OPTIMIZAREA SCHEMELOR DE PREDICTIE PENTRU RAMIFICATIILE DE PROGRAM ÎN PROCESOARELE SUPERSCALARE AVANSATE

1. Scopul lucrarii

Lucrarea de fata insista pe investigarea unor arhitecturi moderne de predictoare a ramificatiilor de program (**branch**), în vederea reducerii penalitatilor introduse de instructiunile de ramificatie în procesoarele pipeline superscalare. Vor fi explorate în mod critic metodologiile de predictie existente, vor fi îmbunatatite, optimizate si stabilite limitarile fundamentale. Se vor stabili schemele de predictie optimale asociate diferitelor tipuri de ramificatii din program.

2. Memento teoretic

Predictia prin hardware reprezinta una din cele mai performante strategii actuale de gestionare a ramificatiilor de program. Aceste strategii hardware de predictie a branch-urilor au la baza un proces de predictie "run - time" a ramurii de salt conditionat precum si determinarea în avans a noului PC. Ele sunt comune atât procesoarelor scalare cât si celor cu executii multiple ale instructiunilor. Cercetari recente insista pe aceasta problema, întrucât s-ar elimina necesitatea reorganizarilor soft ale programului sursa si deci s-ar obtine o independenta fata de masina.

Necesitatea predictiei, mai ales în cazul procesoarelor cu executii multiple ale instructiunilor (VLIW, superscalare) este imperios necesara. Notând cu BP (Branch Penalty) numarul mediu de cicli de asteptare pentru fiecare instructiune din program, introdusi de salturile fals predictionate, se poate scrie relatia:

$$BP = C\,(1-Ap)\,b\,IR \qquad (1)$$

unde s-au notat prin:

C= Numarul de cicli de penalizare introdusi de un salt prost predictionat

Ap= Acuratetea predictiei

b= Procentajul instructiunilor de salt, din totalul instructiunilor, procesate în program

IR= Rata medie de lansare în executie a instructiunilor

Se observa ca BP(Ap=0)=C b IR, iar BP(Ap=1)=0 (normal, predictia este ideala aici). Impunând un BP=0.1 si considerând valorile tipice: C=5, IR=4, b=22.5%, rezulta ca fiind necesara o acuratete a predictiei de peste 97.7% ! Cu alte cuvinte, la o acuratete de 97.7%, IR=4/1.4=2.8 instr./ciclu, fata de IR=4 instr./ciclu, la o predictie perfecta (Ap=100%). Este o dovada clara ca sunt necesare acurateti ale predictiilor foarte apropiate de 100% pentru a nu se "simti" efectul defavorabil al ramificatiilor de program asupra performantei procesoarelor avansate. O metoda consacrata în acest sens o constituie metoda "**branch prediction buffer" (BPB).** BPB-ul reprezinta o mica memorie adresata cu cei mai putin semnificativi biti ai PC-ului aferent unei instructiuni de salt conditionat. Cuvântul BPB este constituit în principiu dintr-un singur bit. Daca acesta e 1 logic, atunci se prezice ca saltul se va face, iar daca e 0 logic, se prezice ca saltul nu se va face. Evident ca nu se poate sti în avans daca predictia este corecta. Oricum, structura va considera ca predictia este corecta si va declansa aducerea instructiunii urmatoare de pe ramura prezisa. Daca predictia se dovedeste a fi fost falsa structura pipeline se evacueaza si se va initia procesarea celeilale ramuri de program. Totodata, valoarea bitului de predictie din BPB se inverseaza.

Dezavantajul schemei BPB cu un singur bit se manifesta îndeosebi în cazul buclelor de program. Bazat pe tehnica BPB în acest caz vom avea uzual 2 predictii false: una la intrarea în bucla (prima parcurgere) si alta la iesirea din bucla (ultima parcurgere a buclei). Astfel, considerând o bucla de program care va fi executata de

N ori, acuratetea predictiei va fi de (N - 2) * 100 / N %, iar saltul se face în proportie de (N - 1) * 100 / N %.

Pentru a elimina acest dezavantaj se utilizeaza 2 biti de predictie modificabili conform grafului de tranzitie de mai jos (numarator saturat). În acest caz acuratetea predictiei unei bucle care se face de (N - 1) ori va fi (N - 1) * 100 / N %.

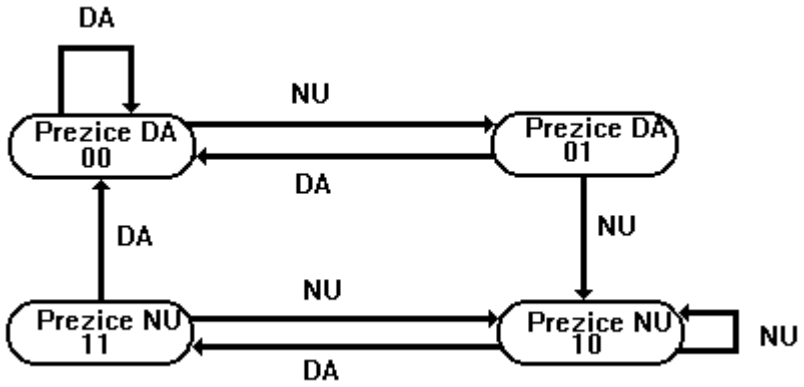

Figura 1. Automat de predictie de tip numarator saturat pe 2 biti

Prin urmare, în cazul în care se prezice ca branch-ul se va face, aducerea noii instructiuni se face de îndata ce continutul noului PC e cunoscut. În cazul unei predictii incorecte, se evacueaza structura pipeline si se ataca cealalta ramura a instructiunii de salt. Totodata, bitii de predictie se modifica în conformitate cu graful din figura numit si numarator saturat (vezi Fig.1).

O alta problema delicata consta în faptul ca desi predictia poate fi corecta, de multe ori adresa de salt (noul PC) nu este disponibila în timp util, adica la finele fazei de aducere IF. Timpul necesar calculului noului PC are un efect defavorabil asupra ratei de procesare. Solutia la aceasta problema este data de metoda de predictie numita **"branch target buffer"** (BTB). Un BTB este constituit dintr-un BPB care contine pe lânga bitii de predictie, noul PC de dupa instructiunea de salt conditionat si eventual alte informatii. De exemplu, un cuvânt din BTB ar putea contine si instructiunea tinta a saltului. Astfel ar creste performanta, nemaifiind necesar un ciclu de aducere a acestei instructiuni, dar în schimb ar creste costurile

de implementare. Diferenta esentiala între memoriile BPB si BTB consta în faptul ca prima este o memorie operativa iar a 2-a poate fi asociativa, ca în figura urmatoare.

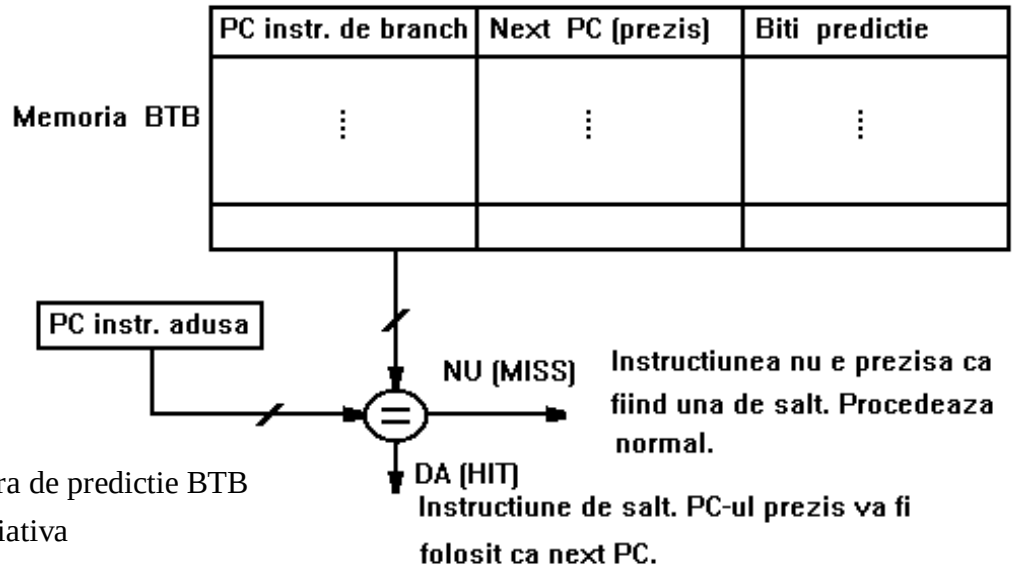

Figura 2. Structura de predictie BTB asociativa

La începutul fazei IF se declanseaza o cautare asociativa în BTB dupa continutul PC-ului în curs de aducere. În cazul în care se obtine hit se obtine în avans PC-ul aferent instructiunii urmatoare. Mai precis, considerând o structura pipeline pe 3 faze (IF, RD, EX) algoritmul de lucru cu BTB-ul este în principiu urmatorul [Hen96]:

IF) Se trimite PC-ul instructiunii ce urmeaza a fi adusa spre memorie si spre BTB. Daca PC-ul trimis corespunde cu un PC din BTB (hit) se trece în pasul RD2, altfel în pasul RD1.

RD1) Daca instructiunea adusa e o instructiune de branch, se trece în pasul EX1, altfel se continua procesarea normala.

RD2) Se trimite PC-ul prezis din BTB spre memoria de instructiuni. În cazul în care conditiile de salt sunt satisfacute, se trece în pasul EX 3, altfel în pasul EX2.

EX1) Se introduce PC-ul instructiunii de salt precum si PC-ul prezis în BTB. De obicei aceasta alocare se face în locatia cea mai de demult neaccesata (Least Recently Used- LRU).

EX2) Predictia s-a dovedit eronata. Trebuie reluata faza IF de pe cealalta ramura cu penalizarile de rigoare datorate evacuarii structurilor pipeline.

EX3) Predictia a fost corecta, însa numai daca si PC-ul predictionat este într-adevar corect, adica neschimbat.În acest caz, se continua executia normala.

În tabelul urmator (tabelul 1) sunt rezumate avantajele si dezavantajele tehnicii BTB, anterior descrise.

Tabelul 1. Penalizarea într-o structura de predictie tip BTB

Instr. în BTB ?	Predictie	Realitate	Cicli penalizare
Da	Da	Da	0(Ctt)
Da	Da	Nu	Ctn
Da	Nu	Nu	0
Da	Nu	Da	Cnt
Nu	-	Da	Ct
Nu	-	Nu	0

În baza celor prezentate anterior, rezulta ca numarul de cicli de penalizare CP este dat de urmatoarea relatie:

$$CP = P_{BTB}\,(P_{tn}*C_{tn} + P_{nt}*C_{nt}) + (1-P_{BTB})*P*C_{t} \qquad (2)$$

unde s-a notat:

P_{BTB} - probabilitatea ca instructiunea de salt sa se afle în BTB;

$\mathbf{P_{tn}}$ - probabilitatea ca saltul sa fie prezis ca se face si în realitate nu se va face;

$\mathbf{C_{tn}}$ - reprezinta ciclii de penalizare corespunzator situatiei.

$\mathbf{P_{nt}}$ - probabilitatea ca saltul sa fie prezis ca nu se face si în realitate se va face;

$\mathbf{C_{nt}}$ - reprezinta ciclii de penalizare în aceasta situatie.

P - probabilitatea ca respectiva instructiune de salt sa se faca. $\mathbf{C_t}$ – reprezinta ciclii de penalizare daca saltul se face si acesta nu este în BTB.

În acest caz, rata de procesare a instructiunilor ar fi data de relatia:

$$\mathbf{IR = 1/(1 + P_b * CP), [instr./tact]} \qquad \mathbf{(3)}$$

unde $\mathbf{P_b}$ = probabilitatea ca instructiunea curenta sa fie una de ramificatie.

Observatii:

1) BTB nu îmbunatateste performanta pentru o predictie corecta de tipul "saltul nu se face" (P_{nn}= 0), întrucât în acest caz structura se comporta în mod implict la fel ca si o structura fara BTB.

2) Un branch trebuie introdus în BTB, **cu prima ocazie când el se va face**. Un salt care ar fi prezis ca nu se va face nu trebuie introdus în BTB pentru ca nu are potentialul de a îmbunatati performanta. Din acest motiv, exista strategii care atunci când trebuie evacuat un branch din BTB îl evacueaza pe cel cu potentialul de performanta minim, care nu coincide neaparat cu cel mai putin folosit (vezi [Dub91, Per93]). Astfel în [Per93] se construieste câte o variabila MPP (Minimum Performance Potential), implementata în hardware, asociata fiecarui cuvânt din BTB. Evacuarea din BTB se face pe baza MPP-ului minim. Acesta se calculeaza ca un produs între probabilitatea ca un branch din BTB sa fie din nou accesat (LRU) si respectiv probabilitatea ca saltul sa se faca. Aceasta din urma se

obtine pe baza unei istorii a respectivului salt (taken / not taken). Minimizarea ambilor factori duce la minimizarea MPP-ului si deci la evacuarea respectivului branch din BTB, pe motiv ca potentialul sau de performanta este minim.

În literatura [Hen96, Dub91, Per93], bazat pe testari laborioase, se arata ca se obtin predictii corecte în cca. 88% din cazuri folosind un bit de predictie si respectiv în cca.93% din cazuri folosind 16 biti de predictie. Microprocesorul Intel Pentium avea un predictor de ramificatii bazat pe un BTB cu 256 de intrari.

O problema dificila este determinata de **instructiunile de tip RETURN** întrucât o aceeasi instructiune, poate avea adrese de revenire diferite, ceea ce va conduce în mod normal la predictii eronate, pe motivul modificarii adresei eronate în tabela de predictii. Desigur, problema se pune atât în cazul schemelor de tip BTB cât si a celor de tip corelat, ce vor fi prezentate în continuare. Solutia de principiu [Kae91], consta în implementarea în hardware a unor asa zise "stack - frame"- uri diferite. Acestea vor fi niste stive, care vor contine perechi CALL/ RETURN cu toate informatiile necesare asocierii lor corecte. Astfel, o instructiune CALL poate modifica dinamic în tabela de predictii adresa de revenire pentru instructiunea RETURN corespunzatoare, evitându-se astfel situatiile nedorite mai sus schitate.

Schemele de predictie anterior prezentate se bazau pe comportarea recenta a unei instructiuni de salt, de aici predictionându-se comportarea viitoare a acelei instructiuni de salt. Este posibila îmbunatatirea acuratetii predictiei daca aceasta se va baza pe comportarea recenta a altor instructiuni de salt, întrucât frecvent aceste instructiuni pot avea o comportare corelata în cadrul programului. Schemele bazate pe aceasta observatie se numesc **scheme de predictie corelata** si au fost introduse pentru prima data în 1992 în mod independent de catre *Yeh* si *Patt* si respectiv de Pan [Hen96, Yeh92, Pan92]. Sa consideram pentru o prima exemplificare a acestei idei o secventa de program C extrasa din benchmark-ul Eqntott din cadrul grupului de benchmark-uri SPECint '92:

```
(b1)   if (x = = 2)
           x = 0;
(b2)   if (y = = 2)
           y = 0;
(b3)   if (x ! = y) {
```

Se observa imediat ca în acest caz daca salturile b1 si b2 nu se vor face atunci saltul b3 se va face în mod sigur (x = y = 0). Asadar saltul b3 nu depinde de comportamentul sau anterior ci de comportamentul anterior al salturilor b1 si b2, fiind deci **corelat** cu acestea. Evident ca în acest caz schemele de predictie anterior prezentate nu vor da randament. Daca doua branch-uri sunt corelate, cunoscând comportarea primului se poate anticipa comportarea celui de al doilea, ca în exemplul de mai jos:

```
if (cond1)
....
if (cond1 AND cond2)
```

Se poate observa ca functia conditionala a celui de al doilea salt este dependenta de cea a primului. Astfel, daca prima ramificatie nu se face atunci se va sti sigur ca nici cea de a doua nu se va face. Daca însa prima ramificatie se va face atunci cea de a doua va depinde exclusiv de valoarea logica a conditiei "cond2". Asadar în mod cert aceste doua ramificatii sunt corelate, chiar daca comportarea celui de al doilea salt nu depinde exclusiv de comportarea primului. Sa consideram acum pentru analiza o secventa de program C simplificata împreuna cu secventa obtinuta în urma compilarii (s-a presupus ca variabila x este asignata registrului R1).

```
if      (x = = 0)          (b1)   BNEZ  R1, L1
          x = 1;                   ADD  R1, R0, #1
if      (x = = 1)           L1:   SUB  R3, R1, #1
                           (b2)   BNEZ  R3, L2
```

Se poate observa ca daca saltul conditionat b1 nu se va face, atunci nici b2 nu se va face, cele 2 salturi fiind deci corelate. Vom particulariza secventa anterioara, considerând iteratii succesive ale acesteia pe parcursul carora x variaza de exemplu între 0 si 5. Un BPB clasic, initializat pe predictie NU, având un singur bit de predictie, s-ar comporta ca în tabelul 2. Asadar o astfel de schema ar predictiona în acest caz, întotdeauna gresit!

Tabelul 2. Modul de predictionare al unui BPB clasic

x	Predictie b1	Actiune b1	Predictie noua b1	Predictie b2	Actiune b2	Predictie noua b2
5	NU	DA	DA	NU	DA	DA
0	DA	NU	NU	DA	NU	NU
5	NU	DA	DA	NU	DA	DA
0	DA	NU	NU	DA	NU	NU

Sa analizam acum comportarea unui predictor corelat având un singur bit de corelatie (se coreleaza deci doar cu instructiunea de salt anterior executata) si un singur bit de predictie. Acesta se mai numeste si predictor corelat de tip (1, 1). Acest predictor va avea 2 biti de predictie pentru fiecare instructiune de salt: primul bit predictioneaza daca instructiunea de salt actuala se va face sau nu, în cazul în care instructiunea anterior executata nu s-a facut iar al doilea analog, în cazul în care instructiunea de salt anterior executata s-a facut. Exista deci urmatoarele 4 posibilitati (tabelul 3).

Tabelul 3. Semnificatia bitilor de predictie pentru o schema corelata

Biti predictie	Predictie daca precedentul salt nu s-a facut	Predictie daca precedentul salt s-a facut
NU / NU	NU	NU
NU / DA	NU	DA
DA / NU	DA	NU
DA / DA	DA	DA

Ca si în cazul BPB-ului clasic cu un bit, în cazul unei predictii care se dovedeste a fi eronata bitul de predictie indicat se va complementa. Comportarea predictorului (1,1) pe secventa anterioara de program este prezentata în continuare (s-a considerat ca bitii de predictie asociati salturilor b1 si b2 sunt initializati pe NU / NU).

Tabelul 4. Modul de predictionare al unei scheme corelate

x	Predictie b1	Actiune b1	Predictie noua b1	Predictie b2	Actiune b2	Predictie noua b2
5	**NU**/NU	DA	DA/NU	NU/**NU**	DA	NU/DA
0	DA/**NU**	NU	DA/NU	**NU**/DA	NU	NU/DA
5	**DA**/NU	DA	DA/NU	NU/**DA**	DA	NU/DA
0	DA/**NU**	NU	DA/NU	**NU**/DA	NU	NU/DA

Dupa cum se observa în tabelul 4, singurele doua predictii incorecte sunt când x = 5 în prima iteratie. În rest, predictiile vor fi întotdeauna corecte, schema comportându-se deci foarte bine spre deosebire de schema BPB clasica. În cazul

general, un predictor corelat de tip (m,n) utilizeaza comportarea precedentelor m instructiuni de salt executate, alegând deci o anumita predictie de tip Da sau Nu din 2^m posibile iar n reprezinta numarul bitilor utilizati în predictia fiecarui salt. O comportare alternativa a unui salt este simplu de predictionat printr-o schema corelata, în schimb printr-o schema clasica este foarte dificil. Asadar schemele corelate sunt eficiente atunci când predictia depinde si de un anumit pattern al istoriei saltului de predictionat, corelatia fiind în acest caz particular cu istoria pe m biti chiar a acelui salt si nu cu istoria anterioarelor m salturi.

Un alt avantaj al acestor scheme este dat de simplitatea implementarii hardware, cu putin mai complexa decât cea a unui BPB clasic. Aceasta se bazeaza pe simpla observatie ca "istoria" celor mai recent executate m salturi din program, poate fi memorata într-un registru binar de deplasare pe m ranguri (registru de predictie). Asadar adresarea cuvântului de predictie format din n biti si situat într-o tabela de predictii, se poate face foarte simplu prin concatenarea c.m.p.s. biti ai PC-ului instructiunii de salt curente cu acest registru de deplasare în adresarea BPB-ului de predictie. Ca si în cazul BPB-ului clasic, un anumit cuvânt de predictie poate corespunde la mai multe salturi. Exista în implementare 2 nivele deci: un **registru de predictie** al carui continut concatenat cu PC- ul c.m.p.s. al instructiunii de salt pointeaza la un cuvânt din **tabela de predictii** (aceasta contine bitii de predictie, adresa destinatie, etc.). În [Yeh92], nu se face concatenarea PC - registru de predictie si în consecinta se obtin rezultate nesatisfacatoare datorita **interferentei diverselor salturi** la aceeasi locatie din tabela de predictii.

În [Pan92], o lucrare de referinta în acest plan, se analizeaza calitativ si cantitativ într-un mod foarte atent, rolul informatiei de corelatie, pe exemple concrete extrase din benchmark-urile SPECint '92. Se arata ca bazat pe predictoare de tip numaratoare saturate pe 2 biti, schemele corelate (5-8 biti de corelatie utilizati) ating acurateti ale predictiilor de pâna la 11% în plus fata de cele clasice.

De remarcat ca un BPB clasic reprezinta un predictor de tip (0,n), unde n este numarul bitilor de predictie utilizati. Numarul total de biti utilizati în implementarea unui predictor corelat de tip (m,n) este:

$$N = 2^m * n * NI \quad (4)$$

unde NI reprezinta numarul de intrari al BPB-ului utilizat.

Exista citate în literatura mai multe implementari de scheme de predictie a ramificatiilor, prima implementare comerciala a unei astfel de scheme facându-se în microprocesorul Intel Pentium Pro. Astfel, implementarea tipica a unui predictor corelat de tip **GAg** (Global History Register, Global Prediction History Table) este prezentata în figura 3. Tabela de predictii PHT (Prediction History Table) este adresata cu un index rezultat din concatenarea a **doua informatii ortogonale:** PClow (i biti), semnificând gradul de localizare al saltului, respectiv registrul de predictie (HR- History Register pe k biti), semnificând "contextul" în care se situeaza saltul în program. Ambele contributii s-au facut cu scopul eliminarii interferentelor branch-urilor în tabela de predictie. Adresarea PHT exclusiv cu HR ca în articolul [Yeh92], ducea la serioase interferente (mai multe salturi puteau accesa aceelasi automat de predictie din PHT), cu influente evident defavorabile asupra performantelor. Desigur, PHT poate avea diferite grade de asociativitate. Un cuvânt din aceasta tabela are un format similar cu cel al cuvântului dintr-un BTB. Se pare ca se poate evita concatenarea HR si PClow în adresarea PHT, cu rezultate foarte bune, printr-o functie de dispersie tip SAU EXCLUSIV între acestea, care sa adreseze tabela PHT. Aceasta are o influenta benefica asupra capacitatii tabelei PHT.

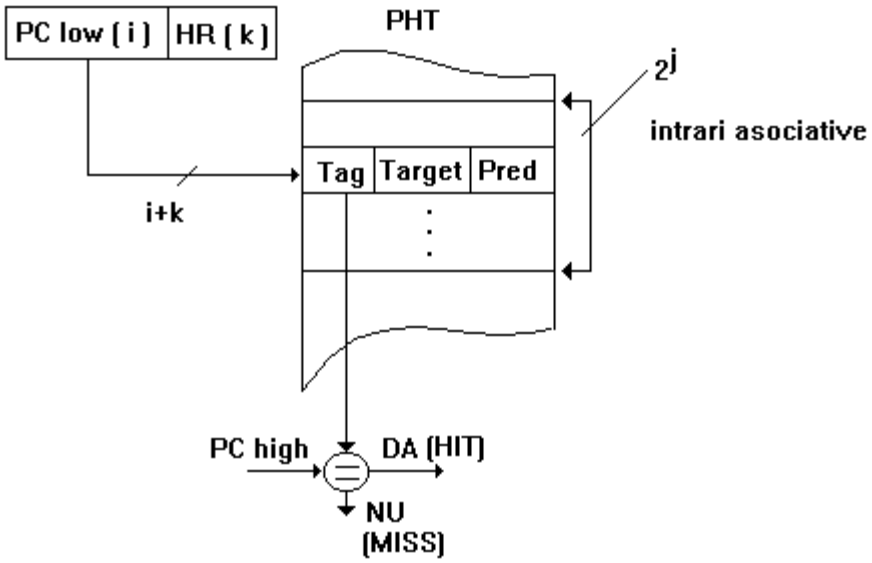

Figura 3. Structura de predictie de tip GAg

În scopul reducerii interferentelor diverselor salturi în tabela de predictii, în [Yeh92] se prezinta o schema numita **PAg-** Per Address History Table, Global PHT, a carei structura este oarecum asemanatoare cu cea a schemei GAg. Componenta HR*(k) a fost introdusa în [Vin97], având semnificatia componentei HR de la varianta GAg, adica un registru global care memoreaza comportarea ultimelor k salturi. Fara aceasta componenta, schema PAg si-ar pierde din capacitatea de adaptare la contextul programului în sensul în care schema GAg o face. *Yeh* si *Patt* renunta la informatia de corelatie globala (HRg) în trecerea de la schemele de tip GAg la cele de tip PAg, în favoarea exclusiva a informatiei de corelatie locala (HRl). În schimb, componenta HR din structura History Table, contine "istoria locala" (taken / not taken) a saltului curent, ce trebuie predictionat. Dupa cum se va arata mai departe, performanta schemei PAg este superioara celei obtinute printr-o schema de tip GAg, cu tributul de rigoare platit complexitatii hardware.

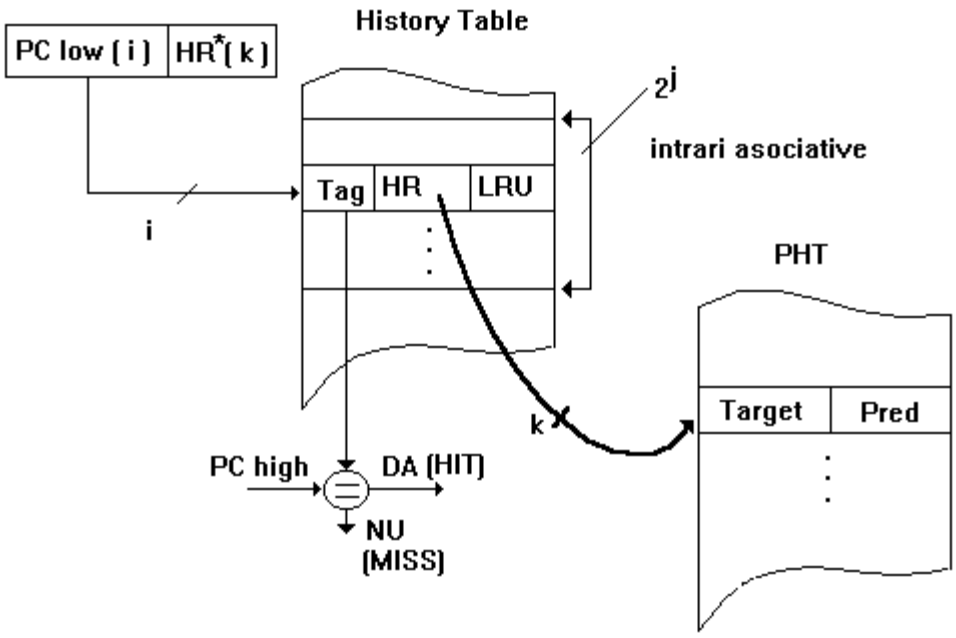

Figura 4. Structura de predictie de tip PAg

Asadar aceasta schema de tip PAg predictioneaza pe baza a **3 informatii ortogonale,** toate disponibile pe chiar timpul fazei IF: istoria HRg a anterioarelor salturi corelate (taken / not taken), istoria saltului curent HRl si PC-ul acestui salt. Daca adresarea tabelei PHT s-ar face în schema PAg cu HR concatenat cu PClow(i), atunci practic fiecare branch ar avea propria sa tabela PHT, rezultând deci o schema si mai complexa numita **PAp** (Per Address History Table, Per Address PHT) [Yeh92], a carei schema de principiu este prezentata mai jos (figura 5). Complexitatea acestei scheme o face practic neimplementabila în siliciu la ora actuala, fiind doar un model utilizat în cercetare.

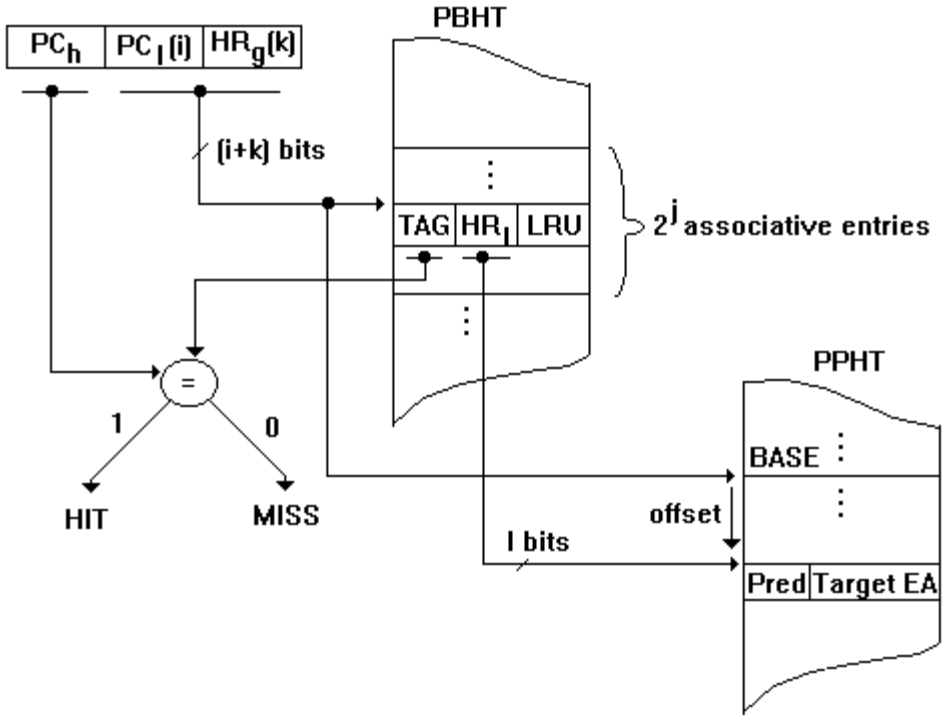

Figura 5. Structura de predictie de tip PAp

Desigur, este posibil ca o parte dintre branch-urile memorate în registrul HR, sa nu se afle în corelatie cu branch-ul curent, ceea ce implica o serie de dezavantaje. În astfel de cazuri pattern-urile din HR pot pointa în mod inutil la intrari diferite în tabela de predictii, fara beneficii asupra performantei predictiei, separându-se astfel situatii care nu trebuiesc separate. Mai mult, aceste situatii pot conduce la un timp de "umplere" a structurilor de predictie mai îndelungat, cu implicatii defavorabile asupra performantei [Eve96].

O problema dificila în predictia branch-urilor o constituie salturile codificate în moduri de adresare indirecte, a caror acuratete a predictiei este deosebit de scazuta prin schemele anterior prezentate (cca.50%). În [Cha97] se propune o structura de predictie numita "target cache" special dedicata salturilor indirecte. În acest caz predictia adresei de salt nu se mai face pe baza ultimei adrese tinta a saltului indirect ca în schemele de predictie clasice, ci pe baza alegerii uneia din ultimele adrese tinta ale respectivului salt, memorate în structura. Asadar, în acest

caz structura de predictie, memoreaza pe parcursul executiei programului pentru fiecare salt indirect ultimele N adrese tinta.

Predictia se va face deci în acest caz pe baza urmatoarelor informatii: PC-ul saltului, istoria acestuia, precum si ultimele N adrese tinta înregistrate. Structura de principiu a target cache-ului e prezentata în figura 6. O linie din acest cache contine ultimele N adrese tinta ale saltului împreuna cu tag-ul aferent.

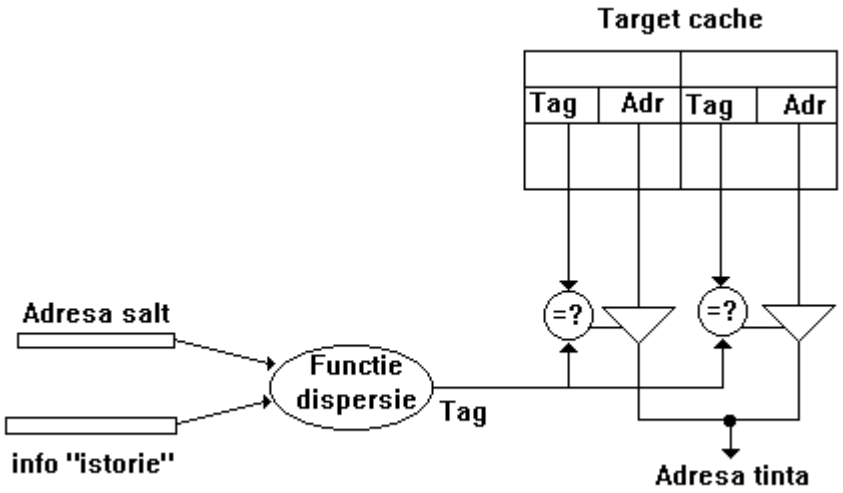

Figura 6. Predictia adresei în cazul salturilor indirecte

Informatia "istorie" provine din doua surse: istoria saltului indirect sau a anterioarelor salturi si respectiv ultimele N adrese tinta, înscrise în linia corespunzatoare din cache. Aceste doua surse de informatie binara sunt prelucrate prin intermediul unei functii de dispersie (SAU EXCLUSIV), rezultând indexul de adresare în cache si tag-ul aferent. Dupa ce adresa tinta a saltului devine efectiv cunoscuta, se va introduce în linia corespunzatoare din cache. Schema actioneaza "în mod disperat", mizând pe faptul ca la acelasi context de aparitie a unui salt indirect se va asocia o aceeasi adresa tinta. Si în opinia mea, aceasta abordare principiala pare singura posibila în cazul acestor salturi greu predictibile. Prin astfel de scheme, masurat pe benchmark-urile SPECint '95 acuratetea predictiei salturilor

indirecte creste si ca urmare, câstigul global asupra timpului de executie este de cca 4.3% - 9% [Cha97].

3. Desfasurarea lucrarii

3.1. Prezentarea simulatorului

Partea practica a lucrarii consta în simulare efectuata pe benchmark-urile Stanford. Pentru simulare, sunt folosite 8 fisiere trace identice ca nume dar cu extensia (*.tra). Aceste fisiere sunt o prelucrare a programelor scrise în mnemonica de asamblare (*.ins) si a trace-urilor originale (*.trc), cu scopul de a evidentia toate salturile (inclusiv cele care nu se fac).

Întregul fisier trace (*.trc) este o înlantuire de triplete <***TipInstr AdrCrt AdrDest***>, unde ***TipInstr*** poate lua una din valorile ‘B’ – branch, ‘L’ – load, ‘S’ – store; ***AdrCrt*** reprezinta valoarea registrului PC – adresa instructiunii curente, iar ***AdrDest*** reprezinta adresa de memorie a datei accesate – în cazul instructiunilor cu referire la memorie (‘L’ / ‘S’) sau adresa destinatie a saltului – în cazul instructiunilor de salt si ramificatie (‘B’).

Desi simularea poate ignora instructiunile Load / Store din trace, deoarece în acest caz suntem interesati numai de comportarea branch-urilor, exista totusi o deficienta a acestor trace-uri: nu evidentiaza salturile care nu se fac. Din acest motiv s-au generat noile trace-uri (fisierele *.tra). Acestea contin doar branch-urile (atât cele care se fac cât si cele care nu se fac) si exclude instructiunile Load / Store. Forma acestor fisiere este urmatoarea:

```
BT      12      30
BS      32      98
BM      100     33
NT      36      37
```

BRA 2 100

MOV PC, RA (RETURN)

Reprezentarea salturilor se face tot sub forma unor triplete <***TipBr AdrCrt AdrDest***>, unde ***TipBr*** se prezinta sub forma unei codificari pe doua caractere, primul dintre ele indica daca saltul se face (‘B’ – saltul se face, ‘N’ – saltul nu se face), iar al doilea caracter indica tipul saltului: ‘T’ sau ‘F’ – salturi conditionate, ‘S’ – apeluri de tip Call, ‘M’ – apeluri de tip Return, ‘R’ – salturi neconditionate. Alegerea acestei codificari a fost inspirata de mnemonicile întâlnite în sursa în limbaj de asamblare (BT, BF – salt conditionat, BSR – instructiune de tip Call, BRA – salt neconditionat si MOV PC, RA – instructiune de tip Return).

Automatul de predictie este descris printr-un sir de caractere cu un format mai special, ce prezinta atât numarul de stari, tranzitiile între stari cât si predictia aferenta fiecarei stari. Pentru o mai buna întelegere a functionarii automatului exemplificam pe doua situatii diferite:

1. automatul **ABAB:2** - pe un bit
2. automatul **BCBAADCD:12** - pe 2 biti

Tabelul care descrie functionarea primului automat este urmatorul:

Stare Curenta	Stare urmatoare		Predictie
	Pentru intrare = 0	Pentru intrare = 1	
A	A	B	0
B	A	B	1

Prima parte a sirului pâna la caracterul ‘:’ reprezinta tranzitiile pentru starea ‘A’ cu intrare 0, apoi cu intrare 1, apoi tranzitiile din ‘B’ pentru aceleasi intrari. Numarul din a doua parte este “vazut” în binar sub forma “0010” si reprezinta

iesirile asociate fiecarei stari în parte (starii ‘A’ îi este asociat cel mai putin semnificativ bit, în acest caz bitul ‘0’, urmatorul bit lui ‘B’, bitul ‘1’, etc).

Tabelul care descrie functionarea celui de-al doilea automat arata astfel:

Stare Curenta	Stare urmatoare		Predictie
	Pentru intrare = 0	Pentru intrare = 1	
A	B	C	0
B	B	A	0
C	A	D	1
D	C	D	1

Cache-ul de predictie simulat este cel din figura 5. Acesta e alcatuit din doua tabele:

- prima cu 2^{i+k+j} intrari asociative care contine istoria locala a saltului, tag-ul de verificat si un câmp LRU asociat fiecarei intrari. Adresarea în aceasta tabela se face cu cei mai putini semnificativi biti ai PC-ului (i) concatenati cu istoria globala a saltului (k).
- a doua tabela are 2^{i+k} intrari. La ofset-ul l(istoria locala anterior citita) în aceasta tabela se afla bitul de predictie si adresa noii instructiuni tinta.

Parametrii de intrare ai simulatorului sunt în ordine:

- *Nume trace (.tra)* – oricare din cele 8 trace-uri cu extensia *.tra*. Fie fsort.tra.
- *Tip automat* – sir de caractere ce descrie automatul. Fie BCBAADCD:12
- *Numar seturi asociative* – se alege o putere de a lui 2. Fie 4 seturi $\Rightarrow$ i=2
- *Dimensiune set asociativ (nr. intrari)* – Presupunem 8 intrari $\Rightarrow$ j=3
- *Dimensiune registru istorie locala* – în biti. Presupunem l=2

- *Dimensiune registru istorie globala* – în biti. Presupunem k=3

Simulatorul genereaza urmatoarele rezultate:

- Numarul de branch-uri predictionate corect.
- Numarul de branch-uri predictionate gresit.
- Numarul de branch-uri negasite în prima tabela (salturi care au generat MISS în procesul de cautare).
- Numarul de adrese predictionate incorect (salturi care sunt în prima tabela, predictionate corect dar adresa tinta predictionata nu este cea corecta – fiind modificata între timp).

3.2. Probleme propuse spre rezolvare

Cu ajutorul simulatorului *pap.exe* generati:

1. Rata de predictie corecta pe fiecare benchmark în parte într-o configuratie data.

2. Stabiliti influenta parametrilor i, j, k si l (câte un singur parametru trebuie variat succesiv) asupra

ratei de predictie corecte a branch-urilor si stabiliti valoarea optima a acestora.

3. Stabiliti influenta corelata a celor trei parametri i, j si k asupra ratei de predictie.

4. Cuantificati câstigul introdus printr-o implementare ideala, în care nu ar exista adresa tinta modificata în tabele.

Bibliografie

[1] [Cha97] **Chang P.Y., Hao E., Patt Y.N**. - *Target Prediction for Indirect Jumps,* ISCA '97 (http://www.eecs.umich.edu/HPS)

[2][Eve96] **Evers M., Chang P.Y., Patt Y.N**. - *Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches,* ISCA '96

[3][Koh95] **Kohonen T., et al.** - *Learning Vector Quantization (LVQ). Program Package Ver. 3.1*, Helsinki University of Technology, SF-02150 Espoo, Finland, 1995

[4][Per93] **Perleberg C., Smith A. J**. - *Branch Target Buffer Design and Optimisation,* IEEE Trans. Computers, No. 4, 1993.

[5][Vin97] **Vintan L., Steven G.B. -** *Memory Hierarchy Limitations in Multiple Instruction- Issue Processor Design*, Proceedings of Euromicro '97 Conference (Short Papers), Budapest 1-4 Sept. 1997, IEEE Computer Society Press, California, USA, 1997

[6][Yeh92] **Yeh T., Patt Y.** - *Alternative Implementations of Two Level Adaptive Branch Prediction,* 19 th Ann. Int.'L Symp. Computer Architecture, 1992

[7][Hen96] **Hennessy J., Patterson D.** - *Computer Architecture*: *A quantitative Approach*, Morgan Kaufmann Publishers, Second Edition, 1996.

[8][Dub91] **Dubey P., Flynn M.** - *Branch Strategies*: *Modeling and Optimization*, IEEE Transaction on Computer, No 10, 1991.

[9][Mud96] **Mudge T.N., et al.** - *Limits of Branch prediction*, Technical Report, Electrical Engineering and Computer Science Department, The University of Michigan, Ann Arbor, Michigan, USA, 1996

[10][Pan92] **Pan S.T., So K., Rahmeh J.T.** - *Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation*, ASPLOS V Conference, Boston, October, 1992

[11][Kae91] **Kaeli D., Emme P. –** *Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns*, 18-th Int.'L Conf. On Computer Architecture, Toronto, May 1991.