ARHITECTURA CALCULATOARELOR 2003/2004
CURSUL 6

2.4 Alte arhitecturi

Pentru a avea o perspectivă mai largă, să studiem alte alternative la arhitectura de calculator prezentată în capitolele anterioare.

2.4.1 Arhitecturi CISC şi arhitecturi RISC

Proiectarea unui limbaj maşină implică luarea multor decizii, una din ele fiind dacă să construim o structură complexă, care sa poată decodifica şi executa o largă varietate de instrucțiuni, sau o maşină mai simplă, care să dispună de un set limitat de instrucțiuni. Ceea ce se obține in primul caz poarta numele de calculator cu set de complex de instrucțiuni (complex instruction set computer CISC); a doua opțiune conduce la realizarea unui calculator cu un set restrâns de instrucțiuni (reduced instruction set computer RISC). Cu cât structura procesorului este mai complexă cu atât este mai simplă efectuarea programării, deoarece poate fi utilizată o singură instrucțiune pentru execuția unei operați, în timp ce în cazul calculatorului mai simplu aceeaşi operație ar necesita o secvență de mai multe instrucțiuni. Dar structurile complexe sunt mai greu şi mai scump de realizat, iar realizarea lor poate fi mai costisitoare. În plus o mare parte dintre instrucțiunile complexe sunt destinate unor aplicații particulare, ceea ce poate duce la o creştere nejustificată a prețurilor.

Pentru a minimiza numărul de circuite necesare, procesoarele CISC sunt adesea realizate într-o arhitectură pe două niveluri în care fiecare instrucțiune maşină este executată de fapt ca o succesiune de instrucțiuni simple. În cadrul acestor arhitecturi, unitatea centrală de prelucrare dispune de o memorie interna specială denumită **micromemorie (micromemory)**, în care este stocat un microprogram – un program care controlează ciclul de extragere - decodificare - execuție al unității centrale de prelucrare. Într-un anumit sens unitatea centrală de prelucrare este de fapt un mic calculator programat să extragă, să decodifice şi să execute instrucțiunile din memoria principală a calculatorului.

Pe lângă faptul că utilizează o arhitectură CISC care nu utilizează circuite extrem de complexe care altfel ar fi necesare pentru a suporta un set bogat de instrucțiuni, soluția cu utilizarea unui microprogram permite adaptarea unei unități centrale de prelucrare pentru a include in limbajul maşină unele instrucțiuni speciale prin simpla modificare a microprogramului. Totuşi susținătorii arhitecturii RISC argumentează ca aceste avantaje nu justifică dezavantajul complexității suplimentare care se datorează existentei microprogramului. Ei susțin că o soluție mai bună o reprezintă proiectarea unei maşini simple care sa dispună de un set mic, dar bine ales, de instrucțiuni. Această abordare elimină complexitatea datorată existenței unei micromemorii şi conduce la obținerea unei unități centrale de prelucrare cu o structură mai simplă. Pe de altă parte, însă, programele reprezentate sunt mai lungi decât cele corespunzătoare unei arhitecturi CISC, deoarece trebuie executate mai multe instrucțiuni pentru realizarea operațiilor complexe, codificate printr-o singură instrucțiune în cadrul arhitecturii CISC.

În prezent pe piață există atât procesoare CISC, cât şi procesoare RISC. Procesorul Pentium dezvoltat de firma Intel Corporation, reprezintă un exemplu de arhitectură CISC; seriile de procesoare PowerPC, dezvoltate de Apple Computer, IBM şi Motorola, urmează arhitectura RISC.

În anii 1970 au fost multe experimente cu instrucțiuni foarte complexe, posibile datorită interpretării. Proiectanții au încercat să inchidă ”golul semantic“ între ceea ce pot să facă maşinile şi ceea ce limbajele de programare de nivel înalt cer. Cu greu cineva s-a gândit la proiectarea unor maşini mai simple, după cum puțini cercetători caută azi să dezvolte sisteme de operare, rețele, procesoare de text mai simple (poate din nefericire).

În 1980 un grup la Berkeley condus de David Patterson şi Carlo Sequin a început proiectarea cipurilor pentru VLSI CPU care nu au folosit interpretarea. Ei au inventat termenul **RISC** pentru acest concept. Puțin mai târziu în 1981, la Stanford, John Hennessy a proiectat şi fabricat un cip cumva diferit pe care el l-a numit **MIPS**. Aceste cipuri au evoluat în produse comerciale importante, respectiv SPARC şi MIPS. Aceste procesoare noi au fost semnificativ diferite față de procesoarele comerciale ale zilei. Pentru ca acestor noi CPU - uri nu li s-a impus să fie compatibile cu produsele existente, proiectanții lor au fost liberi să aleagă noile seturi de instrucțiuni care să maximizeze perormanța sistemului în ansamblu. În timp ce accentul inițial a fost pus pe instrucțiunile simple care au putut fi executate repede, şi-au dat curând seama că proiectarea instrucțiunilor care să fie rapid lansate în execuție este cheia spre o bună performanță. Cât timp durează o instrucțiune este acum mai puțin important decât câte instrucțiuni pot fi lansate pe secundă. În acest timp, caracteristica care a atras atenția fiecăruia a fost numărul mic de instrucțiuni, tipic in jur de 50. Acest număr a fost mai mic decât 200 şi 300 stabilit de companii ca DEC VAX şi IBM pentru main frames.

Suporterii RISC - ului au poclamat cea mai bună metodă de a proiecta un computer era să se obțina un număr mic de instrucțiuni simple care se execută într-un ciclu al căii de date. Argumentul lor a fost că deşi maşina RISC foloseşte 4 sau 5 instrucțiuni pentru a face ceea ce maşina CISC face într-o instrucțiune, pentru că instrucțiunile RISC sunt de 10 ori mai rapide, deoarece nu sunt interpretate, atunci RISC este în avantaj. Deasemenea merită arătat că în acelaşi timp viteza memoriei principale a ajuns la viteza memoriei ROM contol stores, favorizand puternic maşinile RISC. S-ar putea crede că datorită avantajelor performanței oferite de tehnologia RISC, maşinile RISC (cum este DEC Alpha) au câştig de cauză asupra maşinilor CISC (cum este Pentium Intel) pe piață. Nimic de felul acesta nu s-a întâmplat. De ce nu? Mai intâi de toate, există **cerința compatibilității** cu variantele anterioare şi faptul că s-au investit multe miliarde de dolari în software pentru linia Intel. În al doilea rând, surprinzător **Intel a fost capabil să folosească aceleaşi idei chiar în arhitectura CISC**. Începând cu 486, CPU-urile Intel conțin un nucleu RISC care execută cele mai simple (şi tipic cele mai frecvente) instrucțiuni într-un singur ciclu al căii de date, în timp ce interpretarea instrucțiunuilor mai complicate se face într-un mod uzual CISC. Rezultatul net este că instrucțiunile simple sunt rapide şi instrucțiunile mai puțin simple sunt lente. În timp ce această abordare hibridă nu este aşa de rapidă ca proiectarea pura RISC este global competitivă şi pentru că permite încă programelor vechi să ruleze nemodificate.

2.4.2 Principiile proiectarii calculatoarelor moderne

Acum, mai mult decât atunci când primele maşini RISC au fost introduse, anumite principii de proiectare au fost acceptate ca o bună cale pentru a proiecta computere în concordanță cu

nivelul tehnologic hardware disponibil. Există un set de principii de proiectare, numite uneori **principiile de proiectare RISC,** pe care proiectanții fac tot posibilul pentru a le urma. Constrângerile externe, cum ar fi şi cerința de a păstra compatibilitatea cu unele arhitecturi existente, cer uneori compromisuri.

- **Toate instrucțiunile sunt direct executate de hardware:** Toate instrucțiunile uzuale sunt direct executate de hardware. Ele nu sunt interpretate de microinstrucțiuni. Eliminarea nivelului interpretării se dovedeşte o sursă de creştere a vitezei. Pentru calculatoarele care implementează setul de instrucțiuni CISC, instrucțiunile complexe pot fi sparte în instrucțiuni mai mici care sunt executate ca o secvență de microinstrucțiuni. Acest pas în plus poate micşora viteza, dar dacă aceste instrucțiuni complexe sunt intâlnite mai rar atunci pierderea poate fi acceptabilă.

- **Maximizarea ratei la care instrucțiunile sunt executate:** Calculatoarele moderne recurg la diferite trucuri pentru a maximiza performanțele, cea mai impotantă soluție este de a executa cât mai multe instrucțiuni pe secundă. Deci, dacă se pot lansa 500 de milioane de instrucțiuni pe secundă atunci am obținut un procesor 500 MIPS, nu contează cât de mult durează în total execuția unei instrucțiuni. Acest principiu sugerează că metoda paralelismului joacă un rol important în creşterea vitezei de execuție, deci executarea unui număr mare de instrucțiuni lente într-un timp scurt este posibilă numai dacă se execută mai multe instrucțiuni în acelaşi timp. Instrucțiunile sunt întotdeauna încărcate în ordinea programului, dar nu sunt executate în aceeaşi ordine pentru că unele instrucțiuni necesită resurse care momentan pot fi ocupate. Desigur dacă instrucțiunea 1 setează un registru, iar instrucțiunea 2 utilizează acest registru trebuie avut grijă ca instrucțiunea 2 să nu citească registrul până când acesta nu va conține date valide. Pentru a ține totul în ordine e nevoie de multe inregistrări, dar se obține o performanță mare prin executarea unui număr mare de instrucțiuni în acelaşi timp.

- **Instrucțiunile trebuie să fie uşor de decodificat:** Un loc foarte important în rata de execuție a instrucțiunilor il reprezintă decodarea instrucțiunilor, operație care determină ce resurse sunt necesare. Orice mijloc care poate ajuta acest proces este folositor. Aceasta impune construirea instrucțiunilor în aşa fel încât să difere cât mai puțin, adică lungime fixă, cu un număr mic de parametri. Cu cât diferențele sunt mai mici cu atât mai bine.

- **Numai LOAD şi STORE trebuie să se refere la memorie:** Calea cea mai simplă de a impărți operațiile în paşi separați necesită ca operanzii pentru majoritatea instrucțiunilor să vină de la regiştri şi să se ducă la regiştri. Operația de a muta operanzi din memorie în regiştri poate fi facută cu instrucțiuni distincte. Deoarece acesul la memorie poate să dureze mult şi durata este imprevizibilă, aceste instrucțiuni pot fi suprapuse cu alte instrucțiuni dacă ele nu fac altceva decât să mute operanzi între memorie şi regiştri. Aceasta observatie înseamnă că doar LOAD şi STORE trebuie să se refere la memorie.

- **Existența unui număr mare de regiştrii:** Deoarece accesarea memoriei este lentă, este necesar un număr mare de regiştri, astfel încât odată un cuvânt extras el să poată fi păstrat acolo până când nu mai este nevoie de el. Stocarea informației din regiştri în memorie, ănseamnă şi reîncărcarea ei din memorie atunci când este nevoie de ea, ceea ce trebuie evitat pentru ca înseamnă pierdere de timp. Acest lucru poate fi ocolit dacă avem destui regiştri.

2.4.3 Prelucrare paralelă

Într-un subcapitol anterior am precizat că există o limită în ceea ce priveşte dezvoltarea de calculatoare mai rapide. Motivul este acela că semnalele electrice se propagă printr-un cablu cu o viteza care nu poate depăşi viteza luminii.(Calculatoarele optice, deşi sunt in prezent un domeniu promiţător de cercetare, sunt şi ele afectate de această limitare). Cum lumina parcurge o distanţă de aproximativ 30 de centimetri într-o nanosecundă (o miliardime de secundă) înseamnă că este necesar de un interval de cel puţin două nanosecunde pentru ca unitatea de comandă din unitatea centrală de prelucrare sa extragă o instrucţiune dintr-o celulă de memorie aflată la o distanta de 30 de centimetri.(Cererea de citire care trebuie trimisă memoriei solicită cel puţin o nanosecundă, iar trimiterea instrucţiunii din memorie la unitatea de comandă are nevoie şi ea de cel puţin o nanosecundă). În consecinţă extragerea, decodificarea şi execuţia unei instrucţiuni într-un asemenea calculator consumă mai multe nanoscunde. De aceea, creşterea vitezei de execuţie a unui calculator devine in ultimă instanţă o problema de miniaturizare şi, deşi s-a avansat extrem de mult în acest domeniu, există o limită care nu poate fi depăşită.

Într-un efort de rezolvare a acestei dileme, cercetătorii şi-au îndreptat atenţia de la viteza de execuţie către conceptul de capacitate de transfer (troughput). În loc să măsoare timpul consumat pentru realizarea unei anumite operaţii, capacitatea de transfer se referă la cantitatea totală de operaţii pe care le poate efectua calculatorul într-u anumit interval de timp. Un exemplu privind modul în care poate fi îmbunătăţită capacitatea de transfer a unui calculator, fără ca acest fapt să necesite creşterea vitezei de execuţie, îl reprezintă tehnica denumită prelucrare simultană (pipelining). Acest termen provine din analogia cu introducerea de obiecte - in cazul nostru instrucţiuni - intr-o conductă la unul dintre capete şi scoaterea lor pe la celălalt capăt. În orice moment, în conducta (pipe) există mai multe instrucţiuni, fiecare dintre ele aflându-se într-un alt stadiu de prelucrare. În particular, în timp ce se execută o instrucţiune, altă instrucţiune este decodificată şi încă o altă instrucţiune este extrasă din memorie.

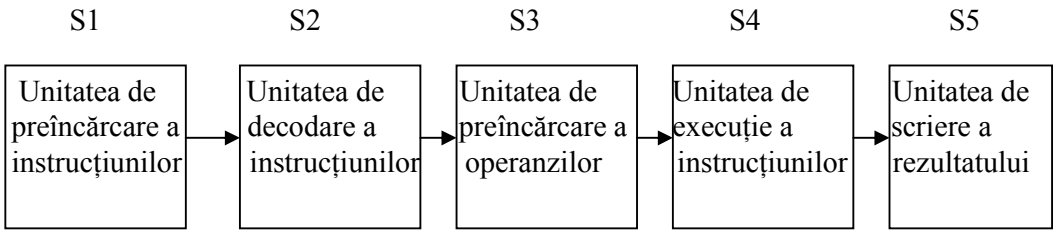
Utilizând un astfel de sistem, deşi fiecare instrucţiune necesită aceeaşi cantitate de timp pentru a fi extrasă din memorie, decodificată şi executată, capacitatea totală de transfer a calculatorului creşte de trei ori, deoarece simultan sunt prelucrate trei instrucţiuni. (În realitate, un factor egal cu trei este arareori atins datorită existenţei instrucţiunilor de salt – JUMP. De exemplu, dacă apare o instrucţiune de salt, conducta trebuie golită deoarece în ultimă instanţă, nu instrucţiunile pe care le conţine sunt necesare. Astfel, orice câştig care s-a obţinut prin preâncărcarea instrucţiunilor respective se pierde.)

2.4.3.1 Paralelismul la nivelul instrucţiunilor

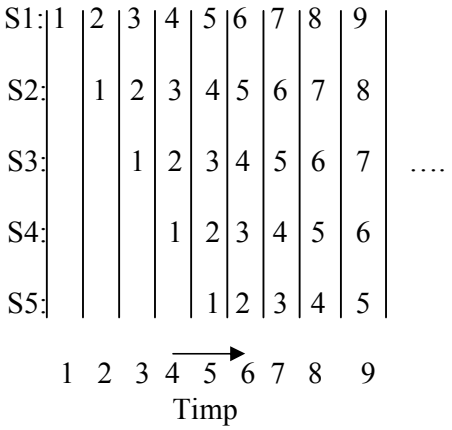
Proiectanţii de calculatoare sunt constant preocupaţi de a înbunătăţi performanţele maşinii pe care o crează. Creşterea frecvenţei de ceas este o cale, dar pentru orice proiect nou există o limită în ceea ce reprezintă această posibilitate în acel moment tehnologic. Frecvent, majoritatea proiectanţilor iau în considerare paralelismul pentru a îmbunătăţi şi mai mult performantă pentru o frecvenţă de ceas dată. Paralelismul se manifestă în două forme generale: paralelism la nivelul instrucţiunilor şi paralelism la nivelul procesorului. În primul caz paralelismul este exploatat cu instrucţiuni individuale pentru a obţine un raport instrucţiuni / secundă cât mai mare. În a doua vaariantă paralelismul se referă la mai multe procesoare care lucrează în paralel la aceeaşi problemă. Fiecare abordare are avantajele ei.

- **Pipelining**

S-a ştiut de mult că preîncărcarea instrucţiunilor din memorie este un element important pentru viteză de execuţie a instrucţiunilor. Pentru a evidenţia această problemă ne intoarcem în istorie la IBM Stretch (1959) care avea capacitate de a preîncărca din memorie în avans instrucţiunile, aşa încât ele să fie acolo când este nevoie de ele. Instrucţiunile erau încărcate într-un set de regiştri denumiţi 'prefetch buffer'. În acest fel când era nevoie de o instrucţiune, aceasta era luată din această memorie tampon decât să se aştepte completarea unui ciclu de citire din memorie. De fapt preîncărcarea împarte execuţia instrucţiunilor în două faze: preîncărcarea şi executarea lor. Conceptul de pipeline oferă o strategie mult mai avantajoasă. Decât să împartă execuţia în două parţi, o împarte în multiple părţi, fiecărei părţi fiindu-i dedicată o componentă hardware, acestea executându-se în paralel. Figura 2.8a ilustrează un pipeline cu 5 unităţi denumite stagii. Stagiul 1 preîncarcă instrucţiunea din memorie şi o pune într-o memorie tampon până când este cerută. Stagiul 2 decodează instrucţiunea, determină tipul ei şi ce operanzi necesită. Stagiul 3 localizează şi preîncarcă operanzii din regiştri sau din memorie. Stagiul 4 face operaţia de a executa instrucţiunea. Stagiul 5 scrie rezultatul instrucţiuni în registru.



a) O conductă (pipe) cu cinci stagii



b) Starea fiecărui stagiu în timp. Sunt ilustrate nouă perioade de ceas.

Figura 2.8 Pipeling

In figura 2.8b se vede cum funcţionează un pipeline în timp. În primul ciclu, stagiul 1 lucrează la instrucţiunea 1, preîncărcând-o în memorie. În ciclul 2 al ceasului, stagiul 2 decodează instrucţiunea 1 în timp ce stagiul 1 preîncarcă instrucţiunea 2. În ciclul 3, stagiul 3 preîncarcă operanzi pentru instrucţiunea 1, stagiul 2 decodează instrucţiunea 2 şi stagiul 1

preîncarcă instrucțiunea 3. În ciclul 4, stagiul S4 execută instrucțiunea 1, S3 preîncarcă operanzii pentru instrucțiunea 2, S2 decodează instrucțiunea 3 şi S1 preîncarcă instrucțiunea 4. În final în timpul ciclului 5, S5 scrie rezultatul instrucțiunii 1, în timp ce celelalte stagii lucrează asupra următoarei instrucțiuni.

Să găsim o analogie pentru a face conceptul de pipeline mai clar. Imaginațivă o fabrică de prăjituri care are o bandă de înpachetat iar de-a lungul acesteia se află cinci lucrători. La fiecare 10 secunde, muncitorul 1 pune o cutie goală pe bandă, aceasta este cărată până la muncitorul 2 care pune o prajitură în ea. Puțin mai târziu cutia ajunge la muncitorul 3 unde este inchisă şi sigilată. Apoi ajunge la muncitorul 4 care pune o etichetă pe ea. În final muncitorul 5 ia cutia de pe bandă şi o pune într-un container care apoi va fi transportat la magazin. Aceasta este metoda în care lucrează pipeline-urile în calculatoare: fiecare instrucțiune trece prin diferite stagii până la comletarea execuției.

Să ne întoarcem la pipeline-urile noastre din figura 2.8, să presupunem că ciclul acestei maşini este de 2ns. Atunci pentru ca o instrucțiune să se execute complet durează 10ns. La prima execuție cu durata de 10ns pe instrucțiune dă impresia că viteza maşinii este de 100 MIPS, dar de fapt este mai bună. La fiecare 2ns o nouă instrucțiune este completată, deci viteza este de 500 MIPS-uri nu 100 MIPS. Pipeline-urile permite un raport între latență - latency (cât de mult durează executarea unei instrucțiuni) şi banda de frecvență a procesorului - processor bandwidth (cât de multe MIPS-uri face un CPU). Cu un ciclu de T nsec şi n stagi în pipeline, "latency" este n x T nsec şi lățimea de bandă de 1000/T MIPS-uri.

- **Arhitectura superscalara**

Dacă un pipeline este bun atunci două pipeline-uri sunt foarte bune. Un proiect posibil cu două pipeline-uri pentru un CPU, bazat pe figura 2.8 este prezentat în figura 2.9. Aici, o singură unitate de preîncărcare preîncarcă instrucțiunile şi le repartizează către celelalte pipeline-uri care au propriul ALU pentru a opera în paralel. Pentru a fi posibul ca cele două instrucțiuni să se execute în paralel trebuie ca ele să nu utilizeze aceleaşi resurse şi nu trebuie să depindă de rezultatul celeilalte. Ca şi cu un singur pipeline, fie compilatorul trebuie să garanteze acest lucru (de exemplu hardware-ul nu verifică şi dă rezultate incorecte în cazul în care instrucțiunile sunt incorecte) sau conflictele trebuie detectate şi eliminate în timpul execuției de o componenta hardware suplimentară.

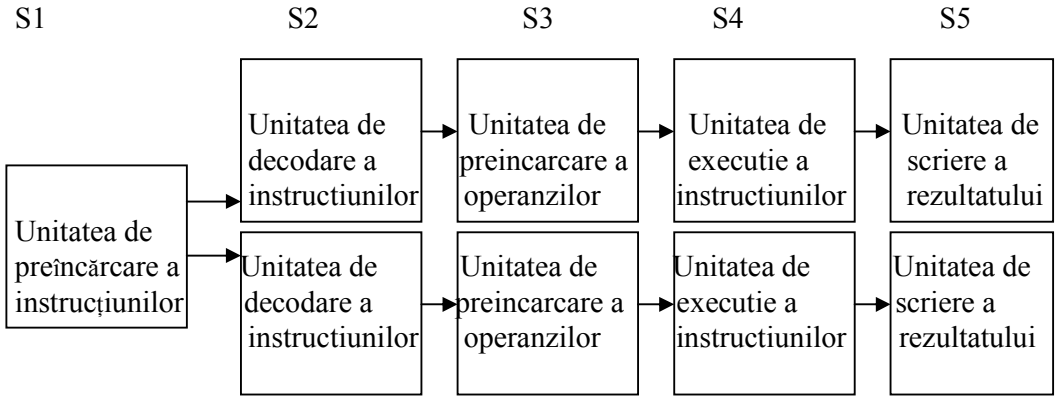


Figura 2.9 Cinci stagii pe două lini cu o unitate de preîncărcare a instrucțiunilor

Pipeline-urile simple sau duble sunt utilizate în general de maşinile RISC iar începând cu 486 Intel a introdus pipeline-urile în propriile procesoare. 486 are un pipeline, iar Pentium are doua pipeline-uri cu câte 5 stagi ca în figura 2.9. Diviziunea exactă a sarcinilor între stagiile 2 şi 3 (numite decode – 1 şi decode – 2) este uşor diferită decât aceea din exemplul nostru. Principalul pipeline denumit "u pipeline" poate executa instrucțiuni arbitrare, iar pipeline-ul secundar denumit "v pipeline" poate executa numai instrucțiuni simple cu întregi (şi o instrucțiune simplă în virgulă mobile FXCH).

Reguli complexe stabilesc dacă un grup de instrucțiuni sunt compatibile şi pot fi executate în paralel. Dacă instrucțiunile din pereche nu sunt suficient de simple sau sunt incompatibile atunci decât una se execută în "u pipeline". Instrucțiunea incompatibilă este păstrată şi grupată u următoarea instrucțiune. Instrucțiunile sunt întotdeauna executate în ordine. Compilatoare adaptate pentru Pentium care generează instrucțiuni compatibile fac ca programele generate să se execute mai repede decât cele obținute cu compilatoare mai vechi. Măsurările arată că un program optimizat pentru Pentium se execută de două ori mai repede decât unul executat pe un 486 la aceeaşi frecvenşă de ceas. Câstigul este datorat în întregime celui de al doilea pipiline. Mergând la 4 pipeline-uri ideea este bună, dar necesită prea mult hardware.

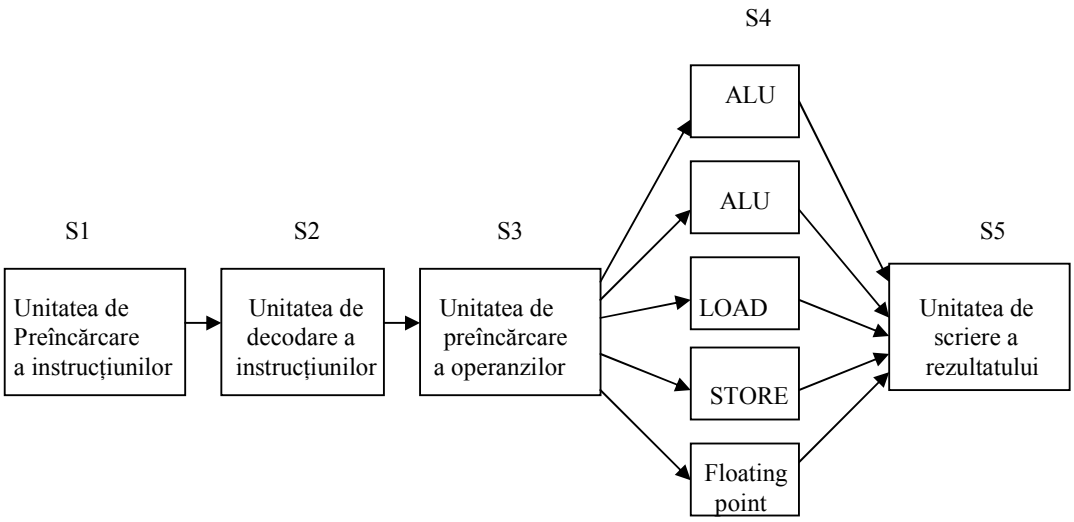


Figura 2.10 Un procesor superscalar cu cinci unități

CPU mai recente abordează altfel ideea. Adică ideea de bază ar fi de a avea un singur pipeline, dar care include mai multe unități funcționale 2.10. De exemplu Pentium 2 are o structură similară cu aceea din figura 2.10. Termenul de arhitectură superscalară a existat din 1987. Rădăcinile ei se găsesc însă mai devreme cu 30 ani pe vremea calculatorului CDC 6600. 6600 încărca o instrucțiune la fiecare 100 nsec şi o pasa la una din cele 10 unități pentru executare în paralel în timp ce CPU lua următoarea instrucțiune. Ideea implicită ar fi că un procesor superscalar poate prelucra instrucțiuni în stagiul S3 mult mai repede decât în stagiul s4. Dacă stagiul S3 poate executa o instrucțiune în 10 nsec şi toate unitățile din S4 pot de asemenea executa o instrucțiune în 10 nsec, atunci nu mai mult de o unitate din S4 poate fi ocupată la un moment dat, ceea ce ar nega întreaga idee. În realitate majoritatea unităților din stagiul S4 au nevoie de mai mult de un ciclu de ceas pentru o execuție, cum sunt cele care accesează memoria sau fac operații în virgulă mobilă. Aşa cum se vede în figura 2.10 pot exista multiple ALU în stagiul S4.

2.4.3.2 Paralelism la nivel de procesor

Cererea pentru calculatoare din ce în ce mai rapide pare a fi de nesatisfacut. Astronomii vor să simuleze ce s-a întâmplat în primele microsecunde după Bing Bang, economiştii vor un model al economiei mondiale, iar tinerii vor să joace jocuri interactive 3D multimedia pe internet cu prieteni lor virtuali. În timp ce CPU este din ce în ce mai rapid, eventual vor ajunge să intre în cursă cu viteza lumini, care este de 20 cm/ns în fir de cupru sau în fibra optică oricât de inteligenţi ar fi inginerii de la Intel. Procesoare mai rapide generează caldura mai mare a cărei disipare este o problemă. Paralelismul la nivel de instrucţiune ajută un pic, dar pipeline-urile şi superscalarea câştigă cu adevărat un factor de 5 sau 10. Pentru a obţine un factor de 50, 100 sau mai mult, singura cale este de a crea calculatoare cu mai multe procesoare.

Alte soluţii pentru creşterea capacităţii de transfer sunt cele din categoria prelucrării paralele (paralel procesing) în care sunt utilizate mai multe procesoare pentru execuţia operaţiei curente. Un argument în favoarea acestei abordări îl reprezintă modelul creierului uman. Tehnologia prezentului se apropie de posibilitatea realizării de circuite electronice cu aproape la fel de multe circuite de comutare câţi neuroni există în creierul uman (se crede că neuronii sunt echivalentul natural al circuitelor de comutare), dar în ciuda acestui fapt capacitatea calculatoarelor din zilele noastre reprezintă încă doar o fracţiune din cea a minţii umane. Se consideră că acest fapt este rezultatul utilizării ineficiente a componentelor calculatorului datorita arhitecturii acestuia . La urma urmei dacă un calculator are o memorie foarte mare, dar o singură unitate de prelucrare, o mare parte din circuite vor fi inactive în majoritatea timpului. Din acest motiv, susţinătorii prelucrării paralele se pronunţă în favoarea calculatoarelor multiprocesor, care constituie în opinia lor configuraţii cu un factor de utilizare mult mai ridicat.

În prezent există diferite tipuri de calculatoare realizate conform acestui principiu. Una din abordări o reprezintă cuplarea mai multor unităţi de prelucrare, fiecare dintre ele fiind similară unei entităţi centrale de prelucrare dintr-un calculator cu un singur microprosesor, la o memorie principală unică. În această configuraţie fiecare procesor poate lucra independent de celelalte, coordonarea eforturilor realizându-se prin transmiterea de mesaje stocate în celulele din memoria comuna. De exemplu, atunci când un procesor trebuie să execute o activitate complexă, el poate stoca în memoria comună un program referitor la o parte a acelei activităţi şi apoi să ceara altui procesor să execute programul. Se obţine un calculator în care diferite secvenţe de instrucţiuni sunt executate cu diferite seturi de date, arhitectura care poartă numele de MIMD (multiple - instruction stream, multiple - data stream - flux multiplu de instrucţiuni, flux multiplu de date) spre deosebire de arhitectura uzuala SISD (single-instruction stream, single - data stream – flux simplu de instrucţiuni, flux simplu de date).

O altă variantă de arhitectură multiprocesor o reprezintă conectarea procesoarelor în aşa fel încât să execute la unison aceeaşi secvenţă de instrucţiuni, fiecare cu un set propriu de date; ceea ce rezultă o arhitectura SIMD (single - instruction stream multiple - data stream). Asemenea calculatoare sunt utile în aplicaţii în care aceeaşi operaţie trebuie efectuată asupra fiecărei mulţimi de elemente similare dintr-un bloc mare de date.

Altă abordare a conceptului de prelucrare paralelă o reprezintă construirea de sisteme de mari dimensiuni alcătuite din conglomerate de calculatoare mai mici fiecare dintre ele având propria memorie şi propria unitate de prelucrare. În cadrul unei asemenea arhitecturi, fiecare dintre calculatoarele mici este conectat la vecinii săi, astfel încât sarcinile repartizate

întregului sistem pot fi împărţite între calculatoarele care-l compun. În acest fel, dacă o operaţie alocată unuia dintre calculatoarele interne poate fi împărţită în operaţii mai mici independente, acel calculator poate cere vecinilor săi să efectueze operaţii în paralel. În consecinţă, operaţia iniţială se poate realiza într-un timp mai scurt decât ar fi fost necesar în cazul în care s-ar fi utilizat un calculator cu un singur procesor.

Problemele curente care apar la dezvoltarea şi utilizarea calculatoarelor multiprocesor se referă la echilibrarea încărcării (load balancing) procesoarelor, adică la alocarea dinamică a operaţiilor solicitate la diverse procese astfel încât toate procesoarele sa fie utilizate în mod eficient. Strâns legată de aceasta este şi problema scalării, respectiv împărţirii operaţiei curente într-un număr de suboperaţii corespunzător numărului de procesoare disponibile. O altă problemă o reprezintă complexitatea ridicată a alocării operaţiilor distribuite . Într-adevăr, pe măsură ce coordonarea interacţiunilor dintre diferite operaţii creşte exponenţial. Dacă avem 4 operaţii atunci exista 6 perechi potenţiale de operaţii care s-ar putea să comunice între ele. Dacă numărul de operaţii este 5, numărul de căi de comunicaţie posibile creste la 10; în cazul a 6 operaţii, numărul respectiv creşte la 15.

- **Calculatoare matriciale**

Multe probleme din ştiinţă şi inginerie implică matrice sau date care au o structură regulată de organizare. Se efectuează aceleaşi calcule asupra multor date diferite în acelasi timp.

Regularitatea şi structura acestor programe le face ideale pentru creşterea vitezei de execuţie prin paralelism. Sunt două metode prin care s-au putut executa mari programe ştincifice în timp scurt.

Un procesor matriceal (array processor) este alcătuit dintr-un număr foarte mare de procesoare identice care execută acelaşi cod pentru diferite date. Primul calculator matriceal din lume a fost proiectat la Universitatea din Illinois şi a fost numit ILLIAC 4 (vezi figura 2.11). Planul original a fost de a construi o maşină alcătuită din 4 cuadranţi fiecare cuadrant avand 8x8 zone de elemente procesor/memorie. Câte o unitate de control pe fiecare cuadrant vehiculează instrucţiunile, fiecare procesor execută instrucţiunile în secvenţă fixă utilizând datele din

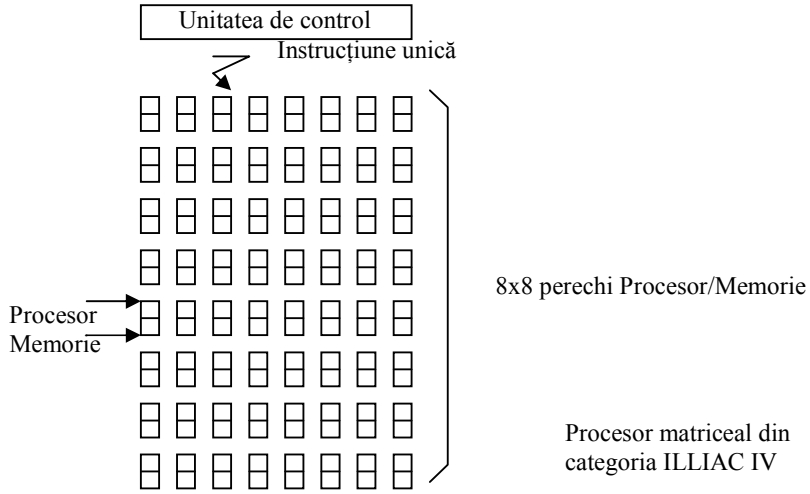


Figura 2.11 Calculator matriceal

propria memorie, date care erau înmagazinate la inițializare. Costul fiind prea mare s-a realizat doar un singur quadrant, dar care avea performanța de 50 megaflops (milioane de operati în virgulă mobilă pe secunda). Se spune că dacă s-ar fi construit în întregime ar fi ajuns la performanța de 1 gigaflop, ar fi dublat puterea de calcul pe intregul glob.

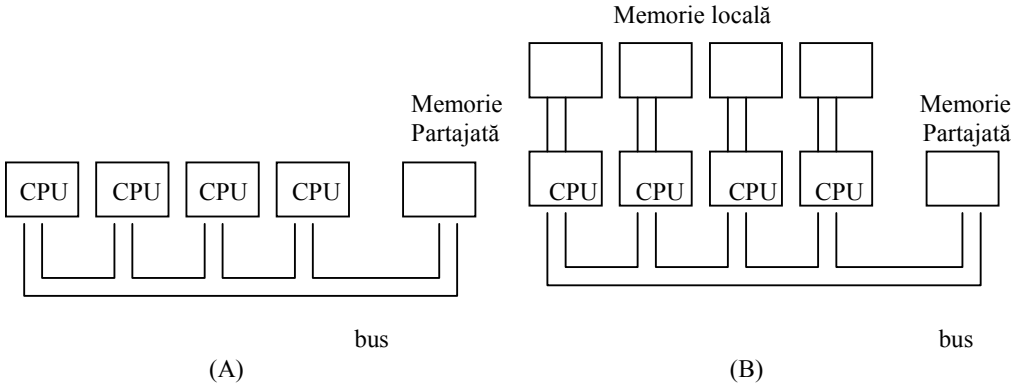
Un procesor vectorial (vector processor) apare pentru un programator ca şi unul matriceal. Ca şi procesorul matriceal, acesta este eficient în executarea unor secvențe de operații asupra unor perechi de date. Dar, spre deosebire de procesorul matriceal, aici toate operațiile adiționale se se execută într-un unic sumator bazat pe tehnica pipeline. Compania Seymour Cray a produs foarte multe procesoare vectoriale, începând cu Cray-1 în 1974 şi continuând cu modelele actuale. Procesorul matriceal şi cel vectorial lucrează ambele asupra unei matrice de date. Amândouă execută o singură instrucțiune, care de exemplu adună elementele dintr-o pereche vector. Dar, în timp ce procesorul matriceal face acest lucru prin atâtea sumatoare câte elemente sunt în vector, procesorul vectorial lucrează cu conceptul de registru vectorial (vector register) care constă într-un set de regiştri convenționali care pot fi încărcați printr-o singură instrucțiune din memorie, care de fapt îi încarcă din memorie serial. Apoi o instrucțiune pentru adunare vectorială realizează adunarea perechilor cu elemente din doi astfel de vectori prin utilizarea unui sumator pipeline. Rezultatul este tot un vector care va fi înregistrat într-un registru vectorial. Procesoarele matriceale se mai fabrică, dar foarte rar, pentru că ele necesită execuția unui singur cod asupra diferitelor date de intrare. Acestea pot executa anumite calcule mai repede decât procesoarele vectoriale, dar necesită foarte mult hardware şi sunt greu de programat. Procesoarele vectoriale, pe de altă parte, pot fi adăugate la procesoarele convenționale. Rezultatul este că partea de program care poate fi vectorizată poate fi executată mai rapid folosindu-se de avantajele unității vectoriale, iar celelalte pot fi executate convențional pe un singur procesor.

- **Multiprocesoare**

Elementele de procesare într-un procesor matriceal nu sunt CPU independente deoarece există o singura unitate de control în tot sistemul. Primul sistem paralel cu multiple procesoare este un multiprocessor, un sistem cu mai mult de un processor împărțind acesul la o singură memorie (precum oamenii dintr-o sală împart o singură tablă). Pentru că fiecare CPU poate citi şi scrie orice parte a memoriei ele trebuie să coopereze (prin program) pentru a nu se încurca reciproc. Diferite implementări schematice sunt posibile. Cel mai simplu este să avem o singură magistrală şi mai multe CPU precum şi o singură memorie. O asemenea structură bazată pe o magistrala este prezentată în figura următoare. Multe companii produc astfel de sisteme

Nu trebuie multă imaginație pentru a ne da seama că având multe procesoare care accesează aceeaşi memorie vor apare conflicte. Proiectanții de sisteme multiprocesr au venit cu diferite soluții pentru această problemă. O variantă este prezentată in figura 2.12, care dă fiecărui procesor ceva memorie locală care nu este acesibilă celorlalte procesoare. Această memorie poate fi folosită pentru instrucțiuni sau pentru date care nu trebuie partajate. Accesul la această memorie privată nu se face prin magistrala comună astfel micşorându-se traficul.

Multiprocesoarele au avantajul, în comparație cu alte calculatoare paralele, că modul de programare pentru sisteme cu o singură memorie comună este uşor de utilizat. De exemplu, imaginați-vă un program care caută o celulă de cancer într-o fotografie luată prin microscop.



(A) Arhitectura multiprocesor cu o singură magistrală
(B) Arhitectura multicalculator cu memorie locală

Figura 2.12 Arhitecturi multiprocesor

Fotografia digitizată este păstrată în memoria la care fiecare procesor are acces. Astfel fiecare procesor poate studia o parte din fotografie în timp ce altă parte este studiată de alt procesor.

- **Multicalculatoare**

Cu toate că multiprocesoarele cu un numar mic de procesoare (<=64) sunt relativ uşor de construit, cele mai mari sunt surprinzător de greu de construit. Dificultatea constă în a conecta toate procesoarele la o aceeaşi memorie. Pentru a ocoli aceste probleme, mulți proiectanți au abandonat ideea de a avea memorii partajate şi construiesc doar sisteme constând dintr-un număr mare de calculatoare conectate, fiecare având memoria sa proprie, dar fără memorie comună. Aceste sisteme se numesc multicalculatoare. CPU este un sistem multicalculator care comunică prin trimiterea de mesaje de la un calculator la altul, ceva de genul unui e-meil, dar mult mai rapid. Pentru sistemele mai mari, conectarea fiecărui calculator la celelalte este ineficient, deci sunt folosite topologii ca rețele 2D, rețele 3D, arborescente şi inelare. Ca rezultat, mesajele de la un calculator la altul trebuie de obicei să treacă prin unul sau mai multe calculatoare intermediere (switches) pentru a ajunge de la sursă la destinație. Cu toate acestea timpul de transmitere de câteva microsecunde poate fi atins fără mare dificultate. Multicalculatoarele cu aproape 10.000 CPU, au fost construite şi puse în acțiune. Pe când multiprocesoarele sunt mai uşor de programat, multicalculatoarele sunt mai uşor de construit. Ca urmare s-a intensificat cercetarea în domeniul sistemelor hibrid care combină proprietățile bune ale fiecărei variante. Asemenea calculatoare încearcă să prezinte iluzia de memorie împărțită, fără a trece prin chinul de a o chiar construi.