Instruction Set Nomenclature:

Status Register (SREG)

SREG: Status register
C: Carry flag in status register
Z: Zero flag in status register
N: Negative flag in status register
V: Two’s complement overflow indicator
S: N ⊕ V, For signed tests
H: Half Carry flag in the status register
T: Transfer bit used by BLD and BST instructions
I: Global interrupt enable/disable flag

Registers and Operands

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
R: Result after instruction is executed
K: Constant data
k: Constant address
b: Bit in the register file or I/O register (3 bit)
s: Bit in the status register (3 bit)
X,Y,Z: Indirect address register
   (X=R27:R26, Y=R29:R28 and Z=R31:R30)
A: I/O location address
q: Displacement for direct addressing (6 bit)
I/O Registers

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

**RAMPD**
Register concatenated with the Z register enabling direct addressing of the whole data space on MCUs with more than 64K bytes data space.

**EIND**
Register concatenated with the instruction word enabling indirect jump and call to the whole program space on MCUs with more than 64K bytes program space.

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

**Flags**
\[\leftrightarrow\]: 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 ⊕ V) = 0</td>
<td>BRLT(1)</td>
<td>Rd ≤ Rr</td>
<td>Z*(N ⊕ V) = 1</td>
<td>BRGE*</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd ≥ Rr</td>
<td>(N ⊕ V) = 0</td>
<td>BRGE</td>
<td>Rd &lt; Rr</td>
<td>(N ⊕ 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 ⊕ V) = 1</td>
<td>BRGE(1)</td>
<td>Rd &gt; Rr</td>
<td>Z*(N ⊕ V) = 0</td>
<td>BRLT*</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd &lt; Rr</td>
<td>(N ⊕ V) = 1</td>
<td>BRLT</td>
<td>Rd ≥ Rr</td>
<td>(N ⊕ V) = 0</td>
<td>BRGE</td>
<td>Signed</td>
</tr>
<tr>
<td>Rd &gt; Rr</td>
<td>C + Z = 0</td>
<td>BRLO(1)</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(1)</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>

Note: 1. Interchange Rd and Rr in the operation before the test. i.e. CP Rd,Rr → CP Rr,Rd
Complete Instruction Set Summary

Notes:
1. Not all instructions are available in all devices. Refer to the device specific instruction summary.
2. Cycle times for data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface. For LD, ST, LDS, STS, PUSH, POP, add one cycle plus one cycle for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI in devices with 16 bit PC, add three cycles plus two cycles for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI in devices with 22 bit PC, add five cycles plus three cycles for each wait state.

Instruction Set Summary

<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>ADD</td>
<td>Rd, Rr</td>
<td>Add without Carry</td>
<td>Rd ← Rd + Rr</td>
<td>Z,C,N,V,S,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,S,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,S</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,S,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,S,H</td>
<td>1</td>
</tr>
<tr>
<td>SBC</td>
<td>Rd, Rr</td>
<td>Subtract with Carry</td>
<td>Rd ← Rd - Rr - C</td>
<td>Z,C,N,V,S,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,S,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,S</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,S</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,S</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,S</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,S</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,S</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,S</td>
<td>1</td>
</tr>
<tr>
<td>NEG</td>
<td>Rd</td>
<td>Two's Complement</td>
<td>Rd ← $00 - Rd</td>
<td>Z,C,N,V,S,H</td>
<td>1</td>
</tr>
<tr>
<td>SBR</td>
<td>Rd,K</td>
<td>Set Bit(s) in Register</td>
<td>Rd ← Rd v K</td>
<td>Z,N,V,S</td>
<td>1</td>
</tr>
<tr>
<td>CBR</td>
<td>Rd,K</td>
<td>Clear Bit(s) in Register</td>
<td>Rd ← Rd • ($FFh - K)</td>
<td>Z,N,V,S</td>
<td>1</td>
</tr>
<tr>
<td>INC</td>
<td>Rd</td>
<td>Increment</td>
<td>Rd ← Rd + 1</td>
<td>Z,N,V,S</td>
<td>1</td>
</tr>
<tr>
<td>DEC</td>
<td>Rd</td>
<td>Decrement</td>
<td>Rd ← Rd - 1</td>
<td>Z,N,V,S</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,S</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,S</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 (UU)</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>MULS</td>
<td>Rd,Rr</td>
<td>Multiply Signed</td>
<td>R1:R0 ← Rd × Rr (SS)</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>MULSU</td>
<td>Rd,Rr</td>
<td>Multiply Signed with Unsigned</td>
<td>R1:R0 ← Rd × Rr (SU)</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>FMUL</td>
<td>Rd,Rr</td>
<td>Fractional Multiply Unsigned</td>
<td>R1:R0 ← (Rd × Rr)&lt;&lt;1 (UU)</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>FMULS</td>
<td>Rd,Rr</td>
<td>Fractional Multiply Signed</td>
<td>R1:R0 ← (Rd × Rr)&lt;&lt;1 (SS)</td>
<td>Z,C</td>
<td>2</td>
</tr>
<tr>
<td>FMULSU</td>
<td>Rd,Rr</td>
<td>Fractional Multiply Signed with Unsigned</td>
<td>R1:R0 ← (Rd × Rr)&lt;&lt;1 (SU)</td>
<td>Z,C</td>
<td>2</td>
</tr>
</tbody>
</table>
### 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>I JMP</td>
<td></td>
<td>Indirect Jump to (Z)</td>
<td>PC(15:0) ← Z, PC(21:16) ← 0</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>E I JMP</td>
<td></td>
<td>Extended Indirect Jump to (Z)</td>
<td>PC(15:0) ← Z, PC(21:16) ← EIND</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 / 4</td>
</tr>
<tr>
<td>ICALL</td>
<td></td>
<td>Indirect Call to (Z)</td>
<td>PC(15:0) ← Z, PC(21:16) ← 0</td>
<td>None</td>
<td>3 / 4</td>
</tr>
<tr>
<td>EICALL</td>
<td></td>
<td>Extended Indirect Call to (Z)</td>
<td>PC(15:0) ← Z, PC(21:16) ← EIND</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>CALL</td>
<td>k</td>
<td>Call Subroutine</td>
<td>PC ← k</td>
<td>None</td>
<td>4 / 5</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td>Subroutine Return</td>
<td>PC ← STACK</td>
<td>None</td>
<td>4 / 5</td>
</tr>
<tr>
<td>RETI</td>
<td></td>
<td>Interrupt Return</td>
<td>PC ← STACK</td>
<td>I</td>
<td>4 / 5</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 / 3</td>
</tr>
<tr>
<td>CP</td>
<td>Rd,Rr</td>
<td>Compare</td>
<td>Rd - Rr</td>
<td>Z,C,N,V,S,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,S,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,S,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 / 3</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 / 3</td>
</tr>
<tr>
<td>SBIC</td>
<td>A, b</td>
<td>Skip if Bit in I/O Register Cleared</td>
<td>if(I/O(A,b)=0) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1 / 2 / 3</td>
</tr>
<tr>
<td>SBIS</td>
<td>A, b</td>
<td>Skip if Bit in I/O Register Set</td>
<td>if(I/O(A,b)=1) PC ← PC + 2 or 3</td>
<td>None</td>
<td>1 / 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>BTC</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>
</tbody>
</table>
### 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>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 1 / 2</td>
<td></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 1 / 2</td>
<td></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 1 / 2</td>
<td></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 1 / 2</td>
<td></td>
</tr>
</tbody>
</table>

#### Data Transfer Instructions

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>Rd, Rr</td>
<td>Copy Register</td>
<td>Rd ← Rr</td>
<td>None 1</td>
</tr>
<tr>
<td>MOVW</td>
<td>Rd, Rr</td>
<td>Copy Register Pair</td>
<td>Rd+1:Rd ← Rr+1:Rr</td>
<td>None 1</td>
</tr>
<tr>
<td>LDI</td>
<td>Rd, K</td>
<td>Load Immediate</td>
<td>Rd ← K</td>
<td>None 1</td>
</tr>
<tr>
<td>LDS</td>
<td>Rd, k</td>
<td>Load Direct from data space</td>
<td>Rd ← (k)</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, X</td>
<td>Load Indirect</td>
<td>Rd ← (X)</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, X+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd ← (X), X ← X + 1</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, -X</td>
<td>Load Indirect and Pre-Decrement</td>
<td>X ← X - 1, Rd ← (X)</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Y</td>
<td>Load Indirect</td>
<td>Rd ← (Y)</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Y+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd ← (Y), Y ← Y + 1</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, -Y</td>
<td>Load Indirect and Pre-Decrement</td>
<td>Y ← Y - 1, Rd ← (Y)</td>
<td>None 2</td>
</tr>
<tr>
<td>LDD</td>
<td>Rd,Y+q</td>
<td>Load Indirect with Displacement</td>
<td>Rd ← (Y + q)</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Z</td>
<td>Load Indirect</td>
<td>Rd ← (Z)</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Z+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd ← (Z), Z ← Z+1</td>
<td>None 2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, -Z</td>
<td>Load Indirect and Pre-Decrement</td>
<td>Z ← Z - 1, Rd ← (Z)</td>
<td>None 2</td>
</tr>
<tr>
<td>LDD</td>
<td>Rd, Z+q</td>
<td>Load Indirect with Displacement</td>
<td>Rd ← (Z + q)</td>
<td>None 2</td>
</tr>
<tr>
<td>STS</td>
<td>k, Rr</td>
<td>Store Direct to data space</td>
<td>Rd ← (k)</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>X, Rr</td>
<td>Store Indirect</td>
<td>(X) ← Rr</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>X+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(X) ← Rr, X ← X + 1</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>-X, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>X ← X - 1, (X) ← Rr</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>Y, Rr</td>
<td>Store Indirect</td>
<td>(Y) ← Rr</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>Y+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(Y) ← Rr, Y ← Y + 1</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>-Y, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>Y ← Y - 1, (Y) ← Rr</td>
<td>None 2</td>
</tr>
<tr>
<td>STD</td>
<td>Y+q,Rr</td>
<td>Store Indirect with Displacement</td>
<td>(Y + q) ← Rr</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>Z, Rr</td>
<td>Store Indirect</td>
<td>(Z) ← Rr</td>
<td>None 2</td>
</tr>
<tr>
<td>ST</td>
<td>Z+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(Z) ← Rr, Z ← Z + 1</td>
<td>None 2</td>
</tr>
</tbody>
</table>
## 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>ST</td>
<td>-Z, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>Z ← Z - 1, (Z) ← 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) ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LPM</td>
<td>Rd, Z</td>
<td>Load Program Memory</td>
<td>R0 ← (Z)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>LPM</td>
<td>Rd, Z+</td>
<td>Load Program Memory and Post-Increment</td>
<td>Rd ← (Z), Z ← Z + 1</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>ELPM</td>
<td>Rd, Z</td>
<td>Extended Load Program Memory</td>
<td>Rd ← (RAMPZ:Z)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>ELPM</td>
<td>Rd, Z+</td>
<td>Extended Load Program Memory and Post-Increment</td>
<td>Rd ← (RAMPZ:Z), Z ← Z + 1</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>SPM</td>
<td></td>
<td>Store Program Memory</td>
<td>(Z) ← R1:R0</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>ESPM</td>
<td></td>
<td>Extended Store Program Memory</td>
<td>(RAMPZ:Z) ← R1:R0</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>IN</td>
<td>Rd, A</td>
<td>In From I/O Location</td>
<td>Rd ← I/O(A)</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>OUT</td>
<td>A, Rr</td>
<td>Out To I/O Location</td>
<td>I/O(A) ← Rr</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>PUSH</td>
<td>Rr</td>
<td>Push Register on Stack</td>
<td>STACK ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>POP</td>
<td>Rd</td>
<td>Pop Register from Stack</td>
<td>Rd ← STACK</td>
<td>None</td>
<td>2</td>
</tr>
</tbody>
</table>

### Bit and Bit-test Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>SBI</td>
<td>A, b</td>
<td>Set Bit in I/O Register</td>
<td>I/O(A, b) ← 1</td>
<td>None</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CBI</td>
<td>A, b</td>
<td>Clear Bit in I/O Register</td>
<td>I/O(A, b) ← 0</td>
<td>None</td>
<td>2</td>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td></td>
<td>Set Carry</td>
<td>C ← 1</td>
<td>C</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CLC</td>
<td></td>
<td>Clear Carry</td>
<td>C ← 0</td>
<td>C</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SEN</td>
<td></td>
<td>Set Negative Flag</td>
<td>N ← 1</td>
<td>N</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CLN</td>
<td></td>
<td>Clear Negative Flag</td>
<td>N ← 0</td>
<td>N</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SEZ</td>
<td></td>
<td>Set Zero Flag</td>
<td>Z ← 1</td>
<td>Z</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CLZ</td>
<td></td>
<td>Clear Zero Flag</td>
<td>Z ← 0</td>
<td>Z</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
## 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>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) \( \text{Rd} \leftarrow \text{Rd} + \text{Rr} + \text{C} \)

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

16-bit Opcode:

```
0001 11rd  dddd  rrrr
```

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: \( \text{Rd3} \cdot \text{Rr3} + \text{Rr3} \cdot \text{R3} \cdot \text{Rd3} \)
Set if there was a carry from bit 3; cleared otherwise.

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

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

C: \( \text{Rd7} \cdot \text{Rr7} + \text{Rd7} \cdot \text{Rr7} \cdot \text{R7} \cdot \text{Rd7} \)
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) \( R_d \leftarrow R_d + R_r \)

Syntax: 
(i) \( \text{ADD } R_d,R_r \)

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

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

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>0000</th>
<th>11rd</th>
<th>dddd</th>
<th>rrrr</th>
</tr>
</thead>
</table>

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>⇐</td>
<td>⇐</td>
<td>⇐</td>
</tr>
</tbody>
</table>

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

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

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

C: \( R_d \cdot R_r + R_r + R_7 \cdot R_7 \cdot R_d \)
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) \( \text{Rd}+1: \text{Rd} \leftarrow \text{Rd}+1: \text{Rd} + K \)

Syntax: \( \text{ADIW Rd, K} \)
Operands: \( d \in \{24,26,28,30\}, \ 0 \leq K \leq 63 \)
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>0110</th>
<th>KKdd</th>
<th>KKKK</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>( \Leftrightarrow )</td>
<td>( \Leftrightarrow )</td>
<td>( \Leftrightarrow )</td>
<td>( \Leftrightarrow )</td>
</tr>
</tbody>
</table>

S: \( N \oplus V \), For signed tests.
V: \( \text{Rdh}7 \bullet \text{R15} \)
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: \( \text{R15} \bullet \text{R14} \bullet \text{R12} \bullet \text{R11} \bullet \text{R10} \bullet \text{R9} \bullet \text{R8} \bullet \text{R7} \bullet \text{R6} \bullet \text{R5} \bullet \text{R4} \bullet \text{R3} \bullet \text{R2} \bullet \text{R1} \bullet \text{R0} \)
Set if the result is $0000$; cleared otherwise.
C: \( \text{R15} \bullet \text{Rdh}7 \)
Set if there was carry from the MSB of the result; cleared otherwise.

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

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

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) \( \text{Rd} \leftarrow \text{Rd} \: \& \: \text{Rr} \)

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

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>( \leftrightarrow )</td>
<td>0</td>
<td>( \leftrightarrow )</td>
<td>( \leftrightarrow )</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: \( R7 \: \& R6 \: \& R5 \: \& R4 \: \& R3 \: \& R2 \: \& R1 \: \& R0 \)
Set if the result is $00$; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```assembly
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) \[ \text{Rd} \leftarrow \text{Rd} \cdot K \]

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

16-bit Opcode:

\[
\begin{array}{|c|c|c|c|}
\hline
\text{0111} & \text{KKK} & \text{dd} & \text{KK} \\
\hline
\end{array}
\]

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

R (Result) equals Rd after the operation.

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

Words: 1 (2 bytes)
Cycles: 1
ASR - Arithmetic Shift Right

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

Operation:

(i)

Syntax: Operands: Program Counter:
(i) ASR Rd 0 ≤ d ≤ 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>

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

V: \( N \oplus C \) (For N and C after the shift)

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

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

C: \( Rd_0 \)
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

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

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

Description:
Clears a single flag in SREG.

Operation:
(i) \( SREG(s) \leftarrow 0 \)

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

16-bit Opcode:

\[
\begin{array}{cccc}
1001 & 0100 & 1sss & 1000 \\
\end{array}
\]

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: 0 if \( s = 7 \); Unchanged otherwise.
T: 0 if \( s = 6 \); Unchanged otherwise.
H: 0 if \( s = 5 \); Unchanged otherwise.
S: 0 if \( s = 4 \); Unchanged otherwise.
V: 0 if \( s = 3 \); Unchanged otherwise.
N: 0 if \( s = 2 \); Unchanged otherwise.
Z: 0 if \( s = 1 \); Unchanged otherwise.
C: 0 if \( s = 0 \); Unchanged otherwise.

Example:

bclr 0 ; Clear carry flag
bclr 7 ; Disable interrupts

Words: 1 (2 bytes)
Cycles: 1
BLD - Bit Load from the T Flag in SREG to a Bit in Register.

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

Operation:
(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 | 0bbb |

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

Words: 1 (2 bytes)
Cycles: 1
BRBC - Branch if Bit in SREG is Cleared

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

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

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

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1111</th>
<th>01kk</th>
<th>kkkk</th>
<th>ksss</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:
cpi r20,5 ; Compare r20 to the value 5
brbc 1,noteq ; Branch if zero flag cleared
...
noteq:nop ; Branch destination (do nothing)

Words: 1 (2 bytes)
Cycles: 1 if condition is false
        2 if condition is true
BRBS - Branch if Bit in SREG is Set

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

Operation:
(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:

<table>
<thead>
<tr>
<th></th>
<th>1111</th>
<th>00kk</th>
<th>kkkk</th>
<th>kss</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:

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

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

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

PC ← PC + 1, if condition is false

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>l111</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:

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
        2 if condition is true
BRCS - Branch if Carry Set

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

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

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

16-bit Opcode:

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

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true
BREQ - Branch if Equal

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

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

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

16-bit Opcode:

| 1111 | 00kk | kkkk | k001 |

Status Register (SREG) and Boolean Formulae:

Example:

```c
cp r1,r0 ; Compare registers r1 and r0
breq equal ; Branch if registers equal
...
equal: nop ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
        2 if condition is true
BRGE - Branch if Greater or Equal (Signed)

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

Operation:
(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:

| 1111 | 01kk | kkkk | k100 |

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

Operation:
(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
PC ← PC + 1, if condition is false

16-bit Opcode:

```
<table>
<thead>
<tr>
<th></th>
<th>01kk</th>
<th>kkkk</th>
<th>k101</th>
</tr>
</thead>
</table>
```

Status Register (SREG) and Boolean Formula:

```
<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:
```
brhc hclear ; Branch if half carry flag cleared
...
hclear:   nop     ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
        2 if condition is true
BRHS - Branch if Half Carry Flag is Set

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

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

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

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

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

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

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

16-bit Opcode:

```
1111 01kk kkkk k111
```

Status Register (SREG) and Boolean Formula:

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

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

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

16-bit Opcode:

| l1l1 | 00kk | kkkk | k1l1 |

Status Register (SREG) and Boolean Formula:

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

Operation:
(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

PC ← PC + 1, if condition is false

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>

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

Operation:
(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:

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true
BRMI - Branch if Minus

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

Operation:
(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 Formula:

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
        2 if condition is true
BRNE - Branch if Not Equal

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

Operation:
(i) If Rd \neq Rr (Z = 0) then PC \leftarrow PC + k + 1, else PC \leftarrow PC + 1

Syntax: Operands: Program Counter:
(i) BRNE k -64 \leq k \leq +63 PC \leftarrow PC + k + 1

PC \leftarrow 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 Formula:

Example:

```
  eor  r27,r27 ; Clear r27
  loop:  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 - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 2,k).

Operation:
(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:

| I111 | 01kk | kkkk | k010 |

Status Register (SREG) and Boolean Formula:

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

Operation:
(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:

\[
\begin{array}{|c|c|c|c|}
\hline
I & T & H & S \\
\hline
- & - & - & - \\
\hline
\end{array}
\]

Status Register (SREG) and Boolean Formula:

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true
BRTC - Branch if the T Flag is Cleared

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

Operation:
(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:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>01kk</td>
<td>kkkk</td>
<td>k110</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:
```
bst r3,5 ; Store bit 5 of r3 in T flag
brtc tclear ; Branch if this bit was cleared
...
tclear: nop ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
         2 if condition is true
BRTS - Branch if the T Flag is Set

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

Operation:
(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:

| l111 | 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 - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 3,k).

Operation:
(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 Formula:
```
I  T  H  S  V  N  Z  C
-  -  -  -  -  -  -  -
```

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

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true
BRVS - Branch if Overflow Set

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

Operation:
(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:

| llll | 00kk | kkkk | k011 |

Status Register (SREG) and Boolean Formula:

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

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>0sss</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>⇔</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:
- \( \text{bset} \ 6 \) ; Set T flag
- \( \text{bset} \ 7 \) ; 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) \( T \leftarrow R_d(b) \)

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

16-bit Opcode:

\[
\begin{array}{cccc}
1111 & 101d & dddd & 0bbb
\end{array}
\]

Status Register (SREG) and Boolean Formula:

<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:

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

Words: 1 (2 bytes)
Cycles: 1
CALL - Long Call to a Subroutine

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

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

Syntax: Operands: Program Counter Stack:
(i) CALL k  0 ≤ k < 64K  PC ← k  STACK ← PC+2
         kS T A C K  ←  kS T A C K  − 2, (2 bytes, 16 bits)
(ii) CALL k  0 ≤ k < 4M  PC ← k  STACK ← PC+2
         kS T A C K  ←  kS T A C K  − 3 (3 bytes, 22 bits)

32-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 Formula:

Example:
```
mov r16,r0 ; Copy r0 to r16
call check ; Call subroutine
nop ; Continue (do nothing)
...
check: cpi r16,$42 ; Check if r16 has a special value
breq error ; Branch if equal
ret ; Return from subroutine
...
error: rjmp error ; Infinite loop
```

Words: 2 (4 bytes)
Cycles: 4, devices with 16 bit PC
5, devices with 22 bit PC
CBI - Clear Bit in I/O Register

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

Operation:
(i) I/O(A,b) ← 0

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

16-bit Opcode:

| 1001 | 1000 | AAAA | Abbb |

Status Register (SREG) and Boolean Formula:

<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:

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

Words: 1 (2 bytes)
Cycles: 2
CBR - Clear Bits in Register

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

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

Syntax:
(i) CBR Rd,K

Operands:
16 \( \leq d \leq 31, 0 \leq K \leq 255 \)

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

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

Status Register (SREG) and Boolean Formula:

<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>( \leftrightarrow )</td>
<td>0</td>
<td>( \leftrightarrow )</td>
<td>( \leftrightarrow )</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: \( R7 \land R6 \land R5 \land R4 \land R3 \land R2 \land R1 \land R0 \)
Set if the result is $00$; cleared otherwise.
R (Result) equals Rd after the operation.

Example:
\[
\begin{align*}
\text{cb} & \quad \text{r16,$SF0$} \quad ; \text{Clear upper nibble of r16} \\
\text{cb} & \quad \text{r18,1} \quad ; \text{Clear bit 0 in r18}
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 1
CLC - Clear Carry Flag

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

Operation:
(i) C ← 0

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

16-bit Opcode:

```
1001 0100 1000 1000
```

Status Register (SREG) and Boolean Formula:

<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: Operands: Program Counter:
(i) CLH None PC \( \leftarrow \) PC + 1

16-bit Opcode:

```
1001 0100 1101 1000
```

Status Register (SREG) and Boolean Formula:

<table>
<thead>
<tr>
<th></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:
```assembly
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: CLI  
Operands: None  
Program Counter: \( PC \leftarrow PC + 1 \)

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>0100</th>
<th>1111</th>
<th>1000</th>
</tr>
</thead>
</table>

Status Register (SREG) and Boolean Formula:

<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:
```
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).

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

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

16-bit Opcode:
```
1001  0100  1010  1000
```

Status Register (SREG) and Boolean Formula:

<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>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

N: 0
Negative flag cleared

Example:
```
add r2, r3 ; Add r3 to r2
cln ; Clear negative flag
```

Words: 1 (2 bytes)
Cycles: 1
CLR - Clear Register

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

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

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

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

<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>

Status Register (SREG) and Boolean Formula:

S: 0
Cleared

V: 0
Cleared

N: 0
Cleared

Z: 1
Set

R (Result) equals Rd after the operation.

Example:

```assembly
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
ITHSVNZC --0----
```

Status Register (SREG) and Boolean Formula:

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

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>1110</td>
<td>1000</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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) \( V \leftarrow 0 \)

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

16-bit Opcode:

| 1001 | 0100 | 1011 | 1000 |

Status Register (SREG) and Boolean Formula:

<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: Operands: Program Counter:
(i) CLZ None PC \( \leftarrow \) PC + 1

16-bit Opcode:

```
1001 0100 1001 1000
```

Status Register (SREG) and Boolean Formula:

<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) \( Rd \leftarrow \$FF - Rd \)

Syntax: Operands: Program Counter:
(i) COM Rd \( 0 \leq d \leq 31 \) PC \( \leftarrow \) 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>( \leftrightarrow )</td>
<td>0</td>
<td>( \leftrightarrow )</td>
<td>( \leftrightarrow )</td>
<td>1</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.

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 \( 0 \leq d \leq 31, 0 \leq r \leq 31 \) PC \( \leftarrow \) PC + 1

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>0001</th>
<th>01rd</th>
<th>dddd</th>
<th>rrrr</th>
</tr>
</thead>
</table>

Status Register (SREG) and Boolean Formula:

<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: \( \overline{Rd} \bullet Rr \bullet R3 \bullet R3 \bullet R3 \bullet \overline{Rd} \)
Set if there was a borrow from bit 3; cleared otherwise

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

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

C: \( Rd7 \bullet R7 \bullet R7 \bullet R7 \bullet R7 \bullet R7 \bullet \overline{Rd7} \)
Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

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

Words: 1 (2 bytes)
Cycles: 1
CPC - Compare with Carry

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

Operation:
(i) \( Rd - Rr - C \)

Syntax: CPC Rd, Rr
Operands: \( 0 \leq d \leq 31, 0 \leq r \leq 31 \)

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

16-bit Opcode:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operand</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>01rd</td>
<td>dddd rrrr</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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 borrow from bit 3; cleared otherwise.

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

V: \( Rd_7 \cdot Rr_7 + Rr_7 \cdot R_7 + R_7 \cdot Rd_7 \)
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: \( R_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0 \cdot Z \)
Previous value remains unchanged when the result is zero; cleared otherwise.

C: \( Rd_7 \cdot Rr_7 + Rr_7 \cdot R_7 + R_7 \cdot Rd_7 \)
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:
```
; 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) Rd - K

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

16-bit Opcode:

| 0011 | KKKK | dddd | KKKK |

Status Register (SREG) and Boolean Formula:

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

H: \[ \text{Rd}_3 \text{•K}_3+ \text{K}_3\text{•} \text{R}_3\text{+} \text{R}_3\text{•} \text{Rd}_3 \]
Set if there was a borrow from bit 3; cleared otherwise

S: \[ N \oplus V \], For signed tests.

V: \[ \text{Rd}_7 \text{•K}_7\text{•} \text{R}_7\text{+} \text{Rd}_7\text{•} \text{K}_7\text{•} \text{R}_7 \]
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: \[ \text{R}_7\text{•} \text{R}_6\text{•} \text{R}_5\text{•} \text{R}_4\text{•} \text{R}_3\text{•} \text{R}_2\text{•} \text{R}_1\text{•} \text{R}_0 \]
Set if the result is $00$; cleared otherwise.

C: \[ \text{Rd}_7\text{•} \text{K}_7+ \text{K}_7\text{•} \text{R}_7\text{+} \text{Rd}_7\text{•} \text{Rd}_7 \]
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:
```assembly
  cpi  r19,3 ; Compare r19 with 3
  brne error ; Branch if r19<>3
  ...
  error:   nop ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1
CPSE - Compare Skip if Equal

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

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

Syntax: Operands: Program Counter:
(i) CPSE Rd,Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31

16-bit Opcode:

| 0001 | 00rd | dddd | rrrr |

Status Register (SREG) and Boolean Formula:

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

Words: 1 (2 bytes)

Cycles:
1 if condition is false (no skip)
2 if condition is true (skip is executed) and the instruction skipped is 1 word
3 if condition is true (skip is executed) and the instruction skipped is 2 words
DEC - Decrement

Description:
Subtracts one -1- from the contents of register Rd and places the result in the destination register Rd. The C flag in SREG is not affected by the operation, thus allowing the DEC instruction to be used on a loop counter in multiple-precision computations.
When operating on unsigned values, only BREQ and BRNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

Operation:
(i) \[ \text{Rd} \leftarrow \text{Rd} - 1 \]

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

16-bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Status Register and Boolean Formula:

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

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

V: \( R_7 \bullet R_6 \bullet R_5 \bullet R_4 \bullet R_3 \bullet R_2 \bullet R_1 \bullet R_0 \)
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: \( R_7 \)
Set if MSB of the result is set; cleared otherwise.

Z: \( R_7 \bullet R_6 \bullet R_5 \bullet R_4 \bullet R_3 \bullet R_2 \bullet R_1 \bullet R_0 \)
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
EICALL - Extended Indirect Call to Subroutine

Description:
Indirect call of a subroutine pointed to by the Z (16 bits) pointer register in the register file and the EIND register in the I/O space. This instruction allows for indirect calls to the entire program memory space. This instruction is not implemented for devices with 2 bytes PC, see ICALL. The stack pointer uses a post-decrement scheme during EICALL.

Operation:
(i) PC(15:0) ← Z(15:0)
     PC(21:16) ← EIND

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

16-bit Opcode:

| 1001 | 0101 | 0001 | 1001 |

Status Register (SREG) and Boolean Formula:

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

Words: 1 (2 bytes)
Cycles: 4 (only implemented in devices with 22 bit PC)
### EIJMP - Extended Indirect Jump

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

**Operation:**
(i) \( \text{PC}(15:0) \leftarrow \text{Z}(15:0) \)
(ii) \( \text{PC}(21:16) \leftarrow \text{EIND} \)

**Syntax:**
EIJMP

**Operands:**
None

**Program Counter:**
See Operation

**Stack:**
Not Affected

**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>0001</td>
<td>1001</td>
</tr>
</tbody>
</table>

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

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

**Words:** 1 (2 bytes)

**Cycles:** 2
ELPM - Extended Load Program Memory

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

The result of these combinations is undefined:

- **ELPM r30, Z+**
- **ELPM r31, Z+**

**Operation:**

(i) \( R0 \leftarrow (\text{RAMPZ:Z}) \)

(ii) \( Rd \leftarrow (\text{RAMPZ:Z}) \)

(iii) \( Rd \leftarrow (\text{RAMPZ:Z}) \) \( (\text{RAMPZ:Z}) \leftarrow (\text{RAMPZ:Z}) + 1 \)

**Comment:**

- **RAMPZ:Z** Unchanged, \( R0 \) implied destination register
- **RAMPZ:Z** Unchanged
- **RAMPZ:Z** Post incremented

**Syntax:**

(i) **ELPM** None, \( R0 \) implied

(ii) **ELPM** Rd, Z \( 0 \leq d \leq 31 \)

(iii) **ELPM** Rd, Z+ \( 0 \leq d \leq 31 \)

**Program Counter:**

- \( PC \leftarrow PC + 1 \)

**16 bit Opcode:**

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>001</th>
<th>1101</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>0110</td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>0111</td>
</tr>
</tbody>
</table>

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

- \( - \)
- \( - \)
- \( - \)
- \( - \)
- \( - \)
- \( - \)
- \( - \)
- \( - \)

**Example:**

- **clr r16** ; Clear RAMPZ
- **out RAMPZ, r16**
- **clr r31** ; Clear Z high byte
- **ldi r30, SF0** ; Set Z low byte
- **elpm r16, Z+** ; Load constant from program
  - memory pointed to by RAMPZ:Z (r31:r30)

**Words:** 1 (2 bytes)

**Cycles:** 3
EOR - Exclusive OR

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

Operation:
(i) \( \text{Rd} \leftarrow \text{Rd} \oplus \text{Rr} \)

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

16-bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>01rd</td>
<td>dddd</td>
<td>rrrr</td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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>(\Leftrightarrow)</td>
<td>0</td>
<td>(\Leftrightarrow)</td>
<td>(\Leftrightarrow)</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: \(R7 \land R6 \land R5 \land R4 \land R3 \land R2 \land R1 \land R0\)
Set if the result is $00$; cleared otherwise.

R (Result) equals Rd after the operation.

Example:
\[
\begin{align*}
&\text{eor } r4,r4 ; \text{Clear } r4 \\
&\text{eor } r0,r22 ; \text{Bitwise exclusive or between } r0 \text{ and } r22
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 1
ESPM - Extended Store Program Memory

Description:
ESPM can be used to erase a page in the program memory, to write a page in the program memory (that is already erased), and to set boot loader lock bits. In some devices, the program memory can be written one word at a time, in other devices an entire page can be programmed simultaneously after first filling a temporary page buffer. In all cases, the program memory must be erased one page at a time. When erasing the program memory, the RAMPZ and Z registers are used as page address. When writing the program memory, the RAMPZ and Z registers are used as page or word address, and the R1:R0 register pair is used as data. When setting the boot loader lock bits, the R1:R0 register pair is used as data. Refer to the device documentation for detailed description of ESPM usage. This instruction can address the entire program memory.

Operation:
(i) (RAMPZ:Z) ← $ffff  
(ii) (RAMPZ:Z) ← R1:R0  
(iii) (RAMPZ:Z) ← R1:R0  
(iv) (RAMPZ:Z) ← TEMP  
(v) BLBITS ← R1:R0  

Comment:
Erase program memory page  
Write program memory word  
Write temporary page buffer  
Write temporary page buffer to program memory  
Set boot loader lock bits

Syntax:
(i)-(v) ESPM

Operands:
None

Program Counter:
PC ← PC + 1

16-bit Opcode:

| 1001 | 0101 | 1111 | 1000 |

Status Register (SREG) and Boolean Formula:

<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:

; This example shows ESPM write of one word for devices with page write
clr r31 ; Clear Z high byte
clr r30 ; Clear Z low byte
ldi r16,$F0 ; Load RAMPZ register
out RAMPZ, r16 ;
ldi r16, $CF ; Load data to store
mov r1, r16
ldi r16, $FF
mov r0, r16
ldi r16,$03 ; Enable ESPM, erase page
out SPMCR, r16 ;
espm ; Erase page starting at $F00000
ldi r16,$01 ; Enable ESPM, store R1:R0 to temporary buffer
out SPMCR, r16 ;
espm ; Execute ESPM, store R1:R0 to temporary buffer location $F00000
ldi r16,$05 ; Enable ESPM, write page
out SPMCR, r16 ;
espm ; Execute SPM, store temporary buffer to program memory page starting at $F00000

Words: 1 (2 bytes)
Cycles: depends on the operation
**FMUL - Fractional Multiply Unsigned**

**Description:**
This instruction performs $8 \times 8 \rightarrow 16$ unsigned multiplication and shifts the result one bit left.

<table>
<thead>
<tr>
<th>Rd</th>
<th>$\times$</th>
<th>Rr</th>
<th>$\rightarrow$</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mulicand</td>
<td>8</td>
<td>Multiplier</td>
<td>8</td>
<td>Product High</td>
<td>Product Low</td>
</tr>
</tbody>
</table>

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

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

**Operation:**
(i) $R1:R0 \leftarrow Rd \times Rr$ (unsigned $(1.15) \leftarrow\text{unsigned } (1.7) \times\text{unsigned } (1.7))$

**Syntax:**
(i) FMUL Rd,Rr  
**Operands:**  
16 $\leq d \leq 23$, 16 $\leq r \leq 23$  
**Program Counter:**  
PC $\leftarrow$ PC + 1

**16-bit Opcode:**

```
0000 0011 00dd 1rrr
```

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

<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: R16  
Set if bit 15 of the result before left shift is set; cleared otherwise.

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

**Example:**
```
fmul r23,r22 ; Multiply unsigned r23 and r22 in (1.7) format, result in (1.15) format
movw r22,r0  ; Copy result back in r23:r22
```

**Words:** 1 (2 bytes) 
**Cycles:** 2
FMULS - Fractional Multiply Signed

Description:
This instruction performs 8-bit \( \times \) 8-bit \( \rightarrow \) 16-bit signed multiplication and shifts the result one bit left.

\[
\begin{array}{ccc}
\text{Rd} & \times & \text{Rr} \\
\text{Multiplicand} & \rightarrow & \text{Multiplier} \\
\hline
8 & \times & 8 \rightarrow 16 \\
\end{array}
\]

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

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

Operation:
(i) \( R1:R0 \leftarrow \text{Rd} \times \text{Rr} \) (signed (1.15) \( \leftarrow \) signed (1.7) \( \times \) signed (1.7))

Syntax: Operands: Program Counter:
(i) FMUL Rd,Rr \( \quad 16 \leq d \leq 23, 16 \leq r \leq 23 \) PC \( \leftarrow \) PC + 1

16-bit Opcode:

| 0000 | 0011 | lddd | 0rrr |

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: R16
Set if bit 15 of the result before left shift is set; cleared otherwise.

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

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

Example:
\[
\begin{align*}
\text{fmuls} \ r23, r22 & \quad ; \text{Multiply signed} \ r23 \text{ and} \ r22 \text{ in (1.7) format, result in (1.15) format} \\
\text{movw} \ r22, r0 & \quad ; \text{Copy result back in r23:r22}
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 2
FMULSU - Fractional Multiply Signed with Unsigned

Description:
This instruction performs 8-bit \( \times \) 8-bit \( \rightarrow \) 16-bit signed multiplication and shifts the result one bit left.

\[
\begin{array}{cccc}
Rd & \times & Rr & \rightarrow \\
\text{Multiplicand} & 8 & \text{Multiplier} & 8 & \text{Product High} & 16 & \text{Product Low}
\end{array}
\]

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

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

Operation:
\((i)\) \(R1:R0 \leftarrow Rd \times Rr\) (signed \((1.15) \leftarrow \text{signed} \,(1.7) \times \text{unsigned} \,(1.7))\)

Syntax: Operands: Program Counter:
\((i)\) FMULSU Rd,Rr \(16 \leq d \leq 23, 16 \leq r \leq 23\) \(PC \leftarrow PC + 1\)

16-bit Opcode:
\[
\begin{array}{cccc}
0000 & 0011 & \text{lddd} & \text{lrrr}
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

\(C:\) \(R16\)
Set if bit 15 of the result before left shift is set; cleared otherwise.

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

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

Example:
\[
\begin{align*}
\text{fmulSU \; r23,r22} & : \text{Multiply signed r23 with unsigned r22 in (1.7) format, signed result in (1.15) format} \\
\text{movw \; r22,r0} & : \text{Copy result back in r23:r22}
\end{align*}
\]

Words: 1 (2 bytes)
Cycles: 2
ICALL - Indirect Call to Subroutine

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

Operation:
(i) $\text{PC}(15:0) \leftarrow Z(15:0)$ 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) $\leftarrow 0$

Syntax: ICALL
Operands: None
Program Counter: See Operation
Stack:
- STACK $\leftarrow \text{PC} + 1$
- SP $\leftarrow \text{SP} - 2$ (2 bytes, 16 bits)

Syntax: ICALL
Operands: None
Program Counter: See Operation
Stack:
- STACK $\leftarrow \text{PC} + 1$
- SP $\leftarrow \text{SP} - 3$ (3 bytes, 22 bits)

16-bit Opcode:

```
1001 0101 0000 1001
```

Status Register (SREG) and Boolean Formula:

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

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

Words: 1 (2 bytes)
Cycles: 3 devices with 16 bit PC
- 4 devices with 22 bit PC
IJMP - Indirect Jump

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

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.
\( \text{PC}(21:16) \leftarrow 0 \)

Syntax: Operands: Program Counter: Stack:
(i),(ii) IJMP None See Operation Not Affected

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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

Example:

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

Words: 1 (2 bytes)
Cycles: 2
IN - Load an I/O Location to Register

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

Operation:
(i) \( \text{Rd} \leftarrow \text{I/O(A)} \)

Syntax: \( \text{IN Rd,A} \)
Operands: \( 0 \leq d \leq 31, 0 \leq A \leq 63 \)
Program Counter: \( \text{PC} \leftarrow \text{PC} + 1 \)

16-bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1011</td>
<td>0AAa</td>
<td>dddd</td>
<td>AAAA</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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{verbatim}
in r25,$16 ; Read Port B
cpi r25,4 ; Compare read value to constant
breq exit ; Branch if r25=4
...
exit: nop ; Branch destination (do nothing)
\end{verbatim}

Words: 1 (2 bytes)
Cycles: 1
INC - Increment

**Description:**
Adds one \(-1\) to the contents of register \(Rd\) and places the result in the destination register \(Rd\).

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

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

**Operation:**
(i) \(Rd \leftarrow Rd + 1\)

**Syntax:**
(i) INC \(Rd\)

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

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

**16-bit Opcode:**
\[
\begin{array}{c|ccc|c}
1001 & 010d & dddd & 0011 \\
\end{array}
\]

**Status Register and Boolean Formula:**

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

<table>
<thead>
<tr>
<th>1001</th>
<th>010k</th>
<th>kkkk</th>
<th>110k</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 Formula:

<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 data space to Register using Index X**

**Description:**

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

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

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

The result of these combinations is undefined:

- LD r26, X+
- LD r27, X+
- LD r26, -X
- LD r27, -X

**Using the X pointer:**

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

**Syntax:**

- (i) LD Rd, X 0 ≤ d ≤ 31
- (ii) LD Rd, X+ 0 ≤ d ≤ 31
- (iii) LD Rd, -X 0 ≤ d ≤ 31

**Program Counter:**

- PC ← PC + 1
- PC ← PC + 1
- PC ← PC + 1

**16-bit Opcode:**

| (i) | 1001 | 000d | dddd | 1100 |
| (ii) | 1001 | 000d | dddd | 1101 |
| (iii) | 1001 | 000d | dddd | 1110 |

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

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

Words: 1 (2 bytes)
Cycles: 2
LD (LDD) - Load Indirect from data space to Register using Index Y

Description:
Loads one byte indirect with or without displacement from the data space to a register. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.
The data location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPY in register in the I/O area has to be changed.
The Y pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for accessing arrays, tables, and stack pointer usage of the Y pointer register. Note that only the low byte of the Y pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPY register in the I/O area is updated in parts with more than 64K bytes data space, and the displacement is added to the entire 24-bit address on such devices.
The result of these combinations is undefined:

\[
\begin{align*}
LD r28, Y+ \\
LD r29, Y+ \\
LD r28, -Y \\
LD r29, -Y \\
\end{align*}
\]

Using the Y pointer:

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

Syntax: \( 0 \leq d \leq 31 \), \( 0 \leq q \leq 63 \)

Program Counter:

| (i) | LD Rd, Y 0 \leq d \leq 31 | PC \leftarrow PC + 1 |
| (ii) | LD Rd, Y+ 0 \leq d \leq 31 | PC \leftarrow PC + 1 |
| (iii) | LD Rd, -Y 0 \leq d \leq 31 | PC \leftarrow PC + 1 |
| (iv) | LDD Rd, Y+q 0 \leq d \leq 31, 0 \leq q \leq 63 | PC \leftarrow PC + 1 |

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>I100</th>
<th>0000d</th>
<th>dddd</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I1001</td>
<td>0000d</td>
<td>dddd</td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td>I1001</td>
<td>0000d</td>
<td>dddd</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>I10q0</td>
<td>qq0d</td>
<td>dddd</td>
<td>1qqq</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<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,$60 ; Set Y low byte to $60
ld r0,Y+ ; Load r0 with data space loc. $60(Y post inc)
ld r1,Y ; Load r1 with data space loc. $61
ldi r28,$63 ; Set Y low byte to $63
ld r2,Y ; Load r2 with data space loc. $63
ld r3,-Y ; Load r3 with data space loc. $62(Y pre dec)
ldd r4,Y+2 ; Load r4 with data space loc. $64
```

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

Description:
Loads one byte indirect with or without displacement from the data space to a register. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.
The data location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.
The Z pointer register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for stack pointer usage of the Z pointer register, however because the Z pointer register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y pointer as a dedicated stack pointer. Note that only the low byte of the Z pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ register in the I/O area is updated in parts with more than 64K bytes data space, and that the displacement is added to the entire 24-bit address on such devices. For devices with more than 64K bytes data memory, the RAMPZ register is only used by the ELPM and ESPM instructions. Hence, RAMPZ is not affected by the ST instruction.
For using the Z pointer for table lookup in program memory see the LPM and ELPM instructions.
The result of these combinations is undefined:

LD r30, Z+  
LD r31, Z+  
LD r30, -Z  
LD r31, -Z

Using the Z pointer:

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

Syntax: Operands: Program Counter:

| (i) LD Rd, Z | 0 ≤ d ≤ 31 | PC ← PC + 1 |
| (ii) LD Rd, Z+ | 0 ≤ d ≤ 31 | PC ← PC + 1 |
| (iii) LD Rd, -Z | 0 ≤ d ≤ 31 | PC ← PC + 1 |
| (iv) LDD Rd, Z+q | 0 ≤ d ≤ 31, 0 ≤ q ≤ 63 | PC ← PC + 1 |
16-bit Opcode:

| (i) | 1000 | 000d | dddd | 0000 |
| (ii) | 1001 | 000d | dddd | 0001 |
| (iii) | 1001 | 000d | dddd | 0010 |
| (iij) | 10q0 | qq0d | dddd | 0qqq |

Status Register (SREG) and Boolean Formula:

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

Words: 1 (2 bytes)

Cycles: 2
LDI - Load Immediate

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

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

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

16-bit Opcode:

| 1110 | KKKK | dddd | KKKK |

Status Register (SREG) and Boolean Formula:

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

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

Description:
Loads one byte from the data space to a register. For parts with SRAM, the data space consists of the register file, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the register file only. The EEPROM has a separate address space.
A 16-bit address must be supplied. Memory access is limited to the current data segment of 64K bytes. The LDS instruction uses the RAMPD register to access memory above 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPD in register in the I/O area has to be changed.

Operation:

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

Syntax: \( \text{LDS Rd},k \)
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>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>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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 data space location $FF00
add r2,r1 ; add r1 to r2
sts $FF00,r2 ; Write back

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

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

The result of these combinations is undefined:

LPM r30, Z+
LPM r31, Z+

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

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

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>0101</th>
<th>1100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1001</td>
<td>0101</td>
<td>1100</td>
<td>1000</td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>0100</td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>0101</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

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 signed and unsigned values by two.

Operation:
(i)

\[
C \leftarrow \begin{array}{c}
\vdots \\
b7 & \cdots & b0 \\
\end{array} \leftarrow 0
\]

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

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

Status Register (SREG) and Boolean Formula:

<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>(\Leftarrow)</td>
<td>(\Leftarrow)</td>
<td>(\Leftarrow)</td>
<td>(\Leftarrow)</td>
<td>(\Leftarrow)</td>
<td>(\Leftarrow)</td>
</tr>
</tbody>
</table>

H: Rd3
S: \(N \oplus V\), For signed tests.
V: \(N \oplus C\) (For N and C after the shift)
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
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:

\[
\begin{align*}
0 & \rightarrow b7 \ldots b0 \rightarrow C \\
\end{align*}
\]

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

16-bit Opcode:

\[
\begin{array}{c|c|c|c}
1001 & 010 & dddd & 0110 \\
\end{array}
\]

Status Register (SREG) and Boolean Formula:

<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>(\Leftrightarrow)</td>
<td>(\Leftrightarrow)</td>
<td>0</td>
<td>(\Leftrightarrow)</td>
<td>(\Leftrightarrow)</td>
</tr>
</tbody>
</table>

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

V: \(N \oplus C\) (For \(N\) and \(C\) after the shift)

N: 0

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: Rd0  
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

\[
\begin{align*}
\text{add} & \quad r0,r4 \quad ; \text{Add } r4 \text{ to } r0 \\
\text{lsr} & \quad r0 \quad ; \text{Divide } r0 \text{ by } 2 \\
\end{align*}
\]

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

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

16-bit Opcode:

```
0010  \text{l1rd} \ \text{dddd} \ \text{rrrr}
```

Status Register (SREG) and Boolean Formula:

<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
...
check:  cpi  r16,$11  ; Compare r16 to $11
...
ret   ; Return from subroutine
```

Words: 1 (2 bytes)
Cycles: 1
MOVW - Copy Register Word

Description:
This instruction makes a copy of one register pair into another register pair. The source register pair \( R_r+1:R_r \) is left unchanged, while the destination register pair \( R_d+1:R_d \) is loaded with a copy of \( R_r + 1:R_r \).

Operation:
(i) \( R_d+1:R_d \leftarrow R_r+1:R_r \)

Syntax: Operands: Program Counter:
(i) MOVW Rd,Rr \( d \in \{0,2,...,30\}, r \in \{0,2,...,30\} \) PC \( \leftarrow \) PC + 1

16-bit Opcode:

| 0000 | 0001 | dddd | rrrr |

Status Register (SREG) and Boolean Formula:

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

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

Description:
This instruction performs 8-bit $\times$ 8-bit $\rightarrow$ 16-bit unsigned multiplication.

\[
\begin{array}{c|c|c|c|c}
\text{Rd} & \times & \text{Rr} & \rightarrow & \text{R1} & \text{R0} \\
\text{Multiplicand} & 8 & \text{Multiplier} & 8 & \text{Product High} & \text{Product Low} \\
\end{array}
\]

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

Operation:
(i) $R1:R0 \leftarrow Rd \times Rr$ (unsigned $\leftarrow$ unsigned $\times$ unsigned)

Syntax:
(i) \text{MUL Rd,Rr}

Operands: \[0 \leq d \leq 31, 0 \leq r \leq 31\]

Program Counter:
\[PC \leftarrow PC + 1\]

16-bit Opcode:
\[1001 \quad l1rd \quad dddd \quad rrrr\]

Status Register (SREG) and Boolean Formulae:

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

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

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

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

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

Words: 1 (2 bytes)
Cycles: 2
MULS - Multiply Signed

Description:
This instruction performs 8-bit $\times$ 8-bit $\rightarrow$ 16-bit signed multiplication.

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

Operation:
(i) $R1:R0 \leftarrow Rd \times Rr$ (signed $\leftarrow$ signed $\times$ signed)

Syntax: Operands: Program Counter:
(i) MULS Rd,Rr 16 $\leq d \leq 31$, 16 $\leq r \leq 31$ PC $\leftarrow$ PC + 1

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

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

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

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

Example:

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

Words: 1 (2 bytes)
Cycles: 2
MULSU - Multiply Signed with Unsigned

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

<table>
<thead>
<tr>
<th>Rd</th>
<th>Rr</th>
<th>R1</th>
<th>R0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd</td>
<td>Rr</td>
<td>→</td>
<td>Product High</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

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

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

Syntax: Operands: Program Counter:
(i) MULSU Rd,Rr 16 ≤ d ≤ 23, 16 ≤ r ≤ 23 PC ← PC + 1

16-bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0011</td>
<td>0ddd</td>
<td>0rrr</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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.

Z: R15 • R14 • R13 • R12 • R11 • R10 • R9 • R8 • R7 • R6 • R5 • R4 • R3 • R2 • R1 • R0
Set if the result is $0000; cleared otherwise.

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

Example:
mulsu r21,r20 ; Multiply signed r21 with unsigned r20, signed result
movw r20,r0 ; Copy result back in r21:r20

Words: 1 (2 bytes)
Cycles: 2
NEG - Two’s Complement

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

Operation:
(i) \( \text{Rd} \leftarrow \$00 - \text{Rd} \)

Syntax: \( \text{NEG Rd} \quad 0 \leq d \leq 31 \)

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

16-bit Opcode:

```
1001 010d dddd 0001
```

Status Register (SREG) and Boolean Formula:

<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: \( R3 + \text{Rd3} \)
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:
```
0000 0000 0000 0000
```

Status Register (SREG) and Boolean Formula:

<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:
```
cir 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
OR - Logical OR

Description:
Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:
(i) Rd ← Rd v Rr

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

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

<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:
```
or r15,r16 ; Do bitwise or between registers
bst r15,6  ; Store bit 6 of r15 in T flag
brts ok   ; Branch if T flag set
...       
ok: nop   ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1
ORI - Logical OR with Immediate

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

Operation:
(i) Rd ← Rd v K

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

16-bit Opcode:

| 0110 | KKKK | dddd | KKKK |

Status Register (SREG) and Boolean Formula:

<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:
\[
\text{ori } r16,\$F0 \quad ; \text{Set high nibble of } r16 \\
\text{ori } r17,1 \quad ; \text{Set bit 0 of } r17
\]

Words: 1 (2 bytes)
Cycles: 1
OUT - Store Register to I/O Location

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

Operation:
(i) I/O(A) ← Rr

Syntax: Operands: Program Counter:
(i) OUT A,Rr 0 ≤ r ≤ 31, 0 ≤ A ≤ 63 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>

Status Register (SREG) and Boolean Formula:

Example:
- clr r16 ; Clear r16
- ser r17 ; Set r17
- out $18,r16 ; Write zeros to Port B
- nop ; Wait (do nothing)
- out $18,r17 ; Write ones to Port B

Words: 1 (2 bytes)
Cycles: 1
POP - Pop Register from Stack

Description:
This instruction loads register Rd with a byte from the STACK. The stack pointer is pre-incremented by 1 before the POP.

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

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

16-bit Opcode:

```
1001 000d dddd 1111
```

Status Register (SREG) and Boolean Formula:

<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. The stack pointer is post-decremented by 1 after the PUSH.

Operation:
(i) STACK ← Rr

Syntax: Operands: Program Counter: Stack:
(i) PUSH Rr 0 ≤ r ≤ 31 PC ← PC + 1 SP ← SP - 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>01</td>
<td>dddd</td>
<td>1111</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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:
Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the stack. (See also CALL). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. The stack pointer uses a post-decrement scheme during RCALL.

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

Syntax: Operands: Program Counter: Stack:
(i) RCALL k  -2K ≤ k < 2K  PC ← PC + k + 1  STACK ← PC + 1
 SP ← SP - 2 (2 bytes, 16 bits)
(ii) RCALL k  -2K ≤ k < 2K  PC ← PC + k + 1  STACK ← PC + 1
 SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

| 1101 | kkkk | kkkk | kkkk |

Status Register (SREG) and Boolean Formula:

<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:

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 devices with 16-bit PC
4 devices with 22-bit PC
RET - Return from Subroutine

Description:
Returns from subroutine. The return address is loaded from the STACK. The stack pointer uses a pre-increment scheme during RET.

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

Syntax: Operands: Program Counter: Stack:
(i) RET None See Operation \( \text{SP} \leftarrow \text{SP} + 2 \), (2bytes,16 bits)
(ii) RET None See Operation \( \text{SP} \leftarrow \text{SP} + 3 \), (3bytes,22 bits)

16-bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0101</td>
<td>0000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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 devices with 16-bit PC
5 devices with 22-bit PC
**RETI - Return from Interrupt**

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

Note that the status register is not automatically stored when entering an interrupt routine, and it is not restored when returning from an interrupt routine. This must be handled by the application program. The stack pointer uses a pre-increment scheme during RETI.

**Operation:**

(i) \( \text{PC}(15:0) \leftarrow \text{STACK} \)  
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:**  
(i) RETI None See Operation

(ii) RETI None See Operation

**Program Counter:**  
(i) SP \( \leftarrow \) SP + 2 (2 bytes, 16 bits)

(ii) SP \( \leftarrow \) SP + 3 (3 bytes, 22 bits)

**16-bit Opcode:**

```
1001 0101 0001 1000
```

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

<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>

**Example:**

```
...  
extint: push r0 ; Save r0 on the stack
...  
pop r0 ; Restore r0
reti ; Return and enable interrupts
```

**Words:** 1 (2 bytes)

**Cycles:** 4 devices with 16-bit PC

5 devices with 22-bit PC
RJMP - Relative Jump

Description:
Relative jump to an address within PC - 2K +1 and PC + 2K (words). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location.

Operation:
(i) PC ← PC + k + 1

Syntax: Operands: Program Counter: Stack
(i) RJMP k -2K ≤ k < 2K PC ← PC + k + 1 Unchanged

16-bit Opcode:

Status Register (SREG) and Boolean Formula:

Example:

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. This operation, combined with LSL, effectively multiplies multi-byte signed and unsigned values by two.

Operation:

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

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

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

Status Register (SREG) and Boolean Formula:

<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)
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:

```
ls1 r18 ; Multiply r19:r18 by two
rol r19 ; r19:r18 is a signed or unsigned two-byte integer
brcs oneenc ; Branch if carry set
...
oneenc: nop ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1
ROR - Rotate Right through Carry

Description:
Shifts all bits in Rd one place to the right. The C flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR it effectively divides multi-byte unsigned values by two. The carry flag can be used to round the result.

Operation:

```
C → b7 ... b0 → C
```

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 Formula:

<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)

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

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

C:  
Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.
Example:

```
lsr  r19 ; Divide r19:r18 by two
ror  r18 ; r19:r18 is an unsigned two-byte integer
brcc zeroenc1 ; Branch if carry cleared
asr  r17 ; Divide r17:r16 by two
ror  r16 ; r17:r16 is a signed two-byte integer
brcc zeroenc2 ; Branch if carry cleared
...
zeroenc1:  nop ; Branch destination (do nothing)
...
zeroenc1:  nop ; Branch destination (do nothing)
```

**Words:** 1 (2 bytes)

**Cycles:** 1
SBC - Subtract with Carry

Description:
Subtracts two registers and subtracts with the C flag and places the result in the destination register Rd.

Operation:
(i) \( Rd \leftarrow Rd - Rr - C \)

Syntax: Operands: Program Counter:
(i) SBC Rd,Rr 0 \( \leq d \leq 31 \), 0 \( \leq r \leq 31 \) 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>0000</td>
<td>10rd</td>
<td>dddd</td>
<td>rrrr</td>
</tr>
</tbody>
</table>

Status Register and Boolean Formula:

<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: \( Rd3 \oplus Rr3 + Rr3 + R3 + R3 \cdot Rd3 \)
Set if there was a borrow from bit 3; cleared otherwise

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

V: \( Rd7 \cdot R7 \cdot R7 + Rd7 \cdot R7 \cdot R7 \)
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 \cdot Z
Previous value remains unchanged when the result is zero; cleared otherwise.

C: \( Rd7 \cdot Rr7 + Rr7 \cdot R7 + R7 \cdot Rd7 \)
Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of the Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

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

Words: 1 (2 bytes)
Cycles: 1
SBCI - Subtract Immediate with Carry

Description:
Subtracts a constant from a register and subtracts with the C flag and places the result in the destination register Rd.

Operation:
(i) \( Rd \leftarrow Rd - K - C \)

Syntax: 
(i) SBCI Rd,K

Operands:
16 \( \leq d \leq 31 \), 0 \( \leq K \leq 255 \)

Status Register and Boolean Formula:

<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: \( Rd^3 \cdot K^3 + K^3 \cdot Rd + R^3 \cdot Rd^3 \)
Set if there was a borrow from bit 3; cleared otherwise

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

V: \( Rd \cdot K^7 \cdot R^7 + Rd^7 \cdot K^7 \cdot R^7 \)
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: \( R^7 \cdot R^6 \cdot R^5 \cdot R^4 \cdot R^3 \cdot R^2 \cdot R^1 \cdot R^0 \cdot Z \)
Previous value remains unchanged when the result is zero; cleared otherwise.

C: \( Rd^7 \cdot K^7 + K^7 \cdot R^7 + R^7 \cdot Rd^7 \)
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:

\[
\text{subi } r16,\$23 \quad \text{; Subtract low byte}
\]
\[
\text{sbc } r17,\$4F \quad \text{; Subtract with carry high byte}
\]

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

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

16-bit Opcode:

\[
\begin{array}{cccc}
1001 & 1010 & AAAA & Abbb \\
\end{array}
\]

Status Register (SREG) and Boolean Formula:

<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:
- `out $1E, r0 ; Write EEPROM address`
- `sbi $1C, 0 ; Set read bit in EECR`
- `in r1, $1D ; Read EEPROM data`

Words: 1 (2 bytes)  
Cycles: 2
SBIC - Skip if Bit in I/O Register is Cleared

Description:
This instruction tests a single bit in an I/O register and skips the next instruction if the bit is cleared. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:
(i) If I/O(A,b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) SBIC A,b 0 ≤ A ≤ 31, 0 ≤ b ≤ 7 PC ← PC + 1, Condition false - no skip
PC ← PC + 2, Skip a one word instruction
PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

| 1001 | 1001 | AAAA | Abbb |

Status Register (SREG) and Boolean Formula:

<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:

```
e2wait: sbic $1C,1 ; Skip next inst. if EEWE cleared
rjmp e2wait ; EEPROM write not finished
nop ; Continue (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false (no skip)
2 if condition is true (skip is executed) and the instruction skipped is 1 word
3 if condition is true (skip is executed) and the instruction skipped is 2 words
SBIS - Skip if Bit in I/O Register is Set

Description:
This instruction tests a single bit in an I/O register and skips the next instruction if the bit is set. This instruction operates on the lower 32 I/O registers - addresses 0-31.

Operation:
(i) If I/O(A,b) = 1 then PC ← PC + 2 (or 3) else PC ← PC + 1

Syntax: Operands: Program Counter:
(i) SBIS A,b 0 ≤ A ≤ 31, 0 ≤ b ≤ 7

PC ← PC + 1, Condition false - no skip
PC ← PC + 2, Skip a one word instruction
PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

| 1001 | 1011 | AAAA | Abbb |

Status Register (SREG) and Boolean Formula:

<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: 1 if condition is false (no skip)
2 if condition is true (skip is executed) and the instruction skipped is 1 word
3 if condition is true (skip is executed) and the instruction skipped is 2 words
SBIW - Subtract Immediate from Word

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

Operation:
(i) \( \text{Rd+1:Rd} \leftarrow \text{Rd+1:Rd} - K \)

Syntax:
(i) SBIW Rd,K
d \in \{24,26,28,30\}, \quad 0 \leq K \leq 63

Program Counter:
PC \leftarrow PC + 1

16-bit Opcode:
1001 0111 KKdd KKKK

Status Register (SREG) and Boolean Formula:

<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>\leftrightarrow</td>
<td>\leftrightarrow</td>
<td>\leftrightarrow</td>
<td>\leftrightarrow</td>
<td>\leftrightarrow</td>
</tr>
</tbody>
</table>

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

V: \quad \text{Rdh7} \cdot \text{R15}
Set if two's complement overflow resulted from the operation; cleared otherwise.

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

Z: \quad \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: \quad \text{R15} \cdot \text{Rdh7}
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R \text{ (Result)} equals Rdh:Rdl after the operation \( \text{Rdh7-Rdh0} = \text{R15-R8}, \text{Rd17-Rd10} = \text{R7-R0} \).

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

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

16-bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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: \( \text{N} \oplus \text{V} \), For signed tests.
V:
\( \text{0} \)
Cleared
N:
\( \text{R7} \)
Set if MSB of the result is set; cleared otherwise.
Z:
\( \text{R7•R6•R5•R4•R3•R2•R1•R0} \)
Set if the result is $00$; cleared otherwise.

R (Result) equals Rd after the operation.

Example:
\[
\begin{align*}
\text{sbr} & \ \text{r16},3 \quad ; \text{Set bits 0 and 1 in r16} \\
\text{sbr} & \ \text{r17},\text{F0} \quad ; \text{Set 4 MSB in r17}
\end{align*}
\]

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 \( PC \leftarrow PC + 2 \) (or 3) else \( PC \leftarrow PC + 1 \)

Syntax: \( \text{SBRC} \ Rr,b \)
Operands: \( 0 \leq r \leq 31, \ 0 \leq b \leq 7 \)

Program Counter:
- \( PC \leftarrow PC + 1 \), Condition false - no skip
- \( PC \leftarrow PC + 2 \), Skip a one word instruction
- \( PC \leftarrow PC + 3 \), Skip a two word instruction

16-bit Opcode:

| 1111 | 110r | rrrr | 0bbb |

Status Register (SREG) and Boolean Formula:

<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) and the instruction skipped is 1 word
- 3 if condition is true (skip is executed) and the instruction skipped is 2 words
SBRS - Skip if Bit in Register is Set

Description:
This instruction tests a single bit in a register and skips the next instruction if the bit is set.

Operation:
(i) If \( Rr(b) = 1 \) then \( PC \leftarrow PC + 2 \) (or 3) else \( PC \leftarrow PC + 1 \)

Syntax: \( \text{SBRS} \) \( Rr,b \)
Operands: \( 0 \leq r \leq 31, 0 \leq b \leq 7 \)

Program Counter:
- \( PC \leftarrow PC + 1 \), Condition false - no skip
- \( PC \leftarrow PC + 2 \), Skip a one word instruction
- \( PC \leftarrow PC + 3 \), Skip a two word instruction

16-bit Opcode:

| 1111 | 111r | rrrr | 0bbb |

Status Register (SREG) and Boolean Formula:

<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) and the instruction skipped is 1 word
- 3 if condition is true (skip is executed) and the instruction skipped is 2 words
SEC - Set Carry Flag

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

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

**Syntax:**
(i) SEC

**Operands:**
None

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

**16-bit Opcode:**
\[
\begin{array}{cccc}
1001 & 0100 & 0000 & 1000 \\
\end{array}
\]

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

<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:**
```
sec ; Set carry flag
adc r0, r1 ; r0=r0+r1+1
```

**Words:** 1 (2 bytes)

**Cycles:** 1
SEH - Set Half Carry Flag

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

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

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

16-bit Opcode:

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

Status Register (SREG) and Boolean Formula:

<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:

```
seh ; 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:

```
1001 0100 0111 1000
```

Status Register (SREG) and Boolean Formula:

```
<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 PC \( \leftarrow \) PC + 1

16-bit Opcode:

```
1001 0100 0010 1000
```

Status Register (SREG) and Boolean Formula:

<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:
```
add r2, r19 ; Add r19 to r2
sen ; Set negative flag
```

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

Description:
Loads $FF$ directly to register Rd.

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

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

16-bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1110</td>
<td>1111</td>
<td>dddd</td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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:
- \( \text{clr} \ r16 \); Clear r16
- \( \text{ser} \ r17 \); Set r17
- \( \text{out} \ \$18, r16 \); Write zeros to Port B
- \( \text{nop} \); Delay (do nothing)
- \( \text{out} \ \$18, r17 \); 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: Operands: Program Counter:
(i) SES None PC \( \leftarrow \) PC + 1

16-bit Opcode:

| 0100 | 0100 | 0100 | 1000 |

Status Register (SREG) and Boolean Formula:

<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:
```
add r2,r19 ; Add r19 to r2
ses ; Set negative flag
```

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

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

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

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

16-bit Opcode:

\[
\begin{array}{cccc}
1001 & 0100 & 0110 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formula:

<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:

\[
\begin{array}{c}
\text{set} \quad ; \text{Set T flag}
\end{array}
\]

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

16-bit Opcode:

| 1001 | 0100 | 0011 | 1000 |

Status Register (SREG) and Boolean Formula:

<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>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

V: 1
Overflow flag set

Example:

```
add r2, r19 ; Add r19 to r2
sev ; Set overflow flag
```

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

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

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

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

16-bit Opcode:

\[
\begin{array}{cccc}
1001 & 0100 & 0001 & 1000 \\
\end{array}
\]

Status Register (SREG) and Boolean Formula:

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

Z: 1
Zero flag set

Example:

\[
\begin{array}{l}
\text{add r2, r19} \quad ; \text{Add r19 to r2} \\
\text{sez} \quad ; \text{Set zero flag}
\end{array}
\]

Words: 1 (2 bytes)
Cycles: 1
SLEEP

Description:
This instruction sets the circuit in sleep mode defined by the MCU control register.

Operation:
Refer to the device documentation for detailed description of SLEEP usage.

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

16-bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0101</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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
ldi r16,(1<<SE) ; Enable sleep mode
out MCUCR, r16
sleep ; Put MCU in sleep mode
```

Words: 1 (2 bytes)
Cycles: 1
SPM - Store Program Memory

Description:
SPM can be used to erase a page in the program memory, to write a page in the program memory (that is already erased), and to set boot loader lock bits. In some devices, the program memory can be written one word at a time, in other devices an entire page can be programmed simultaneously after first filling a temporary page buffer. In all cases, the program memory must be erased one page at a time. When erasing the program memory, the Z register is used as page address. When writing the program memory, the Z register is used as page or word address, and the R1:R0 register pair is used as data. When setting the boot loader lock bits, the R1:R0 register pair is used as data. Refer to the device documentation for detailed description of SPM usage. This instruction can address the first 64K bytes (32K words) of program memory.

Operation:
(i) (Z) ← $fff
(ii) (Z) ← R1:R0
(iii) (Z) ← R1:R0
(iv) (Z) ← TEMP
(v) BLBITS ← R1:R0

Comment:
Erase program memory page
Write program memory word
Write temporary page buffer
Write temporary page buffer to program memory
Set boot loader lock bits

Syntax: Operands: Program Counter:
(i)-(v) SPM None PC ← PC + 1

16-bit Opcode:

| 1001 | 0101 | 1110 | 1000 |

Status Register (SREG) and Boolean Formula:

<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:
; This example shows SPM write of one word for devices with word write
ldi r31, $F0 ; Load Z high byte
clr r30 ; Clear Z low byte
ldi r16, $CF ; Load data to store
mov r1, r16
ldi r16, $FF
mov r0, r16
ldi r16, $03 ; Enable SPM, erase page
out SPMCR, r16 ;
spm ; Erase page starting at $F000
ldi r16, $01 ; Enable SPM, store to program memory
out SPMCR, r16 ;
spm ; Execute SPM, store R1:R0 to program memory location $F000

Words: 1 (2 bytes)
Cycles: depends on the operation
ST - Store Indirect From Register to data space using Index X

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

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

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

The result of these combinations is undefined:

\[
\begin{align*}
\text{ST X+, r26} \\
\text{ST X+, r27} \\
\text{ST -X, r26} \\
\text{ST -X, r27}
\end{align*}
\]

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: Operands: Program Counter:

<table>
<thead>
<tr>
<th>(i)</th>
<th>(ii)</th>
<th>(iii)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ST X, Rr)</td>
<td>(0 ≤ r ≤ 31)</td>
<td>(PC ← PC + 1)</td>
</tr>
<tr>
<td>(ST X+, Rr)</td>
<td>(0 ≤ r ≤ 31)</td>
<td>(PC ← PC + 1)</td>
</tr>
<tr>
<td>(ST -X, Rr)</td>
<td>(0 ≤ r ≤ 31)</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>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1110</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

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

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

Words: 1 (2 bytes)

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

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

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

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

The result of these combinations is undefined:

- ST Y+, r28
- ST Y+, r29
- ST -Y, r28
- ST -Y, r29

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>(iii) (Y+q) ← Rr</td>
<td>Y: Unchanged, q: Displacement</td>
</tr>
</tbody>
</table>

Syntax: Operands: Program Counter:

- ST Y, Rr 0 ≤ r ≤ 31 PC ← PC + 1
- ST Y+, Rr 0 ≤ r ≤ 31 PC ← PC + 1
- ST -Y, Rr 0 ≤ r ≤ 31 PC ← PC + 1
- STD Y+q, Rr 0 ≤ r ≤ 31, 0 ≤ q ≤ 63 PC ← PC + 1

16-bit Opcode:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>16-bit Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1000 001r rrrr 1000</td>
</tr>
<tr>
<td>(ii)</td>
<td>1001 001r rrrr 1000</td>
</tr>
<tr>
<td>(iii)</td>
<td>1001 001r rrrr 1010</td>
</tr>
<tr>
<td>(iii)</td>
<td>10q0 qqqr rrrr 1qqq</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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,$60 ; Set Y low byte to $60
st   Y+,r0 ; Store r0 in data space loc. $60(Y post inc)
st   Y,r1 ; Store r1 in data space loc. $61
ldi  r28,$63 ; Set Y low byte to $63
st   Y,r2 ; Store r2 in data space loc. $63
st   -Y,r3 ; Store r3 in data space loc. $62(Y pre dec)
std  Y+2,r4 ; Store r4 in data space loc. $64
```

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

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

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

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

The result of these combinations is undefined:
- ST Z+, r30
- ST Z+, r31
- ST -Z, r30
- ST -Z, r31

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: Post incremented</td>
</tr>
<tr>
<td>(iii) Z ← Z - 1</td>
<td>Z: Pre decremented</td>
</tr>
<tr>
<td>(iii) (Z+q) ← Rr</td>
<td>Z: Unchanged, q: Displacement</td>
</tr>
</tbody>
</table>

Syntax: Operands: Program Counter:

<table>
<thead>
<tr>
<th>(i)</th>
<th>(ii)</th>
<th>(iii)</th>
<th>(iii)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Z, Rr</td>
<td>ST Z+, Rr</td>
<td>ST -Z, Rr</td>
<td>STD Z+q, Rr</td>
</tr>
<tr>
<td>0 ≤ r ≤ 31</td>
<td>0 ≤ r ≤ 31</td>
<td>0 ≤ r ≤ 31</td>
<td>0 ≤ r ≤ 31, 0 ≤ q ≤ 63</td>
</tr>
<tr>
<td>PC ← PC + 1</td>
<td>PC ← PC + 1</td>
<td>PC ← PC + 1</td>
<td>PC ← PC + 1</td>
</tr>
</tbody>
</table>
16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>001r</th>
<th>rrrr</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ii</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>0001</td>
</tr>
<tr>
<td>iii</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>0010</td>
</tr>
<tr>
<td>iiii</td>
<td>10q0</td>
<td>qq1r</td>
<td>rrrr</td>
<td>0qqq</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<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 r31 ; Clear Z high byte
ldi r30,$60 ; Set Z low byte to $60
st Z+,r0 ; Store r0 in data space loc. $60(Z post inc)
st Z,r1 ; Store r1 in data space loc. $61
ldi r30,$63 ; Set Z low byte to $63
st Z,r2 ; Store r2 in data space loc. $63
st -Z,r3 ; Store r3 in data space loc. $62(Z pre dec)
std Z+2,r4 ; Store r4 in data space loc. $64
```

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

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

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

Operation:
(i) (k) ← Rr

Syntax: Operands: Program Counter:
(i) STS k,Rr 0 ≤ r ≤ 31, 0 ≤ k ≤ 65535 PC ← PC + 2

32-bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>001d</td>
<td>dddd</td>
<td>0000</td>
</tr>
<tr>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formula:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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:
1ds r2,$FF00 ; Load r2 with the contents of data space location $FF00
add r2,r1 ; add r1 to r2
sts $FF00,r2 ; Write back

Words: 2 (4 bytes)

Cycles: 2
SUB - Subtract without Carry

Description:
Subtracts two registers and places the result in the destination register Rd.

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

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

16-bit Opcode:

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & rd & ddd \_ rrrr
\end{array}
\]

Status Register and Boolean Formula:

<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: \( \overline{Rd3 \cdot Rr3 + Rr3} \cdot R3 + Rr3 \cdot R\overline{d3} \)
Set if there was a borrow from bit 3; cleared otherwise.

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

V: \( \overline{Rd7 \cdot Rr7} \cdot \overline{Rr7} + Rr7 \cdot R7 \)
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: \( \overline{R7 \cdot R6} \cdot \overline{Rr5} \cdot R4 \cdot R3 \cdot Rr2 \cdot R1 \cdot R0 \)
Set if the result is $00$; cleared otherwise.

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

\[
\begin{align*}
\text{sub} & \quad r13,r12 \quad \text{; Subtract r12 from r13} \\
\text{brne} & \quad \text{noteq} \quad \text{; Branch if r12<>r13} \\
\ldots & \\
\text{noteq} & \quad \text{nop} \quad \text{; Branch destination (do nothing)}
\end{align*}
\]

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) \( Rd \leftarrow Rd - K \)

Syntax:
(i) SUBI Rd,K

Operands:
16 \( \leq d \leq 31 \), 0 \( \leq K \leq 255 \)

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

16-bit Opcode:

| Opcode | 0101 | KKKK | dddd | KKKK |

Status Register and Boolean Formula:

<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: \( \overline{Rd3} \oplus K3 \oplus R3 \oplus R3 \oplus \overline{Rd3} \)
Set if there was a borrow from bit 3; cleared otherwise.

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

V: \( Rd7 \oplus K7 \oplus R7 \oplus \overline{Rd7} \oplus K7 \oplus R7 \)
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: \( R7 \oplus R6 \oplus R5 \oplus R4 \oplus R3 \oplus R2 \oplus R1 \oplus R0 \)
Set if the result is $00; cleared otherwise.

C: \( \overline{Rd7} \oplus K7 \oplus R7 \oplus R7 \oplus \overline{Rd7} \)
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:
```
subi r22,$11 ; Subtract $11 from r22
brne noteq ; Branch if r22<>$11
...
noteq:  nop  ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1
SWAP - Swap Nibbles

Description:
Swaps high and low nibbles in a register.

Operation:
(i) \( R(7:4) \leftarrow Rd(3:0), R(3:0) \leftarrow Rd(7:4) \)

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

16-bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>010d</th>
<th>dddd</th>
<th>0010</th>
</tr>
</thead>
</table>

Status Register and Boolean Formula:

<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>

R (Result) equals Rd after the operation.

Example:

```assembly
inc r1 ; Increment r1
swap r1 ; Swap high and low nibble of r1
inc r1 ; Increment high nibble of r1
swap r1 ; Swap back
```

Words: 1 (2 bytes)
Cycles: 1
TST - Test for Zero or Minus

Description:
Tests if a register is zero or negative. Performs a logical AND between a register and itself. The register will remain unchanged.

Operation:
(i) $Rd \leftarrow Rd \cdot Rd$

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

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

<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>$\leftrightarrow$</td>
<td>0</td>
<td>$\leftrightarrow$</td>
<td>$\leftrightarrow$</td>
<td>-</td>
</tr>
</tbody>
</table>

Status Register and Boolean Formula:
- $S$: $N \oplus V$, 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:
(i) WDR

Operands:
None

Program Counter:
PC ← PC + 1

16-bit Opcode:

| 1001 | 0101 | 1010 | 1000 |

Status Register and Boolean Formula:

<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