

# Arhitektura i organizacija računara

## Računarska aritmetika

### \* Računari imaju široku primenu u gotovo svim oblastima ljudske delatnosti

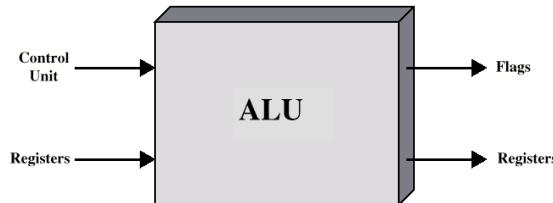
- Osnovna namena računara je u izračunavanjima u naučno-tehničkim aplikacijama
- Ove aplikacije zahtevaju računare visokih performansi kako u radu sa celobrojnim tako i u radu sa realnim podacima.
- Postizanje takvih performansi zahteva
  - Efikasne algoritme za obavljanje aritmetičkih operacija
  - Efikasnu implementaciju tih algoritama
  - Standardizaciju vezanu za način predstavljanja numeričkih podataka u računaru

### \* Govorićemo:

- Načinu predstavljanja numeričkih podataka
- Algoritmima za obavljanje osnovnih aritmetičkih operacija
  - Sabiranje/oduzimanje, množenje, deljenje

## Aritmetičko-logička jedinica

- \* Obavlja izračunavanja
- \* Sve ostalo je u službi ove jedinice
- \* Može biti sačinjena od više različitih funkcionalnih jedinica specijalizovanih za izvršenje određenih operacija



## Predstavljanje numeričkih podataka u računaru

- \* Numerički podaci se mogu klasifikovati prema različitim kriterijumima:
  - vrednostima:
    - celobrojni i realni,
  - osnovi brojnog sistema:
    - binarni, dekadni, heksadecimalni itd.,
  - tome da li se vodi računa o znaku podatka:
    - označeni i neoznačeni,
  - Prema dužinama podataka
    - za celobrojne podatke na
      - bajtove (8 bitova), polureči (16 bitova),
      - reči (32 bita) i dvostrukе reči (64 bita),
    - za realne podatke na
      - format jednostrukе preciznosti (32 bita),
      - format dvostrukе preciznosti (64 bita),
      - prošireni formati itd.

## Odvojeni podskupovi instrukcija

\* Za rad sa celobrojnim odnosno realnim podacima procesori koriste odvojene podskupove instrukcija.

- Naprimer, u MIPS arhitekturi, ADD je instrukcija sabiranja celobrojnih podataka, a
- ADD.S I ADD.D su instrukcije sabiranja realnih podataka predstavljenih u pokretnom zarezu u formatu jednostrukе i dvostrukе preciznosti, respektivno

## Binarna i dekadna aritmetika

\* Logička kola koja se koriste za gradnju računara i memorija, u skoro svim tehnologijama, mogu razlikovati samo dva stanja

- Zbog toga se za predstavljanje podataka mogu koristiti samo 2 vrednosti (cifre 0 i 1)
- Nema posebnih znakova za + i –
- Nema znaka za decimalnu tačku (zarez)

\* Normalni postupak obrade podataka u računaru obuhvata:

- Prevodjenje ulaznih podataka iz dekadnog brojnog sistema u binarni,
- Obradu,
- Prevod rezultata iz binarnog brojnog sistema u dekadni i prikaz korisniku

\* Prevod iz dekadnog brojnog sistema u binarni i obrnuto se ne može uvek izvršiti bez greške

\* Ima primena računara u kojima se ne dopušta ovo dvostruko prevođenje brojeva iz jednog u drugi brojni sistem, jer se time unose greške.

- U takvim slučajevima, podaci se u računaru predstavljaju u dekadnom obliku, tj. dekadni podaci se kodiraju sloganima binarnih cifara, BCD kodovima.
- Svaka dekadna cifra u broju se nezavisno kodira nizom binarnih cifara
  - Npr. broj 27.125 u prirodnom BCD kodu izgleda  
0010 0111. 0001 0010 0101

## Predstavljanje celih neoznačenih brojeva

- \* Celi neoznačeni brojevi se u računaru pamte u binarnom brojnom sistemu, pri čemu su na raspolaganju registri određene dužine.
  - Kada je zapis broja kraći od veličine registra (jedne memorijске reči), on se dopunjuje nevažećim nulama sa leve strane.
- \* Primer: Predstavti broj 25 u 32-bitnom registru računara.
  - Korak 1: prevesti broj u binarni brojni sistem
$$(25)_{10} = (11001)_2$$
  - Korak 2: prikazati sadržaj registra

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	

## Predstavljanje celih označenih brojeva

- \* Prosto kodiranje znaka
- \* Nepotpuni komplement
- \* Potpuni komplement
- \* sa pomerajem (polarizovani oblik)

## Prosto kodiranje znaka

\* Pozicija najveće težine se koristi za predstavljanje znaka broja

- 0 na poziciji najveće težine označava pozitivan broj
- 1 na poziciji najveće težine označava negativan broj

\* Primer: Predstavti brojeve 25 i -25 u 32-bitnom registru računara ako se za predstavljanje znaka koristi prosto kodiranje.

$$(25)_{10} = (11001)_2$$

25: 

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

-25: 

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

## Prosto označavanje

\* Problemi

- Neophodno je voditi računa i o znaku i o vrednosti broja kod sprovodenja operacija (sabiranja, oduzimanja, množenja,..)
- Dva načina za predstavljanje nule (+0 i -0)

## Komplementno predstavljanje brojeva

- \* Potreba da se jedinstveni hardver sabirača koristi za operacije sabiranja i oduzimanja, odnosno sabiranja označenih brojeva, dovela je do komplementnog predstavljanja znaka.
- \* Postoje dva tipa komplementa označenih podataka:
  - komplement osnove i
  - komplement najveće cifre.

## Komplement najveće cifre (Nepotpuni komplement)

- \* Brojevi se transformišu po sledećoj formuli:

$$A = \begin{cases} A, & A \geq 0 \\ q^n - 1 - |A|, & A < 0 \end{cases}$$

gde je:

*q – osnova brojnog sistema,  
n – ukupan broj pozicija predviđen za predstavljanje broja*

## Postupak nalaženja komplementa jedinice

- \* Broj se predstavi u binarnom brojnom sistemu i na poziciju najveće težine se upiše nula.
- \* Ukoliko je broj negativan, komplementira se svaki bit u zapisu.
  - Komplementirati – dopuniti do najveće cifre
  - Nula se zamenjuje jedinicom, a jedinica nulom
- \* Primer: Predstavi brojeve 25 i -25 u 32-bitnom registru računara ako se za predstavljanje znaka koristi komplement jedinice.

$$(25)_{10} = (11001)_2$$

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
25:	0 1 1 0 0 1

-25:	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
	1 0 0 1 1 0

## Jedinični komplement (nast.)

- \* Znak broja se pamti na poziciji najveće težine
  - 0 – broj je pozitivan
  - 1 – broj je negativan
- \* Nedostaci jediničnog komplementa
  - Postoje dva načina za prestavljanje nule:
    - 00...0 i
    - 11...1
  - Pri izvršavanju operacija sabiranja i oduzimanja, prenos sa pozicije najveće težine se dodaje na poziciju najmanje težine

## Komplement osnove (Potpuni komplement )

\* Brojevi se transformišu po sledećoj formuli:

$$A = \begin{cases} A, & A \geq 0 \\ q^n - |A|, & A < 0 \end{cases}$$

gde je:

*q – osnova brojnog sistema,*

*n – ukupan broj pozicija predvidjen za predstavljanje broja*

## Komplement dvojke

\* Ukoliko se koristi binarni brojni sistem, potpuni komplement se naziva

- Dvojični komplement ili
- Komplement dvojke.

\* Primer: Predstavti brojeve 25 i -25 u 32-bitnom registru računara ako se za predstavljanje znaka koristi komplement dvojke.

$$(25)_{10} = (11001)_2$$

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0  
25: 

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0  
-25: 

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

## Prednosti korišćenja dvoičnog komplementa

- \* Jedinstveni način prestavljanja nule
- \* Osnovne operacije se lako sprovode
- \* Izračunavanje komplementa broja se lako sprovodi
  - $3 = 00000011$
  - Jedinični komplement (Boolean komplement)  $11111100$
  - Dodati 1 na poziciju najmanje težine  $11111101$

## Predstavljanje nule

- \*  $0 = 00000000$
- \* Jedinični komplement  $11111111$
- \* Dodavanje 1 na LSB poziciju  $+1$
- \* Rezultat  $1\ 00000000$
- \* Prekoračenje se ignoriše, pa imamo:
  - $-0 = 0 \checkmark$

## Opseg brojeva

- \* Sa 8 bitova korišćenjem dvoičnog komplementa mogu se predstaviti brojevi u opsegu
  - $+127 = 01111111 = 2^7 - 1$
  - $-128 = 10000000 = -2^7$

## \* Sa 16 bitova

- $+32767 = 01111111 11111111 = 2^{15} - 1$
- $-32768 = 10000000 00000000 = -2^{15}$

## Konverzija sa manje na veću dužinu

- \* Pozitivnim brojevima se dodaju nule na početak (tj. Cifra znaka se kopira u više pozicije)
  - \*  $+18 = \quad 00010010$
  - \*  $+18 = 00000000 00010010$
- \* Kod negativnih brojeva se cifra znaka (1) kopira u više pozicije
  - \*  $-18 = \quad 10010010$
  - \*  $-18 = 11111111 10010010$

## Predstavljanje brojeva sa pomerajem (biased) ili polarizacijom

\* Brojevi se transformišu po sledećoj formuli:

$$A = A + p$$

gde je:

p – pomeraj čija je vrednost

obično  $q^{n-1}$  ili  $q^{n-1}-1$

q – osnova brojnog sistema,

n – ukupan broj pozicija predviđen  
za predstavljanje broja

## Primer

\* Primer: Predstavti brojeve 25 i -25 u 32-bitnom registru računara ako se za predstavljanje znaka koristi pomeraj.

$$(25)_{10} = (11001)_2$$

$$25: 11001+10000000000000000000000000000000$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	

$$-25: -11001+10000000000000000000000000000000$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1

Znak broja se pamti na poziciji najveće težine

1 – broj je pozitivan

0 – broj je negativan

## Primer predstavljanja označenih brojeva

Broj	Prosto kodiranje znaka	Jedinični komplement	Dvojični komplement	Polarizovani oblik ( $P=2^{n-1}-1$ )
+11	011	011	011	110
+10	010	010	010	101
+01	001	001	001	100
0	000 i 100	000 i 111	000	011
-01	101	110	111	010
-10	110	101	110	001
-11	111	100	101	000

Tabela 5.1 Prikaz dvocifrenih označenih binarnih brojeva na navedena četiri načina

## Sabiranje i oduzimanje celih brojeva

### \* Sabiranje se obavlja kao i u dekadnoj aritmetici

- Pošto se za predstavljanje brojeva koristi konačan broj cifra mora se voditi računa o prekoračenju (situacija kada je za predstavljanje rezultata potrebno više cifara nego što je na raspolaganju)

➤ Za testiranje prekoračenja u suštini dovoljno je nadgledati samo cifru znaka

### \* Oduzimanje se sprovodi isto kao sabiranje samo se umanjilac predstavlja u dvoičnom komplementu

- i.e.  $a - b = a + (-b)$

### \* I za sabiranje i za oduzimanje koristi se isto kolo

- Pri korišćenju komplemenata osnove pri sabiranju prenos iz pozicije znaka se ignorise (gubi).
- Pri korišćenju komplemenata najveće cifre prenos iz pozicije znaka naknadno se dodaje sumi na poziciju najmanje težine

## Prekoračenje

\* Do prekoračenja kod sabiranja može doći samo kada se sabiraju dva pozitivna ili dva negativna broja

- Suma dva pozitivna broja prevazilazi maksimalnu pozitivnu vrednost koja se može predstaviti korišćenjem n bitova :  
➢  $2^{n-1} - 1$
- Suma dva negativna broja je manja od najmanje negativne vrednosti koja se može predstaviti korišćenjem n bitova :  
➢  $-2^{n-1}$

\* Pravilo

- Ako se sabiraju dva broja istog znaka (oba pozitivna ili oba negativna) do prekoračenja dolazi ako i samo ako je rezultat suprotnog znaka.

## Prekoračenje - nastavak

\* Npr sa 4 bita može se predstaviti :

- 16 različitih vrednosti
- Do pozitivnog prekoračenja dolazi kada je rezultat  $> 7$   
➢ Npr.  $(4+5)_{10} \rightarrow 0100 + 0101 = 1001$
- Do negativnog prekoračenja dolazi ako je rezultat  $< -8$   
➢ Npr.  $(-4)+(-5)_{10} \rightarrow 1100 + 1011 = 1011$

\* Sa 8 bitova :

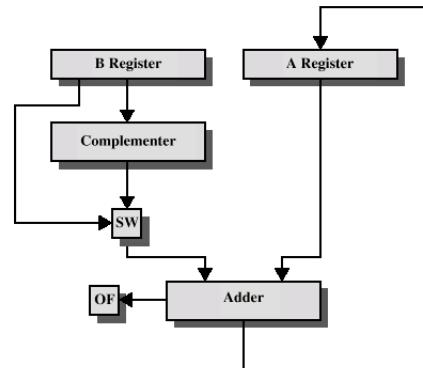
- 256 mogućih vrednosti
- Do pozitivnog prekoračenja dolazi kada je rezultat  $> 127$
- Negative prekoračenje nastupa kada je rezultat  $< -128$

## Upravljanje prekoračenjem

\* MIPS arhitektura na različite načine upravlja prekoračenjem kod sabiranja neoznačenih i označenih brojeva :

- Kod neoznačenih brojeva prekoračenje se jednostavno ignoriše, tj.
  - Implementira se aritmetika po **modulu  $2^n$**
- Kod sabiranja označenih brojeva generiše se prekid (**interrupt**)
  - OS poziva odgovarajuću rutinu za obradu prekida

## Hardver za sabiranje i oduzimanje



OF = overflow bit  
SW = Switch (select addition or subtraction)

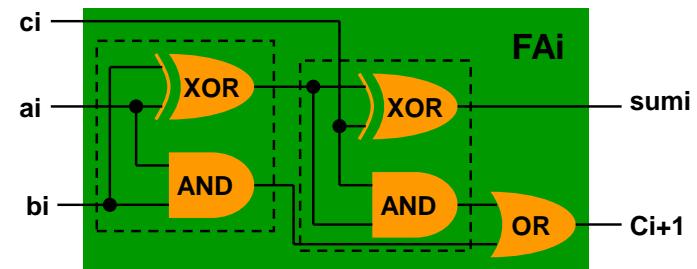
## Potpuni binarni sabirač

a	b	c <sub>i</sub>	s <sub>i</sub>	c <sub>i+1</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$s_i = \bar{a}b\bar{c}_{i-1} + a\bar{b}\bar{c}_i + \bar{a}\bar{b}c_i + abc_i = a \oplus b \oplus c_i$$

$$c_{i+1} = ab + ac_i + bc_i = ab + (a \oplus b)c_i$$

## Jedno-bitni potpuni binarni sabirač

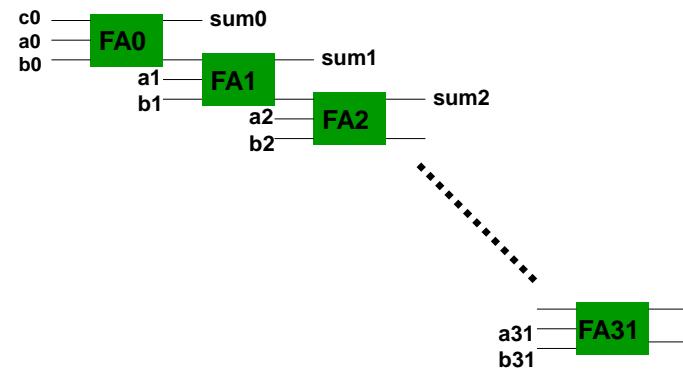


## n-to bitni paralelni binarni sabirač sa serijskim prenosom

\* Kaskadnim povezivanjem potpunih binarnih sabirača može se dobiti sabirač n-to bitnih brojeva

- Ovakav paralelni binarni sabirač naziva se **paralelni sabirač sa serijskim prenosom** (engl. ripple-carry adder, skr. RCA)

## 32-bitni sabirač sa serijskim prenosom



\* Maksimalno vreme sabiranja na n-to bitnom sabiraču određeno je vremenom prostiranja prenosa kroz svih n pozicija sabirača.

\* Neka je

- $T_1$ -vreme prostiranja prenosa kroz jednobitni potpuni sabirač sa dvostepenom logikom,
- $T_e$  -vreme prostiranja signala kroz jedan logički element,

onda je vreme sabiranja

$$T_s^{RCA} = nT_1 = 2nT_e$$

$$s_i = \bar{a}\bar{b}\bar{c}_{i-1} + a\bar{b}\bar{c}_i + \bar{a}\bar{b}c_i + abc_i = a \oplus b \oplus c_i$$

$$c_{i+1} = ab + ac_i + bc_i = ab + (a \oplus b)c_i$$

## Kako ubrzati sabiranje?

\* Problem je što se prenos generiše serijski

\* Moguće je usložnjavanjem logike za izračunavanje prenosa skratiti vreme

\* Uvedimo pomoćne funkcije

- $g_i = a_i \cdot b_i$   
➢ (tzv. Generator prenosa – jer ako je  $g_i=1$  u i-toj poziciji se generiše prenos za (i+1)-poziciju)
- $t_i = a_i \oplus b_i$ .  
➢ (Prosleđivač prenosa- jer se za  $t_i=1$  prenos iz (i-1)-pozicije prosleđuje u (i+1)-poziciju)
- izrazi za  $s_i$  i  $c_i$  mogu napisati u obliku:

$$\begin{aligned}s_i &= a_i \oplus b_i \oplus c_i = t_i \oplus c_i, \\ c_{i+1} &= a_i \cdot b_i + (a_i \oplus b_i) c_i = g_i + t_i \cdot c_i, \quad i=0,1,2,3.\end{aligned}$$

## Kako ubrzati sabiranje? –nast.

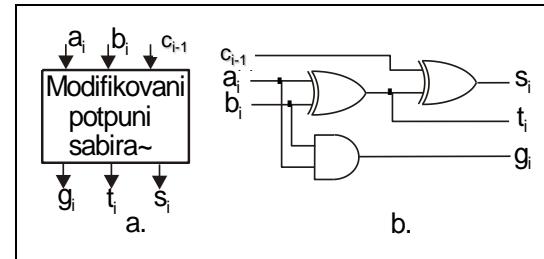
\* Na osnovu prethodnih izraza za  $s_i$  i  $c_{i+1}$ , vidi se da se prenos u poziciju najveće težine može izračunati samo na osnovu  $a_i$ ,  $b_i$  i  $c_0$ .

- Možemo projektovati sabirač sa paralelnim prenosom (engl. Carry Look-Ahead, CLA)

\* Npr za 4-bitni sabirač

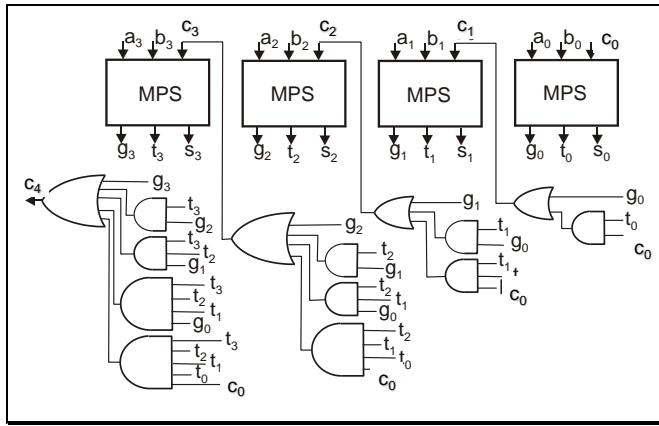
$$\begin{aligned}c_4 &= g_3 + t_3 \cdot c_3 = g_3 + t_3 \cdot (g_2 + t_2 \cdot c_2) = \\&= g_3 + t_3 \cdot (g_2 + t_2 \cdot (g_1 + t_1 \cdot c_1)) = \\&= p_3 + g_3 + t_3 \cdot (g_2 + t_2 \cdot (g_1 + t_1 \cdot (g_0 + t_0 \cdot c_0))) = \\&= g_3 + t_3 \cdot g_2 + t_3 \cdot t_2 \cdot g_1 + t_3 \cdot t_2 \cdot t_1 \cdot g_0 + t_3 \cdot t_2 \cdot t_1 \cdot t_0 \cdot c_0\end{aligned}$$

## Sabirač sa paralelnim prenosom



b.

## Logika za formiranje prenosa



Koliko je vreme sabiranja sa CLA sabiračem?

\* Vreme sabiranja ne zavisi od širine sabirača, tj. konstantno je, ali je obim logike veliki

- Zahteva se korišćenje I i ILI kola s velikim brojem ulaza

\* Kompromis – Sabirači sa izborom prenosa

- Vreme sabiranja na sabiraču sa serijskim prenosom može se skratiti ako se skrati put prostiranja prenosa kroz ovakav sabirač.
- To se može postići deobom  $n$  bitova sabiraka na kraće grupe bitova, recimo dužine  $k$ , tako da je  $n = m \cdot k$ .
- Ideja je da se u okviru svih  $m$  grupa sabiranje vrši paralelno.
- Dakle, umesto jednog  $n$ -bitnog sabirača imamo  $m$   $k$ -bitnih sabirača.

## Sabirači sa izborom prenosa

### \* Problemi

- U svakom k-bitnoim sabiraču, osim u grupi najmanje težine, ulazni podatak je i prenos iz prethodnog sabirača.
- Vrednost ovog prenosa poznata je tek po okončanju sabiranja u prethodnoj grupi.
  - Za paralelan rad svih m k-bitnih sabirača potrebno je ove vrednosti znati na samom početku sabiranja.

### \* Rešenje – udvajanje k-bitnih sabirača

- U svakoj od m-1 grupa koristiti po dva k-bitna sabirača sa serijskim prenosom:
  - jedan sa prepostavljenim prenosom 0 iz prethodne grupe, i
  - drugi sa prepostavljenim prenosom 1 iz prethodne grupe.
- Samo u grupi najmanje težine nije potrebno ovo udvajanje sabirača, jer je vrednost prenosa u ovu grupu unapred poznata.

## Sabirači sa izborom prenosa –nast.

### \* Iz kog od dva sabirača u okviru svake grupe bitova uzeti sumu?

- Iz onog sabirača koji za prepostavljeni prenos iz prethodne grupe ima vrednost koja je upravo određena sabiranjem u prethodnoj grupi.
- Ovaj izbor vrši se korišćenjem multipleksera tipa 2x1 u svakoj grupi

### \* Otuda i naziv sabirač sa izborom prenosa (engl. Carry-Select Adder, skr. CSA) za ovaj tip sabirača.

## Primer 32-bitni sabirači sa izborom prenosa

