10 SISTEME MICROPROGRAMATE

10.1 Procesorul

Apariția conceptului de **procesor** este strâns legată de apariția conceptului de **microprogramare**. Conceptul de microprogramare, ca metodă de proiectare a structurilor de control numerice, este prezentat pentru prima dată în 1951 de Maurice Wilkes de la Universitatea din Cambridge. El a enunțat ideile principale care stau la baza microprogramării şi a propus primul model al unei structuri de control microprogramate. Modelul lui Wilkes este reprodus în figura 10.1, după lucrarea [Lupu, 1995].

Vom lăsa autorul să descrie funcționarea schemei concepute de el, pentru stilul clar şi concis folosit în exprimare, iar noi ne vom limita la precizarea sensului actual al unor termeni folosiți de autor, prin scrierea lui în paranteze.

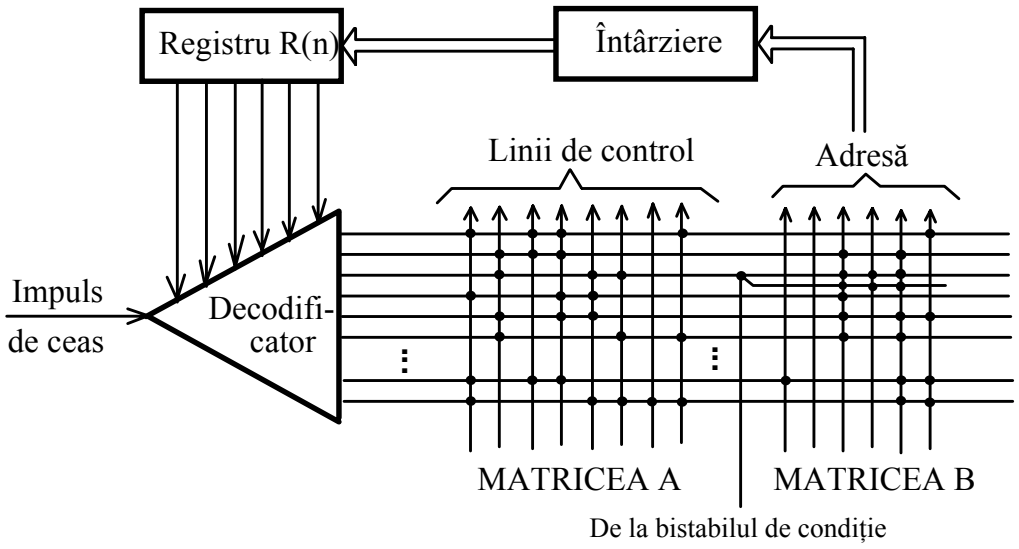

Fig. 10.1 *Structura de control microprogramată propusă de Wilkes*

Sigur că şi până la acea dată au fost proiectate şi realizate procesoare, dar controlul propriu-zis, adică "partea maşinii care generează impulsurile necesare validării porţilor asociate registrelor aritmetice sau de control" erau realizate de proiectant "într-o manieră ad-hoc, proiectând scheme-bloc până când ajunge la un aranjament care îi satisface cerinţele tehnice şi pare a fi acceptabil din punct de vedere economic." Meritul lui Wilkes este de a oferi o metodă sistematică şi clară în locul abordărilor euristice sau intuitive folosite până la el la proiectarea procesoarelor.

"Orice operaţie apelată de un ordin care corespunde **codului de ordin** (instrucţiunii) al maşinii presupune o **secvenţă de paşi** ce poate cuprinde transferuri între memorie şi registrele de control sau aritmetice şi transferuri între registre. Fiecare din aceşti paşi este realizat prin **pulsarea** (tranziţii logice) unor anumite semnale asociate cu registrele de control sau aritmetice. Vom numi aceşti paşi **microoperaţii**. O operaţie-maşină propriu-zisă (instrucţiune) este deci compusă dintr-o secvenţă sau un **microprogram** de microoperaţii. Figura (la noi 10.1) reprezintă o schemă cu ajutorul căreia se obţin microoperaţiile. Impulsul care generează o microoperaţie intră în arborele de decodificare şi este condus la una din ieşiri în funcţie de conţinutul registrului R. El trece prin matricea A şi generează impulsuri la unele din ieşirile matricei în funcţie de aranjamentul impedanţelor de cuplaj. Aceste impulsuri validează porţile asociate registrelor de control sau aritmetice şi deci realizează o microoperaţie. Impulsul generat în arborele de decodificare trece de asemenea şi prin matricea B generând impulsuri la unele din ieşirile acestei matrice. Aceste impulsuri sunt conduse printr-o scurtă linie de întârziere la registrul R, schimbând conţinutul acestuia. Ca rezultat, următorul impuls care intră în arbore va fi condus la alte ieşiri şi în consecinţă va realiza o altă microoperaţie. Se vede deci că fiecare rând din matricea A corespunde unei microoperaţii din secvenţa necesară realizării unei operaţii-maşină".

Wilkes a observat că instrucţiunile-maşină pot fi implementate ca secvenţe de mai multe operaţii elementare care să specifice transferul de informaţie între diferitele elemente ale unei unităţi centrale. Cu ajutorul unor porţi de control proiectantul maşinii poate comanda circulaţia informaţiei între aceste elemente.

Decodificatorul acceptă ieşirea registrului R, un registru pe n biţi, care constituie intrările de selecţie ale decodificatorului. În acest fel va fi validată o singură linie din cele 2^n linii de ieşire ale decodificatorului. Matricea de control A are 2^n linii orizontale şi un număr variabil de linii verticale(liniile de control), conectate în anumite puncte cu cele orizontale, conform secvenţelor de comandă dorite. Matricea A este o memorie ROM, care constituie memoria de microprograme. O linie orizontală poate fi concepută ca o microinstrucţiune, care, atunci când este selectată, va activa un set predeterminat de linii de control (fiecare validează la rândul ei anumite porţi de control din sistem).

Matricea B este o matrice de secvenţiere, comandată tot de ieşirea decodificatorului. Ieşirea acestei memorii stabileşte conţinutul registrului R pentru extragerea următoarei microinstrucţiuni (adresa următoarei microinstrucţiuni). Linia de întârziere are rolul de a asigura prelungirea duratei în care informaţia de control este stabilă, după schimbarea conţinutului registrului de adresă R.

Logica de decizie şi salt, existentă în orice algoritm de control, este asigurată de un bistabil de condiţie, a cărui valoare logică depinde de realizarea unui anumit test din sistem. În funcţie de rezultatul testului, bistabilul selectează una din cele două căi alternative în microprogram. Schema logică de principiu a circuitului care alege una din cele două variante posibile este dată în figura 10.2. Se vede uşor că valoarea ieşirii bistabilului de condiţie alege una din cele două linii, dacă linia de test condiţie este activată.

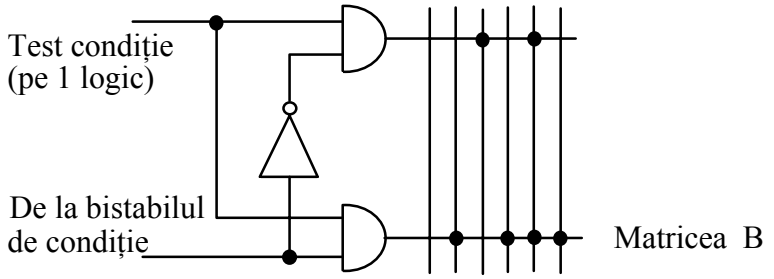

Fig.10.2 *Modul de selecţie a unei variante din două posibile*

Modelul propus de Wilkes foloseşte deci două matrice: una pentru specificarea microoperaţiilor executate într-un ciclu, iar cealaltă pentru determinarea adresei microinstrucţiunii următoare. De fapt este vorba de două memorii PROM, dar această separare este convenabilă din punct de vedere funcţional. În practică folosim de regulă o singură memorie, lucru realizabil datorită faptului că cele două memorii au aceeaşi adresă. Separarea ieşirilor memoriei în linii de control şi linii de adresă se face prin fixarea **câmpurilor** din **formatul unei microinstrucţiuni**.

[Lupu 1995] stabileşte esenţa metodei, cu ajutorul conceptelor de microoperaţie, microinstrucţiune şi microprogram, introduse de Wilkes: "**microprogramarea** înseamnă controlul unei structuri numerice prin intermediul unor cuvinte "citite" secvenţial, pas cu pas, dintr-o memorie. Prin citirea succesivă a acestor cuvinte, **microinstrucţiunile**, se generează semnalele de control, **microoperaţiile**, necesare funcţionării corecte a structurii respective. Proiectarea unei structuri microprogramate este astfel mult mai sistematică şi mai flexibilă decât cea a unei structuri convenţionale realizate prin logică cablată."

În principiu, o **maşină microprogramată** este o maşină în care o secvenţă coerentă de microinstrucţiuni este folosită pentru execuţia operaţiilor mari ce definesc funcţionarea maşinii. Ea este alcătuită din două blocuri principale: **secvenţiatorul de microprograme,** care generează adresa de microinstrucţiune şi **memoria de microprograme**. Dacă maşina este un **procesor**, atunci cele două blocuri sunt nişte automate finite numite **automat de procesare** (sau **unitate de tratare**, după [Stauffer 1989]), respectiv **automat de comandă** (sau **unitate de comandă**, după [Stauffer 1989]). Interconectarea celor două blocuri este prezentată în figura 10.3.

La un **ordin** (instrucţiune) aplicat din exterior, automatul de comandă generează o secvenţă de **comenzi** (microinstrucţiuni) spre automatul de procesare. Automatul de procesare prelucrează **datele de intrare** în funcţie de comenzile primite şi prin **indicatori** ai rezultatelor (flag-uri) influenţează evoluţia ulterioară a automatului de comandă. Rezultatele

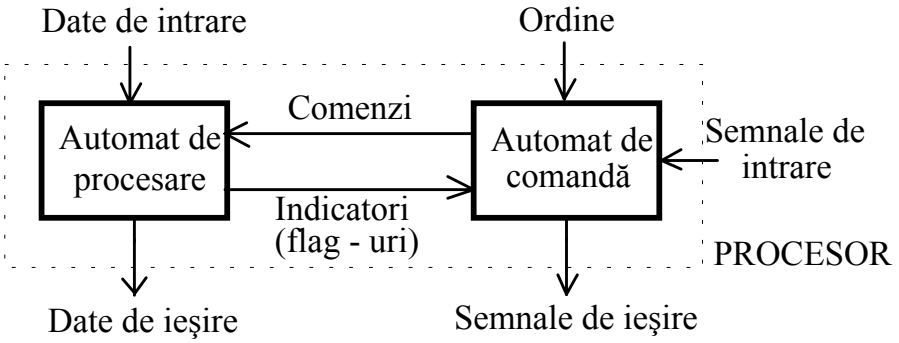

Fig. 10.3 *Structura procesorului elementar*

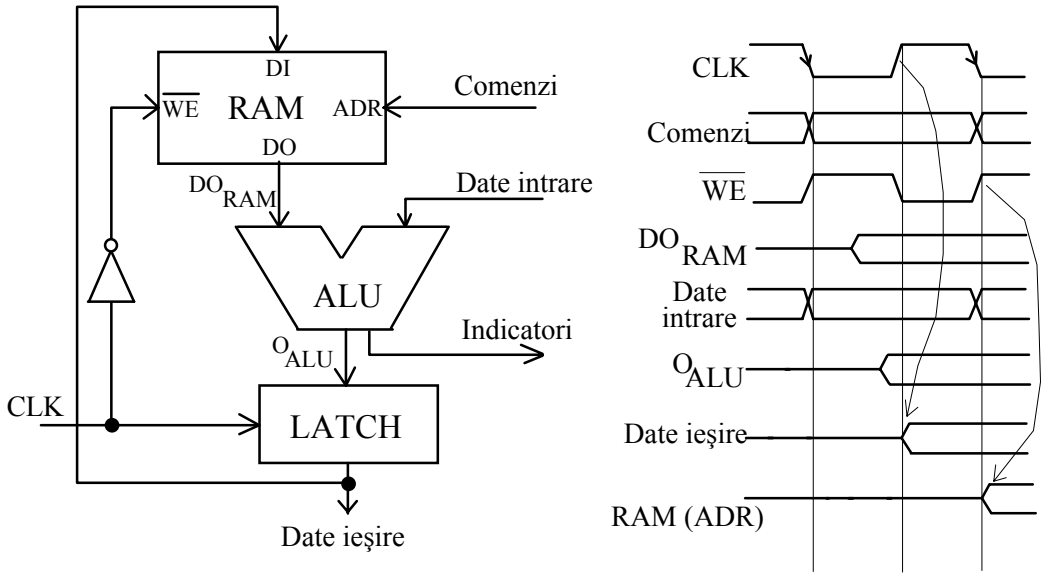

Fig. 10.4 *Structura automatului de procesare (RALU)*

prelucrării sunt livrate ca **date de ieşire**. **Semnalele de intrare** aplicate automatului de comandă sunt necesare pentru sincronizarea procesorului cu evenimente externe, iar **semnalele de ieşire** pot fi comenzi spre alte subsisteme externe sau semnale de sincronizare pentru alte dispozitive externe.

Automatul de procesare este de fapt o structură ce conţine o unitate aritmetico-logică (ALU) şi un set de registre, structură numită pe scurt **RALU**. Schema de principiu a acestui automat şi formele de undă care explică funcţionarea sunt date în figura 10.4. Stările automatului sunt date de conţinutul memoriei RAM, funcţiile de tranziţie ale automatului sunt calculate de ALU, iar LATCH-ul permite definirea structurii de tip master-slave pentru comutarea pe front a întregii structuri. În practică apar de cele mai multe ori structurări suplimentare ale automatului, de tip serie (multiplexoare pe intrările în ALU pentru selecţia operanzilor, precedate de structuri "circuit combinaţional şi latch" pentru memorarea unor operanzi intermediari), paralel (extinderea numărului de operanzi prin multiplexarea intrărilor în ALU, sau extinderea numărului ALU), sau mixt.

Automatul de comandă are o structură complexă şi din acest motiv se preferă o descriere funcţională. Scopul automatului de comandă este generarea unei secvenţe de comenzi spre alt circuit (de exemplu RALU), astfel încât să se realizeze o acţiune complexă prin serializarea unor acţiuni elementare. Evoluţia automatului este determinată de starea proprie şi de intrări şi poate fi modelată, pentru *n* perioade de ceas, cu un arbore cu *n* nivele, în care se parcurge un anumit drum, ca în figura 10.5.

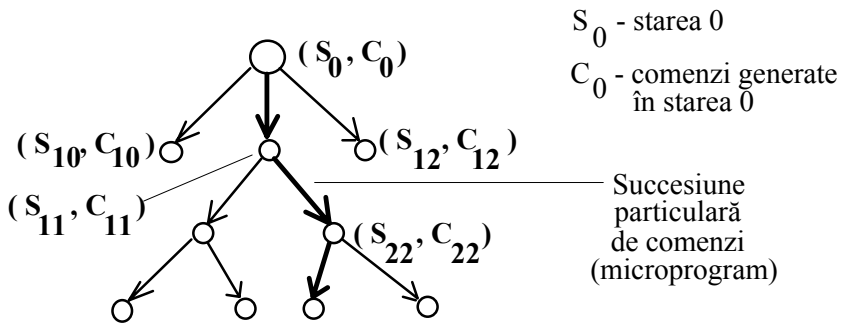

Fig. 10.5 *Parcurgerea unei secvenţe de microprogram*

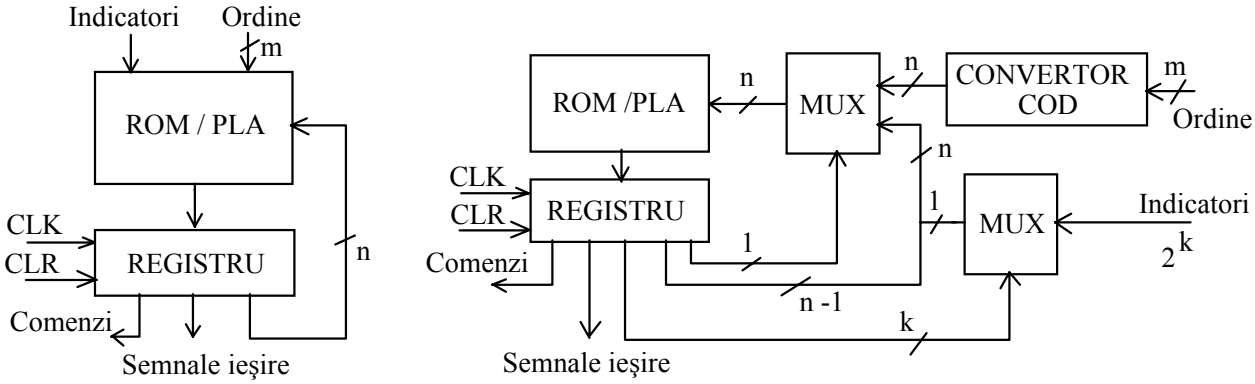

Fig. 10.6 *Două structuri posibile pentru automatul de comandă (CROM)*

Cea mai simplă schemă pentru automatul de comandă poate fi concepută pornind de la schema generală a unui automat cu stări finite de tip Mealy cu întârziere: circuit logic combinaţional (CLC) şi registru, în care ieşirile (comenzile) se iau de la ieşirea registrului (pentru evitarea hazardului combinaţional).

Circuitul combinaţional trebuie să conţină memoria de microprograme şi schema automatului poate fi construită folosind o memorie ROM, sau o arie logică programabilă de tip PLA, ca în figura 10.6. Se observă însă că este necesară o memorie de mare capacitate, care nu este folosită eficient. Pentru reducerea complexităţii memoriei, schema poate fi modificată, aşa cum se poate vedea în aceeaşi figură. Preţul plătit pentru reducerea complexităţii circuitului combinaţional este creşterea mărimii registrului. Acest lucru nu este însă un dezavantaj major, deoarece adăugarea unui număr de *k* bistabile în structura registrului nu modifică semnificativ complexitatea circuitului, care este stabilită în esenţă de complexitatea memoriei. Pentru că elementul central al automatului de comandă este memoria ROM, iar funcţionarea lui controlează modul în care se încarcă registrul cu date din memorie, structura se mai numeşte şi **controler şi ROM**, sau **CROM**.

În exemplul considerat, fiecărui cuvânt din memorie îi corespunde o singură microinstrucţiune. Deşi aceasta este cea mai obişnuită structură de memorie de microprograme, există şi alte moduri de organizare a memoriei (cu 2 microinstrucţiuni pe cuvânt, structurată pe blocuri, sau pe 2 nivele în tehnica nanoprogramării) ([Lupu 1995]).

Împărţirea cuvântului de microprogram în diferite zone de control numite **câmpuri** defineşte **formatul microinstrucţiunii**. Pentru exemplul considerat în figura 10.6, structura câmpurilor rezultă din ieşirile registrului de stare şi este dată în figura 10.7.

Tehnica microprogramării a fost folosită cu succes în domeniul minicalculatoarelor şi a microprocesoarelor de tip bit-slice microprogramabile şi este în continuare folosită de proiectanţii procesoarelor cu arhitecturi de tip **CISC** (**C**omplex **I**nstruction **S**et **C**omputer) (printre care şi procesoarele Pentium) sau a structurilor implementate cu circuite FPGA ([Bomar, 2002]).

Cuplajul dintre CROM şi RALU generează un **procesor elementar**. Procesarea presupune prelucrare în RALU şi control în CROM. Microinstrucţiunile asociate procesorului elementar sunt de cel puţin două feluri: de execuţie şi de salt. Denumirea de

Comenzi	Semnale de ieşire	Selecţie condiţie test (k biţi)	Adresa următoare (n – 1 biţi)	Selecţie MUX adresă (1 bit)

Fig. 10.7 *Alocarea câmpurilor unei microinstrucţiuni*

elementar se datorează faptului că procesorul poate să execute o singură funcție, cum ar fi cea de filtru numeric sau cea de transformator Fourier.

Pentru a realiza un **procesor universal**, la care ne vom gândi de acum încolo când vom auzi cuvântul simplu **procesor**, trebuie să construim un automat de comandă care să poată genera secvențele de comenzi asociate tuturor funcțiilor calculabile, adică un automat cu o infinitate de stări. Chiar dacă ne-am restrânge pretențiile la un număr rezonabil de funcții, acest lucru ar presupune un automat de foarte mare complexitate ([Ştefan, 1991]).

Reducerea complexității acestui automat se poate face prin definirea unui set relativ redus de funcții elementare, care au proprietatea că, prin combinarea lor se poate calcula orice funcție parțial recursivă. Acest lucru nu se realizează printr-o structurare suplimentară, care ar consta din introducerea unei bucle suplimentare, ci prin faptul că elemente din RALU pot comanda declanşarea unor secvențe în CROM.

Structura unui procesor este deja suficient de complexă pentru ca evoluția sa către sistemele de ordin superior să se facă pe două căi distincte:

- definirea unui nivel funcțional inferior microprogramării, ce are asociată o buclă suplimentară internă
- stimularea funcțiilor programabile ce implică o buclă suplimentară exterioară.

În primul caz nivelului microprogramării i se adaugă cel al **nanoprogramării**. Automatul de comandă se împarte în două automate: unul care generează microprograme şi altul care generează nanoprograme şi comunică direct cu RALU. Printre avantajele nanoprogramării se numără implementarea distribuită a structurii CROM, automatele de nanoprograme fiind mai simple funcțional, deci mai rapide şi mai ieftine, apoi posibilitatea de a declanşa paralelisme într-o structură de procesare şi realizarea unor funcții mai puternice la nivelul automatului de microprograme ([Ştefan, 1983]).

În al doilea caz se introduce o buclă externă suplimentară prin care se conectează la procesor o memorie, de regulă de tip RAM, care este folosită în bună parte pentru stocarea programelor. Această nouă structură este forma cea mai rudimentară a unui **calculator**.

10.2 Calculatorul

În memoria RAM externă procesorului se află date şi programe. Această memorie este rezultatul unei exteriorizări a memoriei din RALU într-o structură în afara procesorului, degrevându-l pe acesta de o capacitate de stocare mult prea mare.

Una din ideile cele mai importante ce au propulsat vertiginos din anii ’40 ştiința calculatoarelor a fost aceea de a stoca în aceeaşi memorie datele şi instrucțiunile. Ideea aparține matematicianului John von Neumann şi ea a folosit la realizarea calculatoarelor, fiind modelul dominant în realizările concrete de calculatoare. Reprezentarea simplificată a calculatorului tip von Neumann este dată în figura 10.8, unde procesorul se conectează cu memoria printr-un canal bidirecțional de comunicație.

Fig. 10.8 *Reprezentarea simplificată a calculatorului tip von Neumann*

Implementarea acestui model în maşini concrete s-a făcut fie prin realizarea procesorului în maniere cât mai compacte şi configuraţii cât mai standardizate în paralel cu creşterea posibilităţilor de a construi memorii cu capacităţi cât mai mari, fie prin definirea funcţiilor calculatorului exclusiv pe seama programelor stocate în memorie ([Ştefan, 1991]).

Modelul procesor-canal-memorie s-a constituit într-o **maşină universală** particularizată exclusiv prin informaţia stocată în memorie. Anularea corelaţiei dintre structura fizică şi funcţia maşinii, acţionându-se exclusiv la palierul informaţional, a permis dezvoltarea spectaculoasă a domeniului calculatoarelor prin definirea a două domenii net distincte: domeniul **hardware** care avea drept scop maşina universală realizată pornind de la tehnologii cât mai performante şi domeniul **software** care pornea de la maşina universală particularizând-o funcţional prin programare ([Ştefan, 1991]).

Conceptul de maşină universală a fost o interfaţă de maximă flexibilitate care a permis dezvoltarea paralelă şi independentă a celor două domenii mult timp. Dar evoluţia lor nu s-a făcut uniform. În domeniul hardware tehnologia a avut o evoluţie spectaculoasă care a dus la creşterea extraordinară a performanţelor procesoarelor şi memoriilor. În paralel, domeniul software proliferează abordând o problematică deosebit de bogată, diversitatea problemei fiind stimulată de distanţa mare ce exista între clasa imensă a aplicaţiilor şi maşina universală. Sigur că tehnologiile de realizare a programelor nu au avut o evoluţie la fel de spectaculoasă ca în cazul domeniului hardware, costul realizării programelor rămânând ridicat ([Ştefan, 1991]).

Această incompatibilitate între ritmul de evoluţie al tehnologiilor în cele două domenii va submina conceptul de calculator ca maşină universală programabilă. Vinovat de acest lucru este considerat canalul de legătură între cele două unităţi. Denumit "von Neumann bottleneck" el pare a fi responsabil de ineficienţa calculatoarelor clasice, în primul rând datorită stilului de programare pe care la stimulat, dacă nu chiar l-a impus.

"Von Neumann bottleneck" este o limitare atât fizică cât şi conceptuală. Fizic decuplează suportul informaţiei de structura care operează cu şi asupra ei, iar conceptual împiedică declanşarea unor paralelisme care sunt posibile pentru categorii largi de probleme. Stilul în care sunt realizate programele este stingherit de configuraţia fizică a modelului discutat ([Ştefan, 1991]).

Concluzia care se impune este că pentru structuri cu performanţe medii modelul von Neumann va continua să fie valabil. Pentru structuri care impun o creştere a performanţelor peste o anumită limită, sau pentru necesităţi de abordare a unor clase noi de probleme, cum ar fi de exemplu cele din domeniul inteligenţei artificiale, se impune elaborarea unor modele diferite. Dar ce fel de modele? O soluţie poate fi oferită de **paralelism**.

Înainte de încheierea acestui paragraf trebuie să mai spunem că structura completă de calculator în arhitectură von Neumann mai conţine blocul de comunicaţie cu exteriorul prin intrări/ieşiri. Canalul de comunicaţie între toate cele trei blocuri este bidirecţional şi se numeşte magistrală de date şi adrese.

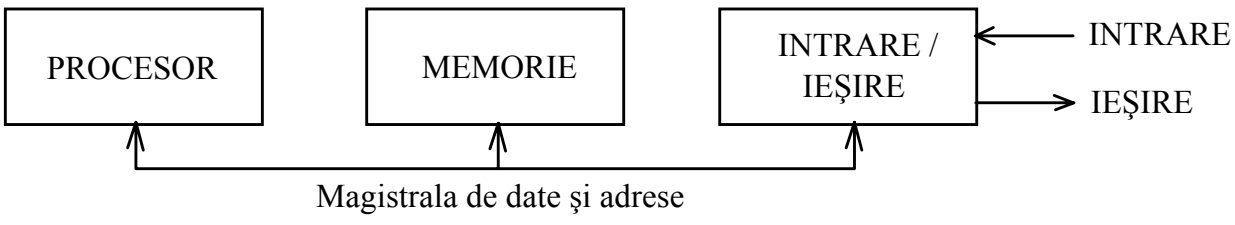

Fig. 10.9 *Structura completă de calculator von Neumann*

10.3 Paralelismul

Deşi ne-am obişnuit foarte mult cu prelucrarea secvenţială a informaţiei, ea nu este absolut obligatorie. Singura justificare pentru abordarea secvenţială o constituie limitările tehnologice. Totuşi, funcţiile definite recursiv, de exemplu, presupun în continuare existenţa secvenţelor. În consecinţă, la ora actuală, nu se poate pune problema unei structuri pur paralele. Ceea ce este realizabil, şi de altfel s-a şi realizat, este abandonarea structurilor pur secvenţiale şi înlocuirea lor prin structuri ce presupun funcţionarea în paralel a mai multor maşini secvenţiale.

Pentru conectarea mai multor procesoare în paralel se conectează mai multe maşini von Neumann ale căror canale de comunicaţie între procesor şi memorie se înlocuiesc printr-o **reţea de interconectare**. Paralelismul apărut generează însă un nou efect şi anume **concurenţa** între resursele disponibile.

În principiu există două posibilităţi de conectare a procesoarelor: arhitectura de tip **MIMD** (**M**ultiple **I**nstruction **M**ultiple **D**ata) în care se execută simultan mai multe instrucţiuni folosind mai multe date, şi arhitectura de tip **SIMD** (**S**ingle **I**nstruction **M**ultiple **D**ata) în care se aplică aceeaşi instrucţiune mai multor operanzi. În realitate această clasificare este pur teoretică. Maşinile reale construite până acum se situează pe poziţii intermediare şi chiar noi modele teoretice nu mai respectă strict această separare netă.

Reţeaua de interconectare de care am vorbit mai sus este o resursă fizică care în anumite cazuri poate deveni împovărătoare pentru o maşină paralelă. S-au conceput diverse reţele cu conectare directă, sau indirectă. Pentru reţelele cu conectare directă nodurile reţelei asigură procesarea şi transferul datelor între noduri. Reţelele cu conectare indirectă presupun existenţa unor noduri în care are loc procesarea şi noduri care asigură transferul între nodurile procesoare.

Se pare că un grad avansat de paralelism şi o eficienţă maximă se obţine atunci când granularitatea sistemului este maximă, adică atunci când folosim foarte multe elemente de procesare foarte mici şi simple. Exemplul cel mai bun îl oferă Connection Machine construit de Thinking Machines şi care conţine 65535 procesoare. Din memoria totală asociată reţelei un număr mare de biţi pot fi procesaţi în orice moment. Reţeaua este conectată totuşi la un calculator gazdă care rezolvă acele porţiuni de program care sunt mai eficient rulate pe un singur procesor, dar nu pe unul foarte simplu ([Ştefan, 1991]).

Ne amintim că structurile programabile de tip FPGA sunt formate din celule care pot realiza prin interconectare arhitecturi paralele. În acest fel, utilizatorul îşi poate crea propriile arhitecturi paralele, optimizate pe cât posibil pe aplicaţia dorită. Pe de altă parte, astfel de structuri pot fi configurate cu ajutorul unor algoritmi evolutivi care, având ca bază modelul evoluţiei din lumea vie, urmăresc generarea unor structuri care să optimizeze o anumită funcţie obiectiv. Care ar putea fi această funcţie obiectiv? Fie corectarea unei erori care poate apare datorită defectării unui tranzistor din structură, prin reconfigurarea circuitului, adică refacerea unor conexiuni pentru înlocuirea componentei defecte cu alta, aflată într-o arie nefolosită din circuitul integrat, fie modificarea unei configuraţii de circuit prin reconfigurare dinamică, în scopul adaptării structurii programabile la mediul în care se află. Adaptare poate însemna aici o modificare automată de circuit destinată implementării unor funcţii noi, poate chiar neprevăzute de proiectant ([Popa, 1998]).