Sklad in delo s podprogrami

Patricio Bulić, Damjan Šonc, Andrej Štrancar

Univerza v Ljubljani, FRI

ARS

Vsebina

Sklad

- Sklad: podatkovna struktura (linearni seznam), organizirana v smislu LIFO.
- Za delo s skladom potrebujemo **skladovni kazalec** (SP) : kazalec, ki kaže na vrh sklada.
- Za delo s skladom uporabljamo dve operaciji: PUSH in POP.
- Več možnih načinov implementacije sklada in operacij PUSH/POP.
- Sklad uporabljamo predvsem za shranjevanje začasnih spremenljivk (npr. lokalnih v podprogramih) za prenos parametrov v podprograme, za shranjevanje registrov v podprogramih in za shranjevanje povratnega naslova pri klicu podprogramov.

Sklad

- Sklad: podatkovna struktura (linearni seznam), organizirana v smislu LIFO.
- Za delo s skladom potrebujemo **skladovni kazalec** (SP) : kazalec, ki kaže na vrh sklada.
- Za delo s skladom uporabljamo dve operaciji: PUSH in POP.
- Več možnih načinov implementacije sklada in operacij PUSH/POP.
- Sklad uporabljamo predvsem za shranjevanje začasnih spremenljivk (npr. lokalnih v podprogramih) za prenos parametrov v podprograme, za shranjevanje registrov v podprogramih in za shranjevanje povratnega naslova pri klicu podprogramov.

Sklad

- Sklad: podatkovna struktura (linearni seznam), organizirana v smislu LIFO.
- Za delo s skladom potrebujemo **skladovni kazalec** (SP) : kazalec, ki kaže na vrh sklada.
- Za delo s skladom uporabljamo dve operaciji: PUSH in POP.
- Več možnih načinov implementacije sklada in operacij PUSH/POP.
- Sklad uporabljamo predvsem za shranjevanje začasnih spremenljivk (npr. lokalnih v podprogramih) za prenos parametrov v podprograme, za shranjevanje registrov v podprogramih in za shranjevanje povratnega naslova pri klicu podprogramov.

Sklad

- Sklad: podatkovna struktura (linearni seznam), organizirana v smislu LIFO.
- Za delo s skladom potrebujemo **skladovni kazalec** (SP) : kazalec, ki kaže na vrh sklada.
- Za delo s skladom uporabljamo dve operaciji: PUSH in POP.
- Več možnih načinov implementacije sklada in operacij PUSH/POP.
- Sklad uporabljamo predvsem za shranjevanje začasnih spremenljivk (npr. lokalnih v podprogramih) za prenos parametrov v podprograme, za shranjevanje registrov v podprogramih in za shranjevanje povratnega naslova pri klicu podprogramov.

Sklad

- Sklad: podatkovna struktura (linearni seznam), organizirana v smislu LIFO.
- Za delo s skladom potrebujemo **skladovni kazalec** (SP) : kazalec, ki kaže na vrh sklada.
- Za delo s skladom uporabljamo dve operaciji: PUSH in POP.
- Več možnih načinov implementacije sklada in operacij PUSH/POP.
- Sklad uporabljamo predvsem za shranjevanje začasnih spremenljivk (npr. lokalnih v podprogramih) za prenos parametrov v podprograme, za shranjevanje registrov v podprogramih in za shranjevanje povratnega naslova pri klicu podprogramov.

Sklad : arhitekturna podpora

- Kaj mora vsebovati arhitektura nekega procesorja, da bo omogočeno delo s skladom?
- Imeti mora register, ki deluje kot skladovni kazalec (SP): lahko je to namenski register ali eden izmed splošnih.
- Pri procesorju, ki operande hrani v registrih moramo imeti podprti operaciji `PUSH reg` in `POP reg`.
- Operaciji `PUSH reg` / `POP reg` sta lahko implementirani kot dva mikroprocesorska ukaza ali kot zaporedje mikroprocesorskih ukazov.

Sklad : arhitekturna podpora

- Kaj mora vsebovati arhitektura nekega procesorja, da bo omogočeno delo s skladom?
- Imeti mora register, ki deluje kot skladovni kazalec (SP): lahko je to namenski register ali eden izmed splošnih.
- Pri procesorju, ki operande hrani v registrih moramo imeti podprti operaciji `PUSH reg` in `POP reg`.
- Operaciji `PUSH reg` / `POP reg` sta lahko implementirani kot dva mikroprocesorska ukaza ali kot zaporedje mikroprocesorskih ukazov.

Sklad : arhitekturna podpora

- Kaj mora vsebovati arhitektura nekega procesorja, da bo omogočeno delo s skladom?
- Imeti mora register, ki deluje kot skladovni kazalec (SP): lahko je to namenski register ali eden izmed splošnih.
- Pri procesorju, ki operande hrani v registrih moramo imeti podprti operaciji `PUSH reg` in `POP reg`.
- Operaciji `PUSH reg` / `POP reg` sta lahko implementirani kot dva mikroprocesorska ukaza ali kot zaporedje mikroprocesorskih ukazov.

Sklad : arhitekturna podpora

- Kaj mora vsebovati arhitektura nekega procesorja, da bo omogočeno delo s skladom?
- Imeti mora register, ki deluje kot skladovni kazalec (SP): lahko je to namenski register ali eden izmed splošnih.
- Pri procesorju, ki operande hrani v registrih moramo imeti podprti operaciji `PUSH reg` in `POP reg`.
- Operaciji `PUSH reg` / `POP reg` sta lahko implementirani kot dva mikroprocesorska ukaza ali kot zaporedje mikroprocesorskih ukazov.

Sklad : različica 1

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP + n;
  ```

  POP reg :

  ```
  SP <- SP - n; reg <- M[SP];
  ```

Sklad : različica 1

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP + n;
  ```

  POP reg :

  ```
  SP <- SP - n; reg <- M[SP];
  ```

Sklad : različica 1

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

PUSH reg :

```
M[SP] <- reg; SP <- SP + n;
```

POP reg :

```
SP <- SP - n; reg <- M[SP];
```

Sklad : različica 1

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP + n;
  ```

  POP reg :

  ```
  SP <- SP - n; reg <- M[SP];
  ```

Sklad : različica 1

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP + n;
  ```

  POP reg :

  ```
  SP <- SP - n; reg <- M[SP];
  ```

Sklad : različica 2

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP + n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP - n;
  ```

Sklad : različica 2

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP + n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP - n;
  ```

Sklad : različica 2

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

PUSH reg :

```
SP <- SP + n; M[SP] <- reg;
```

POP reg :

```
reg <- M[SP]; SP <- SP - n;
```

Sklad : različica 2

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP + n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP - n;
  ```

Sklad : različica 2

- Sklad naj narašča v smeri naraščajočih naslovov in SP naj kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP + n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP - n;
  ```

Sklad : različica 3

- Sklad naj narašča v smeri padajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP - n;
  ```

  POP reg :

  ```
  SP <- SP + n; reg <- M[SP];
  ```

Sklad : različica 3

- Sklad naj narašča v smeri padajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP - n;
  ```

  POP reg :

  ```
  SP <- SP + n; reg <- M[SP];
  ```

Sklad : različica 3

- Sklad naj narašča v smeri padajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

PUSH reg :

```
M[SP] <- reg; SP <- SP - n;
```

POP reg :

```
SP <- SP + n; reg <- M[SP];
```

Sklad : različica 3

- Sklad naj narašča v smeri padajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP - n;
  ```

  POP reg :

  ```
  SP <- SP + n; reg <- M[SP];
  ```

Sklad : različica 3

- Sklad naj narašča v smeri padajočih naslovov in SP naj kaže na prvo prosto mesto na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  M[SP] <- reg; SP <- SP - n;
  ```

  POP reg :

  ```
  SP <- SP + n; reg <- M[SP];
  ```

Sklad : različica 4

- Sklad naj naračča v smeri padajočih naslovov in SP kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP - n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP + n;
  ```

Sklad : različica 4

- Sklad naj naraščа v smeri padajočih naslovov in SP kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP - n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP + n;
  ```

Sklad : različica 4

- Sklad naj naraščа v smeri padajočih naslovov in SP kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

PUSH reg :

```
SP <- SP - n; M[SP] <- reg;
```

POP reg :

```
reg <- M[SP]; SP <- SP + n;
```

Sklad : različica 4

- Sklad naj naraščča v smeri padajočih naslovov in SP kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP - n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP + n;
  ```

Sklad : različica 4

- Sklad naj naraščča v smeri padajočih naslovov in SP kaže na zadnji podatek na skladu.
- Predpostavimo, da registrski operand zaseda n pomnilniških besed.
- Operaciji `PUSH reg` / `POP reg`:

  PUSH reg :

  ```
  SP <- SP - n; M[SP] <- reg;
  ```

  POP reg :

  ```
  reg <- M[SP]; SP <- SP + n;
  ```

HIP - sklad

- HIP nima skladovnega kazalca - zato se moramo dogovoriti, kateri izmed splošnonamenskih registrov naj bo skladovni kazalec. **Dogovor**: naj bo to register `R30`.
- Zaradi zaheteve po poravnanosti pomnilniških operandov, dovolimo prenos samo med registri (32-bitna vsebina) in skladom, tj. le operaciji `PUSH reg` / `POP reg`.
- **Dogovor**: **naj sklad narašča v smeri padajočih naslovov in naj skladovni kazalec kaže na prvo prosto mesto na skladu**.
- Sklad naj se začne na dnu podatkovnega segmenta.

HIP - sklad

- HIP nima skladovnega kazalca - zato se moramo dogovoriti, kateri izmed splošnonamenskih registrov naj bo skladovni kazalec. **Dogovor**: naj bo to register `R30`.
- Zaradi zaheteve po poravnanosti pomnilniških operandov, dovolimo prenos samo med registri (32-bitna vsebina) in skladom, tj. le operaciji `PUSH reg` / `POP reg`.
- **Dogovor**: **naj sklad narašča v smeri padajočih naslovov in naj skladovni kazalec kaže na prvo prosto mesto na skladu**.
- Sklad naj se začne na dnu podatkovnega segmenta.

HIP - sklad

- HIP nima skladovnega kazalca - zato se moramo dogovoriti, kateri izmed splošnonamenskih registrov naj bo skladovni kazalec. **Dogovor**: naj bo to register `R30`.
- Zaradi zaheteve po poravnanosti pomnilniških operandov, dovolimo prenos samo med registri (32-bitna vsebina) in skladom, tj. le operaciji `PUSH reg` / `POP reg`.
- **Dogovor**: **naj sklad narašča v smeri padajočih naslovov in naj skladovni kazalec kaže na prvo prosto mesto na skladu**.
- Sklad naj se začne na dnu podatkovnega segmenta.

HIP - sklad

- HIP nima skladovnega kazalca - zato se moramo dogovoriti, kateri izmed splošnonamenskih registrov naj bo skladovni kazalec. **Dogovor**: naj bo to register `R30`.
- Zaradi zaheteve po poravnanosti pomnilniških operandov, dovolimo prenos samo med registri (32-bitna vsebina) in skladom, tj. le operaciji `PUSH reg` / `POP reg`.
- **Dogovor**: **naj sklad narašča v smeri padajočih naslovov in naj skladovni kazalec kaže na prvo prosto mesto na skladu**.
- Sklad naj se začne na dnu podatkovnega segmenta.

HIP - sklad

- Na začetku vsakega programa **moramo nastaviti skladovni kazalec**:

```
addui r30,r0, #0x4fc ;
```

- HIP nima ukazov PUSH/POP, zato se moramo dogovoriti, kako bomo izvedli sklad in operaciji, tj. makro ukaza PUSH in POP. Makro ukaza PUSH in POP naj imata samo en eksplicitni registrski operand. Izvedemo ju kot zaporedje dveh ukazov procesorja HIP:

PUSH reg :

```
sw 0(r30), reg
subui r30,r30,#4
```

POP reg :

```
addui r30,r30,#4
lw reg, 0(r30)
```

HIP - sklad

- Na začetku vsakega programa **moramo nastaviti skladovni kazalec**:

```
addui r30,r0, #0x4fc ;
```

- HIP nima ukazov PUSH/POP, zato se moramo dogovoriti, kako bomo izvedli sklad in operaciji, tj. makro ukaza PUSH in POP. Makro ukaza PUSH in POP naj imata samo en eksplicitni registrski operand. Izvedemo ju kot zaporedje dveh ukazov procesorja HIP:

PUSH reg :

```
sw 0(r30), reg
subui r30,r30,#4
```

POP reg :

```
addui r30,r30,#4
lw reg, 0(r30)
```

HIP - sklad

- Na začetku vsakega programa **moramo nastaviti skladovni kazalec**:

```
addui r30,r0, #0x4fc ;
```

- HIP nima ukazov PUSH/POP, zato se moramo dogovoriti, kako bomo izvedli sklad in operaciji, tj. makro ukaza PUSH in POP. Makro ukaza PUSH in POP naj imata samo en eksplicitni registrski operand. Izvedemo ju kot zaporedje dveh ukazov procesorja HIP:

PUSH reg :

```
sw 0(r30), reg
subui r30,r30,#4
```

POP reg :

```
addui r30,r30,#4
lw reg, 0(r30)
```

HIP - sklad

- Na začetku vsakega programa **moramo nastaviti skladovni kazalec**:

```
addui r30,r0, #0x4fc ;
```

- HIP nima ukazov PUSH/POP, zato se moramo dogovoriti, kako bomo izvedli sklad in operaciji, tj. makro ukaza PUSH in POP. Makro ukaza PUSH in POP naj imata samo en eksplicitni registrski operand. Izvedemo ju kot zaporedje dveh ukazov procesorja HIP:

PUSH reg :

```
sw 0(r30), reg
subui r30,r30,#4
```

POP reg :

```
addui r30,r30,#4
lw reg, 0(r30)
```

Prenos parametrov in klic podprogramov - kako?

- Kje so parametri (v registrih ali na skladu)?
- Kako in kaj morata storiti klicoči program in klicani podprogram?
- Kdo parametre na koncu pobriše oz. odstrani?
- Kako se vrnemo iz podprograma oz. kje je povratni naslov?
- Kako podprogram vrača vrednosti?
- Kako se ohranja vrednosti v registrih, ki jih je ustvaril klicoči program, med izvajanjem podprograma? Kdo mora te vrednosti shraniti in kam ter kdo in kako jih obnovi?

Prenos parametrov in klic podprogramov - kako?

- Kje so parametri (v registrih ali na skladu)?
- Kako in kaj morata storiti klicoči program in klicani podprogram?
- Kdo parametre na koncu pobriše oz. odstrani?
- Kako se vrnemo iz podprograma oz. kje je povratni naslov?
- Kako podprogram vrača vrednosti?
- Kako se ohranja vrednosti v registrih, ki jih je ustvaril klicoči program, med izvajanjem podprograma? Kdo mora te vrednosti shraniti in kam ter kdo in kako jih obnovi?

Prenos parametrov in klic podprogramov - kako?

- Kje so parametri (v registrih ali na skladu)?
- Kako in kaj morata storiti klicoči program in klicani podprogram?
- Kdo parametre na koncu pobriše oz. odstrani?
- Kako se vrnemo iz podprograma oz. kje je povratni naslov?
- Kako podprogram vrača vrednosti?
- Kako se ohranja vrednosti v registrih, ki jih je ustvaril klicoči program, med izvajanjem podprograma? Kdo mora te vrednosti shraniti in kam ter kdo in kako jih obnovi?

Prenos parametrov in klic podprogramov - kako?

- Kje so parametri (v registrih ali na skladu)?
- Kako in kaj morata storiti klicoči program in klicani podprogram?
- Kdo parametre na koncu pobriše oz. odstrani?
- Kako se vrnemo iz podprograma oz. kje je povratni naslov?
- Kako podprogram vrača vrednosti?
- Kako se ohranja vrednosti v registrih, ki jih je ustvaril klicoči program, med izvajanjem podprograma? Kdo mora te vrednosti shraniti in kam ter kdo in kako jih obnovi?

Prenos parametrov in klic podprogramov - kako?

- Kje so parametri (v registrih ali na skladu)?
- Kako in kaj morata storiti klicoči program in klicani podprogram?
- Kdo parametre na koncu pobriše oz. odstrani?
- Kako se vrnemo iz podprograma oz. kje je povratni naslov?
- Kako podprogram vrača vrednosti?
- Kako se ohranja vrednosti v registrih, ki jih je ustvaril klicoči program, med izvajanjem podprograma? Kdo mora te vrednosti shraniti in kam ter kdo in kako jih obnovi?

Prenos parametrov in klic podprogramov - kako?

- Kje so parametri (v registrih ali na skladu)?
- Kako in kaj morata storiti klicoči program in klicani podprogram?
- Kdo parametre na koncu pobriše oz. odstrani?
- Kako se vrnemo iz podprograma oz. kje je povratni naslov?
- Kako podprogram vrača vrednosti?
- Kako se ohranja vrednosti v registrih, ki jih je ustvaril klicoči program, med izvajanjem podprograma? Kdo mora te vrednosti shraniti in kam ter kdo in kako jih obnovi?

Prenos parametrov in klic podprogramov - procesor HIP - kako?

- Za prenos parametrov bomo uporabljali registra R24 in R25 ter sklad.
- Kako bi izvedli klice podprogramov in prenos parametrov preko sklada na procesorju HIP?
- Sprejeli bomo dogovor o klicu podprogramov in prenosu parametrov (*calling convention*) - **tega se moramo vedno držati!**.

Prenos parametrov in klic podprogramov - procesor HIP - kako?

- Za prenos parametrov bomo uporabljali registra R24 in R25 ter sklad.
- Kako bi izvedli klice podprogramov in prenos parametrov preko sklada na procesorju HIP?
- Sprejeli bomo dogovor o klicu podprogramov in prenosu parametrov (*calling convention*) - **tega se moramo vedno držati!**.

Prenos parametrov in klic podprogramov - procesor HIP - kako?

- Za prenos parametrov bomo uporabljali registra R24 in R25 ter sklad.
- Kako bi izvedli klice podprogramov in prenos parametrov preko sklada na procesorju HIP?
- Sprejeli bomo dogovor o klicu podprogramov in prenosu parametrov (*calling convention*) - **tega se moramo vedno držati!**.

HIP - dogovor o klicu podprogramov

- **Dogovor**: parametri se v podprogram prenašajo od desne proti levi glede na podpis podprograma.
- Prva dva parametra gledano z leve proti desni v podpisu funkcije bo klicoči program prenašal izključno preko registrov R24 in R25, ostale pa izključno preko sklada od desne proti levi.

HIP - dogovor o klicu podprogramov

- **Dogovor**: parametri se v podprogram prenašajo od desne proti levi glede na podpis podprograma.
- Prva dva parametra gledano z leve proti desni v podpisu funkcije bo klicoči program prenašal izključno preko registrov R24 in R25, ostale pa izključno preko sklada od desne proti levi.

HIP - dogovor o klicu podprogramov

- Pri HIP-u za klic podprograma uporabimo ukaz:

```
call rp, odmik(rb)
```

  pri čemer se povratni naslov shrani v register `rp`, v programski števec pa se vpiše naslov podprograma, ki je `rb + odmik`.

- **Dogovor**: naj bo `rp` register `R31`.
- **Dogovor**: za kratke klice naj bo `rb` register `R0`:

```
call r31, PODPROG(r0)
```

- **Dogovor**: za dolge klice naj bo `rb` register `R27`:

```
lhi   r27, #PODPROG
addui r27, r27, #PODPROG
call  r31, 0(r27)
```

HIP - dogovor o klicu podprogramov

- Pri HIP-u za klic podprograma uporabimo ukaz:

```
call rp, odmik(rb)
```

  pri čemer se povratni naslov shrani v register `rp`, v programski števec pa se vpiše naslov podprograma, ki je `rb + odmik`.

- **Dogovor**: naj bo `rp` register `R31`.
- **Dogovor**: za kratke klice naj bo `rb` register `R0`:

```
call r31, PODPROG(r0)
```

- **Dogovor**: za dolge klice naj bo `rb` register `R27`:

```
lhi   r27, #PODPROG
addui r27, r27, #PODPROG
call  r31, 0(r27)
```

HIP - dogovor o klicu podprogramov

- Pri HIP-u za klic podprograma uporabimo ukaz:

```
call rp, odmik(rb)
```

  pri čemer se povratni naslov shrani v register `rp`, v programski števec pa se vpiše naslov podprograma, ki je `rb + odmik`.

- **Dogovor**: naj bo `rp` register `R31`.
- **Dogovor**: za kratke klice naj bo `rb` register `R0`:

```
call r31, PODPROG(r0)
```

- **Dogovor**: za dolge klice naj bo `rb` register `R27`:

```
lhi   r27, #PODPROG
addui r27, r27, #PODPROG
call  r31, 0(r27)
```

HIP - dogovor o klicu podprogramov

- Pri HIP-u za klic podprograma uporabimo ukaz:

```
call rp, odmik(rb)
```

  pri čemer se povratni naslov shrani v register `rp`, v programski števec pa se vpiše naslov podprograma, ki je `rb + odmik`.

- **Dogovor**: naj bo `rp` register `R31`.
- **Dogovor**: za kratke klice naj bo `rb` register `R0`:

```
call r31, PODPROG(r0)
```

- **Dogovor**: za dolge klice naj bo `rb` register `R27`:

```
lhi   r27, #PODPROG
addui r27, r27, #PODPROG
call  r31, 0(r27)
```

HIP - dogovor o klicu podprogramov

- Splošen zapis ukaza za brezpogojni skok je:

```
j odmik(rb)
```

- **Dogovor**: Ker je po dogovoru povratni naslov shranjen v registru R31, naj bo `rb` za vrnitev iz podprograma register `R31`:

```
j 0(r31)
```

- **Dogovor**: za dolge skoke naj bo `rb` register `R26`.

```
lhi   r26, #SKOK
addui r26, r26, #SKOK
j     0(r26)
```

HIP - dogovor o klicu podprogramov

- Splošen zapis ukaza za brezpogojni skok je:

```
j odmik(rb)
```

- **Dogovor**: Ker je po dogovoru povratni naslov shranjen v registru R31, naj bo `rb` za vrnitev iz podprograma register `R31`:

```
j 0(r31)
```

- **Dogovor**: za dolge skoke naj bo `rb` register `R26`.

```
lhi   r26, #SKOK
addui r26, r26, #SKOK
j     0(r26)
```

HIP - dogovor o klicu podprogramov

- Splošen zapis ukaza za brezpogojni skok je:

```
j odmik(rb)
```

- **Dogovor**: Ker je po dogovoru povratni naslov shranjen v registru R31, naj bo `rb` za vrnitev iz podprograma register `R31`:

```
j 0(r31)
```

- **Dogovor**: za dolge skoke naj bo `rb` register `R26`.

```
lhi   r26, #SKOK
addui r26, r26, #SKOK
j     0(r26)
```

HIP - dogovor o klicu podprogramov

Klicoči program pred klicem podprograma:

1. prva dva parametra (od leve proti desni) shrani v registra `R24` in `R25`
2. na sklad porine preostale parametre od desne proti levi,
3. izvede klic podprograma z ukazom `call`.

Klicoči program po vrnitvi iz podprograma:

1. s sklada pobriše parametre : skladovnemu kazalcu (`R30`) prišteje štirikratnik števila prenešenih parametrov na sklad.

HIP - dogovor o klicu podprogramov

- Do parametrov na skladu znotraj klicanega podprograma dostopamo preko enega registra, ki mu pravimo **kazalec na okvir** (*frame pointer*). Ta se med izvajanjem podprograma ne spreminja! Tako vedno vemo, kje so parametri na skladu.
  Dogovor: naj bo kazalec na okvir v registru R29.
- Lokalne spremenljivke (le avtomatske) se hranijo izključno na skladu.
- Iz podprograma vračamo eno samo 32-bitno vrednost v registru.
  Dogovor: naj bo to register R28. Če je parameter, ki ga vračamo zapisan z (do) 32 biti, potem ga vračamo po vrednosti, sicer po referenci!

HIP - dogovor o klicu podprogramov

- Do parametrov na skladu znotraj klicanega podprograma dostopamo preko enega registra, ki mu pravimo **kazalec na okvir** (*frame pointer*). Ta se med izvajanjem podprograma ne spreminja! Tako vedno vemo, kje so parametri na skladu.
  Dogovor: naj bo kazalec na okvir v registru R29.
- Lokalne spremenljivke (le avtomatske) se hranijo izključno na skladu.
- Iz podprograma vračamo eno samo 32-bitno vrednost v registru.
  Dogovor: naj bo to register R28. Če je parameter, ki ga vračamo zapisan z (do) 32 biti, potem ga vračamo po vrednosti, sicer po referenci!

HIP - dogovor o klicu podprogramov

- Do parametrov na skladu znotraj klicanega podprograma dostopamo preko enega registra, ki mu pravimo **kazalec na okvir** (*frame pointer*). Ta se med izvajanjem podprograma ne spreminja! Tako vedno vemo, kje so parametri na skladu.
  Dogovor: naj bo kazalec na okvir v registru `R29`.
- Lokalne spremenljivke (le avtomatske) se hranijo izključno na skladu.
- Iz podprograma vračamo eno samo 32-bitno vrednost v registru.
  Dogovor: naj bo to register `R28`. Če je parameter, ki ga vračamo zapisan z (do) 32 biti, potem ga vračamo po vrednosti, sicer po referenci!

HIP - dogovor o klicu podprogramov

Klicani podprogram ob vstopu:

1. na sklad porine povratni naslov - register R31;
2. na sklad porine kazalec na okvir - register R29;
3. v register R29 (kazalec na okvir) prepiše skladovni kazalec;
4. po potrebi rezervira prostor na skladu za lokalne spremenljivke (spreminja skladovni kazalec!);
5. na sklad **shrani vse registre, ki jih spreminja**;
6. do prenešenih parametrov in lokalnih spremenljivk dostopamo preko kazalca na okvir (register R29): zadnji parameter na skladu je na naslovu R29+12, prva lokalna spremenljivka je na naslovu R29.

HIP - dogovor o klicu podprogramov

Klicani podprogram ob vstopu:

1. na sklad porine povratni naslov - register R31;
2. na sklad porine kazalec na okvir - register R29;
3. v register R29 (kazalec na okvir) prepiše skladovni kazalec;
4. po potrebi rezervira prostor na skladu za lokalne spremenljivke (spreminja skladovni kazalec!);
5. na sklad **shrani vse registre, ki jih spreminja**;
6. do prenešenih parametrov in lokalnih spremenljivk dostopamo preko kazalca na okvir (register R29): zadnji parameter na skladu je na naslovu R29+12, prva lokalna spremenljivka je na naslovu R29.

HIP - dogovor o klicu podprogramov

Klicani podprogram ob vstopu:

1. na sklad porine povratni naslov - register R31;
2. na sklad porine kazalec na okvir - register R29;
3. v register R29 (kazalec na okvir) prepiše skladovni kazalec;
4. po potrebi rezervira prostor na skladu za lokalne spremenljivke (spreminja skladovni kazalec!);
5. na sklad **shrani vse registre, ki jih spreminja**;
6. do prenešenih parametrov in lokalnih spremenljivk dostopamo preko kazalca na okvir (register R29): zadnji parameter na skladu je na naslovu R29+12, prva lokalna spremenljivka je na naslovu R29.

HIP - dogovor o klicu podprogramov

Klicani podprogram ob vstopu:

1. na sklad porine povratni naslov - register R31;
2. na sklad porine kazalec na okvir - register R29;
3. v register R29 (kazalec na okvir) prepiše skladovni kazalec;
4. po potrebi rezervira prostor na skladu za lokalne spremenljivke (spreminja skladovni kazalec!);
5. na sklad **shrani vse registre, ki jih spreminja**;
6. do prenešenih parametrov in lokalnih spremenljivk dostopamo preko kazalca na okvir (register R29): zadnji parameter na skladu je na naslovu R29+12, prva lokalna spremenljivka je na naslovu R29.

HIP - dogovor o klicu podprogramov

Klicani podprogram ob vstopu:

1. na sklad porine povratni naslov - register R31;
2. na sklad porine kazalec na okvir - register R29;
3. v register R29 (kazalec na okvir) prepiše skladovni kazalec;
4. po potrebi rezervira prostor na skladu za lokalne spremenljivke (spreminja skladovni kazalec!);
5. na sklad **shrani vse registre, ki jih spreminja**;
6. do prenešenih parametrov in lokalnih spremenljivk dostopamo preko kazalca na okvir (register R29): zadnji parameter na skladu je na naslovu R29+12, prva lokalna spremenljivka je na naslovu R29.

HIP - dogovor o klicu podprogramov

Klicani podprogram ob vstopu:

1. na sklad porine povratni naslov - register R31;
2. na sklad porine kazalec na okvir - register R29;
3. v register R29 (kazalec na okvir) prepiše skladovni kazalec;
4. po potrebi rezervira prostor na skladu za lokalne spremenljivke (spreminja skladovni kazalec!);
5. na sklad **shrani vse registre, ki jih spreminja**;
6. do prenešenih parametrov in lokalnih spremenljivk dostopamo preko kazalca na okvir (register R29): zadnji parameter na skladu je na naslovu R29+12, prva lokalna spremenljivka je na naslovu R29.

HIP - dogovor o klicu podprogramov

Klicani podprogram pred izstopom:

1. v register `R28` zapiše vrednost, ki jo vrača;
2. s sklada obnovi vse shranjene registre;
3. s sklada 'odstrani' lokalne spremenljivke tako, da register `R29` prepiše v skladovni kazalec `R30`;
4. s sklada obnovi (prebere) staro vrednost registra `R29`;
5. s sklada obnovi (prebere) povratni naslov v register `R31`;
6. z ukazom `jmp` se vrne na naslov `R31+0`.

HIP - dogovor o klicu podprogramov

Klicani podprogram pred izstopom:

1. v register `R28` zapiše vrednost, ki jo vrača;
2. s sklada obnovi vse shranjene registre;
3. s sklada 'odstrani' lokalne spremenljivke tako, da register `R29` prepiše v skladovni kazalec `R30`;
4. s sklada obnovi (prebere) staro vrednost registra `R29`;
5. s sklada obnovi (prebere) povratni naslov v register `R31`;
6. z ukazom `jmp` se vrne na naslov `R31+0`.

HIP - dogovor o klicu podprogramov

Klicani podprogram pred izstopom:

1. v register `R28` zapiše vrednost, ki jo vrača;
2. s sklada obnovi vse shranjene registre;
3. s sklada 'odstrani' lokalne spremenljivke tako, da register `R29` prepiše v skladovni kazalec `R30`;
4. s sklada obnovi (prebere) staro vrednost registra `R29`;
5. s sklada obnovi (prebere) povratni naslov v register `R31`;
6. z ukazom `jmp` se vrne na naslov `R31+0`.

HIP - dogovor o klicu podprogramov

Klicani podprogram pred izstopom:

1. v register `R28` zapiše vrednost, ki jo vrača;
2. s sklada obnovi vse shranjene registre;
3. s sklada 'odstrani' lokalne spremenljivke tako, da register `R29` prepiše v skladovni kazalec `R30`;
4. s sklada obnovi (prebere) staro vrednost registra `R29`;
5. s sklada obnovi (prebere) povratni naslov v register `R31`;
6. z ukazom `jmp` se vrne na naslov `R31+0`.

HIP - dogovor o klicu podprogramov

Klicani podprogram pred izstopom:

1. v register `R28` zapiše vrednost, ki jo vrača;
2. s sklada obnovi vse shranjene registre;
3. s sklada 'odstrani' lokalne spremenljivke tako, da register `R29` prepiše v skladovni kazalec `R30`;
4. s sklada obnovi (prebere) staro vrednost registra `R29`;
5. s sklada obnovi (prebere) povratni naslov v register `R31`;
6. z ukazom `jmp` se vrne na naslov `R31+0`.

HIP - dogovor o klicu podprogramov

Klicani podprogram pred izstopom:

1. v register `R28` zapiše vrednost, ki jo vrača;
2. s sklada obnovi vse shranjene registre;
3. s sklada 'odstrani' lokalne spremenljivke tako, da register `R29` prepiše v skladovni kazalec `R30`;
4. s sklada obnovi (prebere) staro vrednost registra `R29`;
5. s sklada obnovi (prebere) povratni naslov v register `R31`;
6. z ukazom `jmp` se vrne na naslov `R31+0`.

HIP klic podprogramov - zgled

Predpostavimo, da želimo klicati naslednji podprogram:

```
int sestej(int a, int b) {
    int SUM;
    SUM = a + b;

    return SUM;
}
```

HIP klic podprogramov - zgled

- klicoči program :

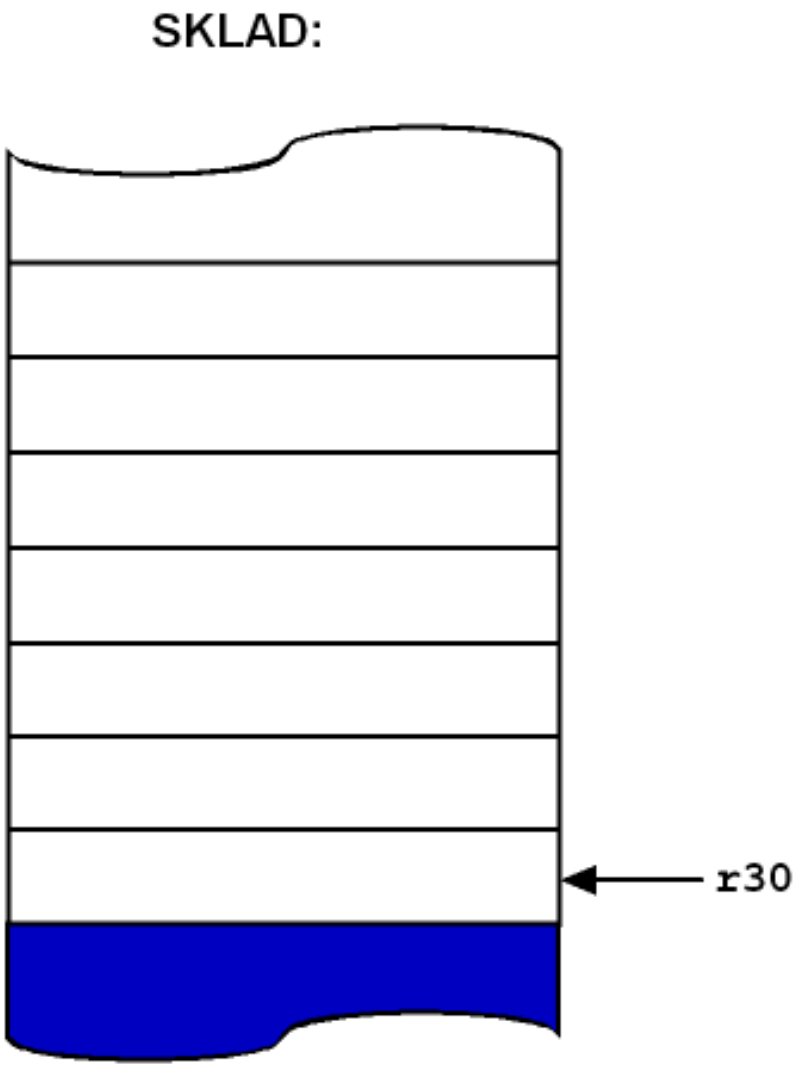
```
addui r30, r0, #0x4fc  ; nalozi skladovni kazalec
...
lw r24, a(r0)          ; prvi parameter prenasamo preko r24
lw r3, b(r0)           ; drugi parameter (tokrat za potrebe zgleda)
push r3                ; prenasamo preko sklada; sicer preko registra
call r31, sestej(r0)   ; klici podprogram in shrani povratni naslov v r31
                       ; Pozor: v nasem primeru se mora podprogram nahajati v prvih
                       ; 32K pomnilnika. Zakaj?
                       ; v splosnem moramo namesto r0 uporabiti drugi bazni
                       ; register, s cimer omogocimo postavitev podprograma
                       ; na poljuben (poravnani) naslov v pomnilniku
addui r30,r30,#4       ; pocisti parameter s sklada
...
```

HIP klic podprogramov - zgled

- klicani podprogram :

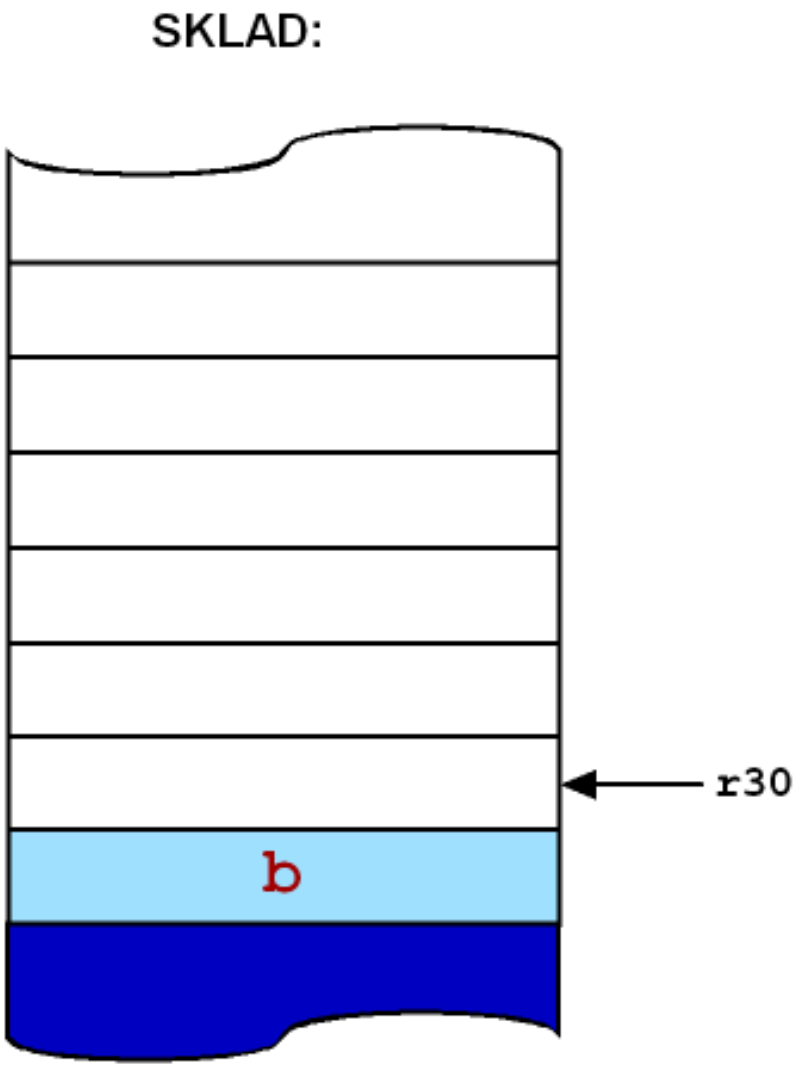
```
; VSTOPNA TOCKA: -----------------------------------
push r31          ; shrani povratni naslov
push r29          ; shrani r29
add r29,r0,r30    ; R29 <- SP : nastavi kazalec na okvir;
                  ;  ...sedaj lahko spreminjamo SP
;-------------------------------------------------

; PROCEDURA: ---------------------------------------
subui r30,r30,#4 ; rezerviraj na skladu prostor za
                  ; 32-bitno spremenljivko SUM
push r6           ; shrani register, ki se
                  ; v podprogramu spreminja
lw r6,12(r29)     ; r6 <- b
add r6,r6,r24     ; r6 = a + b
sw 0(r29), r6     ; SUM <- r6
lw r28, 0(r29)    ; vrednosti vracamo v r28!
pop r6            ; pred izstopom obnovimo r6
;-------------------------------------------------

; IZSTOPNA TOCKA: ----------------------------------
add r30,r0,r29    ; pobrisemo lokalne spremenljivke
                  ; s sklada
pop r29           ; obnovi r29
pop 31            ; povratni naslov v r31
j 0(r31)          ; povratek v klicoci program
```

Sklad - HIP - zgled.



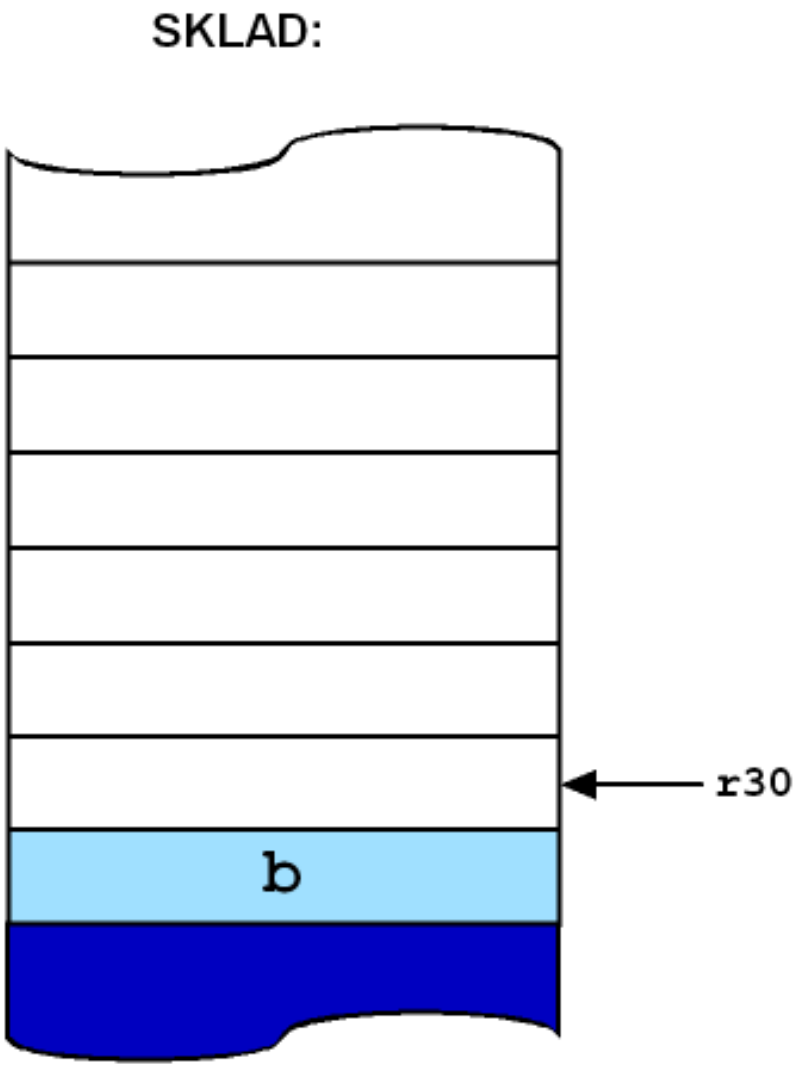

```
; KLICOCI PROGRAM:
lw r24, a(r0)
lw r3, b(r0)
push r3
call r31, sestej(r0)
addui r30,r30,#4
```

Sklad - HIP - zgled.



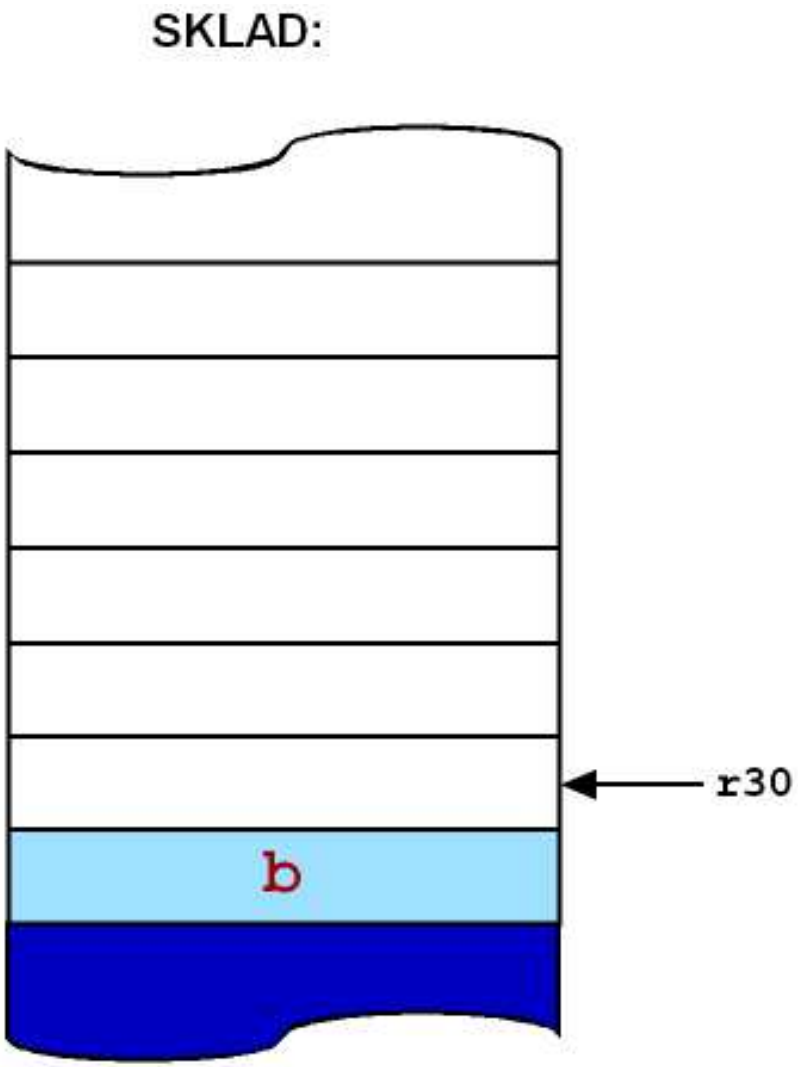

```
; KLICOCI PROGRAM:
lw r24, a(r0)
lw r3, b(r0)
push r3
call r31, sestej(r0)
addui r30,r30,#4
```

Sklad - HIP - zgled.



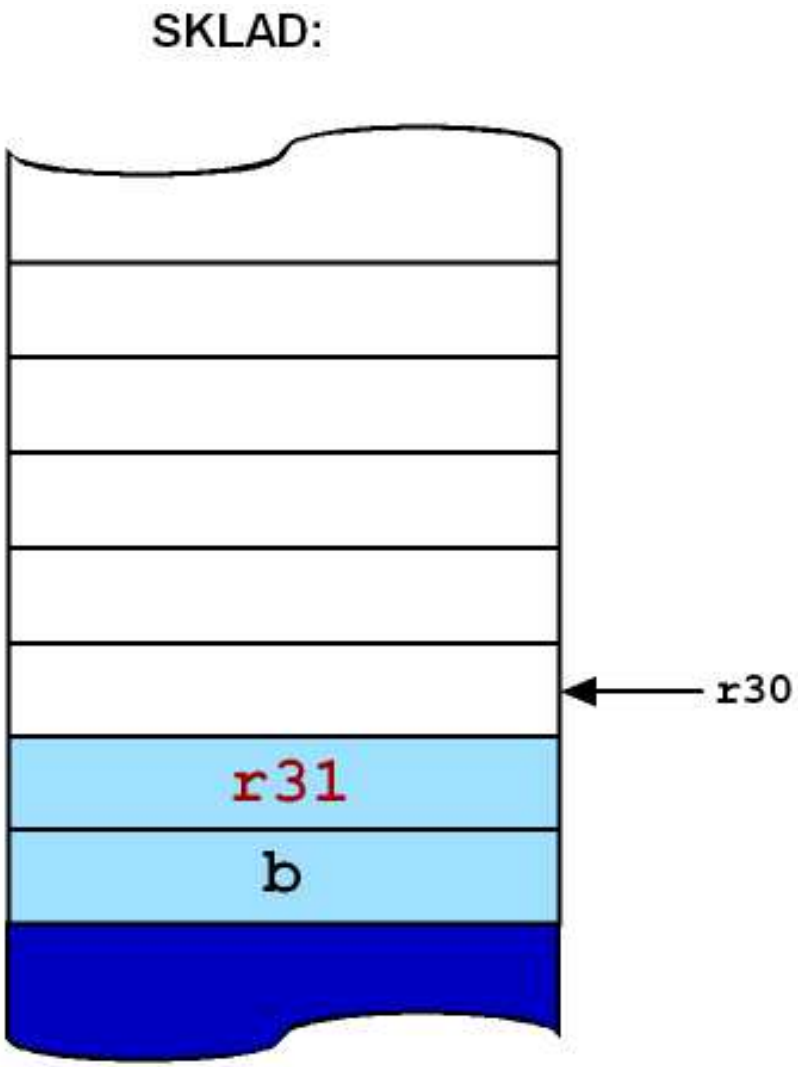

```
; KLICOCI PROGRAM:
lw r24, a(r0)
lw r3, b(r0)
push r3
call r31, sestej(r0)
addui r30,r30,#4
```

Sklad - HIP - zgled.



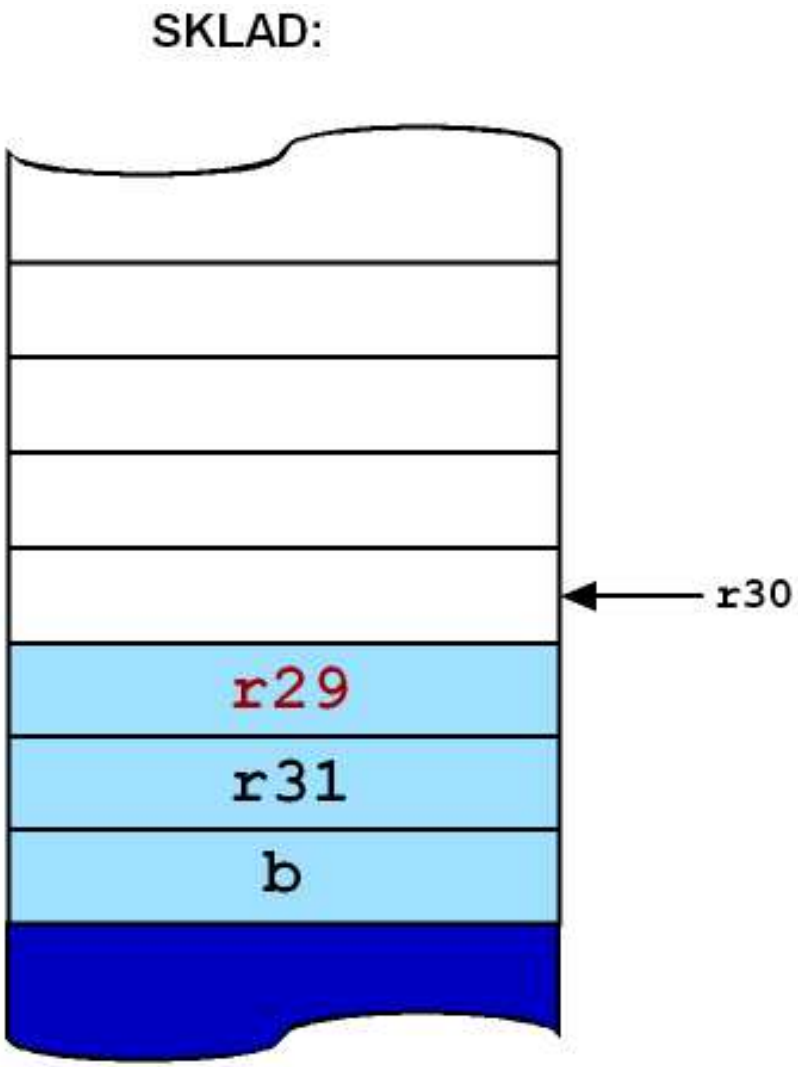

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



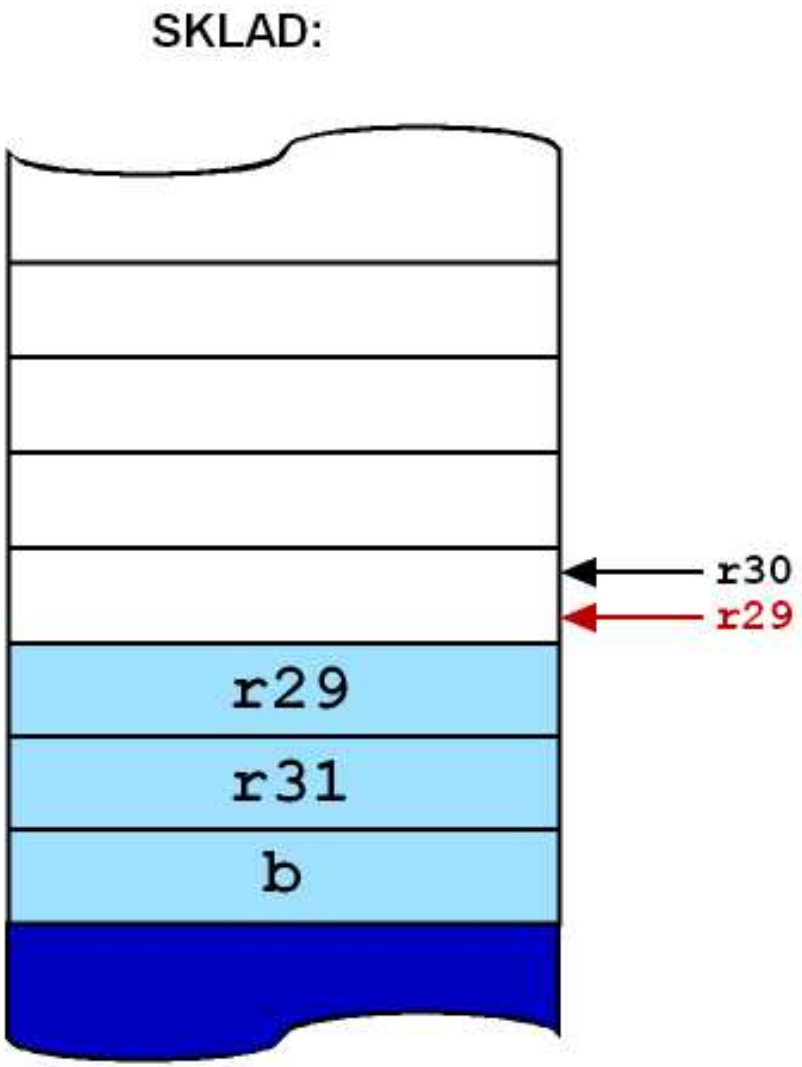

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



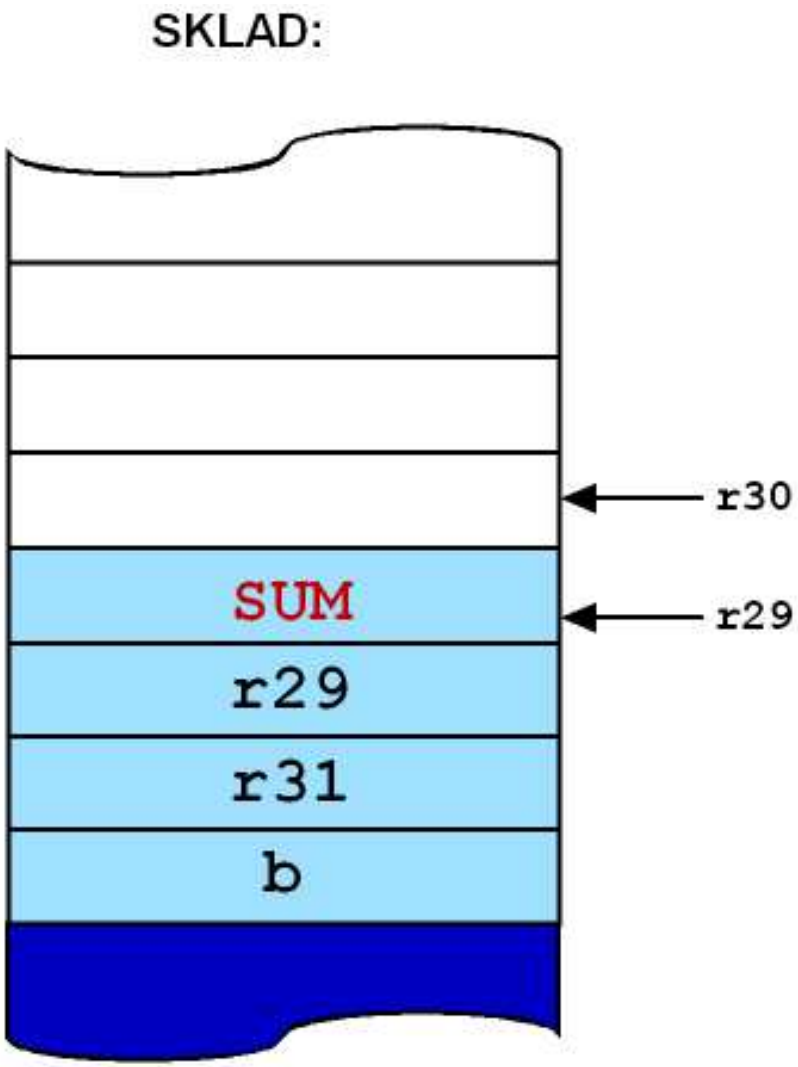

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



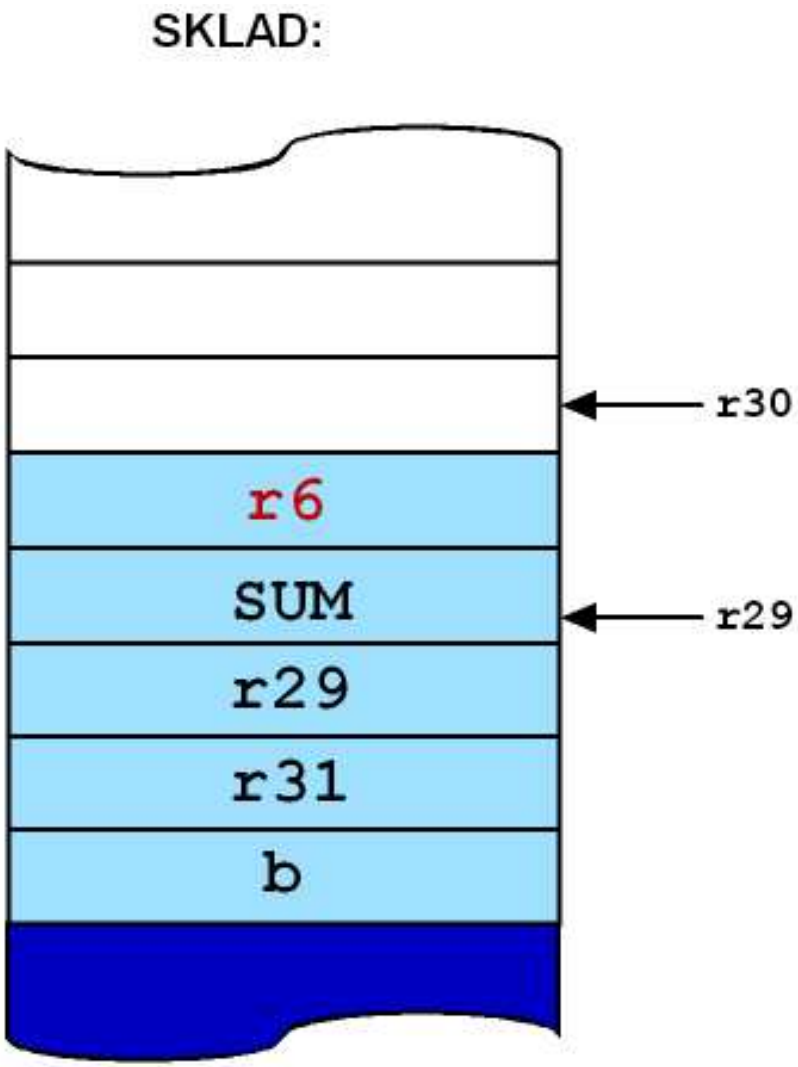

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



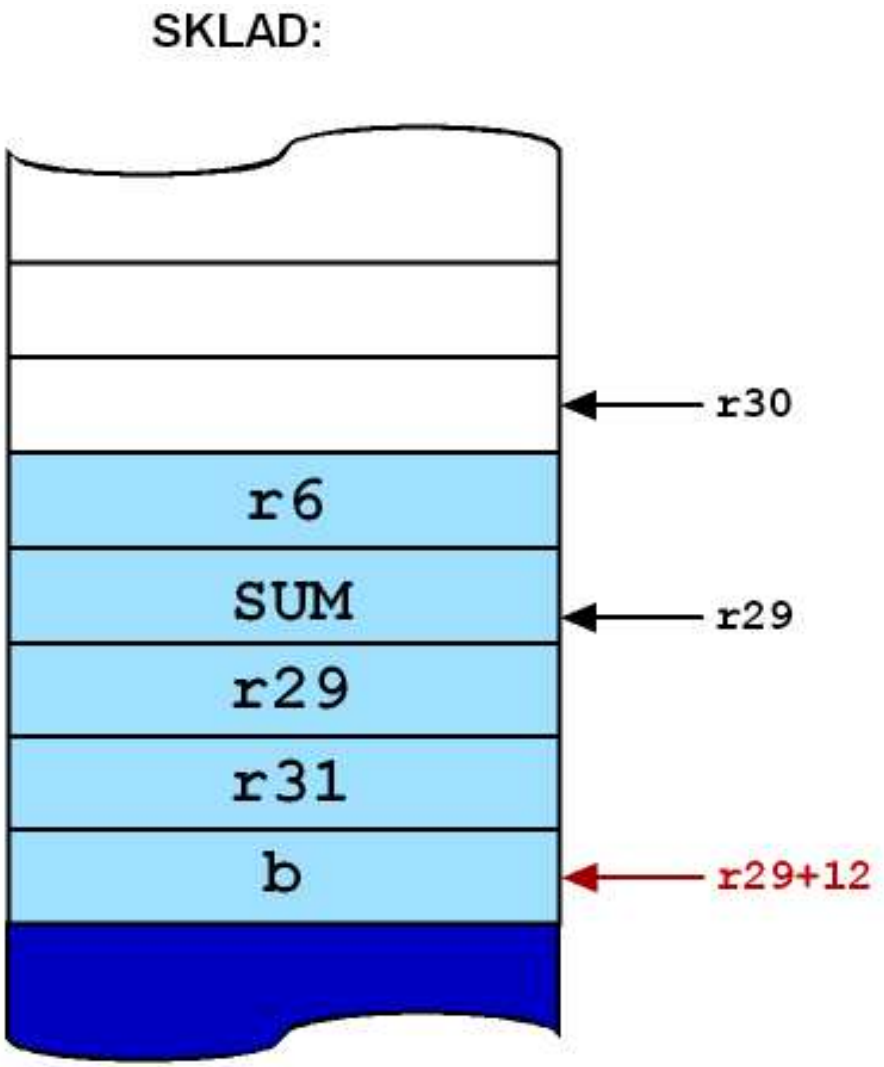

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



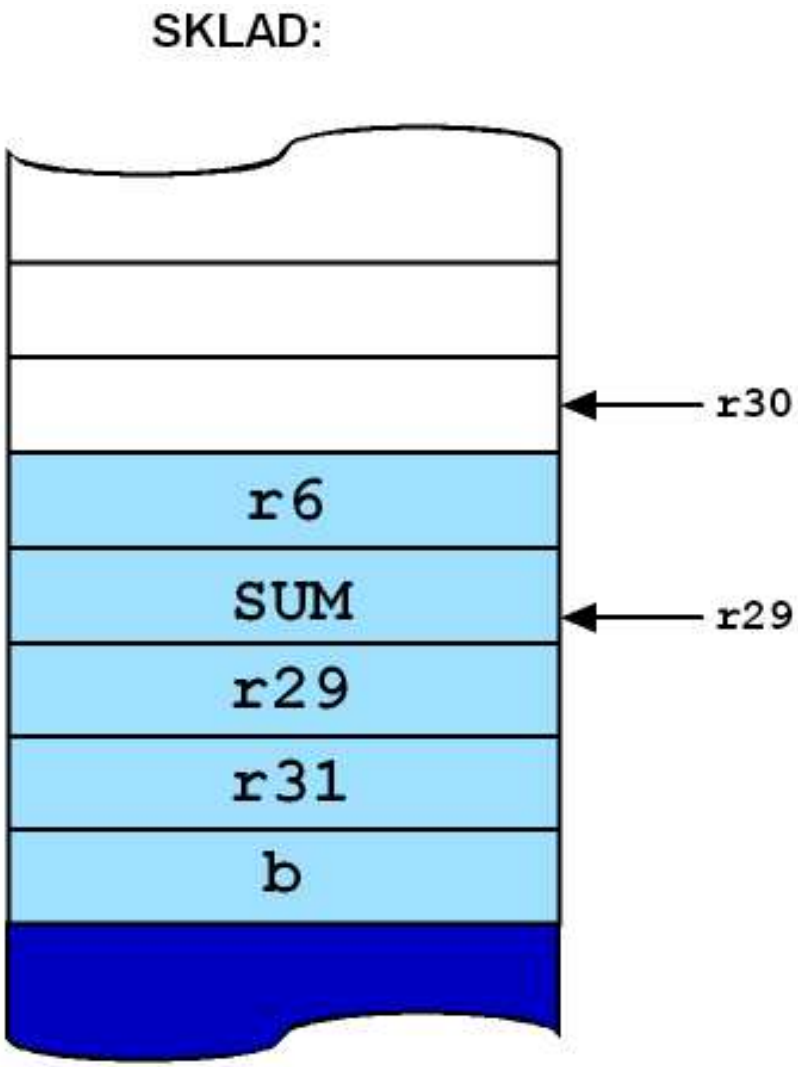

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



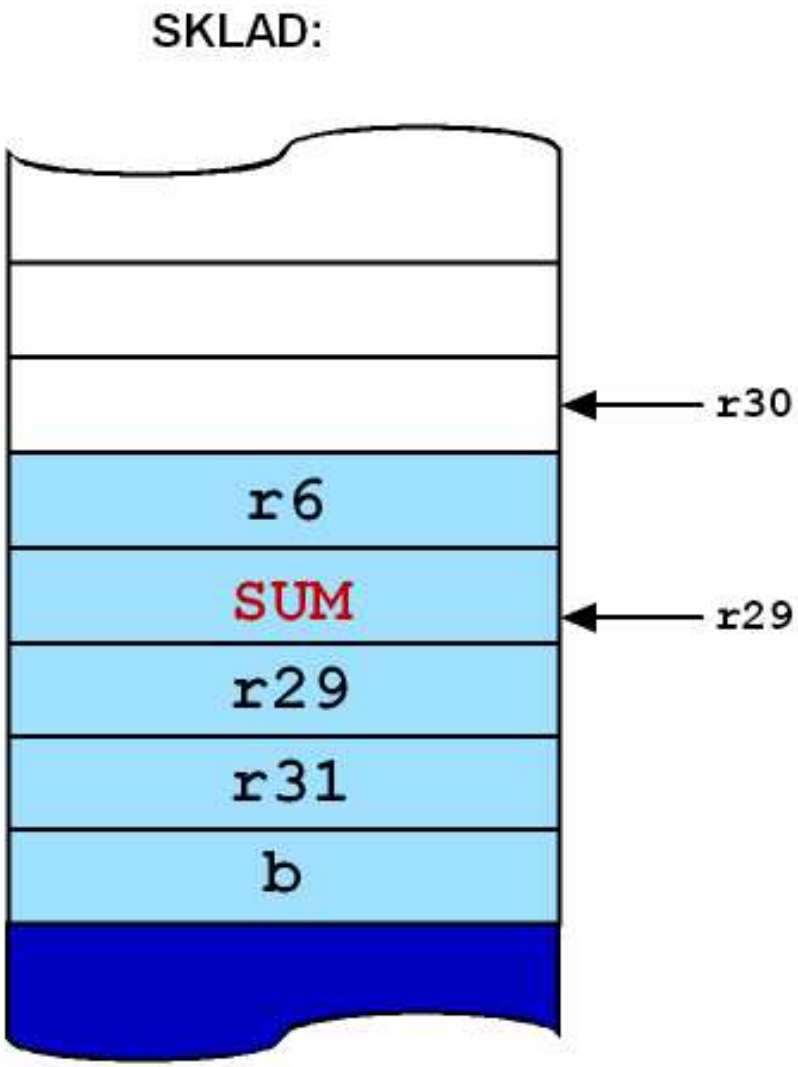

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



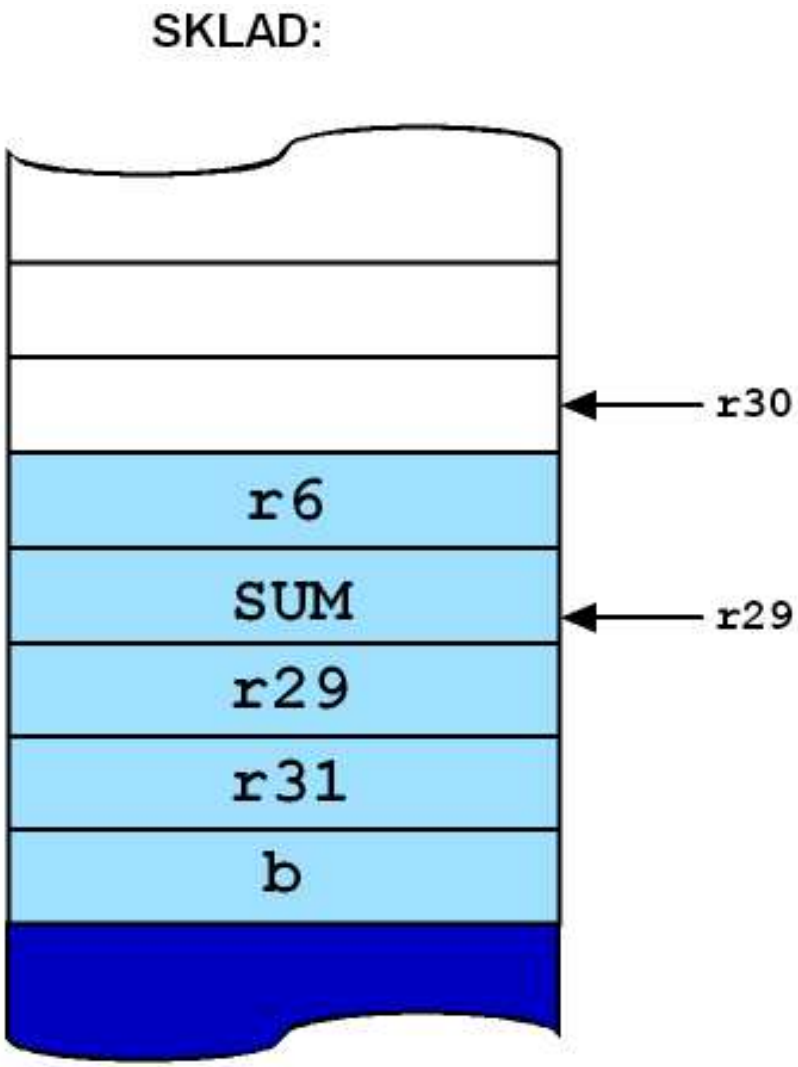

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



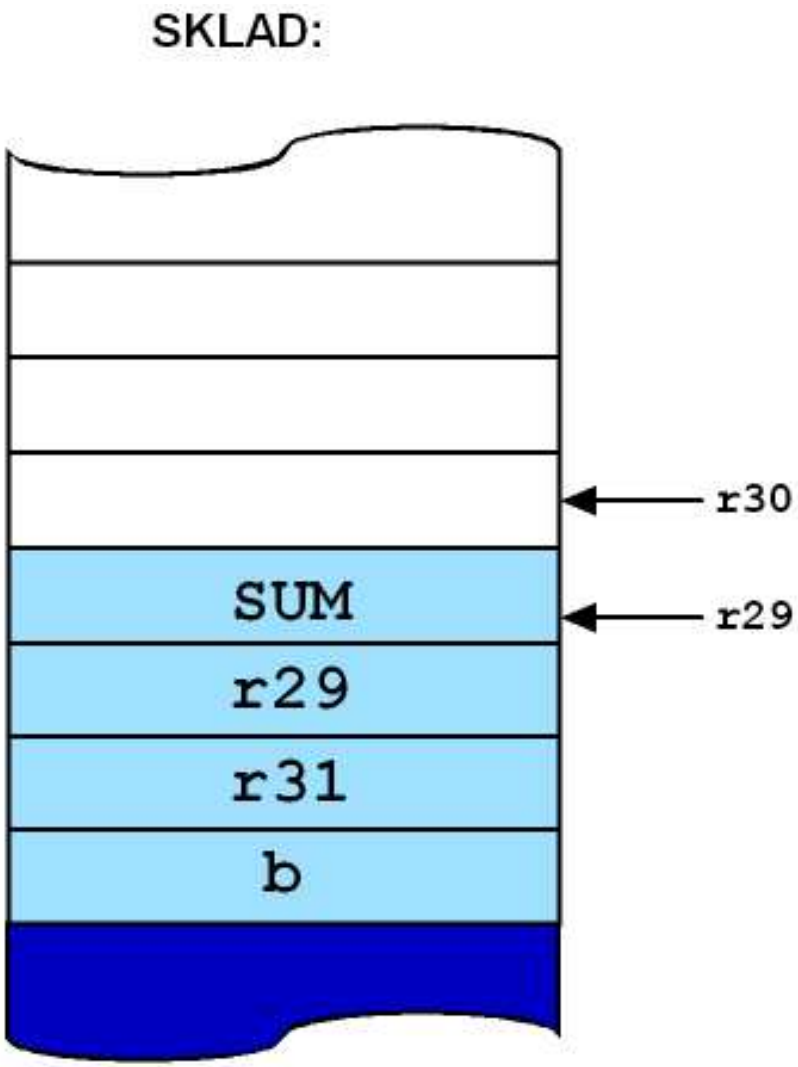

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



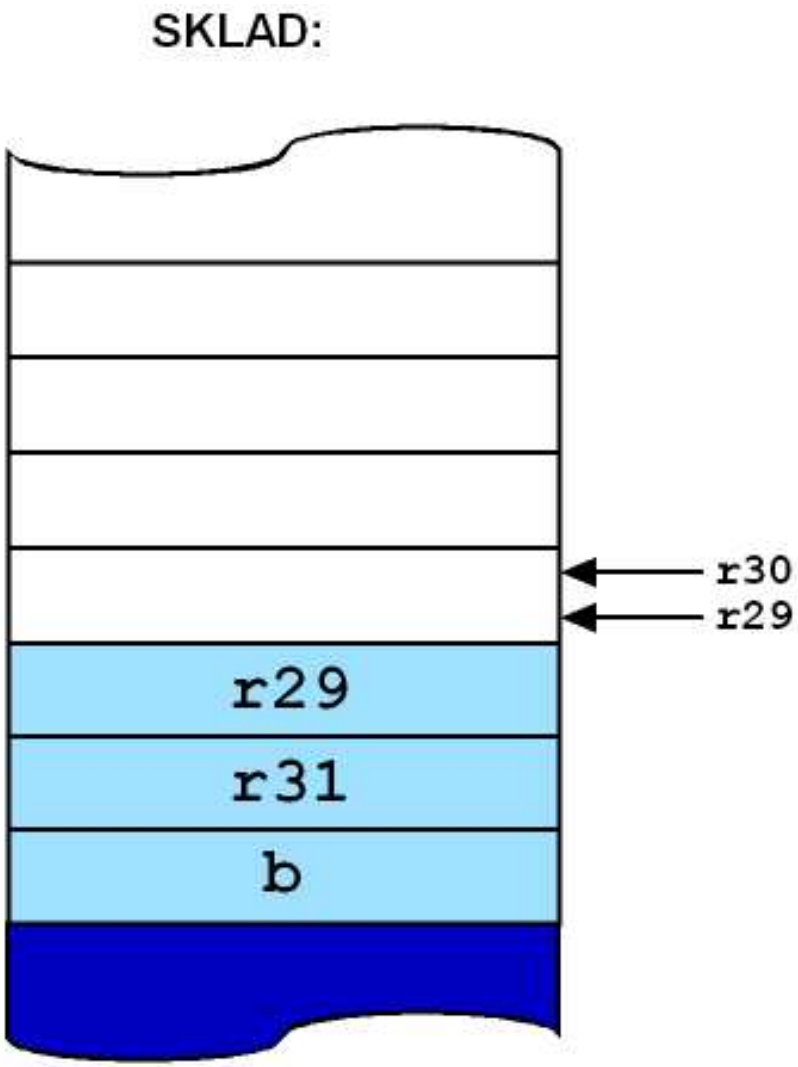

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



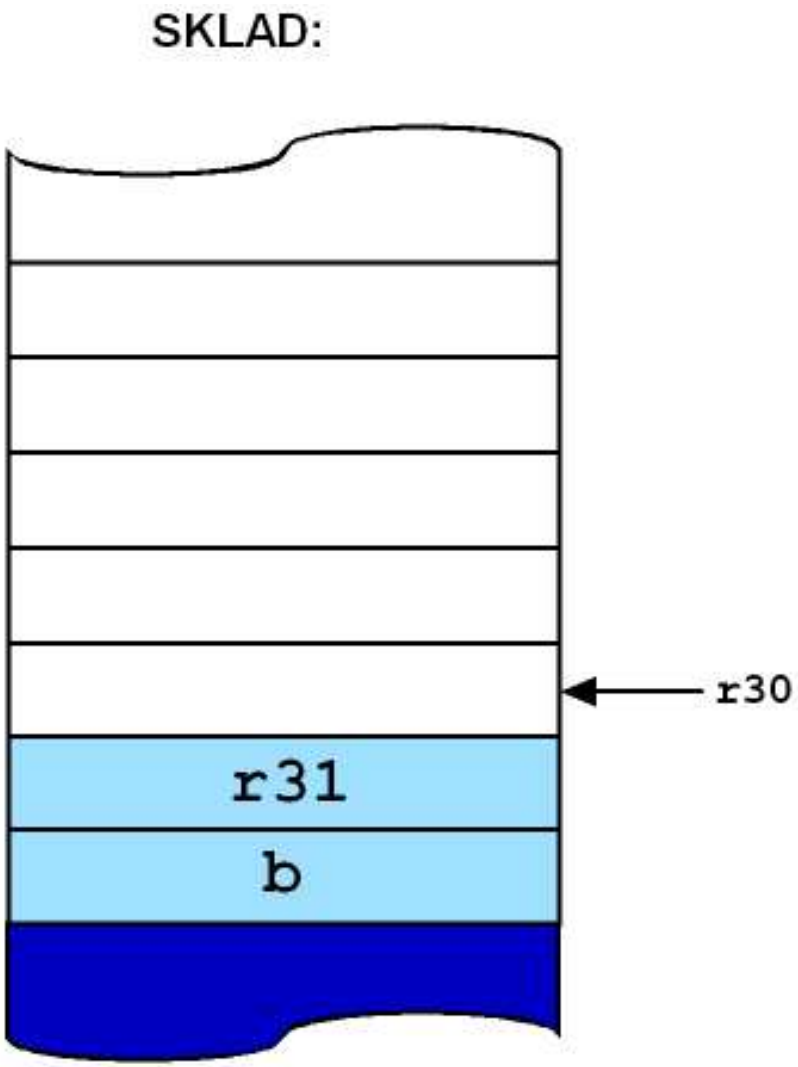

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



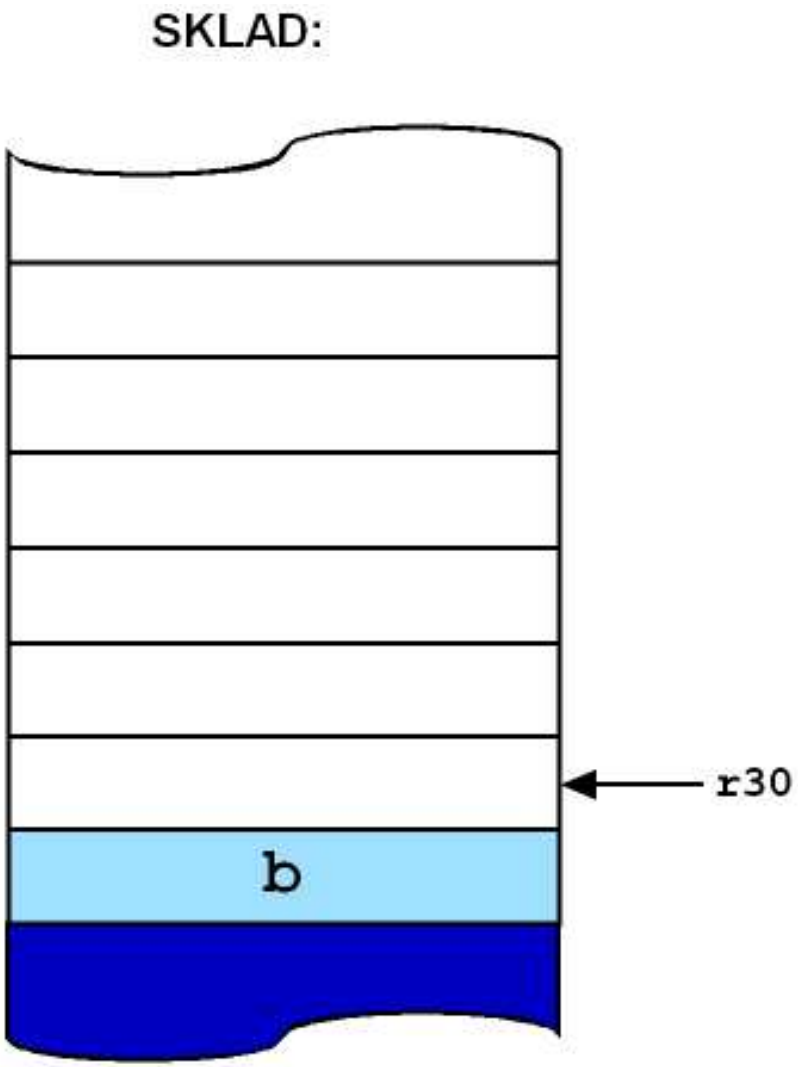

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



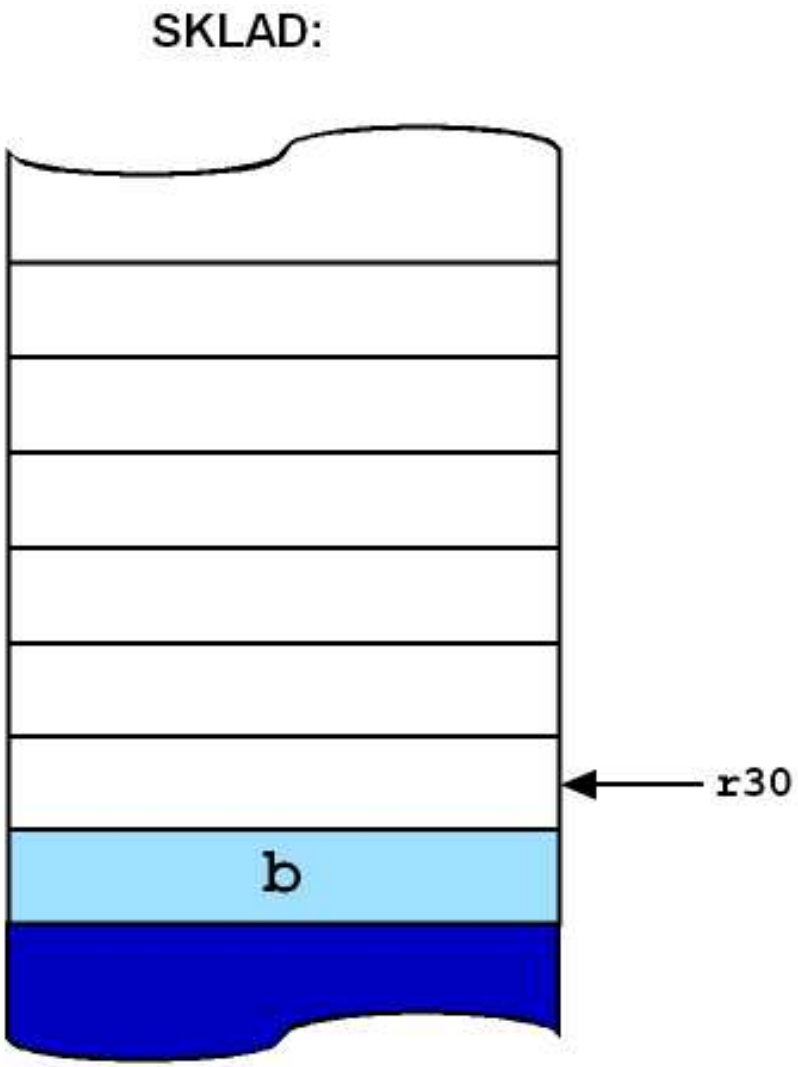

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.



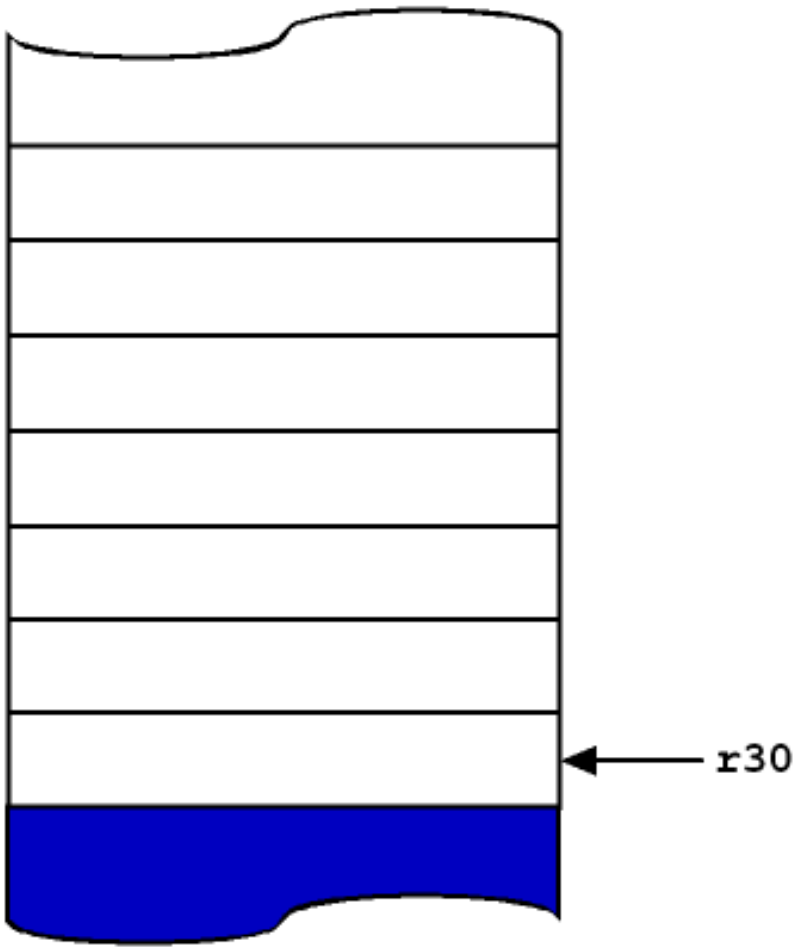

```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.




```
; PODPROGRAM:
; VSTOPNA TOCKA:
push r31
push r29
add r29,r0,r30

; PROCEDURA:
subui r30,r30,#4
push r6
lw r6,12(r29)
add r6,r6,r24
sw 0(r29), r6
lw r28, 0(r29)
pop r6

; IZSTOPNA TOCKA:
add r30,r0,r29
pop r29
pop 31
j 0(r31)
```

Sklad - HIP - zgled.




```
; KLICOCI PROGRAM:
lw r24, b(r0)
lw r3, a(r0)
push r3
call r31, sestej(r0)
addui r30,r30,#4
```

HIP - povzetek uporabe registrov

Register	Namen uporabe
R0	Ničla
R1-R23	Splošno namenski registri
R24	Prvi parameter z leve, ki se prenese v podprogram
R25	Drugi parameter z leve, ki se prenese v podprogram
R26	Bazni register za dolge skoke
R27	Bazni register za dolge klice
R28	Vrednost, ki jo vrača podprogram
R29	Kazalec na okvir
R30	Skladovni kazalec
R31	Povratni naslov

Za tiste, ki želijo znati več:

- Dušan Kodek. Arhitektura računalniških sistemov, 2. popravljena in razširjena izdaja, BI-TIM, 2000.
- David Patterson, John Hennessy. Computer Organization and Design, The Hardware/Software Interface, Third Edition (Appendix A: Assemblers, Linkers, and the SPIM Simulator)
- Procedure Call Standard for the ARM Architecture *http://www.arm.com/miscPDFs/8031.pdf*
- MicroBlaze Processor Reference Guide *http://www.xilinx.com/ise/embedded/edk_docs.htm*