VERMA DORIAN-ALEXANDRU
334 CC

Algoritmul lui Tomasulo

În ziua de astăzi, eficiența sistemelor de calcul este un aspect foarte important. Având în vedere faptul că există numeroase limitări din punct de vedere al vitezei de procesare atunci când se încearcă sporirea eficienței prin intermediul folosirii resurselor materiale, aspect sugerat și de legea lui Moore, soluțiile pentru creșterea performanței pot fi regăsite în implementarea unor arhitecturi care să exploateze idei precum folosirea paralelismului, pipelining, dynamic scheduling etc.

Robert Tomasulo a fost un informatician care a adus contribuții în cadrul subiectului descris mai sus, acesta fiind inventatorul algoritmului care îi poartă numele. În anul 1994, i-a fost acordat premiul Eckert-Mauchly pentru crearea acestui algoritm.

Algoritmul lui Tomasulo este un algoritm hardware care are la bază programarea dinamică a instrucțiunilor. De asemenea, acesta permite execuția Out of Order și folosirea mai multor unități de execuție într-un mod cât mai eficient. A fost dezvoltat în anul 1967 de către Robert Tomasulo și a fost implementat prima dată în cadrul modelului IBM System/360 Model 91 de FPU (floating point unit)

Din punct de vedere al arhitecturii sugerate de algoritmul lui Tomasulo, se impune prezenta unor stații de rezervare. Aceste rezervation stations sunt legate la unitățile funcționale: ADDER și, MULTIPLY/DIVIDE. Aceste legături sunt realizate prin intermediul CDB, Common Data Bus.

Unitățile funcționale pot accesa unele valori rezultate din operațiile anterioare fără să implice scrierea acestora în cadrul unor registre floating-point. Astfel, pentru a realiza o operație, unitățile vor aștepta mai puțin.

De asemenea, stațiile de rezervare implică și o distribuire din punct de vedere al detectării hazardurilor și, al Controlului execuției operațiilor. Astfel, aceste stații se vor ocupa de împreună de aceste task-uri, mai mai fiind nevoie, spre exemplu, de o unitate specială de control a hazardurilor.

Din punct de vedere al organizării, stațiile de rezervare conțin următoarele câmpuri: O_p, V_j/V_k, Q_j/Q_k, și BUSY.

O_p reprezintă un opcode, ce al operației instrucțiunii.

V_j și V_k reprezintă valorile operanzilor sursă folosiți în cadrul instrucțiunilor.

Q_j și Q_k reprezintă stația de rezervare care va genera operandul sursă dorit. Practic, acestea semnifică codificarea pe biți a stației de rezervare.

BUSY este setat pe 1 pentru componentele ocupate (stație de rezervare sau unitatea funcțională) sau pe 0 dacă nu sunt ocupate.

Algoritmul lui Tomasulo implică 3 stagii. O instrucțiune care este trimisă spre execuție trebuie să treacă prin aceste stagii ce vor fi descrise mai jos:

Prima etapă, denumită și Issue, sau etapa de Start, implică încărcarea instrucțiunii spre a fi executată. Aceasta se trimite spre execuție doar dacă unitățile ADDER sau MULTIPLY, DIVIDE și, stațiile de rezervare sunt libere. Notabil este faptul că regiștrele sunt redenumite la acest pas, fiind eliminate astfel hazardele de tipul WAR (Write after Read) și, WAW (write after write). Inițial, instrucțiunilor li se acordă un spațiu într-o coadă (instruction queue). În cadrul acestei etape, instrucțiunea este preluată din această coadă. Apoi, dacă unitatea funcțională aferentă este liberă, executăm instrucțiunea (o trimitem mai departe în următoarea etapă). Dacă nu este liberă, atunci așteptăm eliberarea unității pentru instrucțiunea respectivă. Procedeul de mai sus este realizant în cazul în care instrucțiunea se află în setul de regiștrii. Dacă unitatea funcțională nu este liberă, atunci instrucțiunea este pusă în stația de rezervare. Pentru instrucțiunile Load/Store, aceasta va fi încărcată în stația de rezervare.

A doua etapă este etapa de execuție, etapă în cadrul căreia se realizează operația din cadrul instrucțiunii. În cadrul acestei etape, nu se realizează execuția până când toți operanzii sunt calculați. Astfel, aici se rezolvă și, hazardul de tip RAW (read after write) Dacă lipsește un operand, atunci se așteaptă până când acesta apare pe CDB. Dacă operanzii sunt calculați și, disponibili, în cazul în care instrucțiunea este una care se rezolvă cu o operație specifică ALU, atunci aceasta se execută în unitatea funcțională corespunzătoare. Dacă instrucțiunile sunt de tipul

load sau store, atunci, pentru load, se execută instrucțiunea dacă unitatea de memorie este liberă, iar pentru store, se așteaptă până când valoarea să fie scrisă înainte de a trimite-o spre unitatea de memorie.

A treia etapă este etapa de scriere a rezultatului. Dacă instrucțiunea implică operații de tip ALU, atunci rezultatul este scris în banculă de regiștrii, iar dacă este o instrucțiune de tip store, atunci aceasta este scrisă în unitatea de memorie. Dacă instrucțiunea implică ALU, atunci rezultatul se pune pe CDB înainte de a fi scris în banculă de regiștrii. Rezultatul ajunge la regiștrii prin intermediul magistralei.

Mai jos este un exemplu de rulare a algoritmului, în funcție de instrucțiunile care se află în coadă:

Instrucțiunile care se află în coadă sunt:

```
LD      F6, 34(R1)
LD      F2, 45(R2)
MULTD   F0, F2, F4
SUBD    F8, F6, F2
DIVD    F10, F0, F6
ADDD    F6, F8, F2
```

În cele ce urmează sunt redate tabelele cu conținutul stațiilor de rezervare și a regiștrilor.

Mai jos sunt tabelele pentru statiile de rezervare

Tabelul pentru statiile de rezervare după realizarea primei instructiuni load sunt:

	Busy	Op	Vj	Vk	Qj	Qk
Add1	0	subd	Load1			Load2
Add2	0	addd			Add1	Load2
Add3	1					
Mult1	0	multd		F4	Load2	
Mult2	0	divd		Load1	Mult1	

Tabelul pentru registre (valorile acestora) după realizarea operatiei de load (prima dintre acestea).

F0	F1	F4	F6	F8	F10	F12 ...
Mult1			Add2	Add1	Mult2	

Intrarea specifică din tabela registrilor se schimbă atunci când în registru se scrie o valoare, Qi reprezentând codificarea statiei de rezervare al cărei rezultat ute scris în banculde registrii.

Astfel, tabela registrilor după terminarea instructiunii (a doua de load) este:

F0	F2	F4	F6	F8	F10	F12 ...
Mult1	0		Add2	Add1	Mult2.	

Iar după operatia de multiply, va arăta așa:

F0	F2	F4	F6	F8	F10	F12 ...
0	0		0	0	Mult2	

Tabelul pentru stațiile de rezervare după operația de multiply va arăta așa:

	Busy	Q_i	V_j	V_k	Q_j	Q_k
Add1	0					
Add2	0					
Add3	0					
Mult1	0					
Mult2	1	divd	(mult1)	(Load1)		

Folosirea conceptelor de stații de rezervare (reservation stations) register renaming (redenumire a regiștrilor) și, utilizarea CDB a dus la o dezvoltare importantă a design-ului computerelor performante. Aceste notiuni sunt folosite (sau derivate ale acestora) în industrie. Conceptul de dynamic scheduling folosit în algoritmul lui Tomasulo este implementat (cu modificările aferente) în cadrul multor arhitecturi moderne.