4. Veza prema programskoj podršci

4.1. Instrukcije za poziv potprograma
4.2. Rukovanje upravljačkim stogovima
4.3. Obrada procesorskih iznimki

Poziv potprograma:

- mnemonici: `call` (x86), `jsr` (mc68k), `bl` (ARM), `jal` (MIPS)
- memorijski operand određuje adresu prve instrukcije potprograma
- ciljna adresa se zadaje izravno, relativno ili indirektno

Povratak iz potprograma:

- mnemonici: `ret` (x86), `rts` (m68k), `mov` (ARM), `jr` (MIPS)
- izvođenje se nastavlja nakon odgovarajuće pozivne instrukcije

Faza IZVRŠI instrukcije `call x` (bez obzira na arhitekturu):

1. PC → S
2. X → PC

Lokacija S sadrži povratnu adresu

Faza IZVRŠI instrukcije `ret`:

1. S → PC

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.!

Upravljački stog (call stack)**:**

- dinamička podatkovna struktura, pristup na principu LIFO
- svaki tok izvođenja (dretva) ima dedicirani stog u memoriji
- korisničke stogove koristimo za pohranu: i) povratnih adresa, ii) parametara potprograma iii) lokalnih varijabli, ...
- omogućava rekurzivne pozive!
- temeljne instrukcije koje koriste upravljački stog:

```
call sub →
    push PC
    PC ← #sub

ret →
    pop PC
```

Primjer rekurzivnog pozivanja:

glavni program

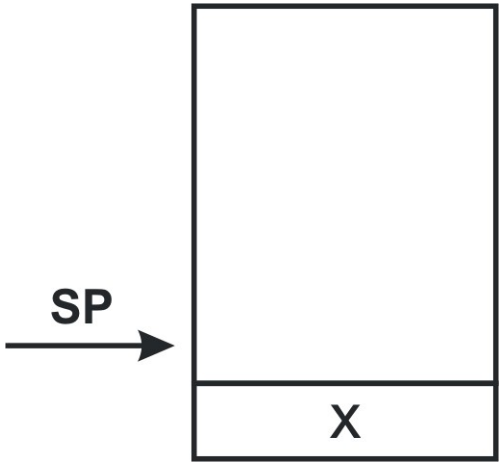
```
                .
                .
            call sub
X:          ; X je povratna adresa
                .
```

potprogram

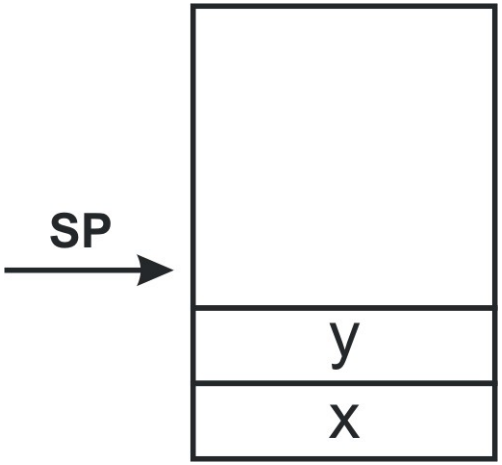
```
        sub:
                .
                .
            call sub
Y:          ; Y – povratna adresa
                .
                .
            ret
```

Stanje stoga (pretp. SP pokazuje na prvu slobodnu lokaciju):

Nakon prvog pozivanja



Nakon drugog pozivanja



Treće pozivanje, četvrto pozivanje, ...

Povratak???

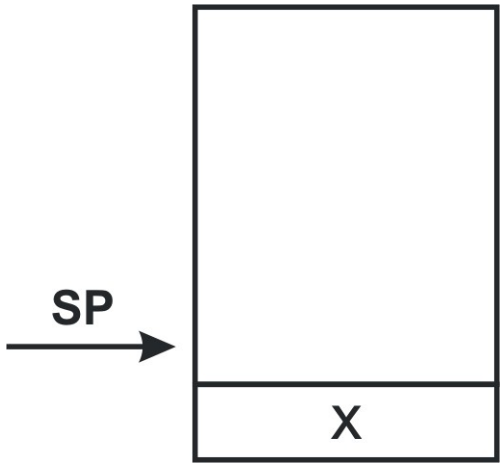
Zadatak: skicirati stanje upravljačkog stoga tijekom izvođenja sljedećeg rekurzivnog programa (pretp. SP pokazuje na prvu slobodnu lokaciju):

```
glavni program:
        N := 2
       call sub
    X:
```

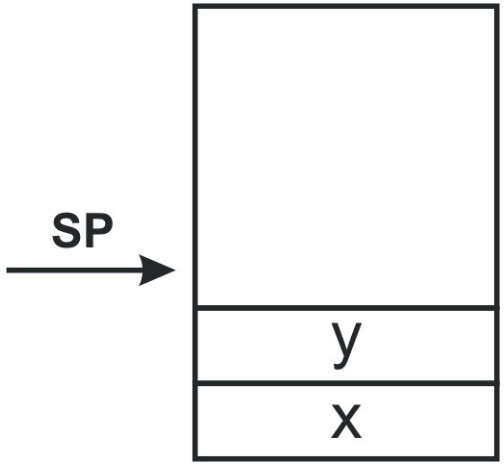
```
potprogram:
      sub:
            .
            .
       N := N – 1
      if N >= 0:
          call sub
   Y:
            .
            .
           ret
```

Stanje stoga:

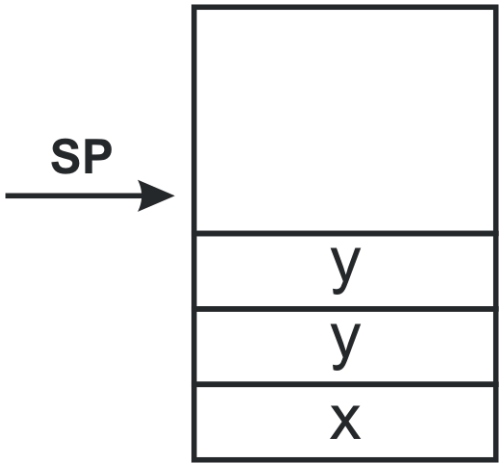
Prvi poziv, N == 2 (glavni program):



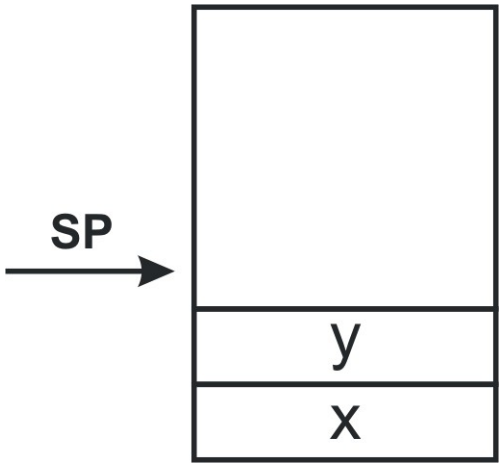
Drugi poziv, N == 1 (potprogram):



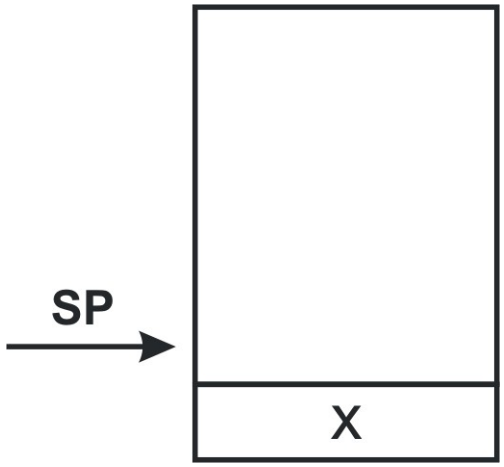
Treći poziv, N = 0 (potprogram)



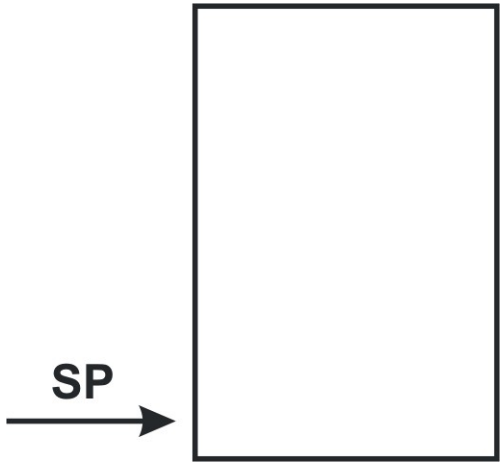
Prvi povratak:



Drugi povratak:



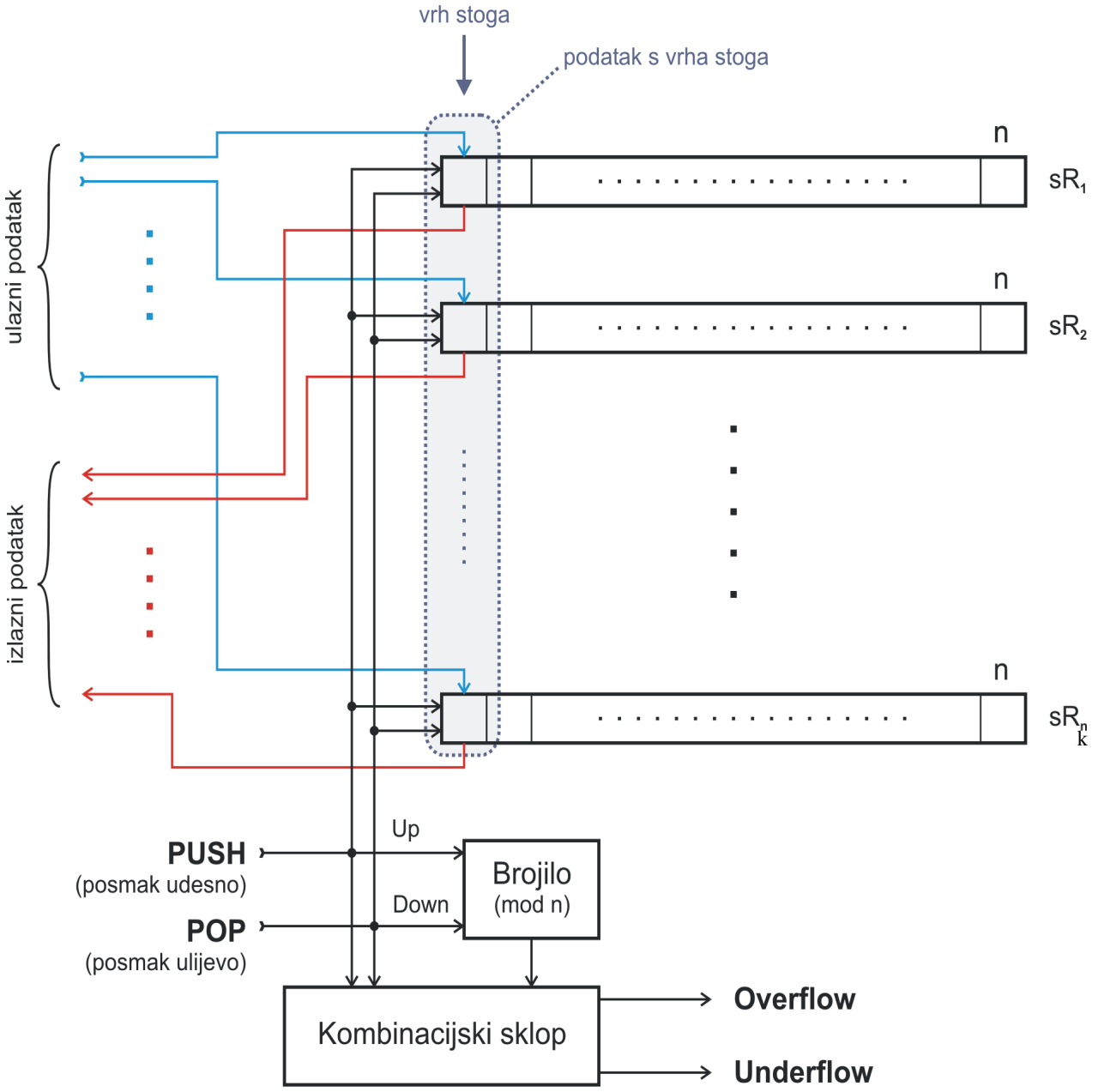
Treći povratak:



Primjeri izvedbe stoga:

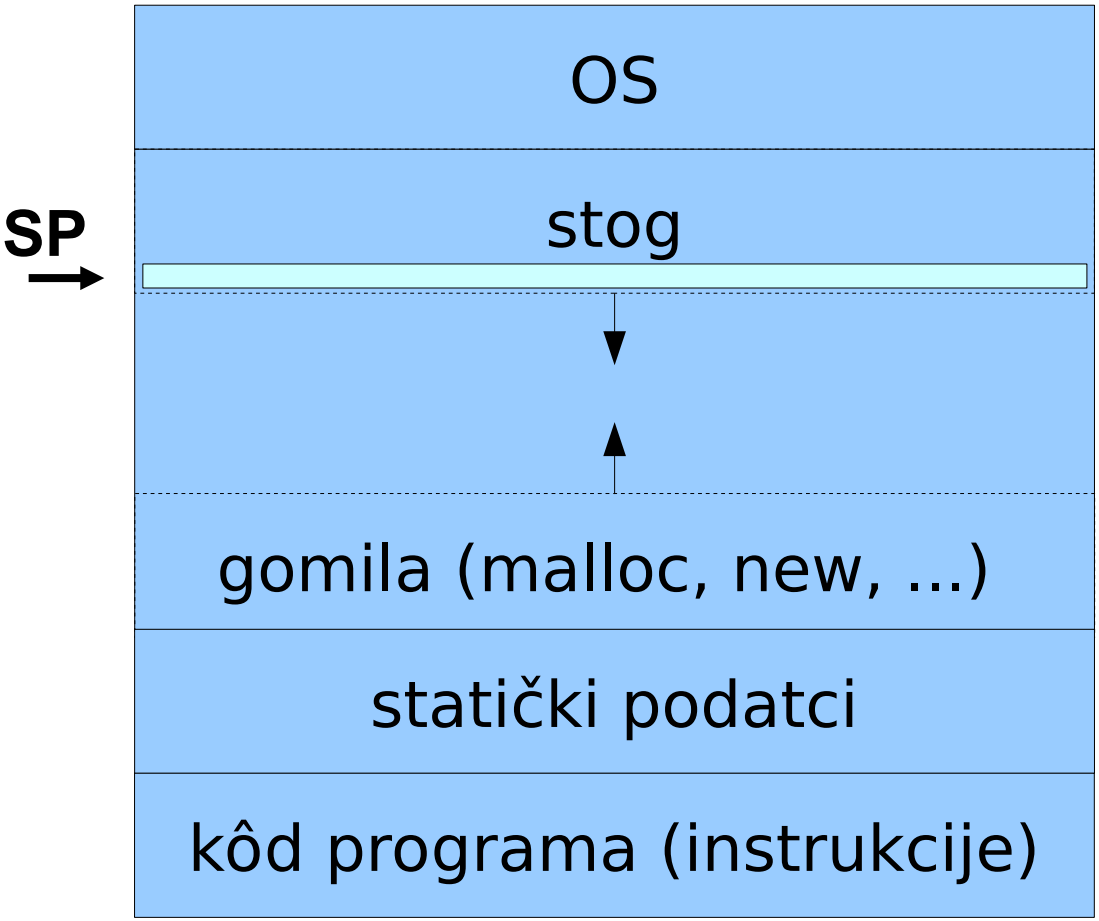
1) sklopovska izvedba temeljena na posmačnim registrima
2) registarska okna (Berkeley RISC, Sun Sparc)
3) rezervirani dio memorije s izravnim pristupom
   - obično se koristi gornji dio podatkovnog segmenta procesa
   - stog obično raste prema dolje
   - dvije mogućnosti:
     1) implicitno korištenje stoga namjenskim instrukcijama
        x86: `call` ... `ret`, m68k: `jsr` ... `rts`
     2) eksplicitno pohranjivanje link registra
        MIPS: jal+`mov` ... `jr`, ARM: `bl`+`mov` ... `mov`

Sklopovska izvedba stoga dubine *n* i duljine riječi *k* bita:



Izvedba stoga u radnoj memoriji

Raspored korištenja memorije procesa (Linux):



Postoje brojne varijacije:
- gomila gore, stog dolje, kôd u sredini
- kôd gore, stog i gomila dolje
- ...

Tipično stog raste prema dolje, a SP pokazuje na posljednju "punu" memorijsku lokaciju odnosno podatak na "vrhu" stoga

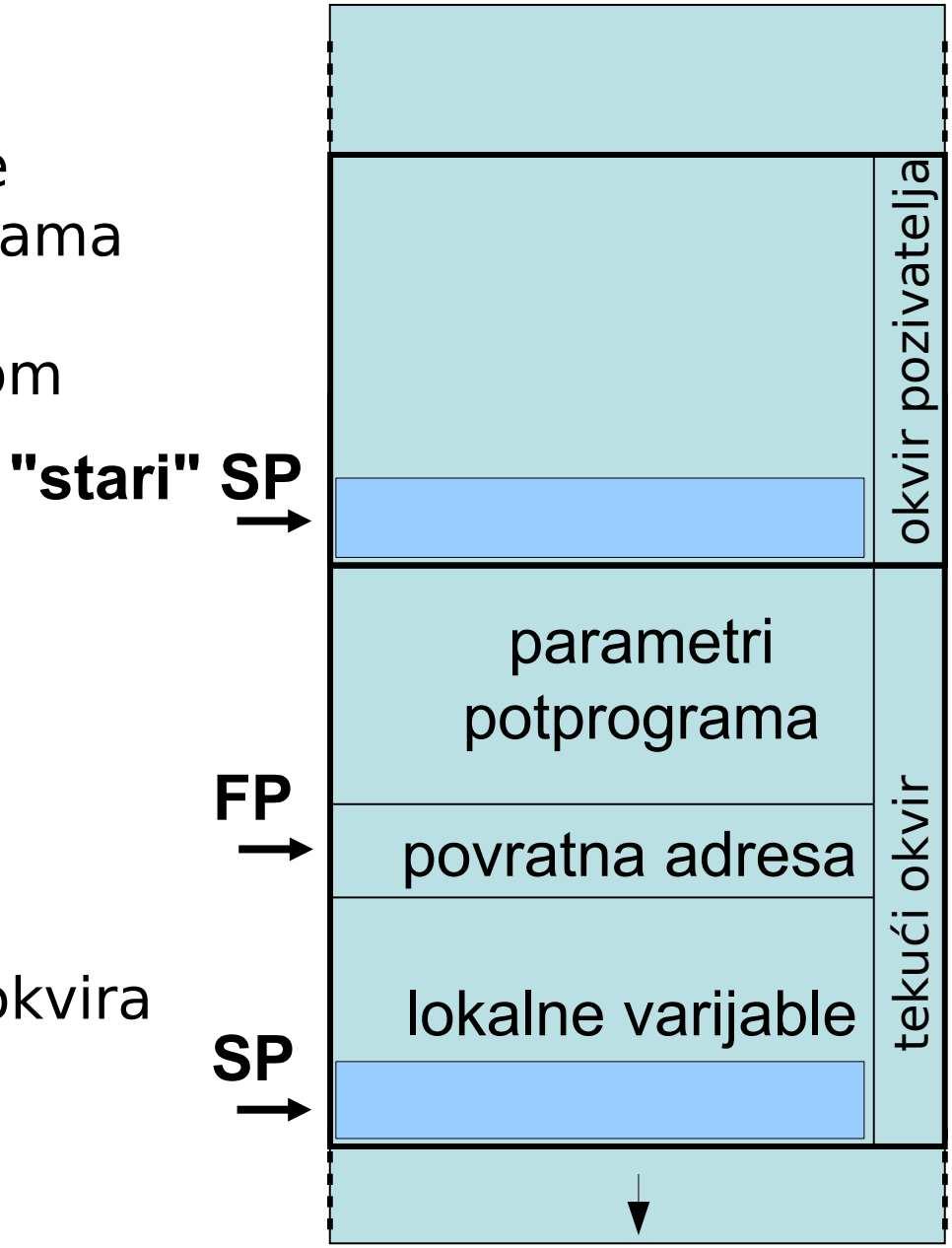
Korištenje stoga u višim programskim jezicima

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:

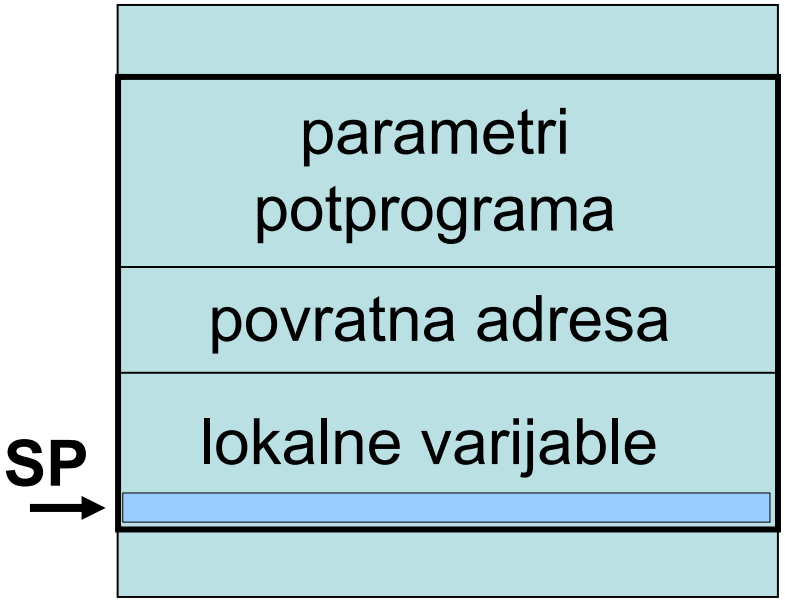


Rukovanje podatcima na stogu

```
// procedure.c
int proc(int i, int j){
   int k, l, m;
      ...
      }
int main(){
  proc(4,5);
      }
```

gcc -S -fomit-frame-pointer
-mpreferred-stack-boundary=2
-masm=intel procedure.c

```
// procedure.s (x86)
          ...
 # poziv potprograma
    sub     esp, 8
mov    DWORD PTR [esp+4], 5
 mov    DWORD PTR [esp], 4
       call    proc
      add     esp, 8
          ...
 # tijelo potprograma
          proc:
    sub     esp, 12
          ...
    add     esp, 12
          ret
```



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,...)

Obrada iznimki za MC 68000

Tipični koraci obrade:

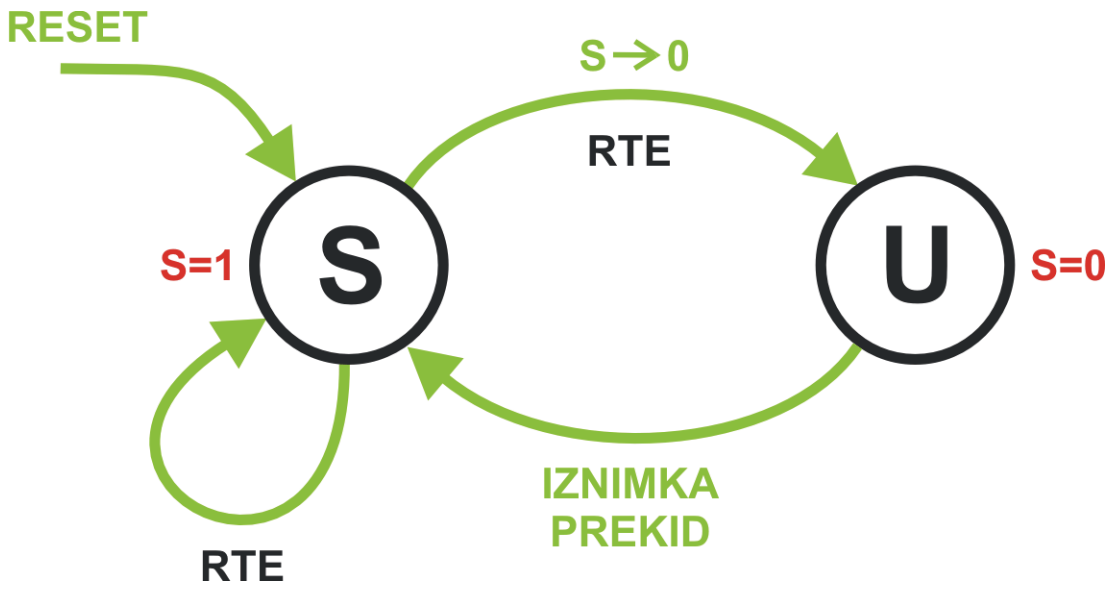
1. procesor prelazi u nadgledni način rada i utvrđuje razlog iznimke

2. ako iznimka dopušta oporavak:
   minimalni kontekst (PC, SR) se sprema na nadgledni stog

3. procesor nastavlja rad izvođenjem potprograma za obradu iznimke

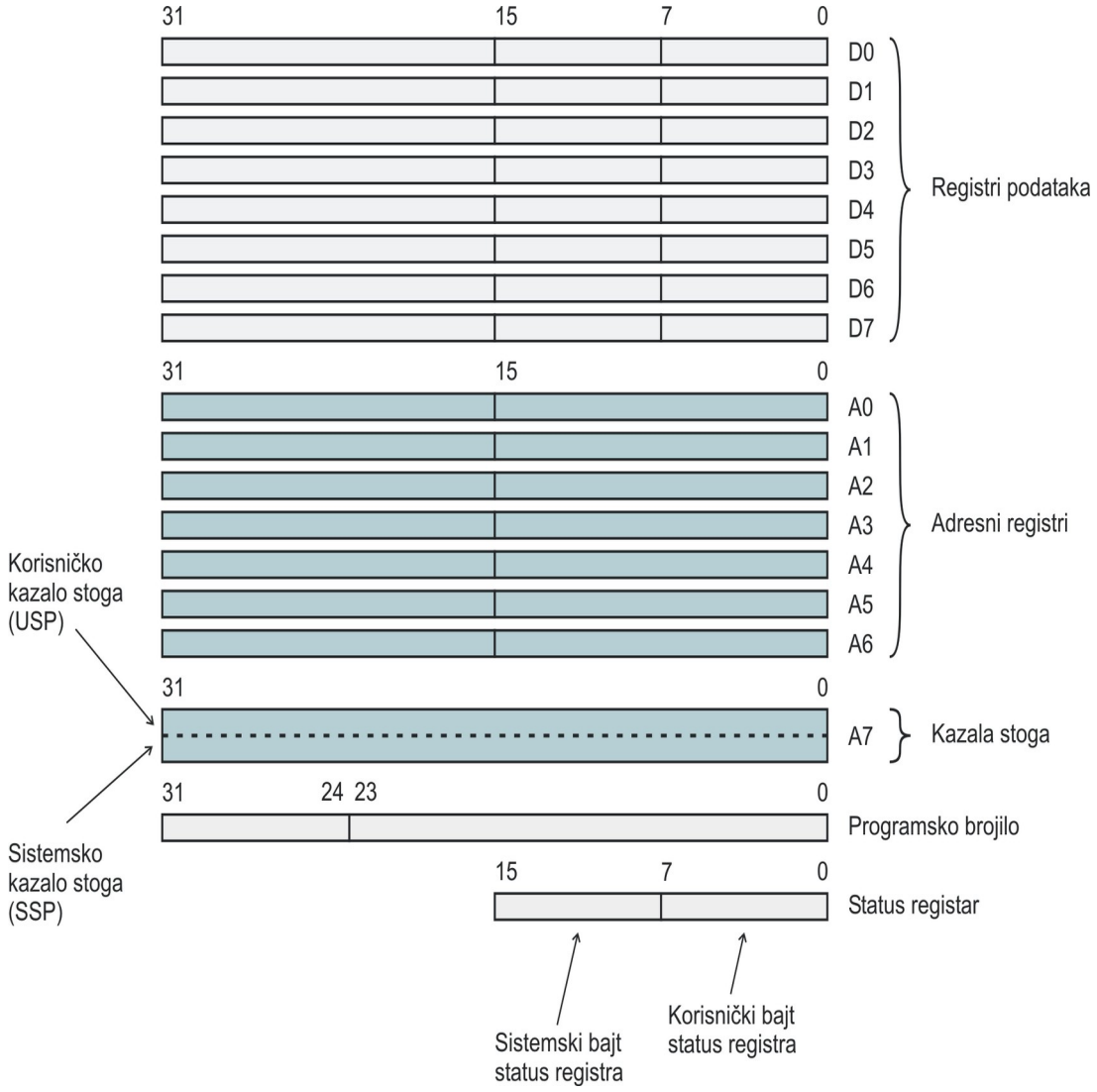
4. instrukcija RTE signalizira povratak u prekinuti program
   minimalni kontekst (PC, SR) se preuzima s nadglednog stoga

Dijagram stanja za MC 68000



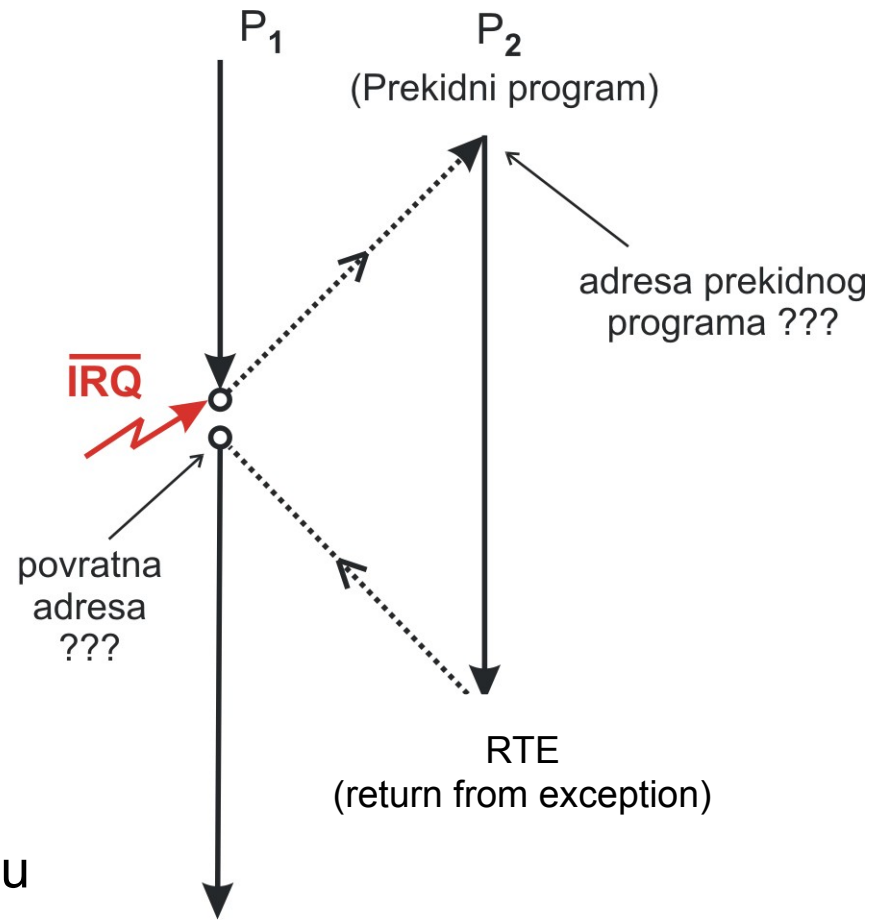
RTE -Return from Exception /povlaštena instrukcija/

Registri programskog modela MC 68000



Tijekom prijenosa upravljanja
s prekinutog na prekidni program
($P_1 \rightarrow P_2$) na stog se pohranjuje
MINIMALNI KONTEKST:

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



Adresa prekidnog programa dobiva se prozivanjem tablice u dediciranom dijelu radne memorije pri čemu je indeks određen vrstom iznimke

Primjer uporabe stoga

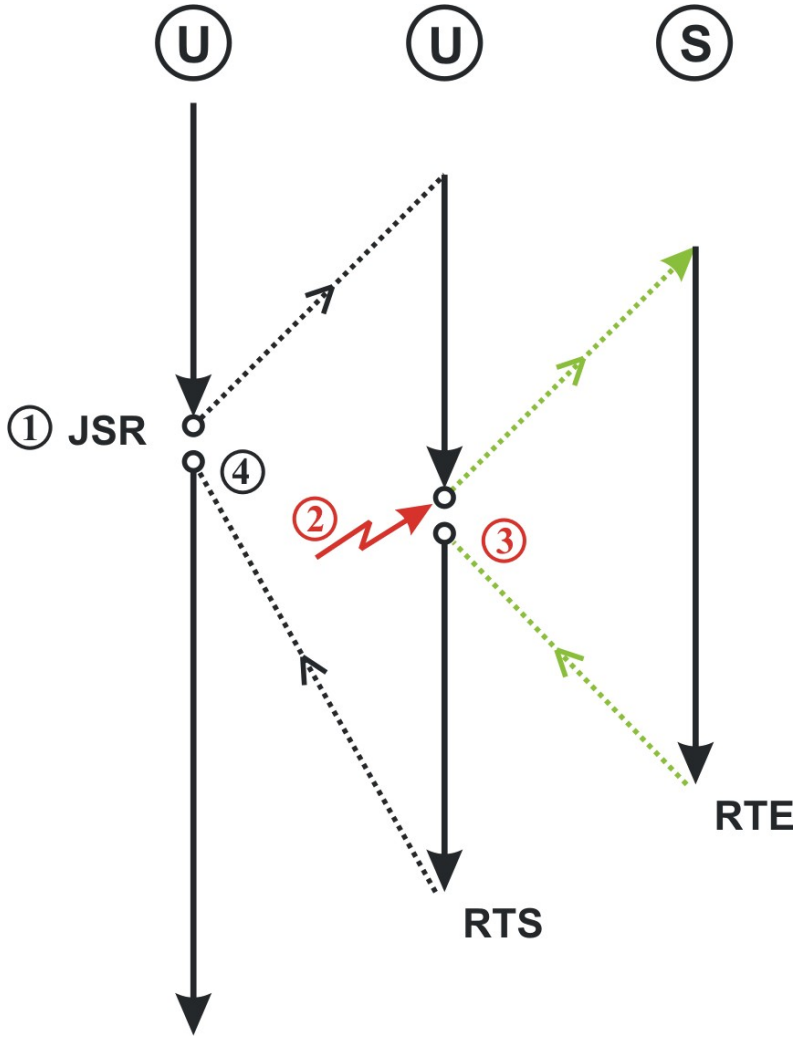
Analiza slučaja: MC 68000

Scenarij:

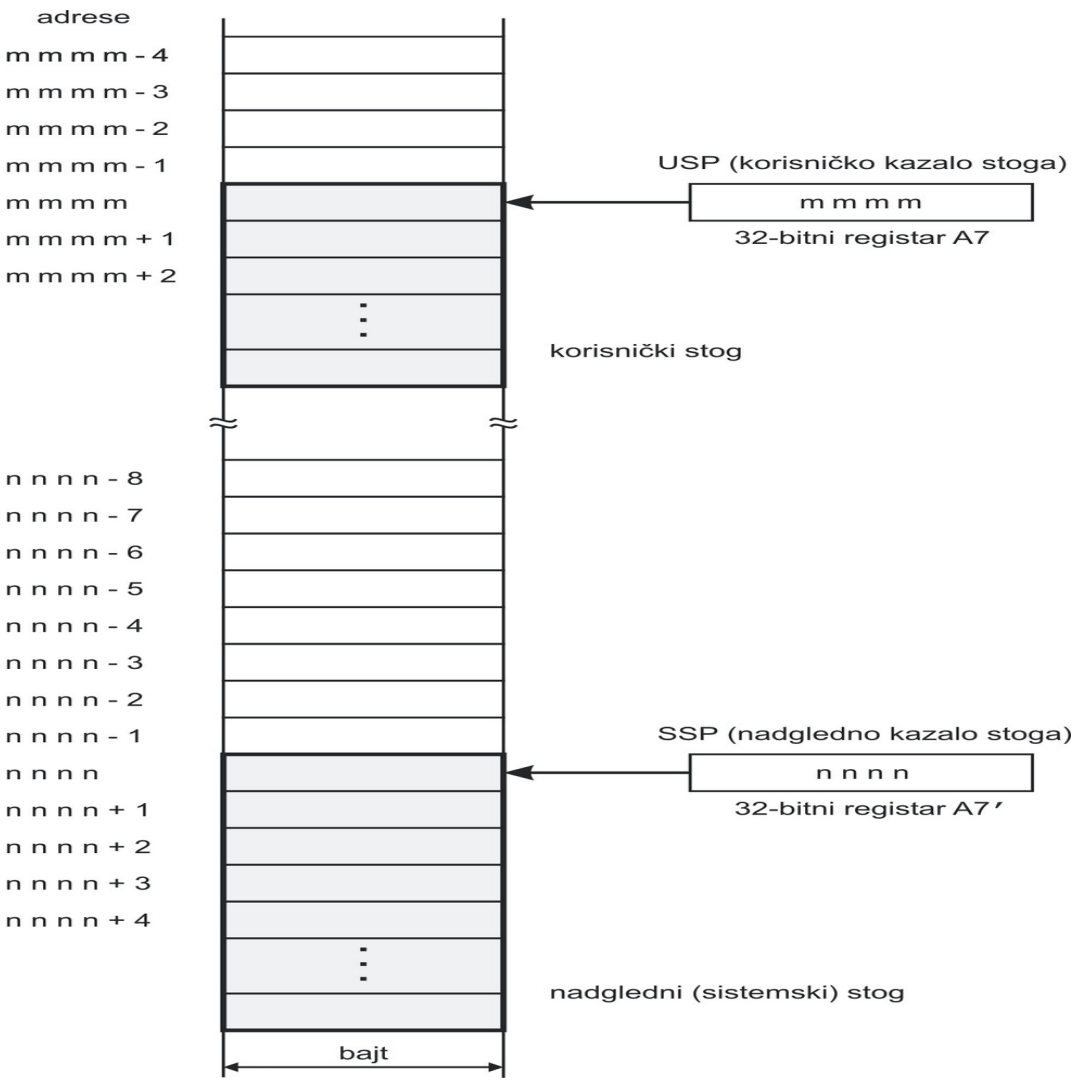
1. Procesor je u korisničkom načinu rada (User Mode)
   - Poziva se potprogram
   - Nastavlja se izvođenje potprograma
2.  Dogodila se iznimka (PREKID)
   - Obrada prekida
3. Vraćanje u potprogram
4. Vraćanje iz potprograma

literatura: S. Ribarić, Naprednije arhitekture mikroprocesora

Grafički prikaz scenarija:

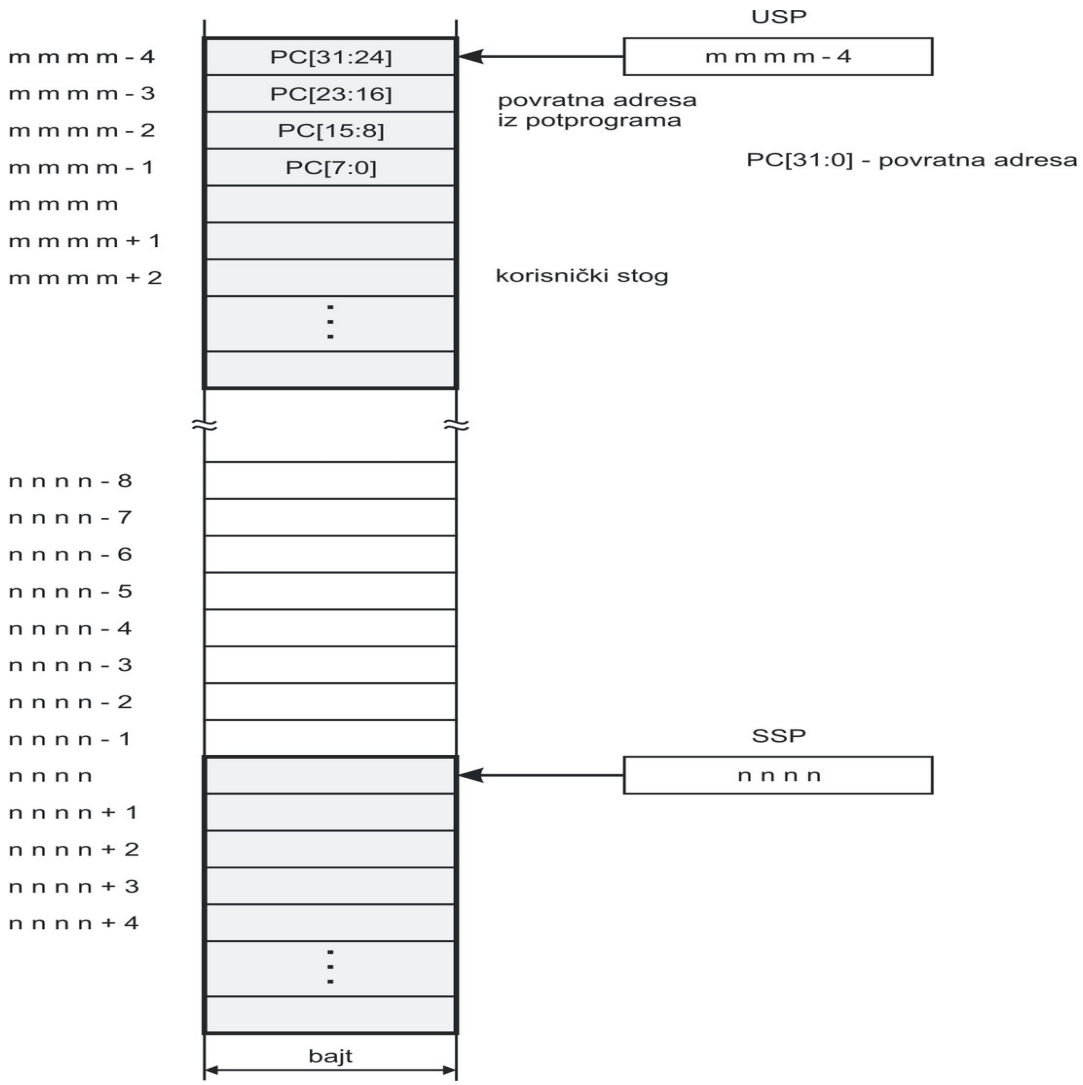


Stanja kazala stoga i stogova prije pozivanja potprograma



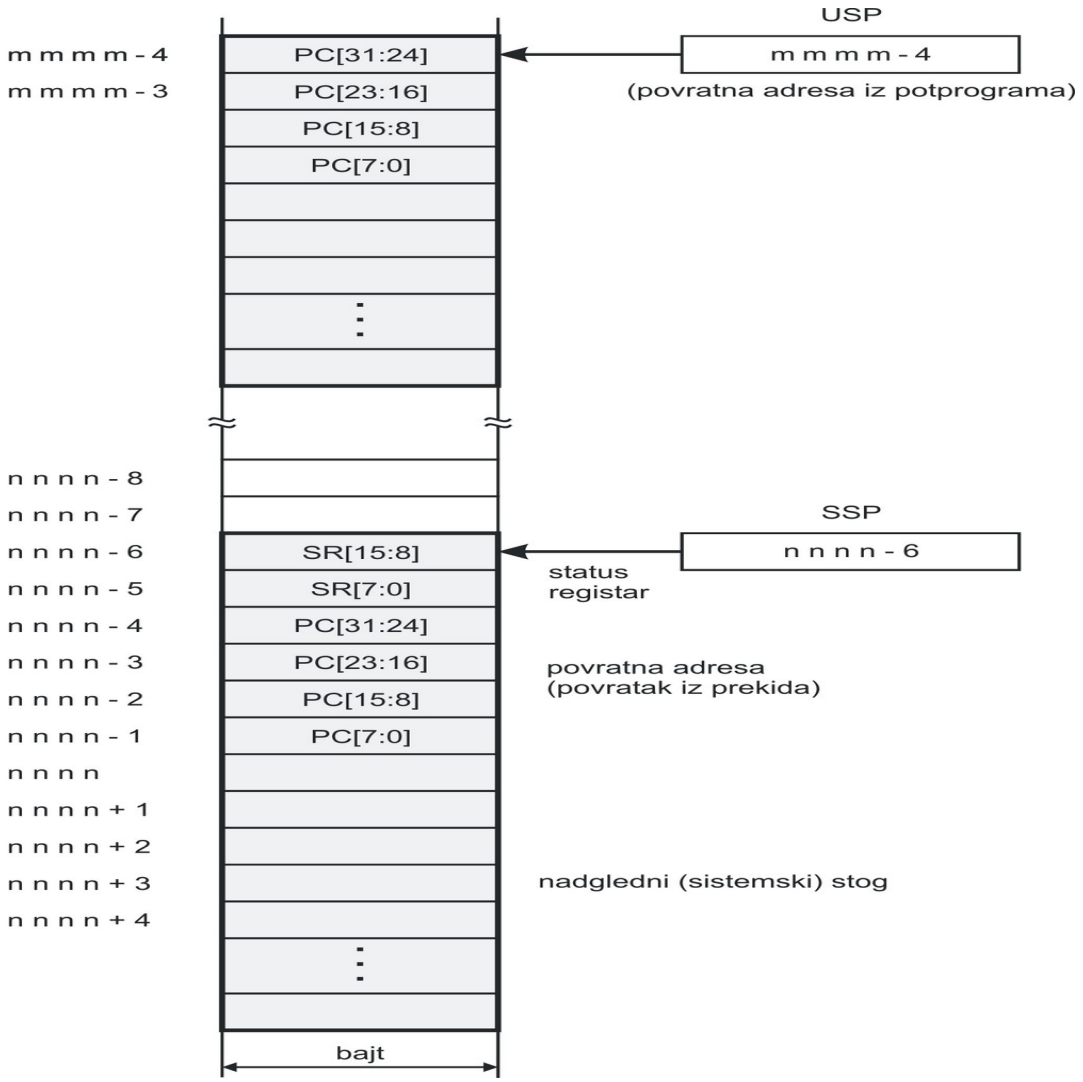
Stanje neposredno
nakon

grananja u potprogram

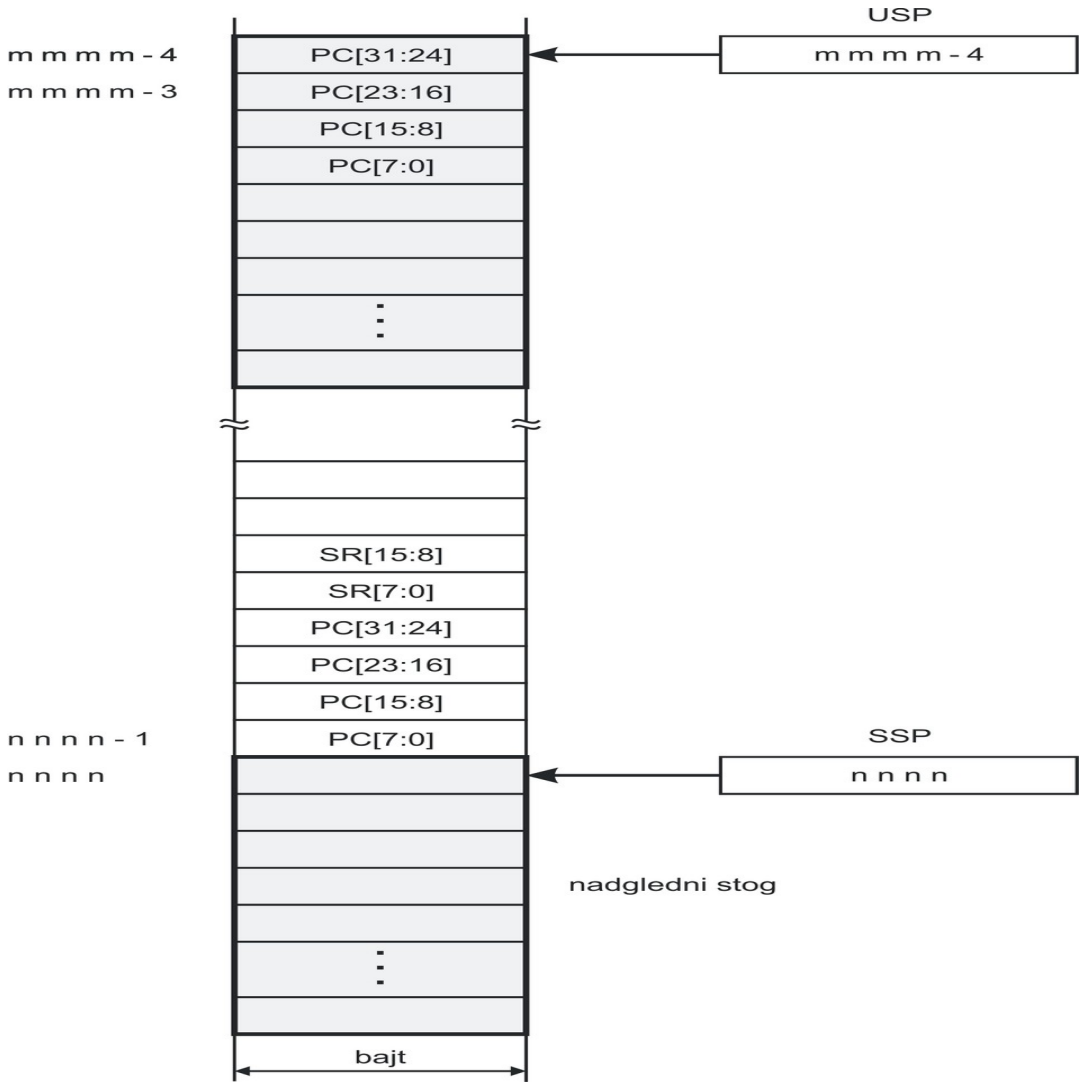


Dogodio se prekid!

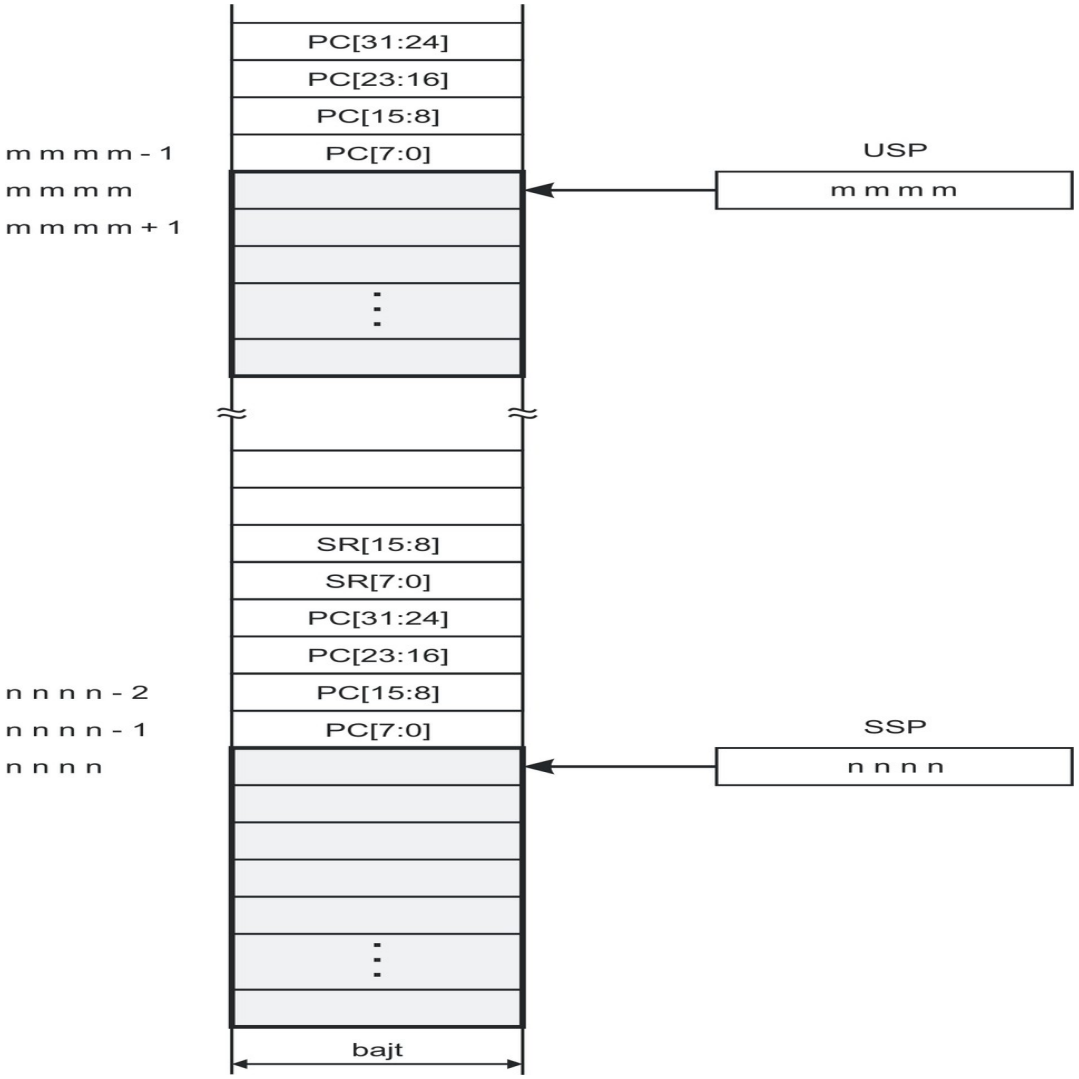
(Iznimka se obrađuje u nadglednom načinu)

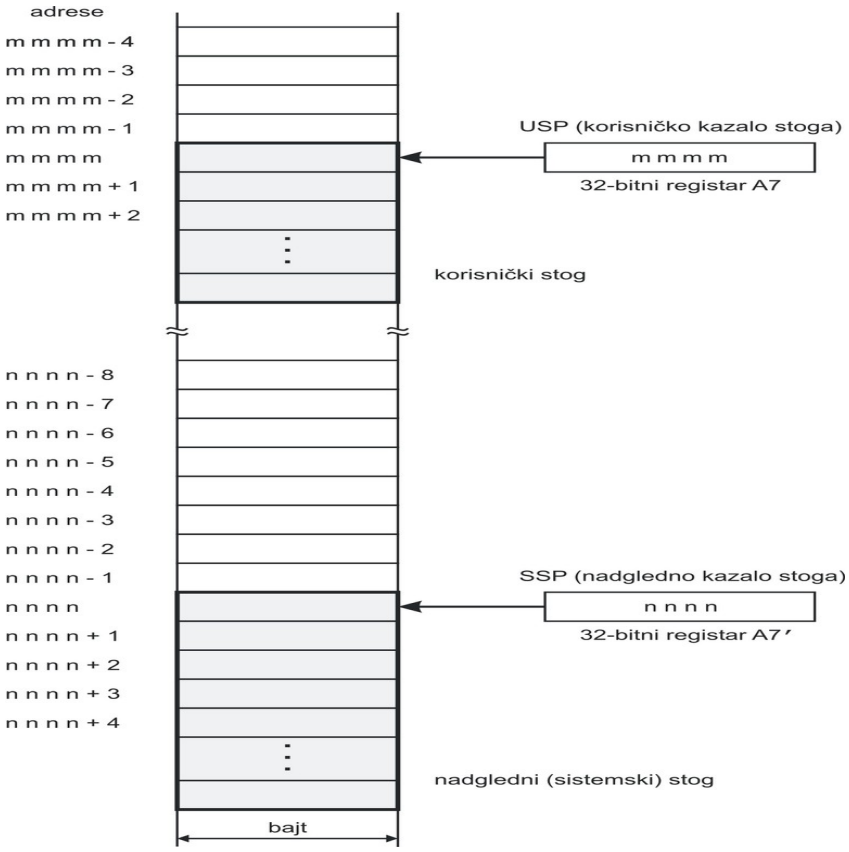


Stanje stogova nakon vraćanja u potprogram

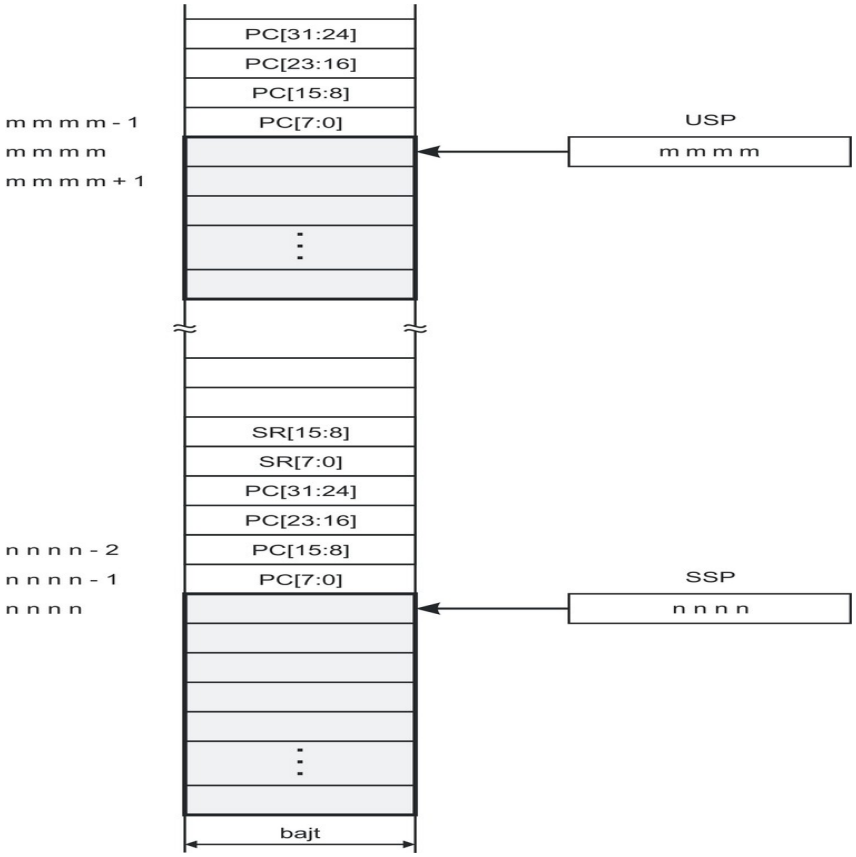


Stanje stoga nakon vraćanja iz potprograma





Stanje prije izvođenja programa



Stanje nakon izvođenja programa