Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

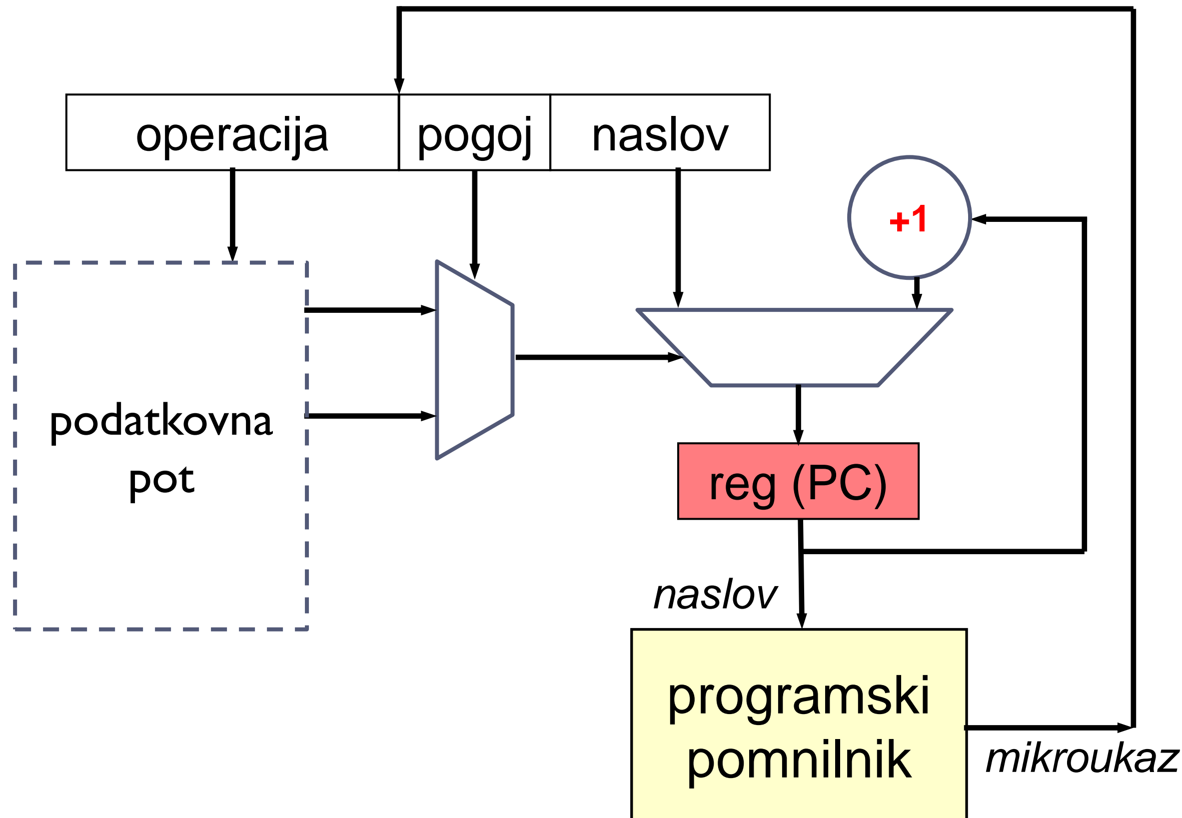
Digitalni Elektronski Sistemi

Procesorji

Model računalnika, mikrokrmilnik

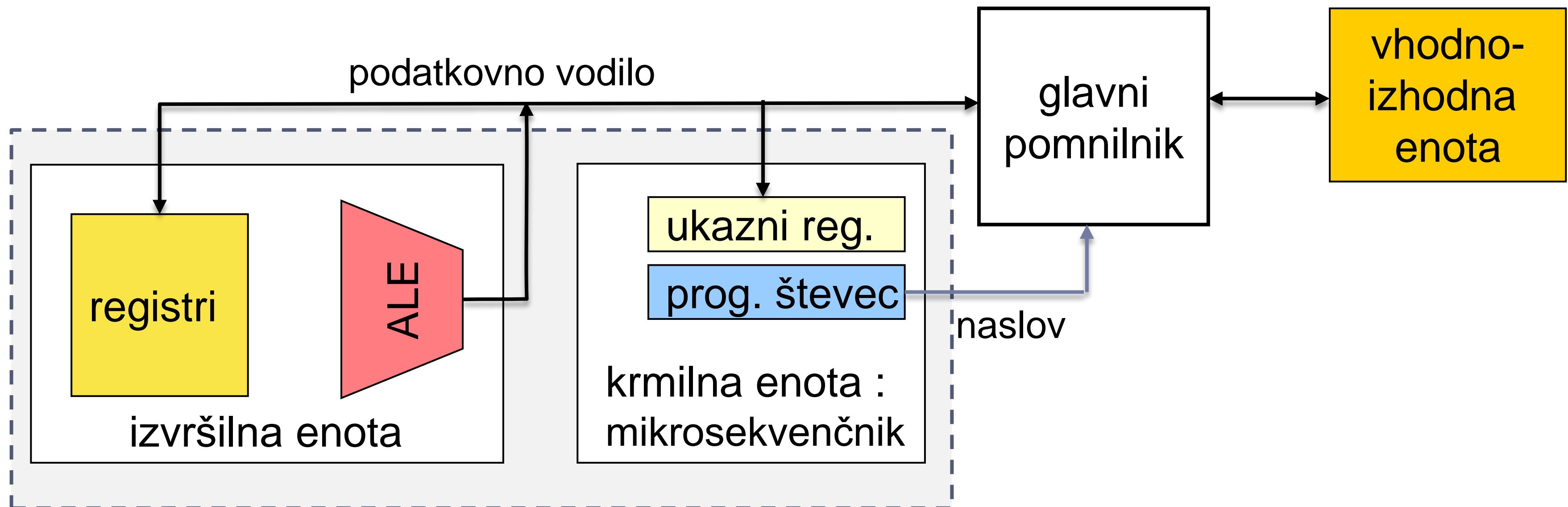
CPE = mikrosekvenčnik + podatkovna pot

- 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



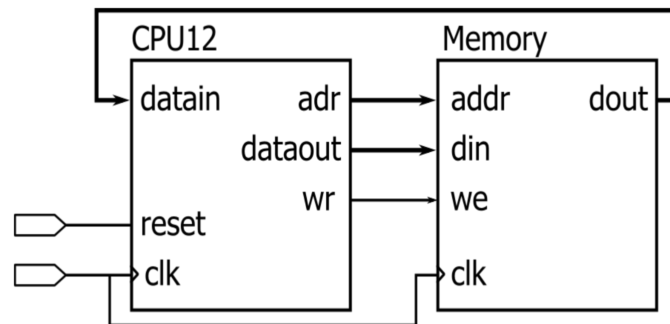
- delovanje CPE določa nabor ukazov
  - ukazi so prilagojeni programskim jezikom (C/C++, Java)

Vrste mikroprocesorjev

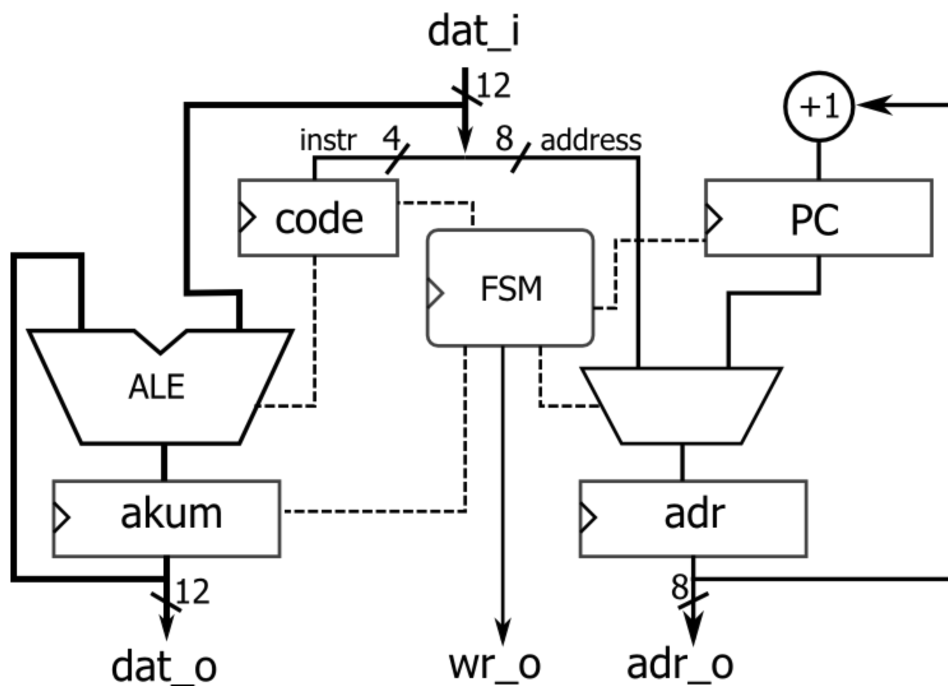
- Mikroprocesorji v zmogljivih računalnikih – CISC
- Mikroprocesorji v vgrajenih sistemih – RISC

  - 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

Enostaven CPE z akumulatorjem



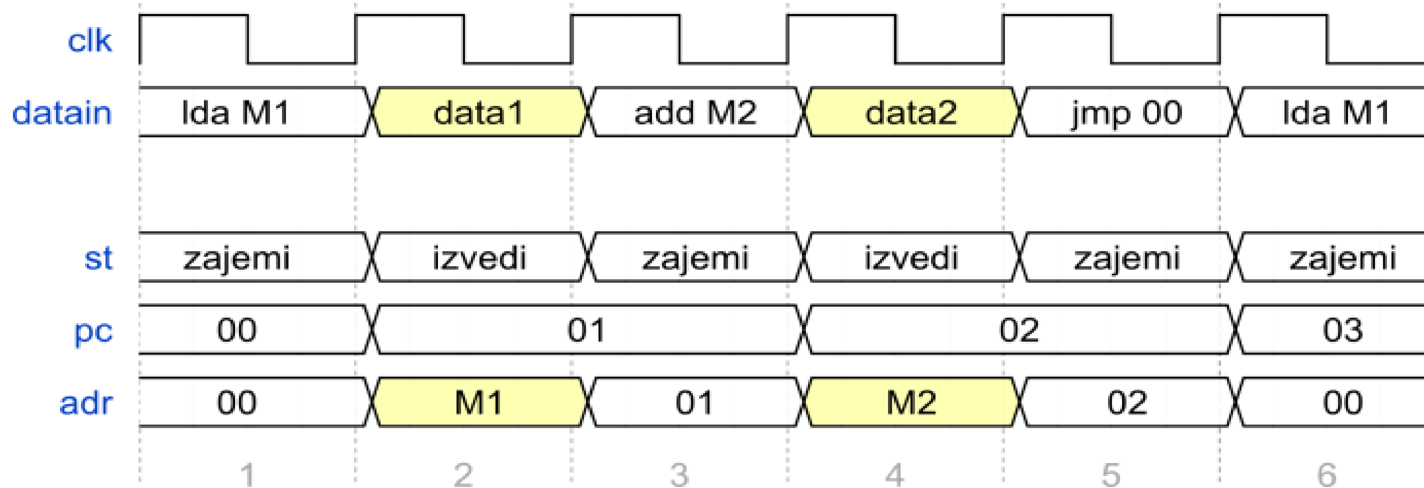
N-bitna podatkovna pot, npr. N=12



Nabor ukazov

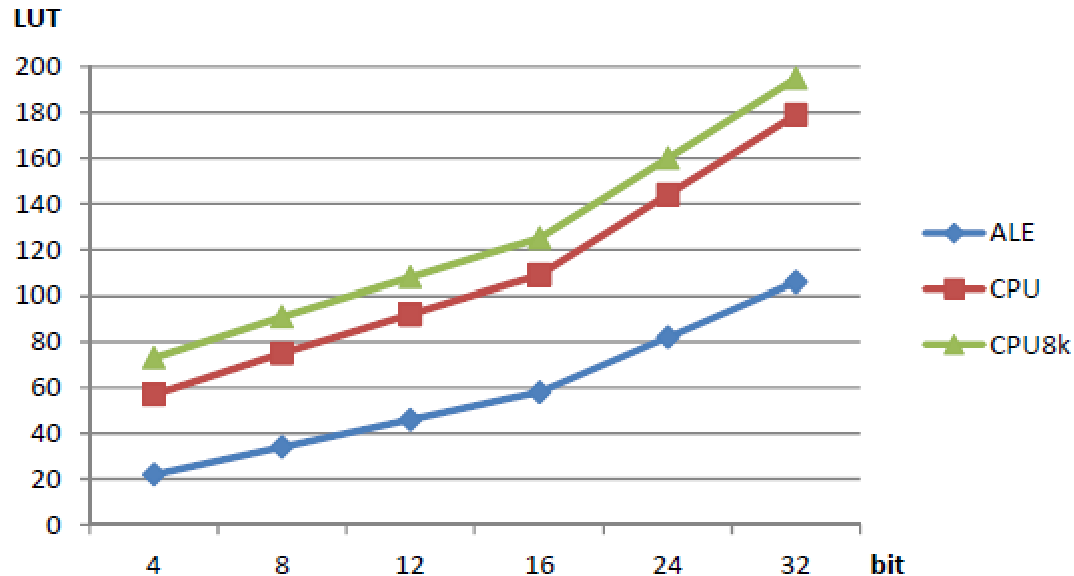
```
subtype  koda is unsigned(3 downto 0);
constant  lda:     koda := "0001";     --  a = [M]
constant  sta:     koda := "0010";     -- [M] = a
constant  add:     koda := "0100";     -- a = a + [M]
constant  sub:     koda := "0101";     -- a = a - [M]
constant  anda:    koda := "0110";     -- a = a and [M]
constant  ora:     koda := "0111";     -- a = a or [M]
constant  jmp:     koda := "1000";     -- jump
constant  jze:     koda := "1001";     -- jump if a=0
```

- časovni potek izvajanja kode

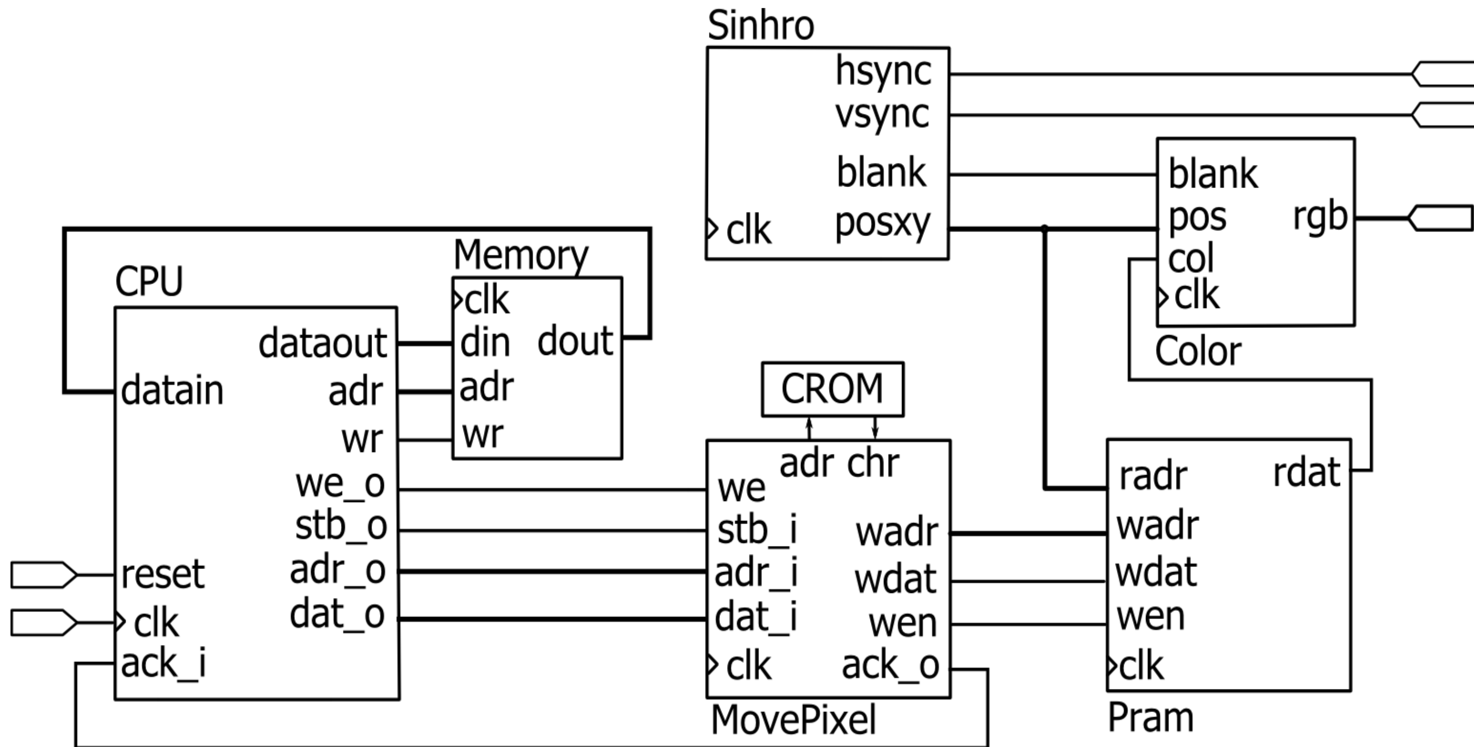


VHDL opis in sinteza procesorja

```
if st=zajemi then      -- shrani kodo ukaza
            code <= instr;
elsif st=izvedi then  -- izvedi ukaz z akumulatorjem
            case code is
              when  lda          =>  akum <= dat_i;
              when  add          =>  akum <= akum + dat_i;
              when  sub          =>  akum <= akum - dat_i;
              when  anda         =>  akum <= akum and dat_i;
              when  ora          =>  akum <= akum or dat_i;
              when others => null;
            end case;
end if;
```



Uporaba CPU v grafičnem krmilniku



Mikroprocesorji za vgrajene naprave

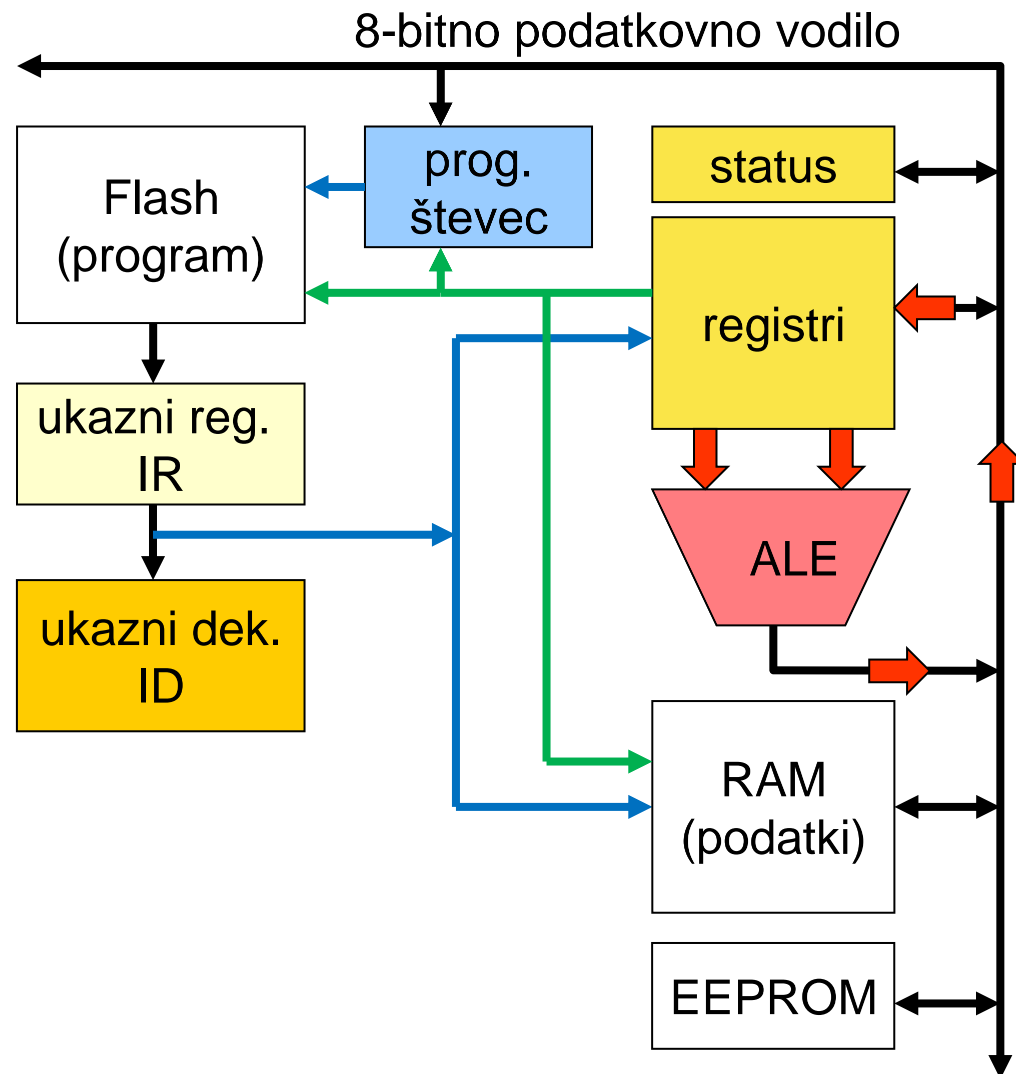
- Mikrokrmilniki so sistemi na integriranem vezju, ki vsebujejo
  - mikroprocesorsko jedro (izvršilno in krmilno enoto),
  - programski in podatkovni pomnilnik,
  - ter različne V/I vmesnike:
    - vzporedna vrata (Port)
    - zaporedne komunikacijske vmesnike: I2C, SPI, UART
    - analogno / digitalne (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

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)

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 vez med programskimi jeziki (npr. C) in strojno kodo, ki je odvisna od arhitekture
- Opis majhne in učinkovite programske kode za
  - nizkonivojske rutine (dostop do perifernih enot)
  - računsko zahtevna opravila (matematika, grafika)
- Pri velikih projektih je delo v zbirniku težko, nefleksibilno, prevajalniki jezika C pa so dobro optimizirani


```
 .cseg
 .org 0

        rjmp      reset     ;reset vector

reset:
        ;setup stack for subroutine usage
        ldi r16,high(RAMEND)
```

Primer ukazov in njihove strojne kode

- 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 — `0000: $E20C LDI R16, $2C`
  - na naslov 1 shranimo $E01F — `0001: $E01F LDI R17, $0F`
  - na naslov 2 shranimo $0F01 — `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…

Ukazi procesorja AVR v zbirniku

- Ukazi v zbirniku predstavljajo procesorske mikrooperacije

aritmet. / logične

a+b	ADD
a-b	SUB
a&b	AND
a|b	OR
a++	INC
a--	DEC
-a	NEG
a=0	CLR
…	…

podatkovni prenos

reg1=reg2	MOV
reg=17	LDI
reg=mem	LDS
reg=*mem	LD
mem=reg	STS
*mem=reg	ST
periferni	IN
periferni	OUT
sklad	PUSH
sklad	POP
…	…

operacije z biti

a<<1	LSL
a>>1	LSR,
Ø C (ni v C)	ROL, ROR
statusni biti	SEI, CLI, CLZ...
ni operacije	NOP
…	…

Skočni ukazi in primeri v jeziku C

- JMP, RJMP: brezpogojni skok

  npr. neskončna zanka:

M_LOOP:
    …**ukazi**…
    jmp M_LOOP
	`while (1) {`
	    `...ukazi...`
	`}`

- CALL, RET: klic podprograma in vrnitev (return)

  npr. podprogram:

M_LOOP:
    …
    CALL FV
    …
FV:…**ukazi**…
    RET
	`void fv() { …ukazi…`
	    `return;`
	`}`
	`void main () {…`
	    `fv();`
	`}`

Pogojni stavek

- Skoči na L1, če je a==b, sicer skoči na L2
  - naredimo s kombinacijo pogojnih in brezpogojnih skokov

M_LOOP: ; primerjaj,
  CPSE a, b ; preskoči, če a=b
  JMP L2
L1:… ; a == b
  JMP M_LOOP
L2:… ; a != b
  JMP M_LOOP
	while (1) {
	  if (a==b) {
	    (L1)
	  } else {
	    (L2)
	  }
	}

CPSE (compare, skip if equal) preskoči naslednji ukaz (JMP L2) če sta operanda enaka, tako da se izvršijo ukazi za oznako L1.

Hitro nastane zmešnjava – NARIŠI DIAGRAM POTEKA!

Pogojni skočni ukazi

- Skok, ki se izvede glede na rezultat prejšnjega ukaza
- Aritmetični in logični ukazi shranijo rezultat in zastavice
- Primerjava (CP) naredi odštevanje, vendar ne shrani rezultat ampak samo postavi zastavice
  - Z=1, če je rezultat 0,
  - N=1,če je rezultat negativen,
  - C=1, če je prišlo do prenosa
- Npr. BRNE: skok, kadar je rezultat različen od 0 (Z=0)
- BREQ, skok, kadar je rezultat enak 0 (Z=1)

Zanke s 16-bitnim števcem

- Primer zanka for(), ki se ponovi 20000-krat

LDI temp0, 32 ; spodnji bajt
LDI temp1, 78 ; zgornji bajt
LOOP:
    ...
    DEC temp0 ; odštej 1
    BRNE LOOP ; skok če !=0
    DEC temp1
    BRNE LOOP
	for (i=20000; i!=0; i--)
	{
	    ...
	};

- Najprej se DEC temp0 ponovi 32-krat, dokler ni 0
- Potem se izvršita vgnezdeni zanki
  - temp0 se ponovi 256-krat, temp1 pa 78-krat
  - 20000 / 256 = 78, ostane 32

Seštevanje 16-bitnih nepredznačenih števil

- Vsak izmed operandov je shranjen v dveh bajtih
- Najprej seštejemo spodnja bajta, nato pa še zgornja bajta z upoštevanjem prenosa od spodnjih
- Npr.
  - prvi operand je v (R1, R0)
  - drugi operand je v (R3, R2)

`ADD R0, R2` `ADC R1, R3`	`unsigned short a, b;` `a = a + b;`

seštej s prenosom

Množenje in deljenje

- Nekateri procesorji AVR imajo 8-bitni strojni množilnik
  - Rezultat se izračuna v dveh urnih ciklih in shrani v R1 R0
    - Npr. R16 vsebuje vrednost 100, R17 vsebuje vrednost 200
    - MUL R16, R17
    - produkt je 16-bitna vrednost, rezultat: R1 <= $4E, R2 <= $20

- Množenje ali deljenje z 2 s pomikanjem bitov
- Pomakni bite v bajtu na levo (LSL) ali desno (LSR)
  - Npr. LDI R16, 0b11011100; (220_{10})
  - LSL R16; rezultat 0b10111000 (184_{10}) in zastavica C=1
  - LSR R16; rezultat 0b01101110 (110_{10}), C=0
    - prazna mesta se zapolnijo z ničlami
    - bit, ki pri prenosu izpade, se shrani v zastavico C

Vhodno-izhodne (periferne) enote

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

nastavi 4 vh. in 4 izh. bite →

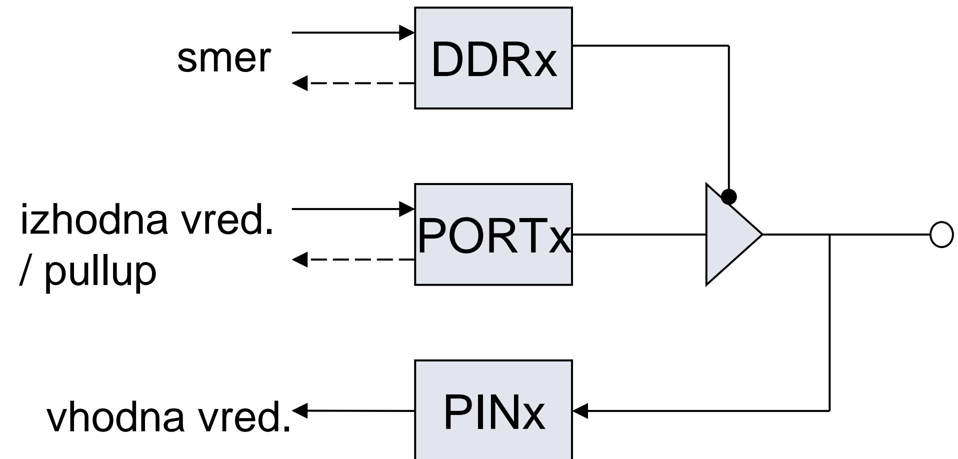
izhodi →

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)