UNIVERSITATEA "AL. I. CUZA" IAŞI
Facultatea de Informatică
Cursuri Postuniversitare

Vlad Rădulescu
Henri Luchian
Adrian Buburuzan

ARHITECTURA CALCULATOARELOR

2005-2006

Adresa autorului: Universitatea "Al. I. Cuza"
Facultatea de Informatică
Str. Berthelot nr. 16
700483 Iaşi
România
e-mail: rvlad@infoiasi.ro
web home page: http://www.infoiasi.ro/~rvlad

1. Introducere

Deşi de-a lungul timpului au existat încercări de a realiza dispozitive capabile să realizeze în mod automat operaţii de calcul, abia începând aproximativ cu perioada celui de-al doilea război mondial se poate vorbi de un efort concertat şi direcţionat în acest sens. Calculatoarele, în forma în care se prezintă astăzi, îşi datorează în mare măsură existenţa rezultatelor obţinute în acea perioadă de către John von Neumann, Alan Turing şi Kurt Gödel.

Odată stabilite principiile de bază, ultima jumătate de secol a cunoscut un efort continuu de perfecţionare a tehnologiilor folosite în construcţia calculatoarelor. Din fericire, dezvoltarea explozivă a domeniului electronicii a permis o creştere exponenţială a puterii de calcul. Una dintre pesonalităţile domeniului, Gordon Moore (cofondator al companiei Intel), a enunţat în urmă cu circa 3 decenii legea care-i poartă numele şi care prevede că puterea sistemelor de calcul se dublează la fiecare 18 luni. Deşi este o lege empirică, bazată numai pe observaţii practice şi fără vreo fundamentare teoretică, experienţa i-a confirmat valabilitatea până în zilele noastre.

Ca urmare a acestei rate deosebite a progresului, calculatoarele au invadat practic toate domeniile de activitate. Astăzi nu mai există vreo ocupaţie care să nu beneficieze de pe urma utilizării tehnicii de calcul. O menţiune aparte trebuie făcută în legătură cu extinderea reţelelor de calculatoare până la apariţia Internetului, care astăzi permite accesul tuturor la un volum de informaţie nemaiîntâlnit în trecut.

În spatele tuturor acestor realizări impresionante stă munca depusă de specialiştii în domeniu. Un sistem de calcul are două părţi: *hardware* (circuitele fizice care îl compun) şi *software* (programele care rulează pe acel sistem). Pentru o funcţionare la parametrii optimi a calculatorului este necesară o bună conlucrare a celor două părţi. Cel mai performant hardware este inutil în absenţa programelor care să realizeze activităţile dorite. La rândul lor, programele au nevoie de hardware pe care să ruleze. Astfel, performanţele obţinute astăzi sunt posibile numai ca urmare a activităţii tuturor celor implicaţi în dezvoltarea echipamentelor şi scrierea programelor.

1.1. Elemente de bază

Activitatea principală a calculatorului, după cum o arată însuşi numele său, este aceea de a efectua calcule. Cu toate acestea, orice persoană care a lucrat cel puţin o dată cu un calculator înţelege imediat că o asemenea caracterizare este cu totul insuficientă pentru a descrie sarcinile îndeplinite de acesta. Într-adevăr, este greu de acceptat că semnificaţia unui program de grafică, de exemplu, s-ar reduce la nişte simple calcule, deşi aceste calcule au un rol cu adevărat foarte important. Într-o accepţiune mai generală, putem spune că funcţionarea unui calculator are ca principal obiectiv prelucrarea informaţiei. Pentru a înţelege modul în care este tratată informaţia, vom vedea mai întâi cum este ea reprezentată într-un calculator.

Cea mai mică unitate de informaţie folosită este *bitul*. Fără a intra în detalii, putem spune că un bit reprezintă o entitate, teoretică sau materială, care are două stări distincte posibile; evident, la un moment dat entitatea se poate afla într-una singură din cele două stări. Observăm deci că prin termenul de bit sunt desemnate atât conceptul teoretic, cât şi implementările sale fizice.

Au existat mai multe forme de implementare practică a biţilor. Cele mai eficiente soluţii s-au dovedit a fi cele bazate pe circulaţia curentului electric, acestea prezentând avantajul unei viteze de operare mult mai mare decât în cazul sistemelor

mecanice sau de altă natură. În circuitele electrice, cele două stări care definesc un bit sunt uşor de definit: putem asocia una dintre stări cu situaţia în care curentul electric străbate o porţiune de circuit, iar cealaltă stare cu situaţia în care curentul nu parcurge aceeaşi porţiune de circuit. (În practică, electroniştii preferă să discute despre cele două stări în termenii nivelelor de tensiune din circuit, dar ideea este de fapt aceeaşi). În timp au fost folosite dispozitive tot mai sofisticate, pornind de la comutatoare, coutinuând cu releele şi diodele şi ajungându-se astăzi la utilizarea tranzistorilor. Toate însă se bazează pe acelaşi principiu: permiterea trecerii curentului electric sau blocarea sa.

Întrucât, aşa cum am văzut mai sus, obiectivul urmărit este de a obţine circuite care să permită efectuarea de calcule, este necesar ca biţii să primească o semnificaţie numerică. Prin convenţie, celor două stări ale unui bit le sunt asociate valorile 0 şi respectiv 1. În acest mod, putem considera că lucrăm de fapt cu cifre în baza 2, iar calculele devin posibile.

O primă consecinţă a acestei abordări o constituie necesitatea grupării biţilor. Într-adevăr, o singură cifră, mai ales în baza 2, conţine prea puţină informaţie pentru a fi utilă. Deoarece în scrierea poziţională numerele sunt şiruri de cifre, apare imediat ideea de a reprezenta numerele prin şiruri de biţi. Deşi pentru om lucrul în baza 2 pare mai dificil, datorită obişnuinţei de a lucra în baza 10, în realitate nu există diferenţe conceptuale majore între diferitele baze de numeraţie.

(Putem răspunde aici unei întrebări care apare natural: de ce se preferă utilizarea biţilor, deci implicit a cifrelor în baza 2, dacă omul preferă baza 10? Răspunsul este de natură tehnologică: nu există o modalitate simplă de a realiza un dispozitiv cu 10 stări distincte, care să permită implementarea cifrelor în baza 10.)

Pe de altă parte, într-un sistem de calcul trebuie să existe o standardizare a dimensiunii şirurilor de biţi prin care sunt reprezentate numerele. Creierul uman se poate adapta pentru a aduna, de exemplu, un număr de 3 cifre cu unul de 6 cifre, iar apoi poate trece imediat la adunarea unui număr de 8 cifre cu unul de 9 cifre; un calculator însă, fiind format din circuite fără inteligenţă, nu poate fi atât de flexibil. Soluţia este de a permite ca şirurile de biţi să aibă numai anumite dimensiuni prestabilite. Astfel, circuitele din calculator se pot înţelege între ele, deoarece lucrează cu operanzi de aceeaşi dimensiune. Ajungem astfel la o altă unitate de informaţie larg folosită, şi anume *octetul* (în engleză *byte*). Acesta reprezintă un şir de 8 biţi şi se constituie într-un standard unanim respectat. Un octet poate avea $2^8 = 256$ valori diferite, ceea ce este evident insuficient pentru unele tipuri de informaţie vehiculate în calculator. Pentru a nu pierde avantajele standardizării, se permite ca operanzii să aibă şi dimensiuni mai mari, dar numai multipli de dimensiunea octetului; mai mult, aceşti multipli pot fi doar puteri ale lui 2. În funcţie de stadiile pe care le-a parcurs tehnologia de-a lungul timpului, dimensiunea maximă a operanzilor a fost de 16, 32 sau 64 biţi (respectiv 2, 4 sau 8 octeţi) şi fără îndoială va continua să crească. Această dimensiune poartă denumirea de *cuvânt*.

Dimensiunea unui circuit de memorie sau a unui disc hard este mult mai mare decât un octet. Pentru a putea exprima aceste dimensiuni s-au introdus denumiri pentru multipli, într-un mod similar celui din lumea ştiinţifică. Reamintim că prefixul kilo- (reprezentat prin simbolul K) desemnează un multiplu egal cu $1000=10^3$. În informatică se preferă exprimarea multiplilor în baza 2, care este cea utilizată în toate situaţiile. Astfel, prefixul kilo- are aici valoarea $2^{10}=1024$, care este foarte apropiată de 1000. Un kilooctet (sau kilobyte) se notează deci Ko (sau KB). Analog se definesc şi ceilalţi multipli: mega- ($1Mo=2^{20}$ octeţi), giga- ($1Go=2^{30}$ octeţi), tera- ($1To=2^{40}$ octeţi). Pot fi definiţi şi multipli cu ordine de mărime superioare, dar pe moment

practica nu face necesară utilizarea lor. După cum era de aşteptat, nu se definesc submultipli, care nu ar avea sens.

Facem observația că, spre deosebire de noțiunea de bit, a cărei definire are o solidă bază teoretică, octetul este doar un standard impus de practică. Nu există nici un motiv conceptual pentru a considera că dimensiunea de 8 biți este specială. Pur şi simplu, la un moment dat în trecut s-a considerat că această dimensiune corespundea necesităților practice din acea vreme. O dată impus un standard pe scară largă, înlocuirea sa devine foarte dificilă. În plus, în acest caz, dacă dimensiunea 8 nu este neapărat mai "bună" decât alte variante, ea nu este nici mai "rea", deci o eventuală schimbare a standardului nu ar aduce nici un câştig. Alegerea unuia dintre multiplii octetului ca standard nu ar reprezenta decât o soluție temporară, întrucât şi noua dimensiune ar deveni curând prea mică şi ar fi necesară utilizarea de multipli.

O problemă importantă provine din modul de implementare al biților. Dat fiind că valoarea unui bit este materializată în practică de o valoare de tensiune, devine clar că până şi operațiile aritmetice cele mai simple, cum ar fi adunarea sau scăderea, nu au sens într-un circuit electric. Cu alte cuvinte, deşi avem posibilitatea fizică de a reprezenta numerele, trebuie să putem realiza şi operațiile dorite.

Soluția a fost găsită în lucrările matematicianului englez George Boole. În jurul anului 1850 (deci mult înainte de apariția calculatoarelor), acesta a realizat că probleme matematice complexe pot fi rezolvate prin reducere la o serie de răspunsuri de tipul "adevărat"/"fals". Astfel, el a elaborat o teorie, numită logica Boole (sau logica booleană), care lucrează cu aceste două valori. Se observă imediat analogia cu noțiunea de bit, care permite tot două valori. Dacă, de exemplu, asociem valoarea "adevărat" din logica Boole cu cifra binară 1 şi valoarea "fals" cu cifra 0 (de altfel se poate şi invers), rezultatele logicii booleene pot fi folosite direct în sistemele de calcul.

Logica Boole defineşte un set de operații elementare (NOT, AND, OR etc.), cu ajutorul cărora poate fi descrisă orice funcție. Din fericire, aceste operații elementare ale logicii Boole pot fi uşor implementate cu ajutorul tranzistorilor. Ca urmare, adunarea, scăderea şi celelalte operații aritmetice, care sunt în fond nişte funcții matematice ca oricare altele, pot fi la rândul lor realizate practic. În concluzie, deşi obişnuim să spunem că un calculator lucrează doar cu numere, în realitate el lucrează cu şiruri de biți, asupra cărora aplică o serie de prelucrări, pe care noi le numim adunare, înmulțire etc.; pentru circuitele din calculator, aceste operații nu au o semnificație specială, ci sunt nişte funcții oarecare.

Pe baza conceptelor prezentate mai sus sunt create circuite din ce în ce mai complexe, capabile să îndeplinească sarcini tot mai dificile. Aceste circuite formează în cele din urmă sistemul de calcul.

1.2. Tipuri de calculatoare

Practic, astăzi nu mai există domeniu al societății care să nu facă apel la calculatoare. Problemele pe care le putem aborda cu ajutorul unui sistem de calcul sunt atât de variate, încât nu este de mirare că nu există un singur tip de calculator, capabil să rezolve în mod optim toate aceste probleme. Caracteristici cum ar fi puterea de calcul, capacitatea de stocare şi nu în ultimul rând prețul trebuie luate în considerare atunci când se intenționează achiziționarea unui calculator. Toate aceste caracteristici variază într-o plajă foarte largă. Putem clasifica sistemele de calcul după gradul de miniaturizare, parametru care dă o imagine suficient de clară asupra performanțelor. Pe măsură ce coborâm pe scara dimensiunilor fizice sistemelor,

constatăm o reducere progresivă a puterii de calcul şi a capacității de stocare, dar şi a prețului.

- *Supercalculatoarele* includ de obicei sute sau mii de procesoare care lucrează în paralel. Rezultă astfel o putere de calcul impresionantă, datorată şi faptului că se utilizează tehnologii aflate la limita posibilităților actuale, în condițiile în care prețul nu este factorul principal în construcția lor. O asemenea putere de calcul îşi găseşte utilizare în rezolvarea unor probleme de foarte mare complexitate din câteva domenii de vârf ale ştiinței, cum ar fi: elaborarea modelelor climatice, studiul cutremurelor, secvențierea genomului, interacțiunile particulelor fundamentale, testarea teoriilor cosmologice. Prețul unui supercalculator este însă pe măsură, fiind exprimat în general în milioane de dolari. O abordare mai recentă se axează pe utilizarea sistemelor distribuite (bazate pe rețele de calculatoare) pentru a obține performanțe comparabile, dar la un preț cu un ordin de mărime mai mic. Asemenea supercalculatoare pot constitui în unele cazuri o alternativă viabilă la cele clasice.

- *Mainframe* reprezintă un tip de calculator de asemenea de mare putere, dar nu la acelaşi nivel cu supercalculatoarele. Sunt utilizate cel mai adesea pentru gestiunea bazelor de date de dimensiuni foarte mari, precum şi a altor aplicații asemănătoare, care necesită o capacitate de stocare foarte mare şi o interacțiune puternică cu un număr mare de utilizatori, concretizată printr-un volum foarte mare de comunicații de date. De asemenea, se pot folosi şi la efectuarea de calcule ştiințifice de o complexitate mai redusă decât în cazul supercalculatoarelor.

- *Serverul* este un calculator care are rolul de a pune la dispoziția altor sisteme de calcul diverse resurse (capacitate de stocare, putere de calcul, informații de un anumit tip), de obicei prin intermediul unei rețele de calculatoare. Fiind destinat să servească de obicei un număr mare de cereri în paralel, serverul trebuie să aibă la rândul său o putere de calcul considerabilă. Un exemplu bine cunoscut (dar nu neapărat cel mai important) îl constituie serverele Web. De altfel, serverele din gama de vârf au tendința de a înlocui sistemele de tip mainframe, profitând şi de progresul tehnologic, care permite obținerea unor performanțe superioare şi de către calculatoarele de dimensiuni mai reduse. Dincolo de imaginea uzuală pe care o are publicul, un server de vârf poate include un număr mare de procesoare, iar capacitatea de stocare poate fi foarte mare.

- *Stațiile de lucru* sunt destinate lucrului individual, dar sunt proiectate pentru a rula aplicații profesionale, de complexitate mare, cum ar fi: grafică 3D, prelucrări audio şi video, aplicații de tip CAD sau GIS etc.

- *Sistemele desktop* intră în categoria calculatoarelor personale, care pot fi folosite pentru aplicații de birou (editare de texte, calcul tabelar, baze de date de dimensiuni reduse etc.) sau pentru jocuri. Sunt în principiu cele mai ieftine calculatoare şi din acest motiv cele mai accesibile publicului larg. De asemenea, se adresează şi utilizatorilor nespecialişti în informatică.

- *Laptop* şi *notebook* sunt termeni care desemnează calculatoarele personale portabile. Acestea au la bază aceleaşi principii şi tehnologii ca şi sistemele desktop şi sunt prin urmare comparabile din toate punctele de vedere (putere, preț etc.). Diferența constă în accentul pus pe mobilitate. Un laptop are dimensiuni şi greutate reduse şi poate funcționa un timp (câteva ore) cu ajutorul bateriilor, fără alimentare de la rețeaua electrică. Ținta principală a acestei categorii de sisteem o reprezintă mediul de afaceri, pentru care mobilitatea este esențială.

2. Arhitectura sistemelor de calcul

Modelul de bază pentru arhitectura unui sistem de calcul a fost introdus de savantul american John von Neumann, ca rezultat al participării sale la construcția calculatorului ENIAC, în anii 1944-1945. Acest model este cunoscut în literatura de specialitate ca arhitectura von Neumann.

2.1. Arhitectura generalizată von Neumann

După cum se observă în figura 2.1., un sistem de calcul este format din 3 unități de bază, care sunt conectate între ele prin 3 căi separate de comunicație, numite magistrale (mai des se foloseşte termenul englezesc - **bus**).

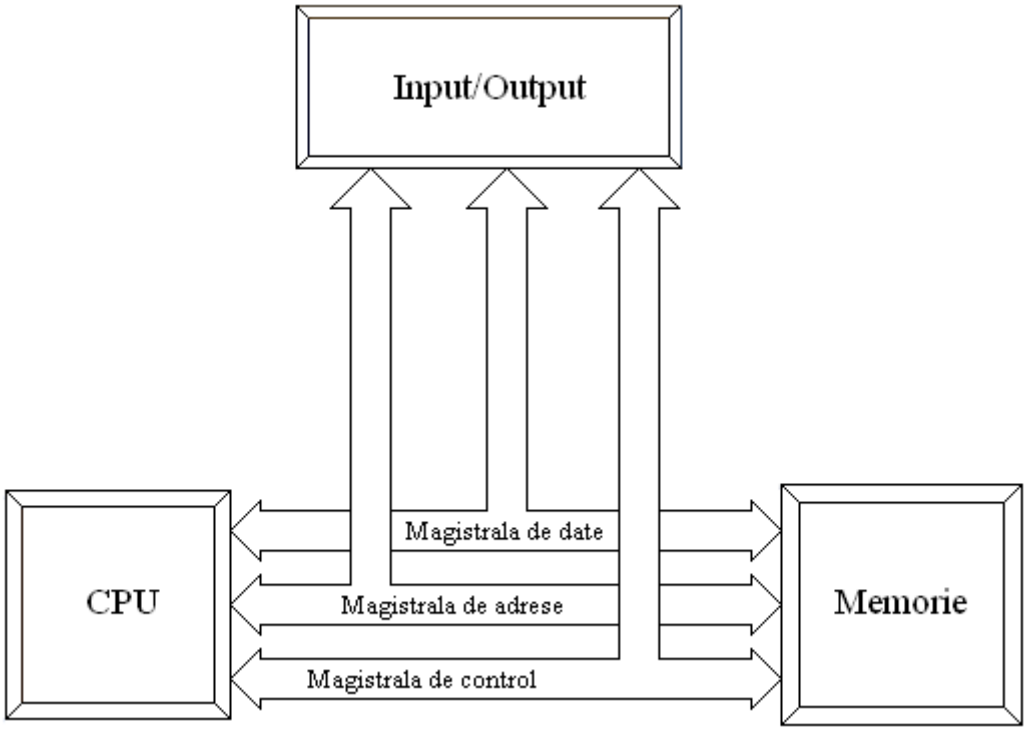

Fig. 2.1.

Informațiile vehiculate în sistemul de calcul se împart în 3 categorii:

- *date* care trebuie prelucrate
- *instrucțiuni* care indică prelucrările ce trebuie efectuate asupra datelor (adunare, scădere, comparare etc.)
- *adrese* care permit localizarea diferitelor date şi instrucțiuni

Simplist spus, sarcina unui sistem de calcul este de a executa instrucțiuni (grupate în secvențe coerente, care urmăresc un obiectiv bine stabilit, numite programe) asupra datelor; adresele joacă un rol auxiliar, dar nu mai puțin important. Privind din această perspectivă, vom analiza pe scurt scopul elementelor din figura 2.1.

Unitatea de memorie are rolul de a stoca atât instrucțiunile, cât şi datele asupra cărora vor opera instrucțiunile (operanzii). Instrucțiunile unui program trebuie aduse în memorie anterior începerii execuției programului respectiv. De asemenea, unele date se vor afla în memorie înaintea pornirii prelucrării, iar rezultatele prelucrării se vor memora în timpul execuției programului. Această memorie, realizată în diverse

tehnologii de-a lungul evoluției calculatoarelor, constituie suportul fizic necesar desfăşurării operațiilor executate de CPU. Structural, memoria este formată dintr-un număr mare de celule independente (numite şi locații), fiecare celulă putând memora o valoare.

Pentru organizarea şi regăsirea informațiilor în memorie se folosesc aşa-numitele adrese. O adresă este de fapt un număr care identifică în mod unic o locație de memorie; cu alte cuvinte, fiecărei locații îi este asociat un număr unic (adresa sa), în aşa fel încât să nu existe două locații diferite cu aceeaşi adresă. Pentru accesarea unei informații din memorie se furnizează adresa acelei informații, iar circuitele de control al memoriei vor furniza conținutul locației care reprezintă informația cerută. Similar se petrec lucrurile şi la scrierea în memorie.

Tehnologic, unele dispozitive de memorie pot reține informația numai când sunt alimentate electric (şi avem de-a face cu aşa-zisa memorie volatilă), în timp ce altele păstrează informația şi atunci când nu sunt alimentate electric, formând memoria nevolatilă. Aceasta din urmă este folosită în mod special la stocarea programelor pentru inițializarea calculatorului şi a sistemului de operare.

Unitatea centrală de prelucrare (CPU) are rolul de a executa instrucțiunile. Din acest motiv, CPU reprezintă componenta cea mai importantă a sistemului de calcul şi poate controla activitatea celorlalte componente. Deoarece atât instrucțiunile, cât şi datele prelucrate de instrucțiuni se găsesc în memorie, execuția unei instrucțiuni presupune efectuarea de către CPU a următoarei secvențe de acțiuni:

- Depunerea pe busul de adrese a unei informații care localizează adresa de memorie ce conține câmpul de cod al instrucțiunii (faza de *adresare*).
- Citirea codului instrucțiunii şi depunerea acestuia într-un registru intern al decodificatorului de instrucțiuni. Această informație este vehiculată pe busul de date (faza de *citire*).
- Decodificarea codului instrucțiunii, în urma căreia CPU va cunoaşte ce instrucțiune are de executat şi ca urmare pregăteşte modulele ce vor participa la instrucțiunea respectivă (faza de *decodificare*).
- Executarea efectivă a operației specificate de de instrucțiune - faza de *execuție* propriu-zisă.

După terminarea execuției unei instrucțiuni, se continuă cu extragerea instrucțiunii următoare şi trecerea ei prin secvențele amintite ş.a.m.d.

Dispozitivele de intrare/ieşire (I/O - input/output), numite şi dispozitive periferice, permit transferul informației între CPU, memorie şi lumea externă.

Funcțional, aceste dispozitive de I/O pot fi adresate (apelate) de către CPU similar cu memoria, ele dispunând de asemenea de câte un set de adrese. În mod clasic, schimbul de informații cu exteriorul se face sub controlul CPU, dar există tehnici, care vor fi amintite mai târziu, prin care accesul la memorie se poate face şi cu o intervenție minimă a CPU (aşa-numitele transferuri DMA - Direct Memory Access).

Cele mai utilizate periferice sunt: monitorul, tastatura, mouse-ul, discul dur, mediile de stocare portabile (dischetă, CD, DVD etc.), imprimanta.

Busul de date este acea cale care leagă cele 3 blocuri funcționale (o parte a sa poate să iasă şi în exteriorul sistemului) şi pe care se vehiculează datele propriu-zise (numere sau caractere) sau instrucțiunile programului.

Busul de adrese este calea pe care sunt transmise de CPU adresele către memorie, când se face o operație cu memoria (citire sau scriere), sau se vehiculează adresele dispozitivului de I/O în cazul unui transfer cu un periferic.

Busul de comenzi vehiculează semnalele de comandă şi control între toate aceste blocuri şi astfel permite o sincronizare armonioasă a funcţionării componentelor sistemului de calcul. În marea majoritate a cazurilor, semnalele de comandă sunt emise de către CPU şi servesc la controlul funcţionării celorlalte componente.

Arhitectura de tipul von Neumann a fost o inovaţie în logica maşinilor de calcul, deosebindu-se de cele care se construiseră până atunci prin faptul că sistemul trebuia să aibă o cantitate de memorie, similar creierului uman, în care să fie stocate atât datele, cât şi instrucţiunile de prelucrare (programul). Acest principiu al memoriei a reprezentat unul din fundamentele arhitecturale ale calculatoarelor. Diferenţa fundamentală consta în stocarea în memorie nu numai a datelor, ci şi a programelor.

A început astfel să apară din ce în ce mai clar care este aplicabilitatea memoriei. Datele numerice puteau fi tratate ca şi valori atribuite unor locaţii specifice ale memoriei. Aceste locaţii erau asemănate cu nişte cutii poştale care aveau aplicate etichete numerotate (de exemplu 1). O astfel de locaţie putea conţine o variabilă sau o instrucţiune. A devenit posibil ca datele stocate la o anumită adresă să se schimbe în decursul calculului, ca urmare a paşilor anteriori. Astfel, numerele stocate în memorie au devenit simboluri ale cantităţilor şi nu neapărat valori numerice, în acelaşi mod în care algebra permite manipularea simbolurilor *x* şi *y* fără a le specifica valorile. Cu alte cuvinte, se putea lucra cu entităţi abstracte.

Calculatoarele ulterioare şi mai târziu microprocesoarele au implementat această arhitectură, care a devenit un standard. În ciuda vechimii sale, arhitectura von Neumann nu a putut fi înlocuită până azi.

2.2. Clasificarea arhitecturilor interne

Într-un efort continuu de îmbunătăţire, arhitectura von Nuemann a fost dezvoltată în mai multe direcţii, rezultând sisteme de calcul cu posibilităţi noi şi adaptate noilor cerinţe cerute de societate. Pentru a vedea aceste noi direcţii, ne vom folosi de o clasificare a sistemelor după arhitectura internă, propusă de Flynn:

SISD (Single Instruction Single Data - o singură instrucţiune, o singură dată de prelucrat)

Sunt sistemele uzuale cu un singur microprocesor. Aici se încadrează microprocesoarele clasice cu arhitectură von Neumann pe 8, 16, 32, 64 biţi cu funcţionare ciclică - preluare instrucţiune, execuţie instrucţiune (rezultă prelucrarea datelor) ş.a.m.d.

Tot în această categorie trebuie incluse şi aşa-numitele procesoare de semnal DSP (Digital Signal Processors) folosite actualmente pe scară largă în plăcile de sunet, telefonia mobilă etc.

SIMD (Single Instruction Multiple Data - o singură instrucţiune, mai multe date)

Sunt sistemele cu microprocesoare matriceale, la care aceleaşi operaţii aritmetice se execută în paralel pentru fiecare element al matricei, operaţia necesitând o singură instrucţiune (se mai numesc şi sisteme de procesare vectorială). Privită ca o cutie neagră, o arhitectură SIMD arată ca în figura 2.2:

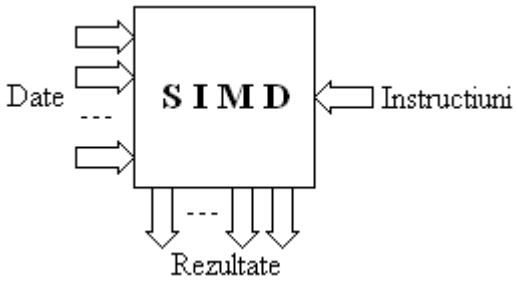

Fig. 2.2.

Dacă intrăm în detalii arhitecturale, devine evident faptul că există mai multe unități de execuție, capabile să execute același tip de prelucrare în paralel pe date diferite, coordonate de o singură unitate de control. Deoarece datele prelucrate de o unitate de execuție sunt independente de datele celorlalte unități, devine clar faptul că fiecare unitate de execuție are memoria sa proprie. Evident, trebuie să existe totuși și o formă de interconectare între unități. Arhitectura SIMD are deci următoarea schemă de principiu:

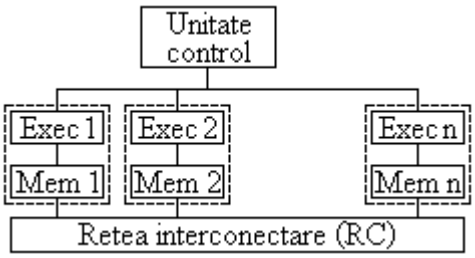

Fig. 2.3.

Eficiența SIMD-urilor se dovedește a fi ridicată în cazul unor programe cu paralelism de date masiv, pus în evidență cel mai adesea de anumite bucle de programe.

Exemplu: arhitectura de tablou sistolic construită în 1984 de General Electric, un tablou de 64 × 64 elemente, rezultând o viteză de procesare de 1 miliard de operații pe secundă.

MISD (Multiple Instruction Single Data - mai multe instrucțiuni, o singură dată)

Sunt sistemele care folosesc microprocesoare de tip **pipeline** (conductă), metodă folosită de către procesoarele recente (Pentium sau echivalente). La un astfel de microprocesor, de exemplu, în paralel se execută instrucțiunea *n*, se decodifică instrucțiunea *n*+1 și se aduce în memorie instrucțiunea *n*+2. Această arhitectură a fost inspirată de banda de montaj a automobilelor.

Celebrele supercomputere Cray din anii 1970 foloseau de asemenea arhitectura MISD.

MIMD (Multiple Instruction Multiple Data - mai multe instrucțiuni, mai multe date) sunt acele sisteme în care se încadrează atât supercalculatoarele cu procesoare dedicate, cât și sistemele multiprocesor (figura 2.4). Sunt cunoscute și sub denumirea de arhitecturi paralele.

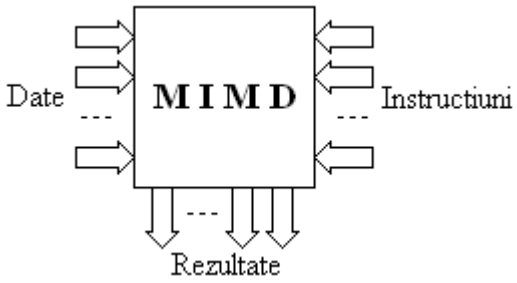

Fig. 2.4.

În cazul arhitecturilor MIMD, procesoarele pot avea fiecare propria sa memorie locală, dar există şi o memorie globală, accesată prin intermediul reţelei de interconectare (figura 2.5). Complexitatea reţelei de interconectare poate varia într-o plajă largă, mergând de la simpla arhitectură de tip *bus comun* (în care un singur procesor poate avea acces la memoria globală) şi până la reţelele de tip *crossbar* (care permit accesul simultan al tuturor procesoarelor la memoria globală, dar la module diferite pentru fiecare procesor).

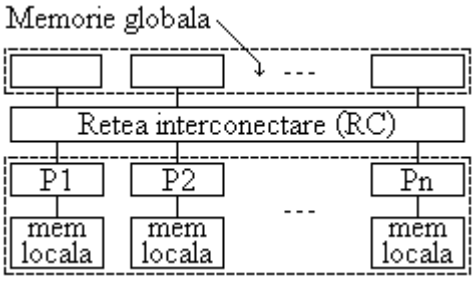

Fig. 2.5.

Acest tip de arhitecturi necesită existenţa unor sisteme de operare capabile să asigure rularea în paralel a diferitelor procese, altfel potenţialul lor de performanţă ar rămâne în mare parte neexploatat. Astfel, avem mai multe tipuri de sisteme de operare:

- master-slave - la care funcţiile sistemului de operare sunt ataşate unui procesor distinct (*master*), iar restul procesoarelor (numite *slave*) accesează aceste funcţii indirect, prin intermediul procesorului master

- divizat - nu există un procesor evidenţiat, fiecare procesor având funcţiile de sistem plasate separat în memorie

- flotant - când funcţiile sistemului de operare sunt plasate în memoria comună, putând fi accesate de oricare microprocesor al sistemului; acest model de calcul poată denumirea de *multiprocesare simetrică* (Symmetrical Multiprocessing - SMP)

Tot în cadrul acestei ultime categorii trebuie amintite aşa-numitele *transputere*, care sunt de fapt microcalculatoare integrate într-un singur circuit, cu memorie proprie şi reţea de conectare punct la punct cu alte transputere din aceeaşi categorie. Cu acestea se pot construi maşini SIMD sau MIMD, folosindu-se limbaje de programare specifice proceselor paralele (de exemplu OCCAM), precum şi algoritmi paraleli.

Putem aminti aici, în afara clasificării propuse de Flynn, şi de *sistemele distribuite*. Un sistem distribuit este de fapt un grup de calculatoare legate în reţea, care cooperează într-un mod asemănător cu procesoarele ditnr-un sistem multiprocesor. Desigur, există şi diferenţe, legate în principal de eterogenitatea sistemului distribuit (este puţin probabil că toate calculatoarele dintr-o reţea sunt identice, putând fi chiar foarte diferite) şi de particularităţile modului de comunicare

într-o rețea de calculatoare. În mod paradoxal, în acest moment sistemele distribuite sunt în general mai eficiente economic decât sistemele multiprocesor. Evident, o rețea de *n* calculatoare este aproape întotdeauna mai scumpă decât un sistem cu *n* procesoare; eficiența vine însă din faptul că practic în orice instituție există deja o bază instalată de calculatoare, cu care se poate realiza un sistem distribuit, în timp ce sistemele multiprocesor trebuie achiziționate. Din acest motiv sistemele distribuite cunosc o dezvoltare remarcabilă (de altfel, ele nu apăreau în clasificarea lui Flynn deoarece, la momentul când a fost propusă această clasificare, încă nu existau sisteme distribuite).

3. Arhitectura internă a microprocesoarelor Intel

În general, când se vorbeşte despre un microprocesor se înţelege că acesta reprezintă CPU (Central Processing Unit) din arhitectura generalizată von Neumann. După ce s-au construit primele microprocesoare pe 8 biţi, s-a căutat ca puterea de calcul a acestora să se mărească prin creşterea numărului de biţi prelucraţi, trecându-se la prelucrări pe 16 biţi, apoi la 32 biţi şi, mai recent, la 64 biţi. Totodată, s-au făcut în permanenţă inovaţii în cadrul arhitecturii interne, care au dus la o creştere a vitezei de prelucrare.

3.1. Microprocesoare pe 16 biţi

Începând cu microprocesoarele pe 16 biţi (8086, 8088, 80286), unitatea de prelucrare nu mai urmează strict schema ciclică descrisă la arhitectura von Neumann, de extragere a instrucţiunii, decodificare, execuţie ş.a.m.d. Noutatea a fost divizarea unităţii de prelucrare în două unităţi (vezi figura 3.1):

- unitatea de execuţie (Execution Unit - EU)
- unitatea de interfaţă cu magistrala (Bus Interface Unit - BIU)

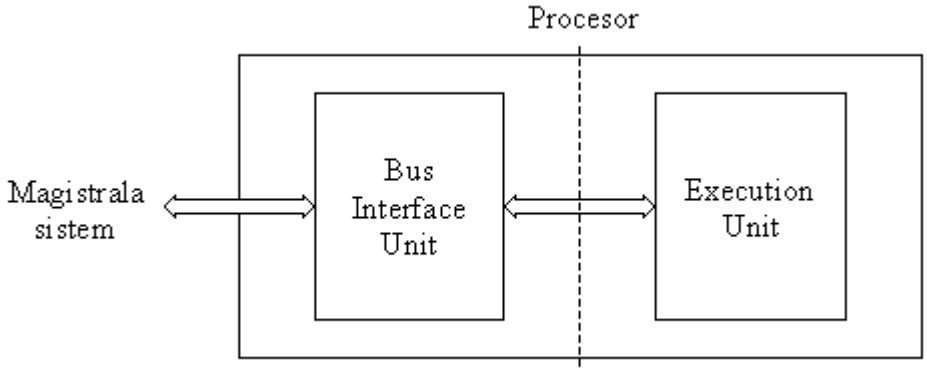

Fig. 3.1.

După cum se observă, cele două unităţi sunt legate între ele cu o conductă (*pipeline*) prin care sunt transferate instrucţiunile extrase din memoria program de către BIU spre EU; aceasta din urmă are numai rolul de a executa instrucţiunile extrase de BIU, EU neavând nici o legătură cu magistrala sistemului. În timp ce EU îşi îndeplineşte sarcina, BIU extrage noi instrucţiuni pe care le organizează într-o coadă de aşteptare (*queue*). La terminarea execuţiei unei instrucţiuni, EU are deja la dispoziţie o nouă instrucţiune din coada de aşteptare menţinută de BIU. Cele două unităţi, EU şi BIU, lucrează deci în paralel, existând momente de sincronizare şi aşteptare între ele, atunci când coada de instrucţiuni este goală, ceea ce se întâmplă însă foarte rar.

Funcţionarea paralelă a celor două unităţi (BIU şi EU) este transparentă utilizatorului. Această arhitectură se mai numeşte şi arhitectură cu prelucrare secvenţial - paralelă *pipeline*.

Unitatea de execuţie EU conţine o unitate logico-aritmetică (ALU) de 16 biţi, registrul indicatorilor condiţionali, registrul operatorilor şi regiştrii generali.

BIU conţine indicatorul de instrucţiuni IP (Instruction Pointer), registrele de segmente, un bloc de control al magistralei şi de generare de adrese şi o memorie organizată sub forma unei cozi, în care sunt depuse instrucţiunile extrase (Instruction Queue).

Vom detalia în continuare blocurile de regiştri, arătând şi rolul unora dintre aceştia în cursul execuţiei programului. Avem următoarele categorii de regiştri pe 16 biţi:

- regiştri generali
- regiştri de segment
- registru pointer de instrucţiune
- registrul indicatorilor de stare şi control

Regiştrii generali sunt în număr de 8 şi sunt împărţiţi în două seturi a câte 4 regiştri (fig. 3.2):

- regiştrii de date AX, BX, CX, DX
- regiştrii de pointer şi de index SP, BP, SI, DI

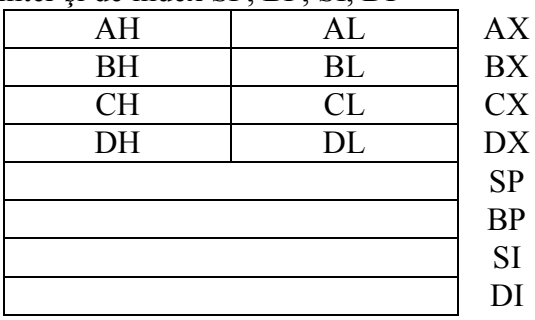

Fig. 3.2.

Fiecare registru de date este format din doi regiştri de câte 8 biţi, care pot fi adresaţi şi individual. Regiştrii pointer şi index pot fi folosiţi numai pe 16 biţi şi pot participa la majoritatea operaţiilor aritmetice şi logice. De asemenea, regiştrii pointer şi index (ca şi BX) sunt utilizaţi şi la adresarea memoriei.

Regiştrii de segment, la rândul lor, sunt folosiţi exclusiv pentru adresarea locaţiilor de memorie. Rolul lor în adresare este însă diferit de cel al regiştrilor prezentaţi mai sus şi se referă la împărţirea memoriei în segmente.

Un segment este o unitate logică de memorie care poate avea cel mult 64 Ko (locaţii contigue), în timp ce cantitatea maximă de memorie acdresabilă de un procesor Intel pe 16 biţi este de 1 Mo . Fiecărui segment i se atribuie o adresă de bază, care este adresa locaţiei de început a segmentului. Valoarea acestei adrese se află memorată într-un registru de segment. Există 4 regiştri segment (conform figurii 3.3) şi ei se găsesc localizaţi în BIU.

Fig. 3.3.

În memorie pot exista, în funcţie de poziţia lor relativă, segmente adiacente, parţial suprapuse sau suprapuse complet şi disjuncte.

Deci fiecare aplicaţie (program aflat în memorie) are la dispoziţie un spaţiu de 64Ko pentru codul instrucţiunilor (segmentul de cod), 64 Ko pentru stivă (segment de stivă) şi 128 Ko pentru date (segmentul de date şi extra segmentul). Unele aplicaţii pot însă gestiona un spaţiu de memorie mult mai mare, făcând gestionarea segmentelor după propriile necesităţi.

Împărţirea memoriei în segmente de 64Ko provine din faptul că microprocesoarele pe 8 biţi anterioare gestionau un spaţiu de numai 64Ko. Proiectanţii de la Intel au căutat ca şi noile microprocesoare pe 16 biţi să folosească eventual programe scrise pentru microprocesoarele anterioare, adoptând această

soluție a segmentului, făcând însă adresarea memoriei mai greu de înțeles şi limitată ca funcționalitate.

Generarea adresei fizice

Fiecare locație de memorie are două tipuri de adresă:

- fizică
- logică

Adresa fizică este o valoare formată din 20 biți care identifică unic fiecare locație din spațiul de adresare de 1 Mo. Adresa fizică se găsesşte în domeniul 00000_h-$FFFFF_h$ şi se mai numeşte adresă absolută.

Pentru a nu depinde de locul unde se află codul în memorie, se folosesc aşa-zisele adrese logice, diferite de cele fizice. Adresa logică constă dintr-o valoare de bază de segment şi o valoare de deplasament (offset). Pentru orice locație de memorie, valoarea de bază a segmentului este adresa primului octet al segmentului care conține locația. Această adresă este exprimată în paragrafe (paragraful fiind o unitate de 16 biți) iar deplasamentul (offset) este distanța în octeți de la începutul segmentului până la locația respectivă. Adresa de bază şi deplasamentul sunt valori pe 16 biți fără semn.

Mai multe adrese logice pot corspunde aceeleiaşi locații fizice dacă se află în segmente diferite, după cum se observă din figura 3.4.

BIU generează întotdeauna o adresă fizică dintr-o adresă logică, după mecanismul prezentat în figura 3.5.

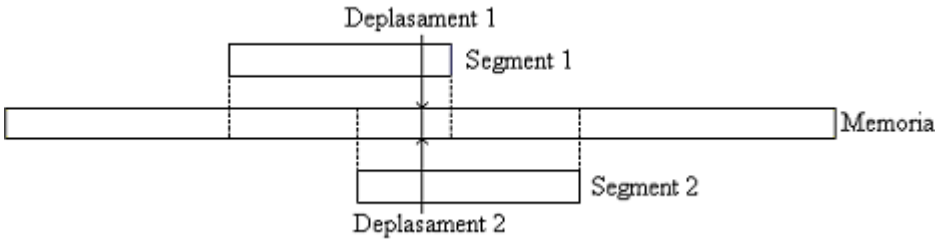

Fig. 3.4.

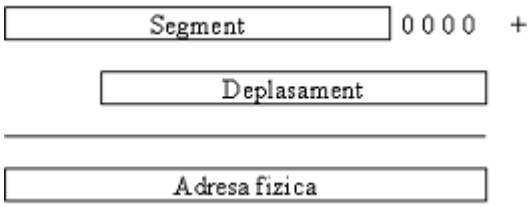

Fig. 3.5.

Se observă că, în principiu, calculul adresei fizice se face prin deplasarea bazei segmentului (conținută într-un registru segment) cu 4 poziții spre stânga (ceea ce echivalează cu o înmulțire cu 16) şi adunarea valorii deplasamentului.

BIU obține în mod diferit adresa logică a unei locații de memorie, în funcție de tipul de referire a memoriei. Instrucțiunile sunt întotdeauna încărcate din segmentul de cod curent, iar registrul IP conține deplasamentul instrucțiunii următoare față de începutul segmentului. Operațiile cu stiva lucrează în segmentul de stivă curent, iar registrul SP conține deplasamentul față de vârful stivei. Variabilele se găsesc de obicei în segmentul de date, iar deplasamentul este dat după modul de adresare specificat în instrucțiune. Rezultatul este aşa-numita adresă efectivă, despre care vom mai vorbi la prezentarea modurilor de adresare.

Acestea sunt atribuțiile segmentelor în mod implicit. Unele din aceste atribuții pot fi schimbate.

Faptul că memoria microprocesorului 8086 sau 8088 este segmentată face posibilă scrierea de programe care sunt independente de poziția lor în memorie, adică sunt *relocabile dinamic*. Aceste programe trebuie însă să îndeplinească o sumă de condiții. Dacă aceste condiții sunt îndeplinite, programul poate fi mutat oriunde în memorie. Un astfel de program poate fi făcut să ocupe o zonă contiguă de memorie, lăsând spațiu nefragmentat şi pentru alte aplicații. De asemenea, acest fapt este important atunci când programul este inactiv în memorie şi sistemul de operare mută programul pe disc; atunci când se doreşte ca programul să fie adus din nou în memorie, pentru a se relua execuția sa, zona în care s-a aflat prima dată este ocupată de un alt program. Prin simpla schimbare a valorilor registrelor de segment, programul poate rula din altă zonă de memorie.

Pointerul de instrucțiuni (IP) este un registru pe 16 biți actualizat de BIU şi conține deplasamentul (offsetul) instrucțiunii următoare față de începutul segmentului de cod curent. Programele nu au acces direct la el, dar există instrucțiuni care îl modifică şi îl încarcă sau îl descarcă de pe stivă.

Registrul de stare şi control (Flags register) conține 6 indicatori de stare şi 3 indicatori de control, notați conform figurii 3.6.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		A		P		C

Fig. 3.6.

EU poziționează cei 6 indicatori de stare pentru a reflecta anumite stări ale unei operații aritmetice sau logice. Un anumit set de instrucțiuni permit modificarea execuției unui program în funcție de starea acestor indicatori - cum ar fi instrucțiunile de salt condiționat. Indicatorii de stare reflectă următoarele condiții:

- C (Carry) indică transportul în exterior al bitului cel mai semnificativ al rezultatului operațiilor aritmetice
- P (Parity) este poziționat dacă rezultatul are paritate pară (conține un număr par de biți cu valoarea 1)
- A (Auxiliar Carry) este poziționat dacă a avut loc un transfer de la semioctetul inferior la semioctetul superior al rezultatului şi este folosit în aritmetica zecimală
- Z (Zero) poziționat dacă rezultatul operației a fost zero
- S (Sign) este poziționat dacă cel mai semnificativ bit al rezultatului este 1 (prin convenție, 0 indică un număr pozitiv, iar 1 - un număr negativ)
- O (Overflow) - poziționat când dimensiunea rezultatului depăşeşte capacitatea locației de destinație şi a fost pierdut un bit

Pentru controlul unor operații ale procesorului, pot fi modificați (prin program) trei indicatori de control:

- D (Direction) stabileşte dacă operațiile pe şiruri lucrează prin incrementare (când are valoarea 0) sau prin decrementare (valoarea 0)
- I (Interrupt) este poziționat pe 1 pentru a permite CPU să recunoască cererile de întrerupere externe mascabile
- T (trap) - când este poziționat pe 1, trece CPU în execuția de pas cu pas, în scopul depanării programului instrucțiune cu instrucțiune

3.2. Microprocesoare pe 32 biți

Apariția microprocesorului 80386 a reprezentat un salt major în familia Intel. Pe lângă creşterea dimensiunii operanzilor de la 16 la 32 biți, au fost introduse o serie de

noi caracteristici care au îmbunătățit substanțial performanțele şi funcționalitatea. Linia inovativă a fost continuată şi de procesoarele care au urmat (80486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium IV), astfel încât astăzi familia procesoarelor Intel pe 32 biți are în urmă o istorie bogată. În cele ce urmează vom creiona noutățile aduse de procesoarele pe 32 biți.

Dimensiunea datelor şi adreselor

Magistrala de adrese, la fel ca şi cea de date, are 32 biți, deci cantitatea maximă de memorie care poate fi accesată este de 4 Go. Deşi au apărut deja aplicații al căror necesar de memorie depăşeşte şi această valoare, saltul de la 1 Mo este mai mult decât semnificativ. Ca dovadă, au trecut aproximativ două decenii de la lansarea microprocesorului 80386 şi marea majoritate a programelor încă nu au probleme cu limita de memorie.

Corespunzător, dimensiunea regiştrilor a crescut şi ea la 32 biți. De fapt, pentru a se păstra compatibilitatea cu procesoarele pe 16 biți, regiştrii acestora există în continuare, exact în forma în care au fost prezentați mai sus. În plus, au fost introduşi o serie de regiştri noi, pe 32 biți, care îi includ pe cei deja existenți (figura 3.7).

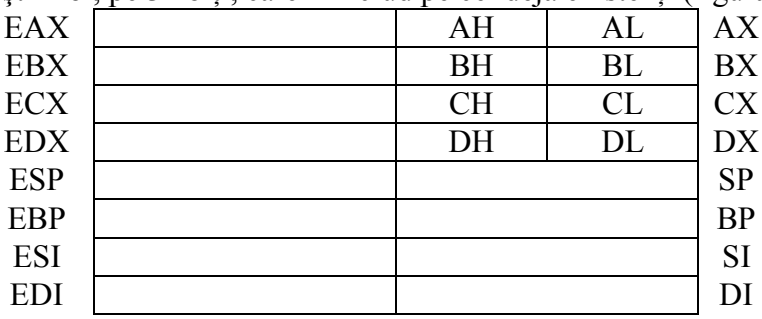

Fig. 3.7.

Fiecare registru pe 32 biți include, la partea mai puțin semnificativă, unul dintre regiştrii pe 16 biți, iar numele său este format din numele registrului vechi, adăugându-i-se în față litera E (*extended*). Astfel, noii regiştri generali ai procesorului sunt EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI. Denumirile de regiştri de date, index şi pointer îşi pierd practic semnificația, deoarece noii regiştri sunt complet interschimbabili, adică nu mai există diferențe între ei cu privire la operațiile la care pot fi folosiți.

Registrul IP (pointerul de instrucțiuni al procesoarelor pe 16 biți) este inclus într-un registru de 32 biți, numit desigur EIP. Acesta din urmă îndeplineşte acum rolul de pointer de instrucțiuni. În plus, registrul indicatorilor de condiții (FLAGS) este la rândul său extins la un registru de 32 biți (EFLAGS). Nu vom intra însă în detalii privitoare la indicatorii de condiții suplimentari.

Există şi o excepție de la regula extinderii. Regiştrii de segment rămân la dimensiunea de 16 biți, fără a fi creați regiştri noi. În schimb sunt adăugați doi regiştri de segment noi, tot pe 16 biți, numiți FS şi GS (figura 3.8). Aceştia nu au un rol anume, ci pot fi folosiți pentru accesarea datelor, la fel ca DS sau ES.

Fig. 3.8.

În final, facem observația că noii regiştri nu sunt întotdeauna disponibili. Orice microprocesor Intel pe 32 biți are două moduri diferite de funcționare:

- modul *real*, în care funcționează la fel ca un procesor pe 16 biți
- modul *protejat*, în care se poate lucra cu regiştrii de 32 biți

Modul real a fost păstrat pentru compatibilitatea cu aplicațiile mai vechi. Trecerea microprocesorului dintr-un mod de funcționare în altul poate fi controlată prin software.

Coprocesorul matematic

Începând cu microprocesorul 80486, coprocesorul matematic a fost inclus în unitatea centrală de procesare, nemaifiind un circuit separat. Aceasta reprezintă o consecință logică a evoluției tehnologiei, care a permis o creştere a numărului de tranzistoare pe o pastilă de siliciu (prin scăderea dimensiunii acestora), dar şi o recunoaştere a importanței calculelor în virgulă mobilă în cadrul aplicațiilor. Prin integrarea coprocesorului matematic în acelaşi circuit cu microprocesorul s-a obținut un spor de performanță datorat scrutării căilor de semnal şi simplificării comunicării între componente.

Pipeline

O altă îmbunătățire importantă este dată de perfecționarea structurii de tip pipeline. Aşa cum am văzut mai sus, la procesoarele pe 16 biți unitatea de prelucrare este împărțită în unitatea de interfață cu magistrala (BIU), care se ocupă de aducerea în avans a instrucțiunilor din memorie şi depunerea lor într-o coadă, şi unitatea de execuție (EU), care preia instrucțiunile din coadă şi le execută. Această împărțire permite lucrul în paralel al celor două unități, ceea ce se traduce printr-o funcționare mai rapidă. La procesoarele pe 32 biți ideea a fost dusă şi mai departe. După cum ne amintim, fiecare instrucțiune constă din 4 faze (adresare, citire, decodificare, execuție). Mai mult, fiecare din aceste faze (mai ales cea de execuție, care este cea mai complexă) poate consta la rândul său din mai multe operații mai simple. Ideea este că fiecare din aceste operații lucrează în principiu cu alte resurse, deci toate operațiile se pot executa în paralel. Astfel, execuția unei instrucțiuni poate fi împărțită într-un număr mare de acțiuni elementare, numite *stagii* ale pipeline-ului. Deci, la un moment dat se pot afla în execuție în procesor mai multe instrucțiuni, în diferite faze; în cazul cel mai fericit există câte o instrucțiune tratată în fiecare stagiu al pipeline-ului. Deşi execuția unei instrucțiuni de la început până la sfârşit necesită un număr mare de acțiuni, o instrucțiune poate începe să fie executată imediat ce instrucțiunea anterioară a trecut de primul stagiu.

De ce este atât de eficientă această structură? Activitatea procesorului este coordonată cu ajutorul semnalului de ceas al sistemului. Trecerea execuției unei instrucțiuni de la un stagiu la altul se poate face numai atunci când "bate" ceasul, deci la intervale regulate de timp. Pe de altă parte, fiecare acțiune elementară (stagiu) se execută într-o anumită durată finită de timp. Dacă semnalul de ceas este prea rapid, acțiunile nu se mai pot realiza pe durata dintre două "bătăi" ale ceasului, ceea ce ar duce la pierderea controlului asupra execuției instrucțiunilor. Ca urmare, frecvența ceasului nu poate fi crescută oricât de mult, ci este limitată de duratele de execuție ale stagiilor. Dacă acțiunile elementare sunt mai simple (ceea ce implică o descompunere mai fină a instrucțiunilor şi deci un număr de stagii mai mare), ele vor consuma mai puțin timp; implicit, frecvența ceasului va putea fi crescută. Dacă analizăm funcționarea unui pipeline, observăm că, în cazul cel mai fericit, la fiecare "bătaie" a ceasului se poate termina de executat câte o instrucțiune, deci performanța procesorului depinde direct de creşterea frecvenței semnalului de ceas.

Procesoarele Intel au evoluat în sensul creşterii continue a numărului de stagii a pipeline-ului. La ultimele microprocesoare Pentium IV s-a ajuns la un pipeline cu 32 stagii, ceea ce este mult mai mult decât oricare variantă anterioară. Cu alte cuvinte, execuția unei instrucțiuni a microprocesorului este împărțită în 32 operații elementare. Pentru microprocesoarele cu număr foarte mare de stagii se foloseşte şi denumirea de unități superpipeline.

Totuşi, structura de tip pipeline are şi dezavantaje. Ideea sa de pornire este că fiecare stagiu lucrează cu alte resurse ale procesorului decât restul stagiilor. Această cerință nu poate fi niciodată satisfăcută în totalitate. Ca un exemplu simplu, o operație de adunare necesită folosirea unității aritmetico-logice (ALU) pentru efectuarea calculului propriu-zis. În acelaşi timp, în faza de adresare a unei instrucțiuni, valoarea registrului indicator de instrucțiuni (IP la procesoarele pe 16 biți) este incrementată, pentru a putea aduce codul următoarei instrucțiuni. Deoarece incrementarea este tot o operație de adunare, va fi nevoie tot de ALU. Astfel, o instrucțiune de adunare aflată în faza de execuție şi o altă instrucțiune aflată în faza de adresare vor concura pentru aceeaşi resursă (ALU). Asemenea situații apar de fapt mult mai des, deoarece între instrucțiuni există relații de dependență rezultate din însăşi logica programului. Se întâmplă foarte des ca o instrucțiune să aibă nevoie de rezultatul unei instrucțiuni anterioare, care încă nu l-a calculat. Din acest motiv, de multe ori o instrucțiune (şi implicit cele care urmează după ea) trebuie să aştepte până când devine disponibilă o resursă de care are nevoie, dar care este momentan folosită de altă instrucțiune. După cum am văzut, o asemenea resursă poate fi fie o componentă hardware a procesorului, fie rezultatul altei instrucțiuni. Ca urmare, în practică se întâmplă rareori ca procesorul să termine de executat câte o instrucțiune la fiecare "bătaie" a ceasului, deci câştigul de performanță nu este atât de mare cât sperăm.

O concluzie importantă care se desprinde de aici este că simpla lungime a pipeline-ului (adică numărul de stagii) nu este singurul factor care influențează performanța procesorului. Împărțirea corectă a instrucțiunilor în operații elementare poate fi la fel de importantă. Din păcate, această alegere nu poate fi făcută ca urmare a unor considerații teoretice sau a unor criterii clare, fiind în mare măsură o problemă de inspirație. În practică se poate vedea cum microprocesoarele AMD, care sunt compatibile cu cele Intel la nivel de limbaj, dar au o implementare diferită pentru pipeline (care este mult mai scurt), reuşesc să obțină performanțe asemănătoare, deşi lucrează la frecvențe mult mai mici. O soluție deja folosită de procesoarele actuale (atât Intel, cât şi AMD) este existența a două sau mai multe pipeline-uri; astfel se pot executa mai multe instrucțiuni în paralel, atunci când dependențele dintre instrucțiuni nu introduc perioade de aşteptare. Procesoarele care utilizează mai multe pipeline-uri se numesc superscalare.

Lucrul cu segmente de memorie

În condițiile în care spațiul de memorie accesibil microprocesorului a crescut până la 4 Go, segmentele de 64 Ko, cu care lucrau procesoarele pe 16 biți, sunt evident anacronice. Procesoarele pe 32 biți au păstrat conceptul de segment de memorie, dar într-un mod adaptat necesităților.

Sistemele actuale permit executarea simultană a mai multor programe. O consecință imediată este că mai multe programe se pot afla simultan în memorie, de unde apare şi riscul ca două sau mai multe asemenea programe să încerce să utilizeze (în scopuri diferite) aceeaşi adresă de memorie. Evident, asemenea situații sunt de natură să ducă la interferențe nedorite între programe, cu efecte dintre cele mai grave, de aceea trebuie evitate prin orice mijloace.

Soluția de principiu constă în a introduce o separare între adresele pe care un program crede că le accesează şi adresele utilizate în realitate. Altfel spus, atunci când un proces încearcă să acceseze o locație aflată la o anumită adresă, accesul în memoria principală se produce la altă adresă. În continuare, adresele pe care un proces crede că le accesează vor fi numite *adrese virtuale*, iar adresele accesate în realitate - *adrese fizice*. Astfel, fiecare proces are la dispoziție un spațiu de adrese (virtuale) propriu, independent de spațiile de adrese ale celorlalte procese. La prima vedere nu se întrevede nici un câştig, ci dimpotrivă. Totuşi, printr-o gestionare atentă a corespondențelor între adresele virtuale ale fiecărui proces şi adresele fizice accesate în realitate, se poate obține efectul dorit: mai multe programe accesează acelaşi adrese (virtuale), fără a se produce suprapuneri în memoria fizică. Avantajul obținut astfel este că o aplicație nu mai trebuie să țină cont de problema interferențelor, ci se poate executa ca şi cum toată memoria ar fi la dispoziția sa.

Să vedem cum se poate folosi lucrul cu segmente pentru a gestiona adresele virtuale şi fizice. După cum am arătat, adresa unui octet dintr-un segment se compune din două părți independente: adresa de început (de bază) a segmentului şi deplasamentul octetului în interiorul segmentului (numit şi offset). Deci, orice adresă poate fi scrisă printr-o pereche de forma: (*adresa_baza_segment*, *deplasament)*. Această pereche este de fapt o adresă virtuală, deoarece este transformată de procesor într-o adresă fizică.

Să analizăm ce se întâmplă în momentul în care două programe diferite încearcă să acceseze aceeaşi adresă virtuală. Adresa de bază a unui segmentului este precizată prin intermediul unui registru de segment. Deci, pentru a evita suprapunerea, este suficient ca valoarea registrului de segment să fie diferită pentru cele două programe. Desigur, gestiunea valorilor regiştrilor de segment nu cade în seama programelor, ci este realizată de sistemul de operare, singurul care are o vedere de ansamblu asupra tuturor aplicațiilor ce rulează pe calculator. În acest mod, programele nu trebuie să opereze nici o modificare pentru arezolva problema.

Totuşi, problema nu este încă rezolvată complet, întrucât este necesară stocarea informațiilor referitoare la valorile regiştrilor de segment pentru fiecare program. În plus, pot apărea şi erori de altă natură. De exemplu, orice segment are o anumită dimensiune. Procesoarele pe 32 biți permit ca fiecare segment să ocupe până la 4 Go, dar în practică fiecare segment va avea o dimensiune impusă de cantitatea de informație (date, instrucțiuni) pe care o conține. Este deci posibil ca deplasamentul să fie prea mare, astfel încât adresa fizică rezultată să fie în afara segmentului. Mai mult, fiecare segment poate fi folosit în general de un singur program; trebuie deci evitat ca alt program să acceseze respectivul segment. Toate aceste situații trebuie detectate, iar programele care încearcă să realizeze un acces incorect trebuie oprite.

Pentru aceasta, fiecărui segment existent în memorie i se asociază o structură de date numită *descriptor de segment*. Principalele informații conținute de acesta sunt următoarele:

- adresa de început a segmentului
- dimensiunea segmentului
- drepturi de acces la segment

Toți descriptorii de segment sunt grupați într-un tabel, astfel încât pentru a putea identifica un segment este suficient să fie cunoscut indicele descriptorului său în tabel. În acest moment, adresele devin perechi de forma: (*indice_descriptor*, *deplasament*). Cu alte cuvinte, registrul de segment nu va mai conține adresa de bază a segmentului, ci indicele acestuia în tabloul descriptorilor. Acesta este motivul pentru care regiştrii de segment au rămas la dimensiunea de 16 biți, în timp ce adresele sunt pe 32 biți.

Concret, în momentul în care un program încearcă să acceseze o adresă de memorie (dată în forma de mai sus), au loc următoarele acțiuni:

- se verifică în descriptorul segmentului drepturile de acces, pentru a se decide dacă programul are dreptul de a accesa adresa dorită
- se verifică dacă deplasamentul nu depăşeşte dimensiunea segmentului
- dacă se produce o eroare la unul din paşii anteriori, accesul programului la adresa de memorie solicitată este oprit
- dacă nu s-a produs nici o eroare, se calculează adresa fizică şi se realizează accesul propriu-zis

Desigur, toate aceste verificări şi calcule nu sunt realizate prin software, ci direct în hardware de către microprocesor. Este vorba de o activitate complexă, astfel încât procesorul a fost dotat cu o componentă specializată, numită *unitate de management al memoriei* (Memory Management Unit - MMU).

Paginarea memoriei

Utilizarea segmentării nu este lipsită de probleme. Deoarece un segment ocupă întotdeauna o zonă continuă în memoria fizică, atunci când este creat acesta trebuie plasat într-o zonă liberă suficient de mare. La un moment dat, datorită diverselor alocări şi eliberări de segmente, memoria arată ca un şir de zone ocupate şi libere, de diferite dimensiuni. Experiența arată că în timp se ajunge la apariția unui număr mare de zone libere de dimensiuni foarte mici, practic inutilizabile. Procesul de formare a acestor zone libere, care nu pot fi folosite din cauza dimensiunilor prea reduse, poartă numele de *fragmentare externă*. Este posibil în acest fel ca un segment să nu mai poată fi plasat în memorie, deşi dimensiunea totală a zonelor libere este mai mare decât dimensiunea segmentului respectiv.

O abordare alternativă, în general mai eficientă, poartă numele de *paginare a memoriei*. Ideea este de a stabili o corespondență mai directă între adresele virtuale şi adresele fizice decât în cazul segmentării. Astfel, spațiul de adrese virtuale ale unui program este împărțit în zone de dimensiuni egale (uzual 4-8 Ko), numite *pagini* (pages). Similar, memoria fizică este împărțită în zone de aceeaşi lungime, numite *cadre de pagină* (page frames). Sistemul de operare construieşte în memorie, pentru fiecare proces, un tabel, numit tabel de paginare, care va conține paginile aparținând procesului şi cadrele de pagină corespunzătoare în memoria fizică. În cazul paginării, adresele virtuale sunt valori pe 32 biți, exact ca adresele fizice.

Practic, lucrurile se desfăşoară astfel:

- De fiecare dată când programul încearcă un acces la o adresă virtuală, MMU determină pagina din care face parte respectiva adresă şi o caută în tabelul de paginare.
- Dacă pagina se află în tabelul de paginare, se determină cadrul de pagină corespunzător şi se calculează adresa fizică. Abia în acest moment se realizează accesul propriu-zis la locația de memorie dorită.
- Dacă pagina nu se găseşte în tabelul de paginare, avem o încercare de acces ilegal la memorie, eroare numită uzual *defect de pagină* (page fault). La fel ca la segmentare, în cazul detectării unei erori, accesul la memorie nu se mai realizează.

Să luăm ca exemplu un tabel de paginare având următoarea structură:

Adrese virtuale	0	1	2	4	5
Adrese fizice	0	2	3	1	6

Considerăm pentru simplitate că dimensiunea unei pagini este de 100 octeți. În acest caz, pagina 0 va conține adresele virtuale 0-99, pagina 1 adresele 100-199, pagina 2 adresele 200-299 etc. Similară este relația dintre cadrele de pagină şi adresele fizice.

Presupunem că programul încearcă să acceseze adresa 124. Aceasta face parte din pagina 1, căreia îi corespunde în tabel cadrul de pagină 2. Astfel, în realitate va fi accesata adresa fizică 224. Dacă programul încearca să acceseze adresa 367, MMU observă că pagina 3, din care face parte acea adresă, nu apare în tabelul de paginare, deci este un acces ilegal.

Deoarece o zonă de memorie oricât de mare poate fi împărțită în pagini de dimensiune fixă, ce pot fi răspândite în memorie în orice mod, nu apare fenomenul de fragmentare externă. În schimb se întâlneşte *fragmentarea internă*: deoarece dimensiunea paginilor este fixă, iar zonele de memorie cu care lucrează procesele pot avea orice dimensiune, în general la sfârşitul unei pagini rămâne o zonă nefolosită. Din acest motiv, dimensiunea paginilor de memorie este stabilită ca un compromis între două cerințe contradictorii:

- o dimensiune prea mare a paginii provoacă o fragmentare internă puternică
- o dimensiune prea mică a paginii duce la ocuparea unui spațiu prea mare de către tabelele de paginare, micşorând memoria care poate fi folosită de aplicații

Dacă dimensiunea paginii este bine aleasă, fragmentarea internă este în general mai redusă decât cea externă (apărută în cazul segmentării).

Există şi posibilitatea de a folosi simultan segmentarea şi paginarea memoriei. Este însă vorba de o tehnică rar folosită în practică, din cauza complexității sale, astfel încât nu vom insista asupra ei.

4. Microprocesoare: funcţionare şi adresarea datelor

4.1. Funcţionarea la nivel de magistrală

După cum s-a văzut, pe placa de bază a calculatorului PC există un circuit de ceas (clock) care generează un semnal de o anumită frecvenţă - 14,3 Mhz la PC original şi peste 3 GHz la calculatoarele actuale -, folosit pentru execuţia sincronizată a operaţiilor din interiorul micriprocesorului şi efectuarea transferurilor între diferite blocuri la nivelul magistralei externe.

În timpul funcţionării, activitatea microprocesorului poate fi descompusă în secvenţe de microoperaţii, care formează aşa-numiţii *cicli maşină*. În funcţie de natura operaţiei care se execută, putem avea 5 tipuri de cicli maşină:

1. *Citire (read)* - atunci când se citesc date din memorie sau de la dispozitivele de I/O
2. *Scriere (write)* - când datele se inscriu in locaţii de memorie sau intr-un dispozitiv de I/O
3. *Recunoaştere întrerupere* - în cazul generării unei cereri de întrerupere pe care μP o identifică
4. *Oprire (halt)* - atunci când microprocesorul este oprit până la primirea unei cereri de întrerupere
5. *Arbitrare magistrală* - când sistemul de calcul este prevăzut cu mai multe microprocesoare care pot avea acces la o magistrală comună

Toate aceste operaţii sunt descrise cu lux de amănunte în foile de catalog ale firmelor ce produc microprocesoare. Fără a le detalia aici, arătăm doar care este semnificaţia unei diagrame de semnal.

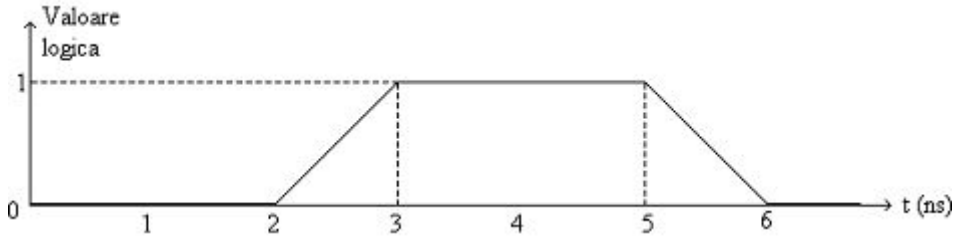

Fig. 4.1.

După cum se observă în figura 4.1, semnalul reprezentat are valoarea "0" logic la t=0, la t=2 începe să crească spre 1 logic, la t=3 deja este "1" logic, se menţine 2 ns şi apoi "cade" înapoi la "0" logic. Acest mod de reprezentare a semnalelor se numeşte diagramă de semnal. În figura 4.1 este prezentată diagrama semnalului de ceas (clock).

În practică există un mare număr de semnale care intervin în desfăşurarea ciclilor maşină. Pentru fiecare tip de ciclu există câte o diagramă de semnal, iar distribuţia în timp a semnalelor este specifică fiecărui tip de ciclu.

4.2. Moduri de adresare la microprocesoarele Intel

Programul care se execută se găseşte memorat in segmentul de cod. După cum am văzut într-un capitol anterior, când se încarcă o instrucţiune din memorie adresa acestuia este furnizată de regiştrii CS (ca adresă de bază) şi respectiv IP sau EIP (ca deplasament). În mod normal, conţinutul registrului (E)IP este incrementat pe măsură ce instrucţiunile se execută, astfel ca totdeauna să fie deja selectată instrucţiunea care

urmează. Instrucțiunile de salt necondiționat sau apel de procedură pot însă modifica valorile regiştrilor (E)IP şi eventual CS, modificându-se astfel ordinea secvențială de execuție a instrucțiunilor.

4.2.1. Adresarea datelor

Datele din memorie care formează operanzii instrucțiunilor pot fi adresate în mai multe moduri. Aceşti operanzi pot fi conținuți în regiştri, în memorie, în instrucțiuni sau porturi de I/O. Operațiile care implică date numai din regiştri sunt cele mai rapide, nefiind nevoie de utilizarea magistralei pentru acces la memorie.

Regiştrii folosiți şi modul de adresare (memorie sau registru) sunt codificați în interiorul instrucțiunii. În practică există următoarele tipuri de adresare:

1. **Adresare imediată**

În acest caz operandul apare chiar în instrucțiune.

```
mov ax,5
mov eax,5
```

Această instrucțiune va inițializa registrul (E)AX cu valoarea 5. Evident, a doua instrucțiune poate fi folosită numai la microprocesoarele pe 32 biți, care conțin registrul EAX.

2. **Adresare directă**

Deplasament în interiorul segmentului curent (de obicei în interiorul segmentului de date) este furnizat de către instrucțiune.

```
add bx,[200]
```

Această instrucțiune adună la registrul (E)BX conținutul locației de la adresa efectivă 200 din segmentul de date.

3. **Adresare indirectă** (prin regiştri)

La procesoarele pe 16 biți, offsetul este furnizat de unul dintre regiştrii BX, SI sau DI, iar registrul implicit este bineînțeles DS. În cazul procesoarelor pe 32 biți, pentru offset poate fi folosit oricare registru general pe 32 biți.

```
mov al,[bx]
mov al,[ebx]
```

Conținutul adresei de memorie de la adresa dată de (E)BX este transferat în AL. Observăm că procesoarele pe 32 biți pot lucra şi cu regiştri de 16 sau 8 biți pentru date, dar nu şi pentru adrese (care au întotdeauna 32 biți).

4. **Adresare bazată sau indexată**

Offsetul se obține adunând la unul din regiştrii de bază (BX sau BP) sau index (SI sau DI) un deplasament constant. Din nou, la procesoarele pe 32 biți se poate folosi suma a oricare 2 regiştri generali.

```
mov ax,[bx+5]
mov ax,[ebx+5]
```

La conținutul adresei din (E)BX se adună deplasamentul 5 şi se obține offsetul operandului. Aceste tipuri de adresare se pot folosi când avem structuri de date de tip tablou, care pot fi localizate în diferite locuri din memorie. Constanta va conține offsetul la care începe structura de date, iar deplasamentul va furniza poziția elementelor în cadrul structurii.

5. **Adresare bazată şi indexată**

Este cea mai complexă formă de adresare, care combină variantele anterioare, permițând adresarea cu 2 indecşi (conținuturile regiştrilor de bază şi indecşi la procesoarele pe 16 biți, oricare 2 regiştri generali la procesoarele pe 32 biți).

```
mov ax,[bx+si+7]
mov eax,[ebx+ecx+9]
```

Procesoarele pe 32 biți permit şi o formă extinsă a ultimelor două tipuri de adresare. Mai exact, registrul folosit pentru offset (pentru adresarea bazată sau indexată), respectiv unul din cei 2 regiştri (pentru adresarea bazată şi indexată), poate fi înmulțit cu 2, 4, sau 8. În acest mod pot fi accesate mai uşor tablourile cu elemente având dimensiunea de 2, 4, sau 8 octeți (cum sunt tipurile standard).

```
mov ax,[ebx*4+5]
mov eax,[ebx+ecx*2+9]
```

În cele arătate anterior, de obicei registrul segment implicit este cel de date, presupunând că se adresează operanzi de calcul obişnuiți. Excepția este dată de cazul când se foloseşte BP, atunci registrul de segment implicit fiind SS. Dacă se doreşte folosirea altui registru de segment decât cel implicit, în instrucțiune se va preciza explicit despre care registru de segment este vorba, cum ar fi în instrucțiunea următoare:

```
mov bx,es:[bp+7]
```

Dacă nu precizam ES, implicit se folosea SS, deoarece registrul de bază este BP.

4.3. Stiva

O zonă specială de memorie este folosită de programe atunci când se execută subprograme sau se transmit parametrii de la un program la altul.

Această zonă poartă numele de *stivă* (în engleză *stack*), fiindcă funcționarea ei este asemănătoare cu cea a unei stive fizice de obiecte. Modul de funționare al stivei este numit LIFO (*Last Input - First Output*). Într-o stivă, datele au voie să fie depuse numai prin partea superioară, astfel încât informația depus ultima dată (Last Input) va fi disponibilă, fiind deasupra stivei, şi va putea fi scoasă prima (First Output).

Stiva este folosită implicit în mecanismul de apel al procedurilor.

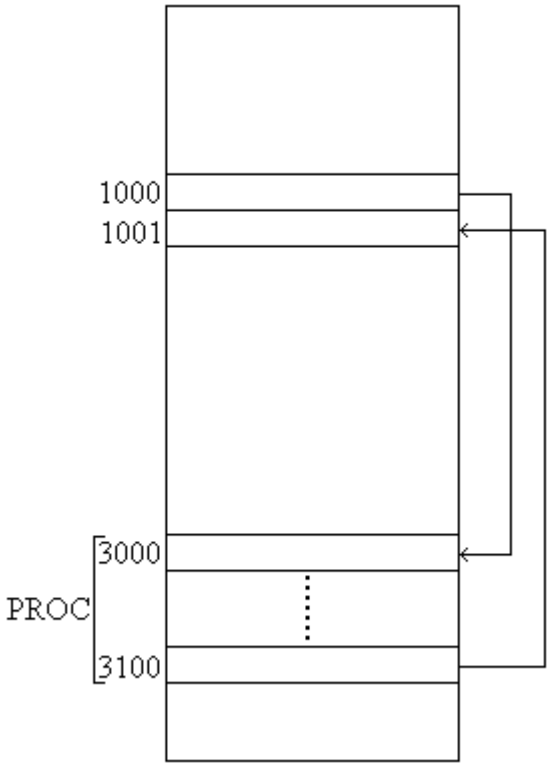

Fig. 4.2.

Figura 4.2 ilustrează un astfel de caz. Atunci când s-a ajuns cu pointerul de instrucțiuni (E)IP la adresa 1000, se execută un apel la procedura PROC care se găseşte în memorie la adresa 3000. Atunci când se termină de executat procedura, trebuie să ne întoarcem la prima instrucțiune de după instrucțiunea de apel procedură (1001 în cazul nostru). Deci, adresa 1001 trebuie să fie memorată undeva pentru ca la revenirea din procedură să reluăm programul din acel loc.

Stiva se foloseşte pentru a memora această adresă de revenire. Microprocesorul este proiectat astfel încât, la execuția unui apel de procedură (CALL), să salveze automat în stivă adresa de memorie care conține instrucțiunea următoare din secvență (de fapt conținutul registrului IP). Când se întâlneşte în procedură instrucțiunea RETURN (revenire din procedură), tot automat microprocesorul ia din vârful stivei adresa memorată anterior şi o încarcă în registrul (E)IP, executând apoi instrucțiunea găsită la această adresă (adică instrucțiunea de la adresa 1001).

Putem defini stiva ca un concept abstract de structură de date, asupra cărora operează instrucțiuni special proiectate în acest scop.

O zonă de stivă este caracterizată de o adresă curentă, numită adresa vârfului stivei, care la microprocesoarele Intel este adresată prin registrul (E)SP (*stack pointer* - indicator de stivă). Operațiile de bază cu stiva sunt PUSH (depune un cuvânt în stivă) şi respectiv POP (extrage un cuvânt din stivă).

La microprocesoarele pe 16 biți, cuvintele transferate în stivă sunt de 16 biți (2 octeți), deci adresa curentă a vârfului stivei se va incrementa sau decrementa cu 2 la fiecare operație. Similar, la microprocesoarele pe 32 biți se poate lucra cu operanzi având fie 16, fie 32 biți, în al doilea caz vârful stivei fiind incrementat sau decrementat cu 4 la fiecare operație. Figura 4.3 prezintă modul de acțiune al unei operații PUSH.

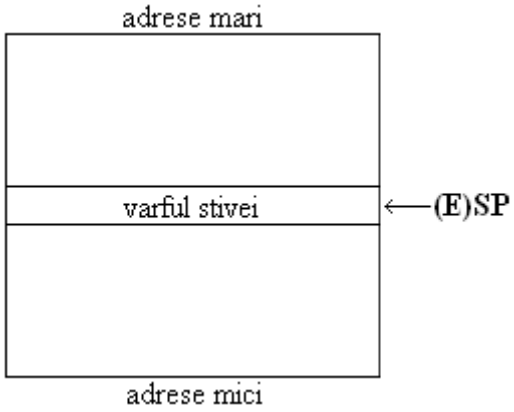

Fig. 4.3.

Se observă că stiva "creşte" în jos pe măsură ce se depun date în ea. La fel ca în cazul oricărui segment, mărimea sa maximă este de 64 Ko la microprocesoarele pe 16 biți şi 4 Go la cele pe 32 biți. Pot exista mai multe stive definite la un moment dat, dar numai una este activă.

Stiva este folosită explicit la salvări şi refaceri de date, transmiterea parametrilor către proceduri etc. Implicit se foloseşte în cazul apelurilor de procedură. De asemenea, programele care permit definirea şi folosirea funcțiilor recursive folosesc stiva pentru memorarea valorilor intermediare ale parametrilor şi valorilor funcțiilor.

4.4. Procesoare CISC şi RISC

Procesoarele Intel din seria x86 fac parte din categoria procesoarelor numite CISC (Complex Instruction Set Computer - calculator cu set complex de instrucțiuni). Acestea sunt procesoare la care setul de instrucțiuni cuprinde un număr mare de operații implementate. Totodată, acestea pot lucra cu mai multe tipuri de operanzi; pentru unele asemenea tipuri, adresa operanzilor este calculată într-un timp relativ lung şi abia după aceea se poate executa instrucțiunea propriu-zisă. S-a văzut începând chiar de la calculatoarele mari (mainframe) ale anilor 1950 că execuția operațiilor simple cu operanzi în regiştri consumă mai puțin timp de calcul. În plus, o dată cu dezvoltarea software-ului de aplicații şi prin studii statistice efectuate de companii ca IBM s-a demonstrat că instrucțiunile complexe din procesoarele CISC sunt rareori folosite, preferându-se folosirea a câtorva instrucțiuni simple în locul uneia mai complexe. Astfel, 10% din setul de instrucțiuni al unui procesor CISC stă la baza a peste 90% din totalul codului generat de un compilator ca PASCAL. Cercetările s-au îndreptat către proiectarea unor procesoare cu instrucțiuni mai puține, cu mulți regiştri şi memorie imediată (cache) în care să fie reținute datele temporare, toate acestea conducând la o viteză mai mare. Astfel au apărut procesoarele RISC (Reduced Instruction Set Computer - calculator cu set redus de instrucțiuni). Acestea s-au dezvoltat imediat ce tehnologia a permis obținerea de memorie ieftină.

Simplitatea setului de instrucțiuni, modul de adresare mai simplu care necesită un singur acces la memoria principală într-un singur impuls de ceas, precum şi execuția instrucțiunilor în structuri de tip pipeline au permis proiectarea unor unități de execuție superscalare, care permit execuția mai multor instrucțiuni simultan. Aceasta au dus la succesul structurilor de tip RISC, care s-au impus alături de cele mai vechi de tip CISC. În ultimii 15 ani nu s-au mai proiectat noi structuri CISC.

Datorită simplității structurale, necesarul de siliciu este mai mic şi astfel a apărut posibilitatea de integrare pe cip a unor memorii şi unități de execuție multiple. Toate acestea au făcut ca procesoarele RISC să fie mai performante decât cele de tip CISC.

Procesoarele noi din seria x86 folosesc în structura lor tehnici RISC, păstrând însă vechea structură CISC. Acest set de instrucțiuni complex face ca software-ul mai vechi să poată rula şi pe procesoarele actuale.

Procesoare reprezentative de tip RISC sunt SPARC, MIP şi ALPHA, care însă nu au reuşit să atingă succesul familiei x86.

5. Sistemul de întreruperi

Unul din marile avantaje ale unui calculator față de orice altă maşină creată de om este capacitatea sa de a răspunde la un număr mare de sarcini imprevizibile. Cheia acestor posibilități o constituie ceea ce numim *întreruperi*.

Cu ajutorul acestora, calculatorul poate să oprească temporar o sarcină pe care o execută şi să comute pe o alta, ca răspuns la întreruperea intervenită (cum ar fi de exemplu apăsarea unei taste sau primirea unor date pe place de rețea). Acest mecanism face ca sistemul de calcul să fie foarte flexibil, permițând răspunsul imediat la un eveniment extern, a cărui tratare poate fi foarte urgentă, prin întreruperea sarcinii curente şi reluarea acestei sarcini după ce s-au rezolvat cerințele impuse de întrerupere.

Noțiunea de întrerupere presupune suspendarea programului în curs de execuție şi transferul controlului către o anumită specializată, numită rutină de tratare a întreruperii. Mecanismul după care se face acest transfer este în esență de tip apel de procedură, ceea ce înseamnă că se revine în programul întrerupt, din locul în care acesta a rămas, după ce s-a terminat rutina de tratare a întreruperii.

5.1. Întreruperi hardware şi software

Clasificarea întreruperilor se poate face după mai multe criterii. O posibilă clasificare este dată în figura 5.1.

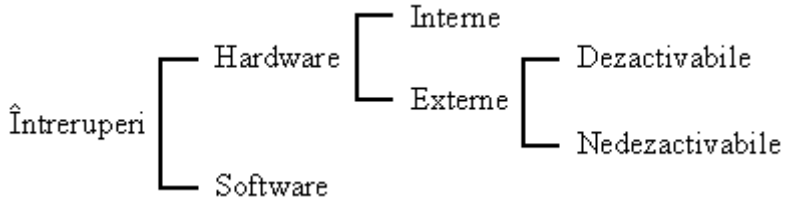

Fig. 5.1.

Întreruperile software apar în urma execuției de către microprocesor a unor instrucțiuni cum ar fi INT, DIV, IDIV, fiind deci cauzate de către software.

Întreruperile hardware interne apar ca urmare a unor situații speciale în care se poate afla procesorul (cum ar fi execuția pas cu pas în cazul depanării programelor).

Întreruperile hardware externe sunt provocate de semnale electrice, care în ultimă instanță sunt aplicate de către dispozitivele periferice pe intrările INT şi NMI ale microprocesorului.

Întreruperile externe dezactivabile se aplică pe intrarea INT şi sunt luate în considerare numai dacă bistabilul IF (Interrupt Flag) din registrul indicatorilor de condiții are valoarea 1.

Întreruperile externe nedezactivabile au loc la aplicarea unui semnal corespunzător pe intrarea NMI (Non Maskable Interrupt) şi sunt luate în considerare întotdeauna. Un exemplu pentru folosirea intrării NMI este semnalizarea căderii tensiunii de alimentare.

5.1.1. Întreruperi hardware dezactivabile

Aceste întreruperi sunt controlate de unul (la PC original) sau mai multe circuite specializate (la AT şi celelalte calculatoare), numite controlere de întreruperi, de tipul Intel 8259. Acest circuit are menirea de a culege cereri de întrerupere de la mai multe dispozitive, de a stabili o prioritate a cererilor (în cazul în care există mai multe cereri

de întrerupere simultane) şi în final de a transmite un semnal de întrerupere pe pinul INT al microprocesorului şi un semnal de identificare al dispozitivului periferic care a făcut cererea.

Figura 5.2 ilustrează schema de întreruperi a microprocesorului 80386 folosită în calculatoarele AT şi următoarele. Observăm modul de legare în cascadă a controlerelor de întreruperi. La microprocesoarele anterioare (8086, 8088) era prezent un singur circuit 8259.

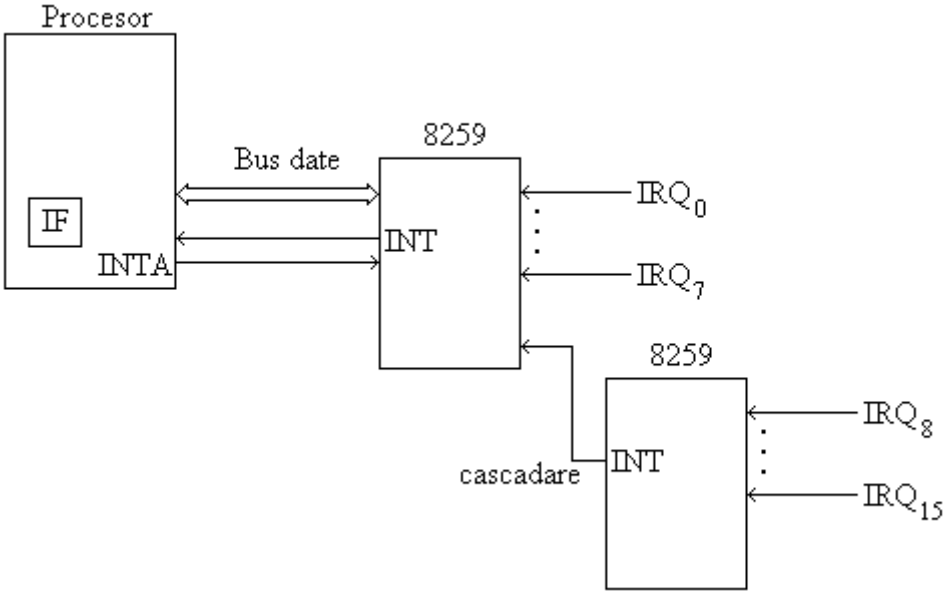

Fig. 5.2.

Dacă la una din liniile IRQ0-IRQ15 (IRQ = Interrupt Request) se primeşte o cerere de întrerupere de la un dispozitiv periferic (s-a apăsat o tastă, trebuie să se facă un ciclu de refresh al memoriei etc.), acest semnal este analizat de controlerul 8259 şi în final acesta va genera o întrerupere pe linia INT către microprocesor. Dacă în microprocesor bistabilul IF are valoarea 1 (adică întreruperile hardware externe sunt activate), microprocesorul va trimite înapoi controlerului un semnal INTA (Interrupt Acknowlege - recunoaştere întrerupere), prin care îl anunță că întreruperea este recunoscută şi acceptată. În continuare, controlerul de întrerupere va depune pe magistrala de date un octet special, numit *octet type*, care va identifica tipul (nivelul) întreruperii. Controlerul de întrerupere 8259 având 8 intrări pentru întreruperi de la echipamente, va putea să trimită 8 valori diferite, fiecare indentificând în mod unic una din cele 8 intrări. Se mai spune că acest controler poate furniza 8 vectori de întrerupere, identificați prin octetul type, iar sistemul de întreruperi la microprocesoarele Intel este un sistem vectorizat.

Cum cu un octet (8 biți) se pot efectua 256 combinații diferite, vor putea exista 256 valori diferite ale octetului type. Deci într-un sistem pot exista maximum 256 nivele de întrerupere diferite. Pentru fiecare nivel (valoare a octetului type) se poate asocia o procedură (rutină sau subprogram) de deservire a întreruperii respective. Deci putem avea maximum 256 proceduri.

Adresele acestor rutine sunt trecute într-o aşa-numită tabelă a vectorilor de întreruperi, care se află în primii 1024 octeți ai memoriei RAM (primul Ko). Datorită faptului că fiecare rutină de întrerupere se poate afla în alt segment, pentru a identifica adresa unei rutine trebuie să-i furnizăm adresa de segment (2 octeți) şi adresa offsetului (încă 2 octeți), deci o adresă va ocupa 4 octeți. Cum sunt 256 întreruperi

posibile, spațiul de memorie ocupat de tabelă este 256 · 4 = 1024 = 1Ko. Se observă că modul în care sunt memorate adresele (cu 2 octeți pentru offset) corespunde microprocesoarelor pe 16 biți, respectiv modului real al microprocesoarelor pe 32 biți. De altfel, orice microprocesor pe 32 biți intră în modul real la pornire (adică la punerea sub tensiune). Astfel, sistemul de întreruperi funcționează în acelaşi mod indiferent de tipul procesorului.

În figura 5.3 se dă o reprezentare a modului în care este ocupată memoria sistemului cu tabela vectorilor de întreruperi. Amplasarea se realizează în funcție de octetul type, care dă nivelul de întrerupere.

Adrese	Octet type
0-3	0
4-7	1
...	...
1020-1023	255

Fig. 5.3.

Am văzut că acest octet poate lua 256 valori diferite.

Valoarea octetului type înmulțită cu 4 va furniza adresa din memorie unde se găseşte vectorul de întrerupere (adresa rutinei de întrerupere) pentru nivelul furnizat de octetul type.

Recapitulând, la apariția unei întreruperi în microprocesor au loc următoarele acțiuni:

- se salvează în stivă regiştrii (E)FLAGS, CS şi (E)IP, în această ordine
- se şterg bistabilii IF şi TF (adică se blochează execuția altei întreruperi în timpul execuției programului pentru întreruperea în curs, iar TF blochează execuția pas cu pas a rutinei de întrerupere)
- se furnizează microprocesorului un octet (8 biți în gama 0-255), numit şi octet type, care identifică nivelul asociat întreruperii
- se execută un salt la adresa de început a rutinei de tratare a întreruperii (folosind tabela vectorilor de întrerupere), de unde se începe execuția rutinei corespunzătoare de tratare a întreruperii

Fiecare componentă a calculatorului care poate să aibă nevoie de microprocesor la un moment dat are propriul său nivel de întrerupere. Tastatura, ceasul intern, unitățile de disc, imprimantele, placa de rețea etc., toate acestea au fiecare un nivel de întrerupere rezervat. La apăsarea unei taste se generează întreruperea de tastatură, la fiecare 55 milisecunde se generează întreruperea de ceas, a cărei rutină de tratare actualizează ceasul intern, discul trimite o întrerupere când este terminat un transfer iar imprimanta când nu are hârtie, placa de rețea când primeşte un pachet de date adresat ei ş.a.m.d. Deci observăm că activitatea microprocesorului se desfăşoară într-o permanentă posibilitate de a fi "întreruptă" de altcineva, care are mai mare nevoie de microprocesor decât aplicația ce rulează.

Interesant este că întreruperile nu au făcut parte din conceptul primelor calculatoare. La început calculatoarele au fost folosite fără acest mecanism. Astăzi este greu de imaginat un calculator fără sistem de întreruperi implementat în cadrul său hardware şi software.

Conceptul de întrerupere s-a dovedit atât de eficient şi util, încât a fost adaptat la o mare varietate de alte necesități ale calculatorului. Pe lângă întreruperile hardware, despre care tocmai am vorbit, există şi întreruperi care sunt generate chiar în interiorul CPU, ca urmare a faptului că s-a întâmplat ceva ce nu are sens - de exemplu s-a

încercat o împărțire la zero. În acest caz se generează o întrerupere internă, numită şi *excepție*.

5.1.2. Întreruperi software

Această categorie de întreruperi nu apare pe neaşteptate, precum cele hardware descrise anterior. Ideea care stă la baza întreruperilor a fost extinsă astfel încât acestea (întreruperile) să fie utilizate şi de către programe, pentru a solicita servicii executate de alte programe din calculator. Întreruperile de acest tip se numesc întreruperi software. După cum am văzut, atunci când se construieşte un calculator există un set de programe interne integrate într-o memorie ROM (Read Only Memory), care formează aşa-numitul BIOS. Dacă programele de aplicații au nevoie de funcții oferite de BIOS, modul de apelare a acestora îl constituie întreruperile software.

Serviciile BIOS sunt puse la dispoziția programului de aplicație prin execuția unei instrucțiuni de întrerupere software de tipul `INT n`, unde `n` reprezintă nivelul de întrerupere solicitat. Întreruperile software funcționează la fel ca şi celelalte tipuri de întrerupere, cu o singură diferență: ele nu sunt declanşate de un eveniment neaşteptat sau aleatoriu, ci sunt produse intenționat de către program cu un anumit scop.

Sistemele de operare folosesc de asemenea întreruperi software pentru apelul unor funcții necesare derulării programelor de aplicație sub controlul său direct. Aceste funcții ale BIOS sau ale sistemului de operare, apelate prin intermediul întreruperilor, sunt tratate de către procesor ca subprograme care, după ce se termină, redau controlul programului apelant. Programul care face apel la o asemenea funcție nu are nevoie să cunoască adresa de memorie a rutinei corespunzătoare, ci este suficient să indice numărul întreruperii alocate acelei funcții şi eventual parametrii auxiliari, necesari funcției. Aceste întreruperi sunt standardizate de către BIOS şi respectiv de către sistemul de operare.

Vom mai face câteva observații asupra modului de funcționare a întreruperilor într-un calculator.

Ultima instrucțiune dintr-o rutină de întreruperi este instrucțiunea IRET (Interrupt Return) care are rolul de a restaura în ordine inversă ceea ce a fost salvat în stivă, adică (E)IP, CS şi registrul (E)Flags, redând controlul programului principal.

Dacă rutina de întreruperi lucrează cu regiştrii procesorului şi distruge valorile conținute în acestea, revine în grija programatorului ca aceste valori să fie salvate explicit în stivă, prin instrucțiuni de tip PUSH, la inceputul rutinei de tratare a întreruperii, iar la sfârşit, înainte de terminare, să fie refăcuți aceşti regiştri prin instrucțiuni POP corespunzătoare. Astfel, programul care a fost întrerupt îşi reia lucrul cu valorile care erau în regiştrii procesorului la momentul întreruperii sale.

După cum am văzut, imediat ce s-a declanşat o întrerupere indicatorul IF este trecut pe 0, ceea ce înseamnă că întreruperile hard care pot surveni din acel moment sunt dezactivate. De aceea este indicat să se utilizeze cât mai repede posibil o instrucțiune de tip STI (Set Interrupt) care activează din nou sistemul de întreruperi - bineînțeles, dacă programul rutinei de întrerupere nu execută o porțiune în care nu are voie să fie întreruptă.

Tabela vectorilor de întreruperi, după cum am văzut, este plasată în memoria RAM (deci cea în care se poate şterge şi înscrie altă valoare). Aceasta face ca adresele rezervate în tabelă pentru desemnarea rutinelor de tratare a întreruperilor să poată fi schimbate chiar de unele programe şi eventual utilizatorii să-şi scrie propriile programe pentru tratarea unor întreruperi. Această poartă de intrare în sistemul de operare prin intermediul întreruperilor este folosită şi de unele programe răuvoitoare cum ar fi viruşii, caii troieni etc. De obicei aceste programe "fură" o întrerupere, adică

îşi introduc în tabela de întrerupere, în locul adresei normale de până atunci, propria lor adresă de început. La declanşarea normală a întreruperii respective se lansează acest program "pirat", care poate realiza diverse acțiuni distructive - ceea ce poate avea un efect catastrofal asupra integrității datelor în calculator. Pe cât de puternic este sistemul de întreruperi în activitatea unui pocesor, pe atât de mare poate fi pericolul folosirii necorespunzătoare a acestuia de către programe rău intenționate.

Sistemele de operare mai noi (Windows, Linux, OS/2) limitează accesul la partea de hardware tocmai din această cauză. Programele de aplicație nu mai pot accesa hardware-ul în mod direct, ci numai prin intremediul sistemului de operare, care blochează astfel un eventual apel rău intenționat, asigurând astfel un mai mare grad de siguranță.

6. Memoria

6.1. Tipuri de memorie

După cum am văzut la prezentarea arhitecturii von Neumann, memoria constituie unul din elementele de bază ale structurii unui sistem de calcul, rolul său fiind acela de a reține în primul rând programul şi datele care se prelucrează. La început, calculatoarele dispuneau de puțină memorie şi era nevoie de multă inventivitate din partea programatorilor pentru ca aceasta să ajungă pentru programe, care tindeau să devină tot mai lungi. Odată ce tehnologia a oferit posibilitatea ca memoria să poată fi obținută la un preț acceptabil, ea s-a diversificat încontinuu, apărând mai multe tipuri de memorie într-un calculator, fiecare din acestea având locul şi rolul său bine stabilit. Tehnologic există două tipuri principale de memorie:

- memorie RAM
- memorie ROM

A. **Memoria RAM**

Numele acesteia provine de la denumirea ei în engleză: Random Access Memory. Informația care se găseşte stocată în ea la diferite adrese (instrucțiuni sau date) poate fi citită sau înscrisă cu o nouă valoare. Se spune că suportă atât operații de citire (*read*), cât şi operații de scriere (*write*). Informația elementară care se memorează este o informație binară, putând deci lua valoarea 0 sau 1, iar circuitul fizic elementar care poate memora această informație se numeşte *bistabil* (are două stări stabile: 0 sau 1).

O primă formă de implementare fizică este memoria RAM de tip *static* sau SRAM (Static RAM), numită astfel deoarece informația odată înscrisă se păstrează nealterată până eventual la oprirea calculatorului, când se pierde. Bistabilii de tip SRAM sunt alcătuiți din doi tranzistori.

Folosind o tehnologie diferită, s-a reuşit ca pentru o celulă de memorie să se folosească un singur tranzistor (deci densitatea de informație memorată va fi dublă față de cea anterioară), obținându-se un tip de memorie numit *dinamic* sau DRAM (Dynamic RAM). Aceşti tranzistori pot să-şi piardă sarcina electrică pe care o înmagazinează (deci informația memorată în ultimă instanță) şi atunci este nevoie de o operație de reîmprospătare periodică (*refresh*), care se realizează cu ajutorul unor circuite concepute în acest scop (în general la fiecare 2 ms). Memoria de tip DRAM este considerabil mai lentă decât cea SRAM, dar are o densitate de integrare mai mare (acelaşi număr de celule de memorie ocupă mai puțin spațiu) şi este mai ieftină, motiv pentru care este folosită pe scară largă în sistemele de calcul.

Blocurile de memorie RAM au în general o organizare matriceală. Dacă la început aveau forma unor circuite integrate distincte, o dată cu evoluția tehnologiei aceste cipuri au fost plasate pe plăcuțe de memorie cu 30, 72, 162... picioare, iar capacitatea lor a crescut în timp de la 256Ko, 1Mo, 2Mo, ... până la 256Mo, 512Mo sau 1Go şi chiar mai mult.

B. **Memoria ROM**

Această memorie nu suportă decât citirea datelor din ea (ROM = Read Only Memory - memorie numai pentru citire). La fel ca şi în cazul memoriei RAM, capacitatea ei a crescut o dată cu evoluția tehnologiei, de la circuite de 1Ko sau 2Ko la 64Ko, 128Ko etc.

Inițial, informația era înscrisă într-un modul ROM la fabricarea acestuia şi nu mai putea fi schimbată. Avansul tehnologiei a permis realizarea unor circuite în care informația se poate şterge şi rescrie (desigur, nu de către procesor, ci cu ajutorul unor

dispozitive dedicate, care nu se găsesc în calculator). Spunem că aceste circuite sunt de tip ROM programabil sau PROM.

Cel mai des folosite sunt circuitele EPROM, la care scrierea informației se realizează pe cale electrică. Dacă ştergerea se face cu lumină ultravioletă printr-o fereastră de cuarț plasată deasupra cipului, avem circuite de tip UVEPROM. Dacă ştergerea se face electric, avem EEPROM. O variantă mai nouă a tehnologiei EEPROM este memoria de tip *Flash*, care este larg utilizată în diverse dispozitive de stocare. Un mare avantaj al acesteia din urmă îl reprezintă posibilitatea de a şterge sau înscrie doar o parte a informației memorate.

Numărul de ştergeri şi reînscrieri care pot fi aplicate asupra unui asemenea circuit, indiferent de tehnologia folosită, este limitat (în majoritatea cazurilor în jurul a 50-100). Excepție face memoria Flash, care permite până la 100000 de ştergeri şi reînscrieri. Oricum, limitarea nu este deranjantă, deoarece memoria ROM se foloseşte în calculator pentru memorarea programelor BIOS. Operația de actualizare a BIOS-ului poate fi necesară în unele situații, dar foarte rar; dincolo de acest caz particular, este chiar de dorit ca BIOS-ul să nu poată fi modificat de programe.

6.2. Memoria video

Una din cele mai importante interfețe dintre utilizator şi calculator o constituie monitorul. Dacă la început ecranul putea afişa numai caractere şi doar în mod monocrom, o dată cu dezvoltarea tehnologiei şi sistemul video care asigură afişarea a trecut prin transformări majore, fiind capabil ca pe lângă caractere să apară şi grafice, eventual în culori. De altfel, majoritatea sistemelor de operare actuale fac din această interfață grafică modul principal de existență.

Elementele principale ale sistemului de afişare sunt echipamentul de afişare (monitorul), adaptorul (controlerul) video şi memoria video.

Putem să definim ca memorie video zona de memorie accesată simultan de procesor şi de controlerul video, care la ieşire este capabilă să producă o secvență serială sincronă de informații capabile să comande un dispozitiv de tip CRT sau LCD.

Această memorie poate fi văzută într-o primă aproximație ca un registru uriaş de deplasare ce conține în el imaginea. Un punct de pe ecran se numeşte *pixel*. Acest punct poate avea anumite caracteristici (atribute) cum ar fi culoarea, strălucirea etc.

Imaginea care trebuie afişată este stocată într-un ecran virtual din memoria video. Controlerul video generează o imagine mişcând fascicolul de electroni de la stânga la dreapta şi de sus în jos pe ecran, similar cu cititul unei pagini. La sfârşitul unei linii orizontale fascicolul este stins şi mutat la începutul liniei următoare, baleind linia, ş.a.m.d. Această baleiere poartă numele de *rastru.*

Prentru fiecare poziție a unui pixel din rastru, datele de afişat sunt citite din ecranul virtual aflat în memoria video. Aceste date sunt aplicate la intrările unor circuite DAC (Digital-Analog Convertor), care le convertesc în nivele de tensiune pentru cele trei culori primare RGB (Red, Green, Blue) folosite în televiziunea color. După terminarea unui cadru, fascicolul se întoarce în stânga sus şi începe un nou cadru ş.a.m.d. Un astfel de tip de afişaj se mai numeşte şi afişaj APA (All Points Addressable).

Pentru ca imaginea să nu aibă efect de pâlpâire, care poate fi deranjant şi chiar vătămător pentru ochi, se impune ca frecvența de reîmprospătare a ecranului (numărul de cadre pe secundă) să fie mai mare de 70 Hz.

Au existat mai multe tipuri de controlere video:

- MDA (Monochrome Dispozitiv Adaptor), construit de IBM în 1981, o dată cu primul PC. Nu avea posibilități grafice, putând afișa numai caractere ASCII standard.

- Hercules, care a rezolvat problema îmbinării textului cu grafica pe același ecran. Placa monocromă HGC (Hercules Graphic Card) putea afișa 720 × 348 puncte monocolor pe ecran.

- CGA (Color Graphic Adaptor), produs de IBM, care putea ajunge la 640 × 200 puncte cu 16 culori.

- EGA (Enhanced Graphic Adaptor), cu rezoluție de 640 × 350 puncte.

- VGA (Video Graphic Array), cu 640 × 480 puncte și 256 culori.

- SVGA (Super Video Graphic Array), cu 1024 × 768 puncte afișabile.

Toate plăcile dintr-o serie pot în general lucra cu programe scrise pentru seriile anterioare. Mai sus am prezentat caracteristicile modului de funcționare grafic al acestor plăci, dar există și un mod de funcționare caracter. În acest mod se consideră că memoria conține coduri de caractere ASCII și nu puncte direct afișabile, iar un circuit special numit generator de caractere livrează pixeli din codul ASCII citit.

Vom mai reveni la placa de tip APA, insistând puțin și asupra afișajului de culoare al punctelor. Un monitor color poate afișa peste 16 milioane de culori, dar adaptorul video limitează această cifră. Dacă utilizăm pentru fiecare pixel câte un octet pentru a memora caracteristicile sale de culoare, putem avea 256 (2^8) culori. Dacă extindem memoria folosită pentru un pixel la 2 octeți, atunci sunt posibile 65536 de culori, iar dacă pentru fiecare culoare fundamentală (RGB) se rezervă un octet (deci 3 octeți pe pixel) depășim 16 milioane.

În terminologia calculatoarelor, modul de afișare cu 1 octet rezervat pentru un pixel se numește *pseudo-color*, cu 2 octeți *high color*, iar cu 3 octeți *true color*.

Pentru ca din informația numerică să se obțină un nivel de tensiune analogic corespunzător, în adaptoarele video se utilizează convertoarele numeric-analogice (DAC). Acestea sunt de obicei pe 8 biți și câte unul pentru fiecare culoare fundamentală. Deci am fi în stare să afișăm peste 16 milioane de culori, dar de obicei informația de culoare pentru aplicații grafice nepretențioase este cuprinsă pe un octet - deci numai 256 culori la un moment dat din cele 16 milioane posibile. Cele 256 culori alese pentru afișare din domeniul de 16 milioane formează așa-numita paletă de culori, iar conversia de la 8 biți la 24 biți se face în adaptorul video printr-o memorie numită CLUT (Color Look-Up Table - tabel de selecție a culorii). Valoarea pe 8 biți a octetului este considerată o adresă în tabela CLUT, care va seleca cuvântul de 3 octeți corespunzător ce va genera culoarea. Dacă se dorește schimbarea paletei, se vor memora alte valori în tabela CLUT pentru aceeași adresă.

În final, trebuie să amintim că plăcile grafice actuale pot avea o memorie video mai mare decât cea alocată în harta memoriei, dar numai o singură pagină se afișează la un moment dat. Pagina afișată este selectată cu un registru intern. De asemenea, plăcile cu accelerare video, care domină actualmente piața, sunt folosite atât pentru grafica în 2 dimensiuni (2D), cât și pentru cea în 3 dimensiuni (3D). Controlerele video au integrate unități cu instrucțiuni grafice speciale pentru prelucrarea și sinteza imaginilor, în timp ce procesorului îi revine numai sarcina de a apela aceste funcții integrate în controler. Câștigul de viteză este impresionant, deoarece aceste circuite specializate realizează prelucrările grafice mult mai rapid decât o poate face procesorul; în plus, sunt eliminate întârzierile legate de comunicarea între procesor și controlerul video.

Plăcile video pot folosi o parte din memoria sistemului sau pot fi dotate cu memorie proprie, care ajunge în unele cazuri până la 256 Mo. Dacă ținem cont de faptul că sunt rare calculatoarele care au 1 Go de memorie principală, este evident cât

de mare este necesarul de memorie al unor aplicații (în special jocuri). Plăcile fără memorie proprie sunt mai ieftine, dar diferența de performanță este mai mult decât semnificativă. Totuşi, este bine de reținut că nici măcar aceste cantități impresionante de memorie proprie nu sunt suficiente pentru unele aplicații, astfel încât până şi plăcile cele mai avansate folosesc şi o parte din memoria sistemului.

6.3. Memoria cache

Microprocesorul 80386 a fost primul care a depăşit viteza memoriei RAM. Cu alte cuvinte, dacă până atunci circuitele de memorie puteau servi cererile procesorului cu o viteză mai mare decât putea acesta să formuleze cererile de acces, începând cu 80386 situația s-a inversat: microprocesorul putea executa instrucțiunile mai rapid decât putea memoria să-i furnizeze datele ceute. Astfel, procesorul trebuia efectiv să aştepte sosirea datelor din memorie, neputând trece mai departe. Decalajul s-a adâncit în timp, astfel încât un microprocesor din zilele noastre este de peste 10 ori mai rapid decât memoria. Urmarea imediată este o penalizare majoră de perfomanță, deoarece procesorul nu-şi poate folosi integral capacitatea de calcul şi este chiar foarte departe de această situație. Într-adevăr, un microprocesor care funcționează la nici o zecime din viteza sa nu este foarte util.

Totuşi, lucrurile nu stau în întregime astfel. Tehnologia actuală permite realizarea unor circuite de memorie mai rapide, care pot face încă față cerințelor impuse de procesoare. Reamintim că, în afară de circuitele DRAM, utilizate în sistemele de calcul, există şi memoriile de tip SRAM, care sunt mult mai rapide. De fapt, tehnologia utilizată la fabricarea circuitelor SRAM este în mare măsură aceeaşi care se foloseşte şi pentru procesoare. Din păcate, tehnologia SRAM are două mari dezavantaje: o densitate de integrare redusă (ceea ce înseamnă circuite de dimensiuni mari) şi un preț prea ridicat, care o face nerentabilă.

Deşi circuitele SRAM nu pot fi folosite în locul celor DRAM din motivele arătate, avem posibilitatea de a le utiliza pe scară mai mică, obținând totuşi o creştere de performanță notabilă. Se porneşte de la două legi determinate empiric, dar a căror valabilitate este permanent confirmată de practică, şi anume *principiile (legile) localității*:

- localitate temporală: dacă o locație de memorie este accesată la un moment dat, este probabil că va fi accesată din nou în viitorul apropiat

- localitate spațială: dacă la un moment dat este accesată o locație de memorie, este probabil că în viitorul apropiat vor fi accesate locațiile din apropierea sa

Facem observația că principiile localității se aplică atât pentru accesul la date (o variabilă este de obicei accesată de mai multe ori în instrucțiuni consecutive, tablourile sunt de obicei parcurse secvențial), cât şi în ceea ce priveşte instrucțiunile executate (programele constând în principal din bucle). S-a ajuns astfel la ideea de a reține conținutul ultimelor locații accesate într-un circuit de memorie separat, foarte mic, numit memorie intermediară sau *cache*; atunci când se realizează un acces la memorie, mai întâi se caută locația respectivă în cache şi abia apoi, dacă este cazul, în memoria principală.

Datorită dimensiunii sale reduse, este posibil ca memoria cache să fie de tip SRAM, fără a afecta în mod semnificativ prețul total. În acest fel, viteza cache-ului este mult mai mare decat a memoriei principale.

Se observă că microprocesorul şi calculatorul pot funcționa şi în absența cache-ului. Folosirea memoriei cache nu aduce nimic în plus din punct de vedere al funcționalității, ci doar un spor de performanță.

Pentru a da o exprimare cantitativă principiilor localității, vom prezenta mai întâi terminologia folosită. Atunci când microprocesorul face un acces la memorie, caută mai întâi dacă nu cumva adresa dorită se găseşte în memoria cache. Dacă găseşte informația, nu se mai face apel la memoria RAM şi se trece mai departe. Spunem în acest caz că s-a realizat un "hit". Dacă informația nu există în cache, se face un apel la memoria principală (am avut un rateu - "miss").

Introducem o mărime numită rata de succes a cache-ului (*hit ratio*), notată H. Aceasta exprimă procentajul (din numărul total de accese la memorie efectuate în unitatea de timp) de cazuri în care informația căutată a fost găsită în cache. Evident, valoarea H nu este o constantă, ci se determină prin măsurători în cazuri practice şi depinde de comportarea programelor care se execută. Similar se defineşte rata de insucces M (*miss ratio*); cele două marimi se află în relația:

$M = 1 - H$

Notăm cu T_c timpul de acces la cache (altfel spus, timpul total de acces în cazul cand valoarea căutată se află în cache) şi cu T_m timpul necesar pentru a accesa locația căutată în cazul în care aceasta nu se află în cache. Cu aceste notații, timpul mediu de acces la memorie este:

$T = T_c \cdot H + T_m \cdot M$

Utilizarea cache-ului este eficientă dacă acest timp mediu este mai mic decât timpul de acces la memoria principală în cazul în care nu există memorie cache (pe care îl notăm T_p). Trebuie observat că, în cazul în care locația căutată nu se află în cache, se fac în total doua accese: întâi la cache (acces eşuat), apoi la memoria principală. Astfel, T_m este mai mare decât T_p. În aceste condiții, este posibil ca $T > T_p$, ceea ce ar însemna ca de fapt cache-ul sa frâneze accesul la memorie, în loc de a-l accelera. În practică însă acest risc nu există. Deoarece în general T_c este mult mai mic decât T_p, T_m este doar puțin mai mare decât T_p, iar $H \geq 90\%$, câştigul de viteză adus de cache este în general considerabil.

Pentru exemplificare, considerăm un sistem cu următoarele caracteristici: $H = 90\%$, $T_c = 2ns$, $T_m = 21$ ns, $T_p = 20$ ns. Aplicând formula de mai sus obținem:

$T = T_c \cdot H + T_m \cdot M = 2ns \cdot 90\% + 21ns \cdot 10\% = 1{,}8ns + 2{,}1ns = 3{,}9ns \Rightarrow T_p / T_c \cong 5{,}13$

Timpul mediu de acces la memorie este deci de peste 5 ori mai mic decât în cazul în care nu ar fi existat cache în sistem. În realitate situația este şi mai favorabilă deoarece, dacă relațiile între diferiții timpi de acces sunt aproximativ aceleaşi ca în exemplul de mai sus, rata de succes a cache-ului ajunge adesea până la 98% şi chiar mai mult.

Să intrăm puțin în detaliile proiectării unui cache. În primul rând, atunci când o locație din memoria principală este cerută de procesor şi nu există deja în cache, este adusă. Deşi procesorul nu are nevoie pe moment decât de locația respectivă, în cache este adus un bloc de date de dimensiune mai mare (uzual 16, 32 sau 64 octeți), format din locații aflate la adrese consecutive în memoria principală, în jurul locației solicitate de procesor. Un asemenea bloc memorat în cache poartă denumirea de *linie de cache*. Motivul pentru care se procedează astfel este evident: în acest mod se ține cont şi de localitatea spațială, nu doar de cea temporală.

În cache se pot afla simultan date care în memoria principală se găsesc la adrese complet diferite. Din acest motiv, este necesar ca în cache să fie reținute nu doar valorile locațiilor, ci şi adresele la care acestea se află în memoria principală. Acest aspect este esențial, deoarece căutarea datelor în cache nu se face ca într-o memorie obişnuită, ci după adresele ocupate în memoria principală. Modul în care sunt

memorate aceste adrese are o influenţă foarte puternică asupra vitezei de regăsire a datelor căutate, deci a timpului de acces la cache.

Strâns legată de problema de mai sus este politica de înlocuire. Datorită capacităţii incomparabil a cache-ului faţă de memoria principală, în mod inevitabil se ajunge la ocuparea totală a cache-ului. Atunci când un nou bloc de date trebuie adus în cache, va trebui deci eliminat un bloc deja existent. Alegerea blocului care va fi înlocuit este o problemă dificilă, deoarece trebuie evitată înlocuirea unui bloc de care ar putea fi nevoie în viitorul apropiat, altfel performanţa globală este afectată.

Există în prezent trei tipuri principale de cache, diferenţiate prin metoda de memorare a adreselor din memoria principală:

a. Cache cu adresare directă (*direct mapped cache*)

În acest caz există o relaţie directă între adresa din memoria principală unde se află o valoare şi adresa din cache în care aceasta este memorată. Mai exact, biţii cei mai puţin semnificativi ai adresei din memoria principală formează adresa din cache.

Ca un exemplu concret, considerăm un sistem al cărui procesor lucrează cu adrese (în memoria principală) de 32 biţi, iar cache-ul are o capacitate de 2048 linii, fiecare linie având 32 octeţi. Se observă că atât numărul de linii, cât şi dimensiunea unei linii de cache sunt puteri ale lui 2, ceea ce uşurează operaţiile executate de către hardware. O adresă de 32 biţi din memoria principală este împărţită în 3 componente:

- cei mai semnificativi 16 biţi formează o etichetă, care este memorată ca atare în cache, împreună cu datele propriu-zise aduse din memoria principală
- următorii 11 biţi indică adresa liniei din cache care memorează datele, din cele 2048 linii existente
- ultimii 5 biţi identifică octetul în cadrul liniei de cache

Se observă că o anumită adresă din memoria principală poate fi memorată într-o singură adresă din cache.

Conţinutul unei linii de cache este următorul:

- un bit care indică dacă linia conţine date valide
- câmpul etichetă, descris mai sus
- datele propriu-zise aduse din memoria principală

Datorită modului de calcul, linia cu adresa **N** din cache poate memora date provenite de la orice adresă din memoria principală ai cărei biţi de pe poziţiile 5-15 formează valoarea **N**. Din acest motiv, pentru a putea determina în orice moment adresa corespunzătoare din memoria principală, în linia respectivă este memorat şi câmpul etichetă.

Cache-ul cu adresare directă permite un acces extrem de rapid, deoarece conversia între cele două tipuri de adrese (din memoria principală şi din cache) este foarte simplă şi poate fi implementată direct în hardware. În schimb, algoritmul are dezavantajul lipsei de flexibilitate. Pentru exemplul de mai sus, dacă un program accesează foarte des mai multe variabile aflate în memoria principală la adrese care diferă printr-un multiplu de 65536 (având deci ultimii 16 biţi identici), aceste variabile vor fi memorate la aceeaşi adresă în cache; ca rezultat, se vor înregistra multe ratări în cache, ceea ce implică multe accese la memoria principală, deci scăderea vitezei.

b. Cache asociativ (*fully associative cache*)

Se bazează pe utilizarea unor circuite hardware speciale, numite memorii asociative. Spre deosebire de memoria obişnuită, care permite doar citirea sau scrierea unei valori într-o locaţie identificată prin adresa sa, memoria asociativă permite în plus regăsirea unei locaţii după conţinutul său.

ÎIntr-un cache asociativ, fiecare linie reţine, pe lângă datele propriu-zise, adresa de început a acestora în memoria principală. Regăsirea se va face pe baza acestei

adrese. Întrucât memoriile asociative sunt relativ lente, accesul la cache este mai puțin rapid decât în cazul cache-ului cu adresare directă. Pe de altă parte, avantajul este că o locație din memoria principală poate fi memorată la orice adresă din cache, eliminându-se problemele de genul celei prezentate mai sus.

c. Cache parțial asociativ (*set-associative cache*)

În ciuda numelui său, este mai apropiat ca structură de cache-ul cu adresare directă. Principalul dezavantaj al acestuia, aşa cum s-a văzut, îl constituie faptul că mai multe adrese din memoria principală concurează pentru aceeaşi adresă din cache. Soluția propusă este următoarea: fiecare adresă din cache conține mai multe linii (uzual 4, 8 sau 16), fiecare cu propriile date, propriul bit de validare şi propriul câmp etichetă. Astfel, un cache asociativ cu *n* căi (linii) permite memorarea simultană a *n* locații din memoria principală care în cazul cache-ului cu adresare directă ar fi concurat pentru aceeaşi adresă în cache. Apare o creştere a timpului de acces, deoarece atât la scriere, cât şi la citire trebuie verificate toate cele *n* căi. În schimb, utilizarea unui număr relativ redus de căi elimină practic total riscul apariției conflictelor.

Vom ridica acum o ultimă problemă. Până acum s-a discutat în mod implicit mai mult de citirea datelor din memorie. La modificarea unei valori care se află deja în cache trebuie să decidem în care din cele două memorii (principală şi cache) se va realiza scrierea. Avem de ales între două politici posibile:

- *write-back* - datele sunt scrise numai în cache; evident, ele vor ajunge şi în memoria principală, dar numai la eliminarea lor din cache

- *write-through* - datele sunt scrise atât în memoria principală, cât şi în cache

Ambele politici sunt larg utilizate, fiecare având avantaje şi dezavantaje. Politica *write-back* este mai rapidă, în schimb pune probleme majore în sistemele multiprocesor, deoarece o modificare făcută în cache-ul unui procesor nu ar putea fi cunoscută de celelalte procesoare. În acest caz sunt necesare protocoale hardware complexe, prin care fiecare cache "ascultă" în permanență magistrala comună, pentru a detecta modificările făcute de celelalte procesoare.

Analizând funcționarea memoriei cache putem formula un principiu mai general: întotdeauna când avem de accesat o sursă de date cu dimensiuni mari şi viteză de acces redusă, putem obține un spor semnificativ de perfomanță dacă interpunem între sursa de date şi "beneficiar" (cel care accesează datele) o formă de stocare mai mică, dar mai rapidă, care să rețină ultimele date aduse de la sursă. Principiul este într-adevăr folosit pe scară largă şi în alte situații, nu doar în cazul procesorului. În continuare vom prezenta câteva asemenea exemple de materializare a conceptului de cache, implementate hardware sau software, luate din activitatea curentă a unui utilizator de PC.

Cache-ul de disc. Deoarece memoria principală este mult mai rapidă decât discul, toate sistemele de operare folosesc o zonă de memorie drept cache pentru operațiile cu sistemul de fişiere. Evident, principiile localității operează la fel de frecvent şi în cazul operațiilor cu discul. Există o singură diferență notabilă: deoarece scopul memorării pe disc este în primul rând de a face datele persistente, conținutul cache-ului este scris în mod periodic pe disc (în general la fiecare 30 de secunde). În acest fel se evită riscul pierderii informațiilor în cazul eventualelor căderi de tensiune sau blocări ale sistemului.

DNS. Sistemul DNS (Domain Name System) de pe rețeaua Internet este format dintr-o serie de cache-uri software, răspândite pe anumite servere, care rețin corespondențe între adrese literale şi IP, pentru creşterea vitezei de căutare a unor site-uri.

Browserele web. Clienții WWW, cum ar fi Internet Explorer, memorează într-un cache software adresele vizitate şi pe unde s-a trecut şi paginile încărcate, pentru ca la o nouă încercare de accesare a acestora să se ia informația citită din cache în locul unui apel către serverul aflat la distanță.

6.4. Memoria virtuală

Existența mai multor programe simultan în memorie (pentru a permite execuția lor în paralel) provoacă o creştere foarte mare a necesarului de memorie. Un prim pas în îmbunătățirea situației a fost făcut, aşa cum s-a văzut deja, prin asigurarea independenței spațiilor de adrese virtuale ale programelor; în acest fel, în cazul procesoarelor actuale, fiecare program are la dispoziție adrese virtuale pe 32 de biți, ceea ce este permite un spațiu de adrese de 4 Go, suficient pentru aplicațiile actuale. În schimb este posibil ca memoria fizică să fie insuficientă pentru necesarul tuturor programelor care se execută în paralel. Pentru a relaxa cât mai mult această limitare, se porneşte de la următoarele constatări:

- nu toate paginile de memorie sunt necesare la un moment dat
- discul hard are în general o capacitate mult mai mare decât memoria fizică disponibilă, deci poate fi utilizat pentru a reține temporar conținutul unora dintre paginile de memorie

Discuția care urmeaza este valabilă în egală măsură atât pentru segmentarea memoriei, cât şi pentru paginare. Vom considera cazul paginării, care este mult mai larg utilizată de sistemele de operare actuale.

Modul de lucru ar fi deci următorul: în orice moment, în memoria fizică se află o parte dintre paginile virtuale ale programelor aflate în execuție; paginile care nu au loc în memoria fizică sunt memorate pe disc într-un fişier special, numit *fişier de paginare*. Se utilizează tot mecanismul de paginare descris anterior, dar uşor modificat:

- Dacă pagina din care face parte adresa căutată se găseşte în memoria fizică (adică apare în tabelul de paginare), totul se desfăşoară în modul descris la prezentarea paginării.
- Dacă pagina virtuală respectivă nu se află în memoria fizică, se caută pagina dorită în fişierul de paginare. Bineînțeles, acest fişier conține şi informațiile necesare pentru a putea fi regăsite paginile pe care le stochează.
- Dacă pagina nu se găseşte nici în fişierul de paginare, atunci avem o eroare de adresare şi accesul la memorie este oprit.
- Dacă pagina căutată se află în fişierul de paginare, va fi adusă în memoria fizică. În general memoria fizică este complet ocupată, de aceea o altă pagină aflată în memoria fizică va fi evacuată şi memorată în fişierul de paginare. Abia când pagina dorită a fost adusă în memoria fizică şi tabelul de paginare al programului a fost modificat corespunzător, se poate realiza accesul propriu-zis.

În acest mod se pot executa în paralel programe al căror necesar total depăşeşte memoria existentă în sistem. Există în continuare limitarea impusă de dimensiunea discului, dar aceasta este mai puțin severă.

Totuşi, soluția nu are numai părți bune. Accesul la disc este incomparabil mai lent decât cel la memorie, astfel că pierderea de performanță este de multe ori vizibilă chiar pe calculatoarele cele mai puternice. Din acest motiv se caută să se reducă la minimum accesele la fişierul de paginare. În primul rand, o pagină virtuală nu va fi scrisă pe disc decât dacă nu mai este loc în memoria fizică. Altfel spus, acest

mecanism nu este utilizat decât dacă este neapărat necesar, ceea ce i-a adus denumirea de *paginare la cerere* (demand paging).

Ca o optimizare suplimentară, dacă o pagină care trebuie evacuată din memorie nu a fost modificată de când a fost adusă ultima dată în memorie, atunci copia sa din fişierul de paginare este identică, deci scrierea înapoi pe disc nu mai este necesară. Evident, în acest caz este necesar un sprijin suplimentar din partea hardware-ului, astfel încât în tabelul de paginare, pentru fiecare pagină, să fie memorat şi actualizat în permanenţă un bit suplimentar care să arate dacă pagina a fost modificată de când se află în memorie. Această optimizare este în mod special eficientă pentru paginile de cod, deoarece instrucţiunile nu sunt în general modificate pe durata execuţiei programelor.

În al doilea rând, atunci când o pagină trebuie adusă în memoria fizică şi nu mai este loc, pagina care va fi evacuată nu trebuie aleasă la întâmplare. Algoritmul utilizat pentru a selecta pagina care va fi evacuată pe disc trebuie să respecte o cerinţă clară: să minimizeze riscul ca pagina aleasă să fie accesată foarte curând în viitor, deci să trebuiască să fie reîncărcată în memoria fizică. Deoarece nu se poate prevedea care pagini vor fi accesate în viitor, există diverşi algoritmi care încearcă să prezică aceasta pe baza comportării programelor în trecutul apropiat.

6.5. Ierarhia de memorii

Dacă analizăm organizarea unui calculator, vedem că "inima" acestuia este o unitate de prelucrare a informaţiilor (localizată în procesor), înconjurată de o serie de circuite al căror rol este, în ultimă instanţă, de a memora informaţiile în diverse forme. Aceste circuite de memorare sunt organizate pe mai multe nivele, într-o structură ierarhică, în funcţie de distanţa faţă de unitatea de prelucrare. Pe măsură ce se depărtează de procesor, nivelele de memorie au o capacitate mai mare, dar şi o viteză mai mică. Putem distinge, în principiu, patru nivele ale ierarhiei de memorii:

- Nivelul regiştrilor procesorului. Aceştia au, în mod evident, cel mai mic timp de acces, aflându-se e acelaşi circuit cu unitatea de prelucrare. Este deci de preferat ca aplicaţiile să utilizeze cât mai mult posibil regiştrii, pentru a mări performanţa. Totuşi, numărul acestora este redus, astfel încât este practic imposibil ca o aplicaţie să se poată executa exclusiv cu ajutorul regiştrilor, fără a face deloc apel la nivelele următoare de memorare. Mai mult, codurile instrucţiunilor nu pot fi reţinute în regiştri.

- Nivelul memoriei cache (numită şi memorie imediată). Este singurul nivel care poate lipsi, fără ca aceasta să implice o schimbare în programele care rulează. Atât lucrul cu regiştrii procesorului, cât şi accesarea nivelelor următoare necesită o formă de gestiune prin software; nu este şi cazul memoriei cache. La rândul său, memoria cache poate fi împărţită pe nivele: poate exista un modul cache chiar în interiorul procesorului (numit cache L1), foarte mic şi care funcţionează practic la viteza procesorului, şi un altul pe placa de bază (cache L2), fabricat tot în tehnologie SRAM, care este puţin mai mare decât cache-ul L1 şi puţin mai lent. Unele implementări pot lucra chiar cu 3 nivele de cache, dintre care 2 sunt integrate în procesor.

- Nivelul memoriei principale. Deşi aici poate fi inclusă şi memoria ROM, în practică se are în vedere doar memoria RAM, deoarece prelucrarea informaţiei înseamnă implicit modificarea acesteia.

- Nivelul memoriei secundare. Acest nivel are caracteristica de stocare persistentă. Spre deosebire de nivelele anterioare, care sunt volatile, la acest nivel informaţiile se păstrează şi după întreruperea alimentării calculatorului. Tot la nivelul

memoriei secundare se găseşte memoria virtuală. Formele de implementare a memoriei secundare sunt: discul dur (cel mai folosit), discheta, mediile optice (CD, DVD), banda magnetică etc.

7. Sistemul I/O

Teoretic, un sistem format numai din procesor şi memorie poate funcţiona singur la infinit. Memoria conţine instrucţiunile programului de executat şi datele care trebuie prelucrate, iar procesorul prelucrează datele pe baza instrucţiunilor citite din memorie. Motivul pentru care nu va exista niciodată un calculator cu această structură minimală este simplu: activităţile realizate de un asemenea sistem ar fi inutile, pentru că nimeni nu ar beneficia de rezultatele lor. Comunicarea cu exteriorul (şi în principal cu utilizatorul) nu este deci o simplă opţiune; în absenţa acesteia, existenţa calculatorului nu ar avea sens. Echipamentele care realizează, în diferite forme, această comuincare se numesc dispozitive de intrare/ieşire (I/O) sau periferice. Diversitatea remarcabilă a acestor dispozitive reflectă de fapt varietatea sarcinilor pe care le poate îndeplini un calculator.

7.1. Porturi

Comunicarea între procesor şi dispozitivele periferice ridică problema conectării fizice. Perifericele fiind în număr atât de mare şi atât de diferite între ele, este necesar să existe o standardizare a modului de conectare la procesor, implicit şi a modului de comunicare. În practică, toate componentele calculatorului (procesorul, memoria, perifericele) sunt conectate între ele prin intermediul plăcii de bază. De modul în care este realizată placa de bază depind tipurile de conexiune disponibile.

Un periferic se conectează la placa de bază (şi indirect la procesor) prin intermediul unor conectori specializaţi, numiţi *porturi*. Fiecare port respectă un anumit standard de conectare. Există mai multe asemenea standarde, plăcile de bază putându-le implementa pe toate sau numai o parte dintre ele. Principalele standarde de conectare sunt:

Interfaţă paralelă

Permite transmiterea către periferic a câte unui octet de date într-o operaţie de transfer. Semnalele definite de acest standard sunt de 3 tipuri:

- liniile de date, care permit transmiterea octetului de date de la procesor către periferic
- liniile de control, prin care procesorul transmite anumite comenzi către periferic, permiţând desfăşurarea în bune condiţii a transferului
- liniile de stare, prin care perifericul transmite procesorului informaţii despre starea sa curentă

Modul de lucrul decsris mai sus, numit SPP (Standard Parallel Port), a fost conceput pentru comunicarea cu imprimantele. În momentul în care a apărut cerinţa conectării şi a altor tipuri de dispozitive, standardul nu a mai corespuns, în principal deoarece nu permitea transferul de date decât într-un singur sens. Ca urmare, a fost propus standardul EPP (Enhanced Parallel Port), care reprezintă o extindere a SPP; în afară de creşterea vitezei de transfer, principala sa îmbunătăţire a fost, cum era de aşteptat, posibilitatea ca şi perifericul să transmită date către procesor. Standardul EPP permite astfel conectarea unei game largi de periferice, cum ar fi scanerele, discurile hard şi unităţile CD externe etc. Pentru perifericele mai performante a fost elaborat şi un standard cu carcateristici superioare EPP, numit ECP (Extended Capabilities Port); totuşi, conceptual nu există diferenţe majore între EPP şi ECP. Astăzi, imprimantele folosesc şi ele facilităţle oferite de modurile EPP şi ECP, nemaifiind compatibile cu mai vechiul SPP.

Interfaţa serială

Spre deosebire de portul paralel, în cazul portului serial există o singură linie de date, deci se poate transmite un singur bit la un moment dat. Din acest motiv şi datorită modului mai sofisticat de gestiune a comunicației, viteza interfeței seriale este sensibil mai mică decât cea a interfeței paralele. În schimb, portul serial a fost proiectat de la început pentru comunicații bidirecționale.

Portul serial este folosit în general pentru conectarea unor periferice cum ar fi mouse-ul, modemul, precum şi alte periferice relativ lente. În ultimii ani, tendința este de înlocuire a porturilor seriale cu standardul USB, care este mult mai flexibil şi performant.

USB

Standardul USB foloseşte tot comunicația serială, dar a fost proiectat să exploateze avantajele tehnologiei moderne. Deşi inițial a fost destinat perifericelor lente, o dată cu apariția versiunii 2.0 a standardului viteza de transfer a crescut sensibil, depăşind cu mult performanțele interfețelor serială şi paralelă. Practic, astăzi nu există periferice care să nu aibă şi variante cu conectare pe portul USB (de multe ori acesta este singurul standard acceptat).

FireWire (IEEE 1394)

Este o interfață destinată perifericelor de foarte mare viteză, pentru care performanțele standardului nu sunt suficiente. Cel mai adesea este întâlnită la camere video digitale, care au de transferat volume mari de date către calculator. Interfața FireWire nu este foarte răspândită, datorită prețului mai mare şi faptului că standardul USB oferă suficientă performanță pentru majoritatea perifericelor.

ATA

Este o interfață de tip paralel pentru conectarea discurilor hard şi a unităților optice (CD, DVD). Originea sa este standardul IDE, elaborat în anii '80. De-a lungul timpului caracteristicile sale au evoluat, rata de transfer crescând spectaculos. În ultima vreme, deşi majoritatea plăcilor de bază încă mai sunt prevăzute cu porturi ATA, pierde teren în fața standardului SATA.

SATA

Este un standard derivat din ATA (şi destinat aceluiaşi tip de periferice), dar cu o interfață de tip serial. Rata de transfer a interfeței este mai mare decât cea a standardului ATA şi va continua să crească în versiunile viitoare. În plus, standardul ATA oferă facilitatea numită *bus mastering*, prin care controlerul de disc să comande magistrala, degrevând procesorul de sarcina gestiunii transferului şi mărind astfel performanțele.

SCSI

Este cel mai vechi standard pentru discurile hard (deşi permite în principiu conectarea oricărui tip de periferic). Versiunile sale succesive au dus la creşterea continuă a performanței, fie prin mărirea lățimii de bandă (volumul de date care poate fi transferat printr-o singură operație), fie prin creşterea frecvenței de lucru. Controlerele SCSI au folosit dintotdeauna bus mastering, iar tehnologia permite obținerea unor rate de transfer net superioare standardelor concurente. Însă, datorită prețului mare, interfața SCSI nu este destinată calculatoarelor personale, ci stațiilor de lucru şi serverelor.

PCI

Reprezintă un standard de conectare destinat plăcilor de extensie. A înlocuit standardul mai vechi ISA, care a fost abandonat de producători după două decenii de utilizare. Frecvența de operare a crescut în timp, ajungând până la 133 MHz, la fel şi lățimea de bandă. Principalele tipuri de periferice care utilizează interfața PCI sunt plăcile video, plăcile de sunet, modemurile interne etc. În ultima vreme, performanțele

oferite de PCI încep să fie considerate insuficiente, mai ales pentru plăcile grafice. Un standard derivat din PCI, numit PCI Express, a început să fie folosit de plăcile de bază de vârf, dar înlocuirea completă a interfeței PCI nu se întrevede în viitorul apropiat.

AGP

Este un standard conceput special pentru deservirea plăcilor grafice. Deoarece într-un calculator există mai multe sloturi PCI, perifericele care ocupă aceste sloturi trebuie să împartă între ele aceeaşi cale de comunicare cu procesorul. Plăcile grafice fiind mari consumatoare de resurse, interfața AGP le oferă o cale de comunicare privilegiată cu procesorul, pe care nu o împart cu alte periferice. Deşi, la fel ca în cezul celorlalte standarde, performanțele au crescut cu fieare versiune, AGP este depăşit de noul standard PCI Express, astfel încât viitorul său este nesigur.

PCMCIA

A fost conceput special pentru sistemele portabile. Datorită miniaturizării, într-un laptop nu există suficient spațiu pentru a dispune de sloturi PCI şi, în general, nici o formă de a conecta periferice interne. Standardul PCMCIA permite conectarea de periferice externe de orice tip, având dimensiuni mici. Ca atare, sloturile PCMCIA reprezintă practic singura posibilitate de a extinde funcționalitatea unui laptop.

8. Multimedia

Fenomenul de "multimedia PC" poate să aibă mai multe interpretări. Aici vom conveni să numim prin acest termen un set de tehnologii care fac posibilă existența aplicațiilor de tip multimedia, cum ar fi: grafică PC, imagini şi animație 2D şi 3D, video, redare directă sau a imaginilor înregistrate şi comprimate, precum şi aplicațiile legate de sunet (înregistrarea şi redarea sunetului, precum şi sinteza vorbirii). Alături de acestea trebuie să amintim şi o serie de tehnologii suport pentru multimedia, cum ar fi CD-ROM şi DVD, rețele locale şi tehnologii de comprimare/decomprimare a datelor. Acest domeniu s-a dezvoltat o dată cu creşterea performanțelor microprocesoarelor, care sunt acum capabile de a prelucra în timp real fluxul de date dintr-o astfel de aplicație.

Vom căuta să explicăm aceste noțiuni făcând apel la câteva aplicații multimedia importante, principiile enunțate putând fi extinse şi la celelalte, netrecute în revistă aici.

Unele din cele mai folosite aplicații multimedia folosite pe calculator sunt jocurile care solicită animație, grafică 3D în timp real, redare video, intrări de date din partea jucătorilor şi redarea de sunet înregistrat sau sintetizat. Educația şi instruirea sunt alte aplicații multimedia care pot solicita aceleaşi mijloace ca şi jocurile. Prezentările făcute pe calculator îşi găsesc utilizarea din ce în ce mai mult în ultimul timp. Videoconferințele folosesc metode cuprinse în acest capitol. Simulările, realitatea virtuală şi comanda calculatorului cu ajutorul vocii completează multitudinea de aplicații legate de această tehnologie.

8.1. Tehnologia multimedia audio

8.1.1. Elemente de bază ale sunetului digital

După cum se ştie, sunetul reprezintă o oscilație care variază continuu în amplitudine (ceea ce determină nivelul sonor) şi/sau în frecvență (ceea ce va determina tonul sunetului). În sistemele analogice, acest sunet este amplificat în circuite electronice, cu tuburi sau tranzistoare, rezultând o tensiune sau un curent variabil, care în final se aplică unui difuzor cu rolul de a-l transforma din nou într-un sunet perceput de ureche. Transformările pe care le suferea sunetul de-a lungul acestui lanț erau cele aplicate acestor oscilații electrice.

Dacă dorim ca sunetul să fie prelucrat într-un calculator, acesta va trebui să transforme mai întâi informația analogică (variația unei tensiuni) în informație digitală (şiruri de numere care reprezintă variația tensiunii inițiale). Această transformare se face cu ajutorul unui dispozitiv numit convertor analogic-digital (ADC).

După ce sunetul se va prezenta ca o secvență digitală, calculatorul va putea să prelucreze această informație după algoritmul cerut, iar rezultatul obținut va fi semnalul digital care va fi reconvertit în sunet de un convertor digital-analogic (DAC).

Dispozitivul numit de noi ADC va transforma semnalul analogic în semnal digital prin eşantionarea valorii semnalului cu o anumită frecvență. Totul apare ca şi cum s-ar realiza nişte instantanee digitale ("fotografii") ale semnalului analogic cu o frecvență foarte mare. Cu cât vor fi mai multe eşantioane într-o secundă şi acestea vor fi mai precis aproximate, cu atât semnalul digital rezultat va reproduce mai fidel semnalul analogic original.

Urechea umană poate sesiza semnale audio în domeniul 20-20000kHz. O teoremă din teoria analizei semnalelor arată că frecvența de eşantionare trebuie să fie mai mare decât dublul frecvenței cele mai mari (deci 40000 Hz în cazul nostru).O altă problemă este pe câți biți reprezentăm dimensiunea eşantionului. Dacă folosim un octet, adică 256 valori (2^8 = 256 - cum am folosit la placa grafică pentru a reprezenta maximum 256 culori pe ecran), vom avea maximum 256 nivele de amplitudine. La redare se va pierde mult din sunetul original. Dacă pentru reprezentarea amplitudinii unui eşantion vom folosi 2 octeți (2^{16} = 65536 valori), acest număr mare de nivele va aproxima cu fidelitate acceptabilă semnalul original.

Având aceste noțiuni despre semnalul digital, putem spune că, de exemplu, semnalul telefonic digital are o frecvență de eşantionare de 8000 Hz şi foloseşte un octet (8 biți) pentru reprezentarea amplitudinii lui, urmărindu-se în primul rând înțelegerea mesajului şi nu chestiuni legate de fidelitatea sa. La înregistrarea digitală a sunetului pe CD se folosesc o frecvență de 44100 Hz şi 2 octeți (16 biți) pentru fiecare eşantion. Dacă se înregistrează semnal stereo, se vor folosi încă 2 octeți pentru al doilea canal. Cunoscând aceste date se poate calcula rata de date pe minut pentru fiecare semnal digital prezentat. Dacă la semnalul telefonic se ajunge la aproximația 1 Mo/minut, la cel pentru CD se obțin peste 10 Mo/minut.

Dacă funcțiile blocurilor ADC şi DAC sunt combinate într-un singur circuit, acesta se va numi *codec* (Codare-DECodare). Pe lângă funcția de conversie, aceste circuite mai pot şi comprima sau decomprima date audio digitale.

În concluzie, sunetele în calculator sunt reprezentate în final ca fişiere şi deci se bucură de toate proprietățile şi posibilitățile de prelucrare specifice acestora: comprimare, decomprimare, prelucrare numerică etc.

Dacă extindem noțiunile la domeniul video, unde informația vizuală apare tot ca un semnal electric oscilant, tot aceea ce s-a spus la sunet rămâne valabil, dar cu alte rate de eşantionare

8.1.2. Prelucrări ale sunetului digital. Plăci de sunet

Odată ce sunetul a fost convertit în formă digitală, el poate fi prelucrat pentru a se crea tot felul de efecte ca reverberații, ecouri, distorsiuni controlate etc. Calculele necesare acestora sunt făcute în procesoare specializate numite DSP (Digital Signal Processor). Tot acestea pot asigura şi sinteza sunetului sau a muzicii, precum şi funcțiile de comprimare şi decomprimare.

Cercetările întreprinse în domeniul sintezei sunetelor au permis generarea acestora din însumarea mai multor semnale sinusoidale cu frecvențe diferite. Un capitol special din matematică se ocupă cu analiza armonică a semnalelor; folosind rezultatele acestor analize s-a reuşit sinteza sunetului prin modularea în frecvență (FM).

Toate acestea au dus la apariția plăcilor de sunet, care reprezintă un element important al posibilităților multimedia legate de sunet. Placa de sunet a devenit o prezență curentă în calculatoarele actuale. Prima placă de sunet a fost creată de firma Creative Labs şi poartă denumirea de *Sound Blaster*. Aceste plăci se cuplează normal pe un conector de extensie al magistralei ISA sau PCI şi cuprind unele blocuri deja amintite. În plus, observăm un bloc mixer, care poate accepta intrări analogice de la, microfon, linie audio sau difuzor PC, pe care le poate controla individual. De asemenea, blocul MIDI (Musical Instrument Digital Interface) primeşte comenzi pentru selectarea unor instrumente sau efecte audio.

8.2. Prelucrări digitale video

După cum am amintit deja, prelucrarea semnalelor video preluate de camerele digitale sau semnalul TV urmează aceleaşi principii ca şi cele de sunet, dar la o altă scară. Semnalul video transformat în semnal digital poate fi comprimat pentru a ocupa un spațiu mai mic la stocare sau în procesul de transmitere. La redare se desfăşoară procesele inverse. În plus, aici apar unele elemente noi. Astfel, s-au imaginat metode specifice de comprimare, care țin cont de faptul că în realitate conținutul imaginii de la un cadru la altul se schimbă foarte puțin, transmițându-se eventual numai schimbările survenite şi păstrând ca bază un cadru inițial. Este ceea ce fac metodele cunoscute sub numele de MPEG. Pentru imagini statice sunt cunoscute fişierele cu extensia *.JPG, ce provin din folosirea metodei JPEG de comprimare a imaginilor. Sateliții de comunicații destinați transmisiilor TV digitale folosesc de asmenea aceste tehnici proprii semnalelor digitale.

8.3. Considerații finale

În general, tehnologia multimedia lucrează cu un volum mare de date. Ca mediu ideal de stocare de la început în acest domeniu s-a impus CD-ROM, cu capacitatea sa de peste 600 Mo. În urrma evoluției tehnologiilor în domeniul stocării optice a informațiilor, au apărut standarde noi cum ar fi videodiscurile (DVD) cu capacități de 4,7 Go sau 9,4 Go, dar standardele încă nu sunt unitare şi acceptate de toți producătorii.

De asemenea, s-a pus problema transmisiilor digitale pentru utilizatorii obişnuiți. Acestea se reaizează în mod curent cu ajutorul echipamentelor numite *modem*-uri (Modulation-DEModulation), care folosesc din plin tehnica digitală pentru transmiterea datelor.

Datorită faptului că rețelele telefonice curente (numite şi rețele comutate) limitează viteza de transfer a datelor la valori care nu fac posibile transmisii multimedia de calitate, atenția s-a îndreptat spre echipamentele cu fibre optice, sateliți sau rețele locale rapide, care permit un flux crescut de date. Cei care dispun o legătură directă la Internet se pot bucura de existența unor posturi de radio digitale care transmit în rețeaua Internet. Se speră că viitorul va aparține aşa-numitelor autostrăzi multimedia, pe care vor fi vehiculate filme sau muzică la cerere.

9. Sistemul de operare

Până acum am discutat numai despre implementarea fizică a componentelor unui calculator. Desigur, buna funcționare a acestora este indispensabilă. Programele nu pot rula pe un calculator ale cărui componente nu funcționează corect. În acest sens, prima problemă care poate apărea este posibilitatea defectării unor circuite, caz în care acestea trebuiesc reparate sau (cel mai adesea) înlocuite. Totuşi, nu este suficient ca toate componentele să fie în stare fizică bună pentru ca sistemul de calcul în ansamblul său să funcționeze corespunzător. Trebuie ținut cont şi de faptul că fiecare dispozitiv are propriile specificații, propriile sarcini pe care le poate îndeplini, propriul mod de comunicare cu alte dispozitive ş.a.m.d. Deoarece calculatorul este format dintr-un număr mare de circuite, în general complexe şi foarte diferite între ele, este necesar să existe un control unic asupra tuturor acestora, pentru a le face să conlucreze în modul dorit de utilizator.

Din punct de vedere hardware, toate componentele calculatorului sunt controlate de către procesor. La rândul său, procesorul realizează acțiunile specificate prin programele pe care le execută. Dar, tocmai datorită structurii extrem de complicate a unui calculator, sarcina gestionării tuturor componentelor sale nu poate fi lăsată în seama programelor de aplicații. Pe de o parte, programatorul ar trebui să se concentreze mai mult asupra acestei gestiuni şi mai puțin asupra problemei propriu-zise pe care îşi propune s-o rezolve. Pe de altă parte, programele ar deveni astfel mult mai voluminoase şi, implicit, mai expuse la apariția erorilor.

Ca urmare, s-a ajuns la introducerea unui program intermediar între nivelul hardware şi programele de aplicații, care să asigure buna funcționare a sistemului de calcul. Acest program, numit *sistem de operare*, deține controlul asupra resurselor calculatorului şi intervine atunci când apar situații nedorite sau neprevăzute. Ca o primă consecință, deşi reprezintă o componentă software, sistemul de operare este strâns legat de hardware, deoarece se ocupă în principal de gestiunea acestuia.

Putem privi din mai multe unghiuri funcțiile pe care trebuie să le îndeplinească sistemul de operare. Din punct de vedere al utilizatorului, rolul său este exclusiv de a asigura rularea în bune condiții a programelor de aplicații. Se desprinde de aici ideea că sistemul de operare nu este un scop în sine, ci un mijloc pentru atingerea altor scopuri.

Pentru programatorul de aplicații, sistemul de operare este în principal un furnizor de servicii la care poate apela pentru rezolvarea problemelor întâlnite. Practic, sistemul de operare pune la dispoziția aplicațiilor un set de funcții predefinite, care fie sunt dificil de scris (şi de aceea nu este eficient să fie implementate de fiecare program în parte), fie pur şi simplu nu pot fi lăsate în seama aplicațiilor din motive de siguranță în funcționare a sistemului în ansamblul său. Aplicațiile pot folosi aceste servicii, conformându-se regulilor impuse de sistemul de operare.

Pentru a îndeplini aceste cerințe, proiectantul unui sistem de operare trebuie să aibă în vedere următoarele obiective:

- să asigure buna funcționare a componentelor hardware, precum şi comunicarea şi cooperarea între acestea

- să prevină interferențele nedorite între diferitele programe de aplicații, inclusiv să împiedice, în măsura posibilului, propagarea efectelor erorilor unui program asupra celorlalte programe

9.2. Clasificarea sistemelor de operare

Putem clasifica sistemele de operare după mai multe criterii. Primul dintre acestea reiese din discuția anterioară; după numărul de programe care pot rula simultan, sistemele de operare pot fi:

- *single-tasking* - permit rularea uni singur program la un moment dat; singurul sistem din această clasă care mai este folosit astăzi (dar din ce în ce mai puțin) este DOS
- *multitasking* - Unix, Windows 9x/NT, OS/2 etc.

O altă clasificare se referă la numărul de utilizatori care pot lucra simultan pe un calculator:

- sisteme monoutilizator (*single-user*)
- sisteme multiutilizator (*multiuser*)

În mod evident, un sistem multiuser este şi multitasking. Cele mai cunoscute sisteme de operare multiuser sunt cele din familia Unix, în timp ce sistemele Windows nu au această facilitate. Trebuie reținut că a da posibilitatea mai multor utilizatori să lucreze simultan pe acelaşi calculator nu este atât o problema de hardware, cât mai ales una specifică sistemului de operare.

9.3. Nucleul sistemului de operare

Datorită multitudinii şi diversității sarcinilor pe care le are de îndeplinit, sistemul de operare nu poate fi conceput sub forma unui program unitar. Practic, sistemul de operare constă dintr-o mulțime de secvențe de program, fiecare îndeplinind o anumită sarcină.

Un argument în favoarea unei asemenea abordări, în afara considerentelor de fiabilitate şi uşurință în dezvoltare, îl constituie evoluția continuă a tehnologiilor utilizate, în special în ceea ce priveşte dispozitivele periferice. Dacă la un moment dat se pune problema înlocuirii într-un calculator a unui asemenea periferic (de exemplu mouse) cu unul mai nou, va trebui schimbată secvența de program care se ocupă de gestionarea sa. În cazul în care sistemul de operare ar fi un program unic, acesta ar trebui înlocuit în intregime, ceea ce este inacceptabil în practică. Asupra acestui aspect vom reveni ulterior.

Pe de altă parte, există o serie de operațiuni fundamentale, care trebuie realizate întotdeauna în acelaşi mod, independent de particularitățile hardware-ului. Părțile de program care îndeplinesc aceste sarcini fundamentale formează *nucleul* sistemului de operare, care dirijează şi controlează funcționarea sistemului de calcul în ansamblul său. În continuare, noțiunile de sistem de operare şi de nucleu al sistemului de operare se vor confunda în mare măsură, deoarece celelalte componente ale sistemului de operare sunt utilizate de către nucleu pentru a-şi îndeplini sarcinile.

Nu există întotdeauna o delimitare clară între nucleu şi celelalte componente. Concepțiile diverşilor producători de sisteme de operare diferă în ceea ce priveşte locul unora dintre funcții - în nucleu sau în afara sa. Totuşi, practic toate sistemele de operare existente includ în nucleu următoarele componente:

- gestiunea proceselor
- gestiunea memoriei
- sistemele de fişiere

Majoritatea activităților pe care le desfăşoară sistemul de operare nu pot fi realizate exclusiv prin software. Este necesar un sprijin, uneori substanțial, din partea

componentelor hardware şi în special din partea procesorului. Natura exactă a acestui sprijin va fi discutată în continuare.

Principala facilitate oferită de către procesor o constituie sistemul de întreruperi, care a fost deja prezentat. În general, programele aflate în execuție rulează în majoritatea timpului fără a ține cont de existența sistemului de operare; totuşi, acesta din urmă trebuie să poată interveni în anumite situații bine definite, cum ar fi:

- o cerere de întrerupere venită din partea unui dispozitiv periferic, care poate să nu aibă legatură cu programul aflat în execuție, dar care trebuie tratată imediat (altfel datele se pot pierde)
- o operație executată de procesor care s-a terminat anormal (de exemplu o operație de împărțire la 0), ceea ce indică încercarea unui program de a efectua o acțiune nepermisă
- o cerere explicită adresată de programul de aplicație, privind efectuarea unui anumit serviciu de către sistemul de operare, serviciu pe care aplicația nu-l poate efectua singură

Sistemul de operare va lăsa deci orice program să se execute fără interferențe până la apariția uneia din situațiile descrise mai sus, dar în acest moment trebuie să preia imediat controlul. Soluția este, aşa cum am precizat deja, de natură hardware şi este reprezentată de sistemul de întreruperi. Concret, acesta oferă tocmai posibilitatea întreruperii execuției programului curent în anumite condiții. Fiecăreia din situațiile prezentate mai sus îi corespunde unul tipurile de întrerupere cunoscute:

- întreruperi hardware (externe)
- excepții (întreruperi hardware interne)
- întreruperi software

Dacă întreruperile hardware externe, care se ocupă de comunicarea cu dispozitivele periferice, au fost descrise pe larg, utilitatea excepțiilor rămâne să fie explicată. Pentru aceasta ne vom întoarce la mecanismele de gestiune a memoriei ale microprocesoarelor pe 32 biți, prezentate anterior. Reamintim că, indiferent dacă se utilizează segmentarea sau paginarea, pentru fiecare acces la memorie al unui program, procesorul realizează o serie de verificări, cu scopul de a determina dacă accesul este corect sau nu. În cazul în care este detectată o eroare (lipsa drepturilor de acces la segment, depăşirea dimensiunii segmentului, acces la o pagină virtuală inexistentă etc.), am arătat că încercarea de acces la memorie este oprită. În realitate, procesorul nu are prea multe posibilități de a opri execuția unui program. Ceea ce se întâmplă în practică este că unitatea de management a memoriei (MMU), care a detectat eroarea, generează o excepție, iar rutina de tratare care se apelează prin mecanismul de întreruperi va trebui să rezolve problema. Există mai multe moduri în care rutina de tratare poate restabili situația; în majoritatea cazurilor însă, un program care a realizat un acces ilegal la memorie este terminat forțat, deoarece acest tip de eroare este considerat foarte grav. Subliniem că întreruperea generată în acest caz este într-adevăr de tip excepție, deoarece MMU este o parte componentă a procesorului.

Excepțiile sunt larg utilizate de către sistemul de operare, deoarece multe erori sunt detectate direct de către procesor. Evident, rutinele care tratează situațiile generatoare de întreruperi fac parte din sistemul de operare, care poate astfel rezolva problemele apărute.

9.3.1. Apeluri sistem

Una din sursele întreruperilor, prezentate mai sus, o constituie solicitările formulate în mod explicit de programele de aplicații către sistemul de operare, pentru efectuarea anumitor servicii. De ce este însă necesar ca aceste servicii să fie

implementate de către sistemul de operare şi nu pot fi lăsate în seama programelor? În primul rând, unele operaţii uzuale (afişarea, căutarea pe disc etc.) se desfăşoară întotdeauna în acelaşi mod; deci, în loc de a scrie practic aceeaşi rutină în fiecare program, este mai economic de a o scrie o singură dată ca parte a sistemului de operare, astfel ca toate aplicaţiile să o poată utiliza. De altfel, apelul către un asemenea serviciu oferit de sistem nu se deosebeşte prea mult de apelul către o procedură sau funcţie din acelaşi program.

Pe de altă parte, o serie de acţiuni, în special accesele la dispozitivele periferice, prezintă riscuri considerabile pentru întregul sistem de calcul în cazul în care nu sunt realizate corect. Nu este deci convenabil de a permite programelor de aplicaţii să realizeze singure acţiunile din această categorie; se preferă ca activităţile de acest tip să fie îndeplinite numai prin intermediul unor rutine incluse în sistemul de operare. Pentru a pune în practică o asemenea abordare, trebuie să se poată interzice pur şi simplu realizarea anumitor operaţii de către programele de aplicaţii. Din nou este necesar un suport hardware. Practic toate procesoarele existente astăzi pot funcţiona în două moduri distincte:

- modul utilizator (*user mode*), în care există anumite restricţii pentru procesor, în principal nu se pot executa instrucţiunile de acces la periferice (încercarea de a executa o asemenea instrucţiune duce la generarea unei excepţii)
- modul supervizor sau nucleu (*kernel mode*), în care procesorul nu are nici o limitare

(Facem observaţia că, în cazul microprocesoarelor Intel, acestă împărţire este valabilă doar când procesorul se află în modul protejat. Modul real, destinat păstrării compatibilităţii cu aplicaţiile mai vechi, nu beneficiază de facilităţile hardware necesare pentru discuţia de faţă.)

În mod uzual, programele de aplicaţii se execută în mod utilizator, iar sistemul de operare rulează în mod nucleu. Se asigură astfel controlul sistemului de operare asupra operaţiilor critice. Deşi aplicaţiile pierd din performanţă prin limitările impuse de modul utilizator, creşterea stabilităţii şi siguranţei în funcţionare justifică din plin această abordare. În acest moment putem studia ce se întâmplă atunci când un program cere sistemului de operare furnizarea unui anumit serviciu. O asemenea cerere poartă numele de *apel sistem* (system call) şi constă din următorii paşi:

- programul, care rulează în modul utilizator al procesorului, depune parametrii apelului sistem pe care îl solicită într-o anumită zonă de memorie; practic, mecanismul este similar apelurilor de proceduri
- se generează o întrerupere software, care trece procesorul în modul nucleu
- se identifică serviciul cerut şi se apelează rutina de tratare corespunzătoare
- rutina respectivă preia parametrii apelului din zona în care au fost depuşi, îi verifica şi, dacă nu sunt erori, realizează acţiunea cerută; în caz contrar, apelul eşuează
- la terminarea rutinei, rezultatele obţinute sunt la rândul lor depuse într-o zonă de memorie cunoscută şi accesibilă programului de aplicaţie
- procesorul revine în modul utilizator şi se reia execuţia programului din punctul în care a fost întrerupt (utilizând informaţiile memorate în acest scop la apariţia întreruperii); programul poate prelua rezultatele apelului din zona în care au fost depuse

Se poate observa că execuţia unui apel sistem este mare consumatoare de timp. Din fericire, puterea de calcul a procesoarelor moderne este suficient de mare încât să reducă în limite acceptabile pierderea de performanţă datorată apelurilor sistem, iar

creşterea fiabilităţii sistemului de calcul în ansamblul său reprezintă un câştig mult mai important.

Anexa A. Reprezentarea datelor în sistemele de calcul

A.1. Reprezentări numerice

Am văzut anterior că numerele sunt reprezentate în calculator sub forma unor şiruri de biţi (corespunzători unor cifre în baza 2). În cele ce urmează vom prezenta în detaliu modurile de reprezentare a informaţiei folosite în sistemele de calcul.

A.1.2. Scrierea poziţională

Scrierea poziţională a reprezentat la apariţia sa un mare pas înainte în matematică. Ca o exemplificare a acestei afirmaţii, putem considera adunarea a două numere naturale oarecare. Dacă numerele sunt reprezentate în scriere romană (care nu este poziţională), se observă imediat că operaţia este foarte dificil de realizat. În schimb, adunarea aceloraşi numere reprezentate în baza 10 (care este o scriere poziţională) este banală. Acest exemplu simplu arată marele avantaj al scrierii poziţionale, şi anume că permite descrierea algoritmică a operaţiilor aritmetice, ceea ce o face indispensabilă pentru sistemele de calcul.

Un concept fundamental în scrierea poziţională îl constituie baza de numeraţie. Spunem că lucrăm în baza de numeraţie *d*, care este un număr natural supraunitar, dacă avem la dispoziţie *d* simboluri (cifre), având asociate respectiv valorile 0, 1, ..., *d*-1. Un număr este reprezentat într-o bază oarecare *d* ca un şir de cifre, fiecare poziţie *i* din şir având ataşat un factor implicit egal cu d^i. Concret, un număr natural *N* va avea ca reprezentare în baza *d* şirul de cifre $a_{n-1}a_{n-2}...a_1a_0$, $a_i \in \{0,1,...,d-1\}$, $\forall i = \overline{0,n-1}$, cu proprietatea că $N = \sum_{i=0}^{n-1} a_i \cdot d^i$. Se demonstrează că fiecare număr are o reprezentare unică pentru o bază de numeraţie dată.

Să luăm ca exemplu numărul 309. (De fapt, corect ar fi fost să spunem "numărul care are reprezentarea 309 în baza 10", deoarece şi atunci când scriem un număr, de fapt folosim o reprezentare a sa. Cum însă un număr este o abstracţiune, iar reprezentarea în baza 10 este cea folosită dintotdeauna de oameni pentru a desemna numerele, vom folosi în continuare aceeaşi convenţie). Reprezentarea sa în baza 10 este evidentă, deoarece $309 = 3 \cdot 10^2 + 0 \cdot 10^1 + 9 \cdot 10^0$. Să considerăm acum reprezentarea aceluiaşi număr în baza 2. Deoarece $309 = 1 \cdot 2^8 + 0 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$, reprezentarea sa în baza 2 este 100110101. Pentru a specifica baza de numeraţie în care este scrisă o reprezentare, se foloseşte o notaţie ca în exemplul de mai jos:

$309_{(10)} = 100110101_{(2)}$

Deoarece baza 2 este folosită de calculatoare, iar baza 10 este preferată de oameni, conversia (în ambele sensuri) între aceste două baze este adesea necesară. O reprezentare în baza 2 este uşor de convertit în baza 10, pur şi simplu prin aplicarea formulei $N_{(10)} = \sum_{i=0}^{n-1} a_i \cdot 2^i$, unde a_i sunt biţii care formează reprezentarea în baza 2. De exemplu, $1101001_{(2)} = 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 105_{(10)}$. Mai dificilă este conversia în sens invers, care se realizează după următorul algoritm:

Se împarte numărul la 2 şi se reţine restul. Câtul se împarte la rândul său la 2, iar procesul de repetă până când se obţine câtul 0. Reprezentarea în baza 2 este şirul resturilor obţinute, luate în ordine inversă.

Pentru exemplificare considerăm din nou numărul 309. Aplicând algoritmul de mai sus, avem (pe coloana din dreapta se găseşte şirul resturilor):

309	1
154	0
77	1
38	0
19	1
9	1
4	0
2	0
1	1
0	

Dacă se inversează şirul resturilor se obţine 100110101, adică exact reprezentarea calculată mai sus.

Să considerăm acum cazul concret al utilizării scrierii poziţionale în sistemele de calcul. Reamintim că într-un calculator operanzii au dimensiuni standardizate, mai precis octeţi sau multipli de octet. O consecinţă imediată este finitudinea reprezentării; cu alte cuvinte, nu putem reprezenta în calculator numere oricât de mari, deoarece numărul de biţi disponibil pentru reprezentări este finit. În aceste condiţii, este important să determinăm domeniul reprezentabil cu ajutorul operanzilor disponibili, altfel spus, intervalul în care se înscriu numerele cu care putem lucra. În mod evident, numărul minim reprezentabil este întotdeauna 0, indiferent de dimensiunea operanzilor. De asemenea, domeniul numerelor reprezentabile nu prezintă "goluri", ci formează un interval: fiind date două numere naturale x şi y care pot fi reprezentate, orice număr natural aflat între x şi y va putea fi la rândul său reprezentat. Mai rămâne deci să determinăm valoarea maximă a domeniului, valoarea care depinde de numărul de biţi alocat unei reprezentări.

Pentru operanzi de 8 biţi, valoarea maximă care poate fi reprezentată este $1\cdot2^7 + 1\cdot2^6 + 1\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = 255$. Mai general, dacă operanzii au n biţi, valoarea maximă care poate fi reprezentată este $1\cdot2^{n-1} + 1\cdot2^{n-2} + ... + 1\cdot2^1 + 1\cdot2^0 = 2^n-1$. Domeniul reprezentabil pe n biţi, folosind scrierea poziţională, este deci $0 \div 2^n-1$. Revenind la exemplul de mai sus, observămcă numărul 309 nu poate fi reprezentat pe 8 biţi, în timp ce pe 16 biţi are reprezentarea 0000000100110101 (valorile de 0 de la stânga fiind, bineînţeles, nesemnificative).

A.1.3. Reprezentări cu semn

Scrierea poziţională nu rezolvă toate problemele legate de reprezentarea numerelor în calculator. De exemplu, dacă dorim să lucrăm cu numere întregi, deci cu semn, scrierea poziţională singură nu mai este suficientă. Motivul este foarte simplu: în acest caz, pe lângă cifrele bazei de numeraţie mai este nevoie şi de un simbol suplimentar (semnul); deoarece biţii nu au decât două valori posibile, asociate cifrelor bazei 2, nu avem la dispoziţie nici un alt simbol.

Este deci necesar să se definească o nouă reprezentare care, folosind în continuare şiruri de biţi, să permită lucrul cu numere cu semn. În acelaşi timp, dorim să păstrăm avantajele scrierii poziţionale, motiv pentru care vom încerca să derivăm noua reprezentare din scrierea poziţională.

În primul rând, deoarece nu avem la dispoziţie un simbol suplimentar, pentru reprezentarea semnului va fi folosit unul dintre biţi, care nu va mai avea semnificaţia obişnuită unei cifre din scrierea poziţională. Concret, bitul cel mai semnificativ (cel cu indicele n-1) va indica semnul numărului şi va fi numit bit de semn. Prin convenţie, valoarea 1 a bitului de semn indică un număr negativ, iar valoarea 0 un număr pozitiv.

Desigur, s-ar fi putut alege şi convenţia inversă; motivul pentru care se preferă forma aleasă are însă avantajul că, în cazul numerelor pozitive, reprezentarea este aceeaşi ca în cazul scrierii poziţionale (bitul de semn fiind în acest caz un 0 nesemnificativ).

Cea mai naturală idee pentru reprezentarea numerelor cu semn este ca bitul cel mai semnificativ să indice exclusiv semnul, iar ceilalţi biţi să reprezinte modulul numărului în scriere poziţională. Formal, reprezentarea unui număr întreg N este şirul de biţi $a_{n-1}a_{n-2}...a_1a_0$, cu proprietatea că $N = (-1)^{a_{n-1}} \sum_{i=0}^{n-2} a_i \cdot 2^i$.

Deşi intuitivă, varianta de mai sus, numită reprezentare *modul-semn*, are unele dezavantaje. Pe de o parte, reprezentarea modul-semn a unui număr este unică, dar cu excepţia numărului 0: se observă imediat că în acest caz, dacă biţii care formează modulul au toţi valoarea 0, în schimb bitul de semn poate fi atât 0, cât şi 1. Desigur, această redundanţă poate crea proleme, în special la compararea numerelor.

Mai important, adunarea numerelor reprezentate cu modul şi semn nu mai urmează algoritmul clasic, specific scrierii poziţionale. Dacă pentru numere cu acelaşi semn situaţia rămâne neschimbată, în cazul numerelor cu semne diferite apar probleme. Considerăm ca exemplu numerele 3 şi -5. Reprezentările modul-semn, cu n=8, ale acestor numere sunt respectiv 00000011 şi 10000101. Dacă adunăm reprezentările după algoritmul clasic obţinem:

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} +$$

Rezultatul corespunde valorii -8, ceea ce este evident incorect. Desigur, se poate găsi un algoritm mai complex, care să realizeze adunarea în mod corect, dar s-ar pierde din performanţă, claritate şi simplitate. Problema este cu atât mai serioasă cu cât adunarea stă la baza celoralte operaţii aritmetice (scădere, înmulţire, împărţire), deci algoritmul prin care este implementată va influenţa şi implementările acestor operaţii.

O variantă propusă pentru rezolvarea acestei deficienţe este următoarea:

- numerele pozitive se reprezintă în continuare la fel ca în scrierea poziţională, cu bitul de semn având valoarea 0

- pentru un număr negativ se porneşte de la reprezentarea modulului său, apoi fiecare bit este complementat (0 se înlocuieşte cu 1 şi reciproc), inclusiv bitul de semn

Se obţine astfel reprezentarea numită în *complement faţă de 1*. Nu vom insista asupra acestei reprezentări, care are încă unele puncte slabe. Vom menţiona doar că ea constituie un pas înainte, iar algoritmul de adunare în acest caz este apropiat de cel clasic.

În final ajungem la soluţia care este folosită în sistemele de calcul actuale, numită reprezentare în *complement faţă de 2*, care preia ideile valoroase de la variantele anterioare. Formal, reprezentarea în complement faţă de 2 a unui număr întreg N este şirul de biţi $a_{n-1}a_{n-2}...a_1a_0$, cu proprietatea că:

$$N = \begin{cases} \sum_{i=0}^{n-2} a_i \cdot 2^i & \text{pentru } a_{n-1} = 0 \ (N \geq 0) \\ -2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i & \text{pentru } a_{n-1} = 1 \ (N < 0) \end{cases}$$

Deşi este mai complicată decât predecesoarele sale, reprezentarea în complement faţă de 2 elimină dezavantajele acestora. Astfel, numărul 0 are o reprezentare unică (toţi biţii cu valoarea 0), iar adunarea se face după acelaşi algoritm

ca la scrierea poziţională, indiferent de semnul operanzilor. Ultima proprietate a fost îndeplinită prin faptul că, pentru orice număr *x*, dacă adunăm după algoritmul clasic reprezentările numerelor *x* şi -*x*, obţinem întotdeauna 0.

Pentru exemplificare, considerăm din nou numerele 3 şi -5 şi *n*=8. Reprezentarea în cod complementar faăţ de 2 a numărului 3 este tot 00000011 (fiind pozitiv, este la fel ca la scrierea poziţională), în schimb pentru numărul -5 obţinem 11111011. Adunăm cele două reprezentări după algoritmul clasic şi obţinem:

$$\begin{array}{r} 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ + \\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0 \end{array}$$

Rezultatul este reprezentarea numărului -2, deci suma a fost calculată corect.

Un aspect interesant, nu lipsit de importantă practică, este modul de obţinere al reprezentării numărului -*x* pornind de la reprezentarea numărului *x*. Algoritmul este următorul:

- se parcurge şirul de biţi de la dreapta la stânga
- cât timp biţii au valoarea 0, sunt lăsaţi nemodificaţi
- primul bit întâlnit cu valoarea 1 este de asemenea lăsat nemodificat
- după acest prim bit cu valoarea 1, toţi biţii care urmează (indiferent de valoarea lor) sunt inversaţi

Algoritmul funcţionează la fel, indiferent de semnul lui *x*. De exemplu, pentru numărul -5, folosit mai sus, pornim de la reprezentarea numărului 5, care este evident 00000101. Deoarece nu avem nici un bit cu valoarea 0 la extremitatea dreaptă, primul bit este singurul nemodificat; toţi ceilalţi biţi sunt inversaţi, deci se obţine 11111011. Este uşor de verificat faptul că, pornind de la reprezentarea lui -5, se obţine în acelaşi mod reprezentarea numărului 5.

Revenim la problema domeniului reprezentabil pe un număr finit de biţi. Reamintim că, în cazul scrierii poziţionale, intervalul numerelor reprezentabile pe *n* biţi este $0 \div 2^n-1$, constând deci din 2^n valori diferite consecutive. Dacă folosim reprezentarea în complement faţă de 2 (sau oricare alta), intervalul nu poate creşte în dimensiune, din simplul motiv că pe *n* biţi se pot reprezenta maximum 2^n valori diferite (sunt posibile 2^n configuraţii diferite ale şirului de biţi). Astfel, intervalul valorilor reprezentabile este deplasat, incluzând atât numere negative, cât şi numere pozitive. Dacă analizăm reprezentarea în complement faţă de 2, observăm că numărul cel mai mic reprezentabil pe *n* biţi corespunde şirului 100...00, având deci valoarea -2^{n-1}. În acelaşi timp, cel mai mare număr reprezentabil corespunde şirului de biţi 011...11, având valoarea $2^{n-1}-1$. Rezumând, dacă folosim operanzi pe *n* biţi şi reprezentarea în complement faţă de 2, putem lucra cu numere în intervalul $-2^{n-1} \div 2^{n-1}-1$. De exemplu, pentru *n*=8 (operanzi pe un octet), putem lucra cu numere cuprinse între -128 şi 127.

A.1.4. Reprezentări zecimale

În mod evident, majoritatea aplicaţiilor implică efectuarea de calcule cu numere reale. Pentru aceste aplicaţii, reprezentările discutate până acum, a căror aplicabilitate este restrânsă la numerele naturale sau cel mult întregi, nu mai sunt suficiente. În continuare vom aborda reprezentarea numerelor cu zecimale.

Începem prin a face o observaţie importantă. Datorită caracterului finit al oricărei reprezentări, nu vom putea lucra cu adevărat cu numere reale, ci doar cu numere raţionale (reamintim că numerele iraţionale necesită un număr infinit de zecimale, indiferent de baza de numeraţie folosită); mai mult, nu vom avea la dispoziţie nici măcar toate numerele raţionale dintr-un anumit interval, ci doar o parte

dintre acestea. Vom vedea de altfel că una dintre caracteristicile importante ale acestor reprezentări este distanța dintre două numere reprezentabile succesive (care în cazul numerelor întregi este, desigur, 1). Totuşi, în practică vom numi "reale" numerele reprezentate în acest mod.

Ideea de pornire este una naturală: din numărul total de biți disponibil pentru o reprezentare, unii vor fi folosiți pentru partea întreagă, iar restul pentru partea zecimală. Întâlnim astfel aceeaşi problemă ca şi la reprezentarea numerelor cu semn: avem nevoie de un simbol suplimentar (virgula), pe lângă cele pentru cifre, simbol de care însă nu dispunem. Rezolvarea este totuşi diferită, aşa cum vom vedea în continuare.

Întrucât toate reprezentările se bazează, în ultimă instanță, pe scrierea pozițională, reluăm problema conversiei între bazele 2 şi 10, de data aceasta pentru numere zecimale. Din nou, conversia din baza 2 în baza 10 este imediată. Fie de exemplu numărul $1010010{,}1011_{(2)}$. La fel ca în cazul numerelor naturale, avem:

$1010010{,}1011_{(2)} = 1\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} + 1\cdot2^{-4} = 82{,}6875_{(10)}$

La conversia din baza 10 în baza 2, partea întreagă şi partea zecimală trebuie calculate separat. Algoritmul pentru calculul părții întregi a fost deja prezentat. Pentru partea zecimală se procedează astfel:

Se înmulțeşte partea zecimală cu 2 şi se reține cifra de la partea întreagă (nu poate fi decât o singură cifră semnificativă, cu valoarea 0 sau 1). Noua parte zecimală se înmulțeşte la rândul său cu 2, iar procesul se repetă până când partea zecimală devine 0. Şirul cifrelor obținute la partea întreagă din fiecare înmulțire formează rezultatul căutat.

De exemplu, fie numărul $309{,}3125_{(10)}$. Partea întreagă, aşa cum s-a văzut deja, se scrie în baza 2 sub forma 100110101. Aplicând algoritmul prezentat mai sus pentru partea zecimală, obținem (în stânga este partea întreagă, iar în dreapta partea zecimală):

0	3125
0	625
1	25
0	5
1	0

Şirul cifrelor obținute la partea întreagă este 0101, deci $0{,}3125_{(10)} = 0{,}0101_{(2)}$. Ca urmare, $309{,}3125_{(10)} = 100110101{,}0101_{(2)}$.

O problemă care apare în acest caz este posibilitatea ca unui număr finit de zecimale în baza 10 să-i corespundă un număr infinit de zecimale în baza 2 (invers nu este posibil). Într-un asemenea caz este evident imposibil să obținem o reprezentare exactă a numărului. Soluția este să calculăm atâtea zecimale în baza 2 câte încap în reprezentare, obținând astfel cea mai bună aproximare pe care o putem memora.

De exemplu, fie numărul $0{,}3_{(10)}$. Vom aplica din nou algoritmul de mai sus:

0	3
0	6
1	2
0	4
0	8
1	6
1	2
⋮	⋮

Este uşor de observat că partea zecimală nu va ajunge niciodată la valoarea 0, deci numărul de cifre în baza 2 necesar pentru reprezentare este infinit. Dacă ne propunem să reprezentăm numărul cu 4 biţi la partea zecimală, vom obţine aproximarea $0{,}0100_{(2)}$; dacă folosim 6 biţi, obţinem $0{,}010011_{(2)}$ etc.

Există două forme diferite de reprezentare a numerelor zecimale, care vor fi prezentate în continuare.

A.1.5. Reprezentări în virgulă fixă

O primă soluţie este ca separarea între partea întreagă şi partea zecimală a reprezentării să se facă întotdeauna la fel. Cu alte cuvinte, vom avea întotdeauna *n* biţi pentru partea întreagă şi *m* biţi pentru partea zecimală, valorile pentru *n* şi *m* rămânând în permanenţă constante. Desigur, numărul de biţi alocat unei reprezentări este *n*+*m*.

Această abordare poartă denumirea de reprezentare în *virgulă fixă*, deoarece poziţia virgulei în cadrul numărului nu se modifică. Principala consecinţă este că virgula nu mai trebuie reprezentată în mod explicit, deoarece poziţia sa, fiind mereu aceeaşi, este oricum cunoscută. Ca urmare, dispare necesitatea unui simbol suplimentar pentru virgulă, deci reprezentarea pe şiruri de biţi este posibilă.

Este uşor de văzut că reprezentarea în virgulă fixă poate fi considerată o generalizare a reprezentărilor pentru numere întregi, discutate anterior. În practică se porneşte de la reprezentarea în complement faţă de 2, care este cea mai completă, şi se decide ca un număr de biţi să fie rezervaţi părţii întregi. Astfel, reprezentarea unui număr N pe *n*+*m* biţi este şirul de biţi $a_{n-1}a_{n-2}...a_1a_0a_{-1}a_{-2}...a_{-m}$, cu proprietatea:

$$N = \begin{cases} \sum_{i=-m}^{n-2} a_i \cdot 2^i & \text{pentru } a_{n-1} = 0 \; (N \geq 0) \\ -2^{n-1} + \sum_{i=-m}^{n-2} a_i \cdot 2^i & \text{pentru } a_{n-1} = 1 \; (N < 0) \end{cases}$$

Pentru a determina intervalul numerelor reprezentabile în virgulă fixă pe *n*+*m* biţi, ne folosim de faptul că toate proprietăţile reprezentării în complement faţă de 2 rămân valabile. Reluând raţionamentul de mai sus, cel mai mic număr reprezentabil corespunde şirului 100...00, având tot valoarea -2^{n-1}. Pe de altă parte, cel mai mare număr reprezentabil corespunde şirului de biţi 011...11, care acum are valoarea $2^{n-1}-2^{-m}$. Intervalul numerelor reprezentabile în virgulă fixă pe *n*+*m* biţi este deci [-2^{n-1} , $2^{n-1}-2^{-m}$]. Reamintim faptul că nu este vorba cu adevărat de un interval în sensul matematic al termenului, ci doar de o parte dintre numerele raţionale din acel interval.

Un alt aspect important, aşa cum s-a văzut mai sus, este distanţa dintre două numere reprezentabile succesive. În acest caz, trecerea de la un număr reprezentabil la următorul (cel imediat superior) se face întotdeauna prin adunarea reprezentării 00...001, care corespunde numărului 2^{-m} (cel mai mic număr strict pozitiv care poate fi reprezentat). În concluzie, distanţa (numită uneori şi pas) dintre două numere reprezentabile succesive este constantă şi are valoarea 2^{-m}.

A.1.6. Reprezentări în virgulă mobilă

Reprezentarea în virgulă fixă are dezavantajul lipsei de flexibilitate. Într-adevăr, odată fixate valorile pentru *n* şi *m*, s-au stabilit în mod definitiv atât ordinul de mărime al numerelor reprezentabile (dat de *n*), cât şi precizia reprezentării (dată de numărul de zecimale, adică *m*). Totuşi, pentru aceeaşi dimensiune totală a unui operand, putem prefera, de exemplu, ca în unele cazuri să lucrăm cu numere mai mici, dar cu o precizie mai bună; aceasta s-ar putea realiza scăzând *n* şi crescând *m*, astfel încât suma lor să rămână aceeaşi. Din păcate, reprezentarea în virgulă fixă nu permite asemenea adaptări, deoarece valorile *n* şi *m* nu pot fi modificate. Reprezentarea în virgulă mobilă vine să corecteze acest neajuns.

Ideea de pornire provine din calculul ştiinţific, unde se foloseşte notaţia cu exponent. Mai concret, un număr ca 243,59 poate fi scris sub forma $2{,}4359 \cdot 10^2$. Observăm că în scrierea ştiinţifică intervin trei elemente:

- mantisa (în cazul de faţă 2,4359), care poate fi privită ca fiind "corpul" numărului
- baza de numeraţie (10)
- exponentul (2) la care este ridicată baza de numeraţie

Putem deci reprezenta numărul ca o tripletă (mantisă, bază, exponent). În practică însă, baza de numeraţie este mereu aceeaşi, astfel încât pentru reprezentarea numărului sunt suficiente mantisa şi exponentul.

Evident, pentru aceeaşi bază de numeraţie pot exista mai multe perechi mantisă-exponent care să reprezinte acelaşi număr. De exemplu, numărul 243,59 poate fi scris şi sub forma $24{,}359 \cdot 10^1$. De fapt, se foloseşte întodeauna perechea în care mantisa are exact o cifră semnificativă înainte de virgulă; se spune în acest caz că mantisa este normalizată. Avantajul acestei notaţii este că nu mai apar probleme privind numărul de biţi alocat părţii întregi şi respectiv părţii zecimale a numărului. Orice variaţie în ce priveşte ordinul de mărime al numărului sau precizia sa se rezolvă exclusiv prin ajustarea exponentului.

Structura generală a unei reprezentări în virgulă mobilă este următoarea (figura A.1):

- Bitul cel mai semnificativ indică semnul numărului, după convenţia deja cunoscută (1 - negativ, 0 - pozitiv). Următoarele câmpuri determină modulul numărului.
- Următorii biţi reţin exponentul într-o formă modificată, numită caracteristică.
- Restul de biţi formează mantisa.

Fig. A.1.

Pentru simplitate, în continuare vom nota bitul de semn cu S, exponentul cu E, caracteristica cu C, iar mantisa cu M.

Deoarece numărul poate fi supraunitar sau subunitar, rezultă că exponentul poate fi pozitiv sau negativ. Ca atare, pentru exponentului trebuie folosită o reprezentare cu semn (separat de bitul de semn al numărului). Totuşi, reprezentarea în complement faţă de 2, care pare a fi prima alegere, are un dezavantaj: comparaţia între numere este relativ greoaie, deoarece se realizează în mod diferit în funcţie de semnele celor două numere. Deşi în general aceasta nu constituie o problemă, în

lucrul cu numere în virgulă mobilă, comparația între exponenți este o operație executată foarte des. Din acest motiv s-a apelat la o reprezentare mai rar folosită, numită reprezentare *în exces*. Concret, dacă avem *n* biți pentru exponent, atunci exponentul poate lua valori între $-2^{n-1} + 1$ şi 2^{n-1}. Apoi, din exponent se obține caracteristica prin formula $C = E + (2^{n-1} - 1)$, unde $2^{n-1} - 1$ este o constantă numită exces. Se observă imediat că întodeauna $C \geq 0$, deci pentru caracteristică se poate folosi scrierea poziţională, ceea ce permite implementarea uşoară a operației de comparare.

În privința mantisei, reamintim că aceasta trebuie să fie normalizată, adică să aibă exact o cifră semnificativă la partea întreagă. În baza 2, singura cifră semnificativă este 1, deci mantisa este de forma 1,...; deoarece partea întreagă a mantisei este întotdeauna la fel, ea nu mai trebuie memorată. Ca urmare, câmpul M va memora numai biții corespunzători părții zecimale a mantisei.

Există două implementări practice ale reprezentării în virgulă mobilă, ambele bazate pe structura prezentată mai sus, definite în standardul IEEE 754:

a) Reprezentarea în simplă precizie, care ocupă 32 biți (4 octeți), repartizați astfel:

- bitul de semn
- 8 biți pentru caracteristică
- 23 biți pentru mantisă

b) Reprezentarea în dublă precizie, care ocupă 64 biți (8 octeți):

- bitul de semn
- 12 biți pentru caracteristică
- 51 biți pentru mantisă

Cele două variante sunt similare, diferind doar prin numărul de biți alocați diferitelor componente.

Pentru a înțelege mai bine toate aceste elemente, vom considera numărul $2157{,}375_{(10)}$ şi vom vedea care sunt paşii care trebuie parcurşi pentru a obține reprezentarea sa în virgulă mobilă, simplă precizie.

- În primul rând, scriem numărul poziţional în baza 2. Aplicând algoritmii descrişi anterior pentru conversia în baza 2 a părții întregi şi respectiv a părții zecimale, obținem:

$2157{,}375_{(10)} = 100001101101{,}011_{(2)}$

- Urmează scrierea sub formă de mantisă şi exponent, cu mantisa normalizată:

$100001101101{,}011 = 1{,}00001101101011 \cdot 2^{11}$

- Câmpul M rezultă imediat, fiind format din partea zecimală a mantisei. Deoarece mantisa are numai 14 biți semnificativi, iar câmpul M are 23 biți, ultimii 9 biți din M primesc valoarea 0 (sunt nesemnificativi):

M = 00001101101011000000000

- Caracteristica se calculează din exponent:

$E = 11 \Rightarrow C = E + (2^7 - 1) = 11 + 127 = 138$

Scris în baza 2, pe 8 biți: C = 10001010

- Bitul de semn are valoarea S = 0, deoarece numărul este pozitiv.

Concatenând şirurile de biți corespunzătoare celor 3 câmpuri, obținem reprezentarea în simplă precizie. Deoarece numărul de biți este mare şi deci dificil de controlat, în practică se preferă scrierea în baza 16, mai concisă:

$01000101000001101101011000000000_{(2)} = 4506D600_{(16)}$

Câteva caracteristici ale reprezentărilor în virgulă mobilă:

a) simplă precizie

- cel mai mic număr nenul reprezentabil (în modul): $1{,}175494351 \cdot 10^{-38}$

- cel mai mare număr reprezentabil (în modul): $3{,}402823466 \cdot 10^{38}$

b) dublă precizie

- cel mai mic număr nenul reprezentabil (în modul): $2{,}2250738585072014 \cdot 10^{-308}$
- cel mai mare număr reprezentabil (în modul): $1{,}7976931348623158 \cdot 10^{308}$

O proprietate a reprezentărilor în virgulă mobilă de care trebuie ținut cont este faptul că distanța dintre două numere reprezentabile succesive nu mai este constantă, ci depinde de valoarea exponentului. Dat fiind un număr oarecare, numărul reprezentabil imediat superior se obține adunând la mantisă valoarea minimă posibilă, adică 00...001; în simplă precizie aceasta corespunde valorii 2^{-23}, iar în dublă precizie valorii 2^{-51}. Dacă vom considera, de exemplu, reprezentarea în simplă precizie, observăm că diferența între cele două numere succesive este $E \cdot 2^{-23}$, în timp ce pentru dublă precizie este $E \cdot 2^{-51}$. Rezultă de aici nu numai că nu putem reprezenta toate numerele iraționale din domeniu, dar şi că, pentru valori mari ale exponentului, există chiar şi numere întregi care nu pot fi reprezentate exact.

adunare, înmulțire

A.2. Reprezentări alfanumerice

Informațiile prelucrate de calculator nu se rezumă la numere. Textele sunt de asemenea larg folosite, poate chiar într-o măsură mai mare, dacă privim din punctul de vedere al unui utilizator obişnuit. Este deci natural să se definească un mod de reprezentare pentru caractere.

O reprezentare alfanumerică asociază fiecărui caracter o valoare unică. Aşa cum am văzut, calculatorul lucrează cu şiruri de biți, organizate în octeți; dat fiind că literele alfabetului latin (mari şi mici), împreună cu cifrele, semnele de punctuație şi cele matematice, totalizează mai puțin de 100 de simboluri diferite, este suficient să se aloce fiecărui caracter o valoare pe un octet. Pentru simplitate, aceste valori ale octeților sunt desemnate tot sub formă numerică, aceasta fiind mai accesibilă şi mai uşor de reținut. De exemplu, nu vom spune că unui caracter îi este asociat octetul 01101000, ci numărul (codul) 104.

O primă încercare de standardizare a reprezentărilor alfanumerice a dus la elaborarea codului EBCDIC (Extended Binary Coded Decimal Interchange Code). Deoarece însă nu au fost urmate nişte principii clare în proiectare, codul EBCDIC nu s-a impus. În principal nu s-a ținut cont de faptul că, dacă lucrul cu caractere nu implică operații aritmetice sau logice, există totuşi unele prelucrări care se aplică asupra acestora, cum ar fi comparația, conversii între litere mari şi mici, conversii între şiruri de caractere şi tipuri numerice etc.

Mult mai bine definit este codul ASCII (American Standard Code for Information Interchange), care s-a impus ca standard, fiind folosit până astăzi. În proiectarea sa au fost avute în vedere câteva aspecte care uşurează prelucrarea caracterelor:

- Literele mici au coduri consecutive, în conformitate cu ordinea alfabetică. La fel s-a procedat şi în cazul literelor mari. Astfel sunt facilitate atât implementarea comparației între două caractere, care se reduce la comparația între două numere naturale, cât şi operațiile de testare (de exemplu, dacă un caracter este literă mare sau nu).

- Cifrele au de asemenea coduri consecutive. Mai mult, ultimii 4 biți ai codului unei cifre dau exact valoarea cifrei respective, ceea ce uşurează conversia între şiruri de caractere şi tipuri numerice.

Inițial, codul ASCII prevedea 128 caractere. Această primă versiune, care conține informațiile fundamentale necesare unei reprezentări alfanumerice, este prezentată în tabelul A.1. Se observă că primele 32 coduri sunt alocate unor caractere speciale. Acestea nu apar explicit în text, dar influențează modul de prezentare (dispunere în pagină) al acestuia, fiind folosite în special de către editoarele de texte. De exemplu, codul 9 corespunde caracterului "tab", care este folosit pentru alinierea unor părți din text.

Deoarece oricum se folosea un octet pentru memorarea unui caracter, codul ASCII a fost extins, ajungând la 256 caractere. Simbolurile nou introduse sunt în general cele folosite în limbile unor țări europene, dar care nu se regăsesc în alfabetul englez (de exemplu "á", "â", "ë" etc.).

Evident, nici după această extindere codul ASCII nu putea cuprinde simbolurile tuturor scrierilor folosite în lume. Pentru a rezolva problema a fost elaborat un standard nou, numit Unicode. Acesta foloseşte coduri de 2 octeți pentru caractere, ceea ce ridică numărul simbolurilor disponibile la 65536. Bineînțeles, sunt incluse şi codurile ASCII. Deocamdată Unicode nu s-a impus pe scară largă, fiind folosit mai ales în aplicațiile care au versiuni într-un număr mare de limbi.

Cod	Caracter	Cod	Caracter	Cod	Caracter	Cod	Caracter
0	NUL	32	spațiu	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	□

Tabelul A.1.

Bibliografie

J. L. Henessy, D. A. Patterson, *Computer Architecture - A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

D. A. Patterson, J. L. Henessy, *Organizarea şi proiectarea calculatoarelor. Interfața hardware/software*, Ed. All, 2002.

A. Tanenbaum, *Organizarea structurată a calculatoarelor*, Ed. Agora, 1999.

A. Tanenbaum, *Modern Operating Systems*, Prentice Hall, 2001.

Cuprins