5.6 DIZAJNIRANJE GLAVNE KONTROLNE JEDINICE MULTITAKTNE IMPLEMENTACIJE

Glavna kontrolna jedinica generiše selekcione ulaze svih multipleksora i kontrolne ulaze svih memorijskih elemenata upotrijebljenih u multitaknoj implementaciji prikazanoj na slicu 11. Svoje izlaze glavna kontrolna jedinica generiše na osnovu operacionog koda, op=Op [5–0]=Instruction [31–26], instrukcije koja se izvršava. Primijetimo da glavna kontrolna jednica multitaktne implementacije generiše značajno veći broj izlaznih signala u odnosu na glavnu kontrolnu jedinicu jednotaktne implementacije sa slike 8. Ova činjenica implicira zaključak o komplikovanijoj, odnosno značajno složenijoj kontroli multitaktne u odnosu na jednotaktnu implementaciju, u što ćemo se uvjeriti u ovom poglavlju. Pojednostavljeno, komplikovanija kontrola multitaktne implementacije može se posmatrati kao još jedan način "plaćanja" jednostavnije, manje gabaritne, enetgetski manje zahtjevne i jeftinije implementacije u odnosu na jednotaktnu implementaciju.

Prije nego predjemo na dizajniranje glavne kontrolne jedinice multitaktne implementacije sa slike 11, pobrojmo najprije ranije razmatrane (pogledaj zapažanja 1 i 2, analizu i napomenu 2 iz sekcije 5.5.1) kontrolne signale koje ona treba da generiše:

- Kontrolne signale na selekcionim ulazima svakog od 6 upotrijebljenih multipleksora:
  IorD signal multipleksora na Raed address ulazu Memory jedinice,
  RegDst signal multipleksora na Write register adresnom ulazu Registers jedinice,
  MemtoReg signal multipleksora na Write data ulazu Registers jedinice,
  ALUSelA signal multipleksora na prvom ulazu ALU,
  2-bitni *ALUSelB* signal multipleksora na drugom ulazu ALU,
  2-bitni *PCSource* signal multipleksora na ulazu PC registra.

- 7 kontrolnih signala memorijskih elemenata:
  IRWrite kontrolni signal upisa instrukcije u IR registar,
  MemWrite kontrolni signal upisa podatka u Memory jedinicu,
  MemRead kontrolni signal čitanja podatka iz Memory jedinice
  RegWrite kontrolni signal upisa podatka/rezultata ALU u Regitars jedinicu,
  TargetWrite kontrolni signal upisa rezultata ALU u Target registar,
  PCWrite kontrolni signal bezuslovnog upisa adrese sljedeće instrukcije u PC registar,
  PCWriteCond kontrolni signal uslovnog upisa adrese sljedeće instrukcije u PC registar.

- Kontrolne signale ALU control jedinice:
  2-bitni ALUOp signal ($ALUOp_{(1,0)}$).

NAPOMENA 1: Kao što je već razmatrano (pogledaj napomenu 2 iz sekcije 5.5.1), adresa instrukcije koja sljedeća treba da se izvrši može biti PC+4 (kod instrukcija koje se sukcesivno izvršavaju), ciljna adresa grananja (nakon izvršavanja instrukcije uslovnog skoka) ili adresa bezuslovnog skoka (nakon izvršavanja instrukcije bezuslovnog sloka – Jump address na slici 11). Adrese PC+4 i Jump address bezuslovno se upisuju u PC registar, dok se ciljna adresa grananja upisuje u PC registar samo ukoliko je zadovoljen uslov grananja (ukoliko je Zero=1). Drugim riječima, potrebno je predvidjeti kontrolu bezuslovnog upisa, ali i kontrolu uslovnog upisa adrese sljedeće instrukcije u PC registar. U tom cilju, *PCWrite* kontrolni signal upotrebljava se za kontrolu bezuslovnog skoka u PC registar, a *PCWriteCond* (zajedno sa signalom Zero) za kontrolu uslovnog upisa u PC registar.

NAPOMENA 2: Primijetimo suštinsku razliku izmedju kontrolnih signala multipleksora i kontrolnih signala memorijskih elemenata. Obje vrijednosti (1 i/ili 0) kontrolnih signala multipleksora proizvode odredjenu akciju, a samo jedinična vrijednost kontrolnog signala memorijskog elementa proizvodi akciju zahtijevanu od odgovarajućeg elementa. Što pod time podrazumijevamo?

- Što se tiče kontrolnih signala multipleksora, odgovor ćemo dati posmatrajući kontrolni signal *IorD* multipleksora na Read address ulazu Memory jedinice. Jasno je da svaka linija u sistemu, time i linija koja odgovara *IorD* kontrolnom signalu, mora da "nosi" vrijednost ili logičke 0 ili logičke 1. Kada je *IorD*=0, na Read address ulaz Memory jedinice dovodi se sadržaj PC registra, a kada je *IorD*=1, na Read address ulaz Memory jedinice dovodi se adresa izračunata

od strane ALU. Dakle, i vrijednost 0 i vrijednost 1 kontrolnog *IorD* signala proizvode odredjenu akciju (dovodjenje sadržaja PC registra ili adrese izračunate od strane ALU na Read address ulaz Memory jedinice) i strogo se mora voditi računa, prilikom upotrebe multipleksora, koja će se vrijednost kontrolnog signala postaviti na njegove selekcione ulaz(e),

– Što se tiče kontrolnih signala memorijskih elemenata, odgovor ćemo dati posmatrajući *MemWrite* kontrolni signal Memory jedinice. Samo 1-na vrijednost *MemWrite* kontrolnog signala proizvodi akciju (upis u Memory jedinicu), dok 0-a vrijednost ovog signala onemogućava upis u Memory jedinicu (ne proizvodi akciju, već onemogućava akciju upisa). Suštinski, kontrolni bitovi memorijskih elemenata proizvode akciju (upis u memorijski element ili čitanje iz elementa, ali čitanje samo u slučaju Memory jedinice) samo kada uzmu vrijednost 1. Za vrijednost 0 kontrolnog signala, odgovarajuća akcija se onemogućava. Stoga, zbog većeg broja kontrolnih signala koje generiše glavna kontrolna jedinica multitaktne implementacije, samo se navodi naziv kontrolnih signala memorijskih elemenata koji treba da uzmu vrijednost 1, bez označavanja =1 (samim njihovim navodjenjem pretpostavlja se njihova 1-na vrijednost). Ukoliko kontrolni signal memorijskog elemnta treba da uzme vrijednost 0, naziv ovog signala jednostavno se ne navodi.

5.6.1 Postavljanje kontrolnih signala neophodnih za relizaciju I koraka izvršavanja (uzimanje/čitanje instrukcije i inkrementiranje sadržaja PC registra)

U I koraku izvršavanja instrukcija (pogledaj tabelu 2 iz sekcije 5.2), uzima se instrukcija iz Memory jedinice i upisuje se u IR registar i, u paraleli, inkrementira se sadržaj PC registra i izračunato PC+4 upisuje se nazad u PC registar, odnosno:

$$IR \leftarrow Mem[PC], \tag{4}$$

$$PC = PC + 4. \tag{5}$$

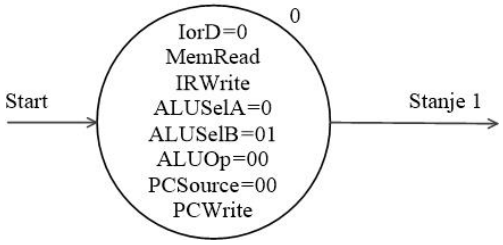
Postavimo najprije kontrolne signale neophodne za implementaciju akcije (4). U tom cilju, neophodno je obezbijediti da sadržaj PC registra adresira Memory jedinicu u svrhu čitanja instrukcije sačuvane u njoj, tj. potrebno je dovesti sadržaj PC registra na Read address ulaz Memory jedinice (*IorD*=0), omogućiti čitanje iz Memory jedinice (*MemRead*=1, uz napomenu da će u dijagramu stanja biti zapisivano samo *MemRead*, pošto je riječ o kontrolnom signalu memorijskog elementa – pogledaj napomenu 2 iz poglavlja 5.6), te obezbijediti upis pročitanje instrukcije u IR registar (*IRWrite*).

Postavimo sada kontrolne signale neophodne za implementaciju akcije (5). Pošto je riječ o izračunavanju, za implementaciju (5) upotrebljava se ALU. U tom cilju, na I ulaz ALU potrebno je dovesti sadržaj PC registra (u ovu svrhu, potrebno je postaviti *ALUSelA*=0), a na II ulaz ALU – konstantu 4 (*ALUSelB*=01). ALU treba da obavi operaciju sabiranja (ALUOp=00). Postavljenim kontrolnim signalima, obezbijedjeno je izračunavanje PC+4. Potrebno je obezbijediti još da se izračunato PC+4 upiše nazad u PC registar. Izračunato PC+4 najprije treba dovesti na ulaz PC registra (*PCSource*=00), te potom obezbijediti bezuslovni upis u PC registar (*PCWrite*, gdje se ne zapisuje =1, jer je riječ o kontrolnom signalu memorijskog elementa – pogledaj napomenu 2 iz sekcije 5.6).

Postavljeni kontrolni signali formiraju početno stanje (stanje 0), slika 12, kojim se implementira I korak izvršavanja instrukcija (pogledaj tabelu 2 iz poglavlja 5.2). Ovo stanje/korak obično se u literaturi naziva *donošenjem instrukcije* (eng. *Instruction fetch* (*IF*)).

NAPOMENA 1: Svi kontrolni signali koji nijesu postavljeni/zapisani u odgovarajućem stanju, nijesu potrebni za implementaciju tog stanja, odnosno pretpostavlja se da uzimaju vrijednost 0. Ovo je posebno važno za kontrolne signale memorijskih elemenata, jer se njihovom vrijednosti 0 spriječava upis u odgovarajuće memorijske elemente, odnosno onemogućiti promjena njegovog sadržaja. Vrijednost 0 kontrolnih signala multipleksora uzrokovat će odredjenu akciju (pogledaj napomenu 2 iz poglavlja 5.6), ali rezultat te akcije neće moći da izmijeni sadržaje memorijskih elemenata.

NAPOMENA 2: PC+4, izračunato od strane ALU prilikom implementacije akcije (5), dolazi na ulaze više elemenata: (i) na ulaz 0 multipleksora koji se nalazi na ulazu PC registra, (ii) na ulaz Target registra, (iii) na ulaz 0 multipleksora koji se nalazi na Write data ulazu Registers jedinice, (iv) na Write



Slika 12. Početno stanje (stanje 0) mašine (sekvencijalnog kola) sa konačnim brojem stanja za implementaciju glavne kontrolne jedinice multitaktne arhitekture sa slike 11.

address ulaz Memory jedinice i (v) na ulaz 1 multipleksora na Read address ulazu Memory jedinice. Medjutim, PC+4 treba da se upiše samo u PC registar, odnosno potreban je na ulazu (i), a ne smije se dopuštiti da uzrokuje štetu dovodjenjem na ulaze (ii)–(v). U cilju upisa PC+4 u PC registar, u stanju 0 postavljeni su kontrolni signali *PCSource*=00 i *PCWrite*. Istovremeno, nepostavljanjem kontrolnih signala *TargetWrite*, *RegWrite*, *MemWrite*, pretpostavlja se da ovi kontrolni signali uzimaju vrijednost 0 (vidi napomenu 1 iz ove sekcije), čime je spriječen upis rezultata PC+4 u Target registar i u odredjeni registar Registers jedinice, kao i upis podatata u Memory jedinicu u lokaciju označenu sa PC+4, odnosno onemogućeno je neodgovarajući uticaj rezultata PC+4. Uz to, u stanju 0 postavljen je signal *IorD*=0, čime je onemogućeno adresiranje Memory jedinice (na Read address ulazu) sa PC+4.

NAPOMENA 3: Postavljanjem kontrolnih signala ALUOp=00, obavlja se bezuslovno sabiranje operanada dovedenih na ulaze ALU (pogledaj tabele 6 i 7 iz sekcije 5.4.1). Ista operacija mogla bi se izvršiti postavljanjem signala ALUOp=10, ali uslovno – operaciju bi tada odredjivalo funct polje mašinskog koda instrukcije (funct=Instruction [5–0]=$32_{(10)}$ za sabiranje – pogledaj tabelu 7 iz sekcije 5.4.1). Ipak, nema razloga funkcionisanje ALU prepuštati funct polju, ako ono može biti odredjeno postavljanjem samo ALUOp bitova. Na koncu, u ovom trenutku (izvršavanja I koraka, kome odgovara stanje 0), ne raspolažemo funct poljem, a takodje sve instrukcije ne raspolažu funct poljem. Drugim riječima, u ovom stanju, operacija sabiranja ALU zadaje se kontrolnom signalima ALUOp=00.

NAPOMENA 4: Paralelno izvršavanje akcija (4) i (5) omogućeno je činjenicom da se za izvršavanje ovih akcija upotrebljavaju različiti funkcionalni elementi. Tokom izvršavanja akcije (4) pristupa se Memory jedinici u cilju čitanja instrukcije, a u cilju izvršavanja akcije (5) upotrebljava se ALU za izračunavanje PC+4. Primijetimo da se ove akcije ne bi mogle paralelno izvršavati ukoliko bi jedna od njih upotrebljavala rezultat one druge akcije, što ovdje nije slučaj.

5.6.2 Postavljanje kontrolnih signala neophodnih za relizaciju II koraka izvršavanja (dekodiranje instrukcije i izračunavanje ciljne adrese grananja)

U II koraku izvršavanja (pogledaj tabelu 2 iz sekcije 5.2), dekodira se instrukcija sačuvana u IR registru (u I koraku), pristupa se sadržajima registara koji su označeni pojedinim poljima instrukcije i, u paraleli, izračunava se ciljna adresa grananja i, nakon izračunavanja, upisuje se u registar Target,

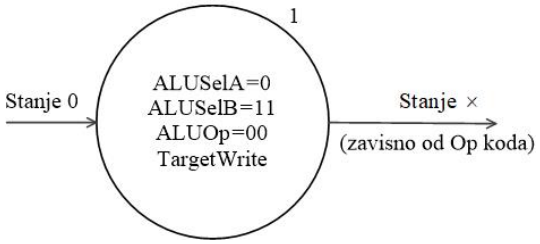
$$\text{pristup/čitanje Reg(IR [25–21])}, \tag{6}$$

$$\text{pristup/čitanje Reg(IR [20–16])}, \tag{7}$$

$$\text{Target=PC+sign\_extend(IR [15–0])<<2}, \tag{8}$$

gdje se, nakon I taktnog interala/koraka, u PC registru nalazi PC+4 (vidji napomenu 1 u sekciji 5.5.1).

Dekodiranje instrukcije podrazumijeva čitanje polja instrukcije sačuvane u IR registru, IR [31–0]≡Instruction [31–0], odnosno čitanje odgovarajućih bitova IR registra, gdje su op=Op[5–0]=IR [31–26], rs=IR [25–21], rt=IR [20–16], rd=IR [15–11], shamt=IR [10–6], address/immediate/offset =IR [15–0], funct=IR [5–0], i address polje instrukcija bezuslovnog skoka address=IR [25–0] (pogledaj zabilješku iz poglavlja 5.2). Nakon dekodiranja, poljima rs=IR [25–21] i rt=IR [20–16] označavaju se registri Registers jedinice u cilju čitanja njihovog sadržaja (akcije (6) i (7)). Pošto IR registar i Registars jedinica ne posjeduju kontrolne signale čitanja, čitanje sadržaja označenih registara može se obaviti u proizvoljnom trenutku i nije potrebno postavljati kontrolne signale u tom cilju.



Slika 13. Stanje 1 mašine (sekvencijalnog kola) sa konačnim brojem stanja za implementaciju glavne kontrolne jedinice multitaktne arhitekture sa slike 11.

Postavimo sada kontrolne signale neophodne za implementaciju akcije (8). Pošto je riječ o izračunavanju, za implementaciju (8) upotrebljava se ALU. U tom cilju, na I ulaz ALU potrebno je dovesti sadržaj PC registra (*ALUSelA*=0), a na II ulaz ALU – 16-bitno address/immediate/offset polje produženo, do 32-bitne dužine, kopiranje znaka broja i shift-ovano u lijevu stranu za 2 mjesta (*ALUSelB*=11). ALU treba da obavi operaciju sabiranja (ALUOp=00). Postavljenim kontrolnim signalima, obezbijedjeno je izračunavanje PC+sign_extend(IR [15–0])<<2. Primijetimo da je, na koncu, potrebno obezbijediti još upis izračunatog rezultata u Target registar (*TargetWrite*).

Postavljeni kontrolni signali formiraju stanje 1, slika 13, kojim se implementira II korak izvršavanja instrukcija (pogledaj tabelu 2 iz poglavlja 5.2). Ovo stanje/korak obično se u literaturi naziva *dekodiranjem instrukcije*/*donošenjem sadržaja registara* (eng. *Instruction decode*/*Register fetch* (*ID*)).

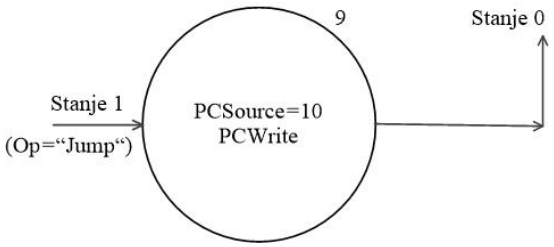
NAPOMENA 1: Postavljanjem kontrolnih signala ALUOp=00, obavlja se bezuslovno sabiranje operanada dovedenih na ulaze ALU (pogledaj tabelu 7 iz sekcije 5.4.1).

NAPOMENA 2: Paralelno dekodiranje instrukcije i izvršavanje akcije (8) omogućeno je činjenicom da se za izvršavanje ovih akcija upotrebljavaju različiti funkcionalni elementi. Tokom dekodiranja instrukcije pristupa se IR registru i Registers jedinici u cilju njihovog čitanja, a u cilju izvršavanja akcije (8) upotrebljava se ALU za zahtijevano izračunavanje. Primijetimo da nijedna od ove dvije akcije ne upotrebljavaju rezultat one druge akcije, tako da se mogu izvršavati u paraleli.

5.6.3 Kompletiranje instrukcije bezuslovnog skoka *j*

Instrukcija bezuslovnog skoka *j*, za svoje kompletiranje, zahtijeva izvršavanje još jednog koraka/taktnog intervala (pogledaj tabelu 2 iz poglavlja 5.2). U tom koraku/taktnom intervalu, adresu bezuslovnog skoka (Jump address na slici 11), koja se ne izračunava, već hardverski kreira (za pojašnjenje pogledaj napomenu 4 iz sekcije 5.5.2), potrebno je bezuslovno upisati u PC registar. U tom cilju, neophodno je dovesti (kroz multipleksor koji se nalazi na ulazu PC registra) Jump address na ulaz PC registra (*PCSource*=10), te potom obezbijediti njen bezuslovan upis u PC registar (*PCWrite*).

Postavljeni kontrolni signali formiraju stanje 9, slika 14, kojim se kompletira izvršavanja instrukcije *j* (pogledaj tabelu 2 iz poglavlja 5.2). Ovo stanje/korak obično se u literaturi naziva *kompletiranjem instrukcije bezuslovnog skoka* (eng. *Jump completion* (*JC*)).



Slika 14. Mašina sa konačnim brojem stanja za implementaciju instrukcije bezuslovnog skoka *j*.

Nakon kompletiranja instrukcije bezuslovnog skoka *j*, izvršavanje se vraća u stanje 0 u cilju uzimanja/čitanja sljedeće instrukcije iz Memory jedinice i njenog izvršavanja u sljedećih 3–5 koraka/taktnih intervala.

5.6.4 Kompletiranje instrukcije uslovnog skoka/grananja *beq*

Instrukcija uslovnog skoka/grananja *beq*, za svoje kompletiranje, takodje zahtijeva izvršavanje još jednog koraka/taktnog intervala (pogledaj tabelu 2 iz poglavlja 5.2). U tom koraku/taktnom intervalu, potrebno je:

(i) Najprije ispitati (upotrebom ALU) uslov grananja (jednakost sadržaja registara označenih poljima rs=Instruction [25–21] i rt=Instruction [20–16]), odnosno postaviti Zero izlaz ALU (ukoliko je Zero=1, uslov grananja je zadovoljen),

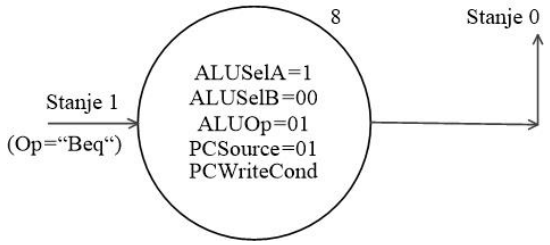
(ii) Upisati ciljnu adresu grananja, sačuvanu u Target registru, u PC registar ukoliko je uslov grananja zadovoljen (ukoliko je Zero=1),

$$PC \leftarrow Target, \text{ ako je } Zero=1. \quad (9)$$

U cilju ispitivanja uslova grananja (postavljanja Zero izlaza ALU), na I ulaz ALU potrebno je dovesti sadržaj registra koji se adresira rs=Instruction [25–21] poljem instrukcije (*ALUSelA*=1), a na II ulaz ALU – sadržaj registra koji se adresira rt=Instruction [20–16] poljem instrukcije (*ALUSelB*=00), te ALU natjerati da izvrši operaciju oduzimanja (ALUOp=01 – bitovi ALUOp uzimaju ove vrijednosti u slučaju implementacije instrukcije *beq* – pogledaj tabele 6 i 7 iz sekcije 5.4.1).

U cilju izvršavanja akcije (9), ciljnu adresu uslovnog skoka/grananja, izračunatu i sačuvanu u Target registru u II koraku/taktnom intervalu (stanje 1), potrebno je dovesti na ulaz PC registra, ali upisati u PC registar samo ukoliko je zadovoljen uslog grananja (Zero=1). Ciljna adresa grananja (sadržaj Target registra) dovodi se na ulaz PC registra kroz multipleksor koji se nalazi na ulazu PC registra (*PCSource*=01), a njen uslovni upis u PC registar obezbjedjuje se set-ovanjem/postavljanjem *PCWriteCond* kontrolnog signala.

Postavljeni kontrolni signali formiraju stanje 8, slika 15, kojim se kompletira izvršavanja instrukcije grananja *beq* (pogledaj tabelu 2 iz poglavlja 5.2). Ovo stanje/korak obično se u literaturi naziva *kompletiranjem instrukcije grananja* (eng. *Branch completion* (*BC*)).



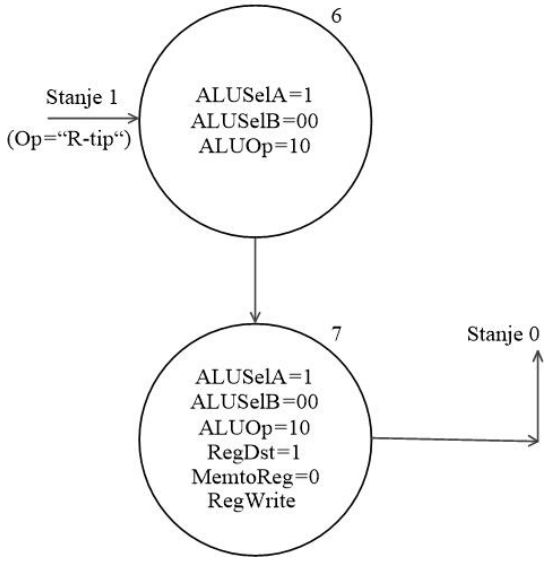
Slika 15. Mašina sa konačnim brojem stanja za implementaciju instrukcije *beq*.

NAPOMENA 1: Kontrolni signal *PCWriteCond* zajedno sa signalom Zero učestvuje u formiranju kontrolnog signala kojim se reguliše uslovni upis u PC registar. U tom cilju, ova 2 signala (*PCWriteCond* i Zero) dovode se na ulaze I kola, a na izlazu I kola generiše se signal koji kontroliše uslovni upis u PC registar. Na taj način, kotrolni signal *PCWriteCond*, generisan od strane glavne kontrolne jedinice, kontroliše upis u PC registar, ali uslovno – samo kada je Zero=1 (kada je zadovoljen uslov grananja zadat *beq* instrukcijom).

NAPOMENA 2: Kontrolni signal *PCWriteCond* upotrebljava se za kontrolu uslovnog upisa u PC registar i samo prilikom implementacije *beq* instrukcije (*beq* je jedina instrukcija uslovnog skoka/grananja koja će biti implementirana u ovom materijalu), dok se kontrolni signal *PCWrite* upotrebljava u cilju kontrole beuslovnog upisa u PC registar.

5.6.5 Kompletiranje instrukcija R-tipa

Za kompletiranje instrukcije R-tipa potrebno je izračunavanje operacije koja se zahtijeva instrukcijom i to nad operandima koji se nalaze u registrima označenim rs=IR [25–21] i rt=IR [20–16] poljima mašinskog koda instrukcije (pogledaj tabelu 2 iz poglavlja 5.2), te upisivanje izračunatog rezultata u registar označen rd=Instruction [15–11] poljem instrukcije,



Slika 16. Mašina sa konačnim brojem stanja za implementaciju instrukcija R-tipa.

$$\text{Reg(IR [15–11])} = \text{Reg(IR [25–21])} \ \underline{\text{op}} \ \text{Reg(IR [20–16])} \qquad (10)$$

gdje je op operacija zahtijevana izvršavanom instrukcijom R-tipa (*add*, *sub*, *and*, *or*, *slt*), a sadržaji registara Reg(IR [25–21]) i Reg(IR [20–16]) pročitani su u prethodnom (II) koraku/taktnom intervalu.

U cilju optimizacije vremena izvršavanja (umanjenja gubitaka u vremenu izvršavanja), akcija (10) uzima 2 koraka/taktna intervala (pogledaj tabelu 2 iz poglavlja 5.2). U prvom od ovih koraka/taktnih intervala izračunava se zahtijevana operacija, dok se u drugom koraku rezultat izračunat u prvom koraku/taktnom intervalu upisuje u odredišni registar,

1. U cilju izračunavanja zahtijevane operacije, na I ulaz ALU potrebno je dovesti sadržaj registra Registers jedinice označenog poljem rs=IR [25–21] (*ALUSelA*=1), a na II ulaz ALU – sadržaj registra Registers jedinice označenog poljem rt=IR [20–16] (*ALUSelB*=00), te, pošto je riječ o implementaciji instrukcija R-tipa, postaviti ALUOp=10 (pogledaj tabele 6 i 7 iz poglavlja 5.4.1). Postavljanje ovih kontrolnih signala odgovara stanju 6 (u literaturi nazivanim *izvršnim stanjem*, eng. *Execution* (*Ex*)), neophodnim za izvršavanje instrukcija R-tipa, slika 16.

   NAPOMENA 1: Postavljanjem kontrolnih bitova ALUOp=10, kontrola funkcionisanja ALU suštinski prepušta se funct=IR [5–0] polju instrukcija R-tipa. Drugim riječima, za ALUOp=10, funct polje instrukcija R-tipa definiše operaciju koja se instrukcijom zahtijeva (pogledaj tabelu 7 iz poglavlja 5.4.1).

2. U cilju upisivanja izračunatog rezultata u odredišni registar (registar označen rd=IR [15–11] poljem instrukcije) potrebno je obezbijediti sljedeće:

   2.1. Nepromjenljivost rezultata (iz prethodnog koraka/taktnog intervala) na izlazu ALU, zadržavanjem istih kontrolnih signala koji se odnose na funkcionisanje ALU, a postavljeni su u prethodnom koraku, odnosno *ALUSelA*=1, *ALUSelB*=00, ALUOp=10.

   NAPOMENA 2: Rezultat ALU, izračunat u III koraku/taktnom intervalu, nije sačuvan na kraju III koraka/taktnog intervala u nekom memorijskom elementu, a upotrebljava se u IV koraku. Da ovaj rezultat ne bi bio prebrisan (kreiranjem novog rezultata na izlazu ALU), neophodno je obezbjediti nepromjenljivost funkcionisanja ALU u III i u IV koraku. To se postiže zaržavanjem iste vrijednosti kontrolnih signala koji odredjuju funkcionisanje ALU (pogledaj paragraf koji se nalazi nakon napomene 2 u poglavlju 5.5.2).

   2.2. Dovesti rezultat sa izlaza ALU na Write data ulaz Registers jedinice (*MemtoReg*=0), poljem rd=IR [15–11] instrukcije označiti/adresirati registar u koji je potrebno upisati dovedeni rezultat (*RegDst*=1), te, na koncu, dozvoliti upis dovedenog rezultata u označeni registar (*RegWrite*).

Postavljeni kontrolni signali formiraju stanje 7, slika 16, kojim se kompletira izvršavanje instrukcija R-tipa (pogledaj tabelu 2 iz poglavlja 5.2). Ovo stanje/korak obično u literaturi se naziva *kompletiranjem instrukcija R-tipa* (eng. *R-type completion* (*RC*)).

NAPOMENA 3: Izvršavanjem akcije (10) u 2 koraka/taktna intervala, svaka od upotrijebljenih funkcionalnih jedinica upotrijebljava se tačno jedan put i to u odgovarajućem koraku,

- U prvom od ova 2 koraka, upotrebljava se ALU za izvršavanje zahtijevane operacije,
- U drugom – Registers jedinica za upis izračunatog rezultata u odgovarajući registar.

Uz to, taktni interval, ranije odredjen vremenom pristupa Memory jedinici (pogledaj napomenu 1 u poglavlju 5.5), dovoljno je dug za izvršavanje operacije ALU, ali i za upis razultata u odgovarajući registar Registers jedinice. Naime, konstatovano je tada da je vrijeme pristupa Memory jedinice zahtjevnije od vremena zahtijevanih od ostalih upotrijebljenih funkcionononalnih jedinica, pa i od vremena potrebnog ALU za izračunavanje i od vremena pristupa Registers jedinici. Medjutim, ukoliko bi se postupilo suprotno, odnosno ukoliko bi se akcija (10) izvršavala u jednom koraku/taktnom intervalu, upisivanje u Registers jedinicu moralo bi sačekati završetak izračunavanja u ALU (ove dvije operacije izvršavale bi se redno, a ne u paraleli kao što je bio slučaj u I i u II koraku – pogledaj sekcije 5.6.1 i 5.6.2 i stanja 0 i 1). Shodno tome, dužina taktnog intervala u ovom slučaju morala bi odgovarati sumi vremena neophodnog za izvršavanje operacije ALU i vremena neophodnog za pristup Registers jedninici. Dužina tako dizajniranog taktnog intervala bila bi veća od intervala odredjenog vremenom pristupa Memory jedinici (pogledaj primjer iz poglavlja 5.5), što bi impliciralo gubitke u vremenu izvršavanja koja se odnosi na ostale korake, a time bi impliciralo i gubitke u ukupnom vremenu izvršavanja pojedinačnih instrukcija. Stoga, taktni interval dizajnirane implementacije ostaje odredjen vremenom pristupa Memory jedinici, a akcija (10) izvršava se u 2 koraka/taktna intervala (III i IV).

NAPOMENA 4: Nakon pojašnjenja iz prethodne napomene 3, nameće se pitanje: kako se u I koraku izvršavanja instrukcija (stanje 0) može pristupiti Memory jedinici (u clju čitanja) i pročitana instrukcija potom upisati u IR registar u toku jednog te istog taktnog intervala, a za izračunavanje ALU i upisivanje dobijenog rezultata u odredjeni registar Registers jedinice potrebna su 2 taktna intervala? Ili, kako se u II koraku izvršavanja instrukcija (stanje 1) može izračunati ciljna adresa grananja u ALU i izračunata adresa potom upisati u Target registar u toku jednog te istog taktnog intervala, a za izračunavanje ALU i upisivanje dobijenog rezultata u odredjeni registar Registers jedinice potrebna su 2 taktna intervala? Odgovor na ovo pitanje je jednostavan. Registers jedinica predstavlja skup svih procesorskih registara ($0–$31), te se pristup odredjenom registru iz ovog skupa obezbjedjuje tek nakon njegovog adresiranja, odnosno nakon dekodiranja adresnih bitova dovedenih na Write register adresni ulaz Registers jedinice. Dekodiranje adrese će zahtijevati odredjeno vrijeme koje je, zajedno sa vremenom pristupa registru, značajno veće od vremena zahtijevanog za pristup pojedninačnom memorijskom elementu, kao što su IR registar ili Target registar.

5.6.6 Kompletiranje memory-reference instrukcija (*lw* i *sw*)

Kompletiranje memory-reference instrukcija (*lw* i *sw*) započinje izračunavanjem adrese memorijske lokacije čiji sadržaj će, do kraja izvršavanja instrukcija, biti razmijenjen sa sadržajem registra koji je označen rt=IR [20–16] poljem instrukcije. U cilju izračunavanja adrese memorijske lokacije, sabira se (od strane ALU) sadržaj registra, koji je označen rs=IR [25–21] poljem instrukcije, i sadržaj address-nog polja mašinskog koda instrukcija *lw*/*sw*, IR [15–0], produžen, kopiranjem znaka, do 32-bitne dužine (pogledaj tabelu 2 iz poglavlja 5.2),

$$\text{ALUOut}=\text{Reg(IR [25–21])} + \text{sign\_extend(IR [15–0])}. \quad (11)$$

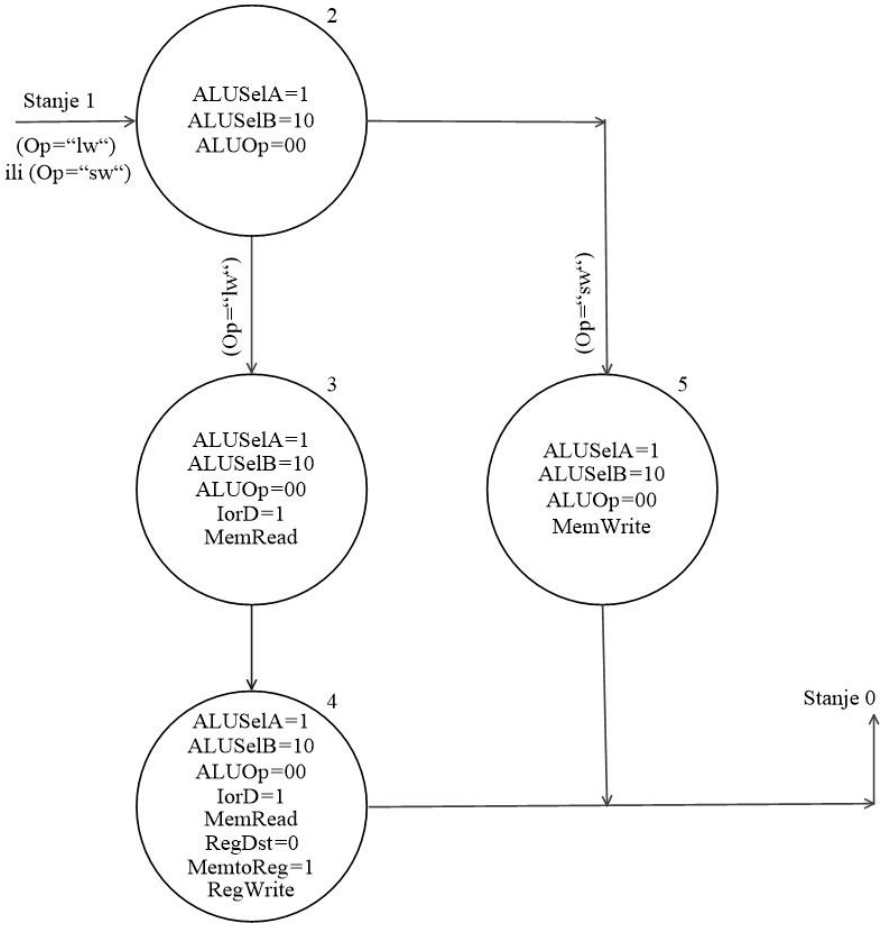
Prilikom implementacije instrukcije *lw*, sadržaj adresirane lokacije, Mem[ALUOut], potrebno je premjestiti u registar označen rt=IR [20–16] poljem instrukcije,

$$\text{Reg(IR [20–16])} \leftarrow \text{Mem[ALUOut]}, \quad (12)$$

dok je, u slučaju instrukcije *sw*, ovaj transfer podataka potrebno obaviti u suprotnom smjeru,

$$\text{Mem[ALUOut]} \leftarrow \text{Reg(IR [20–16])}, \quad (13)$$

gdje je, u slučaju instrukcije *sw*, Reg(IR [20–16]) pročitan u II koraku/taktnom intervalu.



Slika 17. Mašina sa konačnim brojem stanja za impletaciju memory-reference instrukcija (*lw* i *sw*).

Implementacije akcije (11) izvršava se u jednom (III) koraku. Ovaj korak je zajednički za *lw* i *sw* instrukcije (pogledaj tabelu 2 u poglavlju 5.2). U cilju njegove realizacije, na I ulaz ALU potrebno je dovesti sadržaj registra koji je označen poljem rs=IR [25–21] instrukcije (*ALUSelA*=1), na II ulaz ALU – 16-bitno address pilje instrukcije produženo, kopiranjem njegovog znaka, do 32-bitne dužine (*ALUSelB*=10), te omogućiti ALU da obavi operaciju sabiranja (ALUOp=00 – na koncu, implementacija instrukcija *lw*/*sw* zahtijeva postavljanje ALUOp=00 – pogledaj tabelu 6 iz sekcije 5.4.1). Ovim kontrolnim signalima formira se stanje 2 kojim se implementira III korak (u literaturi obično nazivan *izračunavanje memorijeske adrese* – eng. *Memory address computation* (*MC*)). Ovaj korak je neophodno obaviti prilikom izvršavanja instrukcija *lw* i *sw*, kao što je prikazano na slici 17.

Akcija (12), koja se obavlja tokom implementacije instrukcije *lw*, za svoje izvršavanje zahtijeva 2 koraka/taktna intervala (IV i V), dok akcija (13), koja se obavlja tokom implementacije instrukcije *sw*, za svoje izvršavanje zahtijeva 1 korak/taktni interval (IV korak/taktni interval).

NAPOMENA 1: Akcija (12) zahtijeva pristup Memory jedinici u cilju čitanja podatka iz nje, te pristup Registers jedinici u cilju upisa pročitanog podatka u odgovarajući registar. Shodno tome, akcija (12) zahtijeva vrijeme koje odgovara zbiru vremena pristupa Memory jedinici i vremena pristupa Registers jedinici. Naime, navedeni pojedinačni pristupi izvršavaju se redno, tako da ukupno vrijeme izvršavanja akcije (12) odgovara zbiru pojedinačnih vremena pristupa svake od jedinica (Memory i Registers). Medjutim, sumarno vrijeme pristupa Memory jedinici i pristupa Registers jedinici prevazilazi dužinu taktnog intervala odredjenu vremenom pristupa Memory jedinice i stoga, slijedeći principe navedene u razmatranjima u napomeni 3 iz sekcije 5.6.5, akcija (12) obavlja se u 2 taktna intervala (IV i V):

– U IV koraku/taktnom intervalu, vrši se pristup Memory jedinici u cilju čitanja podatka sa adresitrane memorijske lokacije (Mem[ALUOut]),

– U V koraku/taktnom intervalu, vrši se upis pročitanog podatka u registar označen poljem rt=IR [20–16] instrukcije *lw*.

NAPOMENA 2: Akcija (13) takodje zahtijeva serijski (redni) pristup, najprije Registers jedinici (u cilju čitanja podatka iz odgovarajućeg registra), a nakon toga Memory jedinici (u cilju upisa pročitanog podatka u adresiranu memorijsku lokaciju). Medjutim, potrebno je primijetiti da je podatak iz odgovarajućeg registra Registers jedinice pročitan prilikom dekodiranja instrukcije (u II koraku – stanje 1 – pogledaj sekciju 5.6.2), tako da je u toku IV koraka/taktnog intervala potrebno samo pročitani podatak upisati u adresiranu memorijsku lokaciju (Mem[ALUOut]).

NAPOMENA 3: Rezultat ALU (memorijska adresa), izračunat u III koraku/taktnom intervalu, upotrebljava se do kraja izvršavanja instrukcija *lw*/*sw*, a nije sačuvan (od prebrisavanja) u nekom memorijskom elementu na kraju III koraka/taktnog intervala, tako da se mora obezbijediti njegova nepromjenljivost do kraja izvršavanja obje instrukcije (*lw* i *sw*) – za detalje pogledaj napomenu 2 iz sekcije 5.6.5. U tom cilju, do kraja izvršavanja instrukcija *lw* (u IV i V koraku/taktnom intervalu) i *sw* (u IV koraku/taktnom intervalu), zadržavaju se iste vrijednost kontronih signala koji odredjuju funkcionisanje ALU u III koraku/taktnom intervalu (*ALUSelA*=1, *ALUSelB*=10, ALUOp=00).

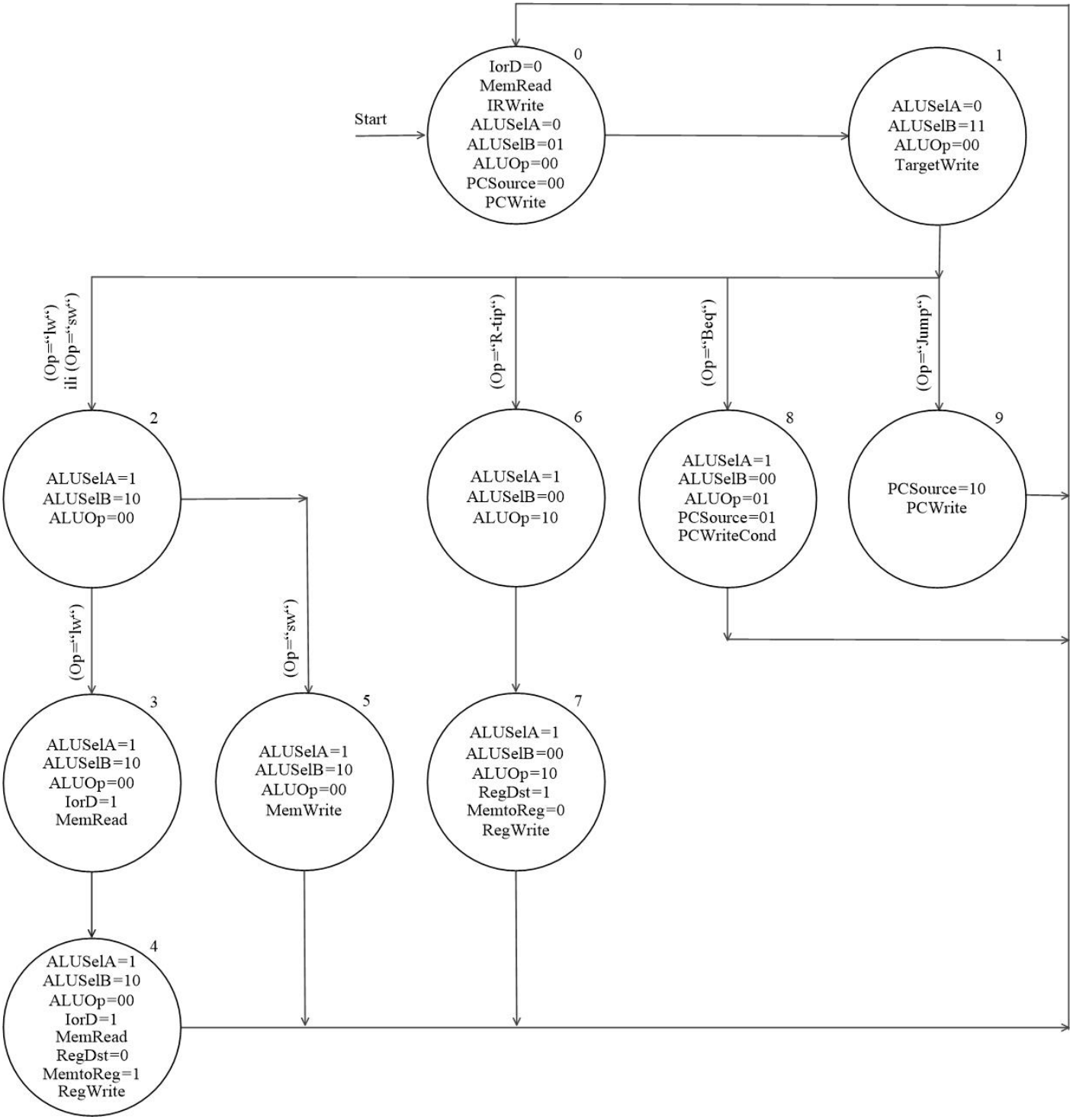
Razmotrimo najprije implementaciju instrukcije *lw*. Za svoje kompletiranje, ova instrukcija zahtijeva 2 dodatna koraka/taktna intervala (IV i V):

1. U IV koraku/taktnom intervalu, adresa izračunata (od strane ALU) u III koraku upotrebljava se za pristup Memory jedinici (*IorD*=1) u clju čitanja podatka (*MemRead*) sa adresirane memorijske lokacije. Ovim kontrolnim signalima, zajedno sa kontrolnim signalima koji obezbjedjuju nepromjenljivost funkcionisanja ALU u odnosu na III korak/taktni interval (*ALUSelA*=1, *ALUSelB*=10, ALUOp=00 – pogledaj napomenu 3 iz ove sekcije), definisano je stanje 3 kojim se implementira IV korak izvršavanja instrukcije *lw* (u literaturi obično nazivan *pristupanje memoriji* – eng. *Memory accress* (*MA*)), slika 17.
2. Podatak, pročitan iz Memory jedinice u IV koraku, nije upisan u neki memorijski elemenat za privremeno smještanje podataka na kraju ovog koraka, tako da se, iz istih razloga navedenih u napomeni 3 u ovoj sekciji i u napomeni 2 u sekciji 5.6.5, mora obezbijediti njegovo neprebrisavanje. Ono se obezbjedjuje zadržavanjem istih kontrolnih signala koji se odnose na čitanje podatka iz Memory jedinice, a koji su postavljeni u IV koraku/taktnom intervalu (*IorD*=1, *MemRead*). Nakon toga, podatak pročitan iz Memory jedinice, potrebno je dovesti na Write data ulaz Registers jedinice (*MemtoReg*=1), poljem rt=IR [20–16] označiti, na Write register ulazu Registers jedinice, registar u koji podatak treba upisati (*RegDst*=0), te, na koncu, obezbijediti upis u označeni registar Registers jedinicu (*RegWrite*). Ovim kontrolnim signalima, zajedno sa kontrolnim signalima koji obezbjedjuju nepromjenljivost funkcionisanja ALU u odnosu na III i IV korak/taktni interval (*ALUSelA*=1, *ALUSelB*=10, ALUOp=00 – pogledaj napomenu 3 iz ove sekcije), definisano je stanje 4 (u literaturi obično nazivano *upisivanje nazad* – eng. *Write back* (*WB*)), kojim se kompletira izvršavanje instrukcije *lw*, slika 17.

Razmotrimo, na koncu, implementaciju instrukcije *sw*. Za svoje kompletiranje, ova instrukcija zahtijeva 1 dodatni (IV) korak/taktni interval. U ovom koraku/taktnom intervalu, podatak koji je pročitan iz registra označenog rt=IR [20–16] poljem instrukcije (u II koraku/taktnom intervalu – tokom dekodiranja instrukcije) i koji je doveden na Write data ulaz Memory jedinice, potrebno je upisati (*MemWrite*) u memorijsku lokaciju adresiranu rezultatom ALU dovedenim na Write address ulaz Memory jedinice. Ovim kontrolnim signalom, zajedno sa kontrolnim signalima koji obezbjedjuju nepromjenljivost funkcionisanja ALU u odnosu na III korak/taktni interval (*ALUSelA*=1, *ALUSelB*=10, ALUOp=00 – pogledaj napomenu 3 iz ove sekcije), definisano je stanje 5 (u literaturi obično nazivano *pristupanje memoriji* – eng. *Memory accress* (*MA*)), kojim se kompletira izvršavanje instrukcije *sw*, slika 17.

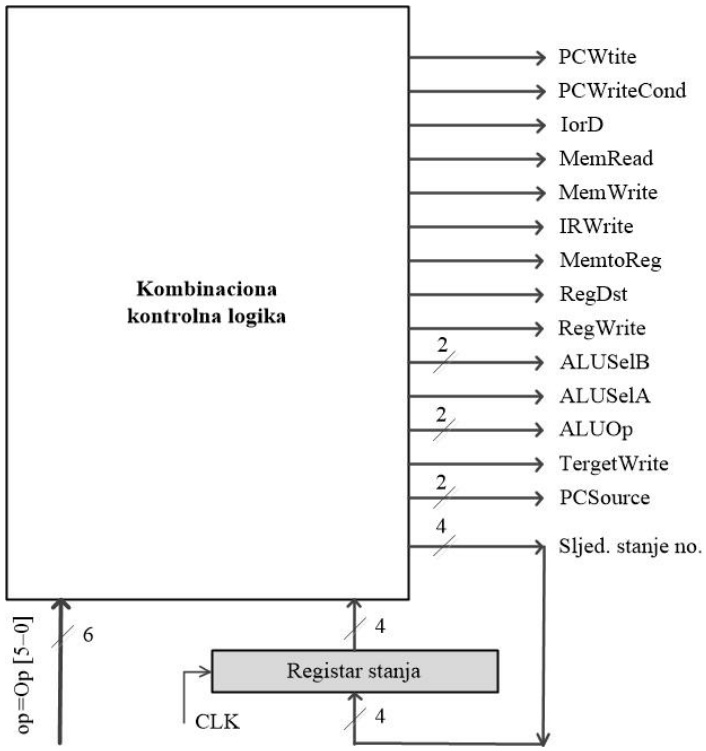
5.6.7 Definisanje glavne kontrolne jedinice multitaktne implementacije

Povezivanjem zajedničkih djelova za izvršavanje svih razmatranih instrukcija, prikazanih na slikama 12 i 13, i djelova za kompletiranje odredjenih tipova instrukcija, prikazanih na slikama 14-17,



Slika 18. Potpuni dijagram mašine sa konačnim brojem stanja kojom se implementira glavna kontrolna jedinica multitaktne arhitekture sa slike 11.

u jedinstvenu cjelinu dobija se potpuna mašina (sekvencijalno kolo) sa konačnim brojem stanja za multitaktnu implementaciju instrukcija R-tipa, *lw*, *sw*, *beq* i *j*. Potpuna mašina sa konačnim brojem stanja prikazana je na slici 18. Ona odgovara Moore-ovom tipu sekvencijalnog kola, pošto se njeni izlazi (kontrolni signali multipleksora i memorijskih elemenata upotrijebljenih u multitaktnoj implementaciji sa slike 11) postavljaju u funkciji stanja u kome se sekvencijalno kolo nalazi.



Slika 19. Logički dizajn glavne kontrolne jedinice multitaktne implementacije sa slike 11.

Strijelama je prikazan smjer prelaza izmedju stanja koji zavisi od implementirane instrukcije (njenog op koda, op=Op [5–0]=IR [31–26]), a oznakama iznad strijela prikazani su uslovi koji moraju biti zadovoljeni da bi se prešlo izmedju 2 stanja koja povezuje odgovarajuća strijela. Ukoliko se prelaz izmedju dva stanja beuslovno obavlja, nije zapisana oznaka iznad stijele koja odgovara tom prelazu.

Na grafičkoj prezentaciji sa slike 18, u svakom pojedinačnom krugu (stanju) navedeni su kontrolni signali koji se u tom stanju postavljaju. Kao što je i ranije navedeno, za kontrolne signale, koji nijesu navedenu u zapisu odredjenog stanja, pretpostavlja se da u tom stanju uzimaju vrijednosti 0 (ovo se posebno odnosi na kontrolne signale memorijskih elemanata, dok i suprotne vrijednosti kontrolnih bitova multipleksora ništa neće izmijeniti u funkcionisanju procesora – pogledaj napomenu 2 u sekciji 5.6).

Mašina sa konačnim brojem stanja, logički prikazana na slici 18, implementira se kombinacionom logikom čiji su ulazi op=IR [31–26] polje implementirane instrukcije i obilježje/broj prethodno izvršenog stanja, a izlazi – kontrolni signali multipleksora i memorijskih elemenata koje treba postaviti u zatečenom stanju i obilježje/broj stanja koje sljedeće treba izvršiti. Stoga, pored kombinacione logike, u implementaciju glavne kontrolne jedinice treba uključiti 4-bitni registar stanja (4-bitni registar, pošto mašina sa slike 18 sadrži ukupno 10 stanja za čije zapisivanje su potrebna 4 bita), kao što je prikazano na slici 19.

NAPOMENA: Mašina sa konačnim brojem stanja jedan je mogući način implementacije glavne kontrolne jedinice multitaktne arhitektire sa slike 11. Drugi mogući (moguće i jednostavniji) način za implementaciju glavne kontrolne jedinice je mikroprogramiranjem. Medjutim, ovo je tema daljeg izučavanja, koje izlazi iz okvira ovog materijala.