AVR Assembler Help

Welcome to the ATMEL AVR Assembler.

- What's New
- Known Issues

Please select between the following Help items:

- General information gives general information about the Assembler
- Assembler source gives a brief description of what a source file looks like
- Instruction mnemonics describes the AVR Instruction set
    - Arithmetic and Logic Instructions
    - Branch Instructions
    - Data Transfer Instructions
    - Bit and Bit-test Instructions
- Assembler directives gives a description of the directives
- Expressions describes how to make constant expressions
    - Expression operands
    - Expression operators
    - Functions in expressions

The Assembler is supplied as a MS-DOS command line program that can be used stand-alone or automatically invoked by AVR Studio. A description of how to use the Command line Assembler is included in this help file

Device specific instruction set summaries:

The actual instruction set varies between the devices. Use these links to verify the instruction set for the desired device.

- AT90S1200
- AT90S2313
- AT90S2323 and AT90S2343
- AT90S2333 and AT90S4433
- AT90S4414 and AT90S8515
- AT90S4434 and AT90S8535
- AT90C8534
- ATtiny10, ATtiny11 and ATtiny12
- ATtiny13/ATtiny2313
- ATtiny15
- ATtiny22
- ATtiny26
- ATtiny28
- ATmega8/ATmega48/ATmega8515/ATmega8535
- ATmega16
- ATmega161
- ATmega162
- ATmega163
- ATmega169
- ATmega32
- ATmega323
- ATmega103
- ATmega64/128

What's New

AVR Assembler v1.73 date nov-2003

- New conditional assembly directives

  #ifdef <symbol>
  #ifndef <symbol>
  #if <expression>
  #elif <expression>
  #else
  #endif

  More details here..
- Output directives
  "list-style: disc;" type=disc>
- Included device directive for ATtiny13.

AVR Assembler v1.56 date 30-apr-2002

- Included device directives for ATmega162, ATmega169, ATmega8515, ATmega8535, ATtiny26, AT86RF401.
- Added *.def.inc files for the above mentioned devices.

Assembler source

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.

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.

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

```
[label:] directive [operands] [Comment]
[label:] instruction [operands] [Comment]
Comment
Empty line
```

A comment has the following form:

```
; [Text]
```

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.

Examples:

```
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
```

Note that there are no restrictions with respect to column placement of labels, directives, comments or instructions.

General information

The Assembler translates assembly source code into object code. The generated object code can be used as input to a simulator such as the ATMEL AVR Simulator or an emulator such as the ATMEL AVR In-Circuit Emulator. The Assembler also generates a PROMable code which can be programmed directly into the program memory of an AVR microcontroller

The Assembler generates fixed code allocations, consequently no linking is necessary.

The instruction set of the AVR family of microcontrollers is only briefly described, refer to the AVR Data Book in order to get more detailed knowledge of the instruction set for the different microcontrollers.

Instruction mnemonics

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1

NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
ADIW	Rdl,K6	Add Immediate to Word	Rdh:Rdl = Rdh:Rdl + K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
EICALL	None	Extended Indirect Call to (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	None	4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3

BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

* Cycle times for data memory accesses assume internal memory accesses, and are not valid for accesses through the external RAM interface. For the instructions CALL, ICALL, EICALL, RCALL, RET and RETI, add three cycles plus two cycles for each wait state in devices with up to 16 bit PC (128KB program memory). For devices with more than 128KB program memory, add five cycles plus three cycles for each wait state.

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*

LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
ELPM	None	Extended Load Program Memory	R0 = (RAMPZ:Z)	None	3
ELPM	Rd,Z	Extended Load Program Memory	Rd = (RAMPZ:Z)	None	3
ELPM	Rd,Z+	Extended Load Program Memory and Post Increment	Rd = (RAMPZ:Z), Z = Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
ESPM	None	Extended Store Program Memory	(RAMPZ:Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

* Cycle times for data memory accesses assume internal memory accesses and are not valid for accesses through the external RAM interface. For the LD, ST, LDD, STD, LDS, STS, PUSH and POP instructions, add one cycle plus one cycle for each wait state.

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1

BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1
BREAK	None	Execution Break	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

Assembler directives

The Assembler supports a number of directives. The directives are not translated directly into opcodes. Instead, they are used to adjust the location of the program in memory, define macros, initialize memory and so on. An overview of the directives is given in the following table.

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
CSEGSIZE	Program memory size
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define Constant word(s)
ENDM, ENDMACRO	EndMacro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn Macro expansion in list file on
MACRO	Begin Macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

Note that all directives above must be preceded by a period.

New directives below must be preceded by a hash symbol #.

New Directive	Description
ELSE,ELIF	Conditional assembly
ENDIF	Conditional assembly
ERROR	Outputs an error message
IF,IFDEF,IFNDEF	Conditional assembly
MESSAGE	Outputs a message string

BYTE - Reserve bytes to a variable

The BYTE directive reserves memory resources in the SRAM. 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 only be used within a Data Segment (see directives CSEG and DSEG). Note that a parameter must be given. The allocated bytes are not initialized.

Syntax:

```
LABEL: .BYTE expression
```

Example:

```
.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.

Syntax:

```
.CSEG
```

Example:

```
.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
```

CSEGSIZE - Program Memory Size

AT94K devices have a user configurable memory partition between the AVR Program memory and the data memory. The program and data SRAM is divided into three blocks: 10K x 16 dedicated program SRAM, 4K x 8 dedicated data SRAM, and 6K x 16 or 12K x 8 configurable SRAM which may be swapped between program and data memory spaces in 2K x 16 or 4K x 8 partitions.
This directive is used to specify the size of the program memory block.

Syntax:

```
.CSEGSIZE = 10 | 12 | 14 | 16
```

Example:

```
.CSEGSIZE = 12              ; Specifies the program meory size as 12K x
16
```

DB - Define constant byte(s) in program memory and EEPROM

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.

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.

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. I*f 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

Syntax:
```
LABEL:  .DB expressionlist
```

Example:
```
.CSEG
consts: .DB 0, 255, 0b01010101, -128, 0xaa

.ESEG
const2: .DB 1,2,3
```

DEF - Set a symbolic name on a register

The DEF directive allows the registers to be referred to through symbols. A defined symbol can be used in the rest of the program to refer to the register it is assigned to. A register can have several symbolic names attached to it. A symbol can be redefined later in the program.

Syntax:
```
.DEF Symbol=Register
```

Example:
```
.DEF temp=R16
.DEF ior=R0

.CSEG
 ldi temp,0xf0  ; Load 0xf0 into temp register
 in ior,0x3f  ; Read SREG into ior register
 eor temp,ior  ; Exclusive or temp and ior
```

DEVICE - Define which device to assemble for

The DEVICE directive allows the user to tell the Assembler which device the code is to be executed on. Using this directive, a warning is issued if an instruction not supported by the specified device occurs. If the Code Segment or EEPROM Segment are larger than supplied by the device, a warning message is given. If the directive is not used, it is assumed that all instructions are supported and that there are no restrictions on Program and EEPROM memory.

Syntax:
```
.DEVICE <device code>
```

Table: Device codes:

Classic	Tiny	Mega	Other
AT90S120	ATtiny11	ATmega8	AT94K

AT90S2313	ATtiny12	ATmega16	AT86RF401
AT90S2323	ATtiny13	ATmega161	
AT90S2333	ATtiny22	ATmega162	
AT90S4414	ATtiny26	ATmega163	
AT90S4434		ATmega169	
AT90S8515		ATmega32	
AT90S8534		ATmega323	
AT90S8535		ATmega103	
AT90S2343		ATmega104	
AT90S4433		ATmega8515	
		ATmega8535	
		ATmega64	
		ATmega128	

Example:
```
.DEVICE AT90S1200  ; Use the AT90S1200

.CSEG
        push r30   ; This statement will generate a warning
                   ; since the specified device does not
                   ; have this instruction
```

Note: There has been a change of names that took effect 14.06.2001. The following devices are affected:

Old name	New name
ATmega104	ATmega128
ATmega32	ATmega323
ATmega164	ATmega16

In order NOT to break old projects, both old and new device directives are allowed for the parts that are affected.

DSEG - Data Segment

The DSEG directive defines the start of a Data Segment. An Assembler file can consist of several Data Segments, which are concatenated into one 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.

Syntax:
```
.DSEG
```

Example:
```
.DSEG                       ; Start data segment
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
```

DW - Define constant word(s) in program memory and EEPROM

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 twos complement of the number will be placed in the program memory or EEPROM memory location.

Syntax:
```
LABEL: .DW expressionlist
```

Example:
```
.CSEG
varlist:  .DW 0, 0xffff, 0b1001110001010101, -32768, 65535

.ESEG
eevarlst: .DW 0,0xffff,10
```

ELIF,ELSE - conditional assembly

Conditional assembly includes a set of commands at assembly time. Will include code to corresponding ENDIF directive if the IFDEF or IFNDE or IF directive failed.

Syntax:
#ELIF<expression>
#ELSE

#IFDEF <symbol> |#IFNDEF <symbol>
...
#ELSE | #ELIF<expression>
...
#ENDIF

Example:
#IFDEF DEBUG
#MESSAGE "Debugging.."
#ELSE
#MESSAGE "Release.."
#ENDIF

ENDIF - conditional assembly

Conditional assembly includes a set of commands at assembly time. The ENDIF directive define the end for the conditional IF or IFDEF or IFNDEF directives.

Syntax:
#ENDIF

#IFDEF <symbol> |#IFNDEF <symbol>
...
#ELSE | #ELIF<expression>
...
#ENDIF

Example:
#IFNDEF DEBUG
#MESSAGE "Release.."
#ELSE
#MESSAGE "Debugging.."
#ENDIF

ENDMACRO - End macro

The ENDMACRO directive defines the end of a Macro definition. The directive does not take any parameters. See the MACRO directive for more information on defining Macros.

Syntax:
```
.ENDMACRO
```

Example:
```
.MACRO SUBI16               ; Start macro definition
        subi r16,low(@0)    ; Subtract low byte
        sbci r17,high(@0)   ; Subtract high byte
.ENDMACRO
```

EQU - Set a symbol equal to an expression

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.

Syntax:
```
.EQU label = expression
```

Example:
```
.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
```

ERROR - Outputs an error message string

The ERROR directive outputs a string and halts the assembling. May be practical when using conditional assembly.

Syntax:
```
#ERROR "<string>"
```

Example:
```
#IFDEF TOBEDONE
#ERROR "Still stuff to be done.."
#ENDIF
```

ESEG - EEPROM Segment

The ESEG directive defines the start of an EEPROM Segment. An Assembler file can consist of several EEPROM Segments, which are concatenated into one EEPROM Segment when assembled. An EEPROM Segment will normally only consist of DB and DW directives (and labels). The EEPROM 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 EEPROM. The directive does not take any parameters.

Syntax:
```
.ESEG
```

Example:
```
.DSEG                    ; Start data segment
var1:   .BYTE 1          ; reserve 1 byte to var1
table:  .BYTE tab_size   ; reserve tab_size bytes.

.ESEG
eevar1: .DW 0xffff        ; initialize 1 word in EEPROM
```

EXIT - Exit this file

The EXIT directive tells the Assembler to stop assembling the file. Normally, the Assembler runs until end of file (EOF). If an EXIT directive appears in an included file, the Assembler continues from the line following the INCLUDE directive in the file containing the INCLUDE directive.

Syntax:
.EXIT

Example:
```
.EXIT  ; Exit this file
```

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.

Syntax:
```
.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 iodefs.asm    ; Include I/O definitions
        in r0,sreg     ; Read status register
```

IF,IFDEF,IFNDEF - conditional assembly

Conditional assembly includes a set of commands at assembly time. The IFDEF directive will include code till the corresponding ELSE directive if <symbol> is defined. The symbol must be defined with the EQU or SET directive. (Will not work with the DEF directive) The IF directive will include code if <expression> is evaluated different from 0. Valid till the corresponding ELSE or ENDIF directive.

Up to 5 levels of nesting is possible.

Syntax:
#IFDEF <symbol>
#IFNDEF <symbol>
#IF <expression>

#IFDEF <symbol> |#IFNDEF <symbol>
...
#ELSE | #ELIF<expression>
...
#ENDIF

Example:
.MACRO SET_BAT
#IF @0>0x3F
#MESSAGE "Address larger than 0x3f"
lds @2, @0
sbr @2, (1<<@1)
sts @0, @2
#ELSE
#MESSAGE "Address less or equal 0x3f"
.ENDIF
.ENDMACRO

LIST - Turn the listfile generation on

The LIST directive tells the Assembler to turn listfile generation on. The Assembler generates a listfile which is a combination of assembly source code, addresses and opcodes. Listfile generation is turned on by default. The directive can also be used together with the NOLIST directive in order to only generate listfile of selected parts of an assembly source file.

Syntax:
.LIST

Example:

```
.NOLIST                ; Disable listfile generation
.INCLUDE "macro.inc"   ; The included files will not
```

```
.INCLUDE "const.def"   ; be shown in the listfile
.LIST                  ; Reenable listfile generation
```

LISTMAC - Turn macro expansion on

The LISTMAC directive tells the Assembler that when a macro is called, the expansion of the macro is to be shown on the listfile generated by the Assembler. The default is that only the macro-call with parameters is shown in the listfile.

Syntax:

```
.LISTMAC
```

Example:

```
.MACRO MACX          ; Define an example macro
        add  r0,@0   ; Do something
        eor  r1,@1   ; Do something
.ENDMACRO            ; End macro definition

.LISTMAC             ; Enable macro expansion
        MACX r2,r1   ; Call macro, show expansion
```

MACRO - Begin macro

The MACRO directive tells the Assembler that this is the start of a Macro. The MACRO directive takes the Macro name as parameter. When the name of the Macro is written later in the program, the Macro definition is expanded at the place it was used. A Macro can take up to 10 parameters. These parameters are referred to as @0-@9 within the Macro definition. When issuing a Macro call, the parameters are given as a comma separated list. The Macro definition is terminated by an ENDMACRO directive.

By default, only the call to the Macro is shown on the listfile generated by the Assembler. In order to include the macro expansion in the listfile, a LISTMAC directive must be used. A macro is marked with a + in the opcode field of the listfile.

Syntax:

```
.MACRO macroname
```

Example:

```
.MACRO SUBI16                   ; Start macro definition
        subi @1,low(@0)         ; Subtract low byte
        sbci @2,high(@0)        ; Subtract high byte
.ENDMACRO                       ; End macro definition

.CSEG                           ; Start code segment
        SUBI16 0x1234,r16,r17   ; Sub.0x1234 from r17:r16
```

MESSAGE - Output a message string

The MESSAGE directive outputs a string. Practical when using conditional assembly.

Syntax:

```
#MESSAGE "<string>"
```

Example:

```
#IFDEF DEBUG
```

```
#MESSAGE "Debug mode"
#ENDIF
```

NOLIST - Turn listfile generation off

The NOLIST directive tells the Assembler to turn listfile generation off. The Assembler normally generates a listfile which is a combination of assembly source code, addresses and opcodes. Listfile generation is turned on by default, but can be disabled by using this directive. The directive can also be used together with the LIST directive in order to only generate listfile of selected parts of an assembly source file.

Syntax:

```
.NOLIST
```

Example:

```
.NOLIST                  ; Disable listfile generation
.INCLUDE "macro.inc"     ; The included files will not
.INCLUDE "const.def"     ; be shown in the listfile
.LIST                    ; Reenable listfile generation
```

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. If the directive is preceded by a label (on the same source code line), the label will be given the value of the parameter. The default values of the Code and the EEPROM location counters are zero, and the default value of the SRAM location counter is 32 (due to the registers occupying addresses 0-31) when the assembling is started. Note that the SRAM and EEPROM location counters count bytes whereas the Program memory location counter counts words.

Syntax:

```
.ORG expression
```

Example:

```
.DSEG                ; Start data segment

.ORG 0x37            ; Set SRAM address to hex 37
variable: .BYTE 1    ; Reserve a byte at SRAM adr.37H

.CSEG
.ORG 0x10            ; Set Program Counter to hex 10
          mov r0,r1  ; Do something
```

SET - Set a symbol equal to an expression

The SET directive assigns a value to a label. This label can then be used in later expressions. A label assigned to a value by the SET directive can be changed later in the program.

Syntax:

```
.SET label = expression
```

Example:

```
.SET io_offset = 0x23
```

```
.SET porta      = io_offset + 2

.CSEG                   ; Start code segment
        clr r2          ; Clear register 2
        out porta,r2    ; Write to Port A
```

Expressions

The Assembler incorporates expressions. Expressions can consist of operands, operators and functions. All expressions are internally 32 bits.

Operands

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

Operators

The Assembler supports a number of operators which are described here. The higher the precedence, the higher the priority. Expressions may be enclosed in parentheses, and such expressions are always evaluated before combined with anything outside the parentheses.

The following operators are defined:

Symbol	Description
!	Logical Not
~	Bitwise Not
–	Unary Minus
*	Multiplication
/	Division
+	Addition

-	Subtraction
<<	Shift left
>>	Shift right
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&	Bitwise And
^	Bitwise Xor
|	Bitwise Or
&&	Logical And
||	Logical Or

Logical Not

```
Symbol:         !
Description:    Unary operator which returns 1 if the expression was zero,
and returns 0 if the expression was nonzero
Precedence:     14
Example:        ldi r16,!0xf0   ; Load r16 with 0x00
```

Bitwise Not

```
Symbol:         ~
Description: Unary operator which returns the input expression with all
bits inverted
Precedence:  14
Example:     ldi r16,~0xf0   ; Load r16 with 0x0f
```

Unary Minus

```
Symbol:      -
Description: Unary operator which returns the arithmetic negation of an
expression
Precedence:  14
Example:     ldi r16,-2   ; Load -2(0xfe) in r16
```

Multiplication

```
Symbol:      *
Description: Binary operator which returns the product of two expressions
Precedence:  13
Example:     ldi r30,label*2 ; Load r30 with label*2
```

Division

```
Symbol:      /
Description: Binary operator which returns the integer quotient of the
left expression divided by the right expression
Precedence:  13
Example:     ldi r30,label/2 ; Load r30 with label/2
```

Addition

```
Symbol:      +
Description: Binary operator which returns the sum of two expressions
Precedence:  12
Example:     ldi r30,c1+c2  ; Load r30 with c1+c2
```

Subtraction

```
Symbol:      -
Description: Binary operator which returns the left expression minus the
right expression
Precedence:  12
Example:     ldi r17,c1-c2  ;Load r17 with c1-c2
```

Shift left

```
Symbol:      <<
Description: Binary operator which returns the left expression shifted
left the number given by the right expression
Precedence:  11
Example:     ldi r17,1<<bitmask  ;Load r17 with 1 shifted left bitmask
times
```

Shift right

```
Symbol:      >>
Description: Binary operator which returns the left expression shifted
right the number given by the right expression
Precedence:  11
Example:     ldi r17,c1>>c2  ;Load r17 with c1 shifted right c2 times
```

Less than

```
Symbol:      <
Description: Binary operator which returns 1 if the signed expression to
the left is Less than the signed expression to the right, 0 otherwise
Precedence:  10
Example:     ori r18,bitmask*(c1<c2)+1  ;Or r18 with an expression
```

Less or equal

```
Symbol:      <=
Description: Binary operator which returns 1 if the signed expression to
the left is Less than or Equal to the signed expression to the right, 0
```

```
otherwise
Precedence:  10
Example:     ori r18,bitmask*(c1<=c2)+1 ;Or r18 with an expression
```

Greater than

```
Symbol:      >
Description: Binary operator which returns 1 if the signed expression to
the left is Greater than the signed expression to the right, 0 otherwise
 Precedence: 10
 Example:    ori r18,bitmask*(c1>c2)+1  ;Or r18 with an expression
```

Greater or equal

```
Symbol:      >=
Description: Binary operator which returns 1 if the signed expression to
the left is Greater than or Equal to the signed expression to the right,
0 otherwise
Precedence:  10
Example:     ori r18,bitmask*(c1>=c2)+1 ;Or r18 with an expression
```

Equal

```
Symbol:      ==
Description: Binary operator which returns 1 if the signed expression to
the left is Equal to the signed expression to the right, 0 otherwise
Precedence:  9
Example:     andi r19,bitmask*(c1==c2)+1 ;And r19 with an expression
```

Not equal

```
Symbol:      !=
Description: Binary operator which returns 1 if the signed expression to
the left is Not Equal to the signed expression to the right, 0 otherwise
Precedence:  9
Example:     .SET flag=(c1!=c2)  ;Set flag to 1 or 0
```

Bitwise And

```
Symbol:      &
Description: Binary operator which returns the bitwise And between two
expressions
Precedence:  8
Example:     ldi r18,High(c1&c2) ;Load r18 with an expression
```

Bitwise Xor

```
Symbol:      ^
Description: Binary operator which returns the bitwise Exclusive Or
between two expressions
Precedence:  7
Example:     ldi r18,Low(c1^c2) ;Load r18 with an expression
```

Bitwise Or

```
Symbol:      |
Description: Binary operator which returns the bitwise Or between two
expressions
Precedence:  6
Example:     ldi r18,Low(c1|c2) ;Load r18 with an expression
```

Logical And

```
Symbol:      &&
Description: Binary operator which returns 1 if the expressions are both
nonzero, 0 otherwise
Precedence:  5
Example:     ldi r18,Low(c1&&c2)  ;Load r18 with an expression
```

Logical Or

```
Symbol:      ||
Description: Binary operator which returns 1 if one or both of the
expressions are nonzero, 0 otherwise
Precedence:  4
Example:     ldi r18,Low(c1||c2)  ;Load r18 with an expression
```

Functions

The following functions are defined:

- LOW(expression) returns the low byte of an expression
- HIGH(expression) returns the second byte of an expression
- BYTE2(expression) is the same function as HIGH
- BYTE3(expression) returns the third byte of an expression
- BYTE4(expression) returns the fourth byte of an expression
- LWRD(expression) returns bits 0-15 of an expression
- HWRD(expression) returns bits 16-31 of an expression
- PAGE(expression) returns bits 16-21 of an expression
- EXP2(expression) returns 2 to the power of expression
- LOG2(expression) returns the integer part of log2(expression)

AT90S1200 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1

SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2

BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1

BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

AT90S2313 Instructions

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles

ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
EICALL	None	Extended Indirect Call to (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	None	4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3

SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*

LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1

SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

AT90S2323 and AT90S2343 Instruction Set Summary

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2

SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2

BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct from Data Space	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*

ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1

SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

AT90S4414 and AT90S8515 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1

OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2

BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2

ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1

SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

AT90S4414 and AT90S8515 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1

CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2

BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1

OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

AT90S4434 and AT90S8535 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression

K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

Instruction mnemonics

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2

IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1

LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1

SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega8 / ATmega48 / ATmega8515 / ATmega8535 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file

b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega16 and ATmega32 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2

FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k		if(H==0) PC = PC + k + 1	None	1/2

		Branch if half carry flag cleared			
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2

LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1

NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1
BREAK	None	Execution Break	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega161 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1

INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2

BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2

ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1

CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega162 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1

ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3

SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*

LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1

SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1
BREAK	None	Execution Break	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega163 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1

ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*

CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*

LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1

BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega169 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2

JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression

K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega64/128 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2

FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k		if(H==0) PC = PC + k + 1	None	1/2

		Branch if half carry flag cleared			
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2

LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1

NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1
BREAK	None	Execution Break	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega103 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1

INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2

BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3

ELPM (Note 1)	None	Extended Load Program Memory	R0 = (RAMPZ:Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Note 1: Not supported in ATmega603

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1

SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATmega64/128 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1

TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 = Rd * Rr	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 = (Rd * Rr) << 1	Z,C	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Jump	PC = k	None	3
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2

BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2

ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1

SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1
BREAK	None	Execution Break	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATtiny10/11/12 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1

EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2

BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1

SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATtiny13/ATtiny2313 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1

ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2

BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*

STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
SPM	None	Store Program Memory	(Z) = R1:R0	None	-
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1

SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1
BREAK	None	Execution Break	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATtiny15 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1

SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2

BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2*
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1

BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATtiny22 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1

SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*

LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1

SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATtiny26 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2

SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
EIJMP	None	Extended Indirect Jump (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3

BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*

STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
LPM	Rd,Z	Load Program Memory	Rd = (Z)	None	3
LPM	Rd,Z+	Load Program Memory and Post-Increment	Rd = (Z), Z=Z+1	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1

CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

ATtiny28 Instruction Set

The Assembler accepts mnemonic instructions from the instruction set. A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1

SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2

BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
LPM	None	Load Program Memory	R0 = (Z)	None	3
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2

CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1
CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1
NOP	None	No operation	None	None	1
SLEEP	None	Sleep	See instruction manual	None	1
WDR	None	Watchdog Reset	See instruction manual	None	1

The Assembler is not case sensitive.

The operands have the following forms:

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

Instruction Set Nomenclature:

Status Register (SREG)

SREG : Status register

C : Carry flag in status register

Z : Zero flag in status register

N : Negative flag in status register

V : Two's complement overflow indicator

S : N [+] V, For signed tests

H : Half Carry flag in the status register

T : Transfer bit used by BLD and BST instructions

I : Global interrupt enable/disable flag

Registers and Operands

Rd : Destination (and source) register in the register file

Rr : Source register in the register file

R : Result after instruction is executed

K : Constant data

k : Constant address

b : Bit in the register file or I/O register (3 bit)

s: Bit in the status register (3 bit)

X,Y,Z : Indirect address register (X=R27:R26, Y=R29:R28 and Z=R31:R30)

A : I/O location address

q : Displacement for direct addressing (6 bit)

I/O Registers

RAMPX, RAMPY, RAMPZ

Registers concatenated with the X, Y and Z registers enabling indirect addressing of the whole data space on MCUs with more than 64K bytes data space, and constant data fetch on MCUs with more than 64K bytes program space.

RAMPD

Register concatenated with the Z register enabling direct addressing of the whole data space on MCUs with

more than 64K bytes data space.

EIND

Register concatenated with the instruction word enabling indirect jump and call to the whole program space on MCUs with more than 128K bytes program space.

Stack

STACK:Stack for return address and pushed registers

SP:Stack Pointer to STACK

Flags

<> :Flag affected by instruction

0 :Flag cleared by instruction

1 :Flag set by instruction

- :Flag not affected by instruction

ADC - Add with Carry

Description:

Adds two registers and the contents of the C flag and places the result in the destination register Rd.

Operation:

(i)Rd Rd + Rr + C

Syntax: Operands: Program Counter:

(i)ADC Rd, Rr0 ≤ d ≤ 31, 0 ≤ r ≤ 31 PC <- PC + 1

16-bit Opcode:

0001 11rd dddd rrrr

Status Register (SREG) Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

H:Rd3 Rr3+Rr3 *R3 + R3* Rd3 Set if there was a carry from bit 3; cleared otherwise

S: N [+] V For signed tests.

V:Rd7 Rr7 *R7 + Rd7 Rr7* R7 Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0* Set if the result is $00; cleared otherwise.

C:Rd7 Rr7 + Rr7 *R7 + R7* Rd7 Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
        ; Add R1:R0 to R3:R2
add r2,r0    ; Add low byte
adc r3,r1    ; Add with carry high byte
```

Words: *1 (2 bytes)*

Cycles: *1*

ADD - Add without Carry

Description:

Adds two registers without the C flag and places the result in the destination register Rd.

Operation:

Rd <- Rd + Rr

Syntax:	Operands:	Program Counter:
ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC <- PC + 1

16-bit Opcode:

0000 11rd dddd rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

H:Rd3 Rr3+Rr3 *R3 + R3* Rd3

Set if there was a carry from bit 3; cleared otherwise

S: N [+] V, For signed tests.

V:Rd7 Rr7 *R7 + Rd7 Rr7* R7 Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: R7 R6 R5 R4 R3 R2 R1 R0 Set if the result is $00; cleared otherwise.

C:Rd7 Rr7 +Rr7 *R7 + R7* Rd7 Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

add r1,r2 ; Add r2 to r1 (r1=r1+r2)

add r28, r28 ; Add r28 to itself (r28=r28+r28)

Words: *1 (2 bytes)*

Cycles: *1*

ADIW - Add Immediate to Word

Description:

Adds an immediate value (0-63) to a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)Rd+1:Rd <- Rd+1:Rd + K

Syntax: Operands: Program Counter:

(i)ADIW Rd+1:Rd,K d E {24,26,28,30}, 0 <= K <= 63 PC <- PC + 1

16-bit Opcode:

1001 0110 KKdd KKKK

Status Register (SREG) and Boolean Formulae:

I T H S V N Z C
- - - <> <> <> <> <>

S: N [+] V, For signed tests.

V:Rdh7 R15

Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R15

Set if MSB of the result is set; cleared otherwise.

Z: *R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $0000; cleared otherwise.

C: *R15* Rdh7

Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
adiw r25:24,1                ; Add 1 to r25:r24
adiw ZH:ZL,63                ; Add 63 to the Z pointer(r31:r30)
```

Words: *1 (2 bytes)*

Cycles: *2*

AND - Logical AND

Description:

Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i)Rd Rd Rr

Syntax: Operands: Program Counter:

(i)AND Rd,Rr $0 \le d \le 31, 0 \le r \le 31$ PC PC + 1

16-bit Opcode:

0010 00rd dddd rrrr

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

S: N [+] V,

For signed tests.

V:0

Cleared

N:

R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

R

(Result) equals Rd after the operation.

Example:

```
    and r2,r3          ; Bitwise and r2 and r3, result in r2
ldi r16,1              ; Set bitmask 0000 0001 in r16
and r2, r16            ; Isolate bit 0 in r2
```

Words: *1 (2 bytes)*

Cycles: *1*

ANDI - Logical AND with Immediate

Description:

Performs the logical AND between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i)Rd Rd K

Syntax: Operands: Program Counter:

(i)ANDI Rd,K 16 ≤ d ≤ 31, 0 ≤ K ≤ 255 PC PC + 1

16-bit Opcode:

0111 KKKK dddd KKKK

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

S: N [+] V, For signed tests.

V:0 Cleared

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0* Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
andi r17,$0F      ; Clear upper nibble of r17
andi r18,$10      ; Isolate bit 4 in r18
andi r19,$AA      ; Clear odd bits of r19
```

Words: *1 (2 bytes)*

Cycles: *1*

ASR - Arithmetic Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C flag of the SREG. This operation effectively divides a signed value by two without changing its sign. The carry flag can be used to round the result.

Operation:

(i)

Syntax:	Operands:	Program Counter:
(i)ASR R	d0 ≤ d ≤ 31	PC PC + 1

16-bit Opcode:

1001 010d dddd 0101

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	<>

S: N [+] V,

For signed tests.

V:N [+] C

(For N and C after the shift)

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $00; cleared otherwise.

C:Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
dir 16,$10          ; Load decimal 16 into r16
asr r16             ; r16=r16 / 2
ldi r17,$FC         ; Load -4 in r17
asr r17             ; r17=r17/2
```

Words: *1 (2 bytes)*

Cycles: *1*

BCLR - Bit Clear in SREG

Description:

Clears a single flag in SREG.

Operation:

(i)SREG(s) 0

Syntax:	Operands:	Program Counter:
(i)BCLR s	0 ≤ s ≤ 7	PC <- PC + 1

16-bit Opcode:

1001 0100 1sss 1000

Status Register (SREG) and Boolean Formulae:

I T H S V N Z C

<> <> <> <> <> <> <> <>

I:0 if s = 7; Unchanged otherwise.

T:0 if s = 6; Unchanged otherwise.

H:0 if s = 5; Unchanged otherwise.

S:0 if s = 4; Unchanged otherwise.

V:0 if s = 3; Unchanged otherwise.

N:0 if s = 2; Unchanged otherwise.

Z:0 if s = 1; Unchanged otherwise.

C:0 if s = 0; Unchanged otherwise.

Example:

bclr 0; Clear carry flag

bclr 7; Disable interrupts

Words: *1 (2 bytes)*

Cycles: *1*

BLD - Bit Load from the T Flag in SREG to a Bit in Register.

Description:

Copies the T flag in the SREG (status register) to bit b in register Rd.

Operation:

Rd(b) <- T

Syntax:	Operands:	Program Counter:
BLD Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC <- PC + 1

16 bit Opcode:

1111 100d dddd 0bbb

Status Register (SREG) and Boolean Formulae:

I T H S V N Z C
- - - - - - - -
Example:

; Copy bit

bst r1,2 ; Store bit 2 of r1 in T flag

bld r0, 4 ; Load T flag into bit 4 of r0

Words: *1 (2 bytes)*

Cycles: *1*

BRBC - Branch if Bit in SREG is Cleared

Description:

Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is
represented in two's complement form.

Operation:

If SREG(s) = 0 then PC <- PC + k + 1, else PC <- PC + 1

Syntax: Operands: Program Counter:

BRBC s, k 0 ≤ s ≤ 7, -64 ≤ k ≤ +63 PC <- PC + k + 1

PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk ksss

Status Register (SREG) and Boolean Formulae:

I T H S V N Z C
- - - - - - - -
Example:

cpi r20,5 ; Compare r20 to the value 5

brbc 1,noteq ; Branch if zero flag cleared

...

noteq:nop ; Branch destination (do nothing)

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRBS - Branch if Bit in SREG is Set

Description:

Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form.

Operation:

If SREG(s) = 1 then PC PC + k + 1, else PC PC + 1

Syntax:	Operands:	Program Counter:
BRBS s,k	0 ≤ s ≤ 7, -64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk ksss

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
bstr 0,3          ; Load T bit with bit 3 of r0
brbs 6, bitset    ; Branch T bit was set
...
bitset:  nop              ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRCC - Branch if Carry Cleared

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k).

Operation:

If C = 0 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRCC k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
add  r22,r23          ; Add r23 to r22
brcc nocarry          ; Branch if carry cleared
...
nocarry:   nop                ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRCS - Branch if Carry Set

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset

from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

If C = 1 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRCS k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
cpi r26, $56              ; Compare r26 with $56
brcs carry                ; Branch if carry set
...
carry:    nop             ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BREAK - Break

Description:

The BREAK instruction is used by the On-Chip Debug system, and is normally not used in the application software. When the BREAK instruction is executed, the AVR CPU is set in the Stopped Mode. This gives the On-Chip Debugger access to internal resources.

If any lock bits are set, or either the JTAGEN or OCDEN fuses are unprogrammed, the CPU will treat the BREAK instruction as a NOP and will not enter the Stopped Mode.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

On-Chip Debug system break.

Syntax:	Operands:	Program Counter:
BREAK	None	PC <- PC + 1

16-bit Opcode:

1001 0101 1001 1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Words: *1 (2 bytes)*

Cycles: *1*

BREQ - Branch if Equal

Description:

Conditional relative branch. Tests the Zero flag (Z) and branches relatively to PC if Z is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 1,k).

Operation:

If Rd = Rr (Z = 1) then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BREQ k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k001

Status Register (SREG) and Boolean Formulae:

I T H S V N Z C

- - - - - - - -

Example:

```
cp      r1, r0          ; Compare registers r1 and r0

breq    equal           ; Branch if registers equal

...

equal:  nop             ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRGE - Branch if Greater or Equal (Signed)

Description:

Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 4,k).

Operation:

If Rd ≥ Rr (N V = 0) then PC PC + k + 1, else PC PC + 1

Syntax:	Operands:	Program Counter:
BRGE	k-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k100

Status Register (SREG) and Boolean Formulae:

I T H S V N Z C
- - - - - - - -

Example:

```
cp     r11,r12          ; Compare registers r11 and r12
brge   greateq          ; Branch if r11 ≥ r12 (signed)
...
greateq:   nop                  ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRHC - Branch if Half Carry Flag is Cleared

Description:

Conditional relative branch. Tests the Half Carry flag (H) and branches relatively to PC if H is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 5,k).

Operation:

If H = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:	Operands:	Program Counter:
BRHC k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k101

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - - - -

Example:

brhc hclear ; Branch if half carry flag cleared

...

hclear: nop ; Branch destination (do nothing)

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRHS - Branch if Half Carry Flag is Set

Description:

Conditional relative branch. Tests the Half Carry flag (H) and branches relatively to PC if H is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 5,k).

Operation:

If H = 1 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRHS k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k101

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - - - -

Example:

```
brhs hset       ; Branch if half carry flag set

...

hset:   nop             ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRID - Branch if Global Interrupt is Disabled

Description:

Conditional relative branch. Tests the Global Interrupt flag (I) and branches relatively to PC if I is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 7,k).

Operation:

If I = 0 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRID k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
brid intdis             ; Branch if interrupt disabled

...

intdis:nop              ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRIE - Branch if Global Interrupt is Enabled

Description:

Conditional relative branch. Tests the Global Interrupt flag (I) and branches relatively to PC if I is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 7,k).

Operation:

If I = 1 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRIE k	-64 ≤ k ≤ +63	PC PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
brie inten          ; Branch if interrupt enabled
...
inten:nop           ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRLO - Branch if Lower (Unsigned)

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned binary number represented in Rd was smaller than the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

If Rd < Rr (C = 1) then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRLO k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
eor  r19,r19                ; Clear r19
loop:incr19                 ; Increase r19
...
cpi  r19,$10                ; Compare r19 with $10
brlo loop                   ; Branch if r19 < $10 (unsigned)
nop                         ; Exit from loop (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRLT - Branch if Less Than (Signed)

Description:

Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was less than the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 4,k).

Operation:

If Rd < Rr (N [+] V = 1) then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRLT k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k100

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
cp r16,r1                    ; Compare r16 to r1
brlt less                    ; Branch if r16 < r1 (signed)
...
less:nop                     ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRMI - Branch if Minus

Description:

Conditional relative branch. Tests the Negative flag (N) and branches relatively to PC if N is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 2,k).

Operation:

If N = 1 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRMI k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k010

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
subi r18,4          ; Subtract 4 from r18
brmi negative       ; Branch if result negative
...
negative:  nop              ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRNE - Branch if Not Equal

Description:

Conditional relative branch. Tests the Zero flag (Z) and branches relatively to PC if Z is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was not equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 1,k).

Operation:

If Rd ≠ Rr (Z = 0) then PC PC + k + 1, else PC PC + 1

Syntax:	Operands:	Program Counter:
BRNE k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k001

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
eor r27,r27          ; Clear r27
loop:inc r27         ; Increase r27
...
cpir27,5             ; Compare r27 to 5
brneloop             ; Branch if r27<>5
nop                  ; Loop exit (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRPL - Branch if Plus

Description:

Conditional relative branch. Tests the Negative flag (N) and branches relatively to PC if N is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 2,k).

Operation:

If N = 0 then PC PC + k + 1, else PC PC + 1

Syntax: Operands: Program Counter:

BRPL k -64 ≤ k ≤ +63 PC <- PC + k + 1

PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k010

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
subi r26,$50          ; Subtract $50 from r26
brpl positive         ; Branch if r26 positive
...
positive:nop          ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRSH - Branch if Same or Higher (Unsigned)

Description:

Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is cleared. If the instruction is executed immediately after execution of any of the instructions CP, CPI, SUB or SUBI the branch will occur if and only if the unsigned binary number represented in Rd was greater than or equal to the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k).

Operation:

If Rd ≥Rr (C = 0) then PC PC + k + 1, else PC PC + 1

Syntax:	Operands:	Program Counter:
BRSH k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
subi r19,4                    ; Subtract 4 from r19
brsh highsm                   ; Branch if r19 >= 4 (unsigned)
...
highsm:nop                    ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRTC - Branch if the T Flag is Cleared

Description:

Conditional relative branch. Tests the T flag and branches relatively to PC if T is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 6,k).

Operation:

If T = 0 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRTC k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k110

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
bst r3,5        ; Store bit 5 of r3 in T flag
brtc tclear     ; Branch if this bit was cleared
...
tclear:nop      ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRTS - Branch if the T Flag is Set

Description:

Conditional relative branch. Tests the T flag and branches relatively to PC if T is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 6,k).

Operation:

If T = 1 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRTS k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k110

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
bst r3,5        ; Store bit 5 of r3 in T flag
brts tset       ; Branch if this bit was set
...
tset:nop        ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRVC - Branch if Overflow Cleared

Description:

Conditional relative branch. Tests the Overflow flag (V) and branches relatively to PC if V is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 3,k).

Operation:

If V = 0 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRVC k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC PC + 1, if condition is false

16-bit Opcode:

1111 01kk kkkk k011

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
add r3,r4                ; Add r4 to r3
brvc noover              ; Branch if no overflow
...
noover:nop               ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BRVS - Branch if Overflow Set

Description:

Conditional relative branch. Tests the Overflow flag (V) and branches relatively to PC if V is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 3,k).

Operation:

If V = 1 then PC <- PC + k + 1, else PC <- PC + 1

Syntax:	Operands:	Program Counter:
BRVS k	-64 ≤ k ≤ +63	PC <- PC + k + 1
		PC <- PC + 1, if condition is false

16-bit Opcode:

1111 00kk kkkk k011

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
add r3,r4              ; Add r4 to r3
brvs overfl            ; Branch if overflow
...
overfl:nop             ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false*

2 if condition is true

BSET - Bit Set in SREG

Description:

Sets a single flag or bit in SREG.

Operation:

SREG(s) 1

Syntax: Operands: Program Counter:

BSET s 0 ≤ s ≤ 7 PC PC + 1

16-bit Opcode:

1001 0100 0sss 1000

Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
<>	<>	<>	<>	<>	<>	<>	<>

I: 1 if s = 7; Unchanged otherwise.

T: 1 if s = 6; Unchanged otherwise.

H: 1 if s = 5; Unchanged otherwise.

S: 1 if s = 4; Unchanged otherwise.

V: 1 if s = 3; Unchanged otherwise.

N: 1 if s = 2; Unchanged otherwise.

Z: 1 if s = 1; Unchanged otherwise.

C: 1 if s = 0; Unchanged otherwie.

Example:

```
bset  6          ; Set T flag
bset  7          ; Enable interrupt
```

Words: *1 (2 bytes)*

Cycles: *1*

BST - Bit Store from Bit in Register to T Flag in SREG

Description:

Stores bit b from Rd to the T flag in SREG (status register).

Operation:

T <- Rd(b)

Syntax:	Operands:	Program Counter:
BST Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC <- PC + 1

16-bit Opcode:

1111 101d dddd 0bbb

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	<>	-	-	-	-	-	-

T:0 if bit b in Rd is cleared. Set to 1 otherwise.

Example:

```
                        ; Copy bit
bst r1,2                ; Store bit 2 of r1 in T flag
bld r0,4                ; Load T into bit 4 of r0
```

Words: *1 (2 bytes)*

Cycles: *1*

CALL - Long Call to a Subroutine

Description:

Calls to a subroutine within the entire program memory. The return address (to the instruction after the CALL) will be stored onto the stack. (See also RCALL). The stack pointer uses a post-decrement scheme during CALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

PC <- kDevices with 16 bits PC, 128K bytes program memory maximum.

PC <- kDevices with 22 bits PC, 8M bytes program memory maximum.

Syntax:	Operands:	Program Counter	Stack:
CALL k	0 ≤ k < 64K	PC <- k	STACK <- PC+2
		SP <- SP-2, (2 bytes, 16 bits)	
CALL k	0 ≤ k < 4M	PC <- k	STACK <- PC+2
		SP <- SP-3 (3 bytes, 22 bits)	

32-bit Opcode:

1001 010k kkkk 111k
kkkk kkkk kkkk kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
mov r16,r0          ; Copy r0 to r16
call check          ; Call subroutine
nop                 ; Continue (do nothing)
...
check: cpi r16, $42 ; Check if r16 has a special value
```

```
breq error          ; Branch if equal
ret                 ; Return from subroutine
...
error: rjmp error   ; Infinite loop
```

Words: *2 (4 bytes)*

Cycles: *4, devices with 16 bit PC*

5, devices with 22 bit PC

CBI - Clear Bit in I/O Register

Description:

Clears a specified bit in an I/O register. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:

I/O(A,b) <- 0

Syntax:	Operands:	Program Counter:
CBI A, b	0 ≤ A ≤ 31, 0 ≤ b ≤ 7	PC PC + 1

16-bit Opcode:

1001 1000 AAAA Abbb

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
cbi $12, 7          ; Clear bit 7 in Port D
```

Words: *1 (2 bytes)*

Cycles: *2*

CBR - Clear Bits in Register

Description:

Clears the specified bits in register Rd. Performs the logical AND between the contents of register Rd and the complement of the constant mask K. The result will be placed in register Rd.

Operation:

Rd <- Rd ($FF - K)

Syntax: Operands: Program Counter:

CBR Rd,K 16 ≤ d ≤ 31, 0 ≤ K ≤ 255 PC <- PC + 1

16-bit Opcode: *(see ANDI with K complemented)*

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	-

S: N V, For signed tests.

V:0 Cleared

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0* Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
cbr r16, $F0          ; Clear upper nibble of r16
cbr r18, 1            ; Clear bit 0 in r18
```

Words: *1 (2 bytes)*

Cycles: *1*

CLC - Clear Carry Flag

Description:

Clears the Carry flag (C) in SREG (status register).

Operation:

C <- 0

Syntax: Operands: Program Counter:

CLC None PC <- PC + 1

16-bit Opcode:

1001 0100 1000 1000

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - - - 0
C: 0 Carry flag cleared

Example:

```
add r0, r0                ; Add r0 to itself
clc                       ; Clear carry flag
```

Words: *1 (2 bytes)*

Cycles: *1*

CLH - Clear Half Carry Flag

Description:

Clears the Half Carry flag (H) in SREG (status register).

Operation:

(i) H 0

Syntax: Operands: Program Counter:

(i) CLH None PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

H: 0

Half Carry flag cleared

Example:

clh; Clear the Half Carry flag

Words: *1 (2 bytes)*

Cycles: *1*

CLI - Clear Global Interrupt Flag

Description:

Clears the Global Interrupt flag (I) in SREG (status register). The interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction.

Operation:

I <- 0

Syntax: Operands: Program Counter:

CLI None PC <- PC + 1

16-bit Opcode:

1001 0100 1111 1000

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
0 - - - - - - -

I: 0

Global Interrupt flag cleared

Example:

```
in temp, SREG                ; Store SREG value (temp must be defined by user)
cli                          ; Disable interrupts during timed sequence
sbi EECR, EEMWE              ; Start EEPROM write
sbi EECR, EEWE
out SREG, temp               ; Restore SREG value (I-flag)
```

Words: *1 (2 bytes)*

Cycles: *1*

CLN - Clear Negative Flag

Description:

Clears the Negative flag (N) in SREG (status register).

Operation:

N <- 0

Syntax:	Operands:	Program Counter:
CLN	None	PC <- PC + 1

16-bit Opcode:

1001 0100 1010 1000

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - 0 - -
N: 0 Negative flag cleared

Example:

add r2,r3 ; Add r3 to r2

cln ; Clear negative flag

Words: *1 (2 bytes)*

Cycles: *1*

CLR - Clear Register

Description:

Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear all bits in the register.

Operation:

Rd <- Rd [+] Rd

Syntax: Operands: Program Counter:

CLR Rd 0 ≤ d ≤ 31 PC <- PC + 1

16-bit Opcode: *(see EOR Rd,Rd)*

0010 01dd dddd dddd

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - 0 0 0 1 -
S: 0 Cleared

V:0 Cleared

N:0 Cleared

Z:1 Set

R (Result) equals Rd after the operation.

Example:

```
clr r18 ; clear r18
loop: inc r18 ; increase r18
...
cpi r18, $50 ; Compare r18 to $50
brne loop
```

Words: *1 (2 bytes)*

Cycles: *1*

CLS - Clear Signed Flag

Description:

Clears the Signed flag (S) in SREG (status register).

Operation:

S <- 0

Syntax:	Operands:	Program Counter:
CLS	None	PC <- PC + 1

16-bit Opcode:

1001 0100 1100 1000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	0	-	-	-	-

S: 0 Signed flag cleared

Example:

```
add r2,r3          ; Add r3 to r2
cls                ; Clear signed flag
```

Words: *1 (2 bytes)*

Cycles:

CLT - Clear T Flag

Description:

Clears the T flag in SREG (status register).

Operation:

T <- 0

Syntax:	Operands:	Program Counter:
CLT	None	PC <- PC + 1

16-bit Opcode:

1001 0100 1110 1000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	0	-	-	-	-	-	-

T: 0 T flag cleared

Example:

```
clt            ; Clear T flag
```

Words: *1 (2 bytes)*

Cycles: *1*

CLV - Clear Overflow Flag

Description:

Clears the Overflow flag (V) in SREG (status register).

Operation:

V <- 0

Syntax:	Operands:	Program Counter:
CLV	None	PC <- PC + 1

16-bit Opcode:

1001 0100 1011 1000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	0	-	-	-

V: 0 Overflow flag cleared

Example:

```
add r2,r3          ; Add r3 to r2
clv                ; Clear overflow flag
```

Words: *1 (2 bytes)*

Cycles: *1*

CLZ - Clear Zero Flag

Description:

Clears the Zero flag (Z) in SREG (status register).

Operation:

Z <- 0

Syntax: Operands: Program Counter:

CLZ None PC <- PC + 1

16-bit Opcode:

1001 0100 1001 1000

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - - 0 -
Z: 0 Zero flag cleared

Example:

add r2,r3 ; Add r3 to r2

clz; Clear zero

Words: *1 (2 bytes)*

Cycles: *1*

COM - One's Complement

Description:

This instruction performs a one's complement of register Rd.

Operation:

Rd <- $FF - Rd

Syntax: Operands: Program Counter:

COM Rd 0 ≤ d ≤ 31 PC <- PC + 1

16-bit Opcode:

1001 010d dddd 0000
Status Register (SREG) and Boolean Formulae:

I	T	H	S	V	N	Z	C
-	-	-	<>	0	<>	<>	1

S:N [+] V For signed tests.

V:0 Cleared.

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0* Set if the result is $00; Cleared otherwise.

C:1 Set.

R (Result) equals Rd after the operation.

Example:

```
com r4                  ; Take one's complement of r4
breq zero               ; Branch if zero
...
zero:  nop              ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

CP - Compare

Description:

This instruction performs a compare between two registers Rd and Rr. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

Rd - Rr

Syntax:	Operands:	Program Counter:
CP Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC <- PC + 1

16-bit Opcode:

0001 01rd dddd rrrr

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - <> <> <> <> <> <>

H: *Rd3* Rr3+ Rr3 R3 +R3 *Rd3* Set if there was a borrow from bit 3; cleared otherwise

S: N [+] V, For signed tests.

V:Rd7 *Rd7* *R7 + Rd7* Rr7 R7 Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7* *R6* *R5* *R4* *R3* *R2* *R1* *R0* Set if the result is $00; cleared otherwise.

C: *Rd7* Rr7+ Rr7 R7 +R7 *Rd7* Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
cp r4,r19                ; Compare r4 with r19
brne noteq               ; Branch if r4 <> r19
...
noteq: nop               ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

CPC - Compare with Carry

Description:

This instruction performs a compare between two registers Rd and Rr and also takes into account the previous carry. None of the registers are changed. All conditional branches can be used after this instruction.

Operation:

Rd - Rr - C

Syntax: Operands: Program Counter:

CPC Rd,Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31 PC <- PC + 1

16-bit Opcode:

0000 01rd dddd rrrr

Status Register (SREG) and Boolean Formula:

I T H S V N Z C

- - <> <> <> <> <> <>

H: *Rd3* Rr3+ Rr3 R3 +R3 *Rd3* Set if there was a borrow from bit 3; cleared otherwise

S: N V, For signed tests.

V:Rd7 *Rr7* *R7 + Rd7* Rr7 R7 Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7* *R6* *R5* *R4* *R3* *R2* *R1* *R0* Z Previous value remains unchanged when the result is zero; cleared otherwise.

C: *Rd7* Rr7+ Rr7 R7 +R7 *Rd7*

Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
; Compare r3:r2 with r1:r0
cp r2,r0                ; Compare low byte
cpc r3,r1               ; Compare high byte
brne noteq              ; Branch if not equal
...
noteq:nop               ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

CPI - Compare with Immediate

Description:

This instruction performs a compare between register Rd and a constant. The register is not changed. All conditional branches can be used after this instruction.

Operation:

Rd - K

Syntax: Operands: Program Counter:

CPI Rd,K 16 ≤ d ≤ 31, 0≤ K ≤ 255 PC <- PC + 1

16-bit Opcode:

0011 KKKK dddd KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	<>	<>	<>	<>	<>	<>

H: *Rd3* K3+ K3 R3+ R3 *Rd3* Set if there was a borrow from bit 3; cleared otherwise

S: N [+] V, For signed tests.

V:Rd7 *K7* *R7* + *Rd7* K7 R7 Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7 Set if MSB of the result is set; cleared otherwise.

Z: *R7* *R6* *R5* *R4* *R3* *R2* *R1* *R0* Set if the result is $00; cleared otherwise.

C: *Rd7* K7 +K7 R7+ R7 *Rd7* Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

```
cpi r19,3                    ; Compare r19 with 3
brne error                   ; Branch if r19<>3
```

...

error: nop ; Branch destination (do nothing)

Words: *1 (2 bytes)*

Cycles: *1*

CPSE - Compare Skip if Equal

Description:

This instruction performs a compare between two registers Rd and Rr, and skips the next instruction if Rd = Rr.

Operation:

If Rd = Rr then PC <- PC + 2 (or 3) else PC <- PC + 1

Syntax:	Operands:	Program Counter:
CPSE Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC <- PC + 1, Condition false - no skip
		PC <- PC + 2, Skip a one word instruction
		PC PC + 3, Skip a two word instruction

16-bit Opcode:

0001 00rd dddd rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
inc r4          ; Increase r4
cpse r4,r0      ; Compare r4 to r0
neg r4          ; Only executed if r4<>r0
nop             ; Continue (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false (no skip)*

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

DEC - Decrement

Description:

Subtracts one -1- from the contents of register Rd and places the result in the destination register Rd.

The C flag in SREG is not affected by the operation, thus allowing the DEC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned values, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

Rd <- Rd - 1

Syntax:	Operands:	Program Counter:
DEC Rd	0 ≤ d ≤ 31	PC <- PC + 1

16-bit Opcode:

1001 010d dddd 1010

Status Register and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	<>	<>	<>	<>	-

S:N [+] V

For signed tests.

V: *R7* R6 R5 R4 R3 R2 R1 R0

Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $80 before the operation.

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $00; Cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
ldi r17, $10                    ; Load constant in r17
loop:  add r1, r2               ; Add r2 to r1
dec r17                         ; Decrement r17
brne  loop                      ; Branch if r17<>0
nop; Continue (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

EICALL - Extended Indirect Call to Subroutine

Description:

Indirect call of a subroutine pointed to by the Z (16 bits) pointer register in the register file and the EIND register in the I/O space. This instruction allows for indirect calls to the entire program memory space. The stack pointer uses a post-decrement scheme during EICALL.

This instruction is not implemented for devices with 2 bytes PC, see ICALL. Refer to the device specific instruction set summary.

Operation:

PC(15:0) <- Z(15:0)

PC(21:16) <- EIND

Syntax:	Operands:	Program Counter:	Stack:
EICALL	None	See Operation	STACK <- PC + 1
			SP -< SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

1001 0101 0001 1001

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - - - -

Example:

```
ldi r16,$05                    ; Set up EIND and Z pointer
out EIND,r16
ldi 30, $00
ldi r31, $10
eicall                         ; Call to $051000
```

Words: *1 (2 bytes)*

Cycles: *4 (only implemented in devices with 22 bit PC)*

EIJMP - Extended Indirect Jump

Description:

Indirect jump to the address pointed to by the Z (16 bits) pointer register in the register file and the EIND register in the I/O space. This instruction allows for indirect jumps to the entire program memory space.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

PC(15:0) <- Z(15:0)

PC(21:16) <- EIND

Syntax:	Operands:	Program Counter:	Stack:
EIJMP	None	See Operation	Not Affected

16-bit Opcode:

1001 0100 0001 1001

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - - - -

Example:

```
ldi r16, $05                ; Set up EIND and Z pointer
out EIND, r16
ldi r30, $00
ldi r31, $10
eijmp                       ; Jump to $051000
```

Words: *1 (2 bytes)*

Cycles: *2*

ELPM - Extended Load Program Memory

Description:

Loads one byte pointed to by the Z register and the RAMPZ register in the I/O space, and places this byte in the destination register Rd. This instruction features a 100% space effective constant initialization or constant data fetch. The program memory is organized in 16 bit words while the Z pointer is a byte address. Thus, the least significant bit of the Z pointer selects either low byte (ZLSB = 0) or high byte (ZLSB = 1). This instruction can address the entire program memory space. The Z pointer register can either be left unchanged by the operation, or it can be incremented. The incrementation applies to the entire 24-bit concatenation of the RAMPZ and Z pointer registers.

Devices with Self-Programming capability can use the ELPM instruction to read the fuse and lock bit value. Refer to the device documentation for a detailed description.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

```
ELPM r30, Z+
ELPM r31, Z+
```

Operation:	Comment:
(1)R0 <- (RAMPZ:Z)	RAMPZ:Z: Unchanged, R0 implied destination register
(2)Rd <- (RAMPZ:Z)	RAMPZ:Z: Unchanged
(3)Rd <- (RAMPZ:Z) (RAMPZ:Z) <- (RAMPZ:Z) + 1	RAMPZ:Z: Post incremented

Syntax:	Operands:	Program Counter:
(1) ELPM	None, R0 implied	PC <- PC + 1

(2) ELPM Rd, Z 0 ≤ d ≤ 31 PC <- PC + 1

(3) ELPM Rd, Z+ 0 ≤ d ≤ 31 PC <- PC + 1

16 bit Opcode:

(1) 1001 0101 1101 1000
(2) 1001 000d dddd 0110
(3) 1001 000d dddd 0111

Status Register (SREG) and Boolean Formula:

I T H S V N Z C
- - - - - - - -

Example:

```
ldi ZL, byte3(Table_1<<1)          ; Initialize Z pointer

out RAMP Z, ZL

ldi ZH, byte2(Table_1<<1)

ldiZL, byte1(Table_1<<1)

elpm r16, Z+                       ; Load constant from program

; memory pointed to by RAMPZ:Z (Z is r31:r30)

...

Table_1:

.dw 0x3738; 0x38 is addressed when ZLSB = 0

; 0x37 is addressed when ZLSB = 1

...
```

Words: *1 (2 bytes)*

Cycles: *3*

EOR - Exclusive OR

Description:

Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

Rd <- Rd [+] Rr

Syntax: Operands: Program Counter:

EOR Rd,Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31 PC <- PC + 1

16-bit Opcode:

0010 01rd dddd rrrr

Status Register (SREG) and Boolean Formula:

I T H S V N Z C

- - - <> 0 <> <> -

S: N [+] V, For signed tests.

V:0

Cleared

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
eor r4,r4                    ; Clear r4
eor r0,r22                   ; Bitwise exclusive or between r0 and r22
```

Words: *1 (2 bytes)*

Cycles: *1*

FMUL - Fractional Multiply Unsigned

Description:

This instruction performs 8-bit × 8-bit 16-bit unsigned multiplication and shifts the result one bit left.

Rd		Rr		R1	R0
Multiplicand	×	Multiplier	->	Product High	Product Low
8		8		16	

Let (N.Q) denote a fractional number with N binary digits left of the radix point, and Q binary digits right of the radix point. A multiplication between two numbers in the formats (N1.Q1) and (N2.Q2) results in the format ((N1+N2).(Q1+Q2)). For signal processing applications, the format (1.7) is widely used for the inputs, resulting in a (2.14) format for the product. A left shift is required for the high byte of the product to be in the same format as the inputs. The FMUL instruction incorporates the shift operation in the same number of cycles as MUL.

The (1.7) format is most commonly used with signed numbers, while FMUL performs an unsigned multiplication. This instruction is therefore most useful for calculating one of the partial products when performing a signed multiplication with 16-bit inputs in the (1.15) format, yielding a result in the (1.31) format. Note: the result of the FMUL operation may suffer from a 2's complement overflow if interpreted as a number in the (1.15) format. The MSB of the multiplication before shifting must be taken into account, and is found in the carry bit. See the following example.

The multiplicand Rd and the multiplier Rr are two registers containing unsigned fractional numbers where the implicit radix point lies between bit 6 and bit 7. The 16-bit unsigned fractional product with the implicit radix point between bit 14 and bit 15 is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

R1:R0 Rd × Rr(unsigned (1.15) unsigned (1.7) × unsigned (1.7))

Syntax:	Operands:	Program Counter:
FMUL Rd,Rr	16 ≤ d ≤ 23, 16≤ r ≤ 23	PC PC + 1

16-bit Opcode:

0000 0011 0ddd 1rrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	<>	<>

C: R16

Set if bit 15 of the result before left shift is set; cleared otherwise.

Z: R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
;***************************************************************************
;* DESCRIPTION
;*Signed fractional multiply of two 16-bit numbers with 32-bit result.
;* USAGE
;*r19:r18:r17:r16 = ( r23:r22 * r21:r20 ) << 1
;***************************************************************************
fmuls16x16_32:
clr r2
fmuls r23, r21                  ;((signed)ah * (signed)bh) << 1
movw r19:r18, r1:r0
fmul r22, r20                   ;(al * bl) << 1
adc r18, r2
movw r17:r16, r1:r0
fmulsu r23, r20                 ;((signed)ah * bl) << 1
sbc r19, r2
add r17, r0
adc r18, r1
adc r19, r2
fmulsu r21, r22                 ;((signed)bh * al) << 1
sbc r19, r2
add r17, r0
adc r18, r1
```

adc r19, r2

Words: *1 (2 bytes)*

Cycles: *2*

FMULS - Fractional Multiply Signed

Description:

This instruction performs 8-bit × 8-bit 16-bit signed multiplication and shifts the result one bit left.

Let (N.Q) denote a fractional number with N binary digits left of the radix point, and Q binary digits right of the radix point. A multiplication between two numbers in the formats (N1.Q1) and (N2.Q2) results in the format ((N1+N2).(Q1+Q2)). For signal processing applications, the format (1.7) is widely used for the inputs, resulting in a (2.14) format for the product. A left shift is required for the high byte of the product to be in the same format as the inputs. The FMULS instruction incorporates the shift operation in the same number of cycles as MULS.

The multiplicand Rd and the multiplier Rr are two registers containing signed fractional numbers where the implicit radix point lies between bit 6 and bit 7. The 16-bit signed fractional product with the implicit radix point between bit 14 and bit 15 is placed in R1 (high byte) and R0 (low byte).

Note that when multiplying 0x80 (-1) with 0x80 (-1), the result of the shift operation is 0x8000 (-1). The shift operation thus gives a two's complement overflow. This must be checked and handled by software.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)R1:R0 Rd × Rr(signed (1.15) <- signed (1.7) × signed (1.7))

Syntax:	Operands:	Program Counter:
FMULS Rd,Rr	16 ≤ d ≤ 23, 16≤ r ≤ 23	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formulae:

C: R16

Set if bit 15 of the result before left shift is set; cleared otherwise.

Z: R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
fmuls r23,r22                ; Multiply signed r23 and r22 in (1.7) format, result in (1.15) format
movw r23:r22,r1:r0           ; Copy result back in r23:r22
```

Words: *1 (2 bytes)*

Cycles: *2*

FMULSU - Fractional Multiply Signed with Unsigned

Description:

This instruction performs 8-bit × 8-bit 16-bit signed multiplication and shifts the result one bit left.

Let (N.Q) denote a fractional number with N binary digits left of the radix point, and Q binary digits right of the radix point. A multiplication between two numbers in the formats (N1.Q1) and (N2.Q2) results in the format ((N1+N2).(Q1+Q2)). For signal processing applications, the format (1.7) is widely used for the inputs, resulting in a (2.14) format for the product. A left shift is required for the high byte of the product to be in the same format as the inputs. The FMULSU instruction incorporates the shift operation in the same number of cycles as MULSU.

The (1.7) format is most commonly used with signed numbers, while FMULSU performs a multiplication with one unsigned and one signed input. This instruction is therefore most useful for calculating two of the partial products when performing a signed multiplication with 16-bit inputs in the (1.15) format, yielding a result in the (1.31) format. Note: the result of the FMULSU operation may suffer from a 2's complement overflow if interpreted as a number in the (1.15) format. The MSB of the multiplication before shifting must be taken into account, and is found in the carry bit. See the following example.

The multiplicand Rd and the multiplier Rr are two registers containing fractional numbers where the implicit radix point lies between bit 6 and bit 7. The multiplicand Rd is a signed fractional number, and the multiplier Rr is an unsigned fractional number. The 16-bit signed fractional product with the implicit radix point between bit 14 and bit 15 is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

R1:R0 <- Rd × Rr(signed (1.15) signed (1.7) × unsigned (1.7))

Syntax: Operands: Program Counter:

FMULSU Rd,Rr 16 ≤ d ≤ 23, 16≤ r ≤ 23 PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formulae:

C: R16

Set if bit 15 of the result before left shift is set; cleared otherwise.

Z: R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
;*****************************************************************************
;* DESCRIPTION
;*Signed fractional multiply of two 16-bit numbers with 32-bit result.
;* USAGE
;*r19:r18:r17:r16 = ( r23:r22 * r21:r20 ) << 1
;*****************************************************************************
fmuls16x16_32:
clr r2
fmuls r23, r21                  ;((signed)ah * (signed)bh) << 1
movw r19:r18, r1:r0
```

```
fmul r22, r20                ;(al * bl) << 1

adc r18, r2

movw r17:r16, r1:r0

fmulsu r23, r20              ;((signed)ah * bl) << 1

sbc r19, r2

add r17, r0

adc r18, r1

adc r19, r2

fmulsu r21, r22;((signed)bh * al) << 1

sbc r19, r2

add r17, r0

adc r18, r1

adc r19, r2
```

Words: *1 (2 bytes)*

Cycles: *2*

ICALL - Indirect Call to Subroutine

Description:

Indirect call of a subroutine pointed to by the Z (16 bits) pointer register in the register file. The Z pointer register is 16 bits wide and allows call to a subroutine within the lowest 64K words (128K bytes) section in the program memory space. The stack pointer uses a post-decrement scheme during ICALL.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) PC(15:0) <- Z(15:0)Devices with 16 bits PC, 128K bytes program memory maximum.

(ii) PC(15:0) <- Z(15:0)Devices with 22 bits PC, 8M bytes program memory maximum.

PC(21:16) <- 0

Syntax:	Operands:	Program Counter:	Stack:
(i) ICALL	None	See Operation	STACK <- PC + 1
			SP <- SP - 2 (2 bytes, 16 bits)
(ii)ICALL	None	See Operation	STACK <- PC + 1
			SP <- SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
mov r30,r0      ; Set offset to call table
icall           ; Call routine pointed to by r31:r30
```

Words: *1 (2 bytes)*

Cycles: *3 devices with 16 bit PC*

4 devices with 22 bit PC

IJMP - Indirect Jump

Description:

Indirect jump to the address pointed to by the Z (16 bits) pointer register in the register file. The Z pointer register is 16 bits wide and allows jump within the lowest 64K words (128K bytes) section of program memory.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) PC <- Z(15:0) Devices with 16 bits PC, 128K bytes program memory maximum.

(ii) PC(15:0) <- Z(15:0)Devices with 22 bits PC, 8M bytes program memory maximum.

PC(21:16) <- 0

Syntax:	Operands:	Program Counter:	Stack:

(i), (ii) IJMP None See Operation Not Affected

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
mov r30,r0     ; Set offset to jump table
ijmp           ; Jump to routine pointed to by r31:r30
```

Words: *1 (2 bytes)*

Cycles: *2*

IN - Load an I/O Location to Register

Description:

Loads data from the I/O Space (Ports, Timers, Configuration registers etc.) into register Rd in the register file.

Operation:

(i)Rd <- I/O(A)

Syntax:	Operands:	Program Counter:
IN Rd,A	0 ≤ d ≤ 31, 0 ≤ A ≤ 63	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
in r25,$16                ; Read Port B
cpi r25,4                 ; Compare read value to constant
breqexit                  ; Branch if r25=4
...
```

```
exit:   nop                     ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

INC - Increment

Description:

Adds one -1- to the contents of register Rd and places the result in the destination register Rd.

The C flag in SREG is not affected by the operation, thus allowing the INC instruction to be used on a loop counter in multiple-precision computations.

When operating on unsigned numbers, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:

(i) Rd <- Rd + 1

Syntax:	Operands:	Program Counter:
(i) INC Rd	0 ≤ d ≤ 31	PC <- PC + 1

16-bit Opcode:

Status Register and Boolean Formula:

S:N [+] V

For signed tests.

V:R7 *R6 R5 R4 R3 R2 R1 R0*

Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $7F before the operation.

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $00; Cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
clr r22                                 ; clear r22
loop:   inc r22                                 ; increment r22
...
cpi r22,$4F                              ; Compare r22 to $4f
brne   loop                                     ; Branch if not equal
nop                                     ; Continue (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

JMP - Jump

Description:

Jump to an address within the entire 4M (words) program memory. See also RJMP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)PC <- k

Syntax:	Operands:	Program Counter:	Stack:
(i)JMP k	0 ≤ k < 4M	PC <- k	Unchanged

32-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
mov r1,r0            ; Copy r0 to r1
jmp farplc           ; Unconditional jump
...
farplc:nop                   ; Jump destination (do nothing)
```

Words: *2 (4 bytes)*

Cycles: *3*

LD - Load Indirect from data space to Register using Index X

Description:

Loads one byte indirect from the data space to a register. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

The data location is pointed to by the X (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPX in register in the I/O area has to be changed.

The X pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the X pointer register. Note that only the low byte of the X pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPX register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes program memory, and the increment/decrement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

LD r26, X+

LD r27, X+

LD r26, -X

LD r27, -X

Using the X pointer:

Operation:		Comment:
(i)Rd <- (X)		X: Unchanged
(ii)Rd <- (X)	X <- X + 1	X: Post incremented
(iii)X <- X - 1	Rd <- (X)	X: Pre decremented

Syntax:	Operands:	Program Counter:
(i) LD Rd, X	0 ≤ d ≤ 31	PC <- PC + 1
(ii)LD Rd, X+	0 ≤ d ≤ 31	PC <- PC + 1
(iii)LD Rd, -X	0 ≤ d ≤ 31	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clr r27          ; Clear X high byte
ldi r26,$60      ; Set X low byte to $60
ld r0,X+         ; Load r0 with data space loc. $60(X post inc)
ld r1,X          ; Load r1 with data space loc. $61
ldi r26,$63      ; Set X low byte to $63
ld r2,X          ; Load r2 with data space loc. $63
ld r3,-X         ; Load r3 with data space loc. $62(X pre dec)
```

Words: *1 (2 bytes)*

Cycles: *2*

LD (LDD) - Load Indirect from data space to Register using Index Y

Description:

Loads one byte indirect with or without displacement from the data space to a register. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

The data location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPY in register in the I/O area has to be changed.

The Y pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the Y pointer register. Note that only the low byte of the Y pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPY register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

LD r28, Y+

LD r29, Y+

LD r28, -Y

LD r29, -Y

Using the Y pointer:

Operation:		Comment:
(i) Rd <- (Y)		Y: Unchanged
(ii) Rd <- (Y)	Y <- Y + 1	Y: Post incremented
(iii) Y <- Y - 1	Rd <- (Y)	Y: Pre decremented
(iiii) Rd <- (Y+q)		Y: Unchanged, q: Displacement

Syntax:	Operands:	Program Counter:
(i) LD Rd, Y	0 ≤ d ≤ 31	PC <- PC + 1
(ii) LD Rd, Y+	0 ≤ d ≤ 31	PC <- PC + 1
(iii) LD Rd, -Y	0 ≤ d ≤ 31	PC <- PC + 1
(iiii)LDD Rd, Y+q	0 ≤ d ≤ 31, 0 ≤ q ≤ 63	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clr r29                    ; Clear Y high byte
ldi r28,$60                ; Set Y low byte to $60
ld r0,Y+                   ; Load r0 with data space loc. $60(Y post inc)
ld r1,Y                    ; Load r1 with data space loc. $61
ldi r28,$63                ; Set Y low byte to $63
ld r2,Y                    ; Load r2 with data space loc. $63
ld r3,-Y                   ; Load r3 with data space loc. $62(Y pre dec)
ldd r4,Y+2                 ; Load r4 with data space loc. $64
```

Words: *1 (2 bytes)*

Cycles: *2*

LD (LDD) - Load Indirect From data space to Register using Index Z

Description:

Loads one byte indirect with or without displacement from the data space to a register. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.

The Z pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for stack pointer usage of the Z pointer register, however because the Z pointer register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y pointer as a dedicated stack pointer. Note that only the low byte of the Z pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

For using the Z pointer for table lookup in program memory see the LPM and ELPM instructions.

The result of these combinations is undefined:

LD r30, Z+

LD r31, Z+

LD r30, -Z

LD r31, -Z

Using the Z pointer:

Operation:	Comment:
(i)Rd <- (Z)	Z: Unchanged
(ii)Rd <- (Z)	Z <- Z + 1Z: Post increment
(iii)Z <- Z -1 Rd <- (Z)	Z: Pre decrement
(iiii) Rd <- (Z+q)	Z: Unchanged, q: Displacement

Syntax:	Operands:	Program Counter:
(i) LD Rd, Z	0 ≤ d ≤ 31	PC <- PC + 1
(ii) LD Rd, Z+	0 ≤ d ≤ 31	PC <- PC + 1
(iii) LD Rd, -Z	0 ≤ d ≤ 31	PC <- PC + 1
(iiii)LDD Rd, Z+q	0 ≤ d ≤ 31, 0 ≤ q ≤ 63	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clr r31                    ; Clear Z high byte
ldi r30,$60                ; Set Z low byte to $60
ld r0,Z+                   ; Load r0 with data space loc. $60(Z post inc)
ld r1,Z                    ; Load r1 with data space loc. $61
ldi r30,$63                ; Set Z low byte to $63
ld r2,Z                    ; Load r2 with data space loc. $63
ld r3,-Z                   ; Load r3 with data space loc. $62(Z pre dec)
ldd r4,Z+2                 ; Load r4 with data space loc. $64
```

Words: *1 (2 bytes)*

Cycles: *2*

LDI - Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i)Rd <- K

Syntax:	Operands:	Program Counter:
(i) LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clrr31          ; Clear Z high byte
ldi r30, $F0    ; Set Z low byte to $F0
lpm             ; Load constant from program
                ; memory pointed to by Z
```

Words: *1 (2 bytes)*

Cycles: *1*

LDS - Load Direct from data space

Description:

Loads one byte from the data space to a register. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The LDS instruction uses the RAMPD register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)Rd <- (k)

Syntax:	Operands:	Program Counter:
(i)LDS Rd,k	0 ≤ d ≤ 31, 0 ≤ k ≤ 65535	PC <- PC + 2

32-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
lds r2,$FF00        ; Load r2 with the contents of data space location $FF00
add r2,r1           ; add r1 to r2
```

```
sts $FF00,r2            ; Write back
```

Words: *2 (4 bytes)*

Cycles: *2*

LPM - Load Program Memory

Description:

Loads one byte pointed to by the Z register into the destination register Rd. This instruction features a 100% space effective constant initialization or constant data fetch. The program memory is organized in 16 bit words while the Z pointer is a byte address. Thus, the least significant bit of the Z pointer selects either low byte (ZLSB = 0) or high byte (ZLSB = 1). This instruction can address the first 64K bytes (32K words) of program memory. The Z pointer register can either be left unchanged by the operation, or it can be incremented. The incrementation does not apply to the RAMPZ register.

Devices with Self-Programming capability can use the LPM instruction to read the fuse and lock bit values. Refer to the device documentation for a detailed description.

Not all variants of the LPM instruction are available in all devices. Refer to the device specific instruction set summary. The LPM instruction is not implemented at all in the AT90S1200 device.

The result of these combinations is undefined:

```
LPM r30, Z+

LPM r31, Z+
```

Operation:	Comment:
(i) R0 <- (Z)	Z: Unchanged, R0 implied destination register
(ii) Rd <- (Z)	Z: Unchanged
(iii) Rd <- (Z)	Z <- Z + 1Z: Post incremented

Syntax:	Operands:	Program Counter:
(i) LPM	None, R0 implied	PC <- PC + 1
(ii)LPM Rd, Z	0 ≤ d ≤ 31	PC <- PC + 1
(iii)LPM Rd, Z+	0 ≤ d ≤ 31	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
ldi ZH, high(Table_1<<1)          ; Initialize Z pointer
ldi ZL, low(Table_1<<1)
lpm r16, Z                        ; Load constant from program
; memory pointed to by Z (r31:r30)
...
Table_1:
.dw 0x5876; 0x76 is addresses when ZLSB = 0
; 0x58 is addresses when ZLSB = 1
...
```

Words: *1 (2 bytes)*

Cycles: *3*

LSL - Logical Shift Left

Description:

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C flag of the SREG. This operation effectively multiplies signed and unsigned values by two.

Operation:

Syntax:	Operands:	Program Counter:
(i) LSL Rd	0 ≤ d ≤ 31	PC <- PC + 1

16-bit Opcode: (see ADD Rd,Rd)

Status Register (SREG) and Boolean Formula:

H: Rd3

S: N [+] V, For signed tests.

V: N [+] C (For N and C after the shift)

N: R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

C: Rd7

Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add r0, r4      ; Add r4 to r0
lsl r0          ; Multiply r0 by 2
```

Words: *1 (2 bytes)*

Cycles: *1*

LSR - Logical Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C flag of the SREG. This operation effectively divides an unsigned value by two. The C flag can be used to round the result.

Operation:

Syntax: Operands: Program Counter:

(i) LSR Rd 0 ≤ d ≤ 31 PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

S: N [+] V, For signed tests.

V: N [+] C (For N and C after the shift)

N: 0

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

C: Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add r0,r4          ; Add r4 to r0
lsr r0             ; Divide r0 by 2
```

Words: *1 (2 bytes)*

Cycles: *1*

MOV - Copy Register

Description:

This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the destination register Rd is loaded with a copy of Rr.

Operation:

(i) Rd <- Rr

Syntax:	Operands:	Program Counter:
(i) MOV Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
mov r16,r0    ; Copy r0 to r16
call check    ; Call subroutine
...
check: cpi r16,$11    ; Compare r16 to $11
...
ret           ; Return from subroutine
```

Words: *1 (2 bytes)*

Cycles: *1*

"

MOVW - Copy Register Word

Description:

This instruction makes a copy of one register pair into another register pair. The source register pair Rr+1:Rr is left unchanged, while the destination register pair Rd+1:Rd is loaded with a copy of Rr + 1:Rr.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd <- Rr+1:Rr

Syntax:	Operands:	Program Counter:
(i) MOVW Rd+1:Rd,Rr+1Rr	d E {0,2,...,30}, r E {0,2,...,30}	PC <- PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
movw r17:16,r1:r0          ; Copy r1:r0 to r17:r16
call check                 ; Call subroutine
...
check: cpi r16,$11         ; Compare r16 to $11
...
cpi r17,$32                ; Compare r17 to $32
...
ret                        ; Return from subroutine
```

Words: *1 (2 bytes)*

Cycles: *1*

MUL - Multiply Unsigned

Description:

This instruction performs 8-bit × 8-bit 16-bit unsigned multiplication.

The multiplicand Rd and the multiplier Rr are two registers containing unsigned numbers. The 16-bit unsigned product is placed in R1 (high byte) and R0 (low byte). Note that if the multiplicand or the multiplier is selected from R0 or R1 the result will overwrite those after multiplication.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 <- Rd × Rr(unsigned <- unsigned × unsigned)

Syntax: Operands: Program Counter:

(i) MUL Rd,Rr $0 \le d \le 31, 0 \le r \le 31$ PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formulae:

C: R15

Set if bit 15 of the result is set; cleared otherwise.

Z: R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
mul r5,r4               ; Multiply unsigned r5 and r4
movw r4,r0              ; Copy result back in r5:r4
```

Words: *1 (2 bytes)*

Cycles: *2*

MULS - Multiply Signed

Description:

This instruction performs 8-bit × 8-bit 16-bit signed multiplication.

The multiplicand Rd and the multiplier Rr are two registers containing signed numbers. The 16-bit signed product is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)R1:R0 Rd × Rr(signed signed × signed)

Syntax: Operands: Program Counter:

(i)MULS Rd,Rr 16 ≤ d ≤ 31, 16 ≤ r ≤ 31 PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

C: R15

Set if bit 15 of the result is set; cleared otherwise.

Z: R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
muls r21,r20      ; Multiply signed r21 and r20
movw r20,r0       ; Copy result back in r21:r20
```

Words: *1 (2 bytes)*

Cycles: *2*

MULSU - Multiply Signed with Unsigned

Description:

This instruction performs 8-bit × 8-bit 16-bit multiplication of a signed and an unsigned number.

The multiplicand Rd and the multiplier Rr are two registers. The multiplicand Rd is a signed number, and the multiplier Rr is unsigned. The 16-bit signed product is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 Rd × Rr(signed signed × unsigned)

Syntax: Operands: Program Counter:

(i) MULSU Rd,R r16 ≤ d ≤ 23, 16 ≤ r ≤ 23 PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

C: R15

Set if bit 15 of the result is set; cleared otherwise.

Z: R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
;***************************************************************************
;* DESCRIPTION
;*Signed multiply of two 16-bit numbers with 32-bit result.
;* USAGE
;*r19:r18:r17:r16 = r23:r22 * r21:r20
;***************************************************************************
muls16x16_32:
clr r2
muls r23, r21               ; (signed)ah * (signed)bh
movw r19:r18, r1:r0
mul r22, r20; al * bl
movw r17:r16, r1:r0
mulsu r23, r20              ; (signed)ah * bl
sbc r19, r2
add r17, r0
adc r18, r1
adc r19, r2
mulsu r21, r22              ; (signed)bh * al
sbc r19, r2
add r17, r0
adc r18, r1
adc r19, r2
ret
```

Words: *1 (2 bytes)*

Cycles: *2*

NEG - Two's Complement

Description:

Replaces the contents of register Rd with its two's complement; the value $80 is left unchanged.

Operation:

(i)Rd $00 - Rd

Syntax: Operands: Program Counter:

(i) NEG Rd 0 ≤ d ≤ 31 PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

H:R3 + *Rd3*

Set if there was a borrow from bit 3; cleared otherwise

S:N [+] V

For signed tests.

V:R7 *R6* *R5* *R4* *R3* *R2* *R1* *R0*

Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. A two's complement overflow will occur if and only if the contents of the Register after operation (Result) is $80.

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; Cleared otherwise.

C:R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0

Set if there is a borrow in the implied subtraction from zero; cleared otherwise. The C flag will be set in all cases except when the contents of Register after operation is $00.

R (Result) equals Rd after the operation.

Example:

```
sub r11,r0    ; Subtract r0 from r11
brpl positive ; Branch if result positive
neg r11       ; Take two's complement of r11
positive:  nop          ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

NOP - No Operation

Description:

This instruction performs a single cycle No Operation.

Operation:

(i)No

Syntax: Operands: Program Counter:

(i) NOP None PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clrr16          ; Clear r16
serr17          ; Set r17
out$18,r16      ; Write zeros to Port B
nop             ; Wait (do nothing)
out$18,r17      ; Write ones to Port B
```

Words: *1 (2 bytes)*

Cycles: *1*

OR - Logical OR

Description:

Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i)Rd Rd v Rr

Syntax:Operands:Program Counter:

(i)OR Rd,Rr0 ≤ d ≤ 31, 0 ≤ r ≤ 31PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

S: N [+] V, For signed tests.

V:0

Cleared

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
or r15,r16    ; Do bitwise or between registers
bst r15,6     ; Store bit 6 of r15 in T flag
brts ok       ; Branch if T flag set
...
ok:   nop           ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

ORI - Logical OR with Immediate

Description:

Performs the logical OR between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i)Rd Rd v K

Syntax: Operands: Program Counter:

(i)ORI Rd,K 16 ≤ d ≤ 31, 0 ≤ K ≤ 255 PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

S: N [+] V, For signed tests.

V:0

Cleared

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: R7 R6 R5 R4 R3 R2 R1 R0

Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

ori r16,$F0 ; Set high nibble of r16

```
ori r17,1        ; Set bit 0 of r17
```

Words: *1 (2 bytes)*

Cycles: *1*

OUT - Store Register to I/O Location

Description:

Stores data from register Rr in the register file to I/O Space (Ports, Timers, Configuration registers etc.).

Operation:

(i)I/O(A) Rr

Syntax: Operands: Program Counter:

(i) OUT A,Rr $0 \leq r \leq 31, 0 \leq A \leq 63$ PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clr r16         ; Clear r16
ser r17         ; Set r17
out $18,r16     ; Write zeros to Port B
nop             ; Wait (do nothing)
out $18,r17     ; Write ones to Port B
```

Words: *1 (2 bytes)*

Cycles: *1*

POP - Pop Register from Stack

Description:

This instruction loads register Rd with a byte from the STACK. The stack pointer is pre-incremented by 1 before the POP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)Rd STACK

Syntax:	Operands:	Program Counter:	Stack:
(i) POP Rd	0 ≤ d ≤ 31	PC PC + 1	SP SP + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
callroutine             ; Call subroutine
...
routine:   pushr14      ; Save r14 on the stack
push r13      ; Save r13 on the stack
...
pop r13       ; Restore r13
pop r14       ; Restore r14
ret           ; Return from subroutine
```

Words: *1 (2 bytes)*

Cycles: *2*

PUSH - Push Register on Stack

Description:

This instruction stores the contents of register Rr on the STACK. The stack pointer is post-decremented by 1 after the PUSH.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) STACK Rr

Syntax: Operands: Program Counter: Stack:

(i) PUSH Rr 0 ≤ r ≤ 31 PC PC + 1 SP SP - 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
callroutine          ; Call subroutine
...
routine:   push r14   ; Save r14 on the stack
push r13   ; Save r13 on the stack
...
pop r13    ; Restore r13
pop r14    ; Restore r14
ret        ; Return from subroutine
```

Words: *1 (2 bytes)*

Cycles: *2*

"

RCALL - Relative Call to Subroutine

Description:

Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the stack. (See also CALL). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. The stack pointer uses a post-decrement scheme during RCALL.

Operation:

(i)PC PC + k + 1 Devices with 16 bits PC, 128K bytes program memory maximum.

(ii)PC PC + k + 1 Devices with 22 bits PC, 8M bytes program memory maximum.

Syntax:	Operands:	Program Counter:	Stack:
(i) RCALL k	-2K ≤ k < 2K	PC PC + k + 1	STACK PC + 1
		SP SP - 2 (2 bytes, 16 bits)	
(ii)RCALL k	-2K ≤ k < 2K	PC PC + k + 1	STACK PC + 1
		SP SP - 3 (3 bytes, 22 bits)	

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
rcallroutine          ; Call subroutine
...
routine:  push r14    ; Save r14 on the stack
...
pop r14     ; Restore r14
ret         ; Return from subroutine
```

Words: *1 (2 bytes)*

Cycles: *3 devices with 16-bit PC*

4 devices with 22-bit PC

RET - Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK. The stack pointer uses a pre-increment scheme during RET.

Operation:

(i)PC(15:0) STACK Devices with 16 bits PC, 128K bytes program memory maximum.

(ii)PC(21:0) STACK Devices with 22 bits PC, 8M bytes program memory maximum.

Syntax:	Operands:	Program Counter:	Stack:
(i) RET	None	See Operation	SP SP + 2, (2bytes,16 bits)
(ii) RET	None	See Operation	SP SP + 3, (3bytes,22 bits)

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
callroutine          ; Call subroutine
...
routine:  push r14   ; Save r14 on the stack
...
pop r14              ; Restore r14
ret                  ; Return from subroutine
```

Words: *1 (2 bytes)*

Cycles: *4 devices with 16-bit PC*

5 devices with 22-bit PC

RETI - Return from Interrupt

Description:

Returns from interrupt. The return address is loaded from the STACK and the global interrupt flag is set.

Note that the status register is not automatically stored when entering an interrupt routine, and it is not restored when returning from an interrupt routine. This must be handled by the application program. The stack pointer uses a pre-increment scheme during RETI.

Operation:

(i)PC(15:0) STACKDevices with 16 bits PC, 128K bytes program memory maximum.

(ii)PC(21:0) STACKDevices with 22 bits PC, 8M bytes program memory maximum.

Syntax:	Operands:	Program Counter:	Stack
(i) RETI	None	See Operation	SP SP + 2 (2 bytes, 16 bits)
(ii) RETI	None	See Operation	SP SP + 3 (3 bytes, 22 bits)

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

I: 1

The I flag is set.

Example:

```
...
extint:  push   r0    ; Save r0 on the stack
...
pop   r0    ; Restore r0
```

```
reti            ; Return and enable interrupts
```

Words: *1 (2 bytes)*

Cycles: *4 devices with 16-bit PC*

5 devices with 22-bit PC

RJMP - Relative Jump

Description:

Relative jump to an address within PC - 2K +1 and PC + 2K (words). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location.

Operation:

(i)PC PC + k + 1

Syntax:	Operands:	Program Counter:	Stack
(i) RJMP k	-2K ≤ k < 2K	PC PC + k + 1	Unchanged

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
cpir16,$42              ; Compare r16 to $42
brneerror               ; Branch if r16 <> $42
rjmpok                  ; Unconditional branch
error:addr16,r17                ; Add r17 to r16
incr16                  ; Increment r16
ok:   nop                       ; Destination for rjmp (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *2*

ROL - Rotate Left trough Carry

Description:

Shifts all bits in Rd one place to the left. The C flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C flag. This operation, combined with LSL, effectively multiplies multi-byte signed and unsigned values by two.

Operation:
Syntax: Operands: Program Counter:

(i)ROL Rd $0 \leq d \leq 31$ PC PC + 1

16-bit Opcode: *(see ADC Rd,Rd)*

Status Register (SREG) and Boolean Formula:

H:Rd3

S: N [+] V, For signed tests.

V:N [+] C (For N and C after the shift)

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

C:Rd7

Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
lsl r18          ; Multiply r19:r18 by two
rol r19          ; r19:r18 is a signed or unsigned two-byte integer
brcsoneenc       ; Branch if carry set
...
oneenc:nop       ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

ROR - Rotate Right through Carry

Description:

Shifts all bits in Rd one place to the right. The C flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR it effectively divides multi-byte unsigned values by two. The carry flag can be used to round the result.

Operation:

Syntax:	Operands:	Program Counter:
(i) ROR Rd	0 ≤ d ≤ 31	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

S: N [+] V, For signed tests.

V:N [+] C (For N and C after the shift)

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

C:Rd0

Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
lsr r19              ; Divide r19:r18 by two
ror r18              ; r19:r18 is an unsigned two-byte integer
brcc zeroenc1        ; Branch if carry cleared
asr r17              ; Divide r17:r16 by two
ror r16              ; r17:r16 is a signed two-byte integer
brcc zeroenc2        ; Branch if carry cleared
...
zeroenc1:nop             ; Branch destination (do nothing)
...
zeroenc2:nop             ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

SBC - Subtract with Carry

Description:

Subtracts two registers and subtracts with the C flag and places the result in the destination register Rd.

Operation:

(i) Rd Rd - Rr - C

Syntax: Operands: Program Counter:

(i) SBC Rd,Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31 PC PC + 1

16-bit Opcode:

Status Register and Boolean Formula:

H: *Rd3* Rr3 + Rr3 R3 + R3 *Rd3*

Set if there was a borrow from bit 3; cleared otherwise

S: N [+] V, For signed tests.

V:Rd7 *Rr7* *R7* + *Rd7* Rr7 R7

Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7* *R6* *R5* *R4* *R3* *R2* *R1* *R0* Z

Previous value remains unchanged when the result is zero; cleared otherwise.

C: *Rd7* Rr7+ Rr7 R7 +R7 *Rd7*

Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of the Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
        ; Subtract r1:r0 from r3:r2
sub r2,r0       ; Subtract low byte
sbc r3,r1       ; Subtract with carry high byte
```

Words: *1 (2 bytes)*

Cycles: *1*

SBR - Set Bits in Register

Description:

Sets specified bits in register Rd. Performs the logical ORI between the contents of register Rd and constant mask K and places the result in the destination register Rd.

Operation:

(i)Rd <- Rd v K

Syntax:	Operands:	Program Counter:
(i)SBR Rd,K	16<=d<=31, 0<=K<=255	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

S: N V, For signed tests.

V:0

Cleared

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *'R7 'R6 'R5 'R4 'R3 'R2 'R1 'R0*

R (Result) equals Rd after the operation.

Example:

```
sbr   r16,3      ; Set bits 0 and 1 in r16
sbr   r17,$F0    ; Set 4 MSB in r17
```

Words: *1 (2 bytes)*

Cycles: *1*

SBRC - Skip if Bit in Register is Cleared

Description:

This instruction tests a single bit in a register and skips the next instruction if the bit is cleared.

Operation:

(i)If Rr(b) = 0 then PC PC + 2 (or 3) else PC PC + 1

Syntax: Operands: Program Counter:

(i)SBRC Rr,b 0 ≤ r ≤ 31, 0 ≤ b ≤ 7 PC PC + 1, Condition false - no skip

PC PC + 2, Skip a one word instruction

PC PC + 3, Skip a two word instruction

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
sub r0,r1          ; Subtract r1 from r0
sbrc r0,7          ; Skip if bit 7 in r0 cleared
sub r0,r1          ; Only executed if bit 7 in r0 not cleared
nop                ; Continue (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false (no skip)*

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

SBRS - Skip if Bit in Register is Set

Description:

This instruction tests a single bit in a register and skips the next instruction if the bit is set.

Operation:

(i)If Rr(b) = 1 then PC PC + 2 (or 3) else PC PC + 1

Syntax:	Operands:	Program Counter:
(i)SBRS Rr,b	0 ≤ r ≤ 31, 0 ≤ b ≤ 7	PC PC + 1, Condition false - no skip
		PC PC + 2, Skip a one word instruction
		PC PC + 3, Skip a two word instruction

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
sub   r0,r1    ; Subtract r1 from r0
sbrs  r0,7     ; Skip if bit 7 in r0 set
neg   r0       ; Only executed if bit 7 in r0 not set
nop            ; Continue (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false (no skip)*

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

SEC - Set Carry Flag

Description:

Sets the Carry flag (C) in SREG (status register).

Operation:

(i)C 1

Syntax:	Operands:	Program Counter:
(i)SEC	None	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

C: 1

Carry flag set

Example:

```
sec         ; Set carry flag
adc r0,r1   ; r0=r0+r1+1
```

Words: *1 (2 bytes)*

Cycles: *1*

SEH - Set Half Carry Flag

Description:

Sets the Half Carry (H) in SREG (status register).

Operation:

(i)H 1

Syntax:	Operands:	Program Counter:
(i) SEH	None	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

H: 1

Half Carry flag set

Example:

```
seh       ; Set Half Carry flag
```

Words: *1 (2 bytes)*

Cycles: *1*

SBI - Set Bit in I/O Register

Description:

Sets a specified bit in an I/O register. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:

(i)I/O(A,b) 1

Syntax: Operands: Program Counter:

(i)SBI A,b $0 \leq A \leq 31, 0 \leq b \leq 7$ PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
out  $1E,r0   ; Write EEPROM address
sbi  $1C,0    ; Set read bit in EECR
in   r1,$1D   ; Read EEPROM data
```

Words: *1 (2 bytes)*

Cycles: *2*

SBCI - Subtract Immediate with Carry

Description:

Subtracts a constant from a register and subtracts with the C flag and places the result in the destination register Rd.

Operation:

(i)Rd Rd - K - C

Syntax: Operands: Program Counter:

(i) SBCI Rd,K 16 ≤ d ≤ 31, 0 ≤ K ≤ 255 PC PC + 1

16-bit Opcode:

Status Register and Boolean Formula:

H: *Rd3* K3 + K3 R3 + R3 *Rd3*

Set if there was a borrow from bit 3; cleared otherwise

S: N [+] V, For signed tests.

V:Rd7 *K7* *R7* + *Rd7* K7 R7

Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7* *R6* *R5* *R4* *R3* *R2* *R1* *R0* Z

Previous value remains unchanged when the result is zero; cleared otherwise.

C: *Rd7* K7+ K7 R7 +R7 *Rd7*

Set if the absolute value of the constant plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

; Subtract $4F23 from r17:r16

subi r16,$23 ; Subtract low byte

sbci r17,$4F ; Subtract with carry high byte

Words: *1 (2 bytes)*

Cycles: *1*

SBIC - Skip if Bit in I/O Register is Cleared

Description:

This instruction tests a single bit in an I/O register and skips the next instruction if the bit is cleared. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:

(i)If I/O(A,b) = 0 then PC PC + 2 (or 3) else PC PC + 1

Syntax: Operands: Program Counter:

(i)SBIC A,b 0 ≤ A ≤ 31, 0 ≤ b ≤ 7 PC PC + 1, Condition false - no skip

PC PC + 2, Skip a one word instruction

PC PC + 3, Skip a two word instruction

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

e2wait: sbic$1C,1 ; Skip next inst. if EEWE cleared

rjmp e2wait ; EEPROM write not finished

nop; Continue (do nothing)

Words: *1 (2 bytes)*

Cycles: *1 if condition is false (no skip)*

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

SBIS - Skip if Bit in I/O Register is Set

Description:

This instruction tests a single bit in an I/O register and skips the next instruction if the bit is set. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:

(i)If I/O(A,b) = 1 then PC PC + 2 (or 3) else PC PC + 1

Syntax: Operands: Program Counter:

(i)SBIS A,b 0 ≤ A ≤ 31, 0 ≤ b ≤ 7 PC PC + 1, Condition false - no skip

PC PC + 2, Skip a one word instruction

PC PC + 3, Skip a two word instruction

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
waitset:   sbis $10,0        ; Skip next inst. if bit 0 in Port D set
rjmp waitset     ; Bit not set
nop              ; Continue (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1 if condition is false (no skip)*

2 if condition is true (skip is executed) and the instruction skipped is 1 word

3 if condition is true (skip is executed) and the instruction skipped is 2 words

SBIW - Subtract Immediate from Word

Description:

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)Rd+1:Rd Rd+1:Rd - K

Syntax:	Operands:	Program Counter:
(i)SBIW Rd+1:Rd,K	d E {24,26,28,30}, 0 ≤ K ≤ 63	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

S: N V, For signed tests.

V:Rdh7 *R15*

Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R15

Set if MSB of the result is set; cleared otherwise.

Z: *R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $0000; cleared otherwise.

C:R15 *Rdh7*

Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
sbiw r25:r24,1 ; Subtract 1 from r25:r24
sbiw YH:YL,63  ; Subtract 63 from the Y pointer(r29:r28)
```

Words: *1 (2 bytes)*

Cycles: *2*

SEI - Set Global Interrupt Flag

Description:

Sets the Global Interrupt flag (I) in SREG (status register). The instruction following SEI will be executed before any pending interrupts.

Operation:

(i) I 1

Syntax: Operands: Program Counter:

(i) SEI None PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

I: 1

Global Interrupt flag set

Example:

```
sei          ; set global interrupt enable
sleep        ; enter sleep, waiting for interrupt
             ; note: will enter sleep before any pending interrupt(s)
```

Words: *1 (2 bytes)*

Cycles: *1*

SEN - Set Negative Flag

Description:

Sets the Negative flag (N) in SREG (status register).

Operation:

(i) N 1

Syntax: Operands: Program Counter:

(i) SEN None PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

N: 1

Negative flag set

Example:

```
add r2,r19      ; Add r19 to r2
sen             ; Set negative flag
```

Words: *1 (2 bytes)*

Cycles: *1*

SER - Set all bits in Register

Description:

Loads $FF directly to register Rd.

Operation:

(i) Rd ← $FF

Syntax:	Operands:	Program Counter:
(i) SER Rd	16 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clr r16             ; Clear r16
ser r17             ; Set r17
out $18,r16         ; Write zeros to Port B
nop                 ; Delay (do nothing)
```

out $18,r17 ; Write ones to Port B

Words: *1 (2 bytes)*

Cycles: *1*

SES - Set Signed Flag

Description:

Sets the Signed flag (S) in SREG (status register).

Operation:

(i) S 1

Syntax: Operands: Program Counter:

(i) SES None PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

S: 1

Signed flag set

Example:

add r2,r19 ; Add r19 to r2

ses ; Set negative flag

Words: *1 (2 bytes)*

Cycles: *1*

SET - Set T Flag

Description:

Sets the T flag in SREG (status register).

Operation:

(i) T 1

Syntax: Operands: Program Counter:

(i) SET None PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

T: 1

T flag set

Example:

set ; Set T flag

Words: *1 (2 bytes)*

Cycles: *1*

SEV - Set Overflow Flag

Description:

Sets the Overflow flag (V) in SREG (status register).

Operation:

(i) V 1

Syntax:	Operands:	Program Counter:
(i)SEV	None	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

V: 1

Overflow flag set

Example:

```
add r2,r19; Add r19 to r2
sev      ; Set overflow flag
```

Words: *1 (2 bytes)*

Cycles: *1*

SEZ - Set Zero Flag

Description:

Sets the Zero flag (Z) in SREG (status register).

Operation:

(i) Z 1

Syntax:	Operands:	Program Counter:
(i) SEZ	None	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Z: 1

Zero flag set

Example:

```
add r2,r19     ; Add r19 to r2
sez            ; Set zero flag
```

Words: *1 (2 bytes)*

Cycles: *1*

SLEEP

Description:

This instruction sets the circuit in sleep mode defined by the MCU control register.

Operation:

Refer to the device documentation for detailed description of SLEEP usage.

Syntax:	Operands:	Program Counter:	
SLEEP	None	PC	PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
mov r0,r11          ; Copy r11 to r0
ldi r16,(1<<SE)     ; Enable sleep mode
out MCUCR, r16
```

```
sleep            ; Put MCU in sleep mode
```

Words: *1 (2 bytes)*

Cycles: *1*

SPM - Store Program Memory

Description:

SPM can be used to erase a page in the program memory, to write a page in the program memory (that is already erased), and to set boot loader lock bits. In some devices, the program memory can be written one word at a time, in other devices an entire page can be programmed simultaneously after first filling a temporary page buffer. In all cases, the program memory must be erased one page at a time. When erasing the program memory, the RAMPZ and Z register are used as page address. When writing the program memory, the RAMPZ and Z register are used as page or word address, and the R1:R0 register pair is used as dataSee R1 determines the instruction high byte, and R0 determines the instruction low byte... When setting the boot loader lock bits, the R1:R0 register pair is used as data. Refer to the device documentation for detailed description of SPM usage. This instruction can address the entire program memory.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

1. R1 determines the instruction high byte, and R0 determines the instruction low byte.

Operation:		Comment:
(i)(RAMPZ:Z)	$ffff	Erase program memory page
(ii)(RAMPZ:Z)	R1:R0	Write program memory word
(iii)(RAMPZ:Z)	R1:R0	Write temporary page buffer
(iv)(RAMPZ:Z)	TEMP	Write temporary page buffer to program memory
(v)BLBITS	R1:R0	Set boot loader lock bits

Syntax:	Operands:	Program Counter:
(i)-(v) SPM	None	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
;This example shows SPM write of one page for devices with page write
;- the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y pointer
; the first data location in Flash is pointed to by the Z pointer
;- error handling is not included
;- the routine must be placed inside the boot space
; (at least the do_spm sub routine)
;- registers used: r0, r1, temp1, temp2, looplo, loophi, spmcrval
; (temp1, temp2, looplo, loophi, spmcrval must be defined by the user)
; storing and restoring of registers is not included in the routine
; register usage can be optimized at the expense of code size

.equ PAGESIZEB = PAGESIZE*2;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
write_page:
;page erase
ldi spmcrval, (1<<PGERS) + (1<<SPMEN)
call do_spm

;transfer data from RAM to Flash page buffer
ldi looplo, low(PAGESIZEB)          ;init loop variable
ldi loophi, high(PAGESIZEB)         ;not required for PAGESIZEB<=256
wrloop:ld r0, Y+
ld r1, Y+
ldi spmcrval, (1<<SPMEN)
call do_spm
adiw ZH:ZL, 2
```

```
sbiw loophi:looplo, 2;use subi for PAGESIZEB<=256
brne wrloop

;execute page write
subi ZL, low(PAGESIZEB)          ;restore pointer
sbci ZH, high(PAGESIZEB)         ;not required for PAGESIZEB<=256
ldi spmcrval, (1<<PGWRT) + (1<<SPMEN)
call do_spm

;read back and check, optional
ldi looplo, low(PAGESIZEB)       ;init loop variable
ldi loophi, high(PAGESIZEB)      ;not required for PAGESIZEB<=256
subi YL, low(PAGESIZEB)          ;restore pointer
sbci YH, high(PAGESIZEB)
rdloop:lpmr0, Z+
ldr 1, Y+
cp ser0, r1
jmp error
sbi wloophi:looplo, 2;use subi for PAGESIZEB<=256
brne rdloop

;return
ret

do_spm:
;input: spmcrval determines SPM action
;disable interrupts if enabled, store status
```

```
in temp2, SREG

cli

;check for previous SPM complete

wait:intemp1, SPMCR

sbr ctemp1, SPMEN

rjm pwait

;SPM timed sequence

out SPMCR, spmcrval

spm

;restore SREG (to enable interrupts if originally enabled)

out SREG, temp2

ret
```

Words: *1 (2 bytes)*

Cycles: *depends on the operation*

ST - Store Indirect From Register to data space using Index X

Description:

Stores one byte indirect from a register to data space. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

The data location is pointed to by the X (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPX in register in the I/O area has to be changed.

The X pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the X pointer register. Note that only the low byte of the X pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPX register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes program memory, and the increment/ decrement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

ST X+, r26

ST X+, r27

ST -X, r26

ST -X, r27

Using the X pointer:

Operation:	Comment:
(i)(X) Rr	X: Unchanged
(ii)(X) Rr X X+1	X: Post incremented
(iii)X X - 1 (X) Rr	X: Pre decremented

Syntax:	Operands:	Program Counter:
(i)ST X, Rr	0 ≤ r ≤ 31	PC PC + 1
(ii)ST X+, Rr	0 ≤ r ≤ 31	PC PC + 1
(iii)ST -X, Rr	0 ≤ r ≤ 31	PC PC + 1

16-bit Opcode :

Status Register (SREG) and Boolean Formula:

Example:

```
clr   r27          ; Clear X high byte
ldi   r26,$60      ; Set X low byte to $60
st    X+,r0        ; Store r0 in data space loc. $60(X post inc)
st    X,r1         ; Store r1 in data space loc. $61
ldi   r26,$63      ; Set X low byte to $63
```

```
st    X,r2          ; Store r2 in data space loc. $63

st   -X,r3          ; Store r3 in data space loc. $62(X pre dec)
```

Words: *1 (2 bytes)*

Cycles: *2*

ST (STD) - Store Indirect From Register to data space using Index Y

Description:

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

The data location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPY in register in the I/O area has to be changed.

The Y pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the Y pointer register. Note that only the low byte of the Y pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPY register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes program memory, and the increment/ decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

```
ST Y+, r28

ST Y+, r29

ST -Y, r28

ST -Y, r29
```

Using the Y pointer:

Operation:	Comment:
(i)(Y) Rr	Y: Unchanged
(ii)(Y) Rr Y Y+1	Y: Post incremented
(iii)Y Y - 1 (Y) Rr	Y: Pre decremented
(iiii) (Y+q) Rr	Y: Unchanged, q: Displacement

Syntax:	Operands:	Program Counter:
(i)ST Y, Rr	0 ≤ r ≤ 31	PC PC + 1
(ii)ST Y+, Rr	0 ≤ r ≤ 31	PC PC + 1
(iii)ST -Y, Rr	0 ≤ r ≤ 31	PC PC + 1
(iiii)STD Y+q, Rr	0 ≤ r ≤ 31, 0 ≤ q ≤ 63	PC PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
clr     r29          ; Clear Y high byte
ldi     r28,$60      ; Set Y low byte to $60
st      Y+,r0        ; Store r0 in data space loc. $60(Y post inc)
st      Y,r1         ; Store r1 in data space loc. $61
ldi     r28,$63      ; Set Y low byte to $63
st      Y,r2         ; Store r2 in data space loc. $63
st      -Y,r3        ; Store r3 in data space loc. $62(Y pre dec)
std     Y+2,r4       ; Store r4 in data space loc. $64
```

Words: *1 (2 bytes)*

Cycles: *2*

ST (STD) - Store Indirect From Register to data space using Index Z

Description:

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.

The Z pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for stack pointer usage of the Z pointer register, however because the Z pointer register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y pointer as a dedicated stack pointer. Note that only the low byte of the Z pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

ST Z+, r30

ST Z+, r31

ST -Z, r30

ST -Z, r31

Using the Z pointer:

Operation:			Comment:
(i)(Z)	Rr		Z: Unchanged
(ii)(Z)	Rr	Z Z+1	Z: Post incremented
(iii)Z	Z - 1	(Z) Rr	Z: Pre decremented
(iiii) (Z+q)	Rr		Z: Unchanged, q: Displacement

Syntax:	Operands:	Program Counter:
(i)ST Z, Rr	0 ≤ r ≤ 31	PC PC + 1
(ii)ST Z+, Rr	0 ≤ r ≤ 31	PC PC + 1
(iii)ST -Z, Rr	0 ≤ r ≤ 31	PC PC + 1
(iiii)STD Z+q, Rr	0 ≤ r ≤ 31, 0 ≤ q ≤ 63	PC PC + 1

16-bit Opcode :

Status Register (SREG) and Boolean Formula:

Example:

```
clr     r31          ; Clear Z high byte
ldi     r30,$60      ; Set Z low byte to $60
st      Z+,r0        ; Store r0 in data space loc. $60(Z post inc)
st      Z,r1         ; Store r1 in data space loc. $61
ldi     r30,$63      ; Set Z low byte to $63
st      Z,r2         ; Store r2 in data space loc. $63
st      -Z,r3        ; Store r3 in data space loc. $62(Z pre dec)
std     Z+2,r4       ; Store r4 in data space loc. $64
```

Words: *1 (2 bytes)*

Cycles: *2*

STS - Store Direct to data space

Description:

Stores one byte from a Register to the data space. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.

A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The STS instruction uses the RAMPD register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be

changed.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i)(k) Rr

Syntax: Operands: Program Counter:

(i)STS k,Rr 0 ≤ r ≤ 31, 0 ≤ k ≤ 65535 PC PC + 2

32-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

```
lds   r2,$FF00   ; Load r2 with the contents of data space location $FF00
add   r2,r1      ; add r1 to r2
sts   $FF00,r2   ; Write back
```

Words: *2 (4 bytes)*

Cycles: *2*

SUB - Subtract without Carry

Description:

Subtracts two registers and places the result in the destination register Rd.

Operation:

(i)Rd Rd - Rr

Syntax: Operands: Program Counter:

(i)SUB Rd,Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31 PC PC + 1

16-bit Opcode:

Status Register and Boolean Formula:

H: *Rd3* Rr3 +Rr3 R3 +R3 *Rd3*

Set if there was a borrow from bit 3; cleared otherwise

S: N V, For signed tests.

V:Rd7 *Rr7* *R7 + Rd7* Rr7 R7

Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7* *R6* *R5* *R4* *R3* *R2* *R1* *R0*

Set if the result is $00; cleared otherwise.

C: *Rd7* Rr7 +Rr7 R7 +R7 *Rd7*

Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
sub     r13,r12       ; Subtract r12 from r13
brne    noteq         ; Branch if r12<>r13
...
noteq:  nop           ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

SUBI - Subtract Immediate

Description:

Subtracts a register and a constant and places the result in the destination register Rd. This instruction is working on Register R16 to R31 and is very well suited for operations on the X, Y and Z pointers.

Operation:

(i)Rd Rd - K

Syntax: Operands: Program Counter:

(i)SUBI Rd,K 16 ≤ d ≤ 31, 0 ≤ K ≤ 255 PC PC + 1

16-bit Opcode:

Status Register and Boolean Formula:

H: *Rd3* K3+K3 R3 +R3 *Rd3*

Set if there was a borrow from bit 3; cleared otherwise

S: N [+] V, For signed tests.

V:Rd7 *K7* *R7 + Rd7* K7 R7

Set if two's complement overflow resulted from the operation; cleared otherwise.

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

C: *Rd7* K7 +K7 R7 +R7 *Rd7*

Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
subi    r22,$11       ; Subtract $11 from r22
brne    noteq         ; Branch if r22<>$11
...
noteq:  nop           ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

SWAP - Swap Nibbles

Description:

Swaps high and low nibbles in a register.

Operation:

(i)R(7:4) Rd(3:0), R(3:0) Rd(7:4)

Syntax:	Operands:	Program Counter:
(i)SWAP Rd	0 ≤ d ≤ 31	PC PC + 1

16-bit Opcode:

Status Register and Boolean Formula:

R (Result) equals Rd after the operation.

Example:

```
inc     r1          ; Increment r1
swap    r1          ; Swap high and low nibble of r1
inc     r1          ; Increment high nibble of r1
swap    r1          ; Swap back
```

Words: *1 (2 bytes)*

Cycles: *1*

TST - Test for Zero or Minus

Description:

Tests if a register is zero or negative. Performs a logical AND between a register and itself. The register will remain unchanged.

Operation:

(i)Rd Rd Rd

Syntax:	Operands:	Program Counter:
(i)TST Rd	0 ≤ d ≤ 31	PC PC + 1

16-bit Opcode: *(see AND Rd, Rd)*

Status Register and Boolean Formula:

S: N [+] V, For signed tests.

V:0

Cleared

N:R7

Set if MSB of the result is set; cleared otherwise.

Z: *R7 R6 R5 R4 R3 R2 R1 R0*

Set if the result is $00; cleared otherwise.

R (Result) equals Rd.

Example:

```
tst r0              ; Test r0
breq zero           ; Branch if r0=0
...
zero:   nop                 ; Branch destination (do nothing)
```

Words: *1 (2 bytes)*

Cycles: *1*

WDR - Watchdog Reset

Description:

This instruction resets the Watchdog Timer. This instruction must be executed within a limited time given by the WD prescaler. See the Watchdog Timer hardware specification.

Operation:

(i)WD timer restart.

Syntax:	Operands:	Program Counter:
(i)WDR	None	PC PC + 1

16-bit Opcode:

Status Register and Boolean Formula:

Example:

```
wdr             ; Reset watchdog timer
```

Words: *1 (2 bytes)*

Cycles: *1*