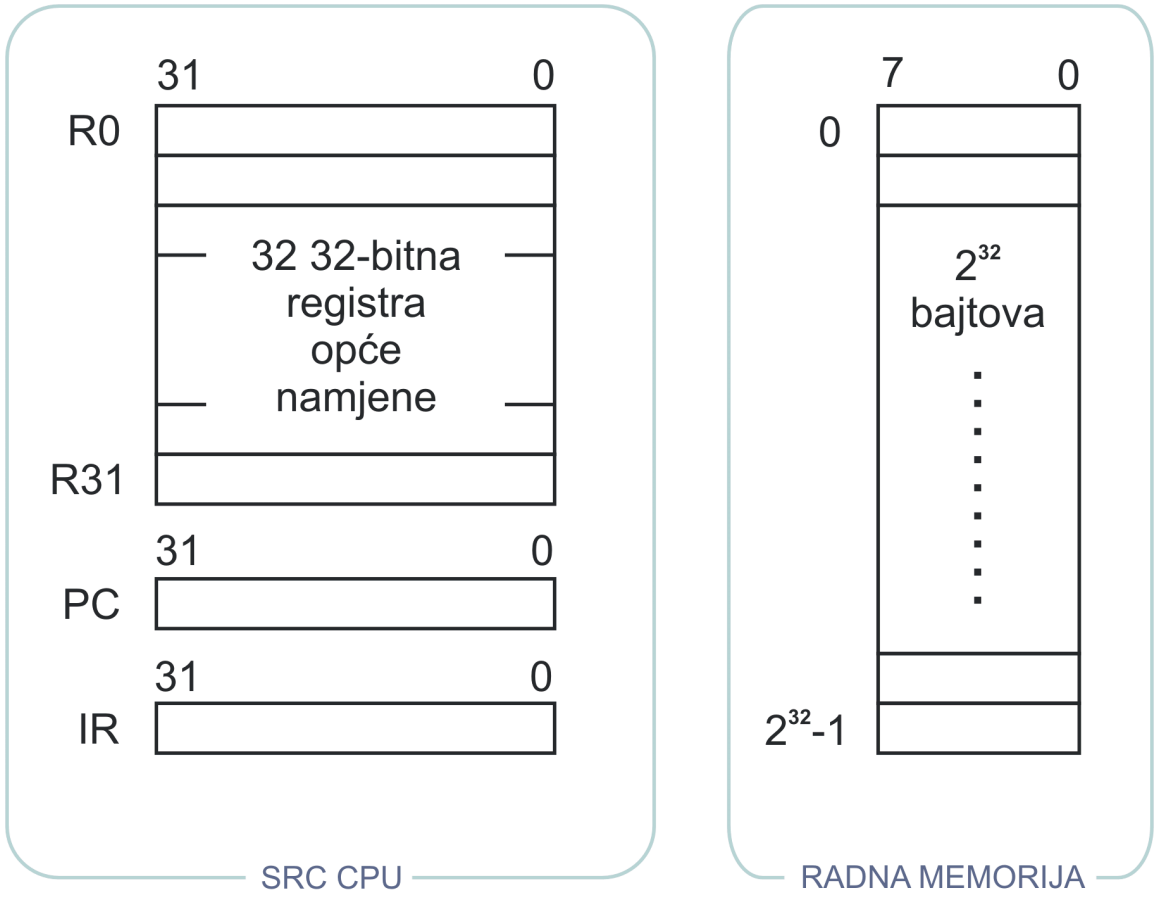
6. Instrukcijska arhitektura (S)RISC i x86

Programski model arhitekture (S)RISC

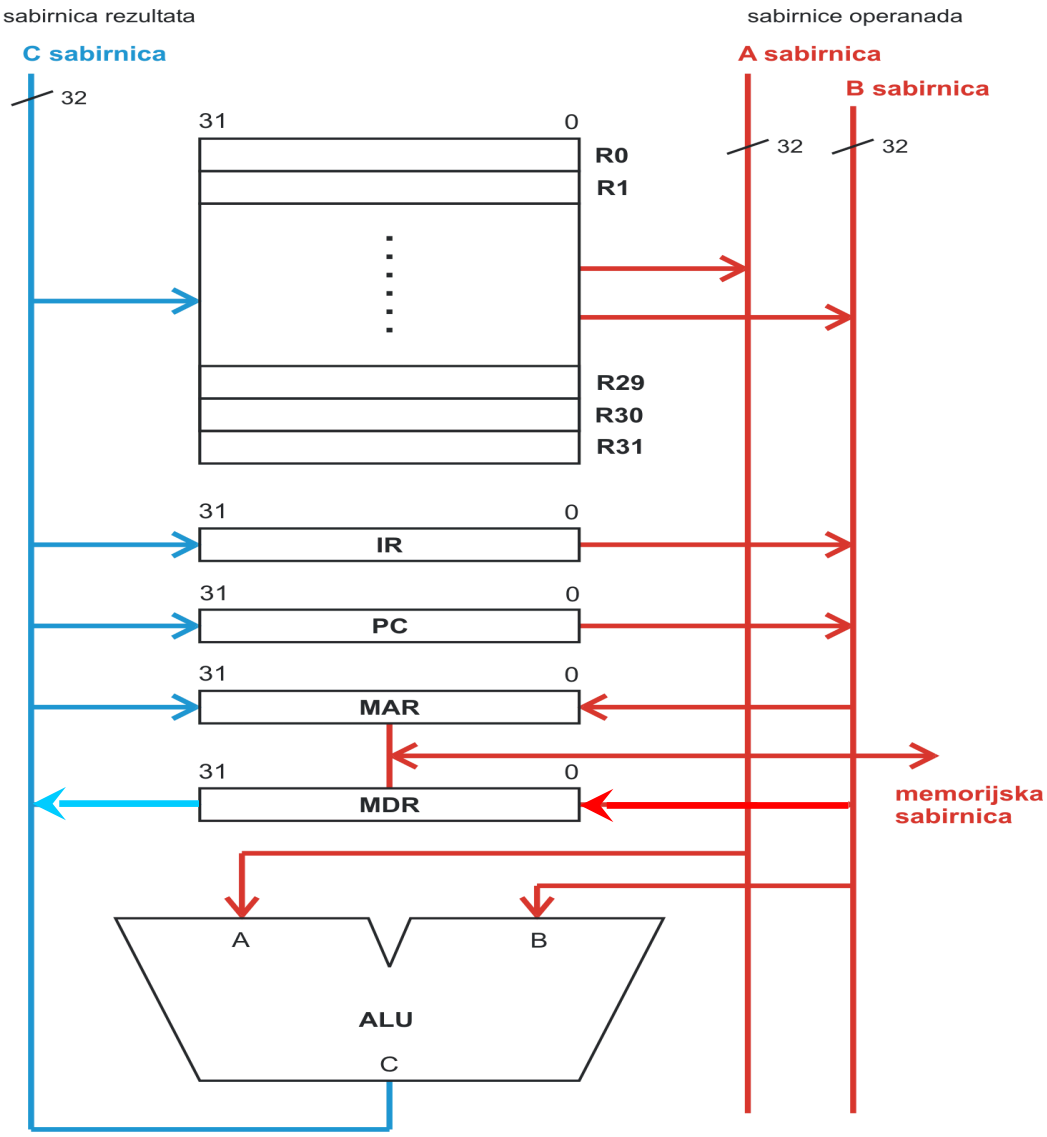


SRISC – Simple RISC

Značajke RISC arhitekture:

- jedine instrukcije za pristup memoriji su *load/store*
- najsloženije adresiranje je **registarsko indirektno s pomakom** (ili indeksni adresni način)
- pravilan (ortogonalan) instrukcijski skup (nema specijalnih registara, instrukcije imaju sličnu složenost)
- prioritet: maksimalno ubrzati najčešću operaciju: `add r1,r2,r3`
- SRISC: podržani samo poravnati 32-bitni memorijski operandi

Trosabirnički SRISC



Instrukcijski skup (S)RISC

ISA – Instruction Set Architecture

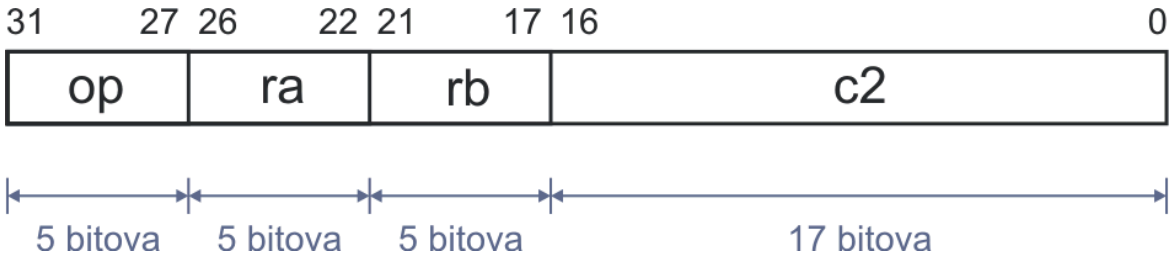
Tipične grupe instrukcija procesora arhitekture RISC:

A. memorijske operacije
B. aritmetičko logičke operacije
C. operacije grananja

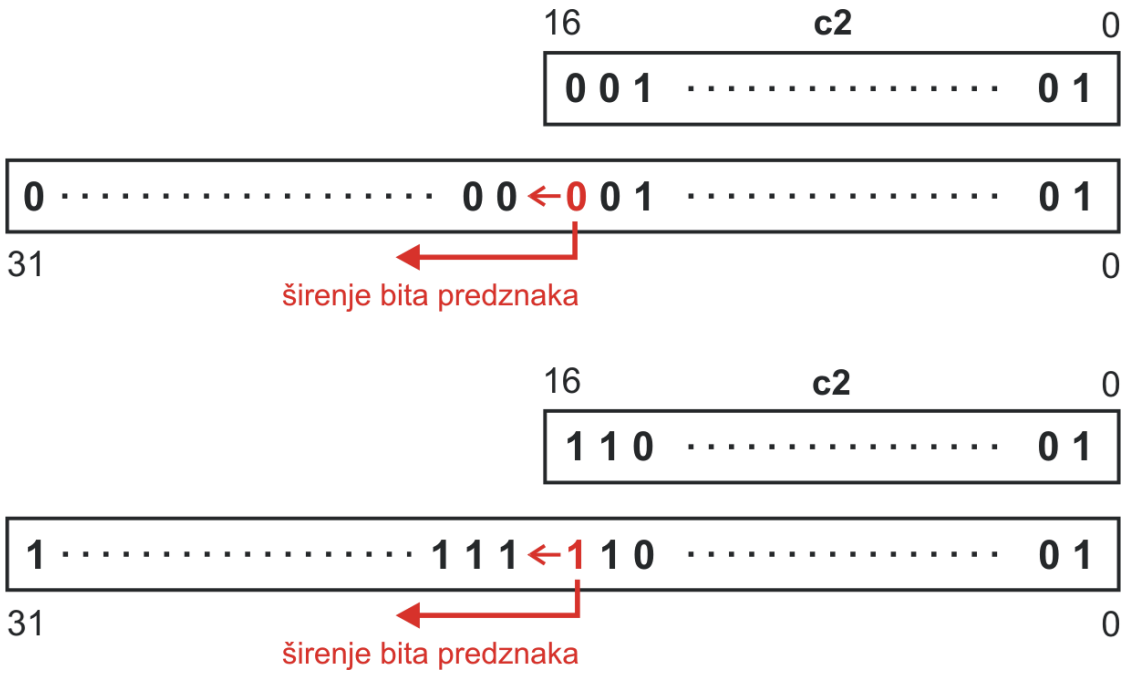
A. memorijske operacije

Instrukcija *load /mnemonik ld/*
- koristi se **instrukcijski format** 1 (slika dolje)

Funkcionalnost instrukcije:
- puni se registar određen 5-bitnim poljem *ra*
- na adresu izvorišta utječe 17-bitnom vrijednosti u polju *c2* koja se pretvara u 32-bitnu širenjem bita predznaka **(engl. sign-extension)**
- polje *rb* označava registar koji se pribraja konstanti *c2*
- specijalan slučaj, rb=0: adresi se pribraja nula



Širenje bita predznaka



ld ra, c2 ; izravno adresiranje (rb=0): R[*ra*] = M[*c2*]

ld ra, c2(rb) ; indeksno adresiranje: R[*ra*] = M[*c2* +R[*rb*]]

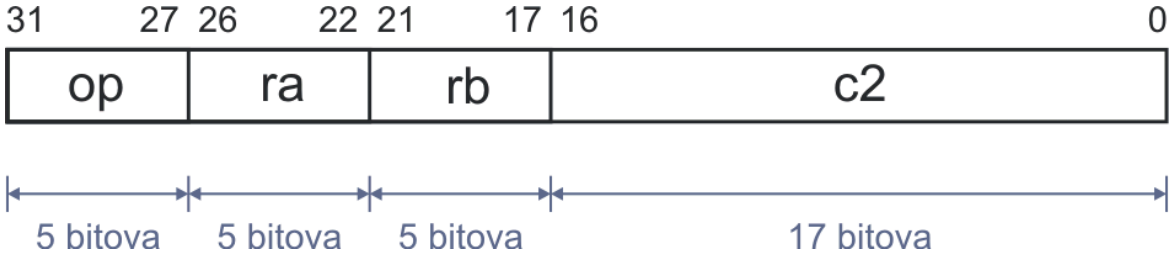
Instrukcija *store /mnemonik st/* (format 1)

Instrukcija *st* obavlja "obrnutu" operaciju u odnosu na *ld*:

1. Ako *rb* = 0 tada se R[*ra*] pohranjuje na lokaciji *c2*
   (uz širenje bita predznaka)

2. Ako je *rb* ≠ *0* tada se R[*ra*] pohranjuje na lokaciji *c2* + R[*rb*]

Instrukcije *load* i *store su* **jedine** instrukcije za rad s memorijskim podatcima!



Primjeri instrukcije store:

st ra, c2 ; izravno adresiranje M[*c2*] = R[*ra*]

st ra, c2(rb) ; indeksno adresiranje (*rb* $\neq$ 0): M[*c2* + R[*rb*]] = R[*ra*]

Instrukcija *load address* */mnemonik la/* (format 1)

računa adresu operanda (kao u prethodnim slučajevima), ali umjesto dohvata operanda izračunata adresa se pohranjuje u R[*ra*]

la ra, c2 ; napuni R[*ra*] s adresnim pomaknućem: R[*ra*] = *c2*

la ra, c2(rb) ; napuni R[*ra*] s adresnim pomaknućem: R[*ra*] = *c2* + R[*rb*]

la se koristi za sintezu složenijih načina adresiranja!

Zadatak:

Na slici je prikazan format instrukcije *la* (*la – load address*).

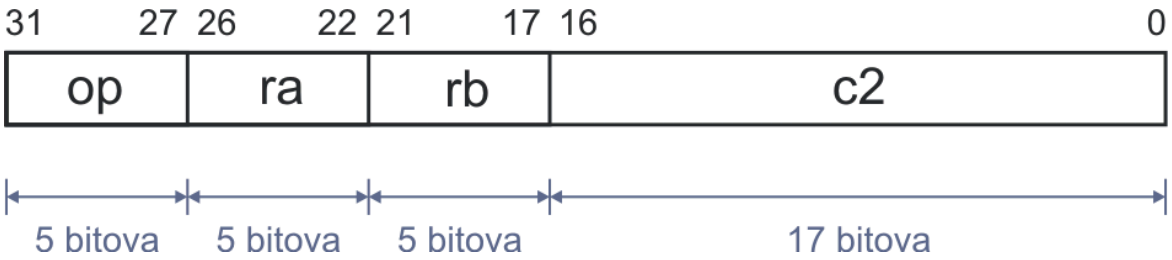
Odredite promjene sadržaja registara procesora za instrukciju

la r7,$100FF

Isto to napravite za instrukciju

la r7,$100FF(r4).

Napomena: Početni sadržaj registra `R[r4]` neka je $12.

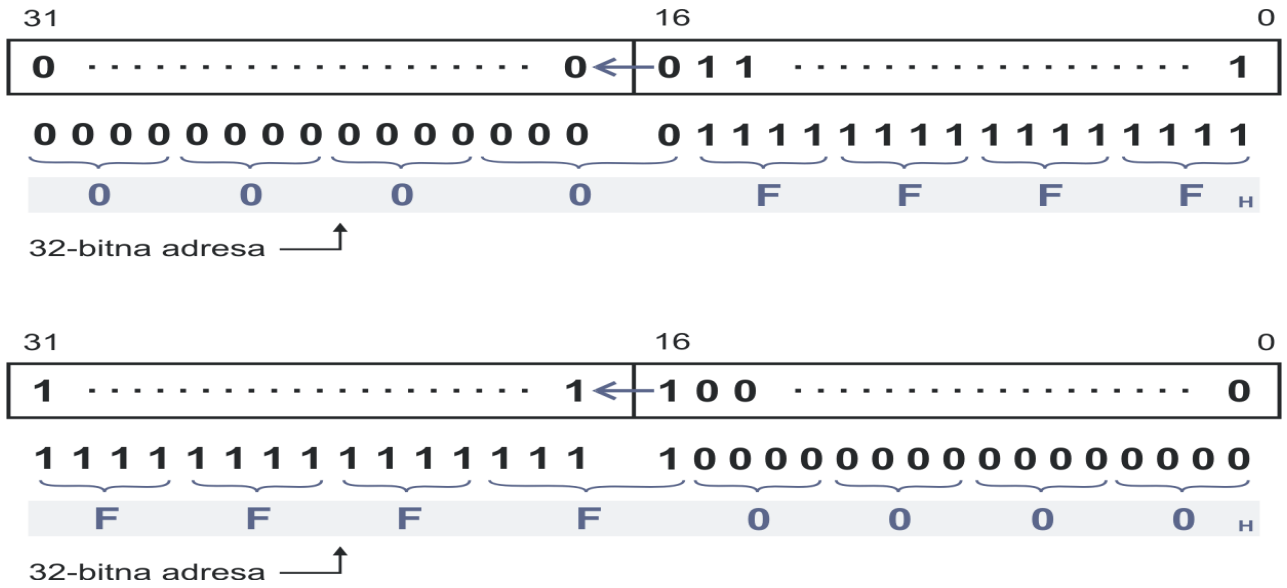


Ograničenje izravnog adresiranja: *c2* je predznačena 17-bitna konstanta!

Usputni operandi formata 1. se nalaze:

1) Na prvih 2^{16} bajtova:

2) Na posljednjih 2^{16} bajtova:



Rješenje:

1. R7=$FFFF00FF
2. R7=$FFFF0111

Dohvatljive adrese izravnim adresiranjem:

00000000 - 0000FFFF	1. prostor
FFFF0000 – FFFFFFF	2. prostor

Kako ostvariti pristup operandima drugdje u memoriji?

Za pristup operandima drugdje u memorijskom prostoru:

Registarsko indirektno adresiranje s pomakom:

R[*rb*] – baza

c2 – pomaknuće

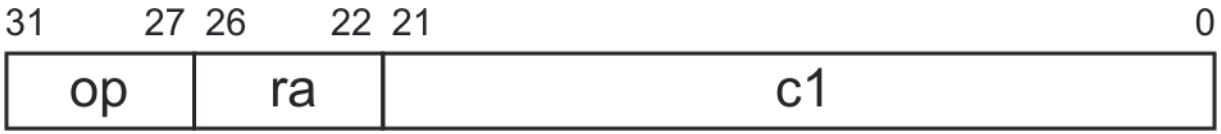
efektivna adresa = R[*rb*] + *c2*

Registarsko indirektno adresiranje (*c2* = 0):

efektivna adresa = R[*rb*] + 0

POZOR: U postupku računanja (zbrajanja) adrese prvo se 17-bitno pomaknuće treba pretvoriti u 32-bitni broj

Instrukcije s relativnim adresiranjem: *ldr, str, lar* (format 2)



ldr ra, c1 ; napuni registar R[*ra*] = M [PC + *c1*]
; relativni način adresiranja

str ra, c1 ; pohrani M [PC + *c1*] = R[*ra*]

lar ra, c1 ; napuni relativnu adresu: R[*ra*] = PC + *C1*

Premjestivost relativnih instrukcija:

- relativne adrese (u odnosu na PC) čine instrukcije premjestivim – programski moduli koji koriste takve instrukcije mogu se postaviti na bilo koju memorijsku adresu (position independent code, PIC)

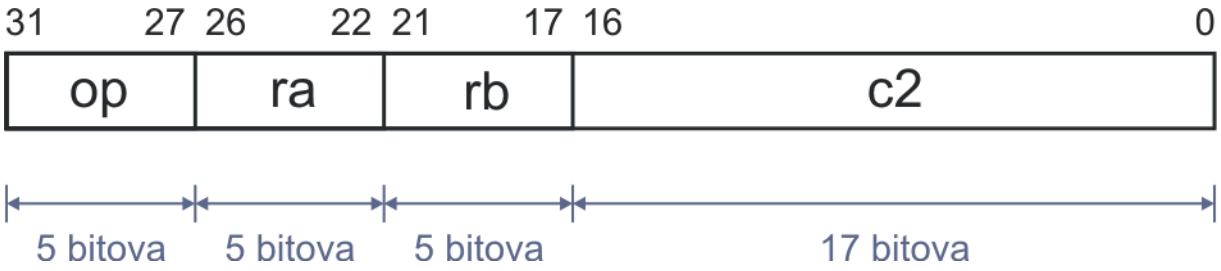
- *c1* je duljine 22 bita – može se specificirati adresa u prostoru od $\pm 2^{21}$ u odnosu na tekuću instrukciju

Primjeri instrukcija *load* i *store*

Instrukcija	op	ra	rb	c1	Komentar	Način adresiranja
`ld r1,32`	1	1	0	32	R[1] ← M[32]	Izravno
`ld r22,24(r4)`	1	22	4	24	R[22] ← M[24+R[4]]	Pomaknuće
`st r4,0(r9)`	3	4	9	0	M[R[9]] ← R[4]	Reg.Indirektno
`la r7,32`	5	7	0	32	R[7] ← 32	Usputno
`ldr r12,-48`	2	12	-	-48	R[12] ← M[PC-48]	Relativno
`lar r3,0`	6	3	-	0	R[3] ← PC	Registar

B. Aritmetičko logičke instrukcije

- instrukcije s usputnim (immediate) adresiranjem (format 1)



addi ra, rb, c2 ; R[*ra*] = R[*rb*] + *c2* : *c2 – usputna konstanta*

andi ra, rb, c2 ; R[*ra*] = R[*rb*] $\wedge$ *c2*

ori ra, rb, c2 ; R[*ra*] = R[*rb*] $\vee$ *c2*

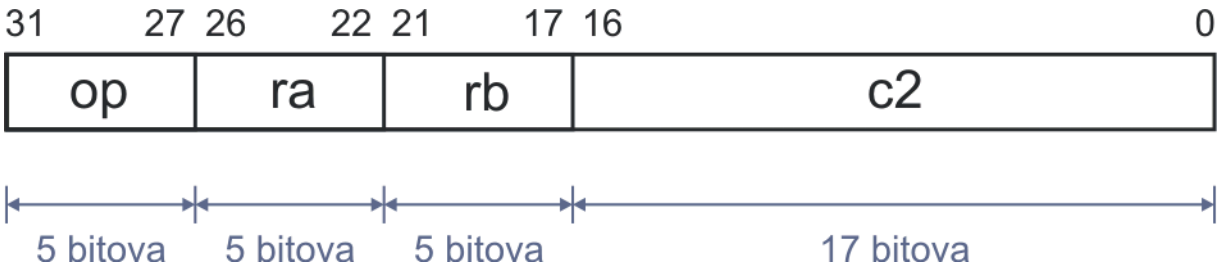
Zadatak:

Za instrukciju

addi r3, r4, $100FF

odredite sadržaje registara *r3* i *r4* nakon njena izvođenja.
Sadržaj registra *r4* je $0010000F

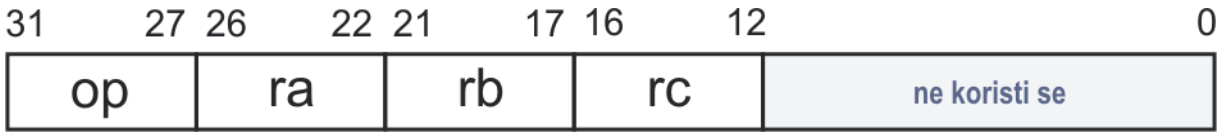
Pozor: 17-bitna konstanta koja se nalazi u polju konstante c2 se operacijom širenja bita predznaka pretvara u 32-bitnu vrijednost prije negoli se obavi aritmetička operacija.



Rješenje:

r3=$000F010E

Aritmetičke i logičke instrukcije s dva operanda (format 6)



add *ra, rb, rc* ; R[*ra*] = R[*rb*] + R[*rc*]

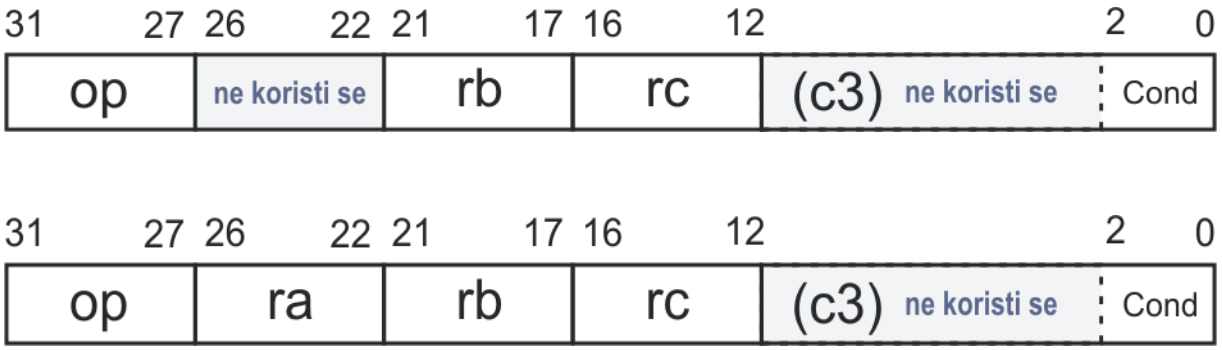
sub *ra, rb, rc* ; R[*ra*] = R[*rb*] - R[*rc*]

and *ra, rb, rc* ; R[*ra*] = R[*rb*] $\wedge$ R[*rc*]

or *ra, rb, rc* ; R[*ra*] = R[*rb*] $\vee$ R[*rc*]

C. Instrukcije grananja

br i *brl* (instrukcije koriste 4. i 5. format):



Instrukcija *br* u PC smješta R[rb], ako R[rc] zadovoljava uvjet:
PC ← R[rb], **ako** uvjet(c3, R[rc])

Instrukcija *brl* **prethodno** PC spremi u R[ra]:
R[ra] ← PC, **neovisno** o R[rc]
PC ← R[rb], **ako** uvjet(c3, R[rc])

Instrukcija *brl* (branch&link) se izvodi tako da kopira PC u povezni (engl. linkage) registar i to prije grananja

Povezni registar dopušta povratak iz potprograma i rabi se za implementaciju procedura i funkcija u HLL

PC se kopira u povezni registar bez obzira na to da li će se grananje izvesti (ili ne)

Instrukcije grananja (uvjeti!):

zbirni jezik	*c3*<2...0>	Uvjeti grananja
brnv, brlnv	*0*	*Nikada*
br, brl	*1*	*Bezuvjetno*
brzr, brlzr	*2*	*Ako je* R[*rc*] = 0
brnz, brlnz	*3*	*Ako je* R[*rc*] ≠ 0
brpl, brlpl	*4*	*Ako je* R[*rc*]<31> = 0
brmi, brlmi	*5*	*Ako je* R[*rc*] < 0

Primjer:

brl ra, rb,rc, c3 ; R[*ra*] ← PC i granaj na ciljnu
; adresu određenu s R[*rb*]
; ako sadržaj registra *rc*
; zadovoljava uvjet *c3*

ili

brlzr r7, r5, r1 ; R[*7*] ← PC i ako je R[1] = 0
; tada PC ← R[5]

Instrukcijska arhitektura (S)RISC, sažetak

- kompaktan i ortogonalan zapis
  (5 bitova operacijskog koda, samo 6 formata)
- podsjeća na vertikalno mikroprogramiranje s promjenljivim formatom mikroinstrukcije
- pristup memoriji instrukcijama load/store
  - registarsko indirektno adresiranje s pomakom
  - relativno adresiranje
- tri glavna razreda instrukcija: memorija, aritmetika, grananje
- nema množenja, dijeljenja, FP

Zastupljenost instrukcija u SPEC-ovim kolekcijama (MIPS):

Instruction class	MIPS examples	SPEC2006 Int	SPEC2006 FP
Arithmetic	`add, sub, addi`	16%	48%
Data transfer	`lw, sw, lb, lbu, lh, lhu, sb, lui`	35%	36%
Logical	`and, or, nor, andi, ori, sll, srl`	12%	4%
Cond. Branch	`beq, bne, slt, slti, sltiu`	34%	8%
Jump	`j, jr, jal`	2%	0%

[Patterson07]

Instrukcijska arhitektura x86 (IA-32)

Povijest:

- 8080 (1974): 8-bitni procesor
- 8086 (1978): 16-bitno proširenje 8080
  - prva implementacija arhitekture x86
  - adresni prostor 1MB dobiven segmentiranjem
- 8087 (1980): matematički koprocesor
  - FP instrukcije, registarski stog
- 80286 (1982): 24-bitne adrese
  - zaštita memorijskog prostora
- 80386 (1985): 32-bitno proširenje (novo ime: IA-32)
  - prvi “pravi” Intelov procesor
  - nesegmentirani memorijski prostor, straničenje

x86 u 32-bitnom dobu (1985-2003)

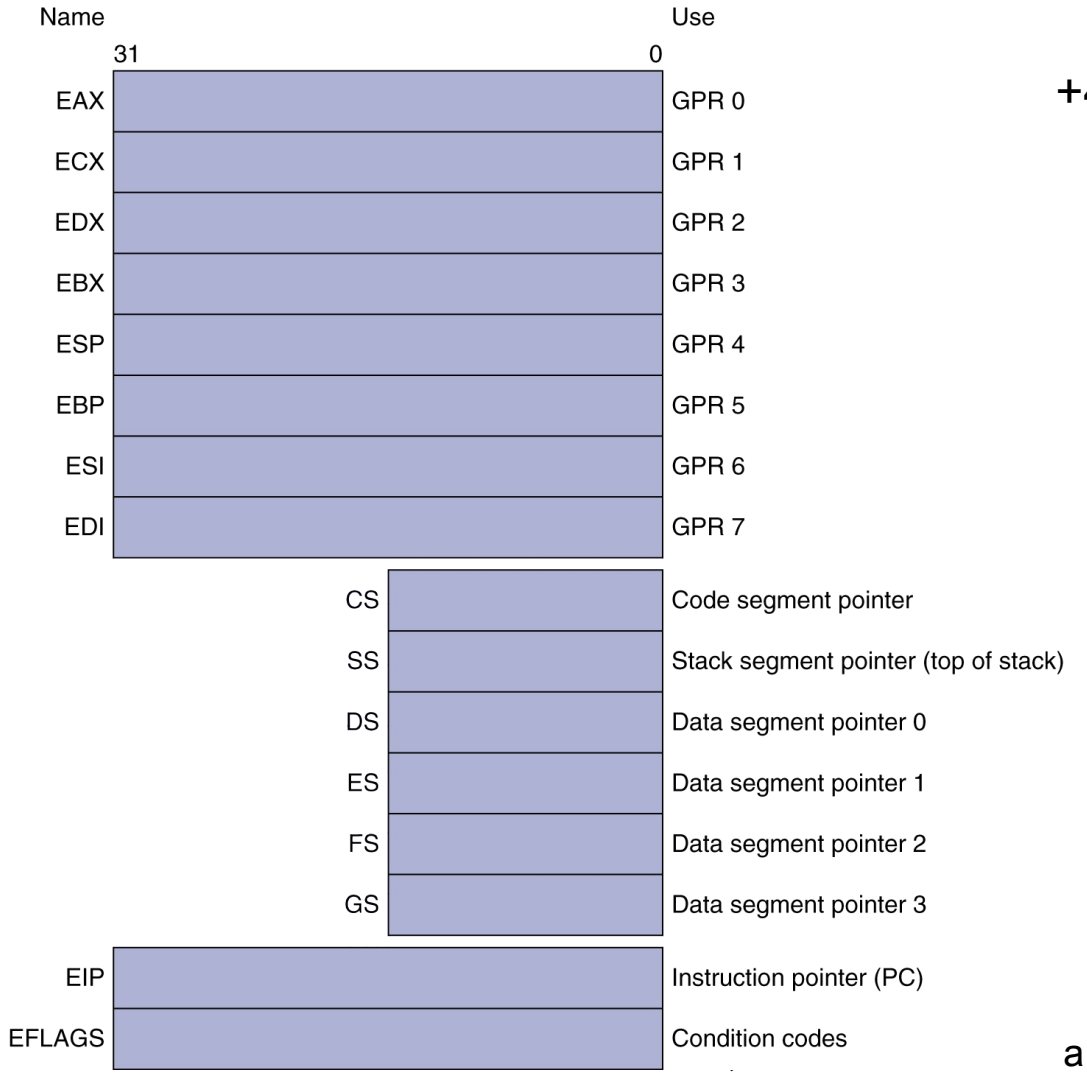
- Evolucija se nastavlja…
  - i486 (1989): protočnost, priručna memorija, koprocesor na čipu
    - pojava konkurencije: AMD, Cyrix
  - Pentium (1993): superskalarnost (bez OoO)
    - vektorske instrukcije (MMX)
  - Pentium Pro (1995), Pentium II (1997), Pentium III (1999)
    - sasvim nova organizacija (P6), SSE
    - drugi Intelov pogodak
  - Pentium 4 (2001)
    - ponovo nova organizacija
    - problemi s disipacijom, preduboka protočnost
    - SSE2

x86 u 64-bitnom dobu

- Saga se nastavlja…
  - AMD Opteron (2003)
    - AMD64: programski model IA32 proširen na 64 bita
    - širi registri, više registara, relativno adresiranje, veći adresni prostor, ...
  - Pentium 4 Netburst (2004)
    - EM64T – Extended Memory 64 Technology (de facto AMD64), SSE3
  - Intel Core (2006)
    - SSE4, VT-X (podrška za virtualizaciju)
  - Najavljena proširenja:
    - SSE5: AMD64, 2007
    - AVX (Advanced Vector Extension): Intel, 2008

- **Zaključak**: Intel ne može odustati od zastarjele arhitekture, jer bi ga inače suparnici istisnuli!
  $\Rightarrow$ x86 će još *dugo* biti s nama

Programski model IA32



+4GB bajtno orijentirane memorije

[Patterson07]

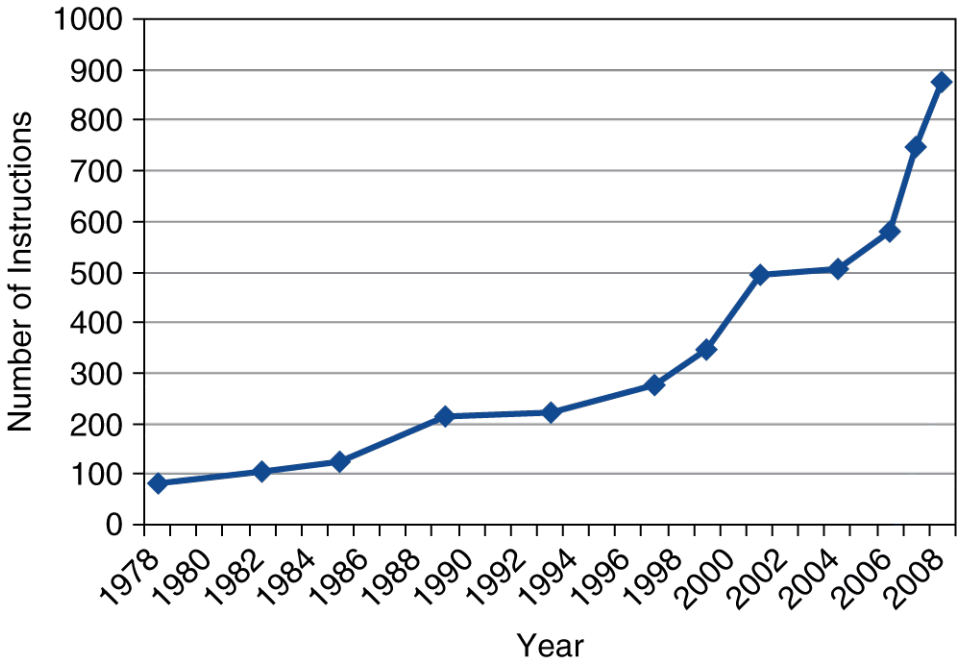
IA32: osnovni format instrukcija

- tipično dva operanda: `instr dst, src`
  - `dst = instr(src)`
    npr: `mov eax, [ebp+8]`
    ≡ `eax ← mem[ebp+8]`
  - `dst = instr(dst, src)`
    npr: `add eax, [ebp-4]`
    ≡ `eax ← eax + mem[ebp-4]`
  - operand `dst` registar ili memorija,
    ali najviše jedan memorijski operand

operand dst	operand src
registar	registar
registar	konstanta
registar	memorija
memorija	registar
memorija	konstanta

- načini adresiranja memorije
  - adresa = R_x
  - adresa = R_b + pomak
  - adresa = $R_b + 2^s \times R_{index}$ (s = 0, 1, 2, or 3)
  - adresa = $R_b + 2^{scale} \times R_{index}$ + pomak

Broj instrukcija (x86, IA32, x86-64)



[Patterson07]

- naravno, kompatibilnost funkcionira samo unatrag: Xeon može izvršavati programe iz 1978, ali 8086 ne može izvršavati programe iz 2009

Moderne implementacije IA-32

- CISC ISA komplicira implementaciju
  - sklopovlje u fazi PRIBAVI prevodi makro instrukcije x86 u jednostavne “interne” instrukcije (tzv. μoperacije, uop)
    - jednostavne makro instrukcije: 1 : 1
    - složene makro instrukcije: 1 : nekoliko
    - složenije makro funkcije: mikroprogramirano upravljanje (sporo!)
  - organizacija jedinice koja izvodi interne instrukcije vrlo slična superskalarnom RISC CPU
  - udio na tržištu omogućava konkurentnost pristupa
- RISC pristup ima komparativnu prednost
  - prevoditelji izbjegavaju složene instrukcije