1. zadatak: *(4 boda)* Za Turingov stroj napisati program koji proizvoljni broj predočen u oktalnom brojevnom sustavu (koji je inicijalno zapisan na vrpci) povećava za dva. Pretpostavlja se da je početni položaj glave za čitanje i pisanje na najznačajnijoj poziciji. Nacrtati Turingov stroj i 6. konfiguraciju stroja za početni izraz 27.

Rješenje:

Zadatak se sastoji od tri dijela, koje ćemo odvojeno riješiti u nastavku:

(1) Napisati program za T.S.
(2) Nacrtati T.S.
(3) Nacrtati 6. konfiguraciju za zadani početni izraz

(1) Napisati program koji uvećava zapisani broj za 2.

Zadatak je moguće riješiti na više načina. Npr., moguće je odjednom uvećati broj za 2, ili ga je moguće dva puta uvećavati za 1. Ovdje će biti prikazana varijanta koja se meni čini najjednostavnijom, a temelji se na prvom predloženom rješenju, tj. uvećavanju broja odjednom za 2.

Ovdje je prikazano samo rješenje u obliku tablice s vrlo kratkim objašnjenjima za pojedina stanja (okvir sa strane). Detaljnija diskusija dana je na idućoj stranici.

	q_0	q_1	q_2
0	$0q_0D$	2!N	1!N
1	$1q_0D$	3!N	2!N
2	$2q_0D$	4!N	3!N
3	$3q_0D$	5!N	4!N
4	$4q_0D$	6!N	5!N
5	$5q_0D$	7!N	6!N
6	$6q_0D$	$0q_2L$	7!N
7	$7q_0D$	$1q_2L$	$0q_2L$
^	^q_1L	^!N	1!N

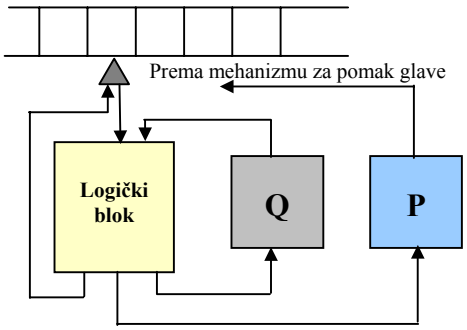
Stanja:

q_0 – pozicioniranje na najmanje značajnu znamenku

q_1 - uvećavanje znamenke za 2.

q_2 - uvećavanje znamenke za 1 (radi prijenosa)

(2) Nacrtati Turingov stroj



(3) Nacrtati 6. konfiguraciju za početni izraz 27.

Općenito, k-ta konfiguracija se definira kao stanje stroja **na početku** k-tog takta. Kao što ne postoji nulti takt, ne postoji ni nulta konfiguracija; **početna konfiguracija je prva.** K-tu konfiguraciju crtamo tako da nacrtamo stanje na traci na početku k-tog takta, a ispod znaka na kojem se nalazi glava za čitanje i pisanje, naznačimo stanje u kojem se stroj u danom trenutku nalazi.

Prateći tablicu iz prvog dijela zadatka, za naš slučaj dobivamo sljedeći niz konfiguracija

1. konfiguracija:

			2	7			
			q_0				

2. konfiguracija:

			2	7			
			q_0				

3. konfiguracija:

			2	7			
				q_0			

4. konfiguracija:

			2	7			
			q_1				

5. konfiguracija:

			2	1			
			q_2				

6. konfiguracija:

			3	7			
			!				

Detaljni opis rješenja

Kao što znamo, Turingov stroj funkcionira tako da na temelju stanja u kojem se trenutno nalazi i na temelju znaka koji trenutno čita s trake, određuje novo stanje u koje će prijeći, novi znak koji će zapisati na traku, te pomak glave za čitanje i pisanje za jedno mjesto ulijevo ili udesno ili ostanak glave na istom mjestu. Program za Turingov stroj pišemo stoga u obliku tablice u kojoj na temelju trenutnog stanja i pročitanog znaka očitavamo novo stanje, novi znak te pomak glave.

Stanja označavamo s q_i ($i \in N_0$). Stroj se, po definiciji, početno nalazi u stanju q_0. U zadatku je također zadano da se glava za čitanje i pisanje stroja početno nalazi na najznačajnijoj (tj. krajnje lijevoj) znamenci broja.

Međutim, uvećavanje broja moramo započeti od najmanje značajne znamenke (tj. one krajnje desne). Stoga ćemo iskoristiti stanje q_0 da bismo glavu za čitanje i pisanje pomaknuli na najmanje značajnu znamenku. Naravno, s obzirom da se glava stroja po definiciji u jednom koraku može pomaknuti najviše za jedno mjesto, nećemo taj pomak moći ostvariti odjednom, nego u više koraka, pomičući glavu jedno po jedno mjesto udesno, ali sve u stanju q_0. To znači da stroju moramo reći otprilike ovo: "Ako si u stanju q_0 i na traci čitaš bilo koju oktalnu znamenku, tada ostavi na traci tu istu znamenku, ostani u istom stanju (q_0) i pomakni se za jedno mjesto u desno." Kako ćemo to reći stroju? To ćemo mu reći tako da, kad budemo pisali tablicu koja odgovara programu Turingovog stroja, u stupcu koji odgovara stanju q_0, za svaki redak koji odgovara nekoj oktalnoj znamenci, definiramo novo stanje jednako starom stanju (q_0), novi znak na traci jednak starom znaku (pročitanoj oktalnoj znamenci) te pomak glave u desno. Na taj način, stroj će sa svojom glavom "protrčati" kroz sve znamenke zapisanog broja i konačno dospjeti na prazninu koja se nalazi desno od najmanje značajne znamenke. Kad smo konačno došli do te praznine, znamo da se najmanje značajna znamenka nalazi jedno mjesto ulijevo. Prema tome, ako je stroj u stanju q_0 i nalazi se na praznini, treba na traci ostaviti prazninu, prijeći u novo stanje q_1 i pomaknuti se ulijevo za jedno mjesto.

Novo stanje q_1 morali smo uvesti jer je proces traženja najmanje značajne znamenke sada završen i treba pristupiti drugoj fazi, tj. samom uvećavanju broja za 2. To ne bismo mogli raditi u stanju q_0 jer smo već ranije definirali da u tom stanju stroj, ako se nalazi na bilo kojoj oktalnoj znamenci (pa tako i onoj najmanje značajnoj, na koju smo se upravo pozicionirali), radi nešto drugo: ostavlja tu znamenku na traci i pomiče se desno.

Dakle, u stanju q_1 uvećavamo broj za 2. To znači, ako se na traci nalazi znamenka 0, treba umjesto nje zapisati 2; ako se na traci nalazi 1, treba zapisati 3, itd... U slučaju, dakle, da je riječ o nekoj maloj znamenci (0 do 5), upisujemo na traku novu vrijednost, uvećanu za 2 i stroj može stati (tj. prijeći u konačno stanje, koje se dogovorno označava kao "q_F" ili jednostavno "!").

Ako je na traci znamenka 6, pošto se radi o oktalnom brojevnom sustavu, uvećanjem za 2 dobivamo 0 i onaj "jedan dalje" tj. prijenos. Slično, za znamenku 7, uvećanjem za 2 dobivamo 1 i prijenos. U tom slučaju, moramo, nakon zapisivanja nove vrijednosti (0 za 6, odnosno 1 za 7) pomaknuti glavu stroja za jedno mjesto u lijevo i pridodati taj prijenos sljedećoj znamenci broja (tj. uvećati tu slijedeću znamenku za 1). To ne možemo ostvariti niti u jednom od već postojećih stanja, pa uvodimo novo stanje q_2, čija je uloga voditi računa o prijenosu. Dakle, u stanju q_2 uvećavamo trenutnu znamenku za 1. Ako je ta znamenka bila između 0 i 6, tada ćemo nju samo povećati za 1 i stati (stroj ide u konačno stanje q_F tj. !). Ako je, međutim, ta znamenka bila 7, uvećavanjem za 1 dobivamo 0 i ponovno "jedan dalje" tj. prijenos. U tom slučaju moramo se opet pomaknuti za jedno mjesto u lijevo i uzeti u obzir prijenos koji se dogodio, a za tu svrhu već imamo stanje q_2 (dakle stroj u njemu i ostaje).

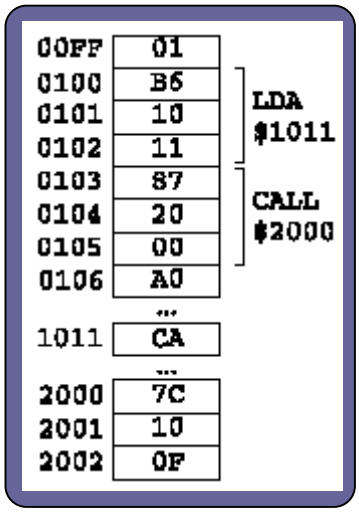
Jedna specifična situacija može se dogoditi ako stroj u stanju $\mathbf{q_1}$ naiđe na prazninu. to se može dogoditi jedino ako nikakav broj nije bio inicijalno zapisan na traci. U tom slučaju, ovdje predloženo rješenje (koje nipošto nije i jedino) predlaže da se stroj naprosto zaustavi i ne piše ništa.

Ukoliko se stroj nađe na praznini dok je u stanju q_2, to znači da smo prilikom uvećavanja najznačajnije znamenke dobili prijenos, pa ćemo, prema tome, od n-znamenkastog broja dobiti (n+1)-znamenkasti. U tom slučaju samo umjesto praznine pišemo 1 (kao da je prethodno pisala 0) i stajemo.

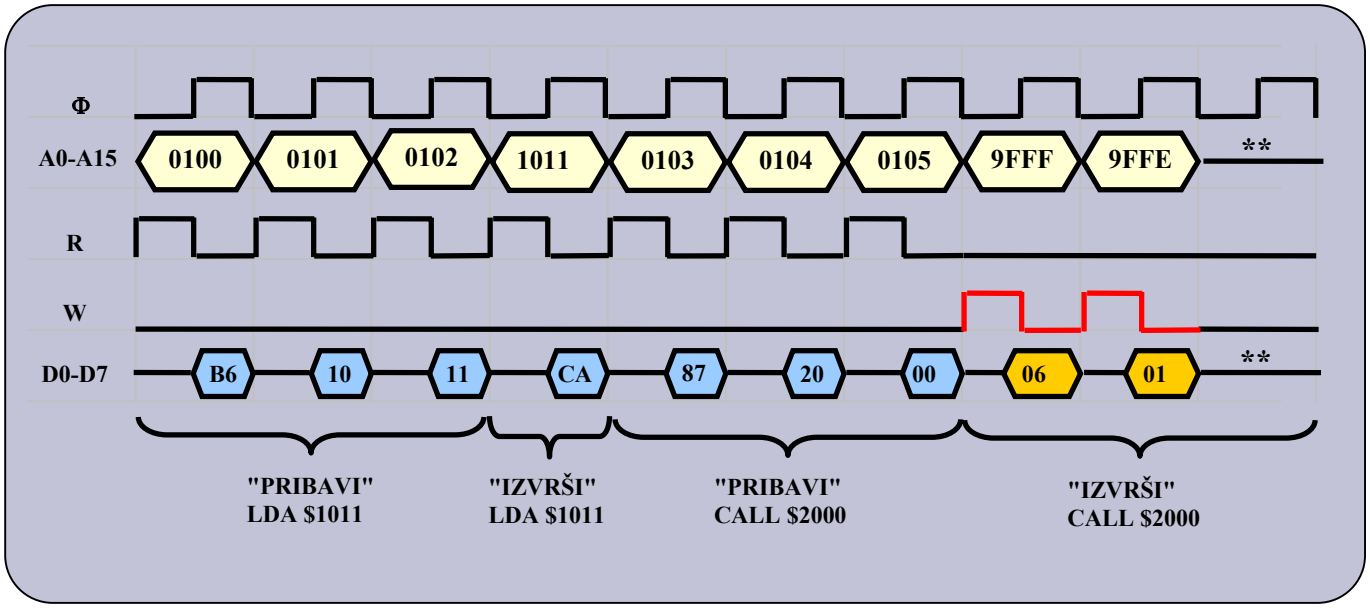
2. zadatak: *(5 bodova)* Pretpostavite da je jednostavnom modelu mikroprocesora prododan još i 16-bitni registar *Kazalo stoga* (SP). Za tako modificirani model nacrtajte stanje na sabirnicama za sljedeći programski odsječak

```
LDA $1011
CALL $2000    ;poziv potprograma
```

Početne vrijednosti registara modela su A=XX, SP=A000 i PC=0100, a sadržaj memorije prikazan je na slici desno. Odredite sadržaje registara modela koji su se promijenili tijekom izvođenja programskog odsječka.



Rješenje:



Sadržaji registara: $A=CA_H$; $DC=2000_H$; $PC=2000_H$; $SP=9FFE_H$; $IR=87_H$

Detaljni opis rješenja

Zadani programski odsječak sastoji se od dvije instrukcije. Prva instrukcija je LDA $1011. Ta instrukcija čita bajt s navedene adrese (1011_H) i stavlja ga u registar A. Kao što možemo vidjeti na slici memorije (koja je zadana u zadatku), instrukcija se sastoji od tri bajta. Prvi bajt je operacijski kod instrukcije, dok druga dva bajta predstavljaju adresu sa koje će se pročitati operand.

Druga instrukcija je CALL $2000, koja poziva potprogram koji se nalazi na adresi 2000_H. Sa slike vidimo da se i ona sastoji od tri bajta. Prvi bajt je operacijski kod, a druga dva predstavljaju adresu potprograma.

Pri izvođenju svake instrukcije, procesor prolazi kroz dvije faze: (1) faza PRIBAVI, u kojoj se pribavlja (tj. čita) instrukcija iz memorije, te (2) faza IZVRŠI, u kojoj se izvršava sama instrukcija. Kako smo rekli da se prva instrukcija sastoji od tri bajta, bit će za njenu fazu PRIBAVI potrebne tri periode signala vremenskog vođenja (takta) – po jedna za čitanje svakog bajta. Čitanje određenog bajta se na sabirnici očituje tako da sa najprije na adresnoj sabirnici pojavljuje adresa memorijske lokacija sa koje želimo pročitati bajt. Tu adresu postavlja procesor, i to na početku same periode takta. Procesor također aktivira i signal R (engl. Read = čitaj), kako bi naznačio memoriji da želi pročitati podatak sa te adrese, a ne zapisati ga. Nakon određenog vremena kašnjenja (tzv. vrijeme pristupa memorije - za koje ćemo pretpostaviti da iznosi oko pola periode signala takta), memorija odgovara tako da na sabirnicu podataka postavlja traženi podatak. Kod pojednostavljenog modela mikroprocesora, adresna sabirnica je 16-bitna (njene linije označavamo s A_0-A_{15}), a podatkovna sabirnica je 8-bitna

(D_0-D_7). Zbog toga što je sabirnica podataka 8-bitna, potrebne su nam već spomenute tri periode signala vremenskog vođenja da bi se pročitala tri bajta instrukcije.

Nakon što je faza PRIBAVI prve instrukcije završena, slijedi njena faza IZVRŠI. U toj fazi, sukladno definiciji same instrukcije, čita se bajt sa željene adrese (1011_H), koji potom odlazi u registar A. Za pročitati jedan bajt dovoljna nam je jedna perioda signala vremenskog vođenja. Tijekom te periode, procesor na adresnu sabirnicu postavlja adresu 1011_H te također postavlja signal R. Nakon isteka vremena pristupa, memorija odgovara postavljanjem sadržaja koji se nalazi na toj adresi (a to je u našem slučaju, kao što se vidi na slici memorije, vrijednost CA_H). Ta vrijednost odmah odlazi u regostar A, te je ovime faza IZVRŠI prve instrukcije dovršena. S obzirom da se A kasnije neće mijenjati, to je i konačna vrijednost u A.

Slijedi faza PRIBAVI druge instrukcije. Princip je potpuno jednak kao i kod prethodne instrukcije: tijekom tri periode signala vremenskog vođenja, čitaju se tri bajta koja odgovaraju instrukciji.

Faza IZVRŠI instrukcije CALL $2000 je najzanimljivija. Da bismo nacrtali stanje na sabirnicama tijekom te faze, moramo razmisliti kako instrukcija CALL radi. Instrukcija CALL je instrukcija za pozivanje potprograma. Kada se govori o pozivu potprograma (a ne o najobičnijem skoku ili grananju), tada se pretpostavlja da mora postojati način **povratka** iz potprograma nazad (instrukcija RET ili nešto slično). Nakon povratka iz potprograma, procesor bi trebao nastaviti s izvođenjem one instrukcije koja se nalazi neposredno iza instrukcije CALL (u našem zadatku ne znamo koja je to instrukcija, ali to nije ni bitno). Ovo je moguće postići samo ako se prilikom izvršavanja naredbe CALL negdje pohrani ta povratna adresa (koja se nakon pribavljanja instrukcije CALL nalazi u programskom brojilu – PC, ali će doskora, kad dođe do prijenosa upravljanja na sam potprogram, biti prebrisana adresom prve instrukcije potprograma, tj. netragom će nestati). S obzirom da smo našem procesoru pridodali mogućnost rada sa stogom, povratnu adresu, tj. sadržaj programskog brojila, pohranit ćemo na stog.

Stog je jedno najobičnije područje u radnoj memoriji, nad kojim su definirane dvije operacije: PUSH (stavi na stog) i POP (skini sa stoga) kako bi se ostvarila LIFO (last in – first out) struktura: dakle na stog je moguće stavljati podatke i s njega skidati podatke, ali samo tako da onaj podatak koji je zadnji stavljen može biti prvi skinut. Vrh stoga mora u svakom trenutku biti dobro definiran. Adresa vrha stoga nalazi se u posebnom registru, nazvanom *kazalo stoga* – SP (engl. Stack Pointer).

Programsko brojilo, kao što znamo, sadrži adresu slijedeće instrukcije. S obzirom da je adresna sabirnica pojednostavljenog modela procesora 16-bitna, programsko brojilo je također 16-bitno (jer sadrži *adresu* instrukcije, a ta adresa će vjerojatno prije ili kasnije završiti na adresnoj sabirnici). Da bismo pohranili sadržaj 16-bitnog programskog brojila u memoriju (tj. na stog) preko 8-bitne podatkovne sabirnice, bit će nam potrebne dvije periode signala vremenskog vođenja (takta). Dakle tu povratnu adresu mi ćemo rastaviti na dva bajta, te bajt po bajt pohraniti na stog.

Kad nešto stavljamo na stog, možemo reći da stog "raste". Slično, kad nešto skidamo sa stoga, kažemo da stog "pada". Postoji nekoliko konvencija vezanih uz način kako stog raste i pada i uz način na koji je definirana adresa vrha stoga koja se nalazi u kazalu stoga. Pri rješavanju ovog zadatka, mi ćemo pretpostaviti onu konvenciju koja je uobičajena kod većine procesora: da stog raste prema nižim memorijskim adresama, te da kazalo stoga sadrži adresu zadnje pune lokacije (a ne prve prazne).

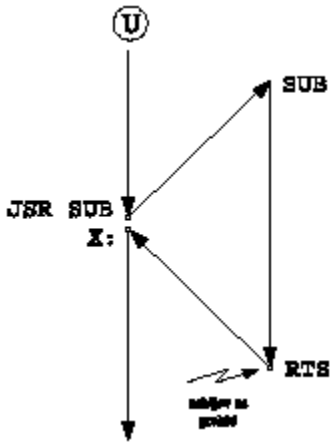
Uz te pretpostavke, da bi se bajt **stavio** na stog, potrebno je najprije dekrementirati (= umanjiti za 1) kazalo stoga (kako bi pokazivalo na prvu praznu, slobodnu lokaciju), te na tako dobivenu adresu zapisati željeni bajt.

Iz slike memorije vidimo da pojednostavljeni model procesora koristi tzv. *big-endian* način pohrane podataka. To znači da su podaci koji se sastoje od više od jednog bajta pohranjeni u memoriji tako da se viši bajt nalazi na nižoj adresi, a niži bajt na višoj adresi. Npr., vrijednost 1011_H u instrukciji LDA $1011, zapisan je tako da se vrijednost 10_H (viši bajt) nalazi na adresi 101 (niža, tj. manja adresa), dok se vrijednost 11_H (niži bajt) nalazi na adresi 102 (viša adresa).

O tome moramo voditi računa kada pohranjujemo programsko brojilo na stog. Pošto smo pretpostavili uobičajenu konvenciju da naš stog raste prema padajućim adresama, u prvom koraku pohrane programskog brojila pristupat ćemo višoj adresi; nakon toga se SP smanjuje i u slijedećem koraku pristupamo tako smanjenoj (dakle nižoj) adresi. Poštujući *big-endian* način zapisa, u prvom koraku ćemo na podatkovnu sabirnicu staviti niži bajt programskog brojila, a u drugom viši.

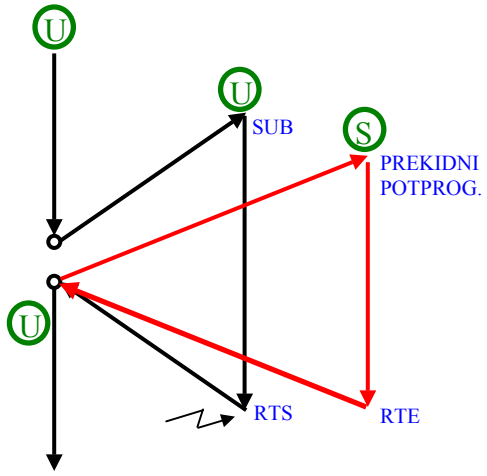
Kad je povratna adresa konačno pohranjena na stog, još je potrebno u programsko brojilo PC staviti novu adresu – onu na kojoj se nalazi potprogram. Ta je adresa (2000_H u našem slučaju) pročitana iz memorije još za vrijeme faze PRIBAVI i nalazi se u registru *brojilo podataka* – DC (engl. Data Counter). Potrebno ju je, dakle, prebaciti iz tog registra u registar PC. Pretpostavit ćemo da nam je za taj prijenos između registara potrebna još jedna perioda takta. Međutim, to je operacija koja se događa interno u samom procesoru, ona nema nikakve veze s memorijom, tako da će procesor za to vrijeme biti odspojen od sabirnice. Na sabirnici se neće ništa vidjeti, nego će ona u za to vrijeme biti u stanju visoke impedancije.

3. zadatak: *(5 bodova)* Na slici desno prikazan je scenarij vezan uz MC68000. Adresa potprograma SUB je 0010FFFA, a povratna adresa X je 0020FFAA.



a) Nacrtajte daljnji slijed događaja uz pretpostavku da se zahtjev za prekid dogodio upravo tijekom instrukcijskog ciklusa za RTS.
b) Označite načine rada procesora.
c) Nacrtajte stanje stogova u ključnim točkama tijekom prijenosa upravljanja (grananje u potprogram, prijenos upravljanja na prekidni potprogram, vraćanje iz potprograma, vraćanje iz prekidnog potprograma). *Opaska:* Za vrijednosti koje nisu zadane ili koje se ne mogu točno utvrditi koristite oznake tipa WWXXYYZZ te pri tome navedite značenje pojedinih bajtova ili riječi.
d) Prikažite, dodatnom skicom, što će se dogoditi ako se umjesto instrukcije RTE zabunom kodira instrukcija RTS. Također odredite stanje stogova i način rada procesora nakon tako izvršene instrukcije RTS.

Rješenje:

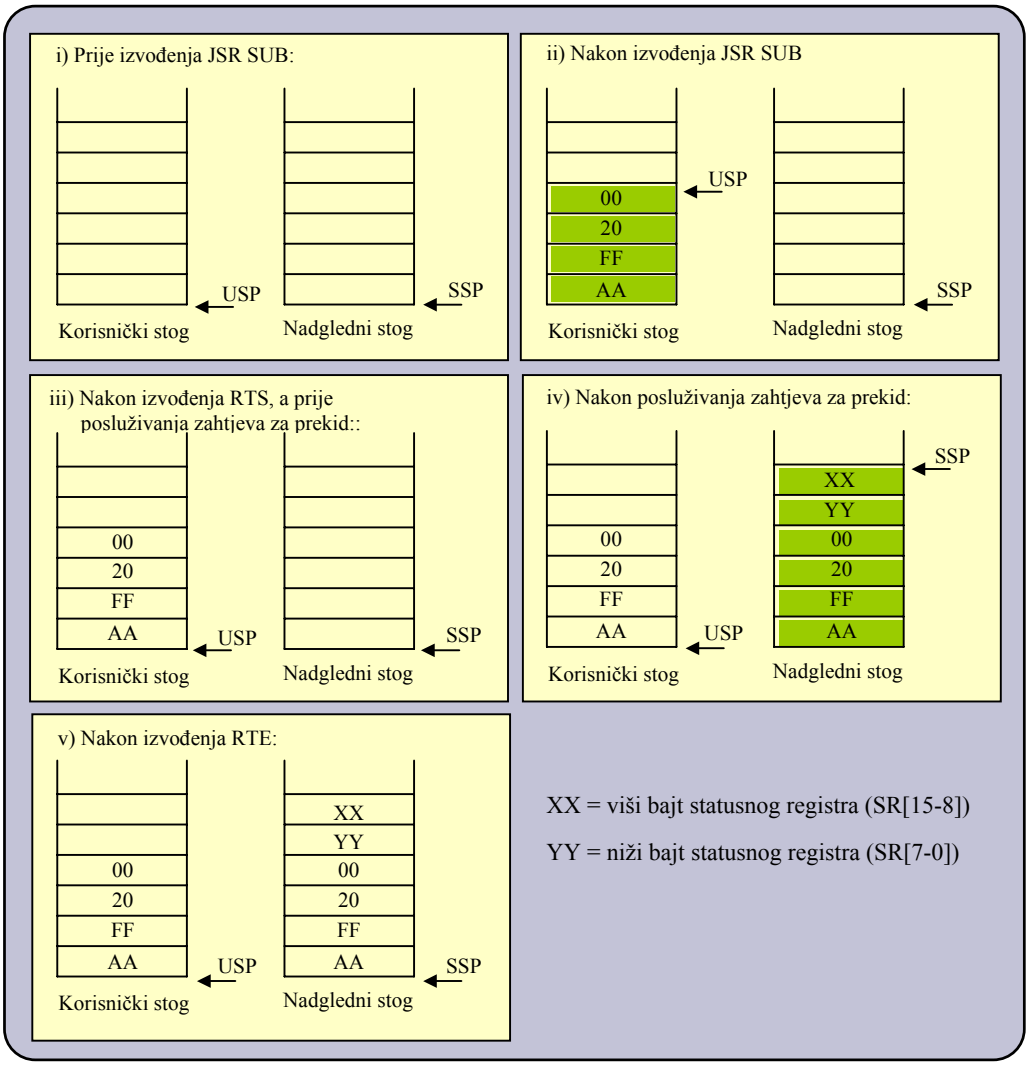


a) Obrada prekida po definiciji započinje tek nakon dovršetka izvođenja tekuće instrukcije. U ovom slučaju, tekuća instrukcija je RTS, kojom se ostvaruje povratak iz potprograma. Dakle, najprije će se ostvariti povratak iz potprograma, a tek potom se odlazi na obradu prekida. Ovaj tijek događaja prikazan je na slici crvenom bojom.

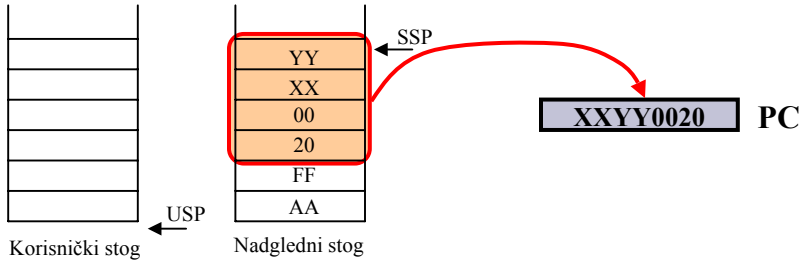
b) Procesor iz korisničkog načina rada u nadgledni prelazi isključivo iznimkom. S obzirom da je i prekid jedna vrsta iznimke, procesor nakon zahtjeva za prekid prelazi u nadgledni način rada. Način rada u kojem se procesor u određenom trenutku nalazi, određen je zastavicom S u statusnom registru (S=0 – korisnički način rada; S=1 – nadgledni način rada). S obzirom da se prilikom posluživanja zahtjeva za prekid (kao i svake druge iznimke) stari sadržaj statusnog registra sprema na stog, a prilikom povratka iz prekida (ili iznimke) vraća sa stoga natrag u statusni registar, tako će prilikom povratka iz prekida biti vraćen i stari sadržaj zastavice S (kakav je bio prije prekida) te se procesor vraća u onaj način rada u kojem je bio prije prekida. U našem slučaju, dakle, vraća se u korisnički način rada. Načini rada procesora u karakterističnim točkama za zadani scenarij prikazani su na slici zelenom bojom.

c) Procesor MC68000 ima dva stoga: korisnički i nadgledni. U svakom trenutku dostupan je i koristi se samo jedan od njih, i to onaj koji odgovara načinu rada u kojem se procesor trenutno nalazi. U korisničkom načinu rada koristit će se korisnički stog; u nadglednom načinu rada koristit će se nadgledni stog. Kod pozivanja potprograma (instrukcija JSR), sprema se na stog samo povratna adresa (sadržaj programskog bojila). Ta povratna adresa je 32-bitna, te se na stog pohranjuje kao 4 bajta, i to slijedeći big-endian način zapisivanja. Pošto procesor MC68000, poput većine drugih procesora ima stog ostvaren tako da raste prema padajućim adresama, najprije će na stog biti zapisan najmanje značajni bajt. U našem slučaju, procesor se prilikom pozivanja potprograma nalazio u korisničkom načinu rada, pa će zato ova četiri bajta povratne adrese biti zapisana na korisnički stog.
Prilikom prekida (odnosno iznimke), na stog se pohranjuje tzv. minimalni kontekst, a on se sastoji od 4 bajta povratne adrese (slično kao i kod pozivanja potprograma) i još dodatno od 2 bajta statusnog registra. To je ukupno 6 bajtova, koji se također spremaju na trenutni stog. No s obzirom da procesor prilikom prekida (odnosno iznimke) obavezno prelazi u nadgledni način rada, ovih 6 bajtova minimalnog konteksta uvijek će se pohranjivati na nadgledni stog.
Prilikom povratka iz potprograma odnosno obrade prekida/iznimke, situacija je inverzna: sa tekućeg stoga skida se 4 bajta povratne adrese, koji se stavljaju u PC (u slučaju instrukcije RTS koja služi za

povratak iz potprograma), odnosno 6 bajtova minimalnog konteksta, koji se stavljaju u SR i PC (u slučaju instrukcije RTE koja služi za povratak iz obrade prekida/iznimke).
Cjelokupni sadržaj stogova u svim karakterističnim točkama scenarija prikazan je slikom.



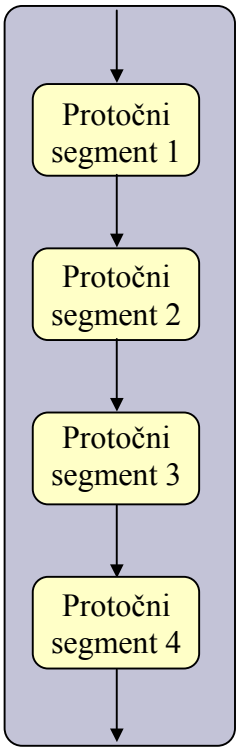
d) Ukoliko bi se umjesto instrukcije RTE zabunom kodirala instrukcija RTS, tada bi, sukladno opisu ponašanja instrukcije RTS danom pod c), s trenutnog stoga (a to je nadgledni) bila skinuta 4 bajta koja bi potom bila stavljena u programsko brojilo. No, kao što vidimo na slici, 4 bajta koja se nalaze na vrhu nadglednog stoga sastoje se od 2 bajta statusnog registra i viša 2 bajta pohranjene povratne adrese. To znači da bi se u programskom brojilu našla nova (i pogrešna) adresa, koja se sastoji od 2 bajta statusnog registra i viša dva bajta povratne adrese. Procesor bi nastavio izvoditi instrukcije od te pogrešne adrese.



4. zadatak: *(3 boda)* Pretpostavite da je zbrajalo za brojeve s pomičnim zarezom ostvareno kao protočna jedinica sa sljedećim protočnim segmentima: *Oduzimanje eksponenta* (vrijeme obrade: 40 ns), *Poravnavanje mantise* (vrijeme obrade: 70 ns), *Zbrajanje* (vrijem obrade: 50 ns) i *Normalizacija* (vrijeme obrade: 80 ns). Pretpostavite da takva protočna jedinica izvodi operaciju za N=10000 parova operanada.

a) Ocijenite efektivno vrijeme obrade za jedan par operanada
b) Ocijenite faktor ubrzanja $S_P=T/T_P$, gdje je T vrijeme obrade za neprotočnu strukturu (zbroj svih vremena sklopovlja za operaciju zbrajanja), a T_P vrijeme obrade za protočnu izvedbu jedinice. *Opaska:* U ovoj ocjeni pretpostavite da $N\to\infty$.

Rješenje:



Općenito, protočna se struktura sastoji od M protočnih segmenata (Npr. na slici, kao i u našem konkretnom slučaju koji je zadan u zadatku, vrijedi M=4). Ako na kratko pretpostavimo da je vrijeme obrade u svim protočnim segmentima jednako (i iznosi t_s), tada vidimo da će prvi rezultat izaći van nakon što prođe kroz sve protočne segmente, tj. nakon $M\cdot t_s$. Svaki sljedeći rezultat dolazit će nakon t_s vremena, jer kad je jedan gotov s obradom, slijedeći se već nalazi u zadnjem protočnom segmentu i treba mu još t_s vremena da bude gotov, itd. Ovo razmišljanje dovodi nas do formule pomoću koje računamo vrijeme za obradu N (parova) operanada u protočnoj strukturi:

$$\mathbf{T_P = M\cdot t_s + (N-1)\cdot t_s}$$

Međutim, što ako su vremena obrade u pojedinim protočnim segmentima različita? Do rješenja za taj slučaj doći ćemo vodeći računa o jednoj vrlo jednostavnoj činjenici. Naime, podatak koji je završio obradu u bržem protočnom segmentu ne može biti prihvaćen od sporijeg, dok ovaj (sporiji) ne završi sa svojom obradom. Zbog toga je protočna jedinica izvedena kao sinkrona struktura, u kojoj je prijenos podataka među protočnim segmentima sinkroniziran s najsporijim protočnim segmentom. To znači da će i dalje vrijediti ista formula, samo za t_s treba uvrstiti vrijeme obrade najsporijeg protočnog segmenta. U našem slučaju, to je segment koji izvodi normalizaciju, dakle $t_s = t_{NO} = 80$ ns. Zadatak se dalje rješava uvrštavanjem u formulu:

a) $$t_{ef} = \frac{M\cdot t_s + (N-1)\cdot t_s}{N} = \frac{(M+N-1)\cdot t_s}{N} = \frac{10003\cdot 80ns}{10000} = \frac{800240}{10000} = 80.024ns$$

b) $$S_P = \lim_{n\to\infty}\frac{T}{T_P} = \lim_{n\to\infty}\frac{(t_{OE}+t_{PM}+t_{ZB}+t_{NO})\cdot N}{M\cdot t_s + (N-1)\cdot t_s} = \lim_{n\to\infty}\frac{t_{OE}+t_{PM}+t_{ZB}+t_{NO}}{\frac{M}{N}\cdot t_s + \frac{(N-1)}{N}\cdot t_s}$$

Tu smo podijelili brojnik i nazivnik s N.

odnosno: $$S_P = \frac{t_{OE}+t_{PM}+t_{ZB}+t_{NO}}{t_s} = \frac{40ns+70ns+50ns+80ns}{80ns} = \frac{240ns}{80ns} = 3$$

5. zadatak: *(3 boda)* Modificirajte upravljačku jedinicu za računalo sa skupom od osam instrukcija tako da pridodate devetu instrukciju koja je tako složena da zahtijeva *12 ΔT* za cijeli instrukcijski ciklus i dodatne upravljačke signale c_i, gdje je $i>12$. Nacrtajte strukturu tako modificirane upravljačke jedinice i označite sve nužne preinake koje ta dodatna instrukcija zahtijeva, uključujući i onu kojom se izbjegava izvođenje nepotrebnih perioda. Pretpostavite da sve ostale instrukcije zahtijevaju *8 ΔT*.

Rješenje:

Tri su osnovne komponente od kojih se sastoji sklopovski izvedena upravljačka jedinica.
To su: - dekoder
- brojilo sekvenci po modulu k
- kombinacijsko sklopovlje (PLA)

Originalno, za 8-instrukcijski model procesora te su komponente bile sljedećih svojstava:

- Dekoder je bio 3/8, jer je ukupno 8 instrukcija, pa je dosta 3 bita za operacijski kod instrukcije. Operacijski kod instrukcije nalazi se u 3-bitnom instrukcijskom registru koji predstavlja ulaz u dekoder
- Brojilo sekvenci bilo je po modulu 8, jer najduža instrukcija traje 8 perioda signala vremenskog vođenja
- Kombinacijsko sklopovlje je na svom izlazu generiralo 13 različitih signala, označenih s $C_0 - C_{12}$, jer je upravo toliko bilo dovoljno za kontrolu svih upravljačkih točaka u računalu.

Dodavanjem nove, devete instrukcije, moramo uvesti sljedeće promjene:

- S obzirom da imamo više od 8 instrukcija, dekoder 3/8 nije više dovoljan, pa uzimamo prvi veći, a to je 4/16. Naravno, od njegovih 16 izlaza koristit ćemo samo prvih 9. Također, pošto se operacijski kod instrukcije više ne da prikazati s 3 bita, moramo proširiti i instrukcijski registar koji će sada postati 4-bitni.
- Pošto najduža instrukcija sada traje 12 perioda signala vremenskog vođenja, brojilo sekvenci biti će po modulu 12.
- U zadatku je zadano da novopridodana instrukcija zahtijeva i dodatne upravljačke signale C_i (gdje je $i>12$), moramo dodati i te signale kao izlaze iz PLA strukture.
- Dakako, samo kombinacijsko sklopovlje unutar PLA strukture se također mijenja kako bi podržalo novu instrukciju, ali pošto nije zadan opis same instrukcije, ne možemo konkretno reći kako će izgledati modificirano sklopovlje.

Sve navedene promjene naznačene su na slici lijevo dolje. Na slici desno dolje prikazana je izvedba sklopa za resetiranje brojila kako bi se izbjeglo izvođenje nepotrebnih perioda. Naime, po pretpostavci, sve ostale instrukcije osim novopridodane I_9, zahtijevaju 8 perioda signala vremenskog vođenja. Zbog toga, ako se izvodi neka od prvih 8 instrukcija, za vrijeme 9. periode resetiramo brojilo.