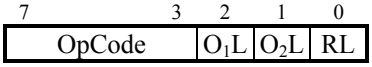
Zadatak 3

Opis arhitekture procesora

Procesor je nula-adresni (stek-mašina). Stek raste prema višim memorijskim lokacijama, a SP pokazuje na prvu slobodnu lokaciju. Memorijske adrese su širine 16 bita, širina magistrale podataka je 8 bita, adresiranje je na nivou jednog bajta, a dvobajtni podaci se u memoriju smeštaju tako da je na nižoj adresi niži bajt. Procesor operiše samo sa 8-bitnim i 16-bitnim celobrojnim veličinama. Na dužine operanada i rezultata ukazuju biti 0 do 2 instrukcijske reči. Vreme odziva memorije je neodređeno, magistrala je asinhrona.

Postoje samo interni prekidi usled nedozvoljenog koda operacije i prekoračenja (*overflow*). Prekidni mehanizam je vektorisan, a navedenim prekidima dodeljeni su fiksni ulazi 0 i 2 u vektor tabeli, respektivno. Na tebelu sa adresama prekidnih rutina ukazuje 16-bitni registar IVTP.

Instrukcije su dužine jedan bajt, a samo u nekim slučajevima dva ili tri bajta, kada se u druga dva bajta nalazi konstanta (instrukcija PUSH *const*). U prvoj reči su uvek kôd operacije i informacije o dužini operanada i rezultata. Instrukcija ima format kao na slici.

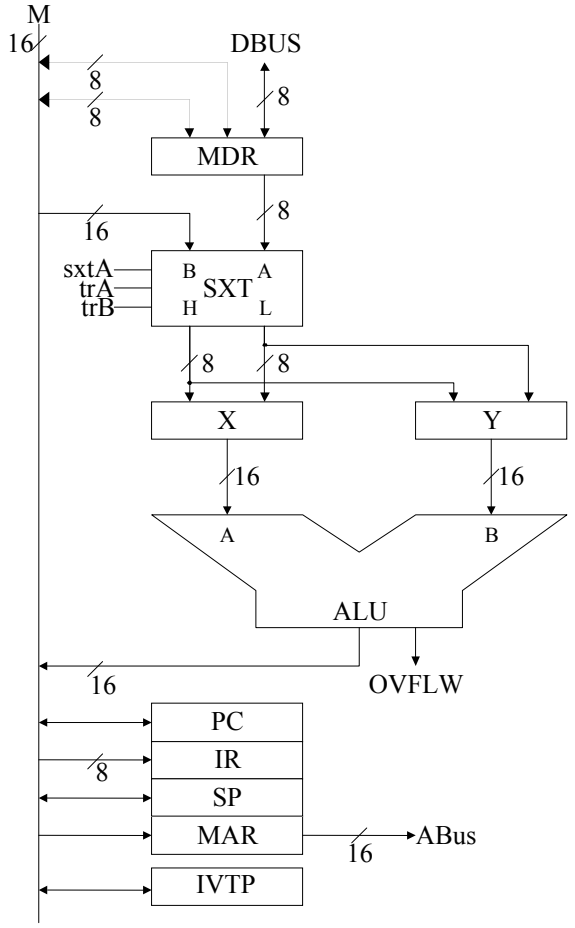


Skup instrukcija sa vrednostima polja *OpCode* dat je na slici. Oznake u uglastim zagradama označavaju dužinu podatka: O_1L označava dužinu $operanda_1$, O_2L dužinu $operanda_2$, a *RL* dužinu rezultata. Isto značenje imaju i odgovarajuća polja u instrukciji, tako da vrednost 0 znači širinu od 8 bita, a vrednost 1 širinu od 16 bita. $Operand_2$ je na vrhu steka, a $operand_1$ ispod njega. Asemblerske naredbe imaju još sufikse B ili W za dužine $operanda_1$, $operanda_2$ i rezultata, respektivno (npr. ADDWBW).

Vrednost	Naredba	Efekat naredbe
00h	**PUSH** *const*	**push** *const*[*RL*]
01h	**PUSH**	**pop** *addr*[16]; **push** (*addr*)[*RL*] ; indirektno adresiranje
02h	**POP**	**pop** *addr*[16]; **pop** *op*[*RL*]; (*addr*)*:=op*[*RL*]
03h	**SWAP**	**pop** $op_1[O_1L]$; **pop** $op_2[O_2L]$; **push** $op_1[O_1L]$; **push** $op_2[O_2L]$
04h	**INC**	**pop** *op*[*RL*]; *op:=op+1*; **push** *op*[*RL*]
05h	**DEC**	**pop** *op*[*RL*]; *op:=op-1*; **push** *op*[*RL*]
06h	**ADD**	**pop** $op_2[O_1L]$; **pop** $op_1[O_2L]$; $res:=op_1 + op_2$; **push** *res*[*RL*]
07h	**SUB**	**pop** $op_2[O_1L]$; **pop** $op_1[O_2L]$; $res:=op_1 - op_2$; **push** *res*[*RL*]
08h	**MUL**	**pop** $op_2[O_1L]$; **pop** $op_1[O_2L]$; $res:=op_1 * op_2$; **push** *res*[*RL*]
09h	**AND**	**pop** $op_2[O_1L]$; **pop** $op_1[O_2L]$; $res:=op_1 \,\&\, op_2$; **push** *res*[*RL*]
0Ah	**OR**	**pop** $op_2[O_1L]$; **pop** $op_1[O_2L]$; $res:=op_1 \mid op_2$; **push** *res*[*RL*]
0Bh	**XOR**	**pop** $op_2[O_1L]$; **pop** $op_1[O_2L]$; $res:=op_1 \,\hat{}\, op_2$; **push** *res*[*RL*]

Organizacija procesora

Na slici prikazana je principijelna šema organizacije procesora. ALU ima mogućnost obavljanja sledećih operacija: *inc, dec, add, sub, mul, and, or, xor*. Kombinaciona mreža označena sa SXT obavlja sledeće operacije: *sxtA* (8-bitni ulaz A proširuje znakom i 16-bitni rezultat daje na izlaz), *transferA* (8-bitni ulaz A daje i na viši i na niži bajt izlaza) i *transferB* (propušta 16-bitni ulaz B na izlaz). Registri X i Y imaju odvojene kontrolne signale za upis u niži i u viši bajt.



Zadatak

a) Nacrtati detaljnu šemu kombinacione mreže označene sa SXT.

b) Nacrtati šemu kombinacione mreže koja će na internu magistralu postaviti vrednost ulaza odgovarajućeg prekida kada se generiše upravljački signal $ivte_{out}$. Signal koji je aktivan kada je kôd operacije nedozvoljen smatrati definisanim, dok signal koji je aktivan kada je nastupilo prekoračenje izlazi iz ALU.

c) Napisati mikroprogram za ovaj procesor (osim za naredbe sa kodom 00h do 05h, ali predvideti njihovo postojanje i postojanje eventualnih ostalih naredbi), tako da bude

prilagođen mikroprogramskoj upravljačkoj jedinici, pri čemu su mikronaredbe sa upravljačkim signalima i mikronaredbe koje definišu skok u mikroprogramu razdvojene.

d) Na asembleru ovog procesora napisati deo programa koji realizuje sledeću naredbu dodele (Pascal):

```
d[i] := a[i] * b + c;
```

Svi podaci u naredbi su 16-bitni celi brojevi.

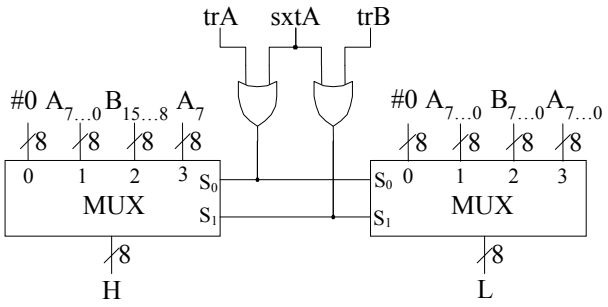
e) Predložiti naredbu uslovnog skoka koja bi se u potpunosti uklopila u nula-adresni format. Ova naredba treba da je dugačka jedan bajt sa kôdom operacije. Predložiti mnemonik i objasniti značenje naredbe. Naredba treba da omogući izvršavanje naredbi višeg programskog jezika tipa:

```
if (expression1 < expression2) then ….
```

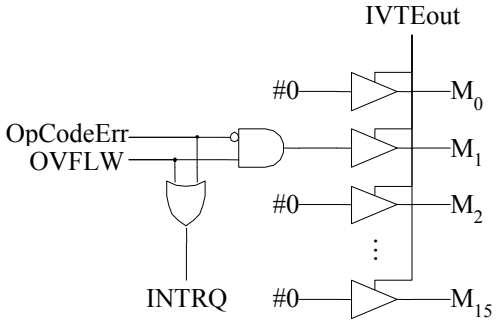
Objasniti kako bi se prevodila data naredba višeg programskog jezika.

Rešenja

a)



b)



Kom prekidu je dat prioritet ovako realizovanom mrežom?

c)

```
BEGIN:      PCout,MARin,trB,XHin,XLin
            read,inc,ALUout,PCin
            wmfc
            MDRout,IRin
            opcase

; Binarne operacije
; Dohvatanje drugog operanda
BIN:        SPout,trB,XHin,XLin
            dec,ALUout,SPin,MARin,trB,XHin,XLin
            read
            wmfc
            branch(O2L,FOP2L)
            sxtA,YHin,YLin
; Dohvatanje prvog operanda:
FOP1:       dec,ALUout,SPin,MARin,trB,XHin,XLin
            read
            wmfc
            branch(O1L,FOP1L)
            sxtA,XHin,XLin
BINOP:      ALUop,ALUout,MDRLin
            write
            wmfc
            ALUOP,ALUout,MDRHin
            SPout,trB,XHin,XLin
            inc,ALUout,SPin,MARin,trB,XHin,XLin
            branch(RL==0,END)
            write,inc,ALUout,SPin
            wmfc
END:        branch(IRQ,INTH)
            bruncnd(BEGIN)
FOP2L:      trA,YHin,dec,ALUout,MARin,SPin
            read,SPout,trB,XHin,XLin
            wmfc
            trA,YLin
            bruncnd(FOP1)
FOP1L:      dec,ALUout,SPin,MARin
            read,trA,XHin
            wmfc
            trA,XLin
            bruncnd(BINOP)
            ;obrada prekida
INTH:       PCout, MDRLin
            SPout, MARin, trB, XHin, XLin
            write
            wmfc
            PCout, MDRHin
            inc, ALUout, MARin, trB, XHin, XLin
            write, inc, ALUout, SPin
            wmfc
            IVTEout, trB, XHin, XLin, YHin, YLin
            add, ALUout, trB, YHin, YLin ; množenje sa dva
            IVTPout, trB, XHin, XLin
            add, ALUout, MARin, trB, XHin, XLin
            read
```

```
                wmfc
                inc, ALUout, MARin
                read, trA, YLin, XLin
                wmfc
                tra, YHin, XHin ; jedini način da propustimo
                or, ALUout, PCin; sadržaj registra X
                                ; ili Y kroz ALU
                bruncnd (BEGIN)
```

d)

```
PUSHW a
PUSHW i
PUSHW
PUSHB 2
MULWBW
ADDWWW
PUSHW
PUSHW b
PUSHW
MULWWW
PUSHW c
PUSHW
ADDWWW
PUSHW d
PUSHW i
PUSHW
PUSHW i  ; drugi način za indeksiranje
PUSHW    ; (bez množenja)
ADDWWW
ADDWWW
POPW
```

e) Tražena instrukcija može da ima mnemonik:

```
BZ    (Branch on Zero)
```

i značenje:

pop *addr*[16]; **pop** *op*[*RL*]; **if** (*op* = 0) **goto** *addr;*

Analogno, mogu da postoje i instrukcije BN (*Branch on Negative*) i BP (*Branch on Positive*).

Mogli smo usvojiti i drugačiji efekat ovakvih instrukcija. Na primer:

```
BZ    (Branch on Zero)
```

pop *addr*[16]; **pop** $op_2[O_2L]$; **pop** $op_1[O_1L]$; $res:=op_1 - op_2$; **if** (*op* = 0) **goto** *addr;*

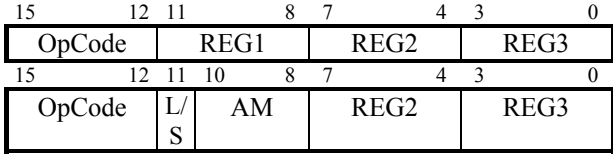
Analogno bi važilo i za instrukcije BN (*Branch on Negative*) i BP (*Branch on Positive*).

Zadatak 4

Opis arhitekture procesora

Procesor je troadresni i ima 16 registara opšte namene, R0 do R15, svi su 16-bitni. Postoje i registri PSW i SP sa uobičajenim značenjem. Memorijske adrese su širine 16 bita, širina magistrale podataka je 16 bita, a adresiranje je na nivou 16-bitnih reči. Procesor operiše samo sa 16-bitnim celobrojnim veličinama (u daljem tekstu *reč* označava 16-bitnu veličinu). Vreme odziva memorije je neodređeno, magistrala je asinhrona.

Postoje sledeće grupe instrukcija: troadresne instrukcije (aritmetičke, logičke itd.), dvoadresne instrikcije za prenos podataka (LOAD i STORE), jednoadresne instrukcije (CLR, INC, DEC, PUSH, POP itd.), instrukcije skokova (bezuslovni i uslovni) i ostale (manipulacije indikatorima, poziv potprograma, povratak iz potprograma ili prekida itd.). Instrukcije su dužine jedne ili dve reči. U prvoj reči su uvek kôd operacije i informacije o načinu adresiranja operanada. Samo kod instrukcija prenosa podataka (LOAD i STORE) postoji druga reč u kojoj je adresa ili neposredni operand. Format troadresnih instrukcija i instrukcija LOAD i STORE dat je na slici. Bit L/S označava da li se radi o LOAD ili STORE instrukciji.



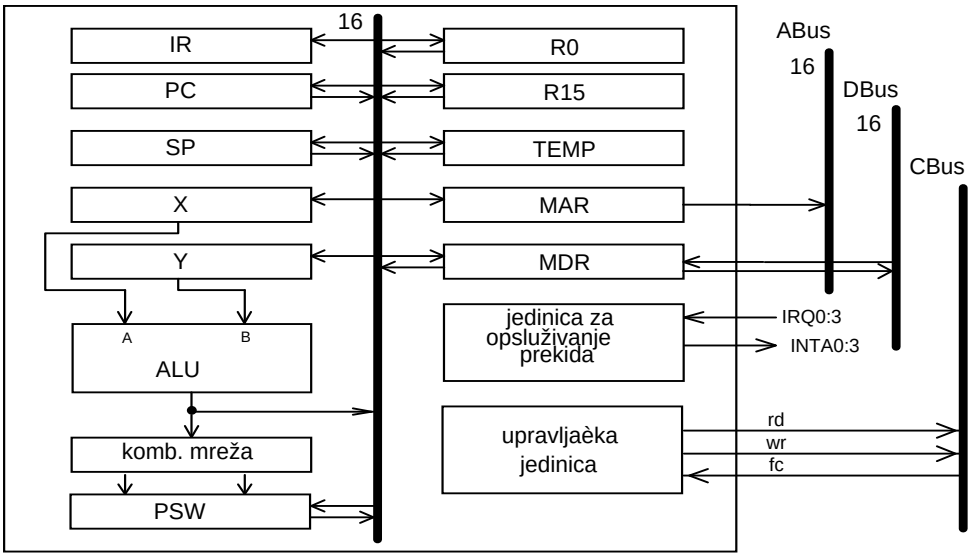
Kod svih instrukcija (osim dvoadresnih LOAD i STORE), operandi i odredište su isključivo u registrima R0 do R15. Troadresne instrukcije imaju sledeći format: polje *OpCode* sadrži kôd operacije, polja *REG1* i *REG2* kodove registara (od 0 do 15) u kojima su prvi i drugi operand, a polje *REG3* kôd registra u koji se smešta rezultat.

Kod dvoadresnih instrukcija za prenos podataka (LOAD i STORE), odredište za LOAD i izvorište za STORE su isključivo registri R0 do R15. Ovaj operand biće nazivan prvim operandom. Drugi operand je neposredni podatak u drugoj reči instrukcije (samo za LOAD), u nekom od registra R0 do R15, ili u memoriji. Samo ove instrukcije operišu podacima u memoriji. Ove instrukcije imaju sledeći format prve reči: polje *OpCode* sadrži kôd operacije (1100), polje L/S određuje smer prenosa (0-LOAD, 1-STORE), polje *AM* sadrži kôd načina adresiranja drugog operanda, polje *REG2* sadrži kôd registra u kome je drugi operand kod registarskog direktnog adresiranja, a polje *REG3* kôd registra u kome je prvi operand koji je uvek u registru (odredište za LOAD i izvorište za STORE). Postoje četiri načina adresiranja drugog operanda, kao što je prikazano na slici.

AM	Značenje	Primer u asembleru
000	Neposredno adresiranje	LOAD R0, #1234h
010	Registarsko direktno	STORE R1, R3
100	Memorijsko direktno	STORE R2, 0100h
101	Memorijsko indirektno	LOAD R1,[0100h]

Organizacija procesora

Organizacija procesora data je na slici. ALU ima, pored ostalih, i kontrolni ulaz *incA* za inkrementiranje vrednosti na A ulazu. Mogu se koristiti sve potrebne ostale instrukcije u programiranju, sa odgovarajućim mnemonicima.



Zadatak:

a) Nacrtati strukturnu šemu mreže koja povezuje izlaze registra R*i* ($i = 0 \ldots 15$) sa internom magistralu M kada je aktivan upravljački signal REGout i jedan od signala regsel1, regsel2 i regsel3, koji služe za selekciju registra pomoću polja REG1, REG2 i REG3 instrukcijske reči.

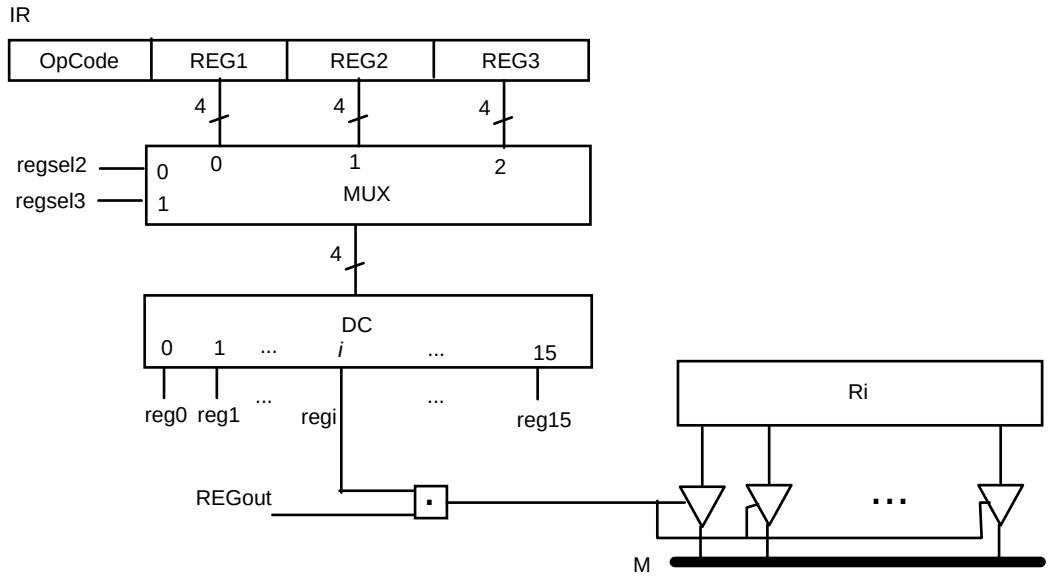
b) Napisati mikroprogram za ovaj procesor, sa fazom izvršavanja samo za instrukciju LOAD (za sve načine adresiranja) i sve binarne aritmetičko/logičke operacije (obrada treba da bude u jedinstvenom mikrokodu), a predvideti postojanje ostalih. Kôd treba da bude prilagođen mikroprogramskoj upravljačkoj 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, ali treba predvideti njegovo postojanje (poziv na odgovarajućim mestima). Dohvatanje eventualne druge reči instrukcije treba da bude u fazi izvršavanja instrukcije.

c) Napisati na asembleru ovog procesora program koji inkrementira svaki element niza reči počev od adrese 100h. Niz je dugačak onoliko koliko pokazuje sadržaj lokacije 99h. Lokacija 98h je slobodna za korišćenje.

d) Koji način adresiranja treba dodati procesoru da bi se broj pristupa memoriji kod pristupa elementima niza, poput onog iz prethodne tačke, smanjio? Objasniti značenje i definisati način kodiranja ovog načina adresiranja, tako da se što bolje uklopi u dati format instrukcija LOAD i STORE. Modifikovati prethodni program tako da se iskoristi nov način adresiranja.

Rešenje

a) Tražena mreža prikazana je na slici.



b)

```
; Dohvatanje instrukcije
BEGIN:      PCout,MARin,Xin
            read,incA,ALUout,PCin
            wmfc
            MDRout,IRin
; Dekodovanje instrukcije
            opcase
; LOAD instrukcija
LOAD:       admodld           ; način adresiranja za LOAD
; Neposredno adresiranje
LDIMM:      PCout,MARin,Xin
            read,incA,ALUout,PCin
            wmfc
            MDRout,regsel3,REGin,branch(IRR,INTH)
            bruncnd (BEGIN)
; Registarsko direktno adresiranje
LDRD:       regsel2,REGout,TEMPin
            TEMPout,regsel3,REGin,branch(IRR,INTH)
            bruncnd (BEGIN)
; Memorijsko direktno adresiranje
LDMD:       PCout,MARin,Xin
            read,incA,ALUout,PCin
            wmfc
            MDRout,MARin
            read
            wmfc
            MDRout,regsel3,REGin,branch(IRR,INTH)
            bruncnd (BEGIN)
; Memorijsko indirektno adresiranje
LDMI:       PCout,MARin,Xin
            read,incA,ALUout,PCin
```

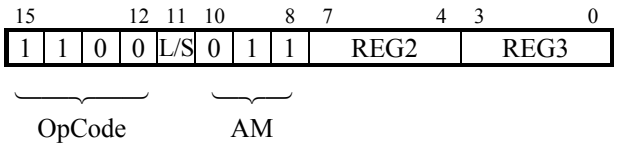
```
                wmfc
                MDRout,MARin
                read
                wmfc
                MDRout,MARin
                read
                wmfc
                MDRout,regsel3,REGin,branch(IRR,INTH)
                bruncnd(BEGIN)
; binarne operacije
BINOP:          REGout, Xin
                Regsel2,REGout Yin
              ALUop,ALUout,ldPSW,regsel3,REGin,branch(IRR,INTH)
                bruncnd(BEGIN)
```

c) Traženi program je dat na slici.

```
START:      LOAD        R0,99h      ; R0:=n
            OR          R0, R0, R0
            JZ          END
            LOAD        R1,#100h    ; R1:=100h
            STORE       R1,98h      ; M[98h]:=&a[0]

LOOP:       LOAD        R2,[98h]    ; R2:=a[i]
            INC         R2          ; R2:=R2+1
            STORE       R2,[98h]    ; a[i]:=R2
            INC         R1          ; R1:=R1+1
            STORE       R1,98h      ; M[98h]:=&a[i]
            DEC         R0          ; R0:=R0-1
            JNZ         LOOP        ; NEXT
END:
```

d) Potrebno je dodati registarsko indirektno adresiranje. Kodovanje može biti:




```
START:      LOAD        R0,99h      ; R0:=n
            OR          R0, R0, R0
            JZ          END
            LOAD        R1,#100h    ; R1:=100h

LOOP:       LOAD        R2,[R1]     ; R2:=a[i]
            INC         R2          ; R2:=R2+1
            STORE       R2,[R1]     ; a[i]:=R2
            INC         R1          ; R1:=R1+1
            DEC         R0          ; R0:=R0-1
            JNZ         LOOP        ; NEXT
END:
```

Zadatak 5

Opis arhitekture procesora

Procesor je troadresni i ima 8 registara opšte namene, R0 do R7, 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). Ulazno/izlazni i memorijski adresni prostori su razdvojeni. Vreme odziva memorije je neodređeno, magistrala je asinhrona.

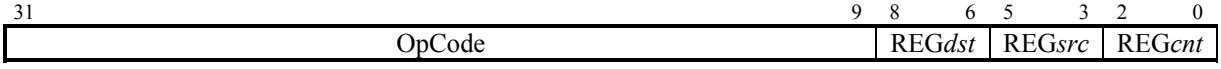
Postoje spoljašnji maskirajući prekidi, za koje zahtevi dolaze po linijama IRQ0 do IRQ7 procesora, pri čemu su svi ulazi istog prioriteta. Pri prekidu se na steku čuvaju PC, PSW i svi R0 do R7, tim redom, i svi maskirajući prekidi se onemogučavaju brisanjem bita I u PSW.

Postoje sledeće grupe instrukcija: troadresne instrukcije (aritmetičke, logičke itd.), dvoadresne instrukcije za prenos podataka (MOV, IN i OUT), jednoadresne instrukcije (CLR, INC, DEC, PUSH, POP itd.), instrukcije skokova (bezuslovni i uslovni) i ostale (manipulacije indikatorima, poziv potprograma, povratak iz potprograma ili prekida itd.). Instrukcije su dužine jedne do tri reči. U prvoj reči su uvek kôd operacije i informacije o načinu adresiranja operanada. Prvi operand je i odredište.

Instrukcija MOV ima dva operanda, i oba mogu biti u svim dozvoljenim načinima adresiranja; prvi operand je i odredište. Instrukcije IN i OUT imaju jedan operand u registru R*i*, a drugi operand može biti određen svim dozvoljenim načinima adresiranja. Ostale instrukcije mogu imati sve operande u svim dozvoljenim načinima adresiranja. Načini adresiranja su dati na slici.

Način adresiranja	Primer u asembleru
Neposredno	MOV R0, #1234h
Registarsko direktno	MOV R1, R3
Registarsko indirektno	MOV (R2), #1234
Registarsko indirektno sa pomerajem	MOV Pom(R1),#0

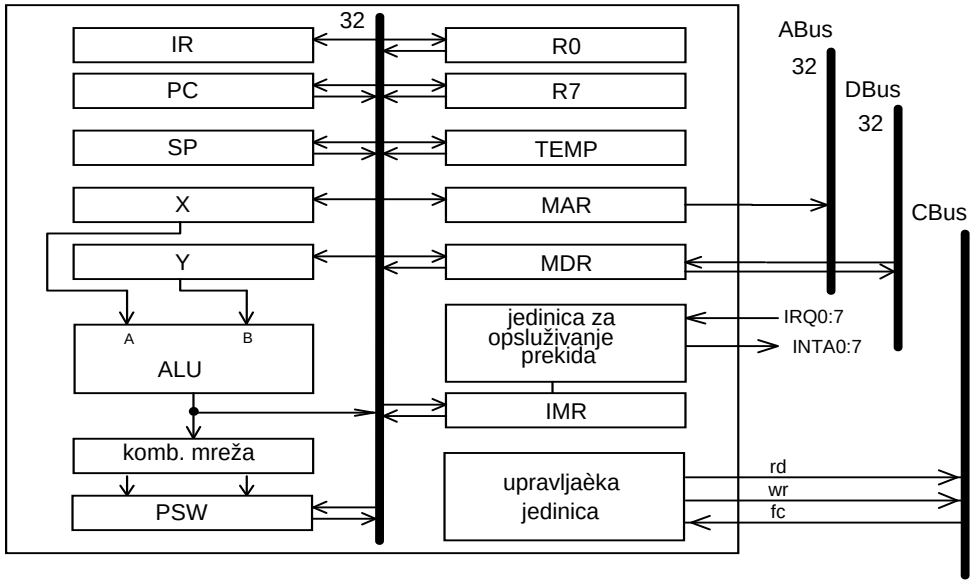
Procesor poseduje posebnu instrukciju za rad sa nizovima reči u memoriji (*string* instrukcija). To je troadresna instrukcija MOVS R*dst*,R*src*,R*cnt*. Ona kopira blok memorijskih reči sa jednog mesta na drugo. Adresa početka izvorišnog bloka je u registru R*src*, adresa početka odredišnog bloka je u registru R*dst*, a dužina bloka je u registru R*cnt*, gde su R*dst*,R*src*,R*cnt* registri opšte namene (R0 do R7). Sadržaj registra R*cnt* može biti i nula. Ova instrukcija menja vrednosti ovih registara. Na kraju izvršavanja ove instrukcije, R*src* i R*dst* ukazuju na prvu memorijsku reč iza bloka izvorišta/odredišta, R*cnt* ima vrednost nula, a indikator Z se postavlja na 1. Prekid se obrađuje tek na završetku cele instrukcije MOVS. Format ove instrukcije je dat na slici.



Za potrebe maskiranja prekida postoji 32-bitni registar IMR. Biti 0 do 7 ovog registra maskiraju ulaze IRQ0 do IRQ7, redom; ostali biti nisu značajni. Jedine instrukcije za rad sa ovim registrom su: MOV IMR,R*i*, i MOV R*i*,IMR, gde je R*i* jedan od R0 do R7. Dejstvo ovih instrukcija, vidljivo za programera, je sledeće. Instrukcija MOV IMR,R*i* prebacuje osam najmlađih bita registra R*i* u najmlađe bite registra IMR, a ostale bite registra IMR postavlja na nulu. Instrukcija MOV R*i*,IMR prebacuje 32 bita registra IMR u registar R*i*. Osim toga, postoje još i instrukcije INTE i INTD za dozvolu/maskiranje svih maskirajućih prekida.

Organizacija procesora

Organizacija procesora data je na slici. ALU ima, pored ostalih, i kontrolne ulaze *incA*, *decA* i *transA* za inkrementiranje, dekrementiranje, odnosno transfer vrednosti na A ulazu. Mogu se koristiti sve potrebne ostale instrukcije u programiranju, sa odgovarajućim mnemonicima.



Zadatak:

a) Sa koliko flip-flopova treba realizovati registar IMR? Koristeći registar sa tim brojem razreda i mogućnošću paralelnog upisa (LD), prikazati način vezivanja registra IMR sa internom magistralom.

b) Napisati mikroprogram za ovaj procesor, sa fazom izvršavanja samo za instrukciju MOVS, 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 i treće reči instrukcije u fazi izvršavanja instrukcija koje poseduju te reči (ne treba realizovati).

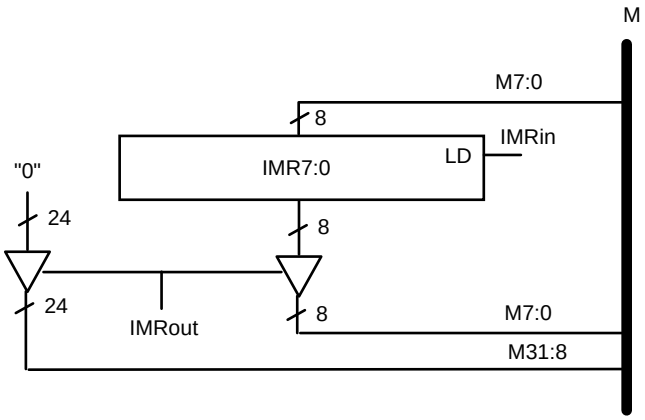
c) Odluka projektanta procesora da se prekid opslužuje tek posle izvršavanja cele instrukcije MOVS nije dobra. Objasniti zašto.

d) Da bi se problem iz prethodne tačke eliminisao, odlučeno je da se realizacija instrukcije MOVS promeni tako da se postojanje prekida ispituje, i eventualni prekid opslužuje, posle svake prenesene reči (posle svake iteracije petlje). Problem je što se tada instrukcija prekida u sredini, pa je povratak iz prekidne rutine na narednu instrukciju nekorektan. Kratko, ali precizno objasniti kompletno rešenje ovog problema.

e) Efekat izvršavanja instrukcije MOVS realizovane na način prikazan u tački b) nije korektan za sve moguće vrednosti odredišnog i izvorišnog operanda. Objasniti zašto i napisati ispravan mikroprogram za ovu instrukciju. Predvideti da se obrada prekida za instrukciju MOVS vrši na bolji način, u skladu sa diskusijom u tačkama c) i d).

Rešenje

a)



b) Pretpostavka je da postoji kombinaciona mreža koja na osnovu signala `regsel1`, `regsel2` i `regsel3` selektuje odgovarajući registar (slično kao u prethodnim zadacima).

```
; Dohvatanje instrukcije
BEGIN:      PCout,MARin,Xin
            read,incA,ALUout,PCin,Xin
            wmfc
            MDRout,IRin
; Dekodovanje instrukcije
            opcase
; MOVS instrukcija
MOVS:       regsel3, REGout, Xin ; provera da li je Rcnt=0
            transA,ldpswz      ; postavljanje indikatora Z
            branch(Z,MOVSEND)  ; if Rcnt=0, goto end
            decA, ALUout, regsel3, REGin
            regsel2, REGout, MARin, Xin
            read, incA, ALUout, regsel2, REGin
            wmfc
            regsel1, REGout, MARin,Xin
            write, incA, ALUout, regsel1, REGin
            wmfc
            bruncnd(MOVS)
MOVSEND:    branch(IReq,INTH)
            bruncnd(BEGIN)
```

c) Izvršavanje MOVS instrukcije može da traje veoma dugo. Na primer, MOVS R0,R0,R1, gde je R0=0, R1=FFFFFFFFh. Za sve vreme izvršavanja ove instrukcije, svi spoljni prekidi su onemogućeni, što nikako nije dobro. Naime, ova instrukcija predstavlja petlju, u čijoj se svakoj iteraciji prenosi po jedna reč, sasvim nezavisno od ostalih. Zbog toga, vreme od trenutka pojave zahteva za prekid, do trenutka njegovog prihvatanja, može da varira u jako velikim granicama. To nije dobro, jer se ni približno ne može predvideti brzina odziva na neki spoljni događaj. To vreme zavisi i od tekuće instrukcije, ali i od vrednosti njenih operanada. Bolje je da se prekid opslužuje posle svake iteracije, jer je time vreme odziva na prekid svedeno na vreme prenosa samo jedne reči.

d) Posle svake iteracije petlje MOVS (posle svakog prenosa jedne reči), stanje procesora je konzistentno, u potpunosti kao što je to i pre početka izvršavanja MOVS instrukcije. Naime, posle svake iteracije, R*src* i R*dst* ukazuju na narednu reč za prenos, a R*cnt* pokazuje na broj preostalih reči za prenos (može biti i nula). Naredba MOVS upravo zahteva takvo stanje pre svog izvršavanja. Prema tome, navedeno stanje registara predstavlja *invarijantu* ove instrukcije. Zato se, posle prekida iza jedne iteracije, može započeti ista MOVS instrukcija, sa tekućim stanjem registara R*src*, R*dst* i R*cnt*, koji se ionako hardverski čuvaju na steku, i restauriraju pri povratku iz prekidne rutine. Potrebno je, dakle, da se procesor iz prekidne rutine vrati na *prekinutu* instrukciju MOVS, a ne na narednu instrukciju. Ovo se lako realizuje, tako što se, ako postoji prekid posle neke iteracije, vrednost PC smanji za 1, tako da ukazuje na tekuću MOVS instrukciju, i ta se vrednost stavlja na stek u mikroprogramu za obradu prekida. To je ujedno i sve što je potrebno za rešenje problema.

e) Problem nastaje u slučaju kada se odredišni i izvorišni niz preklapaju. U mikroprogramu realizovanom u tački b), izvorišni niz se čitao od početka i smeštao u odredišni. Za slučaj kada se nizovi preklapaju a adresa izvorišnog niza je manja od adrese odredišnog, tada će se dogoditi slučaj da se izvorni niz modifikuje pre nego što se odgovarajuće lokacije prepišu u odredište. Da bismo to izbegli, u ovom slučaju moramo da prepisivanje niza vršimo počev od kraja, kao što je to realizovano u nastavku:

```
 ; Dohvatanje instrukcije
BEGIN:      PCout,MARin,Xin
            read,incA,ALUout,PCin,Xin
            wmfc
            MDRout,IRin
; Dekodovanje instrukcije
            opcase
; MOVS instrukcija
MOVS:       Regsel1, REGout, Yin
            Regsel2, REGout, Xin
            sub, ALUout, ldpswn
MOVS0:      regsel3, REGout, Xin
            transA,ldpswz      ; postavljanje indikatora Z
            branch(Z,MOVSEND)  ; if Rcnt=0, goto end
            decA,ALUout,regsel3,REGin,Xin,branch(N, MOVS2)
```

```
                regsel2, REGout, MARin, Xin
                read, incA, ALUout, regsel2, REGin
                wmfc
                regsel1, REGout, MARin,Xin
                write, incA, ALUout, regsel1, REGin
                wmfc
                branch(IReq,MOVSINTH)
                bruncnd(MOVS0)

MOVS2:          regsel2, REGout, Yin
                add, ALUout, MARin
                read, regsel1, REGout, Yin
                wmfc
                add, ALUout, MARin
                write
                wmfc
                branch(IReq,MOVSINTH)
                bruncnd(MOVS0)

MOVSEND:        branch(IR,MOVSINTH)
                bruncnd(BEGIN)
MOVSINTH:       PCout,Xin
                decA, ALUout, PCin, bruncnd(INTH)
```

Treba primetiti da u slučaju kada niz popunjavamo od nazad, registri R*src* i R*dst* zadržavaju svoje stare vrednosti, a R*cnt* se dekrementira dok ne postane nula. Takođe treba primetiti da nismo mogli registre R*src* i R*dst* na početku mikrorograma, u slučaju da se prepisivanje vrši od nazad, povećati za vrednost veličine niza pa dekrementirati u svakoj iteraciji (iako se takvo rešenje prirodno nameće). Ovo nije bilo moguće zbog toga što se provera signala prekida vrši posle svake iteracije. Naime, stanje svih registara posle povratka iz prekida bi bilo konzistentno, ali ne bismo mogli da izbegnemo ponovno uvećavanje registara R*src* i R*dst*, a oni ne bi smeli više od jednom da se uvećaju za dužinu niza.