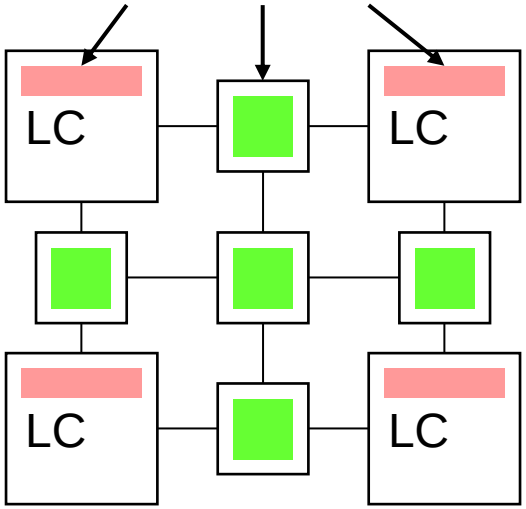
Programirljiva vezja / procesorji

- programirljivo vezje
  - programski biti določajo strukturo (konfiguracijo)

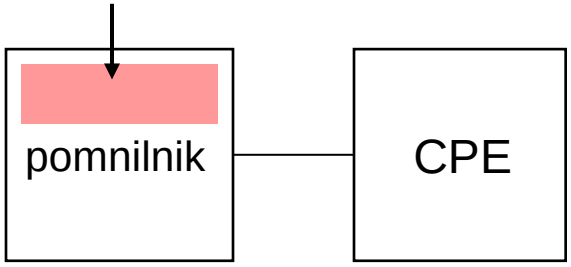


delovanje logičnih celic (LC)

povezave znotraj LC

povezave v povezovalni mreži

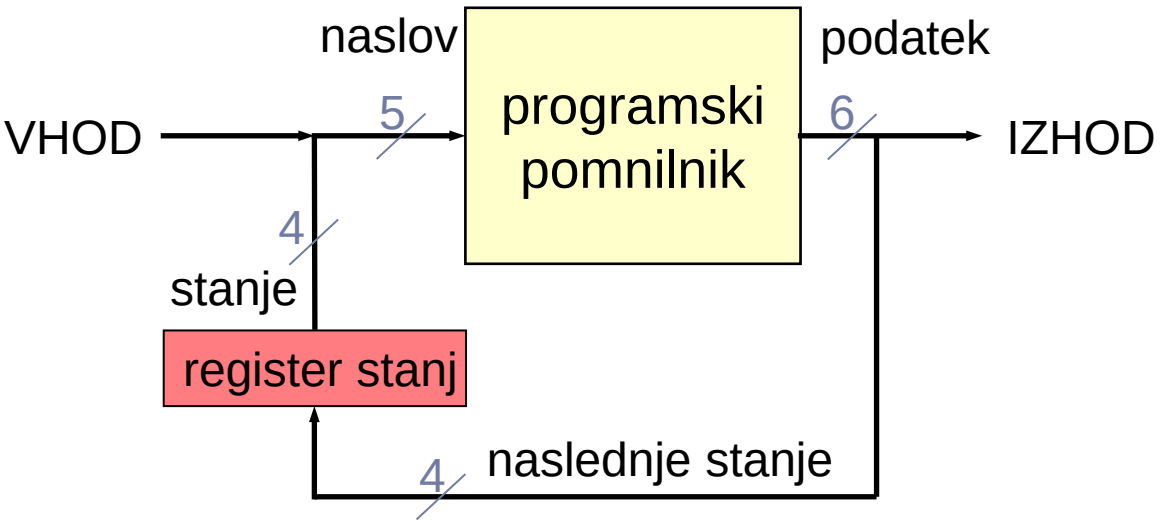
- mikroprocesor
  - program določa ukaze, ki jih CPE izvaja



Mikro-programirano krmilno vezje

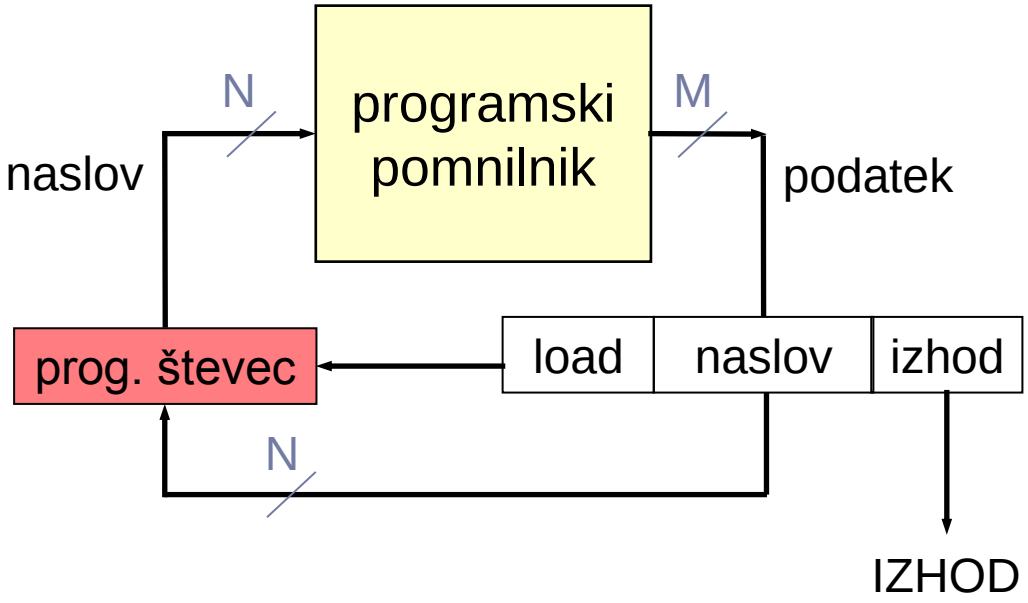
- Krmilni avtomat narejen z LUT je mikro-programiran
  - *mikro-programirano vezje* ima krmilne vrednosti shranjene v pomnilniku vrste ROM ali RAM

- *mikroukazi* so besede v pomnilniku
- *mikroprogram* je zaporedje *mikroukazov*
- *v programski pomnilnik* vrste RAM lahko vpisujemo *mikroukaze*

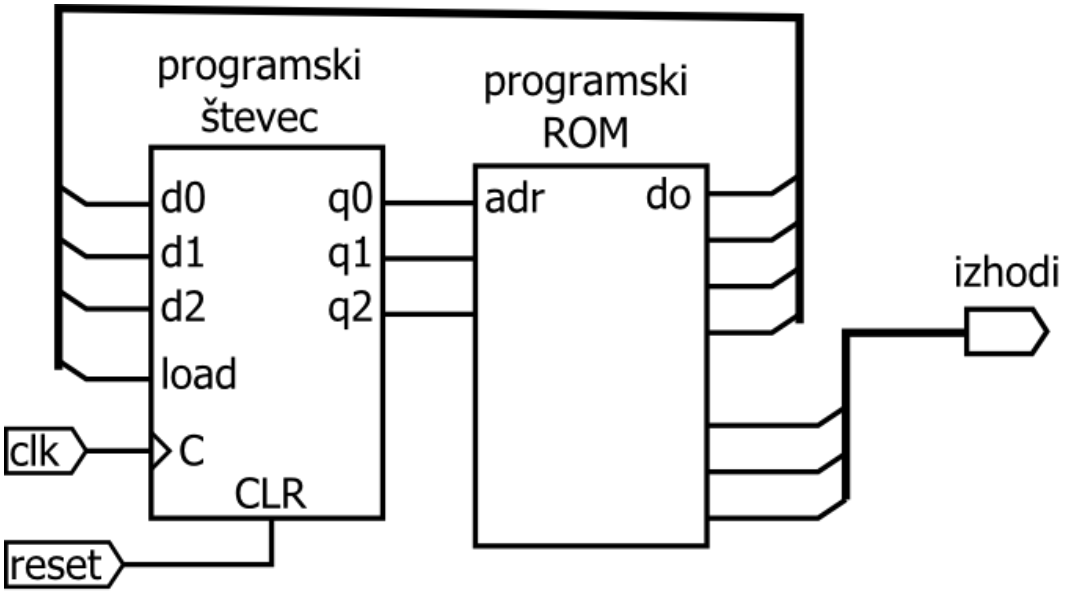


Mikro-programiran krmilnik s števcem

- Programski števec omogoča izvedbo skokov
  - izvrševanje algoritmov, ki nimajo vseh mikroukazov v zaporedju



Izvedba semaforja z mikrosekvenčnikom



adr	d3:d0	load	izhod
000	0 0 0	0	0 1 0
001	0 0 0	0	0 0 0
010	0 0 0	0	0 1 0
011	0 0 0	0	0 0 0
100	0 0 0	0	1 0 0
101	0 0 0	0	1 1 0
110	0 0 0	0	0 0 1
111	1 0 0	1	0 1 0

skok

rdeča

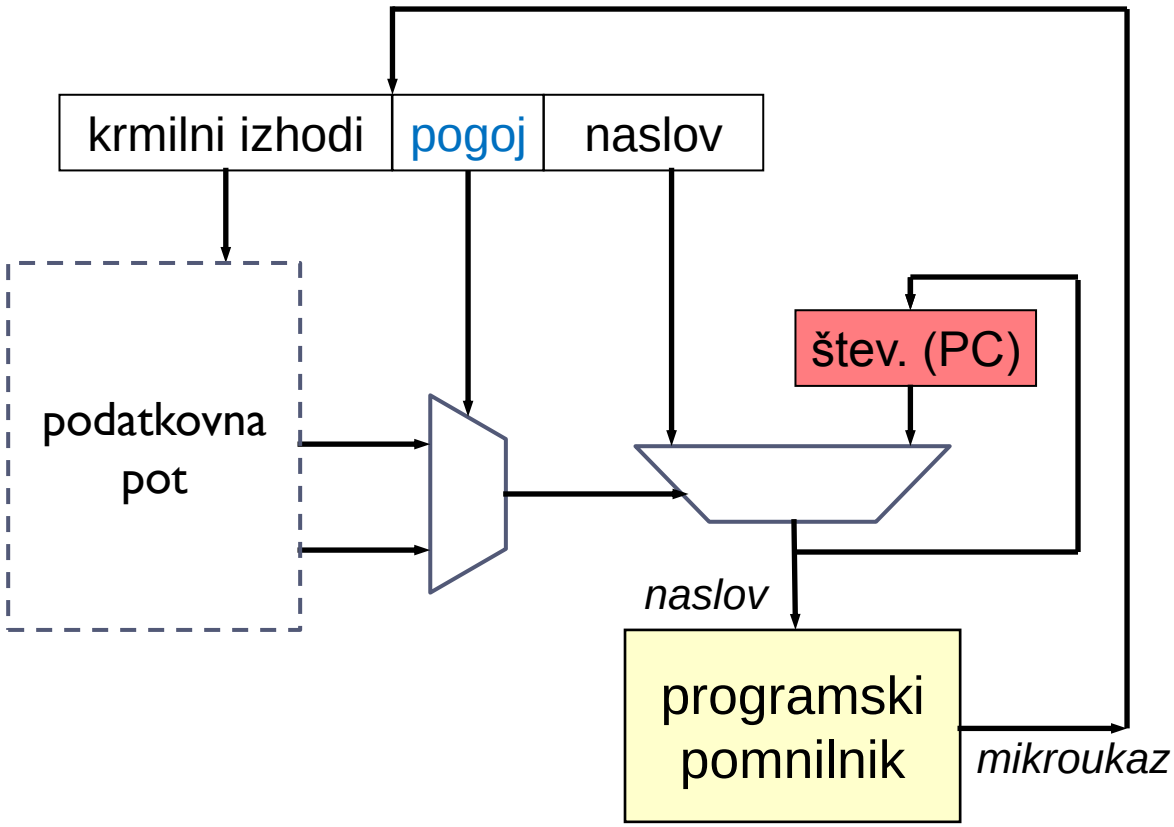
rumena

zelena

Mikro-programiran krmilnik s števcem

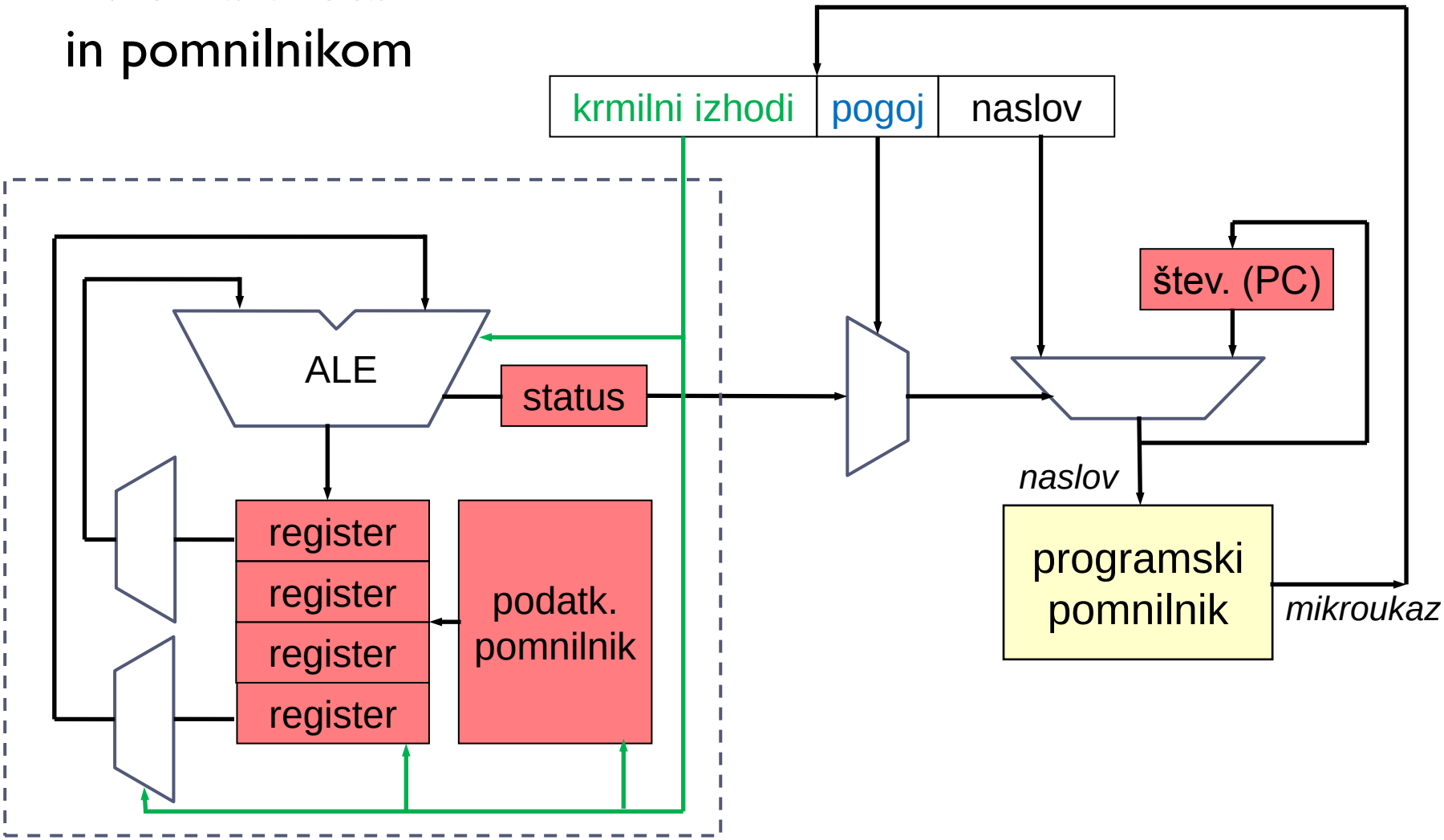
Mikroukazi

- mikroprogram se izvaja kot zaporedje mikroukazov
- mikroukazi določajo krmilne izhode
- *skočni ukazi* ob pogoju izvršijo skok na nov naslov



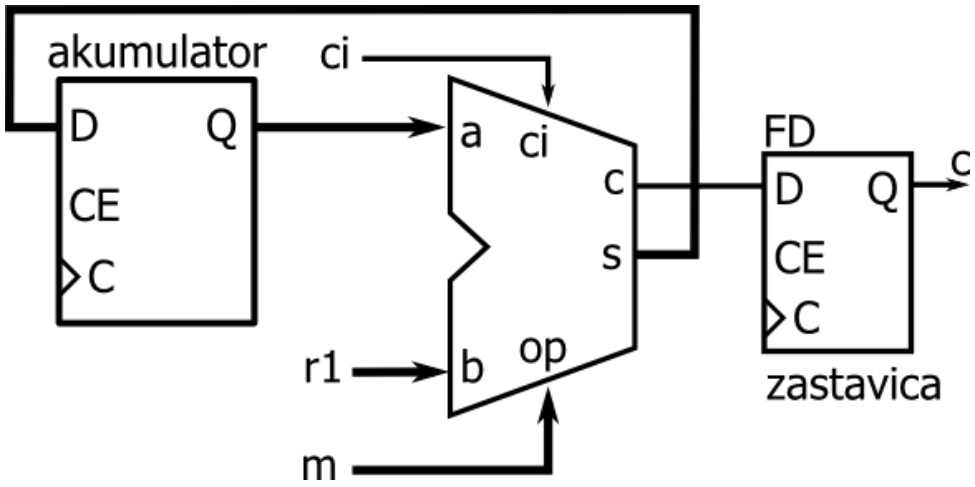
Mikro-programirana krmilna (procesna) enota

- izvršilna enota z ALE in pomnilnikom



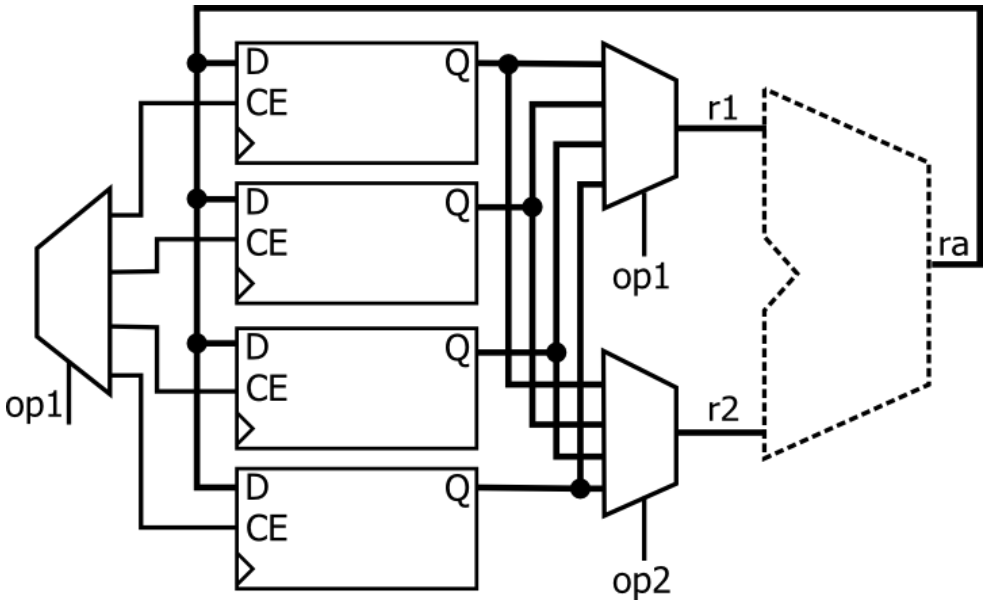
ALE z akumulatorjem

- Rezultat operacije se vedno shrani v akumulator
  - v akumulatorju je tudi eden izmed operandov
  - rezultat nastavi zastavice (nič, negativen, prenos)



ALE z registri

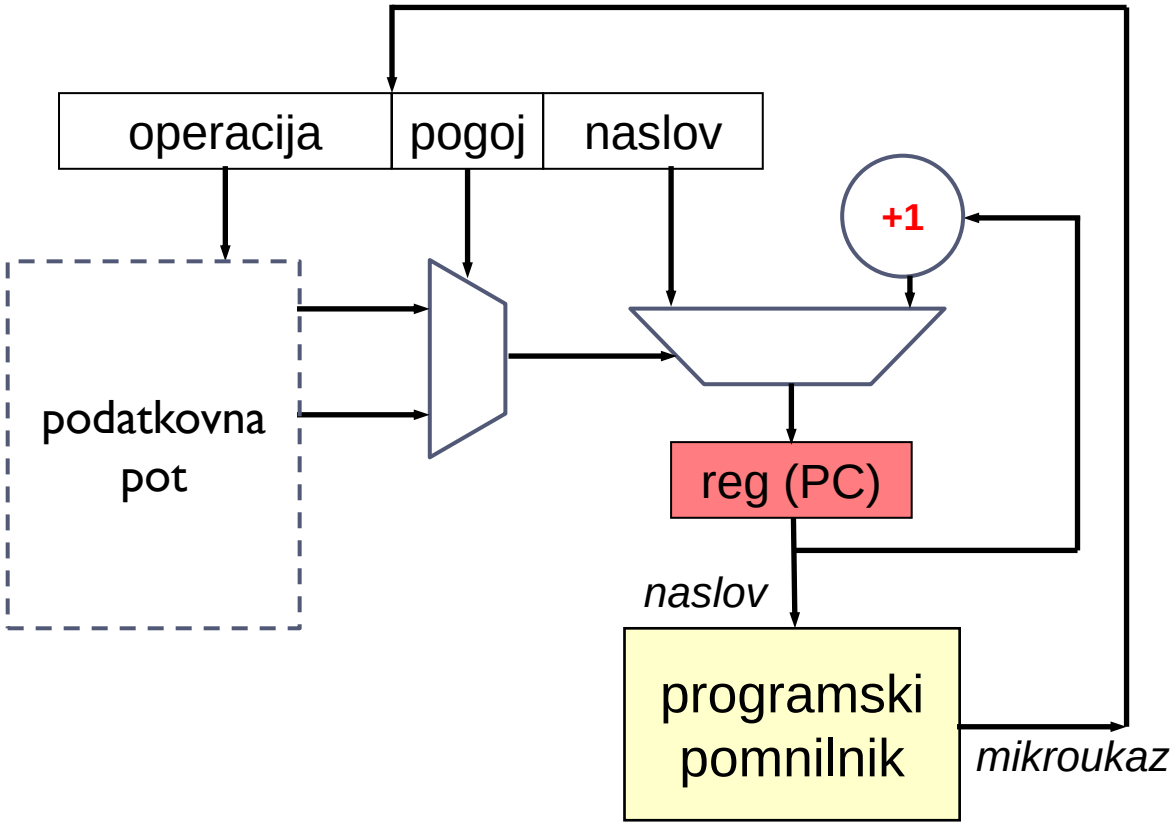
- Krmilni sig. izbirajo operacijo, operande in izhodni register
  - operand je v registru ali v pomnilniku



Kode in koraki izvajanja mikrooperacij

- za direktno krmiljenje enot z mikroukazi potrebujemo veliko bitov
  - npr. 4 bite za ALE, 12 za registre, 16 za pomnilnik….
- krmilni signali so kodirani
  - potrebujemo ukazni dekoder
- mikroukazi se izvršijo v več urnih ciklih
  - bolj optimalna kritična pot in višja frekvenca ure
  - nekaterih operacij ne moremo narediti v enem ciklu
    - npr. branje in pisanje v pomnilnik

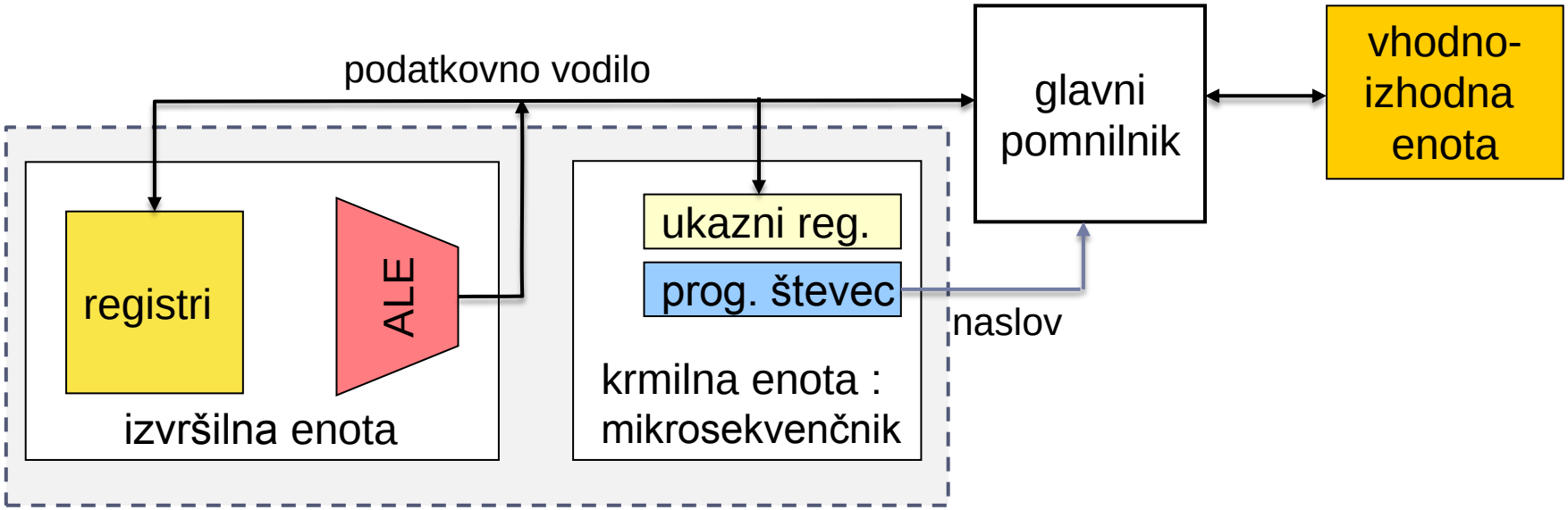
Mikrosekvenčnik in podatkovna pot



- Podatkovna pot izvaja registrske (mikro)operacije
  - registri, ALE, podatkovni pomnilnik

Von Neumannov model računalnika

- Centralno procesna enota (CPE) in glavni pomnilnik
  - vhodno – izhodna enota skrbi za komunikacijo z zunanjostjo



- delovanje CPE določa nabor ukazov
  - ukazi so prilagojeni programskim jezikom (C/C++, Java)

Osnovni gradniki mikroprocesorja

- Mikroprocesor na integriranem vezju vsebuje izvršilno in krmilno enoto
- Mikroprocesor potrebuje za delovanje:
  - zunanjo uro in reset
  - zunanji pomnilnik s programskimi ukazi in podatki
  - vhodno in izhodno (periferno) enoto za komunikacijo z okolico
    - periferna enota je lahko del pomnilnika (na določenih naslovih)
    - ali pa poteka komunikacija preko posebnih V/I ukazov
- V praksi potrebujemo vsaj dve vrsti pomnilnika
  - za program takšnega, ki ohranja vsebino (ROM, Flash)
  - za delovne podatke pa pomnilnik s hitrim branjem in pisanjem (RAM – Random Access Memory)

Vrste mikroprocesorjev

- Mikroprocesorji v zmogljivih računalnikih – CISC
- Mikroprocesorji v vgrajenih sistemih – RISC

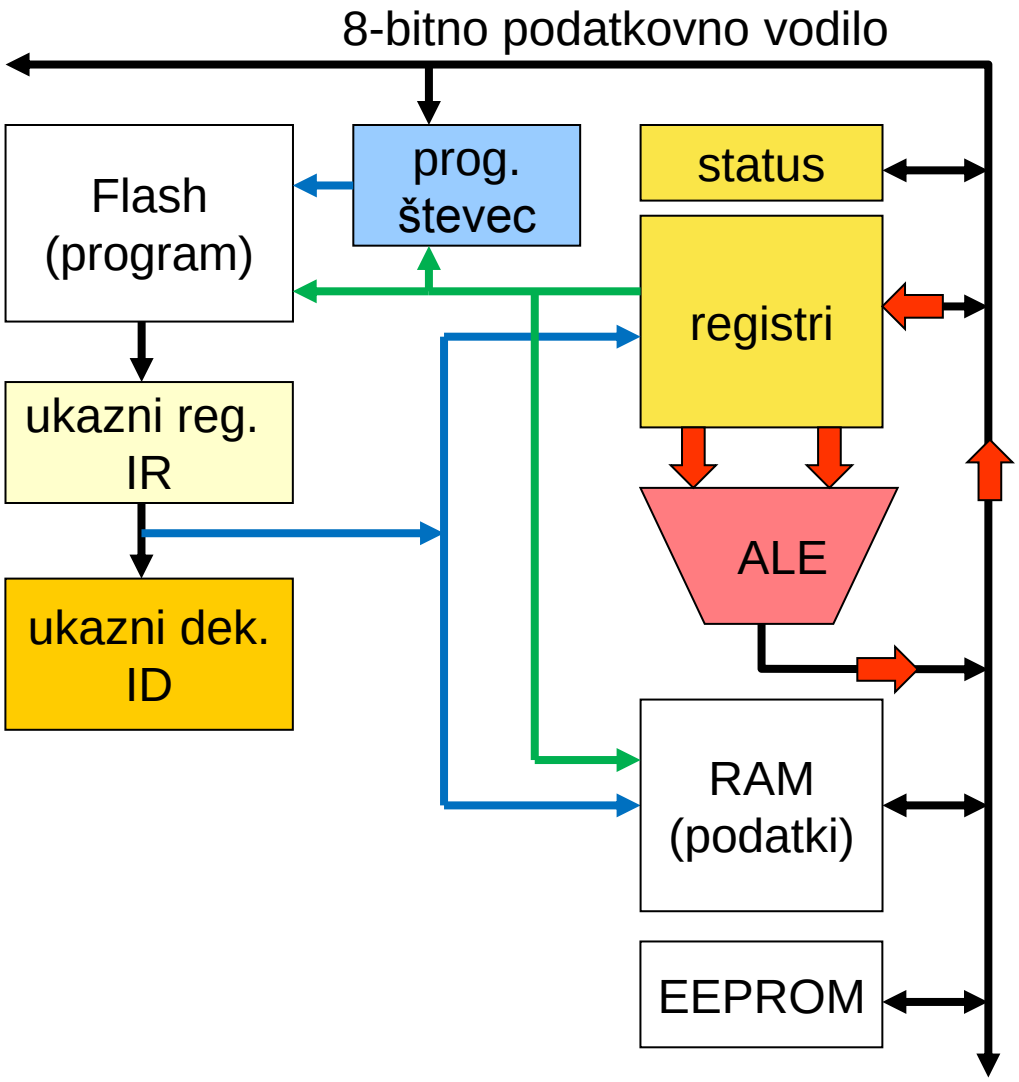
- Complex Instruction Set Computer - CISC
  - veliko število ukazov
  - enostavne mikrooperacije do sestavljenih operacij
  - čas za izvajanje ukazov je zelo različen
- Reduced Instruction Set Computer - RISC
  - manjši nabor ukazov
  - vse operacije so enostavne in učinkovite za izvrševanje
  - večina ukazov se izvede v enem urnem ciklu

Mikroprocesorji za vgrajene naprave

- Mikrokrmilniki so sistemi na integriranem vezju, ki vsebujejo
  - mikroprocesorsko jedro (izvršilno in krmilno enoto),
  - programski in podatkovni pomnilnik,
  - ter različne V/I vmesnike:
    - vzporedna vrata (Port)
    - zaporedne komunikacijske vmesnike: I2C, SPI, UART
    - analogno / digitalne (A / D) in D / A pretvornike
    - modulatorje (PWM), časovnike, števce…
    - komunikacijske krmilnike: Ethernet MAC, USB
- Primer mikrokrmilnikov
  - Atmel AVR, 8-bitni procesor na razvojnem sistemu Arduino
  - ARM-7, 32-bitni procesor na razvojnem sistemu Š-ARM

Delovanje CPE mikrokrmilnika AVR

- programski števec vsebuje naslov naslednjega ukaza
- IR vsebuje naslednji ukaz
- ID vsebuje trenutni ukaz
- Registri: R0-R31
- ALE – glej označeno interno podatkovno pot
- Flash programski pomnilnik 16 oz. 32 bitni ukazi
- RAM vsebuje delovne podatke
- EEPROM trajni podatki
  - počasen dostop, omejeno število vpisov v pomnilnik



Programiranje v zbirniku

- Zbirnik je nizkonivojski programski jezik
- Mikroukazi so odvisni od arhitekture (različni za različne procesorje)
- Zbirnik je med jezikom C in strojno kodo
- Uporaba v manjših vgrajenih sistemih (npr. PIC, AVR)
- Pri velikih projektih je delo v zbirniku težko, nefleksibilno, prevajalniki jezika C pa so dobro optimizirani
- Zbirnik se uporablja za:
  - nizkonivojske rutine
  - računsko zahtevna opravila (matematika, grafika)
  - reverzni inženiring

Ukazi v zbirniku

- Ukazi v zbirniku predstavljajo procesorske mikrooperacije
- Naloži neposredno konstanto ($Rd <= K$)
- LDI Rd, K
  - strojna koda: 1110 bbbb rrrr bbbb
    - omejitve: registri R16-R31; konstanta K < 255
- LDI R16, $2C
  - koda: 1110 0010 0000 1100 ali $E20C

- Seštej in shrani v Rd ($Rd <= Rd + Rr$)
- ADD Rd, Rr
  - strojna koda: 0000 11rd dddd rrrr
    - Rd in Rr sta katerakoli registra R0-R31
- ADD R16, R17
  - ddddd je 10000, rrrrr je 10001, koda ukaza: $0F01

Strojni program

- Naloži program v pomnilnik
  - na naslov 0 shranimo $E20C
  - na naslov 1 shranimo $E01F
  - na naslov 2 shranimo $0F01

```
0000: $E20C LDI R16, $2C
0001: $E01F LDI R17, $0F
0002: $0F01 ADD R16, R17
```

- Izvedemo zaporedje treh ukazov
  - nastavi prog. števec na 0 in izvedi 3 ukazne cikle (prenos/izvedi)
- Pregled rezultata:

```
R16  = R16 + R17
     = $2C + $0F
     = $3B
```

- Kaj se bo izvedlo po teh treh ukazih?
  - ne vemo, odvisno kaj je zapisano v pomnilniku Flash…

Vhodno-izhodne (periferne) enote

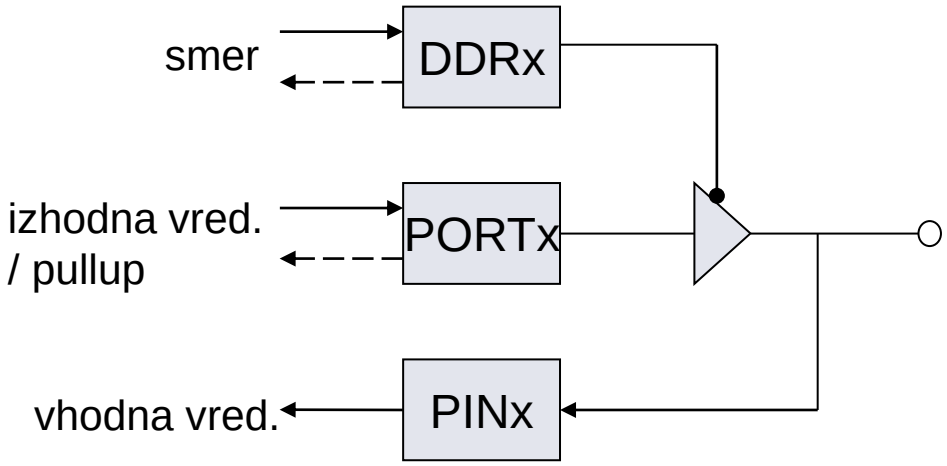
- vzporedna vrata (I/O port) vsebujejo 3 registre

nastavi 4 vh. in 4 izh. bite →
izhodi →
beri vhode →

```
LDI R19, $F0
OUT DDRB, R19

LDI R21, $50
OUT PORTB, R21

IN R20, PORTB
```



DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)