Arhitektura računara

dr.sc. Amer Hasanović

Pregled

- MIPS ISA
  - Logičke operacije
    - `and, andi, or, ori, xor, xori, nor, sll, srl, sra`
  - Petlje i nejednakosti
    - `slt, slti, sltu, sltiu`
  - Operacije na stack-u
    - `jal, jr`

Logičke operacije

- Logičke operacije tretiraju kompletan registar kao skup od 32 pojedinačna bita
- Format:
  - `1 2,3,4`
    - 1 – operacija
    - 2 – registar koji prima vrijednost
    - 3 – registar operand u operaciji
    - 4 – registar ili konstanta operand u operaciji
- Kada je 1 `and, or, xor,` ili `nor,` 4 je registar
- Kada je 1 `andi, ori,` ili `xori` 4 je konstanta
- Korisno za operacije maskiranja i setovanja npr:
  - `andi $t0, $t0, 0xfff`
  - `ori $t0,$t0,0xffff0000`

Šift operacije

- Format:
  - `1 2,3,4`
    - 1 – operacija
    - 2 – registar koji prima vrijednost
    - 3 – registar operand u operaciji
    - 4 – numerička konstanta 0 do 32
- Šift operacije pomjeraju bite iz izvornog registra (operand 3) u lijevo ili desno za broj mjesta dat konstantom (operand 4) a prazne bite pune sa 0 i snimaju rezultat u destinacijski registar (operand 2)
  - opearcija `sll` pomjera bite u lijevo
  - operacija `srl` pomjera bite u desno
  - operacija `sra` radi isto što i `srl`, samo što prazne bite puni u ovisnosti od predznaka
- `sll` i `sra` kompajler često koristi kod operacije množenja tj dijeljenja sa stepenom od 2

Petlje

- Postoji više varijanti za imlemenataciju petlji iz C programskog jezika putem uslovnog grananja i skokova u MIPS kodu
- Prije implementacije, potrebno je prvo preformulirati C petlju u jedan od ekvivalentih oblika sa `goto` i `if` instrukcijama npr:

```
while (Uslov)
    Tijelo
```

```
  goto uslov;
tijelo:
  Tijelo
uslov:
  if (Uslov)
    goto tijelo;
```

```
for (Init; Uslov; Izraz )
    Tijelo
```

```
  Init;
  goto uslov;
tijelo:
  Tijelo
  Izraz
uslov:
  if (Uslov)
     goto tijelo;
```

Petlje primjer

- Neka su C varijable raspoređene u registrima i to: `i → $s0, j → $s1, k → $s2, A → $s3,` i neka je dat slijedeći C kod:

```
while (A[i] == k)
  i = i + j;
```

- Možemo napraviti dvije varijante MIPS implementacije:

Varijanta 1:

```
        j uslov
tijelo: addu $s0, $s0, $s1
uslov:  sll $t0, $s0, 2
        addu $t0, $t0, $s3
        lw $t0, 0($t0)
        beq $t0, $s2, tijelo
izlaz:
```

Varijanta 2:

```
tijelo: sll $t0, $s0, 2
        addu $t0, $t0, $s3
        lw $t0, 0($t0)
        bne $t0, $s2, izlaz
        addu $s0, $s0, $s1
        j tijelo
izlaz:
```

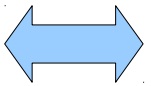
Nejednakosti

- Za test nejednakosti oblika `a < b` MIPS koristi instrukciju `slt`, koja ima format:
  - `slt $reg1,$reg2,$reg3`
  - a značenje je `$reg1 = $reg2 < $reg3`
- Postoje i varijacije instrukcije:
  - `slti` gdje je treći operand numerička konstanta umjesto registra
  - `sltu` i `sltiu` – isto što `slt` i `slti`, samo što ne operiraju sa predznakom
- Npr:
  - `$s0 → 0xFFFFFFFA i  $s1 → 0x0000FFFA`
    - `slt $t0, $s0, $s1`
    - `sltu $t1, $s0, $s1`

Grananja na osnovu nejednakosti

- `slt` se može koristiti u kombinaciji sa `bne` za implemetaciju C `if` izraza npr. neka je dat C izraz:
  - `if (a < b) goto Manje;`
- ekvivalentna MIPS verzija:
  - `slt $t0,$s0,$s1`
  - `bne $t0,$0,Manje`
- Ostale varijante nejednakosti mogu se dobiti zamjenom mjesta `slt` operanada te kombiniranjem sa `beq` ili `bne`
- Npr neka je `$s0 → i, $s1 → j`

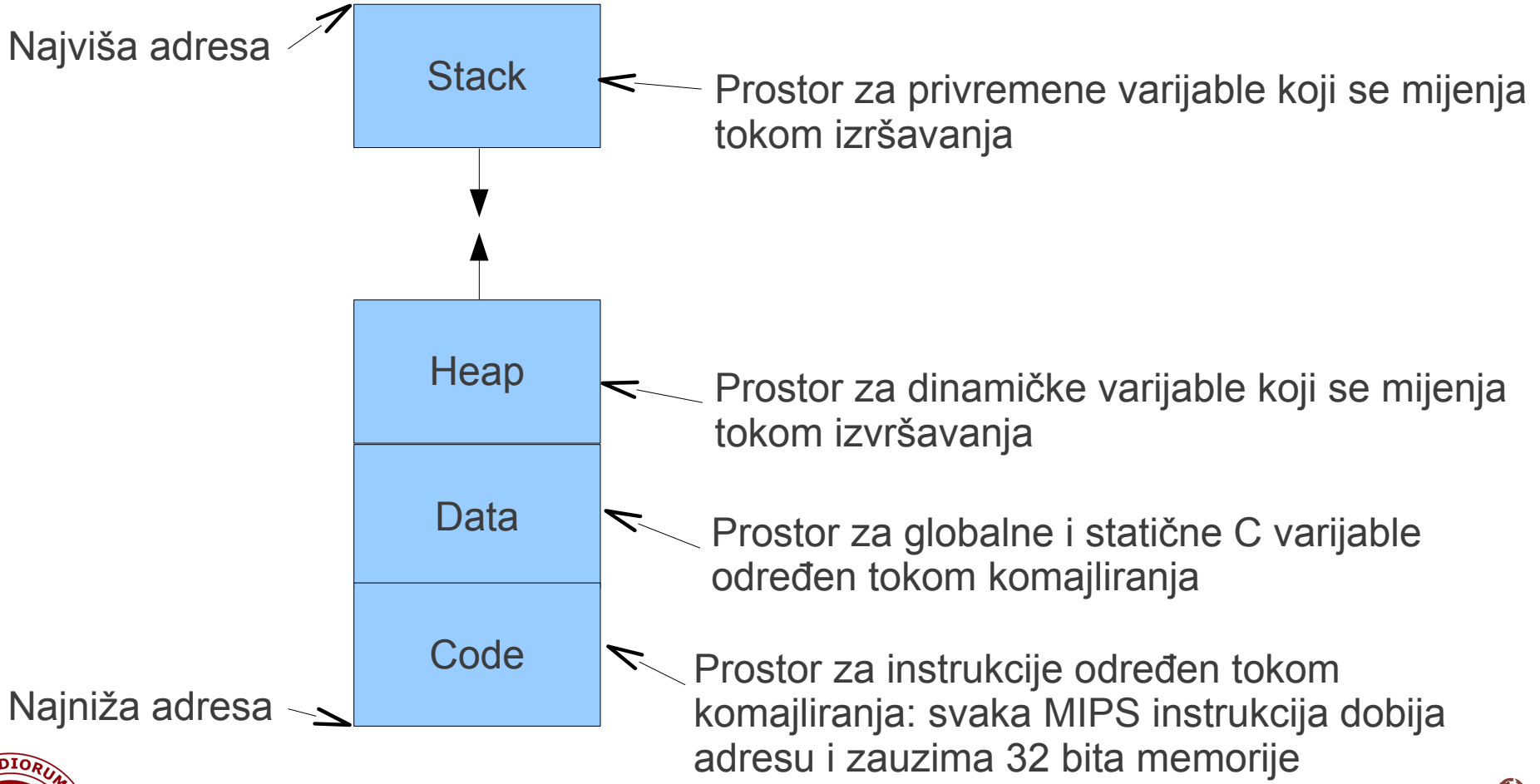
```
do
{
    i--;
} while ( j >= 2 || j < i );
```



```
Tijelo: addi $s0,$s0,-1
        slti $t0,$s1,2
        beq  $t0,$0,Tijelo
        slt  $t0,$s1,$s0
        bne  $t0,$0,Tijelo
```

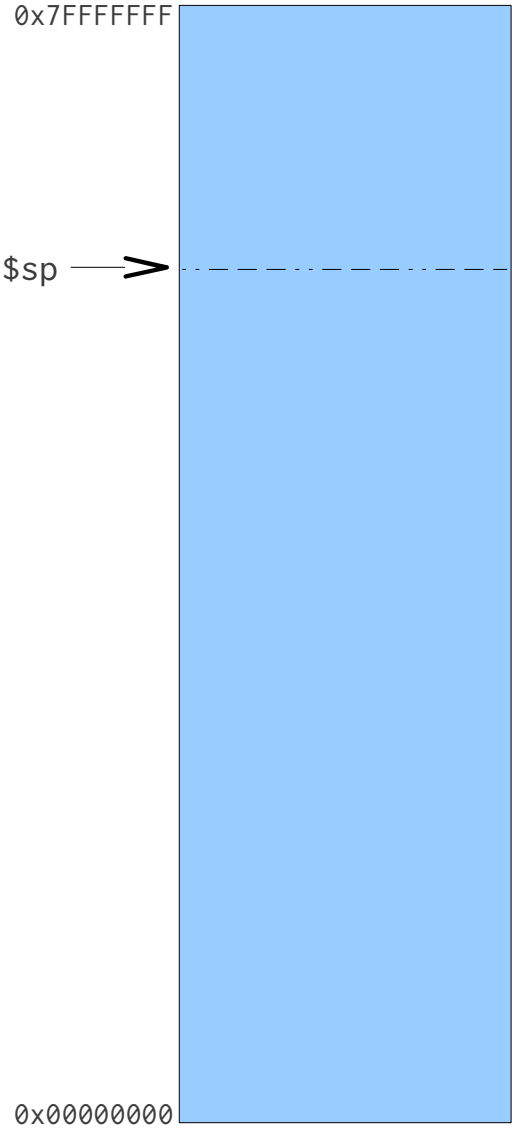
Programi i memorija

- Kada se počnu izvršavati programi dobijaju određeni segment memorije na raspolaganje koji ima raspored kao na slici:



Stack segment

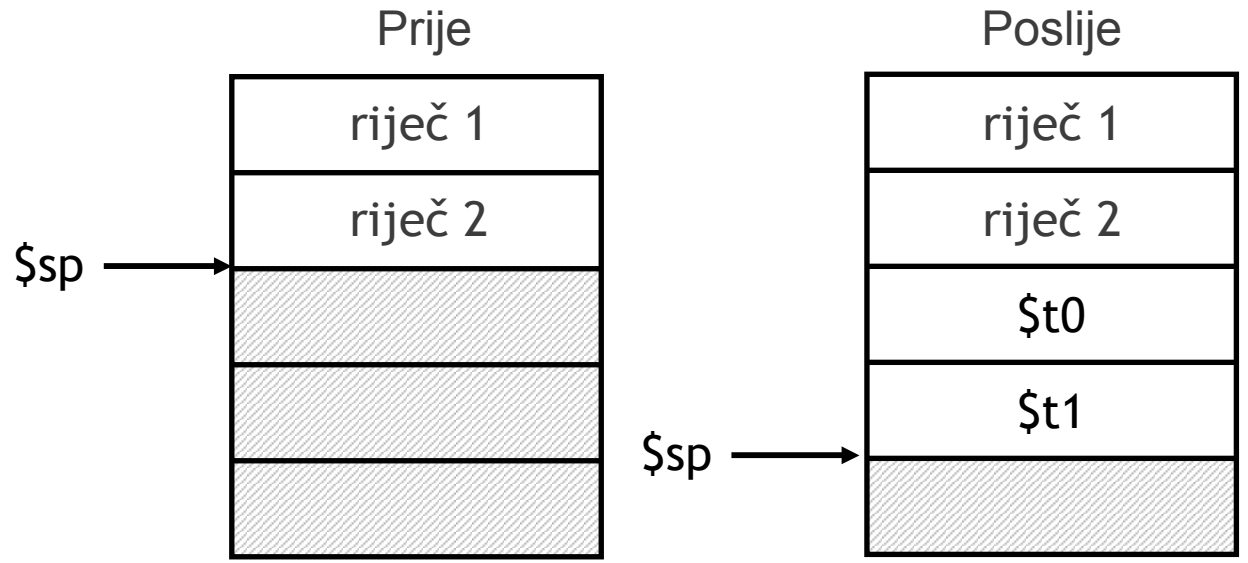
- Stack je segment alociran na vrhu memorijskog prostora programa
- Koristi kontinualni blok memorije, tj nema rupa u stack-u
- Proširuje se ka nižim adresama u memoriji
- Podržava operacije samo na dnu stack-a
  - *push* operacija proširuje stack dodavajući vrijednost na dno stacka
  - *pop* operacija sužava stack preuzimajući vrijednost sa dna stacka
- Adresa trenutnog dna stack drži se u registru `$sp` tj `$29`



Push operacija

- *push* operacija predstavlja dodavanje vrijednosti na stack i obavlja se u dva koraka:
  - Smanjiti vrijednost `$sp` za broj byte-a koji zauzima vrijednost
  - Snimiti vrijednost na adresu preuzetu iz `$sp`

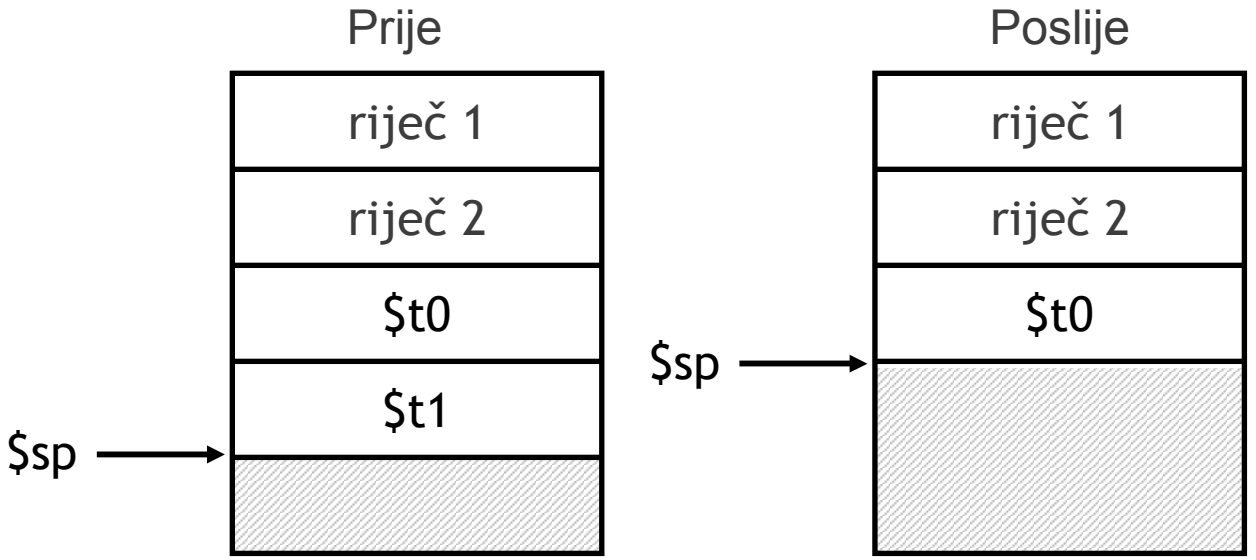
```
addiu $sp,$sp,-8
sw $t0,4($sp)
sw $t1,0($sp)
```



Pop operacija

- *pop* operacija predstavlja preuzimanje vrijednosti sa stack i obavlja se u dva koraka:
  - Ako je potrebno, preuzeti vrijednost sa trenutne adrese koja se nalazi u registru `$sp`
  - Povećati vrijednost registra `$sp` za broj byte-a koliko je zauzimala preuzeta vrijednost

```
lw $s0,0($sp)
addiu $sp,$sp,4
```



Funkcije

- Kod implementacije funkcija na nivou MIPS koda potrebno obratiti pažnju na tri činjenice:
  - Funkcije mijenjaju kontrolu toka programa
    - prvi put prilikom poziva funkcije
    - drugi put prilikom vraćanja iz poziva funkcije
  - Funkcije primaju vrijednosti za parametre (proizvoljan broj u C-u) te vraćaju povratne vrijednosti (tačno jedna vrijednost u C-u)
  - Funkcije mogu alocirati proizvoljan broj lokalnih varijabli (tip svake varijable mora biti poznat prilikom kompajliranja u C-u)

Poziv i povratak iz funkcije

- Ime funkcije sa stanovišta MIPS koda je oznaka (tj adresa) na kojoj počinje prva instrukcija u funkciji.
- Za pozivanje funkcije koristi se instrukcija `jal` i to u formatu:
  - `jal Oznaka`
    - `jal` prvo snima adresu za povratak, tj adresu slijedeće instrukcije iz koda koji je izvršio poziv, u posebni registar `$ra`
    - `jal` zatim vrši preusmjeravanje toka programa na izvršavanje prve instrukcije u pozvanoj funkciji
- Za povratak iz funkcije koristi se instrukcija `jr` i to u formatu:
  - `jr $ra`
    - `jr` odmah preusmjerava tok programa na instrukciju koja se nalazi na adresi čija je vrijednost pročitana iz registra koji je jedini operand u instrukciji

Parametri i povratne vrijednosti

- Za proslijeđivanje vrijednosti u i van funkcije MIPS koristi konvenciju
  - Do četiri vrijednosti mogu biti proslijeđene kao argumenti funkciji njihovim snimanjem u registre `$a0`, `$a1`, `$a2` i `$a3`
  - Funkcija može da vrati do dvije vrijednosti u registrima `$v0` i `$v1`
- Ovo predstavlja konvenciju čije sprovođenje niti jednog trenutka ne kontroliše niti asembler niti procesor

Primjer jednostavna C funkcija

C funkcija

```
int minOd3( int a, int b, int c )
{
  int min = a ;
  if ( b < min )
     min = b ;
  if ( c < min )
     min = c ;
  return min;
}
```

MIPS implementacija

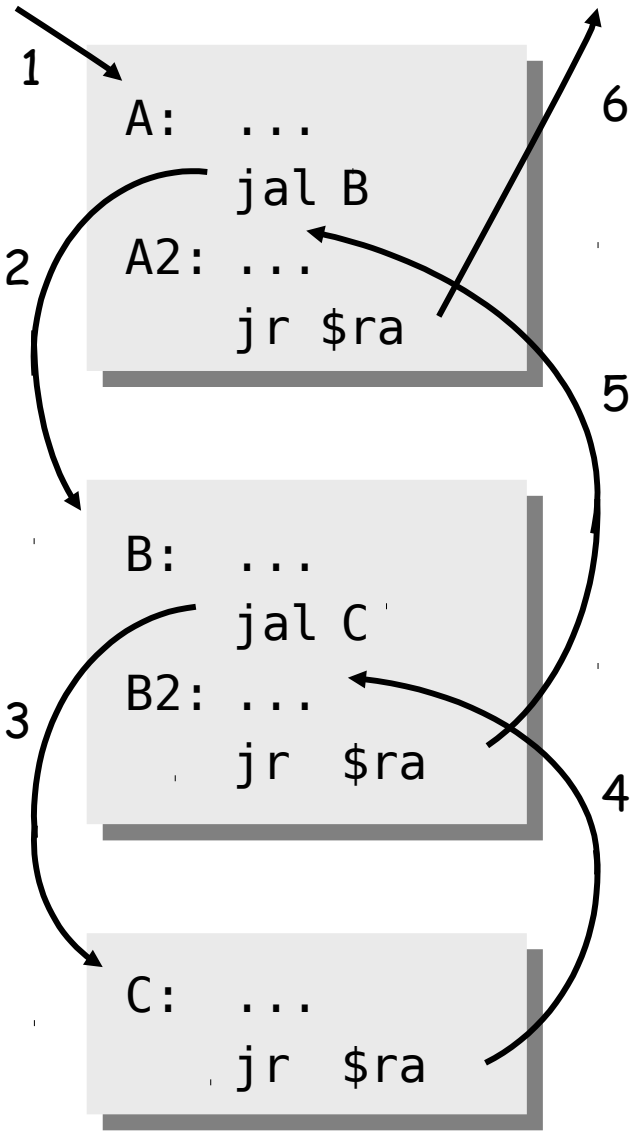
```
minOd3:  add $t0,$a0,$0
         slt $t1,$a1,$t0
         beq $t1,$0, IF2
         add $t0, $a1,$0
IF2:     slt $t1,$a2,$t0
         beq $t1,$0,KRAJ
         add $t0,$a2,$0
KRAJ:    add $v0,$t0,0
         jr $ra
```

Poziv funkcije

```
addi $a0,$0,8
addi $a1,$0,5
addi $a2,$0,23
jal minOd3
add $t0, $v0,$0 # a = minOd3(8,5,23)
```

Ugnježđeni pozivi funkcije

1. neko pozove A
2. A pozove B
3. B pozove C
4. C se vrati u B
5. B se vrati u A
6. povratak A



- Funkcije se izvršavaju po LIFO proceduri
- Šta se događa sa registrima `$ra`, `$a0,...$a3`?
- CPU ima ograničen broj registara i moguće je da će više funkcija koristiti iste registre
- Možemo prije poziva funkcije snimiti bitne registre te vratiti njihovu vrijednost nakon završetka funkcije

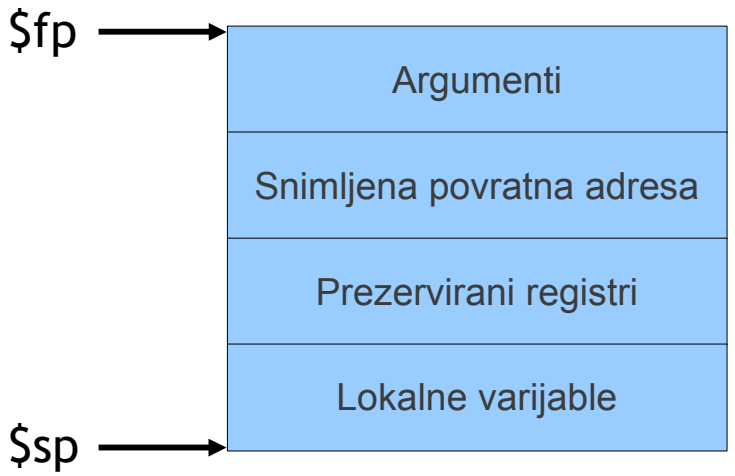
Stack frame

- Svaka funkcija dok traje njeno izvršenje alocira segment memorije na dnu stack-a (tzv *stack frame* ili *activation record*) u koji snima promjenjene registre i lokalne varijable.
- Lokacije početka stack frame funkcije često se snima u registru `$fp`
- Prilikom interakcije dvije funkcije:
  - Funkcije koja poziva – *caller* (pozivatelj)
  - Funkcije koja je pozvana – *callee* (pozvani)
- MIPS dijeli zadatke snimanja registara i to
  - *Caller* je odgovoran da snimi stanje neprezerviranih registara:
    - `$t0,... $t9, $a0,... $a3, $v0, $v1`
  - *Callee* je odgovoran da snimi stanje prezerviranih registara
    - `$s0,... $s7, $ra, $fp, $sp`

Caller – callee interakcija

- *Caller*
  - Snima neprezervirane registre u stack frame
  - Učitava vrijednosti parametara u `$a0,...$a3`, ostatak snima na stack-u
  - Izvršava `jal` instrukciju
- *Callee* postavka
  - Alocira memoriju za novi frame (`$sp=$sp-f_velicina`)
  - Snima prezervirane registre u stack frame
- *Callee* povratak
  - Postavlja povratnu vrijednost u `$v0` (i `$v1`)
  - Sa stack frame vraća originalne vrijednosti u prezervirane registre
  - Pop stack frame-a (`$sp=$sp+f_velicina`)
  - Caller povratak `jr $ra`

Stack frame izgled (MIPS konvencija)



Primjer

```
int foo(x,y)
{
  return bar(x,x)+y;
}

int bar(x,y)
{
  return x & y;
}
```

```
foo:
    addiu $sp,$sp,-8
    sw $ra,4($sp)
    sw $a1,0($sp)
    add $a1,$a0,$0
    jal bar
    lw $a1,0($sp)
    add $v0,$v0,$a1
    lw $ra,4($sp)
    addiu $sp,$sp,8
    jr $ra
bar:
    and $v0,$a0,$a1
    jr $ra
```