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

**Arhitektura računalnikov  
 ATMEL AVR STUDIO**

**ARITMETIČNO LOGIČNE OPERACIJE,   
SKOKI, PSEVDOUKAZI**

# Naloge

Za izvedbo naloge boste potrebovali program Lightshot ali podoben program za zajemanje slike ekrana. Odprite program AVR Studio in ustvarite projekt v zbirniku. Odgovorite na vprašanja in odgovore označite krepko (bold).

## Naloga 1

Spomnite se, kako izgleda podatkovni naslovni prostor v procesorju ATMEGA 328p.   
Izvajajte spodnji program korak za korakom. Kaj opazite? **Da shrani 40 v r8 in 50 v r9 z ukazom STS.**

## 

Kakšna je razlika med ukazoma **lds r0, 0x0010** in **mov r0, r16**? Odgovor se skriva v dokumentaciji!

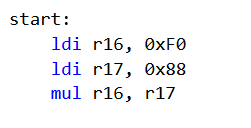
**Razlika je, da lds prebere vrednost iz pomnilnika, mov pa samo kopira vrednost med registri.**

Napiši ekvivalenta ukazov (ukaza, ki naredita isto):  
 sts 0x0008, r16 **mov r8, r16**  
 sts 0x0009, r17 **mov r9, r17**

## Naloga 2

V binarni obliki izračunajte na list papirja, koliko znaša 0x0F0 \* 0x88 ! Rezultat (v hex.): **0x67D0**

Poglej dokumentacijo in opiši, kako deluje ukaz MUL! **Izvaja množenje 8-bitnih števil z 8-bitnimi števili in rezultat je 16-bitno število.**



Izvedi program ukaz za ukazom! Primerjaj stanje registrov z rezultatom množenja zgoraj!   
Pojasni sosedu, kaj se je zgodilo!

Kolikšna je velikost faktorjev (v bitih)? **8 bitov** Kolikšna je velikost rezultata v bitih? **64 bitov**  
Kje in kako vidite rezultat operacije množenja?

## Naloga 3

Preberi spodnje besedilo in na podlagi rezultata zgoraj opredeli, ali ATMEGA 328p uporablja big ali little endian način zapisa števil.

In big-endian format, whenever addressing memory or sending/storing words bytewise, the most significant byte — the byte containing the [most significant bit](https://en.wikipedia.org/wiki/Most_significant_bit) — is stored first (has the lowest address) or sent first, then the following bytes are stored or sent in decreasing significance order, with the least significant byte — the one containing the [least significant bit](https://en.wikipedia.org/wiki/Least_significant_bit) — stored last (having the highest address) or sent last.

Little-endian format reverses this order: the sequence addresses/sends/stores the least significant byte first (lowest address) and the most significant byte last (highest address). Most computer systems prefer a single format for all its data; using the system's native format is automatic. But when reading memory or receiving transmitted data from a different computer system, it is often required to process and translate data between the preferred native endianness format to the opposite format.

Uporablja **little endian** način zapisa.

## Naloga 4

Vnesite naslednji program in ga izvajajte korak za korakom, vse dokler ne pridete na začetek.   
Ob vsakem koraku zapišite vsebino registra PC in koliko ciklov se je izvajal dotični ukaz.   
Odgovorite na vprašanja.

|  |  |  |
| --- | --- | --- |
| PC program counter | Trajanje v ciklih | Program |
| 0x00000000 | 0 |  |
| 0x00000050 | 1 |
|  |  |
| 0x00000050 | 1 |
| 0x00000050 | 1 |
| 0x00000051 | 1 |
| 0x00000052 | 1 |
| 0x00000053 | 2 |
|  |  |
| 0x00000053 | 1 |
| 0x00000054 | 1 |
| 0x00000056 | 2 |
| 0x00000058 | 2 |
| 0x00000040 | 2 |
|  |  |
| 0x00000040 | 0 |
| 0x00000040 | 0 |
| 0x00000041 | 1 |
| 0x00000042 | 1 |
| 0x000000443 | 2 |
|  |  |
| 0x00000043 | 0 |
| 0x00000050 | 2 |

### Vprašanja

|  |  |
| --- | --- |
| Koliko je 0x03 << 3? | 24 |
| Koliko je 0x01 << 3? | 28 |
| Katero operacijo pomeni oznaka << ? | **Levo premikajoči operator (byte shift)** |
| Se rezultat izraza izračuna med delovanjem programa ali pri prevajanju v strojno kodo? | Med delovanjem programa |
| Čemu služita ukaza rjmp in jmp? | Preskoku na določene dele programa |
| .ORG je t.i. psevdo ukaz. Kaj to pomeni? | To je navodilo za prevajalnik. |
| Čemu služi .ORG? | Uporablja se za določitev začetne naslovne lokacije, na katero naj se koda naloži ali prevede v pomnilniku. |
| Če bi oznake nadomestili z vrednostmi PC, kakšne bi bile vrednosti za |  |
| start | 0x1000 |
| cont1 |  |
| cont2 |  |
| cont3 |  |

## Naloga 5

Ukaza LDS (Load from Data Space) in STS (Store to Data Space) sta namenjena branju vrednosti iz podatkovnega naslovnega prostora oz. pisanju vanj. Ukaz BRNE ima enako vlogo, kot ukaz RJMP, le da se skok zgodi le, če je zastavica Z = 0.

Vnesite in preizkusite primer spodaj. Ugotovite, kaj program dela.

|  |  |
| --- | --- |
| .org 0x0000 | Prevajalnik gre na ničlo pozicijo |
| rjmp start | Skoči na start |
|  |  |
| .org 0x0050 | Prevajalnik gre na 0x0050 pozicijo |
| start: |  |
| ldi r16,0x05 | V register 16 naloži 0x05 |
| sts 0x0100, r16 | zapiše vsebino registra r16 v pomnilniško lokacijo 0x0100 v pomnilniku |
|  |  |
| loop: |  |
| lds r16, 0x0100 | V register 16 naloži vsebino iz pomnilniške lokacije 0x0100 |
| dec r16 | Zmanjša register 16 za ena |
| sts 0x100, r16 | zapiše vsebino registra r16 v pomnilniško lokacijo 0x0100 v pomnilniku |
|  |  |
| brne loop | Gre na loop če ni enako nič |
|  |  |
| forever: | Loop |
| rjmp forever | loop |

## Naloga 6 (ponovitev in dodatek):

Napišite program, ki bo seštel dve 16 bitni števili; 0x1234 in 0x9999 in rezultat spravil v RAM na naslov 0x0100 in 0x0101. Preverite rezultat s kalkulatorjem in v razhroščevalniku.

.org 0x0000 ; Začetni naslov

ldi r26, 0x12 ; Naložimo 0x1234 v r26 (nizki bajt)

ldi r27, 0x34 ; Naložimo 0x1234 v r27 (visoki bajt)

ldi r30, 0x99 ; Naložimo 0x9999 v r30 (nizki bajt)

ldi r31, 0x99 ; Naložimo 0x9999 v r31 (visoki bajt)

add r26, r30 ; Seštejemo nizke bajte

adc r27, r31 ; Seštejemo visoke bajte

sts 0x0100, r26 ; Shranimo nizke bajte rezultata na 0x0100

sts 0x0101, r27 ; Shranimo visoke bajte rezultata na 0x0101

done:

rjmp done ; Neskončna zanka

## Naloga 7

Z zanko, podobno nalogi 5, naredite del programa, ki bo porabil natanko 10 ciklov (pri uri 1 Mhz je to natanko 10 μs).

ldi r16, 4

zanka:

dec r16

brne zanka

konc:

rjmp konc