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

**Arhitektura računalnikov  
 ATMEL AVR STUDIO**

**SKOKI, PSEVDOUKAZI, PRIMERJANJE, NASLAVLJANJE - drugič**

# Naloge

## Naloga 1

Ponovi vlogo registrov X, Y in Z, kako so sestavljeni in kako se uporabljajo.

## Naloga 2

Preberi in stestiraj instrukcije CP in CPI v Atmel Studiu. Zapomni si rdeče zapisane stavke.

Instrukcije CP, CPC in CPI se uporabljajo za primerjanje registrov in števil. Delovanje instrukcij je podobno odštevanju, s tem da se razlika uporabi samo za določanje stanja zastavic.

Primer: cp r0, r1

Če r0 in r1 obravnavamo kot pozitivni celi (**=unsigned**, nepredznačeni) števili, potem gledamo samo zastavici Z in C ter velja:

* če sta enaki, je razlika enaka 0 - zastavica Z se postavi na 1, C pa na 0.
* če je prvo število večje, potem je razlika večje od 0 - zastavici Z in C se postavita na 0.
* če je večje drugo število, potem je razlika manjša od 0 - zastavici Z=0, C pa 1.
* v teh primerih za testiranje uporabljamo **BREQ, BRNE, BRSH (=BRCC) , BRLO (=BRCS)**

Če pa r0 in r1 gledamo v dvojiškem komplementu, potem gledamo zastavice Z, N in V:

* če sta enaki, je razlika 0, sta števili enaki - zastavica Z se postavi na 1.
* če je njuna razlika pozitivna (N=0) in ni prišlo do dvojiškega preliva (V=0),   
  potem je r0 ≥ r1
* če je njuna razlika negativna (N=1) in je obenem prišlo do dvojiškega preliva (V=1),   
  potem je ravno tako r0 ≥ r1.
* v nasprotnih primerih (N = 0, V = 1 ali N = 1, V = 0), je r0 ≤ r1.
* v teh primerih za testiranje uporabljamo **BREQ, BRNE, BRLT , BRGE**

Naredite po dva primera za nepredznačena in za predznačena števila ter interpretirajte zastavice (kako so bile postavljene in zakaj).

## Naloga 3

### Podatki v programskem pomnilniku

Podatkovni pomnilnik je tipa RAM in je pozabljiv – vsebina se izbriše takoj, ko zmanjka elektrike. Zato vanj ne moremo shraniti vsega besedila, ki bi ga uporabniku želeli izpisati na ekran.

Prekopirajte spodnji primer in ga preizkusite. Primer prekopira podatke v interni RAM (IRAM). Dopolnite komentarje.

Primer:

; program prekopira podatke iz programskega v podatkovni pomnilnik.

; v programskem pomnilniku imamo niz »Hello world!« s končno ASCII 0

; takemu nizu pravimo niz z zaključno ničlo (null-terminated string)

; pri čemer se pod null-terminated razume en byte z vrednostjo 0.

.cseg **; naslednje vrstice sodijo v programski pomnilnik**

.org 0x0000

rjmp start

start:

ldi zl, low(hello\*2) ; hello je naslov v programskem pomnilniku,

ldi zh, high(hello\*2) ; tam je na vsakem naslovu 16-bitna vrednost

; zato programski pomnilnik pri podatkih obravnavamo kot

; da je 8-bitni – pomnožimo naslov z 2 in nato pomnilnik

; naslavljamo, kot običajni 8-bitni pomnilnik.

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 ; .dseg pove, da se naslednje vrstice nanašajo na

; podatkovni pomnilnik

v\_ramu: .byte 20 ; rezervirali smo 20 byte-ov za naš niz

#### Opomba:

Če niz na oznaki hello nima sodega števila zlogov, ob prevajanju vidimo naslednje opozorilo, ki pove, da je prevajalnik sam dopolnil niz z dodatnim byteom z vrednostjo 0:



To opozorilo lahko ignoriramo, če nam je jasno, zakaj je tam. Lahko pa dodamo še kakšno 0 sami.

### Vprašanja:

Kolikšna je vrednost oznake hello? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Kolikšna je vrednost oznake v ramu? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Naloga 4

Ukaz CALL povzroči, da se program nadaljuje na nekem drugem naslovu, vse dokler ne naleti na ukaz RET, ki povzroži, da se vrne nazaj na mesto naslednjega ukaza. Govorimo o podprogramu pa tudi funkciji, če ta podprogram nekako vrača neko vrednost.

Spodnji program vrača ASCII kodo šestnajstiške števke.

Preizkusite ga:

.cseg

.org 0000

rjmp reset ; ob resetu (zagonu) skočimo na naslov reset

.org 0x0034

reset:

ldi r16,0x0C ; v r16 preizkusno naložimo vrednost 12 '0C'

call getHexAscii ; po klicu v r16 pričakujemo vrednost 0x43 ('C')

forever:

rjmp forever

; podprogram za vrednost iz r16 poišče ustrezen ASCII znak

getHexAscii:

andi r16, 0x0F ; zgornje bite pobrišemo, klicatelj naj poskrbi za večja števila sam

ldi zh, high (hexASC \* 2) ; vsak naslov v programskem pomnilniku ima 2 byte-a

ldi zl, low (hexASC \* 2) ; Z zdaj kaže na tabelo znakov 0123456789ABCDEF

add zl, r16 ; če je R16 enak 0, potem dobimo '0', če je 15, dobimo 'F'

lpm r16, Z ; gremo po vrednost v programski pomnilnik

ret ; se vrnemo nazaj, od koder je prišel CALL

HexASC: .db "0123456789ABCDEF",0 ; znaki za šestnajstiški zapis

Dopolnite program da bo klical podprogram getHexAscii še za vrednosti 2, A in F – vse v istem programu.

## Naloga 5

Popravite zgornji program tako, da bo za vrednost v r16 poiskal obe šestnajstiški števki. (r16 = 0xEF – rezultat sta dva klica, en za 0E in drugi za 0F). Uporabite deljenje ali ukaz za menjavo *nibblov*.

## Naloga 6

Popravite program iz naloge 3 tako, da bo po kopiranju klical podprogram Upper. Ta podprogram naj pregleduje zloge v IRAMu od naslova X dalje, dokler ne naleti na vrednost 0 ter vsako črko besedila pretvori v veliko črko. Spomnite se lastnosti ASCII tabele.

Dodatek: najprej naj postopek dela na vseh vrednostih. Ko bo to delalo, poskrbite še, da se bo to dogajalo le na malih ASCII črkah (zagotovite, da bodo ASCII vrednosti v tem razponu).

## Naloga 7 – ponovitev

Napišite podprogram, ki bo čakal 160 ciklov in bo uporaben za pavzo. 160 ciklov je pri 16Mhz 10us.

Napišite še en podprogram, ki bo čakal milisekundo pri frekvenci 16 MHz.