Arhitectura Calculatoarelor

Curs 11: Pipeline avansat: Execuție speculativă

E-mail: florin.oniga@cs.utcluj.ro

Web: http://users.utcluj.ro/~onigaf, secțiunea Teaching/AC

Pipeline avansat: Execuție speculativă

- Planificarea dinamică este limitată dacă execuția nu merge mai departe în cazul ramificărilor condiționate (se așteaptă până se evaluează condiția)
- Două concepte de bază
  - **Predicţie:**
    - Se referă la IF, aducerea instrucțiunilor de la adresa prezisă de salt
    - Predicția mărește numărul de lansări, respectiv instrucțiuni executate
    - Capacitatea de execuţie trebuie mărită pentru a tine pas cu viteza lansării instrucţiunilor
  - **Speculaţie:**
    - Se referă la execuţia instrucţiunilor aduse pe baza predicţiei, înainte de a şti dacă predicţia a fost corectă
    - Extinde fereastra instrucțiunilor care pot fi testate pentru ILP
    - Se bazează pe predicţia ramificărilor
    - Ideea: separarea execuţiei de validarea terminării (commitment) unei instrucţiuni
    - Execuţie cu rezultate temporare, până când speculația (predicția) este validată / invalidată

Pipeline avansat: Execuție speculativă

Speculația este bazată pe hardware: 3 componente

1. Planificare dinamică - mod de lucru de tip flux de date „out-of-order”, operaţiile se execută în momentul disponibilităţii operanzilor (nu în ordinea scrisă în program), aspect discutat deja în cursul anterior

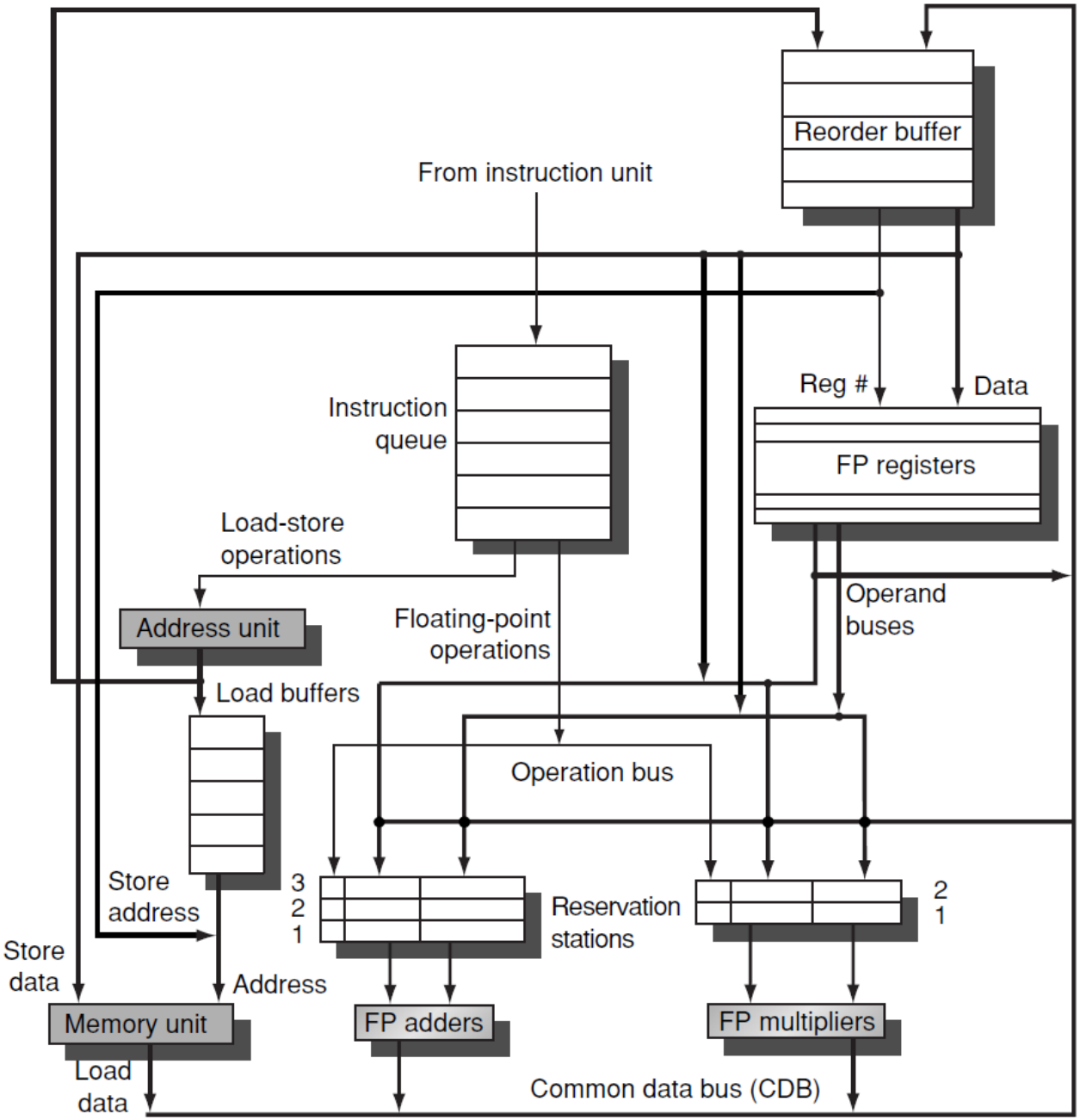
2. Predicţia dinamică a ramificărilor – ajută la aducerea instrucțiunilor pentru execuție speculativă, prin predicția salturilor condiționate

3. Speculaţia - executarea instrucţiunilor înainte de rezolvarea dependențelor de ramificare + abilitatea de a anula efectele speculațiilor incorecte

Pipeline avansat: Tomasulo Speculativ

- PowerPC 603/604/G3/G4, MIPS R10000/R12000, Intel Pentium II/III/4, Alpha 21264, AMD K5/K6/Athlon, …) și procesoarele actuale implementează execuţia speculativă bazată pe algoritmul Tomasulo sau variații/extensii.
- Pas suplimentar, validarea rezultatului: instruction commit/terminare:
  - Când o instrucţiune nu mai este speculativă, i se permite actualizarea blocului de registre sau a memoriei, în ordinea scrisă în program
- Necesită un set adiţional de bufere pentru:
  - Păstrarea rezultatelor instrucţiunilor care au terminat execuţia dar încă n-au fost validate
  - Reordonarea instrucțiunilor terminate „out-of-order”
- Acest set adițional este buferul de reordonare „reorder buffer (ROB)”, fiind folosit și pentru transferul rezultatelor speculative între instrucțiuni speculative

Pipeline avansat: Tomasulo Speculativ



Algoritmul Tomasulo extins pentru execuţie speculativă

Noutăți în schemă [1]:

- ROB (Reorder Buffer)
- Store buffer este eliminat, integrat în ROB

Acest mecanism poate fi extins pentru lansări multiple (multiple issues - **superscalar)** prin lărgirea CDB (common data bus) pentru a permite terminări multiple per ciclu de ceas.

Pipeline avansat: Tomasulo Speculativ

Execuția Load/Store

- Load/Store necesită 2 paşi de execuţie ca și în Tomasulo clasic
- În primul pas se calculează adresa efectivă, (când registrul de bază este valid), apoi adresa efectivă se plasează în buferul load sau store.
- Load-urile din „load buffer” se execută în momentul în care memoria este disponibilă.
- Store-urile în „store buffer” (=ROB) aşteaptă valoarea de memorat înainte de accesarea memoriei
- Al doilea pas de execuţie pentru „Store-uri” este realizat de faza de terminare validată a instrucțiunii, „commit”.
- ROB îndeplineşte rolul buferelor de „store”.

Tomasulo Speculativ: Bufer de reordonare ROB

Conținutul ROB pentru o locație (selecție, în realitate există și alte câmpuri relevante):

- Tipul instrucțiunii
  - Ramificare (nu are rezultat în Destination),
  - Store (destinația este o adresa de memorie), sau
  - Operații cu Registre (ALU sau Load au registre de destinaţie)
- Destination
  - Index de registru (pentru operaţii tip Load sau UAL) sau
  - Adresa de memorie (pentru Store) unde se va scrie operandul
- Value
  - Valoarea rezultatului instrucțiunii până la validarea terminării
- Busy
  - Indică dacă o instrucțiune a terminat execuția și valoarea este calculată
- Câmpuri suplimentare pentru tratarea speculaţiei și a excepţiilor – câmpul speculativ
  - Câmpul speculativ poate avea trei valori: speculative, confirmed și not confirmed
  - Dacă o instrucțiune de ramificare se confirmă, câmpul speculativ se marchează "confirmat" („confirmed").
  - Dacă o instrucțiune de ramificare nu se confirmă, câmpul speculativ se marchează „neconfirmat" ("not confirmed")

Tomasulo Speculativ: Bufer de reordonare ROB

- În algoritmul Tomasulo fără speculații, instrucțiunile înregistrează rezultatele lor în Blocul de Registre (sau Memorie pentru store)
- În varianta speculativă, Blocul de Registre nu este actualizat până la faza de validare a terminării instrucțiunii (commit - se confirmă că instrucțiunea trebuia să fie executată)
- ROB livrează operanzi speculativi în intervalul dintre terminarea execuției instrucțiunii (completion) și validarea terminării (commit)
  - ROB extinde numărul registrelor la fel ca Staţiile de Rezervare (SR)
- Conceptul Buferului de Reordonare:
  - Stochează instrucțiunile în ordine tip FIFO, în ordinea lansării lor
  - Când se termină execuția instrucțiunii, rezultatul se plasează în ROB
    - Livrează operanzi la alte instrucțiuni între terminarea și validarea terminării unei instrucțiuni $\Rightarrow$ mai mulți registre, (ca și SR)
    - Rezultatele sunt etichetate cu numărul lor din ROB (tag), (nu cu numere de SR)
  - La validarea terminării instrucțiunii $\Rightarrow$ rezultatele instrucțiunii din vârful ROB se plasează în registre accesibile programatorului
  - Ca urmare, instrucțiunile executate speculativ după o ramificare prezisă incorect sau după o excepție, se anulează fără probleme

Algoritmul Tomasulo Speculativ: 4 paşi

1. Issue – Lansare (distribuire)

- Se citeşte instrucţiunea următoare din coada de instrucţiuni:
  - Instrucțiunea se lansează (Issue) dacă există o SR liberă (la UF corespunzătoare) și o locaţie goală în ROB, altfel
  - Dacă SR sau ROB nu sunt disponibile, lansarea instrucţiunilor este oprită (stalled) pana când ambele unități vor avea locaţii disponibile.
- Se aloca poziţii în SR și ROB pentru instrucțiunea citită,
- Se redenumesc registrele sursă și destinație (! Poate presupune existența unui set mai larg de registre, în plus față de SR/ROB, invizibile programatorului)
- Instrucţiunea decodificată și cu registre redenumite se transferă la SR și ROB
- Daca operanzii sunt disponibili în Blocul de Registre sau în ROB, se trimit la SR
- Se actualizează pozițiile de control pentru a indica buferele în folosinţă.
- Numărul locaţiei ROB, alocat pentru rezultat, se trimite la SR, astfel acest număr poate fi folosit ca **eticheta (tag)** însoţitoare când rezultatul se plasează pe CDB

Algoritmul Tomasulo Speculativ: 4 paşi

2. Execuție

- **Condiție:**
  - Dacă ambii operanzi sunt disponibili în SR, operaţia se execută, altfel
  - Se monitorizează CDB pentru aducerea operanzilor în SR
- Acest pas tratează hazardurile RAW
- În acest etaj, instrucţiunile pot fi executate în mai multe perioade de ceas (ex. „Load” – 2 paşi).
- Pentru „Store” numai registrul de bază trebuie să fie valabil, deoarece se calculează doar adresa efectivă.

Algoritmul Tomasulo Speculativ: 4 paşi

3. Write result (terminarea execuției / WB)

- **Condiţie:** La o anumită UF, o instrucțiune termină execuţia
- Când rezultatul este disponibil, se difuzează prin CDB (împreuna cu eticheta ROB primită la lansare) și este preluat în ROB și SR care aşteaptă după acest rezultat
- Se marchează SR ca disponibilă (nu mai este alocată unei instrucțiuni).
- Acțiuni speciale pentru „Store”:
  - Dacă valoarea de stocat este disponibilă, se înregistrează în câmpul „Value” al poziţiei ROB alocată pentru „Store”.
  - Daca valoarea de stocat nu este încă disponibilă, se urmărește CDB și în momentul apariţiei etichetei respective se actualizează câmpul „Value” al poziției ROB alocată pentru „Store”.

Algoritmul Tomasulo Speculativ: 4 paşi

4. **Commit (terminare)** – actualizarea registrelor cu date valide

- **Conditie:** ROB nu este gol și instrucţiunea cap de listă în ROB a fost executată
  - Commit pentru instrucțiunile valide din vârful ROB
  - Instrucţiunile se termină în ordinea scrierii (in-order)
- Există trei cazuri de terminare în funcție de tipul instrucţiunii din varful listei: Ramificare (branch) cu predicţie eronată / „Store” / Alta instrucţiune (terminare normală).
- Terminarea normală are loc când o instrucţiune ajunge în vârful ROB și rezultatul ei este în ROB. În acest caz Blocul de Registre este actualizat cu rezultatul obţinut
- Terminarea „Store” este similară, memoria este actualizată, nu Blocul de Registre.
- Dacă o instrucţiune de ramificare cu predicţie eronată ajunge în varful ROB, înseamnă ca speculația a fost incorectă.
  - ROB este „curatat” (flushed) și execuţia este repornită de la următoarea instrucţiune de după ramificare
- După terminarea unei instrucțiuni (actualizare Memorie sau Bloc de Registre), poziţia din ROB se eliberează

Tomasulo Speculativ: Decuplarea Memoriei

- Într-un procesor speculativ, Store se tratează în mod diferit fata de algoritmul Tomasulo.
- In algoritmul Tomasulo, Store poate scrie in memorie când ajunge în etapa Write Results (se presupune ca adresa efectivă a fost calculată și valoarea de scris este disponibilă)
- Într-un procesor speculativ, Store actualizează memoria numai când ajunge în vârful ROB.
- Această abordare asigură că memoria se actualizează numai când Store nu mai este speculativă.
- Este necesară evitarea hazardurilor prin memorie.
- Hazardurile WAW și WAR prin memorie se elimina, fiindcă actualizarea memoriei are loc în ordinea lansării, când Store ajunge în vârful ROB. Astfel, nu există Load sau Store precedent, neexecutat
- Hazardurile RAW prin memorie se rezolvă prin 2 restricţii:
  - Nu se permite pasul doi al Load (acces memorie) daca orice Store in ROB are adresa destinaţie egală cu valoarea adresei sursa din câmpul de adresa al Load-ului, si
  - Se menţine ordinea lansării (în ordine) pentru calculul adresei Load relativ la Store-uri precedente

Tomasulo Speculativ: Limitarea speculației

- Un avantaj însemnat al speculaţiei este abilitatea de a depăşi evenimentele care altfel ar produce aşteptări in pipeline, de ex. accesare nereușită la cache (cache miss).
- Acest avantaj potenţial aduce si dezavantaje posibile: procesorul poate specula ca a avut loc o excepţie costisitoare si începe prelucrarea ei, dar de fapt a fost o speculaţie incorecta
- Pentru a obţine mai multe avantaje decât dezavantaje, majoritatea pipeline-urilor cu speculaţie permit tratarea speculativă numai a excepţiilor de cost redus (de ex. acces neconfirmat la cache nivel 1)
- Daca are loc un eveniment excepţional scump (ex. acces neconfirmat cache nivel 2 sau TLB) procesorul va întârzia tratarea evenimentului pana când instrucțiunea cauzatoare nu mai este speculativa
- Aceasta abordare poate degrada performanța în unele cazuri, dar evita pierderi semnificative în altele (evenimente frecvente corelate cu predicţia mediocră a ramificărilor)

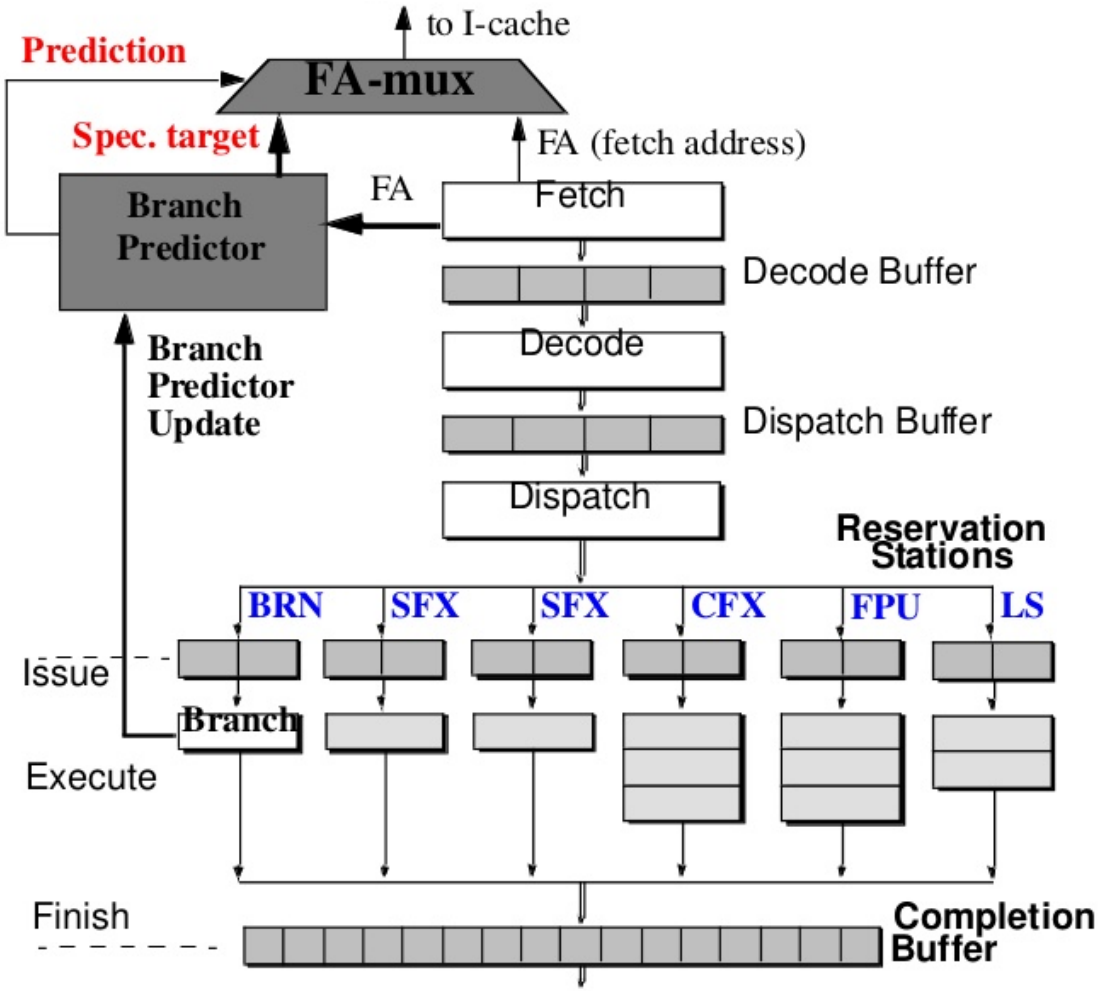
Tomasulo Speculativ: Limitarea speculației

Speculaţie prin mai multe ramificări

- 3 situații în care specularea mai multor ramificări succesive este avantajoasă:
  - Frecvența foarte mare de ramificări
  - Aglomerarea semnificativă a ramificărilor
  - Întârzieri mari în Unitățile Funcționale
- In primele două cazuri, îmbunătățirea se obţine prin specularea rezultatelor mai multor ramificări
- De asemenea, pentru evitarea așteptărilor datorate duratelor mari de execuție în UF-uri, specularea ramificărilor multiple poate fi avantajoasă
- Specularea ramificărilor multiple complică recuperarea efectelor predicțiilor eronate
- Cantitatea paralelismului disponibil depinde de programul executat
- Tehnicile de implementare nu pot obține paralelism mai mare decât cel existent în aplicație

Predicţia Ramificărilor (Branch Prediction)

- Predictorul de ramificări determină **probabilistic** daca un salt condiționat va fi efectuat sau nu



Arhitectură generală de pipeline cu Predictor de Ramificări

Predicţia Ramificărilor

Predicția statică a ramificărilor (Static Branch Prediction) – pe durata compilării

- Predicția statică: compilatorul decide direcția (ex. prin setarea unor biți din codul instrucțiunii)
- Direcția de ramificare – semnul deplasamentului; regula de predicție:
  - Salt înapoi – probabil se va executa
  - Salt înainte – probabil nu se va executa.
  - Algoritm: Backward Taken Forward Not taken (BTFN); predicţie corecta cca.65% pentru SPECint89.

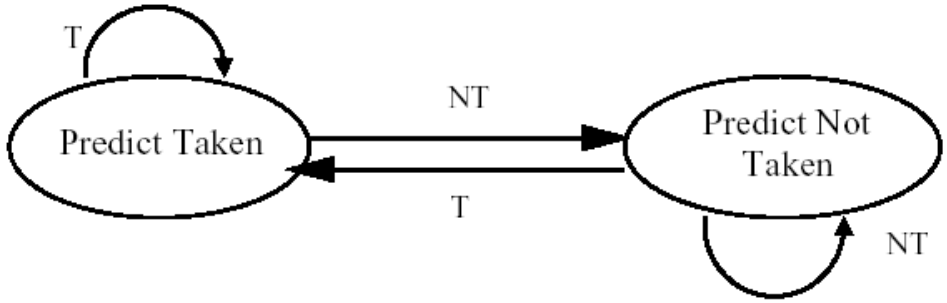
Predicția dinamică a ramificărilor – în timpul execuției

- Hardware-ul decide direcția, pe baza istoriei ramificării
- Algoritmii dinamici iau în considerare informații culese în timpul rulării programului.
- Procesorul învață din greșelile comise și își schimbă predicțiile în concordanță cu comportamentul fiecărei instrucțiuni de ramificare.
- Un algoritm dinamic de predicție înregistrează comportamentul ramificărilor anterioare și pe baza istoriei își îmbunătățește comportarea în timp.

Predicţia Ramificărilor

Predictor cu 1 bit

- O schema simplă, menține un singur bit de istorie pentru fiecare ramificare.
- Algoritm de predicție: pentru o anumită instrucțiune de salt condiționat ramificarea curentă va avea aceeași direcție ca și ceea precedentă
- Aceasta metodă poate atinge precizie de predicție de cca. 80%.

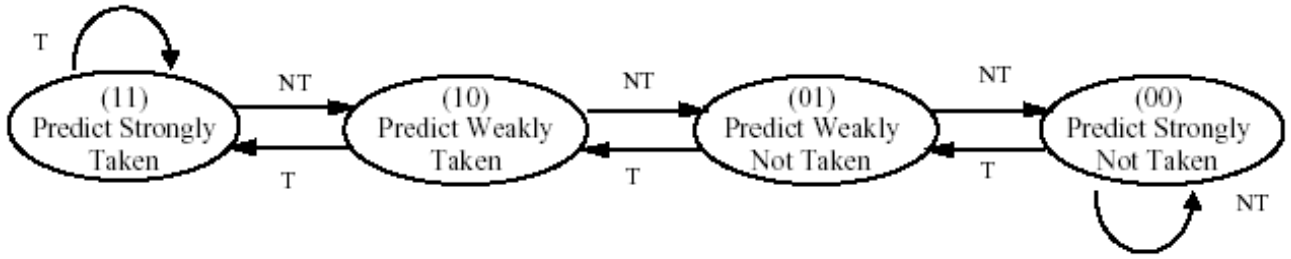


FSM pentru Predictor de 1 bit

- Predictorul de 1 bit prezice corect direcția la capătul unei bucle, cât timp bucla nu se termină
- În bucle imbricate, predictorul de 1 bit cauzează 2 predicții eronate pentru bucla internă
  - Una la terminarea buclei, la ieșirea din bucla („înapoi” în loc de „înainte”)
  - Una la reintrare in bucla („înainte” în loc de „inapoi”).
  - **Predictorul cu 2 biţi –** elimina eroarea dublă de predicție în bucle imbricate.

Predicţia Ramificărilor

Predictor cu 2 biţi

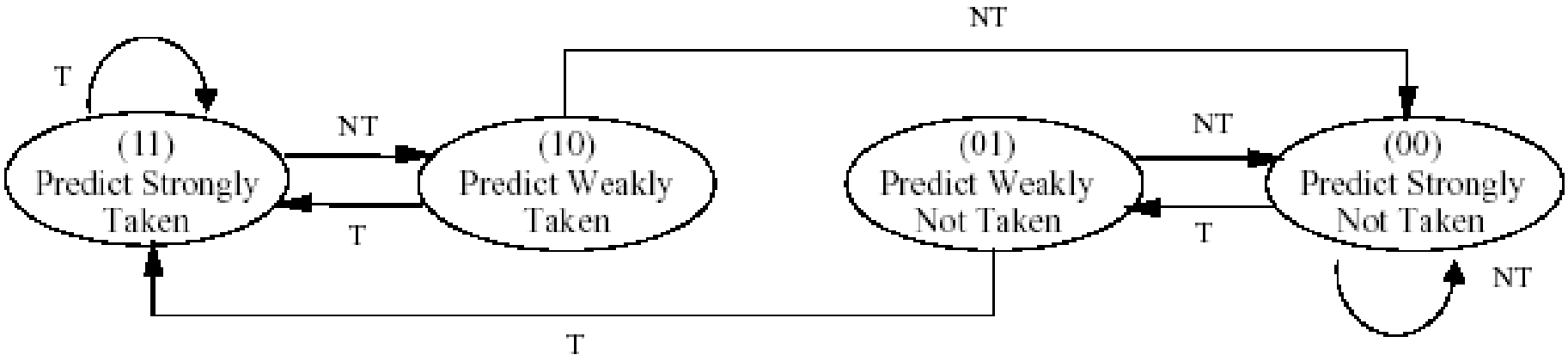


Predictor cu 2 biţi – Numărător cu saturare; Diagrama de stări

- Stările 01, 10: o predicție eronată schimbă direcția;
- Stările 00, 11: două predicții eronate schimbă direcția
- Numărătorul este incrementat/decrementat când ramificarea se execută/nu se execută
- Bitul cel mai semnificativ este folosit pentru predicția comportamentului
- Numărător saturat – nu se decrementează după 00 si nu se incrementează după 11
- Folosind numărătorul de 2 biți, predictorul poate tolera o ramificare într-o direcție neobișnuita o singură dată și să păstreze direcția corectă pentru cazul majoritar
- Acest efect de histerezis poate ridica precizia predicției la 85% (SPECint92), în funcție de mărimea și tipul tabelei de istorie folosită
- Studii efectuate arata ca un predictor cu 2 biţi funcţionează aproape la fel de bine ca un predictor cu n biţi, (n>2); **urmează în curs predictori bazați pe predictorul cu 2 biți**

Predicţia Ramificărilor

Predictori cu 2 biti – Schema UltraSPARC



FSM cu histereză

- Histereză mai pronunțată, numai 2 predicții eronate schimba direcția
- Nu există conexiuni directe între stările WT și WNT

Predicția Ramificărilor

Implementare în procesoare reale

Q: Câți predictori individuali sunt necesari într-un procesor?...Instrucțiunile de salt pot fi oriunde în memorie…

Variante extreme:

a. Un singur predictor => dacă în program există mai multe instrucțiuni de salt succesive, predictorul nu va adapta la niciuna (își va schimba permanent comportamentul fără să „învețe”)
b. Câte un predictor la fiecare element de memorie (unde se pot afla instrucțiuni!) – fiecare predictor va învăța comportamentul instrucțiunii de salt asociate, dar costul de implementare este prohibitiv

Soluția

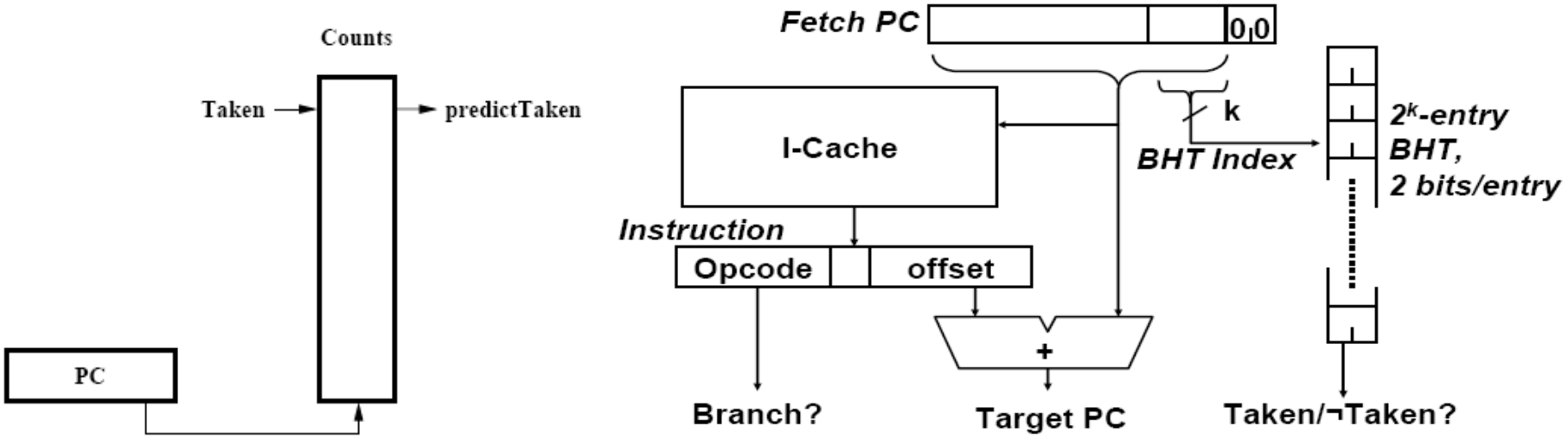
O tabelă cu un număr limitat de predictori si un mecanism de asociere a intrucțiunilor de salt cu predictorii din tabel, minimizând riscul ca instrucțiuni de salt succesive să fie asociate pe același predictor.

Mecanisme de asociere instrucțiune ⇔ predictor din tabel:

- Ultimii k biți din adresa instrucțiunii (exemplu pe slide următor) => la același predictor se vor asocia doar instrucțiunile de salt care au aceeași valoare la ultimii k biți de adresă
- Folosind istoricul ultimelor n salturi executate (se adaptează la tiparul de execuție din program) în combinație cu ultimii biți de adresă ai instrucțiunii sau alte combinații de accesare

Predicţia Ramificărilor

Tabelă de predictori



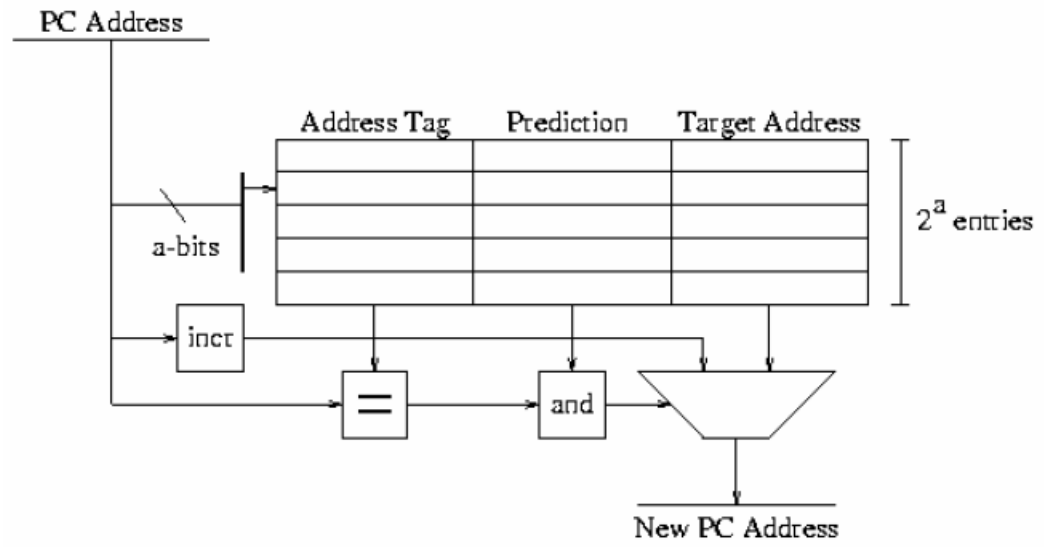
Predictor Bimodal (cu numărător saturat)

- Counts - Tabel (memorie) cu stările numărătorului de 2 biţi, indexat cu biţii de adresă (ordin mic) al PC ⇔ BHT – Brach History Table
- Pentru fiecare ramificare (conform PC), numărătorul respectiv este incrementat / decrementat, conform direcţiei validate a ramificării
- Predicţia direcţiei: pe baza stării curente (neactualizată) a numărătorului respectiv

Predicţia Ramificărilor

Branch-Target Buffer (BTB) ; Bufer pentru ținta ramificării

- O memorie cache care păstrează **adresele țintă** pentru ramificările executate: *branch-target buffer* sau *branch-target cache*.



Branch-target buffer, predicţie memorata

- BTB se foloseşte in etajul IF pentru determinarea noului PC.
- Daca conţinutul curent al PC corespunde cu o adresa (Address Tag) in BTB, valoarea noua a PC se selectează conform predicţiei; (altfel PC+4)
- Următoarea instrucţiune se citeşte conform adresei ţinta din linia BTB selectata prin indexare.
- O varianta BTB poate memora instrucţiunea ţinta in loc de (sau pe langa) adresa ţintei; se reduce un apel la memorie

REFERENCES

1. D.A. Patterson, J.L. Hennessy, Computer Organization and Design, *Morgan Kaufmann Publishers Elsevier*, ultimele editii