LABORATORIJSKE VAJE, 2. letnik - **03 -** Mikroprocesorji in mikrokontrolerji

**Arhitektura računalnikov  
 ATMEL AVR STUDIO**

**SKOKI, PSEVDOUKAZI, PRIMERJANJE, NASLAVLJANJE**

# Naloge

Naloge se nanašajo na delo s pomnilnikom.

## Naloga 1

Opiši, kaj delata ukaza LDS in STS! Način naslavljanja, kako naslovimo določen byte v pomnilniku, se imenuje neposredno ali direktno naslavljanje (== med ukazom in naslovom ni posrednika).

Lds – load from dataspace

Sts – store to dataspace

Napišite primer obeh ukazov in komentar.

Ldi r16, 0x11

Sts 0x100, r16

Lds r16, 0x100

## Naloga 2

Z ukazom STS postavite podatkovne pomnilniške celice od 0x0100 do 0x010A na vrednost 0x20.

## start:

## ldi r16, 0x20

## sts 0x0100, r16

## sts 0x0101, r16

## sts 0x0102, r16

## sts 0x0103, r16

## sts 0x0104, r16

## sts 0x0105, r16

## sts 0x0106, r16

## sts 0x0107, r16

## sts 0x0108, r16

## sts 0x0109, r16

## sts 0x010A, r16

## forever:

## rjmp forever

## Naloga 3

Pozorno preberite besedilo in postavite vprašanja, če kaj ni jasno.

Rešitev prejšnje naloge ni ravno elegantna. Kaj bi šele bilo, če bi morali postopek opraviti na 100 ali več zlogih pomnilnika. Rešitev ponuja indeksno naslavljanje ali naslavljanje s kazalci.

Za indeksno naslavljanje potrebujemo kazalec, ki vsebuje naslov v pomnilniku. Ker je podatkovni naslovni prostor večji od 256 byte-ov, za naslov uporabljamo registrske pare (po dva registra, en vsebuje zgornji del naslova (bite od 8 do 15), en pa spodnji del (bite od 0 do 7)). Skupaj tvorita naslov, ki ga uporabljamo v ukazih za delo s programskim ali podatkovnim pomnilnikom.

AVR pozna tri registrske pare:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 16-bitni  register | je isto kot  registrski par |  | alternativna imena registrov | |
| X | r27:r26 |  | XH | XL |
| Y | r29:r28 |  | YH | YL |
| Z | r31:r30 |  | ZH | ZL |

**NE POZABITE: register XH je v resnici r27, XL je r26 itd.**Registre r26 do r31 lahko uporabljamo kot običajne registre, lahko pa jih uporabljamo za naslove.

Preden nadaljujete, vam mora biti zgornje besedilo jasno.

start:

ldi r16, 0x20

ldi XL, 0x00

ldi XH, 0x01

ldi r17, 0x0B

loop:

st X+, r16

dec r17

brne loop

forever:

rjmp forever

## Naloga 4

Za branje in pisanje z indeksnim naslavljanjem podatkovnega pomnilnika AVR pozna dva ukaza

LD in ST. Ukaza potrebujeta še dva operanda in sicer najprej kam in nato še od kod.

ld r16, X; prekopiraj v register 16 vsebino pomnilnika z lokacije, shranjene v registru X  
st Z, r16; v celico podatkovnega pomnilnika z naslovom Z prekopiraj vrednost registra r16

1. Preizkusi spodnji primer. Program naj bi popisal nekaj pomnilnika z neko vrednostjo.
2. Komentirajte vrstice programa tako, da bodo komentarji odražali širšo sliko (gozd in ne dreves)!
3. Med razhroščevanjem opazujte segment IDATA na naslovu 0x0100. Kaj ste videli?
4. Kaj manjka, da bi program to storil na prvih 16 zlogih?
5. Dopolnite program in ga preizkusite.

.org 0x0000 ; prvi naslednji ukaz bo na naslovu 0000

rjmp start ; skoči na mesto start ( = 0050 )

.org 0x0050

start:

ldi ZH, 0x01 ; v register Z naložimo vrednost 0100

ldi ZL, 0x00 ; kazalnik Z postavimo na 0x100

ldi r16, 0xFF ; naložimo številko, ki jo želimo dajati v ram

ldi r17, 0x10 ; naložimo številko, kolikokrat želimo to ponoviti

loop:

st Z+, r16 ; v Z naložimo r16, in Z zvišamo za 1

dec r17 ; zmanjšajo število ponovitev

brne loop ; ponovno skoči na naslov loop, če je r17 postal 0

forever:

rjmp forever ; ponavlja zanko v neskončnost

## Naloga 5

Problem iz prejšnje naloge z lahko rešimo s samodejnim povečevanjem. Preglejte spodnje primere in komentarje ter ustrezno spremenite zgornji program, tako da bo deloval, kot je mišljeno.

ld r16, X+; prekopiraj v register 16 vsebino pomnilnika z lokacije, shranjene v registru X  
 ; nato povečaj X za 1 (temu se reče **post-increment**, zakaj?)  
  
st Z+, r16; v celico podatkovnega pomnilnika z naslovom Z prekopiraj vrednost registra r16  
 ; nato povečaj Z za 1  
  
ld r16, -X; najprej zmanjšaj X za 1, nato pa   
 ; prekopiraj v register 16 vsebino pomnilnika z lokacije, shranjene v registru X  
 ; prejšnji primer je bil post-increment, kako se pa reče temu tu?  
 ; temu se reče **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  
  
st -Z, r16; najprej Z zmanjšaj za 1, nato pa  
 ; v celico podatkovnega pomnilnika z naslovom Z prekopiraj vrednost registra r16

## Naloga 6

Združimo, kar smo se naučili sedaj.

S programom na naslove od 0x0120 do 0x0150 vpišite števila od 0x00 do 0x2F ter jih nato seštejte.  
Potrebujete znanja: seštevanja s prenosom, pisanje v pomnilnik, branje z pomnilnika.

Potrebujemo še enega od dveh ukazov:

brcc kam ;Branch if carry clear skoči, če je bil rezultat zastavice C v prejšnji operaciji enak 0  
brcs kam ;Branch if carry set skoči, če je bil rezultat zastavice C v prejšnji operaciji enak 1

Namig (če je potreben): Najprej podobno kot zgoraj vpišite v pomnilnik omenjene vrednosti. Nato berite vrednosti iz pomnilnika in jih prištevajte k izbranemu registru. Če pride do prenosa, povečajte en register zraven. Rezultat je 0x0468.

### Naslednjič: Podatki v programskem pomnilniku

Primer

;

; AssemblerApplication1.asm

;

; Created: 5.12.2016 13:21:21

; Author : uporabnik

;

; Replace with your application code

.org 0x0000

rjmp start

start:

ldi zl, low(hello\*2)

ldi zh, high(hello\*2)

ldi xl, low(v\_ramu)

ldi xh, high(v\_ramu)

loop:

lpm r16, Z+

st X+, r16

and r16, r16

brne loop

forever:

rjmp forever

datastuff:

hello: .db "Hello world!",0

.dseg

v\_ramu: .byte 20