Zavisnosti po podacima preko memorije i uloge Load-store bafera i Spekulativnog store bafera

Zavisnosti po podacima preko memorije

U ugrađenoj dataflow mašini superskalarnog procesora preimenovanja i dinamičko određivanje zavisnosti po podacima radi se samo za arhitekturalne registre. Samim tim nije razrešeno pitanje eliminacije antizavisnosti i izlaznih zavisnosti, kada se podaci nalaze van registara, odnosno u memoriji. U skupovima instrukcija većine današnjih procesora, operacije Load i Store rade memorijske operacije između po jednog arhitekturalnog registra i memorijske lokacije. Određivanje svih vrsta zavisnosti po podacima preko memorije mora se obavljati na osnovu komparacije adresa lokacija u memoriji za store i load operacije. Važna je činjenica da se značajan procenat zavisnosti preko memorije ne može odrediti u vreme prevođenja, jer adrese lokacija u memoriji kod store i load operacija nisu tada poznate. Kako se adrese velikog broja lokacija u memoriji za store i load operacije određuju u vreme izvršavanja, očigledno je da opšti mehanizam utvrđivanja zavisnosti po podacima preko memorije mora da podrazumeva dinamičku komparaciju izračunatih adresa lokacija uključenih u memorijske operacije.

U memoriji nije moguće raditi dinamičko preimenovanje, jer postoji veliki broj lokacija, a svaki store bi se morao raditi u novu lokaciju. Zato se za podatke koji se nalaze u memorijskim lokacijama moraju poštovati antizavisnosti i izlazne zavisnosti, pored pravih zavisnosti po podacima. U vreme izvršavanja, prva faza izvršavanja Load ili Store operacija je određivanje adrese lokacije u memoriji. Na osnovu poznatih adresa lokacija u memoriji load i store operacija i poznatog originalnog redosleda tih operacija je moguće da se odrede sve vrste zavisnosti po podacima koje potiču od podataka zapamćenih u memoriji. U daljem tekstu će se analiza paralelizma rada memorijskog podsistema raditi za slučaj tag indexed mašine.

Spekulativnost po kontroli kod memorijskih operacija

Kako se rad ugrađene dataflow mašine zasniva na spekulativnosti po kontroli, osnovno pitanje kod memorijskih operacija je: da li je spekulativnosti po kontroli dozvoljena kod Load i Store operacija. U tom pogledu, značajno se razlikuju Load i Store operacije. Kod load operacije, ako je urađeno pogrešno predviđanje po kontroli, nepoželjan efekat je samo nepotrebno zauzeće memorijskog podsistema (zbog pogrešnog load-a) i fizičkog registra. Rezultat load operacije se smešta u fizički registar i izvršavanje operacija zavisnih po podacima od rezultata load-a kasni sve dok se ne završi load. Zbog sporosti memorijskih operacija, posebno u slučaju Load-a, poželjno je da se što pre počne izvršavanje Load operacija. Zato, uzimajući još u obzir tačnost predikcije grananja i malu verovatnoću pojave izuzetaka, spekulacija po kontroli, odnosno što pre započinjanje izvršavanja, u slučaju Load-a, je izuzetno poželjna i prihvatljiva.

U slučaju Store operacije, pogrešno predviđanje po kontroli, uz pogrešno upisanu vrednost u memorijskoj lokaciji, bi izazvalo velike problem i morao bi da se radi kompleksan oporavak. Pre svega, to bi bilo neprihvatljivo sporo, jer bi se moralo raditi poništavanje spekulativno urađene store operacije, a to bi zahtevalo čuvanje stare vrednosti memorijske lokacije kod svakog spekulativnog store-a. Spekulativno se može uraditi određivanje adrese i priprema podatka koji treba da se pamti u fizičkom registru. Sam upis podatka u memoriju ne sme da bude spekulativan! Zbog zavisnosti po podacima

operacija u instrukcijskom prozoru od sporih Load operacija i potrebe da se Store ne završava spekulativno, kod superskalarnih procesora apsolutni prioritet u pritupu memoriji daje load operacijama! Sa takvim prioritetom, ugrađena dataflow mašina radi brže.

Ako bi se u ROB-u, za commit operacija koje slede iza store operacije, čekao završetak upisa store operacije u memoriju, kočilo bi se ažuriranje in order arhitekturalnog stanja i samim tim nepotrebno zauzimali resursi procesora. Zato su izračunata adresa store operacije i pripremljen (data ready) podatak u registru koji treba da se upiše u memoriju, dovoljan uslov za commit (delimičan) sa stanovišta funkcionisanja ROB. Dakle, za započinjanje faze upisa u memoriju svake Store operacije, potreban je uslov da se mora čekati na delimičan "commit" store operacije u Reorder baferu, jer tek tada store prestaje da bude spekulativna operacija po kontroli.

Store operacije koje dožive commit (delimičan) iz ugla funkcionisanja ROB preuzima deo procesora koji se zove writeback bafer, store bafer ili spekulativni store bafer. On mora da čuva originalni redosled store operacija kada zaista realizuje upis u memoriju zbog izlaznih zavisnosti. Osim toga, writeback bafer, store bafer ili spekulativni store bafer mora da upravlja stvarnim kompletnim završavanjem store operacija (finalni upis u memoriju). Kada se desi exception ili branch misprediction, prazni se ROB. Međutim, na završavanje store operacije za koju je već bio urađen commit u ROB pre ubacivanja u writeback bafer ili spekulativni store bafer, to nema nikakvog uticaja. Spekulativni store bafer tada preuzima na sebe ulogu da kompletira sve komitovane store operacije.

Uočimo proces commit-a operacija za Store i Load u ROB. Očigledno je da se Store operacije sa stanovišta ROB moraju smatrati završenim (ali ne još commit u ROB), ako su samo izračunati adresa lokacije i pripremljen podatak koji tek treba da se zapamti u memoriji! To znači da je podignut flag te store operacije u ROB da je operacija završena, iako još nije stvarno završena. Tek kada se uradi commit te delimično završene store operacije u ROB, stvoren je preduslov da se kompletno završi store operacija – zaista upiše podatak u memoriju. Kako skupovi nekoliko instrukcija u jednom ciklusu mogu da dostignu commit u ROB, nema ograničenja da tu budu i potpuno završene load i poluzavršene store operacije.

Nakon napuštanja ROB ne sme da se gubi originalni redosled load i store operacija, ako se ne obavi kontrola da li postoji neki tip zavisnosti. Međutim, taj originalni redosled tada i dalje čuvaju baferi, pre svega spekulativni store bafer koji je FIFO tipa.

Ograničenja zbog zavisnosti po podacima kod memorijskih operacija

Na prvi pogled, memorijske operacije mogu da dovedu do značajnog ograničavanja paralelizma u izvršavanju operacija u ugrađenoj dataflow mašini. Potpuno naivno rešenje bi izbegavalo analizu zavisnosti po podacima preko memorije komparacijom odgovarajućih adresa memorijskih operacija, ali bi se time znatno gubilo na paralelizmu. Ono bi podrazumevalo da potpuni završetak svake prethodne store operacije po originalnom redosledu predstavlja dodatni preduslov za započinjanje svake sledeće load ili store operacije po originalnom redosledu zbog potencijalnih pravih i izlaznih zavisnosti po podacima preko memorije. Takođe, završetak Load operacije bi bio preduslov za izvršavanje svake naredne store operacije po originalnom redosledu, zbog potencijalnih antizavisnosti. Takva rešenja bi drastično smanjila mogućnost izvršavanja svih operacija van originalnog redosleda i neprihvatljiva su za

ugrađenu dataflow mašinu. Kao rezime, mora postojati mehanizam komparacije adresa store i load adresa i mehanizam pamćenja redosleda memorijskih operacija nezavisan od ROB koji će obezbediti maksimalan paralelizam memorijskih operacija.

U anglosaksonskoj literaturi se koristi termin memory disambiguation za dinamičko određivanje zavisnosti po podacima u memoriji. Da bi se odredile pravilno zavisnosti, moraju se poznavati ne samo originalan redosled store operacija, već uzajamni redosled i store i load operacija. Zato se često koristi termin Load-Store bafer. Logika oko Load-Store bafera mora da obezbedi asocijativno pretraživanje po adresama i da određuje prave zavisnosti i antizavisnosti preko memorije, pa da se na osnovu tako određenih zavisnosti odredi kada se mogu započeti memorijske operacije.

Osnovna pravila za zavisnosti po podacima preko memorije su tada sledeća:

1. Store operacije (faza pamćenja podatka u memoriji) se mogu obaviti samo u programskom-originalnom redosledu (da bi se poštovale izlazne zavisnosti)
2. Load se može započeti pre ranije Store operacije po originalnom redosledu, samo ako ne pristupaju istoj memorijskoj lokaciji, odnosno nemaju istu adresu za operaciju. (da bi se poštovale prave zavisnosti)
3. Store operacija može da započne pre ranije Load operacije, ako ne pristupaju istoj memorijskoj lokaciji, odnosno nemaju istu adresu za operaciju. (Antizavisnost)

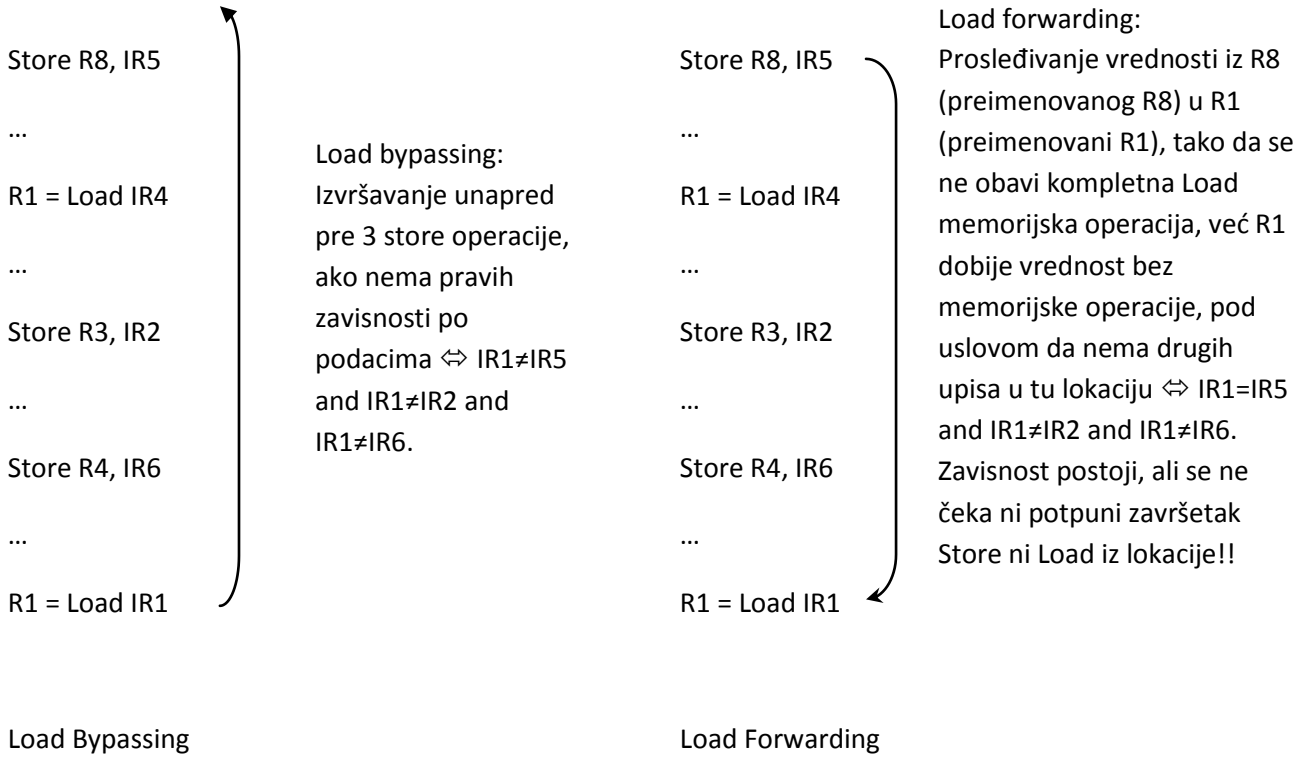
Uslov 1. je ispunjen zahvaljujući writeback (spekulativnom store) baferu, jer on obezbeđuje pamćenje podataka u memoriji po originalnom redosledu FIFO disciplinom. Uslov 3 je ispunjen time što podatak iz store operacije, za koju se radi commit (delimičan) u ROB, još nije upisan kada je prethodna load operacija morala da se kompletno završi, da bi se uradio njen commit. Prema tome, ne može se narušiti antizavisnost jer, zbog nemogućnosti da se store kompletno obavi spekulativno, ranija load operacija mora da prođe commit fazu u ROB, pre nego što se store operacija prebaci u writeback (spekulativni store) bafer. Ključan problem je kako razrešiti prave zavisnosti po podacima, slučaj 2.

Izbegavanje nepotrebnih dugotrajnih memorijskih operacija

U slučaju javljanja prave zavisnosti po podacima preko memorije, kašnjenja mogu biti velika, jer se na prvi pogled prvo mora sačekati potpuni commit Store operacije, a tek zatim započeti load operacija. Kako obe operacije dugo traju, operacije zavisne po podacima od load operacije u ugrađenoj dataflow mašini u tom slučaju morale bi da čekaju na podatak iz memorije, što bi dovodilo do znatnog usporenja rada superskalarnog procesora. Superskalarni procesor sa ugrađenom dataflow mašinom se na visokom nivou apstrakcije može posmatrati kao crna kutija koja prima i velikom prosečnom brzinom izvršava instrukcije koje treba da se završe. Međutim, crna kutija zakašnjava generisanje potvrđenih rezultata za oko stotinak instrukcija, a u slučaju store operacije za više stotiina operacija.

Postoje dve osnovne transformacije koje dovode do bržeg izvršavanja koda. Njihovi nazivi su Load Bypassing i Load Forwarding. Cilj Load Bypassing-a je da se Load operacija obavi pre Store operacija koje su prethodile po originalnom redosledu. Bazični mehanizam kojim se to postiže je sledeći: postoji zajednička rezervaciona stanica za Load i Store i ona izdaje store i load instrukcije u originalnom

redosledu. Kao posledica načina na koji se Store instrukcije smatraju završenim (još nisu delimično kompletne), sve Store instrukcije mogu da se smeste u Store buffer. Store buffer ima dva dela: završene i delimično kompletne store instrukcije u originalnom redosledu. Ako se asocijativnim pretraživanjem utvrdi da nema nijednog upisa u celom store bufferu na adresu sa koje load treba da čita, nema zavisnosti po podacima i load može da krene u izvršavanje pre potpunog završetka svih prethodnih store operacija. Ovde je uočljivo da postoji ograničenje in order izdavanja store i load instrukcija.



Sl. 1. Load Bypassing i Load Forwarding, gde IR predstavljaju registre sa adresama, a prikazan je originalni redosled

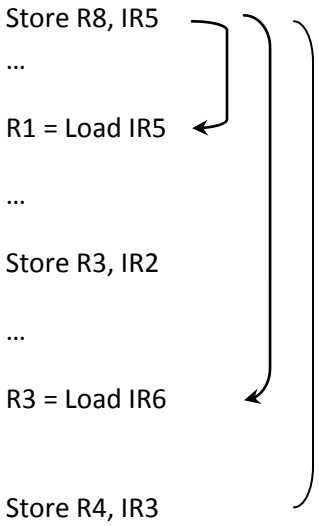
Na osnovu pravila 2., da bi se poštovala prava zavisnost po podacima, Load operacija na istoj lokaciji bi morala da sačeka završetak prethodne store operacije u toku rada ugrađene dataflow mašine. Međutim, ako postoji takva prava zavisnost i store nije kompletno završen, podatak potreban load operaciji se već nalazi u procesoru ili treba da bude izračunat radom ugrađene dataflow mašine. Ako je već izračunat podatak koji treba da se pamti sa store operacijom, Load može odmah da pokupi taj podatak iz fizičkog registra operanda store operacije i da ga prenese u registar gde se pamti rezultat load-a. Ako podatak za pamćenje u store operaciji nije izračunat, čim postane data ready, treba da se pokrene proces prenošenja njegove vrednosti u destinaciju load operacije, ako je moguće. Na ovaj način se izbegavaju kašnjenja koja potiču od vremena: pamćenja podatka u memoriji Store operacije i čitanja podatka iz memorije Load operacija.

Iz prethodne analize se nameće rešenje. Kod ubacivanja load instrukcije u instrukcijski prozor (in order) proverava se asocijativnim pretraživanjem store instrukcija da li postoji jednakost njihovih memorijskih adresa sa memorijskom adresom Load operacije. To pretraživanje se radi za sve store instrukcije koje su smeštene u store baferu. Zanimljivo je da se za store instrukcije mora raditi popunjavanje store buffer-a sa spekulativnim in order store instrukcijama koje su sve završene a nisu delimično komitovane, nezavisno od kompletiranja u ROB. Dakle, skup spekulativnih store instrukcija se deli na sve susedne blizu tail-a ROB koje su završene i smeštene u store buffer i one koje su komplementarne tom podskupu u ROB, među kojima poslednja od njih ili nema izračunat operand ili nema izračunatu adresu odredišta.

Ako nema nijedne store instrukcije za istu lokaciju iz koje se radi Load u store bufferu, znači da nema prave zavisnosti po podacima preko memorije unutar instrukcijskog prozora ugrađene dataflow mašine. Ako postoji jedna store instrukcija za istu lokaciju, load instrukcija dobija operand store instrukcije kao podatak za svoju destinaciju Load Bypass-om. Na ovaj način su izbegnuta kašnjenja i store i load operacije, a prava zavisnost po podacima se poštuje, ali su kašnjenja zavisnosti potpuno uporediva sa kašnjenjima zavisnosti kod registarski operacija.

Ako ima dve ili više store operacija u store bufferu za istu lokaciju u memoriji iz koje se radi novi load, Logika postaje kompleksnija. Tada se mora odrediti od koje Store operacije mora da se uradi forwarding.

Na Sl. 2. je prikazan takav slučaj.



Load Forwarding sa eliminacijom poslednje faze Store operacije Store R8, IR5 (upis u memoriju). Uslovi: IR3 = IR5 and IR2≠IR5, IR6 = IR3 = IR5, IR5 se ne menja u ovom delu koda. Nema drugih Load operacija za koje nije urađen forwarding.

1. Uradi se forwarding da R1 := R8 i R3 := R8.
2. Poslednja faza Store R8, IR5 postaje nepotrebno i smanjuje se broj pristupa memoriji.

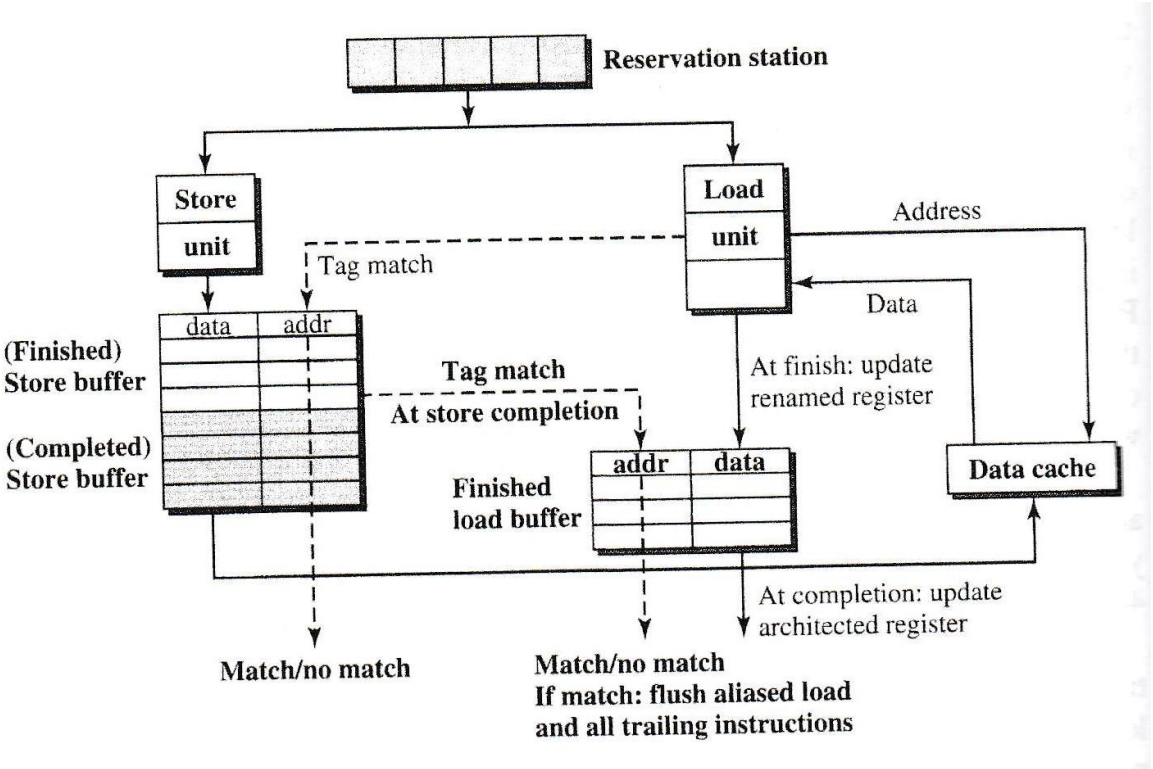
Sl. 2. Višestruki Load Forwarding kombinovan sa eliminacijom nepotrebnih Store operacija, gde IR predstavljaju registre sa adresama, a prikazan je originalni redosled

Na osnovu Sl. 2. je jasno da se mora uspostaviti relativni odnos load i store operacija po originalnom redosledu izvršavanja. Zato i Load mora da ima svoj in order Load buffer u kome se nalaze Load instrukcije koje čekaju na čitanje memorije i mora se uspostaviti in order relacija sa Store bufferom.

Jedno od ključnih pitanja je šta raditi sa load instrukcijama kada može da se započne load, a postoje store instrukcije koje su joj prethodile po originalnom redosledu, a još nisu ušle u Store buffer. Opcije su:

a. Čekati da sve prethodne store instrukcije uđu u store buffer, pa tek tada započeti memorijski deo load operacije
b. Započeti fazu load-a sa čitanjem iz memorije, a istovremeno raditi asocijativno pretraživanje adresa svih load-a u Load buffer-u pri svakom novom ubacivanju Store operacije u Store buffer. Ako se pojavi jednakost adresa, a Store je prethodio load-u, obustaviti čitanje memorije i uraditi forwarding.

Prvo rešenje nije dobro, jer se usporava izvršavanje load operacije, a time i svih operacija zavisnih po podacima od Load operacije. Drugo rešenje zahteva komplikovaniju logiku, ali je paralelizam mnogo veći i nema nepotrebnih kašnjenja u obavljanju Load operacija. To rešenje je prikazano na Sl. 3., ali ono ne pokazuje eliminaciju nepotrebnih Store operacija. U agresivnoj varijanti, ovo rešenje dozvoljava izvršavanje operacija zavisnih od load-a kada još nije provereno da li neki prethodni store po originalnom redosledu ima istu adresu memorijske lokacije. U toj varijanti, moraju se poništavati sve zavisne operacije i ponovo raditi nakon forwarding-a i ispravnog rezultata Load operacije.

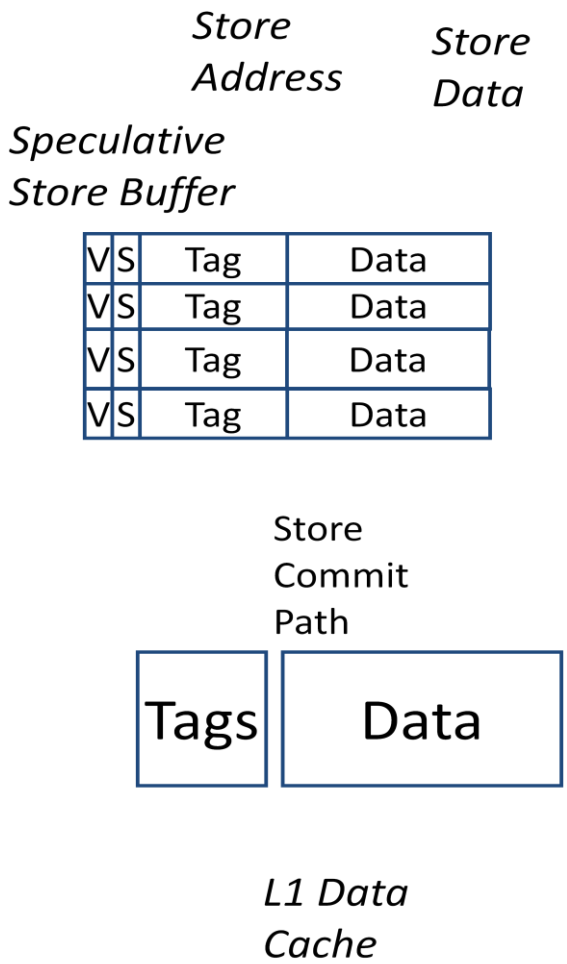


Sl. 3. Započinjanje faze load-a sa čitanjem iz memorije, uz istovremeno asocijativno pretraživanje adresa svih Load-a u Load buffer-u pri svakom novom ubacivanju Store operacije u Store buffer

Logika za eliminaciju nepotrebnih store operacija i odvajanje delova store buffera po spekulativnosti

Ako se uradi forwarding za sve load operacije sa iste lokacije, tada finalna faza upisa ranije store operacije po originalnom redosledu postaje ekvivalent mrtvog koda! Na taj način se može redukovati ukupan broj memorijskih operacija na osnovu lokalizma pristupa istoj lokaciji na dinamičkom tragu.

Da bi se to postiglo, potrebno je dodati odgovarajuće bite kontrole u store buffer i usvojen termin za takvo rešenje je Spekulativni store bafer. On je prikazan na Sl. 4. Osnovni dodatak u odnosu na do sada opisanu logiku je postojanje dva bita uz svaki ulaz u store buffer. Jedan bit odvaja spekulativni od nespekulativnog dela store buffera i označen je sa S. Njega ažurira ROB, a inicijalno u tipičnom slučaju označava da je store spekulativan. Kada Store prestane da bude spekulativan (urađen je delimični commit), podatak je spreman da se upiše u memoriju. Uloga drugog bita je da eliminiše nepotrebne upise u memoriju Store operacija koje postaju mrtav kod zbog forwardinga. On se naziva Valid bit V i kada logika otkrije da je neka Store operacija postala mrtav kod, eliminiše potrebu za upisom u memoriju.



Sl. 4. Spekulativni store buffer

Uz pomoć Load Bypassing, Load Forwarding-a i eliminacije nepotrebnih Store operacija se ubrzava izvršavanje prosečnog koda za preko 20%. Osim toga, uklanjanjem nepotrebnih Store operacija se smanjuje opterećenje memorije, što je veoma važno, pošto memorija predstavlja usko grlo današnjih

procesora. Sa povećanjem broja instrukcija u ugrađenoj dataflow mašini će se još više povećati značaj Load Bypassing, Load Forwarding-a i eliminacije nepotrebnih Store operacija.