



Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*



Digitalni Elektronski Sistemi

Delovanje procesorja AVR

Zbirnik, primer programa

Procesor Atmel AVR ATmega328

► Procesorsko jedro AVR

- 8 bitno, frekvenca do 20 MHz pri 5V / 10 MHz pri 3.3V
- 131 strojnih ukazov, RISC
- zmogljivost 1 ukaz/cikel (20 MIPS pri 20MHz)
(razen skočnih ukazov)
- periferne enote: časovniki, vzporedni in zaporedni vmesniki, večkanalni A/D pretvorniki in modulatorji PWM

► Pomnilni elementi

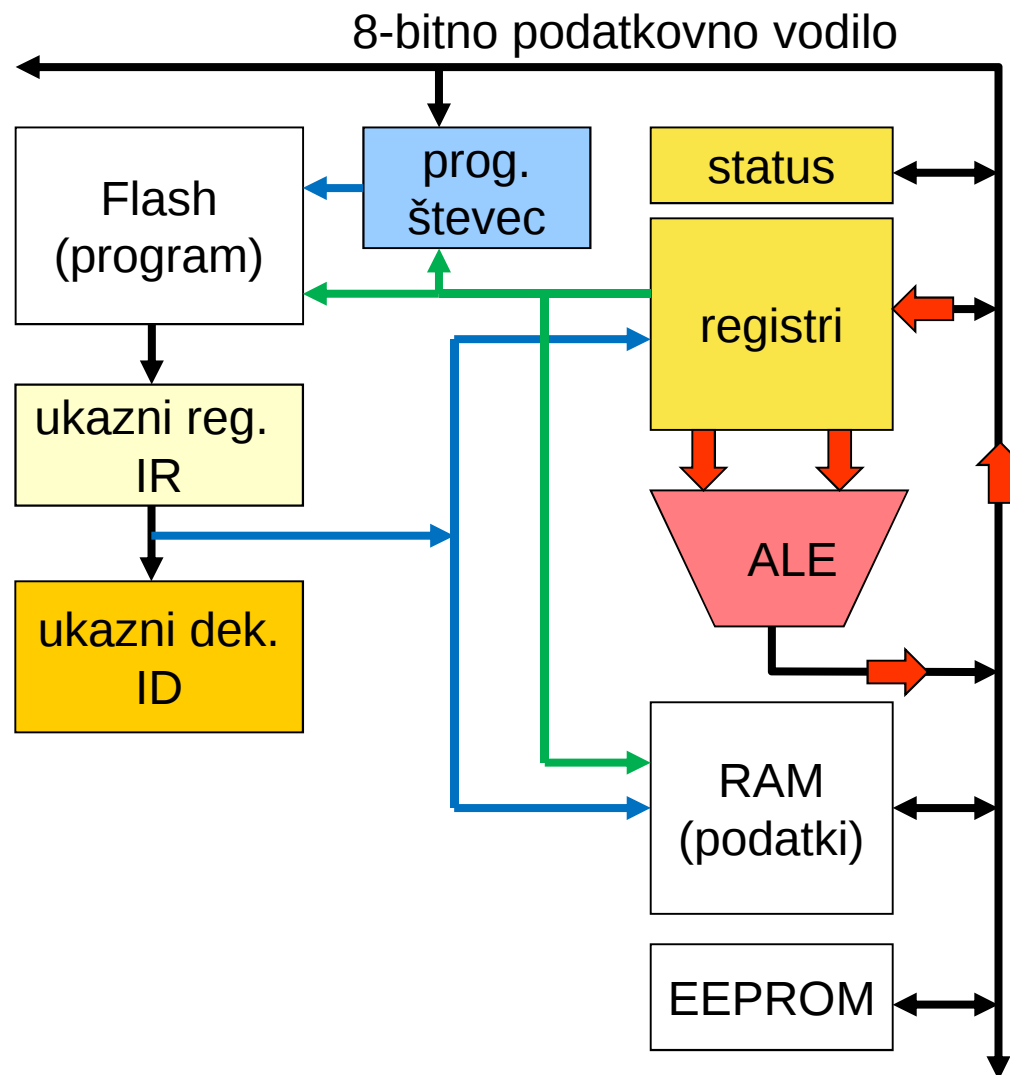
- 32KB Flash (program)
- 1KB EEPROM in 2KB SRAM (podatki)
- 32 registrov
 - 8 bitni registri R0-R31
 - 3 registrski pari za 16 bitne vrednosti (X, Y, Z)

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 izvrševanje ukazov je zelo različen
 - ▶ **Reduced Instruction Set Computer - RISC**
 - ▶ manjši nabor ukazov
 - ▶ vse operacije so enostavne in učinkovite za izvrševanje
 - ▶ večina ukazov se izvrši v enem urnem ciklu

Delovanje AVR procesorskega jedra

- ▶ 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
- ▶ Ukazi 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

aritmet. / logične podatkovni prenos

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

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
...	...

Primer ukazov in njihove strojne kode

- ▶ Naloži neposredno konstanto ($Rd \leq K$)
- ▶ **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 ($Rd \leq Rd + Rr$)
- ▶ **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

- ▶ Izvršimo zaporedje treh ukazov

- ▶ nastavi prog. števec na 0 in izvedi 3 ukazne cikle (prenos/izvrši)

- ▶ Pregled rezultata:

$$\begin{aligned} \mathbf{R16} &= \mathbf{R16} + \mathbf{R17} \\ &= \mathbf{\$2C} + \mathbf{\$0F} \\ &= \mathbf{\$3B} \end{aligned}$$

- ▶ Kaj se bo izvršilo po teh treh ukazih?

- ▶ ne vemo, odvisno kaj je zapisano v pomnilniku Flash...

Sintaksa ukazov v zbirniku (Assembler)

- ▶ 3 možne oblike, [] pomenijo opcijo
 - ▶ [oznaka:] direktiva operandi [;komentar]
 - ▶ [oznaka:] ukaz operandi [;komentar]
 - ▶ [oznaka:] [;komentar]
- ▶ Vsaka vrstica je direktiva ali ukaz ali oznaka ali pa prazna vrstica
 - ▶ komentar začnemo s podpičjem, zaključi se na koncu vrstice

The diagram illustrates the syntax of assembly instructions with the following code and annotations:

```
oznaka      direktiva
            .cseg
            .org 0
            rjmp reset      ;reset vector
reset:
    ;setup stack for subroutine usage
    ldi r16,high(RAMEND)
```

Annotations in the diagram:

- oznaka**: Points to the label `reset:`.
- direktiva**: Points to the directive `.cseg`.
- ukaz in operand**: Points to the instruction `rjmp reset`.
- komentar**: Points to the comment `;reset vector`.

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

- For zanka, ki se ponovi 20000-krat (\$4E20)

<pre>LDI temp0, 0x20 ; spodnji bajt LDI temp1, 0x4E ; zgornji bajt LOOP: ... DEC temp0 ; odštej 1 BRNE LOOP ; skok če !=0 DEC temp1 BRNE LOOP</pre>	<pre>for (i=20000; i!=0; i--) { ... };</pre>
---	--

- Najprej se **DEC temp0** ponovi 32-krat (0x20), dokler ni 0
- Potem se izvršita vgnezdene zanke
 - temp0 se ponovi 256-krat, temp1 pa 78-krat (\$4E), kar je skupaj 19968-krat (če dodamo 32 dobimo 20000)

Dostop do perifernih enot

► vzporedna vrata (I/O port)

nastavi
4 vh. in
4 izh. bite →

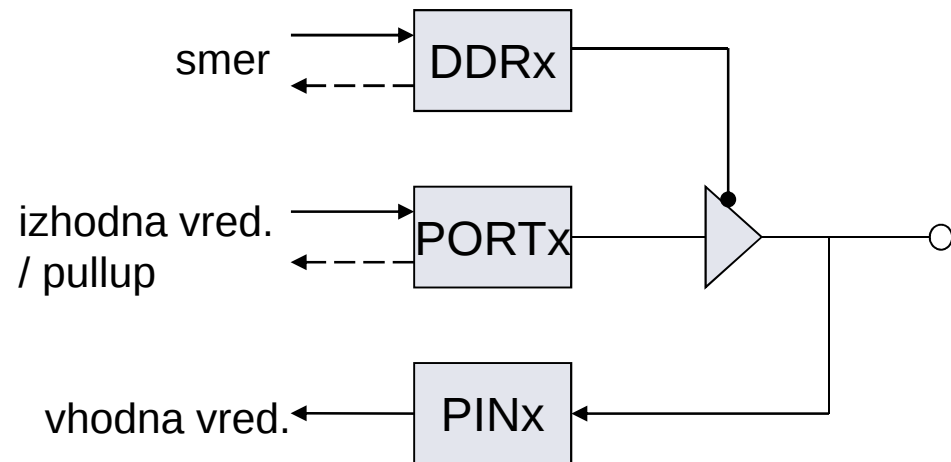
```
LDI R19, $F0  
OUT DDRD, R19
```

beri
vhode →

```
IN R20, PORTC
```

izhodi →

```
LDI R21, $50  
OUT PORTD, R21
```



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)

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	unsigned short a, b;
ADC R1, R3	a = a + b;

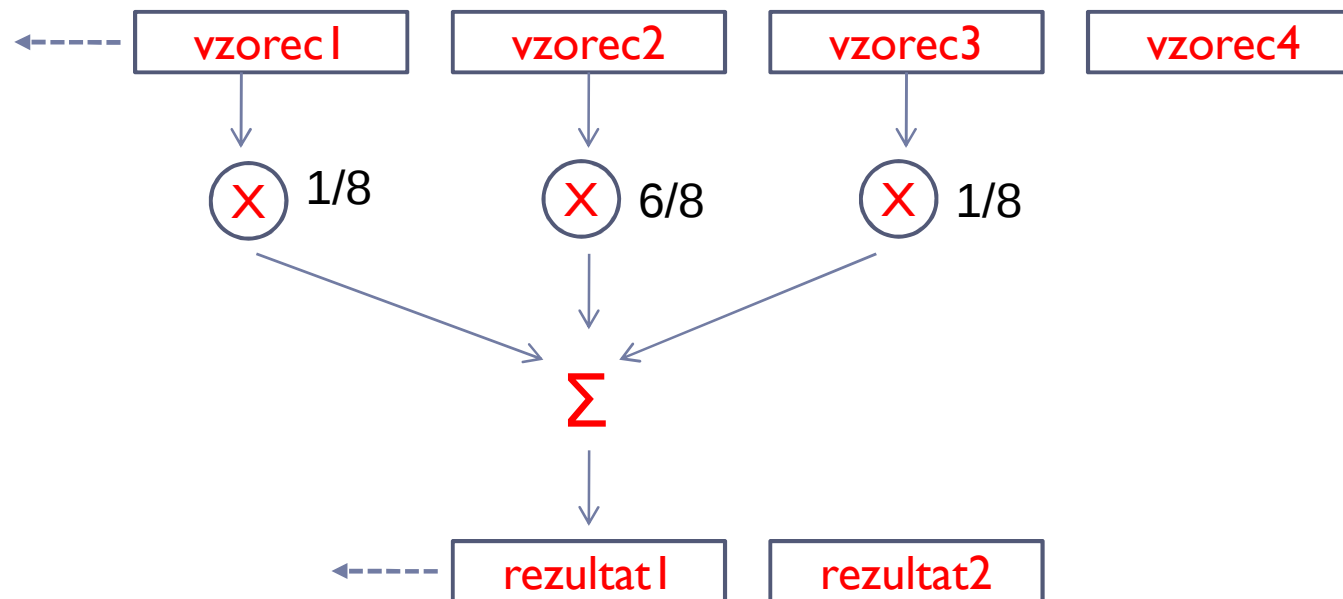
seštej s prenosom

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

Primer: filtriranje podatkovnega toka

- ▶ Zaporedje digitalnih vzorcev pošljemo skozi sito
 - ▶ obdelava zvoka, slike ...
- ▶ Filtriranje izvedemo z matematično konvolucijo, kjer zaporedne vzorce množimo s konstantami in seštejemo
- ▶ Npr. nizko sito s koeficienti $1/8$, $6/8$, $1/8$



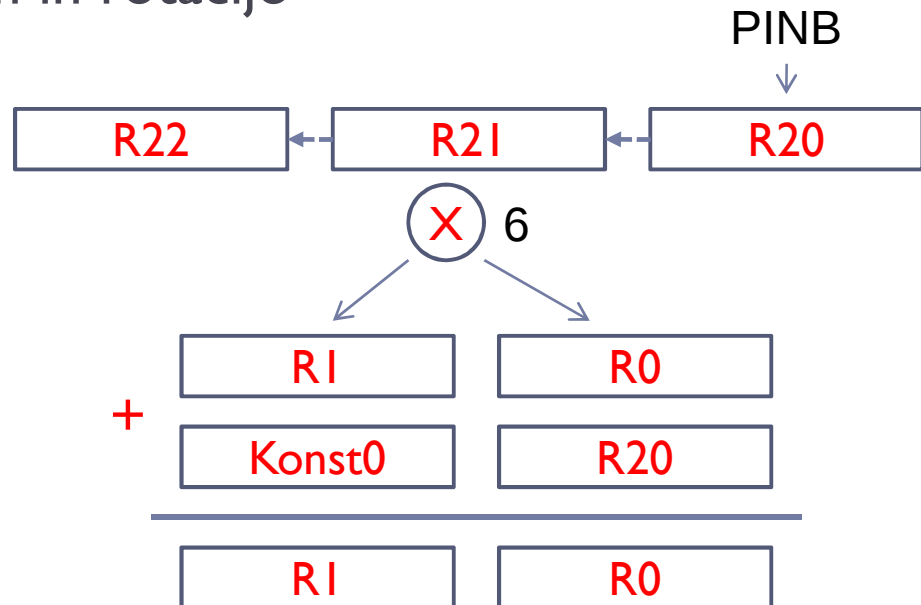
Izvedba konvolucije s procesorjem

- ▶ 8-bitne vzorce preberemo iz PINB
- ▶ naredimo množenje s 6 in seštejemo 16-bitne vrednosti
- ▶ naredimo deljenje z 8 s pomikom in rotacijo

loop:

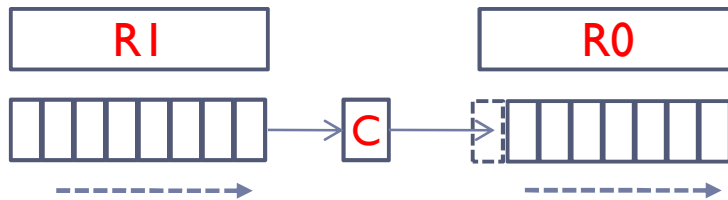
```
MOV r22, r21 ;premakni  
MOV r21, r20 ;premakni  
IN r20, PINB ;beri
```

```
MUL r21, Konst6 ; x6  
ADD r0, r20 ;vsota  
ADC r1, Konst0 ;prenos  
ADD r0, r22 ;vsota  
ADC r1, Konst0 ;prenos
```



Izvedba konvolucije s procesorjem 2

- ▶ naredimo deljenje z 8 s pomikom in rotacijo
- ▶ pošljemo rezultat na PORTD



```
LSR r1    ; 3x pomik  
ROR r0    ; in rotacija  
LSR r1  
ROR r0  
LSR r1  
ROR r0  
  
OUT PORTD, r0 ; izhod  
JMP loop
```

Delovanje procesorja

- ▶ Za izračun enega rezultata potrebujemo 20 urnih ciklov
- ▶ Kako vemo, kdaj je na voljo podatek (rezultat) ?
- 1. Naredimo usklajevalni protokol z uporabo ostalih priključkov
 - ▶ potrebnih je še več ciklov
- 2. Napišemo zanko v prekinitveni rutini
 - ▶ procesor ob zunanem prekinitvenem signalu skoči na rutino in napravi en računski cikel
 - ▶ komunikacijske enote izvedejo prekinitev, ko dobijo nov podatek