

JOVAN ĐORĐEVIĆ

**ARHITEKTURA I
ORGANIZACIJA
RAČUNARA**

**RAČUNARSKI SISTEM ZA RAD U
LABORATORIJI**

**ARHITEKTURA I ORGANIZACIJA
RAČUNARSKOG SISTEMA**

BGD, 2011.

SADRŽAJ

SADRŽAJ.....	I
1 KONFIGURACIJA SISTEMA.....	1
2 ARHITEKTURA SISTEMA.....	5
2.1 PROCESOR	5
2.1.1 Programski dostupni registri	5
2.1.2 Tipovi podataka	6
2.1.3 Formati instrukcija	6
2.1.3.1 Format bezadresnih instrukcija	7
2.1.3.2 Format instrukcije prekida	7
2.1.3.3 Format instrukcija relativnog skoka – B format	7
2.1.3.4 Format instrukcija apsolutnog skoka – J format.....	7
2.1.3.5 Format jednoadresnih registarskih instrukcija – R format.....	7
2.1.3.6 Format jednoadresnih neposrednih instrukcija – IB format	8
2.1.3.7 Format jednoadresnih instrukcija – IW format.....	8
2.1.3.8 Format jednoadresnih memorijskih instrukcija – AP format.....	8
2.1.4 Načini adresiranja	8
2.1.5 Skup instrukcija	10
2.1.5.1 Opis instrukcija	10
2.1.5.1.1 Instrukcija bez dejstva.....	10
2.1.5.1.2 Instrukcija zaustavljanja.....	10
2.1.5.1.3 Instrukcije skoka	10
2.1.5.1.3.1 Instrukcije uslovnog skoka	10
2.1.5.1.3.2 Instrukcija bezuslovnog skoka.....	11
2.1.5.1.3.3 Instrukcije skoka na potprogram i povratka iz potprograma.....	11
2.1.5.1.3.4 Instrukcije prekida i povratka iz prekidne rutine	11
2.1.5.1.4 Instrukcije prenosa	11
2.1.5.1.5 Aritmetičke instrukcije.....	12
2.1.5.1.6 Logičke instrukcije.....	13
2.1.5.1.7 Instrukcije pomeranja i rotiranja	13
2.1.5.1.8 Instrukcije postavljanja indikatora u PSW	14
2.1.5.2 Kodiranje instrukcija.....	14
2.1.6 Mehanizam prekida	17
2.2 MEMORIJA	20
2.3 ULAZNO/IZLAZNI UREĐAJI.....	20
2.3.1 Registri kontrolera bez direktnog pristupa memoriji.....	20
2.3.2 Registri kontrolera sa direktnim pristupom memoriji.....	22
2.3.3 Registri kontrolera za generisanje impulsa	24
3 MAGISTRALA.....	27
4 PROCESOR	31
4.1 OPERACIONA JEDINICA	31
4.1.1 Blok bus	32
4.1.2 Blok fetch	36
4.1.3 Blok addr	39
4.1.4 Blok exec.....	43
4.1.5 Blok intr	52
4.2 UPRAVLJAČKA JEDINICA.....	57
4.2.1 Dijagram toka izvršavanja instrukcija	57
4.2.2 Algoritam generisanja upravljačkih signala	59
4.2.3 Struktura upravljačke jedinice.....	94
5 MEMORIJA.....	115

5.1	OPERACIONA JEDINICA.....	115
5.2	UPRAVLJAČKA JEDINICA	117
5.2.1	<i>Dijagram toka operacija.....</i>	<i>117</i>
5.2.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>118</i>
5.2.3	<i>Struktura upravljačke jedinice.....</i>	<i>119</i>
6	ULAZNO/IZLAZNI UREĐAJI.....	121
6.1	PERIFERIJA.....	121
6.1.1	<i>Operaciona jedinica</i>	<i>121</i>
6.1.2	<i>Upravljačka jedinica.....</i>	<i>122</i>
6.1.2.1	<i>Dijagram toka operacija.....</i>	<i>122</i>
6.1.2.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>123</i>
6.1.2.3	<i>Struktura upravljačke jedinice</i>	<i>124</i>
6.2	KONROLER PERIFERIJE BEZ DIREKTOG PRISTUPA MEMORIJI.....	125
6.2.1	<i>Operaciona jedinica</i>	<i>126</i>
6.2.1.1	<i>Blok registri</i>	<i>127</i>
6.2.1.2	<i>Blok interfejs</i>	<i>129</i>
6.2.2	<i>Upravljačka jedinica.....</i>	<i>131</i>
6.2.2.1	<i>Upravljačka jedinica magistrale.....</i>	<i>132</i>
6.2.2.1.1	<i>Dijagram toka operacija</i>	<i>132</i>
6.2.2.1.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>134</i>
6.2.2.1.3	<i>Struktura upravljačke jedinice</i>	<i>137</i>
6.2.2.2	<i>Upravljačka jedinica periferije.....</i>	<i>139</i>
6.2.2.2.1	<i>Dijagram toka operacija</i>	<i>139</i>
6.2.2.2.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>142</i>
6.2.2.2.3	<i>Struktura upravljačke jedinice</i>	<i>144</i>
6.3	KONTROLER PERIFERIJE SA DIREKTNIM PRISTUPOM MEMORIJI.....	147
6.3.1	<i>Operaciona jedinica</i>	<i>148</i>
6.3.1.1	<i>Blok registri</i>	<i>149</i>
6.3.1.2	<i>Blok interfejs</i>	<i>153</i>
6.3.2	<i>Upravljačka jedinica.....</i>	<i>156</i>
6.3.2.1	<i>Upravljačka jedinica magistrale.....</i>	<i>157</i>
6.3.2.1.1	<i>Dijagram toka operacija</i>	<i>157</i>
6.3.2.1.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>159</i>
6.3.2.1.3	<i>Struktura upravljačke jedinice</i>	<i>161</i>
6.3.2.2	<i>Upravljačka jedinica periferije.....</i>	<i>164</i>
6.3.2.2.1	<i>Dijagram toka operacija</i>	<i>164</i>
6.3.2.2.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>166</i>
6.3.2.2.3	<i>Struktura upravljačke jedinice</i>	<i>168</i>
6.3.2.3	<i>Upravljačka jedinica memorije</i>	<i>171</i>
6.3.2.3.1	<i>Dijagram toka operacija</i>	<i>171</i>
6.3.2.3.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>176</i>
6.3.2.3.3	<i>Struktura upravljačke jedinice</i>	<i>182</i>
6.4	KONTROLER ZA GENERISANJE IMPULSA.....	185
6.4.1	<i>Operaciona jedinica</i>	<i>185</i>
6.4.2	<i>Upravljačka jedinica.....</i>	<i>188</i>
6.4.2.1	<i>Upravljačka jedinica magistrale.....</i>	<i>188</i>
6.4.2.1.1	<i>Dijagram toka operacija</i>	<i>188</i>
6.4.2.1.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>189</i>
6.4.2.1.3	<i>Struktura upravljačke jedinice</i>	<i>191</i>
6.4.2.2	<i>Upravljačka jedinica generisanja impulsa.....</i>	<i>192</i>
6.4.2.2.1	<i>Dijagram toka operacija</i>	<i>192</i>
6.4.2.2.2	<i>Algoritam generisanja upravljačkih signala.....</i>	<i>192</i>
6.4.2.3	<i>Struktura upravljačke jedinice</i>	<i>193</i>
7	LITERATURA.....	195

1 KONFIGURACIJA SISTEMA

Računarski sistem sadrži sledeće module: procesor **CPU**, memoriju **MEM**, ulazno/izlazne uređaje **U/I1** do **U/I7** i uređaj za kontrolu ispravnosti rada sistema **FAULT**. Procesor, memorija i ulazno/izlazni uređaji su međusobno povezani sistemskom magistralom **BUS** (slika 1). Moduli računarskog sistema rade sinhrono na isti signal takta **CLK**.

Arhitektura procesora od programski dostupnih registara ima registre opšte namene. Tipovi podataka sa kojima se radi su celobrojne 8-mo bitne veličine sa znakom i bez znaka. Format instrukcija je jednodresni. Načini adresiranja uključuju registarsko direktno, registarsko indirektno, memorijsko direktno, memorijsko indirektno, registarsko indirektno sa pomerajem, bazno indeksno sa pomerajem, PC relativno sa pomerajem i neposredno adresiranje. Skup instrukcija uključuje instrukcije prenosa, aritmetičke instrukcije, logičke instrukcije, instrukcije pomeranja i rotiranja kao i instrukcije skoka. 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 povezivanje na magistralu, čitanje instrukcije, formiranje adrese i čitanje operanda, izvršavanje operacija i opsluživanje prekida, međusobno povezanih direktnim vezama. Upravljačka jedinica je realizovana tehnikom ožičene realizacije sa brojačem koraka i dekodeom.

Procesor po linijama **intr₁** do **intr₇** dobija signale maskirajućih zahteva za prekid od ulazno/izlaznih uređaja **U/I1** do **U/I7**, respektivno. Procesor po liniji **inm** dobija signal nemaskirajućeg zahteva za prekid od uređaja za kontrolu ispravnosti rada sistema **FAULT**. Procesor po liniji **hreq** dobija signal zahteva korišćenja magistrale od ulazno/izlaznog uređaja **U/I2**, a po liniji **hack** šalje signal dozvole korišćenja magistrale ulazno/izlaznom uređaju **U/I2**.

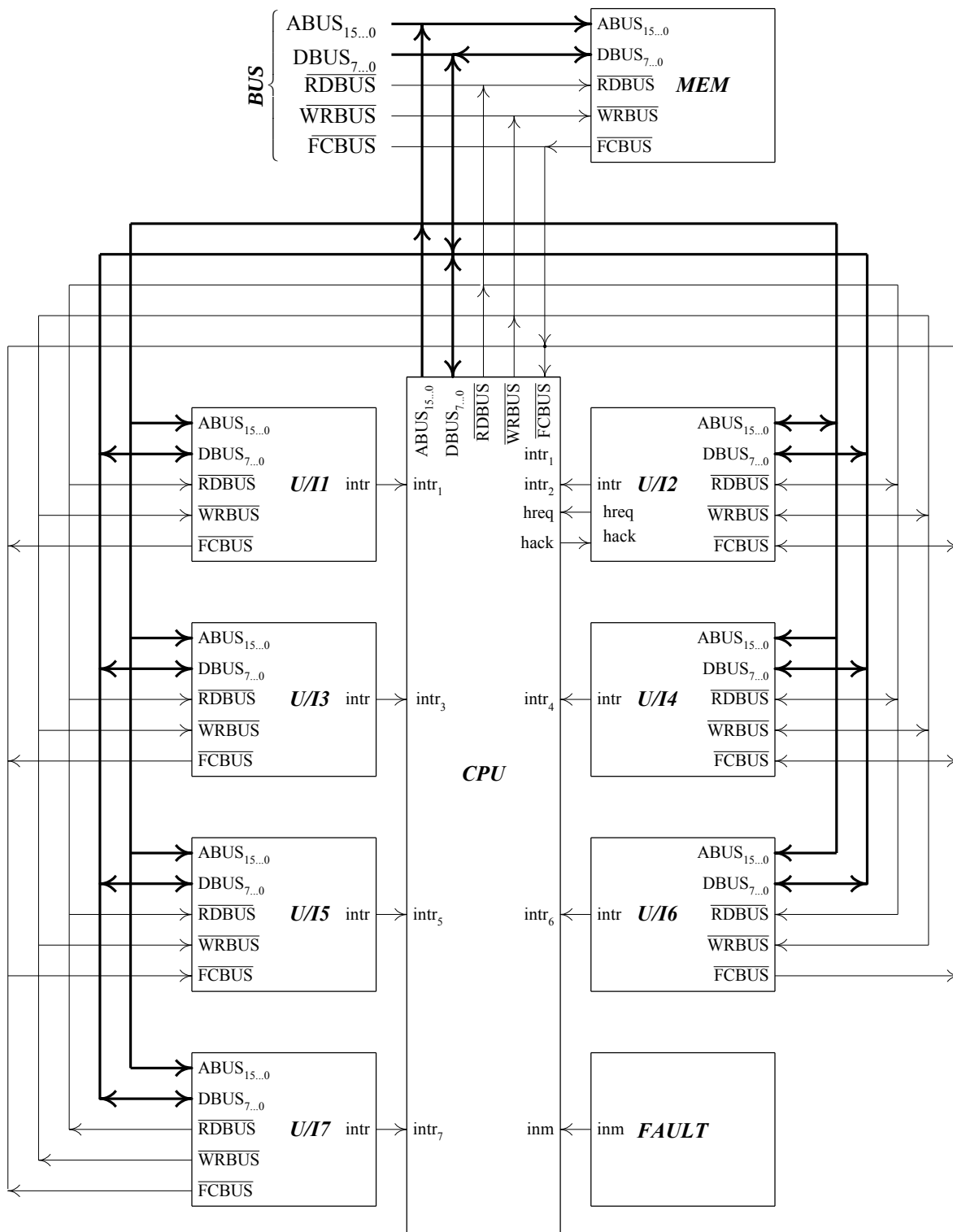
Memorija je kapaciteta 60K 8-mo bitnih reči.

Ulazno/izlaznih uređaja ima 7. Ulazno/izlazni uređaj **U/I1** se sastoji od periferije i kontrolera bez direktnog pristupa memoriji povezanih paralelnim interfejsom. Ulazno/izlazni uređaj **U/I2** se sastoji od periferije i kontrolera za direktnim pristupom memoriji povezanih paralelnim interfejsom. Ulazno/izlazni uređaj **U/I3** se sastoji od periferije i kontrolera bez direktnog pristupa memoriji povezanih serijskim interfejsom. Ulazno/izlazni uređaji **U/I4** do **U/I7** se sastoje samo od kontrolera za generisanje impulsa.

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 **RDBUS**, **WRBUS** i **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 registare 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 **RDBUS** startuje čitanje u slugi. Po završenom čitanju sluga šalje očitani podatak na linije podataka **DBUS_{7...0}** i signalom na upravljačkoj liniji **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 **WRBUS** startuje upis u slugi. Po završenom upisu sluga signalom na upravljačkoj liniji **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. Usvojen je pristup kod koga procesor ima ulogu arbitratora. Ulazno/izlazni uređaj pre realizacije ciklusa na magistrali u kome je gazda signalom **hreq** traži dozvolu korišćenja magistrale od procesora, a procesor mu signalom **hack** daje dozvolu.

2 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.

2.1 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.

2.1.1 Programski dostupni registri

Programski dostupni registri procesora su:

- programski brojač PC,
- akumulator AB,
- akumulator AW,
- 32 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 AB je 8-mo razredni akumulator za operacije prenosa, aritmetičke operacije, logičke operacije i operacije pomeranja i rotiranja nad 8-mo bitnim veličinama u kojima se koristi kao implicitno izvorište i/ili odredište operanda.

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

32 registra opšte namene su 16-to razredni registri koji se koriste kao registri podataka kod direktnog registarskog adresiranja, adresni registri kod registarskog indirektnog adresiranja, bazno/indeksni registri kod registarskog indirektnog adresiranja sa pomerajem, i bazni i indeksni registri kod bazno/indeksnog adresiranja sa pomerajem. Donjih osam razreda ovih registara se koristi kod direktnog registarskog adresiranja u operacijama za rad sa 8-mo bitnim veličinama.

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 poslednju zauzetu 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 2).

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
- L₂, L₁, L₀—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 L₂ do L₀ 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 7 do 1 koji su dodeljeni prekidima koji dolaze po linijama 7 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.

2.1.2 Tipovi podataka

Tipovi podataka koji se koriste u ovom procesoru su celobrojne 8-mo bitne veličine sa znakom i bez znaka, 8-mo bitne binarne reči i 16-to bitne binarne reči. Celobrojne 8-mo bitne veličine sa znakom i bez znaka se koriste kod operacije prenosa, aritmetičkih operacija i operacija pomeranja i rotiranja nad 8-mo bitnim veličinama, 8-mo bitne binarne reči se koriste kod logičkih operacija i operacija pomeranja i rotiranja i 16-to bitne binarne reči se koriste kod operacija prenosa.

2.1.3 Formati instrukcija

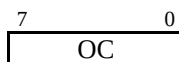
U procesoru se koriste sledeći formati instrukcija:

- 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 neposrednih instrukcija – IB format,
- format jednoadresnih neposrednih instrukcija – IW format i
- format jednoadresnih memorijskih instrukcija – AP format.

2.1.3.1 Format bezadresnih instrukcija

Format ovih instrukcija je dat na slici 3.

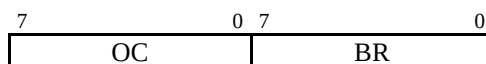


Slika 3 Format bezadresnih instrukcija

Poljem OC se specificira operacija koja se izvršava kao i registri koji se eventualno implicitno koriste.

2.1.3.2 Format instrukcije prekida

Format ove instrukcije je dat na slici 4.

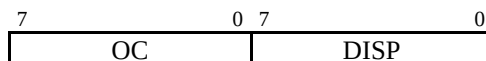


Slika 4 Format instrukcije prekida

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.

2.1.3.3 Format instrukcija relativnog skoka – B format

Format ovih instrukcija je dat na slici 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.

2.1.3.4 Format instrukcija apsolutnog skoka – J format

Format ovih instrukcija je dat na slici 6.

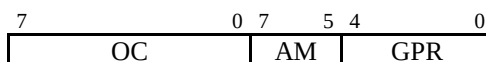


Slika 6 Format instrukcija relativnog skoka – J format

Poljem OC se specificira operacija koja se izvršava, a poljima DISPH i DISPL 8 starijih i 8 mladih bitova 16-to bitne adrese skoka.

2.1.3.5 Format jednoadresnih registarskih instrukcija – R format

Format ovih instrukcija je dat na slici 7.

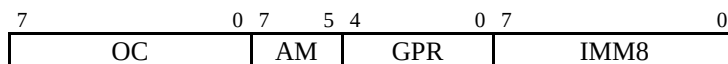


Slika 7 Format jednoadresnih registarskih instrukcija – R format

Poljem OC se specificira kod operacije jednoadresne instrukcije, polje AM registarsko direktno ili registarsko indirektno adresiranje i poljem GPR jedan od 32 registra opšte namene.

2.1.3.6 Format jednoadresnih neposrednih instrukcija – IB format

Format ovih instrukcija je dat na slici 8.

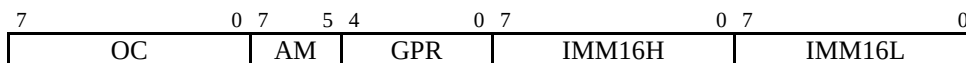


Slika 8 Format jednoadresnih instrukcija – IB format

Poljem OC se specificira kod operacije jednoadresne instrukcije nad 8-mo bitnim veličinama, polje AM neposredno adresiranje, polje GPR se ne koristi i poljem IMM8 neposredna 8-mo bitna veličina.

2.1.3.7 Format jednoadresnih instrukcija – IW format

Format ovih instrukcija je dat na slici 9.

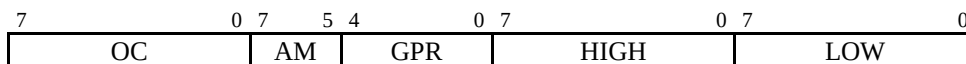


Slika 9 Format jednoadresnih instrukcija – IW format

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

2.1.3.8 Format jednoadresnih memorijskih instrukcija – AP format

Format ovih instrukcija je dat na slici 10.



Slika 10 Format jednoadresnih instrukcija – AP format

Poljem OC se specificira kod operacije jednoadresne instrukcije nad 8-mo bitnim veličinama, polje AM memorijsko direktno, memorijsko indirektno, registarsko indirektno sa pomerajem, bazno indeksno i PC relativno adresiranje. Polje GPR se ne koristi kod memorijskog direktnog i memorijsko indirektnog adresiranja dok kod registarsko indirektnog sa pomerajem, bazno indeksnog i PC relativnog adresiranje predstavlja registar opšte namene. Polja HIGH i LOW predstavljaju 8 starijih i 8 mlađih bitova 16-to bitne veličine koja predstavlja memorijsku adresu za memorijsko direktno i memorijsko indirektno adresiranje i 16-to bitni pomeraj za registarsko indirektno sa pomerajem, bazno indeksno i PC relativno adresiranje.

2.1.4 Načini adresiranja

Procesor poseduje 8 različitih načina adresiranja. Kodiranje polja AM za različite načine adresiranja i nazivi načina adresiranja dati su u tabeli 1.

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 2.1.3.5). Registar opšte namene je specificiran poljem GPR. U zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 16-to bitnim veličinama ili 8-mo bitnim veličinama koristi se svih 16 ili 8 nižih razreda registra.

Registarsko indirektno adresiranje je adresiranje kod koga se operand nalazi u memoriji na adresi određenoj sadržajem jednog od registara opšte namene. Instrukcija sa registarskim direktnim adresiranjem ima R format (poglavlje 2.1.3.5). Polje GPR specificira registar opšte namene. U zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 8-mo bitnim veličinama ili 16-to bitnim veličinama iz memorije se čita jedan ili dva bajta.

Tabela 1 Načini adresiranja

AM	Načini adresiranja
000	registarsko direktno
001	registarsko indirektno
010	memorijsko direktno
011	memorijsko indirektno
100	registarsko indirektno sa pomerajem
101	bazno indeksno sa pomerajem
110	PC relativno sa pomerajem
111	neposredno

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 2.1.3.8). Polje GPR se ne koristi. Polja HIGH i LOW sadrže starijih 8 i mlađih 8 bitova adrese. U zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 8-mo bitnim veličinama ili 16-to bitnim veličinama iz memorije se čita jedan ili dva bajta.

Memorijsko indirektno adresiranje je adresiranje kod koga se operand nalazi u memoriji na adresi određenoj sadržajem memorijske lokacije čija je adresa data u samoj instrukciji. Instrukcija sa memorijskim indirektnim adresiranjem ima AP format (poglavlje 2.1.3.8). Polje GPR se ne koristi. Polja HIGH i LOW sadrže starijih 8 i mlađih 8 bitova adrese sa koje se čita adresa operanda. U zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 8-mo bitnim veličinama ili 16-to bitnim veličinama iz memorije se čita jedan ili dva bajta.

Registarsko indirektno sa pomerajem adresiranje je adresiranje kod koga se operand nalazi u memoriji na adresi koja se dobija sabiranjem sadržaja jednog od registara opšte namene i pomeraja. Instrukcija sa registarskim indirektnim sa pomerajem adresiranjem ima AP format (poglavlje 2.1.3.8). Polje GPR specificira registar opšte namene. Polja HIGH i LOW sadrže starijih 8 i mlađih 8 bitova pomeraja. U zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 8-mo bitnim veličinama ili 16-to bitnim veličinama iz memorije se čita jedan ili dva bajta.

Bazno indeksno sa pomerajem adresiranje je adresiranje kod koga se operand nalazi u memoriji na adresi koja se dobija sabiranjem sadržaja dva registara opšte namene i pomeraja. Instrukcija sa baznim indeksnim sa pomerajem adresiranjem ima AP format (poglavlje 2.1.3.8). Polje GPR specificira adresu sa koje se čita registar opšte namene koji se koristi kao bazni registar, dok se registar opšte namene koji se koristi kao indeksni registar čita sa prve sledeće adrese. Polja HIGH i LOW sadrže starijih 8 i mlađih 8 bitova pomeraja. U zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 8-mo bitnim veličinama ili 16-to bitnim veličinama iz memorije se čita jedan ili dva bajta.

PC relativno sa pomerajem adresiranje je adresiranje kod koga se operand nalazi u memoriji na adresi koja se dobija sabiranjem sadržaja programskog brojača PC i pomeraja. Instrukcija sa PC relativnim sa pomerajem adresiranjem ima AP format (poglavlje 2.1.3.8). Polje GPR se ne koristi. Polja HIGH i LOW sadrže starijih 8 i mlađih 8 bitova pomeraja. U

zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 8-mo bitnim veličinama ili 16-to bitnim veličinama iz memorije se čita jedan ili dva bajta.

Neposredno adresiranje je adresiranje kod koga se operand nalazi u samoj instrukciji. Instrukcija sa neposrednim adresiranjem u zavisnosti od toga da li se poljem OC instrukcije specificira operacija nad 8-mo bitnim veličinama ili 16-to bitnim veličinama ima IB format (poglavlje 2.1.3.6) ili IW format (poglavlje 2.1.3.7). U oba slučaja polje GPR se ne koristi. U slučaju IB formata polje IMM8 predstavlja 8-mo bitni podatak, a u slučaju IW formata polja IMM16H i IMM16L sadrže starijih 8 i mlađih 8 bitova 16-to bitnog podatka.

2.1.5 Skup instrukcija

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

2.1.5.1 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,

2.1.5.1.1 Instrukcija bez dejstva

Instrukcija **NOP** ne proizvodi nikakvo dejstvo. Format ove instrukcije je dat u poglavlju 2.1.3.1. Ova instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

2.1.5.1.2 Instrukcija zaustavljanja

Instrukcija **HALT** zaustavlja izvršavanje instrukcija. Format ove instrukcije je dat u poglavlju 2.1.3.1. Ova instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

2.1.5.1.3 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

2.1.5.1.3.1 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 2). Pomeraj *disp* je 8-mo bitna celobrojna veličina sa znakom. Format ovih instrukcija je dat u poglavlju 2.1.3.3. 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)	$C \vee Z = 1$

2.1.5.1.3.2 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 2.1.3.4. Ova instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

2.1.5.1.3.3 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 2.1.3.4.

Instrukcija **RTS** realizuje povratak iz potprograma tako što restaurira vrednost programskog brojača PC vrednošću sa steka. Format ove instrukcije je dat u poglavlju 2.1.3.1.

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

2.1.5.1.3.4 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 2.1.3.2.

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 2.1.3.1.

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

2.1.5.1.4 Instrukcije prenosa

Instrukcija **LDB** src prenosi sadržaj 8-bitnog operanda src u akumulator AB. Kao operand src , u slučaju direktnog registarskog adresiranja, se koristi samo 8 nižih razreda adresiranog registra opšte namene. Formatu ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5, 2.1.3.6 i 2.1.3.8. Instrukcija postavlja 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.

Instrukcija **LOADW** *src* prenosi sadržaj 16-bitnog operanda *src* u akumulator AW. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5, 2.1.3.7 i 2.1.3.8. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **STB** *dst* prenosi sadržaj akumulatora AB u 8-bitni operand *dst*. Kao operand *dst*, u slučaju direktnog registarskog adresiranja, se koristi samo 8 nižih razreda adresiranog registra opšte namene. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5 i 2.1.3.8. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **STW** *dst* prenosi sadržaj akumulatora AW u 16-bitni operand *dst*. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5 i 2.1.3.8. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **POPB** prenosi sadržaj 8-bitnog operanda sa vrha steka u akumulatora AB. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija postavlja 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.

Instrukcija **POPW** prenosi sadržaj 16-bitnog operanda sa vrha steka u akumulatora AW. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **PUSHB** prenosi sadržaj akumulatora AB u 8-bitni operand na vrh steka. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **PUSHW** prenosi sadržaj akumulatora AW 16-bitni operand na vrh steka. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **LDIVTP** prenosi sadržaj registra IVTP u akumulator AW. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **STIVTP** prenosi sadržaj akumulatora AW u registar IVTP. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **LDIMR** prenosi sadržaj registra IMR u akumulator AW. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **STIMR** prenosi sadržaj akumulatora AW u registar IMR. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **LOADSP** prenosi sadržaj registra SP u akumulator AW. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

Instrukcija **STORESP** prenosi sadržaj akumulatora AW u registar SP. Format ove instrukcije dat je u poglavlju 2.1.3.1. Instrukcija ne utiče na indikatore N, Z, C i V registra PSW.

2.1.5.1.5 Aritmetičke instrukcije

Instrukcija **ADD** *src* sabira celobrojnu 8-bitnu veličinu koja se nalazi u akumulatoru AB sa operandom *src* koji je celobrojna 8-bitna veličina, a rezultat koji je celobrojna 8-bitna veličina

smešta u akumulator AB. Kao operand *src*, u slučaju direktnog registarskog adresiranja, se koristi samo 8 nižih razreda adresiranog registra opšte namene. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5, 2.1.3.6 i 2.1.3.8.

Instrukcija **SUB** *src* oduzima operand *src* koji je celobrojna 8-bitna veličina od celobrojne 8-bitne veličine koja se nalazi u akumulatoru AB, a rezultat koji je celobrojna 8-bitna veličina smešta u akumulator AB. Kao operand *src*, u slučaju direktnog registarskog adresiranja, se koristi samo 8 nižih razreda adresiranog registra opšte namene. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5, 2.1.3.6 i 2.1.3.8.

Instrukcija **INC** inkrementira celobrojnu 8-bitnu veličinu koja se nalazi u akumulatoru AB i rezultat koji je celobrojna 8-bitna veličina smešta u akumulator AB. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **DEC** dekrementira celobrojnu 8-bitnu veličinu koja se nalazi u akumulatoru AB i rezultat koji je celobrojna 8-bitna veličina smešta u akumulator AB. Format instrukcije dat je u poglavlju 2.1.3.1.

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

2.1.5.1.6 Logičke instrukcije

Instrukcija **AND** *src* izračunava logičko I 8-bitne veličine koja se nalazi u akumulatoru AB i operanda *src* koji je 8-bitna veličina, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Kao operand *src*, u slučaju direktnog registarskog adresiranja, se koristi samo 8 nižih razreda adresiranog registra opšte namene. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5, 2.1.3.6 i 2.1.3.8.

Instrukcija **OR** *src* izračunava logičko ILI 8-bitne veličine koja se nalazi u akumulatoru AB i operanda *src* koji je 8-bitna veličina, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Kao operand *src*, u slučaju direktnog registarskog adresiranja, se koristi samo 8 nižih razreda adresiranog registra opšte namene. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5, 2.1.3.6 i 2.1.3.8.

Instrukcija **XOR** *src* izračunava logičko EKSkluzivno ILI 8-bitne veličine koja se nalazi u akumulatoru AB i operanda *src* koji je 8-bitna veličina, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Kao operand *src*, u slučaju direktnog registarskog adresiranja, se koristi samo 8 nižih razreda adresiranog registra opšte namene. Formati ove instrukcije zavise od specificiranog načina adresiranja i dati su u poglavljima 2.1.3.5, 2.1.3.6 i 2.1.3.8.

Instrukcija **NOT** izračunava logičku NEGACIJU 8-bitne veličine koja se nalazi u akumulatoru AB, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Format instrukcije dat je u poglavlju 2.1.3.1.

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.

2.1.5.1.7 Instrukcije pomeranja i rotiranja

Instrukcija **ASR** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB udesno za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju razred AB₇ ostaje nepromenjen, a u indikator C registra PSW se upisuje sadržaj razreda AB₀. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **LSR** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB udesno za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju u razred AB₇ se upisuje 0, a u indikator C registra PSW se upisuje sadržaj razreda AB₀. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **ROR** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB udesno za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju u razred AB₇ se upisuje sadržaj razreda AB₀, a u indikator C registra PSW se upisuje sadržaj razreda AB₀. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **RORC** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB udesno za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju u razred AB₇ se upisuje sadržaj indikatora C registra PSW, a u indikator C registra PSW se upisuje sadržaj razreda AB₀. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **ASL** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB ulevo za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju u razred AB₀ se upisuje 0, a u indikator C registra PSW se upisuje sadržaj razreda AB₇. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **LSL** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB ulevo za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju u razred AB₀ se upisuje 0, a u indikator C registra PSW se upisuje sadržaj razreda AB₇. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **ROL** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB ulevo za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju u razred AB₀ se upisuje sadržaj razreda AB₇, a u indikator C registra PSW se upisuje sadržaj razreda AB₇. Format instrukcije dat je u poglavlju 2.1.3.1.

Instrukcija **ROLC** pomera sadržaj 8-bitne veličine koja se nalazi u akumulatoru AB ulevo za jedno mesto, a rezultat koji je 8-bitna veličina smešta u akumulator AB. Pri pomeranju u razred AB₀ se upisuje sadržaj indikatora C registra PSW, a u indikator C registra PSW se upisuje sadržaj razreda AB₇. Format instrukcije dat je u poglavlju 2.1.3.1.

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

2.1.5.1.8 Instrukcije postavljanja indikatora u PSW

Instrukcija **INTD** postavlja nulu u razred I registra PSW. Format instrukcije je dat u poglavlju 2.1.3.1.

Instrukcija **INTE** postavlja jedinicu u razred I registra PSW. Format instrukcije je dat u poglavlju 2.1.3.1.

Instrukcija **TRPD** postavlja nulu u razred T registra PSW. Format instrukcije je dat u poglavlju 2.1.3.1.

Instrukcija **TRPE** postavlja jedinicu u razred T registra PSW. Format instrukcije je dat u poglavlju 2.1.3.1.

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

2.1.5.2 Kodiranje instrukcija

Kodiranje instrukcija je izvršeno na takav način da bitovi:

- OC_{7...6} predstavljaju četiri grupe od 64 instrukcije,
- OC_{5...3} predstavljaju osam podgrupa od 8 instrukcija i
- OC_{2...0} predstavljaju instrukciju u okviru podgrupe instrukcija.

Kodiranje četiri grupe instrukcija G0 do G3 je dato u tabeli 3.

Tabela 3 Kodiranje četiri grupe instrukcija

OC _{7...6}	Oznaka	Napomena
00	G0	Koristi se
01	G1	Ne koristi se
10	G2	Ne koristi se
11	G3	Ne koristi se

Kodiranje osam podgrupa instrukcija G0_PG0 do G0_PG7 u okviru grupe instrukcija G0 je dato u tabeli 4.

Tabela 4 Kodiranje osam podgrupa instrukcija iz grupe G0

OC _{5...3}	Oznaka	Napomena
000	G0_PG0	Instrukcija bez dejstva (poglavlje 2.1.5.1.1), instrukcija zaustavljanja (poglavlje 2.1.5.1.2) i instrukcije postavljanja indikatora u PSW (poglavlje 2.1.5.1.8)
001	G0_PG1	Instrukcija bezuslovnog skoka (poglavlje 2.1.5.1.3.2), instrukcije skoka na potprogram i povratka iz potprograma (poglavlje 2.1.5.1.3.3), instrukcija prekida i povratka iz prekidne rutine (poglavlje 2.1.5.1.3.4)
010	G0_PG2	Instrukcije uslovnog skoka (poglavlje 2.1.5.1.3.1)
011	G0_PG3	Instrukcije uslovnog skoka (poglavlje 2.1.5.1.3.1)
100	G0_PG4	Instrukcije prenosa (poglavlje 2.1.5.1.4)
101	G0_PG5	Instrukcije prenosa (poglavlje 2.1.5.1.4)
110	G0_PG6	Aritmetičke instrukcije (poglavlje 2.1.5.1.5) i logičke instrukcije (poglavlje 2.1.5.1.6)
111	G0_PG7	Instrukcije pomeranja i rotiranja (poglavlje 2.1.5.1.7)

Kodiranje instrukcija u okviru osam podgrupa instrukcija G0_PG0 do G0_PG7 je dato u tabelama 5 do 12. U svakoj tabeli su date binarne vrednosti tri najmlađa bita OC_{2...0} polja koda operacije kojima se određuje instrukcija unutar date podgrupe instrukcija, oznaka za svaku instrukciju, dužina instrukcije u bajtovima i poglavlje u kome je opisana svaka instrukcija.

Kodiranje podgrupe instrukcija G0_PG0 koju čine instrukcija bez dejstva (poglavlje 2.1.5.1.1), instrukcija zaustavljanja (poglavlje 2.1.5.1.2) i instrukcije postavljanja indikatora u PSW (poglavlje 2.1.5.1.8) je dato u tabeli 5.

Tabela 5 Kodiranje podgrupe instrukcija G0_PG0

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	NOP	1	2.1.5.1.1
001	HALT	1	2.1.5.1.2
010	-	-	-
011	-	-	-
100	INTD	1	2.1.5.1.8
101	INTE	1	2.1.5.1.8
110	TRPD	1	2.1.5.1.8
111	TRPE	1	2.1.5.1.8

Kodiranje podgrupe instrukcija G0_PG1 koju čine instrukcija bezuslovnog skoka (poglavlje 2.1.5.1.3.2), instrukcije skoka na potprogram i povratka iz potprograma (poglavlje 2.1.5.1.3.3) i instrukcije prekida i povratka iz prekidne rutine (poglavlje 2.1.5.1.3.4) je dato u tabeli 6.

Tabela 6 Kodiranje podgrupe instrukcija G0_PG1

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	-	-	-
001	JMP	3	2.1.5.1.3.2
010	JSR	3	2.1.5.1.3.3
011	RTS	1	2.1.5.1.3.3
100	INT	2	2.1.5.1.3.4
101	RTI	1	2.1.5.1.3.4
110	-	-	-
111	-	-	-

Kodiranje podgrupe instrukcija G0_PG2 koju čine instrukcije uslovnog skoka (poglavlje 2.1.5.1.3.1) je dato u tabeli 7.

Tabela 7 Kodiranje podgrupe instrukcija G0_PG2

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	BEQL	2	2.1.5.1.3.1
001	BNEQL	2	2.1.5.1.3.1
010	BNEG	2	2.1.5.1.3.1
011	BNNEG	2	2.1.5.1.3.1
100	BOVF	2	2.1.5.1.3.1
101	BNOVF	2	2.1.5.1.3.1
110	BCAR	2	2.1.5.1.3.1
111	BNCAR	2	2.1.5.1.3.1

Kodiranje podgrupe instrukcija G0_PG3 koju čine instrukcije uslovnog skoka (poglavlje 2.1.5.1.3.1) je dato u tabeli 8.

Tabela 8 Kodiranje podgrupe instrukcija G0_PG3

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	BGRT	2	2.1.5.1.3.1
001	BGRTE	2	2.1.5.1.3.1
010	BLSS	2	2.1.5.1.3.1
011	BLSSE	2	2.1.5.1.3.1
100	BGRTU	2	2.1.5.1.3.1
101	BGRTEU	2	2.1.5.1.3.1
110	BLSSU	2	2.1.5.1.3.1
111	BLSSEU	2	2.1.5.1.3.1

Kodiranje podgrupe instrukcija G0_PG4 koju čine instrukcije prenosa (poglavlje 2.1.5.1.4) je dato u tabeli 9.

Tabela 9 Kodiranje podgrupe instrukcija G0_PG4

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	LDB	2, 3 ili 4	2.1.5.1.4
001	LDW	2 ili 4	2.1.5.1.4
010	STB	2 ili 4	2.1.5.1.4
011	STW	2 ili 4	2.1.5.1.4
100	POPB	1	2.1.5.1.4
101	POPW	1	2.1.5.1.4
110	PUSHB	1	2.1.5.1.8
111	PUSHW	1	2.1.5.1.8

Kodiranje podgrupe instrukcija G0_PG5 koju čine instrukcije prenosa (poglavlje 2.1.5.1.1) je dato u tabeli 10.

Tabela 10 Kodiranje podgrupe instrukcija G0_PG5

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	LDIVTP	1	2.1.5.1.6
001	STIVTP	1	2.1.5.1.6
010	LDIMR	1	2.1.5.1.6
011	STIMR	1	2.1.5.1.6
100	LDSP	1	2.1.5.1.8
101	STSP	1	2.1.5.1.8
110	-	-	-
111	-	-	-

Kodiranje podgrupe instrukcija G0_PG6 koju čine aritmetičke instrukcije (poglavlje 2.1.5.1.5) i logičke instrukcije (poglavlje 2.1.5.1.6) je dato u tabeli 11.

Tabela 11 Kodiranje podgrupe instrukcija G0_PG6

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	ADD	2, 3 ili 4	2.1.5.1.6
001	SUB	2, 3 ili 4	2.1.5.1.6
010	INC	1	2.1.5.1.6
011	DEC	1	2.1.5.1.6
100	AND	2, 3 ili 4	2.1.5.1.8
101	OR	2, 3 ili 4	2.1.5.1.8
110	XOR	2, 3 ili 4	2.1.5.1.8
111	NOT	1	2.1.5.1.8

Kodiranje podgrupe instrukcija G0_PG7 koju čine instrukcije pomeranja i rotiranja (poglavlje 2.1.5.1.7) je dato u tabeli 12.

Tabela 12 Kodiranje podgrupe instrukcija G0_PG7

OC _{2...0}	Oznaka	Dužina	Poglavlje
000	ASR	1	2.1.5.1.6
001	LSR	1	2.1.5.1.6
010	ROR	1	2.1.5.1.6
011	RORC	1	2.1.5.1.6
100	ASL	1	2.1.5.1.8
101	LSL	1	2.1.5.1.8
110	ROL	1	2.1.5.1.8
111	ROLC	1	2.1.5.1.8

2.1.6 Mehanizam prekida

Zahteve za prekid mogu da generišu:

- ① sedam kontrolera periferija po linijama intr₇ do intr₁ 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 256 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 9 do 15 u IV tabeli su rezervisani za adrese prekidnih rutina za sledeće vrste prekida:

- 0 – prekid zbog režima rada sa prekidom posle svake instrukcije,
- 1 – nemaskirajući prekid,
- 2 – prekid zbog greške u adresiranju,
- 3 – prekid zbog greške u kodu operacije,
- 9 do 15 – maskirajući prekidi po linijama intr_1 do intr_7 , respektivno.

Ulazi 4 do 8 i 15 do 255 u IV tabeli su slobodni 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 prioritetima 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.

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 ili
- ne prekida se izvršavanje prekidne rutine, već se zahtev za prekid prihvata tek po završetku prekidne rutine i povratku u prekinuti program.

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 gnežđenje prekida.

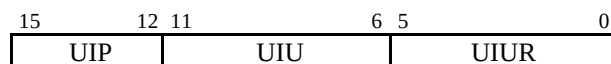
2.2 MEMORIJA

Arhitekturu memorije čine 8-mo bitne programski dostupne memorijske lokacije kapaciteta 60K reči, koje zauzimaju opseg od 0 do EFFFh memorijskog adresnog prostora. Iz memorijskih lokacija se čita i u memorijske lokacije se upisuje instrukcijama LDB i LDW (poglavljje 2.1.5.1.4), ADD i SUB (poglavljje 2.1.5.1.5), AND, OR i XOR (poglavljje 2.1.5.1.6), STB i STW (poglavljje 2.1.5.1.4) uz korišćenje registarskog indirektnog adresiranja, memorijskog direktnog adresiranja, memorijskog indirektnog adresiranja, registarskog indirektnog adresiranja sa pomerajem, bazno indeksnog adresiranja i PC relativnog sa pomerajem adresiranja radi formiranja adrese memorijske lokacije (poglavljje 2.1.5.1.4). Iz memorijskih lokacija se čita i u memorijske lokacije se upisuje instrukcijama POPB, POPW, PUSHB i PUSHW (poglavljje 2.1.5.1.4), JSR i RTS (poglavljje 2.1.5.1.3.3) i INT (poglavljje 2.1.5.1.3.4) uz korišćenje sadržaja ukazivača na vrh steka SP kao adrese memorijske lokacije (poglavljje 2.1.1).

2.3 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. Iz registara kontrolera se čita i u registre kontrolera se upisuje na identičan način na koji se čita iz memorijskih lokacija i upisuje u memorijske lokacije (poglavljje 2.2). Dozvoljeno je čitanje iz svih registara kontrolera i upisivanje u sve registre kontrolera. Adresa registra kontrolera se dobija na identičan način na koji se dobija adresa memorijske lokacije (poglavljje 2.2).

Adresiranje pojedinih registara kontrolera se obezbeđuje preko tri komponente koje formiraju adrese registara (slika 11): adresiranje ulazno-izlaznog adresnog prostora (UIP), adresiranje uređaja u ulazno-izlaznom adresnom prostoru (UIU) i adresiranje registara unutar adresiranog uređaja (UIUR).



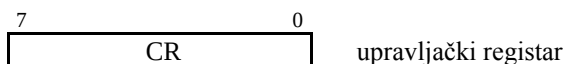
Slika 11 Formiranje adrese registara kontrolera periferije

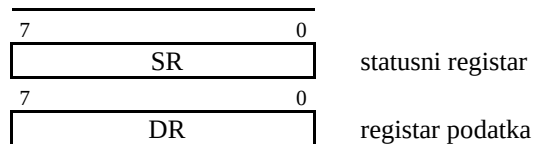
Usvojeno je da je za ulazno-izlazni adresni prostor rezervisano najviših 2^{12} adresa te je sadržaj polja UIP jednak 1111b. Ova vrednost polja UIP je ista za sve ulazno-izlazne uređaje i njihove registre. Polje UIU je širine 6 bita, što omogućava adresiranje do 2^6 ulazno-izlaznih uređaja. Ova vrednost se postavlja mikroprekidačima pri povezivanju ulazno-izlaznog uređaja na računarski sistem. Poljem UIUR se adresira do 2^6 registara unutar ulazno-izlaznog uređaja. Ovakav format adresiranja registara ulazno-izlaznih uređaja je usvojen za sve uređaje.

U ovom odeljku se razmatraju programski dostupni registri kontrolera bez direktnog pristupa memoriji, kontrolera sa direktnim pristupom memoriji i kontrolera za generisanje impulsa.

2.3.1 Registri kontrolera bez direktnog pristupa memoriji

Programski dostupni registri kontrolera su CR, SR i DR (slika 12).





Slika 12 Programski dostupni registri kontrolera bez direktnog pristupa memoriji

Registar CR je upravljački registar širine 8 bita od kojih se koriste samo najmlađa 3 bita. Ovaj registar se koristi da se programskim putem upisivanjem odgovarajućih vrednosti kontroler startuje i zadaje režima rada i da se kontroler zaustavlja. Bitovi registra CR su:

- CR₀ koji vrednošću
 - 0 ukazuje da je zadat režim rada sa izlaznom periferijom i
 - 1 ukazuje da je zadat režim rada sa ulaznom periferijom,
- CR₁ koji vrednošću
 - 0 ukazuje da zadat režim rada bez generisanjem prekida i
 - 1 ukazuje da zadat režim rada sa generisanjem prekida i
- CR₂ koji vrednošću
 - 0 ukazuje da je kontroler zaustavljen i
 - 1 ukazuje da je kontroler startovan.

Bit CR₀ se prilikom startovanja kontrolera postavlja na neaktivnu vrednost da bi se zadao režim rada sa izlaznom periferijom i na aktivnu vrednost da bi se zadao režim rada sa ulaznom periferijom. Kada se zada režim rada sa izlaznom periferijom, programskim putem se podatak najpre prenosi iz memorije u registar DR kontrolera, a potom kontroler prenosi podatak iz registra DR u izlaznu periferiju. Kada se zada režim rada sa ulaznom periferijom, kontroler najpre prenosi podatak iz ulazne periferije u registar DR, a potom se podatak prenosi programskim putem iz registra DR u memoriju. Bit CR₂ se referiše kao bit *ulaz*.

Bit CR₁ se prilikom startovanja kontrolera postavlja na neaktivnu vrednost da bi se zadao režim rada bez generisanja prekida i na aktivnu vrednost da bi se zadao režim rada sa generisanjem prekida. Režim rada bez generisanja prekida se zadaje u slučaju organizacije ulaza/izlaza sa ispitivanjem bita SR₀ statusnog registra kontrolera. Režim rada sa generisanjem prekida se zadaje u slučaju organizacije ulaza/izlaza sa prekidom. Kada prilikom ulaza ili izlaza registar DR postane raspoloživ, na šta ukazuje aktivna vrednost bita SR₀ statusnog registra kontrolera, prekid se ne generiše ukoliko je bit CR₁ neaktivan i generiše ukoliko je bit CR₁ aktivan. Bit CR₁ se referiše kao bit *prekid*.

Bit CR₂ se programskim putem postavlja na aktivnu vrednost da bi se startovao kontroler i na neaktivnu vrednost da bi se zaustavio kontroler. Kontroler koji je startovan za režim rada sa ulaznom periferijom, prenosi podatke iz periferije u registar DR, a kontroler koji je startovan za režim rada sa izlaznim uređajem prenosi podatke iz registra DR u periferiju. Kontroler koji je zaustavljen prekida prenos podataka. Bit CR₂ se referiše kao bit *start*.

Registar SR je statusni registar širine 8 bita od kojih se koristi samo najmlađi bit. Ovaj registar se koristi da se programskim putem čitanjem njegovog sadržaja dobijaju informacije o stanju u kome se nalazi kontroler. Bit statusnog registra je:

- SR₀ koji vrednošću
 - 0 ukazuje da registar DR nije raspoloživ i
 - 1 ukazuje da je registar DR raspoloživ.

Bit SR₀ se postavlja na aktivnu i neaktivnu vrednost po određenim pravilima u zavisnosti od toga da li je prilikom startovanja kontrolera zadat režim sa ulaznom ili izlaznom periferijom. Bit SR₀ se referiše kao bit *spremnost*.

Ukoliko je prilikom starovanja kontrolera u bit CR_0 upisana neaktivna vrednost, čime je zadat režim rada sa izlaznom periferijom, kontroler postavlja bit SR_0 na aktivnu vrednost koja služi kao indikacija da je registar DR raspoloživ da se programskim putem u njega upiše podatak. Prilikom upisa podatka u registar DR kontroler postavlja bit SR_0 na neaktivnu vrednost koja služi kao indikacija da registar DR više nije raspoloživ da se programskim putem u njega upiše novi podatak. Kontroler po završetku prenosa podatka iz registra DR u izlaznu periferiju postavlja bit SR_0 na aktivnu vrednost koja služi kao indikacija da je registar DR ponovo raspoloživ da se programskim putem u njega upiše podatak.

Ukoliko je prilikom starovanja kontrolera u bit CR_0 upisana aktivna vrednost, čime je zadat režim rada sa ulaznom periferijom, kontroler postavlja bit SR_0 na neaktivnu vrednost koja služi kao indikacija da registar DR nije raspoloživ da se programskim putem iz njega čita podatak. Kontroler po završetku prenosa podatka iz ulazne periferije u registar DR postavlja bit SR_0 na aktivnu vrednost koja služi kao indikacija da je registar DR raspoloživ da se programskim putem iz njega čita podatak. Prilikom čitanja podatka iz registra DR kontroler postavlja bit SR_0 na neaktivnu vrednost koja služi kao indikacija da registar DR više nije raspoloživ da se programskim putem iz njega čita podatak. Kontroler po završetku prenosa sledećeg podatka iz ulazne periferije u registar DR postavlja bit SR_0 na aktivnu vrednost koja služi kao indikacija da je registar DR ponovo raspoloživ da se programskim putem iz njega čita podatak..

Registar DR je registar podatka širine 8 bita. Ovaj registar se koristi kao prihvatni registar za podatke koje razmenjuju kontroler i procesor. Slanje podatka u izlazni uređaj se realizuje programskim putem izvršavanjem instrukcija koje upisuju podatak u registar DR. Unos podatka iz ulaznog uređaja se realizuje programskim putem izvršavanjem instrukcija koje čitaju podatak iz registra DR.

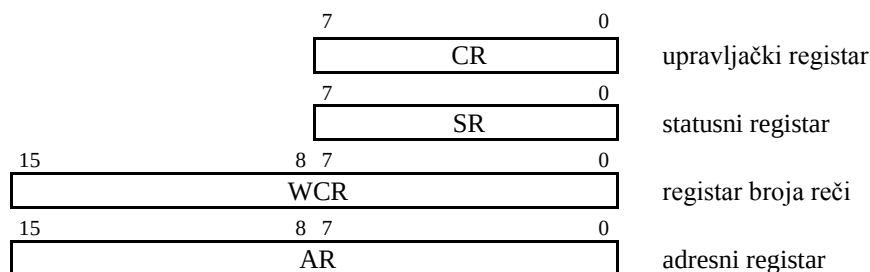
U tabeli 13 su date relativne adrese svih programski dostupnih registara kontrolera bez direktnog pristupa memoriji u okviru opsega od 64 adrese datog kontrolera.

Tabela 13 Relativne adrese registara kontrolera bez direktnog pristupa memoriji

Registar	Adresa
CR	0
SR	1
DR	2

2.3.2 Registri kontrolera sa direktnim pristupom memoriji

Programski dostupni registri kontrolera su CR, SR, WCR i AR (slika 13).



Slika 13 Programski dostupni registri kontrolera sa direktnim pristupom memoriji

Registar CR je upravljački registar širine 8 bita od kojih se koriste samo najmlađa 4 bita. Ovaj registar se koristi da se programskim putem upisivanjem odgovarajućih vrednosti kontroler startuje i zadaje režima rada i da se kontroler zaustavlja. Bitovi registra CR su:

- CR_0 koji vrednošću

- 0 ukazuje da je zadat režim prenosa memorija – izlazna periferija i
- 1 ukazuje da je zadat režim prenosa ulazna periferija - memorija,
- CR₁ koji vrednošću
 - 0 ukazuje da zadat režim rada bez generisanjem prekida i
 - 1 ukazuje da zadat režim rada sa generisanjem prekida,
- CR₂ koji vrednošću
 - 0 ukazuje da je kontroler zaustavljen i
 - 1 ukazuje da je kontroler startovan i
- CR₃ koji vrednošću
 - 0 ukazuje da je zadat režim rada sa pojedinačnim prenosom i
 - 1 ukazuje da je zadat režim rada sa paketskim prenosom.

Bit CR₀ se prilikom startovanja kontrolera i zadavanja režima rada postavlja na neaktivnu vrednost da bi se zadao režim prenosa memorija – izlazna periferija i na aktivnu vrednost da bi se zadao režim prenosa ulazna periferija – memorija. Kada se zada režim prenosa memorija – izlazna periferija kontroler samostalno prenosi podatke iz memorije u izlaznu periferiju, a kada se zada režim prenosa ulazna periferija – memorija kontroler samostalno prenosi podatke iz ulazne periferije u memoriju. Bit CR₀ se referiše kao bit *ulaz*.

Bit CR₁ se prilikom startovanja kontrolera i zadavanja režima postavlja na neaktivnu vrednost da bi se zadao režim rada bez generisanja prekida i na aktivnu vrednost da bi se zadao režim rada sa generisanjem prekida. Režim rada bez generisanja prekida se zadaje u slučaju organizacije prenosa podataka sa ispitivanjem bita SR₀ statusnog registra kontrolera. Režim rada sa generisanjem prekida se zadaje u slučaju organizacije prenosa podataka sa prekidom. Kada se obavi kompletan prenos podataka bit SR₀ statusnog registra kontrolera se postavlja na aktivnu vrednost. Tom prilikom se prekid ne generiše ukoliko je bit CR₁ neaktivan i generiše ukoliko je bit CR₁ aktivan. Bit CR₁ se referiše kao bit *prekid*.

Bit CR₂ se programskim putem postavlja na aktivnu vrednost da bi se startovao kontroler i na neaktivnu vrednost da bi se zaustavio kontroler. Kontroler koji je startovan za režim rada sa ulaznom periferijom, prenosi podatke iz periferije u memoriju, a kontroler koji je startovan za režim rada sa izlaznom periferijom prenosi podatke iz memorije u periferiju. Kontroler koji je zaustavljen prekida prenos podataka. Bit CR₂ se referiše kao bit *start*.

Bit CR₃ se prilikom startovanja kontrolera postavlja na neaktivnu vrednost da bi se zadao režim rada sa pojedinačnim prenosom i na aktivnu vrednost da bi se zadao režim rada sa paketskim prenosom. Režim rada sa pojedinačnim prenosom se normalno koristi prilikom rada sa sporim ulazno/izlaznim uređajima. U ovom slučaju kontroler za svaki podatak traži magistralu od procesora, po dobijanju magistrale prenosi jedan podatak i potom vraća magistralu procesoru. Režim rada sa paketskim prenosom se normalno koristi prilikom rada sa brzim ulazno/izlaznim uređajima. U ovom slučaju kontroler traži magistralu od procesora samo jedanput pre prenosa bloka podataka, po dobijanju magistrale prenosi ceo blok podataka i na kraju prenosa celog bloka podataka vraća magistralu procesoru. Bit CR₂ se referiše kao bit *paket*.

Registar SR je statusni registar širine 8 bita od kojih se koriste samo najmlađi bit SR₀. Ovaj registar se koristi da se programskim putem čitanjem njegovog sadržaja dobijaju informacije o stanju u kome se nalazi kontroler. Bit statusnog registra je:

- SR₀ koji vrednošću
 - 0 ukazuje da prenos podataka nije obavljen do kraja i
 - 1 ukazuje da je prenos podataka obavljen do kraja.

Bit SR_0 se postavlja na neaktivnu vrednost prilikom startovanja kontrolera i na aktivnu vrednost kada kontroler obavi prenos podataka do kraja. Bit SR_0 se referiše kao bit *kraj*.

Registar WCR je registar broja reči koje treba preneti širine 16 bita. Pre startovanja i zadavanja režima rada kontrolera potrebno je u okviru inicijalizacije kontrolera upisati u registar WCR sadržaj koji predstavlja veličinu bloka podataka koji treba preneti. Prilikom prenosa svakog podatka kontroler između ostalog dekrementira sadržaj registra WCR. Sadržaj nula registra WCR je indikacija da je obavljen kompletan prenos bloka podataka što se registruje postavljanjem bita SR_0 na aktivnu vrednost.

Registar AR je adresni registar širine 16 bita koji se koristi izvorišni adresni registar za adresiranje izvorišnog podatka u memoriji u režimu prenosa memorija–izlazni periferija i odredišni adresni registar za adresiranje odredišnog podatka u memoriji u režimu prenosa ulazna periferija – memorija. Pre startovanja i zadavanja režima rada kontrolera potrebno je u okviru inicijalizacije kontrolera upisati u registar AR početnu adresu memorije odakle treba krenuti sa očitavanjem podataka u režimu prenosa memorija–izlazni periferija i upisivanjem podataka u režimu prenosa ulazna periferija – memorija. Sadržaj registra AR se po prenosu svakog podatka inkrementira. Time se blok podataka prenosi iz susednih memorijskih lokacija počev od memorijske lokacije čija je adresa određena sadržajem registra AR u izlaznu periferiju ili iz ulazne periferije u susedne memorijske lokacije počev od memorijske lokacije čija je adresa određena sadržajem registra AR onoliko puta koliko je to određeno sadržajem registra WCR.

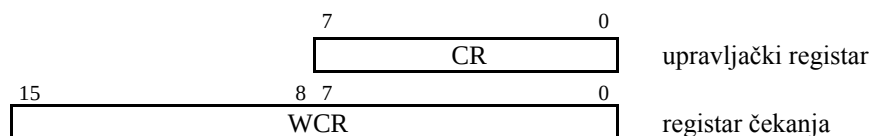
U tabeli 14 su date relativne adrese svih programski dostupnih registara kontrolera sa direktnim pristupom memoriji u okviru opsega od 64 adrese datog kontrolera.

Tabela 14 Relativne adrese registara kontrolera sa direktnim pristupom memoriji

Registar	Adresa
CR	0
SR	1
$WCR_{15...8}$	4
$WCR_{7...0}$	5
$AR_{15...8}$	6
$AR_{7...0}$	7

2.3.3 Registri kontrolera za generisanje impulsa

Programski dostupni registri kontrolera su CR i WCR (slika 14).



Slika 14 Programski dostupni registri kontrolera za generisanje impulsa

Registar CR je upravljački registar širine 8 bita od kojih se koristi samo bit CR_2 . Ovaj registar se koristi da se programskim putem upisivanjem odgovarajućih vrednosti kontroler startuje i da se kontroler zaustavlja. Bit registra CR je:

- CR_2 koji vrednošću
 - 0 ukazuje da je kontroler zaustavljen i
 - 1 ukazuje da je kontroler startovan.

Bit CR_2 se programskim putem postavlja na aktivnu vrednost da bi se startovao kontroler i na neaktivnu vrednost da bi se zaustavio kontroler. Kontroler koji je startovan za režim rada sa ulaznom periferijom, prenosi podatke iz periferije u memoriju, a kontroler koji je startovan za

režim rada sa izlaznom periferijom prenosi podatke iz memorije u periferiju. Kontroler koji je zaustavljen prekida prenos podataka. Bit CR₂ se referiše kao bit *start*.

Registar WCR je 16-to razredni registar čekanja u kome se čuva vreme, zadato kao broj perioda signala takta, koje treba da protekne od trenutka startovanja kontrolera do trenutka generisanja signala prekida trajanja jedna perioda signala takta.

U tabeli 15 su date relativne adrese svih programski dostupnih registara kontrolera sa direktnim pristupom memoriji u okviru opsega od 64 adrese datog kontrolera.

Tabela 15 Relativne adrese registara kontrolera kontrolera za generisanje impulsa

Registar	Adresa
CR	0
WCR _{15...8}	2
WCR _{7...0}	3

3 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.

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 upisuju 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 sluga i to ciklus čitanja i ciklus upisa. Za njihovu realizaciju koriste se tri grupe linija i to adresne linije **ABUS_{15...0}**, linije podataka **DBUS_{7...0}** i upravljačke linije **RDBUS**, **WRBUS** i **FCBUS**. Po adresnim linijama **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 **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 **RDBUS** gazda šalje signal kojim u slugi startuje ciklus čitanja. Po upravljačkoj liniji **WRBUS** gazda šalje signal kojim u slugi startuje ciklus upisa. Po upravljačkoj liniji **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 **DBUS_{7...0}** nalazi sadržaj koji gazda treba da upiše u svoj prihvatni registar.

Gazde na magistrali modu da budu procesor i uređaj sa direktnim pristupom memoriji. Kako je dozvoljeno da u jednom trenutku ili procesor ili uređaj bude gazda magistrale, potrebno je pre svakog ciklusa na magistrali realizovati arbitraciju pristupa magistrali između procesora i uređaja. Arbitracija je tako izvedena da procesor ima funkciju arbitratora i da je magistrala normalno u posedu procesora. S toga uređaj svaki put kada kao gazda treba da započne ciklus čitanja ili upisa na magistrali najpre od procesora traži dozvolu korišćenja magistrale.

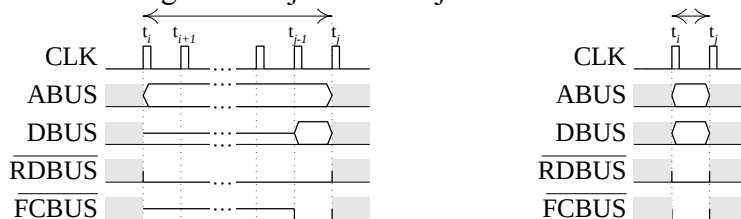
Uređaj traži od procesora dozvolu korišćenja magistrale postavljanjem signala **hreq** na vrednost 1. Ovde se mogu javiti dve situacije. Prva situacija se javlja ako procesor nije već započeo ciklus na magistrali. Tada je magistrala slobodna i procesor može odmah, postavljanjem signala **hack** na vrednost 1, da uređaju da dozvolu da kao gazda započne svoj ciklus. Druga situacija se javlja ako je procesor već započeo ciklus na magistrali. Tada

magistrala nije slobodna i procesor mora najpre da završi započeti ciklus na magistrali pa da tek onda, postavljanjem signala **hack** na vrednost 1, da uređaju dozvolu da kao gazda započne svoj ciklus.

Kada je procesor dao magistralu uređaju mogu se javiti dve situacije. Prva situacija se javlja ako u toku ciklusa na magistrali u kojoj je uređaj gazda procesor nema potrebu da koristi magistralu. U ovoj situaciji uređaj produžava sa svojim ciklusom, a završetak svog ciklusa signalizira procesoru postavljanjem signala **hreq** na vrednost 0. Tada procesor postavlja signal dozvole **hack** na vrednost 0. Ovim je magistrala vraćena procesoru. Druga situacija se javlja ako u toku ciklusa na magistrali u kojoj je uređaj gazda procesor ima potrebu da koristi magistralu. I u ovoj situaciji uređaj produžava sa svojim ciklusom, pri čemu procesor ne sme da započne svoj ciklus već mora da čeka sve dok završetak svog ciklusa uređaj ne signalizira procesoru postavljanjem signala **hreq** na vrednost 0. Tada procesor postavlja signal dozvole **hack** na vrednost 0. Ovim je magistrala vraćena procesoru. S toga procesor sada može da krene sa realizacijom svog ciklusa na magistrali.

Svi moduli računarskog sistema su povezani na adresne linije, linije podatak 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}**, **RDBUS** i **FCBUS** koje na magistrali razmenjuju gazda i sluga prilikom realizacije ciklusa čitanja dati su na slici 15. 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. Uz vremenske oblike signala na magistrali dat je i signal takta CLK koji je zajednički za gazdu i slugu. Signali su dati za vremenski period od takta t_i do takta t_{j-1} u kome jedan modul kao gazda sa drugim modulom kao slugom realizuje ciklus čitanja. Van datog vremenskog perioda signali su osenčeni. Tada je magistrala ili slobodna i sve linije magistrale su u stanju visoke impedanse ili zauzeta od strane neka dva modula i vrednosti signala na linijama magistrale odgovaraju trenutnom stanju ciklusa na magistrali koji se realizuje između ta dva modula.



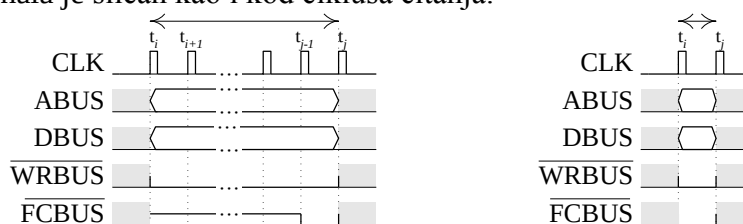
Slika 15 Vremenski oblici signala za ciklus čitanja

Po dobijanju magistrale gazda započinje ciklus čitanja tako što na signal takta t_i otvara bafere sa tri stanja preko kojih šalje adresu na adresne linije magistrale **ABUS_{15...0}** i postavlja upravljački signal **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 **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 na signal takta t_{j-1} otvara bafere sa tri stanja preko kojih šalje očitani sadržaj na linije podataka magistrale **DBUS_{7...0}** i postavlja upravljački signal **FCBUS** na vrednost 0, dok na signal takta t_j zatvara bafere sa tri stanja, pa linije podataka magistrale **DBUS_{7...0}** i

upravljačka linija magistrale $\overline{\text{FCBUS}}$ prelaze u stanje visoke impedanse. Po detektovanju vrednosti 0 signala $\overline{\text{FCBUS}}$ gazda na signal takta t_j upisuje sadržaj sa linija podataka magistrale $\text{DBUS}_{7...0}$ u svoj prihvatni registar i zatvara svoje bafere sa tri stanja, pa adresne linije magistrale $\text{ABUS}_{15...0}$ i upravljačka linija magistrale $\overline{\text{RDBUS}}$ prelaze u stanje visoke impedanse. Time je ciklus čitanja na magistrali završen.

Vremenski oblici signala na slici 15 levo odgovaraju ciklusu čitanja iz memorijskih lokacija za koje je uzeto da vreme pristupa traje određen broj perioda signala takta. U slučaju ciklusa čitanja iz registara kontrolera ulazno/izlaznih uređaja uzeto je da vreme pristupa traje samo jednu perioda signala takta, pa se vremenski oblici signala sa slike 15 levo redukuju na vremenske oblike signala sa slike 15 desno.

Vremenski oblici signala $\text{ABUS}_{15...0}$, $\text{DBUS}_{7...0}$, $\overline{\text{WRBUS}}$ i $\overline{\text{FCBUS}}$ koje na magistrali razmenjuju gazda i sluga prilikom realizacije ciklusa upisa dati su na slici 16. Način predstavljanja signala je sličan kao i kod ciklusa čitanja.



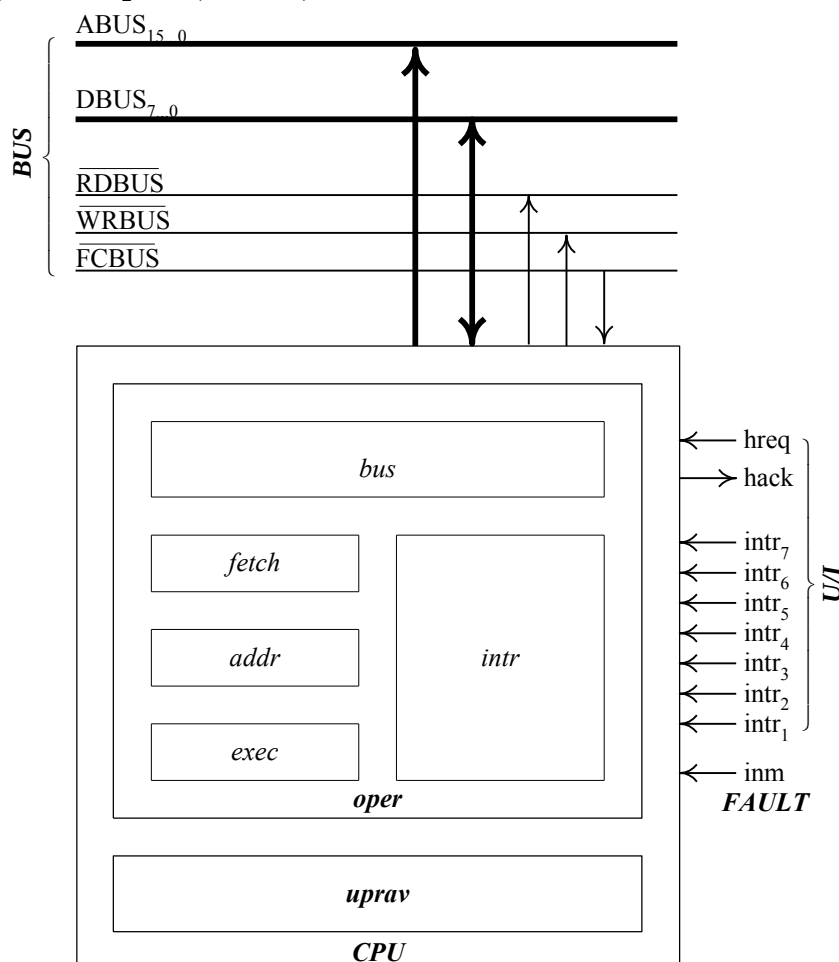
Slika 16 Vremenski oblici signala za ciklus upisa

Po dobijanju magistrale gazda započinje ciklus upisa tako što na signal takta t_i otvara bafere sa tri stanja preko kojih šalje adresu na adresne linije magistrale $\text{ABUS}_{15...0}$, podatak na linije podataka magistrale $\text{DBUS}_{7...0}$ i postavlja upravljački signal $\overline{\text{WRBUS}}$ na vrednost 0. Sve sluge dekoduju sadržaj sa adresnih linija magistrale $\text{ABUS}_{15...0}$ i samo u slugi koji je sadržaj sa adresnih linija $\text{ABUS}_{15...0}$ prepoznao kao svoju adresu upravljački signal $\overline{\text{WRBUS}}$ startuje ciklus upisa sadržaja sa linija podataka magistrale $\text{DBUS}_{7...0}$. 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 na signal takta t_{j-1} otvara bafer sa tri stanja preko koga postavlja upravljački signal $\overline{\text{FCBUS}}$ na vrednost 0, dok na signal takta t_j zatvara bafer sa tri stanja, pa upravljačka linija magistrale $\overline{\text{FCBUS}}$ prelazi u stanje visoke impedanse. Po detektovanju vrednosti 0 signala $\overline{\text{FCBUS}}$ gazda na signal takta t_j zatvara svoje bafere sa tri stanja, pa adresne linije magistrale $\text{ABUS}_{15...0}$, linije podataka magistrale $\text{DBUS}_{7...0}$ i upravljačka linija magistrale $\overline{\text{WRBUS}}$ prelaze u stanje visoke impedanse. Time je ciklus upisa na magistrali završen.

Vremenski oblici signala na slici 16 levo odgovaraju ciklusu upisa u memorijske lokacije za koje je uzeto da vreme pristupa traje određen broj perioda signala takta. U slučaju ciklusa upisa u registre kontrolera ulazno/izlaznih uređaja uzeto je da vreme pristupa traje samo jednu perioda signala takta, pa se vremenski oblici signala sa slike 16 levo redukuju na vremenske oblike signala sa slike 16 desno.

4 PROCESOR

U ovoj glavi se daje organizacija procesora **CPU** koji se sastoji iz operacione jedinice **oper** i upravljačke jedinice **uprav** (slika 17).



Slika 17 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 se daju u daljem tekstu.

4.1 OPERACIONA JEDINICA

Operaciona jedinica **oper** (slika 17) se sastoji od sledećih blokova:

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

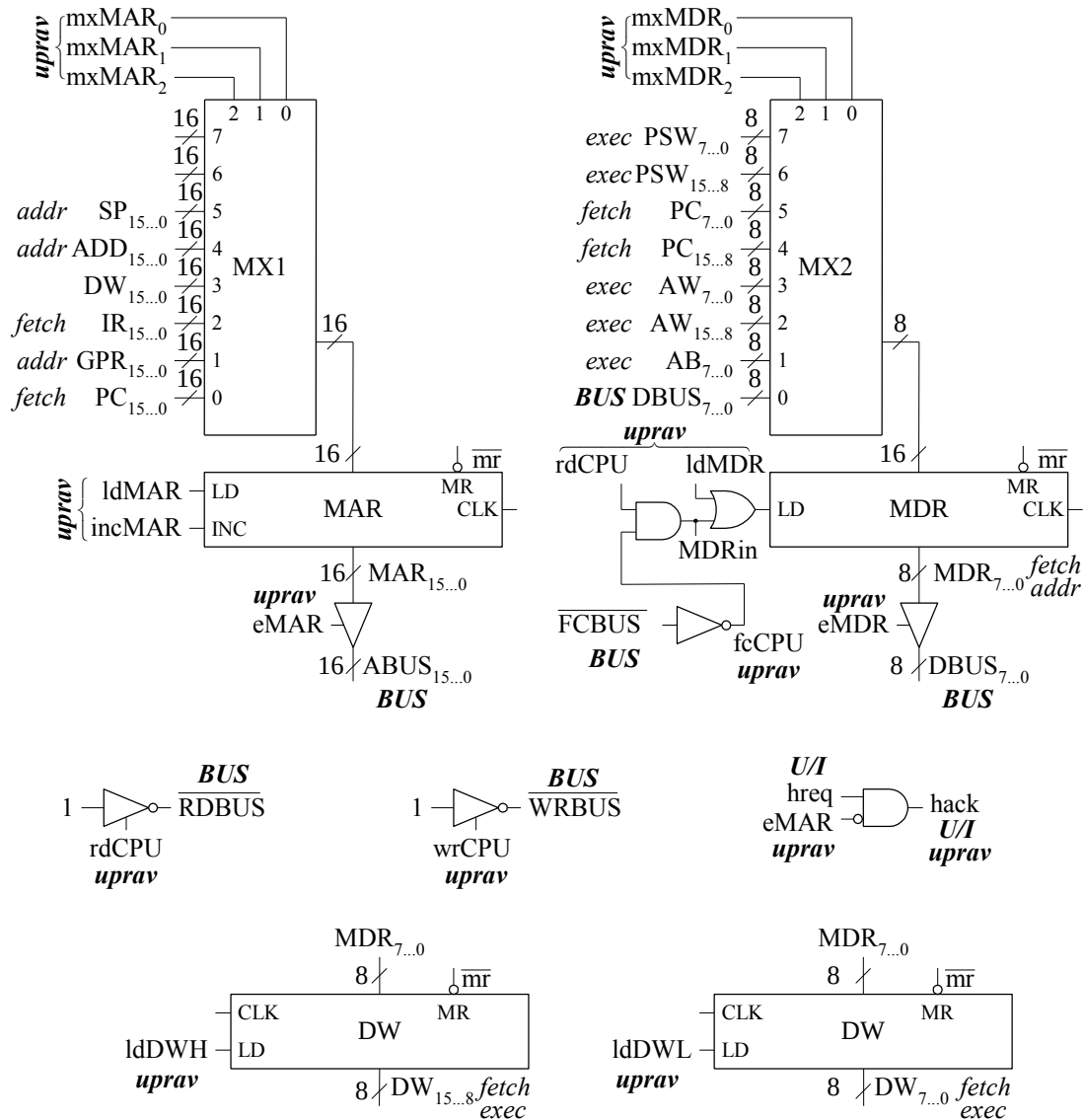
Ovi blokovi su međusobno povezani direktnim vezama.

Blok *bus* (slika 18) služi za arbitraciju procesora i ulazno/izlaznog uređaja *U/I* sa kontrolerom za direktan pristup memoriji pri korišćenju magistrale *BUS* i realizaciju ciklusa na magistrali *BUS*. Blok *fetch* (slike 19, 20, 21) služi za čitanje instrukcije i smeštanje u prihvatni registar instrukcije. Blok *addr* (slika 22) služi za formiranje adrese operanda i čitanje operanda. Blok *exec* (slike 23, 24, 25 i 26) služi za izvršavanje operacija. Blok *intr* (slike 27, 28 i 29) služi za prihvatanje prekida i generisanje broja ulaza u tabelu sa adresama prekidnih rutina

Struktura i opis blokova operacione jedinice *oper* se daju u daljem tekstu.

4.1.1 Blok bus

Blok *bus* (slika 18) sadrži registre $MAR_{15...0}$ i $MDR_{7...0}$ sa multiplekserima *MX1* i *MX2*, respektivno, prihvatni registar $DW_{15...0}$ i kombinacione mreže za realizaciju ciklusa na magistrali *BUS* kada je procesor gazda i za arbitraciju sa kontrolerom sa direktnim pristupom memoriji pri korišćenju magistrale *BUS*.



Slika 18 Blok bus

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 multipleksa MX1 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-mo bitnim lokacijama. Tada se najpre u registar $MAR_{15...0}$ upisuje adresa niže lokacije, a posle se inkrementiranjem dobija adresa više lokacije. Pri realizaciji ciklusa čitanja ili upisa se vrednošću 1 signala **eMAR** sadržaj registra $MAR_{15...0}$ propušta kroz bafere sa tri stanja na adresne linije $ABUS_{15...0}$ magistrale **BUS**.

Multiplekser MX1 je 16-to razredni multiplekser sa 8 ulaza. Na ulaze 0 do 5 multipleksa se vode sadržaji $PC_{15...0}$, $GPR_{15...0}$, $IR_{15...0}$, $DW_{15...0}$, $ADD_{15...0}$ i $SP_{15...0}$, a selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 000 do 101 upravljačkih signala **mxMAR₂**, **mxMAR₁** i **mxMAR₀**. Sadržaj $PC_{15...0}$ predstavlja adresu instrukcije. Sadržaj $GPR_{15...0}$ predstavlja adresu operanda u slučaju registarskog indirektnog adresiranja. Sadržaj $IR_{15...0}$ predstavlja adresu operanda u slučaju memorijskog direktnog adresiranja. Sadržaj $IR_{15...0}$ predstavlja i adresu memorijske lokacije na kojoj se nalazi adresa operanda u slučaju memorijskog indirektnog adresiranja, pa se tada sa ove i prve sledeće adrese čita iz memorije adresa operanda i upisuje u registar $DW_{15...0}$, tako da konačno sadržaj $DW_{15...0}$ predstavlja adresu operanda u slučaju memorijskog indirektnog adresiranja. Sadržaj $ADD_{15...0}$ predstavlja adresu operanda u slučaju registarskog indirektnog adresiranja sa pomerajem, bazno indeksnog sa pomerajem adresiranja i PC relativnog sa pomerajem adresiranja i formira se na izlazu sabirača ADD bloka *addr* saglasno pravilima formiranja adrese za data adresiranja. Sadržaj $SP_{15...0}$ se koristi prilikom stavljanja sadržaja na vrh steka i skidanja sadržaja sa vrha steka. Selektovani sadržaj sa izlaza multipleksa MX1 se vodi na paralelne ulaze registra $MAR_{15...0}$.

Registar $MDR_{7...0}$ je 8-mo razredni registar podatka u koji se vrednošću 1 jednog od signala **MDRin** i **ldMDR** upisuje sadržaj sa izlaza multipleksa MX2. Vrednošću 1 signala **MDRin** i binarnoj vrednosti 000 signala **mxMDR₂**, **mxMDR₁** i **mxMDR₀** sadržaj sa linije podataka $DBUS_{7...0}$ magistrale **BUS** se propušta kroz multiplekser MX2 i upisuje u registar $MDR_{7...0}$. Vrednošću 1 signala **ldDR** i binarnim vrednostima 001 do 111 signala **mxMDR₂**, **mxMDR₁** i **mxMDR₀** jedan od sadržaja $AB_{7...0}$, $AW_{15...8}$, $AW_{7...0}$, $PC_{15...8}$, $PC_{7...0}$, $PSW_{15...8}$ i $PSW_{7...0}$ se propušta kroz multiplekser MX2 i upisuje u registar $MDR_{7...0}$. Signal **MDRin** ima vrednost 1 kada signali **rdCPU** i **fcCPU** imaju vrednost 1. Signal **rdCPU** ima vrednost 1 kada procesor kao gazda realizuje ciklus čitanja na magistrali, dok signal **fcCPU** ima vrednost 1 kada se na upravljačkoj liniji magistrale **FCBUS** pojavi vrednost 0 u trajanju jedna perioda signala takta kao indikacija od memorije ili nekog od kontrolera kao služe da je pročitani podatak raspoloživ na linijama podataka $DBUS_{7...0}$ magistrale. Sadržaj registra $MDR_{7...0}$ se vrednošću 1 signala **eMDR** propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale.

Multiplekser MX2 je 16-to razredni multiplekser sa 8 ulaza. Na ulaze 0 do 7 multipleksa se vode sadržaji $DBUS_{7...0}$, $AB_{7...0}$, $AW_{15...8}$, $AW_{7...0}$, $PC_{15...8}$, $PC_{7...0}$, $PSW_{15...8}$ i $PSW_{7...0}$, a selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 000 do 111 upravljačkih signala **mxMBR₂**, **mxMBR₁** i **mxMBR₀**. Sadržaj $DBUS_{7...0}$ se propušta kod ciklusa čitanja i predstavlja pročitanu vrednost memorijske lokacije ili registra kontrolera koja po linijama podataka magistrale **BUS** dolazi u procesor **CPU**. Sadržaji $AB_{7...0}$, $AW_{15...8}$, $AW_{7...0}$, $PC_{15...8}$, $PC_{7...0}$, $PSW_{15...8}$ i $PSW_{7...0}$ se propuštaju u slučaju ciklusa upisa u memorijsku lokaciju ili registar kontrolera. Sadržaj $AB_{7...0}$ se propušta u slučajevima u kojima se sadržaj akumulatora

AB_{7...0} upisuje, sadržaji AW_{15...8} i AW_{7...0} u slučajevima u kojima se u dva ciklusa na magistrali sadržaji višeg i nižeg bajta akumulatora AW_{15...0} upisuju, sadržaji PC_{15...8} i PC_{7...0} u slučajevima u kojima se u dva ciklusa na magistrali sadržaji višeg i nižeg bajta programskog brojača PC_{15...0} stavljaju na stek i sadržaji PSW_{15...8} i PSW_{7...0} u slučajevima u kojima se u dva ciklusa na magistrali sadržaji višeg i nižeg bajta programske statusne reči PSW_{15...0} stavljaju na stek. Selektovani sadržaja sa izlaza multipleksera MX2 se vodi na paralelne ulaze registra MDR_{7...0}.

Registar DW_{15...0} je 16-to razredni pomoćni registar koji se koristi za prihvatanje 16-to bitne veličine koja se dobija iz dve susedne 8-mo bitne memorijske lokacije u dva posebna ciklusa na magistrali. Vrednošću 1 signala **ldDWH** se u 8 starijih razreda registra DW_{15...8} upisuje sadržaj registra MDR_{7...0} u koji je prethodno upisan sadržaj sa linija podataka DBUS_{7...0} magistrale, dok se vrednošću 1 signala **ldDWL** u 8 mlađih razreda registra DW_{7...0} upisuje sadržaj registra MDR_{7...0} u koji je prethodno upisan sadržaj sa linija podataka DBUS_{7...0} magistrale. Sadržaj registra DW_{15...0} se potom kao 16-to bitna veličina upisuje u odgovarajući 16-to razredni registar. Registar DW_{15...0} se koristi da se prihvati 16-to bitna adresa operanda u slučaju indirektnog memorijskog adresiranja i da se posle upiše u registar MAR_{15...0}. Pored toga registar DW_{15...0} se koristi da se prihvati 16-to bitni operand u slučaju instrukcije LDW koja se izvršava nad 16-to bitnim veličinama i da se posle upiše u registar BW_{15...0}.

Registar DW_{15...0} se koristi i da se prihvati 16-to bitna vrednost sa steka prilikom povratka iz potprograma ili prekidne rutine ili 16-to bitna vrednost adrese prekidne rutine iz tabele sa adresama prekidnih rutina i da se posle upiše u programski brojač PC_{15...0}.

Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je procesor gazda formiraju signale **RDBUS** i **WRBUS** magistrale **BUS** i **fcCPU** upravljačke jedinice **uprav**. Signali **RDBUS** i **WRBUS** se formiraju na osnovu signala **rdCPU** i **wrCPU** upravljačke jedinice **uprav**. Signal **fcCPU** se formira na osnovu signala **FCBUS** magistrale **BUS** koji šalje memorija ili kontroler kao sluga.

Pri realizaciji ciklusa čitanja na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije ABUS_{15..0} i upravljačku liniju **RDBUS** magistrale i na njih izbacuje adresu i vrednost 0 signala čitanja, respektivno, čime se u memoriji ili kontroleru kao slugi startuje čitanje adresirane lokacije. Memorija ili kontroler po završenom čitanju otvara bafere sa tri stanja za linije podataka DBUS_{7..0} i upravljačku liniju **FCBUS** magistrale i na njih izbacuje sadržaj adresirane lokacije i vrednost 0 signala završetka operacije čitanja u memoriji ili kontroleru trajanja jedna perioda signala takta. Procesor prihvata sadržaj sa linija podataka i zatvara bafere sa tri stanja za adresne linije ABUS_{15..0} i upravljačku liniju **RDBUS** magistrale, dok memorija ili kontroler po isteku jedne periode signala takta zatvara bafere sa tri stanja za linije podataka DBUS_{7..0} i upravljačku liniju **FCBUS** magistrale.

Pri realizaciji ciklusa upisa na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije ABUS_{15..0}, linije podataka DBUS_{7..0} i upravljačku liniju **WRBUS** magistrale i na njih izbacuje adresu, podatak i vrednost 0 signala upisa, respektivno, čime se u memoriji ili kontroleru kao slugi startuje upis u adresiranu lokaciju. Memorija ili kontroler po završenom upisu otvara bafer sa tri stanja za upravljačku liniju **FCBUS** magistrale i na nju izbacuje vrednost 0 signala završetka operacije upisa u memoriji ili kontroleru trajanja jedna perioda

signala takta. Procesor zatvara bafere sa tri stanja za adresne linije $\overline{\text{ABUS}}_{15..0}$, linije podataka $\overline{\text{DBUS}}_{7..0}$ i upravljačku liniju $\overline{\text{WRBUS}}$ magistrale, dok memorija ili kontroler po isteku jedne periode signala takta zatvara bafer sa tri stanja za upravljačku liniju $\overline{\text{FCBUS}}$ magistrale.

Kombinacione mreže za generisanje upravljačkih signala $\overline{\text{RDBUS}}$ i $\overline{\text{WRBUS}}$ magistrale generišu ove signale na osnovu signala rdCPU i wrCPU upravljačke jedinice *uprav*, respektivno. Pri vrednosti 0 signala rdCPU na liniji signala $\overline{\text{RDBUS}}$ je stanje visoke impedance, dok je pri vrednosti 1 signala rdCPU na liniji signala $\overline{\text{RDBUS}}$ vrednost 0. Signal rdCPU ima vrednost 1 ili 0 u zavisnosti od toga da li je u memoriji ili kontroleru operacija čitanja u toku ili nije, respektivno. Pri vrednosti 0 signala wrCPU na liniji signala $\overline{\text{WRBUS}}$ je stanje visoke impedance, dok je pri vrednosti 1 signala wrCPU na liniji signala $\overline{\text{WRBUS}}$ vrednost 0. Signal wrCPU ima vrednost 1 ili 0 u zavisnosti od toga da li je u memoriji ili kontroleru operacija upisa u toku ili nije, respektivno.

Signal fcCPU se generiše na osnovu vrednosti upravljačkog signala $\overline{\text{FCBUS}}$ magistrale koji šalje memorija ili kontroler. Signal fcCPU ima vrednost 0 ukoliko je na liniji signala $\overline{\text{FCBUS}}$ stanje visoke impedanse ili vrednost 1 i vrednost 1 ukoliko je na liniji signala $\overline{\text{FCBUS}}$ vrednost 0. Signal $\overline{\text{FCBUS}}$ ima vrednost 0, a time signal fcCPU vrednost 1, samo u trajanju jedna perioda signala takta, a generiše ga memorija ili kontroler kao indikacija procesoru da je završena operacija čitanja neke lokacije memorije ili kontrolera ili operacija upisa u neku lokaciju memorije ili kontrolera.

Signali hreq , koji kontroler sa direktnim pristupom memoriji šalje procesoru, i hack , koji procesor šalje kontroleru sa direktnim pristupom memoriji, se koriste za arbitracije između procesora i kontrolera sa direktnim pristupom memoriji oko korišćenja magistrale *BUS*.

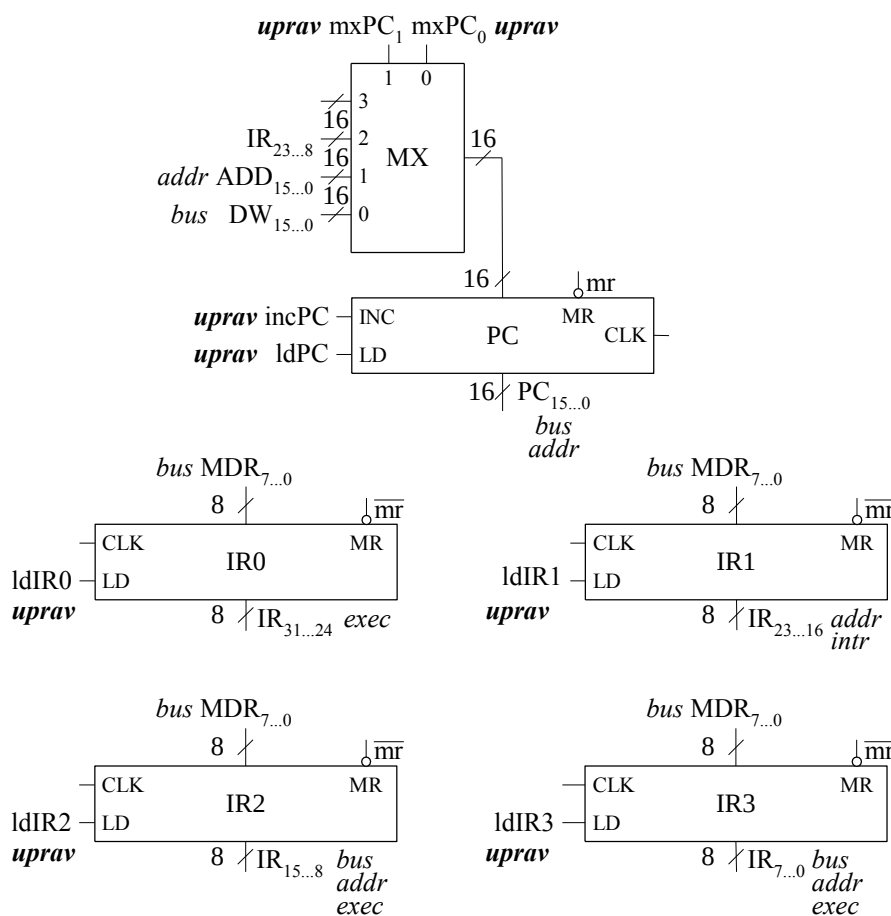
Pri realizaciji ciklusa čitanja ili upisa na magistrali kontroler kao gazda najpre procesoru šalje vrednost 1 signal zahteva korišćenja magistrale hreq i tek po dobijanju vrednosti 1 signala dozvole korišćenja magistrale hack od procesora kreće sa realizacijom ciklusa na magistrali. Ukoliko je zadat pojedinačni režim prenosa kontrolera sa direktnim pristupom memoriji, kontroler po završetku prenosa jednog podatka ukida zahtev korišćenja magistrale postavljanjem signala hreq na vrednost 0, a procesor ukida dozvolu korišćenja magistrale postavljanjem signala hack na vrednost 0. Ovakva razmena signala hreq i hack između kontrolera i procesora se ponavlja pri prenosu svakog podatka bloka podataka. Ukoliko je zadat paketski režim prenosa kontrolera sa direktnim pristupom memoriji, kontroler ne ukida vrednost 1 signala zahteva korišćenja magistrale hreq dok traje prenos celog bloka podataka, pa procesor drži vrednost 1 signala dozvole korišćenja magistrale hack sve vreme dok traje prenos bloka podataka. Tek po završetku prenosa celog bloka podataka kontroler ukida zahtev korišćenja magistrale postavljanjem signala hreq na vrednost 0, a procesor ukida dozvolu korišćenja magistrale postavljanjem signala hack na vrednost 0.

Pri realizaciji ciklusa čitanja ili upisa na magistrali procesor kao gazda najpre proverava da li je magistrala data kontroleru sa direktnim pristupom memoriji ili nije na šta ukazuju vrednost 1 i 0 signala hack , respektivno. Ukoliko signal hack ima vrednost 1, procesor čeka da najpre kontroler završi sa korišćenjem magistrale i postavi signal hreq na vrednost 0, pa tek pošto signal hack postavi na vrednost 0 kreće sa realizacijom ciklusa na magistrali. Ukoliko signal hack ima vrednost 0, procesor odmah kreće sa realizacijom ciklusa na magistrali.

Vrednost 1 signala **eMAR** je indikacija da procesor trenutno realizuje neki od svojih ciklusa na magistralu, pa zato procesor i pored vrednosti 1 signal **hreq** drži vrednost 0 signala **hack** i time ne daje kontroleru dozvolu korišćenja magistrale. Vrednosti 0 signala **eMAR** je indikacija da procesor trenutno ne realizuje neki od svojih ciklusa na magistrali, pa zato procesor pri vrednosti 1 signala **hreq** daje kontroleru dozvolu korišćenja magistrale postavljanjem signala **hack** na vrednost 1.

4.1.2 Blok fetch

Blok *fetch* sadrži registar PC_{15...0} sa multiplekserom MX, registre IR0, IR1, IR2 i IR3 (slika 19), dekodere DC1 do DC11 signala logičkih uslova operacija i načina adresiranja (slika 20) i kombinacione mreže signala logičkih uslova dužina instrukcija (slika 21).



Slika 19 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 do četiri bajta instrukcije. Adresa skoka u programu se iz nekog od blokova operacione jedinice **oper** preko multipleksa MX vodi na ulaze registra PC_{15...0} i u njega upisuje vrednošću 1 signala 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-mo bitnim lokacijama. Sadržaj registra PC_{15...0} se koristi u bloku *addr* i za formiranje adrese memorijske lokacije kada se za adresiranje operanda koristi PC relativno adresiranje.

Multiplekser MX je 16-to razredni multiplekser sa 4 ulaza. Na ulaze 0 do 2 multipleksa se vode sadržaji DW_{15...0}, ADD_{15...0} i IR_{23...8}, koji predstavljaju adrese skoka za upis u registar PC_{15...0}, a selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 10

upravljačkih signala **mxPC₁** i **mxPC₀**. Sadržaj DW_{15...0} predstavlja ili vrednost koja se restaurira sa steka prilikom izvršavanja instrukcija povratka iz potprograma ili povratka iz prekidne rutine ili adresu prekidne rutine koja se čita iz tabele sa adresama prekidnih rutina prilikom opsluživanja prekida. Sadržaj ADD_{15...0} predstavlja adresu skoka koja se dobija sabiranjem sadržaja programskog brojača i pomeraja prilikom izvršavanja instrukcija uslovnog skoka i IR_{23...8} predstavlja adresu skoka koja se uzima iz instrukcije prilikom izvršavanja instrukcija bezuslovnog skoka ili skoka na potprogram.

Registri IR0, IR1, IR2 i IR3 su 8-mo razredni registri koji formiraju razrede 31...24, 23...16, 15...8 i 7...0, respektivno, prihvatnog registra instrukcije IR_{31...0}. Instrukcije mogu, u zavisnosti od formata instrukcije, da budu dužine 1, 2, 3 ili 4 bajta (poglavljje 2.1.3). Saglasno formatu instrukcije prvi, drugi, treći i četvrti bajt instrukcije se smeštaju redom u registre IR0, IR1, IR2 i IR3. Paralelan upis sadržaja prihvatnog registra podatka MDR_{7...0} u jedan od registara IR0, IR1, IR2 i IR3 se realizuje vrednošću 1 jednog od signala **ldIR0**, **ldIR1**, **ldIR2** i **ldIR3**, respektivno. Razredi IR_{31...24} 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 (poglavljje 2.1.3).

Dekoderi DC1 do DC12 se koriste za dekodovanje instrukcija i formiranje signala logičkih uslova operacija **NOP**, ... , **ROLC** i načina adresiranja **regdir**, ..., **imm** (slika 20) saglasno načinu kodiranja instrukcija (poglavljje 2.1.5.2) i načina adresiranja (poglavljje 2.1.4).

Dekoder DC1 služi za formiranje signala četiri grupe operacija G0, G1, G2 i G3 (tabela 3). Dekoder DC2 služi za formiranje signala osam podgrupa operacija G0_PG0 do G0_PG7 grupe G0 (tabela 4). Dekoder DC3 služi za formiranje signala **NOP**, **HALT**, **INTD**, **INTE**, **TRPD** i **TRPE** podgrupe operacija G0_PG0 (tabela 5).

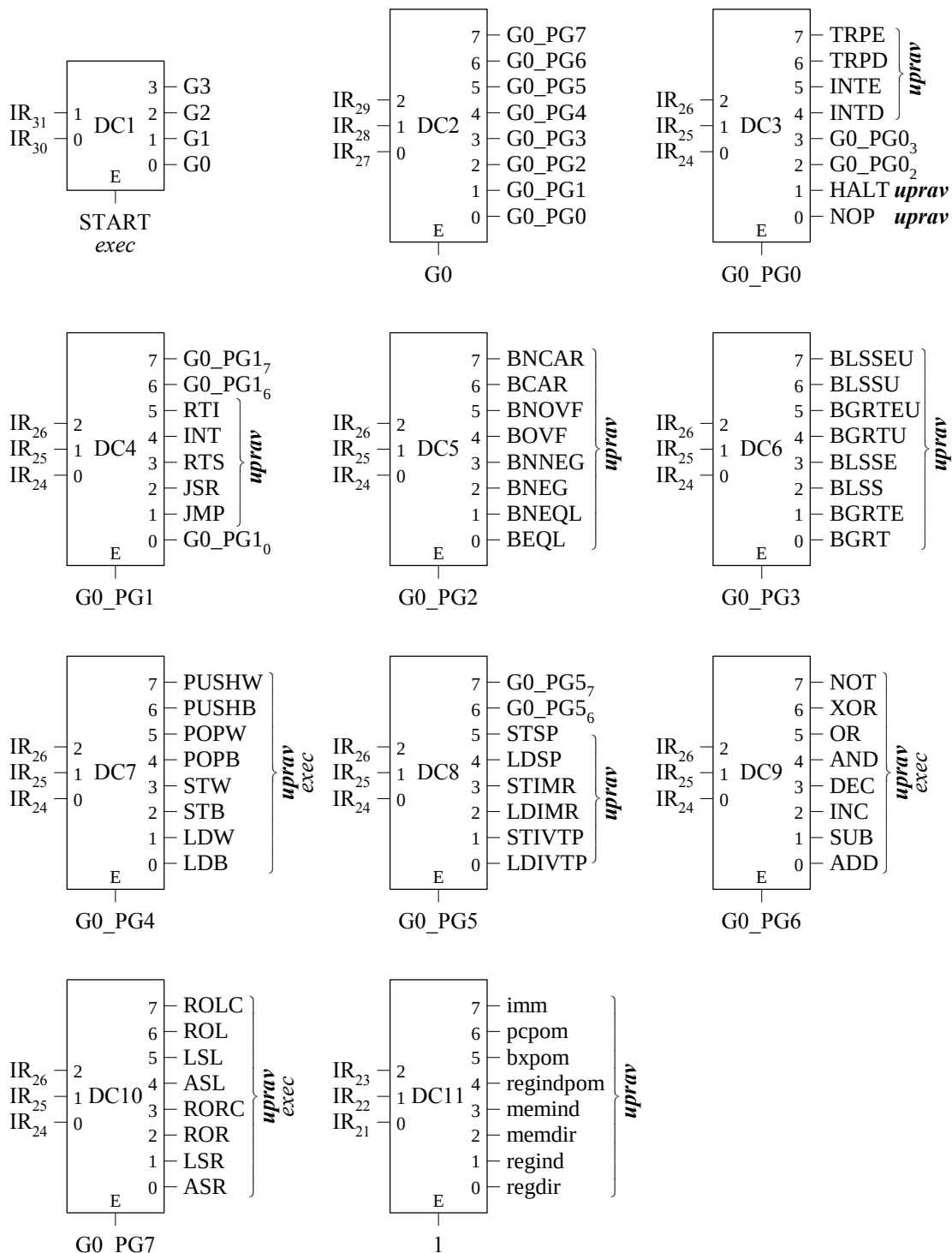
Dekoder DC4 služi za formiranje signala **JMP**, **JSR**, **RTS**, **INT** i **RTI** podgrupe operacija G0_PG1 (tabela 6). Dekoder DC5 služi za formiranje signala **BEQL**, **BNEQL**, **BNEG**, **BNNEG**, **BOVF**, **BNOVF**, **BCAR** i **BNCAR** podgrupe operacija G0_PG2 (tabela 7). Dekoder DC6 služi za formiranje signala **BGRT**, **BGRTE**, **BLSS**, **BLSSE**, **BGRTU**, **BGRTEU**, **BLSSU** i **BLSSEU** podgrupe operacija G0_PG3 (tabela 8).

Dekoder DC7 služi za formiranje signala **LDB**, **LDW**, **STB**, **STW**, **POPB**, **POPW**, **PUSHB** i **PUSHW** podgrupe operacija G0_PG4 (tabela 9). Dekoder DC8 služi za formiranje signala **LDIVTP**, **STIVTP**, **LDIMR**, **STIMR**, **LDSP** i **STSP** podgrupe operacija G0_PG5 (tabela 10). Dekoder DC9 služi za formiranje signala **ADD**, **SUB**, **INC**, **DEC**, **AND**, **OR**, **XOR** i **NOT** podgrupe operacija G0_PG6 (tabela 11). Dekoder DC10 služi za formiranje signala **ASR**, **LSR**, **ROR**, **RORC**, **ASL**, **LSL**, **ROL** i **ROLC** podgrupe operacija G0_PG7 (tabela 12).

Dekoder DC11 služi za formiranje signala **regdir**, **regind**, **memdir**, **memind**, **regindpom**, **bxpom**, **bcpom**, **imm** načina adresiranja (tabela 1).

Kombinacione mreže signala logičkih uslova greška operacije, greška adresiranja i dužine instrukcija formiraju signale koji ukazuju da li postoji greška u pročitanoj instrukciji i ukoliko ne postoji kolika je dužina instrukcije u bajtovima (slika 21).

Signal logičkog uslova greška operacije **gropr** ima vrednost 1 ukoliko se u razredima IR_{31...24} nađe binarna vrednost koja nije dodeljena ni jednoj od operacija iz skupa instrukcija. To se dešava kada je vrednost 1 nekog od signala grupa G1, G2 i G3 koje se ne koriste za kodiranje operacija ili nekog od signala iz podgrupa grupe G0 koji ne odgovaraju ni jednoj od operacija.



Slika 20 Blok fetch (drugi deo)

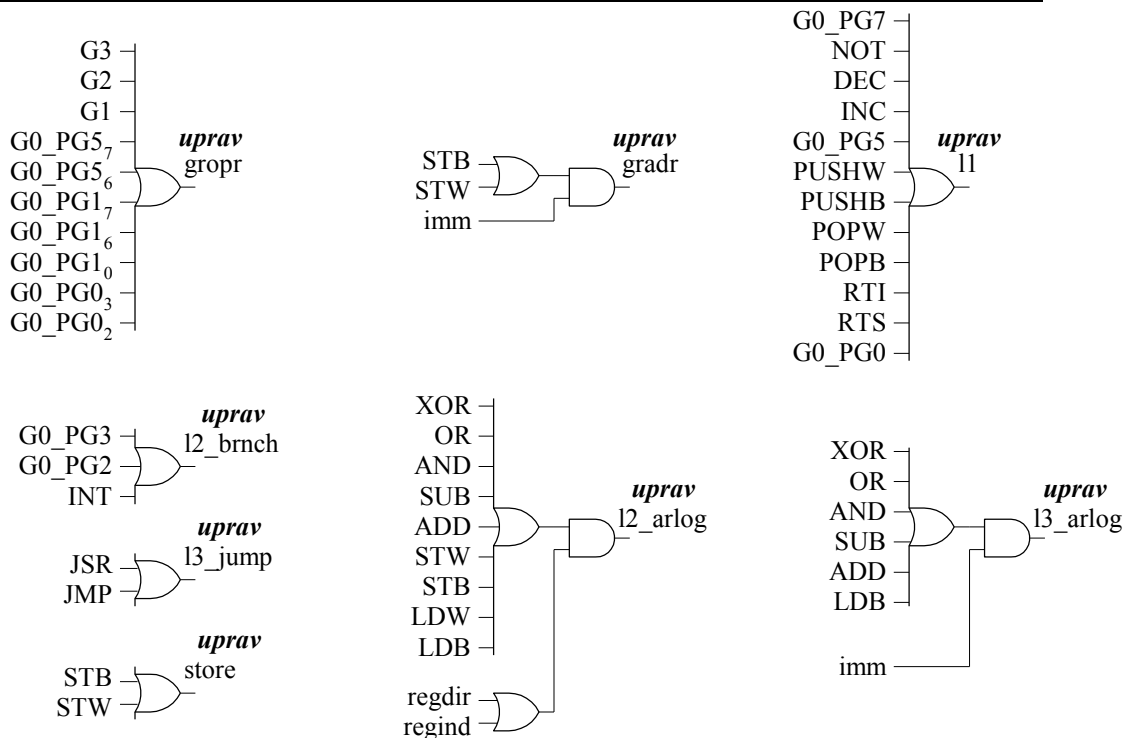
Signal logičkog uslova greška adresiranja **gradr** ima vrednost 1 ukoliko se kao odredište koristi neposredno adresiranje. To se dešava samo ukoliko je u operacijama upisa STB ili STW specificirano neposredno adresiranje imm.

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, tri ili četiri bajta. Signal **l1** ima vrednost 1 ukoliko se radi o nekoj od instrukcija iz podgrupa 0, 2 ili 5 grupe nula ili o instrukcijama RTS, RTI, POPB, POPW, PUSHB ili PUSHW iz ostalih podgrupa grupe nula, dok u ostalim situacijama ima vrednost 0.

Signali logičkih uslova dužina instrukcije dva bajta **I2_brnch** i **I2_arlog** svojom vrednošću 1 određuju da je dužina instrukcije dva bajta, a vrednošću 0 da je dužina instrukcije tri ili četiri bajta.

Signal **I2_brnch** ima vrednost 1 ukoliko se radi o nekoj od instrukcija uslovnog skoka iz podgrupe 2 ili 3 ili o instrukciji prekida INT iz podgrupe 1 grupe 0.

Signal **I2_arlog** ima vrednost 1 ukoliko se radi o nekoj od aritmetičkih, logičkih ili instrukcija prenosa XOR, OR, AND, SUB, ADD, STW, STB, LDW ili LDB za koju je specificirano registarsko direktno regdir ili registarsko indirektno regind adresiranje.



Slika 21 Blok fetch (treći deo)

Signali logičkih uslova dužina instrukcije tri bajta **I3_jump** i **I3_arlog** svojom vrednošću 1 određuju da je dužina instrukcije tri bajta, a vrednošću 0 da je dužina instrukcije četiri bajta.

Signal **I3_jump** ima vrednost 1 ukoliko se radi o instrukcija bezuslovnog skoka JMP ili skoka na potprogram JSR.

Signal **I3_arlog** ima vrednost 1 ukoliko se radi o nekoj od aritmetičkih, logičkih ili instrukcija prenosa XOR, OR, AND, SUB, ADD ili LDB za koju je specificirano neposredno adresiranje.

Signal logičkog uslova **store** svojom vrednošću 1 određuje da se radi o nekoj od instrukcija STB ili STW kojima se upisuje u memoriju.

4.1.3 Blok addr

Blok *addr* sadrži registre opšte namene GPR sa adresnim registrom registara opšte namene GPRAR_{4...0} i multiplexerom MX1, sabirač ADD sa multiplexerima MX2 i MX3, pomoćni registar CW_{15...0} i ukazivač na vrh steka SP_{15...0} (slika 22).

registra $GPRAR_{4...0}$ se inkrementira vrednošću 1 signala **incGPRAR**, što se koristi samo ukoliko se radi o bazno indeksnom sa pomerajem adresiranju. Tada se registar opšte namene čija je adresa zadata bitovima 5...0 drugog bajta instrukcije koristi kao bazni registar, a registar opšte namene sa prve sledeće adrese kao indeksni registar. Sadržaj registra $GPRAR_{4...0}$ se vodi na adresne linije $A_{4...0}$ registarskog fajla GPR.

Multiplekser MX1 je 16-to razredni multiplekser sa 2 ulaza. Na ulaze 0 i 1 multipleksa se vode sadržaji registara $AB_{7...0}$ i $AW_{15...0}$, pri čemu je sadržaj registra $AB_{7...0}$ proširen nulama do dužine 16 bita, a selekcija jednog od ova dva sadržaja se realizuje binarnim vrednostima 0 i 1 upravljačkog signala **mxGPR**. Ovo se koristi u instrukcijama prenosa STB i STW sadržaja akumulatora $AB_{7...0}$ i $AW_{15...0}$, respektivno, kada je registarskim direktnim adresiranjem kao odredište specificiran neki od registara opšte namene. Selektovani sadržaj se vodi na ulazne linije podataka $DI_{15...0}$ 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 izlaza multipleksa MX2 i MX3 dolaze na ulaze $A_{15...0}$ i $B_{15...0}$ sabirača ADD. Sadržaj $ADD_{15...0}$ sa izlaza sabirača se vodi u blok *bus* radi upisa u adresni registar $MAR_{15...0}$ i u blok *fetch* radi upisa u programski brojač $PC_{15...0}$, dok se u bloku *addr* upisuje u registar $CW_{15...0}$.

Multiplekser MX2 je 16-to razredni multiplekser sa 4 ulaza. Na ulaze 0 do 3 multipleksa se vode sadržaji $IVTP_{15...0}$, $GPR_{15...0}$, $PC_{15...0}$ i $CW_{15...0}$, a selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 11 upravljačkih signala **mxADDA₁** i **mxADDA₀**. Sadržaj sa izlaza multipleksa MX2 se vodi na ulaze $A_{15...0}$ sabirača ADD.

Multiplekser MX3 je 16-to razredni multiplekser sa 4 ulaza. Na ulaze 0 do 3 multipleksa se vode sadržaji $IVTDSP_{15...0}$, $IR_{15...0}$, $GPR_{15...0}$ i $IR_{23...16}$ proširen znakom do dužine 16 bita, a selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 11 upravljačkih signala **mxADDB₁** i **mxADDB₀**. Sadržaj sa izlaza multipleksa MX3 se vodi na ulaze $B_{15...0}$ sabirača ADD.

Sadržaji $IVTP_{15...0}$ i $IVTDSP_{15...0}$ se propuštaju kroz multipleksere MX2 i MX3, respektivno, 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 $GPR_{15...0}$ i $IR_{15...0}$ se propuštaju kroz multipleksere MX2 i MX3, respektivno, da bi se njihovim sabiranjem u slučaju registarskog indirektnog sa pomerajem adresiranja na izlazu sabirača ADD dobila adresa sa koje treba da se pročita ili na kojoj treba da se upiše operand. Pri tome je $GPR_{15...0}$ sadržaj registra opšte namene čija je adresa zadata instrukcijom i $IR_{15...0}$ pomeraj. Ovi sadržaji se propuštaju kroz multipleksere i sabiraju i u slučaju baznog indeksnog sa pomerajem adresiranja, pri čemu se rezultat sabiranja upisuje u registar $CW_{15...0}$.

Sadržaji $PC_{15...0}$ i $IR_{15...0}$ se propuštaju kroz multipleksere MX2 i MX3, respektivno, da bi se njihovim sabiranjem u slučaju PC relativnog sa pomerajem adresiranja na izlazu sabirača ADD dobila adresa sa koje treba da se pročita ili na kojoj treba da se upiše operand. Pri tome je $PC_{15...0}$ sadržaj programskog brojača i $IR_{15...0}$ pomeraj.

Sadržaji $CW_{15...0}$ i $GPR_{15...0}$ se propuštaju kroz multipleksere MX2 i MX3, respektivno, da bi se njihovim sabiranjem u slučaju baznog indeksnog sa pomerajem adresiranja na izlazu sabirača ADD dobila adresa sa koje treba da se pročita ili na kojoj treba da se upiše operand. Pri tome je $CW_{15...0}$ sadržaj pomoćnog registra opšte namene u kome se nalazi prethodno dobijena suma registra opšte namene, koji se koristi kao bazni registar i koji se čita sa adrese zadate instrukcijom, i pomeraja, dok je $GPR_{15...0}$ sadržaj registra opšte namene, koji se koristi kao indeksni registar i čita sa prve sledeće adrese u odnosu na adresu zadatu instrukcijom.

Sadržaji $PC_{15...0}$ i $IR_{23...16}$ proširen znakom do dužine 16 bita se propuštaju kroz multipleksere MX2 i MX3, respektivno, 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 $IR_{23...16}$ proširen znakom do dužine 16 bita pomeraj u odnosu na tekuću vrednost programskog brojača.

Registar $CW_{15...0}$ je 16-to razredni pomoćni registar u kome se u slučaju baznog indeksnog sa pomerajem adresiranja čuva suma registra opšte namene, koji se koristi kao bazni registar, i pomeraja. Ova suma se dobija sa izlaza sabirača ADD i u registar $CW_{15...0}$ upisuje vrednošću 1 signala **ldCW**. Sadržaj registra $CW_{15...0}$ se vodi na ulaze multipleksera MX2.

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. U registar $SP_{15...0}$ se vrednošću 1 signala **ldSP** u fazi izvršavanja instrukcije prenosa upisuje sadržaj 16-to razrednog akumulatora $AW_{15...0}$. 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. Sadržaj registra $SP_{15...0}$ se vodi na ulaze adresnog registra memorije bloka *bus*.

4.1.4 Blok exec

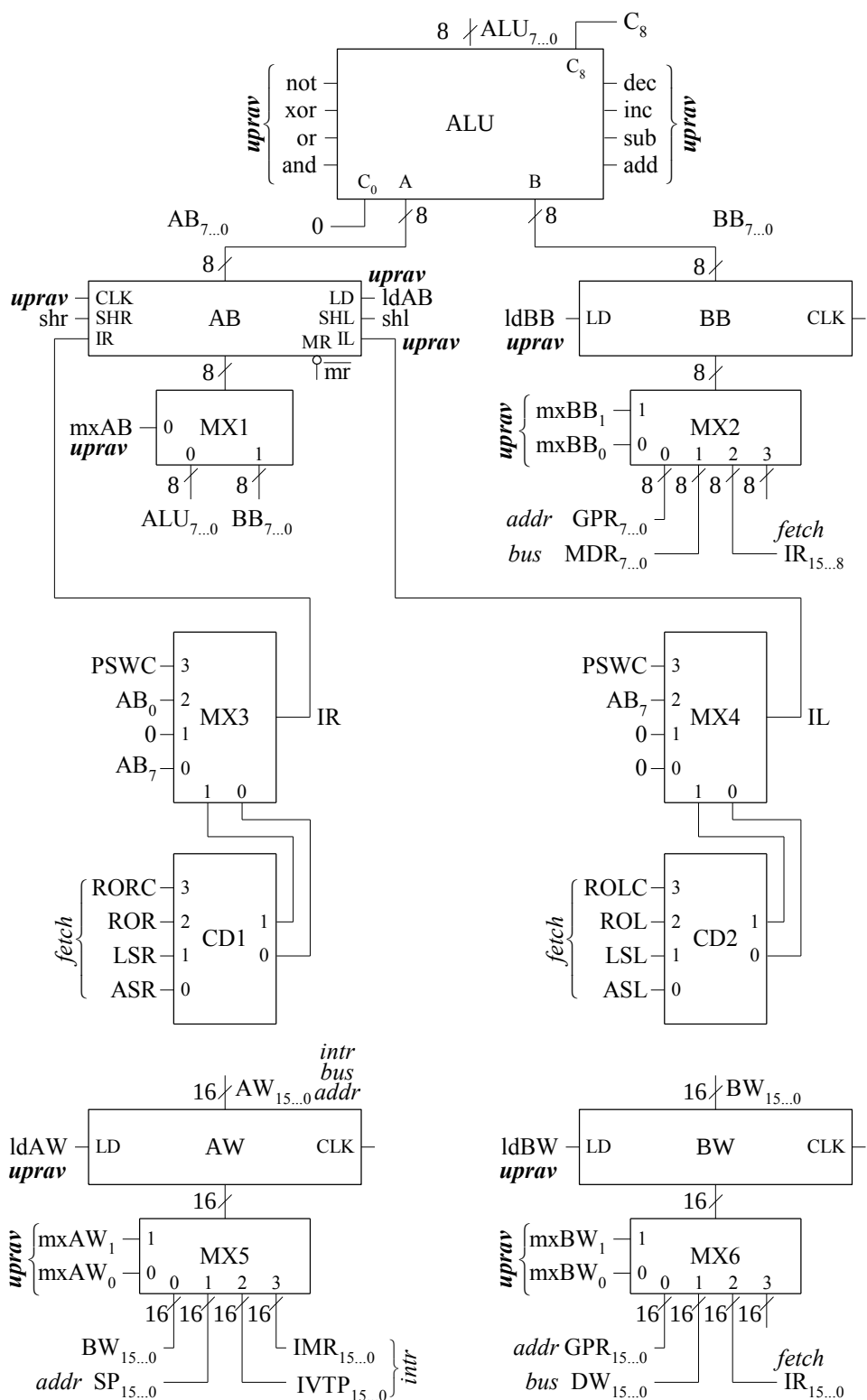
Blok *exec* sadrži aritmetičko logičku jedinicu ALU, registre $AB_{7...0}$ i $BB_{7...0}$, multipleksere MX1, MX2, MX3 i MX4, kodere DC1 i DC2, registre $AW_{15...0}$ i $BW_{15...0}$, multipleksere MX5 i MX6 (slika 23), registar $PSW_{15...0}$, flip-flop START (slika 24), kombinacione mreže za formiranje signala postavljanja indikatora N, Z, C i V (slika 25) i

kombinacione mreže za formiranje signala rezultata operacija **eql**, ..., **nneg**, **brpom** (slika 26).

Aritmetičko logička jedinica ALU realizacije četiri aritmetičke i četiri logičke mikrooperacije nad sadržajima registara $AB_{7...0}$ i $BB_{7...0}$ (slike 23). Mikrooperacija koju treba realizovati se specificira vrednošću 1 ili jednog od upravljačkih signala **add**, **sub**, **inc** i **dec** za jednu od aritmetičkih mikrooperacija sabiranja, oduzimanja, inkrementiranja i dekrementiranja, 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 $ALU_{7...0}$. 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 aritmetičkih mikrooperacija sabiranja i inkrementiranja vrednost 1 na izlazu C_8 označava da postoji prenos a vrednost 0 da nema prenosa. U slučaju aritmetičkih mikrooperacija oduzimanja i dekrementiranja vrednost 1 na izlazu C_8 označava da nema pozajmice, a vrednost 0 da ima pozajmice. U slučaju logičkih mikrooperacija signal na izlazu C_8 nema smisla.

Registar $AB_{7...0}$ je 8-mo razredni akumulator koji se koristi kao implicitno izvorište i odredište u instrukcijama prenosa, aritmetičkim, logičkim i pomeračkim instrukcijama. Sadržaj sa izlaza multipleksera MX1 se vodi na ulaze registra $AB_{7...0}$ i u njega upisuje vrednošću 1 signala **ldAB**.

Sadržaj registra $AB_{7...0}$ se pomera udesno za jedno mesto vrednošću 1 signala **shR**. Tada se u najstariji razred registra AB_7 upisuje signal **IR** koji dolazi sa izlaza multipleksera MX3. Sadržaj registra $AB_{7...0}$ se pomera ulevo za jedno mesto vrednošću 1 signala **shL**. Tada se u najmlađi razred registra AB_0 upisuje signal **IL** koji dolazi sa izlaza multipleksera MX4. Sadržaj registra $AB_{7...0}$ se vodi na ulaze ALU gde se koristi kao prvi izvorišni operand u slučaju aritmetičkih i logičkih operacija.



Slika 23 Blok exec (prvi deo)

Multiplexer MX1 je 8-mo razredni multiplexer sa 2 ulaza. Na ulaze 0 i 1 multiplexera se vode sadržaji $ALU_{7...0}$ i $BB_{7...0}$ koji se propuštaju kroz multiplexer i upisuju u registar $AB_{7...0}$. Selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 0 i 1 upravljačkog signala **mxAB**.

Sadržaj $ALU_{7...0}$ predstavlja rezultat aritmetičke ili logičke operacije koje kao odredište implicitno koriste registar akumulatora $AB_{7...0}$.

Sadržaj $BB_{7...0}$ predstavlja operand koji se u fazi izvršavanja instrukcije LDB prebacuje iz prihvatnog registra operanda $BB_{7...0}$ u registar akumulatora $AB_{7...0}$.

Registar $BB_{7...0}$ je 8-mo razredni prihvatni registar u koji se privremeno smešta izvorišni operand specificiran adresnim delom svih instrukcija sa jednoadresnim formatom. Sadržaj sa izlaza multipleksa MX2 se vodi na ulaze registra $BB_{7...0}$ i u njega upisuje vrednošću 1 signala **ldBB**. Sadržaj registra $BB_{7...0}$ se vodi

na ulaze ALU, gde se koristi kao drugi izvorišni operand u slučaju aritmetičkih i logičkih instrukcija, i na

ulaze multipleksa MX1 kroz koji se propušta i upisuje u registar $AB_{7...0}$ u slučaju instrukcije LDB.

Multiplekser MX2 je 8-mo razredni multiplekser sa 4 ulaza. Na ulaze 0 do 2 multipleksa se vode sadržaji $GPR_{7...0}$, $MDR_{7...0}$ i $IR_{15...8}$ koji se propuštaju kroz multiplekser i upisuju u registar $BB_{7...0}$. Selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 10 upravljačkih signala **mxBB₁** i **mxBB₀**. Sadržaj $GPR_{7...0}$ predstavlja operand u slučaju registarskog direktnog adresiranja, $MDR_{7...0}$ u slučaju memorijskih adresiranja i $IR_{15...8}$ u slučaju neposrednog adresiranja.

Multiplekser MX3 je 1-no razredni multiplekser sa 4 ulaza. Na ulaze 0 do 3 multipleksa se dovode signali **AB₇**, **0**, **AB₀** i **PSWC** koji se propuštaju kroz multiplekser i po liniji IR upisuju u razred AB_7 registra $AB_{7...0}$ pri realizaciji neke od četiri instrukcije pomeranja i rotiranja za jedno mesto udesno sadržaja registra $AB_{7...0}$. Selekcija jednog od ova četiri signala se realizuje binarnim vrednostima 00 do 11 sadržaja sa izlaza koda CD1. Signal **AB₇** se propušta za instrukciju aritmetičkog pomeranja udesno kada signal operacije ASR ima vrednost 1. Signal **0** se propušta za instrukciju logičkog pomeranja udesno kada signal operacije LSR ima vrednost 1. Signal **AB₀** 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 MX4 je 1-no razredni multiplekser sa 4 ulaza. Na ulaze 0 do 3 multipleksa se dovode signali **0**, **0**, **AB₇** i **PSWC** koji se propuštaju kroz multiplekser i po liniji IL upisuju u razred AB_0 registra $AB_{7...0}$ pri realizaciji neke od četiri instrukcije pomeranja i rotiranja za jedno mesto ulevo sadržaja registra $AB_{7...0}$. Selekcija jednog od ova četiri signala se realizuje binarnim vrednostima 00 do 11 sadržaja sa izlaza koda CD2. Signal **0** se propušta za instrukciju aritmetičkog pomeranja ulevo kada signal operacije ASL ima vrednost 1. Signal **0** se propušta za instrukciju logičkog pomeranja ulevo kada je aktivan signal operacije LSL. Signal **AB₇** 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 koda ima vrednost 1, na izlazima koda se pojavljuje binarna vrednost 00, koja omogućuje da se kroz multiplekser MX1 na liniju IR registra $AB_{7...0}$ propusti signal **AB₇**. Ukoliko signal operacije logičkog pomeranja udesno LSR koji je povezan na ulaz 1 koda ima vrednost 1, na izlazima koda se pojavljuje binarna vrednost 01, koja omogućuje da se kroz multiplekser MX1 na liniju IR registra $AB_{7...0}$ propusti signal **0**. Ukoliko signal operacije rotiranja udesno ROR koji je povezan na ulaz 2 koda ima vrednost 1, na izlazima koda se pojavljuje binarna vrednost 10, koja omogućuje

da se kroz multiplekser MX1 na liniju IR registra $AB_{7...0}$ propusti signal **AB₀**. Ukoliko signal operacije rotiranja udesno kroz indikator PSWC RORC koji je povezan na ulaz 3 koda ima vrednost 1, na izlazu koda se pojavljuje binarna vrednost 11, koja omogućuje da se kroz multiplekser MX1 na liniju IR registra $AB_{7...0}$ propusti signal **PSWC**.

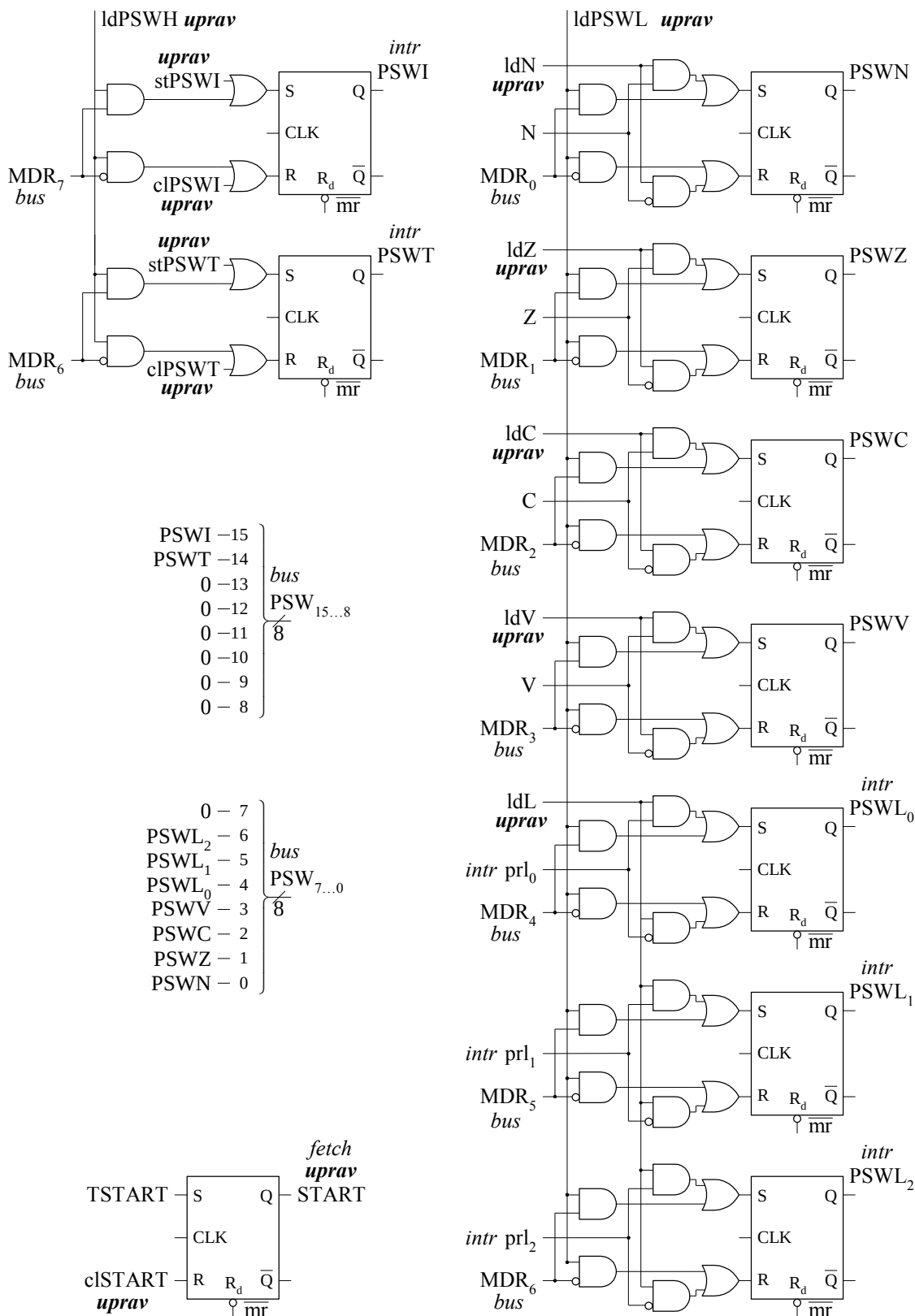
Koder CD2 je koder sa četiri ulaza i dva izlaza koji na izlazima daje binarnu vrednost 00 do 11 u zavisnosti od toga koji od signala operacija pomeranja i rotiranja ulevo za jedno mesto ASL, LSL, ROL i ROLC ima vrednost 1. Ukoliko signal operacije aritmetičkog pomeranja ulevo ASL koji je povezan na ulaz 0 koda ima vrednost 1, na izlazima koda se pojavljuje binarna vrednost 00, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $AB_{7...0}$ propusti signal **0**. Ukoliko signal operacije logičkog pomeranja ulevo LSL koji je povezan na ulaz 1 koda ima vrednost 1, na izlazima koda se pojavljuje binarna vrednost 01, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $AB_{7...0}$ propusti signal **0**. Ukoliko signal operacije rotiranja ulevo ROL koji je povezan na ulaz 2 koda ima vrednost 1, na izlazima koda se pojavljuje binarna vrednost 10, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $AB_{7...0}$ propusti signal **AB₇**. Ukoliko signal operacije rotiranja ulevo kroz indikator PSWC ROLC koji je povezan na ulaz 3 koda ima vrednost 1, na izlazima koda se pojavljuje binarna vrednost 11, koja omogućuje da se kroz multiplekser MX2 na liniju IL registra $AB_{7...0}$ propusti signal **PSWC**.

Registar $AW_{15...0}$ je 16-to razredni akumulator koji se koristi kao implicitno izvorište u instrukcijama prenosa STW, PUSHW, STIVTP, STIMR i STSP i implicitno odredište u instrukcijama prenosa LDW, POPW, LDIVTP, LDIMR i LDSP. Sadržaj sa izlaza multipleksera MX5 se vodi na ulaze registra $AW_{15...0}$ i u njega upisuje vrednošću 1 signala **ldAW**. Sadržaj registra $AW_{15...0}$ se vodi u blok *addr* gde se, ukoliko se izvršava instrukcija prenosa STW sa zadatim direktnim registarskim adresiranjem, upisuje u registarski fajl GPR i ukoliko se izvršava instrukcija STSP upisuje u registar SP. Sadržaj registra $AW_{15...0}$ se vodi i u blok *bus* gde se, ukoliko se izvršava ili instrukcija prenosa STW sa zadatim nekim od memorijskih adresiranja ili instrukcija prenosa PUSHW, upisuje u memoriju. Sadržaj registra $AW_{15...0}$ se, takođe, vodi i u blok *intr* gde se, ukoliko se izvršava instrukcija STIVTP ili instrukcija STIMR, upisuje u registre IVTP ili u registar IMR, respektivno.

Multiplekser MX5 je 16-to razredni multiplekser sa 4 ulaza. Na ulaze 0 do 3 multipleksera se vode sadržaji $BW_{15...0}$, $SP_{15...0}$, $IVTP_{15...0}$ i $IMR_{15...0}$. Selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 11 upravljačkih signala **mxAW₁** i **mxAW₀**. Sadržaj sa izlaza multipleksera MX5 se vodi na ulaze registra $AW_{15...0}$. Sadržaj $BW_{15...0}$ se propušta kroz multiplekser u slučaju instrukcije LDW, sadržaj $SP_{15...0}$ u slučaju instrukcije LDSP, sadržaj $IVTP_{15...0}$ u slučaju instrukcije LDIVTP i sadržaj $IMR_{15...0}$ u slučaju instrukcije LDIMR.

Registar $BW_{15...0}$ je 16-to razredni prihvatni registar u koji se privremeno smešta izvorišni operand specificiran adresnim delom instrukcije prenosa LDW. Sadržaj sa izlaza multipleksera MX6 se vodi na ulaze registra $BW_{15...0}$ i u njega upisuje vrednošću 1 signala **ldBW**. Sadržaj registra $BW_{15...0}$ se vodi na ulaze multipleksera MX5 kroz koji se propušta i upisuje u registar $AW_{15...0}$ u slučaju instrukcije LDW.

Multiplekser MX6 je 16-to razredni multiplekser sa 4 ulaza. Na ulaze 0 do 2 multipleksera se vode sadržaji $GPR_{15...0}$, $DW_{15...0}$ i $IR_{15...0}$. Selekcija jednog od ovih sadržaja se realizuje binarnim vrednostima 00 do 10 upravljačkih signala **mxBW₁** i **mxBW₀**. Sadržaj sa izlaza multipleksera MX6 se vodi na ulaze registra $BW_{15...0}$. Sadržaj $GPR_{15...0}$ predstavlja operand u slučaju direktnog registarskog adresiranja, sadržaj $DW_{15...0}$ u slučaju memorijskih adresiranja i sadržaj $IR_{15...0}$ u slučaju neposrednog adresiranja.



Slika 24 Blok exec (drugi deo)

Registar PSW_{15...0} je 16-to razredni registar koji sadrži indikatore programske statusne reči procesora (slika 24). Registar PSW_{15...0} se sastoji od flip-flopova PSWI, PSWT, PSWL₂, PSWL₁, PSWL₀, PSWV, PSWC, PSWZ i PSWN, koji predstavljaju razrede PSW_{15.14} i PSW_{6...0}, respektivno, dok razredi PSW_{13...7} 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 PSWL₂ do PSWL₀ sadrže indikatore *level*. U flip-flopove PSWL₂ do PSWL₀ se vrednošću 1 signala **ldL** u okviru faze opsluživanja zahteva za prekid od ulazno/izlaznih uređaja upisuju vrednosti signala **prl₂** do **prl₀**, 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 (odeljak 2.1.5) 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 (odeljak 2.1.5) 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 (odeljak 2.1.5) 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 (odeljak 2.1.5) 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*, dok se u razrede PSW_{6...0} koji predstavljaju 7 najmlađih razreda registra PSW_{15...0} vrednošću 1 signala **ldPSWL** upisuje sadržaj razreda MDR_{6...0} prihvatnog registra podatka MDR_{7...0}. Ovo se koristi prilikom izvršavanja instrukcije RTI da se sadržajem sa vrha steka restaurira sadržaj registra PSW_{15...0}. Sadržaji 2 najstarija razreda PSW_{15,14}, 7 najmlađih razreda PSW_{6...0} i vrednosti 0 za nepostojeće razrede PSW_{13...7} registra PSW_{15...0} se vode posebno kao 8 starijih razreda PSW_{15...8} i 8 mlađih razreda PSW_{7...0} registra PSW_{15...0} u blok *bus* u kome se upisuju u prihvatni registar podatka MDR_{7...0}. 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.

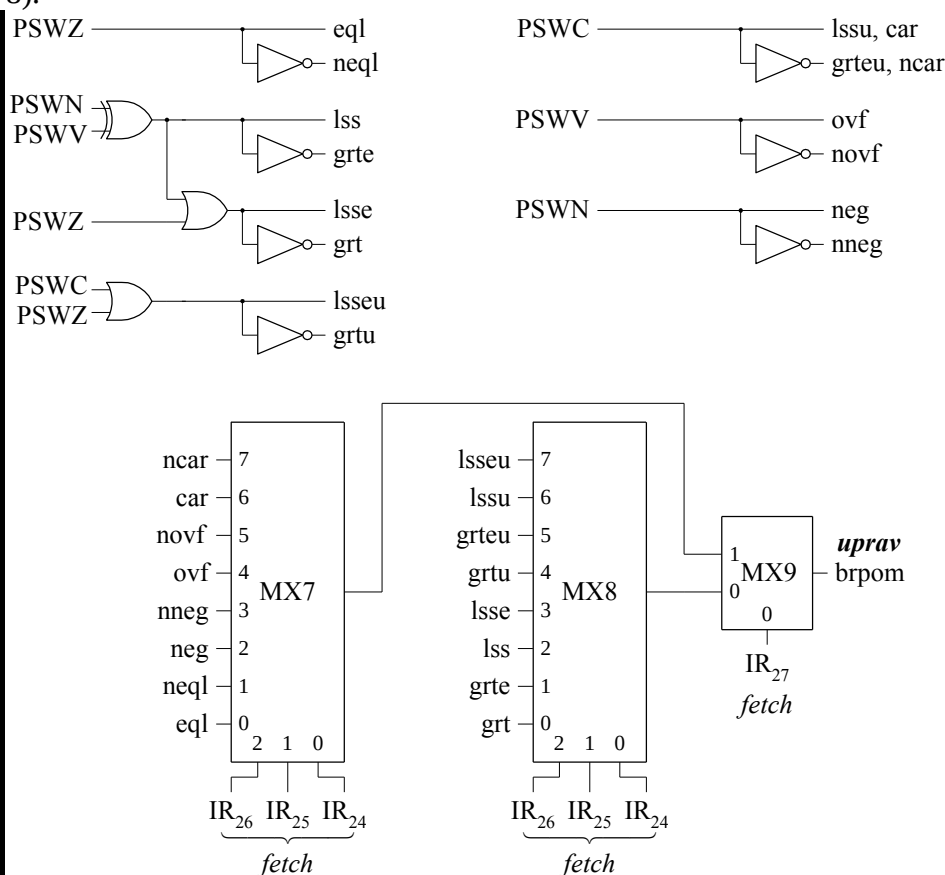
Kombinacione mreže signala postavljanja indikatora N, Z, C i V se sastoje od logičkih elemenata (slika 25). Na izlazima kombinacionih mreža se formiraju signali koji se upisuju u flip-flopove PSWN, PSWZ, PSWC i PSWV programske statusne reči u okviru izvršavanja određenih instrukcija (odeljak 2.1.5). Signal **N** ima vrednost koja odgovara vrednosti razreda **AB₇** ukoliko jedan od signala operacija LDB, ..., ROLC ima vrednost 1, dok u svim ostalim situacijama signal **N** ima vrednost 0. Signal **Z** ima vrednost 1 ukoliko svi razredi registra **AB_{7...0}** imaju vrednost 0 i ukoliko jedan od signala operacija LDB, ..., ROLC ima vrednost 1,

dok u svim ostalim situacijama signal **Z** ima vrednost 0. Signal **C** predstavlja prenos u slučaju operacija ADD i INC i ima vrednost koja odgovara vrednosti signala **C₈** ukoliko jedan od signala operacija ADD ili INC ima vrednost 1. Signal **C** predstavlja pozajmicu u slučaju operacija SUB i DEC i ima vrednost koja odgovara invertovanoj vrednosti signala **C₈** ukoliko jedan od signala operacija SUB ili DEC ima vrednost 1. Pored toga signal **C** ima vrednost koja odgovara vrednosti signala **AB₀** ukoliko jedan od signala operacija ASR, LSR, ROR ili RORC ima vrednost 1 i vrednost koja odgovara vrednosti signala **AB₇** 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 **V_{ADD}**, **V_{SUB}**, **V_{INC}** i **V_{DEC}**, ukoliko jedan od signala operacija ADD, SUB, INC ili DEC ima vrednost 1, respektivno, dok u svim ostalim situacijama signala **V** ima vrednost 0.

Signal **V_{ADD}** ima vrednost 1 ukoliko signali **AB₇** i **BB₇** imaju vrednost 0 i signal **ALU₇** ima vrednost 1 ili ukoliko signali **AB₇** i **BB₇** imaju vrednost 1 i signal **ALU₇** vrednost 0. Signal **V_{SUB}** ima vrednost 1 ukoliko signali **ALU₇** i **BB₇** imaju vrednost 0 i signal **AB₇** vrednost 1 ili ukoliko signali **ALU₇** i **BB₇** imaju vrednost 1 i signal **AB₇** ima vrednost 0. Signal **V_{INC}** ima vrednost 1 ukoliko signal **AB₇** ima vrednost 0 i signal **ALU₇** ima vrednost 1. Signal **V_{DEC}** ima vrednost 1 ukoliko signal **AB₇** ima vrednost 1 i signal **ALU₇** ima vrednost 0.

Kombinacione mreže za formiranje signala rezultata operacija **eq**, ..., **nneg** i **brpom** se sastoje od logičkih elemenata i multipleksera MX7, MX8 i MX9 (slika 26). Logičkim elementima se na osnovu sadržaja flip-flopora PSWN, PSWZ, PSWC i PSWV programske statusne reči formiraju signali **eq**, ..., **nneg**. Multiplekserima MX7, MX8 i MX9 se na osnovu razreda IR_{27...24} prihvatnog registra instrukcije kojima se specificira kod operacije instrukcije uslovnog skoka BEQL, ..., BNNEQ selektuje jedan od signala **eq**, ..., **nneg** 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 **eq**, ..., **nneg** i formiranje signala **brpom** se realizuje na osnovu kodiranja odgovarajućih kodova operacija instrukcija uslovnog skoka BEQL, ..., BNNEQ. Kodiranje instrukcija je tako realizovano da se razredom IR₂₇ selektuje jedna od dve podgrupe G0_PG2 (poglavlje 2.1.5.1.3.1) i G0_PG3 (poglavlje 2.1.5.1.3.1) instrukcija uslovnog skoka (tabela 4), dok se razredima IR_{26...24} unutar selektovane podrupe selektuje odgovarajuća instrukcija (tabele 7 i 8).



Slika 26 Blok exec (četvrti deo)

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

- eq** — rezultat nula,
- neq** — rezultat različit od nule,
- 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,
neg — razred PSWN aktivan,
nng — razred PSWN nije aktivan,
ovf — razred PSWV aktivan i
nvf — razred PSWV nije aktivan.

Signali rezultata operacija **eql**, ..., **nvf** se realizuju prema sledećim izrazima:

$$\text{eql} = \overline{\text{PSWZ}},$$

$$\text{neq} = \overline{\text{PSWZ}},$$

$$\text{grt} = \overline{((\text{PSWN} \oplus \text{PSWV}) + \text{PSWZ})},$$

$$\text{gre} = \overline{(\text{PSWN} \oplus \text{PSWV})},$$

$$\text{lss} = \text{PSWN} \oplus \text{PSWV},$$

$$\text{leq} = (\text{PSWN} \oplus \text{PSWV}) + \text{PSWZ},$$

$$\text{grtu} = \overline{\text{PSWC}} + \overline{\text{PSWZ}},$$

$$\text{greu} = \overline{\text{PSWC}},$$

$$\text{lssu} = \text{PSWC},$$

$$\text{lequ} = \text{PSWC} + \text{PSWZ},$$

$$\text{neg} = \overline{\text{PSWN}},$$

$$\text{nng} = \overline{\text{PSWN}},$$

$$\text{ovf} = \text{PSWV} \text{ i}$$

$$\text{nvf} = \overline{\text{PSWV}}.$$

4.1.5 Blok intr

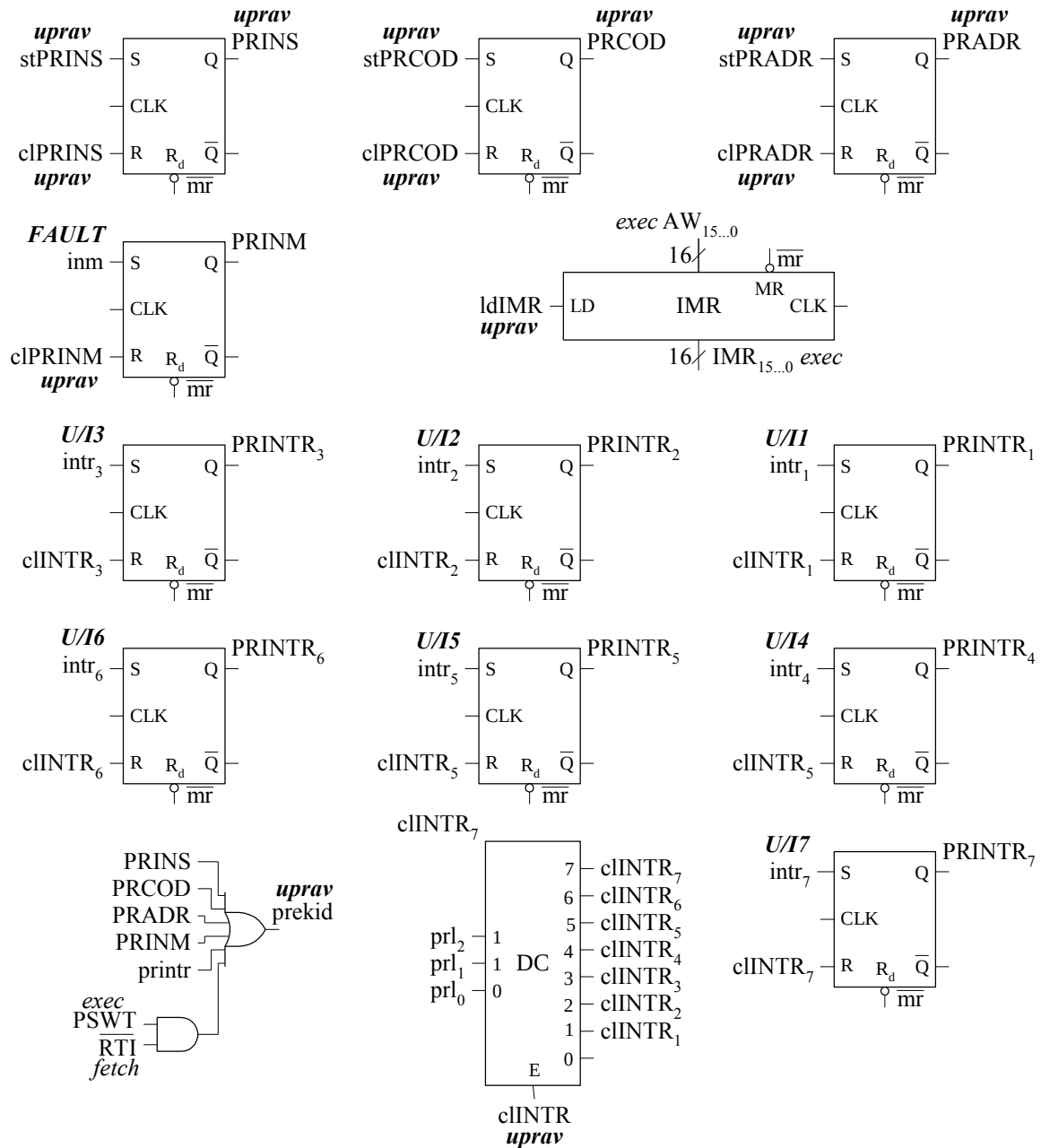
Blok *intr* sadrži logički ILI element za formiranje signala prekida **prekid**, kombinacione i sekvencijalne mreže za prihvatanje unutrašnjih prekida i spoljašnjih nemaskirajućih i maskirajućih prekida, registar IMR_{15...0} (slika 27), kombinacione prekidačke mreže za formiranje signala spoljašnjih maskirajućih prekida **printr** (slika 28), kombinacione i sekvencijalne mreže za formiranje pomeraja za tabelu sa adresama prekidnih rutina IVTDSP_{15...0} i registar IVTP_{15...0} (slika 29).

Logički ILI element za formiranje signala prekida **prekid** daje vrednost 1 signala prekida **prekid** ukoliko barem jedan od signala prekida ima vrednost 1 (slika 27). To može da bude neki od signala unutrašnjih prekida i to **PRINS** za prekid zbog izvršavanja instrukcije prekida INT, **PRCOD** za prekid usled čitanja instrukcije sa nepostojećim kodom operacije i **PRADR** za prekid pri čitanju instrukcije sa neposrednim adresiranjem za određeni operand, 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-floпова PRINS, PRCOD i PRADR.

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 27 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 **grop** (slika 21). 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 određišni 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 određišni operand i formirana vrednost 1 signala **gradr**

(slika 21). 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, PRINTR₁ do PRINTR₇ i dekodera DC (slika 27).

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 PRINTR₁ do PRINTR₇ služe za prihvatanje spoljašnjih maskirajućih zahteva za prekid intr₁ do intr₇. Zahtevi za prekid intr₁ do intr₇ koji dolaze od ulazno/izlaznih uređaja **U/I** kao impuls pamte se u flip-flopovima za prihvatanje prekida PRINTR₁ do PRINTR₇, respektivno. U obradi prekida koriste se sadržaji flip-flopova PRINTR₁ do PRINTR₇. Kada se u okviru opsluživanja zahteva za prekid prihvati neki od maskirajućih zahteva za prekid, odgovarajući flip-flop PRINTR₁ do PRINTR₇ se postavlja na vrednost 0. Postavljanje flip-flopova PRINTR₁ do PRINTR₇ na vrednost 0 se realizuje u okviru opsluživanja spoljašnjeg maskirajućeg zahteva za prekid vrednošću 1 odgovarajućeg signala **clINTR₁** do **clINTR₇**, respektivno. Postavljanje odgovarajućeg flip-flopa PRINTR₁ do PRINTR₇ 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 **prl₂** do **prl₀** daju binarnu vrednost jedne od linija intr₁ do intr₇ po kojoj je prihvaćeni zahtev za prekid stigao, ovi signali se koriste da selektuju odgovarajući flip-flop PRINTR₁ do PRINTR₇ koji se vrednošću 1 signala **clINTR** postavlja na vrednost 0.

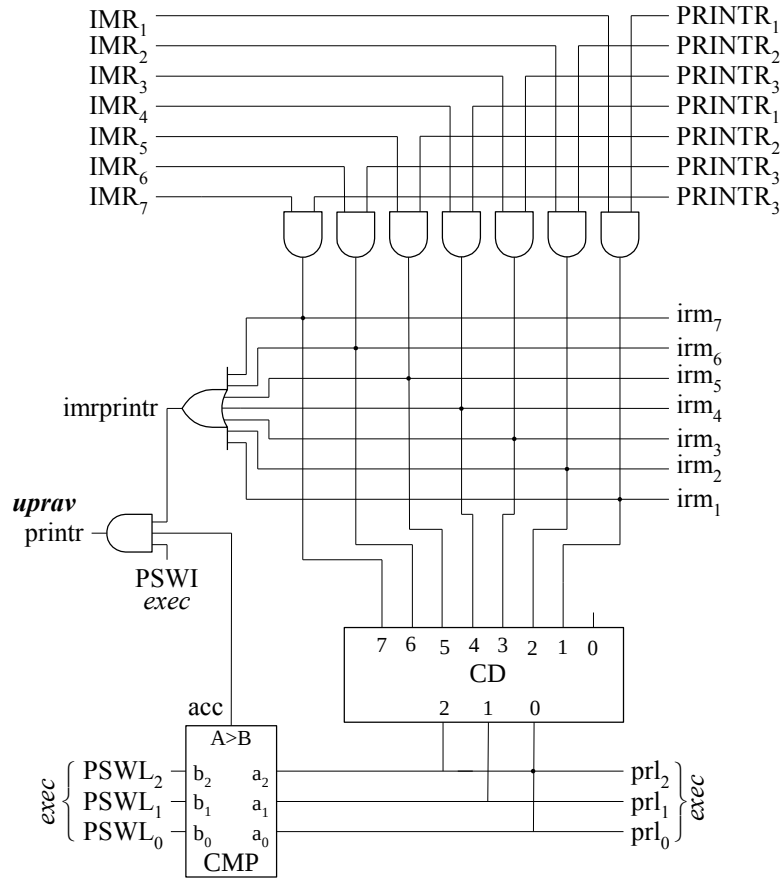
Dekoder DC u okviru opsluživanja spoljašnjeg maskirajućeg zahteva za prekid pri vrednosti 1 signala **clINTR** daje vrednost 1 jednog od signala **clINTR₁** do **clINTR₇**. Koji će od signala **clINTR₁** do **clINTR₇** tada imati vrednost 1 zavisi od vrednosti signala **prl₂** do **prl₀** koji daju binarnu vrednost linije intr₁ do intr₇ po kojoj je prihvaćeni zahtev za prekid stigao.

Registar maske IMR_{15...0} služi za pojedinačno maskiranje spoljašnjih maskirajućih zahteva za prekid intr₁ do intr₇. Zahtevima za prekid intr₁ do intr₇ odgovaraju razredi IMR₁ do IMR₇ registra maske IMR_{15...0}, dok se preostali razredi ne koriste. Upis sadržaja sa linija AW_{15...0} se obavlja ukoliko signal **ldIMR** ima vrednost 1. Upis u registar IMR_{15...0} se realizuje samo kod izvršavanja instrukcije STIMR, koja služi za upis sadržaja akumulatora AW_{15...0} u registar procesora IMR_{15...0}.

Kombinacione prekidačke mreže za formiranje signala spoljašnjih maskirajućih prekida **printr** se sastoji od logičkih ILI i I elemenata, koda CD i komparatora CMP (slika 28).

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 **irm₁** do **irm₇** ima vrednost 1. Ovo će se desiti ukoliko se barem u jednom od flip-flopova PRINTR₁ do PRINTR₇ (slika 27) nalazi vrednost 1 i ukoliko je u odgovarajućem razredu IMR₁ do IMR₇ 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 **prl₂** do **prl₀** na izlazu koda prioriteta CD, dok je prioritet tekućeg programa određenog signalima **PSWL₂** do **PSWL₀** registra PSW_{15...0} (slika 28). 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 24).



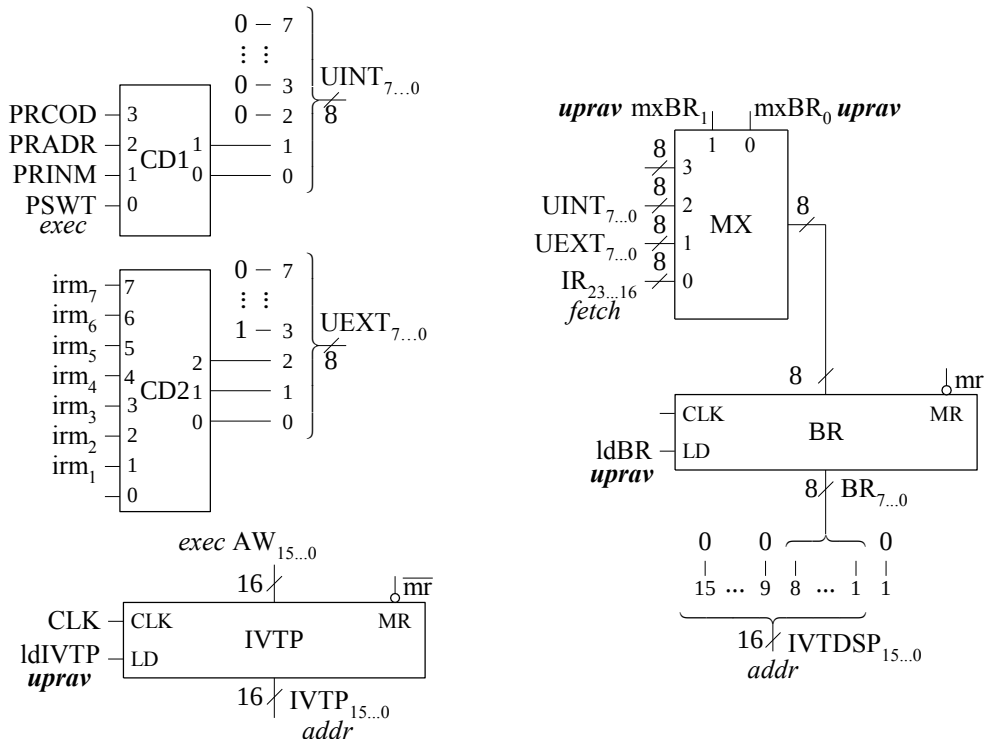
Slika 28 Blok intr (drugi deo)

Kombinacione i sekvencijalne mreže za formiranje pomeraja IVTDSP_{15...0} za tabelu sa adresama prekidnih rutina se sastoje od koda CD1 i CD2, i registra BR_{7...0} sa multiplekserom MX (slika 29).

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, PRADR za unutrašnji prekid zbog korišćenja neposrednog adresiranja za određeni operand 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 koda prioriteta CD1 po rastućim prioritetima. Na izlazu koda 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 koda 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 intr₁ do intr₇. Signali spoljašnjih maskirajućih prekida intr₁ do intr₇ posle eventualnog maskiranja razredima IMR₁ do IMR₇ registra maske IMR_{15...0} pojavljuju se kao signali **irm₁** do **irm₇** (slika 28). Ovi signali se vode na ulaze 1 do 7 koda

prioriteta CD2 po rastućim prioritetima. Na izlazu kodera dobija se broj zahteva za prekid najvišeg prioriteta binarno kodiran sa tri bita koji predstavljaju tri najniža bita broja ulaza. Bit 3, koji ima vrednost 1 i bitovi 4 do 15, koji imaju vrednost 0, sa tri bita na izlazu kodera CD2 formiraju 8-mo bitnu vrednost broja ulaza UEXT_{7...0}. Ovim se dobijaju brojevi ulaza od 9 do 15.



Slika 29 Blok intr (treći deo)

Registar BR_{7...0} sa multiplexerom MX služi za selekciju i pamćenje broja ulaza u tabelu sa adresama prekidnih rutina (slika 29). Upis sadržaja sa izlaza multiplexera 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 prekidnih 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.

Multiplexer MX je 8-mo razredni multiplexer sa 4 ulaza. Na ulaze 0 do 2 multiplexera se vode sadržaji IR_{23...16}, 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 **mxBR₁** i **mxBR₀**. Sadržaj IR_{23...16} se propušta kroz multiplexer za instrukciju prekida INT, UEXT_{7...0} za spoljašnje maskirajuće prekide i UINT_{7...0} za unutrašnje prekide i spoljašnji nemaskirajući prekid. Sadržaj sa izlaza multiplexera MX se vodi na ulaze registra BR_{7...0}.

Registar IVTP_{15...0} je ukazivač na tabelu sa adresama prekidnih rutina i sadrži početnu adresu tabele. Upis sadržaja sa linija AW_{15...0} u registar IVTP_{15...0} se obavlja vrednošću 1 signala **ldIVTP**. Upis u registar IMR_{15...0} se realizuje samo kod izvršavanja instrukcije STIVTP, koja služi za upis sadržaja akumulatora AW_{15...0} u registar procesora IVTP_{15...0}.

4.2 UPRAVLJAČKA JEDINICA

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

4.2.1 Dijagram toka izvršavanja instrukcija

Dijagram toka izvršavanja instrukcija je predstavljen operacionim i uslovnim blokovima (slika 30). 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. 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. 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 postoji greška adresiranja. Greška adresiranja postoji samo ukoliko se radi o instrukciji upisa za koju je zadato neposredno adresiranja, što se utvrđuje na osnovu koda operacije iz prvog bajta instrukcije i načina adresiranja iz prva tri bita drugog bajta instrukcije. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu opsluživanje prekida.

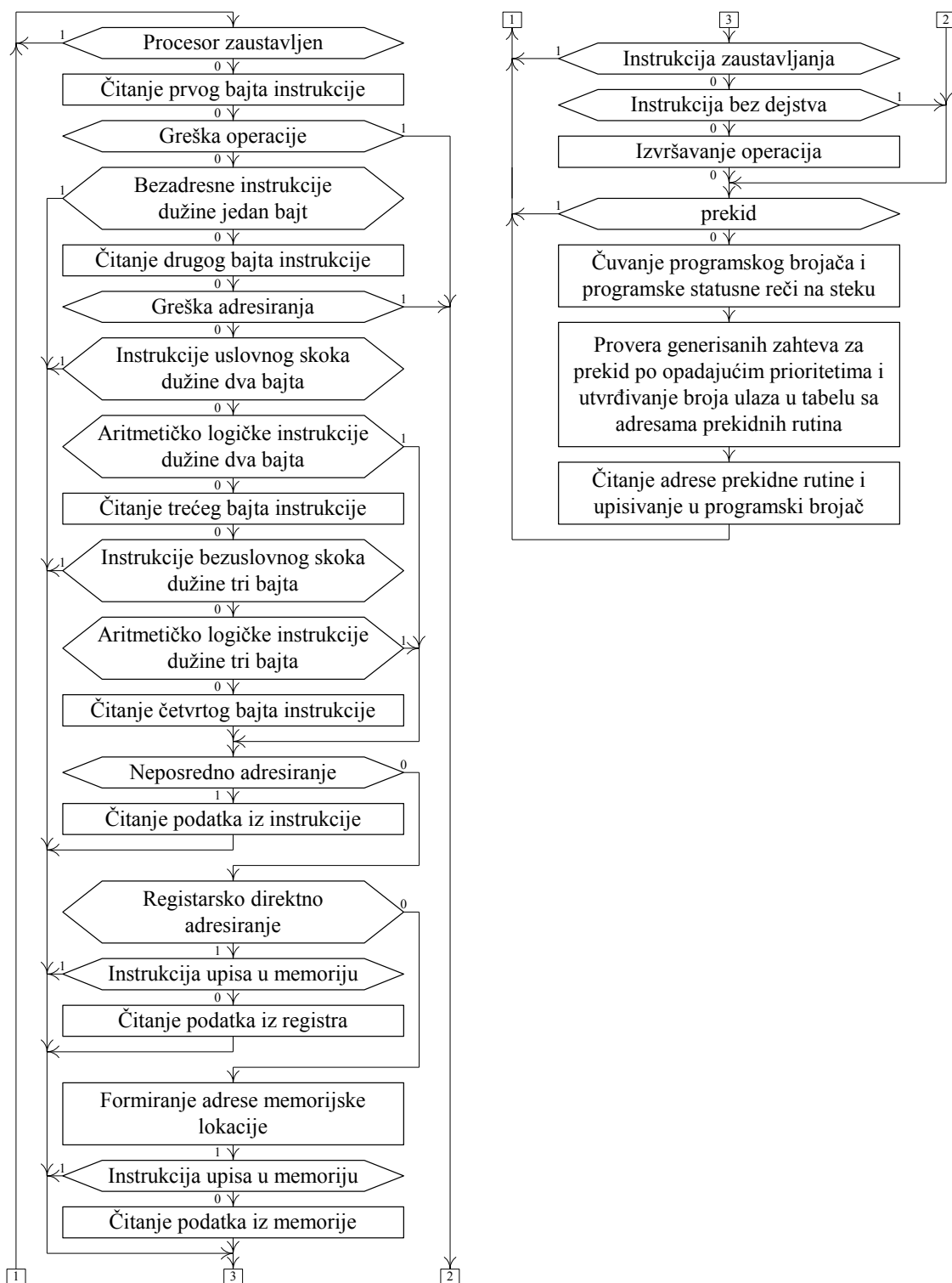
U nastavku faze čitanje instrukcije proverava se da li vrednost koda operacije odgovara vrednostima instrukcija uslovnog skoka čija je dužina dva bajta. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu izvršavanje operacije.

U nastavku faze čitanje instrukcije proverava se da li vrednost koda operacije odgovara vrednostima aritmetičko logičkih instrukcija i da li vrednost načina adresiranja odgovara vrednostima za direktno registarsko adresiranje i indirektno registarsko adresiranje čija je dužina dva bajta. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu formiranje adrese operanda.

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

U nastavku faze čitanje instrukcije proverava se da li vrednost koda operacije odgovara vrednostima aritmetičko logičkih instrukcija i da li vrednosti načina adresiranja odgovara vrednosti za neposredno adresiranje čija je dužina tri bajta. Ukoliko nije nastavlja se sa fazom čitanje instrukcije, a ukoliko jeste prelazi na fazu formiranje adrese operanda.

U nastavku faze čitanje instrukcije čita se četvrti bajt instrukcije i prelazi na fazu formiranje adrese operanda.



Slika 30 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 proverava 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 akumulator 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.

4.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 30) i dat je u obliku dijagrama toka mikrooperacija, dijagrama toka upravljačkih signala (slike 31 do 57) i sekvence upravljačkih signala (tabela 16).

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})).

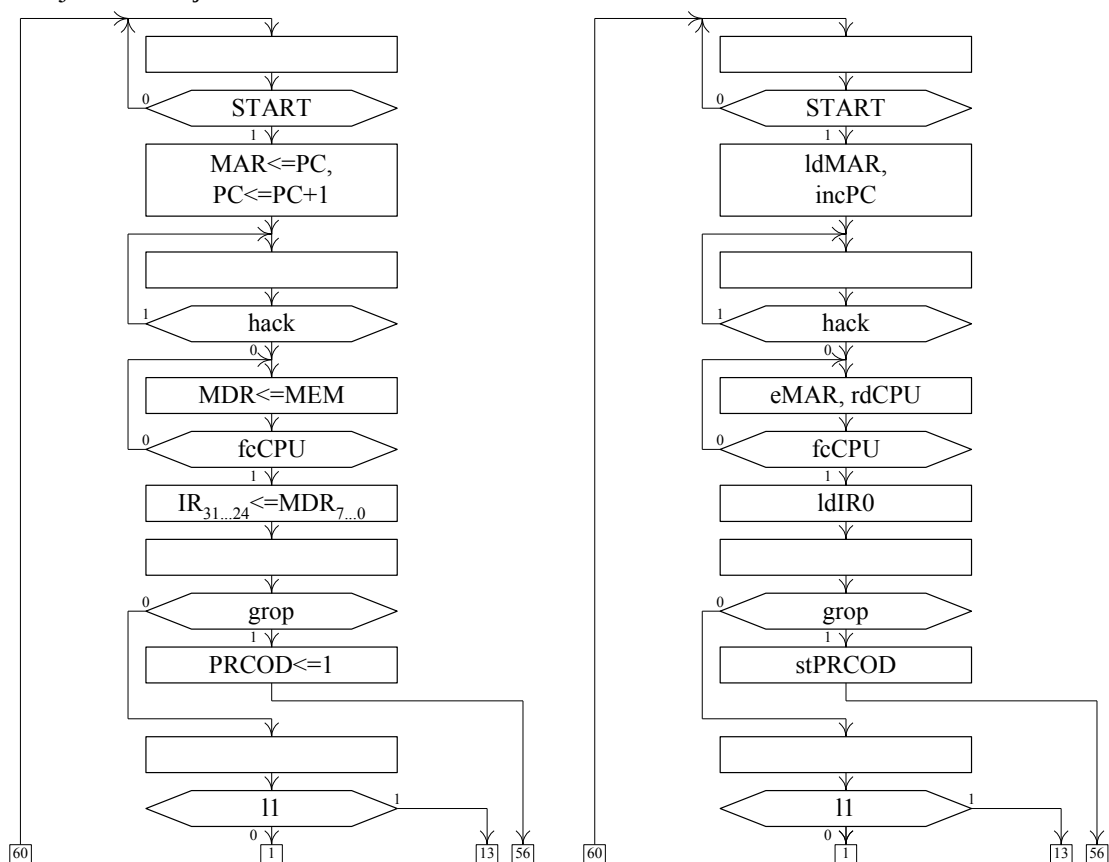
Prvi iskaz sadrži korak step_A na koji treba bezuslovno preći i u daljem tekstu se referiše kao bezuslovni skok. Drugi iskaz sadrži signal **uslov** i korak step_A i određuje korak step_A na koji treba preći ukoliko signal **uslov** ima vrednost 1 i u daljem tekstu se referiše kao uslovni skok. Treći 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 16 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 step₀₀ do step₁₈ koji su zajednički za sve instrukcije. Faza formiranje adrese i čitanje operanda se realizuje u koracima step₂₀ do step₄₃ pri čemu postoje posebni koraci za svaki način adresiranja operanda. Faza izvršavanje operacije se realizuje u koracima step₅₀ do step_{BE}, pri čemu postoje posebni koraci za svaku operaciju. Faza opsluživanje prekida se realizuje u koracima step_{C0} do step_{E3}. !

! Čitanje instrukcije !



Slika 31 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 **START** then step_{00});*

! U koracima step_{01} do step_{04} se čita prvi bajt instrukcije i smešta u razrede $\text{IR}_{31...24}$ prihvatnog registra instrukcije $\text{IR}_{31...0}$. Vrednostima 0 signala **mxMAR₂**, **mxMAR₁** i **maMAR₀** bloka *bus* se sadržaj registra $\text{PC}_{15..0}$ bloka *fetch* propušta kroz multiplexer MX1 i vrednošću 1 signala **ldMAR** upisuje u adresni registar $\text{MAR}_{15..0}$ bloka *bus*. Pored toga, vrednošću 1 signala **incPC** se sadržaj registra $\text{PC}_{15..0}$ inkrementira. U koraku step_{02} se proverava vrednost signala **hack** bloka *bus* koji vrednostima 1 i 0 ukazuje da li je processor prepustio magistralu kontroleru sa direktnim pristupom memoriji ili je magistrala u posedu procesora, respektivno. Ukoliko je procesor prepustio magistralu ostaje se u koraku step_{02} , dok se u suprotnom slučaju prelazi na korak step_{03} . U koraku step_{03} se realizuje čitanje jednog bajta i upisivanje u registar podatka $\text{MDR}_{7..0}$ bloka *bus*. Vrednošću 1 signala **eMAR** se sadržaj registra $\text{MAR}_{15..0}$ pušta na adresne linije $\text{ABUS}_{15..0}$ magistrale **BUS**. Pored toga se vrednošću 1 signala **rdCPU** generiše vrednost 0 signala na upravljačkoj liniji **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 $\text{ABUS}_{15..0}$ i na liniji **RDBUS** je vrednost 0. Po završenom čitanju memorija **MEM** pušta pročitani sadržaj na linije podataka $\text{DBUS}_{7..0}$ magistrale **BUS**. Radi indikacije da je sadržaj na linijama $\text{DBUS}_{7..0}$ važeći memorija generiše vrednost 0 signala na upravljačkoj liniji **FCBUS** magistrale **BUS**. U koraku step_{03} se proverava vrednost signala **fcCPU** bloka *bus* koji vrednostima 0 i 1 ukazuje da čitanje nije završeno i da je čitanje završeno, respektivno. Ukoliko čitanje nije završeno ostaje se u koraku step_{03} , dok se u suprotnom slučaju sadržaj sa linija $\text{DBUS}_{7..0}$ upisuje u registar podatka $\text{MDR}_{7..0}$ i prelazi na korak step_{04} . U koraku step_{04} se vrednošću 1 signala **ldIRO** bloka *fetch* pročitani bajt prebacuje iz registra $\text{MDR}_{7..0}$ bloka *bus* u prihvatni registar instrukcije i to u razrede $\text{IR}_{31...24}$ bloka *bus*. !

step_{01} **ldMAR, incPC;**

step_{02} *br (if **hack** then step_{02});*

step_{03} **eMAR, rdCPU,**

*br (if **fcCPU** then step_{03});*

step_{04} **ldIRO;**

! U koraku step_{05} se proverava vrednost signala **gopr** bloka *fetch* da bi se utvrdilo da li prvi bajt instrukcije sadrži korektnu vrednost koda operacije. U zavisnosti od toga da li signal **gopr** 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_{C0} radi utvrđivanja adrese prekidne rutine. !

step_{05} *br (if **gopr** then step_{07});*

step_{06} **stPRCOD,**

br step_{C0} ;

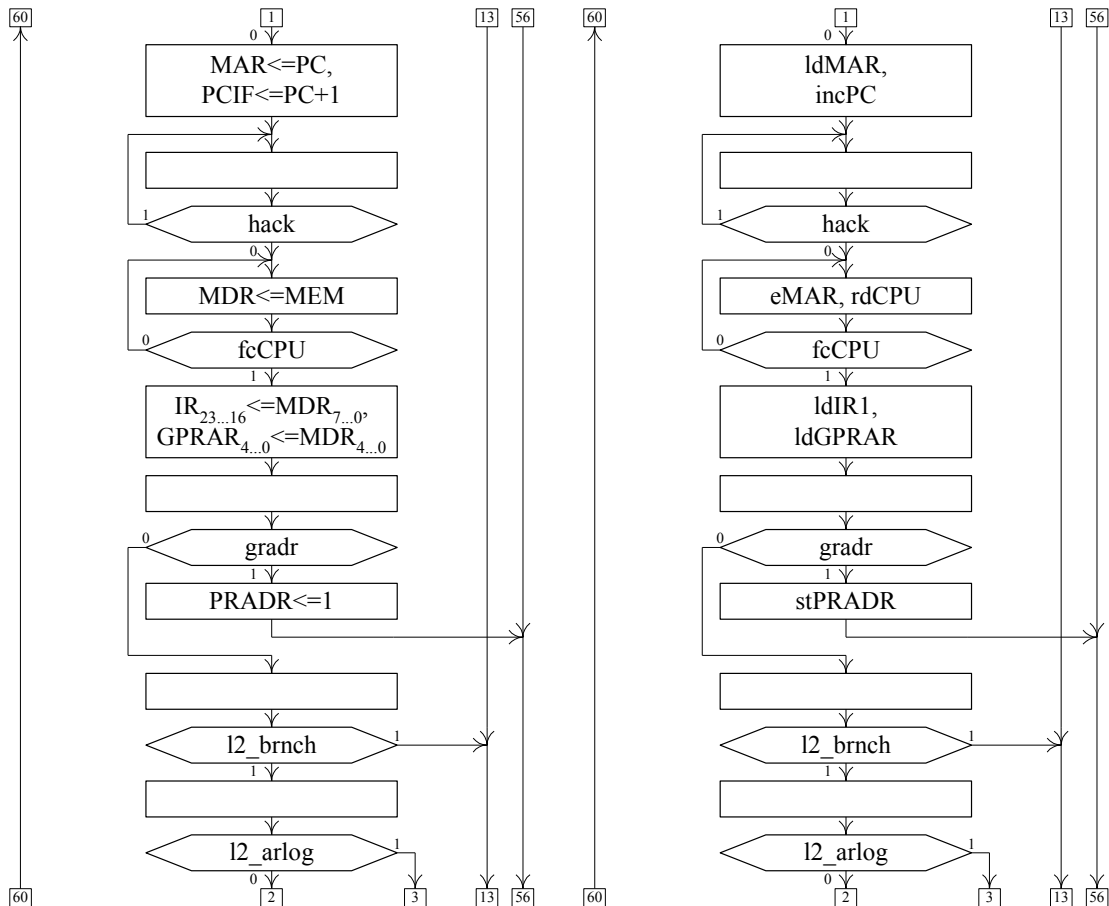
! U koraku step_{07} se proverava vrednost signala **I1** bloka *fetch* da bi se utvrdilo da li je dužina instrukcije jedan bajt ili više bajtova. U zavisnosti od toga da li signal **I1** ima vrednost 1 ili 0, prelazi se na korak step_{50} i fazu izvršavanje operacije ili step_{07} i produžava sa čitanjem bajtova instrukcije. !

step_{07} *br (if **I1** then step_{50});*

! U koracima step_{08} do step_{0B} se čita drugi bajt instrukcije i smešta u razrede $\text{IR}_{23...16}$ prihvatnog registra instrukcije $\text{IR}_{31...0}$. Koraci step_{08} do step_{0A} 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 $\text{MDR}_{7..0}$ bloka *bus* u prihvatni registar instrukcije i to u razrede $\text{IR}_{23...16}$ bloka *bus*. Istovremeno se vrednošću 1 signala **ldGPRADR** bloka *addr* razredi $\text{MDR}_{4..0}$ upisuju u registar $\text{GPRADR}_{4..0}$. Time se u registru $\text{GPRADR}_{4..0}$ nalazi adresa registra opšte

namene koja se koristi samo u slučaju da se radi o instrukciji kojoj je potrebna ta adresa da bi se saglasno specificiranom načinu adresiranja došlo do operanda. !

step₀₈ **ldMAR, incPC;**
 step₀₉ *br (if **hack** then step₀₉);*
 step_{0A} **eMAR, rdCPU,**
*br (if **fcCPU** then step_{0A});*
 step_{0B} **ldIR1, ldGPRADR;**



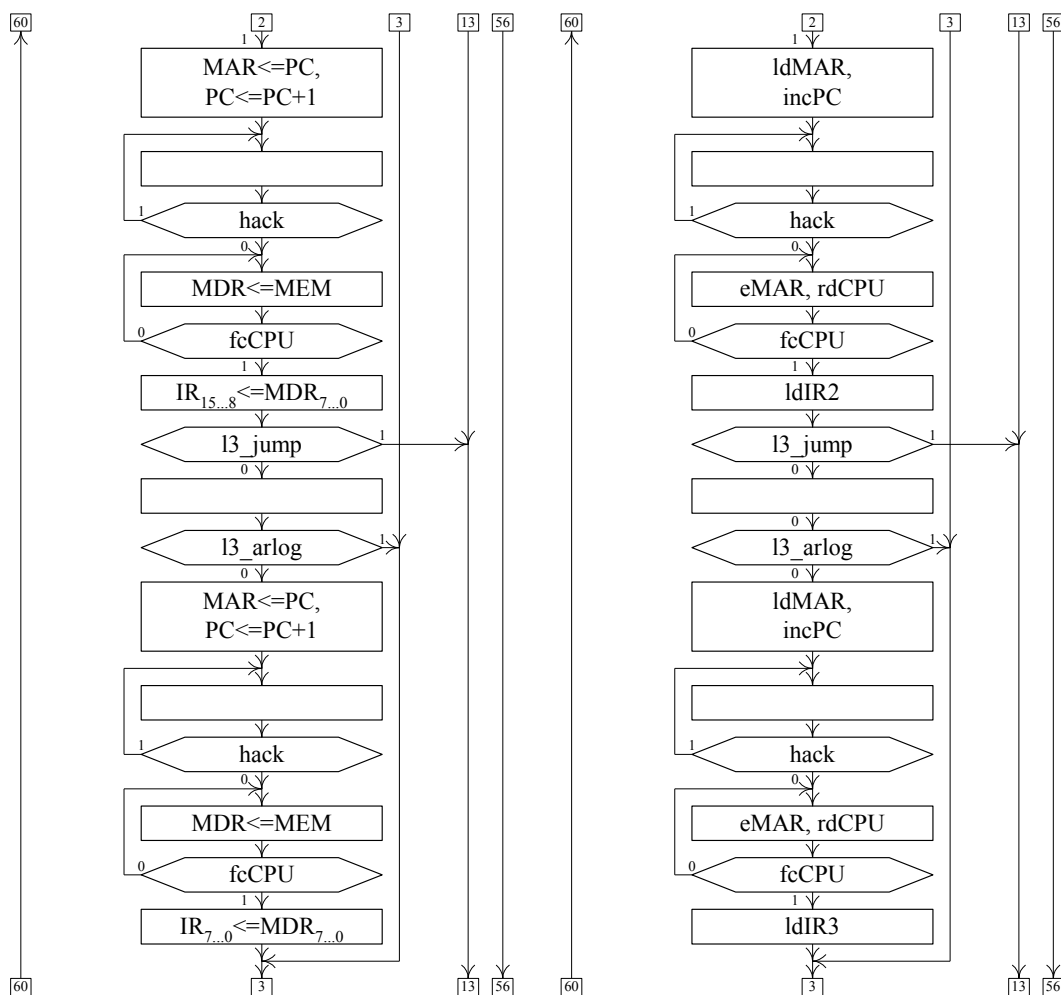
Slika 32 Čitanje instrukcije (drugi deo)

! U koraku step_{0C} se proverava vrednost signala **gradr** bloka *fetch* da bi se utvrdilo da li je nekorektno adresiranje specificirano. Ovo se dešava ukoliko se za instrukcije STB i STW specificira neposredno adresiranje. U zavisnosti od toga da li signal **gradr** ima vrednost 1 ili 0, prelazi se na korak step_{0D} ili step_{0E}, respektivno. Ukoliko je nekorektno adresiranje specificirano, u koraku step_{0D} se vrednošću 1 signala **stPRADR** bloka *intr* u flip-flop **PRADR** upisuje vrednost 1 i bezuslovno prelazi na korak step_{C0} radi utvrđivanja adrese prekidne rutine. !

step_{0C} *br (if **gradr** then step_{0E});*
 step_{0D} **stPRADR,**
br step_{C0};

! U koracima step_{0E} i step_{0F} 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_{0E} se u zavisnosti od toga da li signal **l2_brnch** ima vrednost 1 ili 0, prelazi na korak step₅₀ i fazu izvršavanje operacije ili step_{0F} i proveru vrednosti signala **l2_arlog**. Iz koraka step_{0F} se u zavisnosti od toga da li signal **l2_arlog** ima vrednost 1 ili 0, prelazi na korak step₂₀ i fazu formiranje adrese i čitanje operanda ili step₁₀ i produžava sa čitanjem bajtova instrukcije.!

step_{0E} *br (if **l2_brnch** then step₅₀);*
 step_{0F} *br (if **l2_arlog** then step₂₀);*



Slika 33 Čitanje instrukcije (treći deo)

! U koracima step₁₀ do step₁₂ se čita treći bajt instrukcije i smešta u razrede IR_{15...8} prihvatnog registra instrukcije IR_{31...0}. Koraci step₁₀ do step₁₂ u kojima se čita treći bajt instrukcije su isti kao koraci step₀₁ do step₀₃ u kojima se čita prvi bajt instrukcije.!

```

step10  ldMAR, incPC;
step11  br (if hack then step11);
step12  eMAR, rdCPU,
        br (if fcCPU then step12);

```

! U koraku step₁₃ se vrednošću 1 signala **ldIR2** 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*. !

! U koracima step₁₃ i step₁₄ se proverava vrednost signala **l3_jump** i **l3_arlog** bloka *fetch* da bi se utvrdilo da li je dužina instrukcije tri ili četiri bajta. Iz koraka step₁₃ se u zavisnosti od toga da li signal **l3_jump** ima vrednost 1 ili 0, prelazi na korak step₅₀ i fazu izvršavanje operacije ili step₁₄ i proveru vrednosti signala **l3_arlog**. Iz koraka step₁₄ se u zavisnosti od toga da li signal **l3_arlog** ima vrednost 1 ili 0, prelazi na korak step₂₀ i fazu formiranje adrese i čitanje operanda ili step₁₅ i produžava sa čitanjem četvrtog bajta instrukcije.!

```

step13  ldIR2,
        br (if l3_jump then step50);
step14  br (if l3_arlog then step20);

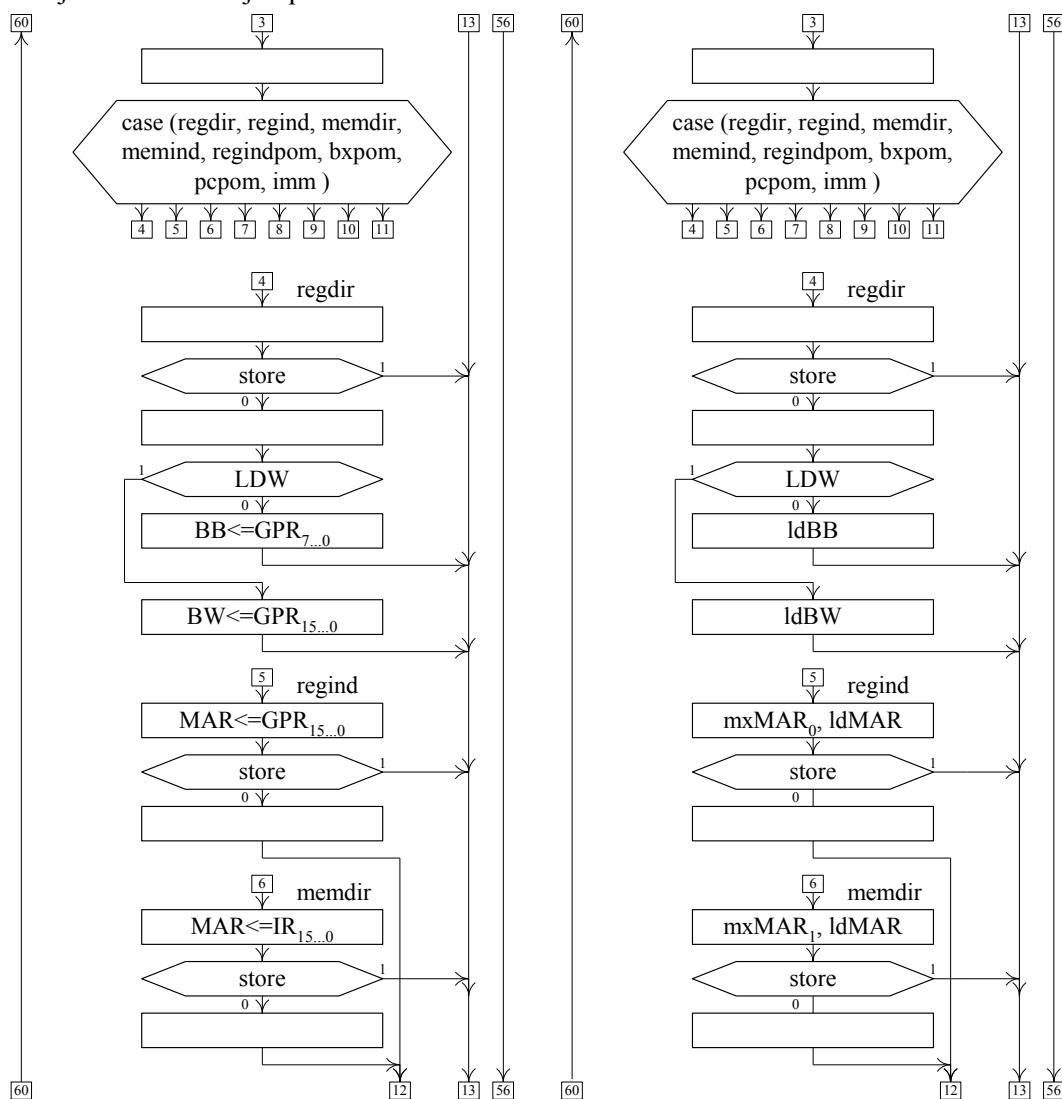
```

! U koracima step₁₅ do step₁₇ se čita četvrti bajt instrukcije i smešta u razrede IR_{7...0} prihvatnog registra instrukcije IR_{31...0}. Koraci step₁₅ do step₁₇ u kojima se čita četvrti bajt instrukcije su isti kao koraci step₀₁ do step₀₃ u kojima se čita prvi bajt instrukcije. U koraku step_{0B} se vrednošću 1 signala **ldIR3** 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*. Iz koraka step₁₈ se bezuslovno prelazi na korak step₂₀ i fazu formiranje adrese i čitanje operanda. !

step₁₅ **ldMAR, incPC;**
 step₁₆ **br (if hack then step₁₆);**
 step₁₇ **eMAR, rdCPU,**
 br (if fcCPU then step₁₇);
 step₁₈ **ldIR3,**
 br step₂₀;

! Formiranje adrese i čitanje operanda !



Slika 34 Formiranje adrese i čitanje operanda (prvi deo)

! U korak step₂₀ se dolazi iz koraka step_{0F}, step₁₄ i step₁₈ ukoliko se radi o instrukcijama dužine dva, tri i četiri bajta koje zahtevaju da se do operanda dođe saglasno specificiranom načinu adresiranja. Za sve instrukcije, sem instrukcija STB i STW, operand se smešta u registar $BB_{7...0}$ ili $BW_{15...0}$. Operand može da bude u nekom od registara opšte namene, u memorijskoj lokaciji ili samoj instrukciji. U slučaju adresiranja kod kojih se operand nalazi u nekom od registara opšte namene ili u samoj instrukciji, ova faza se svodi na prebacivanje operanda u registar $BB_{7...0}$ ili $BW_{15...0}$. 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 su instrukcije STB i STW kod kojih 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. Slučaj kada se upisuje u samu instrukciju se otkriva još u fazi čitanje instrukcije i to kod čitanja drugog bajta instrukcije, pa ovaj slučaj ne može da se javi u ovoj fazi. U koraku $step_{20}$ se realizuje višestruki uslovni skok na jedan od koraka $step_{21}$, $step_{25}$, ..., $step_{41}$ u zavisnosti od toga koji od signala adresiranja **regdir**, **regind**, **memdir**, **memind**, **regindpom**, **bxpom**, **bcpom**, **imm** bloka *addr* ima vrednost 1. !

```
step20  br (case (regdir, regind, memdir, memind, regindpom, bxpom, bcpom, imm) then
|         (regdir,  $step_{21}$ ), (regind,  $step_{25}$ ), (memdir,  $step_{27}$ ), (memind,  $step_{29}$ ),
|         (regindpom,  $step_{32}$ ), (bxpom,  $step_{34}$ ), (pcpom,  $step_{37}$ ), (imm,  $step_{41}$ ));
```

! Registarsko direktno adresiranje !

! U korak $step_{21}$ se dolazi iz $step_{20}$ ukoliko signal za registarsko direktno adresiranje **regdir** ima vrednost 1. Ukoliko signal **store** bloka *addr* ima vrednost 1, što znači da se radi o instrukciji STB ili instrukciji STW za koje nema čitanja operanda, prelazi se na korak $step_{50}$ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak $step_{22}$ u kome se proverava vrednost signala operacije **LDW**. Ukoliko signal **LDW** ima vrednost 1, prelazi se na korak $step_{24}$ u kome se vrednošću 1 signala **ldBW** 16-to razredni sadržaj adresiranog registra opšte namene $GPR_{15...0}$ bloka *addr* upisuje u registar $BW_{15...0}$ bloka *exec*. U suprotnom slučaju se prelazi na sledeći korak u kome se vrednošću 1 signala **ldBB** niži bajt adresiranog registra opšte namene $GPR_{7...0}$ upisuje u registar $BB_{7...0}$ bloka *exec*. U oba slučaja se prelazi na korak $step_{50}$ i fazu izvršavanje operacije. !

```
step21  br (if store then  $step_{50}$ );
step22  br (if LDW then  $step_{24}$ );
step23  ldBB,
|       br  $step_{50}$ ;
step24  ldBW,
|       br  $step_{50}$ ;
```

! Registarsko indirektno adresiranje !

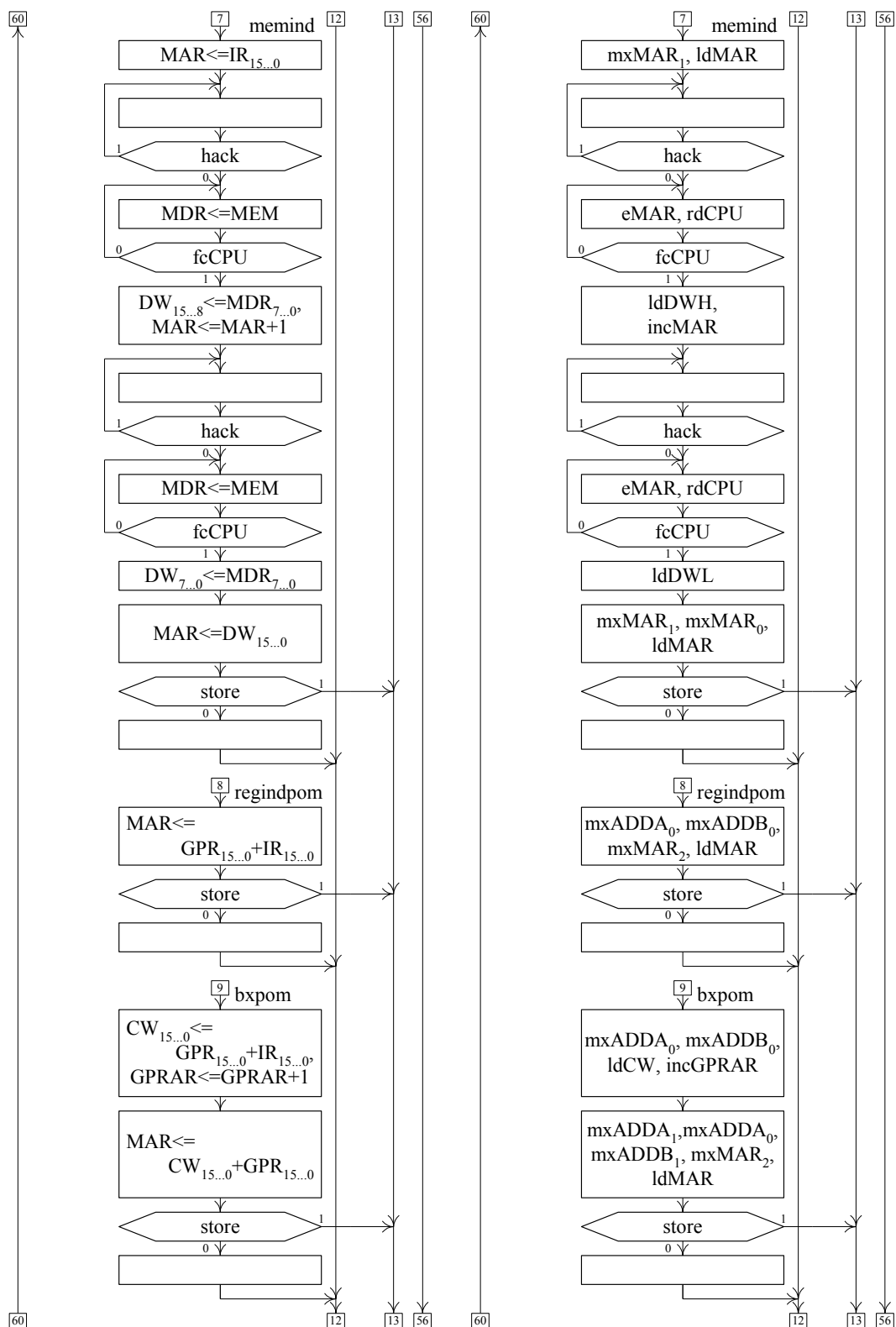
! U korak $step_{25}$ se dolazi iz $step_{20}$ ukoliko signal za registarsko indirektno adresiranje **regind** ima vrednost 1. Vrednošću 1 signala **mxMAR₀** se sadržaj adresiranog registra opšte namene $GPR_{15...0}$ bloka *addr* propušta kroz multiplexer MPX1 bloka *bus* i vrednošću 1 signala **ldMAR** upisuje u registar $MAR_{15...0}$ bloka *bus*. Time se u registru $MAR_{15...0}$ nalazi adresa operanda za slučaj registarskog indirektnog adresiranja. Ukoliko signal **store** bloka *addr* ima vrednost 1, što znači da se radi o instrukciji STB ili instrukciji STW za koje nema čitanja operanda, prelazi se na korak $step_{50}$ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak $step_{26}$ iz koga se bezuslovno prelazi na korak $step_{38}$ i čitanje operanda. !

```
step25  mxMAR0, ldMAR,
|       br (if store then  $step_{50}$ );
step26  br  $step_{38}$ ;
```

! Memorijsko direktno adresiranje !

! U korak $step_{27}$ se dolazi iz $step_{20}$ ukoliko signal za memorijsko direktno adresiranje **memdir** ima vrednost 1. Vrednošću 1 signala **mxMAR₁** se sadržaj registra $IR_{15...0}$ bloka *fetch* propušta kroz multiplexer MX1 bloka *bus* 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 STB ili instrukciji STW za koje nema čitanja operanda, prelazi se na korak $step_{50}$ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak $step_{28}$ iz koga se bezuslovno prelazi na korak $step_{38}$ i čitanje operanda. !

```
step27  mxMAR1, ldMAR,
|       br (if store then  $step_{50}$ );
step28  br  $step_{38}$ ;
```



Slika 35 Formiranje adrese i čitanje operanda (drugi deo)

! Memorijsko indirektno adresiranje !

! U korak step₂₉ se dolazi iz step₂₀ ukoliko signal za memorijsko indirektno adresiranje **memind** ima vrednost 1. Vrednošću 1 signala **mxMAR₁** se sadržaj registra IR_{15..0} bloka *fetch* propušta kroz multiplexer MX1 bloka *bus* 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. Čitanje

prvog bajta se realizuje u koracima step_{2A} i step_{2B}, a drugog bajta u koracima step_{2D} i step_{2E}, na isti način kao u koracima step₀₂ i step₀₃, kod čitanja prvog bajta instrukcije. U koraku step_{2C} se iz registra MDR_{7...0} bloka *bus* prvi bajt vrednošću 1 signala **ldDWH** upisuje u viši bajt registra DW_{15...0} bloka *bus*, a vrednošću 1 signala **incMAR** adresni registar MAR_{15...0} bloka *bus* inkrementira na adresu sledećeg bajta. U koraku step_{2F} se iz registra MDR_{7...0} bloka *bus* drugi bajt vrednošću 1 signala **ldDWL** upisuje u niži bajt registra DW_{15...0}. Na kraju se u koraku step₃₀ vrednostima 1 signala **mxMAR₁** i **mxMAR₀** sadržaj registra DW_{15...0} koji predstavlja adresu operanda propušta kroz multiplekser MX1 bloka *bus* 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 STB ili instrukciji STW za koje nema čitanja operanda, prelazi se na korak step₅₀ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak step₃₁ iz koga se bezuslovno prelazi na korak step₃₈ i čitanje operanda. !

```

step29  mxMAR1, ldMAR;
step2A  br (if hack then step2A);
step2B  eMAR, rdCPU,
        br (if fcCPU then step2B);
step2C  ldDWH, incMAR;
step2D  br (if hack then step2D);
step2E  eMAR, rdCPU,
        br (if fcCPU then step2E);
step2F  ldDWL;
step30  mxMAR1, mxMAR0, ldMAR,
        br (if store then step50);
step31  br step38;

```

! Registarsko indirektno adresiranje sa pomerajem !

! U korak step₃₂ se dolazi iz step₂₀ ukoliko signal za registarsko indirektno adresiranje sa pomerajem **regindpom** ima vrednost 1. Vrednostima 1 signala **mxADDA₀** i **mxADDB₀** se najpre kroz multipleksere MX2 i MX3 na ulaze sabirača ADD bloka *addr* propuštaju adresirani registar opšte namene GPR_{15...0} bloka *addr* i pomeraj iz IR_{15...0} bloka *fetch*, zatim se vrednošću 1 signala **mxMAR₂** dobijeni sadržaja sa izlaza sabirača ADD propušta kroz multiplekser MX1 bloka *bus* i na kraju vrednošću 1 signala **ldMAR** upisuje u registar MAR_{15...0}. Time se u registru MAR_{15...0} nalazi adresa operanda za slučaj registarskog indirektnog adresiranja sa pomerajem. Ukoliko signal **store** bloka *addr* ima vrednost 1, što znači da se radi o instrukciji STB ili instrukciji STW za koje nema čitanja operanda, prelazi se na korak step₅₀ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak step₃₃ iz koga se bezuslovno prelazi na korak step₃₈ i čitanje operanda. !

```

step32  mxADDA0, mxADDB0, mxMAR2, ldMAR,
        br (if store then step50);
step33  br step38;

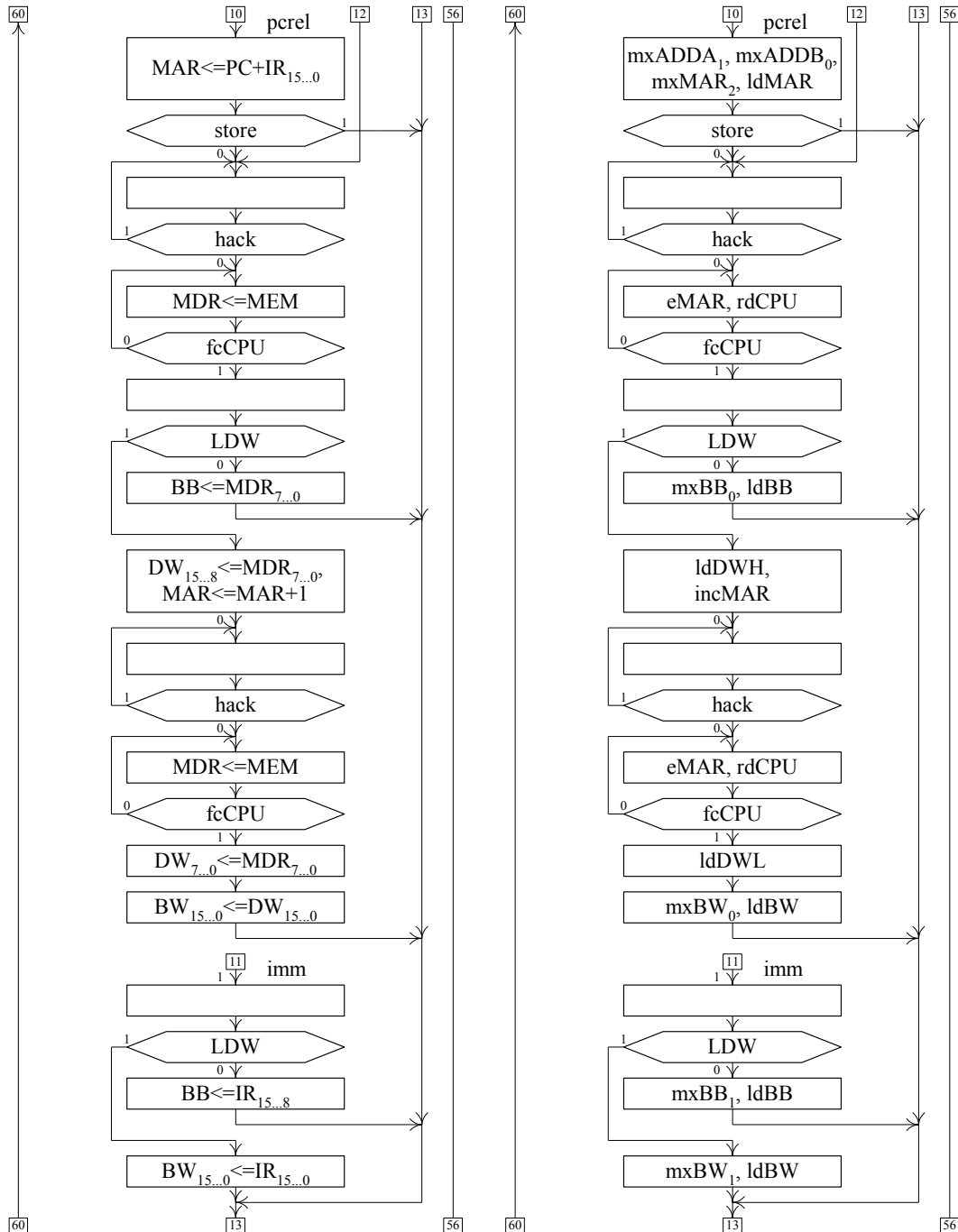
```

! Bazno indeksno adresiranje sa pomerajem !

! U korak step₃₄ se dolazi iz step₂₀ ukoliko signal za bazno indeksno adresiranje sa pomerajem **bxpom** ima vrednost 1. U koraku step₃₄ se vrednostima 1 signala **mxADDA₀** i **mxADDB₀** kroz multipleksere MX2 i MX3 na ulaze sabirača ADD bloka *addr* propuštaju adresirani registar opšte namene GPR_{15...0} bloka *addr* i pomeraj iz IR_{15...0} bloka *fetch*, a vrednošću 1 signala **ldCW** sadržaj ADD_{15...0} sa izlaza sabirača ADD upisuje u registar CW_{15...0} bloka *addr*. Pored toga vrednošću 1 signala **incGPRAR** bloka *addr* se vrši inkrementiranje adresnog registra GPRAR_{4...0} registara opšte namene GPR_{15...0}. U koraku step₃₅ se vrednostima 1 signala **mxADDA₁**, **mxADDA₀** i **mxADDB₁** kroz multipleksere MX2 i MX3 na ulaze sabirača ADD propuštaju sadržaji registra CW_{15...0} i adresiranog registra opšte namene GPR_{15...0}, vrednošću 1 signala **mxMAR₂** se dobijeni sadržaj ADD_{15...0} sa izlaza sabirača ADD propušta kroz multiplekser MX1 bloka *bus* i vrednošću 1 signala **ldMAR** upisuje u registar MAR_{15...0} bloka *bus*. Time se u registru MAR_{15...0} nalazi adresa operanda za slučaj bazno indeksnog adresiranja sa pomerajem. Ukoliko signal **store** bloka *addr* ima vrednost 1, što znači da se radi o instrukciji STB ili instrukciji STW za koje nema čitanja operanda, prelazi se na korak step₅₀ i fazu izvršavanje operacije.

U suprotnom slučaju se prelazi na korak step₃₆ iz koga se bezuslovno prelazi na korak step₃₈ i čitanje operanda. !

step₃₄ **mxADDA₀, mxADDB₀, ldcW, incGPRAR;**
 step₃₅ **mxADDA₁, mxADDA₀, mxADDB₁, mxMAR₂, ldMAR,**
br (if store then step₅₀);
 step₃₆ **br step₃₈;**



Slika 36 Formiranje adrese i čitanje operanda (treći deo)

! PC relativno adresiranje !

! U korak step₃₇ se dolazi iz step₂₀ ukoliko signal za PC relativno adresiranje **pcpom** ima vrednost 1. Vrednostima 1 signala **mxADDA₁** i **mxADDB₀** se kroz multiplexere MX2 i MX3 na ulaze sabirača ADD bloka *addr* propuštaju programski brojač PC_{15...0} bloka *fetch* i pomeraj iz IR_{15...0} bloka *fetch*, vrednošću 1 signala **mxMAR₂** se dobijeni sadržaj ADD_{15...0} sa izlaza sabirača ADD propušta kroz multiplexer MX1 bloka *bus* i vrednošću 1 signala **ldMAR** upisuje u registar MAR_{15...0} bloka *bus*.

Time se u registru $MAR_{15...0}$ nalazi adresa operanda za slučaj PC relativnog adresiranja. Ukoliko signal **store** bloka *addr* ima vrednost 1, što znači da se radi o instrukciji STB ili instrukciji STW za koje nema čitanja operanda, prelazi se na korak $step_{50}$ i fazu izvršavanje operacije. U suprotnom slučaju se prelazi na korak $step_{38}$ i čitanje operanda sa adrese koja se nalazi u registru $MAR_{15...0}$!

$step_{37}$ **mxADDA₁, mxADDB₀, mxMAR₂, ldMAR,**

br (if store then $step_{50}$);

! Čitanje operanda !

! U korak $step_{38}$ se dolazi iz $step_{26}$ kod registarskog indirektnog adresiranja, iz $step_{28}$ kod memorijskog direktnog adresiranja, iz $step_{31}$ kod memorijskog indirektnog adresiranja, iz $step_{33}$ kod registarskog indirektnog adresiranja sa pomerajem, iz $step_{36}$ kod baznog indeksnog adresiranja sa pomerajem i iz $step_{37}$ kod PC relativnog adresiranja sa pomerajem. 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 jedan bajt sem za instrukciju LDW za koju je dužina operanda dva bajta. Zbog toga se najpre u koracima $step_{38}$ i $step_{39}$ 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 $step_{3A}$ u zavisnost od toga da li signal **LDW** ima vrednost 0 ili 1 prelazi na korak $step_{3B}$ ili korak $step_{3C}$, respektivno. Kroz korak $step_{3B}$ se prolazi samo kada je dužina operanda jedan bajt. Tada se vrednostima 1 signala **mxBB₀** i **ldBB** bloka *exec* sadržaj registra $MDR_{7...0}$ bloka *bus* propušta kroz multiplekser MX2 i upisuje u registar $BB_{7...0}$. Iz koraka $step_{3B}$ se prelazi na korak $step_{50}$ i fazu izvršavanje operacije. Kroz korake $step_{3C}$ do $step_{40}$ se prolazi samo kada je dužina operanda dva bajta. Najpre se u koraku $step_{3C}$ vrednošću 1 signala **ldDWH** bloka *bus* sadržaj registra $MDR_{7...0}$ upisuje u stariji bajt registra $DW_{15...0}$ 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 $step_{3D}$ i $step_{3E}$ 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 $step_{3F}$ vrednošću 1 signala **ldDWL** bloka *bus* sadržaj registra $MDR_{7...0}$ upisuje u mlađi bajt registra $DW_{15...0}$. Time se u registru $DW_{15...0}$ nalazi 16-to bitni operand. Na kraju se u koraku $step_{40}$ vrednostima 1 signala **mxBW₀** i **ldBW** bloka *exec* sadržaj registra $DW_{15...0}$ propušta kroz multiplekser MX6 i upisuje u registar $BW_{15...0}$. Iz koraka $step_{40}$ se prelazi na korak $step_{50}$ i fazu izvršavanje operacije. !

$step_{38}$ *br (if hack then $step_{38}$);*

$step_{39}$ **eMAR, rdCPU,**

br (if fcCPU then $step_{39}$);

$step_{3A}$ *br (if LDW then $step_{3C}$);*

$step_{3B}$ **mxBB₀, ldBB,**

br $step_{50}$;

$step_{3C}$ **ldDWH, incMAR;**

$step_{3D}$ *br (if hack then $step_{3D}$);*

$step_{3E}$ **eMAR, rdCPU,**

br (if fcCPU then $step_{3E}$);

$step_{3F}$ **ldDWL;**

$step_{40}$ **mxBW₀, ldBW,**

br $step_{50}$;

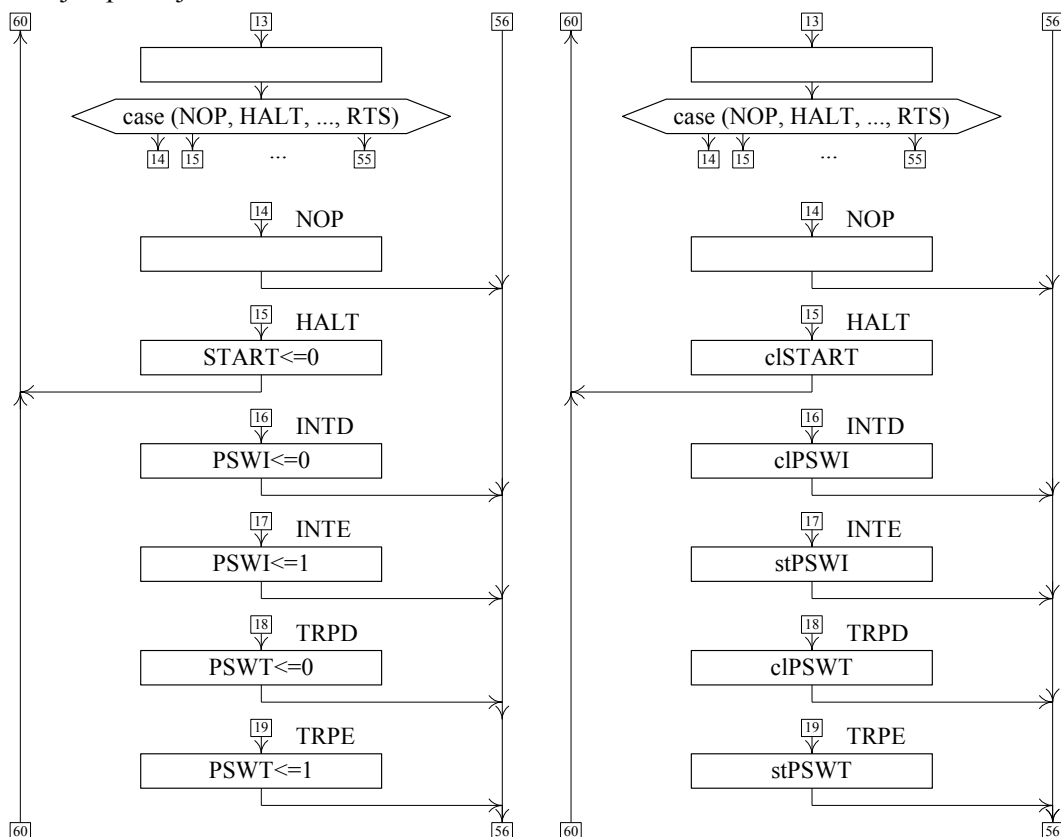
! Neposredno adresiranje !

! U korak $step_{41}$ se dolazi iz $step_{20}$ ukoliko signal za neposredno adresiranje **imm** ima vrednost 1. Iz ovog koraka se ukoliko signal **LDW** ima vrednost 1 prelazi na korak $step_{43}$ u kome se vrednostima 1 signala **mxBW₁** i **ldBW** bloka *exec* 16-to razredna neposredna veličina koja se nalazi u razredima $IR_{15...0}$ bloka *fetch* propušta kroz multiplekser MX6 i upisuje u registar $BW_{15...0}$. U suprotnom slučaju se prelazi na sledeći korak u kome se vrednostima 1 signala **mxBB₁** i **ldBB** bloka *exec* 8-mo razredna neposredna veličina koja se nalazi u razredima $IR_{15...8}$ propušta kroz multiplekser MX2 i upisuje u registar $BB_{7...0}$. U oba slučaja se prelazi na korak $step_{50}$ i fazu izvršavanja operacije. !

$step_{41}$ *br (if LDW then $step_{43}$);*

step₄₂ **mxBB₁, ldBB,**
br step₅₀;
step₄₃ **mxBW₁, ldBW,**
br step₅₀;

! Izvršavanje operacije !



Slika 37 Izvršavanje operacije (prvi deo)

! U korak step₅₀ se dolazi iz koraka step₀₇, step_{0E}, step₁₃, step₂₁, step₂₃, step₂₄, step₂₅, step₂₇, step₃₀, step₃₂, step₃₅, step₃₇, step_{3B}, step₄₀, step₄₂ i step₄₃ radi izvršavanja operacije. U koraku step₅₀ se realizuje višestruki uslovni skok na jedan od koraka step₅₁, step₅₂, ..., step_{B6} u zavisnosti od toga koji od signala operacija **NOP**, **HALT**, ..., **RTS** ima vrednost 1. !

step₅₀ **br (case (NOP, HALT, INTD, INTE, TRPD, TRPE,**
LDB, LDW, STB, STW, POPB, POPW, PUSHB, PUSHW,
LDIVTP, STIVTP, LDIMR, STIMR, LDSP, STSP,
ADD, SUB, INC, DEC, AND, OR, XOR, NOT,
ASR, LSR, ROR, RORC, ASL, LSL, ROL, ROLC,
BEQL, BNEQL, BNEG, BNNEG, BOVF, BNOVF, BCAR, BNCAR,
BGRT, BGRTE, BLSS, BLSS, BGRTU, BGRTEU, BLSSU, BLSSU,
JMP, INT, JSR, RTI, RTS)
then
(NOP, step₅₁), (HALT, step₅₂),
(INTD, step₅₃), (INTE, step₅₄), (TRPD, step₅₅), (TRPE, step₅₆),
(LDB, step₅₇), (LDW, step₅₉), (STB, step_{5A}), (STW, step₆₀),
(POPB, step₆₉), (POPW, step_{6F}), (PUSHB, step₇₉), (PUSHW, step_{7E}),
(LDIVTP, step₈₇), (STIVTP, step₈₈), (LDIMR, step₈₉), (STIMR, step_{8A}),
(LDSP, step_{8B}), (STSP, step_{8C}),
(ADD, step_{8D}), (SUB, step_{8F}), (INC, step₉₁), (DEC, step₉₃),
(AND, step₉₅), (OR, step₉₇), (XOR, step₉₉), (NOT, step_{9B}),
(ASR, step_{9D}), (LSR, step_{9D}), (ROR, step_{9D}), (RORC, step_{9D}),
(ASL, step_{9F}), (LSL, step_{9F}), (ROL, step_{9F}), (ROLC, step_{9F}),

(**BEQL**, step_{A1}), (**BNEQL**, step_{A1}), (**BNEG**, step_{A1}), (**BNNEG**, step_{A1}),
 (**BOVF**, step_{A1}), (**BNOVF**, step_{A1}), (**BCAR**, step_{A1}), (**BNCAR**, step_{A1}),
 (**BGRT**, step_{A1}), (**BGRE**, step_{A1}), (**BLSS**, step_{A1}), (**BLSSE**, step_{A1}),
 (**BGRTU**, step_{A1}), (**BGRTEU**, step_{A1}), (**BLSSU**, step_{A1}), (**BLSSEU**, step_{A1}),
 (**JMP**, step₃₂), (**INT**, step_{A4}), (**JSR**, step_{A5}), (**RTI**, step_{AE}), (**RTS**, step_{B6}));

! NOP !

! U korak step₅₁ se dolazi iz step₅₀ 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_{C0} i fazu opsluživanje prekida. !

step₅₁ br step_{C0};

! HALT !

! U korak step₅₁ se dolazi iz step₅₀ 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 step₀₀. !

step₅₂ clSTART,
 |
 br step₀₀;

! INTD !

! U korak step₅₃ se dolazi iz step₅₀ ukoliko signal operacije **INTD** ima vrednost 1. Vrednošću 1 signala **clPSWI** se razred **PSWI** bloka *exec* postavlja na vrednost 0. Iz koraka step₅₃ se bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step₅₃ clPSWI,
 br step_{C0};

! INTE !

! U korak step₅₄ se dolazi iz step₅₀ ukoliko signal operacije **INTE** ima vrednost 1. Vrednošću 1 signala **stPSWI** se razred **PSWI** bloka *exec* postavlja na vrednost 1. Iz koraka step₅₄ se bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida.!

step₅₄ stPSWI,
 br step_{C0};

! TRPD !

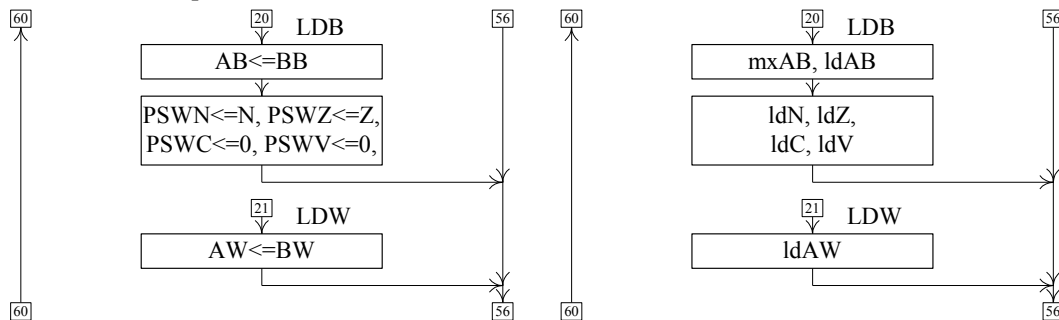
! U korak step₅₅ se dolazi iz step₅₀ ukoliko signal operacije **TRPD** ima vrednost 1. Vrednošću 1 signala **clPSWT** se razred **PSWT** bloka *exec* postavlja na vrednost 0. Iz koraka step₅₅ se bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step₅₅ clPSWT,
 br step_{C0};

! TRPE !

! U korak step₅₆ se dolazi iz step₅₀ ukoliko signal operacije **TRPE** ima vrednost 1. Vrednošću 1 signala **stPSWT** se razred **PSWT** bloka *exec* postavlja na vrednost 1. Iz koraka step₅₆ se bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida.!

step₅₆ stPSWT,
 br step_{C0};



Slika 38 Izvršavanje operacije (drugi deo)

! LDB !

! U korak step₅₇ se dolazi iz step₅₀ ukoliko signal operacije **LDB** ima vrednost 1. U fazi izvršavanja ove instrukcije se operand specificiran adresnim delom instrukcije prebacuje u registar AB_{7...0}. Stoga

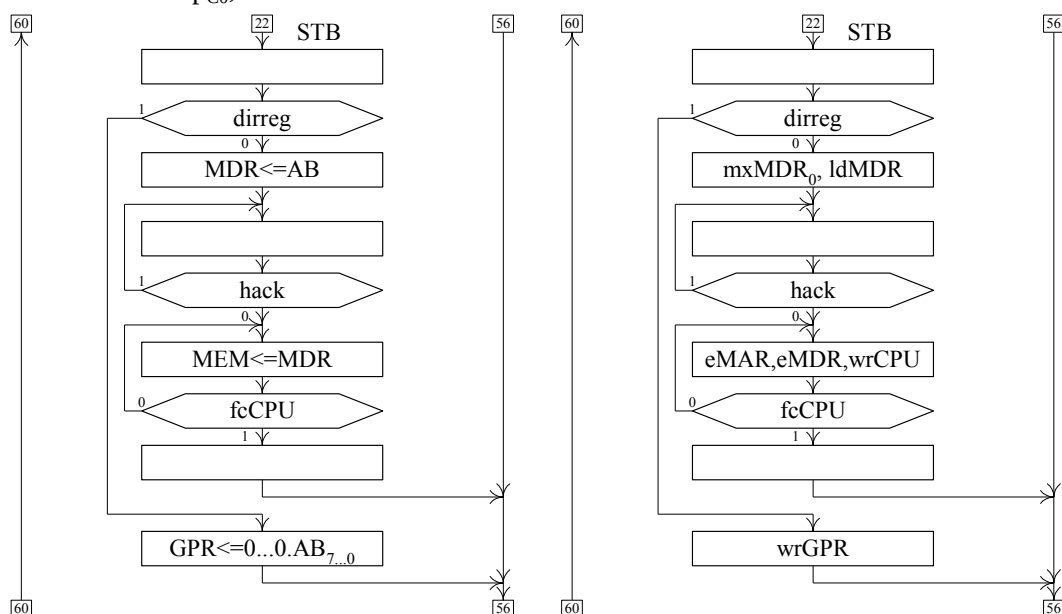
se vrednostima 1 signala **mxAB** i **ldAB** bloka *exec* sadržaj registra $BB_{7...0}$ propušta kroz multiplekser **MX** i upisuje u registar $AB_{7...0}$. Potom se u sledećem koraku vrednostima 1 signala **ldN**, **ldZ**, **ldC** i **ldV** bloka *exec* upisuju u razrede **PSWN** i **PSWZ** programske statusne reči **PSW** vrednosti signala **N** i **Z** formirane na osnovu sadržaja upisanog u registar $AB_{7...0}$ i u razrede **PSWC** i **PSWV** vrednosti 0 i prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

```
step57  mxAB, ldAB;
step58  ldN, ldZ, ldC, ldV,
         br stepC0;
```

! LDW !

! U korak $step_{59}$ se dolazi iz $step_{50}$ ukoliko signal operacije **LDW** ima vrednost 1. U ovom koraku se vrednošću 1 signala **ldAW** bloka *exec* se sadržaj registra $BW_{15...0}$ i upisuje u registar $AW_{15...0}$ i prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

```
step59  ldAW,
         br stepC0;
```



Slika 39 Izvršavanje operacije (treći deo)

! STB !

! U korak $step_{5A}$ se dolazi iz $step_{50}$ ukoliko signal operacije **STB** ima vrednost 1. Iz ovog koraka se u zavisnosti od toga da li signal **dirreg** bloka *fetch* ima vrednost 0 ili 1 prelazi na korak $step_{5B}$ ili $step_{5F}$, respektivno. !

```
step5A  br (if dirreg then step5F);
```

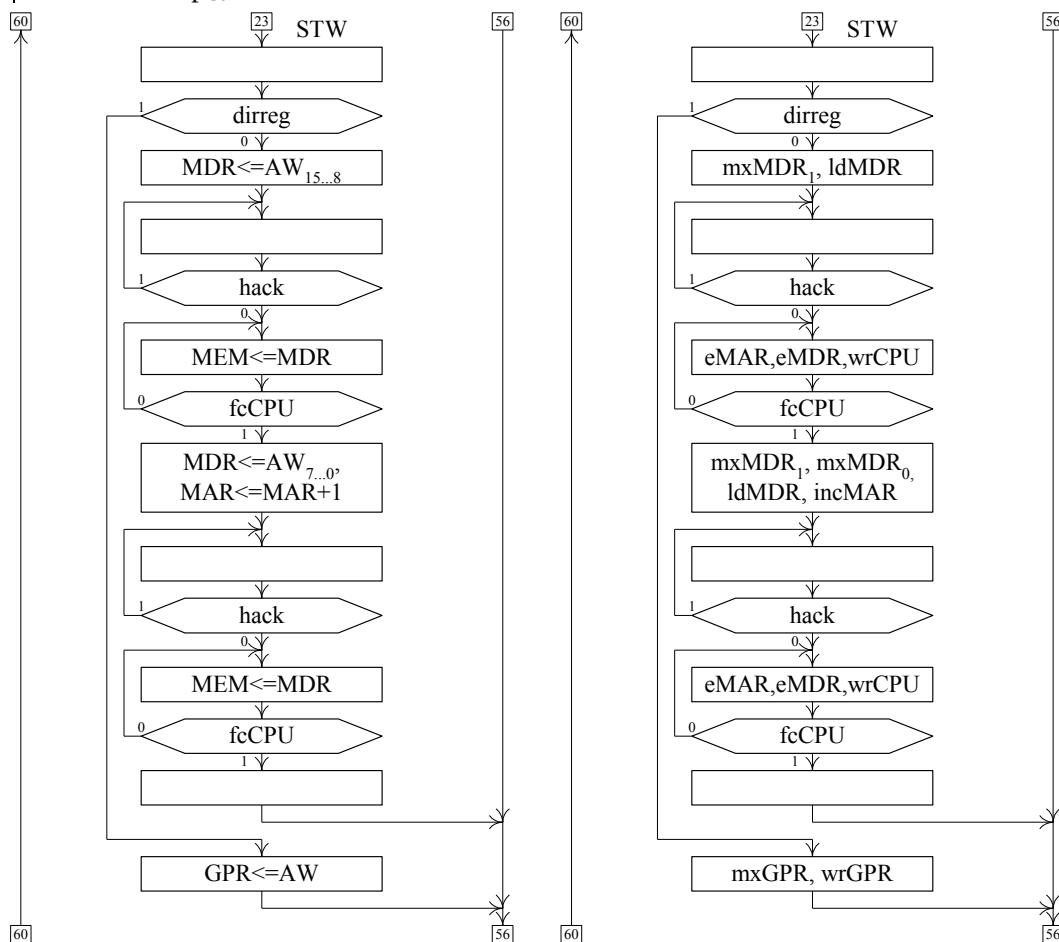
! U koracima $step_{5B}$, $step_{5C}$, $step_{5D}$ i $step_{5E}$ se sadržaj registra $AB_{7...0}$ bloka *exec* upisuje u memorijsku lokaciju na adresi koja je formirana u fazi formiranje adrese i koja se nalazi i registr $MAR_{15...0}$ bloka *bus*. Stoga se, najpre, u koraku $step_{5B}$ vrednostima 1 signala **mxMDR₀** i **ldMDR** bloka *bus* sadržaj registra $AB_{7...0}$ propušta kroz multiplekser **MX** i upisuje u registar $MDR_{7...0}$. Pre upisa se u koraku $step_{5C}$ proverava vrednost signala **hack** bloka *bus* koji vrednostima 1 i 0 ukazuje da li je processor prepustio magistralu kontroleru sa direktnim pristupom memoriji ili je magistrala u posedu procesora, respektivno. Ukoliko je procesor prepustio magistralu ostaje se u koraku $step_{5C}$, dok se u suprotnom slučaju prelazi na korak $step_{5D}$. U koraku $step_{5D}$ se realizuje upis. Vrednostima 1 signala **eMAR** i **eMDR** bloka *bus* sadržaji registara $MAR_{15...0}$ i $MDR_{7...0}$ se puštaju na adresne linije $ABUS_{15...0}$ i linije $DBUS_{7...0}$ podataka magistrale **BUS** i vrednošću 1 signala **wrCPU** formira vrednost 0 signala na upravljačkoj liniji **WRBUS** magistrale **BUS**, čime se u memoriji **MEM** startuje operacija upisa. U koraku $step_{5D}$ se ostaje onoliko perioda signala takta koliko je neophodno da se upis realizuje. Sve vreme dok je processor u koraku $step_{5D}$ adresa i podatak se prisutni na linijama $ABUS_{15...0}$ i $DBUS_{7...0}$ i na liniji **WRBUS** je vrednost 0. Po završenom upisu memorija **MEM** generiše vrednost 0 signala

na upravljačkoj liniji $\overline{\text{FCBUS}}$ magistrale **BUS**. U koraku step_{5D} se proverava vrednost signala **fcCPU** bloka *bus* koji vrednostima 0 i 1 ukazuje da upis nije završen i da je upis završen, respektivno. Ukoliko upis nije završen ostaje se u koraku step_{5D} , dok se u suprotnom slučaju prelazi na korak step_{5E} . Iz koraka step_{5E} se bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step_{5B} **mxMDR₀, ldMDR**;
 step_{5C} *br (if hack then step_{5C})*;
 step_{5D} **eMAR, eMDR, wrCPU**,
br (if $\overline{\text{fcCPU}}$ then step_{5D});
 step_{5E} *br step_{C0}* ;

! U koraku step_{5F} se sadržaj registra $\text{AB}_{7...0}$ bloka *exec* proširen nulama do dužine 16 bita vrednošću 1 signala **wrGPR** bloka *addr* upisuje u adresirani registar opšte namene $\text{GPR}_{15...0}$ i bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step_{5F} **wrGPR**,
br step_{C0} ;



Slika 40 Izvršavanje operacije (četvrti deo)

! STW !

! U korak step_{60} se dolazi iz step_{50} ukoliko signal operacije **STW** ima vrednost 1. Iz ovog koraka se u zavisnosti od toga da li signal **dirreg** bloka *fetch* ima vrednost 0 ili 1 prelazi na korak step_{61} ili step_{68} , respektivno. !

step_{60} *br (if dirreg then step_{68})*;

! U koracima step_{60} do step_{67} se sadržaj najpre višeg bajta a zatim i nižeg bajta registra $\text{AW}_{15...0}$ 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 $\text{MAR}_{15...0}$ bloka *bus*. Stoga se, najpre, u koraku step_{61} vrednostima 1 signala **mxMDR₁** i **ldMDR** bloka *bus* sadržaj registra $\text{AW}_{15...8}$ propušta kroz multiplekser MX2 i upisuje u registar $\text{MDR}_{7...0}$. Upis se realizuje u koracima step_{62} i step_{63} na isti

načina kao što se to radi u koracima step_{5C} i step_{5D} instrukcije STB. Potom se u koraku step_{64} vrednostima 1 signala **mxMDR₁**, **mxMDR₀** i **ldMDR** bloka *bus* sadržaj registra $\text{AW}_{7...0}$ propušta kroz multiplekser MX2 i upisuje u registar $\text{MDR}_{7...0}$. Upis se realizuje u koracima step_{65} i step_{66} na isti načina kao što se to radi u koracima step_{62} i step_{63} za prethodni bajt. Po završetku upisa se prelazi na korak step_{67} , a iz ovog koraka se bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

```

step61  mxMDR1, ldMDR;
step62  br (if hack then step62);
step63  eMAR, eMDR, wrCPU,
        br (if fcCPU then step63);
step64  mxMDR1, mxMDR0, ldMDR, incMAR;
step65  br (if hack then step65);
step66  eMAR, eMDR, wrCPU,
        br (if fcCPU then step66);
step67  br stepC0;

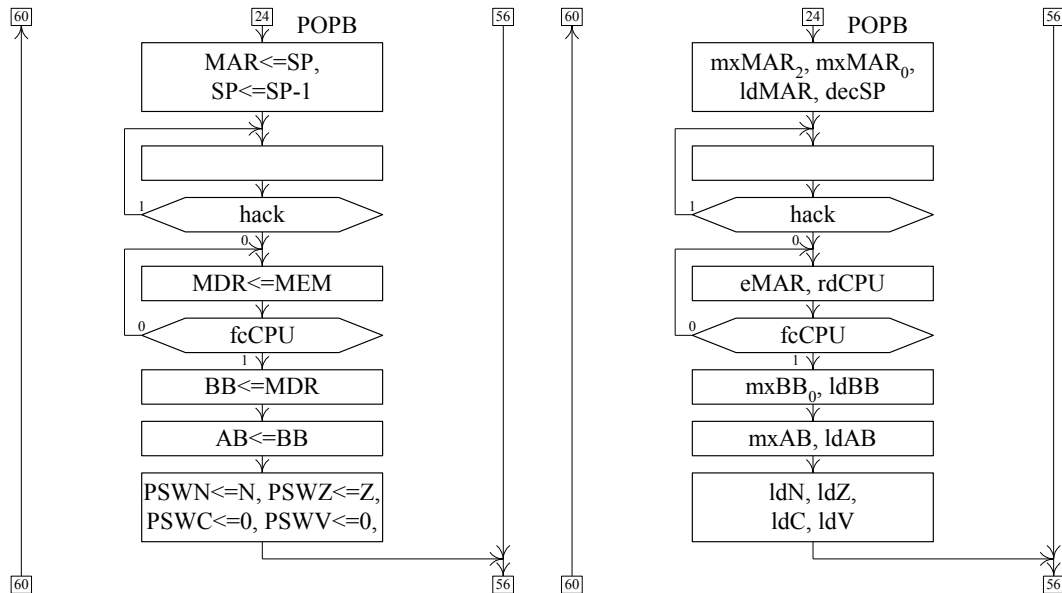
```

! U koraku step_{68} se vrednostima 1 signala **mxGPR** i **wrGPR** bloka *addr* sadržaj registra $\text{AW}_{15...0}$ bloka *exec* propušta kroz multiplekser MX1 i upisuje u adresirani registar opšte namene $\text{GPR}_{15...0}$ i bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

```

step68  mxGPR, wrGPR,
        br stepC0;

```



Slika 41 Izvršavanje operacije (peti deo)

! POPB !

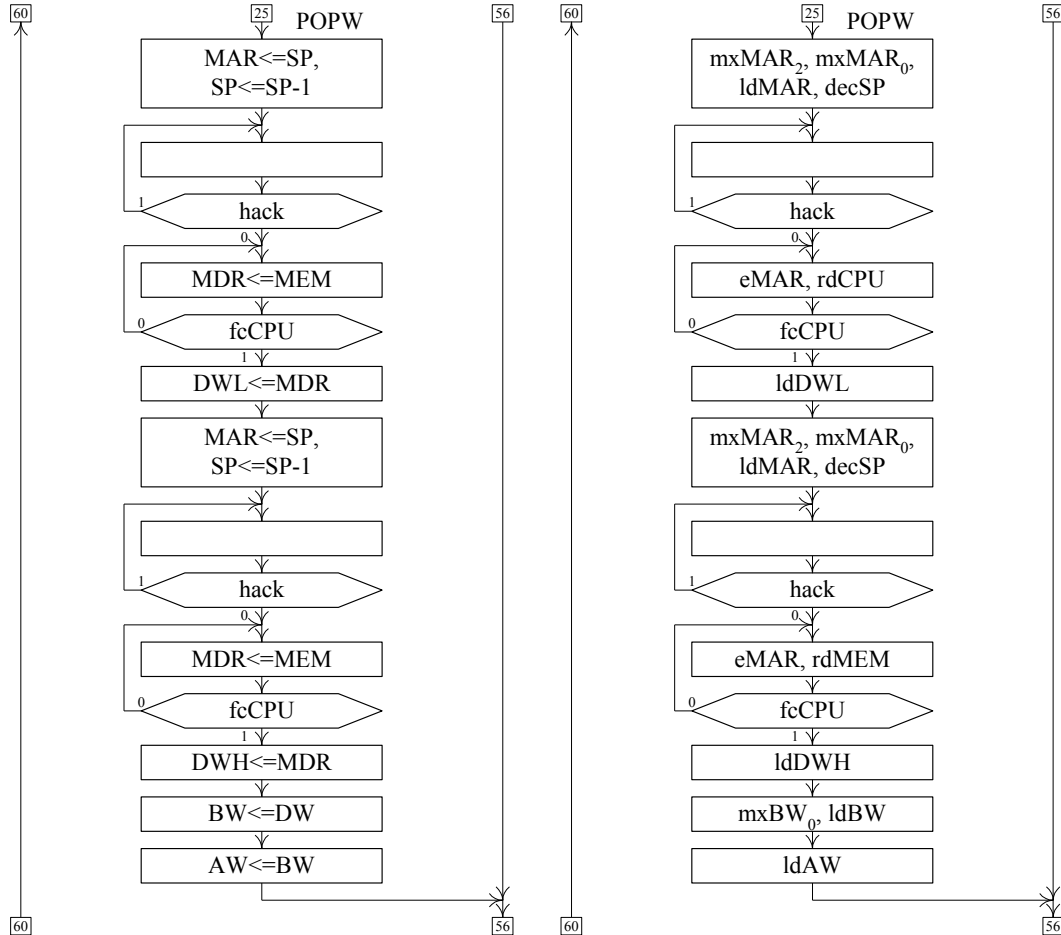
! U korak step_{69} se dolazi iz step_{50} ukoliko signal operacije **POPB** ima vrednost 1. U fazi izvršavanja ove instrukcije sa steka se skida bajt podatka i upisuje u registar $\text{AB}_{7...0}$ bloka *exec*. Stoga se, najpre, u koraku step_{69} vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR** bloka *bus* sadržaj registra $\text{SP}_{15...0}$ propušta kroz multiplekser MX1 i upisuje u registar $\text{MAR}_{15...0}$. Pored toga se i vrednošću 1 signala **decSP** dekrementira sadržaj registra $\text{SP}_{15...0}$ bloka *addr*. Čitanje se realizuje u koracima step_{6A} i step_{6B} na isti načina kao što se to radi u koracima step_{02} i step_{03} u kojima se čita prvi bajt instrukcije. Potom se vrednostima 1 signala **mxBB₀** i **ldBB** sadržaj registra $\text{MDR}_{7...0}$ bloka *bus* propušta kroz multiplekser MX2 i upisuje u registar $\text{BB}_{7...0}$ bloka *exec*. Zatim se vrednostima 1 signala **mxAB** i **ldAB** sadržaj registra $\text{BB}_{7...0}$ propušta kroz multiplekser MX1 i upisuje u registar $\text{AB}_{7...0}$ bloka *exec*. Na kraju se vrednostima 1 signala **ldN**, **ldZ**, **ldC** i **ldV** bloka *exec* upisuju u razrede PSWN i PSWZ programske statusne reči vrednosti signala N i Z formirane na osnovu sadržaja upisanog u registar AB i u razrede PSWC i PSWV vrednosti 0 i prelazi na korak step_{C0} i fazu opsluživanje prekida. !

```

step69  mxMAR2, mxMAR0, ldMAR, decSP;
step6A  br (if hack then step6A);

```

step_{6B} **eMAR, rdCPU**,
br (if $\overline{\text{fcCPU}}$ then step_{6B});
step_{6C} **mxBB₀, ldBB**;
step_{6D} **mxAB, ldAB**;
step_{6E} **ldN, ldZ, ldC, ldV**,
br step_{C0};

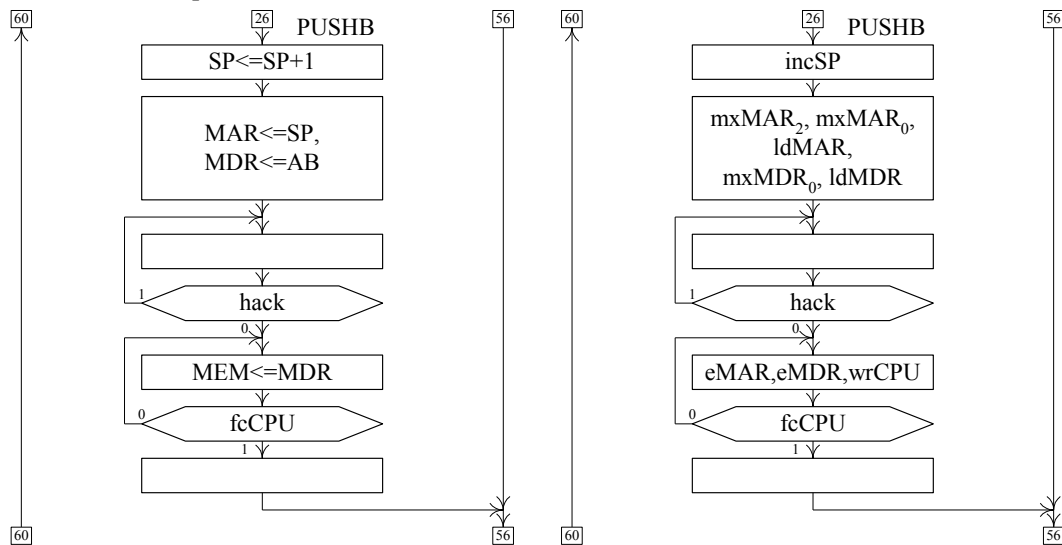


Slika 42 Izvršavanje operacije (šesti deo)

! POPW !

! U korak step_{6F} se dolazi iz step₅₀ ukoliko signal operacije **POPW** ima vrednost 1. U fazi izvršavanja ove instrukcije sa steka se skidaju dva bajta podatka i upisuju u registar **AW_{15...0}** bloka *exec*. Stoga se, najpre, u koraku step₆₉ vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR** bloka *bus* sadržaj registra **SP_{15...0}** bloka *addr* propušta kroz multiplekser **MX1** i upisuje u registar **MAR_{15...0}**. Pored toga se i vrednošću 1 signala **decSP** dekrementira sadržaj registra **SP_{15...0}**. Čitanje se realizuje u koracima step₇₀ i step₇₁ na isti načina kao što se to radi u koracima step₀₂ i step₀₃ u kojima se čita prvi bajt instrukcije. Potom se vrednošću 1 signala **ldDWL** sadržaj registra **MDR_{7...0}** bloka *bus* upisuje u niži bajt registra **DW_{7...0}** bloka *exec*. Zatim se u koraku step₇₃ vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR** sadržaj registra **SP_{15...0}** propušta kroz multiplekser **MX1** i upisuje u registar **MAR_{15...0}**. Pored toga se i vrednošću 1 signala **decSP** dekrementira sadržaj registra **SP_{15...0}**. Čitanje se realizuje u koracima step₇₄ i step₇₅ na isti načina kao što se to radi u koracima step₇₀ i step₇₁ u kojima se čita prethodni bajt. Potom se vrednošću 1 signala **ldDWH** sadržaj registra **MDR_{7...0}** upisuje u viši bajt registra **DW_{15...8}**. Time se u registru **DW_{15...0}** nalazi 16-to bitna vrednost skinuta sa steka. Na kraju se najpre vrednostima 1 signala **mxBW₀** i **ldBW** sadržaj registra **DW_{15...0}** propušta kroz multiplekser **MX6** i upisuje u registar **BW_{15...0}** bloka *exec* i zatim vrednostima 0 signala **mxAW₁** i **mxAW₀** i vrednošću 1 signala **ldAW** sadržaj registra **BW_{15...0}** propušta kroz multiplekser **MX5** i upisuje u registar **AW_{15...0}** i prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step_{6F} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
 step₇₀ *br (if **hack** then step₇₀);*
 step₇₁ **eMAR, rdCPU,**
 *br (if **fcCPU** then step₇₁);*
 step₇₂ **ldDWL;**
 step₇₃ **mxMAR₂, mxMAR₀, ldMAR, decSP;**
 step₇₄ *br (if **hack** then step₇₄);*
 step₇₅ **eMAR, rdCPU,**
 *br (if **fcCPU** then step₇₅);*
 step₇₆ **ldDWH;**
 step₇₇ **mxBW₀, ldBW;**
 step₇₈ **ldAW,**
 br step_{C0};



Slika 43 Izvršavanje operacije (sedmi deo)

! PUSHB !

! U korak step₇₉ se dolazi iz step₅₀ ukoliko signal operacije **PUSHB** ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra AB_{7...0} bloka *exec* stavlja na vrh steka. Stoga se najpre u koraku step₇₉ vrednošću 1 signala **incSP** vrši inkrementiranje registra SP_{15...0} bloka *addr*. Zatim se u koraku step_{7A} vrednostima 1 signala **mxMAR₂, mxMAR₀ i ldMAR**, sadržaj registra SP_{15...0} propušta kroz multiplexer MX1 bloka *bus* i upisuje u registar MAR_{15...0} i vrednostima 1 signala **mxMDR₀ i ldMDR** sadržaj registra AB_{7...0} propušta kroz multiplexer MX2 bloka *bus* i upisuje u registar MDR_{7...0}. Upis se realizuje u koracima step_{7B} i step_{7C} na isti način kao što se to radi u koracima step_{5C} i step_{5D} instrukcije STB. Na kraju se iz koraka step_{7D} bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

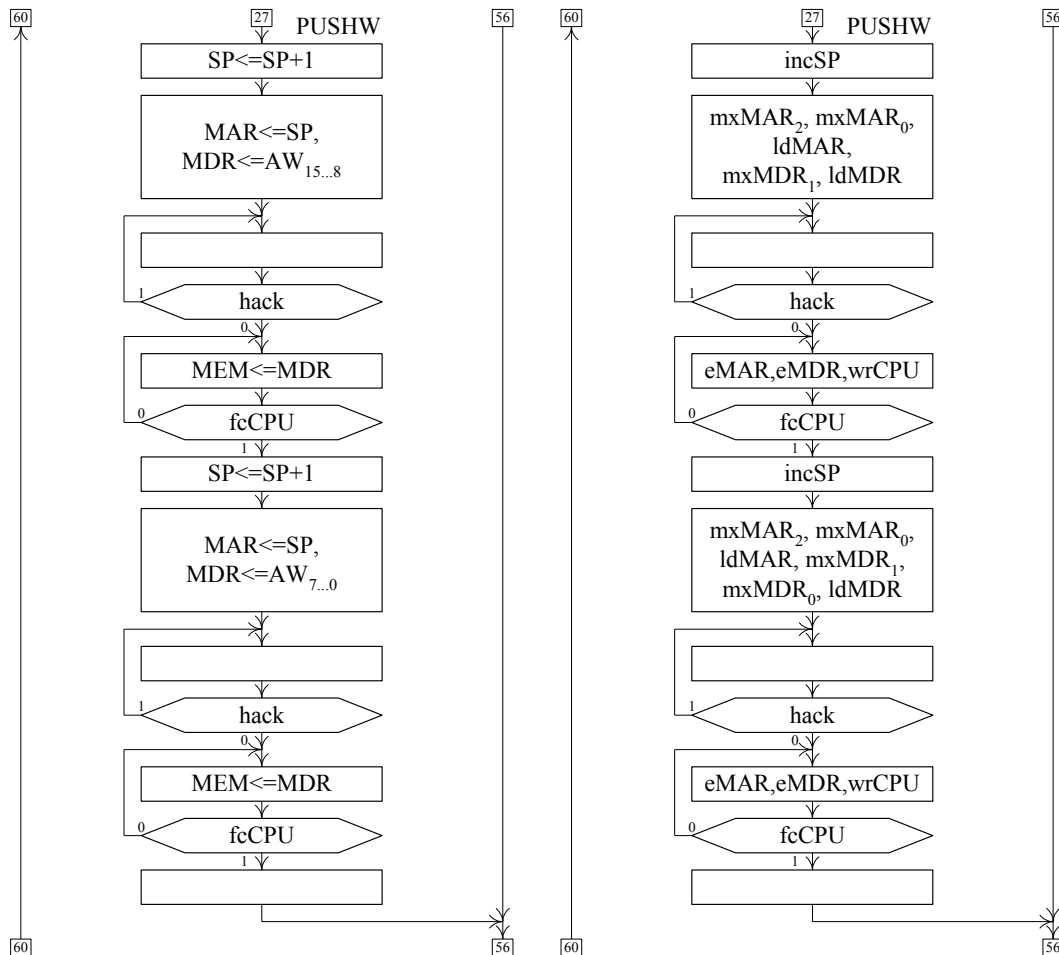
step₇₉ **incSP;**
 step_{7A} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₀, ldMDR;**
 step_{7B} *br (if **hack** then step_{7B});*
 step_{7C} **eMAR, eMDR, wrCPU,**
 *br (if **fcCPU** then step_{7C});*
 step_{7D} *br step_{C0};*

! PUSHW !

! U korak step_{7E} se dolazi iz step₅₀ ukoliko signal operacije **PUSHW** ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra AW_{15...0} bloka *exec* stavlja na stek i to prvo viši a zatim i niži bajt. Stoga se u koraku step_{7E} vrednošću 1 signala **incSP** vrši inkrementiranje registra SP_{15...0} bloka *addr*. Zatim se u koraku step_{7F} vrednostima 1 signala **mxMAR₂, mxMAR₀ i ldMAR**, sadržaj registra SP_{15...0} propušta kroz multiplexer MX1 bloka *bus* i upisuje u registar MAR_{15...0} i vrednostima

1 signala **mxMDR₁** i **ldMDR** sadržaj registra **AW_{15...8}** propušta kroz multiplexer MX2 i upisuje u registar **MDR_{7...0}**. Upis se realizuje u koracima **step₈₀** i **step₈₁** na isti načina kao što se to radi u koracima **step_{5C}** i **step_{5D}** instrukcije **STB**. Ponovo se sada u koraku **step₈₂** vrednošću 1 signala **incSP** vrši inkrementiranje registra **SP_{15...0}**. Zatim se u koraku **step₈₃** vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR** sadržaj registra **SP_{15...0}** propušta kroz multiplexer MX1 bloka *bus* i upisuje u registar **MAR_{15...0}** i vrednostima 1 signala **mxMDR₁**, **mxMDR₀** i **ldMDR** sadržaj registra **AW_{7...0}** propušta kroz multiplexer MX2 i upisuje u registar **MDR_{7...0}**. Upis se realizuje u koracima **step₈₄** i **step₈₅** na isti načina kao što se to radi u koracima **step₈₀** i **step₈₁**. Na kraju se iz koraka **step₈₆** bezuslovno prelazi na korak **step_{C0}** i fazu opsluživanje prekida. !

step_{7E} **incSP**;
step_{7F} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₁, ldMDR**;
step₈₀ *br (if hack then step₈₀);*
step₈₁ **eMAR, eMDR, wrCPU**,
br (if fcCPU then step₈₁);
step₈₂ **incSP**;
step₈₃ **mxMAR₂, mxMAR₀, ldMAR, mxMDR₁, mxMDR₀, ldMDR**;
step₈₄ *br (if hack then step₈₄);*
step₈₅ **eMAR, eMDR, wrCPU**,
br (if fcCPU then step₈₅);
step₈₆ *br step_{C0}*;



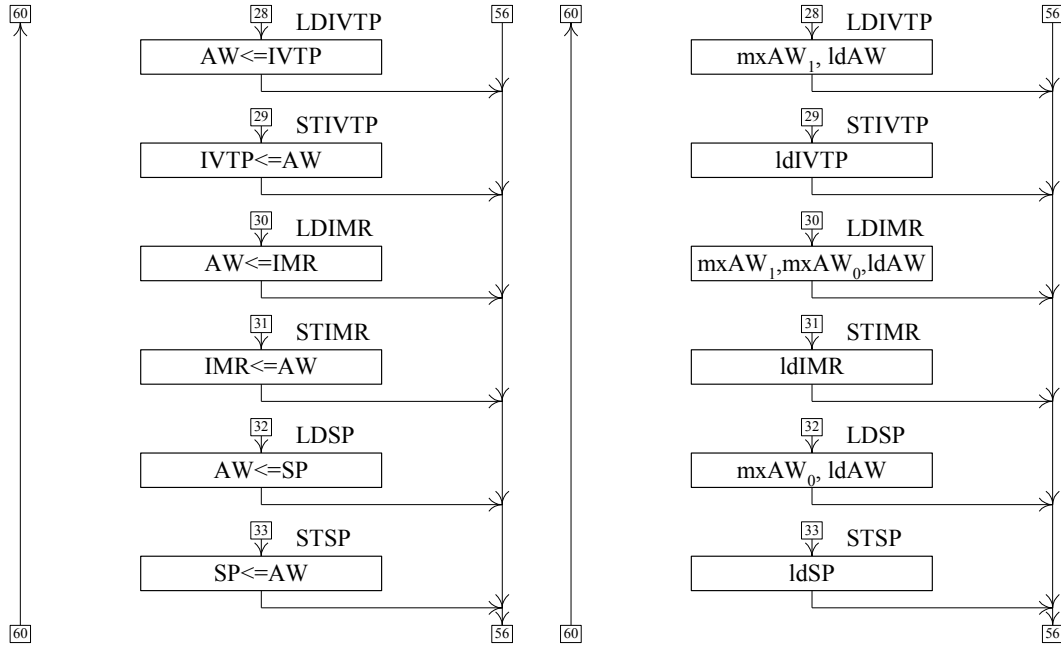
Slika 44 Izvršavanje operacije (osmi deo)

! LDIVTP !

! U korak **step₈₇** se dolazi iz **step₅₀** ukoliko signal operacije **LDIVTP** ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra **IVTP_{15...0}** bloka *intr* upisuje u registar **AW_{15...0}** bloka *exec*. Stoga se vrednostima 1 signala **mxAW₁** i **ldAW** sadržaj registra **IVTP_{15...0}** propušta kroz

multiplekser MX5 bloka *exec* i upisuje u registar $AW_{15...0}$. Na kraju se iz koraka $step_{87}$ bezuslovno prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

$step_{87}$ **$mxAW_1, ldAW,$**
 $br\ step_{C0};$



Slika 45 Izvršavanje operacije (deveti deo)

! STIVTP !

! U korak $step_{88}$ se dolazi iz $step_{50}$ ukoliko signal operacije STIVTP ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra $AW_{15...0}$ bloka *exec* upisuje u registar $IVTP_{15...0}$ bloka *intr*. Stoga se vrednošću 1 signala **$ldIVTP$** sadržaj registra $AW_{15...0}$ upisuje u registar $IVTP_{15...0}$. Na kraju se iz koraka $step_{88}$ bezuslovno prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

$step_{88}$ **$ldIVTP,$**
 $br\ step_{C0};$

! LDIMR !

! U korak $step_{89}$ se dolazi iz $step_{50}$ ukoliko signal operacije LDIMR ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra $IMR_{15...0}$ bloka *intr* upisuje u registar $AW_{15...0}$ bloka *exec*. Stoga se vrednostima 1 signala **$mxAW_1, mxAW_0$** i **$ldAW$** sadržaj registra $IMR_{15...0}$ propušta kroz multiplexer MX5 bloka *exec* i upisuje u registar $AW_{15...0}$. Na kraju se iz koraka $step_{89}$ bezuslovno prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

$step_{89}$ **$mxAW_1, mxAW_0, ldAW,$**
 $br\ step_{C0};$

! STIMR !

! U korak $step_{8A}$ se dolazi iz $step_{50}$ ukoliko signal operacije STIMR ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra $AW_{15...0}$ bloka *exec* upisuje u registar $IMR_{15...0}$ bloka *intr*. Stoga se vrednošću 1 signala **$ldIMR$** sadržaj registra $AW_{15...0}$ upisuje u registar $IMR_{15...0}$. Na kraju se iz koraka $step_{8A}$ bezuslovno prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

$step_{8A}$ **$ldIMR,$**
 $br\ step_{C0};$

! LDSP !

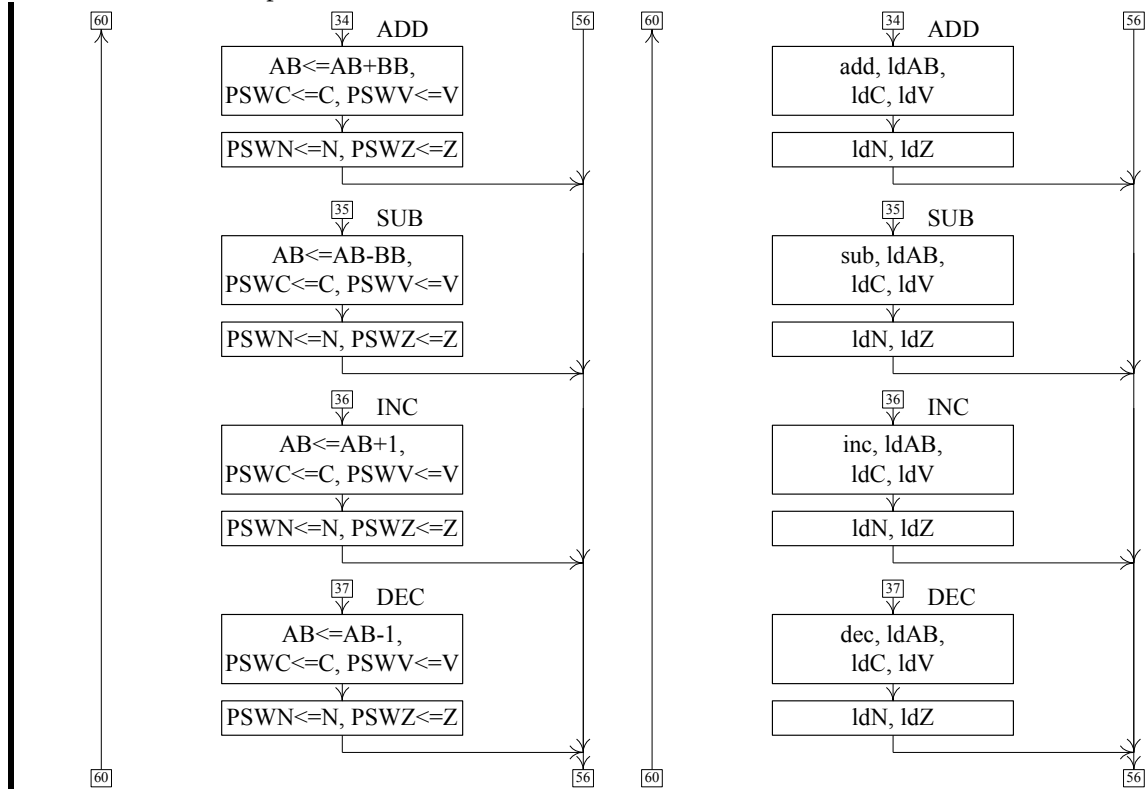
! U korak $step_{8B}$ se dolazi iz $step_{50}$ ukoliko signal operacije LDSP ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra $SP_{15...0}$ bloka *addr* upisuje u registar $AW_{15...0}$ bloka *exec*. Stoga se vrednostima 1 signala **$mxAW_0$** i **$ldAW$** sadržaj registra $SP_{15...0}$ propušta kroz multiplexer MX5 bloka *exec* i upisuje u registar $AW_{15...0}$. Na kraju se iz koraka $step_{8B}$ bezuslovno prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

step_{8B} **mxAW₀, ldAW,**
br step_{C0};

! STSP !

! U korak step_{8C} se dolazi iz step₅₀ ukoliko signal operacije STSP ima vrednost 1. U fazi izvršavanja ove instrukcije se sadržaj registra AW_{15...0} bloka *exec* upisuje u registar SP_{15...0} bloka *addr*. Stoga se vrednošću 1 signala **ldSP** sadržaj registra AW_{15...0} upisuje u registar SP_{15...0}. Na kraju se iz koraka step_{8C} bezuslovno prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step_{8C} **ldSP,**
br step_{C0};



Slika 46 Izvršavanje operacije (deseti deo)

! ADD !

! U korak step_{8D} se dolazi iz step₅₀ ukoliko signal operacije **ADD** ima vrednost 1. U fazi izvršavanja ove instrukcije se sabiraju sadržaji registra AB_{7...0} bloka *exec* koji se koristi kao akumulator i registra BB_{7...0} bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije i rezultat upisuje u registar AB_{7...0}. Stoga se vrednošću 1 signala **add** bloka *exec* na izlazima ALU_{7...0} aritmetičko logičke jedinice ALU formira suma sadržaja registara AB_{7...0} i BB_{7...0} koja se dalje vrednošću 0 signala **mxAB** propušta kroz multiplexer MX1 i vrednošću 1 signala **ldAB** upisuje u registar AB_{7...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_{7...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 AB_{7...0} i prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step_{8D} **add, ldAB, ldC, ldV;**
step_{8E} **ldN, ldZ,**
br step_{C0};

! SUB !

! U korak step_{8D} se dolazi iz step₅₀ ukoliko signal operacije **SUB** ima vrednost 1. U fazi izvršavanja ove instrukcije se od sadržaja registra AB_{7...0} bloka *exec* koji se koristi kao akumulator oduzima sadržaj registra BB_{7...0} bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije i

rezultat upisuje u registar $AB_{7...0}$. Stoga se vrednošću 1 signala **sub** bloka *exec* na izlazima $ALU_{7...0}$ formira razlika sadržaja registara $AB_{7...0}$ i $BB_{7...0}$ koja se dalje vrednošću 0 signala **mxAB** propušta kroz multiplexer MX1 i vrednošću 1 signala **ldAB** upisuje u registar $AB_{7...0}$. 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 $ALU_{7...0}$. 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 $AB_{7...0}$ i prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

step_{8F} **sub, ldAB, ldC, ldV;**

step₉₀ **ldN, ldZ,**

br step_{C0};

! INC !

! U korak $step_{91}$ se dolazi iz $step_{50}$ ukoliko signal operacije **INC** ima vrednost 1. U fazi izvršavanja ove instrukcije se sabira sadržaj registra $AB_{7...0}$ bloka *exec* koji se koristi kao akumulator i vrednost 1 i rezultat upisuje u registar $AB_{7...0}$. Stoga se vrednošću 1 signala **inc** bloka *exec* na izlazima $ALU_{7...0}$ formira suma sadržaja registra $AB_{7...0}$ i vrednosti 1 koja se dalje vrednošću 0 signala **mxAB** propušta kroz multiplexer MX1 i vrednošću 1 signala **ldAB** upisuje u registar $AB_{7...0}$. 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 $ALU_{7...0}$. 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 $AB_{7...0}$ i prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

step₉₁ **inc, ldAB, ldC, ldV;**

step₉₂ **ldN, ldZ,**

br step_{C0};

! DEC !

! U korak $step_{93}$ se dolazi iz $step_{50}$ ukoliko signal operacije **DEC** ima vrednost 1. U fazi izvršavanja ove instrukcije se od sadržaj registra $AB_{7...0}$ bloka *exec* koji se koristi kao akumulator i oduzima vrednost 1 i rezultat upisuje u registar $AB_{7...0}$. Stoga se vrednošću 1 signala **dec** bloka *exec* na izlazima $ALU_{7...0}$ formira razlika sadržaja registra $AB_{7...0}$ i vrednosti 1 koja se dalje vrednošću 0 signala **mxAB** propušta kroz multiplexer MX1 i vrednošću 1 signala **ldAB** upisuje u registar $AB_{7...0}$. 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 $ALU_{7...0}$. 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 $AB_{7...0}$ i prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

step₉₃ **dec, ldAB, ldC, ldV;**

step₉₄ **ldN, ldZ,**

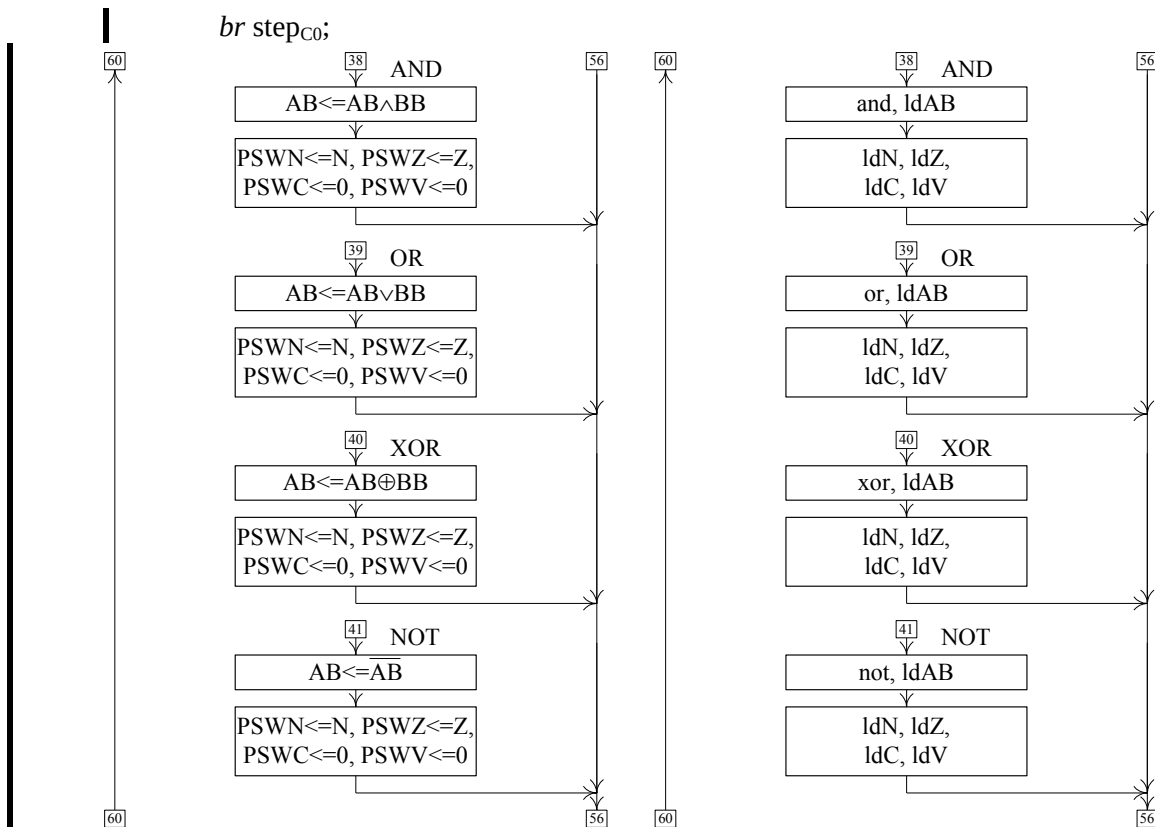
br step_{C0};

! AND !

! U korak $step_{95}$ se dolazi iz $step_{50}$ ukoliko signal operacije **AND** ima vrednost 1. U fazi izvršavanja ove instrukcije se nad sadržajima registra $AB_{7...0}$ bloka *exec* koji se koristi kao akumulator i registra $BB_{7...0}$ bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije realizuje logička I operacija i rezultat upisuje u registar $AB_{7...0}$. Stoga se vrednošću 1 signala **and** bloka *exec* na izlazima $ALU_{7...0}$ aritmetičko logičke jedinice ALU formira rezultat logičke I operacije sadržaja registara $AB_{7...0}$ i $BB_{7...0}$ koji se dalje vrednošću 0 signala **mxAB** propušta kroz multiplexer MX1 i vrednošću 1 signala **ldAB** upisuje u registar $AB_{7...0}$. 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 $AB_{7...0}$ i u razrede PSWC i PSWV neaktivne vrednosti i prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

step₉₅ **and, ldAB;**

step₉₆ **ldN, ldZ, ldC, ldV,**



Slika 47 Izvršavanje operacije (jedanaesti deo)

! OR !

! U korak step₉₇ se dolazi iz step₅₀ ukoliko signal operacije **OR** ima vrednost 1. U fazi izvršavanja ove instrukcije se nad sadržajima registra AB_{7...0} bloka *exec* koji se koristi kao akumulator i registra BB_{7...0} bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije realizuje logička ILI operacija i rezultat upisuje u registar AB_{7...0}. Stoga se vrednošću 1 signala **or** bloka *exec* na izlazima ALU_{7...0} formira rezultat logičke ILI operacije sadržaja registara AB_{7...0} i BB_{7...0} koji se dalje vrednošću 0 signala **mxAB** propušta kroz multiplexer MX1 i vrednošću 1 signala **ldAB** upisuje u registar AB_{7...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 AB_{7...0} i u razrede PSWC i PSWV neaktivne vrednosti i prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step₉₇ **or, ldAB;**

step₉₈ **ldN, ldZ, ldC, ldV,**

br stepC0;

! XOR !

! U korak step₉₉ se dolazi iz step₅₀ ukoliko signal operacije **XOR** ima vrednost 1. U fazi izvršavanja ove instrukcije se nad sadržajima registra AB_{7...0} bloka *exec* koji se koristi kao akumulator i registra AB_{7...0} bloka *exec* u kome se nalazi operand specificiran adresnim delom instrukcije realizuje logička ekskluzivno ILI operacija i rezultat upisuje u registar AB_{7...0}. Stoga se vrednošću 1 signala **xor** bloka *exec* na izlazima ALU_{7...0} formira rezultat logičke ekskluzivno ILI operacije sadržaja registara AB_{7...0} i BB_{7...0} koji se dalje vrednošću 0 signala **mxAB** propušta kroz multiplexer MX1 i vrednošću 1 signala **ldAB** upisuje u registar AB_{7...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 AB_{7...0} i u razrede PSWC i PSWV neaktivne vrednosti i prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step₉₉ **xor, ldAB;**

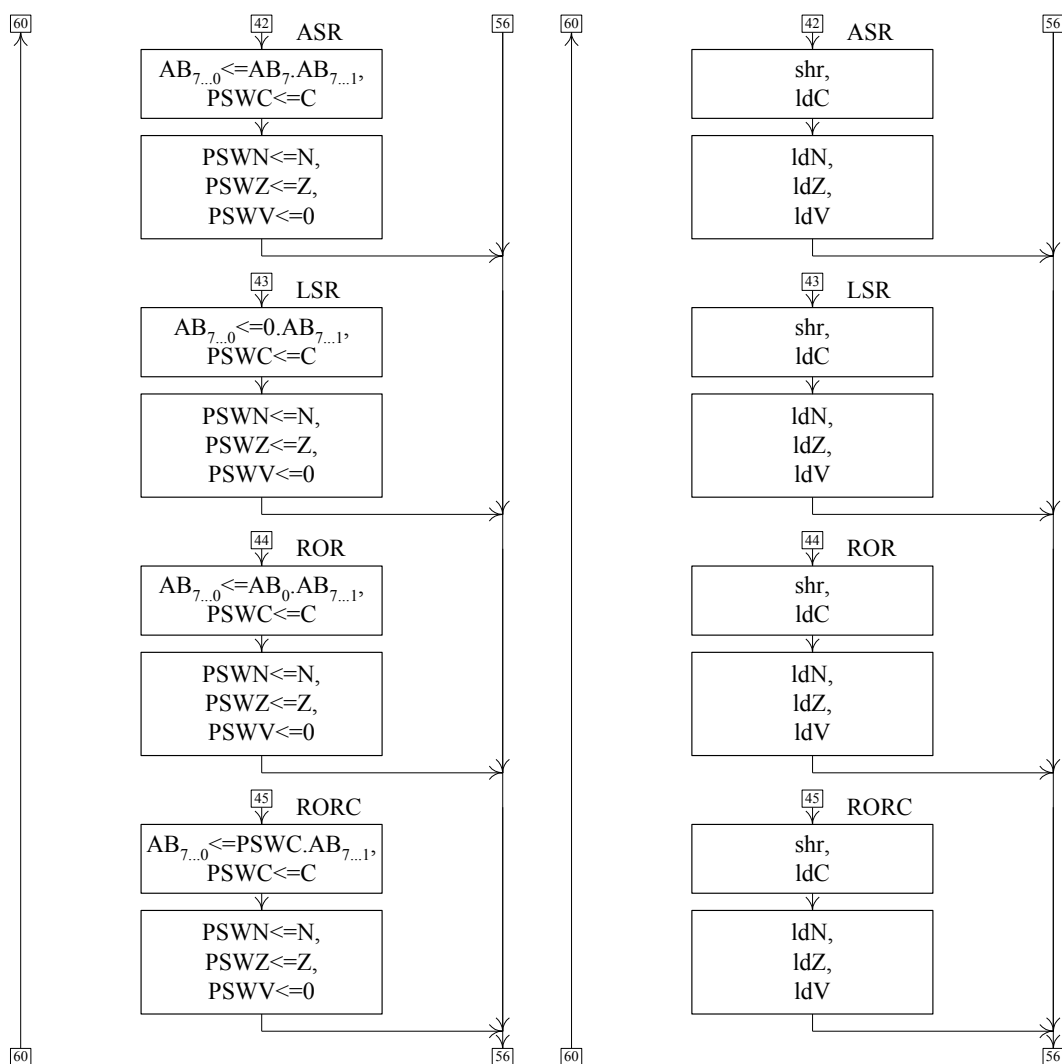
step_{9A} **ldN, ldZ, ldC, ldV,**

br stepC0;

! NOT !

! U korak step_{9B} se dolazi iz step₅₀ ukoliko signal operacije **NOT** ima vrednost 1. U fazi izvršavanja ove instrukcije se invertuju bitovi registra AB_{7...0} bloka *exec* koji se koristi kao akumulator i rezultat upisuje u registar AB_{7...0}. Stoga se vrednošću 1 signala **not** bloka *exec* na izlazima ALU_{7...0} formiraju invertovane vrednosti bitova registra AB_{7...0} koje se dalje vrednošću 0 signala **mxAB** propušta kroz multiplekser MX1 i vrednošću 1 signala **ldAB** upisuju u registar AB_{7...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 AB_{7...0} i u razrede PSWC i PSWV vrednosti 0 i prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step_{9B} **not, ldAB;**
 step_{9C} **ldN, ldZ, ldC, ldV,**
br step_{C0};



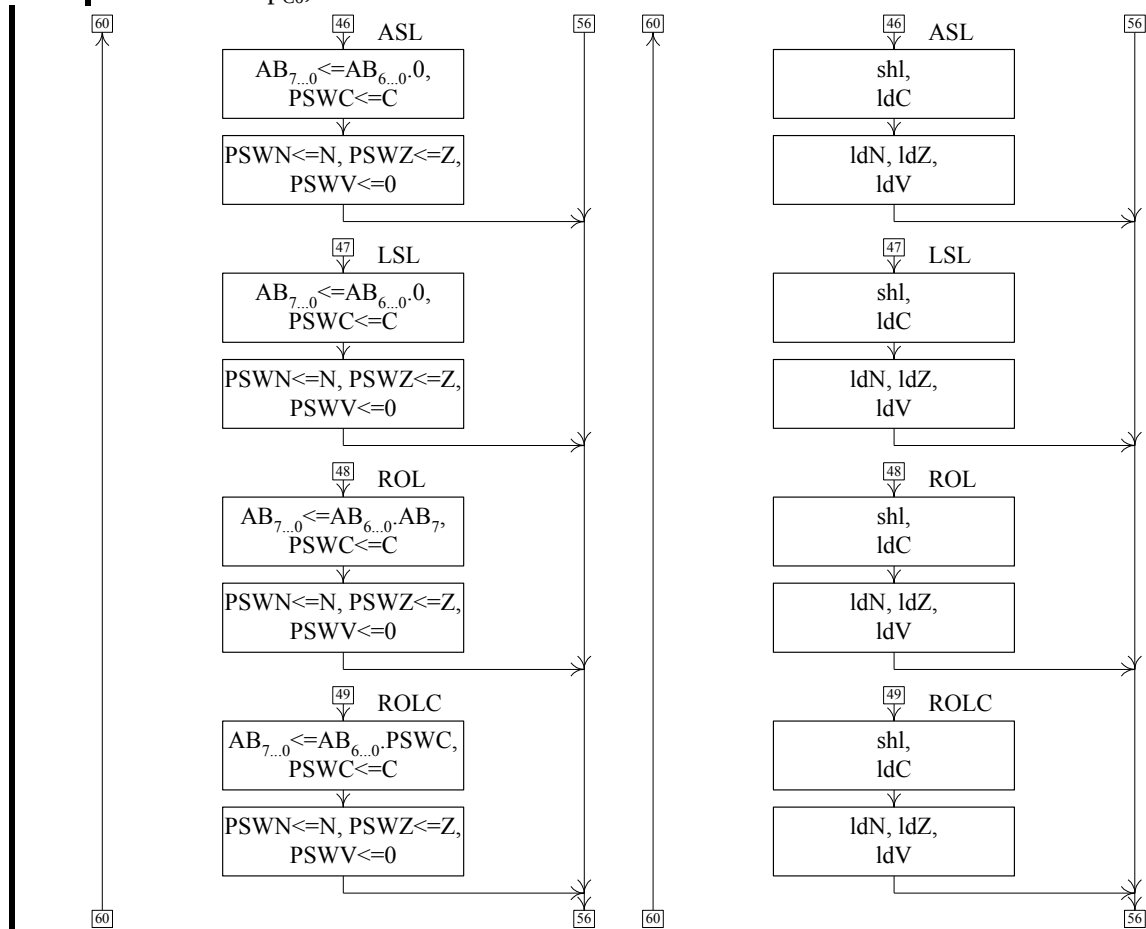
Slika 48 Izvršavanje operacije (dvanaesti deo)

! ASR, LSR, ROR i ROLC !

! U korak step_{9D} se dolazi iz step₅₀ ukoliko neki od signala operacije ASR, LSR, ROR i ROLC ima vrednost 1. U fazi izvršavanja ove instrukcije se bitovi registra AB_{7...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 AB_{7...0} pomeraju udesno za jedno mesto, pri čemu se u razred AB₇ upisuje signal **IR** sa izlaza multipleksera MX3 bloka *exec*. U slučaju aritmetičkog pomeranja za jedno

mesto udesno to je signal **AB₇**. Ovaj signal se selektuje na izlazu IR multipleksera MX3 binarnom vrednošću 0 signala selekcije multipleksera MX3 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 MX3 binarnom vrednošću 1 signala selekcije multipleksera MX3 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 **AB₀**. Ovaj signal se selektuje na izlazu IR multipleksera MX3 binarnom vrednošću 2 signala selekcije multipleksera MX3 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 MX3 binarnom vrednošću 3 signala selekcije multipleksera MX3 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 **AB₀**. U koraku step_{9E} 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 AB_{7...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_{9E} se bezuslovono prelazi na korak step_{C0} i fazu opsluživanje prekida. !

step_{9D} **shr, ldC;**
step_{9E} **ldN, ldZ, ldV,**
br step_{C0};



Slika 49 Izvršavanje operacije (trinaesti deo)

! ASL, LSL, ROL i ROLC !

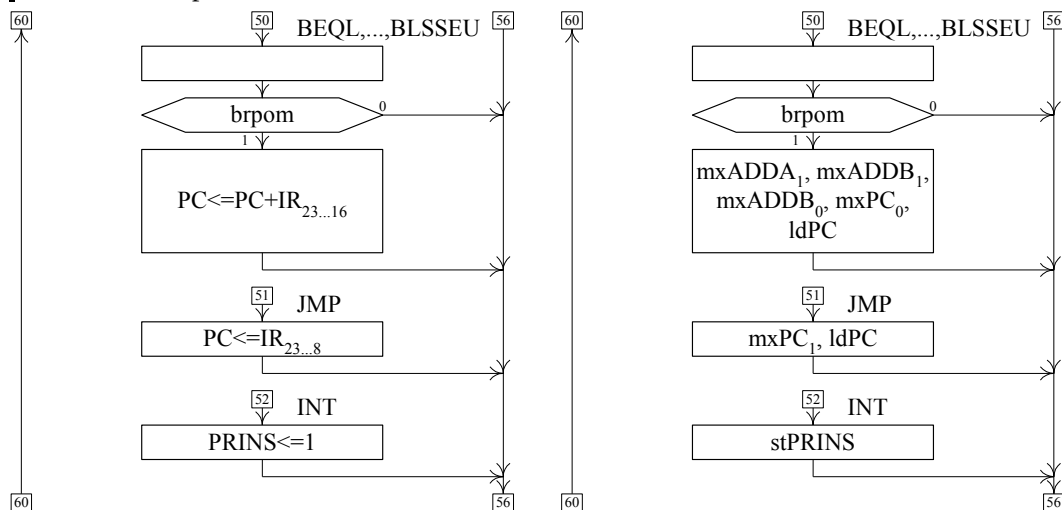
! U korak step_{9F} se dolazi iz step₅₀ ukoliko neki od signala operacije ASL, LSL, ROL i ROLC ima vrednost 1. U fazi izvršavanja ove instrukcije se bitovi registra AB_{7...0} bloka *exec* koji se koristi kao

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 $AB_{7...0}$ pomeraju ulevo za jedno mesto, pri čemu se u u razred AB_0 upisuje signal **IL** sa izlaza multipleksera MX4 bloka *exec*. U slučaju aritmetičkog pomeranja za jedno mesto ulevo to je signal **0**. Ovaj signal se selektuje na izlazu IL multipleksera MX4 binarnom vrednošću 0 signala selekcije multipleksera MX3 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 MX4 binarnom vrednošću 1 signala selekcije multipleksera MX4 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 **AB₇**. Ovaj signal se selektuje na izlazu IL multipleksera MX4 binarnom vrednošću 2 signala selekcije multipleksera MX4 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 MX4 binarnom vrednošću 3 signala selekcije multipleksera MX4 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 **AB₇**. U koraku $step_{A0}$ 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 $AB_{7...0}$ i vrednošću 1 signala **ldV** bloka *exec* u razred PSWV programske statusne reči $PSW_{15...0}$ upisuje vrednost 0 signala **V**. Iz koraka $step_{A0}$ se bezuslovno prelazi na korak $step_{C0}$ i fazu opsluživanje prekida. !

$step_{9F}$ **shl, ldC;**

$step_{A0}$ **ldN, ldZ, ldV,**

br $step_{C0}$;



Slika 50 Izvršavanje operacije (četnaesti deo)

! BEQL, ..., BLSSEU !

! U korak $step_{A1}$ se dolazi iz $step_{50}$ ukoliko neki od signala operacija uslovnog skoka **BEQL**, **BNEQL**, **BNEG**, **BNNEG**, **BOVF**, **BNOVF**, **BCAR**, **BNCAR**, **BGRTE**, **BGRTEU**, **BLSS**, **BLSSE**, **BGRTEU**, **BLSSU**, **BLSSEU** ima vrednost 1 i prelazi na $step_{C0}$ i fazu opsluživanje prekida ukoliko signal **brpom** bloka *exec* ima vrednost 0 i na sledeći korak $step_{A2}$ ukoliko ima vrednost 1. Signal **brpom** ima vrednost 1 ukoliko vrednost 1 ima signal rezultata operacije **eql**, **neql**, **neg**, **nneg**, **ovf**, **novf**, **car**, **ncar**, **grt**, **grte**, **lss**, **lsse**, **grtu**, **grteu**, **lssu**, **lsseu** određen vrednošću 1 signala operacije uslovnog skoka **BEQL**, **BNEQL**, **BNEG**, **BNNEG**, **BOVF**, **BNOVF**, **BCAR**, **BNCAR**, **BGRTE**, **BGRTEU**, **BLSS**, **BLSSE**, **BGRTEU**, **BLSSU**, **BLSSEU**. !

step_{A1} *br* (if **brpom** then step_{C0});

! U korak step_{A2} se dolazi iz step_{A1} ukoliko signal **brpom** ima vrednost 1, čime je uslov za skok ispunjen. U ovom koraku se vrednostima 1 signala **mxADDA₁**, **mxADDB₁** i **mxADDB₀** kroz multipleksere MX2 i MX3 bloka *addr* na ulaze sabirača ADD propuštaju sadržaji registra PC_{15...0} bloka *fetch* i registra IR_{23...16} bloka *fetch* proširen znakom na 16 bita. Signali ADD_{15...0} sa izlaza sabirača ADD koji predstavljaju adresu skoka se vrednostima 1 signala **mxPC₀** i **ldPC** propuštaju kroz multiplekser MX bloka *fetch* i upisuju u registar PC_{15...0}. Pored toga iz koraka step_{A2} se bezuslovno prelazi na step_{C0} i fazu opsluživanje prekida. !

step_{A2} **mxADDA₁**, **mxADDB₁**, **mxADDB₀**, **mxPC₀**, **ldPC**,
br step_{C0};

! JMP !

! U korak step_{A3} se dolazi iz step₅₀ 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 **mxPC₁** i **ldPC** propušta kroz multiplekser MX bloka *fetch* i upisuje u registar PC_{15...0}. Pored toga iz koraka step_{A3} se bezuslovno prelazi na step_{C0} i fazu opsluživanje prekida. !

step_{A3} **mxPC₁**, **ldPC**,
br step_{C0};

! INT !

! U korak step_{A4} se dolazi iz step₅₀ 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_{C0} i fazu opsluživanje prekida. !

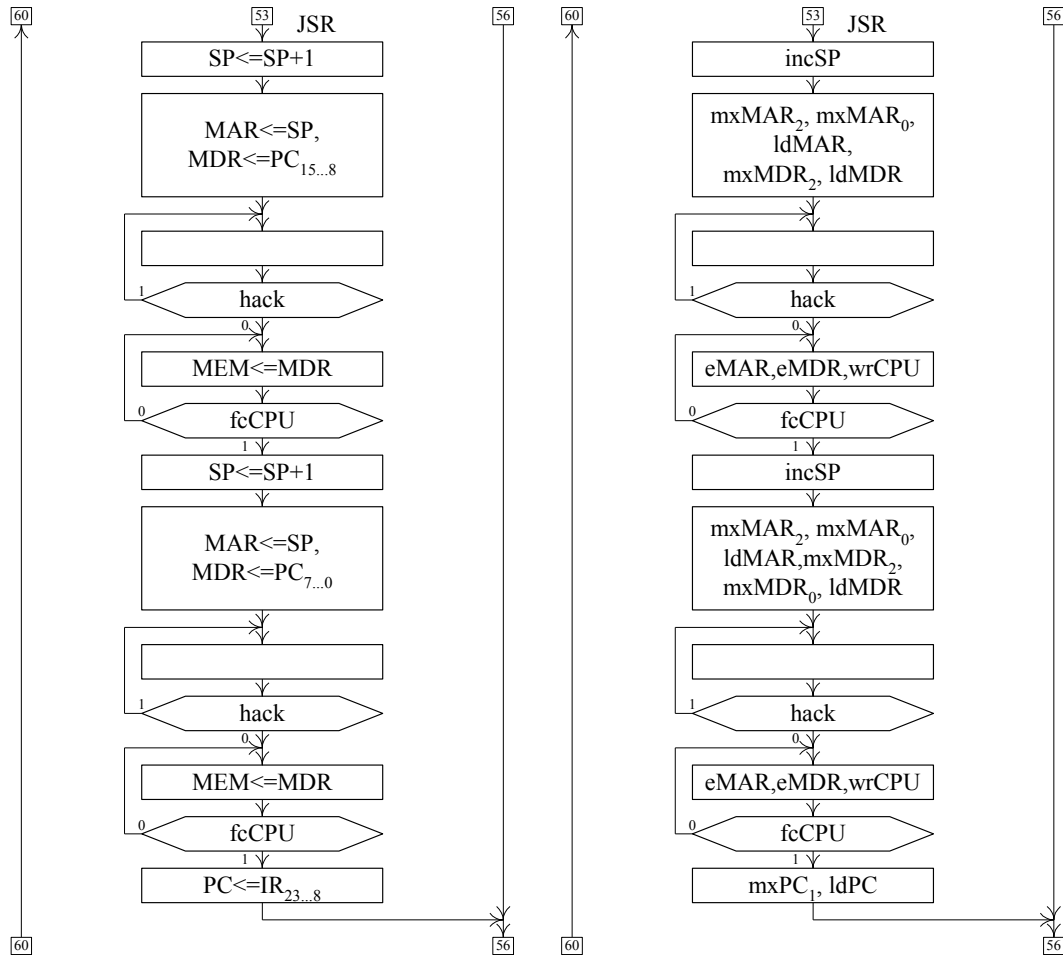
step_{A4} **stPRINS**,
br step_{C0};

! JSR !

! U korak step_{A5} se dolazi iz step₅₀ 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_{A5} vrednošću 1 signala **incSP** vrši inkrementiranje registra SP_{15...0} bloka *addr*. Zatim se u koraku step_{A6} vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR**, sadržaj registra SP_{15...0} propušta kroz multiplekser MX1 bloka *bus* i upisuje u registar MAR_{15...0} i vrednostima 1 signala **mxMDR₂** i **ldMDR** sadržaj višeg bajta registra PC_{15...8} propušta kroz multiplekser MX2 i upisuje u registar MDR_{7...0}. Upis se realizuje u koracima step_{A7} i step_{A8} na isti načina kao što se to radi u koracima step_{5C} i step_{5D} instrukcije STB. Potom se u koraku step_{A9} vrednošću 1 signala **incSP** vrši inkrementiranje registra SP_{15...0}. Zatim se u koraku step_{AA} vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR**, sadržaj registra SP_{15...0} propušta kroz multiplekser MX1 bloka *bus* i upisuje u registar MAR_{15...0} i vrednostima 1 signala **mxMDR₂**, **mxMDR₀**, **ldMDR** sadržaj nižeg bajta registra PC_{7...0} propušta kroz multiplekser MX2 i upisuje u registar MDR_{7...0}. Upis se realizuje u koracima step_{AB} i step_{AC} na isti načina kao što se to radi u koracima step_{A7} i step_{A8} prilikom upisa višeg bajta. Na kraju se u koraku step_{AD} sadržaj registra IR_{23...8} bloka *fetch* koji predstavlja adresu skoka vrednostima 1 signala **mxPC₁** i **ldPC** propušta kroz multiplekser MX bloka *fetch* i upisuje u registar PC_{15...0} i bezuslovno prelazi na step_{C0} i fazu opsluživanje prekida. !

step_{A5} **incSP**;
step_{A6} **mxMAR₂**, **mxMAR₀**, **ldMAR**, **mxMDR₂**, **ldMDR**;
step_{A7} *br* (if **hack** then step_{A7});
step_{A8} **eMAR**, **eMDR**, **wrCPU**,
br (if **fcCPU** then step_{A8});
step_{A9} **incSP**;
step_{AA} **mxMAR₂**, **mxMAR₀**, **ldMAR**, **mxMDR₂**, **mxMDR₀**, **ldMDR**;

step_{AB} *br (if **hack** then step_{AB});*
 step_{AC} **eMAR, eMDR, wrCPU,**
*br (if **fcCPU** then step_{AC});*
 step_{AD} **mxPC₁, ldPC,**
br step_{C0};



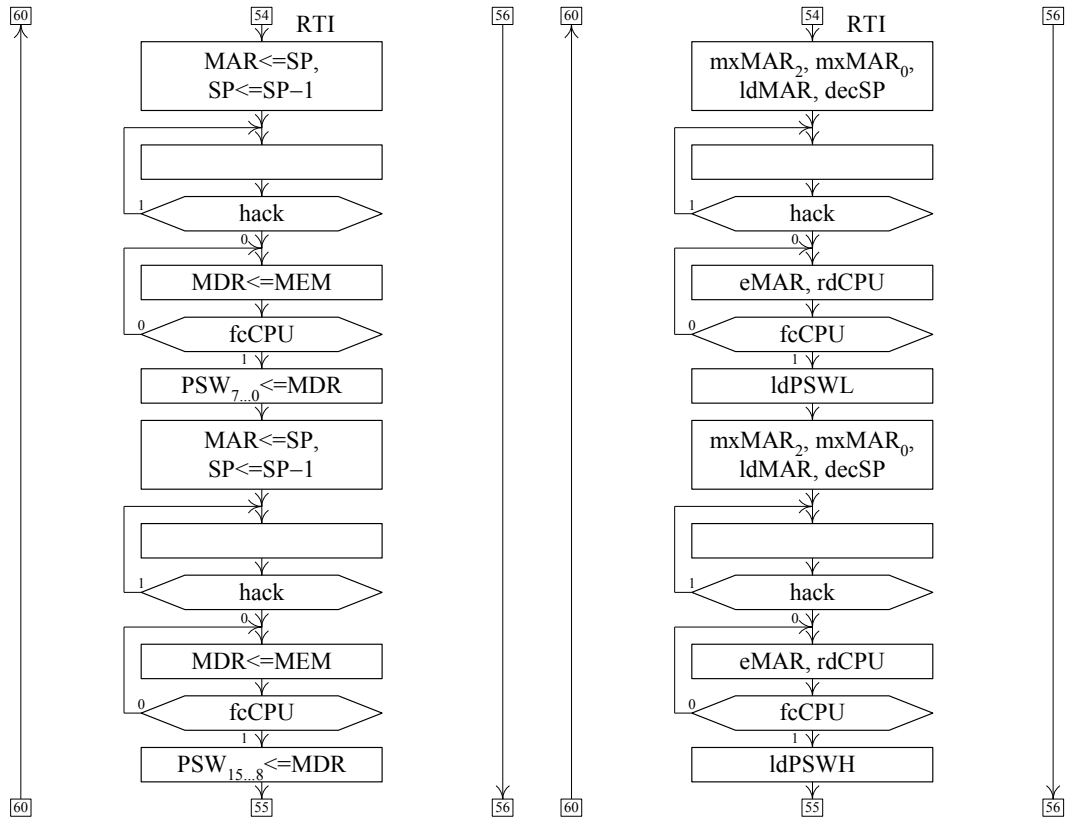
Slika 51 Izvršavanje operacije (petnaesti deo)

! RTI !

! U korak step_{AE} se dolazi iz step₅₀ 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_{AE} do step_{B5} se najpre vrednošću sa steka restaurira programska statusna reč PSW_{15...0} bloka *exec*. Sa steka se najpre skida niži a zatim i viši bajt registra PSW_{15...0}. Stoga se u koraku step_{AE} vrednostima 1 signala **mxMAR₂, mxMAR₀ i ldMAR** bloka *bus* sadržaj registra SP_{15...0} bloka *addr* propušta kroz multiplexer MX1 i upisuje u registar MAR_{15...0}. Pored toga se i vrednošću 1 signala **decSP** dekrementira sadržaj registra SP_{15...0}. Čitanje se realizuje u koracima step_{AF} i step_{B0} na isti načina kao što se to radi u koracima step₀₂ i step₀₃ u kojima se čita prvi bajt instrukcije. Na kraju se u koraku step_{B1} vrednošću 1 signala **ldPSWL** sadržaj registra MDR_{7...0} bloka *bus* upisuje u niži bajt registra PSW_{7...0} bloka *exec*. Potom se u koraku step_{B2} vrednostima 1 signala **mxMAR₂, mxMAR₀ i ldMAR** sadržaj registra SP_{15...0} propušta kroz multiplexer MX1 i upisuje u registar MAR_{15...0}. Pored toga se i vrednošću 1 signala **decSP** dekrementira sadržaj registra SP_{15...0}. Čitanje se realizuje u koracima step_{B3} i step_{B4} na isti načina kao što se to radi u koracima step_{AF} i step_{B0} u kojima se čita niži bajt. Na kraju se u koraku step_{B5} vrednošću 1 signala **ldPSWH** sadržaj registra MDR_{7...0} bloka *bus* upisuje u viši bajt registra PSW_{15...8}. 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_{AE} **mxMAR₂, mxMAR₀, ldMAR, decSP;**

step_{AF} *br (if **hack** then step_{AF});*
 step_{B0} **eMAR, rdCPU,**
 *br (if **fcCPU** then step_{B0});*
 step_{B1} **ldPSWL;**
 step_{B2} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
 step_{B3} *br (if **hack** then step_{B3});*
 step_{B4} **eMAR, rdCPU,**
 *br (if **fcCPU** then step_{B4});*
 step_{B5} **ldPSWH;**

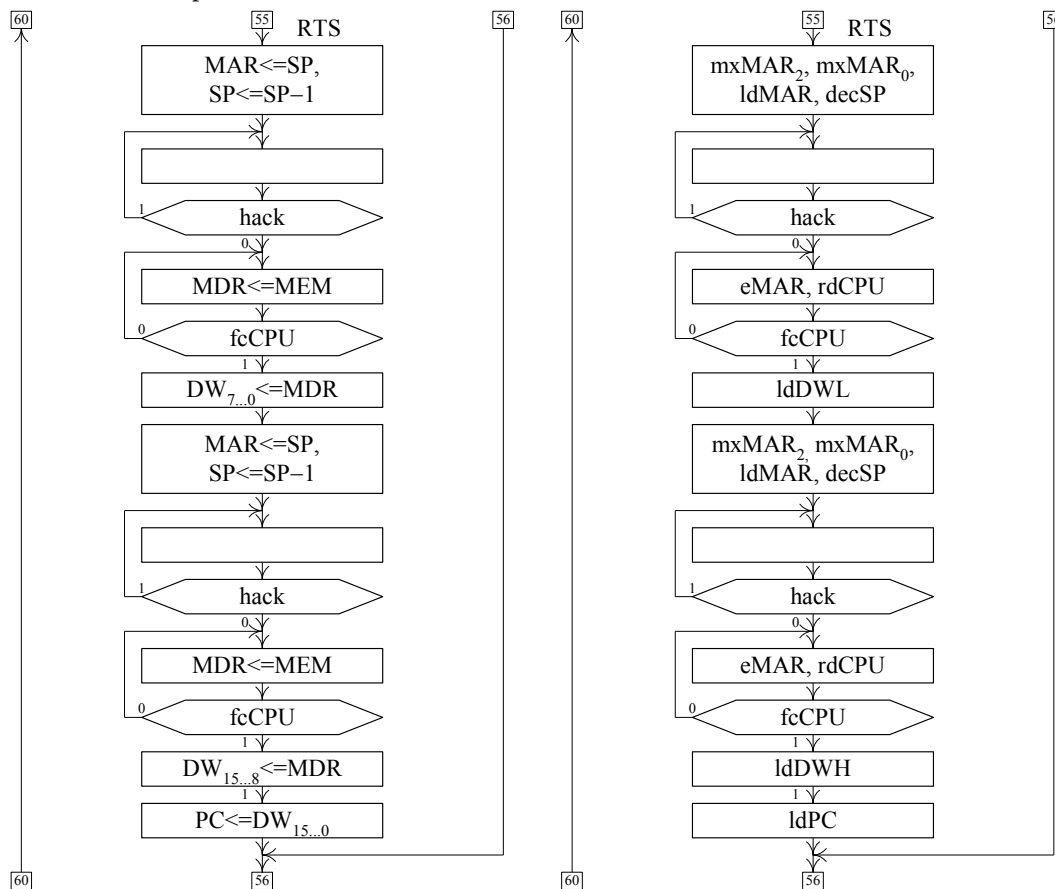


Slika 52 Izvršavanje operacije (šestnaesti deo)

! RTS !

! U korak step_{B6} se dolazi iz step₅₀ ukoliko signal operacije **RTS** ima vrednost 1. Pored toga u korak step_{B6} se dolazi i iz koraka step_{B5} kada signal operacije **RTI** ima vrednost 1. U oba slučaja se u koracima step_{B6} do step_{BE} vrednošću sa steka restaurira programski brojač PC_{15...0}. Sa steka se najpre skida niži a zatim i viši bajt registra PC_{15...0}. Stoga se u koraku step_{B6} vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR** bloka *bus* sadržaj registra SP_{15...0} bloka *addr* propušta kroz multiplekser MX1 i upisuje u registar MAR_{15...0}. Pored toga se i vrednošću 1 signala **decSP** dekrementira sadržaj registra SP_{15...0}. Čitanje se realizuje u koracima step_{B7} i step_{B8} na isti načina kao što se to radi u koracima step₀₂ i step₀₃ u kojima se čita prvi bajt instrukcije. Na kraju se u koraku step_{B9} vrednošću 1 signala **ldDWL** sadržaj registra MDR_{7...0} bloka *bus* upisuje u niži bajt registra DW_{7...0} bloka *bus*. Potom se u koraku step_{BA} vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR** sadržaj registra SP_{15...0} propušta kroz multiplekser MX1 i upisuje u registar MAR_{15...0}. Pored toga se i vrednošću 1 signala **decSP** dekrementira sadržaj registra SP_{15...0}. Čitanje se realizuje u koracima step_{BB} i step_{BC} na isti načina kao što se to radi u koracima step_{B7} i step_{B8} u kojima se čita niži bajt. Na kraju se u koraku step_{BD} vrednošću 1 signala **ldDWH** sadržaj registra MDR_{7...0} upisuje u viši bajt registar DW_{15...8}. Time je 16-to bitna vrednost skinuta sa steka i smeštena u registar DW_{15...0}. Konačno se u koraku step_{BE} sadržaj registra DW_{15...0} propušta kroz multiplekser MX bloka *fetch* i vrednošću 1 signala **ldPC** upisuje u registar PC_{15...0} i bezuslovno prelazi na step_{C0} i fazu opsluživanje prekida. !

step_{B6} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
 step_{B7} **br (if **hack** then step_{B7});**
 step_{B8} **eMAR, rdCPU,**
 br (if **fcCPU then step_{B8});**
 step_{B9} **ldDWL;**
 step_{BA} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
 step_{BB} **br (if **hack** then step_{BB});**
 step_{BC} **eMAR, rdCPU,**
 br (if **fcCPU then step_{BC});**
 step_{BD} **ldDWH;**
 step_{BE} **ldPC,**
 br step_{C0};



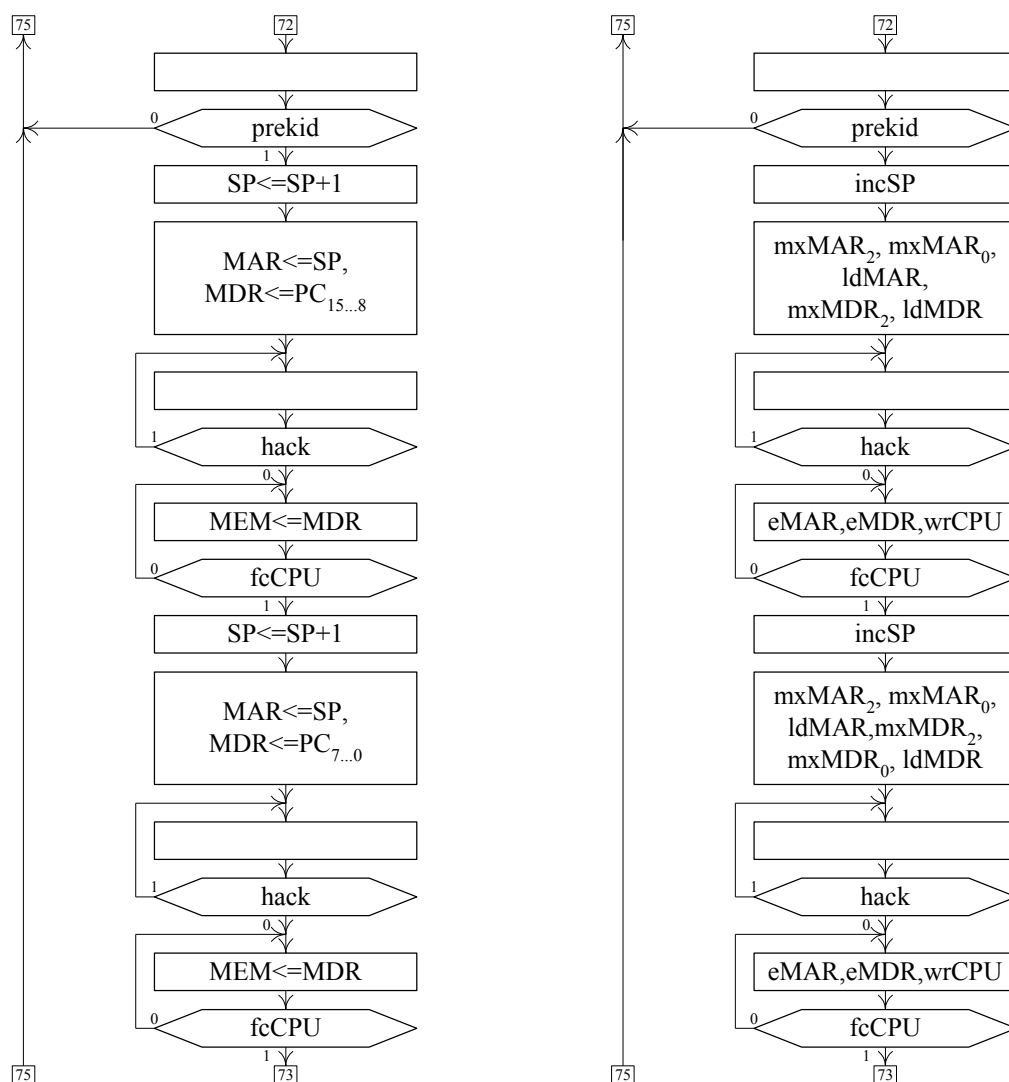
Slika 53 Izvršavanje operacije (sedamnaesti deo)

! Opsluživanje prekida !

! U korak step_{C0} se dolazi koraka step₀₆ ukoliko signal **PRCOD** ima vrednost 1, koraka step_{0D} ukoliko signal **PRADR** ima vrednost 1 i koraka step₅₁, step₅₂, ..., step_{BE} na završetku faze izvršavanje instrukcije. U koraku step_{C0} 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 step₀₀ započinje faza čitanje instrukcije sledeće instrukcije ili se produžava izvršavanje tekuće instrukcije i prelaskom na korak step_{C1} produžava faza opsluživanje prekida tekuće instrukcije. !

step_{C0} **br (if **prekid** then step₀₀);**

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



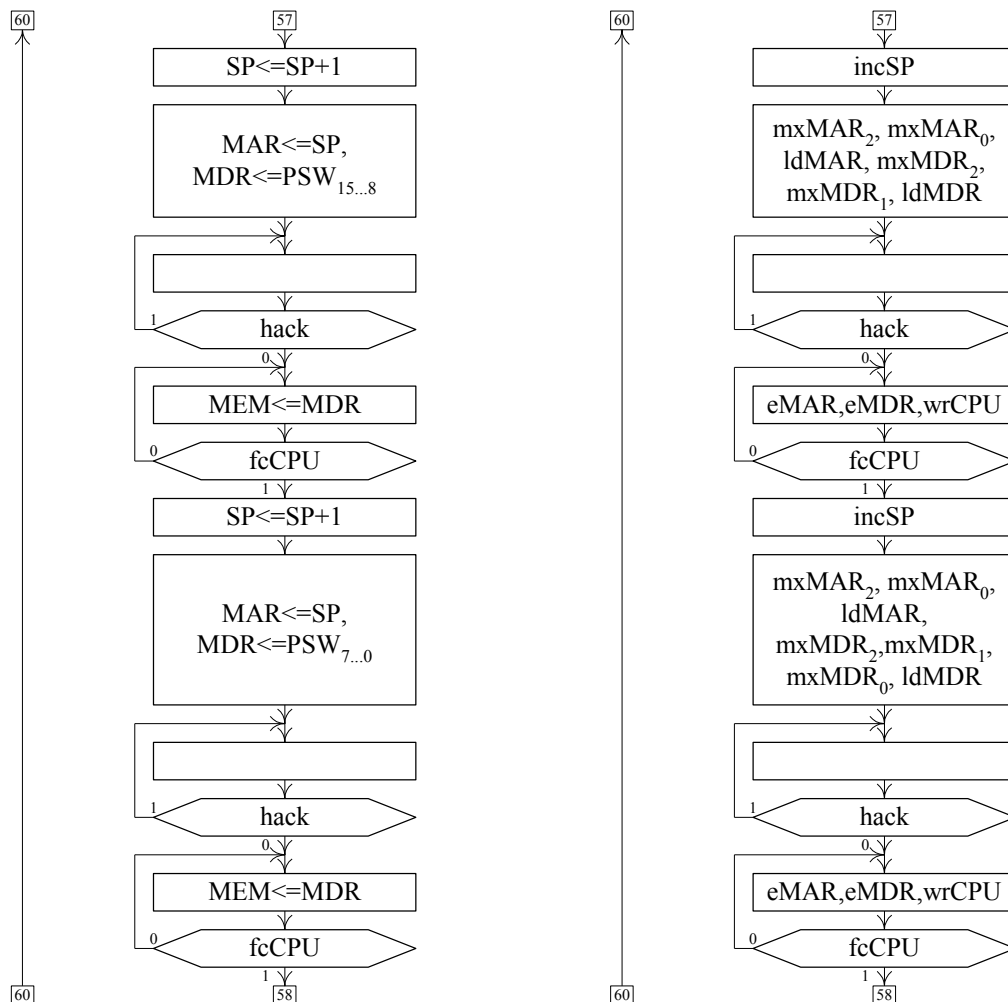
Slika 54 Opsluživanje prekida (prvi deo)

! Čuvanje konteksta procesora !

! Kontekst procesora i to $PC_{15...0}$ i $PSW_{15...0}$ se čuva u koracima $step_{C1}$ do $step_{D0}$. U koracima $step_{C1}$ do $step_{C8}$ se na stek stavlja programski brojač $PC_{15...0}$. Na stek se stavlja prvo viši a zatim i niži bajt registra $PC_{15...0}$. Stoga se najpre u koraku $step_{C1}$ vrednošću 1 signala **incSP** vrši inkrementiranje registra $SP_{15...0}$ bloka *addr*. Zatim se u koraku $step_{C2}$ vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR**, sadržaj registra $SP_{15...0}$ propušta kroz multiplekser MX1 bloka *bus* i upisuje u registar $MAR_{15...0}$ i vrednostima 1 signala **mxMDR₂** i **ldMDR** sadržaj višeg bajta registra $PC_{15...8}$ propušta kroz multiplekser MX2 i upisuje u registar $MDR_{7...0}$. Upis se realizuje u koracima $step_{C3}$ i $step_{C4}$ na isti načina kao što se to radi u koracima $step_{5C}$ i $step_{5D}$ instrukcije STB. Potom se u koraku $step_{C5}$ vrednošću 1 signala **incSP** vrši inkrementiranje registra $SP_{15...0}$. Zatim se u koraku $step_{C6}$ vrednostima 1 signala **mxMAR₂**, **mxMAR₀** i **ldMAR**, sadržaj registra $SP_{15...0}$ propušta kroz multiplekser MX1 bloka *bus* i upisuje u registar $MAR_{15...0}$ i vrednostima 1 signala **mxMDR₂**, **mxMDR₀**, **ldMDR** sadržaj nižeg bajta registra $PC_{7...0}$ propušta kroz multiplekser MX2 i upisuje u registar $MDR_{7...0}$. Upis se realizuje u koracima $step_{C7}$ i $step_{C8}$ na isti načina kao što se to radi u koracima $step_{C3}$ i $step_{C4}$ prilikom upisa višeg bajta.!

$step_{C1}$ **incSP**;
 $step_{C2}$ **mxMAR₂**, **mxMAR₀**, **ldMAR**, **mxMDR₂**, **ldMDR**;
 $step_{C3}$ *br (if hack then step_{C3});*
 $step_{C4}$ **eMAR**, **eMDR**, **wrCPU**,
br (if fcCPU then step_{C4});

step_{C5} **incSP;**
 step_{C6} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, mxMDR₀, ldMDR;**
 step_{C7} **br (if hack then step_{C7});**
 step_{C8} **eMAR, eMDR, wrCPU,**
 br (if fcCPU then step_{C8});



Slika 55 Opsluživanje prekida (drugi deo)

! U koracima step_{C9} do step_{D0} 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}. Stoga se najpre u koraku step_{C9} vrednošću 1 signala **incSP** vrši inkrementiranje registra SP_{15...0} bloka *addr*. Zatim se u koraku step_{CA} vrednostima 1 signala **mxMAR₂, mxMAR₀** i **ldMAR**, sadržaj registra SP_{15...0} propušta kroz multiplekser MX1 bloka *bus* i upisuje u registar MAR_{15...0} i vrednostima 1 signala **mxMDR₂, mxMDR₀** i **ldMDR** sadržaj višeg bajta registra PSW_{15...8} propušta kroz multiplekser MX2 bloka *bus* i upisuje u registar MDR_{7...0}. Upis se realizuje u koracima step_{CB} i step_{CC} na isti način kao što se to radi u koracima step_{5C} i step_{5D} instrukcije STB. Potom se u koraku step_{CD} vrednošću 1 signala **incSP** vrši inkrementiranje registra SP_{15...0}. Zatim se u koraku step_{CE} vrednostima 1 signala **mxMAR₂, mxMAR₀** i **ldMAR**, sadržaj registra SP_{15...0} propušta kroz multiplekser MX1 i upisuje u registar MAR_{15...0} i vrednostima 1 signala **mxMDR₂, mxMDR₁, mxMDR₀** i **ldMDR** sadržaj nižeg bajta registra PSW_{7...0} propušta kroz multiplekser MX2 i upisuje u registar MDR_{7...0}. Upis se realizuje u koracima step_{CF} i step_{D0} na isti način kao što se to radi u koracima step_{CB} i step_{CC} prilikom upisa višeg bajta. !

step_{C9} **incSP;**
 step_{CA} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, mxMDR₁, ldMDR;**
 step_{CB} **br (if hack then step_{CB});**
 step_{CC} **eMAR, eMDR, wrCPU,**

```

        br (if  $\overline{\text{fcCPU}}$  then stepCC);
stepCD incSP;
stepCE mxMAR2, mxMAR0, ldMAR, mxMDR2, mxMDR1, mxMDR0, ldMDR;
stepCF br (if hack then stepCF);
stepD0 eMAR, eMDR, wrCPU,
        br (if  $\overline{\text{fcCPU}}$  then stepD0);

```

! Utvrđivanje broja ulaza !

! U korak step_{D1} se dolazi iz step_{D0}. U koracima step_{D1} do step_{DB} 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_{D1} se vrši provera da li signal **PRINS** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak step_{D2} ili step_{D3}, respektivno. Ukoliko signal **PRINS** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku step_{D2} se vrednostima 0 signala **mxBR₁** i **mxBR₀** bloka *intr* sadržaj razreda IR_{23...16}, 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 **clPRINS** bloka *intr* se flip-flop PRINS postavlja na vrednost 0. Iz koraka step_{D2} se prelazi na korak step_{DC} radi utvrđivanja adrese prekidne rutine. !

```

stepD1 br (if PRINS then stepD3);
stepD2 ldBR, clPRINS,
        br stepDC;

```

! Provera da li postoji zahtev za prekid zbog greške u kodu operacije. !

! U koraku step_{D3} se vrši provera da li signal **PRCOD** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak step_{D4} ili step_{D5}, respektivno. Ukoliko signal **PRCOD** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku step_{D4} se vrednošću 1 signala **mxBR₁** bloka *intr* sadržaj UINT_{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 **clPRCOD** bloka *intr* flip-flop PRCOD postavlja na vrednost 0. Iz koraka step_{D4} se prelazi na korak step_{DC} radi utvrđivanja adrese prekidne rutine. !

```

stepD3 br (if PRCOD then stepD5);
stepD4 mxBR1, ldBR, clPRCOD,
        br stepDC;

```

! Provera da li postoji zahtev za prekid zbog greške u adresiranju. !

! U koraku step_{D5} se vrši provera da li signal **PRADR** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak step_{D6} ili step_{D7}, respektivno. Ukoliko signal **PRADR** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku step_{D6} se vrednošću 1 signala **mxBR₁** bloka *intr* sadržaj UINT_{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 **clPRADR** bloka *intr* flip-flop PRADR postavlja na vrednost 0. Iz koraka step_{D6} se prelazi na korak step_{DC} radi utvrđivanja adrese prekidne rutine. !

```

stepD5 br (if PRADR then stepD7);
stepD6 mxBR1, ldBR, clPRADR,
        br stepDC;

```

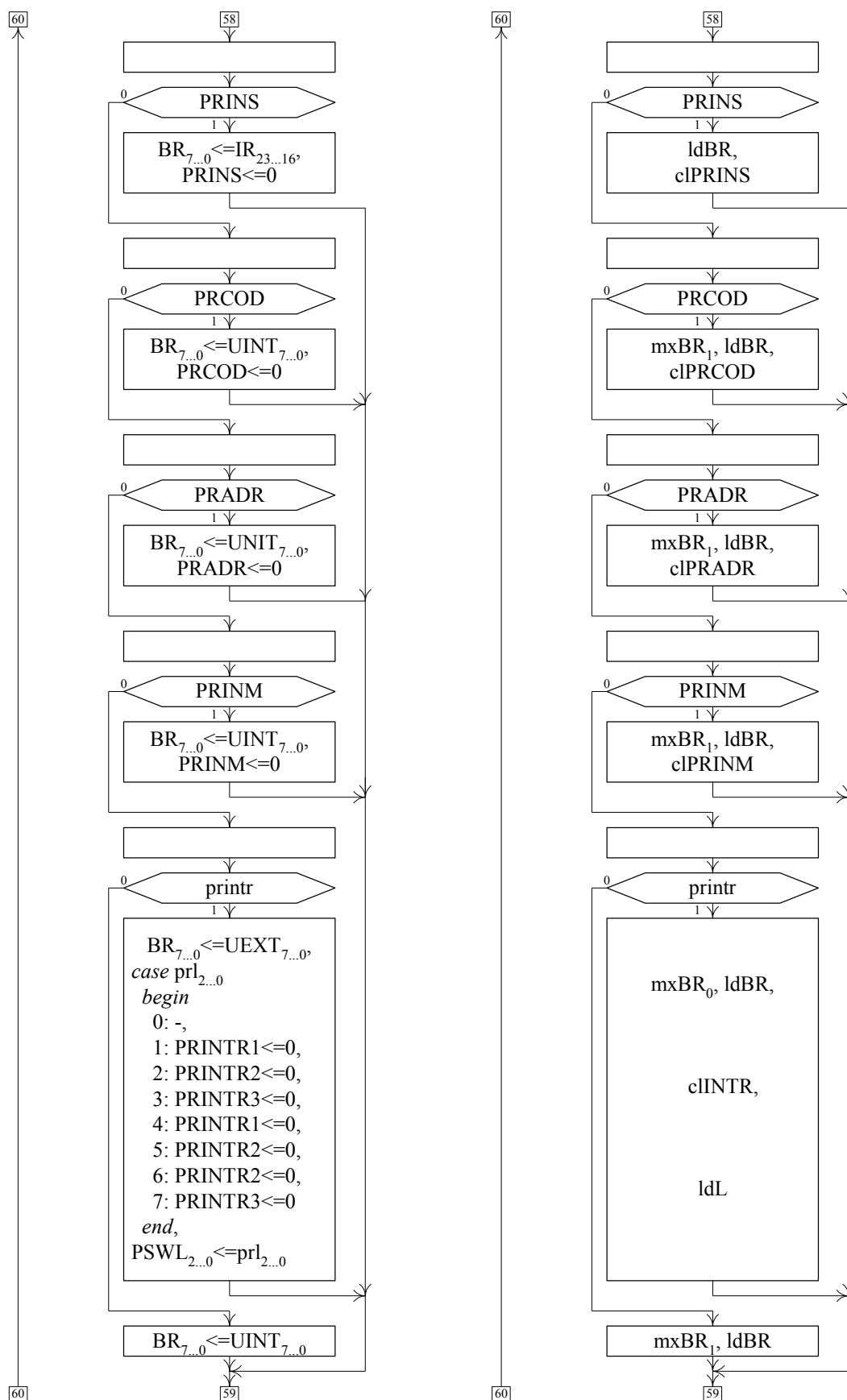
! Provera da li postoji spoljašnji nemaskirajući zahtev za prekid. !

! U koraku step_{D7} se vrši provera da li signal **PRINM** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak step_{D8} ili step_{D9}, respektivno. Ukoliko signal **PRINM** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku step_{D8} se vrednošću 1 signala **mxBR₁** bloka *intr* sadržaj UINT_{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 **clPRINM** bloka *intr* flip-flop PRINM postavlja na vrednost 0. Iz koraka step_{D8} se prelazi na korak step_{DC} radi utvrđivanja adrese prekidne rutine. !

```

stepD7 br (if PRINM then stepD9);
stepD8 mxBR1, ldBR, clPRINM,
        br stepDC;

```



Slika 56 Opsluživanje prekida (treći deo)

! Provera da li postoji spoljašnji maskirajući zahtev za prekid. !

! U koraku step_{D9} se vrši provera da li signal **printr** bloka *intr* ima vrednost 1 ili 0 i prelazi na korak step_{DA} ili step_{DB}, respektivno. Ukoliko signal **printr** ima vrednost 1 jer postoji ovaj zahtev za prekid, u koraku step_{DA} se vrednošću 1 signala **mxBR₀** bloka *intr* sadržaj UEXT_{7...0} sa izlaza koda CD3, 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-flopora PRINTR₇ do PRINTR₁ postavlja na vrednost 0. Flip-flop PRINTR₇ do PRINTR₁ koji se postavlja na vrednost 0 je selektovan binarnim vrednostima 7 do 1, respektivno, signala **prl₂** do **prl₀** 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 **prl₂** do **prl₀**, koji predstavljaju binarnu vrednost nivoa prioriteta prekidne rutine na koju se skače, upisuju u razrede PSWL₂ do PSWL₀ programske statusne reči PSW_{15...0} bloka *exec*. Iz koraka step_{DA} se prelazi na korak step_{DC} radi utvrđivanja adrese prekidne rutine. !

```

stepD9  br (if printr then stepDB);
stepDA  mxBR0, ldBR, clINTR, ldL,
        br stepDC;

```

! Prekid posle svake instrukcije !

! U korak step_{DB} se dolazi iz step_{D9} ukoliko signal **printr** ima vrednost 0. Kako se u ovaj korak dolazi jedino ukoliko se proverom signala **prekid** u koraku step_{C0} utvrđuje da postoji barem jedan zahtev za prekid i proverom signala **PRINS, PRCOD, PRADR, PRINM** i **printr** u koracima step_{D1}, step_{D3}, step_{D5}, step_{D7} i step_{D9} 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 **mxBR₁** bloka *intr* sadržaj UINT_{7...0} sa izlaza koda CD2, koji sadrži broj ulaza, propušta kroz multiplekser MX i vrednošću 1 signala **ldBR** upisuje u registar BR_{7...0}. Iz koraka step_{DB} se prelazi na korak step_{DC} radi utvrđivanja adrese prekidne rutine. !

```

stepDB  mxBR1, ldBR;

```

! Utvrđivanje adrese prekidne rutine !

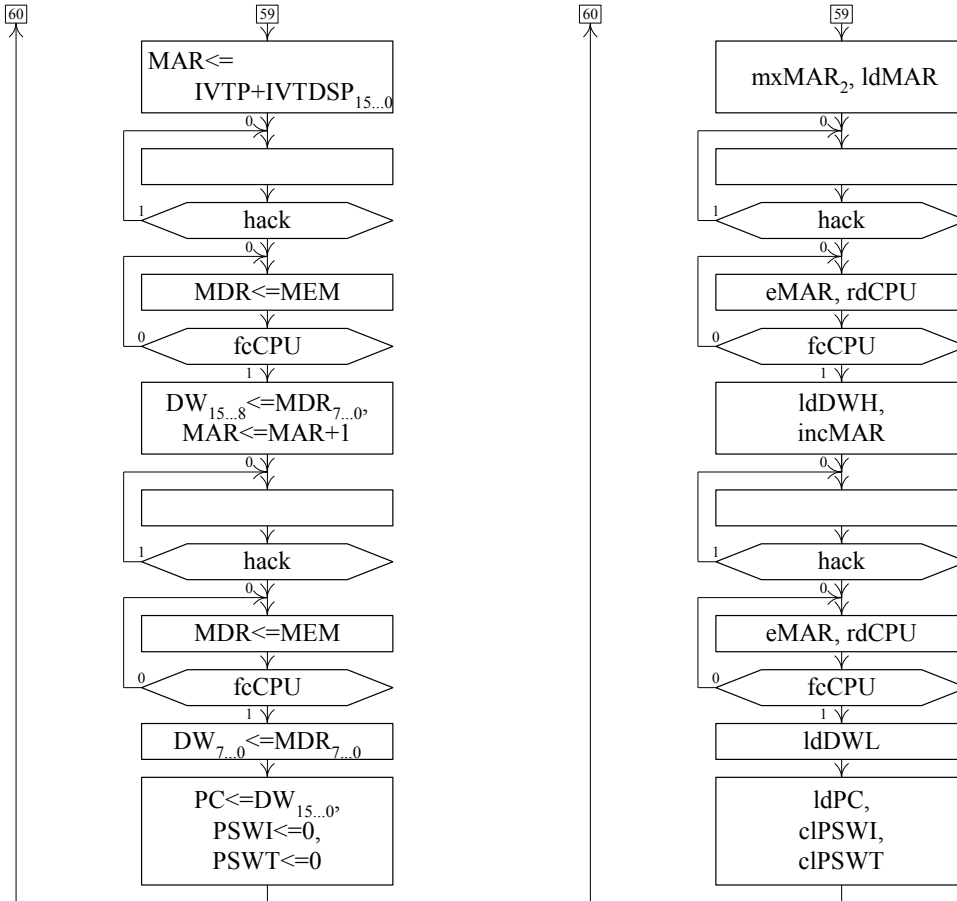
! U koracima step_{DC} do step_{E3} se na osnovu dobijenog broja ulaza i sadržaja registra koji ukazuje na početnu adresu tabele sa adresama prekidnih rutina, iz odgovarajućeg ulaza čita adresa prekidne rutine i upisuje u programski brojač PC_{15...0} bloka *fetch*. U koraku step_{DC} se vrednostima 0 signala **mxADDA₁, mxADDA₀, mxADDB₁** i **MXADDB₀** bloka *addr* kroz multipleksere MX2 i MX3 na ulaze sabirača ADD propuštaju sadržaj registra IVTP_{15...0} i sadržaj IVTDSP_{15...0} koji predstavlja sadržaj registra BR_{7...0} pomeren ulevo za jedno mesto i proširen nulama do dužine 16 bita. Vrednostima 1 signala **mxMAR₂** i **ldMAR** bloka *bus* se sadržaj ADD_{15...0} sa izlaza sabirača ADD propušta kroz multiplekser MX1 bloka *bus* i 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 prekidne rutine. Čitanje prvog bajta se realizuje u koracima step_{DD} i step_{DE}, a drugog bajta u koracima step_{E0} i step_{E1} na isti način kao u koracima step₀₂ i step₀₃ kod čitanja prvog bajta instrukcije. U koraku step_{DF} se prvi bajt vrednošću 1 signala **ldDWH** upisuje u viši bajt registra DW_{15...8} bloka *bus*, a vrednošću 1 signala **incMAR** adresni registar MAR_{15...0} inkrementira na adresu sledećeg bajta. U koraku step_{E2} se drugi bajt vrednošću 1 signala **ldDWL** upisuje u niži bajt registra DW_{7...0}. Time se u registru DW_{15...0} nalazi adresa prekidne rutine. Na kraju se u koraku step_{E3} vrednostima 0 signala **mxPC₁** i **mxPC₀** sadržaj registra DW_{15...0} propušta kroz multiplekser MX1 bloka *fetch* 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_{E3} se bezuslovno prelazi na step₀₀. !

```

stepDC  mxMAR2, ldMAR;
stepDD  br (if hack then stepDD);
stepDE  eMAR, rdCPU,
        br (if fcCPU then stepDE);
stepDF  ldDWH, incMAR;

```

step_{E0} *br (if **hack** then step_{E0});*
 step_{E1} **eMAR, rdCPU,**
 *br (if **fcCPU** then step_{E1});*
 step_{E2} **ldDWL;**
 step_{E3} **ldPC, clPSWI, clPSWT,**
 br step₀₀;



Slika 57 Opsluživanje prekida (četvrti deo)

4.2.3 Struktura upravljačke jedinice

U ovom odeljku se daje struktura upravljačke jedinice ožičene realizacije koja generiše dve vrste upravljačkih signala i to upravljačke signale blokova operacione jedinice **oper** i upravljačke signale upravljačke jedinice **uprav**. Upravljački signali blokova operacione jedinice **oper** se koriste u blokovima operacione jedinice **oper** radi izvršavanja mikrooperacija. Upravljački signali upravljačke jedinice **uprav** se koriste u upravljačkoj jedinici **uprav** radi inkrementiranja brojača koraka ili upisa nove vrednosti u brojač koraka i radi generisanja vrednosti za upis u brojač koraka.

Upravljački signali operacione jedinice bi mogli da se generišu na osnovu sekvence upravljačkih signala po koracima (tabela 16). Za svaki upravljački signal operacione jedinice trebalo bi proći kroz sekvencu upravljačkih signala po koracima, tražiti korake u kojima se pojavljuje dati signal i izraz za dati signal formirati kao uniju signala dekodovanih stanja brojača koraka koji odgovaraju koracima u kojima se pojavljuje dati signal.

Upravljački signali upravljačke jedinice se ne mogu generisati na osnovu sekvence upravljačkih signala po koracima (tabela 16), jer se u njoj ne pojavljuju upravljački signali upravljačke jedinice, već samo iskazi za skokove. Zbog toga je potrebno na osnovu sekvence

upravljačkih signala po koracima formirati sekvencu upravljačkih signala za upravljačku jedinicu ožičene realizacije. U njoj treba da se pored upravljačkih signala operacione jedinice pojave i upravljački signali upravljačke jedinice neophodni za realizaciju bezuslovnih, uslovnih i višestrukih uslovnih skokova specificiranih iskazima za skokove. Prilikom njenog formiranja primenjuje se različiti postupak za upravljačke signale operacione jedinice i za upravljačke signale upravljačke jedinice.

Za upravljačke signale operacione jedinice treba u sekvenci upravljačkih signala za upravljačku jedinicu ožičene realizacije staviti iskaze za signale onako kako se javljaju u sekvenci upravljačkih signala po koracima.

Za upravljačke signale upravljačke jedinice treba u sekvenci upravljačkih signala po koracima tražiti iskaze: *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})).

Umesto iskaza *br* step_A treba u sekvenci upravljačkih signala za upravljačku jedinicu ožičene realizacije staviti signal bezuslovnog skoka koji određuje da se bezuslovno prelazi na korak step_A i signal **val**_A koji određuje da treba formirati binarnu vrednost A za upis u brojač koraka. Simbolička oznaka signala bezuslovnog skoka je **bruncnd**. Koraci step_i u kojima se bezuslovni skokovi javljaju, koraci step_A na koje se bezuslovno skače, simboličke oznake signala **val**_A i vrednosti A u heksadecimalnom datu su u tabeli 17.

Tabela 17 Koraci step_i, step_A, signali **val**_A i vrednosti A za bezuslovne skokove

step _i	step _A	val _A	A
step ₀₆	step _{C0}	val _{C0}	C0
step _{0D}	step _{C0}	val _{C0}	C0
step ₁₈	step ₂₀	val ₂₀	20
step ₂₃	step ₅₀	val ₅₀	50
step ₂₄	step ₅₀	val ₅₀	50
step ₂₆	step ₃₈	val ₃₈	38
step ₂₈	step ₃₈	val ₃₈	38
step ₃₁	step ₃₈	val ₃₈	38
step ₃₃	step ₃₈	val ₃₈	38
step ₃₆	step ₃₈	val ₃₈	38
step _{3B}	step ₅₀	val ₅₀	50
step ₄₀	step ₅₀	val ₅₀	50
step ₄₂	step ₅₀	val ₅₀	50
step ₄₃	step ₅₀	val ₅₀	50
step ₅₁	step _{C0}	val _{C0}	C0
step ₅₂	step ₀₀	val ₀₀	00
step ₅₃	step _{C0}	val _{C0}	C0
step ₅₄	step _{C0}	val _{C0}	C0
step ₅₅	step _{C0}	val _{C0}	C0
step ₅₆	step _{C0}	val _{C0}	C0
step ₅₈	step _{C0}	val _{C0}	C0
step ₅₉	step _{C0}	val _{C0}	C0
step _{5E}	step _{C0}	val _{C0}	C0
step _{5F}	step _{C0}	val _{C0}	C0
step ₆₇	step _{C0}	val _{C0}	C0
step ₆₈	step _{C0}	val _{C0}	C0
step _{6E}	step _{C0}	val _{C0}	C0
step ₇₈	step _{C0}	val _{C0}	C0
step _{7D}	step _{C0}	val _{C0}	C0

step _i	step _A	val _A	A
step ₈₆	step _{C0}	val _{C0}	C0
step ₈₇	step _{C0}	val _{C0}	C0
step ₈₈	step _{C0}	val _{C0}	C0
step ₈₉	step _{C0}	val _{C0}	C0
step _{8A}	step _{C0}	val _{C0}	C0
step _{8B}	step _{C0}	val _{C0}	C0
step _{8C}	step _{C0}	val _{C0}	C0
step _{8E}	step _{C0}	val _{C0}	C0
step ₉₀	step _{C0}	val _{C0}	C0
step ₉₂	step _{C0}	val _{C0}	C0
step ₉₄	step _{C0}	val _{C0}	C0
step ₉₆	step _{C0}	val _{C0}	C0
step ₉₈	step _{C0}	val _{C0}	C0
step _{9A}	step _{C0}	val _{C0}	C0
step _{9C}	step _{C0}	val _{C0}	C0
step _{9E}	step _{C0}	val _{C0}	C0
step _{A0}	step _{C0}	val _{C0}	C0
step _{A2}	step _{C0}	val _{C0}	C0
step _{A3}	step _{C0}	val _{C0}	C0
step _{A4}	step _{C0}	val _{C0}	C0
step _{AD}	step _{C0}	val _{C0}	C0
step _{BE}	step _{C0}	val _{C0}	C0
step _{D2}	step _{DC}	val _{DC}	DC
step _{D4}	step _{DC}	val _{DC}	DC
step _{D6}	step _{D8}	val _{DC}	DC
step _{D8}	step _{D8}	val _{DC}	DC
step _{DA}	step _{D8}	val _{DC}	DC
step _{E3}	step ₀₀	val ₀₀	00

Umesto iskaza *br* (if **uslov** then step_A) treba u sekvenci upravljačkih signala za upravljačku jedinicu ožičene realizacije staviti signal uslovnog skoka pridružen signalu **uslov** koji treba da

ima vrednost 1 da bi se realizovao prelaz na korak $step_A$ i signal val_A koji određuje da treba formirati binarnu vrednost A za upis u brojač koraka u slučaju da signal **uslov** ima vrednost 1. Simboličke oznake pridruženih signala uslovnih skokova i signala uslova za sve iskaze ovog tipa koji se javljaju u sekvenci upravljačkih signala po koracima, dati su u tabeli 18. Koraci $step_i$ u kojima se uslovni skokovi javljaju, signali uslova **uslov**, koraci $step_A$ na koje se uslovno skače, simboličke oznake signala val_A i vrednosti A u heksadecimalnom dati su u tabeli 19.

Tabela 18 Signali uslovnih skokova i signali uslova

signal uslovnog skoka	signal uslova	signal uslovnog skoka	signal uslova
brnotSTART	$\overline{START}$	brstore	store
brhack	hack	brLDW	LDW
brnotfcCPU	$\overline{fcCPU}$	brdirreg	$\overline{dirreg}$
brnotgropr	$\overline{gropr}$	brnotbrpom	$\overline{brpom}$
brl1	l1	brnotprekid	$\overline{prekid}$
brnotgradr	$\overline{gradr}$	brnotPRINS	$\overline{PRINS}$
brl2_brnch	$\overline{l2_brnch}$	brnotPRCOD	$\overline{PRCOD}$
brl2_arlog	$\overline{l2_arlog}$	brnotPRADR	$\overline{PRADR}$
brl3_jump	$\overline{l3_jump}$	brnotPRINM	$\overline{PRINM}$
brl3_arlog	$\overline{l3_arlog}$	brnotprintr	$\overline{printr}$

Tabela 19 Koraci $step_i$, uslovi **uslov**, koraci $step_A$, signali val_A i vrednosti A za uslovne skokove

$step_i$	uslov	$step_A$	val_A	A
$step_{00}$	$\overline{START}$	$step_{00}$	val_{00}	00
$step_{02}$	hack	$step_{02}$	val_{02}	02
$step_{03}$	$\overline{fcCPU}$	$step_{03}$	val_{03}	03
$step_{05}$	gropr	$step_{07}$	val_{07}	07
$step_{07}$	l1	$step_{50}$	val_{50}	50
$step_{09}$	hack	$step_{09}$	val_{09}	09
$step_{0A}$	$\overline{fcCPU}$	$step_{0A}$	val_{0A}	0A
$step_{0C}$	gradr	$step_{0E}$	val_{0E}	0E
$step_{0E}$	$\overline{l2_brnch}$	$step_{50}$	val_{50}	50
$step_{0F}$	$\overline{l2_arlog}$	$step_{20}$	val_{20}	20
$step_{11}$	hack	$step_{11}$	val_{11}	11
$step_{12}$	$\overline{fcCPU}$	$step_{12}$	val_{12}	12
$step_{13}$	$\overline{l3_jump}$	$step_{50}$	val_{50}	50
$step_{14}$	$\overline{l3_arlog}$	$step_{20}$	val_{20}	20
$step_{16}$	hack	$step_{16}$	val_{16}	16
$step_{17}$	$\overline{fcCPU}$	$step_{17}$	val_{17}	17
$step_{21}$	store	$step_{50}$	val_{50}	50
$step_{22}$	LDW	$step_{24}$	val_{24}	24
$step_{25}$	store	$step_{50}$	val_{50}	50
$step_{27}$	store	$step_{50}$	val_{50}	50
$step_{2A}$	hack	$step_{2A}$	val_{2A}	2A
$step_{2B}$	$\overline{fcCPU}$	$step_{2B}$	val_{2B}	2B
$step_{2D}$	hack	$step_{2D}$	val_{2D}	2D
$step_{2E}$	$\overline{fcCPU}$	$step_{2E}$	val_{2E}	2E

$step_i$	uslov	$step_A$	val_A	A
$step_{6B}$	$\overline{fcCPU}$	$step_{6B}$	val_{6B}	6B
$step_{70}$	hack	$step_{70}$	val_{70}	70
$step_{71}$	$\overline{fcCPU}$	$step_{71}$	val_{71}	71
$step_{74}$	hack	$step_{74}$	val_{74}	74
$step_{75}$	$\overline{fcCPU}$	$step_{75}$	val_{75}	75
$step_{7B}$	hack	$step_{7B}$	val_{7B}	7B
$step_{7C}$	$\overline{fcCPU}$	$step_{7C}$	val_{7C}	7C
$step_{80}$	hack	$step_{80}$	val_{80}	80
$step_{81}$	$\overline{fcCPU}$	$step_{81}$	val_{81}	81
$step_{84}$	hack	$step_{84}$	val_{84}	84
$step_{85}$	$\overline{fcCPU}$	$step_{85}$	val_{85}	85
$step_{A1}$	brpom	$step_{C0}$	val_{C0}	C0
$step_{A7}$	hack	$step_{A7}$	val_{A7}	A7
$step_{A8}$	$\overline{fcCPU}$	$step_{A8}$	val_{A8}	A8
$step_{AB}$	hack	$step_{AB}$	val_{AB}	AB
$step_{AC}$	$\overline{fcCPU}$	$step_{AC}$	val_{AC}	AC
$step_{AF}$	hack	$step_{AF}$	val_{AF}	AF
$step_{B0}$	$\overline{fcCPU}$	$step_{B0}$	val_{B0}	B0
$step_{B3}$	hack	$step_{B3}$	val_{B3}	B3
$step_{B4}$	$\overline{fcCPU}$	$step_{B4}$	val_{B4}	B4
$step_{B7}$	hack	$step_{B7}$	val_{B7}	B7
$step_{B8}$	$\overline{fcCPU}$	$step_{B8}$	val_{B8}	B8
$step_{BB}$	hack	$step_{BB}$	val_{BB}	BB
$step_{BC}$	$\overline{fcCPU}$	$step_{BC}$	val_{BC}	BC

step ₃₀	store	step ₅₀	val₅₀	50
step ₃₂	store	step ₅₀	val₅₀	50
step ₃₅	store	step ₅₀	val₅₀	50
step ₃₇	store	step ₅₀	val₅₀	50
step ₃₈	hack	step ₃₈	val₃₈	38
step ₃₉	fcCPU	step ₃₉	val₃₈	39
step _{3A}	LDW	step _{3C}	val_{3C}	3C
step _{3D}	hack	step _{3D}	val_{3D}	3D
step _{3E}	fcCPU	step _{3E}	val_{3E}	3E
step ₄₁	LDW	step ₄₃	val₄₃	43
step _{5A}	dirreg	step _{5F}	val_{5F}	5F
step _{5C}	hack	step _{5C}	val_{5C}	5C
step _{5D}	fcCPU	step _{5D}	val_{5D}	5D
step ₆₀	dirreg	step ₆₈	val₆₈	68
step ₆₂	hack	step ₆₂	val₆₂	62
step ₆₃	fcCPU	step ₆₃	val₆₃	63
step ₆₅	hack	step ₆₅	val₆₅	65
step ₆₆	fcCPU	step ₆₆	val₆₆	66
step _{6A}	hack	step _{6A}	val_{6A}	6A

step _{C0}	prekid	step ₀₀	val₀₀	00
step _{C3}	hack	step _{C3}	val_{C3}	C3
step _{C4}	fcCPU	step _{C4}	val_{C4}	C4
step _{C7}	hack	step _{C7}	val_{C7}	C7
step _{C8}	fcCPU	step _{C8}	val_{C8}	C8
step _{CB}	hack	step _{CB}	val_{CB}	CB
step _{CC}	fcCPU	step _{CC}	val_{CC}	CC
step _{CF}	hack	step _{CF}	val_{CF}	CF
step _{D0}	fcCPU	step _{D0}	val_{D0}	D0
step _{D1}	PRINS	step _{D3}	val_{D3}	D3
step _{D3}	PRCOD	step _{D5}	val_{D5}	D5
step _{D5}	PRADR	step _{D7}	val_{D7}	D7
step _{D7}	PRINM	step _{D9}	val_{D9}	D9
step _{D9}	printr	step _{DB}	val_{DB}	DB
step _{DD}	hack	step _{DD}	val_{DD}	DD
step _{DE}	fcCPU	step _{DE}	val_{DE}	DE
step _{E0}	hack	step _{E0}	val_{E0}	E0
step _{E1}	fcCPU	step _{E1}	val_{E1}	E1

Umesto iskaza *br* (*case* (**uslov₁**, ..., **uslov_n**) *then* (**uslov₁**, step_{A1}), ..., (**uslov_n**, step_{An})) treba u sekvenci upravljačkih signala za upravljačku jedinicu ožičene realizacije staviti signal višestrukog uslovnog skoka pridružen signalima **uslov₁**, **uslov₂**, ..., **uslov_n** od kojih jedan treba da ima vrednost 1 da bi se realizovao prelazak na jedan od koraka step_{A1}, step_{A2}, ..., step_{An}. Koraci step_i u kojima se višestruki uslovni skokovi javljaju i simboličke oznake pridruženih signala višestrukih uslovnih skokova za sve korake ovog tipa koji se javljaju u sekvenci upravljačkih signala po koracima dati su u tabeli 20. Signali uslova **uslov₁**, **uslov₂**, ..., **uslov_n**, koraci step_{A1}, step_{A2}, ..., step_{An} na koje se uslovno skače i vrednosti A1, A2, ..., An u heksadecimalnom koje treba da se upišu u brojač koraka u zavisnosti od toga koji od signala uslova **uslov₁**, **uslov₂**, ..., **uslov_n** ima vrednost 1 za višestruke uslovne skokove u koracima step₂₀ i step₅₀ dati su tabelama 21 i 22.

Tabela 20 Koraci sa višestrukim uslovnim skokovima i signali višestrukih uslovnih skokova

step _i	signal
step ₂₀	bradr
step ₅₀	bropr

Tabela 21 Signali uslova, koraci na koje se skače i vrednosti za upis u brojač koraka za višestruki uslovni skok u koraku step₂₀

uslov	step _A	A
regdir	step ₂₁	21
regind	step ₂₅	25
memdir	step ₂₇	27
memind	step ₂₉	29
regindpom	step ₃₂	32
bxpom	step ₃₄	34
pcpom	step ₃₇	37
imm	step ₄₁	41

Tabela 22 Signali uslova, koraci na koje se skače i vrednosti za upis u brojač koraka za višestruki uslovni skok u koraku step₅₀

uslov	step _A	A
NOP	step ₅₁	51
uslov	step _A	A
RORC	step _{9D}	9D

HALT	step ₅₂	52
INTD	step ₅₃	53
INTE	step ₅₄	54
TRPD	step ₅₅	55
TRPE	step ₅₆	56
LDB	step ₅₇	57
LDW	step ₅₉	59
STB	step _{5A}	5A
STW	step ₆₀	60
POPB	step ₆₉	69
POPW	step _{6F}	6F
PUSHB	step ₇₉	79
PUSHW	step _{7E}	7E
LDIVTP	step ₈₇	87
STIVTP	step ₈₈	88
LDIMR	step ₈₉	89
STIMR	step _{8A}	8A
LDSP	step _{8B}	8B
STSP	step _{8C}	8C
ADD	step _{8D}	8D
SUB	step _{8F}	8F
INC	step ₉₁	91
DEC	step ₉₃	93
AND	step ₉₅	95
OR	step ₉₇	97
XOR	step ₉₉	99
NOT	step _{9B}	9B
ASR	step _{9D}	9D
LSR	step _{9D}	9D
ROR	step _{9D}	9D

ASL	step _{9F}	9F
LSL	step _{9F}	9F
ROL	step _{9F}	9F
ROLC	step _{9F}	9F
BEQL	step _{A1}	A1
BNEQ	step _{A1}	A1
BNEG	step _{A1}	A1
BNNEG	step _{A1}	A1
BOVF	step _{A1}	A1
BNOVF	step _{A1}	A1
BCAR	step _{A1}	A1
BNCAR	step _{A1}	A1
BGRT	step _{A1}	A1
BGRE	step _{A1}	A1
BLSS	step _{A1}	A1
BLSSE	step _{A1}	A1
BGRT	step _{A1}	A1
BGRE	step _{A1}	A1
BLSS	step _{A1}	A1
BLSSE	step _{A1}	A1
BGRTU	step _{A1}	A1
BGRTEU	step _{A1}	A1
BLSSU	step _{A1}	A1
BLSSEU	step _{A1}	A1
JMP	step _{A3}	A3
INT	step _{A4}	A4
JSR	step _{A5}	A5
RTI	step _{AE}	AE
RTS	step _{B6}	B6

Iz izloženog se vidi da su upravljački signali za upravljačku jedinicu ožičene realizacije signal bezuslovnog skoka **bruncnd**, signali uslovnih skokova (tabela 18) i višestrukih uslovnih skokova (tabela 20) i signali **val_A** za bezuslovne (tabela 17) i uslovne (tabela 19) skokove.

Po opisanom postupku je, na osnovu sekvence upravljačkih signala po koracima (tabela tabela 16), formirana sekvenca upravljačkih signala za upravljačku jedinicu ožičene realizacije (tabela 23). Jedna linija u toj sekvenci ima sledeću formu: na levoj strani nalazi se signal dekodovanog stanja brojača koraka, u sredini je niz upravljačkih signala operacione i upravljačke jedinice koji imaju vrednost 1 kada dati signal dekodovanog stanja brojača koraka ima vrednost 1, dok komentar, tamo gde postoji, počinje usklikom (!) i proteže se do sledećeg uskličnika (!).

Upravljački signali operacione jedinice i upravljačke jedinice se generišu na identičan način na osnovu sekvence upravljačkih za upravljačku jedinicu ožičene realizacije (tabela 23). Za svaki upravljački signal operacione jedinice i upravljačke jedinice treba proći kroz sekvencu upravljačkih signala po koracima, tražiti korake u kojima se pojavljuje dati signal i izraz za dati signal formirati kao uniju signala dekodovanih stanja brojača koraka koji odgovaraju koracima u kojima se pojavljuje dati signal.

Tabela 23 Sekvenca upravljačkih signala za upravljačku jedinicu ožičene realizacije

! Čitanje instrukcije !

T₀₀ **brnotSTART, val₀₀;**
T₀₁ **ldMAR, incPC;**
T₀₂ **brhack, val₀₂;**

T ₀₃	eMAR, rdCPU, brnotfcCPU, val₀₃;
T ₀₄	ldIR0;
T ₀₅	brnotgropr, val₀₇;
T ₀₆	stPRCOD, bruncnd, val_{C0};
T ₀₇	brl1, val₅₀;
T ₀₈	ldMAR, incPC;
T ₀₉	brhack, val₀₉;
T _{0A}	eMAR, rdCPU, brnotfcCPU, val_{0A};
T _{0B}	ldIR1, ldGPRADR;
T _{0C}	brnotgradr, val_{0E};
T _{0D}	stPRADR, bruncnd, val_{C0};
T _{0E}	brl2_brnch, val₅₀;
T _{0F}	brl2_arlog, val₂₀;
T ₁₀	ldMAR, incPC;
T ₁₁	brhack, val₁₁;
T ₁₂	eMAR, rdCPU, brnotfcCPU, val₁₂;
T ₁₃	ldIR2, brl3_jump, val₅₀;
T ₁₄	brl3_arlog, val₂₀;
T ₁₅	ldMAR, incPC;
T ₁₆	brhack, val₁₆;
T ₁₇	eMAR, rdCPU, brnotfcCPU, val₁₇;
T ₁₈	ldIR3, bruncnd, val₂₀;
! Formiranje adrese i čitanje operanda !	
T ₂₀	bradr;
! Registarsko direktno adresiranje !	
T ₂₁	brstore, val₅₀;
T ₂₂	brLDW, val₂₄;
T ₂₃	ldBB, bruncnd, val₅₀;
T ₂₄	ldBW, bruncnd, val₅₀;
! Registarsko indirektno adresiranje !	
T ₂₅	mxMAR₀, ldMAR, brstore, val₅₀;
T ₂₆	bruncnd, val₃₈;
! Memorijsko direktno adresiranje !	
T ₂₇	mxMAR₁, ldMAR, brstore, val₅₀;
T ₂₈	bruncnd, val₃₈;
! Memorijsko indirektno adresiranje !	
T ₂₉	mxMAR₁, ldMAR;
T _{2A}	brhack, val_{2A};
T _{2B}	eMAR, rdCPU, brnotfcCPU, val_{2B};
T _{2C}	ldDWH, incMAR;

	T _{2D}	brhack, val_{2D};
	T _{2E}	eMAR, rdCPU,
		brnotfcCPU, val_{2E};
	T _{2F}	ldDWL;
	T ₃₀	mxMAR₁, mxMAR₀, ldMAR,
		brstore, val₅₀;
	T ₃₁	bruncnd, val₃₈;
	! Registarsko indirektno adresiranje sa pomerajem !	
	T ₃₂	mxADDA₀, mxADDB₀, mxMAR₂, ldMAR,
		brstore, val₅₀;
	T ₃₃	bruncnd, val₃₈;
	! Bazno indeksno adresiranje sa pomerajem !	
	T ₃₄	mxADDA₀, mxADDB₀, ldCW, incGPRAR;
	T ₃₅	mxADDA₁, mxADDA₀, mxADDB₁, mxMAR₂, ldMAR,
		brstore, val₅₀;
	T ₃₆	bruncnd, val₃₈;
	! PC relativno adresiranje !	
	T ₃₇	mxADDA₁, mxADDB₀, mxMAR₂, ldMAR,
		brstore, val₅₀;
	! Čitanje operanda !	
	T ₃₈	brhack, val₃₈;
	T ₃₉	eMAR, rdCPU,
		brnotfcCPU, val₃₉;
	T _{3A}	brLDW, val_{3C};
	T _{3B}	mxBB₀, ldBB,
		bruncnd, val₅₀;
	T _{3C}	ldDWH, incMAR;
	T _{3D}	brhack, val_{3D};
	T _{3E}	eMAR, rdCPU,
		brnotfcCPU, val_{3E};
	T _{3F}	ldDWL;
	T ₄₀	mxBW₀, ldBW,
		bruncnd, val₅₀;
	! Neposredno adresiranje !	
	T ₄₁	brLDW, val₄₃;
	T ₄₂	mxBB₁, ldBB,
		bruncnd, val₅₀;
	T ₄₃	mxBW₁, ldBW,
		bruncnd, val₅₀;
	! Izvršavanje operacije !	
	T ₅₀	bropr;
	! NOP !	
	T ₅₁	bruncnd, val_{C0};
	! HALT !	
	T ₅₂	clSTART,
		bruncnd, val₀₀;
	! INTD !	
	T ₅₃	clPSWI,
		bruncnd, val_{C0};
	! INTE !	
	T ₅₄	stPSWI,
		bruncnd, val_{C0};

! TRPD !	
T ₅₅	clPSWT, bruncnd, val_{C0};
! TRPE !	
T ₅₆	stPSWT, bruncnd, val_{C0};
! LDB !	
T ₅₇	mxAB, ldAB;
T ₅₈	ldN, ldZ, ldC, ldV, bruncnd, val_{C0};
! LDW !	
T ₅₉	ldAW, bruncnd, val_{C0};
! STB !	
T _{5A}	brdirreg, val_{5F};
T _{5B}	mxMDR₀, ldMDR;
T _{5C}	brhack, val_{5C};
T _{5D}	eMAR, eMDR, wrCPU, brnotfcCPU, val_{5D};
T _{5E}	bruncnd, val_{C0};
T _{5F}	wrGPR, bruncnd, val_{C0};
! STW !	
T ₆₀	brdirreg, val₆₈;
T ₆₁	mxMDR₁, ldMDR;
T ₆₂	brhack, val₆₂;
T ₆₃	eMAR, eMDR, wrCPU, brnotfcCPU, val₆₃;
T ₆₄	mxMDR₁, mxMDR₀, ldMDR, incMAR;
T ₆₅	brhack, val₆₅;
T ₆₆	eMAR, eMDR, wrCPU, brnotfcCPU, val₆₆;
T ₆₇	bruncnd, val_{C0};
T ₆₈	mxGPR, wrGPR, bruncnd, val_{C0};
! POPB !	
T ₆₉	mxMAR₂, mxMAR₀, ldMAR, decSP;
T _{6A}	brhack, val_{6A};
T _{6B}	eMAR, rdCPU, brnotfcCPU, val_{6B};
T _{6C}	mxBB₀, ldBB;
T _{6D}	mxAB, ldAB;
T _{6E}	ldN, ldZ, ldC, ldV, bruncnd, val_{C0};
! POPW !	
T _{6F}	mxMAR₂, mxMAR₀, ldMAR, decSP;
T ₇₀	brhack, val₇₀;
T ₇₁	eMAR, rdCPU, brnotfcCPU, val₇₁;
T ₇₂	ldDWL;
T ₇₃	mxMAR₂, mxMAR₀, ldMAR, decSP;
T ₇₄	brhack, val₇₄;

T ₇₅	eMAR, rdCPU,
	brnotfcCPU, val₇₅;
T ₇₆	ldDWH;
T ₇₇	mxBW₀, ldBW;
T ₇₈	ldAW,
	bruncnd, val_{C0};
! PUSHB !	
T ₇₉	incSP;
T _{7A}	mxMAR₂, mxMAR₀, ldMAR, mxMDR₀, ldMDR;
T _{7B}	brhack, val_{7B};
T _{7C}	eMAR, eMDR, wrCPU,
	brnotfcCPU, val_{7C};
T _{7D}	bruncnd, val_{C0};
! PUSHW !	
T _{7E}	incSP;
T _{7F}	mxMAR₂, mxMAR₀, ldMAR, mxMDR₁, ldMDR;
T ₈₀	brhack, val₈₀;
T ₈₁	eMAR, eMDR, wrCPU,
	brnotfcCPU, val₈₁;
T ₈₂	incSP;
T ₈₃	mxMAR₂, mxMAR₀, ldMAR, mxMDR₁, mxMDR₀, ldMDR;
T ₈₄	brhack, val₈₄;
T ₈₅	eMAR, eMDR, wrCPU,
	brnotfcCPU, val₈₅;
T ₈₆	bruncnd, val_{C0};
! LDIVTP !	
T ₈₇	mxAW₁, ldAW,
	bruncnd, val_{C0};
! STIVTP !	
T ₈₈	ldIVTP,
	bruncnd, val_{C0};
! LDIMR !	
T ₈₉	mxAW₁, mxAW₀, ldAW,
	bruncnd, val_{C0};
! STIMR !	
T _{8A}	ldIMR,
	bruncnd, val_{C0};
! LDSP !	
T _{8B}	mxAW₀, ldAW,
	bruncnd, val_{C0};
! STSP !	
T _{8C}	ldSP,
	bruncnd, val_{C0};
! ADD !	
T _{8D}	add, ldAB, ldC, ldV;
T _{8E}	ldN, ldZ,
	bruncnd, val_{C0};
! SUB !	
T _{8F}	sub, ldAB, ldC, ldV;
T ₉₀	ldN, ldZ,
	bruncnd, val_{C0};
! INC !	
T ₉₁	inc, ldAB, ldC, ldV;

T ₉₂	ldN, ldZ, bruncnd, val_{C0};
! DEC !	
T ₉₃	dec, ldAB, ldC, ldV;
T ₉₄	ldN, ldZ, bruncnd, val_{C0};
! AND !	
T ₉₅	and, ldAB;
T ₉₆	ldN, ldZ, ldC, ldV, bruncnd, val_{C0};
! OR !	
T ₉₇	or, ldAB;
T ₉₈	ldN, ldZ, ldC, ldV, bruncnd, val_{C0};
! XOR !	
T ₉₉	xor, ldAB;
T _{9A}	ldN, ldZ, ldC, ldV, bruncnd, val_{C0};
! NOT !	
T _{9B}	not, ldAB;
T _{9C}	ldN, ldZ, ldC, ldV, bruncnd, val_{C0};
! ASR, LSR, ROR i ROLC !	
T _{9D}	shr, ldC;
T _{9E}	ldN, ldZ, ldV, bruncnd, val_{C0};
! ASL, LSL, ROL i ROLC !	
T _{9F}	shl, ldC;
T _{A0}	ldN, ldZ, ldV, bruncnd, val_{C0};
! BEQL,..., BLSSEU !	
T _{A1}	brnotbrpom, val_{C0};
T _{A2}	mxADDA₁, mxADDB₁, mxADDB₀, mxPC₀, ldPC, bruncnd, val_{C0};
! JMP !	
T _{A3}	mxPC₁, ldPC, bruncnd, val_{C0};
! INT !	
T _{A4}	stPRINS, bruncnd, val_{C0};
! JSR !	
T _{A5}	incSP;
T _{A6}	mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, ldMDR;
T _{A7}	brhack, val_{A7};
T _{A8}	eMAR, eMDR, wrCPU, brnotfcCPU, val_{A8};
T _{A9}	incSP;
T _{AA}	mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, mxMDR₀, ldMDR;
T _{AB}	brhack, val_{AB};
T _{AC}	eMAR, eMDR, wrCPU, brnotfcCPU, val_{AC};
T _{AD}	mxPC₁, ldPC, bruncnd, val_{C0};

! RTI !

T_{AE} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
T_{AF} **brhack, val_{AF};**
T_{B0} **eMAR, rdCPU,**
 brnotfcCPU, val_{B0};
T_{B1} **ldPSWL;**
T_{B2} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
T_{B3} **brhack, val_{B3};**
T_{B4} **eMAR, rdCPU,**
 brnotfcCPU, val_{B4};
T_{B5} **ldPSWH;**

! RTS !

T_{B6} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
T_{B7} **brhack, val_{B7};**
T_{B8} **eMAR, rdCPU,**
 brnotfcCPU, val_{B8};
T_{B9} **ldDWL;**
T_{BA} **mxMAR₂, mxMAR₀, ldMAR, decSP;**
T_{BB} **brhack, val_{BB};**
T_{BC} **eMAR, rdCPU,**
 brnotfcCPU, val_{BC};
T_{BD} **ldDWH;**
T_{BE} **ldPC,**
 bruncnd, val_{C0};

! Opsluživanje prekida !

T_{C0} **brnotprekid, val₀₀;**

! Čuvanje konteksta procesora !

T_{C1} **incSP;**
T_{C2} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, ldMDR;**
T_{C3} **brhack, val_{C3};**
T_{C4} **eMAR, eMDR, wrCPU,**
 brnotfcCPU, val_{C4};
T_{C5} **incSP;**
T_{C6} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, mxMDR₀, ldMDR;**
T_{C7} **brhack, val_{C7};**
T_{C8} **eMAR, eMDR, wrCPU,**
 brnotfcCPU, val_{C8};
T_{C9} **incSP;**
T_{CA} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, mxMDR₁, ldMDR;**
T_{CB} **brhack, val_{CB};**
T_{CC} **eMAR, eMDR, wrCPU,**
 brnotfcCPU, val_{CC};
T_{CD} **incSP;**
T_{CE} **mxMAR₂, mxMAR₀, ldMAR, mxMDR₂, mxMDR₁, mxMDR₀, ldMDR;**
T_{CF} **brhack, val_{CF};**
T_{D0} **eMAR, eMDR, wrCPU,**
 brnotfcCPU, val_{D0};

! Utvrđivanje broja ulaza !

! Provera da li postoji zahtev za prekid zbog izvršavanja instrukcije prekida INT. !

T_{D1} **brnotPRINS, val_{D3};**
T_{D2} **ldBR, clPRINS,**
 bruncnd, val_{DC};

! Provera da li postoji zahtev za prekid zbog greške u kodu operacije. !

T_{D3} **brnotPRCOD**, val_{D5};
T_{D4} **mxBR₁**, **ldBR**, **clPRCOD**,
 bruncnd, val_{DC};

! Provera da li postoji zahtev za prekid zbog greške u adresiranju. !

T_{D5} **brnotPRADR**, val_{D7};
T_{D6} **mxBR₁**, **ldBR**, **clPRADR**,
 bruncnd, val_{DC};

! Provera da li postoji spoljašnji nemaskirajući zahtev za prekid. !

T_{D7} **brnotPRINM**, val_{D9};
T_{D8} **mxBR₁**, **ldBR**, **clPRINM**,
 bruncnd, val_{DC};

! Provera da li postoji spoljašnji maskirajući zahtev za prekid. !

T_{D9} **brnotprintr**, val_{DB};
T_{DA} **mxBR₀**, **ldBR**, **clINTR**, **ldL**,
 bruncnd, val_{DC};

! Prekid posle svake instrukcije !

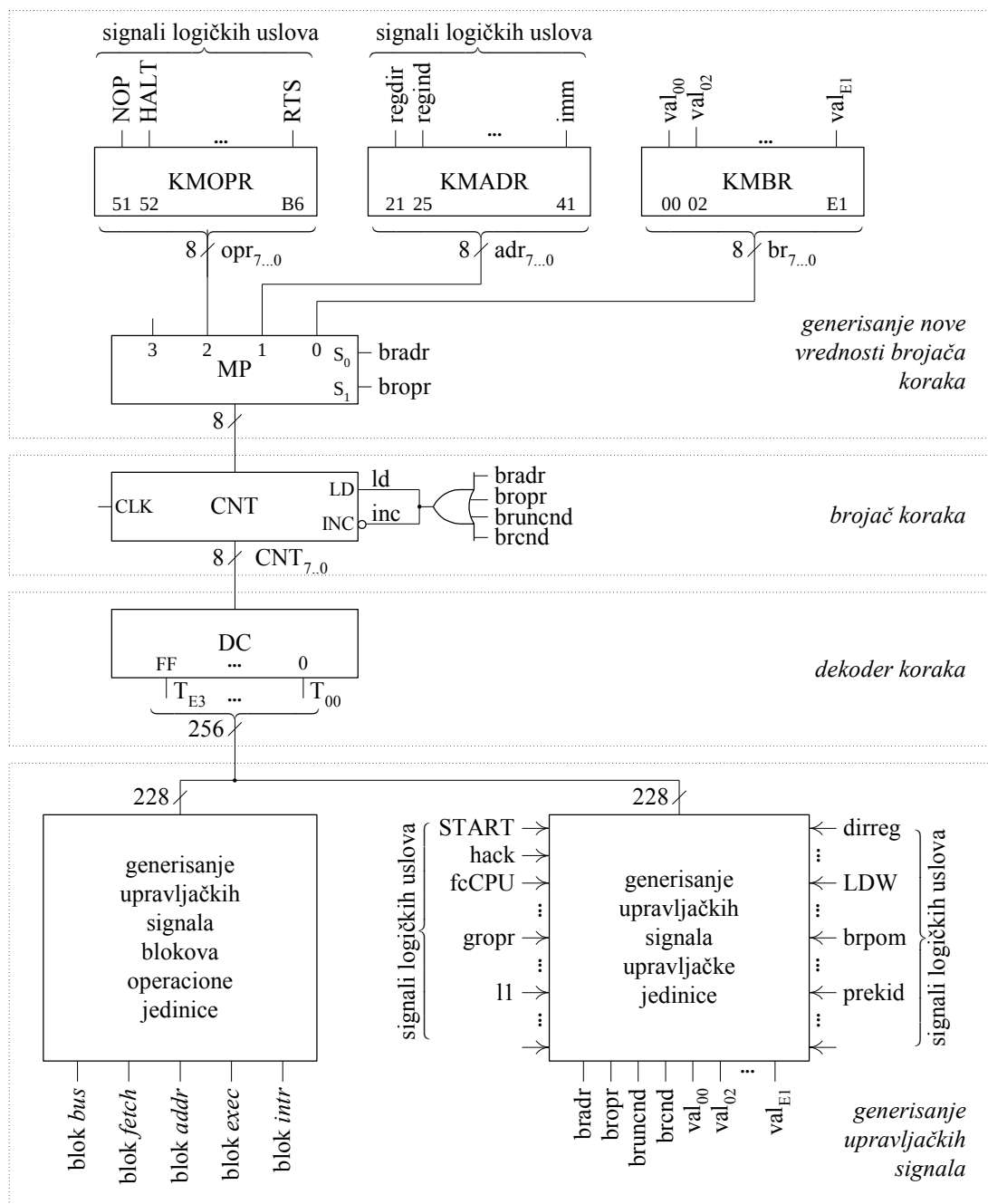
T_{DB} **mxBR₁**, **ldBR**;

! Utvrđivanje adrese prekidne rutine !

T_{DC} **mxMAR₂**, **ldMAR**;
T_{DD} **brhack**, val_{DD};
T_{DE} **eMAR**, **rdCPU**,
 brnotfcCPU, val_{DE};
T_{DF} **ldDWH**, **incMAR**;
T_{E0} **brhack**, val_{E0};
T_{E1} **eMAR**, **rdCPU**,
 brnotfcCPU, val_{E1};
T_{E2} **ldDWL**;
T_{E3} **ldPC**, **clPSWI**, **clPSWT**,
 bruncnd, val₀₀;

Struktura upravljačke jedinice ožičene realizacije je prikazana na slici 58. Upravljačka jedinica se sastoji iz sledećih blokova: blok *generisanje nove vrednosti brojača koraka*, blok *brojač koraka*, blok *dekođer koraka* i blok *generisanje upravljačkih signala*. Struktura i opis blokova upravljačke jedinice se daju u daljem tekstu.

Blok *generisanje nove vrednosti brojača koraka* se sastoji od kombinacionih mreža KMOPR, KMADR i KMBR sa multiplekserom MP i služi za generisanje i selekciju vrednosti koju treba upisati u brojač koraka CNT_{7...0}. Potreba za ovim se javlja kada treba odstupiti od sekvencijalnog izvršavanja mikrooperacija. Vrednosti koje treba upisati u brojač koraka 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 kombinacione mreže KMBR koja formira signale **br_{7...0}**. Selekcija jedne od tri grupe signala koji daju novu vrednost brojača koraka obezbeđuje se signalima **bropr** i **bradr** i to signalima **opr_{7...0}** ako signal **bropr**, ima vrednost 1, signalima **adr_{7...0}** ako signal **bradr** ima vrednost 1 i signalima **br_{7...0}** ako oba signala **bropr** i **bradr** imaju vrednost 0.



Slika 58 Struktura upravljačke jedinice

Kombinacionom mrežom KMOPR generišu se vrednosti (tabela 22) za realizaciju višestrukog uslovnog skoka u koraku $step_{50}$ sekvence upravljačkih signala. U zavisnosti od toga koji od signala **NOP**, ..., **RTS** ima vrednost 1 zavisi koja će od vrednosti iz tabele 22 da se pojavi tada na linijama $opr_{7...0}$. S obzirom da vrednost 1 signala dekovanog stanje brojača koraka T_{50} daje vrednost 1 signala višestrukog uslovnog skoka **bropr**, vrednost na linijama $opr_{7...0}$ prolazi tada kroz multiplekser MP i pojavljuje se na ulazima brojača koraka $CNT_{7...0}$.

Kombinacionom mrežom KMADR generišu se vrednosti (tabela 21) za realizaciju višestrukog uslovnog skoka u koraku $step_{20}$ sekvence upravljačkih signala. U zavisnosti od toga koji od signala **regdir**, ..., **imm** ima vrednost 1 zavisi koja će od vrednosti iz tabele 21 da se pojavi tada na linijama $adr_{7...0}$. S obzirom da vrednost 1 signala dekodovanog stanja brojača koraka T_{20} daje vrednost 1 signala višestrukog uslovnog skoka **bradr**, vrednost na

linijama **adr**_{7...0} prolazi tada kroz multiplekser MP i pojavljuje se na ulazima brojača koraka CNT_{7...0}.

Kombinacionom mrežom KMBR generišu se vrednosti za upis u brojač koraka CNT_{7...0} za bezuslovne skokove (tabela 17) i uslovne skokove (tabela 19) u sekvenci upravljačkih signala po koracima. U zavisnosti od toga koji od signala **val**₀₀, **val**₀₂, ..., **val**_{E1} ima vrednost 1 zavisi koja će od vrednosti iz tabela 17 i 19 tada da se pojavi na linijama **br**_{7...0}. Signali višestrukih uslovnih skokova **bropr** i **bradr** imaju vrednost 1 samo pri vrednostima 1 signala dekodovanih stanja brojača koraka T₅₀ i T₂₀, respektivno, dok u svim ostalim situacijama imaju vrednost 0. S obzirom da nijedan od ova dva signala nema vrednost 1 u stanjima brojača koraka kada treba realizovati bezuslovni ili neki od uslovnih skokova, vrednost na linijama **br**_{7...0} prolazi tada kroz multiplekser MP i pojavljuje se na ulazima brojača koraka CNT_{7...0}.

Blok *brojač koraka* sadrži brojač CNT_{7...0}. Brojač CNT_{7...0} svojom trenutnom vrednošću određuje koji će upravljački signali da imaju vrednost 1. Brojač CNT_{7...0} može da radi u sledećim režimima: režim inkrementiranja i režim skoka.

U režimu inkrementiranja pri pojavi signala takta vrši se uvećavanje sadržaja brojača CNT_{7...0} za jedan čime se obezbeđuje sekvencijalno generisanje upravljačkih signala iz sekvence upravljačkih signala (tabela 23). Ovaj režim rada se obezbeđuje vrednošću 1 signala **inc**. Signal **inc** ima vrednost 1 ukoliko svi signali **bropr**, **bradr**, **bruncnd** i **brcnd** imaju vrednost 0. Signali **bropr**, **bradr**, **bruncnd** i **brcnd** normalno imaju vrednost 0 sem u stanjima brojača koraka koja odgovaraju koracima kada treba realizovati višestruki uslovni skok, bezuslovni skok ili neki od uslovnih skokova i uslov skoka je ispunjen, pa jedan od ovih signala ima vrednost 1.

U režimu skoka pri pojavi signala takta vrši se upis nove vrednosti u brojač CNT_{7...0} čime se obezbeđuje odstupanje od sekvencijalnog generisanja upravljačkih signala iz sekvence upravljačkih signala (tabela 23). Ovaj režim rada se obezbeđuje vrednošću 1 signala **ld**. Signal **ld** ima vrednost 1 ako jedan od signala **bropr**, **bradr**, **bruncnd** i **brcnd** ima vrednost 1. Signali **bropr**, **bradr**, **bruncnd** i **brcnd** normalno imaju vrednost 0 sem u stanjima brojača koraka koja odgovaraju koracima kada treba realizovati višestruki uslovni skok, bezuslovni skok ili neki od uslovnih skokova i uslov skoka je ispunjen, pa jedan od ovih signala ima vrednost 1. Specifičan slučaj režima skoka je režim čekanja kada se pri pojavi signala takta vrši upis tekuće vrednosti brojača CNT_{7...0} u brojač CNT_{7...0} čime se obezbeđuje da se ne menja vrednost brojača CNT_{7...0}. Ovaj režim rada se obezbeđuje vrednošću 1 signala **ld**. Signal **ld** ima vrednost 1 kada signal **brcnd** ima vrednost 1 jer je:

- ciklus procesora na magistrali u toku pa signal **fcCPU** ima vrednost 0 ili
- magistrala prepuštena ulazno/izlaznom uređaju sa kontrolerom za direktan pristup memoriji, a procesor je došao u korak izvršavanja instrukcije u kome mu je magistrala potrebna, pa signal **hack** ima vrednost 1.

Brojač koraka CNT_{7...0} je dimenzionisan prema broju koraka u sekvenci upravljačkih signala (tabela 23). S obzirom da se upravljački signali svih faza izvršavanja instrukcija realizuju u opsegu od koraka T₀₀ do koraka T_{E3} usvojena je dužina brojača koraka CNT_{7...0} od 8 bita.

Blok *dekoder koraka* sadrži dekodeer DC. Na ulaze dekodeera DC vode se izlazi brojača CNT_{7...0}. Dekodovana stanja brojača CNT_{7...0} pojavljuju se kao signali **T**₀, **T**₁, ..., **T**_{FF} na izlazima dekodeera DC. Svakom koraku iz sekvence upravljačkih signala po koracima (tabela

16) dodeljeno je po jedno stanje brojača CNT_{7...0} određeno vrednošću signala T₀ do T_{FF} i to koraku step₀ signal T₀, koraku step₁ signal T₁, itd. (tabela 23).

Blok *generisanje upravljačkih signala* sadrži kombinacione mreže koje pomoću signala T₀, T₁, ..., T_{E3}, koji dolaze sa bloka *dekoder koraka*, signala logičkih uslova **START**, **hack**, ..., **prekid** koji dolaze iz blokova operacione jedinice i saglasno sekvenci upravljačkih signala (tabela 23) generišu dve grupe upravljačkih signala i to upravljačke signale blokova operacione jedinice **oper** i upravljačke signale upravljačke jedinice **uprav**.

Upravljački signali blokova operacione jedinice **oper** se daju posebno za svaki blok.

Upravljački signali bloka *bus* se generišu na sledeći način:

- $\text{mxMAR}_2 = T_{32} + T_{35} + T_{37} + T_{69} + T_{6F} + T_{73} + T_{7A} + T_{7F} + T_{83} + T_{A6} + T_{AA} + T_{AE} + T_{B2} + T_{B6} + T_{BA} + T_{C2} + T_{C6} + T_{CA} + T_{CE} + T_{DC}$
- $\text{mxMAR}_1 = T_{27} + T_{29} + T_{30}$
- $\text{mxMAR}_0 = T_{25} + T_{30} + T_{69} + T_{6F} + T_{73} + T_{7A} + T_{7F} + T_{83} + T_{A6} + T_{AA} + T_{AE} + T_{B2} + T_{B6} + T_{BA} + T_{C2} + T_{C6} + T_{CA} + T_{CE}$
- $\text{ldMAR} = T_{01} + T_{08} + T_{10} + T_{15} + T_{25} + T_{27} + T_{29} + T_{30} + T_{32} + T_{35} + T_{37} + T_{69} + T_{6F} + T_{73} + T_{7A} + T_{7F} + T_{83} + T_{A6} + T_{AA} + T_{AE} + T_{B2} + T_{B6} + T_{BA} + T_{C2} + T_{C6} + T_{CA} + T_{CE} + T_{DC}$
- $\text{incMAR} = T_{2C} + T_{3C} + T_{64} + T_{DF}$
- $\text{mxMDR}_2 = T_{A6} + T_{AA} + T_{C2} + T_{C6} + T_{CA} + T_{CE}$
- $\text{mxMDR}_1 = T_{61} + T_{64} + T_{7F} + T_{83} + T_{CA}$
- $\text{mxMDR}_0 = T_{5B} + T_{64} + T_{7A} + T_{83} + T_{AA} + T_{C6} + T_{CE}$
- $\text{ldMDR} = T_{5B} + T_{61} + T_{64} + T_{7A} + T_{7F} + T_{83} + T_{A6} + T_{AA} + T_{C2} + T_{C6} + T_{CA} + T_{CE}$
- $\text{eMAR} = T_{03} + T_{0A} + T_{12} + T_{17} + T_{2B} + T_{2E} + T_{39} + T_{3E} + T_{5D} + T_{63} + T_{66} + T_{6B} + T_{71} + T_{75} + T_{7C} + T_{81} + T_{85} + T_{A8} + T_{AC} + T_{B0} + T_{B4} + T_{B8} + T_{BC} + T_{C4} + T_{C8} + T_{CC} + T_{D0} + T_{DE} + T_{E1}$
- $\text{eMDR} = T_{5D} + T_{63} + T_{66} + T_{7C} + T_{81} + T_{85} + T_{A8} + T_{AC} + T_{C4} + T_{C8} + T_{CC} + T_{D0} +$
- $\text{rdCPU} = T_{03} + T_{0A} + T_{12} + T_{17} + T_{2B} + T_{2E} + T_{39} + T_{3E} + T_{6B} + T_{71} + T_{75} + T_{B0} + T_{B4} + T_{B8} + T_{BC} + T_{DE} + T_{E1}$
- $\text{wrCPU} = T_{5D} + T_{63} + T_{66} + T_{7C} + T_{81} + T_{85} + T_{A8} + T_{AC} + T_{C4} + T_{C8} + T_{CC} + T_{D0}$
- $\text{ldDWH} = T_{2C} + T_{3C} + T_{76} + T_{BD} + T_{DF}$
- $\text{ldDWL} = T_{2F} + T_{3F} + T_{72} + T_{B9} + T_{E2}$

Upravljački signali bloka *fetch* se generišu na sledeći način:

- $\text{mxPC}_1 = T_{A3} + T_{AD}$
- $\text{mxPC}_0 = T_{A2}$
- $\text{ldPC} = T_{A2} + T_{A3} + T_{AD} + T_{BE} + T_{E3}$
- $\text{incPC} = T_{01} + T_{08} + T_{10} + T_{15}$
- $\text{ldIR0} = T_{04}$
- $\text{ldIR1} = T_{0B}$
- $\text{ldIR2} = T_{13}$
- $\text{ldIR3} = T_{18}$

Upravljački signali bloka *addr* se generišu na sledeći način:

- $\text{ldGPRAR} = T_{0B}$
- $\text{incGPRAR} = T_{34}$
- $\text{mxGPR} = T_{68}$
- $\text{wrGPR} = T_{5F} + T_{68}$

- $ldSP = T_{8C}$
- $incSP = T_{79} + T_{7E} + T_{82} + T_{A5} + T_{A9} + T_{C1} + T_{C5} + T_{C9} + T_{CD}$
- $decSP = T_{69} + T_{6F} + T_{73} + T_{AE} + T_{B2} + T_{B6} + T_{BA}$
- $mxADDA_1 = T_{35} + T_{37} + T_{A2}$
- $mxADDA_0 = T_{32} + T_{34} + T_{35}$
- $mxADDB_1 = T_{35} + T_{A2}$
- $mxADDB_0 = T_{32} + T_{34} + T_{37} + T_{A2}$
- $ldCW = T_{34}$

Upravljački signali bloka *exec* se generišu na sledeći način:

- $mxAB = T_{57} + T_{6D}$
- $ldAB = T_{57} + T_{6D} + T_{8D} + T_{8F} + T_{91} + T_{93} + T_{95} + T_{97} + T_{99} + T_{9B}$
- $shr = T_{9D}$
- $shl = T_{9F}$
- $mxBB_1 = T_{42}$
- $mxBB_0 = T_{3B} + T_{6C}$
- $ldBB = T_{23} + T_{3B} + T_{42} + T_{6C}$
- $mxAW_1 = T_{87} + T_{89}$
- $mxAW_0 = T_{89} + T_{8B}$
- $ldAW = T_{59} + T_{78} + T_{87} + T_{89} + T_{8B}$
- $mxBW_1 = T_{43}$
- $mxBW_0 = T_{40} + T_{77}$
- $ldBW = T_{24} + T_{40} + T_{43} + T_{77}$
- $stPSWI = T_{54}$
- $clPSWI = T_{53} + T_{E3}$
- $stPSWT = T_{56}$
- $clPSWT = T_{55} + T_{E3}$
- $ldL = T_{DA}$
- $ldN = T_{58} + T_{6E} + T_{8E} + T_{90} + T_{92} + T_{94} + T_{96} + T_{98} + T_{9A} + T_{9C} + T_{9E} + T_{A0}$
- $ldZ = T_{58} + T_{6E} + T_{8E} + T_{90} + T_{92} + T_{94} + T_{96} + T_{98} + T_{9A} + T_{9C} + T_{9E} + T_{A0}$
- $ldC = T_{58} + T_{6E} + T_{8D} + T_{8F} + T_{91} + T_{93} + T_{96} + T_{98} + T_{9A} + T_{9C} + T_{9D} + T_{9F}$
- $ldV = T_{58} + T_{6E} + T_{8D} + T_{8F} + T_{91} + T_{93} + T_{96} + T_{98} + T_{9A} + T_{9C} + T_{9E} + T_{A0}$
- $ldPSWL = T_{B1}$
- $ldPSWH = T_{B5}$
- $clSTART = T_{52}$
- $add = T_{8D}$
- $sub = T_{8F}$
- $inc = T_{91}$
- $dec = T_{93}$
- $and = T_{95}$
- $or = T_{97}$
- $xor = T_{99}$
- $not = T_{9B}$

Upravljački signali bloka *intr* se generišu na sledeći način:

- $stPRINS = T_{A4}$
- $clPRINS = T_{D2}$
- $stPRCOD = T_{06}$

- $clPRCOD = T_{D4}$
- $stPRADR = T_{0D}$
- $clPRADR = T_{D2}$
- $clPRINM = T_{D4}$
- $clINTR = T_{D6}$
- $ldIMR = T_{8A}$
- $ldIVTP = T_{88}$
- $mxBR_1 = T_{D0} + T_{D2} + T_{D4} + T_{D7}$
- $mxBR_0 = T_{D6}$
- $ldBR = T_{CE} + T_{D0} + T_{D2} + T_{D4} + T_{D6} + T_{D7}$

Upravljački signali upravljačke jedinice *uprav* se generišu na sledeći način:

- $bradr = T_{20}$
- $bropr = T_{50}$
- $bruncnd = T_{06} + T_{0D} + T_{18} + T_{23} + T_{24} + T_{26} + T_{28} + T_{31} + T_{33} + T_{36} + T_{3B} + T_{40} + T_{42} + T_{43} + T_{51} + T_{52} + T_{53} + T_{54} + T_{55} + T_{56} + T_{58} + T_{59} + T_{5E} + T_{5F} + T_{67} + T_{68} + T_{6E} + T_{78} + T_{7D} + T_{86} + T_{87} + T_{88} + T_{89} + T_{8A} + T_{8B} + T_{8C} + T_{8E} + T_{90} + T_{92} + T_{94} + T_{96} + T_{98} + T_{9A} + T_{9C} + T_{9E} + T_{A0} + T_{A2} + T_{A3} + T_{A4} + T_{AD} + T_{BE} + T_{D2} + T_{D4} + T_{D6} + T_{D8} + T_{DA} + T_{E3}$
- $brncnd = \overline{brnotSTART} \cdot \overline{START} + brhack \cdot \overline{hack} + brnotfcCPU \cdot \overline{fcCPU} + brnotgropr \cdot \overline{gropr} + brl1 \cdot l1 + brnotgradr \cdot \overline{gradr} + brl2_brnch \cdot l2_brnch + brl2_arlog \cdot l2_arlog + brl3_jump \cdot l3_jump + brl3_arlog \cdot l3_arlog + brstore \cdot \overline{store} + brLDW \cdot \overline{LDW} + brdirreg \cdot \overline{dirreg} + brnotbrpom \cdot \overline{brpom} + brnotprekid \cdot \overline{prekid} + brnotPRINS \cdot \overline{PRINS} + brnotPRCOD \cdot \overline{PRCOD} + brnotPRADR \cdot \overline{PRADR} + brnotPRINM \cdot \overline{PRINM} + brnotprinr \cdot \overline{prinr}$
- $brnotSTART = T_{00}$
- $brhack = T_{02} + T_{09} + T_{11} + T_{16} + T_{2A} + T_{2D} + T_{38} + T_{3D} + T_{5C} + T_{62} + T_{65} + T_{6A} + T_{70} + T_{74} + T_{7B} + T_{80} + T_{84} + T_{A7} + T_{AB} + T_{AF} + T_{B3} + T_{B7} + T_{BB} + T_{C3} + T_{C7} + T_{CB} + T_{CF} + T_{DD} + T_{E0}$
- $brnotfcCPU = T_{03} + T_{0A} + T_{12} + T_{17} + T_{2B} + T_{2E} + T_{39} + T_{3E} + T_{5D} + T_{63} + T_{66} + T_{6B} + T_{71} + T_{75} + T_{7C} + T_{81} + T_{85} + T_{A8} + T_{AC} + T_{B0} + T_{B4} + T_{B8} + T_{BC} + T_{C4} + T_{C8} + T_{CC} + T_{D0} + T_{DE} + T_{E1}$
- $brnotgropr = T_{05}$
- $brl1 = T_{07}$
- $brnotgradr = T_{0C}$
- $brl2_brnch = T_{0E}$
- $brl2_arlog = T_{0F}$
- $brl3_jump = T_{13}$
- $brl3_arlog = T_{14}$
- $brstore = T_{21} + T_{25} + T_{27} + T_{30} + T_{32} + T_{35} + T_{37}$
- $brLDW = T_{22} + T_{3A} + T_{41}$
- $brdirreg = T_{5A} + T_{60}$
- $brnotbrpom = T_{A1}$
- $brnotprekid = T_{C0}$
- $brnotPRINS = T_{D1}$
- $brnotPRCOD = T_{D3}$
- $brnotPRADR = T_{D5}$

- **brnotPRINM** = T_{D7}
- **brnotprintr** = T_{D9}
- **val₀₀** = $T_{00} + T_{52} + T_{C0} + T_{E3}$
- **val₀₂** = T_{02}
- **val₀₃** = T_{03}
- **val₀₇** = T_{05}
- **val₀₉** = T_{09}
- **val_{0A}** = T_{0A}
- **val_{0E}** = T_{0C}
- **val₁₁** = T_{11}
- **val₁₂** = T_{12}
- **val₁₆** = T_{16}
- **val₁₇** = T_{17}
- **val₂₀** = $T_{0F} + T_{14} + T_{18}$
- **val₂₄** = T_{22}
- **val_{2A}** = T_{2A}
- **val_{2B}** = T_{2B}
- **val_{2D}** = T_{2D}
- **val_{2E}** = T_{2E}
- **val₃₈** = $T_{26} + T_{28} + T_{31} + T_{33} + T_{36} + T_{38}$
- **val₃₉** = T_{39}
- **val_{3C}** = T_{3A}
- **val_{3D}** = T_{3D}
- **val_{3E}** = T_{3E}
- **val₄₃** = T_{41}
- **val₅₀** = $T_{07} + T_{0E} + T_{13} + T_{21} + T_{23} + T_{24} + T_{25} + T_{27} + T_{30} + T_{32} + T_{35} + T_{37} + T_{3B} + T_{40} + T_{42} + T_{43}$
- **val_{5C}** = T_{5C}
- **val_{5D}** = T_{5D}
- **val_{5F}** = T_{5A}
- **val₆₂** = T_{62}
- **val₆₃** = T_{63}
- **val₆₅** = T_{65}
- **val₆₆** = T_{66}
- **val₆₈** = T_{60}
- **val_{6A}** = T_{6A}
- **val_{6B}** = T_{6B}
- **val₇₀** = T_{70}
- **val₇₁** = T_{71}
- **val₇₄** = T_{74}
- **val₇₅** = T_{75}
- **val_{7B}** = T_{7B}
- **val_{7C}** = T_{7C}
- **val₈₀** = T_{80}

- $\text{val}_{81} = T_{81}$
- $\text{val}_{84} = T_{84}$
- $\text{val}_{85} = T_{85}$
- $\text{val}_{A7} = T_{A7}$
- $\text{val}_{A8} = T_{A8}$
- $\text{val}_{AB} = T_{AB}$
- $\text{val}_{AC} = T_{AC}$
- $\text{val}_{AF} = T_{AF}$
- $\text{val}_{B0} = T_{B0}$
- $\text{val}_{B3} = T_{B3}$
- $\text{val}_{B4} = T_{B4}$
- $\text{val}_{B7} = T_{B7}$
- $\text{val}_{B8} = T_{B8}$
- $\text{val}_{BB} = T_{BB}$
- $\text{val}_{BC} = T_{BC}$
- $\text{val}_{C0} = T_{06} + T_{0D} + T_{51} + T_{53} + T_{54} + T_{55} + T_{56} + T_{58} + T_{59} + T_{5E} + T_{5F} + T_{67} + T_{68} + T_{6E} + T_{78} + T_{7D} + T_{86} + T_{87} + T_{88} + T_{89} + T_{8A} + T_{8B} + T_{8C} + T_{8E} + T_{90} + T_{92} + T_{94} + T_{96} + T_{98} + T_{9A} + T_{9C} + T_{9E} + T_{A0} + T_{A1} + T_{A2} + T_{A3} + T_{A4} + T_{AD} + T_{BE}$
- $\text{val}_{C3} = T_{C3}$
- $\text{val}_{C4} = T_{C4}$
- $\text{val}_{C7} = T_{C7}$
- $\text{val}_{C8} = T_{C8}$
- $\text{val}_{CB} = T_{CB}$
- $\text{val}_{CC} = T_{CC}$
- $\text{val}_{CF} = T_{CF}$
- $\text{val}_{D0} = T_{D0}$
- $\text{val}_{D3} = T_{D1}$
- $\text{val}_{D5} = T_{D3}$
- $\text{val}_{D7} = T_{D5}$
- $\text{val}_{D9} = T_{D7}$
- $\text{val}_{DB} = T_{D9}$
- $\text{val}_{DC} = T_{D2} + T_{D4} + T_{D6} + T_{D8} + T_{DA}$
- $\text{val}_{DD} = T_{DD}$
- $\text{val}_{DE} = T_{DE}$
- $\text{val}_{E0} = T_{E0}$
- $\text{val}_{E1} = T_{E1}$

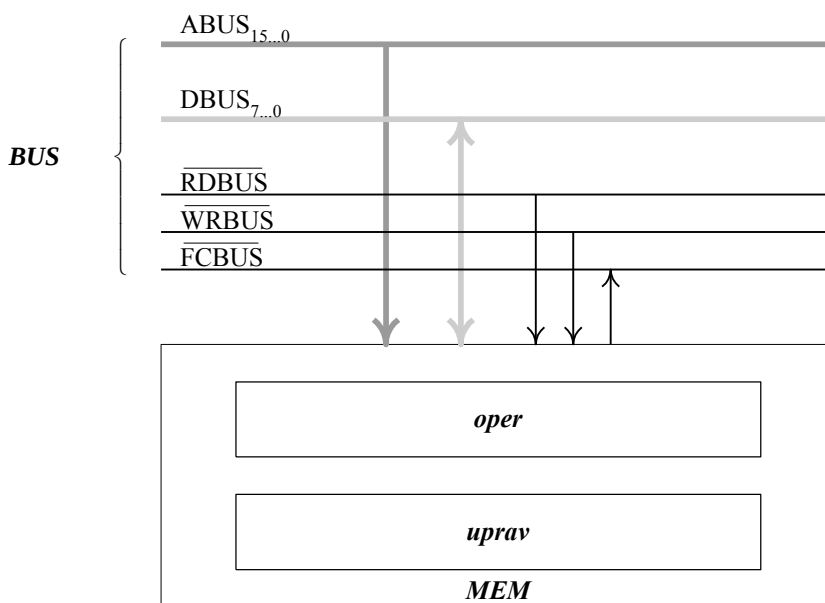
Pri generisanju signala **brcnd** koriste se sledeći signali logičkih uslova koji dolaze iz blokova operacione jedinice **oper** i to:

- | • **START** — blok *exec*,
- **hack** — blok *bus*,
- | • **fcCPU** — blok *bus*,
- **gropr** — blok *fetch*,
- | • **l1** — blok *fetch*,
- **gradr** — blok *fetch*,

- **l2_brnch** — blok *fetch*,
- **l2_arlog** — blok *fetch*,
- **l3_jump** — blok *fetch*,
- **l3_arlog** — blok *fetch*,
- **store** — blok *fetch*,
- **LDW** — blok *fetch*,
- **dirreg** — blok *fetch*,
- **brpom** — blok *exec*,
- **prekid** — blok *intr*
- **PRINS** — blok *intr*,
- **PRCOD** — blok *intr*,
- **PRADR** — blok *intr*,
- **PRINM** — blok *intr* i
- **printr** — blok *intr*.

5 MEMORIJA

U ovoj glavi se daje organizacija memorije **MEM** (slika 59). Memorija **MEM** se sastoji iz operacione jedinice **oper** i upravljačke jedinice **uprav**.



Slika 59 Organizacija memorije **MEM**

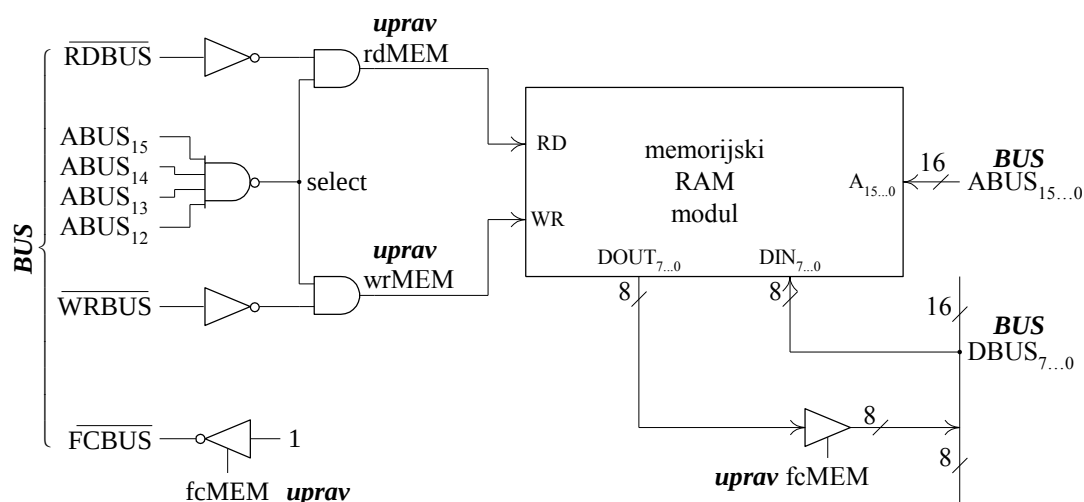
Operaciona jedinica **oper** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za pamćenje binarnih reči, izvršavanje mikrooperacija i generisanje signala logičkih uslova. Upravljačka jedinica **uprav** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe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

5.1 OPERACIONA JEDINICA

Operaciona jedinica **oper** 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 60).

Memorija **MEM** 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 **fcMEM**. Na linije **WR** i **RD** se vode signali **wrMEM** i **rdMEM** koje generišu kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je memorija sluga. Sa memorijom **MEM** mogu se realizovati operacije upisa i čitanja. 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 60 Operaciona jedinica **oper**

Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je memorija sluga formiraju signale **rdMEM** i **wrMEM** upravljačke jedinice **uprav** i **FCBUS** magistrale **BUS** (slika 71). Signali **rdMEM** i **wrMEM** se formiraju na osnovu signala **ABUS_{15..0}** sa adresnih linija magistrale i **RDBUS** i **WRBUS** sa upravljačkih linija magistrale. Signal **FCBUS** se formira na osnovu signala **fcMEM** upravljačke jedinice **uprav**.

Pri realizaciji ciklusa čitanja na magistrali procesor ili kontroler sa direktnim pristupom memoriji kao gazda otvara bafere sa tri stanja za adresne linije **ABUS_{15..0}** i upravljačku liniju **RDBUS** magistrale i na njih izbacuje adresu i aktivnu vrednost signala čitanja, respektivno, čime se u memoriji kao slugi startuje čitanje adresirane memorijske lokacije. Memorija po završenom čitanju otvara bafere sa tri stanja za linije podataka **DBUS_{7..0}** i upravljačku liniju **FCBUS** magistrale i na njih izbacuje sadržaj adresirane memorijske lokacije i aktivnu vrednost signala završetka operacije čitanja u kontroleru. Procesor ili kontroler sa direktnim pristupom memoriji prihvata sadržaj sa linija podataka i zatvara bafere sa tri stanja za adresne linije **ABUS_{15..0}** i upravljačku liniju **RDBUS** magistrale, dok memorija zatvara bafere sa tri stanja za linije podataka **DBUS_{7..0}** i upravljačku liniju **FCBUS** magistrale.

Pri realizaciji ciklusa upisa na magistrali procesor ili kontroler sa direktnim pristupom memoriji kao gazda otvara bafere sa tri stanja za adresne linije **ABUS_{15..0}**, linije podataka **DBUS_{7..0}** i upravljačku liniju **WRBUS** magistrale i na njih izbacuje adresu, podatak i aktivnu vrednost signala upisa, respektivno, čime se u memoriji kao slugi startuje upis u adresiranu memorijsku lokaciju. Memorija po završenom upisu otvara bafere sa tri stanja za i upravljačku liniju **FCBUS** magistrale i na nju izbacuje aktivnu vrednost signala završetka operacije upisa u memoriji. 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 **WRBUS** magistrale, dok memorija zatvara bafere sa tri stanja za upravljačku liniju **FCBUS** magistrale.

Signali **rdMEM** i **wrMEM** imaju vrednosti upravljačkih signala **RDBUS** i **WRBUS** magistrale, respektivno, kada je aktivna vrednost signala **select** i neaktivne vrednosti kada je neaktivna vrednost signala **select**. Signal **rdMEM** je kao i signal **RDBUS** aktivan za vreme operacije čitanja neke memorijske lokacije memorije, dok je signal **wrKTR** kao i signal **WRBUS** aktivan za vreme operacije upisa u neku memorijsku lokaciju memorije.

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 **rdMEM** i **wrMEM** bez obzira na vrednosti signala **RDBUS** i **WRBUS**, respektivno.

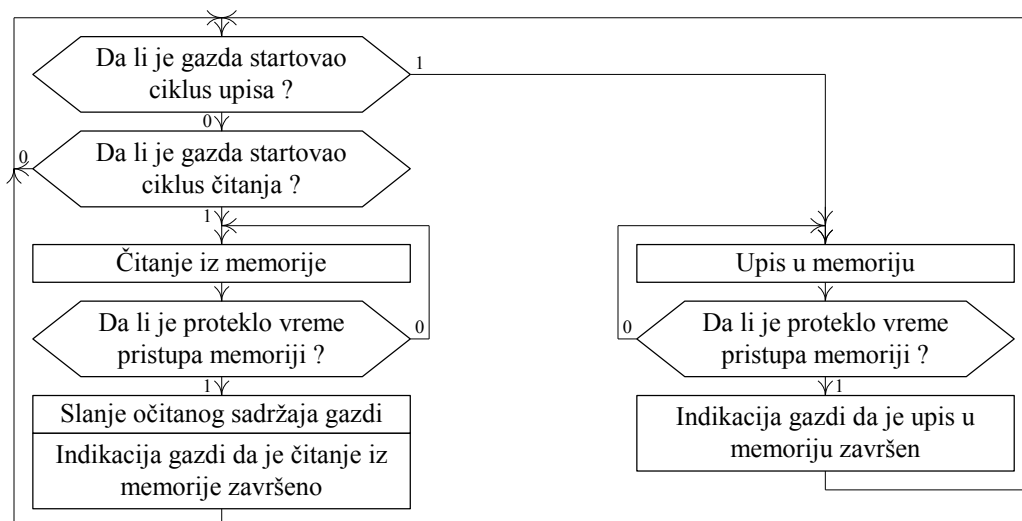
Kombinaciona mreža za generisanje upravljačkog signala **FCBUS** magistrale generiše ovaj signal na osnovu signala **fcMEM**. Pri neaktivnoj vrednosti signala **fcMEM** na liniji signala **FCBUS** je stanje visoke impedance, dok je pri aktivnoj vrednosti signala **fcMEM** na liniji signala **FCBUS** aktivna vrednost. Signal **fcMEM** ima neaktivnu ili aktivnu vrednost u zavisnosti od toga da li je u kontroleru operacija čitanja ili upisa u toku ili je završena, respektivno.

5.2 UPRAVLJAČKA JEDINICA

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

5.2.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 61). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.



Slika 61 Dijagram toka operacija

U dijagrama toka operacija stalno 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. Ako je startovan jedan od ova dva ciklusa izvršavaju se mikrooperacije saglasno ciklusu koji je startovan.

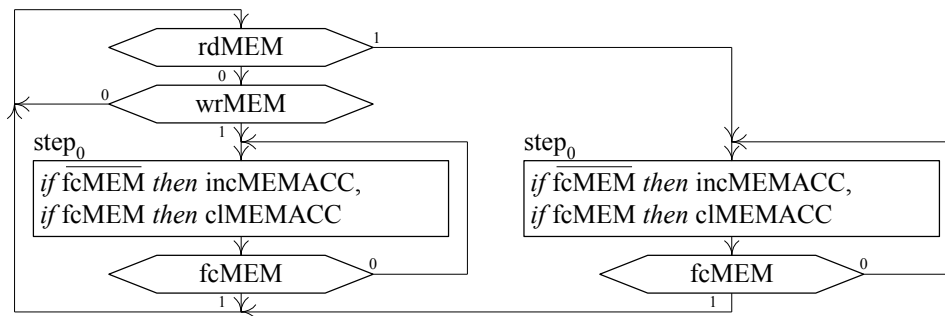
Ako je startovan ciklus čitanja ulazi se u petlju u kojoj se vrši čitanje iz memorije u trajanju vremena pristupa memoriji. Po isteku vremena pristupa memoriji izlaz se iz petlje i gazdi šalje očitani sadržaj i indikacija da je memorija završila čitanje.

Ako je startovan ciklus upisa ulazi se u petlju u kojoj se vrši upis u memoriju u trajanju vremena pristupa memoriji. Po isteku vremena pristupa memoriji izlaz se iz petlje i gazdi šalje indikacija da je memorija završila upis.

5.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 61) i dat u obliku dijagrama toka mikrooperacija, dijagrama toka upravljačkih signala (slika 62) i sekvence upravljačkih signala (tabela 24).

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.



Slika 62 Dijagram toka upravljačkih signala

U sekvenci upravljačkih signala se koriste iskazi za signale. Iskazi za signale su oblika **if uslov then signali**.

Ovi iskazi sadrže uslov i spisak upravljačkih signala blokova operacione jedinice **oper** i određuje koji signali i pod kojim uslovima treba da budu generisani.

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 24 Sekvenca upravljačkih signala

! U koraku $step_0$ se ostaje sve vreme. U ovom koraku se generiše aktivna vrednost signala **fcMEM** operacione jedinice **oper** trajanja jedna perioda signala takta po isteku vremena pristupa memorije i to samo kada je u memoriji startovano čitanje ili upis. U ovom koraku se ne generiše aktivna vrednost signala **fcMEM** sve dok su oba signala **rdMEM** i **wrMEM** memorije neaktivni. Signal **rdMEM** postaje aktivan kada procesor ili kontroler sa direktnim pristupom memoriji postavi na aktivnu vrednost upravljački signal čitanja **RDBUS** čime u započinje čitanje iz memorije. Signal **rdMEM** ostaje aktivan sve dok se čitanje ne završi i pročitani podatak prebaci iz memorije u procesor ili kontroler sa direktnim pristupom memoriji. Aktivna vrednost signala **rdMEM** startuje čitanje iz memorije MEM operacione jedinice. Vreme neophodno da se čitanje završi i da podatak postane raspoloživ na izlaznim linijama podatak $DOUT_{7...0}$ memorije MEM određeno je sadržajem mikroprekidača $TIME_{7...0}$. Zbog toga se pri aktivnoj vrednosti signala **rdMEM** inkrementira brojač vremena pristupa $MEMACC_{7...0}$. Kada se inkrementiranjem sadržaj brojača $MEMACC_{7...0}$ izjednači

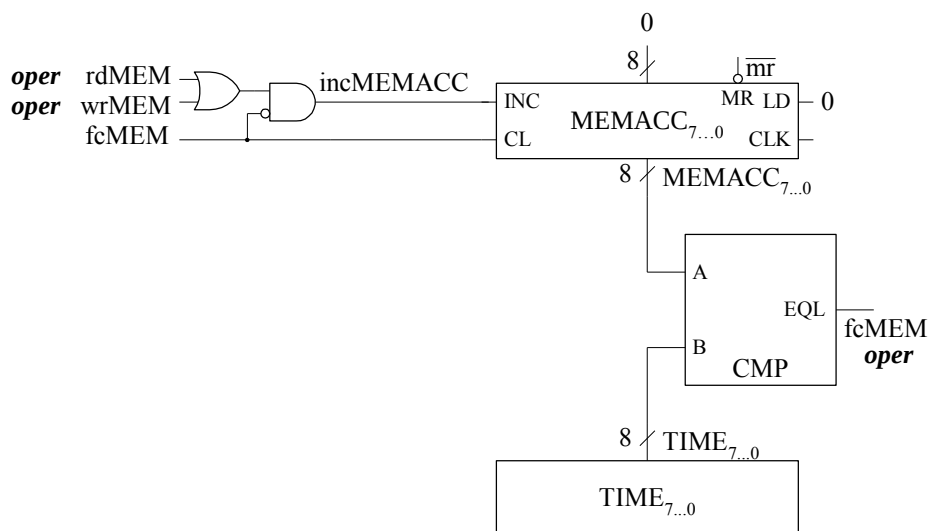
sa sadržajem mikroprekidača $TIME_{7...0}$, signal **fcMEM** na izlazu komparatora CMP postaje aktivan. Pri aktivnoj vrednosti signala **fcMEM** koja daje aktivnu vrednost signala $\overline{FCBUS}$ magistrale **BUS**, procesor ili kontroler sa direktnim pristupom memoriji prihvata pročitani podatak i ukida aktivnu vrednost signala $\overline{RDBUS}$ magistrale što dovodi do ukidanja aktivne vrednosti signala **rdMEM**. Pored toga, pri aktivnoj vrednosti signala **fcMEM** u brojač $MEMACC_{7...0}$ se upisuju sve nule, pa signal **fcMEM** na izlazu komparatora postaje neaktivan. Slična je situacija i prilikom realizaciju upisa u memoriju, pri čemu se sada javlja aktivna vrednost signala **wrMEM** kao rezultat toga što je procesor ili kontroler sa direktnim pristupom memoriji postavio na aktivnu vrednost upravljački signal upisa $\overline{WRBUS}$ magistrale čime započinje upis u memoriju.

step₀ if ($MEMACC_{7...0}$ eql $TIME_{7...0}$) then **fcMEM**

5.2.3 Struktura upravljačke jedinice

Struktura upravljačke jedinice je prikazana na slici 63. Upravljačka jedinica se sastoji od mikroprekidača $TIME_{7...0}$, brojača $MEMACC_{7...0}$, komparatora CMP i logičkih elemenata koji na osnovu signala čitanja **rdMEM** i signala upis **wrMEM** memorije generišu signal **fcMEM** operacione jedinice **oper**.

Mikroprekidači $TIME_{7...0}$ se prilikom konfiguracije sistema ručno postavljaju na vrednost koja pomnožena sa periodom signala takta definiše vreme pristupa memorije. Brojač $MEMACC_{7...0}$ se koristi kao brojač vremena pristupa memorije. Njegov sadržaj je normalno sve nule. Međutim, prilikom startovanja memorije radi čitanja ili upisa, kada se generiše aktivna vrednost signala **rdMEM** ili **wrMEM**, njegov sadržaj se inkrementira i to onoliko puta koliki je sadržaj mikroprekidača $TIME_{7...0}$. Komparator CMP upoređuje sadržaje mikroprekidača $TIME_{7...0}$ i brojača $MEMACC_{7...0}$ i na izlazu EQL daje neaktivnu vrednost signala **fcMEM** dok se ovi sadržaji razlikuju i aktivnu vrednost signala **fcMEM** kada ovi sadržaji postanu jednaki. Aktivna vrednost signala **fcMEM**, koja daje aktivnu vrednost signala $\overline{FCBUS}$ magistrale, je indikacija procesoru ili kontroleru sa direktnom pristupom memoriji da je završeno čitanje iz memorije ili upis u memoriju. Pored toga, aktivna vrednost signala **fcPER** se koristi u upravljačkoj jedinici da se sadržaj brojača $MEMACC_{7...0}$ vrati na početno stanje sve nule.



Slika 63 Struktura upravljačke jedinice

Signali **rdMEM** i **wrMEM** imaju aktivne vrednosti trajanja onoliko perioda signala takta koliko je vreme pristupa memorije određeno sadržajem mikroprekidača $TIME_{7...0}$. To je posledica usvojenog načina generisanja signala **rdCPU** i **wrCPU** procesora **CPU** i signala **rdMKTR** i **wrMKTR** kontrolera sa direktnim pristupom memoriji na osnovu kojih se formiraju signali $\overline{RDBUS}$ i $\overline{WRBUS}$ magistrale **BUS** i signali **rdMEM** i **wrMEM** memorije, i signala **fcMEM** memorije, na osnovu koga se generišu signali $\overline{FCBUS}$ magistrale **BUS** i **fcCPU** procesora **CPU** i **fcMKTR** kontrolera sa direktnim pristupom memoriji.

Kod čitanja iz memorije procesor na i -ti signal takta ulazi u stanje $step_i$ koje se koristi da se generiše aktivna vrednost signala **rdCPU** koja daje aktivne vrednosti signala $\overline{RDBUS}$ i **rdMEM**. U memoriji zbog aktivne vrednosti signala **rdMEM** počinje inkrementiranje brojača vremena pristupa memorije $MEMACC_{7...0}$. Signal **fcMEM** na izlazu EQL komparatora CMP je neaktivan sve vreme dok sadržaj brojača $MEMACC_{7...0}$ inkrementiranjem ne postane jednak sadržaju mikroprekidača $TIME_{7...0}$ što daje neaktivne vrednosti signala $\overline{FCBUS}$ i **fcCPU**. Kako je u procesoru aktivna vrednost signala **fcCPU** uslov za prelazak iz stanja $step_i$ u stanje $step_{i+1}$, to procesor ostaju u ovom stanju j perioda signala takta koliko je neophodno da sadržaj brojača $MEMACC_{7...0}$ inkrementiranjem dostigne sadržaj mikroprekidača $TIME_{7...0}$ i da se na $(i+j)$ -ti signal takta pojavi aktivna vrednost signala **fcMEM** i time formiraju aktivne vrednosti signala $\overline{FCBUS}$ i **fcCPU**. To omogućuje procesoru da na prvi sledeći signal takta prelazi iz koraka $step_i$ na korak $step_{i+1}$. Prelaskom kontrolera na korak $step_{i+1}$ signal **rdCPU** postaje neaktivan, čime postaju neaktivni i signali $\overline{RDBUS}$, **rdMEM**, **fcMEM**, $\overline{FCBUS}$ i **fcCPU**. Na isti način se razmenjuju i signali **wrCPU**, $\overline{WRBUS}$, **wrMEM**, **fcMEM**, $\overline{FCBUS}$ i **fcCPU** kod upisa u memoriju. Ovo važi i za signale **rdMKTR**, $\overline{RDBUS}$, **rdMEM**, **fcMEM**, $\overline{FCBUS}$ i **fcMKTR** kada kontroler sa direktnim pristupom memoriji čita iz memorije i signale **wrMKTR**, $\overline{WRBUS}$, **wrMEM**, **fcMEM**, $\overline{FCBUS}$ i **fcMKTR** kada kontroler sa direktnim pristupom memoriji upisuje u memoriju.

Upravljački signal **fcMEM** koji se koristi u operacionoj jedinici **oper** se generiše na sledeći način:

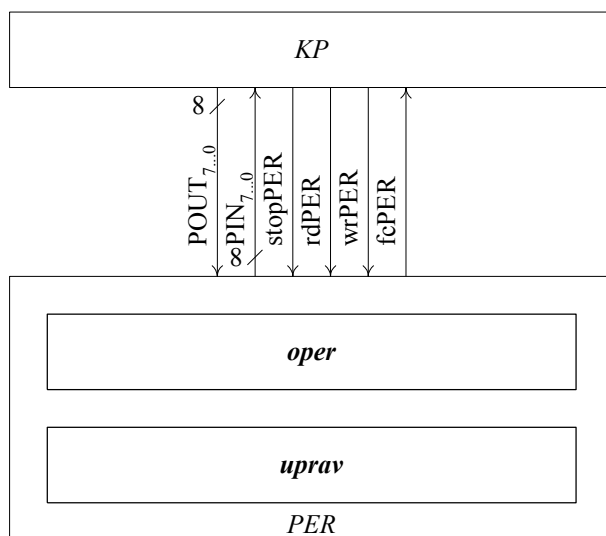
- **fcMEM** = $MEMACC_{7...0}$ eq $TIME_{7...0}$

6 ULAZNO/IZLAZNI UREĐAJI

Ulazno/izlazni uređaji *U/I* se sastoje iz periferije *PER* i kontrolera periferije *KP*. Postoje kontroleri periferija bez i sa direktnim pristupom memoriji. U ovoj glavi se daje organizacija periferije, kontrolera bez direktnog pristupa memoriji i kontrolera sa direktnim pristupom memoriji.

6.1 PERIFERIJA

Periferija *PER* (slika 64) se sastoji iz operacione jedinice *oper* i upravljačke jedinice *uprav*.



Slika 64 Organizacija periferije *PER*

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 na osnovu algoritma generisanja upravljačkih signala operacione jedinice i vrednosti signala logičkih uslova.

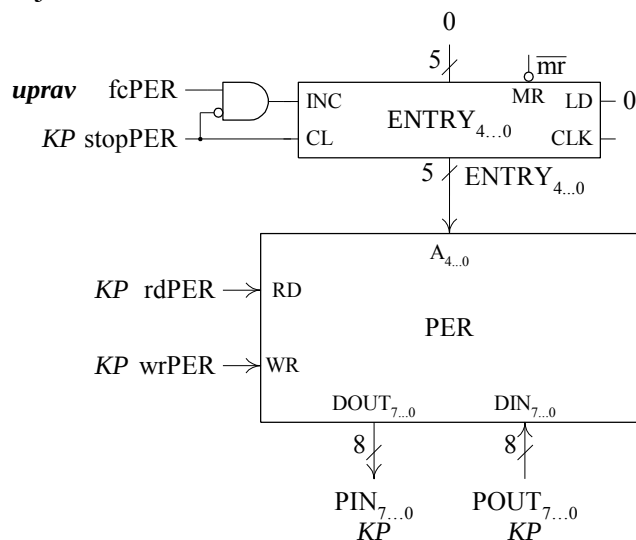
Struktura i opis operacione i upravljačke jedinice se daju u daljem tekstu.

6.1.1 Operaciona jedinica

Operaciona jedinica *oper* sadrži brojač *ENTRY*_{4...0} i memoriju *PER* (slika 65).

Brojač *ENTRY*_{4...0} je 5-to razredni brojač čiji se sadržaj koristi za adresiranje memorijskih lokacija memorije *PER* prilikom upisa u periferiju ili čitanja iz periferije. U zavisnosti od vrednosti signala na ulazima *CL* i *INC* brojača, u brojač se mogu upisati sve nule, sadržaj brojača se može inkrementirati i sadržaj brojača može ostati nepromenjen. Ova tri režima rada brojača se realizuju generisanjem aktivne i neaktivne vrednosti na ulazima *CL* i *INC* onda kada treba upisati sve nule, neaktivne i aktivne vrednosti na ulazima *CL* i *INC* onda kada sadržaj brojača treba inkrementirati i neaktivnih vrednosti na ulazima *CL* i *INC* onda kada sadržaj brojača treba da ostane nepromenjen. Pri aktivnoj vrednosti signala *stopPER* kontrolera *KP* na ulazima *CL* i *INC* će biti aktivna i neaktivna vrednost bez obzira na vrednost signala *fcPER*, pa se tada u brojač upisuju sve nule. Ovo se koristi da se pri zaustavljanju periferije *PER* od strane kontrolera sadržaj brojača *ENTRY*_{4...0} dovede na početno

stanje sve nule. Pri neaktivnoj vrednosti signala **stopPER** na ulazu CL će biti neaktivna vrednost, dok će na ulazu INC biti neaktivna ili aktivna vrednost u zavisnosti od toga da li je vrednost signala **fcPER** neaktivna ili aktivna. Ovo se koristi da se prilikom svakog upisa u memoriju PER periferije ili čitanja iz memoriju PER periferije generisanjem aktivne vrednosti signala **fcPER** sadržaj brojača inkrementira da bi ukazivao na sledeću memorijsku lokaciju memorije PER u koju treba upisati sledeći podatak ili iz koje treba pričitati sledeći podatak. U svim ostalim situacijama pored signal **stopPER** i signal **fcPER** je neaktivan, pa sadržaj brojača ostaje nepromenjen.



Slika 65 Operaciona jedinica **oper**

Memorija PER služe za simulaciji periferije kapaciteta 2^5 8-mo bitnih reči. Sadržaj $POUT_{7...0}$ sa izlaznih linija podataka kontrolera periferije KP se vodi na ulazne linije podataka $DIN_{7...0}$ memorije PER. Ovaj sadržaj se upisuje u memorijsku lokaciju adresiranu sadržajem registra $ENTRY_{4...0}$ jedino ukoliko signal **wrPER** kontrolera KP ima aktivnu vrednost. Sadržaj memorijske lokacije adresirane sadržajem brojača $ENTRY_{4...0}$ se pojavljuje na izlaznim linijama podataka $DOUT_{7...0}$ memorije PER jedino ukoliko signal **rdPER** kontrolera KP ima aktivnu vrednost. Ovaj sadržaj se vodi na ulazne linije podataka $PIN_{7...0}$ kontrolera periferije KP.

6.1.2 Upravljačka jedinica

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

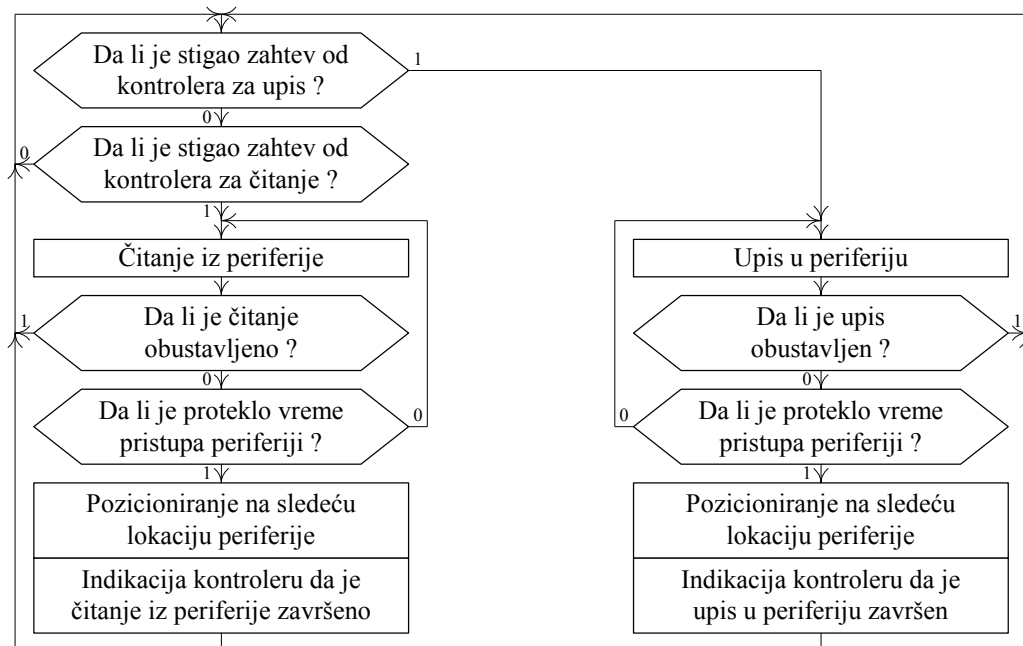
6.1.2.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 66). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

U dijagrama toka operacija vrši se provera da li je od kontrolera stigao zahtev za operaciju upisa ili čitanja. Ako nije stigao nijedan od ovih zahteva, u periferiji nema ni čitanja ni upisa. Ako je stigao neko od zahteva prelazi se na realizaciju čitanja ili upisa.

Ako je stigao zahtev za operaciju čitanja, ulazi se u petlju u kojoj se čita podatak iz periferije i vrši provera da li je programskim putem, upisivanjem neaktivne vrednosti u bit start upravljačkog registra kontrolera, obustavljeno čitanje i da li je proteklo vreme pristupa periferiji. Ako je čitanje obustavljeno, izlazi se iz petlje i vraća na proveru da li je od

kontrolera stigao novi zahtev za operaciju upisa ili čitanja. Ako je proteklo vreme pristupa periferije, uzima se da je čitanje realizovano, pa se izlazi iz petlje i prelazi na pozicioniranje na sledeću lokaciju periferije, slanje indikacije kontroleru da je čitanje završeno i vraćanje na proveru da li je od kontrolera stigao novi zahtev za operaciju upisa ili čitanja.

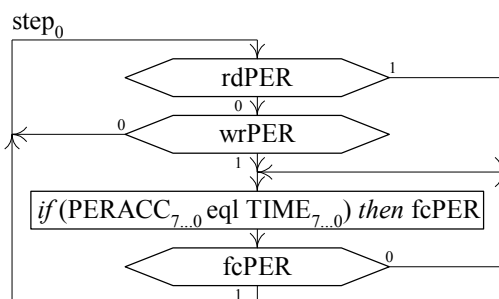


Slika 66 Dijagram toka operacija

Ako je stigao zahtev za operaciju upisa, ulazi se u petlju u kojoj se upisuje podatak u periferije i vrši proveru da li je programskim putem, upisivanjem neaktivne vrednosti u bit start upravljačkog registra kontrolera, obustavljen upis i da li je proteklo vreme pristupa periferiji. Ako je upis obustavljen, izlazi se iz petlje i vraća na proveru da li je od kontrolera stigao novi zahtev za operaciju upisa ili čitanja. Ako je proteklo vreme pristupa periferije, uzima se da je upis realizovan, pa se izlazi iz petlje i prelazi na pozicioniranje na sledeću lokaciju periferije, slanje indikacije kontroleru da je upis završen i vraćanje na proveru da li je od kontrolera stigao novi zahtev za operaciju upisa ili čitanja.

6.1.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 66) i dat u obliku dijagrama toka upravljačkih signala (slika 67) i sekvence upravljačkih signala (tabela 25).



Slika 67 Dijagram toka upravljačkih signala

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 67). U operacionom bloku se nalazi upravljački signal i uslov pod kojima se on generiše. U uslovnim blokovima se nalaze signali logičkih uslova.

U sekvenci upravljačkih signala po koracima se koristi iskaz za signal. Iskaz za signal je oblika

if uslov then signal.

Ovaj iskaz sadrži uslov i upravljački signal kontrolera periferije *KP* i operacione jedinice *oper* i određuje pod kojim uslovima signal treba da bude generisan.

Objašnjenja vezana za generisanje upravljačkih signala su data zajednički za dijagram toka upravljačkih signala (slika 67) i sekvencu upravljačkih signala (tabela 25) i to u okviru sekvence upravljačkih signala.

Tabela 25 Sekvenca upravljačkih signala

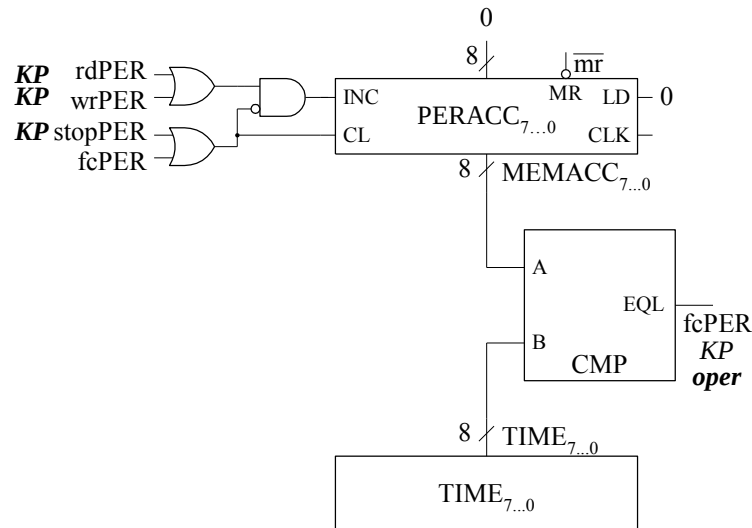
! U koraku $step_0$ se ostaje sve vreme. U ovom koraku se generiše aktivna vrednost signala **fcPER** kontrolera *KP* i operacione jedinice *oper* trajanja jedna perioda signala takta po isteku vremena pristupa periferije i to samo kada je u periferiji startovano čitanje ili upis. U ovom koraku se ne generiše aktivna vrednost signala **fcPER** sve dok su oba signala **rdPER** i **wrPER** kontrolera neaktivni. Signal **rdPER** postaje aktivan kada kontroler uđe u stanje u kome treba da realizuje čitanje iz periferije i ostaje aktivan sve dok se čitanje ne završi i pročitani podatak prebaci iz periferije u kontroler. Aktivna vrednost ovog signala startuje čitanje iz memorije PER operacione jedinice. Vreme neophodno da se čitanje završi i da podatak postane raspoloživ na izlaznim linijama podatak $DOUT_{7...0}$ memorije PER određeno je sadržajem mikroprekidača $TIME_{4...0}$. Zbog toga se pri aktivnoj vrednosti signala **rdPER** inkrementira brojač vremena pristupa $PERACC_{4...0}$. Kada se inkrementiranjem sadržaj brojača $PERACC_{4...0}$ izjednači sa sadržajem mikroprekidača $TIME_{4...0}$, signal **fcPER** na izlazu komparatora CMP operacione jedinice postaje aktivan. Pri aktivnoj vrednosti signala **fcPER** kontroler prihvata pročitani podatak i ukida aktivnu vrednost signala **rdPER**. Pored toga, pri aktivnoj vrednosti signala **fcPER** u brojač $PERACC_{4...0}$ se upisuju sve nule, pa signal **fcPER** na izlazu komparatora postaje neaktivan. Slična je situacija i prilikom realizaciju upisa u periferiju, pri čemu se sada generiše aktivna vrednost signala **wrPER** i njome u memoriji PER operacione jedinice startuje upis.

$step_0$ *if* ($PERACC_{7...0}$ eq $TIME_{7...0}$) *then* **fcPER**

6.1.2.3 Struktura upravljačke jedinice

Struktura upravljačke jedinice je prikazana na slici 68. Upravljačka jedinica se sastoji od mikroprekidača $TIME_{7...0}$, brojača $PERACC_{7...0}$, komparatora CMP i logičkih elemenata koji na osnovu signala čitanja **rdPER** i signala upis **wrPER** kontrolera *KP* generišu signal **fcPER** kontrolera *KP* i operacione jedinice *oper*.

Mikroprekidači $TIME_{7...0}$ se prilikom konfiguracije sistema ručno postavljaju na vrednost koja pomnožena sa periodom signala takta definiše vreme pristupa periferije. Brojač $PERACC_{7...0}$ se koristi kao brojač vremena pristupa periferije. Njegov sadržaj je normalno sve nule. Međutim, prilikom startovanja periferije radi čitanja ili upisa, kada se generiše aktivna vrednost signala **rdPER** ili **wrPER**, njegov sadržaj se inkrementira i to onoliko puta koliki je sadržaj mikroprekidača $TIME_{7...0}$. Komparator CMP upoređuje sadržaje mikroprekidača $TIME_{7...0}$ i brojača $PERACC_{7...0}$ i na izlazu EQL daje neaktivnu vrednost signala **fcPER** dok se ovi sadržaji razlikuju i aktivnu vrednost signala **fcPER** kada ovi sadržaji postanu jednaki. Aktivna vrednost signala **fcPER** je indikacija kontroleru *KP* da je završeno čitanje iz periferije ili upis u periferiju. Pored toga, aktivna vrednost signala **fcPER** se koristi u upravljačkoj jedinici da se sadržaj brojača $PERACC_{7...0}$ vrati na početno stanje sve nule i u operacionoj jedinici da se sadržaj brojača $ENTRY_{4...0}$ inkrementira. Prilikom zaustavljanja kontrolera programskim putem, iz kontrolera stiže u periferiju aktivna vrednost signala **stopPER** koja se koristi da se u brojač $PERACC_{7...0}$ upišu sve nule.



Slika 68 Struktura upravljačke jedinice **uprav**

Signalni **rdPER** i **wrPER** imaju aktivne vrednosti trajanja onoliko perioda signala takta koliko je vreme pristupa periferije određeno sadržajem mikroprekidača $TIME_{7...0}$. To je posledica usvojenog načina generisanja signala **rdPER** i **wrPER** kontrolera i signala **fcPER** periferije. Kod čitanja iz periferije kontroler na i -ti signal takta ulazi u stanje $step_i$ koje se koristi da se generiše aktivna vrednost signala **rdPER**. U periferiji zbog aktivne vrednosti signala **rdPER** počinje inkrementiranje brojača vremena pristupa periferije $PERACC_{7...0}$. Signal **fcPER** na izlazu EQL komparatora CMP je neaktivan sve vreme dok sadržaj brojača $PERACC_{7...0}$ inkrementiranjem ne postane jednak sadržaju mikroprekidača $TIME_{7...0}$. Kako je u kontroleru aktivna vrednost signala **fcPER** uslov za prelazak iz stanja $step_i$ u stanje $step_{i+1}$, to kontroler ostaju u ovom stanju j perioda signala takta koliko je neophodno da sadržaj brojača $PERACC_{7...0}$ inkrementiranjem dostigne sadržaj mikroprekidača $TIME_{7...0}$ i da se na $(i+j)$ -ti signal takta pojavi aktivna vrednost signala **fcPER**. To omogućuje kontroleru da na prvi sledeći signal takta prelazi iz koraka $step_i$ na korak $step_{i+1}$. Prelaskom kontrolera na korak $step_{i+1}$ signal **rdPER** postaje neaktivan. Na isti način se razmenjuju i signali **wrPER** i **fcPER** kod upisa u periferiju.

Upravljački signal **fcPER** koji se koristi u kontroleru **KP** i operacionoj jedinici **oper** se generiše na sledeći način:

- $fcPER = PERACC_{7...0} \text{ eql } TIME_{7...0}$

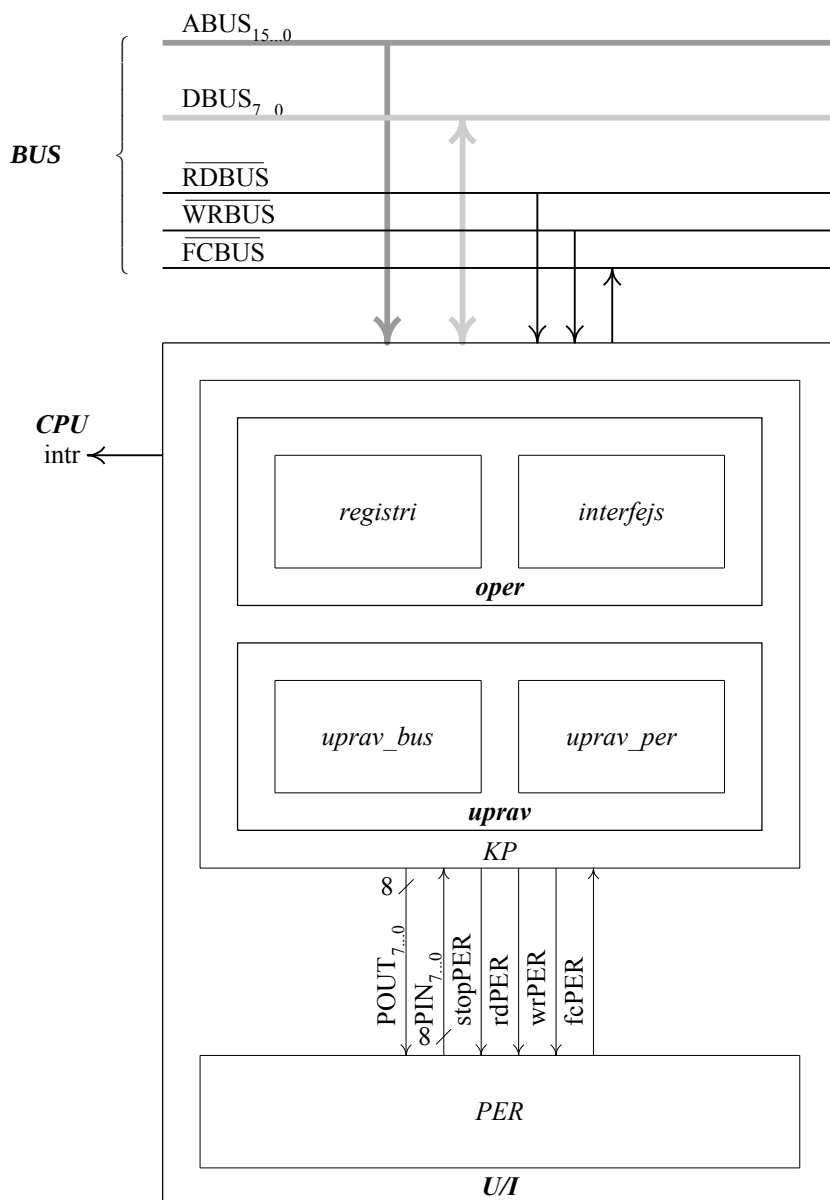
6.2 KONROLER PERIFERIJE BEZ DIREKTOG PRISTUPA MEMORIJI

Kontroler periferije **KP** (slika 69) se sastoji iz operacione jedinice **oper** i upravljačke jedinice **uprav**.

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** se sastoji iz dva dela i to upravljačke jedinice magistrale **uprav_bus** i upravljačke jedinice periferije **uprav_per**. Upravljačka jedinica magistrale **uprav_bus** generiše upravljačke signale neophodne da kontroler periferije **KP** kao sluga realizacije cikluse na magistrali **BUS** kojima se prenose podaci između procesora **CPU** i kontrolera periferije **KP**. Upravljačka jedinica periferije **uprav_per** generiše upravljačke

signale neophodne za prenos podataka između periferije *PER* i kontrolera periferije *KP*. Upravljačke jedinice *uprav_bus* i *uprav_per* rade istovremeno i omogućuju paralelan rad kontrolera periferije *KP* kao sluge sa magistralom **BUS** i kontrolera periferije *KP* sa periferijom *PER*. Svaka od upravljačkih jedinica *uprav_bus* i *uprav_per* je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za generisanje upravljačkih signala prema algoritmu generisanja upravljačkih signala operacione jedinice *oper* i signala logičkih uslova.



Slika 69 Organizacija kontrolera periferije *KP*

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

6.2.1 Operaciona jedinica

Operaciona jedinica (slika 69) se sastoji od sledećih blokova:

- blok *registri* i
- blok *interfejs*.

Blok *registri* (slike 70) služi za čuvanje podataka, upravljačkih i statusnih informacija. Blok *interfejs* (slike 71) služi za realizaciju ciklusa na magistrali u kojima je kontroler sluga i za realizaciju prekida.

Struktura i opis blokova operacione jedinice **oper** se daju u daljem tekstu.

6.2.1.1 Blok registri

Blok *registri* (slika 70) sadrži registre DR_{7...0} i DRAUX_{7...0} sa multiplekserima MP1 i MP2, registar CR_{2...0} i SR₀ i flip-flopove WRDR i RDDR.

Registri DR_{7...0} i DRAUX_{7...0} su 8-mo razredni registar podatka i pomoćni registar podatka, respektivno (slika 70). Ovi registri zajedno sa multiplekserima MP1 i MP2 služe za prenos podataka između periferije *PER* i memorije *MEM*.

Pri prenosu podataka iz periferije u memoriju podatak koji dolazi iz periferije po linijama PIN_{7...0} se, najpre, propušta kroz multiplekser MP2 i upisuje u pomoćni registar podatka DRAUX_{7...0}. Potom se podatak iz registra DRAUX_{7...0} propušta kroz multiplekser MP1 i upisuje u registar podatka DR_{7...0}. Na kraju se podatak iz registra DR_{7...0} preko bafera sa tri stanja propušta na linije podataka DBUS_{7...0} magistrale **BUS** i upisuje u memoriju. Ovakvim načinom prenosa podataka je omogućeno da se istovremeno prenosi tekući podatak iz registra DR_{7...0} u memoriju i sledeći podatak iz periferije u registar DRAUX_{7...0}. Pri tome se sledeći podatak prenosi iz registra DRAUX_{7...0} u registar DR_{7...0}, tek pošto je tekući podatak prenet iz registra DR_{7...0} u memoriju.

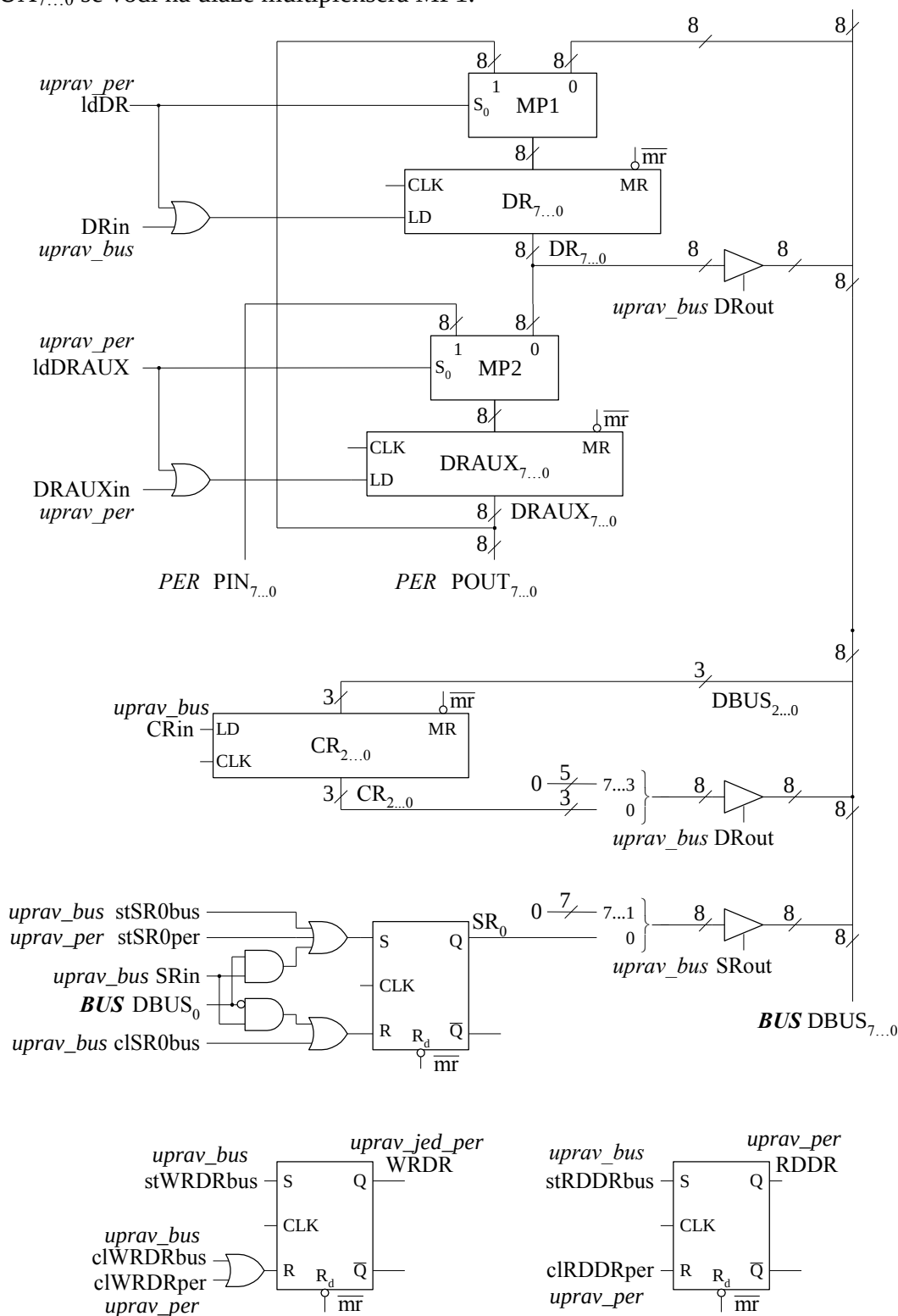
Pri prenosu podataka iz memorije u periferiju podatak koji dolazi iz memorije po linijama podataka DBUS_{7...0} magistrale se, najpre, propušta kroz multiplekser MP1 i upisuje u registar podatka DR_{7...0}. Potom se podatak iz registra DR_{7...0} propušta kroz multiplekser MP2 i upisuje u pomoćni registar podatka DRAUX_{7...0}. Na kraju se podatak iz registra DRAUX_{7...0} po linijama POUT_{7...0} šalje i upisuje u periferiju. Ovakvim načinom prenosa podataka je omogućeno da se istovremeno prenosi tekući podatak iz registra DRAUX_{7...0} u periferiju i sledeći podatak iz memorije u registar DR_{7...0}. Pri tome se sledeći podatak prenosi iz registra DR_{7...0} u registar DRAUX_{7...0}, tek pošto je tekući podatak prenet iz registra DRAUX_{7...0} u periferiju.

Registar DR_{7...0} je 8-mo razredni registar u koji se generisanjem aktivne vrednosti jednog od signala **DRin** i **ldDDR** upisuje sadržaj sa izlaza multipleksa MP1. Pri aktivnoj vrednosti signala **DRin** i neaktivnoj vrednosti signala **ldDDR** sadržaj sa linija podataka DBUS_{7...0} magistrale se propušta kroz multiplekser MP1 i upisuje u registar DR_{7...0}. Pri aktivnoj vrednosti signala **ldDDR** sadržaj sa izlaza registra DRAUX_{7...0} se propušta kroz multiplekser MP1 i upisuje u registar DR_{7...0}. Sadržaj registra DR_{7...0} se pri aktivnoj vrednosti signala **DRout** propušta kroz bafere sa tri stanja na linije podataka DBUS_{7...0} magistrale. Pored toga sadržaj registra DR_{7...0} se vodi na ulaze multipleksa MP2.

Multiplekser MP1 se sastoji od 8 multipleksa sa dva ulaza. Na ulaze 0 i 1 multipleksa MP1 dovodi se sadržaj sa linija podataka DBUS_{7...0} magistrale i sadržaj sa izlaza registra DRAUX_{7...0}, respektivno. Selektovanje sadržaja se obavlja signalom **ldDDR**.

Registar DRAUX_{7...0} je 8-mo razredni registar u koji se, generisanjem aktivne vrednosti jednog od signala **DRAUXin** i **ldDRAUX**, upisuje sadržaj sa izlaza multipleksa MP2. Pri aktivnoj vrednosti signala **DRAUXin** i neaktivnoj vrednosti signala **ldDRAUX** sadržaj registra DR_{7...0} se propušta kroz multiplekser MP2 i upisuje u registar DRAUX_{7...0}. Pri aktivnoj vrednosti signala **ldDRAUX** sadržaj sa ulaznih linija podataka PIN_{7...0} periferije se propušta kroz multiplekser MP2 i upisuje u registar DRAUX_{7...0}. Sadržaj registra DRAUX_{7...0}

se po izlaznim linijama podataka POUT_{7...0} šalje u periferiju. Pored toga sadržaj registra DRAUX_{7...0} se vodi na ulaze multipleksera MP1.



Slika 70 Blok registri

Multiplekser MP2 se sastoji od 8 multipleksera sa dva ulaza. Na ulaze 0 i 1 multipleksera MP2 dovodi se sadržaj sa izlaza registra $DR_{7...0}$ i sadržaj sa ulaznih linija podataka $PIN_{7...0}$ periferije, respektivno. Selektovanje sadržaja se obavlja signalom **ldDRAUX**.

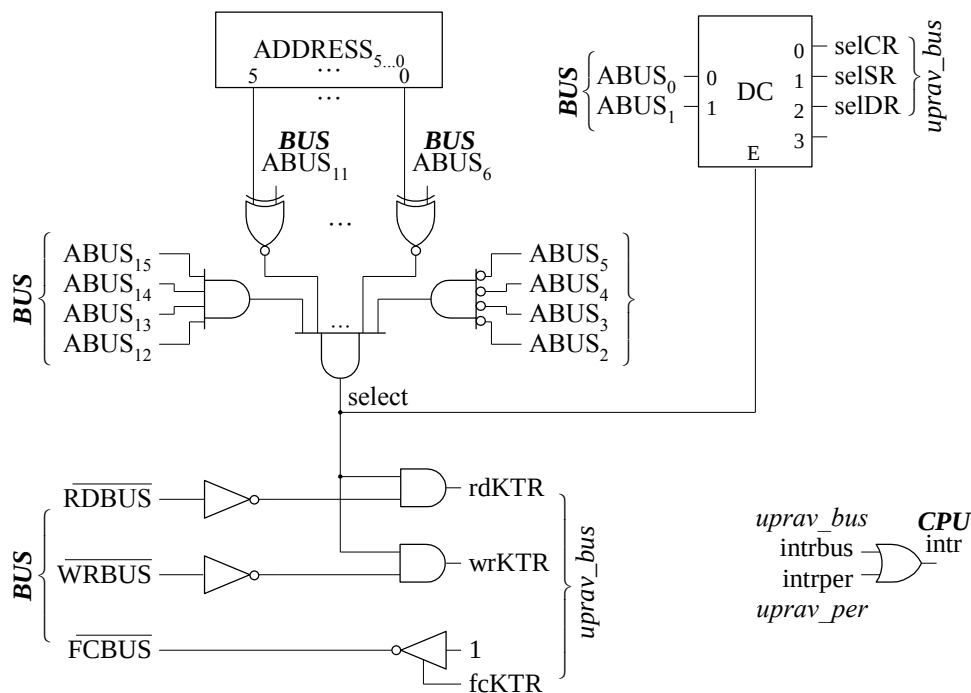
Registar $CR_{2...0}$ je trirazredni upravljački registar koji sadrži bitove start, u/i i prekid, respektivno. U registar $CR_{2...0}$ se, generisanjem aktivne vrednosti signala **CRin** upisuje sadržaj sa linija podataka $DBUS_{2...0}$ magistrale. Sadržaj registra $CR_{2...0}$ proširen nulama do dužine 8 bitova se pri aktivnoj vrednosti signala **CRout** propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale.

Registar SR_0 je jednorazredni upravljački registar koji sadrži bit spremnost. U registar SR_0 se, generisanjem aktivne vrednosti signala **SRin**, upisuje sadržaj sa linija podataka $DBUS_0$ magistrale. Sadržaj registra SR_0 proširen nulama do dužine 8 bitova se pri aktivnoj vrednosti signala **SRout** propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale. Pri prenosu podataka iz periferije u memoriju njegova aktivna vrednost je indikacija da je novi podatak prebačen iz registra $DRAUX_{7...0}$ u registar $DR_{7...0}$ i da je raspoloživ da se programskim putem prebaci iz registra $DR_{7...0}$ u memoriju. S toga se na početku, pri startovanju kontrolera periferije za prenos iz periferije u memoriju, flip-flop registra SR_0 signalom **clSRbus** postavlja na neaktivnu vrednost. Posle toga se, po prenosu podatka iz registra $DRAUX_{7...0}$ u registar $DR_{7...0}$, flip-flop SR_0 signalom **stSR0per** postavlja na aktivnu vrednost, a po prenosu podatka iz registra $DR_{7...0}$ u memoriju signalom **clSR0bus** postavlja na neaktivnu vrednost. Pri prenosu podataka iz memorije u periferiju njegova aktivna vrednost je indikacija da je sadržaj registra $DR_{7...0}$ prebačen u registar $DRAUX_{7...0}$ i da je registar $DR_{7...0}$ raspoloživ da se u njega programskim putem prebaci novi podatak iz memorije. S toga se na početku, pri startovanju kontrolera periferije za prenos iz memorije u periferiju, flip-flop SR_0 signalom **stSR0bus** postavlja na aktivnu vrednost. Posle toga se, pri prenosu podatka iz memorije u registar $DR_{7...0}$, flip-flop SR_0 signalom **clSR0bus** postavlja na neaktivnu vrednost, a pri prenosu podatka iz registra $DR_{7...0}$ u registar $DRAUX_{7...0}$ signalom **stSR0per** postavlja na aktivnu vrednost.

Flip-floptovi **RDDR** i **WRDR** se koriste za neophodnu sinhronizaciju pri prenosu podataka iz periferije u memoriju i iz memorije u periferiju, respektivno. Flip-flop **RDDR** se koristi pri prenosu podataka iz periferije u memoriju. Njegova aktivna vrednost je indikacija da novi podatak može da se prenese iz registra $DRAUX_{7...0}$ u registar $DR_{7...0}$, a neaktivna da tekući podatak još uvek nije prenet iz registra $DR_{7...0}$ u memoriju i da novi podatak ne može da se prenese iz registra $DRAUX_{7...0}$ u registar $DR_{7...0}$. S toga se na početku pri startovanju kontrolera periferije za prenos iz periferije u memoriju flip-flop **RDDR** signalom **stRDDRbus** postavlja na aktivnu vrednost. Posle toga se po prenosu podatka iz registra $DRAUX_{7...0}$ u registar $DR_{7...0}$ flip-flop **RDDR** signalom **clRDDRper** postavlja na neaktivnu vrednost, a po prenosu podatka iz registra $DR_{7...0}$ u memoriju signalom **stRDDRbus** postavlja na aktivnu vrednost. Flip-flop **WRDR** se koristi pri prenosu podataka iz memorije u periferiju. Njegova aktivna vrednost je indikacija da se u registru $DR_{7...0}$ nalazi podatak i da sadržaj registra $DR_{7...0}$ može da se prenese u registar $DRAUX_{7...0}$, a neaktivna da novi podatak još uvek nije prenet iz memorije u registar $DR_{7...0}$ i da sadržaj registra $DR_{7...0}$ ne može da se prenese u registar $DRAUX_{7...0}$. S toga se na početku po startovanju kontrolera za prenos iz memorije u periferiju flip-flop **WRDR** signalom **clWRDRbus** postavlja na neaktivnu vrednost. Posle toga se pri prenosu podatka iz memorije u registar $DR_{7...0}$ flip-flop **WRDR** signalom **stWRDRbus** postavlja na aktivnu vrednost, a pri prenosu podatka iz registra $DR_{7...0}$ u registar $DRAUX_{7...0}$ signalom **clWRDRper** postavlja na neaktivnu vrednost.

6.2.1.2 Blok interfejs

Blok *interfejs* (slika 71) sadrži kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je kontroler sluga i za generisanje prekida.



Slika 71 Blok interfejs (prvi deo)

Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je kontroler sluga formiraju signale **rdKTR** i **wrKTR** upravljačke jedinice *uprav_bus* i **FCBUS** magistrale **BUS** (slika 71). Signali **rdKTR** i **wrKTR** se formiraju na osnovu signala $ABUS_{15..0}$ sa adresnih linija magistrale i **RDBUS** i **WRBUS** sa upravljačkih linija magistrale. Signal **FCBUS** se formira na osnovu signala **fcKTR** upravljačke jedinice *uprav_bus*.

Pri realizaciji ciklusa čitanja na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$ i upravljačku liniju **RDBUS** magistrale i na njih izbacuje adresu i aktivnu vrednost signala čitanja, respektivno, čime se u kontroleru kao slugi startuje čitanje adresiranog registra. Kontroler po završenom čitanju otvara bafere sa tri stanja za linije podataka $DBUS_{7..0}$ i upravljačku liniju **FCBUS** magistrale i na njih izbacuje sadržaj adresiranog registra i aktivnu vrednost signala završetka operacije čitanja u kontroleru. Procesor prihvata sadržaj sa linija podataka i zatvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$ i upravljačku liniju **RDBUS** magistrale, dok kontroler zatvara bafere sa tri stanja za linije podataka $DBUS_{7..0}$ i upravljačku liniju **FCBUS** magistrale.

Pri realizaciji ciklusa upisa na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$, linije podataka $DBUS_{7..0}$ i upravljačku liniju **WRBUS** magistrale i na njih izbacuje adresu, podatak i aktivnu vrednost signala upisa, respektivno, čime se u kontroleru kao slugi startuje upis u adresirani registar. Kontroler po završenom upisu otvara bafere sa tri stanja za i upravljačku liniju **FCBUS** magistrale i na nju izbacuje aktivnu vrednost signala završetka operacije upisa u kontroleru. Procesor zatvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$, linije podataka $DBUS_{7..0}$ i upravljačku liniju **WRBUS** magistrale, dok kontroler zatvara bafere sa tri stanja za upravljačku liniju **FCBUS** magistrale.

Signali **rdKTR** i **wrKTR** imaju vrednosti upravljačkih signala **RDBUS** i **WRBUS** magistrale, respektivno, kada je aktivna vrednost signala **select** i neaktivne vrednosti kada je

neaktivna vrednost signala **select**. Signal **rdKTR** je kao i signal $\overline{\text{RDBUS}}$ aktivan za vreme operacije čitanja nekog registra kontrolera, dok je signal **wrKTR** kao i signal $\overline{\text{WRBUS}}$ aktivan za vreme operacije upisa u neki od registara kontrolera.

Signal **select** ima aktivnu vrednost ukoliko se na adresnim linijama $\text{ABUS}_{15...0}$ magistrale nalazi adresa iz opsega adresa dodeljenih registrima kontrolera periferije *KP*. Celokupan opseg adresa od 0000h do FFFFh podeljen je na opseg adresa od 0000h do EFFFh, koje su dodeljene memoriji **MEM**, i opseg adresa od F000h do FFFFh, koje su dodeljene registrima po svim kontrolerima periferija. Opseg adresa koje su dodeljene registrima po kontrolerima periferija je predviđen za adresiranje registara u najviše 64 kontrolera i to najviše 64 registra unutar određenog kontrolera. Pri tome prilikom adresiranja nekog od registara kontrolera sadržaj na adresnim linijama $\text{ABUS}_{15...0}$ magistrale ima sledeću strukturu: bitovi $\text{ABUS}_{15...12}$ su sve jedinice, bitovi $\text{ABUS}_{11...6}$ određuju broj kontrolera i bitovi $\text{ABUS}_{5...0}$ adresu registra unutar kontrolera. Broj kontrolera periferije za određeni kontroler periferije se postavlja mikroprekidačima na jednu od vrednosti u opsegu 0 do 63. Registrima $\text{CR}_{7...0}$, $\text{SR}_{7...0}$ i $\text{DR}_{7...0}$ su unutar opsega adresa datog kontrolera dodeljene adrese 0 do 2, pa bitovi $\text{ABUS}_{5...2}$ mora da budu nule dok se njihovo pojedinačno adresiranje realizuje bitovima ABUS_1 i ABUS_0 .

Na osnovu usvojene strukture adresa signal **select** ima aktivnu vrednost ukoliko su na adresnim linijama $\text{ABUS}_{15...12}$ magistrale sve jedinice, na adresnim linijama $\text{ABUS}_{11...6}$ se nalazi vrednost koja odgovara vrednosti postavljenoj mikroprekidačima i na adresnim linijama $\text{ABUS}_{5...2}$ sve nule. Signali **selCR**, **selSR** i **selDR** se dobijaju na izlazima 0 do 2 dekodera DC na osnovu vrednosti signala ABUS_1 , ABUS_0 i **select** sa ulaza 1, 0 i E, respektivno. Pri neaktivnoj vrednosti signala **select** i signali **selCR**, **selSR** i **selDR** su neaktivni. Pri aktivnoj vrednosti signala **select** jedan od signala **selCR**, **selSR** i **selDR** postaje aktivan i to signal određen binarnom vrednošću signala ABUS_1 i ABUS_0 .

Signal **select** ima neaktivnu vrednost ukoliko se na adresnim linijama $\text{ABUS}_{15...0}$ magistrale ne nalazi adresa iz opsega adresa dodeljenih registrima kontrolera periferije, pri čemu to može da bude ili adresa memorijske lokacije ili registra iz nekog drugog kontrolera periferije. Tada se formiraju neaktivne vrednosti signala **rdKTR** i **wrKTR** bez obzira na vrednosti signala $\overline{\text{RDBUS}}$ i $\overline{\text{WRBUS}}$, respektivno.

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

Kada kontroler treba u procesoru da izazove prekid, generiše se aktivna vrednost signala **intr**. Signala **intr** ima aktivnu vrednost ukoliko ili signal **intrbus** upravljačke jedinice *uprav_bus* ili signal **intrper** upravljačke jedinice *uprav_per* ima aktivnu vrednost.

6.2.2 Upravljačka jedinica

Upravljačka jedinica **uprav** se sastoji iz dva dela:

- upravljačke jedinice magistrale *uprav_bus* i
- upravljačke jedinice periferije *uprav_per*.

Upravljačke jedinice *uprav_bus* i *uprav_per* rade istovremeno i omogućuju paralelan rad kontrolera periferije *KP* kao sluge sa magistralom **BUS** i kontrolera periferije *KP* sa periferijom *PER*.

Upravljačka jedinica *uprav_bus* omogućuje da processor **CPU** kao gazda realizuje u kontroleru periferije *KP* kao slugi upisivanje sadržaja u registre i čitanje sadržaja iz registara. Upisivanjem u upravljački registar sadržaja koji na poziciji bita start ima vrednost 1 startuje se kontroler za prenos podataka iz periferije u memoriju ili iz memorije u periferiju, dok se upisivanjem sadržaja koji na poziciji bita start ima vrednost 0, zaustavlja kontroler.

Prilikom startovanja kontrolera periferije za prenos podataka iz periferije u memoriju aktivira se upravljačka jedinica *uprav_per*. Upravljačka jedinica *uprav_per* prvo startuje periferiju da pročita podatak, a zatim podatak prenosi iz periferije, najpre, u pomoćni registar podatka kontrolera, a potom u registar podatka kontrolera. Podatak se prenosi iz registra podatka kontrolera u memoriju programskim putem sa dve instrukcije. Prvom instrukcijom se, uz aktiviranje upravljačke jedinice *uprav_bus*, podatak prenosi iz registra podatka kontrolera u procesor. Drugom instrukcijom se podatak prenosi iz procesora u memoriju. Prenos podatka iz registra podatka u memoriju i čitanje novog podatka iz periferije sa prenosom iz registra podatka periferije u pomoćni registar podatka kontrolera su preklopljeni. Upravljačka jedinica *uprav_per* novi podatak prenosi iz pomoćnog registra podatka u registar podatka tek pošto je upravljačka jedinica *uprav_bus* prethodni podatak prenela iz registra podatka u procesor. Po prenosu podatka iz pomoćnog registra podatka u registar podatka, bit spremnost statusnog registra se postavlja na aktivnu vrednost i generiše prekid ukoliko je pri startovanju zadat režim rada sa generisanjem prekida. Prenos podataka iz periferije u memoriju traje dok se kontroler periferije ne zaustavi programskim putem.

Prilikom startovanja kontrolera periferije za prenos podataka iz memorije u periferiju aktivira se upravljačka jedinica *uprav_per*. Podatak se prenosi iz memorije u registar podatka kontrolera programskim putem sa dve instrukcije. Prvom instrukcijom se podatak prenosi iz memorije u procesor. Drugom instrukcijom se, uz aktiviranje upravljačke jedinice *uprav_bus*, podatak prenosi iz procesora u registar podatka kontrolera. Posle toga upravljačka jedinica *uprav_per* podatak prenosi iz registra podatka kontrolera u pomoćni registar podatka kontrolera i startuje upis podatka u periferiju. Upis podatka iz pomoćnog registra podatka u periferiju i prenos novog podatka iz memorije u registar podatka kontrolera su preklopljeni. Upravljačka jedinica *uprav_per* novi podatak prenosi iz registra podatka kontrolera u pomoćni registar podatka kontrolera tek pošto prethodni podatak iz pomoćnog registra podatka kontrolera upiše u periferiju. Po prenosu podatka iz registra podatka u pomoćni registar podatka, bit spremnost statusnog registra se postavlja na aktivnu vrednost i generiše prekid ukoliko je pri startovanju zadat režim rada sa generisanjem prekida. Prenos podataka iz memorije u periferiju traje dok se kontroler periferije ne zaustavi programskim putem.

Struktura i opis upravljačkih jedinica *uprav_bus* i *uprav_per* se daju u daljem tekstu.

6.2.2.1 Upravljačka jedinica magistrale

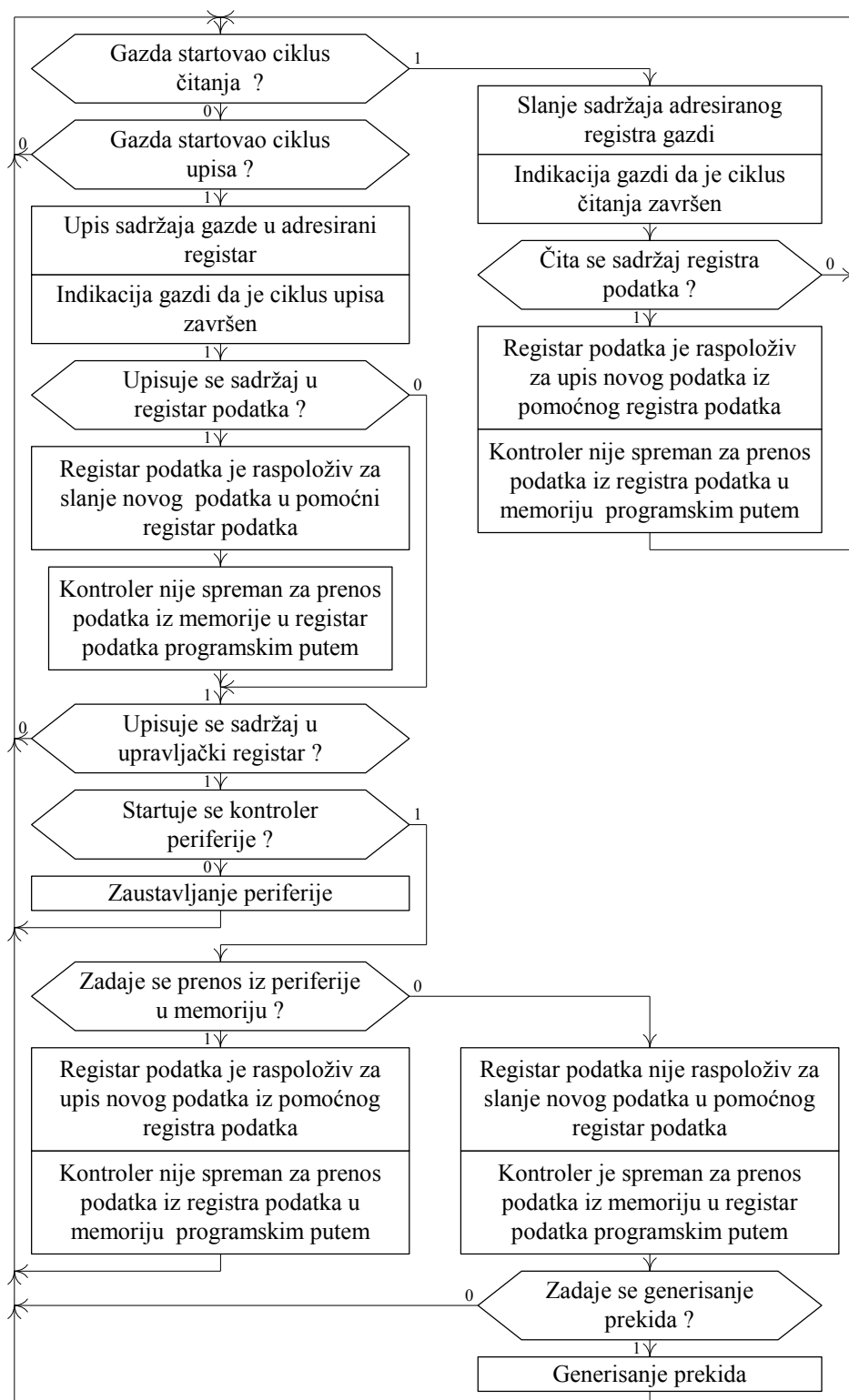
U ovom odeljku se daju dijagram toka operacija, algoritam generisanja upravljačkih signala i struktura upravljačke jedinice *uprav_bus*.

6.2.2.1.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 72). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

U dijagrama toka operacija stalno se vrši provera da li je procesor startovao u kontroleru ciklus čitanja ili ciklus upisa, pri čemu se procesor ponaša kao gazda, a kontroler kao sluga. Ako nije startovan ni ciklus čitanja ni ciklus upisa nema izvršavanja mikrooperacija. Ako je

startovan jedan od ova dva ciklusa izvršavaju se mikrooperacije saglasno ciklusu koji je startovan.



Slika 72 Dijagram toka operacija

Ako je startovan ciklus čitanja, procesoru se kao gazdi šalje sadržaj adresiranog registra kontrolera i indikacija da je kontroler kao sluga završio čitanje. Pored toga, u slučaju kada se čita sadržaj registra podatka, postavljaju se i dve indikacije. Prva je interna indikacija da je sadržaj registar podatka prenet u memoriju i da je on raspoloživ da se u njega prenese novi

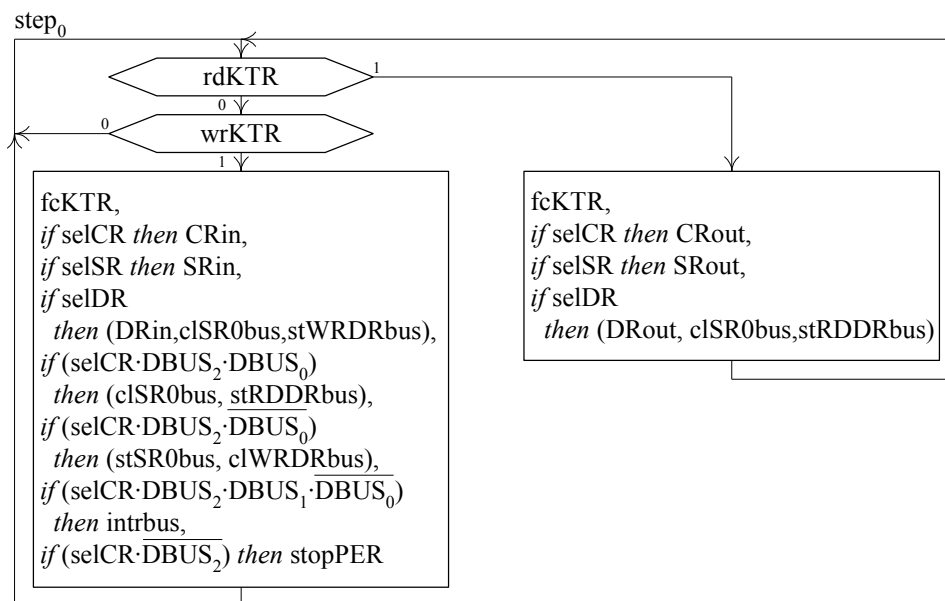
podatak iz pomoćnog registra podatka. Druga je bit spremnost statusnog registra koji se postavlja na neaktivnu vrednost kao indikacija da je registar podatka prenet u memoriju i da ne treba da se čita dok se u njega ne prenese novi podatak iz pomoćnog registra podatka.

Ako je startovan ciklus upisa, u adresirani registar kontrolera se upisuje sadržaj koji procesor kao gazda šalje i gazdi šalje indikacija da je kontroler kao sluga završio upis. U slučaju kada se sadržaj upisuje u registar podatka, postavljaju se i dve indikacije. Prva je interna indikacija da je u registar podatka upisan novi podatak i da je on raspoloživ da se iz njega prenese novi podatak u pomoćni registar podatka. Druga je bit spremnost statusnog registra koji se postavlja na neaktivnu vrednost kao indikacija da je u registar podatka upisan podatak i da ne treba da se vrši upis novog podatka dok se podatak iz njega ne prenese u pomoćni registar podatka.

U slučaju kada se sadržaj upisuje u upravljački registar, mikrooperacije koje se izvršavaju zavise od toga da li se kontroler periferije zaustavlja ili startuje. Ukoliko se vrši zaustavljanje, zaustavlja se i periferija. Ukoliko se vrši startovanje, mikrooperacije koje se izvršavaju zavise od toga da li se kontroler periferije startuje za prenos iz periferije u memoriju ili za prenos iz memorije u periferiju. Ukoliko se vrši startovanje za prenos iz periferije u memoriju postavljaju se i dve indikacije. Prva je interna indikacija da je registar podatka raspoloživ da se u njega prenese podatak iz pomoćnog registra podatka. Druga je bit spremnost statusnog registra koji se postavlja na neaktivnu vrednost kao indikacija da ne treba da se čita dok se u njega ne prenese podatak iz pomoćnog registra podatka. Ukoliko se vrši startovanje za prenos iz memorije u periferiju postavljaju se i dve indikacije. Prva je interna indikacija da u registar podatka nije upisan novi podatak iz memorije i da on nije raspoloživ da se iz njega prenese novi podatak u pomoćni registar podatka. Druga je bit spremnost statusnog registra koji se postavlja na aktivnu vrednost kao indikacija da u registar podatka može da se prenese podatak iz memorije. Pored toga, ukoliko se zadaje rad sa generisanjem prekida, generiše se i prekid.

6.2.2.1.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 72) i dat u obliku dijagrama toka upravljačkih signala (slika 73) i sekvence upravljačkih signala (tabela 26).



Slika 73 Dijagram toka upravljačkih signala

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 73). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova koji određuju grananja u algoritmu.

U sekvenci upravljačkih signala se koriste iskazi za signale (tabela 26). Iskazi za signale su oblika

if uslov then signali

Ovi iskazi sadrže koji sadrže uslov i spisak upravljačkih signala blokova operacione jedinice *oper* i određuje koji signali i pod kojim uslovima treba da budu generisani.

Objašnjenja vezana za generisanje upravljačkih signala su data zajednički za dijagram toka upravljačkih signala (slika 73) i sekvencu upravljačkih signala (tabela 26) i to u okviru sekvence upravljačkih signala.

Tabela 26 Sekvenca upravljačkih signala

! U koraku $step_0$ se ostaje sve vreme. U ovom koraku se ne generiše aktivna vrednost ni jednog od upravljačkih signala sve dok su oba signala **rdKTR** i **wrKTR** bloka *interfejs* neaktivni. Signal **rdKTR** postaje aktivan kada processor **CPU** prebaci adresne linije ABUS_{15..0} magistrale iz stanja visoke impedanse na vrednost adrese nekog od programski dostupnih registara CR_{2...0}, SR₀ ili DR_{7...0} bloka *registri* i upravljačku liniju **RDBUS** magistrale iz stanja visoke impedanse na aktivnu vrednost, čime započinje ciklus čitanja. Signal **wrKTR** postaje aktivan kada processor prebaci adresne linije ABUS_{15..0} i linije podataka DBUS_{7..0} magistrale iz stanja visoke impedanse na vrednost adrese registra i sadržaja za upis u neki od programski dostupnih registara CR_{2...0}, SR₀ ili DR_{7...0}, respektivno, i upravljačku liniju **WRBUS** magistrale iz stanja visoke impedanse na aktivnu vrednost, čime započinje ciklus upisa. Pri aktivnoj vrednosti jednog od signala **rdKTR** i **wrKTR** generiše se signal **fcKTR** bloka *interfejs*. Signalom **fcKTR** se obezbeđuje da upravljačka linija **FCBUS** magistrale pređe iz stanja visoke impedanse na aktivnu vrednost. U oba slučaja se, u zavisnosti od toga koji je od registara CR_{2...0}, SR₀ ili DR_{7...0} adresiran sadržajem na adresnim linijama ABUS_{15..0} magistrale, generiše aktivna vrednost jednog od signala **selCR**, **selSR** i **selDR**, respektivno, bloka *interfejs*. !

! Pri aktivnoj vrednosti signala **rdKTR**, a u zavisnosti od toga koji od signala **selCR**, **selSR** i **selDR** bloka *interfejs* ima aktivnu vrednost, generiše se aktivna vrednost jednog od signala **CRout**, **SRout** i **DRout** bloka *registri*, respektivno. Signalima **CRout**, **SRout** i **DRout** se obezbeđuje da linije podataka DBUS_{7...0} magistrale pređu iz stanja visoke impedanse na vrednost sadržaja jednog od registara CR_{2...0}, SR₀ ili DR_{7...0}, respektivno. Kada signal **rdKTR** postane neaktivan i signali **fcKTR**, **CRout**, **SRout** i **DRout** postanu neaktivni, pa upravljačka linija **FCBUS** magistrale i linije podataka DBUS_{7..0} magistrale se vraćaju u stanje visoke impedanse. !

! Pri aktivnim vrednostima signala **rdKTR** i **selDR**, što znači da se programskim putem čita sadržaj registra DR_{7..0} kao deo postupka prenošenja u memoriju, generišu se signali **clSR0bus** i **stRDDRbus** bloka *registri*. Signalom **clSR0bus** se razred SR₀ statusnog registra SR₀ postavlja na neaktivnu vrednost, što je indikacija da registar DR_{7..0} nije raspoloživ da se čita programskim putem dok se u njega ne prenese novi podatak iz registra DRAUX_{7..0}. Signalom **stRDDRbus** se flip-flop RDDR postavlja na aktivnu vrednost, što je indikacija da je registar DR_{7..0} raspoloživ da se u njega prenese novi podatak iz registra DRAUX_{7..0} bloka *registri*. !

! Pri aktivnoj vrednosti signala **wrKTR**, a u zavisnosti od toga koji od signala **selCR**, **selSR** i **selDR** ima aktivnu vrednost, generiše se aktivna vrednost jednog od signala **CRin**, **SRin** i **DRin** bloka *registri*, respektivno. Signalima **CRin**, **SRin** i **DRin** se obezbeđuje da se sadržaj sa linija podataka DBUS_{7...0} magistrale upiše u jedan od registara CR_{2...0}, SR₀ ili DR_{7...0}, respektivno. Kada signal

wrKTR postane neaktivan i signali **fcKTR**, **CRin**, **SRin** i **DRin** postanu neaktivni, pa se upravljačka linija **FCBUS** magistrale vraća u stanje visoke impedanse. !

! Pri aktivnim vrednostima signala **wrKTR** i **selDR**, što znači da se programskim putem upisuje sadržaj u registar $DR_{7...0}$ kao deo postupka prenošenja iz memorije, generišu se signali **clSR0bus** i **stWRDRbus** bloka *registri*. Signalom **clSR0bus** razred SR_0 statusnog registra SR_0 se postavlja na neaktivnu vrednost, što je indikacija da registar $DR_{7...0}$ nije raspoloživ da se u njega programskim putem upiše novi podatak iz memorije dok se prethodni podatak iz njega ne prenese u registar $DRAUX_{7...0}$. Signalom **stWRDRbus** flip-flop **WRDR** se postavlja na aktivnu vrednost, što je indikacija da je registar $DR_{7...0}$ raspoloživ da se iz njega prenese novi podatak u registar $DRAUX_{7...0}$. !

! Pri aktivnim vrednostima signala **wrKTR**, **selCR**, **DBUS₂** i **DBUS₀**, što znači da se programskim putem upisuje sadržaj u registar $CR_{2...0}$ radi startovanja kontrolera za prenos iz periferije u memoriju, generišu se signali **clSR0bus** i **stRDDRbus** bloka *registri*. Signalom **clSR0bus** razred SR_0 statusnog registra SR_0 se postavlja na neaktivnu vrednost, što je indikacija da registar $DR_{7...0}$ nije raspoloživ da se čita programskim putem dok se u njega ne prenese novi podatak iz registra $DRAUX_{7...0}$. Signalom **stRDDRbus** se flip-flop **RDDR** postavlja na aktivnu vrednost, što je indikacija da je registar $DR_{7...0}$ raspoloživ da se u njega prenese novi podatak iz registra $DRAUX_{7...0}$ bloka *registri*. !

! Pri aktivnim vrednostima signala **wrKTR**, **selCR** i **DBUS₂** i neaktivnoj vrednosti signala **DBUS₀**, što znači da se programskim putem upisuje sadržaj u registar $CR_{2...0}$ radi startovanja kontrolera za prenos iz memorije u periferiju, generišu se signali **stSR0bus** i **clWRDRbus** bloka *registri*. Signalom **stSR0bus** razred SR_0 statusnog registra SR_0 se postavlja na aktivnu vrednost, što je indikacija da je registar $DR_{7...0}$ raspoloživ da se u njega programskim putem upiše podatak iz memorije. Signalom **clWRDRbus** se flip-flop **WRDR** postavlja na neaktivnu vrednost, što je indikacija da je registar $DR_{7...0}$ nije raspoloživ da se iz njega prenese novi podatak u registra $DRAUX_{7...0}$. !

! Pri aktivnim vrednostima signala **wrKTR**, **selCR**, **DBUS₂** i **DBUS₁** i neaktivnoj vrednosti signala **DBUS₀**, što znači da se programskim putem upisuje sadržaj u registar $CR_{2...0}$ radi startovanja kontrolera za prenos iz memorije u periferiju sa generisanjem prekida, generiše se signal prekida **intrbus** bloka *interfejs*. Signal prekida **intrbus** upravljačke jedinice *uprav_bus* sa signalom prekida **intrper** upravljačke jedinice *uprav_per* formira u bloku *interfejs* signal prekida **intr** procesora **CPU**, što je indikacija da je registar $DR_{7...0}$ raspoloživ da se u njega programskim putem skokom na prekidnu rutinu upiše podatak iz memorije. !

! Pri aktivnim vrednostima signala **wrKTR** i **selCR** i neaktivnoj vrednosti signala **DBUS₂**, što znači da se programskim putem upisuje sadržaj u registar $CR_{2...0}$ radi zaustavljanja kontrolera, generiše se signal zaustavljanja **stopPER** periferije *PER*. Signalom **stopPER** se u periferiji *PER* zaustavlja prenos koji je u toku. !

```

step0  if (rdKTR + wrKTR) then fcKTR,
        if (rdKTR·selCR) then CRout,
        if (rdKTR·selSR) then SRout,
        if (rdKTR·selDR) then (DRout, clSR0bus, stRDDRbus),
        if (wrKTR·selCR) then CRin,
        if (wrKTR·selSR) then SRin,
        if (wrKTR·selDR) then (DRin, clSR0bus, stWRDRbus),
        if (wrKTR·selCR·DBUS2·DBUS0) then (clSR0bus, stRDDRbus),
        if (wrKTR·selCR·DBUS2·DBUS0) then (stSR0bus, clWRDRbus),
        if (wrKTR·selCR·DBUS2·DBUS1·DBUS0) then intrbus,
        if (wrKTR·selCR·DBUS2) then stopPER

```

6.2.2.1.3 Struktura upravljačke jedinice

Struktura upravljačke jedinice je prikazana na slici 74. Upravljačka jedinica se sastoji od logičkih elemenata koji na osnovu signala čitanja **rdKTR** i signala upisa **wrKTR** bloka *interfejs* i signala logičkih uslova generišu sledeće signale:

- signal **fcKTR** bloka *interfejs* završetka ili čitanja nekog od registara kontrolera $CR_{2...0}$, SR_0 ili $DR_{7...0}$ bloka *registri* ili upisa u neki od ovih registara kontrolera,
- signale **CRout**, **SRout** i **DRout** bloka *registri* čitanja registara kontrolera $CR_{2...0}$, SR_0 ili $DR_{7...0}$,
- signale **CRin**, **SRin** i **DRin** bloka *registri* upisa u registre kontrolera $CR_{2...0}$, SR_0 ili $DR_{7...0}$,
- signale **clSR0bus** i **stSR0bus** bloka *registri* postavljanja na neaktivnu i aktivnu vrednost, respektivno, bita spremnost SR_0 statusnog registra SR_0 kontrolera,
- signal **stRDDRbus** bloka *registri* postavljanja na aktivnu vrednost flip-flopa **RDDR** i signale **stWRDR** i **clWRDR** bloka *registri* postavljanja na aktivnu i neaktivnu vrednost, respektivno, flip-flopa **WRDR**,
- signal prekida **intrbus** bloka *interfejs* i
- signal zaustavljanja periferije **stopPER** bloka *interfejs*.

Signali **rdKTR** i **wrKTR** imaju aktivne vrednosti trajanja jedna perioda signala takta. To je posledica usvojenog načina generisanja signala **rdCPU** i **wrCPU** procesora **CPU**, na osnovu kojih se formiraju signali **RDBUS** i **WRBUS** magistrale **BUS** i **rdKTR** i **wrKTR** kontrolera **KP**, i signala **fcKTR**, na osnovu koga se generišu signali **FCBUS** magistrale **BUS** i **fcCPU** procesora **CPU**. Kod čitanja procesor na i -ti signal takta ulazi u stanje $step_i$ koje koristi da signal **rdCPU** postavi na aktivnu vrednost, što redom daje i aktivne vrednosti signala **RDBUS**, **rdKTR**, **fcKTR**, **FCBUS** i **fcCPU**. Kako je u procesoru aktivna vrednost signala **fcCPU** uslov za prelazak iz stanja $step_i$ u stanje $step_{i+1}$ i kako ta vrednost dolazi u koraku $step_i$, to procesor na $(i+1)$ -vi signal takta prelazi na korak $step_{i+1}$. Zbog toga se u koraku $step_i$ ostaje samo jedna perioda signala takta, što ima za posledicu da signali **rdCPU**, **RDBUS**, **rdKTR**, **fcKTR**, **FCBUS** i **fcCPU** imaju aktivnu vrednost trajanja jedna perioda signala takta. Iz istih razloga su kod upisa signali **wrCPU**, **WRBUS**, **wrKTR**, **fcKTR**, **FCBUS** i **fcCPU** trajanja jedna perioda signala takta. S obzirom na to da su signali **rdCPU** i **wrCPU** trajanja jedna perioda signala takta i da svi signali koje generiše upravljačke jedinica *uprav_bus* uključuju jedan od signala **rdKTR** i **wrKTR**, to i svi preostali signali imaju aktivne vrednosti trajanja jedna perioda signala takta.

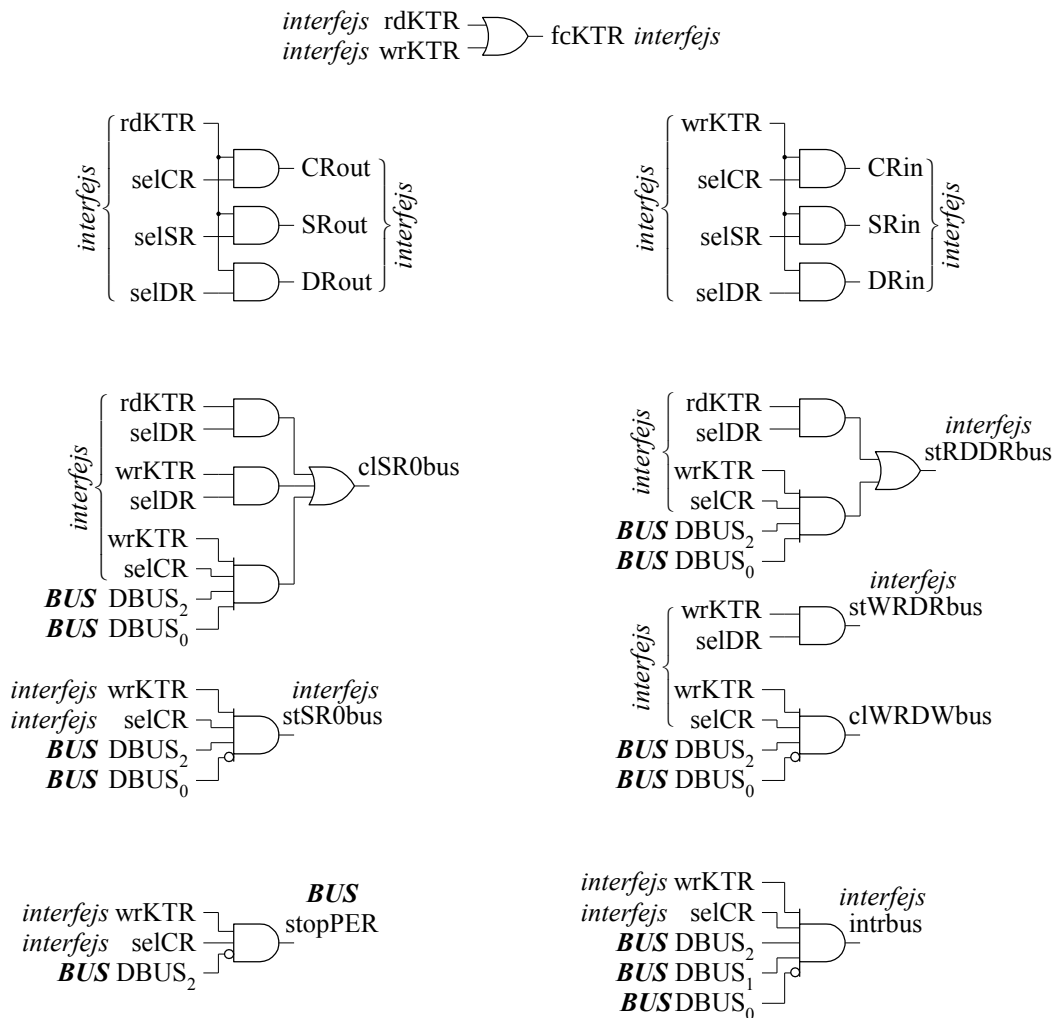
Upravljačka jedinica *uprav_bus* sadrži kombinacione mreže koje pomoću signala **rdKTR** i **wrKTR**, koji dolaze sa bloka *interfejs*, signala logičkih uslova **selCR**, **selSR**, **selDR**, **DBUS₂**, **DBUS₁** i **DBUS₀**, koji dolaze iz bloka *interfejs* i magistrale **BUS**, i saglasno algoritmu generisanja upravljačkih signala (tabela 26) generiše upravljačke signale blokova operacione jedinice.

Upravljački signali blokova operacione jedinice *oper* se daju posebno za blok *registri* i blok *interfejs*.

Upravljački signali bloka *registri* se generišu na sledeći način:

- **CRout** = **rdKTR**·**selCR**
- **SRout** = **rdKTR**·**selSR**
- **DRout** = **rdKTR**·**selDR**
- **clSR0bus** = **rdKTR**·**selDR**

- $\text{stRDDRbus} = \text{rdKTR} \cdot \text{selDR}$
- $\text{CRin} = \text{wrKTR} \cdot \text{selCR}$
- $\text{SRin} = \text{wrKTR} \cdot \text{selSR}$
- $\text{DRin} = \text{wrKTR} \cdot \text{selDR}$
- $\text{clSR0bus} = \text{wrKTR} \cdot \text{selDR}$
- $\text{stWRDRbus} = \text{wrKTR} \cdot \text{selDR}$
- $\text{clSR0bus} = \text{wrKTR} \cdot \text{selCR} \cdot \text{DBUS}_2 \cdot \text{DBUS}_0$
- $\text{stRDDRbus} = \text{wrKTR} \cdot \text{selCR} \cdot \text{DBUS}_2 \cdot \text{DBUS}_0$
- $\text{stSR0bus} = \text{wrKTR} \cdot \text{selCR} \cdot \text{DBUS}_2 \cdot \overline{\text{DBUS}_0}$
- $\text{clWRDRbus} = \text{wrKTR} \cdot \text{selCR} \cdot \text{DBUS}_2 \cdot \overline{\text{DBUS}_0}$



Slika 74 Struktura upravljačke jedinice *uprav_bus*

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz bloka *interfejs* operacione jedinice **oper** i magistrale **BUS** i to:

- **rdKTR** — bloka *interfejs*.
- **wrKTR** — bloka *interfejs*
- **selCR** — bloka *interfejs*
- **selSR** — bloka *interfejs*
- **selDR** — bloka *interfejs*
- **DBUS₂** — magistrale **BUS** i
- **DBUS₀** — magistrale **BUS**.

Upravljački signali bloka *interfejs* se generišu na sledeći način:

- $\text{intrbus} = \text{wrKTR} \cdot \text{selCR} \cdot \text{DBUS}_2 \cdot \text{DBUS}_1 \cdot \overline{\text{DBUS}_0}$
- $\text{stopPER} = \text{wrKTR} \cdot \text{selCR} \cdot \overline{\text{DBUS}_2}$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz bloka *interfejs* operacione jedinice **oper** i magistrale **BUS** i to:

- **wrKTR** — bloka *interfejs*
- **selCR** — bloka *interfejs*
- **DBUS₂** — magistrale **BUS**
- **DBUS₁** — magistrale **BUS** i
- **DBUS₀** — magistrale **BUS**.

6.2.2.2 Upravljačka jedinica periferije

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

6.2.2.2.1 Dijagram toka operacija

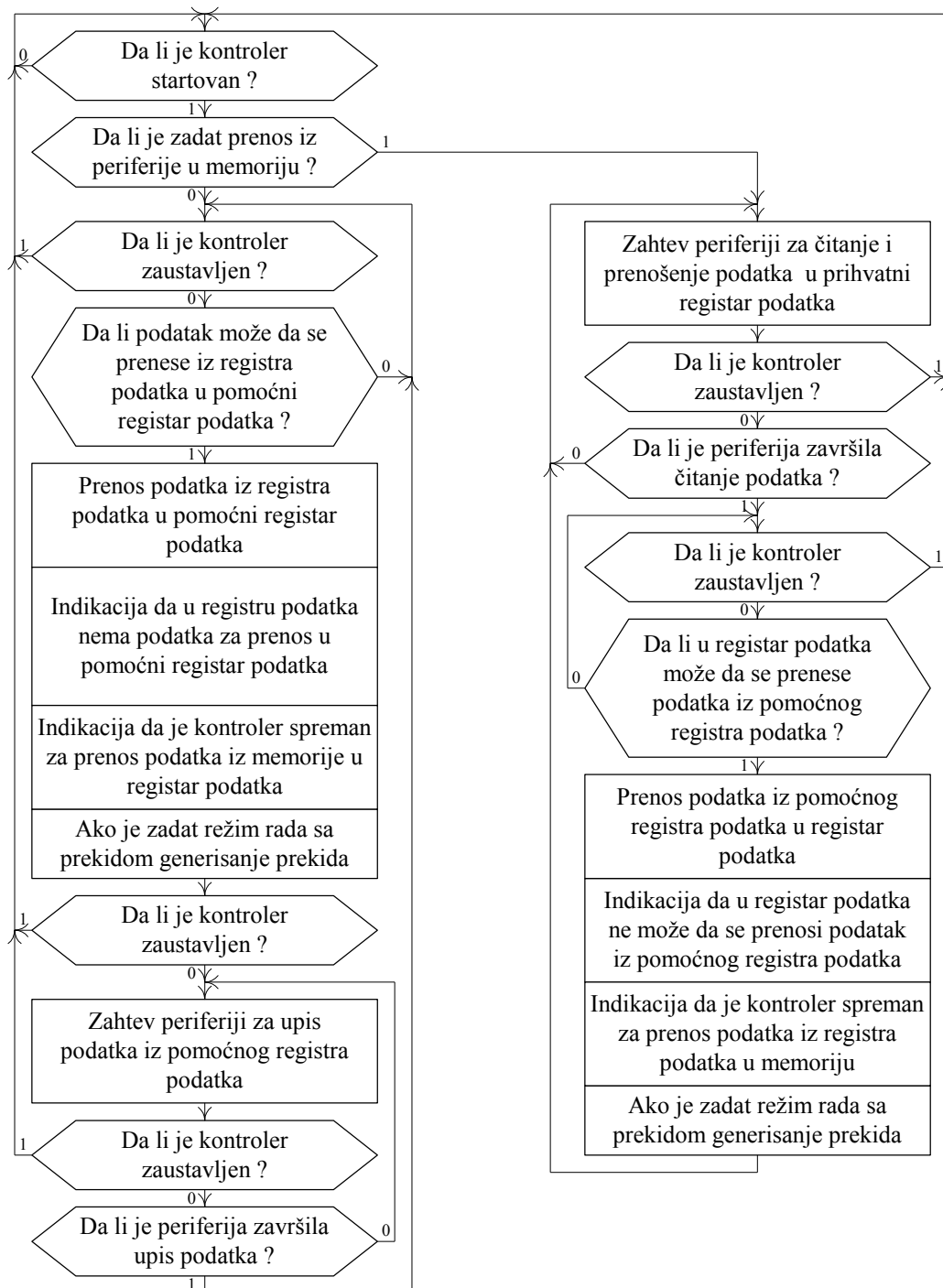
Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 75). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

U početnom koraku vrši se provera da li je kontroler startovan ili ne. Ako kontroler nije startovan ostaje se u početnom koraku i čeka startovanje kontrolera. Ako je kontroler startovan vrši se provera da li je zadat prenos iz periferije u memoriju ili iz memorije u periferiju i prelazi na odgovarajući korak.

Ako je zadat prenos iz periferije u memoriju prelazi se na prenošenje podatka iz periferije u kontroler. Najpre se u petlji drži zahtev periferiji da pročita podatak i vrši provera da li je kontroler zaustavljen programskim putem upisivanjem neaktivne vrednosti u razred start upravljačkog registra i da li je periferija završila čitanje podatka. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz periferije u kontroler, vraća se u početni korak i čeka da se ponovo programskim putem upisivanjem aktivne vrednosti u razred start upravljačkog registra startuje kontroler. Ukoliko se utvrdi da je periferija pročitala podatak izlazi se iz petlje i podatak prenosi iz periferije u pomoćni registar podatka kontrolera.

Potom se u petlji vrši provera da li je kontroler zaustavljen programskim putem i da li je registar podatka raspoloživ da se u njega prenese novi podatak iz pomoćnog registra podatka. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz periferije u kontroler, vraća se u početni korak i čeka da se ponovo programskim putem startuje kontroler. Ukoliko se utvrdi da je registar podatka raspoloživ, izlazi se iz petlje i podatak prenosi iz pomoćnog registra podatka u registar podatka. Istovremeno se postavlja indikacija da registar podatka nije raspoloživ da se u njega prenese novi podatak iz pomoćnog registra podatka sve dok se programskim putem podatak ne prenese iz registra podatka u memoriju. Pored toga bit spremnost statusnog registra kontrolera se postavlja na aktivnu vrednost, čime se postavlja indikacija da je kontroler spreman za prenos podatka iz registra podatka u memoriju, i generiše prekid, ukoliko je pri startovanju zadat režim sa generisanjem prekida. Na kraju se vraća na korak u kome se postavlja zahtev periferiji da pročita sledeći podatak. Opisane aktivnosti se ponavljaju sve dok se kontroler ne zaustavi programskim putem.

Ako je u zadat prenos iz memorije u periferiju prelazi se na prenošenje podatka iz kontrolera u periferiju. Najpre se u petlji vrši provera da li je kontroler zaustavljen programskim putem i da li je registar podatka raspoloživ da se iz njega prenese novi podatak u pomoćni registar podatka. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz kontrolera u periferiju, vraća se u početni korak i čeka da se ponovo programskim putem, startuje kontroler. Ukoliko se utvrdi da je registar podatka raspoloživ izlazi se iz petlje i podatak prenosi iz registra podatka u pomoćni registar podatka. Istovremeno se postavlja indikacija da registar podatka nije raspoloživ da se iz njega prenosi podatak u pomoćni registar podatka sve dok se programskim putem podatak ne prenese iz memorije u registar podatka. Pored toga na aktivnu vrednost se postavlja bit spremnost statusnog registra kontrolera, čime se postavlja indikacija da je kontroler spreman za prenos podatka iz memorije u registar podatka, i generiše prekid, ukoliko je pri startovanju zadat režim sa generisanjem prekida.



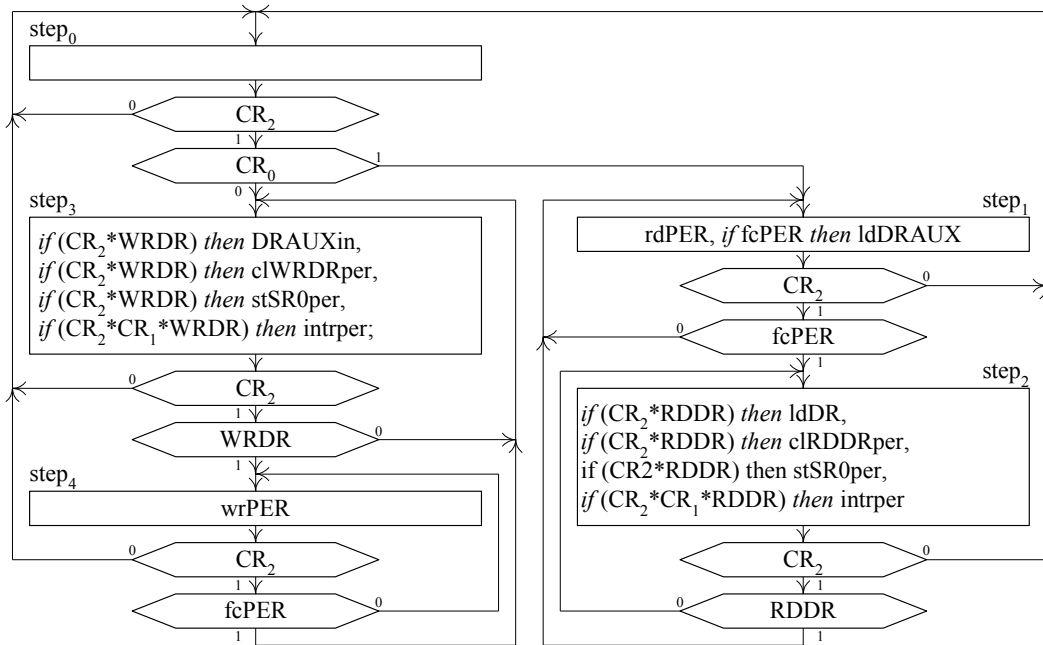
Slika 75 Dijagram toka operacija

Zatim se vrši provera da li je kontroler zaustavljen programskim putem. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz kontrolera u periferiju, vraća se u početni korak i čeka da se ponovo programskim putem startuje kontroler. Ukoliko se utvrdi da kontroler nije zaustavljen u petlji se periferiji drži zahtev da upiše sledeći podatak i vrše provere da li je kontroler zaustavljen programskom putem i da li je periferija završila upis podatka. Ukoliko se utvrdi da je kontroler periferije zaustavljen, prekida se prenošenje podataka iz kontrolera u periferiju, vraća se u početni korak i čeka da se ponovo programskim putem startuje kontroler. Ukoliko se utvrdi da je periferija završila upis podatka iz pomoćnog registra podatka kontrolera, izlazi se iz ove petlje i vraća u petlju u kojoj se vrši provera da li je kontroler zaustavljen programskim putem i da li je registar podatka kontrolera raspoloživ da

se iz njega prenese novi podatak u pomoćni registar podatka kontrolera. Opisane sktivnosti se ponavljaju sve dok se kontroler ne zaustavi programskim putem.

6.2.2.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 75) i dat u obliku dijagrama toka upravljačkih signala (slika 76) i sekvence upravljačkih signala po koracima (tabela 27).



Slika 76 Dijagram toka upravljačkih signala

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 76). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova.

U sekvenci upravljačkih signala po koracima je za svaki korak data simbolička oznaka samog koraka i iskazi za signale i skokove. Iskazi za signale se pojavljuju ukoliko u datom koraku treba da se generiše neki od upravljačkih signala operacione jedinice. Iskazi za signale sadrže spisak upravljačkih signala operacione jedinice koji se generišu bezuslovno i uslovno, a za signale koji se generišu uslovno i signale logičkih uslova pod kojima se signali generišu. Iskazi za skokove se pojavljuju ukoliko u treba odstupiti od sekvencijalnog generisanja upravljačkih signala operacione jedinice. Iskazi za skokove sadrže uslov i korak i određuje na koji korak i pod kojim uslovima treba preći. Notacija koja se koristi je identična kao i notacija za algoritam generisanja upravljačkih signala za upravljačku jedinicu procesora **CPU** (poglavlje \$\$\$).

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 27 Sekvenca upravljačkih signala

! U koraku se step₀ se ostaje i ne generiše se aktivna vrednost ni jednog od upravljačkih signala sve dok je signal **CR₂** bloka *registri* neaktivan. Signal **CR₂** postaje aktivan tek kada se programskim

putem startuje kontroler periferije. Tom prilikom se u upravljački registar $CR_{2...0}$ bloka *registri* upisuje sadržaj koji na poziciji razreda CR_2 ima aktivnu vrednost. Na poziciji razreda CR_0 je aktivna ili neaktivna vrednost i zavisnosti od toga da li je zadat prenos iz periferije u memoriju ili iz memorije u periferiju, respektivno. Na poziciji razreda CR_1 je aktivna ili neaktivna vrednost i zavisnosti od toga da li je zadat režim rada sa generisanjem ili bez generisanja prekida, respektivno. Pri aktivnim vrednostima signala CR_2 i CR_0 se prelazi na korak $step_1$ koji odgovara prenosu iz periferije u memoriju. Pri aktivnoj i neaktivnoj vrednosti signala CR_2 i CR_0 , respektivno, se prelazi na korak $step_3$, koji odgovara prenosu iz memorije u periferiju. !

$step_0$ $br\ (if\ (CR_2 \cdot CR_0)\ then\ step_1),$
 $br\ (if\ (CR_2 \cdot \overline{CR_2})\ then\ step_3);$

! U korak $step_1$ se dolazi iz koraka $step_0$ ili koraka $step_2$. U ovom koraku se generiše signal **rdPER** koji predstavlja zahtev periferiji da pročita sledeći podatak. Ukoliko je neaktivna vrednost signala CR_2 , što znači da je programskim putem upisivanjem neaktivne vrednosti u razred CR_2 upravljačkog registra CR zaustavljen rad kontrolera, prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_1$ ili se prelazi u korak $step_2$ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signala **fcPER** periferije *PER*, respektivno. Neaktivna vrednost signala **fcPER** je indikacija da čitanje podatka u periferiji još uvek traje, a aktivna vrednost da je podatak pročitao i da se nalazi na linijama $PIN_{7...0}$ periferije. Pri aktivnim vrednostima signala CR_2 i **fcPER** se generiše signal **ldDRAUX**. Ovim signalom se sadržaj iz periferije, koji u kontroler dolazi po linijama $PIN_{7...0}$, propušta kroz multiplekser MP2 bloka *registri* i upisuje u pomoćni registar podatka $DRAUX_{7...0}$ bloka *registri*. !

$step_1$ **rdPER**, $if\ (CR_2 \cdot fcPER)\ then\ ldDRAUX,$
 $br\ (if\ \overline{CR_2}\ then\ step_0),$
 $br\ (if\ (CR_2 \cdot fcPER)\ then\ step_2);$

! U korak $step_2$ se dolazi iz koraka $step_1$. Ukoliko je neaktivna vrednost signala CR_2 prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_2$ ili se prelazi u korak $step_1$ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signal **RDDR** bloka *registri*, respektivno. Neaktivna vrednost signala **RDDR** je indikacija da registar podatka $DR_{7...0}$ nije raspoloživ, jer prethodni podatak još uvek nije programskim putem prenet iz registra $DR_{7...0}$ u memoriju. Aktivna vrednost signala **RDDR** je indikacija da je registar podatka $DR_{7...0}$ raspoloživ, pa se generišu signali **ldDDR**, **clRDDRper** i **stSR0per**. Signalom **ldDDR** se sadržaj registra $DRAUX_{15...0}$ bloka *registri* propušta kroz multiplekser MP1 bloka *registri* i upisuje u registar podatka $DR_{7...0}$. Signalom **clRDDRper** se u flip-flop **RDDR** upisuje neaktivna vrednost. Ova vrednost je indikacija da registar $DR_{7...0}$ nije raspoloživ da se u njega prenesu novi podatak iz registra $DRAUX_{7...0}$ sve dok se programskim putem prethodni podatak ne prenesu iz registra $DR_{7...0}$ u memoriju. Signalom **stSR0per** se upisuje aktivna vrednost u razred SR_0 statusnog registra SR_0 bloka *registri*. Ovaj razred odgovara bitu spremnost i njegova aktivna vrednost pri prenosu iz periferije u memoriju služi kao indikacija procesoru da može programskim putem da prenesu podatak iz registra $DR_{7...0}$ u memoriju. Ukoliko je pri aktivnim vrednostima signala CR_2 i **RDDR** i signal CR_1 aktivan, generiše se signal **intrper** bloka *interfejs*. Ovim signalom se pri prenosu iz periferije u memoriju generiše signala prekida **intr**, da bi procesor skočio na odgovarajuću prekidnu rutinu i u okviru nje programskim putem preneo podatak iz registra $DR_{7...0}$ u memoriju. !

$step_2$ $if\ (CR_2 \cdot RDDR)\ then\ (ldDDR,\ stSR0per,\ clRDDRper),$
 $if\ (CR_2 \cdot CR_1 \cdot RDDR)\ then\ intrper,$
 $br\ (if\ \overline{CR_2}\ then\ step_0),$
 $br\ (if\ (CR_2 \cdot RDDR)\ then\ step_1);$

! U korak $step_3$ se dolazi iz koraka $step_0$ ili koraka $step_4$. Ukoliko je neaktivna vrednost signala CR_2 prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_3$ ili se prelazi u korak $step_4$ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signal **WRDR** bloka *registri*, respektivno. Neaktivna vrednost signala **WRDR** je indikacija da registar podatka $DR_{7...0}$ bloka *registri* nije raspoloživ, jer novi podatak još uvek nije programskim putem prenet iz memorije u registar $DR_{7...0}$. Aktivna vrednost signala **WRDR** je indikacija da je registar $DR_{7...0}$ raspoloživ jer je

podatak programskim putem prenet iz memorije u registar podatka DR_{7...0}. Po upisu podatka u registar DR_{7...0} upravljačka jedinica *uprav_bus* signalom **stWRDRbus** postavlja flip-flop WRDR na aktivnu vrednost. Pri aktivnoj vrednosti signala **WRDR** generišu se signali **DRAUXin**, **clWRDRper** i **stSR0per**. Neaktivnom vrednošću signal **ldDRAUX** se sadržaj registra DR_{7...0} propušta kroz multiplexer MP2 bloka *registri*, a aktivnom vrednošću signala **DRAUXin** se upisuje u pomoćni registar podatka DRAUX_{7...0} bloka *registri*. Signalom **clWRDRper** se u flip-flop WRDR upisuje neaktivna vrednost. Ova vrednost je indikacija da registar DR_{7...0} nije raspoloživ i da njegov sadržaj ne može da se prenese u registar DRAUX_{7...0} sve dok se programskim putem novi podatak ne prenese iz memorije u registar DR_{7...0}. Signalom **stSR0per** se upisuje aktivna vrednost u razred SR₀ statusnog registra SR₀ bloka *registri*. Ovaj razred odgovara bitu spremnost i njegova aktivna vrednost pri prenosu iz memorije u periferiju služi kao indikacija procesoru da može programskim putem da prenese podatak iz memorije u registar DR_{7...0}. Ukoliko je pri aktivnim vrednostima signala **CR₂** i **WRDR** i signal **CR₁** aktivan, generiše se signal **intrper** bloka *interfejs*. Ovim signalom se pri prenosu iz memorije u periferiju generiše signal prekida **intr**, da bi procesor skočio na odgovarajuću prekidnu rutinu i u okviru nje programskim putem preneo podatak iz memorije u registar DR_{7...0}. !

```

step3   if (CR2·WRDR) then (DRAUXin, clWRDRper, stSR0per),
          if (CR2·CR1·WRDR) then intrper,
          br (if CR2 then step0),
          br (if (CR2·WRDR) then step4);

```

! U korak step₄ se dolazi iz koraka step₃. U ovom koraku se generiše signal **wrPER** koji predstavlja zahtev periferiji da upiše podatak iz registra DRAUX_{7...0} koji je prisutan na linijama POUT_{7...0}. Ukoliko je neaktivna vrednost signala **CR₂** prelazi se u korak step₀. Ukoliko je aktivna vrednost signala **CR₂**, ili se ostaje u koraku step₄ ili se prelazi u korak step₃ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signal **fcPER** periferije *PER*, respektivno. Neaktivna vrednost signala **fcPER** je indikacija da je upis podatka iz registra DRAUX_{7...0} u periferiju u toku, a aktivna vrednost da je podatak upisan. !

```

step4   wrPER,
          br (if CR2 then step0),
          br (if (CR2·fcPER) then step3);

```

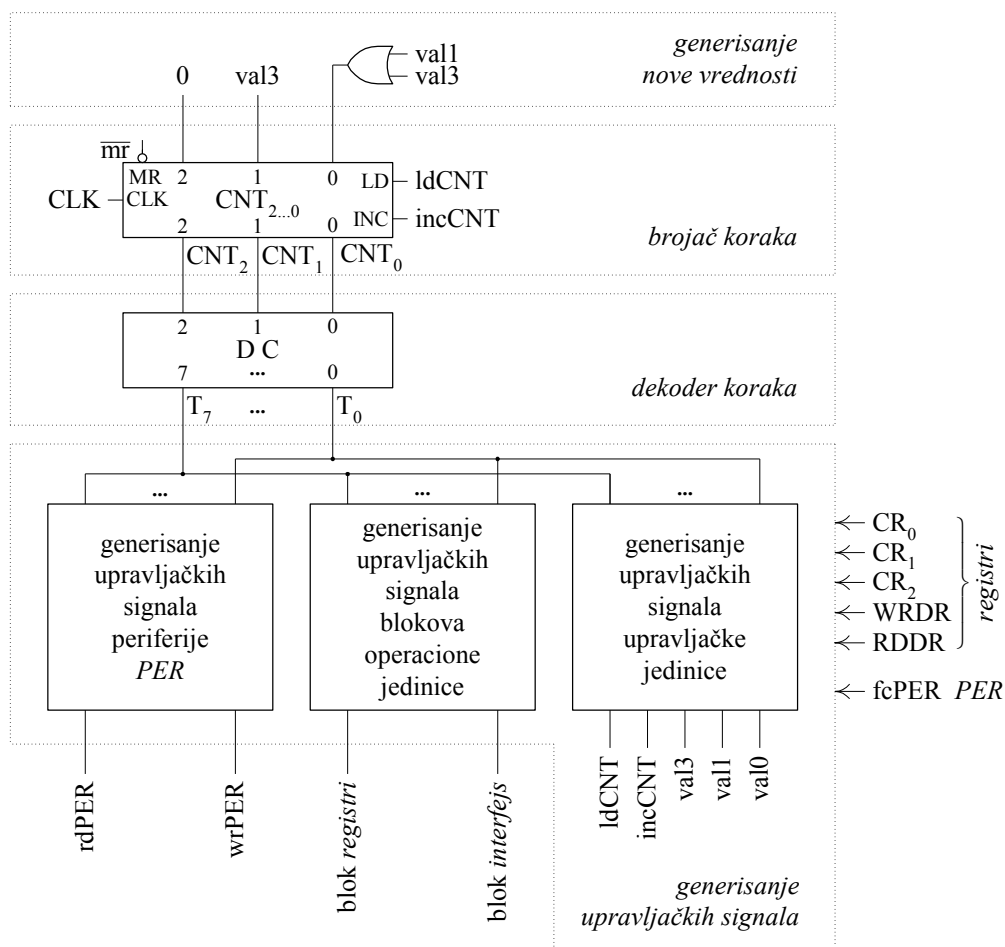
6.2.2.2.3 Struktura upravljačke jedinice

Upravljačka jedinica (slika 77) se sastoji od sledećih blokova:

- blok *generisanje nove vrednosti brojača koraka*,
- blok *brojač koraka*,
- blok *dekoder koraka* i
- blok *generisanje upravljačkih signala*.

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

Blok *generisanje nove vrednosti brojača koraka* služi za generisanje vrednosti koju treba upisati u brojač CNT. Potreba za ovim se javlja kada treba odstupiti od sekvencijalnog generisanja upravljačkih signala. Analizom algoritma generisanja upravljačkih signala operacione jedinice (poglavlje 6.2.2.2.2) se utvrđuje da su 0, 1 i 3 vrednosti koje treba upisati u brojač CNT da bi se realizovala odstupanja od sekvencijalnog generisanja upravljačkih signala. Te vrednosti se formiraju na ulazima 2, 1 i 0 brojača CNT pomoću signala **val₀**, **val₁** i **val₃** pri čemu signal **val₀** ima vrednost 1 samo onda kada treba upisati 0, signal **val₁** ima vrednost 1 samo onda kada treba upisati 1 i signal **val₃** ima vrednost 1 samo onda kada treba upisati 3. Time se obezbeđuje da na ulazima 2, 1 i 0 brojača CNT budu vrednosti 0, 0 i 0 onda kada signal **val₀** ima vrednost 1, vrednosti 0, 0 i 1 onda kada signal **val₁** ima vrednost 1 i vrednosti 0, 1, i 1 onda kada signal **val₃** ima vrednost 1.



Slika 77 Struktura upravljačke jedinice *uprav_per*

Blok *brojač koraka* sadrži brojač CNT. Brojač CNT svojom trenutnom vrednošću obezbeđuje aktivne vrednosti određenih upravljačkih signala. Brojač CNT može da radi u sledećim režimima:

- režim inkrementiranja,
- režim skoka i
- režim mirovanja.

U režimu inkrementiranja pri pojavi signala takta vrši se uvećavanje sadržaja brojača CNT za jedan. Ovim režimom se obezbeđuje sekvencijalno generisanje upravljačkih signala iz algoritma generisanja upravljačkih signala (poglavljje 6.2.2.2.2). Ovaj režim rada se obezbeđuje aktivnom vrednošću signala **incCNT**. Signal **incCNT** je uvek neaktivan sem kada treba obezbediti režim inkrementiranja.

U režimu skoka pri pojavi signala takta vrši se upis nove vrednosti u brojač CNT. Ovim režimom se obezbeđuje odstupanje od sekvencijalnog generisanja upravljačkih signala iz algoritma generisanja upravljačkih signala (poglavljje 6.2.2.2.2). Ovaj režim rada se obezbeđuje aktivnom vrednošću signala **ldCNT**. Signal **ldCNT** je uvek neaktivan sem kada treba obezbediti režim skoka.

U režimu mirovanja pri pojavi signala takta ne menja se vrednost brojača CNT. Ovaj režim rada se obezbeđuje neaktivnim vrednostima signala **incCNT** i **ldCNT**. Ovi signali su neaktivni kada se u koraku

- step₀ pri aktivnoj vrednosti signala T₀ čeka da kontroler bude startovan upisivanjem aktivne vrednosti u razred CR₂ upravljačkog registra kontrolera,

- step₁ pri aktivnoj vrednosti signala T₁ čeka da čitanje u periferiji bude završeno i signal fcPER postane aktivan,
- step₂ pri aktivnoj vrednosti signala T₂ čeka da registar podatka DR_{7...0} bude raspoloživ za prebacivanje podatka iz pomoćnog registra podatka DRAUX_{7...0} i signal RDDR postane aktivan,
- step₃ pri aktivnoj vrednosti signala T₃ čeka da pomoćni registar podatka DRAUX_{7...0} bude raspoloživ za prebacivanje podatka iz registra podatka DR_{7...0} i signal WRDR postane aktivan i
- step₄ pri aktivnoj vrednosti signala T₄ čeka da upis u periferiju bude završen i signal fcPER postane aktivan.

Blok *dekoder koraka* sadrži dekode DC. Na ulaze dekodera DC vode se izlazi brojača CNT. Dekodovana stanja brojača CNT pojavljuju se kao signali T₀ do T₇ na izlazima dekodera DC. Svakom koraku iz algoritma generisanja upravljačkih signala (poglavlje 6.2.2.2.2) dodeljen je jedan od ovih signala i to koraku step₀ signal T₀, koraku step₁ signal T₁, itd.

Blok *generisanje upravljačkih signala* sadrži kombinacione mreže koje pomoću signala T₀ do T₇ koji dolaze sa bloka *dekoder koraka* upravljačke jedinice, signala logičkih uslova koji dolaze iz blokova operacione jedinice i saglasno algoritmu generisanja upravljačkih signala (poglavlje 6.2.2.2.2) generišu upravljačke signale. Upravljački signali se generišu na identičan način kao i upravljački signali procesora CPU (poglavlje 6.2.2.2.2).

Blok *generisanje upravljačkih signala* generiše tri grupe upravljačkih signala i to:

- upravljačke signale periferije PER,
- upravljačke signale blokova operacione jedinice *oper* i
- upravljačke signale upravljačke jedinice *uprav_per*.

Upravljački signali periferije PER se generišu na sledeći način:

- $rdPER = CR_2 \cdot T_1$
- $wrPER = CR_2 \cdot T_4$

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

- CR₂ — blok *registri*.

Upravljački signali blokova operacione jedinice *oper* se daju posebno za blok *registri* i blok *interfejs*.

Upravljački signali bloka *registri* se generišu na sledeći način:

- $stSR0per = CR_2 \cdot RDDR \cdot T_2 + CR_2 \cdot WRDR \cdot T_3$
- $clRDDRper = CR_2 \cdot RDDR \cdot T_2$
- $clWRDRper = CR_2 \cdot WRDR \cdot T_3$
- $ldDR = CR_2 \cdot RDDR \cdot T_2$
- $DRAUXin = CR_2 \cdot WRDR \cdot T_3$
- $ldDRAUX = CR_2 \cdot fcPER \cdot T_1$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz blokova operacione jedinice i periferije PER i to:

- CR₂ — blok *registri*,
- RDDR — blok *registri*,
- WRDR — blok *registri*,
- fcPER — periferije PER.

Upravljački signal bloka *interfejs* se generiše na sledeći način:

- $\text{intrper} = \text{CR}_2 \cdot \text{CR}_1 \cdot \text{RDDR} \cdot \text{T}_2 + \text{CR}_2 \cdot \text{CR}_1 \cdot \text{WRDR} \cdot \text{T}_3$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz blokova operacione jedinice i periferije *PER* i to:

- CR_2 — blok *registri*,
- CR_1 — blok *registri*,
- RDDR — blok *registri*,
- WRDR — blok *registri*.

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

- $\text{ldCNT} = \text{val0} + \text{val1} + \text{val3}$
- $\text{val0} = \overline{\text{CR}_2} \cdot (\text{T}_1 + \text{T}_2 + \text{T}_3 + \text{T}_4)$
- $\text{val1} = \text{CR}_2 \cdot \text{RDDR} \cdot \text{T}_2$
- $\text{val3} = \text{CR}_2 \cdot \overline{\text{CR}_0} \cdot \text{T}_0 + \text{CR}_2 \cdot \text{fcPER} \cdot \text{T}_4$
- $\text{incCNT} = \text{CR}_2 \cdot \text{CR}_0 \cdot \text{T}_0 + \text{CR}_2 \cdot \text{fcPER} \cdot \text{T}_1 + \text{CR}_2 \cdot \text{WRDR} \cdot \text{T}_3$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz periferije *PER* i pojedinih blokova operacione jedinice *oper* i to:

- fcPER — periferija *PER*,
- CR_0 — blok *registri*,
- CR_2 — blok *registri*,
- RDDR — blok *registri* i
- WRDR — blok *registri*.

6.3 KONTROLER PERIFERIJE SA DIREKTNIM PRISTUPOM MEMORIJI

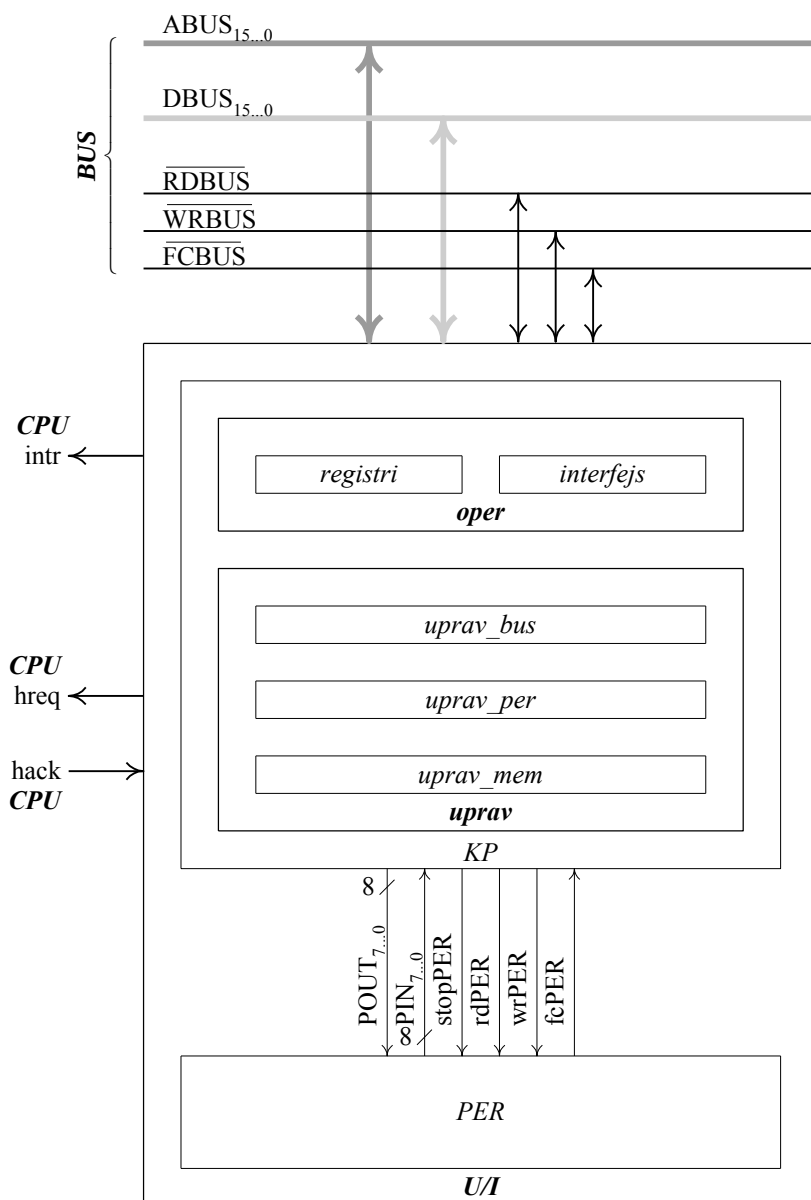
Kontroler periferije *KP* (slika 78) se sastoji iz:

- operacione jedinice *oper* i
- upravljačke jedinice *uprav*.

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* se sastoji iz tri dela i to upravljačke jedinice magistrale *uprav_bus*, upravljačke jedinice periferije *uprav_per* i upravljačke jedinice memorije *uprav_mem*. Upravljačka jedinica magistrale *uprav_bus* generiše upravljačke signale neophodne da kontroler periferije *KP* kao sluga realizacije cikluse na magistrali **BUS** kojima se prenose podaci između procesora **CPU** i kontrolera periferije *KP*. Upravljačka jedinica periferije *uprav_per* generiše upravljačke signale neophodne za prenos podataka između periferije *PER* i kontrolera periferije *KP*. Upravljačka jedinica memorije *uprav_mem* generiše upravljačke signale neophodne da kontroler periferije *KP* kao gazda realizacije cikluse na magistrali **BUS** kojima se prenose podaci između kontrolera periferije *KP* i memorije **MEM**. Upravljačke jedinice *uprav_bus*, *uprav_per* i *uprav_mem* rade istovremeno i omogućuju paralelan rad kontrolera periferije *KP* kao sluge sa magistralom **BUS**, kontrolera periferije *KP* sa periferijom *PER* i kontrolera periferije *KP* kao gazde sa magistralom **BUS**. Svaka od upravljačkih jedinica *uprav_bus*, *uprav_per* i *uprav_mem* je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za generisanje upravljačkih signala prema algoritmu generisanja upravljačkih signala operacione jedinice *oper* i signala logičkih uslova.

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



Slika 78 Organizacija kontrolera periferije *KP*

6.3.1 Operaciona jedinica

Operaciona jedinica (slika 78) se sastoji od sledećih blokova:

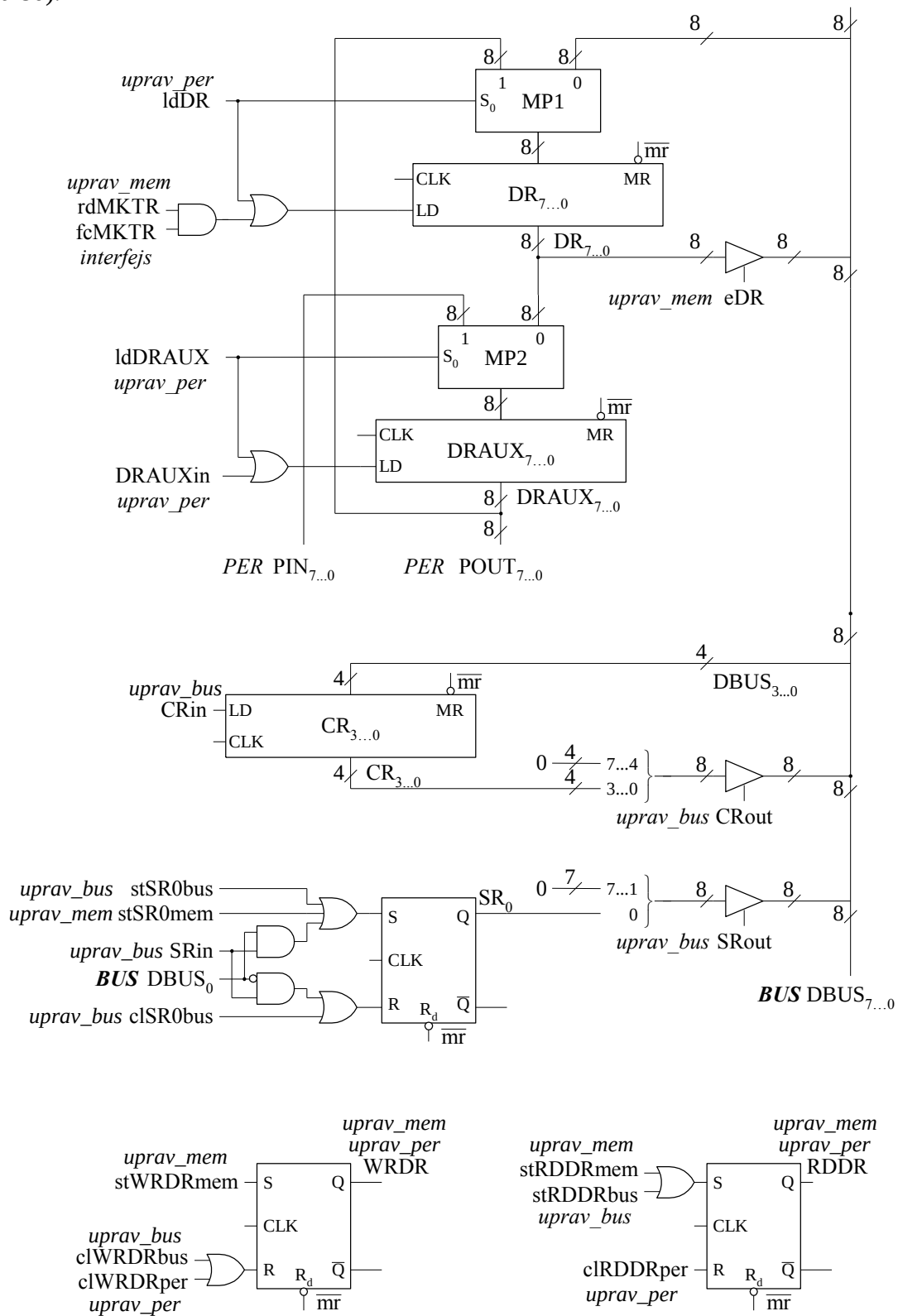
- blok *registri* i
- blok *interfejs*.

Blok *registri* (slike 79 i 80) služi za čuvanje podataka, upravljačkih i statusnih informacija. Blok *interfejs* (slike 81) služi za realizaciju ciklusa na magistrali kada je kontroler sluga, realizaciju prekida, dobijanje dozvole korišćenja magistrale i realizaciju ciklusa na magistrali kada je kontroler gazda.

Struktura i opis blokova operacione jedinice **oper** se daju u daljem tekstu.

6.3.1.1 Blok registri

Blok *registri* sadrži registre DR_{7...0} i DRAUX_{7...0} sa multiplekserima MP1 i MP2, registre CR_{3...0} i SR₀, flip-flobove WRDR i RDDR (slika 80) i registre WCR_{15...0} i AR_{15...0} (slika 80).



Slika 79 Blok registri (prvi deo)

Registri $DR_{7...0}$ i $DRAUX_{7...0}$ su 8-mo razredni registar podatka i pomoćni registar podatka, respektivno (slika 79). Ovi registri zajedno sa multiplekserima MP1 i MP2 služe za prenos podataka između periferije *PER* i memorije *MEM*.

Pri prenosu podataka iz periferije u memoriju podatak koji dolazi iz periferije po linijama $PIN_{7...0}$ periferije *PER* se, najpre, propušta kroz multiplekser MP2 i upisuje u pomoćni registar podatka $DRAUX_{7...0}$. Potom se podatak iz registra $DRAUX_{7...0}$ propušta kroz multiplekser MP1 i upisuje u registar podatka $DR_{7...0}$. Na kraju se podatak iz registra $DR_{7...0}$ preko bafera sa tri stanja propušta na linije podataka $DBUS_{7...0}$ magistrale *BUS* i upisuje u memoriju. Ovakvin načinom prenosa podataka je omogućeno da se istovremeno prenosi tekući podatak iz registra $DR_{7...0}$ u memoriju i sledeći podatak iz periferije u registar $DRAUX_{7...0}$. Pri tome se sledeći podatak prenosi iz registra $DRAUX_{7...0}$ u registar $DR_{15...0}$, tek pošto je tekući podatak prenet iz registra $DR_{7...0}$ u memoriju.

Pri prenosu podataka iz memorije u periferiju podatak koji dolazi iz memorije po linijama podataka $DBUS_{7...0}$ magistrale *BUS* se, najpre, propušta kroz multiplekser MP1 i upisuje u registar podatka $DR_{7...0}$. Potom se podatak iz registra $DR_{7...0}$ propušta kroz multiplekser MP2 i upisuje u pomoćni registar podatka $DRAUX_{7...0}$. Na kraju se podatak iz registra $DRAUX_{7...0}$ po linijama $POUT_{7...0}$ periferije *PER* šalje i upisuje u periferiju. Ovakvin načinom prenosa podataka je omogućeno da se istovremeno prenosi tekući podatak iz registra $DRAUX_{7...0}$ u periferiju i sledeći podatak iz memorije u registar $DR_{7...0}$. Pri tome se sledeći podatak prenosi iz registra $DR_{7...0}$ u registar $DRAUX_{7...0}$, tek pošto je tekući podatak prenet iz registra $DRAUX_{7...0}$ u periferiju.

Registar $DR_{15...0}$ je 8-mo razredni registar u koji se generisanjem aktivne vrednosti jednog od signala **DRin** i **ldDR** upisuje sadržaj sa izlaza multipleksa MP1. Pri aktivnoj vrednosti signala **DRin** i neaktivnoj vrednosti signala **ldDR** sadržaj sa linija podataka $DBUS_{7...0}$ magistrale *BUS* se propušta kroz multiplekser MP1 i upisuje u registar $DR_{7...0}$. Pri aktivnoj vrednosti signala **ldDR** sadržaj sa izlaza registra $DRAUX_{7...0}$ se propušta kroz multiplekser MP1 i upisuje u registar $DR_{7...0}$. Signal **DRin** je aktivan kada se generišu aktivne vrednosti signala **rdMKTR** i **fcMKTR**. Signal **rdMKTR** je aktivan kada kontroler kao gazda realizuje ciklus čitanja na magistrali, dok je signal **fcMKTR** aktivan kada se na upravljačkoj liniji magistrale **FCBUS** pojavi aktivna vrednost kao indikacija od memorije kao sluge da je pročitani podatak raspoloživ na linijama podataka $DBUS_{7...0}$ magistrale. Sadržaj registra $DR_{7...0}$ se pri aktivnoj vrednosti signala **eDR** propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale. Pored toga sadržaj registra $DR_{7...0}$ se vodi na ulaze multipleksa MP2.

Multiplekser MP1 se sastoji od 16 multipleksa sa dva ulaza. Na ulaze 0 i 1 multipleksa MP1 dovodi se sadržaj sa linija podataka $DBUS_{7...0}$ magistrale i sadržaj sa izlaza registra $DRAUX_{7...0}$, respektivno. Selektovanje sadržaja se obavlja signalom **ldDR**.

Registar $DRAUX_{7...0}$ je 8-mo razredni registar u koji se, generisanjem aktivne vrednosti jednog od signala **DRAUXin** i **ldDRAUX**, upisuje sadržaj sa izlaza multipleksa MP2. Pri aktivnoj vrednosti signala **DRAUXin** i neaktivnoj vrednosti signala **ldDRAUX** sadržaj registra $DR_{15...0}$ se propušta kroz multiplekser MP2 i upisuje u registar $DRAUX_{7...0}$. Pri aktivnoj vrednosti signala **ldDRAUX** sadržaj sa ulaznih linija podataka $PIN_{7...0}$ periferije se propušta kroz multiplekser MP2 i upisuje u registar $DRAUX_{7...0}$. Sadržaj registra $DRAUX_{7...0}$ se po izlaznim linijama podataka $POUT_{7...0}$ šalje u periferiju. Pored toga sadržaj registra $DRAUX_{7...0}$ se vodi na ulaze multipleksa MP1.

Multiplexer MP2 se sastoji od 8 multipleksera sa dva ulaza. Na ulaze 0 i 1 multipleksera MP2 dovodi se sadržaj sa izlaza registra DR_{7...0} i sadržaj sa ulaznih linija podataka PIN_{7...0} periferije, respektivno. Selektovanje sadržaja se obavlja signalom **ldDRAUX**.

Registar CR_{3...0} je četvororazredni upravljački registar koji sadrži bitove paketski prenos, start, u/i prenos i generisanje prekida, respektivno. U registar CR_{3...0} se, generisanjem aktivne vrednosti signala **CRin** upisuje sadržaj sa linija podataka DBUS_{3...0} magistrale. Sadržaj registra CR_{3...0} proširen nulama do dužine 8 bitova se pri aktivnoj vrednosti signala **CRout** propušta kroz bafere sa tri stanja na linije podataka DBUS_{7...0} magistrale.

Registar SR₀ je jednorazredni upravljački registar koji sadrži bit *prenos završen*. U registar SR₀ se, generisanjem aktivne vrednosti signala **SRin**, upisuje sadržaj sa linija podataka DBUS₀ magistrale. Sadržaj registra SR₀ proširen nulama do dužine 8 bitova se pri aktivnoj vrednosti signala **SRout** propušta kroz bafere sa tri stanja na linije podataka DBUS_{7...0} magistrale.

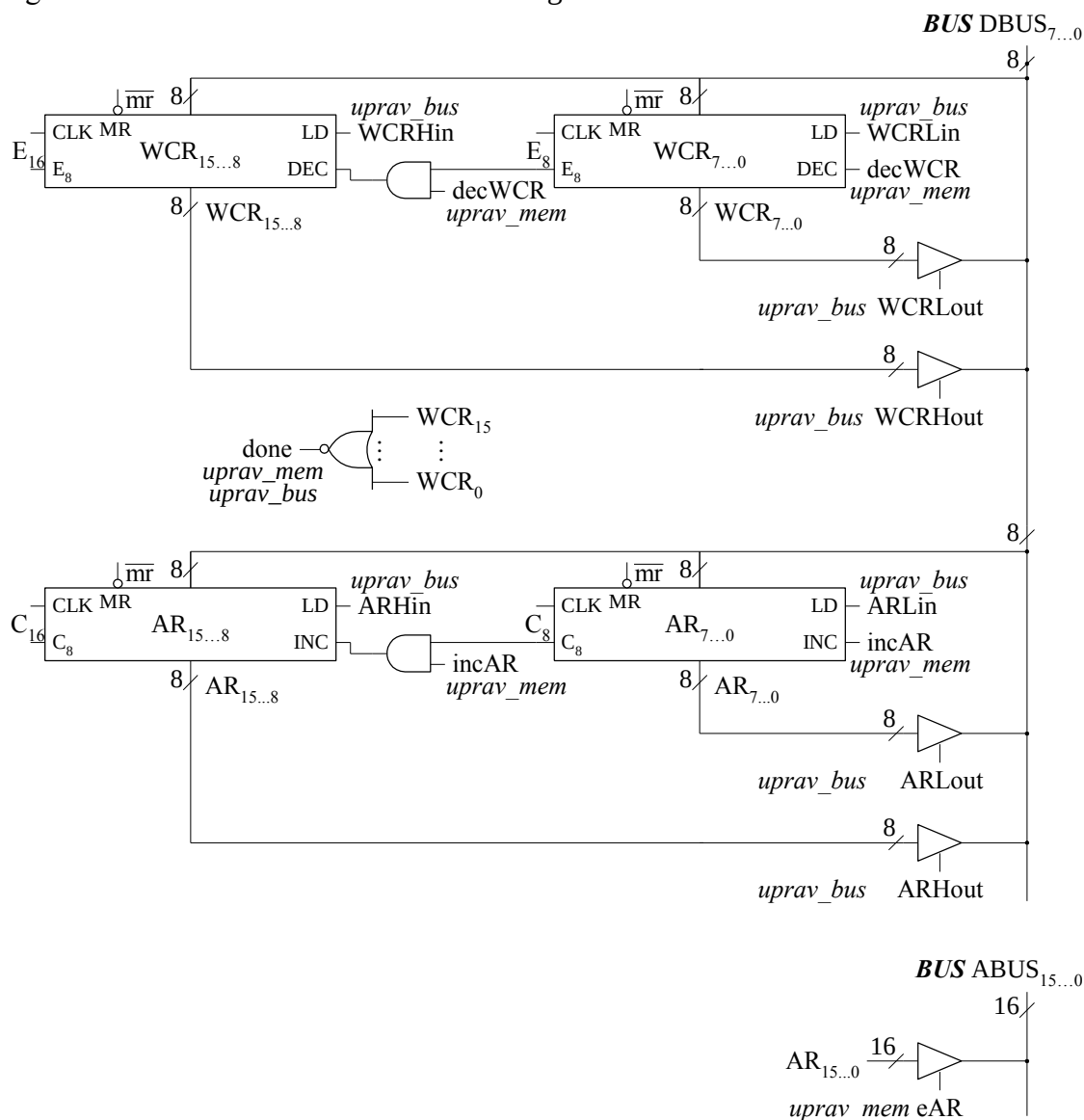
Neaktivna i aktivna vrednost razreda SR₀ ukazuju da li je prenos bloka podataka u toku ili je završen, respektivno. Veličina bloka podataka koje još treba preneti se zadaje programskim putem upisivanjem odgovarajućeg sadržaja u registar WCR_{15...0}. Pri prenosu jedne reči bloka podataka sadržaj registra WCR_{15...0} se dekrementira. Sadržaj sve nule registra WCR_{15...0}, na šta ukazuje aktivna vrednost signala **done**, je indikacija da je prenos završen.

Ukoliko se pri startovanju kontrolera periferije utvrdi da je signal **done** neaktivan jer je zadata veličina bloka podataka za prenos veća od nule, razred SR₀ se signalom **clSRbus** postavlja na neaktivnu vrednost. Po završetku prenosa bloka podataka, na šta ukazuje aktivna vrednost signala **done**, razred SR₀ se signalom **stSR0per** postavlja na aktivnu vrednost. Ukoliko se pri startovanju kontrolera periferije utvrdi da je signal **done** aktivan jer je zadata veličina bloka podataka za prenos nula, nema prenosa podataka i razred SR₀ se signalom **stSR0bus** postavlja na aktivnu vrednost. Pri čitanju registra SR₀ razred SR₀ se signalom **clSR0bus** postavlja na neaktivnu vrednost.

Flip-flopovi RDDR i WRDR se koriste za neophodnu sinhronizaciju pri prenosu podataka iz periferije u memoriju i obrnuto. Flip-flop RDDR se koristi pri prenosu podataka iz periferije u memoriju. Njegova aktivna vrednost je indikacija da novi podatak može da se prenese iz registra DRAUX_{7...0} u registar DR_{7...0}, a neaktivna da tekući podatak još uvek nije prenet iz registra DR_{7...0} u memoriju i da novi podatak ne može da se prenese iz registra DRAUX_{7...0} u registar DR_{7...0}. S toga se pri startovanju kontrolera periferije za prenos iz periferije u memoriju flip-flop RDDR signalom **stRDDRbus** postavlja na aktivnu vrednost. Posle toga se po prenosu podatka iz registra DRAUX_{7...0} u registar DR_{7...0} flip-flop RDDR signalom **clRDDRper** postavlja na neaktivnu, a po prenosu podatka iz registra DR_{7...0} u memoriju signalom **stRDDRmem** postavlja na aktivnu vrednost. Flip-flop WRDR se koristi pri prenosu podataka iz memorije u periferiju. Njegova aktivna vrednost je indikacija da se u registru DR_{7...0} nalazi podatak i da sadržaj registra DR_{7...0} može da se prenese u registar DRAUX_{7...0}, a neaktivna da novi podatak još uvek nije prenet iz memorije u registar DR_{7...0} i da sadržaj registra DR_{7...0} ne može da se prenese u registar DRAUX_{7...0}. S toga se pri startovanju kontrolera periferije za prenos iz memorije u periferiju flip-flop WRDR signalom **clWRDRbus** postavlja na neaktivna vrednost. Posle toga se po prenosu podatka iz memorije u registar DR_{7...0} flip-flop WRDR signalom **stWRDRmem** postavlja na aktivnu vrednost, a po prenosu podatka iz registra DR_{7...0} u registar DRAUX_{7...0} signalom **clWRDRper** postavlja na neaktivnu vrednost.

Brojač $WCR_{15...0}$ je 16-to razredni brojač u kome se čuva broj reči bloka podataka koje treba preneti (slika 80). Veličina bloka podataka koji treba preneti se u brojač $WCR_{15...0}$ upisuje programskim putem u dva ciklusa na magistrali. Signalom **WCRHin** se u 8 starijih razreda brojača $WCR_{15...8}$ upisuje sadržaj sa linija podataka $DBUS_{7...0}$ magistrale, dok se signalom **WCRLin** u 8 mlađih razreda brojača $WCR_{7...0}$ upisuje sadržaj sa linija podataka $DBUS_{7...0}$ magistrale. Sadržaj brojača $WCR_{15...0}$ se čita programskim putem u dva ciklusa na magistrali. Signalom **WCRHout** se 8 starijih razreda brojača $WCR_{15...8}$ propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale, dok se signalom **WCRLout** 8 mlađih razreda brojača $WCR_{7...0}$ propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale.

Pri prenosu svake reči bloka podataka generiše se aktivna vrednost signala **decWCR** čime se sadržaj brojača $WCR_{15...0}$ dekrementira. Sadržaj brojača $WCR_{15...0}$ različit od nule je indikacija da blok podatak nije još uvek prenet, dok je sadržaj nula indikacija da je kompletan blok podataka prenet. Na osnovu toga da li je sadržaja registra $WCR_{15...0}$ različit od nule ili je nula generišu se neaktivna i aktivna vrednost signala **done**.



Slika 80 Blok registri (drugi deo)

Brojač $AR_{15...0}$ je 16-to razredni brojač u kome se čuva adresa sledeće reči bloka podataka koja se prenosi. Početna adresa se u brojač $AR_{15...0}$ upisuje programskim putem u dva ciklusa na magistrali. Signalom **ARHin** se u 8 starijih razreda brojača $AR_{15...8}$ upisuje sadržaj sa linija podataka $DBUS_{7...0}$ magistrale, dok se signalom **ARLin** u 8 mlađih razreda brojača $AR_{7...0}$ upisuje sadržaj sa linija podataka $DBUS_{7...0}$ magistrale. Sadržaj brojača $AR_{15...0}$ se čita programskim putem u dva ciklusa na magistrali. Signalom **ARHout** se 8 starijih razreda brojača $AR_{15...8}$ propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale, dok se signalom **ARLout** u 8 mlađih razreda brojača $AR_{7...0}$ propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale.

Prilikom prenosa svake reči bloka podataka sadržaj brojača $AR_{15...0}$ se najpre aktivnom vrednošću signala **eAR** propušta kroz bafere sa tri stanja na adresna linije $ABUS_{15...0}$ magistrale i zatim signalom **incAR** inkrementira.

6.3.1.2 Blok interfejs

Blok *interfejs* (slika 81) sadrži kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je kontroler sluga, za realizaciju ciklusa na magistrali u kojima je kontroler gazda, za dobijanje dozvole korišćenja magistrale i za generisanje prekida.

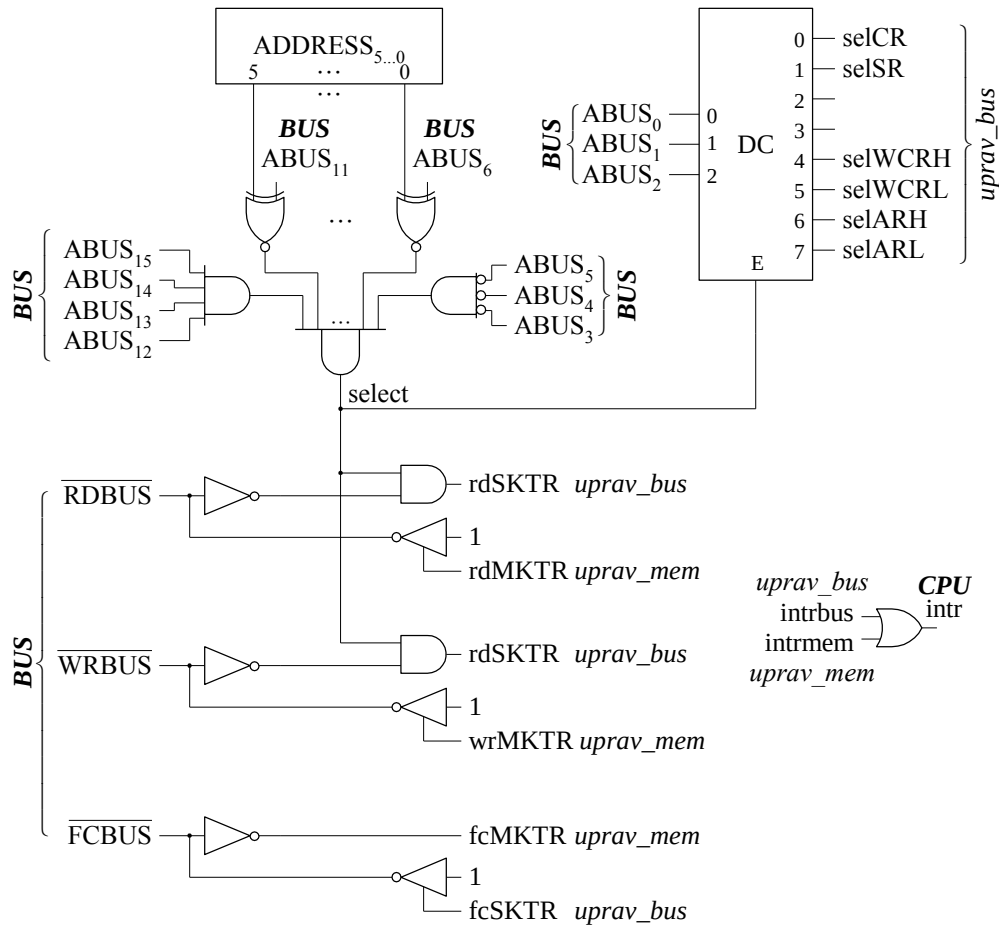
Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je kontroler sluga formiraju signale **rdSKTR** i **wrSKTR** upravljačke jedinice *uprav_bus* i **FCBUS** magistrale **BUS** (slika 71). Signali **rdSKTR** i **wrSKTR** se formiraju na osnovu signala $ABUS_{15...0}$ sa adresnih linija magistrale i **RDBUS** i **WRBUS** sa upravljačkih linija magistrale. Signal **FCBUS** se formira na osnovu signala **fcSKTR** upravljačke jedinice *uprav_bus*.

Pri realizaciji ciklusa čitanja na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije $ABUS_{15...0}$ i upravljačku liniju **RDBUS** magistrale i na njih izbacuje adresu i aktivnu vrednost signala čitanja, respektivno, čime se u kontroleru kao slugi startuje čitanje adresiranog registra. Kontroler po završenom čitanju otvara bafere sa tri stanja za linije podataka $DBUS_{7...0}$ i upravljačku liniju **FCBUS** magistrale i na njih izbacuje sadržaj adresiranog registra i aktivnu vrednost signala završetka operacije čitanja u kontroleru. Procesor prihvata sadržaj sa linija podataka i zatvara bafere sa tri stanja za adresne linije $ABUS_{15...0}$ i upravljačku liniju **RDBUS** magistrale, dok kontroler zatvara bafere sa tri stanja za linije podataka $DBUS_{7...0}$ i upravljačku liniju **FCBUS** magistrale.

Pri realizaciji ciklusa upisa na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije $ABUS_{15...0}$, linije podataka $DBUS_{7...0}$ i upravljačku liniju **WRBUS** magistrale i na njih izbacuje adresu, podatak i aktivnu vrednost signala upisa, respektivno, čime se u kontroleru kao slugi startuje upis u adresirani registar. Kontroler po završenom upisu otvara bafer sa tri stanja za upravljačku liniju **FCBUS** magistrale i na nju izbacuje aktivnu vrednost signala završetka operacije upisa u kontroleru. Procesor zatvara bafere sa tri stanja za adresne linije $ABUS_{15...0}$, linije podataka $DBUS_{7...0}$ i upravljačku liniju **WRBUS** magistrale, dok kontroler zatvara bafer sa tri stanja za upravljačku liniju **FCBUS** magistrale.

Signali **rdKTR** i **wrKTR** imaju vrednosti upravljačkih signala **RDBUS** i **WRBUS** magistrale, respektivno, kada je aktivna vrednost signala **select** i neaktivne vrednosti kada je neaktivna vrednost signala **select**. Signal **rdKTR** je kao i signal **RDBUS** aktivan za vreme

operacije čitanja nekog registra kontrolera, dok je signal **wrKTR** kao i signal **WRBUS** aktivan za vreme operacije upisa u neki od registara kontrolera.



Slika 81 Blok interfejs

Signal **select** ima aktivnu vrednost ukoliko se na adresnim linijama $ABUS_{15...0}$ magistrale nalazi adresa iz opsega adresa dodeljenih registrima kontrolera periferije **KP**. Celokupan opseg adresa od 0000h do FFFFh podeljen je na opseg adresa od 0000h do EFFFh, koje su dodeljene memoriji **MEM**, i opseg adresa od F000h do FFFFh, koje su dodeljene registrima po svim kontrolerima periferija. Opseg adresa koje su dodeljene registrima po kontrolerima periferija je predviđen za adresiranje registara u najviše 64 kontrolera i to najviše 64 registra unutar određenog kontrolera. Pri tome prilikom adresiranja nekog od registara kontrolera sadržaj na adresnim linijama $ABUS_{15...0}$ magistrale ima sledeću strukturu: bitovi $ABUS_{15...12}$ su sve jedinice, bitovi $ABUS_{11...6}$ određuju broj kontrolera i bitovi $ABUS_{5...0}$ adresu registra unutar kontrolera. Broj kontrolera periferije za određeni kontroler periferije se postavlja mikroprekidačima na jednu od vrednosti u opsegu 0 do 63. Registrima $CR_{7...0}$, $SR_{7...0}$, $WCRH_{7...0}$, $WCRL_{7...0}$, $ARH_{7...0}$ i $ARL_{7...0}$ su unutar opsega adresa datog kontrolera dodeljene adrese 0, 1, 4, 5, 6 i 7, pa bitovi $ABUS_{5...3}$ mora da budu nule dok se njihovo pojedinačno adresiranje realizuje bitovima $ABUS_{2...0}$.

Na osnovu usvojene strukture adresa signal **select** ima aktivnu vrednost ukoliko su na adresnim linijama $ABUS_{15...12}$ magistrale sve jedinice, na adresnim linijama $ABUS_{11...6}$ se nalazi vrednost koja odgovara vrednosti postavljenoj mikroprekidačima i na adresnim linijama $ABUS_{5...3}$ sve nule. Signali **selCR**, **selSR**, **selWCRH**, **selWCRL**, **selARH** i **selARL** se dobijaju na izlazima 0, 1, 4, 5, 6 i 7 dekodera DC na osnovu vrednosti signala $ABUS_{2...0}$ i **select** sa ulaza 2, 1, 0 i E, respektivno. Pri neaktivnoj vrednosti signala **select** i signali **selCR**,

selSR, **selWCRH**, **selWCRL**, **selARH** i **selARL** su neaktivni. Pri aktivnoj vrednosti signala **select** jedan od signala **selCR**, **selSR**, **selWCRH**, **selWCRL**, **selARH** i **selARL** postaje aktivan i to signal određen binarnom vrednošću signala **ABUS_{2..0}**.

Signal **select** ima neaktivnu vrednost ukoliko se na adresnim linijama **ABUS_{15..0}** magistrale ne nalazi adresa iz opsega adresa dodeljenih registrima kontrolera periferije, pri čemu to može da bude ili adresa memorijske lokacije ili registra iz nekog drugog kontrolera periferije. Tada se formiraju neaktivne vrednosti signala **rdSKTR** i **wrSKTR** bez obzira na vrednosti signala **RDBUS** i **WRBUS**, respektivno.

Kombinaciona mreža za generisanje upravljačkog signala **FCBUS** magistrale generiše ovaj signal na osnovu signala **fcSKTR**. Pri neaktivnoj vrednosti signala **fcSKTR** na liniji signala **FCBUS** je stanje visoke impedance, dok je pri aktivnoj vrednosti signala **fcSKTR** na liniji signala **FCBUS** aktivna vrednost. Signal **fcSKTR** ima neaktivnu ili aktivnu vrednost u zavisnosti od toga da li je u kontroleru operacija čitanja ili upisa u toku ili je završena, respektivno.

Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je kontroler gazda formiraju signale **RDBUS** i **WRBUS** magistrale **BUS** i **fcMKTR** upravljačke jedinice *uprav_mem* (slika 71). Signali **RDBUS** i **WRBUS** se formiraju na osnovu signala **rdMKTR** i **wrMKTR** upravljačke jedinice *uprav_mem*. Signal **fcMKTR** se formira na osnovu signala **FCBUS** magistrale **BUS**.

Pri realizaciji ciklusa čitanja na magistrali kontroler kao gazda najpre procesoru šalje zahtev korišćenja magistrale. Pri realizaciji ciklusa čitanja na magistrali kontroler kao gazda otvara bafere sa tri stanja za adresne linije **ABUS_{15..0}** i upravljačku liniju **RDBUS** magistrale i na njih izbacuje adresu i aktivnu vrednost signala čitanja, respektivno, čime se u memoriji kao slugi startuje čitanje adresirane lokacije. Memorija po završenom čitanju otvara bafere sa tri stanja za linije podataka **DBUS_{7..0}** i upravljačku liniju **FCBUS** magistrale i na njih izbacuje sadržaj adresirane lokacije i aktivnu vrednost signala završetka operacije čitanja u memoriji. Kontroler prihvata sadržaj sa linija podataka i zatvara bafere sa tri stanja za adresne linije **ABUS_{15..0}** i upravljačku liniju **RDBUS** magistrale, dok memorija zatvara bafere sa tri stanja za linije podataka **DBUS_{7..0}** i upravljačku liniju **FCBUS** magistrale.

Pri realizaciji ciklusa upisa na magistrali kontroler kao gazda najpre procesoru šalje zahtev korišćenja magistrale. Pri realizaciji ciklusa upisa na magistrali kontroler kao gazda otvara bafere sa tri stanja za adresne linije **ABUS_{15..0}**, linije podataka **DBUS_{7..0}** i upravljačku liniju **WRBUS** magistrale i na njih izbacuje adresu, podatak i aktivnu vrednost signala upisa, respektivno, čime se u memoriji kao slugi startuje upis u adresiranu lokaciju. Memorija po završenom upisu otvara bafer sa tri stanja za upravljačku liniju **FCBUS** magistrale i na nju izbacuje aktivnu vrednost signala završetka operacije upisa u memoriji. Kontroler zatvara bafere sa tri stanja za adresne linije **ABUS_{15..0}**, linije podataka **DBUS_{7..0}** i upravljačku liniju **WRBUS** magistrale, dok kontroler zatvar bafer sa tri stanja za upravljačku liniju **FCBUS** magistrale.

Kombinacione mreže za generisanje upravljačkih signala **RDBUS** i **WRBUS** magistrale generišu ove signale na osnovu signala **rdMKTR** i **wrMKTR**, respektivno. Pri neaktivnoj vrednosti signala **rdMKTR** na liniji signala **RDBUS** je stanje visoke impedance, dok je pri aktivnoj vrednosti signala **rdMKTR** na liniji signala **RDBUS** aktivna vrednost. Signal

rdMKTR ima aktivnu ili neaktivnu vrednost u zavisnosti od toga da li je u memoriji operacija čitanja u toku ili je završena, respektivno. Pri neaktivnoj vrednosti signala **wrMKTR** na liniji signala $\overline{\text{WRBUS}}$ je stanje visoke impedance, dok je pri aktivnoj vrednosti signala **wrMKTR** na liniji signala $\overline{\text{WRBUS}}$ aktivna vrednost. Signal **wrMKTR** ima aktivnu ili neaktivnu vrednost u zavisnosti od toga da li je u memoriji operacija upisa u toku ili je završena, respektivno.

Signal **fcMKTR** ima vrednost upravljačkog signala $\overline{\text{FCBUS}}$ magistrale. Signal **fcMKTR** je kao i signal $\overline{\text{FCBUS}}$ aktivan za vreme operacije čitanja neke lokacije memorije ili operacije upisa u neku lokaciju memorije.

Pri realizaciji ciklusa čitanja ili upisa na magistrali kontroler kao gazda najpre procesoru šalje aktivnu vrednost signal zahteva korišćenja magistrale **hreq** i tek po dobijanju aktivne vrednosti signala dozvole korišćenja magistrale **hack** od procesora kreće sa realizacijom ciklusa na magistrali. Ukoliko je zadat pojedinačni režim prenosa, kontroler po završetku prenosa jednog podatka ukida zahtev korišćenja magistrale postavljanjem signala **hreq** na neaktivnu vrednost, a procesor ukida dozvolu korišćenja magistrale postavljanjem signala **hack** na neaktivnu vrednost. Ovakva razmena signala **hreq** i **hack** između kontrolera i procesora se ponavlja pri prenosu svakog podatka bloka podataka. Ukoliko je zadat paketski režim prenosa, kontroler ne ukida aktivnu vrednost signala zahteva korišćenja magistrale **hreq** dok traje prenos celog bloka podataka, pa procesor drži aktivnu vrednost signala dozvole korišćenja magistrale **hack** sve vreme dok traje prenos bloka podataka. Tek po završetku prenosa celog bloka podataka kontroler ukida zahtev korišćenja magistrale postavljanjem signala **hreq** na neaktivnu vrednost, a procesor ukida dozvolu korišćenja magistrale postavljanjem signala **hack** na neaktivnu vrednost. Signal **hreq** koji generiše upravljačka jedinica *uprav_mem* se prosleđuje procesoru **CPU**, dok se signal **hack** koji generiše procesor **CPU** prosleđuje upravljačkoj jedinici *uprav_mem*.

Kada kontroler treba u procesoru da izazove prekid, generiše se aktivna vrednost signala **intr**. Signala **intr** ima aktivnu vrednost ukoliko ili signal **intrbus** upravljačke jedinice *uprav_bus* ili signal **intrmem** upravljačke jedinice *uprav_mem* ima aktivnu vrednost.

6.3.2 Upravljačka jedinica

Upravljačka jedinica *uprav_jed* se sastoji iz tri dela:

- upravljačke jedinice magistrale *uprav_bus*,
- upravljačke jedinice periferije *uprav_per* i
- upravljačke jedinice memorije *uprav_mem*.

Upravljačke jedinice *uprav_bus*, *uprav_per* i *uprav_mem* rade istovremeno i omogućuju paralelan rad kontrolera periferije **KP** kao sluge sa magistralom **BUS**, kontrolera periferije **KP** sa periferijom **PER** i kontrolera periferije **KP** kao gazde sa magistralom **BUS**.

Upravljačka jedinica *uprav_bus* omogućuje da procesor **CPU** kao gazda u kontroleru kao slugi realizuje upisivanje sadržaja u registre i čitanje sadržaja iz registara.

Upisivanjem odgovarajućeg sadržaja u upravljački registar startuje se kontroler za prenos podataka iz periferije u memoriju ili iz memorije u periferiju.

Prilikom startovanja kontrolera za prenos podataka iz periferije u memoriju aktiviraju se upravljačke jedinice *uprav_per* i *uprav_mem*. Upravljačka jedinica *uprav_per* prvo startuje periferiju da pročita podatak, a zatim podatak prenosi iz periferije, najpre, u pomoćni registar podatka kontrolera, a potom u registar podatka kontrolera. Tada upravljačka jedinica

uprav_mem podatak prenosi iz registra podatka kontrolera u memoriju. Prenošnje podatka iz registra podatka kontrolera u memoriju od strane upravljačke jedinice *uprav_mem* i čitanje novog podatka iz periferije i prenošenje u pomoćni registar podatka kontrolera od strane upravljačke jedinice *uprav_per* su preklapljeni. Upravljačka jedinica *uprav_per* novi podatak prenosi iz pomoćnog registra podatka kontrolera u registar podatka kontrolera tek pošto upravljačka jedinica *uprav_mem* prethodni podatak prenese iz registra podatka kontrolera u memoriju. Prenošnje podataka iz periferije u memoriju traje dok se ne prenesu sve reči bloka podataka na šta ukazuje sadržaj nula brojača reči kontrolera. Po prenošenju zadnjeg podatka iz registra podatka kontrolera u memoriju bit spremnost statusnog registra se postavlja na aktivnu vrednost i generiše prekid ukoliko je pri startovanju zadat režim rada sa generisanjem prekida.

Prilikom startovanja kontrolera periferije za prenos podataka iz memorije u periferiju aktiviraju se upravljačke jedinice *uprav_mem* i *uprav_per*. Upravljačka jedinica *uprav_mem* najpre prenosi podatak iz memorije u registar podatka kontrolera. Tada upravljačka jedinica *uprav_per* podatak prenosi iz registra podatka kontrolera u pomoćni registar podatka kontrolera i startuje upis podatka u periferiju. Upis podatka iz pomoćnog registra podatka u periferiju od strane upravljačke jedinice *uprav_per* i prenošenje novog podatka iz memorije u registar podatka kontrolera od strane upravljačke jedinice *uprav_jed_mem* su preklapljeni. Upravljačka jedinica *uprav_jed_per* novi podatak prenosi iz registra podatka kontrolera u pomoćni registar podatka kontrolera tek pošto prethodni podatak iz pomoćnog registra podatka kontrolera upiše u periferiju. Prenošnje podataka iz memorije u periferiju traje dok se ne prenesu sve reči bloka podataka na šta ukazuje sadržaj nula brojača reči kontrolera. Po prenošenju zadnjeg podatka iz memorije u periferiju bit spremnost statusnog registra se postavlja na aktivnu vrednost i generiše prekid ukoliko je pri startovanju zadat režim rada sa generisanjem prekida.

Struktura i opis upravljačkih jedinica *uprav_bus*, *uprav_per* i *uprav_mem* se daju u daljem tekstu.

6.3.2.1 Upravljačka jedinica magistrale

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

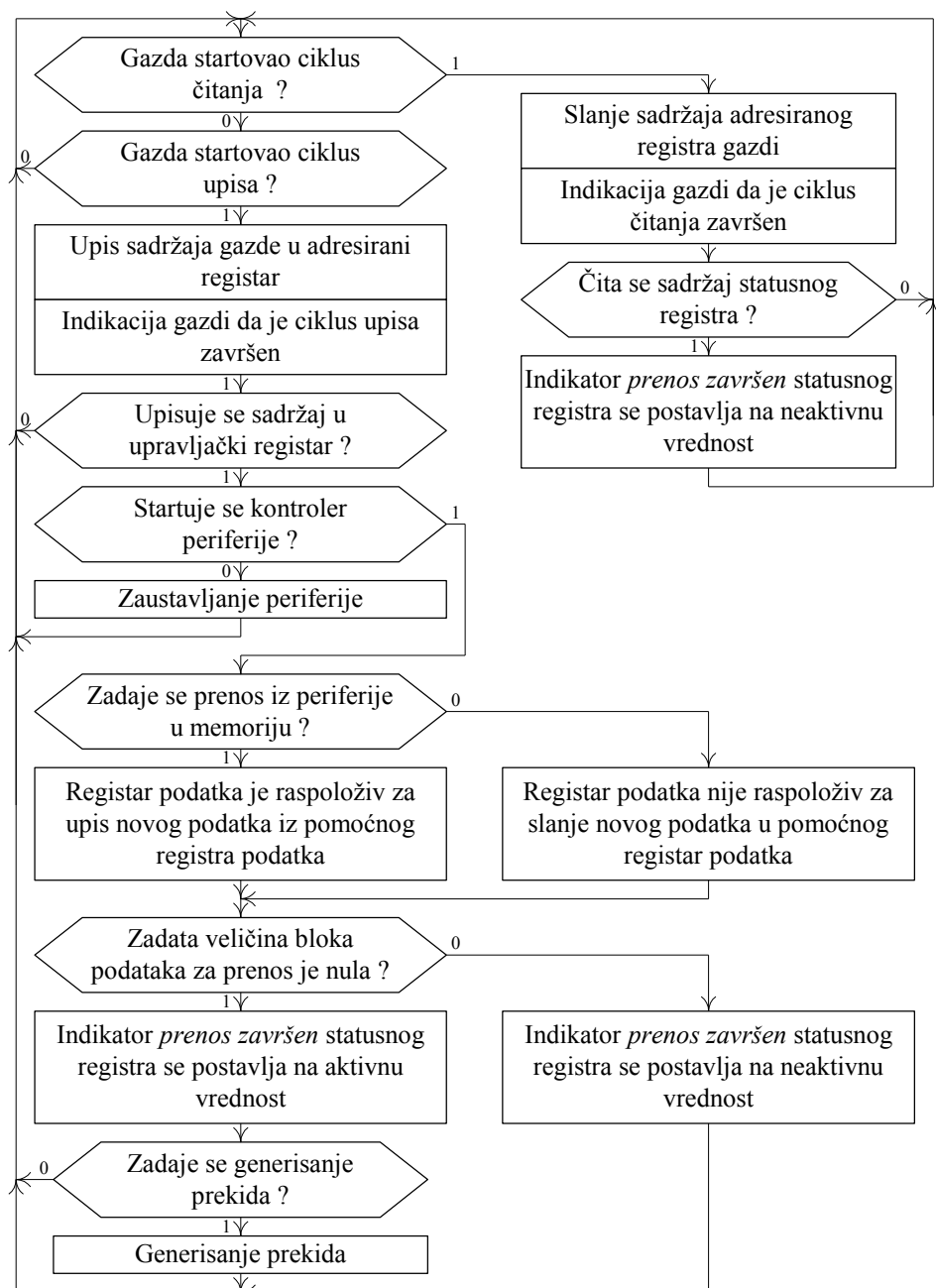
6.3.2.1.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 82). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

U dijagrama toka operacija stalno se vrši provera da li je procesor startovao u kontroleru ciklus čitanja ili ciklus upisa, pri čemu se procesor ponaša kao gazda, a kontroler kao sluga. Ako nije startovan ni ciklus čitanja ni ciklus upisa nema izvršavanja mikrooperacija. Ako je startovan jedan od ova dva ciklusa izvršavaju se mikrooperacije saglasno ciklusu koji je startovan.

Ako je startovan ciklus čitanja, procesoru se kao gazdi šalje sadržaj adresiranog registra kontrolera i indikacija da je kontroler kao sluga završio čitanje. Pored toga, u slučaju kada se čita sadržaj statusnog registra, razred *prenos završen* statusnog registra se postavlja na neaktivnu vrednost.

Ako je startovan ciklus upisa, u adresirani registar kontrolera se upisuje sadržaj koji procesor kao gazda šalje i gazdi šalje indikacija da je kontroler kao sluga završio upis.



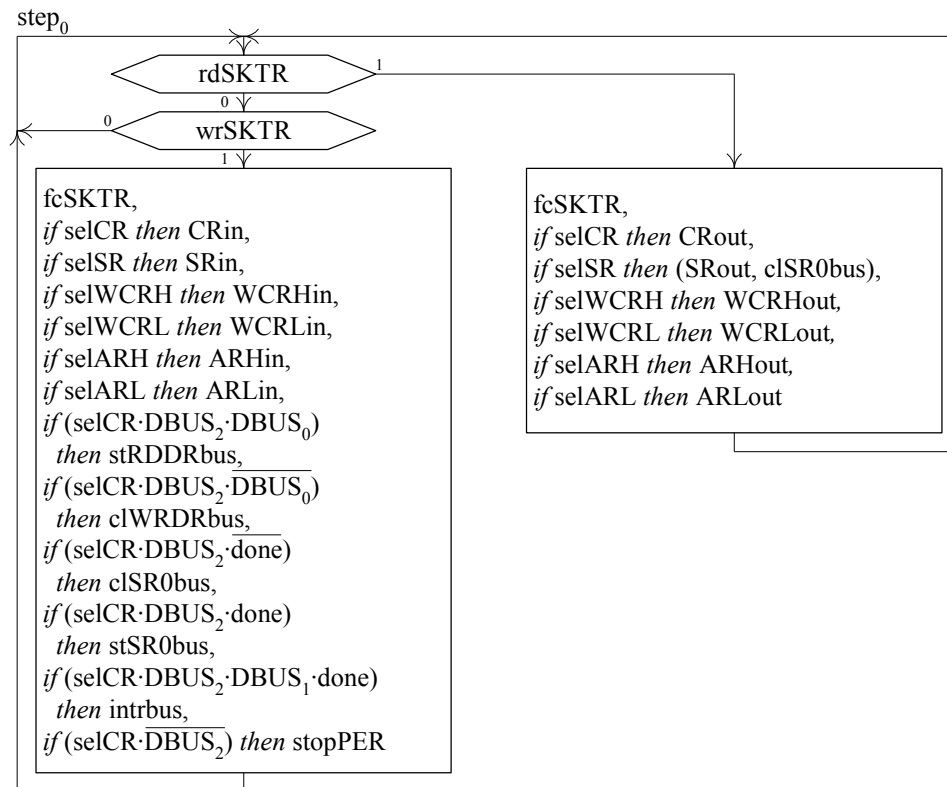
Slika 82 Dijagram toka operacija

U slučaju kada se sadržaj upisuje u upravljački registar, mikrooperacije koje se izvršavaju zavise od toga da li se kontroler periferije zaustavlja ili startuje. Ukoliko se vrši zaustavljanje kontrolera, zaustavlja se i periferija. Ukoliko se vrši startovanje, izvršavanje nekih mikrooperacija zavisi od toga da li je zadata veličina bloka podataka za prenos različita od nule ili je nula. Ukoliko je zadata veličina bloka podataka različita od nule, razred *prenos završen* statusnog registra se postavlja na neaktivnu vrednost kao indikacija da je prenos bloka podataka u toku. Ukoliko je zadata veličina bloka podataka nula, razred *prenos završen* statusnog registra se postavlja na aktivnu vrednost kao indikacija da je prenos bloka podataka završen i generiše prekid ukoliko se zadaje režim rada sa generisanjem prekida. Pored toga ukoliko se vrši startovanje za prenos iz periferije u memoriju, postavlja se interna indikacija da je registar podatka raspoloživ da se u njega prenese podatak iz pomoćnog registra podatka,

dok se, ukoliko se vrši startovanje za prenos iz memorije u periferiju postavlja interna indikacija da u registar podatka nije upisan podatak iz memorije i da on nije raspoloživ da se iz njega prenese novi podatak u pomoćni registar podatka.

6.3.2.1.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 72) i dat u obliku dijagrama toka upravljačkih signala (slika 73) i sekvence upravljačkih signala (tabela 26).



Slika 83 Dijagram toka upravljačkih signala

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 73). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova koji određuju grananja u algoritmu.

U sekvenci upravljačkih signala se koriste iskazi za signale (tabela 26). Iskazi za signale su oblika

if uslov then signali

Ovi iskazi sadrže koji sadrže uslov i spisak upravljačkih signala blokova operacione jedinice **oper** i određuje koji signali i pod kojim uslovima treba da budu generisani.

Objašnjenja vezana za generisanje upravljačkih signala su data zajednički za dijagram toka upravljačkih signala (slika 73) i sekvencu upravljačkih signala (tabela 26) i to u okviru sekvence upravljačkih signala.

Tabela 28 Sekvenca upravljačkih signala

! U koraku step₀ se ostaje sve vreme. U ovom koraku se ne generiše aktivna vrednost ni jednog od upravljačkih signala sve dok su oba signala **rdSKTR** i **wrSKTR** bloka *interfejs* neaktivni. Signal **rdSKTR** postaje aktivan kada processor **CPU** prebaci adresne linije ABUS_{15.0} magistrale iz stanja

visoke impedanse na vrednost adrese nekog od programski dostupnih registara $CR_{2...0}$, SR_0 , $WCRH_{7...0}$, $WCRL_{7...0}$, $ARH_{7...0}$ ili $ARL_{7...0}$ bloka *registri* i upravljačku liniju **RDBUS** magistrale iz stanja visoke impedanse na aktivnu vrednost, čime započinje ciklus čitanja. Signal **wrSKTR** postaje aktivan kada processor prebaci adresne linije $ABUS_{15...0}$ i linije podataka $DBUS_{7...0}$ magistrale iz stanja visoke impedanse na vrednost adrese registra i sadržaja za upis u neki od programski dostupnih registara $CR_{2...0}$, SR_0 , $WCRH_{7...0}$, $WCRL_{7...0}$, $ARH_{7...0}$ ili $ARL_{7...0}$, respektivno, i upravljačku liniju **WRBUS** magistrale iz stanja visoke impedanse na aktivnu vrednost, čime započinje ciklus upisa. Pri aktivnoj vrednosti jednog od signala **rdSKTR** i **wrSKTR** generiše se formira se aktivna vrednost signala **fcSKTR** bloka *interfejs*. Signalom **fcSKTR** se obezbeđuje da upravljačka linija **FCBUS** magistrale pređe iz stanja visoke impedanse na aktivnu vrednost. U oba slučaja se, u zavisnosti od toga koji je od registara $CR_{2...0}$, SR_0 , $WCRH_{7...0}$, $WCRL_{7...0}$, $ARH_{7...0}$ ili $ARL_{7...0}$ adresiran sadržajem na adresnim linijama $ABUS_{15...0}$ magistrale, generiše aktivna vrednost jednog od signala **selCR**, **selSR**, **selWCRH**, **selWCRL**, **selARH** ili **selARL**, respektivno, bloka *interfejs*. !

! Pri aktivnoj vrednosti signala **rdSKTR**, a u zavisnosti od toga koji od signala **selCR**, **selSR**, **selWCRH**, **selWCRL**, **selARH** ili **selARL** bloka *interfejs* ima aktivnu vrednost, generiše se aktivna vrednost jednog od signala **CRout**, **SRout**, **WCRHout**, **WCRLout**, **ARHout** ili **ARLout** bloka *registri*, respektivno. Signalima **CRout**, **SRout**, **WCRHout**, **WCRLout**, **ARHout** ili **ARLout** se obezbeđuje da linije podataka $DBUS_{7...0}$ magistrale pređu iz stanja visoke impedanse na vrednost sadržaja jednog od registara $CR_{2...0}$, SR_0 , $WCRH_{7...0}$, $WCRL_{7...0}$, $ARH_{7...0}$ ili $ARL_{7...0}$, respektivno. Kada signal **rdSKTR** postane neaktivan i signali **fcSKTR**, **CRout**, **SRout**, **WCRHout**, **WCRLout**, **ARHout** ili **ARLout** postanu neaktivni, pa se upravljačka linija **FCBUS** magistrale i linije podataka $DBUS_{7...0}$ magistrale vraćaju u stanje visoke impedanse. !

! Pri aktivnim vrednostima signala **rdSKTR** i **selSR**, što znači da se programskim putem čita sadržaj registra SR_0 , generiše se signal **clSR0bus** bloka *registri*. Signalom **clSR0bus** se razred SR_0 statusnog registra SR_0 postavlja na neaktivnu vrednost, što je indikacija prenos novog bloka podataka nije završen. !

! Pri aktivnoj vrednosti signala **wrSKTR**, a u zavisnosti od toga koji od signala **selCR**, **selSR**, **selWCRH**, **selWCRL**, **selARH** ili **selARL** ima aktivnu vrednost, generiše se aktivna vrednost jednog od signala **CRin**, **SRin**, **WCRHin**, **WCRLin**, **ARHin** ili **ARLin** bloka *registri*, respektivno. Signalima **CRin**, **SRin**, **WCRHin**, **WCRLin**, **ARHin** ili **ARLin** se obezbeđuje da se sadržaj sa linija podataka $DBUS_{7...0}$ magistrale upiše u jedan od registara $CR_{2...0}$, SR_0 , $WCRH_{7...0}$, $WCRL_{7...0}$, $ARH_{7...0}$ ili $ARL_{7...0}$, respektivno. Kada signal **wrSKTR** postane neaktivan i signali **fcSKTR**, **CRin**, **SRin**, **WCRHin**, **WCRLin**, **ARHin** ili **ARLin** postanu neaktivni, pa se upravljačka linija **FCBUS** magistrale vraća u stanje visoke impedanse. !

! Pri aktivnim vrednostima signala **wrKTR**, **selCR**, $DBUS_2$ i $DBUS_0$, što znači da se programskim putem upisuje sadržaj u registar $CR_{3...0}$ radi startovanja kontrolera za prenos iz periferije u memoriju, generiše se signal **stRDDRbus** bloka *registri*. Signalom **stRDDRbus** se flip-flop RDDR postavlja na aktivnu vrednost, što je indikacija da je registar $DR_{7...0}$ raspoloživ da se u njega prenese novi podatak iz registra $DRAUX_{7...0}$ bloka *registri*. !

! Pri aktivnim vrednostima signala **wrKTR**, **selCR** i $DBUS_2$ i neaktivnoj vrednosti signala $DBUS_0$, što znači da se programskim putem upisuje sadržaj u registar $CR_{3...0}$ radi startovanja kontrolera za prenos iz memorije u periferiju, generiše se signal **clWRDRbus** bloka *registri*. Signalom **clWRDRbus** se flip-flop WRDR postavlja na neaktivnu vrednost, što je indikacija da registar $DR_{7...0}$ nije raspoloživ da se iz njega prenese novi podatak u registra $DRAUX_{7...0}$. !

! Pri aktivnim vrednostima signala **wrSKTR**, **selCR**, $DBUS_2$, što znači da se programskim putem upisuje sadržaj u registar $CR_{3...0}$ radi startovanja kontrolera za prenos, i neaktivnoj vrednosti signala **done**, što znači da je sadržaj brojača veličine bloka za prenos $WCR_{15...0}$ nula, generiše se signal **clSR0bus** bloka *registri*. Signalom **clSR0bus** razred SR_0 statusnog registra SR_0 se postavlja na neaktivnu vrednost, što je indikacija da je prenos bloka podataka u toku. !

! Pri aktivnim vrednostima signala **wrSKTR**, **selCR**, **DBUS₂** i **done**, što znači da se programskim putem upisuje sadržaj u registar CR_{3...0} radi startovanja kontrolera za prenos i da je sadržaj brojača veličine bloka za prenos WCR_{15...0} nula, generiše se signal **stSR0bus** bloka *registri*. Signalom **stSR0bus** razred SR₀ statusnog registra SR₀ se postavlja na aktivnu vrednost, što je indikacija da je prenos bloka podataka završen. !

! Pri aktivnim vrednostima signala **wrSKTR**, **selCR**, **DBUS₂**, **DBUS₁** i **done**, što znači da se programskim putem upisuje sadržaj u registar CR_{3...0} radi startovanja kontrolera za prenos sa generisanjem prekida i da je sadržaj brojača veličine bloka za prenos WCR_{15...0} nula, generiše se signal prekida **intrbus** što je indikacija da je prenos bloka podataka završen. Signal prekida **intrbus** upravljačke jedinice *uprav_bus* sa signalom prekida **intrper** upravljačke jedinice *uprav_per* formira u bloku *interfejs* signal prekida **intr** procesora **CPU**. Signal prekida izaziva prelazak procesora na izvršavanje prekidne rutine u kojoj se programskim putem, upisivanjem sadržaja koji na poziciji bita stop ima neaktivnu vrednost, zaustavlja kontroler periferije. !

! Pri aktivnim vrednostima signala **wrSKTR** i **selCR** i neaktivnoj vrednosti signala **DBUS₂**, što znači da se programskim putem upisuje sadržaj u registar CR_{3...0} radi zaustavljanja kontrolera, generiše se signal zaustavljanja **stopPER** periferije *PER*. Signalom **stopPER** se u periferiji *PER* zaustavlja prenos koji je u toku. !

```

step0  if (rdSKTR + wrSKTR) then fcSKTR,
        if (rdSKTR·selCR) then CRout,
        if (rdKTR·selSR) then (SRout, clSR0bus),
        if (rdSKTR· selWCRH) then WCRHout,
        if (rdSKTR· selWCRL) then WCRLout,
        if (rdSKTR· selARH) then ARHout,
        if (rdSKTR· selARL) then ARLout,
        if (wrSKTR·selCR) then CRin,
        if (wrSKTR·selSR) then SRin,
        if (wrSKTR·selWCRH) then WCRHin,
        if (wrSKTR·selWCRL) then WCRLin,
        if (wrSKTR·selARH) then ARHin,
        if (wrSKTR·selARL) then ARLin,
        if (wrSKTR·selCR·DBUS2·DBUS0) then stRDDRbus,
        if (wrSKTR·selCR·DBUS2·  $\overline{\text{DBUS}_0}$ ) then clWRDRbus,
        if (wrSKTR·selCR·DBUS2·  $\overline{\text{done}}$ ) then clSR0bus,
        if (wrSKTR·selCR·DBUS2·done) then stSR0bus,
        if (wrSKTR·selCR·DBUS2· DBUS1· done) then intrbus,
        if (wrSKTR·selCR·  $\overline{\text{DBUS}_2}$ ) then stopPER

```

6.3.2.1.3 Struktura upravljačke jedinice

Struktura upravljačke jedinice je prikazana na slici 74. Upravljačka jedinica se sastoji od logičkih elemenata koji na osnovu upravljačkih signala čitanja **rdSKTR** i upisa **wrSKTR** bloka *interfejs* i signala logičkih uslova generišu sledeće upravljačke signale:

- signal **fcSKTR** bloka *interfejs* završetka ili čitanja nekog od registara kontrolera CR_{2...0}, SR₀, WCRH_{7...0}, WCRL_{7...0}, ARH_{7...0} ili ARL_{7...0} bloka *registri* ili upisa u neki od ovih registara kontrolera,
- signale **CRout**, **SRout**, **WCRHout**, **WCRLout**, **ARHout** ili **ARLout** bloka *registri* čitanja registara kontrolera CR_{2...0}, SR₀, WCRH_{7...0}, WCRL_{7...0}, ARH_{7...0} ili ARL_{7...0},
- signale **CRin**, **SRin**, **WCRHin**, **WCRLin**, **ARHin** ili **ARLin** bloka *registri* upisa u registre kontrolera CR_{2...0}, SR₀, WCRH_{7...0}, WCRL_{7...0}, ARH_{7...0} ili ARL_{7...0},

- signale **clSR0bus** i **stSR0bus** bloka *registri* postavljanja na neaktivnu i aktivnu vrednost, respektivno, razreda *prenos završen* SR₀ statusnog registra SR₀ kontrolera,
- signal **stRDDRbus** bloka *registri* postavljanja na aktivnu vrednost flip-flopa RDDR i signal **clWRDR** bloka *registri* postavljanja na neaktivnu vrednost flip-flopa WRDR,
- signal prekida **intrbus** bloka *interfejs* i
- signal zaustavljanja periferije **stopPER** bloka *interfejs*.

Signali **rdSKTR** i **wrSKTR** imaju aktivne vrednosti trajanja jedna perioda signala takta. To je posledica usvojenog načina generisanja signala **rdCPU** i **wrCPU** procesora **CPU**, na osnovu kojih se formiraju signali **RDBUS** i **WRBUS** magistrale **BUS** i **rdSKTR** i **wrSKTR** kontrolera **KP**, i signala **fcSKTR**, na osnovu koga se generišu signali **FCBUS** magistrale **BUS** i **fcCPU** procesora **CPU**. Kod čitanja procesor na *i*-ti signal takta ulazi u stanje step_{*i*} koje koristi da signal **rdCPU** postavi na aktivnu vrednost, što redom daje i aktivne vrednosti signala **RDBUS**, **rdKTR** **fcKTR**, **FCBUS** i **fcCPU**. Kako je u procesoru aktivna vrednost signala **fcCPU** uslov za prelazak iz stanja step_{*i*} u stanje step_{*i+1*} i kako ta vrednost dolazi u koraku step_{*i*}, to procesor na (*i*+1)-vi signal takta prelazi na korak step_{*i+1*}. Zbog toga se u koraku step_{*i*} ostaje samo jedna perioda signala takta, što ima za posledicu da signali **rdCPU**, **RDBUS**, **rdSKTR**, **fcSKTR**, **FCBUS** i **fcCPU** imaju aktivnu vrednost trajanja jedna perioda signala takta. Iz istih razloga su kod upisa signali **wrCPU**, **WRBUS**, **wrSKTR**, **fcSKTR**, **FCBUS** i **fcCPU** trajanja jedna perioda signala takta. S obzirom na to da su signali **rdCPU** i **wrCPU** trajanja jedna perioda signala takta i da svi signali koje generiše upravljačka jedinica *uprav_bus* uključuju jedan od signala **rdSKTR** i **wrSKTR**, to i svi preostali signali imaju aktivne vrednosti trajanja jedna perioda signala takta.

Upravljačka jedinica *uprav_bus* sadrži kombinacione mreže koje pomoću upravljačkih signala **rdSKTR** i **wrSKTR**, koji dolaze sa bloka *interfejs*, signala logičkih uslova **selCR**, **selSR**, **selWCRH**, **selWCRL**, **selARH**, **selARL**, **done**, **DBUS₂**, **DBUS₁** i **DBUS₀**, koji dolaze iz blokova *interfejs* i *registri* i magistrale **BUS**, i saglasno algoritmu generisanja upravljačkih signala (tabela 26) generiše upravljačke signale blokova operacione jedinice.

Upravljački signali blokova operacione jedinice **oper** se daju posebno za blok *registri* i blok *interfejs*.

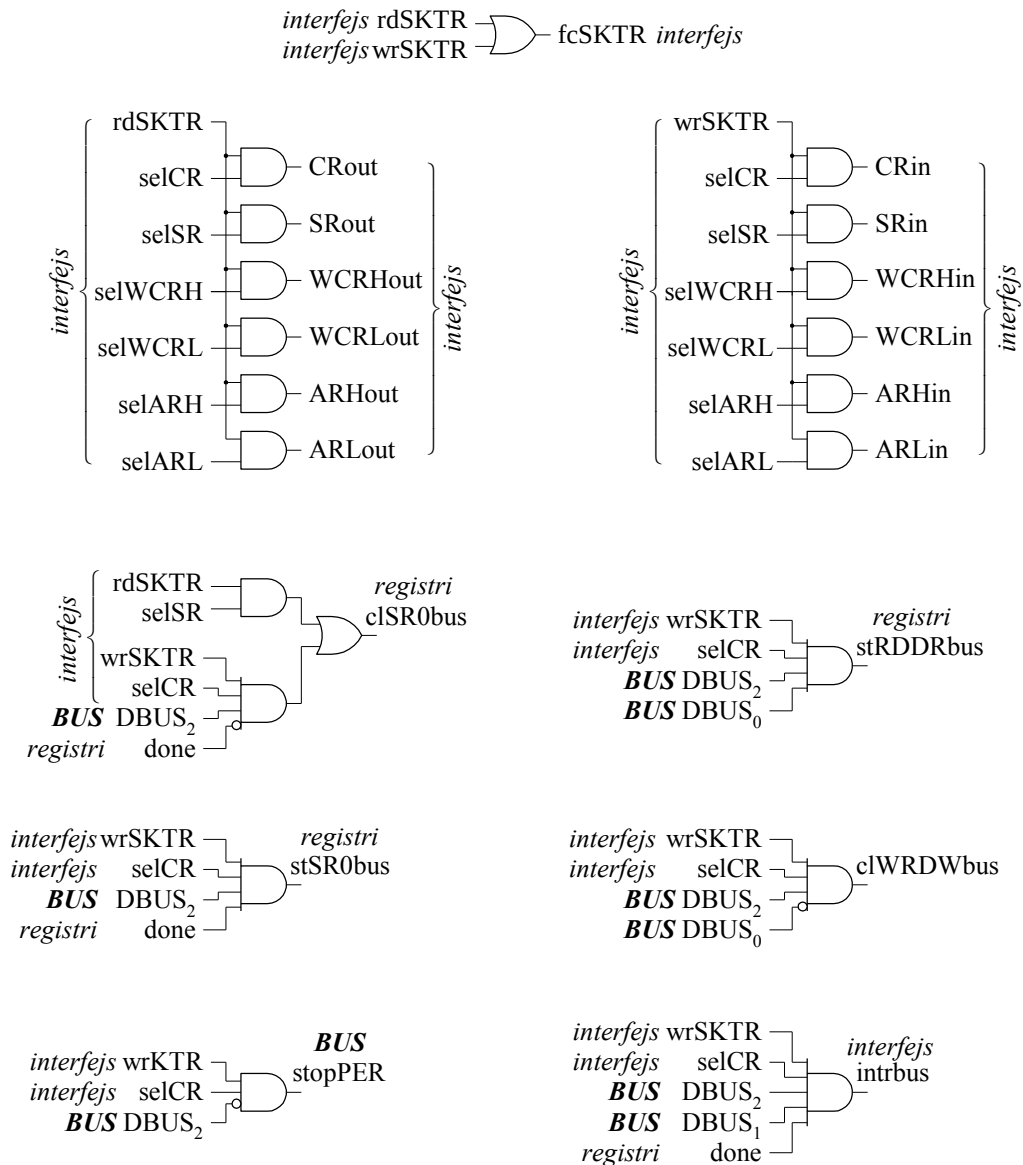
Upravljački signali bloka *registri* se generišu na sledeći način:

- **CRout** = **rdSKTR**·**selCR**
- **SRout** = **rdSKTR**·**selSR**
- **WCRHout** = **rdSKTR**·**selWCRH**
- **WCRLout** = **rdSKTR**·**selWCRL**
- **ARHout** = **rdSKTR**·**selARH**
- **ARLout** = **rdSKTR**·**selARL**
- **CRin** = **wrSKTR**·**selCR**
- **SRin** = **wrSKTR**·**selSR**
- **WCRHin** = **wrSKTR**·**selWCRH**
- **WCRLin** = **wrSKTR**·**selWCRL**
- **ARHin** = **wrSKTR**·**selARH**
- **ARLin** = **wrSKTR**·**selARL**
- **clSR0bus** = **rdSKTR**·**selSR** + **wrSKTR**·**selCR**·**DBUS₂**·**done**
- **stRDDRbus** = **wrSKTR**·**selCR**·**DBUS₂**·**DBUS₀**

- $clWRDRbus = wrSKTR \cdot selCR \cdot DBUS_2 \cdot \overline{DBUS_0}$
- $stSR0bus = wrSKTR \cdot selCR \cdot DBUS_2 \cdot done$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz blokova *interfejs* i *registri* operacione jedinice **oper** i magistrale **BUS** i to:

- **rdSKTR** — bloka *interfejs*.
- **wrSKTR** — bloka *interfejs*
- **selCR** — bloka *interfejs*
- **selSR** — bloka *interfejs*
- **selWCRH** — bloka *interfejs*
- **selWCRL** — bloka *interfejs*
- **selARH** — bloka *interfejs*
- **selARL** — bloka *interfejs*
- **done** — bloka *registri*
- **DBUS₂** — magistrale **BUS** i
- **DBUS₀** — magistrale **BUS**.



Slika 84 Struktura upravljačke jedinice *uprav_bus*

Upravljački signali bloka *interfejs* se generišu na sledeći način:

- **intrbus** = **wrSKTR**·**selCR**·**DBUS₂**· **DBUS₁**·**done**
- **stopPER** = **wrSKTR**·**selCR**· **DBUS₂**

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz bloka *interfejs* operacione jedinice **oper** i magistrale **BUS** i to:

- **wrSKTR** — bloka *interfejs*
- **selCR** — bloka *interfejs*
- **done** — bloka *registri*
- **DBUS₂** — magistrale **BUS**
- **DBUS₁** — magistrale **BUS** i
- **DBUS₀** — magistrale **BUS**.

6.3.2.2 Upravljačka jedinica periferije

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

6.3.2.2.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 75). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

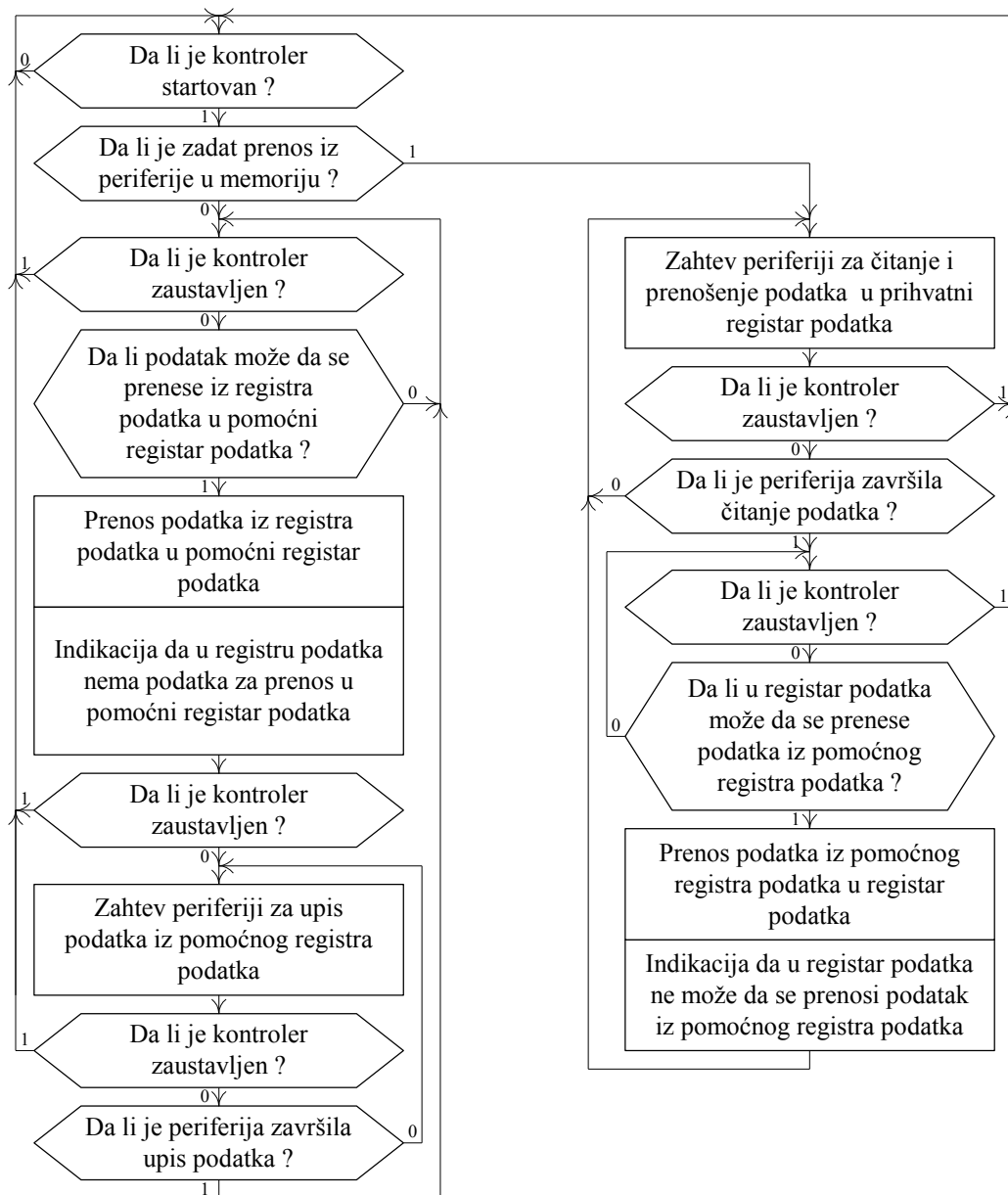
U početnom koraku vrši se provera da li je kontroler startovan ili ne. Ako kontroler nije startovan ostaje se u početnom koraku i čeka startovanje kontrolera. Ako je kontroler startovan vrši se provera da li je zadat prenos iz periferije u memoriju ili iz memorije u periferiju i prelazi na odgovarajući korak.

Ako je zadat prenos iz periferije u memoriju prelazi se na prenošenje podatka iz periferije u kontroler. Najpre se u petlji drži zahtev periferiji da pročita podatak i vrši provera da li je kontroler zaustavljen programskim putem upisivanjem neaktivne vrednosti u razred *start* upravljačkog registra i da li je periferija završila čitanje podatka. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz periferije u kontroler, vraća se u početni korak i čeka da se ponovo programskim putem upisivanjem aktivne vrednosti u razred *start* upravljačkog registra startuje kontroler. Ukoliko se utvrdi da je periferija pročitala podatak izlazi se iz petlje i podatak prenosi iz periferije u pomoćni registar podatka kontrolera.

Potom se u petlji vrši provera da li je kontroler zaustavljen programskim putem i da li je registar podatka raspoloživ da se u njega prenese novi podatak iz pomoćnog registra podatka. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz periferije u kontroler, vraća se u početni korak i čeka da se ponovo programskim putem startuje kontroler. Ukoliko se utvrdi da je registar podatka raspoloživ, izlazi se iz petlje i podatak prenosi iz pomoćnog registra podatka u registar podatka. Istovremeno se postavlja indikacija da registar podatka nije raspoloživ da se u njega prenese novi podatak iz pomoćnog registra podatka sve dok se programskim putem podatak ne prenese iz registra podatka u memoriju. Na kraju se vraća na korak u kome se postavlja zahtev periferiji da pročita sledeći podatak. Opisane aktivnosti se ponavljaju sve dok se kontroler ne zaustavi programskim putem.

Ako je u zadat prenos iz memorije u periferiju prelazi se na prenošenje podatka iz kontrolera u periferiju. Najpre se u petlji vrši provera da li je kontroler zaustavljen programskim putem i da li je registar podatka raspoloživ da se iz njega prenese novi podatak u

pomoćni registar podatka. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz kontrolera u periferiju, vraća se u početni korak i čeka da se ponovo programskim putem, startuje kontroler. Ukoliko se utvrdi da je registar podatka raspoloživ izlazi se iz petlje i podatak prenosi iz registra podatka u pomoćni registar podatka. Istovremeno se postavlja indikacija da registar podatka nije raspoloživ da se iz njega prenosi podatak u pomoćni registar podatka sve dok se programskim putem podatak ne prenese iz memorije u registar podatka. Pored toga na aktivnu vrednost se postavlja bit spremnost statusnog registra kontrolera, čime se postavlja indikacija da je kontroler spreman za prenos podatka iz memorije u registar podatka, i generiše prekid, ukoliko je pri startovanju zadat režim sa generisanjem prekida.



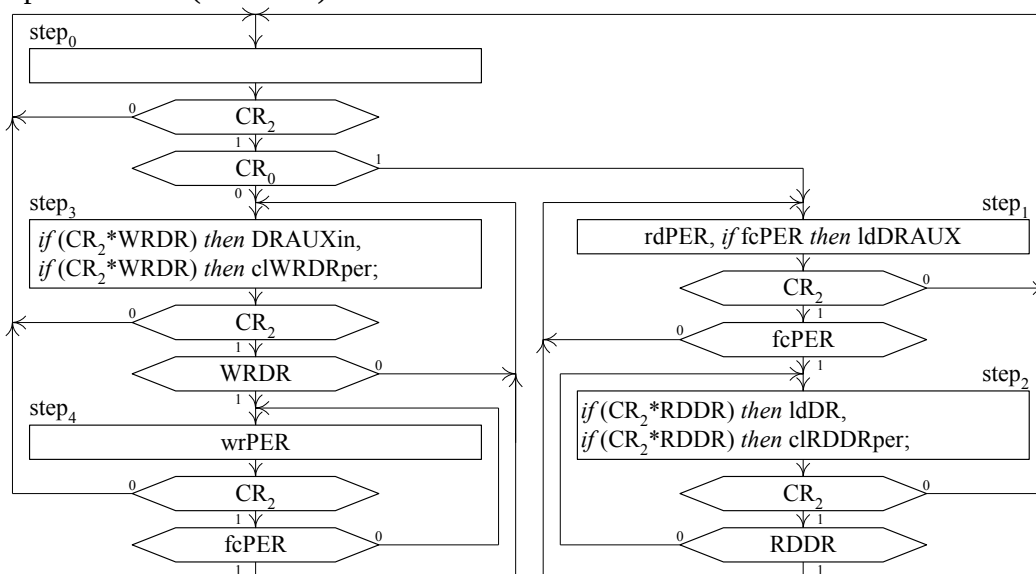
Slika 85 Dijagram toka operacija

Zatim se vrši provera da li je kontroler zaustavljen programskim putem. Ukoliko se utvrdi da je kontroler zaustavljen, prekida se prenošenje podataka iz kontrolera u periferiju, vraća se u početni korak i čeka da se ponovo programskim putem startuje kontroler. Ukoliko se utvrdi da kontroler nije zaustavljen u petlji se periferiji drži zahtev da upiše sledeći podatak i vrše provere da li je kontroler zaustavljen programskom putem i da li je periferija završila upis

podatka. Ukoliko se utvrdi da je kontroler periferije zaustavljen, prekida se prenošenje podataka iz kontrolera u periferiju, vraća se u početni korak i čeka da se ponovo programskim putem startuje kontroler. Ukoliko se utvrdi da je periferija završila upis podatka iz pomoćnog registra podatka kontrolera, izlazi se iz ove petlje i vraća u petlju u kojoj se vrši provera da li je kontroler zaustavljen programskim putem i da li je registar podatka kontrolera raspoloživ da se iz njega prenese novi podatak u pomoćni registar podatka kontrolera. Opisane sktivnosti se ponavljaju sve dok se kontroler ne zaustavi programskim putem.

6.3.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 75) i dat u obliku dijagrama toka upravljačkih signala (slika 76) i sekvence upravljačkih signala po koracima (tabela 27).



Slika 86 Dijagram toka upravljačkih signala

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 76). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova.

U sekvenci upravljačkih signala po koracima je za svaki korak data simbolička oznaka samog koraka i iskazi za signale i skokove. Iskazi za signale se pojavljuju ukoliko u datom koraku treba da se generiše neki od upravljačkih signala operacione jedinice. Iskazi za signale sadrže spisak upravljačkih signala operacione jedinice koji se generišu bezuslovno i uslovno, a za signale koji se generišu uslovno i signale logičkih uslova pod kojima se signali generišu. Iskazi za skokove se pojavljuju ukoliko u treba odstupiti od sekvencijalnog generisanja upravljačkih signala operacione jedinice. Iskazi za skokove sadrže uslov i korak i određuje na koji korak i pod kojim uslovima treba preći. Notacija koja se koristi je identična kao i notacija za algoritam generisanja upravljačkih signala za upravljačku jedinicu procesora **CPU** (poglavlje \$\$\$).

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 29 Sekvenca upravljačkih signala

! U koraku se step0 se ostaje i ne generiše se aktivna vrednost ni jednog od upravljačkih signala sve dok je signal **CR₂** bloka *registri* neaktivan. Signal **CR₂** postaje aktivan tek kada se programskim

putem startuje kontroler periferije. Tom prilikom se u upravljački registar $CR_{2...0}$ bloka *registri* upisuje sadržaj koji na poziciji razreda CR_2 ima aktivnu vrednost. Na poziciji razreda CR_0 je aktivna ili neaktivna vrednost i zavisnosti od toga da li je zadat prenos iz periferije u memoriju ili iz memorije u periferiju, respektivno. Na poziciji razreda CR_1 je aktivna ili neaktivna vrednost i zavisnosti od toga da li je zadat režim rada sa generisanjem ili bez generisanja prekida, respektivno. Pri aktivnim vrednostima signala CR_2 i CR_0 se prelazi na korak $step_1$ koji odgovara prenosu iz periferije u memoriju. Pri aktivnoj i neaktivnoj vrednosti signala CR_2 i CR_0 , respektivno, se prelazi na korak $step_3$, koji odgovara prenosu iz memorije u periferiju. !

$step_0$ *br* (if ($CR_2 \cdot CR_0$) then $step_1$),
 br (if ($CR_2 \cdot \overline{CR_0}$) then $step_3$);

! U korak $step_1$ se dolazi iz koraka $step_0$ ili koraka $step_2$. U ovom koraku se generiše signal **rdPER** periferije *PER* koji predstavlja zahtev periferiji da pročita sledeći podatak. Ukoliko je neaktivna vrednost signala CR_2 , što znači da je programskim putem upisivanjem neaktivne vrednosti u razred CR_2 upravljačkog registra *CR* zaustavljen rad kontrolera, prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_1$ ili se prelazi u korak $step_2$ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signala **fcPER** periferije *PER*, respektivno. Neaktivna vrednost signala **fcPER** je indikacija da čitanje podatka u periferiji još uvek traje, a aktivna vrednost da je podatak pročitao i da se nalazi na linijama $PIN_{7...0}$ periferije. Pri aktivnim vrednostima signala CR_2 i **fcPER** se generiše signal **ldDRAUX**. Ovim signalom se sadržaj iz periferije, koji u kontroler dolazi po linijama $PIN_{7...0}$, propušta kroz multiplekser *MP2* bloka *registri* i upisuje u pomoćni registar podatka $DRAUX_{7...0}$ bloka *registri*. !

$step_1$ **rdPER**, if ($CR_2 \cdot fcPER$) then **ldDRAUX**,
 br (if $\overline{CR_2}$ then $step_0$),
 br (if ($CR_2 \cdot fcPER$) then $step_2$);

! U korak $step_2$ se dolazi iz koraka $step_1$. Ukoliko je neaktivna vrednost signala CR_2 prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_2$ ili se prelazi u korak $step_1$ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signal **RDDR** bloka *registri*, respektivno. Neaktivna vrednost signala **RDDR** je indikacija da registar podatka $DR_{7...0}$ nije raspoloživ, jer upravljačka jedinica memorije *uprav_mem* još uvek nije prenela prethodni podatak iz registra $DR_{7...0}$ u memoriju. Aktivna vrednost signala **RDDR** je indikacija da je registar podatka $DR_{7...0}$ raspoloživ, pa se generišu signali **ldDDR** i **clRDDRper**. Signalom **ldDDR** se sadržaj registra $DRAUX_{15...0}$ bloka *registri* propušta kroz multiplekser *MP1* bloka *registri* i upisuje u registar podatka $DR_{7...0}$. Signalom **clRDDRper** se u flip-flop **RDDR** upisuje neaktivna vrednost. Ova vrednost je indikacija da registar $DR_{7...0}$ nije raspoloživ da se u njega prenesu novi podatak iz registra $DRAUX_{7...0}$ sve dok se programskim putem prethodni podatak ne prenesu iz registra $DR_{7...0}$ u memoriju. !

$step_2$ if ($CR_2 \cdot RDDR$) then (**ldDDR**, **clRDDRper**),
 br (if $\overline{CR_2}$ then $step_0$),
 br (if ($CR_2 \cdot RDDR$) then $step_1$);

! U korak $step_3$ se dolazi iz koraka $step_0$ ili koraka $step_4$. Ukoliko je neaktivna vrednost signala CR_2 prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_3$ ili se prelazi u korak $step_4$ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signal **WRDR** bloka *registri*, respektivno. Neaktivna vrednost signala **WRDR** je indikacija da registar podatka $DR_{7...0}$ bloka *registri* nije raspoloživ, jer upravljačka jedinica memorije *uprav_mem* još uvek nije prenela novi podatak iz memorije u registar $DR_{7...0}$. Aktivna vrednost signala **WRDR** je indikacija da je registar $DR_{7...0}$ raspoloživ jer je upravljačka jedinica memorije *uprav_mem* prenela podatak iz memorije u registar podatka $DR_{7...0}$. Pri prenosu podatka iz memorije u registar $DR_{7...0}$ upravljačka jedinica *uprav_mem* signalom **stWRDRmem** postavlja flip-flop **WRDR** na aktivnu vrednost. Pri aktivnoj vrednosti signala **WRDR** generišu se signali **DRAUXin** i **clWRDRper**. Neaktivnom vrednošću signala **ldDRAUX** se sadržaj registra $DR_{7...0}$ propušta kroz multiplekser *MP2* bloka *registri*, a aktivnom vrednošću signala **DRAUXin** se upisuje u pomoćni registar podatka $DRAUX_{7...0}$ bloka *registri*. Signalom **clWRDRper** se u flip-flop **WRDR** upisuje neaktivna vrednost. Ova vrednost je indikacija da registar $DR_{7...0}$ nije raspoloživ i da njegov sadržaj ne može da se prenesu u registar

DRAUX_{7...0} sve dok upravljačka jedinica memorije *uprav_mem* novi podatak ne prenese iz memorije u registar DR_{7...0}. !

```

step3   if (CR2·WRDR) then (DRAUXin, clWRDRper),
          br (if  $\overline{\text{CR}}_2$  then step0),
          br (if (CR2·WRDR) then step4);

```

! U korak step₄ se dolazi iz koraka step₃. U ovom koraku se generiše signal **wrPER** periferije *PER* koji predstavlja zahtev periferiji da upiše podatak iz registra DRAUX_{7...0} koji je prisutan na linijama POUT_{7...0}. Ukoliko je neaktivna vrednost signala **CR₂** prelazi se u korak step₀. Ukoliko je aktivna vrednost signala **CR₂**, ili se ostaje u koraku step₄ ili se prelazi u korak step₃ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signal **fcPER** periferije *PER*, respektivno. Neaktivna vrednost signala **fcPER** je indikacija da je upis podatka iz registra DRAUX_{7...0} u periferiju u toku, a aktivna vrednost da je podatak upisan. !

```

step4   wrPER,
          br (if  $\overline{\text{CR}}_2$  then step0),
          br (if (CR2·fcPER) then step3);

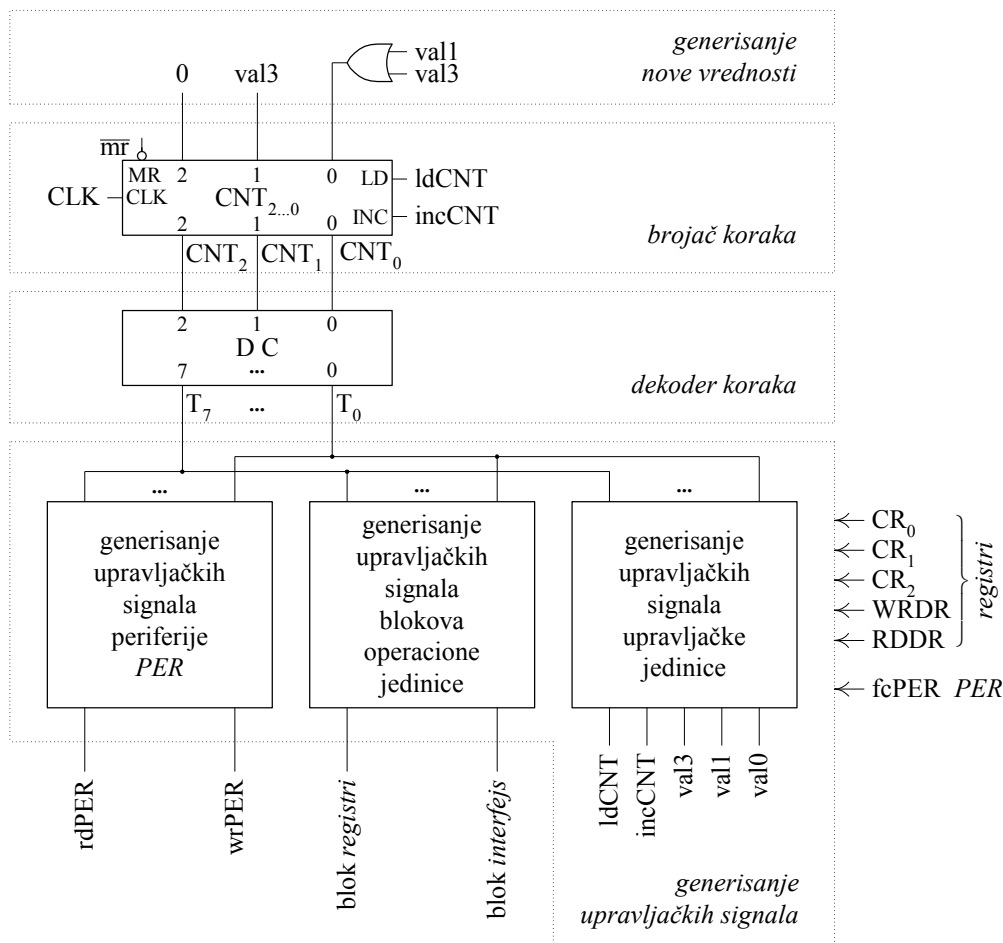
```

6.3.2.2.3 Struktura upravljačke jedinice

Upravljačka jedinica (slika 77) se sastoji od sledećih blokova:

- blok *generisanje nove vrednosti brojača koraka*,
- blok *brojač koraka*,
- blok *dekoder koraka* i
- blok *generisanje upravljačkih signala*.

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



Slika 87 Struktura upravljačke jedinice *uprav_per*

Blok *generisanje nove vrednosti brojača koraka* služi za generisanje vrednosti koju treba upisati u brojač CNT. Potreba za ovim se javlja kada treba odstupiti od sekvencijalnog generisanja upravljačkih signala. Analizom algoritma generisanja upravljačkih signala operacione jedinice (poglavlje 6.2.2.2.2) se utvrđuje da su 0, 1 i 3 vrednosti koje treba upisati u brojač CNT da bi se realizovala odstupanja od sekvencijalnog generisanja upravljačkih signala. Te vrednosti se formiraju na ulazima 2, 1 i 0 brojača CNT pomoću signala **val₀**, **val₁** i **val₃** pri čemu signal **val₀** ima vrednost 1 samo onda kada treba upisati 0, signal **val₁** ima vrednost 1 samo onda kada treba upisati 1 i signal **val₃** ima vrednost 1 samo onda kada treba upisati 3. Time se obezbeđuje da na ulazima 2, 1 i 0 brojača CNT budu vrednosti 0, 0 i 0 onda kada signal **val₀** ima vrednost 1, vrednosti 0, 0 i 1 onda kada signal **val₁** ima vrednost 1 i vrednosti 0, 1, i 1 onda kada signal **val₃** ima vrednost 1.

Blok *brojač koraka* sadrži brojač CNT. Brojač CNT svojom trenutnom vrednošću obezbeđuje aktivne vrednosti određenih upravljačkih signala. Brojač CNT može da radi u sledećim režimima:

- režim inkrementiranja,
- režim skoka i
- režim mirovanja.

U režimu inkrementiranja pri pojavi signala takta vrši se uvećavanje sadržaja brojača CNT za jedan. Ovim režimom se obezbeđuje sekvencijalno generisanje upravljačkih signala iz algoritma generisanja upravljačkih signala (poglavlje 6.2.2.2.2). Ovaj režim rada se obezbeđuje aktivnom vrednošću signala **incCNT**. Signal **incCNT** je uvek neaktivan sem kada treba obezbediti režim inkrementiranja.

U režimu skoka pri pojavi signala takta vrši se upis nove vrednosti u brojač CNT. Ovim režimom se obezbeđuje odstupanje od sekvencijalnog generisanja upravljačkih signala iz algoritma generisanja upravljačkih signala (poglavlje 6.2.2.2.2). Ovaj režim rada se obezbeđuje aktivnom vrednošću signala **ldCNT**. Signal **ldCNT** je uvek neaktivan sem kada treba obezbediti režim skoka.

U režimu mirovanja pri pojavi signala takta ne menja se vrednost brojača CNT. Ovaj režim rada se obezbeđuje neaktivnim vrednostima signala **incCNT** i **ldCNT**. Ovi signali su neaktivni kada se u koraku

- step₀ pri aktivnoj vrednosti signala **T₀** čeka da kontroler bude startovan upisivanjem aktivne vrednosti u razred CR₂ upravljačkog registra kontrolera,
- step₁ pri aktivnoj vrednosti signala **T₁** čeka da čitanje u perifriji bude završeno i signal **fcPER** postane aktivan,
- step₂ pri aktivnoj vrednosti signala **T₂** čeka da registar podatka DR_{7...0} bude raspoloživ za prebacivanje podatka iz pomoćnog registra podatka DRAUX_{7...0} i signal **RDDR** postane aktivan,
- step₃ pri aktivnoj vrednosti signala **T₃** čeka da pomoćni registar podatka DRAUX_{7...0} bude raspoloživ za prebacivanje podatka iz registra podatka DR_{7...0} i signal **WRDR** postane aktivan i
- step₄ pri aktivnoj vrednosti signala **T₄** čeka da upis u perifriju bude završen i signal **fcPER** postane aktivan.

Blok *dekoder koraka* sadrži dekode DC. Na ulaze dekodera DC vode se izlazi brojača CNT. Dekodovana stanja brojača CNT pojavljuju se kao signali **T₀** do **T₇** na izlazima dekodera DC. Svakom koraku iz algoritma generisanja upravljačkih signala (poglavlje

6.2.2.2.2) dodeljen je jedan od ovih signala i to koraku $step_0$ signal T_0 , koraku $step_1$ signal T_1 , itd.

Blok *generisanje upravljačkih signala* sadrži kombinacione mreže koje pomoću signala T_0 do T_7 koji dolaze sa bloka *dekoder koraka* upravljačke jedinice, signala logičkih uslova koji dolaze iz blokova operacione jedinice i saglasno algoritmu generisanja upravljačkih signala (poglavlje 6.2.2.2.2) generišu upravljačke signale. Upravljački signali se generišu na identičan način kao i upravljački signali procesora **CPU** (poglavlje 6.2.2.2.2).

Blok *generisanje upravljačkih signala* generiše tri grupe upravljačkih signala i to:

- upravljačke signale periferije *PER*,
- upravljačke signale blokova operacione jedinice *oper* i
- upravljačke signale upravljačke jedinice *uprav_per*.

Upravljački signali periferije *PER* se generišu na sledeći način:

- $rdPER = CR_2 \cdot T_1$
- $wrPER = CR_2 \cdot T_4$

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

- CR_2 — blok *registri*.

Upravljački signali blokova operacione jedinice *oper* se daju posebno za blok *registri* i blok *interfejs*.

Upravljački signali bloka *registri* se generišu na sledeći način:

- $clRDDR_{per} = CR_2 \cdot RDDR \cdot T_2$
- $clWRDR_{per} = CR_2 \cdot WRDR \cdot T_3$
- $ldDDR = CR_2 \cdot RDDR \cdot T_2$
- $DRAUX_{in} = CR_2 \cdot WRDR \cdot T_3$
- $ldDRAUX = CR_2 \cdot fcPER \cdot T_1$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz blokova operacione jedinice i periferije *PER* i to:

- CR_2 — blok *registri*,
- $RDDR$ — blok *registri*,
- $WRDR$ — blok *registri*,
- $fcPER$ — periferije *PER*.

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

- $ldCNT = \overline{val0} + val1 + val3$
- $val0 = CR_2 \cdot (T_1 + T_2 + T_3 + T_4)$
- $val1 = CR_2 \cdot RDDR \cdot T_2$
- $val3 = CR_2 \cdot CR_0 \cdot T_0 + CR_2 \cdot fcPER \cdot T_4$
- $incCNT = CR_2 \cdot CR_0 \cdot T_0 + CR_2 \cdot fcPER \cdot T_1 + CR_2 \cdot WRDR \cdot T_3$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz periferije *PER* i pojedinih blokova operacione jedinice *oper* i to:

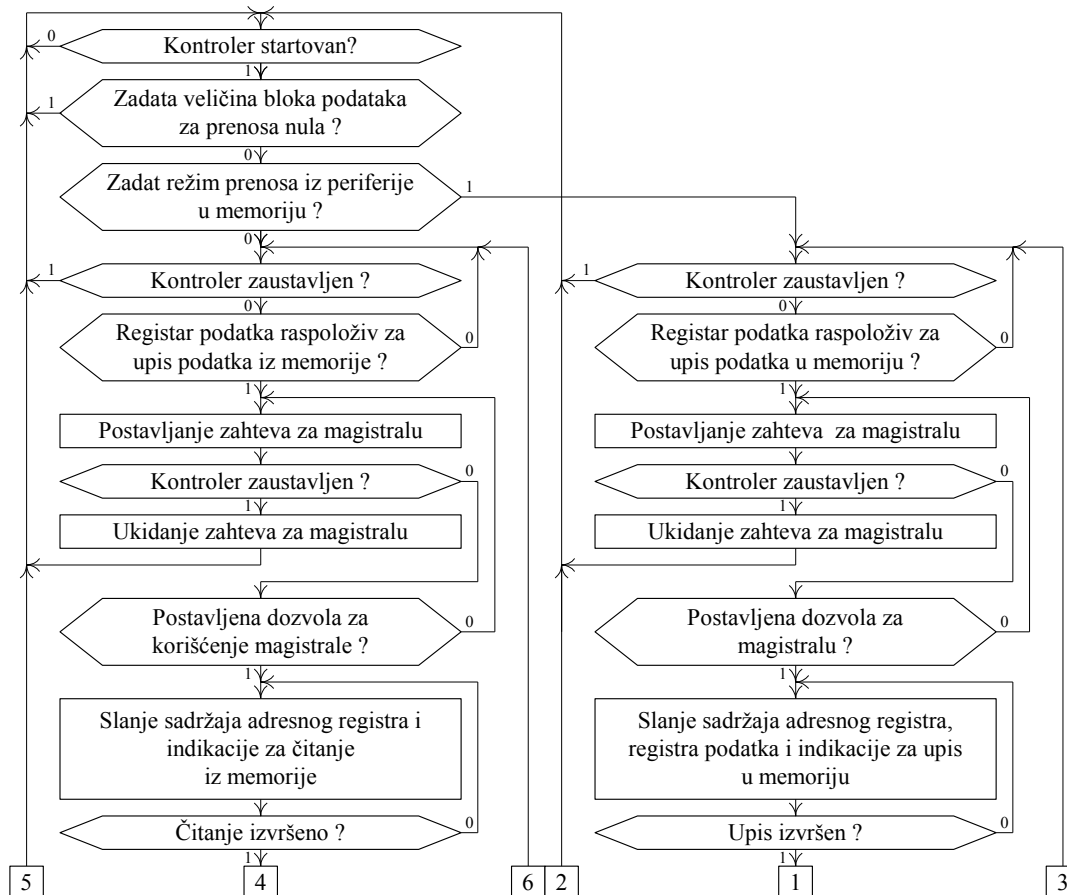
- $fcPER$ — periferija *PER*,
- CR_0 — blok *registri*,
- CR_2 — blok *registri*,
- $RDDR$ — blok *registri* i
- $WRDR$ — blok *registri*.

6.3.2.3 Upravljačka jedinica memorije

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

6.3.2.3.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 88). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.



Slika 88 Dijagram toka operacija

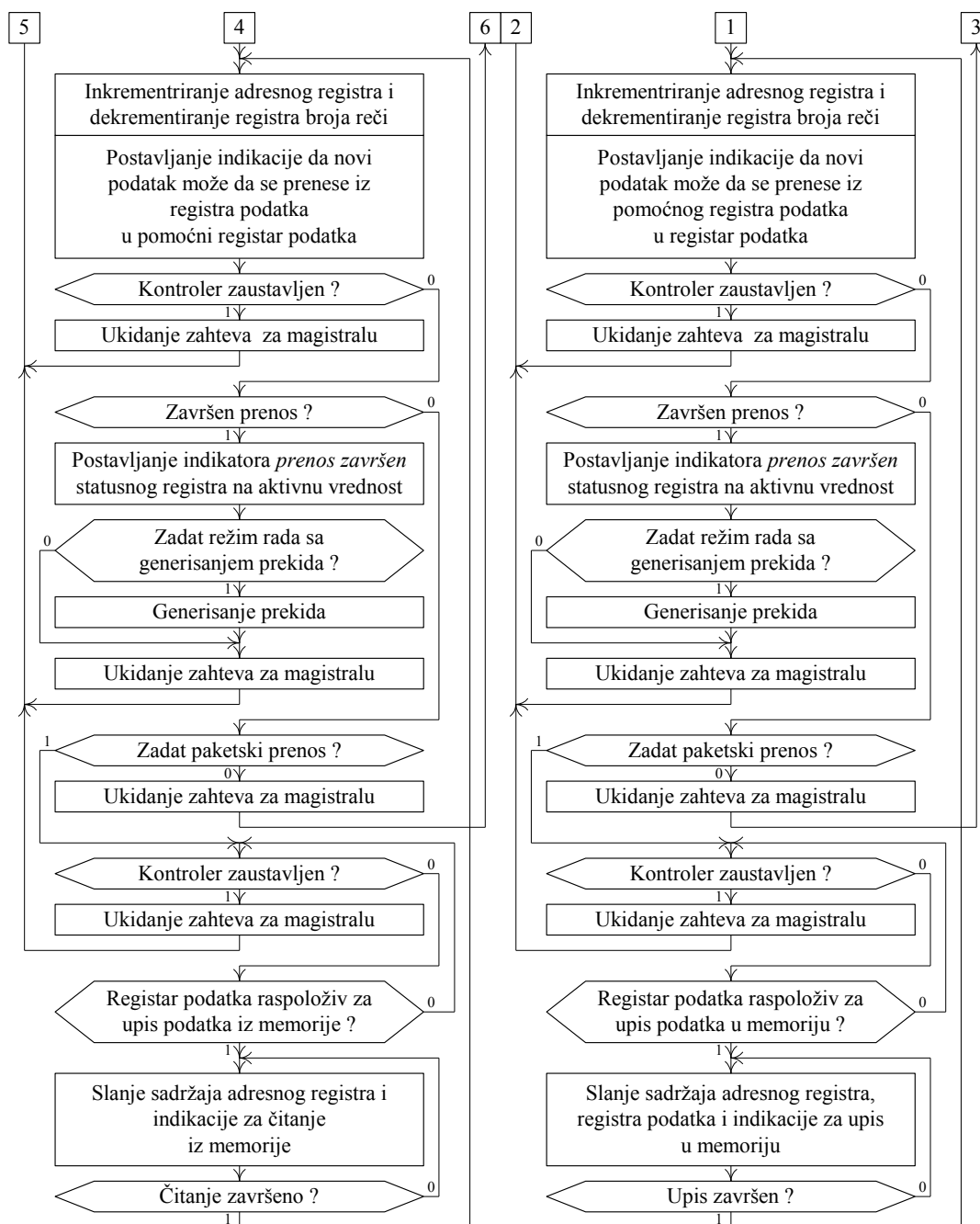
U početnom koraku dijagrama toka operacija vrši se provera da li je kontroler startovan i u slučaju da kontroler nije startovan ostaje se u početnom koraku. Ako je kontroler startovan, vrši se provera da li je zadata veličina bloka podataka za prenos nula i u slučaju da je nula ostaje se u početnom koraku. U slučaju da je kontroler periferije startovan i da je zadata veličina bloka podatak za prenos različita od nule, prelazi se na odgovarajuće korake u zavisnosti od toga da li je zadat prenos iz periferije u memoriju ili iz memorije u periferiju.

Ako je zadat prenos iz periferije u memoriju prelazi se na prenošenje podataka iz registra podatka u memoriju. Najpre se u petlji proverava da li je kontroler programskim putem zaustavljen i da li je registar podatka raspoloživ za upis podatka u memoriju. Ukoliko je kontroler programskim putem zaustavljen, izlazi se iz petlje i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, u petlji se ostaje sve dok registar podatka ne postane raspoloživ za upis podatka u memoriju. To će se dogoditi kada upravljačka jedinica *uprav_per*, koja je odmah po startovanju kontrolera periferije krenula sa čitanjem podatka iz periferije, podatak prenese

najpre iz periferije u pomoćni registar podatka i zatim iz pomoćnog registra podatka u registar podatka.

Potom se prelazi na dobijanje dozvole korišćenja magistrale radi upisa podatka iz registra podatka u memoriju. Najpre se u petlji postavlja zahtev za korišćenje magistrale i proverava da li je kontroler programskim putem zaustavljen i da li je procesor postavio dozvolu za korišćenje magistrale. Ukoliko je kontroler programskim putem zaustavljen, izlazi se iz petlje i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, u petlji se ostaje sve dok procesor ne postavi dozvolu korišćenja magistrale. To će se dogoditi odmah po postavljanju zahteva za korišćenje magistrale ukoliko procesor ne koristi magistralu ili kasnije kada procesor završi sa korišćenjem magistrale.

Kada se utvrdi da je procesor postavio dozvolu za korišćenja magistrale izlazi se iz petlje i ulazi u sledeću petlju u kojoj se adresa, podatak i upravljački signal upisa po linijama magistrale šalju memoriji i čeka da memorija pošalje po liniji magistrale upravljački signal završetka upisa.



Slika 88 Dijagram toka operacija (nastavak)

Po završetku upisa izlazi se iz petlje i prelazi se na korak u kome se inkrementira sadržaj adresnog registra i dekrementira sadržaj registra broja reči za prenos. Kao rezultat u adresnom registru sa sada nalazi adresa memorijske lokacije u koju treba upisati sledeći podatak, a u registru broja reči za prenos broj preostalih reči za prenos iz periferije u memoriju. Pored toga postavlja se indikacija da registar podatka nije raspoloživ za upis u memoriju. Ovo je indikacija da upravljačka jedinica *uprav_per*, koja je odmah po prenošenju prethodnog podatka iz pomoćnog registra podatka u registar podatka krenula sa čitanjem sledećeg podatka iz periferije, podatak prenese najpre iz periferije u pomoćni registar podatka i zatim iz pomoćnog registra podatka u registar podatka.

Potom se ponovo proverava da li je kontroler programskim putem zaustavljen i ukoliko je jeste, ukida se zahtev za korišćenje magistrale i vraća u početni korak u kome se čeka da

kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, proverava se da li je kao rezultat dekrementiranja sadržaj brojača reči postao nula, što je indikacija da je završen prenos podataka, ili je još uvek veći od nule, što je indikacija da treba produžiti sa prenosom podataka.

Ukoliko je završen prenos podataka, indikator *prenos završen* statusnog registra se postavlja na aktivnu vrednost i ukoliko je zadat režim rada sa generisanjem prekida, generiše se prekid. Pored toga se ukida zahtev za korišćenje magistrale i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko nije završen, prenos podataka se produžava na dva načina u zavisnosti od toga da li je zadat pojedinačni ili paketski prenos.

Ukoliko nije zadat paketski već pojedinačni prenos, ukida se zahtev za korišćenje magistrale i vraća na korak u kome se najpre čeka da registar podatka postane raspoloživ za upis podatka u memoriju, što će se desiti kada upravljačka jedinica *uprav_per* prenese sledeći podatak iz pomoćnog registra podatka u registar podatka. Tada se ponavlja već opisani postupak po kome se najpre postavlja dozvola za korišćenje magistrale i po dobijanju dozvole podatak upisuje iz registra podatka u memoriju.

Ukoliko je zadat paketski prenos u petlji se proverava da li je kontroler programskim putem zaustavljen i da li je registar podatka raspoloživ za upis podatka u memoriju. Ukoliko je kontroler programskim putem zaustavljen, izlazi se iz petlje i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, u petlji se ostaje sve dok registar podatka ne postane raspoloživ za upis podatka u memoriju. To će se dogoditi kada upravljačka jedinica *uprav_per*, koja je odmah po prenošenju prethodnog podatka iz pomoćnog registra podatka u registar podatka krenula sa čitanjem sledećeg podatka iz periferije, podatak prenese najpre iz periferije u pomoćni registar podatka i zatim iz pomoćnog registra podatka u registar podatka.

Kada se utvrdi da je registar podatka raspoloživ za upis podatka u memoriju izlazi se iz petlje i ulazi u sledeću petlju u kojoj se adresa, podatak i upravljački signal upisa po linijama magistrale šalju memoriji i čeka da memorija pošalje po liniji magistrale upravljački signal završetka upisa.

Po završetku upisa izlazi se iz petlje i prelazi se na korak u kome se inkrementira sadržaj adresnog registra, dekrementira sadržaj registra broja reči za prenos i ponavlja već opisani postupak na jedan od dva moguća načina u zavisnosti od toga da li je dekrementiranjem sadržaja registra broja reči dobijen sadržaj koju ukazuje da prenos treba završiti ili produžiti.

Ako je zadat prenos iz memorije u periferiju prelazi se na prenošenje podataka iz memorije u registar podatka. Najpre se u petlji proverava da li je kontroler programskim putem zaustavljen i da li je registar podatka raspoloživ za upis podatka iz memorije. Ukoliko je kontroler programskim putem zaustavljen, izlazi se iz petlje i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, u petlji se ostaje sve dok registar podatka ne postane raspoloživ za upis podatka iz memorije. Na početku pri prenosu prve reči registar podatka je uvek raspoloživ, dok će za prenos svake sledeće biti raspoloživ tek kada upravljačka jedinica *uprav_per*, koja je odmah po startovanju kontrolera periferije krenula sa upisivanjem podataka u periferiju, prethodni podatak prenese iz registra podatka u pomoćni registar podatka i zatim započne upis podatka iz pomoćnog registra podatka u periferiju.

Potom se prelazi na dobijanje dozvole korišćenja magistrale radi čitanja podatka iz memorije i upisa u registar podatka. Najpre se u petlji postavlja zahtev za korišćenje magistrale i proverava da li je kontroler programskim putem zaustavljen i da li je procesor

postavio dozvolu za korišćenje magistrale. Ukoliko je kontroler programskim putem zaustavljen, izlazi se iz petlje i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, u petlji se ostaje sve dok procesor ne postavi dozvolu korišćenja magistrale. To će se dogoditi odmah po postavljanju zahteva za korišćenje magistrale ukoliko procesor ne koristi magistralu ili kasnije kada procesor završi sa korišćenjem magistrale.

Kada se utvrdi da je procesor postavio dozvolu za korišćenja magistrale izlazi se iz petlje i ulazi u sledeću petlju u kojoj se adresa i upravljački signal čitanja po linijama magistrale šalju memoriji i čeka da memorija pošalje po linijama podataka magistrale pročitani podatak i po liniji magistrale upravljački signal završetka čitanja.

Po završetku čitanja izlazi se iz petlje i prelazi se na korak u kome se inkrementira sadržaj adresnog registra i dekrementira sadržaj registra broja reči za prenos. Kao rezultat u adresnom registru sa sada nalazi adresa memorijske lokacije sa koje treba pročitati sledeći podatak, a u registru broja reči za prenos broj preostalih reči za prenos iz memorije u periferiju. Pored toga postavlja se indikacija da registar podatka nije raspoloživ za upis iz memorije. Ovo je indikacija da upravljačka jedinica *uprav_per*, koja je odmah po prenošenju prethodnog podatka iz registra podatka u pomoćni registar podatka krenula sa njegovim upisom u periferiju, može kada završi sa njegovim upisom u periferiju, da prenese novi podatak najpre iz registar podatka u pomoćni registar podatka i da krene sa njegovim upisom u periferiju.

Potom se ponovo proverava da li je kontroler programskim putem zaustavljen i ukoliko je jeste, ukida se zahtev za korišćenje magistrale i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, proverava se da li je kao rezultat dekrementiranja sadržaj brojača reči postao nula, što je indikacija da je završen prenos podataka, ili je još uvek veći od nule, što je indikacija da treba produžiti sa prenosom podataka.

Ukoliko je završen prenos podataka, indikator *prenos završen* statusnog registra se postavlja na aktivnu vrednost i ukoliko je zadat režim rada sa generisanjem prekida, generiše se prekid. Pored toga se ukida zahtev za korišćenje magistrale i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko nije završen, prenos podataka se produžava na dva načina u zavisnosti od toga da li je zadat pojedinačni ili paketski prenos.

Ukoliko nije zadat paketski već pojedinačni prenos, ukida se zahtev za korišćenje magistrale i vraća na korak u kome se najpre čeka da registar podatka postane raspoloživ za upis podatka iz memorije, što će se desiti kada upravljačka jedinica *uprav_per* prenese prethodni podatak iz registra podatka u pomoćni registar podatka i krene sa njegovim upisom u periferiju. Tada se ponavlja već opisani postupak po kome se najpre postavlja dozvola za korišćenje magistrale i po dobijanju dozvole podatak upisuje iz memorije u registar podatka.

Ukoliko je zadat paketski prenos u petlji se proverava da li je kontroler programskim putem zaustavljen i da li je registar podatka raspoloživ za upis podatka iz memorije. Ukoliko je kontroler programskim putem zaustavljen, izlazi se iz petlje i vraća u početni korak u kome se čeka da kontroler bude ponovo startovan. Ukoliko kontroler nije programskim putem zaustavljen, u petlji se ostaje sve dok registar podatka ne postane raspoloživ za upis podatka iz memorije što će se desiti kada upravljačka jedinica *uprav_per* prenese prethodni podatak iz registra podatka u pomoćni registar podatka i krene sa njegovim upisom u periferiju.

Kada se utvrdi da je registar podatka raspoloživ za upis podatka iz memorije izlazi se iz petlje i ulazi u sledeću petlju u kojoj se adresa i upravljački signal čitanja po linijama

magistrale šalju memoriji i čeka da memorija pošalje po linijama podataka magistrale pročitani podatak i po liniji magistrale upravljački signal završetka čitanja.

Po završetku čitanja izlazi se iz petlje i prelazi se na korak u kome se inkrementira sadržaj adresnog registra, dekrementira sadržaj registra broja reči za prenos i ponavlja već opisani postupak na jedan od dva moguća načina u zavisnosti od toga da li je dekrementiranjem sadržaja registra broja reči dobijen sadržaj koju ukazuje da prenos treba završiti ili produžiti.

6.3.2.3.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 88) i dat u obliku dijagrama toka upravljačkih signala (slika 89), sekvence upravljačkih signala (tabela 30).

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 89). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova.

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

signali i

if uslov then signali.

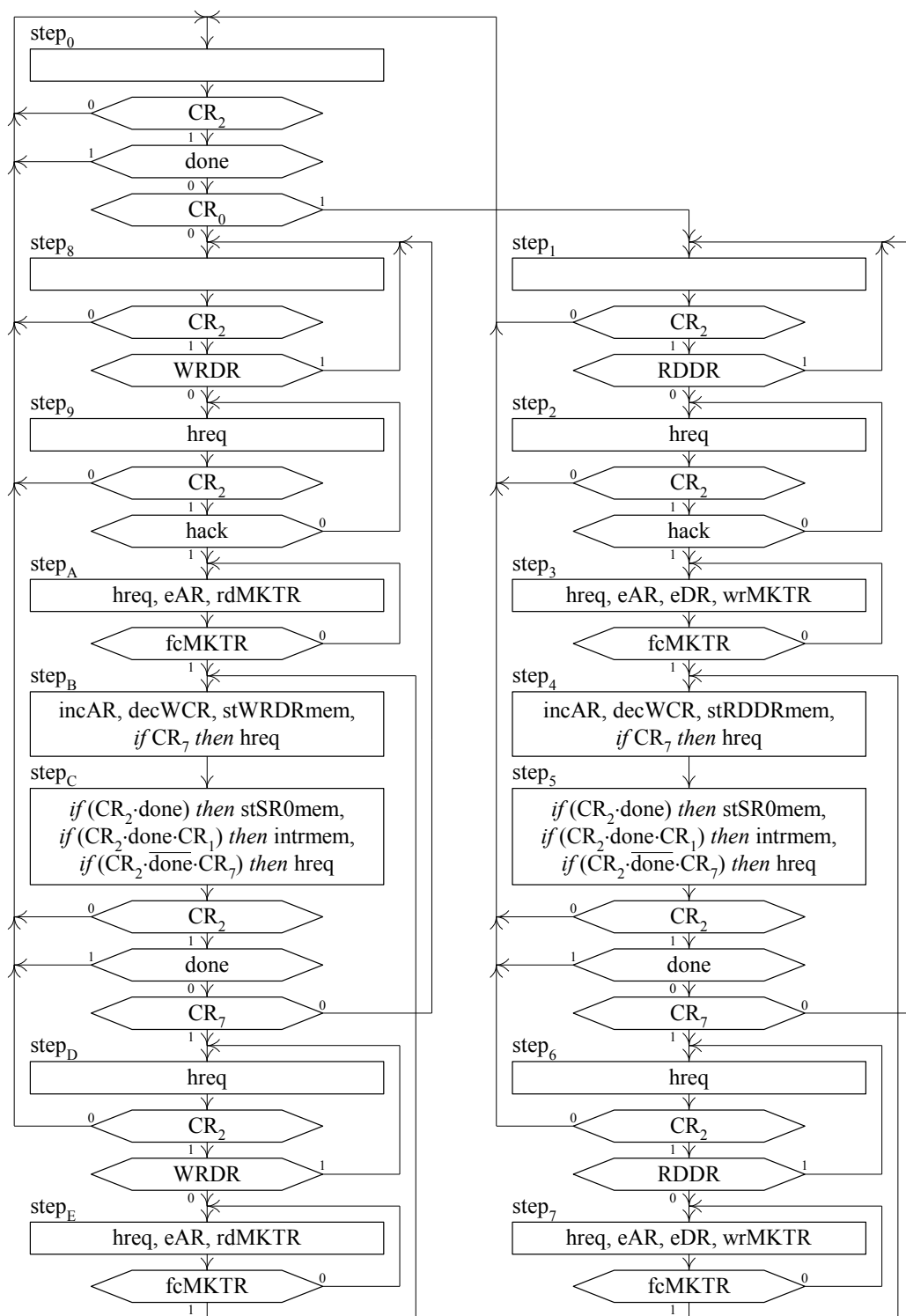
Prvi iskaz sadri spisak upravljačkih signala blokova operacione jedinice **oper** i određuje signale koji se bezuslovno generišu. Drugi iskaz sadrži uslov i spisak upravljačkih signala blokova operacione jedinice **oper** i određuje koji signali i pod kojim uslovima treba da budu generisani. Iskazi za skokove su oblika

br step_A i

br (if uslov then step_A).

Prvi iskaz sadrži korak na koji treba bezuslovno preći. Drugi 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.



Slika 89 Dijagram toka upravljačkih signala

Tabela 30 Sekvenca upravljačkih signala

! U koraku step₀ se ostaje sve dok je ili signal **CR₂** bloka *registri* neaktivan ili signal **done** bloka *registri* aktivan. Signal **CR₂** je neaktivan ili aktivan u zavisnosti od toga da li je ukoliko kontroler periferije zaustavljen ili startovan. Kontroler periferije se startuje programskim putem tako što upravljačka jedinica *uprav_bus* upisuje u upravljački registar CR_{7...0} bloka *registri* sadržaj koji na poziciji razreda CR₂ ima aktivnu vrednost. Upisivanjem aktivne vrednosti u razred CR₂ upravljačkog registra CR_{7...0} od strane upravljačke jedinice *uprav_bus* pokrenuću se upravljačke jedinice

uprav_mem i *uprav_per*. Prilikom startovanja upravljačka jedinica *uprav_bus* signalom **stRDDRbus** postavlja flip-flop RDDR na aktivnu vrednost. Signal **done** je aktivan ukoliko je sadržaj registra $WCR_{15...0}$ bloka *registri* nula. Ovo se dešava kada se programskim putem upisivanjem u registar $WCR_{15...0}$ za veličini bloka zada vrednost nula ili po završetku prenosa bloka podataka. Od preostalih razreda registra $CR_{7...0}$ za korak $step_0$ je bitan razred CR_0 , jer korak na koji se prelazi zavisi od vrednosti signala CR_0 . Na poziciji razreda CR_0 je aktivna ili neaktivna vrednost u zavisnosti od toga da li je zadat prenos iz periferije u memoriju ili iz memorije u periferiju, respektivno. U zavisnost od toga da li je aktivna ili neaktivna vrednost signala CR_0 iz koraka $step_0$ se prelazi na korak $step_1$ ili $step_8$, respektivno. !

$step_0$ $br (if (\overline{CR_2} \cdot \overline{done} \cdot CR_0) then step_1),$
 $br (if (CR_2 \cdot \overline{done} \cdot \overline{CR_0}) then step_8),$

! U korak $step_1$ se dolazi iz koraka $step_0$ ili $step_5$. Ukoliko je neaktivna vrednost signala CR_2 , što znači da je programskim putem upisivanjem neaktivne vrednosti u razred CR_2 upravljačkog registra $CR_{7...0}$ zaustavljen rad kontrolera, prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_1$ ili se prelazi u korak $step_2$ u zavisnosti od toga da li je aktivna ili neaktivna vrednost signal **RDDR** bloka *registri*. Aktivna vrednost signala **RDDR** je indikacija da upravljačka jedinica *uprav_per* još uvek nije prenela podatak iz registra $DRAUX_{7...0}$ bloka *registri* u registar $DR_{7...0}$ bloka *registri* i da upravljačka jedinica *uprav_mem* ne može da krene sa prenošenjem podatka iz registra $DR_{7...0}$ u memoriju. Time se obezbeđuje da upravljačka jedinica *uprav_mem* zbog aktivne vrednosti signala **RDDR** čeka u koraku $step_1$. Tek kada upravljačka jedinica *uprav_per* prenese podatak, najpre, iz periferije u registar $DRAUX_{7...0}$, a zatim iz registra $DRAUX_{7...0}$ u registar $DR_{7...0}$ i na kraju signalom **clRDDRper** postavi flip-flop RDDR na neaktivnu vrednost, upravljačka jedinica *uprav_mem* prelazi iz koraka $step_1$ u korak $step_2$. !

$step_1$ $br (if \overline{CR_2} then step_0),$
 $br (if (CR_2 \cdot \overline{RDDR}) then step_2);$

! U korak $step_2$ se dolazi iz koraka $step_1$. U koraku $step_2$ se generiše aktivna vrednost signala **hreq** kojom se od procesora **CPU** traži dozvola za korišćenje magistrale **BUS**. Ukoliko je neaktivna vrednost signala CR_2 , prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala CR_2 , ili se ostaje u koraku $step_2$ ili se prelazi u korak $step_3$ u zavisnosti od toga da li je neaktivna ili aktivna vrednost signala **hack** procesora **CPU**. Neaktivna vrednost signala **hack** je indikacija procesor još uvek koristi magistralu i da upravljačka jedinica *uprav_mem* ne može da krene sa prenošenjem podatka iz registra $DR_{7...0}$ u memoriju. Time se obezbeđuje da upravljačka jedinica *uprav_mem* zbog neaktivne vrednosti signala **hack** čeka u koraku $step_2$. Tek kada procesor završi sa korišćenjem magistrale i signal **hack** postavi na aktivnu vrednost, upravljačka jedinica *uprav_mem* prelazi iz koraka $step_2$ u korak $step_3$. !

$step_2$ **hreq,**
 $br (if \overline{CR_2} then step_0),$
 $br (if (CR_2 \cdot \overline{hack}) then step_3);$

! U korak $step_3$ se dolazi iz koraka $step_2$. U koraku $step_3$ se obavlja prenos podatka iz registra $DR_{7...0}$ u memoriju. Kako usvojeni interfejs između procesora **CPU** i kontrolera **DMA** predviđa da kontroler **DMA** drži aktivnu vrednost signala **hreq** sve vreme dok koristi magistralu, to se u koraku $step_3$ drži aktivna vrednost signala **hreq**. Pored toga u koraku $step_3$ se aktivnim vrednostima signala **eAR** i **eDR** bloka *registri* sadržaji registara $AR_{15...0}$ i $DR_{7...0}$ puštaju na adresne linije $ABUS_{15...0}$ i linije $DBUS_{7...0}$ podataka magistrale **BUS** i aktivnom vrednošću signala **wrMKTR** formira aktivna vrednost signala na upravljačkoj liniji **WRBUS** magistrale **BUS**, čime se u memoriji **MEM** startuje operacija upisa. U koraku $step_3$ se ostaje onoliko perioda signala takta koliko je neophodno da se upis realizuje. Sve vreme dok je procesor u koraku $step_3$ adresa i podatak se prisutni na linijama $ABUS_{15...0}$ i $DBUS_{7...0}$ i na liniji **WRBUS** je aktivna vrednost. Po završenom upisu memorija **MEM** generiše aktivnu vrednost signala na upravljačkoj liniji **FCBUS** magistrale **BUS**, pa signal **fcMKTR** bloka *interfejs* postaje aktivan. U koraku $step_3$ se proverava vrednost signala **fcMKTR** koji vrednostima 0 i 1 ukazuje da upis nije završen i da je upis završen, respektivno. Ukoliko upis nije završen ostaje se u koraku $step_3$, dok se u suprotnom slučaju prelazi na korak $step_4$. !

step₃ **hreq, eAR, eDR, wrMKTR,**
 br (if (fcMKTR) then step₄);

! U korak step₄ se dolazi iz koraka step₃ ili step₇ ostaje jedna perioda signala takta i prelazi u korak step₅. U koraku step₄ se generišu aktivne vrednosti signala **incAR**, **decWCR** i **stRDDRmem**. Signalom **incAR** se inkrementira sadržaj adresnog brojača AR_{15...0}, tako da njegova vrednost sadrži adresu memorijske lokacije u koju treba upisati sledeći podatak. Signalom **decWCR** se dekrementira sadržaj brojača WCR_{15...0}, tako da njegova trenutna vrednost ukazuje koliko još podataka treba preneti. Signalom **stRDDRmem** se flip-flop RDDR postavlja na aktivnu vrednost. Aktivna vrednost signala flip-flopa RDDR je indikacija da je upravljačka jedinica *uprav_mem* prenela sadržaj registra DR_{7...0} u memoriju i da upravljačka jedinica *uprav_per* može novi podatak da prenese iz registra DRAUX_{7...0} u registar DR_{7...0}. Pri aktivnoj vrednosti signala **CR₇** generiše se aktivna vrednost signala **hreq**. Neaktivna i aktivna vrednost signala **CR₇** određuju da li je zadat režim rada sa pojedinačnim ili paketskim prenosom podataka, respektivno. Ovo je posledica usvojenog interfejsa između procesora **CPU** i kontrolera **DMA** koji predviđa da je pri paketskom prenosu magistrala u posedu kontrolera **DMA** dok se ne obavi prenos kompletnog bloka podataka što se obezbeđuje držanjem aktivne vrednost signala **hreq**. !

step₄ **incAR, decWCR, stRDDRmem,**
 if CR₇ then hreq,
 br step₅;

! U korak step₅ se dolazi iz koraka step₄, ostaje jedna perioda signala takta i prelazi u korak step₀, step₁ ili step₅. U koji će se od ovih korak preći zavisi od vrednosti signala **CR₂**, **done**, **CR₁** i **CR₇** bloka *registri*. Ukoliko je neaktivna vrednost signala **CR₂**, prelazi se u korak step₀. Ukoliko je aktivna vrednost signala **CR₂**, korak u koji će se preći najpre zavisi od vrednosti signala **done**. Neaktivna i aktivna vrednost signala **done** se formiraju u zavisnosti od toga da li je sadržaj registra WCR_{15...0} bloka *registri* različit od nule ili je nula i služe kao indikacija da prenos podataka nije ili jeste kompletiran, respektivno. Pri aktivnoj vrednosti signala **done** prenos podataka se završava prelaskom na korak step₀. Pri aktivnim vrednostima signala **CR₂** i **done** generiše se signal **stSR0mem** bloka *registri*. Signalom **stSR0mem** se upisuje aktivna vrednost u razred SR₀ statusnog registra SR_{7...0} bloka *registri*. Ovaj razred odgovara bitu spremnost i njegova aktivna vrednost služi kao indikacija procesoru da je prenos kompletiran i da procesor može programskim putem da zaustavi kontroler periferije. Ukoliko je pri aktivnim vrednostima signala **CR₂** i **done** i signal **CR₁** aktivan, generiše se signal **intrmem** bloka *interfejs*. Ovim signalom se pokreće generisanje signala prekida, da bi processor skočio na odgovarajuću prekidnu rutinu i u okviru nje programskim putem zaustavio kontroler periferije. Prenos podataka se nastavlja prelaskom iz koraka step₅ u korak step₁ ili step₆ kada je kontroler periferije startovan i kada je zadata veličina bloka različita od nule, na šta ukazuju aktivna i neaktivna vrednost signala **CR₂** i **done**, respektivno. U korak step₁ ili step₆ se prelazi u zavisnosti od toga da li je vrednostima 0 i 1 signala **CR₇** zadat pojedinačni ili paketski prenos bloka podataka, respektivno. Pored toga pri aktivnoj vrednosti signala **CR₇** generiše se aktivna vrednost signala **hreq**. Ovo je posledica usvojenog interfejsa između procesora **CPU** i kontrolera **DMA** koji predviđa da je pri paketskom prenosu magistrala u posedu kontrolera **DMA** dok se ne obavi prenos kompletnog bloka podataka što se obezbeđuje držanjem aktivne vrednost signala **hreq**. !

step₅ *br (if CR₂ then step₀),*
 if (CR₂·done) then stSR0mem,
 if (CR₂·done·CR₁) then intrmem,
 br (if (CR₂·done) then step₀);
 br (if (CR₂·done · CR₇) then step₁),
 if (CR₂· done · CR₇) then hreq,
 br (if (CR₂· done · CR₇) then step₆),

! U korak step₆ se dolazi iz koraka step₅ jedino kada je zadat paketski prenos bloka podataka. U koraku step₆ se aktivnom vrednošću signala **hreq** obezbeđuje da magistrala ostaje u posedu kontrolera do kompletiranja paketskog prenosa bloka podataka. Pored toga na isti način kao i u koraku step₁ se

ukoliko je neaktivna vrednost signala **CR₂** prelazi u korak step₀ i ukoliko je aktivna vrednost signala **CR₂**, ili se ostaje u koraku step₆ ili se prelazi u korak step₇ u zavisnosti od toga da li je aktivna ili neaktivna vrednost signal **RDDR** bloka *registri*.

```
step6   hreq,
        br (if CR2 then step0),
        br (if (CR2·RDDR) then step7);
```

! U korak step₇ se dolazi iz koraka step₆. U koraku step₃ se obavlja prenos podatka iz registra DR_{7...0} u memoriju na isti način kao i u koraku step₃ i prelazi na korak step₄.

```
step7   hreq, eAR, eDR, wrMKTR,
        br (if (fcMKTR) then step4);
```

! U korak step₈ se dolazi iz koraka step₀ ili step_C. Ukoliko je neaktivna vrednost signala **CR₂**, što znači da je programskim putem upisivanjem neaktivne vrednosti u razred CR₂ upravljačkog registra CR_{7...0} zaustavljen rad kontrolera, prelazi se u korak step₀. Ukoliko je aktivna vrednost signala **CR₂**, ili se ostaje u koraku step₈ ili se prelazi u korak step₉ u zavisnosti od toga da li je aktivna ili neaktivna vrednost signal **WRDR** bloka *registri*. Aktivna vrednost signala **WRDR** je indikacija da upravljačka jedinica *uprav_per* još uvek nije prenela prethodni podatak iz registra DR_{7...0} bloka *registri* u registar DRAUX_{7...0} bloka *registri* i da upravljačka jedinica *uprav_mem* ne može da pređe na korak step₉ radi prenošenja sledećeg podatka iz memorije u registar DR_{7...0}. Tek kada upravljačka jedinica *uprav_per* prihvati podatak iz registra DR_{7...0} u registar DRAUX_{7...0}, krene sa njegovim prenošenjem iz registra DRAUX_{7...0} u periferiju i signalom **clWRDRper** postavi flip-flop WRDR na neaktivnu vrednost, upravljačka jedinica *uprav_mem* prelazi iz koraka step₈ u korak step₉. Na početku u registru DR_{7...0} još uvek nema podatka i signal **WRDR** je neaktivan, pa upravljačka jedinica *uprav_mem* odmah prelazi iz koraka step₈ u korak step₉. !

```
step8   br (if CR2 then step0),
        br (if (CR2·WRDR) then step9);
```

! U korak step₉ se dolazi iz koraka step₈. U koraku step₉ se generiše aktivna vrednost signala **hreq** kojom se od procesora **CPU** traži dozvola za korišćenje magistrale **BUS**. Ukoliko je neaktivna vrednost signala **CR₂**, prelazi se u korak step₀. Ukoliko je aktivna vrednost signala **CR₂**, ili se ostaje u koraku step₂ ili se prelazi u korak step_A u zavisnosti od toga da li je neaktivna ili aktivna vrednost signala **hack** procesora **CPU**. Neaktivna vrednost signala **hack** je indikacija procesor još uvek koristi magistralu i da upravljačka jedinica *uprav_mem* ne može da krene sa prenošenjem podatka iz memorije u registar DR_{7...0}. Time se obezbeđuje da upravljačka jedinica *uprav_mem* zbog neaktivne vrednosti signala **hack** čeka u koraku step₉. Tek kada procesor završi sa korišćenjem magistrale i signal **hack** postavi na aktivnu vrednost, upravljačka jedinica *uprav_mem* prelazi iz koraka step₉ u korak step_A. !

```
step9   hreq,
        br (if CR2 then step0),
        br (if (CR2·hack) then stepA);
```

! U korak step_A se dolazi iz koraka step₉. U koraku step₉ se obavlja prenos podatka iz memorije u registar DR_{7...0}. Kako usvojeni interfejs između procesora **CPU** i kontrolera **DMA** predviđa da kontroler **DMA** drži aktivnu vrednost signala **hreq** sve vreme dok koristi magistralu, to se u koraku step_A drži aktivna vrednost signala **hreq**. Pored toga u koraku step_A se aktivnom vrednošću signala **eAR** bloka *registri* sadržaj registra AR_{15...0} pušta na adresne linije ABUS_{15...0} magistrale **BUS** i aktivnom vrednošću signala **rdMKTR** formira aktivna vrednost signala na upravljačkoj liniji **RDBUS** magistrale **BUS**, čime se u memoriji **MEM** startuje operacija čitanja. U koraku step_A se ostaje onoliko perioda signala takta koliko je neophodno da se čitanje realizuje. Sve vreme dok je processor u koraku step_A adresa je prisutna na linijama ABUS_{15...0} i na liniji **RDBUS** aktivna vrednost. Po završenom čitanju memorija **MEM** otvara bafere sa tri stanja i na linije podataka DBUS_{7...0} magistrale pušta pročitane vrednosti i generiše aktivnu vrednost signala na upravljačkoj liniji **FCBUS** magistrale **BUS**. U koraku step_A se proverava vrednost signala **fcMKTR** bloka *registri*

koji vrednostima 0 i 1 ukazuje da upis nije završen i da je upis završen, respektivno. Ukoliko upis nije završen ostaje se u koraku step_A , dok se u suprotnom slučaju prelazi na korak step_B . !

step_A **hreq, eAR, rdMKTR,**
 br (if fcMKTR then step_B);

! U korak step_B se dolazi iz koraka step_A ili step_E ostaje jedna perioda signala takta i prelazi u korak step_C . U koraku step_B se generišu aktivne vrednosti signala **incAR**, **decWCR** i **stWRDRmem**. Signalom **incAR** se inkrementira sadržaj adresnog brojača $\text{AR}_{15...0}$, tako da njegova vrednost sadrži adresu memorijske lokacije sa koje treba pročitati sledeći podatak. Signalom **decWR** se dekrementira sadržaj brojača $\text{WCR}_{15...0}$, tako da njegova trenutna vrednost ukazuje koliko još podataka treba preneti. Signalom **stWRDRmem** se flip-flop WRDR postavlja na aktivnu vrednost. Aktivna vrednost signala **WRDR** je indikacija da je upravljačka jedinica *uprav_mem* prenela podatak iz memorije u registar $\text{DR}_{7...0}$ i da upravljačka jedinica *uprav_per* može da prihvati podatak iz registra $\text{DR}_{7...0}$ u registar $\text{DRAUX}_{7...0}$ i krene sa njegovim slanjem iz registra $\text{DRAUX}_{7...0}$ u periferiju. Pri aktivnoj vrednosti signala **CR₇** generiše se aktivna vrednost signala **hreq**. Neaktivna i aktivna vrednost signala **CR₇** određuju da li je zadat režim rada sa pojedinačnim ili paketskim prenosom podataka, respektivno. Ovo je posledica usvojenog interfejsa između procesora **CPU** i kontrolera **DMA** koji predviđa da je pri paketskom prenosu magistrala u posedu kontrolera **DMA** dok se ne obavi prenos kompletnog bloka podataka što se obezbeđuje držanjem aktivne vrednost signala **hreq**. !

step_B **incAR, decWCR, stWRDRmem,**
 if CR₇ then hreq,
 br step_C ;

! U korak step_C se dolazi iz koraka step_B , ostaje jedna perioda signala takta i prelazi u korak step_0 , step_8 ili step_D . U koji će se od ovih korak preći zavisi od vrednosti signala **CR₂**, **done**, **CR₁** i **CR₇** bloka *registri*. Ukoliko je neaktivna vrednost signala **CR₂**, prelazi se u korak step_0 . Ukoliko je aktivna vrednost signala **CR₂**, korak u koji će se preći najpre zavisi od vrednosti signala **done**. Neaktivna i aktivna vrednost signala **done** se formiraju u zavisnosti od toga da li je sadržaj registra $\text{WCR}_{15...0}$ bloka *registri* različit od nule ili je nula i služe kao indikacija da prenos podataka nije ili jeste kompletiran, respektivno. Pri aktivnoj vrednosti signala **done** prenos podataka se završava prelaskom na korak step_0 . Pri aktivnim vrednostima signala **CR₂** i **done** generiše se signal **stSR0mem** bloka *registri*. Signalom **stSR0mem** se upisuje aktivna vrednost u razred SR_0 statusnog registra $\text{SR}_{7...0}$ bloka *registri*. Ovaj razred odgovara bitu spremnost i njegova aktivna vrednost služi kao indikacija procesoru da je prenos kompletiran i da procesor može programskim putem da zaustavi kontroler periferije. Ukoliko je pri aktivnim vrednostima signala **CR₂** i **done** i signal **CR₁** aktivan, generiše se signal **intrmem** bloka *interfejs*. Ovim signalom se pokreće generisanje signala prekida, da bi processor skočio na odgovarajuću prekidnu rutinu i u okviru nje programskim putem zaustavio kontroler periferije. Prenos podataka se nastavlja prelaskom iz koraka step_C u korak step_8 ili step_D kada je kontroler periferije startovan i kada je zadata veličina bloka različita od nule, na šta ukazuju aktivna i neaktivna vrednost signala **CR₂** i **done**, respektivno. U korak step_8 ili step_D se prelazi u zavisnosti od toga da li je vrednostima 0 i 1 signala **CR₇** zadat pojedinačni ili paketski prenos bloka podataka, respektivno. Pored toga pri aktivnoj vrednosti signala **CR₇** generiše se aktivna vrednost signala **hreq**. Ovo je posledica usvojenog interfejsa između procesora **CPU** i kontrolera **DMA** koji predviđa da je pri paketskom prenosu magistrala u posedu kontrolera **DMA** dok se ne obavi prenos kompletnog bloka podataka što se obezbeđuje držanjem aktivne vrednost signala **hreq**. !

step_C *br (if $\overline{\text{CR}_2}$ then step_0),*
 if ($\text{CR}_2 \cdot \text{done}$) then stSR0mem,
 if ($\text{CR}_2 \cdot \text{done} \cdot \text{CR}_1$) then intrmem,
 br (if ($\text{CR}_2 \cdot \text{done}$) then step_0),
 br (if ($\text{CR}_2 \cdot \text{done} \cdot \text{CR}_7$) then step_8),
 if ($\text{CR}_2 \cdot \text{done} \cdot \text{CR}_7$) then hreq,
 br (if ($\text{CR}_2 \cdot \text{done} \cdot \text{CR}_7$) then step_D),

! U korak $step_D$ se dolazi iz koraka $step_C$. U koraku $step_D$ se aktivnom vrednošću signala **hreq** obezbeđuje da magistrala ostaje u posedu kontrolera do kompletiranja paketskog prenosa bloka podataka. Ukoliko je neaktivna vrednost signala **CR₂**, što znači da je programskim putem upisivanjem neaktivne vrednosti u razred **CR₂** upravljačkog registra **CR_{7...0}** zaustavljen rad kontrolera, prelazi se u korak $step_0$. Ukoliko je aktivna vrednost signala **CR₂**, ili se ostaje u koraku $step_D$ ili se prelazi u korak $step_E$ u zavisnosti od toga da li je aktivna ili neaktivna vrednost signal **WRDR** bloka *registri*. Aktivna vrednost signala **WRDR** je indikacija da upravljačka jedinica *uprav_per* još uvek nije prenela prethodni podatak iz registra **DR_{7...0}** bloka *registri* u registar **DRAUX_{7...0}** bloka *registri* i da upravljačka jedinica *uprav_mem* ne može da pređe na korak $step_E$ radi prenošenja sledećeg podatka iz memorije u registar **DR_{7...0}**. Tek kada upravljačka jedinica *uprav_per* prihvati podatak iz registra **DR_{7...0}** u registar **DRAUX_{7...0}**, krene sa njegovim prenošenjem iz registra **DRAUX_{7...0}** u periferiju i signalom **clWRDRper** postavi flip-flop **WRDR** na neaktivnu vrednost, upravljačka jedinica *uprav_mem* prelazi iz koraka $step_D$ u korak $step_E$. !

```

step_D   hreq,
          br (if CR2 then step_0),
          br (if (CR2 · WRDR) then step_E);

```

! U korak $step_E$ se dolazi iz koraka $step_D$. U koraku $step_E$ se obavlja prenos podatka iz memorije u registar **DR_{7...0}**. Kako usvojeni interfejs između procesora **CPU** i kontrolera **DMA** predviđa da kontroler **DMA** drži aktivnu vrednost signala **hreq** sve vreme dok koristi magistralu, to se u koraku $step_A$ drži aktivna vrednost signala **hreq**. Pored toga u koraku $step_E$ se aktivnom vrednošću signala **eAR** bloka *registri* sadržaj registra **AR_{15...0}** pušta na adresne linije **ABUS_{15...0}** magistrale **BUS** i aktivnom vrednošću signala **rdMKTR** formira aktivna vrednost signala na upravljačkoj liniji **RDBUS** magistrale **BUS**, čime se u memoriji **MEM** startuje operacija čitanja. U koraku $step_E$ se ostaje onoliko perioda signala takta koliko je neophodno da se čitanje realizuje. Sve vreme dok je processor u koraku $step_E$ adresa je prisutna na linijama **ABUS_{15...0}** i na liniji **RDBUS** je aktivna vrednost. Po završenom čitanju memorija **MEM** otvara bafere sa tri stanja i na linije podataka **DBUS_{7...0}** magistrale pušta pročitane vrednosti i generiše aktivnu vrednost signala na upravljačkoj liniji **FCBUS** magistrale **BUS**. U koraku $step_E$ se proverava vrednost signala **fcMKTR** bloka *registri* koji vrednostima 0 i 1 ukazuje da upis nije završen i da je upis završen, respektivno. Ukoliko upis nije završen ostaje se u koraku $step_E$, dok se u suprotnom slučaju prelazi na korak $step_B$. !

```

step_E   hreq, eAR, rdMKTR,
          br (if fcMKTR then step_B);

```

6.3.2.3.3 Struktura upravljačke jedinice

Struktura upravljačke jedinice ožičene realizacije je prikazana na slici 90. Upravljačka jedinica se sastoji od sledećih blokova:

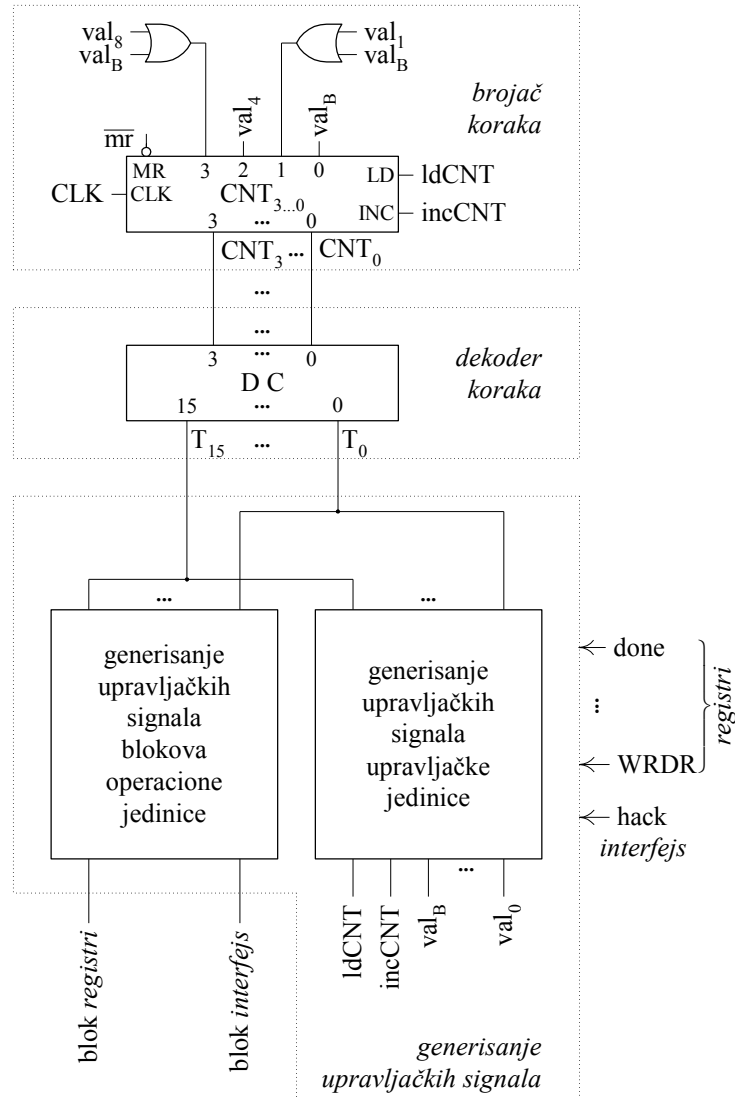
- blok *brojač koraka*,
- blok *dekoder koraka* i
- blok *generisanje upravljačkih signala*.

Blok *brojač koraka* sadrži brojač **CNT_{3...0}**. Brojač **CNT_{3...0}** svojom trenutnom vrednošću obezbeđuje aktivne vrednosti određenih upravljačkih signala. Brojač **CNT_{3...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_{3...0}** za jedan čime se obezbeđuje sekvencijalno generisanje upravljačkih signala iz sekvence upravljačkih signala za upravljačku jedinicu ožičene realizacije (tabela 30). Ovaj

režim rada se realizuje neaktivnom vrednošću signala **ldCNT** i aktivnom vrednošću signala **incCNT**.



Slika 90 Struktura upravljačke jedinice *uprav_mem*

U režimu skoka pri pojavi signala takta vrši se upis nove vrednosti u brojač $CNT_{3...0}$ čime se obezbeđuje odstupanje od sekvencijalnog generisanja upravljačkih signala iz sekvence upravljačkih signala za upravljačku jedinicu ožičene realizacije (tabela 30). Ovaj režim rada se realizuje aktivnom vrednošću signala **ldCNT**, neaktivnom vrednošću signala **incCNT** i kombinacijom aktivnih i neaktivnih vrednosti signala **val₀**, **val₁**, **val₄**, **val₈** i **val_B** koja odgovara vrednosti koju treba upisati u brojač $CNT_{3...0}$.

U režimu čekanja pri pojavi signala takta ne menja se vrednost brojača $CNT_{3...0}$. Ovaj režim rada se obezbeđuje neaktivnim vrednostima signala **ldCNT** i **incCNT**.

Blok *dekoder koraka* sadrži dekode DC. Na ulaze dekodera DC vode se izlazi brojača $CNT_{3...0}$. Dekodovana stanja brojača $CNT_{3...0}$ pojavljuju se kao signali **T₀** do **T_F** na izlazima dekodera DC. Svakom koraku iz sekvence upravljačkih signala po koracima (tabela 30) dodeljeno je po jedno stanje brojača $CNT_{3...0}$ određeno vrednošću signala **T₀** do **T_F** i to koraku step₀ signal **T₀**, koraku step₁ signal **T₁**, itd.

Blok *generisanje upravljačkih signala* sadrži kombinacione mreže koje pomoću signala **T₀** do **T_E**, koji dolaze sa bloka *dekoder koraka*, signala logičkih uslova **done** do **WRDR**, koji

dolaze iz bloka *registri*, i **hack**, koji dolazi iz bloka *interfejs*, i saglasno sekvenci upravljačkih signala za upravljačku jedinicu ožičene realizacije (tabela 30) generišu dve grupe upravljačkih signala i to:

- upravljačke signale blokova operacione jedinice **oper** i
- upravljačke signale upravljačke jedinice *uprav_mem*.

Upravljački signali blokova operacione jedinice **oper** se daju posebno za blok *registri* i blok *interfejs*.

Upravljački signali bloka *registri* se generišu na sledeći način:

- $\text{stSR0mem} = \text{CR}_2 \cdot \text{done} \cdot T_5 + \text{CR}_2 \cdot \text{done} \cdot T_C$
- $\text{stRDDRmem} = T_4$
- $\text{stWRDRmem} = T_B$
- $\text{eAR} = T_3 + T_7 + T_A + T_E$
- $\text{eDR} = T_3 + T_7$
- $\text{incAR} = T_4 + T_B$
- $\text{decWR} = T_4 + T_B$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz blokova *registri* i *interfejs* operacione jedinice **oper** i to:

- **done** — blok *registri*,
- **CR₂** — blok *registri*,

Upravljački signali bloka *interfejs* se generišu na sledeći način:

- $\text{wrMKTR} = T_3 + T_7$
- $\text{rdMKTR} = T_A + T_E$
- $\text{intrmem} = \text{CR}_2 \cdot \text{done} \cdot \text{CR}_1 \cdot T_5 + \text{CR}_2 \cdot \text{done} \cdot \text{CR}_1 \cdot T_C$

Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz blokova *registri* i *interfejs* operacione jedinice **oper** i to:

- **done** — blok *registri*,
- **CR₂** — blok *registri*,
- **CR₁** — blok *registri*.

Upravljački signali upravljačke jedinice *uprav_mem* se generišu na sledeći način:

- $\text{ldCNT} = \text{val}_0 + \text{val}_1 + \text{val}_4 + \text{val}_8 + \text{val}_B$
- $\text{val}_0 = \overline{\text{CR}_2} \cdot T_1 + \overline{\text{CR}_2} \cdot T_2 + \overline{\text{CR}_2} \cdot T_5 + \text{CR}_2 \cdot \text{done} \cdot T_5 + \overline{\text{CR}_2} \cdot T_6 + \overline{\text{CR}_2} \cdot T_8 + \overline{\text{CR}_2} \cdot T_9 + \overline{\text{CR}_2} \cdot T_C + \text{CR}_2 \cdot \text{done} \cdot T_C + \overline{\text{CR}_2} \cdot T_D$
- $\text{val}_1 = \text{CR}_2 \cdot \text{done} \cdot \overline{\text{CR}_7} \cdot T_5$
- $\text{val}_4 = \text{fcMKTR} \cdot T_7$
- $\text{val}_8 = \text{CR}_2 \cdot \text{done} \cdot \overline{\text{CR}_0} \cdot T_0 + \text{CR}_2 \cdot \text{done} \cdot \overline{\text{CR}_7} \cdot T_C$
- $\text{val}_4 = \text{fcMKTR} \cdot T_E$
- $\text{incCNT} = \text{CR}_2 \cdot \text{done} \cdot \text{CR}_0 \cdot T_0 + \text{CR}_2 \cdot \overline{\text{RDDR}} \cdot T_1 + \text{CR}_2 \cdot \text{hack} \cdot T_2 + \text{fcMKTR} \cdot T_3 + T_4 + \text{CR}_2 \cdot \text{done} \cdot \text{CR}_7 \cdot T_5 + \text{CR}_2 \cdot \overline{\text{RDDR}} \cdot T_6 + \text{CR}_2 \cdot \overline{\text{WRDR}} \cdot T_8 + \text{CR}_2 \cdot \text{hack} \cdot T_9 + \text{fcMKTR} \cdot T_A + T_B + \text{CR}_2 \cdot \text{done} \cdot \text{CR}_7 \cdot T_C + \text{CR}_2 \cdot \overline{\text{WRDR}} \cdot T_D$

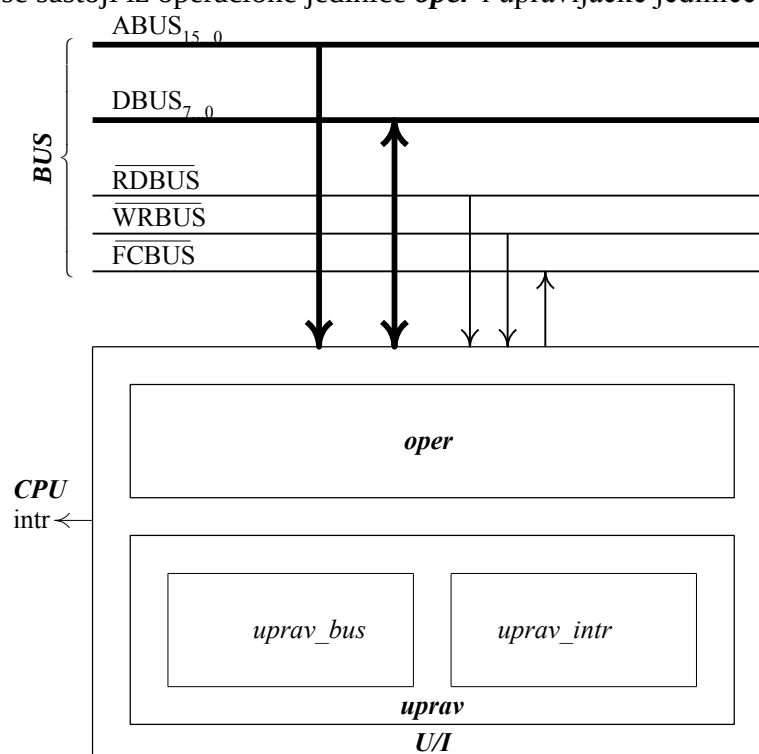
Pri njihovom generisanju koriste se sledeći signali logičkih uslova koji dolaze iz blokova *registri* i *interfejs* operacione jedinice **oper** i to:

- **done** — blok *registri*,

- **CR₀** — blok registri,
- **CR₂** — blok registri,
- **CR₇** — blok registri,
- **fcMKTR** — blok interfejs,
- **RDDR** — blok registri,
- **WRDR** — blok registri,
- **hack** — blok interfejs.

6.4 KONTROLER ZA GENERISANJE IMPULSA

U ovoj glavi se daje organizacija kontrolera za generisanje impulsa **TMR** (slika 91). Kontroler **TMR** se sastoji iz operacione jedinice **oper** i upravljačke jedinice **uprav**.



Slika 91 Organizacija generatora impulsa **TMR**

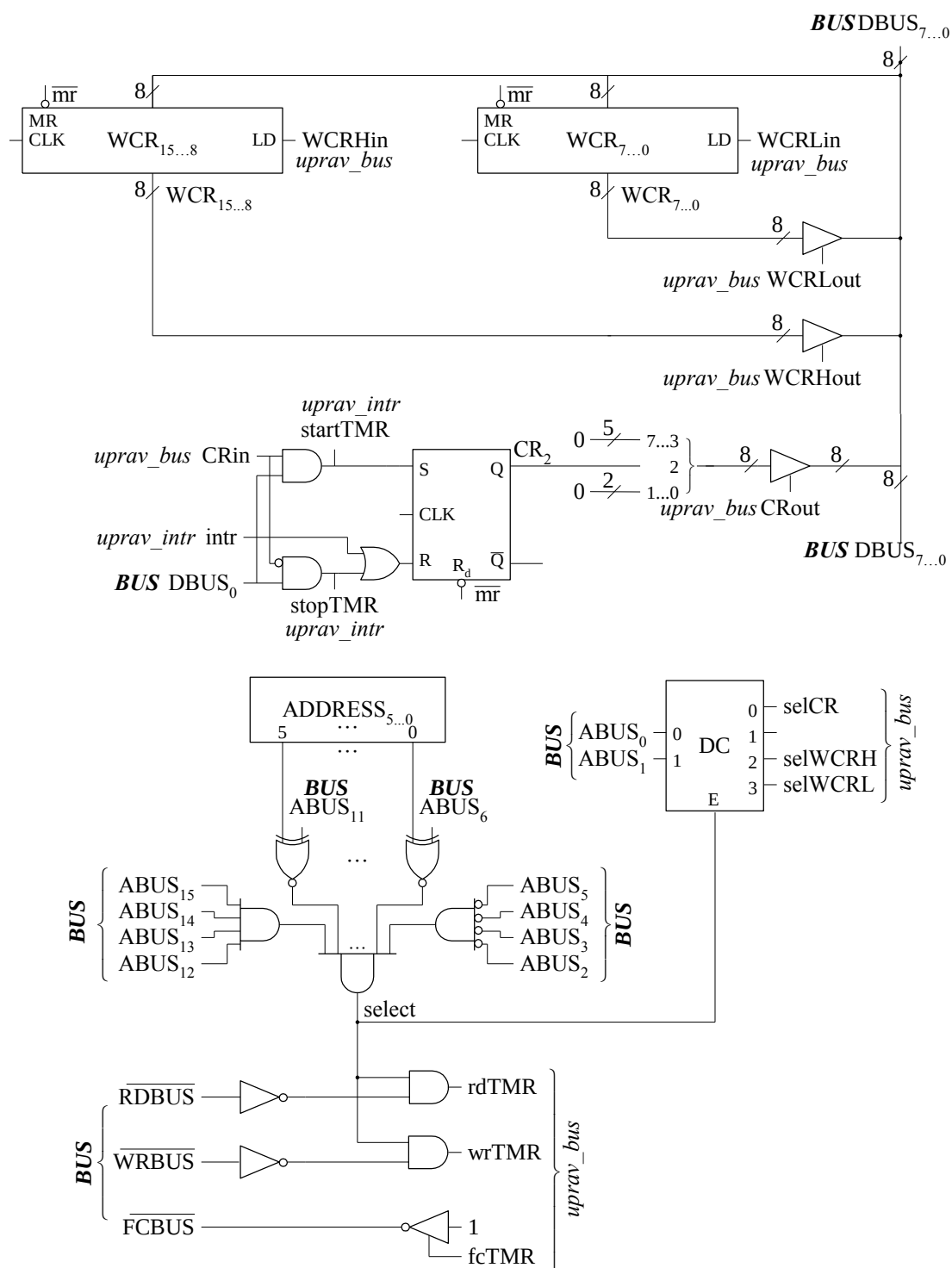
Operaciona jedinica **oper** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe za pamćenje binarnih reči, izvršavanje mikrooperacija i generisanje signala logičkih uslova. Upravljačka jedinica **uprav** je kompozicija kombinacionih i sekvencijalnih prekidačkih mreža koje služe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

6.4.1 Operaciona jedinica

Operaciona jedinica **oper** sadrži upravljačke registre CR₂ i WCR_{15...0} i kombinacione i sekvencijalne prekidačke mreže za realizaciju ciklusa na magistrali u kojima je kontroler sluga (slika 92).

Registar CR₂ je jednorazredni upravljački registar koji sadrži bit *start*. U registar CR₂ se, generisanjem aktivne vrednosti signala **CRin** upisuje sadržaj sa linije podataka DBUS₂ magistrale. Sadržaj registra CR₂ proširen nulama do dužine 8 bitova se pri aktivnoj vrednosti signala **CRout** propušta kroz bafere sa tri stanja na linije podataka DBUS_{7...0} magistrale.



Slika 92 Operaciona jedinica **oper**

Registar $WCR_{15...0}$ je 16-to razredni registar u kome se čuva vreme, zadato kao broj perioda signala takta, koje treba da protekne od trenutka startovanja kontrolera do trenutka generisanja impulsa trajanja jedna perioda signala takta. Broj perioda signala takta se u registar $WCR_{15...0}$ upisuje programskim putem u dva ciklusa na magistrali. Signalom **WCRHin** se u 8 starijih razreda registra $WCR_{15...8}$ upisuje sadržaj sa linija podataka $DBUS_{7...0}$ magistrale, dok se signalom **WCRLin** u 8 mlađih razreda registra $WCR_{7...0}$ upisuje sadržaj sa linija podataka $DBUS_{7...0}$ magistrale. Sadržaj registra $WCR_{15...0}$ se čita programskim putem u dva ciklusa na magistrali. Signalom **WCRHout** se 8 starijih razreda

registra $WCR_{15...8}$ propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale, dok se signalom **WCRLout** 8 mladih razreda registra $WCR_{7...0}$ propušta kroz bafere sa tri stanja na linije podataka $DBUS_{7...0}$ magistrale.

Kombinacione i sekvencijalne mreže za realizaciju ciklusa na magistrali u kojima je kontroler sluga formiraju signale **rdTMR** i **wrTMR** upravljačke jedinice *uprav_bus* i **FCBUS** magistrale **BUS** (slika 92). Signali **rdTMR** i **wrTMR** se formiraju na osnovu signala $ABUS_{15..0}$ sa adresnih linija magistrale i signala **RDBUS** i **WRBUS** sa upravljačkih linija magistrale. Signal **FCBUS** se formira na osnovu signala **fcTMR** upravljačke jedinice *uprav_bus*.

Pri realizaciji ciklusa čitanja na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$ i upravljačku liniju **RDBUS** magistrale i na njih izbacuje adresu i aktivnu vrednost signala čitanja, respektivno, čime se u kontroleru kao slugi startuje čitanje adresiranog registra. Kontroler po završenom čitanju otvara bafere sa tri stanja za linije podataka $DBUS_{7..0}$ i upravljačku liniju **FCBUS** magistrale i na njih izbacuje sadržaj adresiranog registra i aktivnu vrednost signala završetka operacije čitanja u kontroleru. Procesor prihvata sadržaj sa linija podataka i zatvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$ i upravljačku liniju **RDBUS** magistrale, dok kontroler zatvara bafere sa tri stanja za linije podataka $DBUS_{7..0}$ i upravljačku liniju **FCBUS** magistrale.

Pri realizaciji ciklusa upisa na magistrali procesor kao gazda otvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$, linije podataka $DBUS_{7..0}$ i upravljačku liniju **WRBUS** magistrale i na njih izbacuje adresu, podatak i aktivnu vrednost signala upisa, respektivno, čime se u kontroleru kao slugi startuje upis u adresirani registar. Kontroler po završenom upisu otvara bafere sa tri stanja za upravljačku liniju **FCBUS** magistrale i na nju izbacuje aktivnu vrednost signala završetka operacije upisa u kontroleru. Procesor zatvara bafere sa tri stanja za adresne linije $ABUS_{15..0}$, linije podataka $DBUS_{7..0}$ i upravljačku liniju **WRBUS** magistrale, dok kontroler zatvara bafere sa tri stanja za upravljačku liniju **FCBUS** magistrale.

Signali **rdTMR** i **wrTMR** imaju vrednosti upravljačkih signala **RDBUS** i **WRBUS** magistrale, respektivno, kada je aktivna vrednost signala **select** i neaktivne vrednosti kada je neaktivna vrednost signala **select**. Signal **rdTMR** je kao i signal **RDBUS** aktivan za vreme operacije čitanja nekog registra kontrolera, dok je signal **wrTMR** kao i signal **WRBUS** aktivan za vreme operacije upisa u neki od registara kontrolera.

Signal **select** ima aktivnu vrednost ukoliko se na adresnim linijama $ABUS_{15...0}$ magistrale nalazi adresa iz opsega adresa dodeljenih registrima kontrolera za generisanje impulsa **TMR**. Celokupan opseg adresa od 0000h do FFFFh podeljen je na opseg adresa od 0000h do EFFFh, koje su dodeljene memoriji **MEM**, i opseg adresa od F000h do FFFFh, koje su dodeljene registrima po svim kontrolerima periferija. Opseg adresa koje su dodeljene registrima po kontrolerima periferija je predviđen za adresiranje registara u najviše 64 kontrolera i to najviše 64 registra unutar određenog kontrolera. Pri tome prilikom adresiranja nekog od registara kontrolera sadržaj na adresnim linijama $ABUS_{15...0}$ magistrale ima sledeću strukturu: bitovi $ABUS_{15...12}$ su sve jedinice, bitovi $ABUS_{11...6}$ određuju broj kontrolera i bitovi $ABUS_{5...0}$ adresu registra unutar kontrolera. Broj kontrolera periferije za određeni kontroler periferije se postavlja mikroprekidačima na jednu od vrednosti u opsegu 0 do 63. Registrima CR_2 , $WCR_{15...8}$ i $WCR_{7...0}$ su unutar opsega adresa datog kontrolera dodeljene adrese 0, 2 i 3, pa

bitovi $ABUS_{5...2}$ mora da budu nule dok se njihovo pojedinačno adresiranje realizuje bitovima $ABUS_1$ i $ABUS_0$.

Na osnovu usvojene strukture adresa signal **select** ima aktivnu vrednost ukoliko su na adresnim linijama $ABUS_{15...12}$ magistrale sve jedinice, na adresnim linijama $ABUS_{11...6}$ se nalazi vrednost koja odgovara vrednosti postavljenoj mikroprekidačima i na adresnim linijama $ABUS_{5...2}$ sve nule. Signali **selCR**, **selWCRH** i **selWCRL** se dobijaju na izlazima 0, 2 i 3 dekodera DC na osnovu vrednosti signala $ABUS_1$, $ABUS_0$ i **select** sa ulaza 1, 0 i E, respektivno. Pri neaktivnoj vrednosti signala **select** i signali **selCR**, **selWCRH** i **selWCRL** su neaktivni. Pri aktivnoj vrednosti signala **select** jedan od signala **selCR**, **selWCRH** i **selWCRL** postaje aktivan i to signal određen binarnom vrednošću signala $ABUS_1$ i $ABUS_0$.

Signal **select** ima neaktivnu vrednost ukoliko se na adresnim linijama $ABUS_{15...0}$ magistrale ne nalazi adresa iz opsega adresa dodeljenih registrima kontrolera periferije, pri čemu to može da bude ili adresa memorijske lokacije ili registra iz nekog drugog kontrolera periferije. Tada se formiraju neaktivne vrednosti signala **rdTMR** i **wrTMR** bez obzira na vrednosti signala **RDBUS** i **WRBUS**, respektivno.

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

6.4.2 Upravljačka jedinica

Upravljačka jedinica **uprav** se sastoji iz dva dela:

- upravljačke jedinice magistrale *uprav_bus* i
- upravljačke jedinice generisanja impulsa *uprav_intr*.

Upravljačka jedinica *uprav_bus* omogućuje da processor **CPU** kao gazda realizuje u kontroleru kao slugi upisivanje sadržaja u registre i čitanje sadržaja iz registara. Upisivanjem u upravljački registar sadržaja koji na poziciji bita *start* ima vrednost 1 startuje upravljačka jedinica *uprav_intr*, koja generiše impuls **intr** po odbrojanju onoliko perioda signala takta koliko je to zadato sadržajem registra $WCR_{15...0}$. Upisivanjem u upravljački registar sadržaja koji na poziciji bita *start* ima vrednost 0, zaustavlja se upravljačka jedinica *uprav_intr*.

Struktura i opis upravljačkih jedinica *uprav_bus* i *uprav_intr* se daju u daljem tekstu.

6.4.2.1 Upravljačka jedinica magistrale

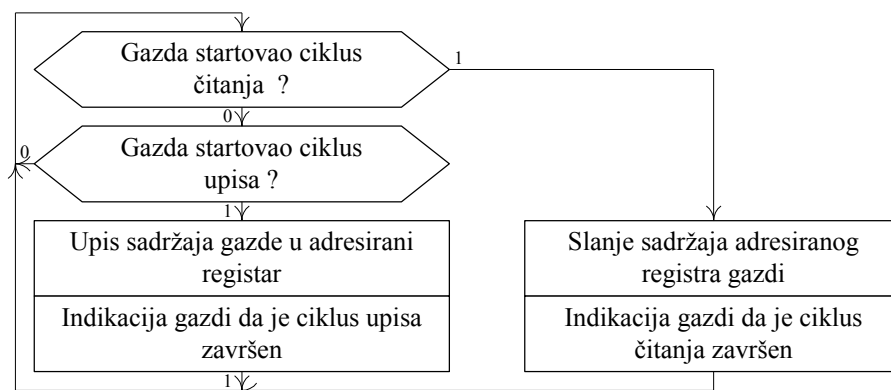
U ovom odeljku se daju dijagram toka operacija, algoritam generisanja upravljačkih signala i struktura upravljačke jedinice *uprav_bus*.

6.4.2.1.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 93). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.

U dijagrama toka operacija stalno se vrši provera da li je procesor startovao u kontroleru ciklus čitanja ili ciklus upisa, pri čemu se procesor ponaša kao gazda, a kontroler kao sluga. Ako nije startovan ni ciklus čitanja ni ciklus upisa nema izvršavanja mikrooperacija. Ako je

startovan jedan od ova dva ciklusa izvršavaju se mikrooperacije saglasno ciklusu koji je startovan.



Slika 93 Dijagram toka operacija

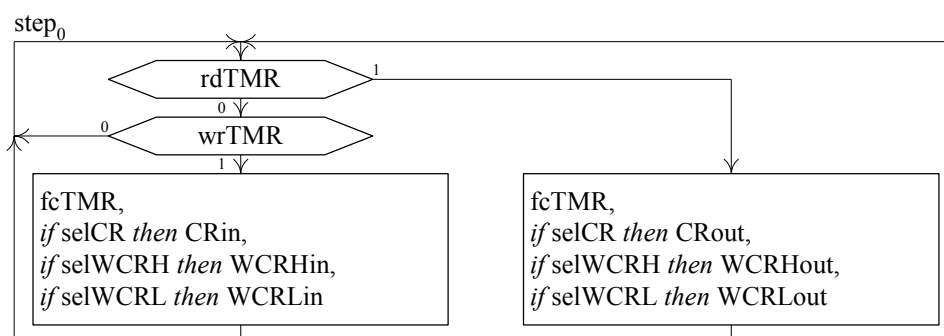
Ako je startovan ciklus čitanja, procesoru se kao gazdi šalje sadržaj adresiranog registra kontrolera i indikacija da je kontroler kao sluga završio čitanje.

Ako je startovan ciklus upisa, u adresirani registar kontrolera se upisuje sadržaj koji procesor kao gazda šalje i gazdi šalje indikacija da je kontroler kao sluga završio upis. U slučaju kada se sadržaj upisuje u upravljački registar, aktivnosti u kontroleru zavise od toga da li se kontroler startuje ili zaustavlja. Ukoliko se vrši startovanje kontrolera, pokreće se upravljačka jedinica *uprav_intr* koja generiše signale neophodne za odbrojavanje onoliko perioda signala takta koliko je zadato sadržajem registra $WCR_{15...0}$. Kada se odbroji zadati broj perioda signala takta, generiše se impuls trajanja jedna perioda signala takta koji se šalje procesoru kao signal prekida. Generisani impuls se koristi i za zaustavljanje daljeg odbrojavanja signala takta do sledećeg startovanja kontrolera. Ukoliko se vrši zaustavljanje kontrolera, zaustavlja se i odbrojavanje signala takta i nema generisanja impulsa.

6.4.2.1.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 93) i dat u obliku dijagrama toka upravljačkih signala (slika 94) i sekvence upravljačkih signala (tabela 31).

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 93). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova.



Slika 94 Dijagram toka upravljačkih signala

U sekvenci upravljačkih signala se koriste iskazi za signale. Iskazi za signale su oblika

signali i

if uslov then signali.

Iskaz

signali

sadrži spisak upravljačkih signala blokova operacione jedinice **oper** i određuje koji signali treba da budu generisani.

Iskaz

if uslov then signali

sadrži uslov i spisak upravljačkih signala blokova operacione jedinice **oper** i određuje koji signali i pod kojim uslovima treba da budu generisani.

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 31 Sekvenca upravljačkih signala

! U koraku $step_0$ se ostaje sve vreme. U ovom koraku se ne generiše aktivna vrednost ni jednog od upravljačkih signala sve dok su oba signala **rdTMR** i **wrTMR** operacione jedinice **oper** neaktivni. Signal **rdTMR** postaje aktivan kada processor **CPU** prebaci adresne linije **ABUS_{15..0}** magistrale iz stanja visoke impedanse na vrednost adrese nekog od programski dostupnih registara **CR₂**, **WCR_{15..8}** ili **WCR_{7..0}** operacione jedinice i upravljačku liniju **RDBUS** magistrale iz stanja visoke impedanse na aktivnu vrednost, čime započinje ciklus čitanja. Signal **wrTMR** postaje aktivan kada processor prebaci adresne linije **ABUS_{15..0}** i linije podataka **DBUS_{7..0}** magistrale iz stanja visoke impedanse na vrednost adrese registra i sadržaja za upis u neki od programski dostupnih registara **CR₂**, **WCR_{15..8}** ili **WCR_{7..0}** i upravljačku liniju **WRBUS** magistrale iz stanja visoke impedanse na aktivnu vrednost, čime započinje ciklus upisa. Pri aktivnoj vrednosti jednog od signala **rdTMR** i **wrTMR** generiše se signal **fcTMR** operacione jedinice. Signalom **fcTMR** se obezbeđuje da upravljačka linija **FCBUS** magistrale pređe iz stanja visoke impedanse na aktivnu vrednost. U oba slučaja se, u zavisnosti od toga koji je od registara **CR₂**, **WCR_{15..8}** ili **WCR_{7..0}** adresiran sadržajem na adresnim linijama **ABUS_{15..0}** magistrale, generiše aktivna vrednost jednog od signala **selCR**, **selWCRH** i **selWCRL**, respektivno, operacione jedinice. !

! Pri aktivnoj vrednosti signala **rdTMR**, a u zavisnosti od toga koji od signala **selCR**, **selWCRH** i **selWCRL** operacione jedinice ima aktivnu vrednost, generiše se aktivna vrednost jednog od signala **CRout**, **WCRHout** i **WCRout** operacione jedinice, respektivno. Signalima **CRout**, **WCRHout** i **WCRout** se obezbeđuje da linije podataka **DBUS_{7..0}** magistrale pređu iz stanja visoke impedanse na vrednost sadržaja jednog od registara **CR₂**, **WCR_{15..8}** ili **WCR_{7..0}**, respektivno. Kada signal **rdTMR** postane neaktivan i signali **fcTMR**, **CRout**, **WCRHout** i **WCRout** postanu neaktivni, pa upravljačka linija **FCBUS** magistrale i linije podataka **DBUS_{7..0}** magistrale se vraćaju u stanje visoke impedanse. !

! Pri aktivnoj vrednosti signala **wrTMR**, a u zavisnosti od toga koji od signala **selCR**, **selWCRH** i **selWCRL** ima aktivnu vrednost, generiše se aktivna vrednost jednog od signala **CRin**, **WCRHin** i **WCRLin** operacione jedinice, respektivno. Signalima **CRin**, **WCRHin** i **WCRLin** se obezbeđuje da se sadržaj sa linija podataka **DBUS_{7..0}** magistrale upiše u jedan od registara **CR₂**, **WCR_{15..8}** ili **WCR_{7..0}**, respektivno. Kada signal **wrTMR** postane neaktivan i signali **fcTMR**, **CRin**, **WCRHin** i **WCRLin** postanu neaktivni, pa se upravljačka linija **FCBUS** magistrale vraća u stanje visoke impedanse. !

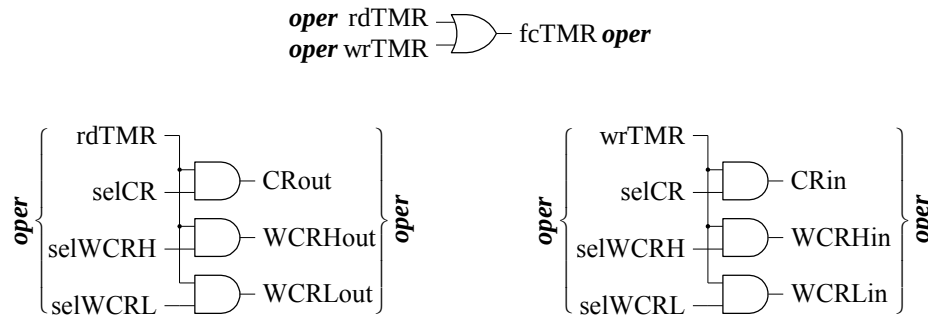
$step_0$ *if (rdTMR + wrTMR) then fcTMR,*
 if (rdTMR·selCR) then CRout,
 if (rdTMR·selWCRH) then WCRHout,
 if (rdTMR·selWCRL) then WCRLout,
 if (wrTMR·selCR) then CRin,
 if (wrTMR·selWCRH) then WCRHin,

$if (wrTMR \cdot selWCRL) then WCRLin$

6.4.2.1.3 Struktura upravljačke jedinice

Struktura upravljačke jedinice je prikazana na slici 95. Upravljačka jedinica se sastoji od logičkih elemenata koji na osnovu signala čitanja **rdTMR** i signala upisa **wrTMR** operacione jedinice i signala logičkih uslova generišu sledeće signale:

- signal **fcTMR** operacione jedinice završetka ili čitanja nekog od registara kontrolera CR_2 , $WCR_{15...8}$ ili $WCR_{7...0}$ ili upisa u neki od ovih registara kontrolera,
- signale **CRout**, **WCRHout** i **WCRLout** operacione jedinice čitanja registara kontrolera CR_2 , $WCR_{15...8}$ ili $WCR_{7...0}$,
- signale **CRin**, **WCRHin** i **WCRLin** operacione jedinice upisa u registre kontrolera CR_2 , $WCR_{15...8}$ ili $WCR_{7...0}$.



Slika 95 Struktura upravljačke jedinice *uprav_bus*

Signali **rdTMR** i **wrTMR** imaju aktivne vrednosti trajanja jedna perioda signala takta. To je posledica usvojenog načina generisanja signala **rdCPU** i **wrCPU** procesora **CPU**, na osnovu kojih se formiraju signali **RDBUS** i **WRBUS** magistrale **BUS** i **rdTMR** i **wrTMR** kontrolera, i signala **fcTMR**, na osnovu koga se generišu signali **FCBUS** magistrale **BUS** i **fcCPU** procesora **CPU**. Kod čitanja procesor na i -ti signal takta ulazi u stanje $step_i$ koje koristi da signal **rdCPU** postavi na aktivnu vrednost, što redom daje i aktivne vrednosti signala **RDBUS**, **rdTMR**, **fcTMR**, **FCBUS** i **fcCPU**. Kako je u procesoru aktivna vrednost signala **fcCPU** uslov za prelazak iz stanja $step_i$ u stanje $step_{i+1}$ i kako ta vrednost dolazi u koraku $step_i$, to procesor na $(i+1)$ -vi signal takta prelazi na korak $step_{i+1}$. Zbog toga se u koraku $step_i$ ostaje samo jedna perioda signala takta, što ima za posledicu da signali **rdCPU**, **RDBUS**, **rdTMR**, **fcTMR**, **FCBUS** i **fcCPU** imaju aktivnu vrednost trajanja jedna perioda signala takta. Iz istih razloga su kod upisa signali **wrCPU**, **WRBUS**, **wrTMR**, **fcTMR**, **FCBUS** i **fcCPU** trajanja jedna perioda signala takta. S obzirom na to da su signali **rdCPU** i **wrCPU** trajanja jedna perioda signala takta i da svi signali koje generiše upravljačke jedinica *uprav_bus* uključuju jedan od signala **rdTMR** i **wrTMR**, to i svi preostali signali imaju aktivne vrednosti trajanja jedna perioda signala takta.

Upravljačka jedinica *uprav_bus* sadrži kombinacione mreže koje pomoću signala **rdTMR** i **wrTMR**, koji dolaze iz operacione jedinice, signala logičkih uslova **selCR**, **selWCRH** i **selWCRL**, koji dolaze iz operacione jedinice, i saglasno algoritmu generisanja upravljačkih signala (tabela 31) generiše upravljačke signale operacione jedinice.

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

- **fcTMR** = **rdTMR** + **wrTMR**
- **CRout** = **rdTMR** · **selCR**
- **WCRHout** = **rdTMR** · **selWCRH**

- $WCRL_{out} = rdTMR \cdot selWCRL$
- $CR_{in} = wrTMR \cdot selCR$
- $WCRH_{in} = wrTMR \cdot selWCRH$
- $WCRL_{in} = wrTMR \cdot selWCRL$

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

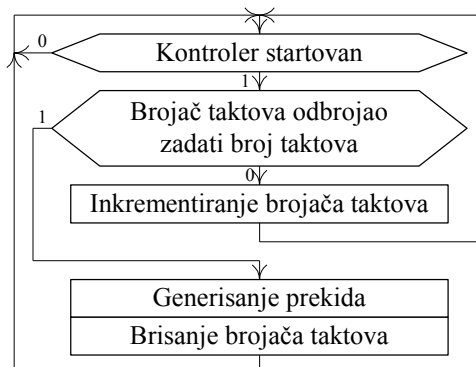
- **rdTMR** — operacione jedinice *oper*,
- **wrTMR** — operacione jedinice *oper*,
- **selCR** — operacione jedinice *oper*,
- **selWCRH** — operacione jedinice *oper*,
- **selIDR** — operacione jedinice *oper*,
- **selWCRL** — operacione jedinice *oper*.

6.4.2.2 Upravljačka jedinica generisanja impulsa

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

6.4.2.2.1 Dijagram toka operacija

Dijagram toka operacija je predstavljen operacionim i uslovnim blokovima (slika 96). U operacionim blokovima se nalaze opisi mikrooperacija koje treba realizovati. U uslovnim blokovima se nalaze opisi logičkih uslova koji definišu grananja algoritma.



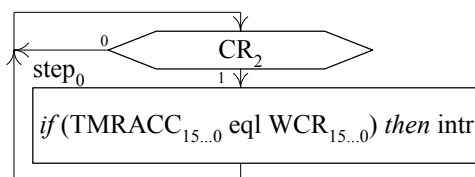
Slika 96 Dijagram toka operacija

U početnom koraku vrši se provera da li je kontroler startovan ili ne. Ako kontroler nije startovan nema izvršavanja mikrooperacija. Ako je kontroler startovan izvršavaju se odgovarajuće mikrooperacije. Najpre se vrši provera da li je brojač taktova odbrojao zadati broj taktova. Ako brojač nije odbrojao, vrši se provera da li je kontroler još uvek startovan ili je zaustavljen programskim putem. Ako je kontroler zaustavljen, nema izvršavanja mikrooperacija. Ako nije, ponavljaju se već opisani koraci. Ako je brojač odbrojao, generiše se impuls i briše brojač taktova, pa se proverava da li je kontroler još uvek startovan ili je zaustavljen programskim putem. Ako je kontroler zaustavljen, nema izvršavanja mikrooperacija. Ako nije, ponavljaju se već opisani koraci.

6.4.2.2.2 Algoritam generisanja upravljačkih signala

Algoritam generisanja upravljačkih signala je formiran na osnovu dijagrama toka operacija (slika 96) i dat u obliku dijagrama toka upravljačkih signala (slika 97) i sekvence upravljačkih signala po koracima (tabela 32).

Dijagram toka upravljačkih signala je predstavljen operacionim i uslovnim blokovima (slika 76). U operacionim blokovima se nalaze upravljački signali i uslovi pod kojima se oni generišu. U uslovnim blokovima se nalaze signali logičkih uslova.



Slika 97 Dijagram toka upravljačkih signala

U sekvenci upravljačkih signala se koriste iskazi za signale. Iskazi za signale su oblika **if uslov then signali**.

Iskaz

if uslov then signali

sadrži uslov i spisak upravljačkih signala blokova operacione jedinice **oper** i određuje koji signali i pod kojim uslovima treba da budu generisani.

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. Notacija koja se koristi je identična kao i notacija za algoritam generisanja upravljačkih signala za upravljačku jedinicu procesora **CPU** (poglavlje \$\$\$).

Tabela 32 Sekvenca upravljačkih signala

! U koraku se $step_0$ se ostaje i ne generiše se aktivna vrednost ni jednog od upravljačkih signala sve dok je signal **CR₂** bloka *registri* neaktivan. Signal **CR₂** postaje aktivan tek kada se programskim putem startuje kontroler periferije. Tom prilikom se u upravljački registar **CR₂** operacione jedinice upisuje sadržaj koji na poziciji razreda **CR₂** ima aktivnu vrednost.. !

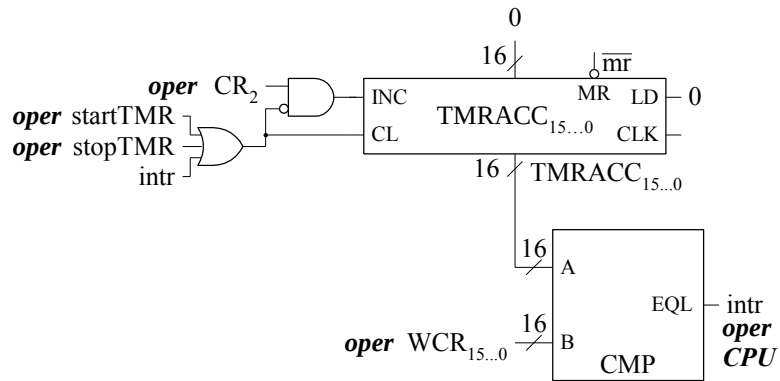
$step_0$ **if** (TMRACC_{15...0} eql WCR_{15...0}) **then intr**

6.4.2.3 Struktura upravljačke jedinice

Struktura upravljačke jedinice je prikazana na slici 63. Upravljačka jedinica se sastoji od brojača TMRACC_{15...0} i komparatora CMP i logičkih elemenata koji generišu signal **intr** operacione jedinice **oper**.

Brojač TMRACC_{15...0} se koristi kao brojač signala takta. Njegov sadržaj se postavlja na početni sadržaj sve nule aktivnom vrednošću jednog od signala stopTMR, startTMR i intr. Aktivna vrednost signala stopTMR se generiše pri upisu neaktivne vrednosti u registar **CR₂**, dok se aktivna vrednost signala startTMR generiše pri upisu aktivne vrednosti u registar **CR₂**. Pri startovanom kontroleru u razredu **CR₂** je aktivna vrednost koja omogućuje inkrementiranje brojača TMRACC_{15...0} sve dok se njegov sadržaj ne izjednači sa sadržajem registra WCRACC_{15...0}. Komparator CMP upoređuje sadržaje registra WCRACC_{15...0} i brojača TMRACC_{15...0} i na izlazu EQL daje neaktivnu vrednost signala **intr** dok se ovi sadržaji razlikuju i aktivnu vrednost signala **intr** kada ovi sadržaji postanu jednaki. Aktivna vrednost signala **intr** se šalje procesoru kao signal prekida. Pored toga, aktivna vrednost signala **intr** se koristi u upravljačkoj jedinici da se sadržaj brojača TMRACC_{7...0} vrati na stanje sve nule.

Kontroler generiše aktivnu vrednost signala rekida **intr** trajanja jedna perioda signala takta kada od trenutka startovanja kontrolera odbroji onoliko perioda signala takta koliki je sadržaj registra WCR_{15...0}.



Slika 98 Struktura upravljačke jedinice *uprav_intr*

Upravljački signal **intr** koji se koristi u procesoru **CPU** i operacionoj jedinici **oper** se generiše na sledeći način:

$$\mathbf{intr} = \mathbf{MEMACC}_{7...0} \text{ eql } \mathbf{TIME}_{7...0}$$

7 LITERATURA

1. Lazić, B., *Logičko projektovanje računara*, Nauka—Elektrotehnički fakultet, Beograd, Srbija, Jugoslavija, **1994**
2. Živković, D., Popović, D., *Impulsna i digitalna elektronika*, Nauka—Elektrotehnički fakultet, Beograd, Srbija, Jugoslavija, **1992**.
3. Aleksić, T., *Računari—organizacija i arhitektura*, Naučna knjiga, Beograd, Srbija, Jugoslavija, **1985**.
4. Stallings, W., *Computer Organization and Architecture*, Prentice-Hall, Englewood Cliffs, New Jersey, USA, **1996**.
5. Patterson, D., Hennessy, J., Goldberg, D., *Computer Architecture—A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Francisco, California, USA, **1996**.
6. Flynn, M., *Computer Architecture: Pipelined and Parallel Processor Design*, Jones and Bartlett, USA, **1995**
7. J. Djordjevic, A. Milenkovic, N. Grbanovic, “An Integrated Educational Environment for Teaching Computer Architecture and Organisation,” IEEE MICRO, May 2000.pp. 66-74.
8. J. Djordjevic, M. Bojovic, A. Milenković, *An Integrated Educational Environment for Computer Architecture and Organisation*, Proceedings of the Symposium on Education and Employment, France, September, 1998.
9. J. Djordjevic, A. Milenkovic, S. Prodanovic “A Hierarchical Memory System Environment,” IEEE TC Computer Architecture Newsletter, March 1999.
10. J. Đorđević, B. Nikolić, *Vizuelni simulator edukacionog računara*, Zbornik radova IT 2001, Žabljak, Jugoslavija, Mart 2001.
11. J. Djordjevic, R. N. Ibbett, M. R. Barbacci, *Evaluation of computer architectures using ISPS*, Proc. of IEE, Vol. 127, Pt. E. No. 4, pp. 126-131, July 1980.
12. J. Djordjevic, M. R. Barbacci, B. Hosler, *A PMS Level Notation for the Description and Simulation of Digital Systems*, The Computer Journal, Vol. 28, No. 4, pp. 357-365, 1985.
13. S. Miladinović, J. Đorđević, A. Milenković, *Programski sistem za grafički opis i simulaciju digitalnih sistema*, Zbornik radova ETRAN 1997, Zlatibor, Jugoslavija, Jun 1997.
14. N. Grbanovic, J. Djordjevic, B. Nikolić, *The Software Package Of An Educational Computer System*, prijavljen za objavljivanje u IJEEE, England, October, 2002.
15. J. Đorđević, B. Nikolić, *Neki Aspekti Realizacije Vizuelnog Simulatora Edukacionog Računara*, , Zbornik radova IT 2001, Žabljak, Jugoslavija, Mart 2001.
16. J. Đorđević, T. Borozan, B. Nikolić, *Softversko okruženje za simulaciju računara*, Zbornik radova ETRAN 2001, Bukovička Banja, Jugoslavija, Jun 2001.
17. J. Djordjevic, A. Milenkovic, I. Todorovic, D. Marinov, “CALCAS: A Computer Architecture Learning and Knowledge Assessment System,” IEEE TC Computer Architecture Newsletter, March 1999.

18. A. Milenkovic, Nikolić, B., J. Djordjevic, "CASTLE, *Computer Architecture Self-Testing and Learning System*," WCAE 2002, Workshop on Computer Architecture Education, Anchorage, Alaska, May 26, 2002.
19. Đorđević, J., *Priručnik iz arhitekture računara*, Elektrotehnički fakultet, Beograd, **1997**.
20. Đorđević, J., *Priručnik iz arhitekture i organizacije računara*, Elektrotehnički fakultet, Beograd, **1997**.
21. Đorđević, J., Grbanović, N., Nikolić, B., *Arhitektura računara, Edukacioni računarski sistem, Priručnik za simulaciju sa zadacima*, Elektrotehnički fakultet, Beograd, **2002**.

