Zadatak 6

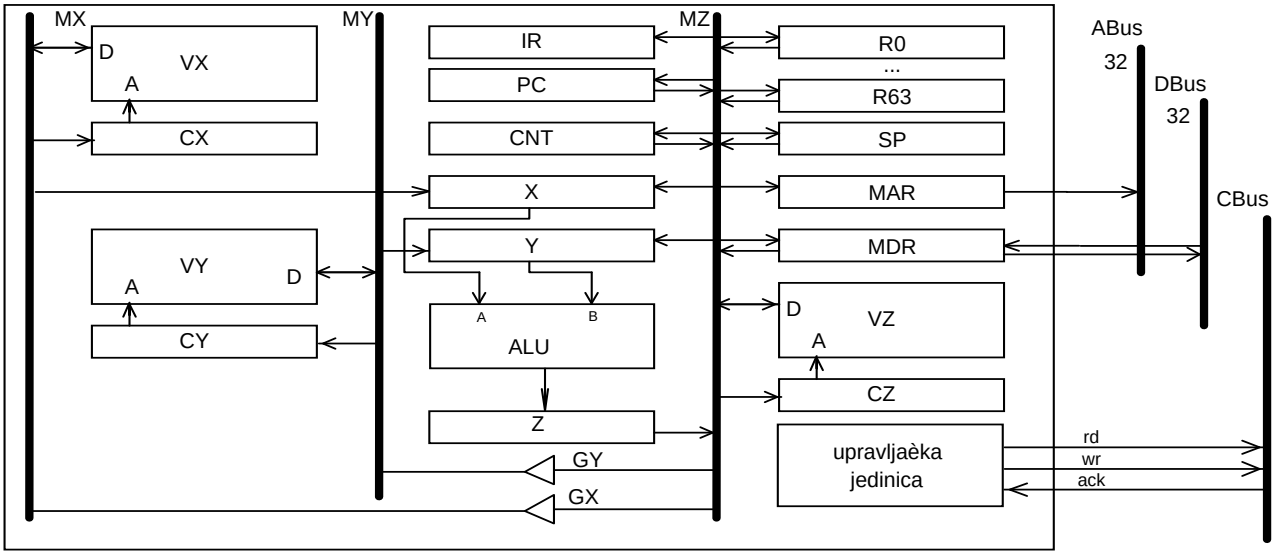
Procesor je specijalizovan za operacije nad vektorima (vektorski procesor). Za te potrebe procesor poseduje dva skupa registara. Prvi skup su registri opšte namene za skalarne veličine, ima ih 64, i označeni su sa R0 do R63. Drugi skup su vektorski registri. Postoje tri vektorska registra, VX, VY i VZ, pri čemu je svaki od ovih registara zapravo registarska memorija sa po 64 registra. Svi registri su 32-bitni. Adrese su široke 32 bita, širina magistrale podataka je 32 bita, a adresiranje je na nivou 32-bitne reči. Procesor operiše samo sa 32-bitnim celobrojnim veličinama (u daljem tekstu *reč* označava 32-bitnu veličinu). Magistrala je sinhrona. Memorija je organizovana sa preklapanjem pristupa memorijskim modulima. Ciklus na magistrali kojim se inicira operacija upisa ili čitanja traje jedan takt, kao i ciklus vraćanja očitanog podatka. Vreme odziva memorije, od završetka ciklusa iniciranja čitanja do početka ciklusa vraćanja očitanog podatka iznosi 3 periode takta. Ciklus vraćanja očitanog podatka iz memorije ima najviši prioritet.

Pored troadresnih instrukcija koje operišu nad skalarnim veličinama u registrima R0 do R63, procesor poseduje i instrukcije za operacije nad vektorskim celobrojnim veličinama u vektorskim registrima. Instrukcija VADD R*i* sabira vektore u VX i VY i rezultat smešta u VZ, pri čemu je dužina vektora definisana sadržajem R*i*. Ukoliko je vrednost u R*i* veća od 64, rezultat je nedefinisan (dejstvo instrukcije zavisi od interne realizacije). Dejstvo ove instrukcije je definisano sa: VZ[*k*]=VX[*k*]+VY[*k*], *k*=0,...,[R*i*]-1. Analogno dejstvo imaju ostale vektorske instrukcije VSUB, VAND, VOR i VXOR. Instrukcija LOADX [R*i*],R*j* učitava vrednosti iz memorije u registar VX. Elementi se učitavaju počev od adrese na koju ukazuje sadržaj R*i*, a R*j* sadrži dužinu vektora. Analogno deluje instrukcija LOADY. Instrukcija STOREZ [R*i*],R*j* vrši upis vektora iz VZ u memoriju.

Mogu se koristiti sve ostale potrebne instrukcije koje operišu sa skalarnim veličinama u registrima R0..63 i potrebni načini adresiranja za ove instrukcije.

Organizacija procesora

Organizacija procesora data je na slici. Svi registri i interne magistrale su 32-bitni. ALU ima, pored ostalih, i kontrolne ulaze *incA* i *decA*. Pretpostaviti da interni registri CX, CY, CZ i CNT imaju sve potrebne upravljačke signale za brisanje, inkrementiranje, dekrementiranje, kao i kombinacione mreže koje generišu signale logičkih uslova, npr. jednakosti sa nulom. Vektorski registri VX, VY i VZ realizovani su kao registarske memorije sa direktnim pristupom. Uz vektorske registre postoje i interni skalarni registri CX, CY i CZ; CX i CY služe kao adresni registar za čitanje ili upis u VX i VY, a CZ služi kao adresni registar za upis ili čitanje iz VZ. Vektorske instrukcije treba realizovati tako da se faza izvršavanja završi što pre. To se obezbeđuje protočnom obradom elemenata vektora (*pipelining*), tako što se protočno vrši čitanje iz VX i VY, operacija u ALU i upis u VZ.



Zadatak:

a) Definisati kombinacionu mrežu koja na izlazima daje upravljačke signale za tip operacije u ALU: *ALUadd*, *ALUsub*, *ALUand*, *ALUor* i *ALUxor*, a na ulazima prima upravljački signal *ALUop* koji dolazi iz upravljačke jedinice, kao i signal dekodovane operacije, tako da instrukcije VADD, VSUB, VAND, VOR i VXOR mogu da imaju isti mikroprogram.

b) Napisati mikroprogram za ovaj procesor, sa fazom izvršavanja samo za opisane vektorske instrukcije VADD, VSUB, VAND, VOR i VXOR, a predvideti postojanje ostalih. Kôd treba da bude prilagođen mikroprogramskoj upravljačkoj jedinici, pri čemu se u jednoj mikronaredbi nalaze i polje sa upravljačkim signalima i polja koja definišu uslovni skok u mikroprogramu. Ne treba pisati mikroprogram za obradu prekida. Pretpostaviti da je dohvatanje eventualne druge reči instrukcije u fazi izvršavanja instrukcija koje poseduju te reči. Faza izvršavanja treba da bude realizovana jedinstvenim mikroprogramom za sve ove instrukcije i to protočnom obradom, prema uslovima postavke zadatka.

c) Na asembleru datog procesora napisati program koji izvršava sledeću vektorsku naredbu: A:=B+C-D, gde su A, B, C i D vektori dužine 30h, a počinju redom na adresama 100h, 200h, 300h i 400h. Ne smeju se koristiti privremene memorijske lokacije.

d) Kako bi se, korišćenjem protočne obrade u koju je uključeno i učitavanje vektora iz memorije i upis vektora u memoriju, realizovala operacija VNOT[R*i*],R*j* koja invertuje svaki element vektora koji se nalazi u memoriji počev od adrese na koju ukazuje *Ri* I , a ne u vektorskom registru? Opisati precizno ideju rešenja.

a)

ALUadd=ALUop(ADD+VADD)+add
ALUsub=ALUop(SUB+VSUB),
ALUand=ALUop(AND+VAND),
ALUor=ALUop(OR+VOR),
ALUxor=ALUop(XOR+VXOR),

b)

```
; Dohvatanje instrukcije

BEGIN:      PCout, MARin, XZin
            read, incA, ldZ
            wmfc
            MDRout, IRin
            Zout, PCin, opcase

; Vektorske instrukcije

VECINS:     clCX,clCY,clCZ,REGout,CNTin; priprema brojača i adresnih registara

            branch(CNT==0, VEND),VXout, XMXin,incCX, VYout, YMYin, incCY

            ALUop,ldZ,decCNT, VXout,XMXin,incCX, VYout,YMYin,incCY ;

VLOOP:
      Zout,wrVZ,incCZ,ALUop,ldZ,decCNT,VXout,XMXin,incCX,VYout,YMYin,incCY,
                                              branch(CNT!=0,VLOOP)

VEND:       branch(IR,INTH)
            bruncnd(BEGIN)
```

c)

```
      LOAD  R0,#30h    ; dužina vektora
      LOAD  R1,#100h   ; A
      LOAD  R2,#200h   ; B
```

```
LOAD  R3,#300h   ; C
LOAD  R4,#400h   ; D
LOADX [R2],R0    ; VX=B
LOADY [R3],R0    ; VY=C
VADD  R0         ; VZ=VX+VY
STOREZ [R1],R0   ; A=B+C
LOADX [R1],R0    ; VX=A
LOADY [R4],R0    ; VY=D
VSUB  R0         ; VZ=VX-VY
STOREZ [R1],R0   ; A=B+C-D
```

d) Potrebno je realizovati dve odvojene hardverske jedinice za čitanje jedne reči iz memorije i upis jedne reči u memoriju. Ove jedinice treba samo da prime zahtev (adresu i podatak) i izvrše zahtev u narednom taktu (vrate podatak pri čitanju). Keš memorija treba da obezbedi odziv pri čitanju u jednom taktu. Ukoliko se dogodi promašaj, upravljačka jedinica se blokira do završetka operacije čitanja. Protočna obrada vršila bi se tako da se u istom taktu obavlja sledeće:
- jedinica za upis upisuje jednu reč rezultata na odgovarajuću lokaciju u memoriji;
- jedan podatak se premešta iz registra Z u registar podatka jedinice za upis u memoriju i toj jedinici se izdaje zahtev za upis;
- rezultat operacije se iz ALU upisuje u Z registar;
- operand se upisuje u X registar iz registra podatka jedinice za čitanje iz memorije, i
- zadaje se operacija čitanja iz memorije jedne reči operanda.

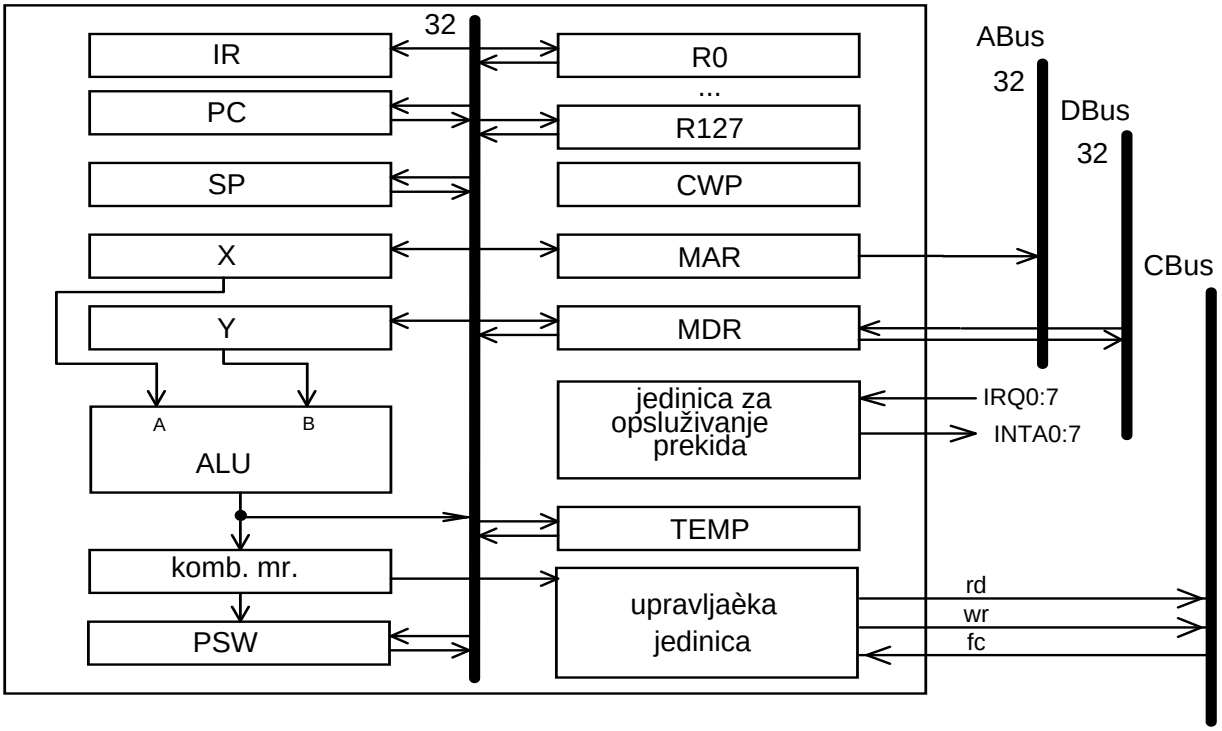
Zadatak 7

Procesor je troadresni, s tim da sve troadresne instrukcije operišu samo sa operandima u registrima opšte namene. Jedine instrukcije koje operišu sa memorijom su instrukcije LOAD, STORE i JSR (*Jump to Subroutine*). Kod instrukcija LOAD i STORE odredišni, odnosno izvorišni operand u registru, a drugi operand je zadat neposrednim (samo za LOAD), ili nekim od memorijskih načina adresiranja: registarskim indirektnim sa pomerajem ili memorijski direktnim. Svi registri opšte namene su 32-bitni.
Adrese su široke 32 bita, širina magistrale podataka je 32 bita, a adresiranje je na nivou 32-bitne reči. Procesor operiše samo sa 32-bitnim celobrojnim veličinama (u daljem tekstu *reč* označava 32-bitnu veličinu). Magistrala je asinhrona, vreme odziva memorije je neodređeno.

U cilju ubrzanja prenosa parametara kod poziva potprograma i pristupa do lokalnih promenljivih, procesor poseduje tzv. registarski fajl sa prozorima (*window register file*), što znači sledeće: procesor poseduje 128 fizičkih registara opšte namene, ali su u svakom trenutku programski dostupna samo 24 od njih, koji se u instrukciji označavaju sa R0 do R23 (u formatu mašinske instrukcije kôd registra koji se koristi je 5-bitni i označava registar 0..23). Od ova 24 registara, registri R0..R7 označeni su kao *Low* registri, R8..R15 kao *Mid* registri, a R16..R23 kao *High* registri. U *Low* registrima se nalaze stvarni parametri sa kojima je pozvan tekući potprogram. *Mid* registri su namenjeni da u njih potrpogram smešta svoje lokalne promenljive. U *High* registre potprogram smešta stvarne parametre potrpograma koji on poziva. Kada se pozove potprogram instrukcijom JSR, menja se aktivni skup dostupnih fizičkih registara (tzv. prozor), tako da novopozvani potprogram vidi *High* registre pozivajućeg potprograma kao svoje *Low* registre. Prilikom povratka iz potprograma prozor se vraća na skup registara pozivajućeg potprograma. Prema tome, prozor se pri pozivu potprograma pomera za 16 fizičkih registara unapred, a pri povratku za 16 unazad. Niz od 128 fizičkih registara se posmatra kao kružna lista po kojoj se pomera registarski prozor od 24 dostupna registra. Na ovaj način se zapravo alokacioni stek (za prenos parametara i smeštanje lokalnih promenljivih) umesto u memoriji nalazi u registrima, čime se ubrzava pozivanje i izvršavanje potprograma.

Instrukcija JSR može koristiti jedan od sledećih načina adresiranja: memorijsko direktno (adresa potprograma je u drugoj reči instrukcije), registarsko indirektno (adresa potprograma je zapisana u registru opšte namene) i PC-relativno (pomeraj je u drugoj reči instrukcije). Prilikom skoka u potprogram na steku (koji se nalazi u memoriji) se čuva samo PC. Stek raste prema nižim lokacijama, a SP ukazuje na prvu slobodnu lokaciju na steku.

Organizacija procesora



Organizacija procesora data je na slici. Prikazani su fizički registri opšte namene označeni sa R0..R127. CWP (*Current Window Pointer*) je 3-bitni registar koji predstavlja redni broj tekućeg prozora (0..7). Ovaj registar poseduje kombinacionu mrežu koja obavlja inkrementiranje i dekrementiranje, koje se zadaje upravljačkim signalima *incCWP* i *decCWP*. ALU ima, pored ostalih, i kontrolne ulaze *incA* i *decA*. Mogu se koristiti sve potrebne instrukcije u programiranju, u skladu sa navedenim pravilima o formatu i načinima adresiranja.

Zadatak:

a) Na asembleru datog procesora napisati prevod date funkcije f. Pretpostaviti da svaka funkcija svoju povratnu vrednost smešta u svoj logički registar R0, da funkcija g vraća rezultat tipa int, i da je int veličine 32 bita (veličina registra).

```
int f (int i, int j) {
    int k;
    k=i+j;
    return g(k);
}
```

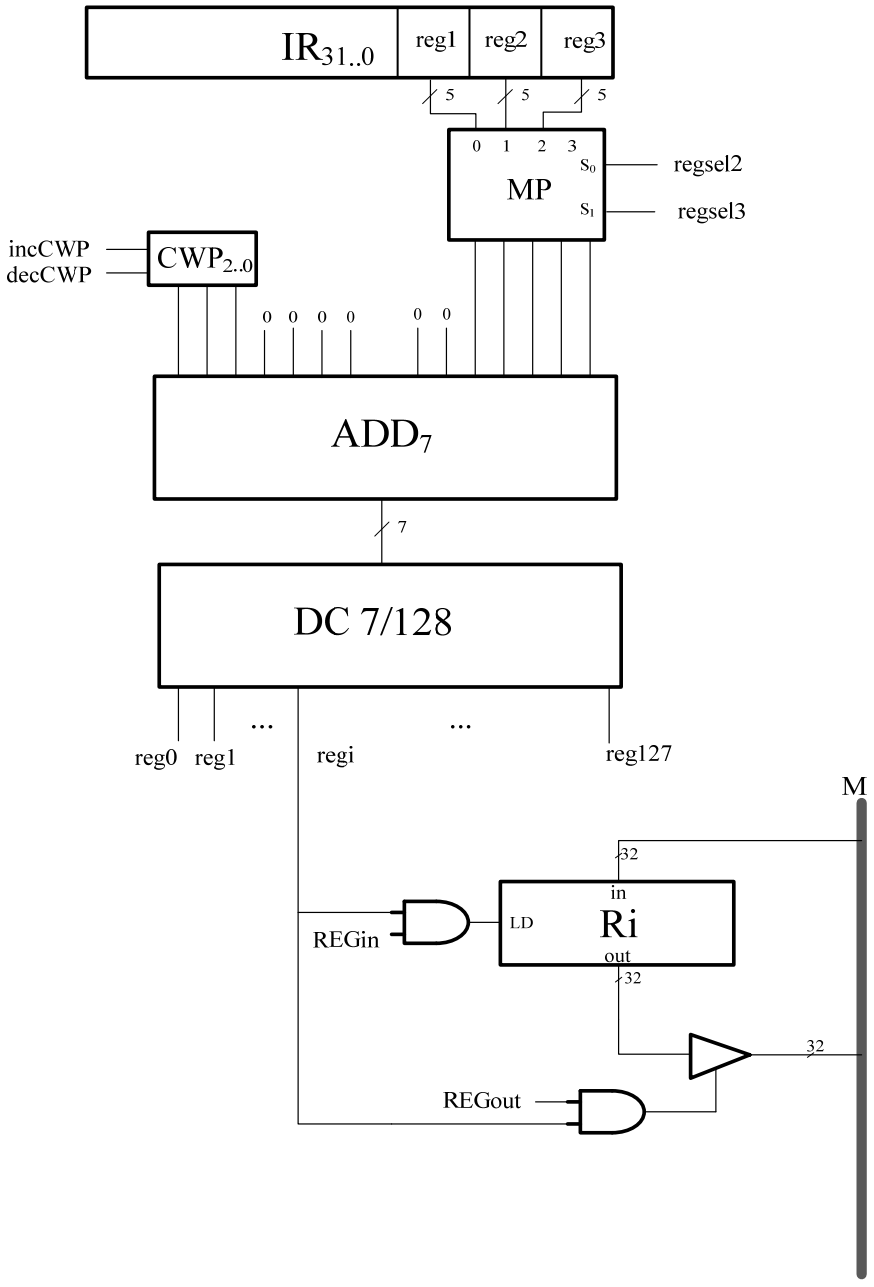
b) Definisati mrežu koja povezuje registar CWP u sistem. Mreža treba da omogući da se signalima *regsel1*, *regsel2* i *regsel3* selektuje odgovarajući registar. Upis i čitanje iz registara treba realizovati signalima *REGout* i *REGin*.

c)Napisati mikroprogram za ovaj procesor, sa fazom izvršavanja **samo** za instrukcije JSR i RTS, a predvideti postojanje ostalih. Zanemariti problem opisan u tački d). Kôd treba da bude prilagođen mikroprogramskoj upravljačkoj jedinici, pri čemu se u jednoj mikronaredbi nalaze i polje sa upravljačkim signalima i polja koja definišu uslovni skok u mikroprogramu. **Ne treba** pisati mikroprogram za obradu

prekida. Pretpostaviti da je dohvatanje eventualne druge reči instrukcije u fazi izvršavanja instrukcija koje poseduju te reči.
d)Zbog ograničenog broja fizičkih registara, postoji problem ograničene veličine alokacionog steka realizovanog pomoću registarskog fajla sa prozorima. Naime, ukoliko se izvrši deveti ugnežđeni poziv potrpograma, prozor novopozvanog potprograma bi "pregazio" prozor koji koristi prvi potprogram, pa bi se povratkom u ovaj prvi potprogram dogodila greška zbog izgubljenih vrednosti u registrima. Predložiti rešenje ovog problema. Uzeti u obzir i način povratka u ranije potprograme posle dubokog ugnežđivanja poziva.

a)
```
ADD  R8,R1,R2    ; i->R1,j->R2,k->R8;
MOV  R17,R8
JSR  g           ; Call g(k)
MOV  R0,R16
RTS
```

b) Tražena mreža prikazana je na slici. Redni broj prvog registra tekućeg prozora dobija se množenjem CWP sa 16 (tako dobijena vrednost ukazuje zapravo na logički registar R0).



c)

```
; Dohvatanje instrukcije
BEGIN:      PCout,MARin,Xin
            read,incA,ALUout,PCin
            wmfc
            MDRout,IRin

; Dekodiranje instrukcije
            Opcase

; JSR instrukcija

JSR:        admodJSR
RI:         regsel1,REGout,TEMPin,bruncnd(JSROP) ; Registarsko indirektno
MD:         PCout,MARin,Xin ; Memorijsko direktno
            read,incA,ALUout,PCin
            wmfc
            MDRout,TEMPin, bruncnd (JSROP)
PCREL:      PCout,MARin,Xin ; PC-relative
            read,incA,ALUout,PCin,Xin
            wmfc
            MDRout,Yin
            ALUadd,ALUout,TEMPin
JSROP:      SPout,MARin,Xin ;
            PCout,MDRin
            write,decA,ALUout,SPin
            wmfc
            TEMPout,PCin,incCWP,branch(IRQ,INTH)
            bruncnd (BEGIN)

RTS:        SPout, Xin
            incA, MARin, SPin
            read
            wmfs
            MDRout, PCin, decCWP, branch(IRQ,INTH)
            bruncnd (BEGIN)
```

d) Kada se stek prozora napuni, potrebno je osloboditi jedan prozor tako što se njegov sadržaj prebaci u memoriju. Zato je potreban još jedan registar, SWP (*Saved Window Pointer*) koji označava "dno" steka prozora, odnosno prvi prozor koji nije sačuvan u memoriji. Kada se izvršava instrukcija JSR i inkrementira CWP, ako CWP dostigne vrednost SWP (zbog logike kružnog bafera prozora), potrebno je sadržaj prozora na koji ukazuje CWP prebaciti u memoriju (to radi procesor kao deo izvršavanja instrukcije JSR), a SWP inkrementirati. Kod povratka iz potprograma (instrukcija RTS), ako je CWP jednako SWP, što znači da je CWP na dnu steka i da je prethodni prozor neažuran, potrebno je njegov sadržaj dovući iz memorije a SWP dekrementirati.