PROJEKAT

IV termin

Dr Nevena Radović

Procedure

- **Procedure** (ili funkcije) se sastoje od niza instrukcija koje se mogu izvršavati *pozivom* u bilo kojoj tački izvršavanja cijelog programa, uključujući slučaj izvršavanja u drugim procedurama.
- Često ponavljani blokovi instrukcija se formatiraju u procedure.
- Sa aspekta pozivanja procedure, postoje dvije primarne aktivnosti: **povezivanje** (*linkage*) i **prosljeđivanje argumenata**.
- Takodje, upotreba procedura u MIPS asembleru podrazumijeva upotrebu seta registara specijalne namjene.

Procedure

- Uopšteno govoreći, postoje dva tipa procedura:
  - Procedure koje ne pozivaju druge procedure (*leaf procedures*),
  - Procedure koje pozivaju druge procedure (*non-leaf procedures*).
- Takođe, definisane su **konvencije pozivanja** (*calling convention*) koje definišu odnos rutine koja vrši pozivanje (***caller***) i rutine koja se poziva (***callee***).
- Tako postoje:
  - Caler konvencija
  - Calee konvencija

Caller konvencija

- Ovom konvencijom su definisani zahtjevi za ***caller***, odnosno za rutinu koja poziva proceduru:
- Pozivajuća rutina treba da sačuva sve nesačuvane registre ($a0 - $a3, $t0 - $t9, $v0, $v1) koji su neophodni nakog što se poziv ostvari;
- Pozivajuća rutina treba da proslijedi sve argumente. Za rad sa integer-ima:
  - Prvi argument se prosljeđuje u $a0;
  - Drugi argument se prosljeđuje u $a1;
  - Treći argument se prosljeđuje u $a2;
  - Četvrti argument se prosljeđuje u $a3;
  - Ostali argumenti (ako ih ima) se prosljeđuju pomoću steka.

Caller konvencija

- Pozivajuća rutina koristi instrukciju:

  jal ImeProcedure

- Nakon završtetka procedure, ***caller*** mora da vrati sve nesačuvane registre i prilagodi pokazivač na vrh steka ($sp) ukoliko su argumenti proslijeđivani pomoću steka.

Povezivanje (*linkage*)

- Pojam povezivanja (*linkage*) u kontekstu procedura se odnosni na osnovni poces „odlaska“ na proceduru i „vraćanja“ na tačnu lokaciju u pozivajućoj rutini.
- Operacija povezivanja se obavlja upotrebom instrukcija ***jal*** i ***jr***. Obije instrukcije koriste **$ra** registar.
- **$ra** register (**$31**) je setovan na povratnu adresu kao sastavni dio poziva procedure.
- **jal** (jump and link) instrukcija kopira $pc u $ra register i skače na proceduru.
- **$pc** sadrži adresu sljedeće instrukcije koja treba da se izvrši. To je upravo adresa instrukcije koja se izvršava odmah nakon poziva procedure, što je ujedno i tačno mjesto za povratak, nakon što se procedura izvrši.

Povezivanje (*linkage*)

- Ukoliko procedura ne poziva neku drugu proceduru ništa dalje nije potrebno raditi sa registrom $ra.
- Povratak iz procedure se obavlja instrukcijom:

jr $ra

- Ukoliko pak procedura poziva neku drugu proceduru, registar $ra mora biti sačuvan. Pošto $ra sadrži povratnu adresu ona će biti promijejena kada procedura pozove sljedeću proceduru.
- Stoga, $ra mora biti sačuvan i vraćen sa steka u pozivajućoj rutini. Ovo se tipično obavlja samo jednom na početku, a potom na kraju procedure.

Prosljeđivanje argumenata

- Integer argumenti se prosljeđuju pomoću registara:

  $a0, $a1, $a2, $a3

  upravo navedenim redosljedom. Ostali argumenti (ako ih ima) se prosljeđuju pomoću steka.

- Procedura vraća rezultate kroz registre: **$v0 i/ili $v1**.

Konvencija čuvanja registara

- MIPS calling konvencija zahtijeva da samo određeni registri (ne svi) budu sačuvani prilikom poziva procedura.
  - Integer registeri $s0 - $s7 moraju biti sačuvani od strane procedure.
- Prilikom pisanja procedure to znači da integer registri $s0 - $s7 budu smješteni/iščitani sa steka ukoliko su korišćeni/mijenjani.
- Integer registri $t0 - $t9 se koriste za čuvanje privremenih vrijednosti tako da ne moraju biti čuvani u okviru procedure.

Konvencija čuvanja registara

- Registri **$at**, **$k0** i **$k1** su rezervisani od strane asemblera i operativnog sistema, i ne trebaju da se koriste u programima.
- Registar **$fp** (frame pointer) se upotrebljava kada se argumenti procedure prosljeđuju pomoću steka.
- Register **$gp** je globalni pokazivač (pokazuje na globalno dostupne podatke). Ovaj registar se u principu ne koristi direktno kada se pišu asemblerski programi.

Callee konvencija

- Ovom konvencijom su definisani zahtjevi za ***callee***, odnosno za pozvanu proceduru:
- Smjesti bilo koji ***promijenjeni*** *saved* registar na stek.
  - Ovo se odnosi na integer registre $s0 - $s7, $ra, $fp, ili $gp.
  - Ako callee procedura poziva neku drugu proceduru, $ra mora biti sačuvan.
  - Ako se $fp mijenja, $fp mora biti sačuvan. Ovo se dešava u situacijama kada se argumenti prosljeđuju putem steka. Ako se argumeti smještaju na stek, $fp se podešava na sljedeći način: **$fp = $sp + (veličina frejma)**. Na ovaj način će $fp pokazivati na prvi argument smješten na steku.
  - Prostor za lokalne varijable treba biti kreiran na steku (dinamičke lokalne promjenljive).
- Kada se mijenja $sp registar, to je potrebno uraditi u jednoj operaciji (ne u više uzastopnih).

Callee konvencija

- Procedura može da pristupi integer argumentima u registrima $a0 - $a3.
- Argumentima koji su proslijeđeni pomoću steka se pristupa korišćenjem $fp.
- Procedura smješta povratne vrijednosti (ukoliko ih ima) u $v0 i $v1.

- **Primjer**: Napisati proceduru *stepen* koja računa funkciju x^y, a potom tu proceduru pozvati u glavnom programu za vrijednosti x=3 i y=5.
- **Rješenje:**

```
#  Deklaracija podataka
.data
        x:              .word 3
        y:              .word 5
        rezultat:       .word 0
#  Glavni program
.text
.globl main
main:
        lw $a0, x       # prosljedjivanje argumenata
        lw $a1, y
        jal stepen      # poziv procedure
        sw $v0, rezultat
# Kraj programa
        li $v0, 10
        syscall         # terminate
.end main
```

```
# Funkcija za racunanje x^y
# Argumenti: $a0 – x, $a1 – y
# Rezultat: $v0 - x^y

stepen:
        li $v0, 1
        li $t0, 0                       # brojac
Loop:
        mul $v0, $v0, $a0
        add $t0, $t0, 1                 # brojis koliko je mnozenja bilo
        blt $t0, $a1, Loop
        jr $ra
```

- **Primjer**: Napisati proceduru *sumiranje* koja sabira šest ulaznih argumenata, a potom tu proceduru pozvati u glavnom programu za vrijednosti: num1=3, num2=5, num3=3, num4=3, num5=3, num6=5.

- **Rješenje:**

```
# Deklaracija podataka
.data
        num1:       .word 3
        num2:       .word 5
        num3:       .word 3
        num4:       .word 5
        num5:       .word 3
        num6:       .word 5
        sum:        .word 0
# Glavni program
# Prva 4 argumenta se prosljedjuju kroz $a0-$a3.
# Naredna 2 argumenta se prosljedjuju pomocu steka.

.text
.globl main
main:
```

```
        lw $a0, num1            # prosljedjivanje argumenata
        lw $a1, num2
        lw $a2, num3
        lw $a3, num4
        lw $t0, num5
        lw $t1, num6
        subu $sp, $sp, 8
        sw $t0, ($sp)           # smjestanje na stek 2 zadnja arg
        sw $t1, 4($sp)
        jal sumiranje           # poziv procedure
        sw $v0, sum
        addu $sp, $sp, 8        # ocisti stek

# Kraj programa
        li $v0,10
        syscall
.end main
```

```
# Argumenti: $a0 - num1, $a1 - num2, $a2 - num3, $a3 - num4
# ($fp) - num5, 4($fp) - num6
# Rezultat: $v0 – num1+num2+num3+num4+num5+num6

sumiranje :
        subu $sp, $sp, 4        # sacuvaj registre
        sw $fp, ($sp)
        addu $fp, $sp, 4        # setuj frame pointer
#  Izracunavanje
        li $v0, 0
        add $v0, $v0, $a0       # num1
        add $v0, $v0, $a1       # num2
        add $v0, $v0, $a2       # num3
        add $v0, $v0, $a3       # num4
        lw $t0, ($fp)           # num5
        add $v0, $v0, $t0
        lw $t0, 4($fp)           # num6
        add $v0, $v0, $t0
```

```
#  Vrati registre
        lw $fp, ($sp)
        addu $sp, $sp, 4
        jr $ra
```