**AVR Instruction Set**

This section describes all instructions for the 8-bit AVR in detail. For a specific device please refer to the specific Instruction Set Summary in the hardware description.

Addressing modes are described in detail in the hardware description for each device.
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: Twos complement overflow indicator
S: \(N \oplus V\), 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 literal or byte data (8 bit)
k: Constant address data for program counter
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)
P: I/O port 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 SRAM area on MCUs with more than 64K bytes SRAM.

Stack:
STACK: Stack for return address and pushed registers
SP: Stack Pointer to STACK

Opcode:
X: Don’t care

Flags:
⇔: Flag affected by instruction
0: Flag cleared by instruction
1: Flag set by instruction
-: Flag not affected by instruction

Conditional Branch Summary

<table>
<thead>
<tr>
<th>Test</th>
<th>Boolean</th>
<th>Mnemonic</th>
<th>Complementary</th>
<th>Boolean</th>
<th>Mnemonic</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd &gt; Rr</td>
<td>Z(N \oplus V) = 0</td>
<td>BRLT*</td>
<td>Rd ≤ Rr</td>
<td>Z(N \oplus V) = 1</td>
<td>BRGE*</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd ≥ Rr</td>
<td>(N \oplus V) = 0</td>
<td>BRGE</td>
<td>Rd &lt; Rr</td>
<td>(N \oplus V) = 1</td>
<td>BRLT</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd = Rr</td>
<td>Z = 1</td>
<td>BREQ</td>
<td>Rd ≠ Rr</td>
<td>Z = 0</td>
<td>BRNE</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd ≤ Rr</td>
<td>(Z+(N \oplus V) = 1)</td>
<td>BRGE*</td>
<td>Rd &gt; Rr</td>
<td>(Z+(N \oplus V) = 0)</td>
<td>BRLT*</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd &lt; Rr</td>
<td>(N \oplus V) = 1</td>
<td>BRLT</td>
<td>Rd ≥ Rr</td>
<td>(N \oplus V) = 0</td>
<td>BRGE</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd &gt; Rr</td>
<td>C + Z = 0</td>
<td>BRLO*</td>
<td>Rd ≤ Rr</td>
<td>C + Z = 1</td>
<td>BRSH*</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Rd ≥ Rr</td>
<td>C = 0</td>
<td>BRSH/BRCC</td>
<td>Rd &lt; Rr</td>
<td>C = 1</td>
<td>BRLO/BRCS</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Rd = Rr</td>
<td>Z = 1</td>
<td>BREQ</td>
<td>Rd ≠ Rr</td>
<td>Z = 0</td>
<td>BRNE</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Rd ≤ Rr</td>
<td>C + Z = 1</td>
<td>BRSH*</td>
<td>Rd &gt; Rr</td>
<td>C + Z = 0</td>
<td>BRLO*</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Rd &lt; Rr</td>
<td>C = 1</td>
<td>BRLO/BRCS</td>
<td>Rd ≥ Rr</td>
<td>C = 0</td>
<td>BRSH/BRCC</td>
<td>Unsigned</td>
</tr>
<tr>
<td>Carry</td>
<td>C = 1</td>
<td>BRCS</td>
<td>No carry</td>
<td>C = 0</td>
<td>BRCC</td>
<td>Simple</td>
</tr>
<tr>
<td>Negative</td>
<td>N = 1</td>
<td>BRMI</td>
<td>Positive</td>
<td>N = 0</td>
<td>BRPL</td>
<td>Simple</td>
</tr>
<tr>
<td>Overflow</td>
<td>V = 1</td>
<td>BRVS</td>
<td>No overflow</td>
<td>V = 0</td>
<td>BRVC</td>
<td>Simple</td>
</tr>
<tr>
<td>Zero</td>
<td>Z = 1</td>
<td>BREQ</td>
<td>Not zero</td>
<td>Z = 0</td>
<td>BRNE</td>
<td>Simple</td>
</tr>
</tbody>
</table>

* Interchange Rd and Rr in the operation before the test. i.e. CP Rd,Rr → CP Rr,Rd
Instruction Set

Complete Instruction Set Summary

<table>
<thead>
<tr>
<th>Mnem-</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Rd, Rr</td>
<td>Add without Carry</td>
<td>Rd ← Rd + Rr</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>ADC</td>
<td>Rd, Rr</td>
<td>Add with Carry</td>
<td>Rd ← Rd + Rr + C</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>ADIW</td>
<td>Rd, K</td>
<td>Add Immediate to Word</td>
<td>Rd+1:Rd ← Rd+1:Rd + K</td>
<td>Z,C,N,V</td>
<td>2</td>
</tr>
<tr>
<td>SUB</td>
<td>Rd, Rr</td>
<td>Subtract without Carry</td>
<td>Rd ← Rd - Rr</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SUBI</td>
<td>Rd, K</td>
<td>Subtract Immediate</td>
<td>Rd ← Rd - K</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBC</td>
<td>Rd, Rr</td>
<td>Subtract with Carry</td>
<td>Rd ← Rd - Rd - C</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBCI</td>
<td>Rd, K</td>
<td>Subtract Immediate with Carry</td>
<td>Rd ← Rd - K - C</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBIW</td>
<td>Rd, K</td>
<td>Subtract Immediate from Word</td>
<td>Rd+1:Rd ← Rd+1:Rd - K</td>
<td>Z,C,N,V</td>
<td>2</td>
</tr>
<tr>
<td>AND</td>
<td>Rd, Rr</td>
<td>Logical AND</td>
<td>Rd ← Rd • Rr</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>ANDI</td>
<td>Rd, K</td>
<td>Logical AND with Immediate</td>
<td>Rd ← Rd • K</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>OR</td>
<td>Rd, Rr</td>
<td>Logical OR</td>
<td>Rd ← Rd v Rr</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>ORI</td>
<td>Rd, K</td>
<td>Logical OR with Immediate</td>
<td>Rd ← Rd v K</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>EOR</td>
<td>Rd, Rr</td>
<td>Exclusive OR</td>
<td>Rd ← Rd ⊕ Rr</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>COM</td>
<td>Rd</td>
<td>One’s Complement</td>
<td>Rd ← $FF - Rd</td>
<td>Z,C,N,V</td>
<td>1</td>
</tr>
<tr>
<td>NEG</td>
<td>Rd</td>
<td>Two’s Complement</td>
<td>Rd ← $00 - Rd</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBR</td>
<td>Rd,K</td>
<td>Set Bit(s) in Register</td>
<td>Rd ← Rd v K</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>CBR</td>
<td>Rd,K</td>
<td>Clear Bit(s) in Register</td>
<td>Rd ← Rd • ($FFh - K)</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>INC</td>
<td>Rd</td>
<td>Increment</td>
<td>Rd ← Rd + 1</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>DEC</td>
<td>Rd</td>
<td>Decrement</td>
<td>Rd ← Rd - 1</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>TST</td>
<td>Rd</td>
<td>Test for Zero or Minus</td>
<td>Rd ← Rd • Rd</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>CLR</td>
<td>Rd</td>
<td>Clear Register</td>
<td>Rd ← Rd ⊕ Rd</td>
<td>Z,N,V</td>
<td>1</td>
</tr>
<tr>
<td>SER</td>
<td>Rd</td>
<td>Set Register</td>
<td>Rd ← $FF</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>MUL</td>
<td>Rd,Rr</td>
<td>Multiply Unsigned</td>
<td>R1, R0 ← Rd × Rr</td>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

√ Not available in base-line microcontrollers

(continued)
### Complete Instruction Set Summary (continued)

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clock Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BRANCH INSTRUCTIONS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RJMP</td>
<td>k</td>
<td>Relative Jump</td>
<td>PC ← PC + k + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>IJMP</td>
<td></td>
<td>Indirect Jump to (Z)</td>
<td>PC ← Z</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>JMP</td>
<td>k</td>
<td>Jump</td>
<td>PC ← k</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>RCALL</td>
<td>k</td>
<td>Relative Call Subroutine</td>
<td>PC ← PC + k + 1</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>ICALL</td>
<td></td>
<td>Indirect Call to (Z)</td>
<td>PC ← Z</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>CALL</td>
<td>k</td>
<td>Call Subroutine</td>
<td>PC ← k</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td>Subroutine Return</td>
<td>PC ← STACK</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>RETI</td>
<td></td>
<td>Interrupt Return</td>
<td>PC ← STACK</td>
<td>I</td>
<td>4</td>
</tr>
<tr>
<td>CPSE</td>
<td>Rd,Rr</td>
<td>Compare, Skip if Equal</td>
<td>if (Rd = Rr) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>CP</td>
<td>Rd,Rr</td>
<td>Compare</td>
<td>Rd - Rr</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>CPC</td>
<td>Rd,Rr</td>
<td>Compare with Carry</td>
<td>Rd - Rr - C</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>CPI</td>
<td>Rd,K</td>
<td>Compare with Immediate</td>
<td>Rd - K</td>
<td>Z,C,N,V,H</td>
<td>1</td>
</tr>
<tr>
<td>SBRC</td>
<td>Rr, b</td>
<td>Skip if Bit in Register Cleared</td>
<td>if (Rr(b)=0) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>SBRS</td>
<td>Rr, b</td>
<td>Skip if Bit in Register Set</td>
<td>if (Rr(b)=1) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>SBIC</td>
<td>P, b</td>
<td>Skip if Bit in I/O Register Cleared</td>
<td>if(I/O(P,b)=0) PC ← PC + 2 or 3</td>
<td>None</td>
<td>2 / 3</td>
</tr>
<tr>
<td>SBIS</td>
<td>P, b</td>
<td>Skip if Bit in I/O Register Set</td>
<td>if(I/O(P,b)=1) PC ← PC + 2 or 3</td>
<td>None</td>
<td>2 / 3</td>
</tr>
<tr>
<td>BRBS</td>
<td>s, k</td>
<td>Branch if Status Flag Set</td>
<td>if (SREG(s) = 1) then PC←PC+k+1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRBC</td>
<td>s, k</td>
<td>Branch if Status Flag Cleared</td>
<td>if (SREG(s) = 0) then PC←PC+k+1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BREQ</td>
<td>k</td>
<td>Branch if Equal</td>
<td>if (Z = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRNE</td>
<td>k</td>
<td>Branch if Not Equal</td>
<td>if (Z = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRCS</td>
<td>k</td>
<td>Branch if Carry Set</td>
<td>if (C = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRCC</td>
<td>k</td>
<td>Branch if Carry Cleared</td>
<td>if (C = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRSH</td>
<td>k</td>
<td>Branch if Same or Higher</td>
<td>if (C = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRLO</td>
<td>k</td>
<td>Branch if Lower</td>
<td>if (C = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRMI</td>
<td>k</td>
<td>Branch if Minus</td>
<td>if (N = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRPL</td>
<td>k</td>
<td>Branch if Plus</td>
<td>if (N = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRGE</td>
<td>k</td>
<td>Branch if Greater or Equal, Signed</td>
<td>if (N ⊕ V = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRLT</td>
<td>k</td>
<td>Branch if Less Than, Signed</td>
<td>if (N ⊕ V = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRHS</td>
<td>k</td>
<td>Branch if Half Carry Flag Set</td>
<td>if (H = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRHC</td>
<td>k</td>
<td>Branch if Half Carry Flag Cleared</td>
<td>if (H = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRTS</td>
<td>k</td>
<td>Branch if T Flag Set</td>
<td>if (T = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRTC</td>
<td>k</td>
<td>Branch if T Flag Cleared</td>
<td>if (T = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRVS</td>
<td>k</td>
<td>Branch if Overflow Flag is Set</td>
<td>if (V = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRVC</td>
<td>k</td>
<td>Branch if Overflow Flag is Cleared</td>
<td>if (V = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRIE</td>
<td>k</td>
<td>Branch if Interrupt Enabled</td>
<td>if (I = 1) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
<tr>
<td>BRID</td>
<td>k</td>
<td>Branch if Interrupt Disabled</td>
<td>if (I = 0) then PC ← PC + k + 1</td>
<td>None</td>
<td>1 / 2</td>
</tr>
</tbody>
</table>

(continued)
## Complete Instruction Set Summary (continued)

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clock Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DATA TRANSFER INSTRUCTIONS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>Rd, Rr</td>
<td>Copy Register</td>
<td>Rd $\leftarrow$ Rr</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>LDI</td>
<td>Rd, K</td>
<td>Load Immediate</td>
<td>Rd $\leftarrow$ K</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>LDS</td>
<td>Rd, k</td>
<td>Load Direct from SRAM</td>
<td>Rd $\leftarrow$ (k)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, X</td>
<td>Load Indirect</td>
<td>Rd $\leftarrow$ (X)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, X+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd $\leftarrow$ (X), X $\leftarrow$ X + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Y</td>
<td>Load Indirect and Pre-Decrement</td>
<td>X $\leftarrow$ X - 1, Rd $\leftarrow$ (X)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Y+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd $\leftarrow$ (Y), Y $\leftarrow$ Y + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, -Y</td>
<td>Load Indirect and Pre-Decrement</td>
<td>Y $\leftarrow$ Y - 1, Rd $\leftarrow$ (Y)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LDD</td>
<td>Rd, Y+q</td>
<td>Load Indirect with Displacement</td>
<td>Rd $\leftarrow$ (Y + q)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Z</td>
<td>Load Indirect</td>
<td>Rd $\leftarrow$ (Z)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Z+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd $\leftarrow$ (Z), Z $\leftarrow$ Z+1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, -Z</td>
<td>Load Indirect and Pre-Decrement</td>
<td>Z $\leftarrow$ Z - 1, Rd $\leftarrow$ (Z)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LDD</td>
<td>Rd, Z+q</td>
<td>Load Indirect with Displacement</td>
<td>Rd $\leftarrow$ (Z + q)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>STS</td>
<td>k, Rr</td>
<td>Store Direct to SRAM</td>
<td>Rd $\leftarrow$ (k)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>ST</td>
<td>X, Rr</td>
<td>Store Indirect</td>
<td>(X) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>X+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(X) $\leftarrow$ Rr, X $\leftarrow$ X + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>-X, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>X $\leftarrow$ X - 1, (X) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Y, Rr</td>
<td>Store Indirect</td>
<td>(Y) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Y+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(Y) $\leftarrow$ Rr, Y $\leftarrow$ Y + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>-Y, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>Y $\leftarrow$ Y - 1, (Y) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>STD</td>
<td>Y+q, Rr</td>
<td>Store Indirect with Displacement</td>
<td>(Y + q) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Z, Rr</td>
<td>Store Indirect</td>
<td>(Z) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Z+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(Z) $\leftarrow$ Rr, Z $\leftarrow$ Z + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>-Z, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>Z $\leftarrow$ Z - 1, (Z) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>STD</td>
<td>Z+q, Rr</td>
<td>Store Indirect with Displacement</td>
<td>(Z + q) $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LPM</td>
<td></td>
<td>Load Program Memory</td>
<td>RO $\leftarrow$ (Z)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>IN</td>
<td>Rd, P</td>
<td>In Port</td>
<td>Rd $\leftarrow$ P</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>OUT</td>
<td>P, Rr</td>
<td>Out Port</td>
<td>P $\leftarrow$ Rr</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>PUSH</td>
<td>Rr</td>
<td>Push Register on Stack</td>
<td>STACK $\leftarrow$ Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>POP</td>
<td>Rd</td>
<td>Pop Register from Stack</td>
<td>Rd $\leftarrow$ STACK</td>
<td>None</td>
<td>2</td>
</tr>
</tbody>
</table>

(continued)
Complete Instruction Set Summary  (continued)

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clock Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSL</td>
<td>Rd</td>
<td>Logical Shift Left</td>
<td>Rd(n+1)←Rd(n), Rd(0)←0, C←Rd(7)</td>
<td>Z.C.N.V.H</td>
<td>1</td>
</tr>
<tr>
<td>LSR</td>
<td>Rd</td>
<td>Logical Shift Right</td>
<td>Rd(n)←Rd(n+1), Rd(7)←0, C←Rd(0)</td>
<td>Z.C.N.V</td>
<td>1</td>
</tr>
<tr>
<td>ROL</td>
<td>Rd</td>
<td>Rotate Left Through Carry</td>
<td>Rd(0)←C,Rd(n+1)←Rd(n), C←Rd(7)</td>
<td>Z.C.N.V.H</td>
<td>1</td>
</tr>
<tr>
<td>ROR</td>
<td>Rd</td>
<td>Rotate Right Through Carry</td>
<td>Rd(7)←C,Rd(n)←Rd(n+1), C←Rd(0)</td>
<td>Z.C.N.V</td>
<td>1</td>
</tr>
<tr>
<td>ASR</td>
<td>Rd</td>
<td>Arithmetic Shift Right</td>
<td>Rd(n)←Rd(n+1), n=0..6</td>
<td>Z.C.N.V</td>
<td>1</td>
</tr>
<tr>
<td>SWAP</td>
<td>Rd</td>
<td>Swap Nibbles</td>
<td>Rd(3..0)←Rd(7..4)</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>BSET</td>
<td>s</td>
<td>Flag Set</td>
<td>SREG(s)←1</td>
<td>SREG(s)</td>
<td>1</td>
</tr>
<tr>
<td>BCLR</td>
<td>s</td>
<td>Flag Clear</td>
<td>SREG(s)←0</td>
<td>SREG(s)</td>
<td>1</td>
</tr>
<tr>
<td>SBI</td>
<td>P, b</td>
<td>Set Bit in I/O Register</td>
<td>I/O(P, b)←1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>CBI</td>
<td>P, b</td>
<td>Clear Bit in I/O Register</td>
<td>I/O(P, b)←0</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>BST</td>
<td>Rr, b</td>
<td>Bit Store from Register to T</td>
<td>T←Rr(b)</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>BLD</td>
<td>Rd, b</td>
<td>Bit load from T to Register</td>
<td>Rd(b)←T</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>SEC</td>
<td>Rd</td>
<td>Set Carry</td>
<td>C←1</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>CLC</td>
<td></td>
<td>Clear Carry</td>
<td>C←0</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>SEN</td>
<td></td>
<td>Set Negative Flag</td>
<td>N←1</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>CLN</td>
<td></td>
<td>Clear Negative Flag</td>
<td>N←0</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>SEZ</td>
<td></td>
<td>Set Zero Flag</td>
<td>Z←1</td>
<td>Z</td>
<td>1</td>
</tr>
<tr>
<td>CLZ</td>
<td></td>
<td>Clear Zero Flag</td>
<td>Z←0</td>
<td>Z</td>
<td>1</td>
</tr>
<tr>
<td>SEI</td>
<td></td>
<td>Global Interrupt Enable</td>
<td>I←1</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>CLI</td>
<td></td>
<td>Global Interrupt Disable</td>
<td>I←0</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>SES</td>
<td></td>
<td>Set Signed Test Flag</td>
<td>S←1</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>CLS</td>
<td></td>
<td>Clear Signed Test Flag</td>
<td>S←0</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>SEV</td>
<td></td>
<td>Set Two’s Complement Overflow</td>
<td>V←1</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>CLV</td>
<td></td>
<td>Clear Two’s Complement Overflow</td>
<td>V←0</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td>Set T in SREG</td>
<td>T←1</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>CLT</td>
<td></td>
<td>Clear T in SREG</td>
<td>T←0</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>SEH</td>
<td></td>
<td>Set Half Carry Flag in SREG</td>
<td>H←1</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>CLH</td>
<td></td>
<td>Clear Half Carry Flag in SREG</td>
<td>H←0</td>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td>No Operation</td>
<td>None</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>SLEEP</td>
<td></td>
<td>Sleep</td>
<td>(see specific descr. for Sleep)</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>WDR</td>
<td></td>
<td>Watchdog Reset</td>
<td>(see specific descr. for WDR)</td>
<td>None</td>
<td>1</td>
</tr>
</tbody>
</table>
### 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) \( Rd \leftarrow Rd + Rr + C \)

**Syntax:**

(i) ADC Rd,Rr  

**Operands:**  

\( 0 \leq d \leq 31, 0 \leq r \leq 31 \)

**Program Counter:**

\( PC \leftarrow PC + 1 \)

**16 bit Opcode:**

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
0001 & 11rd & dddd & rrrr \\
\end{array}
\]

**Status Register (SREG) Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: \( Rd3 \cdot Rr3 + Rr3 \cdot R3 + R3 \cdot Rd3 \)

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

S: \( N \oplus V \), For signed tests.

V: \( Rd7 \cdot Rr7 \cdot R7 + Rd7 \cdot Rr7 \cdot R7 \)

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: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)

Set if the result is \( 00 \); cleared otherwise.

C: \( Rd7 \cdot Rr7 + Rr7 \cdot R7 + R7 \cdot Rd7 \)

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:**
(i) \( Rd \leftarrow Rd + Rr \)

**Syntax:**
(i) ADD Rd,Rr

**Operands:**
\( 0 \leq d \leq 31, 0 \leq r \leq 31 \)

**Program Counter:**
\( PC \leftarrow PC + 1 \)

**16 bit Opcode:**

```
0000 11rd dddd rrrr
```

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: \( Rd_3 \cdot Rr_3 + Rr_3 \cdot R_3 + R_3 \cdot Rd_3 \)
Set if there was a carry from bit 3; cleared otherwise.

S: \( N \oplus V \), For signed tests.

V: \( Rd_7 \cdot Rr_7 \cdot R_7 + Rd_7 \cdot Rr_7 \cdot R_7 \)
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: \( R_7 \)
Set if MSB of the result is set; cleared otherwise.

Z: \( R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \)
Set if the result is \( 00 \); cleared otherwise.

C: \( Rd_7 \cdot Rr_7 + Rr_7 \cdot R_7 + R_7 \cdot Rd_7 \)
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.

**Operation:**
(i) \( \text{Rdh:Rdl} \leftarrow \text{Rdh:Rdl} + K \)

**Syntax:**
(i) \( \text{ADIW RdL,K} \)

**Operands:**
\( \text{dl} \in \{24, 26, 28, 30\}, 0 \leq K \leq 63 \)

**Program Counter:**
\( \text{PC} \leftarrow \text{PC} + 1 \)

**16 bit Opcode:**
\[
\begin{array}{ccccc}
1001 & 0110 & K & K & K
\end{array}
\]

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

- **S:** \( \text{N} \oplus \text{V} \), For signed tests.

- **V:** \( \text{Rdh7} \cdot \text{R15} \)
  Set if two’s complement overflow resulted from the operation; cleared otherwise.

- **N:** \( \text{R15} \)
  Set if MSB of the result is set; cleared otherwise.

- **Z:** \( \text{R15} \cdot \text{R14} \cdot \text{R13} \cdot \text{R12} \cdot \text{R11} \cdot \text{R10} \cdot \text{R9} \cdot \text{R8} \cdot \text{R7} \cdot \text{R6} \cdot \text{R5} \cdot \text{R4} \cdot \text{R3} \cdot \text{R2} \cdot \text{R1} \cdot \text{R0} \)
  Set if the result is $0000$; cleared otherwise.

- **C:** \( \text{R15} \cdot \text{Rdh7} \)
  Set if there was carry from the MSB of the result; cleared otherwise.

**Example:**
\[
\begin{align*}
\text{adiw} & \ r24,1 \quad ; \text{Add 1 to r25:r24} \\
\text{adiw} & \ r30,63 \quad ; \text{Add 63 to the Z pointer(r31:r30)}
\end{align*}
\]

**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) \( Rd \leftarrow Rd \cdot Rr \)

Syntax: \( \text{AND Rd,Rr} \)
Operands: \( 0 \leq d \leq 31, 0 \leq r \leq 31 \)

Program Counter:
\( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:
```
0010 00\text{rd} \text{ dddd} \text{ rrrr}
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>0</td>
<td>⇔</td>
<td>⇔</td>
<td>-</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.
V: 0
Cleared
N: \( R7 \)
Set if MSB of the result is set; cleared otherwise.
Z: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
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) \( \text{Rd} \leftarrow \text{Rd} \land \text{K} \)

Syntax: \( \text{ANDI Rd,K} \)
Operands: \( 16 \leq d \leq 31, \quad 0 \leq K \leq 255 \)
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

| 0111 | KKKK | dddd | KKKK |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>0</td>
<td>⇔</td>
<td>⇔</td>
<td>-</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.

V: 0
Cleared

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

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

R (Result) equals Rd after the operation.

Example:
\[
\begin{align*}
\text{andi} & \quad r17, $0F \quad ; \text{Clear upper nibble of } r17 \\
\text{andi} & \quad r18, $10 \quad ; \text{Isolate bit 4 in } r18 \\
\text{andi} & \quad r19, $AA \quad ; \text{Clear odd bits of } r19
\end{align*}
\]

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 two's complement value by two without changing its sign. The carry flag can be used to round the result.

Operation:

(i) \[ b_7 \ldots b_0 \rightarrow C \]

Syntax: Operands: Program Counter:
(i) ASR Rd \( 0 \leq d \leq 31 \) PC \( \leftarrow \) PC + 1

16 bit Opcode:

\[
\begin{array}{cccccc}
1001 & 010d & dddd & 0101 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.

V: \( N \oplus C \) (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).

N: \( R_7 \),
Set if MSB of the result is set; cleared otherwise.

Z: \( R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \),
Set if the result is $00$; cleared otherwise.

C: \( R_d0 \),
Set if, before the shift, the LSB of \( R_d \) was set; cleared otherwise.

R (Result) equals \( R_d \) after the operation.

Example:

\[
\begin{align*}
\text{ldi } & r16, 16 \quad ; \text{Load decimal 16 into } r16 \\
\text{asr } & r16 \quad ; \quad r16 = r16 / 2 \\
\text{ldi } & r17, 31 \quad ; \quad \text{Load } -4 \text{ in } r17 \\
\text{asr } & r17 \quad ; \quad r17 = r17 / 2 \\
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 1
BCLR - Bit Clear in SREG

Description:
Clears a single flag in SREG.

Operation:
(i) \( \text{SREG}(s) \leftarrow 0 \)

Syntax: \( \text{BCLR } s \)  
Operands: \( 0 \leq s \leq 7 \)  
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0100</td>
<td>1sss</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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:

\[
\begin{align*}
\text{bclr} & \ 0 \quad ; \text{Clear carry flag} \\
\text{bclr} & \ 7 \quad ; \text{Disable interrupts}
\end{align*}
\]

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:
(i) \( \text{Rd}(b) \leftarrow T \)

Syntax: \( \text{BLD Rd,b} \)
Operands: \( 0 \leq d \leq 31, 0 \leq b \leq 7 \)
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

```
1111 100d dddd Xbbb
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

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

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form.

Operation:
(i) If SREG(s) = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) BRBC s,k 0 ≤ s ≤ 7, -64 ≤ k ≤ +63 PC ← PC + k + 1
PC ← PC + 1, if condition is false

16 bit Opcode:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form.

Operation:
(i) If SREG(s) = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) 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:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
bst r0,3 ; 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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 0,k).

Operation:
(i) If C = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: 
(i) BRCC k
Operands: -64 ≤ k ≤ +63
Program Counter:
PC ← PC + k + 1
PC ← PC + 1, if condition is false

16 bit Opcode:
1111 01kk kkkk k000

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 0,k).

Operation:
(i) If C = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) BRCS k -64 ≤ k ≤ +63

16 bit Opcode:

<table>
<thead>
<tr>
<th>1111</th>
<th>00kk</th>
<th>kkkk</th>
<th>k000</th>
</tr>
</thead>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
```assembly
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
**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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 1,k).

**Operation:**
(i) If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1

**Syntax:**
(i) BREQ k

**Operands:**
-64 ≤ k ≤ +63

**Program Counter:**
PC ← PC + k + 1
PC ← PC + 1, if condition is false

**16 bit Opcode:**
```
1111 00kk kkkk k001
```

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>ITHEVNZC</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - - - - - - -</td>
</tr>
</tbody>
</table>

**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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 4,k).

Operation:
(i) If Rd ≥ Rr (N ⊕ V = 0) then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) BRGE k -64 ≤ k ≤ +63 PC ← PC + k + 1
PC ← PC + 1, if condition is false

16 bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>01kk</th>
<th>kkkk</th>
<th>k100</th>
</tr>
</thead>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 5,k).

Operation:
(i) If H = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) BRHC k -64 ≤ k ≤ +63 PC ← PC + k + 1

16 bit Opcode:
1111 01kk kkkk k101

Status Register (SREG) and Boolean Formulae:

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 5,k).

Operation:
(i) If H = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:
(i) BRHS k
-64 ≤ k ≤ +63

Operands: Program Counter:
PC ← PC + k + 1
PC ← PC + 1, if condition is false

16 bit Opcode:
```
1111 00kk kkkk k101
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 7,k).

**Operation:**
(i) If I = 0 then PC ← PC + k + 1, else PC ← PC + 1

**Syntax:**
(i) BRID k

**Operands:**
-64 ≤ k ≤ +63

**Program Counter:**
PC ← PC + k + 1
PC ← PC + 1, if condition is false

**16 bit Opcode:**
```
1111 01kk kkkk k111
```

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 7,k).

Operation:
(i) If I = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: 
(i) BRIE k -64 ≤ k ≤ +63

Program Counter:
PC ← PC + k + 1
PC ← PC + 1, if condition is false

16 bit Opcode:
1111 00kk kkkk k111

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 0,k).

Operation:
(i) If Rd < Rr (C = 1) then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) BRLO k -64 ≤ k ≤ +63 PC ← PC + k + 1

16 bit Opcode:

| 1111 | 00kk | kkkk | k000 |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
eor  r19,r19 ; Clear r19
loop:  inc  r19   ; 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-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 4,k).

Operation:
(i) If Rd < Rr (N ⊕ V = 1) then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) 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:
```
IT 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-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 2,k).

Operation:
(i) If N = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) 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 Formulae:

```
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-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 1,k).

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

Syntax: Operands: Program Counter:
(i) 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 Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:
loop: eor r27, r27 ; Clear r27
      inc r27 ; Increase r27
... cpi r27, 5 ; Compare r27 to 5
brne loop ; 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-64≤destination≤PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 2,k).

Operation:
(i) If N = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) 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 Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 0,k).

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

Syntax: Operands: Program Counter:
(i) BRSH k -64 ≤ k ≤ +63 PC ← PC + k + 1
     PC ← PC + 1, if condition is false

16 bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1111</th>
<th>01kk</th>
<th>kkkk</th>
<th>k000</th>
</tr>
</thead>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 6,k).

Operation:
(i) If T = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) 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:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 6,k).

Operation:
(i) If T = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:       Operands:       Program Counter:
(i) 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:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 3,k).

Operation:
(i) If V = 0 then PC ← PC + k + 1, else PC ← PC + 1

Syntax:         Operands:         Program Counter:
(i) 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 Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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-64 ≤ destination ≤ PC+63). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBS 3,k).

Operation:
(i) If V = 1 then PC ← PC + k + 1, else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) BRVS k -64 ≤ k ≤ +63 PC ← PC + k + 1
PC ← PC + 1, if condition is false

16 bit Opcode:

Status Register (SREG) and Boolean Formulae:

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:
(i) \( \text{SREG}(s) \leftarrow 1 \)

Syntax: \( \text{BSET} \ s \) 0 \( \leq s \leq 7 \)

Operands: Program Counter:
(i) BSET s 0 \( \leq s \leq 7 \) PC \( \leftarrow \) PC + 1

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 0100 & 0\text{sss} & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

- 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 otherwise.

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:
(i) \( T \leftarrow \text{Rd}(b) \)

Syntax:
(i) BST Rd,b

Operands:
0 \( \leq d \leq 31 \), 0 \( \leq b \leq 7 \)

Program Counter:
\( PC \leftarrow PC + 1 \)

16 bit Opcode:

| I | 1111 | 101d | dddd | Xbbb |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>⇔</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

Example:

```assembly
bst r1,2 ; Copy bit
bld r0,4 ; 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).

Operation:

(i) \( PC \leftarrow k \) Devices with 16 bits PC, 128K bytes program memory maximum.
(ii) \( PC \leftarrow k \) Devices with 22 bits PC, 8M bytes program memory maximum.

Syntax: Operands: Program Counter: Stack

(i) CALL \( k \) \( 0 \leq k \leq 64K \) \( PC \leftarrow k \) STACK \( \leftarrow \) PC+2
SP \( \leftarrow \) SP-2, (2 bytes, 16 bits)

(ii) CALL \( k \) \( 0 \leq k \leq 4M \) \( PC \leftarrow k \) STACK \( \leftarrow \) PC+2
SP \( \leftarrow \) SP-3 (3 bytes, 22 bits)

32 bit Opcode:

\[
\begin{array}{cccc}
1001 & 010k & kkkk & 111k \\
kkkk & kkkk & kkkk & kkkk \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

\[
\begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
\end{array}
\]

Example:

\[
\begin{align*}
\text{mov} & \quad r16,r0 \quad ; \text{Copy r0 to r16} \\
\text{call} & \quad \text{check} \quad ; \text{Call subroutine} \\
\text{nop} & \quad \text{; Continue (do nothing)} \\
\ldots & \quad \text{check:} \quad \text{cpi} \quad r16,$42 \quad ; \text{Check if r16 has a special value} \\
\text{breq} & \quad \text{error} \quad ; \text{Branch if equal} \\
\text{ret} & \quad \text{; Return from subroutine} \\
\ldots & \quad \text{error:} \quad \text{rjmp} \quad \text{error} \quad ; \text{Infinite loop}
\end{align*}
\]

Words: 2 (4 bytes)
Cycles: 4
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) \( \text{I/O(P,b)} \leftarrow 0 \)

Syntax:
(i) CBI P,b

Operands:
0 \( \leq P \leq 31 \), 0 \( \leq b \leq 7 \)

Program Counter:
PC \( \leftarrow \) PC + 1

16 bit Opcode:

| 1001 | 1000 | pppp | pbbb |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
\[
\text{cbi } \$12,7 \quad ; \text{Clear bit 7 in Port D}
\]

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:**
(i) \[ \text{Rd} \leftarrow \text{Rd} \times (\text{FF} - K) \]

**Syntax:**
(i) CBR Rd,K

**Operands:** 16 ≤ d ≤ 31, 0 ≤ K ≤ 255

**Program Counter:**
PC ← PC + 1

16 bit Opcode: See ANDI with K complemented.

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>0</td>
<td>⇔</td>
<td>⇔</td>
<td>-</td>
</tr>
</tbody>
</table>

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:**
```plaintext
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:
(i) $C \leftarrow 0$

Syntax: Operands: Program Counter:
(i) CLC None PC $\leftarrow$ PC + 1

16 bit Opcode:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0100</td>
<td>1000</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

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) \( H \leftarrow 0 \)

**Syntax:**
(i) CLH

**Operands:**
None

**Program Counter:**
\( \text{PC} \leftarrow \text{PC} + 1 \)

**16 bit Opcode:**
\[
\begin{array}{cccc}
1001 & 0100 & 1101 & 1000 \\
\end{array}
\]

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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).

Operation:
(i) \( I \leftarrow 0 \)

Syntax: Operands: Program Counter:
(i) CLI None PC \( \leftarrow \) PC + 1

16 bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0100</td>
<td>1111</td>
<td>1000</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I: 0
Global Interrupt flag cleared

Example:

```assembly
cli ; Disable interrupts
in r11,$16 ; Read port B
sei ; Enable interrupts
```

Words: 1 (2 bytes)
Cycles: 1
CLN - Clear Negative Flag

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

\[
\text{Operation:} \quad N \leftarrow 0
\]

Syntax: (i) CLN
Operands: None
Program Counter: PC \leftarrow PC + 1

16 bit Opcode:
\[
\begin{array}{cccc}
1001 & 0100 & 1010 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

\[
\begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
- & - & - & - & - & 0 & - & - \\
\end{array}
\]

N: 0
Negative flag cleared

Example:
\[
\begin{align*}
\text{add} & \quad \text{r2, r3} \quad ; \text{Add r3 to r2} \\
\text{cln} & \quad ; \text{Clear negative flag}
\end{align*}
\]

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:
(i) \( Rd \leftarrow Rd \oplus Rd \)

Syntax: Operands: Program Counter:
(i) CLR Rd \( 0 \leq d \leq 31 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode: (see EOR Rd,Rd)

\[
\begin{array}{cccc}
0010 & 01dd & dddd & dddd \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

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:
(i) \( S \leftarrow 0 \)

Syntax: Operands: Program Counter:
(i) CLS None PC \( \leftarrow \) PC + 1

16 bit Opcode:
```
1001 0100 1100 1000
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

\( S: 0 \)
Signed flag cleared

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

Words: 1 (2 bytes)
Cycles: 1
CLT - Clear T Flag

Description:
Clears the T flag in SREG (status register).

Operation:
(i) \( T \leftarrow 0 \)

Syntax: Operands: Program Counter:
(i) CLT None PC \( \leftarrow \) PC + 1

16 bit Opcode:
\[
\begin{array}{cccc}
1001 & 0100 & 1110 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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:
(i) V ← 0

Syntax: Operands: Program Counter:
(i) CLV None PC ← PC + 1

16 bit Opcode:

| 1001 | 0100 | 1011 | 1000 |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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:
(i) \( Z \leftarrow 0 \)

Syntax:
(i) CLZ

Operands:
None

Program Counter:
PC \( \leftarrow \) PC + 1

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 0100 & 1001 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

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:**
(i) \( \text{Rd} \leftarrow \$FF - \text{Rd} \)

**Syntax:**  
Operands: Program Counter:
(i) COM Rd \( 0 \leq d \leq 31 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

**16 bit Opcode:**
```
1001 010d  dddd  0000
```

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇐</td>
<td>0</td>
<td>⇐</td>
<td>⇐</td>
<td>1</td>
</tr>
</tbody>
</table>

S: \( \text{N} \oplus \text{V} \)  
For signed tests.

V: 0  
Cleared.

N: \( \text{R7} \)  
Set if MSB of the result is set; cleared otherwise.

Z: \( \overline{\text{R7}} \cdot \text{R6} \cdot \text{R5} \cdot \text{R4} \cdot \text{R3} \cdot \text{R2} \cdot \text{R1} \cdot \text{R0} \)  
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:
(i) Rd - Rr

Syntax: Operands: Program Counter:
(i) CP Rd, Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31 PC ← PC + 1

16 bit Opcode:

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: \( \overline{Rd3 \cdot Rr3} + Rr3 \cdot R3 + R3 \cdot Rd3 \)
Set if there was a borrow from bit 3; cleared otherwise

S: \( N \oplus V \), For signed tests.

V: \( Rd7 \cdot \overline{R7} \cdot \overline{R7} + Rd7 \cdot R7 \cdot R7 \)
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: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
Set if the result is $00; cleared otherwise.

C: \( \overline{Rd7 \cdot Rr7} + Rr7 \cdot R7 + R7 \cdot Rd7 \)
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:

```assembly
    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:
(i) \( \text{Rd} - \text{Rr} - \text{C} \)

Syntax: Operands: Program Counter:
(i) CPC Rd,Rr \( 0 \leq d \leq 31, 0 \leq r \leq 31 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

\[
\begin{array}{cccc}
0000 & 01rd & dddd & rrrr \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>⇓</td>
<td>⇓</td>
<td>⇓</td>
<td>⇓</td>
<td>⇓</td>
<td>⇓</td>
</tr>
</tbody>
</table>

H: \( \text{Rd}_3 \cdot \text{Rr}_3 + \text{Rr}_3 \cdot \text{R}_3 + \text{R}_3 \cdot \text{Rd}_3 \)
Set if there was a borrow from bit 3; cleared otherwise

S: \( \text{N} \oplus \text{V} \), For signed tests.

V: \( \text{Rd}_7 \cdot \overline{\text{Rr}_7} \cdot \overline{\text{R}_7} + \overline{\text{Rd}_7} \cdot \text{Rr}_7 \cdot \text{R}_7 \)
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: \( \text{R}_7 \)
Set if MSB of the result is set; cleared otherwise.

Z: \( \overline{\text{R}_7} \cdot \overline{\text{R}_6} \cdot \overline{\text{R}_5} \cdot \overline{\text{R}_4} \cdot \overline{\text{R}_3} \cdot \overline{\text{R}_2} \cdot \overline{\text{R}_1} \cdot \overline{\text{R}_0} \cdot \text{Z} \)
Previous value remains unchanged when the result is zero; cleared otherwise.

C: \( \text{Rd}_7 \cdot \text{Rr}_7 + \text{Rr}_7 \cdot \text{R}_7 + \text{R}_7 \cdot \text{Rd}_7 \)
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:

```plaintext
; 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:
(i) \( \text{Rd} - K \)

Syntax: Operands: Program Counter:
(i) CPI Rd,K \( 16 \leq d \leq 31, 0 \leq K \leq 255 \) PC \( \leftarrow \) PC + 1

16 bit Opcode:

\[
\begin{array}{cccc}
0011 & KKKK & dddd & KKKK \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

\[
\begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
\hline
\end{array}
\]

H: \( \overline{\text{Rd}} \cdot K + K \cdot R + R \cdot \overline{\text{Rd}} \)
Set if there was a borrow from bit 3; cleared otherwise

S: \( N \oplus V \), For signed tests.

V: \( \text{Rd} \cdot K \cdot R \cdot R + R \cdot K \cdot R \cdot R \)
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: \( R \)
Set if MSB of the result is set; cleared otherwise.

Z: \( R \cdot R \cdot R \cdot R \cdot R \cdot R \cdot R \cdot R \)
Set if the result is $00$; cleared otherwise.

C: \( \overline{\text{Rd}} \cdot K + K \cdot R + R \cdot \overline{\text{Rd}} \)
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:
\[
\begin{array}{l}
\text{cpi r19,3} ; \text{Compare r19 with 3} \\
\text{brne error} ; \text{Branch if r19}<3 \\
\ldots \\
\text{error: nop} ; \text{Branch destination (do nothing)}
\end{array}
\]

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:**
(i) If Rd = Rr then PC ← PC + 2 (or 3) else PC ← PC + 1

**Syntax:**
(i) CPSE Rd,Rr

**Operands:**
0 ≤ d ≤ 31, 0 ≤ r ≤ 31

**Program Counter:**
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 Formulae:**

```
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
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:
(i) \( \text{Rd} \leftarrow \text{Rd} - 1 \)

Syntax: Operands: Program Counter:
(i) DEC Rd \( 0 \leq d \leq 31 \)
\( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:
\[
\begin{array}{cccc}
1001 & 010d & dddd & 1010 \\
\end{array}
\]

Status Register and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>*</td>
</tr>
</tbody>
</table>

S: \( \text{N} \oplus \text{V} \)
For signed tests.

V: \( \text{R}_7 \cdot \text{R}_6 \cdot \text{R}_5 \cdot \text{R}_4 \cdot \text{R}_3 \cdot \text{R}_2 \cdot \text{R}_1 \cdot \text{R}_0 \)
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: \( \text{R}_7 \)
Set if MSB of the result is set; cleared otherwise.

Z: \( \text{R}_7 \cdot \text{R}_6 \cdot \text{R}_5 \cdot \text{R}_4 \cdot \text{R}_3 \cdot \text{R}_2 \cdot \text{R}_1 \cdot \text{R}_0 \)
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
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:**
(i) \( Rd \leftarrow Rd \oplus Rr \)

**Syntax:**
(i) EOR Rd,Rr

**Operands:**
0 \( \leq d \leq 31, \ 0 \leq r \leq 31 \)

**Program Counter:**
PC \( \leftarrow PC + 1 \)

**16 bit Opcode:**

| 0010 | 01rd | dddd | rrrr |

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇐</td>
<td>0</td>
<td>⇐</td>
<td>⇐</td>
<td>-</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.

V: 0
Cleared

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

Z: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
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
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 current 64K words (128K bytes) section in the program memory space.

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) is unchanged

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:

```
 1001 0101 XXXX 1001
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

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

Words: 1 (2 bytes)
Cycles: 3
**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 current 64K words (128K bytes) section of program memory.

**Operation:**
(i) \( \text{PC} \leftarrow Z(15 - 0) \)  
   Devices with 16 bits PC, 128K bytes program memory maximum.
(ii) \( \text{PC}(15-0) \leftarrow Z(15-0) \)  
    Devices with 22 bits PC, 8M bytes program memory maximum.
    PC(21-16) is unchanged

**Syntax:**
(i) IJMP
(ii) IJMP

**Operands:**
None

**Program Counter:**
See Operation

**Stack:**
Not Affected

**16 bit Opcode:**
```
1001 0100 XXXX 1001
```

**Status Register (SREG) and Boolean Formulae:**

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

**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 Port to Register

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

Operation:
(i) \( \text{Rd} \leftarrow \text{P} \)

Syntax: \( \text{IN Rd,P} \)
Operands: \( 0 \leq d \leq 31, 0 \leq P \leq 63 \)

Program Counter:
\( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

\[
\begin{array}{c|c|c|c|c|c|c|c}
\text{I} & \text{T} & \text{H} & \text{S} & \text{V} & \text{N} & \text{Z} & \text{C} \\
\hline
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

Example:
\[
\begin{array}{l}
in\ r25,\$16 \quad ; \text{Read Port B} \\
cpi\ r25,4 \quad ; \text{Compare read value to constant} \\
breq\ \text{exit} \quad ; \text{Branch if r25=4} \\
\ldots \\
exit: \quad \text{nop} \quad ; \text{Branch destination (do nothing)}
\end{array}
\]

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:**
\[(i) \quad \text{Rd} \leftarrow \text{Rd} + 1\]

**Syntax:**
\[(i) \quad \text{INC Rd} \quad 0 \leq d \leq 31\]

**Program Counter:**
\[\text{PC} \leftarrow \text{PC} + 1\]

**16 bit Opcode:**
\[
\begin{array}{cccc}
1001 & 010d & dddd & 0011 \\
\end{array}
\]

**Status Register and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
</table>

- \[S: \quad \text{N} \oplus \text{V}\]
  For signed tests.
- \[V: \quad R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0\]
  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.
- \[N: \quad R7\]
  Set if MSB of the result is set; cleared otherwise.
- \[Z: \quad R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0\]
  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.

Operation:
(i) PC ← k

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

32 bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>010k</td>
<td>kkkk</td>
<td>110k</td>
<td></td>
</tr>
<tr>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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 SRAM to Register using Index X**

**Description:**
Loads one byte indirect from SRAM to register. The SRAM location is pointed to by the X (16 bits) pointer register in the register file. Memory access is limited to the current SRAM page of 64K bytes. To access another SRAM page the RAMPX in register in the I/O area has to be changed.

The X pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the X pointer register.

**Using the X pointer:**

### Operation:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>Rd ← (X)</td>
<td>X: Unchanged</td>
</tr>
<tr>
<td>(ii)</td>
<td>Rd ← (X)</td>
<td>X ← X + 1</td>
</tr>
<tr>
<td>(iii)</td>
<td>X ← X - 1</td>
<td>Rd ← (X)</td>
</tr>
</tbody>
</table>

### Syntax:

<table>
<thead>
<tr>
<th></th>
<th>Operands:</th>
<th>Program Counter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>LD Rd, X</td>
<td>0 ≤ d ≤ 31</td>
</tr>
<tr>
<td>(ii)</td>
<td>LD Rd, X+</td>
<td>0 ≤ d ≤ 31</td>
</tr>
<tr>
<td>(iii)</td>
<td>LD Rd,-X</td>
<td>0 ≤ d ≤ 31</td>
</tr>
</tbody>
</table>

### 16 bit Opcode :

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>1100</td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>1101</td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>1110</td>
</tr>
</tbody>
</table>

### Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Example:

```asm
clr r27 ; Clear X high byte
ldi r26,$20 ; Set X low byte to $20
ld r0,X+ ; Load r0 with SRAM loc. $20 (X post inc)
ld r1,X ; Load r1 with SRAM loc. $21
ldi r26,$23 ; Set X low byte to $23
ld r2,X ; Load r2 with SRAM loc. $23
ld r3,-X ; Load r3 with SRAM loc. $22 (X pre dec)
```

**Words:** 1 (2 bytes)

**Cycles:** 2
LD (LDD) - Load Indirect from SRAM to Register using Index Y

Description:
Loads one byte indirect with or without displacement from SRAM to register. The SRAM location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current SRAM page of 64K bytes. To access another SRAM page the RAMPY register in the I/O area has to be changed.

The Y pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the Y pointer register.

Using the Y pointer:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Comment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Rd ← (Y)</td>
<td>Y: Unchanged</td>
</tr>
<tr>
<td>(ii) Rd ← (Y) Y ← Y + 1</td>
<td>Y: Post incremented</td>
</tr>
<tr>
<td>(iii) Y ← Y - 1 Rd ← (Y)</td>
<td>Y: Pre decremented</td>
</tr>
<tr>
<td>(iv) Rd ← (Y+q)</td>
<td>Y: Unchanged, q: Displacement</td>
</tr>
</tbody>
</table>

Using the Y pointer:

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>Operands:</th>
<th>Program Counter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) LD Rd, Y</td>
<td>0 ≤ d ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(ii) LD Rd, Y+</td>
<td>0 ≤ d ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(iii) LD Rd,-Y</td>
<td>0 ≤ d ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(iv) LDD Rd, Y+q</td>
<td>0 ≤ d ≤ 31, 0 ≤ q ≤ 63</td>
<td>PC ← PC + 1</td>
</tr>
</tbody>
</table>

16 bit Opcode :

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1000</td>
<td>000d</td>
<td>dddd</td>
</tr>
<tr>
<td>ii</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
</tr>
<tr>
<td>iii</td>
<td>10q0</td>
<td>qq0d</td>
<td>dddd</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

clr r29 ; Clear Y high byte
ldi r28,$20 ; Set Y low byte to $20
ld r0,Y+ ; Load r0 with SRAM loc. $20(Y post inc)
lr r1,Y ; Load r1 with SRAM loc. $21
ldi r28,$23 ; Set Y low byte to $23
ld r2,Y ; Load r2 with SRAM loc. $23
ld r3,-Y ; Load r3 with SRAM loc. $22(Y pre dec)
ldd r4,Y+2 ; Load r4 with SRAM loc. $24

Words: 1 (2 bytes)
Cycles: 2
LD (LDD) - Load Indirect From SRAM to Register using Index Z

Description:
Loads one byte indirectly with or without displacement from SRAM to register. The SRAM location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current SRAM page of 64K bytes. To access another SRAM page the RAMPZ register in the I/O area has to be changed.

The Z pointer register can either be left unchanged after the operation, or it can be incremented or 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.

For using the Z pointer for table lookup in program memory see the LPM instruction.

Using the Z pointer:

<table>
<thead>
<tr>
<th>Operation:</th>
<th>Comment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)  Rd ← (Z)</td>
<td>Z: Unchanged</td>
</tr>
<tr>
<td>(ii) Rd ← (Z) Z ← Z + 1</td>
<td>Z: Post increment</td>
</tr>
<tr>
<td>(iii) Z ← Z - 1 Rd ← (Z)</td>
<td>Z: Pre decrement</td>
</tr>
<tr>
<td>(iv) Rd ← (Z+q)</td>
<td>Z: Unchanged, q: Displacement</td>
</tr>
</tbody>
</table>

Syntax:

<table>
<thead>
<tr>
<th>Operands:</th>
<th>Program Counter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) LD Rd, Z</td>
<td>0 ≤ d ≤ 31 PC ← PC + 1</td>
</tr>
<tr>
<td>(ii) LD Rd, Z+</td>
<td>0 ≤ d ≤ 31 PC ← PC + 1</td>
</tr>
<tr>
<td>(iii) LD Rd, Z-</td>
<td>0 ≤ d ≤ 31 PC ← PC + 1</td>
</tr>
<tr>
<td>(iv) LDD Rd, Z+q</td>
<td>0 ≤ d ≤ 31, 0 ≤ q ≤ 63 PC ← PC + 1</td>
</tr>
</tbody>
</table>

16 bit Opcode:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Desc</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) 1000 000d dddd 0000</td>
<td></td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(ii) 1001 000d dddd 0001</td>
<td></td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(iii) 1001 000d dddd 0010</td>
<td></td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(iv) 10q0 qq0d dddd 0qqq</td>
<td></td>
<td>PC ← PC + 1</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

clr r31 ; Clear Z high byte
ldi r30, $20 ; Set Z low byte to $20
ld r0, Z+ ; Load r0 with SRAM loc. $20 (Z post inc)
ld r1, Z ; Load r1 with SRAM loc. $21
ldi r30, $23 ; Set Z low byte to $23
ld r2, Z ; Load r2 with SRAM loc. $23
ld r3, -Z ; Load r3 with SRAM loc. $22 (Z pre dec)
ldd r4, Z+2 ; Load r4 with SRAM loc. $24

Words: 1 (2 bytes)
Cycles: 2
LDI - Load Immediate

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

Operation:
(i) \( \text{Rd} \leftarrow K \)

Syntax: Operands: Program Counter:
(i) LDI Rd,K \( 16 \leq d \leq 31, 0 \leq K \leq 255 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

\[
\begin{array}{cccc}
1110 & KKKK & dddd & KKKK \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
clr r31 ; Clear Z high byte
ldi r30,$F0 ; Set Z low byte to $F0
lpm ; Load constant from program
\text{memory pointed to by Z}
```

Words: 1 (2 bytes)
Cycles: 1
LDS - Load Direct from SRAM

Description:
Loads one byte from the SRAM to a Register. A 16-bit address must be supplied. Memory access is limited to the current SRAM Page of 64K bytes. The LDS instruction uses the RAMPZ register to access memory above 64K bytes.

Operation:
(i) \( \text{Rd} \leftarrow (k) \)

Syntax:
(i) \( \text{LDS Rd,k} \)

Operands: \( 0 \leq d \leq 31, 0 \leq k \leq 65535 \)

Program Counter:
\( \text{PC} \leftarrow \text{PC} + 2 \)

32 bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>000d</th>
<th>dddd</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

lds r2,$FF00 ; Load r2 with the contents of SRAM location $FF00
add r2,r1 ; add r1 to r2
sts $FF00,r2 ; Write back

Words: 2 (4 bytes)
Cycles: 3
LPM - Load Program Memory

Description:
Loads one byte pointed to by the Z register into register 0 (R0). This instruction features a 100% space effective constant initialization or constant data fetch. The program memory is organized in 16 bits words and the LSB of the Z (16 bits) pointer selects either low byte (0) or high byte (1). This instruction can address the first 64K bytes (32K words) of program memory.

Operation:  Comment:
(i) R0 ← (Z)  Z points to program memory

Syntax:  Operands:  Program Counter:
(i) LPM  None  PC ← PC + 1

16 bit Opcode:

```
1001 0101 110X 1000
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
clr r31 ; Clear Z high byte
ldi r30,$F0 ; Set Z low byte
lpm ; Load constant from program
       ; memory pointed to by Z (r31:r30)
```

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 an unsigned value by two.

Operation:

\[
\begin{array}{c}
\text{C} \\
\downarrow
\end{array} 
\begin{array}{c}
\text{b7 - - - - - - - - - - - - - - - - b0} \\
\downarrow 0
\end{array}
\]

Syntax: LSL Rd
Operands: 0 ≤ d ≤ 31
Program Counter: PC ← PC + 1

16 bit Opcode: (see ADD Rd,Rd)

\[
0000 \quad 11dd \quad dddd \quad dddd
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: Rd3
S: N ⊕ V, For signed tests.
V: N ⊕ C (For N and C after the shift)
   Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of 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:

\[ 0 \rightarrow \overline{b7 - \cdots - b0} \rightarrow C \]

Syntax: Operands: Program Counter:
(i) LSR Rd 0 ≤ d ≤ 31 PC ← PC + 1

16 bit Opcode:
1001 010d dddd 0110

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⇔</td>
<td>⇔</td>
<td>0</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

S: \(N \oplus V\), For signed tests.

V: \(N \oplus C\) (For \(N\) and \(C\) after the shift)
Set if \((N\) is set and \(C\) is clear\) or \((N\) is clear and \(C\) is set\); Cleared otherwise (for values of \(N\) and \(C\) after the shift).

N: 0

Z: \(\overline{R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0}\)
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) \(Rd \leftarrow Rr\)

Syntax:
(i) MOV Rd,Rr

Operands:
0 \(\leq d \leq 31\), 0 \(\leq r \leq 31\)

Program Counter:
PC \(\leftarrow\) PC + 1

16 bit Opcode:
```
0010 11rd dddd rrrr
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

Words: 1 (2 bytes)
Cycles: 1
MUL - Multiply

**Description:**
This instruction performs 8-bit × 8-bit → 16-bit unsigned multiplication.

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>×</th>
<th>Multiplier</th>
<th>→</th>
<th>Product High</th>
<th>Product Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rr</td>
<td>8</td>
<td>Rd</td>
<td></td>
<td>R1</td>
<td>R0</td>
</tr>
</tbody>
</table>

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

**Operation:**
(i) \( R1,R0 \leftarrow Rr \times Rd \)

**Syntax:**
(i) MUL Rd,Rr

**Operands:**
0 ≤ d ≤ 31, 0 ≤ r ≤ 31

**Program Counter:**
PC ← PC + 1

**16 bit Opcode:**

1001 11rd dddd rrrr

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>⇔</td>
</tr>
</tbody>
</table>

C: \( R15 \)
Set if bit 15 of the result is set; cleared otherwise.

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

**Example:**
mul r6,r5 ; Multiply r6 and r5
mov r6,r1 ; Copy result back in r6:r5
mov r5,r0 ; Copy result back in r6:r5

**Words:** 1 (2 bytes)

**Cycles:** 2

Not available in base-line microcontrollers.
NEG - Two’s Complement

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

Operation:
(i) \( Rd \leftarrow 00 - Rd \)

Syntax: 
(i) NEG Rd

Operands: \( 0 \leq d \leq 31 \)

Program Counter:
PC \( \leftarrow \) PC + 1

16 bit Opcode:

\[
\begin{array}{cccccc}
1001 & 010d & dddd & 0001 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: \( R3 \cdot \overline{Rd3} \)
Set if there was a borrow from bit 3; cleared otherwise

S: \( N \oplus V \)
For signed tests.

V: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
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: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
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:

\[
\begin{array}{cccccccc}
0000 & 0000 & 0000 & 0000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>clr r16</td>
<td>Clear r16</td>
</tr>
<tr>
<td>ser r17</td>
<td>Set r17</td>
</tr>
<tr>
<td>out $18,r16</td>
<td>Write zeros to Port B</td>
</tr>
<tr>
<td>nop</td>
<td>Wait (do nothing)</td>
</tr>
<tr>
<td>out $18,r17</td>
<td>Write ones to Port B</td>
</tr>
</tbody>
</table>

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) \( \text{Rd} \leftarrow \text{Rd} \lor \text{Rr} \)

Syntax: Operands: Program Counter:
(i) OR Rd, Rr \( 0 \leq d \leq 31, 0 \leq r \leq 31 \) PC \( \leftarrow \) PC + 1

16 bit Opcode:
\[
\begin{array}{cccc}
0010 & 10rd & dddd & rrrr \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>0</td>
<td>⇔</td>
<td>⇔</td>
<td>-</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.

V: 0
Cleared

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

Z: \( \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0} \)
Set if the result is $00$; cleared otherwise.

R (Result) equals Rd after the operation.

Example:
\[
\begin{align*}
or & \quad r15, r16 \quad ; \text{Do bitwise or between registers} \\
\text{bst} & \quad r15, 6 \quad ; \text{Store bit 6 of r15 in T flag} \\
\text{brts} & \quad \text{ok} \quad ; \text{Branch if T flag set} \\
\ldots & \quad \\
\text{ok:} & \quad \text{nop} \quad ; \text{Branch destination (do nothing)}
\end{align*}
\]

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:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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,$FO ; 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 port

Description:
Stores data from register Rr in the register file to I/O space (Ports, Timers, Configuration registers etc.).

Operation:
(i) \( P \leftarrow Rr \)

Syntax: OUT P,Rr
Operands: \( 0 \leq r \leq 31, 0 \leq P \leq 63 \)
Program Counter: \( PC \leftarrow PC + 1 \)

16 bit Opcode:

| 1011 | P | rrrr | PPPP |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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.

Operation:
(i) Rd ← STACK

Syntax:  Operands:  Program Counter:  Stack
(i) POP Rd  0 ≤ d ≤ 31  PC ← PC + 1  SP ← SP + 1

16 bit Opcode:
1001 000d dddd 1111

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
call routine ; 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
**PUSH - Push Register on Stack**

**Description:**
This instruction stores the contents of register Rr on the STACK.

**Operation:**
(i) \( \text{STACK} \leftarrow \text{Rr} \)

**Syntax:**
(i) \( \text{PUSH Rr} \quad 0 \leq r \leq 31 \quad \text{PC} \leftarrow \text{PC} + 1 \quad \text{SP} \leftarrow \text{SP} - 1 \)

**16 bit Opcode:**
```
1001 001d dddd 1111
```

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Example:**
```
call routine ; 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:**
Calls a subroutine within ±2K words (4K bytes). The return address (the instruction after the RCALL) is stored onto the stack. (See also CALL).

**Operation:**
(i) \( PC \leftarrow PC + k + 1 \) Devices with 16 bits PC, 128K bytes program memory maximum.
(ii) \( PC \leftarrow PC + k + 1 \) Devices with 22 bits PC, 8M bytes program memory maximum.

**Syntax:**
(i) RCALL \( k \) \(-2K \leq k \leq 2K\)
(ii) RCALL \( k \) \(-2K \leq k \leq 2K\)

**Operands:**
(i) \( PC \leftarrow PC + k + 1 \)
(ii) \( PC \leftarrow PC + k + 1 \)

**Program Counter:**
(i) \( PC \leftarrow PC + k + 1 \)
(ii) \( PC \leftarrow PC + k + 1 \)

**Stack:**
(i) STACK ← PC+1
SP ← SP-2 (2 bytes, 16 bits)
(ii) STACK ← PC+1
SP ← SP-3 (3 bytes, 22 bits)

**16 bit Opcode:**

```
1101 kkkk kkkk kkkk
```

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**
```assembly
rcall routine ; Call subroutine
... routine: push r14 ; Save r14 on the stack
... pop r14 ; Restore r14
ret ; Return from subroutine
```

**Words:** 1 (2 bytes)
**Cycles:** 3
RET - Return from Subroutine

Description:
Returns from subroutine. The return address is loaded from the STACK.

Operation:
(i) \( \text{PC}(15-0) \leftarrow \text{STACK} \) Devices with 16 bits PC, 128K bytes program memory maximum.
(ii) \( \text{PC}(21-0) \leftarrow \text{STACK} \) Devices with 22 bits PC, 8M bytes program memory maximum.

Syntax: Operands: Program Counter: Stack

(i) RET None See Operation \( \text{SP} \leftarrow \text{SP} +2, \) (2 bytes, 16 bits pulled)
(ii) RET None See Operation \( \text{SP} \leftarrow \text{SP} +3, \) (3 bytes, 22 bits pulled)

16 bit Opcode:

| 1001 | 0101 | 0XX0 | 1000 |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
call routine ; Call subroutine
...routine: push r14 ; Save r14 on the stack
...pop r14 ; Restore r14
ret ; Return from subroutine

Words: 1 (2 bytes)
Cycles: 4
RET I - Return from Interrupt

Description:
Returns from interrupt. The return address is loaded from the STACK and the global interrupt flag is set.

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) 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:

\[
\begin{array}{cccc}
1001 & 0101 & 0XX1 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

\[
\begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
1 & - & - & - & - & - & - & - \\
\end{array}
\]

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
RJMP - Relative Jump

Description:
Relative jump to an address within PC-2K 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) \( PC \leftarrow PC + k + 1 \)

Syntax: Operands: Program Counter: Stack
(i) RJMP k -2K \( \leq k \leq 2K \) PC \( \leftarrow PC + k + 1 \) Unchanged

16 bit Opcode:
\[
\begin{array}{cccc}
1100 & kkkk & kkkk & kkkk \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
- `cpi r16,$42` ; Compare r16 to $42
- `brne error` ; Branch if r16 <> $42
- `rjmp ok` ; Unconditional branch
- `error:` `add r16,r17` ; Add r17 to r16
- `inc r16` ; Increment r16
- `ok:` `nop` ; Destination for rjmp (do nothing)

Words: 1 (2 bytes)
Cycles: 2
**ROL - Rotate Left through 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.

**Operation:**

\[
\begin{array}{c}
C \\
\downarrow \quad \downarrow \\
\Rightarrow \quad \Rightarrow \\
\end{array}
\]

**Syntax:**
(i) ROL Rd

**Operands:**
0 ≤ d ≤ 31

**Program Counter:**
PC ← PC + 1

**16 bit Opcode:** (see ADC Rd,Rd)

\[
\begin{array}{c}
0001 \\
1idd \\
dddd \\
dddd
\end{array}
\]

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: Rd3

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of 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:**

\[
\begin{array}{c}
\text{rol} \\
\text{r15} \quad \quad \text{; Rotate left} \\
\text{brcs} \quad \text{oneenc} \quad \text{; Branch if carry set} \\
\ldots \\
\text{oneenc:} \quad \text{nop} \quad \text{; Branch destination (do nothing)}
\end{array}
\]

**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.

Operation:

\[
\begin{array}{c}
C \\
\rightarrow \\
\rightarrow
\end{array}
\]

Syntax: Operands: Program Counter:
(i) ROR Rd 0 ≤ d ≤ 31 PC ← PC + 1

16 bit Opcode:

\[
1001 \ 010d \ dddd \ 0111
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).

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

Z: \(\overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}\)
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:

```
ror  r15 ; Rotate right
brcr  zeroenc ; Branch if carry cleared
... zeroenc:  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) \( \text{Rd} \leftarrow \text{Rd} - \text{Rr} - \text{C} \)

Syntax: Operands: Program Counter:
(i) \( \text{SBC Rd,Rr} \) \( 0 \leq d \leq 31, \ 0 \leq r \leq 31 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Status Register and Boolean Formulae:

H: \( \overline{\text{Rd}}_3 \cdot \overline{\text{Rr}}_3 + \text{Rr}_3 \cdot \overline{\text{R3}} + \text{R3} \cdot \overline{\text{Rd}}_3 \)
Set if there was a borrow from bit 3; cleared otherwise

S: \( \text{N} \oplus \text{V} \), For signed tests.

V: \( \text{Rd}_7 \cdot \overline{\text{Rr}}_7 \cdot \overline{\text{R7}} + \overline{\text{Rd}}_7 \cdot \text{Rr}_7 \cdot \text{R7} \)
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: \( \text{R7} \)
Set if MSB of the result is set; cleared otherwise.

Z: \( \overline{\text{R7}} \cdot \overline{\text{R6}} \cdot \overline{\text{R5}} \cdot \overline{\text{R4}} \cdot \overline{\text{R3}} \cdot \overline{\text{R2}} \cdot \overline{\text{R1}} \cdot \overline{\text{R0}} \cdot \text{Z} \)
Previous value remains unchanged when the result is zero; cleared otherwise.

C: \( \overline{\text{Rd}}_7 \cdot \text{Rr}_7 \cdot \overline{\text{Rr}}_7 \cdot \text{R7} + \text{R7} \cdot \overline{\text{Rd}}_7 \)
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:

```
sub r2,r0 ; Subtract r1:r0 from r3:r2
sbc r3,r1 ; Subtract low byte
             ; Subtract with carry high byte
```

Words: 1 (2 bytes)
Cycles: 1
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) \( \text{Rd} \leftarrow \text{Rd} - K - C \)

Syntax: Operands: Program Counter:
(i) \( \text{SBCI Rd,K} \) \( 16 \leq d \leq 31, 0 \leq K \leq 255 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:
\[
\begin{array}{cccc}
0 & 1 & 0 & 0 & K & K & K & K & d & d & d & d & K & K & K
\end{array}
\]

Status Register and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: \( \overline{\text{Rd}3} \cdot K3 + K3 \cdot R3 + R3 \cdot \overline{\text{Rd}3} \)
Set if there was a borrow from bit 3; cleared otherwise

S: \( \overline{\text{N}} \oplus \overline{\text{V}}, \text{For signed tests.} \)

V: \( \text{Rd}7 \cdot \overline{\text{K7}} \cdot \overline{\text{R7}} + \text{Rd}7 \cdot \text{K7} \cdot \text{R7} \)
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: \( \overline{\text{R7}}, \text{Set if MSB of the result is set; cleared otherwise.} \)

Z: \( \overline{\text{R7}} \cdot \overline{\text{R6}} \cdot \overline{\text{R5}} \cdot \overline{\text{R4}} \cdot \overline{\text{R3}} \cdot \overline{\text{R2}} \cdot \overline{\text{R1}} \cdot \overline{\text{R0}} \cdot \overline{\text{Z}} \)
Previous value remains unchanged when the result is zero; cleared otherwise.

C: \( \overline{\text{Rd}7} \cdot \overline{\text{K7}} + \text{K7} \cdot \overline{\text{R7}} + \text{R7} \cdot \overline{\text{Rd}7} \)
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:
```
subi r16,$23     ; Subtract $4F23 from r17:rl6
sbci r17,$4F     ; Subtract with carry high byte
```

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) \( I/O(P,b) \leftarrow 1 \)

Syntax: \( \text{SBI } P,b \)  
Operands: \( 0 \leq P \leq 31, 0 \leq b \leq 7 \)  
Program Counter: \( PC \leftarrow PC + 1 \)

16 bit Opcode: \[
\begin{array}{cccc}
1001 & 1010 & pppp & pbbb \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:
\[
\begin{align*}
\text{out } & \$1E, r0 ; \text{Write EEPROM address} \\
\text{sbi } & \$1C, 0 ; \text{Set read bit in EECR} \\
\text{in } & r1, \$1D ; \text{Read EEPROM data}
\end{align*}
\]

Words: 1 (2 bytes)  
Cycles: 2
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(P,b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) SBIC P,b 0 ≤ P ≤ 31, 0 ≤ b ≤ 7 PC ← PC + 1, If condition is false, no skip.

PC ← PC + 2, If next instruction is one word.
PC ← PC + 3, If next instruction is JMP or CALL

16 bit Opcode:
1001 1001 pppp pbbb

Status Register (SREG) and Boolean Formulae:

\[
\begin{array}{cccccccc}
I & T & H & S & V & N & Z & C \\
- & - & - & - & - & - & - & -
\end{array}
\]

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: 2 if condition is false (no skip)
3 if condition is true (skip is executed)
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(P,b) = 1 then PC ← PC + 2 (or 3) else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) SBIS P.b 0 ≤ P ≤ 31, 0 ≤ b ≤ 7 PC ← PC + 1, Condition false - no skip
PC ← PC + 2, Skip a one word instruction
PC ← PC + 3, Skip a JMP or a CALL

16 bit Opcode:
```
1001 1011 pppp pbbb
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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: 2 if condition is false (no skip)
        3 if condition is true (skip is executed)
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.

Operation:
(i) \( \text{Rdh:Rdl} \leftarrow \text{Rdh:Rdl} - K \)

Syntax: Operands: Program Counter:
(i) \( \text{SBIW Rdl,K} \) \( dl \in \{24,26,28,30\}, 0 \leq K \leq 63 \) \( PC \leftarrow PC + 1 \)

16 bit Opcode:

| 1001  | 0111  | KKdd | KKKK |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.

V: \( \text{Rdh}7 \cdot \text{R15} \)
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: \( \text{R15} \)
Set if MSB of the result is set; cleared otherwise.

Z: \( \text{R15} \cdot \text{R14} \cdot \text{R13} \cdot \text{R12} \cdot \text{R11} \cdot \text{R10} \cdot \text{R9} \cdot \text{R8} \cdot \text{R7} \cdot \text{R6} \cdot \text{R5} \cdot \text{R4} \cdot \text{R3} \cdot \text{R2} \cdot \text{R1} \cdot \text{R0} \)
Set if the result is $0000; cleared otherwise.

C: \( \text{R15} \cdot \text{Rdh}7 \)
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:
\[
\text{sbiw r24,1} \quad \text{; Subtract 1 from r25:r24} \\
\text{sbiw r28,63} \quad \text{; Subtract 63 from the Y pointer(r29:r28)}
\]

Words: 1 (2 bytes)
Cycles: 2
SBR - Set Bits in Register

Description:
Sets specified bits in register Rd. Performs the logical ORI between the contents of register Rd and a constant mask K and places the result in the destination register Rd.

Operation:
(i) \( \text{Rd} \leftarrow \text{Rd} \lor K \)

Syntax: Operands: Program Counter:
(i) SBR Rd,K \( 16 \leq d \leq 31, \ 0 \leq K \leq 255 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>KKKK</th>
<th>dddd</th>
<th>KKKK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>0</td>
<td>⇔</td>
<td>⇔</td>
<td>-</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.

V: 0

Cleared

N: \( R7 \)

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

Z: \( \overline{R7} \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)

Set if the result is \( \$00 \); cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```assembly
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, If condition is false, no skip.
PC ← PC + 2, If next instruction is one word.
PC ← PC + 3, If next instruction is JMP or CALL

16 bit Opcode:

```
1111 110r rrrr Xbbb
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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)
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 JMP or a CALL

16 bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>r</td>
<td>r</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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)
SEC - Set Carry Flag

Description:
Sets the Carry flag (C) in SREG (status register).

Operation:
(i) \( C \leftarrow 1 \)

Syntax: \( \text{SEC} \)  
Operands: None  
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

\[
\begin{array}{cccc}
  1 & 0 & 0 & 1 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  1 & 0 & 0 & 0 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

C: 1  
Carry flag set

Example:

\[
\text{sec} \quad ; \text{Set carry flag} \\
\text{adc} \ r0, r1 \quad ; r0 = r0 + r1 + 1
\]

Words: 1 (2 bytes)  
Cycles: 1
SEH - Set Half Carry Flag

Description:
Sets the Half Carry (H) in SREG (status register).

Operation:
(i) \( H \leftarrow 1 \)

Syntax: SEH
Operands: None
Program Counter:
\( PC \leftarrow PC + 1 \)

16 bit Opcode:

\[
\begin{array}{c|c|c|c|c}
1001 & 0100 & 0101 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

H: 1
Half Carry flag set

Example:

\[ \text{seh} \quad ; \text{Set Half Carry flag} \]

Words: 1 (2 bytes)
Cycles: 1
SEI - Set Global Interrupt Flag

Description:
Sets the Global Interrupt flag (I) in SREG (status register).

Operation:
(i) \( I \leftarrow 1 \)

Syntax: Operands: Program Counter:
(i) SEI None PC \( \leftarrow \) PC + 1

16 bit Opcode:
```
0100 0100 0111 1000
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

I: 1
Global Interrupt flag set

Example:
```
cli ; Disable interrupts
in r13,$16 ; Read Port B
sei ; Enable interrupts
```

Words: 1 (2 bytes)
Cycles: 1
SEN - Set Negative Flag

Description:
Sets the Negative flag (N) in SREG (status register).

Operation:
(i) \( N \leftarrow 1 \)

Syntax: Operands: Program Counter:
(i) SEN None \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:
\[
\begin{array}{cccc}
1001 & 0100 & 0010 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

N: 1
Negative flag set

Example:
\[
\begin{align*}
\text{add r2,r19} & \quad ; \text{Add r19 to r2} \\
\text{sen} & \quad ; \text{Set negative flag}
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 1
SER - Set all bits in Register

Description:
Loads $FF$ directly to register Rd.

Operation:
(i) \( \text{Rd} \leftarrow \text{SFF} \)

Syntax: \( \text{SER Rd} \)
Operands: \( 16 \leq d \leq 31 \)
Program Counter:
\( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:
\[
\begin{array}{cccc}
1110 & 1111 & \text{dddd} & 1111 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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) \( S \leftarrow 1 \)

Syntax: \( \text{SES} \)  
Operands: None  
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 0100 & 0100 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

S: 1
Signed flag set

Example:

\begin{verbatim}
add r2, r19  ; Add r19 to r2
ses          ; Set negative flag
\end{verbatim}

Words: 1 (2 bytes)  
Cycles: 1
### SET - Set T Flag

**Description:**
Sets the T flag in SREG (status register).

**Operation:**

(i) \( T \leftarrow 1 \)

**Syntax:**

(i) SET

**Operands:** None

**Program Counter:**

\( PC \leftarrow PC + 1 \)

**16 bit Opcode:**

| 1001 | 0100 | 0110 | 1000 |

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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) \( V \leftarrow 1 \)

Syntax: SEV
Operands: None
Program Counter: PC \( \leftarrow \) PC + 1

16 bit Opcode:

| 1001 | 0100 | 0011 | 1000 |

Status Register (SREG) and Boolean Formulae:

\[
\begin{array}{ccccccccc}
I & T & H & S & V & N & Z & C \\
- & - & - & - & 1 & - & - & -
\end{array}
\]

V: 1
Overflow flag set

Example:

\[
\begin{align*}
\text{add r2,r19} & \quad ; \text{Add r19 to r2} \\
\text{sev} & \quad ; \text{Set overflow flag}
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 1
SEZ - Set Zero Flag

Description:
Sets the Zero flag (Z) in SREG (status register).

Operation:
(i) \( Z \leftarrow 1 \)

Syntax: \( \text{SEZ} \)
Operands: None
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:
\[
\begin{array}{cccc}
1001 & 0100 & 0001 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

\( Z: 1 \)
Zero flag set

Example:
\[
\begin{align*}
\text{add r2, r19} & \quad ; \text{Add r19 to r2} \\
\text{sez} & \quad ; \text{Set zero flag}
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 1
SLEEP

Description:
This instruction sets the circuit in sleep mode defined by the MCU control register. When an interrupt wakes up the MCU from a sleep state, the instruction following the SLEEP instruction will be executed before the interrupt handler is executed.

Operation:

Syntax: Operands: Program Counter:
SLEEP None PC ← PC + 1

16 bit Opcode:

| 1001 | 0101 | 100X | 1000 |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
mov r0,r11 ; Copy r11 to r0
sleep       ; Put MCU in sleep mode
```

Words: 1 (2 bytes)
Cycles: 1
**ST - Store Indirect From Register to SRAM using Index X**

**Description:**
Stores one byte indirect from Register to SRAM. The SRAM location is pointed to by the X (16 bits) pointer register in the register file. Memory access is limited to the current SRAM Page of 64K bytes. To access another SRAM page the RAMPX register in the I/O area has to be changed.

The X pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for stack pointer usage of the X pointer register.

**Using the X pointer:**

<table>
<thead>
<tr>
<th>Operation:</th>
<th>Comment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) (X) ← Rr</td>
<td>X: Unchanged</td>
</tr>
<tr>
<td>(ii) (X) ← Rr X ← X+1</td>
<td>X: Post incremented</td>
</tr>
<tr>
<td>(iii) X ← X - 1 (X) ← Rr</td>
<td>X: Pre decremented</td>
</tr>
</tbody>
</table>

**Syntax:**

| (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:**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) 1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>(ii) 1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1101</td>
<td></td>
</tr>
<tr>
<td>(iii) 1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1110</td>
<td></td>
</tr>
</tbody>
</table>

**Status Register (SREG) and Boolean Formulae:**

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

```plaintext
clr r27 ; Clear X high byte
ldi r26,$20 ; Set X low byte to $20
st X+,r0 ; Store r0 in SRAM loc. $20(X post inc)
st X,r1 ; Store r1 in SRAM loc. $21
ldi r26,$23 ; Set X low byte to $23
st r2,X ; Store r2 in SRAM loc. $23
st r3,-X ; Store r3 in SRAM loc. $22(X pre dec)
```

**Words:** 1 (2 bytes)

**Cycles:** 2
ST (STD) - Store Indirect From Register to SRAM using Index Y

Description:
Stores one byte indirect with or without displacement from Register to SRAM. The SRAM location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current SRAM Page of 64K bytes. To access another SRAM page the RAMPY register in the I/O area has to be changed.

The Y pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for stack pointer usage of the Y pointer register.

Using the Y pointer:

<table>
<thead>
<tr>
<th>Operation:</th>
<th>Comment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) (Y) ← Rr</td>
<td>Y: Unchanged</td>
</tr>
<tr>
<td>(ii) (Y) ← Rr Y ← Y+1</td>
<td>Y: Post incremented</td>
</tr>
<tr>
<td>(iii) Y ← Y - 1 (Y) ← Rr</td>
<td>Y: Pre decremented</td>
</tr>
<tr>
<td>(iv) (Y+q) ← Rr</td>
<td>Y: Unchanged, q: Displacement</td>
</tr>
</tbody>
</table>

Syntax: Operands: Program Counter:

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>Operands:</th>
<th>Program Counter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) ST Y, Rr</td>
<td>0 ≤ r ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(ii) ST Y+, Rr</td>
<td>0 ≤ r ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(iii) ST -Y, Rr</td>
<td>0 ≤ r ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(iv) STD Y+q, Rr</td>
<td>0 ≤ r ≤ 31, 0 ≤ q ≤ 63</td>
<td>PC ← PC + 1</td>
</tr>
</tbody>
</table>

16 bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1000</td>
<td>001r</td>
<td>rrrr</td>
<td>1000</td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1001</td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1010</td>
</tr>
<tr>
<td>(iv)</td>
<td>10q0</td>
<td>qqlr</td>
<td>rrrr</td>
<td>1qqq</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
clr r29 ; Clear Y high byte
ldi r28,$20 ; Set Y low byte to $20
st Y+,r0 ; Store r0 in SRAM loc. $20(Y post inc)
st Y,r1 ; Store r1 in SRAM loc. $21
ldi r28,$23 ; Set Y low byte to $23
st Y,r2 ; Store r2 in SRAM loc. $23
st -Y,r3 ; Store r3 in SRAM loc. $22(Y pre dec)
std Y+2,r4 ; Store r4 in SRAM loc. $24
```

Words: 1 (2 bytes)
Cycles: 2
ST (STD) - Store Indirect From Register to SRAM using Index Z

Description:
Stores one byte indirect with or without displacement from Register to SRAM. The SRAM location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current SRAM Page of 64K bytes. To access another SRAM page the RAMPZ register in the I/O area has to be changed.

The Z pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are very suited for stack pointer usage of the Z pointer register, but 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.

Using the Z pointer:

<table>
<thead>
<tr>
<th>Operation:</th>
<th>Comment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) (Z) ← Rr</td>
<td>Z: Unchanged</td>
</tr>
<tr>
<td>(ii) (Z) ← Rr</td>
<td>Z ← Z+1</td>
</tr>
<tr>
<td>(iii) Z ← Z - 1</td>
<td>(Z) ← Rr</td>
</tr>
<tr>
<td>(iv) (Z+q) ← Rr</td>
<td>Z: Unchanged, q: Displacement</td>
</tr>
</tbody>
</table>

Syntax:

<table>
<thead>
<tr>
<th>Program Counter:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) ST Z, Rr</td>
<td>0 ≤ r ≤ 31</td>
</tr>
<tr>
<td>(ii) ST Z+, Rr</td>
<td>0 ≤ r ≤ 31</td>
</tr>
<tr>
<td>(iii) ST -Z, Rr</td>
<td>0 ≤ r ≤ 31</td>
</tr>
<tr>
<td>(iv) STD Z+q, Rr</td>
<td>0 ≤ r ≤ 31, 0 ≤ q ≤ 63</td>
</tr>
</tbody>
</table>

16 bit Opcode:

<table>
<thead>
<tr>
<th>Operation:</th>
<th>Comment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) 1000 001r rrrr 0000</td>
<td></td>
</tr>
<tr>
<td>(ii) 1001 001r rrrr 0001</td>
<td></td>
</tr>
<tr>
<td>(iii) 1001 001r rrrr 0010</td>
<td></td>
</tr>
<tr>
<td>(iv) 10q0 qq1r rrrr 0qqq</td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

clr r31 ; Clear Z high byte
ldi r30,$20 ; Set Z low byte to $20
st Z+,r0 ; Store r0 in SRAM loc. $20(Z post inc)
st Z,r1 ; Store r1 in SRAM loc. $21
ldi r30,$23 ; Set Z low byte to $23
st Z,r2 ; Store r2 in SRAM loc. $23
st -Z,r3 ; Store r3 in SRAM loc. $22(Z pre dec)
std Z+2,r4 ; Store r4 in SRAM loc. $24

Words: 1 (2 bytes)
Cycles: 2
STS - Store Direct to SRAM

Description:
Stores one byte from a Register to the SRAM. A 16-bit address must be supplied. Memory access is limited to the current SRAM Page of 64K bytes. The SDS instruction uses the RAMPZ register to access memory above 64K bytes.

Operation:
(i) \( (k) \leftarrow Rr \)

Syntax: 
(i) STS k,Rr

Operands: 
0 \( \leq r \leq 31 \), 0 \( \leq k \leq 65535 \)

Program Counter:
PC \( \leftarrow \) PC + 2

32 bit Opcode:

<table>
<thead>
<tr>
<th>1001</th>
<th>00ld</th>
<th>dddd</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

```
lds r2,$FF00 ; Load r2 with the contents of SRAM location $FF00
add r2,r1 ; add r1 to r2
sts $FF00,r2 ; Write back
```

Words: 2 (4 bytes)
Cycles: 3
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:
```
0001 10rd dddd rrrr
```

Status Register and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: \( \overline{Rd3} \cdot \overline{Rr3} + \overline{Rr3} \cdot \overline{R3} + R3 \cdot \overline{Rd3} \)
Set if there was a borrow from bit 3; cleared otherwise

S: \( N \oplus V \), For signed tests.

V: \( Rd7 \cdot \overline{Rr7} \cdot \overline{R7} + \overline{Rd7} \cdot Rr7 \cdot R7 \)
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: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
Set if the result is \(00\); cleared otherwise.

C: \( Rd7 \cdot Rr7 + Rr7 \cdot R7 + \overline{R7} \cdot \overline{Rd7} \)
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:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

H: Rd^3 ∙ K^3 + K^3 ∙ R^3 + R^3 ∙ Rd^3
Set if there was a borrow from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: Rd^7 ∙ K^7 ∙ R^7 + Rd^7 ∙ K^7 ∙ R^7
Set if two’s complement overflow resulted from the operation; cleared otherwise.

N: R^7
Set if MSB of the result is set; cleared otherwise.

Z: R^7 ∙ R^6 ∙ R^5 ∙ R^4 ∙ R^3 ∙ R^2 ∙ R^1 ∙ R^0
Set if the result is $00; cleared otherwise.

C: Rd^7 ∙ K^7 + K^7 ∙ R^7 + R^7 ∙ Rd^7
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) \( R(7-4) \leftarrow R(3-0), R(3-0) \leftarrow R(7-4) \)

Syntax: Operands: Program Counter:
(i) SWAP Rd \( 0 \leq d \leq 31 \) PC \( \leftarrow PC + 1 \)

16 bit Opcode:

\[
\begin{array}{c|c|c|c|c|}
\text{I} & \text{T} & \text{H} & \text{S} & \text{V} & \text{N} & \text{Z} & \text{C} \\
\hline
\end{array}
\]

R (Result) equals Rd after the operation.

Example:

\[
\begin{align*}
\text{inc} & \ r1 \quad ; \text{Increment r1} \\
\text{swap} & \ r1 \quad ; \text{Swap high and low nibble of r1} \\
\text{inc} & \ r1 \quad ; \text{Increment high nibble of r1} \\
\text{swap} & \ r1 \quad ; \text{Swap back}
\end{align*}
\]

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) \( \text{Rd} \leftarrow \text{Rd} \cdot \text{Rd} \)

Syntax: Operands: Program Counter:
(i) TST Rd \( 0 \leq d \leq 31 \) PC \( \leftarrow \) PC + 1

16 bit Opcode:

| Opcode | 0010 | 00dd | dddd | dddd |

Status Register and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>0</td>
<td>⇔</td>
<td>⇔</td>
<td>-</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.

V: 0
Cleared

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

Z: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
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:

| 1001 | 0101 | 101X | 1000 |

Status Register and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Example:

wdr ; Reset watchdog timer

Words: 1 (2 bytes)
Cycles: 1