# Arhitektura računara

dr.sc. Amer Hasanović

# Pregled

- Rekurzivne funkcije
- Binarne reprezentacije instrukcija

# Rekurzivne funkcije

- Tretiraju se isto kao i obične funkcije na nivou MIPS koda
- Funkcija se ujedno ponaša kao *caller* i *callee*

```
int fib(int n)
{
  if ( n < 2 )
    return 1;
  else
    return fib(n-1)+fib(n-2);
}

int main()
{
  fib(6);
  return 0;
}
```

# fib MIPS ekvivalent

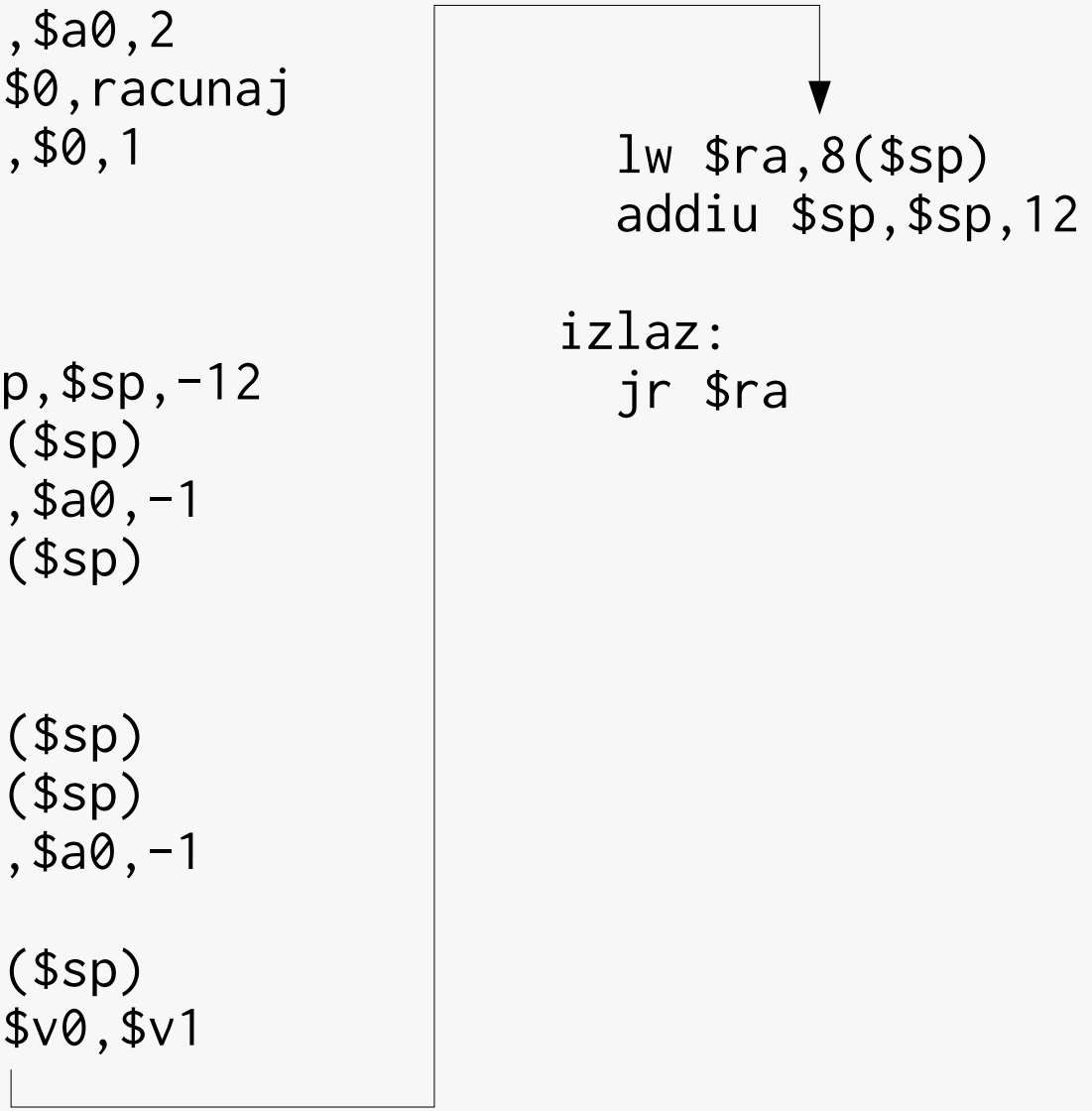
```
fib:
  slti $t0,$a0,2
  beq $t0,$0,racunaj
  addi $v0,$0,1
  j izlaz

racunaj:
  addiu $sp,$sp,-12
  sw $ra,8($sp)
  addi $a0,$a0,-1
  sw $a0,4($sp)

  jal fib
  sw $v0,0($sp)
  lw $a0,4($sp)
  addi $a0,$a0,-1
  jal fib
  lw $v1,0($sp)
  add $v0,$v0,$v1

  lw $ra,8($sp)
  addiu $sp,$sp,12

izlaz:
  jr $ra
```



# Mašinske instrukcije

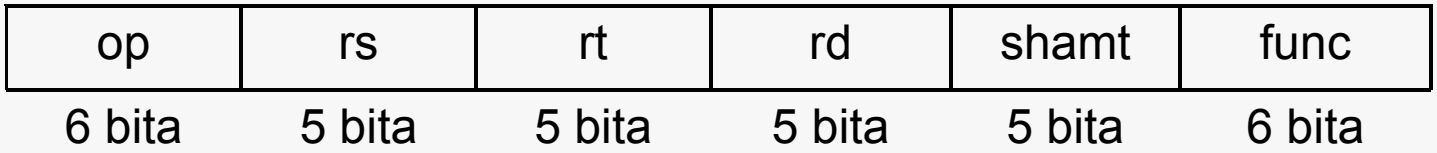
- Dosada korišten MIPS assembly:
  - Operacije imaju imena (npr. `add`), registri imaju imena (npr `$t0`)
  - Destinacije branch operacija za preusmjeravanje toka programa koriste oznake
- Programi se moraju kovertirati u binarni zapis kako bi se mogli unijeti u memoriju i izvršavati na CPU-u.
  - Svaka MIPS instrukcija kodira se u 32 bit-a
  - Instrukcije se učitavaju u text (tj. code) segment programa u memoriji
  - Adresa instrukcije koju trenutno izvršava procesor nalazi se u registru `PC` (Program Counter)
  - Nakon izvršetka instrukcije PC se inkrementira za vrijednost 4, sem ukoliko prethodna instrukcija nije grananje

# Reprezentacija instrukcija

- Svaka instrukcija kodira se u 32 bit-a tj jednu riječ
- U binarnoj reprezentaciji riječ se izdijeli na polja
- Svako polje, kod dekodiranja, nosi segment informacije za CPU a vezano za instrukciju
- Spram organizacije bita u polja, MIPS dijeli instrukcije u tri generalne forme i to:
  - Format `I` koristi se za kodiranje instrukcija sa numeričkim konstantama, `lw`, `sw`, `beq` i `bne`
  - Format `J` koristi se za kodiranje skokova tj instrukcije `j` i `jal`
  - Format `R` koristi se za kodiranje svih ostalih instrukcija

# R format

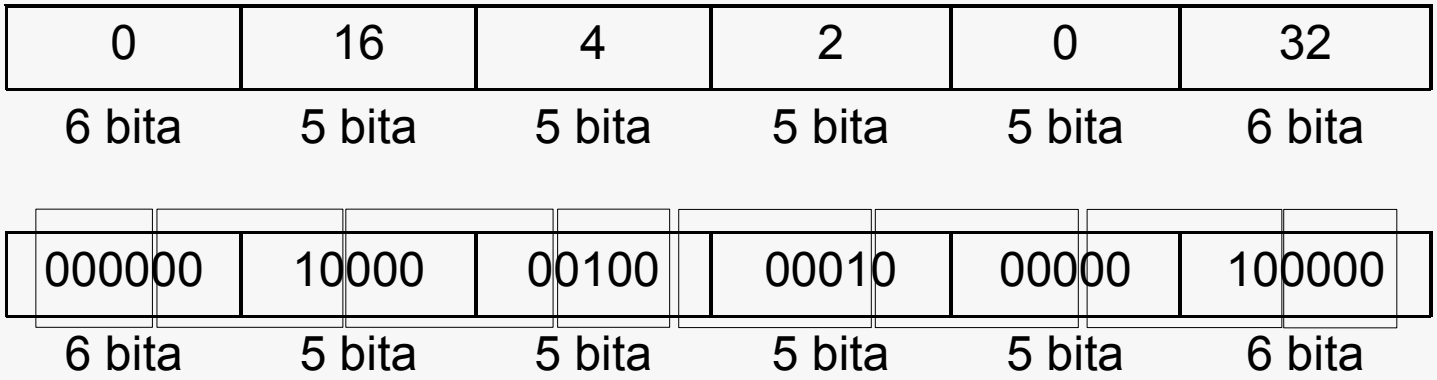
- Dijeli riječ koja predstavlja instrukciju na 5 polja
- Svako polje posmatra se kao separatni cijeli broj bez predznaka:




- Pojedinačna polja reprezentiraju:
  - `rs` prvi registar operand
  - `rt` drugi registar operand
  - `rd` destinacijski registar operand
  - `shamt` broj bita za šift operacije, 0 za ostale operacije
  - `op` opcode, 0 za R format
  - `func` u kombinaciji sa `opcode` određuje instrukciju

# R format primjer

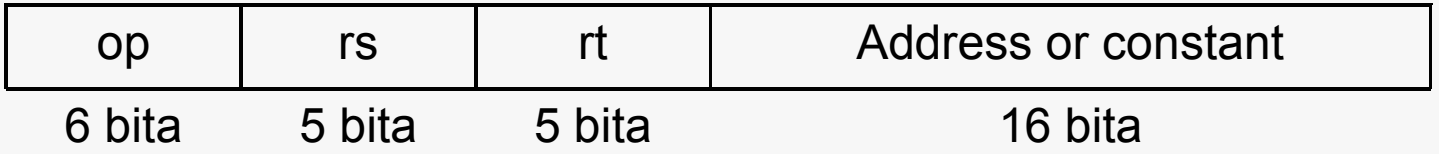
- Neka je data instrukcija:
  - `add $v0,$s0,$a0`
  - tj: `add $2,$16,$4`




- Instrukcija ima vrijednost:
  - `0x02041020` tj
  - `33820704`

# I format

- Load, store, branch i operacije sa konstantama koriste `I` format:




- Radi uniformnosti `op`, `rs`, `rt` su na istim pozicijama kao i za `R` format
  - `op` unikatno identificira instrukciju `I` formata, svi `op` sem `000000`, `00001x`, and `0100xx` su `I` instrukcije
  - `rs` adresni registar za load i store, a izvorni operand registar za druge instrukcije
  - `rt` destinacijski registar za sve instrukcije sem branch i store, za koje predstavlja izvorni registar
  - `Address` 16 bitna cjelobrojna vrijednost sa predznakom
    - u intervalu -32,768 to +32,767

# I format primjer konstanta

- Neka je data instrukcija:
  - `addi $v0,$s0,-34`
  - tj: `addi $2,$16,-34`

| 8 | 16 | 2 | -34 |
|---|---|---|---|
| 6 bita | 5 bita | 5 bita | 16 bita |

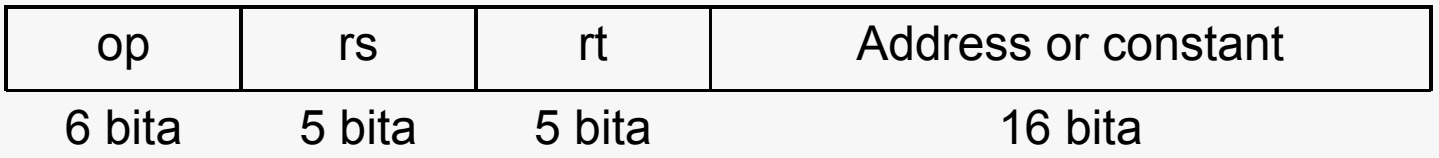
| 001000 | 10000 | 00010 | 1111111111011110 |
|---|---|---|---|
| 6 bita | 5 bita | 5 bita | 16 bita |

- Instrukcija ima vrijednost:
  - `0x2202ffde` tj
  - `570621918`

# 32 bitne konstante

- Za rad sa konstantama većim od 16 bit-a koristiti se instrukcija `lui`:
  - Prva dva byte-a destinacijskog registra `lui` instrukcije dobijaju vrijednost 16 bit-ne numeričke konstante u operaciji
  - Niža dva byte-a destinacijskog registra postavljaju se na vrijednost `0`
- Primer:
  - Da bi postavili vrijendnost `0xababefef` u registar `$s0` potrebno je dati dvije instrukcije:
    - `lui $s0,0xabab`
    - `ori $s0,$s0,0xefef`

# I format grananje




- Za branch instrukcije `bne` i `beq`, zadnjih 16 bit-a `I` formata predstavlja lokaciju (oznaku) na koju se vrši preusmjerenje toka programa
  - Branch se koristi za implementaciju `if`, `while`, i `for` C izraza u kojima skokovi u kodu nisu veliki tj vrši se mala promjena vrijednosti `PC` registra
  - Oznaka se u ovom slučaju mijenja relativnom adresom u memoriji spram vrijednosti adrese `PC` registra
  - Relativna adresa vrednuje se brojem riječi u odnosu na adresu slijedeće instrukcije tj `PC+4`

# I format grananje primjer

- Neka je dat kod:

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

```
Tijelo: addi $16,$16,-1
        slti $8,$17,2
        beq  $8,$0,-3
        slt  $8,$17,$16
        bne  $8,$0,-5
```

- Instrukcija `beq $8, $0,-3` se kodira kao:

| 4 | 8 | 0 | -3 |
|---|---|---|---|
| 6 bita | 5 bita | 5 bita | 16 bita |

| 000100 | 01000 | 00000 | 1111111111111101 |
|---|---|---|---|
| 6 bita | 5 bita | 5 bita | 16 bita |

# J format

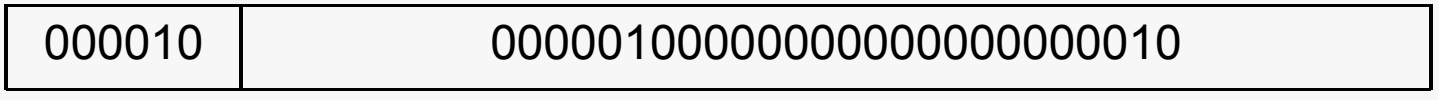
- Korisiti se za kodiranje `j` i `jal` instrukcija


- Pojedinačna polja reprezentiraju:
  - `op` radi uniformnosti identično kao za `I` format
  - `address` lokacija u memoriji na koju se vrši skok
- Adresa je apsolutna i specificira se u riječima a ne byte-ima
  - Skok modificira samo zadnjih 28 bita PC registra
    - tj skok se odvija unutar 256 MB memorijskih segmenata (64 mil. instrukcija)

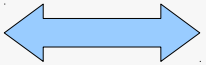
# J format primjer

```
 Adresa      Kod          Instrukcije 1                    Instrukcije 2

0x00400000  0x08100002   j 1048578                        j uslov
0x00400004  0x02118021   addu $16,$16,$17       tijelo:   addu $s0, $s0, $s1
0x00400008  0x00104080   sll $8,$16,2           uslov:    sll $t0, $s0, 2
0x0040000c  0x01134021   addu $8,$8,$19                   addu $t0, $t0, $s3
0x00400010  0x8d080000   lw $8,0($8)                      lw $t0, 0($t0)
0x00400014  0x1112fffb   beq $8,$18,-5                    beq $t0, $s2, tijelo
```


```
00000100000000000000000010 << 2
```



```
0x00400008
```