PROCESOARE ÎN SISTEME PARALELE

În trecut, calculatoarele paralele conţineau procesoare proiectate special pentru a deservi sarcina paralelizării. Astăzi calculatoarele paralele utilizează procesoare obişnuite de uz general, soluţie mult mai convenabilă din punct de vedere economic dar şi tehnologic computaţional.

Un exemplu de procesor special dezvoltat pentru calcul paralel este **transputerul**, care apărut in anii 1980. O unitate de calcul bazată pe transputer conţine (Fig. 1):

- un procesor capabil să execute operaţii aritmetice de bază precum şi operaţii de intrare/ieşire;
- o memorie locală de tip SRAM de dimensiuni mici dar foarte rapidă;
- 4 canale de comunicaţie cu exteriorul;
- o interfaţa cu memoria externă.

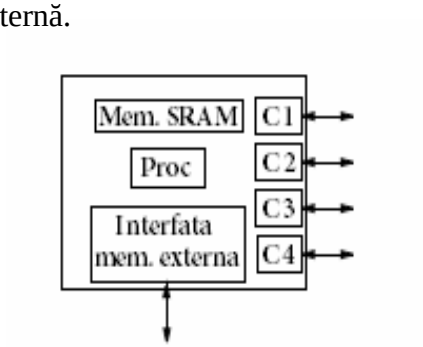

Figura 1. Structura de bază a unui transputer

Pe cele 4 canale de comunicaţie se pot lega alte transputere, formându-se o reţea multiprocesor cu topologie (configuraţie) diversă. Un exemplu de celulă cu 5 transputere este ilustrat în figura 2:

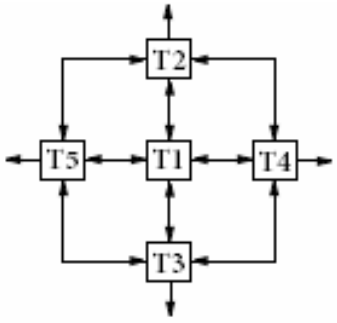

Figura 2. O celulă formată din 5 transputere

Dezavantajul folosirii transputerelor era dat de o putere computaţionala redusă, insuficientă pentru aplicaţiile tot mai complexe care au fost dezvoltate.

În anii 1990 au apărut procesoarele **RISC** (*Reduced Instruction Set Computer* – Calculatoare cu set redus de instrucţiuni), spre deosebire de procesoarele **CISC** (*Complex Instruction Set Computer*) care existau până atunci. Aceste procesoare RISC dădeau puterea computaţională de care era nevoie în aplicaţie şi ele au devenit din ce în ce mai ieftine, fiind produse pe scară largă.

Câteva exemple de procesoare RISC: procesorul MIPS de la SGI, procesorul POWER (IBM), SPARC (SUN), procesorul ALPHA (Cray).

Caracteristici ale procesoarelor RISC:

- au puţine moduri de adresare;
- au un format fix al instrucţiunilor (32b sau 64b);
- au un număr mare de registre care ajută la comutarea rapidă a contextului între programe;
- folosesc instrucţiuni de tip LOAD / STORE, instrucţiuni dedicate pentru încărcarea / salvarea datelor din memoria calculatorului în registrele procesorului (Fig.3):

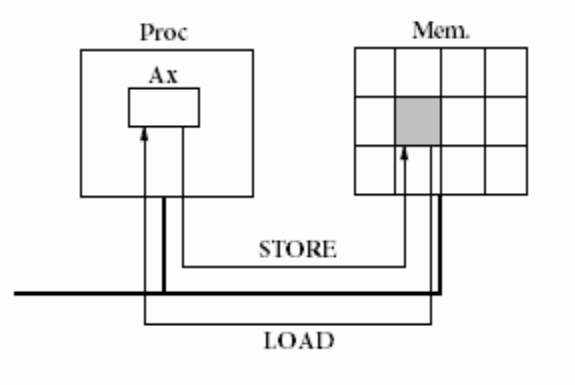

Figura 3. Instrucţiunile LOAD / STORE

- utilizează memoria cache pentru încărcarea datelor în procesor. Memoria cache aduce în avans instrucţiunile procesorului sau datele din memoria sistem DRAM, conferind avantajul unei viteze mărite de acces faţă de accesarea directă a datelor din memoria externă. (Fig.4):

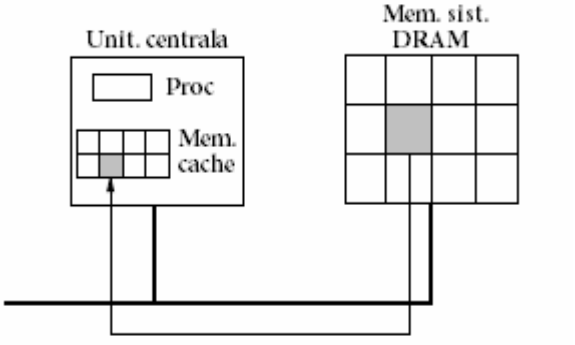

Figura 4. Memoria cache

Procesoarele RISC utilizează **arhitectura *pipeline*** (bandă de asamblare), în care o instrucţiune sau un task se descompune în mai multe operaţii care sunt executate independent de către unităţi de execuţie specializate (Fig. 5):

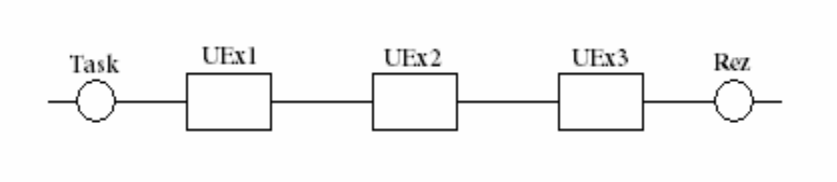

Figura 5. Arhitectura *pipeline*

În cazul procesoarelor RISC, arhitectura *pipeline* defineşte mai multe stagii (etape) de execuţie ale unei instrucţiuni. De exemplu, putem avea următoarele 4 stagii (Fig. 6):

Stagiul 1 – IF (*instruction fetch*): încărcarea instrucţiunii din memorie
Stagiul 2 – ID (*instruction decoding*): decodificarea instrucţiunii
Stagiul 3 – EX (*execute*): execuţia propriu-zisă a instrucţiunii
Stagiul 4 – WB (*write back*): scrierea rezultatului în registru sau memorie

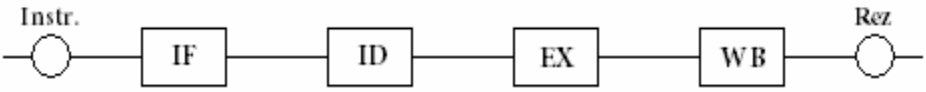

Figura 6. Stagiile de execuţie ale unei instrucţiuni

Diagrama de funcţionare în timp a acestei structuri este ilustrată în figura 7:

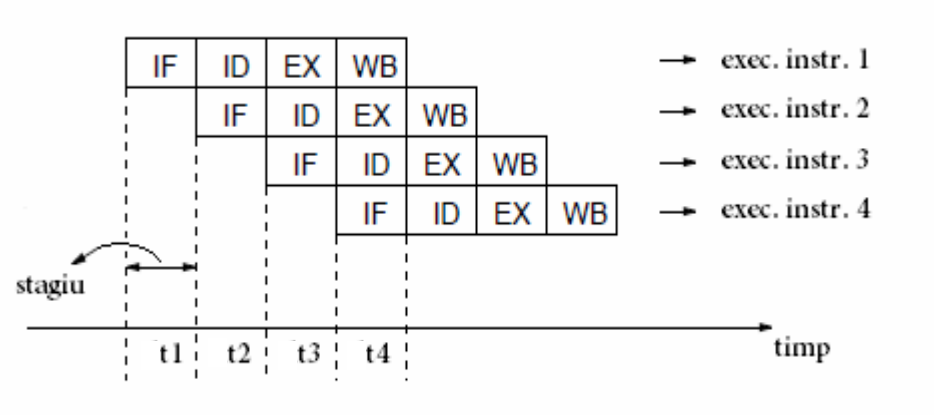

Figura 7. Diagrama execuţiei în timp pentru structura *pipeline*

Din această diagramă se poate observa că în orice moment de timp (de exemplu la momentul t4) se execută 4 stagii în paralel pentru 4 instrucţiuni diferite, rezultând astfel un paralelism temporal. La fiecare ciclu temporal, structura *pipeline* obţine un rezultat, astfel că putem executa câte o instrucţiune la fiecare ciclu procesor (şi nu în 4 cicli cum ar fi fost fără structura *pipeline*). Acest exemplu se aplică pentru procesoarele scalare. În acest caz există o singură linie *pipeline* şi se poate executa maxim o instrucţiune pe ciclu.

În prezent, majoritatea procesoarelor au o structură superscalară. La această structură există mai multe linii *pipeline* care operează în paralel. Ca rezultat, se pot executa mai multe instrucţiuni pe un ciclu procesor.

În figura 8 se prezintă o structură superscalară cu 3 linii *pipeline*:

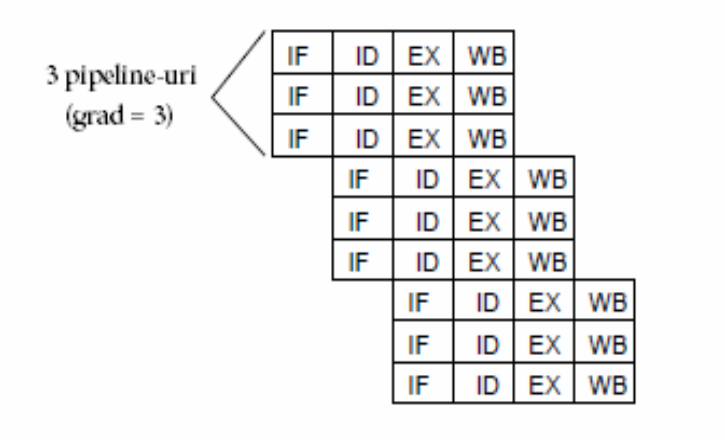

Figura 8. Diagrama de timp pentru structura superscalară

Câteva caracteristici ale arhitecturii superscalare:
- arhitectura permite executarea în paralel a mai multor instrucţiuni;
- se pot executa mai multe instrucţiuni pe un ciclu procesor;
- permite execuţia *out of order* (în afara ordinii) a instrucţiunilor: instrucţiuni independente pot fi executate în afara ordinii normale din program; condiţia pentru execuţia *out of order* este să nu existe dependenţe de date între instrucţiuni.

O altă metodă pentru creşterea eficienţei *pipeline* este folosirea arhitecturii *superpipeline*. În acest caz se reduce numărul de operaţii care se efectuează într-un stagiu, dar se măreşte numărul de stagii prin împarţirea fiecărui stagiu în mai multe substagii. Astfel putem creşte frecvenţa de tact a procesorului, pentru că se execută mai puţine operaţii într-un ciclu procesor decât la un stagiu obişnuit.

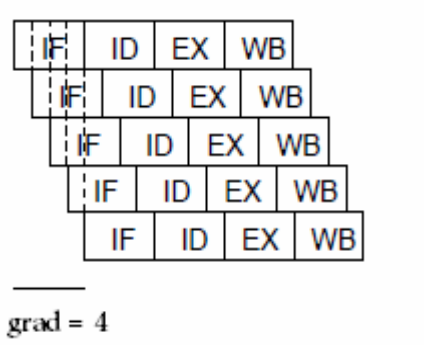

Figura 9. Diagrama de timp pentru structura *superpipeline*

În exemplul de mai sus, datorită faptului că un stagiu a fost împărţit în 4 substagii, vom putea creşte frecvenţa procesorului de 4 ori.

Avantajele arhitecturii *superpipeline:*

- creşte viteza de execuţie a instrucţiunilor datorită creşterii frecvenţei de ceas a procesorului;
- creşte numărul de operaţii procesate în paralel datorită creşterii numărului de stagii;
- creşte eficienţa operaţiilor executate într-un stagiu. De exemplu, dacă într-un *pipeline* obişnuit avem o operaţie care se execută într-un timp mai mic decât un ciclu procesor, rezultatul va fi disponibil tot la sfârşitul ciclului (Fig. 10 a). În schimb, la arhitectura *superpipeline*, rezultatul unei operaţii este disponibil cu frecvenţa tactului, care acum este mai rapid (Fig.10 b):

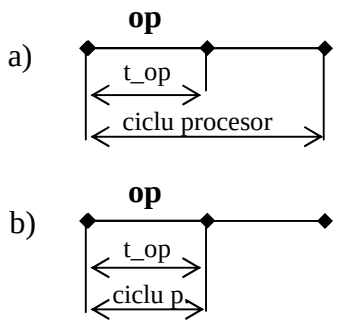

Figura 10. Execuţia unei operaţii corespunzătoare unui substagiu: a) într-o structură *pipeline*; b) într-o structură *superpipeline*

Dezavantaje ale arhitecturii *super pipeline*:

Principalul dezavantaj e reprezentat de faptul că dacă *pipeline*-ul trebuie golit de instrucţiunile încărcate în avans (de exemplu când apar instrucţiuni de salt în program), atunci există un număr mai mare de stagii care trebuie reiniţializate.

Aşa cum am văzut, procesoarele RISC sunt procesoare de uz general, dar există şi alte tipuri de procesoare care se aplică unui domeniu mai restrâns, obţinând performanţe mai bune pe acel domeniu specific (de exemplu procesoare DSP, procesoare VLIW si procesoare vectoriale). În continuare vor fi prezentate pe scurt procesoarele VLIW si procesoarele vectoriale.

Procesoarele VLIW (Very Large Instruction Word)

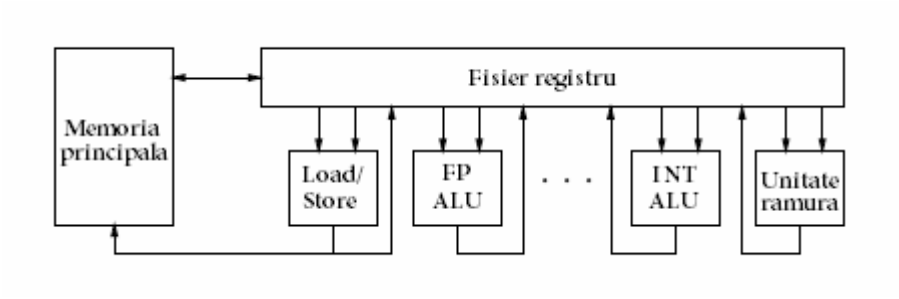

Figura 10. Structura internă a unui procesor VLIW

Procesorul VLIW e alcătuit din mai multe unităţi funcţionale echivalente unităţilor de execuţie din procesoarele superscalare, şi un fişier registru care conţine un număr mare de registre procesor (pot exista mai mult de 128 registre). O caracteristică a acestor procesoare este că ele dispun de un format mare al instrucţiunilor. Fiecare instrucţiune e compusă din mai multe sloturi de operaţii în care se plasează diferite operaţii de tip RISC, în aşa fel încât toate operaţiile dintr-o instrucţiune să poată fi executate in paralel.

Load/Store	FP add	FP mult	Ramura	. . .	INT ALU

Figura 11. Formatul instrucţiunii la procesorul VLIW

Programul compilator este cel care umple instrucţiunile cu operaţii care se pot executa în paralel. De aici rezultă caracteristica de *static scheduling* (planificare statică), adică determinarea paralelismului se face în timpul compilării şi nu al execuţiei. Rezultă astfel un avantaj al procesoarelor VLIW: ele nu necesită un hardware complex, cum era cazul la procesoarele superscalare (unde determinarea paralelismului se făcea prin hardware în timpul execuţiei).

Ca şi dezavantaj al procesoarelor VLIW: codul obiect rezultat în urma compilării este mai puţin compact decât la un procesor obişnuit, datorită faptului că nu toate sloturile dintr-o instrucţiune pot fi încărcate cu operaţii care se execută în paralel.

Diagrama execuţiei în timp a unei instrucţiuni este ilustrată în figura 12 (pentru un procesor având 3 sloturi pe instrucţiune).

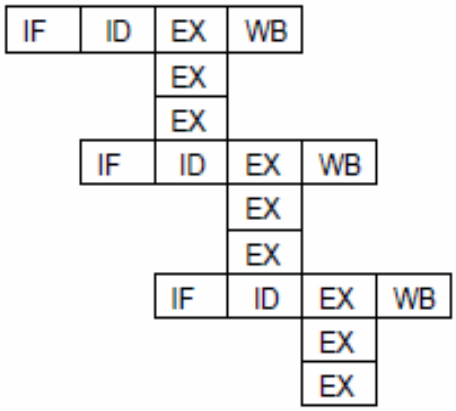

Figura 12. Diagrama execuţiei unei instrucţiuni la procesorul VLIW

Câteva exemple de procesoare VLIW:
- Philips Trimedia (utilizat pentru aplicaţii multimedia);
- arhitectura Intel pe 64 biţi (IA64) - o combinaţie între RISC si VLIW.

Procesoare vectoriale

În cele mai multe cazuri, procesoarele vectoriale nu funcţionează independent, ci sunt folosite pe post de coprocesoare alături de un procesor principal. Spre deosebire de procesoarele obişnuite, care operează cu scalari, procesoarele vectoriale pot opera pe vectori, adică pe şiruri de scalari. Ele pot fi de tipul registru-registru, în care vectorii se încarcă din registre vectoriale, iar rezultatele se salvează tot în registre vectoriale, sau pot să fie de tipul memorie-memorie, în care vectorii şi rezultatele se iau / se pun în memoria sistemului. Unităţile de execuţie vectoriale au o funcţionare de tip *pipeline* (fig.13).

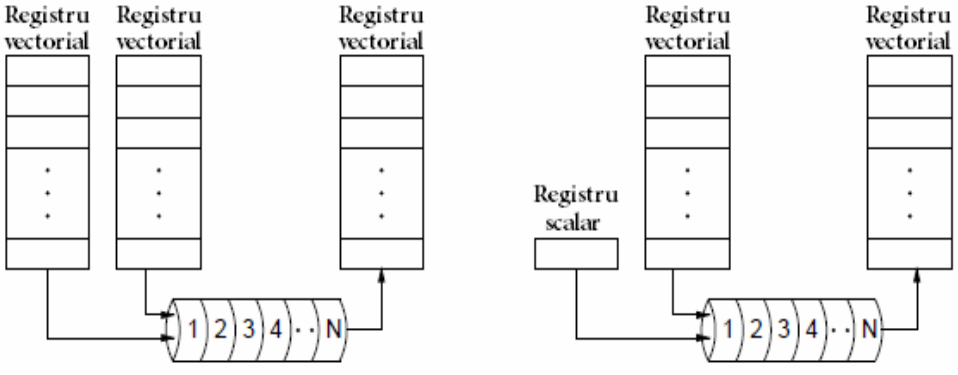

Figura 13. Unităţile de execuţie într-un procesor vectorial

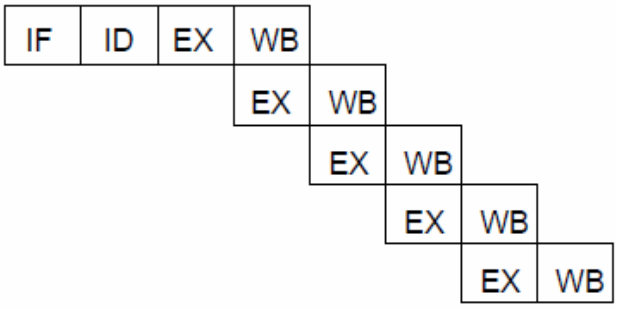

Figura 14. Diagrama de execuţie a unei instrucţiuni vectoriale

Domeniul de aplicabilitate al procesoarelor vectoriale este reprezentat de calculele ştiinţifice şi aplicaţiile multimedia, unde este necesară realizarea unui număr mare de operaţii pe structuri vectoriale.