Arhitektura računara

dr.sc. Amer Hasanović

Registri, varijable i instrukcije

- C/C++ varijable imaju asociran tip npr `int`, `float`, `char` itd. na osnovu kojeg se određuje kako se skup bita asociran sa varijablom tretira kod operacija.
- registri nemaju tip, konkretna instrukcija određuje kako se sadržaj registra tretira kod izvršenja
- za razliku od C/C++, svaka linija asembli koda izvršava tačno jednu instrukciju
  - Format 1 za MIPS instrukcije:
  - `1 2, 3, 4`
    - `1` – ime instrukcije
    - `2` prvi operand u instrukciji ili destinacija
    - `3` i `4` treći I četvti operand ili izvori

Sabiranje i oduzimanje

- `add $s0,$s1,$s2 # a=b+c;`
  - gdje je: `a->$s0, b->$s1, c->$s2`
- `sub $s3,$s4,$s5 # d=e-f;`
  - `gdje je: d->$s3, e->$s4, f->$s5`
- Primjer:
  - `a = b + c + d - e;`
    - `add $t0, $s1, $s2 # $t0 = b + c`
    - `add $t1, $t0, $s3 # $t1 = $t0 + d`
    - `sub $s0, $t1, $s4 # a = $t1 – e`
- Registar `$0` ili `$zero` ima specijalni tretman
  - Ima vrijednost `0` i koristi se u izrazima gdje je potrebna vrijednost `0`
  - Nije moguće promjeniti vrijednost registra

Sabiranje sa konstantama

- identično operaciji `add` samo se koristi `addi` i zadnji argument je konstanta sa kojom se sabira npr:
  - `addi $s0,$s1,5 # f = a + 5`
- Koristi se i za oduzimanje npr:
  - `f=a-5`
- imamo:
  - `addi $s0,$s1,-5`

Prekoračenja

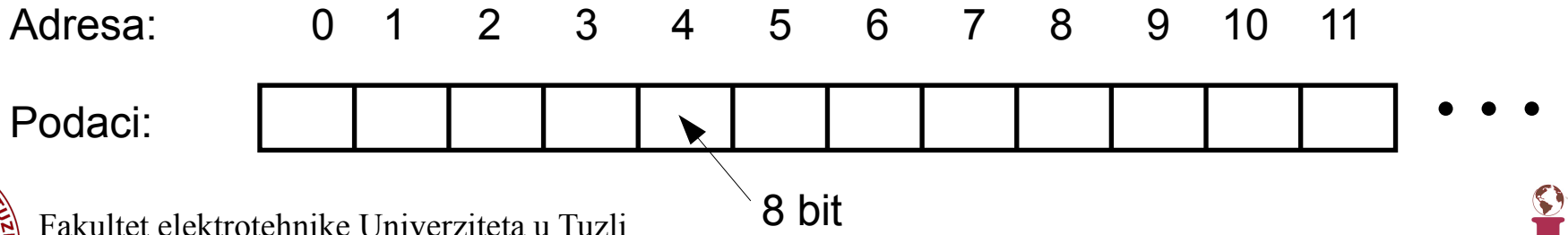
- Prekoračenja nastaju kada se dogodi greška u računanju usljed ograničenja u preciznosti računara:
- Primjer računica sa četiri bita:

```
+15     1111
 +3     0011
+18    10010
```

- MIPS nudi dvije varijante
  - Aritmetika sa detekcijom prekoračenja I generiranjem iznimke `add, sub, addi`
  - Aritmetika bez detekcije prekoračenja `addu,subu,addiu`

Memorija i registri

- Registara ima ograničeni broj i ograničeni su u količini podataka koje mogu sadržavati
  - Problematičan tretman struktura i podataka koji zahtijevaju više od 32 bita
- Iako značajno sporiji RAM je značajno jeftiniji i kompaktniji od registara
  - Programe organizirati na način da se što više koriste registri
  - Kompajleri inteligentno prilikom generiranja asembli koda reduciraju učestanost pristupa RAM-u
- MIPS memorija se adresira u byte-ima
  - Svaka adresa referecira vrijednost od 8 bit-a
  - Za kodiranje adrese koristi se 32 bit-a



Pristup memoriji

- MIPS koristi indeksirani pristup memoriji:
  - Za pristup memoriji potreban je registar i konstanta sa predznakom
  - Konkretna adresa u memoriji dobija se sabiranjem vrijednosti iz registra sa konstantom
  - Vrijednost iz registra se u ovom slučaju tretira kao pointer
  - Notacija:
    - `br(reg)`
- Format instrukcija 2:
  - `1 2,3(4)`
  - `1` – ime instrukcije
  - `3` – numerička konstanta
  - `2` i `4` su registri

Transfer byte-a mem → reg

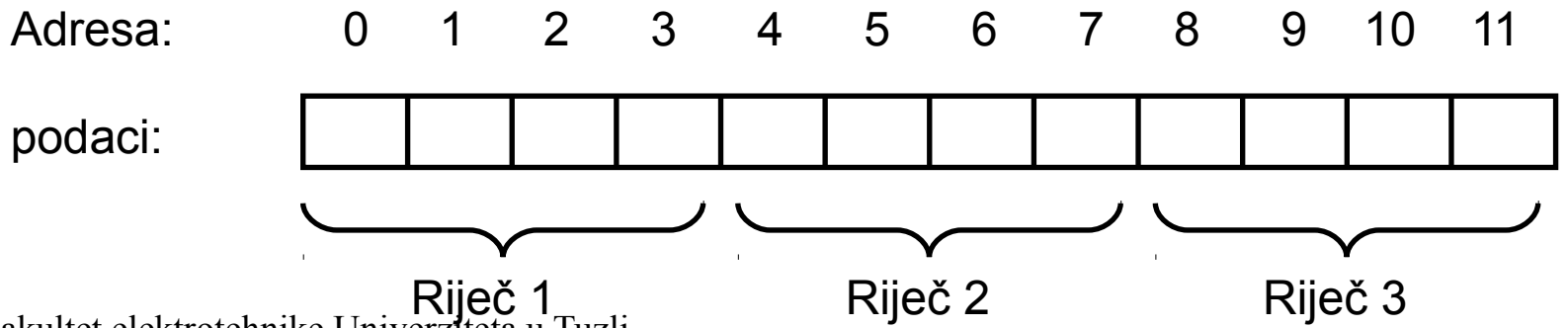
- Za transfer jednog byte-a iz memorije u registar koristi se instrukcija `lb` primjer:
  - `lb $t0, 20($a0)   # $t0 = M[$a0 + 20]`
- Instrukcija uzima pointer snimljen u registru `$a0`, na tu vrijednost dodaje broj `20` te sa memorijske adrese dobivene ovom sumom čita 1 byte podataka i isti zapisuje u registar `$t0`
  - `$a0` se zove i bazni (*base*) registar
  - `20` se označava kao ofset (*offset*)
- Operacija uzima u obzir predznak vrijednosti koja je u memoriji i zadržava ga u registru za slučaj da ovo nije poželjno može se koristiti instrukcija `lbu`

Transfer byte-a reg → mem

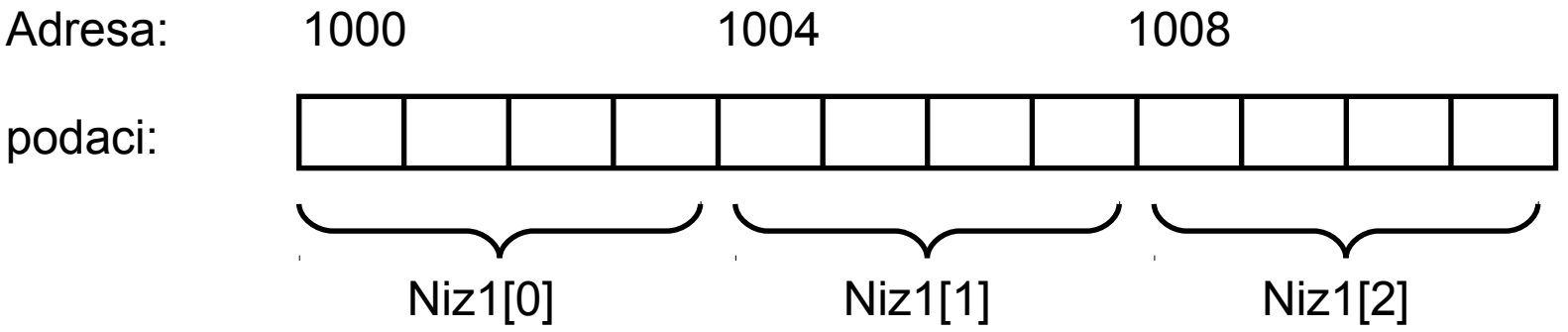
- Instrukcija za transfer iz registra u memoriju je `sb` i ima identičan format kao `lb`:
  - `sb $t0, 20($a0)`
- Instrukcija uzima pointer snimljen u registru `$a0`, na tu vrijednost dodaje broj `20` te u memorijsku adresu dobivenu ovom sumom upisuje `1` byte i to najniži byte iz registara `$t0`

Word pristup memoriji

- Programski jezici obično podržavaju tipove koji zauzimaju 32 bita npr `int`, `float`, `int*` itd...
- Sa stanovišta MIPS arhitekture 32 bit-a predstavljaju *riječ* (*word*) tj osnovu jedinicu za podatke
- MIPS dozvoljava pristup memoriji sa punom riječi instrukcijama `sw` i `lw` koje rade identične operacije kao `sb` i `lb` samo sa 32 bit-a.
- MIPS zahtijeva da ovakav pristup bude poravnat (*aligned*), tj kao adrese za ovakav pristup mogu se koristiti samo one koje su dijeljive sa `4`
  - U suprotnom generira se iznimka tipa *bus error*



Primjer word pristupa



- Neka je na memorijskoj lokaciji `1000` učitan niz cijelih brojeva `Niz1`, i neka je `$s3 = 1000`
- Neka je data linija c koda
  - `a = b + Niz1[3];`
- Linija ima ekvivalent u slijedećim instrukcijama:
- `lw $t0, 12($s3)`
- `add $s0, $s1, $t0`

Promjena toka programa

- Jedna od ključnih karakteristika računara je mogućnost evaluacija izraza i mogućnost promjene toka programa na osnovu rezultata:
  - C/C++:
    - `if-else, switch`
    - `while, for`
  - MIPS:
    - Grananje - instrukcije `bne, bqe`
    - Skokovi – instrukcija `j`
    - Kao destinacija za grananje ili skok koristi se oznaka (label) u MIPS kodu

Test jednakosti

- Test se izvodi putem instrukcije `beq` u formatu
  - `beq reg1, reg2, oznaka`
- Npr za c kod
  - `if (a == b) goto L1;`
  - `// uradi nesto`
  - `L1: // nastavak`
- Ekvivalentni MIPS kod je:
  - `beq $s0, $s1, L1`
  - `# uradi nesto`
  - `L1: # nastavak`

Test nejednakosti i skokovi

- Test se izvodi putem instrukcije `beq` u formatu
  - `bne reg1, reg2, oznaka`
- Npr za c kod
  - `if (a != b) goto L1;`
  - `// uradi nesto`
  - `L1: // nastavak`
- Ekvivalentni MIPS kod je:
  - `bne $s0, $s1, L1`
  - `# uradi nesto`
  - `L1: # nastavak`
- Bezuslovni skok na oznaku L1 izvodi se instrukcijom `j` i to u formatu:
  - `j L1`

Primjer

- Neka je dat C kod:

```
if (i == j)
  f = g + h;
else
  f = g – h;
```

- MIPS implementacija

```
      beq $s3, $s4, Tacno  # grananje ako i == j
      sub $s0, $s1, $s2    # f = g – h
      j Izlaz              # bezuslovan skok
Tacno: add $s0, $s1, $s2   # f = g + h
Izlaz:
```