ELEKTROTEHNIČKI FAKULTET
U BEOGRADU

Projekat iz predmeta:

ARHITEKTURA I ORGANIZACIJA RAČUNARA II

Profesor: Dr. Jovan Đorđević

Student: Petar Parabucki

Broj indeksa : 115/2006

Beograd, Avgust 2013.

SADRŽAJ

1 TEKST ZADATKA

Posmatra se deo računara koji čine memorija, procesor i magistrala.

Memorija je kapaciteta 2^{16} bajtova. Širina memorijske reči je 1 bajt.

Procesor je sa jednoadresnim formatom instrukcija. Podaci su celobrojne veličine sa znakom i bez znaka dužine 2 bajta. Podaci i adrese u memoriji zauzimaju dve susedne memorijske lokacije, pri čemu se stariji bajt nalazi na nižoj, a mlađi bajt na višoj adresi.

U procesoru postoje bezadresne instrukcije, instrukcije uslovnog skoka, instrukcije bezuslovnog skoka i adresne instrukcije.

Bitovi 7, 6, 5 i 4 prvog bajta instrukcije su 0000 za sve instrukcije uslovnog skoka. Bitovima 3 do 0 prvog bajta instrukcije se specificira kod operacije za instrukcije uslovnog skoka. Instrukcije uslovnog skoka se realizuju kao relativni skok u odnosu tekuću vrednost programskog brojača PC, a pomeraj je 8 bitna celobrojna vrednist sa znakom data drugim bajtom instrukcije. Dužina instrukcije je dva bajta.

Bitovi 7, 6, 5 i 4 prvog bajta instrukcije su 0001 za sve instrukcije bezuslovnok skoka. Bitovima 3 do 0 prvog bajta instrukcije se specificira kod operacije za instrukcije bezuslovnog skoka. Instrukcije bezuslovnog skoka se realizuju kao apsolutni skokovi, a adresa skoka je data 2. i 3. bajtom instrukcije, pri cemu je stariji bajt adrese skoka dat drugim bajtom instrukcije a mlađi bajt adrese skoka trećim bajtim instrukcije. Dužina instrukcija je 3 bajta.

Bitovi 7, 6, 5 i 4 prvog bajta instrukcije su 1111 za bezadresne instrukcije. Bitovima 3 do 0 prvog bajta instrukcije se specificira kod operacije za bezadresne instrukcije. Dužina instrukcija je jedan bajt.

Bitovi 7, 6, 5 i 4 prvog bajta instrukcije u opsegu vrednosti 0010 do 1110 specificiraju kod operacije za adresne instrukcije. Dužina instrukcija je 1, 2, ili 3 bajta i zavisi od specificiranog načina adresiranja.

Načini adresiranja su specificirani bitovima 3 i 2 prvog bajta instrukcije. Procesor poseduje sledeće načine adresiranja: neposredno adresiranje (immed), memorijsko direktno adresiranje (memdir), registarsko indirektno sa pomerajem adresiranje (regindpom) i registarsko direktno adresiranje (regdir). Kod neposrednog adresiranja 16 bitni operand je dat drugim i trećim bajtom instrukcije, pri čemu je stariji bajt operanda dat drugim a mlađi bajt trećim bajtom. Bitovi 1 i 0 prvog bajta instrukcije se ne koriste. Dužina instrukcija je 3 bajta. Kod memorijskog direktnog adresiranja 16 bitna adresa memorijske lokacije data je drugim I trećim bajtom instrukcije, pri čemu je stariji bajt adrese dat drugim, a mlađi bajt trežim bajtom. Bitovi 1 i 0 prvog bajta instrukcije se ne koriste. Dužina instrukcije je 3 bajta. Kod registarskog indirektnog adresiranja sa pomerajem 8 bitni pomeraj je celobrojna veličina sa znakom u drugom komplementu data drugim bajtom instrukcije. Bitovi 1 i 0 prvog bajta instrukcije se koriste za adresiranje jednog od registra opšte namene R[0] do R[3]. Dužina instrukcije je 2 bajta. Kod registarskog direktnog adresiranja bitovi 1 i 0 prvog bajta instrukcije se koriste za adresiranje jednog od registara opšte namene R[0] do R[3]. Dužina instrukcije je 1 bajt.

Stek raste prema višim memorijskim lokacijama, a registar SP ukazuje na prvu slobodnu memorijsku lokaciju.

Bezadresne instrukcije su instrukcija povratka iz potprograma (RTS), instrukcija povratka iz prekidne rutine (RTI), instrukcija aritmetičkog pomeranja sadržaja akumulatora udesno za jedno mesto (ASR), instrukcija logičkog pomeranja sadržaja akumulatora udesno za jedno mesto (LSR), instrukcija rotiranja sadržaja akumulatora udesno za jedno mesto (ROR), instrukcija rotiranja sadržaja akumulatora i indikatora C udesno za jedno mesto (RORC), instrukcija aritmetičkog pomeranja sadržaja akumulatora ulevo za jedno mesto (ASL), instrukcija logičkog pomeranja sadržaja akumulatora ulevo za jedno mesto (LSL), instrukcija rotiranja sadržaja akumulatora ulevo za jedno mesto (ROL), instrukcija rotiranja sadržaja akumulatora i indikatora C ulevo za jedno mesto (ROLC), instrukcija postavljanja indikatora I na 1 (INTE), instrukcija postavljanja indikatora I na 0 (INTD), instrukcija postavljanja indikatora T na 1 (TRPE), instrukcija postavljanja indikatora T na 0 (TRPD), instrukcija prenosa sadržaj akumulatora A u registar IVTP (STIVTP) i instrukcija prenosa sadržaj akumulatora A u registar SP (STSP).

Instrukcije uslovnog skoka su:

Instrukcija	**Značenje**	**Uslov**
BEQL	skok na jednako	Z = 1
BNEQ	skok na nejednako	Z = 0
BNEG	skok na N = 1	N = 1
BNNG	skok na N = 0	N = 0
BOVF	skok na V = 1	V = 1
BNVF	skok na V = 0	V = 0
BCR	skok na C = 1	C = 1
BNCR	skok na C = 0	C = 0
BGRT	skok na veće nego (sa znakom)	$(N \oplus V) \vee Z = 0$
BGRE	skok na veće nego ili jednako (sa znakom)	$N \oplus V = 0$
BLSS	skok na manje nego (sa znakom)	$(N \oplus V) = 1$
BLEQ	skok na manje nego ili jednako (sa znakom)	$(N \oplus V) \vee Z = 1$
BGRTU	skok na veće nego (bez znaka)	$C \vee Z = 0$
BGREU	skok na veće nego ili jednako (bez znaka)	C = 0
BLSSU	skok na manje nego (bez znaka)	C = 1
BLEQU	skok na manje nego ili jednako (bez znaka)	$C \vee Z = 1$

Instrukcije bezuslovnog skoka su instrukcija bezuslovnog skoka (JMP) i instrukcija skoka na potprogram (JSR).

Instrukcija skoka je i instrukcija prekida (INT). Broj ulaza u tabelu sa adresama prekidnih rutina je 8-mo bitna celobrojna veličina bez znaka data 2. bajtom instrukcije. Dužina instrukcije je 2 bajta.

Adresne instrukcije su instrukcija prenosa u akumulator (LD), instrukcija prenosa iz akumulator (ST), aritmetička instrukcija sabiranja (ADD), aritmetička instrukcija oduzimanja (SUB), logička instrukcija I (AND), logička instrukcija ILI (OR), logička instrukcija ekskluzivno ILI (XOR), logička instrukcija invertovanja (NOT).

Spoljašnji maskirajući zahtevi za prekid dolaze od 3 ulazno/izlazna uređaja po linijama $intr_3$ do $intr_1$. Ovi prekidi se nazivaju spoljašnji prekidi jer dolaze od uređaja van procesora, kao i maskirajući prekidi, jer su dozvoljeni ili maskirani i procesor na njih reaguje ili ne reaguje u zavisnosti od toga da li se u razredu PSWI registra programske statusne reči $PSW_{15...0}$ nalazi vrednost 1 ili 0, respektivno. Dozvoljavanje i maskiranje prekida se realizuje programskim putem izvršavanjem instrukcija INTE i INTD kojima se u razred PSWI registra $PSW_{15...0}$ upisuju vrednosti 1 ili 0, respektivno. Zahtevi za prekid po linijama $intr_3$ do $intr_1$ se mogu selektivno maskirati razredima 3 do 1 registra maske

$IMR_{15...0}$. Prekidi koji dolaze po linijama $intr_3$ do $intr_1$ uređeni su po prioritetima pri čemu linija $intr_3$ ima najviši, a linija $intr_1$ najniži nivo prioriteta. Zahtevi za prekid se prihvataju ukoliko je njihov nivo viši od nivoa prioriteta tekućeg programa. Spoljašnji nemaskirajući zahtev za prekid dolazi po liniji inm. Unutrašnji prekidi su prekid zbog greške u kodu operacije, prekid zbog greške u adresiranju i prekid zbog zadatog režima rada prekid posle svake instrukcije.

Opsluživanje zahteva za prekid se sastoji iz dve grupe koraka.

U okviru prve grupe koraka na steku se čuvaju programski brojač $PC_{15...0}$ i programska statusna reč $PSW_{15...0}$, zatim se u razrede PSWI i PSWT programske statusne reči $PSW_{15...0}$ upisuju vrednosti 0 i u slučaju maskirajućih prekida u razrede PSWL1 i PSWL0 registra $PSW_{15...0}$ upisuje nivo prioriteta prekidne rutine na koju se skače. U okviru druge grupe koraka utvrđuje se adresa prekidne rutine. Utvrđivanje adrese prekidne rutine se realizuje na osnovu sadržaja tabele adresa prekidnih rutina, koja se obično naziva IV tabela (Interrupt Vector Table), i broja ulaza u IV tabelu. Stoga je u postupku inicijalizacije celog sistema u memoriji, počev od adrese na koju ukazuje sadržaj registra $IVTP_{15...0}$ (*Interrupt Vector Table Pointer*), kreirana IV tabela sa 8 ulaza, tako da se u ulazima 7 do 5 nalaze adrese prekidnih rutina za svaki od prekida koji dolaze po linijama $intr_3$ до $intr_1$, respektivno. Adrese prekidnih rutina za prekid zbog greške u kodu operacije, greške u adresiranju, spoljašnji nemaskirajući prekid i prekid zbog zadatog režima rada prekid posle svake instrukcije, nalaze se u ulazima 3, 2, 1 i 0 tabele sa adresama prekidnih rutina. Kako je memorijska reč 8 bitna veličina, a adresa prekidne rutine 16 bitna veličina, to svaki ulaz u IV tabeli zauzima po dve susedne memorijske lokacije. Zbog toga se najpre broj ulaza množenjem sa dva pretvara u pomeraj, pa zatim pomeraj sabira sa sadržajem registra $IVTP_{15...0}$ i na kraju dobijena vrednost koristi kao adresa sa koje se čita adresa prekidne rutine i upisuje u registar $PC_{15...0}$.

Magistrala je asinhrona sa atomskim ciklusima čitanja i upisa. Koristi se paralelni arbitrator sa 4 para linija za zahtev i potvrdu. Procesor i memorija imaju posebne signale takta, a postoji i poseban signal takta magistrale. Postoji i signal zauzeća magistrale.

Realizovati procesor, memoriju i arbitrator magistrale. Operaciona jedinica treba da bude realizovana sa dve magistrale. Upravljačku jedinicu procesora realizovati mikroprogramskom realizacijom sa mešovitim kodiranjem upravljačkih signala sa jednim tipom mikroinstrukcija.

2 KONFIGURACIJA SISTEMA

Računarski sistem sadrži sledeće module: procesor ***CPU***, memoriju ***MEM***, ulazno/izlazne uređaje ***U/I1*** do ***U/I3*** i uređaj za kontrolu ispravnosti rada sistema ***FAULT***. Procesor, memorija i ulazno/izlazni uređaji su međusobno povezani sistemskom magistralom ***BUS*** (slika **Error! Reference source not found.**). Moduli računarskog sistema rade asihrono.

Arhitektura procesora od programski dostupnih registara ima registre opšte namene R[0] do R[3]. Tipovi podataka sa kojima se radi su celobrojne 16-to bitne veličine sa znakom i bez znaka. Format instrukcija je jednoadresni. Načini adresiranja uključuju neposredno adresiranje (immed), memorijsko direktno adresiranje (memdir), registarsko indirektno sa pomerajem adresiranje (regindpom) i registarsko direktno adresiranje (regdir). Skup instrukcija uključuje bezadresne instrukcije, instrukcije uslovnog skoka, instrukcije bezuslovnog skoka i adresne instrukcije. Prekidi uključuju unutrašnje i spoljašnje prekide sa maskiranjem i prioritiranjem prekida, pri čemu se kontekst procesora se čuva na steku, a adresa prekidne rutine utvrđuje tehnikom vektorisanog mehanizma prekida.

Organizacija procesora je tako odabrana da je čine dve odvojene jedinice i to operaciona jedinica i upravljačka jedinica. Operaciona jedinica se sastoji od blokova za povezivanje na magistralu, čitanje instrukcije, formiranje adrese i čitanje operanda, izvršavanje operacija i opsluživanje prekida, međusobno povezanih sa dve magistrale. Upravljačka jedinica je realizovana tehnikom mikroprogramske realizacije sa mešovitim kodiranjem upravljačkih signala sa jednim tipom mikroinstrukcija.

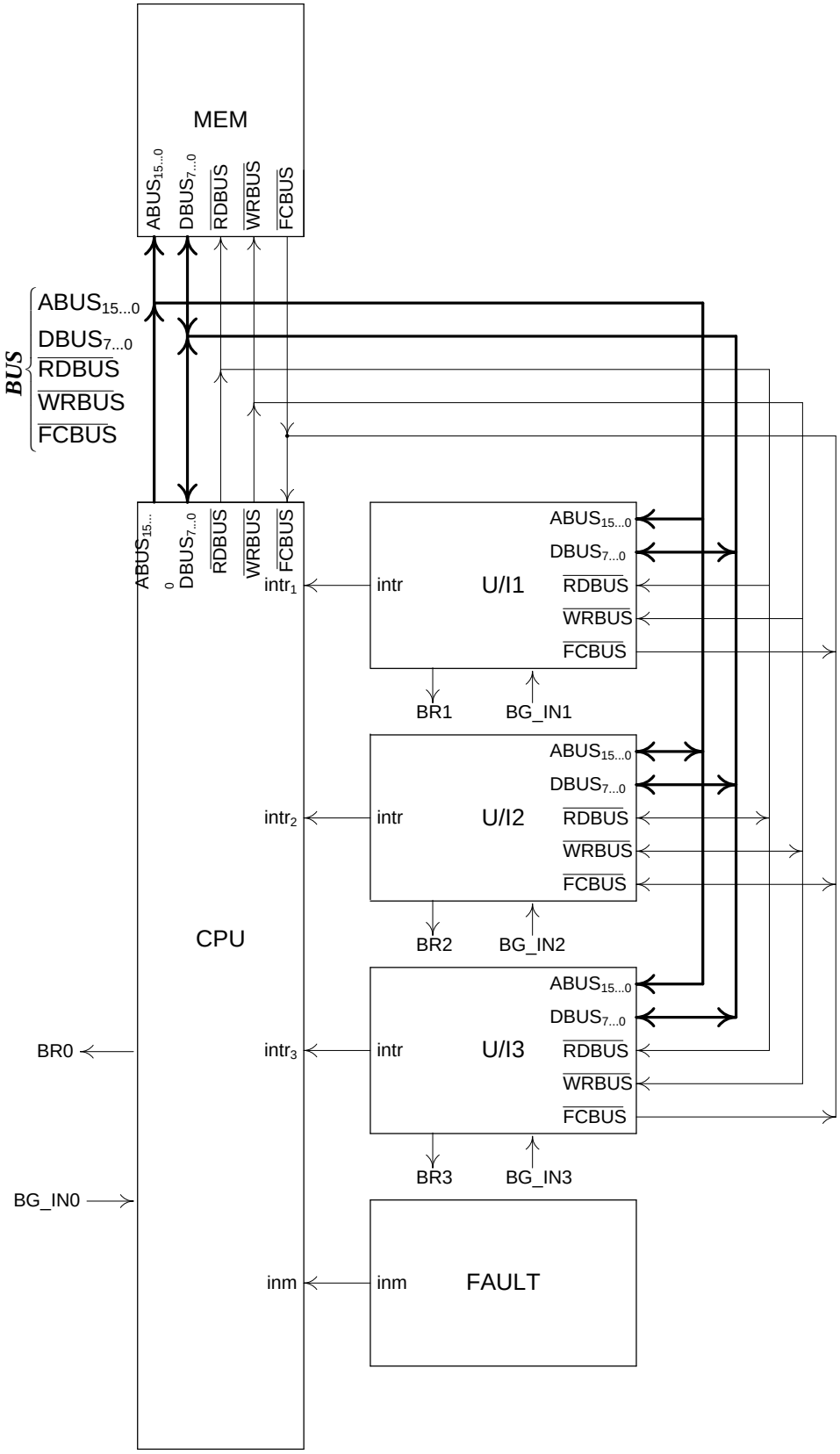
Procesor po linijama $\mathbf{intr_1}$ do $\mathbf{intr_3}$ dobija signale maskirajućih zahteva za prekid od ulazno/izlaznih uređaja ***U/I1*** do ***U/I3***, respektivno. Procesor po liniji **inm** dobija signal nemaskirajućeg zahteva za prekid od uređaja za kontrolu ispravnosti rada sistema ***FAULT***.

Memorija je kapaciteta 64K 8–mo bitnih reči.

Ulazno/izlaznih uređaja ima 3. Ulazno/izlazni uređaji se sastoje od periferije i kontrolera za direktan pristup memoriji povezanih paralelnim interfejsom.

Uređaj za kontrolu ispravnosti rada sistema ***FAULT*** generiše nemaskirajući prekid **inm** u slučaju otkrivanja neke neispravnosti.

Procesor, memorija i ulazno/izlazni uređaji su povezani asinhronom sistemskom magistralom koju čine adresne linije $ABUS_{15...0}$, linije podataka $DBUS_{7...0}$ i upravljačke linije $\overline{\mathbf{RDBUS}}$, $\overline{\mathbf{WRBUS}}$ i $\overline{\mathbf{FCBUS}}$. Na magistrali mogu da se realizuju ciklus čitanja i ciklus upisa. Procesor realizuje cikluse čitanja prilikom čitanja instrukcija i podataka iz memorije i prilikom čitanja statusnih informacija i podataka iz registara kontrolera ulazno/izlaznih uređaja. Procesor realizuje cikluse upisa prilikom upisa podataka u memoriju i prilikom upisa upravljačkih informacija i podataka u registre kontrolera ulazno/izlaznih uređaja. Ulazno/izlazni uređaj sa kontrolerom za direktan pristup memoriji realizuje ciklus čitanja sa memorijom prilikom prenosa iz memorije u uređaj i ciklus upisa sa memorijom prilikom prenosa iz uređaja u memoriju.



Slika 1 Konfiguracija sistema

Uređaj koji započinje neki ciklusa na magistrali naziva se gazda, a uređaj koji realizuje ciklus sluga. Pri ciklusu čitanja gazda šalje adresu na adresne linije $ABUS_{15...0}$ i signalom na upravljačkoj liniji $\overline{\textbf{RDBUS}}$ startuje čitanje u slugi. Po završenom čitanju sluga šalje očitani podatak na linije podataka $DBUS_{7...0}$ i signalom na upravljačkoj liniji $\overline{\textbf{FCBUS}}$ signalizira gazdi da je čitanje završeno i da je podatak raspoloživ. Pri ciklusu upisa gazda šalje adresu na adresne linije $ABUS_{15...0}$, podatak na linije podataka $DBUS_{17...0}$ i signalom na upravljačkoj liniji $\overline{\textbf{WRBUS}}$ startuje upis u slugi. Po završenom upisu sluga signalom na upravljačkoj liniji $\overline{\textbf{FCBUS}}$ signalizira gazdi da je upis završen.

Kako gazde na magistrali mogu da budu procesor i ulazno/izlazni uređaj sa kontrolerom za direktan pristup memoriji, postoji potreba za arbitracijom među njima. U tu svrhu se koristi paralelni arbitrator sa 4 para linija za zahtev i potvrdu. Procesor ili ulazno/uzlazni uređaj pre realizacije ciklusa na magistrali u kome je gazda signalom **BR** traži dozvolu korišćenja magistrale od arbitratora, a arbitrator mu signalom **BG_IN** daje dozvolu.

3 ARHITEKTURA SISTEMA

U ovoj glavi se razmatra arhitektura računarskog sistema. Pri tome se pod arhitekturom računarskog sistema podrazumevaju svi oni njegovi elementi koji treba da budu poznati da bi za njega mogao da se napiše asemblerski program koji će se uspešno izvršavati i uvek davati isti rezultat bez obzira na to kako je dati računarski sistem realizovan. U okviru arhitekture računarskog sistema razmatraju se arhitekture procesora, memorije i ulazno/izlaznih uređaja.

3.1 PROCESOR

Arhitekturu procesora čine:

- programski dostupni registri,
- tipovi podataka,
- formati instrukcija,
- načini adresiranja,
- skup instrukcija i
- mehanizam prekida.

U daljem tekstu biće razmotren svaki od ovih elemenata arhitekture procesora.

3.1.1 Programski dostupni registri

Programski dostupni registri procesora su:

- programski brojač PC,
- akumulator A,
- 4 registra opšte namene GPR,
- ukazivač na vrh steka SP,
- programska statusna reč PSW,
- registar maske IMR i
- ukazivač na tabelu adresa prekidnih rutina IVTP.

Registar PC je standardni 16-to razredni programski brojač procesora. Adrese generisane na osnovu vrednosti registra PC su adrese 8-mo bitnih reči. Njime se mogu generisati adrese unutar adresnog prostora od 2^{16} 8-mo bitnih reči.

Registar A je 16-to razredni akumulator za operacije prenosa 16-to bitnih veličina u kojima se koristi kao implicitno izvorište ili odredište operanda.

Četiri registra opšte namene (GPR) su 16-to razredni registri koji se koriste kao registri podataka kod registarskog indirektnog adresiranja sa pomerajem, kao i kod direktnog registarskog adresiranja.

Registar SP je standardan 16-to razredni ukazivač na vrh steka. Stek je organizovan u operativnoj memoriji i raste prema višim adresama. Registar SP ukazuje na prvu slobodnu lokaciju na steku. Adrese generisane na osnovu vrednosti registra SP su adrese 8-mo bitnih reči. Njime se mogu generisati adrese unutar adresnog prostora od 2^{16} 8-mo bitnih reči.

Registar PSW je standardna 16-to razredni programska statusna reč procesora čiji razredi, koji se obično nazivaju indikatori, sadrže bitove statusnog i upravljačkog karaktera (slika Slika 2 Struktura registra PSW).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	T	-	-	-	-	-	-	-	L_2	L_1	L_0	V	C	Z	N

Slika 2 Struktura registra PSW

Bitovi statusnog karaktera su:

- N—bit (indikator *negative*) koji se postavlja na 1 kada je rezultat operacije negativan,
- Z—bit (indikator *zero*) koji se postavlja na 1 kada je rezultat operacije nula,
- C—bit (indikator *carry/borrow*) koji se postavlja na 1 kada je takav rezultat operacije da postoji prenos/pozajmica u aritmetici celobrojnih veličina bez znaka,
- V—bit (indikator *overflow*) koji se postavlja na 1 kada je takav rezultat operacije da postoji prekoračenje u aritmetici celobrojnih veličina sa znakom i
- L_2, L_1, L_0—bitovi (indikatori *level*) kojima se pamti nivo prioriteta tekućeg programa.

Bitovi upravljačkog karaktera su:

- T—bit (indikator *trap*) koji se postavlja na 1 kada se zada režim rada procesora sa prekidom posle svake instrukcije i
- I—bit (indikator *interrupt*) koji se postavlja na 1 kada se zada režim rada procesora sa prihvatanjem maskirajućih prekida.

Bitovi statusnog karaktera N, Z, C i V se postavljaju hardverski na osnovu rezultata izvršavanja instrukcija. Bitovi statusnog karaktera L_2 do L_0 se postavljaju hardverski u okviru opsluživanja prekida i softverski kao rezultat izvršavanja instrukcije povratak iz prekidne rutine. Bitovi upravljačkog karaktera I i T se postavljaju softverski kao rezultat izvršavanja posebnih instrukcija, hardverski u okviru opsluživanja prekida i softverski kao rezultat izvršavanja instrukcije povratak iz prekidne rutine.

Registar IMR je standardni 16-to razredni registar maske za selektivno maskiranje maskirajućih prekida. Koriste se samo razredi 3 do 1 koji su dodeljeni prekidima koji dolaze po linijama 3 do 1 od ulazno/izlaznih uređaja, respektivno.

Registar IVTP je standardni 16-to razredni ukazivač na tabelu adresa prekidnih rutina koja se koristi u okviru vektorisanog mehanizma prekida. Adresa generisana na osnovu vrednosti registra IVTP je adresa 8-mo bitne reči.

3.1.2 Tipovi podataka

Tipovi podataka koji se koriste u ovom procesoru su celobrojne 16-to bitne veličine sa znakom i bez znaka. Celobrojne veličine bez znaka su u opsegu od 0 do 2^{16}–1. Celobrojne veličine sa znakom su u opsegu od -2^{15} do 2^{15}-1.

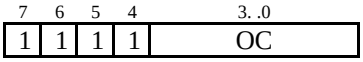
3.1.3 Formati instrukcija

U procesoru se koriste sledeći formati instrukcija:

- format bezadresnih instrukcija,
- format instrukcije prekida,
- format instrukcija relativnog skoka – B format,
- format instrukcija apsolutnog skoka – J format,
- format jednoadresnih registarskih instrukcija – R format,
- format jednoadresnih registarskih instrukcija – RB format,
- format jednoadresnih neposrednih instrukcija – IW format i
- format jednoadresnih memorijskih instrukcija – AP format.

3.1.3.1 Format bezadresnih instrukcija

Format ovih instrukcija je dat na slici Slika 3.



Slika 3 Format bezadresnih instrukcija

Poljem OC prvog bajta instrukcije specificira se kod operacije za bezadresne instrukcije. Dužina instrukcije je jedan bajt.

Ovaj format imaju sledeće instrukcije: RTS, RTI, ASR, LSR, ROR, RORC, ASL, LSL, ROL, ROLC, INTE, INTD, TRPE,TRPD, STIVTP, STSP.

3.1.3.2 Format instrukcije prekida

Format ove instrukcije je dat na slici Slika 4.

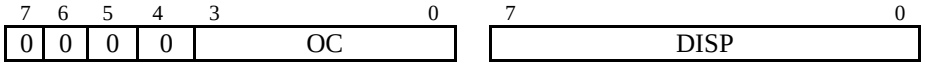
Slika 4 Format instrukcije prekida

Poljem OC se specificira operacija prekida, a poljem BR broj ulaza u tabelu sa adresama prekidnih rutina. Broj ulaza je 8-mo bitna celobrojna veličina bez znaka data drugim bajtom instrukcije. Dužina instrukcije je 2 bajta.

Ovaj format ima instrukcija INT.

3.1.3.3 Format instrukcija relativnog skoka – B format

Format ovih instrukcija je dat na slici Slika 5.



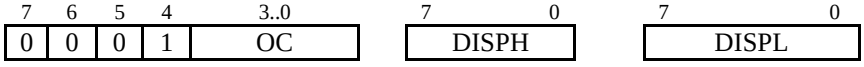
Slika 5 Format instrukcija relativnog skoka – B format

Poljem OC se specificira operacija koja se izvršava, a poljem DISP pomeraj koji se sabira sa PC da bi se dobila adresa skoka. Pomeraj je 8-mo bitna celobrojna veličina sa znakom. Dužina instrukcije je 2 bajta.

Ovaj format imaju sledeće instrukcije: BEQL, BNEW, BNEG, BNNG, BOVF, BNVF, BCR, BNCR, BGRT, BGRE, BLSS, BLEQ, BGRTU, BGREU, BLSSU, BLEQU.

3.1.3.4 Format instrukcija apslolutnog skoka – J format

Format ovih instrukcija je dat na slici Slika 6.



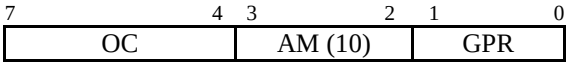
Slika 6 Format instrukcija relativnog skoka – J format

Poljem OC se specificira operacija koja se izvršava, a poljima DISPH i DISPL se definišu 8 starijih i 8 mlađih bitova 16-to bitne adrese skoka, respektivno. Dužina instrukcije je 3 bajta.

Ovaj format imaju instrukcije bezuslovnog skoka (JMP) i skoka na potprogram (JSR). Ove instrukcije se realizuju kao apsolutni skokovi.

3.1.3.5 Format jednoadresnih registarskih instrukcija – R format

Format ovih instrukcija je dat na slici Slika 7.



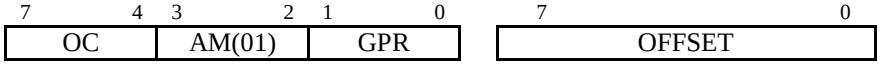
Slika 7 Format jednoadresnih registarskih instrukcija – R format

Dužina instrukcije je 1 bajt. Poljem OC(od 0010 do 1110) se specificira kod operacije jednoadresne instrukcije, poljem AM registarsko direktno (10) i poljem GPR jedan od 4 registra opšte namene – R0 do R3.

Ovaj format mogu imati instrukcije: LD, ST, ADD, SUB, AND, OR, XOR i NOT.

3.1.3.6 Format jednoadresnih registarskih instrukcija – RB format

Format ovih instrukcija je dat na slici Slika 8.



Slika 8 Format jednoadresnih instrukcija – RB format

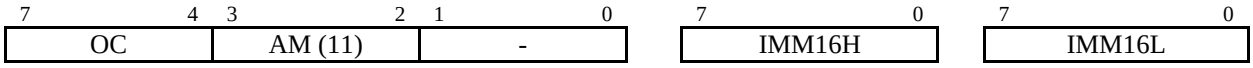
Dužina instrukcije je 2 bajta. Poljem OC (od 0010 do 1110) se specificira kod operacije jednoadresne instrukcije, poljem AM(01) registarsko indirektno adresiranje sa pomerajem (10) i poljem GPR jedan od 4 registra opšte namene – R0 do R3.

Drugim bajtom instrukcije, poljem OFFSET, se zadaje celobrojna veličina sa znakom u drugom komplementu koja predstavlja pomeraj.

Ovaj format mogu imati instrukcije: LD, ST, ADD, SUB, AND, OR, XOR i NOT.

3.1.3.7 Format jednoadresnih neposrednih instrukcija - IW format

Format ovih instrukcija je dat na slici Slika 9.



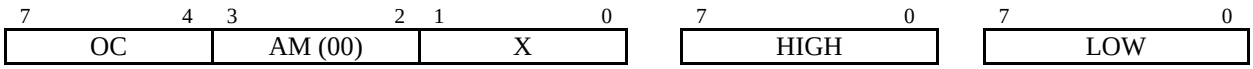
Slika 9 Format jednoadresnih neposrednih instrukcija – IW format

Poljem OC se specificira kod operacije jednoadresne instrukcije nad 16-to bitnim veličinama, polje AM (11) neposredno adresiranje, polje - se ne koriste i poljima IMM16H i IMM16L 8 starijih i 8 mlađih bitova neposredne 16-to bitne veličine.

Ovaj format mogu imati instrukcije: LD, ADD, SUB, AND, OR, XOR i NOT.

3.1.3.8 Format jednoadresnih memorijskih instrukcija - AP format

Format ovih instrukcija je dat na slici Slika 10 .



Slika 10 Format jednoadresnih memorijskih instrukcija – AP format

Poljem OC se specificira kod operacije jednoadresne instrukcije nad 16-to bitnim veličinama, polje AM (00) memorijsko direktno. Polja HIGH i LOW predstavljaju 8 starijih i 8 mlađih bitova 16-to bitne veličine koja predstavlja memorijsku adresu za memorijsko direktno adresiranje.

Ovaj format mogu imati instrukcije: LD, ST, ADD, SUB, AND, OR, XOR i NOT.

3.1.4 Načini adresiranja

Procesor poseduje 4 različitih načina adresiranja. Kodiranje polja AM za različite načine adresiranja i nazivi načina adresiranja dati su u tabeli Tabela 1.

AM	Načini adresiranja
00	memorijsko direktno
01	registarsko indirektno sa pomerajem
10	registarsko direktno
11	neposredno

Tabela 1 Načini adresiranja

Memorijsko direktno adresiranje je adresiranje kod koga se operand nalazi u memoriji na adresi datoj u samoj instrukciji. Instrukcija sa registarskim direktnim adresiranjem ima AP format (poglavlje 3.1.3.8). Polje GPR se ne koristi. Polja HIGH i LOW sadrže starijih 8 i mlađih 8 bita adrese.

Registarsko indirektno adresiranje sa pomerajem je adresiranje kod koga se operand nalazi u jednoj od memorijskih lokacija, a adresa memorijske lokacije se dobija sabiranjem sadržaja jednog od registara opšte namene i pomeraja. Instrukcija sa registarskim indirektnim adresiranjem sa pomerajem ima RB format (poglavlje 3.1.3.6). Drugim bajtom instrukcije, poljem OFFSET, se zadaje celobrojna veličina sa znakom u drugom komplementu koja predstavlja pomeraj.

Registarsko direktno adresiranje je adresiranje kod koga se operand nalazi u jednom od registara opšte namene. Instrukcija sa registarskim direktnim adresiranjem ima R format (poglavlje 3.1.3.5). Registar opšte namene je specificiran poljem GPR. Koristi se svih 16 razreda registra.

Neposredno adresiranje je adresiranje kod koga se operand nalazi u samoj instrukciji. Instrukcija sa neposrednim adresiranjem ima IW format (poglavlje 3.1.3.7). Polje GPR se ne koristi. Polja IMM16H i IMM16L sadrže starijih 8 i mlađih 8 bitova 16-to bitnog podatka.

3.1.5 Skup instrukcija

U ovom poglavlju se, najpre, daje opis instrukcija, a zatim tabelarni pregled kodiranja instrukcija.

3.1.5.1 Opis instrukcija

Instrukcije procesora se mogu svrstati u sledećih osam grupa:

- instrukcija bez dejstva,
- instrukcija zaustavljanja,
- instrukcije skoka,
- instrukcije prenosa,
- aritmetičke instrukcije,
- logičke instrukcije,
- instrukcije pomeranja i rotiranja,
- instrukcije postavljanja indikatora u PSW,

3.1.5.1.1 Instrukcija bez dejstva

Instrukcija **NOP** ne proizvodi nikakvo dejstvo. Format ove instrukcije je dat u poglavlju 3.1.3.1. Ova instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

3.1.5.1.2 Instrukcija zaustavljanja

Instrukcija **HALT** zaustavlja izvršavanje instrukcija. Format ove instrukcije je dat u poglavlju 3.1.3.1. Ova instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

3.1.5.1.3 Instrukcije skoka

Instrukcije skoka se svrstavaju u sledeće grupe:

- instrukcije uslovnog skoka,
- instrukcije bezuslovnog skoka,
- instrukcije skoka na potprogram i povratka iz potprograma i
- instrukcija prekida i povratka iz prekidne rutine

3.1.5.1.3.1 Instrukcije uslovnog skoka

Instrukcije uslovnog skoka **BEQL** *disp*, **BNEQL** *disp*, **BNEG** *disp*, **BNNEG** *disp*, **BOVF** *disp*, **BNOVF** *disp*, **BCAR** *disp*, **BNCAR** *disp*, **BGRT** *disp*, **BGRTE** *disp*, **BLSS** *disp*, **BLSSE** *disp*, **BGRTU** *disp*, **BGRTEU** *disp*, **BLSSU** *disp* i **BLSSEU** *disp* realizuju relativni skok sa pomerajem *disp* u odnosu na programski brojač PC ukoliko je uslov specificiran kodom operacije ispunjen (tabela Tabela 2). Pomeraj *disp* je 8-mo bitna celobrojna veličina sa znakom. Format ovih instrukcija je dat u poglavlju 3.1.3.3. Ni jedna od ovih instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Tabela 2 Instrukcije uslovnog skoka

Instrukcija	Značenje	Uslov
BEQL	skok na jednako	Z = 1
BNEQ	skok na nejednako	Z = 0
BNEG	skok na N = 1	N = 1
BNNG	skok na N = 0	N = 0
BOVF	skok na V = 1	V = 1
BNVF	skok na V = 0	V = 0
BCR	skok na C = 1	C = 1
BNCR	skok na C = 0	C = 0
BGRT	skok na veće nego (sa znakom)	$(N \oplus V) \vee Z = 0$
BGRE	skok na veće nego ili jednako (sa znakom)	$N \oplus V = 0$
BLSS	skok na manje nego (sa znakom)	$(N \oplus V) = 1$
BLEQ	skok na manje nego ili jednako (sa znakom)	$(N \oplus V) \vee Z = 1$
BGRTU	skok na veće nego (bez znaka)	$C \vee Z = 0$
BGREU	skok na veće nego ili jednako (bez znaka)	C = 0
BLSSU	skok na manje nego (bez znaka)	C = 1
BLEQU	skok na manje nego ili jednako (bez znaka)	$C \vee Z = 1$

3.1.5.1.3.2 Instrukcija bezuslovnog skoka

Instrukcija bezuslovnog skoka **JMP** *a* realizuje skok na adresu *a* koja je data u samoj instrukciji. Format ove instrukcije je dat u poglavlju 3.1.3.4. Ova instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

3.1.5.1.3.3 Instrukcije skoka na potprogram i povratka iz potprograma

Instrukcija **JSR** *a* realizuje skok na potprogram tako što čuva vrednost programskog brojača PC na steku i realizuje skok na adresu *a* koja je data u samoj instrukciji. Format ove instrukcije je dat u odeljku 3.1.3.4.

Instrukcija **RTS** realizuje povratak iz potprograma tako što restaurira vrednost programskog brojača PC vrednošću sa steka. Format ove instrukcije je dat u poglavlju 3.1.3.1.

Ni jedna od ovih instrukcija ne utiče na indikatore N, Z, C i V iz registra PSW.

3.1.5.1.3.4 Instrukcije prekida i povratka iz prekidne rutine

Instrukcija **INT** *br* programskim putem realizuje prekid i skok na prekidnu rutinu čija se adresa nalazi u ulazu *br* tabele sa adresama prekidnih rutina. Ulaz *br* je 8-mo bitna celobrojna veličina bez znaka. Format ove instrukcije je dat u poglavlju 3.1.3.2.

Instrukcija **RTI** realizuje povratak iz prekidne rutine tako što restaurira vrednosti programske statusne reči PSW i programskog brojača PC vrednostima sa steka. Format ove instrukcije je dat u poglavlju 3.1.3.1.

Ni jedna od ovih instrukcija ne utiče na indikatore N, Z, C i V iz registra PSW.

3.1.5.1.4 Instrukcije prenosa

Instrukcija **LD** *src* prenosi sadržaj 16-bitnog operanda *src* u akumulator A. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavlju Formati instrukcija 3.1.3. . Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **ST** *dst* prenosi sadržaj akumulatora A u 16-bitni operand *dst*. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavlju Formati instrukcija 3.1.3. .Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **STIVTP** prenosi sadržaj akumulatora A u registar IVTP. Format ove instrukcije i dat je u poglavlju Formati instrukcija 3.1.3. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **STSP** prenosi sadržaj akumulatora A u registar SP. Format ove instrukcije dat je u poglavlju Formati instrukcija 3.1.3. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

3.1.5.1.5 Aritmetičke instrukcije

Instrukcija **ADD** *src* sabira celobrojnu 16-bitnu veličinu koja se nalazi u akumulatoru A sa operandom *src* koji je celobrojna 16-bitna veličina, a rezultat koji je celobrojna 16-bitna veličina smešta u akumulator A. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavlju Formati instrukcija 3.1.3.

Instrukcija **SUB** *src* oduzima operand *src* koji je celobrojna 16-bitna veličina od celobrojne 8-bitne veličine koja se nalazi u akumulatoru A, a rezultat koji je celobrojna 16-bitna veličina smešta u akumulator A. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavlju Formati instrukcija 3.1.3.

Istrukcije postavljaju indikatore N, Z, C i V registra PSW saglasno dobijenoj vrednosti koja se upisuje u akumulator A.

3.1.5.1.6 Logičke instrukcije

Instrukcija **AND** *src* izračunava logičko I 16-bitne veličine koja se nalazi u akumulatoru A i operanda *src* koji je 16-bitna veličina, a rezultat koji je 16-bitna veličina smešta u akumulator A. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavlju Formati instrukcija 3.1.3.

Instrukcija **OR** *src* izračunava logičko ILI 16-bitne veličine koja se nalazi u akumulatoru A i operanda *src* koji je 16-bitna veličina, a rezultat koji je 16-bitna veličina smešta u akumulator A. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavlju Formati instrukcija 3.1.3.

Instrukcija **XOR** *src* izračunava logičko EKSLUZIVNO ILI 16-bitne veličine koja se nalazi u akumulatoru A i operanda *src* koji je 16-bitna veličina, a rezultat koji je 16-bitna veličina smešta u akumulator A. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavlju Formati instrukcija 3.1.3.

Instrukcija **NOT** izračunava logičku NEGACIJU 16-bitne veličine koja se nalazi u akumulatoru A, a rezultat koji je 16-bitna veličina smešta u akumulator A. Format instrukcije i dat je u poglavlju Formati instrukcija 3.1.3.

Istrukcije postavljaju indikatore N i Z registra PSW saglasno vrednosti koja se upisuje u akumulator AB, dok se indikatori C i V registra PSW postavljaju na 0.

3.1.5.1.7 Instrukcije pomeranja i rotiranja

Instrukcija **ASR** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A udesno za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju razred A_{15} ostaje nepromenjen, a u indikator C registra PSW se upisuje sadržaj razreda A_0. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **LSR** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A udesno za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju u razred A_{15} se upisuje 0, a u indikator C registra PSW se upisuje sadržaj razreda A_0. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **ROR** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A udesno za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju u razred A_{15} se upisuje sadržaj razreda A_0, a u indikator C registra PSW se upisuje sadržaj razreda AB_0. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **RORC** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A udesno za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju u razred A_{15} se upisuje sadržaj indikatora C registra PSW, a u indikator C registra PSW se upisuje sadržaj razreda A_0. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **ASL** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A ulevo za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju u razred AB_0 se upisuje 0, a u indikator C registra PSW se upisuje sadržaj razreda A_{15}. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3..

Instrukcija **LSL** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A ulevo za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju u razred A_0 se upisuje 0, a u indikator C registra PSW se upisuje sadržaj razreda A_{15}. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.Instrukcija **ROL** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A ulevo za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju u razred A_0 se upisuje sadržaj razreda A_{15}, a u indikator C registra PSW se upisuje sadržaj razreda A_{15}. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **ROLC** pomera sadržaj 16-bitne veličine koja se nalazi u akumulatoru A ulevo za jedno mesto, a rezultat koji je 16-bitna veličina smešta u akumulator A. Pri pomeranju u razred A_0 se upisuje sadržaj indikatora C registra PSW, a u indikator C registra PSW se upisuje sadržaj razreda A_{15}. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Istrukcije postavljaju indikatore N i Z registra PSW saglasno vrednosti koja se upisuje u akumulator A, dok se indikator V registra PSW postavlja na 0.

3.1.5.1.8 Instrukcije postavljanja indikatora u PSW

Instrukcija **INTD** postavlja nulu u razred I registra PSW. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **INTE** postavlja jedinicu u razred I registra PSW. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **TRPD** postavlja nulu u razred T registra PSW. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Instrukcija **TRPE** postavlja jedinicu u razred T registra PSW. Format instrukcije dat je u poglavlju Formati instrukcija 3.1.3.

Ni jedna od ovih instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

3.1.5.2 Kodiranje instrukcija

Kodiranje podgrupe bezadresnih instrukcija je dato u tabeli Tabela 3 . Dužina ove grupe instrukcija je 1 bajt. Sam kod operacije je dat prvim bajtom.

Tabela 3 Kodiranje bezadresnih instrukcija

7	6	5	4	3	2	1	0	**Oznaka**
1	1	1	1	0	0	0	0	**RTS**
1	1	1	1	0	0	0	1	**RTI**
1	1	1	1	0	0	1	0	**ASR**
1	1	1	1	0	0	1	1	**LSR**
1	1	1	1	0	1	0	0	**ROR**
1	1	1	1	0	1	0	1	**RORC**
1	1	1	1	0	1	1	0	**ASL**
1	1	1	1	0	1	1	1	**LSL**
1	1	1	1	1	0	0	0	**ROL**
1	1	1	1	1	0	0	1	**ROLC**
1	1	1	1	1	0	1	0	**INTE**
1	1	1	1	1	0	1	1	**INTD**
1	1	1	1	1	1	0	0	**TRPE**
1	1	1	1	1	1	0	1	**TRPD**
1	1	1	1	1	1	1	0	**STIVTP**
1	1	1	1	1	1	1	1	**STSP**

Kodiranje podgrupe instrukcija uslovih skokova je dato u tabeli Tabela 4. Dužina ove grupe instrukcije je 2 bajta. Kod operacije je dat u okviru prvog bajta instrukcije.

Tabela 4 Kodiranje instrukcija uslovnog skoka

7	6	5	4	3	2	1	0	**Oznaka**
0	0	0	0	0	0	0	0	**BEQL**
0	0	0	0	0	0	0	1	**BNEQ**
0	0	0	0	0	0	1	0	**BNEG**
0	0	0	0	0	0	1	1	**BNNG**
0	0	0	0	0	1	0	0	**BOVF**
0	0	0	0	0	1	0	1	**BNVF**
0	0	0	0	0	1	1	0	**BCR**
0	0	0	0	0	1	1	1	**BNCR**
0	0	0	0	1	0	0	0	**BGRT**
0	0	0	0	1	0	0	1	**BGRE**
0	0	0	0	1	0	1	0	**BLSS**
0	0	0	0	1	0	1	1	**BLEQ**
0	0	0	0	1	1	0	0	**BGRTU**
0	0	0	0	1	1	0	1	**BGREU**
0	0	0	0	1	1	1	0	**BLSSU**
0	0	0	0	1	1	1	1	**BLEQU**

Kodiranje podgrupe instrukcija bezuslovih skokova je dato u tabeli Tabela 5. Dužina ove grupe instrukcija je tri bajta, sa izuzetkom instrukcije INT koja ima dužinu dva bajta. Kod operacije je dat u okviru prvog bajta instrukcije.

Tabela 5 Kodiranje instrukcija bezuslovnog skoka

7	6	5	4	3	2	1	0	**Oznaka**
0	0	0	1	0	0	0	0	**JMP**
0	0	0	1	0	0	0	1	**JSR**
0	0	0	1	0	0	1	0	**INT**

Kodiranje podgrupe adresnih instrukcija je dato u tabeli Tabela 6. Dužina ove grupe instrukcije zavisi od načina adresiranja i može biti jedan, dva ili tri bajta. Kod operacije je dat u okviru prvog bajta instrukcije.

Tabela 6 Kodiranje adresnih instrukcija

7	6	5	4	**Oznaka**
0	0	1	0	**LD**
0	0	1	1	**ST**
0	1	0	0	**ADD**
0	1	0	1	**SUB**
0	1	1	0	**AND**
0	1	1	1	**OR**
1	0	0	0	**XOR**
1	0	0	1	**NOT**

3.1.6 Mehanizam prekida

Zahteve za prekid mogu da generišu:
① tri kontrolera periferija po linijama intr3 do $intr_1$ da bi signalizirali spremnost za prenos podataka (maskirajući prekidi),
② jedan uređaj računara koji kontroliše ispravnost napona napajanja (nemaskirajući prekidi),
③ procesor, kao rezultat otkrivene nekorektnosti u izvršavanju tekuće instrukcije (nelegalan kod operacije i nelegalno adresiranje),
④ procesor, ako je zadat takav režim rada procesora, kroz postavljanje indikatora T u programskoj statusnoj reči PSW, da se posle svake instrukcije skače na određenu prekidnu rutinu i
⑤ procesor kao rezultat izvršavanja instrukcije prekida INT.
Prekidi pod ① i ② se nazivaju spoljašnji, a pod ③, ④ i ⑤ unutrašnji.

Izvršavanje svake instrukcije pored faza čitanje instrukcije, formiranje adrese operanda i izvršavanje operacije sadrži i fazu opsluživanje zahteva za prekid. Na početku faze opsluživanje zahteva za prekid vrši se provera da li je u toku izvršavanja prethodne tri faze tekuće instrukcije generisan neki od navedenih zahteva za prekid. Ukoliko nije, vraća se na fazu čitanje instrukcije sledeće instrukcije. Ukoliko jeste, izvršavanje tekuće instrukcije se produžava sa koracima faze opsluživanje zahteva za prekid. Opsluživanje zahteva za prekid se sastoji iz tri grupe koraka.

U okviru prve grupe koraka na steku se čuvaju programski brojač PC i programska statusna reči PSW.

U okviru druge grupe koraka utvrđuje se adresa prekidne rutine. Utvrđivanje adrese prekidne rutine se realizuje na osnovu sadržaja tabele adresa prekidnih rutina (IV tabela) i broja ulaza u IV tabelu. Stoga je u postupku inicijalizacije celog sistema u memoriji, počev od adrese na koju ukazuje sadržaj registra procesora IVTP (*Interrupt Vector Table Pointer*), kreirana IV tabela sa 8 ulaza u kojima se nalaze adrese prekidnih rutina za sve vrste prekida. Broj ulaza u IV tabelu se dobija na više načina i to:

- predstavlja fiksnu vrednost za prekide iz tačke ① i generiše ga sam procesor,
- predstavlja fiksnu vrednost za prekide iz tačaka ②, ③ i ④ i generiše ga sam procesor i
- specificiran je adresnim delom instrukcije INT za prekid iz tačke ⑤.

Ulazi 0 do 3 i 5 do 7 u IV tabeli su rezrvisani za adrese prekidnih rutina za sledeće vrste prekida:

0 – prekid zbog režima rada sa prekidom posle svake instrukcije,
1 – spoljasni nemaskirajući prekid,
2 – prekid zbog greške u adresiranju,
3 – prekid zbog greške u kodu operacije,
5 do 7 – maskirajući prekidi po linijama $intr_1$ do $intr_3$, respektivno.
Ulaz 4 u IV tabeli je slobodan za adrese prekidnih rutina za prekide izazvane instrukcijom INT.

Više prekida se može javiti istovremeno, a može se prihvatiti samo jedan zahtev za prekid i skočiti na njegovu prekidnu rutinu. Zato su zahtevima za prekid dodeljeni prioriteti, pa se od generisanih zahteva za prekid prihvata onaj zahtev za prekid koji je među njima najvišeg prioriteta. Redosled zahteva za prekid po opadajućim prioritetima je sledeći: najviši je ⑤, zatim redom ③, ②, ① i na kraju najniži je ④. Prekidi pod ① koji dolaze od kontrolera periferija mogu se javiti istovremeno pa se i oni prihvataju po redosledu opadajućih prioriteta. Pošto svaki kontroler periferije ima posebnu liniju u procesoru za slanje svog zahteva za prekid, na osnovu pozicije linije se određuje prioritet datog zahteva za prekid. Zbog toga se radi određivanja broja ulaza, po opadajućim proritetima redom proveravaju generisani zahtevi za prekid i utvrđuje koji je to zahtev za prekid koji se u tekućoj instrukciji prihvata i na osnovu toga određuje broj ulaza. – $intr_3$ ima najviši prioritet, zatim $intr_2$, a $intr_1$ ima najniži prioritet.

Kako je memorijska reč 8-mo bitna veličina a adresa prekidne rutine 16-to bitna veličina, to svaki ulaz u IV tabeli zauzima po dve susedne memorijske lokacije. Zbog toga se najpre broj ulaza množenjem sa dva pretvara u pomeraj, pa zatim pomeraj sabira sa sadržajem registra IVTP i na kraju dobijena vrednost koristiti kao adresu sa koje se čita adresa prekidne rutine i upisuje u registar PC.

U okviru treće grupe koraka se:
- brišu indikatori I i T registra PSW kod prekida svih vrsta i
- upisuje u bitove L_2 do L_0 registra PSW nivo prioriteta prekidne rutine na koju se skače u slučaju maskirajućeg prekida.

Povratak iz prekidne rutine se realizuje instrukcijom **RTI**. Ovom instrukcijom se sa steka restauriraju registri PSW i PC.

Za maskirajuće prekide postoji u procesoru poseban registar IMR koji se naziva registar maske. Od 16 razreda ovog registra koristi se samo tri. Svakoj liniji po kojoj mogu da se šalju zahtevi za prekid od periferija pridružen je poseban razred registra maske. Zahtev za prekid koji stiže po određenoj liniji u procesoru će biti opslužen jedino ukoliko se u odgovarajućem razredu registra maske nalazi vrednost 1. Instrukcijom **STIMR** se u registar maske IMR upisuje odgovarajuća vrednost. Time se programskim putem selektivno dozvoljava ili zabranjuje opsluživanje maskirajućih prekida.

Maskirajući zahtevi za prekid, bez obzira na to da li su selektivno maskirani sadržajem registra maske ili ne, mogu se svi maskirati bitom maske I u registru PSW. Instrukcijama **INTE** i **INTD** u ovaj razred registra PSW upisuju se vrednosti 1 ili 0, respektivno. Time se programski putem dozvoljava ili zabranjuje opsluživanje maskirajućih prekida koji nisu selektivno maskirani sadržajem registra maske IMR.

Postoji mogućnost da se zada takav režim rada procesora da se posle svake izvršene instrukcije skače na određenu prekidnu rutinu. Ovakav režim rada procesora se naziva prekid posle svake instrukcije. U njemu se procesor nalazi ukoliko je u razredu T registra PSW vrednost 1. Instrukcijama **TRPE** i **TRPD** u ovaj razred registra PSW upisuju se

vrednosti 1 ili 0, respektivno. Time se programskim putem dozvoljava ili ne režim rada procesora prekid posle svake instrukcije.

U skupu instrukcija postoji instrukcija **INT** kojom se može programskim putem izazvati prekid i skok na željenu prekidnu rutinu. Prekidna rutina na koju treba skočiti određuje se adresnim delom ove instrukcije koji sadrži broj ulaza u tabelu adresa prekidnih rutina. Izvršavanje ove instrukcije realizuje sve ono što je nabrojano u okviru hardverskog dela opsluživanja zahteva za prekid, s tim što je broj ulaza u tabeli adresa prekidnih rutina dat samom instrukcijom.

Kada procesor izvršava prekidnu rutinu može stići novi zahtev za prekid. Na ovaj zahtev za prekid može se reagovati na sledeće načine:

- prekida se izvršavanje tekuće prekidne rutine i skače na novu prekidnu rutinu (ukoliko je njegov nivo prioriteta niži ili jednak nivou prioriteta tekuće prekidne rutine) ili
- ne prekida se izvršavanje prekidne rutine, već se zahtev za prekid prihvata tek po završetku prekidne rutine i povratku u prekinuti program (ukoliko je njegov nivo prioriteta niži ili jednak nivou prioriteta tekuće prekidne rutine).

Procesor reaguje na oba načina u zavisnosti od situacije u kojoj se nalazi. Ta situacija zavisi od čitavog niza elemenata kao što su:

- ima više tipova zahteva za prekid,
- kod ulaska u bilo koju prekidnu rutinu brišu se razredi I i T u registru PSW,
- programskim putem se može, upisivanjem vrednosti 0 ili 1 u razrede I i T, odrediti kada će se reagovati na maskirajuće prekide i prekidati program posle svake instrukcije, a kada ne i
- maskirajući prekidi su uređeni po prioritetima.

Pri normalnom funkcionisanju procesora generišu se samo maskirajući zahtevi za prekid. Da bi zahtev za maskirajući prekid bio prihvaćen potrebno je da bude ispunjen svaki od sledećih uslova:

- da je odgovarajući bit u registru maske IMR postavljen,
- da je razred I u registru PSW postavljen,
- da je nivo prioriteta kontrolera periferije koji je uputio zahtev za prekid veći od nivoa prioriteta tekućeg programa.

Prekidanje izvršavanja tekuće prekidne rutine i skok na novu prekidnu rutinu naziva se gneždenje prekida.

Povratak iz prekidne rutine se realizuje instrukcijom **RTI**. Ovom instrukcijom se sa steka restauriraju registri PSW i PC, tim redom.

3.2 MEMORIJA

Arhitekturu memorije čine 8-mo bitne programski dostupne memorijske lokacije kapaciteta 64K reči, koje zauzimaju opseg od 0 do EFFFh memorijskog adresnog prostora. Iz memorijskih lokacija se čita i u memorijske lokacije se upisuje instrukcijama LD (poglavlje 3.1.5.1.4), ADD i SUB (poglavlje 3.1.5.1.5), AND, OR i XOR (poglavlje 3.1.5.1.6), ST (poglavlje 3.1.5.1.4) uz korišćenje registarskog direktnog adresiranja, memorijskog direktnog adresiranja, memorijskog indirektnog adresiranja, neposrednog adresiranja (poglavlje 3.1.5.1.4). Iz memorijskih lokacija se čita i u memorijske lokacije se upisuje instrukcijama JSR i RTS (poglavlje 3.1.5.1.3.3) i INT (poglavlje 3.1.5.1.3.4) uz korišćenje sadržaja ukazivača na vrh steka SP kao adrese memorijske lokacije (poglavlje 3.1.1).

3.3 ULAZNO/IZLAZNI UREĐAJI

Arhitekturu ulazno/izlaznih uređaja čine 8-mo bitni programski dostupni registri kontrolera ulazno/izlaznih uređaja kapaciteta 4K reči, koji zauzimaju opseg od F000h do FFFFh memorijskog adresnog prostora.

4 MAGISTRALA

U ovoj glavi se razmatra organizacija sistemske magistrale ***BUS***. Najpre se daje struktura magistrale, zatim arbitracija na magistrali i na kraju ciklusi na magistrali.

U slučaju magistrale sa atomskim ciklusima postoje ciklus čitanja, ciklus upisa i ciklus prihvatanja broja ulaza, a magistrala je zauzeta sve vreme dok se realizuje prenos podatka između gazde i sluge. U zavisnosti od toga kako se utvrđuje šta i kada gazda i sluga treba da urade prilikom realizacije ciklusa na magistrali, razlikuju se asinhrone i sinhrone magistrale.

Sistemska magistrala je asinhrona magistrala, koja služi za povezivanje modula računarskog sistema i to procesora ***CPU***, memorije ***MEM*** i ulazno/izlaznih uređaja ***U/I***. Preko magistrale se prenose sadržaji između registara procesora, memorijskih lokacija i registara uređaja. Ceo tok prenosa nekog sadržaja između dva modula naziva se ciklus na magistrali. Modul koji započinje ciklus na magistrali naziva se gazda (master), a modul sa kojim gazda realizuje ciklus naziva se sluga (slave). Gazda može da bude procesor i uređaj sa direktnim pristupom memoriji. Sluga može da bude memorija i uređaji bez i sa direktnim pristupom memoriji. Procesor čita sadržaje memorijskih lokacija i upisuje sadržaje u memorijske lokacije prilikom čitanja instrukcija i operanada i upisa rezultata kao sastavnog dela izvršavanja instrukcija. Pored toga procesor čita sadržaje registara uređaja i upisuje sadržaje u registre uređaja prilikom izvršavanja instrukcija kojima se dobija status uređaja, vrši inicijalizacija uređaja, zadaje režim rada i vrši startovanje i zaustavljanje uređaja i vrši prenos podataka između procesora i uređaja i obratno. Uređaj sa direktnim pristupom memoriji čita sadržaje memorijskih lokacija i upisuje sadržaje u memorijske lokacije u okviru prenosa podataka iz memorije u izlazni uređaj i iz ulaznog uređaja u memoriju.

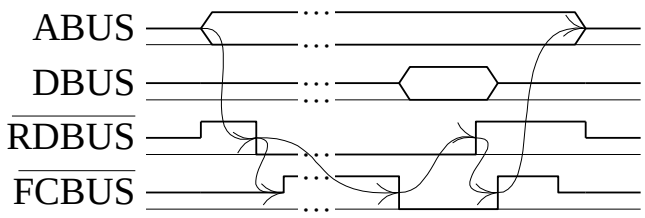
Na magistrali mogu da se realizuju dva tipa ciklusa između gazde i sluge i to ciklus čitanja i ciklus upisa. Za njihovu realizaciju koriste se tri grupe linija i to adresne linije $\mathbf{ABUS_{15...0}}$, linije podataka $\mathbf{DBUS_{7...0}}$ i upravljačke linije $\overline{\mathbf{RDBUS}}$, $\overline{\mathbf{WRBUS}}$ i $\overline{\mathbf{FCBUS}}$. Po adresnim linijama $\mathbf{ABUS_{15...0}}$ gazda šalje slugi adresu memorijske lokacije ili registra uređaja sa koje treba očitati sadržaj kod ciklusa čitanja ili na kojoj treba upisati sadržaj kod ciklusa upisa. Po linijama podataka $\mathbf{DBUS_{7...0}}$ sluga šalje gazdi očitani sadržaj u slučaju ciklusa čitanja i gazda šalje slugi sadržaj koji treba upisati u slučaju ciklusa upisa. Po upravljačkoj liniji $\overline{\mathbf{RDBUS}}$ gazda šalje signal kojim u slugi startuje ciklus čitanja. Po upravljačkoj liniji $\overline{\mathbf{WRBUS}}$ gazda šalje signal kojim u slugi startuje ciklus upisa. Po upravljačkoj liniji $\overline{\mathbf{FCBUS}}$ sluga u slučaju oba ciklusa šalje signal gazdi kao indikaciju da je on svoj deo ciklusa završio. U slučaju ciklusa čitanje to je i indikacija da se na linijama podataka $\mathbf{DBUS_{7...0}}$ nalazi sadržaj koji gazda treba da upiše u svoj prihvatni registar.

Kako je ovo magistrala sa atomskim ciklusima na njoj mogu da se realizuju ciklus čitanja i ciklus upisa, a magistrala je zauzeta sve vreme dok se realizuje prenos podatka između gazde i sluge. Moduli ove magistrale rade asinhrono svaki na svoj signal takta i trajanje svakog ciklusa je određeno vremenom pristupa modula sluge.

Svi moduli računarskog sistema su povezani na adresne linije, linije podataka i upravljačke linije magistrale preko bafera sa tri stanja. Pri tome na linije magistrale odgovarajuće sadržaje mogu preko bafera sa tri stanja da propuštaju samo modul koji je trenutno gazda i modul koji kao sluga sa njim realizuje ciklus na magistrali. Svi ostali

moduli svoje bafere sa tri stanja drže u stanju visoke impedanse i ne opterećuju linije magistrale.

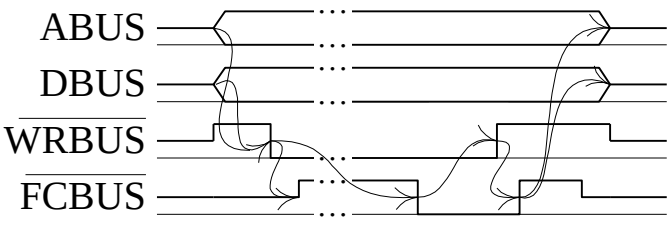
Vremenski oblici signala **ABUS$_{15...0}$**, **DBUS$_{7...0}$**, $\overline{\textbf{RDBUS}}$ i $\overline{\textbf{FCBUS}}$ koje na magistrali razmenjuju gazda i sluga prilikom realizacije ciklusa čitanja dati su na slici Slika 11. Signali su predstavljeni logičkim vrednostima jedan i nula, kao i stanjem visoke impedanse koje je predstavljeno linijom po sredini između logičke vrednosti jedan i nula.



Slika 11 Vremenski oblici signala za ciklus čitanja

Po dobijanju magistrale gazda započinje ciklus čitanja tako što otvara bafere sa tri stanja preko kojih šalje adresu na adresne linije magistrale **ABUS$_{15...0}$** i postavlja upravljački signal $\overline{\textbf{RDBUS}}$ na vrednost 0. Sve sluge dekoduju sadržaj sa adresnih linija magistrale **ABUS$_{15...0}$** i samo u slugi koji je sadržaj sa adresnih linija **ABUS$_{15...0}$** prepoznao kao svoju adresu upravljački signal $\overline{\textbf{RDBUS}}$ startuje ciklus čitanja. Za čitanje sadržaja na strani sluge potrebno je neodređeno vreme koje se u opštem slučaju razlikuje od sluge do sluge. Zato gazda ostaje u stanju čekanja sve dok sluga ne završi sa čitanjem. Po završenom čitanju sluga otvara bafere sa tri stanja preko kojih šalje očitani sadržaj na linije podataka magistrale **DBUS$_{7...0}$** i postavlja upravljački signal $\overline{\textbf{FCBUS}}$ na vrednost 0, zatim zatvara bafere sa tri stanja, pa linije podataka magistrale **DBUS$_{7...0}$** i upravljačka linija magistrale $\overline{\textbf{FCBUS}}$ prelaze u stanje visoke impedanse. Po detektovanju vrednosti 0 signala $\overline{\textbf{FCBUS}}$ gazda upisuje sadržaj sa linija podataka magistrale **DBUS$_{7...0}$** u svoj prihvatni registar i zatvara svoje bafere sa tri stanja, pa adresne linije magistrale **ABUS$_{15...0}$** i upravljačka linija magistrale $\overline{\textbf{RDBUS}}$ prelaze u stanje visoke impedanse. Time je ciklus čitanja na magistrali završen.

Vremenski oblici signala **ABUS$_{15...0}$**, **DBUS$_{7...0}$**, $\overline{\textbf{WRBUS}}$ i $\overline{\textbf{FCBUS}}$ koje na magistrali razmenjuju gazda i sluga prilikom realizacije ciklusa upisa dati su na slici Slika 12. Način predstavljanja signala je sličan kao i kod ciklusa čitanja.



Slika 12 Vremenski oblici signala za ciklus upisa

Po dobijanju magistrale gazda započinje ciklus upisa tako što otvara bafere sa tri stanja preko kojih šalje adresu na adresne linije magistrale **ABUS$_{15...0}$** , podatak na linije podataka magistrale **DBUS$_{7...0}$** i postavlja upravljački signal $\overline{\textbf{WRBUS}}$ na vrednost 0. Sve sluge dekoduju sadržaj sa adresnih linija magistrale **ABUS$_{15...0}$** i samo u slugi koji je sadržaj sa adresnih linija **ABUS$_{15...0}$** prepoznao kao svoju adresu upravljački signal

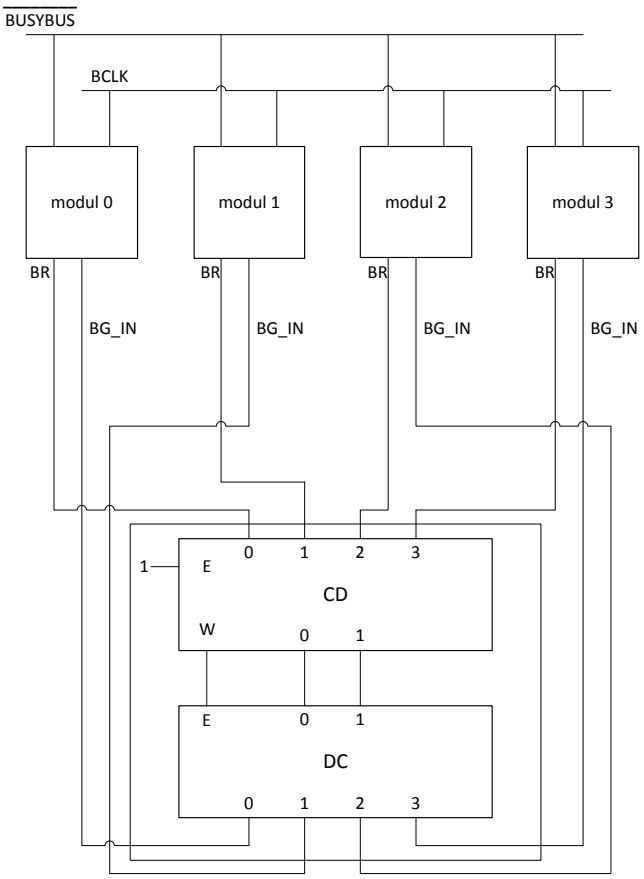
$\overline{\textbf{WRBUS}}$ startuje ciklus upisa sadržaja sa linija podataka magistrale $\textbf{DBUS}_{7...0}$. Za upis sadržaja na strani sluge potrebno je neodređeno vreme koje se u opštem slučaju razlikuje od sluge do sluge. Zato gazda ostaje u stanju čekanja sve dok sluga ne završi sa upisom. Po završenom upisu sluga otvara bafer sa tri stanja preko koga postavlja upravljački signal $\overline{\textbf{FCBUS}}$ na vrednost 0, dok na signal takta t_j zatvara bafer sa tri stanja, pa upravljačka linija magistrale $\overline{\textbf{FCBUS}}$ prelazi u stanje visoke impedanse. Po detektovanju vrednosti 0 signala $\overline{\textbf{FCBUS}}$ gazda zatvara svoje bafere sa tri stanja, pa adresne linije magistrale $\textbf{ABUS}_{15...0}$, linije podataka magistrale $\textbf{DBUS}_{7...0}$ i upravljačka linija magistrale $\overline{\textbf{WRBUS}}$ prelaze u stanje visoke impedanse. Time je ciklus upisa na magistrali završen.

4.1.Paralelni Arbitrator

Kod ovih sistema svi moduli koji bi želeli da realizuju neki ciklus na magistrali, a to su procesor i ulazno/izlazni uređaj sa direktnim pristupom memoriji, moraju najpre da učestvuju u arbitraciji, pa tek potom modul koji dobije dozvolu korišćenja magistrale može da realizuje ciklus na magistrali. U zavisnosti od toga kako se utvrđuje koji od modula dobija dozvolu korišćenja magistrale razlikuju se paralelna i serijska arbitracija.

Pošto se radi o paralelnoj arbitraciji imamo poseban uređaj koji se naziva arbitrator i koji se sastoji od kodera prioriteta CD i dekodera DC (slika Slika 13). Između modula koji žele da realizuju neki ciklus na magistrali i arbitratora postoji par linija BR i BG_IN. Po liniji BR modul šalje zahtev za korišćenje magistrale, a po liniji BG_IN arbitrator šalju modulu dozvolu korišćenja magistrale. Uzeto je da arbitrator ima n linija ulaza i izlaza, pri čemu je $n=2^m$, i da je linija 0 najvišeg a linija (n-1) najnižeg prioriteta. U zavisnosti od toga na koju od n linija arbitratora su povezane linije BR i BG_IN modula određuje se prioritet modula. S toga je modul koji je povezan na liniju 0 najvišeg a modul koji je povezan na liniju (n-1) najnižeg prioriteta.

Ukoliko zahtevi za korišćenje magistrale stignu po više linija BR istovremeno, dozvola se daju samo po jednog liniji BG_IN i to po onoj koja odgovara liniji BR najvišeg prioriteta po kojoj je upućen zahtev za korišćenje magistrale.



Slika 13 Paralelni arbitrator

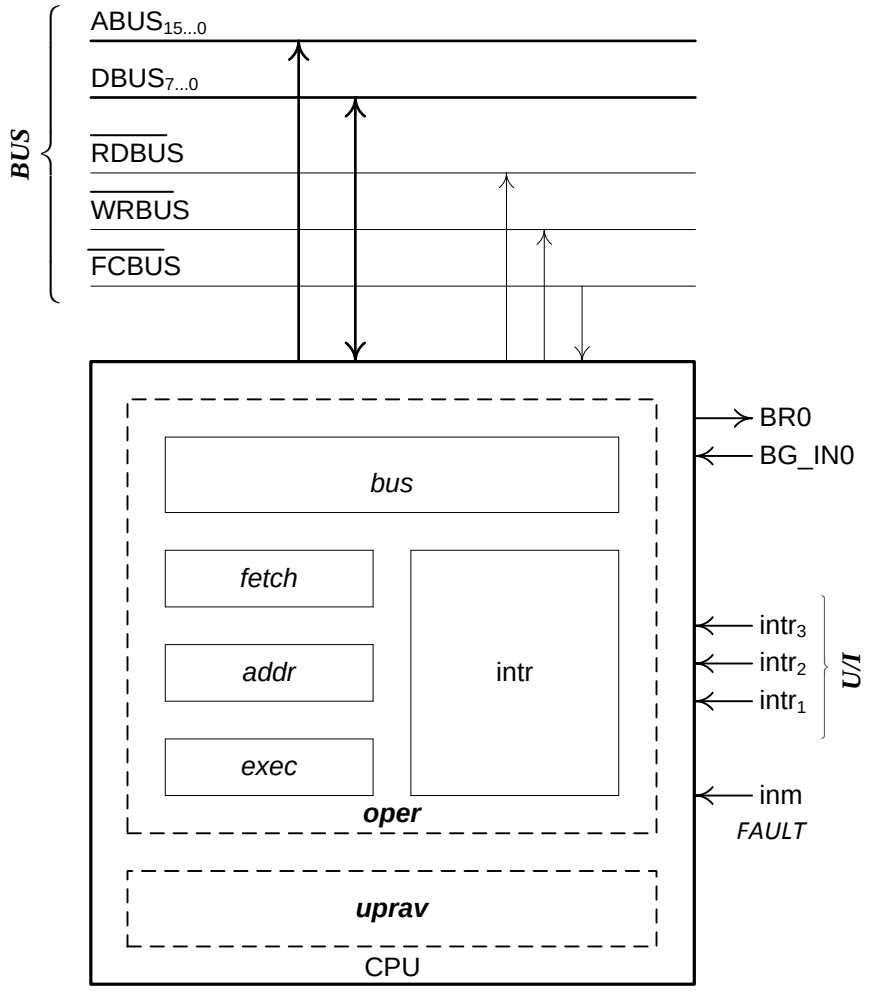
Zahtevi korišćenja magistrale BR predstavljaju interno generisane zahteve korišćenja magistrale sinhronizovane na signal takta magistrale BCLK. Ovo se nameće kao posledica pretpostavke da svaki modul radi sa svojom internom periodom signala takta, pa interno generisani zahtevi korišćenja magistrale dobijaju vrednosti 1 i 0 u trenucima signala takta modula. Tako generisani zahtevi su vremenski nesinhronizovani pa bi, ukoliko bi se oni koristili u arbitraciji, mogli da stvore probleme i u slučaju paralelne i u slučaju serijske arbitracije. Sinhronizovanjem tih interno generisanih zahteva na signal takta magistrale BCLK obezbeđuje se da signali zahteva BR svih modula dobijaju vrednosti 1 i 0 u trenucima signala takta magistrale BCLK. Pri tome perioda signala takta magistrale BCLK mora da bude veća od propagacije kroz.

Dok modul koji je gazda realizuje ciklus na magistrali u trajanju više perioda signala takta modula, u drugim modulima mogu da se jave zahtevi za korišćenje magistrale. S toga se paralelno sa ciklusom na magistrali odvija arbitracija i u toku trajanja ciklusa neki modul može da dobije dozvolu korišćenja magistrale. Taj modul ne sme da krene sa realizacijom svog ciklusa na magistrali dok modul koji je započeo ciklus na magistrali ne kompletira ciklus. Da bi se to osiguralo uvodi se i signal zauzeća magistrale $\overline{\text{BUSBUSY}}$. Vrednost 0 ovog signala označava da je magistrala zauzeta, a stanje visoke impedanse da je slobodna. Modul koji kao gazda realizuje ciklus na magistrali na početku ciklusa postavlja ovaj signal na 0, a po završetka ciklusa u stanje visoke impedanse. Modul koji dobije dozvolu korišćenja magistrale pre nego što krene sa realizacijom ciklusa na magistrali mora da proveri vrednost signala $\overline{\text{BUSBUSY}}$. Ukoliko signal $\overline{\text{BUSBUSY}}$ ima vrednost 0 magistrala je zauzeta i modul koji je dobio dozvolu ne sme

da krene sa realizacijom ciklusa na magistrali, već mora da sačeka da signal $\overline{\text{BUSBUSY}}$ pređe u stanje visoke impedanse. Ukoliko je signal $\overline{\text{BUSBUSY}}$ u stanju visoke impedanse magistrala nije zauzeta i modul koji je dobio dozvolu najpre postavlja signal $\overline{\text{BUSBUSY}}$ na vrednost nula pa tek onda kreće sa realizacijom ciklusa na magistrali. Po završetku ciklusa na magistrali dati modul postavlja signal $\overline{\text{BUSBUSY}}$ u stanje visoke impedanse. Time se omogućuje da modul koji je u međuvremenu dobio dozvolu korišćenja magistrale ali nije mogao da krene da realizuje cikilus na magistrali jer je magistrala bila zauzeta, sada može, postavljanjem signala $\overline{\text{BUSBUSY}}$ na 0, da zauzme magistralu i krene sa realizacijom svog ciklusa na magistrali.

5 PROCESOR

U ovoj glavi se daje organizacija procesora ***CPU*** koji se sastoji iz operacione jedinice ***oper*** i upravljačke jedinice ***uprav*** (slika Slika 14).



Slika 14 Organizacija procesora ***CPU***

Operaciona jedinica ***oper*** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za pamćenje binarnih reči, izvršavanje mikrooperacija i generisanje signala logičkih uslova upravljačke jedinice ***uprav***. Upravljačka jedinica ***uprav*** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za generisanje upravljačkih signala operacione jedinice ***oper*** na osnovu algoritama operacija i signala logičkih uslova.

Struktura i opis operacione i upravljačke jedinice će biti detaljnije objašnjene u poglavljima koja slede.

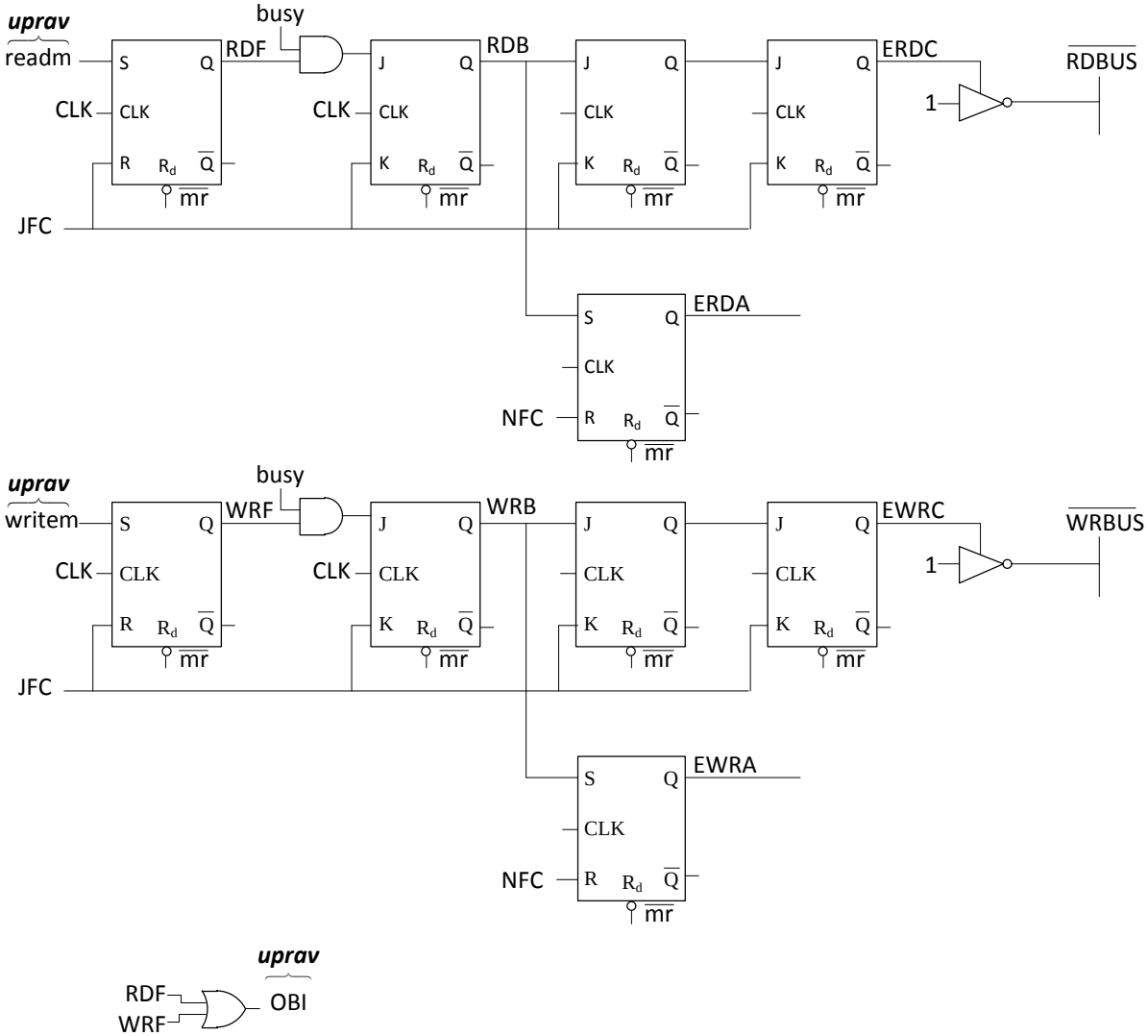
5.1 OPERACIONA JEDINICA

Operaciona jedinica ***oper*** (Slika 14) se sastoji od sledećih blokova:

- blok *bus*,
- blok *fetch*,
- blok *addr*,
- blok *exec* i
- blok *intr*.

Ovi blokovi su međusobno povezani sa dve magistale. Jedna izvorišna SBUS i jedna odredišna DBUS.

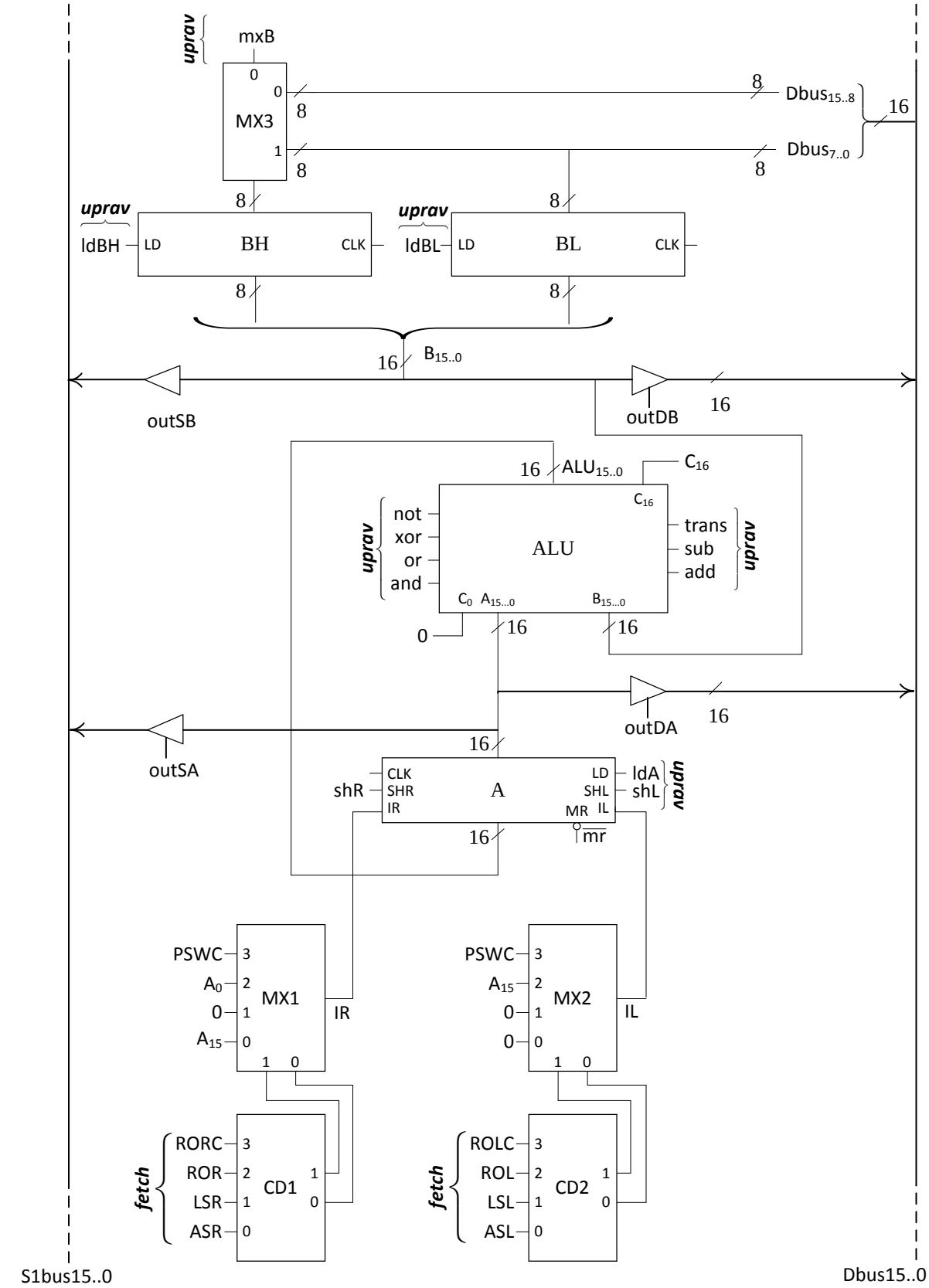
Blok *bus* (povezivanje na magistralu) (slike Slika 15,



Slika 16 i

Slika 17) služi za arbitraciju procesora i ulazno/izlaznog uređaja ***U/I2*** sa kontrolerom za direktan pristup memoriji pri korišćenju magistrale ***BUS*** i realizaciju ciklusa na

magistrali ***BUS***. Blok *fetch*(slike Slika 18, Slika 19 i Slika 20) služi za čitanje instrukcije i smeštanje u prihvatni registar instrukcije. Blok *addr* (Slika 21) služi za formiranje adrese operanda i čitanje operanda. Blok *exec* (slike



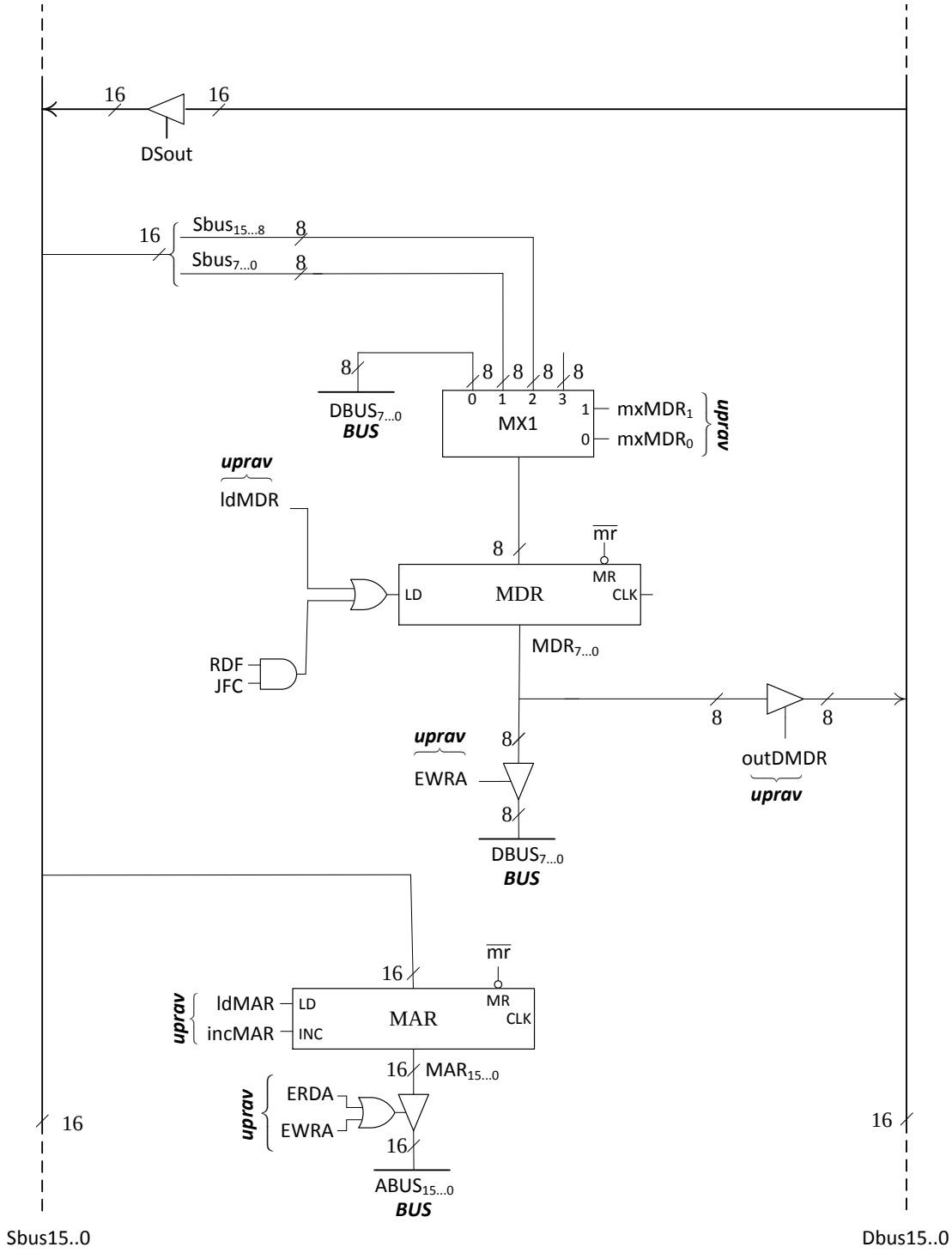
Slika 22,Slika 23Slika 24 i Slika 25) služi za izvršavanje operacija. Blok *intr*

(opsluživanje prekida) (slike Slika 26, Slika 27 i Slika 28) služi za prihvatanje prekida i generisanje broja ulaza u tabelu sa adresama prekidnih rutina

Struktura i opis svih blokova operacione jedinice ***oper*** dati su u daljem tekstu.

5.1.1 Blok bus

Blok *bus* (Slika 15) koji služi za povezivanje na magistralu sadrži registre $\mathbf{MAR_{15...0}}$ i $\mathbf{MDR_{7...0}}$ sa multiplekserom **MX1** i kombinacione mreže za realizaciju ciklusa na magistrali ***BUS*** kada je procesor gazda i za komunikaciju sa paralelnim arbitratorom kada se vrši arbitracija sa kontrolerima sa direktnim pristupom memoriji pri korišćenju magistrale ***BUS***. Registri su povezani na 2 interne magistrale.



Slika 15 Blok *bus* (prvi deo)

Registar **MAR$_{15...0}$** je 16-to razredni adresni registar, čiji se sadržaj koristi pri realizaciji ciklusa čitanja ili upisa na magistrali ***BUS***. Adresa se iz nekog od blokova operacione jedinice ***oper*** preko interne magistrale **Sbus$_{15..0}$** vodi na ulaze registra **MAR$_{15...0}$** i u njega upisuje vrednošću 1 signala **ldMAR**. Sadržaj registra **MAR$_{15...0}$** se inkrementira vrednošću 1 signala **incMAR**, što se koristi u situacijama kada treba pročitati ili upisati 16-to bitnu veličinu koja se nalazi u dvema susednim 8-bitnim

lokacijama. Tada se najpre u registar **$MAR_{15...0}$** upisuje adresa više lokacije, a posle se inkrementiranjem dobija adresa niže lokacije.

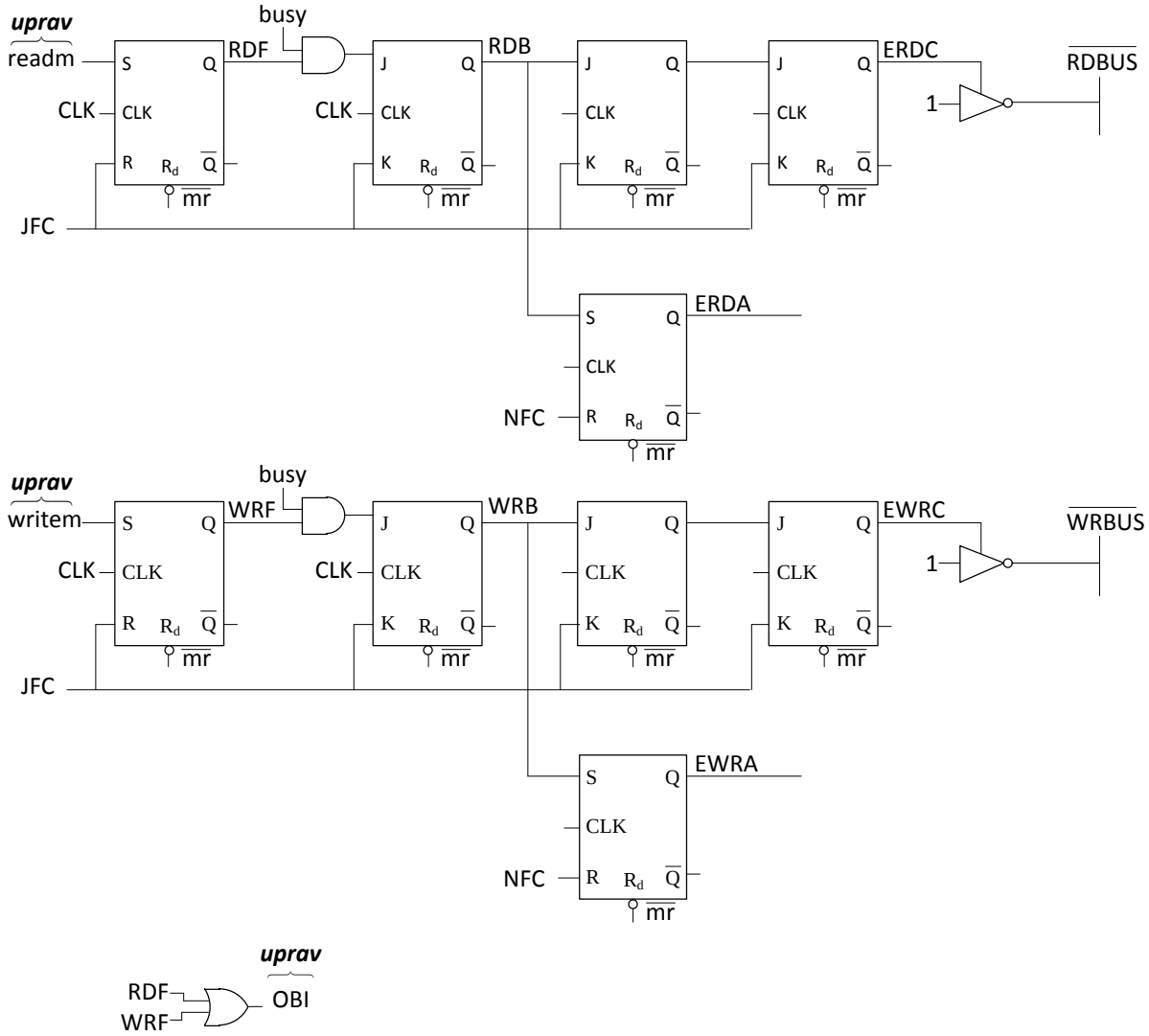
Preko interne magistrale **$Sbus_{15..0}$** se na ulaz registra **$MAR_{15...0}$** mogu dovesti sadržaji $PC_{15...0}$, $IR_{15...0}$, i $SP_{15...0.}$ Sadržaj $PC_{15...0}$ predstavlja adresu instrukcije. Sadržaj $IR_{15...0}$ (registri IR3 i IR4) predstavlja adresu operanda u slučaju memorijskog direktnog adresiranja. Sadržaj $SP_{15...0}$ se koristi prilikom stavljanja sadržaja na vrh steka i skidanja sadržaja sa vrha steka. Pri realizaciji ciklusa čitanja ili upisa se vrednošću 1 signala **ERDA** i **EWRA** sadržaj registra $MAR_{15...0}$ propušta kroz bafere sa tri stanja na adresne linije **$ABUS_{15...0}$** magistrale ***BUS***.

Signal **JFC** ima vrednost 1 kada se na upravljačkoj liniji $\overline{\textbf{FCBUS}}$ magistrale ***BUS*** pojavi vrednost 0 u trajanju jedna perioda signala takta kao indikacija od memorije ili nekog od kontrolera kao sluge da je on svoj deo ciklusa završio. U slučaju ciklusa čitanje to je i indikacija da se na linijama podataka **$DBUS_{7...0}$** magistrale ***BUS*** nalazi sadržaj koji gazda treba da upiše u svoj prihvatni registar. Signal **RDF** ima aktivnu vrednost kada je u toku operacija čitanja iz memorije. Sadržaj registra **$MDR_{7...0}$** se vrednošću 1 signala **EWRA** propušta kroz bafere sa tri stanja na linije podataka **$DBUS_{7...0}$** magistrale ***BUS***, a vrednošću 1 signala **outDMDR**, proširen do 16 bita, kroz bafere sa tri stanja na linije interne magistrale $Dbus_{15...0}$.

Registar **$MDR_{7...0}$** je 8-bitni registar podatka u koji se vrednošću 1 signala **ldMDR** upisuje sadržaj sa izlaza multipleksera **MX1**. Vrednošću 1 signala **ldMDR** i binarnoj vrednosti 00 signala **$mxMDR_1$ i $mxMDR_0$** sadržaj sa linija podataka **$DBUS_{7...0}$** magistrale ***BUS*** se propušta kroz multiplekser **MX1** i upisuje u registar **$MDR_{7...0}$**. Vrednošću 1 signala **ldMDR** i binarnim vrednostima 01 signala **$mxMDR_1$** , **$mxMDR_0$** kroz multiplekser se propušta nižih 8 bita dok se vrednošću 10 signala **$mxMDR_1$** , **$mxMDR_0$** propušta viših 8 bita sadržaja interne magistrale **$Sbus_{15..0}$**, i upisuje u registar $MDR_{7...0}$.

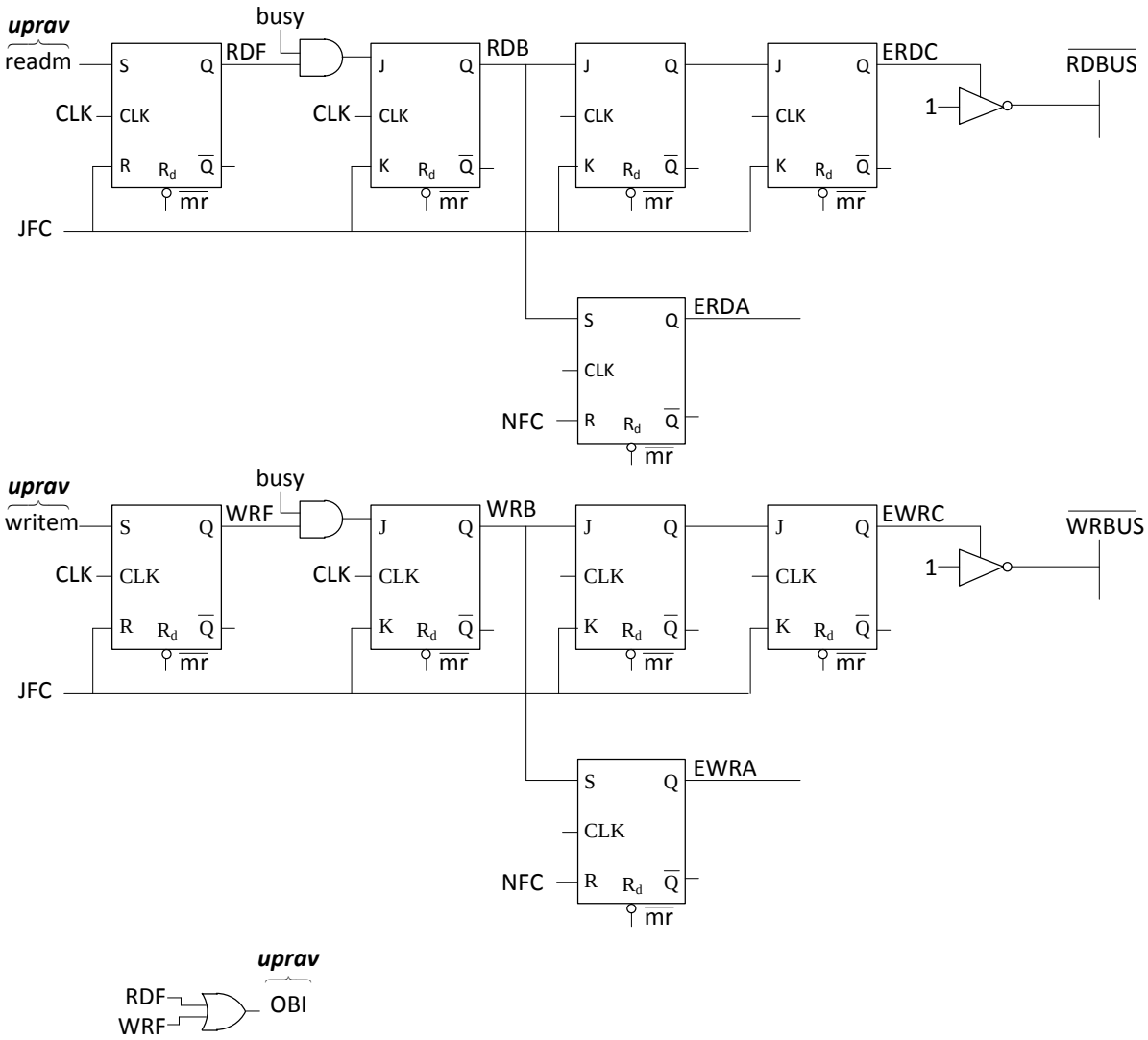
Multiplekser **MX1** je 8-bitni multiplekser sa 4 ulaza. Na ulaze 0 do 2 multipleksera se vode sadržaji linije podataka **$DBUS_{7...0}$** magistrale ***BUS***, **$Sbus_{7...0}$** i **$Sbus_{15...8}$**, a selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 10 upravljačkih signala **$mxMBR_1$** i **$mxMBR_0$**. Sadržaj linije podataka **$DBUS_{7...0}$** magistrale ***BUS*** se propušta kod ciklusa čitanja i predstavlja pročitanu vrednost memorijske lokacije ili registra kontrolera koja po tim linijama dolazi u procesor ***CPU***. Sadržaji **$Sbus_{7...0}$** i **$Sbus_{15...8}$** se propuštaju u slučaju ciklusa upisa u memorijsku lokaciju ili registar kontrolera. Takođe, sadržaji **$Sbus_{7...0}$** i **$Sbus_{15...8}$** se propuštaju u slučajevima u kojima se u dva ciklusa na magistrali sadržaji višeg i nižeg bajta akumulatora $A_{15...0}$ upisuju u memoriju, sadržaji višeg i nižeg bajta programskog brojača $PC_{15...0}$ stavljaju na stek i sadržaji višeg i nižeg bajta programske reči $PSW_{15...0}$ stavljaju na stek. Selektovani sadržaj sa izlaza multipleksera **MX1** se vodi na paralelne ulaze registra **$MDR_{7...0}$**.

Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je procesor gazda formiraju signale $\overline{\mathbf{RDBUS}}$ i $\overline{\mathbf{WRBUS}}$ magistrale ***BUS*** i **BR0** kojim se od arbitratora zahteva korišćenje magistrale date su na slici

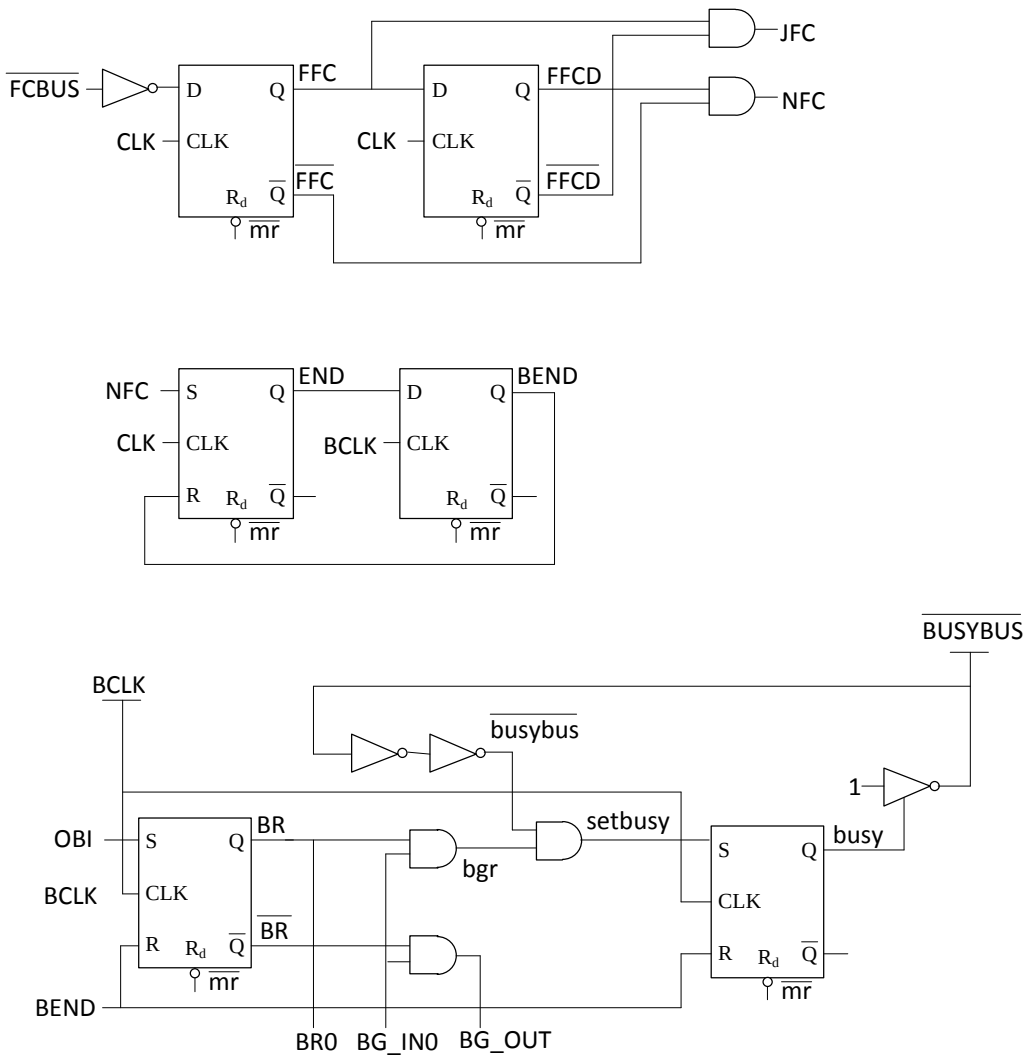


Slika 16. Signali $\overline{\mathbf{RDBUS}}$ i $\overline{\mathbf{WRBUS}}$ se formiraju na osnovu signala **ERDC** i **ERWC** upravljačke jedinice ***uprav***. Signal **JFC** se formira na osnovu signala $\overline{\mathbf{FCBUS}}$ magistrale ***BUS*** koji šalje memorija ili kontroler kao sluga (Slika

Slika 17).



Slika 16 Blok *bus* (drugi deo)



Slika 17 Blok *bus* (treći deo)

Realizacija ciklusa čitanja započinje postavljanjem na vrednost 1 upravljačkog signala **readm** koji na izlazu iz flip-flopa, na signal takta procesora **CLK**, postavlja signal **RDF** na vrednost 1. Signal **RDF** će dalje uzrokovati aktivnom vrednošću signala **OBI**, koji će tu vrednost imati sve dok procesor ne dobije magistralu od arbitratora i dok ne završi proces čitanja iz memorije. Signal **OBI** će na signal takta magistrale **BCLK**, postaviti signal **BR0** na aktivnu vrednost 1. Ovaj signal direktno je povezan na ulaz 0 arbitratora, koji je i najvećeg prioriteta. Kada arbitrator na ulazu 0 primi od procesora zahtev za magistralom, on, na signal takta magistrale **BCLK**, vraća procesoru aktivan signal **BG_IN0** (pošto se zahtev procesora uvek šalje na ulaz 0 koji je najvećeg prioriteta, on će uvek biti prihvaćen). Aktivne vrednosti signala **BG_IN0** i **BR0**, aktiviraju signal **bgr**. Ukoliko neki uređaj već realizuje ciklus na magistrali, procesor će čekati da se magistrala oslobodi. Kada magistrala postane slobodna, odnosno signal zauzeća magistrale $\overline{\text{BUSYBUS}}$ bude u visokoj impedansi, tada signal **setbusy** postaje aktivan. Signal **setbusy** će, na signal takta magistrale **BCLK**, postaviti signal **busy** na aktivnu vrednost 1. Ovim signalom će se na liniju zauzeća magistrale $\overline{\text{BUSYBUS}}$, kroz trostatički

bafer, propustiti vrednost 0 čime će se nekom sledećem uređaju koji traži magistralu signalizirаće se da je ona zauzeta.

Signal **busy**, sinhronizovan na signal takta magistrale BCLK i signal **RDF**, sinhronizovan na signal takta procesora CLK su sada aktivni. Oni svojim aktivnim vrednostima postavljaju signal **RDB** na aktivnu vrednost 1, ali tek na izlazu iz flip-flopa, koji radi na signal takta procesora **CLK**, sve zbog sinhronizacije signala **busy** na signal takta procesora **CLK**. Aktivnom vrednošću signala **RDB** se na sledeći signal takta procesora na aktivnu vrednost postavlja signal **ERDA** kojim se sadržaj registra $\mathbf{MAR_{15...0}}$ propušta na adresne linije $\mathbf{ABUS_{15..0}}$ magistrale ***BUS***. U narednom taktu se istom aktivnom vrednošću **RDB** na aktivnu vrednost postavlja signal **ERDC**, kojim se na upravljačku liniju $\overline{\mathbf{RDBUS}}$ propušta aktivna vrednost 0 kako bi moglo da započne čitanje podatka iz adresirane memorijske lokacije.

Pošto je za čitanje sadržaja potrebno neodređeno vreme procesor ostaje u stanju čekanja sve dok se ne završi sa čitanjem. Po završenom čitanju memorija, na signal takta magistrale **BCLK**, otvara bafere sa tri stanja preko kojih šalje očitani sadržaj na linije podataka $\mathbf{DBUS_{7...0}}$ magistrale ***BUS*** i postavlja upravljački signal $\overline{\mathbf{FCBUS}}$ na vrednost 0. Aktivna vrednost ovog signala će, na sledeći signal takta procesora **CLK**, aktivirati signal **FFC**, koji će dalje prouzrokovati aktivnu vrednost signala **JFC** u trajanju od jedne periode signala takta procesora **CLK**. Aktivna vrednost signala **JFC** označava da se adresirani podatak nalazi na linijama podataka $\mathbf{DBUS_{7..0}}$ magistrale ***BUS***. Sada će signal **ldMDR** postati aktivan što će prouzrokovati upis pročitane vrednosti u registar $\mathbf{MDR_{7...0}}$. Aktivan signal **JFC** prouzrokuje resetovanje signala **RDF** i **ERDC** na vrednost 0. Signal **OBI** će zbog neaktivne vrednosti signala **RDF** postati i sam neaktivan, a sa upravljačke linije $\overline{\mathbf{RDBUS}}$ će biti ukinuta aktivna vrednost 0 (prelazi u stanje visoke impedanse).

U sledećem taktu procesora će kao posledica rada istog D flip-flopa, na signal takta procesora **CLK**, signal **JFC** postati neaktivan, a signal **NFC** aktivan u trajanju jedne periode takta (jer na sledeći signal takta upravljački signal $\overline{\mathbf{FCBUS}}$ dobija vrednost 1). Signal **NFC** će svojom aktivnom vrednošću resetovati signal **ERDA** čime će se sa adresnih linija ukinuti vrednost registra $\mathbf{MAR_{15...0}}$. On će takođe prouzrokovati postavljanje signala **END** na aktivnu vrednost. Za sinhronizaciju ovog signala na signal takta magistrale **BCLK**, on se propušta kroz D flip-flopa koji radi na signal takta magistrale **BCLK** i generiše aktivnu vrednost signala **BEND**. Aktivan signal **BEND** će na jedan takt magistrale resetovati signal **BR0**, a na sledeći takt magistrale i signal **busy.** Time će signal zauzeća magistrale $\overline{\text{BUSYBUS}}$ preći u stanje visoke impedanse što za rezultat će imati da magistrala postaje slobodna. Ovim je završen ciklus čitanja podatka iz adresirane memorijske lokacije.

Ciklus upisa započinje postavljanjem na vrednost 1 upravljačkog signala **writem** koji na izlazu iz flip-flopa, na signal takta procesora **CLK**, postavlja signal **WRF** na vrednost 1. Signal **WRF** će dalje uzrokovati aktivnom vrednošću signala **OBI**, koji će tu vrednost imati sve dok procesor ne dobije magistralu od arbitratora i dok ne završi proces upisa u memoriju. Signal **OBI** će na signal takta magistrale **BCLK**, postaviti signal **BR0** na aktivnu vrednost 1. Ovaj signal direktno je povezan na ulaz 0 arbitratora, koji je i najvećeg prioriteta. Kada arbitrator na ulazu 0 primi od procesora zahtev za magistralom, on, na signal takta magistrale **BCLK**, vraća procesoru aktivan signal **BG_IN0** (pošto se zahtev procesora uvek šalje na ulaz 0 koji je najvećeg prioriteta, on će uvek biti

prihvaćen). Aktivne vrednosti signala **BG_IN0** i **BR0**, aktiviraju signal **bgr**. Ukoliko neki uređaj već realizuje ciklus na magistrali, procesor će čekati da se magistrala oslobodi. Kada magistrala postane slobodna, odnosno signal zauzeća magistrale $\overline{\text{BUSYBUS}}$ bude u visokoj impedansi, tada signal **setbusy** postaje aktivan. Signal **setbusy** će, na signal takta magistrale **BCLK**, postaviti signal **busy** na aktivnu vrednost 1. Ovim signalom će se na liniju zauzeća magistrale $\overline{\text{BUSYBUS}}$, kroz trostatički bafer, propustiti vrednost 0 čime će se nekom sledećem uređaju koji traži magistralu signalizirače se da je ona zauzeta.

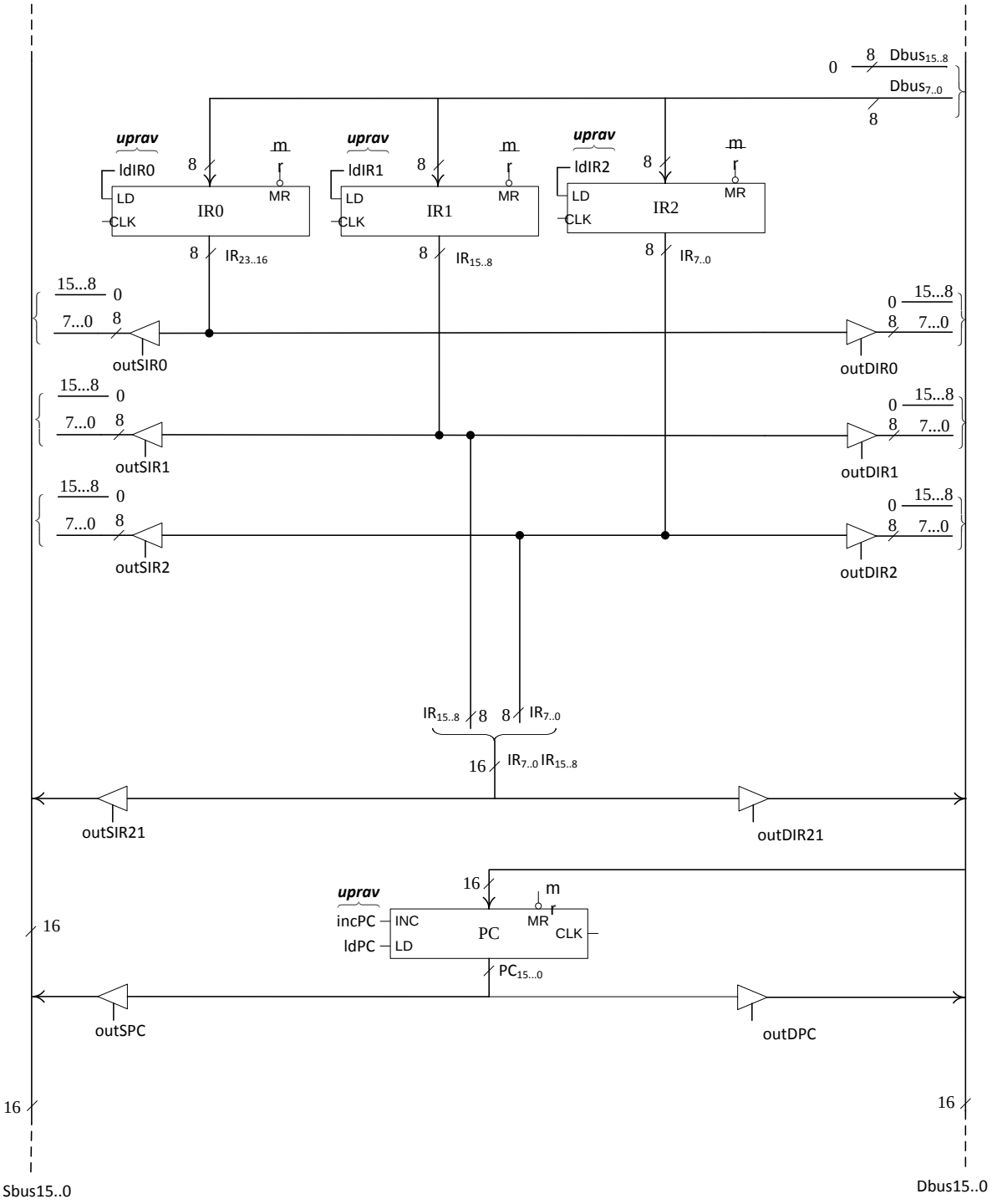
Signal **busy**, sinhronizovan na signal takta magistrale BCLK i signal **WRF**, sinhronizovan na signal takta procesora CLK su sada aktivni. Oni svojim aktivnim vrednostima postavljaju signal **WRB** na aktivnu vrednost 1, ali tek na izlazu iz flip-flopa, koji radi na signal takta procesora **CLK**, sve zbog sinhronizacije signala **busy** na signal takta procesora **CLK**. Aktivnom vrednošću signala **WRB** se na sledeći signal takta procesora na aktivnu vrednost postavlja signal **EWRA** kojim se sadržaj registra $\mathbf{MAR_{15...0}}$ propušta na adresne linije $\mathbf{ABUS_{15..0}}$ magistrale ***BUS*** i sadržaj registra $\mathbf{MDR_{7...0}}$ na linije podataka $\mathbf{DBUS_{7..0}}$. U narednom taktu se istom aktivnom vrednošću **WRB** na aktivnu vrednost postavlja signal **EWRC**, kojim se na upravljačku liniju $\overline{\mathbf{WRBUS}}$ propušta aktivna vrednost 0 kako bi mogalo da započne upis podatka u adresiranu memorijsku lokaciju.

Pošto je za upis sadržaja potrebno je neodređeno vreme procesor ostaje u stanju čekanja sve dok se ne završi sa upisom. Po završenom upisu memorija, na signal takta magistrale **BCLK**, otvara bafere sa tri stanja i postavlja upravljački signal $\overline{\mathbf{FCBUS}}$ na vrednost 0. Aktivna vrednost ovog signala će, na sledeći signal takta procesora **CLK**, aktivirati signal **FFC**, koji će dalje prouzrokovati aktivnu vrednost signala **JFC** u trajanju od jedne periode signala takta procesora **CLK**. Aktivna vrednost signala **JFC** označava da je upis podatka u adresiranu memorijsku lokaciju završen. Ovaj signal će prouzrokovati resetovanje signala **WRF** i **EWRC** na vrednost 0. Signal **OBI** će zbog neaktivne vrednosti signala **WRF** postati i sam neaktivan, a sa upravljačke linije $\overline{\mathbf{WRBUS}}$ će biti ukinuta aktivna vrednost 0 (prelazi u stanje visoke impedanse).

U sledećem taktu procesora će kao posledica rada istog D flip-flopa, na signal takta procesora **CLK**, signal **JFC** postati neaktivan, a signal **NFC** aktivan u trajanju jedne periode takta (jer na sledeći signal takta upravljački signal $\overline{\mathbf{FCBUS}}$ dobija vrednost 1). Signal **NFC** će svojom aktivnom vrednošću resetovati signal **EWRA** čime će se sa adresnih linija ukinuti vrednost registra $\mathbf{MAR_{15...0}}$ i sa linija podataka ukinuti vrednost registra $\mathbf{MDR_{7...0}}$. On će takođe prouzrokovati postavljanje signala **END** na aktivnu vrednost. Za sinhronizaciju ovog signala na signal takta magistrale **BCLK**, on se propušta kroz D flip-flopa koji radi na signal takta magistrale **BCLK** i generiše aktivnu vrednost signala **BEND**. Aktivan signal **BEND** će na jedan takt magistrale resetovati signal **BR0**, a na sledeći takt magistrale i signal **busy.** Time će signal zauzeća magistrale $\overline{\text{BUSYBUS}}$ preći u stanje visoke impedanse što za rezultat će imati da magistrala postaje slobodna. Ovim je završen ciklus upisa podatka u adresiranu memorijsku lokaciju.

5.1.2 Blok fetch

Blok *fetch* sadrži registar $PC_{15...0}$ sa multiplekserom MX, registre IR0, IR1, IR2 (Slika 18), dekodere DC1 do DC6 signala logičkih uslova operacija i načina adresiranja (Slika 19) i kombinacione mreže signala logičkih uslova dužina instrukcija (Slika 20).



Slika 18 Blok *fetch* (prvi deo)

Registar $PC_{15...0}$ je 16-to razredni programski brojač čiji sadržaj predstavlja adresu memorijske lokacije počev od koje treba pročitati jedan, dva ili tri bajta. Adresa skoka u

programu se iz nekog od blokova operacione jedinice ***oper*** preko interne magistrale **Dbus$_{15..0}$** vodi na ulaze registra $PC_{15...0}$ i u njega upisuje vrednošću 1 signala **ldPC**. Sadržaj registra $PC_{15...0}$ se inkrementira vrednošću 1 signala **incPC**, što se koristi prilikom čitanja svakog bajta instrukcije koji se nalaze u susednim 8-bitnim lokacijama.

Sadržaj registra $PC_{15...0}$ se vrednošću 1 signala **PCout** propušta kroz bafere sa tri stanja na internu magistralu **Sbus$_{15...0}$**. To omogućava da se vrednost iz registra $PC_{15...0}$ prenese u registar $MAR_{15...0}$ prilikom čitanja reči instrukcije, na ulaz sabirača ADD bloka *addr* prilikom instrukcija relativnog skoka gde je potrebno trenutnu vrednost iz registra $PC_{15...0}$ sabrati sa odgovarajućim pomerajem, ili u registar $MDR_{15...0}$ prilikom skoka na prekidnu rutinu gde je potrebno staru vrednost $PC_{15...0}$ staviti na stek.

Registri IR0, IR1 i IR2 su 8-mo razredni registri koji formiraju razrede 23…16, 15…8 i 7…0, respektivno, prihvatnog registra instrukcije $IR_{23...0}$. Instrukcije mogu, u zavisnosti od formata instrukcije, da budu dužine 1, 2 ili 3 bajta. Saglasno formatu instrukcije prvi, drugi i treći bajt instrukcije se smeštaju redom u registre IR0, IR1 i IR2. Paralelan upis sadržaja prihvatnog registra podatka $MDR_{7...0}$ u jedan od registara IR0, IR1 i IR2 se realizuje vrednošću 1 jednog od signala **ldIR0**, **ldIR1** i **ldIR2** respektivno. Razredi $IR_{23...16}$ se uvek čitaju i njihov sadržaj predstavlja kod operacije. Broj preostalih razreda koji se čita zavisi od koda operacije, a u slučaju aritmetičkih i logičkih operacija, i od načina adresiranja i njihov sadržaj ima različito značenje.

Ukoliko se, nakon čitanja prvog bajta instrukcije, utvrdi da se radi o bezadresnim instrukcijama, više se ne čitaju bajtovi. Cela instrukcija je pročitana i može se izvršiti.

Ukoliko se, nakon čitanja prvog bajta instrukcije, utvrdi da se radi o instrukcijama uslovnog skoka, čita se još jedan bajt. Tada razredi $IR_{15...8}$ predstavljaju pomeraj koji se aktivnom vrednošću signala **outSIR1** propušta kroz bafer sa tri stanja na internu magistralu **Sbus$_{15...0}$**. To omogućava da se vrednost pomeraja, zajedno sa vrednošću iz registra $PC_{15...0}$ dovede na ulaz sabirača ADD bloka *addr*, na čijem izlazu će se dobiti apsolutna adresa na koju je potrebno skočiti.

Ukoliko se, nakon čitanja prvog bajta instrukcije, utvrdi da se radi o instrukcijama bezuslovnog skoka, a da u pitanju nije instrukcija INT, čitaju se još dva bajta. Tada razredi $IR_{15...8}IR_{7...0}$ predstavljaju apsolutnu adresu memorijske lokacije na koju je potrebno skočiti. Ova vrednost se aktivnom vrednošću signala **outDIR21** propušta kroz bafer sa tri stanja na internu magistralu **Dbus$_{15...0}$**. To omogućava da se vrednost apsolutne adrese upise u registar $PC_{15...0}$.

Ukoliko se, nakon čitanja prvog bajta instrukcije, utvrdi da se radi o instrukcijama bezuslovnog skoka i da je u pitanju instrukcija INT, čita se još jedan bajta. Tada razredi $IR_{15...0}$ predstavljaju broj ulaza u tabelu adresa prekidnih rutina prekida na čiju se prekidnu rutinu skače. Ova vrednost se aktivnom vrednošću signala **outDIR1** propušta kroz bafer sa tri stanja na internu magistralu **Dbus$_{15...0}$**. To omogućava da se broj ulaza upise u registar $BR_{7...0}$.

Ukoliko se, nakon čitanja prvog bajta instrukcije, utvrdi da se radi o adresnim instrukcijama i u odnosu na način adrsiranja čitaju se još jedan ili dva bajta. U slučaju registarskog direktnog načina adresiranja ne čita se vise ni jedan bajt. Odgovarajući registar opšte namene je specificiran bitima registra IR0, $IR_{17..16}$. U slučaju memorijskog direktnog načina adresiranja čitaju se još dva bajta. Tada razredi $IR_{15...8}IR_{7...0}$ predstavljaju apsolutnu adresu memorijske lokacije sa koje je potrebno pročitati operand. Ta vrednost se aktivnom vrednošću signala **outSIR21** propušta kroz bafer sa tri stanja na internu magistralu **Sbus$_{15...0}$**, što omogućava da se vrednost apsolutne adrese upiše u

registar $MAR_{15...0}$. U slučaju registarskog indirektnog sa pomerajem adresiranja čitaju se dva bajta. Tada razredi $IR_{17...16}$ predstavljaju registar opšte namene a razredi $IR_{15...8.}$ predstavljaju pomeraj koji je potrebno dodati vrednost koja je pročitana iz određenog registra opšte namene. Ta vrednost se aktivnom vrednošću signala **outSIR1** propušta kroz bafer sa tri stanja na internu magistralu $\mathbf{Sbus_{15...0}}$, što omogućava da se vrednost apsolutne adrese upiše u registar $MAR_{15...0}$.

Aktivnom vrednošću signala **outSIR0** kroz bafer sa tri stanja na internu magistralu $\mathbf{Sbus_{15...0}}$ propuštaju se biti $IR_{23...16}$ registra IR0. To omogućava da se prilikom intrukcija uslovnog skoka u multiplekseru MX4 bloka *exec* selektuje odgovarajuća vrednost signala na osnovu koje se određuje da li je uslov za skok ispunjen ili nije.

Dekoderi DC1 do DC6 se koriste za dekodovanje istrukcija i formiranje signala logičkih uslova operacija **NOP**, ... , **ROLC** i načina adresiranja **regdir**, ..., **imm** (Slika 19) saglasno načinu kodiranja instrukcija i načina adresiranja.

Dekoder DC1 služi za formiranje signala četiri grupe operacija uslovni skok(G0), bezuslovni skok(G1), bezadresne(G2) i u koliko se radi o adresnim instrukcijama(G3) konkretno o kojoj instrukciji formirajući signale LD, ST, ADD, SUB, AND, OR, XOR, NOT. O kojoj grupi operacija se radi će biti određeno bitima IR_{23} i IR_{20}.

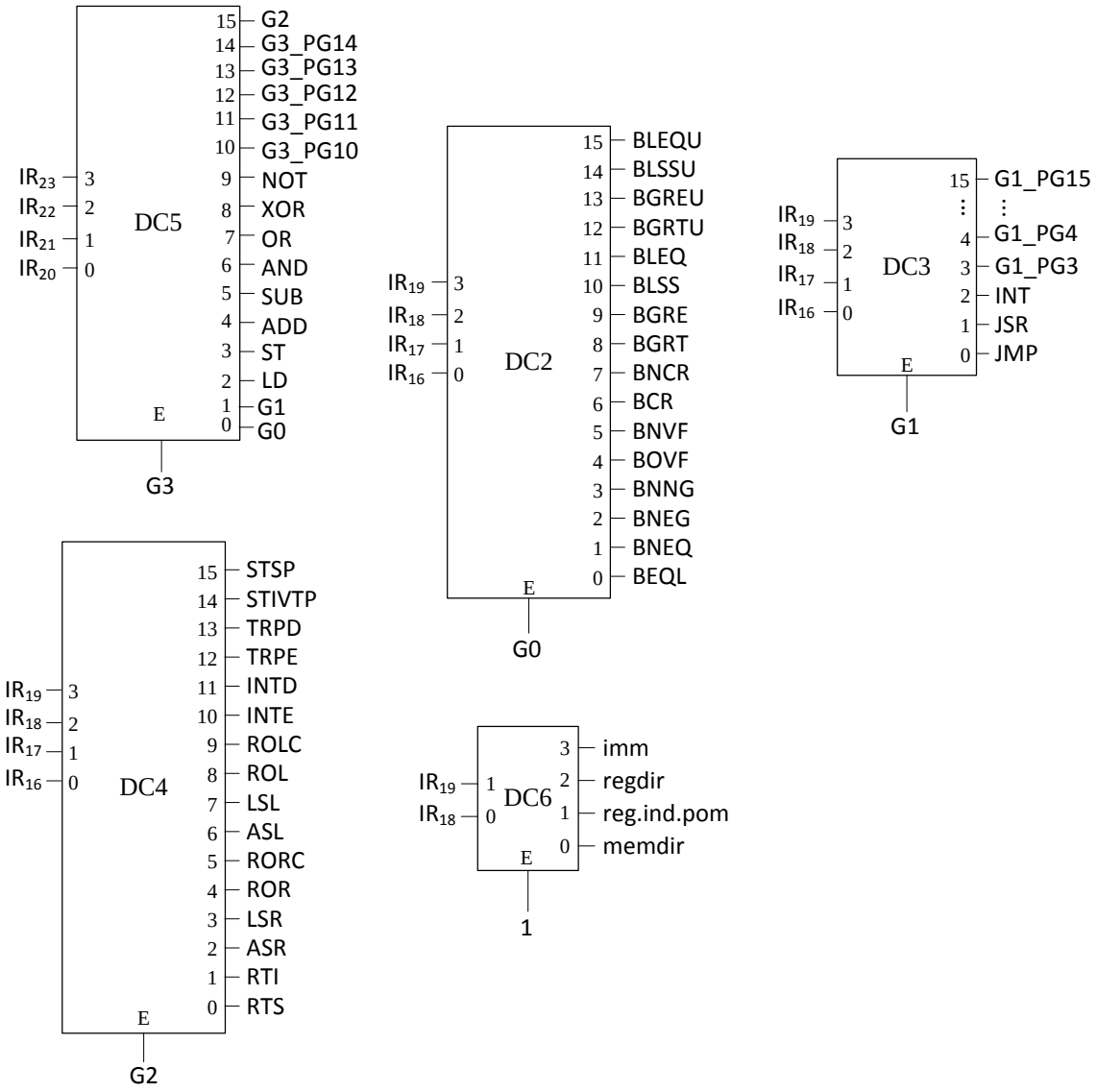
Dekoder DC2 služi za formiranje signala instrukcija iz grupe operacija uslovnog skoka(G0). Ovaj dekoder formira sledeće signale BEQL, BNEQ, BNEG, BNNG, BOVF, BNOVF, BCR, BNCR, BGRT, BGRE, BLSS, BLEQ, BGRTU, BGREU, BLSSU, BLEQU. O kojoj instrukciji se radi biće određeno bitima $IR_{19..16}$.

Dekoder DC3 služi za formiranje signala instrukcija iz grupe operacija bezuslovnog skoka(G1). Ovaj dekoder formira sledeće signale JMP, JSR, INT. O kojoj instrukciji se radi biće određeno bitima $IR_{19..16}$.

Dekoder DC4 služi za formiranje signala instrukcija iz grupe operacija bezadresnih instrukcija(G2). Ovaj dekoder formira sledeće signale RTS, RTI, ASR, LSR, ROR, RORC, ASL, LSL, ROL, ROLC, INTE, INTD, TRPE, TRPD, STIVTP, STSP, NOP, HALT. O kojoj instrukciji se radi biće određeno bitima $IR_{19..16}$.

Dekoder DC5 služi za formiranje signala instrukcija iz grupe operacija adresnih instrukcija(G3). Ovaj dekoder formira sledeće signale LD, ST, ADD, SUB, AND, OR, XOR, NOT. O kojoj instrukciji se radi biće određeno bitima $IR_{19..16}$.

Dekoder DC6 služi za formiranje signala **imm**, **regdir**, **regindpom, memdir** načina adresiranja (Tabela 1). O kom načinu adresiranja se radi će biti određeno bitima IR_{19} i IR_{18}.



Slika 19 Blok *fetch* (drugi deo)

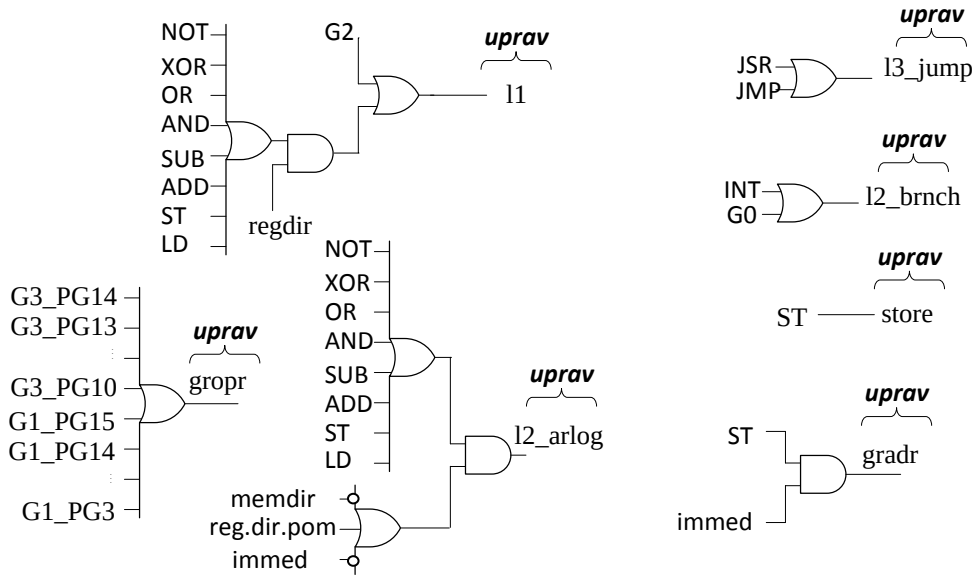
Kombinacione mreže signala logičkih uslova greška operacije, greška adresiranja i dužine instrukcija formiraju signale koji ukazuje da li postoji greška u pročitanoj instrukciji i ukoliko ne postoji kolika je dužina instrukcije u bajtovima (Slika 20).

Signal logičkog uslova greška operacije **gropr** ima vrednost 1 ukoliko se u razredima $IR_{23...16}$ nađe binarna vrednost koja nije dodeljena ni jednoj od operacija iz skupa instrukcija. To se dešava kada je vrednost 1 nekog od signala grupa G0, G1, G2 i G3 koji ne odgovaraju ni jednoj od operacija.

Signal logičkog uslova greška adresiranja **gradr** ima vrednost 1 ukoliko je aktivan signal **ST** i signal **immed**. Greška adresiranja postoji samo ukoliko se radi o instrukciji upisa za koju je zadato neposredno adresiranja, što se utvrđuje na osnovu koda operacije iz 7,6,5 i 4 bita prvog bajta instrukcije i načina adresiranja iz 3 i 2 bita prvog bajta instrukcije.

Signal logičkog uslova dužina instrukcije jedan bajt **l1** svojom vrednošću 1 određuje da je dužina instrukcije jedan bajt, a vrednošću 0 da je dužina instrukcije dva ili tri bajta. Signal **l1** ima vrednost 1 ukoliko se radi o nekoj od bezadresnih instrukcija (G2) ili o adresnoj instrukciji koja koristi registarsko direktno adresiranje. U ostalim situacijama ima vrednost 0.

Signali logičkih uslova dužina instrukcije dva bajta **l2_brnch** i **l2_arlog** svojom vrednošću 1 određuju da je dužina instrukcije dva bajta, a vrednošću 0 da je dužina instrukcije tri bajta. Signal **l2_brnch** ima vrednost 1 ukoliko se radi o instrukciji prekida INT iz grupe G1 ili uslovnim skokovima iz grupe G0. Signal **l2_arlog** ima vrednost 1 ukoliko se radi o nekoj od aritmetičkih, logičkih ili instrukcija prenosa LD, ST, ADD, SUB, AND, OR, XOR, NOT iz grupe adresnih instrukcija G3 za koju je specificirano memorijsko direktno, registarsko direktno sa pomerajem ili neposredno adresiranje.



Slika 20 Blok *fetch* (treći deo)

Signali logičkih uslova dužina instrukcije tri bajta **l3_jump** svojom vrednošću 1 određuju da je dužina instrukcije tri bajta. Signal **l3_jump** ima vrednost 1 ukoliko se radi o instrukcijama bezuslovnog skoka iz grupe G1 (osim instrukcije INT).

5.1.3 Blok addr

Blok *addr* (formiranje adrese i čitanje operanda) sadrži registre opšte namene GPR (R[0] do R[3]) sa adresnim registrom registara opšte namene $GPRAR_{1...0}$, sabirač ADD, pomoćni registar $CW_{15...0}$ i ukazivač na vrh steka $SP_{15...0}$ (Slika 21).

Registri opšte namene GPR su realizovani kao registarski fajl sa 3 registra širine 16 bita (R[0] do R[3]). Adresa registra opšte namene koji se čita ili u koji se upisuje određena je sadržajem registra $GPRAR_{1...0}$ čiji se sadržaj vodi na adresne linije $A_{1...0}$ registarskog fajla GPR. Sadržaj koji se upisuje vodi se sa interne magistrale **$Sbus_{15..0}$** na ulazne linije podataka $DI_{15...0}$ registarskog fajla, a sadržaj koji se čita $GPR_{15...0}$ pojavljuje se na izlaznim linijama podataka $DO_{15...0}$ registarskog fajla. Vrednostima 1 i 0 signala **wrGPR** na upravljačkoj liniji WR registarskog fajla realizuje se upis u registarski fajl i čitanje iz registarskog fajla, respektivno.

Vrednost registarskog fajla se aktivnom vrednošću signala **outDGPR** kroz bafere sa tri stanja propušta na internu magistralu **$Dbus_{15...0}$**. To kod regitarskog direktnog adresiranja

omogućava da se vrednost iz specificiranog registra opšte namene prenese u registar $B_{15...0}$ koji služi za čuvanje operanada.

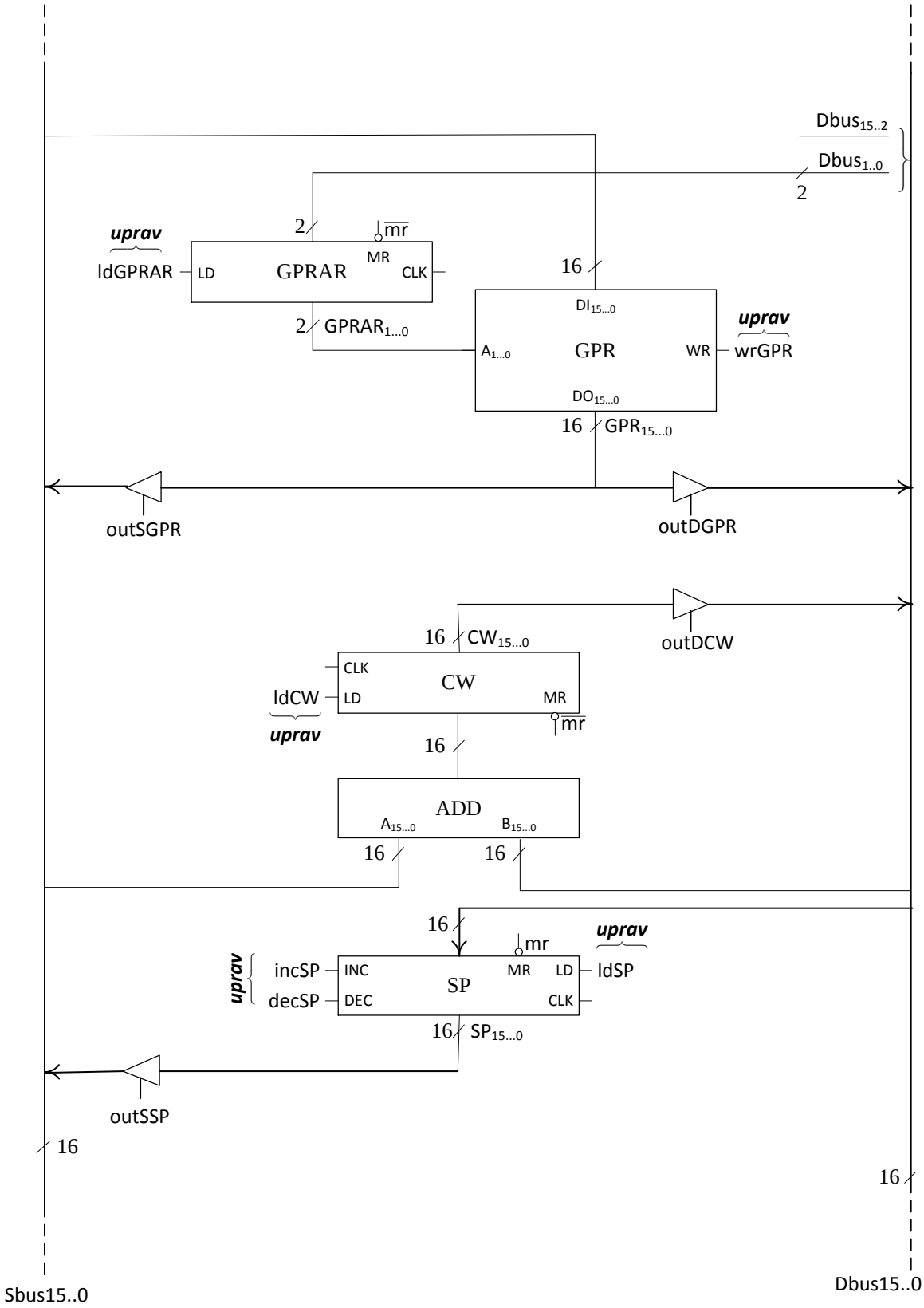
Adresni registar opšte namene $GPRAR_{1...0}$ je dvorazredni registar čiji se sadržaj koristi kao adresa registra registarskog fajla prilikom upisa u registarki fajl i čitanja iz registarskog fajla. U registar $GPRAR_{1...0}$ se vrednošću 1 signala **ldGPRAR** upisuju razredi 1...0 prihvatnog registra podatka $MDR_{7...0}$ bloka *bus*. Ovo se koristi samo u fazi čitanja instrukcije i to prilikom čitanja prvog bajta instrukcije. Tada ova grupa bitova, ukoliko se radi o aritmetičkim ili logičkim instrukcijama sa adresiranjima koja koriste registre opšte namene, predstavlja adresu registra opšte namene. Sadržaj registra $GPRAR_{1...0}$ se vodi na adresne linije $A_{1...0}$ registarskog fajla GPR.

Sabirač ADD je 16-to razredni sabirač koji se koristi za formiranje 16-to bitne vrednosti koja može da bude adresa sa koje treba da se pročita adresa prekidne rutine, zatim adresa sa koje treba da se pročita ili na kojoj treba da se upiše operand i adresa skoka. Sabiranje se realizuje nad sadržajima koji sa internih magistrala $\mathbf{Sbus_{15...0}}$ i $\mathbf{Dbus_{15...0}}$ dolaze na ulaze $A_{15...0}$ i $B_{15...0}$ sabirača ADD, respektivno. Sadržaj $ADD_{15...0}$ sa izlaza sabirača se u bloku *addr* upisuje u registar $CW_{15...0}$ aktivnom vrednošću signala ldCW, a zbog kasnijeg prenosa u blok *bus* radi upisa u adresni registar $MAR_{15...0}$ i u blok *fetch* radi upisa u programski brojač $PC_{15...0}$.

Na ulaze sabirača ADD dolaze sadržaji $IVTP_{15...0}$ i $IVTDSP_{15...0}$, da bi se njihovim sabiranjem na izlazu sabirača ADD dobila adresa sa koje treba da se pročita adresa prekidne rutine. Pri tome je $IVTP_{15...0}$ sadržaj registra koji ukazuje na početak tabele sa adresama prekidnih rutina i $IVTDSP_{15...0}$ pomeraj u odnosu na početak tabele formiran na osnovu broja ulaza u tabelu.

Sadržaji $PC_{15...0}$ i $IR_{15...8}$ dolaze na ulaze ADD sabirača, da bi se njihovim sabiranjem na izlazu sabirača ADD u slučaju instrukcija uslovnog skoka dobila adresa skoka. Pri tome je $PC_{15...0}$ sadržaj programskog brojača i $IR_{15...8}$ pomeraj u odnosu na tekuću vrednost programskog brojača.

Registar $CW_{15...0}$ je 16-to razredni pomoćni registar u kome se čuva suma sa izlaza sabirača ADD i u registar $CW_{15...0}$ upisuje vrednošću 1 signala **ldCW**. Vrednost ovog registra se aktivnom vrednošću signala **outDCW** kroz bafere sa tri stanja propušta na internu magistralu $\mathbf{Dbus_{15...0}}$.



Slika 21 Blok *addr*

Registar $SP_{15...0}$ je 16-to razredni ukazivač na vrh steka čiji sadržaj predstavlja adresu memorijske lokacije prilikom upisa na stek i čitanja sa steka. SP ukazuje na prvu slobodnu lokaciju na vrhu steka. U registar $SP_{15...0}$ se vrednošću 1 signala **ldSP** u fazi izvršavanja instrukcije prenosa upisuje sadržaj 16-to razrednog akumulatora $A_{15...0}$ a preko interne magistrale $\mathbf{Dbus_{15...0}}$. Sadržaj registra $SP_{15...0}$ se inkrementira i dekrementira vrednostima 1 signala **incSP** i **decSP**, što se koristi pri upisu na stek i čitanju sa steka, respektivno. Aktivnom vrednošću signala **outSSP** sadržaj registra $SP_{15...0}$ se propušta kroz bafer sa tri stanja na internu magistralu $\mathbf{Sbus_{15...0}}$. To omogućava da se vrednost iz $SP_{15...0}$ smesti u registar $MAR_{15..0}$, kao adresa sa koje treba pročitati ili na koju treba smestiti podatak.

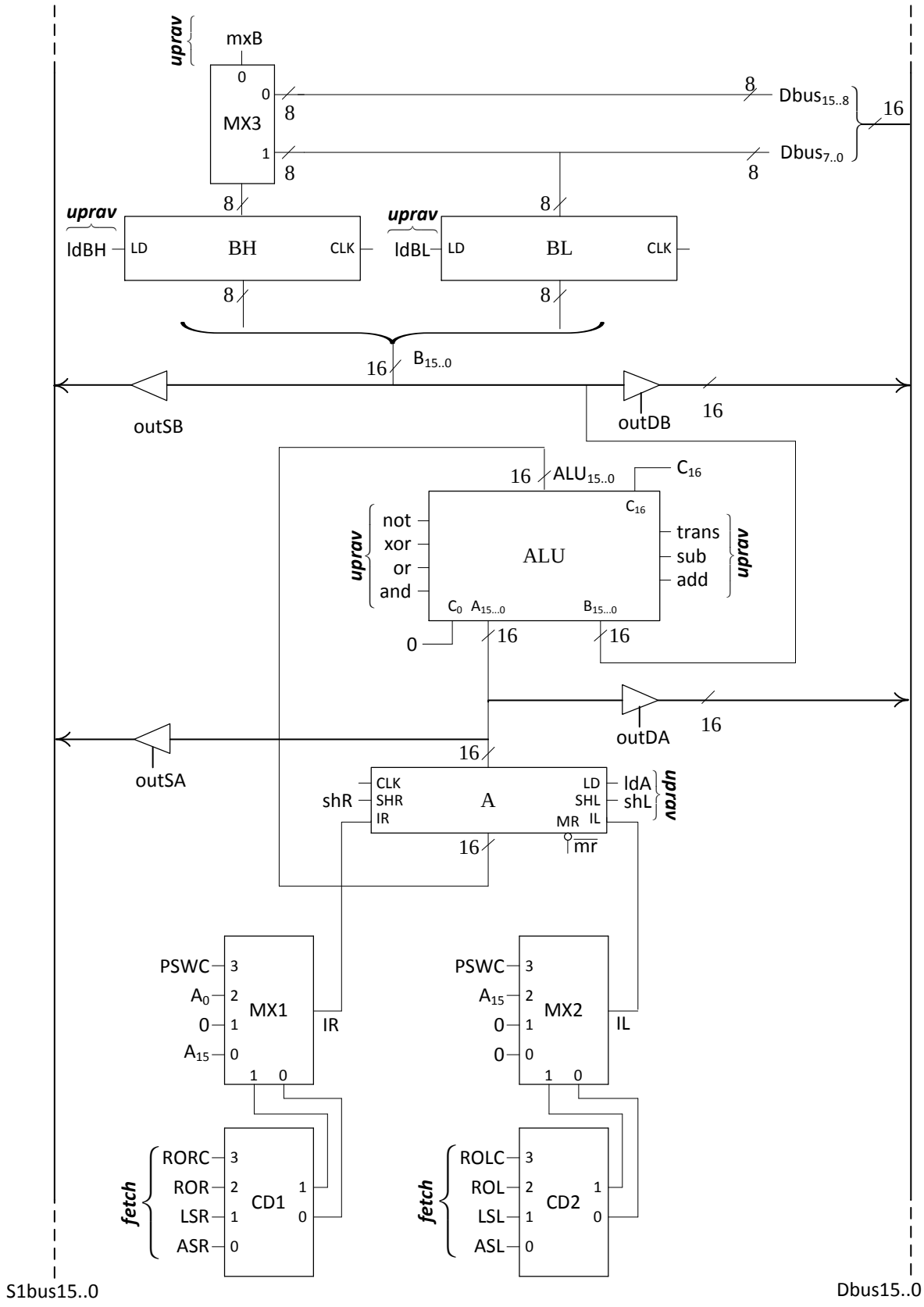
5.1.4 Blok exec

Blok *exec* sadrži aritmetičko logičku jedinicu ALU, registre $A_{15...0}$ (akumulator) i $B_{15...0}$ ($BH_{7...0}$ i $BL_{7...0}$), multipleksere MX1, MX2 i MX3, kodere CD1 i CD2, (Blok exec (prvi deo)), registar $PSW_{15...0}$, flip-flop START (Slika 23), kombinacione mreže za formiranje signala postavljanja indikatora N, Z, C i V (Slika 24) i kombinacione mreže za formiranje signala rezultata operacija **eql**, ..., **nneg**, **brpom** (Slika 25).

Aritmetičko logička jedinica ALU realizuje tri aritmetičke i četiri logičke mikrooperacije nad sadržajima registara $A_{15...0}$ i $B_{15...0}$ (Blok exec (prvi deo)). Mikrooperacija koju treba realizovati se specificira vrednošću 1 ili jednog od upravljačkih signala **add**, **sub** i **trans** za jednu od aritmetičkih mikrooperacija sabiranja, oduzimanja i propuštanje sadržaja registra $B_{15...0}$, respektivno, ili jednog od upravljačkih signala **and**, **or**, **xor** i **not** za jednu od logičkih mikrooperacija I, ILI, ekskluzivno ILI i komplementiranja, respektivno. Rezultat realizovane mikrooperacije se dobija na linijama $ALU_{15...0}$. Na ulaz C_0 je dovedena vrednost 0, pri čemu je vrednost signala C_0, bitna samo u slučaju aritmetičkih mikrooperacija, dok ta vrednost nije bitna u slučaju logičkih mikrooperacija. U slučaju aritmetičke mikrooperacije sabiranja na izlazu C_{16} se dobija prenos, a u slučaju aritmetičke mikrooperacije oduzimanja na izlazu C_{16} se dobija pozajmica. U slučaju logičkih mikrooperacija signal na izlazu C_{16} nema smisla.

Registar $A_{15...0}$ je 16-to razredni akumulator koji se koristi kao implicitno izvorište i odredište u aritmetičkim, logičkim i pomeračkim instrukcijama. Rezultat izvršavanja aritmetičko logičke operacije se vodi na ulaze registra $A_{15...0}$ i u njega upisuje vrednošću 1 signala **ldA**. Sadržaj registra $A_{15...0}$ se pomera udesno za jedno mesto vrednošću 1 signala **shR**. Tada se u najstariiji razred registra A_{15} upisuje signal **IR** koji dolazi sa izlaza multipleksera MX1. Sadržaj registra $A_{15...0}$ se pomera ulevo za jedno mesto vrednošću 1 signala **shL**. Tada se u najmlađi razred registra A_0 upisuje signal **IL** koji dolazi sa izlaza multipleksera MX2. Sadržaj registra $A_{15...0}$ se vodi na ulaze ALU gde se koristi kao prvi izvorišni operand u slučaju aritmetičkih i logičkih operacija.

Sadržaj registra $A_{15...0}$ se aktivnom vrednošću signala **outSA** propušta kroz bafere sa tri stanja na interne magistrale $\mathbf{Sbus_{15...0}}$ respektivno.



Slika 22 Blok *exec* (prvi deo)

Multiplekser MX1 je 1-no razredni multiplekser sa 4 ulaza. Na ulaze 0 do 3 multipleksera se dovode signali $\mathbf{A_{15}}$, **0**, $\mathbf{A_0}$ i **PSWC** koji se propuštaju kroz multiplekser i po liniji IR upisuju u razred A_{15} registra $A_{15...0}$ pri realizaciji instrukcije aritmetičkog pomeranja sadržaja akumulatora udesno za jedno mesto (ASR), instrukcije logičkog pomeranja sadržaja akumulatora udesno za jedno mesto (LSR), instrukcije rotiranja sadržaja akumulatora udesno za jedno mesto (ROR) i instrukcije rotiranja sadržaja akumulatora i indikatora C udesno za jedno mesto (RORC), respektivno. Selekcija jednog od ova četiri signala ($\mathbf{A_{15}}$, **0**, $\mathbf{A_0}$ i **PSWC**) se realizuje binarnim vrednostima 00 do 11 sadržaja sa izlaza kodera CD1. Signal $\mathbf{A_{15}}$ se propušta za instrukciju aritmetičkog pomereranja udesno kada signal operacije ASR ima vrednost 1. Signal **0** se propušta za instrukciju logičkog pomereranja udesno kada signal operacije LSR ima vrednost 1. Signal $\mathbf{A_0}$ se propušta za instrukciju rotiranja udesno kada signal operacije ROR ima vrednost 1. Signal **PSWC** se propušta za instrukciju rotiranja kroz indikator PSWC udesno kada signal operacije RORC ima vrednost 1.

Multiplekser MX2 je 1-no razredni multiplekser sa 4 ulaza. Na ulaze 0 do 3 multipleksera se dovode signali **0**, **0**, $\mathbf{A_{15}}$ i **PSWC** koji se propuštaju kroz multiplekser i po liniji IL upisuju u razred A_0 registra $A_{15...0}$ pri realizaciji instrukcije aritmetičkog pomeranja sadržaja akumulatora ulevo za jedno mesto (ASL), instrukcije logičkog pomeranja sadržaja akumulatora ulevo za jedno mesto (LSL), instrukcije rotiranja sadržaja akumulatora ulevo za jedno mesto (ROL) i instrukcije rotiranja sadržaja akumulatora i indikatora C ulevo za jedno mesto (ROLC), respektivno. Selekcija jednog od ova četiri signala se realizuje binarnim vrednostima 00 do 11 sadržaja sa izlaza kodera CD2. Signal **0** se propušta za instrukciju aritmetičkog pomereranja ulevo kada signal operacije ASL ima vrednost 1. Signal **0** se propušta za instrukciju logičkog pomereranja ulevo kada je aktivan signal operacije LSL. Signal $\mathbf{A_{15}}$ se propušta za instrukciju rotiranja ulevo kada signal operacije ROL ima vrednost 1. Signal **PSWC** se propušta za instrukciju rotiranja kroz indikator PSWC ulevo kada signal operacije ROLC ima vrednost 1.

Koder CD1 je koder sa četiri ulaza i dva izlaza koji na izlazima daje binarnu vrednost 00 do 11 u zavisnosti od toga koji od signala operacija pomeranja i rotiranja udesno za jedno mesto ASR, LSR, ROR i RORC ima vrednost 1. Ukoliko signal operacije aritmetičkog pomeranja udesno ASR koji je povezan na ulaz 0 kodera ima vrednost 1, na izlazima kodera se pojavljuje binarna vrednost 00, koja omogućuje da se kroz multiplekser MX1 na liniju IR registra $A_{15...0}$ propusti signal $\mathbf{A_{15}}$. Ukoliko signal operacije logičkog pomeranja udesno LSR koji je povezan na ulaz 1 kodera ima vrednost 1, na izlazima kodera se pojavljuje binarna vrednost 01, koja omogućuje da se kroz multiplekser MX1 na liniju IR registra $A_{15...0}$ propusti signal **0**. Ukoliko signal operacije rotiranja udesno ROR koji je povezan na ulaz 2 kodera ima vrednost 1, na izlazima kodera se pojavljuje binarna vrednost 10, koja omogućuje da se kroz multiplekser MX1 na liniju IR registra $A_{15...0}$ propusti signal $\mathbf{A_0}$. Ukoliko signal operacije rotiranja udesno kroz indikator PSWC RORC koji je povezan na ulaz 3 kodera ima vrednost 1, na izlazima kodera se pojavljuje binarna vrednost 11, koja omogućuje da se kroz multiplekser MX1 na liniju IR registra $A_{15...0}$ propusti signal **PSWC**.

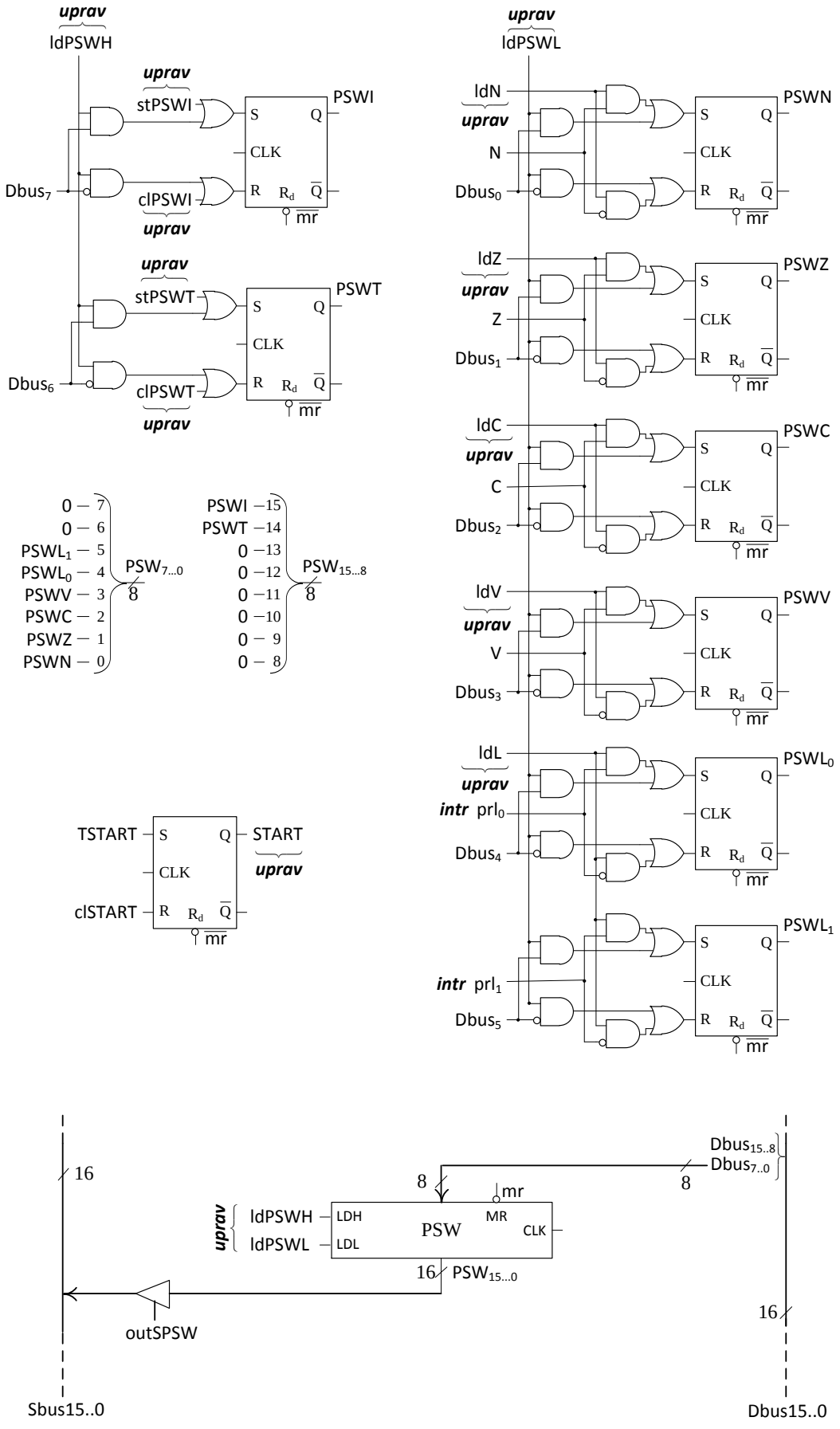
Koder CD2 je koder sa četiri ulaza i dva izlaza koji na izlazima daje binarnu vrednost 00 do 11 u zavisnosti od toga koji od signala operacija pomeranja i rotiranja ulevo za jedno mesto ASL, LSL, ROL i ROLC ima vrednost 1. Ukoliko signal operacije aritmetičkog pomeranja ulevo ASL koji je povezan na ulaz 0 kodera ima vrednost 1, na izlazima kodera se pojavljuje binarna vrednost 00, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $A_{15...0}$ propusti signal **0**. Ukoliko signal operacije logičkog pomeranja ulevo LSL koji je povezan na ulaz 1 kodera ima vrednost 1, na

izlazima kodera se pojavljuje binarna vrednost 01, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $A_{15...0}$ propusti signal **0**. Ukoliko signal operacije rotiranja ulevo ROL koji je povezan na ulaz 2 kodera ima vrednost 1, na izlazima kodera se pojavljuje binarna vrednost 10, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $A_{15...0}$ propusti signal $\mathbf{A_{15}}$. Ukoliko signal operacije rotiranja ulevo kroz indikator PSWC ROLC koji je povezan na ulaz 3 kodera ima vrednost 1, na izlazima kodera se pojavljuje binarna vrednost 11, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $A_{15...0}$ propusti signal **PSWC**.

Registar $B_{15...0}$ je 16-to razredni registar koji se koristi za prihvatanje 16-to bitne veličine koja se dobija iz dve susedne 8- bitne memorijske lokacije u dva posebna ciklusa na magistrali i prihvatni registar u koji se privremeno smešta izvorišni operand specificiran adresnim delom svih instrukcija sa jednoadresnim formatom. On se sastoji iz dva 8-mo razredna registra $BH_{7..0}=B_{15..8}$ i $BL_{7..0}=B_{7..0.}$ Na ulaze registra $BL_{7...0}= B_{7...0}$ se vodi viših 8 bitova sadržaja ($GPR_{15...0}$, $MDR_{7...0}$ proširen nulama…) sa interne magistrale $\mathbf{Dbus_{15...0}}$($\mathbf{Dbus_{7...0}}$), koji predstavljaju vrednost iz registra $MDR_{7...0}$ pri čitanju prvog bajta 16-bitne veličine koja se dobija iz dve susedne 8- bitne memorijske lokacije ili nižih 8 bitova sadržaja nekog od registara opšte namene $GPR_{15...0}$. U registar $BL_{7...0}$ se sadržaj upisuje vrednošću 1 signala **ldBL**. Na ulaze registra $BH_{7...0}= B_{15...0}$ se vodi izlaz multipleksera MX3. Binarnom vrednošću 1 signala **mxB** kroz multiplekser MX3 se propušta sadržaj sa interne magistrale $\mathbf{Dbus_{7...0}}$, tj. nižih 8 bitova sadržaja sa interne magistrale $\mathbf{Dbus_{15...0}}$ koji zapravo predstavljaju vrednost iz registra $MDR_{7...0}$ pri čitanju drugog bajta 16-bitne veličine koja se dobija iz dve susedne 8- bitne memorijske lokacije. Binarnom vrednošću 0 signala **mxB** kroz multiplekser MX3 se propušta sadržaj sa interne magistrale $\mathbf{Dbus_{15...8}}$, tj. viših 8 bitova sadržaja sa interne magistrale $\mathbf{Dbus_{15...0}}$ koji predstavljaju viših 8 bitova sadržaja nekog od registara opšte namene $GPR_{15...0}$. Sadržaj se u registar $BH_{7...0}$ upisuje vrednošću 1 signala **ldBH**.

Prihvatanje 16-to bitne veličine koja se dobija iz dve susedne 8- bitne memorijske lokacije se obavlja u dva ciklusa na magistrali, dok se prihvatanje vrednosti iz nekog od registara opšte namene $GPR_{15...0}$ obavlja u jednom ciklusu istovremenim upisom viših 8 bitova sadržaja interne magistrale $\mathbf{Dbus_{15...0}}$ u registar $BL_{7...0}$ i nižih 8 bitova sadržaja interne magistrale $\mathbf{Dbus_{15...0}}$ u registar $BH_{7...0}$.

Vrednošću 1 signala **Bout** sadržaj registra $B_{15...0}$ se propušta kroz bafere sa tri stanja na internu magistralu $\mathbf{Dbus_{15...0}}$.



Slika 23 Blok *exec* (drugi deo)

Registar $PSW_{15...0}$ je 16-to razredni registar koji sadrži indikatore programske statusne reči procesora (Slika 23). Registar $PSW_{15...0}$ se sastoji od flip-flopova PSWI, PSWT, $PSWL_1$, $PSWL_0$, PSWV, PSWC, PSWZ i PSWN, koji predstavljaju razrede $PSW_{15.14}$ i $PSW_{5...0}$, respektivno, dok razredi $PSW_{13...6}$ ne postoje.

Flip-flop PSWI sadrži indikator *interrupt*. Flip-flop se postavlja na vrednost 1 vrednošću 1 signala **stPSWI** u okviru faze izvršavanja instrukcije INTE i na vrednost 0 vrednošću 1 signala **clPSWI** u okviru faze izvršavanja instrukcije INTD kao i u okviru faze opsluživanja zahteva za prekid svih instrukcija ukoliko je tokom izvršavanja instrukcije stigao zahtev za prekid pa se prolazi kroz ovu fazu.

Flip-flop PSWT sadrži indikator *trap*. Flip-flop PSWT se postavlja na vrednost 1 vrednošću 1 signala **stPSWT** u okviru faze izvršavanja instrukcije TRPE i na vrednost 0 vrednošću 1 signala **clPSWT** u okviru faze izvršavanja instrukcije TRPD kao i u okviru faze opsluživanja zahteva za prekid svih instrukcija ukoliko je tokom izvršavanja instrukcije stigao zahtev za prekid pa se prolazi kroz ovu fazu.

Flip-flopovi $PSWL_1$ do $PSWL_0$ sadrže indikatore *level*. U flip-flopove $PSWL_1$ do $PSWL_0$ se vrednošću 1 signala **ldL** u okviru faze opsluživanja zahteva za prekid od ulazno/izlaznih uređaja upisuju vrednosti signala $\mathbf{prl_1}$ do $\mathbf{prl_0}$, čime se ovi flip-flopovi postavljaju na vrednost nivoa prioriteta ulazno/izlaznog uređaja na čiju se prekidnu rutinu prelazi.

Flip-flop PSWV sadrži indikator *overflow*. U flip-flop PSWV se vrednošću 1 signala **ldV** u okviru faze izvršavanja određenih instrukcija upisuje vrednost signala **V**.

Flip-flop PSWC sadrži indikator *carry/borrow*. U flip-flop PSWC se vrednošću 1 signala **ldC** u okviru faze izvršavanja određenih instrukcija upisuje vrednost signala **C**.

Flip-flop PSWZ sadrži indikator *zero*. U flip-flop PSWZ se vrednošću 1 signala **ldZ** u okviru faze izvršavanja određenih instrukcija upisuje vrednost signala **Z**.

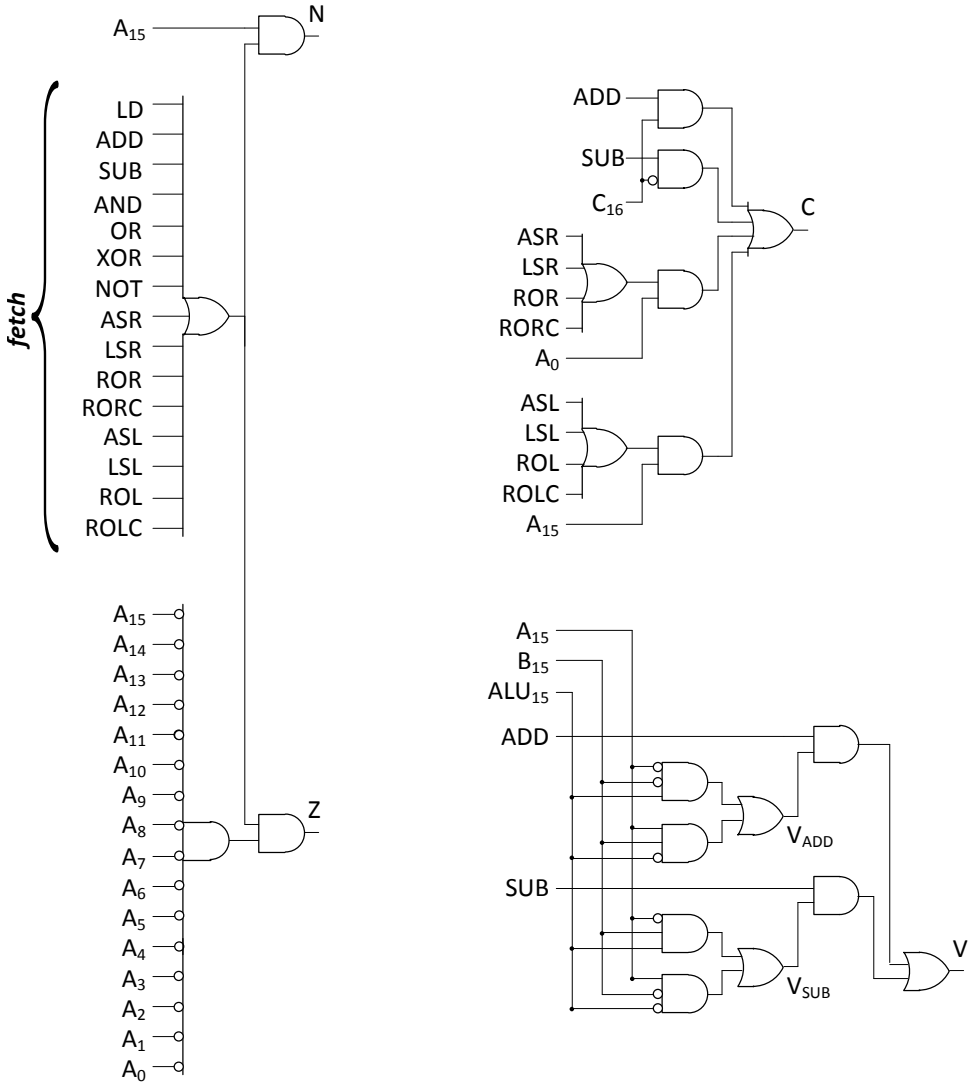
Flip-flop PSWN sadrži indikator *negative*. U flip-flop PSWN se vrednošću 1 signala **ldN** u okviru faze izvršavanja određenih instrukcija upisuje vrednost signala **N**.

U razrede $PSW_{15.14}$ koji predstavljaju 2 najstarija razreda registra $PSW_{15...0}$ se vrednošću 1 signala **ldPSWH** upisuje sadržaj razreda $MDR_{7.6}$ prihvatnog registra podatka $MDR_{7...0}$ bloka *bus*, koji na ulaz registra $PSW_{15...0}$ dolaze preko interne magistrale $\mathbf{Dbus_{15...0}}$, dok se u razrede $PSW_{6...0}$ koji predstavljaju 7 najmlađih razreda registra $PSW_{15...0}$ vrednošću 1 signala **ldPSWL** upisuje sadržaj razreda $MDR_{5...0}$ prihvatnog registra podatka $MDR_{7...0}$, koji na ulaz registra $PSW_{15...0}$ dolaze preko interne magistrale $\mathbf{Dbus_{15...0}}$. Ovo se koristi prilikom izvršavanja instrukcije RTI da se sadržajem sa vrha steka restaurira sadržaj registra $PSW_{15...0}$. Sadržaj registra $PSW_{15..0}$ se vodi na internu magistralu $\mathbf{Sbus_{15..0}}$ aktivnom vrednošću signala **outSPSW**. Ovo se koristi prilikom opsluživanja zahteva za prekida da se sadržaj registra $PSW_{15...0}$ stavi na vrh steka.

Flip-flop START svojom vrednošću 1 omogućuje izvršavanje instrukcija, dok vrednošću 0 zaustavlja. U flip-flop START se upisuje vrednost 1 vrednošću 1 signala **TSTART** koja se generiše ili pri uključenju napajanja ili aktiviranjem tastera START. U flip-flop se upisuje vrednost 0 vrednošću 1 signala **clSTART** koja se generiše u okviru faze izvršavanja instrukcije HALT.

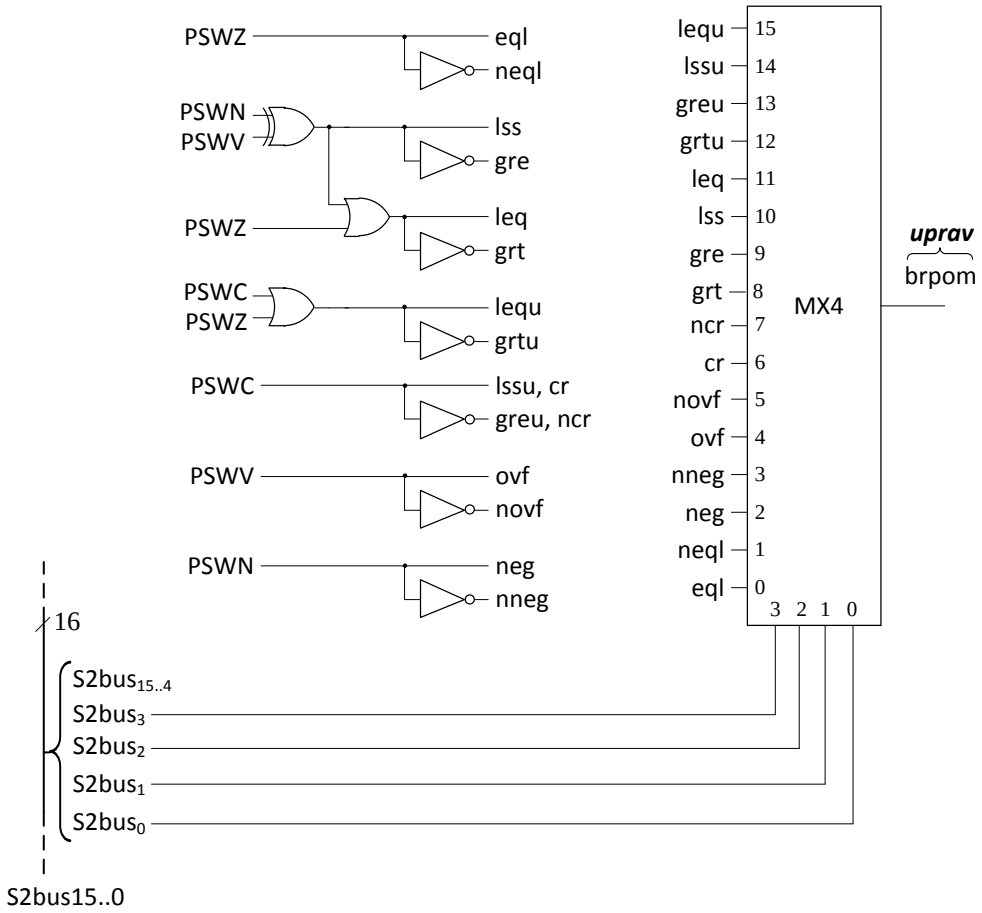
Kombinacione mreže signala postavljanja indikatora N, Z, C i V se sastoje od logičkih elemenata (Slika 24). Na izlazima kombinacionih mreža se formiraju signali koji se upisuju u flip-flopove PSWN, PSWZ, PSWC i PSWV programske statusne reči u okviru izvršavanja određenih instrukcija. Signal **N** ima vrednost koja odgovara vrednosti razreda

A_{15} ukoliko jedan od signala operacija LD, ..., ROLC ima vrednost 1, dok u svim ostalim situacijama signal **N** ima vrednost 0. Signal **Z** ima vrednost 1 ukoliko svi razredi registra $A_{15...0}$ imaju vrednost 0 i ukoliko jedan od signala operacija LD, ..., ROLC ima vrednost 1, dok u svim ostalim situacijama signal **Z** ima vrednost 0. Signal **C** ima vrednost koja odgovara vrednosti signala $\mathbf{C_{16}}$ ukoliko jedan od signala operacija ADD ili SUB ima vrednost 1. Pored toga signal **C** ima vrednost koja odgovara vrednosti signala $\mathbf{A_0}$ ukoliko jedan od signala operacija ASR, LSR, ROR ili RORC ima vrednost 1 i vrednost koja odgovara vrednosti signala $\mathbf{A_{15}}$ ukoliko jedan od signala operacija ASL, LSL, ROL ili ROLC ima vrednost 1. U svim ostalim situacijama signal **C** ima vrednost 0. Signal **V** ima vrednost koja odgovara vrednosti jednog od signala $\mathbf{V_{ADD}}$ i $\mathbf{V_{SUB}}$, ukoliko jedan od signala operacija ADD ili SUB ima vrednost 1, respektivno, dok u svim ostalim situacijama signala **V** ima vrednost 0. Signal $\mathbf{V_{ADD}}$ ima vrednost 1 ukoliko signali $\mathbf{A_{15}}$ i $\mathbf{B_{15}}$ imaju vrednost 0 i signal $\mathbf{ALU_{15}}$ ima vrednost 1 ili ukoliko signali $\mathbf{A_{15}}$ i $\mathbf{B_{15}}$ imaju vrednost 1 i signal $\mathbf{ALU_{15}}$ vrednost 0. Signal $\mathbf{V_{SUB}}$ ima vrednost 1 ukoliko signali $\mathbf{ALU_{15}}$ i $\mathbf{B_{15}}$ imaju vrednost 0 i signal $\mathbf{A_{15}}$ vrednost 1 ili ukoliko signali $\mathbf{ALU_{15}}$ i $\mathbf{B_{15}}$ imaju vrednost 1 i signal $\mathbf{A_{15}}$ ima vrednost 0.



Slika 24 Blok *exec* (treći deo)

Kombinacione mreže za formiranje signala rezultata operacija **eql**, ..., **lequ** i **brpom** se sastoje od logičkih elemenata i multipleksera MX4 (Slika 25). Logičkim elementima se na osnovu sadržaja flip-flopova PSWN, PSWZ, PSWC i PSWV programske statusne reči formiraju signali **eql**, ..., **lequ**. Multiplekserom MX4 se na osnovu razreda $IR_{19...16}$ prihvatnog registra instrukcije kojima se specificira kod operacije instrukcije uslovnog skoka BEQL, ..., BLEQU, a koji se do multipleksera dovode preko razreda $Sbus_{3...0}$ interne magistrale $Sbus_{15..0}$, selektuje jedan od signala **eql**, ..., **lequ** i pojavljuje kao signal **brpom** koji se koristi u upravljačkoj jedinici ***uprav*** da bi se prilikom izvršavanja instrukcija uslovnog skoka utvrdilo da li je uslov za skok ispunjen ili nije. Selekcija jednog od signala **eql**, ..., **lequ** i formiranje signala **brpom** se realizuje na osnovu kodiranja odgovarajućih kodova operacija instrukcija uslovnog skoka BEQL, ..., BLEQU. Kodiranje instrukcija je tako realizovano da se razredima $IR_{19...16}$ selektuje odgovarajuća instrukcija iz grupe instrukcija uslovnog skoka.



Slika 25 Blok *exec* (četvrti deo)

Signali rezultata operacija **eql**, ..., **nneg** imaju sledeće značenje:

eql — rezultat nula,
neql — rezultat različit od nule,
neg — razred PSWN aktivan,
nneg — razred PSWN nije aktivan,
ovf — razred PSWV aktivan,
novf — razred PSWV nije aktivan,
cr — razred PSWC aktivan,

ncr — razred PSWC nije aktivan,
grt — rezultat veći od nule u aritmetici sa znakom,
gre — rezultat veći od nule ili jednak nuli u aritmetici sa znakom,
lss — rezultat manji od nule u aritmetici sa znakom,
leq — rezultat manji od nule ili jednak nuli u aritmetici sa znakom,
grtu — rezultat veći od nule u aritmetici bez znaka,
greu — rezultat veći od nule ili jednak nuli u aritmetici bez znaka,
lssu — rezultat manji od nule u aritmetici bez znaka,
lequ — rezultat manji od nule ili jednak nuli u aritmetici bez znaka.

Signali rezultata operacija **eql**, ..., **lequ** se realizuju prema sledećim izrazima:
eql = **PSWZ**,
neq = $\overline{\mathbf{PSWZ}}$,
neg = **PSWN**,
nneg = $\overline{\mathbf{PSWN}}$,
ovf = **PSWV**,
novf = $\overline{\mathbf{PSWV}}$,
cr = **PSWC**,
ncr = $\overline{\mathbf{PSWC}}$,
grt = $\overline{((\mathbf{PSWN} \oplus \mathbf{PSWV}) + \mathbf{PSWZ})}$,
gre = $\overline{(\mathbf{PSWN} \oplus \mathbf{PSWV})}$,
lss = **PSWN** $\oplus$ **PSWV**,
leq = (**PSWN** $\oplus$ **PSWV**) + **PSWZ**,
grtu = $\overline{\mathbf{PSWC}} + \overline{\mathbf{PSWZ}}$,
greu = $\overline{\mathbf{PSWC}}$,
lssu = **PSWC**,
lequ = **PSWC** + **PSWZ**.

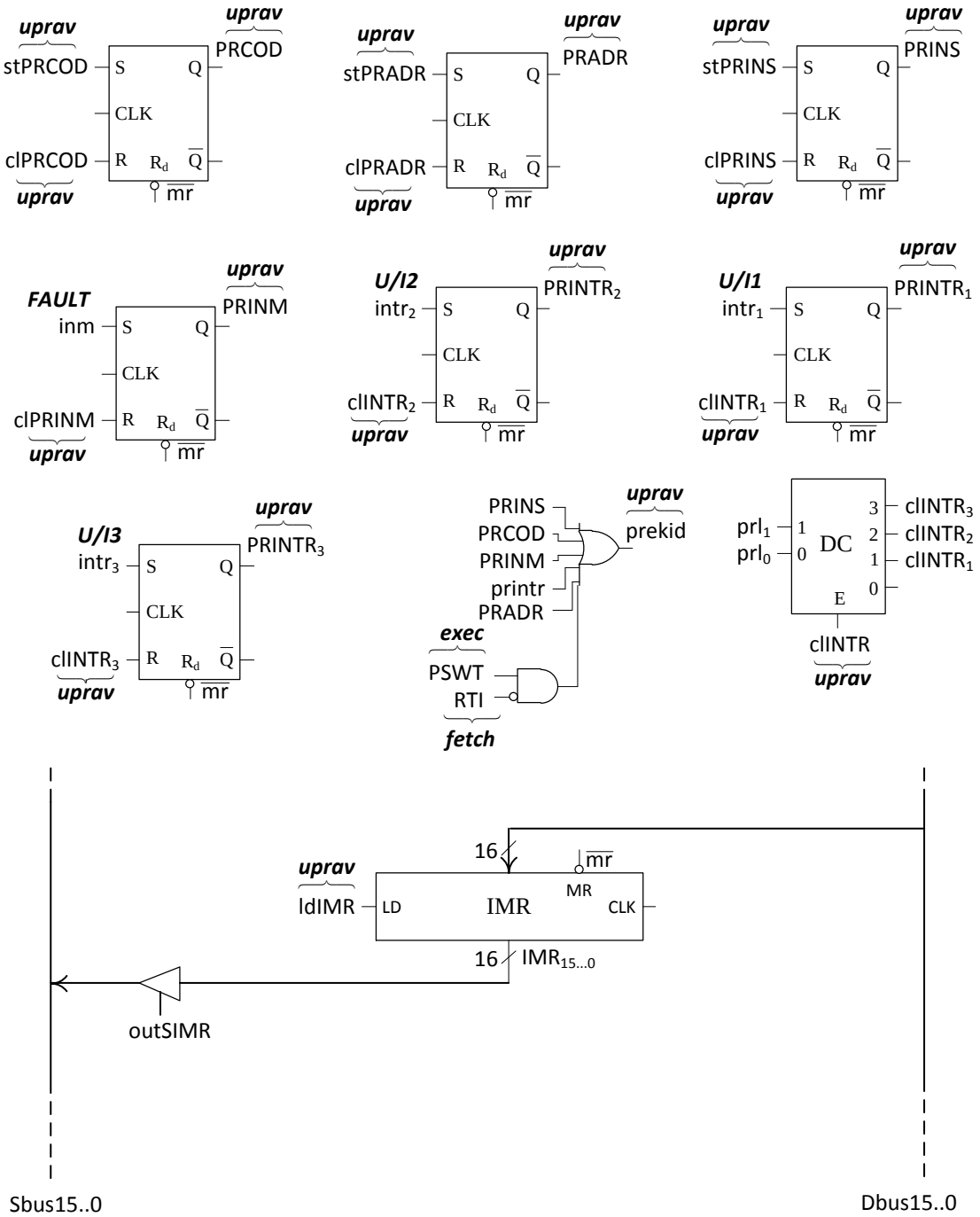
5.1.5 Blok intr

Blok *intr*(opsluživanje prekida) sadrži logički ILI element za formiranje signala prekida **prekid**, kombinacione i sekvencijalne mreže za prihvatanje unutrašnjih prekida i spoljašnjih nemaskirajućih i maskirajućih prekida, registar $IMR_{15...0}$ (Slika 26), kombinacione prekidačke mreže za formiranje signala spoljašnjih maskirajućih prekida **printr** (Slika 27), kombinacione i sekvencijalne mreže za formiranje pomeraja za tabelu sa adresama prekidnih rutina $IVTDSP_{15...0}$ i registar $IVTP_{15...0}$ (Slika 28).

Logički ILI element za formiranje signala prekida **prekid** daje vrednost 1 signala prekida **prekid** ukoliko barem jedan od signala prekida ima vrednost 1 (Slika 26). To može da bude neki od signala unutrašnjih prekida i to **PRINS** za prekid zbog izvršavanja instrukcije prekida INT i **PRCOD** za prekid usled čitanja instrukcije sa nepostojećim kodom operacije, zatim signal spoljašnjeg prekida **PRINM** zbog generisanja nemaskirajućeg prekida, potom signal spoljašnjeg prekida **printr** zbog prisustva nekog od maskirajućih prekida i na kraju signal koji predstavlja I funkciju signala unutrašnjeg prekida **PSWT** usled prekida zbog zadatog režim rada procesora prekid posle svake instrukcije i komplementa signala operacije povratka iz prekidne rutine **RTI**. Treba uočiti da time što se uzima I funkciju signala **PSWT** i komplementa signala se obezbeđuje da režim rada prekid posle svake instrukcije nije dozvoljen za instrukciju povratka iz prekidne rutine RTI .

Kombinacione i sekvencijalne mreže za prihvatanje unutrašnjih zahteva za prekid (Slika 27) se sastoje od flip-flopova PRINS i PRCOD.

Flip-flop PRINS pamti unutrašnji zahtev za prekid izazvan izvršavanjem instrukcije prekida INT. Ovaj flip-flop se postavlja na vrednost 1 vrednošću 1 signala **stPRINS** u fazi izvršavanja instrukcije prekida INT i na vrednost 0 vrednošću 1 signala **clPRINS** u okviru opsluživanja ovog zahteva za prekid.



Slika 26 Blok *intr* (prvi deo)

Flip-flop PRCOD pamti unutrašnji zahtev za prekid zbog čitanja instrukcije sa nepostojećim kodom operacije. Flip-flop PRCOD se postavlja na vrednost 1 vrednošću 1 signala **stPRCOD** koji se generiše u fazi čitanja instrukcije ukoliko je u bloku *fetch* otkriven nepostojeći kod operacije i formirana vrednost 1 signala **gropr** (Slika 20). Flip-flop PRCOD se postavlja na vrednost 0 vrednošću 1 signala **clPRCOD** u okviru opsluživanja ovog zahteva za prekid.

Flip-flop PRADR pamti unutrašnji zahtev za prekid zbog korišćenja neposrednog adresiranja za odredišni operand. Flip-flop PRADR se postavlja na vrednost 1 vrednošću 1 signala **stPRADR** koji se generiše u fazi čitanja instrukcije ukoliko je u bloku *fetch* otkriveno korišćenja neposrednog adresiranja za odredišni operand i formirana vrednost 1 signala **gradr** (Slika 20). Flip-flop PRADR se postavlja na vrednost 0 vrednošću 1 signala **clPRADR** u okviru opsluživanja ovog zahteva za prekid

Signali **PRINS**, **PRCOD** i **PRADR** se koriste u upravljačkoj jedinici ***uprav*** kao signali logičkih uslova prilikom opsluživanja prekida.

Kombinacione i sekvencijalne mreže za prihvatanje spoljašnjih nemaskirajućih i maskirajućih prekida se sastoje od flip-flopova PRINM, $PRINTR_1$ do $PRINTR_3$ i dekodera DC (Slika 26).

Flip-flop PRINM služi za prihvatanje spoljašnjeg nemaskirajućeg zahteva za prekid **inm**. Zahtev za prekid **inm** dolazi od uređaja ***FAULT*** kao impuls i pamti se u flip-flopu PRINM na prvi signal takta. Flip-flop PRINM se postavlja na vrednost 0 vrednošću 1 signala **clPRINM** u okviru opsluživanja ovog zahteva za prekid.

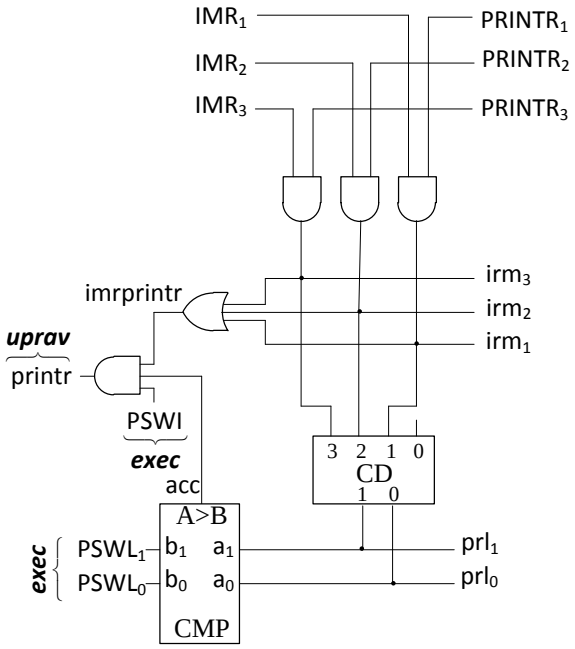
Flip-flopovi $PRINTR_1$ do $PRINTR_3$ služe za prihvatanje spoljašnjih maskirajućih zahteva za prekid $intr_1$ do $intr_3$. Zahtevi za prekid $intr_1$ do $intr_3$ koji dolaze od ulazno/izlaznih uređaja ***U/I*** kao impuls i pamte se u flip-flopovima za prihvatanje prekida $PRINTR_1$ do $PRINTR_3$, respektivno. U obradi prekida koriste se sadržaji flip-flopova $PRINTR_1$ do $PRINTR_3$. Kada se u okviru opsluživanja zahteva za prekid prihvati neki od maskirajućih zahteva za prekid, odgovarajući flip-flop $PRINTR_1$ do $PRINTR_3$ se postavlja na vrednost 0. Postavljanje flip-flopova $PRINTR_1$ do $PRINTR_3$ na vrednost 0 se realizuje u okviru opsluživanja spoljašnjeg maskirajućeg zahteva za prekid vrednošću 1 odgovarajućeg signala **$clINTR_1$** do **$clINTR_3$**, respektivno. Postavljanje odgovarajućeg flip-flopa $PRINTR_1$ do $PRINTR_3$ na vrednost 0 se realizuje tako što se u okviru opsluživanja spoljašnjeg maskirajućeg zahteva za prekid generiše vrednost 1 signala **clINTR**. S obzirom da signali **prl_1** do **prl_0** daju binarnu vrednost jedne od linija $intr_1$ do $intr_3$ po kojoj je prihvaćeni zahtev za prekid stigao, ovi signali se koriste da selektuju odgovarajući flip-flop $PRINTR_1$ do $PRINTR_3$ koji se vrednošću 1 signala **clINTR** postavlja na vrednost 0.

Dekoder DC u okviru opsluživanja spoljašnjeg maskirajućeg zahteva za prekid pri vrednosti 1 signala **clINTR** daje vrednost 1 jednog od signala **$clINTR_1$** do **$clINTR_3$**. Koji će od signala **$clINTR_1$** do **$clINTR_3$** tada imati vrednost 1 zavisi od vrednosti signala **prl_1** do **prl_0** koji daju binarnu vrednost linije $intr_1$ do $intr_3$ po kojoj je prihvaćeni zahtev za prekid stigao.

Registar maske $IMR_{15...0}$ služi za pojedinačno maskiranje spoljašnjih maskirajućih zahteva za prekid $intr_1$ do $intr_3$. Zahtevima za prekid $intr_1$ do $intr_3$ odgovaraju razredi IMR_1 do IMR_3 registra maske $IMR_{15...0}$, dok se preostali razredi ne koriste. U ovom procesoru ne postoji instrukcije koja služi za upis sadržaja u registar procesora $IMR_{15...0}$. Sadržaj registra $IMR_{15..0}$ se vodi na internu magistralu **$Sbus_{15..0}$** aktivnom vrednošću signala.

Kombinacione prekidačke mreže za formiranje signala spoljašnjih maskirajućih prekida **printr** se sastoji od logičkih ILI i I elemenata, kodera CD i komparatora CMP (Slika 27).

Signal maskirajućeg prekida **printr** ima vrednost 1 ukoliko signali **imrprintr**, **acc** i **PSWI** imaju vrednosti 1. Signal **imrprintr** ima vrednost 1 ukoliko barem jedan od signala $\mathbf{irm_1}$ do $\mathbf{irm_3}$ ima vrednost 1. Ovo će se desiti ukoliko se barem u jednom od flip-flopova $PRINTR_1$ do PRINTR3 (Slika 26) nalazi vrednost 1 i ukoliko je u odgovarajućem razredu IMR_1 do IMR_3 registra maske $IMR_{15...0}$ takođe vrednost 1. Signal **acc** na izlazu komparatora CMP ima vrednost 1 ukoliko je prioritet pristiglog spoljašnjeg maskirajućeg zahteva za prekid najvišeg prioriteta koji nije maskiran viši od prioriteta tekućeg programa. Prioritet pristiglog spoljašnjeg maskirajućeg zahteva za prekid najvišeg prioriteta koji nije maskiran određen je signalima $\mathbf{prl_1}$ do $\mathbf{prl_0}$ na izlazu kodera prioriteta CD, dok je prioritet tekućeg programa određenog signalima $\mathbf{PSWL_1}$ do $\mathbf{PSWL_0}$ registra $PSW_{15...0}$ (slika 27). Signal **PSWI** dolazi sa izlaza odgovarajućeg razreda registra $PSW_{15...0}$ i njegovim vrednostima 1 i 0 se dozvoljavaju i maskiraju svi maskirajući prekidi, respektivno (Slika 23).

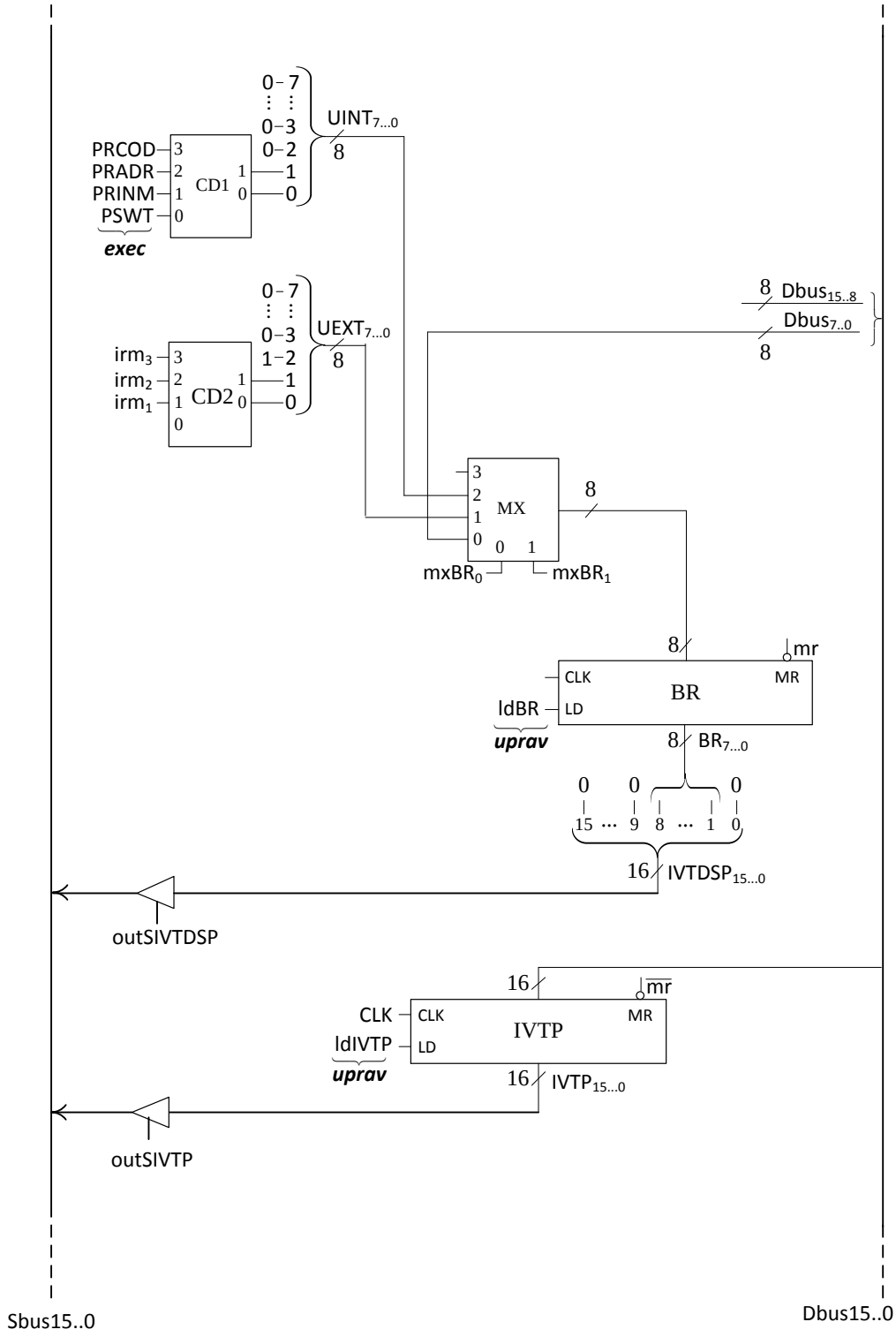


Slika 27 Blok *intr* (drugi deo)

Kombinacione i sekvencijalne mreže za formiranje pomeraja $IVTDSP_{15...0}$ za tabelu sa adresama prekidnih rutina se sastoje od kodera CD1 i CD2, i registra $BR_{7...0}$ sa multiplekserom MX (Slika 28).

Koder CD1 služi za formiranje broja ulaza u tabelu sa adresama prekidnih rutina za unutrašnje prekide, spoljašnji nemaskirajući prekid i unutrašnji prekid zbog zadatog režima rada prekid posle svake instrukcije. Ovi prekidi su određeni signalima na izlazima flip-flopova PSWT za unutrašnji prekid usled zadatog režima rada prekid posle svake instrukcije, PRINM za spoljašnji nemaskirajući prekid i PRCOD za unutrašnji prekid zbog čitanja instrukcije sa nepostojećim kodom operacije. Signali sa izlaza flip-flopova PSWT, PRINM, PRADR i PRCOD dovedeni su na ulaze 0 do 3 kodera prioriteta CD1 po rastućim prioritetima Na izlazu kodera dobija se broj zahteva za prekid najvišeg prioriteta binarno kodiran sa dva bita koji predstavljaju dva najniža bita broja ulaza. Bitovi 2 do 7 broja ulaza imaju vrednost 0 i sa dva bita na izlazu kodera CD1 formiraju 8-mo bitnu vrednost broja ulaza $UINT_{7...0}$. Ovim se dobijaju brojevi ulaza od 0 do 3.

Koder CD2 služi za formiranje broja ulaza u tabelu sa adresama prekidnih rutina za spoljašnje maskirajuće prekide $intr_{\mathbf{1}}$ do i $intr_{\mathbf{3}}$. Signali spoljašnjih maskirajućih prekida $intr_{\mathbf{1}}$ do $intr_{\mathbf{3}}$ posle eventualnog maskiranja razredima IMR_1 do IMR_3 registra maske $IMR_{15...0}$ pojavljuju se kao signali $\mathbf{irm_1}$ do $\mathbf{irm_3}$ (Slika 27). Ovi signali se vode na ulaze 1 do 3 kodera prioriteta CD2 po rastućim prioritetima. Na izlazu kodera dobija se broj zahteva za prekid najvišeg prioriteta binarno kodiran sa tri bita koji predstavljaju dva najniža bita broja ulaza. Bit 2, koji ima vrednost 1 i bitovi 3 do 7, koji imaju vrednost 0, sa dva bita na izlazu kodera CD2 formiraju 8-mo bitnu vrednost broja ulaza $UEXT_{7...0}$. Ovim se dobijaju brojevi ulaza od 5 do 7.



Slika 28 Blok *intr* (treći deo)

Registar $BR_{7...0}$ sa multiplekserom MX služi za selekciju i pamćenje broja ulaza u tabelu sa adresama prekidnih rutina (Slika 28). Upis sadržaja sa izlaza multipleksera MX u registar $BR_{7...0}$ se obavlja vrednošću 1 signala **ldBR**. Sadržaj registra $BR_{7...0}$ se u okviru opsluživanja zahteva za prekid koristi za formiranje pomeraja $IVTDSP_{15...0}$ za ulazak u tabelu sa adresama prekidnoh rutina. Kako adresa prekidne rutine zauzima dve susedne memorijske lokacije, to se pomeraj $IVTDSP_{15...0}$ dobija pomeranjem sadržaja registra $BR_{7...0}$ za jedno mesto ulevo tj množi se sa 2.

Multiplekser MX je 8-mo razredni multiplekser sa 4 ulaza. Na ulaze 0 do 2 multipleksera se vode sadržaji interne magistrale $Dbus_{15..0}$ i to nižih 8 bitova, $UEXT_{7...0}$ i $UINT_{7...0}$, koji predstavljaju brojeve ulaza u tabelu sa adresama prekidnih rutina za instrukciju prekida INT, za spoljašnje maskirajuće prekide, i za za unutrašnje prekide i spoljašnji nemaskirajući prekid, respektivno. Selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 10 upravljačkih signala **$mxBR_1$** i **$mxBR_0$**. Sadržaj interne magistrale $Dbus_{15..0}$ i to njenih nižih 8 bitova se propušta kroz multiplekser za instrukciju prekida INT vrednošću 00 upravljačkih signala **$mxBR_1$** i **$mxBR_0$**, $UEXT_{7...0}$ za spoljašnje maskirajuće prekide vrednošću 01 upravljačkih signala **$mxBR_1$** i **$mxBR_0$** i $UINT_{7...0}$ za unutrašnje prekide i spoljašnji nemaskirajući prekid vrednošću 10 upravljačkih signala **$mxBR_1$** i **$mxBR_0$**. Sadržaj sa izlaza multipleksera MX se vodi na ulaze registra $BR_{7...0}$.

Vrednost koja se dobija na izlazu registar BR, tj. pomeraj za tabelu sa adresama prekidnih rutina $IVTDSP_{15...0}$, se aktivnom vrednošću signala **outSIVTDSP** propušta kroz bafer sa tri stanja na internu magistralu **$Sbus_{15...0}$**.

Registar $IVTP_{15...0}$ je ukazivač na tabelu sa adresama prekidnih rutina i sadrži početnu adresu tabele. Upis sadržaja sa interne magistrale $Dbus_{15...0}$ u registar $IVTP_{15...0}$ se obavlja vrednošću 1 signala **ldIVTP**. Upis u registar $IVTP_{15...0}$ se realizuje samo kod izvršavanja instrukcije STIVTP, koja služi za upis sadržaja akumulatora $A_{15...0}$ u registar procesora $IVTP_{15...0}$. Aktivnom vrednošću signala **outSIVTP** sadržaj registra $IVTP_{15...0}$ se propušta kroz bafer sa tri stanja na internu magistralu **$Sbus_{15...0}$**.

5.2 UPRAVLJAČKA JEDINICA

U ovom odeljku se daju dijagram toka izvršavanja instrukcija, algoritam generisanja upravljačkih signala i struktura upravljačke jedinice ***uprav***.

5.2.1 Dijagram toka izvršavanja instrukcija

Dijagram toka izvršavanja instrukcija je predstavljen operacionim i uslovnim blokovima (Slika 29). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

U početnom koraku dijagram toka operacija vrši se provera da li je procesor zaustavljen ili ne. Ako je procesor zaustavljen ostaje se u početnom koraku i čeka startovanje procesora. Ako procesor nije zaustavljen prelazi se na korake u kojima se realizuje faza čitanje instrukcije.

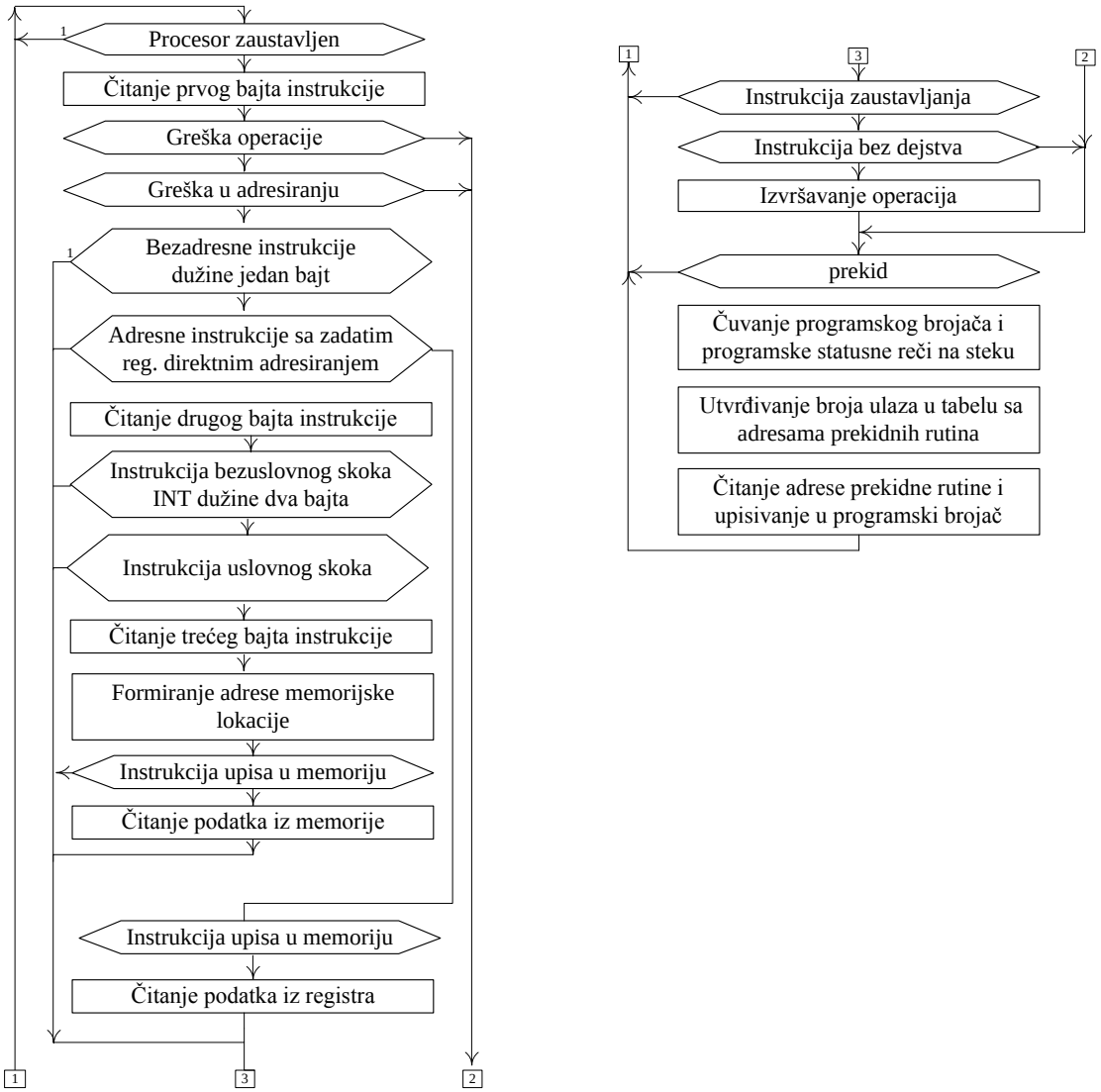
U okviru faze čitanje instrukcije najpre se čita prvi bajt instrukcije u kome se nalazi kod operacije i vrši provera da li je očitana instrukcija sa nepostojećim kodom operacije ili je došlo do greške u adresiranju. Greška adresiranja postoji samo ukoliko se radi o instrukciji upisa za koju je zadato neposredno adresiranja. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu opsluživanje prekida.

U nastavku faze čitanje instrukcije proverava se da li vrednost koda operacije odgovara vrednostima bezadresnih instrukcija čija je dužina jedan bajt ili odgovara adresnim instrukcijama za koje je zadato registarsko direktno adresiranje koje su takođe dužine jedan bajt. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu izvršavanje operacije.

U nastavku faze čitanje instrukcije čita se drugi bajt instrukcije i vrši provera da li je očitana instrukcija sa nepostojećim kodom operacije. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu opsluživanje prekida.

U nastavku faze čitanje instrukcije proverava se da li vrednost koda operacije odgovara vrednostima instrukcija uslovnog skoka čija je dužina dva bajta. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu izvršavanje operacije.

U nastavku faze čitanje instrukcije čita se treći bajt instrukcije i prelazi na fazu formiranje adrese operanda. Ovaj slučaj se dešava kod instrukcija bezuslovnog skoka i kod adresnih instrukcija kod kojih je zahtevano memorijsko direktno, memorijsko indirektno ili neposredno adresiranje.



Slika 29 Dijagram toka izvršavanja instrukcija

U okviru faze formiranje adrese operanda se postupa različito u zavisnosti od toga da li se radi o instrukcijama upisa ili čitanja operanda, a u slučaju da se radi o instrukcijama čitanja operanda i od toga da li se operand nalazi u instrukciji, nekom od registara opšte namene ili memorijskoj lokaciji. U slučaju neposrednog adresiranja operand se čita neposredno iz instrukcije i prelazi na fazu izvršavanje operacije. Slučaj sa neposrednim adresiranjem u operaciji upisa ne može da se javi u fazi formiranje adrese operanda, jer se provera na ovaj slučaj vrši još u fazi čitanje instrukcije i ukoliko se otkrije detektuje se greška adresiranja i prelazi na fazu opsluživanje prekida. U slučaju direktnog registarskog adresiranja operand se čita iz registra opšte namene i prelazi na fazu izvršavanje operacije ukoliko se ne radi o instrukciji upisa i odmah se prelazi na fazu izvršavanje operacije ukoliko se radi o instrukciji upisa. U svim ostalim slučajevima se radi o nekom od memorijskih adresiranja, pa se saglasno specificiranom načinu adresiranja formira adresa memorijske lokacije. Operand se čita iz memorijske lokacije i prelazi na fazu izvršavanje operacije ukoliko se ne radi o instrukciji upisa i odmah se prelazi na fazu izvršavanje operacije ukoliko se radi o instrukciji upisa.

U okviru faze izvršavanje operacija se postupa različito u zavisnosti od vrednosti koda operacije. U slučaju instrukcije zaustavljanja, procesor se zaustavlja i prelazi na početni

korak u kome se i ostaje jer je procesor zaustavljen. U slučaju instrukcije bez dejstva odmah se prelazi na fazu opsluživanje prekida. U slučaju preostalih instrukcija prelazi se na izvršavanje odgovarajuće operacije saglasno vrednosti koda operacije i prelazi na fazu opsluživanje prekida. Instrukcijama prenosa se ostvaruju prenosi iz različitih izvorišta u akumulatore procesora i obrnuto. Aritmetičkim instrukcijama se realizuju aritmetičke operacije nad sadržajima akumulatora i specificiranog operanda, a rezulta smešta u akumulator. Logičkim instrukcijama se realizuju logičke operacije nad sadržajima akumulatora i specificiranog operanda, a rezulta smešta u akumulator. Instrukcijama pomeranja i rotiranja se vrše pomeranja ulevo i udesno sadržaja akumulatora, a rezultat smešta u akumulator. Instrukcijama skoka se realizuju bezuslovni i uslovni skokovi u programu, skok na potprogram, povratak iz potprograma i programski generiše prekid. Instrukcijama postavljanja indikatora u PSW se zadaju ili zabranjuju režim rada reakcije na maskirajuće prekide i režim rada sa prekidom posle svake instrukcije.

U fazi opsluživanje prekida sa najpre na stek stavljaju programski brojač i programska statusna reč. Posle toga se po opadajućim prioritetima utvrđuje prekid najvišeg prioriteta za koji je zahtev generisan i saglasno tome formira broj ulaza u tabelu sa adresam prekidnih rutina. Na kraju se, sabiranjem broja ulaza pretvorenog u pomeraj i registra koji ukazuje na početnu adresu tabele sa adresama prekidnih rutina, dobija adresa sa koje se čita adresa prekidne rutine, upisuje u programski brojač i vraća u početni korak. Time se kreće sa fazom čitanje instrukcije prve instrukcije prekidne rutine.

5.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 29) i dat je u obliku dijagrama toka mikrooperacija, dijagrama toka upravljačkih signala (Slike 30 do 56) i sekvence upravljačkih signala (tabela 1).

Dijagram toka mikrooperacija i dijagram toka upravljačkih signala su dati istovremeno i predstavljeni su operacionim i uslovnim blokovima. U operacionim blokovima dijagrama toka mikrooperacija se nalaze mikrooperacije i uslovi pod kojima se one izvršavaju, dok se u operacionim blokovima dijagrama toka upravljačkih signala nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima dijagrama toka mikrooperacija i dijagrama toka upravljačkih signala se nalaze signali logičkih uslova.

U sekvenci upravljačkih signala po koracima se koriste iskazi za signale i skokove. Iskazi za signale su oblika
signali.

Ovaj iskaz sadrži spisak upravljačkih signala blokova operacione jedinice ***oper*** i određuje koji se signali bezuslovno generišu. Iskazi za skokove su oblika

br $step_A$,

br (*if* **uslov** *then* $step_A$) i

br (*case* (**$uslov_1$**, ..., **$uslov_n$**) *then* (**$uslov_1$**, $step_{A1}$), ..., (**$uslov_n$**, $step_{An}$).

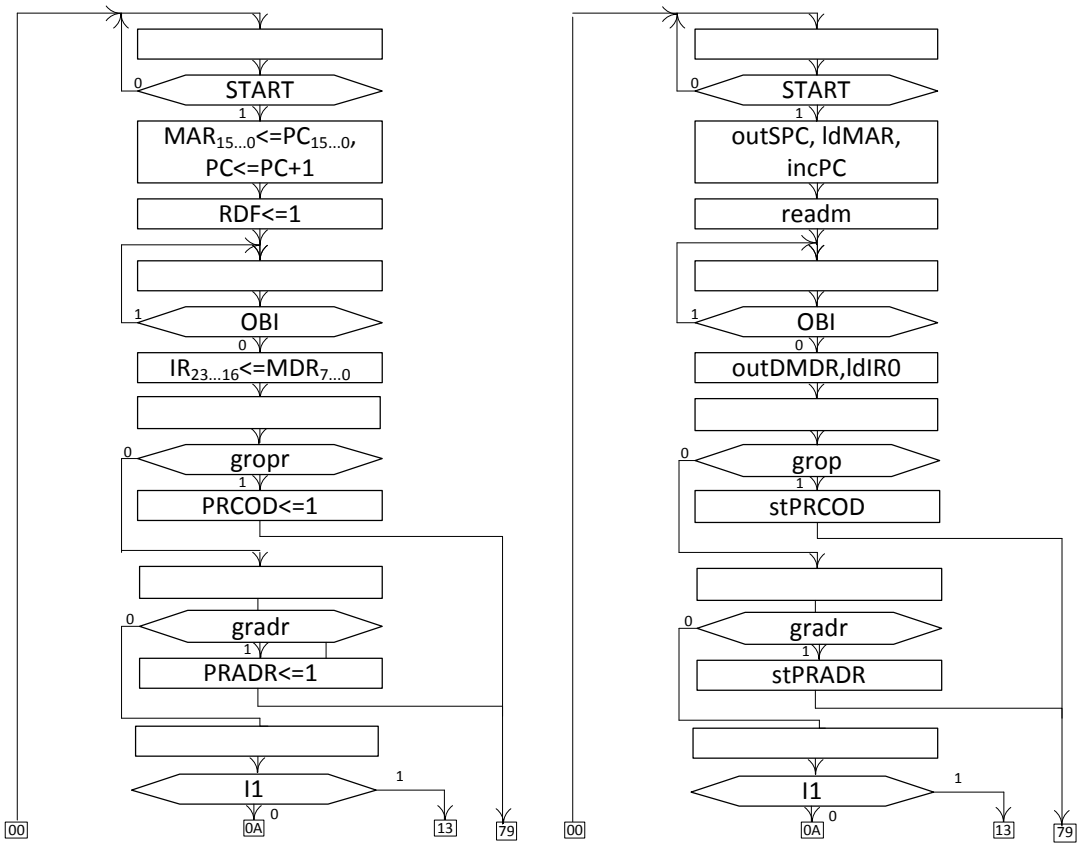
Prvi iskaz sadrži korak $step_A$ na koji treba bezuslovno preći i u daljem tekstu se referiše kao bezuslovni skok. Drugi iskaz sadrži signal **uslov** i korak $step_A$ i određuje korak $step_A$ na koji treba preći ukoliko signal **uslov** ima vrednost 1 i u daljem tekstu se referiše kao uslovni skok. Treći iskaz sadrži signale **$uslov_1$**, ..., **$uslov_n$** i korake $step_{A1}$, ..., $step_{An}$ i određuje na koji od koraka $step_{A1}$, ..., $step_{An}$ treba preći u zavisnosti od toga koji od signala **$uslov_1$**, ..., **$uslov_n$** ima vrednost 1 i u daljem tekstu se referiše kao višestruki uslovni skok.

Objašnjenja vezana za generisanje upravljačkih signala su data zajednički za dijagram toka upravljačkih signala i sekvencu upravljačkih signala i to u okviru sekvence upravljačkih signala.

Tabela 1 Sekvenca upravljačkih signala

! Sekvenca upravljačkih signala ima četiri celine koje odgovaraju fazama čitanje instrukcije, formiranje adrese i čitanje operanda, izvršavanje operacije i opsluživanje prekida. Faza čitanje instrukcije se realizuje u koracima $step_{00}$ do $step_{12}$ koji su zajednički za sve instrukcije. Faza formiranje adrese i čitanje operanda se realizuje u koracima $step_{13}$ do $step_{28}$ pri čemu postoje posebni koraci za svaki način adresiranja operanda. Faza izvršavanje operacije se realizuje u koracima $step_{29}$ do $step_{6C}$, pri čemu postoje posebni koraci za svaku operaciju. Faza opsluživanje prekida se realizuje u koracima $step_{6D}$ do $step_{91}$. !

! Čitanje instrukcije !



Slika 1 Čitanje instrukcije (prvi deo)

! U koraku $step_{00}$ se proverava vrednost signala **START** bloka *exec* koji vrednostima 0 i 1 ukazuje da li je processor neaktivan ili aktivan, respektivno. Ukoliko je procesor neaktivan ostaje se u koraku $step_{00}$, dok se u suprotnom slučaju prelazi na korak $step_{01}$. !

$step_{00}$ *br* (*if* $\overline{\textbf{START}}$ *then* $step_{00}$);

! U koracima $step_{01}$ do $step_{04}$ se čita prvi bajt instrukcije i smešta u razrede $IR_{23...16}$ prihvatnog registra instrukcije $IR_{23...0}$. Vrednošću 1 signala **outSPC** bloka *fetch* se sadržaj registra $PC_{15..0}$ propušta na internu magisrtralu $Sbus_{15..0}$ i vrednošću 1 signala **ldMAR** bloka *bus* upisuje u adresni registar $MAR_{15..0}$ bloka *bus*. Istovremeno se, vrednošću 1 signala **incPC** bloka *fetch* sadržaj registra $PC_{15..0}$ inkrementira.
U koraku $step_{02}$ se vrednošću 1 signala **readm** aktivira signal **RDF**. Aktivnom vrednošću signala **readm** aktivira se i signal **OBI** bloka *bus* kojim se dalje aktivira signal **BR0** čime procesor zahteva magistralu od paralelnog arbitratora za realizaciju ciklusa čitanja jedne reči.
U koraku $step_{03}$ se realizuje čitanje jednog bajta i upisivanje u registar podatka $MDR_{7..0}$ bloka *bus*. Vrednošću 1 signala **ERDA** se sadržaj registra $MAR_{15...0}$ pušta na adresne linije $ABUS_{15...0}$ magistrale ***BUS***. Pored toga se vrednošću 1 signala **ERDC** generiše vrednost 0 signala na upravljačkoj liniji $\overline{\textbf{RDBUS}}$ magistrale ***BUS***, čime se u memoriji ***MEM*** startuje operacija čitanja. U koraku $step_{03}$ se ostaje onoliko perioda signala takta koliko je neophodno da se čitanje realizuje. Sve vreme dok je processor u koraku $step_{03}$ adresa je prisutna na linijama $ABUS_{15...0}$ i na liniji $\overline{\textbf{RDBUS}}$ je vrednost 0. Po završenom čitanju memorija ***MEM*** pušta pročitani sadržaj na linije podataka $DBUS_{7...0}$ magistrale ***BUS***. Radi indikacije da je sadržaj na linijama $DBUS_{7...0}$ važeći memorija generiše vrednost 0 signala na upravljačkoj liniji $\overline{\textbf{FCBUS}}$ magistrale ***BUS***. Ovde se proverava vrednost signala **OBI** bloka *bus* koji vrednostima 0 i 1 ukazuje da je magistrala u posedu kontroleru sa direktnim pristupom memoriji ili da je magistrala u posedu procesora, respektivno. Ukoliko je magistralu u posedu kontroleru sa direktnim pristupom memoriji, to znači da je procesor završio ciklus na magistrali, da se traženi podatak nalazi u registru podatka $MDR_{7..0}$ bloka *bus* i prelazi se na korak $step_{04}$. U suprotnom procesor još nije dobio magistralu jer je ona u posedu kontroleru sa direktnim pristupom memoriji ili je procesor dobio magistralu ali još nije završio ciklus čitanja i ostaje se u koraku $step_{03}$. Kada je čitanje završeno, tj. signal **OBI** postane neaktivan, sadržaj sa linija $DBUS_{7...0}$ upisuje u registar podatka $MDR_{7..0}$ i prelazi na korak $step_{04}$. U koraku $step_{04}$ se vrednošću 1 signala **outDMDR** i **ldIR0** bloka *fetch* pročitani bajt prebacuje iz registra $MDR_{7..0}$ bloka *bus* u prihvatni registar instrukcije i to u razrede $IR_{23...16}$ bloka *bus*. !

$step_{01}$ **outSPC, ldMAR**, **incPC**;
$step_{02}$ **readm**;
$step_{03}$ *br* (*if* **OBI** *then* $step_{03}$);
$step_{04}$ **outDMDR, ldIR0**;

! U koraku $step_{05}$ se proverava vrednost signala **gropr** bloka *fetch* da bi se utvrdilo da li prvi bajt instrukcije sadrži korektnu vrednost koda operacije. U zavisnosti od toga da li signal **gropr** ima vrednost 1 ili 0, prelazi se na korak $step_{06}$ ili $step_{07}$, respektivno. Ukoliko prvi bajt instrukcije sadrži nekorektnu vrednost koda operacije, u koraku $step_{06}$ se vrednošću 1 signala **stPRCOD** bloka *intr* u flip-flop **PRCOD** upisuje vrednost 1 i bezuslovno prelazi na korak $step_{7E}$ radi utvrđivanja adrese prekidne rutine. !

$step_{05}$ *br* (*if* $\overline{\textbf{gropr}}$ *then* $step_{07}$);
$step_{06}$ **stPRCOD**,
br $step_{6D}$

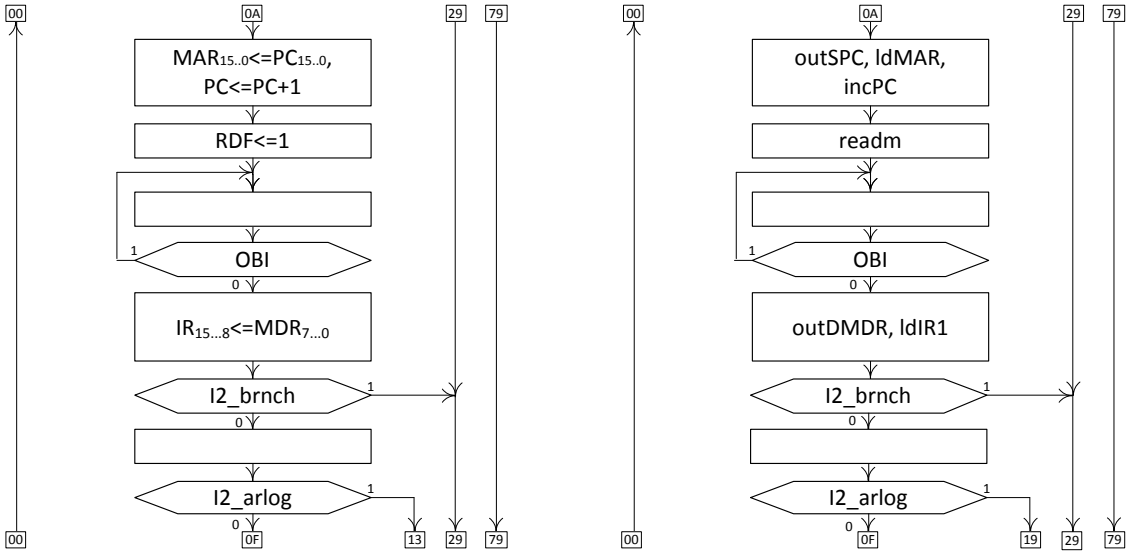
! U koraku $step_{07}$ se proverava vrednost signala **gradr** bloka *fetch* da bi se utvrdilo da li prvi bajt instrukcije sadrži korektnu vrednost adrese operacije. U zavisnosti od toga da li signal **gradr** ima vrednost 1 ili 0, prelazi se na korak $step_{08}$ ili $step_{09}$, respektivno. Ukoliko prvi bajt instrukcije sadrži nekorektnu vrednost adresiranja, u koraku $step_{08}$ se vrednošću 1 signala **stPRADR** bloka

intr u flip-flop **PRADR** upisuje vrednost 1 i bezuslovno prelazi na korak $step_{65}$ radi utvrđivanja adrese prekidne rutine. !

$step_{07}$ *br* (*if* $\overline{gradr}$ *then* $step_{08}$);
$step_{08}$ **stPRADR**,
br $step_{6D}$

! U koraku $step_{09}$ se proverava vrednost signala **l1** bloka *fetch* da bi se utvrdilo da li je dužina instrukcije jedan bajt ili više bajtova. U zavisnosti od toga da li signal **l1** ima vrednost 1 ili 0, prelazi se na korak $step_{23}$ i fazu izvršavanje operacije ili $step_{08}$ i produžava sa čitanjem bajtova instrukcije. !

$step_{09}$ *br* (*if* **l1** *then* $step_{13}$);



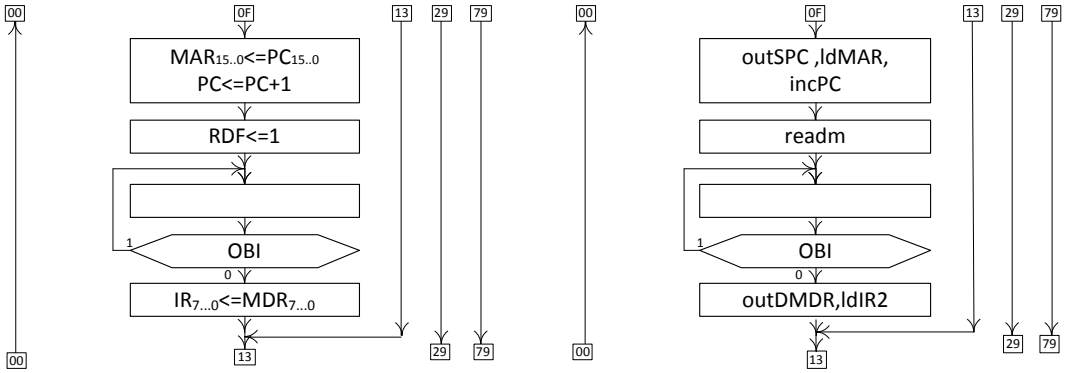
Slika 31 Čitanje instrukcije (drugi deo)

! U koracima $step_{0A}$ do $step_{0C}$ se čita drugi bajt instrukcije i smešta u razrede $IR_{15...8}$ prihvatnog registra instrukcije $IR_{23...0}$. Koraci $step_{0A}$ do $step_{0C}$ u kojima se čita drugi bajt instrukcije su isti kao koraci $step_{01}$ do $step_{03}$ u kojima se čita prvi bajt instrukcije. U koraku $step_{0B}$ se signalom **ldIR1** bloka *fetch* pročitani bajt prebacuje iz registra $MDR_{7..0}$ bloka *bus* u prihvatni registar instrukcije i to u razrede $IR_{15...8}$ bloka *bus*.

$step_{0A}$ **outSPC, ldMAR**, **incPC**;
$step_{0B}$ **readm**;
$step_{0C}$ *br* (*if* **OBI** *then* $step_{0C}$);
$step_{0D}$ **outDMDR, ldIR1**;

! U koracima $step_{0D}$ i $step_{0E}$ se proverava vrednost signala **l2_brnch** i **l2_arlog** bloka *fetch* da bi se utvrdilo da li je dužina instrukcije dva bajta ili više bajtova. Iz koraka $step_{0D}$ se u zavisnosti od toga da li signal **l2_brnch** ima vrednost 1 ili 0, prelazi na korak $step_{29}$ i fazu izvršavanje operacije ili $step_{0E}$ i proveru vrednosti signala **l2_arlog**. Iz koraka $step_{0E}$ se u zavisnosti od toga da li signal **l2_arlog** ima vrednost 1 ili 0, prelazi na korak $step_{13}$ i fazu formiranje adrese i čitanje operanda ili $step_{0E}$ i produžava sa čitanjem bajtova instrukcije.!

$step_{0D}$ *br* (*if* **l2_brnch** *then* $step_{29}$);
$step_{0E}$ *br* (*if* **l2_arlog** *then* $step_{13}$);



Slika 32 Čitanje instrukcije (treći deo)

! U koracima $step_{0F}$ do $step_{11}$ se čita treći bajt instrukcije i smešta u razrede $IR_{7...0}$ prihvatnog registra instrukcije $IR_{23...0}$. Koraci $step_{0F}$ do $step_{11}$ u kojima se čita treći bajt instrukcije su isti kao koraci $step_{01}$ do $step_{03}$ u kojima se čita prvi bajt instrukcije.!

$step_{0F}$ **outSPC, ldMAR**, **incPC**;

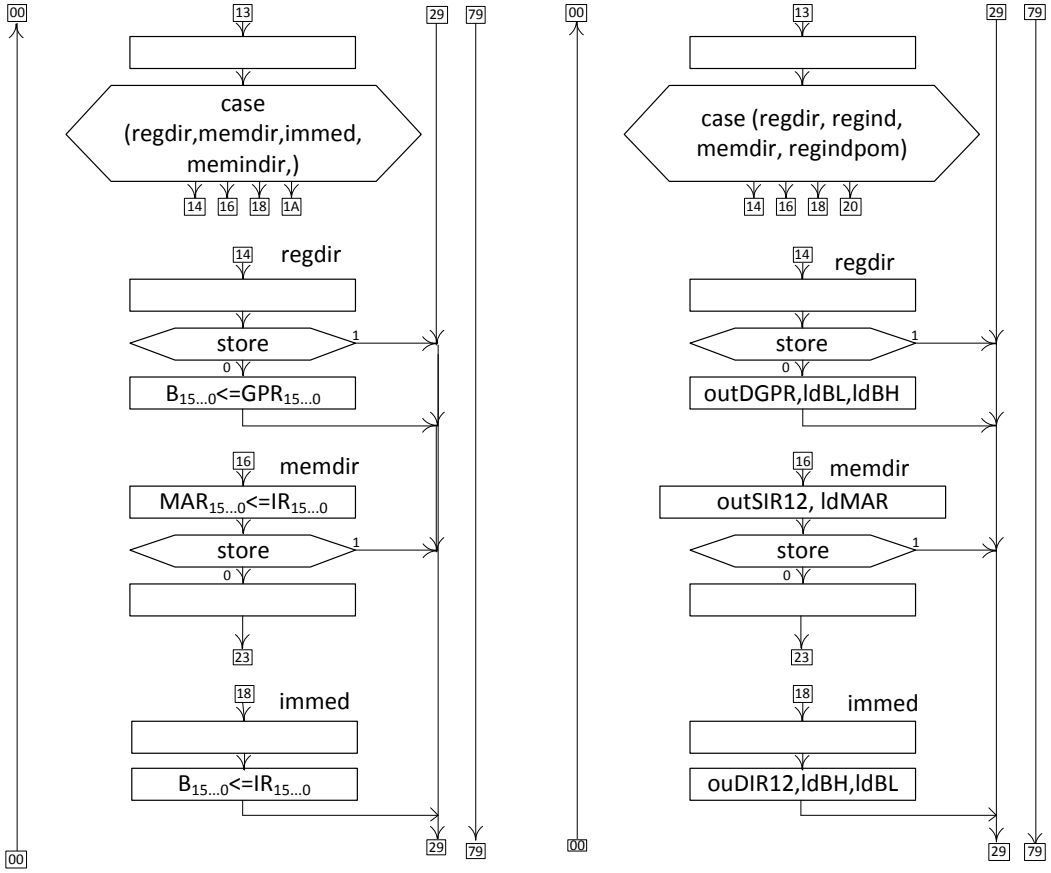
$step_{10}$ **readm**;

$step_{11}$ *br* (*if* **OBI** *then* $step_{11}$);

! U koraku $step_{12}$ se vrednošću 1 signala **outDMDR** i **ldIR2** bloka *fetch* pročitani bajt prebacuje iz registra $MDR_{7..0}$ bloka *bus* u prihvatni registar instrukcije i to u razrede $IR_{7...0}$ bloka *bus*. !

$step_{12}$ **outDMDR , ldIR2**;

! Formiranje adrese i čitanje operanda !



Slika 33 Formiranje adrese i čitanje operanda (prvi deo)

! U korak $step_{13}$ se dolazi iz koraka $step_{09}$, $step_{0E}$ i $step_{12}$ ukoliko se radi o instrukcijama dužine jedan, dva, i tri bajta koje zahtevaju da se do operanda dođe saglasno specificiranom načinu adresiranja. Za sve instrukcije, sem instrukcije ST, operand se smešta u registar $B_{15...0}$, koji se sastoji iz dva 8-bitna registra $BL_{7...0}$ ili $BH_{7...0}$. Operand može da bude u nekom od registara opšte namene ili u memorijskoj lokaciji. U slučaju adresiranja kod kojih se operand nalazi u nekom od registara opšte namene, ova faza se svodi na prebacivanje operanda u registar $B_{15...0}$. U slučaju adresiranja kod kojih se operand nalazi u memoriji, ova faza se sastoji od koraka u kojima se prvo formira adresa operanda u memoriji i zatim čita operand. Izuzetak je instrukcija ST kod koje se operand upisuje. U slučaju adresiranja kod koga se operand upisuje u registar opšte namene, odmah se prelazi na fazu izvršavanje operacije u kojoj se operand upisuje u registar opšte namene. U slučaju nekog od adresiranja kod kojih se operand upisuje u memoriju, u ovoj fazi se samo formira adresa operanda u registru $MAR_{15...0}$, pa se prelazi na fazu izvršavanje operacije u kojoj se operand upisuje u memoriju na formiranoj adresi. U koraku $step_{13}$ se realizuje višestruki uslovni skok na jedan od koraka $step_{14}$, $step_{16}$, $step_{18}$, $step_{20}$ u zavisnosti od toga koji od signala adresiranja **regdir**, **memdir**, **immed, memindir** bloka *addr* ima vrednost 1. !

$step_{13}$ *br* (*case* (**regdir**, **memdir**, **immed, memindir**) *then*
(**regdir**, $step_{14}$), (**memdir**, $step_{16}$), (**immed**, $step_{18}$), (**memindir**, $step_{20}$));

! Registarsko direktno adresiranje !
! U korak $step_{16}$ se dolazi iz $step_{13}$ ukoliko signal za registarsko direktno adresiranje **regdir** ima vrednost 1. Ukoliko signal **store** bloka *fetch* ima vrednost 1, što znači da se radi o instrukciji ST za koje nema čitanja operanda, prelazi se na korak $step_{29}$ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak $step_{19}$ u kome se vrednošću 1 signala **outDGPR**, **ldBL, ldBH** 16-to razredni sadržaj adresiranog registra opšte namene $GPR_{15...0}$ bloka *addr* upisuje u registar $B_{15...0}$ bloka *exec*. Nakon toga se prelazi na korak $step_{29}$ i fazu izvršavanje operacije. !

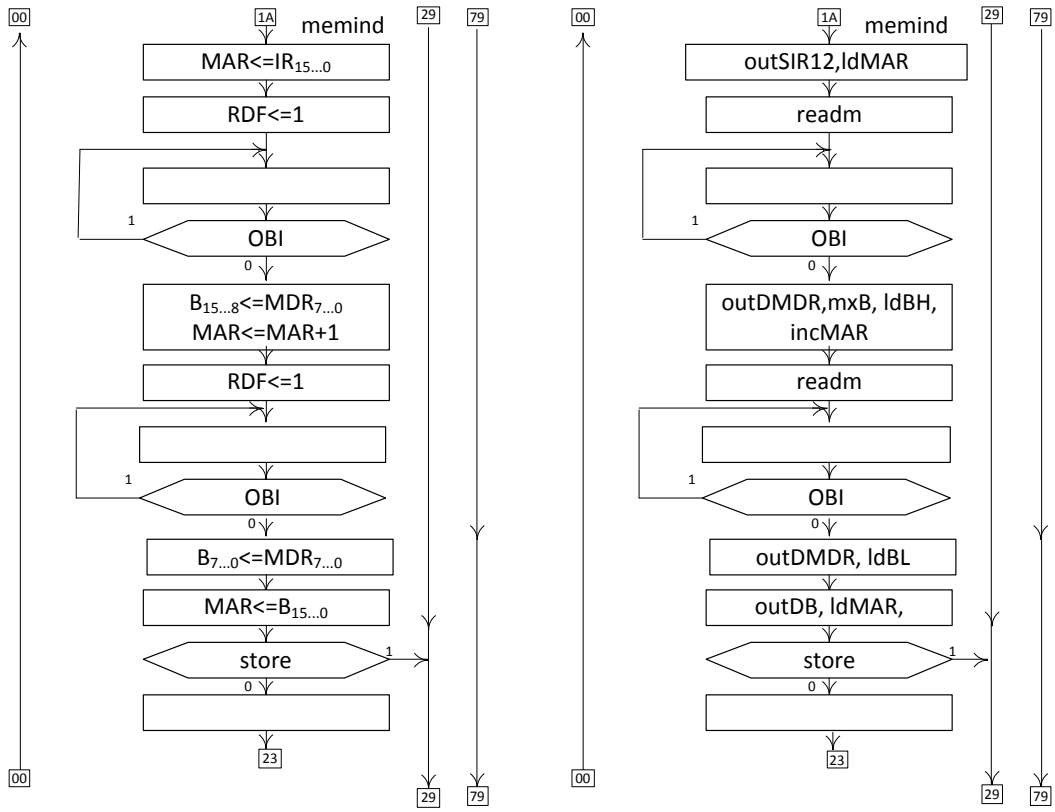
$step_{14}$ *br* (*if* **store** *then* $step_{29}$);
$step_{15}$ **outDGPR**, **ldBL, ldBH**,
br $step_{29}$;

! Memorijsko direktno adresiranje !
! U korak $step_{16}$ se dolazi iz $step_{13}$ ukoliko signal za memorijsko direktno adresiranje **memdir** ima vrednost 1. Vrednošću 1 signala **outSIR21** se sadržaj registra $IR_{15...8.}IR_{7...0}$ bloka *fetch* propušta na internu magistralu $Sbus_{15..0}$ i vrednošću 1 signala **ldMAR** upisuje u registar $MAR_{15...0}$. Time se u registru $MAR_{15...0}$ nalazi adresa operanda za slučaj memorijskog direktnog adresiranja. Ukoliko signal **store** bloka *addr* ima vrednost 1, što znači da se radi o instrukciji ST za koju nema čitanja operanda, prelazi se na korak $step_{29}$ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak $step_{17}$ iz koga se bezuslovno prelazi na korak $step_{29}$ i čitanje operanda. !

$step_{16}$ **outSIR21, ldMAR**,
$step_{17}$ *br* (*if* **store** *then* $step_{29}$);
br $step_{19}$;

! Neposredno adresiranje !
! U korak $step_{18}$ se dolazi iz $step_{13}$ ukoliko signal za neposredno adresiranje **immed** ima vrednost 1. U koraku $step_{18}$ se vrednostima 1 signala **outDIR21** i se sadržaj registra $IR_{15...8.}IR_{7...0}$ bloka *fetch* propušta na internu magistralu $Sbus_{15..0}$ i vrednošću 1 signala **ldBL, ldBH** upisuje u registar $B_{15...0}$. Bezuslovno se prelazi na korak $step_{29}$ i fazu izvršavanja operacije. !

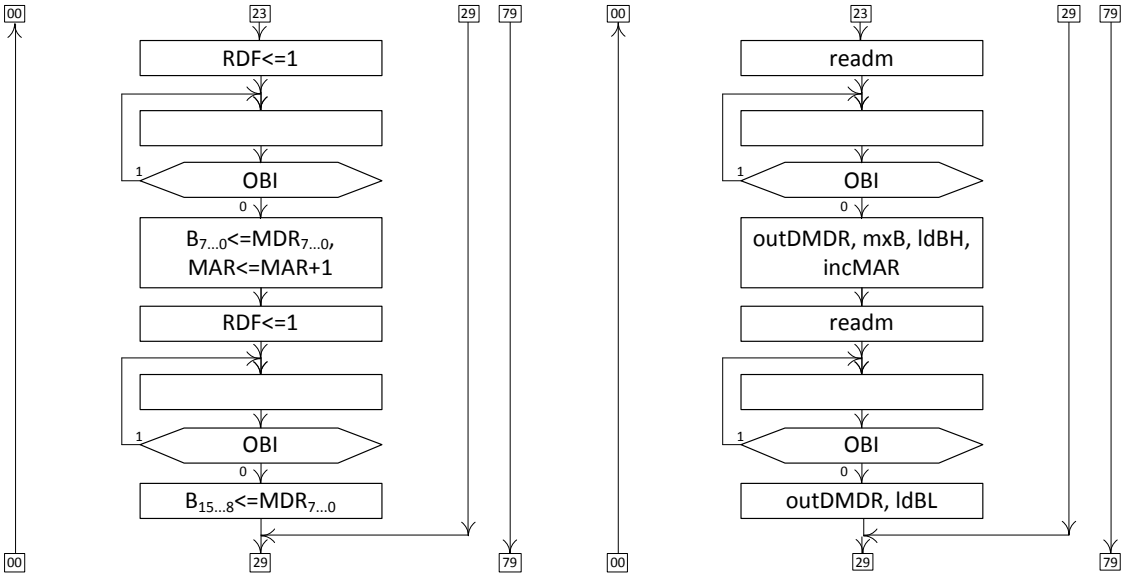
$step_{19}$ **outDIR21, ldBL, ldBH**;
br $step_{29}$;



! Memorijsko indirektno adresiranje !

! U korak $step_{1A}$ se dolazi iz $step_{13}$ ukoliko signal za memorijsko indirektno adresiranje **memind** ima vrednost 1. Vrednošću 1 signala **outSIR21** se sadržaj registra $IR_{15...0}$ bloka *fetch* na internu magistralu $Sbus_{15..0}$ i vrednošću 1 signala **ldMAR** upisuje u registar $MAR_{15...0}$. Time se u registru $MAR_{15...0}$ nalazi adresa memorijske lokacije počev od koje treba pročitati dva bajta koji predstavljaju viši i niži bajt adrese operanda za slučaj memorijskog indirektnog adresiranja. U koraku $step_{1D}$ se iz registra $MDR_{7...0}$ bloka *bus* prvi bajt vrednošću 1 signala **ldBH, mxB** upisuje u viši bajt registra $B_{15...7}$ bloka *bus*, a vrednošću 1 signala **incMAR** adresni registar $MAR_{15...0}$ bloka *bus* inkrementira na adresu sledećeg bajta. U koraku $step_{20}$ se iz registra $MDR_{7...0}$ bloka *bus* drugi bajt vrednošću 1 signala **ldBL** upisuje u niži bajt registra $B_{7...0}$. Na kraju se u koraku $step_{21}$ vrednostima 1 signala **outSB** sadržaj registra $B_{15...0}$ koji predstavlja adresu operanda propušta na internu magistralu $Sbus_{15..0}$ i vrednošću 1 signala **ldMAR** upisuje u registar $MAR_{15...0}$. Ukoliko signal **store** bloka *addr* ima vrednost 1, što znači da se radi o instrukciji ST za koju nema čitanja operanda, prelazi se na korak $step_{29}$ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak $step_{22}$ iz koga se bezuslovno prelazi na korak $step_{23}$ i čitanje operanda. !

$step_{1A}$ **outSIR21**, **ldMAR**;
$step_{1B}$ **readm**
$step_{1C}$ *br* (*if* **OBI** *then* $step_{1C}$);
$step_{1D}$ **outDMDR, ldDL**, **ldMAR**;
$step_{1E}$ **readm**,
$step_{1F}$ *br* (*if* **OBI** *then* $step_{1F}$);
$step_{20}$ **outDMDR, ldDH**
$step_{21}$ **outDB**, **ldMAR**,
$step_{22}$ *br* (*if* **store** *then* $step_{29}$);
br $step_{23}$;



Slika 34 Formiranje adrese i čitanje operanda (drugi deo)

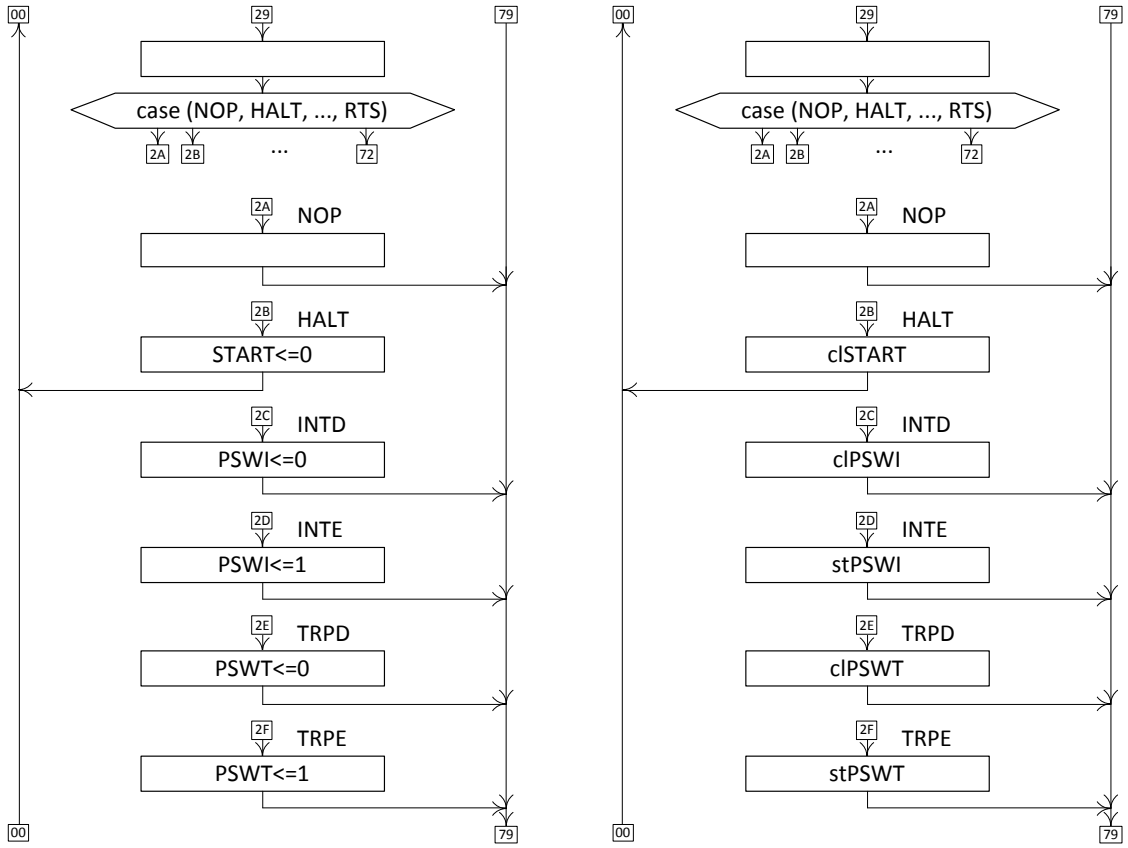
! Čitanje operanda !

! U korak $step_{23}$ se dolazi iz $step_{17}$ kod memorijskog direktnog adresiranja, iz $step_{12}$ kod memorijskog indirektnog adresiranja. U svim ovim situacijama adresa memorijske lokacije sa koje treba pročitati operand je sračunata u saglasnosti sa specificiranim načinom adresiranja i nalazi se u registru $MAR_{15...0}$ bloka *bus*. Za sve instrukcije za koje se čita operand iz memorije dužina operanda je dva bajta. Zbog toga se najpre u koracima $step_{23}$ i $step_{24}$ čita jedan bajt i to na isti način na koji se to radi u koracima $step_{02}$ i $step_{03}$ u kojima se čita prvi bajt instrukcije.

Potom se u koraku $step_{22}$ vrednošću 1 signala **outDMDR** bloka *bus* i **ldBH, mxB** bloka *exec* sadržaj registra $MDR_{7...0}$ upisuje u stariji bajt registra $B_{15...0}$ i vrednošću 1 signala **incMAR** inkrementira sadržaj registra $MAR_{15...0}$ da bi ukazivao na memorijsku lokaciju na kojoj se nalazi mlađi bajt 16-to razrednog operanda. Potom se u koracima $step_{26}$ i $step_{27}$ čita jedan bajt i to na isti način na koji se to radi u koracima $step_{02}$ i $step_{03}$ u kojima se čita prvi bajt instrukcije. Zatim se u koraku $step_{28}$ vrednošću 1 signala **outDMDR** bloka *bus* i **ldBL** bloka *bus* sadržaj registra $MDR_{7...0}$ upisuje u mlađi bajt registra $B_{15...0}$. Time se u registru $B_{15...0}$ nalazi 16-to bitni operand. Iz koraka $step_{28}$ se bezuslovno prelazi na korak $step_{29}$ i fazu izvršavanje operacije. !

$step_{23}$ **readm**;
$step_{24}$ *br* (*if* **OBI** *then* $step_{24}$);
$step_{25}$ **outDMDR**, **mxB**, **ldBH**, **incMAR**;
$step_{26}$ **readm**;
$step_{27}$ *br* (*if* **OBI** *then* $step_{25}$);
$step_{28}$ **outDMDR, ldBL**;

! Izvršavanje operacije !



Slika 35 Izvršavanje operacije (prvi deo)

! U korak $step_{29}$ se dolazi iz koraka $step_{07}$, $step_{0C}$, $step_{11}$, $step_{18}$, $step_{19}$, $step_{1A}$, $step_{1C}$, $step_{1F}$, $step_{25}$ radi izvršavanja operacije. U koraku $step_{29}$ se realizuje višestruki uslovni skok na jedan od koraka $step_{2A}$, $step_{2B}$, ..., $step_{64}$ u zavisnosti od toga koji od signala operacija **NOP**, **HALT**, ..., **RTS** ima vrednost 1. !

$step_{29}$ *br* (*case* (**NOP**, **HALT**, **INTD**, **INTE**, **TRPD**, **TRPE**,
LD, **ST**, **STIVTP**, **STSP**, **ADD**, **SUB**, **AND**,**OR**, **XOR**, **NOT**,
ASR, **LSR**, **ROR**, **RORC**, **ASL**, **LSL**, **ROL**, **ROLC**,
BEQL, BNEQ, BNEG, BNNG, BOVF, BNOVF, BCR, BNCR,
BGRT, BGRE, BLSS, BLEQ, BGRTU, BGREU, BLSSU, BLEQU,
JMP, **INT**, **JSR**, **RTI**, **RTS**)
then
(**NOP**, $step_{2A}$), (**HALT**, $step_{2B}$),
(**INTD**, $step_{2C}$), (**INTE**, $step_{2D}$), (**TRPD**, $step_{2E}$), (**TRPE**, $step_{2F}$),
(**LD**, $step_{30}$), (**ST**, $step_{31}$), (**STIVTP**, $step_{3A}$), (**STSP**, $step_{3B}$),
(**ADD**, $step_{3E}$), (**SUB**, $step_{40}$), (**AND**, $step_{42}$), (**OR**, $step_{44}$), (**XOR**, $step_{46}$), (**NOT**, $step_{48}$),
(**ASR**, $step_{4A}$), (**LSR**, $step_{4A}$), (**ROR**, $step_{4A}$), (**RORC**, $step_{4A}$),
(**ASL**, $step_{4C}$), (**LSL**, $step_{4C}$), (**ROL**, $step_{4C}$), (**ROLC**, $step_{4C}$),
(**BEQL**, $step_{4E}$), (**BNEQ**, $step_{4E}$), (**BNEG**, $step_{4E}$), (**BNNG**, $step_{4E}$),
(**BOVF**, $step_{4E}$), (**BNOVF**, $step_{4E}$), (**BCR**, $step_{4E}$), (**BNCR**, $step_{4E}$),
(**BGRT**, $step_{4E}$), (**BGRE**, $step_{4E}$), (**BLSS**, $step_{4E}$), (**BLEQ**, $step_{4E}$),
(**BGRTU**, $step_{4E}$), (**BGREU**, $step_{4E}$), (**BLSSU**, $step_{4E}$), (**BLEQU**, $step_{4E}$),
(**JMP**, $step_{51}$), (**INT**, $step_{52}$), (**JSR**, $step_{53}$), (**RTI**, $step_{5C}$), (**RTS**, $step_{64}$));

! NOP !
! U korak $step_{2A}$ se dolazi iz $step_{29}$ ukoliko signal operacije **NOP** ima vrednost 1. Pošto se radi o instrukciji bez dejstva, nema generisanja upravljačkih signala, već se bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{2A}$ *br* $step_{6D}$

! HALT !
! U korak $step_{2B}$ se dolazi iz $step_{29}$ ukoliko signal operacije **HALT** ima vrednost 1. Pošto se radi o instrukciji zaustavljanja procesora, generiše se vrednost 1 signala **clSTART** bloka *exec*, kojom se u flip-flop START upisuje vrednost 0, i bezuslovno prelazi na početni korak $step_{00}$. !

$step_{2B}$ **clSTART**,
br $step_{00}$;

! INTD !
! U korak $step_{2C}$ se dolazi iz $step_{29}$ ukoliko signal operacije **INTD** ima vrednost 1. Vrednošću 1 signala **clPSWI** se razred PSWI bloka *exec* postavlja na vrednost 0. Iz koraka $step_{2C}$ se bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{2C}$ **clPSWI**,
br $step_{6D}$

! INTE !
! U korak $step_{2D}$ se dolazi iz $step_{29}$ ukoliko signal operacije **INTE** ima vrednost 1. Vrednošću 1 signala **stPSWI** se razred PSWI bloka *exec* postavlja na vrednost 1. Iz koraka $step_{2D}$ se bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida.!

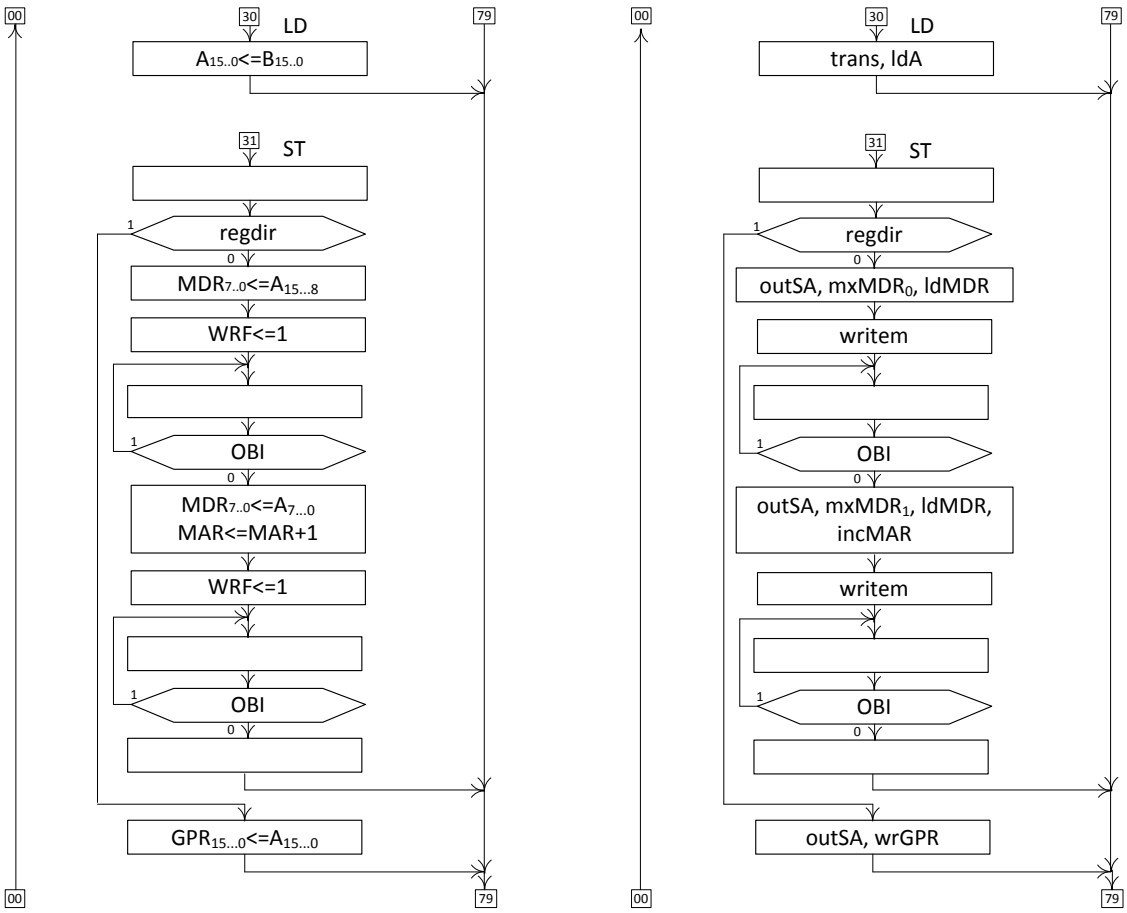
$step_{2D}$ **stPSWI**,
br $step_{6D}$;

! TRPD !
! U korak $step_{2E}$ se dolazi iz $step_{29}$ ukoliko signal operacije **TRPD** ima vrednost 1. Vrednošću 1 signala **clPSWT** se razred PSWT bloka *exec* postavlja na vrednost 0. Iz koraka $step_{2E}$ se bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{2E}$ **clPSWT**,
br $step_{6D}$

! TRPE !
! U korak $step_{2F}$ se dolazi iz $step_{29}$ ukoliko signal operacije **TRPE** ima vrednost 1. Vrednošću 1 signala **stPSWT** se razred PSWT bloka *exec* postavlja na vrednost 1. Iz koraka $step_{2F}$ se bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida.!

$step_{2F}$ **stPSWT**,
br $step_{6D}$



Slika 36 Izvršavanje operacije (drugi deo)

! LD !

! U korak $step_{30}$ se dolazi iz $step_{29}$ ukoliko signal operacije **LD** ima vrednost 1. U ovom koraku se vrednošću 1 signala **ldA** i **trans** bloka *exec* sadržaj registra $B_{15...0}$ i upisuje u registar $A_{15...0}$ i prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{30}$ **trans, ldA**,
br $step_{6D}$

! ST !

! U korak $step_{31}$ se dolazi iz $step_{29}$ ukoliko signal operacije **ST** ima vrednost 1. Iz ovog koraka se u zavisnosti od toga da li signal **regdir** bloka *fetch* ima vrednost 0 ili 1 prelazi na korak $step_{32}$ ili $step_{39}$, respektivno. !

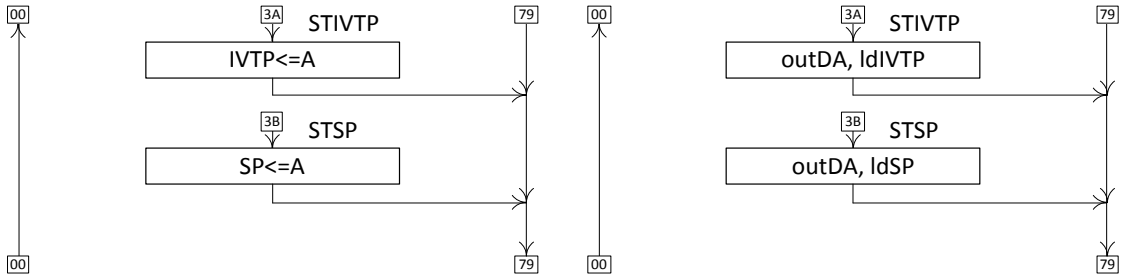
$step_{31}$ *br* (*if* **regdir** *then* $step_{39}$);

! U koracima $step_{32}$ do $step_{38}$ se sadržaj najpre višeg bajta a zatim i nižeg bajta registra $A_{15...0}$ bloka *exec* upisuje u dve susedne memorijske lokacije počev od memorijske lokacije čija je adresa formirana u fazi formiranje adrese i koja se nalazi i registru $MAR_{15...0}$ bloka *bus*. Stoga se, najpre, u koraku $step_{32}$ vrednostima 1 signala **outSA, mxMDR$_0$** i **ldMDR** bloka *bus* sadržaj registra $A_{15...8}$ propušta kroz multiplekser MX i upisuje u registar $MDR_{7...0}$. Upis se realizuje u koracima $step_{33}$ i $step_{34}$. Potom se u koraku $step_{35}$ vrednostima 1 signala **outSA, mxMDR$_1$** i **ldMDR** bloka *bus* sadržaj registra $A_{15...8}$ propušta kroz multiplekser MX i upisuje u registar $MDR_{7...0}$, a vrednošću 1 signala **incMAR** inkrementita vrednost koja se nalazi u registru $MAR_{15...0}$. Upis se realizuje u koracima $step_{36}$ i $step_{37}$ na isti načina kao što se to radi u koracima $step_{33}$ i $step_{34}$ za prethodni bajt. Po završetku upisa se prelazi na korak $step_{38}$, a iz ovog koraka se bezuslovno prelazi na korak $step_{7E}$ i fazu opsluživanje prekida. !

$step_{32}$ **outSA, mxMDR$_0$** i **ldMDR;**
$step_{33}$ **writem;**
$step_{34}$ *br* (*if* **OBI** *then* $step_{31}$);
$step_{35}$ **outSA, mxMDR$_1$** i **ldMDR, incMAR**;
$step_{36}$ **writem;**
$step_{37}$ *br* (*if* **OBI** *then* $step_{37}$);
$step_{38}$ *br* $step_{6D}$;

! U koraku $step_{39}$ se vrednostima 1 signala **outSA** i **wrGPR** bloka *addr* sadržaj registra $A_{15...0}$ bloka *exec* propušta kroz bafer tri stanja na internu magistralu $Sbus_{15..0}$ i upisuje u adresirani registar opšte namene $GPR_{15...0}$ i bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{39}$ **outSA, wrGPR**,
br $step_{6D}$



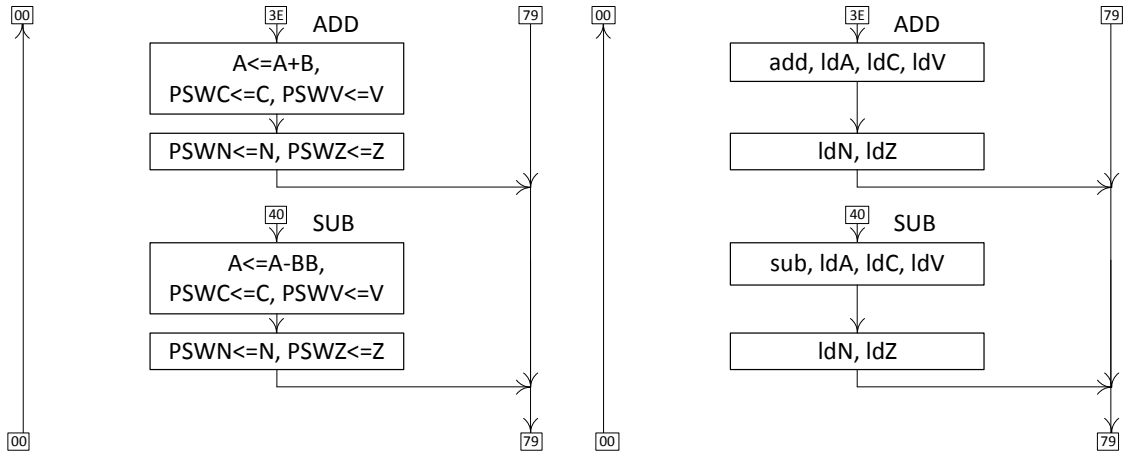
Slika 37 Izvršavanje operacije (treći deo)

! STIVTP !
! U korak $step_{3A}$ se dolazi iz $step_{29}$ ukoliko signal operacije STIVTP ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra $A_{15...0}$ bloka *exec* upisuje u registar $IVTP_{15...0}$ bloka *intr*. Stoga se vrednošću 1 signala **outDA, ldIVTP** sadržaj registra $A_{15...0}$ upisuje u registar $IVTP_{15...0}$. Na kraju se iz koraka $step_{3A}$ bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{3A}$ **outDA, ldIVTP**,
br $step_{6D}$

! STSP !
! U korak $step_{3B}$ se dolazi iz $step_{26}$ ukoliko signal operacije STSP ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra $A_{15...0}$ bloka *exec* upisuje u registar $SP_{15...0}$ bloka *addr*. Stoga se vrednošću 1 signala **outDA, ldSP** sadržaj registra $A_{15...0}$ upisuje u registar $SP_{15...0}$. Na kraju se iz koraka $step_{3B}$ bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{3B}$ **outDA, ldSP**,
br $step_{6D}$



Slika 38 Izvršavanje operacije (četvrti deo)

! ADD !

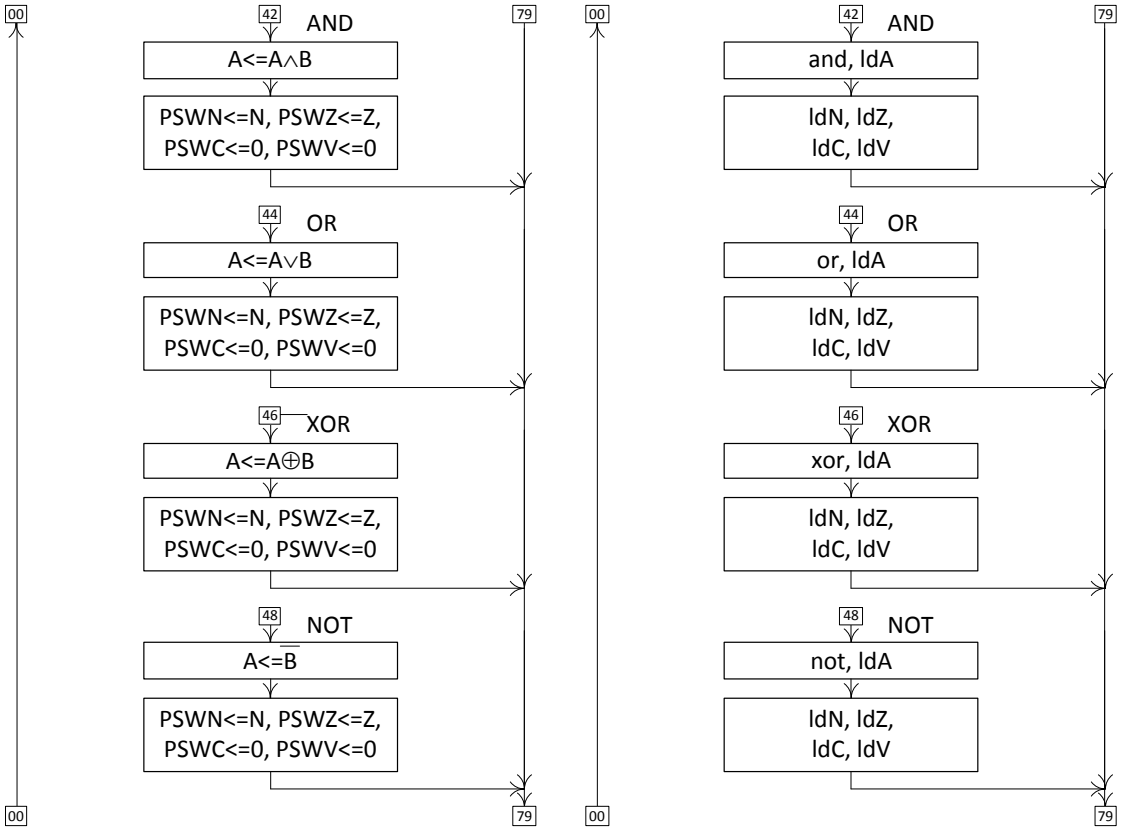
! U korak $step_{3E}$ se dolazi iz $step_{29}$ ukoliko signal operacije **ADD** ima vrednost 1. U fazi izvršavanja ove instrukcije se sabiraju sadržaji registra $A_{15...0}$ bloka *exec* koji se koristi kao akumulator i registra $B_{15...0}$ bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije i rezultat upisuje u registar $A_{15...0}$. Stoga se vrednošću 1 signala **add** bloka *exec* na izlazima $ALU_{15...0}$ aritmetičko logičke jedinice ALU formira suma sadržaja registara $A_{15...0}$ i $B_{15...0}$ koja se dalje vrednošću 1 signala **ldA** upisuje u registar $A_{15...0}$. Istovremeno se vrednostima 1 signala **ldC** i **ldV** bloka *exec* u razrede PSWC i PSWV programske statusne reči $PSW_{15...0}$ upisuju vrednosti signala **C** i **V** bloka *exec* formirane na osnovu dobijenog rezultata na izlazima $ALU_{15...0}$. U sledećem koraku se vrednostima 1 signala **ldN** i **ldZ** bloka *exec* u razrede PSWN i PSWZ programske statusne reči $PSW_{15...0}$ upisuju vrednosti signala **N** i **Z** bloka *exec* formirane na osnovu sadržaja upisanog u registar $A_{15...0}$ i prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{3E}$ **add**, **ldA**, **ldC**, **ldV**;
$step_{3F}$ **ldN**, **ldZ**,
br $step_{6D}$

! SUB !

! U korak $step_{40}$ se dolazi iz $step_{29}$ ukoliko signal operacije **SUB** ima vrednost 1. U fazi izvršavanja ove instrukcije se od sadržaja registra $A_{15...0}$ bloka *exec* koji se koristi kao akumulator oduzima sadržaj registra $B_{15...0}$ bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije i rezultat upisuje u registar $A_{15...0}$. Stoga se vrednošću 1 signala **sub** bloka *exec* na izlazima $ALU_{15...0}$ formira razlika sadržaja registara $A_{15...0}$ i $B_{15...0}$ koja se dalje vrednošću 1 signala **ldA** upisuje u registar $A_{15...0}$. Istovremeno se vrednostima 1 signala **ldC** i **ldV** bloka *exec* u razrede PSWC i PSWV programske statusne reči $PSW_{15...0}$ upisuju vrednosti signala **C** i **V** bloka *exec* formirane na osnovu dobijenog rezultata na izlazima $ALU_{15...0}$. U sledećem koraku se vrednostima 1 signala **ldN** i **ldZ** bloka *exec* u razrede PSWN i PSWZ programske statusne reči $PSW_{15...0}$ upisuju vrednosti signala **N** i **Z** bloka *exec* formirane na osnovu sadržaja upisanog u registar $A_{15..0}$ i prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{40}$ **sub**, **ldA**, **ldC**, **ldV**;
$step_{41}$ **ldN**, **ldZ**,
br $step_{6D}$



Slika 39 Izvršavanje operacije (peti deo)

! AND !

! U korak $step_{42}$ se dolazi iz $step_{29}$ ukoliko signal operacije **AND** ima vrednost 1. U fazi izvršavanja ove instrukcije se nad sadržajima registra $A_{15...0}$ bloka *exec* koji se koristi kao akumulator i registra $B_{15...0}$ bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije realizuje logička I operacija i rezultat upisuje u registar $A_{15...0}$. Stoga se vrednošću 1 signala **and** bloka *exec* na izlazima $ALU_{15...0}$ aritmetičko logičke jedinice ALU formira rezultat logičke I operacije sadržaja registara $A_{15...0}$ i $B_{15...0}$ koji se dalje vrednošću 1 signala **ldA** upisuje u registar $A_{15...0}$. Potom se u sledećem koraku vrednostima 1 signala **ldN**, **ldZ**, **ldC** i **ldV** bloka *exec* u razrede PSWN i PSWZ programske statusne $PSW_{15...0}$ reči upisuju vrednosti signala **N** i **Z** formirane na osnovu sadržaja upisanog u registar $A_{15...0}$ i u razrede PSWC i PSWV neaktivne vrednosti i prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{42}$ **and**, **ldA**;
$step_{43}$ **ldN**, **ldZ**, **ldC**, **ldV**,
br $step_{6D}$

! OR !

! U korak $step_{44}$ se dolazi iz $step_{29}$ ukoliko signal operacije **OR** ima vrednost 1. U fazi izvršavanja ove instrukcije se nad sadržajima registra $A_{15...0}$ bloka *exec* koji se koristi kao akumulator i registra $B_{15...0}$ bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije realizuje logička ILI operacija i rezultat upisuje u registar $A_{15...0}$. Stoga se vrednošću 1 signala **or** bloka *exec* na izlazima $ALU_{15...0}$ formira rezultat logičke ILI operacije sadržaja registara $A_{15...0}$ i $B_{15...0}$ koji se dalje vrednošću 1 signala **ldA** upisuje u registar $A_{15...0}$. Potom se u sledećem koraku vrednostima 1 signala **ldN**, **ldZ**, **ldC** i **ldV** bloka *exec* u razrede PSWN i PSWZ programske statusne reči $PSW_{15...0}$ upisuju vrednosti signala **N** i **Z** formirane na osnovu sadržaja upisanog u

registar $A_{15...0}$ i u razrede PSWC i PSWV neaktivne vrednosti i prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

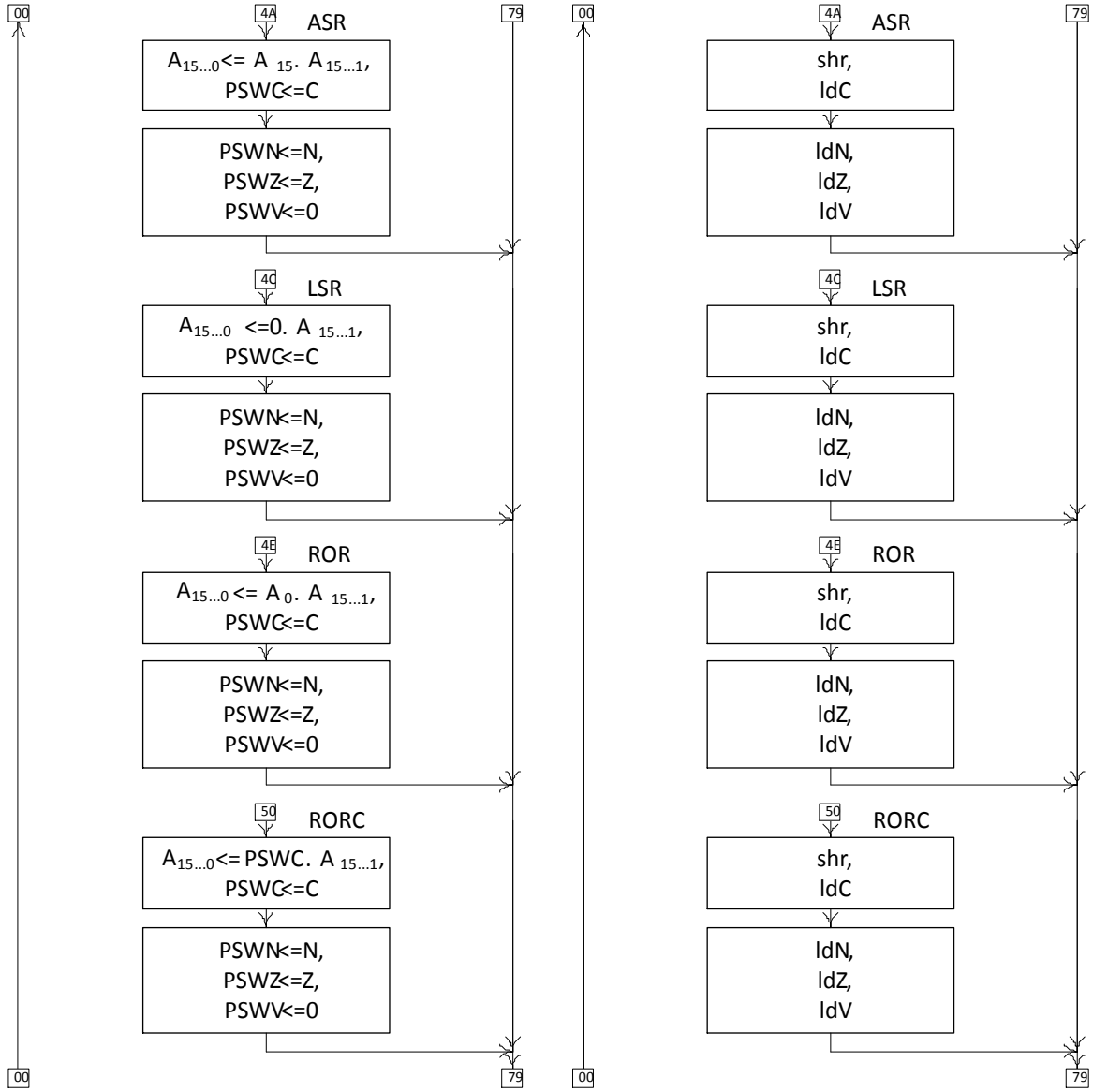
$step_{44}$ **or**, **ldA**;
$step_{45}$ **ldN**, **ldZ**, **ldC**, **ldV**,
br $step_{6D}$

! XOR !
! U korak $step_{46}$ se dolazi iz $step_{29}$ ukoliko signal operacije **XOR** ima vrednost 1. U fazi izvršavanja ove instrukcije se nad sadržajima registra $A_{15...0}$ bloka *exec* koji se koristi kao akumulator i registra $A_{15...0}$ bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije realizuje logička ekskluzivno ILI operacija i rezultat upisuje u registar $A_{15...0}$. Stoga se vrednošću 1 signala **xor** bloka *exec* na izlazima $ALU_{15...0}$ formira rezultat logičke ekskluzivno ILI operacije sadržaja registara $A_{15...0}$ i $B_{15...0}$ koji se dalje vrednošću 1 signala **ldA** upisuje u registar $A_{15...0}$. Potom se u sledećem koraku vrednostima 1 signala **ldN**, **ldZ**, **ldC** i **ldV** bloka *exec* u razrede PSWN i PSWZ programske statusne reči $PSW_{15...0}$ upisuju vrednosti signala **N** i **Z** formirane na osnovu sadržaja upisanog u registar $A_{15...0}$ i u razrede PSWC i PSWV neaktivne vrednosti i prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{46}$ **xor**, **ldA**;
$step_{47}$ **ldN**, **ldZ**, **ldC**, **ldV**,
br $step_{6D}$

! NOT !
! U korak $step_{48}$ se dolazi iz $step_{29}$ ukoliko signal operacije **NOT** ima vrednost 1. U fazi izvršavanja ove instrukcije se invertuju bitovi operand koji se nalazi u registru $B_{15...0}$ bloka *exec* i rezultat upisuje u registar $A_{15...0}$. Stoga se vrednošću 1 signala **not** bloka *exec* na izlazima $ALU_{15...0}$ formiraju invertovane vrednosti bitova registra $B_{15...0}$ koje se dalje vrednošću 1 signala **ldA** upisuju u registar $A_{15...0}$. Potom se u sledećem koraku vrednostima 1 signala **ldN**, **ldZ**, **ldC** i **ldV** bloka *exec* u razrede PSWN i PSWZ programske statusne reči $PSW_{15...0}$ upisuju vrednosti signala **N** i **Z** formirane na osnovu sadržaja upisanog u registar $A_{15...0}$ i u razrede PSWC i PSWV vrednosti 0 i prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{48}$ **not**, **ldA**;
$step_{49}$ **ldN**, **ldZ**, **ldC**, **ldV**,
br $step_{6D}$



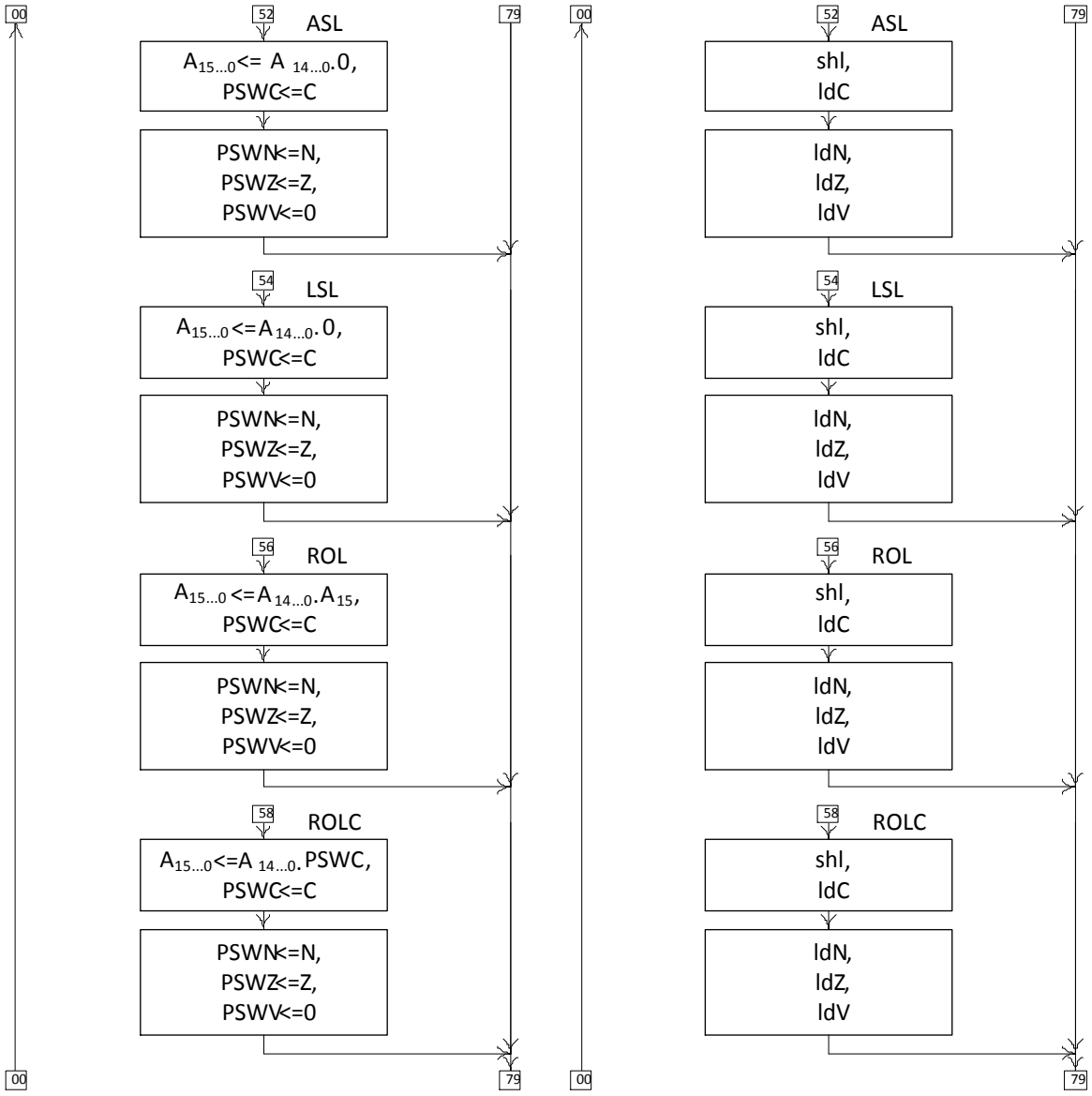
Slika 40 Izvršavanje operacije (šesti deo)

! ASR, LSR, ROR i ROLC !

! U korak $step_{4A}$ se dolazi iz $step_{29}$ ukoliko neki od signala operacije ASR, LSR, ROR i ROLC ima vrednost 1. U fazi izvršavanja ove instrukcije se bitovi registra $A_{15...0}$ bloka *exec* koji se koristi kao akumulator aritmetički pomeraju za jedno mesto udesno ukoliko signal **ASR** ima vrednost 1, logički pomeraju za jedno mesto udesno ukoliko signal **LSR** ima vrednost 1, rotiraju za jedno mesto udesno ukoliko signal **ROR** ima vrednost 1 i rotiraju zajedno sa razredom PSWC programske statusne reči $PSW_{15...0}$ za jedno mesto udesno ukoliko signal **RORC** ima vrednost 1. Stoga se vrednošću 1 signala **shr** bloka *exec* bitovi registra $A_{15...0}$ pomeraju udesno za jedno mesto, pri čemu se u u razred A_{15} upisuje signal **IR** sa izlaza multipleksera MX1 bloka *exec*. U slučaju aritmetičkog pomeranja za jedno mesto udesno to je signal $\mathbf{A_{15}}$. Ovaj signal se selektuje na izlazu IR multipleksera MX1 binarnom vrednošću 0 signala selekcije multipleksera MX1 koji se dobijaju na izlazu kodera CD1 zbog vrednosti 1 signala **ASR** na ulazu 0 kodera CD1. U slučaju logičkog pomeranja za jedno mesto udesno to je signal **0**. Ovaj signal se selektuje na izlazu IR multipleksera MX1 binarnom vrednošću 1 signala selekcije multipleksera MX1 koji se dobijaju na izlazu kodera CD1 zbog vrednosti 1 signala **LSR** na ulazu 1 kodera CD1. U slučaju rotiranja za jedno mesto udesno to je signal $\mathbf{A_0}$. Ovaj signal se selektuje na izlazu IR multipleksera MX1 binarnom vrednošću 2 signala selekcije multipleksera MX1 koji se dobijaju

na izlazu kodera CD1 zbog vrednosti 1 signala **ROR** na ulazu 2 kodera CD1. U slučaju rotiranja zajedno sa razredom PSWC programske statusne reči $PSW_{15...0}$ za jedno mesto udesno to je signal **PSWC**. Ovaj signal se selektuje na izlazu IR multipleksera MX1 binarnom vrednošću 3 signala selekcije multipleksera MX1 koji se dobijaju na izlazu kodera CD1 zbog vrednosti 1 signala **RORC** na ulazu 3 kodera CD1. Istovremeno se vrednošću 1 signala **ldC** bloka *exec* u razred PSWC programske statusne reči $PSW_{15...0}$ upisuje vrednost signala **C** bloka *exec*. U slučaju svih pomeranja i rotiranja za jedno mesto udesno to je signal $\mathbf{A_0}$. U koraku $step_{4B}$ se vrednostima 1 signala **ldN** i **ldZ** bloka *exec* u razrede PSWN i PSWZ programske statusne $PSW_{15...0}$ reči upisuju vrednosti signala **N** i **Z** bloka *exec* formirane na osnovu sadržaja upisanog u registar $A_{15...0}$ i vrednošću 1 signala **ldV** bloka *exec* u razred PSWV programske statusne reči $PSW_{15...0}$ upisuje vrednost 0 signala **V**. Iz koraka $step_{4B}$ se bezuslovno prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{4A}$ **shr**, **ldC**;
$step_{4B}$ **ldN**, **ldZ**, **ldV**,
br $step_{6D}$

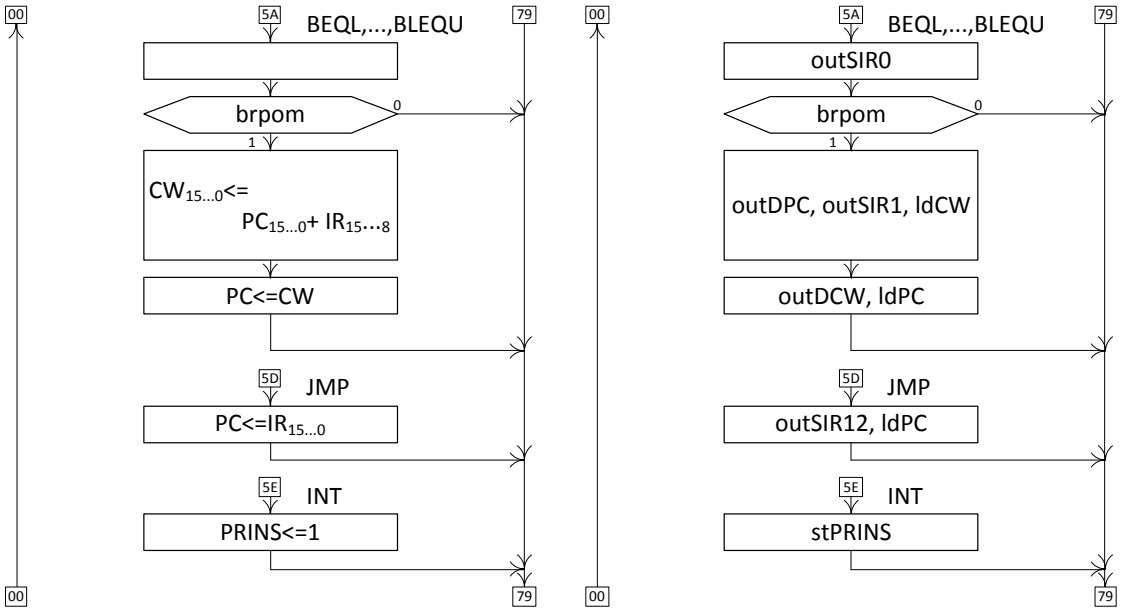


Slika 41 Izvršavanje operacije (sedmi deo)

! ASL, LSL, ROL i ROLC !

! U korak $step_{4C}$ se dolazi iz $step_{29}$ ukoliko neki od signala operacije ASL, LSL, ROL i ROLC ima vrednost 1. U fazi izvršavanja ove instrukcije se bitovi registra $A_{15...0}$ bloka *exec* koji se koristi kao akumulator aritmetički pomeraju za jedno mesto ulevo ukoliko signal **ASL** ima vrednost 1, logički pomeraju za jedno mesto ulevo ukoliko signal **LSL** ima vrednost 1, rotiraju za jedno mesto ulevo ukoliko signal **ROL** ima vrednost 1 i rotiraju zajedno sa razredom PSWC programske statusne reči $PSW_{15...0}$ za jedno mesto ulevo ukoliko signal **ROLC** ima vrednost 1. Stoga se vrednošću 1 signala **shl** bloka *exec* bitovi registra $A_{15...0}$ pomeraju ulevo za jedno mesto, pri čemu se u u razred A_0 upisuje signal **IL** sa izlaza multipleksera MX2 bloka *exec*. U slučaju aritmetičkog pomeranja za jedno mesto ulevo to je signal **0**. Ovaj signal se selektuje na izlazu IL multipleksera MX2 binarnom vrednošću 0 signala selekcije multipleksera MX2 koji se dobijaju na izlazu kodera CD2 zbog vrednosti 1 signala **ASL** na ulazu 0 kodera CD2. U slučaju logičkog pomeranja za jedno mesto ulevo to je signal **0**. Ovaj signal se selektuje na izlazu IL multipleksera MX2 binarnom vrednošću 1 signala selekcije multipleksera MX2 koji se dobijaju na izlazu kodera CD2 zbog vrednosti 1 signala **LSL** na ulazu 1 kodera CD2. U slučaju rotiranja za jedno mesto ulevo to je signal $\mathbf{A_{15}}$. Ovaj signal se selektuje na izlazu IL multipleksera MX2 binarnom vrednošću 2 signala selekcije multipleksera MX2 koji se dobijaju na izlazu kodera CD2 zbog vrednosti 1 signala **ROL** na ulazu 2 kodera CD2. U slučaju rotiranja zajedno sa razredom PSWC programske statusne reči $PSW_{15...0}$ za jedno mesto ulevo to je signal **PSWC**. Ovaj signal se selektuje na izlazu IL multipleksera MX2 binarnom vrednošću 3 signala selekcije multipleksera MX2 koji se dobijaju na izlazu kodera CD2 zbog vrednosti 1 signala **ROLC** na ulazu 3 kodera CD2. Istovremeno se vrednošću 1 signala **ldC** bloka *exec* u razred PSWC programske statusne reči $PSW_{15...0}$ upisuje vrednost signala **C** bloka *exec*. U slučaju svih pomeranja i rotiranja za jedno mesto ulevo to je signal $\mathbf{A_{15}}$. U koraku $step_{4D}$ se vrednostima 1 signala **ldN** i **ldZ** bloka *exec* u razrede PSWN i PSWZ programske statusne $PSW_{15...0}$ reči upisuju vrednosti signala **N** i **Z** bloka *exec* formirane na osnovu sadržaja upisanog u registar $A_{15...0}$ i vrednošću 1 signala **ldV** bloka *exec* u razred PSWV programske statusne reči $PSW_{15...0}$ upisuje vrednost 0 signala **V**. Iz koraka $step_{4D}$ se bezuslovano prelazi na korak $step_{6D}$ i fazu opsluživanje prekida. !

$step_{4C}$ **shl, ldC**;
$step_{4D}$ **ldN**, **ldZ**, **ldV**,
br $step_{6D}$



Slika 42 Izvršavanje operacije (osmi deo)

! BEQL,…, BLSSEU !

! U korak $step_{4E}$ se dolazi iz $step_{29}$ ukoliko neki od signala operacija uslovnog skoka **BEQL**, **BNEQL**, **BNEG**, **BNNEG**, **BOVF**, **BNVF**, **BCR**, **BNCR**, **BGRT**, **BGRE**, **BLSS**, **BLSQ**, **BGRTU**, **BGREU**, **BLSSU**, **BLEQU** ima vrednost 1 i prelazi na $step_{6D}$ i fazu opsluživanje prekida ukoliko signal **brpom** bloka *exec* ima vrednost 0 i na sledeći korak $step_{4F}$ ukoliko ima vrednost 1. Signal **brpom** ima vrednost 1 ukoliko vrednost 1 ima signal rezultata operacije **eql**, **neql**, **neg**, **nneg**, **ovf**, **novf**, **cr**, **ncr**, **grt**, **grte**, **lss**, **leq**, **grtu**, **grteu**, **lssu**, **lequ** određen vrednošću 1 signala operacije uslovnog skoka skoka **BEQL**, **BNEQL**, **BNEG**, **BNNEG**, **BOVF**, **BNVF**, **BCR**, **BNCR**, **BGRT**, **BGRE**, **BLSS**, **BLSQ**, **BGRTU**, **BGREU**, **BLSSU**, **BLEQU**. !

$step_{4E}$ **outSIR0,**

br (*if* $\overline{\textbf{brpom}}$ *then* $step_{4F}$);

! U korak $step_{4F}$ se dolazi iz $step_{4E}$ ukoliko signal **brpom** ima vrednost 1, čime je uslov za skok ispunjen. U ovom koraku se vrednošću 1 signala **outSPC** propusti sadržaj registra $PC_{15..0}$ bloka *fetch* na internu magistralu $Sbus_{15...0}$, a vrednošću 1 signala **outDIR12** propusti sadržaj registra $IR_{15...8}IR_{7..0}$ bloka *fetch* na internu magistralu $Dbus_{15...0}$. Ove dve vrednosti dolaze na ulaze sabirača ADD. Signali $ADD_{15...0}$ sa izlaza sabirača ADD koji predstavljaju adresu skoka se vrednostima 1 signala **ldCW** upisuju u registar CW. Vrednošću 1 signala **outDCW** sadržaj registra $CW_{15..0}$ se propušta kroz bafere sa tri stanja na internu magistralu $Dbus_{15..0}$, a zatim vrednošću 1 signala **ldPC** upisuje u registar $PC_{15...0}$. Pored toga iz koraka $step_{50}$ se bezuslovno prelazi na $step_{6D}$ i fazu opsluživanje prekida. !

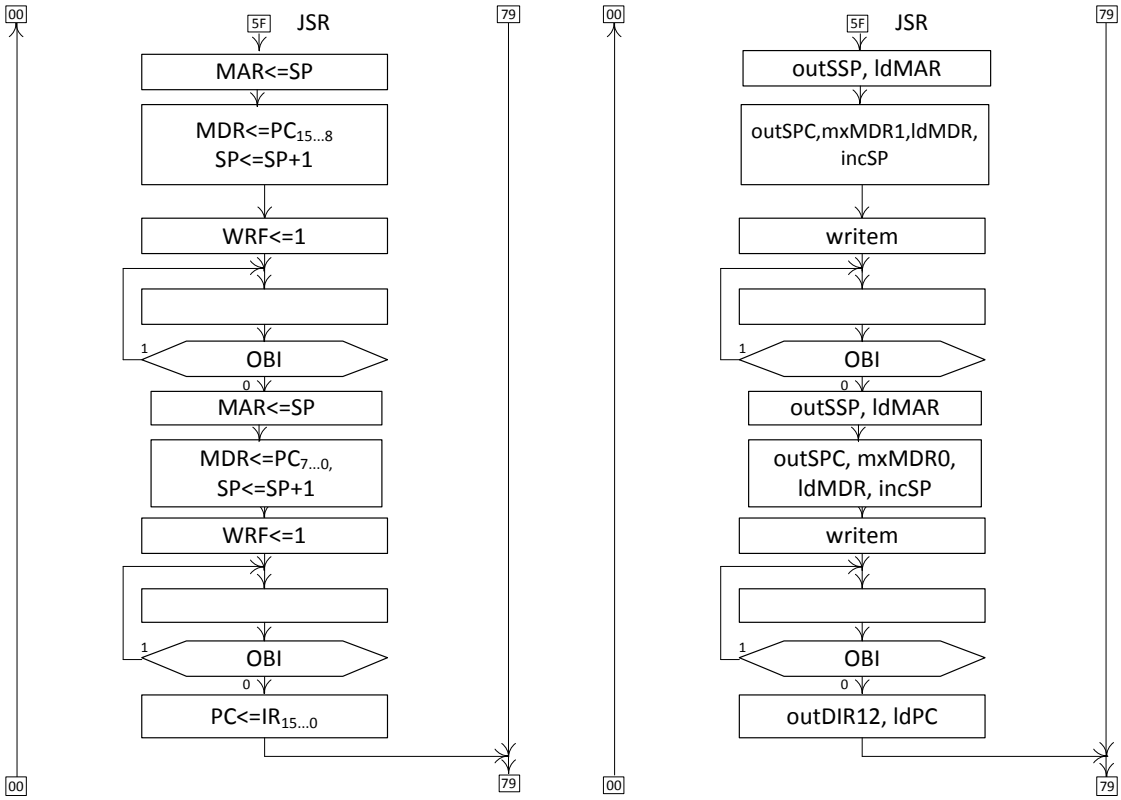
$step_{4F}$ **outDPC, outDIR21, ldCW**;
$step_{50}$ **outDCW, ldPC**,
br $step_{6D}$;

! JMP !
! U korak $step_{51}$ se dolazi iz $step_{29}$ ukoliko signal operacije **JMP** ima vrednost 1. U fazi izvršavanja ove instrukcije se realizuje bezuslovni skok na adresu koja je data u samoj instrukciji. Stoga se sadržaj registra $IR_{23...8}$ bloka *fetch* koji predstavlja adresu skoka vrednostima 1 signala **outDIR21** i **ldPC** propušta na internu magistralu $Dbus_{15...0}$ i upisuje u registar $PC_{15...0}$. Pored toga iz koraka $step_{51}$ se bezuslovno prelazi na $step_{6D}$ i fazu opsluživanje prekida. !

$step_{51}$ **outDIR21**, **ldPC**,
br $step_{6D}$

! INT !
! U korak $step_{52}$ se dolazi iz $step_{29}$ ukoliko signal operacije **INT** ima vrednost 1. U fazi izvršavanja ove instrukcije se samo evidentira da se radi o instrukciji prekida i prelazi na fazu opsluživanje prekida iz koje se zatim skače na prekidnu rutinu na osnovu broja ulaza u tabelu sa adresama prekidnih rutina koji je dat u samoj instrukciji. Stoga se vrednošću 1 signala **stPRINS** flip-flop PRINS bloka *intr* postavlja na vrednost 1 i bezuslovno prelazi na $step_{6D}$ i fazu opsluživanje prekida. !

$step_{52}$ **stPRINS**,
br $step_{6D}$



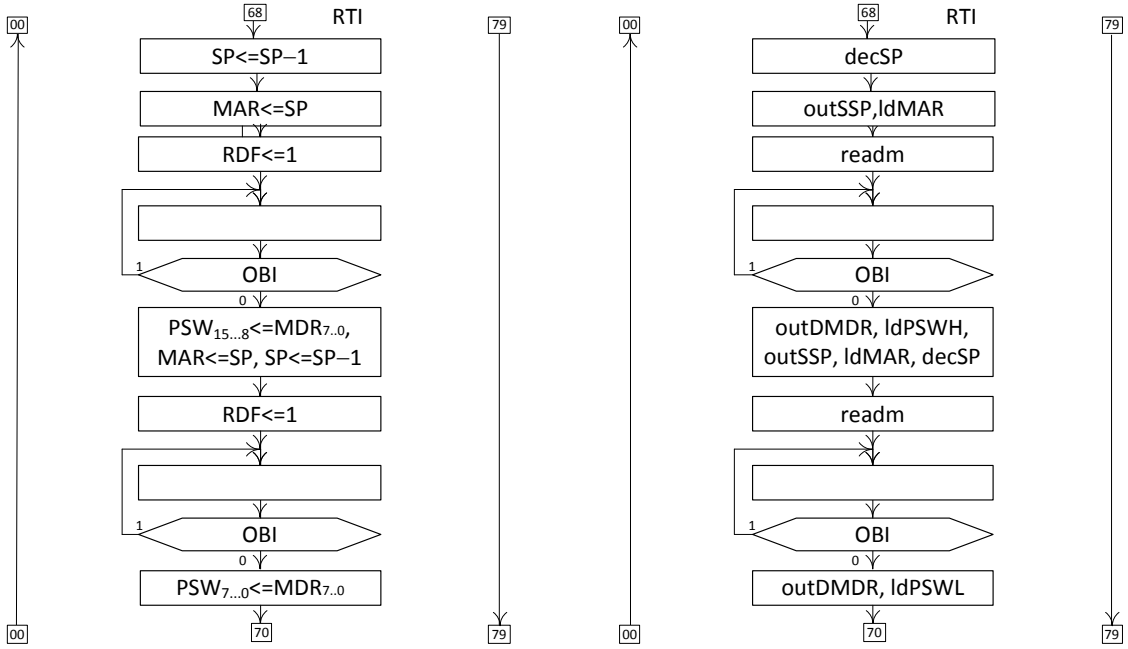
Slika 43 Izvršavanje operacije (deveti deo)

! JSR !

! U korak $step_{53}$ se dolazi iz $step_{29}$ ukoliko signal operacije **JSR** ima vrednost 1. U fazi izvršavanja ove instrukcije se realizuje skok na potprogram tako što se prvo na stek stavi tekući sadržaj programskog brojača i zatim realizuje bezuslovni skok na adresu koja je data u samoj instrukciji. Na stek se stavlja prvo viši a zatim i niži bajt registra $PC_{15...0}$. Stoga se najpre u koraku $step_{53}$ vrednostima 1 signala **outSSP** i **ldMAR**, sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ pa zatim u koraku $step_{65}$ vrednostima 1 signala **outSPC**, $\mathbf{mxMDR_1}$ i **ldMDR** sadržaj višeg bajta registra $PC_{15...8}$ propušta kroz multiplekser MX i upisuje u registar $MDR_{7...0}$ i vrednošću 1 signala **incSP** vrši inkrementiranje registra $SP_{15...0}$ bloka *addr*. Upis se realizuje u koracima $step_{55}$ i $step_{56}$. Zatim se u koraku $step_{57}$ vrednostima 1 signala **outSSP** i **ldMAR**, sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ pa zatim u koraku $step_{58}$ vrednostima 1 signala **outSPC**, $\mathbf{mxMDR_0}$ i **ldMDR** sadržaj nižeg bajta registra $PC_{7...0}$ propušta kroz multiplekser MX i upisuje u registar $MDR_{7...0}$ a zatim se vrednošću 1 signala **incSP** vrši inkrementiranje registra $SP_{15...0}$ bloka *addr*. Upis se realizuje u koracima $step_{59}$ i $step_{5A}$ na isti načina kao što se to radi u koracima $step_{55}$ i $step_{56}$ prilikom upisa nižeg bajta. Na kraju se u koraku $step_{5B}$ sadržaj registra $IR_{15...0}$ bloka *fetch* koji predstavlja adresu skoka vrednostima 1 signala **outDIR21** i **ldPC** propušta na internu magistralu $Dbus_{15...0}$ i upisuje u registar $PC_{15...0}$ i bezuslovno se prelazi na $step_{6D}$ i fazu opsluživanje prekida. !

$step_{53}$ **outSSP**, **ldMAR ;**
$step_{54}$ **outSSP**, $\mathbf{mxMDR_1}$, **ldMDR, incSP**;
$step_{55}$ **writem;**
$step_{56}$ *br* (*if* **OBI** *then* $step_{56}$);
$step_{57}$ **outSSP**, **ldMAR;**
$step_{58}$ **outSPC**, $\mathbf{mxMDR_0}$, **ldMDR**;
$step_{59}$ **writem;**
$step_{5A}$ *br* (*if* **OBI** *then* $step_{5A}$);
$step_{5B}$ **outDIR2**, **ldPC**,

br $step_{6D}$

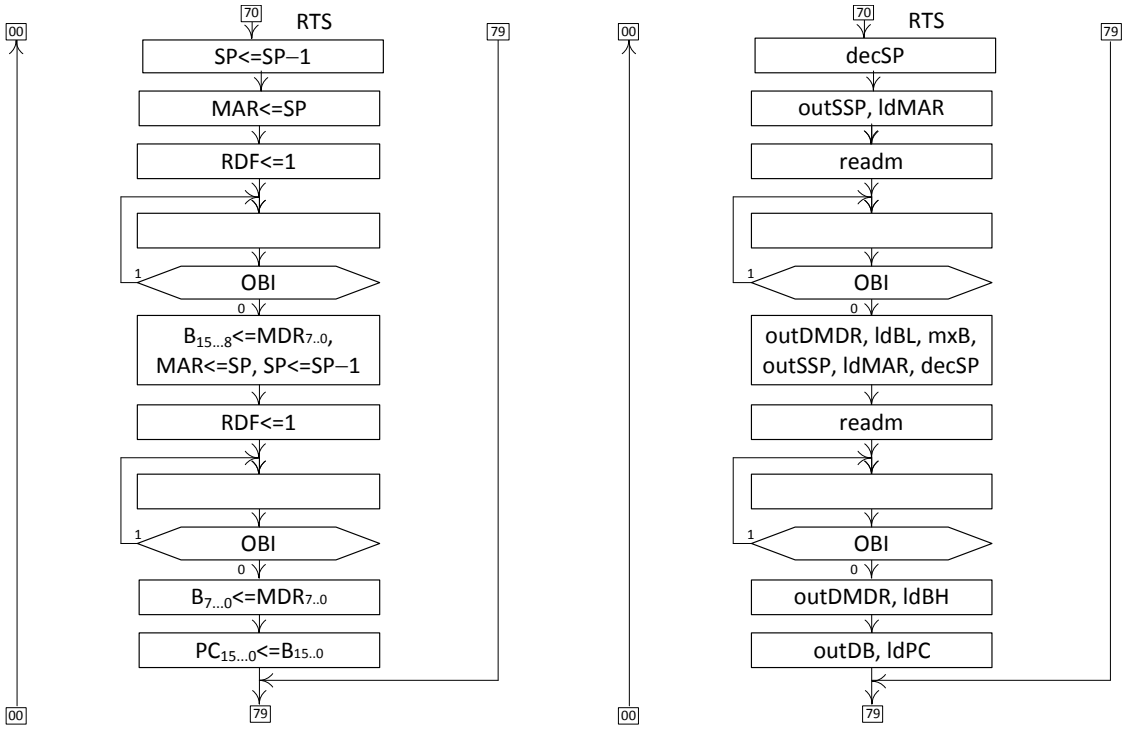


Slika 44 Izvršavanje operacije (deseti deo)

! RTI !

! U korak $step_{5C}$ se dolazi iz $step_{29}$ ukoliko signal operacije **RTI** ima vrednost 1. U fazi izvršavanja ove instrukcije vrednostima sa steka se restauriraju programska statusna reč $PSW_{15..0}$ i programski brojač $PC_{15...0}$. U koracima $step_{5C}$ do $step_{6C}$ se najpre vrednošću sa steka restaurira programska statusna reč $PSW_{15...0}$ bloka *exec*. Sa steka se najpre skida viši a zatim i niži bajt registra $PSW_{15...0}$. Prvo se u koraku $step_{5C}$ vrednošću 1 signala **decSP** dekrementira sadržaj registra $SP_{15...0}$ a potom u koraku $step_{5D}$ vrednostima 1 signala **outSSP** i **ldMAR** bloka *bus* sadržaj registra $SP_{15...0}$ bloka *addr* upisuje u registar $MAR_{15...0}$. Čitanje se realizuje u koracima $step_{5E}$ i $step_{5F}$ na isti načina kao što se to radi u koracima $step_{02}$ i $step_{03}$ u kojima se čita prvi bajt instrukcije. Na kraju se u koraku $step_{60}$ vrednošću 1 signala **outDMDR** i **ldPSWH** sadržaj registra $MDR_{7...0}$ bloka *bus* upisuje u viši bajt registra $PSW_{15...8}$ bloka *exec*. Istovremeno se vrednostima 1 signala **outSSP** i **ldMAR** sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ i vrednošću 1 signala **decSP** dekrementira sadržaj registra $SP_{15...0}$. Čitanje se realizuje u koracima $step_{61}$ i $step_{62}$ na isti načina kao što se to radi u koracima $step_{5E}$ i $step_{5F}$ u kojima se čita viši bajt. Na kraju se u koraku $step_{63}$ vrednošću 1 signala **outDMDR** i **ldPSWL** sadržaj registra $MDR_{7...0}$ bloka *bus* upisuje u niži bajt registra $PSW_{7...0}$. Time je 16-to bitna vrednost skinuta sa steka i smeštena u registar $PSW_{15...0}$, pa se prelazi na sledeći korak počev od koga se kao i u slučaju instrukcije RTS sa steka skida 16-to bitna vrednost i smešta u registar $PC_{15...0}$. !

$step_{5C}$ **decSP**;
$step_{5D}$ **outSSP, ldMAR**;
$step_{5E}$ **readm;**
$step_{5F}$ *br* (*if* **OBI** *then* $step_{5F}$);
$step_{60}$ **outDMDR, ldPSWH, outSSP, ldMAR, decSP**;
$step_{61}$ **readm;**
$step_{62}$ *br* (*if* **OBI** *then* $step_{62}$);
$step_{63}$ **outDMDR, ldPSWL**;



Slika 45 Izvršavanje operacije (jedanaesti deo)

! RTS !

! U korak $step_{64}$ se dolazi iz $step_{29}$ ukoliko signal operacije **RTS** ima vrednost 1. Pored toga u korak $step_{64}$ se dolazi i iz koraka $step_{63}$ kada signal operacije **RTI** ima vrednost 1. U oba slučaja se u koracima $step_{64}$ do $step_{6C}$ vrednošću sa steka restaurira programski brojač $PC_{15...0}$. Prvo se u koraku $step_{64}$ vrednošću 1 signala **decSP** dekrementira sadržaj registra $SP_{15...0}$ pa se sa steka najpre skida niži a zatim i viši bajt registra $PC_{15...0}$. Stoga se u koraku $step_{65}$ vrednostima 1 signala **outSSP** i **ldMAR** bloka *bus* sadržaj registra $SP_{15...0}$ bloka *addr* upisuje u registar $MAR_{15...0}$. Čitanje se realizuje u koracima $step_{66}$ i $step_{67}$ na isti načina kao što se to radi u koracima $step_{02}$ i $step_{03}$ u kojima se čita prvi bajt instrukcije. Na kraju se u koraku $step_{68}$ vrednošću 1 signala **outDMDR** i **ldBL** sadržaj registra $MDR_{7...0}$ bloka *bus* upisuje u viši bajt registra $B_{7...0}$ bloka *bus*. Istovremeno se vrednostima 1 signala **outSSP** i **ldMAR** sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ i vrednošću 1 signala **decSP** dekrementira sadržaj registra $SP_{15...0}$. Čitanje se realizuje u koracima $step_{69}$ i $step_{7A}$ na isti načina kao što se to radi u koracima $step_{66}$ i $step_{67}$ u kojima se čita niži bajt. Na kraju se u koraku $step_{6B}$ vrednošću 1 signala **outDMDR** i **ldBH** sadržaj registra $MDR_{7...0}$ upisuje u niži bajt registra $B_{7...0}$. Time je 16-to bitna vrednost skinuta sa steka i smeštena u registar $B_{15...0}$. Konačno se u koraku $step_{6C}$ sadržaj registra $B_{15...0}$ vrednošću 1 signala **outDB** propušta na internu magistralu $Dbus_{15..0}$ i vrednošću 1 signala **ldPC** upisuje u registar $PC_{15...0}$ i bezuslovno prelazi na $step_{6D}$ i fazu opsluživanje prekida. !

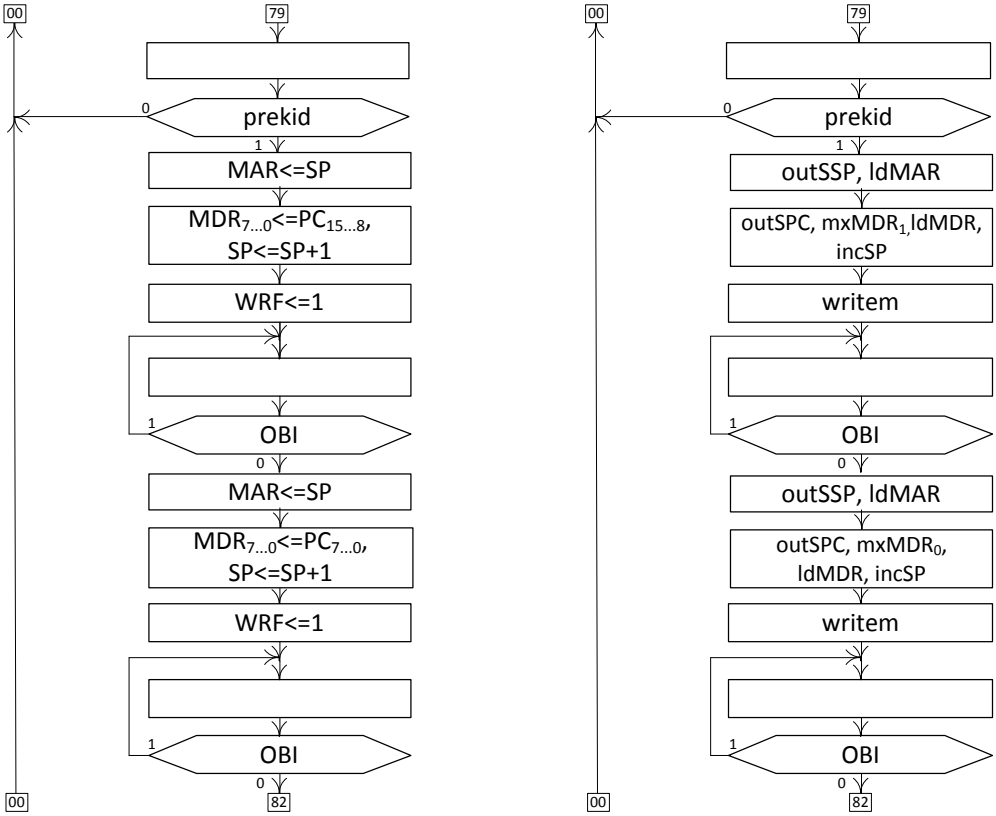
$step_{64}$ **decSP**;
$step_{65}$ **outSSP, ldMAR**;
$step_{66}$ **readm;**
$step_{67}$ *br* (*if* **OBI** *then* $step_{77}$);
$step_{68}$ **outDMDR, ldBL, mxB, outSSP, ldMAR, decSP**;
$step_{69}$ **readm;**
$step_{6A}$ *br* (*if* **OBI** *then* $step_{7A}$);
$step_{6B}$ **outDMDR, ldBH**;
$step_{6C}$ **outDB, ldPC**;

! Opsluživanje prekida !

! U korak $step_{6D}$ se dolazi iz koraka $step_{06}$ ukoliko signal **PRCOD** ima vrednost 1, iz koraka $step_{08}$ ukoliko signal **PRADR** ima vrednost 1 i iz koraka na završetku faze izvršavanje instrukcije. U koraku $step_{6D}$ se, u zavisnosti od toga da li signal **prekid** bloka *intr* ima vrednost 0 ili 1, ili završava izvršavanje tekuće instrukcije i prelaskom na korak $step_{00}$ započinje faza čitanje instrukcije sledeće instrukcije ili se produžava izvršavanje tekuće instrukcije i prelaskom na korak $step_{6D}$ produžava faza opsluživanje prekida tekuće instrukcije. !

$step_{6D}$ br (*if* $\overline{\textbf{prekid}}$ *then* $step_{00}$);

! Opsluživanje prekida se sastoji iz tri grupe koraka u kojima se realizuje čuvanje konteksta procesora, utvrđivanje broja ulaza i utvrđivanje adrese prekidne rutine. !



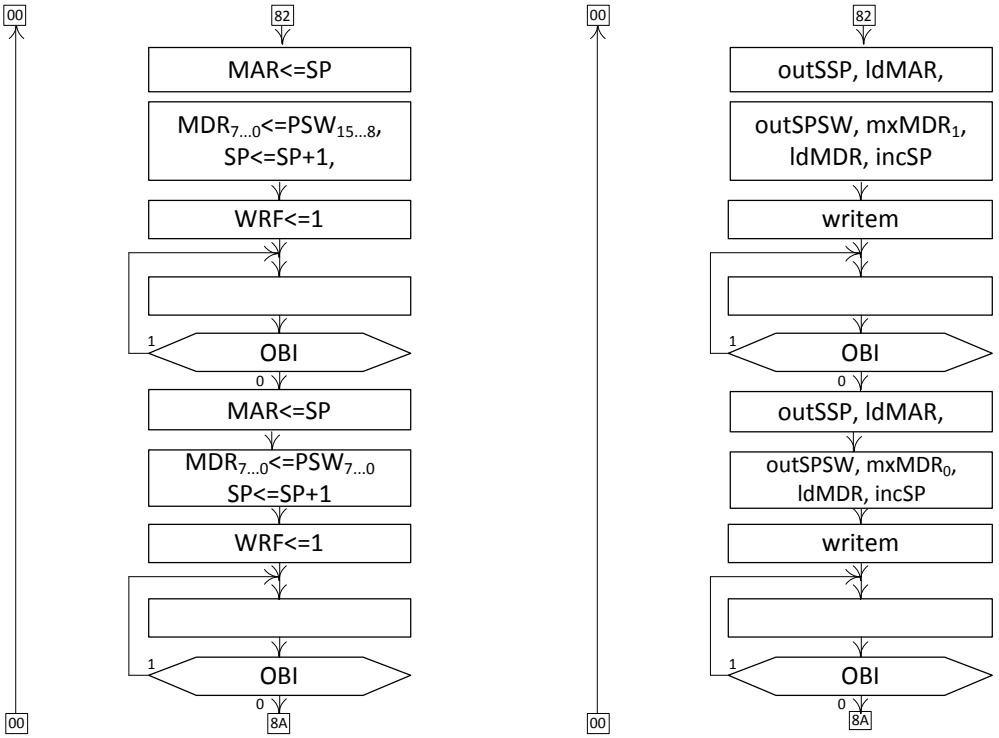
Slika 46 Opsluživanje prekida (prvi deo)

! Čuvanje konteksta procesora !

! Kontekst procesora i to $PC_{15...0}$ i $PSW_{15...0}$ se čuva u koracima $step_{6D}$ do $step_{91}$. U koracima $step_{6D}$ do $step_{75}$ se na stek stavlja programski brojač $PC_{15...0}$. Na stek se stavlja prvo viši a zatim i niži bajt registra $PC_{15...0}$. Najpre se u koraku $step_{6E}$ vrednostima 1 signala **outSSP** i **ldMAR**, sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ a u koraku $step_{6F}$ vrednostima 1 signala **outSPC, mxMDR$_1$** i **ldMDR** sadržaj višeg bajta registra $PC_{15...8}$ propušta kroz multiplekser MX i upisuje u registar $MDR_{7...0}$. Istovremeno se vrednošću 1 signala **incSP** vrši inkrementiranje registra $SP_{15...0}$ bloka *addr*. Upis se realizuje u koracima $step_{70}$ i $step_{71}$. Potom se u koraku $step_{72}$ vrednostima 1 signala **outSSP,** i **ldMAR**, sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ pa u koraku $step_{73}$ vrednostima 1 signala **outSPC, mxMDR$_0$** i **ldMDR** sadržaj višeg bajta registra $PC_{7...0}$ propušta kroz multiplekser MX i upisuje u registar $MDR_{7...0}$. Upis se realizuje u koracima

$step_{74}$ i $step_{75}$ na isti načina kao što se to radi u koracima $step_{70}$ i $step_{71}$ prilikom upisa nižeg bajta.!

$step_{6E}$ **outSSP**, **ldMAR ;**
$step_{6F}$ **outSPC**, **mxMDR$_1$**, **ldMDR, incSP**;
$step_{70}$ **writem;**
$step_{71}$ *br* (*if* **OBI** *then* $step_{71}$);
$step_{72}$ **outSSP**, **ldMAR ;**
$step_{73}$ **outSPC**, **mxMDR$_0$**, **ldMDR**, **incSP**;
$step_{74}$ **writem;**
$step_{75}$ *br* (*if* **OBI** *then* $step_{75}$);

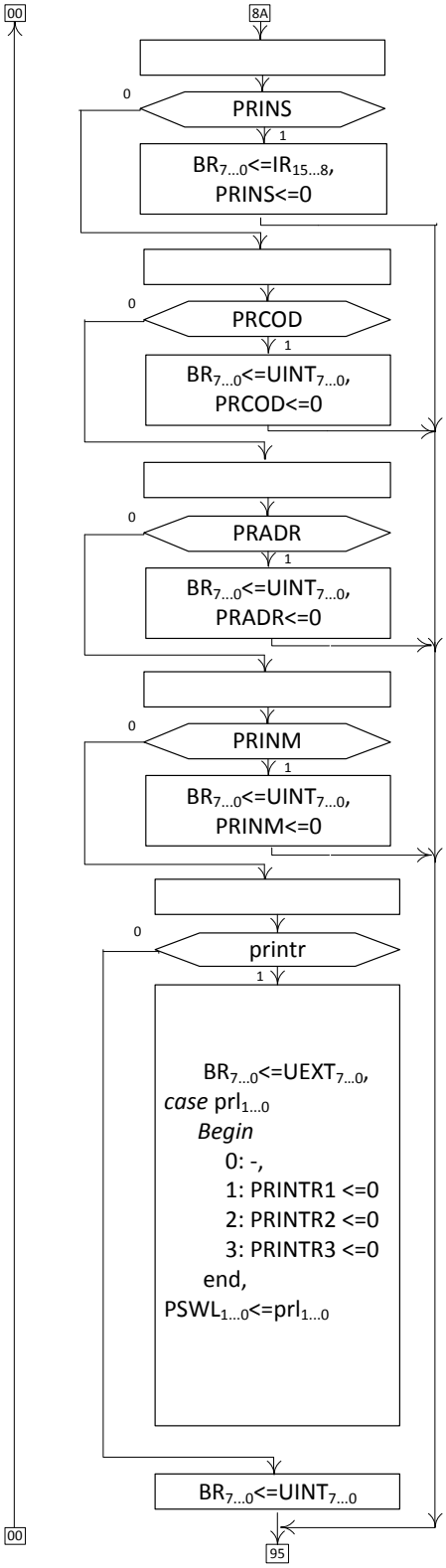


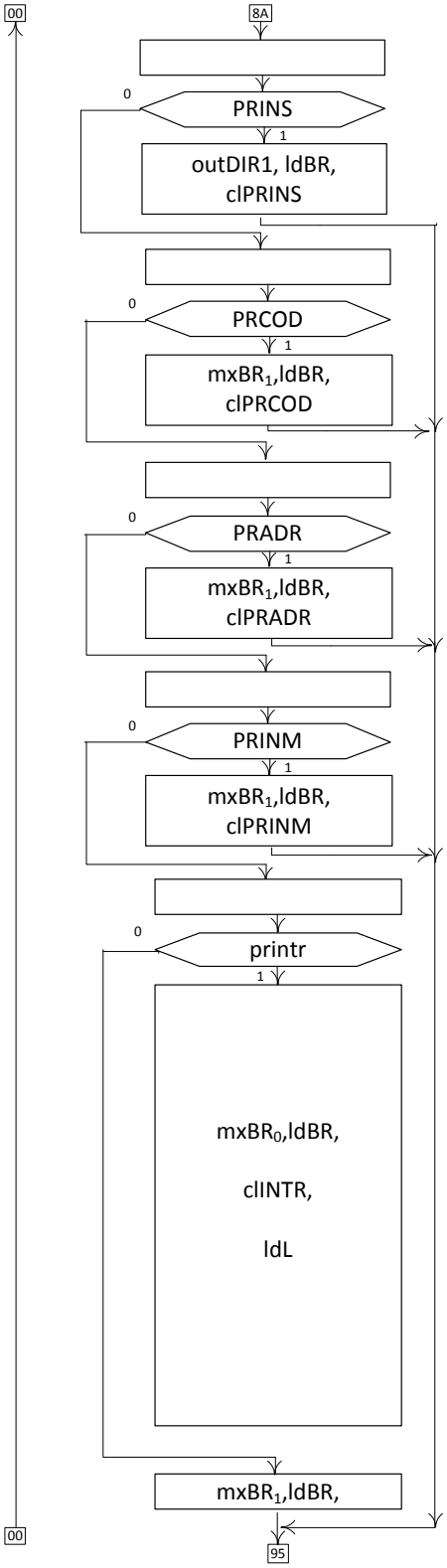
Slika 47 Opsluživanje prekida (drugi deo)

! U koracima $step_{76}$ do $step_{4D}$ se na stek stavlja programska statusna reč $PSW_{15...0}$ bloka *exec*. Na stek se stavlja prvo viši a zatim i niži bajt registra $PSW_{15...0}$. Najpre se u koraku $step_{76}$ vrednostima 1 signala **outSSP** i **ldMAR**, sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ pa u koraku $step_{77}$ vrednostima 1 signala **outSPSW, mxMDR$_1$** i **ldMDR** sadržaj višeg bajta registra $PSW_{7...0}$ propušta kroz multiplekser MX bloka *bus* i upisuje u registar $MDR_{7...0}$. Istovremeno se vrednošću 1 signala **incSP** vrši inkrementiranje registra $SP_{15...0}$ bloka *addr*. Upis se realizuje u koracima $step_{78}$ i $step_{79}$. Zatim se u koraku $step_{7A}$ vrednostima 1 signala **outSSP** i **ldMAR**, sadržaj registra $SP_{15...0}$ upisuje u registar $MAR_{15...0}$ i pa u koraku $step_{7B}$ vrednostima 1 signala **outSPSW, mxMDR$_0$** i **ldMDR** sadržaj nižeg bajta registra $PSW_{15...8}$ propušta kroz multiplekser MX i upisuje u registar $MDR_{7...0}$ pa se zatim vrednošću 1 signala **incSP** vrši inkrementiranje registra $SP_{15...0}$ bloka *addr*. Upis se realizuje u koracima $step_{7C}$ i $step_{7D}$ na isti načina kao što se to radi u koracima $step_{77}$ i $step_{78}$ prilikom upisa višeg bajta. !

$step_{76}$ **outSSP**, **ldMAR ;**
$step_{77}$ **outSPSW**, **mxMDR$_1$**, **ldMDR, incSP**;
$step_{78}$ **writem;**
$step_{79}$ *br* (*if* **OBI** *then* $step_{79}$);
$step_{7A}$ **outSSP**, **ldMAR ;**

step$_{7B}$ **outSPSW, mxMDR$_0$, ldMDR, incSP**;
step$_{7C}$ **writem;**
step$_{7D}$ *br* (*if* **OBI** *then* step$_{7D}$);





Slika 48 Opsluživanje prekida (treći deo)

! Utvrđivanje broja ulaza !
! U korak $step_{7E}$ se dolazi iz $step_{7D}$. U koracima $step_{7E}$ do $step_{89}$ se utvrđuje broj ulaza u tabelu sa adresama prekidnih rutina i upisuje u registar $BR_{7...0}$ bloka *intr*. U ovim koracima se po opadajućim prioritetima utvrđuje zahtev za prekid najvišeg prioriteta i za njega određuje broj ulaza u tabelu sa adresama prekidnih rutina. !
! Provera da li postoji zahtev za prekid zbog izvršavanja instrukcije prekida INT. !
! U koraku $step_{7E}$ se vrši provera da li signal **PRINS** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak $step_{7E}$ ili $step_{7D}$, respektivno. Ukoliko signal **PRINS** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku $step_{7E}$ se vrednostima 1 signala **outDIR21** i **ldBR** i vrednostima 0 signala $\mathbf{mxBR_1}$ i $\mathbf{mxBR_0}$ bloka *intr* sadržaj razreda $IR_{15...8}$, koji sadrži broj ulaza, preko interne magistrale $Dbus_{15..0}$ upisuje u registar $BR_{7...0}$. Istovremeno se vrednošću 1 signala **clPRINS** bloka *intr* se flip-flop PRINS postavlja na vrednost 0. Iz koraka $step_{7F}$ se prelazi na korak $step_{89}$ radi utvrđivanja adrese prekidne rutine. !

$step_{7E}$ *br* (*if* $\overline{\mathbf{PRINS}}$ *then* $step_{7D}$);
$step_{7F}$ **outDIR1, ldBR**, **clPRINS**,
br $step_{89}$;

! Provera da li postoji zahtev za prekid zbog greške u kodu operacije. !
! U koraku $step_{80}$ se vrši provera da li signal **PRCOD** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak $step_{81}$ ili $step_{82}$, respektivno. Ukoliko signal **PRCOD** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku $step_{81}$ se vrednošću 1 signala $\mathbf{mxBR_1}$ bloka *intr* sadržaj $UINT_{7...0}$ sa izlaza kodera CD1, koji sadrži broj ulaza, propušta kroz multiplekser MX bloka *intr* i vrednošću 1 signala **ldBR** upisuje u registar $BR_{7...0}$. Istovremeno se vrednošću 1 signala **clPRCOD** bloka *intr* flip-flop PRCOD postavlja na vrednost 0. Iz koraka $step_{81}$ se prelazi na korak $step_{89}$ radi utvrđivanja adrese prekidne rutine. !

$step_{80}$ *br* (*if* $\overline{\mathbf{PRCOD}}$ *then* $step_{82}$);
$step_{81}$ $\mathbf{mxBR_1}$, **ldBR**, **clPRCOD**,
br $step_{89}$;

! Provera da li postoji zahtev za prekid zbog greške u načinu adresiranja. !
! U koraku $step_{82}$ se vrši provera da li signal **PRADR** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak $step_{83}$ ili $step_{84}$, respektivno. Ukoliko signal **PRADR** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku $step_{83}$ se vrednošću 1 signala $\mathbf{mxBR_1}$ bloka *intr* sadržaj $UINT_{7...0}$ sa izlaza kodera CD1, koji sadrži broj ulaza, propušta kroz multiplekser MX bloka *intr* i vrednošću 1 signala **ldBR** upisuje u registar $BR_{7...0}$. Istovremeno se vrednošću 1 signala **clPRADR** bloka *intr* flip-flop PRADR postavlja na vrednost 0. Iz koraka $step_{83}$ se prelazi na korak $step_{89}$ radi utvrđivanja adrese prekidne rutine. !

$step_{82}$ *br* (*if* $\overline{PRADR}$ *then* $step_{84}$);
$step_{83}$ $\mathbf{mxBR_1}$, **ldBR**, **clPRADR**,
br $step_{89}$;

! Provera da li postoji spoljašnji nemaskirajući zahtev za prekid. !
! U koraku $step_{84}$ se vrši provera da li signal **PRINM** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak $step_{85}$ ili $step_{86}$, respektivno. Ukoliko signal **PRINM** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku $step_{85}$ se vrednošću 1 signala $\mathbf{mxBR_1}$ bloka *intr* sadržaj $UINT_{7...0}$ sa izlaza kodera CD1, koji sadrži broj ulaza, propušta kroz multiplekser MX bloka *intr* i vrednošću 1

signala **ldBR** upisuje u registar $BR_{7...0}$. Istovremeno se vrednošću 1 signala **clPRINM** bloka *intr* flip-flop PRINM postavlja na vrednost 0. Iz koraka $step_{85}$ se prelazi na korak $step_{89}$ radi utvrđivanja adrese prekidne rutine. !

$step_{84}$ *br* (*if* $\overline{\textbf{PRINM}}$ *then* $step_{86}$);
$step_{85}$ **mxBR$_1$**, **ldBR**, **clPRINM**,
br $step_{89}$;

! Provera da li postoji spoljašnji maskirajući zahtev za prekid. !
! U koraku $step_{86}$ se vrši provera da li signal **printr** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak $step_{87}$ ili $step_{89}$, respektivno. Ukoliko signal **printr** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku $step_{87}$ se vrednošću 1 signala **mxBR$_0$** bloka *intr* sadržaj $UEXT_{7...0}$ sa izlaza kodera CD2, koji sadrži broj ulaza, propušta kroz multiplekser MX bloka *intr* i vrednošću 1 signala **ldBR** upisuje u registar $BR_{7...0}$. Istovremeno se vrednošću 1 signala **clINTR** bloka *intr* jedan od flip-flopova $PRINTR_3$ do $PRINTR_1$ postavlja na vrednost 0. Flip-flop $PRINTR_3$ do $PRINTR_1$ koji se postavlja na vrednost 0 je selektovan binarnim vrednostima 3 do 1, respektivno, signala **prl$_1$** do **prl$_0$** bloka *intr* i odgovara liniji najvišeg nivoa prioriteta po kojoj je stigao spoljašnji maskirajući zahtev za prekid koji nije selektivno maskiran odgovarajućim razredom registra maske IMR. Pored toga vrednošću 1 signala **ldL** bloka *exec* se signali **prl$_1$** do **prl$_0$** , koji predstavljaju binarnu vrednost nivoa prioriteta prekidne rutine na koju se skače, upisuju u razrede $PSWL_1$ do $PSWL_0$ programske statusne reči $PSW_{15...0}$ bloka *exec*. Iz koraka $step_{87}$ se prelazi na korak $step_{89}$ radi utvrđivanja adrese prekidne rutine. !

$step_{86}$ *br* (*if* $\overline{\textbf{printr}}$ *then* $step_{88}$);
$step_{87}$ **mxBR$_0$**, **ldBR**, **clINTR**, **ldL**,
br $step_{89}$;

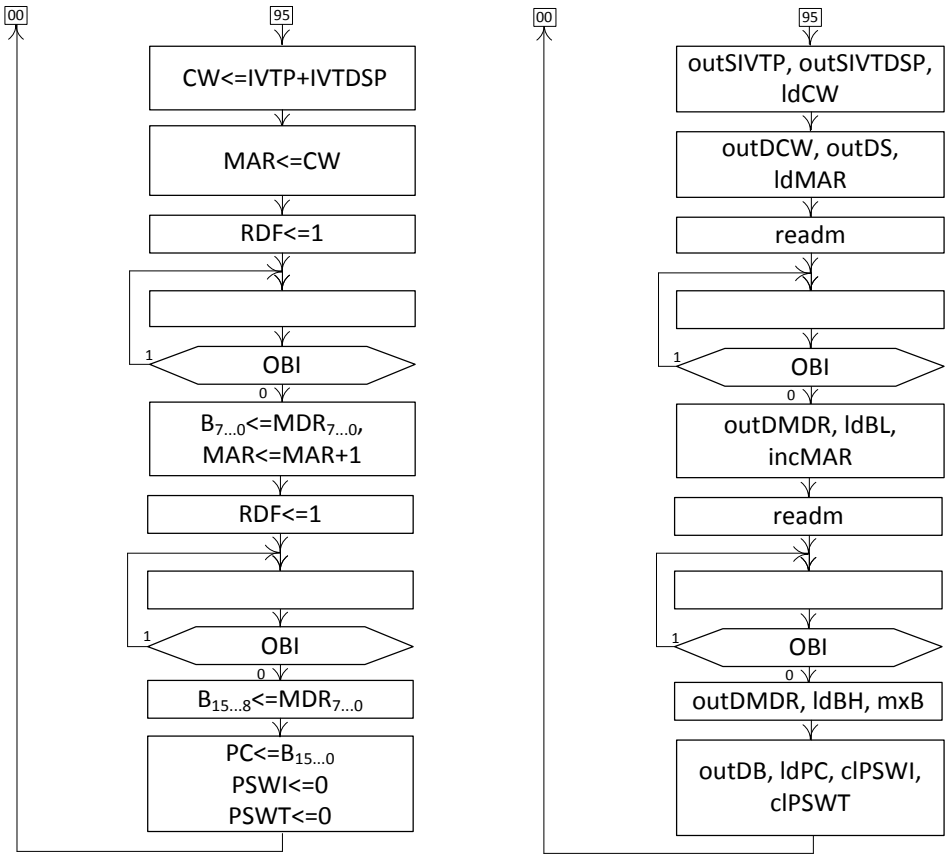
! Prekid posle svake instrukcije !
! U korak $step_{88}$ se dolazi iz $step_{87}$ ukoliko signal **printr** ima vrednost 0. Kako se u ovaj korak dolazi jedino ukoliko se proverom signala **prekid** u koraku $step_{6D}$ utvrđuje da postoji barem jedan zahtev za prekid i proverom signala **PRINS, PRCOD, PRADR, PRINM** i **printr** u koracima $step_{80}$, $step_{82}$, $step_{84}$ i $step_{86}$ utvrđuje da ovi signali imaju vrednost 0 i da odgovarajućih zahteva za prekid nema, to znači da postoji zahtev za prekid zbog zadatog režima rada prekid posle svake instrukcije. Stoga se vrednošću 1 signala **mxBR$_1$** bloka *intr* sadržaj $UINT_{7...0}$ sa izlaza kodera CD1, koji sadrži broj ulaza, propušta kroz multiplekser MX i vrednošću 1 signala **ldBR** upisuje u registar $BR_{7...0}$. Iz koraka $step_{88}$ se prelazi na korak $step_{89}$ radi utvrđivanja adrese prekidne rutine. !

$step_{88}$ **mxBR$_1$**, **ldBR**;

! Utvrđivanje adrese prekidne rutine !
! U koracima $step_{89}$ do $step_{91}$ se na osnovu dobijenog broja ulaza i sadržaja registra koji ukazuje na početnu adresu tabele sa adresama pekidnih rutina, iz odgovarajućeg ulaza čita adresa prekidne rutine i upisuje u programski brojač $PC_{15...0}$ bloka *fetch*. U koraku $step_{89}$ se vrednostima 1 signala **outSIVTP** i **outIVTDSP**, sadržaji registara $IVTP_{15...0}$ i $IVTDSP_{15...0}$, koji predstavlja sadržaj registra $BR_{7...0}$ pomeren ulevo za jedno mesto i proširen nulama do dužine 16 bita, preko interne magistrala $Sbus_{15..0}$ vodi na ulaze sabirača ADD. Istovremeno se vrednošću 1 signala **ldCW** sadržaj $ADD_{15...0}$ sa izlaza sabirača ADD upisuje u registar $CW_{15...0}$. Vrednošću 1 signala **outDCW** se sadržaj registra $CW_{15..0}$ propušta kroz trostatički bafer na internu magistralu $Dbus_{15..0}$, zatim vrednošću 1 signala **outDS** sa interne magistrale $Dbus_{15..0}$ na internu magistralu $Sbus_{15..0}$ i na kraju vrednošću 1 signala **ldMAR** upisuje u registar $MAR_{15...0}$. Time se u registru $MAR_{15...0}$ nalazi adresa memorijske lokacije počev od koje treba pročitati dva bajta koji predstavljaju niži i viši bajt adrese prekidne rutine. Čitanje prvog bajta se realizuje u koracima $step_{8B}$ i $step_{8C}$, a

drugog bajta u koracima $step_{8E}$ i $step_{8F}$ na isti način kao u koracima $step_{02}$ i $step_{03}$ kod čitanja prvog bajta instrukcije. U koraku $step_{8D}$ se prvi bajt vrednošću 1 signala **ldBL** upisuje u niži bajt registra $B_{7...0}$ bloka *bus*, a vrednošću 1 signala **incMAR** adresni registar $MAR_{15...0}$ inkrementira na adresu sledećeg bajta. U koraku $step_{90}$ se drugi bajt vrednošću 1 signala **ldBH** upisuje u viši bajt registra $B_{15...8}$. Time se u registru $B_{15...0}$ nalazi adresa prekidne rutine. Na kraju se u koraku $step_{91}$ vrednošću 1 signala **outDB** sadržaj registra $B_{15...0}$ propšta na internu magistralu $Dbus_{15...0}$ i vrednošću 1 signala **ldPC** upisuje u registar $PC_{15...0}$. Time se u registru $PC_{15...0}$ nalazi adresa prve instrukcije prekidne rutine. Vrednostima 1 signala **clPSWI** i **clPSWT** se u razrede PSWI i PSWT bloka *exec* upisuju vrednosti 0. Time se u prekidnu rutinu ulazi sa režimom rada u kome su maskirani svi maskirajući prekidi i u kome nema prekida posle svake instrukcije. Iz koraka $step_{91}$ se bezuslovno prelazi na $step_{00}$. !

$step_{89}$ **outSIVTP, outSIVTDSP, ldCW**;
$step_{8A}$ **outDCW, outDS, ldMAR;**
$step_{8B}$ **readm;**
$step_{8C}$ *br* (*if* **OBI** *then* $step_{8C}$);
$step_{8D}$ **outDMDR, ldBL, incMAR;**
$step_{8E}$ **readm;**
$step_{8F}$ *br* (*if* **OBI** *then* $step_{8F}$);
$step_{90}$ **outDMDR, ldBH, mxB**;
$step_{91}$ **outDB, ldPC, clPSWI, clPSWT,**
br $step_{00}$;



Slika 49 Opsluživanje prekida (četvrti deo)

5.2.3 Struktura upravljačke jedinice

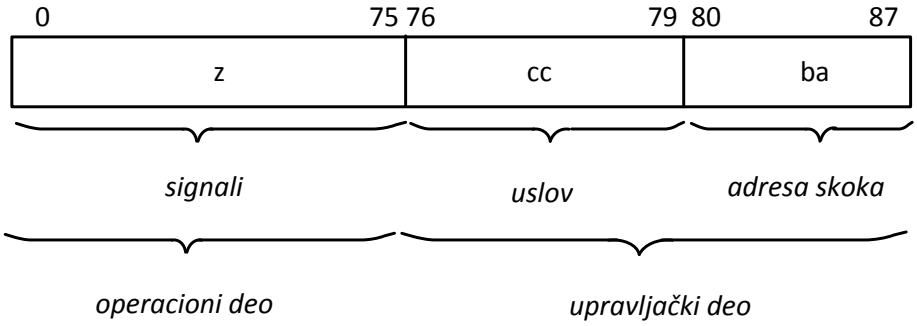
U sekvenci upravljačkih signala po koracima se svakom operacionom koraku, u kome se generišu upravljački singali operacione jedinice, pridružuje binarna reč čiji je format dat na slici 50 i svakom upravljačkom koraku, u kome se realizuju skokovi, pridružuje

binarna reč čiji je format dat u narednom delu odeljka. Te binarne reči se nazivaju mikroinstrukcijama tj. mikronarebama. Miktroinstrukcije pridružene operacionim koracima nazivaju se operacione mikroinstrukcije, dok se mikroinstrukcije pridružene upravljačkim koracima nazivaju upravljačke mikroinstrukcije. Uređeni niz mikroinstrukcija pridruženih operacionim koracima i upravljačkim koracima, naziva se mikroprogram.

Poljem *z* dužine 18 bita operacione mikroinstrukcije određuju se vrednosti svih upravljačkih signala. Uzeto je da svakom upravljačkom singalu odgovara poseban bit.

Poljem *cc* dužine 4 bita upravljačke mikroinstrukcije specificira se bezuslovni skok, uslovni skokovi na osnovu vrednosti svakog od signala logičkog uslova i višestruki uslovni skokovi. Polje *ba* dužine 8 bita upravljačke mikroinstrukcije poredstavlja adresu mikroinstrukcije u mikroprogramu na koju se skače u slučaju bezuslovnog skoka u u slučaju uslovnog skoka ukoliko je vrednost odgovarajućeg signala logičkog uslova aktivna.

Upravljačka jedinica se sastoji iz mikroprogramske memorije, mikroprogramskog brojača, prihvatnog registra mikroinstrukcije, kombinacione mreže za generisanje upravljačkih signala i kombinacione mreže za generisanje nove vrednosti mikroprogramskog brojača. Mikroprogramska memorija služi za smeštanje mikroprograma. Mikroprogramski brojač određuje adresu mikroinstrukcije u mikroprogramskoj memoriji. Prihvatni registar mikroinstrukcije služi za prihvatanje mikroinstrukcije očitane iz mikroprogramske memorije. Kombinaciona mreža za generisanje upravljačkih signala generiše dve grupe signala i to upravljačke signale operacione jedinice i upravljačke signale upravljačke jedinice. Upravljački signali operacione jedinice se generišu na osnovu vrednosti bitova polja *z*. Upravljački signali upravljačke jedinice se generišu na osnovu vrednosti bitova polja *cc* i signala logičkih uslova. Njima se sadržaj mikroprogramskog brojača ili inkrementira ili se u mikroprogramski brojač preko kombinacione mreže za generisanje nove vrednosti mikroprogramskog brojača upisuje vrednost određena poljem *ba* i time realizuje skok u mikroprogramskoj memoriji.



Slika 50 Format mikroinstrukcije

U slučaju razmatrane operacione jedinice format operacione i upravljačke mikroinstrukcije je dat na slici 51

0	1	2	3	4	5	6	7
0	outSPC	outDGPR	outSIR21	outDB	trans	outSA	outDA

8	9	10	11	12	13	14	15
add	ldN	sub	and	or	xor	not	shr

16	17	18	19	20	21	22	23
shl	outSIR0	outDPC	outDCW	outSSP	outDIR21	outDIR1	mxBR

24	25	26	27	28	29	30	31
mxBR1	mxBR0	outSIVTP	outSPSW	stPRCOD	stPRADR	ldMAR	ldIR0

32	33	34	35	36	37	38	39
ldIR1	ldIR2	ldMDR	wrGPR	ldSP	ldA	ldZ	outSIR1

40	41	42	43	44	45	46	47
ldPC	ldPSWH	ldPSWL	ldBR	outSIVTDSP	outDS	readm	writem

48	49	50	51	52	53	54	55
ldIVTP	incPC	ldC	ldCW	outSSP	clPRINS	clPRCOD	clPRADR

56	57	58	59	60	61	62	63
clPRINM	clPSWI	mxMDR0	mxMDR1	outDMDR	ldV	ldL	clPSWI

64	65	66	67	68	69	70	71
clPSWT	incMAR	ldBL	ldBH	stPRINS	decSP	clStart	-

72	73	74	75	76	77	78	79
-	-	-	-	cc			

80	81	81	83	84	85	86	87
ba							

Slika 51 mikroinstrukcija

Bitovi polja *cc* mikroinstrukcije koriste se za kodiranje upravljačkih signala kojima se određuje da li treba realizovati skok u mikroprogramu i to: bezuslovni skok, uslovni skok i višestruki uslovni skok ili preći na sledeću mikroinstrukciju.

Bezuslovni skok se realizuje u onim koracima sekvence upravljačkih signala po koracima u kojima se pojavljuju iskazi tipa *br* $step_A$. Simbolička oznaka signala bezuslovnog skoka koji za svaki od njih treba generisati i način njegovog kodiranja bitovima polja *cc* mikroinstrukcije dati su u tabeli narednoj tabeli :

cc	signal bezuslovnog skoka

01	**bruncnd**

Uslovni skokovi se realizuju u onim koracima sekvence upravljačkih signala po koracima u kojima se pojavljuju iskazi tipa *br* (*if* **uslov** *then* step$_A$). Način kodiranja signala uslovnih skokova bitovima polja *cc* mikroinstrukcije, simboličke oznake signala uslovnih skokova i signal uslova koji treba da je aktivan da bi se realizovao skok dati su u narednoj tabeli:

cc	Signal uslovnog skoka	Signal uslova
02	brnotSTART	$\overline{\textbf{START}}$
03	brOBI	**OBI**
04	brnotgropr	$\overline{\textbf{gropr}}$
05	brI1	**I1**
06	brnotgradr	$\overline{\textbf{gradr}}$
07	brl2_brnch	**l2_brnch**
08	brl2_arlog	**l2_arlog**
09	brl3_jump	**l3_jump**
0A	brl3_arlog	**l3_arlog**
0B	brstore	**store**
0C	brregdir	**regdir**
0D	brnotbrpom	$\overline{brpom}$
OE	brnotprekid	$\overline{\textbf{prekid}}$
0F	brnotPRCOD	$\overline{\textbf{PRCOD}}$
10	brnotPRADR	$\overline{PRADR}$
11	brnotPRINM	$\overline{\textbf{PRINM}}$
12	brnotprintr	$\overline{\textbf{printr}}$

Višestruki uslovni skokovi se realizuju u onim koracima sekvence upravljačkih signala po koracima u kojima se pojavljuju iskazi tipa *br* (*case* (**uslov$_1$**, ..., **uslov$_n$**) *then* (**uslov$_1$**, step$_{A1}$), ..., (**uslov$_n$**, step$_{An}$). Način kodiranja signala višestrukih uslovnih skokova bitovima polja *cc* mikroinstrukcije, koraci u sekvenci upravljačkih signala po koracima u kojima se pojavljuju iskazi ovog tipa i simboličke oznake signala višestrukih uslovnih skokova dati su sledećoj tabeli:

cc	korak	Signal višestrukog uslovnog skoka
13	step$_{15}$	**bradr**
14	step$_{2F}$	**bropr**

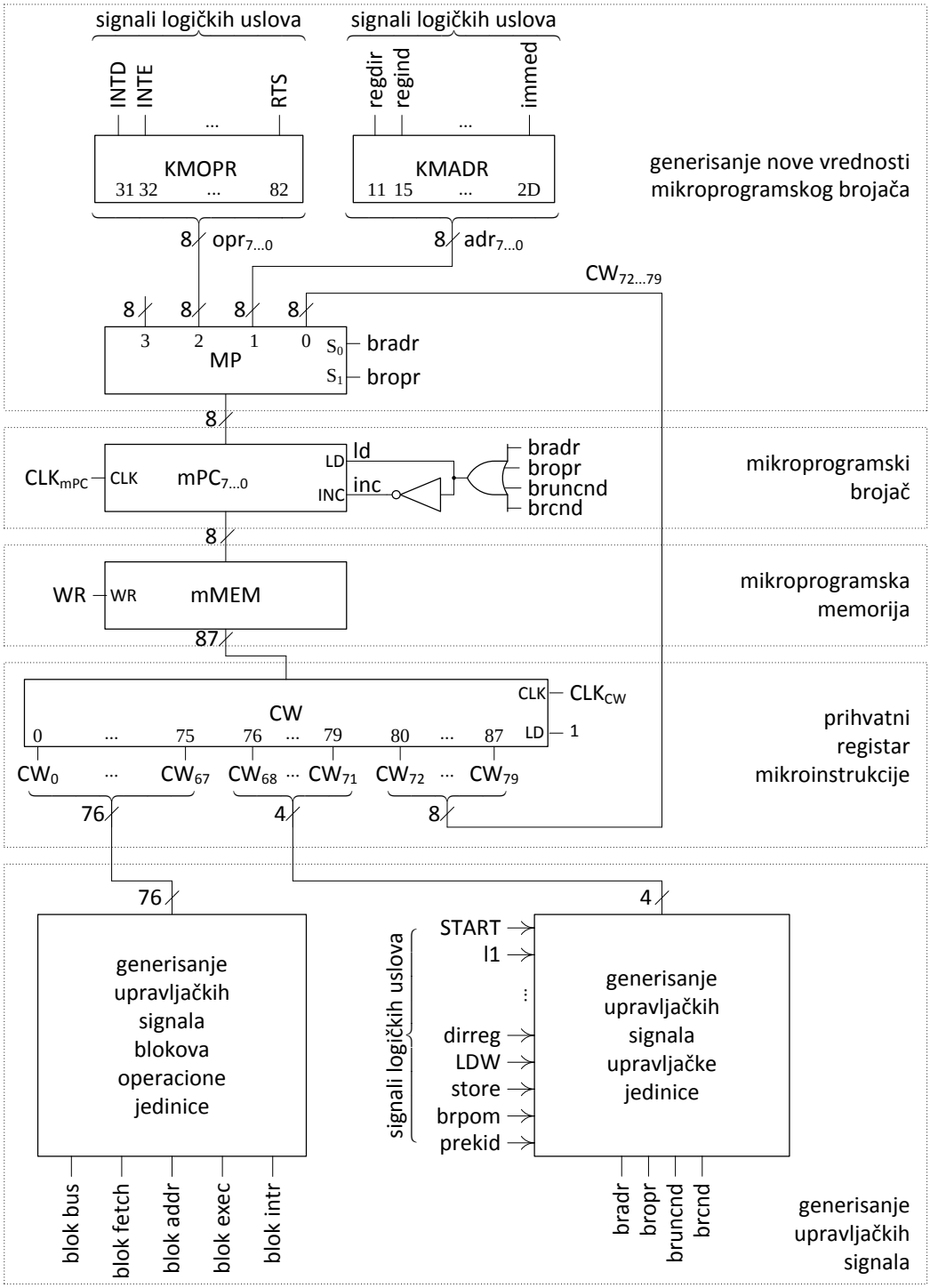
Vrednost polja *cc* 00 i sve ostale vrednosti koje nisu dodeljene signalu bezuslovnog skoka, signalima uslovnih skokova i signalima višestrukih uslovnih skokova određuje da treba preći na sledeću mikroinstrukciju.

Bitovi *ba* mikroinstrukcije koriste se za specificiranje adrese mikroinstrukcije na koju treba skočiti kod uslovnih i bezuslovnih skokova u sekvenci upravljačkih signala po koracima. Ovi bitovi sadrže vrednost koju treba upisati u mikroprogramski brojač u slučaju bezuslovnih skokova i ukoliko je signal uslova aktivan u slučaju uslovnih skokova. Kod pisanja mikroprograma ovo polje se simbolički označava sa madr$_{xx}$, pri čemu xx odgovara heksadekadnoj vrednosti ovog polja. Na primer, sa madr$_{56}$ je simbolički označena

heksadekadna vrednost 56 ovog polja. Za kodiranje polja *adresa skoka* usvojeno je 8 bitova, jer je za kompletan mikroprogram dovoljan kapacitet mikroprogramske memorije od 256 reči.

Struktura upravljačke jedinice mikroprogramske realizacije je prikazana na slici 52. Upravljačka jedinica se sastoji iz sledećih blokova: blok *generisanje nove vrednosti mikroprogramskog brojača*, blok *mikroprogramski brojač*, blok *mikroprogramska memorija*, blok *prihvatni registar mikroinstrukcije* i blok *generisanje upravljačkih signala*.

Blok *generisanje nove vrednosti mikroprogramskog brojača* se sastoji od kombinacionih mreža KMOPR i KMADR sa multiplekserom MP i služi za generisanje i selekciju vrednosti koju treba upisati u mikroprogramski brojač. Potreba za ovim se javlja kada treba odstupiti od sekvencijalnog izvršavanja mikroprograma. Vrednosti koje treba upisati u mikroprogramski brojač generišu se na tri načina i to pomoću: kombinacione mreže KMOPR koja formira signale **opr**$_{7...0}$, kombinacione mreže KMADR koja formira signale **adr**$_{7...0}$ i razreda CW$_{8...15}$ prihvatnog registra mikroinstrukcije CW. Selekcija jedne od tri grupe signala koje daju novu vrednost mikroprogramskog brojača obezbeđuje se signalima **brop** i **bradr** i to: signali **opr**$_{7...0}$ ako je aktivan signal **bropr**, signali **adr**$_{7...0}$ ako je aktivan signal **bradr** i signali CW$_{8...15}$ ako su neaktivni signali **bropr** i **bradr**.



Slika 52 Struktura upravljačke jedinice

Kombinacionom mrežom KMOPR generišu se vrednosti za realizaciju višestrukog uslovnog skoka na adresi 3C mikroprograma. U zavisnosti od toga koji od signala **HALT**, **INTD**, …, **RTS** ima aktivnu vrednost zavisi koja će od vrednosti da se pojavi na linijama **opr**$_{7...0}$. S obzirom da se na adresi 3C mikroprograma nalazi upravljačka mikroinstrukcija sa tako kodiranim poljem *cc* da njeno izvršavanje daje aktivnu vrednost signala višestrukog uslovnog skoka **bropr**, vrednost na linijama **opr**$_{7...0}$ prolazi tada kroz multiplekser MP i pojavljuje se na ulazima mikroprogramskog brojača mPC.

Kombinacionom mrežom KMADR generišu se vrednosti za realizaciju višestrukog uslovnog skoka na adresi 19 mikroprograma. U zavisnosti od toga koji od signala **regdir**, **regind**,..., **immed** ima aktivnu vrednost zavisi koja će od vrednosti da se pojavi tada na linijama **adr**$_{7...0}$. S obzirom da se na adresi 19 mikroprograma nalazi mikroinstrukcija sa tako kodiranim poljem *cc* da njeno izvršavanje daje aktivnu vrednost signala višestrukog uslovnog skoka **bradr**, vrednost na linijama **adr**$_{7...0}$ prolazi kroz multiplekser MP i pojavljuje se na ulazima mikroprogramskog brojača mPC.

Prihvatni registar mikroinstrukcije CW u svojim razredima CW$_{8...15}$ sadrži vrednost za upis u mikroprogramski brojač mPC$_{7...0}$ za bezuslovne skokove i uslovne skokove. Signali višestrukih uslovnin skokova **bropr** i **bradr** su aktivni samo prilikom izvršavanja mikroinstrukcija na adresama 19 i 3C mikroprograma, respektivno, a u svim ostalim situacijama neaktivni. S obzirom da nijedan od ova dva signala nije aktivan prilikom izvršavanja mikroinstrukcija kojima se realizuju bezuslovni ili uslovni skokovi u mikroprogramu, vrednost određena razredima CW$_{8...15}$ prolazi kroz multiplekser MP i pojavljuje se na ulazima mikroprogramskog brojača mPC$_{7...0}$.

Blok *mikroprogramski brojač* sadrži mikroprogramski brojač mPC$_{7...0}$. Mikroprogramski brojač mPC$_{7...0}$ svojom trenutnom vrednošću određuje adresu mikroprogramske memorije mMEM sa koje treba očitati mikroinstrukciju. Mikroprogramski brojač mPC$_{7...0}$ može da radi u sledećim režimima: režim inkrementiranja i režim skoka.

U režimu inkrementiranja pri pojavi signala takta **CLK$_{mPC}$** vrši se uvećavanje sadržaja mikroprogramskog brojača mPC$_{n-1...0}$ za jedan čime se obezbeđuje sekvencijalno očitavanje mikroinstrukcija iz mikroprogramske memorije. Ovaj režim rada se obezbeđuje neaktivnom vrednošću signala **ld**. Signal **ld** je neaktivan ako su svi signali **bropr**, **bradr, branch i bruncnd** neaktivni. Signali **bropr**, **bradr, branch i bruncnd** su uvek neaktivni sem kada treba obezbediti režim skoka.

U režimu skoka pri pojavi signala takta **CLK$_{mPC}$** vrši se upis nove vrednosti u mikroprogramski brojač mPC$_{n-1...0}$ čime se obezbeđuje odstupanje od sekvencijalnog očitavanja mikroinstrukcija iz mikroprogramske memorije. Ovaj režim rada se obezbeđuje aktivnom vrednošću signala **ld**. Signal **ld** je aktivan ako je jedan od signala **bropr**,**bradr, branch i bruncnd** aktivan. Jedan od signala **bropr**, **bradr, branch i bruncnd** je aktivan samo prilikom izvršavanja mikroinstrukcije koja ima takvo polje *cc* da je specificiran neki višestruki uslovni skok, bezuslovni skok ili neki od uslovnih skokova i uslov skoka je ispunjen.

Mikroprogramski brojač mPC$_{7...0}$ je dimenzionisan prema veličini mikroprograma. S obzirom da se mikroprogram svih faza izvršavanja instrukcija nalazi u opsegu od adrese 00 do adrese 96, usvojena je dužina mikroprogramskog brojača mPC$_{7...0}$ od 8 bita.

Blok *mikroprogramski memorija* sadrži mikroprogramsku memoriju mMEM, koja služi za smeštanje mikroprograma. Širina reči mikroprogramske memorije je određena dužinom mikroinstrukcija i iznosi 76 bita, a kapacitet veličinom mikroprograma svih instrukcija procesora i iznosi 256 lokacija. Adresiranje mikroprogramske memorije se realizuje sadržajem mikroprogramskog brojača mPC$_{7...0}$.

Blok *prihvatni registar mikroinstrukcije* sadrži prihvatni registar mikroinstrukcije CW$_{0...75}$. Prihvatni registar mikroinstrukcije CW$_{0...75}$ služi za prihvatanje mikroinstrukcije očitane iz mikroprogramske memorije mMEM. Na osnovu sadržaja ovog registra generišu se upravljački signali. Razredi CW$_{0...75}$ I CW$_{76...79}$ se koriste u bloku *generisanje upravljačkih signala* za generisanje upravljačkih signala operacione i upravljačke jedinice, respektivno , dok se razredi CW$_{80...87}$ koriste u bloku *generisanje nove vrednosti mikroprogramskog brojača* kao adresa skoka u mikrorogramu u slučaju bezuslovnih i uslovnih skokova. Upis u ovaj registar se realizuje signalom takta **CLK**. Signal takta **CLK** kasni za signalom takta **CLK$_{mPC}$** onoliko koliko je potrebno da se pročita sadržaj sa odgovarajuće adrese mikroprogramske memorije

Upravljački signali operacione jedinice se generišu na sledeći način :

- **outSPC = CW_1**
- **outDB = CW_4**
- **ldBH = CW_{67}**

Na identičan način se generišu i preostali upravljački signali operacione jedinice.

Upravljački signali upravljačke jedinice se generišu na sledeći način:

- **bropr = $CW_{76} \cdot CW_{77} \cdot \overline{CW_{78}} \cdot \underline{CW_{79}}$**
- **bradr = $CW_{76} \cdot CW_{77} \cdot CW_{78} \cdot CW_{79}$**
- **branch = bruncnd + brnotSTART* $\overline{\text{START}}$ + brOBI*OBI + brnotgropr* $\overline{\text{gropr}}$ + brI1*I1 + brnotgradr* $\overline{\text{gradr}}$ + brl2_brnch*l2_brnch + l2_brnch*l2_arlog + brl3_jump*l3_jump + brl3_arlog*l3_arlog + brstore*store + brregdir*regdir + brnotbrpom* $\overline{brpom}$ + brnotprekid* $\overline{\text{prekid}}$ + brnotPRCOD* $\overline{\text{PRCOD}}$ + brnotPRADR* $\overline{PRADR}$ + brnotPRINM* $\overline{\text{PRINM}}$ + brnotprintr* $\overline{\text{printr}}$**

Preostali signali se kodiraju na osnovu već date tabele, slično kao bropr i bradr.

! Čitanje instrukcije !	
$madr_{00}$	**cnt, brnotSTART, $madr_{00}$;**
$madr_{01}$	**outSPC, ldMAR, incPC;!**
$madr_{02}$	**readm; !**
$madr_{03}$	**cnt, brOBI, $madr_{03}$;**
$madr_{04}$	**outDMDR, ldIR0;!**
$madr_{05}$	**cnt, brnotgropr, $madr_{07}$;**
$madr_{06}$	**stPRCOD, !**
$madr_{07}$	**cnt, bruncnd, $madr_{85}$;**
$madr_{08}$	**cnt, brnotgradr, $madr_{09}$;**
$madr_{09}$	**stPRADR !**
$madr_{0A}$	**cnt, bruncnd, $madr_{85}$;**
$madr_{0B}$	**cnt, brl1, $madr_{15}$;**
$madr_{0C}$	**outSPC, ldMAR, incPC;!**
$madr_{0D}$	**readm; !**
$madr_{0E}$	**cnt, brOBI, $madr_{0E}$;**
$madr_{0F}$	**cnt, brl2_brnch, $madr_{2D}$;**
$madr_{10}$	**cnt, brl2_arlog, $madr_{15}$;**
$madr_{11}$	**outSPC, ldMAR, incPC;!**
$madr_{12}$	**readm; !**
$madr_{13}$	**cnt, brOBI, $madr_{13}$;**
$madr_{14}$	**outDMDR , ldIR2;!**
! Formiranje adrese i čitanje operanda !	
$madr_{15}$	**cnt, bradr**
! Registarsko direktno adresiranje !	

```
        madr 16     cnt, brstore, madr2D;
        madr 17     outDGPR, ldBL, ldBH;!
        madr 18     cnt, bruncnd, madr2D;

        ! Memorijsko direktno adresiranje !
        madr 19     outSIR21, ldMAR!
        madr 1A     cnt, brstore, madr2D;
        madr 1B     cnt, bruncnd, madr1C;


        ! Neposredno adresiranje !

        madr 1C     outDIR21, ldBL, ldBH;!
        madr 1D     cnt, bruncnd,  madr2D;

        ! Memorijsko indirektno adresiranje !
        madr 1E     outSIR21, ldMAR;!
        madr 1F     readm; !

        madr 20     cnt, brOBI, madr20;
        madr 21     outDMDR, ldBL, ldMAR;!
        madr 22     readm; !
        madr 23     outDMDR, ldBH!
        madr 24     outDB, ldMAR,!
        madr 25     cnt, brstore , madr2D;
        madr 26     cnt, bruncnd,  madr27;

        ! Čitanje operanda !

        madr 27     readm; !
        madr 28     cnt, brOBI, madr28;
        madr 29     outDMDR, mxB, ldBH, incMAR;!
        madr 2A     readm; !
        madr 2B     cnt, brOBI, madr2B;
        madr 2C     outDMDR, ldBL;!

        ! Izvršavanje operacije !
         madr 2D    cnt, bropr

        ! NOP !
        madr 2E     cnt, bruncnd, madr85;

        ! HALT !
        madr 2F     clSTART; !
        madr 30     cnt, bruncnd, madr85;
        ! INTD !
        madr 31     clPSWI!
         madr 32    cnt, bruncnd, madr85;
        ! INTE !
        madr 33     stPSWI!
        madr 34     cnt, bruncnd, madr85;
```

- ! TRPD !
- madr 35 **clPSWT**!
- madr 36 **cnt, bruncnd, madr$_{85}$;**
-
- ! TRPE !
- madr 37 **stPSWT**!
- madr 38 **cnt, bruncnd, madr$_{85}$;**
- ! LD !
-
- madr 39 **trans, ldA**,!
- madr 3A **cnt, bruncnd, madr$_{85}$;**
- ! ST !
- madr 3B **cnt, brregdir, madr$_{3C}$;**
- madr 3C **outSA, mxMDR$_0$** i **ldMDR;!**
-
-
- madr 3D **writem;!**
- madr 3E **cnt, brOBI, madr$_{3E}$;**
-
- madr 3F **outSA, mxMDR$_1$, ldMDR, incMAR**;!
- madr 40 **writem;!**
- madr 41 **cnt, brOBI, madr$_{41}$;**
- madr 42 **cnt, bruncnd, madr$_{85}$;**
- madr 43 **outSA, wrGPR!**
- madr 44 **cnt, bruncnd, madr$_{85}$;**
-
- ! STIVTP !
-
- madr 45 **outDA, ldIVTP**,!
- madr 46 **cnt, bruncnd, madr$_{85}$;**
-
- ! STSP !
-
- madr 47 **outDA, ldSP!**
- madr 48 **cnt, bruncnd, madr$_{85}$;**
-
- ! ADD !
- madr 49 **add**, **ldA**, **ldC**, **ldV**;!
- madr 4A **ldN**, **ldZ!**
- madr 4B **cnt, bruncnd, madr$_{85}$;**
-
- ! SUB !
-
- madr 4C **sub**, **ldA**, **ldC**, **ldV**;!
- madr 4D **ldN**, **ldZ**,!
- madr 4E **cnt, bruncnd, madr$_{85}$;**
- ! AND !
-
- madr 4F **and**, **ldA**;!
- madr 50 **ldN**, **ldZ**, **ldC**, **ldV**,!
- madr 51 **cnt, bruncnd, madr$_{85}$;**
- ! OR !

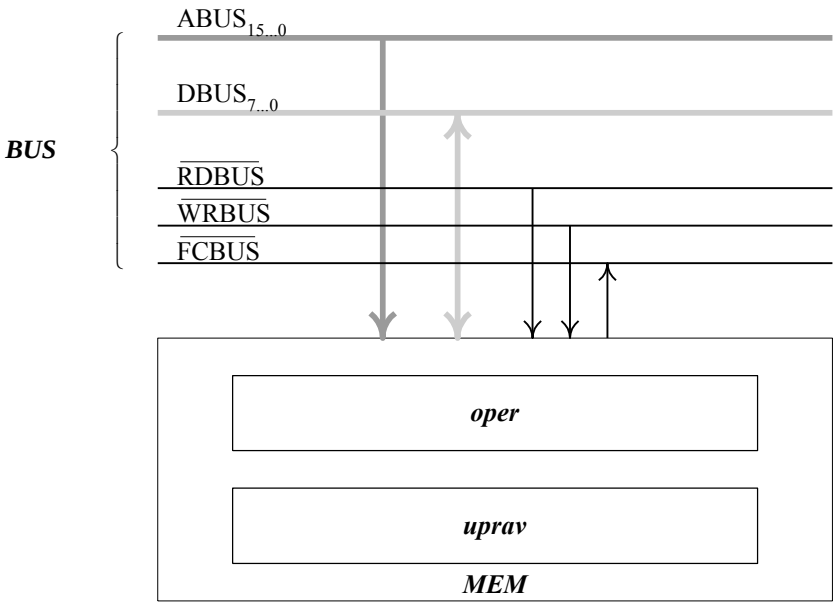
```
                madr 52         or, ldA;!
                madr 53         ldN, ldZ, ldC, ldV,!
                madr 54         cnt, bruncnd, madr85;
                ! XOR !

                 madr 55        xor, ldA;!
                madr 56         ldN, ldZ, ldC, ldV !
                madr 57         cnt, bruncnd, madr85;

                ! NOT !

                madr 58         not, ldA;!
                madr 59         ldN, ldZ, ldC, ldV!
                madr 5A         cnt, bruncnd, madr85;

                ! ASR, LSR, ROR i ROLC !
                madr 5B         shr, ldC;!
                madr 5C         ldN, ldZ, ldV!
                madr 5D         cnt, bruncnd, madr85;

                ! ASL, LSL, ROL i ROLC !
                madr 5E         shl, ldC;!
                madr 5F         ldN, ldZ, ldV!,
                madr 60         cnt, bruncnd, madr85;

                ! BEQL,…, BLSSEU !

                madr 61         outSIR0!
                madr 62         cnt, brnotbrpom, madr63;

                madr 63         outSPC, outDIR21, ldCW;!
                madr 64         outDCW, ldPC!
                madr 65         cnt, bruncnd, madr85;
                ! JMP !

                madr 66         outDIR21, ldPC,!
                madr 67         cnt, bruncnd, madr85;

                ! INT !
                madr 68         stPRINS!
                madr 69         cnt, bruncnd, madr85;

                ! JSR !
                madr 6A         outSSP, ldMAR ;!
                madr 6B         outSSP, mxMDR1, ldMDR, incSP;!

                madr 6C         writem;!
                madr 6D         cnt, brOBI, madr6D;
                madr 6E         outSSP, ldMAR;!
```

```
madr 6F     outSPC, mxMDR0, ldMDR;
madr 70     writem;
madr 71     cnt, brOBI, madr71;

madr 72     outDIR2, ldPC,!
madr 73     cnt, bruncnd, madr85;
! RTI !

madr 74     decSP;!
madr75      outSSP, ldMAR;!
madr 76     readm;!

 madr 77    cnt, brOBI, madr77;
madr 78     outDMDR, ldPSWH, outSSP ,ldMAR, decSP;!

 madr 79    readm;!
madr 7A     cnt, brOBI, madr7A;

madr 7B     outDMDR, ldPSWL;!

! RTS !
madr 7C     decSP;!
 madr 7D    outSSP, ldMAR; !
madr 7E     !readm;!
madr 7F     cnt, brOBI , madr7F;
 madr 80    outDMDR, ldBL, mxB, outSSP, ldMAR, decSP;!
madr 81     readm;!
madr 82     cnt, brOBI , madr82;
madr 83     outDMDR, ldBH;!
madr 84     outDB, ldPC; !

! Opsluživanje prekida !

madr 85     cnt, brnotprekid, madr00;

! Čuvanje konteksta procesora !

madr 86     outSSP, ldMAR ;!
madr 87     outSPC, mxMDR1, ldMDR, incSP;!
madr 88     writem;!
madr 89     cnt, brOBI , madr89;
 madr 8A    outDMDR, ldPSWH; !
madr 8B     incSP, outSSP, ldMAR; !
madr 8C     readm; !
madr 8D     cnt, brOBI , madr8D;
madr 8E     outSSP, ldMAR ;!

madr 8F     outSPSW, mxMDR1, ldMDR, incSP;!
madr 90     writem; !
madr 91     cnt, brOBI , madr91;
```

```
                madr 92         outSSP, ldMAR ;!
                madr 93         outSPSW, mxMDR0, ldMDR, incSP;!
                madr 94         writem; !
                madr 95         cnt, brOBI , madr95;

                ! Utvrđivanje broja ulaza !
                madr 96         cnt, brnotPRINS, madr99;
          madr 97  outDIR1, ldBR, clPRINS,!
                madr 98         cnt, bruncnd, madrA1;

                madr 99         cnt, brnotPRCOD, madr9C;
          madr 9A         outDIR1, ldBR, clPRCOD,!
                madr 9B         cnt, bruncnd, madrA1;

                madr 9C         cnt, brnotPRADR, madr9F;
          madr 9D         outDIR1, ldBR, clPRADR,!
                madr 9E         cnt, bruncnd, madrA3;

                madr 9F         cnt, brnotprintr, madrA2;
          madr A0         outDIR1, ldBR, clPRINM,!
                madr A1         cnt, bruncnd, madrA3;

                madr A2         mxBR1, ldBR;!


                ! Utvrđivanje adrese prekidne rutine !

                 madr A3        outSIVTP, outSIVTDSP, ldCW; !

                madr A4         outDCW, outDS, ldMAR; !
                madr A5         readm; !
                 madr A6        cnt, brOBI , madrA6;
                madr A7         outDMDR, ldBH, mxB;!
                madr A8         readm; !
                 madr A9        cnt, brOBI , madrA9;
                madr AA         outDMDR, ldBH, mxB;!
                madr AB         outDB, ldPC, clPSWI, clPSWT,!
                madr AC         cnt, bruncnd, madr00;
```

6 MEMORIJA

U ovoj glavi se daje organizacija memorije ***MEM*** (slika 53). Memorija ***MEM*** se sastoji iz operacione jedinice ***oper*** i upravljačke jedinice ***uprav***.



Slika 53 Organizacija memorije ***MEM***

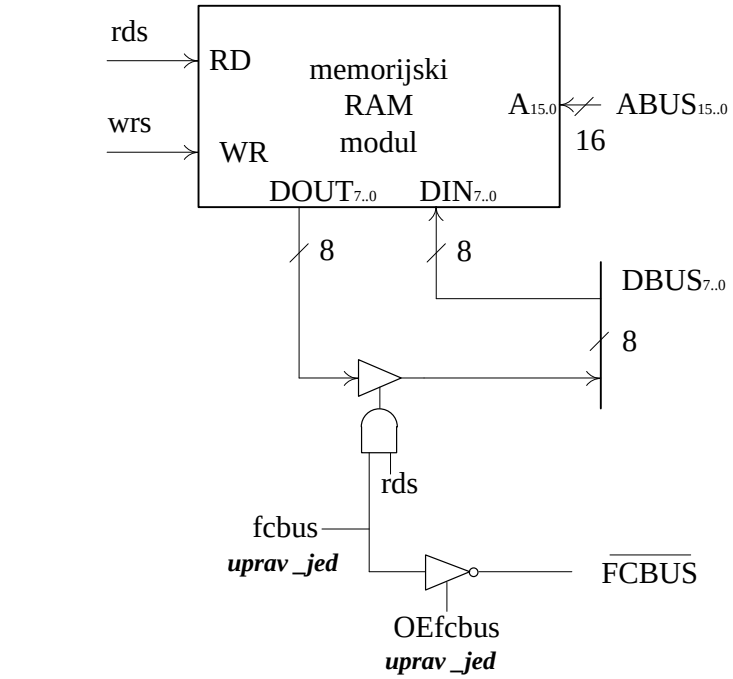
Operaciona jedinica ***oper*** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za pamćenje binarnih reči, izvršavanje mikrooperacija i generisanje signala logičkih uslova. Upravljačka jedinica ***uprav*** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za generisanje upravljačkih signala prema algoritmu generisanja upravljačkih signala operacione jedinice i signala logičkih uslova.

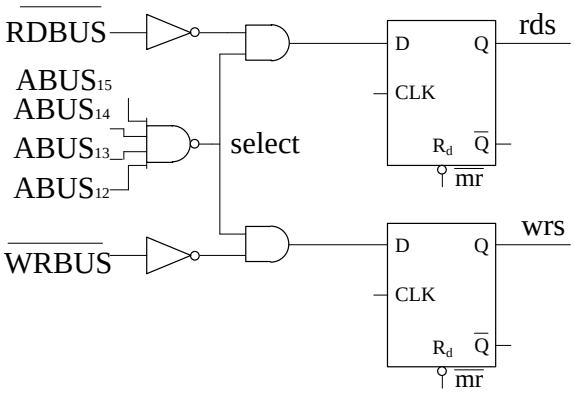
Struktura i opis operacione i upravljačke jedinice se daju u daljem tekstu.

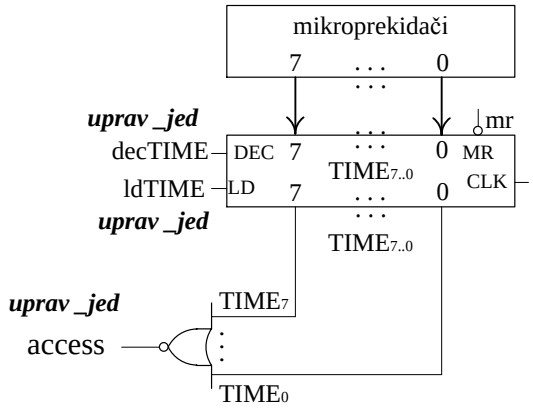
6.1 OPERACIONA JEDINICA

Operaciona jedinica ***oper_jed*** sadrži memoriju MEM, kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je memorija sluga i kombinacione i sekvencijalne mreže za realizaciju vremena pristupa memorijskom RAM modulu (slika 54).

Memorijski RAM modul služi za pamćenje (2^{16}-2^{12}) 8-mo bitnih reči. Na adresne linije $A_{15...0}$ se vodi sadržaj sa adresnih linija $ABUS_{15...0}$ magistrale ***BUS***. Na ulazne linije podataka $DIN_{7...0}$ se vodi sadržaj sa linija podataka $DBUS_{7...0}$ magistrale. Sadržaj sa izlaznih linija podataka $DOUT_{7...0}$ se preko bafera sa tri stanja vodi na linije podataka $DBUS_{7...0}$ magistrale i na njih propušta pri aktivnoj vrednosti signala **fcbus i rds**. Na linije RD i WR se vode signali **rds** i **wrs** koje generišu kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je memorija sluga. Sa memorijskim RAM modulom mogu se realizovati operacije čitanja i upisa. Sadržaj sa linija $DIN_{7...0}$ se upisuje na adresi određenoj sadržajem na linijama $A_{15...0}$ pri aktivnoj vrednosti signala na liniji WR. Sadržaj memorijske lokacije sa adrese određene sadržajem na linijama $A_{15...0}$ pojavljuje se na linijama $DOUT_{7...0}$ pri aktivnoj vrednosti signala na liniji RD.







Slika 54 Operaciona jedinica ***oper_jed***

Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je memorija sluga formiraju signale **rds** i **wrs** upravljačke jedinice ***uprav_jed*** i $\overline{\textbf{FCBUS}}$

magistrale ***BUS*** (slika 54). Signali **rds** i **wrs** se formiraju na osnovu signala $ABUS_{15..0}$ sa adresnih linija magistrale i $\overline{\textbf{RDBUS}}$ i $\overline{\textbf{WRBUS}}$ sa upravljačkih linija magistrale. Signal $\overline{\textbf{FCBUS}}$ se formira na osnovu signala **OEfcbus** i **fcbus** upravljačke jedinice ***uprav***.

Pri realizaciji ciklusa čitanja na magistrali procesor ili kontroler sa direktnim pristupom memoriji kao gazda, prvo, otvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$ i, zatim, upravljačku liniju $\overline{\textbf{RDBUS}}$ magistrale i na njih izbacuje adresu i vrednost 0 signala $\overline{\textbf{RDBUS}}$, respektivno, čime se u memoriji kao slugi startuje čitanje adresirane memorijske lokacije.

Memorija otvara bafere sa tri stanja za upravljačku liniju $\overline{\textbf{FCBUS}}$ magistrale i na nju izbacuje neaktivnu vrednost signala završetka operacije upisa u memoriji. Po završenom čitanju memorija otvara odgovarajuće bafere sa tri stanja, izbacuje sadržaj memorijske lokacije na linije podataka magistrale $DBUS_{7..0}$ i formira aktivnu vrednost upravljačkog signala završetka operacije čitanja magistrale $\overline{\textbf{FCBUS}}$. Procesor ili kontroler sa direktnim pristupom memoriji prihvata sadržaj sa linija podataka i upravljačku liniju $\overline{\textbf{RDBUS}}$ magistrale postavlja na neaktivnu vrednost. Memorija zatim zatvara bafere sa tri stanja za linije podataka $DBUS_{7..0}$ i upravljačku liniju $\overline{\textbf{FCBUS}}$ magistrale postavlja na neaktivno stanje. Na kraju procesor ili kontroler sa direktnim pristupom memoriji zatvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$, dok memorija zatvara bafere sa tri stanja za upravljačku liniju $\overline{\textbf{FCBUS}}$ magistrale.

Pri realizaciji ciklusa upisa na magistrali procesor ili kontroler sa direktnim pristupom memoriji kao gazda, prvo, otvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$, linije podataka $DBUS_{7..0}$ i upravljačku liniju $\overline{\textbf{WRBUS}}$ magistrale i na njih izbacuje adresu, podatak i vrednost 0 signala upisa, respektivno, čime se u memoriji kao slugi startuje upis u adresiranu memorijsku lokaciju.

Memorija otvara bafere sa tri stanja za upravljačku liniju $\overline{\textbf{FCBUS}}$ magistrale i na nju izbacuje neaktivnu vrednost signala završetka operacije upisa u memoriji. Po završenom upisu memorija otvara bafer sa tri stanja i upravljačku liniju $\overline{\textbf{FCBUS}}$ magistrale i na nju izbacuje aktivnu vrednost signala završetka operacije upisa u memoriji. Procesor ili kontroler sa direktnim pristupom memoriji prihvata sadržaj sa linija podataka i upravljačku liniju $\overline{\textbf{WRBUS}}$ magistrale postavlja na neaktivnu vrednost. Memorija zatim zatvara bafere sa tri stanja za upravljačku liniju $\overline{\textbf{FCBUS}}$ magistrale. Na kraju procesor ili kontroler sa direktnim pristupom memoriji zatvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$, linije podataka $DBUS_{7..0}$ i upravljačku liniju $\overline{\textbf{WRBUS}}$ magistrale.

Signali **rds** i **wrs** su upravljački signali magistrale $\overline{\textbf{RDBUS}}$ i $\overline{\textbf{WRBUS}}$ sinhronizovani na signal takta memorije. U flip-flop rds se na signal takta memorije upisuje vrednost koja odgovara vrednosti signala $\overline{\textbf{RDBUS}}$ ukoliko je aktivna vrednost signala **select**. U flip-flop wrs se na signal takta memorije upisuje vrednost koja odgovara vrednosti signala $\overline{\textbf{WRBUS}}$ ukoliko je aktivna vrednost signala **select**. Signal **rds** je kao i signal $\overline{\textbf{RDBUS}}$ aktivan za vreme operacije čitanja neke memorijske lokacije, dok je signal **wrs** kao i signal $\overline{\textbf{WRBUS}}$ aktivan za vreme operacije upisa u neku memorijsku lokaciju.

Signal **select** ima aktivnu vrednost ukoliko se na adresnim linijama $ABUS_{15...0}$ magistrale nalazi adresa iz opsega adresa dodeljenih memoriji i neaktivnu vrednost ukoliko se na adresnim linijama $ABUS_{15...0}$ magistrale nalazi adresa iz opsega adresa dodeljenih registrima po kontrolerima periferija. Celokupan opseg adresa od 0000h do FFFFh podeljen je na opseg adresa od 0000h do EFFFh, koje su dodeljene memoriji, i opseg adresa od F000h do FFFFh, koje su dodeljene registrima po svim kontrolerima periferija. Zbog toga signal **select** ima aktivnu vrednost ukoliko na adresnim linijama $ABUS_{15...12}$ magistrale nisu sve jedinice i neaktivnu vrednost ukoliko se na adresnim linijama $ABUS_{15...12}$ magistrale sve jedinice. Pri neaktivnoj vrednosti signala **select** se formiraju neaktivne vrednosti signala **rds** i **wrs** bez obzira na vrednosti signala $\overline{\textbf{RDBUS}}$ i $\overline{\textbf{WRBUS}}$, respektivno.

Kombinaciona mreža za generisanje upravljačkog signala $\overline{\textbf{FCBUS}}$ magistrale generiše ovaj signal na osnovu signala **OEfcbus** i **fcbus**. Pri neaktivnoj vrednosti signala **OEfcbus** na liniji signala $\overline{\textbf{FCBUS}}$ je stanje visoke impedance, dok je pri aktivnoj vrednosti signala **OEfcbus** na liniji signala $\overline{\textbf{FCBUS}}$ vrednost koja odgovara vrednosti signala **fcbus**. Signal **fcbus** ima neaktivnu ili aktivnu vrednost u zavisnosti od toga da li je u memorijskom modulu operacija čitanja ili upisa u toku ili je završena, respektivno.

Kombinacina i sekvencijalna mreža za realizaciju vremena pristupa memorijskom RAM modulu formira signal **access** upravljačke jedinice ***uprav_jed***. Signal **access** svojom aktivnom vrednošću određuje da je isteklo vreme pristupa. Vreme pristupa se izražava u periodama signal takta memorije i zadaje se postavljanjem mikropekidača. Po startovanju operacije čitanja ili upisa u memorijskom RAM modulu generiše se aktivna vrednost signala **ldTIME** kojim se u brojač $TIME_{7...0}$ kao početna vrednost upisuje sadržaj mikroprekidača. U opštem slučaju taj sadržaj je različit od nule, zbog čega po upisu sadržaja mikroprekidaća u brojač $TIME_{7...0}$ i signal **access** postaje neaktivan. Neaktivna vrednost signala **access** je indikacija da vreme pristupa memorijskom RAM modulu nije isteklo i da startovana operacija čitanja ili upisa u memorijskom RAM modulu nije završena. Aktivnom vrednošću signala **decTIME** se dekrementira sadržaj brojača $TIME_{7...0}$. Signal **decTIME** je aktivan i dekrementiranje brojača $TIME_{7...0}$ se nastavlja sve dok sadržaj brojača $TIME_{7...0}$ ne postane nula. Kada sadržaj brojača $TIME_{7...0}$ postane nula signal **access** postaje aktivan. Aktivna vrednost signala **access** je indikacija da je vreme pristupa memorijskom RAM modulu isteklo i da je operacija čitanja ili upisa u memorijskom RAM modulu završena.

6.2 UPRAVLJAČKA JEDINICA

U ovom odeljku se daju dijagram toka operacija, algoritam generisanja upravljačkih signala i struktura upravljačke jedinice.

6.2.1 Dijagram toka operacija

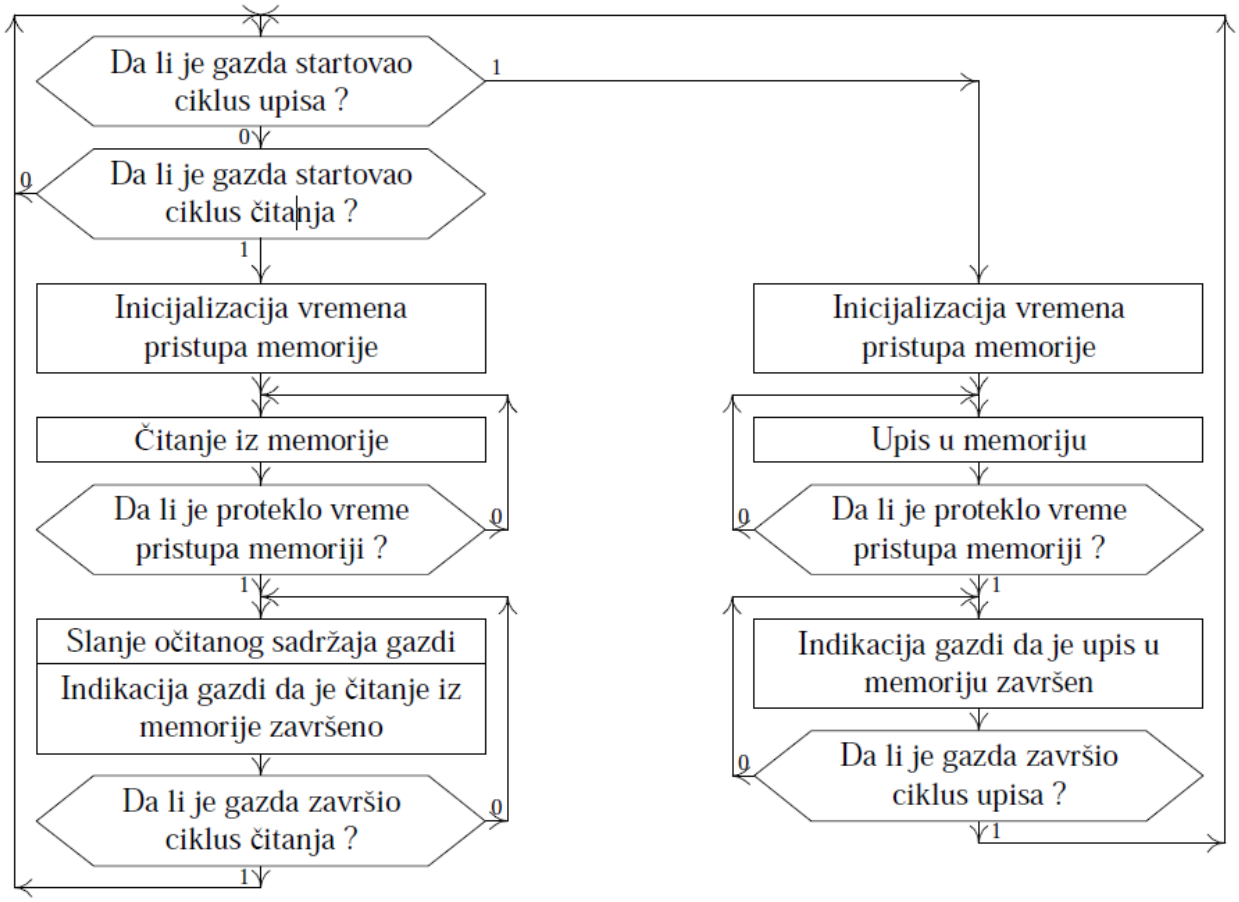
Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 55). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

U početnom koraku se vrši provera da li je procesor ili kontroler sa direktnim pristupom memoriji startovao u memoriji ciklus čitanja ili ciklus upisa, pri čemu se procesor ili kontroler sa direktnim pristupom memoriji ponaša kao gazda, a memorija kao sluga. Ako nije startovan ni ciklus čitanja ni ciklus upisa nema izvršavanja

mikrooperacija i ostaje se u početnom koraku. Ako je startovan jedan od ova dva ciklusa izvršavaju se mikrooperacije i prelazi se na sledeći korak.

Ako je startovan ciklus čitanja vrši se inicijalizacija simuliranog vremena pristupa memoriji. Potom se ulazi u petlju u kojoj se vrši čitanje iz memorije u trajanju simuliranog vremena pristupa memoriji. Po isteku vremena pristupa memoriji uzima se da je čitanje završeno, pa se izlazi iz petlje. Potom se ulazi u petlju u kojoj se, saglasno protokolu za ciklus čitanja, gazdi šalje očitani sadržaj i indikacija da je memorija završila čitanje i čeka da gazda pošalje indikaciju da je prihvatio očitani sadržaj i završio ciklus čitanja. Po dobijanju date indikacije izlazi se iz petlje i vraća na početni korak.

Ako je startovan ciklus upisa vrši se inicijalizacija simuliranog vremena pristupa memoriji. Potom se ulazi u petlju u kojoj se vrši upis u memoriju u trajanju simuliranog vremena pristupa memoriji. Po isteku vremena pristupa memoriji uzima se da je upis završen, pa se izlazi iz petlje. Potom se ulazi u petlju u kojoj se, saglasno protokolu za ciklus upisa, gazdi šalje indikacija da je memorija završila upis i čeka da gazda pošalje indikaciju da je završio ciklus upisa. Po dobijanju date indikacije izlazi se iz petlje i vraća na početni korak.

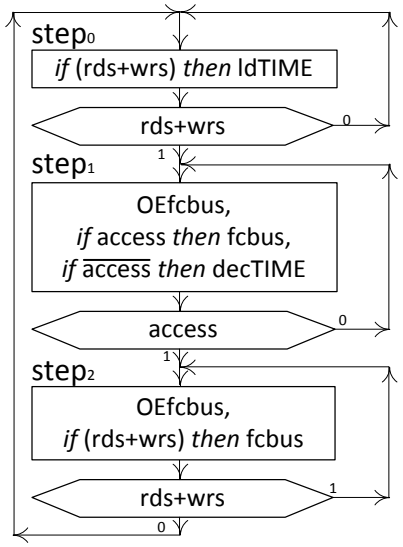


Slika 55 Dijagram toka operacija

6.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 55) i dat je u obliku dijagrama toka mikrooperacija, dijagrama toka upravljačkih signala (slika 56) i sekvence upravljačkih signala.

Dijagram toka upravljačkih je predstavljen operacionim i uslovnim blokovima (slika 55). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova.



Slika 56 Dijagram toka upravljačkih signala

U sekvenci upravljačkih signala po koracima se koriste iskazi za signale i skokove. Iskazi za signale su oblika:

signali i

if **uslov** *then* **signali**.

Prvi iskaz sadrži spisak upravljačkih signala blokova operacione jedinice **oper*_jed*** i određuje koji se signali bezuslovno generišu. Drugi iskaz sadrži uslov i spisak upravljačkih signala blokova operacione jedinice **oper*_jed*** i određuje koji signali i pod kojim uslovima treba da budu generisani

Iskazi za skokove su oblika

br (*if* **uslov** *then* step$_A$).

Ovaj iskaz sadrži uslov i korak i određuje na koji korak i pod kojim uslovima treba preći.

Objašnjenja vezana za generisanje upravljačkih signala su data zajednički za dijagram toka upravljačkih signala i sekvencu upravljačkih signala i to u okviru sekvence upravljačkih signala.

Sekvenca upravljačkih signala

! U koraku step$_0$ se ostaje i ne generiše se aktivna vrednost ni jednog od upravljačkih signala sve dok su signali **rds** i **wrs** neaktivni. Signal **rds** postaje aktivan kada processor ***CPU*** ili ulazno/izlazni uređaj ***U/I*** sa kontrolerom za direktan pristup memoriji prebaci adresne linije ABUS$_{15..0}$ iz stanja visoke impedanse na vrednost adrese memorijske lokacije i započne ciklus čitanja. Signal **wrs** postaje aktivan kada processor ili ulazno/izlazni uređaj sa kontrolerom za direktan pristup memoriji prebaci adresne linije ABUS$_{15..0}$ i linije podataka DBUS$_{7..0}$ iz stanja visoke impedanse na vrednost adrese memorijske lokacije i sadržaja za upis, respektivno, i započne ciklus upisa. Pri aktivnoj vrednosti signala **rds** ili **wrs** generiše se signal **ldTIME** i prelazi na korak step$_1$. Signalom **ldTIME** se sadržaj mikroprekidača, koji određuje vreme pristupa memorijskog RAM modula, upisuje u registar TIME$_{7...0}$. !

step$_0$ *if* (**rds**+**wrs**) *then* **ldTIME**,
br (*if* (**rds**+**wrs**) *then* step$_1$);

! U korak $step_1$ se dolazi iz koraka $step_0$, bez obzira na to da li je startovan ciklus čitanja ili upisa. U ovom koraku se generiše signal **OEfcbus**. Signalom **OEfcbus** se obezbeđuje da upravljačka linija $\overline{\mathbf{FCBUS}}$ magistrale ***BUS*** pređe iz stanja visoke impedanse na neaktivnu vrednost signala **fcbus** upravljačke jedinice ***uprav_jed***. U ovom koraku se ostaje sve vreme dok je signal **access** neaktivan. Dok je signal **access** neaktivan, generiše se aktivna vrednost signala **decTIME** i dekrementira sadržaj brojača $TIME_{7...0}$. Kada sadržaj brojača $TIME_{7...0}$ postane nula, signal **access** postane aktivan. Pri aktivnoj vrednosti signala **access** i startovanom ciklusu čitanja, na šta ukazuje aktivna vrednost signala **rds**, generiše se signal **fcbus**. Ovim signalom se očitani sadržaj sa izlaznih linija podataka $DOUT_{7...0}$ memorijskog RAM modula pušta na linije podataka $DBUS_{7..0}$. Pri aktivnoj vrednosti signala **access** se, bez obzira na to da li je startovan ciklus čitanja ili upisa, prelazi u korak $step_2$. !

$step_1$ **OEfcbus**,
if **access** *then* **fcbus**,
if $\overline{\text{access}}$ *then* **decTIME**,
br (*if* **access** *then* $step_2$);

! U korak $step_2$ se dolazi iz koraka $step_1$, bez obzira na to da li je startovan ciklus čitanja ili upisa. U ovom koraku se generiše signal **OEfcbus**, čime se obezbeđuje da se i dalje preko bafera sa tri stanja na upravljačku liniju $\overline{\mathbf{FCBUS}}$ magistrale prosleđuje vrednost signala **fcbus**. U ovom koraku se ostaje sve vreme dok je jedan od signala **rds** ili **wrs** aktivan. Dok je jedan od signala **rds** ili **wrs** aktivan, generiše se aktivna vrednost signala **fcbus**. Time je i na upravljačkoj liniji $\overline{\mathbf{FCBUS}}$ magistrale aktivna vrednost. Pri aktivnoj vrednosti signala **rds** na linije podataka $DBUS_{15...0}$ magistrale šalje se očitani sadržaj sa izlaznih linija podataka $DOUT_{7...0}$ memorijskog RAM modula. Kada signal **rds** postane neaktivan linije $DBUS_{15...0}$ prelaze u stanje visoke impedanse. Kada oba signala **rds** i **wrs** postanu neaktivni i signal **fcbus** postane neaktivan. Na liniji $\overline{\mathbf{FCBUS}}$ se pojavljuje neaktivna vrednost, jer je signal **OEfcbus** još uvek aktivan. Pri neaktivnim vrednostima oba signala **rds** i **wrs** prelazi se u korak $step_0$. Time i signal **OEfcbus** postaje neaktivan, čime se obezbeđuje da linija $\overline{\mathbf{FCBUS}}$ pređe sa neaktivne vrednosti na stanje visoke impedanse. !

$step_2$ **OEfcbus**,
if (**rds**+**wrs**) *then* **fcbus**,
br (*if* $\overline{(\text{rds} + \text{wrs})}$ *then* $step_0$);

6.2.3 Struktura upravljačke jedinice

Upravljačka jedinica ožičene realizacije generiše dve vrste upravljačkih signala i to:

- upravljačke signale operacione jedinice ***oper_jed*** i
- upravljačke signale upravljačke jedinice ***uprav_jed***.

Upravljački signali operacione jedinice ***oper_jed*** se koriste u operacionoj jedinici ***oper_jed*** radi izvršavanja mikrooperacija. Upravljački signali upravljačke jedinice ***uprav_jed*** se koriste u upravljačkoj jedinici ***uprav_jed*** radi inkrementiranja brojača koraka ili upisa nove vrednosti u brojač koraka.

Upravljački signali operacione jedinice ***oper_jed*** se mogu generisati na osnovu sekvence upravljačkih signala po koracima. Za svaki upravljački signal operacione jedinice ***oper_jed*** treba u sekvenci upravljačkih signala po koracima tražiti korake sa iskazima za signale u kojima se pojavljuje dati signal. Za svaki takav korak treba uzeti signal dekodovanog stanja brojača koraka, ukoliko se signal bezuslovno generiše, i

logički proizvod signala dekodovanog stanja brojača koraka i signala uslova, ukoliko se signal uslovno generiše, i formirati njihovu uniju.

Upravljački signali upravljačke jedinice ***uprav_jed*** se ne mogu formirati na osnovu sekvence upravljačkih signala po koracima, jer se u njoj ne pojavljuju upravljački signali upravljačke jedinice ***uprav_jed*** već samo iskazi za skok. Zbog toga je potrebno, na osnovu sekvence upravljačkih signala po koracima, najpre, formirati sekvencu upravljačkih signala za upravljačku jedinicu mešovite realizacije.

$step_A$	*br* (*if* **uslov** *then* $step_A$)	*if* **uslov** *then* (**ldCNT**, $\mathbf{bv_1}$,$\mathbf{bv_0}$)
$step_2$	*br* (*if* $\overline{(rds + wrs)}$ *then* $step_0$)	*if* $\overline{(rds + wrs)}$ *then* **ldCNT**

Signali **incCNT,** $\mathbf{bv_1}$,$\mathbf{bv_0}$

$step_A$	*br* (*if* **uslov** *then* $step_A$)	*if* **uslov** *then* (**ldCNT**, $\mathbf{bv_1}$,$\mathbf{bv_0}$)
$step_0$	*br* (*if* (**rds**+**wrs**) *then* $step_1$)	*if* (**rds**+**wrs**) *then* **incCNT**
$step_1$	*br* (*if* **access** *then* $step_2$)	*if* **access** *then* **incCNT**

Signali **incCNT**

Po opisanom postupku je, na osnovu sekvence upravljačkih signala po koracima, formirana sekvenca upravljačkih signala za upravljačku jedinicu ožičene realizacije. Ona ima sledeću formu:

- na levoj strani nalaze se dekodovani signali stanja brojača koraka,
- u sredini je iskaz tipa

signali,
if **uslov** *then* **signali**,
if **uslov** *then* (**ldCNT**, **bv1**,**bv0**) ili
if **uslov** *then* **incCNT**,

- dok komentar, u koracima gde se to radi lakšeg razumevanja smatra korisnim, uvek počinje uskličnikom (!) i proteže se do sledećeg uskličnika (!).

Iz izloženog se vidi da su upravljački signali za upravljačku jedinicu ožičene realizacije **ldCNT**, **bv1**, **bv0** i **incCNT**. Oni se formiraju na osnovu dobijene sekvence upravljačkih signala za upravljačku jedinicu ožičene realizacije na identičan način kao i upravljački signali kontrolera i operacione jedinice ***oper_jed***.

Tabela 19 Sekvenca upravljačkih signala za upravljačku jedinicu ožičene realizacije

T_0 *if* (**rds**+**wrs**) *then* **ldTIME**,
if (**rds**+**wrs**) *then* **incCNT**;
T_1 **OEfcbus**,
if **access** *then* **fcbus**,
if $\overline{access}$ *then* **decTIME**,
if **access** *then* **incCNT**;
T_2 **OEfcbus**,
if (**rds**+**wrs**) *then* **fcbus**,
if $\overline{(rds + wrs)}$ *then* **ldCNT**;

Struktura upravljačke jedinice

Struktura upravljačke jedinice je prikazana na slici 57. Upravljačka jedinica se sastoji od sledećih blokova:

· blok *brojač koraka*,

· blok *dekoder koraka* i

· blok *generisanje upravljačkih signala*.

Blok *brojač koraka* sadrži brojač $CNT_{1...0}$. Brojač $CNT_{1...0}$ svojom trenutnom vrednošću obezbeđuje aktivne vrednosti određenih upravljačkih signala. Brojač $CNT_{1...0}$može da radi u sledećim režimima:

· režim inkrementiranja,

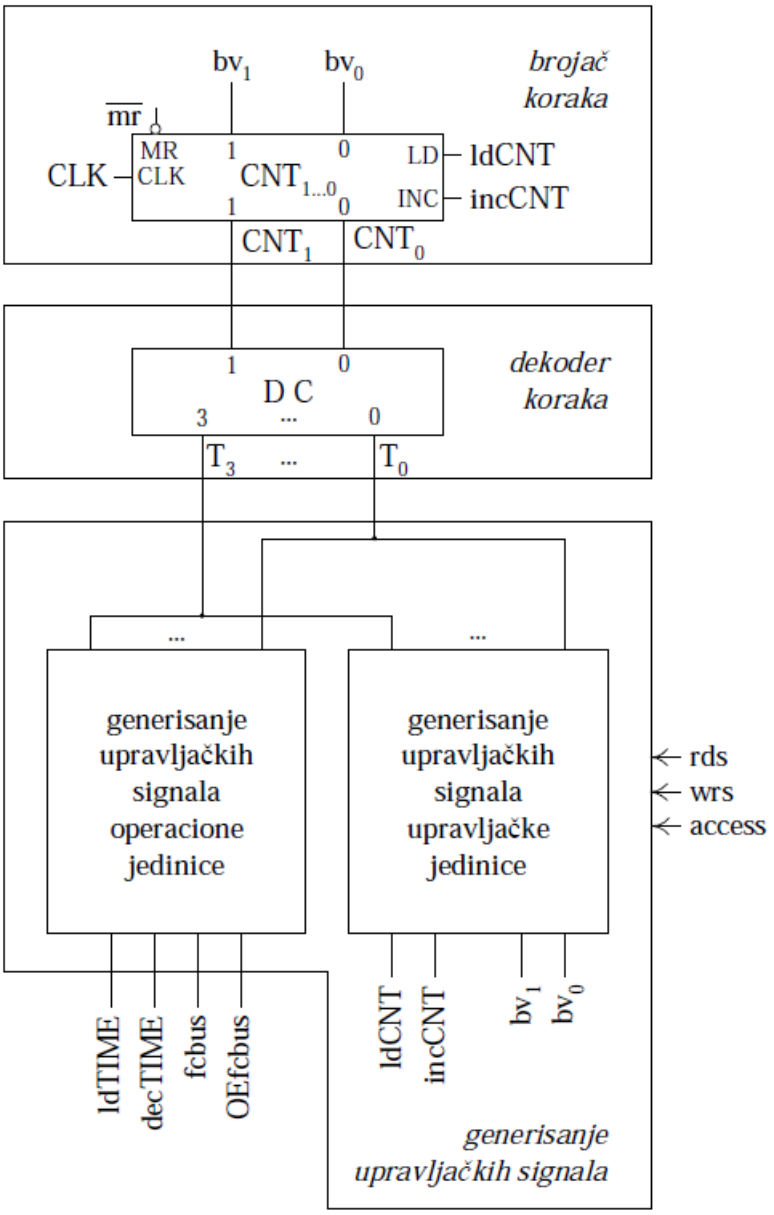
· režim skoka i

· režim čekanja.

U režimu inkrementiranja pri pojavi signala takta vrši se uvećavanje sadržaja brojača $CNT_{1...0}$ za jedan čime se obezbeđuje sekvencijalno generisanje upravljačkih signala iz sekvence upravljačkih signala za upravljačku jedinicu ožičene realizacije. Ovaj režim rada se realizuje neaktivnom vrednošću signala **ldCNT** i aktivnom vrednošću signala **incCNT.**

U režimu skoka pri pojavi signala takta vrši se upis nove vrednosti u brojač $CNT_{1...0}$ čime se obezbeđuje odstupanje od sekvencijalnog generisanja upravljačkih signala iz sekvence upravljačkih signala za upravljačku jedinicu ožičene realizacije. Ovaj režim rada se realizuje aktivnom vrednošću signala **ldCNT**, neaktivnom vrednošću signala **incCNT** i kombinacijom aktivnih i neaktivnih vrednosti signala $\mathbf{bv_1}$ i $\mathbf{bv_0}$ koja odgovara vrednosti koju treba upisati u brojač $CNT_{1...0}$.

U režimu čekanja pri pojavi signala takta ne menja se vrednost brojača $CNT_{1...0}$. Ovaj režim rada se obezbeđuje neaktivnim vrednostima signala **ldCNT** i **incCNT**.



Slika 57 Sruktura upravljačke jedinice

Blok *dekoder koraka* sadrži dekoder DC. Na ulaze dekodera DC vode se izlazi brojača $CNT_{1...0}$. Dekodovana stanja brojača $CNT_{1...0}$ pojavljuju se kao signali $\mathbf{T_0}$ do $\mathbf{T_3}$ na izlazima dekodera DC. Svakom koraku iz sekvence upravljačkih signala po koracima dodeljeno je po jedno stanje brojača $CNT_{1...0}$ određeno vrednošću signala $\mathbf{T_0}$ do $\mathbf{T_3}$ i to koraku $step_0$ signal $\mathbf{T_0}$, koraku $step_1$ signal $\mathbf{T_1}$, koraku $step_2$ signal $\mathbf{T_2}$, koraku $step_3$ signal $\mathbf{T_3}$.

Blok *generisanje upravljačkih signala* sadrži kombinacione mreže koje pomoću signala $\mathbf{T_0}$ do $\mathbf{T_3}$, koji dolaze sa bloka *dekoder koraka*, signala logičkih uslova **rds**, **wrs** i **access**, koji dolaze iz operacione jedinice ***oper_jed***., i saglasno sekvenci upravljačkih signala za upravljačku jedinicu ožičene realizacije generišu dve grupe upravljačkih signala i to:

- upravljačke signale operacione jedinice ***oper_jed*** i

- upravljačke signale upravljačke jedinice ***uprav_jed***.

Upravljački signali operacione jedinice ***oper_jed*** se generišu na sledeći način:

- **ldTIME** = (**rds** + **wrs**)*$\mathbf{T_0}$
- **OEfcbus** = $\mathbf{T_1} + \mathbf{T_2}$
- **decTIME** = $\overline{\text{access}} * \mathbf{T_1}$
- **fcbus** = **access** * $\mathbf{T_1}$ + (**rds** + **wrs**) * $\mathbf{T_2}$

Upravljački signali upravljačke jedinice ***uprav_jed*** se generiše na sledeći način:

- **ldCNT** = $\overline{\text{rds} + \text{wrs}} * \mathbf{T_2}$
- $\mathbf{bv_0} = \mathbf{0}$
- $\mathbf{bv_1} = \mathbf{0}$
- **incCNT** = (**rds** + **wrs**) * $\mathbf{T_0}$ + **access** * $\mathbf{T_1}$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz operacione jedinice ***oper_jed***. i to:

- **rds** — operaciona jedinica ***oper_jed***.,
- **wrs** — operaciona jedinica ***oper_jed***. i
- **access** — operaciona jedinica ***oper_jed***..