

## 6

# Ukazi (splošno)

1

## Ukazi

- CPE lahko izvršuje ukaze na 2 načina:

### 1. Trdo ozičena logika

- vezje (logična vrata, pomnilne celice, povezave)
- spremembe so možne le s fizičnim posegom

### 2. Mikrogramiranje

- v CPE je vgrajen poseben računalnik za izvajanje ukazov
- pri vsakem ukazu se aktivira ustrezen zaporedje **mikroukazov (mikrogram)**
- mikrogrami so shranjeni v kontrolnem pomnilniku CPE
- mikroukazi so primitivnejši od običajnih in jih izvršuje trdo ozičena logika
- počasnejše, vendar lahko spremenjamo ali dodajamo ukaze, ne da bi spreminali vezje

- Uporabnika način izvajanja ukazov ne zanima

2

- Vsak ukaz vsebuje

1. Informacijo o operaciji, ki naj se izvrši
    - to polje se imenuje **operacijska koda**
  2. 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
    - formati so odvisni od števila registrov, dolžine pomnilniške besede, dolžine pomnilniškega naslova, število operacij, ...



... 1 ... 414 ... 3

## Lastnosti ukazov - 5 dimenzij

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

4

### D1. Načini za shranjevanje operandov v CPE

- CPE vsebuje **programske dostopne registre**
  - to je majhen pomnilnik, v katerega lahko shranimo enega ali več operandov
  - prednosti:
    1. Večja hitrost
      - registri so hitrejši od GP
      - bližji so aritmetično-logični in kontrolni enoti (krajši čas potovanja signalov)
      - možen je istočasen dostop do več registrov naenkrat
    2. Krajši ukazi
      - krajši naslov (ker je registrov malo) kot pri GP

5

- 3 načini shranjevanja operandov v CPE:

#### 1. Akumulator.

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

6

## 2. Sklad (stack).

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

7

- Lukasiewicz: **postfiksna oz. reverzna poljska notacija**
  - npr.  $(B \times (C + D)) / (E + F)$  zapišemo kot  $B C D + \times E F + /$   
 $B C D + \times E F + / \quad G = C + D$   
 $H E F + / \quad H = B \times G$   
 $H I / \quad I = E + F$   
 $J \quad J = H / I$
- sklad je idealno sredstvo za računanje v tej notaciji
- **skladovni računalniki** (firma Burroughs, 60. leta)
- prednost pri dolgih aritmetičnih ukazih
  - danes jih ni tako veliko

8

## 3. Množica registrov (register set).

- najbolje
  - danes edina rešitev
  - toda nekdaj so bili dragi, pa tudi prevajalniki jih niso znali dobro uporabljati
- vsak register ima svoj naslov
- 2 rešitvi:
  - splošnionamenski registri (vsi ekvivalentni)
  - 2 skupini:
    - » za AL operande
    - » za naslove
- lahko shranjujemo vmesne rezultate
  - pri skladu: v pomnilnik

9

## D2: Število eksplisitnih operandov v ukazu

- **m-operandni** (m-naslovni) računalnik
  - običajno se podajajo naslovi operandov
  - danes m največ 3
- 5 skupin:
  - **3+1-operandni** računalniki

OP3  $\leftarrow$  OP2 + OP1  
(PC  $\leftarrow$ ) OP4                   naslov nasl. ukaza

- GP (krožni dostop) drugačen kot danes
  - PC-ja ni bilo
  - s primerno razporeditvijo ukazov in operandov je deloval mnogo hitreje, kot bi, če bi bili ukazi zapisani po naraščajočih naslovih
- EDVAC

10

### – **3-operandni**

- pomnilniki z naključnim dostopom odpravijo potrebo po 'pametnem' razmeščanju ukazov
  - s tem tudi po ekspl. podajanju naslova nasl. ukaza
- pojavi se izvrševanje ukazov po narašč. naslovih
  - PC  $\leftarrow$  PC + 1

OP3  $\leftarrow$  OP2 + OP1  
PC  $\leftarrow$  PC + 1

- operandi so običajno v registrih

11

### – **2-operandni**

- enostavnejši, a malo počasnejši

OP2  $\leftarrow$  OP2 + OP1  
PC  $\leftarrow$  PC + 1

12

- **1-operandni**

- imajo akumulator

$$AC \leftarrow AC + OP1$$

$$PC \leftarrow PC + 1$$

- mikroprocesorji iz 70. in 80. let

- Intel 8080, Motorola 6800, Zilog Z80
- Intel 8086, Intel 80186, Intel 80286

- običajno imajo še kak dodaten register

- model brez se uporablja za teoretične študije izračunljivosti

13

- **Brez-operandni (skladovni)**

- najkrajši ukazi

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

$$PC \leftarrow PC + 1$$

- toda: potrebna sta vsaj 2 ukaza z ekspl. operandom!

- PUSH, POP (prenos med GP in skladom)

14

## D3: Lokacija operandov in načini naslavljjanja

- 2 vprašanji:

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

- **Lokacija operandov**

- registri CPE
- GP (oz. predpomnilnik)
- (registri krmilnika V/I naprave)

15

- 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**

- 1 operand lahko v pomnilniku, drugi v registru

- **pomnilniško-pomnilniške**

- vsak operand lahko v pomnilniku
- zapleteni ukazi, CISC (npr. VAX)

16

- **Načini naslavljanja**

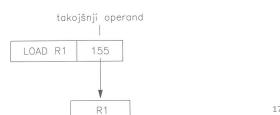
- Kako je v ukazu podana informacija o operandih

- Tičejo se predvsem pomnilniških operandov

- pri registrskih je enostavno

- 1. **Takošnje naslavljanje** (immediate addressing)

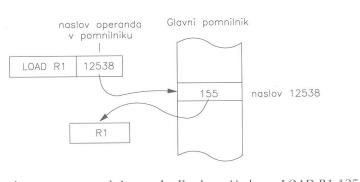
- operand je v ukazu podan z vrednostjo (je kar del ukaza)
- **takošnji operandi (literali)** so kar konstante
- npr. ukaz LOAD R1, #155, pri katerem se konstanta 155 prenese v register R1



17

- 2. **Neposredno naslavljanje** (direct addressing)

- tudi **absolutno**
- operand je podan z naslovom
  - če je to naslov registra, je to **registrsko naslavljanje**
- primerno za operande, ki se jim ne spremnjajo naslovi



18

- Težave:

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

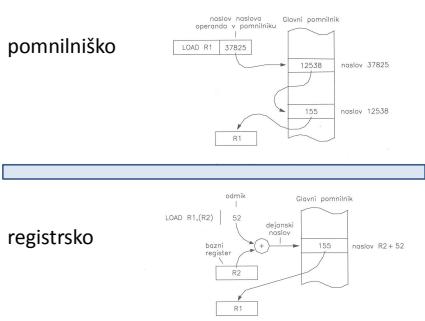
19

### 3. Posredno naslavljjanje (indirect addressing)

- v ukazu je naslov lokacije, na kateri je shranjen naslov operanda
  - **Pomnilniško posredno naslavljjanje**, če gre za naslov pomnilniške lokacije (herodno, ni pogosto)
  - **Registrsko posredno naslavljjanje**, če gre za naslov registra in **odmik** (displacement) – iz tega dvojega se izračuna pomnilniški naslov
    - » najpogosteji način naslavljanja
    - » imenuje se tudi **relativno naslavljjanje**
- naslov operanda določen relativno na vsebino registra

20

Posredno naslavljjanje:



21

- Glavne vrste relativnega naslavljanja:

### 3.1 Bazno naslavljjanje (base addressing)

- najpogosteje
- reče se tudi **naslavljanje z odmikom** (displacement addressing)
- v ukazu sta podana tudi register R2 in odmik D
- naslov operanda  $A = R2 + D$ 
  - k vsebinu registra R2 prištejemo odmik D
- R2 je **bazni register**, A pa **dejanski naslov** (effective address)

22

### 3.2 Indeksno naslavljjanje (indexed addressing)

- odmik D obsega celoten pomnilniški naslovni prostor
- $A = R2 + R3 + D = R2 + D_1$
- $D_1$  je dolžine pom. naslova
- 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
  - $R3 \leftarrow R3 + \Delta$  in  $R3 \leftarrow R3 - \Delta$
  - $\Delta$  je dolžina operanda, merjena v številu pomnilniških besed (**korak indeksiranja**)

23

### 3.3 Pred-dekrementno naslavljjanje (pre-decrement addressing)

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

### 3.4 Po-inkrementno naslavljjanje (post-increment addressing)

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

– Zadnja 2 načina v paru tvorita **skladovno naslavljjanje** (stack addressing)

- sklad v GP (ne v CPE)
- določeni računalniki imajo register **skladovni kazalec** (stack pointer)

24

### 3.5 Velikostno indeksno naslavljjanje (scaled indexed addressing)

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

25

Še 2 pojma:

#### • Pozicijsko neodvisno naslavljjanje

- pozicijsko neodvisni programi
  - lahko jih prenestimo 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 naslavljjanje

- kot bazni register služi kar programski števec (PC)

26

## Primeri naslavljjanja

Način naslavljjanja	Primer ukaza	Pomen	Kdaj se uporablja
Takošnje	ADD R1,#3	R1 $\leftarrow R1 + 3$	konstante
Neposredno	ADD R1,(1001)	R1 $\leftarrow R1 + M[1001]$	operandi na konst. naslovih
(Neposredno) registrsko	ADD R1,R2	R1 $\leftarrow R1 + R2$	operandi v registrih
Pomnilniško posredno	ADD R1,@(1001)	R1 $\leftarrow R1 + M[M[1001]]$	v pomnilniku kazalec
Bazno	ADD R1,100(R2)	R1 $\leftarrow R1 + M[R2+100]$	lokalni operandi
Bazno brez odmika	ADD R1,(R2)	R1 $\leftarrow R1 + M[R2]$	kazalci ali izračunani naslovi
Bazno posredno	ADD R1,@(R2)	R1 $\leftarrow R1 + M[M[R2]]$	v R2 naslov kazalca
Indeksno	ADD R1,100(R2+R3)	R1 $\leftarrow R1 + M[R2+R3+100]$	dostop do elementov polja
Indeksno brez odmika	ADD R1,(R2+R3)	R1 $\leftarrow R1 + M[R2+R3]$	dostop do elementov polja

27

## 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

28

- 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

29

- 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
      - z AND lahko tudi izločamo bite iz besede
      - z OR lahko tudi vstavljamo bite v besedo

30

- Pomiki:
  - **navadni pomiki** (shift)
  - **rotacije** (rotate)
- Levi in desni pomiki
- Pri desnih navadnih pomikih razlikujemo:
  - **logične pomike** (logical shifts)
    - » v izpraznjena mesta gredo ničle
  - **aritmetične** (arithmetical shifts)
    - » najbolj levi bit se ne spreminja in se vstavlja v izpraznjena mesta (število smatramo kot predznačeno)
    - » to je deljenje z večkratniki števila 2
- Levi pomiki predstavljajo množenje z (večkratniki) 2

31

- S pomiki in seštevanjem/odštevanjem je možno realizirati tudi množenje/deljenje
- Logični pomiki se uporabljajo tudi za izločanje/vstavljanje bitov iz/v operand

32

## 2. Prenosi podatkov (data transfer)

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

33

### **3. Kontrolne operacije.**

- spreminja vrstni red ukazov (pomembno!)
- 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

34

**3.2 Brezpogojni skoki** (uncond. branch, jump).

**3.3 Klici in vrnite 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

### **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

35

### **5. Sistemske operacije.**

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

### **6. Vhodno/izhodne operacije.**

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

36

- 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

37

## D5: Vrsta in dolžina operandov

- Vrste operandov:
  - bit**
    - v višjih jezikih jih običajno ni
    - koristno pri sistemskih operacijah
  - znak**
    - običajno 8-bitni ASCII
    - več znakov tvori **niz** (string)
  - celo število**
    - predznačeno (signed) ali nepredznačeno (unsigned)
    - dolžine 8, 16, 32, 64 bitov
  - realno število**
    - št. v plavajoči vejici (običajno po standardu IEEE 754)
    - enojna natančnost (single precision) 32 bitov, dvojna natančnost (double precision) 64 bitov; obstajajo tudi 128-bitna
  - desetiško število**
    - v 8 bitih 2 BCD števili ali 1 ASCII znak

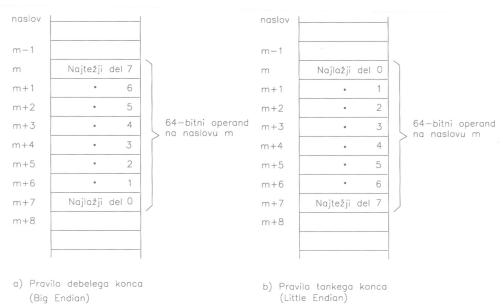
38

- Operandi dolžin večkratnikov 2 (bitov) 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

39

- **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

40



41

- **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 (naslov XX..XX000)
      - **poravan** (aligned) operand
      - sicer **neporavan** (misaligned)
      - » potreben več kot en dostop
      - » pri nekaterih računalnikih se celo sproži past

42

## Zgradba ukazov

- **Zgradba ali format ukaza**
  - 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 doličih pom. besedah: dolžina ukaza  $\frac{1}{2}$  ali  $\frac{1}{4}$  besede
  2. Število eksplisitnih operandov v ukazu
  3. Vrsta in število registrov v CPE
    - št. registrov vpliva na št. bitov za naslavljjanje
  4. Dolžina pom. naslova
    - predvsem, če se uporablja neposredno naslavljjanje

43

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

44

- 3 načini:
  1. Spremenljiva dolžina
    - št. eksplisitnih operandov spremenljivo
    - različni načini naslavljanja
    - veliko formatov
      - npr. 1..15 bajtov pri 80x86, 1..51 VAX
    - kratki formati za pogoste ukaze
  2. Fiksna dolžina
    - št. eksplisitnih operandov fiksno
    - majhno št. formatov (RISC)
      - Alpha, ARM, MIPS, PowerPC, SPARC
  3. Hibridni način

45

Spremenljiva dolžina:

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

Fiksna dolžina:

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

Hibridno:

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
----------	--------------------	------------------	------------------

46

- **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

47

## 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
  - DEC Alpha, IBM/Motorola Power PC, MIPS R4000
- Oboji imajo svoje prednosti in slabosti
  - na začetku so bili računalniki tipa CISC, RISC pa so se pojavili kasneje
    - RISC so enostavnnejši in imajo hitrejše ukaze, vendar pa program potrebuje več ukazov

48

- 2 ugotovitvi v 80. letih:
  1. **Stalno povečevanje števila ukazov**
    - IAS (1951): 23 ukazov in 1 način nasi.
    - 70. leta: stotine ukazov
  2. **Velik del ukazov redko uporabljan**

49

- Razlogi za povečevanje števila ukazov:
  1. **Semantični prepad**
    - v 60. letih so proizvajalci zato povečevali št. ukazov
  2. **Mikroprogramiranje**
    - dodajanje novih ukazov preprosto
  3. **Razmerje med hitrostjo CPE in GP**
    - faktor vsaj 10
    - kompleksen ukaz hitrejši kot zaporedje preprostih ukazov

50

- Razlogi za zmanjševanje števila ukazov:
  1. **Težave prevajalnikov**
    - velik del ukazov redko uporabljan
  2. **Pojav predpomnilnikov**
    - v primeru zadetka v PP je dostop skoraj enako hiter kot do mikroukazov
  3. **Uvajanje paralelizma v CPE**
    - npr. cevovod (lažja realizacija pri preprostih ukazih)

51

- **Definicija arhitekture RISC:**

1. Večina ukazov se izvrši v enem ciklu CPE
  - lažja real. cevovoda
2. Registrsko-registrska zasnova (load/store)
  - zaradi zahteve 1
3. Ukazi realizirani s trdo ozičeno logiko
  - ne mikroprogramsko
4. Malo ukazov in načinov naslavljjanja
  - hitrejše in enostavnejše dekodiranje in izvrševanje
5. Enaka dolžina ukazov
6. Dobri prevajalniki
  - upoštevajo zgradbo CPE