

6. Upravljačka jedinica

- 6.1. Funkcija upravljačke jedinice
- 6.2. Prijenos upravljanja između programa
- 6.3. Rekurzivni programi
- 6.4. LIFO ili stožna struktura
- 6.5. Uporaba stoga (Analiza slučaja)

Funkcije upravljačke jedinice:

- Pribavljanje instrukcija
- Tumačenje instrukcija
- Generiranje upravljačkih signala tijekom instrukcijskog ciklusa
- Upravljanje slijedom instrukcija: izbor instrukcije – prijenos upravljanja s jedne na drugu instrukciju (engl. Instruction sequencing)

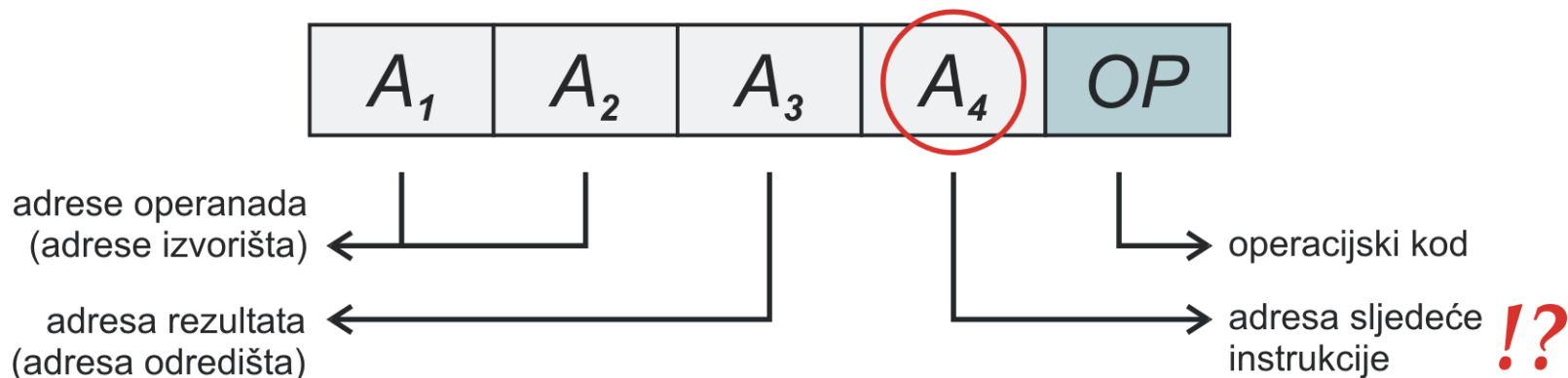
Najjednostavnija metoda upravljanja slijedom instrukcija:

- 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 procesor RISC k=4 ako je 32 – bitni procesor/

Prijenos upravljanja između instrukcija koje si nisu slijedne:

3) Instrukcije grananja

4) Instrukcije za prijenos upravljanja između programa

1) Instrukcije grananja

- Instrukcije bezuvjetnog grananja

$PC \leftarrow X$, 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 $PC \leftarrow X$, u drugim slučajevima *PC* se ne mijenja (tijekom faze IZVRŠI)

2) Instrukcije za prijenos upravljanja između programa

P1 – program (glavni program)

P2 – potprogram

ili

P1 – pozivajući program

P2 – pozvani program

Dva glavna slučaja:

14) Pozivanje potprograma

15) Prekidi

1) $P1 \longrightarrow P2$ 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.

Tijekom faze IZVRŠI instrukcije CALL obavljaju se dva slijedeće koraka:

4. Korak: Sadržaj PC-a (koji pokazuje na slijedeću instrukciju u P1) se pohranjuje na za to predodređenu lokaciju **S**,
7. Korak: X (ili adresa izračunata na temelju X) prenosi se u PC

Lokacija **S** sadrži **povratnu adresu**

Povratak iz programa P2 ($P2 \rightarrow P1$):

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

$$PC \leftarrow S$$

Problem: Gniježđenje potprograma!

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

JMS SUB ; Jump to Subroutine

- vi) PC se pohranjuje na lokaciju SUB
- vii) 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!

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

Rekurzivni program P može se prikazati kao kompozicija Π osnovnih instrukcija s_i (koje ne sadrže P) i samog programa P :

$$P = \Pi [s_i, P]$$

Primjer: PROLOG

predak_od (X,Y) :- roditelj_od (X, Y).

predak_od (X, Y) :- roditelj_od (Z, Y), predak_od (X, Z).

Primjer:

$$n! = 1 \times 2 \times 3 \times 4 \times \dots \times n$$

$$1! = 1$$

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

Hanojski tornjevi

Fibonaccijevi brojevi:

$$10) F_0 = 0, F_1 = 1; i$$

$$11) F_n = F_{n-1} + F_{n-2}, \text{ za } n \geq 2$$

Rješenje problema rekurzije: **Uporaba upravljačkih stogova**

Stog – LIFO (Last In First Out) služi za pohranu povratnih adresa.

CALL SUB

5) **PUSH PC**

6) $PC \leftarrow SUB$

RETURN

1) **POP PC**

Rekurzivno pozivanje:

Begin

.

.

CALL SUB

X: . ; *X* je povratna adresa

.

.

SUB: .

.

CALL SUB

Y: . ; *Y* – povratna adresa

.

.

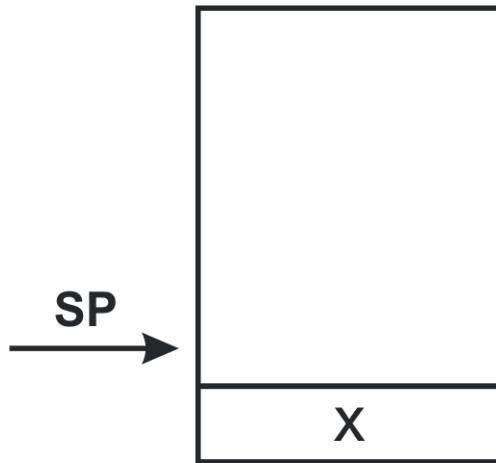
RETURN

.

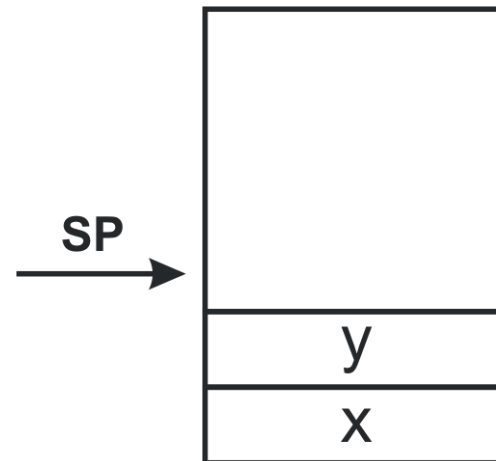
End

Stanje stoga:

Nakon prvog pozivanja



Nakon drugog pozivanja



Treće pozivanje, četvrto pozivanje, ...

Povratak???

Rekurzivno pozivanje:

Begin

.

$N := 3$

CALL SUB

X: .

.

SUB: .

.

$N := N - 1$

IF (N > 0) THEN CALL SUB

Y: .

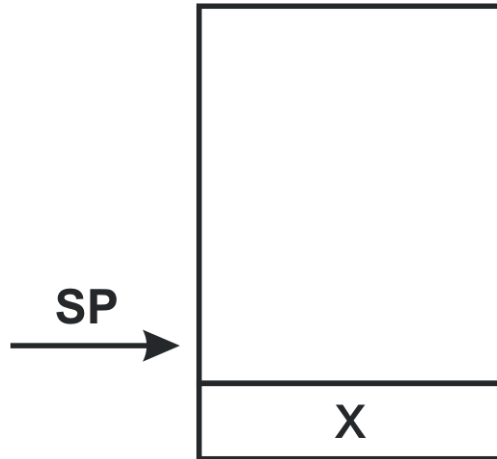
.

RETURN

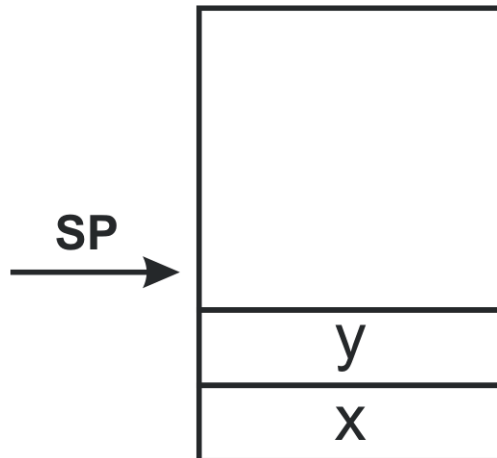
End

Stanje stoga:

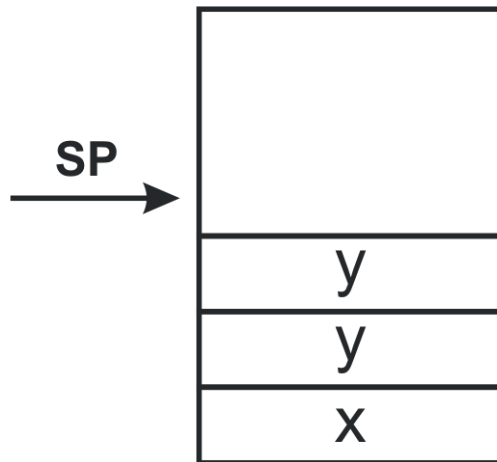
- Prvi poziv ($N = 3$) / poziv iz glavnog programa/:



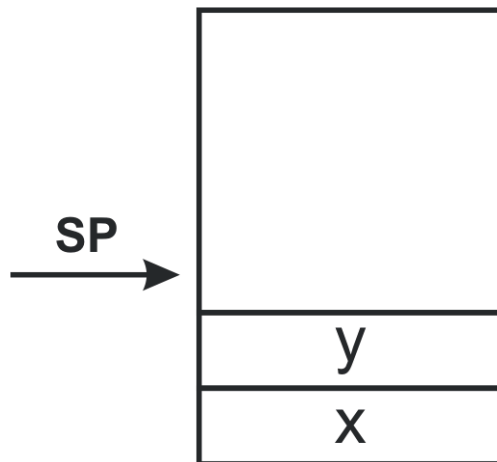
- Drugi poziv ($N = 2$) / poziv iz SUB/:



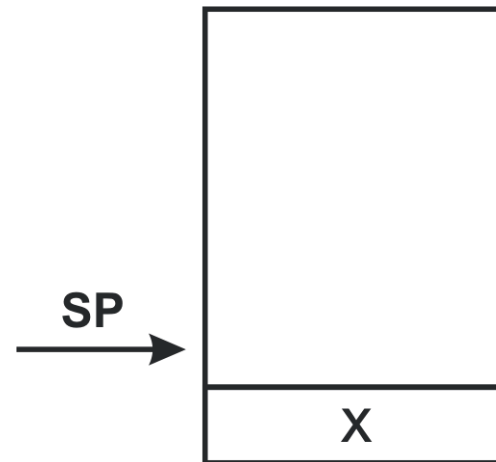
Treći poziv ($N = 1$) /poziv iz SUB/:



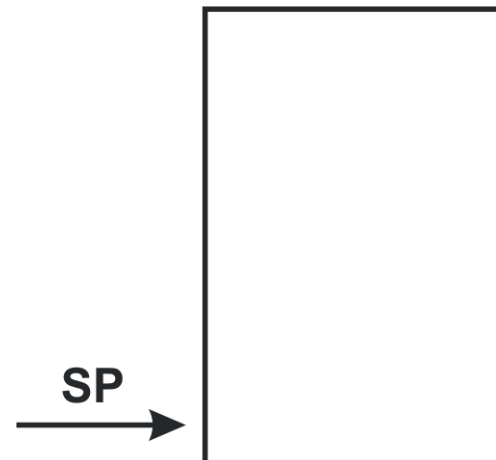
Prvi povratak:



Drugi povratak:



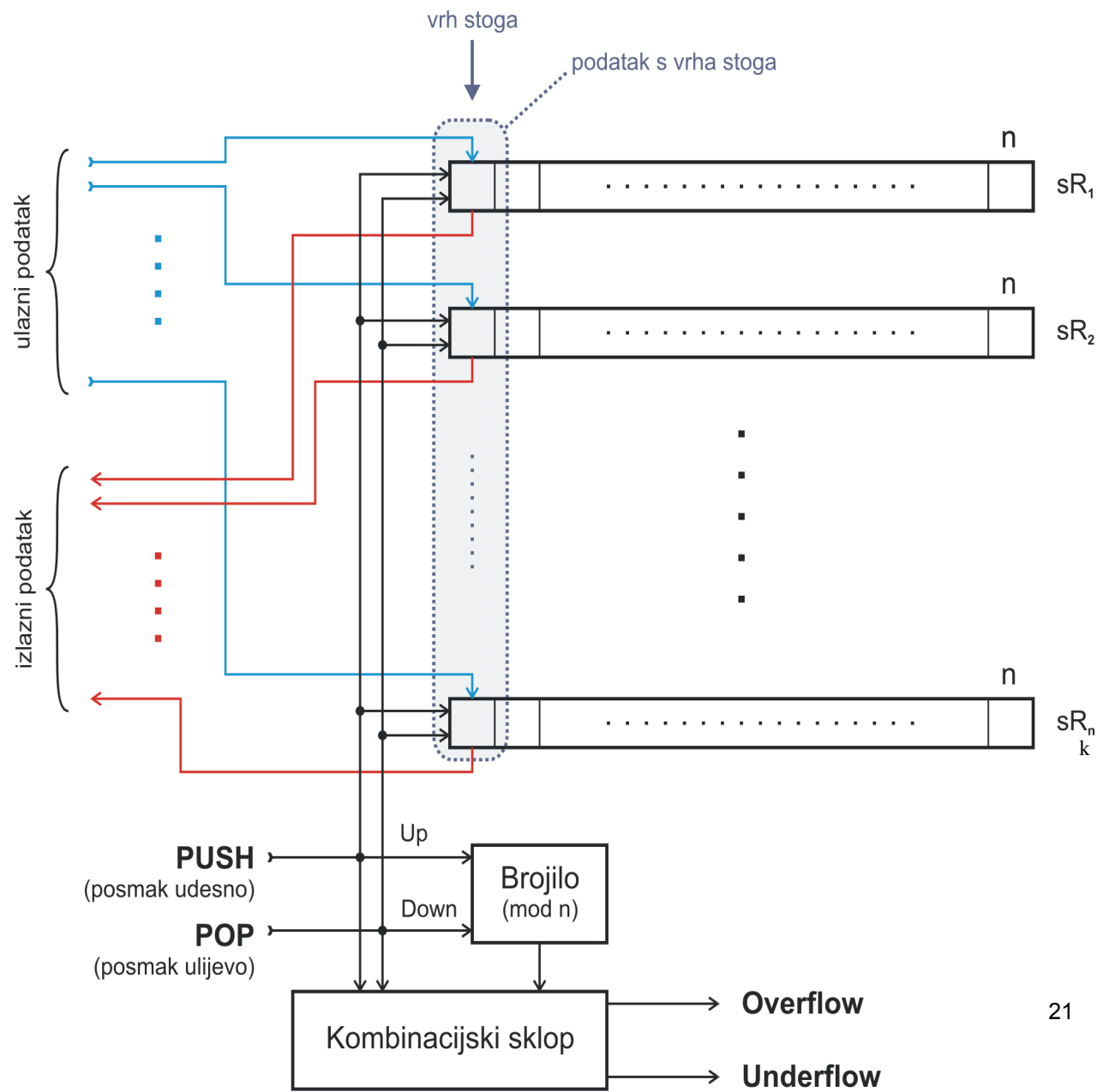
Treći povratak:



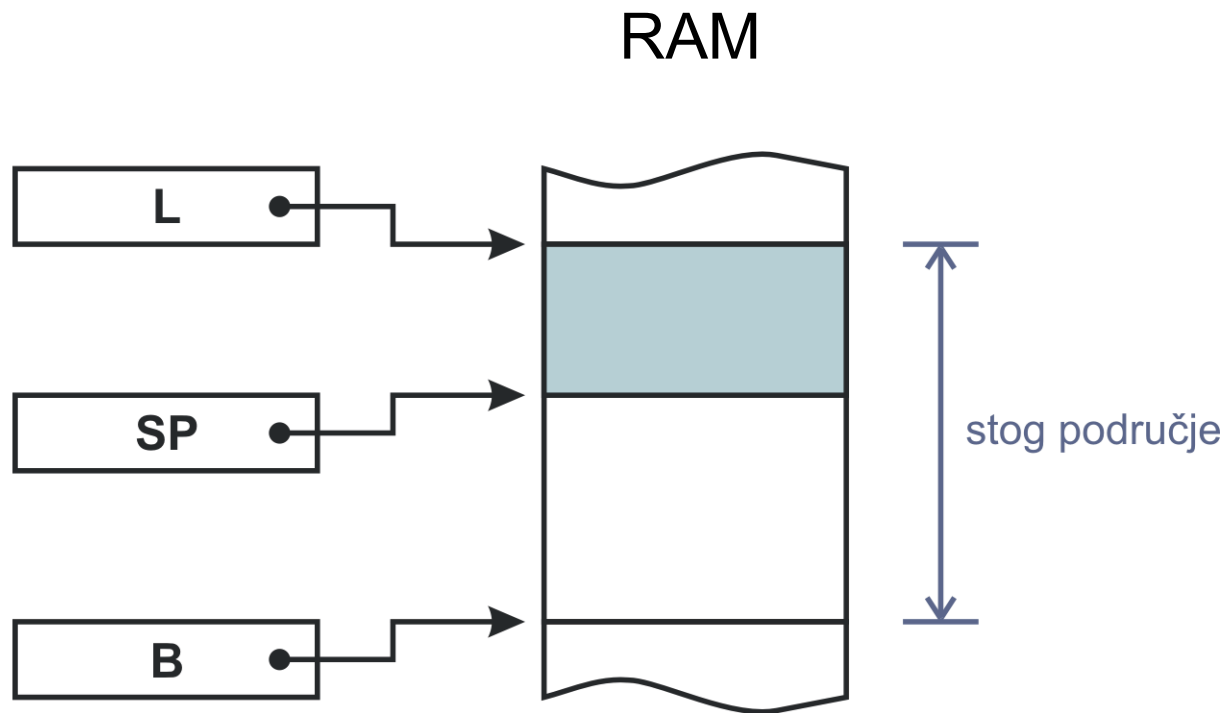
Primjeri izvedbe stoga:

- 3) Stog od n k -bitnih riječi izveden pomoću posmačnih registara
- 4) Uporaba memorije s izravnim pristupom kao područje stoga

1.
sklopovska
izvedba stoga
dubine n i
duljine riječi k
bita




2. programska izvedba stoga



Primjer uporabe stoga

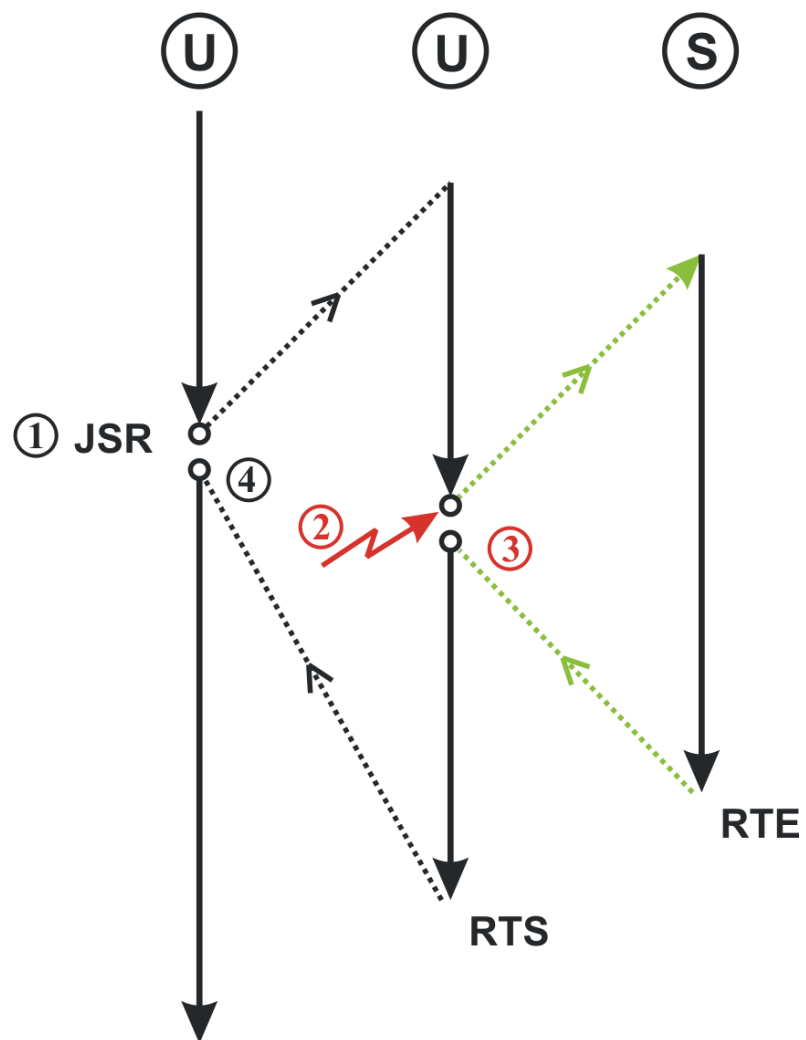
Analiza slučaja: MC 68000

Scenarij:

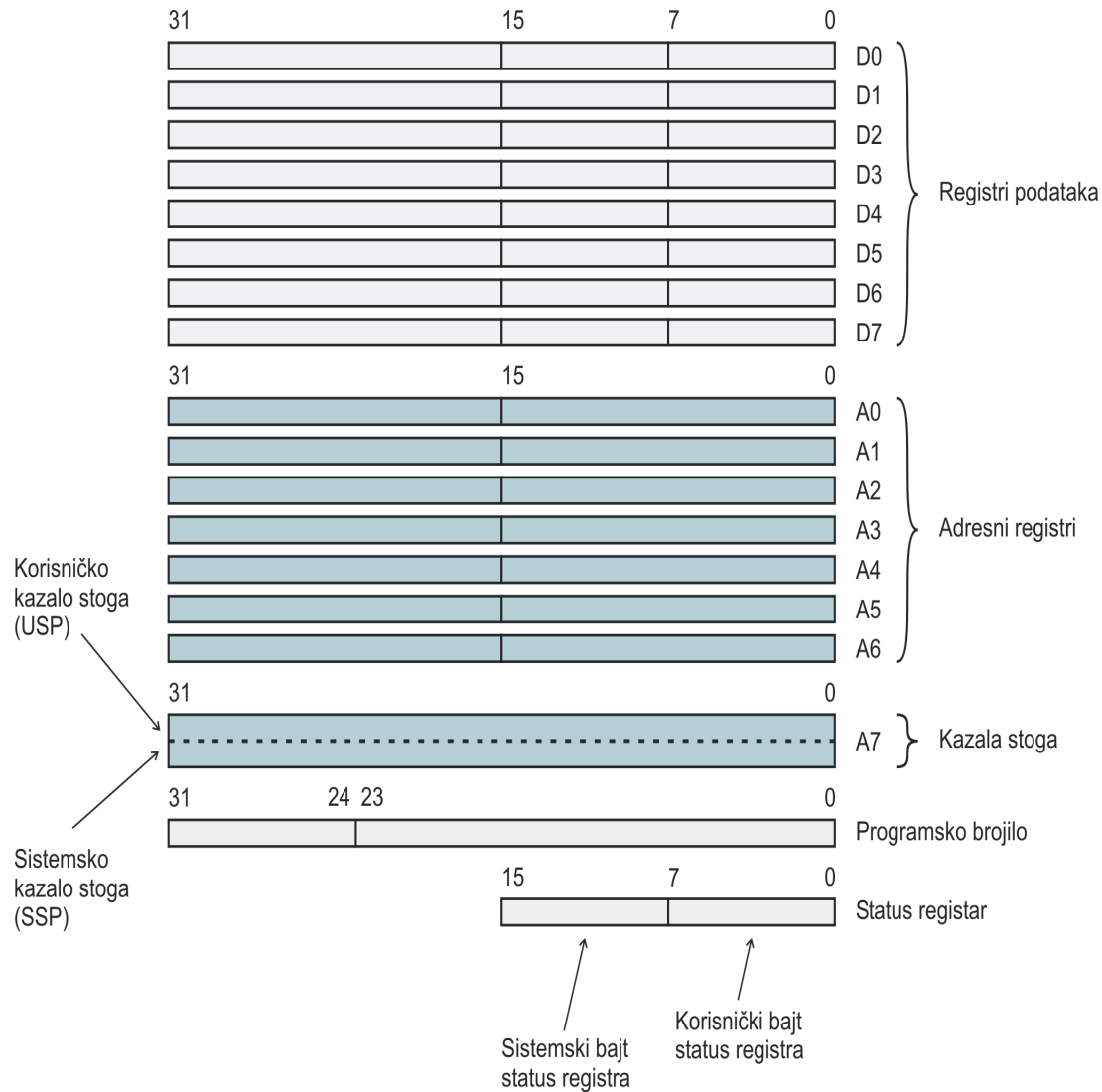
- ①. Procesor je u korisničkom načinu rada (User Mode)
 - Poziva se potprogram
 - Nastavlja se izvođenje potprograma
- ②.  **Dogodila se iznimka (PREKID)**
 - Obrada prekida
- ③. Vraćanje u potprogram
- ④. Vraćanje iz potprograma

/primjer detaljno opisan u S. Ribarić, Naprednije arhitekture mikroprocesora,
Element, 1997., Zagreb/

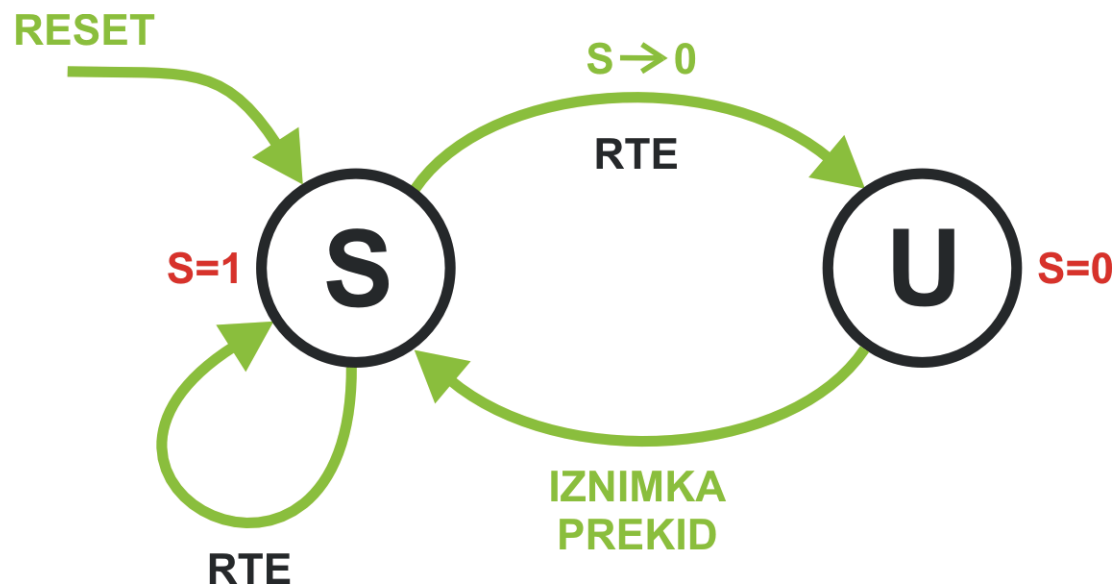
Grafički prikaz scenarija:



Programski model MC 68000



Dijagram stanja za MC 68000



RTE -Return from Exception /povlaštena instrukcija/

Iznimka (engl. exception)

- posebne okolnosti kojima se narušava normalno stanje procesora i izvođenje programa
- nasilan prekid normalnog izvođenja programa i prijenos upravljanja bez upotrebe posebnih instrukcija

Iznimke mogu biti dvojake:

1. vanjske iznimke (prekid, sabirnička pogreška, reset)
2. unutarnje iznimke (dijeljenje nulom, uvjetne i bezuvjetne zamke (engl. trap), povreda privilegiranosti,...)

Iznimke:

- a) asinkrone
- b) sinkrone

Obrada iznimke za MC 68000

Tipični koraci:

1. korak: 16-bitni status-registar SR interno se pohranjuje u interni (privremeni) registar. U status-registru postavljaju se zastavice u stanje koje odgovara obradi iznimke:
$$S \rightarrow 1 \text{ i } T \rightarrow 0;$$
2. korak: utvrđuje se vektorski broj iznimke, odnosno vektor iznimke. Vektorski se broj iznimke, prema vrsti iznimke, dobiva od vanjskog uređaja ili ga procesor generira interno;
3. korak: pohranjuje se sadržaj programskog brojila PC i sadržaj interno pohranjenog status-registra SR;
4. korak: na temelju vektorskog broja iznimke i tablice vektora iznimaka određuje se novi sadržaj programskog brojila PC i procesor nastavlja rad izvođenjem prve instrukcije iznimke.

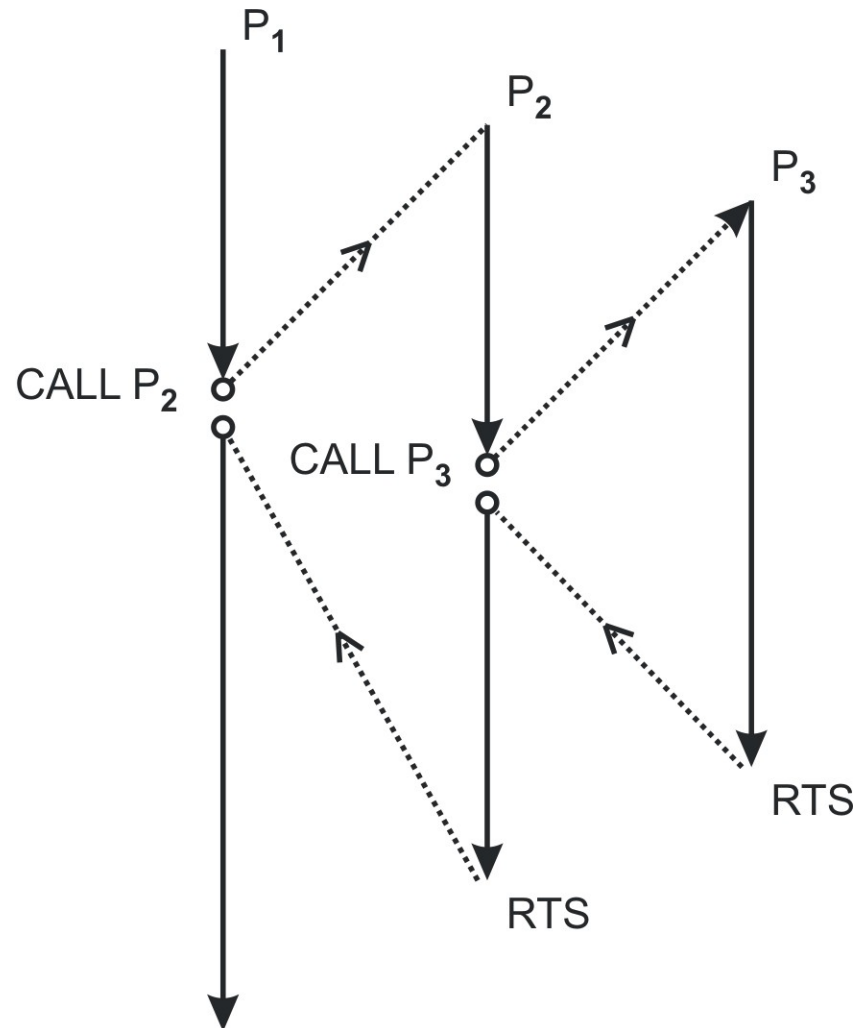
- CALL POT se izvodi u dva koraka:

- PUSH PC
- $PC \leftarrow POT$

Povratak:

- RET:
POP PC

Uporaba stoga!!!

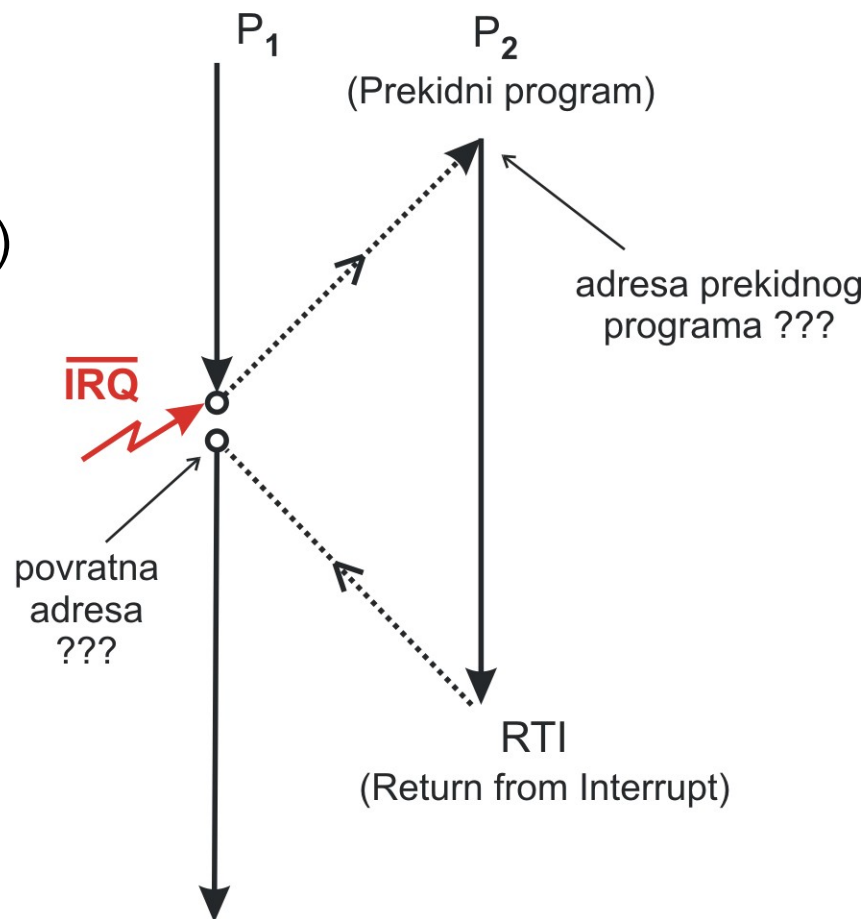


Na stog se tijekom prijenosa upravljanja s prekinutog na prekidni program ($P_1 \rightarrow P_2$) **pohranjuje**

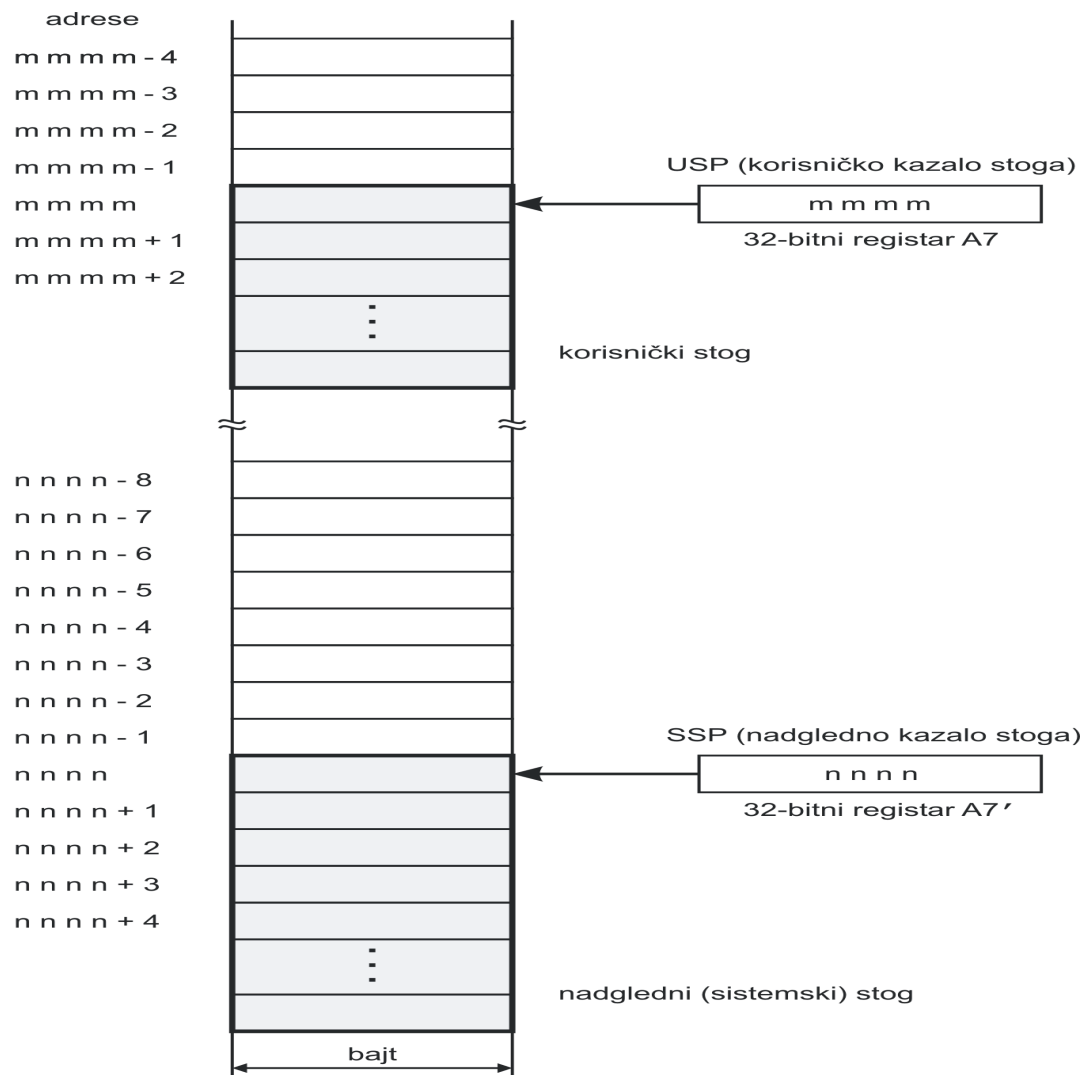
MINIMALNI KONTEKST:

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

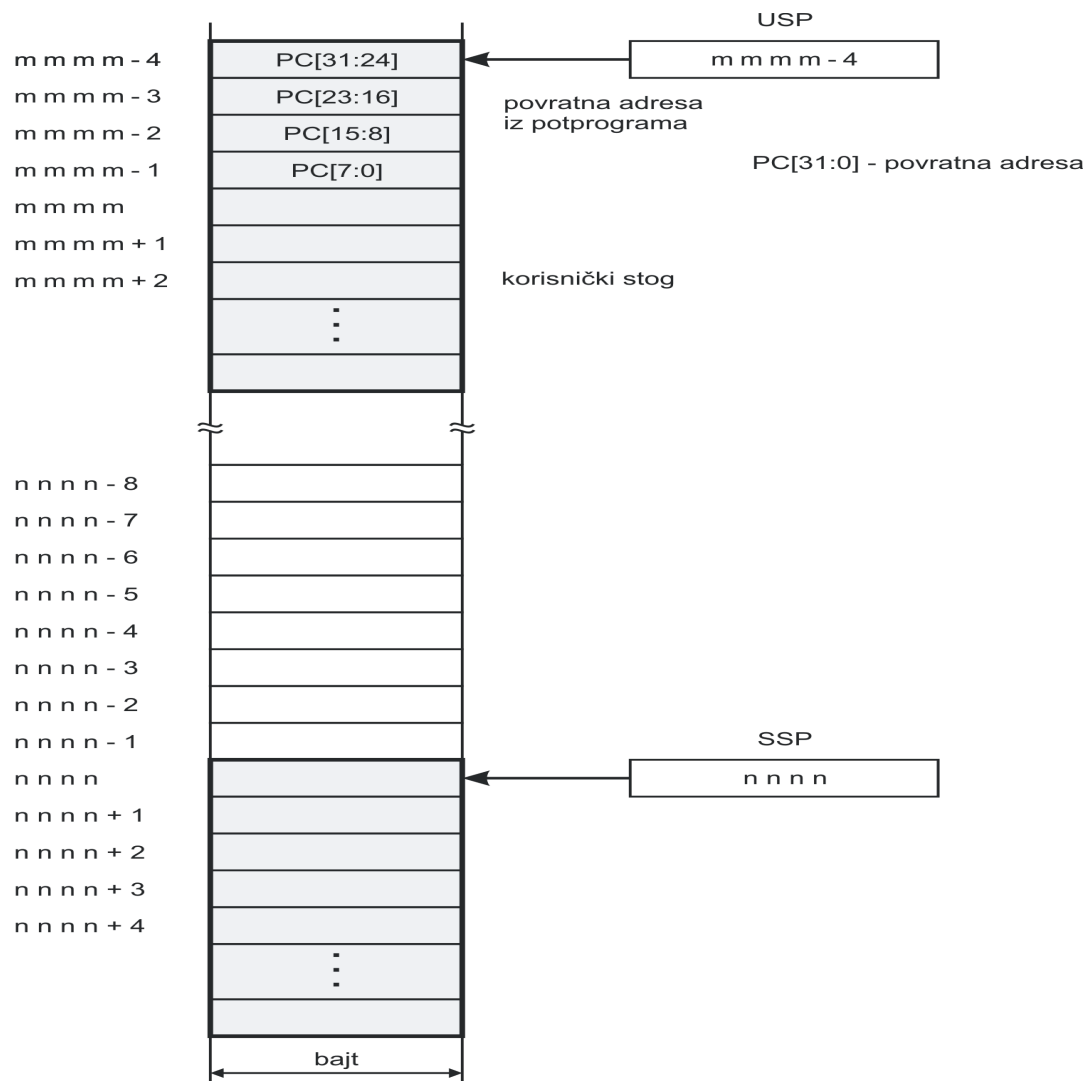
Adresa prekidnog programa?



Stanja kazala stoga
i stogova **prije**
pozivanja potprograma

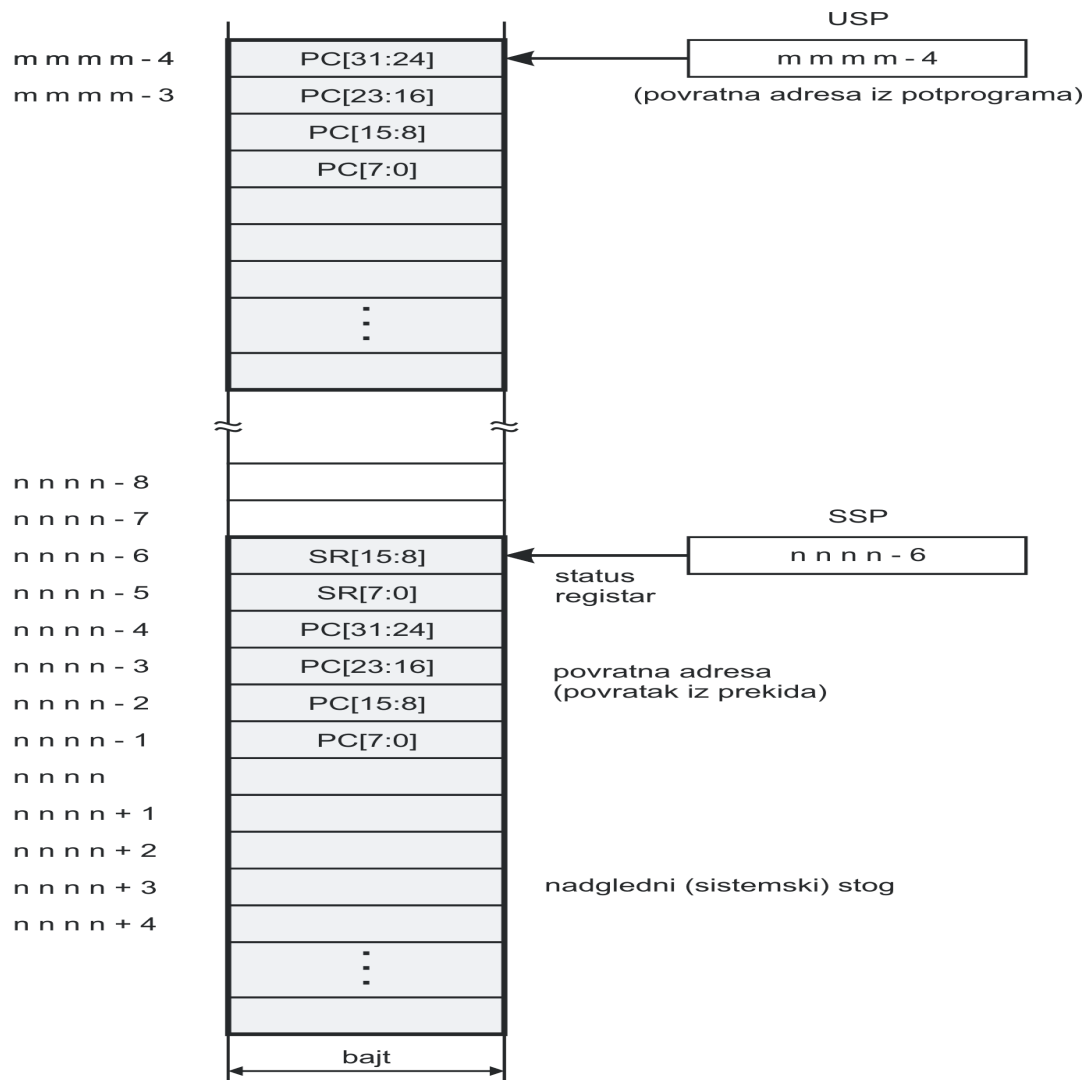


Stanje neposredno
nakon
grananja u potprogram

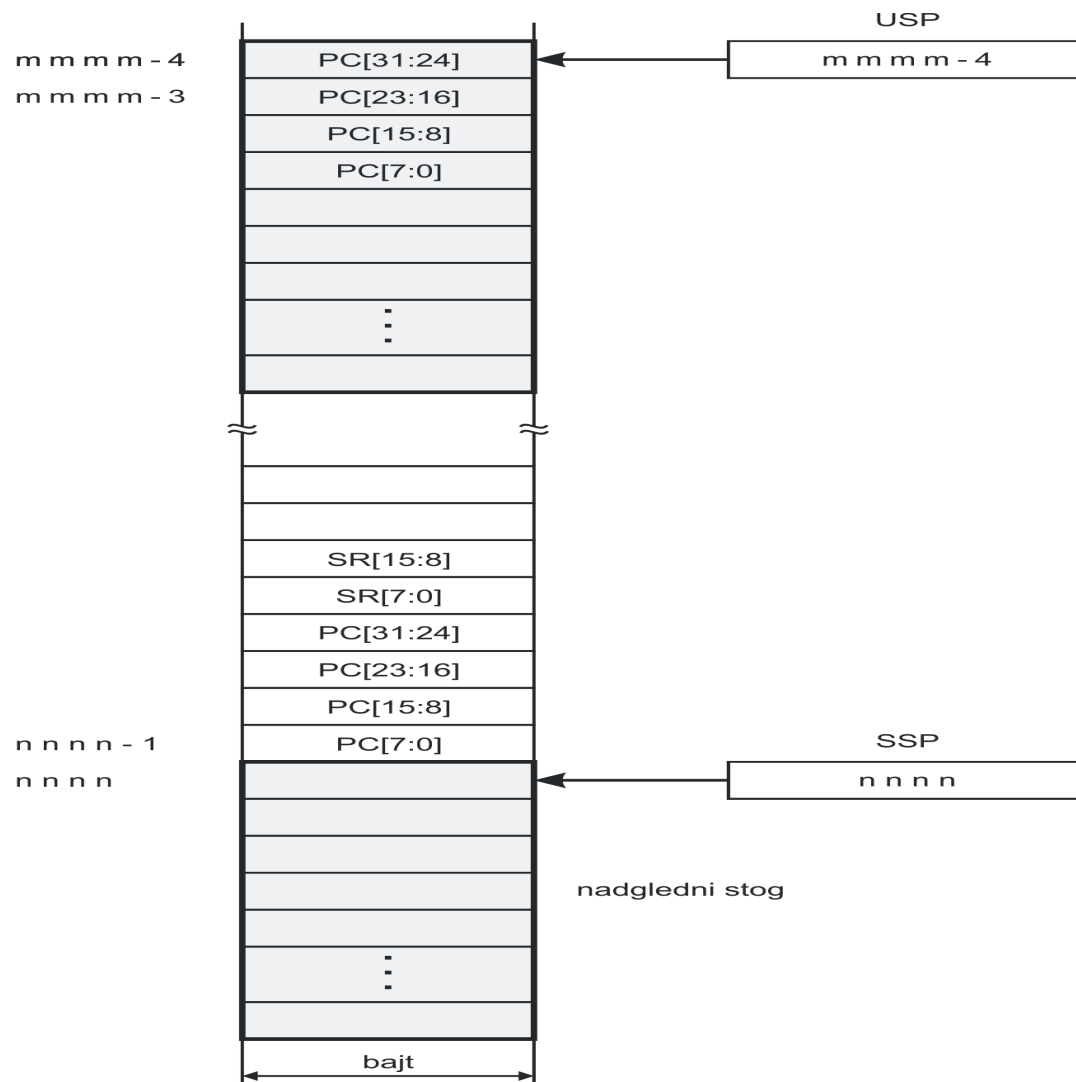


Dogodio se **prekid!**

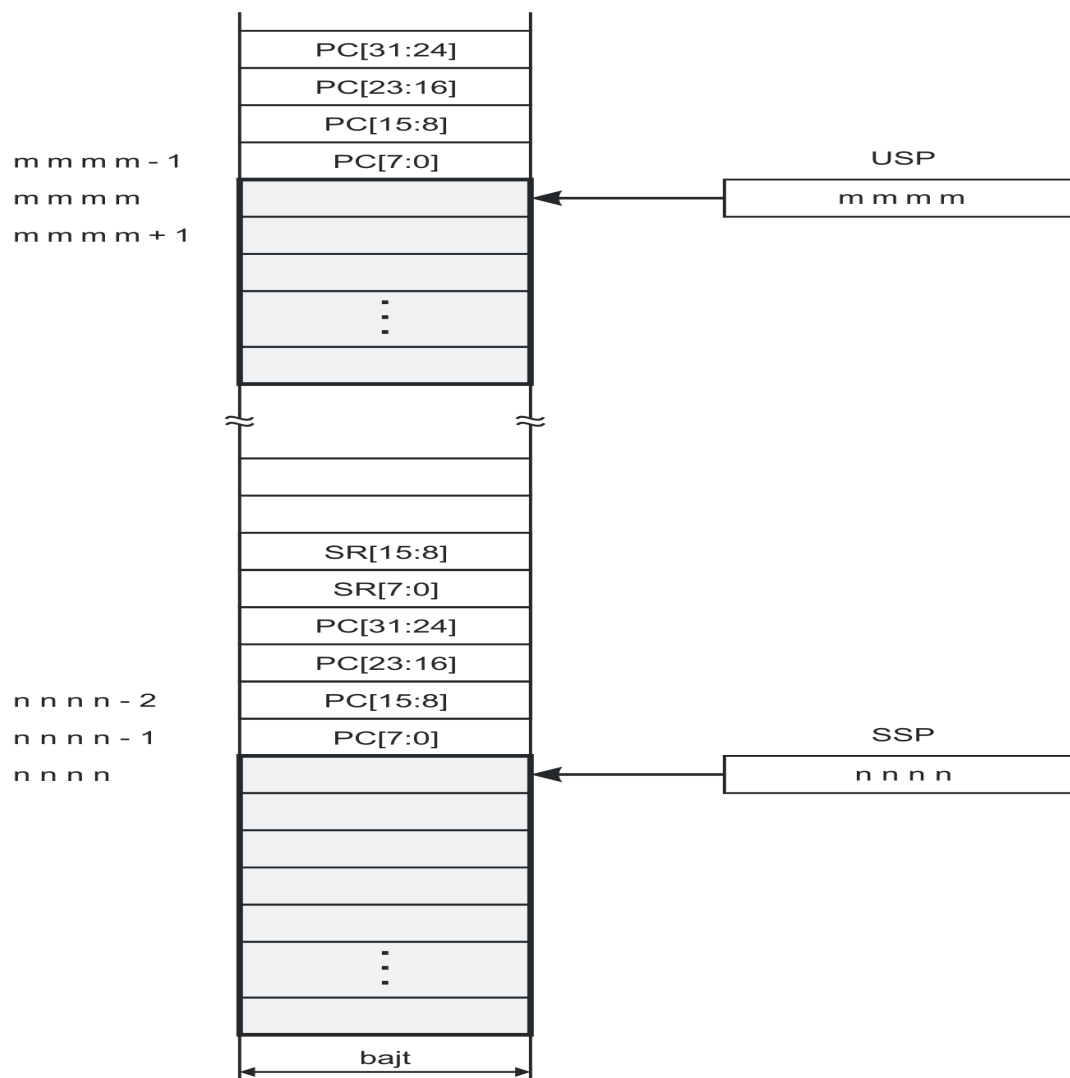
(Iznimka se
obrađuje u
nadglednom načinu)

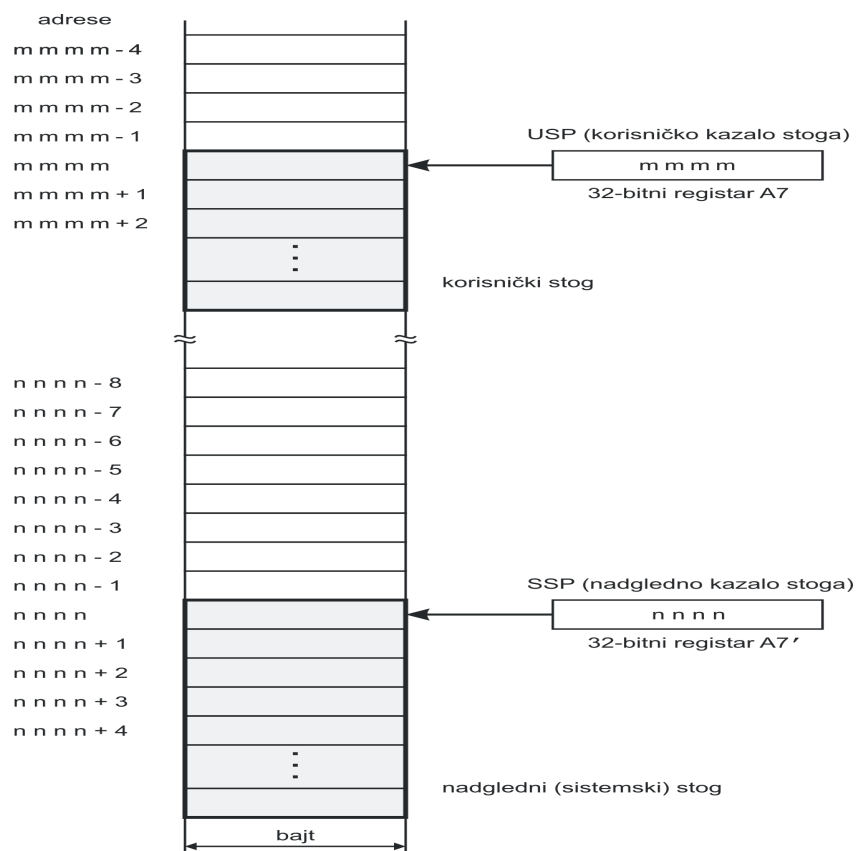


Stanje stogova **nakon**
vraćanja u
potprogram

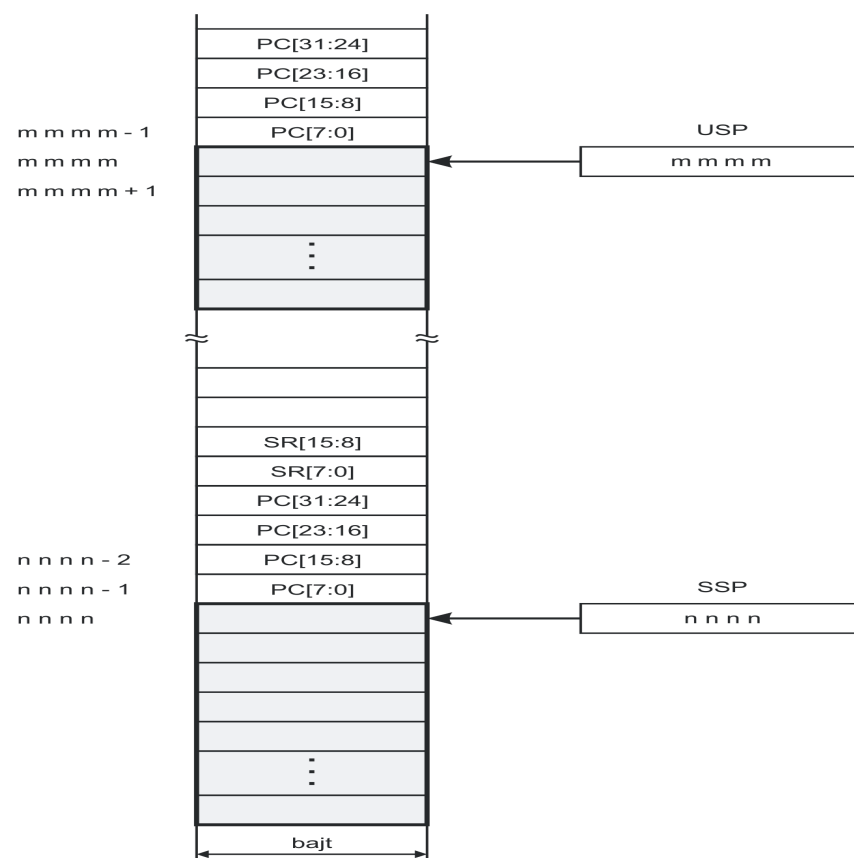


Stanje stoga **nakon**
vraćanja iz
potprograma





Stanje **prije** izvođenja programa



Stanje **nakon** izvođenja programa