Tomasulo algoritam bez preciznih izuzetaka

Bazični Tomasulo algoritam je prva primena ideje ugradnje *data flow* mašine za ograničen broj instrukcija (instrukcijski prozor) u procesor, a da se pritom eliminišu antizavisnosti i izlazne zavisnosti. Ugrađena *data flow* mašina treba da funkcioniše tako da izračunavanje rezultata neke instrukcije dovede do toga da sve instrukcije u instrukcijskom prozoru, koje su zavisne po podacima od te instrukcije, postanu *Data Ready* po izračunatom argumentu. Označimo instrukciju koja generiše rezultat sa Ir, a skup svih instrukcija zavisnih po podacima od Ir sa IrDDSet (*data dependent set of Ir*). Instrukcije iz IrDDSet postaju *Data Ready* po izračunavanju Ir akko je:

1. Imaju samo jedan argument
2. Drugi argument je već bio *Data Ready*

Rezultat Ir se mora nekako distribuirati do svih instrukcija iz IrDDSet. Osim toga, kod ugrađenih *data flow* mašina za sve instrukcije mora da se u hardveru obezbedi logika koja vodi računa o tome kada svaki od argumenata postaje *Data Ready* i kada ona sama postaje *Data Ready*.

Tomasulo je spojio ideju distribucije rezultata Ir sa osobinom hardvera da je neki rezultat najbolje poslati - emitovati preko *data bus*-a do svih primalaca rezultata, jer to zahteva najmanje hardvera i kontrole. Ako bi sve instrukcije IrDDSet bile nekako nakačene na zajednički *data bus*, one bi u paraleli mogle da pročitaju rezultat Ir koji se emituje na zajednički *data bus* i da ga zapamte kao svoj *Data Ready* argument. Naravno da bi u tom konceptu sve instrukcije u instrukcijskom prozoru morale da budu nakačene na taj zajednički *data bus*. Zato instrukcije primaoci argumenta - IrDDSet moraju da prepoznaju podatak na osnovu jedinstvene identifikacije instrukcije Ir koja generiše rezultat, dok kod ostalih instrukcija u prozoru neće biti promena, jer ne prepoznaju emitovani rezultat kao svoj argument.

Zajednički *data bus*

Na osnovu prethodnog, potrebno je da zajednički *data bus* bude realizovan tako da se pored rezultata operacije Ir šalje i jedinstvena identifikacija instrukcije koja emituje. Taj identifikator instrukcije mora da bude jedinstven na nivou instrukcijskog prozora, dokle god rezultat instrukcije nije izračunat i emitovan na zajednički *data bus*. Kada se emituje, očekuje se da će sve zavisne instrukcije da pokupe rezultat i posle toga instrukcija prestaje da bude deo dinamičkog DDG ugrađene *data flow* mašine.

Sve instrukcije koje se nalaze u ugrađenoj *data flow* mašini (ceo instrukcijski prozor) treba da budu nekako povezane na zajednički *data bus*, kako bi prepoznavale svoje argument, sve dok ne postanu *Data Ready*. To se obavlja preko hardvera nazvanog rezervacione stanice, pri čemu jednoj instrukciji u instrukcijskom prozoru odgovara jedna rezervaciona stanica (korespodencija 1:1). Uloga rezervacione stanice je da poredi identifikaciju instrukcije koja emituje na zajednički *data bus* sa identifikacijama za argumente svoje instrukcije koji nisu *Data Ready*. Ako je neka od identifikacija argument identična, treba

da pokupi podatak sa zajedničkog *data bus*-a i da ga zapamti kao svoj odgovarajući argument. Ujedno, rezervaciona stanica treba da označi da taj argument postaje *Data Ready*. Osluškujući zajednički *data bus*, instrukcije u rezervacionim stanicama postaju *Data Ready* i idu na izvršavanje, a kada završe izvršavanje, šalju svoj rezultat na zajednički *data bus*, zajedno sa identifikatorom rezervacione stanice u kojoj se instrukcija nalazila. Onog momenta kada je instrukcija generisala rezultat i poslala identifikaciju svoje rezervacione stanice na zajednički *data bus*, oslobađa se njena rezervaciona stanica za prijem novih instrukcija u instrukcijski prozor. Tako se postepeno izvršava tekući dinamički graf zavisnosti po podacima i oslobađaju rezervacione stanice za prijem novih instrukcija u instrukcijski prozor. U procesu napuštanja grafa ugrađene *data flow* mašine – oslobađanja rezervacionih stanica i dodavanja novih instrukcija u graf – zauzimanja slobodnih rezervacionih stanica, održava se dinamički DDG ugrađene *data flow* mašine približno konstantne veličine. On je ograničen pre svega ukupnim brojem rezervacionih stanica u hardveru. Uslov za ulazak nove instrukcije u instrukcijski prozor je da se nalazi na predviđenom dinamičkom tragu, da su sve instrukcije pre nje na tragu već ubačene u instrukcijski prozor i da postoji slobodna odgovarajuća rezervaciona stanica za prihvat te instrukcije u instrukcijski prozor. Prethodni stav važi ako ubacujemo instrukcije u prozor jednu po jednu. Naravno da u opštem slučaju može da se ubaci više instrukcija sa dinamičkog traga – paket instrukcija po ciklusu, a tada za paket mora da važi isti stav kao i za pojedinačne instrukcije. Taj slučaj se u ovom poglavlju neće razmatrati, a najvažnija mana prethodnog razmatranja je ograničeni paralelizam, jer se generiše samo jedan rezultat na zajedničkom *data bus*-u po ciklusu.

Korespodencija instrukcija-rezultat-rezervaciona stanica

Prilikom prikazivanja osnovnog koncepta Tomasulo algoritma, spominjana je samo apstrakcija identifikator instrukcije. Pre ubacivanja instrukcija u prozor, postoji originalni redosled sekvencijalnog izvršavanja dinamičkog traga koji određuje prediktor grananja. Kôd instrukcija sadrži arhitekturalne registre za argumente i rezultate i kôd operacije koje treba prilagoditi ugrađenoj *data flow* mašini. Za kôd operacije treba dekodovanjem generisati odgovarajući mikrokôd, kao i za mašinu bez instrukcijskog prozora. Međutim, oslonac na arhitekturalne registre u utvrđivanju zavisnosti bi sprečavao eliminaciju antizavisnosti i izlaznih zavisnosti. Zato se mora raditi preimenovanje argumenata i rezultata i sve to uskladiti sa osnovnim konceptom zajedničkog *data bus*-a i jedinstvenog identifikatora instrukcija.

Formiranje dinamičkog grafa zavisnosti po podacima se mora raditi na osnovu predviđenog dinamičkog traga, a instrukcije su naređane na tom tragu pre ulaska u prozor u ispravnom sekvencijalnom redosledu koji nazivamo programskim redosledom. Ubacivanje instrukcija u prozor mora da se obavlja u programskom redosledu - in-order, jer se svako formiranje grafa zavisnosti po podacima mora oslanjati na taj jedan ispravni sekvencijalni redosled. Uostalom, tako se formiraju DDG i u vreme prevođenja za bazične blokove. Prilikom ubacivanja instrukcija u prozor mora da se uradi preimenovanje argumenata i rezultata kako bi se jednoznačno definisale prave zavisnosti po podacima. Ovo je neophodno zbog ponovnog korišćenja registara za upis u programima, što će se posebno javljati kada je instrukcijski prozor veći od broja arhitekturalnih registara. Ujedno, eliminacija antizavisnosti i izlaznih zavisnosti zagarantovana je ako se radi preimenovanje za svaki rezultat instrukcije koja ulazi u instrukcijski prozor. Tako se uspostavlja korespodencija 1:1:1 između instrukcija, preimenovanog rezultata i rezervacione

stanice u kojoj će se nalaziti instrukcija. Kako je rezultat poznat tek na kraju kada je izračunat, a instrukcija nema više adresu iz memorije kada je već ubačena u instrukcijski prozor, ostaje da jedini oslonac za jedinstvenu identifikaciju instrukcije Ir u ugrađenoj *data flow* mašini bude adresa rezervacione stanica u kojoj će se ona nalaziti do izračunavanja rezultata. Ta jedinstvenost važi samo do trenutka dok se na zajednički *data bus* ne emituje rezultat Ir, čime se omogućava svim IrDDSet instrukcijama da pokupe rezultat Ir. Dakle, adresa rezervacione stanice (instrukcije) koja šalje rezultat na zajednički *data bus* predstavlja jedinstveni identifikator za sve ostale instrukcije u rezervacionim stanicama koje svakog ciklusa proveravaju da li je sadržaj na zajedničkom *data bus-u* neki od njihovih argumenata. Na osnovu toga, rezultat instrukcije se praktično implicitno preimenuje u adresu rezervacione stanice koja će emitovati taj rezultat na zajednički *data bus*.

Preimenovanje argumenata

U trenutku ubacivanja instrukcije u instrukcijski prozor, potrebno je obaviti pravilno preslikavanje adresa registara za argumente na izvore argumenata preko jedinstvenih identifikatora instrukcija koje proizvode te argumente. Za sada ćemo pretpostaviti da argumenti instrukcije za koju se radi preslikavanje argumenata nisu već izračunati. Kako su jedinstveni identifikatori instrukcija i rezultata instrukcija adrese rezervacionih stanica u kojima se nalaze te instrukcije, umesto adresa arhitekturalnih registara za argumente moraju da se pojave adrese odgovarajućih rezervacionih stanica (instrukcija) koje generišu argumente. Još ostaje da se definiše koje su odgovarajuće rezervacione stanice.

Unutar instrukcijskog prozora, jedan arhitekturalni registar rezultata se može preslikati na više rezervacionih stanica u kojima se generišu rezultati koji su preslikane vrednosti tog arhitekturalnog registra. Koje preslikavanje je relevantno za argumente instrukcije koja se ubacuje? To je očigledno preslikavanje rezultata poslednje takve instrukcije ubačene u instrukcijski prozor koja po sekvencijalnom tragu radi upis u odgovarajući arhitekturalni registar za argument. Drugačije rečeno, za sve registre argumenata važi da se preslikavanje mora raditi na osnovu **arhitekturalnog stanja** tih registara (i odgovarajućih instrukcija u arhitekturalnom stanju) u trenutku ubacivanja nove instrukcije u instrukcijski prozor. Sa ubacivanjem instrukcije u prozor se menja arhitekturalno stanje, jer se pravi novo preslikavanje za arhitekturalni registar u koji se radi upis njenog rezultata. Izuzetak su instrukcije koje ne rade upis, npr. CMP (compare) koje samo generišu flag i instrukcija store koja nema rezultat unutar instrukcijskog prozora. Očigledno je da u hardveru kojim se izvršava Tomasulo algoritam mora postojati tabela preslikavanja adresa arhitekturalnih registara argumenata u lokacije rezervacionih stanica instrukcija u arhitekturalnom stanju koje treba da izračunaju te argumente. Ovo preimenovanje je implicitno, jer ne postoji registar u nekoj registarskoj memoriji koji zauzimamo zbog preimenovanja, već nam adresa rezervacione stanice služi kao referenca za granu zavisnosti po podacima. Tačnije, u dinamičkom grafu zavisnosti po podacima će referenca na rezervacionu stanicu sa instrukcijom Ir označavati sve zavisnosti po podacima (grane) u dinamičkom DDG koji polaze iz te instrukcije (rezervacione stanice zbog 1:1) i završavaju u svim instrukcijama iz IrDDSet skupa u prozoru. Važno je uočiti da su za rad ugrađene *data flow* mašine bitne samo informacije kada su neke instrukcije *Data Ready*, nije nam u hardveru potreban jedinstveni identifikator svake grane dinamičkog DDG, već je dovoljan identifikator za IrDDSet!

Prethodna analiza je podrazumevala da se sve instrukcije koje čekaju neki rezultat (IrDDSet skup) već nalaze u instrukcijskom prozoru. U tom slučaju nam nisu potrebni nikakvi registri za pamćenje rezultata, jer se rezultati odmah bez kašnjenja prosleđuju preko zajedničkog *data bus*-a do SVIH rezervacionih stanica koje koriste taj rezultat. Da li se sve instrukcije iz IrDDSet skupa nalaze u prozoru se može utvrditi na osnovu toga da li je instrukcija Ir koja generiše rezultat u arhitekturalnom stanju. Ako nije u arhitekturalnom stanju, sve instrukcije iz IrDDSet skupa su već u rezervacionim stanicama, pa će SVE instrukcije korisnici rezultata bez posredstva registara prihvatiti rezultat Ir sa zajedničkog *data bus-a* kada bude emitovan. To nije ništa drugo nego *forwarding* rezultata do svih instrukcija kojima je taj rezultat argument.

Ako je instrukcija još uvek u arhitekturalnom stanju u trenutku kada je emitovan njen rezultat na zajednički *data bus*, nove instrukcije koje tek treba da se ubace u prozor, a treba da koriste taj rezultat kao argument, su "propustile" trenutak da pokupe svoj argument sa zajedničkog *data bus*-a. Za taj slučaj je neophodno da se sadržaj zajedničkog *data bus*-a zapamti u registarskom fajlu iste veličine kao što je arhitekturalni registarski fajl. Preko arhitekturalnog stanja može da se uradi obrnuto preslikavanje sa adrese rezervacione stanice na originalni arhitekturalni registar za rezultat. U tom registarskom fajlu se pamte isključivo nepreimenovane izračunate vrednosti unutar instrukcijskog prozora (što će biti retko za veće instrukcijske prozore), pa ga možemo nazvati registarskim fajlom nepreimenovanih *Data Ready* vrednosti. Kada se instrukcija koja treba da koristi rezultat iz registarskog fajla nepreimenovanih *Data Ready* vrednosti bude ubacivala u instrukcijski prozor, u trenutku prebacivanja u rezervacionu stanicu će se pokupiti i odgovarajuća vrednost argumenta korišćenjem adrese arhitekturalnog registra i čitanjem odgovarajućeg sadržaja iz registarskog fajla nepreimenovanih izračunatih vrednosti.

Izmena sadržaja lokacija registarskog fajla nepreimenovanih *Data Ready* vrednosti se obavlja tako da se svaki registar tog fajla posmatra kao da ima svoju rezervacionu stanicu koja ne generiše podatke na zajednički *data bus*, ali prepoznaje rezervacionu stanicu koja bi trebalo da upisuje rezultat u registar. Kod ovih specifičnih rezervacionih stanica se vrednost izvora koji treba da emituje podatak može promeniti sa izmenom arhitekturalnog stanja u svakom ciklusu. Zbog toga će se u većini slučajeva, za veći instrukcijski prozor, događa da se ne upisuju podaci za neki identifikator, jer se izmeni identifikator za prepoznavanje (instrukcija u arhitekturalnom stanju), pre nego što je emitovan sadržaj na zajednički *data bus* za prethodni identifikator. To je upravo slučaj kada sve instrukcije iz IrDDSet-a pokupe rezultat sa zajedničkog *data bus*-a.

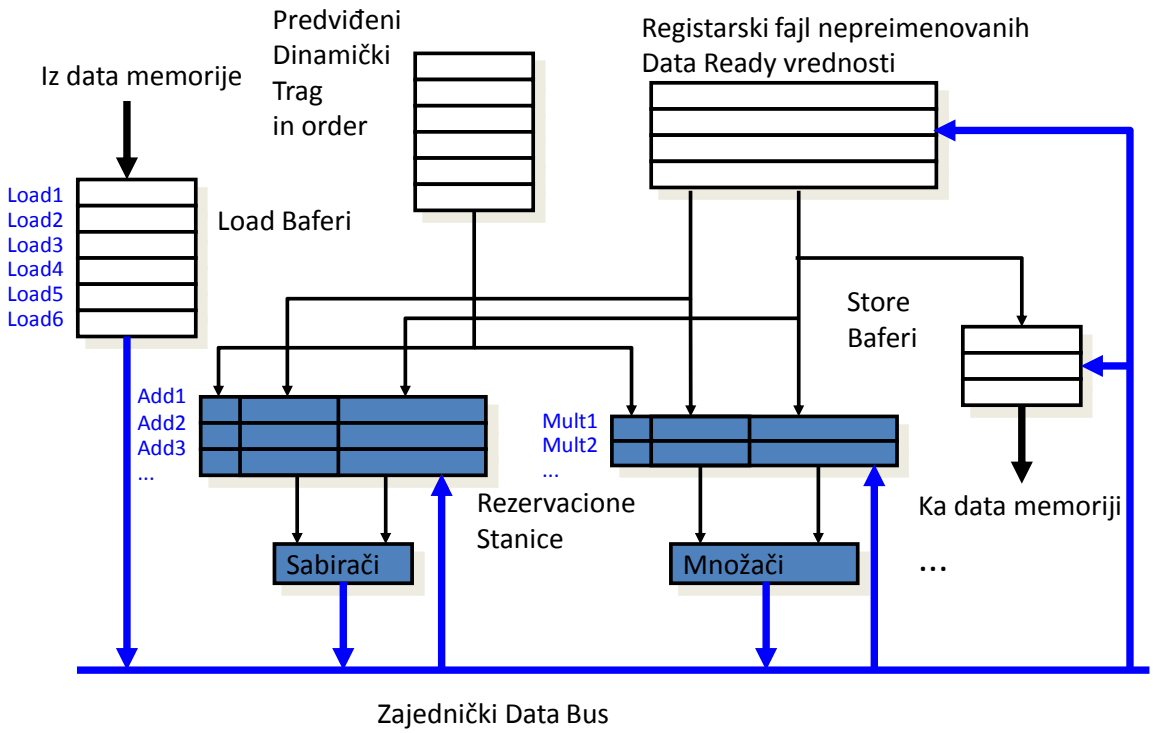
Drugi način da se obezbedi ažurna vrednost registarskog fajla nepreimenovanih *Data Ready* vrednosti je da se obezbedi da noviju preimenovanu vrednost za neki arhitekturalni registar ne može prepisati starija preimenovana vrednost za isti arhitekturalni registar. To se može dogoditi u principu zbog *out of order* izvršavanja instrukcija. U hardveru se zato mora voditi u evidenicija o *in order* upisima, posebno za svaki registar registarskog fajla nepreimenovanih *Data Ready* vrednosti. Pri takvoj realizaciji upisa u registarski fajl nepreimenovanih *Data Ready* vrednosti, treba istaći da evidencija nije vezana za ukupno *in order* izvršavanje instrukcija, već samo za upise svakog arhitekturalnog registra posebno.

Realizacija u hardveru

Osnovni blokovi u hardveru su prikazani na Sl. Xx. Instrukcijski prozor se popunjava sa instrukcijama koje se nalaze na predviđenom dinamičkom tragu u *in order* redosledu. U trenutku ubacivanja u prozor, instrukciji treba da se dodeli slobodna rezervaciona stanica za odgovarajuću funkcionalnu jedinicu. Da bi se znalo koja je odgovarajuća funkcionalna jedinica, mora se dekodovati operacija. Rezervacione stanice su hardverski pridružene funkcionalnim jedinicama da se ne bi komplikovao hardver potrebom za mapiranjem rezervacionih stanica na odgovarajuću funkcionalnu jedinicu. Broj rezervacionih stanica po svakom tipu funkcionalnih jedinica i broj samih funkcionalnih jedinica treba da bude prilagođen statistikama "prosečnog programa". Dakle, uslov za ubacivanje nove instrukcije u instrukcijski prozor je postojanje slobodne rezervacione stanice odgovarajuće funkcionalne jedinice. Broj instrukcija u instrukcijskom prozoru u svakom trenutku jednak je ukupnom broju rezervacionih stanica umanjenim za ukupan broj slobodnih rezervacionih stanica. Kako dinamički DDG instrukcijskog prozora ima u proseku veći paralelizam ukoliko je broj instrukcija u instrukcijskom prozoru veći, logika mašine treba da bude da se u instrukcijski prozor ubaci što više instrukcija. Naravno, kada naiđe instrukcija koja traži rezervacionu stanicu za funkcionalnu jedinicu koja je njoj potrebna, a nema takve slobodne rezervacione stanice, mora se zakočiti ubacivanje instrukcija u prozor, sve do trenutka dok se ne oslobodi odgovarajuća rezervaciona stanica.

U dosadašnjim analizama nisu razmatrane specifičnosti *Load* i *Store* instrukcija. *Load* instrukcije uzimaju argumente iz memorije i zasada će se smatrati da su ti argumenti uvek *Data Ready*. Naravno *Load* instrukcija će morati da dobije odgovarajuću rezervacionu stanicu i kada se pročita memorija, pročitana vrednost će se emitovati na zajednički *data bus*. Naravno, po svakom ciklusu će morati da postoji arbitriranje ako postoji više izračunatih rezultata instrukcija koji treba da emituju na zajednički *data bus*. *Store* instrukcija je karakteristična po tome što se njen rezultat ne javlja na zajedničkom *data bus*u, jer se pamti u data memoriji, a rezervacione stanice za *Store* treba samo da prepoznaju kada je na zajednički *data bus* emitovana vrednost koja treba da se pamti u memoriji.

Tomasulo Organizacija



Sl. Xx Elementi hardvera ugrađene data flow mašine kod osnovnog Tomasulo rešenja

Na Sl. Xx je pokazano da se sadržaj iz registarskog fajla nepreimenovanih *Data Ready* vrednosti dovodi do rezervacionih stanica posebnim magistralama. Alternativa tom rešenju je da na kraju FIFO-a sa predviđenim dinamičkim tragom bude logika koja odmah generiše inicijalni sadržaj rezervacionih stanica, tako da se odmah ubacuju *Data Ready* vrednosti iz registarskog fajla nepreimenovanih *Data Ready* vrednosti.

Na Sl. xx nije prikazana logika za preimenovanje argumenata. Za preimenovanje služi tabela statusa rezultata registara. Ona definiše arhitekturalno stanje i samim tim implicitno preimenovanje argumenata. Ta tabela za svaki arhitekturalni registar sadrži rezervacionu stanicu koja će poslednja po programskom redosledu da radi upis u taj arhitekturalni registar. To nije ništa drugo nego predstava arhitekturalnog stanja preko rezervacionih stanica. U slučaju da nijedna rezervaciona stanica neće da radi upis u neki arhitekturalni registar, mora da postoji indikacija da se argument mora tražiti u registarskom fajlu nepreimenovanih *Data Ready* vrednosti. Jedno od rešenja je da imamo 2^n-1 rezervacionih stanica, da se adresiraju sa n bita i da preostali adresni kôd, npr. 00..0 označava da se argument nalazi izračunat u registarskom fajlu nepreimenovanih *Data Ready* vrednosti.

Preostale tabele su vezane za sadržaj rezervacionih stanica i stanja instrukcija. Instrukcije u prozoru mogu da budu u stanjima izdavanja (ulaska u prozor i zauzimanja rezervacione stanice), izračunavanja i upisa rezultata.

Rezervacione stanice imaju sledeći sadržaj:

1. Operacija koja treba da se izvrši (napomena: funkcionalna jedinica je već izabrana), npr. integer + ili AND bit po bit kada je već rezervaciona stanica pridružena integerskoj instrukciji
2. *Vj*, *Vk*: Vrednosti argumenata ako su već izračunate
3. *Qj*, *Qk*: Rezervacione stanice koje proizvode argumente *Vj* i *Vk* (vrednost koja treba da bude upisana). Ako je *Qj* ili *Qk*=0, to označava da je odgovarajuća vrednost argumenta izračunata i već upisana u polja *Vj* i *Vk* rezervacione stanice. Tako nema *Ready flag*-a, a instrukcija postaje *Ready* kada su i *Qj* i *Qk* jednaki 0. *Load* baferi nemaju ni *Qj* ni *Qk*, jer ne čekaju nijedan argument izračunat u instrukcijskom prozoru, ali imaju *Vj* u kome pamte vrednost iz memorije i kada *Load* instrukcija dođe u fazu upisa rezultata (*write*), emituju na zajednički *data bus*. *Store* baferi imaju samo *Qj* u svojoj rezervacionoj stanici, jer po definiciji operacije *Store* postoji samo jedan argument. Ako neka operacija ima samo jedan argument, a nije *Store* ili *Load*, definiše se u rezervacionim stanicama da je *Qk*=0, sadržaj *Vk* je nebitan i instrukcija postaje *Data Ready* kada *Qj* postane jednak 0.
4. Busy bit označava da je rezervaciona stanica zauzeta. Rezervaciona stanica ostaje zauzeta sve dok se ne završi emitovanje rezultata na zajednički *data bus*. Ukoliko u dve ili više rezervacionih stanica povezanih na istu funkcionalnu jedinicu u istom ciklusu postanu *Data Ready*, na ulazu u tu funkcionalnu jedinicu se mora uspostaviti arbitriranje (prioritiranje) između rezervacionih stanica (instrukcija) kojimse određuje koja od tih instrukcija počinje izvršavanje.

Način obrade svake instrukcije

Instrukcije se po redosledu izvršavanja na predviđenom dinamičkom tragu ubacuju u instrukcijski prozor. Kada Instrukcija dođe na red za ubacivanje, uradi se prvo njeno dekodovanje. Na osnovu kôda operacije se radi preslikavanje na funkcionalnu jedinicu i za tu funkcionalnu jedinicu se traži slobodna rezervaciona stanica. Ako takva stanica postoji, nastavlja se postupak ubacivanja u prozor, a rezervaciona stanica obeleži kao zauzeta. Polja *Qj* i *Qk* zauzete rezervacione stanice za izvor argumenata se dobijaju istovremenim preslikavanjem obe adrese arhitekturalnih registara za argumente. To se radi korišćenjem Tabele statusa rezultata registara, na adrese rezervacionih stanica koje će generisati rezultat, ukoliko vrednosti nisu 0. Ukoliko je neka od vrednosti u tabeli statusa registara rezultata 0, učitava se vrednost već ranije izračunatog argumenta iz registarskog fajla nepreimenovanih *Data Ready* vrednosti, korišćenjem adrese odgovarajućeg arhitekturalnog registra. Sa celokupnim sadržajem kôda operacije, izvora argumenata i eventualno vrednosti argumenata se popuni izabrana, ranije slobodna, a sada zauzeta, rezervaciona stanica i instrukcija postaje deo instrukcijskog prozora. Istovremeno se mora za

arhitekturalni registar u koji se upisuje rezultat instrukcije u Tabeli statusa rezultata registara uraditi novo preimenovanje na upravo popunjenu rezervacionu stanicu, čime menjamo arhitekturalno stanje.

Rezervaciona stanica sa instrukcijom komparira svoja polja *Qj* i *Qk* sa adresom rezervacione stanice koja emituje rezultat na zajednički *data bus* svakog ciklusa i kada se utvrdi jednakost, tada se učitava argument za koji je utvrđena jednakost, a odgovarajuća vrednost *Qj* ili *Qk* koja je bila jednaka postaje 0. Sada vrednost 0 označava da je argument *Data Ready.* Rezervaciona stanica, kada internom logikom utvrdi da su i *Qj* i *Qk* jednaki 0 proglašava rezervacionu stanicu (instrukciju) *Data Ready*. Ako dve ili više rezervacionih stanica povezanih na istu funkcionalnu jedinicu postanu u istom ciklusu *Data Ready*, jednostavnom logikom arbitriranja se određuje samo jedna koja započinje izvršavanje. Rezervaciona stanica koja je *Data Ready* i koja je dobila funkcionalnu jedinicu dostavlja oba argumenta i kod operacije funkcionalnoj jedinici. U pipeline funkcionalne jedinice se ubacuje i kôd (adresa) rezervacione stanice instrukcije koja se izvršava. Kada instrukcija dođe u fazu upisa, pokušava da dobije zajednički *data bus*. Kada ga dobije, emituje na zajednički *data bus* vrednost rezultata i kôd (adresu) rezervacione stanice instrukcije koja je upravo generisala rezultat. Tada sve rezervacione stanice koje sadrže instrukcije iz IrDDSet-a prepoznaju da je emitovan njihov argument i unose emitovanu vrednosti u odgovarajuće V polje svojih rezervacionih stanica, menjajući odgovarajuća Q polja na 0. Eventualno se radi upis u registarski fajl nepreimenovanih *Data Ready* vrednosti, ako je potrebno, a prema pravilima opisanim ranije. U narednom ciklusu se oslobađa rezervaciona stanica koja je emitovala i naravno zajednički *data bus*. Na taj način se završava izvršavanje instrukcija i oslobađaju rezervacione stanice za nove instrukcijee koje se ubacuju u ugrađenu *Data Flow* mašinu.

Poboljšanja Tomasulo algoritma

Osnovna poboljšanja su vezana za sledeće elemente: Povećanje broja zajedničkih *data bus*-eva, završavanje *in order* čime se postižu precizni prekidi i razrešavanje zavisnosti po podacima preko memorije, što smo u dosadašnjem izlaganju zanemarili kao mogućnost.