# ZBIRNIK ATMEL AVR

V0.91b

Gradivo za Računalništvo in Laboratorijske vaje za 2. letnik tehniške gimnazije

# Assembler source Izvorna koda v zbirniku

The Assembler works on source files containing instruction mnemonics, labels and directives. The instruction mnemonics and the directives often take operands.Code lines should be limited to 120 characters.

Za zbirnik uporabljamo izvorne datoteke, ki vsebujejo mnemonike instrukcij (okrajšave, s pomočjo katerih si zapomnimo pomen ukaza, v nadaljevanju bomo temu rekli kar instrukcija ali ukaz), oznake in direktive.

Every input line can be preceded by a label, which is an alphanumeric string terminated by a colon. Labels are used as targets for jump and branch instructions and as variable names in Program memory and RAM.

Na začetku vsake vrstice je lahko oznaka (labela), niz alfanumeričnih znakov (črk in številk), ki jim sledi dvopičje. Z oznakami zaznamujemo mesto v izvorni kodi, na katerega se lahko sklicujemo pri pogojnih in brezpogojnih skokih ter imenih spremenljivk v programskem in podatkovnem pomnilniku.

In input line may take one of the four following forms:

[label:] instruction [operands] [Comment][label:] directive [operands] [Comment]

Comment

(Empty line)

V vhodni datoteki ima vsaka vrstica eno izmed naslednjih oblik:

[oznaka:] instrukcija [operandi] [komentar][oznaka:] direktiva [operandi] [komentar]

komentar

(prazna vrstica)

A comment has the following form:

; [Text]

Komentar ima naslednjo obliko

; [Tu pride besedilo komentarja]

Items placed in braces are optional. The text between the comment-delimiter (;) and the end of line (EOL) is ignored by the Assembler. Labels, instructions and directives are described in more detail later. See also [AVR Assembler Syntax](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_Syntax.html).

Stvari v oglatih oklepajih so neobvezne. Oklepajev okoli njih ne pišemo. Če ima instrukcija npr. le vsebovane operande, potem lahko vrstico zaključimo. Prav tako ne pišemo oznak ali komentarjev, če niso nujno potrebne. Oznake, instrukcije in direktive bomo spoznali v nadaljevanju.

## Example / primer:

label: .EQU var1=100 ; Set var1 to 100 (Directive)

.EQU var2=200 ; Set var2 to 200

test: rjmp test ; Infinite loop (Instruction)

; Pure comment line

; Another comment line

oznaka: .EQU var1=100 ; Nastavi spremenljivko var1 to na 100 (direktiva)

.EQU var2=200 ; Nastavi spremenljivko var2 to na 200

test: rjmp test ; neskončna zanka (instrukcija)

; vrstica, ki vsebuje zgolj komentar

; še ena vrstica s komentarjem

# Direktive

Izmed vseh direktiv bomo obravnavali zgolj direktive, ki imajo za nas večjo uporabno vrednost in nas obenem ne bodo »zamorile« pri spoznavanju zbirnika.

Vsaka direktiva se začne s piko (.) in imenom direktive, ki je lahko zapisana z velikimi ali malimi črkami. Zaželeno je, da se uporabljajo velike črke.

## BYTE - reserve bytes for a variable BYTE – rezervira pomnilnik za spremenljivko

The BYTE directive reserves memory resources in the SRAM or EEPROM. In order to be able to refer to the reserved location, the BYTE directive should be preceded by a label. The directive takes one parameter, which is the number of bytes to reserve. The directive can not be used within a Code segment (see directives ESEG, CSEG, DSEG) .Note that a parameter must be given. The allocated bytes are not initialized.

Direktiva BYTE rezervira dano število zlogov (bytov) SRAM pomnilnika ali EEPROMa. Direktiva mora biti označena z oznako (label), da rezerviran pomnilnik lahko uporabljamo v programu. Direktiva ima en parameter, ki določi, koliko zlogov pomnilnika rezerviramo. Upoštevati je treba, da tako določen ni inicializiran (torej ob zagonu programa v njem ni smiselnih podatkov in jih mora program vrednosti še posebej nastaviti).

Direktiva se od direktive DB razlikuje po tem, da samo rezervira dano število zlogov. .BYTE 30 pomeni, da smo rezervirali 30 zlogov pomnilnika, medtem ko direktiva .DB 30 pomeni, da bo na danem naslovu en zlog z vrednostjo 30.

### **Syntax / sintaksa:**

LABEL: .BYTE expression

### **Example / Primer:**

.DSEG

var1: .BYTE 1 ; reserve 1 byte to var1

table: .BYTE tab\_size ; reserve tab\_size bytes

.CSEG

ldi r30,low(var1) ; Load Z register low

ldi r31,high(var1) ; Load Z register high

ld r1,Z ; Load VAR1 into register 1

## CSEG - Code segment

The CSEG directive defines the start of a Code Segment. An Assembler file can consist of several Code Segments, which are concatenated into one Code Segment when assembled. The BYTE directive can not be used within a Code Segment. The default segment type is Code. The Code Segments have their own location counter which is a word counter. The ORG directive can be used to place code and constants at specific locations in the Program memory. The directive does not take any parameters.

**Direktiva CSEG definira, kje se začne blok z delom programa (CODE) oz. programski blok. Program v zbirniku ima lahko več delov programa raztresenih po eni ali več datotekah, ki se pred prevajanjem v strojno kodo zberejo skupaj in prevedejo kot en programski blok (zato ime »zbirnik«). Znotraj CSEG bloka ne moremo uporabljati določenih direktiv, kot je npr. BYTE. Kadar ne navedemo tipa segmenta, se zibrnik obnaša kot da gre za programski blok. Programski blok ima tudi svoj števec dolžine ene besede, torej 16 bitov. Če želimo postaviti ukaze ali podatke na točno mesto znotraj programskega pomnilnika, uporabimo direktivo ORG. Direktiva CSEG nima parametrov.**

### **Syntax / sintaksa:**

.CSEG

### **Example / Primer:**

.DSEG ; Start data segment

vartab: .BYTE 4 ; Reserve 4 bytes in SRAM

.CSEG ; Start code segment

const: .DW 2 ; Write 0x0002 in prog.mem.

mov r1,r0 ; Do something

## DB – Define constant byte(s) in program memory and EEPROM DB – Definiraj konstante kot zloge v programskem pomnilniku ali EEPROMu

The DB directive reserves memory resources in the program memory or the EEPROM memory. In order to be able to refer to the reserved locations, the DB directive should be preceded by a label. The DB directive takes a list of expressions, and must contain at least one expression. The DB directive must be placed in a Code Segment or an EEPROM Segment.

Direktiva DB rezervira prostor velikosti 1 zloga (1 byte, 8 bits) in definira vire podatkov v programskem pomnilniku ali EEPROMu. Da bi do teh podatkov lažje dostopali (oz. se na njih sklicevali), ponavadi pred direktivo DB postavimo oznako (labelo). Direktivi sledi spisek, v katerem so lahko števila, besedilo ali računski izrazi. Spisek mora imeti vsaj eno vrednost. DB lahko uporabljamo samo v programskem (CSEG) ali EEPROM segmentu (ESEG).

The expression list is a sequence of expressions, delimited by commas. Each expression must evaluate to a number between -128 and 255. If the expression evaluates to a negative number, the 8 bits twos complement of the number will be placed in the program memory or EEPROM memory location.

Spisek je zaporedje izrazov, ki so med seboj ločeni z vejico. Rezultat vsakega izraza mora biti možno zapisati kot število od -128 do 255. Če gre za negativno vrednost, se ta zapiše v dvojiškem komplementu.

Včasih podatkov ne moremo imeti kar v RAM, ker ta pomnilnik izgubi vsebino ob izklopu napetosti. Če bi imeli niz »Hello, World!« shranjen v podatkovnem pomnilniku, ob vklopu tam ne bi bilo ničesar uporabnega, ker je RAM ob izklopu pozabil vsebino. Zato pogosto shranjujemo določene podatke (ki jih seveda v programu potem ne moremo spreminjati) kar v programskem pomnilniku. Pogosto shranjujemo nize ali neke konstante). DB ne moremo uporabljati v podatkovnem (DSEG) segmentu, ker v RAM pač ne moremo trajno shraniti vrednosti.

If the DB directive is given in a Code Segment and the expressionlist contains more than one expression, the expressions are packed so that two bytes are placed in each program memory word. If the expressionlist contains an odd number of expressions, the last expression will be placed in a program memory word of its own, even if the next line in the assemby code contains a DB directive. The unused half of the program word is set to zero. A warning is given, in order to notify the user that an extra zero byte is added to the .DB statement.

Ker so spominske celice programskega pomnilnika organizirane po dva zloga (bytea) skupaj, so tudi vrednosti s spisak pakirane po 2 skupaj. Če je število zlogov liho, se zadnji zlog shrani v nižji del nove celice, višji pa dobi vrednost 0. Prevajalnik v tem primeru generira opozorilo. To se zgodi tudi v primeru, da se izvorna koda nadaljuje z novo DB direktivo.

**Včasih podatkov ne moremo imeti kar v RAM, ker ta pomnilnik izgubi vsebino ob izklopu napetosti.** Če bi imeli niz »Hello, World!« shranjen v podatkovnem pomnilniku, ob vklopu tam ne bi bilo ničesar uporabnega, ker je RAM ob izklopu pozabil vsebino. **Zato pogosto shranjujemo** določene **podatke** (ki jih seveda v programu potem ne moremo spreminjati) **kar v programskem pomnilniku**. Pogosto shranjujemo nize ali neke konstante). **DB ne moremo uporabljati v podatkovnem (DSEG) segmentu**, ker v **RAM** pač ne moremo trajno shraniti vrednosti.

### **Syntax / sintaksa:**

LABEL: .DB expressionlist

### Example / Primer:

.CSEG

consts: .DB 0, 255, 0b01010101, -128, 0xaa

.ESEG

const2: .DB 1,2,3

## DSEG - Data Segment

The DSEG directive defines the start of a Data segment. An assembler source file can consist of several data segments, which are concatenated into a single data segment when assembled. A data segment will normally only consist of BYTE directives (and labels). The Data Segments have their own location counter which is a byte counter. The ORG directive can be used to place the variables at specific locations in the SRAM. The directive does not take any parameters.

**Direktiva označuje pričetek podatkovnega bloka. Podobno kot pri programskem bloku je tudi tu podatkovni blok sestavljen iz večih delov v eni ali več datotekah. Pred prevajanjem se ti bloki zberejo v enoten podatkovni blok. Običajno podatkovni blok sestavljajo direktive BYTE, ki jim sledi število zlogov tako rezerviranega pomnilnika.**

### **Syntax / sintaksa:**

.CSEG

### **Example / Primer:**

.DSEG ; Start data segment / od tu dalje je podatkovni segment

var1: .BYTE 1 ; reserve 1 byte to var1 / rezerviraj 1 zlog za var1

table: .BYTE tab\_size ; reserve tab\_size bytes / rezerviraj tab\_size  
zlogov

.CSEG

ldi r30,low(var1) ; Load Z register low

ldi r31,high(var1) ; Load Z register high

ld r1,Z ; Load var1 into register 1

## DW – Define constant word(s) in program memory and EEPROM DW – Definiraj konstante kot besede v progr. pomnilniku ali EEPROMu

The DW directive reserves memory resources in the program memory or the EEPROM memory. In order to be able to refer to the reserved locations, the DW directive should be preceded by a label. The DW directive takes a list of expressions, and must contain at least one expression. The DB directive must be placed in a Code Segment or an EEPROM Segment.

The expression list is a sequence of expressions, delimited by commas. Each expression must evaluate to a number between -32768 and 65535. If the expression evaluates to a negative number, the 16 bits two's complement of the number will be placed in the program memory or EEPROM memory location.

Direktiva DW rezervira prostor 1 besede (1 word, 16 bits) in definira vire podatkov v programskem pomnilniku ali EEPROMu. Da bi do teh podatkov lažje dostopali (oz. se na njih sklicevali), ponavadi pred direktivo DB postavimo oznako (labelo). Direktivi sledi spisek, v katerem so lahko števila, besedilo ali računski izrazi. Spisek mora imeti vsaj eno vrednost. DB lahko uporabljamo samo v programskem (CSEG) ali EEPROM segmentu (ESEG).

Spisek je zaporedje izrazov, ki so med seboj ločeni z vejico. Rezultat vsakega izraza mora biti možno zapisati kot število od -32768 do 65535. Če gre za negativno vrednost, se ta zapiše v dvojiškem komplementu.

### **Syntax / Sintaksa:**

LABEL: .DB expressionlist

### Example / Primer:

.CSEG

varlist: .DW 0, 0xffff, 0b1001110001010101, -32768, 65535

.ESEG

eevarlst: .DW 0,0xffff,10

## DD - Define constant doubleword(s) in program memory and EEPROM DQ - Define constant quadword(s) in program memory and EEPROM

These directives are very similar to the DW directive, except they are used to define 32-bit (doubleword) and 64-bit (quadword) respectively. The data layout in memory is strictly little-endian.

Direktivi sta sorodni direktivama DB in DW, le da z njima lahko definiramo vrednosti velikosti 32 (doubleword) in 64 bitov (quadword).

### **Syntax / Sintaksa:**

LABEL: .DD expressionlist

LABEL: .DQ expressionlist

### Example / Primer:

.CSEG

varlist: .DD 0, 0xfadebabe, -2147483648, 1 << 30

.ESEG

eevarlst: .DQ 0,0xfadebabedeadbeef, 1 << 62

## EQU – Set a symbol equal to an expression EQU – Priredi simbolu vrednost danega izraza

The EQU directive assigns a value to a label. This label can then be used in later expressions. A label assigned to a value by the EQU directive is a constant and can not be changed or redefined.

Direktiva EQU priredi neki oznaki dano vrednost. To oznako lahko potem uporabljamo v vseh ostalih izrazih. Vrednost oznake potem ko smo ji priredili vrednost z EQU naknadno ne moremo več spreminjati ali jo ponovno definirati.

### **Syntax / Sintaksa:**

.EQU label = expression

### Example / Primer:

.EQU io\_offset = 0x23

.EQU porta = io\_offset + 2.CSEG ; Start code segment

clr r2 ; Clear register 2

out porta,r2 ; Write to Port A

## INCLUDE - Include another file

The INCLUDE directive tells the Assembler to start reading from a specified file. The Assembler then assembles the specified file until end of file (EOF) or an EXIT directive is encountered. An included file may itself contain INCLUDE directives.

Direktiva INCLUDE pove zbirniku, naj na tem mestu vključi vsebino dane datoteke. Zbirnik vključi v prevajanje vse njene vrstice do konca datoteke ali do direktive EXIT. Datoteka, ki jo tako vključujemo, lahko vključuje še druge datoteke.

### Syntax:

.INCLUDE "filename"

.INCLUDE <filename>

### Example:

; iodefs.asm:

.EQU sreg = 0x3f ; Status register

.EQU sphigh = 0x3e ; Stack pointer high

.EQU splow = 0x3d ; Stack pointer low

; incdemo.asm

**.INCLUDE "m328pdef.asm"** ; Include all definitions for ATmega328p

in r0,sreg ; Read status register

## ORG - Set program origin

The ORG directive sets the location counter to an absolute value. The value to set is given as a parameter. If an ORG directive is given within a Data Segment, then it is the SRAM location counter which is set, if the directive is given within a Code Segment, then it is the Program memory counter which is set and if the directive is given within an EEPROM Segment, it is the EEPROM location counter which is set.

Direktiva ORG postavi lokacijski števec na dano absolutno vrednost, ki jo navedemo kot parameter. Če se ORG nahaja znotraj podatkovnega segmenta, parameter nastavi lokacijski števec v SRAMu. Če je ORG znotraj programskega segmenta, potem se na to vrednost nastavi lokacijski števec programa (po domače: instrukcije se bodo odvijale od tega naslova dalje). Podobno velja za EEPROM segment in direktivo ESEG.

The default values of the Code and the EEPROM location counters are zero, and the default value of the SRAM location counter is the address immediately following the end of I/O address space (0x60 for devices without extended I/O, 0x100 or more for devices with extended I/O) when the assembling is started. Note that the SRAM and EEPROM location counters count bytes whereas the Program memory location counter counts words. Also note that some devices lack SRAM and/or EEPROM.

Začetne vrednosti lokacij programskega dela in dela v EEPROMu se začnejo pri 0, medtem ko pri podatkovnem delu lokacijski števec začenja na mestu takoj za vhodno/izhodnimi registri (0x60 za naprave brez dodatnih vhodno/izhodnih naprav ter 0x0100 ali več za ostale). Upoštevati je treba še, da lokacijska števca za SRAM in EEPROM štejeta zloge (byte), programski lokacijski števec pa besede (word = 2 byte). Prav tako je možno, da neka naprava nima ne statičnega rama in ne EEPROMa.

### Syntax:

.ORG expression

### Example:

.DSEG ; Start data segment

.ORG 0x120 ; Set SRAM address to hex 120

variable: .BYTE 1 ; Reserve a byte at SRAM adr. 0x120

.CSEG

.ORG 0x10 ; Set Program Counter to hex 10

mov r0,r1 ; Do something

# Ostale direktive

* DEF
* SET
* LIST
* NOLIST
* UNDEF
* MESSAGE
* WARNING
* MACRO
* ENDM, ENDMACRO
* IF, IFDEF, IFNDEF
* ELSEIF, ELIF

<http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_directives.html#avrassembler.wb_directives.BYTE>

## Status register Statusni register

|  |  |  |
| --- | --- | --- |
|  | Slovenski prevod | Angleški vir |
| C | Zastavica prenosa | Carry flag in status register |
| Z | Zastavica ničle | Zero flag in status register |
| N | Zastavica negativnosti | Negative flag in status register |
| V | Zastavica preliva pri dvojiškem komplementu | Two's complement overflow indicator |
| S | N [+] V, za testiranje števil v dv. kompl. | N [+] V, For signed tests |
| H | Zastavica prenosa polovičke | Half Carry flag in the status register |
| T | Prenosni bit T, ki ga uporabljata BLD in BST | Transfer bit used by BLD and BST instructions |
| I | Globalna zastavica za  o(ne)mogočanje prekinitev | Global interrupt enable/disable flag |

## Operands Operandi

The following operands can be used:

* User defined labels which are given the value of the location counter at the place they appear.
* User defined variables defined by the SET directive
* User defined constants defined by the EQU directive
* Integer constants: constants can be given in several formats, including
  + Decimal (default): 10, 255
  + Hexadecimal (two notations): 0x0a, $0a, 0xff, $ff
  + Binary: 0b00001010, 0b11111111
  + Octal (leading zero): 010, 077
  + PC - the current value of the Program memory location counter
  + Floating point constants

Uporabljajo se lahko naslednji operandi:

* Uporabniško določene oznake (labele), ki se jim dodeli vrednost (številka) lokacije, kjer se pojavijo v programu
* Spremenljivke, ki jih je uporabnik nastavil z direktivama SET ali EQU
* Celoštevilske spremenljivke: konstante v eni izmed oblik, kot npr.
  + Desetiška števila (privzeto): 10, 255
  + Šestnajstiška števila (dve možnosti): 0x0a, $0a, 0xff, $ff
  + Dvojiški zapis: 0b00001010, 0b11111111
  + Osmiški zapis (**obvezna** vodilna ničla): 010, 077
  + PC – trenutna vrednost programskega lokacijskega števca
  + Konstanta v obliki plavajoče vejice

## Operators Operatorji

<http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_expressions.html>

Operatorje lahko uporabljamo ko preračunavamo posamezne izraze, ki jih uporabljamo v direktivah ali instrukcijah pri izračunih konstant. Ne moremo jih uporabljati kot del samega programa.

|  |  |  |
| --- | --- | --- |
| Symbol | Opis | Description |
| ! | Logični NE | Logical not |
| ~ | Bitni NE | Bitwise Not |
| - | Predznak - | Unary Minus |
| \* | Množenje | Multiplication |
| / | Deljenje | Division |
| % | Ostanek pri deljenju | Modulo ( AVR Assembler 2 only) |
| + | Seštevanje | Addition |
| - | Odštevanje | Subtraction |
| << | Pomik levo | Shift left |
| >> | Pomik desno | Shift right |
| < | Manjše kot | Less than |
| <= | Manjše ali enako | Less than or equal |
| > | Večje kot | Greater than |
| >= | Večje ali enako | Greater than or equal |
| == | Enako | Equal |
| != | Ni enako | Not equal |
| & | Bitni IN | Bitwise And |
| ^ | Bitni ekskluzivni ALI | Bitwise Xor |
| | | Bitni ALI | Bitwise Or |
| && | Logični IN | Logical And |
| || | Logični ALI | Logical Or |
| ? | Pogojni operator | Conditional operator |

## Registri in operandi

|  |  |
| --- | --- |
| Oznaka | Pomen |
| Rd | Destination (and source) register in the register file |
| Ciljni (in na začetku izvorni) register v sklopu splošnih registrov |
| Rr | Source register in the register file |
| Izvorni register v sklopu splošnih registrov |
| R | Result after instruction is executed |
| Rezultat po zaključki operacije |
| K | Constant data |
| Konstanta – nespremenljivo število |
| K8 | 8-bit Constant data |
| 8-bitna konstanta |
| k | Constant address |
| Konstanten – nespremenljiv – naslov |
| b | Bit in the register file or I/O register (3 bit) |
| Bit v splošnem ali vhodno-izhodnem registru (po trije biti skupaj) |
| s | Bit in the status register (3 bit) |
| Bit v statusnem registru (po trije biti skupaj) |
| X,Y,Z | Indirect address register (X=R27:R26, Y=R29:R28 and Z=R31:R30) |
| Registri za posredno naslavljanje, sestavljeni kot pari registrov  (X=R27:R26, Y=R29:R28 in Z=R31:R30) |
| A | I/O location address |
| Naslov vhodno izhodnega registra |
| q | Displacement for direct addressing (6 bit) |
| odmik za neposredno naslavljanje |

## Functions Funkcije

|  |  |  |
| --- | --- | --- |
| Function / Funkcija | Result / rezultat | Example / Primer |
| LOW(expression) | returns the low byte of an expression | LOW(0x1234) = 0x34 |
| Vrne nižji zlog izraza |
| HIGH(expression) | returns the second byte of an expression | HIGH(0x1234) = 0x12 |
| Vrne višji zlog izraza |
| BYTE2(expression) | is the same function as HIGH | BYTE2(0x1234) = 0x12 |
| Ista funkcija kot HIGH |
| BYTE3(expression) | returns the third byte of an expression | BYTE3(0x10203040) = 0x20 |
| Vrne tretji zlog izraza |
| BYTE4(expression) | returns the fourth byte of an expression | BYTE4(0x10203040) = 0x10 |
| Vrne 4 zlog izraza |
| LWRD(expression) | returns bits 0-15 of an expression | LWRD(0x10203040) = 0x3040 |
| iz izraza vrne spodnjo besedo (word) – bite 0 do 15 |
| HWRD(expression) | returns bits 16-31 of an expression | HWRD(0x10203040) = 0x1020 |
| iz izraza vrne zgornjo besedo (word) – bite 16 do 31 |
| PAGE(expression) | returns bits 16-21 of an expression | PAGE(0x12781278) = 0x38  0001 0010 01**11 1000** 0001 0010 0111 1000 |
| iz izraza vrne samo bite 16-21 |
| EXP2(expression) | returns 2 to the power of expression | EXP2(5) = 32 |
| vrne potenco števila 2 |
| LOG2(expression) | returns the integer part of log2(expression) | LOG2(10) = 3 |
| vrne celi del dvojiškega logaritma |
| INT(expression) | Truncates a floating point expression to integer (ie discards fractional part) | INT(3,1415) = 3 |
|  |
| FRAC(expression) | Extracts fractional part of a floating point expression (ie discards integer part). | FRAC(3,1415) = 0,1415 |
| Izlušči neceli del izraza v obliki plavajoče vejice (kot da bi stran vrgli celi del števila) |
| Q7(expression) | Converts a fractional floating point expression to a form suitable for the FMUL/FMULS/FMULSU instructions. (sign + 7-bit fraction) |  |
| Pretvori število v zapis primeren za uporabo z instrukcijami FMUL, FMULS (predznak in 7bitni ulomek) |
| Q15(expression) | Converts a fractional floating point expression to a form suitable for the FMUL/FMULS/FMULSU instructions. (sign +15-bit fraction) |  |
| Pretvori število v zapis primeren za uporabo z instrukcijami FMUL, FMULS (predznak in 15bitni ulomek) |
| ABS() | Returns the absolute value of a constant expression. | ABS(5-8) = 3 |
| Vrne absolutno vrednost izraza |
| DEFINED(symbol) | Returns true if symbolis previously defined using .equ/.set/.def directives. Normally used in conjunction with .if directives (.if defined(foo)), but may be used in any context. It differs from other functions in that parentheses around its argument are not required, and that it only makes sense to use a single symbol as argument. | st1: .EQU  .if DEFINED(st1) ... |
|  | Vrne true, če je bil simbol kje prej definiran s .def, .set ali .equ direktivo. Izraz je lahko samo simbol. |  |
| STRLEN(string) | returns the length of a string c+onstant, in bytes. | STRLEN("Hello!") = 6 |
| Vrne dolžino niza znakov v zlogih (bytih) |

## Arithmetic and logic instructions Aritmetične in logične operacije

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mnemonic | Oper. | Description | Operation | Flags | Cycles |
| [ADD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ADD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Add without Carry | Rd = Rd + Rr | Z,C,N,V,H,S | 1 |
| [ADC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ADC.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Add with Carry | Rd = Rd + Rr + C | Z,C,N,V,H,S | 1 |
| [ADIW](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ADIW.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Add Immediate To Word | Rd+1:Rd,K | Z,C,N,V,S | 2 |
| [SUB](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SUB.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Subtract without Carry | Rd = Rd - Rr | Z,C,N,V,H,S | 1 |
| [SUBI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SUBI.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Subtract Immediate | Rd = Rd - K8 | Z,C,N,V,H,S | 1 |
| [SBC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBC.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Subtract with Carry | Rd = Rd - Rr - C | Z,C,N,V,H,S | 1 |
| [SBCI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBCI.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Subtract with Carry Immedtiate | Rd = Rd - K8 - C | Z,C,N,V,H,S | 1 |
| [AND](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_AND.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Logical AND | Rd = Rd · Rr | Z,N,V,S | 1 |
| [ANDI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ANDI.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Logical AND with Immediate | Rd = Rd · K8 | Z,N,V,S | 1 |
| [OR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_OR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Logical OR | Rd = Rd V Rr | Z,N,V,S | 1 |
| [ORI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ORI.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Logical OR with Immediate | Rd = Rd V K8 | Z,N,V,S | 1 |
| [EOR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_EOR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Logical Exclusive OR | Rd = Rd EOR Rr | Z,N,V,S | 1 |
| [COM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_COM.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | One's Complement | Rd = $FF - Rd | Z,C,N,V,S | 1 |
| [NEG](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_NEG.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Two's Complement | Rd = $00 - Rd | Z,C,N,V,H,S | 1 |
| [CBR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CBR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Set Bit(s) in Register | Rd = Rd V K8 | Z,C,N,V,S | 1 |
| [CBR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CBR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Clear Bit(s) in Register | Rd = Rd · ($FF - K8) | Z,C,N,V,S | 1 |
| [INC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_INC.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Increment Register | Rd = Rd + 1 | Z,N,V,S | 1 |
| [DEC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_DEC.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Decrement Register | Rd = Rd -1 | Z,N,V,S | 1 |
| [TST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_TST.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Test for Zero or Negative | Rd = Rd · Rd | Z,C,N,V,S | 1 |
| [CLR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Clear Register | Rd = 0 | Z,N,V,S | 1 |
| [SER](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SER.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Set Register | Rd = $FF |  | 1 |
| [SBIW](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBIW.html) | [Rdl](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rdl),[K6](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K6) | Subtract Immediate from Word | Rdh:Rdl = Rdh:Rdl - K 6 | Z,C,N,V,S | 2 |
| [MUL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_MUL.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Multiply Unsigned | R1:R0 = Rd \* Rr | Z,C | 2 |
| [MULS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_MULS.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Multiply Signed | R1:R0 = Rd \* Rr | Z,C | 2 |
| [MULSU](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_MULSU.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Multiply Signed with Unsigned | R1:R0 = Rd \* Rr | Z,C | 2 |
| [FMUL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_FMUL.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Fractional Multiply Unsigned | R1:R0 = (Rd \* Rr) << 1 | Z,C | 2 |
| [FMULS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_FMULS.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Fractional Multiply Signed | R1:R0 = (Rd \*Rr) << 1 | Z,C | 2 |
| [FMULSU](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_FMULSU.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Fractional Multiply  Signed with Unsigned | R1:R0 = (Rd \* Rr) << 1 | Z,C | 2 |

## Branch Instructions Skoki in vejitve

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [RJMP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_RJMP.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Relative Jump | PC = PC + k +1 |  | 2 |
| [IJMP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_IJMP.html) |  | Indirect Jump to ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) | PC = Z |  | 2 |
| [EIJMP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_EIJMP.html) |  | Extended Indirect Jump ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) | STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND |  | 2 |
| [JMP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_JMP.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Jump | PC = k |  | 3 |
| [RCALL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_RCALL.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Relative Call Subroutine | STACK = PC+1, PC = PC + k + 1 |  | 3/4\* |
| [ICALL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ICALL.html) |  | Indirect Call to ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) | STACK = PC+1, PC = Z |  | 3/4\* |
| [EICALL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_EICALL.html) |  | Extended Indirect Call to ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) | STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND |  | 4\* |
| [CALL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CALL.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Call Subroutine | STACK = PC+2, PC = k |  | 4/5\* |
| [RET](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_RET.html) |  | Subroutine Return | PC = STACK |  | 4/5\* |
| [RETI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_RETI.html) |  | Interrupt Return | PC = STACK | I | 4/5\* |
| [CPSE](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CPSE.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Compare, Skip if equal | if (Rd ==Rr) PC = PC 2 or 3 |  | 1/2/3 |
| [CP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CP.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Compare | Rd –Rr | Z,C,N,V,H,S | 1 |
| [CPC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CPC.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Compare with Carry | Rd - Rr – C | Z,C,N,V,H,S | 1 |
| [CPI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CPI.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Compare with Immediate | Rd - K | Z,C,N,V,H,S | 1 |
| [SBRC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBRC.html) | [Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Skip if bit in register cleared | if(Rr(b)==0) PC = PC + 2 or 3 |  | 1/2/3 |
| [SBRS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBRS.html) | [Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Skip if bit in register set | if(Rr(b)==1) PC = PC + 2 or 3 |  | 1/2/3 |
| [SBIC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBIC.html) | [P](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_P),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Skip if bit in I/O register cleared | if(I/O(P,b)==0)   PC = PC + 2 or 3 |  | 1/2/3 |
| [SBIS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBIS.html) | [P](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_P),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Skip if bit in I/O register set | if(I/O(P,b)==1) PC = PC + 2 or 3 |  | 1/2/3 |
| [BRBC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRBC.html) | [s](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_s),[k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if Status flag cleared | if(SREG(s)==0) PC = PC + k + 1 |  | 1/2 |
| [BRBS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRBS.html) | [s](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_s),[k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if Status flag set | if(SREG(s)==1) PC = PC + k + 1 |  | 1/2 |
| [BREQ](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BREQ.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if equal | if(Z==1) PC = PC + k + 1 |  | 1/2 |
| [BRNE](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRNE.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if not equal | if(Z==0) PC = PC + k + 1 |  | 1/2 |
| [BRCS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRCS.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if carry set | if(C==1) PC = PC + k + 1 |  | 1/2 |
| [BRCC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRCC.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if carry cleared | if(C==0) PC = PC + k + 1 |  | 1/2 |
| [BRSH](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRSH.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if same or higher | if(C==0) PC = PC + k + 1 |  | 1/2 |
| [BRLO](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRLO.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if lower | if(C==1) PC = PC + k + 1 |  | 1/2 |
| [BRMI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRMI.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if minus | if(N==1) PC = PC + k + 1 |  | 1/2 |
| [BRPL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRPL.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if plus | if(N==0) PC = PC + k + 1 |  | 1/2 |
| [BRGE](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRGE.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if greater than or equal (signed) | if(S==0) PC = PC + k + 1 |  | 1/2 |
| [BRLT](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRLT.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if less than (signed) | if(S==1) PC = PC + k + 1 |  | 1/2 |
| [BRHS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRHS.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if half carry flag set | if(H==1) PC = PC + k + 1 |  | 1/2 |
| [BRHC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRHC.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if half carry flag cleared | if(H==0) PC = PC + k + 1 |  | 1/2 |
| [BRTS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRTS.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if T flag set | if(T==1) PC = PC + k + 1 |  | 1/2 |
| [BRTC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRTC.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if T flag cleared | if(T==0) PC = PC + k + 1 |  | 1/2 |
| [BRVS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRVS.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if overflow flag set | if(V==1) PC = PC + k + 1 |  | 1/2 |
| [BRVC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRVC.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if overflow flag cleared | if(V==0) PC = PC + k + 1 |  | 1/2 |
| [BRIE](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRIE.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if interrupt enabled | if(I==1) PC = PC + k + 1 |  | 1/2 |
| [BRID](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BRID.html) | [k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Branch if interrupt disabled | if(I==0) PC = PC + k + 1 |  | 1/2 |

## Data transfer instructions Instrukcije za premik podatkov

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [MOV](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_MOV.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Copy register | Rd = Rr |  | 1 |
| [MOVW](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_MOVW.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Copy register pair | Rd+1:Rd = Rr+1:Rr, r,d even |  | 1 |
| [LDI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LDI.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[K8](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_K8) | Load Immediate | Rd = K |  | 1 |
| [LDS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LDS.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[k](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_k) | Load Direct | Rd = (k) |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect | Rd = (X) |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect and Post-Increment | Rd = (X), X=X+1 |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect and Pre-Decrement | X=X-1, Rd = (X) |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect | Rd = (Y) |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect and Post-Increment | Rd = (Y), Y=Y+1 |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect and Pre-Decrement | Y=Y-1, Rd = (Y) |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LDD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)+[q](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_q) | Load Indirect with displacement | Rd = (Y+q) |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect | Rd = (Z) |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect and Post-Increment | Rd = (Z), Z=Z+1 |  | 2\* |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Indirect and Pre-Decrement | Z=Z-1, Rd = (Z) |  | 2\* |
| [LAC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LAC.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load and Clear | Z = Rd •($FF-Z) |  | 2 |
| [LAT](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LAT.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load and Toggle | Z = Rd ⊕ (Z) |  | 2 |
| [LAS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LAS.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load and Set | Z = Rd v (Z) |  | 2 |
| [XCH](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_XCH.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Exchange | Z = Rd, Rd = Z |  | 2 |
| [LD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LDD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)+[q](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_q) | Load Indirect with displacement | Rd = (Z+q) |  | 2\* |
| [STS](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_STS.html) | k, [Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Direct | (k) = Rr |  | 2\* |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect | (X) = Rr |  | 2\* |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect and Post-Increment | (X) = Rr, X=X+1 |  | 2\* |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect and Pre-Decrement | X=X-1, (X)=Rr |  | 2\* |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect | (Y) = Rr |  | 2\* |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect and Post-Increment | (Y) = Rr, Y=Y+1 |  | 2 |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect and Pre-Decrement | Y=Y-1, (Y) = Rr |  | 2 |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)+[q](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_q),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect with displacement | (Y+q) = Rr |  | 2 |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect | (Z) = Rr |  | 2 |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect and Post-Increment | (Z) = Rr, Z=Z+1 |  | 2 |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect and Pre-Decrement | Z=Z-1, (Z) = Rr |  | 2 |
| [ST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ST.html) | [X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)+[q](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_q),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Store Indirect with displacement | (Z+q) = Rr |  | 2 |
| [LPM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LPM.html) |  | Load Program Memory | R0 = ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) |  | 3 |
| [LPM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LPM.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Program Memory | Rd = ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) |  | 3 |
| [LPM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LPM.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Load Program Memory and Post-Increment | Rd = ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)), Z=Z+1 |  | 3 |
| [ELPM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ELPM.html) |  | Extended Load Program Memory | R0 = (RAMPZ:[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) |  | 3 |
| [ELPM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ELPM.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Extended Load Program Memory | Rd = (RAMPZ:[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) |  | 3 |
| [ELPM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ELPM.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z) | Extended Load Program Memory and Post Increment | Rd = (RAMPZ:[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)), Z = Z+1 |  | 3 |
| [SPM](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SPM.html) |  | Store Program Memory | ([X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) = R1:R0 |  | - |
| ESPM |  | Extended Store Program Memory | (RAMPZ:[X,Y,Z](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_X_Y_Z)) = R1:R0 |  | - |
| [IN](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_IN.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[P](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_P) | In Port | Rd = P |  | 1 |
| [OUT](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_OUT.html) | [P](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_P),[Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Out Port | P = Rr |  | 1 |
| [PUSH](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_PUSH.html) | [Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr) | Push register on Stack | STACK = Rr |  | 2 |
| [POP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_POP.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Pop register from Stack | Rd = STACK |  | 2 |

## Bit and Bit-test Instructions Bitne instrukcije in testiranje bitov

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [LSL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LSL.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Logical shift left | Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7) | Z,C,N,V,H,S | 1 |
| [LSR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_LSR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Logical shift right | Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0) | Z,C,N,V,S | 1 |
| [ROL](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ROL.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Rotate left through carry | Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7) | Z,C,N,V,H,S | 1 |
| [ROR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_OR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Rotate right through carry | Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0) | Z,C,N,V,S | 1 |
| [ASR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_ASR.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Arithmetic shift right | Rd(n)=Rd(n+1), n=0,...,6 | Z,C,N,V,S | 1 |
| [SWAP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SWAP.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd) | Swap nibbles | Rd(3..0) = Rd(7..4),  Rd(7..4) = Rd(3..0) |  | 1 |
| [BSET](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BSET.html) | [s](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_s) | Set flag | SREG(s) = 1 | SREG(s) | 1 |
| [BCLR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BCLR.html) | [s](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_s) | Clear flag | SREG(s) = 0 | SREG(s) | 1 |
| [SBI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SBI.html) | [P](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_P),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Set bit in I/O register | I/O(P,b) = 1 |  | 2 |
| [CBI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CBI.html) | [P](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_P),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Clear bit in I/O register | I/O(P,b) = 0 |  | 2 |
| [BST](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BST.html) | [Rr](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rr),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Bit store from register to T | T = Rr(b) | T | 1 |
| [BLD](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BLD.html) | [Rd](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_Rd),[b](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_instructions.Bit_and_Bit-test_Instructions.html#avrassembler.wb_b) | Bit load from register to T | Rd(b) = T |  | 1 |
| [SEC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SEC.html) |  | Set carry flag | C =1 | C | 1 |
| [CLC](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLC.html) |  | Clear carry flag | C = 0 | C | 1 |
| [SEN](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SEN.html) |  | Set negative flag | N = 1 | N | 1 |
| [CLN](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLN.html) |  | Clear negative flag | N = 0 | N | 1 |
| [SEZ](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SEZ.html) |  | Set zero flag | Z = 1 | Z | 1 |
| [CLZ](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLZ.html) |  | Clear zero flag | Z = 0 | Z | 1 |
| [SEI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SEI.html) |  | Set interrupt flag | I = 1 | I | 1 |
| [CLI](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLI.html) |  | Clear interrupt flag | I = 0 | I | 1 |
| [SES](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SES.html) |  | Set signed flag | S = 1 | S | 1 |
| [CLN](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLN.html) |  | Clear signed flag | S = 0 | S | 1 |
| [SEV](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SEV.html) |  | Set overflow flag | V = 1 | V | 1 |
| [CLV](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLV.html) |  | Clear overflow flag | V = 0 | V | 1 |
| [SET](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SET.html) |  | Set T-flag | T = 1 | T | 1 |
| [CLT](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLT.html) |  | Clear T-flag | T = 0 | T | 1 |
| [SEH](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SEH.html) |  | Set half carry flag | H = 1 | H | 1 |
| [CLH](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CLH.html) |  | Clear half carry flag | H = 0 | H | 1 |
| [NOP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_NOP.html) |  | No operation |  |  | 1 |
| [SLEEP](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_SLEEP.html) |  | Sleep | See instruction manual |  | 1 |
| [WDR](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_WDR.html) |  | Watchdog Reset | See instruction manual |  | 1 |
| [BREAK](http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_BREAK.html) |  | Execution Break | See instruction manual |  | 1 |
|  |  |  |  |  |  |