ / performanţă. Performanţele se referă în primul rând la viteza de calcul

(exprimată uneori ca latenţă) şi performanţă grafică. Costul calculatoarelor desktop este puternic dependent nu doar de hardware dar şi de sistemul de operare utilizat, respectiv de licenţele plătite pentru aplicaţiile instalate pe maşină.

3. Servere

   Sunt calculatoare superperformante de tip servere de reţea, sau servere Web**,** pentru furnizarea de servicii de calcul şi de transfer al fişierelor. Rolul serverelor a crescut în ultimele decenii pentru a oferi pe scară largă servicii de calcul și de fișiere mai fiabile. Astfel de servere au devenit coloana vertebrală al reţelelor pe scară largă, înlocuind tradiţionalele mainframes.

   Principalele caracteristici de proiectare impuse sunt puterea generală de calcul (throughput[1]), disponibilitatea (availability[2]) şi scalabilitatea (scalability[3]).

   Disponibilitatea serverelor este extrem de importantă, pentru că dacă nu este îndeplinită această caracteristică, pot rezulta pierderi financiare mari nu doar pentru cei ce furnizează serviciile ci şi pentru beneficiarii serviciilor. Luaţi în considerare de exemplu serverele care controlează bacomatele (ATM) pentru bănci sau sistemele de rezervare ale unei companii aeriene [Hennesy 2012]. Scalabilitatea se referă posibilitatea de creştere a resurselor hardware şi software ale serverului, fără a opri serviciile furnizate de acesta.

4. Clusters/ Warehouse-Scale Computers (WSC) [Hennesy 2012]

   WSC se referă la grupuri mari de calculatoare (ca într-un "depozit" de calcul) organizate ca noduri de calcul conectate local printr-o reţea şi care pot oferi servicii de calcul globale, ca şi cum s-ar lucra cu o sigură maşină de calcul extrem de puternică. În reţeaua WSC fiecare nod rulează propriul său sistem de operare, existând comunicare între noduri pe baza unui protocol de comunicaţie. Resursele hardware şi software ale "depozitului" trebuie să lucreze concertat pentru a oferi servicii prin Internet performante, de aici rezultând necesitatea de a trata centrul de date în sine ca un centru de calcul masiv cu dimensiunea unui depozit de calculatoare (WSC)

   WSC sunt direcţionate spre aplicaţii de tip SaaS (Software as a Service) cum sunt: căutare, reţele sociale, partajare de fişiere video, cumpărături on-line, jocuri multiplayer etc. Clusterele WSC oferă servicii în aplicații interactive, servicii de stocare pe scară largă, în condiţii de fiabilitate și lățime de bandă mare la comunicarea prin Internet.

   Supercalculatoarele sunt înrudite cu WSC, dar ele oferă servicii de înaltă performanţă la calcule în virgulă mobilă și prin rularea unori loturi mari de programe cu caracteristici intensive de comunicare, ele putând rula uneori perioade lungi de timp (săptămâni).

5. Calculatoare încorporate în aplicaţie (embedded systems)

---

[1] Throughput = performanţe generale exprimate ca numărul de tranzacţii pe minut sau pagini web servite pe secundă

[2] Availability = Disponibilitate - sistemul poate furniza serviciul eficient şi sigur, 24 de ore din 24.

[3] Scalability = proprietatea de scalare a capacităţii de calcul, a memoriei, a stocării externe, lărgimea de bandă IO a serverului

Un sistem cu calculator încorporat este un sistem pe bază de microprocesor construit pentru a controla anumite funcţii particulare şi care nu este construit pentru a fi programat de utilizatorul final, aşa cum se întâmplă la calculatoarele desktop. Sunt sisteme ce încorporează un microcalculator în produse cum ar fi: automobile, frigidere, semafoare automate, echipamente industriale etc. Componenta software a calculatorului încorporat este stocată în memorii doar cu citire (firmware).

Principalele caracteristici şi constrângeri de proiectare pentru calculatoarele încorporate sunt: costuri minime, putere consumată, gabarit şi execuţie în timp real (furnizare funcţii cu constrângeri - limite de timp).

Dispozitive de calcul personale şi mobile au fost incluse în altă categorie de calculatoare pentru că ele au unele caracteristici comune atât cu sistemele embedded cât şi cu calculatoarele desktop (programabile).

1.3. Definirea noţiunii de arhitectură a microprocesorului

Termenul "arhitectură" este o abstractizare a proprietăţilor unui microprocesor. Ca urmare a evoluţiei funcţiilor şi structurii UCP, a evoluat şi noţiunea de arhitectură. Aceasta reprezintă mai mult decât "interfaţa" între hardware şi software definită iniţial de cercetătorii de la firma IBM (International Business Machine). Un arhitect de calculatoare proiectează maşini care vor rula programe, iar sarcinile sale de proiectare vor include: proiectarea setului de instrucţiuni, organizarea funcţională, proiectarea logică şi implementarea. Implementarea cuprinde totul începând de la circuitele integrate utilizate, până la aspectele privind puterea consumată şi tipul de răcire.

Arhitectura, în sensul iniţial de definire, ca interfaţă între hardware şi software, îngloba funcţiuni în hardware, prin micro-programare (microcod). Orice maşină ulterioară unei maşini din aceeaşi familie, este obligată - prin arhitectura definită - să recunoască setul de instrucţiuni al celei vechi, chiar dacă ma;ina nouă are şi funcţiuni suplimentare, [Patterson94]. La momentul introducerii sale, noţiunea de arhitectură a calculatorului se referea doar la ceea ce astăzi se înţelege prin *arhitectura setului de instrucţiuni*. Aceasta este o interfaţa între resursele hardware şi resursele cele mai "rudimentare" de software (cod maşina - forma binară a programelor înţeleasă de maşină).

Arhitectura setului de instrucţiuni este ceea trebuie să ştie un programator pentru a scrie programe în limbaj de asamblare, respectiv pentru a concepe şi construi un program de tip compilator, sau rutine destinate sistemului de operare. Setul de instrucţiuni, ca şi componentă arhitecturală, permite atât proiectantului cât şi utilizatorului procesorului, să vorbească despre

funcţii, independent de suportul hardware specific care le realizează. Arhitectura setului de instrucţiuni (notată în continuare cu ASI; în limba engleză ISA = Instruction Set Architecture), ca interfaţă între hardware şi software, permite mai multor implementări, cu costuri şi performanţe diferite, să ruleze acelaşi software.

Noţiunea de arhitectură trebuie privită, prin analogie cu înţelesul clasic al noţiunii de arhitectură, care se referă la ştiinţa şi arta de a proiecta şi a construi clădiri. Astfel arhitectura microprocesorului nu este înţeleasă doar în sens declarativ al ASI (care defineşte un set de proprietăţi abstracte) dar şi în sens procedural, al unei discipline de proiectare, care implică procesul de producere şi implementare a acestor proprietăţi abstracte. Aceasta a doua componenta a noţiunii de arhitectura, care se refera la aspectele de *implementare* a funcţiilor (proiectare şi realizare a circuitelor ce realizează funcţiile), are la rândul său două componente: organizare şi hardware.

Termenul *organizare* include aspectele de nivel înalt ale unui proiect de microprocesor, ca de exemplu organizarea căii de date, organizarea de tip pipeline a căii de control, organizarea magistralelor, organizarea memoriei, sau proiectul intern al UCP. Noţiunea de *hardware* (resurse fizice) e utilizată pentru a ne referi la aspectele specifice ale implementării maşinii. Acestea includ proiectul logic de detaliu şi tehnologia de realizare a microprocesorului.

În concluzie arhitectura microprocesorului cuprinde două componente principale:

1. arhitectura setului de instrucţiuni (ASI)
2. implementarea maşinii, cu cele două sub-componente:
    - organizare
    - hardware

1.4. Organizarea generală a unui sistem uni - procesor

Ca organizare generală, independent de tehnologia de realizare, resursele fizice (hardware) ale unui calculator numeric uni-procesor cuprind (figura 1.2.):

a. *Microprocesorul*. Este procesorul central (de uz general) al unui calculator fiind numit şi Unitate Centrală de Prelucrare (UCP). Este format din cale de date şi unitate de control. Calea de date cuprinde unitatea aritmetică şi logică (ALU - de la Arithmetic and Logic Unit în limba engleză), setul de registre interne, eventuale blocuri de memorie temporară şi magistralele interne procesorului, necesare transferului informaţiei. Componentele căii de date execută, în fiecare moment, operaţii elementare conform comenzilor primite de la Unitatea de Control (UC). Unitatea de control a procesorului este automatul care, în funcţie de informaţiile primite din exterior, comandă celelalte unităţi funcţionale ale procesorului, cu scopul execuţiei instrucţiunilor. La toate calculatoarele moderne unitatea centrală de prelucrare este formată dintr-unul sau mai multe microprocesoare.

b. *Memoria principală* (numită şi memorie internă sau memorie operativă). Este adresabilă, prin adresă lansată de UCP, la nivel de cuvânt (octet sau multiplu de octet) şi este selectată şi prin semnale de comandă de către procesor. În memoria principală, dacă se consideră arhitectura cu memorie unică de date şi instrucţiuni, se stochează instrucţiunile programelor rulate de procesor şi se pot scrie / citi date aferente programelor. Este o memorie de tip semiconductor, putând fi memorie doar cu citire (ROM - Read Only Memory), sau memorie cu citire scriere (RWM = Read Write Memory, numită adesea RAM - Random Access Memory). Din punctul de vedere al modului de funcţionare şi construcţie al celulelor de memorie se pot folosi două tipuri de memorie RAM semiconductoare: RAM static şi RAM dinamic.

c. *Sistemul de intrare / ieşire*. Sistemul de intrare - ieşire este, "poarta" prin care se face legătura calculatorului cu lumea externă, pentru a citi şi furniza date. Echipamentele cuplate la acest sistem sunt numite echipamente periferice, iar conversaţia între calculator şi aceste echipamente se face prin *logică adresabilă*. Fiecare circuit de comunicare cu exteriorul adresabil printr-o *adresă unică*, este numit "*port*". Dintre echipamentele periferice standard pentru un calculator de uz general amintim: tastatura, echipamentul de afişare alfanumerică (display) şi memoria externă (Hard-disc).

După cum se observă din figura 1.2. Unitatea de Control şi Calea de date (registre, ALU, registre de adresare, registre de interfaţă cu magistralele) formează Unitatea Centrală de

Procesare - UCP (procesorul calculatorului numeric). Calculatorul numeric cuprinde cel puţin o unitate UCP şi memoria principală. Un sistem de calcul este format dintr-un calculator numeric şi diferite dispozitive periferice.

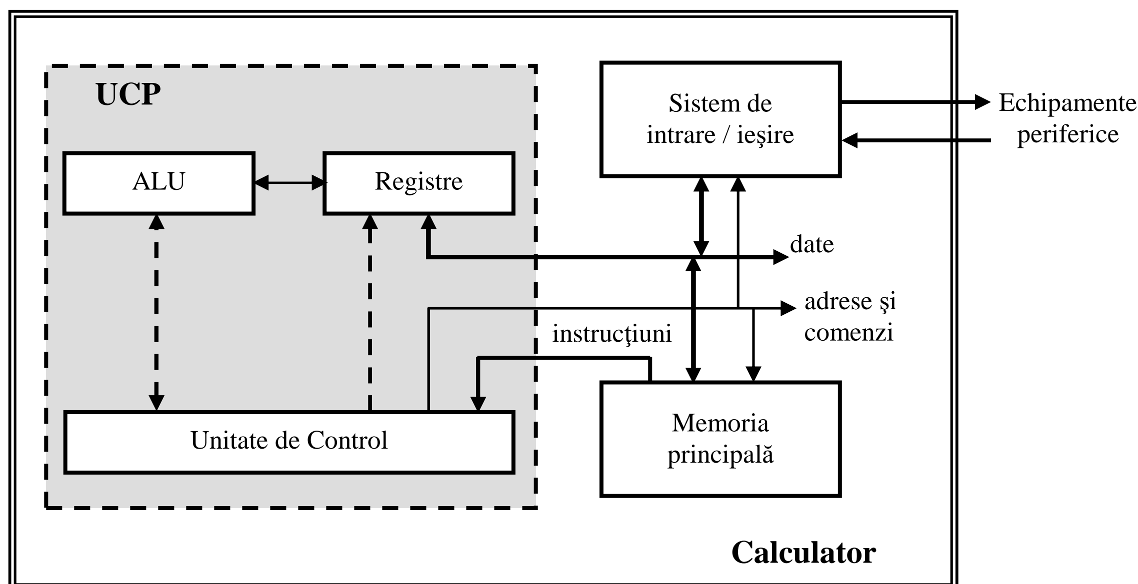


Figura 1.2. Organizarea generală a unui calculator uni-procesor. ALU = Unitate aritmetică şi logică. Liniile punctate reprezintă semnale de comandă şi sincronizare interne procesorului.

Toată această structură hardware a (micro)procesorului (fixă - încastrată în siliciu) poate realiza funcţiuni pentru utilizatorul uman, prin aplicaţii dedicate, doar dacă pe suportul hardware se execută un set coerent de instrucţiuni care indică funcţiile elementare ce trebuie executate de suportul hardware. Funcţiile pot fi foarte diverse şi de aceea se poate spune că instrucţiunile ce alcătuiesc un program realizează o *reconfigurare logică* periodică a căii de date a procesorului, pentru a se realiza funcţiunile cerute.

Pentru a prezenta, la modul foarte general, informaţii elementare ce se vehiculează între procesor şi memoria principală, în figura 1.3. se prezintă schematic interacţiunea dintre principalele componente ale unui procesor şi memoria principală (externă procesorului).

Conform figurilor 1.2. şi 1.3 informaţiile principale pe care microprocesorul, ca UCP de calculator, le schimbă cu exteriorul sunt: date (operanzi şi rezultate) , instrucţiuni, adrese, informaţii de control. Toate acestea se transmit ca semnale electrice prin linii conductoare grupate funcţional în magistrale. În mod tradiţional magistralele externe procesorului sunt clasificate în:

- magistrală de date (Bus de date)
- magistrală de adrese (Bus de adrese)
- magistrală de control (Bus de control)

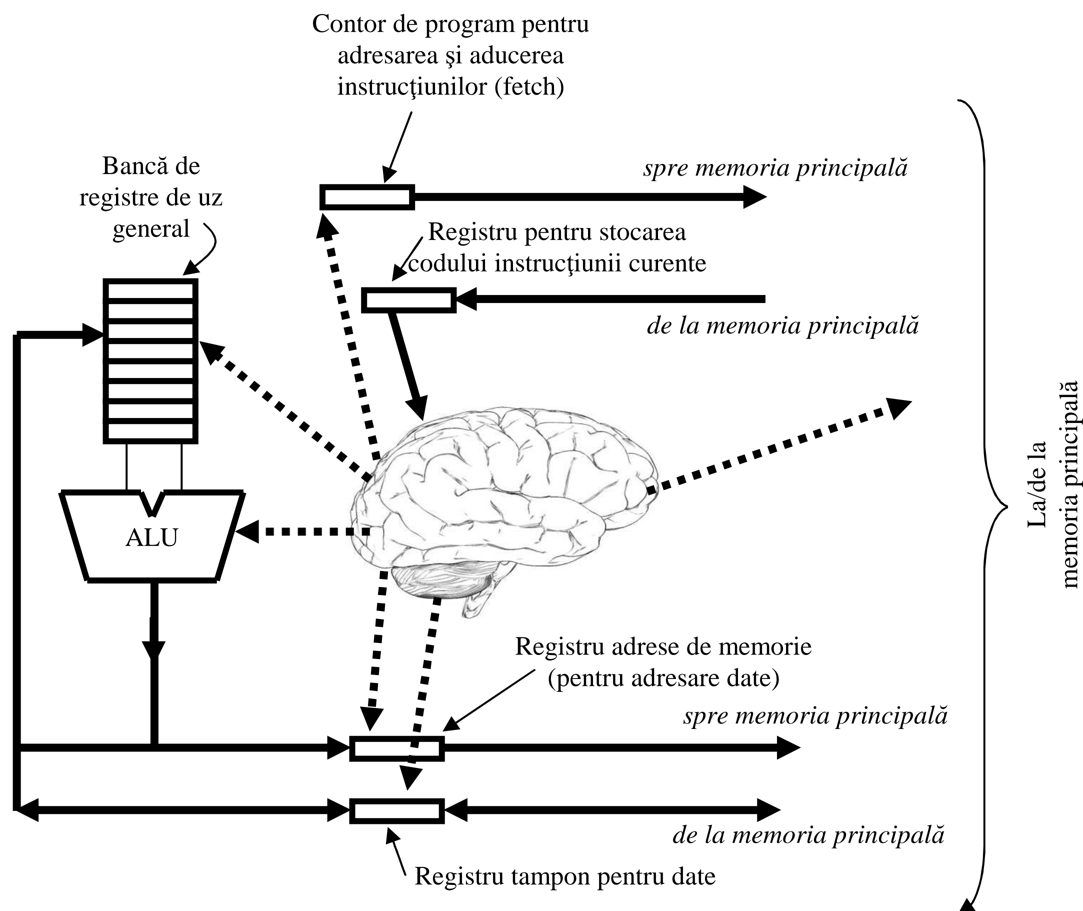


Figura 1.3. Organizarea generală a unui procesor, cu prezentarea interacţiunilor principale dintre unităţile funcţionale (prin linie punctată s-au simbolizat comenzile interne procesorului)

1.5. Tendinţe ale tehnologiei – Legea lui Moore

De-a lungul timpului, performanţele microprocesoarelor nu au crescut în ritm constant. Există două forţe care stimulează dezvoltarea calculatoarelor: *îmbunătăţirea tehnologică* şi *inventarea unor noi arhitecturi - organizări*.

Apariţia microprocesorului a adus posibilitatea de integrare a unui număr mare de tranzistoare ceea ce a permis un ritm de creştere a performanţelor de circa 35% pe an. Ritmul de creştere a fost şi mai mare după 1980, când acesta a fost determinat şi de apariţia unor idei arhitecturale noi ce au condus la crearea procesoarelor RISC.

Între microprocesoarele existente în prezent pe piaţă şi de asemenea între calculatoarele construite cu ajutorul acestor microprocesoare există o mare diversitate, diversitate ce se manifestă din mai multe puncte de vedere:

- al tehnologiei folosite pentru implementarea lor,

- al caracteristicilor funcţionale (proprietăţi şi funcţii observabile din exterior) şi
- al organizării structurii[4] lor interne.

În 1965, Dr. Gordon E. Moore, la acel moment vicepreşedinte executiv la Intel Corporation, a prezis că numărul de tranzistoare echivalente pe un singur chip va creşte în ritm constant şi se prevedea o creştere în acelaşi ritm pentru următoarele decenii. Această predicţie, numită *Legea lui Moore*, spune că "puterea de calcul va creşte exponenţial, dublându-se la fiecare 18 - 24 luni, pentru un viitor previzibil". Legea lui Moore este adesea criticată, pentru că utilizatorii sunt interesaţi în primul rând de creşterea performanţelor microprocesoarelor. Este greu să se echivaleze creşterea numărului de tranzistoare pe un chip cu creşterea performanţei microprocesoarelor. În plus adesea creşterea numărului echivalent de tranzistoare nu a produs o creştere a performanţelor în aceeaşi măsură - de aceea criticii spun că în ultimele decenii îmbunătăţirile aduse circuitelor (procesoare, memorii) au fost dictate mai mult de sistemul de operare Windows al firmei Microsoft, decât de cerinţele de putere de calcul din partea utilizatorilor.

Tehnologia calculatoarelor electronice a evoluat într-un ritm incredibil în ultimii zeci de ani, de la conceperea primului calculator electronic. În prezent un calculator de tip PC (Personal Computer) cu preţ mai mic de 1000 de dolari, are viteză şi capacitate de memorare mult mai mare decât un calculator din anii '80 al cărui preţ era la nivelul sutelor de mii sau milioanelor de dolari. Aşa cum s-a menţionat anterior această dezvoltare rapidă s-a făcut nu numai pe baza dezvoltărilor tehnologice (ce au permis mărirea gradului de integrare şi reducerea puterii consumate şi a preţului) dar şi pe baza inovaţiilor în domeniul proiectării arhitecturale.

Apariţia microprocesoarelor, în anii '70 ai secolului trecut, a permis apariţia unor dezvoltări arhitecturale esenţiale pentru performanţa calculatoarelor, dintre care amintim aici doar câteva: generalizarea utilizării limbajelor de programare de nivel înalt şi a sistemelor de operare, organizări noi - ce permit execuţie paralelă a instrucţiunilor şi generalizarea utilizării memoriilor intermediare ("cache")

---

[4] Structură [MDE72] = Alcătuirea internă a unui corp sau sistem fizic, mod de dispunere a elementelor componente ale unui ansamblu, mod specific de organizare a elementelor constitutive ale unei limbi, mod de organizare a unei mulţimi de elemente, prin definirea uneia sau mai multor operaţii care au anumite proprietăţi.

1.6. Evaluarea performanţelor unităţii centrale de procesare

1.6.1. Timpul, ca măsură a performanţelor calculatorului numeric

Performanţele procesorului unui calculator numeric se evaluează în primul rând prin determinarea timpului de execuţie al programelor. *Timpul* este măsura ce indică performanţa. Comparând două maşini de calcul, concluzia de genul "Maşina A este mai rapidă decât maşina B" se referă la faptul că timpul de execuţie al aceluiaşi program pe maşina A este mai mic decât pe maşina B. Dacă maşina A este *de n ori* mai rapidă decât maşina B putem scrie:

$$\frac{Timp_executie_B}{Timp_executie_A} = n \tag{1.1}$$

sau considerând că performanţa (ca viteză de execuţie) este inversul timpului:

$$\frac{Timp_executie_B}{Timp_executie_A} = \frac{1/Performanta_B}{1/Performanta_A} = \frac{Performanta_A}{Performanta_B} = n \tag{1.2}$$

Comparaţia între cele două maşini poate fi exprimată şi procentual: "Maşina A este cu p% mai rapidă decât maşina B". Aceasta se poate exprima analitic astfel:

$$\frac{Timp_executie_B}{Timp_executie_A} = 1 + \frac{p}{100} \tag{1.3}$$

sau

$$\frac{Performanta_A}{Performanta_B} = 1 + \frac{p}{100} \tag{1.4}$$

adică

$$p = 100 \cdot \left(\frac{Performanta_A - Performanta_B}{Performanta_B} \right) \tag{1.5}$$

O expresie de genul "Maşina A este mai performantă cu 50% faţă de maşina B" indică că numărul de sarcini de lucru terminate în unitatea de timp pe maşina A este de 1.5 ori mai mare decât pe maşina B.

Cele spuse mai sus sunt adevărate pentru un UCP de calculator care execută un singur program de testare, fără întrerupere de la lansare, până la terminarea sarcinii de lucru. În alte cazuri, cum ar fi de exemplu un sistem cu multi-programare, UCP poate abandona temporar un program (întreruperea execuţiei) în timp ce se aşteaptă operaţii de I/O şi să se treacă la execuţia altui program. Ca urmare, în măsurarea timpului de execuţie al unui program pe un anumit procesor trebuie ţinut cont şi de eventuala întrerupere temporară a programului. Evident că din

punctul de vedere al utilizatorului timpul este un *timp de răspuns* (de la lansare până la terminarea execuţiei) sau un *timp de aşteptare*, care include şi intervalele de timp când se rulează altceva, prin partajarea timpului procesorului. De aceea, în [Patterson96] se introduce noţiunea de *timp al UCP* (pe care îl vom nota în continuare T_{UCP}), care este o măsură a timpului efectiv cât procesorul (UCP) rulează programul de testare a performanţelor. Acest timp reprezintă timpul cât UCP realizează prelucrări fără a include aşteptarea operaţiilor de I/O şi rularea altor programe. Timpul T_{UCP} include însă şi intervalele de timp ce corespund timpului UCP consumat de către sistemul de operare pentru realizarea unor sarcini cerute de programul de test.

Evaluarea performanţelor unui calculator se face pentru a putea compara aceste performanţe cu performanţele obţinute de alte calculatoare. Un utilizator al calculatorului poate spune că un calculator este mai rapid decât altul dacă acesta rulează un program într-un timp mai scurt, pe când administratorul unui centru de calcul poate spune că un calculator este mai rapid dacă acesta termină mai multe sarcini de lucru într-un anumit interval de timp. Utilizatorul calculatorului este interesat în primul rând de reducerea *timpului de execuţie* (sau a *timpul de răspuns* între momentele startului şi terminării unei sarcini de lucru), sau altfel spus în micşorarea timpului de aşteptare (exprimat adesea ca latenţă). Managerul centrului de calcul este interesat în creşterea *performantelor globale* ale sistemului de calcul exprimate prin denumirea “*throughput*“ (putere de calcul). Throughput este termenul în limba engleză folosit pentru a descrie performanţele unui procesor prin cantitatea de date pe care le poate prelucra în unitatea de timp, sau performantele globale exprimate prin capacitatea de a transmite / recepţiona cât mai multe date în unitatea de timp cu toate componentele sistemului.

Pentru a exprima performantele calculatoarelor cercetătorii în domeniu au introdus mai întâi unităţi de măsură care ţin seama de perioada de ceas a UCP. Astfel, timpul de execuţie al unui program, specific unei UCP, poate fi exprimat ca [Patterson94]:

$$T_{UCP} = NrI \times CPI \times T_{clk} \quad (1.6)$$

unde:

- NrI = numărul de instrucţiuni din program
- CPI = numărul mediu de perioade de ceas pe instrucţiune; poate fi calculat prin împărţirea numărului de perioade de ceas pentru un program, la numărul de instrucţiuni din program (în limba engleză Clock Cycles Per Instruction).
- T_{clk} = perioada impulsului de ceas a procesorului.

Exemplul 1.1.

Dacă frecvenţa ceasului unui procesor este de 1 MHz, iar rularea unui program necesită 5 milioane de perioade de tact pentru execuţie completă, care este timpul de execuţie al programului?

R:

$$T_{UCP} = NrI \times CPI \times T_{clk}$$
$$= Perioade_tact_pe_program \times T_{clk}$$
$$= \frac{Perioade_tact_pe_program}{f_{clk}}$$

$$T_{UCP} = \frac{5 \times 10^6}{1 \times 10^6} = 5 \text{ secunde}$$

Încercând să comparăm diferite sisteme, apar următoarele problemele:

- frecvenţele de ceas sunt diferite pentru diferite calculatoare
- arhitecturile celor două calculatoare comparate pot fi diferite, conducând la valori diferite pentru componentele din relaţia timpului de execuţie pe UCP.

De aceea, pentru a putea face o comparaţie cât de cât acceptabilă, trebuie rulat acelaşi program de testare a performanţelor pentru ambele calculatoare care se compară.

Pentru a micşora timpul de execuţie ar trebui să micşorăm cele trei componente din relaţia (1.6), componente care nu sunt însă independente.

Numărul mediu de perioade de ceas pe instrucţiune (*CPI*) este determinat de:

- arhitectura procesorului, prin componentele: set de instrucţiuni şi modul de organizare
- viteza de lucru a procesorului cu memoria principală, datorită faptului că instrucţiunile şi datele iniţiale / finale se stochează în această memorie.

Valoarea medie *CPI* poate fi exprimată fie ca raport între numărul total de cicluri de ceas şi numărul de instrucţiuni, fie în funcţie de numărul mediu al perioadelor de ceas, (P), necesare procesorului pentru decodificarea şi execuţia instrucţiunilor, numărul mediu de referinţe-accesări la memorie (A) şi raportul k dintre timpul de acces la memorie şi perioada de ceas a procesorului, conform relaţiei:

$$CPI = P + A \times k \qquad (1.7)$$

Din cele pomenite se conturează câteva direcţii ce pot fi folosite pentru a micşora CPI şi implicit şi T_{UCP}:

- proiectare adecvată a setului de instrucţiuni
- organizare ce permite operaţii paralele în execuţia instrucţiunilor

- proiectare a unităţii de control în scopul micşorării timpului de decodificare şi a numărului de paşi (stări ale automatului) utilizaţi în ciclul de execuţie a instrucţiunilor
- micşorarea timpului de acces la memoria principală prin diverse metode (tehnologia de realizare şi organizarea internă a circuitelor de memorare, introducerea unor niveluri intermediare de memorie de mare viteză, optimizarea organizării spaţiului de adresare)

A doua componentă din relaţia (1.6), numărul de instrucţiuni (NrI), depinde de tipul programului rulat, de arhitectura setului de instrucţiuni a procesorului şi de structura şi calitatea compilatorului folosit la translatarea din program sursă (în limbaj de programare de nivel înalt), în program obiect. Remarcăm că între direcţiile de micşorare a apărut din nou arhitectura setului de instrucţiuni, dar în sens contrar celui de la CPI. Aceasta pentru că un număr mai mic de instrucţiuni pe program înseamnă, în general, instrucţiuni mai complexe, deci care durează un număr mai mare de cicluri de ceas. Dar un număr mai mic de instrucţiuni pe program, presupune şi un număr total mai mic de instrucţiuni recunoscute de procesor (funcţii implementate în setul de instrucţiuni). Aceasta înseamnă un număr mai mic de biţi necesari pentru codificarea instrucţiunilor şi ca urmare o posibilitate de micşorare a lungimii instrucţiunilor, cu efect şi asupra dimensiunilor necesare în memoria locală, intermediară, sau principală.

Frecvenţa impulsului de ceas (clock) a UCP depinde în primul rând de tehnologia de realizare a procesorului, dar şi de organizarea internă a acestuia. Din păcate creşterea frecvenţei de oscilaţie a impulsului de ceas conduce la creşterea puterii consumate şi a căldurii disipate de circuite.

Îmbunătăţirea tehnologică este o condiţie necesară, dar nu şi suficientă pentru creşterea spectaculoasă a performanţelor. *Procesarea paralelă* a informaţiei conduce la îmbunătăţirea performanţelor globale ale maşinii, fără a modifica ciclurile de bază ale procesorului. Condiţia de funcţionare a arhitecturilor care permit execuţia în paralel a mai multor instrucţiuni este ca instrucţiunile să fie independente. Procesarea paralelă a instrucţiunilor într-o arhitectură *pipeline*, poate conduce, în cazul ideal, la terminarea unei instrucţiuni la fiecare perioadă de ceas. În cazul ideal performanţa se va îmbunătăţi de un număr de ori egal cu numărul de nivele ale arhitecturii pipeline.

Îmbunătăţirea performanţelor dincolo de nivelul obţinut prin pipeline (ideal - o instrucţiune pe ciclu de ceas) se poate face în două moduri [Nicula97]:

- creşterea numărului de etaje pipeline şi lansarea *succesivă* a mai multor instrucţiuni pe o perioadă de ceas (*arhitectură superpipeline*);
- lansarea *simultană* a mai multor instrucţiuni pe o perioadă de ceas, spre unităţi de execuţie multiple (*arhitectură superscalară*).

Ambele variante se bazează pe aceeaşi idee: “lansarea mai multor instrucţiuni într-o perioadă de ceas nu garantează terminarea mai multor instrucţiuni pe perioadă de ceas; dar, lansarea unei singure instrucţiuni pe perioadă de ceas *în mod sigur nu* poate determina terminarea mai multor instrucţiuni pe perioadă de ceas”.

Cu privire la termenul CPI, care este o valoare medie, calculată pentru un anumit program, a numărului de perioade de ceas pe instrucţiune, trebuie să observăm următoarele:

- într-un program există mai multe tipuri de instrucţiuni, fiecare tip de instrucţiune putând avea un număr diferit de cicluri de ceas pentru execuţie completă. De exemplu o instrucţiune de transfer între două registre va dura mult mai puţin decât o instrucţiune de înmulţire a două numere.
- arhitectura setului de instrucţiuni, prin modurile de adresare folosite, poate conduce la diferenţe de timp, chiar între instrucţiuni similare. Astfel o instrucţiune de scădere a doi operanzi aflaţi în registrele interne ale UCP va dura mai puţine cicluri de ceas decât o instrucţiune la care unul din operanzi se găseşte în memorie.

Exemplul 1.2.

Într-un program de testare a performanţelor (benchmark) cu 250 instrucţiuni, există următoarele tipuri de instrucţiuni:

- 100 instrucţiuni de tip load/store, fiecare consumând 5 perioade de ceas.
- 80 instrucţiuni de transfer între registre interne ale UCP, fiecare consumând 1 perioadă de ceas.
- 50 instrucţiuni de adunare sau scădere, fiecare consumând 3 perioade de ceas.
- 20 instrucţiuni de înmulţire cu întregi, fiecare consumând 50 perioade de ceas.

Să se calculeze CPI pentru programul de test.

R:

$$CPI = \frac{(100\times5)+(80\times1)+(50\times3)+(20\times50)}{250} = \frac{1730}{250} = 6.92$$

La calculul CPI, ca valoare medie, este adesea util să se calculeze numărul total de perioade de ceas ale CPU ($NrClk_p$), pentru un program, conform relaţiei:

$$NrClk_p = \sum_{i=1}^{n} \left(CPI_i * I_i\right) \tag{1.8}$$

unde:

I_i - indică de câte ori este executată instrucţiunea *i* într-un program

CPI_i - numărul mediu de perioade de ceas necesare pentru execuţia instrucţiunii *i*.

Ca urmare timpul de execuţie de către UCP al unui program, cu n tipuri de instrucţiuni, poate fi exprimat ca:

$$T_{UCP} = T_{clk} * \sum_{i=1}^{n} CPI_i * I_i \tag{1.9}$$

Dacă se introduce noţiunea de "frecvenţă a instrucţiunii" pentru a indica care este procentul reprezentat de instrucţiunea de tip *i* din numărul total de instrucţiuni ale unui program, expresia termenului CPI total, mediu pentru un program poate fi scrisă şi ca:

$$CPI = \frac{NrClk_P}{NrI} = \frac{\sum_{i=1}^{n}(CPI_i * I_i)}{NrI} \tag{1.10}$$

sau

$$CPI = \sum_{i=1}^{n}\left(CPI_i * \frac{I_i}{NrI}\right)$$

Ultima formă a expresiei CPI, indică ca mod de calcul înmulţirea fiecărei valori CPI_i cu frecvenţa de producere a instrucţiunii de tip $_i$ într-un program, expresie care se scrie de obicei ca:

$$CPI = \sum_{i=1}^{n} CPI_i * F_i \tag{1.11}$$

unde $$F_i = \frac{I_i}{NrI} \tag{1.12}$$

Exemplul 1.3.

Se va calcula CPI, pentru un program de test ce rulează pe o maşină RISC (arhitectură load / store), conform datelor din tabelul următor şi al relaţiilor 1.11 şi 1.12. La arhitectura load / store singurele instrucţiuni ce lucrează direct cu memoria principală sunt instrucţiunile de încărcare/citire din memorie (load) sau cele de stocare/scriere în memorie (store).

Tip operaţie	**CPI_i**	**Frecvenţa**	**CPI_i*F_i**	**(% Timp)**
Operaţie ALU	1	60%	0.60	39%
Load (citire memorie)	2	22%	0.44	28%
Store (scriere memorie)	2	3%	0.06	4%
Ramificare	3	15%	0.45	29%
Total:		**100%**		**100%**
CPI mediu total:			**1.55**	

Importanţa practică a acestei ultime forme de exprimare a CPI constă în indicarea locului unde proiectantul trebuie să investească mai mult efort şi resurse pentru a scădea valoarea CPI şi ca urmare să mărească performanţele procesorului. Concluzia poate fi exprimată în două moduri:

- trebuie investit (optimizat) acolo unde se pierde timpul, dacă frecvenţa de apariţie a instrucţiunilor respective este semnificativă în programe.
- trebuie investit în instrucţiunile cu frecvenţă mare şi nu în cele ce se folosesc rar.

1.6.2 Alte unităţi de măsură

În scopul măsurării şi comparării performanţelor s-a introdus mărimea **MIPS** = Milioane de Instrucţiuni Pe Secundă, (Millions of Instructions per Second, în limba engleză) exprimată ca:

$$MIPS = \frac{NrI}{T_{UCP} \times 10^6} \qquad (1.13)$$

sau prin înlocuirea lui T_{UCP} din ecuaţia (1.6) şi simplificare:

$$MIPS = \frac{f_{clk}}{CPI \times 10^6} \qquad (1.14)$$

unde $f_{clk} = 1/T_{clk}$ este frecvenţa oscilatorului de ceas.

Ultima relaţie (1.14) este mai des folosită, fiind mai uşor de calculat, pentru că depinde de frecvenţa de ceas şi nu de numărul de instrucţiuni din program. De asemenea relaţia (1.14) pare să indice corect faptul că un procesor cu frecvenţă de ceas mai mare are o valoare MIPS mai mare. De fapt nu este chiar aşa, pentru că frecvenţa ceasului este inclusă şi în relaţia termenului CPI. Avantajul introducerii parametrului MIPS constă în exprimarea uşoară şi în faptul că este deosebit de sugestiv pentru clienţii cumpărători de maşini de calcul. Măsura prezintă însă dezavantaje atunci când este folosită pentru a compara diferite maşini, pentru că: [Patterson94]:

- valoarea MIPS este dependentă de setul de instrucţiuni, făcând dificilă şi neconcludentă compararea între calculatoare cu seturi diferite de instrucţiuni;
- valoarea MIPS, pentru acelaşi calculator, variază în funcţie de programele rulate. La programele cu operaţii complexe valoarea medie CPI este mare;
- valoarea MIPS poate varia uneori invers proporţional cu performanţa. Exemplul clasic folosit pentru a demonstra aceasta, se referă la operaţiile cu numere reale, reprezentate în virgulă mobilă. Calculatoarele care au (co-)procesor aritmetic şi care pot efectua operaţii în virgulă mobilă, opţional, fie prin hardware (operaţii cablate, prin intermediul procesorului aritmetic), fie prin software, (prin emularea operaţiilor în virgulă mobilă prin unităţi de prelucrare pe întregi), conduc la valori MIPS invers proporţionale cu performanţa. În cazul utilizării variantei cablate, numărul de

instrucţiuni va fi mai mic, dar o instrucţiune în virgulă mobilă necesită mai multe perioade de ceas (CPI mare) decât una în virgulă fixă (întregi). Cu toate că timpul de execuţie al programului ce operează pe numere reale va fi mai scurt decât în cazul folosirii rutinelor de emulare, valoarea lui MIPS va rezulta mică. În varianta de efectuare prin software a operaţiilor în virgulă mobilă, se execută mai multe instrucţiuni simple, rezultă un MIPS mai mare, deşi timpul total de execuţie va fi mai mare.

Exemplul 1.4.

Exemplul doreşte să prezinte o interpretare greşită a performanţelor, utilizând MIPS.

Se compară două maşini, notate cu A şi B. Maşina A recunoaşte o instrucţiune specială de înmulţire a întregilor cu semn, care consumă 50 de perioade de ceas pentru a fi executată. Maşina B nu are instrucţiune specială de înmulţire, dar o realizează prin instrucţiuni simple de adunare, deplasare, comparare, fiecare dintre instrucţiunile simple consumând două perioade de ceas. Dacă ne referim doar la operaţia de înmulţire, iar frecvenţa de ceas pentru ambele maşini este de 200 MHz, rezultă valorile MIPS astfel:

Maşina A = 200/50 MIPS = 4 MIPS
Maşina B = 200/2 = 100 MIPS

Următoarea măsură introdusă a încercat să se refere doar la operaţiile în virgulă mobilă: **MFLOPS** = milioane de operaţii în virgulă mobilă pe secundă (Millions of FLoating-point Operations Per Second). Bazat pe operaţii şi nu pe instrucţiuni, MFLOPS are intenţia să facă o comparaţie corectă între diferite maşini, cu referire nu la instrucţiuni ci doar la operaţii în virgulă mobilă. Ideea de la care s-a pornit a fost că acelaşi program, rulat pe diferite calculatoare, va executa un număr diferit de instrucţiuni, dar acelaşi număr de operaţii în virgulă mobilă. Relaţia folosită pentru calcul este:

$$MFLOPS = \frac{\text{Nr. operatii în VM din program}}{T_{UCP} \times 10^6} \quad (1.15)$$

Din păcate nici această măsură nu este pe deplin edificatoare în compararea performanţelor. Operaţiile în virgulă mobilă implementate pe diferite maşini nu sunt aceleaşi. Unele procesoare pot executa, prin implementare cablată, operaţii în virgulă mobilă de împărţire, radical, sinus etc., iar altele pot recunoaşte doar instrucţiuni aritmetice simple ca adunare şi scădere cu numere reale. Mai mult, operaţiile în virgulă mobilă nu au aceeaşi complexitate; unele

se efectuează mai rapid, altele mai lent. De exemplu, pe aceeaşi maşină, vom obţine o valoare mai mare pentru MFLOPS, dacă programul de test are doar adunări în virgulă mobilă, faţă de cazul când programul de test are operaţii complexe de împărţire, sinus, radical ş.a.m.d.

1.6.3. Programe etalon de testare a performanţelor

Altă direcţie în crearea de măsurători în scopul comparării performanţelor diferitelor maşini e constituită de rularea unor *programe etalon de testare a performanţelor (benchmarks)*. Utilizarea programelor "benchmarks" doreşte să facă evaluarea şi compararea performanţelor maşinilor, prin rularea aceloraşi programe, analizând în final timpul de execuţie.

Programele etalon de testare se pot referi la diferite componente ale sistemului de calcul:

- UCP (Unitatea centrală de procesare)
- aritmetică în virgulă fixă / mobilă
- sistemul de memorie
- sistemul de I/O
- sistemul de operare

Programele de test pot fi împărţite în următoarele categorii [Patterson94]:

1. **Programe reale**: de exemplu compilatoare de C, programe de procesare a textelor, programe de proiectare asistată (Spice). Testarea se face în condiţii similare privind intrările, ieşirile şi setarea opţiunilor. Aplicaţiile reale folosite pentru testarea performanţelor şi comparare au dezavantajul că uneori se pot întâlni probleme de portabilitate, mai ales datorită dependenţei de sistemul de operare folosit pe calculator.
2. **Aplicaţii modificate:** adesea aplicaţiile reale sunt folosite ca şi blocuri componente ale unor programe de testare. Modificările constau în eliminarea unor componente (de exemplu eliminarea componentelor de I/O dacă programul de test este destinat performanţelor UCP) sau prin introducerea unor componente care măresc gradul de portabilitate al programelor. În această categorie sunt incluse şi aplicaţiile controlate prin fişiere de comenzi ("scripted applications"). Script-urile sunt folosite pentru a simula programe de aplicaţie şi de asemenea cu scopul de a reproduce o comportare interactivă, de exemplu prin afişări pe ecranul calculatorului, sau pentru a simula interacţiunea multi-user produsă pe un sistem de tip server.
3. **Nuclee (kernels) din programe reale**: s-au extras porţiuni semnificative, numite nuclee, din programe reale, pentru a fi folosite ca rutine de test. Ele sunt utile pentru a indica performanţa, referită la caracteristici individuale ale unei maşini, permiţând să se explice

motivele diferenţelor în performanţă pentru programele reale. Exemple: Livermore Loops, Linpack.

4. **Toy benchmarks** (programe de test amuzante - jucărie): sunt programe scurte (maxim 100 de linii de cod) care produc un rezultat cunoscut înainte de rulare, folosite doar pentru a testa viteza de execuţie. Exemplu Sieve of Erastosthenes, Puzzle, Quicksort. Nu sunt foarte concludente, acesta fiind un motiv suplimentar pentru care sunt privite ca jucării.
5. **Programe de test sintetice** (synthetic benchmarks): sunt programe artificial create, dar asemănătoare din punctul de vedere al intenţiilor cu programele de test de tip "kernels", încercând să facă testarea pe baza unor operaţii şi operanzi ce se consideră a fi caracteristici pentru frecvenţa medie de apariţie a acestora în programele reale. Dintre testele des folosite, amintim Drystone şi Whetstone. Programul de test Dhrystone se bazează pe analiză statistică şi este utilizat pentru testarea performanţelor la aritmetica cu numere întregi şi pentru modul de transmitere a parametrilor la apelul unor funcţii. A fost implementa în limbajele Ada şi C. Programul de test raportează un număr de Dhrystones/secundă - o valoare utilă doar în compararea cu alte valori obţinute la execuţia programului de test. Numele programului Dhrystone s-a dat comparativ cu numele celuilalt test sintetic numit Whetstone (traducerea din engleză a numelui *Whetstone* înseamnă *tocilă, piatră de ascuţit*). Faptul că Dhrystone nu foloseşte operaţii în virgulă mobilă (în engleză "floating point", iar "float" înseamnă *a pluti*) adică, dacă "nu pluteşte" trebuie să fie *uscat* (în engleză *Dry*). Pentru aritmetica în virgulă mobilă se foloseşte testul Whetstone. Whetstone este o colecţie de coduri care testează performanţa la rularea unor biblioteci matematice în virgulă mobilă. A fost scris iniţial în Algol, dar ulterior a fost implementat în mai multe limbaje de programare de nivel înalt. Programul de test raportează un număr de Whetstones/secundă - o valoare intenţionat fără valoare intrinsecă, utilă doar în compararea cu alte valori obţinute la execuţia programului de test. Sunt utilizate în calcule funcţii sinus, cosinus şi funcţii transcendente, pentru a măsură performanţa coprocesorului sau a simulării acestuia de către UCP prin unitatea de prelucrare a întregilor (ALU).

Teste ca Drystone şi Whetstone, testează amănunţit doar UCP, dar nu testează performanţa celorlalte componente ale sistemului, cum ar fi unităţile de disc, dispozitivul de afişare, etc. De aceea au apărut rutine de test specializate pentru testarea periferiei şi a interfeţelor cu periferia sistemului de calcul.

În 1988 s-a înfiinţat corporaţia SPEC (**The Standard Performance Evaluation Corporation**) ca organizaţie non-profit formată cu scopul de *a stabili, menţine şi garanta* seturi standardizate de programe de test (benchmarks) ce pot fi aplicate la noile generaţii de calculatoare de înaltă performanţă. Organizaţia s-a numit iniţial "System Performance Evaluation Cooperative" şi ea cuprindea firmele HP, DEC, MIPS şi Sun. SPEC [SPEC] a crescut în timp, incluzând în prezent peste 60 de companii, şi a devenit una dintre organizaţiile de succes privind standardizarea programelor

de test în multe domenii ale sistemelor de calcul, al reţelelor de calculatoare, al sistemelor cu calculator integrat (embedded systems”) şi al prelucrării datelor.

Organizaţia SPEC nu realizează testare de maşini de calcul, ea doar dezvoltă seturi de programe de test şi de asemenea analizează şi publică rezultatele propuse/obţinute de membrii organizaţiei şi alte instituţii licenţiate pentru aceasta. Rezultatele ce se găsesc pe web-site-ul organizaţiei au fost transmise de parteneri. SPEC nu rulează nici un program de test, ci doar dezvoltă programe de test, analizează şi publică rezultatele obţinute şi transmise către SPEC de membrii organizaţiei şi alte instituţii licenţiate pentru aceasta. SPEC nu are control asupra companiilor partenere şi nici asupra sistemelor sau configuraţiilor pe care s-au făcut testele, dar SPEC analizează şi validează doar rezultatele ce apar pe web-site-ul lor.

Consorţiul stabileşte *setul de programe de test* (doar programe reale, nuclee, sau aplicaţii modificate) şi intrările folosite la rularea acestor programe. Principala caracteristică a programelor de test SPEC este că sunt formate din *seturi (suite)* de aplicaţii pentru testarea procesoarelor. Avantajul setului de programe de test este dat de faptul că eventuala slăbiciune a uneia dintre aplicaţiile de testare poate fi compensată rulând celelalte aplicaţii din setul etalon de testare (benchmark).

În cazul programelor pentru testarea UCP, există în acest moment cinci generaţii de seturi de programe de test, pentru operare în virgulă fixă (INT - întregi) şi virgulă mobilă (reale, FP - Floating Point): SPEC89, SPEC92, SPEC95, SPEC2000. De exemplu SPEC CPU2000 [SPEC] este un produs software pentru testare realizat de SPEC împreună cu un grup non-profit format din comercianţi de calculatoare, integratori de sisteme, universităţi, organizaţii de cercetare, edituri şi consultanţi din întreaga lume. A fost proiectat pentru a face măsurători de performanţă ce se pot utiliza în compararea sarcinilor intense de calcul pe diferite sisteme. SPEC CPU2000 conţine două serii de programe de test: CINT2000 pentru măsurarea şi compararea performanţelor sarcinilor intense de calcul cu întregi şi CFP2000 pentru măsurarea performanţelor sarcinilor intense de calcul în virgulă mobilă. Cele două componente măsoară performanţa pentru procesorul calculatorului, pentru arhitectura memoriei şi pentru compilator.

Un alt set de programe de test standardizate sunt programele de test TP (“Transaction-Processing”). Acestea măsoară capacităţile unui sistem în realizarea tranzacţiilor, tranzacţii ce constau în accesări şi actualizări de baze de date. De exemplu, sisteme simple TP sunt: un sistem de rezervare de bilete de avion, sau un bancomat (ATM). În ani ’80 s-a creat o organizaţie non-profit numită Transaction Processing Council (TPC) cu scopul de a crea un set de programe de test TP de bună calitate. Primul program de test a apărut în 1985 şi s-a numit TPC-A. Au apărut multe alte variante de programe de test, inclusiv TPC-W pentru testarea performanţelor tranzacţiilor pe bază de Web. Programele de test TPC sunt descrise la adresa www.tpc.org/.

În domeniul sistemelor cu calculator integrat, (Embedded Systems) care au căpătat o extrem de mare răspândire în ultimul deceniu, s-au dezvoltat de asemenea seturi de programe de

test. Acestea au apărut în cadrul organizaţiei EDN Embedded Microprocessor Benchmark Consortium (EEMBC - pronunţat “embassy”). Există cinci tipuri de seturi de programe de test în funcţie de domeniile de aplicaţie:

- auto şi industrial
- bunuri de consum
- interconectare
- automatizări de birou
- telecomunicaţii

În plus faţă de programele etalon de test pomenite mai sus, există o sumedenie de programe de testare realizate prin cooperare universitară, sau realizate de către periodice din domeniul calculatoarelor.

1.7. Legea lui Amdahl

Câştigul în performanţă ce se poate obţine prin îmbunătăţirea unei componente a unui calculator poate fi calculat cu ajutorul Legii lui Amdahl. Legea stabileşte valoarea creşterii globale a vitezei sistemului în condiţiile utilizării unei metode de creştere a vitezei doar pentru una din componentele sistemului, creşterea globală depinzând de fracţiunea de timp cât acea componentă este utilizată. Legea permite calculul valorii numerice a creşterii în viteză a sistemului pe baza îmbunătăţirii unei caracteristici particulare ("speedup" - creştere în viteză, ca raport între timpul consumat înainte de îmbunătăţire şi timpul consumat după îmbunătăţire).

În legătură cu legea lui Amdahl trebuie observat că:

- dacă se creşte viteza de execuţie de n ori doar pentru o fracţiune F din timpul de execuţie al unui program nu înseamnă că am crescut viteza de execuţie a programului de n ori.
- dacă se ştie frecvenţa cu care se utilizează o anumită îmbunătăţire locală dintr-un sistem, se poate calcula cu cât creşte viteza întregului sistem datorită îmbunătăţirii. Îmbunătăţirea locală pentru o UCP se referă de exemplu la: dublarea frecvenţei de ceas, reducerea la jumătate a timpului mediu de acces la memoria principală, dublarea vitezei de lucru la circuitul înmulţitor, etc.
- dacă se doreşte o creştere globală cât mai mare a performanţelor unui sistem trebuie în primul rând crescute performanţele subsistemelor utilizate în majoritatea timpului de execuţie ("*alege cazul comun!*").

Pentru a exprima analitic legea lui Amdahl vom nota cu T_{nou} timpul de execuţie consumat după îmbunătăţirea unei componente şi cu T_{vechi} timpul de execuţie înainte de îmbunătăţire, raportul lor reprezentând creşterea în viteză a sistemului.

$$Speedup = \frac{T_{vechi}}{T_{nou}} = \frac{Performanta_{noua}}{Performanta_{veche}} \qquad (1.20)$$

adică

$$Speedup = \frac{T_{vechi}}{T_{nou}} = \frac{n}{n - nF + F} \qquad (1.21)$$

sau expresia finală:

$$Speedup = \frac{1}{(1-F) + \frac{F}{n}} \qquad (1.22)$$

unde

T_{nou} = timpul de execuţie consumat după îmbunătăţirea de n ori a unei componente

T_{vechi} = timpul de execuţie înainte de îmbunătăţire

F = fracţiunea de timp cât este utilizată componenta îmbunătăţită

n = de câte ori au fost crescute performanţele unei componente.

Relaţia legii lui Amdahl, conform ecuaţiei (1.22) poate fi dedusă pe baza unui exemplu, pe care-l prezentăm în continuare. (vezi şi figura 1.4). Se presupune că un sistem execută un program de test în 100 de secunde. Dacă se creşte de 10 ori viteza unei componente a sistemului, iar această componentă este utilizată doar 20% din timpul total de execuţie al programului, să se determine cu cât creşte viteza întregului sistem.

Pentru timpul de rulare înainte de îmbunătăţirea unei componente putem scrie

$$T_{vechi} = t_1 + t_2 \tag{1.23}$$

în care $t_1 = F \times T_{vechi}$, $t_2 = (1-F) \times T_{vechi}$, iar F = fracţiunea de timp cât este utilizată componenta ce va fi îmbunătăţită. Pentru exemplul numeric din figura 1.2., F = 20%.

După realizarea îmbunătăţirii, prin creşterea vitezei componentei de n = 10 ori putem scrie:

$$\begin{aligned} T_{nou} &= t_{11} + t_2 \\ &= \frac{t_1}{n} + t_2 \end{aligned} \tag{1.24}$$

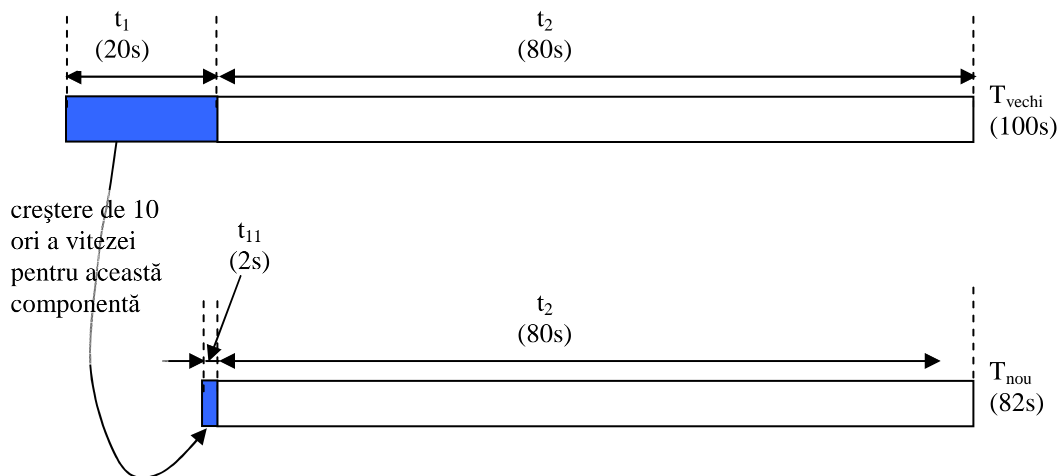


Figura 1.4. Exemplificare, pentru justificarea expresiei analitice a Legii lui Amdhal.

sau

$$T_{nou} = \frac{F \times T_{vechi}}{n} + (1-F) \times T_{vechi} \tag{1.25}$$

şi aranjată pentru comparaţie cu ecuaţia (1.22):

$$\frac{T_{nou}}{T_{vechi}} = \frac{F}{n} + (1 - F) \tag{1.26}$$

Exemplul 1.9.

Un program de test rulează timp de 200 milisecunde pe un sistem uni-procesor. Doar 20% din program poate fi paralelizat pentru a rula pe un sistem multiprocesor. Presupunând că se utilizează un multiprocesor cu 10 procesoare să se calculeze care este creşterea în viteză şi timpul de execuţie al programului de test.

R: F = 20% = 0.2

n = 10

$$Speedup = \frac{T_{vechi}}{T_{nou}} = \frac{1}{(1-0,2) + \frac{0,2}{10}} = \frac{1}{0.8 + 0,02} = \frac{1}{0,82} = 1,22$$

$$T_{nou} = {T_{vechi}}/{1,22} = 163,9 \text{ milisecunde}$$

Exemplul 1.10.

La un procesor unitatea aritmetică în virgulă mobilă este îmbunătăţită, astfel că rulează de 8 ori mai repede. Calculaţi creşterea în viteză a procesorului, pentru un program la care doar 20% din instrucţiuni folosesc operanzi în virgulă mobilă.

R:

F = 20%

n = 8

$$Speedup = \frac{1}{(1-0.2) + \frac{0.2}{8}} = 1.212$$

1.8. Reprezentarea funcţională a unui calculator

Microprocesorul, ca principală resursă hardware a unui calculator recunoaşte şi execută doar un set limitat de *instrucţiuni elementare*, codificate în formă binară. Aceste instrucţiuni binare sunt recunoscute şi decodificate de microprocesor, pentru a se putea genera secvenţa de comenzi necesară execuţiei fiecărei instrucţiuni. Indiferent de tipul de maşina fizică, instrucţiunile recunoscute sunt rareori mai complicate decât [Tanenbaum99] operaţii de tip: adună două numere; verifică dacă un rezultat binar este egal cu zero; copiază date dintr-o zonă a memoriei calculatorului în altă zonă.

De exemplu, la procesoarele Intel din seria 80x86 codul binar al instrucţiunii următoare:

0000 0100 0000 0110

comandă adunarea conţinutului unui registru intern de 8 biţi (numit registrul al) cu valoarea imediată 6. Pentru a uşura munca unui programator la acest nivel, adesea, se foloseşte reprezentarea în hexazecimal a instrucţiunii, care este mai compactă şi mai uşor de citit. Codul, în hexazecimal, al instrucţiunii corespunzătoare succesiunii binare de mai sus este 04 06 hex. După cum cititorul poate să observe şi acest mod de scriere a instrucţiunilor este extrem de complicat pentru programator.

Instrucţiunile elementare, recunoscute de un procesor formează limbajul maşină al procesorului. Vom nota acest limbaj cu *L0*, el corespunzând maşinii fizice *M0*. Limbajul maşină cuprinde însă doar instrucţiuni codate sub formă de cuvinte binare şi de aceea este dificil de utilizat de către om, după cum se observă din exemplul de mai sus.

Presupunem că programatorul poate scrie programele sale într-un limbaj *L1*, format din instrucţiuni simbolice (succesiuni de litere şi cifre) mai apropiat de limbajul uman. Pentru a putea executa acest program pe maşina M0, fiecare instrucţiune a programului scris în L1 trebuie *translatată* în instrucţiuni din limbajul L0. De exemplu programatorul poate scrie în L1 instrucţiunea:

add al,6

iar programul de translatare va genera succesiunea binară, corespunzătoare lui L0:

0000 0100 0000 0110

Se poate spune că limbajul L1 corespunde unei maşini virtuale programabile numită M1. Numele acestui limbaj simbolic de programare este “limbaj de asamblare”, iar programul de translatare este numit “asamblor”.

Translatarea presupune că tot programul scris în L1 este mai întâi transformat în program scris în L0. Apoi programul în L0 este încărcat în memoria calculatorului şi executat, iar programul în L1 este abandonat. Calculatorul este controlat de programul în L0, lansat în execuţie. Exemple de programe de translatare: compilator, asamblor. Conform presupunerii anterioare, compilatorul este un translator din program scris în limbaj superior lui L1 (L2 sau L3). Programele scrise în limbajele L1, L2, L3 etc. sunt numite *programe sursă*, iar programul generat pentru maşina fizică este numit *program executabil*.

Există însă şi varianta rulării programelor pe M0 prin *interpretare*, de către un program numit "interpretor". Interpretorul este un program în L0 care rulează pe maşina M0 şi care preia instrucţiunile programului scris în L1 ca pe date de intrare. Interpretorul citeşte şi decodifică fiecare instrucţiune din L1 şi apoi trece imediat la execuţia acesteia. În cazul interpretorului nu se generează un program executabil ca la translatare. În acest caz calculatorul este controlat de programul interpretor.

Programatorii doresc adesea să dispună de un limbaj de programare mult mai apropiat de limbajul uman, în care să existe comenzi de genul: "tipăreşte un şir la imprimantă", "afişează un text pe ecranul calculatorului", "extrage radical dintr-un număr" etc. Diferenţele dintre un limbaj de acest fel şi L0 sunt mari, iar la transformarea directă către L0 a instrucţiunilor noului limbaj fiecărei instrucţiuni îi va corespunde o succesiune de instrucţiuni elementare din L0. Dacă noul limbaj ar fi numit L1, iar transformarea s-ar face direct în L0 programul de translatare ar fi extrem de complex. De aceea programul scris în noul limbaj, să-l numim limbajul L2, corespunzător maşinii virtuale M2, va fi translatat mai întâi către un program în L1 şi apoi către programul în L0. Conform ierarhiei de limbaje descrise mai sus, un calculator poate fi privit, *din punct de vedere funcţional*, ca o succesiune de maşini virtuale M_i, fiecare corespunzătoare unui limbaj L_i (i = 1, n), ca în figura 1.5. Fiecare nivel are un set de funcţiuni specifice care prelucrează un set de intrări specifice şi generează un set de ieşiri specifice. O maşină virtuală de pe un anumit nivel poate utiliza toate funcţiile oferite de maşina de pe nivelul inferior.

Structura ierarhică a maşinii de calcul indică că între fiecare două niveluri vecine există o interfaţă, deci există şi o ierarhie verticală de interfeţe. Pentru un calculator de uz general, cu procesor având unitatea de control microprogramată, această ierarhie de maşini virtuale şi ierarhie de interfeţe se indică în figura 1.6. În partea dreaptă a figurii s-a exemplificat aspectul programului la nivelul respectiv.

În figura 1.5. nivelurile succesive de maşini virtuale au fost imbricate. Aceasta pentru că o maşina virtuală[5] de pe un nivel superior foloseşte toate funcţiile oferite de nivelurile inferioare. Un nivel este constituit din mulţimea aplicaţiilor asupra elementelor mulţimii de intrare pentru nivelul dat, cât şi asupra elementelor mulţimilor de intrare şi ieşire de la nivelul imediat inferior.

[5] **virtuală,** pentru că utilizatorul unui anumit nivel lucrează cu instrucţiuni specifice, vede doar nivelul la care lucrează, fără să conteze pentru el ce se întâmplă mai jos

Aplicaţiile de la un nivel dat pot constitui aplicaţii şi pentru nivelul superior următor. Cele mai multe calculatoare moderne au 2 sau mai multe niveluri. Există însă şi maşini de calcul la care se pot identifica mai mult de 5 niveluri ierarhice de maşini virtuale.

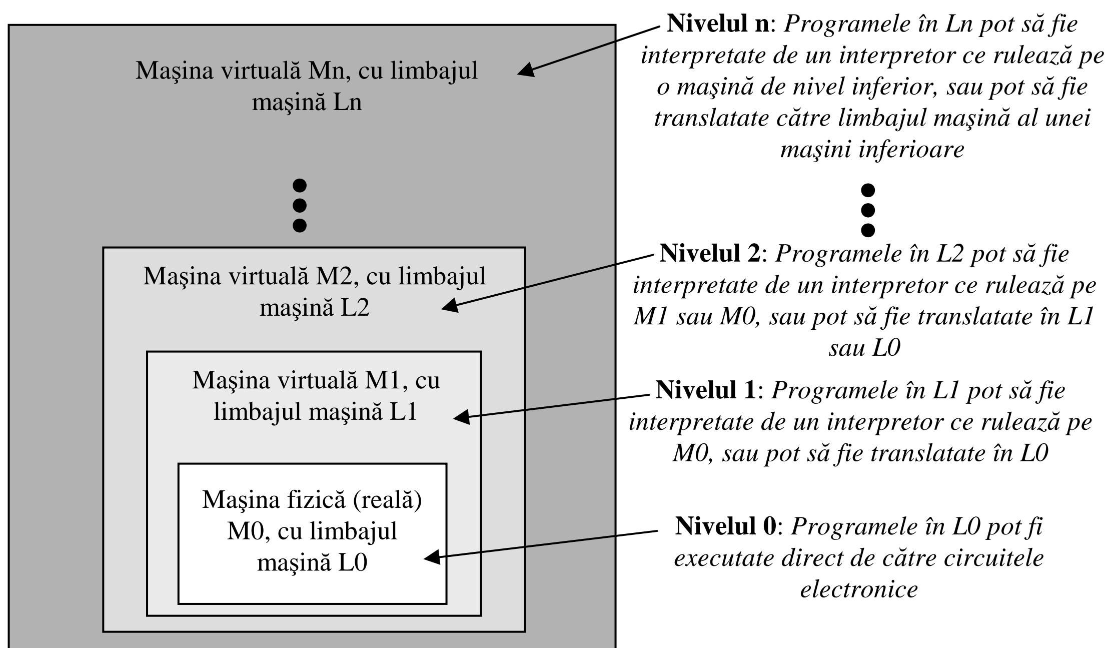


Figura 1.5. O maşină de calcul organizată din punct de vedere funcţional ca o succesiune de mai multe niveluri de maşini virtuale.

În figura 1.6. se observă că la nivelul cel mai de jos se găsesc dispozitivele şi circuitele electronice digitale care alcătuiesc structura hardware a calculatorului. La următoarele două niveluri există un nivel combinat hardware - software (firmware = software încorporat în hardware) care cuprinde pe lângă unităţile logice funcţionale (set de registre, ALU, unitate de control, etc.) şi un nivel de microprogram, încastrat într-o memorie locală de control. Acest nivel face parte din unitatea de control microprogramată, iar microprogramele se constituie ca un interpretor al instrucţiunilor de la nivelul inferior al unităţilor funcţionale. În cazul în care unitatea de control este cablată, nivelul de microprogram nu apare explicit în ierarhia maşinilor virtuale.

La următorul nivel se observă nivelul instrucţiunilor elementare recunoscute de procesor. Este nivelul funcţiilor de bază pe care le oferă un anumit procesor de calculator şi el este descris de către producători în manualele de descriere a procesoarelor. Acesta este nivelul ASI (*nivelul arhitecturii setului de instrucţiuni*), iar maşina virtuală *maşina de bază*.

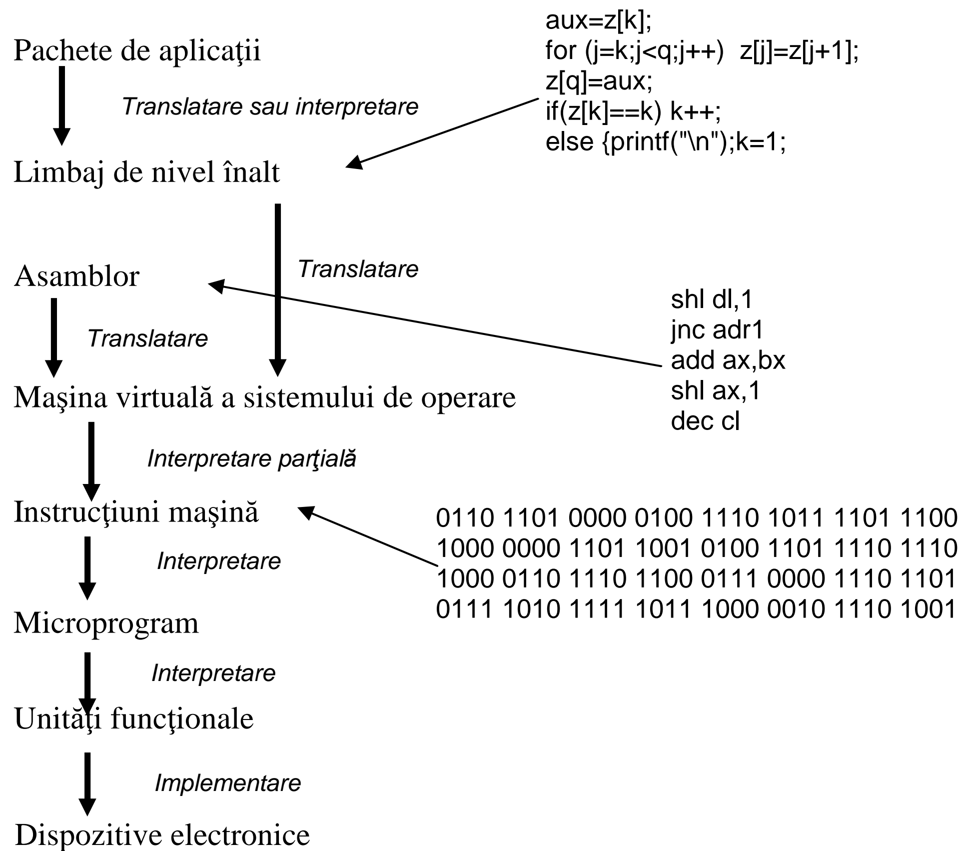


Figura 1.6. Ierarhia maşinilor virtuale şi a interfeţelor pentru un calculator de uz general cu procesor având control microprogramat.

Deşi maşina "înţelege" setul specific de instrucţiuni binare şi toate unităţile funcţionale sunt asamblate, la acest nivel maşina poate fi folosită cu greu şi numai dacă utilizatorul scrie în cod maşină nu numai programul de aplicaţie utilizator, dar şi rutinele necesare pentru accesul la perifericele de intrare şi de ieşire (tastatură, dispozitiv de afişare, disc magnetic, etc.). La acest nivel, dispozitivele de intrare / ieşire (I/O) pot fi controlate prin încărcarea de valori în registre speciale ale circuitelor de control ale acestor dispozitive I/O (controllere I/O).

De exemplu, controlul unei unităţi de disc magnetic, pentru citirea informaţiei de pe disc, se poate face prin încărcarea valorilor corespunzătoare adresei de pe disc, adresei memoriei principale, a numărului de octeţi de transferat, şi alte informaţii de stare şi sincronizare. În practică, sunt necesari mult mai mulţi parametrii decât cei pomeniţi, iar informaţia de stare returnată de circuitul de control al discului după o operaţie este foarte complexă. Mai mult, pentru multe dispozitive de I/O, timpul joacă un rol important în programare. Una din funcţiile majore ale unui sistem de operare este să ascundă toate aceste complexităţi şi să pună la dispoziţia programatorului un set de instrucţiuni simple cu care să lucreze. De exemplu o comandă de genul: "citeşte blocul X din fişierul Y" e mult mai simplă pentru că nu trebuie să ne

mai facem probleme privind detaliile mişcării capetelor de citire/scriere şi timpul de aşteptare pentru stabilizarea lor.

Nivelul maşinii virtuale a sistemului de operare este de fapt un sistem hibrid. [Tanenbaum99], pentru că cele mai multe dintre instrucţiunile (funcţiile) oferite sunt de asemenea instrucţiuni şi pentru nivelul imediat inferior. Funcţiile suplimentare oferite se referă la administrarea resurselor hardware şi software ale calculatorului (alocarea memoriei principale, administrarea memoriei secundare, posibilitatea de execuţie concurentă a programelor etc.). Uneori acest nivel este numit *nivelul maşinii extinse*, pentru că la acest nivel calculatorul este văzut de utilizator prin intermediul sistemului de operare (SO). Comunicarea utilizatorului, sau programului utilizator, cu SO se realizează pe două niveluri: prin intermediul limbajului de comandă cu ajutorul căruia utilizatorul solicita diferite resurse ale sistemului şi prin intermediul unor instrucţiuni speciale (apeluri sistem) la execuţia cărora sunt activate unele componente ale SO

Sistemul de operare nu este doar o interfaţă între utilizator şi maşina de calcul. El are şi sarcina de *administrator* al resurselor calculatorului. El trebuie sa administreze toate procesele (programele în cursul execuţiei, împreună cu contextul de lucru), memoriile internă şi externă, discurile magnetice, dispozitive de intrare / ieşire, etc. Sarcina sistemului de operare este să facă o alocare ordonată şi controlată a proceselor, blocurilor de memorie, dispozitivelor de I/O, între diferitele programe ce se află în competiţie pentru a le folosi. Atunci când un calculator are utilizatori multiplii, sarcinile de *administrare şi protecţie* ale sistemului de operare sunt şi mai evidente.

Sistemele de operare au o componenta de nivel inferior (un nucleu central, monitor, executiv, sau supervizor), care depinde de arhitectura maşinii de bază (de nivelul ISA) şi care controlează maşina de bază tot timpul cu excepţia momentelor când se rulează un program de aplicaţie (care însă poate folosi rutine ale executivului), iar la terminare, controlul e preluat din nou de executiv. Colecţia de rutine ce formează nucleul SO este componenta cea mai apropiată de hardware care realizează gestiunea la nivel fizic a operaţiilor de I/O, tratare a întreruperilor, încărcarea şi lansarea programului, citirea/scrierea unui caracter pe disc sau pe monitor etc. Orice modificare a configuraţiei maşinii de bază implică modificarea acestui nucleu al SO dar nu implică şi modificarea nivelurilor superioare. Nucleul SO este păstrat de obicei în memorie de tip ROM, dar în unele cazuri se poate stoca pe discul magnetic. De exemplu, sistemul de operare MS-DOS, are o componentă numită BIOS (Basic Input Output System) stocată în memorie de tip ROM, sau memorie RAM cu baterie tampon pentru păstrarea datelor şi la oprirea calculatorului. BIOS conţine programe de control ("drivers") pentru dispozitivele standard de I/O, acestea putând oferii o serie de servicii care degrevează programatorul şi rutinele sistemului DOS, de toate detaliile hardware ale circuitelor de interfaţa cu dispozitivele I/O. La PC aceste servicii pot fi apelate prin mecanismul întreruperilor software (INT nn), după o încărcare prealabilă cu valori

adecvate, ale registrelor maşinii. Celelalte componente ale sistemului MS DOS numite "io.sys" (conţine programele de control de I/O) şi "msdos.sys" (care conţine rutine de gestionare a proceselor, memoriei şi a sistemului de fişiere de pe discul magnetic, interpretarea apelurilor sistem) se încarcă de pe discul magnetic. Programele aplicative pot cere servicii sistemului de operare DOS (pot apela funcţiile DOS), prin intermediul întreruperii soft INT 21H, codul corespunzător funcţiei dorite fiind încărcat în registrul AH.

Notă privind programele de translatare asamblor şi compilator.

Chiar dacă s-a vorbit de programul executabil generat direct de translator (asamblor sau compilator) trebuie înţeles că programele lansate în execuţie rulează pe o maşină cu sistem de operare multitasking. Ca urmare în operaţia de translatare intervine un program de sistem suplimentar numit editor de legături (link-editor). De exemplu, la operaţia de asamblare, translatarea se face în două etape succesive:

1. în prima etapă face o **analiză lexicală şi sintactică** a programului sursă şi se generează o tabelă cu adresele simbolice folosite;
2. în a doua trecere, pe baza tabelei, se **genereaza codul obiect relocatabil** (toate adresele sunt relative la adresa de început a programului, nefiind legate de adrese fizice de memorie).

Editorul de legături ce rulează după asamblarea propriuzisă acceptă unul sau mai multe fişiere (module) obiect rezolvând în principal următoarele sarcini:

- ✓ rezolvarea referinţelor externe
- ✓ includerea de alte module obiect (scrise de utilizator, sau rutine din biblioteci);
- ✓ legarea codului de o adresă fizică (de încărcare a programului executabil în memorie);
- ✓ generarea fişierului executabil ce va putea fi încărcat şi lansat în execuţie de către sistemul de operare. Pentru aceasta editorul de legături generează în faţa fişierului executabil un antet (header) utilizat de sistemul de operare pentru încărcare, deschidere şi execuţie.
- ✓ generarea opţională, a unui listing prin care indică modul de link-edidare a modulelor (fişierelor) obiect.

1.9. Reprezentarea structurală a unui calculator

Structural, un calculator este format din unităţi funcţionale interconectate pentru a putea prelucra informaţia, pe baza comenzilor transmise prin program. Pentru un calculator uni-procesor, structura generală este cea din figura 1.8., ea corespunzând structurii propuse de von Neumann în 1945 pentru calculatorul secvenţial, cu program memorat. În cadrul structurii calculatorului secvenţial din figura 1.8. se disting următoarele unităţi [Sztojanov87]:

- *unitatea de intrare*, formată din echipamente *periferice de intrare* şi *sistemul de intrare* al calculatorului,
- *unitatea centrală* a calculatorului, formată din *memoria principală* (memorie ce conţine date şi instrucţiuni) şi *Unitatea Centrală de Procesare* (UCP),
- *unitatea de ieşire*, formată din echipamente *periferice de ieşire* şi *sistemul de ieşire* al calculatorului.

Echipamentul periferic de intrare, preia datele din exterior (prin diverse traductoare) şi le transformă ca natură fizică şi format de reprezentare, în aşa fel încât informaţia să fie compatibilă cu nivelurile de tensiuni şi formatul de reprezentare binar folosit de sistemul de intrare al calculatorului.

Echipamentul periferic de ieşire are funcţia inversă perifericului de intrare, el preluând datele de la sistemul de ieşire al calculatorului. Echipamentul periferic de ieşire transformă datele primite într-o formă corespunzătoare elementului de execuţie comandat de calculator. Dacă este vorba de dispozitive de memorie externă informaţia se stochează pe suportul extern, conversia informaţiei binare făcându-se în conformitate cu tipul circuitului de control şi cu natura fizică a suportului de stocare (hârtie, semiconductor, material magnetic etc.).

Sistemul de intrare / ieşire (I/O) al calculatorului este locul prin care se face schimbul de informaţii între unitatea centrală şi echipamentele periferice. Acest transfer se face prin intermediul unor locaţii adresabile de către procesor (UCP), numite porturi, ele făcând parte din sistemul I/O al calculatorului. Sistemul I/O realizează operaţii de genul: modifică formatul de reprezentare a datelor (din serie în paralel, sau invers), verifică corectitudinea informaţiei transferate, asigură sincronizarea dintre echipamentul periferic şi UCP din punctul de vedere al vitezelor de transfer (sincronizarea este necesară datorită vitezelor de lucru mult diferite între UCP şi echipamentele periferice).

Unitatea centrală asigură prelucrarea automată a datelor, prin interpretarea instrucţiunilor unui program. Programul rulat (aflat în execuţie) este stocat împreună cu datele aferente în memoria principală (internă calculatorului). Introducerea datelor iniţiale, comanda lansării în

execuţie, afişarea, tipărirea sau transmisia la alte dispozitive a rezultatelor se face prin unităţile de intrare, respectiv de ieşire ale sistemului de calcul.

Ca structură internă, unitatea centrală este formată din memoria principală şi din Unitatea Centrală de Procesare (UCP). La rândul ei UCP include calea de date (ALU, registre, magistrale interne pentru transferul informaţiilor) şi unitatea de control.

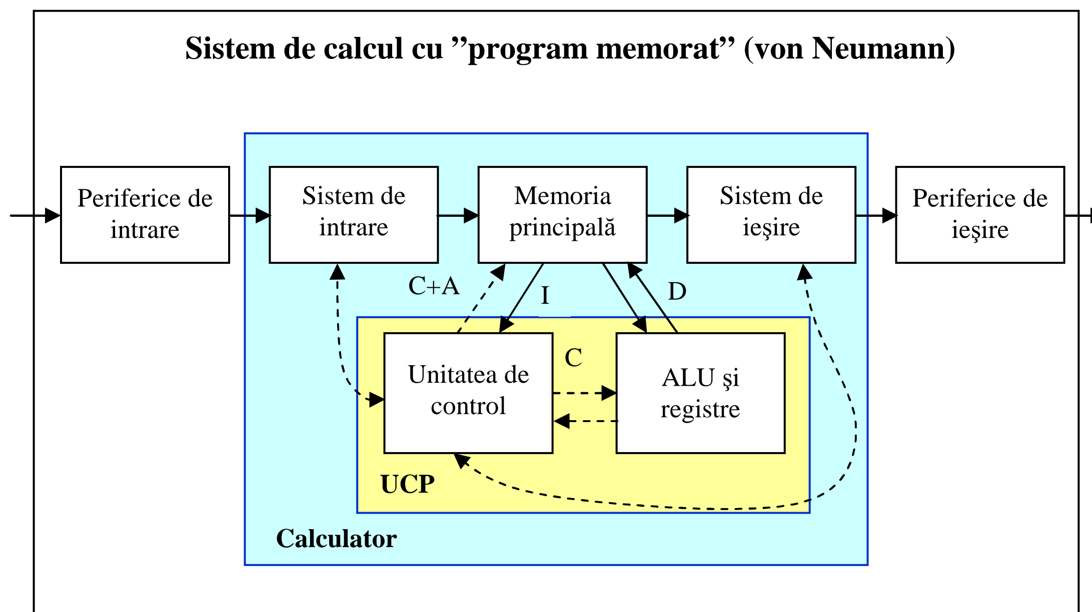


Figura 1.8. Reprezentare structurală a unui calculator uni-procesor de uz general. Cu linie punctată s-au reprezentat comenzile (C) transmise de la unitatea de control şi semnalele de stare-sincronizare ca răspuns al unităţilor comandate către unitatea de control. În cadrul semnalelor de control pentru memoria principală a fost inclusă şi informaţia de adresare (A) a memoriei. Căile de transfer pentru date (D) şi instrucţiuni (I) sunt reprezentate cu linie plină. ALU = Unitate Aritmetică şi Logică.

Unitatea de control (UC), adresează (A) şi citeşte din memorie instrucţiunile binare ale programului, le interpretează şi generează în consecinţă semnale de comandă către toate celelalte unităţi funcţionale ale calculatorului. Ca urmare a acestor semnale instrucţiunea curentă (I) este executată. În plus UC analizează semnale de stare şi sincronizare, ce provin de la celelalte unităţi funcţionale şi ca urmare poate schimba succesiunea semnalelor de control pe care le generează, sau poate modifica momentele de timp ale succesiunii semnalelor de comandă, pentru a realiza sincronizarea cu celelalte unităţi. Există două moduri de implementare a unităţii de control:

- control cablat: dacă unitatea de control este realizată ca un automat secvenţial convenţional, la care algoritmul de interpretare a fiecărei instrucţiuni binare este inclus în circuitele fizice (hardware). Pentru modificarea setului de instrucţiuni recunoscut unitatea de control trebuie reproiectată complet.
- control microprogramat (micro-codat): dacă succesiunea de semnale de comandă specifice fiecărei instrucţiuni (succesiune numită microprogram) se înscrie într-o memorie

de control locală, de obicei, memorie doar cu citire. Pentru modificarea setului de instrucţiuni recunoscut trebuie rescris conţinutul memoriei de control.

Tot în cadrul unităţii de control includem şi circuitele pentru generarea adreselor, care calculează adresele corecte (adrese pentru memoria principală, pentru porturi de I/O, sau pentru registrele interne ale UCP) pe baza informaţiilor binare din corpul instrucţiunilor.

Unitatea aritmetică şi logică (ALU[6]), realizează operaţii logice sau aritmetice cu operanzii adresaţi de UC. Înainte de prelucrare, operanzii se stochează într-un set de registre de uz general, folosite ca memorie temporară. Registrele reprezintă o memorie locală UCP, de foarte mare viteză. Setul de registre de uz general poate fi folosit însă şi pentru salvarea diferitelor informaţii privind adresarea memoriei principale.

În funcţie de rezultatul operaţiilor efectuate, ALU setează anumiţi indicatori de condiţii (indicatori de stare, fanioane[7]) care pot fi citiţi de UC şi pot astfel modifica secvenţa de tranziţie a stărilor acestui automat.

Unitatea de memorie principală (sau memorie internă) are funcţia de stocare a programelor şi datelor aferente acestora.
Transferul datelor cu registrele UCP se face conform comenzilor date de unitatea de control. Memoria principală este realizată în prezent exclusiv în tehnologie semiconductoare. Aceasta unitate de stocare a informaţiei are, în principiu, o organizare liniară, constând din locaţii (registre) de memorare, fiecare de câte d biţi, locaţia fiind selectabilă printr-o adresă unică. Adresa poate lua valori cuprinse între 0 şi 2^a-1, unde prin *„a”* s-a notat numărul de biţi ai cuvântului de adresă fizică.
Din punctul de vedere al denumirilor folosite facem următoarele observaţii:

- Ansamblul format din UC, ALU şi registre este numit *Unitate Centrala de Procesare* (UCP) fiind o structură de *procesor* de uz general cu set de instrucţiuni.
- UCP împreună cu memoria principală formează Unitatea Centrală, iar
- Unitatea centrală împreună cu sistemul de I/O şi setul de programe de sistem constituie structura de *calculator*.
- Un calculator împreună cu diversele echipamente periferice formează un *sistem de calcul*.

Structura logică de bază a calculatorului uni-procesor, prezentată anterior, conform figurii 1.8. corespunde , în bună măsură, celei stabilite de John von Neumann în 1945. În lucrarea "Prima schiţă de Raport asupra lui EDVAC", el a definit structura logică de bază a ***calculatorului cu program memorat***, menţionând cinci criterii necesare a fi îndeplinite de acesta [Hayes88]:

[6] ALU = Arithmetic and Logic Unit (engl.) - Unitate aritmetică şi logică
[7] flag (engl.) = fanion, indicator de condiţii / de stare

1. să posede un mediu de intrare care să permită introducerea unui număr nelimitat de operanzi şi instrucţiuni.
2. să posede o memorie din care să se citească operanzi şi instrucţiuni şi în care să se poată introduce, în ordinea dorită, rezultatele.
3. să dispună de o secţiune de calcul, capabilă să efectueze operaţii aritmetice şi logice asupra operanzilor citiţi din memorie.
4. să posede un mediu de ieşire, care să permită livrarea unui număr nelimitat de rezultate către utilizator.
5. să posede o unitate de comandă capabilă să interpreteze instrucţiunile citite din memorie şi să selecteze diverse variante de desfăşurare a operaţiilor, în funcţie de rezultatele obţinute pe parcurs.

Marea majoritate a calculatoarelor construite până în prezent se bazează pe principii rezumate mai sus, fiind numite calculatoare de tip von Neumann, sau cu arhitectură von Neumann.

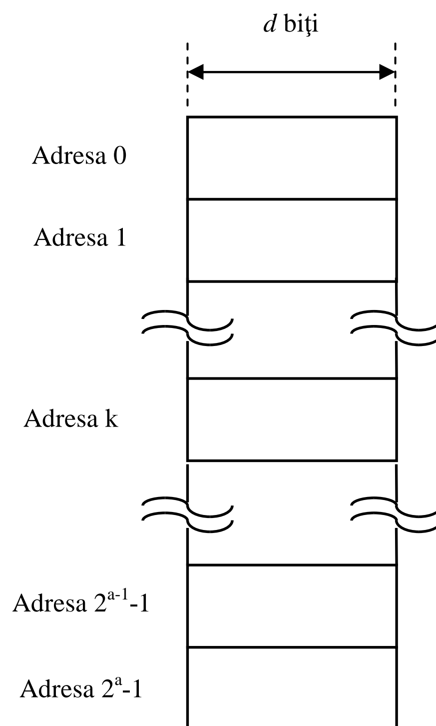


Figura 1.9. Memoria principală prezentată ca o succesiune de locaţii de memorie, fiecare locaţie având o adresă specifică. S-a presupus că adresa de memorie are a biţi, iar în fiecare locaţie, cu adresă unică, se poate stoca un cuvânt binar cu lungimea de d biţi.

1.10. Organizarea ierarhică a sistemului de memorie

Dispozitivele de memorare din cadrul unui sistem de calcul au rolul de a stoca informaţia numerică (programe, date iniţiale, rezultate intermediare şi finale) cu posibilitatea ca informaţia numerică să poată fi accesata printr-o adresa, pentru a fi citita sau scrisa.

Dispozitivele de stocare a informaţiei plus algoritmii de control ai memoriei (implementaţi fie în software fie în hardware) constituie *sistemul de memorie al calculatorului*.

Pentru un sistem de calcul unul din scopurile principalele este prelucrarea unei cantităţi cât mai mari de date în unitatea de timp (pentru a defini aceasta performanţă globală de viteză a sistemului se foloseşte termenul "throughput" din limba engleză). Pentru creşterea performanţelor sistemului de calcul se urmăreşte nu numai creşterea vitezei de lucru a procesorului (UCP), dar şi creşterea vitezei de transfer a datelor cu sistemul de memorie, şi cu celelalte echipamente de intrare-ieşire (I/O). De obicei în cadrul calculatorului, sistemul de memorie are "onoarea" de a fi frânarul. Cantitatea maxima de date ce se pot transfera cu memoria în unitatea de timp e numita lărgime de banda a memoriei (in engleza "bandwidth").

Din punctul de vedere al procesorului, ideal ar fi ca să existe acces la o memorie unică, rapidă şi de dimensiuni suficient de mari pentru aplicaţiile rulate. Practic însă, memoriile ce lucrează la viteze comparabile cu cea a procesorului sunt scumpe. Nu este posibil (cu excepţia calculatoarelor foarte mici) să se folosească o singură memorie, ce utilizează doar un singur tip de tehnologie. De aceea, în proiectarea şi construcţia unui sistem de memorie, se face un compromis, al cărui obiectiv este obţinerea unei viteze de acces medii cât mai mari în condiţiile unui cost total minim, sau rezonabil, pentru sistemul de memorie. Datorită restricţiilor impuse de costuri, gabarit şi consum de putere, în condiţiile în care se doresc performante cât mai bune de la sistemul de memorie se folosesc următoarele căi de realizare ale acestui sistem:

- informaţia stocată este *distribuită* pe o varietate de dispozitive de memorie, dispozitive cu raport cost / performanţă şi caracteristici fizice foarte diferite. De aceea ele sunt astfel organizate pentru a asigura o *viteza medie cât mai mare*, la un *cost specific mediu cât mai mic*. Dispozitivele de memorie formează o ierarhie de dispozitive de stocare (vezi figura 6.1).
- *transferul de informaţii* între diferitele niveluri ale memoriei trebuie să se desfăşoare *transparent* pentru utilizator (intră în sarcina sistemului de operare). De asemenea se urmăreşte realizarea unor metode de *alocare dinamică* a spaţiului disponibil de memorie, pentru o utilizare cât mai eficientă.
- realizarea conceptelor *memoriei virtuale* (bazate pe metodele de alocare dinamică a spaţiului de memorie principală) pentru a elibera programele utilizator de funcţia administrării spaţiului de memorie şi pentru a face programele relativ independente de

configuraţia dată a memoriei fizice.

- creşterea *lărgimii de bandă* a sistemului de memorie şi asigurarea unor mecanisme de *protecţie*, pentru a preveni ca programele utilizator sa altereze zonele de memorie ale altor programe utilizator sau ale sistemului de operare.

În ceea ce priveşte organizarea ierarhica a dispozitivelor de memorie, pomenită mai sus, fiecare nivel de stocare are utilizare, viteză şi dimensiuni specifice. Nivelul cel mai apropiat de UCP are capacitatea de stocare cea mai mică dar viteza cea mai mare. Vom numi acest nivel "*memoria locală a procesorului*". Ea este constituită dintr-un set de registre, integrate în UCP şi este o memorie de mare viteză folosită pentru stocarea temporara a instrucţiunilor şi datelor. UCP are *acces direct* la această memorie (nivelul 1 în figura 1.10). Noţiunile privind modul de lucru al registrelor interne s-au descris în capitole anterioare şi deci nu vor fi tratate în continuare.

Următorul nivel are o capacitate relativ mare de stocare şi o viteză mare, dar cu valoare mai mică decât viteza memoriei locale a UCP. Memoria de pe acest nivel este numita "*memorie principală*" (dar şi memorie internă, sau memorie operativă). Memoria principala este folosită pentru stocarea programelor şi datelor necesare la un anumit moment de timp. Programele utilizator se încarcă (copiază) în memoria principală doar atunci când se lansează în execuţie.

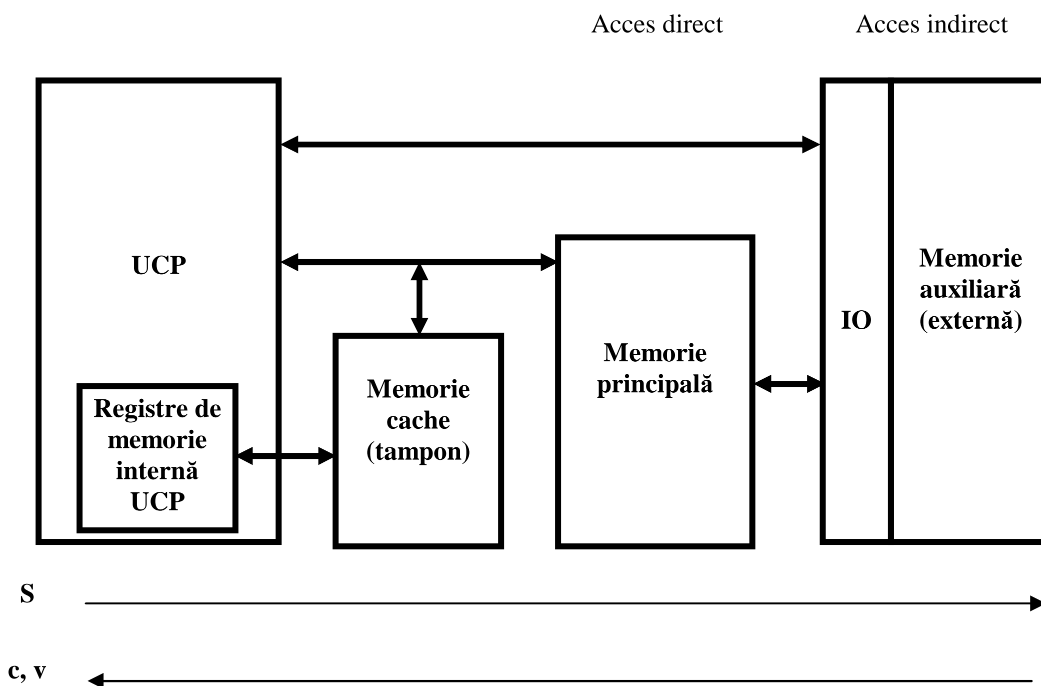


Figura 1.10. Evidenţierea organizării ierarhizate a memoriei într-un sistem de calcul: Nivelul 1 este nivelul de memorie internă UCP, iar pe nivelurile 2 şi 3 se găsesc memoria principală, respectiv memoria auxiliară (externă), privită ca un echipament periferic de I/O pentru stocarea datelor. Parametrii viteza (v), cost specific (c) şi capacitate de stocare (S), cresc în direcţiile indicate de săgeţile de sub figura. Memoria cache este o memorie tampon, intermediară nivelurilor 1 şi 2.

Memoria principală este caracterizată de faptul că locaţiile sale pot fi accesate direct şi

rapid de setul de instrucţiuni al UCP. Memoria principală este realizată în tehnologie semiconductoare (circuite integrate).

Ultimul nivel, cel mai îndepărtat de UCP este numit "memorie secundară" (memorie auxiliara, sau memorie externă) şi are o capacitate foarte mare de stocare şi o viteză mult mai redusă decât memoria principală. Este folosită pentru stocarea programelor de sistem, a fişierelor de program şi date şi altele de acest gen, care nu sunt cerute permanent de UCP. Informaţia din memoria auxiliara este accesată *indirect* de către UCP, prin intermediul sistemului de intrare - ieşire (I/O), cu ajutorul căruia informaţia cerută de UCP este adusă mai întâi în memoria principală. Tehnologii reprezentative pentru realizarea memoriei auxiliare sunt: magnetică (discuri şi benzi), optică, semiconductoare.

Utilizarea unei ierarhii de dispozitive de memorie este determinată de asigurarea unui compromis între cerinţele impuse de programele rulate (capacitatea de stocare, viteză medie de transfer cu memoria, protecţia informaţiei) şi costurile sistemului de memorie (tehnologie disponibilă, preţ, gabarit, algoritmi de transfer între diferitele niveluri ale ierarhiei de memorie).

Datorită faptului că viteza de lucru a procesoarelor de uz general a crescut continuu în ultimii ani (în mod curent peste 1GHz), iar viteza dispozitivelor de memorie folosite în cadrul memoriei principale nu a crescut în aceeaşi măsură, în multe din maşini se utilizează un nivel intermediar de memorie (intermediar ca poziţie faţă de UCP, dar şi ca viteză şi capacitate de memorare) numită memorie "cache". Această memorie are doar rol de memorie temporară, fiind un tampon de viteză mare între registrele interne ale procesorului şi registrele ce constituie locaţiile memoriei principale. Memoria cache conţine în permanenţă doar copii din memoria principală ale zonelor de program şi date active la un moment dat. Spre deosebire de cele trei niveluri de organizare a memoriei, memoria cache este transparentă pentru programator / utilizator (nu este vazută efectiv, pentru că nu există adrese separate pentru cache şi pentru memoria principală. S-a mers chiar mai departe, încât la multe din procesoarele moderne se folosesc două sau mai multe niveluri intermediare de cache, cel mai apropiat de UCP fiind mai mic şi mai rapid şi încastrat în acelaşi circuit integrat cu UCP. Conform aceluiaşi principiu de compensare a diferenţelor de viteză dintre niveluri, între memoria principală şi memoria auxiliară pot apărea niveluri de memorie tampon, de asemenea transparente utilizatorului. Asemenea memorii tampon sunt specifice fiecărui dispozitiv periferic folosit ca memorie externă şi se găsesc de obicei în cadrul circuitelor de control ale acestor dispozitive.

Este important de observat (figura 1.10) că datorită vitezei mari de lucru a memoriei cache, este de preferat ca toate operaţiile procesorului să se desfăşoare cu memoria cache, care permite de obicei citiri / scrieri într-o singură perioadă de ceas. Pentru ca acest lucru să fie posibil, circuitul care controlează memoria cache (controller de cache) trebuie să facă o anticipare a zonei din memoria principală în care se găsesc informaţiile ce vor fi cerute de programul rulat de UCP. Controllerul este responsabil de copierea acestor informaţii din memoria principală în memoria

cache şi de scrierea în memoria principală a zonelor de date ce se elimină din cache şi care au fost modificate în cache.

Memoria principală are un timp de acces moderat ca valoare (zeci de nanosecunde) şi o capacitate de stocare mai mare de câţiva zeci de mega-octeţi. Transferul direct între UCP şi memoria principală se face numai în cazul când informaţia cerută nu a fost găsită în cache (evenimentul este numit ratare la acces cache[8]).

Memoria auxiliară poate fi realizată în diferite tehnologii, având capacităţi de stocare mari şi foarte mari (peste ordinul giga-byte în prezent), iar timpul de acces depinde puternic de tipul memoriei auxiliare şi de modul cum se face transferul de informaţie între memoria auxiliară şi cea principală. Legătura directă dintre cele doua niveluri de memorie, simbolizată în figura 1.10, indică un transfer prin acces direct la memorie (DMA[9]) între perifericul memorie auxiliară şi memoria principală. Transferul de informaţie între memoriile principală şi auxiliară poate fi făcut însă şi prin intermediul UCP, când UCP citeşte sau scrie prin program memoria auxiliară.

Schimbul de informaţii între diversele niveluri ale memoriei, trebuie să aibă un caracter transparent pentru utilizator. Aceasta se realizează prin folosirea unor resurse hardware şi a unor tehnici software pentru administrarea şi alocarea spaţiului memoriei (principale) cu acces direct.

De obicei, la rularea unui program se foloseşte proprietatea numită *"localizarea referinţelor la memorie"*. Pe baza acestei proprietăţi sistemul de memorie păstrează cele mai recent accesate articole, pe cât posibil, în memoria cea mai rapidă (care are şi cea mai mică capacitate de stocare). Localizarea referinţelor la memorie, observabilă în majoritatea programelor, se exprimă sub două forme:

- localizare *temporală* a referinţelor: dacă programul face acces la o celulă de memorie la momentul t, este foarte probabil ca programul să facă din nou acces la aceeaşi celulă la momentul $t + \Delta$
- localizare *spaţială* a referinţelor: dacă la momentul t programul face acces la o celulă de memorie de adresă X, este foarte probabil ca la momentul $t + \Delta$ programul să facă acces la celula de adresă $X + \varepsilon$.

Pentru a folosi proprietatea referinţelor localizate în timp, datele cele mai recent accesate trebuie păstrate cât mai aproape de procesor. Pentru a folosi proprietatea referinţelor localizate în spaţiu la transferul între niveluri trebuie mutate blocuri continue de date şi nu cuvinte individuale.

Dar organizarea ierarhică şi folosirea proprietăţii localizării referinţelor la memorie, face ca pe diferite niveluri ierarhice să existe informaţii copii ale aceluiaşi bloc de date, date ce pot fi modificate doar pe nivelul cel mai apropiat de UCP, sau şi pe nivelurile inferioare. De aceea

[8] cache mis
[9] DMA = Direct Memory Access

sistemele de memorie multi-nivel, cu posibilitatea de scriere-citire trebuie să satisfacă două proprietăţi privind informaţia stocată:

- proprietatea de *incluziune*
- proprietatea de *coerenţă*

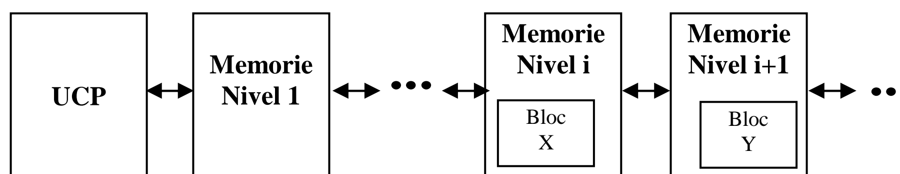


Figura 1.11. Sistemul de memorie indicat ca o succesiune de niveluri de memorie. Creşterea distanţei faţă de UCP reprezintă, în general, o creştere a timpului de acces, dar şi o creştere a capacităţii de stocare.

Pentru explicarea celor două proprietăţi se desenează ierarhia de memorie ca o succesiune de niveluri de memorie (M_1, M_2, ..., M_n) ca în figura 1.11. Proprietatea de *incluziune* exprimă faptul că totdeauna informaţiile care se găsesc pe un anumit nivel de memorie se vor găsi şi pe nivelul de memorie inferior ($M_1 \subset M_2 \subset ... \subset M_n$). Este posibil ca o informaţie existentă pe nivelul M_{i+1} să nu se găsească pe nivelul M_i, dar să fie accesată (cerută) pe nivelul *i* de memorie. În acest caz (evenimentul este numit "word miss") cuvântul necesar a fi accesat trebuie adus pe un nivel de memorie superior (de pe nivelul i+1, pe nivelul i). Cu alte cuvinte, la accesul memoriei organizate multi-nivel, atunci când informaţia este accesată prin adresă pe nivelul i, iar cuvântul cu adresa căutată se găseşte pe nivelul i (de exemplu este inclusă în blocul X stocat în nivelul i din figura 6.2) evenimentul este o *reuşită* ("*hit*") şi nu se pierde timp suplimentar pentru căutarea informaţiei pe nivelurile inferioare de memorie. Pentru a evalua numărul de reuşite la accesul pe nivelul i de memorie, într-un interval de timp, se foloseşte parametrul numit *raport de reuşită ("hit ratio")*, numit şi *rată de reuşită ("hit rate")*. Se calculează ca raport între numărul total de reuşite şi numărul total de accesări la nivelul i de memorie. Atunci când informaţia este accesată prin adresă pe nivelul i, iar cuvântul cu adresa căutată nu se găseşte pe nivelul i ci doar pe nivelul inferior i+1 (de exemplu este inclusă în blocul Y stocat în nivelul i+1 din figura 6.2) evenimentul este o *ratare* ("*miss*") şi se pierde un timp suplimentar ("*miss penalty*") pentru căutarea informaţiei pe nivelurile inferioare de memorie.

Proprietatea de *coerenţă* exprimă faptul că informaţia existentă la o anumită adresă în spaţiul de memorie trebuie să fie aceeaşi, indiferent de nivelul de memorie pe care se află. Dacă un cuvânt este modificat pe nivelul i, atunci va trebui modificat şi pe nivelul i+1 şi pe toate nivelurile inferioare. Coerenţa nivelurilor de memorie se poate obţine fie prin propagarea valorii

scrise / modificate spre toate nivelele inferioare ("write-through"), sau prin actualizarea nivelurilor inferioare doar în momentul înlocuirii informaţiei de pe nivelul curent ("write-back"). Se va vorbi mai mult despre reuşită / ratare şi modalităţile de păstrare a coerenţei informaţiei în capitolul care descrie memoria cache.

1.11. Organizarea magistralelor calculatorului

1.11.1. Organizarea ierarhică a magistralelor calculatorului

Conform aspectelor structurale, prezentate în paragraful precedent, informaţiile principale pe care microprocesorul le schimbă cu exteriorul sunt: date (operanzi şi rezultate) , instrucţiuni, adrese, informaţii de control. Toate acestea se transmit ca semnale electrice (codificate binar) prin linii conductoare grupate în magistrale. Ansamblul liniilor electrice la care se conectează microprocesorul, memoria şi sistemul de I/O al unui calculator este numit: *magistrală sistem*. În mod tradiţional liniile incluse în magistralele externe microprocesorului sunt clasificate funcţional în:

- magistrală de date (Bus de date)
- magistrală de adrese (Bus de adrese)
- magistrală de control (Bus de control)

Legarea diferitelor componente ale calculatorului la aceste trei magistrale este simbolizată în figura 1.12. În descriere se va folosi şi termenul împrumutat din limba engleză pentru magistrală: "bus"

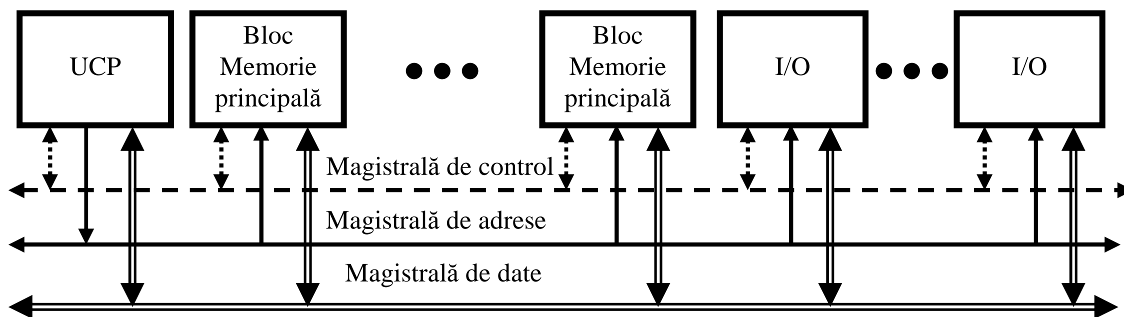


Figura 1.12. Schemă bloc generală ce indică conectarea componentelor calculatorului la cele trei tipuri de magistrale externe UCP. Prin I/O sau simbolizat circuitele de interfaţă cu echipamentele de intrare / ieşire.

Magistrala este un canal de comunicaţie partajat în timp de mai multe unităţi legate la liniile magistralei. Partajarea (multiplexarea) în timp este necesară, pentru că doar un dispozitiv poate transmite date pe magistrală la un moment dat. Din acest motiv, conectarea a mai multe dispozitive la aceeaşi magistrală sistem poate deveni ineficientă din punctul de vedere al vitezei de comunicaţie pe o singură magistrală partajată. Pentru a înlătura supraîncărcarea canalului de comunicaţie, şi a creşte performanţele acestuia, se construieşte o ierarhie de mai multe

magistrale interconectate. Principalele motive care impun construcţia mai multor magistrale, organizate ca o ierarhie de magistrale, în funcţie de viteză, sunt:

- Cu cât sunt mai multe dispozitive conectate la o magistrală, cu atât mai mare este lungimea fizică a magistralei. Aceasta poate conduce la întârzieri mari la transmisia semnalului digital cu frecvenţe foarte mari ale ceasului.
- Cu cât sunt mai multe dispozitive conectate la o magistrală, cu atât mai mare va fi întârzierea produsă la transmisia datelor, datorită timpilor de aşteptare. În plus cu cât sunt mai multe dispozitive conectate la aceleaşi linii ale magistralei, creşte capacitatea parazită a fiecărei linii, ceea ce produce un efect de integrare a semnalului digital şi posibilitatea ca un semnal digital transmis să nu mai respecte nivelul logic High.
- Dispozitivele conectate la o magistrală au viteze mult diferite de funcţionare. Comparaţi viteza microprocesorului (UCP) cu a unui echipament periferic

În calculatoarele moderne transferul dintre UCP şi memorie, respectiv dispozitivele de I/O se face prin magistrale organizate ierarhic, în funcţie de viteza dispozitivelor cuplate la fiecare magistrală. Există mai multe variante de organizare, în funcţie de tipul calculatorului (de uz general sau pentru aplicaţii specifice) şi diferenţele de viteză între dispozitive. De aceea marea majoritate a calculatoarelor utilizează mai multe magistrale, organizate ca o ierarhie de magistrale cu viteze de operare diferite. O structură clasică de magistrale multiple organizate pe niveluri este prezentată în figura 1.13. [Stallings00].

Există o magistrală locală care conectează procesorul cu memoria cache şi la care se pot conecta şi câteva dispozitive de I/O (locale) de mare viteză. Controllerul de memorie cache interfaţează această memorie cu procesorul şi de asemenea cu magistrala sistem la care este conectată memoria principală. Avantajul acestui mod de organizare este că transferul direct între dispozitivele de I/O şi memorie nu interferează cu activitatea procesorului care lucrează direct cu memoria cache. Dispozitivele de I/O sunt conectate la al treilea nivel ierarhic numit magistrală de extensie. Acest ultim mod de aranjare permite conectarea la magistrala de extensie a unui mare număr şi o mare diversitate de dispozitive de I/O şi în acelaşi timp izolează traficul procesor - memorie faţă de traficul de I/O.

În figura 1.13 se exemplifică câteva dispozitive de I/O ce pot fi cuplate la magistrala de extensie. Conexiunile de tip reţea includ reţele locale (LAN - local area network), sau conexiuni la reţele pe arii extinse. Controllerul SCSI (small computer system interface) conectează la magistrala de extensie o magistrală SCSI la care se pot conecta controllere de hard disc locale şi alte periferice. Portul serial poate fi folosit pentru conectarea unei imprimante sau a unui scanner. Magistralele de extensie (numite şi magistrale de I/O) suportă o gamă largă de rate de transfer, pentru a permite conectarea unei game largi de dispozitive I/O.

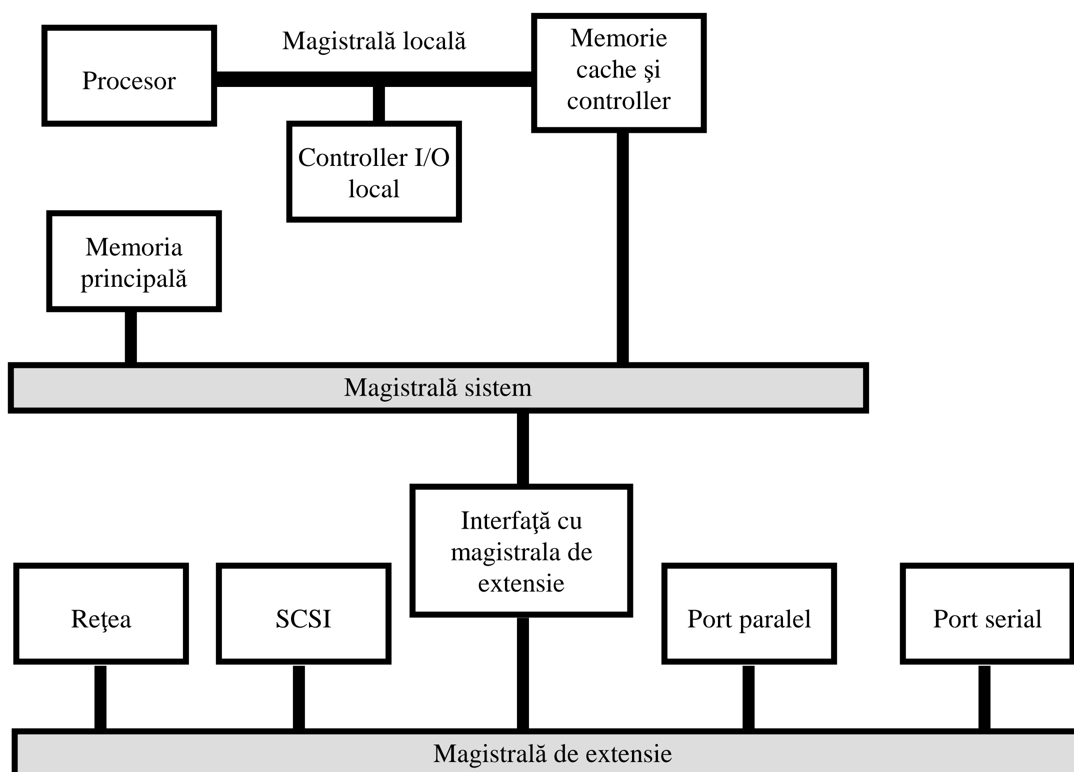


Figure 1.13. Exemplu de organizare ierarhică a magistralelor calculatorului [Stallings00]

Arhitectura clasică a magistralelor este eficientă, dar nu face faţă la noile dispozitive de I/O ce funcţionează la viteze din ce în ce mai mari. Ca urmare au apărut noi organizări, care introduc un nivel intermedia de magistrală de mare viteză (numit uneori magistrală la mezanin) care se interfaţează cu o punte (bridge) cu magistrala locală a procesorului. Figura 1.14 prezintă această abordare, în care controllerul de cache este integrat într-o punte, sau dispozitiv tampon, care se conectează la magistrala de mare viteză. La această magistrală de mare viteză se pot conecta circuite controller de mare viteză pentru LAN, (cum ar fi Fast Ethernet la 100 Mbps, controller video-grafic). Dispozitivele de I/O cu viteză mică se cuplează în continuare la magistrala de extensie conectată printr-o interfaţa cu magistrala de mare viteză. Avantajul acestei aranjări este că dispozitivele de I/O de mare viteză sunt integrate mai aproape de procesor şi în acelaşi timp pot funcţiona în paralel cu procesorul.

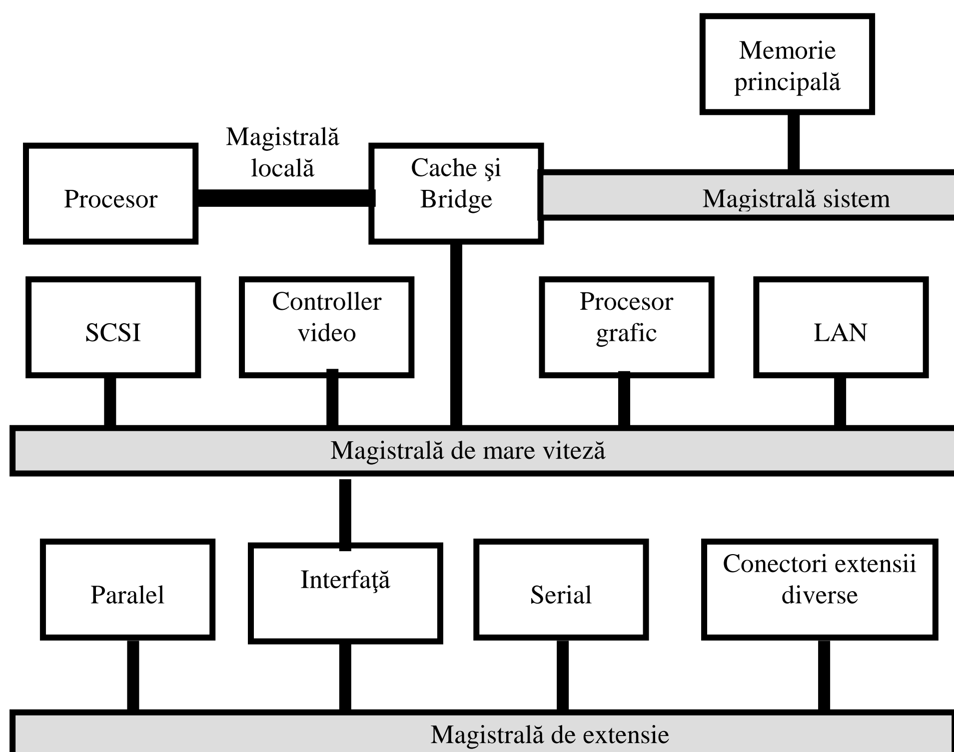


Figura 1.14. Exemplu de organizare ierarhică a magistralelor la care se introduce magistrala de mare viteză "de la mezanin" [Stallings00]

1.11.2. Caracteristici principale ale magistralelor

Caracteristicile principale ale magistralelor includ: *modul de arbitrare* (permisiunea de a transmite date pe liniile magistralei se poate face centralizat, sau descentralizat), sincronizare (dacă semnalele de pe magistrală sunt sincronizate cu un semnal de ceas, sau sunt trimise asincron condiţionat de semnale transmise anterior) și *lățimea magistralei* (numărul de linii de adresă / date).

Arbitrarea magistralelor este necesară în toate calculatoarele, chiar şi în cele uni-procesor. Aceasta pentru că de exemplu un periferic poate solicita transfer de date direct cu memoria, cu ajutorul interfeţei de I/O şi a unui controler de acces direct la memorie (DMA - Direct Memory Access). Pentru efectuarea transferului controlerul DMA va prelua temporar controlul magistralelor. Diferitele metode de arbitrare a magistralelor pot fi încadrate în două categorii: arbitrare *centralizată* şi arbitrare *descentralizată*.

Într-un sistem *centralizat* de arbitrare, există un singur dispozitiv hardware, numit controler de magistrală, sau arbitru, alocă timp pentru preluarea controlului magistralelor. Modul de arbitrare poate fi programabil, sau fix în funcţie de priorităţile alocate diverselor dispozitive master conectate la magistrală Cea mai simplă schemă de arbitrare centralizată fixă poate fi

concepută cu ajutorul unui codificator prioritar, în aşa fel ca în cazul solicitării de control a magistralelor de două sau mai multe dispozitive, să fie servit cel cu prioritatea cea mai mare. Pentru prioritatea *descentralizată*, după cum sugerează şi denumirea, nu mai există arbitru central. Fiecare dispozitiv ce poate prelua controlul magistralelor conţine logica necesară pentru stabilirea priorităţii. Priorităţile sunt fixe şi încorporate în hardware - prin modalitatea de interconectare a modulelor. Adesea în acest caz se foloseşte legarea dispozitivelor în lanţ de priorităţi. A se vedea capitolul referitor la sistemul de I/O pentru o descriere a lanţului de priorităţi.

Legat de *lăţimea* magistralei, dimensiunea în număr de biţi indică numărul total de biţi ce pot fi transferaţi în unitatea de timp. Cu cât numărul de linii este mai mare cu atât creşte viteza de transfer a informaţiei pe magistrală, dar de asemeneaa cresc şi costurile implementării.

Din punctul de vedere al modului de transfer a informaţiilor pe magistrale, acestea pot fi sincrone, sau asincrone. Magistralele *sincrone* sunt comandate de un semnal de ceas local. Toate transferurile pe aceste magistrale, numite *cicluri de magistrală*, respectă un protocol fix ce impune un anumit număr de impulsuri de ceas. Datorită sincronizării controlul transferurilor este extrem de simplu. Blocurile de interfaţă (bridge) între magistralele sincrone ce funcţionează la diferite frecvenţe de ceas trebuie să realizeze adaptarea de viteză în aşa fel încât transferurile între magistrale să se facă corect şi cât mai rapid.

La magistralele *asincrone* transferurile nu mai trebuie să se încadreze într-un interval fix de timp, iar controlul transferurilor se face cu ajutorul unor semnale de control între cei doi corespondenţi (*handshaking)*. Pentru majoritatea procesoarelor ce pot organiza spaţiu separat de adrese pentru porturile de I/O, în ciclurile de transfer cu porturile se introduc automat (de către UCP) stări de aşteptare (wait), care adaptează viteza UCP la viteza scăzută a dispozitivelor de I/O. Dacă porturile sunt organizate în spaţiul de memorie, acest mecanism de sincronizare, cu ajutorul stărilor de wait, trebuie construit în exteriorul UCP şi el trebuie să acţioneze ori de câte ori se face acces la o adresă ce corespunde spaţiului de I/O.

Transferurile *asincrone* de date între două unităţi independente cer să se transmită semnale de control între unităţile ce comunică, pentru a se indica momentul la care datele sunt disponibile. Chiar dacă transferurile asincrone sunt mai lente decât cele sincrone implementarea este adesea necesară dacă vitezele celor două module participante la transfer sunt mult diferite. De exemplu, există diferenţe mari de viteză între UCP şi unele periferice – nu putem face comunicaţie sincronă! În plus, dacă distanţa (lungimea magistralei) între cele două dispozitive este mare varianta asincronă de transfer este cea indicată. De exemplu, dacă semnalul de ceas are o perioadă de 10 ns (frecvenţă de 100 MHz), iar datele se transmit la o distanţă echivalentă cu 60 m, semnalul este primit după 200 ns (20 de impulsuri de ceas), considerând viteza egală cu viteza luminii în vid (3×10^8 m/s).

Pentru descrierea pe scurt a transferului asincron cu două semnale de control vom nota

cele două unităţi care comunică ca "*sursă*" şi "*destinaţie*" a datelor fără a specifica care sunt cei doi interlocutori (microprocesor, memorie, interfaţă de I/O). Unul dintre semnalele de control realizează iniţierea transferului de date, iar cel de-al doilea confirmă recepţia acestora. Procedeul este numit *transfer asincron al datelor, cu confirmare* ("handshaking"). Se realizează astfel un protocol de transmisie la nivel electric, între sursă şi destinaţie. Şi aici, transferul poate fi iniţiat de sursă (cu strobe - marcare date de transmis), sau de către destinaţie (prin strobe - cerere de date). În ambele cazuri interlocutorul va răspunde cu un semnal care indică că datele au fost acceptate, respectiv că datele au fost furnizate (Ready).

Secvenţa evenimentelor pentru transferul iniţiat de sursa datelor este următoarea (figura 1.15):

- sursa plasează datele pe magistrala de date comună
- sursa validează datele prin semnalul de marcare (Strobe)
- destinaţia confirmă că este gata (Ready) pentru transfer
- sursa dezactivează semnalul Strobe, care de obicei produce şi memorarea datelor la destinaţie iar după un timp dezactivează şi datele de pe magistrală
- destinaţia dezactivează semnalul Ready

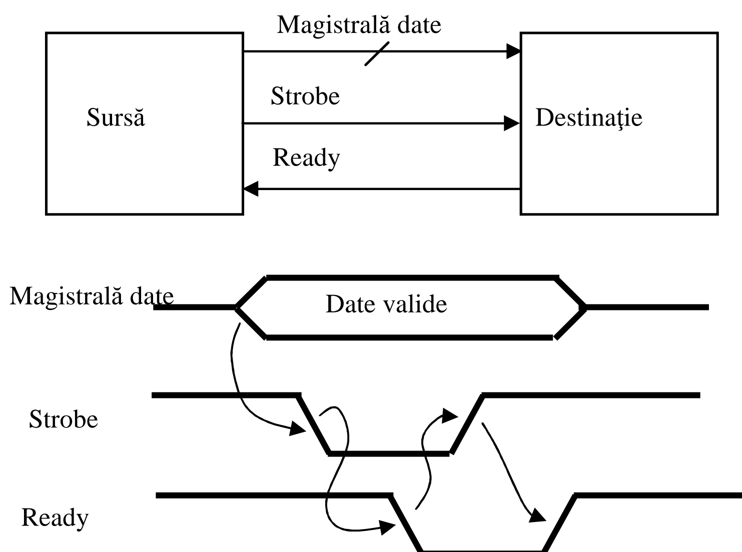


Figura 1.15. Diagrama bloc şi evoluţia semnalelor în timp pentru transfer de tip handshaking iniţiat de sursa datelor.

Se observă că sursa menţine semnalul de strobe activ până când destinaţia este în măsură să răspundă cu semnalul de ready. Duratele semnalelor active se pot prelungi până când cel mai lent dintre corespondenţi poate efectua corect transferul de date.

În cazul când iniţierea transferului se face de către destinaţie formele de undă corespunzătoare controlului transferului sunt prezentate în figura 1.16.

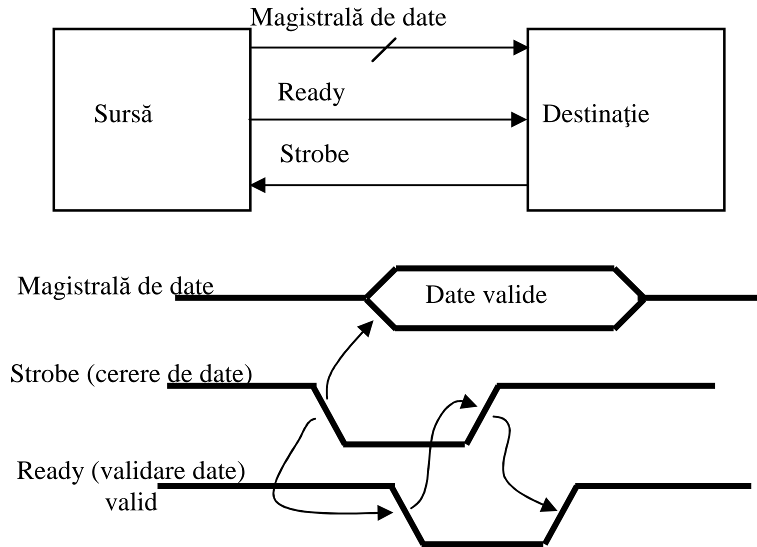


Figura 1.16. Diagrama bloc şi evoluţia semnalelor în timp pentru transfer de tip handshaking iniţiat de destinaţia datelor.

Secvenţa evenimentelor pentru transferul iniţiat de destinaţie este următoarea:

- destinaţia lansează semnalul Strobe, cu semnificaţia "cerere de date"
- sursa răspunde prin punerea datelor pe magistrala comună şi apoi prin activarea semnalului de confirmare (Ready)
- destinaţia citeşte datele şi confirmă acest lucru prin dezactivarea semnalului de strobe
- sursa consideră transferul încheiat şi dezactivează semnalul de ready.

1.12. Exerciţii

1. Ce este un microprocesor şi care sunt componentele sale structurale principale?
2. Ce este un microcontroler şi prin ce se caracterizează această arhitectură de calcul ?
3. Definiţi noţiunea de arhitectura a calculatorului şi explicaţi pe scurt fiecare componenta
4. Care sunt deosebirile esenţiale între arhitecturile von Neuman şi Harvard?
5. Explicaţi caracteristica programelor numită „localizarea referinţelor la memorie". Localizare temporală şi spaţială. Utilizare caracteristică *locality of reference* în administrarea memoriei organizate multi-nivel.
6. Intr-o organizare ierarhizată multi-nivel de memorie, ce reprezintă incluziunea?
7. Intr-o organizare ierarhizată multi-nivel de memorie, ce reprezintă coerenţa datelor?
8. Explicaţi pe scurt în ce constă evenimentul de „word miss" (ratare la apel cuvânt) pentru o organizare multi-nivel de memorie

9. Explicaţi pe scurt în ce constă evenimentul de „word hit” (reuşită la apel cuvânt) pentru o organizare multi-nivel de memorie
10. Explicaţi pe scurt caracteristicile distinctive pentru magistralele sistem: tip (dedicat/multiplexat), sincronizare, arbitrare.
11. Pe o maşină construită M0 cu limbajul propriu L0 se structurează succesiv alte niveluri (maşini virtuale) M1, M2, M3, M4, cu limbajele respective L1, L2, L3, L4. Interpretoarele, identice pentru toate cele patru maşini virtuale, produc pentru fiecare instrucţiune din limbajul L_i, n instrucţiuni în limbajul L_{i-1}. Interpretarea unei instrucţiuni din limbajul L1 consumă timpul T1 = m secunde. Cât timp consumă interpretarea unei instrucţiuni de la maşinile virtuale M2, M3, M4?
12. Explicaţi următorii termeni: (a) translator, (b) interpretor, (c) compilator, (d) maşină virtuală
13. Descrieţi modul de transfer asincron (handshaking)
14. Definiţi următoarele mărimi pentru măsurarea performanţelor procesorului: CPI, MIPS, MFLOPS
15. Pentru o maşină de calcul se poate obţine o creştere de 2 ori a vitezei la execuţia instrucţiunilor în virgulă mobilă. Dacă instrucţiunile cu operanzi în virgulă mobilă reprezintă doar 10% din numărul total de instrucţiuni, care este creşterea în viteză obţinută?
16. Un web server consumă 40% din timp pentru calcule şi 60% aşteptând transferuri de I/O. Se înlocuieşte UCP cu unul nou, mai rapid de 10 ori faţă de vechiul UCP. Care este creşterea globală în performanţă prin utilizarea noului procesor?
17. Dacă o maşină A rulează un program în 2 secunde, iar maşina B rulează acelaşi program în 6 secunde, care din următoarele răspunsuri este adevărat? (a) A este cu 50% mai rapidă decât B; (b) A este cu 300% mai rapidă decât B; (c) A este cu 200% mai rapidă decât B; (d) A este cu 100% mai rapidă decât B
18. Dacă o maşină A rulează un program în 2 secunde, iar maşina B rulează acelaşi program în 6 secunde, care din următoarele răspunsuri este adevărat? (a) A este de 1,5 ori mai rapidă decât B; (b) A este de 2 ori mai rapidă decât B; (c) A este de 2,5 ori mai rapidă decât B; (d) A este de 3 ori mai rapidă decât B

Capitolul 2. Arhitectura setului de instrucţiuni

Conţinut:

2.1. Introducere în arhitectura setului de instrucţiuni

2.2. Formatul instrucţiunilor

2.3. Interdependenţa set de instrucţiuni - organizare internă a microprocesorului

2.4. Scurtă privire comparativă între arhitecturile RISC şi CISC

2.5. Tipuri de instrucţiuni

- 2.5.1. Instrucţiuni aritmetice
- 2.5.2. Instrucţiuni logice
- 2.5.3. Instrucţiuni pentru transferul informaţiei
- 2.5.4. Instrucţiuni pentru deplasarea şi rotirea datelor
- 2.5.5. Instrucţiuni de ramificare (pentru controlul secvenţei de program)
- 2.5.6. Instrucţiuni pentru controlul procesorului
- 2.5.7. Instrucţiuni pentru lucrul cu şiruri

2.6. Moduri de adresare

- 2.6.1. Adresare imediată
- 2.6.2. Adresare directă
- 2.6.3. Adresare indirectă
- 2.6.4. Adresare relativă
- 2.6.5. Adresare indexată

2.7. Exerciţii

2.1. Introducere în arhitectura setului de instrucţiuni

Arhitectura setului de instrucţiuni specifică unui microprocesor cuprinde: setul de instrucţiuni (instrucţiuni specifice la nivel maşină) recunoscute de microprocesor, tipurile de date care pot fi manipulate cu aceste instrucţiuni şi contextul în care aceste instrucţiuni operează.

Arhitectura setului de instrucţiuni (ASI) este o componentă a maşinii de calcul, vizibilă programatorului la nivel de limbaj de asamblare, componentă care realizează interfaţa între software şi hardware (figura 2.1.).

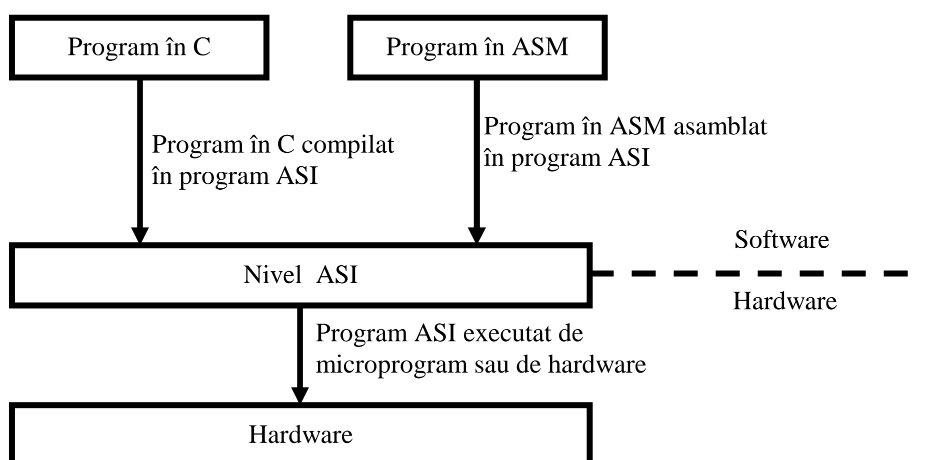


Figura 2.1. Arhitectura setului de instrucţiuni ca interfaţă între hardware şi software (adaptată după [Patterson90])

Pentru a putea comanda componenta hardware a unui calculator trebuie cunoscute instrucţiunile recunoscute de acesta la nivelul maşinii de bază. Instrucţiunile implementate hardware pe o anumită maşină formează setul de instrucţiuni, sau limbajul inclus în ASI. De aceea, în figura 2.1. programele în limbaj de nivel înalt (C în figură) sunt compilate pentru a se genera un program recunoscut de maşină, numit în figură program ASI. Acesta este programul în cod binar recunoscut şi interpretat de o maşină cu algoritm de tranziţie a stărilor interne, numită unitatea de control (UC) a microprocesorului. Algoritmul de tranziţie al stărilor poate fi implementat ca microprogram sau poate fi cablat în hardware. În mod similar programele scrise în limbaj de asamblare (ASM) trebuie translatate cu ajutorul unui program translator numit asamblor în programe executabile, conform ASI specific maşinii. Cunoaşterea ASI pentru un microprocesor este esenţială pentru programatorul care scrie un program de translatare (asamblor, compilator)

pentru acea maşină fizică, sau pentru programatorul la nivel de asamblare.

Setul de instrucţiuni implementat pe un anumit microprocesor poate fi clasificat după mai multe criterii, dintre care menţionăm [Patterson90]:

- *Numărul de adrese de operanzi pe instrucţiune*. Numărul de adrese pe instrucţiune afectează în mod direct complexitatea şi dimensiunile (în număr de biţi) ale unei instrucţiuni. Indiferent de tipul de format al instrucţiunii şi de tipul operaţiilor numărul maxim de operanzi /adrese) este 3, iar numărul minim este 0.
- *Locul de stocare a operanzilor* pentru instrucţiunile executate de UCP. La operaţiile aritmetice şi logice operanzii pot fi imediaţi, se pot găsi în registrele interne ale UCP, în memoria principală, în spaţiul de I/O, sau în memorie de tip stivă. Există arhitecturi care efectuează operaţii aritmetice şi logice doar cu operanzi din registrele interne, iar cu exteriorul permit doar operaţii de încărcare şi stocare a operanzilor (arhitectură load / store). Alte tipuri de arhitecturi permit operanzi doar în memoria stivă (stack architecture – arhitectură pe bază de stivă). Locul de stocare al operanzilor afectează nu numai formatul instrucţiunilor, dar şi modurile de adresare ce trebuie codificate în formatul instrucţiunii.
- *Tipul operaţiilor* efectuate de setul de instrucţiuni. La implementarea unui set de instrucţiuni se urmăreşte optimizarea operaţiilor cele mai frecvent efectuate de o anumită maşină. Tipul instrucţiunilor recunoscute depinde în special de aplicaţiile pentru care este destinat microprocesorul. De exemplu, un microprocesor destinat aplicaţiilor de tip DSP (prelucrare a semnalelor digitale) are instrucţiuni ce optimizează operaţiile de: înmulţire şi adunare repetată şi manipulare de matrice.
- *Tipul datelor prelucrate* de unităţile aritmetice şi logice. Diferitele maşini pot lucra cu numere în virgulă fixă (întregi sau fracţionare) şi numere în virgulă mobilă (numere reale).

Setul de instrucţiuni al unui microprocesor poate avea două caracteristici principale:

1. **regularitate (ortogonalitate)** - această caracteristică impune ca oricare instrucţiune din setul de instrucţiuni să poată opera cu orice tip de date, conţinute în oricare registru sau oricare locaţie de memorie, şi să poată fi adresate prin oricare mod de adresare. Regularitatea setului de instrucţiuni permite implementarea unor programe translatoare (compilatoare) extrem de rapide şi optimizate pentru generarea codului maşină. Pentru a scădea costul microprocesoarelor seturile de instrucţiuni nu sunt complet ortogonale, transferând astfel o parte din costuri către activitatea de programare - scriere şi optimizare a compilatoarelor.
2. **completitudine** - această cerinţă ar impune ca setul de instrucţiuni să cuprindă tot setul de operatori disponibili într-un limbaj. Dar cuprinderea întregului set de operatori, operatori cu frecvenţe foarte diferite de utilizare, ar duce la implementări şi performanţe

departe de cele optime. În consecinţă se introduce o cenzură pentru operatorii cu frecvenţă scăzută de apariţie în programe, în avantajul operatorilor cu frecvenţă ridicată. Uneori se introduce regula (cam generală dar practică): "*Execută rapid operaţiile frecvente şi corect pe cele rare*". În faza de proiectare a unui microprocesor, setul de instrucţiuni se croieşte anticipând spectrul de aplicaţii ale microprocesorului, astfel ca acestea să poată fi uşor exprimate sau programate. Aceasta este faza în care se pot determina operaţiile ce vor fi folosite frecvent.

2.2. Formatul instrucţiunilor

Din punctul de vedere al lungimii instrucţiunilor pot exista două tipuri de ASI:

- arhitecturi la care *lungimea* instrucţiunilor este *fixă*, pentru tot setul recunoscut
- arhitecturi cu *lungime variabilă* a instrucţiunilor recunoscute

În acest capitol, pentru reprezentarea grafică a formatului instrucţiunilor se vor folosi dreptunghiuri ce simbolizează câmpurile binare incluse în instrucţiuni.

La microprocesoarele cu *lungime fixă* a instrucţiunilor (de exemplu arhitecturile RISC[1], dar şi unele arhitecturi CISC[2] cum ar fi cea a bătrânului calculator Felix C256) unitatea de control, care interpretează instrucţiunile şi lansează comenzile pentru execuţia acestora, este mai simplă, dar programele ocupă un spaţiu mai mare în memorie. Formatul de lungime variabilă a instrucţiunilor (în general microprocesoare cu arhitectura CISC) implică o complexitate mai mare a unităţii de control. Lungimea variabilă a instrucţiunilor (lungime mică pentru operaţii simple şi mare pentru cele complexe) permite programe cu cod mai compact (ocupă în general o zonă mai mică de memorie) şi de asemenea permite extinderea uşoară a setului de instrucţiuni, pentru o familie de procesoare cu aceeaşi ASI, pentru că nu mai există limitarea unui număr fix de octeţi.

Instrucţiunile se reprezintă în memoria principală ca octeţi succesivi (unul sau mai mulţi octeţi în funcţie de tipul de microprocesor). Scopul fiecărei instrucţiuni recunoscute de microprocesor este să specifice operaţia ce trebuie executată de hardware, operanzii utilizaţi şi locul unde se stochează rezultatul operaţiei. Formatul unei instrucţiuni include două tipuri principale de informaţii:

- codul operaţiei pe care o comandă instrucţiunea respectivă
- câmpuri de specificatori de adrese pentru indicarea modului de referire la operanzi şi la locul unde se stochează rezultatul. Vom numi aceste câmpuri specificatori de operanzi, pentru că ei indică adresa sau modul de calcul al adresei efective a operanzilor (în figură s-

[1] RISC = Reduced Instruction Set Computer
[2] CISC = Complex Instruction Set Computer

au considerat trei specificatori de adresă de operanzi: OP1, OP2, OP3).

Op-Code	OP1	OP2	OP3

Figura 2.2. Exemplu al formatului general pentru o instrucţiune ASI

Primul câmp binar al instrucţiunii reprezintă codul numeric identificator al instrucţiunii (Op-Code = codul operaţiei efectuate de instrucţiune). Lungimea acestuia este în general diferită în funcţie de complexitatea operaţiei şi eventualele informaţii suplimentare pe care le cuprinde. Aceste informaţii suplimentare pot fi de exemplu: modul de adresare al operanzilor sursă, un indicator de condiţie testat pentru a lua o decizie la execuţia instrucţiunii, un prefix al codului operaţiei care modifică modul de lucru al acelei instrucţiuni. Prefixul poate avea efecte diverse, cum ar fi: repetare de un număr de ori a instrucţiunii cu prefix, indicaţia că se lucrează cu operanzi sau adrese extinse (dimensiune dublă faţă de dimensiunea magistralei de date), identifică o instrucţiune specială destinată coprocesorului aritmetic.

Apoi urmează specificatorii pentru operanzi (modul de găsire al operanzilor se numeşte *mod de adresare* al operanzilor). Ordinea în care sunt dispuşi specificatorii de operanzi în formatul instrucţiunii diferă de la microprocesor la microprocesor. Vom considera, doar pentru exemplificare, un format de instrucţiune cu trei operanzi, ca cel din figura 2.2, şi la care specificatorul OP1 se referă la destinaţie (aceasta poate fi un registru intern, o locaţie de memorie sau porturi de intrare-ieşire), iar OP2 şi OP3 sunt specificatori de operanzi sursă (operanzii putând fi date imediate - constante, sau stocaţi în registre, memorie, ori porturi de I/O).

Instrucţiunea recunoscută de maşină este scrisă deci în cod binar care codifică o acţiune a microprocesorului. Specificatorii de operand ce apar *explicit* în formatul instrucţiunii după codul operaţiei se referă de obicei la adrese din memoria principală, adrese de porturi, operanzi imediaţi (date sau adrese imediate), registre interne de uz general ale UCP. Unele din registrele interne ale UCP au funcţii specifice în cazul anumitor instrucţiuni, iar prin decodificarea codului operaţiei acestor instrucţiuni, registrele respective fiind recunoscute *implicit* ca sursă sau destinaţie de operanzi. De exemplu, în cazul unui microprocesor organizat pe bază de registru acumulator, utilizat ca sursă sau destinaţie exclusivă de multe instrucţiuni adresa acumulatorului nu mai trebuie codată binar în corpul instrucţiunii, adresarea sa fiind implicită.

După modul în care se stochează în memoria principală adresele operanzilor imediaţi din câmpul de specificatori, există două convenţii. Convenţia "Little Endian" (aluzie la şirul indian care are în faţă pe cel mai mic) plasează întotdeauna pe prima poziţie, la adresa cea mai mică, cuvântul cel mai puţin semnificativ al unei adrese multi - cuvânt. Stocarea în memorie se face în ordine, de la cuvântul cel mai puţin semnificativ (adresa mică), către cuvântul cel mai semnificativ stocat la adresa cea mai mare. În figura 2.3. se prezintă un exemplu de stocare în memorie a unui cuvânt

de 32 biţi, în cele două convenţii. Memoria din figură este organizată pe octet, având deci alocată câte o adresă fiecărui cuvânt de 8 biţi. Valorile numerice au fost scrise în hexazecimal.

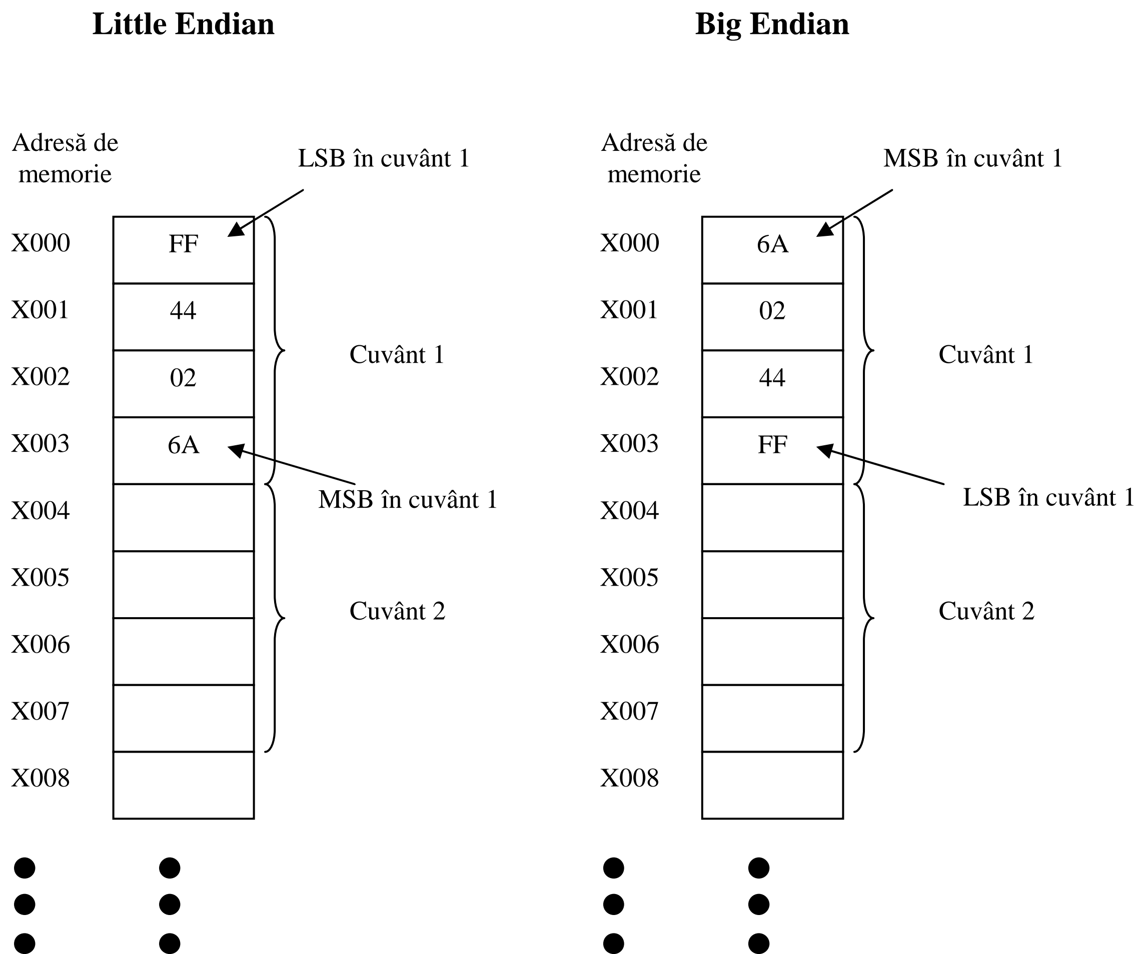


Figura 2.3. Exemplu de stocare a unui cuvânt de 32 de biţi (6A0244FF hex), într-o memorie organizată pe octet, în cele două convenţii. Într-o instrucţiune ce adresează această dată pe 32 de biţi se va specifica doar adresa cea mai mică, de unde începe stocarea datei multi-octet (adresa x0000)

Pentru cealaltă convenţie, numită "Big Endian" (marele End), ordinea este inversă, adică “în faţă”, la adresa cea mai mică se plasează octetul cel mai semnificativ al cuvântului multi-octet.

Legat de convenţia de stocare în memorie a informaţiilor (instrucţiuni, date, adrese) trebuie menţionat aici că la unele microprocesoare accesul la obiecte mai mari decât un octet se poate face rapid doar dacă informaţia este aliniată. Alinierea se referă la faptul că accesul la un obiect de dimensiunea *D* octeţi se face prin specificarea adresei *A* primului octet ocupat, unde adresa A respectă ecuaţia: A *mod* D = 0

Considerând o memorie adresabilă la nivel de octet şi un microprocesor de 32 biţi, alinierea şi nealinierea informaţiilor pe 32 de biţi se exemplifică grafic în figura 2.4. Nealinierea

produce complicaţii la citirea memoriei şi conduce la pierdere de timp. În exemplul din figura 2.4. pentru informaţia aliniată citirea se poate face într-un singur ciclu de acces la memorie. Pentru informaţie ne-aliniată trebuiesc cel puţin două cicluri de acces la memorie. Dar situaţia de ne-aliniere prezentată nu este atât de critică cât ar fi dacă nealinierea s-ar realiza prin stocarea informaţiei pe 32 de biţi la adrese succesive, de exemplu în ordinea: 0B, 0C, 0D şi 0E hex, când citirea ar trebui făcută pentru fiecare octet în parte.

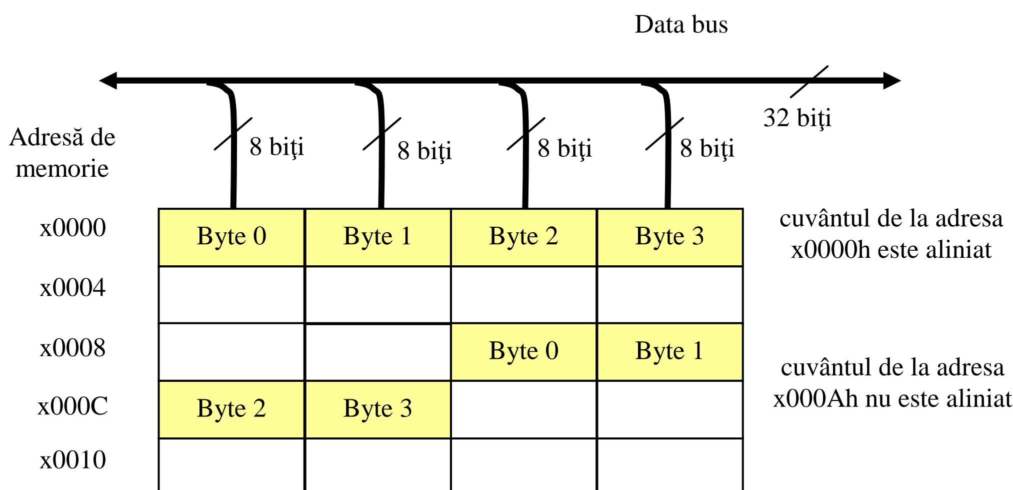


Figura 2.4. Exemplu de adrese de acces la obiecte aliniate şi nealiniate. Cuvântul pe 32 de biţi nealiniat poate fi citit în cel mai bun caz prin două accesări succesive la memorie pentru citirea a câte 2 octeţi.

2.3. Interdependenţa set de instrucţiuni - organizare internă a microprocesorului

O sursă de controverse începând de la prima generaţie de calculatoare a fost: câte *adrese explicite* pentru operanzi să includă o instrucţiune? Cu cât sunt mai puţine adrese cu atât e mai scurtă instrucţiunea. Dar limitând numărul de adrese se limitează domeniul de funcţii pe care fiecare instrucţiune le poate realiza. În general, mai puţine adrese înseamnă mai multe instrucţiuni primitive şi deci programe mai lungi pentru o anumită sarcină de rezolvat.

În timp ce necesităţile de stocare în memorie pentru instrucţiuni mai scurte şi programe mai lungi tind să se echilibreze cu instrucţiunile lungi şi programe scurte, programe mai lungi cer un timp mai lung de execuţie. Pe de alta parte, instrucţiunile lungi, cu adrese multiple, cer de obicei circuite mai complexe pentru decodare şi prelucrare. Uneori microprocesoarele sunt clasificate după numărul maxim de adrese de memorie principală în câmpurile instrucţiunilor lor. Microprocesoarele conţinând un număr diferit de adrese au în general instrucţiuni de lungimi diferite. Instrucţiunile cu doi operanzi necesită cel mult trei adrese. De exemplu, pentru o operaţie de adunare, simbolizată prin mnemonica “add” pot fi implementate instrucţiuni cu

minim 0 şi maxim 3 operanzi, aşa cum se observă în tabelul 2.1.

Numărul de câmpuri de adresă în formatul instrucţiunii unui microprocesor depinde organizarea internă a registrelor sale, dar în acelaşi timp influenţează modul în care este proiectat microprocesorul ca organizare, din punctul de vedere al registrelor interne. Numărul de adrese poate fi diferit, de la instrucţiune la instrucţiune la acelaşi microprocesor. De exemplu instrucţiunea de scădere poate avea trei adrese de operanzi (doi sursă şi unul destinaţie), iar instrucţiunea de complementare poate avea un singur operand (dacă registrul sursă este şi destinaţie).

Tabelul 2.1.

Număr adrese	Format instrucţiune	Comentariu
3	**add z,x,y**	; z ← x+y , unde x,y,z, sunt locaţii de ;memorie, nume de registre, sau date imediate
2	**add x,y**	; x ←x+y sau Ac ← x+y , în funcţie de ; microprocesor (Ac = registru acumulator)
1	**add x**	; Ac ← Ac+x
0	**add**	; adună primele două articole din vârful stivei şi ; stochează rezultatul în noul vârf al stivei

Conform legăturii dintre organizarea UCP şi numărul de adrese pe instrucţiune, vom putea încadra cele mai multe microprocesoare într-una din următoarele trei categorii:

1. *organizare pe bază de registru acumulator (Ac)* la care pentru majoritatea operaţiilor unul dintre operanzi se găseşte implicit în acumulator. De asemenea rezultatul operaţiei se "acumulează" în acest registru cu funcţie specială.
2. *organizare pe bază de registre de utilizare generala (RG)*. La acest tip de organizare operanzii sunt explicit localizaţi fie în registrele interne (fără restricţii), fie în registrele din locaţiile memoriei principale.
3. *organizare de tip stivă (ST)* la care operanzii se găsesc implicit în vârful stivei şi tot aici se stochează rezultatul. Există unele procesoare de tip stivă care folosesc două stive: una pentru operaţii asupra datelor şi alta pentru operaţii asupra adreselor.

Fiecare dintre cele trei tipuri de organizări au avantaje şi dezavantaje. Astfel organizarea pe bază de acumulator conduce la instrucţiuni scurte, pentru că acumulatorul este referit ca registru implicit la multe dintre instrucţiunile maşinii. Conduce şi la o densitate relativ mare a codului (programelor) stocate în memorie, dar acumulatorul constituie un puternic factor de limitare al vitezei de lucru al microprocesorului. Maşinile pe bază de registre de uz general

prezintă avantaje din punctul de vedere al uşurinţei generării codului de către programe compilatoare. Dacă nu ar fi existat aceste registre, foarte multe din rezultatele intermediare ar fi trebuit stocate în memoria externă microprocesorului ceea ce ar fi micşorat viteza de prelucrare. Dezavantajul arhitecturilor de tip RG este că fiecare registru trebuie adresat de instrucţiunea ce-l utilizează, astfel că lungimea instrucţiunilor este relativ mare. Ultimul tip de organizare, pe bază de stivă, permite o evaluare extrem de uşoară a operaţiilor pe baza notaţiei inverse (notaţia "poloneză" - vezi exemplul de mai jos). Aceasta conduce şi la instrucţiuni cu lungime foarte redusă. Din păcate o stivă este mai dificil de adresat în mod aleator, iar codul rezultat este relativ ineficient. Stiva introduce şi o limitare în viteză şi de aceea acest tip de organizare este rar implementat la procesoare de uz general.

Tabelul 2.2. Exemple de instrucţiuni

Tip arhitectură	**Instrucţiuni**	**Comentarii**
Ac	add x	;Ac ← Ac + x sau Ac ← Ac + @x ;adună la acumulator conţinutul registrului x sau al ;memoriei de la adresa dată de x

RG	add r1,r2,r3
	add r1,r2
	mov r1,r2
	add r1,x
		; r1 ← r2 + r3
		; r1 ← r1 + r2 sau r2 ← r1 + r2
		; r1 ← r2 sau invers
		; r1 ← r1 + @x

ST	push x add	;trimite cuvântul de la adresa x în vârful stivei, ;VS ← @x ;aduce primele 2 articole din stivă, le adună şi ; stochează rezultatul în stivă

În tabelul 2.2. s-au prezentat câteva exemple de instrucţiuni pentru cele trei tipuri de arhitecturi. Trebuie menţionat că multe microprocesoare pot fi încadrate în cele 3 categorii de mai sus, dar cele mai multe combină caracteristici din mai multe moduri de organizare.

Pentru a ilustra influenţa numărului de adrese în programele executate de microprocesor vom evalua egalitatea:

$$x = a \times (b + c) \tag{2.1}$$

utilizând exemple de instrucţiuni cu 0, 1, 2, şi 3 adrese. Vom folosi simbolurile (mnemonicele):

- add, mul, - pentru operaţiile aritmetice elementare de adunare şi înmulţire
- load, store - transfer de la/la memorie spre/dinspre acumulator
- push, pop - înscriere respectiv extragere din stivă.

- r# - registre interne ale UCP, unde # este un indice
- @b - operandul de la adresa conţinută în registrul b (registru intern sau registru de memorie cu adresa b)

Presupunem că operanzii a, b şi c din relaţia (2.1) sunt stocaţi în memorie, în locaţiile de adrese a, b, c, iar rezultatul se va introduce la adresa x în memorie.

a) pentru arhitecturi pe baza de registre de uz general:

- *instrucţiuni cu 3 adrese care pot efectua operaţii aritmetice sau logice cu operanzi din memorie (CISC):*

```
add    r1,b,c        ;r1 ← @b + @c
mul    x,r1,a        ;@x ← r1 * @a
```

Se observă că acest format poate utiliza fiecare câmp de adresă operand pentru a specifica un operand dintr-un registru sau din memorie. Deşi programul este foarte scurt, el conţine instrucţiuni cu format de lungime mare şi se desfăşoară relativ lent. Astfel la prima instrucţiune se fac două accesări la memorie pentru citirea operanzilor b şi c. La fel pentru a doua instrucţiune sunt necesare două accesări succesive la memorie (operand şi rezultat).

- *instrucţiuni cu 3 adrese (RISC);*

```
load   r1,a          ; r1 ← @a
load   r2,b          ; r2 ← @b
load   r3,c          ;
add    r2,r2,r3      ; r2 ← (r2 + r3)
mul    r1,r1,r2      ; r1 ← (r1 * r2)
store  x,r1          ; @x ← r1
```

La acest tip de arhitecturi se folosesc doar instrucţiuni de *load (încarcă) şi store (stochează)* când se comunică cu memoria, iar toate celelalte instrucţiuni se referă la operanzi din registrele UCP (deci prelucrare foarte rapidă). Viteza mai mare rezultă şi din faptul că datele din memorie au fost pre-încărcate în memoria tampon de date locală (cache sau coadă / FIFO[3])

[3] FIFO - First In First Out - primul intrat, primul ieşit.

- *instrucţiuni cu 2 adrese (CISC)*

```
load   r1,b     ;r1 ← @b
add    r1,c     ;r1 ← r1 + @c
mul    r1,a     ;r1 ← r1 * @a
store  x,r1     ;@x ← r1
```

b) pentru arhitecturi pe baza de acumulator (instrucţiuni cu o adresa, acumulatorul fiind adresat implicit):

```
load   b        ; ac ← @b
add    c        ; ac ← ac + @c
store  t        ; @t ← ac
load   a        ; ac ← @a
mul    t        ; ac ← ac * @t
store  x        ; @x ← ac
```

unde s-a notat cu "t" adresa unei locaţii de memorie folosita pentru stocarea temporara a rezultatelor intermediare

c) pentru arhitecturi pe bază de stivă (instrucţiuni cu zero adrese)

```
push  a    ;vs ← @a, fie "adr" adresa curentă a vârfului stivei (vs)
           ; iar stiva creşte spre adrese mici
push  b    ;vs ← @b, la adr-1
push  c    ;vs ← @c, la adr-2
add        ;vs ← (b + c), stocat la adr-1          zero adrese
mul        ;vs ← a*(b+c), stocat la adr            zero adrese
pop   x    ;@x ← vs, vs pointează la adr+1
```

Pentru a evalua expresiile aritmetice într-un calculator stivă, e necesar să se convertească expresia în notaţia poloneză (inversă). Ideea a pornit de la matematicianul polonez Lukasievics J, care a notat o expresie de forma "a + b" ca "ab+".
După acelaşi principiu putem exemplifica notaţia poloneză şi pentru alte ecuaţii:

x = a* (b + c) = abc+*.
x = (a+b)*(c+d) = ab+cd+*
x = (a+b)*(c+d)*(e+f) = ab+cd+ef+**

După cum se vede, notaţia poloneză are avantajul că nu foloseşte paranteze. Ba mai mult, pornind de la stânga la dreapta se indică toate operaţiile ce trebuiesc efectuate, în ordine, la o organizare de tip stivă.

2.4. Scurtă privire comparativă între arhitecturile RISC şi CISC

Multe calculatoare au seturi de instrucţiuni ce includ mai mult de 100 - 200 instrucţiuni. Ele folosesc o varietate de tipuri de date şi un mare număr de moduri de adresare. Tendinţa aceasta de a mări numărul de instrucţiuni a fost influenţată de mai mulţi factori, dintre care menţionăm:

- perfecţionarea unor modele de procesoare existente anterior, pentru a pune la dispoziţia utilizatorilor (programelor utilizator) cât mai multe funcţii cablate (recunoscute în setul de instrucţiuni)
- adăugarea de instrucţiuni care să faciliteze translatarea din limbajele de nivel înalt în programe cod executabil (limbaj maşină)
- tendinţa de a deplasa cât mai multe funcţii de la implementarea programată (software) către cea cablată (hardware), în scopul obţinerii unor performanţe de viteză cât mai mari.

Ideea simplificării setului de instrucţiuni, în scopul măririi performanţelor microprocesorului, provine atât din proiectul calculatorului CDC 6600, cât şi din proiectele realizate la universităţile americane din Berkeley (RISC I, RISC II şi SOAR) şi Stanford (proiectul MIPS). Proiectele RISC (Reduced Instruction Set Computer - Calculator cu set redus de instrucţiuni) au urmărit ca instrucţiunile microprocesorului să fie de aceeaşi lungime, instrucţiunile să se *execute într-o singură perioadă de ceas* (cu ajutorul tehnicii de tip conductă de execuţie - pipeline), iar unitatea de control să fie implementată cablat, pentru a reduce complexitatea circuitului integrat şi pentru a creşte viteza. La RISC se urmăreşte de asemenea ca accesările la memorie (consumatoare de timp) să se efectueze doar pentru operaţiile de încărcare şi stocare (arhitectura fiind numită în consecinţă: "*load/store*"), iar celelalte operaţii să se efectueze cu operanzi stocaţi în registrele interne ale UCP. Unele din proiectele de arhitecturi RISC folosesc un set mare de ferestre de registre (numit şi *stack cache* - memorie stivă de registre) pentru a accelera operaţiile de apel al procedurilor / subrutinelor. Cu toate aceste specificaţii, denumirea RISC nu conduce şi la nişte criterii stricte de proiectare, ea fiind doar o descriere a cerinţelor generale impuse microprocesorului. De aceea multe din maşinile anterioare (numite de atunci arhitecturi CISC - Complex Instruction Set Computer - calculator cu set complex de instrucţiuni) au împrumutat aceste idei de proiectare (pipeline, execuţia unei instrucţiuni pe ciclu de ceas,

ferestre de registre, etc.) dar ele nu pot fi numite RISC-uri (MC68040, 80486, Pentium, H16, etc.). Concepţia RISC porneşte de la faptul că un circuit mai simplu pentru microprocesor, poate lucra la frecvenţe de ceas mai mari, iar setul mai simplu de instrucţiuni este mai potrivit pentru optimizări ale codului de către un compilator.

Arhitectura numită CISC, se referă la un set complex de instrucţiuni; aceasta însă, doar prin comparaţie cu RISC, căci nu există un set de caracteristici generale, asa cum s-au definit la RISC. Ideea care s-a manifestat în dezvoltarea acestor maşini, de a introduce structuri hardware suplimentare pentru creşterea vitezei, nu este greşită ca principiu. Proiectarea unui set de instrucţiuni pentru un microprocesor trebuie să ţină seama nu numai de construcţia limbajului maşină ci şi de cerinţele impuse de utilizarea limbajelor de programare de nivel înalt. Translatarea se face cu un program compilator, deci scopul final ar fi pentru arhitectura CISC să permită folosirea unei singure instrucţiuni maşină pentru fiecare instrucţiune din programul scris în limbaj de nivel înalt. Aceasta ar conduce la simplificarea compilatorul, căci sarcina acestuia este mult uşurată dacă există o instrucţiune maşină care implementează direct o instrucţiune de limbaj înalt. Dar asta presupune realizarea câte unei arhitecturi pentru fiecare tip de limbaj de programare. În plus, s-a constatat că, de cele mai multe ori, instrucţiunile complexe sunt rar utilizate de compilatoare. Alt dezavantaj al CISC constă în faptul că este dificil de mărit frecvenţa de ceas pentru un circuit complex.

Rezumând, putem enumera câteva din elementele caracteristice pentru fiecare din cele doua arhitecturi.

Rezumat al caracteristicilor definitorii pentru *maşinile RISC*:

1. Acces la memorie limitat, doar prin instrucţiuni de *încărcare (load) şi stocare (store)*;
2. Format de *lungime fixă* pentru instrucţiuni, deci uşor de decodificat; caracteristică care contribuie la simplificarea structurii unităţii de control;
3. Structură *simplă a unităţii de control* implementată sub formă cablată, deci cu viteză mare de funcţionare;
4. Relativ *puţine tipuri de instrucţiuni* (tipic sub 100 de instrucţiuni) şi puţine moduri de adresare (din nou această caracteristică contribuie şi la simplificarea structurii unităţii de control);
5. Execuţia instrucţiunilor într-un singur ciclu prin tehnici pipeline (conductă / bandă de execuţie); se face prin suprapunerea diferitelor faze de execuţie (fetch, decodificare, aducere operanzi etc.) ale mai multor instrucţiuni succesive. Tehnica de tip pipeline este utilizată şi la arhitecturile CISC, dar la RISC tehnica este mai eficientă şi mai uşor de implementat, datorită lungimii constante a instrucţiunilor;
6. Un număr relativ mare de registre în interiorul UCP;
7. Utilizarea compilatoarelor optimizatoare - pentru a optimiza performanţele codului obiect, în special pentru înlăturarea conflictelor în conductele de execuţie.

La *CISC* câteva din caracteristicile care le deosebesc de RISC-uri sunt:

1. Multe instrucţiuni aritmetice şi logice care prelucrează operanzi din memorie;
2. Format de *lungime variabilă* pentru instrucţiuni;
3. Unitate de control microprogramată (micro-codată), avantajoasă din punctul de vedere al flexibilităţii implementării, dar de viteză redusă;
4. Set complex (extins) de instrucţiuni şi o mare varietate de moduri de adresare;
5. Un număr relativ mic de registre în interiorul UCP;
6. Exista instrucţiuni complexe de mare eficienţă, dar uneori acestea sunt rar utilizate de compilatoare.

Aşa cum s-a arătat mai sus, arhitecturile RISC restricţionează numărul de instrucţiuni care accesează direct memoria principală. Cele mai multe instrucţiuni ale RISC presupun doar operaţii între registrele interne UCP. Pentru că instrucţiunile complexe nu există în setul de instrucţiuni, dacă este nevoie de ele, acestea se implementează prin rutine cu ajutorul instrucţiunilor existente. În final, într-un program executabil vor fi mai multe instrucţiuni decât la CISC, dar execuţia pe ansamblu poate fi mai rapidă. Totuşi, dacă într-un program, frecvenţa operaţiilor complexe este mare, este foarte posibil ca performanţele unei maşini CISC să fie mai bune (execuţie mai rapidă).

2.5. Tipuri de instrucţiuni

Prezentarea tipurilor de instrucţiuni se va face cu exemplificări din setul de instrucţiuni specifice procesoarelor Intel 80x86. Pentru toate instrucţiunile prezentate ca exemplu, dacă acestea cuprind doi specificatori de operand, întotdeauna primul specificator se referă la destinaţie (D - rezultat) şi implicit la una din sursele de operand, iar al doilea la sursă (S).

2.5.1. Instrucţiuni aritmetice

Asigură prelucrarea unor operanzi aflaţi în memorie sau în registrele generale interne UCP, având la bază operaţii de o complexitate foarte diferită la diferitele microprocesoare. Un set tipic de instrucţiuni aritmetice este rezumat în continuare:

Tip instrucţiuni aritmetice	Exemple I8086
• adunare	ADD D,S
• adunare cu considerarea transportului	ADC D,S
• scădere	SUB D,S
• scădere cu considerarea împrumutului	SBB D,S
• incrementare	INC D
• decrementare	DEC D
• înmulţire (fără semn)	MUL S *
• înmulţire (cu semn)	IMUL S *
• împărţire (fără semn)	DIV S *
• împărţire (cu semn)	IDIV S *
• realizare cod complementar (C2)	NEG D
• comparare	CMP D,S
• ajustare zecimală	DAA (După adunare în NBCD)
	DAS (După scădere în NBCD)

Notă: * = deîmpărţitul sau deînmulţitul se găsesc implicit în registrele acumulator şi eventual DX. Din această cauză la I8086 există un singur operand. La procesoare Intel pe 32 de biţi (de la I80386), operaţia de înmulţire cu semn a întregilor poate avea şi forma cu doi sau trei operanzi, caz în care se pot folosi şi alte registre generale, decât AX şi DX.

În urma efectuării operaţiilor aritmetice este *influenţată valoarea indicatorilor de condiţii*,

în funcţie de rezultat. Instrucţiunile de comparaţie a două valori binare ar putea fi asociate instrucţiunilor logice dacă comparaţia s-ar face prin instrucţiuni logice; comparaţia este inclusă însă la instrucţiuni aritmetice pentru că se efectuează prin scădere şi testarea rezultatului (de fapt testarea indicatorilor de condiţii). La compararea prin scădere conţinutul registrelor sursă nu este afectat, iar operaţia se face fără memorarea rezultatului, doar prin setarea indicatorilor de condiţii.

Tot în cadrul instrucţiunilor aritmetice pot fi incluse şi operaţii aici pot fi incluse instrucţiuni de conversie a dimensiuni (de exemplu la I 8086 din 8 în 16 biţi (CBW), din 16 în 32 (CWD) la I80386: CWDE (16→32) şi CDQ (32→64).

2.5.2. Instrucţiuni logice

Acestea efectuează operaţii logice cu operanzii specificaţi de instrucţiune. Deşi, în principiu, ar fi suficiente operaţiile operatorilor compleţi NAND[4] şi / sau NOR, setul de instrucţiuni al unui microprocesor include ca instrucţiuni logice cel puţin operaţiile logice de tip ŞI, SAU, NU, SAU-Exclusiv. Aceste operaţii se efectuează bit cu bit între operanzi, cu excepţia complementării (NU) care are un singur operand, pentru care se complementează fiecare bit.

În urma efectuării operaţiilor logice este *influenţată valoarea indicatorilor de condiţii*, în funcţie de rezultat. Un set tipic de instrucţiuni logice este rezumat în continuare:

Tip instrucţiuni logice	**Exemple I8086**
• ŞI logic	AND D,S
• SAU logic	OR D,S
• SAU-Exclusiv	XOR D,S
• complement (NU)	NOT D
• test logic	TEST D,S *

Nota: * = se efectuează ŞI logic între sursă şi destinaţie, rezultatul nu se stochează nici-unde, dar se poziţionează indicatorii de condiţii.

Tot în cadrul instrucţiunilor logice sunt incluse şi instrucţiunile de prelucrare şi test logic a biţilor unui cuvânt. De exemplu, la I80386 există instrucţiunile de examinare (scan) a biţilor unui octet / multi-octet (BSF - începând cu LSbit, BFR - începând cu MSbit), sau de testare şi prelucrare

[4] NAND / NOR = ŞI-NU / SAU-NU

a biţilor unui octet / multi-octet, cu poziţionarea indicatorului CF (BT, BTC, BTR, BTS).

Operaţiile de prelucrare pe bit sunt importante în special pentru modificarea registrelor de control, sau la microcontrollere integrate pentru scrierea selectivă la anumite ieşiri ale portului paralel.

2.5.3. Instrucţiuni pentru transferul informaţiei

Aceste instrucţiuni se folosesc pentru transferul (*copierea*) informaţiei între registre, între registre şi memorie, sau între registre şi porturi de I/O. În general, sunt operaţii de copiere a informaţiei din sursă în destinaţie. Aceste instrucţiuni nu afectează de obicei valorile indicatorilor de condiţii.

Pot fi clasificate in:

- Instrucţiuni de transfer de uz general
- Instrucţiuni de transfer cu stiva
- Instrucţiuni pentru transferul adreselor (pointeri)
- Instrucţiuni de transfer I/O.

Pentru microprocesoarele care au porturile de I/O mapate în spaţiul de adrese de memorie nu există instrucţiuni specializate de transfer de I/O. De obicei, pentru transferul I/O există multe instrucţiuni de I/O specifice fiecărui microprocesor, care se referă la transferul indirect în registre dedicate, transferul repetat de la o adresă de port etc.

Exemplificare pentru instrucţiunile de transfer:

Tip instrucţiuni de transfer	**Exemple I8086**
• transfer de uz general	MOV D,S ;D←S
	XCHG D,S ;D↔S
• transfer cu stiva	PUSH S; înscriere în stivă
	POP D ; extragere din stivă
	PUSHF ; scriere indicatori în stivă
	POPF ; extragere indicatori din stivă
• transferul adreselor	LEA r16,adr ;(r16)←adr
	LDS r16,adr ;r16Low ← [DS:adr]
	;r16High ←[DS:(adr+1)]
	;DSLow ← [DS:(adr+2)]
	;DSHigh ← [DS:(adr+3)]
• transfer de I/O	IN AL,adr ; AL ← adr (port)
	IN AX,adr ; AL ← adr (port), 16 biţi
	IN AL,DX ; AL ← (DX)
	OUT adr,AL
	OUT DX,AL

2.5.4. Instrucţiuni pentru deplasarea şi rotirea datelor

Instrucţiunile propriu-zise de deplasare pot fi asociate instrucţiunilor logice, în cazul în care deplasarea se efectuează în scopul izolării unor biţi dintr-un cuvânt binar (pentru a testa sau prelucra acei biţi), sau pot fi asociate instrucţiunilor aritmetice (înmulţire / împărţire cu puteri ale lui 2). În cazul deplasărilor aritmetice se conservă bitul de semn al operandului deplasat.

Operaţiile de deplasare pot fi:

- ⇒ deplasări închise (rotaţii stânga / dreapta)
- ⇒ deplasări deschise (stânga / dreapta)

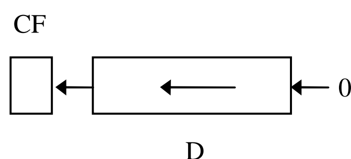
La toate operaţiile de deplasare, se influenţează şi valoarea indicatorului de transport (CF - Carry Flag), care se încarcă cu ultima valoare de bit ieşită din registrul folosit pentru deplasarea operandului.

Deplasările închise (rotirile) sunt doar de tip logic. Rotaţiile se pot efectua pe n biţi (n fiind dimensiunea registrelor ce conţin datele) sau pe n+1 biţi (rotaţie prin bitul de Carry). Deplasările deschise pot fi de tip logic sau de tip aritmetic. Pentru toate exemplele care se dau mai jos (I8086), al doilea operand poate fi 1 sau CL; dacă al doilea operand este 1 se indică deplasare cu o poziţie binară, iar dacă acest operand este CL, rezultă o deplasare cu p poziţii binare, p fiind valoarea conţinută în registrul CL.

Exemple (Intel x86):

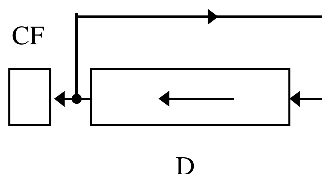
- deplasare aritmetica / logică la stânga

SAL D,1
SHL D,1



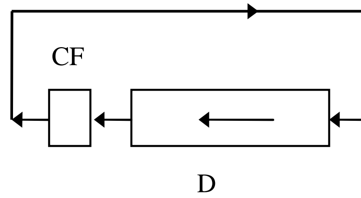
- rotire la stânga

ROL D,1



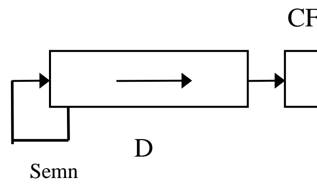
- rotire la stânga prin CF (rotire pe n+1 biţi, dacă D are n biţi)

RCL D,1



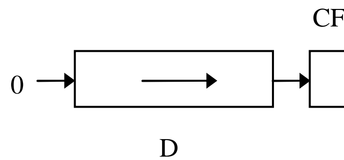
- deplasare aritmetică la dreapta (se păstrează valoarea bitului de semn)

SAR D,1



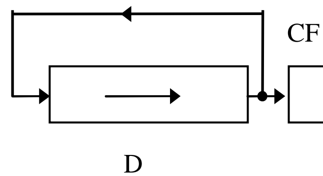
- deplasare logică dreapta

SHR D,1



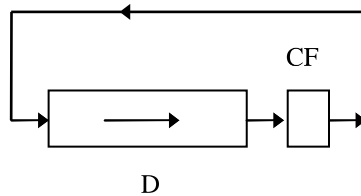
- rotire dreapta

ROR D,1



- rotire dreapta prin CF

RCR D,1



Începând cu microprocesorul I80386 au apărut instrucţiuni de deplasare suplimentare (de exemplu SHLD, SHRD), la care se folosesc 3 operanzi, primul fiind destinaţia, următorul sursa, iar ultimul contorul care indică numărul de biţi de deplasare.

2.5.5. Instrucţiuni de ramificare (pentru controlul secvenţei de program)

Aceste instrucţiuni determină modificarea secvenţei normale, liniare, de execuţie a unui program, în funcţie de condiţiile în care s-a terminat execuţia unei instrucţiuni precedente. Se asigură astfel ramificaţiile în program şi implementarea diferitelor structuri de control specificate de un anumit algoritm. Aceste instrucţiuni nu afectează starea indicatorilor de condiţii, dar unele din instrucţiunile de salt sunt influenţate de valoarea conţinuta în indicatori.

Instrucţiunile de ramificare (salt) pot fi:

- Necondiţionate: La execuţia acestor instrucţiuni saltul se efectuează întotdeauna. Exemple de salturi necondiţionate sunt instrucţiunile de apel de procedură (CALL op), salturi absolute (JMP op), revenire din procedură / return (RET), instrucţiuni de apel a unor funcţii sistem (INT #).
- Condiţionate: Execuţia acestor instrucţiuni produce salt doar dacă este îndeplinită o anumită condiţie. Condiţia se poate referi la un indicator de condiţii sau la conţinutul unor registre. Tipice sunt aici salturile condiţionale (Jcc - Jump if condition = TRUE), apeluri condiţionale de proceduri (CALL if condition = TRUE) sau "repetă până când condition = TRUE".

Exemplificări pentru I8086:

JMP op

Este operaţia de salt necondiţionat, prin care contorul de program (IP - pointer de instrucţiuni) este înlocuit de offset-ul etichetei ţintă în toate salturile inter-segment; acelaşi lucru şi pentru salturile indirecte în cadrul aceluiaşi segment.

CALL op

Instrucţiunile de chemare a procedurilor lucrează similar cu JMP (salt intersegment sau extra-segment), cu excepţia faptului că la execuţia instrucţiunii CALL se salvează automat în stivă adresa de revenire din procedură.

RET

Instrucţiunile de tip RET produc revenirea dintr-o procedură, cu refacerea conţinutului contorului de program şi eventuala incrementare suplimentară (RET n) a indicatorului de stiva (registrul SP).

Jcc op

Instrucţiunile de salt condiţional se referă la instrucţiuni ce se execută doar dacă o anumită condiţie este îndeplinită. Îndeplinirea acestei condiţii se verifică prin testarea indicatorilor de condiţii. Vom încerca o prezentare sistematizată a salturilor condiţionale pe baza

tabelului 2.3. Fiecare mnemonică se referă la iniţialele cuvintelor următoare, ce indică condiţia în limba engleză: Above (peste, mai mare), Below (sub, mai mic), Equal (egal), Not (nu), Greater (mai mare), Less (mai mic), Carry (transport), Zero, Overflow (depăşire de capacitate), Parity (PEven - paritate pară, POdd - paritate impară), Sign (semn). Din tabel se observă că se folosesc condiţii diferite şi litere diferite pentru a exprima relaţia mai mare, sau mai mic, pentru numerele cu semn, respectiv pentru cele fără semn.
(*Exemplu de citire: JNBE = jump if not below nor equal, salt (J) daca nu (N) e mai mic (B) sau egal (E)*).
Tabelul 2.3. indică mnemonicele instrucţiunilor pentru testarea condiţiei din prima coloană, la scăderea a două numere A - B.

Tabelul 2.3. Explicativă la salturile condiţionate pentru I8086

	Condiţia de salt pentru:	
	numere fără semn	numere cu semn
A > B	JA/JNBE (CF = 0 şi ZF = 0)	JG/JNLE (ZF = 0 şi SF = OF)
A ≥ B	JAE/JNB/JNC (CF = 0)	JGE/JNL (SF = OF)
A = B	JE/JZ (ZF = 1)	
A ≤ B	JBE/JNA (CF = 1 sau ZF = 1)	JLE/JNG (dacă SF != OF sau ZF = 1)
A < B	JB/JNAE/JC (CF = 1)	JL/JNGE (SF != OF)

Acolo unde în tabel apar doua tipuri de mnemonice separate prin semnul "/", formele sunt echivalente, oricare formă conducând la aceeaşi instrucţiune binară.

In afara de salturile descrise în tabel, mai există instrucţiunile: JCXZ (salt dacă CX=0), JNZ/JNE, JO, JNO, JP/JPE, JNP/JPO, JS, JNS (ultimele se referă la testarea indicatorilor de condiţii, condiţia fiind valoarea 1 a indicatorului, sau valoarea zero daca în fata numelui de indicator apare N(ot).

2.5.6. Instrucţiuni pentru controlul microprocesorului

Sunt instrucţiuni care controlează anumite funcţii ale microprocesorului, ce acţionează fie prin intermediul unor indicatori de control (sau registre de control), fie prin introducerea unor stări sau semnale necesare pentru sincronizarea cu evenimentele externe.

Exemple (I8086):

```
CMC        ;complementarea valorii indicatorului CF
CLC        ;poziţionarea pe 0 a indicatorului CF
STC        ;poziţionarea pe 0 a indicatorului CF
NOP        ;Nici o operaţie, dar consumă 3 perioade de ceas.
```

CLD	;poziţionarea pe 0 a indicatorului DF
STD	;poziţionarea pe 1 a indicatorului DF
CLI	;poziţionarea pe 0 a indicatorului IF, dezactivare întreruperi mascabile
STI	;poziţionarea pe 1 a indicatorului IF
HLT	;Oprire microprocesor până la RESET, NMI, sau INT (dacă sunt activate)
WAIT	;aşteptare până când vine semnalul exterior test=0
ESC	;operaţie destinată coprocesorului
LOCK	;prefix ce activează semnalul /lock, astfel că microprocesorul anunţă
	;că nu va răspunde la o cerere de cedare a controlului magistralelor.

2.5.7. Instrucţiuni pentru lucrul cu şiruri

În afară de tipurile de bază amintite mai sus, există şi posibilitatea efectuării unor operaţii de transfer, sau operaţii aritmetice şi logice cu şiruri de date (cu informaţii aflate în zone continue de memorie). Operaţiile pe şiruri pot fi efectuate individual, pentru fiecare cuvânt din şir, sau automat - cu repetare, numărul de repetări al instrucţiunii fiind dictat de conţinutul unui registru contor. Operaţiile tipic efectuate sunt:

- transferul unui şir din zonă sursa în zonă destinaţie
- comparare între două şiruri
- căutarea unei valori într-un şir
- încărcarea acumulatorului cu elementele unui şir.
- citirea unui şir de la un port de intrare
- scrierea unui şir la un port de ieşire

Exemple (I8086):

MOVSB(W)	;transfer pe 8 (16) biţi [DS:SI]→[ES:DI]
	;SI←SI+1; DI←DI+1 (decrementare pentru DF=1)
CMPSB(W)	;comparare pe 8(16) biţi [DS:SI] cu [ES:DI]
	;SI←SI+1; DI←DI+1 (decrementare pentru DF=1)
SCASB(W)	;comparare pe 8(16) biţi intre AL(AX) şi [ES:DI]
	;DI←DI+1 (decrementare pentru DF=1)
LODSB(W)	;se încarcă AL(AX) de la [DS:SI]
	;SI←SI+1 (decrementare pentru DF=1)
STOSB(W)	;se stochează AL(AX) la [ES:DI]
	;DI←DI+1 (decrementare pentru DF=1)
REP	;prefix de repetare. Contorul este CX

2.6. Moduri de adresare

Pe măsura evoluţiei arhitecturii calculatoarelor, s-a dezvoltat şi o mare varietate de moduri de adresare a instrucţiunilor şi datelor, scopurile principale fiind:

- optimizarea spaţiului de memorie ocupat de instrucţiuni;
- facilităţi sporite privind prelucrarea unor structuri de date de tip vector (matrice):
- facilităţi privind implementarea în programe a unor structuri de control complexe.

Noţiunea de *mod de adresare* se referă la modul în care în formatul instrucţiunii se specifică adresa unui operand de prelucrat, adresa de stocare a rezultatului operaţiei, sau adresa următoarei instrucţiuni. Prin decodificarea instrucţiunii microprocesorul va recunoaşte modurile de adresare folosite (pentru operanzi şi rezultat) şi va putea calcula locul în care se găsesc operanzii ceruţi de execuţia instrucţiunii.

Operanzii pot fi imediaţi, sau se pot găsi în registre interne UCP, în memorie, sau în registre port de I/O. În cazul unui program executabil, valoarea operanzilor este calculată în momentul compilării / asamblării programului sursă pentru operanzii imediaţi, în momentul încărcării în memoria principală pentru adresarea directă la memorie şi respectiv, în momentul execuţiei pentru operanzii adresaţi în registre, sau adresaţi indirect.

În general instrucţiunile care specifică numai operanzi în registre sunt compacte şi se execută rapid, pentru că registrele sunt incluse în UCP şi nu mai sunt necesare transferuri pe magistrala externă. Registrele pot conţine operanzi sursă şi/sau destinaţie).

Operanzii imediaţi sunt date constante de 8, 16, 32, sau 64 biţi conţinute în instrucţiune. La microprocesoarele care au memorie intermediară internă pentru instrucţiuni şi date, (de exemplu coada de instrucţiuni a lui 8086, sau memorie cache internă la alte microprocesoare) operanzii imediaţi pot fi adresaţi la fel de rapid ca şi cei din registre, la momentul execuţiei instrucţiunii, ne-fiind necesare operaţii suplimentare pentru extragerea lor. Limitările operanzilor imediaţi se datorează faptului că ei au doar valori constante şi pot servi numai ca operanzi sursă.

Pentru fiecare mod de adresare descris în continuare, automatul de adresare al UCP calculează o *adresă efectivă (AE)* a operandului. Obţinerea AE se face în funcţie de modul de adresare selectat prin cuvântul de cod al instrucţiunii. AE este adresa care va fi folosită de instrucţiune pentru a găsi operandul. La microprocesoarele cu memoria segmentată, AE rezultă ca un întreg fără semn ce reprezintă deplasamentul (adresa relativă a operandului) faţă de începutul segmentului în care se află.

În descrierea modurilor de adresare se vor folosi notaţiile:

- **r** - registru (eventual r8, r16 etc pentru a indica dimensiunea), litera referindu-se la conţinutul acestui registru.
- **d** - deplasament (adresă relativă). Este o valoare care participa la calculul AE, fiind tratat de obicei ca un număr cu semn în cod complement faţă de doi. Adunarea deplasamentului poate genera o adresa mai mică sau mai mare decât adresa de bază (pentru reprezentarea lui d cu 8 biţi în C2, variaţia faţă de adresa de bază se face în intervalul -128 ... +127. Restul componentelor care intră în calculul AE sunt tratate ca numere fără semn.
- **x** - adresă de memorie; se referă la conţinutul locaţiei de memorie de adresă x.
- **@x** - adresa de la adresa x (adresa conţinută în locaţia de memorie cu adresa x; la adresa x se găseşte nu o simplă dată, ci un pointer către altă adresă).
- **@r** - operandul este conţinutul locaţiei de memorie cu adresa conţinuta în registrul r.

Nu se va presupune un anumit tip de microprocesor; deci cuvintele pentru codul operaţiei, registrele implicate, valorile numerice numite deplasări şi operanzi, pot avea 8, 16 sau mai mulţi biţi, în funcţie de tipul instrucţiunii şi de tipul microprocesorului. În documentaţiile de firmă apar de obicei mult mai multe moduri de adresare decât folosim în clasificarea de mai jos, pentru că se fac particularizări ale fiecărui mod de adresare. Motivele pot fi şi comerciale, dar toate modurile de adresare pot fi încadrate în clasificarea de mai jos.

Pentru a simplifica explicaţia, se va presupune în continuare că memoria este organizată pe octeţi (fiecare octet are o adresă specifică), că se foloseşte convenţia de memorare Little Endian şi că este adresată liniar (şi nu segmentat ca la procesoarele I80x86).

Dacă ne referim la segmentarea memoriei, la microprocesorul I8086, memoria este segmentată în segmente logice de câte 64 KB. Pentru I8086 pot exista 4 segmente active simultan în memorie, segmente a căror adresă de bază se găseşte în registrele segment CS, DS, SS, ES. Prin modurile de adresare specifice lui I8086 se calculează o adresă efectivă de 16 biţi (care este un deplasament în cadrul segmentului curent).

Adresa fizică (AF) se obţine din adresa efectiva (AE) la care se adaugă adresa de segment (AS) înmulţită cu 16, ca în ecuaţia:

$$AF = AS \times 2^4 + AE \qquad (2.2)$$

Această sumare nu se face de către ALU ci de către un sumator separat de 20 biţi şi este transparentă pentru programator.

Se va face clasificarea modurilor de adresare în 5 tipuri:

1. adresarea imediată
2. adresarea directă
3. adresarea indirectă
4. adresarea relativă
5. adresarea indexată

2.6.1. Adresare imediată

Corpul (formatul) instrucţiunii înglobează şi operandul care apare *imediat* după codul operaţiei, aşa cum se indică în reprezentarea grafică a modului de adresare din figura 2.5.

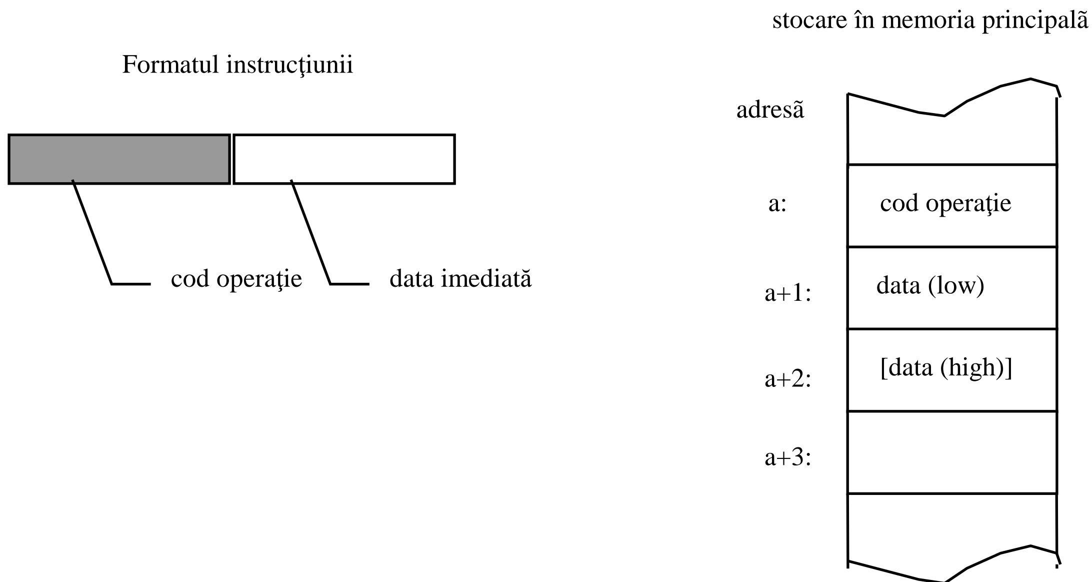


Figura 2.5. Explicativă la modul de adresare imediată

Operandul este format dintr-unul sau mai mulţi octeţi succesivi. Modul imediat de adresare este folosit numai pentru adresarea datelor constante ale programului. La microprocesoarele de 8 biţi sunt necesare 2 sau mai multe citiri din memorie (cod + data). La microprocesoarele de la 16 biţi în sus, cu tampon local (de tip FIFO sau cache) de date şi instrucţiuni, aceste instrucţiuni sunt executate rapid, pentru că operandul se afla în tamponul microprocesorului.

Exemple (I8086):

```
and al,0FFh        ; adresare imediată la data 0FF
mov ax,3A40h       ;adresare imediată la data 3A40hex, în memorie
                   ;se scrie b8 40 3a , unde b8 este codul operaţiei iar data
                   ;multi-octet se stochează în convenţia little endian
mov al,5           ;în hexa, în memorie= b0 05
mov ax,5           ;în hexa, în memorie = b8 05 00
```

2.6.2. Adresare directă

În instrucţiune apare un câmp ce specifică o adresă de date (fixă - nu mai poate fi modificată în timpul rulării). Adresa apare după codul operaţiei instrucţiunii, ca şi la adresarea imediată, dar de data aceasta ea nu reprezintă un operand, ci adresa unde se găseşte operandul. Această adresă poate fi identificatorul (numele) unui fanion de condiţii, adresa unui registru intern UCP (operandul este conţinut de registru), sau o adresa din memoria principală (adresa la care se găseşte operandul). Din punctul de vedere al locului specificat de adresă există variantele acestui mod de adresare, numite:

- *adresare directă la registru (implicită),*
- *adresare directă la memorie (absolută sau extinsă),*
- *adresare directă la fanion/indicator* (pentru testarea acestuia); este cazul adresărilor directe ale instrucţiunilor, atunci când se face un salt condiţionat sau o chemare de procedură condiţionată.

În cazul *adresării directe la registru, (AE = r)*, operandul este conţinut într-un registru. Informaţia pentru AE este de multe ori inclusă în codul operaţiei, prin codul ce indică adresa registrului intern UCP implicat în instrucţiune. Avantajele acestui mod de adresare se referă în primul rând la codificarea uşoară a adresei (numărul de registre interne nu este foarte mare) şi execuţia rapidă, pentru că nu este necesar un acces suplimentar la memoria principală, exterioară UCP.

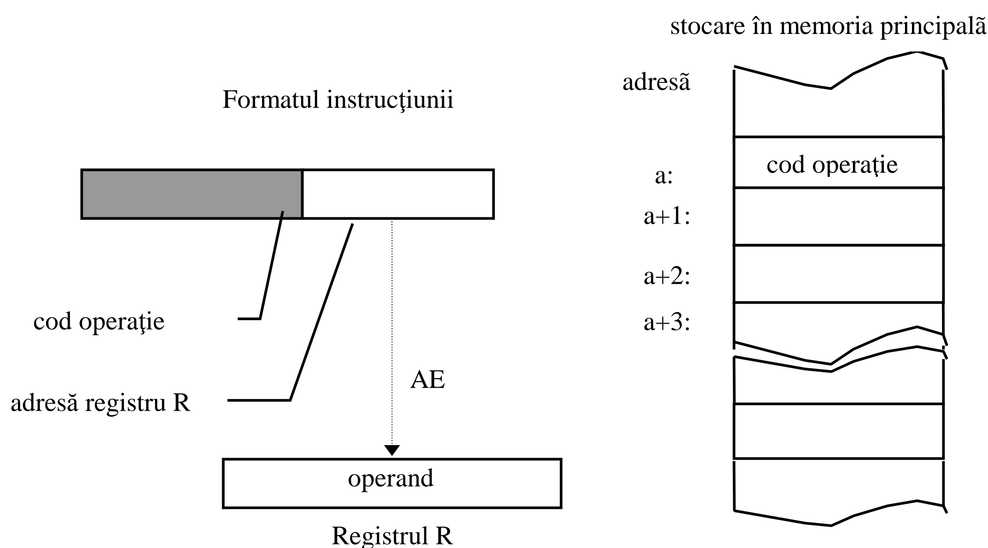


Figura 2.6.a. Explicativă la modul de adresare directă la registru

Atunci când adresa registrelor este inclusă implicit în codul operaţiei, acea instrucţiune lucrează doar cu acele registre, iar adresarea este numită adresare implicită. În cazul arhitecturilor RISC, adresele registrelor interne apar întotdeauna explicit în formatul binar al instrucţiunilor. Pentru unele microprocesoare, termenul "*implicit*" are o semnificaţie mai aparte: informaţia se află implicit (sau rezultatul se va transfera implicit) într-un registru dedicat numit acumulator (deci nu mai exista biţi de adresare pentru acel registru).

Adresarea directă la memorie mai este numită *adresare absolută*, sau *adresare extinsă.* Adresa efectivă se poate scrie ca fiind AE = x. Instrucţiunile cu acest mod de adresare conţin chiar adresa efectivă a operandului. AE apare în corpul instrucţiunii, după codul operaţiei. Adresarea directă la memorie se foloseşte mai ales la instrucţiunile de transfer între registrele interne şi memorie, precum şi la instrucţiunile de salt absolut.

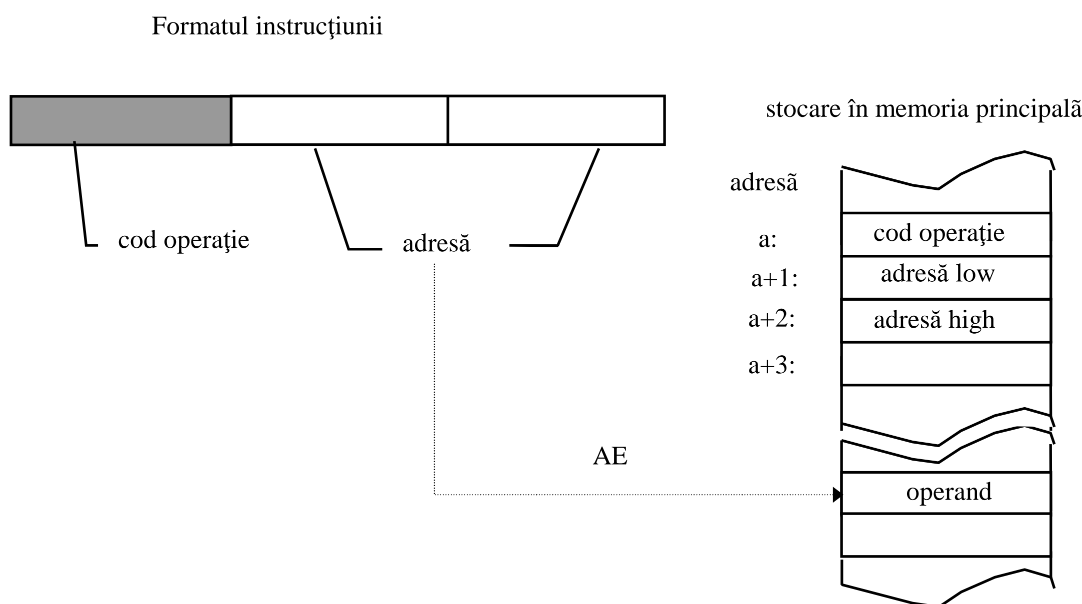


Figura 2.6.b. Explicativă la modul de adresare directă la memorie

Numărul de octeţi din câmpul operand depinde de tipul microprocesorului. De exemplu la microprocesoarele de 8 biţi, adresa este de obicei pe 16 biţi (spaţiu maxim de adresare de 64 KB). La cele de 16 biţi, cu adresa segmentată, cum sunt procesoarele Intel, există două situaţii de referire la o adresă de memorie:

- adresă în interiorul aceluiaşi segment (AE = adresă pe 16 biţi)
- adresă în afara segmentului curent, care necesită AE pe 32 de biţi.

La microprocesoarele care au spaţii de adresare separate pentru memorie şi pentru dispozitivele de I/O există o variantă a modului de adresare directă, numită *adresare directă la port*. De obicei în aceste cazuri adresa portului are dimensiunea de 8 biţi, deci pot fi adresate maximum 256 porturi

Exemplu pentru I8086:

```
out adr_p8,al      ;Se transferă conţinutul registrului de 8 biţi numit
                   ;"al" la adresa de port adr_p8
in ax,adr_p8       ;Se citesc 16 biţi de la portul de adresă adr_p8 şi se
                   ;transferă în registrul ;de 16 biţi numit "ax"
```

Adresarea directă la memorie este *nerecomandată*, pentru că programele nu sunt relocatabile. În plus lungimea instrucţiunilor ce folosesc adresare directă la memorie au lungime foarte mare. Toate programele de translatare în cod maşină avertizează programatorul, dacă se foloseşte acest mod de adresare.

2.6.3. Adresare indirectă

La acest mod de adresare în câmpul operand al instrucţiunii se specifică un registru sau o adresă de memorie principală din care se va citi adresa operandului. Există astfel adresare indirectă prin registru (AE = @r) şi adresare indirectă prin memorie (AE = @x).

De exemplu, se poate scrie:

```
mov ax,@bx         ;adresa de memorie se găseşte în registrul bx. Se transferă în
                   ;registrul ax informaţia din memorie de la adresa dată de
                   ;conţinutul registrului bx
mov ax,@there      ;locaţia de memorie de adresa simbolică there conţine adresa datelor.
                   ;Se transferă în registrul ax informaţia din memorie de la adresa
                   ;dată de conţinutul locaţiei cu adresa simbolică there
```

La *adresarea indirectă prin registru*, adresa operandului este conţinută în registrul specificat de codul instrucţiunii. Acest mod de adresare permite o flexibilitate sporită în localizarea informaţiilor în memorie. În formatul instrucţiunii, pentru un operand, se adresează un registru intern UCP, în care se află AE a operandului. Este un mod util de adresare atunci când adresa efectivă a operandului este variabilă. Adresa registrului de adresare se poate specifica *explicit* (la procesoarele cu un mare număr de registre generale, unde oricare registru poate fi folosit la adresare indirectă) sau *implicit*, (adresa binară a registrului nu apare explicit în vectorul

binar al instrucţiunii) în cazul în care la acest mod de adresare se poate folosi doar un registru (de obicei de lungime dublă faţă de dimensiunea magistralei de date).

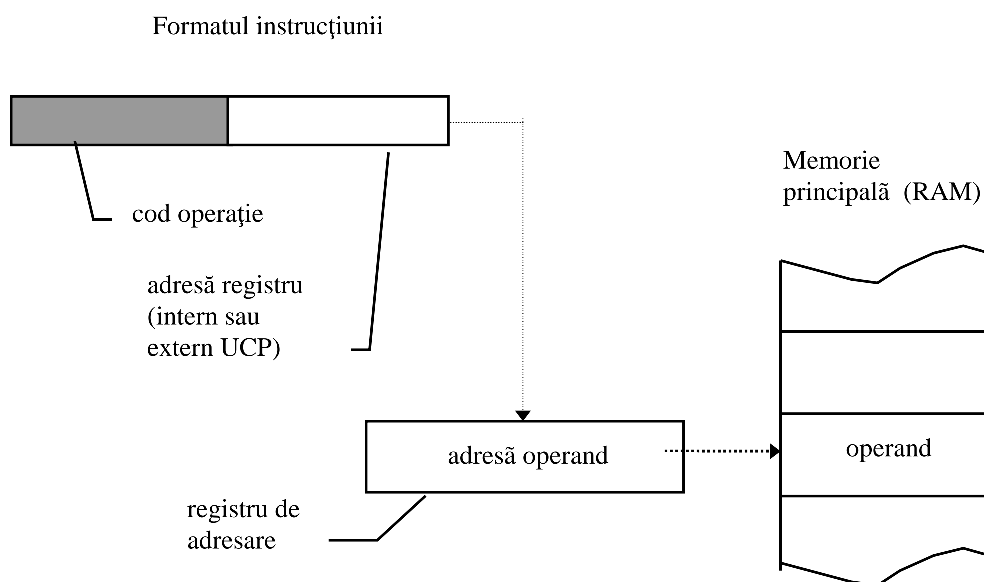


Figura 2.7. Explicativă la modul de adresare indirectă la memorie

Adresarea indirectă prin registru (figura 2.7) poate fi folosită şi la instrucţiunile de apel al procedurilor, adresa efectivă găsindu-se indirect, prin citirea conţinutului unui registru de adresare.

De exemplu la Intel 8086 se poate scrie:

call bx ;bx → ip, care face o adresare *directă* la registrul bx

sau

call [bx] ;@bx → IP_L şi @(bx+1) → IP_H, acesta fiind o adresare indirectă prin ;registrul bx. Adresa de început a procedurii se găseşte în memorie ;la adresa dată de registrul bx.

La microprocesoarele care au spaţii de adresare separate pentru memorie şi pentru dispozitivele de I/O există o variantă interesantă a *adresării indirecte prin registru a porturilor*. Această adresare este similară cu adresarea indirectă prin registre a memoriei. Numărul porturilor posibil de adresat este dat de dimensiunea registrului folosit în adresare. De exemplu, la I8086, adresa portului este stocată în registrul "dx" de 16 biţi, deci numărul maxim de porturi adresabile este 65536. La acest microprocesor, registrul dx este singurul ce poate fi folosit la adresare indirectă a porturilor şi de aceea registrul apare implicit în codul operaţiei. De exemplu codurile (în hexazecimal) ale instrucţiunilor pentru I8086 sunt:

```
in ax,dx        ;cod instrucţiune = ED hex. Aduce (input) în registrul intern ax
                ;data de la portul specificat prin adresa din registrul dx
out dx,al       ; cod instrucţiune = EE hex. Transmite (output) din registrul intern
                ;al data de la portul specificat prin adresa din registrul dx
out dx,ax       ; cod instrucţiune = EF hex. Transmite (output) din registrul intern
                ;ax data de la portul specificat prin adresa din registrul dx
```

Din ultimele două exemple se observă că există un singur bit care specifică transferul pe 8 biţi (în registrul al) sau pe 16 biţi (în registrul ax).

Adresarea *indirectă prin memorie, AE = @x,* este dezavantajoasă din punctul de vedere al vitezei, pentru că, pentru a găsi operandul sunt necesare două citiri succesive ale memoriei. Acest mod de adresare este folosit de multe microprocesoare în cazul servirii cererilor de întrerupere, adresarea indirectă făcându-se prin intrările tabelului vectorilor de întrerupere.

Unele calculatoare permit incrementarea sau decrementarea automată a registrului folosit pentru adresare, fie înainte fie după utilizare. Rezultă astfel variante de adresare pentru adresarea indirectă prin registru:

a) indirectă prin registru cu auto-post-incrementare: se face acces la o locaţie de memorie a cărei adresă este conţinută într-un registru, iar după fiecare acces de acest fel registrul este incrementat automat cu un număr egal cu dimensiunea datei.

De exemplu, dacă registrele r1 şi r2 sunt de 16 biţi, iar memoria este adresabilă pe octet, se poate scrie:

```
mov r1,@r2+     ;registrul r2 conţine adresa de memorie unde se găsesc datele,
                ;să zicem adresa 61h. După execuţia instrucţiunii (copiere
                ;din memorie în registrul r1, conţinutul lui r2 va fi 63h pentru
                ;că s-au citit doi octeţi, de la adresele 61h şi 62h
```

b) indirectă prin registru cu auto-post-decrementare: se face acces la o locaţie de memorie a cărei adresă este conţinută într-un registru, iar după fiecare acces de acest fel registrul este decrementat automat cu un număr egal cu dimensiunea datei.

De exemplu:

```
    mov r1,@r2–     ;registrul r2 conţine adresa de memorie unde se găsesc datele,
                    ;să zicem adresa 61h. După execuţia instrucţiunii (copiere
                    ;din memorie în registrul r1, conţinutul lui r2 va fi 5Fh (61hex–2)
                    ;pentru că s-au citit doi octeţi, de la adresele 61h şi 62h
```

b) indirectă prin registru cu auto-pre-incrementare: se face acces la o locaţie de memorie a cărei adresă este conţinută într-un registru şi al cărui conţinut a fost incrementat automat înainte de utilizare, cu un număr egal cu dimensiunea datei.

De exemplu:

mov r1,+@r2 ;registrul r2 conţine adresa de memorie unde se găsesc datele,
;să zicem adresa 61h. Citirea datelor şi copierea în r1 începe
;de la adresa 63h.

b) indirectă prin registru cu auto-pre-decrementare: se face acces la o locaţie de memorie a cărei adresă este conţinută într-un registru şi al cărui conţinut a fost decrementat automat înainte de utilizare, cu un număr egal cu dimensiunea datei.

De exemplu:

mov r1,–@r2 ;registrul r2 conţine adresa de memorie unde se găsesc datele,
;să zicem adresa 61h. Citirea datelor şi copierea în r1 începe
;de la adresa 5Fh.

În acest grup de adresări indirecte prin registru, cu actualizarea automată a registrului de adresare este inclusă şi tehnica de adresare a memoriei stivă. La memoria stivă data este conţinută în locaţia de memorie din vârful curent al stivei, locaţie a cărei adresă este totdeauna conţinută în registrul indicator de stivă (SP).

Adresarea stivei se face indirect prin registrul SP, cu incrementarea sau decrementarea automată a conţinutului lui SP, în funcţie de operaţia efectuată cu stiva. Pentru introducerea informaţiei în stiva ("push" - adăugarea unui element în stivă se consideră conform convenţiei Intel - adăugare către adrese mici) se decrementează mai întâi SP (operaţie de pre-decrementare), iar apoi informaţia se stochează în memorie în locaţia indicată de adresa efectiva conţinută în SP. Pentru operaţia de extragere din stivă ("pop") se extrage din stivă conţinutul locaţiei adresate de SP şi apoi se incrementează SP (post-incrementare).

Operaţia de scriere în stivă ("*push sursa*") poate fi scrisă

mov –@SP,sursa

care presupune succesiunea de operaţii:

SP← SP - N ; N indică numărul de octeţi înscrişi
;în stivă în cadrul operaţiei curente.
@SP← sursa ;sursa este conţinutul unui registru
;sau locaţie de memorie

Operaţia de extragere din stivă ("*pop dest*") într-un registru destinaţie poate fi scrisă

mov dest,@SP+

care presupune succesiunea de operaţii:

dest ← @SP
SP ← SP + N

2.6.4. Adresare relativă

La acest mod de adresare în corpul instrucţiunii se găseşte un operand (un deplasament, notat cu d) care este adunat la adresa de start a unei secţiuni de memorie, adresă conţinută într-un registru desemnat explicit sau implicit în instrucţiune.

Putem deosebi următoarele variante de adresare relativă:

a) *adresare relativă bazată* (numită şi *indirectă prin registru bază, cu deplasare*);

AE = @(r + d)

De exemplu:

mov ax,[bx+8] ;daca în bx se gaseste adresa 07h citirea operandului ce se va copia în ax se face de la adresa 7+8 = 15 = 0fh. Pentru că registrul este pe 16 biţi iar memoria este adresată pe octet, se citesc locaţiile 0fh şi 10h

Adresa efectivă (AE) este suma dintre conţinutul; registrului bază şi deplasarea. Modul de adresare este folosit pentru adresarea unor structuri de tip tablou. Prin registru de bază se înţelege un registru utilizat pentru calculul de adresă efectivă. Registrul de bază păstrează adresa de început a unei zone de memorie, unde sunt încărcate programe aflate în execuţie sau blocuri de date.

Este un mod de adresare util atunci când se urmăreşte accesul la un operand aflat într-o tabelă ce poate ocupa diferite locuri în memorie. Pentru calculul AE la I8086 (cu memorie segmentată), se adună conţinutul registrului bp sau bx la un deplasament furnizat în cadrul instrucţiunii. Pentru calculul adresei fizice se foloseşte registrul ds sau ss, după cum calculul adresei efective foloseşte registrul bx respectiv bp.

Adresa_fizică = ds * 16 + bx + d sau

Adresa_fizică = ss * 16 + bp + d

b) adresare relativa la contorul de program (PC)

În mod obişnuit la execuţia de secvenţe succesive de instrucţiuni din memorie avem de-a face cu o adresare indirectă prin registrul PC cu auto-post-incrementare. Exemplul următor, poate deci fi trecut ca exemplu şi la adresarea indirectă: AE = @PC+

Adesea acest mod de adresare relativă este numit *adresare secvenţială*. În cazul unor ramificaţii însă, la conţinutul PC se adună deplasamentul cu semn din câmpul operand.

De exemplu:

```
IT:     mov dx,there        ; instrucţiune cu lungime de 3 octeţi
        mov ax,1732h        ; lungime de 3 octeţi
        mov bx,cx           ; lungime de 2 octeţi
```

```
jmp *-9          ; lungime de 2 octeţi. La primul octet se decodifică
                 ;salt relativ se face salt la instrucţiunea cu eticheta IT.
```

2.6.5. Adresare indexată

Adresarea indexată este asemănătoare ca principiu, cu adresarea relativă bazată, dar spre deosebire de aceasta ea este folosită *doar în adresarea datelor*. Prin acest mod de adresare, adresa datelor din memoria principală este modificată cu ajutorul conţinutului unui registru index. Adresarea indexată se utilizează pentru a referi elementele unui vector, poziţia unui element din vector fiind determinată printr-un indice, iar indicele se poate modifica prin modificarea valorii conţinute în registrul index.

Numărul de registre index dintr-un procesor indică numărul de tablouri potenţiale care se pot adresa în memorie, iar mărimea deplasamentului determină dimensiunea maximă a tabloului. Numărul registrelor index este un criteriu de performanţă pentru microprocesoare.

Indexarea se combină de obicei cu alte moduri de adresare putând exista variantele: adresare indexată directă la memorie, adresare pre-indexată indirectă la registru, adresare post-indexată indirectă la registru, adresare pre-indexată indirectă la memorie, adresare post-indexată indirectă la memorie, adresare indexată relativă la adresa de bază (ultima fiind adesea numită şi adresare bazată şi indexată). De observat că modurile combinate dintre indexare şi adresările indirecte pot specifica că indexarea se va realiza înainte sau după ce indirectarea este efectuată. Astfel, similar cu cele expuse la adresarea indirectă prin registre:

AE = @(ri + d), sau

AE = @(ri + x), sau

AE = @(ri + r)

De exemplu, la microprocesorul I8086 se pot utiliza adresări indexate de genul:

a) Adresare indexată-directă la memorie

```
mov ax,adr1[si]     ;fie adresa lui adr1 = 29h. Adresa datelor este egală cu
                    ;adr1+ si, iar conţinutul registrului index si se consideră a
                    ;fi 2. Ca urmare copierea în registrul ax se face de la
                    ;adresele 2Bh şi 2Ch pentru că (29+02=2B hex)
```

b) Adresare indexată indirectă prin registru bază (adresare bazată şi indexată)

```
mov ax,[bp][si]     ;dacă conţinuturile bazei şi indexului sunt bp=34h, si=04h,
                    ;datele se citesc începând cu adresa 38h
```

2.7. Exerciţii

1. Clasificaţi şi descrieţi modurile de adresare tipice pentru un microprocesor de uz general.
2. Definiţi noţiunea de aliniere la stocarea informaţiilor multi-octet în memorie şi explicaţi convenţiile de adresare Little Endian şi Big Endian
3. Pentru o memorie organizată pe octet, informaţia cu lungimea de 4 octeţi stocată la adresa de memorie: 4A22 hex este: (a) aliniată; (b) ne-aliniată, Justificaţi răspunsul ales ca fiind corect
4. Tipuri de instrucţiuni caracteristice unui microprocesor de uz general.
5. Considerând cele două variante de implementare a unui set de instrucţiuni, cu lungime fixă, respectiv cu lungime variabilă , descrieţi pe scurt avantajele şi dezavantajele fiecăreia dintre variante
6. Enumeraţi modurile de adresare şi descrieţi modul de adresare indirectă (inclusiv variantele acesteia)
7. Realizaţi o analiză comparativă între caracteristicile arhitecturilor CISC şi RISC Descrieţi pe scurt modul de adresare utilizat pentru înscrierea şi extragerea în / din memoria stivă şi încadraţi aceste moduri de adresare în modurile de adresare generale ale unui microprocesor.
8. Structură elementară de UCP pe bază de acumulator (schemă bloc, descriere componente, funcţionare, extinderea UCP elementare prin perfecţionarea unor blocuri funcţionale).
9. Convenţia folosită pentru adresarea datelor şi instrucţiunilor la care adresarea informaţiei se face prin adresa LSByte este numită:
   (a) Little Endian
   (b) Big Endian
   (c) Indexată
   (d) FIFO Justificaţi răspunsul ales ca fiind corect
10. Convenţia folosită pentru adresarea datelor şi instrucţiunilor la care adresarea informaţiei se face prin adresa MSByte este numită:
    (a) Little Endian
    (b) Big Endian
    (c) Indexată
    (d) FIFO Justificaţi răspunsul ales ca fiind corect
11. Definiţi următoarele caracteristici principale ale unui set de instrucţiuni: regularitate (ortogonalitate), completitudine
12. Arhitectura de tip load / store se referă la o arhitectură:
    a. CISC
    b. von Neumann

c. RISC

d. Harvard

e. toate cele de mai sus Justificaţi răspunsul ales ca fiind corect

13. Scopul unei instrucţiuni este să specifice:
    (a) frecvenţa de ceas a microprocesorului
    (b) o operaţie ce va fi realizată de microprocessor
    (c) comenzi pentru magistrala de date
    (d) specificatorii de operand ce vor fi utilizaţi
    (e) tipul microprocesorului

    Explicaţi pe scurt răspunsul (răspunsurile) alese ca fiind corecte
14. Explicaţi modurile de adresare utilizate la următoarele instrucţiuni:
    a) and al,0FEh
    b) mov ax,bx
    c) mov r1,@r2-
    d) mov r1,+@r2
    e) mov ax,[bx+8]
15. Dacă se utilizează o memorie adresabilă la nivel pe octet, iar conţinutul registrelor de 16 biţi r1 şi r2 este: r1=2FAC hex; r2=A000 hex, să se indice conţinutul registrelor după execuţia instrucţiunilor:
    a. mov r1,@r2+
    b. mov r1,@r2–
    c. mov r1,+@r2
    d. mov r1,–@r2

Capitolul 3.

Organizarea şi funcţionarea microprocesorului de uz general

Conţinut

3.1. Structura de procesor de uz general

3.2. Structura unui microprocesor elementar

3.3. Alte registre interne semnificative pentru UCP

3.4. Exemple privind operaţiile UCP şi modul de setare al indicatorilor de condiţii

3.5. Semnale la interfaţa UCP cu exteriorul

3.5.1. Magistrala de date

3.5.2. Magistrala de adrese

3.5.3. Magistrala de control

3.6. Sistemul de întreruperi

3.6.1. Rolul memoriei stivă

3.6.2. Utilizarea ferestrelor de registre

3.7. Exerciţii

3.1. Structura de procesor de uz general

Microprocesorul, ca unitate centrală de prelucrare (UCP) a calculatorului numeric este o structură de procesor de uz general, care recunoaşte un set de instrucţiuni. UCP este un procesor *de uz general*, spre deosebire de alte procesoare cu set de instrucţiuni, cum ar fi procesoarele de I/O, sau procesoarele aritmetice, care îndeplinesc doar funcţii limitate, specifice. De asemenea UCP este un procesor *cu set de instrucţiuni*, pentru că UCP recunoaşte şi execută un set specific de instrucţiuni binare, furnizate din exteriorul său.

În cadrul unui sistem de calcul, UCP are responsabilitatea generală de *interpretare* şi *execuţie* a instrucţiunilor unui program.

În accepţiunea clasică (von Neumann) a unui sistem de calcul, maşina conţine o singură Unitate Centrală de Prelucrare. Un astfel de calculator este numit *uniprocesor*, pentru a-l deosebi de calculatoarele *multiprocesor*, care conţin două sau mai multe UCP.

Ca structură generală, un procesor cuprinde o unitate de prelucrare a datelor (care conţine o unitate aritmetică şi logică şi registrele folosite ca memorie locală) şi o unitate de control. *Sistemul digital format dintr-un automat de prelucrare aritmetică şi logică şi o unitate de control este numit **procesor**.*

Unitatea Aritmetică şi Logică (ALU), efectuează operaţii cu datele de intrare (operanzi) şi conţine pe lângă structurile combinaţionale pentru efectuarea acestor operaţii, circuite de tip registru pentru memorarea locală, sau prelucrarea datelor şi circuite necesare transferului de informaţie între registre şi cu exteriorul ALU (circuite de decodificare, codificare, multiplexare, comparare, circuite tampon, etc.). Registrele locale folosite ca memorie de mare viteză, pot fi folosite şi pentru efectuarea unor operaţii aritmetice sau logice. Principalele operaţii realizate de ALU sunt:

- operaţii aritmetice şi logice cu operanzi de intrare codificaţi pe n biţi. Valoarea lui n este egală de obicei cu numărul de biţi ai magistralei interne de date a procesorului, dar există şi excepţii de la aceasta regulă.
- furnizează indicaţii privind transportul, împrumutul, sau alte caracteristici privind rezultatul operaţiilor efectuate, prin intermediul unor valori binare numite indicatori de condiţii (în engleză *flags*).

În figura 3.1. datele de intrare în unitatea de prelucrare a datelor sunt intrările pentru operanzii citiţi din exterior. Aceşti operanzi se citesc ca şi cuvinte binare (cu n biţi în paralel) şi se stochează în registre temporare, interne unităţii de prelucrare a datelor. Rezultatul operaţiilor efectuate, sau conţinutul unor registre de stocare temporară, se poate furniza în exterior prin liniile notate "date ieşire".

Intrările în unitatea de prelucrare notate "comenzi interne procesorului", reprezintă comenzi către ALU generate de unitatea de control. Conform acestor comenzi ALU va prelucra datele de la intrare şi va furniza datele la ieşire (registre interne sau externe). Aceste comenzi pot reprezenta:

- codul funcţiei ce trebuie executată de ALU. Dacă considerăm, de exemplu un cod al funcţiei pe 4 biţi, ALU poate efectua maximum 16 operaţii aritmetice şi logice;
- informaţii de adresare a registrelor interne unităţii de prelucrare, (de exemplu pentru a se indica registrele sursă şi / sau destinaţie ale operaţiilor efectuate de ALU, pentru a se selecta / valida registre tampon de memorare a valorilor binare de la "Intrare date " sau la "Ieşire date ");
- semnale de sincronizare (destinată logicii secvenţiale interne a unităţii de prelucrare); aceste semnale indică momentele de timp când se face transferul între registre, respectiv momentele de încărcare a unor registre.

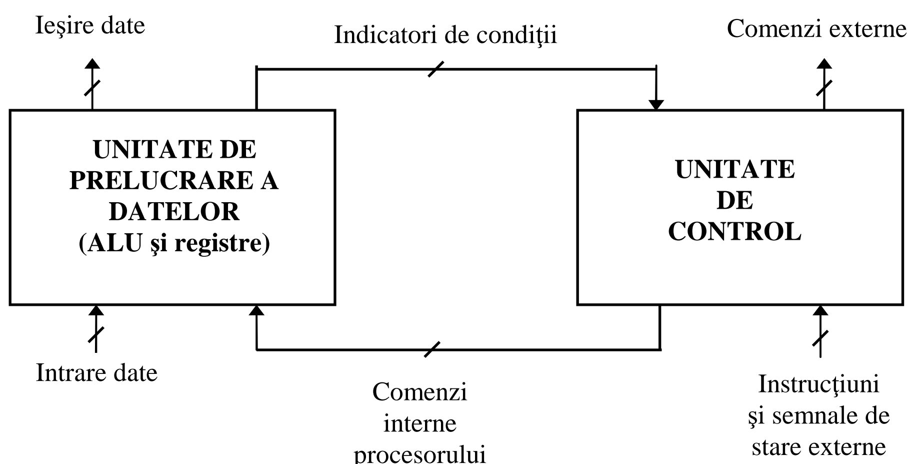


Figura 3.1. Structura bloc a unui procesor de uz general. Liniile oblice pe căile de intrare şi ieşire a semnalelor indică existenţa mai multor linii de semnal, fără specificarea numărului de biţi transferaţi în paralel

Indicatorii de condiţii sunt poziţionaţi de unitatea de prelucrare în conformitate cu "evenimentele" ce s-au produs la efectuarea operaţiei comandate anterior pentru ALU. Aceşti indicatori (numiţi uneori fanioane) sunt memoraţi de obicei în bistabile, incluse ca celule ale unor registre de indicatori, registre interne UCP. Indicatorii de condiţii pot reprezenta, de exemplu,

semnul rezultatului, rezultat egal cu zero, paritatea rezultatului etc.

Succesiunea de comenzi primită de ALU este generată de un automat cu algoritm de stare încorporat, numit *Unitate de Control* (UC). Algoritmul de stare defineşte succesiunea stărilor automatului, respectiv succesiunea şi natura comenzilor. Algoritmul de stare poate fi implementat fie în structura fizică (hardware) a UC, fie prin înscrierea comenzilor binare secvenţiale într-o memorie de control de mare viteză. În primul caz vorbim de unitate de *control cablată*, iar în al doilea caz de unitate de control *microprogramată*.

Datorită reacţiei pe care UC o primeşte de la Unitatea de prelucrare, UC examinează caracteristicile rezultatelor (prin testarea valorii indicatorilor de condiţii) şi astfel poate modifica secvenţa de comenzi în mod corespunzător. UC primeşte din exterior "instrucţiuni", fiecare din acestea producând generarea unei secvenţe de comenzi elementare către unitatea de prelucrare şi către exteriorul procesorului. Aceste comenzi controlează succesiunea micro-operaţiilor pe care le efectuează procesorul pentru a *executa* operaţia ce corespunde instrucţiunii primite. Instrucţiunile se păstrează într-o memorie externă procesorului. Microprocesorul este cel care decide şi succesiunea în care instrucţiunile vor fi aduse din memoria operativă (funcţia de ***secvenţiere***). Astfel că UC trebuie să genereze în exterior şi comenzi (informaţii) de adresare a instrucţiunilor în memorie. Pentru ca transferul instrucţiunilor binare să se facă corect, alte comenzi de la UC vor activa şi circuitele ce comandă liniile de legătură între intrările UCP şi ieşirile circuitelor de memorie. Ordinea şi momentele de timp la care se generează comenzile externe poate fi influenţată de semnalele de stare externe pe care le primeşte UC (de exemplu, semnale care indică unităţii de control că un circuit comandat nu poate răspunde în timp la aceste comenzi, deci solicită o întârziere). De aceea, se poate spune că unitatea de control poate fi împărţită funcţional, în două sub-unităţi de control: prima sub-unitate face controlul la nivel de micro-operaţii, iar cea de-a doua face control la nivel de secvenţiere a instrucţiunilor din memoria externă (automat de adresare).

Interacţiunea unităţii de control cu dispozitive externe necesită "sincronizarea" acţiunilor. ***Sincronizarea*** operaţiilor UC (şi implicit a UCP) cu exteriorul poate fi clasificată în *trei niveluri de sincronizare*:

1. nivelul elementar al semnalului de *ceas* aplicat din exteriorul UCP;
2. nivelul semnalelor de sincronizare şi control pentru *transferurile de date* cu memoria principală şi porturile de I/O;
3. nivelul semnalelor de *întrerupere* pentru sincronizarea UCP cu evenimente externe.

Funcţionarea microprocesorului este ciclică, conform schemei bloc din figura 3.2, până la terminarea execuţiei tuturor instrucţiunilor programului.

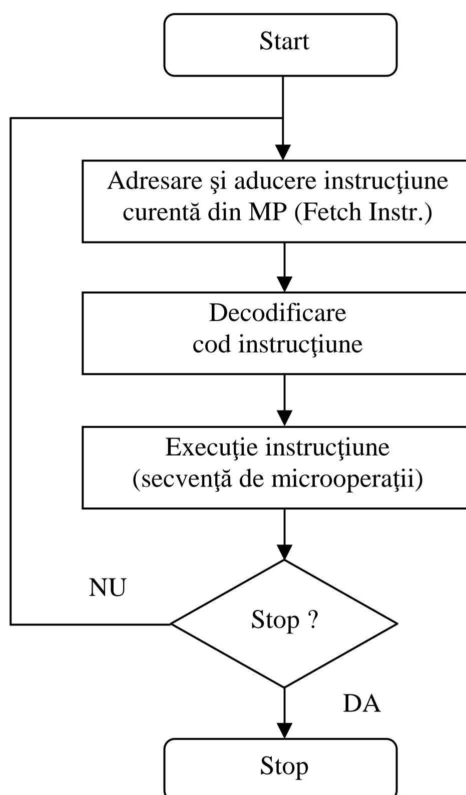


Figura 3.2. Schema bloc care indică funcţionarea ciclică a procesorului. (MP = memoria principală)

În prima fază unitatea de control adresează şi controlează aducerea unei instrucţiuni din memoria principală (ciclu maşină numit *„fetch instruction”* - aducere cod instrucţiune). Apoi UC decodifică codul instrucţiunii şi generează secvenţa de comenzi specifică instrucţiunii către unitatea de prelucrare şi către exterior (de exemplu pentru citirea operanzilor). În faza de execuţie ALU execută comenzile (realizând operaţii aritmetice sau logice asupra operanzilor) şi setează indicatori de condiţii, care sunt citiţi de UC. Stocarea rezultatului prelucrării se poate face în registre interne ale ALU, sau în registre externe UCP (de exemplu memorie, sau port de intrare / ieşire). Dacă mai sunt instrucţiuni în program unitatea de control revine la prima fază, de aducere a instrucţiunii următoare (ciclul de fetch).

Funcţia principală a unui procesor de uz general, ca Unitate Centrală de Procesare (UCP) a calculatorului, este să execute secvenţa de instrucţiuni stocată într-o memorie externă calculatorului, numită memoria principală (MP). Secvenţa de operaţii implicate în aducerea şi execuţia fiecărei instrucţiuni, descrisă în figura 3.2, se produce într-un interval de timp, multiplu al perioadei impulsurilor de ceas, interval numit **ciclu instrucţiune.**

Comportarea UCP în timpul ciclului instrucţiune e definită printr-o secvenţă de micro-operaţii, fiecare din acestea implicând operaţii de transfer între registre. Timpul t_{UCP} cerut pentru cea mai scurtă şi bine definită micro-operaţie a UCP este numit **timp de ciclu al UCP** (sau **stare** a

UCP, corespunzătoare unei stări a automatului UC) şi este unitatea de timp de bază pentru măsurarea tuturor acţiunilor UCP. Inversul acestui timp este frecvenţa de ceas (clock) la care funcţionează UCP, măsurată în MHz sau GHz. Aceasta frecvenţă este dependentă direct de arhitectura şi de tehnologia de fabricaţie a UCP. În cele mai multe dintre (micro-)procesoarele implementate frecvenţa la care funcţionează UCP nu este identică cu frecvenţa oscilatorului de ceas cu cuarţ, care fixează baza de timp a microprocesorului. Frecvenţa de lucru a UCP este un multiplu, sau submultiplu al frecvenţei oscilatorului de ceas.

În plus, pentru a executa programele stocate în memoria principală, UCP controlează celelalte componente ale sistemului, prin linii de comandă şi stare speciale. De exemplu, UCP controlează direct sau indirect operaţiile de I/O, cum ar fi transferul datelor între dispozitivele de I/O şi memoria principală (MP).

De multe ori operaţiile cu porturile de I/O cer rar intervenţia UCP şi de aceea este mai eficient pentru UCP să ignore dispozitivele de I/O până când acestea cer servicii de la UCP. Asemenea cereri sunt numite **întreruperi**. Dacă apare o cerere de întrerupere, UCP poate suspenda execuţia programului executat pentru a trece temporar controlul la o rutină de tratare a cererii de întrerupere. Un test al prezentei cererilor de întrerupere se face la sfârşitul fiecărui ciclu instrucţiune.

În cadrul unui calculator pot însă să existe şi cereri simultane de acces la un anume dispozitiv sau la o magistrală. Motivul principal îl constituie comunicarea, în general asincronă (în sensul că diferitele dispozitive nu pot fi sincronizate direct de un semnal comun de ceas), dintre procesor şi alte dispozitive. Comunicarea asincronă este dictată de mai multe cauze:

- Există un mare grad de independenţă între componentele sistemului. De exemplu, într-un calculator, UCP şi procesoarele de intrare-ieşire (IOP) pot executa programe diferite, ceea ce înseamnă că cele două componente interacţionează relativ rar şi la momente de timp nepredictibile.
- Viteza de funcţionare a componentelor fizice ale calculatorului variază în limite largi. De exemplu UCP poate lucra la viteze de zeci de ori mai mari faţă de dispozitivele de memorie principală, în timp ce viteza MP poate fi cu câteva ordine de mărime mai mare ca viteza dispozitivelor de I/O.
- Distanţa fizică între componente poate fi prea mare pentru a permite transmisia sincronă a informaţiei între ele.

Operaţiile efectuate într-un ciclu instrucţiune pot fi împărţite în operaţii intermediare de tip: aducere cod instrucţiune (citire memorie), aducere operand (citire memorie sau registru I/O), scriere memorie sau registru I/O, etc. Ciclul ce corespunde desfăşurării în timp a acestor operaţii intermediare va fi numit **ciclu maşină (CM)**. El cuprinde deci suma activităţilor ce rezolvă o operaţie intermediară bine definită, cu o finalitate clară. Primul ciclu maşină al oricărui ciclu

instrucţiune este ciclul de *fetch instruction* (extragere-aducere a codului operaţiei).

Concluzionând putem spune că un ciclu instrucţiune (CI) cuprinde mai multe cicluri maşină, fiecare ciclu maşină desfăşurându-se de-a lungul mai multor stări ale UCP.

Din punctul de vedere al liniilor de legătură cu exteriorul UCP (blocul procesor a fost redesenat în figura 3.3.), de obicei liniile de intrare date (D_{in}) şi ieşire date (D_{out}) se unifică în linii comune de date de intrare-ieşire ce sunt legate la o magistrală (bus) externă bidirecţională (fig. 3.3b).

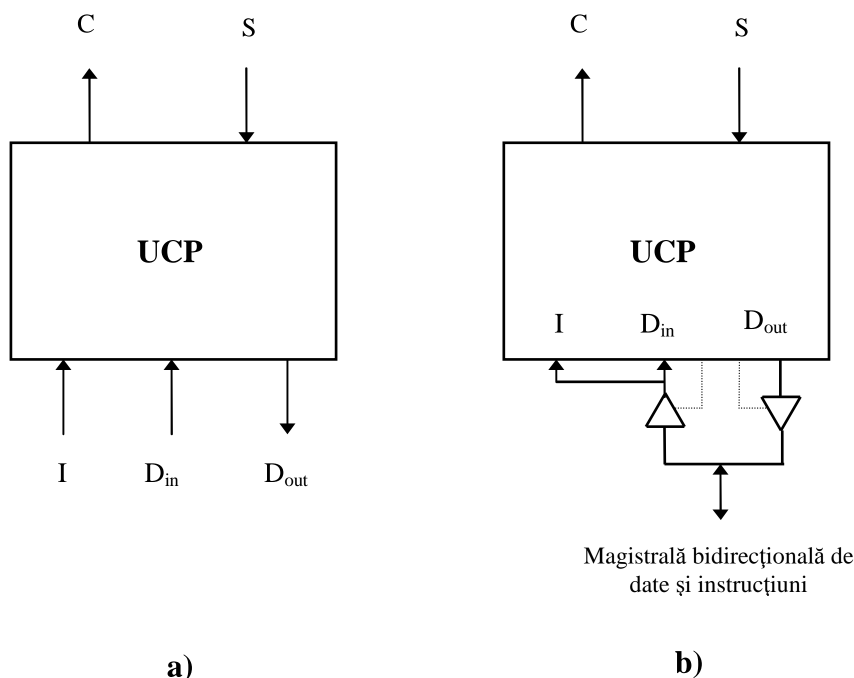


Figura 3.3. Blocul procesor, cu indicarea legăturilor externe. Din - date de intrare, Dout - date de ieşire, C - comenzi externe-inclusiv adrese de MP, I - instrucţiuni, S - semnale de stare din exterior

În acest caz UC trebuie să comande (liniile punctate din figură) şi sensul de transfer a datelor prin intermediul unor circuite tampon (buffer) de interfaţă, pentru a elimina conflictele electrice pe liniile bidirecţionale (circuitele au posibilitatea invalidării - blocării ieşirilor pe baza circuitelor de tip TSL - trei stări logice, starea blocat corespunzând unei înalte impedanţe / HiZ la ieşire).

La majoritatea procesoarelor pe magistrala bidirecţională de date se aduc şi instrucţiunile (I), reducând astfel numărul de conexiuni către exterior. Este clar, că din punctul de vedere al vitezei de funcţionare, o organizare cu magistrale separate pentru date şi instrucţiuni este mai rapidă. Acest ultim mod de organizare corespunde aşa-numitei *arhitecturi Harvard* a procesorului. Arhitecturile Harvard construite în prezent au o magistrală externă unică de date şi instrucţiuni, dar magistrale interne (şi circuite de memorie tampon locale) separate pentru date şi instrucţiuni.

Magistrala unică de date şi instrucţiuni este numită magistrală de date bidirecţională şi ea oferă anumite avantaje sistemului, din punctul de vedere al flexibilităţii modului de lucru cu exteriorul. Câteva din aceste avantajele unei magistrale unice de date şi instrucţiuni sunt:

- se foloseşte o memorie unică pentru stocarea datelor şi instrucţiunilor (memoria principală, externă procesorului). Rezultă simplificarea UCP din punctul de vedere al automatului de adresare a memoriei şi al numărului de registre implicate.
- folosirea unei memorii unice, unde se stochează atât date cât şi instrucţiuni creşte flexibilitatea sistemului, pentru că nu exista restricţii privind adresele de stocare a datelor şi instrucţiunilor în memoria principală.

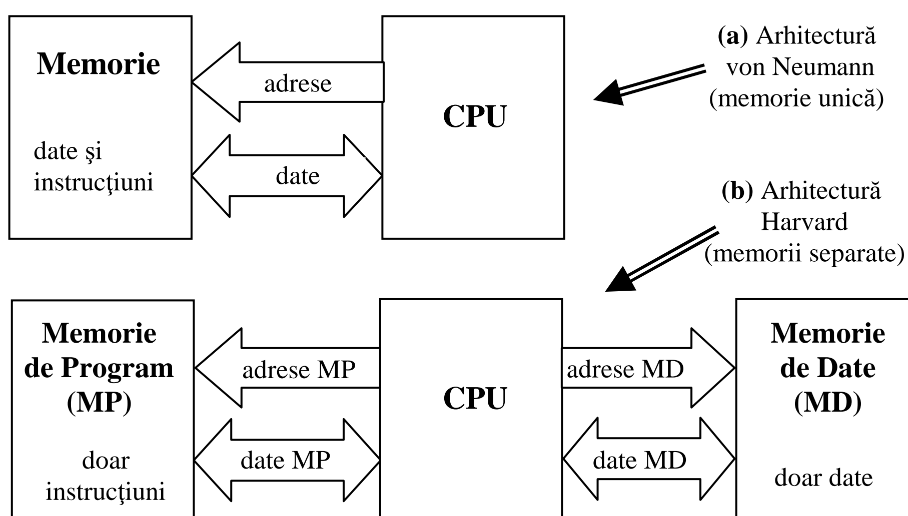


Figura 3.4. Diagrame bloc pentru exemplificarea deosebirilor dintre arhitecturile: (a)von Neumann şi (b) Harvard.

Acest mod de organizare are şi dezavantaje, din punctul de vedere al vitezei de lucru şi din punctul de vedere al unei relative creşteri în complexitate a unităţii de control, care trebuie să interpreteze în mod corect cuvintele citite din memorie ca fiind date, sau instrucţiuni.

În cadrul liniilor notate ca şi comenzi de ieşire (C), în figura 3.3, am inclus şi adresele binare pe care procesorul le trimite către memorie sau circuitele de intrare - ieşire ale calculatorului. Acestea sunt grupate de obicei într-o magistrală separată, numită *magistrală de adrese*.

Deosebirile dintre arhitecturile von-Neumann şi Harvard pentru organizarea memoriei principale şi a magistralelor de acces cu această memorie sunt indicate în schemele bloc din figura 3.4. Schema bloc din figura 3.5. indică faptul că procesorul are circuite de interfaţă (tampoane) pe toate liniile de conectare cu exteriorul. Acestea permit "deconectarea" procesorului de la magistralele externe de date şi adrese şi de asemenea eliberarea unor linii de control externe, astfel încât alte dispozitive active (procesoare) din sistem să poată prelua temporar controlul

magistralelor. Uneori tampoanele de la interfaţa microprocesorului cu exteriorul, au în afara capacităţii de trecere în stare de înaltă impedanţă (HiZ) şi capacitatea de memorare (latch).

Trebuie menţionat un alt rol al tampoanelor la interfaţa microprocesorului cu exteriorul: asigură factorii de încărcare de ieşire (fan-out[1]) şi intrare (fan-in[2]) necesari circuitelor logice, pentru asigurarea nivelurilor corecte de tensiune High şi Low. Circuitele tampon au de asemenea rol de izolare între circuitele interne şi cele externe procesorului şi rolul de a permite controlul unor valori de capacitate parazită limită, pe fiecare linie.

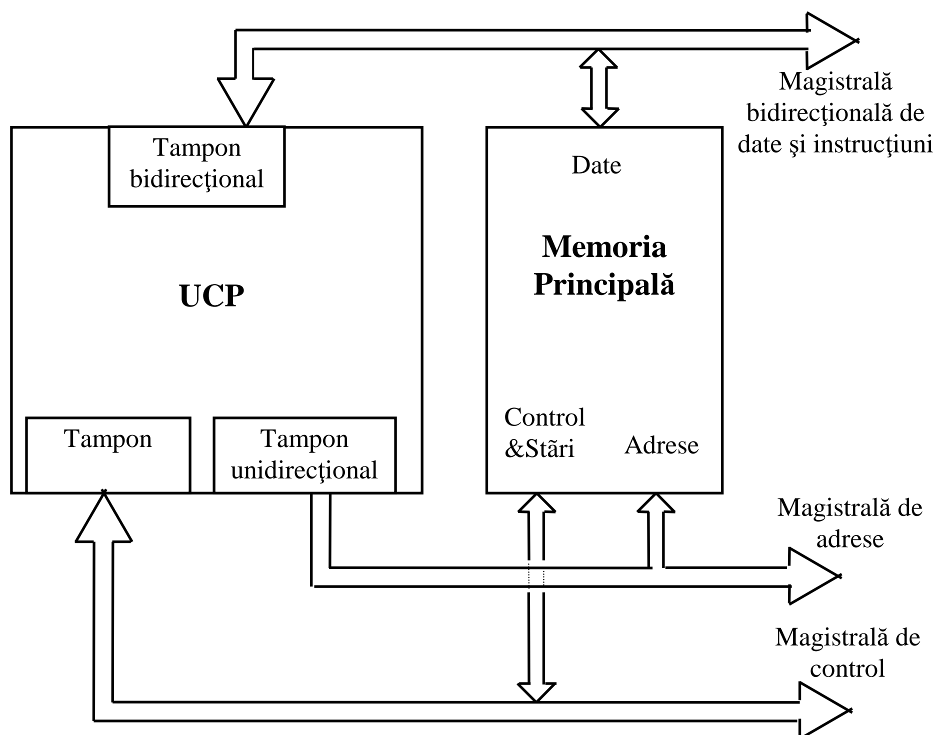


Figura 3.5. Structură de calculator, formată din UCP şi memorie principală, cu indicarea circuitelor tampon la interfaţa procesorului cu exteriorul.

[1] Fan-out = este capacitatea maximă de încărcare pentru ieşirea unui circuit digital. Valoarea sa depinde de curentul maxim ce poate fi generat / absorbit pentru nivel logic 1 / 0 la ieşire şi se exprimă ca numărul maxim de porţi ce poate fi comandat de ieşire, cu păstrarea nivelurilor corecte de tensiune la ieşire.

[2] Fan-in = este capacitatea maximă de încărcare la intrare, definit similar cu fan-out pe baza curentului maxim la intrare, deci număr maxim de porţi conectate la intrare, astfel ca nivelurile de tensiune corespunzătoare lui 1 şi 0 logic să se păstreze.

3.2. Structura unui microprocesor elementar

Cea mai simplă structură de UCP, bazată pe registru acumulator poate fi descrisă ca în figura 3.6. În fiecare dintre unităţile componente ale microprocesorului s-au figurat elementele principale ce permit interpretarea instrucţiunilor şi execuţia acestora.

În Unitatea de prelucrare a datelor s-au inclus Unitatea Aritmetică şi Logică (ALU), registrul Acumulator (AC) şi căile interne de transfer a datelor. În afara acestor elemente pot exista şi registre de memorare temporară, registre folosite de procesor pentru operaţii intermediare, dar inaccesibile programatorului. ALU este în general un circuit combinaţional şi, aşa cum s-a pomenit anterior, realizează operaţii aritmetice sau logice cu operanzii de la intrare. Operaţia este comandată de semnale de control intern provenite la unitatea de control.

Pentru un microprocesor de uz general, în structura unităţii de prelucrare a datelor pe lângă logica combinaţională de prelucrare, decodificare, multiplexare, există de obicei şi un set de registre de uz general, vizibile programatorului şi folosite pentru stocarea temporară a operanzilor şi a rezultatelor. Aceste registre sunt selectate cu ajutorul unor câmpuri binare din cadrul codului fiecărei instrucţiuni. În structura UCP elementare se consideră, pentru moment, doar existenţa registrului *acumulator*. Acumulatorul este prezent în arhitectura majorităţii microprocesoarelor de 8 biţi şi este un registru cu funcţie specială, el fiind atât registru de stocare pentru un operand sursă, cât şi registru de stocare pentru operandul destinaţie, pentru multe din instrucţiunile microprocesorului. În codul acestor instrucţiuni, selecţia acumulatorului se face implicit (deci fără câmp binar de adresare explicită). Folosirea acumulatorului prezintă avantaje din punctul de vedere al simplificării arhitecturii (adresare implicită a acumulatorului pentru multe din instrucţiuni, deci cod mai scurt al acestora), dar prezintă dezavantaje din punctul de vedere al vitezei de execuţie şi al flexibilităţii instrucţiunilor (acumulatorul este un loc îngust, o "ştrangulare", prin care trebuie să treacă fluxul informaţional pentru toate instrucţiunile executate). Dacă se notează cu săgeată sensul de transfer al datelor şi cu *f* operaţia efectuată de ALU, funcţionarea AC poate fi simbolizată prin relaţia:

$$AC \longleftarrow f(AC,RD) \tag{3.1}$$

unde RD este registrul tampon de date, la interfaţa cu magistrala externă de date.

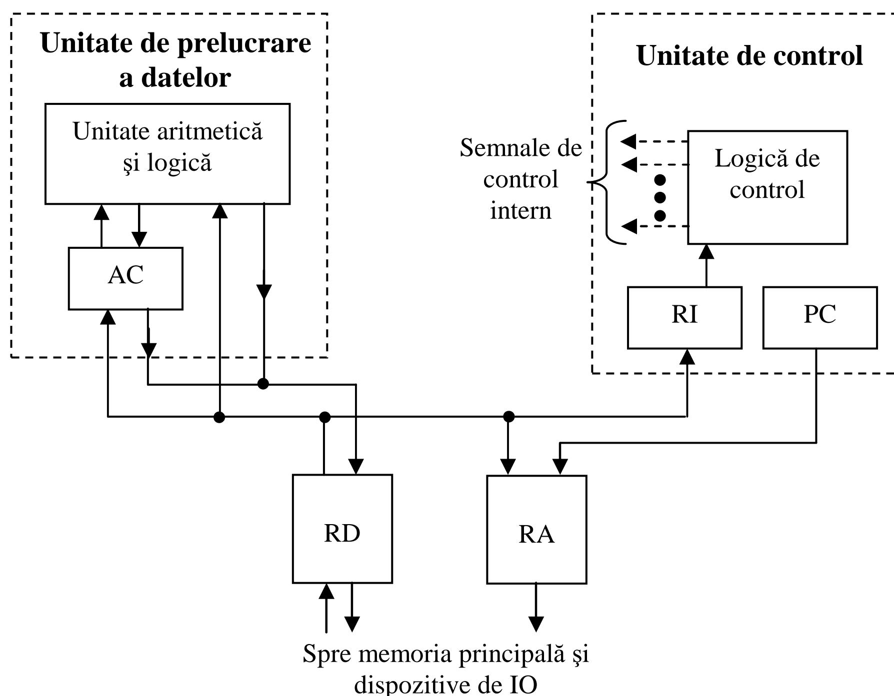


Figura 3.6. O structură simplă de microprocesor de uz general, pe bază de acumulator:

În cadrul unităţii de control s-a simbolizat blocul ce realizează generarea semnalelor de control (logica secvenţială de control) în care se implementează algoritmul de tranziţie al stărilor automatului de control. Succesiunea specifică a stărilor corespunde codului instrucţiunii curente care se stochează într-un registru de instrucţiuni (RI) pe toată durata execuţiei instrucţiunii. Un al doilea registru important al unităţii de control este numit Contor de Program (simbolizat prin PC[3]). Acest registru are capacitatea de autoincrementare. El conţine adresa instrucţiunii următoare din program (sau *informaţie pe baza căreia se construieşte adresa* din memoria principală unde se găseşte instrucţiunea următoare). Ori de câte ori s-a terminat execuţia unei instrucţiuni, conţinutul PC se transmite către registrul tampon de adrese (RA – transparent pentru programator), iar PC se incrementează pentru a pregăti adresa către următoarea instrucţiune. Se consideră că instrucţiunile programului sunt aranjate secvenţial, în adrese succesive ale memoriei principale. Registrul PC, pentru adresarea instrucţiunilor funcţionează în majoritatea timpului ca un contor, cu incrementare, (pentru secvenţe liniare de instrucţiuni). La majoritatea microprocesoarelor de 8 biţi conţinutul acestui registru se transmite pe magistrala de adrese, indicând adresa cuvântului (octet, sau multi-octet) ce trebuie citit din memorie, cuvânt ce face parte din câmpul binar al unei instrucţiuni. După fiecare ciclu de extragere (fetch) a unui cuvânt din memorie, PC se auto-incrementează (PC:=PC+1), pentru a pregăti extragerea următorului

[3] PC = Program Counter (engl.) – Contor de program

cuvânt. Dacă lărgimea magistralei de date a procesorului permite să se aducă simultan mai mult de un cuvânt operaţia poate fi scrisă: PC=PC+N, unde N este numărul de cuvinte citite simultan. La pornirea execuţiei unui program, sau ori de câte ori se face un salt în program conţinutul PC se încarcă automat cu o valoare de adresă, urmând ca în continuare să funcţioneze ca şi contor de instrucţiuni.

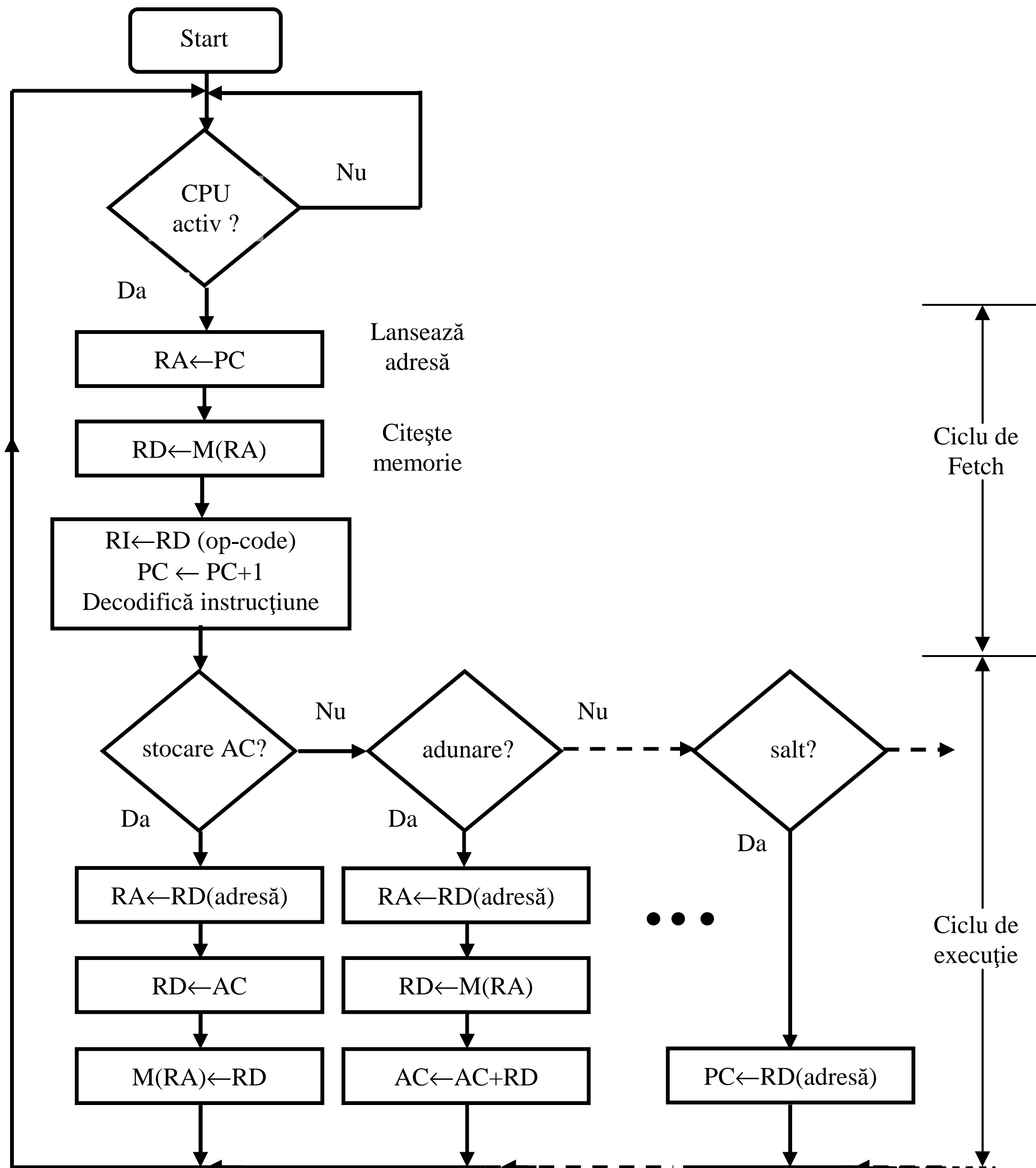


Figura 3.7. Organigramă privind funcţionarea UCP elementare din figura 3.6

În figura 3.7. se indică un exemplu de secvenţă a stărilor (microoperaţii) pentru aducerea (fetch) şi execuţia instrucţiunilor pentru structura elementară a UCP din figura 3.6. Fiecărei stări (simbolizată cu dreptunghi în figură) îi sunt asociate semnale de control intern UCP, pentru

realizarea operaţiilor simbolizate pe figură. Orice transfer este indicat cu săgeată, pe poziţia din stânga fiind indicată destinaţia, iar pe cea din dreapta sursa. Prin notaţia M(RA) s-a indicat o adresare la memoria principală, adresa fiind cea conţinută în registrul de adrese RA. La unele dintre operaţii s-a folosit notaţia RD (*câmp*) care semnifică faptul că se citeşte doar *câmpul* de adresă sau cod al operaţiei (op-code) din formatul binar al instrucţiunii aduse în registrul RD.

Secvenţa din figura 3.7 este doar teoretică, funcţionarea procesorului elementar nefiind optimizată din punctul de vedere al minimizării stărilor maşinii cu algoritm de stare. De exemplu blocurile de comparare, pentru testarea codului instrucţiunii (op-code[4]) funcţionează secvenţial şi ar însemna că pentru ultima instrucţiune trecută în lista de comparări s-ar parcurge toate operaţiile de comparare. Operaţia de tip *switch* (sau *case*) în limbaj de nivel înalt poate fi implementată în hardware, pentru saltul direct la ramura corespunzătoare execuţiei instrucţiunii.

La structura elementară de UCP din figura 3.6 se pot aduce câteva tipuri de îmbunătăţiri pentru a creşte performanţele şi flexibilitatea în ceea ce priveşte perfomanţele microprocesorului. O parte se vor discuta în acest paragraf, altele în paragrafele şi capitolele următoare.

Principalele tipuri de extindere în organizarea microprocesorului elementar sunt următoarele:

1. Introducerea unor registre speciale pentru stocarea *stării procesorului* în care să se salveze anumiţi indicatori de condiţii şi de control. Adesea acest tip de registru este numit registru pentru cuvântul de stare al programului[5]. Indicatorii de condiţii[6] pot fi testaţi de instrucţiuni de salt condiţionat şi astfel pot modifica succesiunea instrucţiunilor executate de procesor în cadrul unui program. Indicatorii sunt importanţi pentru că permit implementarea instrucţiunilor de testare şi decizie în limbaj de nivel înalt. Indicatorii de control au un alt rol. Aşa cum le spune şi numele ei sunt destinaţi pentru a modifica modul de funcţionare al unităţii de control, sau al unor registre de adresare ale acestei unităţi, în funcţie de valoarea introdusă în indicatori. Cei mai simpli dintre aceşti indicatori pot indica de exemplu: mod de lucru supervizor / utilizator, mod de lucru cu auto-incrementarea sau auto-decrementarea unor registre index, activare / blocare a cererilor de întrerupere mascabile, informaţii privind drepturile de acces la unele zone de memorie etc. În general indicatorii pot fi setaţi prin instrucţiuni recunoscute de procesor.
2. Introducerea unor *registre suplimentare de adresare*. Adesea se folosesc şi registre pentru moduri complexe de adresare (indexată, bazată, cu bază şi index).
3. Introducerea unei *bănci de registre* de uz general vizibile programatorilor (sau compilatorului), pentru stocarea temporară şi manipularea datelor.
4. Adăugarea unei unităţi aritmetice pentru numere în *virgulă mobilă* (numere reale), în plus

[4] op-code (engl.) este prescurtarea de la operation code (codul operaţiei)
[5] PSW - program status word
[6] flags în limba engleză

faţă de unitatea ALU pe întregi.

5. Introducerea de suport hardware (logică plus registre de adresare specializate) pentru implementarea *memoriei stivă*. Memoria stivă este utilă în primul rând în operaţiile de apelare şi revenire din proceduri, dar are şi alte utilizări. Uneori memoria stivă este implementată hardware, prin registre construite în cadrul UCP, dar cel mai adesea ea este organizată software (stocare în memoria principală şi administrare printr-un registru indicator al vârfului stivei – SP = stack pointer).

O structură de UCP îmbunătăţită cu o parte din extensiile pomenite mai sus se prezintă în figura 3.8.

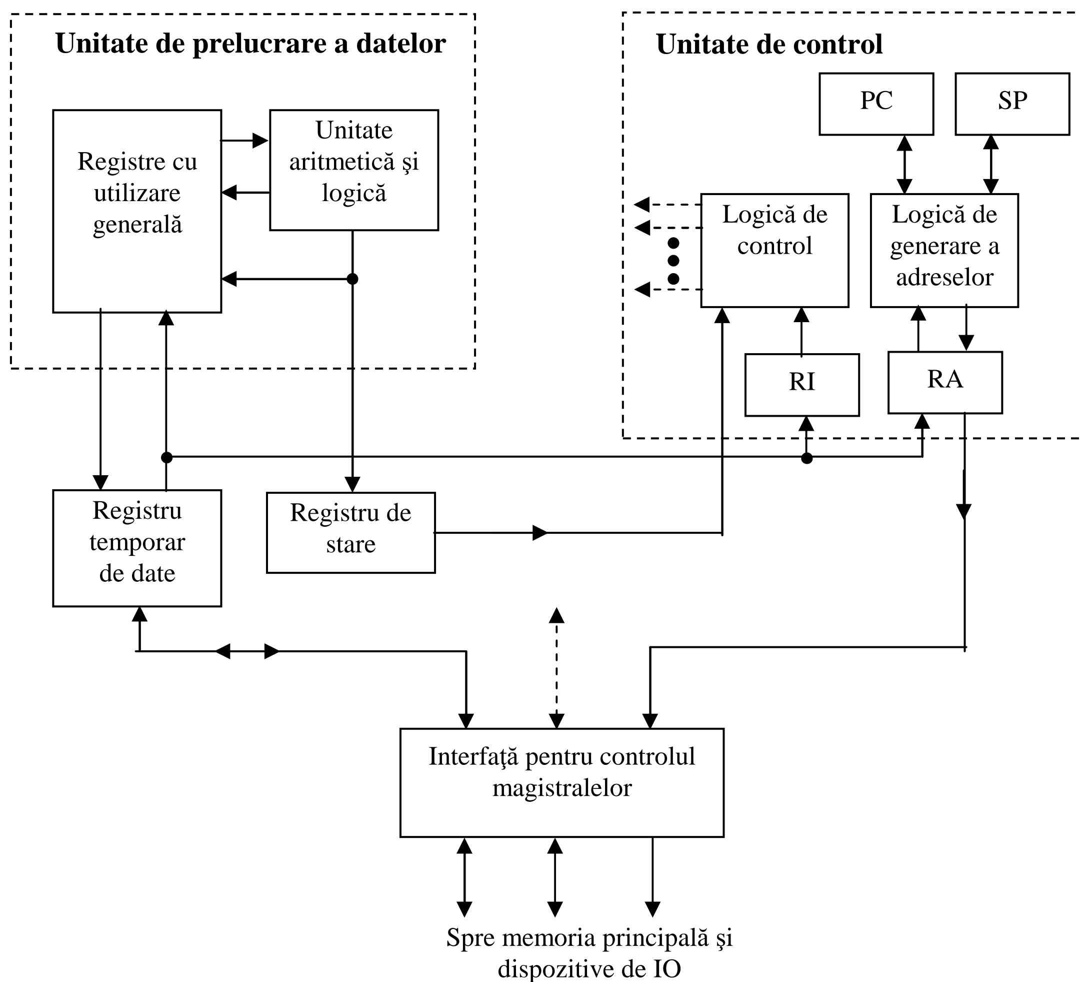


Figura 3.8. UCP tipică pentru organizarea pe bază de registre de uz general.

În unitatea de prelucrare a datelor s-a introdus o bancă de registre de uz general, adresabile individual de către program. În acelaşi bloc se consideră a fi inclusă şi logica de selecţie (adresare) şi de multiplexare a ieşirilor către căile interne de date. Acestea sunt transparente pentru utilizator (deci inaccesibile explicit prin instrucţiuni). S-a figurat de asemenea un registru

pentru stocarea indicatorilor de condiţii (registru de stare). Indicatorii sunt setaţi în funcţie de rezultatul operaţiilor efectuate de ALU (unitatea aritmetică şi logică) şi sunt testaţi de unitatea de control, reacţie simbolizată în figură prin liniile de intrare în blocul de logică de control. Toate circuitele tampon cu magistralele externe au fost introduse într-un bloc numit "interfaţă pentru controlul magistralelor". Pe calea de date apare de asemenea un registru temporar pentru stocarea operanzilor şi a rezultatelor care se citesc / stochează în memoria principală (registrul temporar nu este vizibil de instrucţiuni – programator). Pentru că se consideră un procesor cu magistrală bidirecţională de date şi instrucţiuni registrul de instrucţiuni recepţionează codul instrucţiunii curente din acelaşi registru temporar de date. În cazul în care instrucţiunile conţin adrese (sau informaţii de adresare) pentru operanzi din memoria principală, acestea se pot citi în registrul de adrese (RA) care poate citi registrul temporar de date.

Pentru automatul de adresare al unităţii de control s-a prevăzut logica necesară pentru generarea adreselor din memoria principală. Informaţia de adresare poate proveni de la contorul de program (PC) pentru instrucţiuni, sau de la registrul indicator de stivă (SP[7]) în cazul când se efectuează operaţii cu memoria stivă. Registrul SP indică întotdeauna adresa vârfului stivei, memoria stivă lucrând pe principiul LIFO[8]. Vârful stivei conţine ultimul articol (cuvânt) înscris în stivă.

Nu s-au desenat căile interne de control, dar s-a indicat prin linie punctată existenţa semnalelor de control şi sincronizare cu exteriorul UCP, care prin interfaţa cu magistralele vor forma magistrala de control externă procesorului.

3.3. Alte registre interne semnificative pentru UCP

Aşa cum s-a spus în paragraful anterior, în urma efectuării operaţiilor aritmetice şi logice de ALU se setează, conform cu rezultatul, biţii unui registru special, numit *registru de stare*, sau *registru al indicatorilor de condiţii*. Aceşti indicatori (Flags - fanioane) sunt numiţi "*de condiţii*" pentru că ei reflectă anumite stări ale rezultatului operaţiilor aritmetico-logice, fiind testaţi de către Unitatea de Control, care poate astfel efectua devierea secvenţei programului prin aşa-numitele salturi condiţionate. Indicatorii de condiţii (stare) sunt modificaţi ca urmare a execuţiei instrucţiunilor aritmetice, logice sau prin execuţia unor instrucţiuni de control, a căror singură funcţie este modificarea indicatorilor. Conţinutul lor, în general, nu poate fi citit direct, ci doar testat indirect prin instrucţiuni de salt condiţionat ce transferă controlul programului la diferite ramuri, în funcţie de valoarea indicatorului testat.

Vom descrie în continuare un *set tipic de indicatori* de condiţii şi apoi vom da câteva

[7] SP – Stack Pointer (engl.) = Indicator sau pointer de stivă.

exemple pentru microprocesoare existente pe piaţă:

- ***Semn*** (**S**) - Indicatorul de semn ia valoarea 1 logic dacă rezultatul unei operaţii cu numere cu semn este un număr negativ. În cazul în care se lucrează cu numere fără semn, de obicei, indicatorul va copia valoarea logică a celui mai semnificativ bit (MSb) al rezultatului (dar atunci nu mai are interpretarea de indicator de semn).
- ***Zero*** (**Z**) - acest indicator ia valoarea 1 logic dacă toţi biţii rezultatului unei operaţii aritmetice sau logice sunt zero; altfel Z = 0.
- ***Paritate*** (**P**) - indică paritatea rezultatului în convenţia de imparitate (suma modulo 2 a tuturor biţilor rezultatului, inclusiv a bitului P, este întotdeauna 1). Acest indicator este deci setat (poziţionat pe 1 logic) dacă rezultatul conţine un număr par de biţi de valoare "1". Dacă numărul este impar, atunci P=0. Se setează, în general, în urma operaţiilor logice.
- ***Depăşire*** (**V**, oVerflow)) a capacităţii de reprezentare. Prin depăşire a capacităţii de reprezentare se înţelege faptul că rezultatul unei operaţii aritmetice nu poate fi reprezentat pe n biţi ca număr cu semn. Indicatorul V se poziţionează la 1 logic în urma operaţiilor aritmetice, indicând depăşirea capacităţii de reprezentare în cazul operării cu numere cu semn, depăşire ce se manifestă prin alterarea bitului de semn (deci V=1 indică rezultat eronat). Valoarea indicatorului poate fi descrisă printr-o operaţie "SAU EXCLUSIV" între transporturile de la rangul n-2 (MSb) şi respectiv n-1 (poziţie bit de semn), pentru numere cu semn reprezentate pe n biţi, în cod complementar (complement faţă de doi).
- ***Transport*** (**CY**, Carry) - indicator de transport. Se poziţionează pe 1 logic dacă operaţia anterioară a produs un transport sau un împrumut la/de la rangul n-1.
- ***Transport auxiliar*** (**AC**, Auxiliary Carry) - Se poziţionează pe 1 dacă o operaţie de adunare în cod NBCD a produs transport de la o cifră zecimală codificată în NBCD către cifra zecimală de rang superior (transport de la o tetradă binară inferioară către o tetradă binară superioară). În cazul operaţiilor de scădere în NBCD acest indicator indică împrumutul.
- ***Adunare sau scădere în NBCD*** (**N**) - apare doar la unele microprocesoare. Prin valoarea 1 arată că operaţia anterioara în NBCD a fost o scădere. La microprocesoarele care folosesc acest tip de indicator există o singură instrucţiune de corecţie zecimală, folosită atât pentru adunare cât şi pentru scădere, spre deosebire de microprocesoarele ce nu au indicator de tipul N şi care au instrucţiuni de corecţie zecimală separate pentru adunare şi scădere. Indicatorii specifici operaţiilor în NBCD (AC şi N) nu pot fi testaţi prin program, ei fiind destinaţi doar circuitului de corecţie zecimală inclus în unitatea aritmetică şi logică.

[8] LIFO – Last In First Out (engl.) – Ultimul intrat, primul ieşit din memoria stivă

Exemple de registre de indicatori, la diferite microprocesoare:

Tabelul 3.1.

Tip	**Poziţia binară în registrul de indicatori**															
procesor	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
I8080									**S**	**Z**	*	**AC**	*	**P**	*	**CY**
Z80									**S**	**Z**	*	**H**	*	**P/V**	**N**	**C**
MC6800									*	*	**AC**	**I**	**S**	**Z**	**V**	**C**
I8086	*	*	*	*	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	*	**AF**	*	**PF**	*	**CF**
MC68000	**T**	*	**S**	*	*	**I2**	**I1**	**I0**	*	*	*	**X**	**N**	**Z**	**V**	**C**

Peste tot, biţii liberi (care nu reprezintă indicatori) sunt notaţi cu steluţă (*). Pentru unele din exemplele de mai sus se observă că în registrul de stare sunt conţinuţi nu numai *indicatori de condiţii* ci şi alte tipuri de indicatori; aceştia sunt *indicatori de control* ai procesorului, influenţând funcţionarea acestuia.

La microprocesorul Z80 există un indicator comun paritate / depăşire (P/V), iar indicatorul de transport auxiliar e notat cu H. Indicatorul N este folosit de circuitul de corecţie zecimală (la operaţii în NBCD) el indicând scădere (N=1) sau adunare (N=0) la operaţia anterioară în NBCD. La procesorul MC6800 al firmei Motorola în registrul de indicatori apare şi un indicator de control al întreruperilor mascabile (permise pentru I=1). La fel, în cazul microprocesorului Intel 8086 registrul de indicatori conţine şi 3 indicatori de control (controlează unele operaţii ale UCP). Notaţia indicatorilor are adăugată şi litera F (de la flag), AF fiind indicatorul de transport auxiliar, iar OF cel de depăşire. În privinţa indicatorilor de control la Intel 8086 menţionăm următoarele:

- indicatorul de întrerupere (Interrupt Flag = IF) determină modul în care procesorul 8086 reacţionează la cererile de întrerupere mascabile aplicate pe pinul INT. Valoarea IF = 1 arată că cererile de întrerupere sunt validate. Indicatorul poate fi poziţionat pe 1 sau şters prin instrucţiuni ale programului.
- indicatorul de direcţie (Direction Flag = DF) este utilizat în operaţiile cu şiruri, când este necesar ca şirul să fie definit prin adresa de bază şi prin sensul descrescător (DF=1), sau crescător (DF=0) al adreselor index (indice) care definesc fiecare element din şir faţă de o adresă de bază;
- indicatorul de mod "pas cu pas" (Trap Flag = TF) permite implementarea unui mod de funcţionare util în depanarea programelor. Astfel, dacă TF = 1, după execuţia fiecărei instrucţiuni se iniţiază o întrerupere care determină transferul controlului într-o zonă de memorie definită de utilizator, unde sub acţiunea unui program adecvat, se poate face analiza stării interne a procesorului. Acest indicator este folosit de programele

depanatoare[9].

La MC68000, octetul mai semnificativ al registrului de stare este octetul de control al sistemului (System Byte conţine **T**race mode, **S**upervisor state, Interrupt mask: **I2,I1,I0**), iar octetul mai puţin semnificativ (b_7-b_0) este octetul indicatorilor pentru programul utilizator (User Byte care conţine indicatorii e**X**tend, **N**egative, **Z**ero, o**V**erflow, **C**arry). În funcţie de valoarea lui S există indicator de stivă (SP) separat pentru supervizor şi pentru utilizator; dacă S=1 se foloseşte SSP (=A7' Supervisor stack pointer) iar dacă S=0 se foloseşte USP (=A7, User stack pointer). SSP şi implicit stiva sunt protejate în acest fel faţă de modificările utilizatorilor. Masca de întrerupere indică nivelul de întrerupere curent. Orice întrerupere cu nivel mai mare decât nivelul măştii fiind recunoscută. Există 14 combinaţii ale indicatorilor C, Z, N şi V ce pot fi testate pentru salturi condiţionale. Indicatorul X este rezervat exclusiv pentru aritmetica în precizie multiplă.

Pentru I8080, al firmei Intel, notaţia indicatorilor corespunde cu cea din prezentarea setului tipic de indicatori.

Cu privire la indicatorul de depăşire (V – oVerflow) remarcăm următoarele aspecte importante. Dacă rezultatul unei operaţii aritmetice cu numere cu semn reprezentate pe n biţi este prea mare ca să poată fi reprezentat pe n biţi se spune că se produce depăşire a capacităţii de reprezentare. Aceste depăşiri trebuie determinate pentru că adesea ele sunt o indicaţie a erorilor de programare. Fie exemplul următor de adunare a două numere binare cu semn, reprezentate în cod complementar:

$$\begin{array}{l} x_{n-1}x_{n-2}\cdots x_0 \ + \\ \underline{y_{n-1}y_{n-2}\cdots y_0} \\ z_{n-1}z_{n-2}\cdots z_0 \end{array} \tag{3.2}$$

Pentru fiecare rang *i* , prin adunarea biţilor x_i, y_i şi c_i se generează un bit de sumă z_i şi un bit de transport (carry bit) c_{i+1} către rangul următor, unde z_i şi c_i se consideră pentru $0 \le i \le n-1$, c_0=0, iar valorile lor se calculează conform relaţiilor unui sumator binar complet:

$$z_i = x_i \oplus y_i \oplus c_i \tag{3.3}$$

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i \tag{3.4}$$

Notăm cu V variabila binară care indică depăşirea (oVerflow) atunci când V = 1. Tabelul 3.2. indică modul de determinare al bitului de semn z_{n-1} şi al variabilei V în funcţie de biţii de semn ai celor doi operanzi, x_{n-1}, y_{n-1}, şi bitul de transport c_{n-1} de la rangul n-2, care este MSb[10].

Rezultă că indicatorul de depăşire poate fi definit prin ecuaţia:

[9] Debugger (engl.) = Program depanator
[10] MSb – Most significant bit (engl.) = cel mai semnificativ bit

$$V = \overline{x_{n-1}}\ \overline{y_{n-1}}\ c_{n-1} + x_{n-1}\ y_{n-1}\ \overline{c_{n-1}} \qquad (3.5)$$

Dacă din tabelul 3.2 se elimină liniile pentru care V = 1 va rezulta definirea corectă, conform ecuaţiei (3.3)) şi pentru bitul de semn al sumei:

$$z_{n-1} = x_{n-1} \oplus y_{n-1} \oplus c_{n-1} \qquad (3.6)$$

Acest lucru este deosebit de important, pentru că arată că la operaţiile în cod complement faţă de 2 biţii de semn ai operanzilor pot fi trataţi în acelaşi mod ca şi biţii ce indică mărimea numărului.

Tabelul 3.2.

Intrare			**Ieşire**		**Observaţii**
x_{n-1}	y_{n-1}	c_{n-1}	z_{n-1}	V	
0	**0**	**0**	**0**	**0**	
0	**0**	**1**	**0**	**1**	Se adună două numere pozitive, deci suma trebuie să fie pozitivă
0	**1**	**0**	**1**	**0**	
0	**1**	**1**	**0**	**0**	
1	**0**	**0**	**1**	**0**	
1	**0**	**1**	**0**	**0**	
1	**1**	**0**	**1**	**1**	Se adună două numere negative, deci suma trebuie să fie negativă
1	**1**	**1**	**1**	**0**	

3.4. Exemple privind operaţiile UCP şi modul de setare al indicatorilor de condiţii

1. Dacă se adună, sau se scad, două numere reprezentate în cod NBCD, corecţia zecimală (adunarea, respectiv scăderea codului lui 6) se aplică fie dacă rezultatul nu este un cod NBCD, fie dacă se produce transport de la bitul cel mai semnificativ al codului (AC=1):

a)

```
zecimal         binar în cod NBCD

   07+          0000   0111+
   05           0000   0101
   ---          ---------------
   12           0000   1100 + (>1001)        S=0, Z=0, P=1, V=0, C=0, AC=0
                       0110  corecţie zecimală
                ---------------
                0001   0010
```

Se observă că pentru codul cifrei zecimale de pondere mai mică (tetradă inferioară) s-a obţinut un cod nepermis (1100 care este mai mare decât codul lui 9, ce are valoare maximă); de aceea se aplică corecţia zecimală, prin adunarea codului cifrei 6. Se observă de asemenea ca indicatorii de condiţii sunt setaţi, chiar dacă unii din ei nu au semnificaţie pentru operaţia efectuată.

b)

```
zecimal         binar în cod NBCD
12-             0001   0010-
08                     1000
--              ----------------
04              0000   1010 -            S=0, Z=0, P=1, V=0, C=0, AC=1
                       0110   corecţie zecimală
                ---------------
                0000   0100
```

Aici corecţia zecimală se aplică unei operaţii de scădere a două numere zecimale codificate în NBCD şi de aceea se scade codul lui 6 din cauza apariţiei împrumutului (AC=1).

2. Dacă se adună două numere binare cu lungimea de un octet, se scriu mai jos, pe prima coloana reprezentările în zecimal, numerele fiind interpretate ca numere cu semn (corespunzând la valoarea binară în cod complementar), pe a doua coloană zecimal, numerele fiind interpretate ca numere fără semn, pe a treia în binar şi pe ultima coloană valorile indicatorilor de condiţii:

a)

zecimal (cu semn)	zecimal (fără semn)	binar
+ 8 +	8 +	0000 1000 +
- 128	128	1000 0000
----	----	-------------
- 120	136	1000 1000

S=1, Z=0, P=1, V=0, C=0, AC=0

Dacă numerele sunt considerate cu semn, rezultatul obţinut este un număr cu semn, corect, pentru că nu s-a produs depăşire a capacităţii de reprezentare (V=0). De asemenea rezultatul este corect şi pentru numere fără semn, căci CY=0. La aceasta operaţie AC nu are nici o semnificaţie. Indicatorul de paritate P=1 indică un rezultat cu număr par de biţi 1.

b)

zecimal (cu semn)	zecimal (fără semn)	binar
- 128 +	128 +	1000 0000 +
+ 127	127	0111 1111
----	----	-------------
- 1	255	1111 1111

S=1, Z=0, P=1, V=0, CY=0, AC=0

Indicatorii arată rezultat corect pe 8 biţi, atât pentru numere cu semn cât şi pentru numere fără semn.

c)

zecimal binar
(cu semn) (fără semn)

zecimal (cu semn)	zecimal (fără semn)	binar
- 72 +	184 +	1011 1000 +
+ 104	104	0110 1000
----	----	-------------
+ 32	288	0010 0000

S=0, Z=0, P=0, V=0, CY=1, AC=1

Aici V=0 indică că pentru operaţie în cod complementar (cu semn) nu s-a produs depăşire a capacităţii de reprezentare, deci rezultatul este corect (+32 în zecimal). Dacă numerele binare sunt considerate fără semn, rezultatul pe 8 biţi nu este corect pentru că indicatorul CY=1, deci s-a produs o depăşire a capacităţii de reprezentare pe 8 biţi (rezultatul fără semn este mai mare cu 32 decât 255, valoarea maximă reprezentabilă). Din nou AC=1 nu are nici o semnificaţie aici.

Din exemplele anterioare se observă importanţa definirii corecte a operanzilor cu care lucrează instrucţiunile procesorului. Interpretarea corectă a numerelor, într-un anumit cod intern (binar cu semn, fără semn, NBCD, ASCII, etc.) se face de către programul compilator, sau de către programator, dacă se lucrează în limbaj de asamblare.

În ceea ce priveşte registrele suplimentare introduse în unitatea de adresare, exemplificăm pe scurt următoarele tipuri de registre:

- *Registre index*: Registrele index realizează modificarea unei adrese de operand. În unele cazuri funcţiile aritmetice ale registrelor index, pot fi la fel de generale ca şi al registrelor aritmetice (adunări, scăderi, înmulţiri, inversări ale biţilor conţinuţi pentru realizarea unor funcţii speciale), iar în altele, aceste funcţii sunt limitate sever. Microprocesoarele simple nu au registre specializate pentru adresarea indexată, ele folosind pentru acest scop registrele de uz general.
- *Registre (de adresă) de bază*: Aceste registre sunt în mod obişnuit folosite în calculatoare pentru tehnicile de adresare ce permit adresarea unui spaţiu de memorie mai mare, utilizând un număr limitat de biţi pentru a reprezenta adresa operandului în instrucţiune.
- *Registre de protecţie a memoriei (sau registre limită)* se folosesc în cadrul unităţilor de adresare din calculatoarele destinate să faciliteze utilizarea unor sisteme de operare de tip multi-tasking, multi-programming. Pentru că la acestea există mai mult decât un program rezident (sistemul de operare plus programele utilizator), trebuie prevăzut un anumit mod de control pentru a preveni accesele întâmplătoare sau intenţionate la locaţii de memorie ce nu aparţin programului utilizator. Într-un sistem cu uni-programare (cum este şi MS-DOS), este de obicei suficient să se interzică programului utilizator să acceseze locaţiile sistemului de operare, situat la adresele mici din memoria principală.

3.5. Semnale la interfaţa UCP cu exteriorul

Semnalele de la interfaţa Unităţii Centrale de Procesare (UCP) cu exteriorul pot fi grupate, funcţional, în 3 tipuri de magistrale:

1. magistrala de adrese
2. magistrala de date
3. magistrala de control

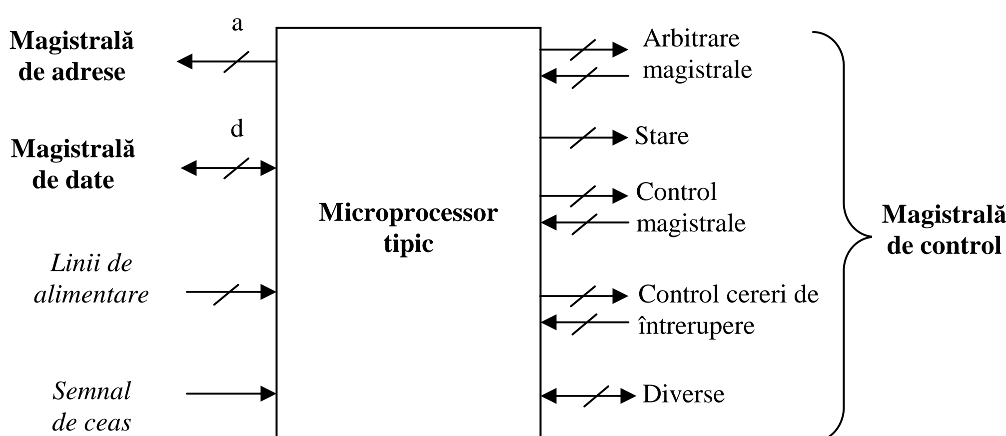


Figura 3.9. Semnale tipice la interfaţa unui microprocesor cu exteriorul

3.5.1. Magistrala de date

Lărgimea magistralei de date este de obicei multiplu de octet (d = 8, 16, 32, 64...). Cele *d* linii ale magistralei de date au posibilitatea de transmitere bidirecţională a informaţiilor (intrare sau ieşire din UCP), cu posibilitatea de trecere în starea de înaltă impedanţă (HiZ). Pentru economie de pini, unele microprocesoare multiplexează în timp liniile magistralei de date, astfel că în primul ciclu maşină al fiecărei instrucţiuni pe magistrala de date se pot transmite informaţii de adresă sau informaţii de control. În acest caz, în prima parte a ciclului maşină, pe pinii multiplexaţi, se generează semnalele de adresă sau control, însoţite de un semnal indicator pe magistrala de control care serveşte pentru memorarea informaţiei în registre externe procesorului. De exemplu la procesoarele Intel 8085 şi 80x86 pini ai magistralei de date sunt multiplexaţi pentru a se putea transmite şi informaţie de adresă. Semnalul de control care comandă stocarea adresei într-un registru extern este numit ALE (Address Latch Enable). Apoi,

pentru tot restul ciclului instrucţiune liniile magistralei de date transferă date propriu-zise sau instrucţiuni. De obicei dimensiunea magistralei de date este folosită, în limbajul curent, în denumirea microprocesorului: "microprocesor pe 32 de biţi" indică că magistrala sa de date are lărgimea de 32 de biţi.

3.5.2. Magistrala de adrese

Această magistrală cuprinde linii de adresă, ce transmit doar semnale de ieşire din microprocesor, fiind deci o magistrală *unidirecţională*. Ieşirile UCP spre această magistrală externă de adrese pot trece în starea de înaltă impedanţă (HiZ) la o cerere externă, când UCP cedează controlul magistralelor către alt dispozitiv. Liniile de pe această magistrală se folosesc pentru adresarea locaţiilor de memorie şi a porturilor de intrare-ieşire. De obicei valorile binare cele mai semnificative (cu ponderea binară maximă) din adresă se folosesc pentru selectarea blocurilor de memorie (ca în figura 3.10)

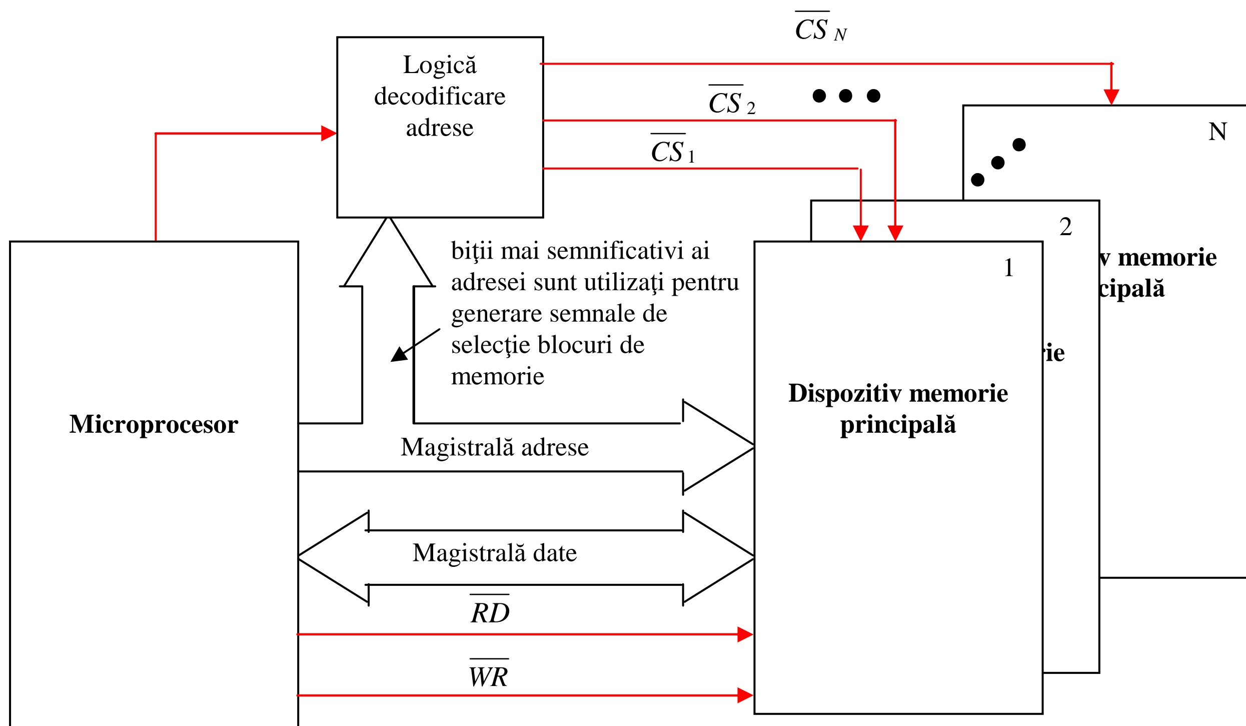


Figura 3.10. Un exemplu de selecţie a blocurilor de memorie prin semnale generate cu ajutorul biţilor mai semnificativi din adresa lansată de microprocesor.

Se utilizează trei moduri de obţinere a semnalelor de selecţie pe baza informaţiei de adresare:

a. adresare liniară;
b. adresare complet decodificată;
c. adresare combinată - combinaţie a primelor două moduri

a. Adresare liniară

Utilizată doar pentru sistemele mici, unde se utilizează doar o parte din spaţiul total de adresare. În aceste cazuri linii ale magistralei de adrese pot fi utilizate împreună cu semnale de control pentru a selecta direct blocuri de memorie. Pentru exemplificare, se presupune că pentru un microprocesor numărul de biţi ai adresei este a = 16, deci spaţiul maxim de adresare este de 2^{16} locaţii de memorie (64 k locaţii). Presupunem că un sistem simplu foloseşte doar 12 k de memorie, format din 3 blocuri de memorie a câte 4 k. Fiecare bloc poate fi adresat cu 12 biţi de adresă (2^{12} = 4096 = 4 k).

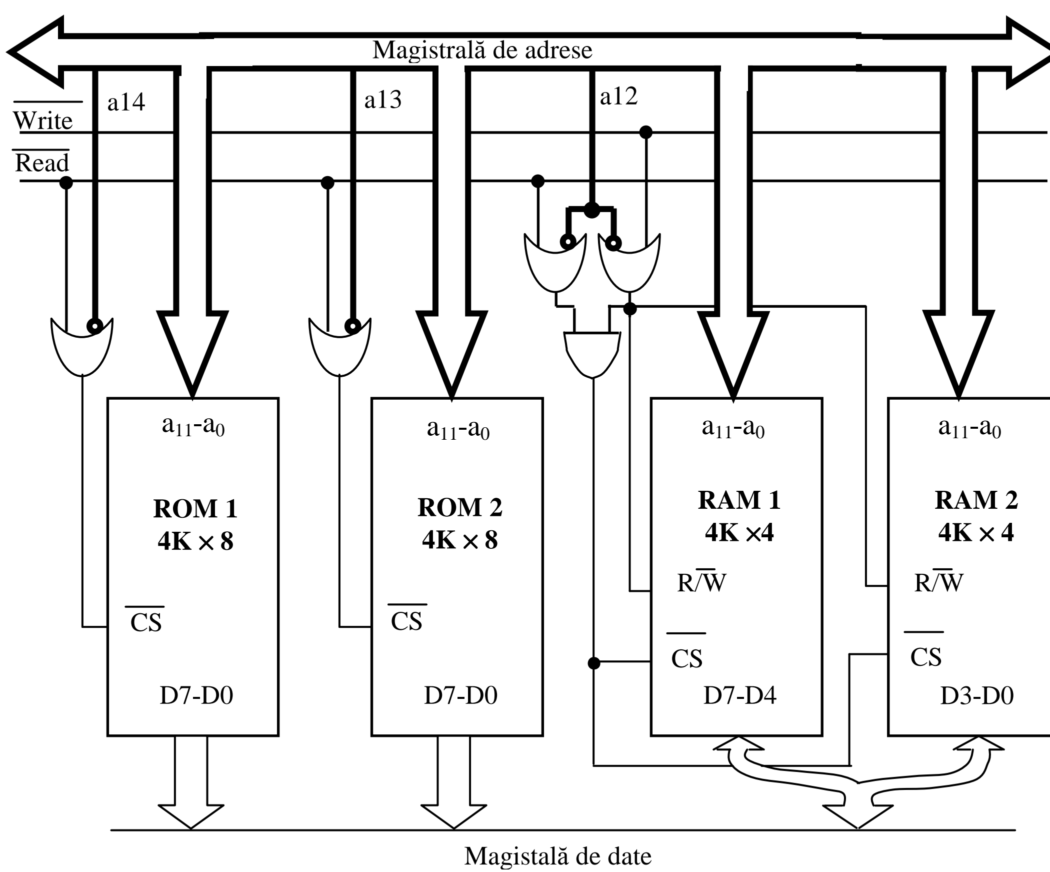


Figura 3.11. Un exemplu de selecţie liniară pentru o memorie de 12k×8, din care 8k×8 memorie ROM

Dacă considerăm cei 12 biţi mai puţini semnificativi pentru adresarea în cadrul blocurilor, ceilalţi 4 biţi din magistrala de adrese pot fi folosiţi pentru selecţia blocurilor de memorie. O variantă a acestui mod de selecţie liniară se prezintă în figura 3.11, în care s-a considerat că există două blocuri de ROM[11] a câte 4 kilo-octeţi (4 kB) şi un bloc de RAM[12].

Blocul de memorie cu citire - scriere, notat RAM 4k×8 este format în exemplul din figură din două circuite RAM cu capacitatea 4K×4 biţi. Pentru selecţia liniară se folosesc biţii a_{14}, a_{13} şi a_{12} de pe magistrala de adrese, împreună cu semnalele de control de citire (Read) şi scriere (Write), iar biţii a_{11} - a_0 sunt folosiţi pentru adresarea locaţiilor interne circuitelor de memorie. Semnalele de control de citire şi scriere sunt active la nivel logic JOS (de aceea au bară deasupra).

Pentru a înţelege logica de selecţie (intrările $\overline{CS}$[13] - selecţie circuit) pentru circuitele de memorie se indică mai jos ecuaţiile care definesc porţile logice.

$$\begin{aligned} \mathrm{CS} &= \mathrm{RD} \cdot \mathrm{A14} \\ \overline{\mathrm{CS}} &= \overline{\mathrm{RD} \cdot \mathrm{A14}} \\ \overline{\mathrm{CS}} &= \overline{\mathrm{RD}} + \overline{\mathrm{A14}} \end{aligned} \tag{3.7}$$

$$\mathrm{R}/\overline{\mathrm{W}} = \overline{\mathrm{WR} \cdot \mathrm{A14}} = \overline{\mathrm{WR}} + \overline{\mathrm{A14}} \tag{3.8}$$

$$\begin{aligned} \overline{\mathrm{CS}} &= \overline{\mathrm{A12} \cdot (\mathrm{RD} + \mathrm{WR})} = \overline{\mathrm{A12} \cdot \mathrm{RD} + \mathrm{A12} \cdot \mathrm{WR}} = \overline{\mathrm{A12} \cdot \mathrm{RD}} \cdot \overline{\mathrm{A12} \cdot \mathrm{WR}} \\ &= (\overline{\mathrm{A12}} + \overline{\mathrm{RD}}) \cdot (\overline{\mathrm{A12}} + \overline{\mathrm{WR}}) \end{aligned} \tag{3.9}$$

b. *Adresare complet decodificată*

La acest mod de selectare a circuitelor de memorie toate liniile de adresă (*a* biţi) sunt folosite pentru adresare, deci se utilizează întreg spaţiul disponibil de adresare. Biţii mai semnificativi ai adresei sunt decodificaţi astfel încât să se genereze semnale de selecţie pentru toate dispozitivele de memorie utilizate.

---

[11] ROM = Read Only Memory (engl.) - memorie doatr cu citire
[12] RAM = Random Access Memory (engl.) - memorie cu citire scriere (traducerea exactă este “memorie cu acces aleator)
[13] CS = Chip Select

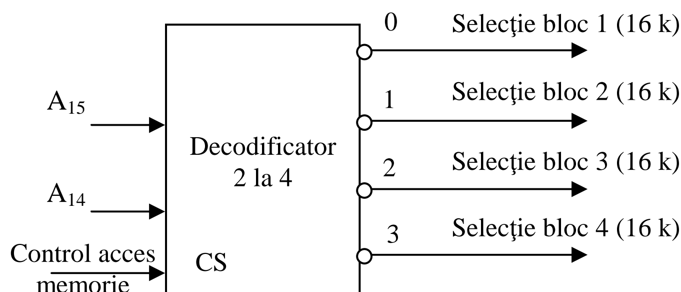


Figura 3.12. Decodificare pentru generarea semnalelor de selecţie şi adresarea unei memorii de 64 k prin decodificare completă. În cazul utilizării în sistem şi a unor circuite mai mici de 16k se mai poate introduce un bloc de decodificare circuitul decodificator fiind selectat de ieşirile decodificatorului 2:4, iar pe intrările de adrese se introduc următorii biţi mai semnificativi ai adresei, de la 13 în jos.

Ca exemplificare, în figura 3.12, se prezintă modul de decodificare şi generare de semnale de selecţie pentru un microprocesor similar cu cel de la adresarea liniară unde a = 16 biţi. Prin decodificare completă se pot adresa 2^{16} locaţii de memorie.

Folosind adresarea complet decodificată, adesea circuitele integrate de memorie sunt utilizate pentru a construi blocuri de memorie la care se doreşte creşterea capacităţii de stocare, deci a numărului de cuvinte adresabile. În exemplele de mai jos se indică modul de construcţie al blocurilor de memorie, în două variante: extinderea numărului de cuvinte, respectiv extinderea numărului de biţi pe cuvânt de aceeaşi adresă.

În ambele exemple se consideră circuite integrate de memorie cu citire-scriere, dar pentru a nu încărca figurile semnalele de citire / scriere nu au fost figurate. Ele oricum activează simultan toate circuitele integrate.

Exemplul 1: Să se construiască o memorie ce foloseşte adresarea complet decodificată cu următorii parametrii: capacitate totală de stocare 256 KB, circuite integrate de memorie RAM a câte 64 KB. Conform schemei bloc concepute să se deducă harta adreselor de memorie

Primul lucru pe care+l facem este să calculăm numărul de circuite integrate de memorie necesare:

256 KB / 64 KB = 4

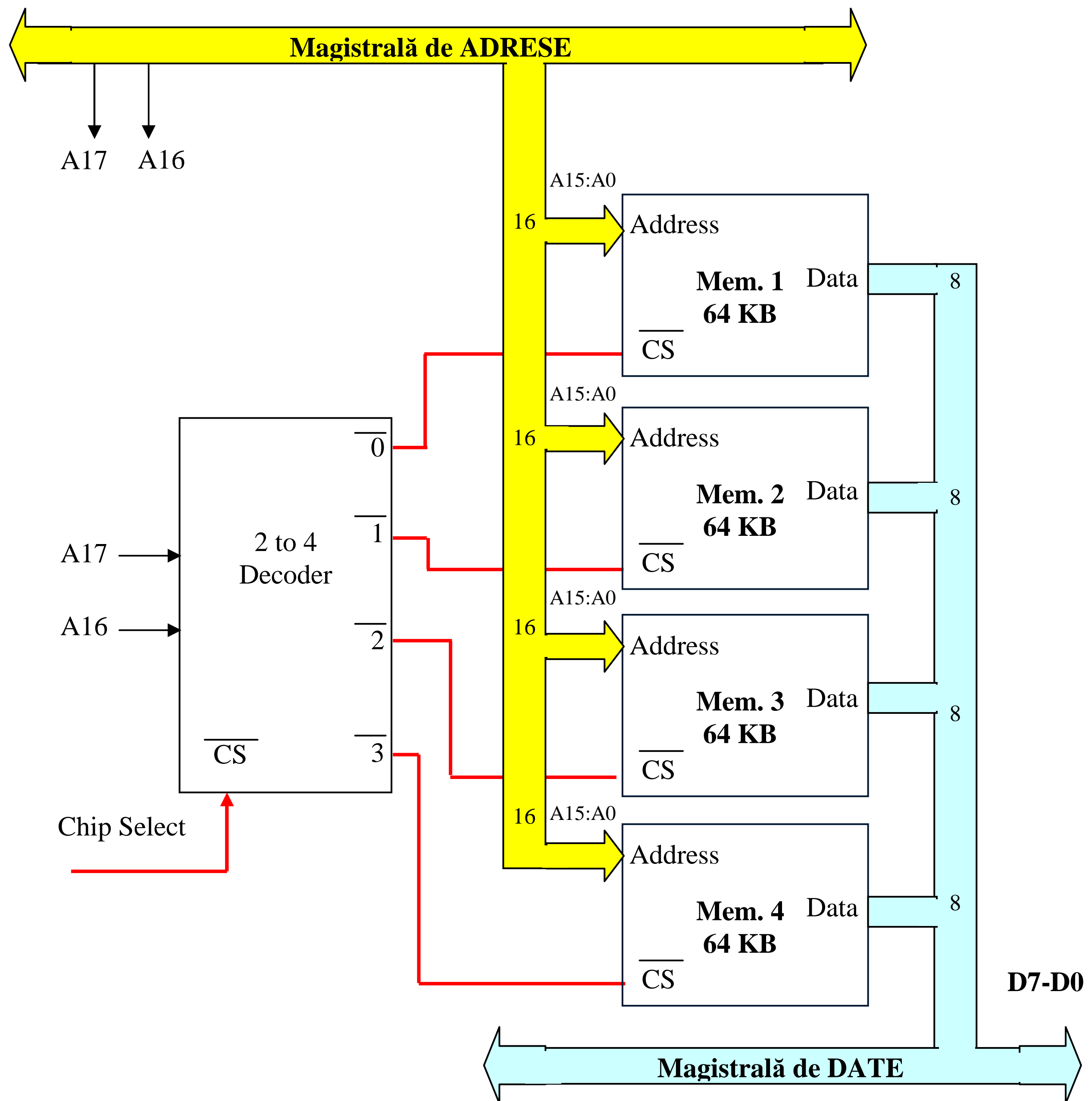
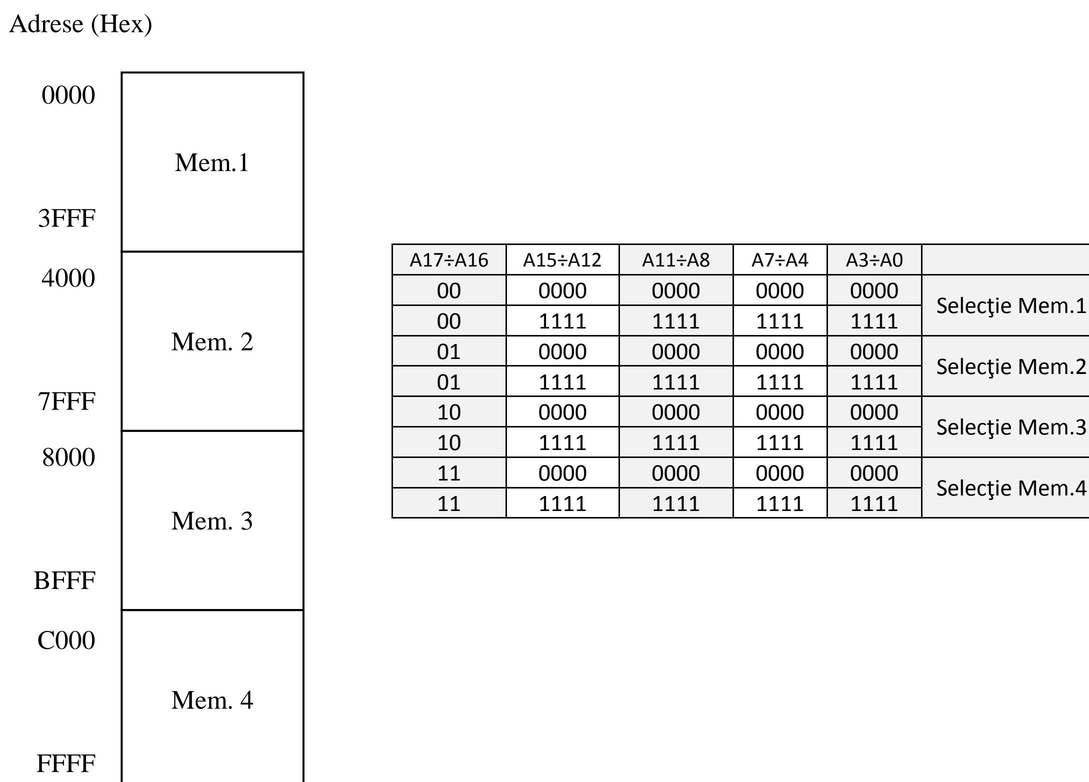


Figura 3.13. Extinderea numărului de cuvinte stocate - structura blocului de memorie de 256 KB construit cu CI de memorie de 64KB

Pentru fiecare CI de memorie există un semnal de selecţie ($\overline{CS}$) care dacă este activat permite lucrul cu memoria respectivă pentru operaţie de citire/scriere. Dacă această intrare nu este activă, tampoanele de intrare/ieşire date ale memoriei sunt inactive (în stare HiZ). Pe baza acestei observaţii, putem lega galvanic liniile de date cu acelaşi nume (D0 până la D7) ale tuturor CI împreună (Di, i=0-7, de la toate memoriile formează cele 8 linii care se leagă la magistrala de date de 8 biţi). Cu ajutorul decodificatorului selectăm un singur CI la un moment dat, în funcţie de

adresa de selecţie de 2 biţi de la intrare. Toate circuitele de memorie sunt adresate prin aceleaşi linii de adresă (16 biţi, pentru că 2^{16} = 64K). Harta adreselor de memorie, care specifică prima şi ultima adresă de stocare a datelor în fiecare din cele 4 CI de memorie este prezentată în figura 3.14, în care adresele sunt specificate în hexazecimal.



A17÷A16	A15÷A12	A11÷A8	A7÷A4	A3÷A0	
00	0000	0000	0000	0000	Selecţie Mem.1
00	1111	1111	1111	1111	
01	0000	0000	0000	0000	Selecţie Mem.2
01	1111	1111	1111	1111	
10	0000	0000	0000	0000	Selecţie Mem.3
10	1111	1111	1111	1111	
11	0000	0000	0000	0000	Selecţie Mem.4
11	1111	1111	1111	1111	

Figura 3.14. Harta adreselor de memorie pentru blocul din figura 3.13. În partea dreaptă s-a explicat modul în care au fost repartizate adresele şi conversia din binar în hexazecimal.

Exemplul 2: Să se construiască o memorie cu capacitate totală de stocare 128 KB, organizată ca 64K x 16 biţi, folosind circuite integrate de memorie RAM a câte 64 KB (extinderea numărului de biţi pe cuvânt).

În acest caz se vor folosi 2 CI de memorie pentru a forma cei 16 biţi. În figura 3.15 s-a exemplificat extinderea număruilui de biţi pe cuvânt. Nu are nici o importanţă modul cum notăm partea low (D7-D0) a cuvântului, din punctul de vedere al poziţiei memopriilor. Importantă este legarea corectă a ieşirilor la cele 16 linii ale magistralei.
Cele două circuite integrate de memorie sunt selectate de acelaşi semnla de Chip Select.

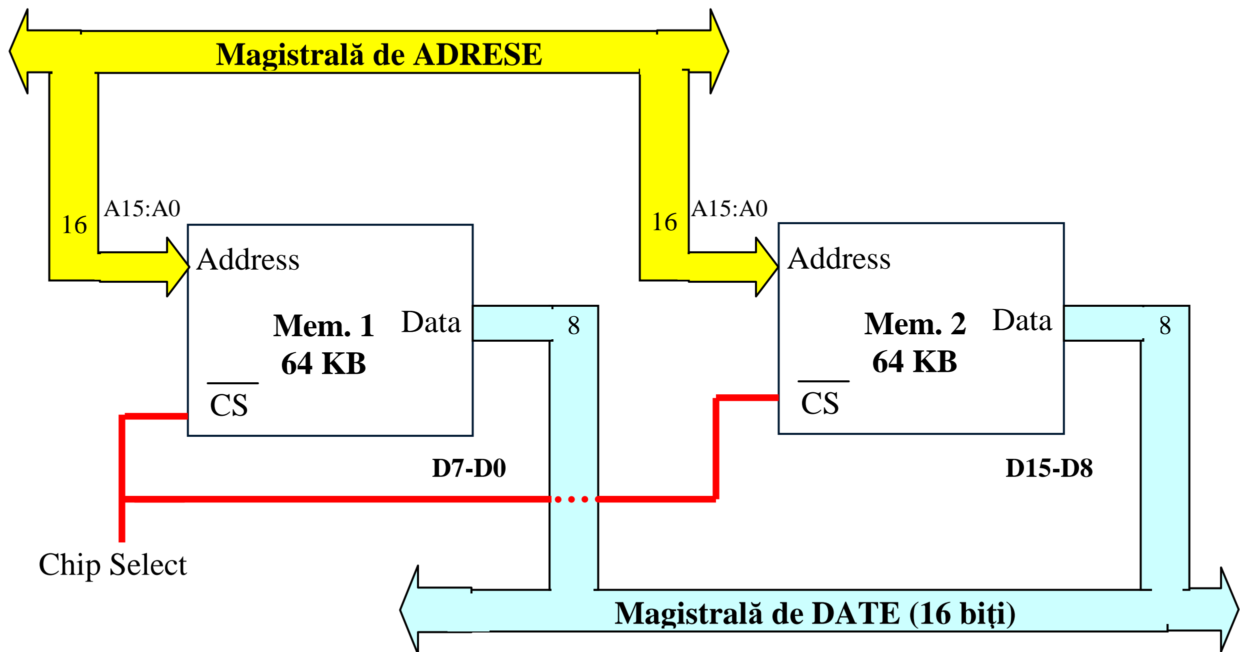


Figura 3.15. Extinderea numărului de biţi pe cuvânt- Exemplificare pentru construcţie cuvânt pe 16 biţi cu două CI de memorie cu ieşiri pe octet.

3.5.3. Magistrala de control

Magistrală de control conţine o diversitate de linii de control şi sincronizare, unele fiind doar uni-direcţionale (intrare sau ieşire din UCP), iar altele permiţând transferul bidirecţional de semnale. De aceea adesea se spune că magistrala de control este bidirecţională. Această magistrală cuprinde semnalele de control cu funcţie diferită de la un tip de microprocesor la altul. Cu toate acestea, funcţional, liniile magistralei de control pot fi clasificate în următoarele categorii generale [Sztojanov87]:

a. Semnale de control şi sincronizare pentru transferuri de date cu memoria şi dispozitivele de I/O;
b. Semnale de control şi sincronizare a cererilor de cedare a controlului magistralelor;
c. Semnale de control şi sincronizare cu evenimente externe ce generează cereri de întrerupere;
d. Semnale indicatoare de stare a UCP;
e. Semnale utilitare, cum ar fi reset, clock, alimentare cu tensiune;
f. Semnale diverse, specifice tipurilor de microprocesoare şi scopului pentru care au fost proiectate. În categoria semnalelor diverse se încadrează de exemplu: intrări testabile prin software, intrări pentru comanda activităţii pas cu pas a UCP, intrări

pentru semnalizarea unor erori, intrări / ieşiri pentru lucrul în sistem multi-procesor etc.

3.5.3.a Semnale de control pentru transferul datelor cu memoria şi dispozitivele de I/O

Rolul acestor semnale este de control şi sincronizare al transferurilor pe magistrala de date. Sensul de circulaţie al informaţiei pe magistrala de date este controlat de semnale care indică operaţie de citire, sau scriere. Sensul, pentru ambele operaţii sunt privite din punctul de vedere al UCP. În general semnalele de control sunt active pe nivel JOS, în aşa fel încât nivelul SUS să corespundă stării inactive a semnalului. Situaţia este avantajoasă, pentru că durata semnalelor de control activate este mult mai mică decât durata când ele nu sunt active, iar "în pauza" semnalului se foloseşte avantajul marginii de zgomot mai mare pentru nivelul logic 1 (SUS în logică pozitivă). Vom nota aceste două semnale cu prescurtările lor din limba engleză, bara de deasupra reprezentând faptul că sunt active pe nivelul logic JOS ("0"):

- $\overline{RD}$ (ReaD, citeşte), semnal ce indică că se efectuează o operaţie de citire a datelor de către UCP. De obicei frontul posterior al semnalului de citire (aici frontul crescător) este utilizat de UCP pentru citirea propriu-zisă a datelor depuse pe magistrala de date de către memorie sau un dispozitiv de I/O.
- $\overline{WR}$ (WRite, scrie), semnal ce indică că se efectuează o operaţie de scriere a datelor de către UCP. De obicei frontul posterior al semnalului de scriere este utilizat de dispozitivul adresat şi selectat pentru ciclul de scriere, pentru a stoca datele într-un registru propriu.

La unele microprocesoare (de exemplu microprocesorul Motorola MC68000) există un singur semnal de ieşire ($R/\overline{W}$), care îndeplineşte ambele funcţii, pentru nivel "1" făcându-se citire, iar pentru nivel "0" scriere.

Transferurile între UCP şi un dispozitiv interlocutor lent (memorie sau port) se fac de obicei asincron, pentru ca UCP să poată lucra la viteza maximă cu dispozitivele rapide şi lent cu cele lente. Pentru aceasta sunt necesare semnale pentru realizarea protocolului asincron de comunicaţie (de tip handshake). Setul minimal de semnale de sincronizare este constituit din următoarele:

- un semnal generat de UCP ce indică dispozitivelor interlocutoare, că UCP a furnizat (pe magistrala de adrese) o informaţie validă de adresă. La locaţia cu această adresă se va face scriere sau citire la un moment imediat următor. Acest semnal, împreună cu informaţiile de adresă şi de sens ($\overline{RD}$ sau $\overline{WR}$) al circulaţiei pe magistrala de date, produc selecţia/activarea dispozitivului interlocutor. Vom nota semnalul de validare al adresei cu $\overline{AS}$ (Address Strobe = validare adresă)

- un semnal recepţionat de UCP, care să indice faptul că interlocutorul este gata pentru efectuarea transferului. Vom nota acest semnal cu *GATA*. Acest semnal informează UCP că dispozitivul a livrat cuvântul de date (în cazul unui ciclu de citire) sau că este în măsură să primească date (în cazul ciclului de scriere). Dacă dispozitivul nu poate răspunde într-un anumit interval de timp, corespunzător unui număr de perioade ale impulsului de ceas, el nu poate activa GATA, iar UCP va introduce stări suplimentare de aşteptare, până când transferul este posibil; semnalul GATA este deci folosit pentru sincronizarea celor doi interlocutori. Recapitulând: dacă GATA este inactiv (= 0 logic), UCP va prelungi ciclul maşină curent, "îngheţând" toate celelalte semnale de comandă, prin adăugarea de stări de aşteptare (WAIT) până când recepţionează GATA=1, după care ciclul maşină de scriere (sau citire) se încheie.

Sincronizările cu memoria şi cu dispozitivele de intrare-ieşire sunt asemănătoare. La multe dintre calculatoare însă, prin construcţie, se cunoaşte viteza de lucru a memoriei principale, iar lucrul cu memoria principală poate fi privit ca fiind sincron, nefiind necesar semnalul de tip GATA. În această situaţie transferul se desfăşoară strict în cadrul unui interval pre-specificat de timp (ca număr de stări ale UCP). În toate celelalte cazuri transferurile sunt asincrone, UCP prelungindu-şi ciclul maşină curent cu stări de aşteptare, până când semnalul GATA devine activ sau până când un alt eveniment (excepţie sau reset) întrerupe acest ciclu maşină. Generarea sau nu a unui semnal de "GATA" ţine de modul de lucru asincron sau sincron. Într-un mod de lucru sincron toate evenimentele se desfăşoară în cadrul unui interval specificat de timp.

UCP eşantionează intrarea GATA, de obicei într-una din primele stări ale ciclului maşină.

Exemple de semnale de control al transferurilor la diferite microprocesoare se indică în tabelul 3.3. Pentru $\overline{RD}$ şi $\overline{WR}$ nu s-au exemplificat denumirile semnalelor la procesoare reale, pentru că în majoritate se păstrează aceeaşi notaţie pe care am folosit-o şi noi în acest paragraf.

Tabelul 3.3.

Tip microprocesor	Echivalent $\overline{\mathrm{AS}}$	Echivalent GATA
I8080+ I8228	$\overline{IOW}$, $\overline{IOR}$, $\overline{MEMR}$, $\overline{MEMW}$	*READY*
Z80	$\overline{MREQ}$, $\overline{IOREQ}$, $\overline{RFSH}$	$\overline{WAIT}$
I8086, I8088, I80x86	*ALE*, $M/\overline{IO}$, $DT/\overline{R}$, *DEN*	*READY*
MC680x0	$\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$	$\overline{DTACK}$

La multe dintre microprocesoare semnalul de tip $\overline{AS}$ depinde de tipul ciclului maşină curent executat de procesor. Din exemplele date în tabelul 3.3 se vede că aceste semnale, din categoria validare adresă ($\overline{AS}$), pot fi separate pentru adresele din spaţiul memoriei respectiv spaţiul de I/O. Astfel la Intel 8080 cu controller de sistem există scriere/citire pentru dispozitiv de intrare-ieşire ($\overline{IOW}$[14] şi $\overline{IOR}$[15]) sau scriere/citire adresată memoriei principale ($\overline{MEMW}$ / $\overline{MEMR}$[16]). Se observă că la acest procesor semnalele de validare a adresei sunt combinate cu cele ce indică scrierea sau citirea. La microprocesorul Zilog, Z80, în afara semnalelor $\overline{RD}$ şi $\overline{WR}$, există semnale diferite de validare a adreselor pentru memorie ($\overline{\mathrm{MREQ}}$[17] indică existenţa unei adrese de memorie pe magistrala de adrese) şi spaţiu de I/O ($\overline{\mathrm{IOREQ}}$ indică existenţa unei adrese de intrare - ieşire pe magistrala de adrese). Aceste tipuri de semnale pot fi privite şi ca semnale indicatoare de stare, pentru că ele indică existenţa unui ciclu maşină de lucru cu memoria, respectiv cu porturile de I/O.

În cazul microprocesorului I8086, în modul minim, există un singur semnal, numit $\mathrm{M}/\overline{\mathrm{IO}}$, care indică transfer cu memoria pentru valoarea 1 logic şi cu porturile de I/O pentru 0 logic. Semnalul ALE[18] de la I8086 (mod minim) validează existenţa informaţiilor de adresă pe pinii magistralei de date multiplexate. Pentru că microprocesoarele de 16 şi 32 de biţi pot face şi transferuri pe octeţi au fost necesare semnale de tip $\overline{\mathrm{BHE}}$ (la Intel 8086, validarea celor 8 linii mai semnificative de pe magistrala de date), sau $\overline{\mathrm{UDS}}$ (Upper Data Strobe - validare a octetului mai semnificativ de date) şi $\overline{\mathrm{LDS}}$ (Lower) la MC68000. La procesorul Motorola 68000 nu există semnal de tipul $\mathrm{M}/\overline{\mathrm{IO}}$ ca la I8086, pentru că 68000 nu are spaţiu separat pentru memorie şi dispozitive I/O (deşi există un semnal numit $\overline{\mathrm{VMA}}$ care contribuie la selecţia porturilor programabile de 8 biţi din familia MC6800). Specific microprocesorului Zilog Z80, (care are inclus în UCP şi controlerul de reîmprospătare a memoriei dinamice) este semnalul $\overline{\mathrm{RFSH}}$[19] care validează adresa de reîmprospătare a DRAM pe biţii A6-A0 ai magistralei de adrese.

În figura 3.16 se prezintă un exemplu de transfer asincron, cu sincronizare cu semnal de tip GATA între un microprocesor (UCP) şi un dispozitiv lent de memorie, pentru un ciclu de citire.

[14] Input Output Write
[15] Input Output Read
[16] Memory Write / Read
[17] Memory Request
[18] Address Latch Enable
[19] Refresh

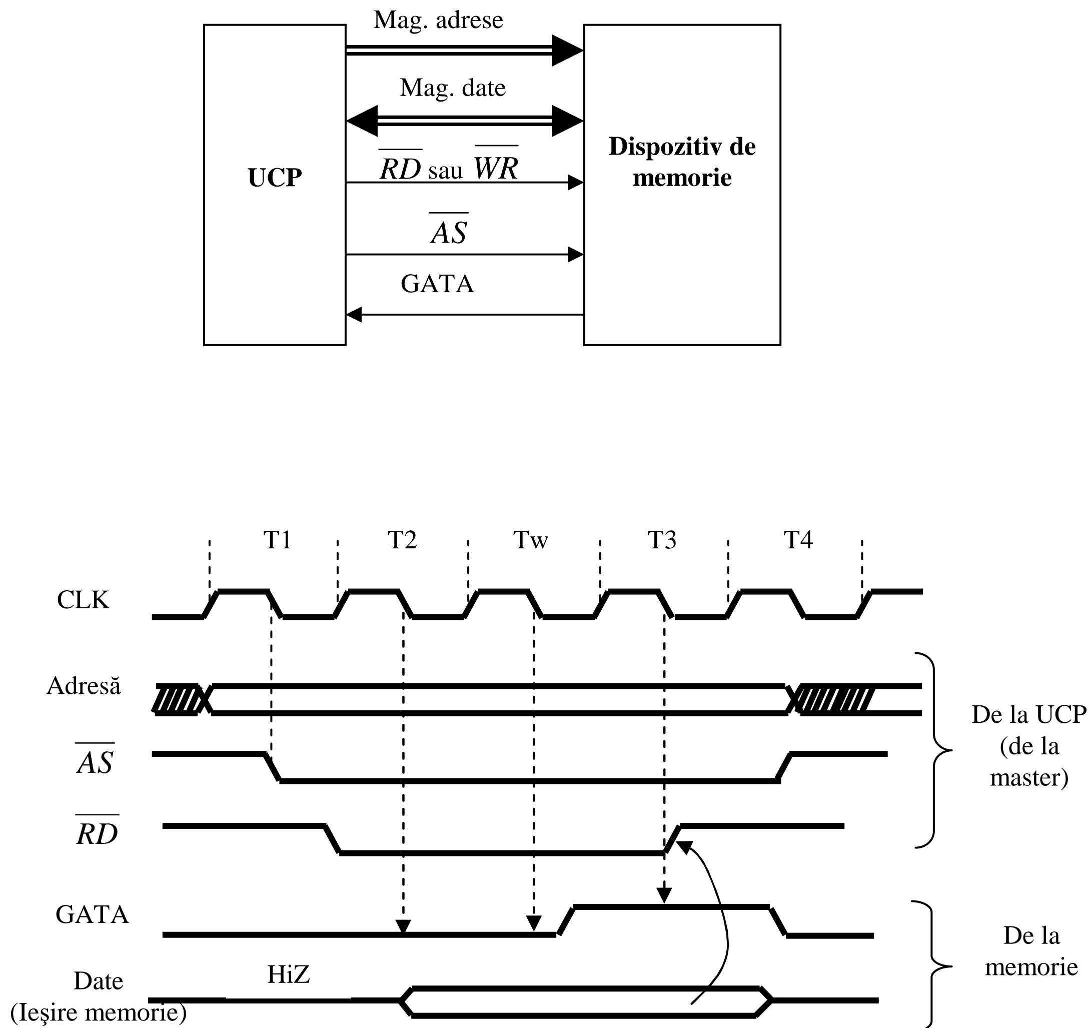


Figura 3.16. Exemplificare generală a unui ciclu de citire asincron, care are nevoie de semnale de control de tip handshake. În schema bloc de deasupra s-au reprezentat liniile de semnal la care sunt cuplaţi cei doi corespondenţi, iar în diagrama de timp se prezintă evoluţia în timp a semnalelor pe magistrale (notaţii CLK = impulsuri de ceas, Adresă = magistrală de adrese, Date = ieşirile dispozitivului de memorie către magistrala de date)

Diagrama desfăşurării în timp a semnalelor presupune că intrarea GATA este testată de microprocesor începând cu starea T2, pe fiecare front descrescător al impulsului de ceas. Din punctul de vedere al simbolurilor grafice folosite menţionăm că pentru semnalele singulare diagrama reprezintă tranziţiile între nivelurile logice “1” (SUS) şi “0” (JOS). În cazul reprezentării magistralelor, care conţin mai multe linii de semnal, cele două linii paralele la magistralele de date şi adrese reprezintă faptul că informaţia pe magistrală este stabilă. Intersecţia liniilor indică schimbarea informaţiei pe magistrală, iar zonele haşurate arată că informaţia (stabilă) respectivă este ne-semnificativă pentru ciclul descris de diagramă. Pe ultima linie a diagramei din figura 3.16 s-a reprezentat informaţia la ieşirile dispozitivului de memorie adresat. Aceste ieşiri sunt cuplate la magistrala de date, dar atunci când dispozitivul nu este selectat ieşirile se găsesc în stare de

înaltă impedanţă (HiZ) dacă ieşirile sunt de tip TSL. În urma selecţiei şi adresării dispozitivului de memorie, starea la ieşiri se schimbă doar după trecerea timpului de acces la celulele de memorie.

3.5.3.b Semnale de control a cererilor de cedare a controlului magistralelor

Problema cererii / cedării controlului magistralelor se poate pune doar în sisteme de calcul care conţin mai multe dispozitive ce pot controla cele trei tipuri de magistrale. Aceste dispozitive *active* pe magistrală (dispozitive master[20] care controlează dispozitive slave[21]) pot fi de exemplu: alte procesoare de uz general (UCP), procesoare cu sarcini specifice (de exemplu procesoare de I/O), circuite de tip controller care au şi posibilitatea de transfer al datelor direct cu memoria principală (transfer DMA[22]). Pentru ca un UCP să poată decide cu privire la cedarea controlului magistralelor, sistemul de arbitrare a controlului magistralelor trebuie să conţină cel puţin două tipuri de semnale:

- un semnal de intrare în UCP care face o *cerere de acces la controlul magistralelor*, semnal pe care-l notăm în continuare cu BR (Bus Request[23]).
- un semnal de ieşire din UCP care *confirmă cedarea controlului magistralelor*, notat în continuare BG (Bus Grant). În urma cedării controlului magistralelor, UCP către care s-a lansat cererea BR trece ieşirile sale către magistralele de date, adrese şi control (doar o parte dintre semnalele de control) în stare de înaltă impedanţă (HiZ). În acest fel se permite controlul (din punct de vedere electric, al nivelurilor logice) liniilor de către cel căruia i s-a cedat.

În tabelul 3.4. se prezintă câteva exemple de semnale echivalente cu semnalele tip pe care le-am numit BR şi BG.

Tabelul 3.4.

Tip microprocesor	Echivalent BR	Echivalent BG
I8080, I8085, I8086 în mod minim	HOLD	HLDA
Z80	$\overline{\mathrm{BUSRQ}}$	$\overline{\mathrm{BUSAK}}$
MC680x0	$\overline{\mathrm{BR}}$	$\overline{\mathrm{BG}}$ şi $\overline{\mathrm{BGACK}}$

[20] master (engl.) = stăpân
[21] slave (engl.) = sclav
[22] DMA = Direct Memory Access - Acces direct la memorie.
[23] Bus Request / Grant = Cerere / Acordare de control a magistralelor.

UCP prevăzut cu această pereche de semnale sondează de obicei semnalul de cerere, de tip BR, la sfârşitul fiecărui *ciclu maşină* şi cedează controlul magistralelor cât mai curând posibil, cu excepţia unor cazuri speciale în care operaţiile nu pot fi întrerupte (de exemplu operaţii prevăzute cu prefixul LOCK la I8086). Întârzierea maximă cu care este servită o cerere de acces la controlul magistralelor este deci un ciclu maşină. În cazul mai multor cereri de acordare a controlului magistralelor, analiza priorităţilor se face de obicei cu un circuit de arbitrare centralizată a cererilor de magistrală.

3.5.3.c Semnale de sincronizare cu evenimente externe ce generează cereri de întrerupere

Aceste semnale au rolul sincronizării evenimentelor externe cu UCP. Semnalele de control şi sincronizare pentru cereri de întrerupere externă sunt extrem de importante pentru că ele permit ca un dispozitiv periferic să lanseze un semnal de cerere de servicii către UCP (servicii de exemplu care se referă la transferul de date), iar UCP să întrerupă temporar programul rulat la acel moment, să sară la execuţia unui program de servire a întreruperii, iar apoi să revină la programul întrerupt. La recepţia unei cereri de întrerupere, dacă UCP acceptă întreruperea activităţii curente, se va informa dispozitivul întreruptor asupra acestui lucru. Ca urmare dispozitivul care a lansat întreruperea va genera (doar pentru întreruperile "vectorizate") pe magistrala de date un cod de identificare (vector de întrerupere) care să permită dirijarea execuţiei la subrutina de servire a întreruperii.

Există două tipuri de cereri de întrerupere hardware:

- întreruperi mascabile, a căror recunoaştere şi servire poate fi validată sau invalidată prin setarea unor indicatori de control
- întreruperi nemascabile, al căror efect nu poate fi blocat / mascat, fiind întotdeauna recunoscute.

Un set minim de semnale pentru manevrarea cererilor de întrerupere cuprinde:

- cel puţin un semnal de intrare în UCP, reprezentând o ***cerere de întrerupere***, prin care dispozitivul ce a lansat cererea aşteaptă servicii de la UCP (notat în continuare cu INT). La unele microprocesoare există mai multe intrări de cerere de întrerupere.
- cel puţin un semnal de ieşire din UCP care reprezintă ***confirmarea acceptării întreruperii*** (semnal notat în continuare cu INTA) La multe dintre microprocesoare acest semnal are şi funcţia de control pentru citirea vectorului de întrerupere depus de dispozitivul întreruptor pe magistrala de date.

Câteva exemple de semnale de tip INT şi INTA se prezintă în tabelul 3.5.

Tabelul 3.5.

Tip microprocesor	Echivalent INT	Echivalent INTA
I8080 şi I80x86	INTR (mascabil), NMI (nemascabil)	$\overline{INTA}$
Z80	$\overline{INT}$ (mascabil), $\overline{NMI}$ (nemascabil)	$\overline{IORQ} + \overline{M1}$
MC680x0	IPL2, IPL1, IPL0	FC2, FC1, FC0

3.5.3.d Semnale indicatoare de stare a UCP

Acestea sunt de obicei semnale, sau combinaţii de semnale de ieşire care indică starea în care se găseşte microprocesorul. Starea automatului este importantă pentru diferite circuite suplimentare conectate la interfaţa microprocesorului cu exteriorul, aceste circuite având diverse de funcţii de control în ceea ce priveşte magistralele, memoria cache externă şi memoria principală.

Semnalele de stare sunt extrem de diverse, de la tip la tip de microprocesor dar funcţiile lor pot fi clasificate după informaţia furnizată. Astfel semnalele de stare pot indica:

- tipul ciclului maşină curent. Informaţia este utilă circuitelor controller de magistrală, care pe baza stării citite furnizează semnale specifice de control. De exemplu, la Intel 8086 în mod "maxim" semnalele S0-S2 indică tipul ciclului maşină curent. La MC68000 există trei linii de ieşire numite "Function Codes", FC2 - FC0 care informează exteriorul cu privire la starea procesorului.
- informaţii despre registrele interne implicate în calculul de adresă. De exemplu la I8086 în mod "maxim" semnalele S3 şi S4 indică registrele segment utilizate pentru adresa curentă, iar semnalele QS1 şi QS0 dau informaţii cu privire la starea cozii de instrucţiuni.
- informaţii de sincronizare cu alte module master de magistrală (de exemplu de tip *lock*)
- starea memoriei tampon (cache, coadă) internă, informaţie utilă circuitului controller de memorie şi cache extern.
- speciale, pentru lucrul cu coprocesoare aritmetice externe. De exemplu cerere şi acceptare de transfer operand (PEREQ/PEACK – la Intel 80286), test busy (BUSY la I8086), informare de eroare ($\overline{BERR}$ la 68000).

3.5.3.e Semnale utilitare

În această categorie sunt incluse mai multe tipuri de semnale utilitare, dintre care amintim:

- semnale care aduc procesorul într-o stare predeterminată (RESET),
- semnalele de ceas (CLOCK)
- alimentarea cu tensiune.

RESET

Semnalul de tip RESET este prezent la toate microprocesoarele şi el aduce conţinutul registrelor microprocesorului într-o stare prefixată prin proiectare. În această stare predeterminată contorul de program (PC) se iniţializează la o valoare fixă (de obicei la zero), iar întreruperile mascabile sunt invalidate. Scopul principal al acestui semnal de intrare în UCP este ca să se ştie cu precizie starea în care se găseşte procesorul la alimentarea cu tensiune. Din această cauză la această intrare se conectează circuite R-C de tip Power-on Reset (Resetare la alimentarea circuitului). Chiar dacă pentru majoritatea microprocesoarelor acest semnal este doar intrare pentru UCP, există şi excepţii. Astfel, la microprocesorul Motorola MC68000 linia de $\overline{Reset}$ este bidirecţională. Dacă este folosită ca intrare şi este forţată la "0" logic din exterior, atunci când intrarea $\overline{Halt}$ e activă în acelaşi timp, se produce iniţializarea internă a microprocesorului. Aceeaşi linie este folosită ca ieşire, pentru iniţializarea dispozitivelor de I/O, atunci când microprocesorul execută instrucţiune "Reset" instrucţiune care însă nu modifică şi starea internă a microprocesorului.

Semnale de ceas

Semnalele de ceas sunt generate de către un oscilator intern sau extern pe baza unui cristal de cuarţ extern procesorului. La unele microprocesoare circuitele de ceas pot efectua şi "sincronizarea" unor semnale externe, cu caracteristică asincronă, cum ar fi: RESET şi READY. Unele din circuitele oscilator de ceas pot avea şi alte funcţii. De exemplu, la circuitul I8224 folosit pentru Intel 8080, şi la I8284 al lui Intel 8086 se generează şi semnale RESET pentru celelalte unităţi funcţionale ale calculatorului. Cristalele cu cuarţ au frecvenţe de oscilaţie diferite în funcţie de direcţia de prelucrare faţă de reţeaua cristalină şi în funcţie de grosime. Pot exista cristale echivalente cu circuite rezonante (RLC) serie şi paralel.
Producătorii specifică dacă cristalele sunt rezonante serie sau paralel. Cristalele cu cuarţ sunt construite cu toleranţe (frecvenţă şi stabilitate a frecvenţei) ce pot ajunge până la 0,002% - 0,005%. Pentru sistemele de calcul o este acceptabilă o toleranţă în jurul valorii de 0,01%.

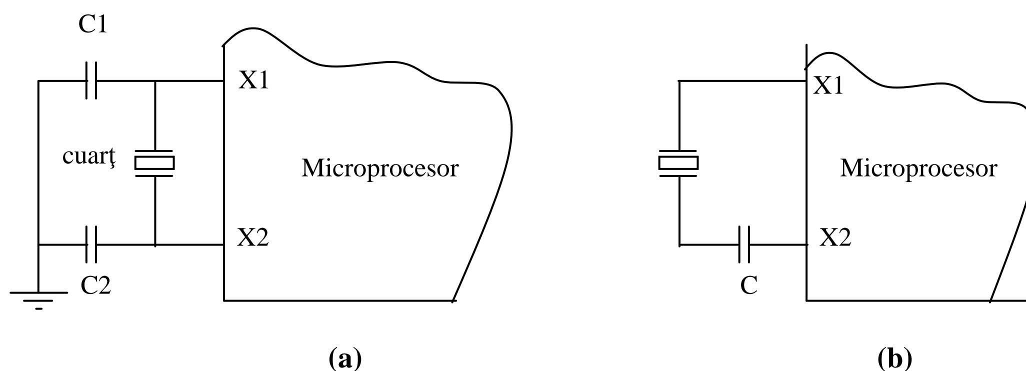


Figura 3.17. Exemplu de conectare a cristalului cu cuarţ cu schemă echivalentă paralel (a), respectiv serie (b).

În figura 3.17 se indică două moduri de conectare a cristalelor de curaţi la capsula microprocesorului. Prima, din figura 3.17 (a) este legarea corespunzătoare microprocesoarelor cu oscilator intern. Tipic fiecare dintre condensatoarele acestei scheme are valoarea de 10 - 30 pF. În figura 3.17 (b) se indică conectarea unui cuarţ cu circuit oscilant echivalent de tip serie. Capacitatea introdusă în serie cu cristalul previne polarizarea în curent continuu de către circuitele interne microprocesorului.

3.6. Sistemul de întreruperi

În timpul rulării programelor pot apărea unele evenimente neobişnuite (de excepţie). Aceste evenimente pot conduce la suspendarea temporară a programului aflat curent în execuţie; în acest interval de *întrerupere* a rulării programului se va executa, de obicei, o rutină de tratare a evenimentelor ce au întrerupt programul. Toate aceste evenimente produse de condiţii neobişnuite / neaşteptate pentru programul curent în execuţie sunt numite la modul general "***întreruperi***" pentru că ele produc întreruperea programului si devierea / saltul către o rutină specială de tratare a evenimentului "întreruptor". Putem clasifica aceste evenimente "întreruptoare" în:

1. ***cereri de întrerupere***: evenimente generate din exteriorul UCP, *asincrone* cu programul rulat, care cer tratare. Aceste cereri de întrerupere pot proveni:
   - 1.a de la echipamente periferice care cer servicii de la UCP (cerere de transfer de date, sau informări cu privire la starea perifericului), sau
   - 1.b de la circuite specializate pentru supravegherea funcţionării normale a componentelor hardware (eroare de paritate la citirea memoriei, eroare de paritate pe magistrală, căderea iminentă a tensiunii de alimentare etc.);

   Evenimentele ce produc acest tip de întrerupere a programelor sunt numite şi *întreruperi hardware*.
2. ***excepţii***: evenimente neobişnuite *sincrone* cu programul (produse la rularea programului), care cer tratare. De obicei indică situaţii de excepţie cum ar fi:
   - 2.a întâlnirea unor instrucţiuni de control ilegale în programul utilizator,
   - 2.b încercarea de violentare a sistemului de protecţie a informaţiilor din memorie,
   - 2.c condiţii aritmetice speciale (depăşiri, împărţire la zero),
   - 2.d accesul la pagini virtuale de memorie nerezidente în memoria principală.

   Excepţiile produc *devieri*, ("traps") sau *întreruperi software*.

Efectul general al unei întreruperi, atât pentru întreruperea hardware cât şi pentru întreruperea software este o deviere către o rutină de tratare a întreruperii (rutină de servire a întreruperii). Indiferent de sursa ce a generat întreruperea cele două tipuri sunt servite în mod asemănător, prin saltul la o rutină de tratare a evenimentului întreruptor sau de excepţie. În literatura actuală, termenii de "întrerupere" şi "cerere de întrerupere" sunt folosiţi, în general, pentru evenimente ce produc cereri asincrone cu programul curent rulat. Excepţiile - sincrone cu programul rulat - produc devieri, care la rândul lor pot fi clasificate în mai multe tipuri.

De exemplu, după modul de manifestare şi tratare a excepţiilor, Intel împarte aceste excepţii în

următoarele categorii:

- *devieri ('traps').* O deviere poate fi recunoscută doar după terminarea ciclului instrucţiune curent (recunoscută doar înainte de începerea ciclului de execuţie al unei instrucţiuni ce urmează imediat după instrucţiunea în care s-a detectat excepţia). După tratare *devierile* produc reluarea programului întrerupt începând cu instrucţiunea următoare.
- *erori ('faults')* . Sunt excepţii ce sunt detectate fie înainte de începutul execuţiei instrucţiunii, fie in timpul execuţiei acestora. Dacă erorile se detectează în timpul execuţiei unei instrucţiuni, după tratarea erorii reluarea programului se va face începând cu instrucţiunea întreruptă.
- *esecuri - terminari anormale ('abort').* Produc abandonarea procesului, iar rutina de tratare afişează mesaje "stresante" pentru utilizator. Eşecurile sunt utilizate pentru a trata erori severe cum sunt valori ilegale si / sau inconsecvente în tabelele sistemului sau erori de hardware.

Din cele expuse de mai sus se observă că indiferent dacă evenimentul ce produce întreruperea provine de la circuitele calculatorului (hardware), sau de la rularea instrucţiunilor programului (software), tratarea evenimentului se face în mod asemănător:

- când UCP recunoaşte o eroare sau o condiţie neobişnuită, el opreşte obligatoriu rularea programului aflat în execuţie, şi produce saltul la o rutină care va trata corespunzător condiţia apărută.
- când UCP recepţionează o cerere de întrerupere, el opreşte rularea programului aflat în execuţie (presupunem că sunt permise şi întreruperile mascabile) şi produce saltul la o rutină de tratare corespunzătoare a cererii de întrerupere

Conform clasificării de mai sus, cererile de întrerupere (asincrone) sunt întreruperile generate hardware (prin semnal electric). O întrerupere generată hardware se produce ca răspuns la activarea unui pin de intrare în UCP. După modul de servire al cererilor de întrerupere, există două tipuri de intrări în microprocesor (vezi şi figura 3.18):

⇒ cerere de întrerupere este *mascabilă* (notată INTR în figura 3.18), la care se poate bloca ("masca") acţiunea de recunoaştere a acesteia de către UCP. Ca urmare a mascării programul rulat de UCP nu va fi întrerupt, iar semnalul de cerere de întrerupere va fi neglijat;

⇒ cerere de întrerupere *nemascabilă* (notată NMI în figura 3.18). Acest tip de întrerupere este obligatoriu, va fi recunoscut necondiţionat de UCP şi ca urmare va produce deviere către o rutină de tratare.

Figura 3.18. prezintă schematic logica de intrare în microprocesor, pentru întreruperile

mascabile şi nemascabile.

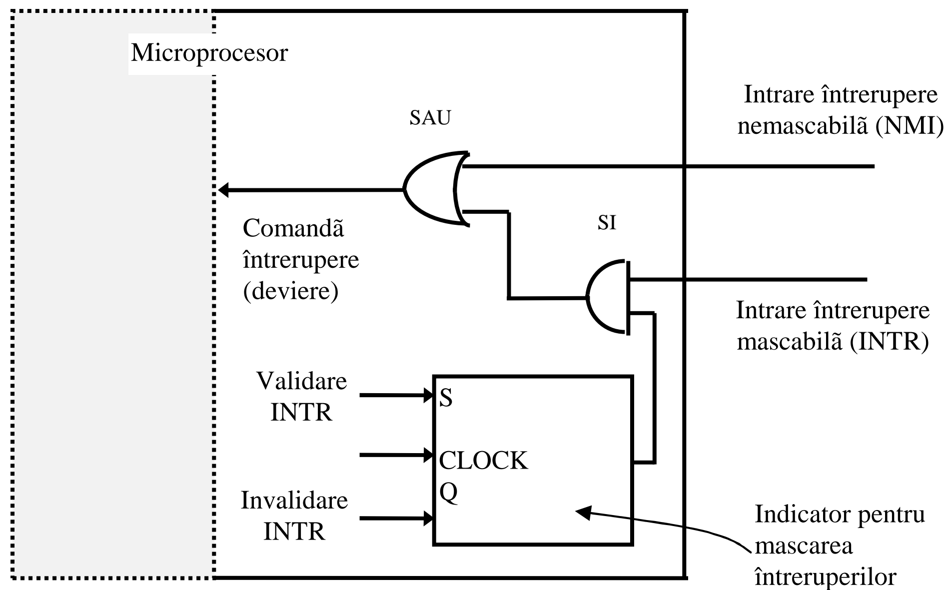


Figura 3.18. Reprezentare a modului de recunoaştere pentru cererile de întreruperi mascabile şi nemascabile

La intrarea de întreruperi mascabile se conectează de obicei linii de semnal de la dispozitivele de I/O. Electric acest lucru este posibil pentru că la majoritatea microprocesoarelor intrarea este activă pe nivel logic JOS fiind destinată ieşirilor cu colector în gol de la dispozitive. Intrarea INTR permite dispozitivelor externe să întrerupă execuţia unui program. Pentru a asigura recunoaşterea cererii, intrarea INTR trebuie să fie menţinuta la nivel activ până când procesorul confirmă întreruperea prin realizarea unei secvenţe de confirmare a acceptării întreruperii. Intrarea INTR este eşantionată la sfârşitul execuţiei fiecărei instrucţiuni. În plus, întreruperile mascabile trebuie sa fie validate prin software (instrucţiuni ce produc setarea indicatorului de control a întreruperilor mascabile - reprezentat ca un bistabil în figură) pentru a putea fi recunoscute. Semnalul INTR poate fi furnizat direct de un dispozitiv periferic, sau este furnizat de circuite specializate pentru controlul si arbitrarea mai multor semnale externe de cerere de întrerupere (controller programabil de întreruperi).

Intrarea NMI (pentru cerere de întrerupere ne-mascabilă) este folosită de obicei de circuitele externe UCP care supraveghează buna funcţionare a sistemului hardware. Ele pot semnala de exemplu erori de paritate la transferul datelor, erori de memorie, creşterea peste limite a temperaturii interne, căderea iminentă a tensiunii de alimentare etc. Activarea intrării NMI face ca procesorul să întrerupă programul la sfârşitul ciclului instrucţiune curent şi să execute automat o rutină de servire care corespunde, de obicei, unei locaţii fixe din memoria principală

(întrerupere ne-vectorizată). Procesorul nu va servi cereri succesive NMI până când tratarea cererii curente nu este terminată.

Dacă mecanismul hardware de recunoaştere a unei întreruperi nu este complicat, în cazul excepţiilor produse de rularea instrucţiunilor lucrurile sunt un pic mai complexe. Cauzele excepţiilor sunt diverse şi ele se manifestă în diferite moduri. De exemplu, cauzele aritmetice, cum ar fi depăşirea domeniului de reprezentare de rezultatul unei adunări, pot fi tratate şi rezolvate relativ uşor. Dar dacă cuvântul binar adus ca şi cod al instrucţiunii nu reprezintă nici un cod recunoscut de procesor evenimentul va produce sigur abandonarea programului. Codul nerecunoscut este tratat ca "instrucţiune ilegală" şi el poate apărea fie din cauza erorilor umane fie datorită erorilor hardware ale maşinii. Tratarea acestei excepţii trebuie făcută nu numai pentru a informa utilizatorul despre evenimentul produs, dar şi pentru a închide execuţia acelui program. Cea mai defavorabilă soluţie este bineînţeles blocarea calculatorului. Pentru cazul depăşirilor, unele calculatoatoare au instrucţiuni speciale de test pentru overflow (ex. BOV Branch if OVerflow). Este însă complet neconvenabil ca fiecare instrucţiune aritmetică să fie urmată de un test. În cazul instrucţiunilor ilegale nu este posibil să se testeze condiţia, pentru că trebuiesc luate imediat măsuri. Calculatoarele timpurii, care nu aveau un sub-sistem de întreruperi, s-ar fi oprit în cazul unor asemenea condiţii, dar asta n-ar fi făcut decât să împingă problema mai jos cu un nivel, către *operatorul* de serviciu, care trebuia să rezolve problema. La maşinile actuale, de mare viteză, este de dorit să se acţioneze imediat pentru a se elimina timpul pierdut de calculator. Se presupune că de obicei condiţia ce a produs eroarea şi deci întreruperea este înţeleasă, tratată şi apoi se va continua cu sarcina întreruptă dacă este posibil. Dacă eroarea este fatală sarcina curentă este abandonată şi se va trece la execuţia unei alte sarcini de lucru. Descriem în continuare câteva din condiţiile care pot deveni surse de deviere a programului rulat.

Condiţii aritmetice. Aceste excepţii sunt date de condiţii aritmetice neobişnuite cum sunt depăşirile (overflow şi underflow), încercare de împărţire la zero etc. Deşi utilizatorul poate testa aceste condiţii cu instrucţiuni de test, este preferabil să existe o monitorizare continuă, hardware - prin circuite specializate, a acestor condiţii. Acest procedeu este mai eficient din punctul de vedere al timpului de execuţie pentru că nu trebuie executate instrucţiuni suplimentare decât în cazul când se produc condiţiile testate. De multe ori se permite utilizatorului (uneori în faza de compilare) să specifice ce trebuie făcut, separat, pentru fiecare din tipurile de evenimente provenite din condiţii aritmetice. Dacă nu se indică nici o specificaţie, acţiunea implicită este de obicei abandonarea sarcinii (abort).

Instructiuni ilegale. O deviere datoarată unei instrucţiuni ilegale (instrucţiune de control nepermisă utilizatorului, sau cod ne-recunoscut) este de obicei fatală, adică termină execuţia programului utilizator pentru că se presupune că există o eroare în program, sau o eroare severă de hardware. La abandonare utilizatorul este informat despre eroare şi despre starea programului utilizator (conţinutul registrelor şi a memoriei).

Protecţie. Cele mai multe sisteme de calcul au în memorie la un moment dat mai mult decât un program pentru ca există mai mult decât un proces ce se execută. (de exemplu mai multe programe lansate în execuţie pe un calculator uni-procesor utilizat prin partajarea timpului. *Time sharing* = partajarea timpului procesor pentru execuţia mai multor programe. Fiecărui program i se alocă un interval de timp conform unui mecanism de planificare implementat de sistemul de operare). De aceea cele mai multe sisteme prevăd un anumit sistem de protecţie. Protecţia memoriei se referă la capabilitatea calculatorului de a marca acele zone de memorie care sunt disponibile unui program particular şi să detecteze referinţele interzise la acele zone. O zonă de memorie nedisponibilă pentru programul aflat în rulare este numită *regiune protejată* a memoriei. O încercare a programului utilizator de a scrie într-o regiune de memorie protejată este în mod clar o eroare de programare (intenţionată sau nu) şi produce o deviere (trap). În mod normal acest eveniment de excepţie este fatal, la fel ca şi devierile pentru instrucţiuni ilegale.

Erori ale maşinii. În mod ideal, calculatorul nu face niciodată erori, dar în practică, pot fi acceptate două feluri de erori hardware. O eroare catastrofală se produce când eroarea de hardware este atât de serioasă încât opreşte funcţionarea unui subsistem major. Asemenea erori sunt evidente imediat şi sunt relativ uşor de rezolvat pentru tehnicianul de service. Totuşi, cea mai frecventă formă de erori se produce ocazional, iar erorile pot să nu fie repetitive, chiar în aceleaşi condiţii. Aceste erori nu se produc foarte des ci probabil doar la intervale mari, de ore sau zile. De aceea sunt dificil de localizat, deci dificil de depanat. Totuşi unele erori sunt detectate automat de hardware; iar detecţia unei erori produce o deviere (trap) şi astfel sistemul de operare poate decide ce trebuie făcut. Rutina de monitorizare a sistemului de operare poate repeta acţiunea pentru a vedea dacă se produce fără eroare a doua oară. Daca repetarea se face cu succes, controlul poate fi returnat programului utilizator şi nu trebuie să ne mai facem probleme cu privire la acea eroare. Dacă eroarea se produce repetat, programul utilizator va fi abandonat (aborted) şi se va afişa un mesaj informativ adecvat pentru utilizator. De obicei dispozitivele de memorie au unele echipamente de detecţie a erorilor, tipic pentru generarea şi detecţia bitului de paritate. Paritatea permite doar detectarea erorilor izolate, dar erorile multiple în acelaşi grup de biţi pot să treacă nedetectate.

Unitatea centrală de procesare răspunde la o cerere de întrerupere (hardware sau software) şi tratează evenimentul ce a produs întreruperea în paşi succesivi care pot fi rezumaţi astfel:

1. UCP identifică sursa ce a produs întreruperea. Această identificare se poate face fie printr-o informaţie furnizată de evenimentul întreruptor (informaţie numită de obicei "vector de întrerupere") fie prin testarea conţinutului unor registre interne pre-definite sau a unor zone pre-definite din memoria principală. Uneori, în cazul cererilor de întrerupere hardware există intrări separate de întrerupere la care se conectează dispozitive de I/O, identificarea fiind făcută automat prin identificarea intrării de cerere

de întrerupere activată.

2. Pe baza informaţiei de identificare UCP calculează adresa unde se găseşte rutina de tratare a întreruperii.
3. Se salvează informaţia de stare a programului ce va fi întrerupt (în primul rând conţinutul contorului de program) într-o memorie de tip stivă. Se face saltul la rutina de tratare prin încărcarea PC cu adresa de început a rutinei de tratare.
4. Se execută rutina de tratare a întreruperii.
5. Se face revenire la programul întrerupt prin restaurarea informaţiei de stare.

De observat că după ce rutina de tratare a fost executată, ea va face revenire la programul întrerupt (printr-o instrucţiune specială de tip “return from interrupt" - întoarcere din rutina de tratare a întreruperii) şi se va continua cu execuţia acestuia. Este important ca rutina de tratare să salveze conţinutul registrelor interne pe care urmează să le modifice, ca să poată reface conţinutul acestora înainte de revenirea la programul întrerupt. Dacă rutina de tratare nu este scrisă în acest mod este foarte posibil ca programul întrerupt să nu mai funcţioneze corect. Pentru a se realiza operaţia de revenire la programul întrerupt este necesar să se ştie locaţia ultimei instrucţiuni executate din programul întrerupt. De aceea în momentul saltului la o rutină de tratare, sistemul de întreruperi al UCP salvează automat adresa următoarei instrucţiuni (adresa de întoarcere). Salvarea şi restaurarea contorului de program (PC) cu această adresă sunt automate şi nu cad în sarcina programului întrerupt. Vom numi "*vector de stare*" informaţia salvată în momentul devierii; ea cuprinde de obicei nu numai adresa locaţiei următoarei instrucţiuni, dar şi conţinutul registrelor de stare şi control ai UCP. Informaţia vectorului de stare, împreună cu conţinutul spaţiului de memorie alocat programului utilizator, furnizează o descriere completă a stării procesului la momentul devierii, numita context de lucru al procesului. Procesul poate fi repornit prin restaurarea acestei informaţii în registrele potrivite şi în locaţiile de memorie.

Cu privire la identificarea evenimentului ce a produs întreruperea, unele procesoare produc saltul către o adresă diferită pentru fiecare tip diferit de condiţie / întrerupere (*întreruperi vectorizate*). Altele produc saltul la aceeaşi adresă şi stochează informaţia pentru identificarea şi tratarea întreruperii la anumite adrese prestabilite din registrele procesorului sau din memorie (*întreruperi nevectorizate*). În oricare din cazuri, efectul este acelaşi, prin saltul la un program diferit (program care deviază cursul normal al programului întrerupt şi care este numit "rutină de tratare"), cunoscându-se care este condiţia care a produs aceasta devierea. Existenţa mecanismului de deviere, implementat în aşa numitul "subsistem de întreruperi" prezent în toate tipurile de procesoare, face sarcinile de programare ale utilizatorului mai simple.

În cazul întreruperilor vectorizate identificarea se face pe baza unui vector de întrerupere. Cu ajutorul acestuia UCP calculează o adresă pointer într-un tabel al vectorilor de întrerupere unde UCP găseşte adresa de început a rutinei de tratare a întreruperii (RTI) respective. Modul de

calcul al adresei pointer şi modul de construcţie al tabelei vectorilor de întrerupere este diferit de la microprocesor la microprocesor deşi principiul este asemănător. Ca exemplificare al modului de determinare al adresei RTI se propune exemplul din figura 3.19. Se presupune pentru simplificare că la acest sistem pot exista doar patru dispozitive ce pot lansa cerere de întrerupere, deci vectorul de întrerupere (notat VI) poate fi codificat cu doi biţi. Memoria este adresabilă pe octet, iar adresele au lărgimea de 16 biţi. La recunoaşterea unei cereri de întrerupere UCP citeşte VI de 2 biţi şi îl salvează în PC pe poziţiile binare cu ponderile 2^3 şi 2^2 iar în acelaşi timp ceilalţi 14 biţi ai PC sunt poziţionaţi la zero.

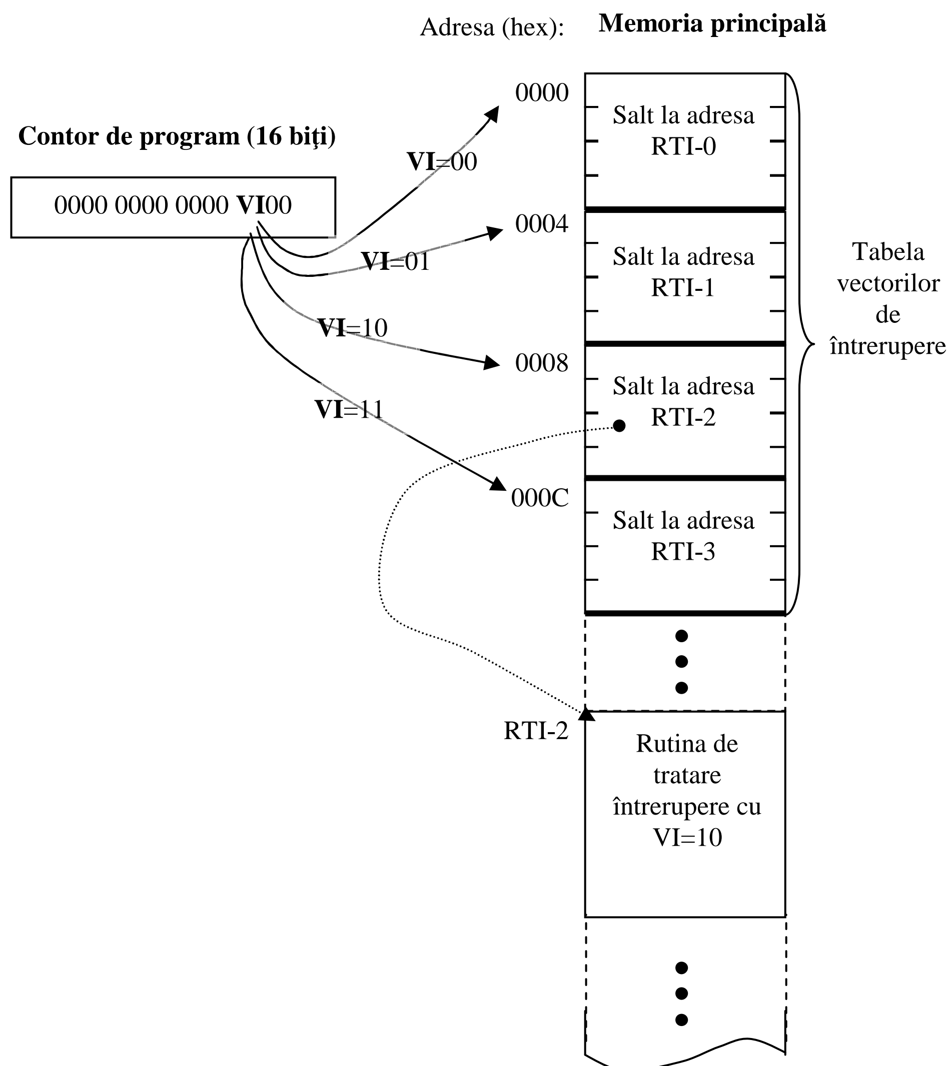


Figura 3.19. Exemplificare privind calculul adresei rutinei de tratare a întreruperilor (RTI) pentru un sistem cu doar patru cereri de întrerupere posibile. Cu VI s-a notat vectorul de întrerupere furnizat de dispozitivul întreruptor.

În acest fel fiecare intrare în tabela vectorilor de întrerupere din memoria principală corespunde la 4 octeţi unde se poate înscrie o instrucţiune de salt către RTI specifică. În figură s-a indicat prin linie punctată saltul la rutina de tratare a cererii de întrerupere cu vector de

întrerupere egal cu “10”. La unele microprocesoare între adresele pointer de intrare în tabela vectorilor de întrerupere poate exista o distanţă de 8 adrese în aşa fel încât o eventuală RTI foarte scurtă să poată fi introdusă aici. Mai mult, tabela poate fi stocate oriunde în memoria principală şi nu doar începând cu adresa zero, ca în exemplificarea din figura 3.19. Pentru exemplul anterior, poziţionarea tabelei în altă zonă a memoriei se poate face simplu, de exemplu prin setarea la 1 a bitului b15 din registrul PC, atunci când a fost recunoscută o cerere de întrerupere (tabelul începe în acest caz la adresa de memorie: 8000 hexa). O altă variantă este construcţia adreselor prin concatenarea PC cu informaţie din alt registru de adresare în cazul întreruperilor (ca informaţie mai semnificativă a adresei compuse generate).

De observat că mecanismul de salt la RTI face o adresare indirectă prin locaţie de memorie. Avantajul determinării adresei rutinei de tratare, prin acest mod indirect (relativ lent) este dat de faptul că rutinele de tratare pot fi stocate la orice adresă în memoria principală. De observat că o soluţie “rigidă” care ar stoca rutinele de tratare la adrese pre-fixate din memoria principală, pe lângă faptul că ar impune nişte restricţii în alocarea memoriei, ar fi foarte puţin flexibilă, fiind aproape imposibil de rescris o anumită rutină de tratare, dacă aceasta are lungime mai mare decât spaţiul alocat iniţial.

Exemplul 1 pentru întreruperi vectorizate

În cazul microprocesorului Z80 (mod 2 de lucru al sistemului de întreruperi), dacă procesorul recunoaşte cererea de întrerupere, lansează pe magistrala de control semnale de acceptare a cererii de întrerupere (M1+IORQ în figura 3.20). Ca urmare a acestei acceptări, dispozitivul întreruptor va pune pe magistrala de date o informaţie de identificare de 8 biţi pe care o vom numi vector de întrerupere (VI). Vectorul de întrerupere formează octetul mai puţin semnificativ al unei adrese pointer de 16 biţi. Octetul mai semnificativ al adresei pointer se citeşte dintr-un registru special utilizat pentru întreruperi (I), intern microprocesorului. Adresa pointer este adresa din memoria principală (din tabelul vectorilor de întrerupere) unde se găseşte adresa de salt către rutina de tratare a întreruperii. Pot exista maximum 255 de intrări în tabela vectorilor de întrerupere.

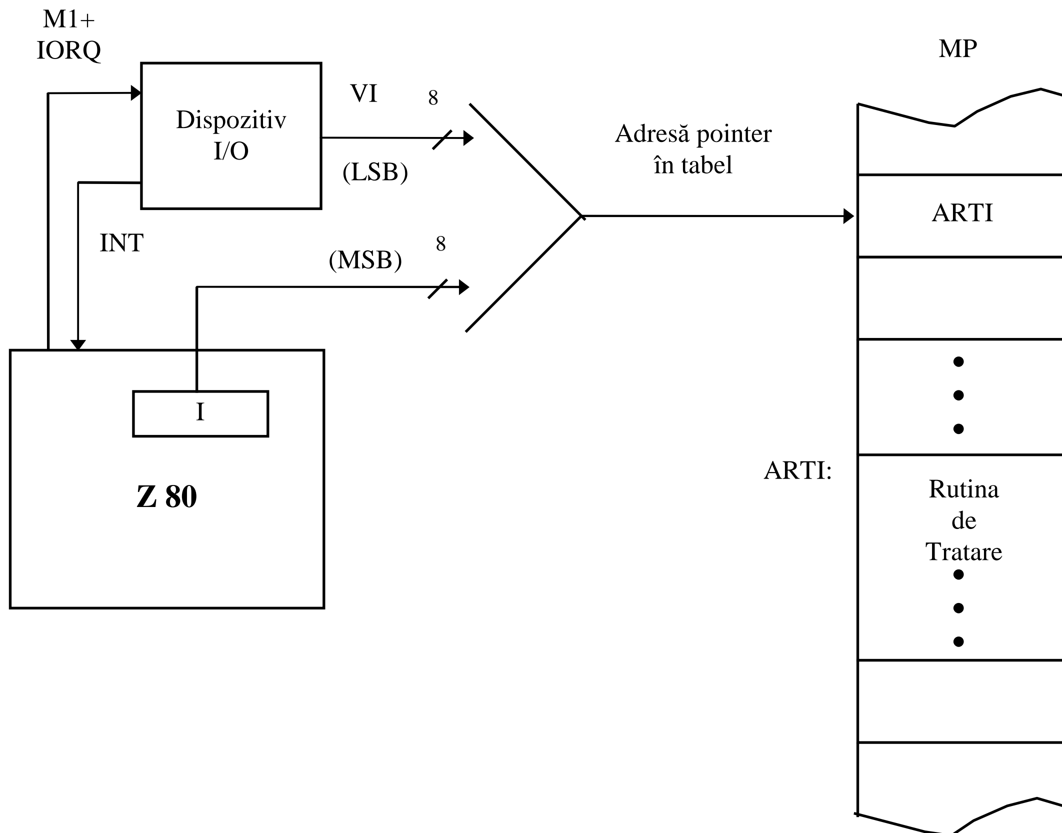


Figura 3.20. Exemplu simplificat pentru salt la rutina de tratare a unei întreruperii, în cazul procesorului Z80, în modul 2 de întreruperi. I = registru de întreruperi (păstrează octetul mai semnificativ - MSB - al adreselor din tabelul vectorilor de întrerupere); VI=vector de întrerupere, furnizat de dispozitivul de intrare/ieşire (I/O) căruia i s-a acceptat cererea de întrerupere; ARTI=adresa rutinei de tratare a întreruperii (este o adresă pe 16 biţi, iar memoria Z80 e organizată pe octet; de aceea pentru a citi ARTI se citesc doi octeţi începând de la adresa pointer construită).

Exemplul 2 pentru întreruperi vectorizate

*La procesorul I8086 (şi seria I80x86 în mod real) toate adresele rutinelor de tratare a întreruperilor se păstreaza într-o tabelă a vectorilor de întrerupere (TVI) care în modul ne-protejat de lucru se găseşte începând cu adresa 0 din memoria principală. La procesoarele Intel memoria este segmentată, aşa încât adresa de 20 de biţi se formează din două componente: adresa de bază a segmentului de cod (conţinută într-un registru segment numit CS) şi adresa în cadrul segmentului (conţinută pentru programe în registrul IP). Fiecare dispozitiv întreruptor sau eveniment de excepţie, va furniza un vector de întrerupere pe 8 biţi (Intel numeşte "**tip**" acest vector) cu ajutorul căruia se calculează adresa pointer (P) de intrare în tabela TVI. În tabelă se păstrează adresele rutinelor de tratare în felul următor: începând cu adresa cea mai mică dintr-un grup de 4 octeţi se găsesc valorile ce se vor înscrie în registrele de 16 biţi IP (mai întâi octetul LSB şi apoi MSB) şi respectiv CS (LSB, iar la adresa cea mai mare MSB).*

Relaţiile de calcul de adresare în TVI, respectiv transferurile ce se efectuează sunt:

*P = 4 * (VI) - pointer care va conţine adresa de intrare în TVI*

IP $\leftarrow$ TVI(P+1,P)

CS $\leftarrow$ TVI(P+3,P+2)

În acest fel întreruperea 0 începe de la adresa 0, întreruperea 1 de la adresa 4, iar întreruperea 255 începând de la adresa 1020 (în zecimal).

Pentru toate evenimentele care pot produce întrerupere, este deosebit de importantă viteza cu care sistemul răspunde la cerere. ***Timpul*** care se scurge înainte ca o întrerupere să fie tratată depinde de mai mulţi factori. Sursa de întrerupere trebuie să ia în considerare aceasta întârziere. Toţi factorii următori pot afecta timpul scurs până la recunoaşterea şi tratarea unei întreruperi:

- dacă întreruperile mascabile sunt invalidate, o cerere INTR nu va fi recunoscută până când întreruperile nu sunt re-validate (prin indicatorul de întreruperi).
- dacă se serveşte o întrerupere nemascabilă, o nouă cerere de întrerupere nemascabilă nu va fi recunoscută până când procesorul nu execută instrucţiunea de revenire din rutina de tratare a întreruperii (return from interrupt).
- salvarea registrelor de indicatori şi a altor registre consumă timp.
- de obicei instrucţiunile sunt neîntreruptibile. O cerere de întrerupere poate fi recunoscută doar la sfârşitul ciclului instrucţiune curent. Există şi excepţii de la această regulă, care se referă la întreruperi software de tip “faults” (erori) şi instrucţiuni cu prefix de repetare a execuţiei. Acestea din urmă pot fi de obicei întrerupte după fiecare execuţie.
- dacă mai mult de o întrerupere sau excepţie aşteaptă terminarea unei instrucţiuni, procesorul le serveşte pe rând, în ordinea priorităţii. În general, cea mai mare prioritate din listă o au excepţiile, sincrone cu programul (nu se încadrează aici excepţiile (trap) folosite pentru a depana un program). Urmează apoi cererile de întrerupere nemascabile. Cea mai mică prioritate o au întreruperile mascabile.

3.6.1. Rolul memoriei stivă

Memoria stivă ("stack memory") este o structură de date specială ce lucrează pe principiul LIFO (Last In First Out = *ultimul intrat - primul ieşit*). Aşa cum s-a văzut anterior, subsistemul de întreruperi, este responsabil pentru salvarea şi restaurarea automată din stivă a informaţiilor vectorului de stare pentru programul întrerupt.

Memoria stivă este utilizată ca memorie de stocare temporară nu numai de sistemul de întreruperi, ci şi de tehnicile de apelare a procedurilor (subrutinelor), sau poate fi folosită de programe pentru stocarea temporară-şi regăsirea rapidă a datelor. De obicei prin stivă se transmit şi alte informaţii, cum ar fi transmiterea parametrilor către proceduri.

O primă metodă de implementare a memoriei stivă (avantajoasă din punctul de vedere al vitezei, dar problematică din punctul de vedere al capacităţii de memorare), constă în folosirea unor *registre interne* ale UCP, (registre cu posibilitatea de deplasare stânga / dreapta a informaţiei). Acest tip de memorie stivă este numită *stiva "construită" hardware*. A doua metodă utilizată la majoritatea microprocesoarelor este implementarea memoriei stivă în cadrul memoriei principale, într-o zonă a cărei dimensiune poate fi satisfăcătore pentru orice aplicaţie (*stiva software*).

Numele de "stivă" sugerează modul de funcţionare al acestei zone de memorare. Astfel fiecare cuvânt nou introdus în stivă se aşează peste anterioarele cuvinte, iar atunci când se doreşte extragerea articolelor din stivă se începe de la "*vârful stivei*" (ultimul articol introdus).

Stiva hardware, reprezentată schematic în figura 3.21, este implementată de obicei cu ajutorul unui set de registre de deplasare (R1 - RN), interne UCP. Vârful stivei (VS) se va găsi întotdeauna în celulele registrelor de pe primul rang binar (c1i, unde i = $\overline{1,N}$), celule cuplate la intrările seriale (I1 - IN). Pentru scriere în stivă (PUSH), se face deplasare "sus-jos" a conţinutului registrelor (informaţia din vârful stivei trece pe următoarea poziţie binară din registre (c2i), iar în vârful stivei apare ultimul articol înscris. La extragere din stivă (POP) informaţia din vârful stivei este citită la ieşirile Y1-YN şi se face deplasare în sens "jos-sus" a informaţiei, pentru fiecare din registrele din figură. Capacitatea de memorare a stivei este limitată la un număr de cuvinte egal cu numărul de celule inclus în fiecare registru component (M în figura 3.21). Numărul de registre folosite determină dimensiunea în biţi a fiecărui cuvânt. De obicei stiva hardware are o adâncime maximă de 4 cuvinte, iar la microprocesoarele unde este implementată oferă o viteză mare de salvare / restaurare a informaţiilor stocate la apelul procedurilor, sau la saltul către rutine de tratare a întreruperilor.

Ultimul rând de celule ce pot stoca informaţie în stivă formează Baza Stivei (BS). Dacă în stiva hardware se introduc mai multe cuvinte decât capacitatea acesteia informaţia cea mai veche

introdusă în stivă se pierde.

Cele mai multe dintre microprocesoarele ce au implementat mecanismul pentru stivă hardware pot administra şi o stivă software.

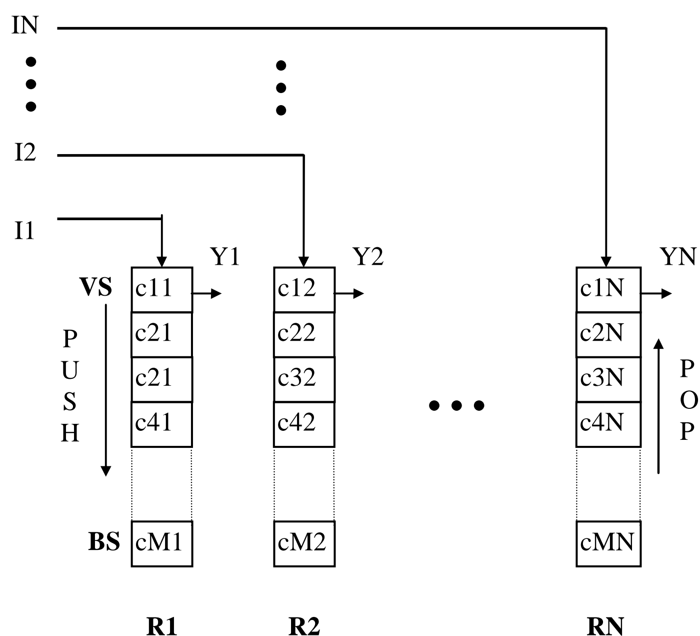


Figura 3.21. Implementarea hardware a stivei, cu ajutorul unui set de registre cu deplasare serială bidirecţională.

În cazul implementării stivei în memoria principală, informaţia nu mai este deplasată sus sau jos, iar gestiunea informaţiilor din stivă este realizată cu ajutorul unui registru ce memorează întotdeauna informaţia de adresare a vârfului stivei. Acest registru este numit *registru Indicator de Stivă (SP - Stack Pointer)*. Stiva construită software, în memoria principală, nu are dimensiuni fixe, acestea putând fi stabilite de programator. Chiar dacă dimensiunea mare este un avantaj, faptul că scrierea sau citirea din stivă înseamnă acces la memoria principală, externă UCP, conduce la o viteză de lucru mai mică decât la stiva hardware.

La înscrierea unui articol în stivă se spune că stiva creşte. Creşterea stivei înseamnă şi reactualizarea conţinutului registrului SP care va conţine întotdeauna adresa vârfului stivei, deci a ultimului cuvânt introdus în stivă, sau adresa primului articol ce a rămas în vârful stivei după o extragere din stivă.

În lumea microprocesoarelor există două convenţii pentru creşterea stivei: creştere către adrese mici şi creştere către adrese mari. Aici vom considera creşterea stivei doar către adrese mici, ceea ce înseamnă că ori de câte ori se introduce un nou articol în stivă acesta va avea adresă mai mică decât anteriorul cuvânt înscris şi că la scriere registrul SP se decrementează. Dacă memoria este organizată pe octet (fiecare octet are o adresă specifică), la scriere, micşorarea

valorii de adresă conţinută în SP constă în scăderea din valoarea iniţială a lui SP a unui număr egal cu numărul de octeţi înscrişi în stivă. În cazul extragerii cuvintelor din stivă operaţiile sunt inverse, conţinutul registrului SP fiind incrementat

Valoarea iniţială a conţinutului lui SP se numeşte ***adresa de bază a stivei*** (baza stivei) şi aceasta se fixează la lansarea programului în execuţie conform directivelor introduse de utilizator în programul sursă. Transferul între UCP şi stiva construită în memoria principală se poate face în două moduri:

- **automat** (de către unitatea de control) la întreruperea unui program prin încărcarea în VS a adresei de revenire şi a altor informaţii de stare. Întreruperea poate să constea într-un salt la o rutină de tratare de întrerupere, sau într-un salt la o procedură.
- **prin instrucţiuni speciale** pentru transfer cu stiva (numite PUSH şi POP ca în figura 3.21, sau MOV - mută în stivă). La fiecare încărcare sau extragere din stivă se actualizează SP.

După cum s-a menţionat la începutul paragrafului memoria stivă este folosită pentru salvare şi restaurare automată nu numai de către sistemul de întreruperi dar şi de apelurile de proceduri. O procedură este o secvenţă de instrucţiuni apelată într-un punct al alteia, executată, după care are loc revenirea în secvenţa apelantă, la adresa imediat următoare celei în care s-a făcut apelarea. Rutinele de tratare a întreruperilor produc o secvenţă similară de acţiuni, dar apelarea nu se face prin instrucţiuni de tip CALL, ci ca urmare a recunoaşterii unui eveniment extern, numit cerere de întrerupere. La unele microprocesoare există în plus instrucţiuni numite "întreruperi software", care de asemenea folosesc stiva şi care se apelează prin furnizarea codului instrucţiunii şi a unei valori (“tip”) de identificare. De exemplu la Intel instrucţiunea se scrie "*INT tip*"

Atunci când se face apel la o procedură, unitatea de control trebuie să facă salvarea valorii curente (adresa de revenire) conţinută în registrului contor de program. Pentru prezentarea lucrului cu memoria stivă de mai jos este deocamdată neimportant dacă informaţia salvată provine dintr-un registru numit contor de program (PC), sau dacă în afară de registrul contor (IP - Instruction Pointer la Intel), în stivă se mai salvează şi conţinutul unui registru de adresă de segment, cum se întâmplă la Intel 8086 pentru proceduri far. După salvare urmează încărcarea PC cu adresa la care se face saltul (adresa unde începe procedura). Procedura se termină cu o instrucţiune de salt de tip RETURN prin care se produce încărcarea automată a PC cu adresa de revenire la programul întrerupt. Alte salvări /restaurări ale variabilelor contextului întrerupt se pot efectua în cadrul procedurii, prin intermediul unor instrucţiuni de înscriere (push) / extragere (pop) din stivă.

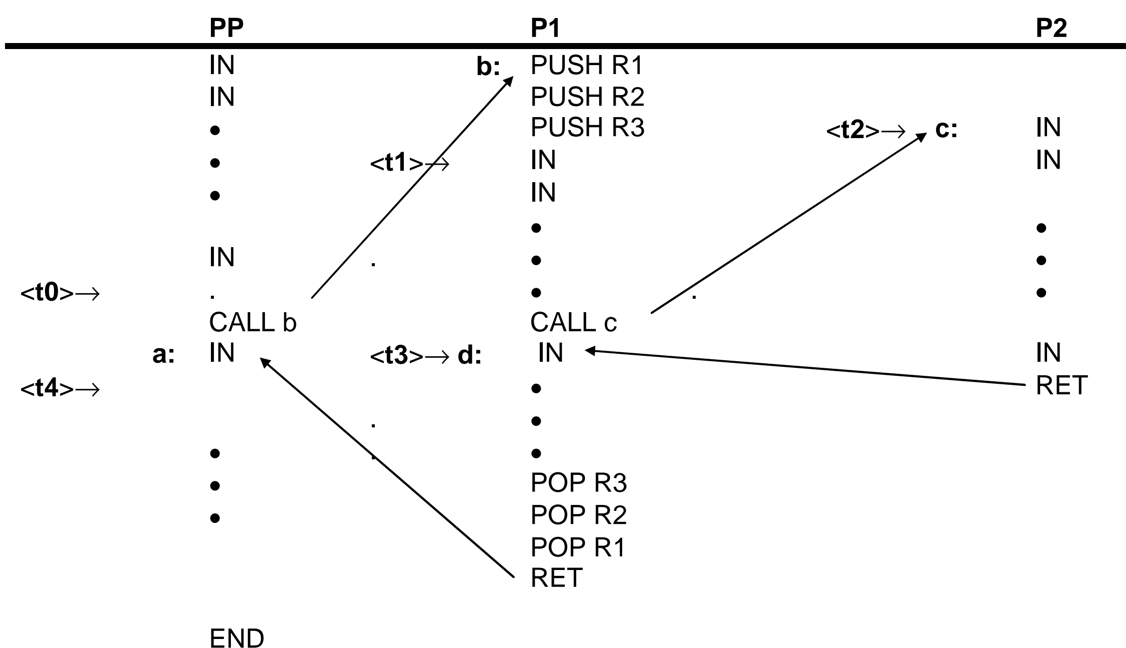
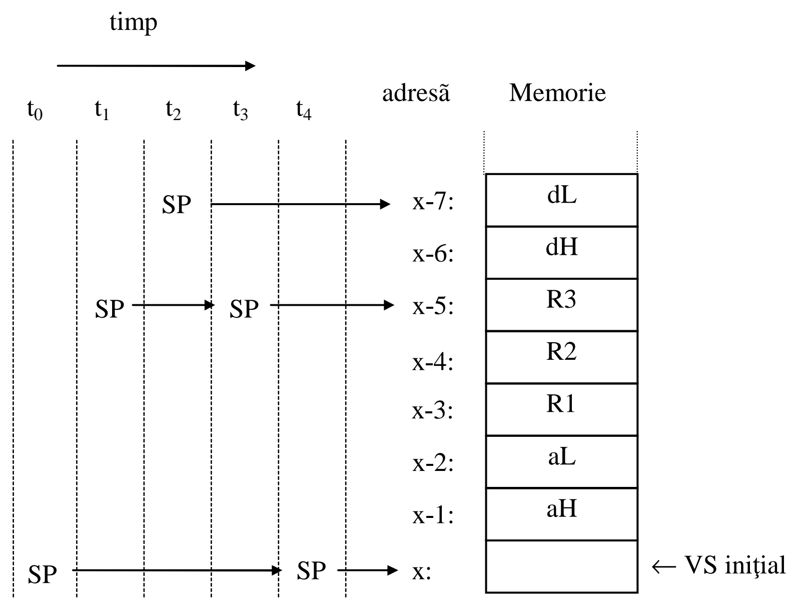


Figura 3.22a. Exemplu de apel a două proceduri notate cu P1 şi P2. Pe figură s-au notat momentele de timp la care în figura 3.22b se analizează conţinutul memoriei stivă.

Pentru a se urmări conţinutul memoriei stivă la apelul procedurilor în figura 3.22a se prezintă un exemplu de apel de proceduri, P1 din programul principal PP şi P2 din procedura P1. În cadrul fiecăreia dintre rutine s-au notat cu IN instrucţiuni oarecare ale programelor (cu excepţia instrucţiunilor de tip call, ret, int, push, pop etc., care afectează şi conţinutul stivei); cu litere mici şi două puncte s-au notat adresele, <t> reprezintă momente de timp succesive (ordinea corespunde cu indicele folosit).

În figura 3.22b s-a prezentat poziţia vârfului stivei la momente de timp succesive.
Se observă că s-a notat baza stivei (adresa iniţială a vârfului stivei cu x). De asemenea s-au notat cu aH si aL, octeţii mai semnificativ (High), respectiv mai puţin semnificativ (Low) ai adresei a. În procedura P1 s-au făcut şi salvări ale conţinutului unor registre (presupuse cu dimensiune jumătate faţă de adresă, în număr de biţi). Restaurarea conţinutului acestora trebuie făcută *în ordine inversă* salvării în stivă, datorită modului de lucru al acesteia.



3. 22b. Conţinutul stivei şi indicarea adresei vârfului stivei pentru exemplul din figura 3.22a.

3.6.2. Utilizarea ferestrelor de registre

Dacă în paragrafele anterioare s-a considerat accepţiunea clasică a salvării şi restaurării informaţiei la o schimbare de context, pentru procesoarele RISC au apărut noi metode de salvare a vectorului de stare al unui program întrerupt şi de asemenea metode noi pentru transferul parametrilor către proceduri. Ideea de bază este să se înlăture accesul repetat la o memorie stivă externă pentru salvarea / restaurarea informaţiilor.

Schimbarea de context presupusă de un apel de procedură duce la o mare risipă de timp, pentru că presupune salvări sau restaurări din memoria stivă construită în memoria principală. La fiecare apelare de procedură variabilele locale programului apelant trebuie salvate din registre în memorie pentru ca registrele să poată fi utilizate la programul apelat. În plus, la apelarea procedurilor trebuie transferaţi şi parametrii către procedură. La revenire trebuie restaurate variabilele programului părinte (încărcate din nou în registre) iar rezultatele trebuie trecute către programul părinte. Din această cauză soluţia este să se introducă un set mare de registre interne procesorului, în care să se poată păstra informaţia necesară schimbării contextului, inclusiv pentru transmiterea de parametrii şi returnarea de valori din proceduri. Mai mult, o bancă de registre locală permite adresarea cu un număr mai mic de biţi, decât în cazul stivei externe, pentru că dimensiunea acestei bănci este în general mică. Astfel strategia este să se păstreze informaţia cât mai mult în registre locale şi să se minimizeze operaţiile de transfer registre - memorie.

Există două abordări pentru rezolvarea acestei probleme. Prima este o abordare software şi se bazează pe un compilator care maximizează utilizarea registrelor în operaţiile necesare de

transfer. Compilatorul va încerca să aloce registre pentru acele variabile care sunt utilizate în cea mai mare perioadă a timpului de execuţie. Această abordare cere utilizarea unor algoritmi sofisticaţi de analiză a programului. Abordarea hardware se referă pur şi simplu la implementarea mai multor registre interne astfel ca să se poată menţine cât mai multe variabile în memoria locală. Abordarea hardware a fost propusă pentru prima oară pentru procesorul RISC1 proiectat şi implementat în anii '70 ai secolului trecut la University of California at Berkeley.

La acest procesor s-a utilizat un set mare de registre interne UCP. Setul conţine 138 de registre de 32 de biţi ce au fost aranjate în 8 ferestre a câte 22 de registre cu suprapunere parţială a câte 6 registre vecine. Se folosesc de asemenea 10 registre globale, pentru toate procedurile active. Fiecare fereastră a fost împărţită astfel încât 6 registre pot fi utilizate pentru transmiterea parametrilor la apelarea procedurilor. Pentru selecţia grupului de 6 registre se modifică doar un registru pointer. Pentru efectuarea unei apelări ("call") sau reveniri ("return") se modifică doar registrul pointer. Numărul mare de registre este necesar pentru a minimiza numărul de transferuri cu exteriorul procesorului. Prin această tehnică a ferestrelor de registre, cu suprapunere parţială apelarea procedurilor poate fi efectuată foarte rapid. Viteza este avantajoasă în special la aplicaţii în timp real unde este necesar un răspuns foarte rapid.

Totuşi există şi dezavantaje, pentru că dacă apelul procedurilor cere mai mult de şase variabile unul dintre registre trebuie să adreseze un vector stocat în memoria externă. Aceste date trebuie încărcate înainte de orice prelucrare iar tehnica ferestrelor de registre pierde mult din performanţa propusă. Dacă se utilizează toate ferestrele suprapuse sistemul poate rezolva situaţia prin urmărirea utilizării ferestrelor şi salvarea în memoria externă fie a unei ferestre fie a întregului set de registre. Acest eveniment poate face ca avantajul ferestrelor de registre locale să fie anulat, mai ales dacă la schimbarea de context trebuie salvate toate cele 138 de registre, pentru că creşte timpul de răspuns.

Principiul ferestrelor de registre presupune că fiecare fereastră are acelaşi număr de registre, pentru fiecare procedură apelantă alocându-se o fereastră. Un apel de procedură va face doar o comutare a setului de registre utilizat de procesor în loc de a salva registrele în memorie. Ferestrele procedurilor adiacente se suprapun pentru a permite transferarea parametrilor. Conceptul este explicat în figura 3.23.

La fiecare moment de timp doar o fereastră de registre este vizibilă şi adresabilă, ca şi cum nu ar exista decât acel număr limitat de registre. Fereastra este împărţită în trei zone de lungime fixă. Registrele de intrare păstrează parametrii primiţi de la programul (părinte) ce a apelat procedura şi de asemenea păstrează rezultatele ce vor fi returnate părintelui. Registrele locale sunt utilizate pentru variabile locale, aşa cum sunt atribuite de compilator. Registrele de ieşire sunt utilizate pentru a schimba parametrii şi rezultate cu următorul nivel inferior (procedură apelată din procedura curentă). Registrele de ieşire de pe un nivel sunt *fizic aceleaşi* cu registrele de intrare de pe nivelul inferior. Această suprapunere permite transmiterea parametrilor fără un

transfer suplimentar de date între programele apelant şi apelat.

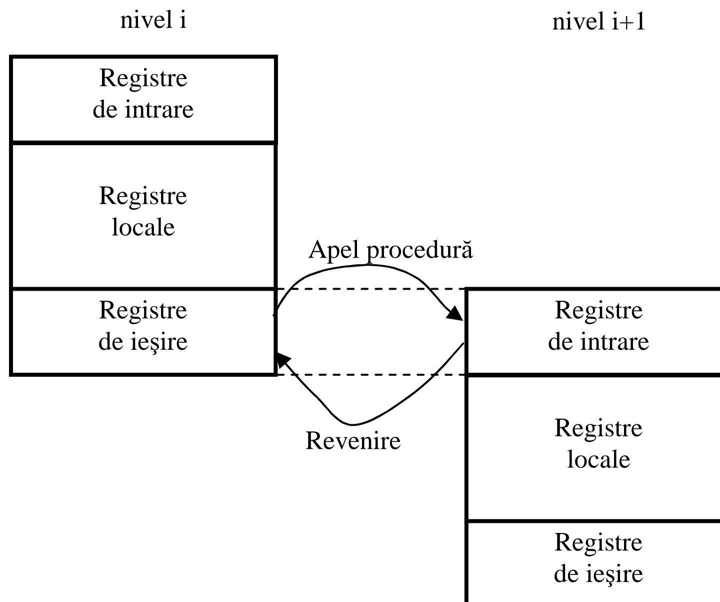


Figura 3.23. Explicativă la suprapunerea ferestrelor de registre

Pentru a permite oricât de multe apelări de proceduri numărul de ferestre ar trebui să fie nelimitat. Această situaţie este rezolvată prin organizarea ferestrelor ca un tampon circular, ultima fereastră fiind suprapusă cu prima, ferestrele de registre păstrând doar cele mai recente proceduri apelate. Cele apelate anterior şi care nu au loc în registrele locale sunt salvate în memoria principală şi apoi retaurate de acolo. Pentru exemplificare vom presupune un set de patru ferestre de registre organizate ca un tampon circular în figura 3.24. În exemplu s-a presupus că procedura P1 (ca părinte) a apelat P2 (ca fiu), astfel că registrul pointer al ferestrei curente (PFC) a ajuns la fereastra 2, care este fereastra procedurii active la momentul actual. Se pot face în continuare apelări de genul P2 apelează (ca părinte) P3 (fiu) şi P3 apelează P4, cu actualizarea registrului PFC. Adresa registrelor în corpul instrucţiunilor procesorului cu ferestre de registre constituie deplasament faţă de acest pointer (PFC), astfel că numărul de biţi alocaţi adresării registrelor este determinat doar de dimensiunea fiecărei ferestre.

Pointerul pentru fereastra salvată identifică cea mai recent fereastră salvată în memoria principală. Astfel dacă procedura P4 ar apela o altă procedură, de exemplu P5, registrul de ieşire al ferestrei 4 (identic cu registrul de intrare al ferestrei 1) ar încerca să scrie parametrii către procedura P5 peste registrele de intrare alocate procedurii P1.

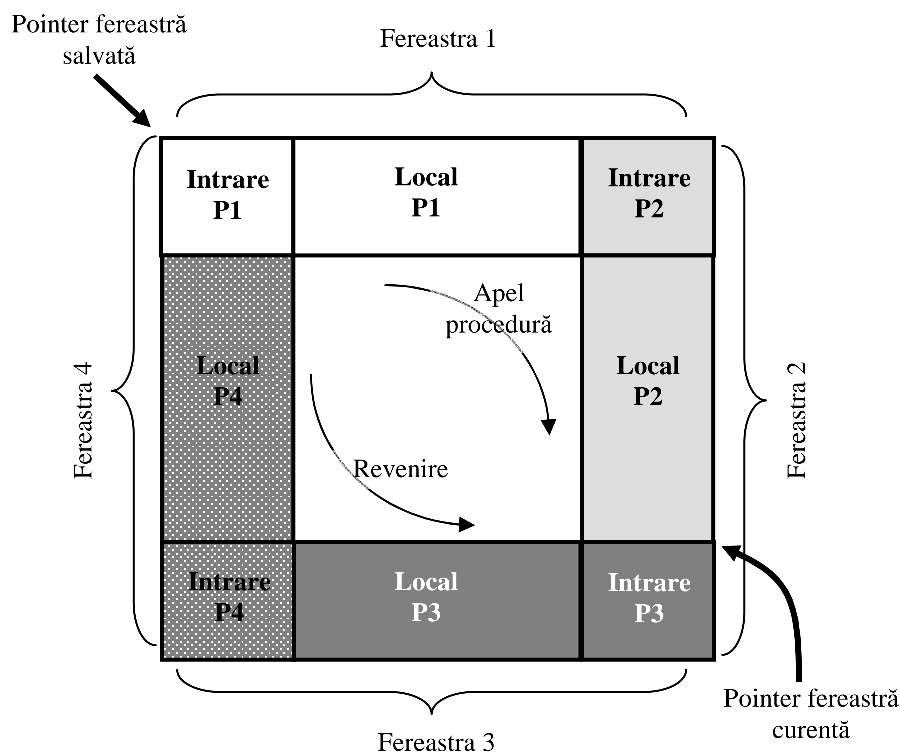


Figura 3.24. Ferestre suprapuse de registre cu organizare circulară şi indicarea sensului de apel al procedurilor respectiv revenire din procedură.

Astfel că atunci când PFC este incrementat (modulo 4 pentru acest exemplu) el devine egal cu pointerul ferestrei salvate ceea ce va produce o întrerupere (excepţie) iar fereastra procedurii P1 este salvată în memoria externă. La revenire, dacă prin depăşirea capacităţii numărului de ferestre a fost necesară salvarea în memoria principală, apare o nouă întrerupere, care va restaura informaţia corespunzătoare procedurii P1. Ori de câte ori se face apelare, registrul PFC se incrementează, iar la revenire (return) se decrementează.

Un bun exemplu al acestei abordări este procesorul SPARC. Procesorul SPARC[24] are o arhitectură RISC pe 32 de biţi dezvoltată de firma Sun Microsystems. Arhitectura de bază este conform modelului RISC Berkeley şi utilizează ferestre de registre pentru a îmbunătăţi comutarea de context şi transmiterea parametrilor.

Se utilizează o bancă de 120 de registre de 32 de biţi împărţită în 7 ferestre de registre plus un set de opt registre disponibile global (16 registre $\times$ 7 ferestre =112 registre plus cele 8 globale). Fiecare fereastră conţine 24 de registre împărţite în trei secţiuni pentru a furniza 8 registre de intrare, 8 registre locale şi 8 registre de ieşire. Cele 8 registre de ieşire furnizează cele 8 intrări pentru următoarea fereastră. Dacă se selectează o nouă fereastră în timpul unei schimbări de

context sau ca urmare a apelării unei proceduri, datele sunt copiate în registrele de ieşire, care constituie registre de intrare pentru procedura noului context de lucru. Ferestrele sunt suprapuse formând un lanţ continuu (circular) ultima fereastră având registre comune cu prima fereastră.
La revenire, informaţia returnată la rutina apelantă este pusă în registrele de intrare şi se execută return.

3.7. Exerciţii

1. Descrieţi pe scurt structura şi funcţionarea unui procesor de uz general (funcţionalitate unitate de control şi unitate de prelucrare, interconectare, sincronizare cu exteriorul, interfaţă cu exteriorul, cicluri caracteristice: stare, ciclu maşină, ciclu instrucţiune)
2. Descrieţi rolul următoarelor componente în funcţionarea unui microprocesor: (a) program counter, (b) registru de indicatori, (c) registru stack pointer.
3. Explicaţi în maximum 100 de cuvinte ce înţelegeţi prin noţiunea de ferestre de registre suprapuse (la arhitecturile RISC) şi rolul structurii.
4. Ce reprezintă operaţia de "corecţie zecimală" în cadrul ALU şi de ce se utilizează ?
5. Enumeraţi indicatorii de condiţii tipici pentru un microprocesor de uz general şi explicaţi pe scurt rolul fiecăruia.
6. Realizaţi sumarea algebrică a celor două numere, indicaţi modul de setare al indicatorilor de condiţii (S, Z, P, V, CY, AC) şi convertiţii în zecimal pentru numere fără semn şi apoi cu semn.
   (a) 0111 1111 + 0111 1111;
   (b) 1000 0001 + 1111 1110;
   (c) 1000 0001 + 1011 1110
7. Descrieţi acţiunile executate de UCP ca răspuns la o cerere de întrerupere
8. Explicaţi deosebirea între întreruperile vectorizate şi cele nevectorizate
9. Clasificaţi şi caracterizaţi în maximum 100 de cuvinte principalele tipuri de întreruperi şi excepţii
10. Descrieţi pe scurt modurile de implementare ale memoriei stivă.
11. Explicaţi afirmaţia: „vârful stivei este identic cu baza stivei”
12. Construiţi o memorie ce foloseşte adresarea complet decodificată cu următorii parametrii: capacitate totală de stocare 128 KB (organizată 64K x 16), circuite integrate de memorie RAM a câte 16 KB.

[24] SPARC = Scalable Processor ARChitecture

13. Construiţi o memorie ce foloseşte adresarea complet decodificată cu următorii parametrii: capacitate totală de stocare 64 KB (organizată 64K x 8), circuite integrate de memorie RAM a câte 16 KB. Conform schemei bloc concepute deduceţi harta adreselor de memorie.
14. Construiţi o memorie ce foloseşte adresarea complet decodificată cu următorii parametrii: capacitate totală de stocare 64 KB (organizată 32K x 16),, circuite integrate de memorie RAM a câte 8 KB. Conform schemei bloc concepute deduceţi harta adreselor de memorie.
15. Indicaţi conţinutul succesiv al contorului de program şi al stivei pentru secvenţa de program de mai jos. Programul începe cu instrucţiunea de la adresa 0.

0	call 9	3	call 5	6	return	9	call 2
1	halt	4	return	7	call 6	10	call 1
2	call 7	5	return	8	return	11	....

16. Indicaţi conţinutul succesiv al contorului de program şi al stivei pentru secvenţa de program de mai jos. Programul începe cu instrucţiunea de la adresa 1.

1	call 6	5	halt	9	call 2	13	return
2	halt	6	call 10	10	call 3	14	halt
3	call 13	7	return	11	return		……….
4	return	8	call 1	12	call 10		

17. Descrieţi indicatorii de condiţii tipici pentru un microprocesor de uz general şi explicaţi pe scurt modul de setare şi rolul fiecăruia.
18. Indicaţi conţinutul succesiv al contorului de program şi al stivei pentru secvenţa de program de mai jos. Programul începe cu instrucţiunea de la adresa 0.

0	call 7	3	call 5	6	add	12	....
1	halt	4	return	7	call 13	13	call 2
2	call 8	5	return	8	return	14	return
							....

Capitolul 4.

Noţiuni privind reprezentarea şi prelucrarea datelor

Conţinut

4.1. Reprezentarea binară a numerelor în calculator

- 4.1.1. Introducere
- 4.1.2. Virgulă fixă - virgulă mobilă
- 4.1.3. Reprezentarea cu virgulă fixă
- 4.1.4. Coduri binare folosite pentru reprezentarea numerelor cu semn

4.2. Coduri zecimal - binare (coduri BCD)

4.3. Reprezentarea numerelor în virgulă mobilă

4.4. Coduri alfanumerice

4.5. Adunare şi scădere pentru numere reprezentate în virgulă fixă

4.6. Suma algebrică a mai multor operanzi reprezentaţi în cod complementar

4.7. Adunarea numerelor reprezentate în virgulă mobilă

4.8. Unitatea de prelucrare a datelor pentru virgulă fixă

4.9. Înmulţirea numerelor binare reprezentate în virgulă fixă

4.10. Circuite de înmulţire construite cu reţele combinaţionale

4.11. Împărţirea numerelor binare

4.12. Exerciţii

4.1. Reprezentarea binară a numerelor în calculator

În cadrul sistemelor numerice de calcul, construite în prezent, informaţia vehiculată, indiferent că reprezintă instrucţiuni (specifice procesorului), adrese sau operanzi, este de tip binar. Reprezentarea se face prin codificarea informaţiilor. Prin codificare, se face o corespondenţă între elementele unei mulţimi de informaţii I, care are **N** elemente (notate {i_1, i_2, ..., i_N}) şi elementele mulţimii binare B={0,1}, astfel încât fiecărui element i aparţinând lui I să-i corespundă o secvenţă de elemente din mulţimea binară. Pentru a codifica cele N elemente ale mulţimii I este necesară o secvenţă de **n** biţi, relaţia între *n* şi *N* fiind:

$$N \leq 2^n \quad (4.1)$$

Pentru descrierea codurilor folosite în calculatoare se va considera că datele sunt organizate în cuvinte cu lungimea de n biţi. Datele stocate în memorie sau în registrele interne ale UCP pot fi clasificate într-una din următoarele categorii:

a. valori numerice folosite ca date propriu-zise în calcule
b. valori numerice care reprezintă operaţii codificate în binar (instrucţiuni) sau valori binare ce codifică adrese.
c. litere ale alfabetului şi semne grafice folosite de obicei pentru legătura cu lumea exterioară (afişare, tipărire, introducere date, etc.)

4.1.1. Introducere

Încercăm în acest paragraf să facem o trecere în revistă, extrem de sumară, a noţiunilor privind bazele de numeraţie 2 şi 10, a reprezentării numerelor în cele 2 baze de numeraţie şi a conversiei între diferitele reprezentări. De exemplu, o valoare constantă egală cu 131 (în baza de numeraţie 10) poate fi reprezentată în binar cu 8 biţi conform figurii 4.1.

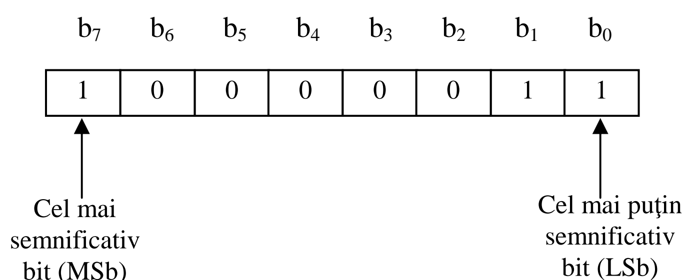
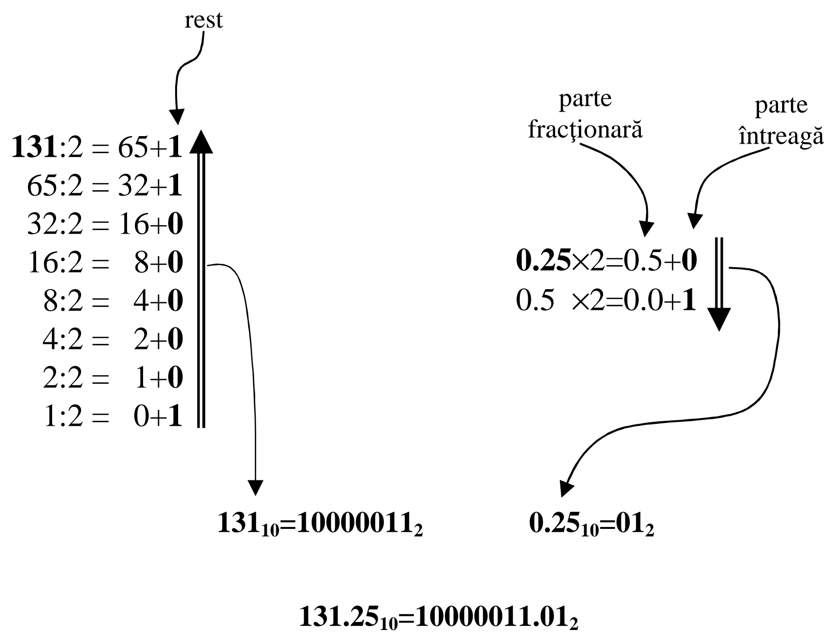


Figura 4.1. Explicativă cu privire la reprezentarea numerelor binare şi la termenii folosiţi pentru a specifica diferitele poziţii binare dintr-un octet.

În figura 4.1 bitul cel mai semnificativ (îl vom nota în continuare **MSb** = Most significant bit, cel mai semnificativ bit) are ponderea cea mai mare (2^7 pentru exemplul din figură), iar bitul cel mai puţin semnificativ (notat în continuare ca **LSb** = Least significant bit, cel mai puţin semnificativ bit), are ponderea 2^0. Reprezentarea pe 8 biţi va fi numită ***octet (byte)***. Notaţia de deasupra valorilor binare indică poziţia bit-ului în cuvântul binar indicând de asemenea prin valoare indicelui *i* şi ponderea valorii binare.

Pentru conversia din zecimal în binar pentru întregi se face împărţire repetată la 2 şi se păstrează în ordine inversă resturile, iar pentru fracţionare din zecimal în binar se face înmulţire repetată cu 2 şi se păstrează valorile întregi.

De exemplu 131.25 în zecimal se scrie:



La conversia inversă, din binar în zecimal se înmulţeşte fiecare bit cu ponderea sa conform relaţiei 4.7. în care M este numărul de biţi ai părţii fracţionare, iar N ai părţii întregi.

$$\sum_{i=M}^{N-1} b_i 2^i \qquad (4.2)$$

De exemplu se poate scrie:

$$10000011.01_2 =$$
$$= 1\times 2^7 + 0\times 2^6 + 0\times 2^5 + 0\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} =$$
$$= 128 + 2 + 1 + \frac{1}{4} = 131.25_{10}$$

4.1.2. Virgulă fixă - virgulă mobilă

Modul în care se reprezintă datele numerice într-un calculator are legătură directă cu structura UCP, având influenţă asupra dimensiunii registrelor de uz general locale, al dimensiunii magistralei interne şi al complexităţii unităţii de execuţie. În funcţie de poziţia virgulei, reprezentările pot fi:

- $\Rightarrow$ reprezentare în virgulă fixă;
- $\Rightarrow$ reprezentare în virgulă mobilă.

Virgula (punctul zecimal sau binar, în notaţia engleza) *nu se reprezintă fizic nicăieri* în registrele calculatorului, dar poziţia sa este cunoscută (stabilită) pentru fiecare dintre modurile de reprezentare.

Reprezentarea în virgulă fixă este folosită de toate calculatoarele numerice de uz general. Majoritatea procesoarelor de uz general lucrează cu numere întregi în virgulă fixă, datorită avantajelor în ceea ce priveşte structura hardware a circuitelor pentru operaţii aritmetice şi logice. Reprezentarea în virgulă fixă cu numere subunitare are avantaje din punctul de vedere al operaţiilor de înmulţire, pentru că această operaţie nu va duce la depăşirea capacităţii de reprezentare pentru produs.

Reprezentarea în virgulă mobilă se face pentru numere reale. Nu toate calculatoarele conţin însă suportul hardware necesar pentru aritmetica în virgulă mobilă, dar acestea pot efectua operaţiile cu numere în virgulă mobilă prin emulare (interpretare) cu ajutorul unor programe de emulare.

Din punctul de vedere al codurilor folosite în calculatoarele numerice, în figura 4.2 se prezintă o *clasificare generală*.

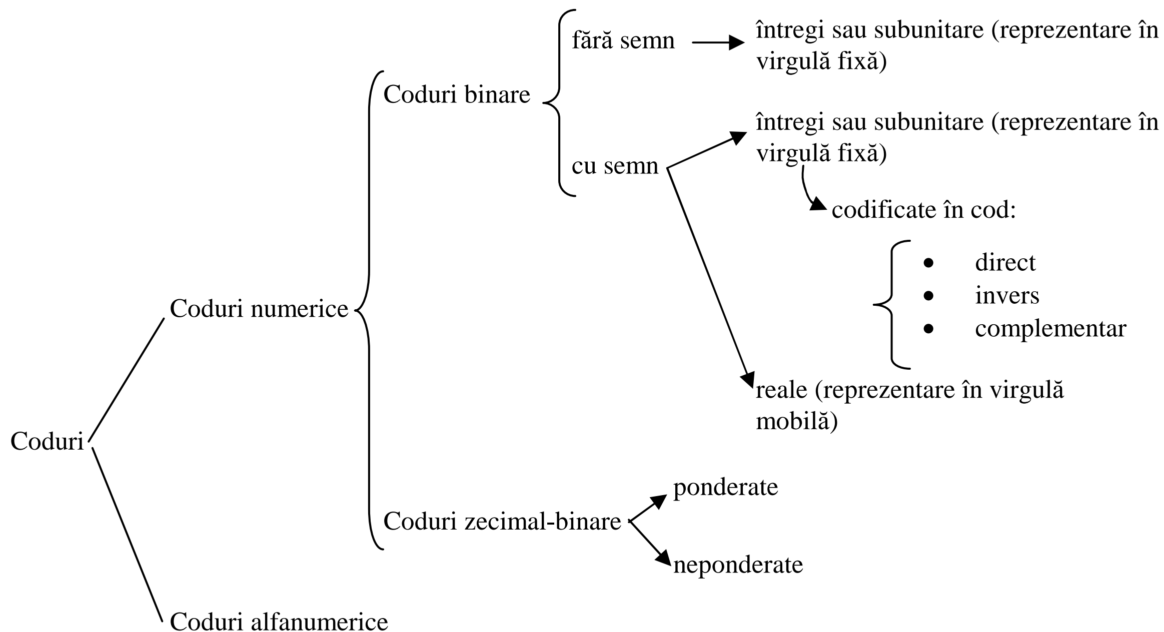


Figura 4.2. Clasificare generală a codurilor folosite pentru reprezentarea datelor în calculatoarele numerice.

4.1.3. Reprezentarea cu virgulă fixă

Reprezentarea se numeşte cu virgulă fixă, pentru că poziţia acesteia este fixă faţă de şirul de biţi prin care se reprezintă data numerică. Reprezentarea numerelor în virgulă fixă se face, în general, sub formă de numere întregi cu semn sau fără semn. Există de asemenea posibilitatea de a reprezenta datele în virgulă fixă sub forma de numere subunitare. Daca procesorul calculatorului lucrează în virgulă fixă cu numere subunitare, la introducerea datelor, fiecărui număr i se ataşează un factor de scară, care să-l transforme într-un număr subunitar. Acest aliniament este făcut de programul de încărcare în memorie. În compilatoarele scrise pentru virgulă fixă, factorul de scară este tratat de compilator, astfel încât utilizatorul limbajelor de nivel înalt nu trebuie să ia în considerare acest aspect la efectuarea operaţiilor.

Pentru numere *întregi fără semn*, reprezentate pe n biţi, se va folosi notaţia:

$$N = b_{n-1}b_{n-2} \ldots b_2 b_1 b_0 \tag{4.3}$$

MSb
Cel mai semnificativ bit

LSb
Cel mai puţin semnificativ bit

unde b_{n-1} este bitul cel mai semnificativ (îl vom nota MSb şi el are ponderea 2^{n-1}), iar b_0 este bitul cel mai puţin semnificativ (notat în continuare ca LSb, cu ponderea 2^0). Se consideră că virgula are poziţia imediat în dreapta bitului LSb (b_0).

În cazul numerelor *subunitare fără semn*, reprezentate pe n biţi, se va folosi notaţia:

$$N_s = b_{-1}b_{-2}b_{-3}\ ...\ b_{-(n-1)}b_{-n} \qquad (4.4)$$

MSb (b_{-1}), LSb (b_{-n})

unde b_{-1} este bitul cel mai semnificativ (MSb cu ponderea 2^{-1}), iar b_{-n} este bitul cel mai puţin semnificativ (LSb cu ponderea 2^{-n}). Se consideră că virgula are poziţia fixă, imediat în stânga MSb (b_{-1}).

Pentru ***numerele cu semn***, întotdeauna pe prima poziţie binară din partea stânga se reprezintă semnul, folosindu-se convenţia:

S = 0, semn pozitiv (N >= 0)
S = 1, semn negativ (N < 0) (4.5)

Pentru numere *întregi cu semn*, reprezentate pe n biţi, se va folosi notaţia:

$$N = b_{n-1}b_{n-2}\ ...\ b_2b_1b_0 \qquad (4.6)$$

Semn (b_{n-1}), MSb (b_{n-2}), LSb (b_0)

identică cu cea din relaţia (4.3), dar la care semnificaţia biţilor este diferită. Bitul b_{n-1} este bitul de semn, bitul b_{n-2} este cel mai semnificativ bit (MSb, cu ponderea 2^{n-2}), iar b_0 este bitul cel mai puţin semnificativ (LSb). Se consideră că virgula are poziţia fixă imediat în dreapta LSb (b_0).

Pentru numerele *subunitare cu semn*, reprezentate pe n biţi, se va folosi notaţia:

$$N_s = b_0b_{-1}b_{-2}b_{-3}\ ...\ b_{-(n-1)} \qquad (4.7)$$

Semn (b_0), MSb (b_{-1}), LSb ($b_{-(n-1)}$)

unde b_0 este bitul de semn, b_{-1} este bitul cel mai semnificativ (MSb, cu ponderea 2^{-1}), iar $b_{-(n-1)}$ este bitul cel mai puţin semnificativ (LSb cu ponderea $2^{-(n-1)}$). Se consideră că virgula se află între bitul de semn şi MSb (b_{-1}).

4.1.4. Coduri binare folosite pentru reprezentarea numerelor cu semn

În electronica digitală se folosesc trei tipuri de coduri binare:

- cod *direct* (numit şi cod *mărime / modul şi semn***,** notat pe scurt **MS**)
- cod *invers* (numit şi cod *complement faţă de 1***,** notat pe scurt **C1**)
- cod *complementar* (numit şi cod *complement faţă de 2***,** notat pe scurt **C2**)

Se va face o descriere sintetică a celor 3 moduri de codificare considerând că numerele sunt reprezentate cu n biţi. De asemenea specificăm încă de la început că *pentru numere pozitive toate cele trei moduri de reprezentare sunt identice* (conduc la aceleaşi numere în binar).

4.1.4.a Reprezentarea numerelor cu semn în cod direct (cod mărime şi semn, MS)

Mod de reprezentare: Modulul numărului are aceeaşi reprezentare atât pentru numere pozitive cât şi pentru cele negative, iar semnul ia valoarea 0 pentru numere pozitive, respectiv 1 pentru numere negative.

numere întregi	numere subunitare	
$N = b_{n-1}\, b_{n-2} \ldots.\, b_1\, b_0$	$N_s = b_0\, b_{-1}\, b_{-2}\, b_{-3} \ldots\, b_{-(n-1)}$	
pentru $N > 0$, $b_{n-1} = 0$ pentru $N < 0$, $b_{n-1} = 1$	pentru $N_s > 0$, $b_0 = 0$ pentru $N_s < 0$, $b_0 = 1$	(4.8)

Domeniul de reprezentare:

numere întregi	numere subunitare	
$-(2^{n-1}-1) \leq N \leq 2^{n-1}-1$	$-(1-2^{1-n}) \leq N_s \leq (1-2^{1-n})$	(4.9)

Calculul valorii zecimale pentru numărul reprezentat în binar în MS:

numere întregi	numere subunitare	
$N\big\|_{10} = (-1)^{b_{n-1}} \sum_{i=0}^{n-2} b_i 2^i$	$N_s\big\|_{10} = (-1)^{b_0} \sum_{i=1}^{n-1} b_{-i} 2^{-i}$	(4.10)

Avantaje ale codului MS:

Codificarea în MS prezintă avantaje din punctul de vedere al operaţiilor de înmulţire şi împărţire, algoritmii de calcul şi rutinele sau circuitele de implementare cablată rezultând cu o structură simplă. De asemenea codificarea în MS se face foarte simplu pentru numerele negative, doar prin modificarea bitului de semn.

Dezavantaje ale codului MS:

Codul prezintă dezavantaje din punctul de vedere al operaţiilor de adunare şi scădere. Asta în primul rând pentru că numerele pozitive şi negative trebuie tratate diferit în operaţii şi ca urmare rezultă algoritmi de adunare şi scădere mai complecşi decât în alte moduri de codificare.
Un alt dezavantaj îl reprezintă faptul ca există două reprezentări pentru numărul zero:

000000...0 şi 100000...0

Exemplu (pentru n=8):

Pentru n=8, domeniul de reprezentare este:

$|N| \leq 127$ pentru întregi şi $|N_S| \leq 1 - 2^{-7}$ pentru subunitare.

zecimal	binar
N = +6 sau N = +6/128	0000 0110 (primul zero este bitul de semn)
N = -6 sau N = -6/128	1000 0110 (primul unu este bitul de semn)

Se observă că numerele binare 00000110, respectiv 10000110 pot fi interpretate şi ca +6/128, (pentru alte valori ale lui n valoarea este $+6/2^{(n-1)}$)respectiv -6/128, dacă reprezentarea se face cu numere subunitare.

Calculul valorii zecimale pentru N = 1000 0110:

N = 1000 0110:

$$N|_{10} = (-1)^1 (0*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0) = -6$$

sau

$$N|_{10} = (-1)^1 (0*2^{-1} + 0*2^{-2} + 0*2^{-3} + 0*2^{-4} + 1*2^{-5} + 1*2^{-6} + 0*2^{-7}) =$$
$$= - (1/32 + 1/64) = - 6/128$$

4.1.4.b Reprezentarea numerelor cu semn în cod invers
(cod complement faţă de 1, C1)

Mod de reprezentare: Pentru numere pozitive reprezentarea este identică cu cea făcută în MS. Pentru numere negative, bitul de semn este 1, iar biţii ce reprezintă valoarea absolută a numărului sunt formaţi prin complementarea bit cu bit a reprezentării în cod direct.

	numere întregi	numere subunitare	
numere pozitive	$N = 0\ b_{n-2} \ldots\ b_1\ b_0$	$Ns = 0\ b_{-1}\ b_{-2}\ b_{-3} \ldots\ b_{-(n-1)}$	
numere negative	$N = 1\ a_{n-2} \ldots\ a_1\ a_0$	$Ns = 0\ a_{-1}\ a_{-2}\ a_{-3} \ldots\ a_{-(n-1)}$	
	unde $a_i = 1-b_i$, (i=0.n-2)	unde $a_i = 1-b_i$, (i=1.n-1)	(4.11)

Reprezentarea în cod invers a unui număr N, binar negativ, se poate face în două moduri:

a) se complementează toţi biţii reprezentării modulului numărului (|N|) inclusiv bitul de semn;
b) pe baza relaţiilor generale:

numere întregi	numere fracţionare	
$(N)_{C1} = 2^n - \lvert N \rvert - 1$	$(N_f)_{C1} = 2 - \lvert N_S \rvert - 1/(2^{(n-1)})$	(4.12)

Domeniul de reprezentare:

numere întregi	numere subunitare	
$-(2^{n-1}-1) \leq N \leq 2^{n-1}-1$	$-(1-2^{1-n}) \leq N_s \leq (1-2^{1-n})$	(4.13)

Calculul valorii zecimale pentru numărul reprezentat în binar în C1:

numere întregi	numere subunitare	
$N\big\rvert_{10} = -a_{n-1}(2^{n-1} - 1) + \sum_{i=0}^{n-2} a_i 2^i$	$N_s\big\rvert_{10} = -a_0\left(2^0 - 1/2^{n-1}\right) + \sum_{i=1}^{n-1} a_{-i} 2^{-i}$	(4.14)

(relaţiile au fost scrise în acest fel pentru a putea fi folosite şi pentru numerele pozitive)

Avantaje şi dezavantaje ale codului C1:

Reprezentarea este foarte uşor de realizat în hardware, dar algoritmii de înmulţire şi împărţire pentru numere în C1 sunt mai complecşi decât la MS. Codul prezintă unele avantaje pentru adunare şi scădere, dar la aceste operaţii se preferă codul complementar.

Un alt element dezavantajos este faptul că există două reprezentări pentru zero (plus şi minus zero):

000000...0 şi 111111...1

Exemplu (pentru n=8):

zecimal	binar
N = +6 sau N = +6/128	0000 0110
N = -6 sau N = -6/128	1111 1001

$(-6)_{C1} = 2^8 - |N| - 1$ = 1111 1111 -
0000 0110

1111 1001

Calculul valorii zecimale pentru $(N)_{C1}$ = 1111 1001:

$N|_{10} = (-1) * 2^7 + 1 + (1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0) = -6$

sau

$N|_{10} = -1 + 1*2^{-7} +(1*2^{-1} + 1*2^{-2} + 1*2^{-3} + 1*2^{-4} + 0*2^{-5} + 0*2^{-6} + 1*2^{-7}) =$

$= -1 + 1/2 + 1/4 + 1/8 + 1/16 + 1/128 + 1/128 = -6/128$

4.1.4.c Reprezentarea numerelor cu semn în cod complementar
(cod complement faţă de 2, C2)

Mod de reprezentare: Pentru numere pozitive reprezentarea este identică cu cea făcută în MS. Pentru reprezentarea în cod complementar a unui număr N, binar negativ, se pot folosi variantele descrise în tabelul de mai jos:

varianta	**numere întregi**	**numere subunitare**	
a	$(N)_{C2} = (N)_{C1} + 1$	$(N_s)_{C2} = (N_s)_{C1} + 1/(2^{(n-1)})$	(4.15)
	sau prin înlocuirea expresiei pentru C1:		
b	$(N)_{C2} = 2^n - \|N\|$	$(N_s)_{C2} == 2 - \|N_s\|$	(4.16)
c	Folosind reprezentarea modulului numărului negativ de codificat, pornind de la dreapta (de la LSb) către stânga se copiază toţi biţii inclusiv primul bit 1 întâlnit, iar apoi se complementează toţi ceilalţi biţi inclusiv bitul de semn.		

Note:

- *La operaţiile de scădere sau adunare pentru calcularea codului se consideră şi transportul sau împrumutul.*
- *Relaţiile de la varianta b indică de ce codul poartă numele complement faţă de 2.*
- *Metoda de la varianta c este des folosită în codificarea manuală şi de asemenea uşor de implementat cu porţi logice.*

Domeniul de reprezentare:

numere întregi	**numere subunitare**	
$-2^{n-1} \leq N \leq 2^{n-1}-1$	$-1 \leq N_s \leq 1-2^{1-n}$	(4.17)

Calculul valorii zecimale pentru numărul reprezentat în binar în cod C2:

numere întregi	**numere subunitare**	
$N\Big\|_{10} = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$	$N_s\Big\|_{10} = -b_0 + \sum_{i=1}^{n-1} b_{-i} 2^{-i}$	(4.18)

(relaţii valabile şi pentru numere pozitive)

Avantaje şi dezavantaje ale reprezentării în codul C2. Reprezentarea în cod complementar a numerelor negative s-a generalizat la procesoarele actuale, datorită avantajelor următoare:

- Scăderea unui număr din alt număr este echivalentă matematic cu adunarea complementului de doi a scăzătorului la descăzut. Se implementează doar operaţia de adunare a numerelor reprezentate în cod complementar (sumare algebrică).
- Codificarea în C2 printr-un circuit electronic este foarte uşor de realizat (vezi un exemplu în figura 4.3).

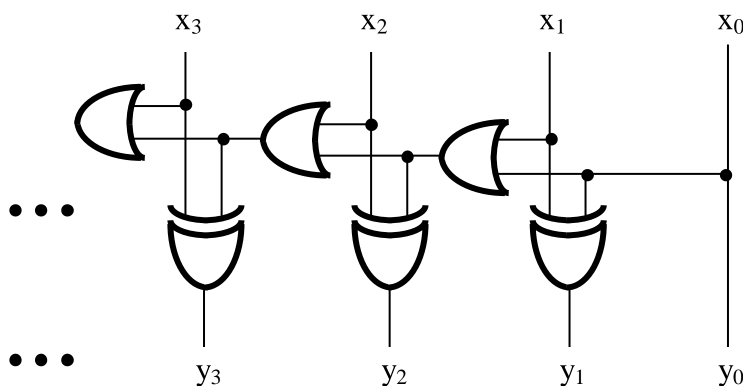


Figura 4.3. Schemă simplificată a unui circuit care transformă un număr binar întreg pozitiv (introdus la intrările ... $x_3x_2x_1x_0$) în codul complement faţă de 2 pentru echivalentul negativ al numărului.

- Acest cod are o singură reprezentare pentru zero (00...0), (având deci un cod în plus faţă de celelalte coduri, cu acelaşi număr de biţi).
- Un întreg în reprezentarea complement faţă de doi poate fi uşor extins la un format mai mare (un număr mai mare de biţi) fără schimbarea valorii sale. Pentru aceasta este necesar doar ca bitul de semn sa fie repetat în toate poziţiile binare de ordin superior apărute. Această ; operaţie se numeşte *extensie de semn*. Exemplu pentru un întreg pe 16 biţi extins apoi la 32 de biţi:

16b: Snnn nnnn nnnn nnnn (S: semn; n: 0 sau 1)
32b: SSSSSSSSSSSSSSSS Snnn nnnn nnnn nnnn

Principalul dezavantaj al codului complementar îl constituie algoritmii pentru înmulţire şi împărţire mai complecşi decât cei corespunzători codului MS.

Datorită avantajelor pe care le prezintă faţă de celelalte două coduri, codul complementar (complement faţă de 2) este în prezent singurul cod folosit pentru reprezentarea numerelor cu semn în calculatoarele digitale.

Exemplu de conversie în C2, (pentru n=8):

$N|_{10} = -6$

a) cu C1: $(-6)_{C2} = (-6)_{C1} + 1$

```
1111 1001 +
        1
-------------
1111 1010
```

b)

$(-6)_{C2} = 2^8 - |6|$

```
1 0000 0000 -
  0000 0110
----------------
  1111 1010
```

c) pe reprezentarea lui +6 = 0000 0110

←

1111 1010

primul 1 copiat, ceilalţi complementaţi

Calculul valorii zecimale pentru $(-6)_{C2}$ = 1111 1010:

$N|_{10} = (-1) * 2^7 + (1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0) = -6$

sau

$N|_{10} = -1 + (1*2^{-1} + 1*2^{-2} + 1*2^{-3} + 1*2^{-4} + 0*2^{-5} + 1*2^{-6} + 0*2^{-7}) =$

$= -1 + 1/2 + 1/4 + 1/8 + 1/16 + 1/64 = -6/128$

Pentru comparaţia numerelor codificate în cele trei coduri pentru întregi cu semn, în figura 4.4 se prezintă tabela codurile pe 4 biţi (3 de mărime şi unul de semn).

Aşa cum s-a spus anterior operaţia de scădere poate fi înlocuită cu sumarea algebrică, dacă numerele sunt reprezentate în cod complementar. Astfel A - B se poate scrie ca A + (-B) unde ambele numere sunt reprezentate cu semn şi în complement faţă de 2. Sumarea se face inclusiv pentru bitul de semn, care nu necesită o abordare specifică.

Număr zecimal	**Cod binar**		
	direct	**invers**	**complementar**
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	***0000***	***0000***	***0000***
-0	***1000***	***1111***	***0000***
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8			1000

Figura 4.4. Comparaţie între codurile pentru reprezentarea cu 4 biţi a numerelor cu semn.

4.2. Coduri zecimal - binare (coduri BCD)

Acestea sunt coduri pentru reprezentarea cifrelor zecimale, de la 0 la 9. De aceea traducerea mai corectă în limba română ar fi "zecimal codificat binar", dar s-a răspândit foarte mult denumirea de coduri zecimal-binare. În continuare vom folosi mai ales denumirea de coduri BCD (Binary Coded Decimal). Pentru codificare sunt necesari cel puţin patru biţi. În funcţie de principiul de construcţie a codurilor, acestea se pot împărţi în:

- coduri BCD ponderate
- coduri BCD neponderate

La codurile *ponderate* calculul oricărei cifre zecimale, c = {0,1,2,...,9}, se poate face pe baza ponderilor zecimale p_i ce apar în denumirea codului, conform relaţiei:

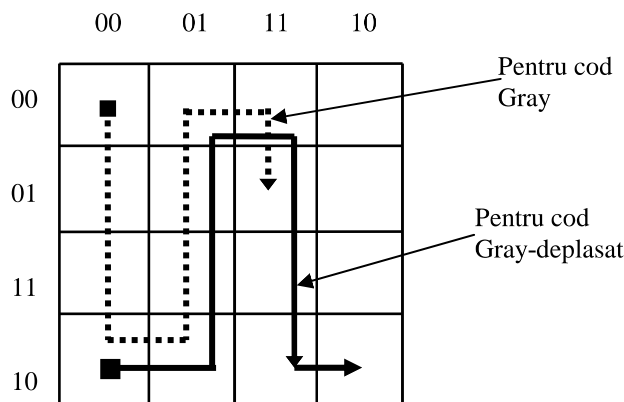
$$c = \sum_{i=0}^{3} b_i \cdot p_i \qquad (4.19)$$

unde b_i este valoarea binară ce corespunde ponderii *i*. La codurile *neponderate* nu se respectă relaţia (4.19). Construcţia lor este făcută pentru a asigura corectitudinea datelor reprezentate, în diferite aplicaţii digitale, prin introducerea în reprezentare a unor constrângeri ce permit verificarea corectitudinii codurilor.

În figura 4.5 se prezintă câteva coduri ponderate şi neponderate. Codul 8421, numit de asemenea şi cod NBCD (Natural Binary Coded Decimal) - zecimal codificat în binar natural, foloseşte pentru a codifica fiecare cifră zecimală cu reprezentarea din numărarea binară naturală.

Cifra zecimală	**Coduri BCD**					
	Ponderate			**Neponderate**		
	8421	**4221**	**2421**	**Gray**	**Gray deplasat**	**Exces 3**
0	0000	0000	0000	0000	0010	0011
1	0001	0001	0001	0001	0110	0100
2	0010	0010	0010	0011	0111	0101
3	0011	0011	0011	0010	0101	0110
4	0100	0110	0100	0110	0100	0111
5	0101	1001	1011	0111	1100	1000
6	0110	1100	1100	0101	1101	1001
7	0111	1101	1101	0100	1111	1010
8	1000	1110	1110	1100	1110	1011
9	1001	1111	1111	1101	1010	1100

(a)



(b)

Figura 4.5. (a) Coduri BCD ponderate şi neponderate. (b) posibilitatea de deducere a codului Gray, pe baza diagramei Karnaugh, ce se bazează pe proprietatea de adiacenţă.

Codul NBCD este codul BCD folosit în majoritatea procesoarelor de uz general, pentru operaţii în ALU, motiv pentru care în documentaţia acestor procesoare codul NBCD este numit doar "BCD", pentru că nu se mai foloseşte şi alt cod pentru numere zecimale.

Codurile 2421 (cod Aiken) şi 4221 sunt coduri auto-complementare. (codul lui 5 este complementul bit cu bit al codului lui 4, codul lui 6 este complementul codului lui 3, codul lui 7 este complementul codului lui 2, etc.). Diferenţa între cele două moduri de codificare se referă doar la cifrele 4 şi 5.

Codul exces 3 nu prezintă nici o valoare de cod cu toţi biţii 0, lucru important în anumite aplicaţii. Codul Gray este un cod ce are proprietatea de adiacenţă, fiind avantajos în metode de minimizare sau în aplicaţii unde pe baza adiacenţei se poate verifica corectitudinea informaţiei (de exemplu traductoare incrementale). Codul Gray simplu nu prezintă însă adiacenţă între codurile pentru 0 şi pentru 9 (primul şi ultimul cod) şi de aceea uneori se foloseşte codul Gray deplasat, căruia nu-i mai lipseşte proprietatea amintită. Cu privire la codul Gray, în figura 4.5.b s-a desenat o diagramă Karnaugh (de asemenea cu proprietatea de adiacenţă), din care se pot deduce cele două tipuri de cod Gray. Codul Gray a fost descris aici la codurile BCD, dar uneori el este folosit cu toate cele 16 combinaţii binare, în acest ultim caz nefiind un cod BCD. Pentru codul Gray care codifică numerele zecimale de la 0 la 15 ultima combinaţie binară este 1000 (pentru 15 zecimal) fiind adiacentă cu 0000, aşa cum se observă în figura 4.5.b completând linia punctată până la ultima căsuţă din diagramă. Acest cod este numit adesea şi cod binar reflectat, pentru că valorile binare ale ultimilor 3 biţi se reflectă faţă de o linie imaginară trasă între codurile din mijloc. Astfel pentru codul Gray cu 16 combinaţii reflectarea se produce faţă de linia trasă între codurile cifrelor zecimale 7 şi 8, iar pentru codul *BCD Gray deplasat* faţă de linia trasă între codurile pentru 4 şi 5. De exemplu, în figura 4.5.a oglindirea se produce pentru cei 3 biţi mai puţini semnificativi pentru 4 cu 5, 3 cu 6, 2 cu 7, 1 cu 8 şi 0 cu 9.

4.3. Reprezentarea numerelor în virgulă mobilă

Pentru stocarea şi prelucrarea numerelor reale în calculator se foloseşte reprezentarea în virgulă mobilă. Denumirea provine de la faptul că virgula nu are o poziţie fixă faţă de şirul de biţi ce reprezintă valoarea numărului. Poziţia virgulei se poate modifica fie ca să permită efectuarea de operaţii aritmetice şi logice cu numere reale, fie pentru reprezentarea standardizată, în virgulă mobilă, a numerelor reale.

În calculatorul numeric, pentru fiecare număr real N trebuie reprezentate fiecare din componentele următoarei relaţii:

$$N = M \times B^{E} \tag{4.20}$$

unde

M = *mantisa*, reprezentată în calculatorul numeric ca un număr binar subunitar cu semn;

E = *exponentul*, reprezentat ca un număr întreg cu semn;

B = *baza*, este 2 sau o putere a lui 2, nu este reprezentată în calculatorul numeric, dar se ţine cont de valoarea bazei atunci când se efectuează operaţii aritmetice. La un anumit calculator numeric valoarea bazei este aceeaşi pentru toate numerele reale reprezentate.

Reprezentarea numerelor în calculator se face prin cuvinte binare cu lungime egală cu lungimea registrelor de stocare. Precizia de reprezentare a unui număr real este dată în primul rând de numărul de biţi folosiţi pentru reprezentarea **M**antisei. Domeniul maxim de reprezentare este determinat de valoarea adoptată pentru **B**ază şi de numărul de biţi folosiţi pentru a reprezenta **E**xponentul.

Spre deosebire de numerele întregi, numerele reale se reprezintă cu o anumită aproximaţie în calculator. Dacă pentru reprezentarea numerelor reale în calculator se folosesc n biţi, se vor putea reprezenta maxim 2^n numere, dintr-un anumit interval continuu $\pm$R al numerelor reale. Ca urmare, numerele reale sunt reprezentate cu o anumită eroare, determinată de numărul limitat de biţi folosit pentru reprezentare. *Eroarea de reprezentare* depinde de distanţa dintre două numere succesive reprezentabile cu cei n biţi. Toate numerele reale cuprinse între cele două valori vor trebui aproximate prin una din cele două valori. Dacă baza folosită implicit pentru reprezentare nu este 2 ci 4 sau 16, dezavantajul este că numerele reprezentabile vor fi şi mai rar distribuite pe intervalul $\pm$R deşi există avantajul creşterii domeniului de reprezentare cu n biţi.

Pentru reprezentarea mantisei numerelor în virgulă mobilă se foloseşte *forma normalizată* în care, pentru reprezentarea în mărime şi semn, prima poziţie binară după virgulă este diferită de zero. Normalizarea restrânge mărimea valorii absolute a mantisei binare la domeniul:

$$\frac{1}{2} \leq |M| < 1 \tag{4.21}$$

Normalizarea în binar este uşor de făcut, prin deplasarea mantisei către dreapta sau stânga şi incrementarea respectiv decrementarea corespunzătoare a exponentului.

De exemplu, numărul cu mantisa normalizată $0.11011000 * 2^{10}$ poate fi scris:

$$0.11011000 * 2^{10} = 0.011011000 * 2^{11} = 0.0011011000 * 2^{12}$$

Dacă, de exemplu, considerăm B=2, exponentul E reprezentat cu semn pe 8 biţi, iar mantisa normalizată un număr cu semn reprezentat pe 24 de biţi în cod mărime şi semn (deci numărul stocat în calculator va avea 8 + 24 = 32 biţi), domeniul maxim de reprezentare va fi cuprins între -2^{127} şi $+ 2^{127}$.

În valoare absolută, cel mai mic număr reprezentabil în calculator este

$$N_{minim_reprezentabil} = 0.100...0 * 2^{-127}$$

iar cel mai mare număr reprezentabil este

$$N_{maxim_reprezentabil} = 0.111...1 * 2^{+127}$$

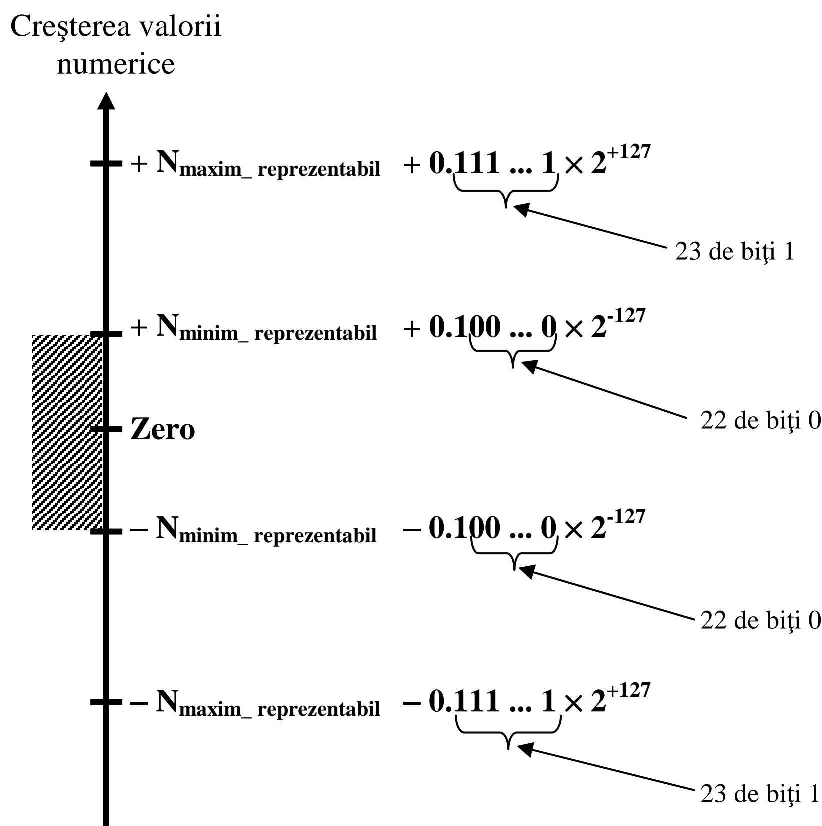


Figura 4.6. Scară a numerelor reale pentru reprezentarea pe 32 de biţi, din care pentru mantisă sunt rezervaţi 24 biţi = 23 mărime + 1 semn.

Reprezentând grafic toate numerele pe o scară, la care în partea de sus se reprezintă numerele mari, iar jos cele mai mici, ca în figura 4.6. se pot face câteva observaţii importante:

- Mantisele se reprezintă pe 24 biţi, din care 23 sunt rezervaţi pentru mărime.
- Toate mantisele reprezentate în cod direct în exemplul din figura 4.6 sunt normalizate (primul bit după virgulă este 1). Dacă în urma efectuării operaţiilor aritmetice rezultatul este în afara domeniului [$-N_{maxim_reprezentabil}$, $+N_{maxim_reprezentabil}$] se spune ca s-a produs o depăşire a capacităţii de reprezentare (pentru cei 32 de biţi din exemplu). Depăşirea poate fi superioară (mai mare decât numărul maxim reprezentabil) sau inferioară. Programul în care se produce depăşirea se întrerupe, iar evenimentul este tratat de o rutină (de servire a cazului de excepţie "depăşire"), care va transmite un mesaj către utilizator. Daca însă rezultatul se încadrează în intervalul cu numere prea mici pentru a fi reprezentate cu

mantise normalizate [-$N_{minim_reprezentabil}$, +-$N_{minim_reprezentabil}$] numărul este aproximat, de obicei, cu zero.

- Reprezentarea lui zero pune unele probleme, pentru că mantisa poate rezulta zero (nenormalizată) dar exponentul său poate avea orice valoare pentru că $0 \times B^E = 0$ pentru oricare valoare a lui E. Din punctul de vedere a circuitelor care testează dacă un număr este egal cu zero, ar fi avantajos ca să avem aceeaşi reprezentare ca la întregi (succesiune numai de biţi 0 în toate câmpurile: semn, exponent, mantisă). Valoarea zero este considerată a fi cea care corespunde la exponentul maxim negativ (-127 în exemplul de mai sus).
- De asemenea, pune probleme de reprezentare faptul că este necesar să se reprezinte două semne pentru fiecare număr (semnul mantisei şi semnul exponentului).

O cale de rezolvare a problemei reprezentării numărului zero este folosirea *exponentului deplasat* (sau *caracteristică*, notată în continuare cu C) cu valoarea constantă K:

$$C = E + K \tag{4.22}$$

Biţii exponentului	**Valoarea zecimală fără semn pentru C**	**Valoare cu semn E**	
		K=127	**K=128**
111...11	255	+128	+127
111...10	254	+127	+126
...	...	...	...
100...01	129	+2	+1
100...00	128	+1	0
011...11	127	0	-1
011...10	126	-1	-2
...	...	...	...
000...01	1	-126	-127
000...00	0	-127	-128

Figura 4.7. Exponent pe 8 biţi cu deplasament 127 (cod exces 127) şi 128 (cod exces 128).

Deplasarea, pentru reprezentare pe 8 biţi a exponentului se poate face cu K=127 (2^{n-1}-1) sau cu K=128 (2^{n-1}). Ambele variante sunt exemplificate în tabelul din figura 4.7.

Dacă în exemplul prezentat mai sus deplasarea se face conform relaţiei:

$$C = E + 127 \tag{4.23}$$

caracteristica poate lua valori intre 0 şi 254.

În acest caz, formatul reprezentării pe 32 biţi în virgulă mobilă poate fi cel din figura 4.8.

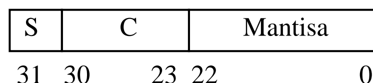


Figura 4.8. Reprezentare în virgulă mobilă pe 32 biţi cu exponent deplasat caracteristică (C) şi semnul mantisei (S).

În plus, dacă mantisa este reprezentată în cod direct şi este normalizată, primul bit al acesteia este 1 pentru toate numerele. Acest prim bit poate să nu mai fie reprezentat (bit ascuns, "hidden bit" - HB). Ca urmare, în formatul pe 32 de biţi, mantisa poate fi reprezentată pe 24 de biţi (23 biţi reprezentaţi efectiv + 1 HB) precizia de reprezentare crescând de două ori. De asemenea, faptul că bitul 1 este ascuns, permite numerelor pozitive foarte mici ($+0.1 * 2^{-127}$, reprezentat cu HB) să aibă numai zerouri pe toţi cei 32 de biţi, deci reprezentarea numărului zero este identică cu cea de la reprezentarea întregilor, putându-se folosi aceleaşi tipuri de circuite de detecţie a rezultatului nul.

Ca urmare, la efectuarea operaţiei de normalizare, virgula mobilă este mutată până când în stânga sa se găseşte un bit 1, iar ceilalţi biţi se găsesc în dreapta virgulei.

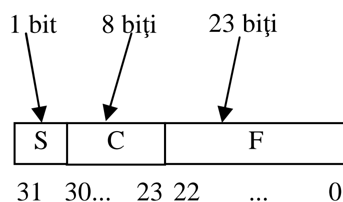
Formatul de reprezentare în virgulă mobilă este standardizat de către standardul IEEE 754, apărut în 1980 şi completat ulterior. Până la apariţia sa, codurile numerice în virgulă mobilă variau de la o familie de calculatoare la alta, făcând dificilă portabilitatea programelor între diferite maşini, căci apăreau dificultăţi în domenii ca: erori de aproximare, tratarea depăşirilor superioare şi inferioare şi tratarea altor condiţii de excepţie. Standardul se referă la reprezentările pe 32 biţi şi pe 64 biţi. În oricare dintre formate standardul impune folosirea *exponentului deplasat*, a *formei normalizate* pentru mantisă, a *bitului ascuns*, iar baza B este considerată a fi egală cu 2. Bitul ascuns nu se reprezintă în formatul binar în virgulă mobilă, dar de valoarea sa se ţine cont atunci când se lucrează cu numerele (unităţile aritmetice şi logice ţin cont de bitul ascuns). Modul standardizat de reprezentare poate fi rezumat astfel:

- *pentru 32 de biţi:* 1 bit de semn, 8 biţi pentru exponentul deplasat (+127) şi 23 de biţi pentru mantisă). Implicit se ştie ca primul bit (înainte de virgulă) al mantisei nu este reprezentat şi el are valoarea 1.
- *pentru 64 de biţi:* 1 bit de semn, 11 biţi pentru exponentul deplasat (+1023) şi 52 de biţi pentru mantisă). Implicit se ştie ca primul bit 1 al mantisei nu este reprezentat şi el are valoarea 1.

Un bun exemplu de utilizare a standardului IEEE pentru reprezentarea în virgulă mobilă este circuitul coprocesor Intel 8087 al familiei I8086. El foloseşte un format cu maximum 80 de

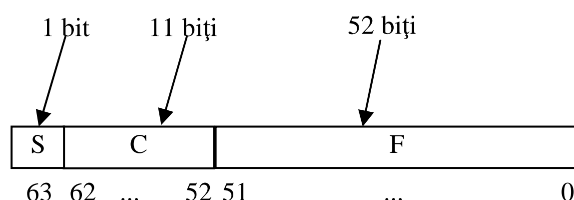
biţi, având registre interne cu această dimensiune. Numerele pot fi reprezentate în formatele 32, 64, sau 80 de biţi. În figura 4. 9. s-au notat cu S semnul, cu C exponentul deplasat şi cu F fracţia, prin care înţelegem partea de după virgulă a mantisei, deci fără bitul ascuns:

Formatul real temporar se foloseşte la familia de coprocesoare Intel x87, dar şi pentru emulările software, pentru reprezentarea temporară internă. La acest format temporar nu se mai impune reprezentarea cu bit ascuns.



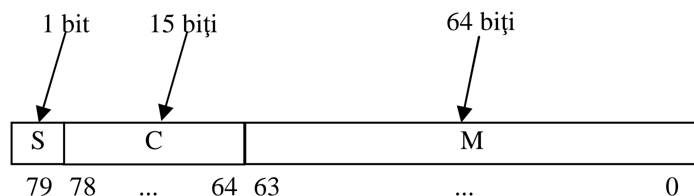
Real scurt:

- Reprezentare pe 32 biţi
- Precizie mantisă: 24 biţi cu HB
- Caracteristică C = E + 127



Real lung:

- Reprezentare pe 64 biţi
- Precizie mantisă: 53 biţi cu HB
- Caracteristică C = E + 1023



Real temporar:

- Reprezentare pe 64 biţi
- Precizie mantisă: 64 biţi fără HB
- Caracteristică C = E + 16383

Figura 4.9. Formatele numerelor reale reprezentate în virgulă mobilă pentru coprocesorul Intel 8087, conform standardului IEEE 754.

În standardul IEEE de reprezentare se indică şi modul de *interpretare al rezultatelor*, prin definirea unor situaţii de excepţie. Excepţiile din standardul IEEE (caracteristică egală cu zero sau mai mare de 254 pentru reprezentarea pe 32 de biţi, respectiv 2046 pe 64 de biţi) sunt definite în aşa fel ca să seteze indicatori de condiţii ai procesorului gazdă şi să genereze ca urmare evenimente ce trebuie tratate ca să nu existe erori de reprezentare sau de precizie. Dacă pentru un număr normalizat, mantisa este zero, iar exponentul are cea mai mică valoare posibilă, este reprezentată valoarea zero. Dacă însă exponentul deplasat are cea mai mică valoare posibilă (C=0), iar mantisa are o valoare diferită de zero, mantisa nu este normalizată (este ne-normalizată), iar biţii stocaţi ai mantisei dau mărimea acesteia. Acest număr este prea mic pentru a fi reprezentat. Termenul „ne-normalizat" (denormalized în limba engleză) se referă la numerele

care pot avea orice valoare pentru C, dar mantisa nu poate fi adusă la forma normalizată, cu bit ascuns, pentru că se depăşesc limitele de reprezentare pentru caracteristică (ar trebui să rezulte o caracteristică negativă). Cele mai multe din procesoarele ce lucrează în virgulă mobilă fac, în acest caz, aproximarea la zero a numărului, eroarea fiind proporţională cu mărimea mantisei ne-normalizate.

În figura 4.10 se prezintă modul de tratare al excepţiilor în standardul IEEE 754 pentru numere reale reprezentate pe 32 de biţi. Simbolul NAN (*Not-A-Number*, rezultatul nu este un număr) este folosit pentru toate operaţiile care conduc la erori, de exemplu împărţire la zero, sau extragere de radical dintr-un număr negativ, pe baza interpretării simultane a fracţiei diferite de zero F ≠ 0 (unde Mantisa M = 1.F, bitul 1 din faţa virgulei fiind bitul ascuns) şi a exponentului deplasat (caracteristică) cu valoare C = 255 (la formatul pe 32 de biţi) respectiv C = 2047 (la formatul pe 64 de biţi).

Atunci când se produce o depăşire a capacităţii de reprezentare (numărul este prea mare pentru a putea fi reprezentat pe cei n biţi) rezultatul este definit ca fiind + sau – infinit (± ∞) şi identificarea se face de asemenea pe baza valorilor fracţiei şi exponentului deplasat (C = 255 respectiv C = 2047 şi F = 0).

Caracteristica	**Fracţie**	**Interpretare rezultat**
C = 255	F ≠ 0, cu HB, (mantisă normalizată)	simbol de eroare NAN (nu este număr)
C = 255	F = 0, cu HB, (mantisă normalizată)	depăşire $N = (-1)^S \infty = \pm\infty$
0 < C < 255	Mantisă normalizată, (M = 1.F)	Număr real, $N = (-1)^S \times 2^{C-127} \times (1.F)$
C = 0	F = 0, cu HB, (mantisă normalizată)	Număr = 0
C = 0	F ≠ 0, (mantisă ne-normalizată)	Număr prea mic pentru a fi reprezentat

Figura 4.10. Modul de interpretare al excepţiilor în standardul IEEE 754 pentru numere reale reprezentate pe 32 de biţi.

Alt aspect definit de standardul IEEE, cu implicaţii mari în ceea ce priveşte proiectarea hardware, sunt regulile de rotunjire. Standardul defineşte patru moduri de rotunjire:

- rotunjire la valoarea cea mai apropiată (*round to nearest*)
- rotunjire către valoarea mai apropiată de zero (*round toward 0*)
- rotunjire către +∞ (*round toward +∞*)
- rotunjire către -∞ (*round toward -∞*)

Modul implicit de rotunjire este rotunjire la valoarea cea mai apropiată. În cazul în care numărul ce trebuie aproximat se găseşte exact la jumătatea intervalului dintre două numere succesive reprezentabile, rotunjirea se face la un număr par (*round to nearest even*).

Ca o concluzie, conform standardului IEEE 754 un număr real reprezentat pe 32 de biţi, la care se respectă condiţia 0<C<255, se poate calcula cu relaţia, :

$$N = (-1)^S 2^{C-127}(1.F) \quad (4.24)$$

Pentru formatul real lung, pe 64 de biţi, dacă se respectă condiţia 0<C<2047, numărul poate fi calculat cu relaţia:

$$N = (-1)^S 2^{C-1023}(1.F) \quad (4.25)$$

Exemple numerice de reprezentare pe 32 de biţi conform IEEE 754:

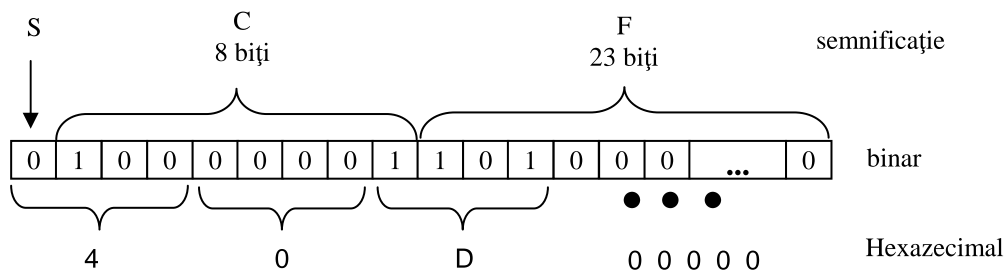
a) N_{10} = +6.5

în binar: M = 110.1 * 2^0 = 1.101 * 2^2

C = 127_{10} + 2_{10} = 129_{10} = 1000 0001_2

S = 0 (număr pozitiv)

Reprezentarea pe 32 de biţi va fi:



În cazul vizualizării cu ajutorul unui program depanator a conţinutului registrului ce stochează numărul real reprezentat pe 32 de biţi informaţia va fi afişată în hexazecimal ca: 40D00000 hex.

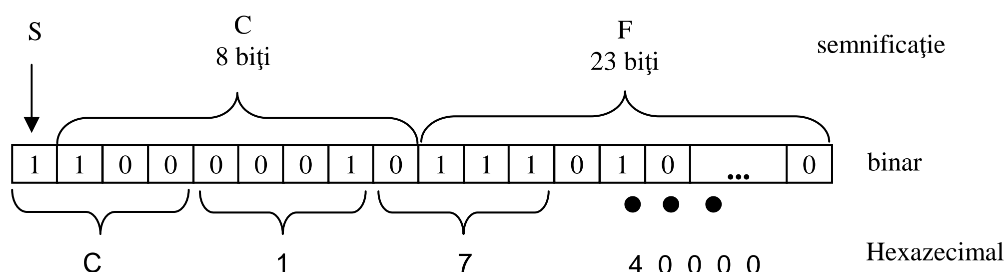
b) N_{10} = -15.375

în binar: M = 1111.011 * 2^0 = 1.111011 * 2^3

C = 127_{10} + 3_{10} = 130_{10} = 1000 0010_2

S = 1 (număr negativ)

Reprezentarea pe 32 de biţi va fi:



În cazul vizualizării cu ajutorul unui program depanator a conţinutului registrului ce stochează numărul real reprezentat pe 32 de biţi informaţia va fi afişată în hexazecimal ca: C1740000 hex.

4.4. Coduri alfanumerice

Sunt coduri care stabilesc o corespondenţa între mulţimea formată din litere, cifre şi semne speciale, denumite în general caractere şi mulţimea binară B = {0,1}.

Cel mai utilizat cod alfanumeric folosit în prezent, este codul ASCII (American Standard Code for Information Interchange). Iniţial a fost un cod de 7 biţi, puţindu-se reprezenta 2^7 caractere. În prezent se foloseşte codul ASCII extins, cu 8 biţi, apelat adesea doar prin numele de cod “ASCII”. Primele 32 de caractere ASCII codifică coduri de control şi sunt utilizate pentru a transmite diferite caractere de control privind mesaje. Aceasta pentru că ASCII a fost dezvoltat iniţial pentru transmisia datelor (de exemplu caracterul de control ^D, EOT = End Of Transmission, este utilizat pentru a indica sfârşitul unui flux de date). Restul codurilor se referă la 64 litere şi cifre şi 196 caractere de control, semne de punctuaţie, caractere grafice şi alte caractere. Scopul principal al codului ASCII este reprezentarea textului. Astfel că adesea datele de tip *text* sunt referite ca date de tip *ASCII*. O secvenţa de caractere reprezentate prin codurile lor ASCII, e numită *şir ("string")*. Şirul poate sa fie gol (nici un caracter) sau poate cuprinde o frază, un paragraf, sau chiar un bloc întreg de caractere. De aceea trebuie ştiută lungimea şirului. Pentru aceasta, de exemplu în limbajul BASIC se foloseşte un descriptor de şir, care include lungimea şi adresa şirului. La sistemul de operare DOS şi la unele limbaje de programare, cum este C, şirul de caractere se termină cu un cod ASCII 0 (zero); acesta se numeşte şir ASCIIZ.

În figura 4.11. se prezintă tabelul codurilor ASCII cu 7 biţi.

CODUL ASCII

	$b_6b_5b_4$							
$b_3b_2b_1b_0$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	/	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	¬
1111	SI	US	/	?	O	—	o	DEL

Figura 4.11. Tabelul codurilor ASCII pe 7 biţi

4.5. Adunare şi scădere pentru numere reprezentate în virgulă fixă

La proiectarea unui procesor, se urmăreşte, pe lângă creşterea vitezei de funcţionare şi minimizarea numărului de circuite. În această direcţie, acelaşi circuit trebuie să poată realiza cât mai multe din funcţiile apelate prin setul de instrucţiuni. Dacă ne referim la unitatea aritmetică, este dezavantajos să se folosească circuite separate pentru scădere şi pentru adunare. ***În calculator numerele întregi negative se codifică în binar în cod complementar*.** Motivul pentru care se foloseşte reprezentarea în complement faţă de doi pentru reprezentarea numerelor negative, este că scăderea unui număr din alt număr este echivalentă matematic cu adunarea complementului de doi a scăzătorului la descăzut. Implementarea complementului de doi printr-un circuit electronic este foarte uşor de realizat. De asemenea, un întreg în reprezentarea complement faţă de doi poate fi uşor extins la un format mai mare fără schimbarea valorii sale. Pentru aceasta este necesar doar ca bitul de semn să fie re-scris în toate poziţiile binare de ordin superior apărute (operaţie numită *extensie de semn*).

De exemplu pentru întreg pe 16 biţi ce trebuie extins la 32 biţi: (s-au notat: cu S semnul, iar cu n poziţiile binare ce indica mărimea numărului):

16 biţi:					**Snnn**	**nnnn**	**nnnn**
Extensie la 32 biţi:	**SSSS**	**SSSS**	**SSSS**	**SSSS**	**Snnn**	**nnnn**	**nnnn**

Ca urmare a folosirii codului complementar, în calculator se poate folosi un singur circuit sumator, care face atât adunările cât şi scăderile; scăderea se efectuează prin adunarea complementului scăzătorului la descăzut şi neglijarea eventualului transport de la bitul cel mai semnificativ al rezultatului:

$$A - B = A+(-B) \qquad (4.26)$$

Exemplu de sumare algebrică în cod complementar, pentru n = 4 biţi:

```
 7-    0111 -     echivalent cu     0111 +
 2     0010                         1110      (-2 în cod C2)
---    ------                     --------
 5     0101                       1̸ 0101
```

Este interesant de observat că aceasta posibilitate de efectuare a scăderilor nu este specifică bazei 2, ea fiind valabilă, de exemplu, şi pentru baza 10. Fie operaţia de scădere în zecimal:

$$897-271=626$$

care se poate scrie după calcularea complementului faţă de 10 a scăzătorului ca:

```
897 - (1000-271) =    897 +
                      729
                    -------
                     1̸ 626
```

Se poate spune că suma algebrică a doi operanzi reprezentaţi în cod complementar, va produce un rezultat care este reprezentarea în cod complementar a sumei sau diferenţei făcute cu numerele naturale, cu condiţia ca să nu existe depăşiri ale capacităţii de reprezentare (depăşirea indică că rezultatul obţinut nu poate fi reprezentat în cod complementar cu n biţi).

Depăşirea capacităţii apare la sumarea algebrica a numerelor cu acelaşi semn. Efectul depăşirii este alterarea bitului de semn, care astfel este diferit de semnul operanzilor.

Exemple de depăşire a capacităţii de reprezentare în binar, pentru n=8 biţi (domeniul maxim între -128 şi +127):

```
+127 +      0111 1111 +
+127        0111 1111
 ------     -------------
  254       1111 1110    (care este reprezentarea lui -2 în C2)
```

```
-127 +        1000 0001 +
-   2         1111 1110
 -----       -------------
-129        1 0111 1111
```

(care, dacă neglijăm transportul, este reprezentarea lui +127 în C2)

De notat că:

- adunarea a două numere care au semne diferite nu va genera niciodată depăşire a capacităţii de reprezentare
- dacă operanzii au acelaşi semn, dar rezultatul are semn diferit s-a generat o depăşire a capacităţii de reprezentare.

Aşa cum s-a demonstrat în paragraful 4.4 implementarea hardware a circuitului care detectează depăşirea capacităţii de reprezentare pe n biţi se poate face printr-o poartă SAU-EXCLUSIV care se poziţionează pe 1 dacă există depăşire:

$$V = T_n \oplus T_{n-1} \quad (4.27)$$

Un exemplu simplificat de circuit care poate efectua sumarea algebrică a numerelor întregi reprezentate în C2 se prezintă în figura 4.12.

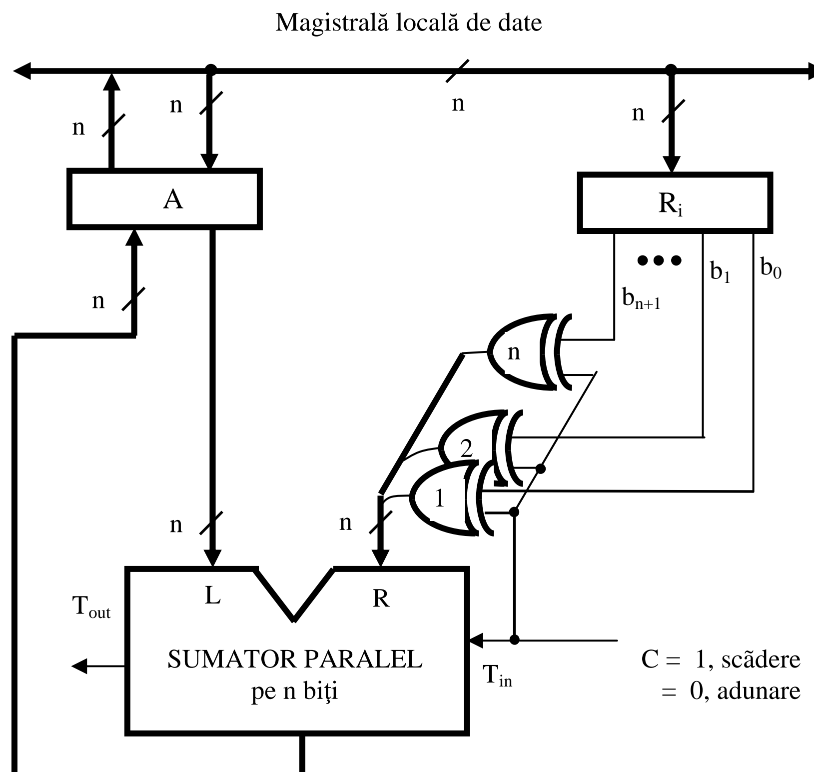


Figura 4.12. Structura de principiu a unităţii aritmetice pentru numere reprezentate în virgulă fixă;

Elementele principale reprezentate în figura 4.12 au următoarele semnificaţii:

- A = registru acumulator. El conţine unul dintre operanzii sursă (pentru intrarea L - Left / stânga a sumatorului din figură) şi tot el este destinaţia în care se "acumulează" rezultatul de la sumatorul de n biţi;
- R_i este un registru de utilizare generală. În acest registru se stochează datele de intrare care vor fi transmise intrării R (Right = dreapta) a circuitului sumator;

Registrele A şi R_i au câte n celule, iar sumatorul paralel poate aduna cuvinte reprezentate pe n biţi. Operaţia de adunare sau scădere pentru numerele reprezentate în C2 este comandată de către semnalul de control C, care ia valoarea 1 în cazul când se comandă scădere şi zero pentru adunare. Dacă acest semnal este 1 logic el va produce complementarea numărului binar din R_i, prin intermediul celor n porţi "sau-exclusiv", activate toate simultan pe una dintre intrări de către C=1. Dacă la codul invers se adună 1 la poziţia cea mai puţin semnificativă (aici prin C legat la intrarea de transport de intrare, T_{in}) se obţine codul complementar al numărului din R_i. În structura din figura 4.20 s-a presupus existenţa unui registru acumulator, similar acumulatoarelor din microprocesoarele de 8 biţi. Cazul se poate generaliza, dacă se consideră că ieşirea sumatorului se depune pe magistrala locală de date, iar de aici rezultatul poate fi transferat în oricare registru de uz general.

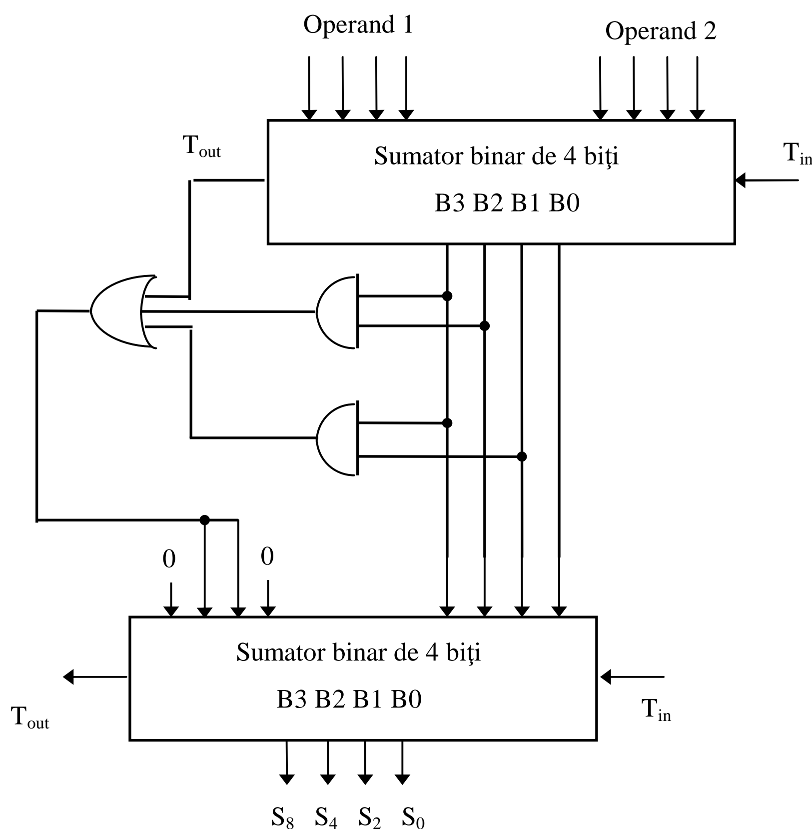


Figura 4.13. Circuit simplificat pentru adunarea şi corecţia zecimală a numerelor zecimale codificate binar în cod NBCD

În virgulă fixă procesoarele pot lucra şi în cod zecimal-binar natural (NBCD). La sumarea numerelor în NBCD este necesară *corecţia zecimală*. Corecţia se efectuează când la sumarea codurilor a două cifre zecimale se produce transport, sau dacă combinaţia binară de 4 biţi la ieşirea sumatorului este mai mare decât codul cifrei 9.

Pentru exemplificare vom considera un circuit care ar putea implementa această corecţie la sumarea a doua cifre în NBCD. Circuitul din figura 4.13. [Mano93] ţine cont de faptul că toate codurile mai mari de 1001 au un bit 1 pe poziţia de pondere 3 (ca şi codurile pentru 8 şi 9), dar în plus au un bit 1 şi pentru ponderea 2, sau ponderea 1. Doar pentru variantele acestea şi pentru generare de transport la sumare va trebui efectuată corecţia zecimală la adunare (adunarea lui 6). Acelaşi lucru se poate spune şi despre scăderea numerelor zecimale scrise în NBCD. Corecţia se face însă prin scăderea codului lui 6.

Operanzii codificaţi binar se interpretează în forma corectă, ca număr binar, sau ca număr zecimal codificat în NBCD, de către programul care lucrează cu aceşti operanzi. Operaţiile de adunare (sau scădere) se indică prin aceleaşi instrucţiuni către procesor, indiferent că operanzii sunt în binar, sau în cod NBCD. De aceea corecţia nu se face automat, aşa ca în circuitul exemplificat în figura 4.13 ci ea este comandată de unitatea de control cu ajutorul unor instrucţiuni maşină specializate ce generează operaţiile de conversie zecimală. Acest lucru permite utilizarea unui singur sumator, atât pentru sumarea binară cât şi pentru corecţia zecimală.

4.6. Suma algebrică a mai multor operanzi reprezentaţi în cod complementar

O proprietate importantă a sumei algebrice a mai multor operanzi în complement faţă de 2 este faptul că depăşirile care apar eventual la executarea sumelor parţiale se pot neglija, dacă valoarea finală a sumei se încadrează în domeniul de reprezentare corespunzător. Considerând de exemplu numere cu 4 biţi în C2, cuprinse în gama (-8) ÷ (+7) se poate scrie [Sztojanov87]:

$$5 + (-3) + 4 = 6$$

în care S1=5+(-3) = 2, iar S2 = S1+4 = 2+4 = 6. Dar conform comutativităţii se poate calcula şi o sumă S2 astfel:

$$S1 = 5+4 = 9 \text{ (depăşire)}$$

$$S2 = 9 + (-3) = 6$$

Reprezentând în C2 operanzii, şi notând depăşirea către numere pozitive cu V_+ şi către numere negative cu V_- cu vom avea:

$$S_1 = 5+4 = 9 > 7 \Rightarrow \mathbf{V_-}\,; \qquad 9 = 2^4-7 = (-7)_2$$

$$S_2 = 9+(-3) = 9 + (2^4-3) = 2^4+6 \Rightarrow \mathbf{V_+}$$

$$S_2 = 6$$

La prima sumare apare o depăşire V_- către numere negative, iar rezultatul 9 reprezintă (-7) în C2.

A doua operaţie generează de asemenea o depăşire dar în sens contrar, către numere pozitive, V_+, justificat prin interpretarea (-7) + (-3) = -10 < -8.

Din exemplul numeric anterior se observă că printr-o depăşire în sens contrar rezultatului s-a revenit în domeniul corect. Pentru un număr de operaţii succesive mai mare decât doi, dacă numărul depăşirilor V_+ este egal cu cel al depăşirilor V_- (indiferent de succesiunea lor), atunci ele se "compensează" iar rezultatul este corect. În practică, un numărător reversibil poate fi incrementat, respectiv decrementat, la apariţia celor două tipuri de depăşiri. Dacă valoarea finală a numărătorului este egală cu cea iniţială atunci rezultatul este corect.

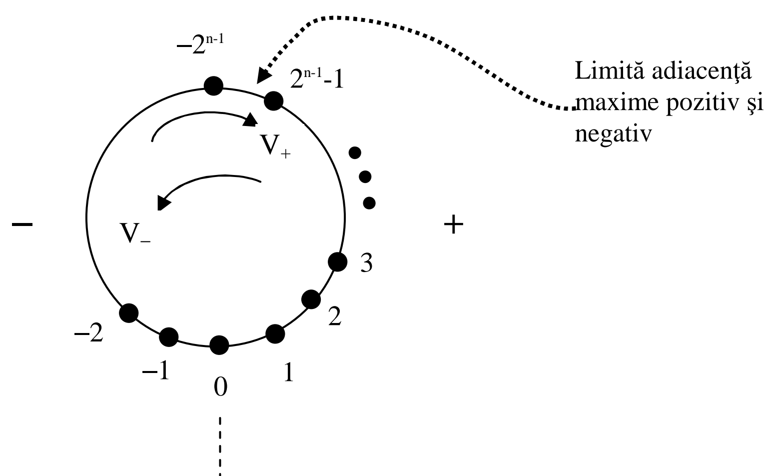
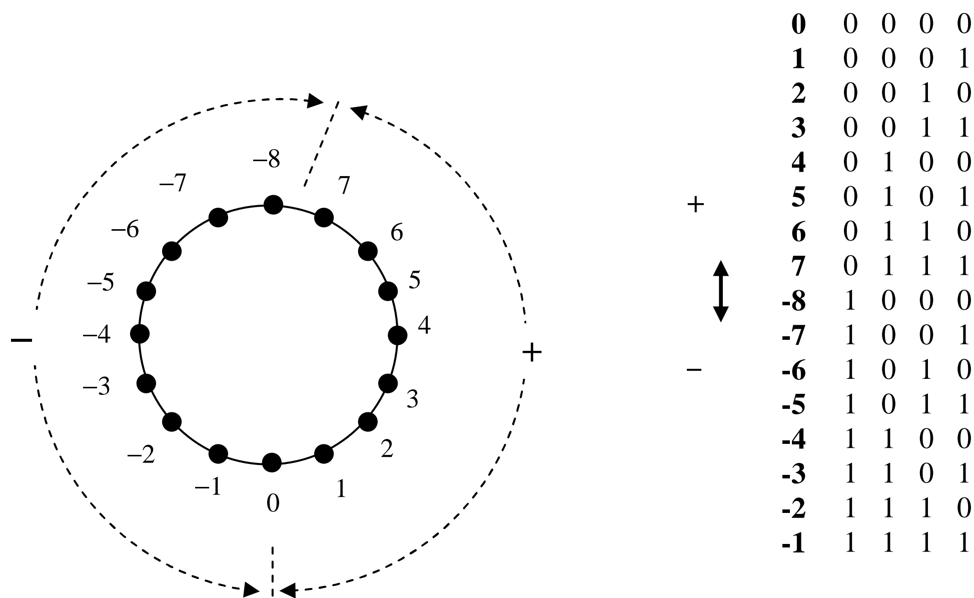


Fig. 4.14. Efectul depăşirilor aritmetice la operaţii în complement faţă de doi cu întregi reprezentaţi pe n biţi.

Justificarea proprietăţii sumei algebrice repetate pentru operanzi în cod complementar este legată de codul complement faţă de 2. La reprezentarea întregilor pe n biţi numărul negativ minim (-2^{n-1}) reprezentat prin 2^{n-1} în cod complementar (pentru că $2^n-2^{n-1} = 2^{n-1}(2-1) = 2^{n-1}$) este adiacent cu numărul maxim pozitiv reprezentat ca $2^{n-1}-1$. Dacă numerele sunt reprezentate pe un cerc ca în figura 4.14 depăşirile spre V_- sau V_+ reprezintă traversarea *limitei de demarcaţie* între maximul pozitiv şi maximul negativ.

Rezultatul fiecărei operaţii parţiale se poate deplasa pe un cerc cu condiţia ca în final să revină în domeniul iniţial. Proprietatea menţionată este utilă în unităţile aritmetice prevăzute cu sumare şi acumulare a rezultatelor. Pentru verificarea celor spuse mai sus, în figura 4.15 se prezintă tabelul codurilor complementare pentru întregi pe 4 biţi şi poziţionarea pe un cerc a acestor reprezentări pentru verificarea adiacenţei.

În cazurile în care operanzii depăşesc ca lungime capacitatea unităţii aritmetice de n biţi, atunci ei se pot exprima prin mai multe cuvinte cu câte n biţi juxtapuse iar operaţiile de adunare şi scădere se vor executa în mai mulţi paşi efectuând într-un pas o sumă algebrică pe primii n biţi, în continuare pe următorii n biţi (ţinându-se seama şi de transportul de la suma anterioară) ş.a.m.d.



0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

Figura 4.15. Exemplificare pentru depăşiri aritmetice la operaţii în complement faţă de doi, pentru n=4.

4.7. Adunarea numerelor reprezentate în virgulă mobilă

La adunarea numerelor în virgulă mobilă, exponenţii se tratează separat faţă de mantise. Secvenţa operaţiilor necesare pentru adunarea a doi operanzi în virgulă mobilă în C2 este:

- se compară exponenţii deplasaţi (caracteristicile)
- se deplasează (shift la dreapta) mantisa termenului cu exponent mai mic până când cei doi exponenţi sunt egali
- se sumează mantisele
- se normalizează rezultatul, se testează pentru depăşire şi se face rotunjire.

Operaţiile de adunare în virgulă mobilă se pot efectua fie cu ajutorul unităţii aritmetice pentru virgulă fixă (ALU) şi cu programe de emulare corespunzătoare reprezentării în virgulă mobilă, fie cu circuite (hardware) dedicate acestor operaţii. Prima variantă este economică, dar este lentă. A doua, implementează cablat funcţia de adunare. Aritmetica în virgulă mobilă (pentru toate cele 4 operaţii aritmetice elementare) presupune operaţii cu exponenţi (numere întregi) şi operaţii aritmetice, inclusiv deplasări (shift) aritmetice, cu mantise subunitare. Rezultă că cea mai mare parte a timpului de executare a operaţiilor în virgulă mobilă se pierde pentru operaţii cu numere în virgulă fixă:

- deplasări de mantise,

- adunări de mantise,
- înmulţire / împărţire de mantise,
- adunări / scăderi de exponenţi.

Pentru a mari viteza de lucru, pentru operaţii repetate în virgulă mobilă, structura sumatorului în virgulă mobilă este de obicei de tip conductă (pipeline), ca în figura 4.16 (prelucrată după [Hayes88]). Marea problemă la operaţia de adunare în virgulă mobilă o reprezintă aproximările necesare a fi introduse. Eroarea rezultată este numită eroare de aproximare (*roundoff error)* şi ea rezultă din faptul că fiecare număr se reprezintă printr-un număr fix, limitat de biţi. Frecvent, o operaţie cu numere pe n biţi, de exemplu o înmulţire, poate conţine până la 2n biţi, dintre care până la n din ei trebuie eliminaţi. Reţinerea celor mai semnificativi n biţi a rezultatului fără nici o altă modificare se numeşte trunchiere (*truncation*). Deci numărul rezultat are o eroare de aproximare dată de mărimea părţii eliminate. Această eroare poate fi redusă prin procesul numit rotunjire (*rounding*). Cea mai simplă cale este rotunjirea unui număr fracţionar către cel mai apropiat număr reprezentabil pe n biţi. Această rotunjire constă în adunarea unui bit 1 la bitul rezultatului cu indicele -(n+1) (ponderea $2^{-(n+1)}$) adunând şi eventualul transport generat către rangul 2^{-n} şi apoi să se facă trunchiere la n biţi.

Dacă ne referim la structura sumatorului pipeline pentru numere în virgulă mobilă din figura 4.16 în partea stângă se poate distinge unitatea de prelucrare a exponenţilor, iar în partea dreaptă unitatea de prelucrare a mantiselor. Pentru registrele de stocare ale exponenţilor şi mantiselor pentru operanzi s-au folosit notaţiile E1, E2, respectiv M1, M2. Registrele pentru mantisa şi exponentul rezultatului sunt notate cu M şi E. Blocurile notate cu D şi indice (D1 şi D2) sunt circuite pentru deplasarea stânga / dreapta a informaţiei conţinute; iar valoarea de control aplicată în partea stângă a acestor registre de deplasare indică numărul de poziţii binare pe care se comandă deplasarea. Circuitele sumatoare binare paralele sunt notate S1, S2, S3. Nu s-au figurat separat circuitele multiplexoare de la intrările registrului de deplasare D1 şi de la intrările sumatoarelor S2 şi S3, dar s-a indicat necesitatea lor prin semnul grafic . Circuitul notat “Detector” calculează numărului de biţi 0 după virgulă, pentru a putea comanda operaţia de normalizare a mantisei şi actualizare a exponentului, având şi sarcina de detectare şi corectare a eventualelor depăşiri în urma sumării în sumatorul S2.

Mantisele celor două numere ce se adună se stochează în registrele M1 şi M2, iar exponenţii în registrele E1 şi E2. Mai întâi mantisele trebuie aliniate (pentru a avea acelaşi exponent). Se deplasează întotdeauna mantisa care are exponentul mai mic, pentru că astfel deplasarea în D1 trebuie făcută la dreapta, deci se pierd cei mai puţin semnificativi biţi. Operaţia de aliniere se face în funcţie de rezultatul şi semnul diferenţei d = E1-E2 calculată cu sumatorul S1.

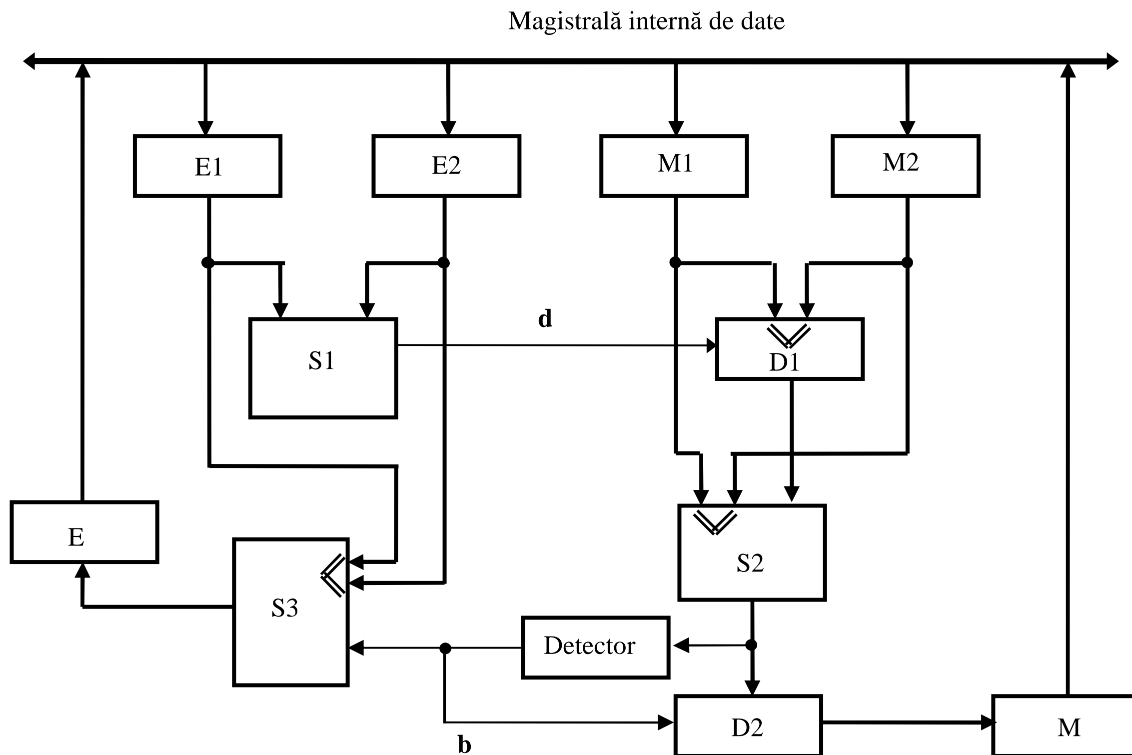


Figura 4.16. Schemă bloc a unui sumator în virgulă mobilă cu structură de tip pipeline.

După operaţia de aliniere a celor doi operanzi (exponenţi egali), mantisele se pot aduna cu sumatorul S2, care conţine şi un multiplexor, astfel încât adună fie $M1 + M2\times2^{-d}$, fie $M1\times2^{-d} + M2$. Dacă în urma sumării se produce depăşire a capacităţii de reprezentare (oVerflow) pe n biţi, aceasta se corectează prin deplasarea numărului (în D2) către dreapta o poziţie binară (si incrementarea corespunzătoare a exponentului). Detectarea depăşirii este luată în considerare de circuitul *Detector*, care în cazul când nu apare depăşire, detectează numărul de biţi 0 de după virgulă, pentru ca mantisa rezultatului să se poată normaliza în D2. Dacă mantisa se deplasează dreapta *b* poziţii binare, la exponentul E (egal cu E1 daca E1>E2, sau cu E2 dacă relaţia este inversă) se va aduna *b* în sumatorul S3. Ultima operaţie care se face (si pentru care nu apare un bloc distinct în figură) constă în testarea rezultatului la depăşiri superioare (overflow, $E > 2^e -1$, unde e este numărul de biţi pentru reprezentarea exponentului deplasat), respectiv la depăşiri inferioare (underflow, exponentul deplasat E = 0).

4.8. Unitatea de prelucrare a datelor pentru virgulă fixă

Structura circuitelor aritmetice din cadrul unităţii de prelucrare a datelor pentru un anumit procesor, depinde în mod direct de operaţiile aritmetice şi logice pe care le poate efectua procesorul. Unitatea de prelucrare este formată din circuite aritmetice şi logice, registre folosite pentru stocare temporară sau prelucrare (deplasare, rotire) a informaţiei binare, circuite de decodificare şi magistrale de transmitere a datelor între blocurile componente. Elementul central al unităţii de prelucrare este Unitatea Aritmetică şi Logică (ALU) care cuprinde un sumator paralel de n biţi şi circuite logice aferente pentru implementarea funcţiilor logice (AND, OR, NOT, XOR etc.) pentru operanzii de n biţi de la intrarea ALU. De asemenea, ALU poate cuprinde circuite logice de decodificare a funcţiilor de executat, generare de biţi indicatori asupra tipului rezultatului, generator de transport anticipat, circuite de comparare etc. Complexitatea unităţii de prelucrare a datelor este determinată de maniera în care se realizează operaţiile sale aritmetice. Unităţile simple pentru operaţii în virgulă fixă pot fi construite în jurul ALU. Dacă se implementează hardware pentru aritmetica în virgulă mobilă complexitatea unităţii de prelucrare creşte. Unele familii de microprocesoare care au ALU în virgulă fixă folosesc unităţi auxiliare (externe UCP) speciale numite *(co-)procesoare aritmetice* pentru a realiza funcţii în virgulă mobilă şi alte funcţii numerice complexe. Alte microprocesoare au integrat pe acelaşi chip de siliciu mai multe unităţi în virgulă fixă şi mobilă.

În figura 4.17 se prezintă o structură elementară de unitate de prelucrare a datelor în virgulă fixă.

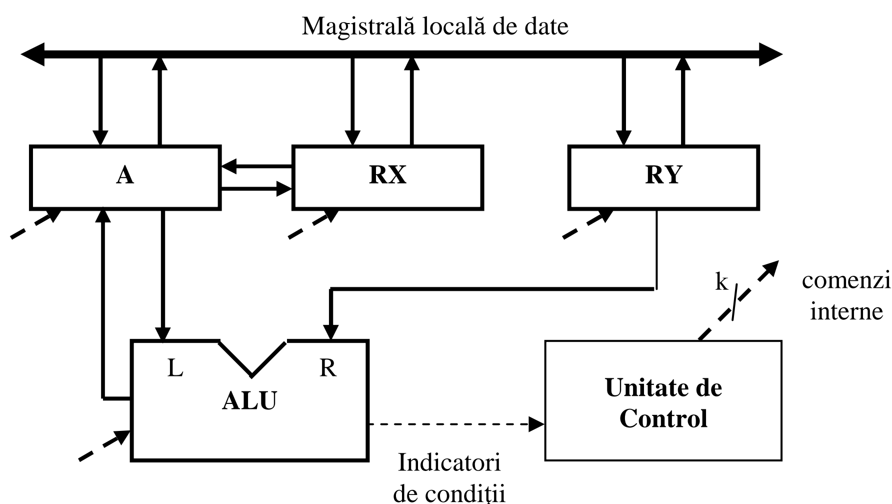


Figura 4.17. Structură elementară de unitate de prelucrare a datelor în virgulă fixă.

Această unitate permite efectuarea de operaţii logice şi operaţii aritmetice elementare. Mai adăugăm că toate registrele din figură se consideră de n biţi, şi că s-au folosit următoarele notaţii:

- A - registru acumulator, acesta conţine unul dintre operanzii sursă şi tot el este destinaţia în care se stochează ("acumulează") rezultatul. Poate fi încărcat din exterior, având legătură la o magistrală locală de date, sau datele din acumulator pot fi transmise în exterior;
- RX - registru de uz general, folosit aici ca registru ce dublează capacitatea acumulatorului de la n biţi la 2n biţi. Registrul RX poate fi încărcat din exterior, iar informaţia conţinută poate fi deplasată stânga sau dreapta, sincron cu informaţia conţinută în A. La deplasare cu o poziţie binara spre dreapta LSb din A (vom nota aceasta poziţie binara cu A(0), după ponderea pe care o are) va trece în poziţia MSb din RX (daca RX are 8 biţi poziţia va fi notata R(7)).
- RY - registru de uz general necesar pentru stocarea celui de-al doilea operand.
- ALU - unitate aritmetică şi logică. Efectuează cu operanzii de la intrările L (left) şi R (right) operaţia aritmetică sau logică comandată de unitatea de control. În funcţie de natura rezultatului furnizează către unitatea de control indicatori de condiţii.

Structura din figura 4.17 a fost folosită prima oară în calculatorul IAS (1951) dar şi în alte calculatoare construite ulterior [Hayes88]. Se poate folosi pentru operaţii aritmetice sau logice unul dintre operanzi şi rezultatul stocându-se în registrul acumulator (A).

A	$\leftarrow$ A + RY	;suma algebrica în C2 (adunare sau scădere)
A.RX	$\leftarrow$ RX * RY	;înmulţire
A.RX	$\leftarrow$ A.RX/RY	;împărţire
A	$\leftarrow A \cap RY$	;ŞI logic
A	$\leftarrow A \cup RY$	;SAU logic
A	$\leftarrow A \oplus RY$	;SAU Exclusiv
A	$\leftarrow \overline{A}$	;NOT, negaţie logică

Figura 4.18. Câteva operaţii aritmetice şi logice ce se pot implementa cu unitatea de prelucrare din figura 4.17.

Registrul RX are posibilitatea de deplasare dreapta / stânga împreună cu acumulatorul, ca şi cum ar forma un registru dublu. Această comportare a lui RX uşurează implementarea operaţiilor de înmulţire şi împărţire, în RX stocându-se înmulţitorul la operaţiile de înmulţire, sau

câtul la cele de împărţire. Registrul RY stochează deînmulţitul sau împărţitorul (divizorul), iar rezultatele (produs respectiv cât şi rest) se stochează în registrul dublu format prin concatenarea acumulatorului cu RX, registru dublu notat A.RX.

În figura 4.18 se prezintă câteva operaţii semnificative ce se pot efectua cu unitatea de prelucrare din figura 4.17, împreună cu notaţia simbolică de transfer. Se presupune că în stânga săgeţii ce indică sensul transferului se găseşte destinaţia, iar în dreapta operanzii sursă.

Spre deosebire de structura elementară din figura 4.17 toate microprocesoarele actuale conţin un set de registre cu utilizare generală, oricare dintre ele putând avea rolul registrelor A, RX sau RY. Unitatea de prelucrare tipică a unui microprocesor de uz general este prezentată în figura 4.19. Cele două registre de la intrarea în ALU (notate RT1 şi RT2) sunt registre temporare, inaccesibile prin instrucţiuni. Banca internă de registre este formată în figură din 8 registre a câte 8 biţi. Ele pot fi folosite şi pentru operanzi de lungime dublă, 2d, prin folosirea perechilor de registre cu acelaşi indice numeric. Partea mai semnificativă a registrului dublu este indicată prin litera H (Higher word), iar partea mai puţin semnificativă prin litera L (Lower word). Selecţia registrului implicat într-o operaţie anume se face prin intermediul decodificatorului la scriere şi a multiplexorului la citire.

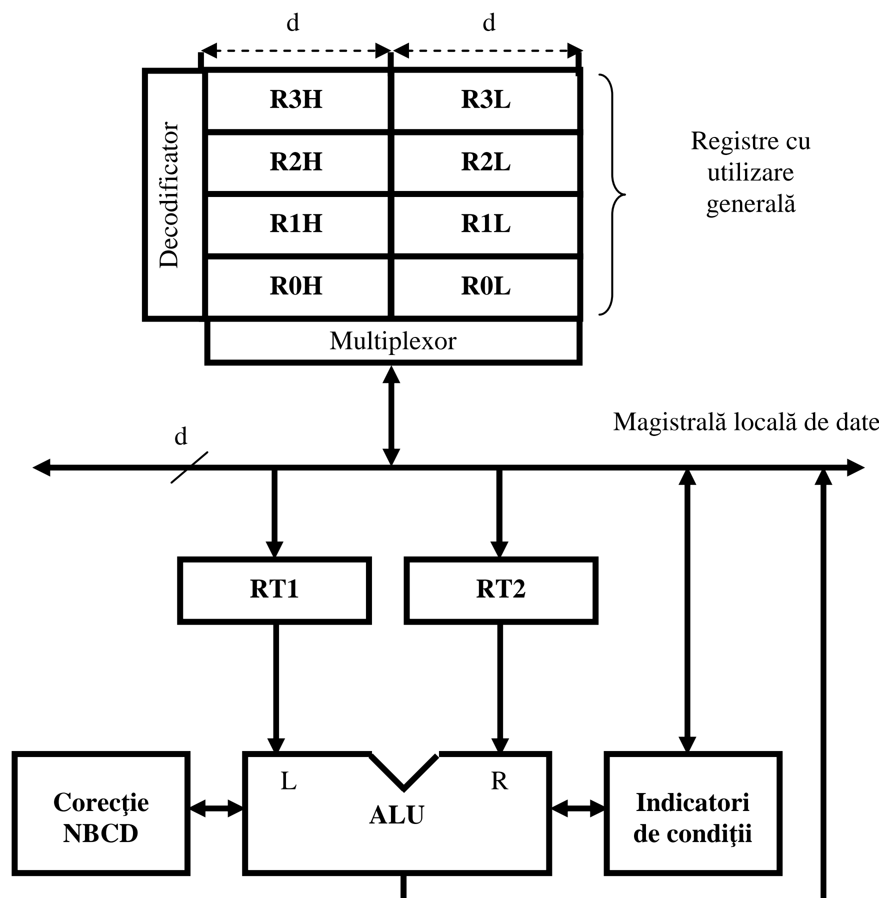


Figura 4.19. Unitate de prelucrare tipică pentru microprocesoare de 8 biţi.

Pentru operaţiile aritmetice se foloseşte aritmetica binară în cod complement faţă de doi, sau în cod NBCD. Pentru operanzii reprezentaţi în NBCD s-a indicat şi circuitul de corecţie zecimală.

Pentru structura unităţii de prelucrare din figura 4.19 o operaţie ce presupune o singură trecere a operanzilor prin ALU se poate executa în trei etape succesive

- transferul datelor din registrele sursă în registrele temporare. Registre sursă pot fi cele opt registre interne, sau pot fi registre (locaţii) din memoria principală de unde informaţia este adusă pe magistrala internă de date.
- efectuarea operaţiei propriu-zise în ALU
- transferul rezultatului în registrul destinaţie. Daca destinaţia este un registru intern ultimele două etape se pot parcurge într-un singur ciclu de lucru.

Dacă se notează cu $*$ un operator corespunzător unei operaţii aritmetice sau logice generice efectuate de ALU, o operaţie cu un singur registru are loc în două etape succesive:

RT1 ← RiL ;conţinutul lui RiL este copiat în registrul temporar RT1
RiL ← *RT1 ;conţinutul lui RT1 este introdus în ALU cu execuţia
;operaţiei simbolizate prin $*$, iar rezultatul de la ieşirea
;ALU se transferă în registrul RiL

O operaţie cu 2 operanzi are forma generală: Ri ← Ri * Rj.

Ca urmare o operaţie de genul R0L ← R0L * R1L (citită ca "R0L operat cu R1L se transferă în R0L") poate fi executată în trei etape succesive:

RT2 ← R0L
RT1 ← R1L
R0L ← (RT1 * RT2)

O operaţie cu operanzi în lungime dubla are forma generală:

(RiH,RiL) ← (RiH,RiL) * (RjH,RjL)

De exemplu operaţia R0 ← R0 + R1 se poate executa în 6 etape:

RT1 ← R0L
RT2 ← R1L
R0L ← RT1 + RT2
RT1 ← R0H

RT2 ← R1H

R0H ← RT1 + RT2 + C ;la adunarea cuvântului mai semnificativ

;se consideră şi transportul (Carry) de la

:rangul inferior.

4.9. Înmulţirea numerelor binare reprezentate în virgulă fixă

Circuitele aritmetice pentru înmulţirea (şi similar pentru împărţirea) numerelor în virgulă fixă se bazează pe algoritmi specifici a căror construcţie urmăreşte realizarea unui compromis acceptabil între costul circuitelor (se urmăreşte reducerea complexităţii circuitelor, deci scăderea costurilor) şi viteza de execuţie a operaţiilor. Un alt scop de urmărit, în limita unei viteze acceptabile, este ca circuitele construite pentru realizarea adunării să poată fi folosite şi de operaţiile de înmulţire. De asemenea, faptul că operaţia de împărţire este inversul celei de înmulţire indică posibilitatea utilizării, cu mici modificări, a aceloraşi circuite atât pentru înmulţire cât şi pentru împărţire. Acest lucru este însă valabil doar dacă operaţiile se efectuează prin metoda secvenţială (asemănătoare modului de înmulţire / împărţire cu creionul şi hârtia) care presupune calcul de-a lungul a n paşi succesivi de produse parţiale, respectiv resturi parţiale. Înmulţirea secvenţială presupune n paşi succesivi de adunare urmată de deplasare în cadrul unui singur registru de stocare a produsului. Complementar, împărţirea secvenţială se reduce la scăderea succesivă a împărţitorului (divizorului) din deîmpărţit, adică în cod complementar împărţirea se poate face prin sumarea algebrică şi deplasarea repetată, pentru calculul succesiv al resturilor parţiale.

Operaţiile de înmulţire şi împărţire pot fi realizate însă cu viteza mult mai mare, cu ajutorul unor reţele de circuite combinaţionale, unele dintre ele având o structură complexă, deci soluţia fiind scumpă. În cazul implementării combinaţionale soluţia cea mai rapidă de implementare se poate obţine prin proiectarea unui circuit combinaţional cu 2n intrări (doi operanzi a câte n biţi) şi cu 2n ieşiri conform funcţiilor scrise şi minimizate după un tabel de adevăr. Produsul rezultă la ieşire cu o întârziere dată de timpul de propagare prin reţea. Pe lângă faptul că un asemenea circuit este mare şi scump el este şi dificil de integrat, pentru că nu prezintă regularitate a reţelei de porţi. De aceea circuitele înmulţitoare şi separat împărţitoare construite ca reţele combinaţionale folosesc reţele de celule identice de bază (sumator complet, sau scăzător complet modificate), interconectate după un anumit algoritm.

Din punctul de vedere al dimensiunii registrelor care memorează operanzii şi produsul, dacă se consideră 2 numere întregi, notate X şi Y, şi reprezentate în binar cu n biţi ca în relaţiile (4.28) produsul lor P = (X ×Y) va fi un număr cu maximum 2n biţi, dacă X şi Y se consideră a fi fără

semn, respectiv cu 2n-1 biţi, dacă X şi Y se consideră a fi cu semn.

$$X = x_{n-1} x_{n-2} ... x_2 x_1 x_0$$
$$Y = y_{n-1} y_{n-2} ... y_2 y_1 y_0 \quad \text{unde } x_i, y_i \in \{0,1\} \quad (4.28)$$

Pentru a se reprezenta produsul cu semn sunt necesari 2n-2 biţi pentru mărime şi un bit de semn. De aceea în unităţile aritmetice care lucrează cu operanzi cu lungimea de n (biţi), pentru produs se utilizează un registru cu lungime dublă (2n). Dacă se doreşte reprezentarea cu 2n biţi a produselor numerelor cu semn, atunci bitul cel mai semnificativ (p_{2n-1}) din cei 2n ai produsului va copia bitul de semn obţinut în p_{2n-2} , iar p_{2n-2} va primi valoarea zero doar pentru reprezentarea în mărime şi semn, pentru că la reprezentările în cod invers şi cod complementar bitul p_{2n-2} are aceeaşi valoare cu bitul de semn p_{2n-1} al produsului P.

Chiar dacă în continuare vom descrie algoritmii de înmulţire atât pentru întregi cât şi pentru fracţionare, pentru aritmetica în virgulă fixă, acelaşi procesor lucrează cu un singur fel de numere: fie cu numere întregi, fie cu subunitare cu semn. Dacă procesorul poate efectua şi calcule aritmetice în virgulă mobilă el va folosi pentru mantise aritmetica cu numere subunitare cu semn. Astfel că la acelaşi procesor, avem pentru virgulă fixă aritmetică cu întregi, iar pentru virgulă mobilă aritmetică cu numere subunitare (pentru mantise) şi întregi (pentru exponenţi). Din această cauză în continuare se vor descrie algoritmi şi exemple pentru ambele moduri de reprezentare în virgula fixa: întregi şi subunitare cu semn.

4.9.1. Metoda înmulţirii directe a numerelor în MS

Dacă se înmulţesc manual două numere binare prin metoda clasică (creion şi hârtie) operaţia se face prin calcularea unor produse parţiale, deplasate fiecare cu câte o poziţie binară către stânga şi apoi prin sumarea acestora, ca în exemplul următor:

```
   1101 ×    deînmulţit
   1001      înmulţitor
   1101
  0000
 0000
1101
1110101      Produs
```

În cazul implementării secvenţiale în calculator a acestei operaţii apar câteva deosebiri faţă de

metoda manuală. Aceste deosebiri, enumerate în continuare au drept scop creştere vitezei de execuţie şi micşorarea spaţiului de memorare a rezultatelor parţiale:

- După fiecare înmulţire cu bitul curent (copiere deînmulţit sau 00...0) se face o adunare pentru a se obţine un produs parţial ce va fi stocat într-un registru. Aceasta pentru că scrierea tuturor rezultatelor înmulţirilor dintre bitul curent şi deînmulţit şi apoi sumarea mai multor cuvinte binare stocate presupune mai multe registre de stocare temporară.
- La fiecare pas, nu se face deplasarea deînmulţitului (sau 00....0) copiat către stânga ci se va deplasa către dreapta conţinutul registrului ce stochează produsul parţial, astfel încât poziţiile binare relative sunt aceleaşi ca la înmulţirea cu creion şi hârtie.
- Atunci când bitul curent al înmulţitorului este zero, nu se mai adună zero ci se face doar deplasarea către dreapta a produsului parţial.

Pentru numere binare întregi codificate în cod direct (mărime şi semn) se poate scrie:

$$X = (-1)^{x_{n-1}} \sum_{i=0}^{n-2} x_i 2^i = (-1)^{x_{n-1}} \cdot X_M$$

$$Y = (-1)^{y_{n-1}} \sum_{i=0}^{n-2} y_i 2^i = (-1)^{y_{n-1}} \cdot Y_M \tag{4.29}$$

unde

$$X_M = \sum_{0}^{n-2} x_i 2^i \quad \text{iar} \quad Y_M = \sum_{0}^{n-2} y_i 2^i \tag{4.30}$$

La aceasta metodă ***semnul se analizează separat***. Astfel semnul se poate calcula prin SAU Exclusiv între semnele celor doi operanzi:

$$p_{2n-2} = x_{n-1} \oplus y_{n-1} \tag{4.31}$$

Produsul se poate scrie:

$$P = (-1)^{p_{2n-2}} \cdot \sum_{i=0}^{n-2} (Y \cdot x_i) \cdot 2^i = (-1)^{p_{2n-2}} \cdot P_M \tag{4.32}$$

iar modulul acestuia:

$$P_M = x_0 2^0 Y_M + x_1 2^1 Y_M + x_2 2^2 Y_M + \ldots x_{n-3} 2^{n-3} Y_M + x_{n-2} 2^{n-2} Y_M \tag{4.33}$$

Dacă folosim notaţia $Y^* = Y_M \cdot 2^{n-1}$, adică modulul lui Y deplasat n-1 poziţii binare la stânga, rezultă dezvoltarea:

$$P_M = 2^{-1}(x_{n-2}Y^* + 2^{-1}(x_{n-3}Y^* + \quad \ldots \quad + 2^{-1}(x_1 Y^* + 2^{-1}(x_0 Y^*))\ldots) \tag{4.34}$$

Termenii $Y^* x_i$ din relaţia (4.34) se numesc produse parţiale. Algoritmul de calcul este sugerat de relaţia de descompunere (4.34). Înmulţirea cu 2^{n-1} reprezintă o deplasare spre stânga a termenului Y cu n-1 poziţii binare, iar înmulţirile cu 2^{-1} reprezintă deplasări spre dreapta cu o

poziţie binară. Algoritmul de implementare al produsului pentru numere cu semn în cod direct poate fi scris:

Input X, Y

P=0, $Y^* = 2^{n-1}Y_M$, $p_{2n-2} = x_{n-1} \oplus y_{n-1}$

Pentru i=0 la n-2

$P = 2^{-1}(P + Y^* \cdot x_i)$

$P = (-1)^{p_{2n-2}} \cdot P$

STOP

Dacă se consideră n = 8 (7 biţi de mărime plus un semn), schema bloc a unui circuit ce poate realiza înmulţirea numerelor întregi cu semn, în cod direct, pentru calculul produsului în n paşi succesivi este prezentată în figura 4.20.

Modulele celor două numere se stochează în registrele RX şi RY. Produsele parţiale se vor acumula în registrul RA (registru acumulator) care ar trebui să aibă, în principiu, dimensiunea dublă faţă de RX şi RY. Dimensiunea dublă se obţine prin deplasarea simultană dreapta a registrelor RA şi RX pentru fiecare produs parţial. Asta pentru că se scanează conţinutul lui RX nu de la dreapta la stânga cu X fix în registru, ci se deplasează conţinutul lui RX pas cu pas către dreapta, astfel încât bitul testat să fie întotdeauna citit din bistabilul cel mai din dreapta al registrului RX.

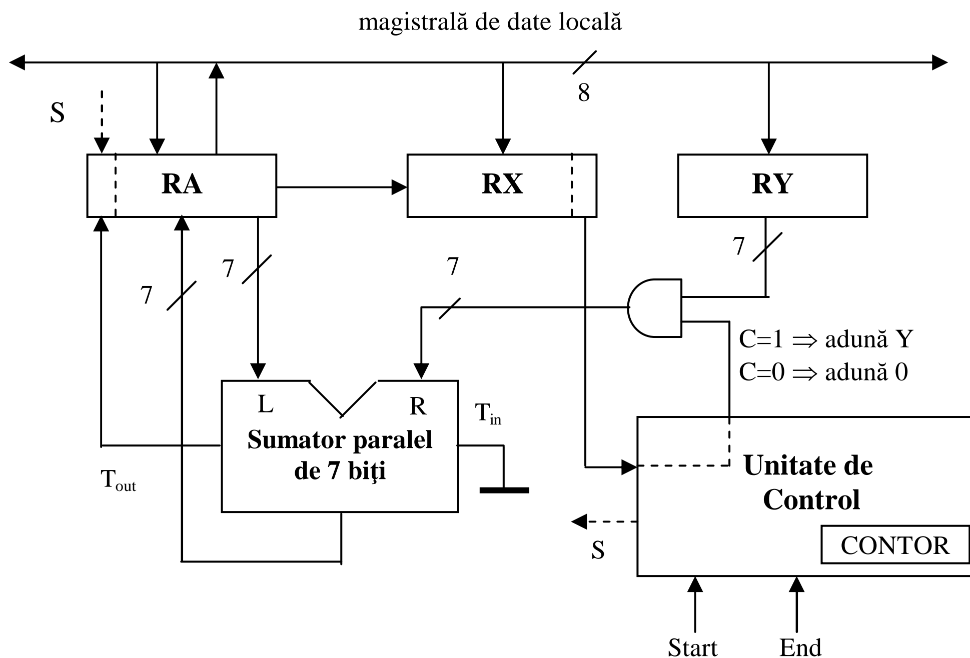


Figura 4.20. Schema bloc a unui circuit de înmulţire a numerelor întregi cu semn în cod direct, în 8 paşi succesivi.

Folosind această metodă de deplasare a lui X către dreapta, pe măsură ce X scade de la 8 biţi (7+1) la 0 produsul P se extinde de la 8 la 16 biţi (tot prin deplasare dreapta). Putem deci folosi cele doua registre RA şi RX ca un registru comun de 16 biţi. Unitatea de control controlează numărul de paşi ai algoritmului şi de aceea s-a simbolizat în interior un contor.

Exemplul 2.
Pentru (-6) × (+6), numerele fiind reprezentate pe n = 4 biţi
(1 semn + 3 mărime)

X_M	=	110		
Y_M	=	110		
P	=	000000		
$Y^*=Y_M\,2^3$	=	110000		
$Y^*\,x_0$	=	000000		$x_0 = 0$
$\times\,2^{-1}$	=	000000	+	(operaţia de deplasare dreapta)
Y^*x_1	=	110000		$x_1 = 1$

		110000		
$\times\,2^{-1}$	=	011000	+	
Y^*x_2	=	110000		$x_2 = 1$

		1001000		
$\times\,2^{-1}$	=	100100		
P_M	=	100100		
P	=	1100100		(primul bit 1 este bitul de semn

Deducerea relaţiei de înmulţire pentru subunitare cu semn în cod direct se face în mod similar cu cea de la întregi. Reamintim că semnul se calculează separat.

$$X = (-1)^{x_0} \cdot \sum_{i=1}^{n-1} x_{-i} \cdot 2^{-i} = (-1)^{x_0} \cdot X_M$$

$$Y = (-1)^{y_0} \cdot \sum_{i=1}^{n-1} y_{-i} \cdot 2^{-i} = (-1)^{y_0} \cdot Y_M \qquad (4.35)$$

Bitul de semn este:

$$p_0 = x_0 \oplus y_0 \qquad (4.36)$$

iar

$$P = (-1)^{p_0} \cdot \sum_{i=1}^{n-1} (Y_M \cdot x_{-i}) \cdot 2^{-i} = (-1)^{p_0} \cdot P_M \qquad (4.37)$$

$$P_M = 2^{-1}(x_{-1}Y_M + 2^{-1}(x_{-2}Y_M + \quad \ldots \quad + 2^{-1}(x_{-(n-2)}Y_M + 2^{-1}(x_{-(n-1)}Y_M))\ldots)$$

(4.38)

Algoritm:

Input X, Y

P=0, $p_0 = x_0 \oplus y_0$

Pentru i=0 la n-2

$P = 2^{-1}(P + Y_M x_{-(n-1)+i})$

$P = (-1)^{p0} \times P$

STOP

Exemplul 3. de calcul secvenţial pentru 11/16 * 13/16 reprezentate pe 5 biţi (1+4)

X_M	=	1011		11/16
Y_M	=	1101		13/16
P	=	0000 0000		
$Y_M\ x_{-4}$	=	1101 0000		
$\times 2^{-1}$	=	0110 1000	+	
$Y_M\ x_{-3}$	=	1101		

		10011 1000		
$\times 2^{-1}$	=	1001 1100	+	
$Y_M\ x_{-2}$	=	0000		
	=	-------------		
$\times 2^{-1}$	=	0100 1110	+	
$Y_M\ x_{-1}$	=	1101		

		10001 1110		
$\times 2^{-1}$	=	1000 1111		
P	=	01000 1111	=	+143/256 în baza 10.

4.9.2. Metoda Robertson pentru înmulţirea directă în cod complementar (C2)

Există mai multe variante ale algoritmului Robertson pe care-l prezentăm în continuare, dar principiul de bază este acelaşi: se folosesc numere reprezentate în cod complementar, iar în descompunerea secvenţială a relaţiei lui P = X*Y se vor considera şi biţii de semn la fel ca şi cei ce indică mărimea numărului. Ca urmare la deplasările efectuate către dreapta a numerelor binare (împărţiri repetate la doi) trebuie ţinut cont de faptul că se păstrează nemodificată valoarea bitului de semn. Dacă la adunarea anterioară deplasării apare depăşire a capacităţii de reprezentare (lucru sesizat prin valoarea V=1 (fanion overflow) atunci bitul de semn rezultat trebuie modificat deoarece este eronat. Ca urmare, dacă V=1 în urma adunării, se face deplasarea registrului care conţine produsul parţial şi apoi se modifică valoarea bitului de semn conform

relaţiilor (4.39) şi (4.40):

Pentru întregi:

$p_{i+1} \to p_i$ şi

$p_{2n-1} \oplus V \to p_{2n-1}$ pentru i între 0 şi n-2 (4.39)

iar pentru subunitare

$p_{-i} \to p_{-(i+1)}$ şi

$p_0 \oplus V \to p_0$ pentru i între 0 şi n-1 (4.40)

Pentru deducerea algoritmului se va face descompunerea relaţiei produsului în paşi succesivi, iar pentru a considera semnul ultimei operaţii, care dictează adunare sau scădere la pasul i se introduce o variabilă notată δ_i. În ultima etapa (n-1) valoarea bitului de semn determină prin valoarea sa adunarea lui 0 (pentru 0), respectiv scăderea lui Y (pentru 1). În practică există diverse variante ale algoritmului prezentat ce rezultă tocmai din modul de interpretare a operaţiei la ultimul pas. De exemplu pentru a evita scăderea la ultimul pas se poate modifica semnul înmulţitorului astfel încât el să fie întotdeauna pozitiv, modificând în mod corespunzător, dacă este necesar, semnul rezultatului final prin complementare.

$$X = (-1)x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i \ ; \qquad Y = (-1)y_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} y_i \cdot 2^i \ ; \qquad (4.41)$$

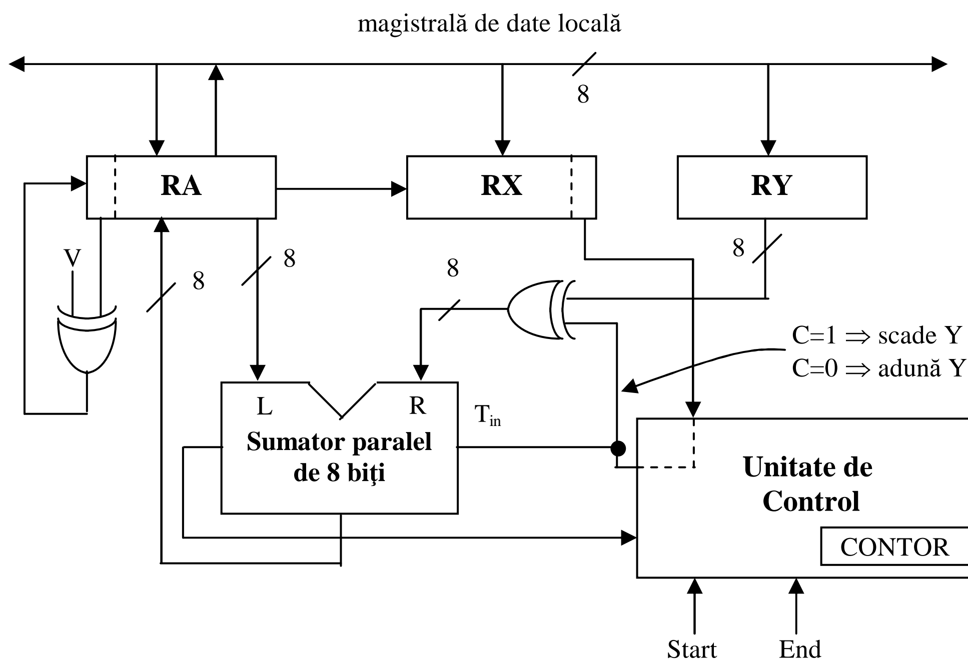


Figura 4.21. Schema bloc a unui circuit ce poate realiza înmulţirea secvenţială a numerelor întregi cu semn codificate pe 8 biţi în cod complementar

$$P = x_0 2^0 Y + x_1 2^1 Y + x_2 2^2 Y + \dots x_{n-2} 2^{n-2} Y - x_{n-1} 2^{n-1} Y$$

$$P = 2^{-1}(-x_{n-1}Y^* + 2^{-1}(x_{n-2}Y^* + 2^{-1}(\quad \dots \quad + 2^{-1}(x_1 Y^* + 2^{-1}(x_0 Y^*))\dots) \qquad (4.42)$$

cu notaţia:

$$Y^* = Y \cdot 2^n \qquad (4.43)$$

Conform relaţiei (4.42) se va implementa algoritmul următor:

Input X, Y,
$Y^* = 2^n Y$
P=0, $\delta_i = 0$ pentru i=0,(n-2); $\delta_{n-1} = 1$
Pentru i=0 la n-1
    $P = 2^{-1}(P + (-1)^{\delta i} Y^* x_i)$
STOP

În figura 4.21 se prezintă o variantă a circuitului pe care se poate implementa algoritm. De notat micile deosebiri faţă de circuitul din figura 4.20: sumatorul este pe 8 biţi, la intrarea ALU s-a prevăzut un bloc de porţi sau-exclusiv pentru a putea complementa factorul Y, iar bitul de semn se setează şi în funcţie de indicatorul de depăşire a capacităţii de reprezentare.

Pentru numere subunitare cu semn, descompunerea relaţiei produsului se face ca în relaţiile următoare:

$$X = (-1)x_0 + \sum_{i=1}^{n-1} x_{-i} \cdot 2^{-i}; \qquad Y = (-1)y_0 + \sum_{i=1}^{n-1} y_{-i} \cdot 2^{-i} \qquad (4.44)$$

$$P = -x_0 2^0 Y + x_{-1} 2^{-1} Y + x_{-2} 2^{-2} Y + \dots + x_{-(n-1)} 2^{-(n-1)} Y$$

$$P = -x_0 Y + 2^{-1}(x_{-1}Y + 2^{-1}(x_{-2}Y + 2^{-1}(\quad \dots \quad + 2^{-1}(x_{-(n-1)}Y))\dots) \qquad (4.45)$$

Exemplul 4. Funcţionarea algoritmului pentru (-3)×(-6) reprezentate cu 3+1 biţi.

X	=	1101		-3 în cod complementar
Y	=	1010		-6 în cod complementar
-Y		0110		+6
P	=	0000 0000		
$Y^*=Y\ 2^4$	=	1010 0000		
$Y^*\ x_0$	=	1010 0000		$x_0 = 1$
$\times 2^{-1}$	=	1101 0000		(deplasare dreapta cu păstrare semn)
$Y^* x_1$	=	0000 0000		$x_1 = 0$
$\times 2^{-1}$	=	1110 1000	+	
$Y^* x_2$	=	1010 0000		$x_2 = 1$
	=	1000 1000		V=0, semn corect
$\times 2^{-1}$	=	1100 0100	+	
$-Y^* x_3$	=	0110 0000		$x_3 = 1$
	=	0010 0100		V=0, semnul este corect
$\times 2^{-1}$	=	0001 0010		PRODUSUL

Exemplul 5. pentru (-5/8)×(-6/8), n=4.

X	=	1011		-5/8
Y	=	1010		-6/8
-Y	=	0110		+6/8
P	=	0000	+	
i=3, $Y \cdot x_3$	=	1010		
	=	1010		V=0, semn corect
$\times 2^{-1}$	=	1101 0	+	
i=2, $Y \cdot x_2$	=	1010		
	=	0111 0		V=1, semnul se schimbă după shift
$\times 2^{-1}$	=	**1**011 10		
i=2, $Y \cdot x_1$		0000		
$\times 2^{-1}$		1101 110		
$-Y \cdot x_0$		0110		corecţie
		0011 110		V=0, semn corect
		0011 110		PRODUS

4.9.3. Algoritmul Booth pentru înmulţirea numerelor reprezentate în complement faţă de doi

Spre deosebire de algoritmul anterior, algoritmul Booth tratează în mod uniform atât operanzii pozitivi cât şi pe cei negativi. Scopul construcţiei algoritmului este creşterea vitezei de execuţie, lucru valabil însă doar dacă în numerele înmulţite nu există secvenţe de doi sau mai mulţi biţi succesivi cu valoare identică (deci şiruri de 0 sau de 1).

Algoritmul se bazează pe transformările de mai jos. Astfel, pentru numere întregi cu semn se poate scrie:

$$2X = -2x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^{i+1} = -2x_{n-1} \cdot 2^{n-1} + \sum_{i=1}^{n-1} x_{i-1} \cdot 2^i \qquad (4.46)$$

$$X = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i = -2x_{n-1} \cdot 2^{n-1} + \sum_{i=1}^{n-1} x_i \cdot 2^i + x_0 2^0 \qquad (4.47)$$

Se poate scrie că:

$$X = 2X - X = \sum_{i=1}^{n-1} (x_{i-1} - x_i) \cdot 2^i - x_0 2^0 = \sum_{i=1}^{n-1} \Delta x_i \cdot 2^i - x_0 2^0 \qquad (4.48)$$

şi folosind notaţia: $\Delta x_i = x_{i-1} - x_i$, pentru i = 1,(n-1), expresia produsului se poate scrie:

$$P = X \cdot Y = -x_0 2^0 Y + \Delta x_1 2^1 Y + \Delta x_2 2^2 Y + \ldots + \Delta x_{n-1} 2^{n-1} Y$$

$$P = Y \cdot 2^n (\Delta x_{n-1} 2^{-1} + \Delta x_{n-2} 2^{-2} + \ldots + \Delta x_1 2^{n-1} - x_0 2^{-n})$$

$$P = 2^{-1}(\Delta x_{n-1} Y^* + 2^{-1}(\Delta x_{n-2} Y^* + 2^{-1}(\quad \ldots \quad + 2^{-1}(\Delta x_1 Y^* + 2^{-1}(-x_0 Y^*))\ldots) \qquad (4.49)$$

Pe baza relaţiei de implementare a produsului se observă că în funcţie de Δx_i se efectuează următoarele operaţii:

$\Delta x_i =$ 0 - se face doar deplasare (există o succesiune de forma 00 sau 11)

$\Delta x_i =$ 1 - se face adunarea lui Y la produsul parţial şi apoi deplasare (succesiune de forma 01)

$\Delta x_i =$ -1 - se face scăderea lui Y din produsul parţial şi apoi deplasare (succesiune 10)

Pentru numere subunitare algoritmul se deduce conform relaţiilor:

$$2X = -2x_0 + \sum_{i=1}^{n-1} x_{-i} \cdot 2^{-i+1} = -2x_0 + \sum_{i=0}^{n-1} x_{-(i+1)} \cdot 2^{-i} \qquad (4.50)$$

cu $x_{-n} = 0$

$$X = -x_0 + \sum_{i=1}^{n-1} x_{-i} \cdot 2^{-i} = -2x_0 + \sum_{i=0}^{n-1} x_{-i} \cdot 2^{-i} \qquad (4.51)$$

Sau efectuând acelaşi artificiu ca la întregi:

$$X = 2X - X = \sum_{i=0}^{n-1} (x_{-(i+1)} - x_{-i}) \cdot 2^{-i} = \sum_{i=0}^{n-1} \Delta x_i \cdot 2^{-i} \qquad (4.52)$$

care conduce la o dezvoltare similară a produsului cu cea din ecuaţia (4.49).

Aşa cum s-a menţionat la începutul acestui paragraf dacă biţii 1 şi biţii 0 alternează, nu rezultă nici o reducere a numărului de operaţii efectuate. Din această cauză adesea se lucrează cu algoritmul *Booth modificat*, numit şi "***înregistrarea Booth***". Explicaţia constă în re-codificarea numerelor binare conform descrierii făcute în continuare. Procesul inspectării biţilor se face folosind o codificare cu 3 caractere: 0, 1, şi $\underline{1}$ unde:

0 înseamnă deplasare (relativ la acumulator)

1 înseamnă adună Y înainte de deplasare

($\underline{1}$) înseamnă scade Y înainte de deplasare

De exemplu pentru numărul binar corespunzător lui +24226 în baza 10:

$$X = 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0$$

se înlocuieşte prin codul X*

$$X^* = 1\ \underline{1}\ 1\ 0\ 0\ 0\ \underline{1}\ 1\ \underline{1}\ 1\ \underline{1}\ 0\ 0\ 1\ \underline{1}\ 0$$

unde înregistrarea Booth notată X* este un număr format din "*cifre binare cu semn*". Este o variaţie a notaţiei poziţionale obişnuite care permite ca un bit să aibă ponderea -2^i (pentru $\underline{1}$) sau $+2^i$ (pentru 1). Ca urmare putem scrie

$$X^* = 2^{15} - 2^{14} + 2^{13} - 2^9 + 2^8 - 2^7 + 2^6 - 2^5 + 2^2 - 2^1 = 24226$$

După această operaţie urmează re-codarea înmulţitorului astfel încât să se indice doar operaţiile ce se vor efectua. Acolo unde există biţi 1 izolaţi (succesiune 1$\underline{1}$) se efectuează doar adunare (în loc de adunare şi scădere, deci ***1$\underline{1}$* se înlocuieşte cu 01**), iar acolo unde există biţi 0 izolaţi (succesiune $\underline{1}$1) se efectuează doar scădere (în loc de scădere şi adunare, deci ***$\underline{1}$1 se re-codează cu 0$\underline{1}$***). Pentru codul X* cu "cifre binare cu semn", se face re-codarea după exemplul de mai jos (citirea tuturor combinaţiilor binare cu semn s-a făcut de la stânga la dreapta):

$$X^* = \underbrace{1\ \underline{1}}\ 1\ 0\ 0\ 0\ \underbrace{\underline{1}\ 1}\ \underbrace{\underline{1}\ 1}\ \underline{1}\ 0\ 0\ \underbrace{1\ \underline{1}}\ 0$$

$$Xr = 0\ 1\ 1\ 0\ 0\ 0\ 0\ \underline{1}\ 0\ \underline{1}\ \underline{1}\ 0\ 0\ 0\ 1\ 0$$

Dacă se consideră înmulţirea (24226) $\times$ (2), putem scrie în zecimal cele două variante de înmulţire, cu numărul din înregistrarea Booth şi apoi cu numărul din înregistrarea Booth re-codată, ca în figura 4.30. Înlocuirea codurilor din înregistrarea Booth prin re-codare se face pentru a înlătura

două operaţii succesive (una de adunare şi alta de scădere) atunci când există biţi 1 sau biţi 0 izolaţi

Motivul re-codării este simplu de explicat, dacă considerăm că la ponderea i (2^i) a cuvântului se găseşte un 1 izolat între două zerouri aflate la ponderile binare 2^{i+1} şi respectiv 2^{i-1} . Prin parcurgerea de la dreapta la stânga a secvenţei ...010... se pot scrie diferenţele Δx_i din algoritmul Booth ca:

$$(0-1)\times 2^i + (1-0)\times 2^{i+1} = -2^i + 2^{i+1} = -2^i + (2^i + 2^i) = 1\times 2^i$$

deci în loc de o scădere urmată de o adunare (*11*), se face doar adunare.

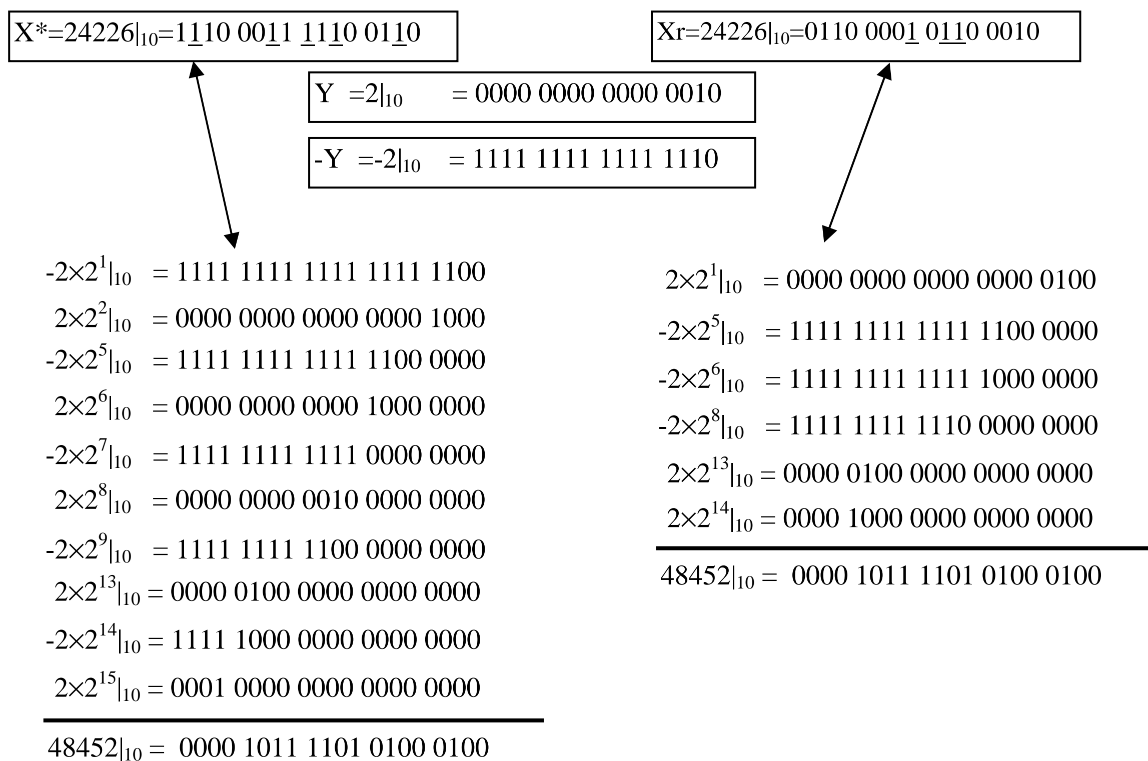


Figura 4.22. Exemplu de înmulţire prin algoritmul Booth modificat, în partea stângă cu numere binare cu semn, iar în partea dreaptă pentru înregistrarea Booth re-codată care elimină pierderile de timp la biţi izolaţi cu valoarea 1 sau 0.

Pentru ca algoritmul Booth modificat să poată fi implementat corect înainte de codare şi înmulţire trebuie identificaţi biţii izolaţi. Există diferite metode de identificare a biţilor izolaţi, care se bazează pe diferenţa calculată între doi biţi vecini şi setarea unor fanioane de control când semnul se schimbă la doi paşi succesivi.

4.10. Circuite de înmulţire construite cu reţele combinaţionale

Dezvoltarea tehnologiei VLSI a făcut posibilă construirea de circuite combinaţionale ce realizează înmulţiri de n × n biţi pentru valori relativ mari ale lui n. Aceste circuite multiplicatoare digitale, seamănă cu înmulţitoarele secvenţiale în n paşi discutate mai sus, dar au de n ori mai multă logică, pentru a permite calcularea produsului într-un singur pas cu viteză foarte mare. Aceste circuite sunt compuse de obicei din reţele mari de elemente combinaţionale simple, fiecare din ele implementând o operaţie de tip adunare sau scădere şi deplasare ale operanzilor înmulţirii.

Dacă se consideră înmulţirea a două numere binare întregi fără semn, notate X şi Y, cu expresiile din relaţia (4.53), se poate calcula produsul P cu expresia din relaţia (4.54).

$$X = \sum_{i=0}^{n-1} x_i 2^i \qquad Y = \sum_{i=0}^{n-1} y_i 2^i \tag{4.53}$$

$$P = \sum_{i=0}^{n-1} x_i 2^i \times Y = \sum_{i=0}^{n-1} 2^i \left(\sum_{j=0}^{n-1} x_i y_j 2^j \right) \tag{4.54}$$

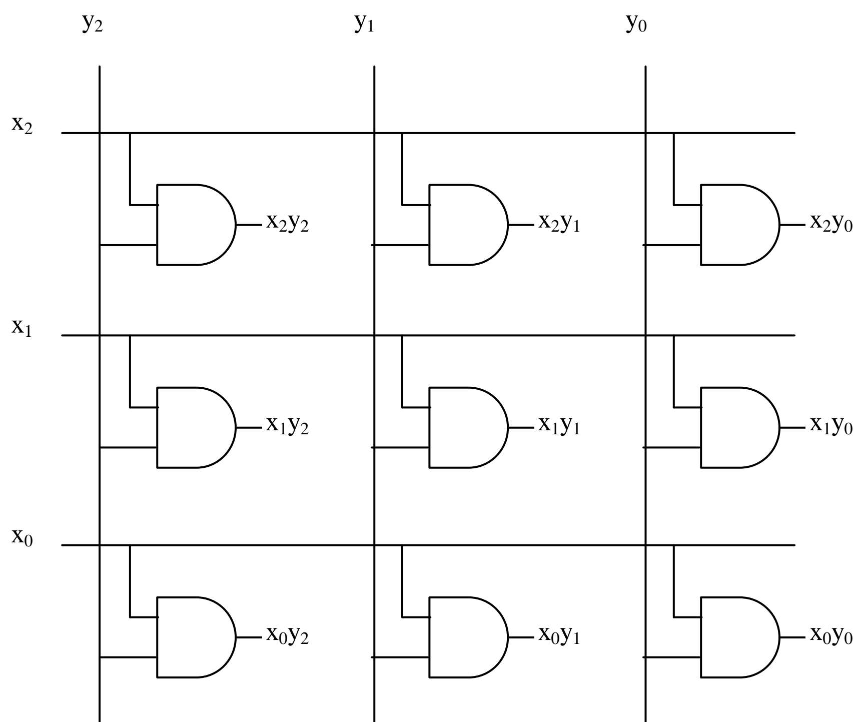


Figura 4.23. Reţea de n × n porţi ŞI logic pentru calcularea produselor între doi biţi, pentru cazul n = 3.

Fiecare din cele n^2 produse $x_i y_j$ de câte un bit din relaţia (4.54) poate fi calculat cu o poartă ŞI (AND) cu 2 intrări. De observat că produsele aritmetic şi logic coincid în cazul operaţiilor cu un

singur bit. În acest fel un circuit cu n × n porţi ŞI cu două intrări poate calcula termenii $x_i y_j$. În figura 4.23 se exemplifică un asemenea circuit pentru n = 3.

În afară de această reţea de porţi ŞI, sumarea se face cu un circuit complex ce cuprinde n(n-1) sumatoare complete. Deplasările implicate sunt implementate prin deplasări spaţiale. Dacă luăm de exemplu două numere binare de câte 3 biţi putem scrie produsul ca în figura 4.24.

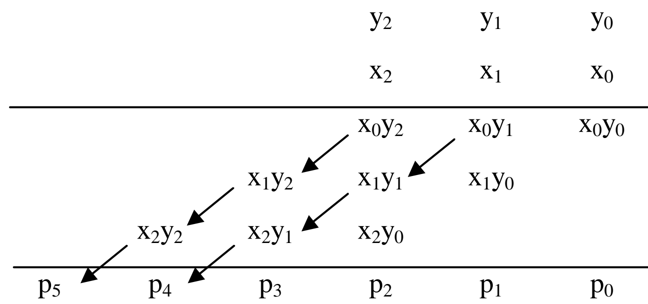


Figura 4.24. Exemplu de implementare a deplasărilor spaţiale pentru înmulţirea a două numere binare de câte 3 biţi.

Circuitul de înmulţire va avea structura simplificată din figura 4.25 în care blocurile componente sunt sumatoare binare complete de 2 biţi:

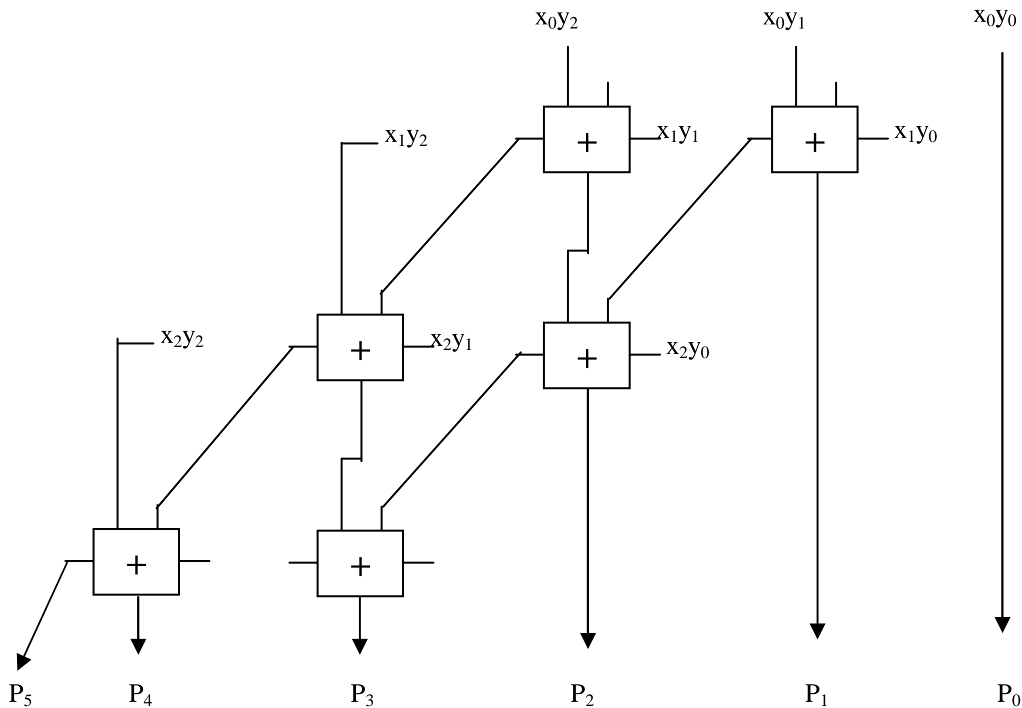


Figura 4.25. Exemplu de circuit combinaţional pentru înmulţirea a două numere binare fără semn de câte 3 biţi. Dreptunghiurile notate cu + sunt sumatoare complete de 2 biţi, care au sus intrările, în dreapta intrarea de transport de la rangul anterior, în stânga transportul către rangul următor, iar în partea de jos bitul sumă

4.11. Împărţirea numerelor binare

La operaţia de împărţire apar dificultăţi suplimentare faţă de înmulţire, deoarece nu există un algoritm de calcul "direct" ca cel dedus la înmulţire. De asemenea la împărţire apar două probleme ce trebuiesc rezolvate:

1. Determinarea sfârşitului operaţiei; mai ales dacă este utilizată reprezentarea cu numere fracţionare, numărul de biţi ai câtului nu mai este limitat apriori (de exemplu: 0,2/0,3 = 0,66...). Este deci necesar să se limiteze numărul de biţi ai câtului unei împărţiri (prin trunchiere sau rotunjire)
2. Trebuie implementat procedeul pentru rotunjirea rezultatului.

Un circuit de împărţire secvenţială în n paşi poate avea schema bloc din figura 4.26. Se observă că structura generală este identică cu cea de la înmulţirea secvenţială. Pentru a calcula raportul dintre deîmpărţit (D) şi împărţitor (V), se încarcă D în registrul RX, V în registrul RY, iar RA se umple cu zero, iar apoi se efectuează paşii:

1. Se deplasează perechea de registre RA.RX un bit la stânga
2. Se scade conţinutul lui RY din RA
3. Dacă rezultatul la pasul 2 este negativ se poziţionează LSb al lui RX la 0, iar dacă este pozitiv sau zero se poziţionează la 1.
4. Dacă rezultatul pasului 2 este negativ se reface vechea valoare a lui P prin adunarea lui RY la RA.

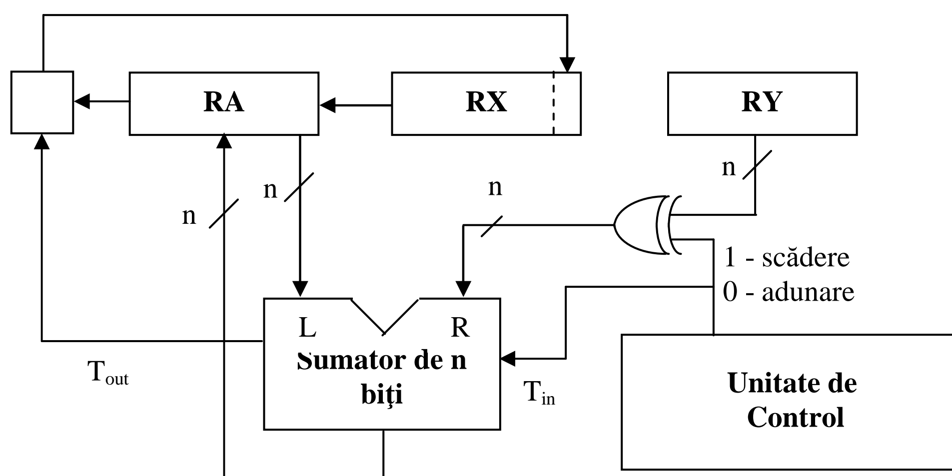


Figura 4.26. Schema bloc a unui circuit de împărţire a numerelor întregi fără semn, în n paşi secvenţiali.

După n etape în registrul RA se va găsi restul împărţirii iar în registrul RX câtul împărţirii. descrierea secvenţială a celor patru paşi enumeraţi mai sus constituie varianta binară a metodei împărţire cu creion şi hârtie.

Putem scrie:

$$D = Q \times V + R \tag{4.55}$$

unde Q este câtul, iar R este restul împărţirii, care respectă condiţia $0 \le R < V$

Dacă se consideră împărţirea a două numere întregi pozitive, D (deîmpărţit) şi V (împărţitor) pe n biţi, în primă fază se va face alinierea deîmpărţitului cu împărţitorul, la fel ca la împărţirea manuală. Aliniere constă în deplasarea stânga a împărţitorului până este aliniat cu cel mai semnificativ bit cu valoarea 1 din D: V' = V 2^k. Notăm resturile parţiale cu R şi indice. Presupunem ca exista q_k astfel încât:

$$R_0 = D = V' \cdot q_k + R_1 \qquad q_k \in \{0,1\} \text{ şi } R_1 < V'$$

$$2R_1 = V' \cdot q_{k-1} + R_2$$

$$2R_2 = V' \cdot q_{k-2} + R_3$$

....

$$2R_k = V' \cdot q_0 + R_{k+1}$$

rezultă:

$$D = V'(q_k 2^0 + q_{k-1} 2^{-1} + q_{k-2} 2^{-2} + ... + q_0 2^{-k}) + 2^{-k} R_{k+1} \quad (4.56$$

calcul care se poate efectua în k paşi secvenţiali după dezvoltarea:

$$D = V' q_k + 2^{-1}(V' q_{k-1} + 2^{-1}(V' q_{k-2} + 2^{-1}(... + 2^{-1}(V' q_0 + R_{k+1}))...) \tag{4.57}$$

în care q_k = 0 din condiţia de ne-depăşire a capacităţii.

Înlocuind $V' = 2^k \cdot V$ rezultă

$$D = V\underbrace{(q_k 2^k + q_{k-1} 2^{k-1} + q_{k-2} 2^{k-2} + ... + q_0 2^0)}_{Q = \text{cât întreg}} + \underbrace{2^{-k} R_{k+1}}_{\text{corecţie rest}} \tag{4.58}$$

Se observă că numărul de paşi de calcul este k+1. Continuarea procesului cu mai mulţi paşi va începe să producă biţi fracţionari ai lui Q. În acest fel Q poate depăşi domeniul numerelor reprezentabile pe n biţi. La numere întregi fără semn cu n biţi, dacă se interzice împărţirea prin zero, atunci Q∈ {1 - 2^{n-1}}. Dacă D are 2n biţi iar V are n biţi, atunci Q poate depăşi capacitatea registrului cât de n biţi; în acest caz trebuie semnalată depăşirea.

La numere fracţionare condiţia de ne-depăşire pentru Q este |D| < |V|.

Ca urmare a celor spuse anterior, pentru două numere întregi, dacă D este reprezentat pe 2n biţi iar V pe n biţi, se verifică mai întâi condiţia de ne-depăşire a capacităţii pe n biţi pentru cât:

$D < V \cdot 2^n$ iar $R_1 = D < V'$ condiţie de ne-depăşire

$$2R_1 = V' \cdot q_{n-1} + R_2$$

$$2R_2 = V' \cdot q_{n-2} + R_3$$

....

$$2R_k = V' \cdot q_0 + R_{k+1}$$

Pentru subunitare:

$$D = R_1$$

$$2R_1 = V \cdot q_{-1} + R_2$$

$$2R_2 = V \cdot q_{-2} + R_3$$

....

$$2R_k = V \cdot q_{-k} + R_{k+1}$$

rezultă:

$$D = V(q_{-1}2^{-1} + q_{-2}2^{-2} + q_{-3}2^{-3} + ... + q_{-k}2^{-k}) + 2^{-k}R_{k+1} \quad (4.59)$$

Rezultă că operaţia de împărţire se poate efectua astfel:

1. se aliniază biţii MSb diferiţi de zero ai lui D şi V, (sau deplasare stânga cu k poziţii pentru V)
2. se verifică condiţia de ne-depăşire
3. V' se deplasează o poziţie la dreapta şi se face comparare (comparare prin scădere); dacă rezultatul este pozitiv, se face scădere iar bitul curent (q) al câtului este 1. Dacă rezultatul este negativ, nu se face scădere iar q curent este 0
4. se repetă pasul anterior (3) până când V' ajunge aliniat la dreapta.

Operaţia efectuată în pasul 3 poate fi descrisă prin relaţia:

$$R_{i+1} \leftarrow R_i - q_i 2^{-i} V' \quad (4.60)$$

În calculator este mai uşor de făcut însă deplasarea restului parţial la stânga, faţă de împărţitorul fix şi nu deplasarea (alinierea) împărţitorului. Ca urmare operaţia ce se va efectua va fi scrisă:

$$R_{i+1} \leftarrow 2R_i - q_i V \quad (4.61)$$

Exemplul 6. Împărţire pentru 17:3

```
          D   =    0001 0001      17
          V   =         0011      3
          V'  =    0011 0000      nu există depăşire (qn = 0)
        2R1   =    0010 0010  –
                   0011 0000
                 ----------------  < 0  rezultă q3 = 0
        2R2   =    0100 0100  –
                   0011 0000
                 ----------------
                   0001 0100      > 0  rezultă q2 = 1
        2R3   =    0010 1000  –
                   0011 0000
                 ----------------  < 0  rezultă q1 = 0
        2R4   =    0101 0000  –
                   0011 0000
                 ----------------
                   0010 0000      > 0  rezultă q0 = 1
   deplasare       0010 0000
R5×2^-4=Rest  =    0000 0010
```

În plus faţă de cele spuse până aici, de obicei compararea în calculator se face prin scădere. Ca urmare dacă rezultatul scăderii dintre restul parţial şi împărţitor este negativ, restul parţial iniţial (de dinaintea scăderii) trebuie refăcut. Există 2 metode de refacere a restului parţial:

a) refacere cu regenerarea restului parţial:

- dacă $R_i - V \geq 0$ în pasul i, restul parţial nu trebuie refăcut şi rezultă că în pasul i+1: $R_{i+1} = 2(R_i - V)$
- dacă $R_i - V < 0$ în pasul i, în acelaşi pas i se reface restul parţial prin adunarea lui V la rezultatul obţinut prin $R_i - V$, iar în pasul i+1 vom avea: $R_{i+1} = 2[(R_i - V) + V]$

b) fără regenerarea restului parţial (mai exact cu refacere în pasul următor)

- dacă $R_i - V \geq 0$ în pasul i, rezulta că în pasul i+1: $R_{i+1} = 2(R_i - V)$
- dacă $R_i - V < 0$ în pasul i, în pasul i+1 vom avea: $R_{i+1} = 2(R_i - V) + V$, unde operaţia din paranteza rotundă a fost efectuată în pasul i şi a rezultat un număr negativ; deci operaţia din pasul i+1 este adunare dacă în pasul i restul parţial a rezultat negativ.

Rezultatul este similar cu cel de la metoda refacerii restului parţial, căci

$$2(R_i - V) + V = 2R_i - 2V + V = 2R_i - V,$$

deci comparaţia pe care dorim să o facem în pasul i+1.

În acest capitol nu ne-am referit la operaţiile de înmulţire şi împărţire în virgulă mobilă, pentru că operaţiile presupun înmulţire (împărţire) de mantise şi adunare (scădere) de exponenţi, deci operaţii ce se reduc la operaţii cu întregi sau fracţionare în virgulă fixă.

4.12. Exerciţii

1. Câţi biţi trebuie să aibă codul binar corespunzător fiecărui element, la codificarea elementelor unei mulţimi A cu 2000 elemente ?:

   a) 100; b) 9; c) 10; d) 11; e) 12; f) 1000.

2. Reprezentaţi fiecare din numerele zecimale următoare, în binar, în codurile direct(MS), invers(C1) si complementar(C2): -2014 ; +22 ; -18 ; -2/8 ; 0 ;
3. Convertiţi următoarele numere binare cu semn reprezentate în cod complementar (complement faţă de 2) în zecimal: 10110110; 11101010; 11011010
4. Convertiţi în binar fără semn numerele zecimale

   a. 9.375 b. 10.375 c. 9.125 d. 9.875
5. Reprezentaţi valorile zecimale următoare în cod NBCD (8421): 2014, 17, 335

Capitolul 5

Unitatea de control a UCP

Conţinut:

5.1. Funcţiile principale ale unităţii de control a UCP
5.2. Control cablat
5.3. Controlul microprogramat
    5.3.1. Metode de optimizare a UC microprogramate
5.4. Paralelism în execuţia instrucţiunilor prin tehnici pipeline
5.5. Exerciţii

5.1. Funcţiile principale ale unităţii de control a UCP

Structura calculatorului, cu componentele de bază descrise de von Neumann, rămâne, în general, valabilă şi astăzi. Partea cea mai importantă a calculatorului este Unitatea Centrală de Procesare (UCP), sau procesorul, în a cărei structură am deosebit în capitolul 1 unităţile de prelucrare a datelor şi de control, astfel că într-un calculator, procesorul conţine totul cu excepţia memoriei şi a sub-sistemului de intrare - ieşire. Funcţional, putem spune că UCP este formată dintr-o parte de calcul (unitate de prelucrare a datelor, sau *cale de date*) şi o parte de control (*cale de control*). Calea de date este un ansamblu de unităţi funcţionale capabile să regăsească datele şi să le prelucreze. Ea cuprinde: unitatea / unităţile aritmetice şi logice, registrele de uz general şi căile de comunicare (magistrale interne UCP) dintre acestea. Rolul unităţii de control a unui procesor ce recunoaşte un set de instrucţiuni, este să aducă instrucţiunile codificate binar din memoria principală (externă procesorului) şi să transmită semnale de control către unitatea de prelucrare (calea de date), şi de asemenea către memorie şi sub-sistemul de I/O, în scopul executării instrucţiunilor. Semnalele de control selectează funcţiile ce trebuiesc executate în calea de date la momente discrete de timp şi conduc datele de la / la unităţile funcţionale potrivite. Momentele de timp individuale sunt definite de către impulsurile ce provin de la circuitul de ceas al UCP. Conform semnalelor emise, unitatea de control (UC) efectuează periodic o *reconfigurare* din punct de vedere logic a unităţii de prelucrare, astfel încât aceasta să execute un set impus de (micro)operaţii. Pentru execuţia corectă a unui program UC are doua funcţii principale: *secvenţierea instrucţiunilor* şi *interpretarea* acestora.

(a) *Secvenţierea instrucţiunilor* se referă la maniera în care controlul procesorului este transferat de la o instrucţiune la alta, prin ordinea de generare a adreselor către memoria principală a calculatorului. Instrucţiunile sunt selectate pentru execuţie într-o anumită ordine, dictată de UC prin interpretarea informaţiei codificate în cadrul instrucţiunilor şi pe baza unor semnale recepţionate de UC de la calea de date sau din exteriorul procesorului. Semnalele din exteriorul procesorului sunt asincrone cu programul executat.

În principiu, fiecare instrucţiune ar putea conţine adresa următoarei instrucţiuni ce trebuie adusă din memorie. Cele mai multe instrucţiuni dintr-un program au un succesor unic; astfel că dacă o instrucţiune I_i este stocată în memorie într-o locaţie de adresă A, iar I are ca unic succesor instrucţiunea I_{i+1}, atunci este natural să se stocheze I_{i+1} în locaţia de adresă A+1 ce urmează imediat după A. Adresa instrucţiunii I_i este conţinută într-un registru de adresare a instrucţiunilor, iar adresa instrucţiunii I_{i+1} poate fi obţinută pentru cazul de mai sus prin incrementare. Din această cauză vom numi registrul de adresare al instrucţiunilor "*contor de*

program”, notat pe scurt PC. Contorul de program este un indicator al adresei instrucţiunii curente. Adresa instrucţiunii I_{i+1} se calculează prin operaţia:

$$PC \leftarrow PC + c \quad (5.1)$$

unde c reprezintă numărul de cuvinte de memorie pe care se întinde instrucţiunea I_i.

În programe există de asemenea instrucţiuni de ramificare, sau se pot produce situaţii independente de program, care impun UC să facă trecerea către instrucţiuni aflate la adrese ne-consecutive. UC trebuie să controleze modul în care se fac aceste salturi, care pot fi necondiţionate, condiţionate (prin testarea unor indicatori de condiţii), sau pot fi salturi pentru transferul controlului către un alt program sau subprogram (proceduri, întreruperi, excepţii). În cazul unei ramificări, registrul de adresare PC se încarcă direct cu adresa la care se face saltul. După aceasta funcţionarea PC se face conform relaţiei (5.1) până la următoarea instrucţiune de ramificare.

(b) Interpretarea instrucţiunilor se referă la modul în care UC decodifică codul fiecărei instrucţiuni şi la modalitatea de generare a semnalelor de control către calea de date, pentru a comanda execuţia instrucţiunilor. Comportarea UC din punctul de vedere al semnalelor de control generate (ca funcţie şi ca succesiune în timp) pentru comanda operaţiilor se poate descrie pe baza unor tabele de tranziţie a stărilor, organigrame, limbaje de descriere, dar şi pe baza unor combinaţii ale metodelor amintite. Fiecărei instrucţiuni interpretate de UC, îi corespunde o succesiune de funcţii logice de ieşire din UC, conform algoritmului dorit de proiectantul UC.

Tipurile de semnale de intrare şi ieşire dintr-o unitate de control tipică pot fi descrise cu ajutorul figurii 5.1.

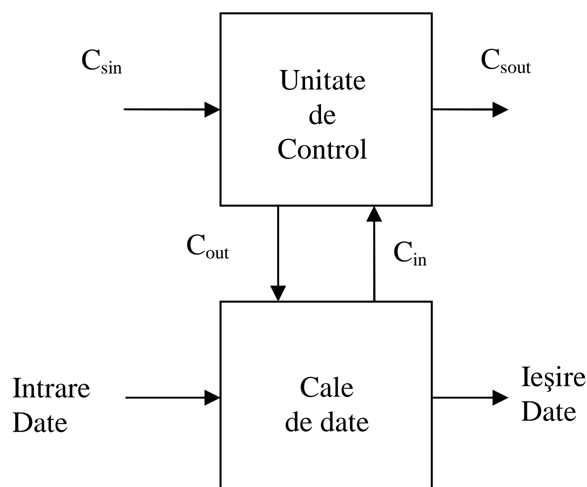


Figura 5.1. Semnale de intrare şi ieşire al unităţii de control la interfaţa cu calea de date şi cu alte sub-sisteme

Cele 4 grupuri de semnale de control indicate în figura 5.1. au următoarele funcţii:

1. C_{out} - sunt semnale ce controlează direct funcţionarea căii de date. Generarea acestor semnale reprezintă principală sarcină a UC.
2. C_{in} - sunt semnale ce permit căii de date să influenţeze funcţionarea unităţii de control în sensul modificării secvenţei de comenzi C_{out} generată. De exemplu, semnalele C_{in} pot indica prezenţa unor condiţii neobişnuite în unitatea de prelucrare a datelor, cum ar fi depăşirea capacităţii de reprezentare ca urmare a unei operaţii aritmetice.
3. C_{sout} - aceste semnale sunt transmise la alte unităţi de control, (ale altor procesoare de uz general sau specific) şi pot indica condiţii de stare, şi semnale de sincronizare, cum ar fi "busy / ocupat", cerere de cedare, sau acceptare a cedării controlului magistralelor sistemului, sau semnale de informare că o operaţie cerută anterior s-a terminat.
4. C_{sin} - Semnale primite de la alte unităţi de control, de exemplu de la un controller supervizor. De obicei includ semnale de *start* şi *stop* şi informaţii de sincronizare, funcţiile fiind similare cu cele ale C_{sout}. Grupurile de semnale C_{sin} şi C_{sout} sunt în primul rând folosite pentru a sincroniza unitatea de control cu operaţiile altor unităţi de control.

Organizarea internă aleasă pentru UC şi aspectele specifice de implementare folosite la proiectarea unităţii, influenţează în mod direct viteza cu care lucrează UC, costurile de proiectare şi construcţie şi suprafaţa ocupată pe circuitul integrat de către aceasta unitate. Acest ultim aspect indică faptul că la microprocesoare, metodele de proiectare ale UC influenţează întreaga arhitectură a UCP prin constrângerile impuse datorită suprafeţei limitate a chip-ului de siliciu.

Orice proiect de unitate de control porneşte de la descrierea setului de instrucţiuni recunoscute şi de la descrierea proiectului căii de date.

Există două metode de proiectare şi implementare a UC: cablat şi microprogramat.

1. *Unitatea de control cablată* este un automat secvenţial proiectat şi construit pentru generarea unui set specific şi într-o secvenţă fixă de semnale de control. Odată ce UC a fost construită, algoritmul de control nu mai poate fi modificat decât prin reproiectarea întregii structuri. La controlul cablat, scopurile principale urmărite la proiectarea automatului se referă la minimizarea numărului de componente utilizate şi maximizarea vitezei de operare.
2. La *unitatea de control microprogramată* semnalele de control sunt incluse în cuvinte binare succesive (microinstrucţiuni) stocate într-o memorie de mare viteză, inclusa în UC, numită *memorie de control*. Implementarea algoritmului de control se face prin programarea conţinutului memoriei de control. Fiecărei instrucţiuni recunoscute de procesor îi corespunde o secvenţă de microoperaţii generate prin citirea unei secvenţe de microinstrucţiuni din memoria de control (secvenţă de microinstrucţiuni numită *microprogram*). Microprogramele

conţinute în memoria de control formează un interpretor al setului de instrucţiuni recunoscut de procesor.

Tehnica cablată are câteva avantaje:

- dimensiuni mici ale UC
- viteză mare de operare

Tehnica microprogramată are şi ea câteva avantaje:

- un set de instrucţiuni recunoscut poate fi uşor schimbat prin schimbarea microprogramelor şi nu prin reproiectarea întregii UC
- permite o metoda sistematica de proiectare a UC
- dacă este nevoie, permite modificarea dinamică (în timpul rulării programelor) a setului de instrucţiuni recunoscut de UC prin utilizarea unei memorii de control cu scriere-citire

5.2. Control cablat

Unitatea de control este un automat secvenţial proiectat special pentru a recunoaşte un anumit set de instrucţiuni, respectiv pentru a genera set corespunzător (pre-fixat) de semnale de control. Cu cât setul de instrucţiuni este mai complex, cu atât proiectarea şi optimizarea proiectului este mai dificilă. De aceea acest tip de control este folosit pentru procesoare care recunosc un set regulat şi relativ redus de instrucţiuni (de obicei instrucţiuni cu format de lungime fixă).

Unitatea de control trebuie să lanseze semnalele de comandă pentru aducerea instrucţiunilor din memoria principală, să decodifice codul fiecărei instrucţiuni şi ca urmare să genereze toate semnalele de control către calea de date pentru execuţia completă a instrucţiunii. Cu cât unitatea de control a procesorului recunoaşte un set mai mare şi mai eterogen (lungimi diferite, câmpuri diferite în funcţie de modurile de adresare etc.) de instrucţiuni, cu atât complexitatea unităţii de control creşte.

Ca schemă de principiu, descrisă în figura 5.2., unitatea de control cablată este un automat ce cuprinde un registru de memorare a stării curente, registrul de instrucţiuni, care memorează codul operaţiei şi logică de control (de obicei combinaţională). În schema simplificată din figură nu s-a ţinut cont de faptul că unitatea de control cuprinde şi registre de adresare a memoriei principale şi nu s-au desenat semnalele de control şi sincronizare pentru registrele interne ale unităţii de control. Registrul de instrucţiuni păstrează codul operaţiei instrucţiunii curente pe toată durata execuţiei instrucţiunii. Execuţia poate fi descrisă prin mai multe stări ale automatului, stare următoare fiind generată de logica de control conform algoritmului cablat şi conform reacţiilor de la calea de date. Starea următoare se încarcă în registrul de stare sincron cu impulsul de ceas. Complexitatea logicii de control este proporţională cu numărul de intrări (biţi pentru cod

instrucţiune, reacţie cale de date şi cod al stării curente) şi de numărul maxim de semnale de control de ieşire.

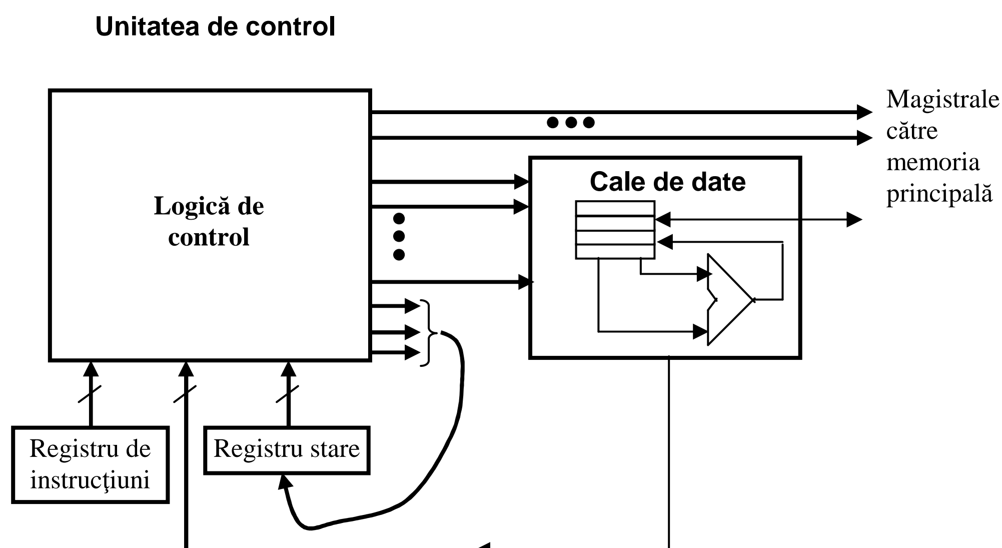


Figura 5.2. Unitate de control cablată, cu o structură simplificată şi relaţia cu calea de date şi memoria principală.

De exemplu, pentru o structură de control cablată, în [Patterson96] se consideră că numărul de linii de intrare este 21 iar numărul de ieşiri 40 şi se calculează dimensiunea logicii de control, dacă ar fi implementată cu ajutorul unui circuit ROM, deci *fără minimizare*. Dimensiunea rezultă $2^{21} \times 40$ biţi = 2M $\times$ 5 $\times$ 8 biţi, adică 10 MB de ROM. Rezultatul indică obligativitatea minimizării logicii de control, din cauza dimensiunii exagerate a unităţii de control, deşi în exemplu s-a considerat un număr relativ mic de intrări şi ieşiri.

Metodele de proiectare ale unităţilor de control cablate sunt metodele folosite pentru implementarea cablată a oricărui automat secvenţial. Metodele moderne de proiectare asistată de calculator includ limbajele de descriere hardware, care permit automatizarea diferitelor faze ale proiectări [Toacse96]. Cu titlu de exemplu prezentăm pe scurt o introducere în metoda de proiectare cu ajutorul unui tabel de tranziţie a stărilor.

Metoda *tabelului de tranziţie a stărilor* este o metodă standard de descriere şi proiectare a automatelor secvenţiale şi poate încorpora tehnici sistematice de minimizare a porţilor şi bistabililor. Tabelul de tranziţie a stărilor cuprinde toate stările automatului de control şi corespondenţa cu stările următoare şi cu semnalele de control generate, în funcţie de combinaţia semnalelor de intrare în automat. În figura 5.3. se prezintă un exemplu general de tabel, în care s-au notat cu C_{in} variabilele de intrare ale UC. Liniile tabelului corespund setului de stări interne $\{S_i\}$ ale maşinii. O stare internă este determinată de informaţia stocată în unitate în momente discrete de timp (perioade de ceas / clock). Informaţia ce corespunde liniei stării S_j şi coloanei

I_j (combinaţia I_j a semnalelor de intrare C_{in} în UC) are forma S_{ij}, C_{ij}, unde S_{ij} reprezintă starea următoare a unităţii de control, iar C_{ij} reprezintă setul de semnale de ieşire corespunzător (C_{out}), semnale activate prin aplicarea setului I_j la intrările unităţii de control când aceasta este în starea S_i.

Stări interne	**Combinaţii de intrare C_{in}**			
	I_1	**I_2**	**...**	**I_m**
S_1	S_{11},C_{11}	S_{12},C_{12}	...	S_{1m},C_{1m}
S_2	S_{21},C_{21}	S_{22},C_{22}	...	S_{2m},C_{2m}
...	...	...	...	...
S_n	S_{n1},C_{n1}	S_{n2},C_{n2}	...	S_{nm},C_{nm}

Figura 5.3. Exemplu general de tranziţie al stărilor. Pentru fiecare stare S_i a automatului la aplicarea unui set de intrări I_j (pe coloană) se indică combinaţia (S_{ij}, C_{ij}) între stare următoare în care va trece automatul şi setul de semnale de control generate în starea actuală.

Această metodă, deşi sugestivă, este destul de greoaie în cazul în care UC are multe stări şi multe combinaţii ale semnalelor de intrare.

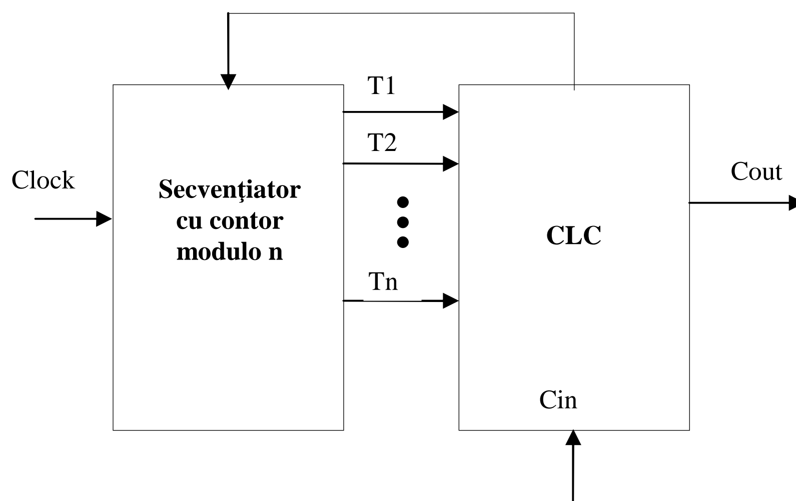


Figura 5.4. Exemplu de implementare cablată a unei unităţi de control cu ajutorul unui secvenţiator modulo n şi a logicii combinaţionale.

mai menţionăm că la descrierea prin tabel de tranziţie a stărilor modul de codificare al fiecărei stări este deosebit de important din punctul de vedere al minimizării funcţiilor rezultate. Pentru aceasta se folosesc programe de proiectare asistată, care ajută la o codificare optimă a stărilor automatului şi apoi la minimizarea circuitelor unităţii de control.

5.3. Controlul microprogramat

Acest mod de construcţie a Unităţii de Control a fost enunţat în 1951 de către Maurice Wilkes [Hayes88]. Soluţia sa a fost să transforme UC într-un calculator în miniatură, ce are două tabele în memoria de control. Primul dintre tabele specifică modul de control a căii de date, iar al doilea determină ordinea de generare (secvenţierea) semnalelor de control pentru operaţiile elementare. Soluţia a fost numita de Wilkes "microprogramare" şi de aici alte denumiri derivate: microinstrucţiune, microcod, microprogram, etc.

Microinstrucţiunile specifică toate semnalele de control pentru calea de date plus posibilitatea de a decide condiţional care microinstrucţiune va fi următoarea executată (figura 5.5). Aşa cum sugerează şi numele "microprogramare", o dată ce calea de date şi memoria pentru microinstrucţiuni, sunt proiectate (şi construite), controlul devine în special o problemă de programare care constă în scrierea unui interpretor pentru setul de instrucţiuni. Microprogramarea permite astfel schimbarea unui set de instrucţiuni prin schimbarea conţinutului memoriei de control (în care se stochează "tabelelor" amintite de Wilkes), fără ca, de cele mai multe ori, să fie necesară şi schimbarea structurii hardware. Structura microprogramului este asemănătoare cu diagrama de stări, în care pentru fiecare stare din diagramă corespunde o microinstrucţiune.

Fiecare bit dintr-o microinstrucţiune reprezintă o comandă elementară care fie merge direct la o resursă comandată (registru, sau CLC), fie este prelucrată împreună cu alte informaţii (pentru decodificare şi sincronizare) pentru ca apoi să se comande resursa fizică.

Fiecare cod de instrucţiune citit şi decodificat, va trebui să producă saltul la o adresa din memoria de control unde se găseşte microprogramul asociat execuţiei acelei instrucţiuni. O posibilitate de efectuare a acestui salt, este chiar folosirea informaţiei din op-code (codul operaţiei) ca index într-un tabel cu adrese de start de microprograme. O altă posibilitate, care creşte eficienţa utilizării memoriei de control, este convertirea codului instrucţiunii curente într-un alt cod care se constituie în index de adresă.

La realizarea unei UC sub forma microprogramată, principalul scop este reducerea costurilor generale de proiectare şi construcţie a UC, de reducere a dimensiunilor microprogramelor şi a dimensiunii microinstrucţiunilor. În mod normal, instrucţiunile care cer cel mai mare timp pentru execuţie sunt optimizate ca viteză, iar celelalte instrucţiuni sunt optimizate

ca spaţiu ocupat în memoria de control.

Avantajele microprogramării, identificate de Wilkes în 1953, rămân valabile şi astăzi:

- microprogramarea permite schimbarea uşoară a algoritmului de control. Această calitate este utilă în faza de dezvoltare când simpla schimbare a unui 0 în 1 în memoria de control, salvează uneori reproiectarea componentelor hardware.
- prin emularea altui set de instrucţiuni, în microcod, se simplifică compatibilitatea software.
- reduce costurile în cazul adăugării unei instrucţiuni mai complexe la o microarhitectura standard.
- flexibilitate - construcţia hardware poate începe înainte ca setul de instrucţiuni şi microcodul să fie complet scris, pentru că specificarea controlului este doar o problema de software.

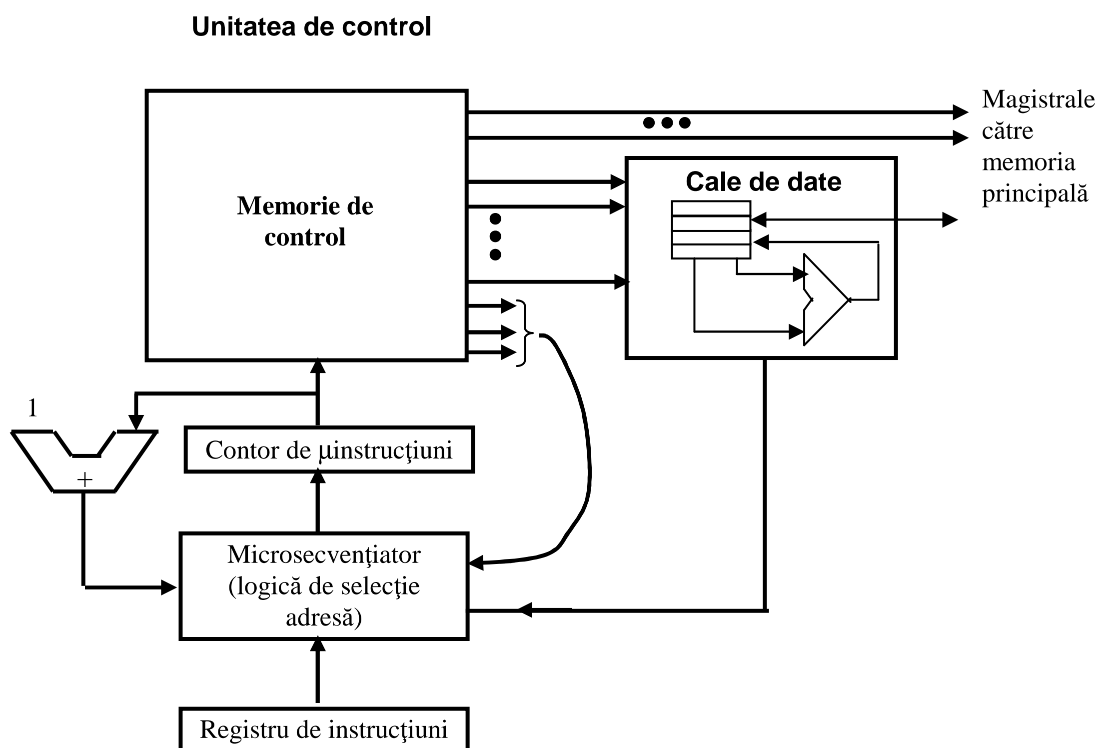


Figura 5.5. O unitate de control microprogramată simplificată. Deosebirile faţă de figura 5.2, cu UC cablată, se referă în primul rând la modul în care se calculează intrările în logica de control (microadresa pentru microinstrucţiune). În plus logica de control este implementată cu ajutorul unei memorii de control , de obicei de tip ROM, care specifică semnalele de control şi semnalele de secvenţiere.

Dezavantajul microcodului este viteza redusă de lucru, pentru că microprogramarea este dependentă de tehnologia de realizare a memoriilor.

Din punct de vedere istoric conceptul microprogramării (1953) a fost cu un pas înaintea tehnologiei timpului şi a rămas aproximativ 10 ani doar o posibilitate virtuală de implementare. Wilkes a construit totuşi unitatea de control a calculatorului numit EDSAC 2 utilizând controlul microprogramat, în 1958, cu o memorie de control ROM construită din miezuri magnetice. De

abia în 1964 firma IBM a folosit microprogramarea la familia IBM 360. S-a realizat chiar emularea lui unei maşini anterioare IBM 7090 cu ajutorul noii maşini IBM 360, care a demonstrat că maşina hardware diferită poate rula aceleaşi programe uneori chiar mai rapid decât pe maşina originală. Pentru operaţia de emulare un calculator microprogramabil C1 poate fi utilizat pentru a executa programe scrise în limbajul maşină L2 al unui alt calculator, C2, prin plasarea unui emulator pentru L2 în memoria de control a lui C1. Calculatorul C1 este deci capabil să emuleze calculatorul C2. Maşina pe care se face emularea este denumita maşină gazdă (host-machine), iar maşina care este simulată, maşină ţintă (target machine). Prin emulare se poate face astfel o testare pe maşina gazdă a unui procesor înainte ca acesta să fie realizat fizic.

Cele mai multe din calculatoarele incluse astăzi în categoria CISC folosesc implementarea microprogramată a algoritmului de control al UC. Principiul structurii de control microprogramate poate fi descris prin schema bloc din figura 5.6. Aşa cum s-a descris în primul paragraf al acestui capitol unitatea de procesare a datelor este comandată pentru a executa funcţii asupra unei structuri de date şi ca urmare pot fi generate semnale ce caracterizează operaţia efectuata (indicatori de condiţii). Comanda funcţiilor şi interpretarea efectului produs se face aici cu un automat construit ca o structură de control microprogramată. Orice operaţie apelată printr-o instrucţiune (recunoscută prin câmpul cod al operaţiei) a maşinii, presupune o secvenţă de paşi ce poate cuprinde transferuri între registre interne, sau registre interne şi registre de memorie. Fiecare din aceşti paşi, realizează o microoperaţie, care se efectuează prin activarea unor semnale asociate registrelor respective. Execuţia unei instrucţiuni maşină este deci compusă dintr-o secvenţă de microoperaţii comandată de un microprogram. Microprogramul este memorat de obicei într-un ROM de control.

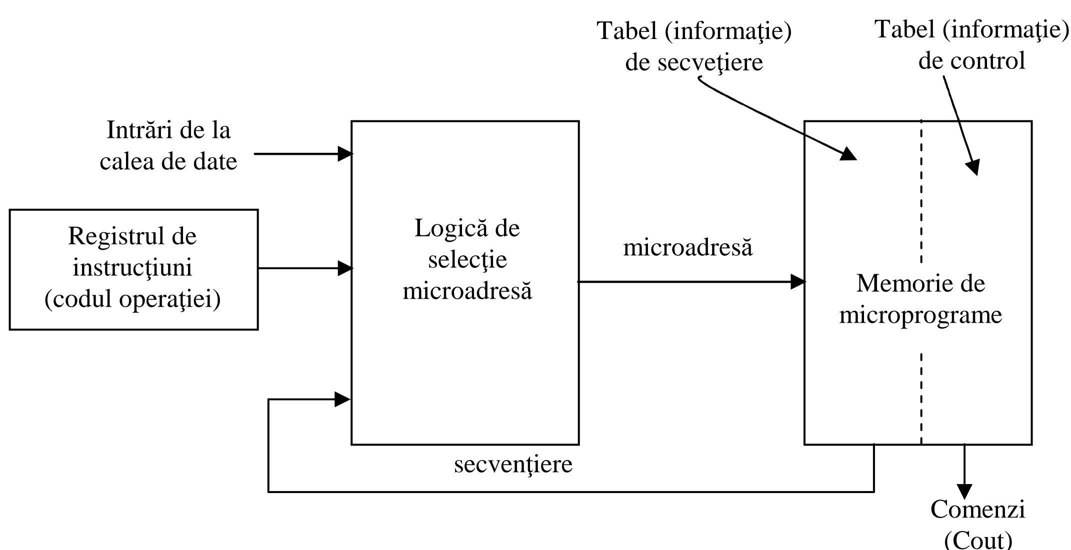


Figura 5.6. Schema bloc de principiu a unei structuri de control microprogramate. Informaţiile de secvenţiere şi control incluse în microinstrucţiunile din memoria de microprograme au fost denumite "tabele" ca în enunţul structurii microprogramate a lui Wilkes.

Logica de selecţie a adresei (microadresei) memoriei de control (sau de microprograme), calculează adresa următoarei microinstrucţiuni pe baza informaţiei din registrul de instrucţiuni (în care se stochează codul operaţiei instrucţiunii), pe baza unor informaţii externe ce provin de la calea de date, sau pe baza informaţiei de secvenţiere (de reacţie) conţinută în corpul microinstrucţiunii curente.

Fiecare locaţie adresabilă a memoriei de control este privită ca o microinstrucţiune. Atunci când locaţia este adresată ea va activa un anumit set de linii de control şi linii de adresă de secvenţiere. Un exemplu de microinstrucţiune, cu exemplificarea unor câmpuri specifice se prezintă în figura 5.7.

Destinaţie	Operaţie ALU	Adr. sursa 1	Adr. sursa 2	Divers	Cond	Adresa următoare

Figura 5.7. Exemplu de microinstrucţiune cu mai multe câmpuri de control şi secvenţiere. În afara operaţiei comandate pentru ALU se specifică codul adresă pentru registrele sursă şi destinaţie, eventuali biţi pentru testarea de condiţii (Cond.) şi informaţii diverse (Divers) de control. Câmpul de secvenţiere (adresa următoare) se dă de obicei ca adresă relativă faţă de adresa microinstrucţiunii curente sau la secvenţe liniare indică doar necesitatea incrementării micro-contorului de program.

Cea mai simplă secvenţă de comandă a unei căi de date este o secvenţă liniară, în care starea următoare a acestui automat simplu ar putea generată de un numărător cu incrementare. În mod obişnuit adresa generată pe baza codului operaţiei instrucţiunii este o adresă de start a unui microprogram din memoria de control. Microprogramul poate fi rulat secvenţial, dar unitatea de control trebuie să aibă şi capacitatea de a răspunde la semnale externe, sau condiţii, prin efectuarea de salturi, deci de întrerupere a secvenţei liniare de microinstrucţiuni. Salturile sunt efectuate pe baza testării semnalelor de la calea de date şi pe baza informaţiei de secvenţiere. Circuitul de generare a adresei următoare din microprogram este numit microsecvenţiator. Astfel că adresarea unei microinstrucţiuni se poate face:

- secvenţial, similar cu adresarea la nivel macro a instrucţiunilor prin contorul de program (PC), micro-adresa fiind generată de un micro-contor de program (µPC);
- salturi condiţionate, prin testarea de către microinstrucţiuni a reacţiilor de la calea de date;
- salturi absolute prin specificatori de micro-adrese următoare(secvenţiere), saltul fiind de obicei relativ la adresa microinstrucţiunii curente.

5.3.1. Metode de optimizare a UC microprogramate

În scopul creşterii vitezei de lucru a UC microprogramate şi de reducere a dimensiunilor memoriei de control principalele metode folosite sunt:

a. reducerea lăţimii microinstrucţiunilor
b. utilizarea structurilor de tip pipeline la nivelul microprogramului
c. utilizarea mai multor niveluri ierarhice de memorii de control

a) Reducerea lăţimii microinstrucţiunilor

Adesea, microinstrucţiunile sunt clasificate în microinstrucţiuni *orizontale* şi *verticale*, în funcţie de numărul de resurse hardware ale maşinii controlate simultan de o microinstrucţiune.

Microinstrucţiunile orizontale controlează foarte multe resurse hardware care funcţionează în paralel. O singură microinstrucţiune orizontală ar putea să controleze funcţionarea simultană şi independentă a uneia sau mai multor unităţi ALU, accesul la memoria principală, generarea condiţionată a adresei următoare, accesul la diverse registre de lucru, bistabili de stare etc. O asemenea microinstrucţiune are însă un număr mare de biţi (tipic peste 64 de biţi).

Microinstrucţiunile verticale controlează simultan un număr mic de resurse hardware şi comandă operaţii simple, de tipul încărcare, adunare, memorare, salt etc. Acest tip de microinstrucţiune se aseamănă în principiu cu instrucţiunile în limbaj maşină, care conţin un cod al operaţiei şi unul sau mai mulţi operanzi. De aceea microprogramarea verticală mai este numită microprogramare software, iar prin comparaţie microprogramarea cu instrucţiuni orizontale este numită microprogramare hardware.

Aceeaşi maşină poate fi controlată cu microprograme lungi şi lente, scrise cu microinstrucţiuni verticale, sau cu microprograme scurte şi rapide, scrise cu microinstrucţiuni orizontale.

Lungimea unei microinstrucţiuni depinde de gradul de codificare al informaţiei conţinute. La o microinstrucţiune fără codificare, fiecare bit controlează o singură resursă sau micro-operaţie. La o microinstrucţiune cu codificare (de exemplu în figura 5.8), câmpuri binare controlează resurse mutual exclusive. Prin codificare, un câmp al microinstrucţiunii va putea activa în acelaşi timp numai un singur semnal de control. De aceea două semnale de control sunt *compatibile* pentru codificare şi pot fi incluse în acelaşi câmp de control numai dacă sunt mutual exclusive (nu este necesar să fie activate simultan de o microinstrucţiune). Pot exista însă mai mult decât două semnale compatibile pentru fiecare câmp ce foloseşte codarea controlului.

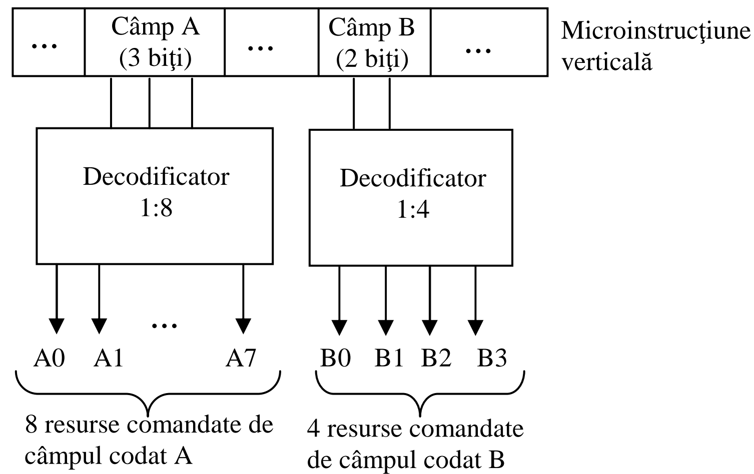


Figura 5.8. Exemplu de microinstrucţiune verticală ce utilizează codarea informaţiei de control în două câmpuri binare. Câmpul A poate comanda simultan una singură din resursele A0 - A7 prin semnalul de la ieşirea decodificatorului.

Concluzionând putem spune că microinstrucţiunile orizontale au caracteristicile: format lung, capacitate de a exprima un grad mare de paralelism (multe comenzi simultane), codare redusă a informaţiei de control. Microinstrucţiunile verticale au ca principale caracteristici; format scurt, capacitate mică de control al microoperaţiilor paralele, codare puternică a informaţiei de control.

b) Utilizarea structurilor pipeline, pentru operaţii în paralel

Deşi execuţia microinstrucţiunii se desfaşoară conform unei secvenţe ce cuprinde fazele de citire, decodificare şi execuţie, detaliile de implementare pot fi diferite de la o soluţie de implementare la alta. Gradul de suprapunere al operaţiilor de execuţie pentru microinstrucţiunea curentă respectiv de acces (găsire şi citire) a microinstrucţiunii următoare este dependent de structura hardware a UC.

Dacă metoda folosită pentru execuţia microprogramelor este serială, citirea microinstrucţiunii următoare nu începe până când nu se termină execuţia microinstrucţiunii curente. Durata unui ciclu de microinstrucţiune (CMI) va fi suma dintre timpul de acces, TA, la memoria de control şi timpul de execuţie, TE, al microoperaţiilor ce controlează resursele hardware ale calculatorului:

$$CMI = TA + TE \qquad (5.2)$$

În acest fel, dacă se consideră valori medii pentru TA şi TE se poate calcula durata medie de execuţie a unui microprogram cu M microinstrucţiuni ca M*(TA+TE).

În relaţia de mai sus timpul de acces (TA) se referă la timopul necesar din momentul lansării

micro-adresei până când microinstrucţiunea poate fi citită la ieşirea memoriei de control. Timpul de execuţie (TE) este timpul minim cât semnalele de control trebuie să rămână active pentru ca operaţia de control să se desfăşoare corect.

Avantajul modului serial de lucru al UC (figura 5.9a şi b) este dat de simplitatea hardware a unităţii şi ca urmare de costul redus. În plus UC microprogramată serială nu trebuie să controleze simultan execuţia şi citirea şi nu pune probleme în cazul apariţiei microinstrucţiunilor de salt condiţionat.

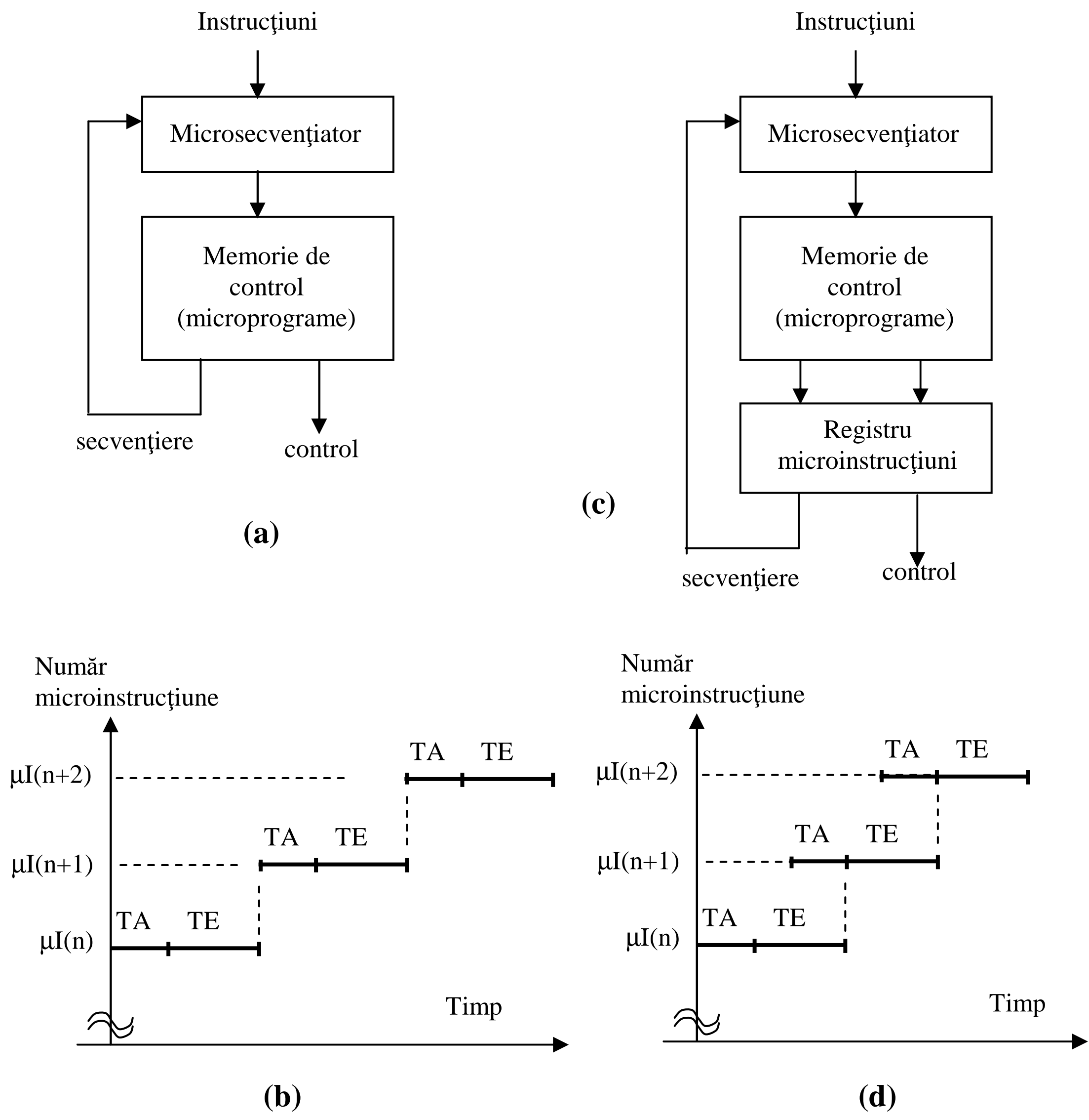


Figura 5.9. Structura bloc a unităţii de control şi distribuirea în timp a execuţiei microinstrucţiunilor. (a) structura bloc cu funcţionare serială, (b) sincronizarea în timp a execuţiei a trei microinstrucţiuni succesive, (c) structura bloc cu registru pipeline de microinstrucţiuni, (d) sincronizarea în timp a execuţiei a trei microinstrucţiuni succesive.

Prin adăugarea unui etaj suplimentar în UC (etaj constituit dintr-un registru de microinstrucţiuni) citirea microinstrucţiunii următoare este realizată în paralel cu execuţia microinstrucţiunii curente. Această structură de tip pipeline (bandă de operaţii, sau conductă de operaţii) are avantajul unei viteze de lucru mai mari, pentru că execuţia microinstrucţiunii curente începe imediat după terminarea execuţiei celei precedente. În această implementare valoarea CMI va fi dată de maximul dintre timpul de acces la memoria de control şi timpul de execuţie: CMI = max(TA,TE). Durata minimă a unui microprogram de M microinstrucţiuni este în acest caz TA+M*TE. Ca dezavantaj, în afara de circuitele suplimentare, reprezentate în principal de registrul pentru memorarea microinstrucţiunii curente (registrul pipeline), la UC microprogramată din figura 5.9 c şi d se pune şi problema salturilor condiţionate. În acest caz este nevoie de un mecanism de anticipare a adresei următoarei microinstrucţiuni. Dacă anticiparea este corectă, în majoritatea cazurilor, nu se va pierde din viteză. În cazul unei anticipări greşite se va pierde însă cel puţin un ciclu pentru citirea corectă a microinstrucţiunii ce urmează.

c) *Organizarea pe mai multe niveluri a memoriei de control*

Memoria de control poate fi organizată pe unul sau pe două niveluri. Memoria structurată pe două niveluri foloseşte noţiunile de microinstrucţiune şi microprogram pentru nivelul superior şi nanoinstrucţiune, respectiv nanoprogram pentru nivelul inferior. Aşa cum o instrucţiune este executată prin rularea unui microprogram (la un singur nivel de memorie de control), fiecare microinstrucţiune este executată prin rularea unui nanoprogram asociat. Tehnica nanoprogramării este echivalentă conceptual cu microprogramarea, iar structurarea memoriei de control pe două niveluri (vezi figura 5.10) oferă o flexibilitate mai mare în proiectarea microinstrucţiunilor de la nivelul superior.

În calculatoarele microprogramate convenţionale (de exemplu seria Intel 80x86), fiecare instrucţiune adusă din memoria principală (MP) este interpretată de un microprogram stocat într-o singura memorie de control. În unele maşini însă, microinstrucţiunile nu emit direct semnalele ce controlează resursele hardware. Semnalele de ieşire sunt folosite pentru a accesa, printr-un nanosecvenţiator o memorie suplimentară de control numită memorie de nanocontrol. Nanoinstrucţiunile conţinute în această din urmă memorie controlează direct resursele hardware. Memoria ne microcontrol (μMC) conţine microprograme cu *microinstrucţiuni verticale*, interpretate de *nanoinstrucţiuni orizontale* ce se găsesc în memoria de nanocontrol (nMC). Mai multe microinstrucţiuni (ce fac parte din microprograme diferite) pot fi interpretate de acelaşi nanoprogram. Rezultă astfel o dimensiune mai mică, pe ansamblu, a memoriei de control. Un alt avantaj este flexibilitatea mai mare a operaţiei de proiectare a arhitecturii, ce rezultă din slăbirea legăturilor dintre instrucţiuni şi hardware, prin două niveluri intermediare de control, în loc de

unul. Dezavantajul principal al nanoprogramării este reducerea vitezei datorită accesului suplimentar la memoria nMC şi a structurii ceva mai complexe a organizării unităţii de control.

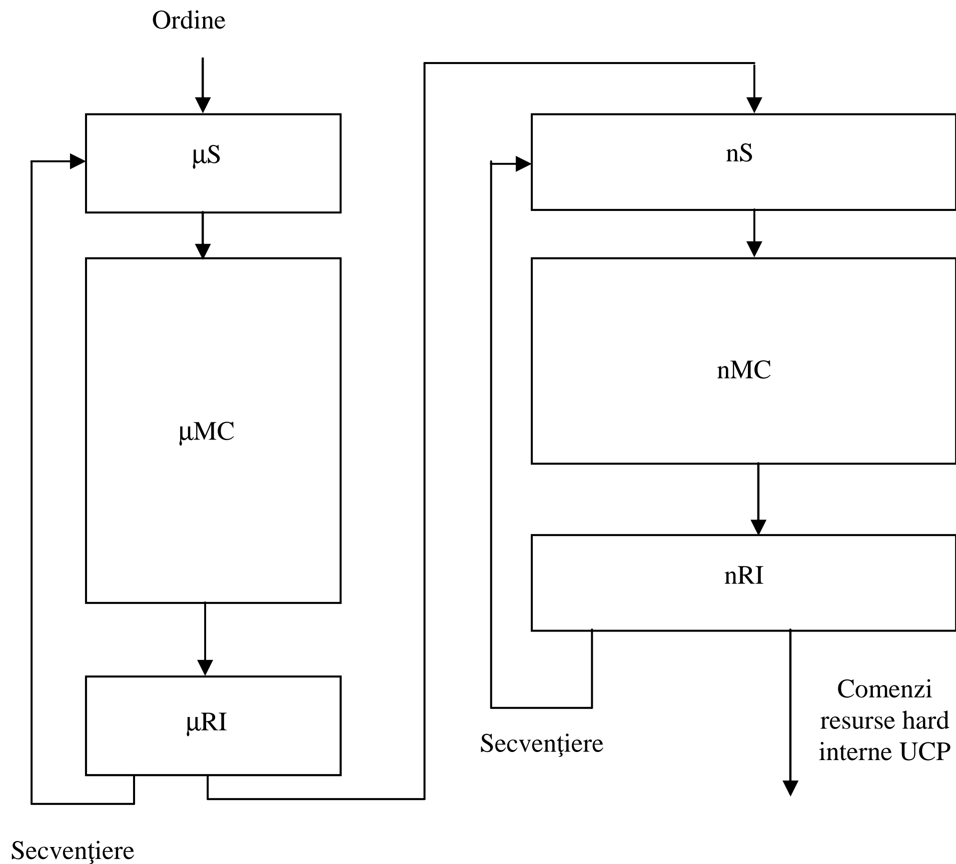


Figura 5.10. Organizarea memoriei de control pe două niveluri. Notaţii folosite: μS / nS = microsecvenţiator / nanosecvenţiator; μRI/nRI = micro / nano registru de instrucţiuni, μMC / nMC = memorie de microprogram / nanoprogram.

Posibilitatea optimizării mărimii totale a memoriei de control prin nanoprogramare a fost folosită la seria de microprocesoare Motorola 680x0. De exemplu MC 68000 conţine o memorie de microprogram de 640 x 10 biţi respectiv o memorie de nanocontrol de 280 x 70 biţi, adică în total circa 26 kbiţi. O proiectare microprogramată clasică ar necesita peste 45 kbiţi.

5.4. Paralelism în execuţia instrucţiunilor prin tehnici pipeline

Tehnica pipeline (pipeline = bandă de execuţie/ conductă de execuţie) este o tehnică de introducere a paralelismului în execuţia instrucţiunilor. Aceasta se face prin împărţirea ciclului de execuţie al unei instrucţiuni în mai multe faze de prelucrare, iar execuţia paralelă a mai multor instrucţiuni se bazează pe faptul că fiecare dintre aceste instrucţiuni se află în altă fază de prelucrare. Tehnica de tip pipeline poate fi aplicată atât pentru execuţia instrucţiunilor cât şi pentru circuitele aritmetice de prelucrare. Aici ne vom referi doar la execuţia paralelă a instrucţiunilor prin tehnica pipeline.

Se spune că prima conductă de execuţie a fost introdusă de Henry Ford prin linia de asamblare de la fabrica sa de automobile în anul 1913, pentru a creşte productivitatea. Să ne gândim la un alt exemplu de conductă ipotetică, din viaţa socială. Dacă vă duceţi la o bancă pentru a face operaţiuneaa de extragere din contul dumneavoastră curent, într-o situaţie neoptimizată (non-pipeline), se fac următoarele operaţiuni secvenţiale (presupunem că fiecare operaţie durează un minut):

1. Mergeţi la ghişeu, solicitaţi formularul de extragere şi îl completaţi
2. Funcţionarul băncii vă legitimează şi preia formularul completat şi semnat
3. Funcţionarul verifică dacă aveţi bani în cont, apoi tipăreşte un extras cu aprobarea sumei de retras pe care trebuie să-l semnaţi
4. În final primiţi şi număraţi mulţumit banii.

Dacă însă banca organizează această secvenţă ca o conductă de execuţie, cu 4 funcţionari, pentru fiecare din operaţiile ipotetice de mai sus, în acelaşi timp, vor fi patru clienţi în curs de servire, fiecare la alt ghişeu, iar pentru un număr foarte mare de clienţi timpul de servire din punctul de vedere al băncii, se micşorează, tinzând să se servească un client pe minut. Este adevărat că dumneavoastră faceţi un efort suplimentar pentru deplasare de la un ghişeu la celălat, iar timpul pierdut la bancă este aproximativ acelaşi. În mod asemănător se organizează conductele de instrucţiuni din microprocesor, pentru a realiza mai multe operaţii în paralel pe date / instrucţiuni aflate în faze diferite de execuţie. Avantajul maşinii, faţă de funcţionarii băncii, este că fiecare etaj al conductei nu stă de vorbă, nu fumează, nu bea cafea, nu butonează telefonul etc.

Pentru o conductă cu doar 4 etaje, creşterea ideală în viteză poate fi înţeleasă şi din reprezentarea gafică din figura 5.11.

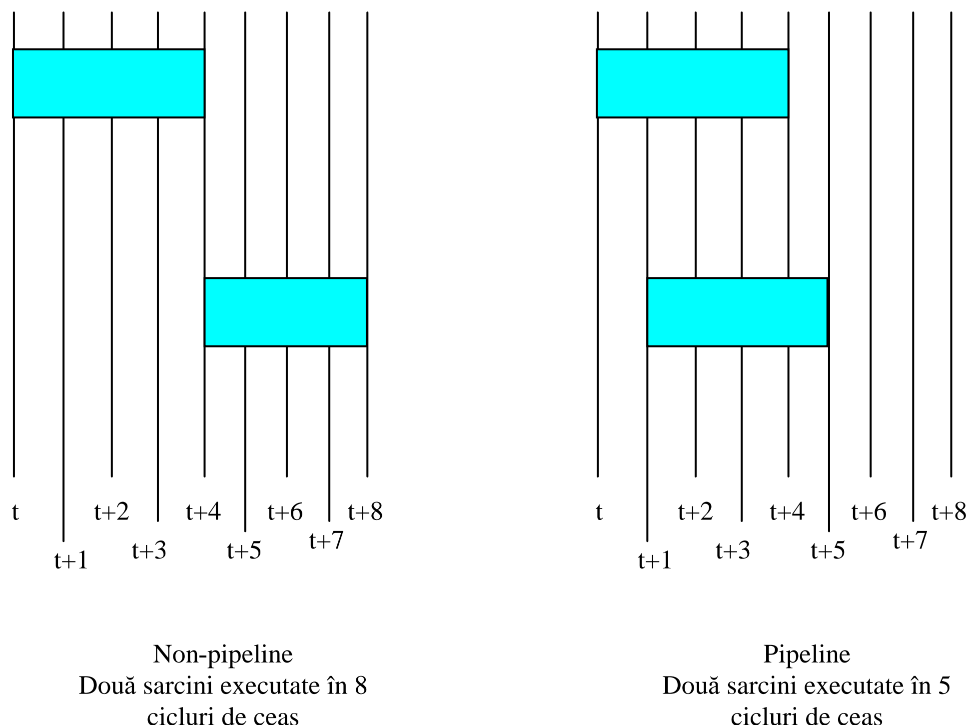


Figura 5.11. Prin conducta de execuţie (dreapta) se creşte viteza de execuţie pentru un set de sarcini de lucru aplicate la intrarea conductei.

Vom folosi în continuare termenul de conductă, care este sugestiv şi de asemenea scurt, pentru a exprima cuvântul din limba engleză “pipeline”. O conductă este similară unei linii de asamblare / montaj din industrie. Fiecare pas din conductă completează o parte din operaţiile necesare în execuţia instrucţiunii. La fel ca într-o linie de asamblare pentru autovehicule, munca care trebuie făcută într-o instrucţiune este spartă în părţi mai mici, fiecare din aceste părţi necesitând o fracţiune din timpul necesar pentru a termina complet execuţia instrucţiunii. Fiecare din aceşti paşi este numit *un etaj* al conductei sau *segment* al conductei. Etajele sunt conectate unul după celălalt pentru a realiza o conductă (bandă) de instrucţiuni ce intră la un capăt şi ies prelucrate la celălalt capăt. Pentru că toate segmentele conductei sunt legate între ele, toate segmentele trebuie să fie gata pentru a avansa (pentru a furniza informaţia către următorul etaj) în acelaşi timp.

Timpul necesar pentru deplasarea unei instrucţiuni cu un pas în interiorul conductei constituie un ***ciclu maşină al conductei***. Lungimea acestui ciclu este determinată de timpul cerut pentru prelucrarea în cel mai lent segment din conductă, pentru că toate segmentele avansează în acelaşi moment. Un ciclu maşină al conductei este egal cu unul sau mai multe cicluri de ceas, dar obiectivul oricărei maşini ce implementează această tehnică este să se ajungă la o medie de o instrucţiune executată pe ciclu de ceas. Tehnica de tip pipeline conduce la o reducere a timpului de execuţie mediu pentru o instrucţiune.

Cea mai simplă cale de a înţelege principiul conductei este să imaginăm fiecare etaj ca fiind format dintr-un registru urmat de un circuit combinaţional de prelucrare. (vezi figura 5.12 cu reprezentarea bloc a unei conducte cu 4 etaje / faze).
Registrele stochează datele de intrare pentru combinaţionale pe durata necesară ca sub-operaţia corespunzătoare segmentului să fie terminată şi rezultatele să apară corect la ieşirea circuitului combinaţional. Aceste ieşiri ale etajului se aplică la intrările unui registru dintr-un alt etaj similar şi aşa mai departe. Dacă încărcarea registrelor cu datele de la etajul anterior se face sincron cu un impuls de ceas, perioada ceasului trebuie să fie suficient de mare astfel ca cel mai lent dintre etaje să termine prelucrarea.

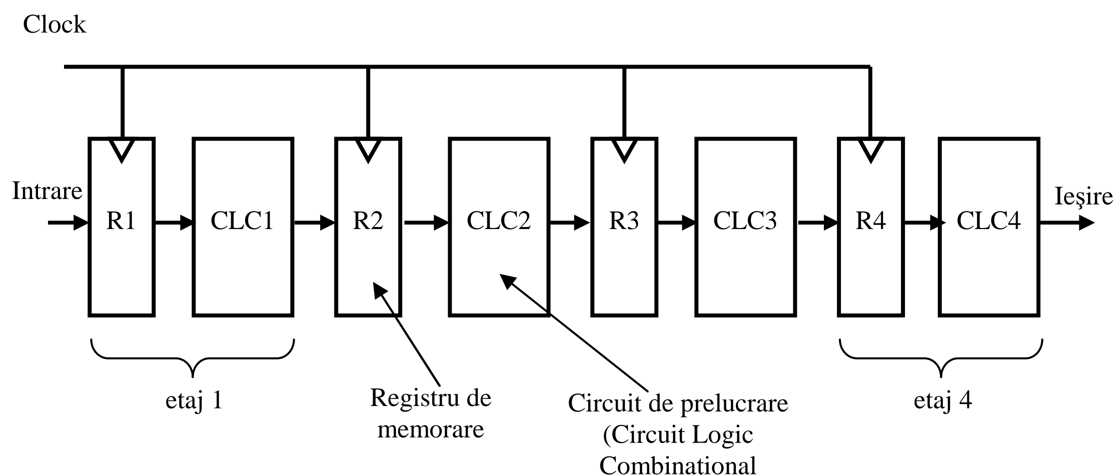


Figura 5.12. Structură de tip conductă (pipeline) cu patru etaje.

Revenind la conducta pentru instrucţiuni, vom presupune că este formată din patru etaje, a căror notaţie şi funcţie este:

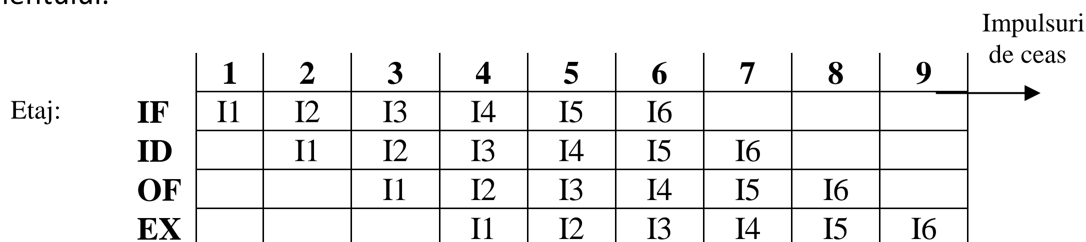
Etaj 1 IF (Instruction Fetch) - aducere instrucţiune din memoria principală

Etaj 2 ID (Instruction Decode) - decodificare instrucţiune

Etaj 3 OF (Operand Fetch) - aducere operanzi

Etaj 4 EX (Execution) - execuţie operaţie.

Funcţionarea unei conducte este explicată adeseori cu ajutorul unei diagrame timp-spaţiu cum este cea din figura 5.13, corespunzătoare unei conducte cu 4 etaje cu notaţiile propuse mai sus. Axa orizontală reprezintă timpul în multiplii de impulsuri de ceas iar axa verticală dă numărul segmentului.



Etaj:	1	2	3	4	5	6	7	8	9
IF	I1	I2	I3	I4	I5	I6			
ID		I1	I2	I3	I4	I5	I6		
OF			I1	I2	I3	I4	I5	I6	
EX				I1	I2	I3	I4	I5	I6

Figura 5.13 Diagrama timp-spaţiu pentru o conductă cu 4 etaje (IF, ID, OF, EX) corespunzător unui ciclu instrucţiune. S-a presupus introducerea în conducta de execuţie a 6 instrucţiuni, notate de la I1 la I6.

Dacă se consideră o conductă cu k etaje şi n instrucţiuni de executat, vom încerca să deducem creşterea în viteză pentru structura de tip pipeline. Dacă timpul de ciclu al conductei este notat cu T_{CLK} va fi nevoie de un timp egal cu $k \times T_{CLK}$ ca să umplem conducta (sau altfel spus ca prima instrucţiune să treacă prin toate etajele) şi de timpul suplimentar de execuţie a celorlalte n-1 instrucţiuni, adică $(n-1) \times T_{CLK}$. Timpul total pentru execuţia celor n instrucţiuni într-o conductă cu k etaje va fi:

$$T_P = (k + n - 1) \times T_{CLK} \tag{5.3}$$

Considerăm de asemenea cazul unei unităţi de execuţie a instrucţiunilor fără folosirea tehnicii pipeline. Fie un timp notat T_{mediu} necesar pentru execuţia fiecărei instrucţiuni.

Pentru un program cu n instrucţiuni, fără pipeline (suprapunere) timpul mediu total de execuţie va fi egal cu:

$$T = n \times T_{mediu} \tag{5.4}$$

Creşterea în viteză datorită folosirii conductei poate fi calculată ca raport între cele două valori de timp:

$$S = \frac{n \cdot T_{mediu}}{(k + n - 1) \cdot T_{CLK}} \tag{5.5}$$

Pe măsură ce numărul instrucţiunilor creşte este valabilă relaţia $n >> k - 1$, iar creşterea în viteză tinde către o valoare maximă egală cu:

$$S = \frac{T_{mediu}}{T_{CLK}} \tag{5.6}$$

Dacă presupunem că timpul mediu de execuţie al unei instrucţiuni (T_{mediu}) este acelaşi cu timpul mediu de execuţie al unei instrucţiuni în unitatea construită ca pipeline (adică $k \cdot T_{CLK}$), creşterea în viteză tinde către k, numărul de etaje al conductei. Aceasta este doar o limită teoretică, pentru că pe măsura creşterii numărului de etaje diferenţele între T_{mediu} şi $k \cdot T_{CLK}$ se măresc. Mai mult, cu cât creşte numărul de etaje cu atât mai greu este să echilibreze timpii de execuţie pe fiecare etaj.

În plus apar probleme de care nu s-a ţinut cont în relaţia calculului de creştere în viteză: salturile în program şi dependenţa de date şi resurse hardware inter-instrucţiuni. Aceste din urmă elemente creează ceea ce se numeşte hazardul structurii de tip conductă.

În general există trei probleme mari care pot face ca o conductă de instrucţiuni să nu funcţioneze normal:

1. *Conflicte de resurse* produse de accesul la aceleaşi resurse din două etaje diferite ale conductei. De exemplu accesul simultan al două etaje la memoria principală, unul din

accese pentru aducerea codului unei instrucţiuni, iar celălalt pentru accesarea operanzilor. Aceste conflicte se rezolvă prin mărirea numărului de unităţi funcţionale (de exemplu ALU multiple) şi prin utilizarea de spaţii separate de memorie pentru date şi instrucţiuni. (hazard structural)

2. *Dependenţa datelor* se referă la instrucţiuni care folosesc ca operanzi rezultate ale instrucţiunilor anterioare din conductă, dar acestea încă nu au fost executate, iar rezultatul nu este disponibil. (hazard de date). Hazardul de date se înlătură fie prin metode hardware (bypass) fie prin reordonarea instrucţiunilor din program.
3. *Ramificările* produc probleme pentru că dacă se face saltul stiva trebuie golită şi început procesul de execuţie al instrucţiunilor de la adresa de salt (hazard de control). Problemele hazardului de control se rezolvă prin predicţia salturilor, sau prin instrucţiuni de salt întârziat, la care se execută întotdeauna câteva dintre instrucţiunile ce urmează după instrucţiunea de salt condiţionat, în aşa fel conducta să rămână plină până la execuţia instrucţiunii de salt şi aflarea adresei următoare de instrucţiune.

Pentru exemplificarea tipurilor de conflicte (hazard) ale tehnicii pipeline vom considera o conductă (pipeline) de execuţie a instrucţiunilor cu 5 etaje. Etajele sunt numite [Stallings00]:

- FI = Fetch Instruction (aducere instrucţiune)
- DI = Decode Instruction (decodifică instrucţiune)
- FO = Fetch operands (aduce operanzi)
- EX = Execută
- WO = Write Operand (scrie rezultat)

Hazard structural

Hazardul se produce ori de câte ori mai multe instrucţiuni solicită în acelaşi timp aceeaşi resursă hardware. În exemplul din figura 5.14 instrucţiunea ADD R3,X va încerca să acceseze memoria pentru a aduce operandul X şi accesul instrucţiunii i+2 este blocat ("stall") în acel ciclu de ceas pentru că există o singură magistrală de date şi instrucţiuni.

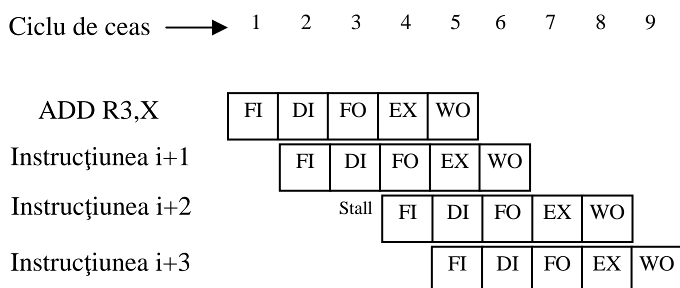


Figura 5.14. Exemplificarea hazardului structural (conflict de resurse)

Pentru înlăturarea hazardului descris (care se manifestă şi la celelalte instrucţiuni pentru faza FI simultană cu faza FO a instrucţiunilor anterioare) se introduc de obicei magistrale (interne) separate de date şi instrucţiuni şi memorii cache separate. În cazul conflictului introdus de alte resurse, acestea pot fi dublate (ALU, unităţi în virgulă mobilă etc.).

Hazard de date

De obicei se referă la încercarea unei instrucţiuni de a citi o dată (operand) înainte ca o instrucţiune anterioară să o producă ca rezultat. Hazardul de date este împărţit adesea în trei categorii, descrise pe scurt în continuare. Dintre acestea doar hazardul de tip *RAW* este hazard real, celelalte sunt considerate hazarduri artificiale pentru maşinile RISC.

- *Read After Write (RAW) – citire după scriere*

  Instrucţiunea i+1 încearcă să citească operandul înainte ca instrucţiunea i să-l scrie

```
i:      add r1,r2,r3        ; r1 <-- r2+r3
i+1:    sub r4,r1,r3
```

Hazardul este produs de "dependenţa" datelor. Soluţia obişnuită este întârzierea introducerii instrucţiunii i+1 în conductă prin execuţia altor instrucţiuni din program, deci prin schimbarea ordinii de execuţie din programul secvenţial iniţial (Out-of-order execution). O altă soluţie *ocolitoare* (forwarding sau by-pass) aduce rezultatul de la ieşirea ALU, conform operaţiei executate de instrucţiunea i direct la intrarea ALU cu ajutorul unui multiplexor, deci înainte de stocarea în r1. De multe ori ultima metodă nu elimină complet hazardul de tip RAW.

- *Write After Read (WAR)*

  Instrucţiunea i+1 scrie operandul înainte ca instrucţiunea i să-l citească

```
i:      sub r4,r1,r3
i+1:    add r1,r2,r3
i+2:    mul r6,r1,r7
```

Numit anti-dependenţă, hazardul rezultă din reutilizarea numelui registrului „r1".

- Write After Write (WAW)

Instrucţiunea i+1 scrie operandul înainte ca instrucţiunea i să-l scrie

```
i:      sub r1,r4,r3
i+1:    add r1,r2,r3
i+2:    mul r6,r1,r7
```

Numită dependenţă de ieşire, ea poate altera operanzii pentru instrucţiunile ce ar urma după secvenţa exemplificată. Hazardul rezultă de asemenea din cauza re-utilizării numelui "r1"

Hazard produs de salturi (Hazard de control)

Ori de câte ori în program apare o instrucţiune de salt condiţionat sau absolut curgerea continuă a datelor în conductă este blocată. La saltul absolut blocarea stivei din cauza hazardului de control este exemplificată în figura 5.15.

Pentru hazardul de control în cazul saltului condiţionat există două variante. Ele vor fi explicate pe un exemplu de secvenţă de program:

```
        SUB     R1,R2           ;R1 ← R1 - R2
        BEZ     TARGET          ;salt (branch) dacă rezultatul este zero
        Instrucţiunea  i+1
        ..................................
TARGET..................................
```

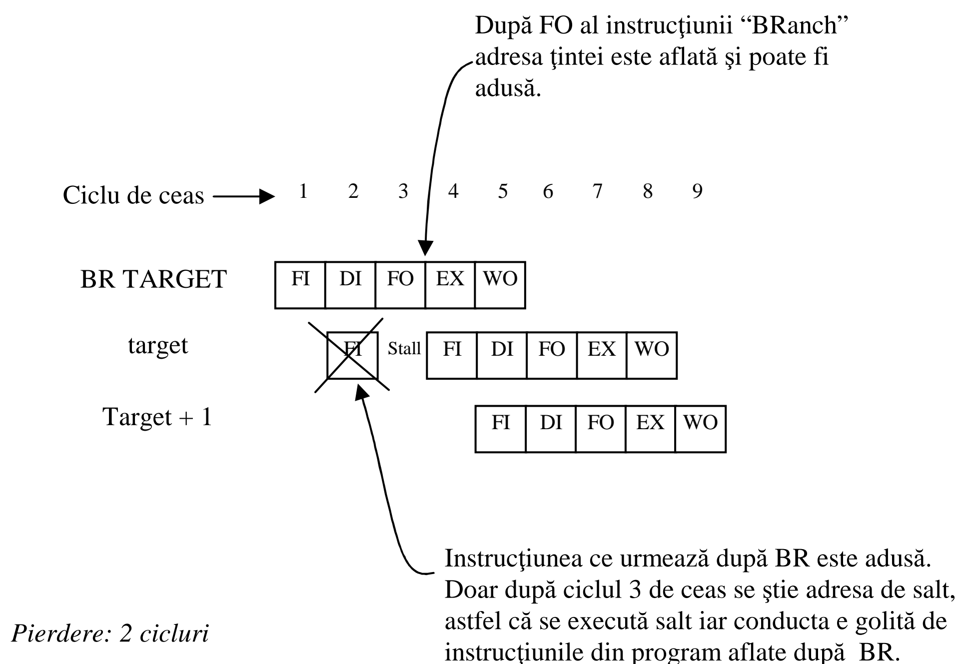


Figura 5.15 Exemplificare a hazardului de control în cazul unui salt necondiţionat.

În cazul salturilor condiţionate dacă se consideră că saltul se execută (în cazul testării unei condiţii) şi presupunerea nu este adevărată, toate instrucţiunile introduse în stivă se elimină şi se face re-umplerea stivei de la adresa corectă, imediat după instrucţiunea de salt (vezi figura 5.16)

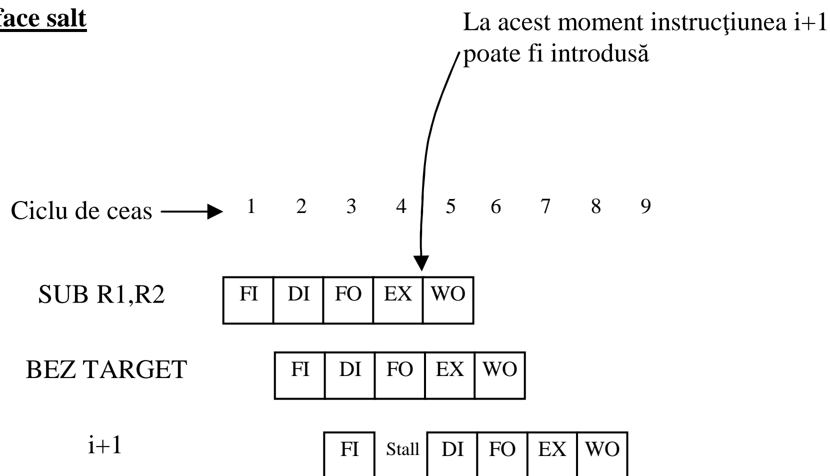


Figura 5.16. Exemplu de hazard la salt condiţionat, când saltul nu se produce

Şi în cazul complementar rezultatul este acelaşi – blocarea curgerii instrucţiunilor prin conductă pentru câteva cicluri de ceas (figura 4.17).

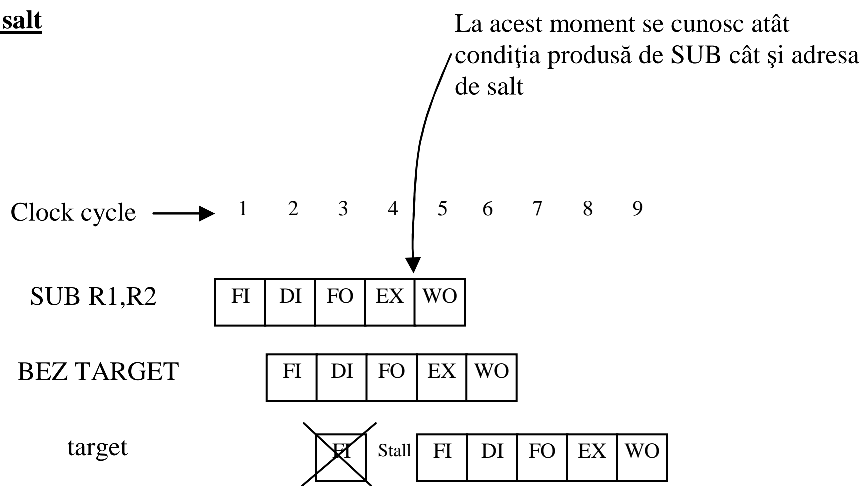
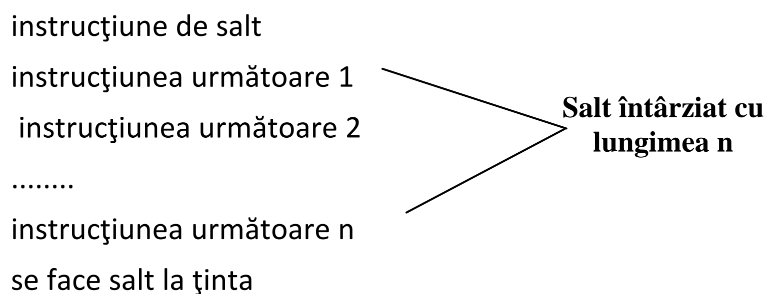


Figura 5.17. Exemplu de hazard la salt condiţionat, când saltul se efectuează

Există patru alternative privind decizia de luat în cazul hazardului de control:

1. Oprire până când adresa de salt (direcţia de ramificare) este clară
2. Se face predicţia că nu se va face salt
   - Execută succesorii instrucţiunii în secvenţă
   - Goleşte conducta dacă se face totuşi salt
   - Se aduce următoarea instrucţiune.
3. Se face predicţia că se va face salt
   - Dacă se cunoaşte adresa de salt totul este în regulă, dar dacă adresa trebuie calculată pot rezulta blocări.
4. Salt întârziat
   - Întotdeauna se execută una sau mai multe instrucţiuni successive instrucţiunii de salt. Chiar dacă rezultă condiţia de salt, saltul se face DUPĂ instrucţiunea n din exemplul următor (în general n are valoare între 1 şi 4).



5.5. Exerciţii

1. Care este rolul unităţii de control a UCP?
2. Descrieţi deosebirile dintre unitatea de control a UCP construită cablat şi unitatea de control construită microprogramat.
3. Explicaţi pe scurt funcţiile unităţii de control a microprocesorului: secvenţiere, interpretare
4. Demonstraţi creşterea în viteză a conductei de execuţie a instrucţiunilor
5. Ce semnificaţie are noţiunea de instrucţiune de salt întârziat
6. Descrieţi pe scurt metodele de înlăturare a hazardului structural
7. Descrieţi pe scurt metodele de înlăturare a hazardului de date
8. Descrieţi pe scurt metodele de înlăturare a hazardului de control

Capitolul 6

Sistemul de intrare - ieşire

Conţinut:

6.1. Circuite de interfaţă
6.2. Modalităţi de transfer de intrare-ieşire
6.2.1. Transferul de I/O prin program
6.2.2. Transferul I/O prin întreruperi
6.2.3. Transferul prin acces direct la memorie
6.3. Exerciţii

6.1. Circuite de interfaţă

Subsistemul de intrare-ieşire (I/O) al unui calculator asigură calea de comunicaţie a informaţiilor între calculator şi mediul extern (logica externă care nu lucrează direct cu UCP). Un sistem de calcul de uz general, nu are utilitate practică dacă nu poate recepţiona şi transmite informaţii cu exteriorul, într-o formă accesibilă utilizatorului uman. Prin intermediul acestui subsistem, utilizatorul introduce programele şi datele dorite în memoria calculatorului, pentru prelucrare, şi tot cu ajutorul său rezultatele se înregistrează, sau se afişează în exterior.

Legătura între UCP şi dispozitivele de I/O (dispozitivele periferice) se face prin intermediul unor *circuite de interfaţă de I/O* care asigură, din punct de vedere hardware, schimbul corect de date. Aceste circuite de interfaţă se cuplează la magistralele calculatorului şi ele sunt adresabile la nivel de *registru-port* de intrare-ieşire. Prin *port* înţelegem aici un loc (în general un registru), cu adresă specifică, adresă care constituie o "poartă" prin care calculatorul realizează schimb de informaţie cu exteriorul; fie culege informaţia, iar portul este port de intrare (PI), fie transmite informaţia la un port de ieşire (PO). Registrele port pot fi adresate în spaţiul de adrese al memoriei (vorbim atunci de *mapare-organizare a porturilor în spaţiul de memorie*) sau pot fi adresate în spaţiu separat de cel de memorie (*mapare în spaţiul de I/O*). Fiecare port de intrare sau ieşire are o adresă specifică.

Ca urmare adresele porturilor de intrare-ieşire pot fi tratate în două moduri:

1. *ca porturi cu adrese distincte faţă de adresele de memorie* (semnalele de control şi selecţie fiind specifice pentru memorie, respectiv pentru spaţiul de intrare-ieşire) - *mapare în spaţiu separat de I/O*;
2. *adresele porturilor sunt înglobate în spaţiul de adrese al memoriei principale*. Această organizare a adreselor (numita şi "*mapare a adreselor de I/O în spaţiul de memorie*") face ca porturile să fie selectate prin aceleaşi semnale de adresă şi control ca şi memoria. În lucrul cu porturile de I/O se vor utiliza instrucţiunile de transfer specifice memoriei.

Este important de observat că, exceptând procesorul şi memoria principală, toate circuitele conectate la magistrala de date a sistemului, deci care partajează această resursă a sistemului, sunt privite de procesorul central (UCP) ca porturi de I/O, chiar dacă funcţiile acestora nu se limitează strict la operaţii de intrare / ieşire. Este o manieră unitară de lucru a UCP cu orice circuit de interfaţă, circuit care poate include mai multe porturi de intrare / ieşire , indiferent de funcţiile specifice ale acestuia (figura 6.1). Aceasta face ca din punctul de vedere al UCP să se lucreze cu porturile adresabile într-un mod asemănător cu cel folosit la adresarea locaţiilor de memorie.

Unele circuite de interfaţă sunt incluse în circuitele specializate de control ("controlere") ale perifericelor. De exemplu, controlerul unităţii de disc, controlează operaţiile fizice efectuate

de discul magnetic. Pentru ca să se poată face cuplarea la calculator controlerul trebuie să respecte nişte specificaţii standardizate de interfaţare. Standardul IDE ("Integrated Drive Electronics") a realizat transferarea circuitelor de control către mecanismul discului hard, circuitele cuplate direct la magistralele calculatorului având o structură simplă. Pentru cazul specific al discului hard, standardul de interfaţă stabileşte nu numai modul de lucru cu partea electronică de control, ci şi modul în care se face codificarea datelor pe suportul magnetic. Funcţia principală a circuitelor interfaţă de I/O este de a rezolva diferenţele *funcţionale, electrice şi informaţionale* dintre calculator şi periferic. Circuitele controler au în plus şi sarcina specifică de control a unui anumit periferic. Principalele diferenţe între UCP şi periferie, care impun folosirea circuitelor de interfaţă, constau în următoarele:

a. perifericele sunt dispozitive a căror funcţionare se bazează pe diferite tehnologii (electromecanice, electromagnetice, electronice). De aceea trebuie să existe dispozitive de conversie a valorilor semnalului, pentru o adaptare din punct de vedere electric cu calculatorul;
b. ritmul de transfer al datelor este mult mai scăzut la periferice faţă de UCP. Pentru transferul de date între periferice şi UCP sau memorie trebuie deci să existe mecanisme de sincronizare.
c. codurile şi formatele datelor în echipamentele periferice pot fi diferite faţă de codurile şi formatele folosite în UCP şi memorie.
d. există o varietate de periferice, cu moduri de funcţionare diferite şi de aceea acestea trebuie controlate adecvat, pentru a nu perturba celelalte periferice conectate la UCP.

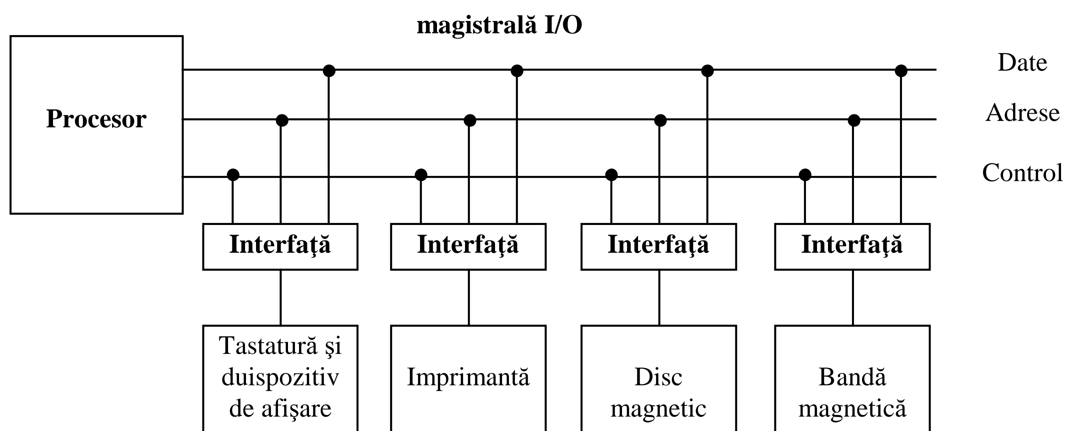


Figura 6.1. Conectarea la magistrala de I/O a dispozitivelor de intrare - ieşire.

Circuitul de interfaţă rezolvă toate aceste diferenţe, fiind inclus între UCP şi periferice, pentru a superviza şi sincroniza toate transferurile de intrare / ieşire. În plus aşa cum am amintit, fiecare periferic poate avea propriul controler care supervizeaza funcţionarea corectă, specifică a respectivului periferic. Interfaţarea a două dispozitive fizice (*hardware interfacing*) constă în a proiecta circuitele de interconectare fizică dintre aceste două dispozitive. Iată câteva exemple obisnuite de sarcini de "hardware interfacing":

- interfaţarea unei imprimante (printer) la un calculator (de exemplu, pentru PC, posibilă doar prin respectarea standardului de interfaţare USB[1] sau Centronix);
- interfaţarea unei legaturi de comunicaţie seriala la un calculator
- interfaţarea unui port de I/O digital la un calculator
- interfaţarea unor convertoare D/A sau A/D la un calculator
- interfaţarea unei unităţi de dischetă sau a unui controler de disc fix (hard disc) la un calculator.

Notiunea de interfaţare poate fi folosită însă şi dacă ne referim la programele rulate de calculator (*software interfacing*). Când cele două obiecte ce definesc interfaţa sunt programe de calculator, sarcinile de interfaţare constau în proiectarea unui alt program ce asigură comunicarea dintre primele două programe. De exemplu, un program poate fi un program de control al imprimantei (printer-driver), iar celălalt poate fi un program general de aplicaţie al utilizatorului. Programul de control al imprimantei este o procedură care tipăreşte un caracter la imprimantă ori de câte ori este apelat. Interfaţa software se referă la faptul ca programul de aplicaţie trebuie să cunoască locul în care trebuie să plaseze caracterul de tipărit, înainte de a invoca programul printer driver. După cum se vede din acest exemplu, interfaţarea software face ca parametrii să fie transmişi corect de la un program la celălalt. Protocolul de transmitere a parametrilor este stabilit doar de unul din cele doua programe ce comunică prin intermediul interfeţei. De exemplu la un PC, ce rulează sub sistemul de operare DOS, dintre cele doua programe ce se interfaţează, un program este, în general, un program de aplicaţie; celălalt program poate fi un program de control pentru un dispozitiv (“device driver”), de exemplu un printer-driver, sau poate realiza doar o anumită funcţie simplă (ca de exemplu alocarea memoriei). Al doilea program este cel care defineşte protocolul pentru transmiterea parametrilor. Distincţia dintre un hardware driver (program de control al unui dispozitiv fizic) şi o funcţie nu prezintă importanţă aici, elementul important este modul cum se transmit parametrii.

Circuitul de interfaţă fiind legat între magistrala de I/O a sistemului şi echipamentul periferic, are semnale specifice de cuplare - interfaţare cu magistrala, respectiv cu echipamentul periferic. Dacă spre periferic tipurile de semnale şi modul de lucru cu acestea depind în mare

[1] USB = Universal Serial Bus

măsură de caracteristicile dispozitivului periferic, partea dinspre magistrală, face legătura cu memoria şi UCP şi cuprinde semnale pentru selecţia circuitelor de I/O, pentru controlul transferului datelor şi semnale de cerere de servicii către UCP.

Transferul datelor între un port de I/O şi UCP are loc, în principiu, asemănător cu transferul datelor între procesor şi memorie.

Există patru tipuri de semnale de comandă pe care le poate recepţiona/transmite o interfaţă:

1. semnale de control
2. semnale de stare
3. date de ieşire
4. date de intrare

Semnalele de control sunt transmise de UCP pentru a activa perifericul şi pentru a-l informa ce operaţie trebuie să efectueze. De exemplu, o unitate de bandă magnetică poate fi comandată să deplaseze banda înainte cu o înregistrare de date, să deruleze rapid banda la început, sau să pornească citirea unui bloc de înregistrări. Semnalele de control emise sunt specifice pentru fiecare tip de periferic.

Semnalele de stare sunt utilizate pentru a testa diferite condiţii ale interfeţei şi ale perifericului. De exemplu, calculatorul poate testa dacă un periferic este gata pentru un viitor transfer de date. În timpul transferului se pot produce erori care sunt detectate de interfaţă. Aceste erori sunt marcate prin setarea unor biţi dintr-un registru de stare al interfeţei, registru ce poate fi citit de procesor.

Semnalele de comandă pentru date de ieşire fac ca interfaţa să răspundă prin transferarea datelor de la magistrala de date către registrele sale interne. Consideraţi exemplul cu unitatea de bandă. Calculatorul porneşte derularea benzii prin semnale de control. Apoi procesorul monitorizează efectuarea comenzii trimise prin citirea informaţiilor de stare. Când banda a ajuns la poziţia corectă, procesorul transmite comenzile (adresă şi control) şi datele de ieşire, iar interfaţa transferă informaţia către registrele interne, iar apoi către controlerul unităţii de bandă, pentru stocare.

Comenzile pentru date de intrare sunt similare cu cele pentru ieşirea datelor, diferind doar sensul de circulaţie al informaţiei. Procesorul testează starea interfeţei pentru a verifica dacă datele cerute pot fi transferate către magistrala de date.

6.2. Modalităţi de transfer de intrare-ieşire

Din punctul de vedere al circuitelor care controlează transferul şi de asemenea al dispozitivului care iniţiază un transfer de I/O, modalităţile de transfer se pot clasifica în:

1. *transfer prin program* - transfer iniţiat şi controlat în totalitate de programul rulat de UCP;
2. *transfer prin întreruperi* - transferul este controlat de UCP ca răspuns la o cerere de întrerupere externă, care iniţiază transferul;
3. *transfer prin acces direct la memorie* (DMA - "Direct Memory Acces") - transferul este controlat de un circuit controler DMA (care preia controlul magistralelor sistemului), iar iniţierea transferului este făcută fie de o cerere de transfer de la un periferic, fie la iniţiativa programului rulat de UCP.

6.2.1. Transferul de I/O prin program

La acest tip de transfer, toate transferurile se iniţiază şi se controlează prin program. Transferul poate fi *direct*, caz în care procesorul citeşte un port de intrare (PI) sau scrie un port de ieşire (PO), fără nici o verificare prealabilă a stării perifericului corespunzător portului. Acest mod de transfer se foloseşte pentru sistemele simple de control numeric, dedicate unor aplicaţii fixe, în faza de proiectare şi testare a transferului (program plus hardware), sau în cazul rulării unor rutine de iniţializare a circuitelor programabile de interfaţă.

Al doilea mod de transfer prin program este transferul prin *interogare*. Intrarea în conversaţie cu un periferic se face sub controlul programului, de obicei într-o *buclă de interogare* (*polling* - scrutare a stării circuitelor periferice implicate în transfer). UCP interoghează dispozitivele periferice, după o anumită strategie stabilită prin program, dacă doresc sau nu schimb de date sau mesaje, sau dacă sunt active (gata pentru a primi informaţii). Pentru a determina dacă o operaţie de I/O este cerută, sau dacă poate avea loc, se pot folosi bistabile de condiţie (fanioane) locale, setate conform condiţiilor portului. Aceste fanioane se implementează fizic fie ca bistabile singulare, fie sunt incluse în registre de stare ale porturilor. De exemplu un program poate controla transferul de date de la un circuit de conversie analog-numerică. Procesorul declanşează conversia, iar apoi testează periodic o ieşire (un bit) de stare al convertorului. Când bitul indică terminarea conversiei, se încheie rularea buclei de aşteptare, preluându-se datele convertite.

Datorită simplităţii ei, tehnica de comunicare prin interogare programată este recomandată în sistemele în care timpul pierdut în rutina de interogare nu este critic pentru aplicaţia respectivă. Atunci când programul are de controlat mai multe periferice, pe baza unei liste a porturilor asociate perifericelor, procesorul testează pe rând starea perifericelor şi le

serveşte din punctul de vedere al transferului, dacă starea citită permite acest lucru. În acest fel perifericele sunt deservite în mod secvenţial şi implicit apar întârzieri la servirea acestora, mai ales dacă numărul de periferice controlate este mare. Dacă unele dintre periferice se consideră mai importante decât altele, adresa acestora se poate introduce de mai multe ori în lista de testare din bucla de interogare. Dezavantajul principal al modului de transfer programat este constituit de timpul pierdut de UCP pentru testarea stării perifericelor, chiar dacă acestea nu cer servicii la momentele de timp respective. Acest mod de transfer prezintă însă avantaje din punctul de vedere al costurilor (minim de echipament fizic suplimentar pentru transfer). Alt avantaj este că se cunoaşte exact momentul de timp când se testează sau când se face transfer de date cu un periferic; de aceea se spune că acest tip de transfer este *sincron cu programul*.

Ca exemplu al transferului prin program vom presupune mai întâi un transfer între memorie şi un port de ieşire, indicând, mai întâi printr-o listă secvenţa de comenzi folosite:

```
                        START
                        iniţializare POINTER adresă de memorie;
                        iniţializare CONTOR al lungimii transferului;
CITIRE_MEM:             UCP citeşte date din memorie
TEST_STARE:     citire stare port
                        dacă e gata de transfer sari la TRANSF
                        dacă nu, incrementează CONTOR_RATĂRI
                        dacă CONTOR_RATĂRI = W salt la EROARE
                        altfel, salt la TEST_STARE
TRANSF:                 UCP scrie data la port, şi actualizează CONTOR şi POINTER
                        dacă mai sunt date de transmis, salt la CITIRE_MEM
                        altfel salt la FINAL
EROARE:                 .....

FINAL:          .....

                        END
```

Pentru I8086, dacă la adresa simbolica "**port**" se găseşte informaţia de stare (bitul 0 indicând prin 1 logic, stare gata de transfer), iar la adresa **port+1** se vor transfera datele, fragmentul de program arată astfel:

```
start:    mov dx, port
          lea bx, buffer
          mov cx, count
          mov si, wait
```

```
test_st:    in al,dx
            test al,1
            jnz transf
            dec si
            jnz test_st
transf:     inc dx
            mov al,[bx]
            out dx,al
            dec dx
            inc bx
            dec cx
            jnz test_st
```

Dacă sunt mai multe porturi, (portul I având adresa registrului de stare "port1", iar bitul 0 al cuvântului de stare arată dacă portul e gata (1) sau nu (0) de transfer), bucla de interogare ar putea arata astfel:

```
test1:      in al,port1
            test al,1
            jz test2
            call transf1
test2:      in al,port2
            test al,1
            jz test3
            call transf2

            :::::::

testN:      in al,portN
            test al,1
            jz test1
            call transfN
```

6.2.2. Transferul I/O prin întreruperi

Aşa cum s-a arătat în capitolul 3 evenimentele externe procesorului, deci independente de programul rulat, pot produce *cereri de întrerupere.* Dacă acestea sunt acceptate, se produce *întreruperea* (suspendarea temporară a) programului rulat şi *saltul la o rutină specifică de tratare* a cererii de întrerupere. După execuţia rutinei de tratare se revine la execuţia programului întrerupt. Subsistemul de întreruperi al calculatorului nu este destinat special doar pentru operaţii de transfer de I/O, dar aici ne vom referi doar la acest aspect. Din punctul de vedere al transferurilor de I/O, avantajele sistemului de întreruperi sunt următoarele [Sztojanov87]:

- permite sincronizarea procesorului cu evenimente externe;
- eliberează procesorul de sarcina testării periodice a perifericelor, sarcină consumatoare de timp. Ca urmare transferul prin întreruperi prezintă o viteză de răspuns mai mare decât transferul prin program;
- posibilitatea de tratare ierarhizată a cererilor de întrerupere simultane sau succesive;

Cererile de întrerupere pot fi recunoscute doar la sfârşitul ciclului instrucţiune curent. În general, pentru efectuarea transferurilor de I/O se folosesc cererile de întrerupere mascabile.

Răspunsul UCP la acceptarea unei cereri de întrerupere mascabilă, consta în următoarele operaţii succesive:

1. Salvarea stării programului întrerupt. În această fază se salvează automat în stivă cel puţin adresa de revenire la programul întrerupt. Alte informaţii privind starea programului se pot salva prin instrucţiuni introduse la începutul rutinei de tratare, iar înainte de revenirea la programul întrerupt trebuie introduse instrucţiuni pentru restaurarea acestor informaţii.
2. Confirmarea acceptării şi identificarea întreruptorului. Confirmarea acceptării întreruperii se face prin transmiterea de UCP a unui semnal de acceptare, în urma căruia perifericul trimite, de obicei, un vector de întrerupere pentru identificare. Există şi sisteme de identificare care nu folosesc vector de întrerupere.
3. Calcularea adresei de început a rutinei de tratare a întreruperii. Pe baza informaţiei de identificare se calculează unde se va face saltul pentru ca cererea de întrerupere, pentru transferul de date, să fie servită.
4. La terminarea rutinei de tratare, se execută o instrucţiune specială, care comanda refacerea conţinutului contorului de program şi deci revenirea la programul întrerupt.

Generarea vectorului de întrerupere se face fie direct de către dispozitivul întreruptor (de fapt de către circuitul de interfaţă cu un anumit periferic), fie centralizat, pentru toate întreruperile mascabile, de către un dispozitiv special numit controler de întreruperi.

Arbitrarea întreruperilor multiple.

În cazul întreruperilor multiple, simultane sau succesive, subsistemul de întreruperi trebuie să realizeze un arbitraj al acestora pentru a stabili care dintre cererile de întrerupere multiple va fi servită la un moment dat. Sistemul de arbitrare alocă priorităţi cererilor de întrerupere, servind întotdeauna cererile cu prioritatea cea mai mare. Se permite de asemenea ca o rutină de tratare să poată fi la rândul ei întreruptă de o cerere cu prioritate mai mare. De obicei, pentru un procesor de uz general, priorităţile intrinseci sunt atribuite în următoarea ordine: întreruperile interne (excepţiile) cu prioritatea cea mai mare, apoi urmează întreruperile externe (hardware) nemascabile şi în final cele mascabile. Arbitrarea întreruperilor multiple se poate realiza prin mai multe metode:

Arbitrare
- controlată de UCP
  - prin software
  - prin hardware
- controlată de circuit controler de întreruperi
- controlată prin hardware, în lanţ de priorităţi

În cazul arbitrării *controlate de UCP prin hardware*, UCP conţine intern circuitele necesare realizării arbitrării. Circuitul de arbitrare primeşte mai multe intrări de cereri de întrerupere, acestea, sau combinaţii ale acestora, corespunzând la un anumit nivel prefixat de prioritate. Adresele rutinelor de tratare pot fi fixe (cum este de exemplu cazul microprocesorului I8085) sau variabile (de exemplu la microprocesorul MC68000).

De exemplu, la microprocesorul Intel 8085 există 5 intrări de cereri de întrerupere. Una dintre intrări este numită INT şi ea funcţionează pentru întreruperi vectorizate, deci pe bază de vector de întrerupere. Celelalte intrări nu au nevoie de vector de întrerupere, pentru că intern ele au alocată câte o prioritate fixă, iar saltul către rutina de tratare se face tot la o adresă fixă. În tabelul 6.1. se indică intrările de întrerupere la microprocesorul I8085, împreună cu ordinea de prioritate alocată (prioritatea 1 fiind maximă) şi adresele de salt.

Tabel 6.1. Întreruperi la I8085.

Prioritate fixă	Nume intrare întrerupere	Adresă de salt
1	TRAP	24h
2	RST 5.5	2Ch
3	RST 6.5	34h
4	RST 7.5	3Ch
5	RST 7.5	prin Vector

Arbitrarea *controlată de UCP prin software*, constă într-un suport hardware minimal, extern UCP, care să sprijine operaţiile de recepţie a cererilor de întrerupere şi respectiv de identificare şi validare sau nu a întreruptorului, în funcţie de prioritatea alocată. Circuitele au ca element principal un registru-port de intrare, în care se alocă câte un bit pentru fiecare dispozitiv întreruptor. Dacă acest bit este setat, dispozitivul a cerut cerere de întrerupere. Simultan cu setarea bitului din registru, semnalul de cerere de întrerupere se transmite şi către UCP. Prin citirea registrului port de intrare şi compararea cu situaţia anterioară (memorată într-un registru intern), UCP detectează dacă a apărut o cerere de întrerupere cu prioritate mai mare decât cea a programului curent. În cazul unei întreruperi cu prioritate mai mică se continuă programul curent (după eventuala înregistrare a cererilor în vederea servirii ulterioare). În cazul priorităţii mai mari, UCP începe execuţia subrutinei de servire asociată noii cereri, întrerupând astfel programul curent şi actualizând registrul de serviciu. După terminarea execuţiei subrutinei de servire se reia ultimul program întrerupt. De asemenea, după comutarea contextului, UCP trebuie să revalideze întreruperile pentru a permite unor eventuale cereri de întrerupere prioritare să fie luate în considerare. Cererile de întrerupere ale dispozitivelor individuale de I/O, pot fi activate sau dezactivate prin program, prin intermediul unor bistabili de mască, asamblaţi de obicei într-un registru de mascare. Astfel că la momente diferite de timp, prin aplicarea unor măşti diferite, programul poate modifica ordinea de (prioritate de) servire a cererilor de întrerupere.

Arbitrarea controlată de circuit controler de întreruperi, este cea mai des întâlnită la microcalculatoarele moderne. Controlerul de întreruperi programabil (PIC[2]) este un circuit specializat care preia, în principiu, toate sarcinile pe care le avea UCP la arbitrarea prin software: acceptă cereri de întrerupere de la mai multe surse, determină dacă există cel puţin o cerere de întrerupere cu prioritate superioară celei a programului curent, iar în caz afirmativ, generează către UCP o cerere de întrerupere. La primirea confirmării acceptării cererii de întrerupere, circuitul PIC generează pe magistrala de date vectorul de întrerupere asociat celei mai prioritare

[2] Programmable Interrupt Controller

cereri existente. UCP întrerupe programul în curs şi serveşte cererea prin mecanismul descris anterior. Circuitul controler, este numit şi "programabil", pentru că poate funcţiona în mai multe moduri de lucru programabile prin înscrierea de către UCP a codurilor corespunzătoare modurilor de lucru în registrele de control ale PIC. De obicei programarea modului de lucru se face printr-o secvenţă de iniţializare, executată de UCP, la pornirea calculatorului. Modul de lucru poate însă să fie modificat ulterior, tot prin program. Circuitul PIC estre văzut de UCP ca un circuit de interfaţă ce cuprinde mai multe porturi de I/O. Modul de alocare a priorităţilor poate fi:

1. fix
2. rotitor. Acest mod implementează “politeţea” în servirea cererilor de întrerupere. Iniţial priorităţile sunt ordonate în ordine descrescătoare, dar după servire, fiecare nivel de prioritate devine automat cel mai puţin prioritar.
3. cu mascare. În acest caz priorităţile pot fi modificate dinamic, masca invalidând temporar anumite niveluri de prioritate.

Arbitrarea realizată prin PIC asigură un timp mic de răspuns pentru întreruperile hardware şi o flexibilitate mare în alocarea sau modificarea priorităţilor. În plus, multe dintre circuitele de tip PIC pot fi cascadate, ceea ce permite extinderea uşoară a sistemului într-un calculator.

Metoda de *arbitrare descentralizată prin lanţ de priorităţi* a întreruperilor presupune existenţa în fiecare dispozitiv întreruptor a unei logici capabile să asigure protocolul electric de tratare a întreruperilor. Dispozitivele întreruptoare se conectează într-un "lanţ de priorităţi" (daisy chain), ca în figura 6.2.

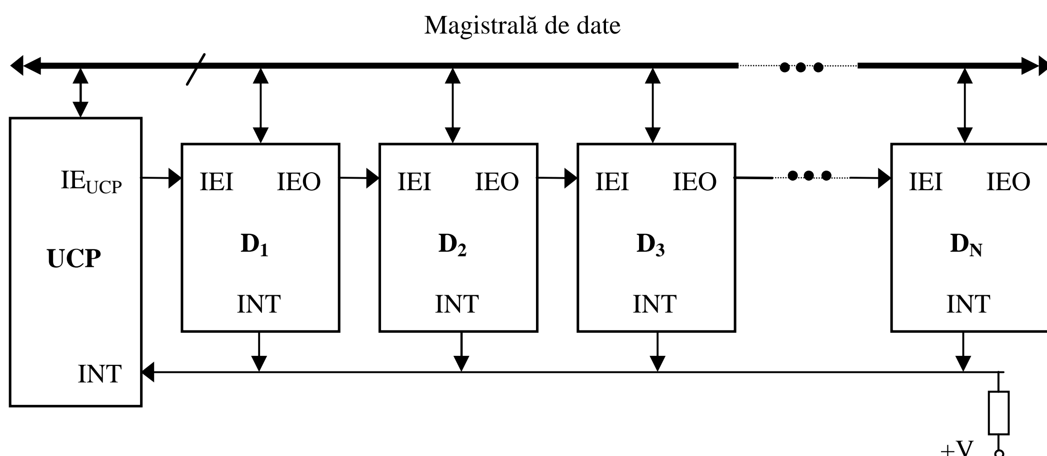


Figura 6.2. Exemplu de arbitrare a cererilor multiple de întrerupere prin lanţ de priorităţi

În figură fiecare dispozitiv D_i , i = 1,N dispune de câte o ieşire $\overline{INT}$ (cu colector / drenă în gol), ceea ce permite legarea împreună a acestor ieşiri la linia de intrare $\overline{INT}$ a UCP. Se realizează astfel un SAU cablat pe această intrare a UCP. (de fapt legând ieşirile împreună avem un SI cablat între ieşiri active în stare JOS, adică:

$$\overline{A} \cdot \overline{B} \cdot \overline{C} \cdots = \overline{A + B + C + \cdots} \quad (6.1)$$

Lanţul de dispozitive conduce la priorităţi fixe, ce ţin de poziţia dispozitivului în lanţ. Prioritatea cea mai mare este alocată celui mai apropiat dispozitiv faţă de UCP (D_1), iar prioritate minimă o are ultimul dispozitiv din lanţ.

UCP generează semnalul de validare a întreruperilor externe (IE_{UCP}) care se propagă prin lanţ, trecând prin celulele succesive ale lanţului. Pentru fiecare dispozitiv, starea intrării de validare IEI (Interrupt Enable Input) este copiată la ieşirea IEO (Interrupt Enable Output) a fiecărui dispozitiv, dacă dispozitivul respectiv nu cere întrerupere către UCP. Dacă un dispozitiv a cerut întrerupere, care a fost recunoscută şi se face tratarea acesteia, pe tot timpul servirii între intrarea IEI şi ieşirea IEO ale dispozitivului servit nu mai există egalitate. Pe toată această perioadă dispozitivul aduce IEO la zero pentru a nu permite dispozitivelor cu prioritate mai mică să-i întrerupă rutina de tratare. În momentul în care un dispozitiv D_i cere întrerupere, pot exista două situaţii:

- dacă IEI_i=0, atunci un dispozitiv mai prioritar decât D_i a cerut întrerupere şi servirea sa de către rutina corespunzătoare nu s-a încheiat. Ca urmare dispozitivul D_i va aştepta până când IEI_i trece în stare activă "1".
- dacă IEI_i=1, atunci D_i poate genera cererea de întrerupere către INT (aducând această linie la zero logic); aceasta pentru că D_i este dispozitivul cel mai prioritar care cere întrerupere la momentul respectiv de timp. Ca urmare, după lansarea cererii de întrerupere, IEO_i devine zero, invalidând cererile de întrerupere de la dispozitivele cu prioritate mai mică din lanţ.

Din cele de mai sus rezultă că dacă a apărut o cerere de întrerupere către UCP, în lanţ, un singur dispozitiv are IEI = 1 şi IEO = 0, şi anume cea servită. Pentru toate celelalte dispozitive ieşirea IEO are valoare logică identică cu intrarea IEI. Dacă UCP acceptă cererea de întrerupere, dispozitivul întreruptor depune pe magistrala de date a sistemului vectorul de întrerupere, pentru a fi identificat şi servit corespunzător.

6.2.3. Transferul prin acces direct la memorie

Transferul, prin intermediul UCP, a blocurilor mari de date între memoria principală şi periferice, se face relativ lent, pentru că de fiecare dată informaţia trece şi prin registre ale UCP. În plus pentru o cantitate mare de date transferate între memorie şi periferic UCP va consuma foarte mult timp.

Transferul prin acces *direct la memorie* (*DMA* - Direct Memory Acces) este executat direct între memoria principală şi dispozitivul periferic, fără ca datele să mai treacă prin UCP, sub comanda unui circuit controler DMA, care controlează temporar magistralele sistemului.

În figura 6.3. se presupune că iniţierea transferului se face de către periferic (prin intermediul interfeţei) care efectuează o cerere de transfer prin acces DMA spre circuitul controler de DMA (DMAC). Acesta solicită de la UCP controlul magistralelor prin semnalul de *cerere de control a magistralelor*, *BR*, (Bus Request). Cu o mică întârziere (la sfârşitul ciclului maşină curent) UCP cedează controlul magistralelor, îşi trece ieşirile către acestea în HiZ (stare de înaltă impedanţă) şi informează despre aceasta prin semnalul *acordare a controlului magistralelor*, *BG*, (Bus Grant).

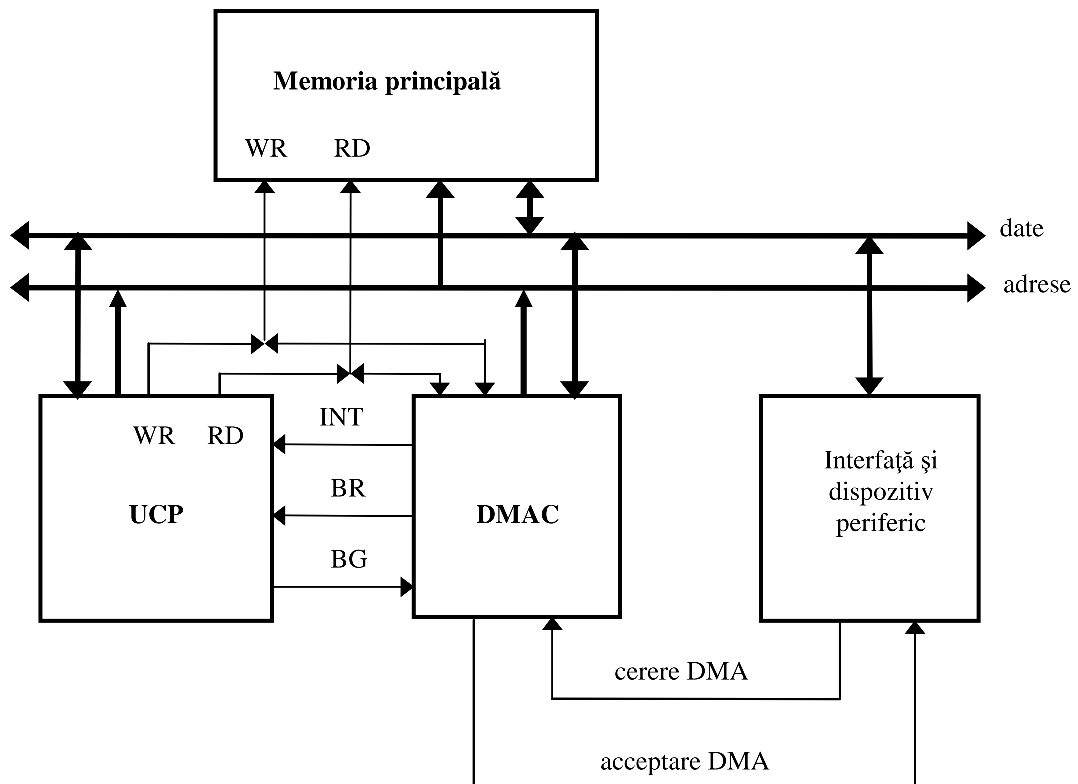


Figura 6.3. Exemplu de transfer DMA (DMAC = Controler DMA)

Circuitul controler DMAC furnizează adresele pe magistrala de adrese, preia controlul semnalelor de scriere (WR) şi citire (RD) şi trimite către periferic semnalul de acceptare a transferului prin DMA. Când dispozitivul I/O primeşte acest semnal de acceptare, el pune un cuvânt pe magistrala de date (pentru scriere în memoria principală) sau citeşte un cuvânt de pe magistrala de date (pentru citire din memoria principală).

Transferul de date se va face *direct* între periferic şi memoria principală, fără ca UCP să mai fie intermediar al acestui transfer (aşa cum se întâmpla la transferul prin program şi prin întreruperi). Un *ciclu DMA* reprezintă transferul unui cuvânt din sau în memorie. După terminarea unui ciclu, controlerul DMA poate continua cu alte cicluri DMA (dacă a fost programat să facă acest lucru şi dacă cererea DMA de la periferic se păstrează activă), sau poate returna controlul magistralelor către UCP prin inactivarea semnalului BR. Pentru transfer *pe cuvânt*, secvenţa de semnale de cerere DMA (cerere $\rightarrow$ BR $\rightarrow$ BG $\rightarrow$ acceptare) conduce la desfăşurarea unui singur ciclu DMA. Pentru un alt ciclu DMA, secvenţa de semnale trebuie repetată şi abia apoi se transferă un nou cuvânt. Cealaltă variantă de transfer, în *mod rafală-bloc*, presupune transferul mai multor cuvinte pentru fiecare cerere de transfer DMA. Este un mod de mare viteză, care însă dacă controlerul şi UCP sunt legate la aceleaşi magistrale poate menţine procesorul în inactivitate pentru o perioadă de timp. În cazul transferului pe cuvânt, întreruperea poate fi insesizabilă, ca timp, pentru procesor, şi de aceea uneori acest mod de transfer este numit "*cu furt de ciclu*".

De obicei circuitele controler DMA pot gestiona transferuri cu mai multe periferice simultan, pentru fiecare periferic existând (intern controlerului) un *canal DMA*.

Conform funcţiilor pomenite pe scurt mai sus, structura controlerului DMA trebuie să cuprindă:

- Logică de *comandă şi sincronizare*. Modul de lucru al controlerului DMA este programabil. De aceea el este iniţializat printr-o rutină de iniţializare rulată de UCP, fixându-se astfel: modul de transfer (cuvânt sau bloc-rafală), sensul transferului (la memoria principală sau de la memoria principală), nivelul activ al semnalelor de interfaţă cu echipamentul periferic, priorităţile acordate canalelor, modul de tratare al sfârşitului de transfer etc. Circuitul controler DMA are de asemenea o interfaţă cu UCP, care permite iniţializarea registrelor controlerului, comanda ulterioară de către UCP a controlerului, precum şi realizarea protocolului BR/BG;
- *Circuite tampon* pentru conectarea la magistralele calculatorului;
- *Logică de arbitrare a priorităţilor canalelor*;
- *Logică specifică fiecărui canal DMA*, care cuprinde cel puţin următoarele componente:
  - $\Rightarrow$ un registru *numărător de adrese* care generează adresa curentă (din memoria principală) de transfer;
  - $\Rightarrow$ un *numărător de cuvinte de transferat*, care se decrementează la fiecare ciclu DMA efectuat (valoarea iniţială a acestui contor este egală cu numărul de cuvinte ce se

doreşte a fi transferat. Atunci când conţinutul a ajuns la zero, ciclul de transfer s-a sfârşit);

⇒ un *registru de stare a canalului*, care poate fi citit de UCP şi care indică: sensul transferului prin canal, canal activat / dezactivat, prioritatea alocată etc.

Transferul prin DMA prezintă avantaje din punctul de vedere al vitezelor mari de transfer, pentru blocuri mari de date. Este un transfer folosit în aplicaţii de genul: transfer cu discurile magnetice, transfer cu plăci periferice ce conţin convertoare AD sau DA rapide etc.

6.3. Exerciţii

1. Explicaţi necesitatea introducerii circuitelor de interfaţă
2. Descrieţi pe scurt avantajele şi dezavantajele celor două moduri de mapare a adreselor de port (în spaţiul adreselor de memorie, sau în spaţiu separat)
3. Realizaţi în maximum 100 de cuvinte o analiză comparativă între transferurile prin program, prin întreruperi şi prin DMA (referire la inţierea şi controlul transferurilor, avantaje şi dezavantaje, viteză de transfer, costuri)
4. Descrieţi pe scurt structura internă a unui controler DMA

Capitolul 7

Sistemul de memorie

Conţinut:

7.1. Caracteristici principale ale dispozitivelor de memorie
7.2. Structura si funcţionarea memoriilor statice cu acces aleator (SRAM)
7.3. Structura şi funcţionarea memoriilor dinamice cu acces aleator (DRAM)
   7.3.1. Structura şi modul de lucru al dispozitivelor DRAM convenţionale
   7.3.2. Moduri de lucru de mare viteză ale DRAM
   7.3.3. Moduri de reîmprospătare
   7.3.4. Exemple de circuite DRAM cu interfaţă asincronă
   7.3.5. Circuite DRAM sincrone (SDRAM)
7.4. Memoria cache
7.5. Tehnici de adresare şi alocare a memoriei
7.6. Tehnici de translatare a adreselor
   7.6.1. Maparea adreselor folosind pagini
   7.6.2. Mapare prin segmentare
   7.6.3. Mapare segmentat-paginată
   7.6.4. Memoria virtuală
   7.6.5. Noţiuni privind protecţia memoriei
7.7. Exerciţii

7.1. Caracteristici principale ale dispozitivelor de memorie

Deşi există o mare varietate de dispozitive de memorie, toate acestea au la bază un număr relativ mic de fenomene fizice şi folosesc un număr relativ mic de principii de organizare. În continuare se vor descrie câteva caracteristici principale, comune atât pentru dispozitivele de memorie ce compun memoria principală cât şi pentru cele din componenţa memoriei auxiliare.

Capacitatea de stocare a memoriei (S) reprezintă numărul total de biţi ce pot fi stocaţi într-un dispozitiv sau circuit de memorie. Exprimarea parametrului S se face destul de rar în biţi, folosindu-se multiplii de octeţi (vom nota bit cu b iar Byte (octet) cu B):

1 octet (Byte) are 8 biţi
2^{10} biţi = 1 kilo-bit (kb)
2^{10} Bytes = 1 kilo-Byte (kB)
2^{10} kB = 2^{20} B = 1024 kB = 1 MB
2^{10} MB = 1024 Mb = 1 GB
1024 GB = 1 TB

Pentru capacitatea de stocare exprimarea în multiplii de biţi sau de octeţi nu ţine seama de modul de organizare internă şi adresare a cuvintelor dispozitivului de memorie.

Timpul de acces la dispozitivul de memorie exprimă un timp mediu necesar pentru a scrie sau a citi o cantitate fixă de informaţie din / în memorie (un cuvânt de memorie). Se calculează din momentul când dispozitivul de memorie primeşte o cerere de citire (scriere) până în momentul când informaţia este disponibilă la terminalele de ieşire (sau este înscrisă în locaţia de memorie adresată). Acest timp depinde de caracteristicile fizice ale mediului de stocare şi de tipul (modul) de acces folosit. Timpul de acces pentru memoriile semiconductoare moderne se exprimă în nanosecunde. Dintre tehnologiile semiconductoare de realizare, cele mai rapide sunt memoriile bipolare, dar pe de alta parte ele au un grad mai mic de integrare decât cele unipolare (CMOS sau celule dinamice) şi consumă putere mai mare de la sursa de alimentare.

Inversul timpului de acces (t_A) este numit viteză de acces (b_A) la dispozitivul de memorie:

$$b_A = \frac{1}{t_A} \tag{7.1}$$

Timpul mediu de acces este o caracteristică a dispozitivului de memorie indiferent de viteza cu care lucrează procesorul.

Modul de acces (sau tipul de acces) spune modul în care se accesează informaţia dintr-un anumit dispozitiv de memorie. Modul de acces depinde în primul rând de tehnologia de realizare

a dispozitivului de memorie şi de principiul constructiv adoptat. Accesul la celulele individuale de stocare se poate face fie într-o anumită ordine (*acces poziţional, sau serial*), fie aleator (*acces aleator*). Accesul aleator este independent de poziţia informaţiei în cadrul dispozitivului de memorare. La memoriile cu acces serial (bandă magnetică, CD-ROM) scrierea sau citirea locaţiilor de stocare se face în intervale diferite de timp, timpul de acces depinzând de poziţia în care se găseşte informaţia pe suportul fizic. Accesarea unei anumite locaţii presupune mişcarea mecanică a unui cap de citire / scriere şi / sau a suportului de stocare.

Memoriile cu acces aleator sunt memoriile semiconductoare, la care accesarea oricărei locaţii de memorie (de la oricare adresa) se face în acelaşi interval de timp. Scrierea sau citirea informaţiei se face direct la locaţia adresată fără parcurgerea unei secvenţe de adrese.

Timp de ciclu şi viteză de transfer a datelor. La memoriile cu citire distructivă (prin operaţia de citire informaţia stocată se pierde şi trebuie refăcută) şi la memoriile dinamice, nu este posibil să se înceapă un nou ciclu de acces la memorie până când nu s-a efectuat o restaurare a informaţiei sau o reîmprospătare a acesteia. Ca urmare, intervalul minim de timp ce se scurge între iniţierea a două accesări pentru transferul succesiv la / de la acelaşi dispozitiv de memorie, este mai mare decât timpul de acces; vom numi acest interval, *timp de ciclu (t_M) al dispozitivului de memorie*. Inversul acestui interval de timp exprimă cantitatea maximă de informaţie ce poate fi transferată la sau de la memorie în fiecare secundă:

$$b_M = \frac{1}{t_M} \quad \text{[cuvinte/sec]} \qquad (7.2)$$

unde b_M este numită *viteză de transfer a datelor*, sau *lărgime de bandă* ("bandwidth"). Un factor care limitează viteza de transfer a datelor este lăţimea magistralei de date pentru lucrul cu memoria, notată *w*. Valoarea lui *w*, indică numărul de biţi ce pot fi transferaţi în paralel pe magistrala de date / instrucţiuni. Această valoare, numită uneori cuvânt calculator, este specifică fiecărui procesor şi este, în general, un multiplu de octet. Ca urmare viteza de transfer poate fi exprimată şi ca:

$$b_M = \frac{w}{t_M} \quad \text{[biţi/sec]} \qquad (7.3)$$

Atunci când t_M este diferit de t_A se folosesc amindouă mărimile caracteristice pentru a măsura viteza memoriei. Din punctul de vedere al cuplării memoriei la UCP timpul de acces (t_A) este deosebit de important pentru că arată timpul cât procesorul trebuie să aştepte după iniţierea unui acces la memorie.

Timpul de ciclu al memoriei este definit ca timpul minim ce trebuie alocat între iniţierea a doua operaţii succesive cu acelaşi dispozitiv de memorie (scriere sau citire). Timpul de ciclu pentru un dispozitiv de memorie, poate să fie diferit, în funcţie de operaţiile de scriere sau citire. Pentru memoriile semiconductoare statice (bipolare sau CMOS) cele două intervale de timp sunt foarte

apropiate. Cel mai des însă $t_M > t_A$ pentru alte tipuri de memorie, cum ar fi cele semiconductoare de tip RAM dinamic (DRAM), din cauza naturii distructive a operaţiei de citire şi a necesităţii reîmprospătării ulterioare a informaţiei stocate. Desfăşurarea în timp a operaţiilor unui ciclu de lucru cu memoria poate fi descrisă prin figura 7.1.

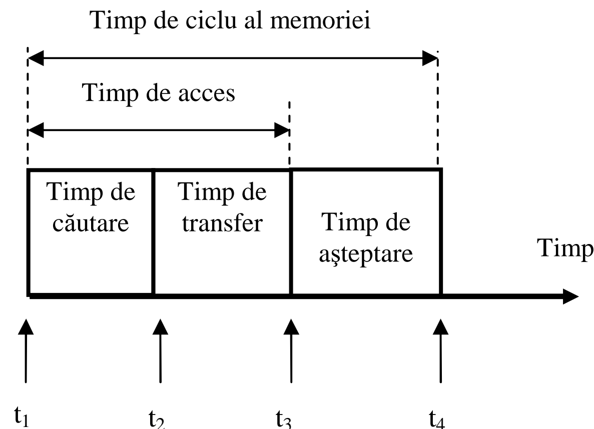


Figura 7.1. Explicativă la desfăşurarea în timp a operaţiilor de acces la un dispozitiv de memorie.

În figura 7.1. s-au notat momentele de timp semnificative ale unui ciclu de acces la memorie. Astfel la momentul t1 se face referirea la memorie (prin adresă şi semnale de control - selecţie). La momentul t2 locaţia adresată a fost găsită. La momentul t3 informaţia adresată este complet transferată (scriere sau citire), iar la momentul t4 este permisă o nouă referire la acelaşi dispozitiv (circuit) de memorie.

Costul dispozitivului de memorie nu cuprinde de obicei doar costul celulelor de stocare ci şi pe cel al echipamentelor periferice sau circuitelor de acces necesare în funcţionarea memoriei. Se exprimă de obicei global, pe niveluri de memorie, sau pentru întreaga memorie a unui calculator, în unităţi specifice, (cost/bit). Valoarea specifică se obţine prin împărţirea costului total al blocului, sau sistemului de memorie la capacitatea totală de stocare. Există o contradicţie între viteza memoriei şi costul pe bit: costul specific al memoriei scade pe măsură ce creşte timpul de acces (scade viteza de acces) la memorie şi, de fapt, această relaţie constituie fundamentul tehnico-economic al realizării memoriei sub formă de mai multe niveluri ierarhice.

Puterea consumată se exprimă de obicei ca putere consumată raportată la un bit de informaţie, obţinându-se prin împărţirea puterii totale consumate de memorie la capacitatea de stocare.

$$p = \frac{P_{tot}}{S} \quad \text{[W/bit]} \tag{7.4}$$

Geometria sau modul de organizare al memoriei este exprimată sub forma M $\times$ N, unde M reprezintă numărul de cuvinte posibil de memorat, iar N dimensiunea cuvântului, în biţi. Caracteristica numită geometrie este folosită doar pentru dispozitive de memorie din memoria principală. Este o caracteristică importantă din punctul de vedere al accesării informaţiei din memoria principală, pentru că indică cantitatea minimă de informaţie adresabilă. Dacă memoria este organizată pe octet, atunci N = 8 şi fiecărui octet i se alocă o adresă specifică, chiar dacă magistrala de date este de 32 sau de 64 de biţi.

Alterabilitate. Este caracteristica care se referă la posibilitatea modificării, de către procesor (on-line) a informaţiei înscrise într-un dispozitiv de memorie. La circuitele semiconductoare ROM, metoda utilizată pentru scrierea informaţiei în memorie este ireversibila; nu mai poate fi modificată în timpul funcţionării memoriei. La PROM-urile semiconductoare schimbarea, (PROM şi EPROM) schimbarea conţinutului se poate face off-line prin programarea memoriilor cu ajutorul unui circuit programator. Toate aceste circuite sunt incluse în categoria celor ne-alterabile. Dispozitivele de memorie alterabile sunt cele la care conţinutul poate fi modificat de către UCP în timpul rulării programelor. Ca exemple putem specifica dispozitivele RAM statice şi dinamice, discul magnetic, banda magnetică etc.

Permanenta stocării. Procesul fizic folosit pentru stocarea informaţiei este uneori instabil, astfel că informaţia s-ar pierde după un timp dacă nu se iau măsuri de restaurare. Există trei evenimente care pot duce la pierderea informaţiei stocate:

- citire distructivă
- stocare dinamică
- volatilitatea

Citirea distructivă se referă la faptul că dacă o celulă de memorie este citită, informaţia din celulă este alterată şi ea trebuie restaurată imediat după citire. Citirea distructivă se datorează în primul rând fenomenului fizic folosit pentru stocarea informaţiei şi apoi structurii hardware a celulei. Exemple de celule la care citirea este distructiva: celule dinamice cu un singur tranzistor unipolar de acces şi celule pe bază de ferită. Citirea trebuie urmată obligatoriu de operaţia de scriere, pentru restaurarea informaţiei. Operaţia este efectuată de către circuite specifice dispozitivului de memorie şi nu intră de obicei în sarcina UCP.

Stocarea dinamică a informaţiei se referă la stocarea în celule ce folosesc capacitoare MOS. Aceste capacitoare se descarcă în timp, şi dacă informaţia conţinută nu este reîmprospătată dinamic (condensatoarele re-încărcate), la intervale pre-stabilite informaţia se pierde.

Memoriile volatile sunt cele care pierd informaţia la pierderea tensiunii de alimentare.

Încadrăm aici toate memoriile RAM semiconductoare, indiferent că sunt dinamice sau statice. Ca exemple de memorii nevolatile: ROM, PROM, EPROM, EEPROM, disc, bandă etc.

7.2. Structura si funcţionarea memoriilor statice cu acces aleator (SRAM)

Memoriile cu acces aleator sunt caracterizate prin faptul că fiecare locaţie poate fi accesată independent, timpul de acces pentru oricare locaţie fiind acelaşi - independent de poziţia locaţiei. Fiecare celulă conţine linii de date (citire sau scriere - numite linii de bit la majoritatea celulelor bipolare sau unipolare) şi linii de selecţie.

În acest paragraf ne ocupăm de memoriile cu acces aleator de tip citeşte / scrie, notate cu sigla RAM = Random-Access Memory, la care celulele de memorie sunt bistabile ce păstrează informaţia pe toată durata alimentării cu energie.

Dispozitivele interne RAM necesare pentru decodificarea cuvântului de adresă, pentru comanda semnalelor de selecţie a celulelor, pentru circuitele de citire sau scriere a informaţiei şi circuitele de control intern sunt numite *circuite de acces* la unitatea de memorie.

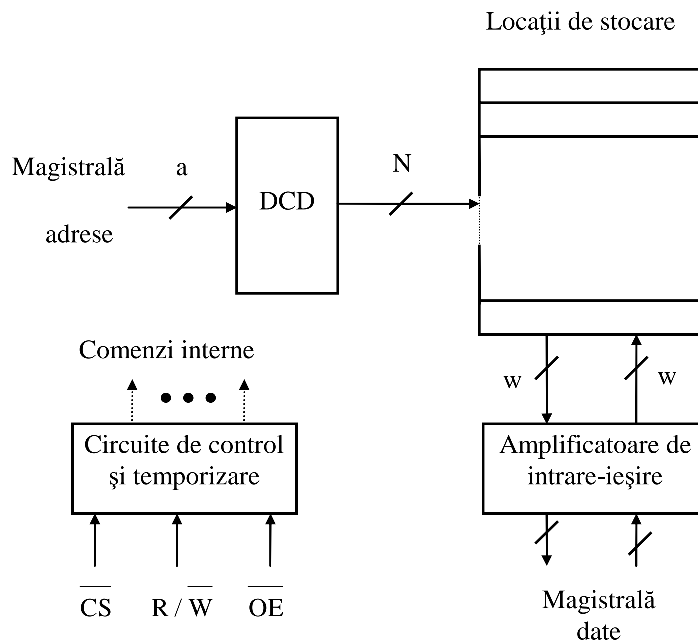


Figura 7.2. Schemă bloc pentru memorie RAM statică cu adresare uni-dimensională a locaţiilor de memorie

Modul de organizare al circuitelor de acces au un efect important asupra costului total al circuitului de memorie. Dacă un singur circuit decodificator (DCD) selectează fiecare locaţie de memorie (în care se găseşte o celulă de un bit sau un registru de w biţi) avem de-a face cu o schemă de adresare unidimensională (d=1) a locaţiilor de memorie. (figura 7.2).

Pentru fiecare adresă cu lungimea de *a* biţi la intrarea circuitului de memorie, decodificatorul activează una din cele N (= 2^a) linii de ieşire care selectează o locaţie (celulă sau cuvânt de w biţi) de memorie, pentru citire sau scriere. Semnalele de control din figură au semnificatiile următoare:

- $\overline{CS}$ = chip select - selecţie circuit. Semnalul este activ în nivel jos (0 logic)
- $R/\overline{W}$ = read/write, scrierea făcându-se pe nivel 0 logic, iar citirea pe nivel 1
- $\overline{OE}$ = output enable - validarea ieşirilor dispozitivului de memorie. Daca $\overline{OE}$ =0 atunci amplificatoarele de ieşire sunt deschise. Altefel amplificatoarele de ieşire ele sunt în stare de înaltă impedanţă (HiZ), pentru a permite cuplarea mai multor ieşiri la aceeaşi magistrală comună de date. La multe din circuitele integrate SRAM intrarea de control $\overline{OE}$ lipseşte funcţia sa fiind preluată de $\overline{CS}$ care pe lângă funcţia de selecţie a circuitului are şi sarcina controlului circuitelor de ieşire, eventual în combinaţie cu semnalul de citire / scriere.

De exemplu pentru o memorie cu organizarea 8K $\times$ 8 biţi cu o astfel de organizare a circuitelor de selecţie, a=13, N = 2^{13} , w = 8.

Întotdeauna este însă mai avantajoasă o organizare bidimensională (d=2) a selecvţiei circuitului de memorie, prin împărţirea câmpului de adrese în două părţi (a_X biţi pentru linii si a_Y linii pentru coloane) şi implicit împărţirea decodificatorului în două blocuri: decodificator de linii (decodificator X) şi decodificator de coloane (decodificator Y), ca în figura 7.3.

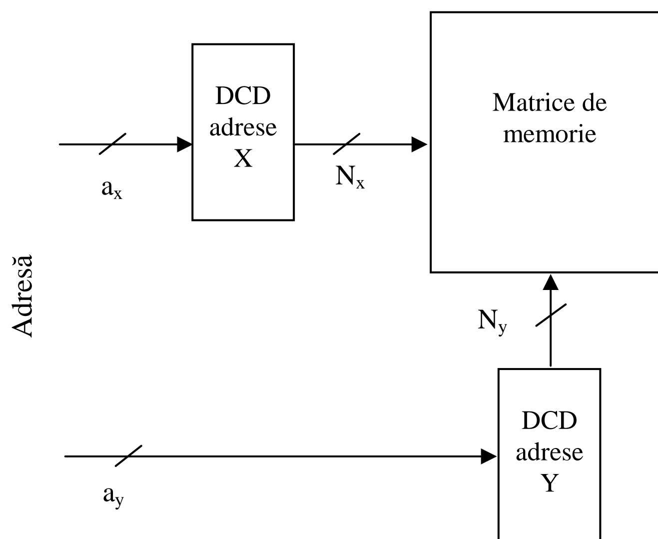


Figura 7.3. Schemă bloc care indică selecţia bidimensională a matricei de celule de memorie

Celulele de memorie din figura 7.3 sunt selectabile individual, prin adresa de linii şi adresa de coloane. Numărul total de celule este $N = N_x \times N_y$, unde de obicei $N_x = 2^{a_x}$ iar $N_y = 2^{a_y}$. Acest mod de selecţie necesită circuite de acces mult mai simple (de unde reyultă şi costuri mai reduse) decât în cazul d=1. De exemplu, pentru a compara cele două moduri de selecţie, dacă $N_x = N_y = \sqrt{N}$, numărul de dispozitive ce comandă direct liniile de selecţie ale matricei de memorie este $2\sqrt{N}$ în loc de N ieşiri ca la d=1.

Pentru *N>>4* diferenţa este semnificativă. În plus organizarea bidimensională este adecvată si cu structura bidimensională a suprafeţei chip-ului de siliciu folosit de tehnologia de realizare a circuitului integrat (CI). Aceste avantaje se diminuează, sau chiar dispar pentru dimensiuni d mai mari decât 2. De aceea organizări cu d>2 sunt rar utilizate.

Bineînţeles că în figura 7.3, celulele adresate pot fi înlocuite cu registre de w biţi selectate de aceleaşi semnale. Rezultă astfel un SRAM organizat pe cuvânt. Considerând acelaşi exemplu numeric ca şi la selecţia unidimensională (8K cuvinte de 8 biţi) structura de principiu a memoriei ar fi cea din figura 7.4, în care se observă introducerea a 8 matrice de memorie selectabile la nivel de celulă (bit) extinse în "adâncime". Aceleaşi decodificatoare fac selecţia pentru toate cele 8 planuri, dar sunt necesare 8 amplificatoare de citire / scriere (nefigurate).

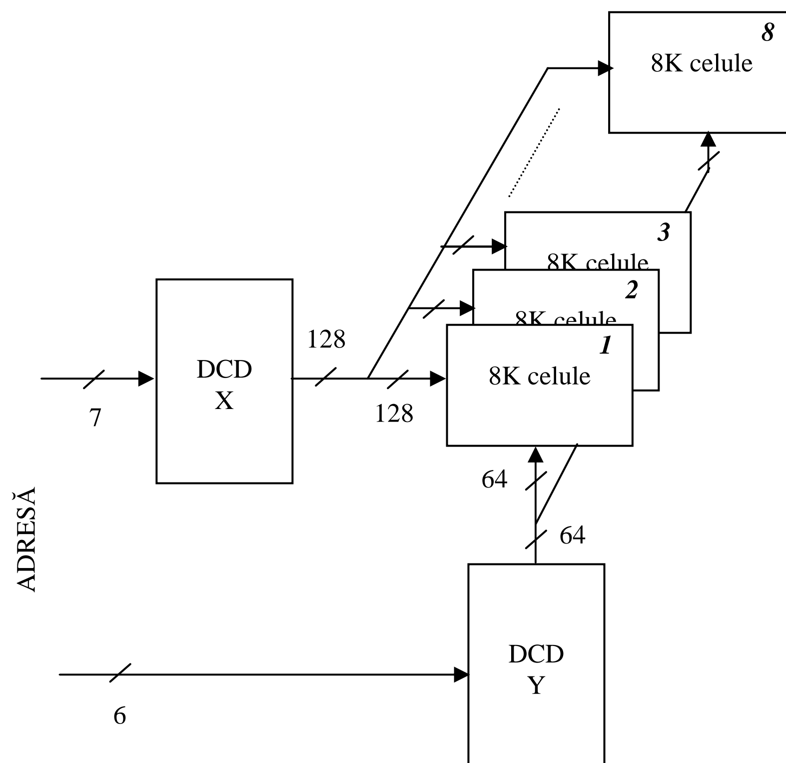


Figura 7.4. Schemă bloc de principiu pentru o memorie SRAM cu capacitatea 8k $\times$8 biţi şi selecţie bidimensională.

Structura bloc din figura 7.4. pune însă probleme de implementare, din cauza planurilor suprapuse de celule de memorie. De aceea se încearcă construirea unui astfel de circuit de memorie în aşa fel încât matricea de memorie să aibă aproximativ acelaşi număr de linii cu numărul de coloane (“matrice pătrată”), iar celulele să fire construite în acelaşi plan al circuitului integrat. Varianta corectă din punctul de vedere al implementării, pentru exemplul memoriei cu 8k cuvinte fiecare de câte 8 biţi se indică în figura 7.5.

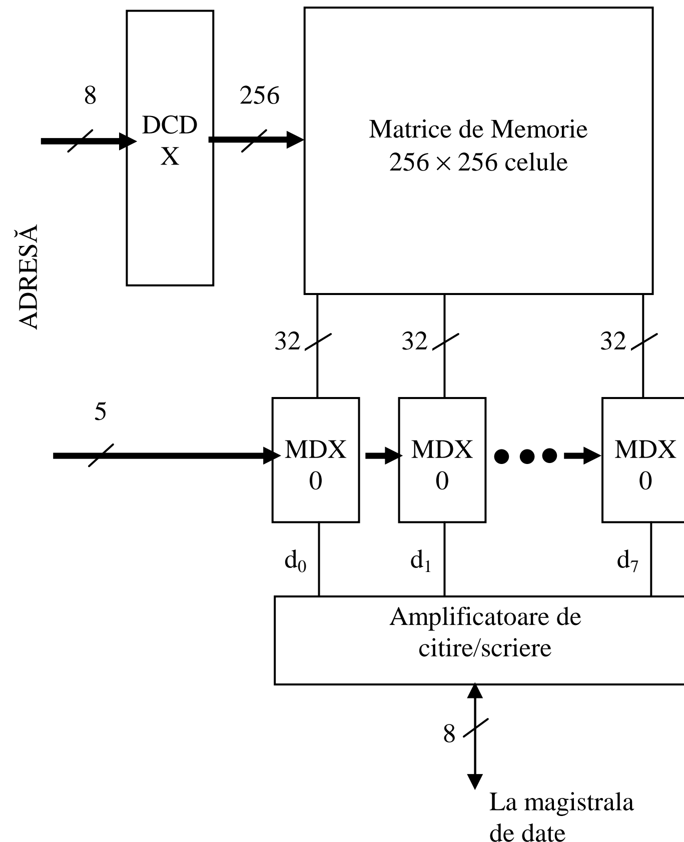


Figura 7.5. Exemplu de memorie RAM statică cu dimensiunea de 8k ×8 şi selecţie bidimensională a matricei de memorie.

Selecţia este bidimensională, iar în locul decodificatorului de coloane s-au introdus circuite multiplexor / demultiplexor (MDX). Structura unui asemenea circuit poate fi găsită în [Stefan93]. Circuitul MDX conţine un decodificator, iar la ieşirea sa se comandă porţi de transfer de tip MOS. Atunci când se face citire funcţionarea circuituitului este cea specifică demultiplexorului, iar când se face scriere funcţionarea este de multiplexor.

Conform caracteristicilor prezentate în paragraful anterior, fiecărui dispozitiv de memorie care lucrează cu procesorul i se impun anumite *condiţii şi restricţii de timp*. Acestea sunt descrise de obicei prin diagrame de semnale în timp, pe baza unor cicluri caracteristice de funcţionare ale dispozitivelor de memorie.

Pentru o memorie statică cu acces aleator ciclurile caracteristice de funcţionare pot fi doar cele de citire şi de scriere. În figura 7.6.a se prezintă un ciclu tipic de citire, iar în figura 7.6.b un ciclu tipic de scriere.

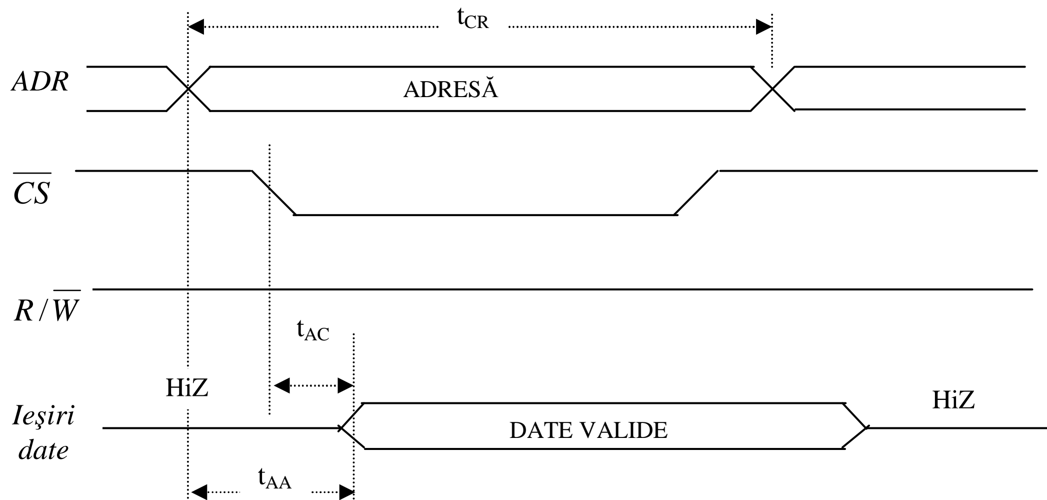


Figura 7.6.a. Exemplu de ciclu de citire la SRAM

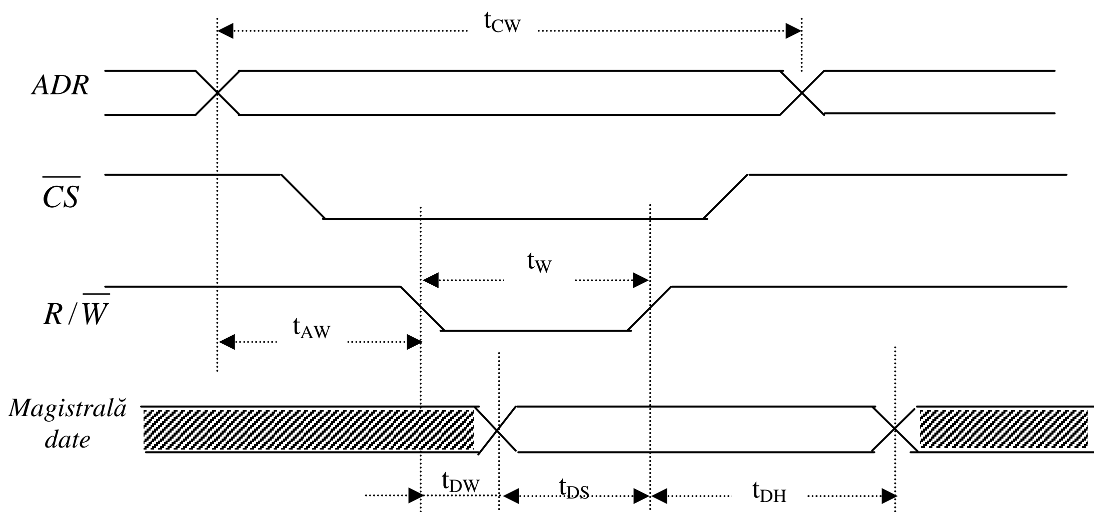


Figura 7.6.b. Exemplu de ciclu de scriere la SRAM

În cele două figuri s-au notat intervalele de timp importante, cu următoarea semnificaţie:

t_{CR} = durata ciclului de citire (se impune ca timp minim pentru funcţionarea corectă a memoriei)

t_{AA} = timp de acces de la activarea adreselor (se garantează ca o valoare maximă ce se

poate obţine)

t_{AC} = timp de acces de la activarea semnalului $\overline{CS}$ (se garantează ca o valoare maximă ce se poate obţine)

t_{HD} = timp de menţinere a datelor (valoare minimă impusă)

t_{CW} = durata ciclului de scriere (se impune ca timp minim pentru funcţionarea corectă a memoriei)

t_{W} = durata semnalului de comandă de scriere (write - este valoare minimă în catalog)

t_{DS} = timp de prestabilire date (valoare minimă impusă)

t_{DH} = timp de menţinere a datelor (valoare minimă impusă)

t_{DW} = întârzierea validării datelor la intrare

t_{AW} = întârziere între stabilirea adreselor şi semnal de control de scriere

Toate restricţiile de timp se indică în foile de catalog ale dispozitivelor de memorie şi ele trebuie privite ca limitări în viteza cu care se desfăşoară comunicaţia dintre UCP şi respectivul dispozitiv. În general valorile de catalog ale acestor timpi trebuiesc considerate ca valori minime, sub care nu este bine sa se scadă. Acestea limitează frecvenţa maximă a operaţiilor efectuate de UCP cu memoria. De asemenea în cazul valorilor timpilor de acces este bine să se considere valorile maxime date în catalog ca valori după care UCP este sigur că datele au fost citite (de pe magistrala de date) sau înscrise în memorie. De aici rezultă uneori necesitatea unor mecanisme de sincronizare (de tip READY) ale UCP cu dispozitivele de memorie.

În figura 7.6.a s-au reprezentat terminalele de ieşire ale dispozitivului de memorie, terminale ce se cuplează la magistrala de date. Ele se găsesc în stare de înaltă impedanţă până în momentul când s-a scurs timpul de acces şi atunci când dispozitivul de memorie este neselectat. Pentru ciclul de scriere din figura 7.6.b zona haşurată a magistralei de date reprezintă valori neimportante pentru ciclul de scriere. Porţiunea ne-haşurată reprezintă datele stabile puse de microprocesor pe magistrala de date, pentru a putea fi scrise corect în dispozitivul de memorie.

7.3. Structura şi funcţionarea memoriilor dinamice cu acces aleator (DRAM)

Acest tip de memorii se numesc "dinamice" datorită principiul de lucru al celulelor interne. În fiecare celulă se reţine informaţia sub formă de sarcină electrică într-un capacitor MOS[1]. Indiferent de tehnologia de realizare sarcina electrică dintr-un capacitor se descarcă lent. În cazul memoriei, pierderea sarcinii, înseamnă pierderea informaţiei - circuitul "uită" informaţia stocată. Pentru a preveni acest fenomen, din timp în timp, informaţia trebuie reîmprospătată, adică

[1] MOS = Metal Oxid Semiconductor, tehnologie de realizare a dispozitivelor semiconductoare unipolare

condensatoarele trebuie reîncărcate cu informaţia stocată anterior. Aceasta înseamnă că informaţia poate fi păstrată doar printr-o *funcţionare dinamică*, ce presupune încărcarea periodică a condensatoarelor interne.

Dispozitivele DRAM[2] sunt în prezent cele folosite dispozitive pentru construirea memoriei principale a calculatoarelor de uz general, pentru că ele prezintă cea mai mare densitate de integrare dintre tehnologiile adecvate memoriei principale şi totodată prezintă cel mai mic cost pe bit.

7.3.1. Structura şi modul de lucru al dispozitivelor DRAM convenţionale

Dispozitivele DRAM convenţionale au următoarele caracteristici structurale şi funcţionale:

- *Matricea* de memorie este *pătrată* ($m \times m$; iar $m = 2^n$)
- Se foloseşte o schemă de adresare bidimensională în care sarcinile de selecţie ale matricei de memorie sunt împărţite între un decodificator de linii şi unul de coloane.
- Se foloseşte *multiplexarea* biţilor pentru adresele de linie şi adresele de coloană. Adresa furnizată dispozitivului de memorie se face printr-un număr de linii egal cu jumătate din numărul total de biţi de adresă (de exemplu pentru 2^{22} celule = 4 Mbit cu matrice de 2048 x 2048 celule, numărul de pini de adresă este 11 şi nu 22). Principalul avantaj îl reprezintă un număr mai mic de pini la capsulă, deci un gabarit mai redus al circuitului. Gabaritul mai mic se referă nu doar la numărul de terminale ci şi la circuitele tampon - interfaţă cu exteriorul necesare pentru fiecare pin.
- au o ***interfaţă asincronă*** cu exteriorul (în particular cu procesorul) operaţiile fiind comandate, nu de un semnal de ceas ci, prin semnale la care se impun durate şi întârzieri minime între fronturi, în aşa fel încât operaţiile să se desfăşoare corect.

Structura bloc a unui circuit DRAM convenţional este prezentată în figura 7.7. Amplificatoarele senzor din figură sunt circuite care generează nivel logic 0 sau 1 pornind de la tensiunea de pe condensatorul - celulă selectat. De câte ori se selectează o linie, acelaşi circuit amplificator face şi reînscrierea tuturor condensatoarelor de pe linia respectivă.

Structura formată din decodificatorul (DCD) de coloane şi porţile de transfer de I/O funcţionează ca o structură de demultiplexare / multiplexare similară blocului MDX pomenit la memoriile SRAM.

Pe structura din figura 7.7. s-a presupus că circuitul de control al operaţiei de re-împrospătare se găseşte pe acelaşi circuit cu memoria dinamică, deşi la primele memorii DRAM aceste circuite de control erau externe.

[2] DRAM = Dynamic Random Access Memory

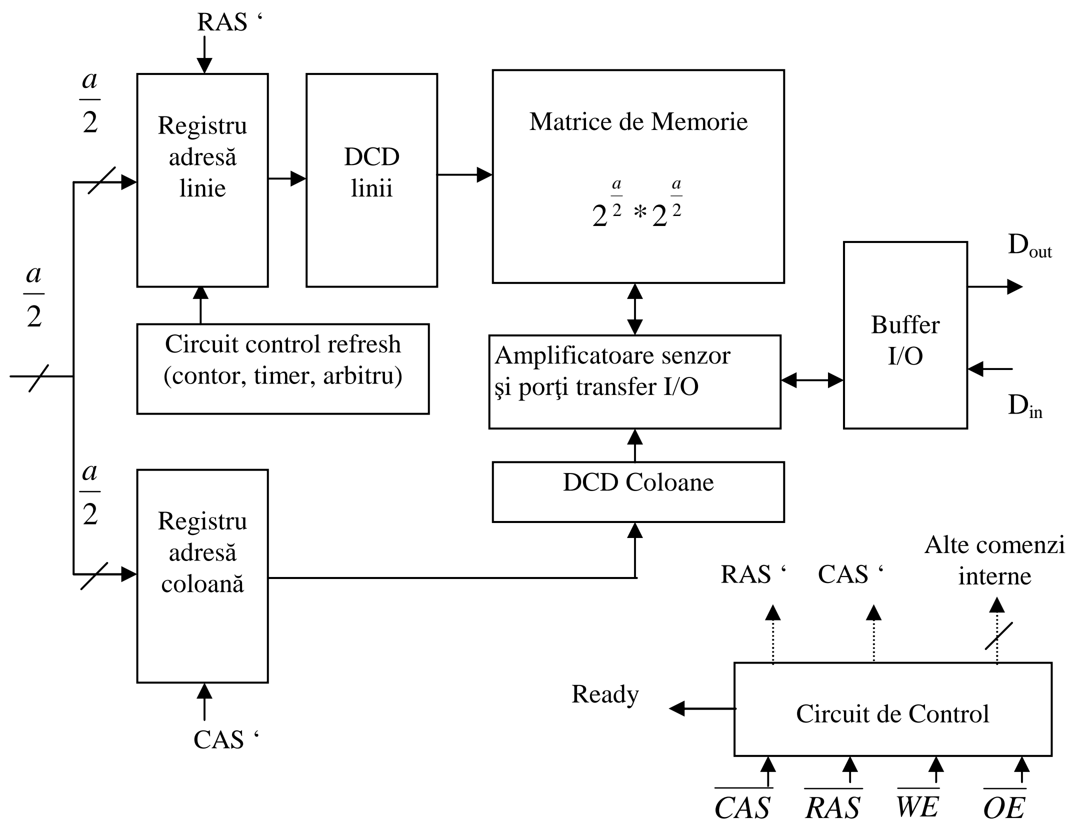


Figura 7.7. Structura bloc a unui circuit DRAM convenţional şi semnalele tipice de interfaţare cu procesorul

Atunci când se doreşte accesul la o locaţie de memorie, adresele se introduc în ordinea:

1. *adresa de linie*, validată de semnalul $\overline{RAS}$ (row address strobe), care produce memorarea acestei adrese în registrul de adresă linie, intern circuitului de memorie, iar apoi
2. *adresa de coloană*, validată de semnalul $\overline{CAS}$ (column address strobe), care produce memorarea acestei adrese în registrul de adrese coloane. Semnalul $\overline{CAS}$ mai este folosit de logica de control internă circuitului pentru a controla tampoanele de ieşire ale memoriei. În cazul în care $\overline{CAS}$ nu a trecut în zero pentru a valida adresa de coloană, tampoanele de ieşire rămân în stare de înaltă impedanţă (HiZ). Lungimea semnalelor $\overline{RAS}$ şi $\overline{CAS}$ ca şi întârzierea dintre ele trebuie să satisfacă cerinţele de catalog pentru fiecare tip de circuit DRAM. Celula adresată furnizează data stocată prin amplificatorul senzor (care realizează niveluri logice de tensiune la ieşirea sa) către tamponul de ieşire date, via o poartă de I/O. Daca se doreşte scriere, se activează $\overline{WE}$ iar datele se furnizează pe la intrările D_{in}. Informaţia este preluată de poarta I/O, amplificată de amplificatoarele senzor şi înscrisă în celula adresată.

Circuitele de control ale memoriei principale construite cu *DRAM* au trei sarcini principale:

- separarea adreselor de la UCP în adrese de linii şi adrese de coloană;
- activarea corectă a semnalelor $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$
- transmisia şi recepţia datelor citite sau scrise
- generarea semnalelor de control pentru reîmprospătare (la unele memorii trebuie generate din exterior şi adresele de reîmprospătare)

Noile concepte ale DRAM moderne cum sunt modul de lucru prin *întreţesere* şi modul *pagină*, creează sarcini suplimentare pentru circuitul care controlează DRAM. De aceea controllerul extern de memorie reprezintă o parte extrem de importantă a calculatorului. În ceea ce priveşte semnalele furnizate DRAM (figura 7.8), adresele de linii şi coloane trebuie să fie stabile înainte (timp de pre-stabilire[3]) şi după apariţia frontului descrescător al $\overline{RAS}$ respectiv $\overline{CAS}$ (timp de menţinere[4]).

În mod normal selecţia celulei pentru citire sau scriere nu mai are loc dacă succesiunea $\overline{RAS}$ - $\overline{CAS}$ nu s-a produs (Există însă situaţii când această ordine este încălcată, la memoriile moderne unde reîmprospătarea se face cu activare $\overline{CAS}$ înainte $\overline{RAS}$). Mai specificăm pe scurt câteva caracteristici pentru semnalele de control şi temporizările la DRAM convenţionale:

- $\overline{WE}$ = 0 comandă scriere, iar $\overline{WE}$ = 1 citire. Dacă $\overline{WE}$ trece în zero înaintea lui $\overline{CAS}$ ("ciclu de scriere în avans") ieşirile de date rămân în HiZ de-a lungul întregului ciclu de scriere. Dacă însă $\overline{WE}$ trece în zero după $\overline{CAS}$ ("ciclu de scriere - citire") după operaţia de scriere datele apar şi la ieşirile de date ale circuitului. La scriere, ultimul dintre semnalele $\overline{CAS}$ şi $\overline{WE}$ (fronturi căzătoare) captează datele pentru scriere în celulă.
- Semnalul $\overline{RAS}$ mai contribuie la activarea decodificatorului de linii şi a circuitelor de reîmprospatare.
- $\overline{CAS}$ face selecţia circuitului (activează decodificatorul de coloană şi tampoanele de I/O). Cât timp $\overline{CAS}$ = 1 ieşirile de date sunt în HiZ. În cazul în care memoria are şi semnal de intrare de tip $\overline{OE}$ (validare ieşire - Output Enable) funcţiile semnalului $\overline{CAS}$ asupra circuitelor tampon de ieşire sunt preluate de semnalul $\overline{OE}$ suplimentar.
- Intervalele de timp trecute în figura 7.10 se referă la:
  - t_{RAC} = timp de acces faţă de RAS; timpul minim de la activarea (cădere în 0) a liniei RAS până la apariţia datelor valide la ieşire. Adesea este valoarea înscrisă pe dispozitivul DRAM (de exemplu t_{RAC} = 60 ns)

[3] setup time = timp de prestabilire
[4] hold time = timp de menţinere

- t_{RC} = timp de ciclu la operaţia de citire; timp minim între începerea unui acces la o linie şi începutul următorului ciclu de acces.
- t_{CAC} = timp de acces faţă de CAS; timpul minim de la activarea (cădere în 0) a liniei CAS până la apariţia datelor valide la ieşire. Pentru o memorie cu t_{RAC} = 60 ns t_{CAC} este de aproximativ 15 ns.
- t_{PC} = intervalul de timp minim între două accesări succesive la adrese de coloană. Pentru o memorie cu t_{RAC} = 60 ns t_{PC} este de aproximativ 35 ns.

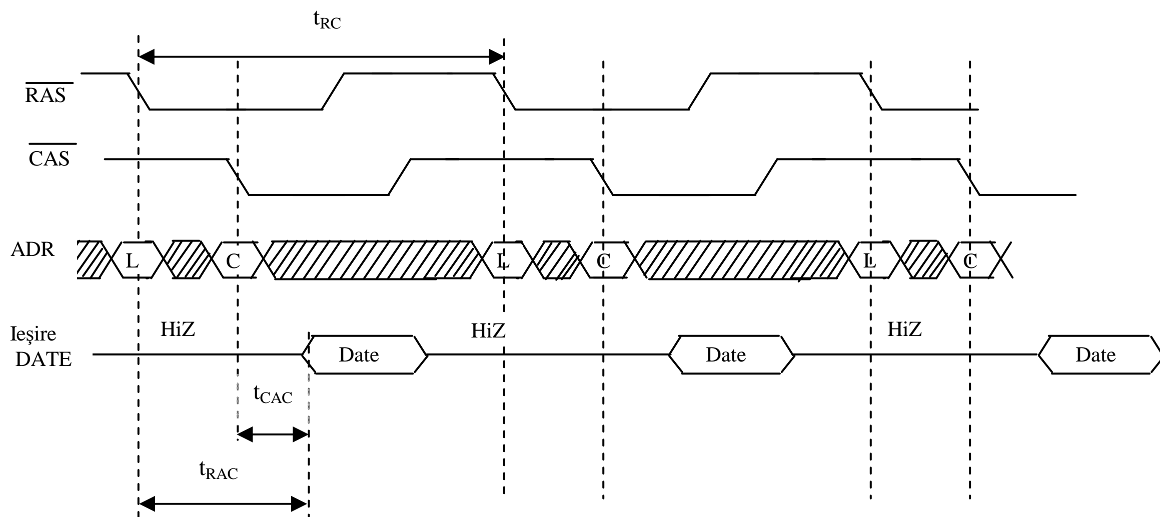


Figura 7.8. Cicluri de acces pentru citire la DRAM convenţional. Pe formele de undă ce simbolizează magistrala de adrese s-a notat cu L adresa de Linie şi cu C adresa de Coloană.

7.3.2. Moduri de lucru de mare viteză ale DRAM

În plus faţă de modul convenţional de funcţionare, descris mai sus, la care adresele de linii şi coloane sunt furnizate pentru fiecare ciclu de acces s-au realizat şi alte moduri de acces la coloane, pentru a reduce timpul mediu de acces la memoria DRAM.

Mod Pagină

Principiul de lucru în modul pagină consideră că pagina corespunde unei adrese de linie şi toate adresele de coloană, cu aceeaşi adresă de linie se găsesc în aceeaşi pagină. În mod pagină controllerul de memorie schimbă în mod repetat doar adresa de coloană. De aceea o “pagină” este echivalentă unei linii din matricea de celule de memorie.

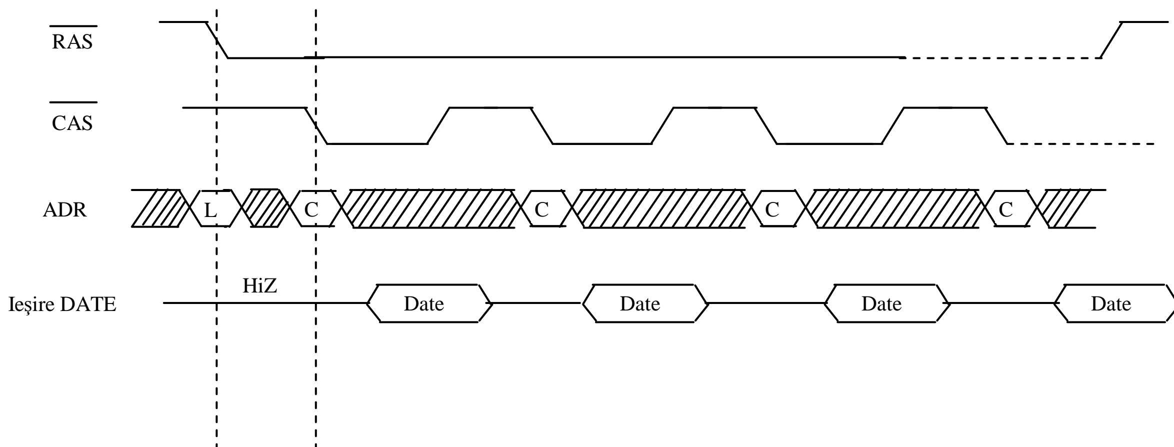


Figura 7.9. Cicluri de acces la DRAM în mod pagină

De exemplu, pentru un DRAM 4 Mbit (cu matrice 1024 linii x 4096 coloane; 1M x 4) o pagină are 1024 locaţii (a câte 4 biţi).

La citire, sau scriere, $\overline{RAS}$ rămâne activat, iar $\overline{CAS}$ se activează de mai multe ori, pentru a valida adrese din aceeaşi pagină. Toate tranzistoarele de acces ale liniei selectate (prin adresa de linie unică paginii) de către decodificatorul de linii, rămân deschise. Toate datele citite de la liniile de bit sunt menţinute într-o stare stabilă de către amplificatoarele senzor. Decodificatorul de coloane va decodifica noua adresă de coloană şi va deschide porţile de transfer corespunzătoare. Se observă că în modul pagină, timpul necesar pentru încărcarea adresei de linie şi timpul de transfer şi decodificare a adresei de linie nu se mai adaugă la timpul de acces, începând cu cel de-al doilea acces pe aceeaşi linie. După iniţializarea adresei de linie (pagină) se transferă şi se decodifică doar adresa de coloană. Astfel că, în comparaţie cu modul standard de acces, timpul de acces este redus cu aproximativ 50% iar timpul de ciclu cu aproximativ 70%.

Durata activa a lui $\overline{RAS}$ nu poate fi însă foarte mare din motive de stabilitate. Tipic, se pot executa aproximativ 200 de accesări succesive într-o pagină înaintea necesităţii ca dispozitivul de control a memoriei să dezactiveze semnalul $\overline{RAS}$ pentru un interval de un ciclu.

Procedurile de scriere şi citire pot fi mixate în mod pagină, fără a fi nevoie să se părăsească acest mod când se comută de la scriere la citire sau invers.

Mod coloană statică ("static column mode")

Este asemănător modului pagină (vezi figura 7.10). La modul coloană statică semnalul $\overline{CAS}$ nu mai comută pentru fiecare adresă nouă de coloană, ci rămâne stabil la nivel jos. După un scurt timp de răspuns circuitul controller DRAM intern recunoaşte că adresa de coloană furnizată s-a modificat. Spre deosebire de modul pagină se economiseşte timpul suplimentar pentru comutarea $\overline{CAS}$ şi timpul de răspuns, deci acest mod este mai rapid decât anteriorul.

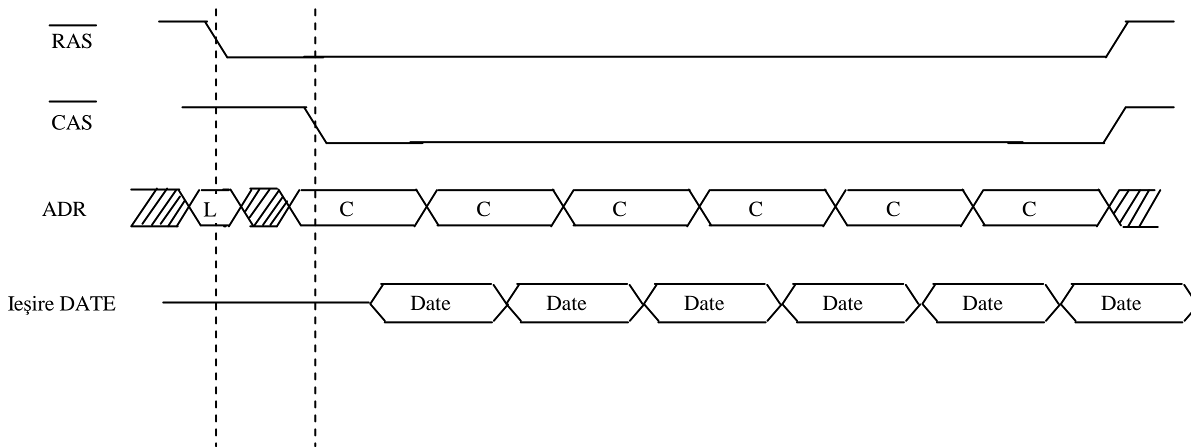


Figura 7.10. Cicluri de acces la DRAM în mod coloană statică

Nici la acest mod de lucru nu este permis ca $\overline{RAS}$ şi $\overline{CAS}$ sa rămână în stare activă (nivel jos) pentru o perioadă foarte lungă de timp. Prin logică internă pe chip se face deschiderea (către tamponul I/O) doar a porţilor ce corespund noii adrese de coloană.

Mod tetradă ("nibble mode, rafale mici, serial4)

Este un mod rafală simplu, prin care 4 biţi de date dintr-o linie şi de la patru adrese succesive de coloană, sunt transmişi secvenţial spre exterior (un nibble este echivalent cu jumătate de octet) prin comutarea lui $\overline{CAS}$ care e repetat de 4 ori. Prima dată accesată este determinată de adresa de coloană furnizată circuitului, iar următoarele trei sunt de la adresele succesive de coloană.

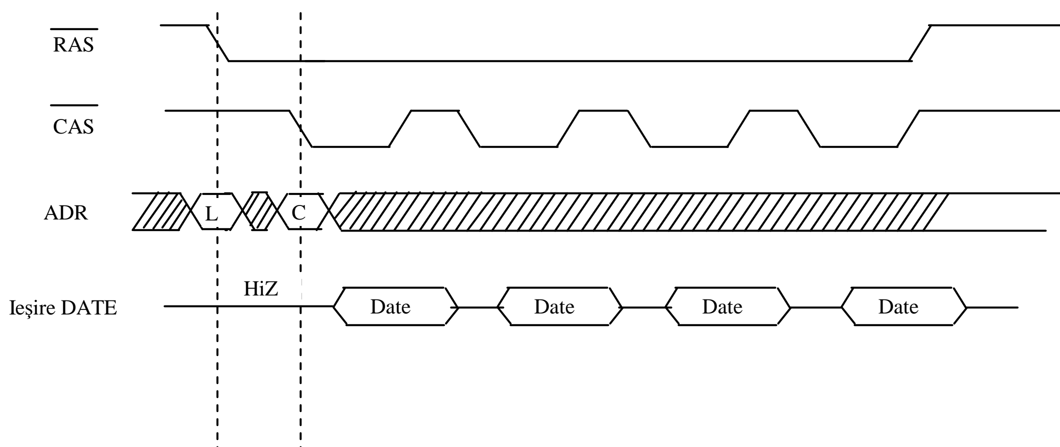


Figura 7.11. Cicluri de acces la DRAM în mod nibble

În mod nibble DRAM are intern (pe chip) un tampon intermediar de 4 biţi, care recepţionează simultan cei 4 biţi succesivi de date şi îi scrie apoi unul după altul la tamponul de iesire, sincronizat cu semnalul de $\overline{CAS}$. Operaţia de transfer se efectuează foarte rapid, pentru că toţi cei 4 biţi adresaţi (unul explicit şi 3 implicit) au fost deja transferaţi către tamponul (buffer-ul) intermediar.

Modul rafală - burst (sau mod serial)

Poate fi privit ca o extindere (o versiune extinsă) a modului nibble. Şi aici, biţii de pe o linie adresată sunt furnizaţi sincronizat cu $\overline{CAS}$. Diferenţa faţă de modul nibble este că numărul de comutări a lui $\overline{CAS}$ nu mai este limitat la 4. În principiu, se poate accesa pe chip o întreagă linie de date succesive - în rafală. În acest caz organizarea internă a chip-ului joacă un rol foarte important, pentru că la un chip de 4 Mbit linia poate avea de exemplu 1024, 2048 sau 4096 coloane.

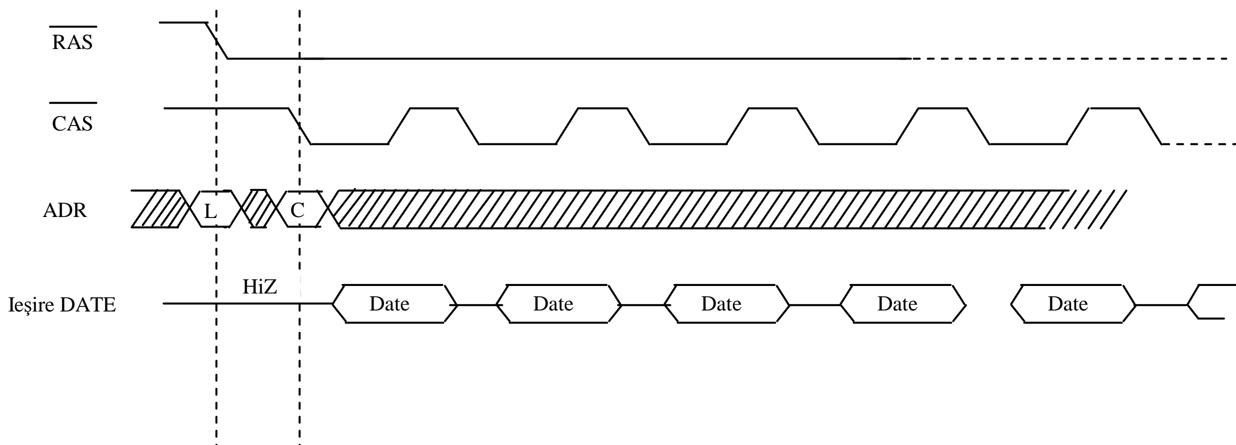


Figura 7.12. Cicluri de acces la DRAM în mod burst

Adresele de linie şi coloană furnizate identifică doar adresa de start a accesului. Circuitele interne DRAM incrementează adresa de coloană independent. Modul rafală (serial) este avantajos pentru citirea memoriei video sau pentru umplerea unei linii de cache, pentru că accesările de citire pentru un controller CRT sau pentru un controller de cache sunt în mod natural seriale pe număr mare de biţi.

7.3.3. Moduri de reîmprospătare

Nu există capacitor perfect, nici chiar în chip-ul DRAM. Descărcarea se face în timp, prin tranzistorul de acces sau prin dielectricul condensatorului. În funcţie de tipul constructiv dispozitivele DRAM obişnuite trebuie să fie reîmprospătate la fiecare 4-16 ms. Formele de reîmprospătare folosite sunt:

1. reîmprospătare numai cu RAS ("ras only refresh")
2. reîmprospătare cu CAS înaintea lui RAS ("CAS-before-RAS refresh")
3. reîmprospătare ascunsă ("hidden refresh")

Reîmprospătare numai cu RAS

Este modul clasic de reîmprospătare. Pentru efectuarea operaţiei se execută un ciclu fictiv de citire, iar odată cu semnalul $\overline{RAS}$ se furnizează adresa de linie de reîmprospătare. $\overline{CAS}$ rămâne inactiv aşa cum se indică în figura 7.13. Dezavantajul acestui mod simplu de reîmprospătare este dat de circuitele externe suplimentare necesare pentru generarea adreselor şi semnalelor de reîmprospătare. De exemplu, la PC-urile cu DRAM canalul 0 al chip-ului DMA din PC este activat de contorul 1 al chip-ului timer 8253/8254 pentru a se emite un ciclu fictiv de transfer.

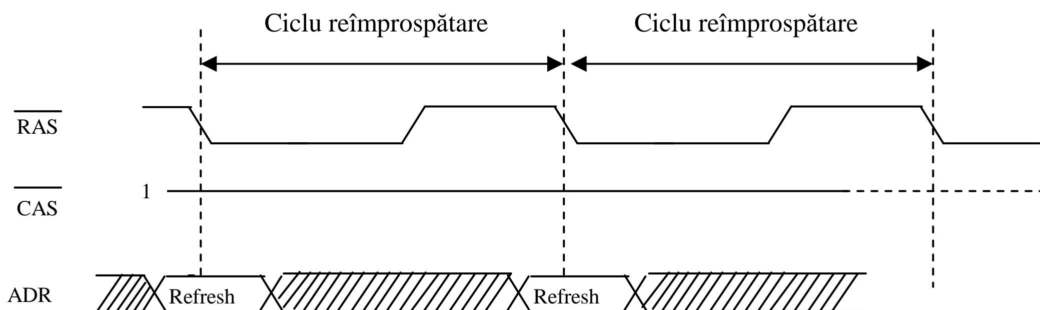


Figura 7.13. Reîmprospătare prin ciclu fictiv de citire (numai cu RAS)

Reîmprospătare cu CAS înaintea lui RAS

Cele mai multe chip-uri DRAM moderne au acces la unul sau mai multe moduri de reîmprospătare interne. Cea mai importantă este aşa numita reîmprospătare cu CAS înainte de RAS (figura 7.14). Pentru asta pe chip-ul DRAM se implementează o logică de reîmprospătare individuală ce are şi contor de adrese. În cazul folosirii acestui mod de reîmprospătare, controllerul extern de memorie menţine semnalul CAS la nivel jos pentru o perioadă predeterminată de timp înainte ca RAS să treacă la nivel jos. Logica internă de reîmprospătare se activează la sesizarea acestei tranziţii şi produce o reîmprospătare internă automată. Adresa de reîmprospătare este produsă intern de un contor de adresă din logica de reîmprospătare. După fiecare ciclu CAS-before-RAS, contorul intern de adresa este incrementat. În acest mod de

reîmprospătare declanşarea procesului de reîmprospătare se face de către controllerul extern de memorie, dar reîmprospătarea se produce intern în chip-ul DRAM.

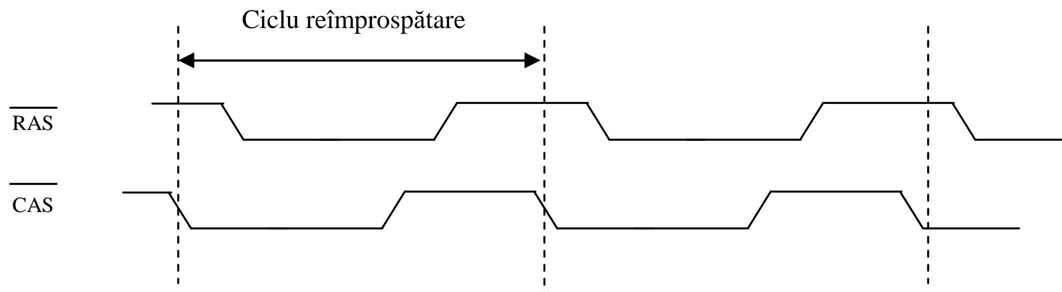


Figura 7.14. Reîmprospătare prin activarea CAS înaintea lui RAS

Reîmprospătare ascunsă

Este de asemenea o metoda elegantă, care economiseşte timpul de lucru cu DRAM (figura 7.15). Aici ciclul de reîmprospătare se produce ascuns în spatele unui ciclu de acces de citire normal. Când se face această reîmprospătare, semnalul $\overline{CAS}$ e menţinut jos, iar $\overline{RAS}$ este comutat.

Datele citite în timpul ciclului vor fi prezente în continuare la ieşirea circuitului, chiar şi în timpul reîmprospătării. Ca durată, un ciclu de reîmprospătare este mai scurt decât un ciclu de citire şi de aceea acest mod de reîmprospătare economiseşte timpul. Şi aici există un contor de adrese de reîmprospătare intern în DRAM. Dacă semnalul $\overline{CAS}$ este menţinut jos un timp suficient de lung, pot fi efectuate mai multe cicluri de reîmprospătare, unul după altul, prin comutarea lui $\overline{RAS}$.

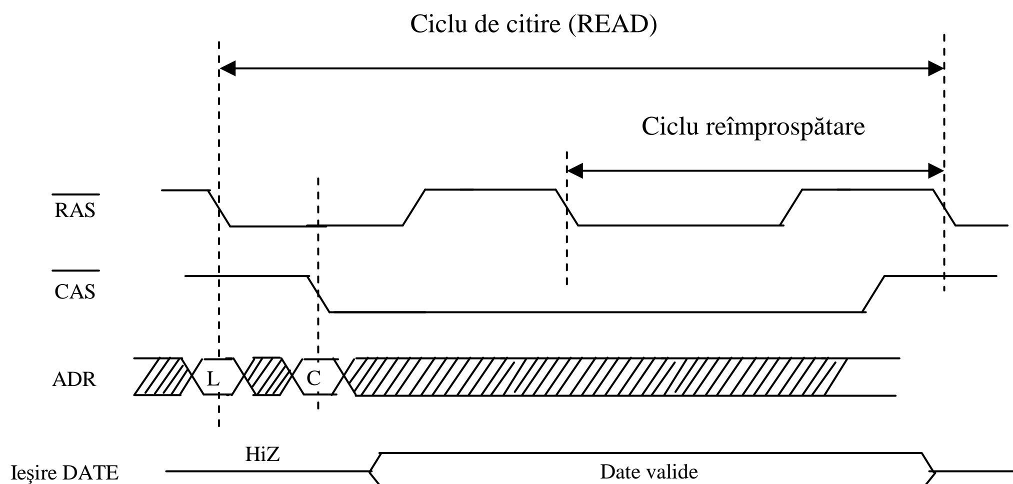


Figura 7.15. Reîmprospătare ascunsă în cadrul unui ciclu de citire

7.3.4. Exemple de circuite DRAM cu interfaţă asincronă

Primele apărute au fost circuitele care lucrau în mod pagină (*FPM* - "fast page mode"). Funcţionează după modul pagină descris mai sus. Au în plus faţă de circuitele DRAM convenţionale o intrare de validare a tampoanelor de ieşire ($\overline{OE}$). La un ciclu de citire, tampoanele de ieşire a datelor se blochează (trec în HiZ) după frontul crescător al $\overline{CAS}$, chiar dacă $\overline{OE}$ rămâne pe nivel jos.

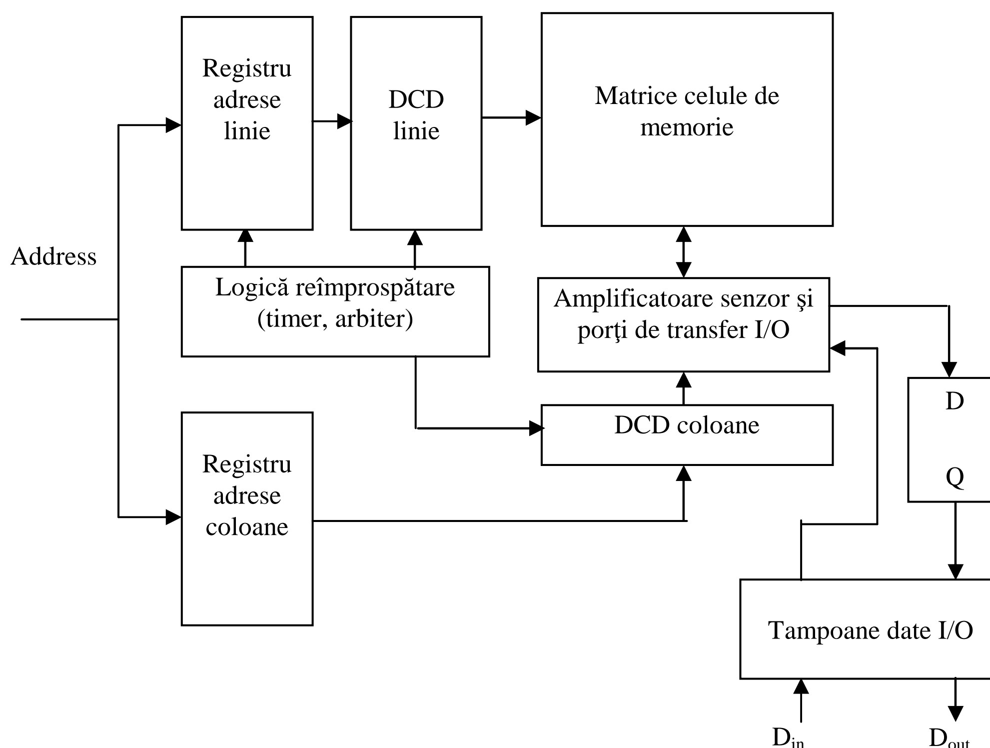


Figura 7.16. Diagramă bloc Extended Data Out (EDO). Se adaugă un latch de memorare la ieşire care permite comutarea mai rapidă decât la FPM a CAS

La circuitele numite *EDO* (Extended data-out DRAM - circuite DRAM cu extindere pe ieşirea datelor) la structura similară cu FPM s-a adăugat un latch la ieşirea amplificatoarelor senzor (figura 7.16). Acest latch permite ca semnalul $\overline{CAS}$ sa treacă pe nivel "sus" în timp ce se aşteaptă ca datele valide să treacă la ieşire; controlul ieşirii datelor făcându-se de $\overline{OE}$. EDO pot sa lucreze la viteze mari, viteza maximă fiind determinată de timpul de acces faţă de adresă (tipic 30 nsec). De aceea transferurile în rafală se pot produce cu 30 % mai rapid decât la FPM.

Burst EDO (*BEDO*, EDO în rafală) a fost creat prin înlocuirea latch-ului de ieşire cu un registru (adică un etaj suplimentar de latch-uri) şi adăugând un latch de adresă şi un contor de 2 biţi. Din cauza registrului de ieşire datele nu ajung la ieşiri după primul ciclu CAS. Totuşi, etajul acestei structuri de tip pipeline (conductă) interne permite datelor să apară într-un timp mai scurt faţă de frontul CAS de activare din al doilea ciclu (adică t_{CAC} mai scurt). Valoarea t_{PC} pentru un BEDO de 52 ns este de 15 ns. BEDO sunt DRAM-uri cu acces-rafală în care toate ciclurile de citire sau scriere se produc în rafale de 4. Pinul CAS produce incrementarea contorului intern de rafală.

Pe măsură ce viteza procesoarelor şi a magistralelor de memorie a continuat să crească, viteza DRAM standard a devenit tot mai inadecvată. Pentru a creşte performanţele generale ale sistemului funcţionarea DRAM a fost sincronizată cu clock-ul sistemului, apărând ceea ce se numeşte *SDRAM* (synchronous DRAM).

7.3.5. Circuite DRAM sincrone (SDRAM)

Spre deosebire de DRAM la SDRAM există în plus: semnal de intrare de ceas (clock), semnal de validare a ceasului, controlul prin RAS şi CAS pe *tranziţia* semnalului de ceas şi operaţie simultană cu două bănci de memorie adresate întreţesut (prin biţii cei mai puţini semnificativi ai adresei). Pentru a uşura interfaţa între sistemul de calcul şi SDRAM toate comenzile SDRAM sunt referite faţă de semnalul de ceas al sistemului.

Performanţele de viteză obţinute cu SDRAM sunt net superioare, faţă de vechile DRAM care aveau o interfaţă asinconă. Una din cauze este că SDRAM poate face transfer în mod rafală de la o serie de adrese (1, 2, 4, 8, sau 256) pe baza furnizării unei adrese de început. Lungimea rafalei poate fi programată de către sistem. Spre deosebire de DRAM-urile convenţionale, care cer ca fiecare acces să fie iniţiat printr-o adresă de coloană trimisă chip-ului, la SDRAM se pot accesa intern până la 256 de adrese succesive, înainte ca o altă adresă de coloană să fie trimisă circuitului. De asemenea SDRAM foloseşte o magistrală internă de date mai largă pentru a obţine viteze mai mari de transfer.

La SDRAM întârzierea dintre validarea coloanei şi obţinerea datelor este pre-programată de procesor şi este numită *întârziere la citire* ("*read latency*"). Întârzierea la citire poate fi programată ca să fie de la 1 la 3 impulsuri de ceas. Această întârziere depinde, la un anumit sistem, de frecvenţa de ceas a sistemului.

Rezumând, principalele diferenţe funcţionale dintre SDRAM şi DRAM sunt:

DRAM

- Nu au intrare de ceas (sistem)
- Control RAS pe nivel
- operare cu o singură bancă de memorie
- un transfer pentru fiecare adresă de coloană (sau impuls de selecţie coloană) furnizată
- Întârzierea la citire (read latency) este neprogramabilă

SDRAM

- lucrează cu frecvenţa ceasului de sistem
- control RAS prin impuls (tranziţie)
- două bănci de memorie pe chip cu întreţeserea adreselor
- transfer rafală (burst) cu 1, 2, 4, 8, sau 256 transferuri pentru o singură adresă de coloană furnizată
- Întârzierea la citire este programabilă

7.3.6. Note privind parametrii de timp ai SDRAM comerciale

Pentrun toate memoriile dinamice sincrone, caracteristica principală este frecvenţa de funcţionare la interfaţa cu exteriorul şi nu timpul de acces cum se întâmplă la memoriile dinamice asincrone. Cu toate acestea, în cataloage se specifică şi parametrii importanţi de timp.
Timpii de acces pentru un SDRAM modern este explicat în tabelul următor [Wiki15]:

<table>
<tr><th>Acces la linie</th><th colspan="5">Accesări coloane</th><th>Preîncărcare</th></tr>
<tr><td rowspan="4">t_{RCD} = 3</td><td colspan="3">t_{CAS} (R) = 3</td><td colspan="2"></td><td rowspan="5">t_{RP} = 3</td></tr>
<tr><td></td><td colspan="4">t_{WR} (W) = 5</td></tr>
<tr><td colspan="2"></td><td colspan="2">t_{CAS} (R)</td><td></td></tr>
<tr><td colspan="3"></td><td colspan="2">t_{CAS} (R)</td></tr>
<tr><td colspan="6">t_{RAS} = 9</td></tr>
</table>

unde:

- **t_{CAS}** (notat uneori doar cu **CL** = **C**AS **L**atency) este . Timpul (calculat ca multiplu al perioadei de ceas) dintre transmiterea adresei de coloană către memorie şi începutul răspunsului prin date la ieşire. Acest timp este cerut pentru a citi primul bit din SDRAM cu adresa de linie corectă deja accesată.
- **t_{RCD}** (**R**ow Address to **C**olumn Address **D**elay) este numărul de cicluri de ceas cerute între deschiderea unei linii de memorie şi accesarea unei coloane din acea linie. Timpul de citire al primului bit din SDRAM fără o linie activă este t_{RCD} + CL
- **t_{RP}** (**R**ow **P**recharge Time) reprezintă numărul de cicluri de ceas cerute între emiterea comenzii de precharge şi deschiderea viitoarei linii din aceeaşi bancă de memorie. Timpul de citire al primului bit de memorie din SDRAM cu deschidere incorectă a liniei este t_{RP} +

t_{RCD} + CL. Termenul de "preîncărcare" reprezintă pregătirea băncii de celule pentru accesul şi deschiderea unei noi linii. Liniile rămân active până la un nou precharge.

- **t_{RAS}** (**R**ow **A**ctive Time) reprezintă numărul de cicluri de ceas între o comandă de activare linie şi emiterea comenzii de precharge. Acesta este timpul cerut intern pentru reîmprospătarea liniei şi incluzând t_{RCD}. În modulele SDRAM este pur şi simplu t_{RCD} + CL.
- **t_{WR}** (Write Recovery Time) este timpul ce trebuie să se scurgă între ultima comandă de scriere într-o linie şi preîncărcarea acesteia

Timpii specifici memoriilor SDRAM se menţionează utilizând 4 parametrii de timp în unităţi de perioade de ceas, de forma: 7-8-8-24**.** Valorile din exemplu corespund succesiv la timpii: CL, t_{RCD}, t_{RP}, t_{RAS}. Aceşti parametrii specifică latenţele ce afectează viteza dispozitivelor de memorie SDRAM. Performanţa absolută a sistemului este determinată de este timpul de latenţă real, măsurat de obicei în nanosecunde. De exemplu pentru CL = 7 la o magistrală de memorie cu clock de 1000 MHz (perioadă de 1ns) rezultă latenţa absolută de 7 ns.

Toate modulele ce conţin CI de memorie SDRAM de tip DIMM moderne (capsulă Dual In-line Memory Module) includ pe chip o memorie ROM serială numită SPD (Serial Presence Detect) care conţine timpii recomandaţi pentru memorie, valori utilizate la configurarea automată. BIOS de pe calculator poate permite utilizatorului să facă ajustări la aceste valori recomandate, în scopul creşterii performanţelor (cu riscul posibil al scăderii stabilităţii).

Pentru citire / scriere, iniţial se transmite adresa de linie. După t_{RCD} linia este deschisă şi poate fi accesată. La SDRAM se poate face acces la mai multe coloane din aceeaşi linie, fiecare citire necesitând un timp t_{CAS}. După terminarea accesului coloanei se face precharge care produce întoarcerea la starea anterioară, după timpul t_{RP}. Celelalte două limite de timp ce trebuie menţinute sunt t_{RAS}, timpul de finalizare al reîmprospătării liniei înainte de a fi închisă şi t_{WR} timpul ce trebuie să se scurgă după ultima scriere înainte ca linia să poată fi închisă.

De-a lungul timpului s-au dezvoltat memorii sincrone dinamice ce lucrează la frecvenţe din ce în ce mai mari. Aşa a apărut noţiunea de DDR (Double Data Rate). Inţial SDRAM erau comandate de un singur front al semnalului de ceas (numite în prezent Modulele SDRAM cu SDR - Single Data Rate). Modulele SDRAM cu SDR au o magistrală internă de de date de 64 biţi, iar transferul se face conform cu frecvenţa magistralei de date, teoretic, pentru fiecare impuls de ceas. Modulul SDR este alimentat la 3,3V. Modulele sunt numite de obicei "PCaaa", unde valoarea numerică "aaa" indică frecvenţa maximă la care poate lucra modulul de memorie. De exemplu: PC100 lucrează la maximum 100 MHz şi are o rată de transfer de 800 MB/s (64/8 x 100).

Memoriile DDR (Double Data Rate) se bazează pe SDRAM, dar nu sunt compatibile cu acestea. Memoriile DDR pot transfera două cuvinte de date pe ciclu de ceas al magistralei de memorie (pe fiecare front al impulsului de ceas) având o performanţă dublă faţă de SDR, la aceeaşi frecvenţă

de ceas a magistralei de memorie. Modulele DDR sunt alimentate la 2,5 V. Interfața cu exteriorul folosește aşa numită pompare dublă (double pumping) care înseamnă pur şi simplu transferul de date de pe ambele fronturi ale semnalului de ceas, pentru a reduce în final frecvența de ceas. Numele de "double data rate" se referă la faptul că un SDRAM DDR cu o anumită frecvență de ceas atinge aproape de două ori lățimea de bandă a unui SDR SDRAM dacă lucrează la aceeași frecvență de ceas. Aceste memorii sunt denumite cu dublul frecvenţei maxime reale de ceas la care pot lucra. De exemplu, memoria DDR-200 lucrează la 100 MHz, DDR2-800 lucrează la 400 MHz, DDR2-1066 lucrează 533 MHz și așa mai departe. Implementarile DDR trebuie să utilizeze de multe ori scheme, cum ar fi bucle PLL şi de autocalibrare pentru a ajunge la precizia necesară a sincronizărilor.

Modulele de memorie sunt de obicei marcate cu frecvenţa maximă de transfer în MB/s. Astfel, cu o frecvență de bus de 100 MHz, DDR-200 ofera o rata de transfer maxima de 1600 MB/s, iar modul memorie este notat PC-1600. Similar memoriile PC-3200, lucrează la o frecvenţă de 200 MHz (DDR-400).

Ulterior au apărut nume "ciudate" ca DDR2, DDR3, DDR4, DDR5 etc. De exemplu DDR2, în plus față de dubla pompare DDR permite viteze mai mari pe magistrală și necesită o putere mai mică pentru funcţionarea ceasului intern, prin înjumătăţirea frecvenţei ceasului intern faţă de ceasul magistralei de date. Rezultă, pentru DDR2, un total de patru (2^2) transferuri de date per ciclu de ceas intern, iar pentru DDR3 un total de 8 transferuri (2^3), de 8 ori viteza matricei interne de memorie). Din păcate diversele module DDRx folosesc tensiuni de alimentare diferite şi interfeţe cu protocoale diferite neexistând compatibilitate între ele.

Pentru utilizator este important să înțeleagă că aceste rate de ceas reprezintă maximul teoretic la care memoria poate fi folosită. De exemplu, dacă se instalează memorie DDR2-1066 pe un calculator care poate accesa subsistemul de memorie la 400 MHz (800 MHz DDR), memoriile vor fi accesate la 400 MHz (800 MHz DDR) și nu la 533 MHz (1066 MHz DDR). Acest lucru se întâmplă pentru că semnalul de ceas este furnizat de către controlerul de memorie, un circuit care este situată în afara memoriei.

7.4. Memoria cache

Performanţele sistemelor moderne de calcul pot fi serios afectate de viteza de lucru a dispozitivelor ce compun memoria principală, viteză în general mai mică decât viteza de lucru a procesoarelor. La ce bun un procesor ce funcţionează cu o frecvenţă de ceas mai mare de 1GHz, dacă memoria principală este lentă şi se introduc multe stări de aşteptare. Circuitele integrate de memorie dinamică tradiţionale (care constituie şi nucleul memoriilor sincrone ce pot fi citite la frecvenţe de ordinul sutelor de MHz) au timpi de acces standard de aproximativ 60 ns, dar timpii medii de acces, prin diferite organizări şi tehnici de sincronizare pot, în prezent, să ajungă sub 10 ns. Se consideră că un sistem funcţionează ineficient, cu multe stări de aşteptare pentru procesor, dacă viteza procesorului este mai mare de cel puţin 3 ori faţă de viteza cu care se transmit datele pe magistrala comună cu memoria; excepţie face cazul când memoria cache este substanţial mărită.

Câteva din soluţiile folosite pentru a compensa aceasta incompatibilitate de viteză între procesor şi memoria principală (MP) sunt:

- introducerea unei memorii intermediare între procesor şimemoria principală, numită memorie tampon (cache) care lucrează cu mare viteză şi are dimensiuni medii / mici;
- suprapunerea operaţiilor de adresare la memoria principală cu alte activităţi efectuate de procesor, prin realizarea unei unităţi de interfaţă cu magistrala care face o aducere (fetch) prealabilă a instrucţiunilor dinmemoria principală şi le stochează într-o coadă locală a UCP;
- organizarea memoriei principale sub forma de blocuri, cu porturi distincte de acces şi cu adrese întreţesute, ceea ce va asigură o viteză medie de transfer mai mare decât inversul timpului de acces la un bloc.

Soluţiile menţionate mai sus nu sunt complet independente. Crucial pentru performanţele sistemului sunt creşterea lărgimii de bandă a memoriei şi scăderea latenţei (întârzierii răspunsului în timp) accesului la memorie.

Memoriile cache (tampon) sunt folosite pentru stocarea unor fragmente de program, date şi rezultate intermediare, frecvent folosite de către procesor. Memoria cache este amplasată între procesor şi memoria principală.

Utilizarea unei memorii cache, de mare viteză, în care se pot stoca atât instrucţiuni, cât şi date, frecvent folosite, are ca scop principal obţinerea unei valori mici pentru timpul mediu de acces al UCP la memorie. Având un timp mic de acces, memoria cache permite ca o operaţie de citire /scriere să se efectueze într-o singură perioadă de tact, ceea ce face ca procesorul să nu între în stări de aşteptare. Prezenţa memoriei cache va fi vizibilă pentru utilizator numai prin sporirea vitezei de lucru la execuţia programelor.

Transferul unor zone de cod sau date între memoria principală şi memoria cache se

realizează automat, prin hardware, fără intervenţia utilizatorului. Circuitele de control ale memoriei cache pot fi construite pe acelaşi chip ca şi UCP sau pot fi externe UCP. Există şi variante mixte, cu două niveluri de memorie cache, un nivel (L1) intern UCP şi celălalt (L2) extern, sau chiar ambele niveluri integrate în aceeaşi capsulă cu UCP. Memoriile cache interne au dimensiuni de ordinul sutelor de KBytes, iar memoriile cache externe au valori de ordinul MBytes. Indiferent de varianta de procesor, dimensiunea memoriei cache se încadrează ca şi capacitate între a zecea şi a mia parte din capacitatea memoriei principale.

Ideea de la care s-a plecat în introducerea memoriei cache este bazată pe *proprietatea programelor de a folosi referinţe localizate*. Proprietatea de localizare a referinţelor se bazează pe structura secvenţială a programelor, cu numeroase cicluri, apeluri de subrutine etc. Astfel încât, pe baza acestei proprietăţi, pentru intervale scurte de timp, adresele generate de un program tipic se referă în mod repetat la câteva zone restrânse ca dimensiune de memorie, în timp ce restul memoriei este accesat relativ rar. Dacă porţiunile active ale programelor şi datelor sunt plasate într-o memorie rapidă, timpul mediu de acces la memorie poate fi redus, asta reducând timpul de execuţie al programelor. Conform proprietăţii referinţelor localizate dacă un întreg bloc de instrucţiuni este copiat din memoria principală în cache, există o mare probabilitate ca un timp toate accesările să se facă la cache.

Atunci când UCP lansează o adresă pentru a face acces la memoria principală se examinează mai întâi conţinutul memoriei cache. Daca cuvântul este găsit, este citit din cache, iar dacă nu este găsit, se accesează memoria principală pentru citirea cuvântului. Deoarece accesul la cache se efectuează foarte rapid, de obicei într-o perioadă de tact, procesorul va controla direct aceste operaţii, în timp ce accesul la memoria principală, mai lentă, se va efectua sub controlul unităţii de comandă a memoriei cache. Transferul între memoria principală şi cache nu se face însă la nivel de cuvânt ci la nivel de bloc de cuvinte, bloc care cuprinde şi cuvântul adresat de UCP şi negăsit în cache. Dacă un cuvânt căutat se găseşte în cache se spune ca s-a produs un *acces reuşit la cache* (sau pe scurt o *reuşită*, "*cache hit*" în limba engleză). Dacă cuvântul căutat nu este în cache s-a produs o *ratare* ("*cache miss*"). Performanţa memoriei cache este adesea măsurată cantitativ prin aşa-numitul *raport de reuşită* ("*hit ratio*", notat în continuare cu hr). Valoarea acestuia se calculează ca raport între numărul total de reuşite şi numărul total de accesări la memorie (ratări plus reuşite). Acest raport se măsoară de obicei prin rularea unor programe de test reprezentative pentru clase de programe. Valoarea lui hr este subunitară dar mai mare decât 0,9. Timpul mediu de acces la o memorie se îmbunătăţeşte considerabil dacă se utilizează memorie cache. De exemplu dacă timpul de acces la memoria cache este de 10 ns, iar la memoria principală de 100 ns, iar hr = 0,9 rezultă un timp de acces mediu de 19 ns. (din 10 accesări nouă sunt reuşite).

$$t_{acm} = \frac{9 \cdot 10 + 100}{10} = 19\text{ns}$$

Un algoritm special de anticipare aduce în memoria cache secvenţele de program care urmează logic unei instrucţiuni deja executate. Procesorul execută programele fără să simtă practic că memoria de lucru este mai lentă. Factorul de succes al memoriei cache este legat de dimensiunea sa şi de eficienţa algoritmului de înlocuire a informaţiei din cache.

Din punctul de vedere al structurii şi organizării memoriei cache există trei tehnici principale (tehnici de mapare):

1. mapare asociativă
2. mapare directă
3. mapare asociativă pe seturi.

Mapare Asociativă

Organizarea memoriei cache ca o memorie asociativă (adresabilă prin conţinut) conduce la viteza şi flexibilitatea cea mai bună. Dar pentru că o memorie asociativă este scumpă şi are un grad mic de integrare, metoda este destul de rar utilizată şi doar pentru implementarea unor memorii de mici dimensiuni.

În cadrul memoriei asociative ce funcţionează ca memorie cache se stochează atât adresele cât şi conţinutul (datele) unor locaţii din memoria principală (figura 7.17).

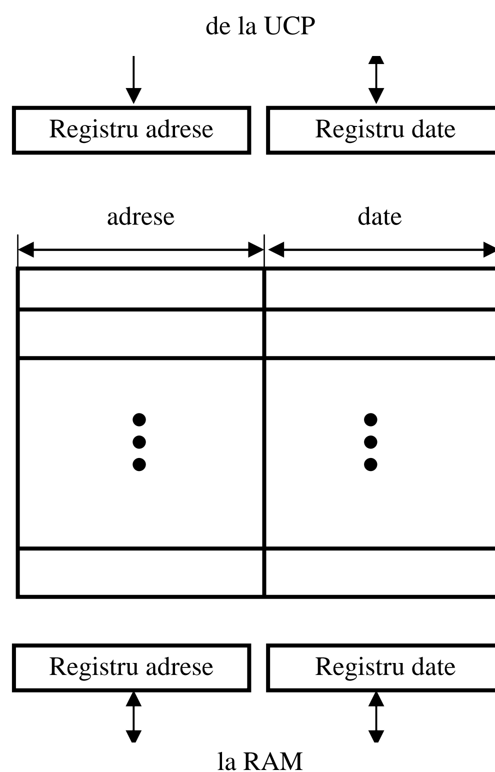


Figura 7.19. Structură de memorie cache asociativă

Asta face ca orice locaţie din cache să poată memora oricare cuvânt din memoria principală. Dacă se produce o ratare, se face acces la memoria principală pentru aducerea unui bloc de date ce poate fi stocat oriunde în memoria cache asociativă. Dacă memoria cache este plină, trebuie ales un mod prin care să se elimine informaţia ce nu mai este necesară din cache. Pentru decizie se folosesc *algoritmi de înlocuire*. O procedură simplă este de tip FIFO (algoritm pe bază de "vârsta în cache" = primul intrat primul ieşit). Alţi algoritmi de înlocuire folosiţi sunt LRU (Least Recently Used), înlocuire aleatoare, LFU (Least Frequently Used).

La lansarea unei adrese de către UCP aceasta se compară cu toate adresele memorate în matricea de adrese a memoriei cache asociative. Dacă compararea combinaţională conduce la o reuşită data corespunzătoare apare la ieşire şi se înscrie în registrul de date (pentru operaţia de citire).

Mapare directă

Memoriile asociative sunt scumpe şi de aceea s-au căutat metode de organizare cu ajutorul unor memorii RAM statice. Exemplificarea acestei mapări este prezentată în figura 7.18. Adresa de *a* biţi (în exemplul din figură a = 32) este logic împărţită în două câmpuri, numite *index* (cel mai puţin semnificativ, care din punct de vedere logic poate fi privit ca fiind format din alte două câmpuri: *bloc şi cuvânt*) şi respectiv *etichetă*. Indexul are un număr *k* de biţi, număr determinat de dimensiunea memoriei cache (2^k cuvinte adresabile prin index), valoarea sa constituind adresa pentru memoria cache. În cache se stochează nu numai data corespunzătoare ci şi eticheta (din formată din *a-k* biţi) asociată datei. De asemenea, lângă etichetă mai pot fi înscrise şi alte informaţii cum ar fi:

- bit ce indică dacă datele sunt valide sau locaţia e privită ca fiind goală,
- bit de protecţie la scriere a locaţiei etc. Protecţia la scriere este utilă atunci când datele se referă la rutine ale sistemului de operare.

La citire câmpul etichetă al adresei de memorie de la UCP este comparat cu eticheta cuvântului găsit în cache şi având aceeaşi adresă de index cu adresa de memorie. Dacă se potrivesc, s-a produs un eveniment de reuşită. Pot exista 2^{a-k} cuvinte cu acelaşi index deci stocabile în aceeaşi linie de memorie cache. Dacă se accesează repetat două sau mai multe adrese cu acelaşi index (distanţa între două asemenea adrese succesive este 2^k) factorul de reuşită scade mult. Dar atunci când se produce o ratare nu se aduce un singur cuvânt ci un întreg bloc de cuvinte, cu aceeaşi adresă de bloc. Blocurile sunt alese cu dimensiunea putere a lui doi. Mai mult, în cache se stochează o singură dată eticheta unui bloc, pentru că ea este aceeaşi pentru toate cuvintele blocului. În figura 7.18 s-a presupus că se foloseşte o adresă de memorie de 32 de biţi, dintre care 8 biţi sunt folosiţi pentru etichetă, 20 de biţi pentru adresarea blocurilor de memorie şi 4 biţi pentru adresarea cuvintelor în blocuri (deci 16 cuvinte pe bloc de cache).

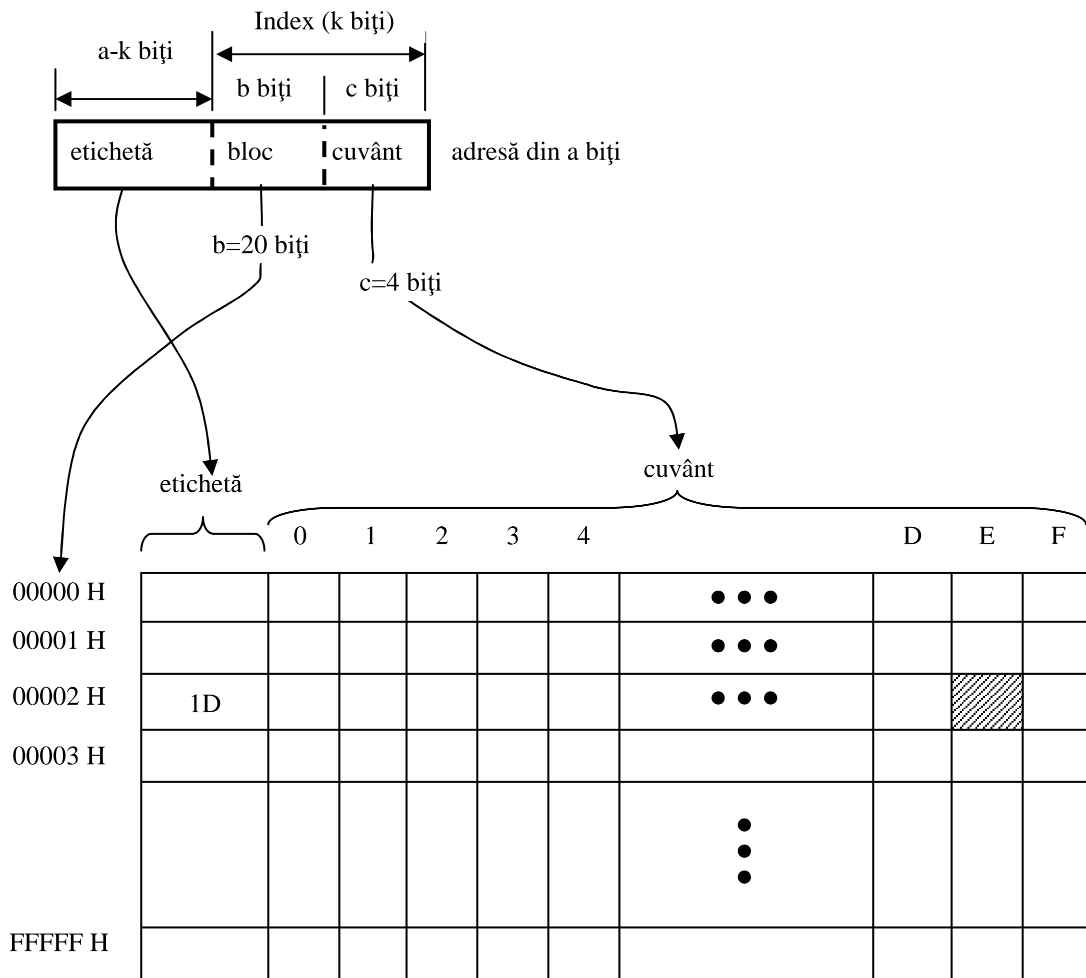


*Figura 7.18. Exemplu de mapare directă la memoria cache. S-a haşurat cuvântul găsit în cache la adresa de memorie: **1D00002E hex***

Maparea asociativa pe seturi:

Înlătura dezavantajul care face ca la maparea directă două cuvinte cu acelaşi index (dar etichetă diferită) nu pot să fie simultan în cache. Ca structură, maparea asociativă pe două seturi ar putea considera două structuri asemănătoare cu cea de la maparea directă, pentru fiecare adresă de bloc putându-se stoca în stivă două blocuri cu etichete diferite (figura 7.19).

Altă deosebire faţă de maparea directă este că atunci când se lansează o adresă, compararea câmpului etichetă al adresei cu etichetele blocurilor stocate în cache se face prin metode combinaţionale (deci prin *asociere* cu conţinutul câmpurilor de etichetă), de unde şi denumirea de mapare asociativă. De obicei partea de etichetă şi comparare combinaţională se păstrează în circuitul controller de cache.

Se pot organiza memorii cache asociative cu două sau mai multe seturi. Ca urmare pentru fiecare adresă de bloc se pot stoca două sau mai multe blocuri, cu etichete diferite.

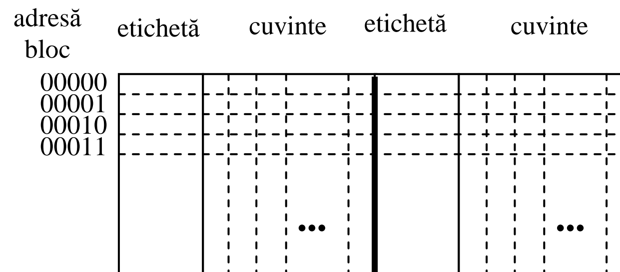


Figura 7.19. Structură bloc simplificată pentru mapare asociativă pe seturi.

Pentru scrierea în memoria cache se folosesc două strategii principale, pe care le-am amintit şi la organizarea pe niveluri ierarhice a memoriei:

- scriere cu propagare către nivelurile inferioare de memorie ("write-through"), notată pe scurt WT
- scriere prin actualizarea nivelurilor inferioare în momentul înlocuirii informaţiei din cache ("write-back"), notată pe scurt WB.

La pornire, pentru scrierea în cache, se forţează ratări pentru toate liniile memoriei cache, până când memoria cache se umple cu date valide.

7.5. Tehnici de adresare şi alocare a memoriei

De regulă la microprocesoarele de 8 biţi, adresa generată şi depusă pe magistrala externă de adrese este folosita direct în adresarea unei locaţii de memorie sau a unui dispozitiv de I/O. Începând cu microprocesoarele de 16 biţi apar noţiunile de adresă logică şi adresă fizică. Adresele generate de un program sunt considerate *adrese logice* şi totalitatea acestora formează *spaţiul adreselor logice*. Totalitatea adreselor ce corespund memoriei (şi/sau dispozitivelor de I/O) formează *spaţiul adreselor fizice*. Cele 2 spaţii, al adreselor logice şi fizice, pot să fie egale sau inegale. Ca urmare, trebuie să existe un mecanism de conversie a adreselor, de translatare din adrese logice în adrese fizice. La microprocesoarele Intel din seria x86 mecanismul de translatare este inclus pe acelaşi substrat de siliciu cu microprocesorul. La unele din microprocesore (ca de exemplu Motorola MC68000) mecanismul de translatare din adresele logice în cele fizice este preluat de un circuit extern microprocesorului denumit "unitate de administrare a memoriei" (MMU - Memory Management Unit).

În general, tehnicile folosite pentru lucrul cu sistemul de memorie într-un calculator, fie că este vorba de selecţie, adresare, alocare de spaţiu în memoria principală, folosesc un ansamblu de resurse hardware şi software, operaţiile executate de acestea fiind incluse în noţiunea generală

de *management al memoriei*. Administrarea întregii memorii (de pe toate nivelurile ierarhice) a calculatorului, incluzând şi modalităţile de partiţionare logică şi adresare în spaţiul adreselor logice (*L*) şi în spaţiul de memorare (*M*, adrese fizice) se face cu ajutorul unei unităţi hardware pe care o notăm în continuare MMU (Figura 7.20).
Când un program trebuie rulat (lansat în execuţie), el este mai întâi adus din memoria auxiliară înmemoria principală (în întregime sau parţial).

Pentru MMU se impun următoarele cerinţe de bază:

1. *să realizeze translatarea adreselor şi să susţină alocarea dinamică a memoriei*
   Înainte ca un program să fie executat, este necesar ca acestuia să i se aloce un spaţiu în memoria principală (spaţiu de adrese în memoria fizic existentă în calculator). Dacă această atribuire de spaţiu (adrese) se face doar la încărcarea programului în memoria principală, procedeul este numit *alocare statică* şi prezintă dezavantajul că adresele de încărcare sunt fixe. *Alocarea dinamică* a memoriei se caracterizează prin atribuirea spaţiului necesar programelor, sau proceselor, în timpul execuţiei acestora. Adresa la care se încarcă programul în memoria principală nu mai este fixă, ea putând fi modificată, la momente de timp diferite.

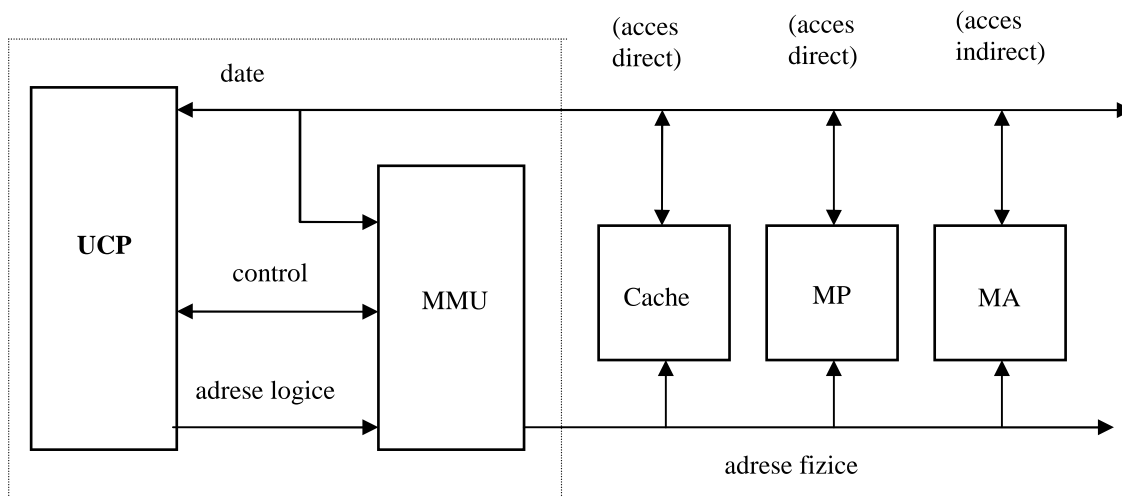


Figura 7.20.: Organizarea ierarhica a memoriei, împreună cu MMU

Se folosesc trei procedee de translatare a adreselor logice în adrese fizice toate bazându-se pe o divizare a spaţiului adreselor logice în blocuri de adrese logice continue, după cum urmează:

- *segmente* de lungime arbitrară;
- *pagini* (în spaţiul adreselor logice) şi *cadre-pagină / blocuri* (în cadrul adreselor fizice), de lungime fixă;
- combinaţii de adresare *segmentat-paginată.*

Mecanismul de translatare, indiferent că se folosesc pagini sau segmente, se produce în faza de execuţie a programului şi include printre altele, un *tabel de translatare* prin care se face corespondenţa între adrese logice şi adrese fizice (principiul de lucru al mecanismului este prezentat în figura 7.21).

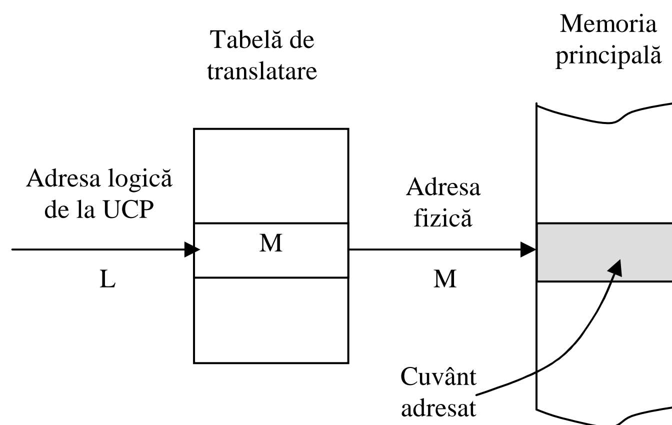


Figura 7.21. Principiul tehnicilor de translatare a adreselor

De la microprocesor se recepţionează adresa logică L. Dacă elementul adresat nu se găseşte în memoria principală, atunci registrul (de adresă) L va conţine adresa din memoria externă unde se găseşte elementul respectiv şi el va fi adus de rutine ale sistemului de operare în memoria principală, la o adresă fizică a cărei valoare M se înscrie în registrul de adresă L din tabela de translatare. Daca elementul adresat este în memoria principală rezultă că adresa fizică este M. Acest mecanism este totuşi ineficient, căci tabela de translatare are dimensiunile egale cu ale memoriei principale. Pentru micşorarea dimensiunii tabelelor se foloseşte divizarea spaţiului logic în segmente / pagini, iar referirile se fac doar la nivelul de adresă logică de bloc. Trebuie menţionat că dacă sunt lansate două procese în execuţie, fiecare cu spaţiul său de adrese logice, există două tabele de translatare spre memoria principală. Sistemul de operare va reactualiza conţinutul tabelei de translatare ori de câte ori va avea loc o relocare (schimbare de poziţie în spaţiul adreselor fizice) a programelor în memorie.

2. *să susţină mecanismele de implementare ale memoriei virtuale*

   Mecanismul memoriei virtuale permite programelor să fie executate chiar dacă numai o parte din instrucţiunile lor se încarcă în memoria principală, iar restul se găsesc în memoria auxiliară de tip disc magnetic. Adresele logice sunt numite adrese virtuale, în cazul implementării memoriei virtuale, pentru că nu există fizic în memoria principală ci ele se găsesc în memoria auxiliară.

3. *să poată furniza protecţie şi securitate pentru memorie*

   Cerinţa se referă la protecţia informaţiei, la alocarea de drepturi de acces la zone din

memoria principală şi la zone din spaţiul adreselor logice, (protecţie la citire, scriere, ştergere, copiere, execuţie). Se protejează memoria disponibilă alocată programelor de sistem şi programelor utilizator, asigurându-se securitatea informaţiei (acces limitat la informaţii), prin controlul accesului la resursele sistemului.

Aşa cum s-a pomenit şi mai sus există două strategii folosite pentru implementarea MMU:

a) MMU se află pe chip-ul procesorului; se aplică la microprocesoarele cu spaţiul segmentat al adreselor logice. De exemplu: Intel x86, Pentium, Zilog Z800, Z8000)
b) MMU este construită separat de UCP, variantă aleasă la microprocesoarele cu spaţiu liniar (organizare liniară a) al adreselor logice. De exemplu: MC 680x0, Z8001).

În cazul în care organizarea logică a memoriei este liniară, adresele încep în mod obişnuit de la 0 şi avansează liniar. Memoria poate fi apoi structurată prin software, la nivelul translaţiei de adrese. În cazul în care organizarea memoriei este segmentată, programele nu sunt scrise ca secvenţe liniare de instrucţiuni şi date, ci ca segmente separate de cod, date, sau stivă. Spaţiul adreselor logice este spart în mai multe spaţii cu adresare liniară, fiecare cu lungime diferită, definită de compilator sau de programator. O adresă logică efectivă este calculată ca o combinaţie între numărul segmentului (care indică adresa de bază a unui bloc de memorie) şi un deplasament (offset) în cadrul segmentului.

În general schemele de adresare liniară sunt mai potrivite pentru manipularea structurilor mari de date, în timp ce segmentarea facilitează programarea, permiţând programatorului să structureze programele în module segment. În plus adresarea segmentată simplifică protecţia şi relocarea obiectelor în memorie. Segmentarea facilitează gestiunea memoriei în sistemele de calcul multiuser, multitasking.

7.6. Tehnici de translatare a adreselor

Indiferent de schema de organizare a memoriei (liniară sau segmentată) procesorul trebuie să aibă un mecanism de translatare a adreselor, util în implementarea memoriei virtuale. Acest mecanism este de asemenea util pentru protejarea informaţiei din memorie. Translatarea de adrese este un proces atribuire şi organizare ("mapare") a adreselor logice în adrese fizice de memorie. Mecanismul de translatare împarte memoria principală în *blocuri (cadre pagină)*. Aşa cum am amintit mai sus se folosesc 3 scheme de translatare:

(1) translatare prin paginare
(2) translatare prin segmentare
(3) translatare segmentat-paginată

În *sistemele paginate*, memoria principală este împărţită în blocuri de *lungime fixă* în timp

ce în *sistemele segmentate* blocurile sunt de *lungime variabilă*.

Paginile au în general lungimi de ordinul 256 - 4096 cuvinte, în timp ce segmentele cu lungimea definită de compilator sau programator au 64 K-cuvinte, sau mai mult. În sistemele cu multiprogramare şi time-sharing[5], mai mulţi utilizatori folosesc aceleaşi programe cum sunt editoare, compilatoare, programe utilitare, biblioteci de programe etc. Atât sistemele cu paginare cât şi cele cu segmentare permit mecanisme de partajare, între procesele utilizator, a paginilor, respectiv a segmentelor. Aceste mecanisme se bazează pe intrări în tabele de mapare de pagină (sau segment) în care intrările diferitelor procese indică către acelaşi bloc din memoria principală.

Combinaţia între segmente şi pagini presupune că un segment conţine una sau mai multe pagini virtuale. Mecanismul de segmentare administrează spaţiul virtual, împărţind programele în segmente, în timp ce paginarea este destinată administrării memoriei fizice care este împărţită în cadre pagina (blocuri).

Vom descrie în continuare mecanismele folosite pentru maparea segmentelor şi paginilor.

7.6.1. Maparea adreselor folosind pagini

În cazul paginării, implementarea tabelului pentru maparea adreselor este simplă pentru că informaţia din spaţiile de adresare şi memorare este divizată în blocuri de dimensiune fixă. Memoria fizică este împărţită logic în blocuri de aceeaşi dimensiune (64-4096 cuvinte fiecare). Termenul *pagină* se refera la blocuri, *de aceeaşi dimensiune*, de adrese din spaţiul de adresare. De exemplu, presupunem un calculator care are 20 de biţi de adresă şi foloseşte doar 32KB în memoria principală (memorie implementată fizic prin circuite de memorie); dacă o pagină, respectiv un bloc, au dimensiuni egale cu 1 kB, atunci spaţiul de adrese e divizat în 1024 pagini, iar spaţiul de memorie e divizat în 32 de blocuri.

Se consideră că şi programele sunt împărţite în pagini. Porţiuni din programe sunt mutate din memoria auxiliară în memoria principală în înregistrări egale cu mărimea paginii. În loc de bloc de memorie e folosit uneori termenul de cadru pagină ("page frame").

În cazul mapării prin paginare adresa virtuală este reprezentată printr-un singur cuvânt de adresă împărţit într-un câmp corespunzător numărului paginii (adresa paginii) şi un câmp pentru deplasament. La maparea prin segmentarea un singur cuvânt de adresă nu mai este suficient; dimensiunea variabilă a segmentelor conduce la existenţa a două cuvinte de adresare, în care primul indică numărul (adresa) segmentului, iar cel de-al doilea deplasamentul în cadrul segmentului.

Într-un calculator cu 2^p cuvinte pe pagină, p biţi sunt folosiţi pentru a specifica o adresă de linie iar cei mai semnificativi biţi rămaşi în adresa virtuală specifică numărul de pagină. Fie de

[5] time-sharing = partajarea timpului UCP

exemplu un sistem simplu cu o adresă virtuală cu dimensiunea de 16 biţi şi pagini cu 4096 cuvinte. Pentru că o pagină are 2^{12} cuvinte, cei patru biţi mai semnificativi vor specifica una din cele 16 pagini, iar cei 12 biţi mai puţin semnificativi indică adresa liniei în cadrul paginii. Ca urmare maparea trebuie făcută doar de la un număr de pagină la un număr de bloc din memoria principală, pentru că adresa liniei e aceeaşi pentru ambele spaţii. Organizarea de principiu a tabelului de mapare a memoriei într-un sistem paginat poate arăta ca în figura 7.22.

În tabelul paginii de memorie, adresa conduce la numărul paginii, iar conţinutul indică numărul blocului unde pagina este stocată în memoria principală. Se observă că paginile virtuale 2, 3, 5 şi 8 se află în memoria principală, în blocurile 3, 0, 1 şi respectiv 2.

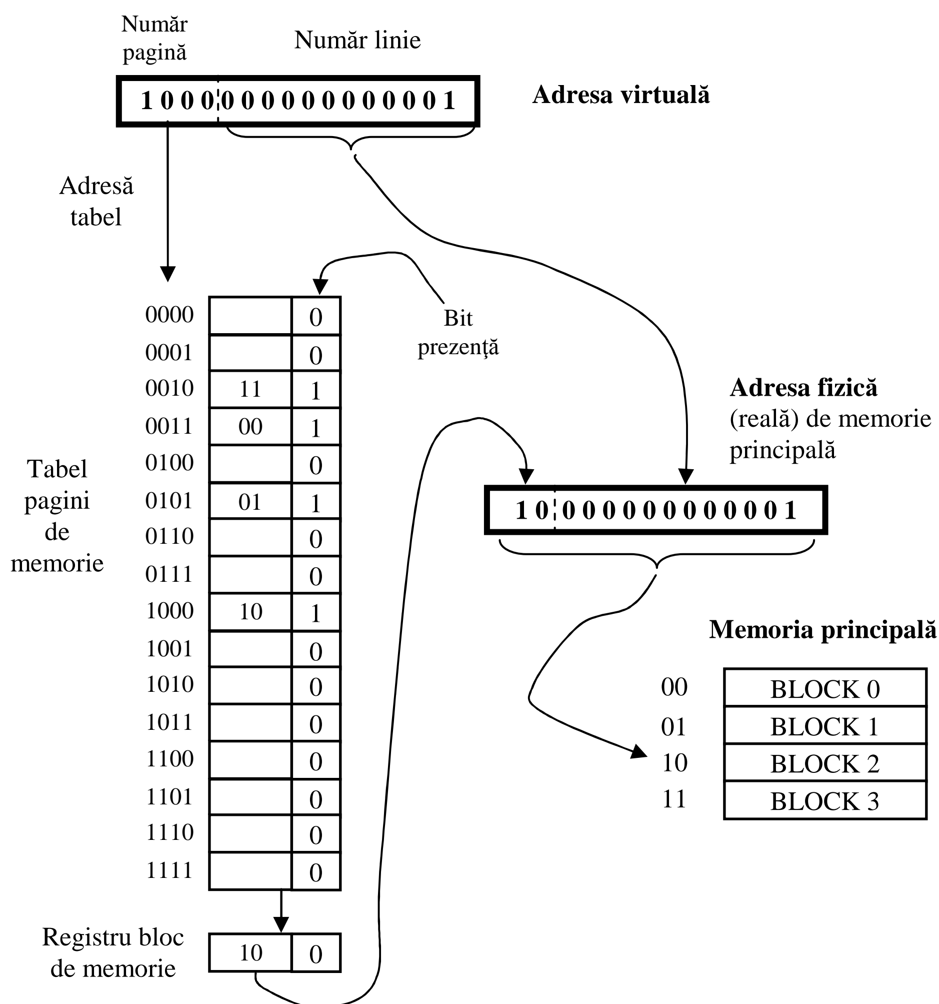


Figura 7.22: Schemă de principiu, explicativă la operaţia de paginare (modalitatea de translatare din figură nu se aplică din cauza dimensiunii mari a tabelului de translatare a paginilor de memorie)

Un bit de prezenţă adăugat fiecărei locaţii a tabelului indică dacă pagina respectivă a fost mapată în memoria principală (i s-a alocat spaţiu şi a fost transferată din memoria auxiliară în memoria principală). Valoarea 0 a bitului de prezenţă indică că pagina nu este în memoria

principală. În exemplul dat, cei 4 biţi ai adresei virtuale specifică atât numărul paginii cât şi adresa în tabelul paginilor de memorie.

Conţinutul tabelului este citit în registrul ce indică blocul de memorie. Dacă bitul de prezenţă este 1, numărul de bloc e transferat în registrul ce păstrează adresa fizică din memoria principală. Dacă bitul de prezenţă este 0 rezultă că adresa virtuală se referă la un articol ce nu se găseşte în memoria principală. În acest caz, se generează o cerere (în forma unei întreruperi software, de tip "eroare de pagină[6]") către sistemul de operare (SO) pentru a aduce pagina cerută din memoria auxiliară în memoria principală, înainte de reluarea programului care a accesat adresa din pagina respectivă.

O astfel de organizare în memoria principală a tabelului de translatare este însă ineficientă, căci tabelul are multe locaţii goale. Dacă presupunem un calculator cu spaţiul de adrese de 4 GB (2^{32}), cu pagini de memorie de 32 KB (2^{15}), iar spaţiul memoriei fizice este 32 MB (2^{25}) rezultă 128 K-pagini virtuale de memorie şi 1024 blocuri (1 K-blocuri). Conform principiului din figura 7.22 tabelul de translatare ar avea 131071 (2^{17} = 128 K) linii şi doar 1024 locaţii ar fi ocupate (cu bit de prezenţă 1), restul fiind goale, neutilizate. O soluţie a acestei probleme o constituie folosirea unei memorii asociative, ca în figura 7.23, (*tabel asociativ de pagini de memorie*) cu număr de locaţii egal cu numărul de blocuri din memoria principală.

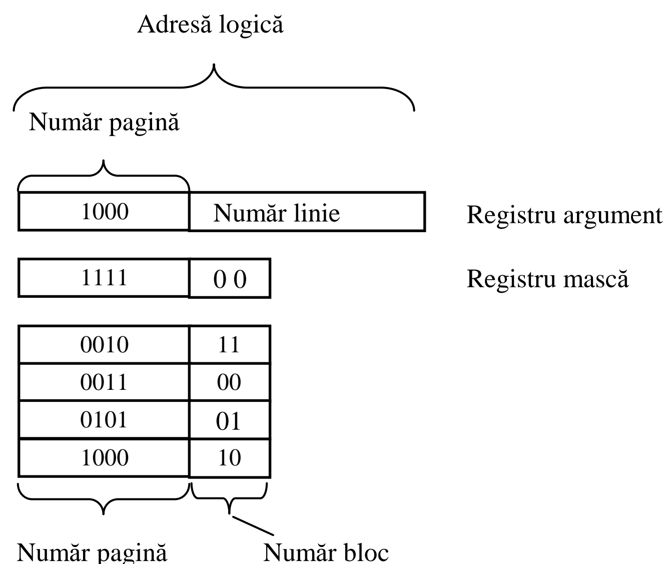


Figura 7.23. Exemplificare a tabelului pentru translatarea adresei de pagină construit cu memorie asociativă (adresabilă prin conţinut)

În fiecare locaţie de memorie se stochează numărul paginii virtuale şi numărul corespunzător al blocului de memorie principală alocat. Adresa logică se compară combinaţional cu conţinutul tabelului asociativ, doar pentru poziţiile binare ce corespund la 1 logic în registrul de

[6] page fault = eroare de pagină

mascare, deci în figura 7.23, doar pentru câmpul de adresă al numărului de pagină virtuală. Dacă la apelarea unei pagini virtuale, aceasta nu este mapată în memoria principală, operaţia de adresare este o ratare (miss) şi evenimentul este rezolvat printr-o rutina de servire a sistemului de operare care aduce pagina respectivă din memoria auxiliară. Şi acest tabel asociativ de translatare poate avea dimensiuni mari uneori. De aceea se folosesc tabele asociative de mare viteză la care numărul liniilor este mai mic decât numărul de blocuri (cadre pagină) şi în care se salvează numai o parte din mapările realizate. Acest registru asociativ este numit TLB (Translation lookaside buffer) şi este descris mai jos.

7.6.2. Mapare prin segmentare

La acest tip de mapare spaţiul adreselor logice este împărţit în *segmente*, care spre deosebire de pagini, *nu au o dimensiune fixă*. Segmentele sunt spaţii de adrese continue (liniare), care includ programe sau porţiuni din programe cu atribute comune. Mecanismul de translatare este prezentat în schema de principiu din figura 7.24.

Spre deosebire de operaţia de paginare, la segmentare nu se mai face concatenarea (alăturarea) adresei de deplasament (din adresa logică) la adresa fizică. Aici adresa de deplasament este adunată cu adresa fizică a segmentului (adresa sa de bază) citită din tabelul de translatare al segmentelor, specific unui anumit proces.

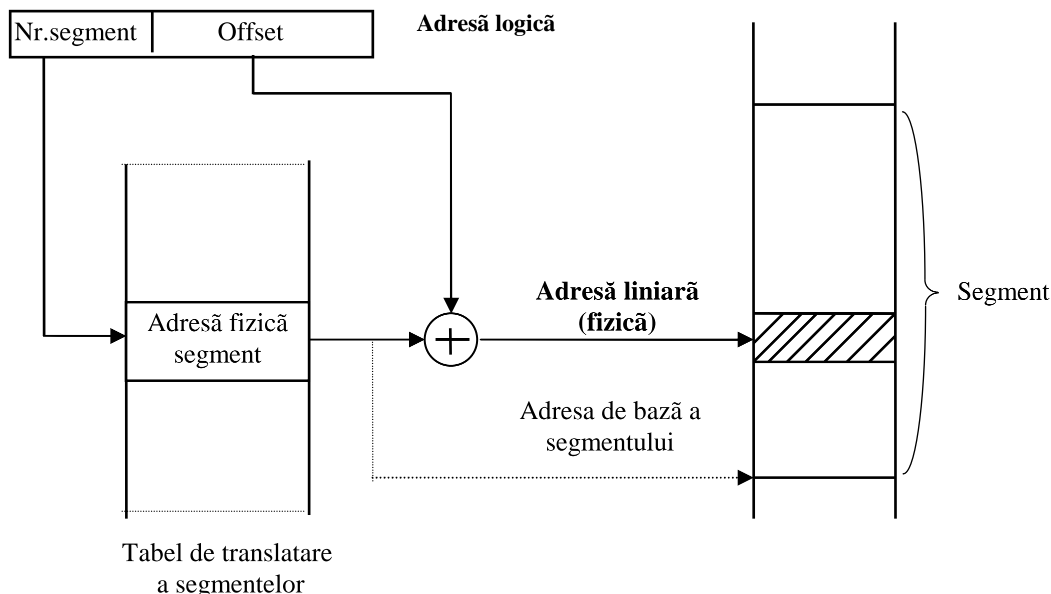


Figura 7.24: Schemă de principiu, explicativă la operaţia de segmentare

7.6.3. Mapare segmentat-paginată

Maparea combinată paginat-segmentată presupune de obicei că un segment include mai multe pagini. Operaţia de translatare începe cu segmentarea şi apoi continuă cu paginarea. Ca urmare doar translatarea de pagină produce transformarea adresei liniare ce rezultă din segmentare într-o adresă fizică. Ca regulă generală la metoda combinată cu segmentare-paginare, segmentele cu lungime variabilă conţin una sau mai multe pagini cu lungime constantă. Acest mod de împărţire conduce la algoritmi mai simpli pentru administrarea şi protecţia memoriei. Există şi excepţii de la această regulă. De exemplu, la procesoarele Intel din seria x86, de la '386 în sus, poate să nu existe o relaţie directă între pagini şi structura logică a unui program (un cadru pagină la 386 are 4KB). La aceste procesoare, segmentele pot avea dimensiunea mai mare sau mai mică decât o pagină. Arhitectura nu impune nici o relaţie între marginile paginilor şi segmentelor. Astfel un segment poate conţine sfârşitul unei pagini şi începutul alteia şi similar pentru relaţia pagină-segment. Chiar dacă procesorul permite această libertate în relaţia dintre pagini şi segmente, se recomandă ca un segment să conţină una sau mai multe pagini, iar paginile să fie incluse complet într-un segment, deci să nu aparţină simultan la două segmente.

Dacă fiecare segment va conţine una sau mai multe pagini de memorie (de lungime fixă), lungimea segmentului se poate specifica atât în multiplu de octeţi (Bytes) dar şi în multiplu de pagini de memorie. Adresa logică va fi împărţită în trei câmpuri: primul indică numărul segmentului (ca intrare într-un tabel de translatare pentru segmentele procesului), al doilea un număr de pagină, (care indică o adresă relativă într-un tabel de translatare paginată în cadrul segmentului), iar al treilea câmp este deplasamentul. Dacă numărul de biţi ai câmpului pagină este p, rezultă maxim 2^p pagini în segment.

Valoarea citită din tabelul segment este un pointer către baza tabelului de pagină. Suma dintre baza tabelului de pagină şi numărul paginii (din adresa logică) este pointer către o intrare în tabelul de pagină, aşa cu se indică în figura 7.25.

Valoarea găsită în tabelul de pagină furnizează numărul de bloc din memoria fizică. Cele două tabele pot fi construite în memoria principală, dar de asemenea pot fi construite separat de memoria principală, eventual într-o memorie locală procesorului. Ultima variantă este mai avantajoasă ca viteză, pentru că organizarea în memoria principală conduce la trei accesări succesive, deci viteză mică. Se foloseşte de aceea o memorie asociativă pentru a stoca în tabele intrările cele mai recent referite. Aceasta memorie asociativă este numită "*Translation Lookaside Buffer*" = tabel pentru memorarea funcţiilor de translatare (*TLB*), şi are structura din figura 7.26.

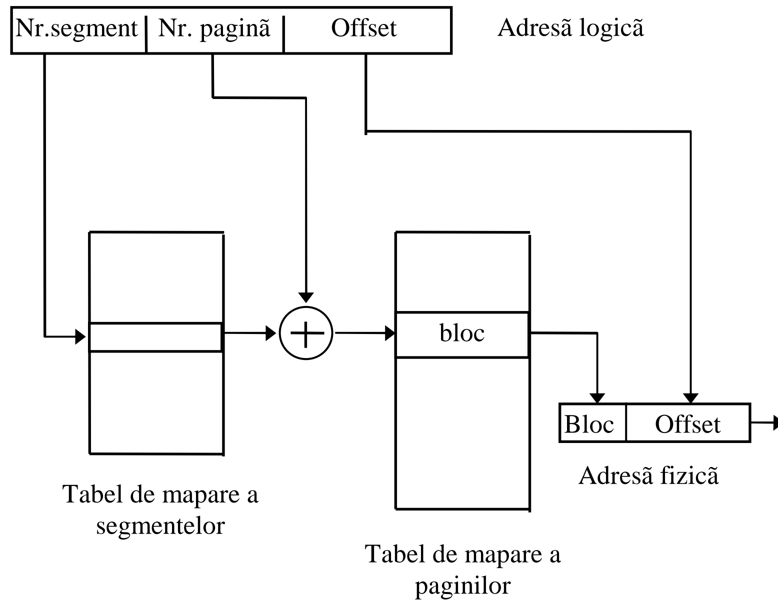


Figura 7.25. Principiul operaţiei de translatare segmentat-paginată

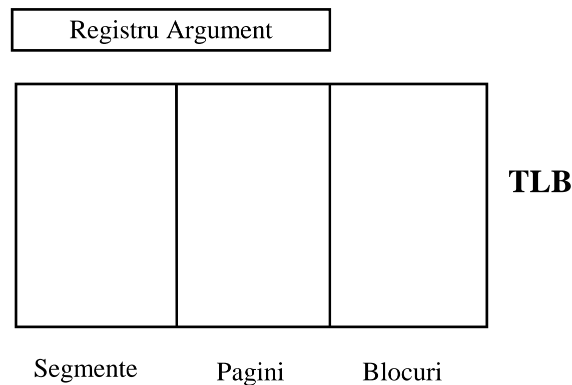


Figura 7.26. Structura de principiu a unui TLB (tablou asociativ pentru memorarea funcţiilor de translatare)

Prima oară când un anume bloc e referit, valoarea sa împreună cu numerele corespunzătoare de segment şi pagină intră în TLB. Dacă, la adresare, adresa cerută nu este mapată în TLB se foloseşte maparea mai lentă, cu tabele în memorie, descrisă mai sus, iar rezultatul e înscris în memoria asociativa (TLB) pentru referirile ulterioare.

7.6.4. Memoria virtuală

Într-un sistem ierarhic de memorie, programele şi datele sunt mai întâi stocate în memoria auxiliară. Porţiuni din programe sau date sunt aduse în memoria principală pe măsură ce ele sunt

necesare unităţii centrale de procesare. Memoria virtuală reprezintă un ansamblu de tehnici (software) şi mecanisme (hardware) folosit pentru a mări capacitatea aparentă a memoriei principale RAM. Astfel, numărul de biţi ai adresei ce rezultă prin execuţia unei instrucţiuni, depăşeşte ca posibilităţi de adresare, capacitatea memoriei principale, fizic instalată. În aceste condiţii, din programul stocat în memoria externă vor fi aduse în memoria principală numai zonele active la un moment dat (figura 7.27). Procesul acesta de transfer între memoria externă (disc magnetic) şi memoria principală este transparent pentru utilizator, el desfăşurându-se automat, sub controlul sistemului de operare. Adresele logice furnizate de UCP şi MMU explorează un spaţiu de adresare ce corespunde ca dimensiune, capacităţii memoriei virtuale, mult mai mare decât dimensiunile memoriei principale.

Din punct de vedere *fizic*, memoria virtuala va fi organizată în *memoria auxiliară* (disc magnetic). Se măreşte deci capacitatea aparentă a memoriei cu "acces direct". Numărul de adrese accesibile în mod direct aflate la dispoziţia programatorului este substanţial mai mare decât numărul de locaţii din memoria principală fizic existentă; spaţiul de adresare corespunde capacităţii memoriei virtuale. Memoria virtuala este organizată, din punct de vedere *logic*, sub formă de pagini, astfel că o adresă furnizată de o instrucţiune a procesorului conţine două câmpuri: numărul paginii şi respectiv, adresa în pagină.

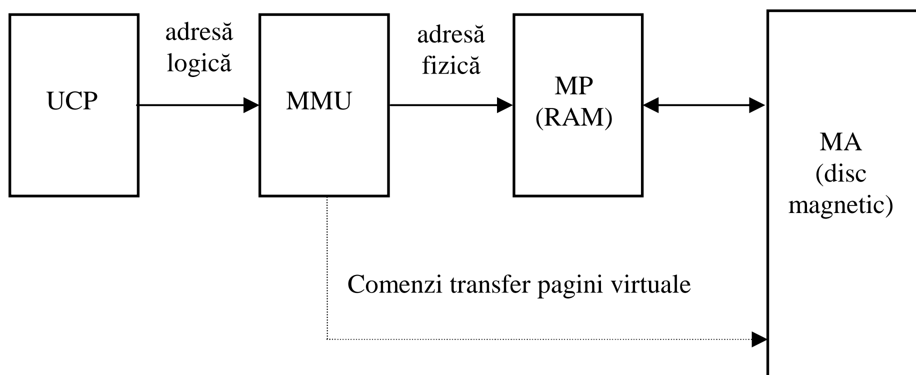


Figura 7.27. Schema bloc de principiu a sistemului de memorie virtuală (MMU =unitate de control a memoriei, MP = memoria principală, MA = memoria auxiliară)

Dimensiunile unei pagini sunt mult mai mici decât dimensiunea memoriei principale. În acest mod, în memoria principală pot fi aduse, din memoria secundară, mai multe pagini active, ale unui program sau ale mai multor programe. Un sistem cu memorie virtuală furnizează un mecanism pentru translatarea adreselor generate de program în adrese fizice ale locaţiilor memoriei principale. Operaţia de translatare se face dinamic, în timp ce programele se executa în UCP.

Se va considera, pentru exemplificare, organizarea unei memorii virtuale cu capacitatea de

64 MB (2^{26}), structurată pe cuvinte de 32 de biţi: 16 Mega-cuvinte x 32 de biţi/cuvânt (2^{24} x 32). Într-un cuvânt - instrucţiune de 32 de biţi, 24 de biţi sunt alocaţi pentru adresarea memoriei virtuale. Primii 12 biţi, mai semnificativi, vor specifica numărul paginii, iar ultimii 12 biţi, mai puţin semnificativi, vor constitui adresa în pagină. În figura 7.28. se prezintă organizarea logică a memoriei virtuale în 2^{12} pagini, fiecare pagină având capacitatea de 2^{12} cuvinte de câte 32 de biţi. Organizarea fizică a memoriei virtuale în cadrul memoriei auxiliare este prezentată de asemenea în figura 7.28. Programul utilizatorului este constituit dintr-un număr oarecare de pagini, organizate într-un segment. Prima pagină, din cadrul fiecărui segment, conţine, pe lângă un identificator de pagină, un identificator de segment şi numărul de pagini din segmentul dat. Celelalte pagini vor fi constituite dintr-un antet de identificare pagină şi din pagina propriu-zisă. În cazul de faţă se consideră că sistemul de operare se ocupă cu gestiunea segmentelor/fişierelor, în memoria externă.

Fie de exemplu o memorie principală cu capacitatea de 256 k $\times$ 32 de biţi. Se pot stoca în cadrul memoriei principale maximum 2^6 = 64 pagini, având fiecare o capacitate de 2^{12} = 4096 cuvinte. Nucleul sistemului de operare va fi rezident în memoria principală şi va ocupa, de exemplu, primele 8 pagini (32 k-cuvinte). Sistemul de operare va gestiona o serie de tabele de translatare a adreselor logice / virtuale în adrese fizice.

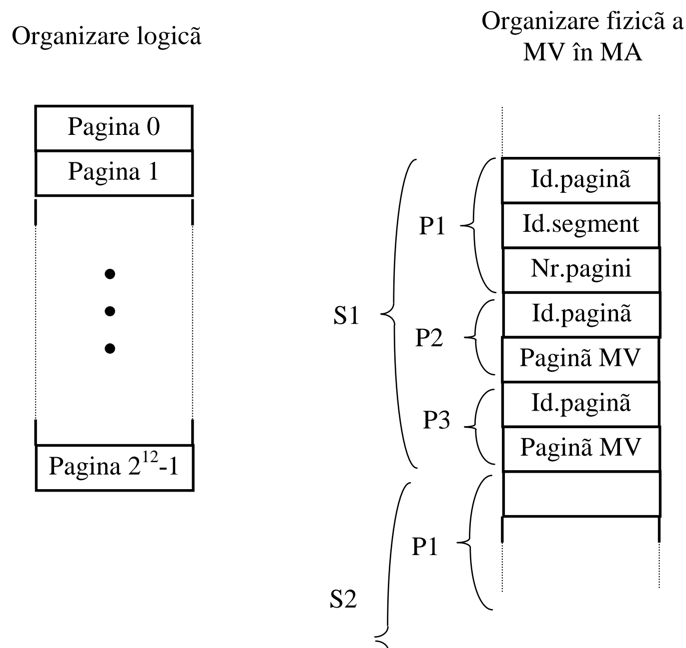


Figura 7.28. Organizarea logică şi fizică a memoriei virtuale. Cu P_i s-a notat structura paginii fizice i din memoria virtuală, iar cu Si s-a notat structura segmentului i. (MV = memorie virtuală,MA = memorie auxiliară)

Schimbarea paginilor între memoria principală şi memoria auxiliară se face după algoritmi

de înlocuire asemănători cu cei de la memoria cache. Pentru exemplul dat anterior în figura 7.29 se prezintă modalitatea de translatare a adreselor virtuale, folosind următoarele notaţii:

- TA - Tablou Asociativ în celulele căruia sunt stocate numerele paginilor virtuale aduse în memoria principală. Conţine 64 de cuvinte de câte 12 biţi.
- CU - Contoare de Utilizare a paginilor prezente în memoria principală. Se folosesc în mecanismul de înlocuire. La fiecare acces contoarele ce corespund intrărilor neselectate se decrementează, până la zero, iar contorul liniei selectate se setează la valoarea maximă. Dacă este nevoie de spaţiu pentru noi intrări vor fi înlocuite liniile care au contoarele cu conţinut 0;
- BLOC - Memorie cu conţinut permanent (doar cu citire), adresată asociativ, care transformă numărul paginii virtuale în număr de pagină fizică din memoria principală RAM;
- AM - Registrul de adrese pentru memoria principală (adresa fizică), de 18 biţi
- RI - Registrul de Instrucţiuni, cu 32 de biţi
- PC - Contorul de Program cu lungimea de 24 de biţi;
- MP - Memoria principală RAM (2^{18} x 32 biţi);

Dacă o pagină oarecare din memoria virtuală a fost adusă în memoria principală, ea poate fi localizată, ca număr de pagină fizică, prin referirea la o memorie asociativă cu conţinut fix pe care o notăm BLOC. În cazul coincidentei între numărul paginii virtuale, şi conţinutul unei celule din Tabloul Asociativ, celula corespunzătoare, situată pe aceeaşi linie, în BLOC, va furniza primii 6 biţi mai semnificativi ai adresei fizice. Aceştia vor fi concatenaţi în registrul AM, cu ultimii 12 biţi din RI sau PC, pentru a forma adresa fizică de acces la memoria principală. Primele opt celule din BLOC au conţinuturi fixe, corespunzând numerelor primelor opt pagini din memoria principală în care se află nucleul sistemului de operare.

Figura 7.29 Mecanismul asociativ care asigura suportul pentru translatarea adreselor virtuale în adrese fizice.

Primele opt celule din TA vor conţine numerele paginilor virtuale corespunzătoare nucleului sistemului de operare. Conţinutul acestora va fi fix. De asemenea, conţinuturile primelor opt contoare de utilizare nu vor suferi decrementări. Gestiunea memoriei virtuale începe prin transferul unui segment sau al unui număr de pagini, dintr-un segment, corespunzător unui program din memoria secundară în memoria principală, sub controlul sistemului de operare, şi lansarea în execuţie.

Dacă se constată lipsa unei pagini, se trece la aducerea paginii virtuale în memoria principală. În figura 7.30. operaţiile din blocurile 1 şi 4 se execută prin software, în timp ce operaţiile din blocurile 2 şi 3 se execută prin hardware. Operaţiile din cadrul blocului 2 sunt operaţii obişnuite de citire-interpretare şi execuţie a instrucţiunilor programului. După generarea întreruperii de tip lipsă-pagină în memoria principală întreaga responsabilitate pentru transferul paginilor revine sistemului de operare. Într-o memorie auxiliară pe disc magnetic organizată cu scopul de a facilita implementarea memoriei virtuale, pagina de memorie auxiliară va înlocui înregistrarea, drept cea mai mică unitate de date adresabilă pe disc. Sistemul de operare va gestiona o serie de tabele de translatare a adreselor logice în adrese fizice.

După terminarea transferului programul întrerupt se poate relua. La transfer, dacă memoria principală este plină, trebuie ştearsă o pagină veche pentru a face loc celei noi.

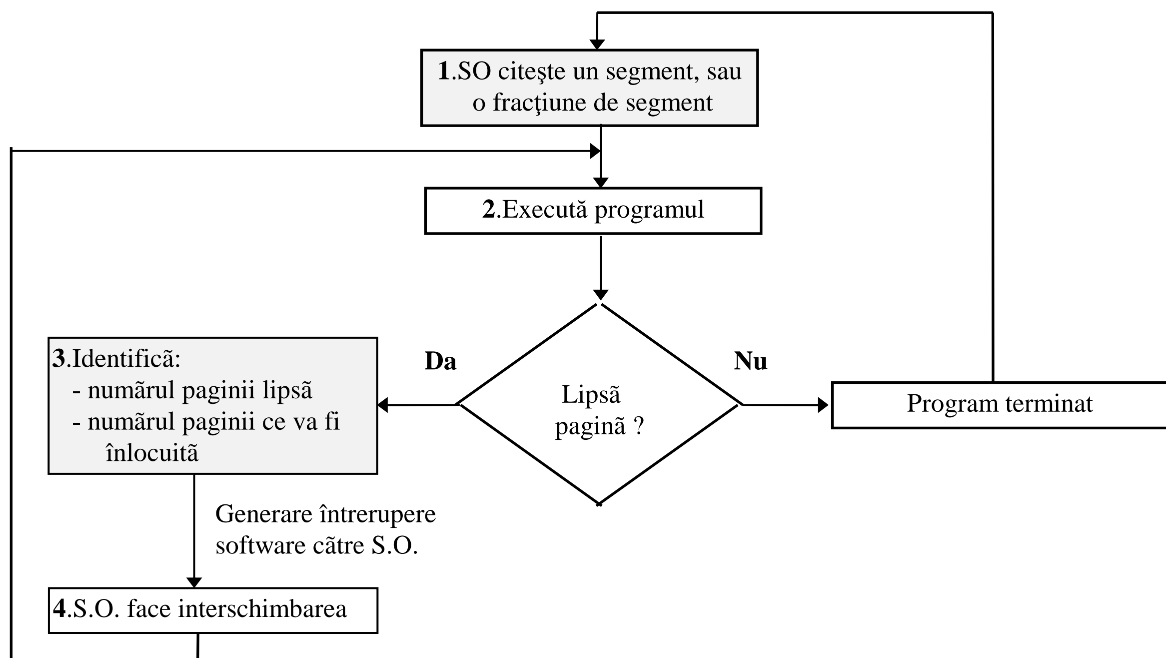


Figura 7.30. Explicativă la mecanismul de încărcare a paginilor virtuale.

Algoritmul de înlocuire poate fi de tip, de exemplu:

- FIFO - selectează pentru înlocuire paginile care au stat în memorie cel mai lung timp. De fiecare data când o pagina e încărcată în memorie, numărul sau de identificare este

introdus într-o stivă FIFO (primul intrat, primul ieşit). Dacă tamponul FIFO este plin şi trebuie adusă altă pagină, se elimină pagina cea mai "veche" în tampon şi care e în vârful stivei FIFO. Este un algoritm uşor de implementat dar nu foarte eficient.

- LRU - least recently used (cel mai puţin recent utilizat) este un algoritm mai dificil de implementat, dar mai eficient. Se înlocuieşte cea mai puţin recent folosită pagină. Foloseşte câte un contor pentru fiecare pagină ce se află în memoria principală. Când se face referire la o pagină, contorul asociat ei este setat la la valoarea maximă, iar celelalte contoare asociate paginilor se decrementează. Ele sunt numite adesea registre de vârstă, pentru ca ele indică vârsta cât n-au mai fost referite.

7.6.5. Noţiuni privind protecţia memoriei

Inventarea multiprogramării, la care un procesor este partajat între mai multe programe aflate în execuţie, a condus la noi cerinţe privind protecţia între programe. Mecanismele de protecţie sunt strâns legate de implementarea memoriei virtuale.

Multiprogramarea conduce la conceptul de proces. Un proces este constituit dintr-un program aflat în execuţie plus orice stare necesară pentru continuarea rulării sale. Partajarea timpului ("time-sharing") este o variantă a multi-programării care partajează UCP şi memoria între diferiţi utilizatori simultani, iar mecanismul de partajare dă iluzia că toţi utilizatorii au propriile maşini. Astfel că, în oricare dintre variante, trebuie să fie posibil să se comute de la un proces la altul. Această comutare este numită *comutare de proces* sau *comutare de context*.

Un proces trebuie să se execute corect indiferent dacă se execută continuu de la start până la terminare, sau dacă este întrerupt în mod repetat pentru a se face comutarea către alte procese. Responsabilitatea menţinerii comportării corecte a proceselor cade atât în sarcina resurselor hardware, care asigură salvarea şi restaurarea corectă a proceselor ce rulează pe UCP cât şi în sarcina sistemului de operare care trebuie să garanteze că un proces nu interferează cu celelalte şi că un proces nu poate modifica datele altor procese. Aceasta înseamnă asigurarea protecţiei datelor de către sistemul de operare. În afară de protecţie, sistemul de operare permite, în anumite condiţii, şi partajarea codului şi a datelor între diferite procese pentru salvare de spaţiu de memorie prin reducerea numărului de copii de informaţii identice.

Cel mai simplu mecanism de protecţie este constituit de o pereche de registre care verifică fiecare adresă, astfel încât să nu se permită accesul în afara unor limite alocate în spaţiul de adrese. Aceste registre sunt numite registru - bază şi registru limită. O adresă este validă dacă se încadrează între cele două adrese conţinute în registre:

Bază ≤Adresă ≤Limită

La unele sisteme adresa este considerată ca un număr fără semn, care se adună întotdeauna la o adresă de bază, astfel că testul de adresă limită se reduce la:

(Bază + Adresă) ≤ Limită

Dacă proceselor utilizator li s-ar permite să modifice registrele de adrese de bază şi limită, atunci acest mecanism de protecţie nu ar putea funcţiona. Modificarea respectivelor registre este permisă doar sistemului de operare, pentru asigurarea protecţiei între procese.

Pentru asigurarea protecţiei sistemul de operare (SO) are trei responsabilităţi principale:

1. Să furnizeze cel puţin două moduri de execuţie indicând că procesul aflat în rulare este un proces utilizator, sau un proces al sistemului de operare. La diferite SO ultimul tip de proces este numit în diferite feluri: *proces kernel* (nucleu), *proces supervizor*, sau *proces executiv*.
2. Să prevadă o porţiune a stării CPU pe care procesul utilizator o poate doar citi, dar nu o poate scrie. Aceasta include registrele bază / limită, indicatori (biţi) pentru moduri utilizator / supervizor şi indicatori pentru validare / invalidare evenimente de tip excepţie.
3. Să prevadă un mecanism prin care UCP poate trece din mod utilizator (user) în mod supervizor şi viceversa. Primul sens de trecere este specific apelurilor sistem ("system calls" – apeluri de servicii oferite de SO), implementate ca instrucţiuni speciale care transferă temporar controlul la o locaţie precisă din spaţiul de cod al supervizorului. Registrul contor de program corespunzător locului unde se face un apel sistem este salvat, iar UCP trece în mod supervizor. Revenirea la modul utilizator este similară cu cea produsă la o revenire din procedură.

Adresele de bază şi limită constituie minimul unui sistem de protecţie. Mecanismul memoriei virtuale oferă alternative mai complexe şi mai sigure decât modelul simplu prin bază şi limită. Aşa cum s-a văzut, adresele virtuale sunt translatate în adrese fizice pe baza unor tabele de translatare. Acest mod de mapare, prin tabele, oferă posibilitatea introducerii de informaţii pentru controlul erorilor de program (intenţionate sau nu) care încearcă să treacă peste mecanismele de protecţie. Cea mai simplă cale este introducerea unor indicatori de permisie pentru fiecare pagină sau segment. De exemplu pot exista indicatori care să permită doar citire, doar execuţie, sau care să interzică accesul unui proces utilizator la anumite pagini / segmente. Fiecare proces poate face adresare doar către paginile proprii de memorie, procesul utilizator neavând dreptul să modifice tabelele sale de pagină / segment.

Protecţia poate fi extinsă chiar pe mai mult decât două niveluri (nu doar utilizator şi supervizor), privite ca şi inele concentrice de protecţie, în centru găsindu-se nivelul de protecţie cel mai înalt. În această ierarhie de niveluri de protecţie un program poate accesa doar date de pe nivelul său de protecţie şi de pe nivelurile inferioare în ierarhie. Poate face însă operaţii de apelare (call) a serviciilor sistemului de operare, servicii oferite de rutine ce se află pe niveluri

superioare de protecţie. Adesea se face comparaţia, în oarecare măsură nefericită, cu clasificări de tip militar ale sistemelor: top – secret, secret, confidenţial şi neclasificat. Programele utilizator ("civilii" în exemplul militar) au doar dreptul de acces la nivelul de protecţie cel mai de jos: "neclasificat. La sistemele de protecţie de tip inele concentrice, deosebirea faţă de exemplul anterior, este că se pot face apelări la rutine situate pe niveluri superioare de protecţie, dacă există "chei" de acces către acele niveluri. Poate exista dreptul de apelare a unor servicii ale sistemului de operare, prin mecanismul de comutare a proceselor.

Aşa cum am pomenit şi mai sus, informaţia de protecţie este setată în registre speciale ataşate fiecărei intrări în tabelele de translatare, registre setate doar de rutine de control al sistemului de operare. Drepturile de acces pot fi de tipul:

a. Atribuirea de privilegii complete de citire şi scriere. Aceste drepturi se atribuie programului atunci când execută propriile instrucţiuni.
b. Read-only (protecţie la scriere). Protecţia la scriere este utilă la operaţiile de partajare a unor rutine de sistem (utilitare, biblioteci, etc.).
c. Execute only (program protection). Protejează programul la copiere. Restricţionează referirea la segment doar în timpul fazei de fetch a instrucţiunii şi nu şi în timpul fazei de execuţie. Asta permite utilizatorului să execute instrucţiunile segmentului de program, dar nu permite citirea instrucţiunilor ca date cu scopul de a copia conţinutul lor.

Sistemul de protecţie a memoriei, în sensul celor spuse mai sus, se construieşte pentru:

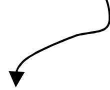
(a) memorie (detectează orice eroare de adresare înainte ca aceasta să creeze erori accidentale sau voite);
(b) programe (previne ca programele utilizator (de aplicaţii) să facă modificări ilegale în rutinele SO);
(c) utilizatori (programele utilizatorilor între ele);
(d) securitate informaţie (acces limitat la informaţiile unde utilizatorul nu are drept de acces).

7.7. Exerciţii

1. Se consideră o memorie cache cu opt linii (ce pot stoca opt blocuri din MP), iar MP împărţită logic în 32 de blocuri. Dacă memoria cache este goală indicaţi (inclusiv prin schema bloc a memoriei cache) unde se poate stoca blocul numărul 15 al MP în cazul în care memoria cache foloseşte: (a) Mapare directă; (b) Mapare asociativă pe seturi cu dimensiunea doi; (c) Mapare complet asociativă

2. Se presupune o memorie cache mapată direct, cu 4 linii (cadre bloc de memorie). Pentru o succesiune de referinţe la adresele de bloc descrise mai jos (în format hexazecimal), determinaţi dacă la adresa de bloc marcată se obţine HIT sau MISS.

   0, 1, 3. 3, A, B, A, B, C, 3, 1, 0, 4, **3,** 4, ...



HIT sau MISS ?

3. Pentru un sistem de memorie (memorie principală şi memorie cache cu mapare directă) având parametrii:
   - Memoria principală (MP) este adresabilă pe octet şi are capacitatea de 16 MB
   - Pentru maparea în cache MP este împărţită (logic) în 4M blocuri de câte 4Bytes
   - Cache cu capacitate de stocare 64 KB
   - Dimensiunea liniei memoriei cache (bloc) = 4Bytes

   Determinaţi:

   a. Dimensiunea adresei de memorie principală
   b. Numărul de linii (bloc) adresabile în cache
   c. Dimensiunea etichetei (tag)
   d. Desenaţi schema bloc a memoriei cache cu mapare directă şi indicaţi în care linie din cache se va stoca cuvântul cu adresa de memorie: 1D000A (hex)

4. Pentru aceleaşi valori numerice ca în problema anterioară, dacă maparea este complet asociativă, în care din liniile memoriei cache se poate stoca cuvântul cu adresa de memorie: 1D000A (hex)
5. Descrieţi caracteristicile principale ale dispozitivelor de memorie
6. Care este deosebirea între tip de acces şi timp de acces?
7. Faceţi o comparaţie între memoriile RAM statice şi dinamice (moduri de funcţionare, structură internă, timp de acces, grad de integrare, mod de selecţie şi adresare)

Capitolul 8

Arhitectura microcontrolerelor

Conţinut

8.1. Introducere

8.2. Caracteristici principale ale microcontrolerelor

- 8.2.1. Grad mare de integrare
- 8.2.2. Organizarea memoriei şi accesul la memorie
- 8.2.3. Organizarea sistemului de I/O
- 8.2.4. Utilizarea circuitelor Counter - Timer
- 8.2.5. Sistemul de întreruperi
- 8.2.6. Sistemul de management al puterii

8.3. Exerciţii

8.1. Introducere

În primele capitole anterioare ale acestei cărţi am tratat aspectele generale ale arhitecturii microprocesoarelor şi de asemenea am introdus principalele noţiuni privind memoria şi sistemul de I/O pe care microprocesorul, ca unitate centrală de procesare le controlează. În acest capitol vom face o introducere în arhitectura microcontrolerelor. De ce doar un capitol despre microcontrolere? Motivul este că foarte multe dintre aspectele tratate până aici sunt aplicabile şi microcontrolerelor. Vom purcede în continuare doar la sublinierea diferenţelor de organizare şi implementare în lumea microcontrolerelor comparativ cu microprocesoarele de uz general.

Microcontrolerele sunt mici calculatoare care conţin procesor, memorie şi periferice pe acelaşi circuit integrat. Denumirea indică un *microcalculator complet pe un chip*, destinat *aplicaţiilor de control*. Astfel că în cele mai multe din cazuri tot ce este necesar pentru a le utiliza este să se introducă *software* în ele (*firmware* = software încorporat în hardware).

Chiar dacă un microcontroler conţine toate subsistemele pe care le conţine şi un calculator de uz general, acestea sunt mult simplificate pentru a putea fi integrate în aceeaşi capsulă de circuit integrat (CI). De exemplu memoria inclusă are dimensiuni foarte reduse. Tipic, capacitatea de stocare pentru memoria de program este între 1 şi 128 KB, iar pentru memoria de date între 128 B şi 4 KB. Nucleul microprocesor inclus funcţionează la frecvenţe de ceas mici (kHz - MHz) pentru a reduce consumul de putere al microcontrolerului. Circuitele de interfaţă de I/O pot fi paralele (porturi digitale de 8 biţi) sau seriale sincrone sau asincrone.

Microcontrolerele sunt utilizate pentru controlul automat al unor procese din lumea externă, cum ar fi: automobile şi alte echipamente de transport, aparatură medicală, telecomenzi, jucării, industrie şi energie, tehnică militară, electronică de consum etc.

Microcontrolerele sunt proiectate pentru sistemele numite în prezent sisteme cu calculator încorporat (embedded systems), spre deosebire de microprocesoarele de uz general folosite, în special, în calculatoare de uz general. Un sistem cu calculator încorporat este un sistem pe bază de microprocesor construit pentru a controla anumite *funcţii particulare* şi care nu este construit pentru a fi programat de utilizatorul final, cum este la calculatoare desktop. Bineînţeles că funcţiile particulare pentru controlul unui proces pot fi oferite şi de un calculator de uz general (construit prin asamblare de microprocesor, memorie şi sistem de intrare ieşire), dar microcontrolerele sunt net superioare în acest domeniu ăn primul rând datorită dimensiunilor şi costurilor scăzute. În ultimele decenii au apărut pe piaţă circuite integrate de tip microcontroler care conţin pe lângă circuitele digitale şi circuite analogice de prelucrare. Au fost numote procesoare mixte de semnal. Dar necesitatea controlului lumii externe (analogică) a făcut ca multe unităţi centrale de procesare să fie înlocuite cu procesoare digitale de semnal (DSP Digital Signal Processor) care optimizează prin setul de instrucţiuni operaţiile de tip înmulţire şi sumare

repetată, specifice prelucrării digitale a semnalului. Ca urmare a acestor modificări a apărut noţiunea de CI Controler digital de semnal (DSC - Digital Signal Controller), numit adesea tot microcontroler, singura deosebire faţă de microcontrolerele clasice (microcalculator pe un chip care are ca unitate centrală un microprocesor), este că UCP este un DSP (Digital Signal Processor).

Mai trebuie menţionat că pe piaţă se găsesc familii de microcontrolere (compatibile binar, la nivelul setului de instrucţiuni) deosebirile în cadrul unei familii fiind legate de reusrsele implementate (memorie, interfeţe) şi dimensiunile acestora.

Dacă ne referim la dezvoltarea numărului de microcontrolere pentru domeniul embedded, se remarcă că, în prezent, peste 98% din dispozitivele digitale programabile produse sunt încorporate în echipamente electronice şi maşini prezente atât în casele noastre cât şi la locul de muncă [16]. Se estimează că în 2010 existau circa 16 miliarde de componente embedded programabile, ceea ce înseamnă că pentru fiecare locuitor al planetei există în medie mai mult de 2 dispozitive embedded. În viitor, considerând actuala rată de creştere se va ajunge la peste 40 miliarde de sisteme embedded în 2020 [14].

Ca o sinteză a deosebirilor dintre microprocesoarele de uz general şi microcontrolere (o parte tratate şi în capitolul 1) enumerăm pe scurt principalele deosebiri:

Microcoprocesor	**Microcontroler**
• Poate funcţiona ca maşină de calcul doar dacă adăgăm în exterior memorie şi sistem de I/O	• Este o maşină de calcul autonomă, având toate componentele necesare incluse în circuitul integrat (nucleu microprocesor, memorie, sistem I/O)
• Este destinat aplicaţiilor de uz general	• Este destinat aplicaţiilor de control
• Performanţe ridicate în viteza de prelucrare, pentru a satisface o gamă largă de aplicaţii rulate	• Performanţe mai reduse de viteză, dar suficiente pentru aplicaţiile rulate
• În setul de instrucţiuni există puţine instrucţiuni ce pot manipula individual biţii unui cuvânt	• În setul de instrucţiuni există multe instrucţiuni ce pot manipula individual biţii unui cuvânt
• Sunt utilizate în PC-uri, staţii de lucru, servere, laptop-uri, unde compatibilitatea software, performanţa, generalitatea şi flexibilitatea sunt importante	• Sunt utilizate în sisteme cu calculator încorporat, unde fiabilitatea, gabaritul şi costul sunt extrem de importante
• La capsulă există puţini pini cu funcţii multiple, multiplexate	• La capsulă există mulţi pini cu funcţii multiple, multiplexate
• Capacitatea de adresare a memoriei principale este foarte mare (tipic maxim sute de MB - GB)	• Capacitatea de adresare totală a memoriei principale este redusă (tipic KB)
• Pentru a creşte viteza medie de acces la memorie foloseşte o ierarhie de niveluri de memorie interne şi externe	• Memoria de program şi date este inclusă în CI şi au rar nevoie de niveluri externe de memorie

8.2. Caracteristici principale ale microcontrolerelor

Vom împărţi principalele caracteristici distictive ale microcontrolerelor în următoarele categorii [Viena07]:

1. Grad mare de integrare
2. Organizarea memoriei şi accesul la memorie
3. Organizarea sistemului de I/O
4. Utilizarea circuitelor Counter - Timer
5. Sistemul de întreruperi
6. Sistemul de management al puterii

8.2.1. Grad mare de integrare

Indiferent de tipul circuitului microcontroler, structura generală poate fi descrisă conform figurii 8.1., în care se descriu prin diagrama bloc principalele componente ale unui microcontroler. Toate sistemele componente sunt conectate prin magistrale interne, iar *legătura cu lumea externă* se face de obicei prin *terminalele de I/O* şi nu prin terminale dedicate unor magistrale funcţionale ca la microprocesoarele de uz general. Nu veţi putea descoperi la terminalele unui microcontroler magistrală de adrese, sau magistrală de date.

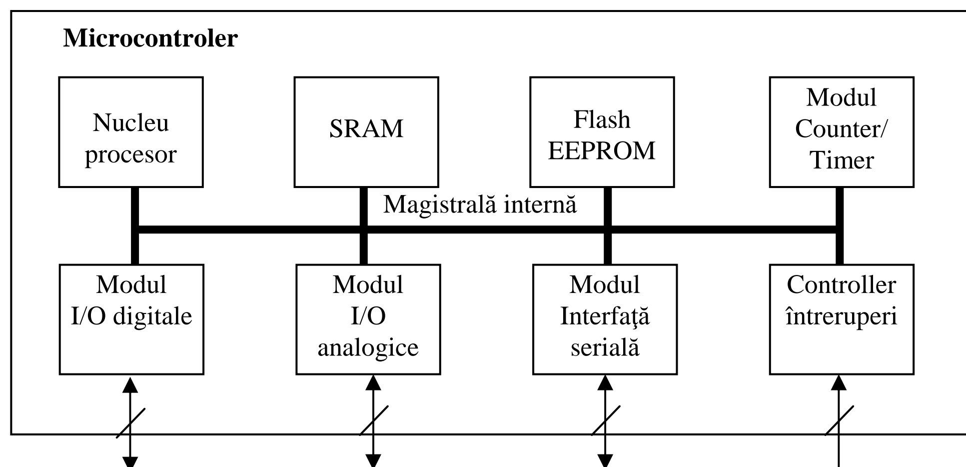


Figura 8.1. Structură bloc generală a componentelor unui microcontzroler, cu indicarea tipurilor de interfeţe cu exteriorul.

Un microcontroler are un grad mare de integrare incluzând mai multe componente decât un microprocesor de uz general. Prin integrarea mai multor componente necesare în acelaşi CI se asigură în primul rând cerinţele de asigurare a funcţiilor cu minim de hardware extern, dar şi cerinţele de preţ redus şi putere mică. Numărul şi tipul componentelor integrate diferă de la circuit la circuit, fiind influenţate de piaţa ţintită de procesor. Cel mai adesea procesoarele microcontrolerele includ:

- Nucleu microprocesor, cu ALU, Unitate de control, registre, uneori cu unităţi funcţionale multiple: ALU, FPU, shifter, MAC
- Memorie pe chip (de date şi program), RAM, ROM în diverse tehnologii de realizare.
- Circuite timer-counter. numărare evenimente, măsurare intervale de timp, determinarea momentului de timp a unui eveniment, generarea de semnale dreptunghiulare cu o anumită frecvenţă, controlul acţionărilor electrice etc.
- Controller de întreruperi interne şi externe
- Circuitele de administrare a puterii consumate
- Registre programabile pentru configurarea funcţionării fiecărei unităţi funcţionale interne
- Circuite periferice şi pentru controlul periferiei, printre cele mai obişnuite incluzând:
  - Ieşiri / Intrări binare
  - Intrări/ieşiri analogice – convertoare analog/digitale (ADC), comparatoare analogice, filtre antialiere
  - Interfeţe de comunicaţie serială SPI, SCI (UART, USART), I^2C, CAN etc.
  - controllere diverse: LCD, DMA, Co-procesoare comunicaţie reţea
  - Interfeţe pentru depanare şi testare standardizate

8.2.2. Organizarea memoriei şi accesul la memorie

La microcontrolere se foloseşte arhitectura Harvard pentru spaţiul de memorie, existând memorii separate pentru program, respectiv pentru date. Uneori memoria de program (cod) poate fi împărţită în două tipuri de memorie doar cu citire (ROM). Un modul de memorie de program este de tip flash memory, iar aici se stochează programul executabil, iar celălalt modul poate fi de exemplu o memorie EEPROM ce poate fi folosită pentru stocarea unor tabele, sau a unor variabile ce se modifică rar şi se păstrează în memorie ne-volatilă.

Modul de organizare şi modul de acces la spaţiile de adresare interne sunt de două tipuri, în funcţie de tipul de microcontroler:

a) Spaţii diferite de adresare în funcţie de tipul de memorie
b) Spaţiu comun de adresare, indiferent de tipul de memorie

a) Spaţii diferite de adresare în funcţie de tipul de memorie

La aceste microcontrolere fiecare tip de memorie este accesată separat, de obicei prin moduri diferite de adresare, fiecărui modul corespunzându-i un anumit mod de adresare. Se folosesc modurile de adresare: direct, indirect, acces printr-un registru index la memoria EEPROM). În figura 8.2 se prezintă un exemplu al acestui mod de organizare şi adresare, considerând că există 3 tipuri de memorie internă: 32KB memorie Flash, 16KB memorie EEPROM, 16 KB memorie de date (SRAM - static RAM). Se observă că fiecare spaţiu de adresare începe cu adresa 0, acesta fiind motivul pentru care fiecare tip de memorie se adresează cu propriul mod de adresare.

SRAM	EEPROM	Flash
$0000	$0000	$0000
16K	**16K**	**32K**
$3FFF	$3FFF	$7FFF

Figura 8.2: Adresare în spaţii diferite de memorare

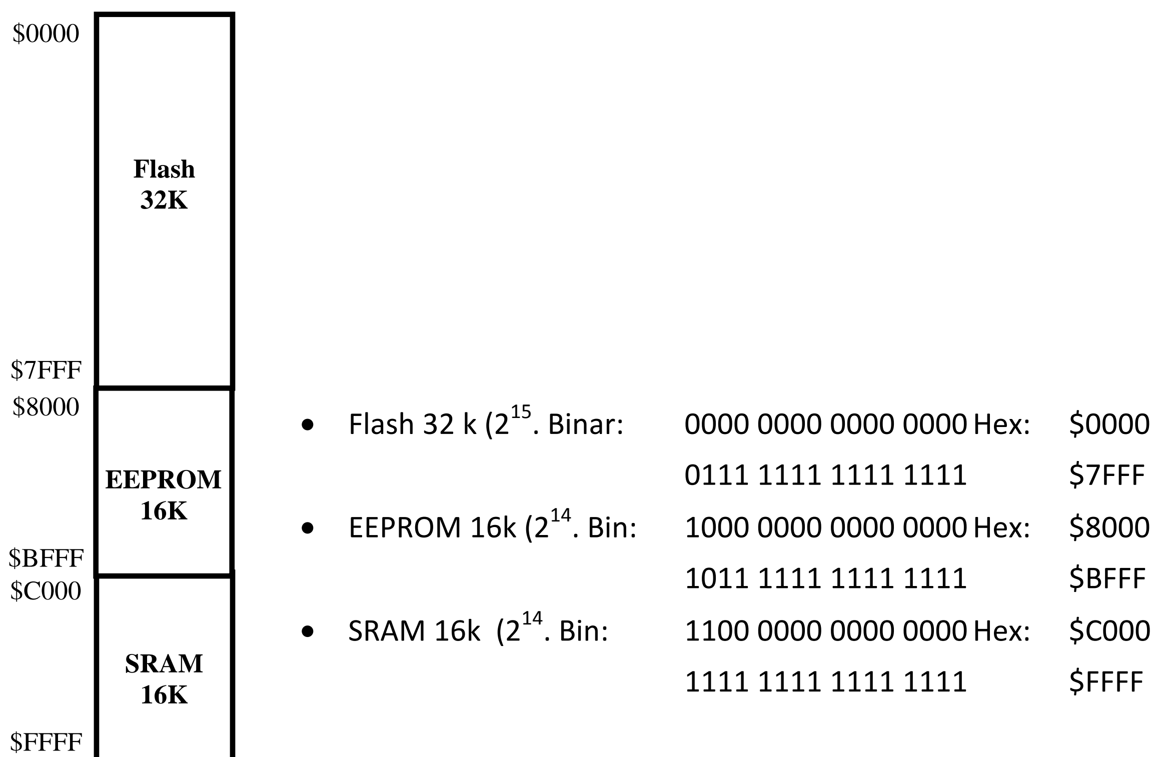
Pentru a înţelege modul în care s-au trecut adresele locaţiilor în hexa, se indică modul de codificare din binar în hexazecimal:

- Flash 32 k (2^{15}) Binar: 0000 0000 0000 0000 Hex: $0000
  0111 1111 1111 1111 $7FFF
- EEPROM 16k (2^{14}) Binar: 0000 0000 0000 0000 Hex: $0000
  0011 1111 1111 1111 $3FFF
- SRAM 16k (2^{14}) Binar: 0000 0000 0000 0000 Hex: $0000
  0011 1111 1111 1111 $3FFF

b) Spaţiu comun de adresare, indiferent de tipul de memorie

La aceste microcontrolere tot spaţiul de adresare este comun fiecare din tipurile de memorie având adrese specifice de acces. În figura 8.3 se prezintă un exemplu al acestui mod de organizare şi adresare, considerând aceleaşi tipuri de memorie şi aceleşi dimensiuni ca în figura 8.2. Pentru aceste microcontrolere, de cele mai multe ori se reyervă spaţiu şi pentru eventuală memorie conectată în exterior. Această metodă de adresare, deşi este utilizată la multe

microcontrolere, este mai puţin sigură pentru că se poate cere acces la o adresă greşită, de exemplu scriere în memoria flash de program, ceea ce nu se va executa.



Figură 8.3: Diferite tipuri de memorie organizate într-un singur domeniu de adresare.

8.2.3. Organizarea sistemului de I/O

La interfeţele pentru transfer al datelor de intrare/ieşire pot exista: intrări / ieşiri digitale şi intrări / ieşiri analogice. Multe dintre terminalele microcontrolerului sunt multiplexate, pentru că fiecare terminal poate avea între două şi patru funcţii diferite. De aceea pentru setarea funcţiei specifice utilizate pentru un terminal, programatorul trebuie să configureze *registre speciale de configurare* pentru terminalele respective. În general aceste registre de configurare sunt volatile şi se înscriu prin program în faza de iniţializare a sistemului. Pentru fiecare familie de microcontrolere există o aplicaţie software specifică, numită mediu integrat de dezvoltare a programelor (IDE = Integrated Development Environment) care pe lângă alte funcţii de sprijinere a programatorului (editare program sursă, compilare, asamblare, editor de legături, depanator) permite şi configurarea sub formă grafică a tuturor registrelor de configurare.

Intrările / ieşirile digitale sunt cuplate în porturi, cu dimensiunea tradiţională de 8 biţi. Prin registrele de configurare specifice porturilor se pot selecta funcţiile (acolo unde terminalele sunt

multiplexate pentru mai multe funcţii) şi sensul de circulaţie a informaţiei digitale (intrare/ieşire). Nu există restricţii în ceea ce priveşte setarea unor terminale ca ieşiri şi a altora ca intrări, la acelaşi port. Registrele tipice de configurare pentru porturile de I/O digitale sunt următoarele:

- Registrul de direcţie a datelor (îl notăm cu DDR = Data Direction Register, deşi sunt folosite şi alte prescurtări). Fiecare port bidirecţional are un DDR cu alocare de 1 bit/pin. După un semnal de *Reset* biţii din registrul DDR trec automat ca intrări pentru a preîntâmpina distrugerea terminalelor configurate ca ieşiri, din cauza logicii externe conectate la terminal.
- Registru port de ieşire (îl notăm cu PORT deşi sunt folosite şi alte prescurtări) - utilizat pentru a controla nivelul tensiunii la pinii de ieşire (JOS sau SUS). La multe microcontrolere, SUS corespunde la scrierea lui 0 în bitul corespunzător. Pentru a nu scrie toţi biţii portului atunci când se setează un anume bit se pot folosi Instrucţiuni de prelucrare pe bit, sau dacă acestea nu există în setul de instrucţiuni, o combinaţie de operaţii ce presupune citire port, modficare octet prin mascare şi apoi scriere. Descrierea celor două variante este exemplificată (exemplul 2) mai jos.
- Registru port de ieşire (îl notăm cu PIN, deşi la diversele microcontrolere sunt folosite şi alte prescurtări) - este în general un registru doar cu citire şi conţine starea curentă a tuturor pinilor indiferent dacă sunt configuraţi ca intrări sau ca ieşiri. Este utilizat pentru a citi starea pinilor de intrare.

Dacă registrele de configurare descrise pe scurt mai sus, există la toate microcontrolerele, la unele familii există şi registre suplimentare de control. De exemplu:

- poate exista registru de selecţie pentru pini de intrare cu funcţie analogică / digitală (ANSEL la PIC)
- registru de control al rezistenţelor interne de pull-up (WPUx la PIC)
- registru care permite unui pin să genereze o întrerupere, dacă este configurat ca intrare

Atunci când programatorul selectează intrările / ieşirile sistemului, în cazul în care se utilizează doar intrări şi ieşiri digitale, oricare pin de port poate fi configurat ca intrare sau ieşire. Dacă se utilizează şi intrări analogice (la terminale-pini cu funcţii multiplexate) aceste terminale şi porturi vor fi rezervate pentru funcţia analogică, pentru că nu toate porturile au capabilitatea de citire a semnalelor analogice. În plus, dacă există intrări comandate prin comutator mecanic, se vor utiliza acele porturi care au rezistenţe de pull-up (trage nivel sus) interne, pentru că altfel aceste rezistenţe trebuie conectate în exterior.

Figurile următoare (8.4, 8.5 şi 8.6) indică schematic structura electrică generală a unui terminal de I/O de port, în diverse situaţii: la alimentarea circuitului, la scriere, la citire, cu

comutatoare electronice ce simbolizează comutările realizate prin registrele de configurare a portului.

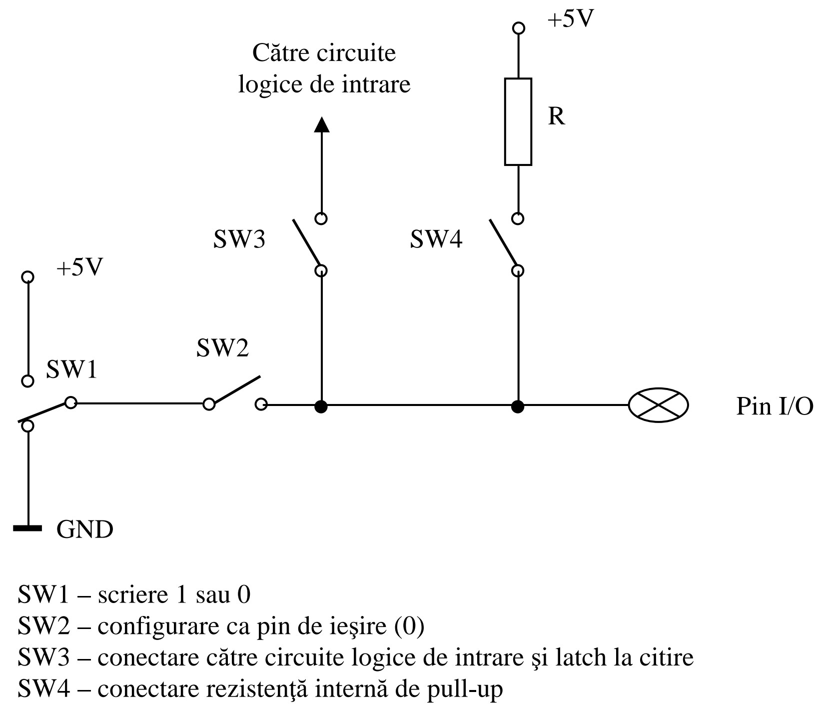


Figura 8.4. Structura electrică de principiu a unui pin de I/O la alimentarea microcontrolerului

Figura 8.5. ilustrează un port configurat ca ieşire şi programat să scrie nivel 1 logic (HIGH) către exterior.

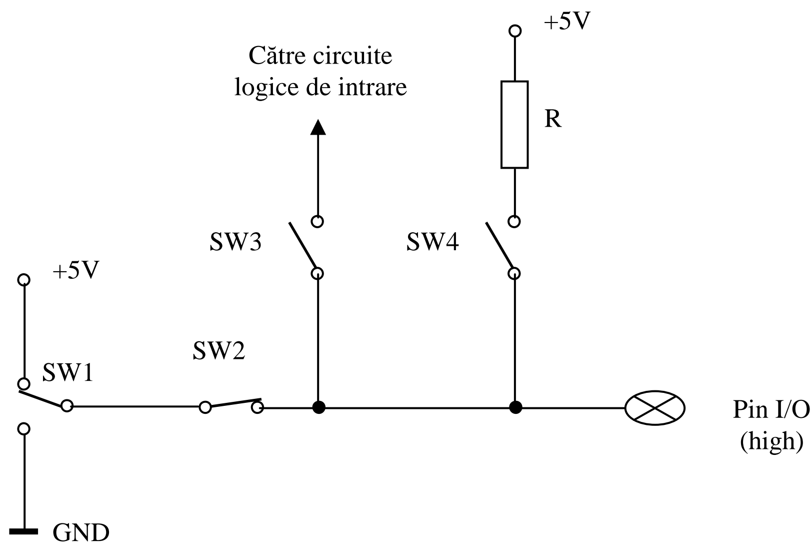


Figura 8.5. Structura electrică de principiu a unui pin de ieşire şi scriere nivel HIGH (1 logic)

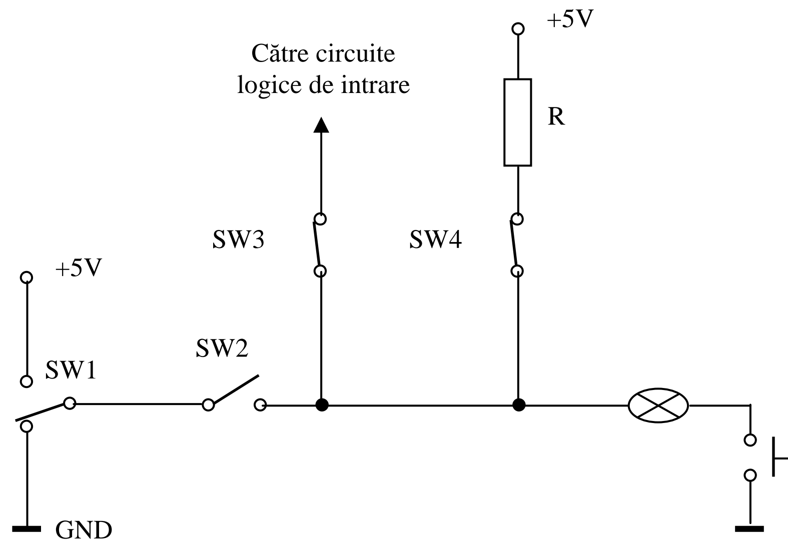


Figura 8.6. Structura electrică de principiu a unui pin de intrare, cu citire nivel logic LOW (0 logic) la închiderea comutatorului extern. Se observă conectarea rezistenţei obligatorii de "pull-up", pentru fixarea nivelului logic la pin atunci când comutatorul este deschis

Exemplul 1: Exemplu de succesiune de operaţii de configurare a registrului port X, ca ieşire, cu 1 şi 0 transmişi alternativ, considerând că lucrăm cu pin corespunzător bitului 0:

1. SetPortDirection() ;DDRX=0b00000000, toţi pinii portului X pe output
2. PORTX=0b00000001 ; scrie 1 la bit 0
3. delay_ms(500) ; întârziere de 500 ms
4. PORTX=0b00000000 ; scrie 0 la bit 0
5. delay_ms(500) ; întârziere de 500 ms
6. Salt la pas 2

Exemplul 2: Se doreşte citirea valorii bitului 3 a portului A prin variabila x. Dacă microcontrolerul are instrucţiuni de manipulare pe bit vom scrie: *x=PINA.3*. Dacă microcontrolerul nu are instrucţiuni de manipulare pe bit vom citi portul, modificăm valoarea citită prin mască şi rescriem la port. Succesiunea de operaţii se poate scrie *x=PINA&0b00001000* , unde simbolul "&" reprezintă funcţia logică ŞI. Ca urmare x va fi 0 dacă PINA.3 este setat jos şi va avea o valoare diferită de zero dacă PINA.3 este setat sus.

Exemplul 3: Pentru ieşire la PORT D, după setarea pinul ca ieşire, să se transmită 1 la pinul 5 al portului D. Dacă microcontrolerul are instrucţiuni de manipulare pe bit vom scrie: *PORTD.5=1*. Dacă microcontrolerul nu are instrucţiuni de manipulare pe bit vom citi portul, modificăm

valoarea citită prin mască şi rescriem la port. Succesiunea de operaţii se poate scrie *PORTD=PORTD|0b00100000*, unde simbolul "|" reprezintă funcţia logică SAU.

Intrări digitale

Funcţionalitatea intrărilor digitale e utilizată ori de câte ori se monitorizează un semnal ce va fi interpretat ca digital. Asta înseamnă că semnalul se poate schimba doar între două stări: SUS (corespunzător lui logic 1) şi JOS. Interpretarea nivelelor ca SUS sau JOS depinde de nivelul tensiunilor conform specificaţiilor microcontrolerului, care la rândul lor depind de tensiunea la care funcţionează microcontrolerul.

De exemplu la Atmega16 tensiunea Vcc trebuie să fie în intervalul 4,5 – 5,5 V, iar tensiunea JOS la intrare e recunoscută între [-0,5 … 0,2×Vcc] volţi pe când nivelul SUS al tensiunilor de intrare trebuie să fie în intervalul [0,6×Vcc … Vcc+0,5] volţi. Astfel că rămâne un interval ce cuprinde valorile [0,2×Vcc … 0,6×Vcc] volţi între care se spune că semnalul de intrare este nedefinit.

Pentru că semnalul digital de intrare poate fi doar o valoare de tensiune, se pune problema cum se transformă această valoare în formă binară în cadrul unui registru. Ca primă soluţie putem folosi elemente de memorare simple pentru registrul PIN pentru a memora starea curentă a fiecărui terminal de intrare în registru. Dacă elementul de memorare este triggerat de ceasul sistemului, el va stoca starea curentă a pin-ului în registru. Dacă latch-ul este triggerat cu clock-ul sistem el va stoca starea doar la începutul fiecărui ciclu de ceas. Pentru că eşantionăm *doar cu granularitatea ceasului*, înseamnă că recunoaştem schimbarea stării doar cu întârzierea unui impuls de ceas. Putem chiar să pierdem schimbări mai rapide decât impulsul de ceas (ca în exemplul din figura 8.7).

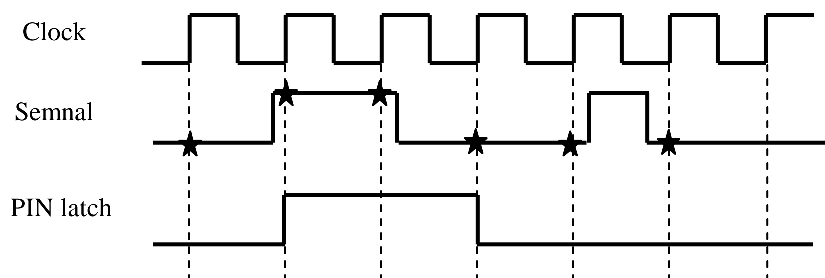


Figura 8.7. Exemplu privind eşantionarea digitală a semnalului de intrare la începutul fiecărei perioade de ceas

Chiar dacă metoda simplă de citire este utilă pentru multe dintre intrările digitale, adesea în sistemele ce lucrează în timp real se întâlnesc şi situaţii diferite. De exemplu se citeşte un semnal de intrare digital la care tranziţia se face foarte lent, sau peste care se suprapune zgomot. Pentru a scădea probabilitatea de citire tensiune în starea nedefinită, sau de oscilaţie circuitul folosit ca intrare include un *trigger Schmitt* pentru a rezulta fronturi bine definite ale semnalului de intrare.

Trigger Schmitt

Circuitele trigger Schmitt au două praguri de basculare V_{lo} şi V_{hi}, $V_{lo}<V_{hi}$, . Trecerea ieşirii din logic 0 în 1 se produce doar dacă semnalul de intrare depăşeşte V_{hi}. Trecerea ieşirii din logic 1 în 0 se produce doar dacă semnalul de intrare scade sub V_{lo}. Ca urmare circuitul nu transferă către ieşirea digitală fluctuaţii mici ale tensiunii de intrare, ieşirea având în permanenţă timpi scurţi de tranziţie crescătoare şi descrescătoare, indiferent de semnalul de intrare. Durata tranziţiilor este constantă iar valoarea exactă depinde de parametrii circuitelor folosite pentru implementare.

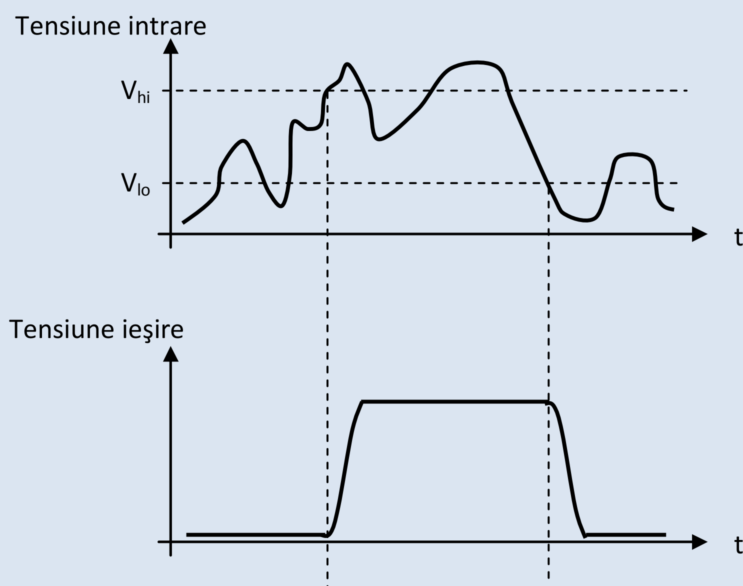


Figura 8.8. Intrarea şi ieşirea unui trigger Schmitt

Aşa cum s-a văzut în figura 8.6 microcontrolerele folosesc rezistenţe pentru forţarea nivelului logic la intrările digitale care ar pot fi lăsate „în aer” de circuitele externe. O intrare flotantă este susceptibilă la zgomot. Rezistorul respectiv va fi numit, în funcţie de tragere nivel la +V, sau la masă, rezistor de *pull-up (trage sus)* sau *pull-down (trage jos)*. Rolul este *conectarea pinului de intrare la o tensiune definită ca valoare, dacă pinul nu este controlat de hardware extern*

către un anumit nivel. Rezistoarele interne microcontrolerelor sunt controlate printr-un registru de configurare ce poate controla fiecare pin. Cele mai multe dintre microcontrolere au doar rezistenţe interne trage-sus ce pot fi controlate prin programarea registrelor de configurare, dar există şi microcontrolere ce au rezistenţe interne de trage – jos (de ex. HCS12).

Ieşiri digitale

Ieşirile digitale controlează logică din lumea externă prin nivelurile de tensiune SUS şi JOS transmise prin program. Legat de eşantionarea digitală şi rezistenţe de pull-up lucrurile sunt similare cu intrările digitale. Semnalul digital transmis către exterior are forma corectă, deci nu este nevoie de formare suplimentară a semnalului, aşa cum se întâmplă uneori la intrările digitale. Nivelurile de tensiune la ieşirile digitale sunt uşor diferite faţă de intrări. De exemplu pentru ATmega16 alimentat la V_{CC}= 5V, tensiunea maximă de ieşire pentru nivel JOS este tipic 0.7 V (faţă de 1V la intrări), iar tensiunea minimă la ieşire pentru nivel SUS este 4.2 V (faţă de 3V la intrări).
Ieşirile digitale sunt mai critice decât intrările pentru că dacă pe un terminal de ieşire se generează nivel SUS, iar logica externă leagă terminalul la masă scurt-circuitul poate distruge circuitul.

Intrări / ieşiri analogice

Aceste tipuri de interfeţe analogice sunt extrem de diverse pentru fiecare tip de microcontroler. Din punctul de vedere al intrărilor analogice, în prezent, majoritatea microcontrolerelor conţin un *convertor analog/digital* (ADC) şi un comparator analogic. Conversia digital /analogă este mai rar implementată pe circuitul integrat, pentru că un semnal analogic poate fi obţinut prin conectarea de filtre trece-jos la ieşirile digitale (de exemplu la ieşirea controlerului pentru modulaţia în lăţime a impulsurilor - PWM).

În cazul *comparatoarelor analogice*, dacă există, acestea constituie o cale uşoară de comparare a două valori de tensiune (exterioare, sau o tensiune exterioară ca intrare comparată cu o tensiune de referinţă generată intern). Unele microcontrolere conţin două sau mai multe comparatoare analogice. Aceste comparatoare sunt amplificatoare integrate, fără reacţie având astfel un câştig foarte mare. Când tensiunea la intrarea neinversoare este mai mare decât cea de la intrarea inversoare, tensiunea la ieşire se fixează la valoarea +V de alimentare a circuitului. Dacă relaţia dintre tensiunile de la intrare este inversă, ieşirea se va găsi la tensiune 0. Prin registrele de configurare specifice comparatorului se poate selecta sursa tensiunilor pe cele două intrări, se poate seta un nivel de tensiune de referinţă (la unele microcontrolere in 64 de niveluri sub valoarea tensiunii de alimentare) şi modul în care se comportă comparatorul. Ieşirea sa nu este accesibilă la pini, dar valoarea logică poate fi citită, sau dacă se configurează pentru întreruperi, poate genera întrerupere pe fronturile tranziţiei la ieşire, sau pe palier de tensiune la ieşire.

8.2.4. Utilizarea circuitelor Counter - Timer

Toate microcontrolerele conţin circuite counter - timer (în traducere contor - temporizator) extrem de utile în multe dintre aplicaţii. La nivel hardware ambele funcţii (contor, sau temporizator) sunt implementate cu numărătoare digitale de 8 sau mai mulţi biţi (multiplu de 8). Denumirea diferită indică faptul că circuitul fie poate să înregistreze succesiune regulată de impulsuri de ceas (funţie de timer), fie poate să contorizeze impulsuri de la evenimente neregulate (funcţie de counter), dar în continuare vom apela circuitele cu denumirea de timer, specificând explicit termenul de contor, doar acolo unde este cazul. Trebuie înţeles că aceste circuite timer / counter oferă suport hardware pentru operaţiile efectuate, spre deosebire de temporizările implementate software, prin rutine cu funcţii de temporizare.

Un timer / counter constă dintr-un numărător digital de n biţi (8, 16, 32 de biţie) şi circuite asociate funcţionării, controlate prin registre de configurare specifice: divizor de prescalare, prescaler, registru de captare a valorii curente din timer, registre de fixare a valorii de iniţializare şi comparare cu valoarea numărată, registre de configurare a modurilor de funcţionare. Dimensiunea circuitului timer nu este legată în general de dimensiunea magistralei de date a microcontrolerului, pentru că, de exemplu, există microcontrolere pe 8 biţi sau pe 32 de biţi, care folosesc timere pe 16 biţi.

Divizoarele de prescalare, asociate circuitelor timer realizează o divizare a frecvenţei ceasului de la intrarea lor, cu o valoare întreagă între 1 şi 2^n, unde n este numărul de biţi ai numărătorului. De notat că unele microcontrolere permit prescalare doar prin valori egale cu puteri ale lui 2. Ceasul rezultat la ieşirea divizorului este cel care controlează circuitul timer propriu-zis. Scopul prescalării este să se crească domeniul de numărare (temporizare ca număr întreg de ciclii de ceas de intrare). Cu cât încercăm să creştem domeniul de numărare al unui temporizator, cu atât vom pierde în precizia cu care măsurăm intervalele de timp. De exemplu dacă folosim un timer pe 16 biţi, iar perioada ceasului este de 1 microsecundă, valoarea maximă ce poate fi măsurată este de 65536 microsecunde = 65,5 milisecunde (2^{16}), care poate fi crescută doar prin prescalarea ceasului sau prin mărirea dimensiunii, în număr de biţi a, timerului.

Registrele asociate circuitului timer permit captarea valorii curente din timer, de exemplu, atunci când testarea valorii contorizate se face prin interogare, dar de asemenea permit scrierea dinamică a valorii binare de la care se doreşte să înceapă numărarea. Când valoarea din timer depăşeşte lungimea cuvântului microcontrolerului (de exemplu utilizare de contor pe 16 biţii pe un microcontroler de 8 biţi) accesul la valoarea contorizată se face în doi paşi succesivi: citirea începe obligatoriu cu octetul care se schimbă cel mai repede, adică octetul cel mai puţin

semnificativ (LSB), iar apoi se citeşte octetul mai semnificativ (MSB). Din aceleaşi motive la scriere, ordinea este inversă: se va înscrie mai întâi MSB şi apoi LSB.

Aplicaţiile la care pot fi folosite circuitele timer sunt extrem de diverse şi, se spune că numărul de aplicaţii ţine de experienţa şi inspiraţia programatorului. Câteva exemple: controlul luminozității unui LED, recepţia de date de la un senzor digital care transmite în PWM (Pulse Width Modulation), introducerea de întârzieri cu valoare programată, măsurare intervale de timp, generare forme de undă digitală cu frecveţă variabilă şi factor de umplere variabil (de exemplu pentru ton taste sau avertizări sonore), generare de impulsuri de declanşare a unor evenimente, numărare evenimente externe sau interne, determinarea momentului de timp al unui eveniment, controlul acţionărilor electrice etc.

Prin registrele de configurare se poate comanda direcţia de numărare (incrementare / decrementare a conţinutului), selecţia frontului activ al ceasului, modul specific de funcţionare (de exemplu funcţionare modulo - după depşirea capacităţii de numărare se reia numărarea de la o valoare prefixată, cascadare de timere pentru funcţionare pe lungime dublă), programarea sursei de ceas interne sau externe, generarea de întrerupere la producerea de overflow (trecere prin valoare maximă la incrementare, sau trecere prin zero la decrementare).

Un circuit timer special inclus în microcontrolere este cel numit w*atchdog timer* (timer de supraveghere). Este un timer folosit pentru revenirea din situaţii dificile (probleme de software, ca de exemplu bucle infinite, sau probleme de hardware nepermanente care blochează funcţionarea corectă a programului). Acest timer este programabil pentru a număra un interval de timp după care trebuie să acţioneze. Dacă intervalul de timp programat s-a scurs, fără ca watchdog timerul să fie resetat explicit prin instrucţiune, acesta va activa intrarea de RESET a microcontrolerului. Watchdog timer ajută sistemul să iasă din bucla infinită, sau din aşteptarea la infinit a unui eveniment extern, care a produs blocarea programului.

O altă funcţie specială a circuitelor counter / timer este cea de controler PWM (Pulse Width Modulator). De exemplu dacă microcontrolerul conţine 2 canale independente pentru generarea de impulsuri modulate, de durate şi perioade programabile, funcţionarea poate fi descrisă ca în figura 8.9. Divizorul de prescalare şi numărătorul sunt comune ambelor canale şi se consideră ca au lungimea de 8 biţi. Secvenţa de numărare este 0 - 254. Conţinutul numărătorului este comparat cu conţinutul a 2 registre, PWM0 şi PWM1. Dacă conţinutul registrelor este mai mare decât cel al numărătorului, ieşirea corespunzătoare va avea valoarea 0 iar în caz contrar va avea valoarea 1. Ca urmare forma de undă a ieşirii digitale este controlată de factorul de prescalare şi de conţinuturile registrelor. Programul utilizator poate modifica dinamic conţinutul registrelor PWM0 şi PWM1, pentru a modifica factorul de umplere al fiecărui canal.

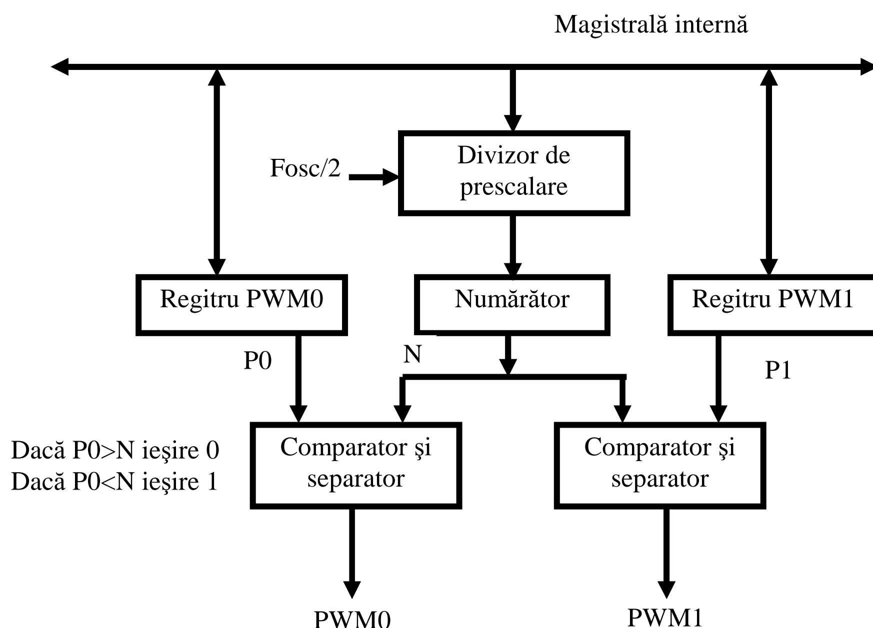


Figura 8.9. Explicativă la circuitele de generare a ieşirilor cu factor de umplere programabil, pentru două canale PWM

Controlerul PWM este cel folosit pentru comanda vitezei de rotaţie a motoarelor electrice, dar poate avea şi alte utilizări, cum ar fi generarea de forme de undă analogice prin conversie digital analogă (prin adăugarea unui filtru trece jos în exterior), sau comanda reglabilă a intensităţii luminoase printr-un LED.

Ceasul ce acţionează un circuit timer / counter poate proveni din mai multe *surse*:

- Ceas sistem intern
- Ceas cu prescalare
- Impulsuri externe (Pulse Accumulator)
- Semnal de la cuarţ extern (Mod asincron)

În cazul utilizării *ceasului sistem*, se foloseşte frecvenţa ceasului de la microcontroler. Circuitul timer este incrementat la fiecare impuls al ceasului sistem. În general acesta este modul implicit de setare pentru sursa semnalului de ceas. Pentru varianta ceas cu *prescalare* se utilizează de asemenea ceasul sistem, divizat printr-un contor de prescalare care este incrementat de către ceasul sistem. Aşa cum s-a mai menţionat circuitul timer cu prescalare permite măsurarea unor perioade mai mari de timp, faţă de utilizarea directă a ceasului sistem, dar precizia scade. Sursa ceasului numită *impuls extern* se referă situaţia când circuitul timer primeşte ceas de declanşare de la un semnal digital extern conectat la un pin de intrare al MC. Pentru că semnalul extern este eşantionat ca orice alt semnal extern, timpul dintre fronturile active trebuie să fie mai mare decât

perioada ceasului sistem. Ultimul tip de sursă de ceas, cu *semnal de la un cuarţ independent* de cel al microprocesorului conduce la denumirea surprinzătoare (e cu ceas!) folosită adesea: mod asincron, dar asincronismul se referă la cele două ceasuri utilizate. În acest caz circuitul timer este activat cu ceas de la un cuarţ extern conectat la doi pini de intrare ai MC, cu frecvenţa de ceas de 32.768 kHz care poate fi utilizată pentru implementarea ceasului de timp real (RTC - real-time clock). RTC măsoară timpul în unităţile folosite de utilizatorul uman: secunde, minute, ore. Contorul intern este incrementat conform semnalului extern şi lucrează asincron faţă de restul circuitelor microcontrolerului.

8.2.5. Sistemul de întreruperi

Sistemul de întreruperi este organizat asemănător cu sistemul de servire al cererilor de întrerupere harware descris pentru microprocesorul de uz general. Aici se vor puncta doar câteva elemente specifice microcontrolerelor.

Există doi biţi principali pentru logica de întreruperi a microcontrolerului:

- Bitul de interrupt enable (IE), setabil de aplicaţia utilizator pentru a indica că la apariţia cererii de întrerupere, dacă aceasta este recunoscută,microcontrelerul va apela o rutină de tratare a întreruperii (RTI).
- Bitul de interrupt flag (IF), setat automat de microcontroler atunci când s-a produs evenimentul de întrerupere. Acest indicator poate fi resetat automat la saltul către RTI, sau resetat prin program. Practic IF indică faptul că s-a produs evenimentul de întrerupere, în timp ce IE poate permite servirea respectivului eveniment.

Pentru microcontrolere există un bit de validare / invalidare globală a cererilor de întrerupere (IE global). De asemenea, fiecare sursă, sau tip de sursă de întreruperi are biţi IE şi IF alocaţi. Dacă mai multe surse similare de întreruperi sunt mapate către un singur bit IE, se alocă câte un bit IF fiecărei surse de întreruperi, pentru a se putea determina de unde vine cererea de întrerupere. IF se setează când apare evenimentul chiar dacă o întrerupere este invalidată prin fanionul său de IE. Microcontrolerul conţine registre speciale de configurare, pentru setarea modului de întreruperi ("interrupt mode") al unora din sursele de întrerupere. Modul de întreruperi fixează şi schimbarea specifică a semnalului de la intrarea de cerere de întrerupere; de exemplu: frontul crescător, palier etc.

Pentru calculul adreselor specifice fiecărei RTI microcontrolerul foloseşte o tabelă a vectorilor de întrerupere, cu funcţionare similară cu înteruperile vectorizate de la microprocesoare de uz general. Tabela vectorilor de întrerupere (TVI) conţine câte o intrare pentru fiecare vector de întrerupere şi are de obicei poziţie fixă în memoria de program. Fiecare vector are o adresă pre-fixată în TVI , care este referită faţă de o adresă de bază fixă în memoria

de program. La adresa vectorului, aplicaţia înscrie fie adresa de început a RTI, fie o instrucţiune de salt către RTI (cum se întâmplă la ATmega16).

Sistemul de întreruperi exte extrem de util pentru programarea aplicaţiilor cu calculator încorporat, dar de multe ori programatorii utilizează în exces întreruperile. O întrerupere serveşte lucruri ce nu pot aştepta să fie preluate prin program de către microprocesor. Uneori sunt folosite doar pentru a reduce complexitatea hardware. Dar întotdeauna numărul mare de întreruperi utilizate va produce creşterea timpului de depanare, iar întreruperile multiple au un mare potenţial de erori greu de descoperit. Întreruperile sunt utilizate pentru a atenţiona procesorul despre evenimente speciale cum ar fi un timer care a terminat perioada impusă de temporizare sau o componentă hardware care necesită atenţie. La numărarea evenimentelor ce sunt folosite pe întreruperi trebuie ţinut cont şi de sursele interne de întrerupere.

8.2.6. Sistemul de management al puterii

Puterea consumată este probabil cel mai important element pentru sistemele cu calculator încorporat. Multe din sisteme au constrângeri de putere (în special cele alimentate la baterii) şi ele nu-şi permit utilizarea de ventilatoare sau alte sisteme de răcire.

La tehnologia CMOS puterea disipată este produsă în special de comutaţia circuitelor logice (circa 90% din puterea consumată). Puterea de comutaţie poate fi exprimată ca: $P_{comutare} = C \cdot V_{dd}^2 \cdot f$, unde: C este capacitatea parazită, V_{dd} tensiunea de alimentare, iar f frecvenţa de comutaţie.

Ca urmare, principalele metode disponibile utilizatorului pentru reducerea puterii consumate sunt:

a. reducerea nivelului de tensiune s-a ajuns la valori sub 1.2V. Pentru că puterea de comutaţie este proporţională cu pătratul tensiunii, de exemplu scăzând tensiunea de alimentare de la 5V la 3,3 V se produce o scădere a consumului de putere de $5^2/3{,}3^2$ = 2,29 ori
b. modificarea frecvenţei de ceas. Prin program se poate controla frecvenţa prin programarea diferitelor divizoare de ceas. Microcontrolerele sunt proiectate în general într-o tehnologie HCMOS statică, ceea ce înseamnă că frecvenţa de ceas poate fi modificată între zero şi o valoare maximă. Spre deosebire, cele mai multe microprocesoare de uz general nu pot funcţiona corect sub o anumită valoare a frecvenţei de ceas.
c. dezactivarea unităţilor funcţionale nefolosite - dacă ele nu sunt utilizate în timpul unui ciclu, ele pot fi complet închise, pentru scăderea .consumului.
d. utilizarea facilităţilor de management a puterii, incluse în microcontrolere. De exemplu

UCP se poate găsi într-unul din următoarele moduri:

- complet operaţional (fully operational)- semnalul de ceas se propagă la întregul procesor
- mod rezervă, în aşteptare (standby mode) - procesorul nu execută instrucţiuni, dar întreaga informaţie stocată este disponibilă şi se poate întoarce la modul complet operaţional în câteva cicluri.
- mod clock oprit (clock-off mode) - semnalul de ceas este oprit complet; pentru a ieşi din acest mod sistemul trebuie restartat.

În figura 8.10. se exemplifică modul de economisire al energiei la microprocesorul StrongARM SA-1100. Procesorul are două tensiuni de alimentare: alimentare principală la 3.3V şi auxiliară la 1.5V. De asemenea permite trei moduri de putere: rulare (run - mod normal de funcţionare), inactiv (idle - se opreşte ceasul UCP, dar circuitele logice sunt încă alimentate) şi adormit (sleep - dezactivează cea mai mare parte a activităţii circuitului). Se observă durata mare a timpului cerut pentru revenire din starea de foarte mică putere sleep în starea de run.

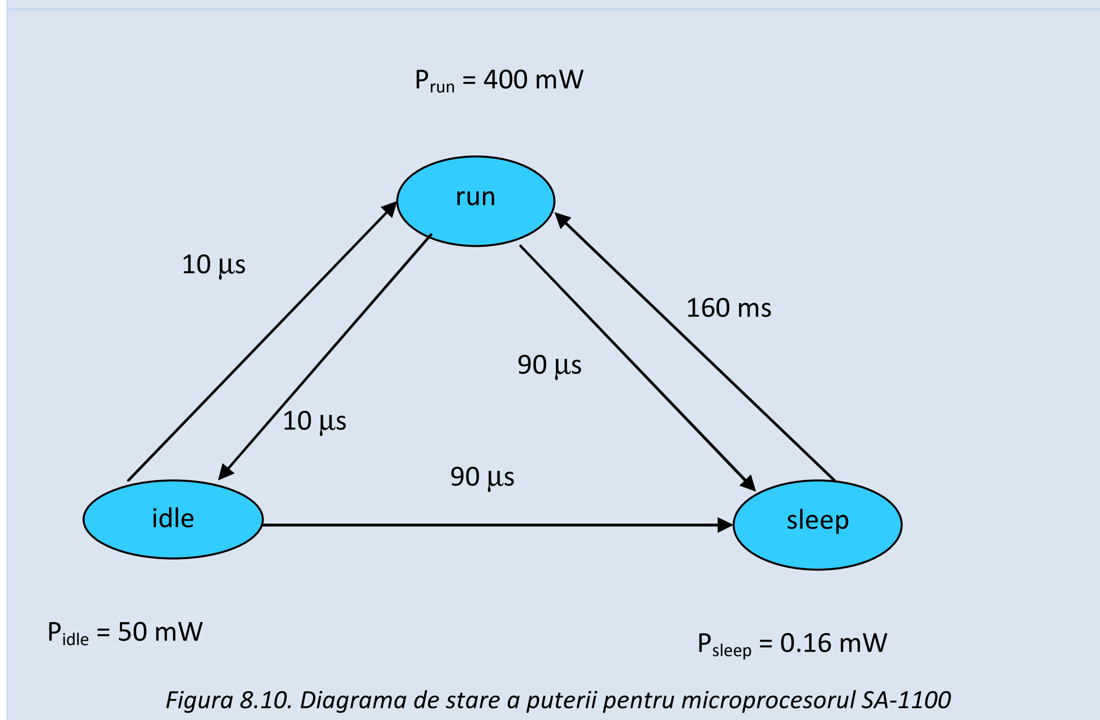


Figura 8.10. Diagrama de stare a puterii pentru microprocesorul SA-1100

Alţi factori care influenţează consumul de putere sunt magistralele (în mod specific tranziţiile pe magistrale) şi dimensiunea memoriei (dacă este o memorie dinamică în plus este necesară şi reîmprospătarea). Pentru a minimiza consumul de putere, magistralele şi memoriile sun ţinute atât de mici cât poate accepta aplicaţia.

8.3. Exerciţii

1. Care este înţelesul şi rolul stării *High Impedance (HiZ) state*, la pinii de ieşire dintr-un microprocesor
2. Ce este o rezistenţă de trage - sus la un pin al microcontrolerului şi care este rolul său
3. Explicaţi de ce la unele intrări digitale de microcontroler se impune utilizarea unui circuit de tip trigger-Schmitt
4. Circuite timer - counter, structuri şi aplicaţii tipice (enumerare minim 5 aplicaţii).
5. Ce este un watchdog timer (rol şi funcţionare)?
6. Descrieţi modul de funcţionare al unui controller PWM
7. Descrieţi funcţiile registrelor tipice de configurare pentru porturile digitale de IO ale microcontrolerelor
8. Trataţi pe scurt problemele ce apar la MC privind consumul de putere şi metodele folosite pentru managementul puterii.
9. Definiţi soluţia pentru realizarea unei întârzieri de 1 secundă, dacă se lucrează cu un microcontroler cu frecvenţa de ceas de 1 MHz şi aveţi la dispoziţie un timer pe 16 biţi.

Prescurtări utilizate (în ordine alfabetică)

- ADC = convertor analog/digital (Analog to Digital Converter)
- AE = Adresa efectivă
- ALU = Unitate aritmetică şi logică (Arithmetic and Logic Unit)
- ASI = arhitectura setului de instrucţiuni
- BIOS = Sistem de bază de intrare / ieşire (Basic Input Output System)
- CI = circuit integrat
- CISC = Calculator cu set complex de instrucţiuni (Complex Instruction Set Computer)
- CPI = număr de ciclii de ceas pentru o instrucţiune (Clock cycles Per Instruction)
- CS = Chip Select (selecţie circuit)
- DAC = convertor digital / analog (Digital to Analog Converter)
- DMA = Acces direct la memorie (Direct Memory Access)
- DRAM = memorie dinamică cu acces aleator - interfaţă asincronă (Dynamic Random Access Memory)
- FIFO = Primul intrat, primul ieşit (First-In, First-Out)
- HiZ = stare de înaltă impedanţă
- I/O = sistem de Intrare / ieşire (Input/Output System)
- LIFO = Ultimul intrat, primul ieşit (Last-In, First-out)
- LSB = Cel mai puţin semnificativ Byte/octet (Least significant Byte)
- LSb = Cel mai puţin semnificativ bit (Least significant bit)
- MA = memoria auxiliară, sau externă
- MP = Memoria principală sau operativă
- MSB = Cel mai semnificativ Byte/octet (Most significant Byte)
- MSb = Cel mai semnificativ bit (Most significant bit)
- PC = Contor de program (Program Counter or Instruction Pointer)
- PWM = modulaţie în laţime a impulsurilor (Pulse Width Modulation)
- RAM = memorie cu acces aleator (Random Access Memory)
- RISC = Calculator cu set redus de instrucţiuni (Reduced Instruction Set Computer)
- ROM = memorie doar cu citire (Read Only Memory)
- RTI = Rutina de tratare a întreruperii
- RTC = Ceas de timp real (real-time clock).
- SO = sistem de operare
- UC = Unitate de control (a UCP)
- UCP = Unitatea Centrală de Procesare
- SP = pointer de stivă (Stack Pointer)
- SDRAM = memorie dinamică cu acces aleator - interfaţă sincronă (Synchronous Dynamic Random Access Memory)
- SRAM = memorie statică cu acces aleator (Static Random Access Memory)
- TSL = Trei stări logice (Three State Logic)

Bibliografie

1. [Borcoci95] Borcoci E., Zoican S., Popovici E., Arhitectura microprocesoarelor, partea I, Ed. Media Publishing, Bucureşti, 1995.
2. [Burileanu94] Burileanu,C., Arhitectura microprocesoarelor, Editura DENIX, Bucureşti, 1994;
3. [Crutu87] Crutu, Gh., Romanca, M., Fratu, A., Calculatoare, microsisteme de calcul, Universitatea din Brasov, 1987;
4. [Dodescu80] Dodescu, Gh., Ionescu, D., Misdolea, T., Nisipeanu, L., Pilat, F., Sisteme electronice de calcul şi teleprelucrare, Ed.Didactică şi Pedagogică, Bucuresti, 1980;
5. [Furht87] Furht, B., Milutinovic, V., A survey of microprocessor architecture for memory management, IEEE Computer, March 1987, vol.20, no3, pp. 48-66.
6. [Hayes88] Hayes, J., Computer Architecture and Organisation, McGrow Hill Comp., 1988.
7. [Lupu86] Lupu, C., s.a., Microprocesoare, Ed. Militară, Bucureşti 1986;
8. [Mano93] Mano, M., Computer System Architecture, Prentice-Hall Inc. 1993.
9. [MDE72] Mic dicţionar enciclopedic, Editura enciclopedică română, Bucureşti, 1972
10. [Nicula97] Nicula, D., Arhitecturi de microprocesoare de performanţe ridicate, Teză de doctorat, Universitatea TRANSILVANIA Braşov, 1997
11. [Patterson90] Patterson, D., Hennessy, J., Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers, Inc. 1990;
12. [Patterson12] Patterson, D., Hennessy, J., Computer Architecture A Quantitative Approach (fifth edition), Morgan Kaufmann Publishers, Inc. 2012, ISBN: 978-0-12-383872-8
13. [Patterson96] Patterson, D., Hennessy, J., Computer Architecture - A Quantitative Approach, second edition, Morgan Kaufmann Publishers, Inc. 1996;
14. [Patterson94] Patterson, D., Hennessy, J., Computer Organization & Design, the Hardware Software Interface, Morgan Kaufmann Publishers, Inc. 1994;
15. [Pfaffenberger96] Pfaffenberger, B., Dicţionar explicativ de calculatoare, Ed. Teora, Bucureşti, 1996
16. [Pop2000] http://www.ida.liu.se/~paupo, Slides for Advanced Computer Architecture, Paul Pop, Institutionen för Datavetenskap, Linköpings Universitet
17. [SPEC] The Standard Performance Evaluation Corporation, http://www.spec.org
18. [Stallings00] Stallings, W., Computer Organization and Architecture, 5th edition, Prentice Hall International, Inc., 2000.
19. [Stefan83] Ştefan, Gh., Drăghici, I., Mureşan, T., Barbu, E., Circuite integrate digitale, Ed.Didactică şi Pedagogică, Bucureşti 1983;

20. [Stefan91] Ştefan, Gh., Funcţie şi structură în sistemele digitale Ed. Academiei Române, 1991;
21. [Strugaru92] Strugaru, C., Popa, M., Microprocesoare pe 16 biţi, Editura TM, Timişoara 1992;
22. [Sztojanov87] Sztojanov, I., Borcoci, E., Tomescu, N., Bulik, D., Petrec, M., Petrec, C., De la poarta TTL la microprocesor, Ed.Tehnică, Bucureşti, 1987;
23. [Tanenbaum99] Tanenbaum, A., Organizarea structurată a calculatoarelor, ediţia a IV-a, Computer Press AGORA, Tg. Mureş, 1999.
24. [Toacse85] Toacse, Gh., Introducere în microprocesoare, Ed. Ştiinţifică şi Enciclopedică, 1985;
25. [Toacse96] Toacşe, Gh. Nicula, D. Electronică digitală, Ed. Teora, 1996
26. [Weems96] Weems, Ch. Jr., Computer Science Course 635, Notes from Lecture 3, at www.cs.umass.edu/~weems/index.hrml
27. [Wiki15] Wikipedia, Memory timings, https://en.wikipedia.org/wiki/Memory_timings
28. [EU 15-1] Digital agenda for Europe, https://ec.europa.eu/digital-agenda/en/embedded-systems
29. [EU 05] Alfred Helmerich, Nora Koch and Luis Mandel, Study of Worldwide Trends and R&D Programmes in Embedded Systems in View of Maximising the Impact of a Technology Platform in the Area, 2005, ftp://ftp.cordis.europa.eu/pub/ist/docs/embedded/final-study-181105_en.pdf
30. [Emb Prog] Introduction to embedded programming, http://www.scriptoriumdesigns.com/embedded/index.php
31. [Viena07] Gridling G., Weiss B., Introduction to Microcontrollers, Courses 182.064 & 182.074, Vienna University of Technology, Institute of Computer Engineering, Embedded Computing Systems Group, 2007