# SADRŽAJ

# 1. PROTOČNOST

## 1.1 Osnovne tehnike za eksploataciju paralelizma

Kod današnjih računara postoje sledeće dve osnovne tehnike za eksploataciju paralelizma:

**A. protočnost** (*pipelining*)
**B. umnožavanje** (*replication*)

**A. Protočnost:** Sa ciljem da se obavi izračunavanje, kaskadno se povezuje veći broj funkcionalnih jedinica. Ovaj način obrade sličan je fabričkoj liniji za asembliranje nekog proizvoda, kakav je recimo TV aparat, putničko vozilo i dr. Svaka funkcionalna jedinica odgovara odredjenom stepenu izračunavanja, a svako izračunavanje prelazi kroz ceo protočno organizovan sistem. Za slučaj da se obavlja samo jedno izračunavanje protočni sistem ne može da izdvoji paralelizam. No u slučajevima kada se isto izračunavanje obavlja više puta, tada se izračunavanja u funkcionalnim jedinicama mogu preklapati.

Neka protočni sistem čine $n$ funkcionalnih jedinica (stepena) i neka je $T$ vreme potrebno najsporijem stepenu da izvrši svoju funkciju. Pod ovim uslovom novo izračunavanje može da započne svakog $T$-tog trenutka. Protočni sistem je popunjen kada sve funkcionalne jedinice obavljaju različito izračunavanje. Nakon što se protočni sistem popuni, po jedan rezultat (izračunavanje) se dobija svakog $T$-tog trenutka. Saglasno prethodnom, kada je protočni sistem pun dobija se pojačanje $n$ pod uslovom da se zanemare granični efekti koji odgovaraju punjenju (karakteristično za početak) i pražnjenju (tipično za kraj izračunavanja) protočnog sistema.

Na osnovu prethodne diskusije jasno se uočava da je protočnost veoma moćna tehnika za ubrzanje izvršenja sličnih izračunavanja u nizu, pa se zbog toga često koristi kod savremenih paralelnih arhitektura.

Kada su u pitanju paralelni sistemi protočnost se može koristiti na sledeća dva nivoa:

a) **na nivou procesora**: mikro-nivo
b) **izmedju procesora ili čvorova**: makro-nivo

Klasični primeri procesora koji koriste protočnu obradu na mikro-nivou su vektor-procesori, skalarni procesori, superprotočni i superskalarni procesori, VLIW procesori, i dr.

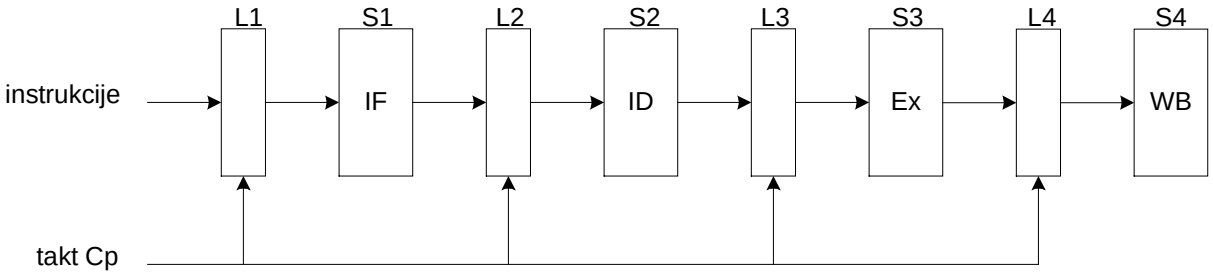
Tipični reprezenti procesora koji koriste protočnu obradu na makro-nivou su komunikacioni procesori, MIMD mašine i dr.

**B. Umnožavanje:** drugi prirodan način da se uvede paralelizam kod računara je da se umnože funkcionalne jedinice. Umnožene (replicirane) funkcionalne jedinice mogu istovremeno da obavljaju istu operaciju nad onoliko različitih podataka koliko postoje različite funkcionalne jedinice. Svi MIMD računari, kao VLIW i superskalarni procesori da bi ostvarili paralelizam u radu koriste pristup umnožavanja funkcionalnih jedinica.

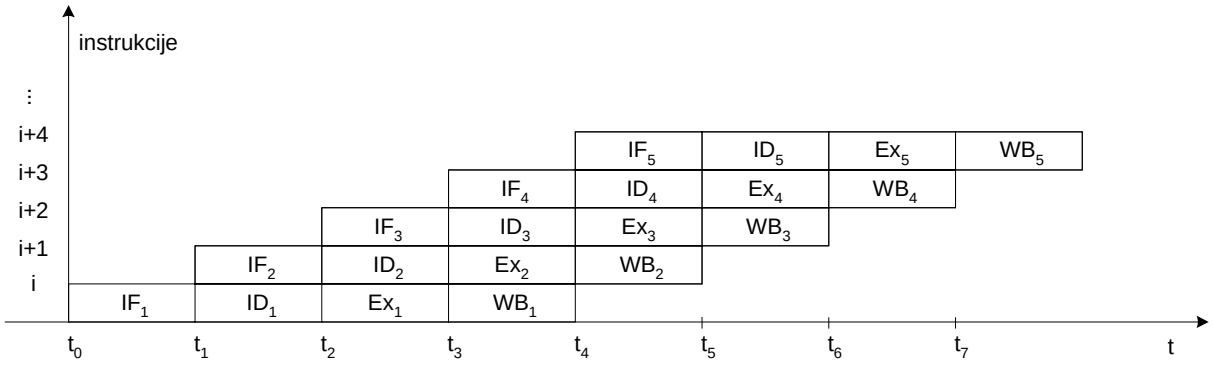
## 1.2 Protočna obrada na nivou instrukcije – osnovni koncepti

Protočnost je standardna hardverska tehnika koja se kod računara koristi za postizanje boljih performansi. Kada se govori o izvršenju instrukcije ono se ostvaruje na taj način što se obrada instrukcije deli na veći broj fiksnih koraka koji se izvršavaju sekvencijalno. Protočnu realizaciju instrukcije čine nekolliko hardverskih stepena (*stages*), *S1,...,Sn*, medjusobno razdvojeni lečevima, $L_1,.....,L_n$. Kako instrukcija prolazi kroz protočni sistem (*pipeline*), hardver svakog stepena obavlja odredjeni tip obrade. Kada instrukcija napusti protočni sistem ona je u potpunosti izvršena.

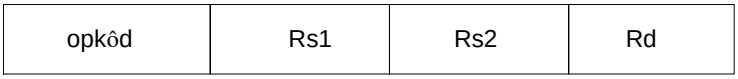
Princip protočne obrade objasnićemo na jednom jednostavno organizovanom procesoru nazvan *FX-4P*. Protočni sistem *FX-4P* u stanju je da obavlja sve *Integer* i *Boolean* operacije, a čine ga protočni stepeni IF, ID, EX i WB (vidi Sliku 1a)).




a). Četvoro-stepeni protočni sistem




b). Princip rada protočnog sistema




c). Tro-adresni format

Slika 1 *FX-4P* protočni sistem

Funkcije koje pojedinačno, u okviru protočnog sistema *FX-4P*, obavljaju protočni stepeni *S1* do *S4* (vidi Sliku 1a)) su sledeće:

(1) S1-**Pribavljanje instrukcije**, IF (*Instruction Fetch*): na osnovu sadržaja programskog brojača, iz adresirane memorijske lokacije pribavlja se instrukcija (format instrukcije je prikazan na Slici 1c))
(2) S2-**Dekodiranje instrukcije**, ID (*Instruction Decode*): u toku ovog ciklusa obavljaju se sledeće dve aktivnosti: (a) dekodira se opkod i inicijalizira proces interpretacije instrukcije; (b) na osnovu sadržaja registarskih specifikatora (polja *Rs1* i *Rs2* koji odredjuju prvi i drugi izvorišni operand instrukcije- vidi Sliku 1c) pribavljaju se (čitaju) operandi iz internog registarskog polja CPU-a (*internal register-file* , tzv. *RF* polje).
(3) S3-**Izvršenje**, EX (*Instruction Execution*): obavlja se operacija specificirana opkôd poljem instrukcije.
(4) S4- **Upis rezultata**, WB (*Result Write Back*): rezultat operacije upisuje se u odredišni registar, specificiran *Rd* poljem sa Slike 1c). *Rd* registar je jedan od registara RF polja.

**Napomena**: Kod procesora *FX-4P*, na osnovu formata sa Slike 1c), uočava se da su oba izvorišna operanda, kao i odredišni (specificirana poljima *Rs1* i *Rs2* i *Rd*, respektivno) registarska. To znači da se ulazne vrednosti operacija dobavljaju iz RF polja, a takodje se i rezultat operacije upisuje u jedan od registara *RF* polja. O načinima kako se vrši punjenje registra *RF* polja (operacija tipa *Load*-register) kao i upis sadržaja registre u memoriju, tj. operacije tipa *Store*-register), ili kako se realizuje operacije tipa *Branch* i druge, govorićemo nešto kasnije.

Izvršenje instrukcije čini sekvenca akcija. Akcije se specificiraju semantikom instrukcija. U prethodnom poglavlju smo ukazali da se jedna akcija može obaviti u toku trajanja jednog ili većeg broja ciklusa. Identifikovaćemo sada karakteristične akcije koje se obično, u toku izvršenja instrukcije, izvršavaju u sekvencijalnom redosledu. Drugim rečima, ukazaćemo na problem **particije**.

Svaki protočni stepen se sastoji od kombinacione logike i/ili veoma brze memorije izvedene u formi registarskog polja ili keš memorije. Stepeni su medjusobno razdvojeni, kako je to prikazano na Slici 1, lečevima, *L1* do *L4*.

Zajedničkim taktnim signalom, *Cp*, vrši se sinhronizacija rada sistema na taj način što svi lečevi istovremeno prihvataju podatke od strane protočnih stepena. Zajedničkim taktnim signalom *Cp* vrši se "pumpanje" (tj. guranje) instrukcija kroz protočni sistem. Na početku procesorskog ciklusa (taktnog perioda) podaci, i/ili upravljačka informacija iz delimično procesirane instrukcije, se pamti (čuva) u leč *Li* ( $i = 1,..,4$). Ovi podaci, i/ili upravljačka informacija, predstavljaju ulaz za logiku stepena *Si* koju leč *Li* direktno pobudjuje. U toku taktnog perioda, signali se propagiraju kroz kombinacionu logiku, ili pristupaju veoma brzim memorijama stepena *Si*. Na kraju taktnog perioda procesirani signali se prihvataju od strane leča $L_{i+1}$.

Kada je protočni sistem dobro projektovan, svi stepeni sadrže logiku sa približno jednakim (izbalansiranim) vremenom propagacije signala. Taktni period treba da bude dovoljno dugačak i da njegovo trajanje odgovara najsporijem stepenu. Protočni stepen treba tako projektovati da početak nove instrukcije (a takodje i kraj instrukcije) koincidira sa početkom svakog novog taktnog perioda (Slika 1b)). Zbog toga, taktni period odredjuje **propusnost** (*throughput*), koja ukazuje na brzinu sa kojom se izvršavaju instrukcije. Vreme koje je potrebno jednoj instrukciji da prodje kroz sve blokove protočnog sistema zove se **latencija** (*latency*). Na primer, ako se protočni sistem sa Slike 2a taktuje sa frekvencijom od 100 MHz, tada je maksimalna propusnost jedna instrukcija na 10ns ili 100 MIPS-a, dok latentnost ovog protočnog sistema iznosi 40ns.

## 1.3 Performansne mere

Osnovna ideja protočne obrade je da se ostvari preklapanje u izvršenju izmedju sukcesivnih instrukcija čime se štedi na vremenu i povećava propusnost. Skoro svi današnji savremeni procesori koriste tehniku protočne obrade čime na jedan indirektan način postižu bolje performanse u odnosu na strogo sekvencijalno izvršenje. Pre nego što ukažemo na neke performansne mere definisaćemo dva pojma koja se odnose na protočni dizajn:

1. **protočni ciklus** (alternativni pojmovi su **procesorski ciklus** ili **taktni period**): perioda taktnog signala, *Cp* kojim se pobudjuje protočni sistem. Perioda je odredjena od najdužeg vremenskog kašnjenja potrebnog da se izvrši jedna operacija kroz neki od protočnih stepena. Tekući protočni procesori operišu sa procesorskim ciklusom koji je reda od 200 ps - 20 ns (tj. 5 GHz – 50 MHz)

2. **Latencija iniciranja instrukcija**: broj procesorskih ciklusa izmedju iniciranja dve susedne instrukcije u programskoj sekvenci.

Od performansnih mera protočnog sistema ukazaćemo na:

a) **broj procesorskih ciklusa** potreban za procesiranje sekvence od *k* instrukcija iznosi

broj_procesorskih_ciklusa = broj_procesorskih_ciklusa_da_se_generiše_prvi_rezultat
+ broj_instrukcija_u_sekvenci - 1
= n + k -1

gde je: *n*-broj protočnih stepeni, a *k* je dužina sekvence instrukcija.

b) **broj procesorskih ciklusa po instrukciji** (*CPI*) iznosi

CPI = (n + k - 1) / k
= 1 + (n - 1) / k

kada $k \rightarrow 1$, tada CPI $\rightarrow n$, a u slučaju kada $k \rightarrow \infty$ , tada CPI $\rightarrow 1$, tj. $1 < CPI < n$.

c) **ubrzanje** (*speedup*) protočnog sistema u odnosu na neprotočni sistem se definiše kao

$S_{pro} = T_{neprotocno} / T_{protocno}$

$= n * k / (n + k - 1)$

to znači da $S_{pro} \rightarrow n$, kada $k \rightarrow \infty$

Važan faktor koga treba uzeti u obzir kod procene performansi predstavlja dodatno kašnjenje koje unosi svaki leč u lancu. U suštini vreme potrebno protočnom stepenu da procesira podatak jednako je zbiru $T_{ci} + T_{ri}$ , gde je $T_{ci}$ vreme procesiranja signala od strane stepena *Si*, a $T_{ri}$ predstavlja kašnjenje koje unosi leč *Li* . Zbog uticaja kašnjenja koje unosi leč ubrzanje realnog protočnog sistema biće dato sledećom relacijom

$S_{pro} = ((n - \alpha) * k)/ (k + n - 1)$

to znači da $S_{pro} \rightarrow n - \alpha$, kada $k \rightarrow \infty$

gde je $\alpha$ prekoračenje usled $T_{ri}$, tj. prekoračenje usled dodatnog vremena, svakog protočnog stepena, potrebnog da se ostvari sinhronizovan rad.

d) **efikasnost** protočnog sistema koji obavlja izračunavanje (ili niz izračunavanja) se definiše kao

$$E_{ideal} = S_{pro} / n$$
$$= (n * k) / ( n^2 + n * k - n)$$

za $n = 5$ i $k = 10$, $E = 0{,}714$; za $n = 5$ i $k = 100$, $E = 0{,}962$; za $n = 5$ i $k = 1000$, $E = 0{,}996$;

## 1.4 Tehnika projektovanja protočnog procesora

Skup instrukcija koje izvršava protočni procesor ima veoma veliki uticaj na njegovu strukturu i funkcionisanje. Imajući ovo u vidu još u fazi projektovanja protočnog procesora mora prvo da se sagleda na koji način se različite klase mašinskih instrukcija preslikavaju, tj. uklapaju, u protočnu strukturu. Klasifikovaćemo instrukcije na osnovu sledećeg kriterijuma: kakav je i gde se ostvaruje tok podataka u toku izvršenja instrukcije. Shodno odabranom kriterijumu moguće je identifikovati sledeća tri tipa instrukcija:

a) *ALU* – aritmetičko/logičke
b) *Load*/*Store* – punjenje registra iz memorije/upis stanja registra u memoriju
c) *Branch* – grananja, mogu biti uslovna i bezuslovna

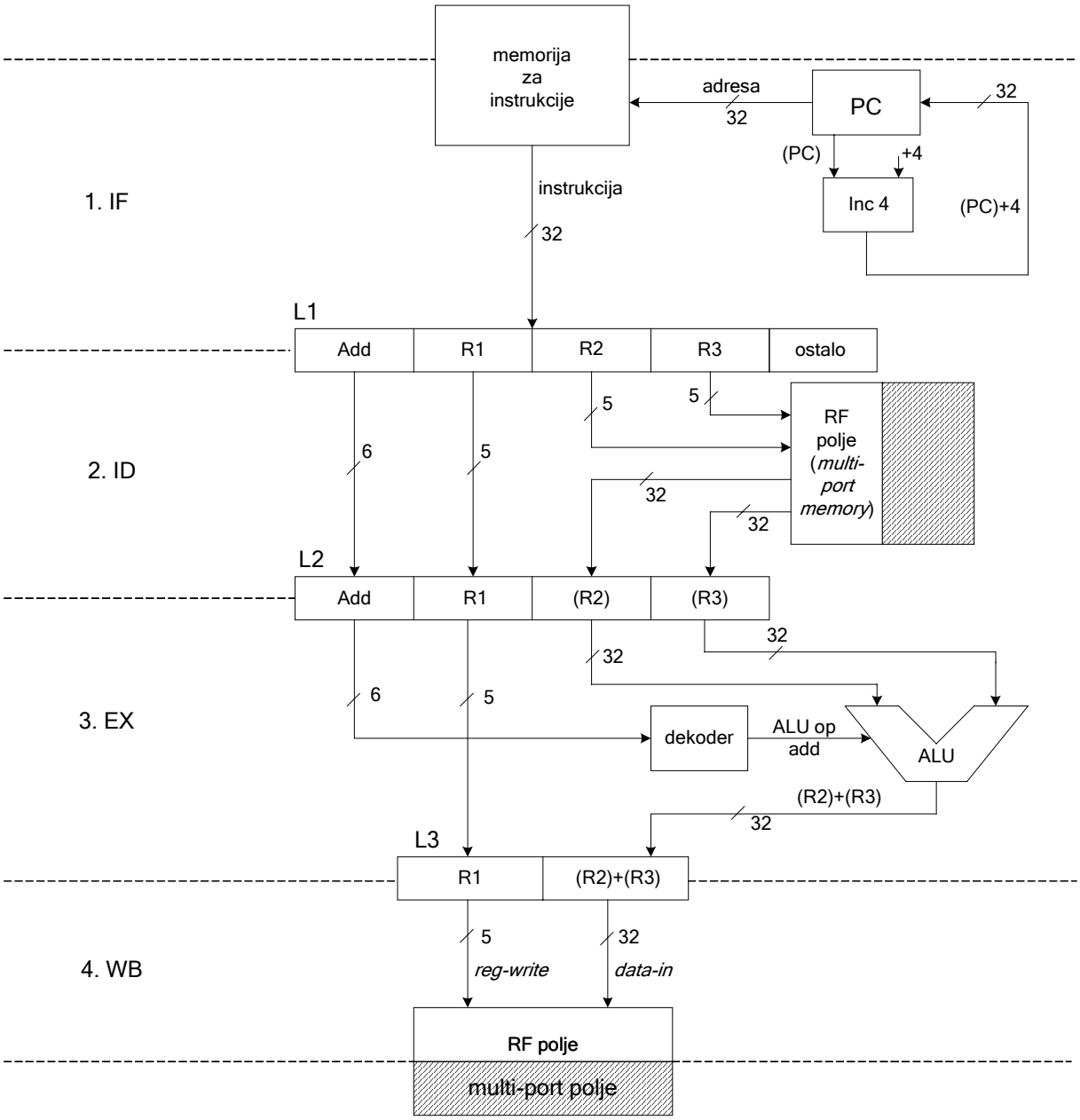
Sa ciljem da se bolje razume ponašanje ovih klasa instrukcija opisaćemo njihov rad na jedan slikovit način. Naime u zavisnosti od preduzetih aktivnosti u toku obrade instrukcije biće prikazana odgovarajuća operacija kao i struktura protočnog sistema neophodna da se obavi ta operacija.

### 1.4.1 ALU instrukcije

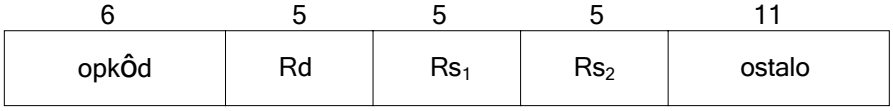
Analiziraćemo prvo kakva treba da bude struktura 32-bitnog procesora kada on izvršava aritmetičke instrukcije tipa *registar-u-registar*. U konkretnom slučaju četvoro-stepeni protočni sistem, koga čine stepeni IF, ID, EX i WB, sličan onom prikazan na Slici 1 (tj. *FX-4P*), ispunjava postavljene zahteve. Detaljnija struktura ovakvog procesora je prikazana na Slici 2. Operacija koja se izvršava je tipa

Add R1, R2, R3 ; (R1) $\leftarrow$ (R2) + (R3)

Format instrukcije *Add* (vidi Sliku 2b)) čine polja *Rs1* = *Rs2* = *Rd* svi obima po 5 bitova (*RF* polje čine 32 registra), opkôd polje obima 6 bitova (moguće je kodirati do 64 različite instrukcije) i polje *ostalo* veličine 11 bitova (interpretaciju ovih bitova za sada ćemo zanemariti).




a) struktura *FX-4P*




b) format instrukcije

Slika 2. Izvršenje ALU instrukcije kod četvoro-stepenog protočno organizovanog sistema *FX-4P*

***Napomena**: Gradivni blok RF polje prikazan u stepenima ID i WB je jedan isti blok. U suštini to je memorija sa većim brojem pristupa. Čitanje ovog polja se vrši u stepenu ID, a upis u RF polje u stepenu WB.*

Stepen *IF* prenosi kompletnu instrukciju u leč *L1*. U stepenu *ID* pristupa se izvornim operandima instrukcije koji su locirani u *RF* polju i dobavljaju se vrednosti izvornih operanada (u konkretnom slučaju to su sadržaji registara *R2* i *R3*). Sadržaji izvornih operanada zajedno sa opkôdom instrukcije i polje odredišni registar, *R1*, se upisuju u leč *L2*, tj. predaju stepenu *EX*. Stepen *EX* prihvata ulazne vrednosti i

obavlja operaciju u zavisnosti od opkoda instrukcije, a rezultat zajedno sa poljem odredišni registar prosledjuje leču *L3*. Stepen *WB* pristupa *RF* polju koristeći odredišnu adresu i upisuje (prenosi) rezultantnu vrednost (*R2*) + (*R3*) u odredišni registar *R1*.

Analizirajući Sliku 2 možemo da zaključimo sledeće:

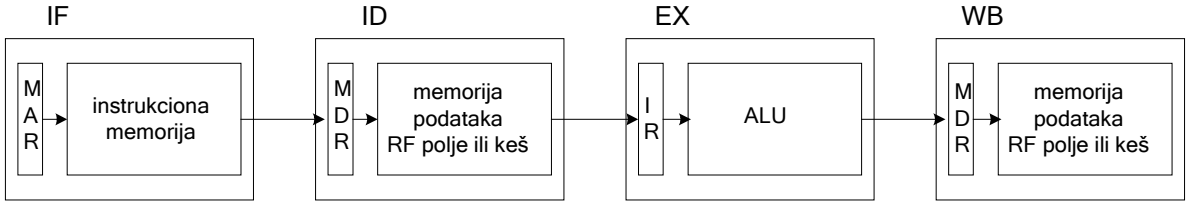
a) sadržaj opkôda polja instrukcije ostaje nepromenjen sve do stepena *EX*, tj informacija o opkôd polju se jednostavno prenosi kroz stepene *IF* i *ID*.

b) na sličan način polje instrukcije kojim se specificira odredišni registar, *Rd* (konkretno je to *R1*), se prenosi kroz stepene *IF*, *ID* i *EX* sve dok se njegov sadržaj ne interpretira u stepenu *WB*, za potrebe selekcije registra *R1* iz *RF* polja, radi upisa rezultata izračunavanja.

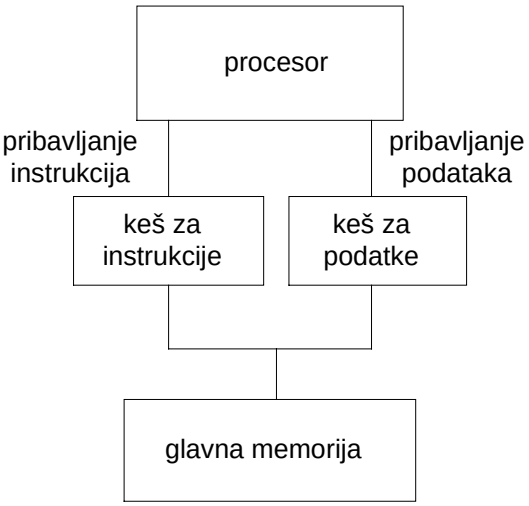
Zadržavajući nepromenjen sadržaj odredjenih polja instrukcije, u toku prolaska te instrukcije kroz protočne stepene moguće je insertovati nove instrukcije a da pri tome prethodne još nisu završene. Isto tako se može uočiti da u trenutku kada *n*-ta instrukcija vrši upis u *RF* polje (*n*+2)-a instrukcija čita *RF* polje. Istovremeno operacije upis i čitanje sadržaja istog registra nisu dozvoljene, ali je dozvoljen istovremeni pristup različitim registrima (*RF* polje je multiport memorija, koja u konkretnom slučaju (Slika 2) ima dva porta za čitanje i jedan za upis).

### 1.4.2 Instrukcije tipa *Load* i *Store*

Kod instrukcija tipa *Load* i *Store* u okviru aktivnosti koje obavljaju protočni stepeni vrši se pristup memorijama za podatke. Saglasno ovome može se izvesti jedan alternativni način prikazivanja rada protočnog sistema koji za odgovarajući protočni stepen u prvi plan postavlja pristup memoriji (kešu) za podatke (vidi Sliku 3)




a)




b)

Slika 3. Alternativni način prikazivanja rada protočnog sistema u toku jedinstvenog procesorskog ciklusa

***Napomena**: MAR-memorijsko adresni registar; MDR-registar za čuvanje podataka iz/ka memoriji; IR-instrukcioni registar*.

Kao što se može uočiti sa Slike 3 u toku jednog procesorskog ciklusa memoriji za podatke istovremeno se pristupa na dva mesta, u stepenu *ID* i stepenu *WB*, a memoriji za instrukcije jedanput, u stepenu *IF*. Ovakav način rada automatski sugeriše da memoriju za instrukcije treba razdvojiti od memorije za podatke. Obično, na nivou sistema obe memorije se implementiraju kao *keš-za-instrukcije* i *keš-za-podatke*.(vidi Sliku 3 b)).

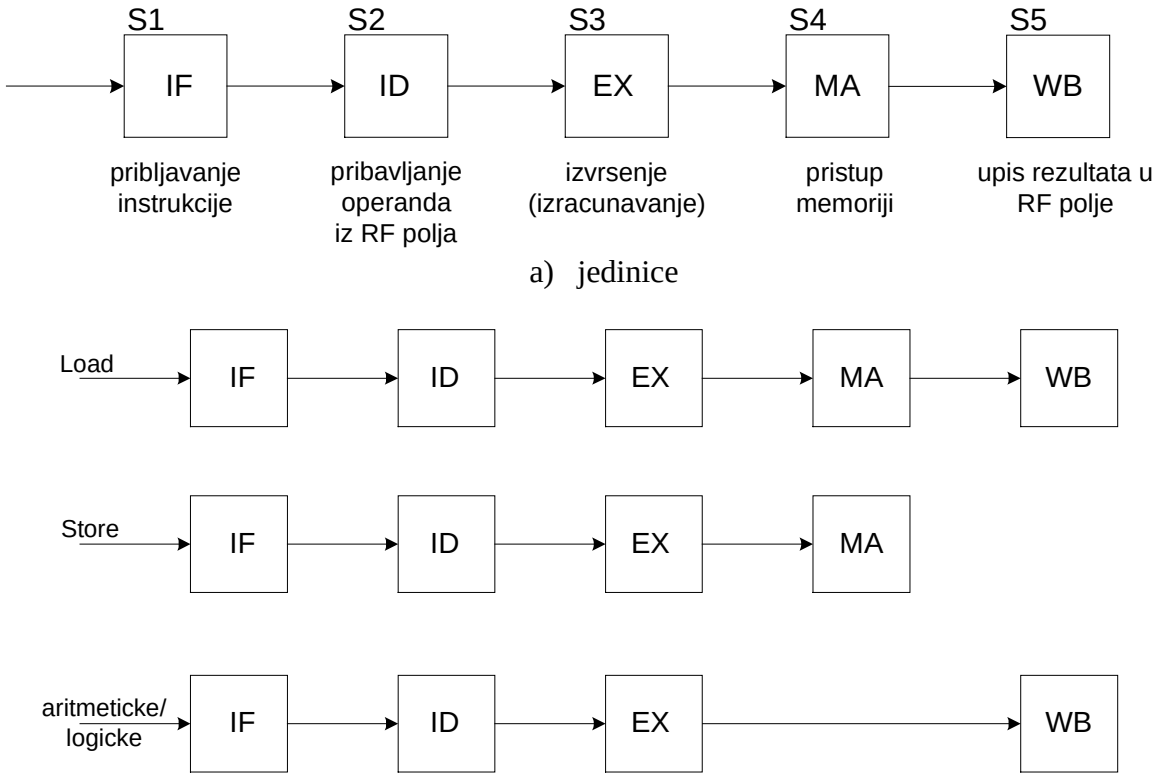
Uobičajeno, današnji RISC procesori se realizuju kao *Load*/*Store* mašine, pa je pogodno kod ovakvih mašina da postoji još jedan protočni stepen koji bi obavljao aktivnost tipa pristup-memoriji. Memorijske adrese za obe, *Load* i *Store*, operacije, izračunavale bi se putem registarsko indirektnog načina adresiranja. Ovakav pristup nalaže da protočni sistem čine sledećih pet stepeni:

S1: **FI** – pribavljanje instrukcije
S2: **DI** – dekodiranje/pribavaljanje operanada iz *RF* polja
S3: **EX** – izvršenje *ALU* operacije ili izračunavanje efektivne adrese
S4: **MA** – pristup memoriji radi upisa podataka ili čitanje memorijskog operanda
S5: **WB** – upis rezultata operacije u *RF* polje

Stepen *S3* se koristi za izračunavanje efektivne adrese instrukcije kojom se vrši pristup memoriji, ili za obavljanje aritmetičko/logičkih operacija kod instrukcija tipa regista-u-registar. Stepen *S5* se ne koristi od strane instrukcija koje vrše upis u memoriju (*Store* instrukcija).

Na Slici 4 prikazan je način korišćenja protočnih stepena za instrukcije tipa *Load*, *Store* kao i aritmetičko/logičkih. Sistem sa Slike 4 a) je peto-stepeni sa jednim stepenom namenjen za pristup memoriji. Kao što se vidi sa Slike 4 b) samo se instrukcija *Load* procesira u sva pet stepena; aritmetičko/logičke instrukcije ne koriste usluge *MA* stepena (samo prolaze kroz njega bez procesiranja); instrukcija *Store* ne koristi usluge stepena *WB*. Treba ipak naglasiti da neiskorišćavanje nekog od protočnih stepena od strane odredjene instrukcije ne doprinosi smanjenju propusnosti sistema.

Peto-stepeni protočno organizovan sistem prikazan na Slici 4 a) nazivaćemo ga u daljem tekstu procesor *FX-5P*.




Slika 4. Peto-stepeni protočni sistem, *FX-5P*, sa jednim stepenom za pristup memoriji

Na Slici 5a), ilustracije radi, prikazano je izvršenje instrukcije

Lw R1, 100(R2) ; (R1)← M(100+R2)

Format instrukcije *Load* prikazan je na Slici 5b).
Aktivnosti koje se, kod *FX-5P*, preduzimaju su sledeće:

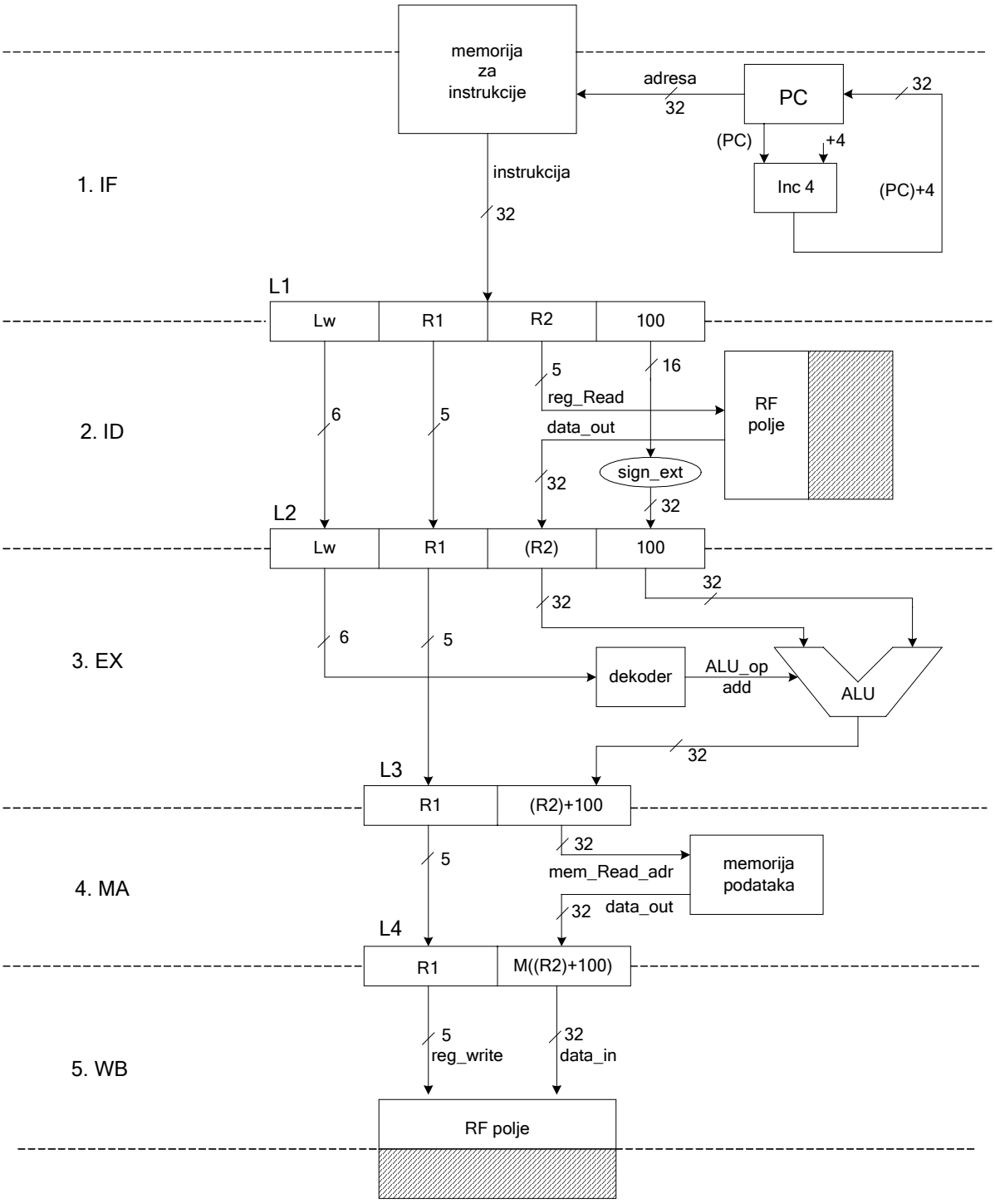
1. u okviru stepena *IF* pribavlja se instrukcija i upisuje u leč *L1*

2. u stepenu *ID*, na osnovu specifikacije adrese izvorišnog registra *R2*, čita se iz *RF* polja sadržaj registra *R2* i upisuje u *L2*. Neposredna vrednost *imm* = 100 koja, u konkretnom slučaju, predstavlja 16-bitna konstanta, pomoću specijalizovanog hardverskog bloka *sign-ext* znakovno se proširuje na 32 bita i upisuje u *L2*

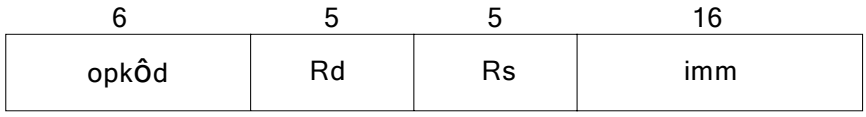
3. U trećem stepenu vrši se dekodiranje opkôd polja, tako da blok dekoder generiše upravljački signal *ALU_op = add* pomoću koga se izdaje nalog ALU jedinici da sabere vrednosti oba ulazna operanda. Rezultat ALU-a je (*R2*)+100 i predstavlja izračunatu efektivnu adresu koja se upisuje u *L3*.

4. U četvrtom stepenu, *MA*, na osnovu efektivne adrese, pristupa se memoriji za podatke. Sa memorijske lokacije M(*R2*+100) čita se podatak koji se upisuje u leč *L4*.

5. U zadnjem stepenu u registar *R1* koji pripada *RF* polju upisuje se prethodno pročitani sadržaj dobavljen iz memorije podataka.




a) način izvršenja instrukcije




b) Format instrukcije *Load*

Slika 5. Izvršenje instrukcije Lw R1, 100(R2)

Na Slici 6a) prikazan je način izivršenja instrukcije *Store* (konkretnije *Sw R4, 200(R3))* na procesoru *FX-5P*. Aktivnosti koje se preduzimaju u okviru izvršenja ove instrukcije su sledeće:

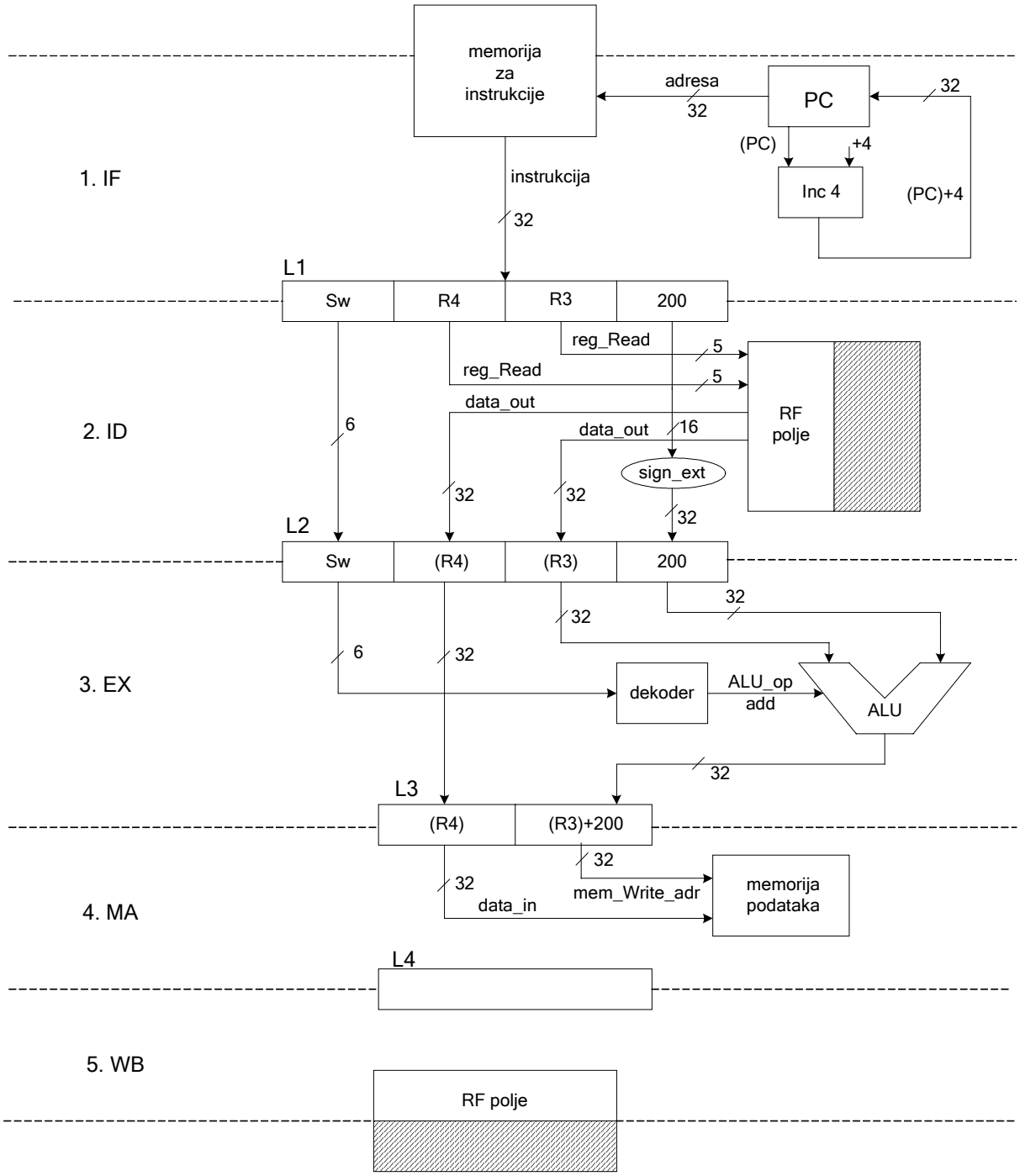
1. U okviru stepena *IF* pribavlja se instrukcija i upisuje u leč *L1*

2. U stepenu *ID* obavljaju se sledeće akcije:
   a) na osnovu specifikacije registarskih polja *R4* i *R3* čitaju se odgovarajući sadržaji registara iz RF polja i upisuju u leč *L2*
   b) 16-bitna neposredna vrednost *imm* = 200 znakovno se proširuje na 32 bita i upisuje u leč *L2*

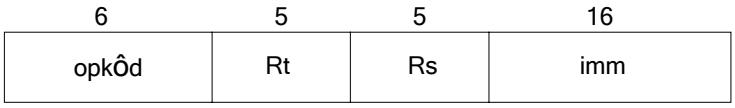
3. U okviru stepena *EX* dekodira se opkôd instrukcije i ALU jedinici izdaje nalog za sabiranje ulaznih operanada. Izlaz ALU-a predstavlja efektivna memorijska adresa memorije podataka u koju treba upisati sadržaj registra *R4*

4. U stepenu *MA* sadržaj registra *R4* upisuje se u memoriju podataka na lokaciji (*R3*) + 200.

5. U okviru stepena *WB*, instrukcijom *Store* nije specificirana nikakva aktivnost.




a) Način izvršenja instrukcije




b) format instrukcije *Store*

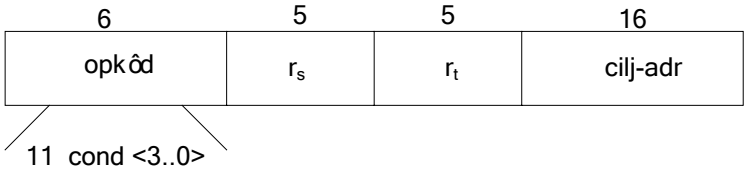
Slika 6. Izvršenje instrukcije Sw R4, 200 (R3)

### 1.4.3 Instrukcija tipa Branch

Instrukcije grananja (*Branch*) formiraju različitu klasu instrukcija u odnosu na aritmetičko/logičke i *Load*/*Store* instrukcije. Razlika se sastoji u tome što instrukcije tipa *Branch* upisuju novu vrednost u programski brojač, *PC*.

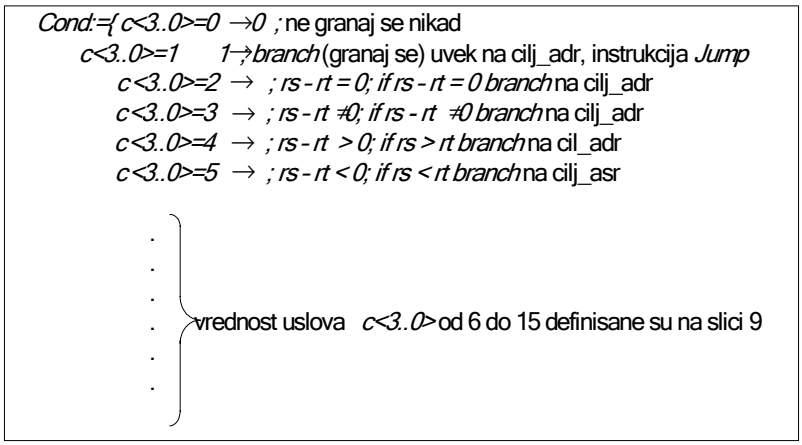
Instrukcije grananja mogu biti uslovne, pa ih tada nazivamo *Bcc* (*Branch Conditional*), ili bezulsovne, poznate kao *Jump* (*Jump Unconditional*).

Format instrukcije *Branch* za uslovne i bezuslovne instrukcije grananja je jedinstven i oblika je kao na Slici 7.




Slika 7. Format instrukcije *Branch*

Jedna od interpretacije četvoro-bitnog polja *cond* koje je sastavni deo opkôd polja je prikazana na Slici 8:




Slika 8 Format instrukcije tipa *Branch*

Polja $r_s$ i $r_t$ su registri čije se vrednosti uporedjuju, a polje *cilj_adr* predstavlja ciljnu adresu grananja.

Spisak svih instrukcija tipa *Branch* prikazan je na Slici 9.

| Tip instrukcije | Mnemonik | Marker uslova koji se testira | |
|---|---|---|---|
| nikad se ne granaj | *Jnev* | --- | *C<3..0>=0* |
| uvek se granaj (bezuslovni skok) | *Jump* | --- | *C<3..0>=1* |
| granaj se ako je nula | *Bz* | *If {z}=1 go to cilj_adr else next_adr;* | *C<3..0>=2* |
| granaj se ako je nije nula | *Bnz* | *If {z}=0 go to cilj_adr else next_adr;* | *C<3..0>=3* |
| granaj se ako je parnost parna | *Bpe* | *If {p}=1 go to cilj_adr else next_adr;* | *C<3..0>=6* |
| granaj se ako je parnost neparna | *Bpo* | *If {p}=0 go to cilj_adr else next_adr;* | *C<3..0>=7* |
| granaj se ako postoji prenos | *Bc* | *If {c}=1 go to cilj_adr else next_adr;* | *C<3..0>=8* |
| granaj se ako ne postoji prenos | *Bnc* | *If {c}=0 go to cilj_adr else next_adr;* | *C<3..0>=9* |
| granaj se ako je negativan | *Bsn* | *If {s}=1 go to cilj_adr else next_adr;* | *C<3..0>=10* |
| granaj se ako je pozitivan | *Bsp* | *If {s}=0 go to cilj_adr else next_adr;* | *C<3..0>=11* |
| granaj se ako ima premasaj | *Bo* | *If {o}=1 go to cilj_adr else next_adr;* | *C<3..0>=12* |
| granaj se ako nema premašaj | *Bno* | *If {o}=0 go to cilj_adr else next_adr;* | *C<3..0>=13* |
| Kombinacije C<3..0>=4 i 5 su rezervisane za *Bgt (branch if greater then), i Bet (branch if less then),* dok su ostale dve kombinacije ostavljene za buduća proširenja | | | |

Slika 9. Spisak instrukcija tipa *Branch*

Analizom Slike 9 možemo da zaključimo sledeće:

- bezuslovna instrukcija grananja *Jump* je specijalan slučaj instrukcije *Branch*
- instrukcija *Jnev* sa programerske tačke gledišta nema smisla, ali je uvedena zbog jednostavnije realizacije hardvera za grananje

Na Slici 10 prikazano je izvršenje uslovne instrukcije grananja

Bz R4, R0, 500 ; *if* R4 - R0 = 0 *go to* (PC) + 4 + 500, *else next_adr*

Instrukcija *Br* (*branch zero*) komparira sadržaje registara *R4* i *R0*. Za slučaj da su njihove vrednosti identične razlika će biti 0, i postaviće se marker uslova $Z = \{1\}$ . Ako je $Z = \{1\}$ obaviće se grananje na cilju adresu *cilj_adr*, a za slučaj da je $Z = \{0\}$ izvršava se naredna instrukcija u sekvenci ( na adresu *next_adr*), tj. ona koja neposredno sledi iza instrukcije *Bz*.

Uobičajena je praksa, da projektanti procesora uvek rezervišu registar *R0* kao lokaciju gde se čuva konstanta 0 (konstanta 0 se često puta javlja u programima pa je bolje da se ona generiše u okviru samog procesora, a ne da se dobavlja iz memorije kao neposredna vrednost čime se značajno postiže na ubrzanju programa). Implikacije ovakvog pristupa, na primer kod instrukcije *Bz R4, R0, adr1*, su sledeće:

Razlika *R4 - R0* jednaka je pravoj vrednosti registra *R4* jer se oduzimanjem 0 ništa ne menja, a postavljaju se samo markeri uslova kakvi su *Z – zero*, *S – sign*, *P – parity, O – overflow*, *C – carry*, i dr.

Aktivnosti koje su tipične za rad svakog od protočnih stepena u toku izvršenja instrukcije *Bz R4, R0, 500* su sledeće:

1. u okviru aktivnosti stepena *IF* iz memorije se pribavlja opkôd instrukcije *Bz R4, R0, adr1* i upisuje u leč *L1*. Dodatno u *L1* upisuje se i stanje *PC*-a koje ukazuje na adresu naredne instrukcije u programu (tj. (*PC*)+4)

2. u okviru aktivnosti stepena *ID* obavlja se sledeće:
   a) na osnovu specifikacije polja *R4* i *R0* iz odgovarajućih registara *RF* polja pribavljaju se sadržaji (*R4*) i (*R0*), respektivno
   b) 16-bitno polje *ciljna_adr* = 500 znakovno se poširuje na 32 bita. Znakovno proširena vrednost upisuje se u leč *L2*.

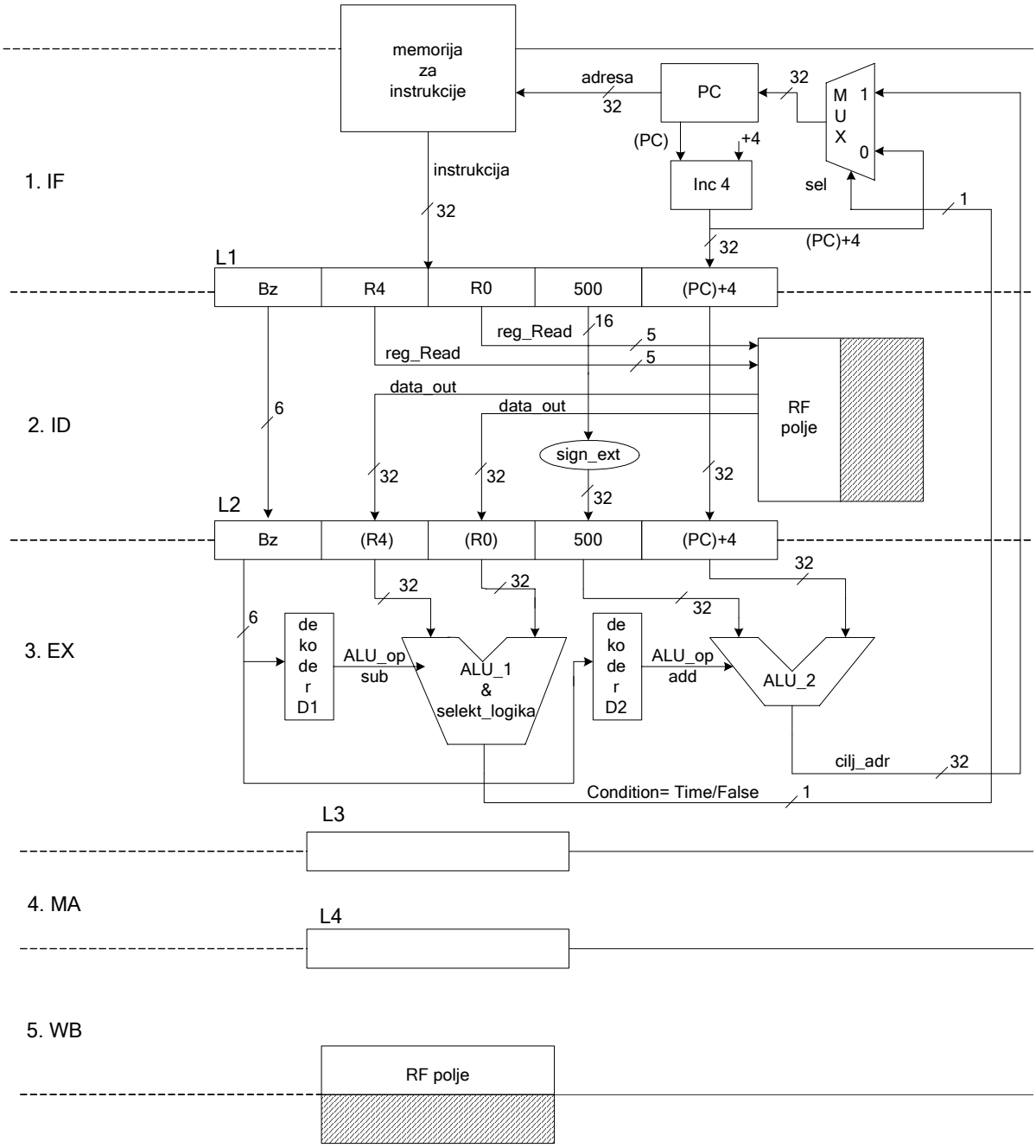
3. Protočni stepen *EX* obavlja sledeće aktivnosti:

i) Gradivni blok *ALU_1&Select_Logika* uporedjuje sadržaj registra *R4* i *R0*. Uporedjivanje se obavlja operacijom oduzimanja, (*R4*) - (*R0*). Ovu funkciju obavlja *ALU_1*. Kao rezultat oduzimanja postavljaju se markeri uslova (*Z*,*S*,*P*,*C*,*O*). *Select_logika* u zavisnosti od stanja markera uslova, kao i koji se od uslova testira, postavlja svoj izlaz *Condition* na vrednost *True* $\rightarrow$ 1 (kada je uslov grananja ispunjen), ili *False* $\rightarrow$ 0 (kada uslov grananja nije ispunjen). Izlaz *Condition* se dovodi na *Sel* multipleksera MUX koji je sastavni deo logike *IF* stepena, a koristi se da selektuje koji će se od ulaza MUX-a propustiti na njegov izlaz, tj. dovesti na ulaz *PC*-a. Kada se *ulaz_0* MUX-a propusti na izlaz, program produžava sa izvršenjem naredne instrukcije u sekvenci, a kada se *ulaz-1* propusti na izlaz obavlja se grananje na adresi *cilj_adr*.

ii) Gradivni blok *ALU_2* obavlja sabiranje vrednosti (*PC*)+4 i znakovno proširenog razmeštaja 500. To znači da protočni sistem koga analiziramo, kada je reč o instrukcijama tipa *Branch*, podržava *PC* relativno adresiranje sa razmeštajem. Vrednost na izlazu *ALU_2* jednaka je *cilj_adr* = ((*PC*)+4) + 500, i odgovara adresi grananja za slučaj da je uslov grananja ispunjen

4. Protočni stepen *MA* ne obavlja nikakvu aktivnost

5. Protočni stepen *WB* ne obavlja nikakvu aktivnost.




Slika 9. Izvršenje instrukcije *Bz R4, R0, 500*