Tag indexed varijanta superskalarnih procesora

Da bi se lakše prikazao rad ugrađene data flow mašine u ovoj varijanti superskalarnog procesora, inicijalno će se prikazati kako radi instrukcijski prozor kada se ubacuje samo jedna instrukcija po ciklusu. Nakon objašnjavanja rada sa jednom instrukcijom po ciklusu, biće opisan rad kada se veći broj instrukcija ubacuje u instrukcijski prostor.

Dinamička preimenovanja

Prilikom ubacivanja nove instrukcije u prozor, mora se uraditi preimenovanje u cilju uklanjanja antizavisnosti ili izlaznih zavisnosti. Preimenovanje se u ovom slučaju radi preko registara, za razliku od Tomasulo algoritma. Zato je neophodno da postoje dve kategorije registara. Pre svega, moraju da postoje registri vidljivi sa stanovišta arhitekture, kompajlera i mašinskih instrukcija koje nazivamo arhitekturalnim registrima. Sa stanovišta korisnika procesora, JEDINO ovi ARHITEKTURALNI registri su vidljivi. Njihove adrese se nalaze u kôdu mašinskih instrukcija. Druga kategorija registara su fizički registri koji sadrže preimenovane vrednosti arhitekturalnih registara. Unutar instrukcijskog prozora, kao mesta odakle se uzimaju argumenti instrukcija i gde se pamte rezultati instrukcija, egzistiraju SAMO fizički registri i rezultati zapamćeni unutar fizičkih registara. Broj fizičkih registara mora očigledno da bude veći od broja arhitekturalnih registara zbog višestrukih preimenovanja arhitekturalnih registara. Međutim, broj aktivnih preimenovanja (preimenovanja u jednom trenutku potrebnih za instrukcije u dataflow prozoru) za različite arhitekturalne registre je u principu različit i zavisi od programa koji se trenutno izvršava. Ukoliko je instrukcijski prozor veći (ugrađena data flow mašina ima veći dinamički graf zavisnosti po podacima), utoliko je potrebno da postoji veći prosečan broj fizičkih registara potrebnih za preimenovanje po arhitekturalnom registru.

Zbog postojanja dinamičkog preimenovanja, nadalje će se razmatrati samo direktne prave zavisnosti po podacima, pa svako referisanje zavisnosti po podacima će podrazumevati tu vrstu zavisnosti. Današnje tipične vrednosti za prosečan broj preimenovanja je 4 fizička registra po jednom arhitekturalnom registru. Eliminisanje antizavisnosti i izlaznih zavisnosti preimenovanjem, unutar instrukcijskog prozora, ekvivalentno je sa pravilom da se unutar instrukcijskog prozora mora obezbediti da je samo jednom dozvoljen upis u registar (single assignement rule). Zato pri ubacivanju instrukcije u prozor mora da postoji slobodan fizički registar čiji se tag, adresa tog fizičkog registra, odmah koristi za adresiranje lokacije budućeg rezultata instrukcije. Otuda naziv tag-indexed za ovu klasu superskalarnih procesora.

Osim zauzimanja slobodnih fizičkih registara za budući rezultat svake ubačene instrukcija u instrukcijskom prozoru, mora da se obezbedi i mehanizam oslobađanja fizičkih registara za ponovnu upotrebu. Oslobađanje sme da se obavi tek kada su sve instrukcije zavisne po podacima od tog registra pročitale vrednosti svojih argumenata. Detalji ovog mehanizma su opisani kasnije.

Implementacije dinamičkog preimenovanja

Prvobitna rešenja su podrazumevala da broj fizičkih registara za preimenovanja po svakom arhitekturalnom registru bude isti i jednak za sve arhitekturalne registre. Ovakvo rešenje ima dve dobre osobine i jednu ozbiljnu manu. Dobra je osobina da se na adresu arhitekturalnog registra povećanjem

broja bita adrese dodaje polje koje nosi informaciju o kojem je konkretnom preimenovanju reč. Dakle, u ovom rešenju, ako imamo maksimalno 4 preimenovanja po svakom arhitekturalnom registru, dva bita polja dodatog na adresu arhitekturalnog registra određuju jedinstveno o kojem od maksimalno 4 preimenovanja se radi.

Činjenica da ukupna adresa fizičkog registra i dalje sadrži adresu arhitekturalnog registra koji je preimenovan omogućava da u hardveru procesora ne treba da postoji gotovo nikakav hardver za inverzna preslikavanja iz fizičke adrese u adresu arhitekturalnog registra, jer je ta adresa već sastavni deo adrese fizičkog registra. To je druga prednost. Dakle, jednostavnom maskom, iz adrese fizičkog registra, se dobija adresa arhitekturalnog registra koji je preimenovan.

Loša strana ovog naizgled jednostavnog i praktičnog rešenja je što u praksi, tokom izvršavanja nekog programa, broj potrebnih preimenovanja po različitim arhitekturalnim registrima može da bude značajno različit. Nedostatak slobodnog fizičkog registra za arhitekturalni registar rezultata instrukcije koja treba da se ubaci u prozor dovodi do zaustavljanja popunjavanja instrukcijskog prozora. U isto vreme, neki drugi arhitekturalni registar može da ima samo jedno aktivno preimenovanje i tri neiskorišćena pridružena fizička registra. To neefikasno korišćenje fizičkih registara predstavlja veliku manu, jer zahteva značajno veći broj fizičkih registara nego što bi bilo potrebno da nema takvog formiranja adrese.

Proizvoljna preslikavanja

Da bi se ta mana izbegla, uvedeno je opšte preslikavanje u kome preslikavanje nekog arhitekturalnog registra može da bude na bilo koji slobodan fizički registar. Da bi se to postiglo, mora da postoji tabela (memorija) koja obezbeđuje proizvoljno preslikavanje iz adrese arhitekturalnog registra u adresu fizičkog registra. Pretpostavimo da je broj arhitekturalnih registara 2^m, a broj fizičkih registara 2^n, gde je n>m. Tada ova memorija sadrži 2^m lokacija, a broj bita podataka je n. Reči koje se čitaju (podaci) su ustvari adrese fizičkih registara u koje se preslikavaju adrese arhitekturalnih registara. Treba uočiti da ovakvo preslikavanje mora da se odnosi na **samo jedno** od preslikavanja arhitekturalni registar – fizički registar, jer ih u principu ima više u jednom trenutku u ugrađenoj dataflow mašini. Od logike preslikavanja zavisi koje će preimenovanje biti izabrano za svaki arhitekturalni registar.

Kod nekih implementacija, potrebno je i inverzno preslikavanje iz adrese aktivnog fizičkog registra u adresu arhitekturalnog registra za koji je taj fizički registar jedna od preimenovanih vrednosti. To znači da kada se neaktivan (slobodni) fizički registar iskoristi za preimenovanje nekog arhitekturalnog registra, odmah mora da se popuni i lokacija u tabeli za inverzno preslikavanje koje omogućava da se kasnije, kada je potrebno, dobije arhitekturalni registar za koji je fizički registar preimenovana vrednost. Tada postoji 2^n lokacija, a broj bita podataka (adresa arhitekturalnog registra) ima m bita.

Rešenje sa proizvoljnim preslikavanjem značajno povećava iskorišćenost fizičkih registara, jer se ubacivanje u instrukcijski prozor zbog fizičkih registara koči samo kada su svi fizički registri aktivni, odnosno nema nijednog slobodnog fizičkog registra. Pod aktivnim registrima se smatraju registri koji još nemaju izračunatu vrednost rezultata instrukcije kojoj su pridruženi i registri koji sadrže validan rezultat instrukcije, a nisu dealocirani. Iako je instrukcija izračunavanjem rezultata nestala iz dinamičkog DDG-a

ugrađene data flow mašine, sve dok još postoje instrukcije kojima je rezultat u fizičkom registru potreban, potrebno je sačuvati te vrednosti i ne sme dealocirati fizički registar.

Osnovna mana ovog rešenja je što u inverznom preslikavanju mora da se doda jedan protočni stepen zbog čitanja tabele inverznog preslikavanja adresa fizičkog registra - adresa arhitekturalnog registra za koji se radi preimenovanje. Dakle, zbog slobode preslikavanja postoji ciklus kašnjenja kod inverznog preslikavanja.

Izmene instrukcije po ubacivanju u instrukcijski prozor

U daljem tekstu će se ukupna adresa fizičkog registra zvati tag-om, kako se dalje ne bi posebno razmatrale varijante formiranja adrese fizičkog registra. Dakle, unutar instrukcijskog prozora, svaki rezultat i svaki argument mora da se jedinstveno referišu preko tagova fizičkih registara. Jednaka vrednost taga rezultata instrukcije i taga argumenta instrukcije, ako su obe instrukcije u prozoru ugrađene data flow mašine, znači da postoji direktna prava zavisnost po podacima između te dve instrukcije.

Prilikom ubacivanja instrukcije u prozor, već je definisano da se zbog single assignement rule-a (u stvari preimenovanja rezultata svih instrukcija) mora zauzeti slobodan fizički registar za budući rezultat te instrukcije. To zauzimanje mora da se uradi odmah po ubacivanju instrukcije u instrukcijski prozor iz više razloga. Jedan razlog je što argumenti te instrukcije mogu odmah da budu data ready, pa je u kratkom intervalu potrebno pamtiti rezultat. Međutim, mnogo je važnije da se za potencijalno referenciranje tog rezultata obezbedi jedinstveni identifikator rezultata, a tag zauzetog fizičkog registra nam upravo to pruža.

Prilikom ubacivanja instrukcija u prozor, one se još nalaze u originalnom sekvencijalnom redosledu. Potrebno je odrediti i argumente tih instrukcija koje moraju da budu vezane jedinstvenom referencom - tagom fizičkog registra u kome se nalazi ili u kome će da se nalazi argument. Koji fizički registar treba da sadrži argument nove instrukcije koju ubacujemo u instrukcijski prozor, može da se odredi samo na osnovu adrese arhitekturalnog registra, argumenta instrukcije. Međutim, kako je preslikavanje iz arhitekturalnog registra u fizički registar jedan na više, treba da postoji način na koji se nalazi pravi tag fizičkog registra koji sadrži, ili će sadržavati vrednost za argument instrukcije. Koji je tag pravi, može da se odredi samo na osnovu redosleda ubacivanja instrukcija u prozor, odnosno originalnog sekvencijalnog redosleda. Ako je adresa arhitekturalnog registra argumenta instrukcije koja treba da se izvrši u sekvencijalnom kôdu Ri, tada se kod sekvencijalne mašine argument uzima iz arhitekturalnog registarskog file-a, a to potiče od poslednje instrukcije u originalnom redosledu koja je upisala rezultat u Ri. Arhitekturalno stanje kod superskalara nije ništa drugo nego stanje svih Ri registara mašine, jer poslednje preimenovanje arhitekturalnih registara u fizičke registre po sekvencijalnom redosledu čuva sve potencijalne argumente i predstavlja ekvivalent arhitekturalnih registara, kada bi se sekvencijalno izvršavao kôd. Zato se za preslikavanje adresa arhitekturalnih registara instrukcije koja se ubacuje u instrukcijski prozor u adrese fizičkih registara koristi remap (rename) file koji čuva aktuelno arhitekturalno stanje.

Kada se **statički** formira graf zavisnosti po podacima za bazični blok, tada je neohodno pronaći u in order redosledu poslednju instrukciju koja upisuje u arhitekturalni registar pre nego što instrukcija zavisna po podacima čita iz istog arhitekturalnog registra, da bi kompajler utvrdio da postoji prava direktna zavisnost po podacima. Kod superskalara se u instrukcijski prozor ubacuje predviđeni dinamički trag, koji je u osnovi **predviđeni dinamički bazični blok**. Razlika je samo u tome što se tokom rada procesora dinamički dodaju nove instrukcije u dinamičkom grafu zavisnosti kako se ubacuju nove instrukcije u instrukcijski prozor i udaljavaju završene instrukcije. Nove instrukcije u prozoru u trenutku ubacivanja postaju nove instrukcije slobodne na dnu dinamičkog grafa zavisnosti po podacima, jer u osnovi imamo klizeći dinamički prozor ugrađene dataflow mašine za dobijanje dinamičkog grafa zavisnosti po podacima trenutno aktuelnog ekvivalenta bazičnog bloka.

Prilikom ubacivanja svake pojedinačne instrukcije u prozor, rezervišemo fizički registar preko jedinstvenog taga za instrukcijski prozor P_I, koji zamenjuje adresu arhitekturalnog registra rezultata R_i. Do sledeće instrukcije kojom se radi upis u isti arhitekturalni registar R_i po in-order redosledu, taj tag P_I treba da bude jedinstvena referenca za argumente svih instrukcija koje u kôdu referišu taj arhitekturalni registar R_i, a ubacuju se u prozor. Da bi se to postiglo, **svako** novo preslikavanje adrese arhitekturalnog registra rezultata instrukcije u tag slobodnog fizičkog registra, prilikom ubacivanja instrukcija u ugrađenu dataflow mašinu, mora da se zapamti u remap (rename) file. Tako se u brzoj maloj memoriji čuva poslednje preslikavanje svih arhitekturalnih registara u fizičke po originalnom redosledu. To je upravo ono što je potrebno narednoj instrukciji koja se ubacuje u instrukcijski prozor, da bi se odredili tagovi argumenata i predstavlja arhitekturalno stanje.

Dinamički DDG ugrađene data flow mašine

Promena grafa se odigrava gotovo svakog ciklusa, jer skoro svakog ciklusa se ubacuje bar jedna instrukcija i vrlo verovatno završava bar neka druga instrukcija! Instrukcije, kada se završe, nestaju iz takvog dinamičkog grafa zavisnosti po podacima.

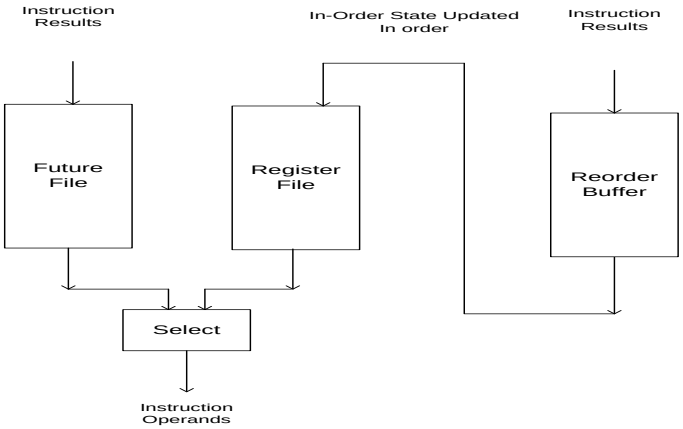
Prava direktna zavisnost po podacima se **dinamički** uspostavlja referenciranjem fizičkog registra u koji neka instrukcija upisuje rezultat u preimenovanom argumentu instrukcije zavisne po podacima(direktna prava zavisnost po podacima). Dakle tag, osim adrese registra, praktično označava podatak od koga je zavisan skup instrukcija koje kao argument referenciraju taj tag (podatak). Dakle, u hardveru za preimenovanje nemamo pojedinačno referenciranje grana dinamičkog grafa zavisnosti po podacima, već referenciranje grupe grana koje polaze iz istog čvora. Da li to može da podrži rad ugrađene data flow mašine? Za dataflow mašine važi osnovno pravilo da instrukcija kreće u izvršavanje (izdaje se na izvršavanje) kada svi argumenti postaju data ready. Zato se mora, svakoj od lokacija za rezultate instrukcija, odnosno aktivnim registrima, pridružiti ready bit, kako bi za zavisne instrukcije moglo da se odluči kada mogu da krenu u izvršavanje. Dakle, za pokretanje podacima je dovoljno takvo referenciranje skupa grana preko tag-a.

Ready bit fizičkog registra mora da se prosledi do svih zavisnih instrukcija u prozoru, koje treba nekako otkriti. To otkrivanje se obavlja asocijativnim pretraživanjem istovetnih tagova za argumente i

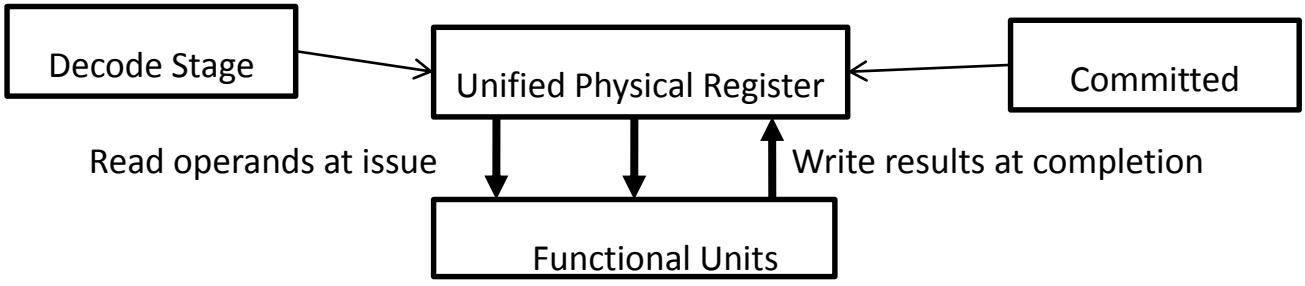
paralelnim setovanjem ready bita svih tih argumenata instrukcija zavisnih po tom podatku. Tada se ujedno upisuju vrednosti svih odgovarajućih argumenata u polja instrukcija koje čekaju da postanu spremne za izvršavanje. Asocijativna memorija u tag-indexed varijanti ima istu ulogu kao i zajednički data bus kod Tomasulo varijante superskalara. Ovo asocijativno pretraživanje i setovanje se radi u reorder bufferu ili issue queue (redu za izdavanje instrukcija), zavisno od izvedbe mašine.

Reorder buffer (ROB)

Funkcija ROB je gotovo identična kao kod Tomasulo algoritma. On čuva instrukcije iz instrukcijskog prozora u originalnom redosledu i vodi evidenciju o arhitekturalnom registru u koji se upisuje rezultat, preimenovanom fizičkom registru za taj arhitekturalni registar i ima ready bit za rezultat. Svaki upis rezultata povlači i ažuriranje odgovarajućeg ready bita u ROB. Kao i kod Tomasulo algoritma, pokazivač na najstariju instrukciju u originalnom redosledu koja je u instrukcijskom prozoru (kojoj nije izračunat rezultat, čim je u prozoru) pokazuje ujedno da su sve instrukcije pre nje u originalnom redosledu komitovane. Deo posla ROB je da stalno ažurira koja je najstarija nezavršena instrukcija, oslobađa lokacije komitovanih instrukcija i da obavlja ažuriranje vrednosti odgovarajućih arhitekturalnih registara in order stanja u komitovanom registarskom fajlu. Realizacija sa posebnim registarskim file-om za in order komitovano stanje je prikazan na slici, pri čemu future file čuva lookahead stanje

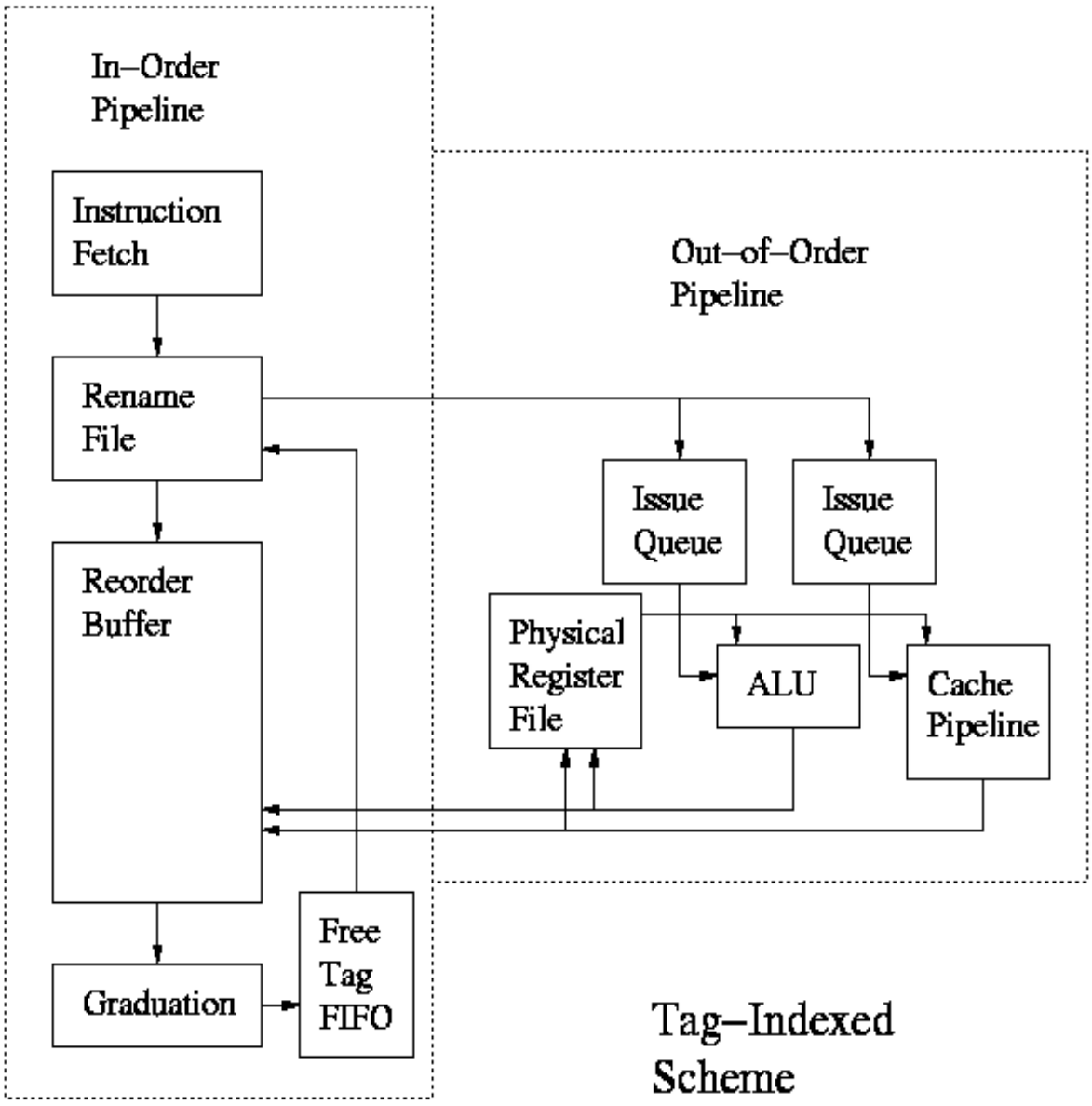


Druga realizacija je da se komitovane vrednosti čuvaju u registarskom file-u fizičkih registara. To rešenje se zove Unified physical register file (jedinstveni registarski file). Ideja je da se u njemu čuvaju vrednosti i za in order i za lookahead stanja. Tada se pri komitovanju samo ažuriraju preslikavanja arhitekturalni registri u in order stanju na fizičke registre jedinstvenog registarskog file-a koji čuva te vrednosti. U trenutku nastanka izuzetaka će onda da ostanu zauzeti samo ti fizički registri jedinstvenog file-a



Kratatak opis mašine

Nakon razmatranja osnovnih koncepata za preimenovanje, definisanje argumenata, setovanje ready bita rezultata i argumenata, asocijativno upisivanje vrednosti argumenata i održavanje preciznih izuzetaka, razmotrimo celovito funkcionisanje tag indexed mašine na Slici.



Posmatrajmo prvo ubacivanje instrukcije u ugrađenu dataflow mašinu. Nakon dekodovanja instrukcije, neohodno je uraditi preimenovanje rezultata i argumenata. Za rezultat se koristi tag slobodnog fizičkog registra iz Free Tag FIFO. Pretpostavimo instrukciju sa dva argumenta. Za argumente se koristi simultano preslikavanje iz arhitekturalnih adresa oba argumenta u fizičke registre koji su dodeljeni na osnovu arhitekturalnog stanja u tom trenutku. Time se uspostavljaju direktne prave zavisnosti po podacima i

praktično formira dinamički graf zavisnosti po podacima. Funkciju preslikavanja radi Remap (Rename) file koji čuva preslikavanja adresa za ažurno arhitekturalno stanje. Svaka nova instrukcija koja je takvog tipa da generiše rezultat u registru (nije store, cmp ili grananje) izaziva ažuriranje preslikavanja za arhitekturalni registar njenog rezultata u remap file-u. Uneta instrukcija se odmah potom pamti u ROB, kako bi se moglo održavati in-order stanje.

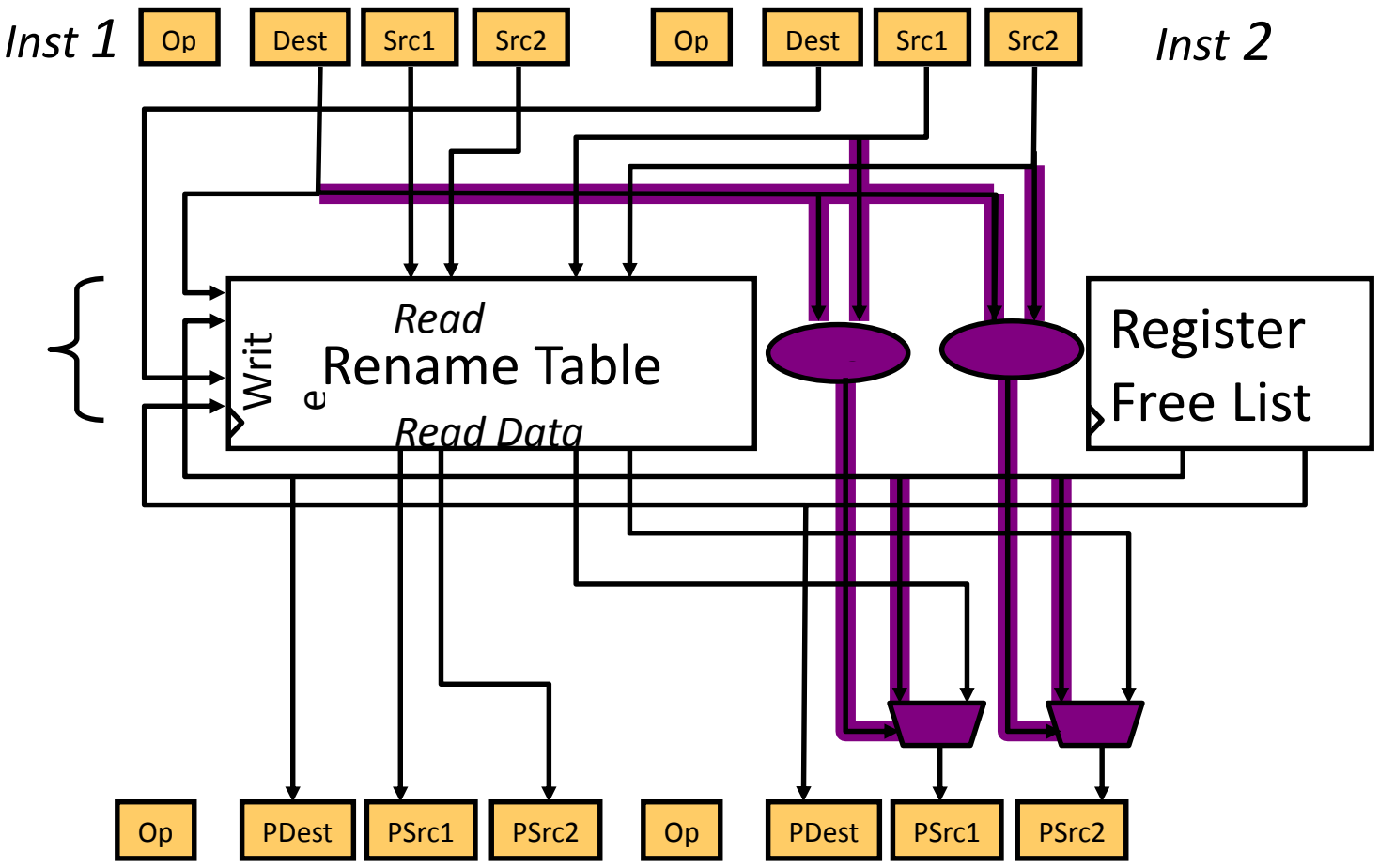
U izvedbi mašine na slici, instrukcije se nakon preimenovanja ubacuju u Issue Queue u kome se nalaze sve instrukcije koje čekaju da argumenti postanu data ready. Na osnovu tipa instrukcija i funkcionalne jedinice koja može da izvrši instrukciju, može postojati više issue queue. One instrukcije kojima svi argumenti postanu data ready odlaze iz Issue Queue na izvršavanje i nakon nekoliko ciklusa će rezultat da se upisuje u fizički registar određen tagom za rezultat. Tada se za taj fizički registar setuje ready bit i inicira se asocijativno pretraživanje u Issue Queue sa dodelom vrednosti svim odgovarajućim argumentima i setovanjem njihovih ready bita. Tako nove instrukcije u Issue Queue mogu da postanu data ready.

Paralelno se obavlja i ažuriranje vrednosti u ROB, za instrukciju koja je generisala rezultat i ukoliko je potrebno, ažuriranje in order stanja.

Superskalarni rad

Već je napomenuto da svaka nova instrukcija koja upisuje u registar ažurira remap file sa arhitekturalnim stanjem. Kada se samo jedna instrukcija ubacuje po ciklusu, remap file mora da ima dva porta za čitanje adresa preimenovanih argumenata i jedan port za upis. Tako za svaku novu instrukciju koja se ubacuje u ugrađenu dataflow mašinu postoji ažurno preslikavanje adresa (preslikavanje) koje obavlja Remap file. Međutim, ako se ubacuje više instrukcija (paket) u istom ciklusu u instrukcijski prozor, javlja se problem kada postoji zavisnost po podacima između instrukcija iz paketa. Tada Remap file ima ažurno preslikavanje samo za prvu novu instrukciju, a za ostale instrukcije iz paketa se mora obaviti provera pravih zavisnosti po podacima između njih.

Na slici je pokazana logika za slučaj simultanog ubacivanja dve instrukcije u prozor. Tada druga instrukcija može da ima argument(e) koji je rezultat prve instrukcije koja se ubacuje u prozor. Preimenovanje takvog argumenta mora da se obavi tako da tag rezultata prve instrukcije uzet iz Free Tag FIFO-a mora odmah da postane i tag zavisnog argumenta druge instrukcije, umesto da se koristi remap file. Remap file mora u ovom slučaju da ima 4 simultana čitanja i dva simultana upisa.



Određivanje da li postoji zavisnost između te dve instrukcije se obavlja komparacijom adrese arhitekturalnog registra rezultata za prvu instrukciju i adresa arhitekturalnih registara oba argumenta naredne instrukcije pomoću dva komparatora. Ukoliko komparatori pokažu jednakost za neki ili oba argumenta, to znači da postoji direktna prava zavisnost po podacima i preimenovanje za te argumente druge instrukcije u tom slučaju ne obavlja Rename (Remap) file. Tada slobodni tag dodeljen rezultatu prve instrukcije postaje i tag argumenta druge instrukcije zavisne po podacima. Multiplekseri upravljani rezultatom komparacije određuju ko definiše fizičku adresu svakog od argumenata druge instrukcije.