Zadatak 8

Procesor je troadresni i ima 32 registra opšte namene, R0 do R31, svi su 32-bitni. Adrese su široke 32 bita, širina magistrale podataka je 32 bita, a adresiranje je na nivou 32-bitnih reči. Procesor operiše samo sa 32-bitnim celobrojnim veličinama (u daljem tekstu *reč* označava 32-bitnu veličinu). Ne postoje registri SP ni PSW, a ni poseban registar PC, nego registar R31 služi kao programski brojač. Između procesora i magistrale vezana je keš memorija. Registar R0 je hardverski ožičen na nulu, a registar R1 na 1 (drugim rečima, čitanje iz ovih registara uvek daje 0, odnosno 1, a upis se ignoriše).

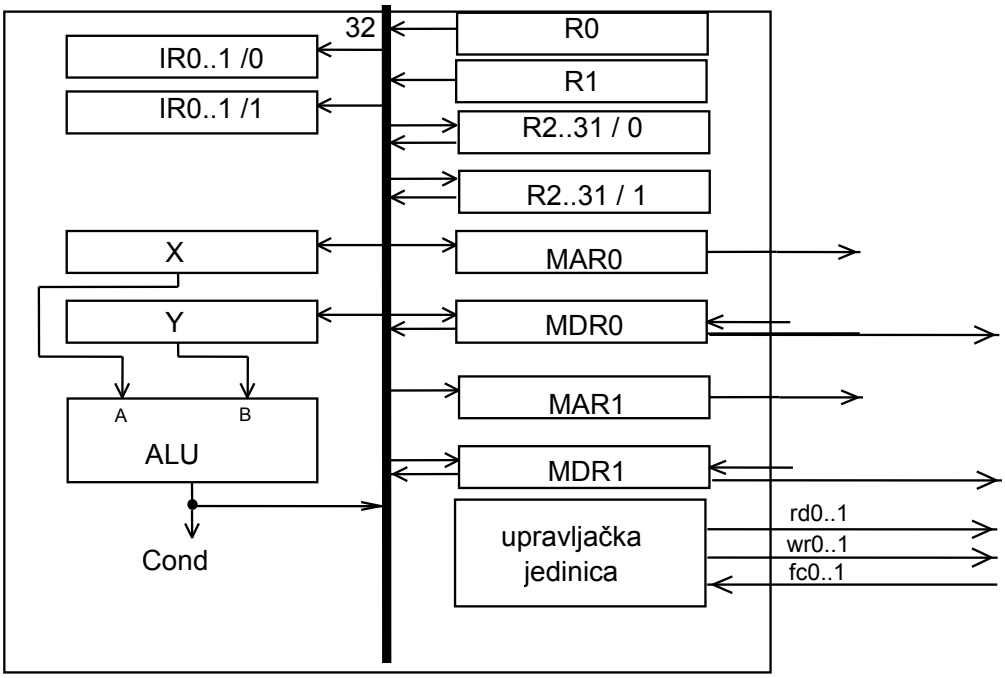
Keš kontroler generiše signal *fc* (*Function Complete*) procesoru kao znak da je operacija završena. Ovaj signal procesor koristi za zaustavljanje rada svoje upravljačke jedinice u slučaju promašaja u kešu, kada operacija dovlačenja bloka podataka iz memorije traje neodređeno dugo. Procesor nema mogućnost obrade prekida. Ulazno/izlazni adresni proctor je memorijski presklikan.

Postoje samo tri grupe instrukcija. Prva su aritmetičko/logičke instrukcije. One su troadresne i operišu isključivo nad registrima opšte namene. Ove instrukcije su veličine jedne reči. Druga grupa su instrukcije prenosa podataka LOAD i STORE. Kod ovih instrukcija jedan operand je uvek u registru opšte namene (odredište za LOAD i izvorište za STORE), a drugi operand je u memoriji, na lokaciji zadatoj registarskim indirektnim adresiranjem sa 32-bitnim pomerajem. Ove instrukcije su veličine dve reči (u drugoj reči je pomeraj). Treća grupa instrukcija su uslovni skokovi. Uslov skoka je definisan relacijom između dva registra opšte namene. Adresa skoka je data relativno u odnosu na treći registar opšte namene, a pomeraj je 32-bitna konstanta. Ove instrukcije su dužine dve reči (u drugoj reči je pomeraj).

U cilju eliminisanja negativnog efekta čekanja na završetak operacije sa kešom u slučaju promašaja, procesor podržava izvršavanje više procesa kreiranih unutar istog programa (tzv. “laki procesi”, ili “niti”, engl. *threads*). Procesor u svakom trenutku poseduje zapamćen kontekst (programski dostupne registre) za dva procesa. To je izvedeno tako što postoje dva skupa registara opšte namene (koji su i jedini programski dostupni registri), svaki za po jedan proces, tako da jedan proces može da pristupa samo svojim registrima. Ovakav procesor naziva se *multithreaded* procesor. Za vezu sa keš memorijom postoje dve grupe registara MAR i MDR, kao i dva signala *fc* kojima keš kontroler obaveštava procesor o završenoj operaciji. Mikroprogramska upravljačka jedinica procesora poseduje samo jednu kopiju mikroprograma u mikromemoriji i radi na sledeći način. U datom trenutku izvršava se mikroprogram za instrukciju jednog (tekućeg) procesa. Ukoliko je potrebno da instrukcija pristupa kešu, a keš kontroler ne vrati *fc* u istom taktu, mikroupravljačka jedinica “napušta” taj proces i nastavlja izvršavanje instrukcije drugog procesa, koji tada postaje tekući. Samo ukoliko se izvršavanje za oba procesa nalazi u stanju čekanja na keš, upravljačka jedinica prelazi u stanje čekanja, i potom nastavlja izvršavanje procesa koji je prvi završio keš operaciju. Keš kontroler je, dakle, u stanju da istovremeno obrađuje dva zahteva za čitanje ili upis. Vrednost posebnog flip-flopa CT (*Current Thread*) ukazuje na tekući proces. Pomoću ovog flip-flopa vrši se i selekcija registra iz jednog od dva skupa registara opšte namene sa kojima mikroinstrukcija operiše.

Organizacija procesora

Na slici je data blok-šema operacione jedinice procesora. ALU ima sve potrebne upravljačke signale.



Zadatak:

a) Napisati kompletan mikroprogram za ovaj procesor.

b) Na asembleru datog procesora napisati prevod sledećeg dela programa na jeziku C:

```
extern int n, a[];
for (int i=0; i<n; i++) a[i]+=i;
```

Tip int je veličine jedne mašinske reči. Promenljive **n** i **a** su globalne, statičke, alocirane u nekom drugom delu programa na adresama koje su simbolički označene imenima tih promenljivih. Promenljivu **i** smestiti u registar R2.

c) Nad sledećim delom programa:

```
XOR  R3, R3, R3 ; R0:=R0 xor R0
LOAD R2,x[R3]
```

kreirana su dva procesa čiji se konteksti nalaze u procesoru. Inicijalno je keš prazan, a kod ovog dela programa se nalazi u različitom bloku memorije od podatka na adresi x. Dohvatanje bloka iz memorije traje mnogo ciklusa takta. Označiti mikroinstrukcije iz mikroprograma u tački c) adresama počev od 0, a zatim napisati sekvencu adresa prvih 30 mikroinstrukcija koje se redom izvršavaju u procesoru za ovaj slučaj.

d) Nacrtati šemu mikroprogramske upravljačke jedinice, ali to samo onog dela koji se odnosi na mikroprogramski brojač.

```
a)    ; Dohvatanje instrukcije
00    BEGIN:     regsel0,REGout,MARin,Xin ; regsel0 selektuje R31
01               read,ALUinc,ALUout,regsel0,REGin
02               wmfc
03               MDRout,IR0in
04               regsel0,REGout,MARin,Xin, branch(not Displ,ALU); samo ALU
                                    instrukcije nemaju pomeraj
05               read,ALUinc,ALUout,regsel0,REGin
```

```
06              wmfc
07              MDRout,IR1in, opcase
     ; ALU instrukcije
08   ALU:       regsel1,REGout,Xin
09              regsel2,REGout,Yin
0A              ALUop,ALUout,regsel3,REGin,bruncnd(BEGIN)
     ; LOAD instrukcija
0B   LOAD:      regsel2,REGout,Xin
0C              IR1out,Yin
0D              ALUadd,ALUout,MARin
0E              read
0F              wmfc
10              MDRout,regsel3,REGin,bruncnd(BEGIN)
     ; STORE instrukcija
11   STORE:     regsel2,REGout,Xin
12              IR1out,Yin
13              ALUadd,ALUout,MARin
14              regsel3,REGout,MDRin
15              write
16              wmfc
18              bruncnd(BEGIN)
     ; BRANCH instrukcija
18   BRANCH:    regsel1,REGout,Xin
19              regsel2,REGout,Yin
1A              ALUop,branch(not Cond,BEGIN), regsel3,REGout,Xin
1B              IR1out,Yin
1C              ALUadd,ALUout,regsel0,REGin,bruncnd(BEGIN)
```

S obzirom da je ulazno/izlazni adresni prostor memorijski mapiran, nije potrebna realizacija instrukcija IN i OUT, tako da su sve neophodne operacije realizovane ovim kratkim mikroprogramom.
Operaciju MOV *Ri, Rj* (premeštanje između registara) moguće je simulirati operacijom ADD *Ri, Rj, R0*, operaciju inkrementiranja je moguće realizovati sa ADD *Ri, Ri, R1* itd.
Primetiti da bi dati mikroprogram izgledao identično i u slučaju da procesor nije *multithreaded*. Svaki signal koji generiše upravljačka jedinica (npr. `MDRout`) aktiviraće tačno jedan od dva moguća signala operacione jedinice (`MDR0out` ili `MDR1out`, zavisno od vrednosti flip-flopa CT). Upravljačka jedinica toga nije svesna, već se konačan izbor signala koji će se aktivirati radi hardverski. Jedina stvar o kojoj treba voditi računa u mikroprogramu jeste sadržaj registara X i Y, kao jedinih sekvencijalnih mreža operacione jedinice koje nisu duplirane. Međutim, treba primetiti da je u svakom trenutku kada je moguće blokiranje procesa, tj. kada se zada signal `wmfc`, (to su mikroinstrukcije na adresama `02h, 06h, 10h, 17h`), sadržaji registara X i Y nisu više bitni tekućem procesu, tj. njihov sadržaj je već iskorišćen za generisanje adrese (upisan u MAR registar koji je privatan za svaki proces) ili je rezultat ALU operacije na osnovu sadržaja tih registara upisan u registar opšte namene ili je spreman za upis u memoriju (upisan u registar MDR0 ili MDR1).

b)
```
     ; i->R2, n->R3, a[i]->R4
     ; Alokacija registara:
          XOR   R2,R2,R2 ; R2:=0. Druga varijanta je ADD R2, R0, R0
          LOAD  R3,n[R0]
     ; for (...; i<n; ...)
Loop:     BNLT  R2,R3,Exit[R0]
     ; a[i]+=i
Loop1:    LOAD  R4,a[R2]
          ADD   R4,R4,R2
          STORE R4,a[R2]
     ; i++
          ADD   R2,R2,R1  ; INC R2
          BEQ   R0,R0,Loop[R0] ; ili BLT  R2,R3,Loop1[R0]
Exit:
```

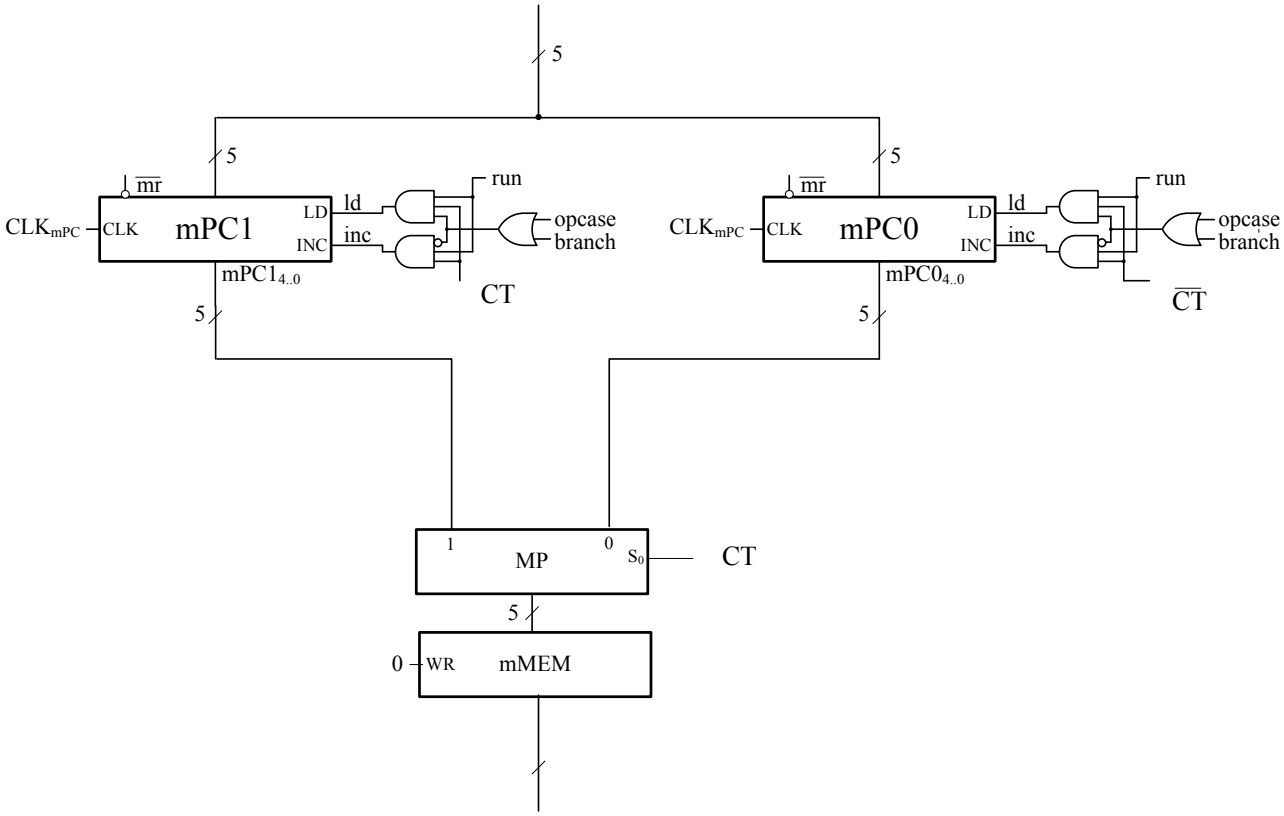
Prethodni kod se može i optimizovati tako da izbegnemo dva skoka u svakoj iteraciji:

```
            XOR   R2,R2,R2
            LOAD  R3,n[R0]
            BNLT  R2,R3,Exit[R0]
Loop1:      LOAD  R4,a[R2]
            ADD   R4,R4,R2
            STORE R4,a[R2]
            ADD   R2,R2,R1
            BLT   R2,R3,Loop1[R0]
Exit:
```

c)

```
 (P0)00, 01, 02,
 (P1)00, 01, 02,
 (P0)03, 04, 08, 09, 0A, 00, 01, 02, 03, 04, 05, 06, 07,
       // ovog puta je podatak u kešu
     0B,0C,0D,0E,0F,
 (P1)03, 04, 08, 09, 0A, 00, 01, 02, 03, 04, 05, 06, 07...
```

d) Tražena šema prikazana je na narednoj slici:



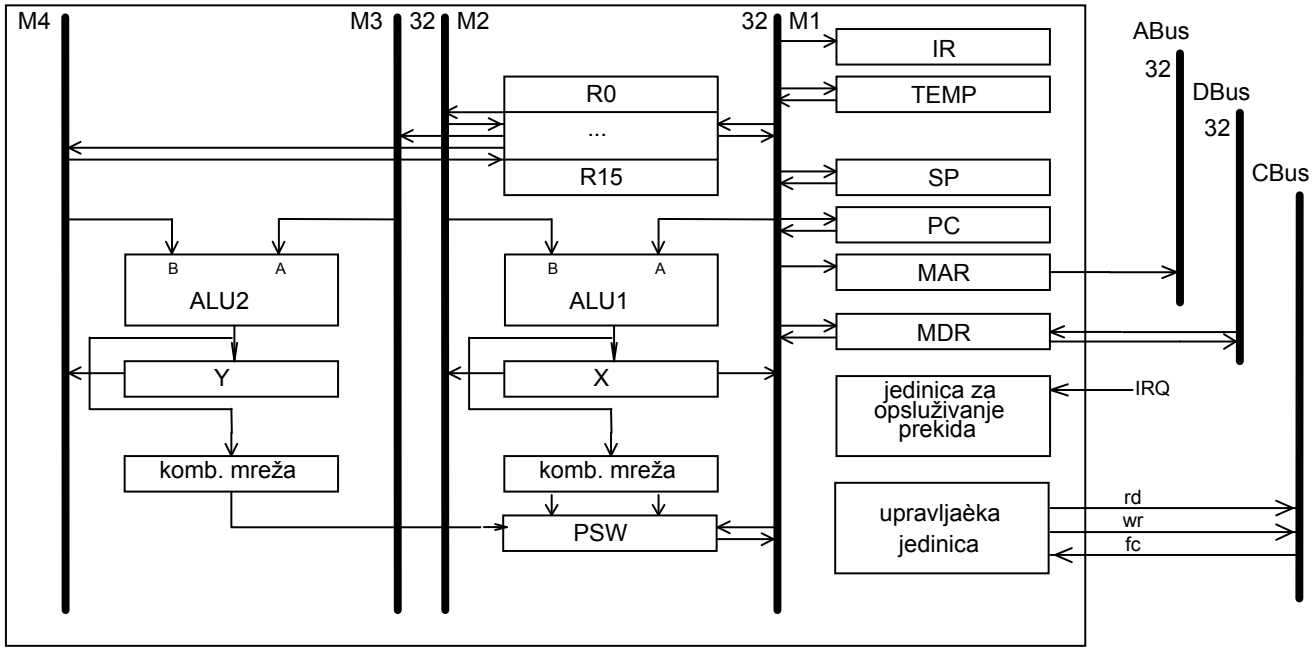
Zadatak 9

Procesor je troadresni i ima 16 registara opšte namene, R0 do R15, svi su 32-bitni. Postoje i registri PSW i SP sa uobičajenim značenjem. Memorijske adrese su širine 32 bita, širina magistrale podataka je 32 bita, a adresiranje je na nivou 32-bitnih reči. Procesor operiše samo sa 32-bitnim celobrojnim veličinama (u daljem tekstu *reč* označava 32-bitnu veličinu). Vreme odziva memorije je neodređeno, magistrala je sinhrona. Postoji samo jedan ulaz IRQ za spoljašnji maskirajući prekid. Pri prekidu se na steku čuvaju PC i PSW tim redom i maskirajući prekid se onemogućava brisanjem bita I u registru PSW. Ovom prekidu odgovara fiksno ulaz 0 u vektor tabeli koja počinje od adrese 0. Stek raste prema višim adresama, a SP ukazuje na prvu slobodnu lokaciju.

Pored uobičajenih troadresnih aritmetičko-logičkih instrukcija (svi operandi su u registrima), procesor poseduje posebnu instrukciju za obavljanje dve troadresne aritmetičko-logičke operacije u isto vreme (paralelno). Svi operandi su isključivo u registrima, a prvi operand je u oba slučaja odredište. Nijedna od operacija ne može biti CMP. Ova instrukcija ima sledeću sintaksu u asembleru: najpre se navodi mnemonik ALU, a zatim se u istom redu pišu dve operacije, npr.:

```
ALU  ADD R0,R2,R3, SUB R15,R2,R1
```

U registru PSW postoje dva skupa indikatora N, Z, C i V. Na prvi skup dejstvuje prva, a na drugi druga operacija. Ova instrukcija se koduje pomoću jedne reči. U poljima *REGi*, *i*=5,...,0 instrukcijske reči nalaze se kodovi 6 registara koji se upotrebljavaju u ovoj instrukciji. Za razmenu podataka sa memorijom postoje samo instrukcije LOAD i STORE koje imaju dvoadresni format. Izvorište za STORE i odredište za LOAD su isključivo registri R0..R15, a drugi operand je definisan nekim od načina adresiranja (usvojiti proizvoljan skup načina adresiranja).



Mogu se koristiti sve potrebne ostale instrukcije u programiranju, sa odgovarajućim mnemonicima. Obe ALU imaju, pored ostalih, i kontrolni signal *incA*. Registar X ima mogućnost brisanja (*clX*).

Zadatak:

a) Definisati mrežu koja će vršiti pravilnu selekciju registara R_0 do R_{15}. Upis u registre i čitanje iz registara se vrši signalima REGin i REGout. Potrebno je definisati i sve selekcione signale.

b) Napisati mikroprogram za ovaj procesor, sa fazom izvršavanja **samo** za instrukciju ALU, a predvideti postojanje ostalih. Kôd treba da bude prilagođen mikroprogramskoj upravljačkoj jedinici, pri

čemu se u jednoj mikronaredbi nalazi i polje sa upravljačkim signalima i polja koja definišu uslovni skok u mikroprogramu. **Treba** pisati i mikroprogram za obradu spoljašnjeg prekida. Pretpostaviti da je dohvatanje eventualne druge reči instrukcije u fazi izvršavanja instrukcija koje poseduju tu reč.

c) Napisati na asembleru ovog procesora program koji realizuje sledeću sekvencu naredbi iz programskog jezika C:

```
c=a-b;
d=c-b;
e+=13;
if(e<0) goto lab1;
else if(d==0) goto lab2;
```

Pretpostaviti da su promenljive a, b, c, d i e celobrojne i da se nalaze u registrima R0 do R4, respektivno.

d) Ukoliko prevodilac može da sastavi bilo koje dve susedne aritmetičke operacije iz izvornog programa u jednu ALU složenu instrukciju, navesti koji sve konflikti mogu da nastupe zbog zavisnosti po podacima, a koji ne mogu. Precizno objasniti zašto (mogu ili ne mogu). Navesti primer za sve slučajeve. Predložiti hardver kojim bi procesor detektovao postojanje svakog od konflikata koji može da nastane.

Rešenje:

a) Neka upravljački signal REGSEL*ij* definiše da se za selekciju registra koji će biti povezan sa internom magistralom M*j* (bilo ulaz ili izlaz) koristi polje REG*i* u instrukcijskoj reči. Neka signal R*ij**REGout (logičko I) otvara trostatičke bafere registra R*i* prema magistrali M*j*, a signal R*ij**REGin upisuje u registar R*i* sa magistrale M*j*. Na izlaze IR treba vezati 6 dekodera 4/16, redom na polja REG5..REG0. Označimo te dekodere sa DC5..DC0. Neka je izlazna linija *j* dekodera DC*k* označena sa DC*kl*, *k*=5..0, *l*=15..0. Signal R*ij* određen je izrazom:

$$R_{i,j} = \sum_{k=0}^{5} DC_{k,i} \cdot REGSEL_{k,j}$$

Glavna razlika ovakve mreže i mreža realizovanih u prethodnim zadacima posledica je činjenice da u istom taktu možemo čitati registre (iste ili različite) adresiranih različitim poljima instrukcije, pa nam nije dovoljan jedan multiplekser koji će uvek propustati jednu vrednost i time adresirati samo jedan registar.

b)

```
; Dohvatanje instrukcije
BEGIN:     PCout,MARin,incA1,ldX
           read,Xout1,PCin
           wmfc
           MDRout,IRin
; Dekodiranje instrukcije
           opcase     ; za ilegalan kod operacije, skok na INTH
; ALU
ALU:       REGSEL41,REGSEL32,REGSEL13,REGSEL04,REGout,ALUOP1,
                          ALUOP2,ldX,ldY,ldPSW1,ldPSW2
           Xout2,Yout,REGSEL52,REGSEL24,REGin,branch(!IReq,begin)
           bruncnd(INTH)
; Interrupt handler za spoljašnji prekid (IRQ)
INTH:      SPout,MARin,incA1,ldX
           PCout,MDRin
           write,Xout1,incA1,ldX,SPin
           wmfc
           SPout,MARin
           PSWout,MDRin
           write,Xout1,SPin,clX
           wmfc
           Xout1,MARin
           read
           wmfc
           MDRout,PCin,clIRR,clT,bruncnd(BEGIN)
```

c)

```
LOAD  R5,#13
SUB   R2,R0,R1
ALU   SUB R3,R2,R1 ADD R4,R4,R5
BRN2  lab1
BRZ   lab2
```

Na ovom primeru je ilustrovano i kako bi izgledala naredba uslovnog skoka. Zavisno od toga koji uslov želimo da ispitujemo, naredba skoka će imati sufiks 2 ili ne. Sa sufiksom 2, naredba ispituje drugi set flegova u registru PSW koji je rezultat druge operacije, a bez sufiksa se ispituje prvi set flegova koji je rezultat prve operacije. Naredba skoka bez sufiksa treba da se koristi i kada se radi uobičajene ALU operacije, koje nisu složene, jer se tada rezultat operacije odražava na prvi set indikatora.

d) Mogu da nastupe sledeći konflikti zbog:

1. Prave zavisnosti po podacima: druga operacija koristi rezultat prve kao svoj operand; nastupa zato što se operandi obe operacije čitaju paralelno, što znači da se operandi druge operacije čitaju pre nego što je prva upisala rezultat (*Read after Write*). Primer:

```
ADD   R3, R1, R2
SUB   R4, R5, R3
```

Ne može se zameniti sa:

```
ALU   ADD  R3, R1, R2  SUB  R4, R5, R3
```

Zbog identične situacije nismo mogli spojiti prve dve naredbe u tački c) već smo spojili drugu i treću.

2. Izlazne zavisnosti po podacima (*Write after Write*): obe operacije imaju isti registar za rezultat, pa upisuju u isti registar u isto vreme; rezultat treba da bude određen drugom operacijom. Primer:

```
ADD   R3, R1, R2
SUB   R3, R5, R4
```

Zamena koja izaziva konflikt:

```
ALU   ADD  R3, R1, R2  SUB  R3, R5, R4
```

Zavisno od realizacije registara, nespre čavanje ovakvih situacija može dovesti do oštećenja hardvera.

Ne može da nastupi sledći konflikt zbog:

3. Antizavisnosti po podacima (*Write after Read*).: ako druga operacija upisuje u registar koji je operand prve operacije, to neće dogoditi, jer se upis izvršava u taktu posle takta u kome je prva operacija već pročitala operande. Primer:

```
ADD R4, R5, R3
SUB R3, R1, R2
```

može se zameniti sa:

```
ALU   ADD R4, R5, R3  SUB  R3, R1, R2
```

Konflikt se može detektovati pomoću komparatora koji upoređuju kodove registara koji se koriste kao operandi/rezultati operacija, prema navedenim pravilima. Ukoliko se nađe jednakost odgovarajućih kodova za bilo koji konfliktni slučaj (ILI funkcija signala jednakosti iz komparatora), postoji konflikt.