

5

UKAZI

BRANKO ŠTER

PO KNJIGI - DUŠAN KODEK: ARHITEKTURA IN
ORGANIZACIJA RAČUNALNIŠKIH SISTEMOV

UKAZI

1

Prevajanje ukazov

Prevajalnik programe, napisane v višjem programskem jeziku, prevede v zbirni jezik (zbirnik pa nato v strojni jezik), ali pa kar neposredno v strojni jezik

➤ Primer 1 (iz jezika C v zbirni jezik):

```
a = b + c; // predpostavimo, da je a v r1, b v r2 in c v r3  
add  r1, r2, r3 ; r1 ← r2 + r3
```

➤ Primer 2:

```
a = b + c + d + e; // r1: a, r2: b, r3: c, r4: d, r5: e  
add r1, r2, r3  
add r1, r1, r4  
add r1, r1, r5
```

UKAZI

2

➤ Primer 3:

```
A[12] = h + A[8];           // r1: A, r3: h
```

```
lw  r2, 32(r1)      ; r2 ← M[r1+32]
```

```
add r2, r2, r3      ; r2 ← r2 + r3
```

```
sw  r2, 48(r1)      ; M[r1+48] ← r2
```

➤ Operand je lahko tudi konstanta

- **takojšnji (immediate) operand**

```
addi r1, r2, 5           ; r1 ← r2 + 5
      (add immediate)
```

UKAZI

3

Splošne lastnosti ukazov

➤ Vsak ukaz vsebuje

- Informacijo o operaciji, ki naj se izvrši (operacijska koda)
- Informacijo o operandih, nad katerimi naj se izvrši operacija

➤ Ukaz je shranjen v eni ali več (sosednih) pomnilniških besedah

➤ Format ukaza pove, kako so biti ukaza razdeljeni na operacijsko kodo in operande

UKAZI

4

5 dimenzij lastnosti ukazov

Dimenzija

1. Način shranjevanja operandov v CPE
2. Število eksplicitnih operandov v ukazu
3. Lokacija operandov in načini naslavljanja
4. Operacije
5. Vrsta in dolžina operandov

UKAZI

5

D1. Načini shranjevanja operandov v CPE

➤ 3 načini shranjevanja operandov v CPE:

1. Akumulator

- najstarejši način
- edini register
 - zato ga v ukazih ni treba eksplicitno navajati
- ukaza LOAD, STORE za prenos v in iz akumulatorja
- veliko prometa z GP (shranjevanje vmesnih rezultatov), zato počasnost

UKAZI

6

2. Sklad (stack)

- v danem trenutku je dostopna samo najvišja lokacija
 - podobno kot sklad pladnjev
- LIFO
- ukaza PUSH, POP (ali PULL)
- podobno akumulatorju (tako dostopen le 1 operand)
 - preprosta realizacija, kratki ukazi, preprosti prevajalniki
 - vendar je prostora za več operandov

UKAZI

7

3. Množica registrov (register set)

- Najbolje (danes edina rešitev)
 - nekdanj dragi, pa tudi prevajalniki jih niso znali dobro uporabljati
- Register je skupina pomnilniških celic, ki imajo skupne krmilne signale
 - Vsak register ima svoj naslov
- Namen: shranjevanje vmesnih rezultatov
 - pri skladu: v pomnilnik
- 2 rešitvi:
 - splošnonamenski registri (vsi ekvivalentni)
 - 2 skupini: za operande, za naslove
- 2 vrsti:
 - programsko nedostopni
 - programsko dostopni
 - programer jih lahko uporablja kot nek hiter pomnilnik

UKAZI

8

- **Programsko dostopni registri**
 - majhen pomnilnik, v katerega lahko shranimo operande
 - prednosti pred GP:
 1. Hitrost
 - registri so hitrejši od GP
 - bližji so aritmetično-logični in kontrolni enoti
 - možen je istočasen dostop do več registrov naenkrat
 2. Krajši ukazi
 - krajši naslov (ker je registrov malo) kot pri GP

D2: Število eksplicitnih operandov v ukazu

- **m-operandni računalnik**
 - običajno se podajajo naslovi operandov
 - danes m največ 3
- 4 skupine:
 - **3-operandni**

$$OP3 \leftarrow OP2 + OP1$$
 - operandi so običajno v registrih

■ 2-operandni

- enostavnejši, a malo počasnejši

$$OP2 \leftarrow OP2 + OP1$$

■ 1-operandni

- akumulator

$$AC \leftarrow AC + OP1$$

- mikroprocesorji iz 70. in 80. let
 - Intel 8080, Motorola 6800, Zilog Z80
 - Intel 8086, Intel 80186, Intel 80286

■ Brez-operandni (skladovni)

- najkrajši ukazi

$$Sklad_{VRH} \leftarrow Sklad_{VRH} + Sklad_{VRH-1}$$

- toda: potrebna sta vsaj 2 ukaza z ekspl. operandom!
 - PUSH, POP (prenos med GP in skladom)

D3: Lokacija operandov in načini naslavljanja

➤ 2 vprašanji:

- Kje so operandi?
- Kako je v ukazu podana informacija o njih?

➤ Lokacija operandov

- registri CPE
- GP
- (registri krmilnika V/I naprave)

UKAZI

13

➤ 2- in 3-operandni računalniki se delijo še na:

- **registrsko-registrske** računalnike
 - najbolj razširjeni
 - vsi operandi v registrih CPE
 - reče se tudi **load/store** računalniki (ker rabimo load in store)
- **registrsko-pomnilniške**
 - en operand v registru, drugi *lahko* v pomnilniku
- **pomnilniško-pomnilniške**
 - vsak operand *lahko* v pomnilniku
 - zapleteni ukazi, CISC (npr. VAX)

UKAZI

14

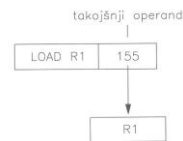
Načini naslavljanja

➤ Načini naslavljanja: Kako je v ukazu podana informacija o operandih

- Tičejo se predvsem pomnilniških operandov
 - pri registrskih je enostavno

1. Takojšnje naslavljanje (immediate addressing)

- operand je v ukazu podan z vrednostjo (je del ukaza)
- **takojšnji operandi (literali)** so kar konstante
 - `LOAD R1,#155, (R1 ← 155)`
 - `ADD R1,#3 (R1 ← R1 + 3)`



UKAZI

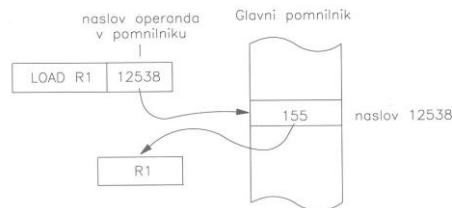
15

2. Neposredno naslavljanje (direct addressing)

- operand je podan z naslovom
 - če je to naslov registra, je to **registrsko naslavljanje**
 - če je to naslov v GP, je to **(neposredno) pomnilniško naslavljanje**
- primerno za operande, ki se jim ne spreminjajo naslovi

Registrsko: `ADD R1, R2`

Pomnilniško: `LOAD R1, (12538)` ali pa `ADD R1, (1001)`



UKAZI

16

- Težave:

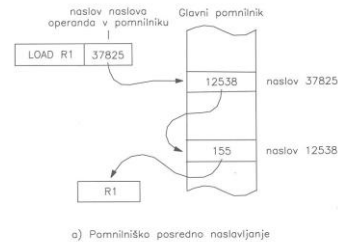
- velik naslovni prostor → dolg naslov → dolgi ukazi
- povečanje pom. prostora → drugačni ukazi → nezdružljivost za nazaj
- primeri, ko operand ni na stalnem naslovu

3. Posredno naslavljanje (indirect addressing)

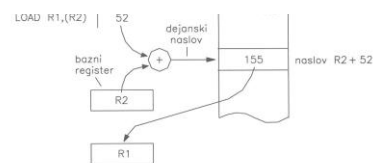
- v ukazu je naslov lokacije, na kateri je shranjen naslov operanda
 - **Pomnilniško posredno naslavljanje**, če gre za naslov pomnilniške lokacije (nerodno, ni pogosto)
 - $\text{ADD R1},@(1001) \quad R1 \leftarrow R1 + M[M[1001]]$
 - **Registrsko posredno naslavljanje**, če gre za naslov registra
 - uporablja se tudi **odmik** (displacement)
 - iz obojega se izračuna pomnilniški naslov
 - imenuje se tudi **relativno naslavljanje**
 - naslov operanda določen relativno na vsebino registra
 - najpogostejši način naslavljanja

Posredno naslavljanje:

pomnilniško



registrsko



UKAZI

19

Glavne vrste relativnega naslavljanja

3.1 Bazno naslavljanje (base addressing)

- reče se tudi **naslavljanje z odmikom** (displacement addressing)
- najpogostejše
- naslov operanda $A = R2 + D$
 - k vsebini registra R2 prištejemo odmik D
- R2 je **bazni register**, A pa **dejanski naslov** (effective address)
- Npr.: `ADD R1,100(R2)` $R1 \leftarrow R1 + M[R2+100]$
- Če $D=0$: Bazno brez odmika
 - `ADD R1,(R2)` $R1 \leftarrow R1 + M[R2]$

UKAZI

20

3.2 Indeksno naslavljanje (indexed addressing)

- odmik D
- $A = R2 + R3 + D = R2 + D_1$
- R3 je indeksni register
- glavno področje uporabe so polja, strukture in sezname
 - elementi se običajno obdelujejo zaporedoma po naraščajočih (ali padajočih) indeksih, zato sta pogosti operaciji

$$R3 \leftarrow R3 + \Delta \quad \text{in} \quad R3 \leftarrow R3 - \Delta$$
 - Δ je dolžina operanda, merjena v številu pomnilniških besed (*korak indeksiranja*)
- Npr.:
 - $\text{ADD } R1, 100(R2+R3), R1 \leftarrow R1 + M[R2+R3+100]$ (dostop do elementov polja)

3.3 Pred-dekrementno naslavljanje (pre-decrement addressing)

- $R3 \leftarrow R3 - \Delta$
- $A = R2 + D$ ali $A = R2 + R3 + D$
- bazno ali indeksno

3.4 Po-inkrementno naslavljanje (post-increment addressing)

- $A = R2 + D$ ali $A = R2 + R3 + D$
- $R3 \leftarrow R3 + \Delta$

3.5 Velikostno indeksno naslavljanje (scaled indexed addressing)

- $A = R2 + R3 \times \Delta + D$
- dovolj je inkrementirati R3

- Pred-dekrementno in po-inkrementno naslavljanje v paru tvorita **skladovno naslavljanje** (stack addressing)
 - sklad je v GP
 - določeni računalniki imajo register **skladovni kazalec** (stack pointer)

Še 2 pojma:

➤ **Pozicijsko neodvisno naslavljanje**

- pozicijsko neodvisni programi
 - lahko jih premestimo v drug del pomnilnika
 - ne smejo vsebovati absolutnih naslovov
 - neposredno, pomnilniško posredno nasl.
- možna rešitev je preslikovanje naslovov
 - če program ni pozicijsko neodvisen

➤ **PC-relativno naslavljanje**

- kot bazni register služi kar programski števec (PC)

UKAZI

23

D4: Operacije

➤ Operacije niso ključnega pomena

- Npr., možno je narediti računalnik, ki ima en sam ukaz:

SBN A,B,C

Pomen: $M[A] \leftarrow M[A] - M[B]$; če $M[A] < 0$, skoči na C

UKAZI

24

-
- Operacij je manj kot ukazov
 - Imena ukazov so **mnemoniki**
 - okrajšava ang. imena ukaza
 - vsebuje tudi operacijo
 - npr. A, D, AD, ADD, S ... za seštevanje v fiksni vejici

Skupine operacij

1. Aritmetične in logične operacije

- izvajajo se v ALE (nad operandi v fiksni vejici)
- Aritmetične operacije: seštevanje, odštevanje, množenje, deljenje, aritm. negacija, absolutna vrednost, inkrement, dekrement
 - za vsako je več ukazov (različne dolžine operandov)
- Logične operacije: AND, OR, NOT, XOR, pomiki

2. Prenosi podatkov (data transfer)

- izvor, ponor
- v resnici gre za kopiranje
- Običajni **mnemoniki**:
 - LOAD: GP \rightarrow R
 - STORE: R \rightarrow GP
 - MOVE: R \rightarrow R ali GP \rightarrow GP
 - PUSH: GP ali R \rightarrow Sklad
 - POP (PULL): Sklad \rightarrow GP ali R
- tudi CLEAR in SET

UKAZI

27

3. Kontrolne operacije

- spreminjajo vrstni red ukazov
- ### 3.1 Pogojni skoki (conditional branches).
- 3 načini za izpolnjenost pogoja:
 - **Pogojni biti** se postavijo kot rezultat določenih operacij.
 - Z (zero), N (negative), C (carry), V (overflow), itd.
 - Npr. ukaz BEQ (branch if equal) skoči, če je Z=1
 - **Pogojni register**
 - poljuben register
 - Npr. ali je njegova vsebina 0
 - **Primerjaj in skoči** (compare and branch)
 - skok, če je primerjava izpolnjena
- ### 3.2 Brezpogojni skoki (unconditional branch, jump)
- ### 3.3 Klici in vrnitve iz podprogramov
- ukaz za klic podprograma mora shraniti **povratni naslov** (return address)
 - tipična mnemonika sta CALL in JSR (jump to subroutine)
 - RET (return) za vrnitev

UKAZI

28

4. Operacije v plavajoči vejici.

- izvaja jih posebna enota (FPU – Floating Point Unit), ki ni del ALE
- poleg osnovnih štirih operacij so še koren, logaritem, eksponentna in trigonometrične funkcije

5. Sistemske operacije.

- vplivajo na način delovanja računalnika
- običajno spadajo med **privilegirane ukaze**

6. Vhodno/izhodne operacije.

- obstajajo na nekaterih računalnikih
 - na drugih se uporabljajo običajni ukazi za prenos podatkov
- prenosi med GP in V/I ter med CPE in V/I

➤ Ukaze lahko delimo tudi na

- **skalarne** in
- **vektorske**
 - na vektorskih računalnikih se lahko ista operacija izvrši na N skupinah operandov
 - pri skalarnih je treba za to uporabiti zanko
 - vektorske ukaze srečamo na superračunalnikih

D5: Vrsta in dolžina operandov

➤ Vrste operandov:

1. bit

- v višjih jezikih jih običajno ni
- koristno pri sistemskih operacijah

2. znak

- običajno 8-bitni ASCII
- več znakov tvori **niz** (string)

3. celo število

- predznačeno ali nepredznačeno
- dolžine 8, 16, 32, 64 bitov

4. realno število

- št. v plavajoči vejici (običajno po standardu IEEE 754)
- enojna natančnost 32 bitov, dvojna natančnost 64 bitov; obstajajo tudi 128-bitna

5. desetiško število

- v 8 bitih 2 BCD števili ali 1 ASCII znak

UKAZI

31

➤ Operandi dolžin večkratnikov 2 imajo posebna imena:

- 8 Bajt (byte)
- 16 Polovična beseda (halfword)
- 32 Beseda (word)
- 64 Dvojna beseda (double word)
- 128 Štirikratna beseda (quad word)

- to sicer ne velja za vse računalnike

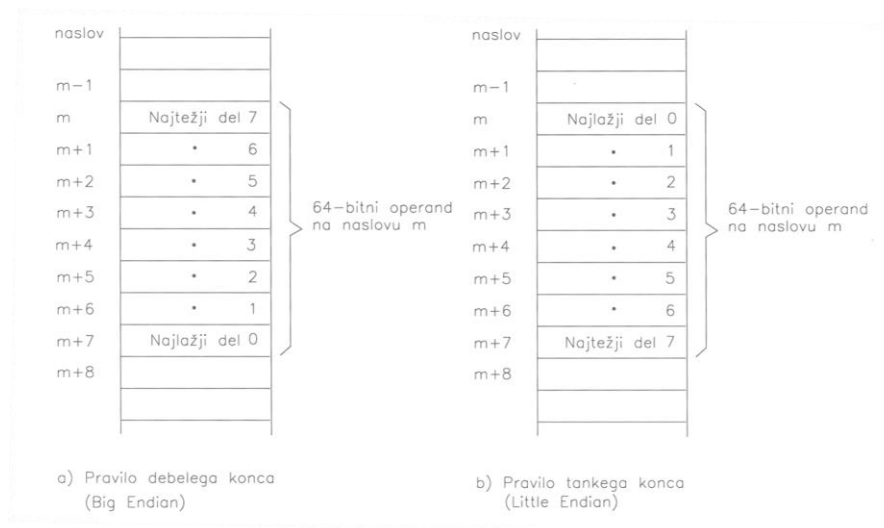
UKAZI

32

- **Sestavljeni pomnilniški operandi** so sestavljeni iz več pomnilniških besed
 - v pomnilniku morajo biti na zaporednih lokacijah, sicer bi težko podali naslov takega operanda
- Obstajata 2 načina (glede na vrstni red), kako jih shranimo v pomnilnik:
 - **pravilo debelega konca** (Big Endian Rule)
 - najtežji del operanda na najnižjem naslovu
 - **pravilo tankega konca** (Little Endian Rule)
 - najlažji del operanda na najnižjem naslovu

UKAZI

33



UKAZI

34

➤ Problem poravnosti

- pomnilnik, ki omogoča dostop do 8 8-bitnih besed hkrati, je narejen kot 8 paralelno delujočih pomnilnikov
- istočasen dostop do s besed dolgega operanda na naslovu A je možen le, če je A deljiv z s ($A \bmod s = 0$)
 - pri 8-bitni pomnilniški besedi mora imeti 64-bitni operand zadnje 3 bite enake 0
 - **poravnan** (aligned) operand
 - sicer **neporavnan** (misaligned)
 - potreben več kot en dostop
 - pri nekaterih računalnikih se sproži past

Ukazna arhitektura (ISA)

➤ Ukazna arhitektura (Instruction Set Architecture, ISA)

- natančno definira vse ukaze (nabor ukazov) nekega procesorja
- ne govori pa o implementaciji

➤ HIP je malo poenostavljena verzija procesorjev MIPS

- MIPS spada med najbolj priljubljene RISC arhitekture

HIP

- HIP je model za študij CPE
 - temelji na procesorju MIPS R2000 oz. R3000
- Lastnosti:
 - 8-bitna pomnilniška beseda
 - 32-bitni pomnilniški naslov
 - dostop do PP traja pri zadetku en cikel ure, pri zgrešitvi 11
 - pomnilniško preslikan vhod/izhod

UKAZI

37

Ostale lastnosti HIP

- Način shranjevanja operandov v CPE
 - 32 32-bitnih splošnonamenskih registrov R0, R1, ..., R31
 - Vsebina R0 je vedno 0 (pri pisanju vanj se ne zgodi nič)
- Število eksplicitnih operandov v ukazu
 - vsi ALE ukazi imajo 3 eksplicitne operande
 - pri dveh se tretji ignorira (NOT, LHI)
- Lokacija operandov in načini naslavljanja
 - Lokacija operandov
 - registrsko-registrski (load/store) računalnik
 - pomnilniški operandi nastopajo samo v ukazih load in store
 - pri ALE ukazih 2 operanda v registrih
 - tretji v registru ali takojšnji
 - dostop do operandov v pomnilniku le z load in store

UKAZI

38

■ Naslavljanje: samo 2 načina

- **Takojšnje** naslavljanje
 - takojšnji operand je 16-biten
 - razširitev predznaka (ali ničle) na 32 bitov
- **Bazno** naslavljanje
 - odmik D_i je 16-bitno predznačeno število v 2^K
 - dejanski naslov $A = R_b + D_i$
 - kot bazni register R_b katerikoli splošnonamenski
 - če R_0 : neposredno naslavljanje s 16-bitnim naslovom, ki se razširi na 32 bitov z razširitvijo predznaka
 - če je odmik 0: bazno naslavljanje brez odmika

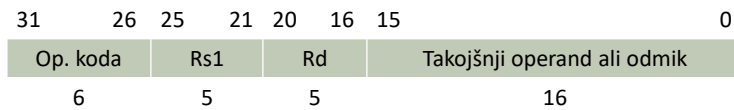
➤ Operacije in operandi

- pomnilniški operandi so lahko 8-, 16- ali 32-bitni (bajt, polbeseda, beseda)
- 16- in 32-bitni operandi so v pomnilniku shranjeni po **pravilu debelega konca (big endian rule)** in morajo biti obvezno poravnani (sicer past)
- tudi biti (GP in R) po pravilu debelega konca
- vse ALE operacije so 32-bitne
 - 8- in 16-bitni operandi se pri load pretvorijo v 32-bitne
 - Razširitev ničle pri nepredznačenih (LBU, LHU)
 - Razširitev predznaka pri predznačenih (LB, LH)
- vse ALE operacije se izvršijo v eni urini periodi

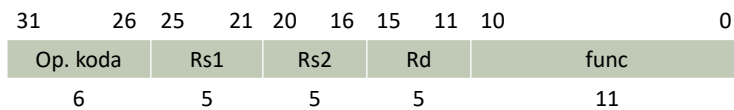
Zgradba ukazov pri HIP:

- vsi ukazi so 32-bitni
- 2 formata s 6-bitno operacijsko kodo
 - Format 2: bita 30 in 31 enaka 1
 - Format 1: sicer

Format 1:



Format 2:



UKAZI

41

- če bit 30 enak 1, imamo bazno naslavljanje, sicer takojšnje
- v formatu 2 parameter func razširja operacijsko kodo
 - v formatu 2 je možnih $2^4 \cdot 2^{11}$ ukazov ($=2^{15}$), HIP jih uporablja 21
- kot Rs1, Rs2 ali Rd se lahko uporabi vsak register (R0 do R31)

➤ Število ukazov

- vseh ukazov je 52
 - 31 v formatu 1, 21 v formatu 2
- ni ukazov za množenje, deljenje
- ni ukazov v plavajoči vejici

UKAZI

42

Vrste ukazov

➤ Ukaze HIP delimo v več skupin:

1. ukazi za prenos podatkov (load, store)
 - gre za prenos *operandov* med registri in pomnilnikom
 - nalaganje (iz pomnilnika) in shranjevanje (v pomnilnik)
2. ALE ukazi
 - aritmetične in logične operacije
3. kontrolni ukazi
 - skoki
4. sistemske ukazi

UKAZI

43

Ukazi za prenos podatkov (load/store)

- Uporabljajo format 1 z baznim naslavljanjem (bazni register je Rs1)

Load word:

`lw r1, 30(r2)` ; $r1 \leftarrow_{32} M[30 + r2]$

Format 1:

31	26	25	21	20	16	15	0
Op. koda	Rs		Rd		(Takojšnji operand ali) odmik		
6 bitov	5 bitov		5 bitov		16 bitov		
100110	00010		00001		0000000000011110		
lw	r2		r1		30		
38	2		1		30		

Celoten ukaz v strojni kodi: 0x9841001E

UKAZI

44

-
- Kaj pa, če je vrednost, ki jo želimo naložiti v (32-bitni) register, krajša od njegove dolžine?
 - Na katero vrednost postavimo preostale bite?
 - 2 možnosti:
 - razširitev predznaka
 - lb (load byte (signed))
 - lh (load halfword (signed))
 - razširitev ničle
 - lbu (load byte unsigned)
 - lhu (load halfword unsigned)

UKAZI

45

Load byte:

lb r1, 80(r2)

pomen: $r1_{31..8} \leftarrow_{\text{raz}} M[80 + r2]_7$, $r1_{7..0} \leftarrow_8 M[80 + r2]$

npr.: 0x6F se razširi drugače kot 0x94

Format 1:

31	26	25	21	20	16	15	0
Op. koda	Rs	Rd	(Takojšnji operand ali) odmik				
6 bitov	5 bitov	5 bitov	16 bitov				
100100	00010	00001	0000000001010000				
lb	r2	r1	80				

UKAZI

46

Load byte unsigned

`lbu r1, 80(r2)`

pomen: $r1_{31..8} \leftarrow_{\text{raz}} 0, r1_{7..0} \leftarrow_8 M[80 + r2]$

tu se 0x6F razširi enako kot 0x94

Podobno za 16-bitne besede:

➤ Load halfword

- halfword (polbeseda) je 16 bitov: 2B

➤ Load halfword unsigned

UKAZI

47

Ukazi za prenos podatkov (load/store):

Format	Op. koda	Ukaz	Opis
1	100000	LBU	Load byte unsigned
1	100001	LHU	Load halfword unsigned
1	100100	LB	Load byte
1	100101	LH	Load halfword
1	100110	LW	Load word
1	101000	SB	Store byte
1	101001	SH	Store halfword
1	101010	SW	Store word

- Odmik je 16-biten
- Pri ukazih load za 8- in 16-bitne operande sta 2 varianti:
 - običajna (signed): razširitev predznaka (do 32 bitov)
 - unsigned: razširitev ničle (do 32 bitov)

UKAZI

48

Primeri ukazov load/store

Primer ukaza		Ime ukaza	Opis
LW	R1, 30(R2)	Load word	$R1 \leftarrow_{32} M[30 + R2]$
LW	R1, 1000(R0)	Load word	$R1 \leftarrow_{32} M[1000]$
LB	R1, 80(R3)	Load byte	$R1_{0..7} \leftarrow_8 M[80 + R3], R1_{8..31} \leftarrow_{\text{raz}} M[80 + R3]_7$
LBU	R1, 80(R3)	Load byte unsigned	$R1_{0..7} \leftarrow_8 M[80 + R3], R1_{8..31} \leftarrow 0$
LH	R1, 80(R3)	Load halfword	$R1_{0..15} \leftarrow_{16} M[80 + R3], R1_{16..31} \leftarrow_{\text{raz}} M[80 + R3]_{15}$
LHU	R1, 80(R3)	Load halfword unsigned	$R1_{0..15} \leftarrow_{16} M[80 + R3], R1_{16..31} \leftarrow 0$
SW	70(R5), R6	Store word	$M[70 + R5] \leftarrow_{32} R6$
SB	70(R5), R6	Store byte	$M[70 + R5] \leftarrow_8 R6_{0..7}$
SH	70(R5), R6	Store halfword	$M[70 + R5] \leftarrow_{16} R6_{0..15}$

UKAZI

49

-
- $M[x]$ je vsebina pomnilniške besede na naslovu x
 - Znak \leftarrow_{32} pomeni 32-bitni prenos iz (ali v) naslovov $x, x+1, x+2, x+3$ po pravilu debelega konca
 - Znak \leftarrow_{16} pomeni 16-bitni prenos iz (ali v) naslovov $x, x+1$
 - Znak \leftarrow_8 pomeni 8-bitni prenos iz (ali v) naslov x
 - Znak \leftarrow_{raz} pomeni razširitev bita

Pri ukazih store je Rd izvor

UKAZI

50

Primer programa v zbirnem jeziku za procesor HIP

```

        .data
        .org 0x400
var1:   .byte 5
var2:   .byte 6
sum:    .space 1
        .align 2      ; zahtevana je poravnost
ABC:    .word16 -33, 0x1C

        .code
        .org 0
        lb r1, var1(r0)
        lb r2, var2(r0)
        add r3, r1, r2
        sb sum(r0), r3
        halt

```

UKAZI

51

Psevdo-ukazi

- .data** – začetek podatkovnega segmenta
- .text, .code** – začetek ukaznega segmenta
- .org <n>** – določen začetni naslov
- .space <n>** – rezerviraj n bajtov prostora (naključne vred.)
- .word <n1>,<n2>..** – določi zaporedna 32-bitna števila
- .word16 <n1>,<n2>..** – določi zaporedna 16-bitna števila
- .byte <n1>,<n2>..** – določi zaporedna 8-bitna števila
- .align <n>** – poravnaj naslov, da bo deljiv z n

Psevdo-ukazi so namenjeni zbirniku (programu), ne procesorju!

UKAZI

52

-
- Glej dokument (pdf) **Nabor ukazov procesorja HIP**, ki je na spletni učilnici (v poglavju Druga gradiva in programi)
 - ta vsebuje tudi psevdo-ukaze zbirnika za procesor HIP

ALE ukazi

1. **aritmetične operacije** (+, −)
 - add, addu (+ addi, addui)
 - sub, subu (+ subi, subui)
2. **logične operacije** (&, V, \oplus , not)
 - and, or, xor (+ andi, ori, xori), not
3. **pomiki** (shift) (levi, desni; logični, aritmetični)
 - sll, srl, sra (+ slli, srli, srai)
 - lhi
4. **ukazi za primerjavo oz. set operacije** (pogoji: =, ≠, <, >, ≤, ≥)
 - seq, sne, slt, sgt, sltu, sgtu (+ seqi, snei, slti, sgti, sltui, sgtui)

➤ Logične operacije:

- & (and), V (or), \oplus (xor), not
- Delujejo po bitih (*bitwise operations*)
 - 0110 and 1010 = 0010
 - 0xABCD and 0x1357
 - 0110 or 1010 = 1110
 - 0xABCD or 0x1357
 - 0110 xor 1010 = 1100
 - 0xABCD xor 0x1357
 - not 0110 = 1001
- Uporabljajo se tudi za branje in vpisovanje posameznih bitov
 - branje bita: xxxx & 0100 primerjamo z 0000
 - nastavljanje bita: xxxx | 0100
 - brisanje bita: xxxx & 1011

UKAZI

55

➤ Pomiki:

- **navadni pomiki** (shift)
- **rotacije** (rotate)

➤ Navadni pomiki:

- levi (0110 v 1100)
- desni
 - **logični** (0110 v 0011)
 - v izpraznjena mesta gredo ničle
 - **aritmetični** (0110 v 0011, 1011 v 1101)
 - najbolj levi bit se ne spreminja in se vstavlja v izpraznjena mesta (število smatramo kot predznačeno – ta bit je predznak)

UKAZI

56

- Levi pomik (za n mest) predstavlja tudi množenje z 2^n
 - $00000101 \ll 3 = 00101000$
- Desni pomik (za n mest) pa je deljenje z 2^n
 - $00110010 \gg 4 = 00000011$
- Aritmetični pomik ohrani predznak
 - število obravnava kot predznačeno
 - $11000 \gg 1 = 11100$
 - ni pa to več pravo celoštevilsko deljenje!
 - $11001 \gg 1 = 11100$ ($-7 \gg 1 = -4$)
- S pomiki in seštevanjem/odštevanjem je možno realizirati tudi poljubno množenje/deljenje
- Tudi pomiki (logični) se uporabljajo za izločanje/vstavljanje bitov
 - npr. $0x1 \ll 2 = 0100$

UKAZI

57

Seznam vseh ALE ukazov

- ALE ukazi so 3-operandni
 - razen NOT in LHI, ki sta v resnici 2-operandna, zato se eden od treh operandov ignorira
- 2 operanda sta v registrih
 - tretji je lahko v registru ali takojšnji (immediate)

$Rd \leftarrow Rs1 \text{ op } Rs2$

$Rd \leftarrow Rs1 \text{ op } \text{Takojšnji operand}$

UKAZI

58

ALE ukazi (1): aritmetične in logične operacije

Format	Op. koda	func	Ukaz	Opis	Tip operacije
2	110000	0	ADD	Add	Aritmetične
2	110001	0	SUB	Subtract	
2	110010	0	ADDU	Add unsigned	
2	110011	0	SUBU	Subtract unsigned	
2	110100	0	AND	And	Logične
2	110101	0	OR	Or	
2	110110	0	XOR	Exclusive or	
2	111110	0	NOT	Not (1's complement)	
1	000000	x	ADDI	Add immediate	Aritmetične takojšnje
1	000001	x	SUBI	Subtract immediate	
1	000010	x	ADDUI	Add unsigned imm.	
1	000011	x	SUBUI	Sub. unsigned imm.	
1	000100	x	ANDI	And immediate	Logične takojšnje
1	000101	x	ORI	Or immediate	
1	000110	x	XORI	Exclusive or imm.	

UKAZI

59

ALE ukazi (2): Ukazi za primerjavo

Format	Op. koda	func	Ukaz	Opis	Tip operacije
2	111000	0	SEQ	Set if equal	set
2	111001	0	SNE	Set if not equal	set
2	111010	0	SLT	Set if less than	set
2	111011	0	SGT	Set if greater than	set
2	111100	0	SLTU	Set if less than unsigned	set
2	111101	0	SGTU	Set if greater than unsigned	set
1	001000	x	SEQI	Set if equal immediate	set imm.
1	001001	x	SNEI	Set if not equal immediate	set imm.
1	001010	x	SLTI	Set if less than immediate	set imm.
1	001011	x	SGTI	Set if greater than imm.	set imm.
1	001100	x	SLTUI	Set if less than unsig. imm.	set imm.
1	001101	x	SGTUI	Set if greater than uns. imm.	set imm.

Če je pogoj izpolnjen, se v *Rd* zapiše 1, sicer 0

UKAZI

60

ALE ukazi (3): Pomiki

Format	Op. koda	func	Ukaz	Opis	Tip operacije
2	110000	1	SLL	Shift left logical	shift
2	110001	1	SRL	Shift right logical	shift
2	110010	1	SRA	Shift right arithmetic	shift
1	010000	x	SLLI	Shift left logical immediate	shift imm.
1	010001	x	SRLI	Shift right logical immediate	shift imm.
1	010010	x	SRAI	Shift right arithmetic imm.	shift imm.
1	000111	x	LHI	Load high immediate	

Ukazi za pomike uporabljajo pomikalnik (barrel shifter)

- kombinacijsko vezje, ki izvede poljuben pomik (za 0, ..., 31 mest) v eni urini periodi
- število mest pomika je podano v *Rs2* ali v takojšnjem operandu

UKAZI

61

Load high immediate (Lhi)

- poseben ukaz, ki 16-bitno (konstantno) vrednost naloži v gornjo polovico registra
- Zakaj sploh potrebujemo tak ukaz?
 - Problem je, kako naložiti 32-bitno konstanto v register
 - z enim 32-bitnim ukazom ni možno
 - zato to storimo v 2 korakih:
 1. naložimo zgornjih 16 bitov
 2. naložimo spodnjih 16 bitov
 - Npr.: 0x12345678


```
lhi    r1, 0x1234
addui  r1, r1, 0x5678    (lahko tudi ori)
```

UKAZI

62

➤ Kaj pa, če konstante ne poznamo?

- npr., da gre za oznako (labelo)
 - ta predstavlja naslov
 - npr. oznaka ABC, ki ima vrednost 0x12345678
 - to vrednost izračuna zbirnik (assembler)
- v tem primeru lahko zbirnik zasnujemo tako, da
 - v ukazu LHI podamo 32-bitno vrednost, zbirnik pa upošteva samo zgornjo polovico
`lhi r1, ABC` se tolmači kot `lhi r1, 0x1234`
 - v ukazu ADDUI podamo 32-bitno vrednost, zbirnik pa upošteva samo spodnjo polovico
`addui r1, r1, ABC` se tolmači kot `addui r1, r1, 0x5678`

UKAZI

63

➤ To velja tudi, če so podatki na naslovu, ki se ne da zapisati s 16 biti ("visok" naslov)

- v tem primeru je treba visok naslov naložiti v register
- to je ekvivalentno nalaganju (poljubne) 32-bitne konstante v register

UKAZI

64

- Primer programa v zbirnem jeziku za procesor HIP, ki vrednost 32-bitne spremenljivke A prepíše v 32-bitno spremenljivko B. Ukazi naj se nahajajo od naslova 0x0 naprej, spremenljivka A je na naslovu 0x400, B pa na naslovu 0x25000000.

```
.data
.org 0x400
A: .word 2014

.data
.org 0x25000000
xy: .space 4
B: .space 4

.code
.org 0
lw r1, A(r0)
lhi r2, B           ; assembler: lhi r2, 0x2500
addui r2, r2, B     ; assembler: addui r1, r2, 0x0004
sw 0(r2), r1
halt
```

UKAZI

65

Add:

add r3, r5, r6 ; $r3 \leftarrow r5 + r6$

Format 2:

31	26	25	21	20	16	15	11	10	0
Op. koda		Rs1		Rs2		Rd		func	
6		5		5		5		11	
110000		00101		00110		00011		00000000000	
add (func ₀ =0)		r5		r6		r3		add (func ₀ =0)	
ali									
sll (func ₀ =1)									

UKAZI

66

Add immediate:

addi r3, r5, 20 ; $r3 \leftarrow r5 + 20$

Format 1:

31	26	25	21	20	16	15	0
Op. koda	Rs	Rd	Takojšnji operand (ali odmik)				
6 bitov	5 bitov	5 bitov	16 bitov				
000000	00101	00011	0000000000010100				
addi	r5	r3	20				

UKAZI

67

Zakaj imata add in addi različen format?

- Ukaza **addu** in **addui** sta enaka kot **add** in **addi**, le da se operanda tolmačita kot nepredznačena (unsigned)
 - zato ne more priti do preliva, sam izračun pa je enak
- Ukazi **sub**, **subi**, **subu** in **subui** so namenjeni odštevanju (i ... immediate, u ... unsigned)

UKAZI

68

And:

and r1, r2, r3 ; $r1 \leftarrow r2 \& r3$

Format 2:

31	26	25	21	20	16	15	11	10	0
Op. koda		Rs1		Rs2		Rd		func	
6		5		5		5		11	
110100		00010		00011		00001		00000000000	
and (oz. di)		r2		r3		r1		func ₀ =0 (and)	

UKAZI

69

And immediate:

andi r18, r2, 0x890F ; $r18 \leftarrow r2 \& 0x890F$

Format 1:

31	26	25	21	20	16	15	0
Op. koda		Rs1		Rd		Takojšnji operand	
6		5		5		16	
000100		00010		10010		1000100100001111	
andi		r2		r18		0x890F	

UKAZI

70

Shift left logical:

sll r1, r2, r3

 $; r1 \leftarrow r2 \ll r3 \text{ (oz. } r2 \times 2^{r3})$ **Format 2:**

31	26	25	21	20	16	15	11	10	0
Op. koda		Rs1		Rs2		Rd		func	
6		5		5		5		11	
110000		00010		00011		00001		00000000001	
sll (oz. add)		r2		r3		r1		func ₀ = 1 (sll)	

UKAZI

71

Shift right arithmetic:

sra r6, r7, r8

 $; r6 \leftarrow r7 \gg r8$ $; r6_{31} \leftarrow r7_{31}$ **Format 2:**

31	26	25	21	20	16	15	11	10	0
Op. koda		Rs1		Rs2		Rd		func	
6		5		5		5		11	
110010		00111		01000		00110		00000000001	
sra (oz. addu)		r7		r8		r6		func ₀ = 1 (sra)	

UKAZI

72

Set if greater than:

sgt r2, r3, r4 ; r2 \leftarrow (r3 > r4)

Format 2:

31	26	25	21	20	16	15	11	10	0
Op. koda		Rs1		Rs2		Rd		func	
6		5		5		5		11	
111011		00011		00100		00010		0000000000	
sgt		r3		r4		r2		func ₀ = 0	

UKAZI

73

Load high immediate:

lhi r5, 38 ; r5_{31..16} \leftarrow 38, r5_{15..0} \leftarrow 0

Format 1:

31	26	25	21	20	16	15	0
Op. koda		Rs1		Rd		Takojšnji operand	
6		5		5		16	
000111		00000		00101		0000000000100110	
lhi		r0		r5		38	

UKAZI

74

- Primer: program, ki na osnovi pomikov in seštevanja 32-bitno nepredznačeno spremenljivko A množi z 10 in jo shrani v B:

```

        .data
        .org 0x400
A:      .space 4      ( ali .word vrednost)
B:      .space 4

        .code
        .org 0x0
        lw r1, A(r0)
        slli r2, r1, 3
        slli r3, r1, 1
        add r4, r2, r3
        sw B(r0), r4
        halt

```

UKAZI

75

-
- Register R0, lahko uporabimo za tvorbo vrste novih operacij iz obstoječih:
- MOV R1,R2 (Move one register to another) z ukazom ADD R1,R0,R2.
 - Vsebina R2 se naloži v R1
 - NOP (No operation) z ukazom ADDI R0,R0,#0
 - LI R1,#const (Load halfword immediate) z ukazom ADDI R1,R0,#const
 - 16-bitna predznačena konstanta const se naloži v R1 z razširitvijo predznaka
 - LUI R1,#const (Load halfword unsigned immediate) z ukazom ADDUI R1,R0,#const
 - 16-bitna nepredznačena konstanta const se naloži v R1 z razširitvijo ničle

UKAZI

76

Kontrolni ukazi

- Kontrolni ukazi omogočajo spremembo vrstnega reda izvajanja ukazov
 - takim ukazom rečemo skoki
- 2 vrsti skokov:
 - brezpogojni
 - vedno se izvede
 - omogoča preskok dela programa, pa tudi vrnitev nazaj
 - pogojni
 - izvede se, če je izpolnjen določen pogoj
 - omogoča pogojni preskok dela programa in končne zanke
- Kontrolni ukazi omogočajo vejitve in zanke

UKAZI

77

-
- Skoki pri HIP:
 - Brezpogojni skok:
 - j (jump)
 - Pogojna skoka:
 - beq (branch if equal zero) pri $Rd = 0$
 - bne (branch if not equal zero) pri $Rd \neq 0$
 - Kontrolni ukazi pri HIP uporabljajo format 1
 - Pogojna skoka uporabljata PC-relativno naslavljanje
 - za bazni register je uporabljen $PC + 4$
 - register Rs1 se ignorira

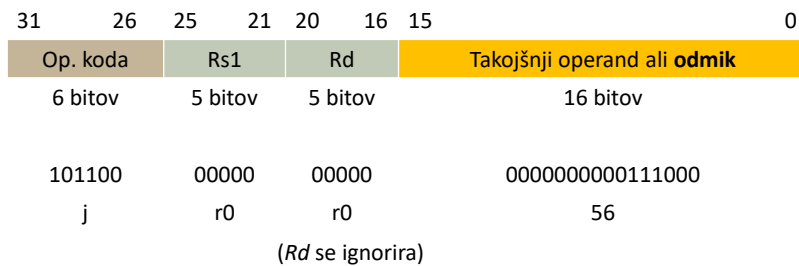
UKAZI

78

Jump:

j LABEL(Rs1) ; $PC \leftarrow LABEL + Rs1$

Npr.: j Nekam(r0) ; $PC \leftarrow Nekam + r0$

Format 1:

UKAZI

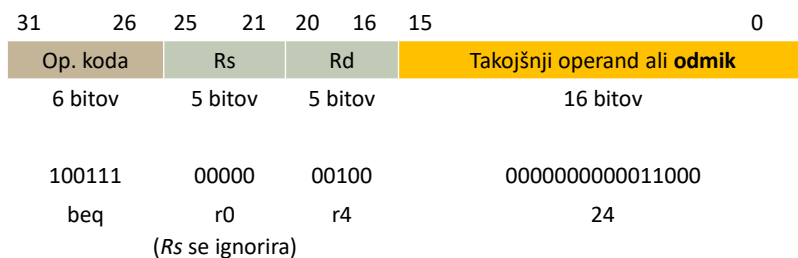
79

Branch if equal (to) zero:

beq Rd, LABEL (PC-relativno)
(pogoj se nanaša na vsebino registra *Rd*)

Npr.:

beq r4, 24 ; če $r4 == 0$, potem $PC \leftarrow PC + 4 + 24$, sicer $PC \leftarrow PC + 4$

Format 1:

UKAZI

80

Vejitve

```
if ( pogoj)
    blok1
else
    blok2
```

- Če pogoj ni izpolnjen, je treba skočiti na blok2

- Ukaz beq izvaja pogojni skok

```
beq  Rd, LABEL
```

- Če je register Rd enak 0, CPE skoči na naslov LABEL

- Podoben ukaz je

```
bne  Rd, LABEL
```

- Če je register Rd različen od 0, CPE skoči na naslov LABEL

UKAZI

81

➤ Primer:

```
if (x > 5)
    a = b + 1;
else
    a = b + 2;
```

Predpostavimo r1: x, r2: a, r3: b

UKAZI

82

➤ 2 možnosti:

```

        sgti   r4, r1, 5      ; if (x > 5) r4 = 1;
        bne   r4, Blk1        ; if (r4!=0) goto Blk1;
        addi  r2, r3, 2      ; a = b + 2;
        j     Ven(r0)        ; goto Ven;
Blk1:    addi  r2, r3, 1      ; Blk1: a = b + 1;
Ven:     naslednji ukaz      ; Ven: naslednji ukaz

```

```

        sgti   r4, r1, 5      ; if (x > 5) r4 = 1;
        beq   r4, Blk2        ; if (r4==0) goto Blk2;
        addi  r2, r3, 1      ; a = b + 1;
        j     Ven(r0)        ; goto Ven;
Blk2:    addi  r2, r3, 2      ; Blk2: a = b + 2;
Ven:     naslednji ukaz      ; Ven: naslednji ukaz

```

UKAZI

83

Zanke

```

while ( pogoj)
    Blok;

```

Pogosto je zanka WHILE take oblike:

```

i = I1;
while ( i < I2)
{
    ...
    i = i + K;
}

```

V takem primeru lahko uporabimo tudi zanko FOR:

```

for (i = I1; i < I2; i=i+K)
{
    ...
}

```

UKAZI

84

Primer 1

Jezik C	Zbirni jezik za HIP (r1: sum, r2: i)
<pre>sum = 0; i = 5; while (i > 2) { sum = sum + i; i--; }</pre>	<pre>addi r1, r0, #0 addi r2, r0, #5 Loop: sgti r3, r2, #2 beq r3, Ven add r1, r1, r2 subi r2, r2, #1 j Loop(r0) Ven: ...</pre>
<pre>sum = 0; for (i=5; i>2; i--) sum = sum + i;</pre>	isto kot zgoraj
<pre>sum = 0; i = 5; do { sum = sum + i; i--; } while (i > 2);</pre>	<pre>addi r1, r0, #0 addi r2, r0, #5 Loop: add r1, r1, r2 subi r2, r2, #1 sgti r3, r2, #2 bne r3, Loop</pre>

- Zanka do-while je v zbirnem jeziku enostavnejša od while
- Seveda pa while in do-while v splošnem nista ekvivalentna!
 - pri slednjem se blok prvič vedno izvede

UKAZI

85

Primer 2

	<pre>while (a[i] == j) i++;</pre>
	(r1: i, r2: j, r3: bazni naslov a, tj. naslov od a[0])
Loop:	<pre>sll r4, r1, 2 add r4, r4, r3 lw r5, 0(r4) ; v r4 je naslov a[i] sub r5, r5, r2 bne r5, Exit addi r1, r1, #1 j Loop(r0)</pre>
Exit:	...

UKAZI

86

Nabor vseh kontrolnih ukazov

Format	Op. koda	Ukaz	Opis
1	100011	BNE	Branch if Rd not equal to zero
1	100111	BEQ	Branch if Rd equal to zero
1	101100	J	Jump
1	101101	CALL	Jump to subroutine
1	101110	TRAP	Jump to vector address
1	101111	RFE	Return from exception

- Ukaz za klic procedure CALL shrani naslov naslednjega ukaza (PC + 4) v Rd, v PC pa se zapiše Rs+odmik
 - vrnitev iz proc. je brezpogojni skok z odmikom 0 (bazni reg. Rd)

UKAZI

87

➤ Ukaz TRAP

- $EPC \leftarrow PC$
- onemogoči se prekinitve ($I \leftarrow 0$)
- skok na servisni program
 - njegov naslov je na naslovu $FFFFFF00 + 4 \times n$ (to je vektorski naslov ali vektor, n pa je številka vektorja)
 - shrani se v PC

➤ Pri RFE se EPC zapiše v PC

UKAZI

88

Primeri kontrolnih ukazov

Primer ukaza	Ime ukaza	Opis
J 84 (R8)	Jump	$PC \leftarrow R8 + 84$
BEQ R7, 800	Branch if equal to zero	če je $R7 = 0$, potem $PC \leftarrow PC + 4 + 800$ sicer $PC \leftarrow PC + 4$
BNE R7, 800	Branch if not equal to zero	če je $R7 \neq 0$, potem $PC \leftarrow PC + 4 + 800$ sicer $PC \leftarrow PC + 4$
CALL R9, 84(R8)	Jump to subroutine	$R9 \leftarrow PC + 4$, $PC \leftarrow R8 + 84$
TRAP #n	Jump to vector address	$EPC \leftarrow PC + 4$, $PC \leftarrow_{32} M[FFFFFF00 + 4 \times n]$, $I \leftarrow 0$, $0 \leq n \leq 63$
RFE	Return from exception	$PC \leftarrow EPC$

UKAZI

89

Sistemiški ukazi

Format	Op. koda	func	Ukaz	Opis
2	110011	1	EI	Enable interrupts
2	110100	1	DI	Disable interrupts
2	110101	1	MOVER	Move from EPC to register
2	110110	1	MOVRE	Move from register to EPC

- Pri sistemskih ukazih se registri ignorirajo (ali pa se uporablja le eden)

UKAZI

90

Zgradba ukazov

Ukaz je shranjen v eni ali več (sosednih) pomnilniških besedah

Vsak ukaz vsebuje

1. Operacijsko kodo (informacijo o operaciji, ki naj se izvrši)
2. Informacijo o operandih, nad katerimi naj se izvrši operacija



UKAZI

91

➤ Zgradba ali format ukaza

- pove, kako so biti ukaza razdeljeni na operacijsko kodo in operande
 - število polj, njihova velikost in pomen posameznih bitov v njih

➤ Možni so različni formati

➤ Parametri, ki najbolj vplivajo na format:

1. Dolžina pom. besede
 - pri 8: dolžina ukaza večkratnik 8
 - pri dolgih pom. besedah: dolžina ukaza $\frac{1}{2}$ ali $\frac{1}{4}$ besede
2. Število eksplicitnih operandov v ukazu
3. Vrsta in število registrov v CPE
 - št. registrov vpliva na št. bitov za naslavljanje
4. Dolžina pom. naslova
 - predvsem, če se uporablja neposredno naslavljanje
5. Število operacij

UKAZI

92

➤ Optimalne rešitve za format ukazov ni

- kaj je kriterij?
- neke vrste umetnost
- medsebojna odvisnost parametrov
- možno je minimizirati velikost programov
 - pogostost ukazov, Huffmanovo kodiranje
 - v praksi se ni izkazalo (Burroughs)

UKAZI

93

➤ 3 načini:

1. Spremenljiva dolžina

- št. eksplicitnih operandov spremenljivo
- različni načini naslavljanja
- veliko formatov
 - npr. 1..15 bajtov pri 80x86, 1..51 VAX
- kratki formati za pogoste ukaze

Op. koda	Način naslavljanja 1	Naslovno polje 1	.	.	.	Način naslavljanja n	Naslovno polje n
----------	----------------------	------------------	---	---	---	----------------------	------------------

UKAZI

94

2. Fiksna dolžina

- št. eksplicitnih operandov fiksno
- majhno št. formatov (RISC)
 - Alpha, ARM, MIPS, PowerPC, SPARC

Op. koda	Naslovno polje 1	Naslovno polje 2	Naslovno polje 3
----------	------------------	------------------	------------------

3. Hibridni način

Op. koda	Način naslavljanja	Naslovno polje
----------	--------------------	----------------

Op. koda	Naslovno polje 1	Način naslavljanja 2	Naslovno polje 2
----------	------------------	----------------------	------------------

Op. koda	Način naslavljanja	Naslovno polje 1	Naslovno polje 2
----------	--------------------	------------------	------------------

UKAZI

95

➤ Ortogonalnost ukazov (medsebojna neodvisnost parametrov ukaza)

1. Informacija o operaciji neodvisna od info. o operandih
2. Informacija o enem operandu neodvisna od info. o ostalih operandih

UKAZI

96

Število ukazov in RISC

➤ CISC računalniki

- Complex Instruction Set Computer
- imajo veliko število ukazov
- IBM 370, VAX, Intel

➤ RISC računalniki

- Reduced Instruction Set Computer
- imajo majhno število ukazov
- MIPS, ARM, DEC Alpha, IBM/Motorola Power PC

➤ Oboji imajo svoje prednosti in slabosti

- na začetku so bili računalniki tipa CISC, RISC pa so se pojavili kasneje
- RISC so enostavnejši in imajo hitrejše ukaze, vendar pa program potrebuje več ukazov

UKAZI

97

➤ 2 ugotovitvi v 80. letih:

1. **Stalno povečevanje števila ukazov**
 - IAS (1951): 23 ukazov in 1 način nasl.
 - 70. leta: stotine ukazov
2. **Velik del ukazov redko uporabljan**

UKAZI

98

Razlogi za povečevanje števila ukazov

- Semantični prepad
 - v 60. letih so proizvajalci zato povečevali št. ukazov
- Mikroprogramiranje
 - dodajanje novih ukazov preprosto
- Razmerje med hitrostjo CPE in GP
 - faktor vsaj 10
 - kompleksen ukaz hitrejši kot zaporedje preprostih ukazov

Razlogi za zmanjševanje števila ukazov

- Težave prevajalnikov
 - velik del ukazov redko uporabljan
- Pojav predpomnilnikov
 - v primeru zadetka v PP je dostop skoraj enako hiter kot do mikroukazov
- Uvajanje paralelizma v CPE
 - cevovod (lažja realizacija pri preprostih ukazih)

Definicija arhitekture RISC

- Večina ukazov se izvrši v enem ciklu CPE
 - lažja real. cevovoda
- Registrsko-registrska zasnova (load/store)
 - zaradi zahteve 1
- Ukazi realizirani s trdo ožičeno logiko
 - ne mikroprogramsko
- Malo ukazov in načinov naslavljanja
 - hitrejša in enostavnejša dekodiranje in izvrševanje
- Enaka dolžina ukazov
- Dobri prevajalniki
 - upoštevajo zgradbo CPE