PARALELIZAM

1. Stride based prediktor. U kojim situacijama može imati veliku uspešnost u predikciji?

   - State={Init, Transient, Steady}

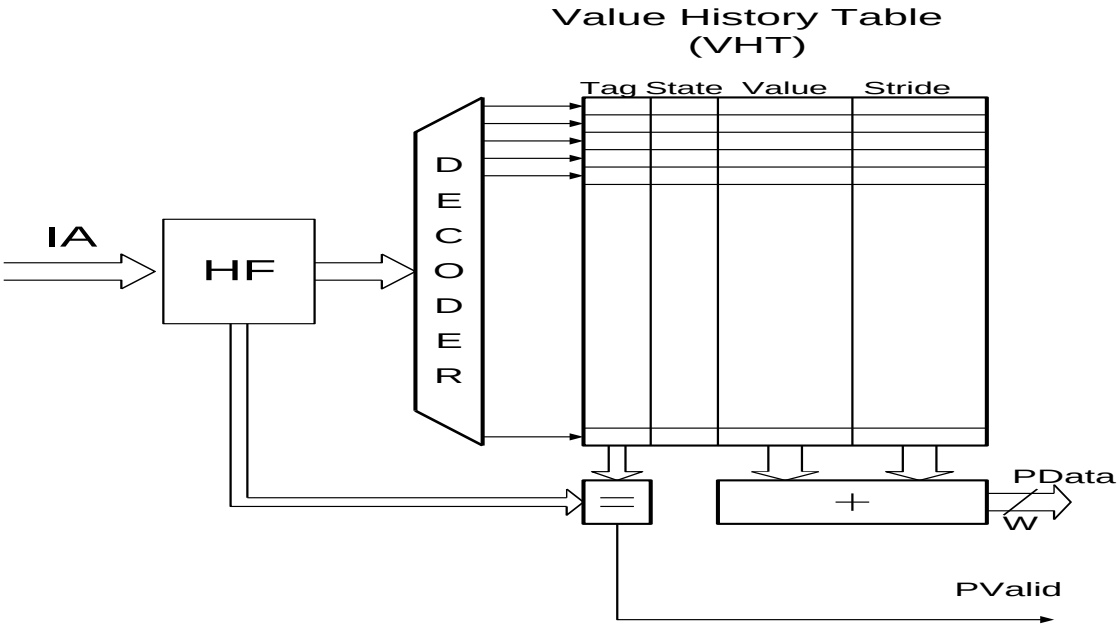
     Tek u steady stanju radimo predikciju

     U prvom izvršavanju postavimo D1 u VALUE polje, a state u init

     Sledeće izvršavanje prelazimo u transient, VALUE = D2, S1 = D2- VALUE(VHT), STRIDE=S1

     Sledeće izvršavanje VALUE =D3, S2 = D3-Value(VHT), STRIDE=S2 , ako je S1=S2 prelazimo u Steady stanje

     Iz Steady stanja prelazimo u Transient ako Sj<>Sk

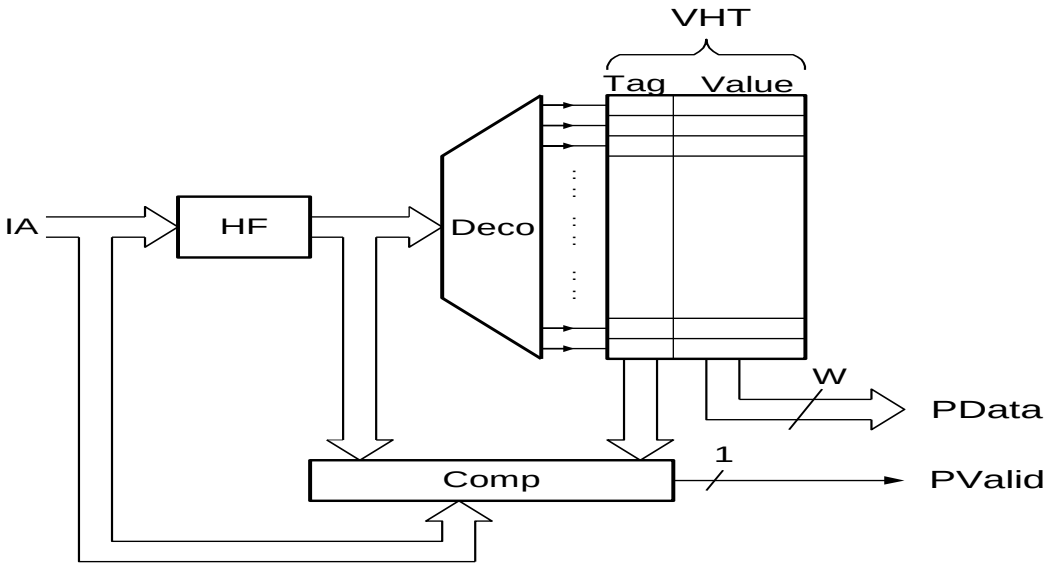


   - Ovaj prediktor pretpostavlja da će svako sledeće izvršavanje instrukcije na istoj adresi rezultovati prethodnom vrednošću uvećanom za neki konstantni pomeraj (polje stride). Najčešća primena – ažuriranje kontrolnih promenljivih u petljama.

2. Nacrtati šemu last outcome prediktora. Zašto ovaj prediktor nije dobar?

   - Predikcija iste vrednosti kad je ista instrukcija (koja prozivodi neki rezultat) bila izvršena poslednji put

     Samo neke instrukcije proizvode rezultat



   - *VHT* je ograničen resurs što za posledicu ima da se više instrukcijskih adresa preslikava u isti ulaz ove tabele. Zato se na dekoder vode samo niže linije adrese instrukcije (heš funkcija se najčešće implementira prostim odsecanjem viših bita). *Tag* polje sadrži odsečene bite adrese instrukcije koja je okupirala posmatrani ulaz. Poređenjem viših bita adrese instrukcije i sadržaja selektovanog *tag* polja utvrđuje se validnost spekulativne vrednosti (tj. da li se selektovani *value* odnosi na tekuću ili neku drugu instrukciju; ovo ne treba brkati sa korektnošću same spekulacije).
     Preostale tri startegije se mogu shvatiti kao nekakve nadogradnje ili modifikacije *last outcome* prediktora.

     Last outcome prediktor može da generiše tačnu ili netačnu vrednost. Ostali prediktori mogu i da zaobiđu spekulaciju (npr. *stride-based* prediktor u *Init* i *Transient* stanju). Dijagram ne prikazuje procenat promašaja (*VHT miss* odnosno neaktivnu vrednost *PValid* signala).

3. Objasniti predicated instrukcije.

   - Hibridna tehnika predikcije skokova

     Instrukcija obuhvata uslov koji se ispituje za vreme izvršenja

     Ako je uslov true – normalno izvršavanje, ako ne onda noop

     Predicated reg to reg move eliminiše skokove u nekim situacijama, npr if-then sa minimalnim telom

     Ograničenja:

    1. Canceled instrukcije troše taktove procesora, ne smeta ako bi ciklus bio svejedno idle
    2. Što pre otkrijemo skok to bolje
    3. Problem hvatanje izuzetaka
4. Razlika između fine grained i simultaneous multithreading? Koji nedostatak fine grained je nadomestio SMT?
    - Coarse-grained i fine-grained multithreading. Coarse-grained multithreading podrazumeva da do smenjivanja niti dolazi samo pri većim zastojima (eng. stalls) kao što su npr. page fault, L2 ili L3 keš promašaji. Fine-grained multithreading podrazumeva da do smenjivanja niti može doći nakon svake instrukcije. Fine-grained multithreading obezbeđuje veću propusnu moć jer do smenjivanja niti dolazi pri velikim ali i pri malim zastojima kao što je npr. L1 keš promašaj. S druge strane, fine-grained multithreading boluje od drugih nedostataka (npr. učestale promene konteksta dovode do narušavanja prostorne i vremenske lokalnosti zbog čega može nastupiti cache trashing).
5. Dati primer gde dvobitni prediktor skoro uvek pravi grešku.
    - Nekakva petlja u kojoj se skok dvaput izvršava, zatim dvaput ne izvršava i tako u krug. Ovakav primer ga zbuni jer nakon dva ista ishoda dobije čvrstu predikciju da će se ishod ponoviti.
6. Koja je prednost 2bit prediktora naspram 1bit? Napisati primer koda i jasno označiti gde je ta prednost.
    - Ako bismo imali petlju u petlji, i 1-bitni prediktor se koristi, prediktor bit bi se odmah invertovao na promašaju, pa bi novi ulazak u unutrašnju petlju na početku ponovo rezultirao promašajem.
7. Objasniti tehnike za prevazilaženje data zavisnosti, i opisati ih u jednoj rečenici.
    - Tehnike eliminisanja međuzavisnosti – Spajanje data-dependent instrukcija u jednu

      Tehnike skrivanja međuzavisnosti – Dok čekamo da podatak postane dostupan, druge data-independent instrukcije mogu biti izvršene u paraleli

      Tehnike predviđanja – Data value prediction
8. Koje su sve tehnike iskorišćenja na instrukcijskom nivou?
9. Koje su prednosti i mane oo (out-of-order issue, out-of-order completion)?
    - OoO predviđa izvršavanje instrukcija koje se nalaze iza one koja bi dovela do zaustavljanja pipeline zbog jednog od tri tipa hazarda u IO. Ovo je moguće zahvaljujući instruction window baferu koji se smešta između issue i execution jedinice. OoO u odnosu na IO uvodi novu vrstu hazarda - kasnije izdata instrukcija se može kompletirati ranije i prebrisati ulazne podatke za ranije izdate instrukcije.
10. Varijante ILP-a?

- SS – superscalar, VLIW – Very Long Instruction Word, SP – superpipelined ili hibridno (nekakva kombinacija). U osnovi svakog od ovih načina podrazumeva se pipeline organizacija.

11. Razlika izmedju VLIW i superskalarnih.

- VLIW – jedna instrukcija specificira veći broj operacija koje treba izvršiti u paraleli za razliku od SS koji izdaje više instrukcija istovremeno od kojih svaka treba da obavi tačno jednu operaciju. VLIW programi se sporije prevode i brže izvršavaju. Pogodan je za arhitekture i aplikacije specijalne namene. Nisu pogodni za binarnu kompatibilnost sa mašinama koje nisu VLIW.

12. MOESI, objasniti stanja.

- *MOESI* je dobio naziv po stanjima predviđenim za ovaj protokol: *Modified*, *Owned*, *Exclusive*, *Shared* i *Invalid*. Stanja se pridružuju keš linijama. Ovih pet stanja se može predstaviti koristeći tri bita:
  (1) *V* (*Validity*) – validnost bloka podataka u posmatranoj keš liniji;
  (2) *E (Exclusiveness)* – da li se blok podataka nalazi u tačno jednom kešu;

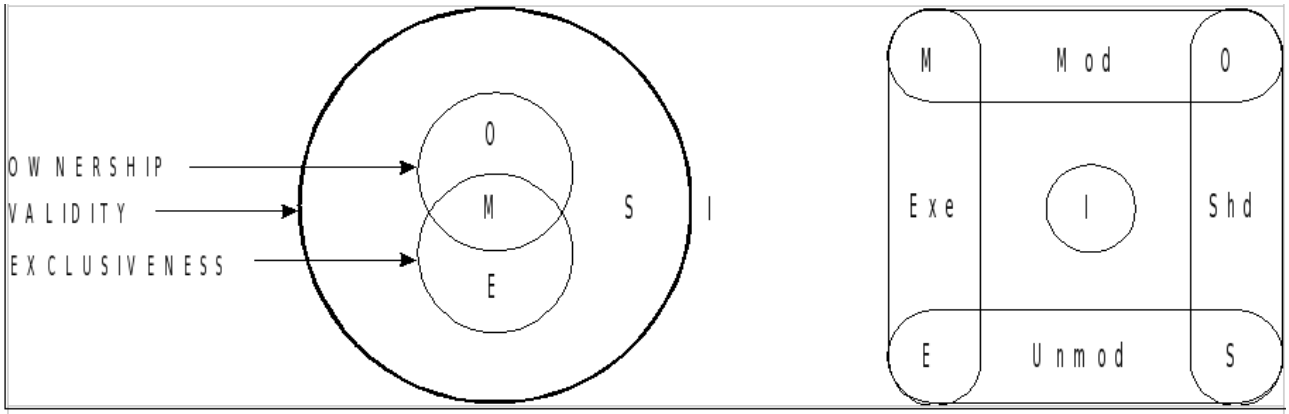


Figure SMPU3: Explanation of the MOESI protocol (source: [Tomasevic96]).

(3) *O* (*Ownership*) – da li je blok podataka modifikovan.

E i *O* nemaju smisla ako je *V* ukazuje na to da je blok nevalidan (zato je i broj stanja 5 a ne 8).
Sličnost između *Modified* i *Exclusive* stanja je u tome što tačno jedan keš sadrži posmatrani blok podataka (*E* bit). Razlika ovih stanja je u ažurnosti operativne memorije (*O* bit; za *Modified* je neažurna dok je za *Exclusive* stanje kopija bloka identična onoj u operativnoj memoriji).
Zajedničko za *Owned* i *Shared* stanje je to što više od jednog keša sadrži posmatrani blok podataka (*E* bit). Najviše jedna keš linija sa deljenim blokom podataka može biti u *Owned* stanju dok su preostale keš linije sa tim blokom u *Shared* stanju (*O* bit). Keš linije u *Shared* i *Owned* stanju uvek sadrže najažurniju kopiju deljenog bloka podataka. Kopija deljenog bloka u operativnoj memoriji može ili ne mora biti ažurna u zavisnosti od toga da li ne postoji ili postoji keš linija sa tim blokom koja je u *Owned* stanju.

13. Objasniti chained directory protokol.

- Deljene kopije bloka povezane u listu

Zauzeće prostora ~ O(logN), bez restrikcije broja kopija

Performanse bliske full-map šemama, ali veće kašnjenje pri upisu

- *Chained directory* protokol ublažava sve nedostake *full-map* metode žrtvujući performanse:
  (1) manje dimenzije *directory* kontrolera – M x $\log_2 N$ (na račun keš linija);
  (2) skalabilnost – broj procesora koji se može povezati u listu nije ničim ograničen;
  (3) sistem je više distribuiran pa se smanjuju nedostaci centarilzovane realizacije (*directory* kontroler kao usko grlo sistema i uzrok slabe otpornosti na otkaze).

14. Ukratko o operacijama čitanja i upisivanja za *Dir(i)NB* i *Dir(i)B* protokole. Kako su ovi protokoli dobili baš te oznake? Prednosti i mane *Dir(i)NB* i *Dir(i)B* protokola u poređenju sa *Dir(N)NB* protokolom?

- Presence bit po procesoru + dirty bit (sve to za svaki podatak)

  Dva bita u cache directory ulazu(valid+modified)

  Najbolje performanse ali ogromno zauzece prostora~O(N^2)

- Umesto N bita prisutnosti (gde je N broj procesora) koristi se i pokazivača na procesore ($i < N$; svaki pokazivač je širine $\log_2 N$). Postavlja se pitanje kako *directory* kontroler vodi evidenciju kada više od i procesora istovremeno želi da kešira isti blok podataka. U tom smislu se *limited directory* protokoli dele na *Dir(i)NB* i *Dir(i)B*.
  Kod *Dir(i)NB* protokola došlo je do usložnjavanja operacije čitanja; *i + 1.* procesor dovešće do preusmeravanja pokazivača i invalidacije odgovarajuće keš linije za neki od procesora koji su ranije pristupali posmatranom bloku podataka. To praktično znači da u *Dir(i)NB* varijanti *limited directory* protokola najviše *i* procesora može istovremeno

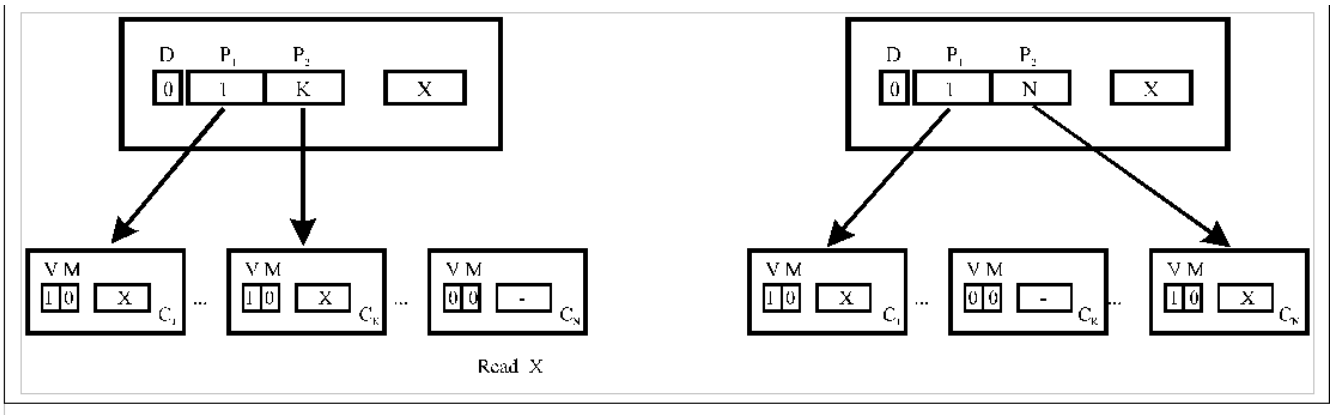


Figure SMPU6: Explanation of the no-broadcast scheme (source: [Tomasevic96]).

deliti neki blok podataka.

Kod *Dir(i)B* protokola došlo je do usložnjavanja operacije upisa; svakom ulazu *directory* kontrolera pridružuje se *B* (*Broadcast*) bit koji se setuje u slučaju da je broj procesora koji kešira posmatrani blok podataka veći od broja pokazivača po ulazu (engl. ***pointer overflow***); upisivanje u blok za koji je setovan *B* bit dovodi do *broadcast* invalidacije što u opštem slučaju predstavlja *overhead* jer nepotrebno opterećuje i procesore koji ne

keširaju sporni blok podataka.

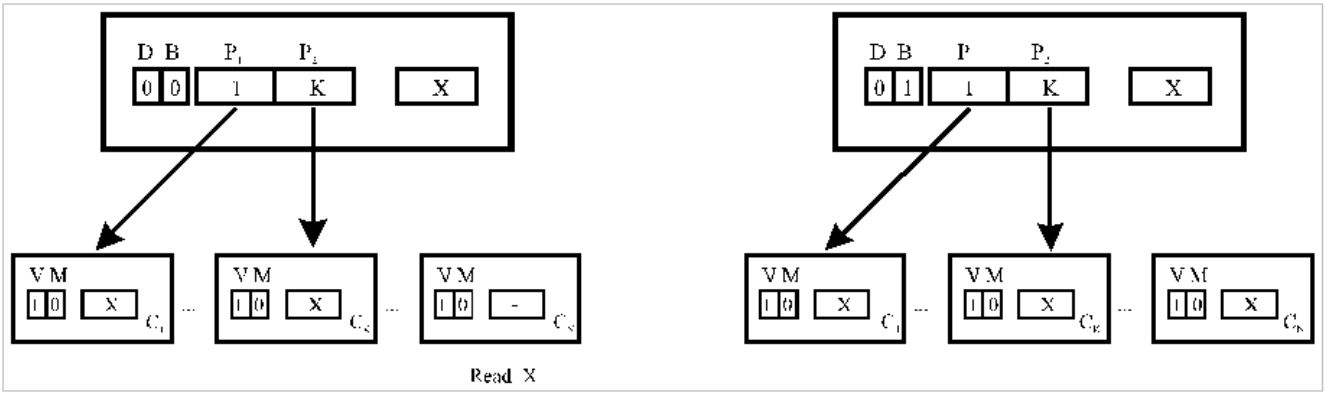


Figure SMPU7: Explanation of the broadcast scheme (source: [Tomasevic96]).

Dir označava da se radi o *directory* protokolu; *(i)* je broj pokazivača na procesore po jednom ulazu directory kontrolera; *NB* i *B* određuju da li *broadcast* operacija nije ili jeste neophodna za funkcionisanje tog protokola.
Limited protokoli žrtvuju performanse (usložnjavanjem operacije čitanja za *Dir(i)NB* ili upisivanja za *Dir(i)B*) kako bi: (1) smanjili dimenzije *directory* kontrolera – M x ($i \cdot \log_2 N$);
(2) učinili sistem skalabilnijim (broj procesora nije ograničen brojem pokazivača).
Nedostaci koji su posledica centralizovanog pristupa (usko grlo sistema po pitanju performansi i slaba otpornost na otkaze) i dalje postoje.