

Tomasulo

Algoritmul lui Tomasulo a fost implementat pentru a rezolva diverse probleme din acea perioadă (dezvoltat în 1967 de către Robert Tomasulo, IBM) precum: numărul mic de registrii de tip floating point disponibili pe procesor, latența joașă stocării a memoriei (fiind inventată introducerea memoriei cache ca parte standard a arhitecturii), multe unități de procesare de același tip nu puteau să folosească capacitatea lor de calcul dar și problema dependențelor a operațiilor în funcție de ambele valori din registrele generând hazarduri de tip Write after write (WAW) acolo unde se crează situația unei operații care să lea să scrie în același loc în alt registru sau hazard de tip Write after Read.

Exemplu hazard WAW: add r_1, r_2, r_3
ld $\underline{S\bar{r}_1}, 100(r_5)$

Exemplu hazard WAR: add $r_1, \underline{r_2}, r_3$
ld $\underline{r_2}, 100(r_5)$

Acest algoritm a venit cu scopul de a rezolva aceste probleme și pentru a eficientiza operațiile, permitând execuția acestora folosind mai multe unități de execuție.

Pentru ca acest algoritm să poată fi implementat, sunt necesare mai multe elemente care pe care le voi descrie mai jos:

Magistralele de date comuni (CDB) au rolul de a păstra precedentele (anterioare) esențiale, permitând în același timp, ca mai multe suprapunere a operațiilor independente.

De asemenea, datorită magistralelor de date comuni, multe funcționale pot accesa rezultatul oricarei operații fără să implice regisztrul de rezultat (floating-point-register), permitând astfel mai multor unități să aștepte un rezultat al unui calcul anterior fără a aștepta rezolvarea conflictului accesării regisztrului.

În cadrul magistralei de date cîine se desfășoară și detectarea răzădănilor și controlul execuției. Stepile de rezolvare conținând caud a instrucțiunii poale fi executate, proces mult mai ușor decât cazul în care ar fi existat o singură unitate de răzădăni dedicată.

Astfel, magistrala de date cîine oferă un algoritm hardware pentru a exploata eficiente și automatizat o mulțime mare de excepții. Prin intermediul magistralei de date cîine se scriu rezultatele.

Ordinea execuției instrucțiunilor este servențială. Este doar ca efectul executării instrucțiunilor preluate

prim acest algoritm, ca exemplu, exceptiile generate în traiul rutinei, să se producă în același ordine în care s-ar fi produs în rutinea aceluiși set de instrucțiuni pe un procesor bazat pe rutinări instrucțiunilor rezervate în cîndă capitolul că acesta sunt executate rezervativ (cont-of-order).

Pentru ca instrucțiunile să fie executate corect non-sequential, algoritmul lui Tomasulo folosește redemnările registrilor.

Functia executată trebuie împărțită în clăi părțile independente: execuție pentru punct fix și una pentru virgule mobile. Un avantaj al acestei împărțiri ar fi executarea simultană de astfel în tip de execuție mai mic.

Pot apărea însă și probleme decuante programul trebuie să conțină emblele flori, și virgule fixă, dar și virgule mobile, astfel acest tip de operație nu este întotdeauna benefic programatorului.

Pentru a genera independența în execuția instrucțiunilor în casul operațiilor cu virgule mobile, arhitectura extinsă generează set de instrucțiuni la propriul set de acumulatori.

Pentru operațiile cu virgule fixă, există un singur set de acumulatori generali și următoarele operații dependente specifice ale program-

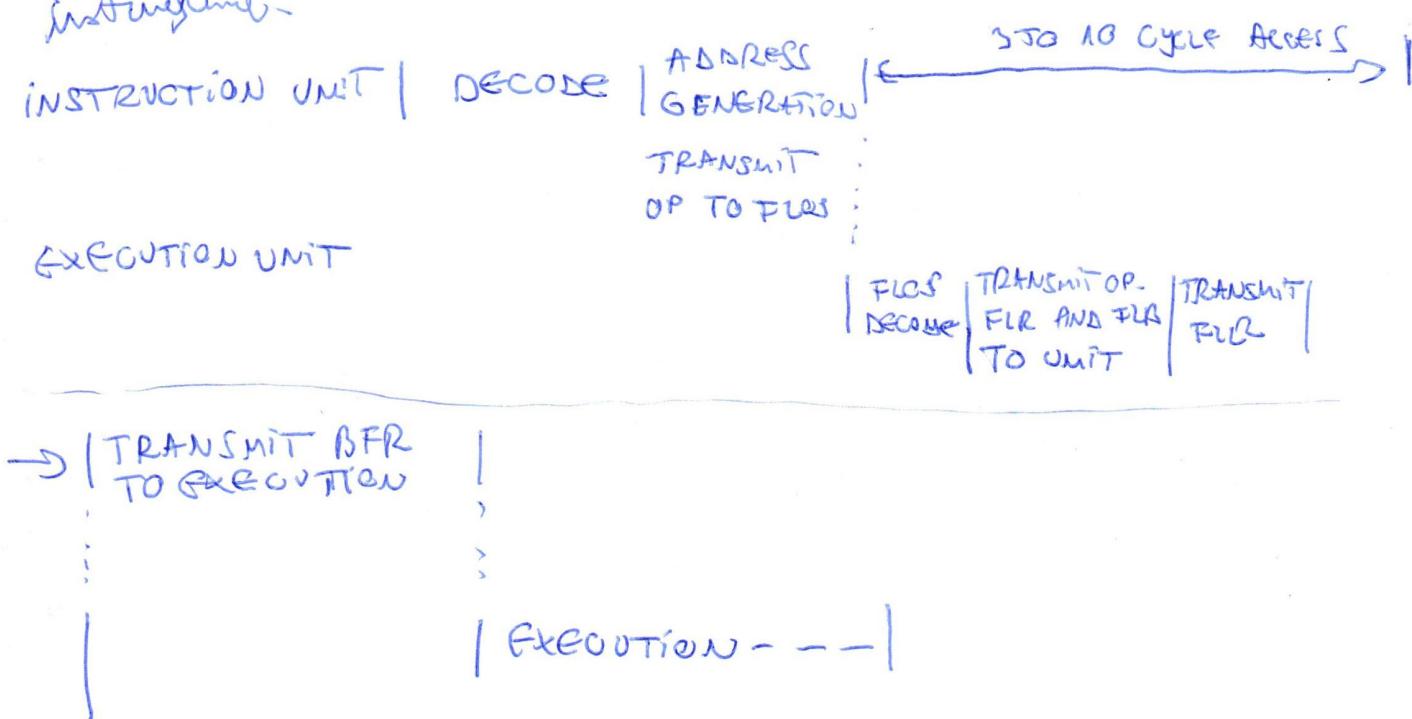
Din acestă cauză, clasificarea fecalelor într-un fel
în funcție de tipul ei, vîrghii fix sau vîrghii
mobili nu mai este suficientă.

Pentru o funcționare corectă, trebuie să obțină rețeaua sănătății fizice și nutriționale curente cu cele anterioare care nu au fost temibile, funcționare care este asigurată de majoritatea de date come.

Unitatea de instrucții este responsabilă cu prepararea instrucțiunilor de stocare-la-dispozitiv și a celor pe care le registrează cu reținere notabilă (FLR).

Contentul registratoriilor verișor mobile (FLR) este setat în egalitate cu rezultatul operațiunii de excepție a operațiunilor de stocare, acolo unde se specifică sursele operațiunilor care urmăresc să fie salvate în memorie.

Relationship between a relay and its instruction will be
an FLO (regular recall) decode path processed with
instruction:



Conform figurii de la pagina anterioră, când FLOPs decodă procesarea din memoriă, bufferul care va primi operandul nu a fost încă lăsat în memoria.

Bitii de control asociati cu bufferul care determină transmiterea conținutului la suitor sunt setați de FLOPs.

Suportul protejează informațiile de control atunci când registruul său este setat complet din buffer, informațiile de control determină suportul să transmită date către Ry.

În cazul în care instrucția preluată este o funcție aritmetică de stocare - enregistrare, registruul de verificare mobilă este transmis de decoder la unitatea de compresie/stocare iar operandul de stocare este trimit ca în sarcină.

Unitatea va executa operația după care va trimite rezultatul la registru R₂ alia după primirea bufferului.

Pentru instrucțiunile aritmétice de registru - register, datele regisotre în verificare mobilă sunt transmise pe calea successivă către unitatea de execuție.

Memoria va fi tratată ca o funcție aritmetică storage-to-register, conținutul registruului în verificare mobilă este transmis către un buffer de stocare în loc să fie transmis către o unitate de execuție.

Nicăieri registrul în verificare mobilă nu poate participa la o operație / instrucție dacă este alocat altor instrucții care nu a fost terminată.

Exemplu:

LD FO FLB1 LOAD register FO from buffer 1

MUL FO FLB2 MULTIPLY register FO by buffer 2

⇒ The cardinal precedence principle (descrie valoarea).

Pentru soluționarea acestor situații, trebuie să fie reușită existența disponibilităților, trebuie realizată corectă a instrucțiunilor disponibile și trebuie să se facă distincție între sevența dată și numărul

pă ex. exemple precum: LD FO, PLB1; LD F2 PLB2.

Astfel, respectând privile date arafe se asigură integritatea logică a programării, iar ceea ce-a fi înăsajurănd se poate obține performanțe ridicate.

Fiecare dintre cele patru registre cu privire la adresele sunt asociate un set „ocupat”.

Exemplu:

LD	FO, D	$FO = D$
LD	F2, C	$F2 = C$
LD	Fn, B	$Fn = B$
AD	FO, E	$FO = D \cdot E$
AD	F2, FO	$F2 = C + D \cdot E$
AS	Fn, A	$Fn = A + B$
AD	F2, Fn	$F2 = A + B + C + D \cdot E$

Prin intermediul seturii adreselor ocupate) ar trebui ca adresele să împărtășească să se execute simultan decarece același utilizator folosește diverse adrese.

Din grafoul de sincronizare reiese că aceasta

suprapunere nu se produce și se pierd multe cicluri de cca pe fiecare turnărătore primă adreare folosind rezultatul înmulțirii se sumatorul are nevoie de acest rezultat deoarece. Soluția în acestă problemă ar putea fi folosirea mai multor sumatoare. Problema ar fi că în unele situații, circuitele de execuție ar fi utilizate o mulțime mare de turnărătore.

Cea mai buna soluție este asocierea mai multor seturi de registre cu fiecare unitate de execuție. Fiecare set se menține în stătățile de rezervare^b. Înmulțirea pentru instrucțiunile în stătățile de rezervare și prin redemulsarea registrilor hardware, algoritmul microprogramabil RAR și elerulă folosinduile WAW ar că am deserviț la beneficiile acestui algoritm. Algoritmul în Toronto a contribuit semnificativ la dezvoltarea calculatoarelor moderne performante.