6. Upravljačka jedinica

Građa računala, Arhitektura i organizacija računarskih sustava
str. 123 – 158

- Funkcija upravljačke jedinice
- Prijenos upravljanja između programa
- Rekurzivni programi
- LIFO ili stožna struktura
- Uporaba stoga

Temeljne funkcije upravljačke jedinice:

- Uspostavljanje određenog stanja tijekom svakog instrukcijskog ciklusa
  - Pribavljanje instrukcija
  - Tumačenje instrukcija
  - Izvršavanje instrukcija
- Pohranjivanje informacije koja opisuje tekuće stanje u kojem se procesor nalazi
- Određivanje sljedećeg stanja na temelju trenutnog stanja, stanja zastavica u statusnom registru i stanja na ulaznim upravljačkim linijama procesora
- Generiranje upravljačkih signala tijekom instrukcijskog ciklusa
- Upravljanje slijedom instrukcija: izbor instrukcije – prijenos upravljanja s jedne na drugu instrukciju (engl. instruction sequencing)

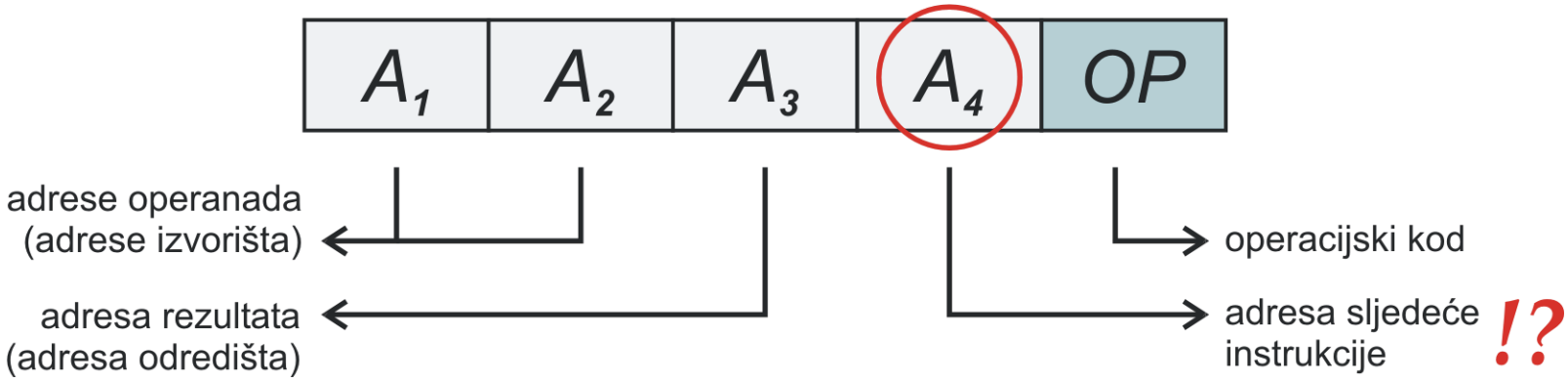
Najjednostavnija metoda upravljanja slijedom instrukcija:

- svaka instrukcija jednoznačno određuje adresu sljedeće:
  (značajka prvih računala – prije "von Neumannovog doba")

Primjer:

EDVAC (Electronic Discrete Variable Computer)

Oblik aritmetičke instrukcije:



Nedostatak: Povećana duljina instrukcije

(Dataflow arhitektura)

Druga metoda:

- Instrukcija I nalazi se na memorijskoj lokaciji s adresom A i ima jedinstvenu nasljednicu instrukciju I' na adresi A+1

PC – programsko brojilo sadrži adresu A + 1 (adresu slijedeće instrukcije)

Adresa slijedeće instrukcije (I') određuje se povećanjem PC-a:

$$PC \leftarrow PC + k,$$

gdje je k duljina instrukcije I izražena u riječima (bajtovima)

/za 32-bitni procesor RISC arhitekture sa bajtnom adresnom zrnatosti memorije k=4/

/ procesor CISC arhitekture - k promjenjiv npr. 2 do 22/

Prijenos upravljanja između instrukcija koje si nisu slijedne:

1) Instrukcije grananja unutar istog programa
2) Instrukcije za prijenos upravljanja između programa

1) Instrukcije grananja unutar istog programa

- Instrukcije bezuvjetnog grananja

  PC ← X, gdje je X adresa ciljne instrukcije

POZOR: Gornja se operacija izvodi tijekom faze IZVRŠI

- Instrukcije uvjetnog grananja

- Tijekom faze IZVRŠI ispituje se da li je neki uvjet *C* *zadovoljen* (*C* je obično posljedica neke ranije instrukcije)

- Ako je *C* zadovoljen, tada PC ← X, u drugim slučajevima PC se ne mijenja (tijekom faze IZVRŠI)

2) Instrukcije za prijenos upravljanja između programa

Suvremeni stil programiranja u prvi plan stavlja uporabu programskih procedura – temelj strukturiranih programa

P1 – program (glavni program)
P2 – potprogram

ili

P1 – pozivajući program
P2 – pozvani program

Dva glavna slučaja:
1) Pozivanje potprograma
2) Prekidi (engl. interrupt) ili iznimke (engl. exception)

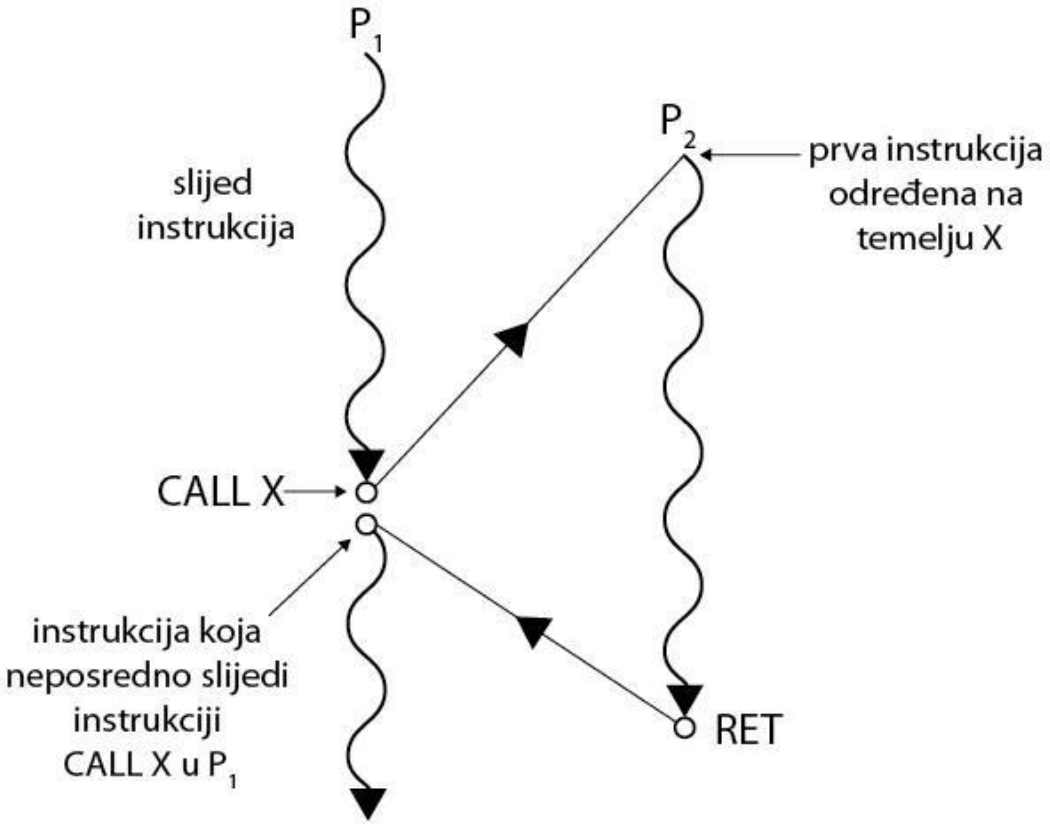
1) P1 ⟶ P2 instrukcije za pozivanje potprograma

(engl. Call, jump to subroutine):

CALL X,

gdje je X ciljna adresa (ili se ciljna adresa računa na temelju X) prve instrukcije (pot)programa P2.

Odnos između pozivajućeg i pozvanog programa



Tijekom faze IZVRŠI instrukcije CALL obavljaju
se dva slijedeće koraka:

1. Korak: Sadržaj PC-a (koji pokazuje na slijedeću instrukciju u P1) se pohranjuje na za to predodređenu lokaciju S,

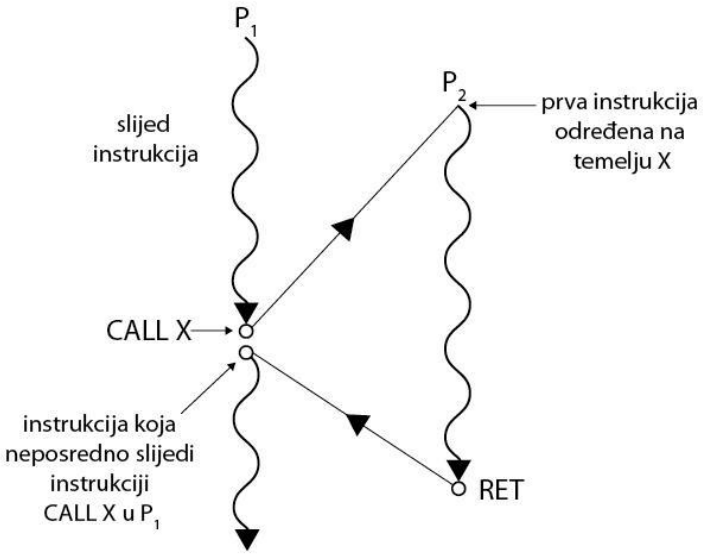
2. Korak: X (ili adresa izračunata na temelju X) prenosi se u PC

Lokacija S sadrži povratnu adresu

Povratak iz programa P2 (P2→ P1):

Instrukcija povratka (Return; RET) – posljednja instrukcija u P2

PC ← S



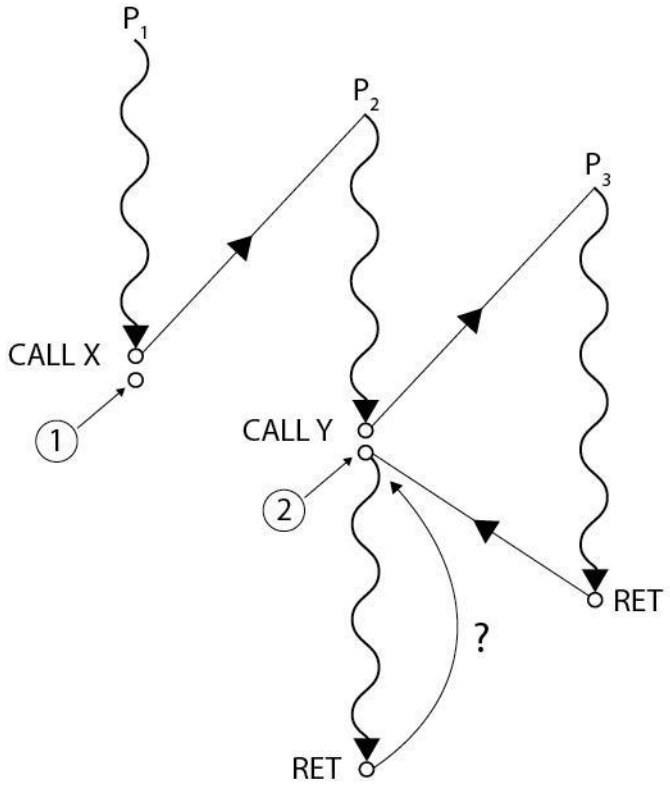
• Gdje locirati **S?**

• **Rješenje 1**: **S** je poseban registar u upravljačkoj jedinici
• problem: gniježdenje potprograma!

• **Rješenje 2**: **S** je prva lokacija potprograma
• instrukcija jms X (Jump to Subroutine, PDP-8):
• PC se pohranjuje na lokaciju X (!) i automatski inkrementira
• Povratak ostvarujemo indirektnim skokom JMP I X
• problem: rekurzija!

• **Rješenje 3**: **S** se nalazi u posebnom dijelu radne memorije
• taj dio radne memorije nazivamo upravljačkim stogom
• Moderna računala kombiniraju rješenja 1. i 3.!

Problem gniježđenja poziva u slučaju instrukcije CALL X
kada se ona izvodi u sljedeća dva koraka: Registar ← PC, PC ← X



Primjer: PDP –8
(ili kako se nekad radilo)

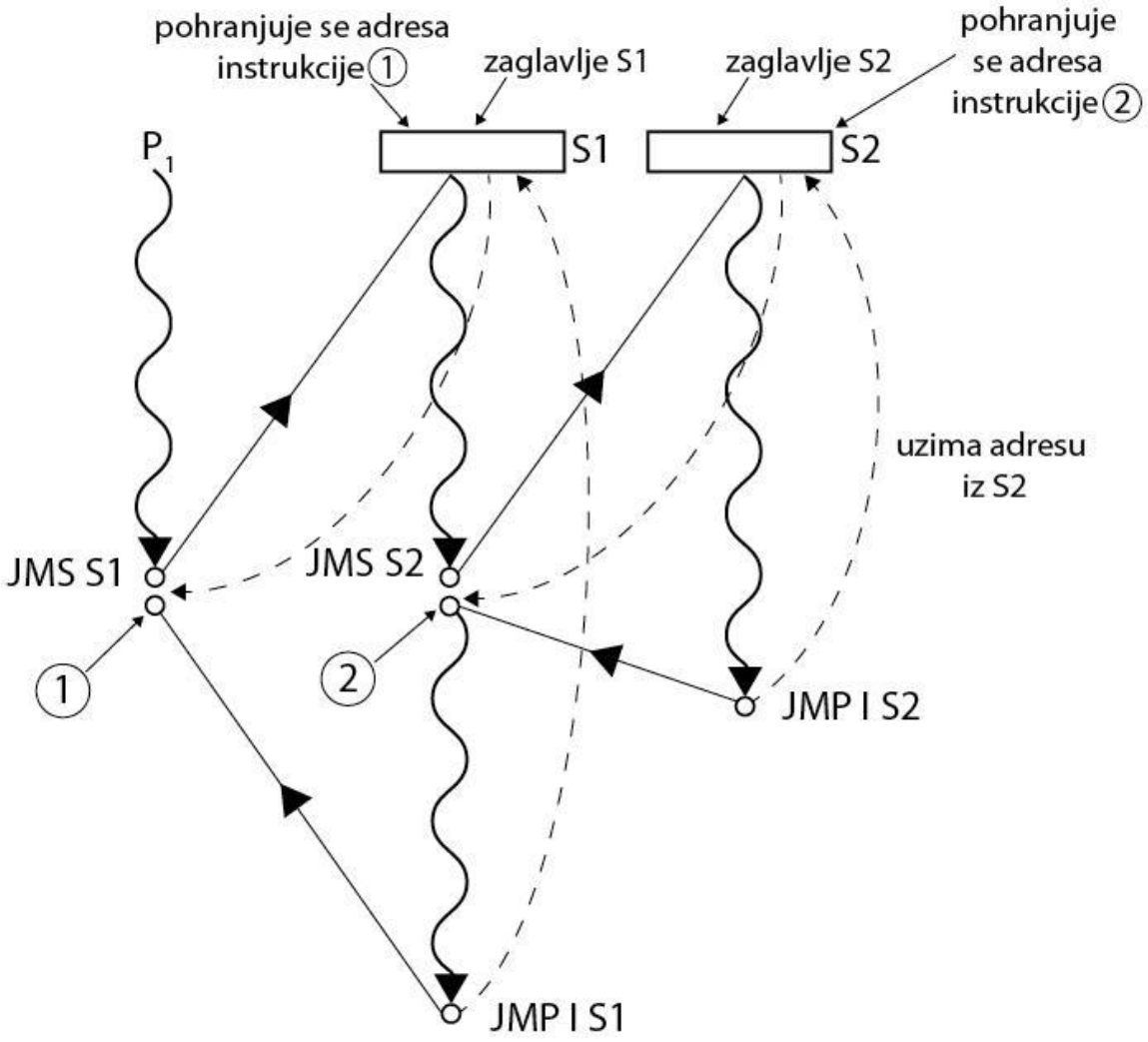
JMS SUB ; Jump to Subroutine

i) PC se pohranjuje na lokaciju SUB
ii) PC se automatski inkrementira podrazumijevajući da se prva instrukcija potprograma SUB nalazi na adresi SUB + 1

Prijenos upravljanja s potprograma na pozivajući program:

JMP I SUB ; Indirektni skok na SUB !!!

Problem gniježđenja potprograma riješen!
Ostaje problem rekurzije!


pohranjuje se adresa instrukcije 1
zaglavlje S1
zaglavlje S2
pohranjuje se adresa instrukcije 2
P1
S1
S2
uzima adresu iz S2
JMS S1
JMS S2
1
2
JMP I S2
JMP I S1

Drugo rješenje problema gniježđenja za SRISC model procesora

instrukcija *brl* (Branch and link):

brl ra, rb ; *R*[ra] ← PC, PC ← R[rb]

/instrukcija *brl ra, rb* pohranjuje povratnu adresu u *R*[ra], a zatim puni PC s 32-bitnim sadržajem registra R[rb]/

Povratak se obavlja instrukcijom bezuvjetnog grananja:

br ra ; *PC* ← *R*[ra]

Primjer:
Pretpostavimo da registar R[28] sadržava adresu potprograma P2 , registar R[29] potprograma P3 i registar R[23] adresu potprograma P4

- iz glavnog programa P1 pozvat će se potprogram P2, instrukcijom:

brl r2, r28 ; R[2] ← PC, PC ← R[28];

/povratna adresa u *R*[2] /

- potprogram P2, poziva potprogram P3 izvođenjem instrukcije:

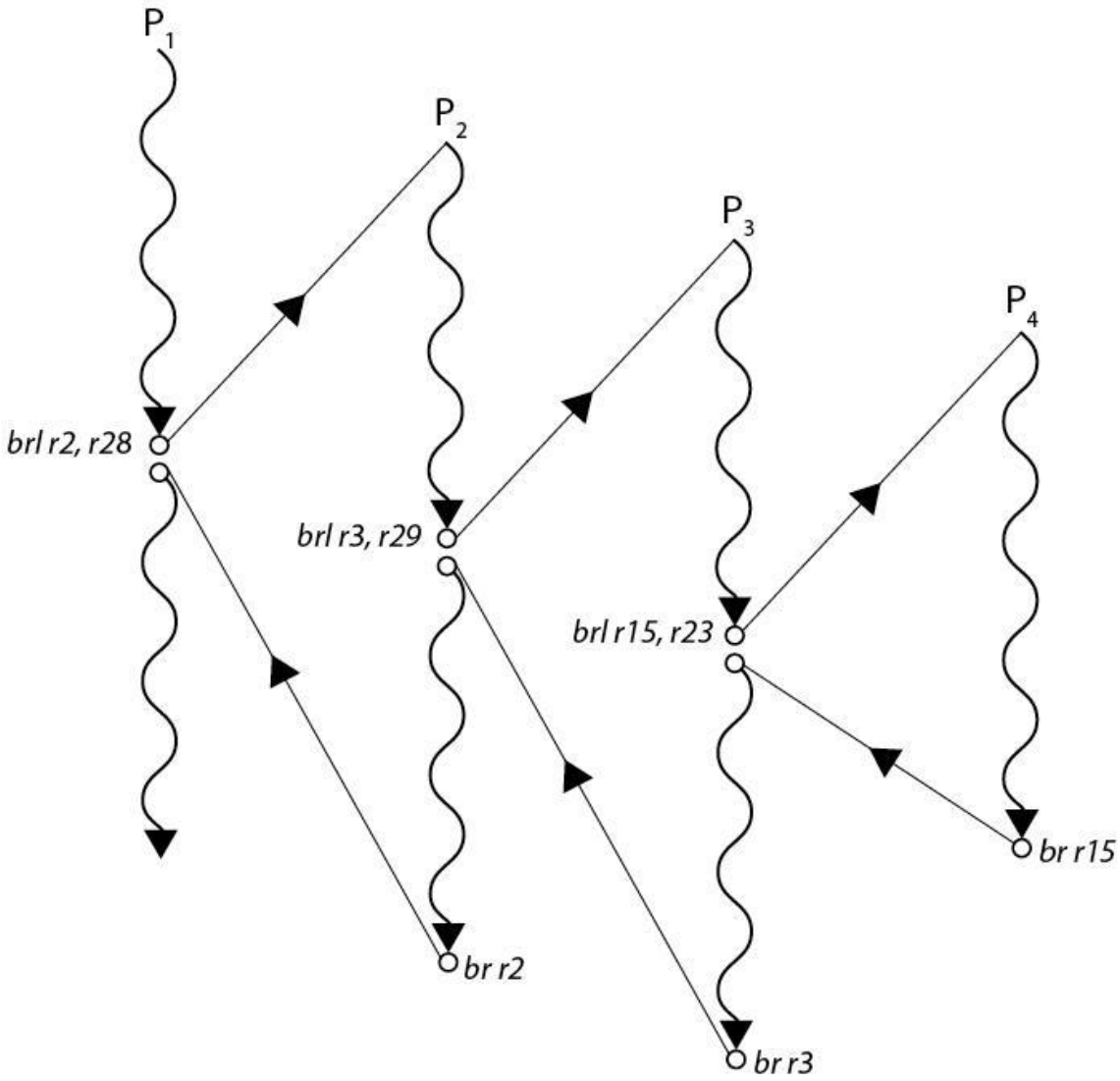
brl r3, r29

/povratna adesa u R[3]/

- iz potprograma P3 poziva se potprogram P4:

brl r15, r23

/povratna adresa u R[15]/



Problem gniježđenja potprograma riješen!
Ostaje problem rekurzije!

Rekurzija - primjer:

$n! = 1 \times 2 \times 3 \times 4 \times ... \; n$

$1! = 1$

$n! = n \times (n-1)!$

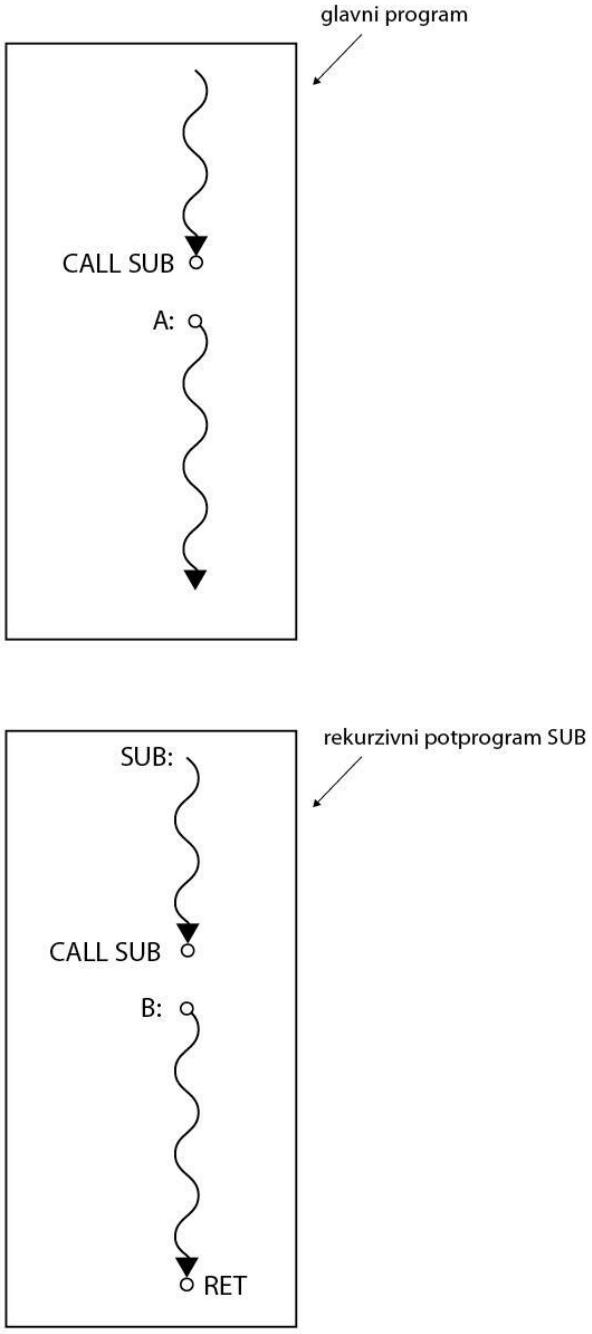
Hanojski tornjevi

Fibonaccijevi brojevi:

1) $F_0 = 0, F_1 = 1$; i
2) $F_n = F_{n-1} + F_{n-2}$, za $n \geq 2$

Prikaz rekurzivnog pozivanja:

Glavni program u jednom trenutku instrukcijom CALL X poziva potpogram SUB. Povratna adresa u glavni program je A. Prenosi se upravljanje na potprogram SUB. Izvode se instrukcije u potprogramu SUB i u jednom se trenutku izvodi instrukcija CALL SUB u pozvanom potprogramu – potprogram SUB poziva samog sebe! Predviđena povratna adresa je B.



Rješenje problema rekurzije: Uporaba upravljačkih stogova

Stog – dinamička podatkovna struktura, pristup na principu LIFO (Last In First Out)

služi za pohranu povratnih adresa
(i parametara potprograma, lokalnih varijabli, …)

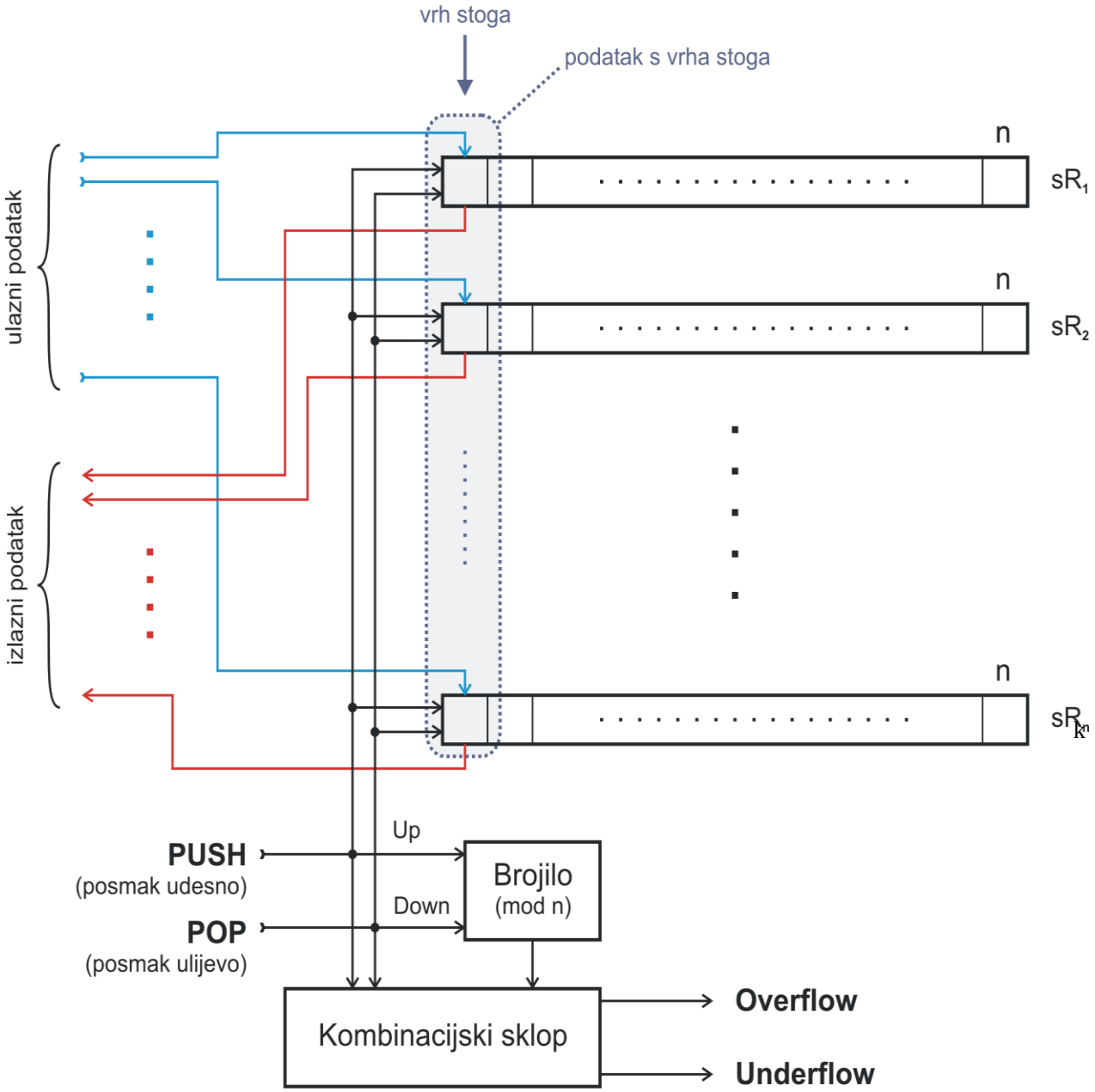
Operacije: PUSH (stavi podatak na stog), POP (skini podatak sa stoga)

CALL SUB	RETURN
1) PUSH PC	1) POP PC
2) PC ← SUB	

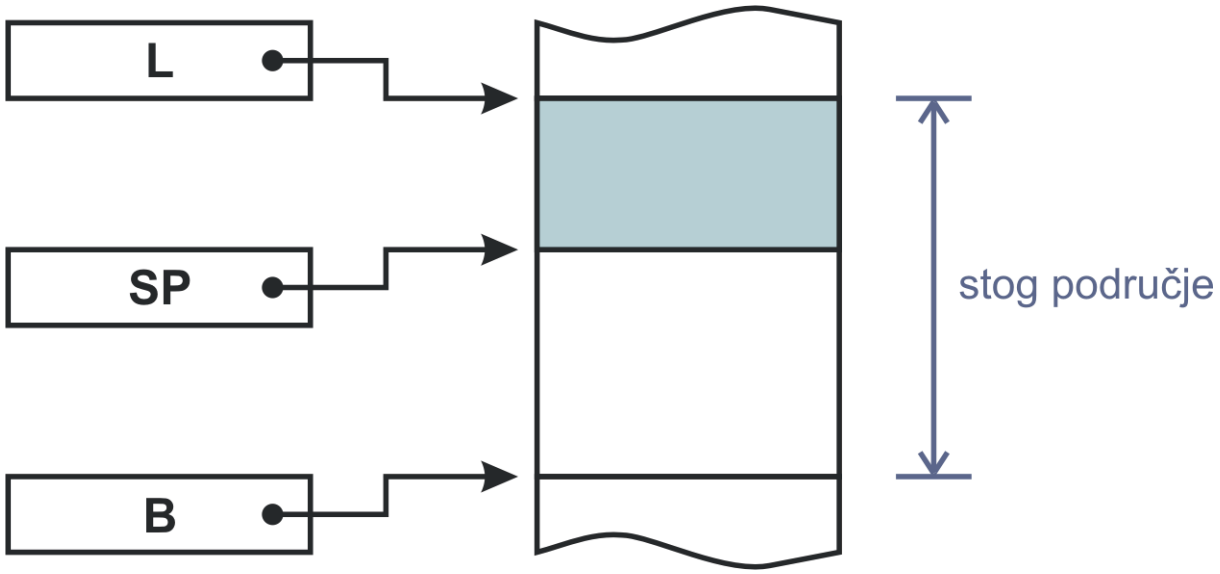
Izvedba stoga:

1) Sklopovski
   - Posmačni registri
   - Koristi se u nekim mikrokontrolerima
2) Područje u memoriji (češće)
   - Posebni registri:
     - SP (engl. Stack pointer) – kazalo stoga; sadrži adresu vrha stoga
     - Gornja i donja granica memorijskog područja za stog

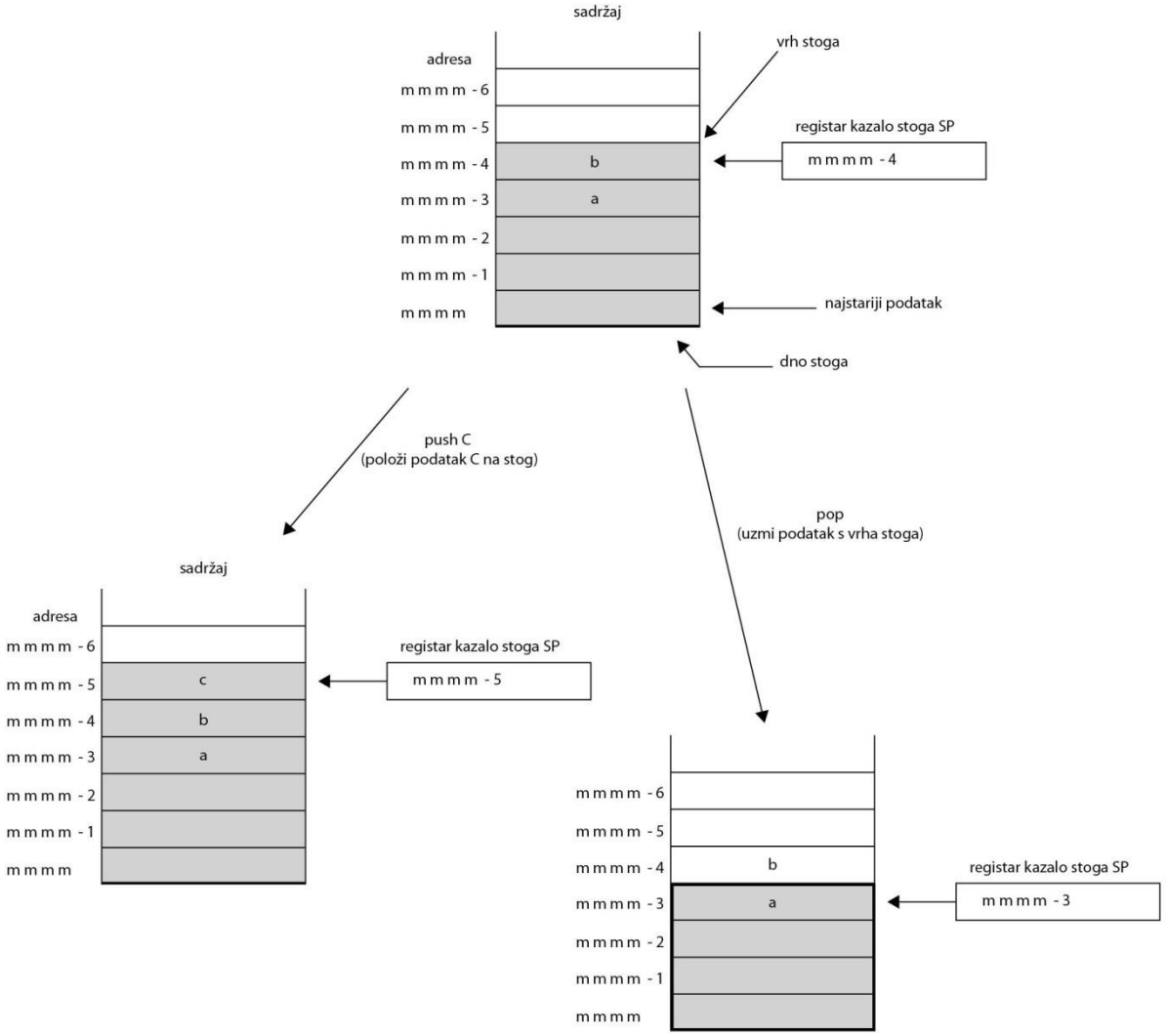
1)


vrh stoga
podatak s vrha stoga
ulazni podatak
izlazni podatak
n
sR1
n
sR2
n
sRkn
PUSH
(posmak udesno)
POP
(posmak ulijevo)
Up
Down
Brojilo
(mod n)
Kombinacijski sklop
Overflow
Underflow

2)



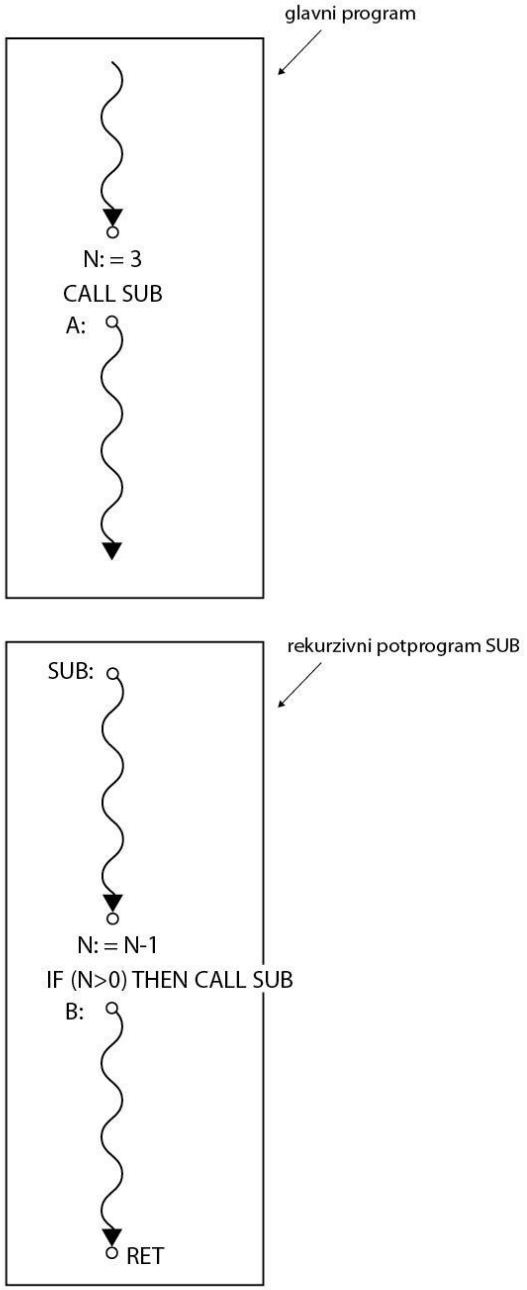
Stanje stoga i sadržaja kazala stoga prije i poslije izvođenja operacija *push* i *pop*



Rekurzivno pozivanje:

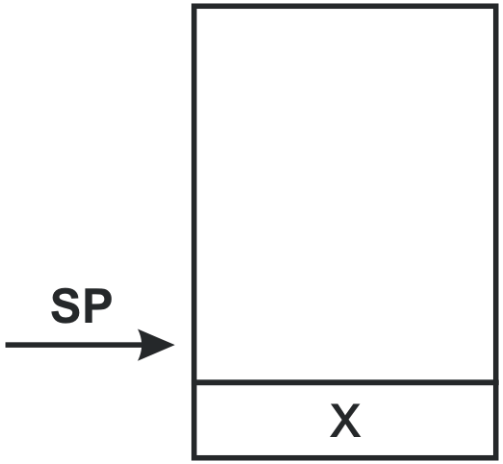
```
Begin
        .
    N  := 3
    CALL SUB
X:      .
        .
SUB:    .
        .
        N := N – 1
        IF (N > 0) THEN CALL SUB
Y:      .
        .
        RETURN
End
```

Kontrolirano samopozivanje potprograma:

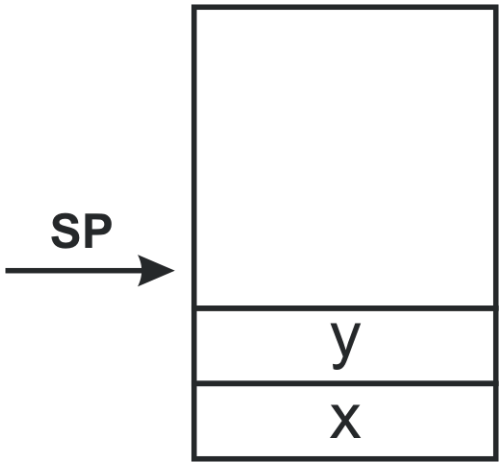


Stanje stoga:

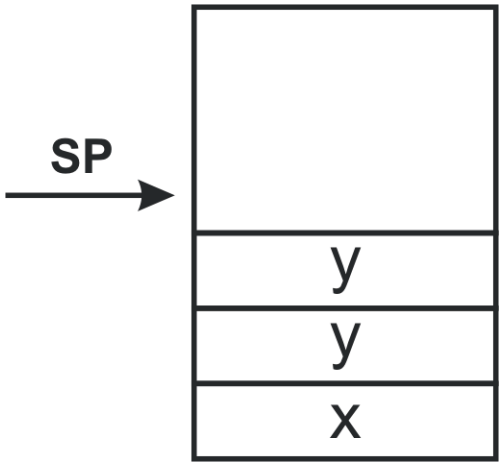
- Prvi poziv (N = 3) / poziv iz glavnog programa/:



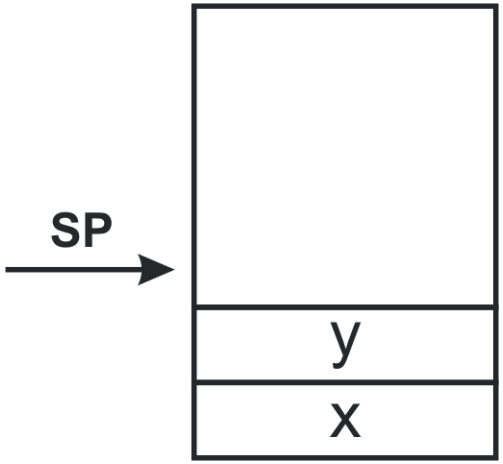
- Drugi poziv (N = 2) / poziv iz SUB/:



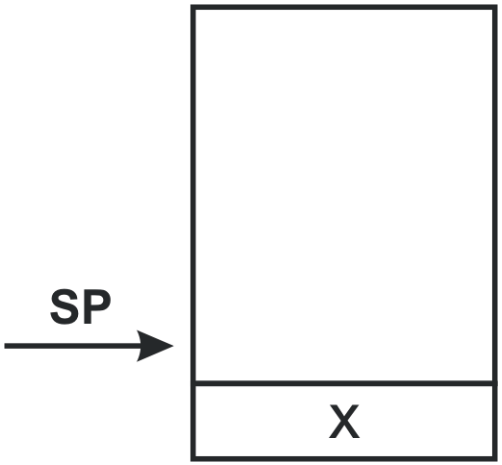
Treći poziv (N = 1) /poziv iz SUB/:



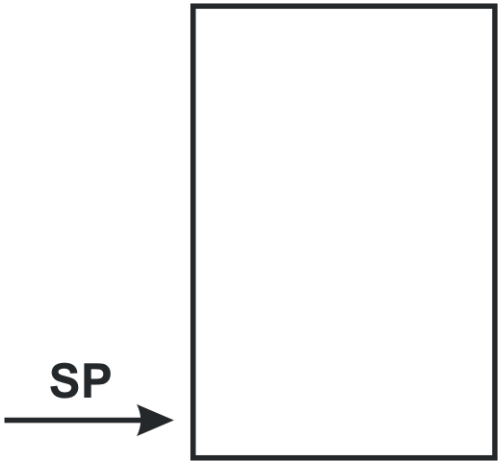
Prvi povratak:



Drugi povratak:



Treći povratak:



Procesor MC 68000 nema instrukcije *push* i *pop*

Instrukcija

MOVE.L D0, -(A7)

pohranjuje 32-bitni sadržaj registra D0 na vrh stoga

Instrukcija

MOVE.L (A7)+, D0

s vrha stoga uzima 32-bitni sadržaj i premješta ga u registar D0

Također, za CALL i RET koristi drukčije mnemonike
(no funkcija im je jednaka kao CALL i RET):

- JSR (Jump to Subroutune)
- RTS (Return from Subroutine)

Načini prijenosa parametara između između pozivajućeg i pozvanog programa:

i) prijenos parametara preko registara (engl. *parameter passing via registers*)
ii) prijenos parametara preko memorijskih lokacija (engl. *parameter passing in memory*)
iii) prijenos parametara preko stoga (engl. *passing parameters on stack*)

Primjer – prijenos parametara preko registara

Pretpostavimo da treba prenijeti potprogramu SUBR dva parametra: duljinu spremnika podataka BUFL i njegovu početnu adresu BUFFER.
Programski odsječak pozivajućeg (glavnog) programa:

```
MOVE.W      #BUFL, D0
MOVEA.L     #BUFFER, A0
JSR         SUBR
```

Nedostatak rješenja: Potprogrami ne mogu pozivati sami sebe
(nije omogućena rekurzija)

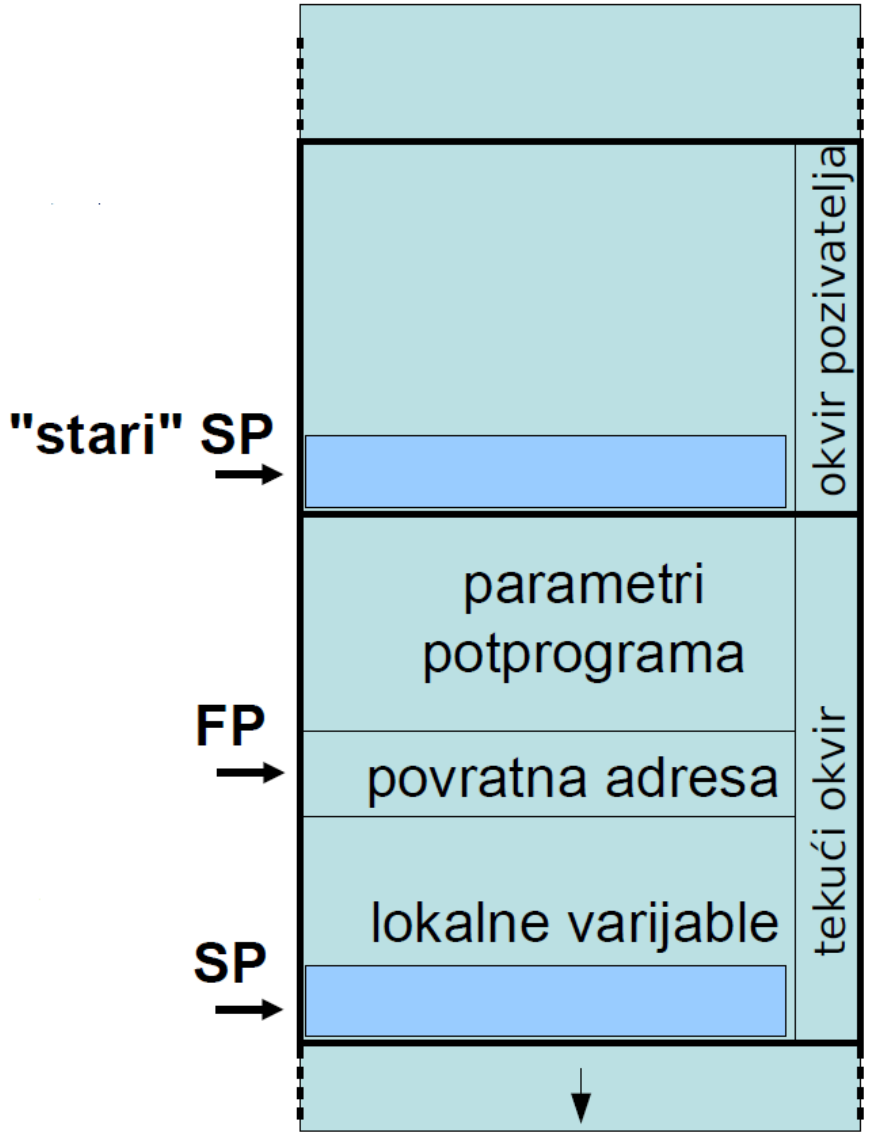
Korištenje stoga u višim programskim jezicima

Korisnički stog pohranjuje sve informacije o pozivu potprograma

Dio stoga koji odgovara jednom pozivu nazivamo **okvirom** (engl. stack frame)

Stogovni okvir tipično sadrži:
- povratne adrese
- parametri potprograma
- lokalne varijable

Tipična struktura stogovnog okvira prikazana je na slici desno:



Primjena stoga pri obradi procesorskih iznimki

Procesorskim iznimkama (engl. processor exception) nazivamo
- posebne okolnosti u kojima je normalno stanje procesora narušeno
- procesor obrađuje iznimke prekidom normalnog izvođenja
- prijenos upravljanja se odvija bez upotrebe posebnih instrukcija
- nakon obrade iznimke prekinuti proces se nastavlja (ako je moguće)
- iznimke se mogu gnijezditi (prioritet!)
- pazi: iznimke u programskim jezicima su nešto sasvim drugo!

Dvije osnovne grupe iznimki:
1. vanjske iznimke
- bez mogućnosti oporavka: sabirnička pogreška, reset
- s oporavkom: sklopovski prekidi
2. unutarnje iznimke
- pozivi jezgre OS-a (engl. trap, software interrupt)
- programske greške (dijeljenje nulom, povreda privilegiranosti,...)

Primjer uporabe stoga

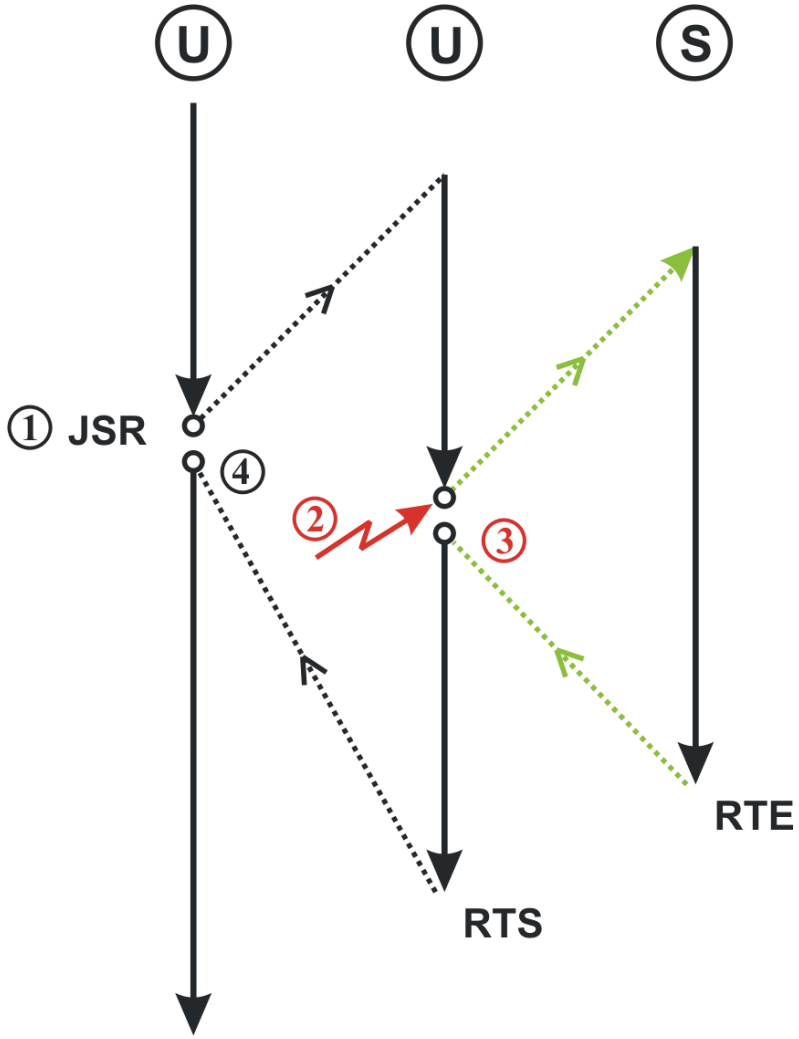
Analiza slučaja: MC 68000

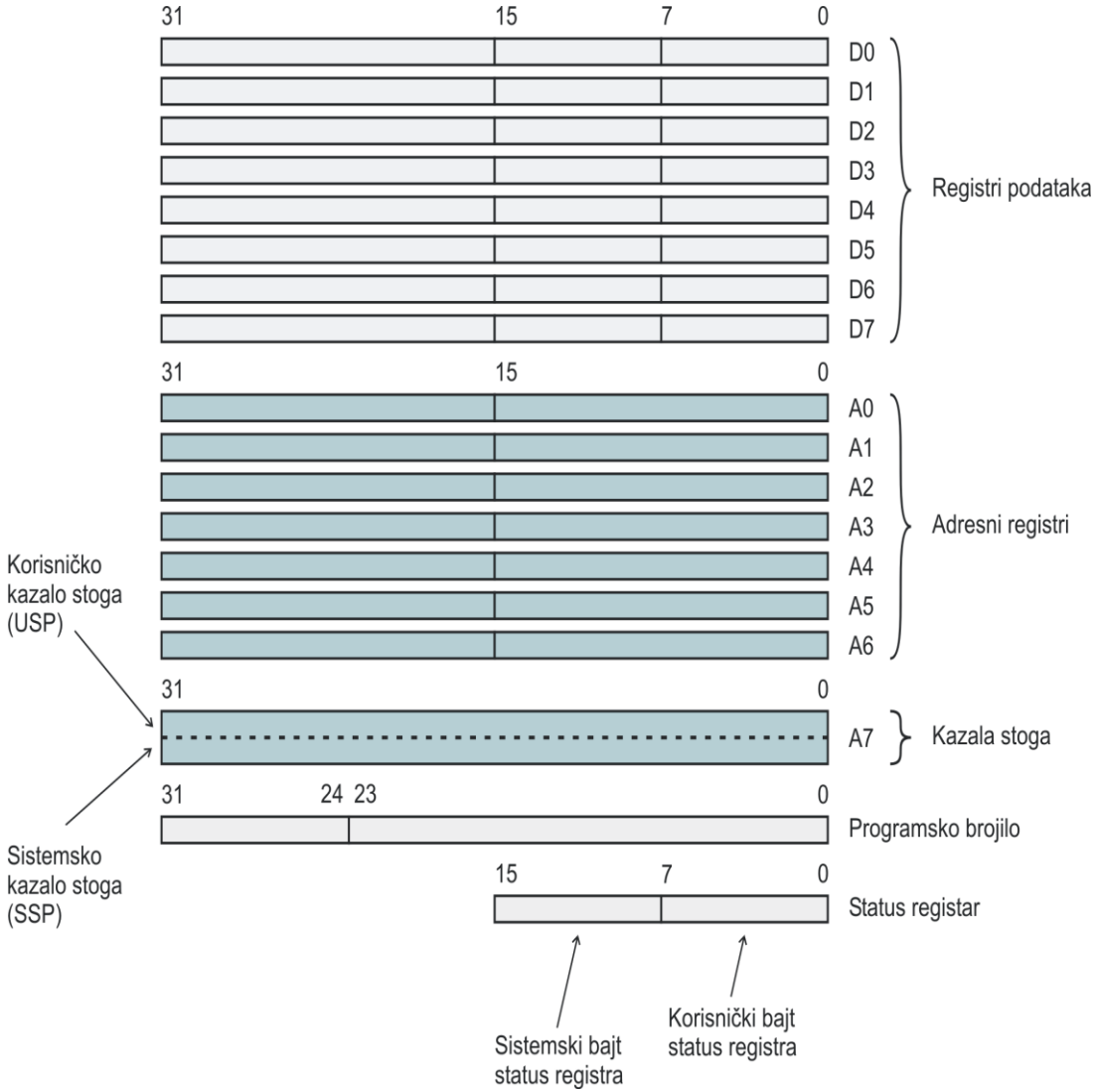
Scenarij:

1. Procesor je u korisničkom načinu rada (User Mode)
   - Poziva se potprogram
   - Nastavlja se izvođenje potprograma
2. Dogodila se iznimka (PREKID)
   - Obrada prekida
3. Vraćanje u potprogram
4. Vraćanje iz potprograma

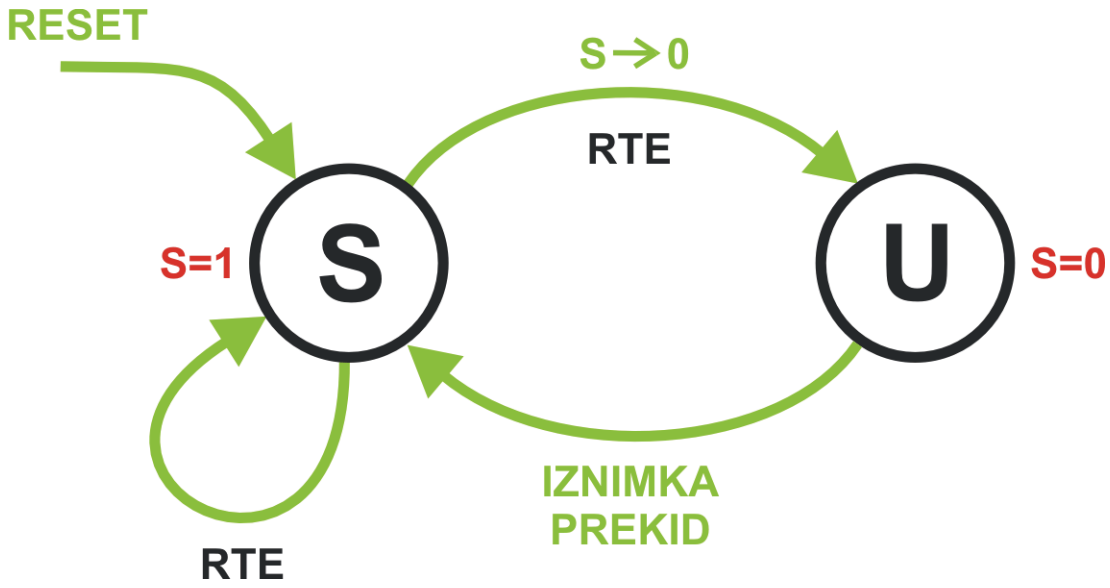
/primjer detaljno opisan u S. Ribarić, Građa računala,
Algebra, Zagreb, 2011. pp.147 - 158/

Grafički prikaz scenarija:



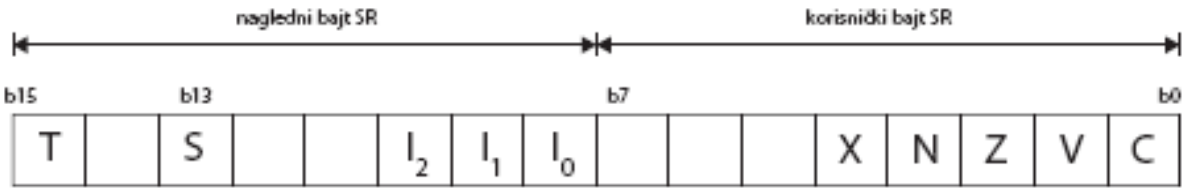

31
15
7
0
D0
D1
D2
D3
D4
D5
D6
D7
Registri podataka
31
15
0
A0
A1
A2
A3
A4
A5
A6
Adresni registri
Korisničko kazalo stoga (USP)
31
0
A7
Kazala stoga
31
24 23
0
Programsko brojilo
Sistemsko kazalo stoga (SSP)
15
7
0
Status registar
Sistemski bajt status registra
Korisnički bajt status registra

Dijagram stanja za MC 68000



RTE -Return from Exception /povlaštena instrukcija/

Statusni registar



Zastavuce: C (*Carry*) - zastavica prijenosa
V (*Overflow*) - zastavica preljeva (aritmetičkog)
Z (*Zero*) - zastavica nule
N (*Negative*) - zastavica negativna vrijednost
X (*Extend*) - zastavica proširenja
$I_2 I_1 I_0$ (*InterruptMask*) - prekidne zastavice
S (*Supervisor*) - nadgledna zastavica
T (*TraceMode*) - zastavica praćenja

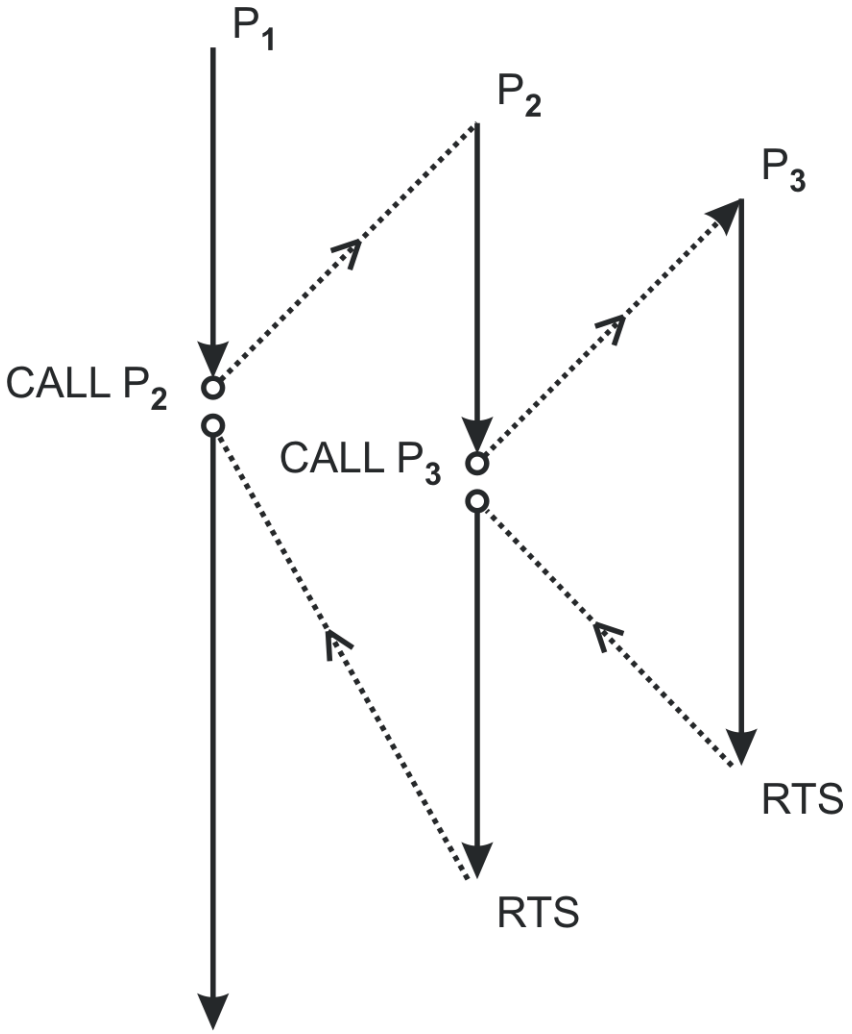
- CALL POT se izvodi u dva koraka:

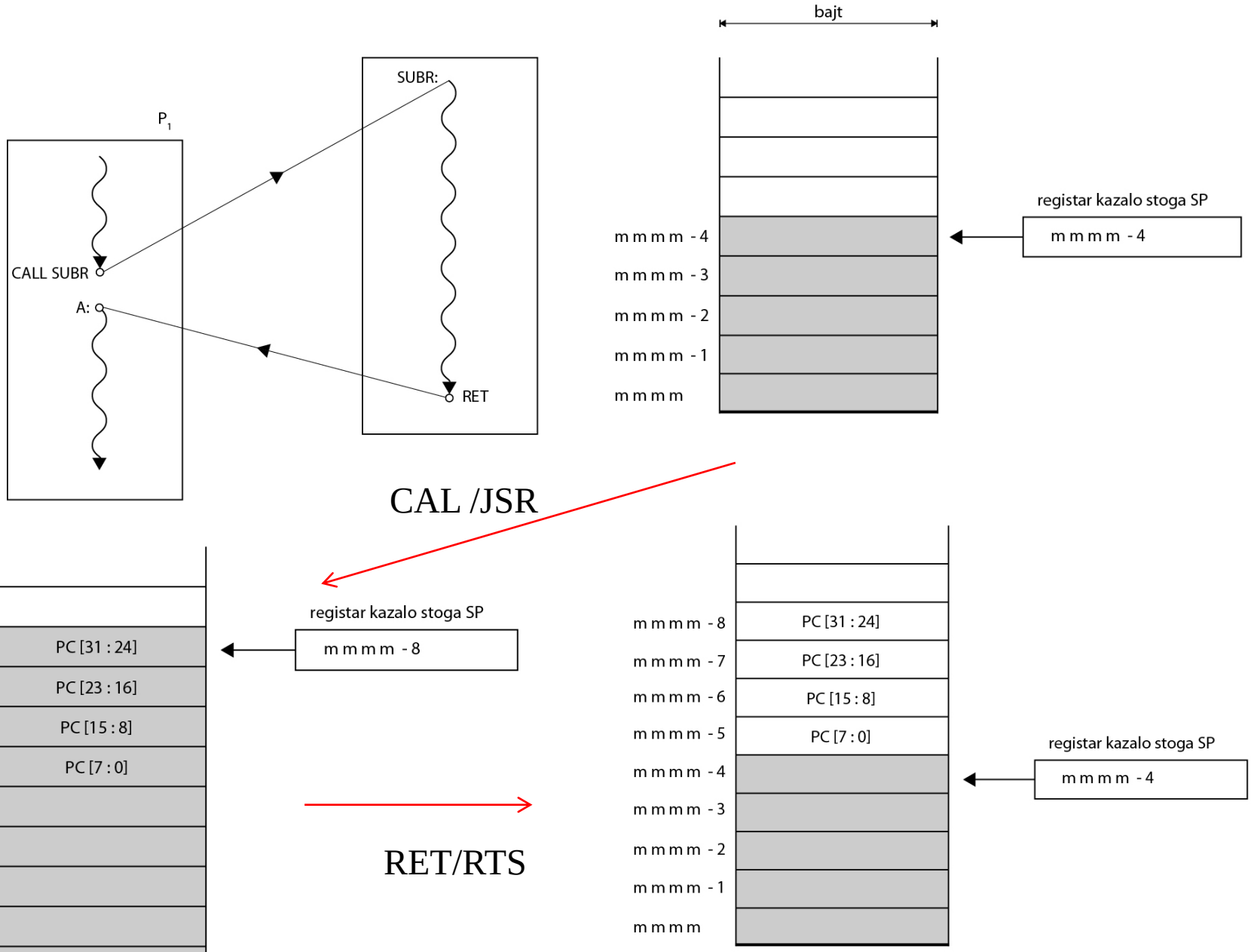
- PUSH PC

- PC ← POT

- Povratak (RET):
  - POP PC

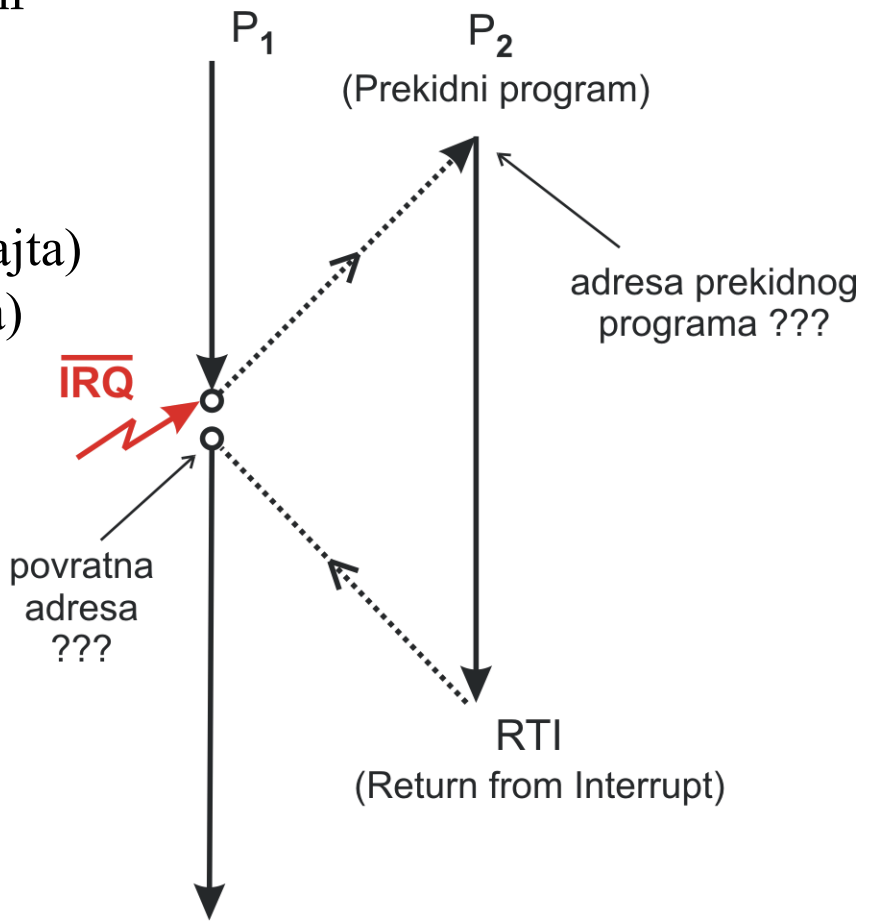
Uporaba stoga!!!




bajt
SUBR:
P1
CALL SUBR
A:
RET
registar kazalo stoga SP
m m m m - 4
m m m m - 3
m m m m - 2
m m m m - 1
m m m m
CAL /JSR
m m m m - 9
m m m m - 8
m m m m - 7
m m m m - 6
m m m m - 5
m m m m - 4
m m m m - 3
m m m m - 2
m m m m - 1
m m m m
PC [31 : 24]
PC [23 : 16]
PC [15 : 8]
PC [7 : 0]
registar kazalo stoga SP
m m m m - 8
RET/RTS
registar kazalo stoga SP
m m m m - 4

Na stog se tijekom prijenosa upravljanja s prekinutog na prekidni program ($P_1 \longrightarrow P_2$) pohranjuje MINIMALNI KONTEKST:

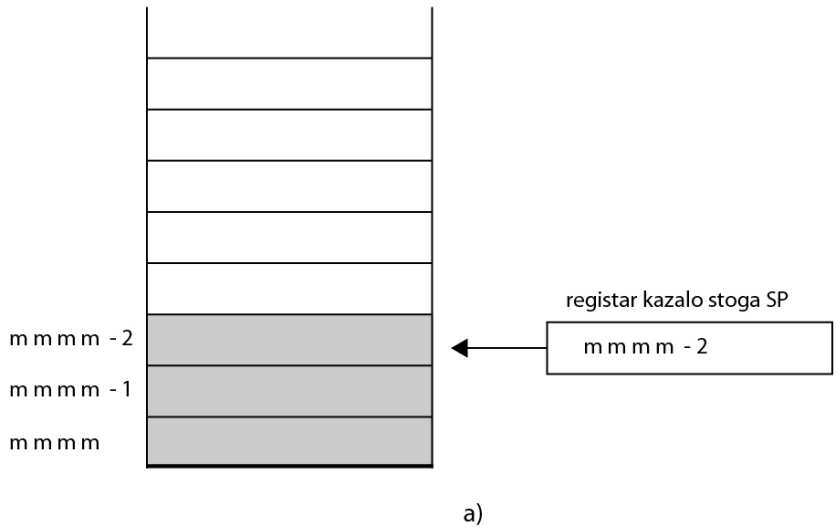
- Sadržaj programskog brojila (4 bajta)
- Sadržaj statusnog registra (2 bajta)



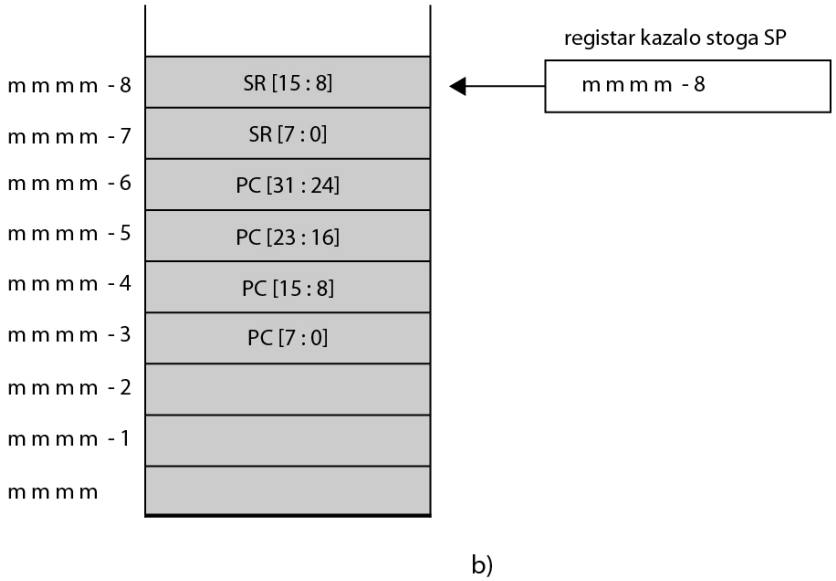
Adresa prekidnog programa?

Detaljan opis: Građa računala - str. 417 – 425

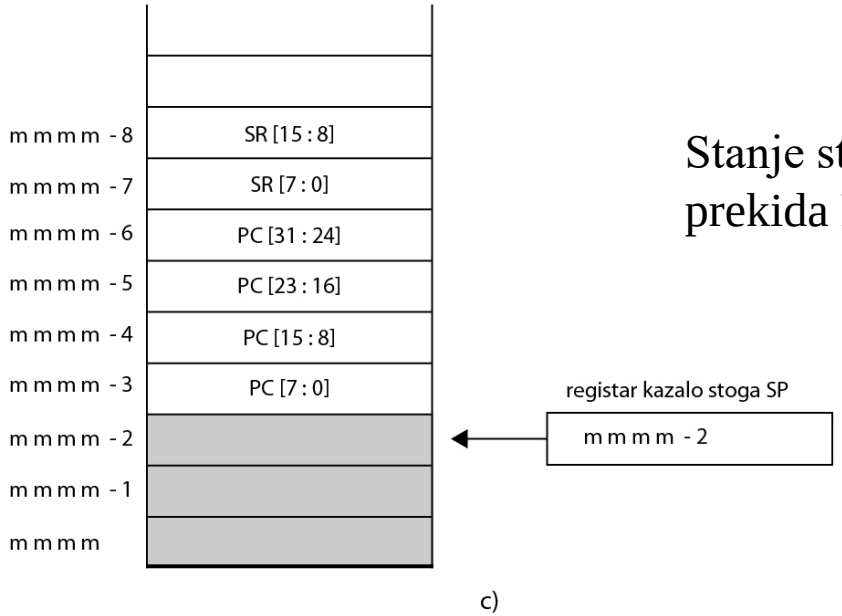
Pohrana minimalnog konteksta na stog



a)



b)



c)

Stanje stoga nakon izvođenja instrukcije povratka iz prekida RTI, odnosno RTE (za MC 68000)

Kako se generira zahtjev za prekid?

Zahtjev za prekid 

generira vanjski U/I uređaj
postavljanjem linija za zahtjevanje prekida IPL0*, IPL1* i IPL2*
u aktivno stanje. (Opaska:* označava aktivno stanje signala logičko 0)

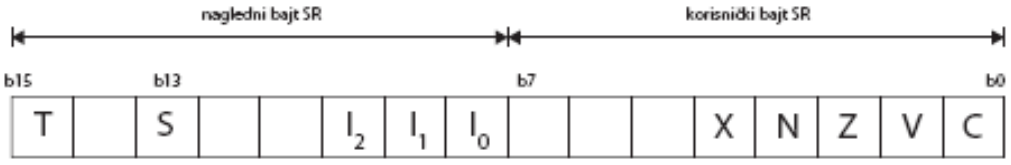
- aktivno stanje IPL0* = 0 ima "težinu" $2^0 = 1$
- aktivno stanje IPL1* = 0 ima "težinu" $2^1 = 2$
- aktivno stanje IPL2* = 0 ima "težinu" $2^2 = 4$

MC 68000 ima hijerarhijski ustroj prekida – 7 razina!

IPL0* = 1, IPL1* = 1, IPL2*= 1 – nema zahtjeva za prekid
IPL0* = 0, IPL1* = 1, IPL2*= 1 – razina prekida $1x2^0 + 0x2^1+0x2^2 = 1$ (prva razina)
…..
IPL0* = 0, IPL1* = 0 , IPL2*= 0 – razina prekida 7!

Kad će zahtjev za prekid biti prihvaćen?

Statusni registar



Zastavuce: C (*Carry*) - zastavica prijenosa
V (*Overflow*) - zastavica preljeva (aritmetičkog)
Z (*Zero*) - zastavica nule
N (*Negative*) - zastavica negativna vrijednost
X (*Extend*) - zastavica proširenja
$I_2 I_1 I_0$ (*Interrupt Mask*) - prekidne zastavice
S (*Supervisor*) - nadgledna zastavica
T (*Trace Mode*) - zastavica praćenja

Prekidne zastavice I_0, I_1, I_2 određuju razinu prekida koja će se prihvatiti:
I_0 ima "težinu" 2^0
I_1 ima "težinu" 2^1
I_2 ima "težinu" 2^2

Zahtjev za prekid se prihvaća ako je "težina" zahtjeva kodirana stanjem linija IPL0*, IPL1* i IPL2* **VEĆA** od "težine" kodirane stanjem prekidnih zastavica
Npr., ako je
$I_0 = 1$, $I_1 = 0$, $I_2 = 1$ – "težina" je $1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = 5$ a
IPL0* =1, IPL1* = 0 i IPL2* = 0 – "težina" je $0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 = 6$
Zahtjev za prekid se prihvaća!

Samo u slučaju kada je IPL0* = 0, IPL1* = 0 i IPL2* = 0 (najviša razina zahtjeva za prekid!) on se prihvaća bez obzira na stanje zastavica I_0, I_1, I_2.
/nemaskirajući zahtjev za prekid – procesor se ne može zaštititi od tog prekida/

Npr., $I_0 = 1$, $I_1 = 1$ i $I_2 = 1$ – razina zaštite je 7 a ako je zahtjev za prekid IPL0* = 0, IPL1* = 0 i IPL2* = 0 (razina zahtjeva 7)
zahtjev za prekid bit će prihvaćen!

Kako se određuje adresa prve instrukcije prekidnog programa?

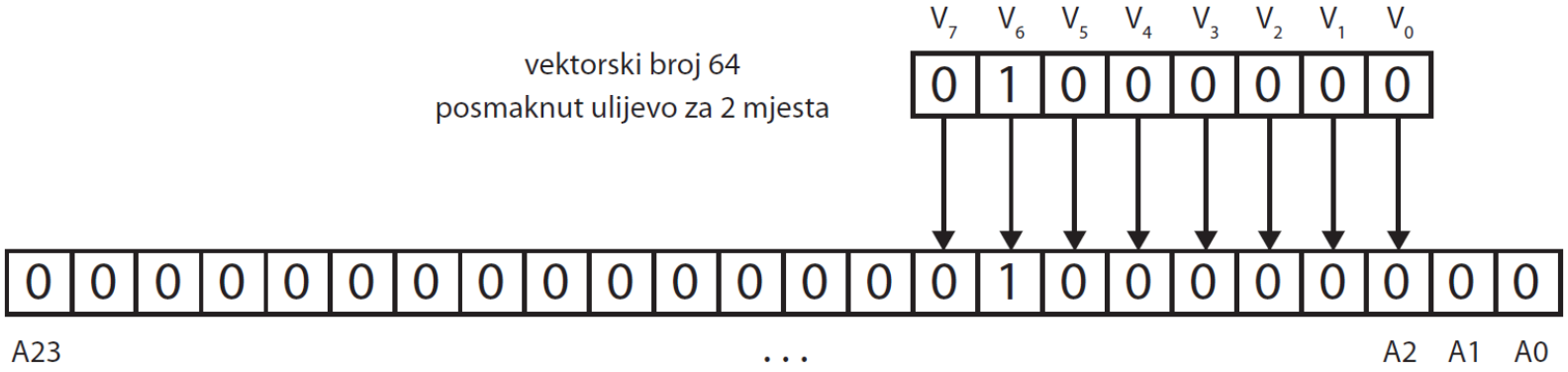
Ako je procesor prihvatio zahtjev za prekid, započine poseban sabirnički ciklus potvrde prekida te na adresne linije A_1, A_2 i A_3 postavlja razinu prihvaćenog prekida

Procesor obavještava vanjske uređaje da je prekid prihvaćen tako da svoje izlazne linije FC0, FC1 i FC2 sve postavlja u "1"
Logičko I FC0, FC1 i FC2 predstavlja signal potvrde prihvaćanja prekida!

Ulazno/izlazni uređaj koji je uzročnik prekida prima informaciju o prihvaćanju prekida i informaciju o razini prihvaćenog prekida te odgovara procesoru tako da na linije sabirnice podataka D0 – D7 postavlja **vektorski broj (osambitni kod) i aktivira signal DTACK***
Vektorski broj ima ulogu OIB-a! – jednoznačno identificira uzročnika prekida

Procesor prihvaća vektorski broj i na temelju njega oblikuje adresu memorijske lokacije na kojoj se nalazi adresa prve instrukcije prekidnog programa!

Vektorski broj se interno u procesoru množi sa četiri i oblikuje se 24-bitna adresa koja pokazuje na memorijsku lokaciju (u nultoj stranici) na kojoj se nalazi adresa prve instrukcije prekidnog programa – vektor iznimke/prekida:

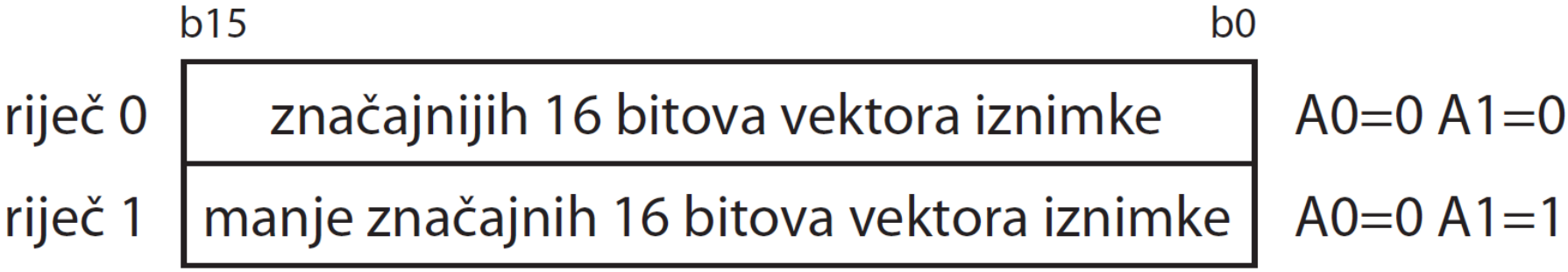


Vektorski brojevi (korisnički prekidni vektori) od 64 do 255 (dekadno) rezervirani su za prekide

0-191 identifikatora prekida (ukupno 192 uzročnika prekida!)

Vektori iznimke/korisničkog prekida nalaze se na adresama u tzv. nultoj stranici memorije (lokacije s adresama od 0- 1023) i to adrese
000100 (heksadekadno) za vektorski broj 64
do
0003FF (heksadekadno) za vektorski broj 255

Format adrese prve instrukcije prekidnog programa (pohranjen u nultoj stranici) je

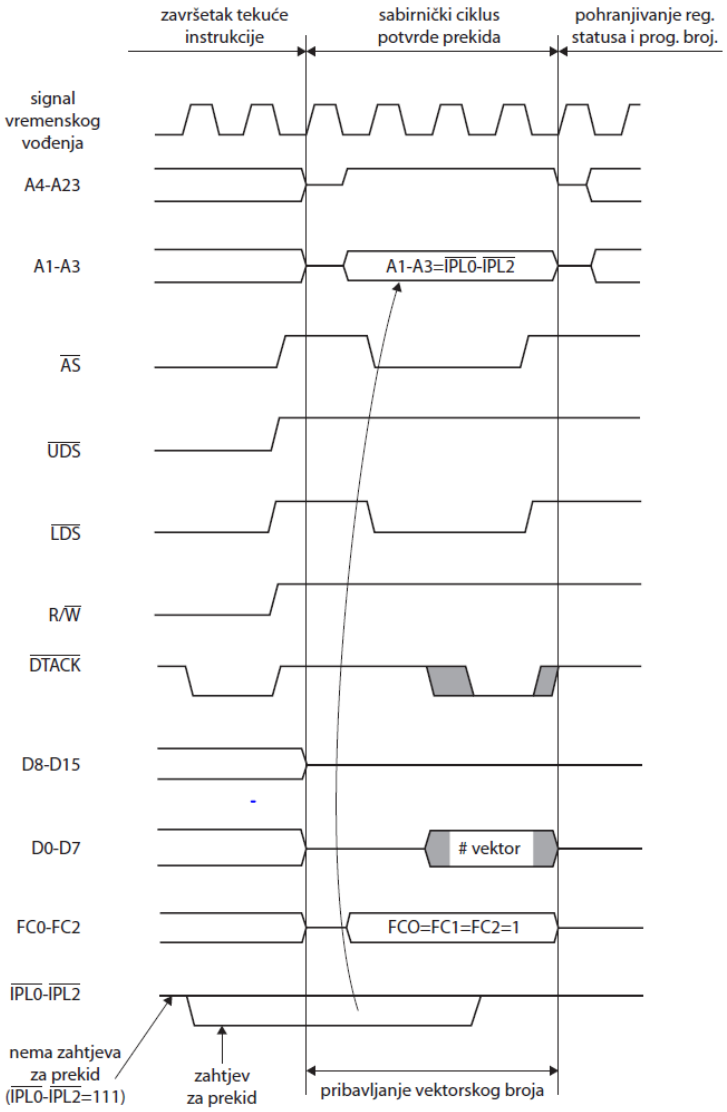


Koraci u slučaju prihvaćanja prekida

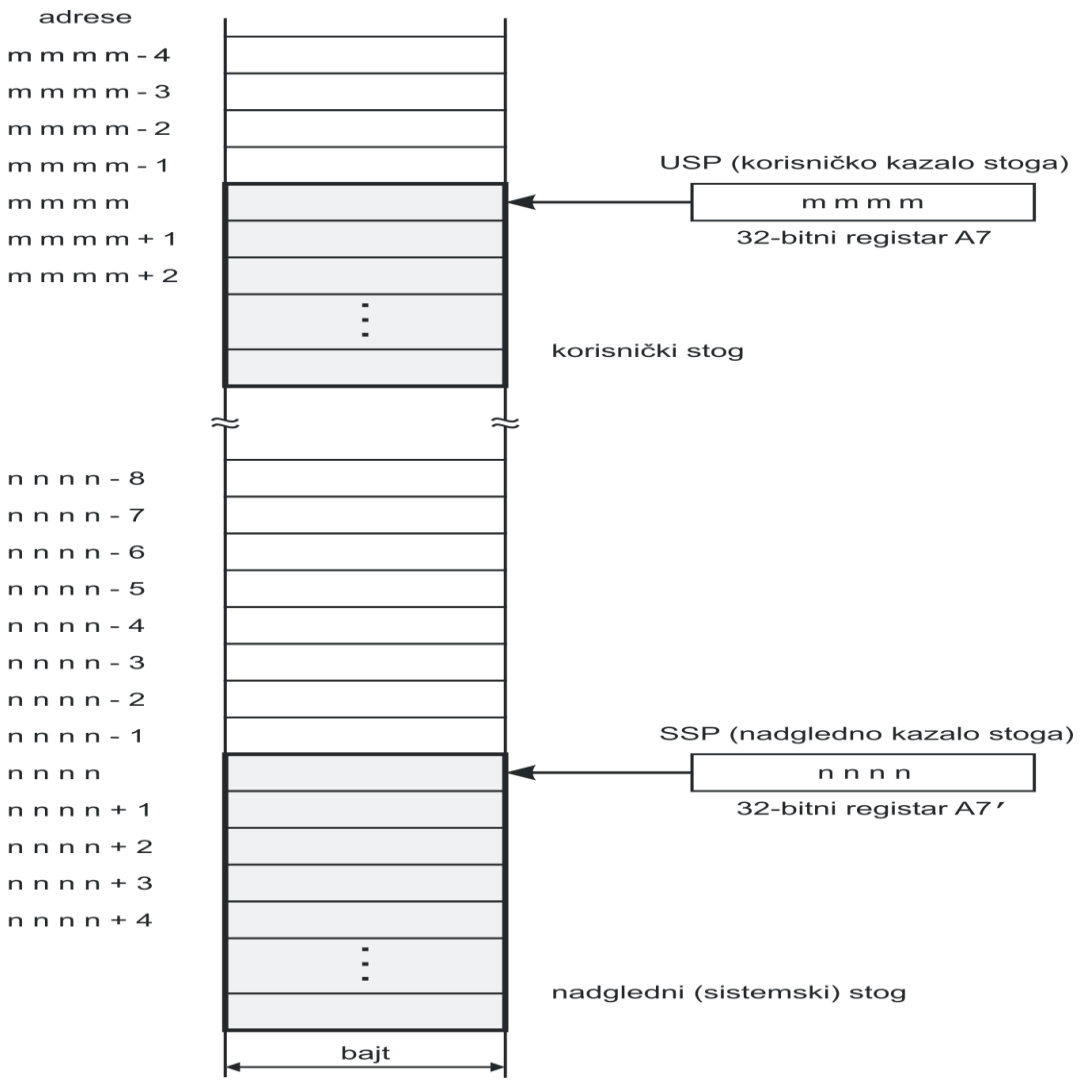
Opaska: instrukcija tijekom koje je generiran zahtjev za prekid mora završiti!

1. Sadržaj statusnog registra pohranjuje se interno u procesoru
2. Zastavica S postavlja u 1
3. Zastavica praćenja T se briše
4. Prekidne zastavice $I_0 – I_2$ poprimaju vrijednost koja odgovara razini prihvaćenog prekida
5. Procesor MC 68000 izvodi sabirnički ciklus potvrde prekida
6. Sadržaj PC (4 bajta) smještava se u nadgledni stog
7. Sadržaj interno pohranjenog statusnog registra smješta se u nadgledni stog
8. Na temelju vektorskog broja i tablice vektora procesor određuje novi sadržaj programskog brojila PC - PC se puni sa četiri bajta pribavljenih iz odgovarajućih mem. lokacija nulte stranice u kojoj je pohranjena tablica vektora iznimaka/prekida

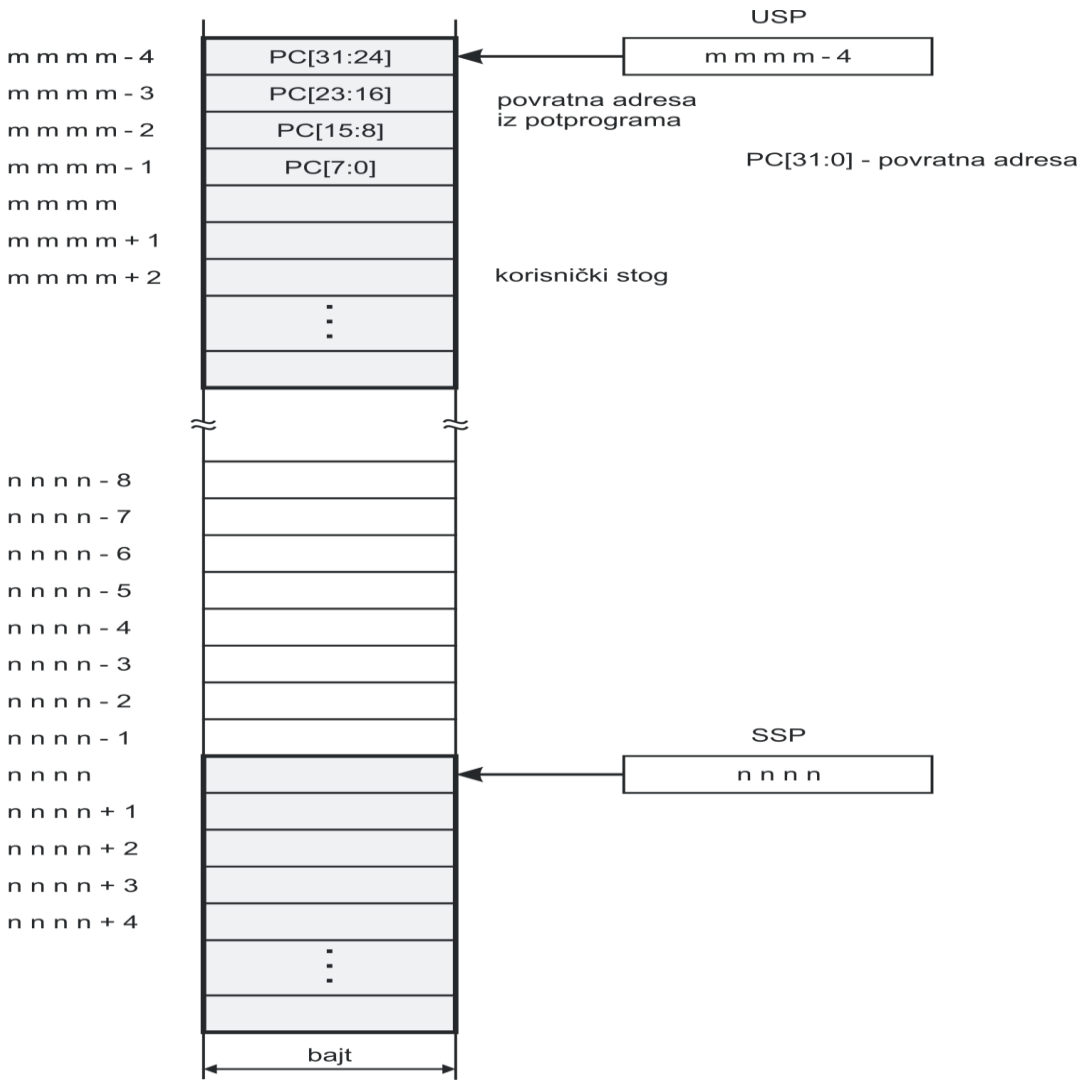
Sabirnički ciklus potvrde prekida za MC 68000



Stanja kazala stoga
i stogova prije
pozivanja potprograma

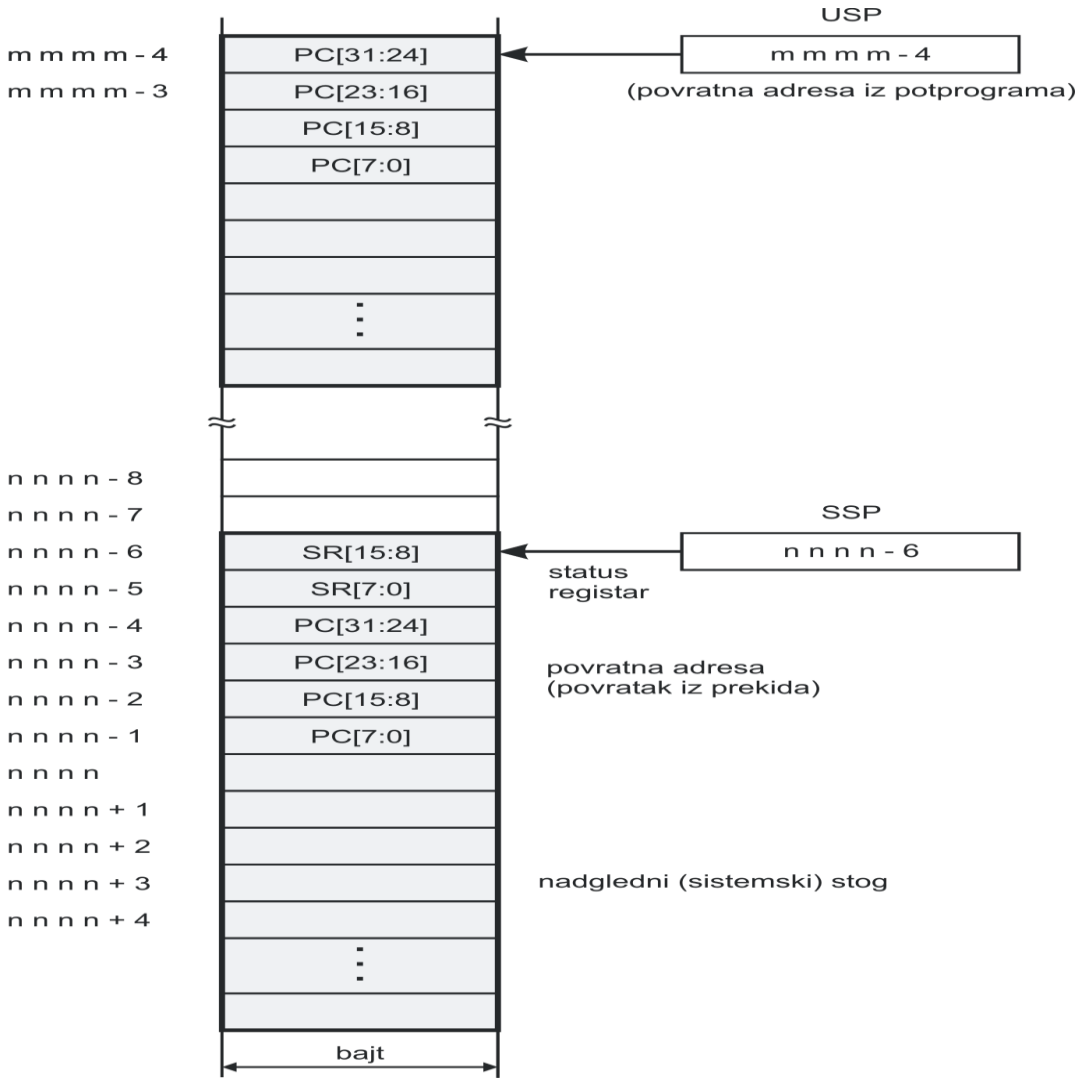


Stanje neposredno nakon grananja u potprogram

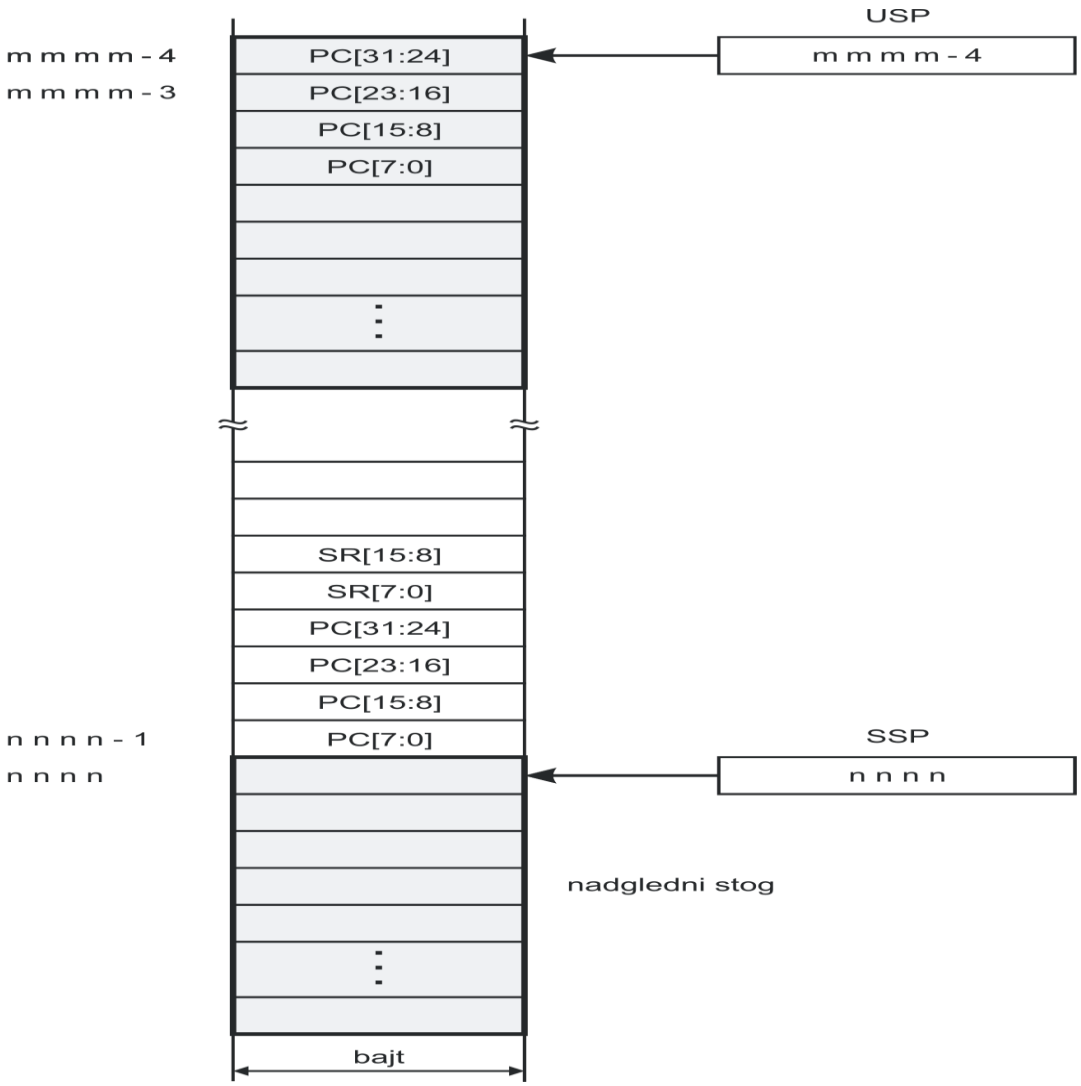


Dogodio se prekid!

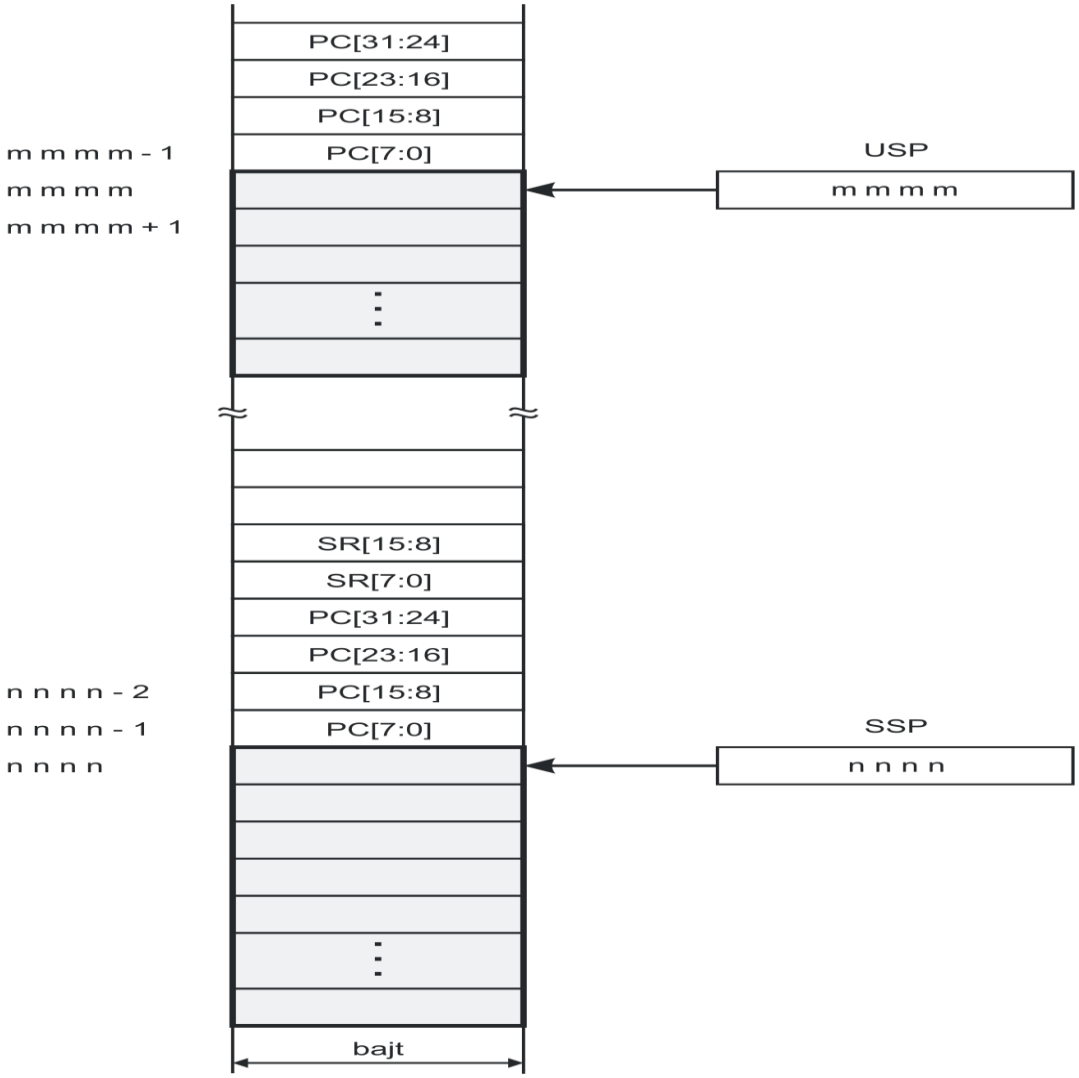
(Iznimka se obrađuje
u nadglednom načinu)

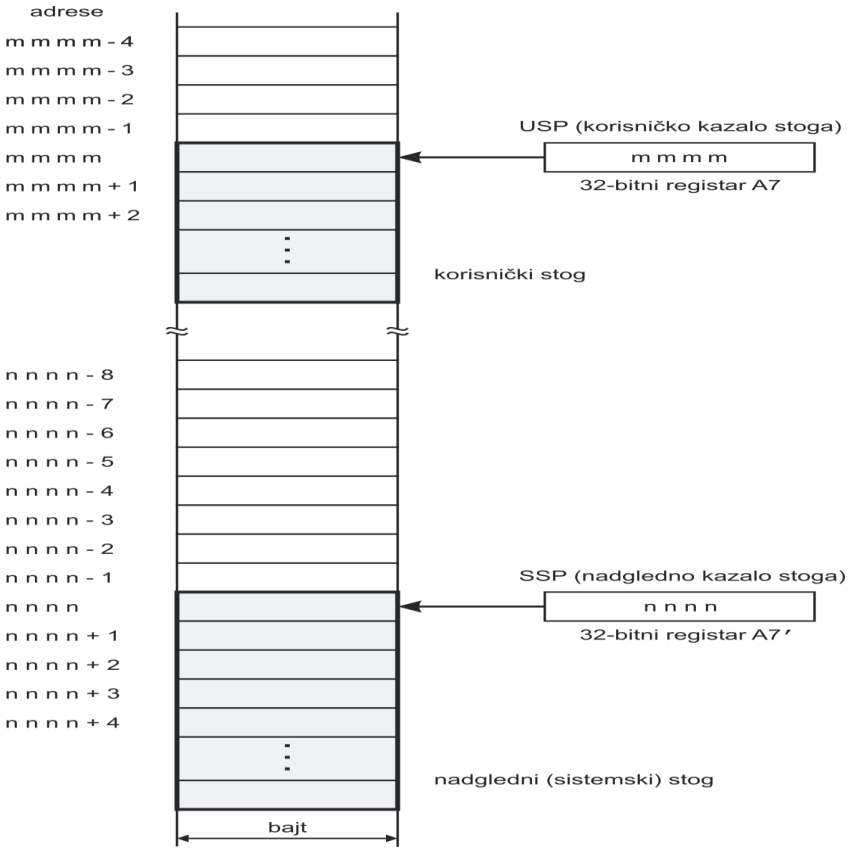


Stanje stogova nakon vraćanja u potprogram

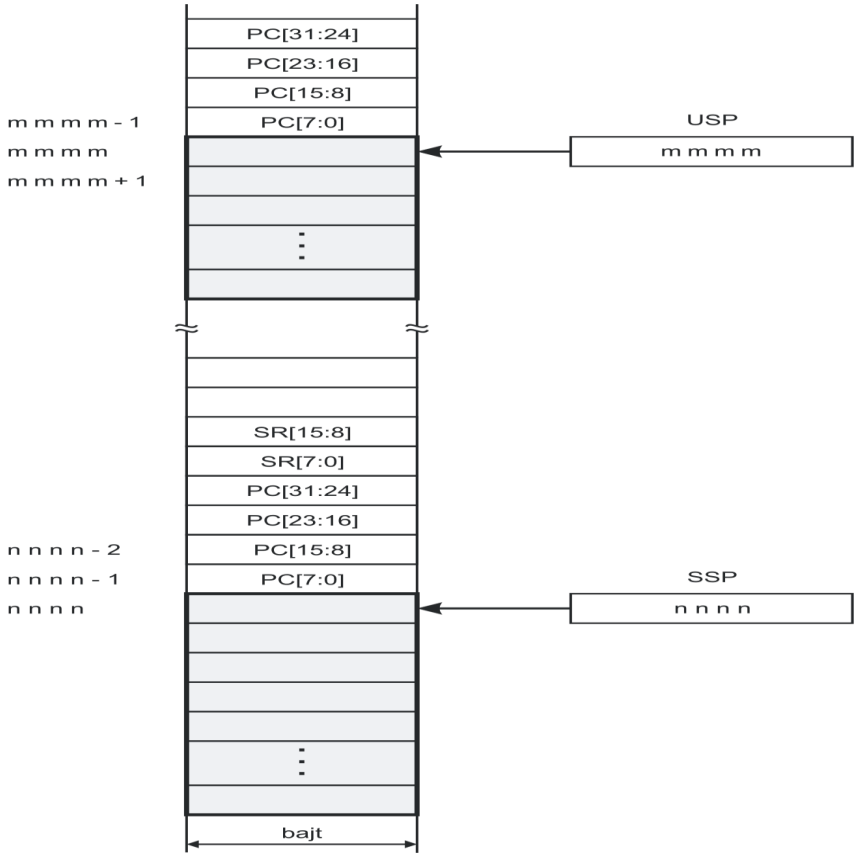


Stanje stoga nakon vraćanja iz potprograma





Stanje prije izvođenja programa

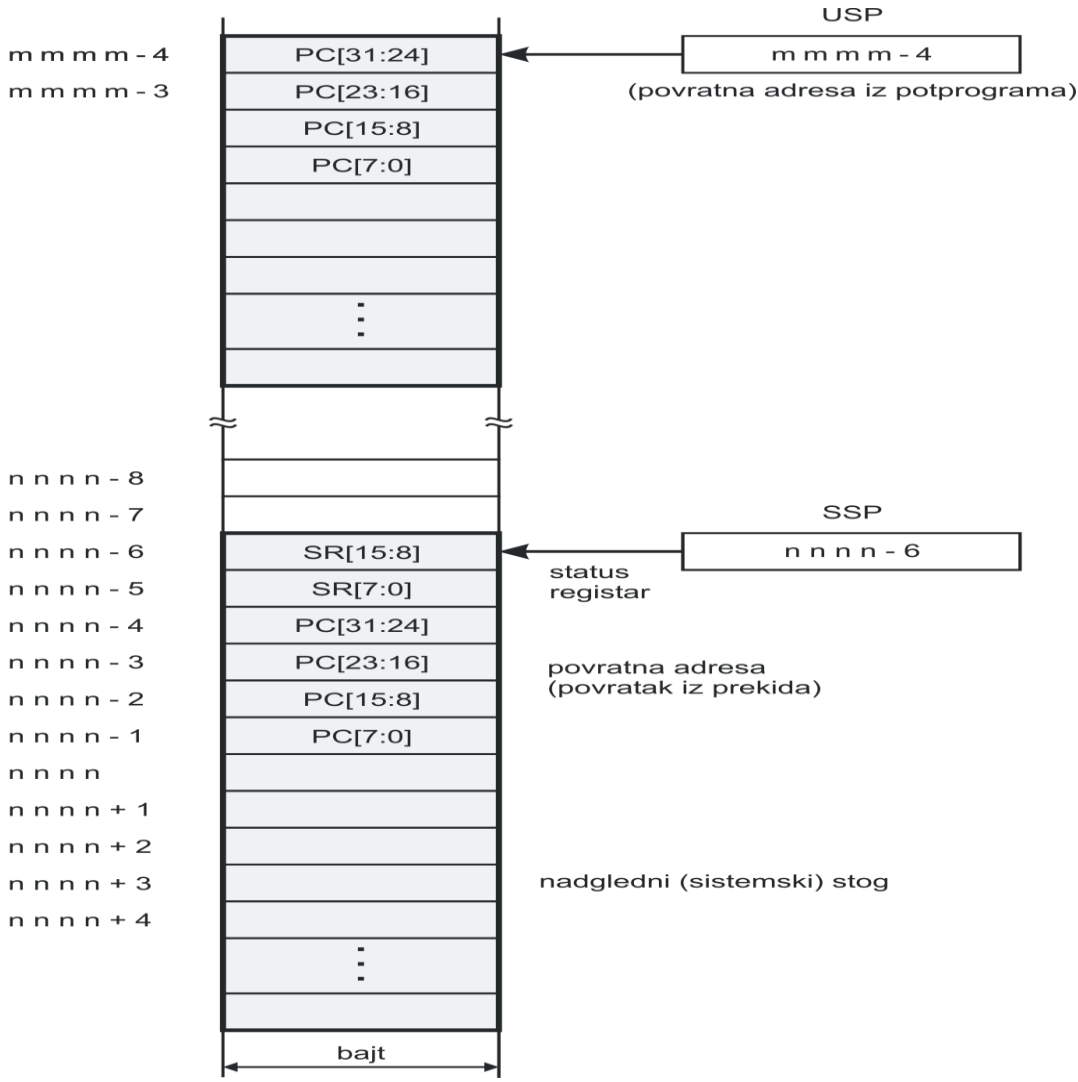


Stanje nakon izvođenja programa

Instrukcija RTE uzima s vrha stoga minimalni kontekst (6 bajtova) i smještava ih u SR i PC

Pitanje: Da li se može RTE zamijeniti s sljedećim programskim odsječkom?

MOVE.W (A7)+, SR
RTS

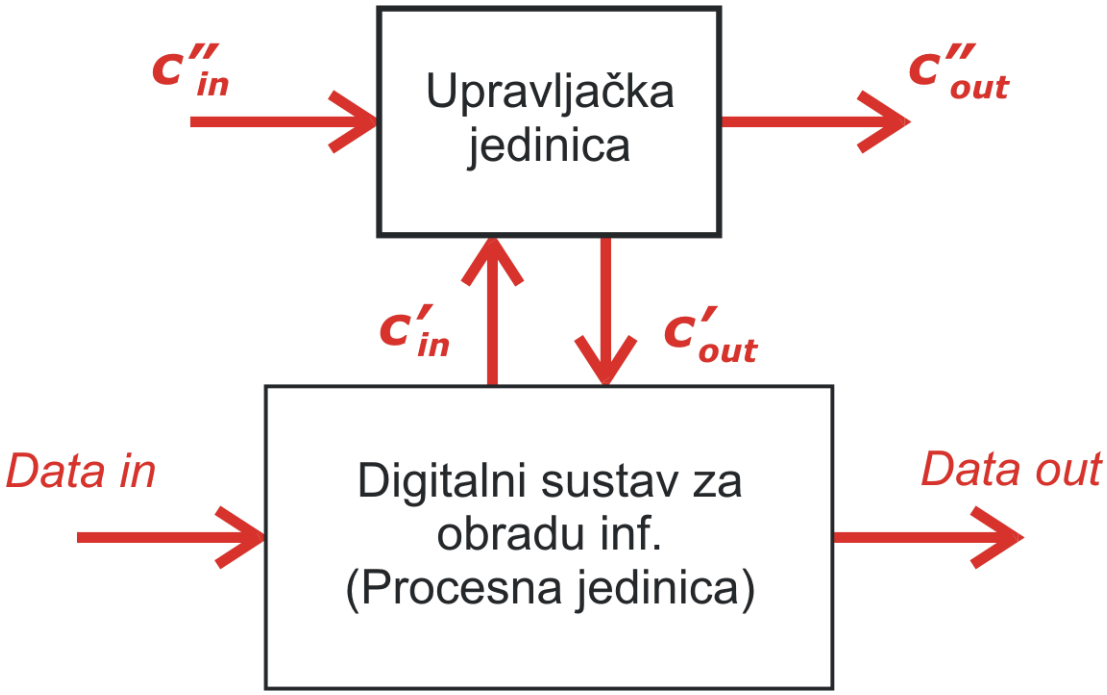


Programi za obradu iznimke (prekida) imaju općenito sljedeći oblik:

ime-iznimke MOVEM.L D_p- D_q, / A_r – A_s, -(SP)
.
.
. instrukcije programa za obradu iznimke
.
.
.
MOVEM.L (SP)+, D_p- D_q, / A_r – A_s
RTE

Instrukcija MOVEM – Move Multiple

Načini izvedbe upravljačke jedinice



Opis upravljačkih signala

C'_{OUT} - signali koji izravno upravljaju djelovanjem digitalnog sustava za obradu informacije /glavna funkcija upravljačke jedinice/

C'_{IN} - signali koji omogućuju utjecaj podataka tijekom njihove obrade – omogućavaju donošenje odluke u zavisnosti od rezultata operacije na podacima

C''_{IN} - prijemni signali (od drugih upravljačkih jedinica ili od središnje nadgledne jedinice, npr. *start*, *stop*)

C''_{OUT} - signali prema drugim upravljačkm jedinicama (npr. *busy*, *operation completed*)

Opisivanje djelovanja upravljačke jedinice

• Dijagram toka i opisni jezici (engl. Description language)

- dijagram toka opisuje mikrooperacije i njihov slijed
  svakoj mikrooperaciji se u dijagramu toka pridružuje skup upravljačkih signala $\{C_{i,j}\}$koji se moraju aktivirati da bi se mikrooperacija izvršila
- sklopovski opisni jezici: RTL (Register Transfer Language), CDL (Computer Design Language), VHDL,...

Načini izvedbe upravljačke jedinice

1) Sklopovska izvedba - sekvencijalni sklopovi (engl. Hardwired control unit)

2) Mikroprogramska izvedba

Sklopovska izvedba

-tri pristupa:

1) Standardni pristup oblikovanju sekvencijalnih sklopova
   (uporaba tablica ili dijagrama stanja)

2) Uporaba elemenata za kašnjenje

3) Uporaba brojila sekvenci

Tablica stanja

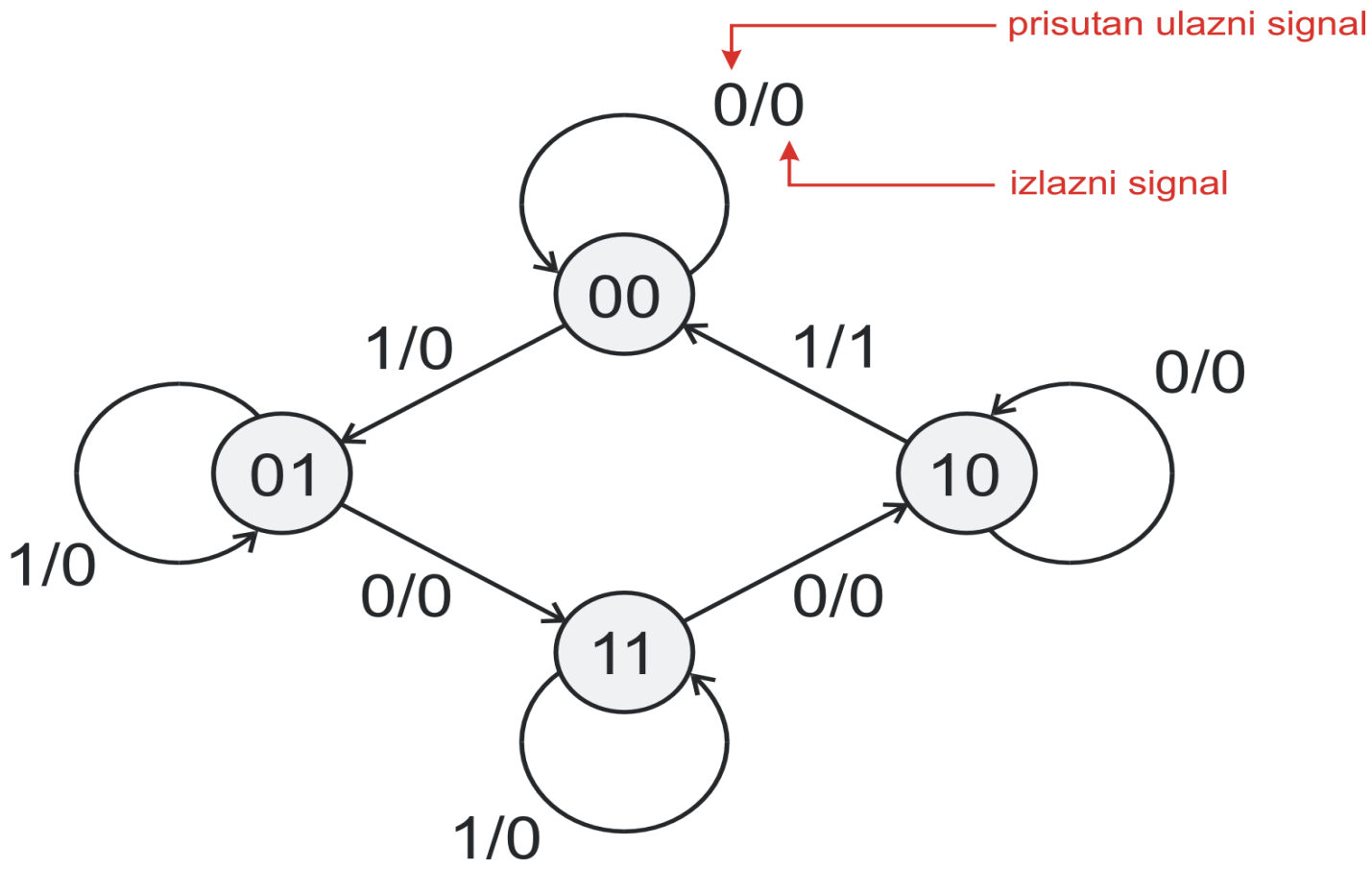
Ulazne kombinacije

Stanja	I_1	I_2	...	I_m
S_1	$S_{1,1}$, $Z_{1,1}$	$S_{1,2}$, $Z_{1,2}$	...	$S_{1,m}$, $Z_{1,m}$
S_2	$S_{2,1}$, $Z_{1,2}$	$S_{2,2}$, $Z_{2,2}$	...	$S_{2,m}$, $Z_{2,m}$
S_3	...	...	...	...
...	...	...	...	...
...	...	...	...	...
S_n	$S_{n,1}$, $Z_{n,1}$	$S_{n,2}$, $Z_{n,2}$	...	$S_{n,m}$, $Z_{n,m}$

$S_{i,j}$ – označava slijedeće stanje

$Z_{i,j}$ – označava skup izlaznih signala koji se aktiviraju s I_j ako je upravljačka jedinica u stanju S_i

Dijagram stanja



Uporaba elemenata za kašnjenje

-razmotrimo problem generiranja slijeda upravljačkih signala u vremenskim trenucima $t_1, t_2, \ldots, t_n$

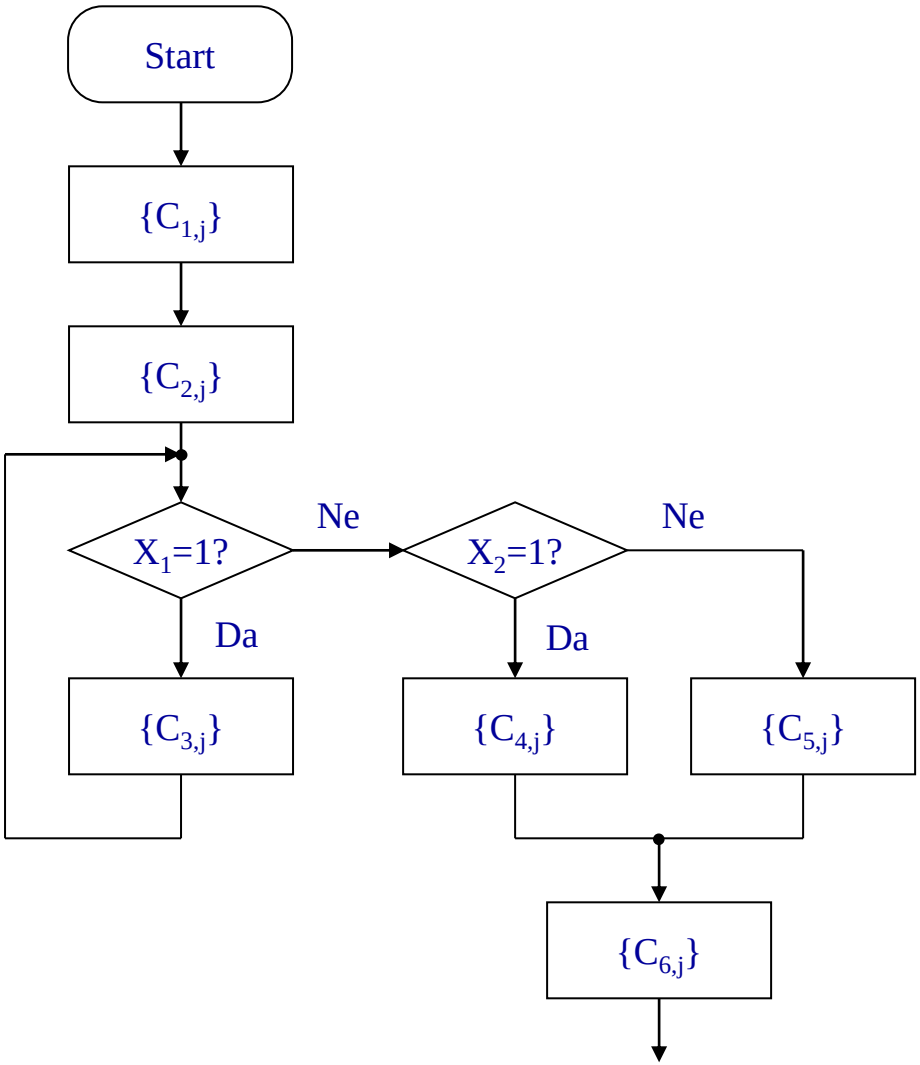
t_1: aktiviraj $\{C_{1,j}\}$
t_2: aktiviraj $\{C_{2,j}\}$
. ...
. ...
$t_{n:}$ aktiviraj $\{C_{n,j}\}$

- Pretpostavimo da je u trenutku t_1 prisutan signal *Start(t_1)* i on će u trenutku t_1 aktivirati skup signala $\{C_{1,j}\}$ koji će pobuditi jednu (prvu) ili više mikrooperacija
- Signal *Start(t_1)* dovodimo na element za kašnjenje (kašnjenje = $t_2 - t_1$) – izlaz iz elementa za kašnjenje aktivira $\{C_{2,j}\}$ itd.

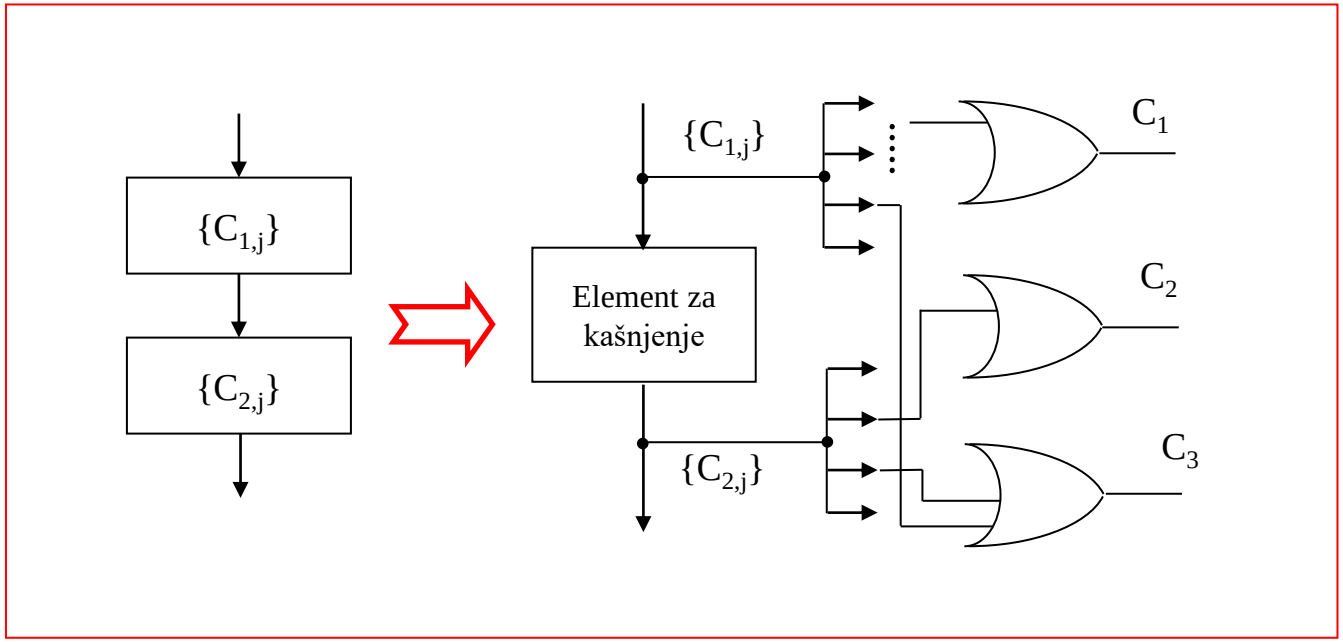
-Slijed elemenata za kašnjenje može se upotrijebiti za generiranje upravljačkih signala

Oblikovanje upravljačke jedinice – izravno iz dijagrama toka

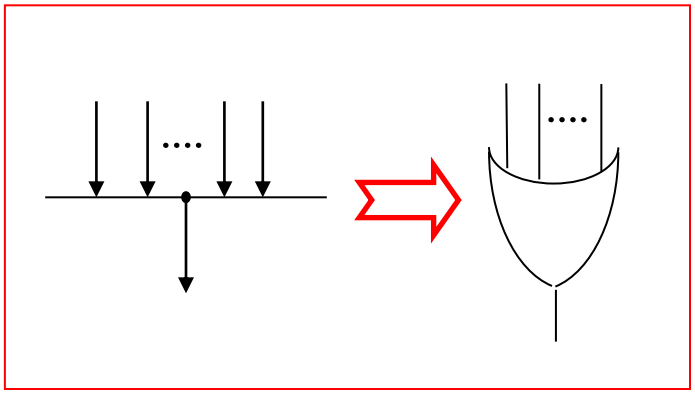
Primjer

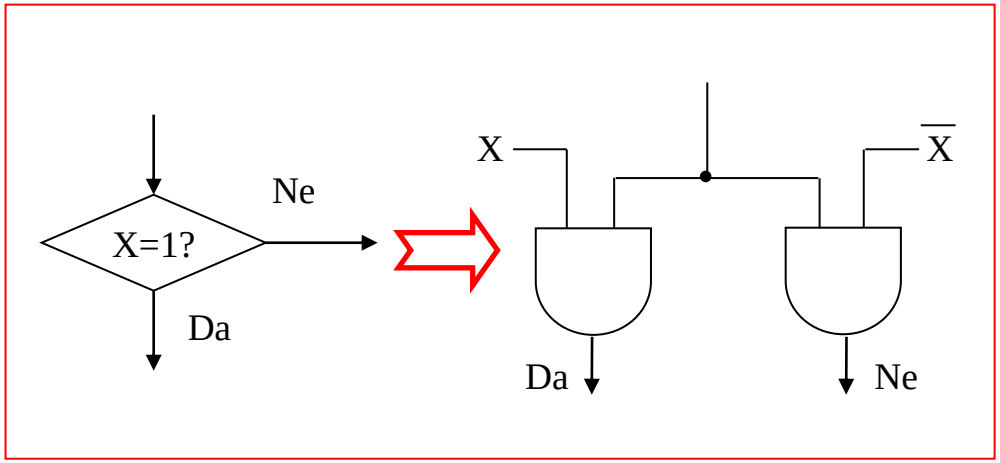


Odnos: dijagram toka ⟷ sklopovi

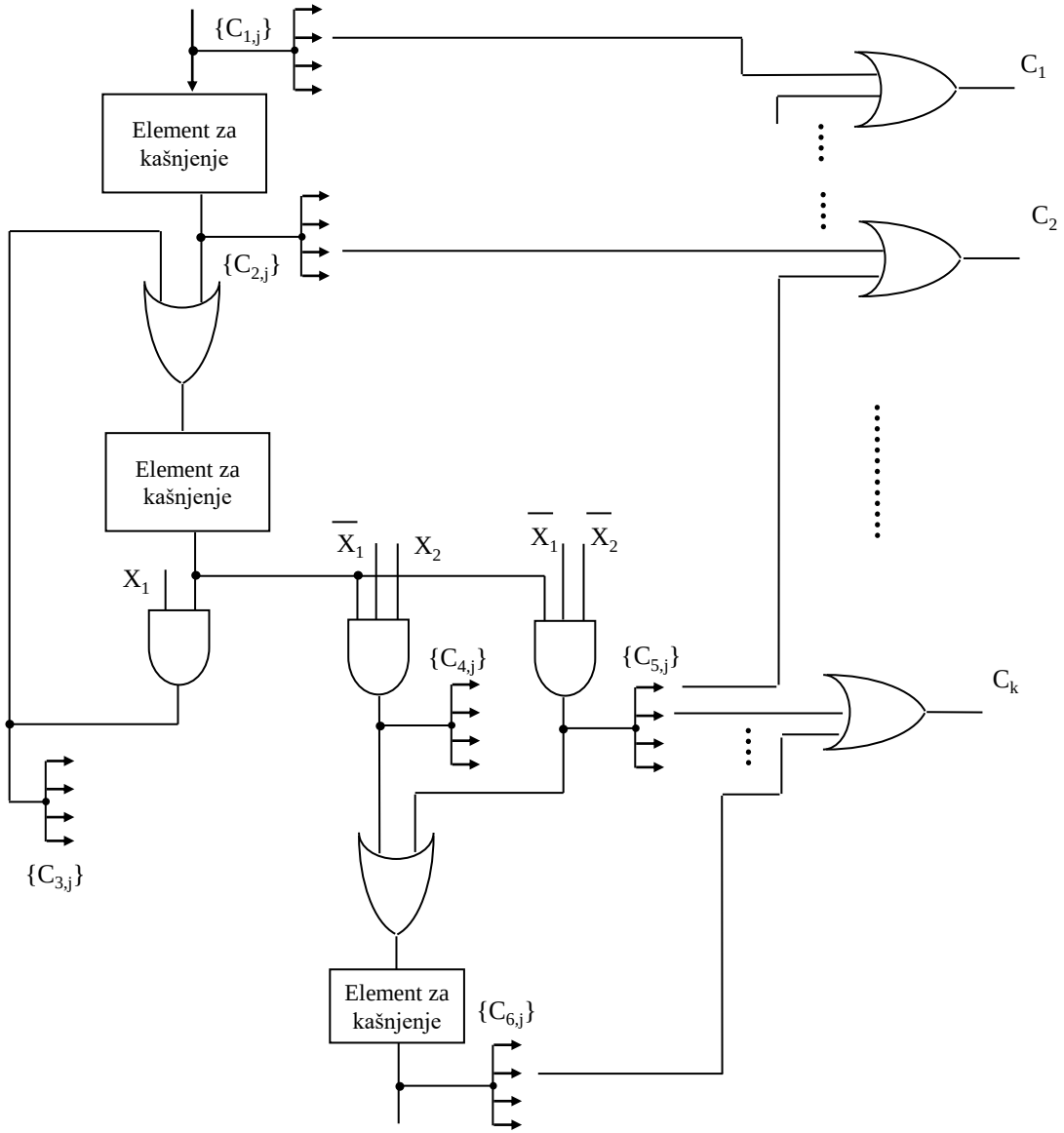


Odnos: dijagram toka ↔ sklopovi

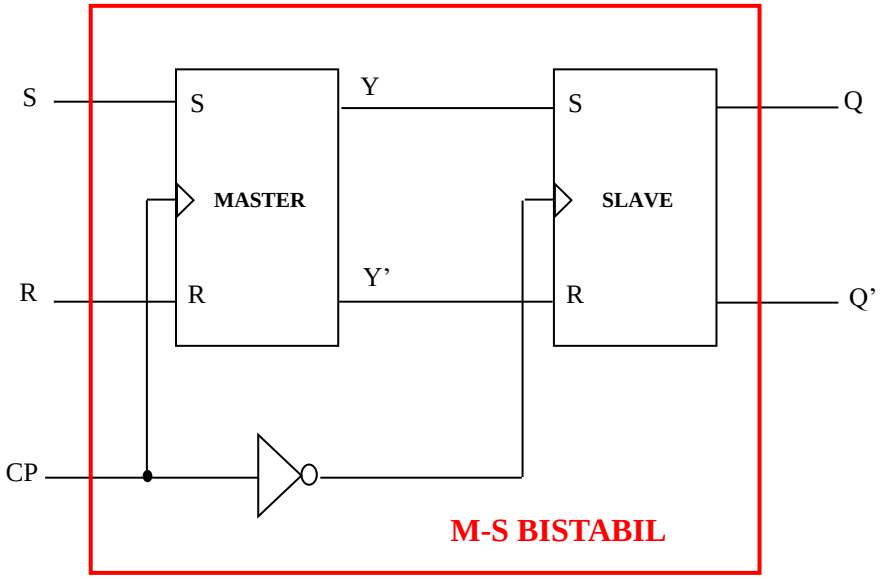


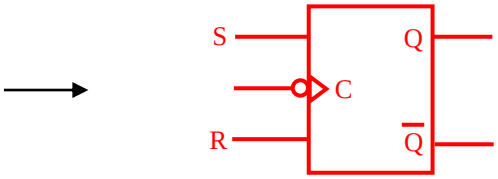


Izvedba:


{C1,j}
Element za kašnjenje
{C2,j}
Element za kašnjenje
X1
X2
X1
X2
{C4,j}
{C5,j}
{C3,j}
Element za kašnjenje
{C6,j}
C1
C2
Ck

Izvedba jednostavnog elementa za kašnjenje – dvostruki bistabil (engl. Master-slave)





Ulaz SR

Stanje Q	00	01	10
0	0	0	1
1	1	0	1