LAMPIRAN A

LISTING PROGRAM MIKROKONTROLER
.nolist ; list file untuk include file tidak perlu ditampilkan
.include "d:\Program Files\Atmel\AVR Tools\AvrAssembler\Appnotes\2313def.inc"

.list
.EQU fq=4000000 ; XTal-frequency
.EQU baud=9600 ; Baudrate
.EQU bdteiler=(fq/(16*baud))-1 ; Baud-Divider
.EQU RamStart = 0x0060
.def temp1=r1
.def temp2=r2
.def temp3=r3
.def temp4=r4
.def temp5=r5
.DEF mpr=R16 ; Universal register
.DEF cc=R17 ; Char copy
.DEF h=R18 ; Various values
.cseg
.org 0

ldi XH,HIGH(RamStart)
  ldi XL,LOW(RamStart)
  ldi YH,HIGH(RamStart)
  ldi YL,LOW(RamStart)
  ldi mpr,0x0D ; Start with a new line
  st X+,mpr
  ldi mpr,0x0A
  st X+,mpr
  ldi mpr,bdteiler ; Set baudrate generator
  out UBRR,mpr ; to divider port
  ldi mpr,0b00011000 ; Enable TX and RX
  out UCR,mpr ; to UART

  tloop:
    sbic USR,RXC ; Jump if receiver is empty
    rjmp rx ; Receive the next char
    rjmp tloop

  rx:
    ldi mpr,','
    st X+,mpr ; Store it in the SRAM buffer and inc the pointer
    in mpr,UDR ; Get a char from the UART receiver port
    mov cc,mpr
    swap mpr
    andi mpr,0x0F
cpi mpr,10
brcs rx1
ldi h,7 ; Add 7 to get hex A to E
add mpr,h

rx1:
    ldi h,'0' ; from 0 to '0'
    add mpr,h
    st X+,mpr ; and copy to SRAM
    andi cc,0x0F ; Same procedure with the lower nibble
    cpi cc,10
    brcs rx2
    ldi h,7
    add mpr,h

rx2:
    ldi h,'0'
    add cc,h
    st X+,cc
    ldi cc,'h'
    st X+,cc
    rjmp tloop

    ldi temp4,'a'
    cpse mpr,temp4,tulisa
    ldi temp4,'u'
    cpse mpr,temp4,tulisu
    ldi temp4,'i'
    cpse mpr,temp4,tulisi
    ldi temp4,'e'
    cpse mpr,temp4,tulise
    ldi temp4,'o'
    cpse mpr,temp4,tuliso
    ldi temp4,'0'
    cpse mpr,temp4,tuliso
    ldi temp4,'1'
    cpse mpr,temp4,tulis1
    ldi temp4,'2'
    cpse mpr,temp4,tulis2
    ldi temp4,'3'
    cpse mpr,temp4,tulis3
    ldi temp4,'4'
    cpse mpr,temp4,tulis4
    ldi temp4,'5'
    cpse mpr,temp4,tulis5
    ldi temp4,'6'
cpse mpr,temp4,tulis6
ldi temp4,‘7’
cpse mpr,temp4,tulis7
ldi temp4,‘8’
cpse mpr,temp4,tulis8
ldi temp4,‘9’
cpse mpr,temp4,tulis9

tulisa: ldi temp2,0hff
    ldi ddrb,temp2
    ldi temp3,0h3f
        out portb,temp3
        call delay
        ldi temp3,0h48
        out portb,temp3
        call delay
        ldi temp3,0h48
        out portb,temp3
        call delay
        ldi temp3,0h48
        out portb,temp3
        call delay
        ldi temp3,0h3f
        out portb,temp3
        call delay
        ldi temp3,0h00
        out portb,temp3
        call delay
        call delay
        call delay
        call delay
    ret

tulisi: ldi temp2,0hff
    ldi ddrb,temp2
    ldi temp3,0h41
        out portb,temp3
        call delay
        ldi temp3,0h41
        out portb,temp3
        call delay
        ldi temp3,0h41
        out portb,temp3
        call delay
        ldi temp3,0h7f
        out portb,temp3
        call delay

delay
ldi temp3,0h41
out portb,temp3
call delay
ldi temp3,0h41
out portb,temp3
call delay
ldi,temp3,0h00
out portb,temp3
call delay
call delay
call delay
ret


tulisu: ldi temp2,0hff
    ldi ddrb,temp2
    ldi temp3,0h7e
        out portb,temp3
call delay
    ldi temp3,0h01
        out portb,temp3
call delay
    ldi temp3,0h01
        out portb,temp3
call delay
    ldi temp3,0h01
        out portb,temp3
call delay
    ldi temp3,0h7e
        out portb,temp3
call delay
    ldi,temp3,0h00
        out portb,temp3
call delay
call delay
call delay
ret

tulise: ldi temp2,0hff
    ldi ddrb,temp2
    ldi temp3,0h7f
        out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h00
out portb,temp3
call delay
call delay
call delay
ret

tuliso: ldi temp2,0hff
  ldi ddrb,temp2
  ldi temp3,0h3e
    out portb,temp3
call delay
  ldi temp3,0h41
    out portb,temp3
call delay
  ldi temp3,0h41
    out portb,temp3
call delay
  ldi temp3,0h41
    out portb,temp3
call delay
  ldi temp3,0h41
    out portb,temp3
call delay
  ldi temp3,0h3e
    out portb,temp3
call delay
  ldi,temp3,0h00
    out portb,temp3
call delay
call delay
call delay
ret
tulis1: ldi temp2,0xff
    ldi ddrb,temp2
    ldi temp3,0x00
    out portb,temp3
    call delay
    ldi temp3,0x00
    out portb,temp3
    call delay
    ldi temp3,0x7f
    out portb,temp3
    call delay
    ldi temp3,0x00
    out portb,temp3
    call delay
    ldi temp3,0x00
    out portb,temp3
    call delay
    ldi temp3,0x00
    out portb,temp3
    call delay
    ret

    call delay
    call delay

    call delay

    call delay

    call delay

    call delay

tulis2: ldi temp2,0xff
    ldi ddrb,temp2
    ldi temp3,0x21
    out portb,temp3
    call delay
    ldi temp3,0x43
    out portb,temp3
    call delay
    ldi temp3,0x45
    out portb,temp3
    call delay
    ldi temp3,0x49
    out portb,temp3
    call delay
    ldi temp3,0x31
    out portb,temp3
    call delay
    ldi temp3,0x00
    out portb,temp3
    call delay
    ldi,temp3,0x00
    out portb,temp3
    call delay

    call delay
call delay
ret

tulis3: ldi temp2,0xff
  ldi ddrb,temp2
  ldi temp3,0x22
  out portb,temp3
  call delay
  ldi temp3,0x41
  out portb,temp3
  call delay
  ldi temp3,0x49
  out portb,temp3
  call delay
  ldi temp3,0x49
  out portb,temp3
  call delay
  ldi temp3,0x36
  out portb,temp3
  call delay
  ldi temp3,0x00
  out portb,temp3
  call delay
  call delay
  ret

tulis4: ldi temp2,0xff
  ldi ddrb,temp2
  ldi temp3,0x04
  out portb,temp3
  call delay
  ldi temp3,0x0b
  out portb,temp3
  call delay
  ldi temp3,0x14
  out portb,temp3
  call delay
  ldi temp3,0x24
  out portb,temp3
  call delay
  ldi temp3,0x7f
  out portb,temp3
call delay
ldi,temp3,0h00
out portb,temp3
call delay
call delay
call delay
ret

ret

tulis5: ldi temp2,0hff
   ldi ddrb,temp2
   ldi temp3,0h79
      out portb,temp3
call delay
   ldi temp3,0h49
      out portb,temp3
call delay
   ldi temp3,0h49
      out portb,temp3
call delay
   ldi temp3,0h49
      out portb,temp3
call delay
   ldi temp3,0h46
      out portb,temp3
call delay
   ldi,temp3,0h00
      out portb,temp3
call delay
call delay
call delay
ret

tulis6: ldi temp2,0hff
   ldi ddrb,temp2
   ldi temp3,0h3e
      out portb,temp3
call delay
   ldi temp3,0h49
      out portb,temp3
call delay
   ldi temp3,0h49
      out portb,temp3
call delay
   ldi temp3,0h49
      out portb,temp3
call delay
ret
out portb,temp3
call delay
ldi temp3,0h26
out portb,temp3
call delay
ldi,temp3,0h00
out portb,temp3
call delay
call delay
call delay
ret


tulis7: ldi temp2,0hff
ldi ddrb,temp2
ldi temp3,0h41
out portb,temp3
call delay
ldi temp3,0h42
out portb,temp3
call delay
ldi temp3,0h4c
out portb,temp3
call delay
ldi temp3,0h48
out portb,temp3
call delay
ldi temp3,0h68
out portb,temp3
call delay
ldi,temp3,0h00
out portb,temp3
call delay
call delay
call delay
ret


tulis8: ldi temp2,0hff
ldi ddrb,temp2
ldi temp3,0h36
out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h49
out portb,temp3
call delay
ldi temp3,0h36
out portb,temp3
call delay
ldi,temp3,0h00
out portb,temp3
call delay
call delay
ret

tulis9: ldi temp2,0hff
  ldi ddrb,temp2
  ldi temp3,0h32
    out portb,temp3
call delay
  ldi temp3,0h49
    out portb,temp3
call delay
  ldi temp3,0h49
    out portb,temp3
call delay
  ldi temp3,0h49
    out portb,temp3
call delay
  ldi temp3,0h49
    out portb,temp3
call delay
  ldi temp3,0h3e
    out portb,temp3
call delay
  ldi,temp3,0h00
    out portb,temp3
call delay
call delay
call delay
ret
delay: ldi temp4,0b00000000
  ldi temp3,0b10000000
lg:  nop
    nop
nop
nop
nop
dec temp3
cpse temp3,temp4,selesai
jmp lg
selesai: ret
.exit
LAMPIRAN B
LISTING PROGRAM BORLAND DELPHI
unit U_HANDY;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, AfrPortControls, AfrDataDispatcher, AfrComPort, Buttons,
  ExtCtrls, ComCtrls;

type
  TForm1 = class(TForm)
    AfrComPort1: TAfComPort;
    AfrPortComboBox1: TAfPortComboBox;
    GroupBox1: TGroupBox;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn7: TBitBtn;
    BitBtn8: TBitBtn;
    BitBtn9: TBitBtn;
    BitBtn10: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Image1: TImage;
    Label3: TLabel;
    BitBtn11: TBitBtn;
    BitBtn12: TBitBtn;
    BitBtn13: TBitBtn;
    BitBtn14: TBitBtn;
    BitBtn15: TBitBtn;
    BitBtn16: TBitBtn;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    Panel1: TPanel;
  procedure BitBtn1Click(Sender: TObject);
  procedure BitBtn2Click(Sender: TObject);
  procedure BitBtn3Click(Sender: TObject);
  procedure BitBtn4Click(Sender: TObject);
  procedure BitBtn5Click(Sender: TObject);
  procedure BitBtn16Click(Sender: TObject);
  procedure BitBtn7Click(Sender: TObject);
  procedure BitBtn8Click(Sender: TObject);
procedure BitBtn9Click(Sender: TObject);
procedure BitBtn10Click(Sender: TObject);
procedure BitBtn11Click(Sender: TObject);
procedure BitBtn12Click(Sender: TObject);
procedure BitBtn13Click(Sender: TObject);
procedure BitBtn14Click(Sender: TObject);
procedure BitBtn15Click(Sender: TObject);

private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.dfm}$

var
  x : string ;
procedure TForm1.BitBtn1Click(Sender: TObject);
beginn
  x := 'a';
  form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
beginn
  x := 'u';
  form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.BitBtn3Click(Sender: TObject);
beginn
  x := 'i';
  form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.BitBtn4Click(Sender: TObject);
begin
x := 'o';
form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn5Click(Sender: TObject);
begin
x := 'e';
form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn16Click(Sender: TObject);
begin
x := '1';
form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn7Click(Sender: TObject);
begin
x := '2';
form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn8Click(Sender: TObject);
begin
x := '3';
form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn9Click(Sender: TObject);
begin
x := '4';
form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn10Click(Sender: TObject);
begin
x := '5';
form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn11Click(Sender: TObject);
begin
  x := '6';
  form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn12Click(Sender: TObject);
begin
  x := '7';
  form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn13Click(Sender: TObject);
begin
  x := '8';
  form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn14Click(Sender: TObject);
begin
  x := '9';
  form1.AfComPort1.WriteData(x,1);
end;

procedure TForm1.Btn15Click(Sender: TObject);
begin
  x := '0';
  form1.AfComPort1.WriteData(x,1);
end;
end.
Instruction Set Nomenclature:

**Status Register (SREG):**
SREG: Status register
C: Carry flag in status register
Z: Zero flag in status register
N: Negative flag in status register
V: Twos complement overflow indicator
S: \( N \oplus V \), For signed tests
H: Half Carry flag in the status register
T: Transfer bit used by BLD and BST instructions
I: Global interrupt enable/disable flag

**Registers and operands:**
Rd: Destination (and source) register in the register file
Rr: Source register in the register file
R: Result after instruction is executed
K: Constant literal or byte data (8 bit)
k: Constant address data for program counter
b: Bit in the register file or I/O register (3 bit)
s: Bit in the status register (3 bit)
X,Y,Z: Indirect address register (X=R27:R26, Y=R29:R28 and Z=R31:R30)
P: I/O port address
q: Displacement for direct addressing (6 bit)

**I/O Registers**
RAMPX, RAMPY, RAMPZ: Registers concatenated with the X, Y and Z registers enabling indirect addressing of the whole SRAM area on MCUs with more than 64K bytes SRAM.

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

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

* Interchange Rd and Rr in the operation before the test. i.e. CP Rd,Rr \( \rightarrow \) CP Rr,Rd
## Complete Instruction Set Summary

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

* ) Not available in base-line microcontrollers

(continued)
### Instruction Set

#### Complete Instruction Set Summary  (continued)

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clock</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 \leftarrow PC + k + 1 )</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>IJMP</td>
<td></td>
<td>Indirect Jump to (Z)</td>
<td>( PC \leftarrow Z )</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>JMP</td>
<td>k</td>
<td>Jump</td>
<td>( PC \leftarrow k )</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>RCALL</td>
<td>k</td>
<td>Relative Call Subroutine</td>
<td>( PC \leftarrow PC + k + 1 )</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>ICALL</td>
<td></td>
<td>Indirect Call to (Z)</td>
<td>( PC \leftarrow Z )</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>CALL</td>
<td>k</td>
<td>Call Subroutine</td>
<td>( PC \leftarrow k )</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>RET</td>
<td></td>
<td>Subroutine Return</td>
<td>( PC \leftarrow \text{STACK} )</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>RETI</td>
<td></td>
<td>Interrupt Return</td>
<td>( PC \leftarrow \text{STACK} )</td>
<td>None</td>
<td>4</td>
</tr>
<tr>
<td>CPSE</td>
<td>Rd,Rr</td>
<td>Compare, Skip if Equal</td>
<td>if ((Rd = Rr)) then (PC \leftarrow PC + 2 ) or 3 (PC \leftarrow PC + 2 ) or 3</td>
<td>None</td>
<td>1 / 2 / 3</td>
</tr>
<tr>
<td>SBRC</td>
<td>Rr, b</td>
<td>Skip if Bit in Register Cleared</td>
<td>if ((Rr(b)=0)) then (PC \leftarrow PC + 2 ) or 3 (PC \leftarrow 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)) then (PC \leftarrow PC + 2 ) or 3 (PC \leftarrow PC + 2 ) or 3</td>
<td>None</td>
<td>1 / 2 / 3</td>
</tr>
<tr>
<td>SBIC</td>
<td>P, b</td>
<td>Skip if Bit in I/O Register Cleared</td>
<td>if((I/O(P,b)=0)) then (PC \leftarrow PC + 2 ) or 3 (PC \leftarrow PC + 2 ) or 3</td>
<td>None</td>
<td>1 / 2 / 3</td>
</tr>
<tr>
<td>SBIS</td>
<td>P, b</td>
<td>Skip if Bit in I/O Register Set</td>
<td>if((I/O(P,b)=1)) then (PC \leftarrow PC + 2 ) or 3 (PC \leftarrow 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 \leftarrow PC+) if ((SREG(s) = 0)) then (PC \leftarrow PC+)</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) = 1)) then (PC \leftarrow PC+) if ((SREG(s) = 0)) then (PC \leftarrow PC+)</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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \oplus V = 0)) then (PC \leftarrow PC + k + 1) (PC \leftarrow 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 \oplus V = 1)) then (PC \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow 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 \leftarrow PC + k + 1) (PC \leftarrow PC + k + 1)</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRTC</td>
<td>k</td>
<td>Branch if T Flag Cleared</td>
<td>if ((T = 0)) then (PC \leftarrow PC + k + 1) (PC \leftarrow PC + k + 1)</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRVS</td>
<td>k</td>
<td>Branch if Overflow Flag is Set</td>
<td>if ((V = 1)) then (PC \leftarrow PC + k + 1) (PC \leftarrow PC + k + 1)</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRVC</td>
<td>k</td>
<td>Branch if Overflow Flag is Cleared</td>
<td>if ((V = 0)) then (PC \leftarrow PC + k + 1) (PC \leftarrow PC + k + 1)</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRIE</td>
<td>k</td>
<td>Branch if Interrupt Enabled</td>
<td>if ((I = 1)) then (PC \leftarrow PC + k + 1) (PC \leftarrow PC + k + 1)</td>
<td>None</td>
<td>1/2</td>
</tr>
<tr>
<td>BRID</td>
<td>k</td>
<td>Branch if Interrupt Disabled</td>
<td>if ((I = 0)) then (PC \leftarrow PC + k + 1) (PC \leftarrow PC + k + 1)</td>
<td>None</td>
<td>1/2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Operands</th>
<th>Description</th>
<th>Operation</th>
<th>Flags</th>
<th>#Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>Rd, Rr</td>
<td>Copy Register</td>
<td>Rd ← Rr</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>LDS</td>
<td>Rd, K</td>
<td>Load Immediate</td>
<td>Rd ← K</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>LDS</td>
<td>Rd, k</td>
<td>Load Direct from SRAM</td>
<td>Rd ← (k)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, X</td>
<td>Load Indirect</td>
<td>Rd ← (X)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, X+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd ← (X), X ← X + 1</td>
<td>None</td>
<td>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</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Y</td>
<td>Load Indirect</td>
<td>Rd ← (Y)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Y+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd ← (Y), Y ← Y + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, -Y</td>
<td>Load Indirect and Pre-Decrement</td>
<td>Y ← Y - 1, Rd ← (Y)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LDD</td>
<td>Rd, Y+q</td>
<td>Load Indirect with Displacement</td>
<td>Rd ← (Y + q)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Z</td>
<td>Load Indirect</td>
<td>Rd ← (Z)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, Z+</td>
<td>Load Indirect and Post-Increment</td>
<td>Rd ← (Z), Z ← Z+1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>Rd, -Z</td>
<td>Load Indirect and Pre-Decrement</td>
<td>Z ← Z - 1, Rd ← (Z)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>LDD</td>
<td>Rd, Z+q</td>
<td>Load Indirect with Displacement</td>
<td>Rd ← (Z + q)</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>STS</td>
<td>k, Rr</td>
<td>Store Direct to SRAM</td>
<td>Rd ← (k)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>ST</td>
<td>X, Rr</td>
<td>Store Indirect</td>
<td>(X) ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>X+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(X) ← Rr, X ← X + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>-X, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>X ← X - 1, (X) ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Y, Rr</td>
<td>Store Indirect</td>
<td>(Y) ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Y+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(Y) ← Rr, Y ← Y + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>-Y, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>Y ← Y - 1, (Y) ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>STD</td>
<td>Y+q, Rr</td>
<td>Store Indirect with Displacement</td>
<td>(Y + q) ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Z, Rr</td>
<td>Store Indirect</td>
<td>(Z) ← Rr</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>Z+, Rr</td>
<td>Store Indirect and Post-Increment</td>
<td>(Z) ← Rr, Z ← Z + 1</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>ST</td>
<td>-Z, Rr</td>
<td>Store Indirect and Pre-Decrement</td>
<td>Z ← 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></td>
<td>Load Program Memory</td>
<td>R0 ← (Z)</td>
<td>None</td>
<td>3</td>
</tr>
<tr>
<td>IN</td>
<td>Rd, P</td>
<td>In Port</td>
<td>Rd ← P</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>OUT</td>
<td>P, Rr</td>
<td>Out Port</td>
<td>P ← 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>

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

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

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

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

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

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

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

**16 bit Opcode:**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>l</td>
<td>rd</td>
<td>dddd</td>
</tr>
</tbody>
</table>

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

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
<td>⇔</td>
</tr>
</tbody>
</table>

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

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

**V:** \( Rd7 \& Rr7 + Rd7 \& R7 \& R7 \)
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:** \( Rd7 \& Rr7 \& R7 \& R7 \& R7 \& Rd7 \)
Set if the result is $00$; cleared otherwise.

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

R (Result) equals Rd after the operation.

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

**Words:** 1 (2 bytes)

**Cycles:** 1
ADD - Add without Carry

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

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

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

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

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

**16 bit Opcode:**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>l1rd</td>
<td>dddd</td>
<td>rrrr</td>
<td></td>
</tr>
</tbody>
</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:** \(Rd3 \cdot Rr3 \cdot R3 + Rr3 + R3 \cdot Rd3\)
Set if there was a carry from bit 3; cleared otherwise

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

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

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

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

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

**R (Result) equals Rd after the operation.**

**Example:**

```
add r1,r2 ; Add r2 to r1 (r1 = r1+r2)
add r28,r28 ; Add r28 to itself (r28 = r28+r28)
```

**Words:** 1 (2 bytes)

**Cycles:** 1
**ADIW - Add Immediate to Word**

Description:

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

**Operation:**

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

**Syntax:**

(i) \( \text{ADIW Rdl,K} \)

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

**Program Counter:**

\( \text{PC} \leftarrow \text{PC} + 1 \)

**16 bit Opcode:**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0110</td>
<td>KKdd</td>
<td>KKKK</td>
</tr>
</tbody>
</table>

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

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

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

V: \( \text{Rdh7 R15} \)

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: \( \text{R15 \times R14 \times R13 \times R12 \times R11 \times R10 \times R9 \times R8 \times R7} \)

Set if the result is \( \text{0000} \); cleared otherwise.

C: \( \text{R15 \times Rdh7} \)

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

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

**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) \( R_d \leftarrow R_d \land R_r \)

Syntax: Operands: Program Counter:
(i) \( \text{AND } R_d,R_r \) \( 0 \leq d \leq 31, 0 \leq r \leq 31 \) \( \text{PC} \leftarrow \text{PC} + 1 \)

16 bit Opcode:

```
0010 00rd dddd rrrr
```

Status Register (SREG) and Boolean Formulae:

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

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

V: 0
Cleared

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

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

R (Result) equals Rd after the operation.

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

Words: 1 (2 bytes)
Cycles: 1
ANDI - Logical AND with Immediate

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

Operation:
(i) Rd ← Rd • K

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

16 bit Opcode:

<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>

Status Register (SREG) and Boolean Formulae:

S: N ⊕ V, For signed tests.

V: 0
Cleared

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

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

R (Result) equals Rd after the operation.

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

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

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

**Operation:**

(i)  
\[
\begin{array}{cccc}
\text{b7} & \text{b6} & \cdots & \text{b0} \quad \text{C}
\end{array}
\]

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

**16 bit Opcode:**

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

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

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

**S:** N ⊕ V, For signed tests.

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

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

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

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

**Example:**

```plaintext
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) \( \text{SREG}(s) \leftarrow 0 \)

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

16 bit Opcode:

| 1001 | 0100 | 1sss | 1000 |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
</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:
\[
\text{bclr} \ 0 \quad ; \text{Clear carry flag}
\text{bclr} \ 7 \quad ; \text{Disable interrupts}
\]

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

16 bit Opcode:

\[
\begin{array}{c|c|c|c}
1111 & 100d & dddd & 0bbb \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

Example:
\[
\text{bst r1,2} \quad ; \text{Store bit 2 of r1 in T flag}
\text{bld r0,4} \quad ; \text{Load T flag into bit 4 of r0}
\]

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

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

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

16 bit Opcode:

```
1111 01kk kkkk ksss
```

Status Register (SREG) and Boolean Formulae:

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

Example:
```
cpi r20,5 ; Compare r20 to the value 5
brbc 1,noteq ; Branch if zero flag cleared
...
noteq:nop ; Branch destination (do nothing)
```

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

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

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

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

16 bit Opcode:

| 1111 | 00kk | kkkk | ksss |

Status Register (SREG) and Boolean Formulae:

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

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

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

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

Operation:

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

Syntax: Operands: Program Counter:
(i) BRCC 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>0lkk</td>
<td>kkkk</td>
<td>k000</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:

```
addr22,r23 ; Add r23 to r22
brccnocarry ; Branch if carry cleared
...
ncarry: nop ; Branch destination (do nothing)
```

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

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

**Operation:**
(i) If \( C = 1 \) then \( PC \leftarrow PC + k + 1 \), else \( PC \leftarrow PC + 1 \)

**Syntax:**
(i) BRCS k

**Operands:**
-64 ≤ k ≤ +63

**Program Counter:**
- \( PC \leftarrow PC + k + 1 \)
- \( PC \leftarrow PC + 1 \), 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 Formulae:**

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

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

Syntax: 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:

```
<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:
```
cpr1,r0 ; Compare registers r1 and r0
breqequal ; Branch if registers equal
...
equal: nop ; Branch destination (do nothing)
```

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

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

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

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

PC ← PC + k + 1
PC ← PC + 1, if condition is false

16 bit Opcode:

|   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:
cpr11,r12 ; Compare registers r11 and r12
brgegreateq ; Branch if r11 >= r12 (signed)
...
greateq: nop ; Branch destination (do nothing)

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

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

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

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

PC ← PC + 1, if condition is false

16 bit Opcode:

```
1111 01kk kkkk k101
```

Status Register (SREG) and Boolean Formulae:

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

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

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

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

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

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

16 bit Opcode:

```
1111 00kk kkkk k101
```

Status Register (SREG) and Boolean 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:
```
brhshset ; Branch if half carry flag set
...

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

Words: 1 (2 bytes)

Cycles: 1 if condition is false
        2 if condition is true
BRID - Branch if Global Interrupt is Disabled

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

Operation:

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

Syntax: 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 Formulae:

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

Example:
```
brid intdis ; Branch if interrupt disabled
...
intdis:   nop ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true
BRIE - Branch if Global Interrupt is Enabled

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

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

Syntax:
(i) BRIE k -64 ≤ k ≤ +63

Operands:

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

16 bit Opcode:

Status Register (SREG) and Boolean Formulae:

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true
BRLO - Branch if Lower (Unsigned)

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

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

Syntax:
(i) BRLO k -64 ≤ k ≤ +63

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

16 bit Opcode:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>00kk</td>
</tr>
<tr>
<td>kkkk</td>
<td>k000</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:

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true
BRLT - Branch if Less Than (Signed)

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

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

**Syntax:**
(i) BRLT k

**Operands:**
-64 ≤ k ≤ +63

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

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

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

PC ← PC + 1, if condition is false

16 bit Opcode:

| 1111 | 00kk | kkkk | k010 |

Status Register (SREG) and Boolean Formulae:

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

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

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

16 bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>0lkk</td>
<td>kkkk</td>
<td>k001</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:

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 \((PC-64 \leq \text{destination} \leq PC+63)\). The parameter \(k\) is the offset from PC and is represented in two’s complement form. (Equivalent to instruction BRBC 2,\(k\)).

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

Syntax: Operands: Program Counter:
(i) BRPL \(k\) \(-64 \leq k \leq +63\) \(PC \rightarrow PC + k + 1\)
\(PC \rightarrow PC + 1\), if condition is false

16 bit Opcode:

```
1111 01kk kkkk k010
```

Status Register (SREG) and Boolean Formulae:

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

Example:

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

Words: 1 (2 bytes)
Cycles: 1 if condition is false
        2 if condition is true
BRSH - Branch if Same or Higher (Unsigned)

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

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

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

16 bit Opcode:

<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>k000</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:
subi r19,4 ; Subtract 4 from r19
brsh highsm ; Branch if r19 >= 4 (unsigned)
... 
highsm: nop ; Branch destination (do nothing)

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

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

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

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

16 bit Opcode:

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

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

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

16 bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>k</td>
<td>k</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
brts tset ; Branch if this bit was set
...
tset: nop ; Branch destination (do nothing)

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true
BRVC - Branch if Overflow Cleared

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

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

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

16 bit Opcode:

1111 01kk kkkk k011

Status Register (SREG) and Boolean Formulae:

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

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

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

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

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

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

PC ← PC + 1, if condition is false

16 bit Opcode:

```
1111 00kk kkkk k011
```

Status Register (SREG) and Boolean Formulae:

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

Example:
```
add r3,r4 ; Add r4 to r3
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) $\text{BSET } s  \quad \text{0} \leq s \leq 7 \quad \text{PC} \leftarrow \text{PC} + 1$

16 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

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

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

Example:

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

Words: 1 (2 bytes)
Cycles: 1
BST - Bit Store from Bit in Register to T Flag in SREG

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

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

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

16 bit Opcode:

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

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>$\leftrightarrow$</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).

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

Syntax: Operands: Program Counter:Stack
(i) \( \text{CALL} \ k \quad 0 \leq k \leq 64K \quad \text{PC} \leftarrow k \text{STACK} \leftarrow \text{PC}+2 \quad \text{SP} \leftarrow \text{SP}-2, \text{2 bytes, 16 bits} \)
(ii) \( \text{CALL} \ k \quad 0 \leq k \leq 4M \quad \text{PC} \leftarrow k\text{STACK} \leftarrow \text{PC}+2 \quad \text{SP} \leftarrow \text{SP}-3 \text{ (3 bytes, 22 bits)} \)

32 bit Opcode:

<table>
<thead>
<tr>
<th>1001</th>
<th>010k</th>
<th>kkkk</th>
<th>111k</th>
</tr>
</thead>
<tbody>
<tr>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td>kkkk</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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

Example:

```
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
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) \( I/O(P,b) \leftarrow 0 \)

Syntax: Operands: Program Counter:
(i) CBI P,b \( 0 \leq P \leq 31, 0 \leq b \leq 7 \) \( PC \leftarrow PC + 1 \)

16 bit Opcode:

```
1001 1000 pppp pbbb
```

Status Register (SREG) and Boolean Formulae:

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

Example:
```
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) \( Rd \leftarrow Rd \cdot (\$FF - K) \)

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

16 bit Opcode: See ANDI with K complemented.

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>( \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: \( R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \)
Set if the result is $00$; cleared otherwise.

R (Result) equals Rd after the operation.

Example:
\[
\begin{align*}
cbr & \ r16, \$F0 \quad ; \text{Clear upper nibble of r16} \\
cbr & \ r18, 1 \quad ; \text{Clear bit 0 in r18}
\end{align*}
\]

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

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

Operation:
(i) \[ C \leftarrow 0 \]

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

16 bit Opcode:

| 1001 | 0100 | 1000 | 1000 |

Status Register (SREG) and Boolean Formulae:

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

C: 0
Carry flag cleared

Example:
```
add r0,r0 ; Add r0 to itself
cic ; 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: CLH
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>1101</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>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

H: 0
Half Carry flag cleared

Example:

```
clh ; Clear the Half Carry flag
```

Words: 1 (2 bytes)
Cycles: 1
CLI - Clear Global Interrupt Flag

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

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>Operands:</th>
<th>Program Counter:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>None</td>
<td>PC ← PC + 1</td>
</tr>
</tbody>
</table>

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

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 0100 & 1111 & 1000 \\
\end{array}
\]

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

16 bit Opcode:

| 1001 | 0100 | 1010 | 1000 |

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>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:
C 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: Operands: Program Counter:
(i) CLR Rd \( 0 \leq d \leq 31 \) PC \( \leftarrow \) PC + 1

16 bit Opcode: (see EOR Rd,Rd)

| 0010 | 01dd | dddd | dddd |

Status Register (SREG) and Boolean Formulae:

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

S: 0
Cleared

V: 0
Cleared

N: 0
Cleared

Z: 1
Set

R (Result) equals Rd after the operation.

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

Words: 1 (2 bytes)
Cycles: 1
CLS - Clear Signed Flag

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

Operation:
\[(i) \quad S \leftarrow 0\]

Syntax: \(\text{CLS}\)  
Operands: None  
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>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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

\(S: 0\)
Signed flag cleared

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

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

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

Operation:
(i) $T \leftarrow 0$

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

16 bit Opcode:

<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 Formulae:

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

T: 0
T flag cleared

Example:

```
clt ; Clear T flag
```

Words: 1 (2 bytes)
Cycles: 1
CLV - Clear Overflow Flag

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

Operation:
(i) V ← 0

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

16 bit Opcode:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0100</td>
<td>1011</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>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 Formulae:

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

Z: 0
Zero flag cleared

Example:
```
add r2,r3 ; Add r3 to r2
ciz ; 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:

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>010d</th>
<th>dddd</th>
<th>0000</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>0</td>
<td>⇔</td>
<td>⇔</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 \& R6 \& R5 \& R4 \& R3 \& R2 \& R1 \& R0
Set if the result is \( \$00 \); Cleared otherwise.

C: 1
Set.

R (Result) equals Rd after the operation.

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

Words: 1 (2 bytes)
Cycles: 1
CP - Compare

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

Operation:
(i) Rd - Rr

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

16 bit Opcode:

```
0001  01rd  dddd  rrrr
```

Status Register (SREG) and Boolean Formulae:

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

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

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

V: \( Rd7 \oplus R7 \oplus \overline{Rd7} \oplus \overline{R7} \oplus R7 \oplus R7 \)
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: \( Rd7 \oplus Rr7 \oplus R7 \oplus R7 \oplus \overline{Rd7} \oplus \overline{R7} \oplus R7 \)
Set if the result is $00; cleared otherwise.

C: \( Rd7 \oplus Rr7 \oplus R7 \oplus R7 \oplus \overline{Rd7} \oplus \overline{R7} \oplus R7 \)
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: Operands: Program Counter:
(i) CPC Rd,Rr 0 ≤ d ≤ 31, 0 ≤ r ≤ 31 PC ← PC + 1

16 bit Opcode:

```
0000 01rd dddd rrrr
```

Status Register (SREG) and Boolean Formulae:

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

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

S: N + V, For signed tests.

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

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

Z: \(R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0 \cdot Z\)
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 Rd; cleared otherwise.

R (Result) after the operation.

Example:
```
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 \( 16 \leq d \leq 31, 0 \leq K \leq 255 \) PC \( \leftarrow PC + 1 \)

16 bit Opcode:

```
0011  KKKK  dddd  KKKK
```

Status Register (SREG) and Boolean Formulae:

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

H: \( R_{d3} \cdot K_{3}+ R_{3} \cdot R_{3} \cdot R_{d3} \)
Set if there was a borrow from bit 3; cleared otherwise

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

V: \( R_{d7} \cdot K_{7} \cdot R_{7} \cdot R_{d7} + K_{7} \cdot R_{7} \)
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: \( R_{7} \cdot R_{5} \cdot R_{5} \cdot R_{3} \cdot R_{2} \cdot R_{1} \cdot R_{0} \)
Set if the result is $00$; cleared otherwise.

C: \( R_{d7} \cdot K_{7} + K_{7} \cdot R_{7} + R_{7} \cdot R_{d7} \)
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

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

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

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

Operation:
(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 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:

\[
\begin{array}{cccc}
0001 & 00rd & dddd & rrrr \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

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

Words: 1 (2 bytes)
Cycles: 1
**DEC - Decrement**

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

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

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

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

**Syntax:**
(i) DEC Rd

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

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

**16 bit Opcode:**

```
1001 010d dddd 1010
```

**Status Register and Boolean Formulae:**

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

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

V: \( R7 \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0 \)
Set if two’s complement overflow resulted from the operation; cleared otherwise. Two’s complement overflow occurs if and only if \( Rd \) was $80 before the operation.

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

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

R (Result) equals Rd after the operation.

**Example:**

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

**Words:** 1 (2 bytes)

**Cycles:** 1
EOR - Exclusive OR

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

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

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

16 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

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

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

V: 0
Cleared

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

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

R (Result) equals Rd after the operation.

Example:

\[
eor \ \ r4,r4 \quad ; \text{Clear \ r4} \\
eor \ \ r0,r22 \quad ; \text{Bitwise exclusive or between \ r0 \ and \ r22}
\]

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

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

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

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

16 bit Opcode:

| 1001 | 0101 | XXXX | 1001 |

Status Register (SREG) and Boolean Formulae:

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

Example:

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

Words: 1 (2 bytes)
Cycles: 3
IJMP - Indirect Jump

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

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

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

16 bit Opcode:

| 1001 | 0100 | XXXX | 1001 |

Status Register (SREG) and Boolean Formulae:

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

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

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

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

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

Syntax:
(i) \( \text{IN Rd,P} \)

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

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

16 bit Opcode:

| 1011 | 0PPd | dddd | PPPP |

Status Register (SREG) and Boolean Formulae:

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

Example:
```
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)
```

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: Operands: Program Counter:
(i) INC Rd \( 0 \leq d \leq 31 \) PC \( \leftarrow \) 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>1001</td>
<td>010d</td>
<td>dddd</td>
<td>0011</td>
<td></td>
</tr>
</tbody>
</table>

Status Register and Boolean Formulae:

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

S: \( N \oplus V \)

For signed tests.

V: \( R7 \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0 \)
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: \( R7 \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0 \)
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) \( \text{PC} \leftarrow k \)

Syntax: Operands: Program Counter: Stack

(i) JMP k \( 0 \leq k \leq 4M \) \( \text{PC} \leftarrow k \) Unchanged

32 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

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

Example:

\[
\begin{align*}
\text{mov} & \quad r1,r0 \quad ; \text{Copy r0 to r1} \\
\text{jmp} & \quad \text{farplc} \quad ; \text{Unconditional jump} \\
\ldots & \quad \text{farplc: nop} \quad ; \text{Jump destination (do nothing)}
\end{align*}
\]

Words: 2 (4 bytes)
Cycles: 3
LD - Load Indirect from SRAM to Register using Index X

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

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

Using the X pointer:

<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 ← X + 1</td>
</tr>
<tr>
<td>(iii) X ← X - 1</td>
<td>Rd ← (X)</td>
</tr>
</tbody>
</table>

Syntax: Operands: Program Counter:

| (i) LD Rd, X | 0 ≤ d ≤ 31 | PC ← PC + 1 |
| (ii) LD Rd, X+ | 0 ≤ d ≤ 31 | PC ← PC + 1 |
| (iii) LD Rd,-X | 0 ≤ d ≤ 31 | PC ← PC + 1 |

16 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

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

Example:

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

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

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

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

Using the Y pointer:

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

Syntax: Operands: Program Counter:

(i) LD Rd, Y 0 ≤ d ≤ 31 PC ← PC + 1
(ii) LD Rd, Y+ 0 ≤ d ≤ 31 PC ← PC + 1
(iii) LD Rd,-Y 0 ≤ d ≤ 31 PC ← PC + 1
(iv) LDD Rd, Y+q 0 ≤ d ≤ 31, 0 ≤ q ≤ 63 PC ← PC + 1

16 bit Opcode:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1000</td>
<td>000d</td>
<td>dddd</td>
<td>1000</td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>1001</td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>000d</td>
<td>dddd</td>
<td>1010</td>
</tr>
<tr>
<td>(iv)</td>
<td>10q0</td>
<td>qq0d</td>
<td>dddd</td>
<td>1qqq</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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

Example:

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

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

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

The Z pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are especially suited for stack pointer usage of the Z pointer register, however because the Z pointer register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y pointer as a dedicated stack pointer.

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

Using the Z pointer:

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

Syntax:

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

16 bit Opcode:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Operands</th>
<th>Program Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1000 000d dddd 0000</td>
<td>0 ≤ d ≤ 31</td>
<td>PC ← PC + 1</td>
</tr>
<tr>
<td>(ii)</td>
<td>1001 000d dddd 0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iii)</td>
<td>1001 000d dddd 0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iii)</td>
<td>10q0 qq0d dddd 0qqq</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:

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

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

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

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

Syntax: \( \text{LDI Rd,K} \)
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 Formulae:

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

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

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

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

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

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

Status Register (SREG) and Boolean Formulae:

Example:

- \( \text{lds r2,$FF00} \); Load r2 with the contents of SRAM location $FF00
- \( \text{add r2,r1} \); add r1 to r2
- \( \text{sts $FF00,r2} \); Write back

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

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

Operation:  
(i) \( R0 \leftarrow (Z) \)  

Comment:  
Z points to program memory

Syntax:  
(i) LPM

Operands:  
None

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

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 0101 & 110X & 1000 \\
\end{array}
\]

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>

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

Words: 1 (2 bytes)

Cycles: 3
LSL - Logical Shift Left

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

Operation:

(i)

\[ 
C \leftarrow \begin{array}{c}
\cdots \\
\downarrow \\
0
\end{array}
\]

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

16 bit Opcode: (see ADD Rd,Rd)

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & d & d & d & d & d
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

H: \( \text{Rd3} \)
S: \( \text{N} \oplus \text{V} \), For signed tests.
V: \( \text{N} \oplus \text{C} \) (For N and C after the shift)
Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).
N: \( \text{R7} \)
Set if MSB of the result is set; cleared otherwise.
Z: \( \text{R7} \bullet \text{R6} \bullet \text{R5} \bullet \text{R4} \bullet \text{R3} \bullet \text{R2} \bullet \text{R1} \bullet \text{R0} \)
Set if the result is $00$; cleared otherwise.
C: \( \text{Rd7} \)
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

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

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:

\[
\begin{align*}
0 \rightarrow & \begin{array}{c}
\overline{b7} - - - - - - - - - - - - - - - - - - b0 \\
\rightarrow & C
\end{array}
\end{align*}
\]

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

16 bit Opcode:

\[
\begin{array}{cccccc}
1001 & 010d & dddd & 0110
\end{array}
\]

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

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

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

N: 0

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

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

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) Rd ← Rr

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

16 bit Opcode:

```
0010 11rd dddd rrrr
```

Status Register (SREG) and Boolean Formulae:

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

Example:

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

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

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

\[
\begin{array}{c|c|c}
\text{Rr} & \times & \text{Rd} \\
\text{Multiplicand} & 8 & \text{Multiplier} & 8 & \rightarrow \\
\text{R1} & \text{Product High} & \text{Product Low} & 16
\end{array}
\]

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

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

Syntax: Operands: Program Counter:
(i) MUL Rd,Rr \( 0 \leq d \leq 31, \ 0 \leq r \leq 31 \)

PC \( \leftarrow PC + 1 \)

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 11rd & dddd & rrrr
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

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

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

Example:
mulr6,r5; Multiply r6 and r5
movr6,r1; Copy result back in r6:r5
movr5,r0; Copy result back in r6:r5

Words: \( 1 \) (2 bytes)
Cycles: \( 2 \)

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

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

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

Syntax:
(i) NEG Rd

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

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

16 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

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

H: \( R3 \cdot \text{Rd3} \)
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:         ; 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 Formulae:

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

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
OR - Logical OR

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

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

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

16 bit Opcode:

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

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

R (Result) equals Rd after the operation.

Example:

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

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

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

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

Syntax:
(i) ORI Rd,K

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

Program Counter:
PC \leftarrow PC + 1

16 bit Opcode:

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

\[
\begin{align*}
\text{ori} & \quad \text{r16}, \$00 \quad ; \text{Set high nibble of r16} \\
\text{ori} & \quad \text{r17}, 1 \quad ; \text{Set bit 0 of r17}
\end{align*}
\]

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

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

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

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

16 bit Opcode:

| 1011 | lPPr | rrrr | PPPP |

Status Register (SREG) and Boolean Formulae:

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

Example:

```
cir r16 ; Clear r16
sel r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Wait (do nothing)
out $18,r17 ; Write ones to Port B
```

Words: 1 (2 bytes)

Cycles: 1
POP - Pop Register from Stack

Description:
This instruction loads register Rd with a byte from the STACK.

Operation:
(i) Rd ← STACK

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

16 bit Opcode:

```
1001  000d  dddd  1111
```

Status Register (SREG) and Boolean Formulae:

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

Example:
```
call routine ; Call subroutine
...
routine: push r14 ; Save r14 on the stack
push r13 ; Save r13 on the stack
...
pop r13 ; Restore r13
pop r14 ; Restore r14
ret ; Return from subroutine
```

Words: 1 (2 bytes)
Cycles: 2
PUSH - Push Register on Stack

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

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

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

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 001d & dddd & 1111 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

Example:

```
call routine ; Call subroutine
...
routine:
push r14 ; Save r14 on the stack
push r13 ; Save r13 on the stack
...  
pop r13 ; Restore r13
pop r14 ; Restore r14
ret ; Return from subroutine
```

Words: 1 (2 bytes)
Cycles: 2
RCALL - Relative Call to Subroutine

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

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

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

Operands:
(i) \(k\) Program Counter: \(PC \leftarrow PC + k + 1\) Stack: \(STACK \leftarrow PC+1\) \(SP \leftarrow SP-2\) (2 bytes, 16 bits)
(ii) \(k\) Program Counter: \(PC \leftarrow PC + k + 1\) Stack: \(STACK \leftarrow PC+1\) \(SP \leftarrow SP-3\) (3 bytes, 22 bits)

16 bit Opcode:

```
1101 kkkk kkkk kkkk
```

Status Register (SREG) and Boolean Formulae:

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

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
RET - Return from Subroutine

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

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

Syntax: Operands: Program Counter: Stack
(i) RET None See Operation SP ← SP+2,(2 bytes,16 bits pulled)
(ii) RET None See Operation SP ← SP+3,(3 bytes,22 bits pulled)

16 bit Opcode:

```
1001 0101 0XX0 1000
```

Status Register (SREG) and Boolean Formulae:

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

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

Words: 1 (2 bytes)
Cycles: 4
# RETI - Return from Interrupt

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

**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:**
SP \( \leftarrow \) SP +2 (2 bytes, 16 bits)
(i) SP \( \leftarrow \) SP +3 (3 bytes, 22 bits)

**16 bit Opcode:**

```
1001 0101 0XX1 1000
```

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

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

**I:**
1
The I flag is set.

**Example:**

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

**Words:**
1 (2 bytes)

**Cycles:**
4
RJMP - Relative Jump

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

Operation:
(i) \( \text{PC} \leftarrow \text{PC} + k + 1 \)

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

16 bit Opcode:

| 1100 | kkkk | kkkk | kkkk |

Status Register (SREG) and Boolean Formulae:

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

Example:

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

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

Description:
Shifts all bits in Rd one place to the left. The C flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C flag.

Operation:

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

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

16 bit Opcode: (see ADC Rd,Rd)

\[
\begin{array}{cccc}
0001 & 11dd & dddd & dddd
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

H: Rd3

S: N \oplus V, For signed tests.

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

N: 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: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

\[
\begin{array}{l}
\text{rolr15} \quad ; \text{Rotate left} \\
\text{brcsoneenc} \quad ; \text{Branch if carry set} \\
\ldots \\
\text{oneenc: nop} \quad ; \text{Branch destination (do nothing)}
\end{array}
\]

Words: 1 (2 bytes)

Cycles: 1
ROR - Rotate Right through Carry

Description:
Shifts all bits in Rd one place to the right. The C flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C flag.

Operation:

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

Syntax: 
ROR Rd

Operands: 
0 ≤ d ≤ 31

Program Counter: 
PC ← PC + 1

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 010d & dddd & 0111 \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

S: 
N ⊕ V, For signed tests.

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

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

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

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

R (Result) equals Rd after the operation.

Example:

rorr15 ; Rotate right
brcczeroenc ; Branch if carry cleared
...
zeroenc: nop ; Branch destination (do nothing)

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

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

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

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

16 bit Opcode:

```
0000 10rd dddd rrrr
```

Status Register and Boolean Formulae:

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

H: \( \overline{Rd7} \cdot Rr3 + Rr3 \cdot R3 + R3 \cdot \overline{Rd3} \)
Set if there was a borrow from bit 3; cleared otherwise.

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

V: \( Rr7 \cdot \overline{R7} \cdot R7 + \overline{Rd7} \cdot \overline{R7} \cdot R7 \)
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: \( \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0} \cdot Z \)
Previous value remains unchanged when the result is zero; cleared otherwise.

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

R (Result) equals Rd after the operation.

Example:

```
sub  r2,r0  ; Subtract 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) \( \text{Rd} \leftarrow \text{Rd} - K - C \)

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

16 bit Opcode:

\[
\begin{array}{cccc}
0100 & \text{KKKK} & \text{dddd} & \text{KKKK}
\end{array}
\]

Status Register and Boolean Formulae:

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

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

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

V: \( \text{Rd}7 \cdot K7 \cdot \text{R7} + \text{Rd}7 \cdot K7 \cdot \text{R7} \)
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: \( \text{R7} \cdot \text{R6} \cdot \text{R5} \cdot \text{R4} \cdot \text{R3} \cdot \text{R2} \cdot \text{R1} \cdot \text{R0} \cdot \text{Z} \)
Previous value remains unchanged when the result is zero; cleared otherwise.

C: \( \text{Rd}7 \cdot K7 + K7 \cdot \text{R7} + R7 \cdot \text{Rd}7 \)
Set if the absolute value of the constant plus previous carry is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

; Subtract $4F23 from r17:r16
subi r16,$23 ; Subtract low byte
sbc i r17,$4F ; Subtract with carry high byte

Words: 1 (2 bytes)
Cycles: 1
SBI - Set Bit in I/O Register

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

\[
\text{Operation:}
\]

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

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

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 1010 & \text{pppp} & \text{pbbb}
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

Example:

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

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

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

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

Syntax:
(i) SBIC P,b 0 ≤ P ≤ 31, 0 ≤ b ≤ 7

Program Counter:
PC ← PC + 1, If condition is false, no skip.
PC ← PC + 2, If next instruction is one word.
PC ← PC + 3, If next instruction is JMP or CALL

16 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

Example:

```
e2wait: sbic $1C,1 ; Skip next inst. if EEWE cleared
         jmp 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)
SBIS - Skip if Bit in I/O Register is Set

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

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

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

16 bit Opcode:

```
1001 1011 pppp pbbb
```

Status Register (SREG) and Boolean Formulae:

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

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)
SBIW - Subtract Immediate from Word

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

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

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

16 bit Opcode:

```
1001 0111 KKdd KKKK
```

Status Register (SREG) and Boolean Formulae:

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

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

V: \( \text{Rdh7} \oplus \text{R15} \)
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: \( \text{RT5} \oplus \text{RT4} \oplus \text{RT3} \oplus \text{RT2} \oplus \text{RT1} \oplus \text{RT0} \oplus \text{R9} \oplus \text{R8} \oplus \text{R7} \oplus \text{R6} \oplus \text{R5} \oplus \text{R4} \oplus \text{R3} \oplus \text{R2} \oplus \text{R1} \oplus \text{R0} \)
Set if the result is 0000; cleared otherwise.

C: \( \text{R15} \oplus \text{Rdh7} \)
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

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

Example:
```
sbiw r24,1 ; Subtract 1 from r25:r24
sbiw r28,63 ; 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) \( Rd \leftarrow Rd \lor K \)

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

16 bit Opcode:

\[
\begin{array}{cccc}
0110 & KKKK & dddd & KKKK \\
\end{array}
\]

Status Register (SREG) and Boolean Formulae:

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

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

V: 0
Cleared

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

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

R (Result) equals Rd after the operation.

Example:

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

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:  Operands:  Program Counter:
(i) SBRC $Rr,b$  $0 \leq r \leq 31$, $0 \leq b \leq 7$  

16 bit Opcode:

1111 110r  rrrr  Xbbb

Status Register (SREG) and Boolean Formulae:

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

Example:

```assembly
sub r0,r1 ; Subtract r1 from r0
sbrc r0,7 ; Skip if bit 7 in r0 cleared
sub r0,r1 ; Only executed if bit 7 in r0 not cleared
nop ; Continue (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false (no skip)
2 if condition is true (skip is executed)
SBRS - Skip if Bit in Register is Set

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

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

Syntax: Operands: Program Counter:
(i) SBRS Rr,b 0 ≤ r ≤ 31, 0 ≤ b ≤ 7 PC ← PC + 1, Condition false - no skip

PC ← PC + 2, Skip a one word instruction
PC ← PC + 3, Skip a JMP or a CALL

16 bit Opcode:

```
1111 lllr rrrr Xbbb
```

Status Register (SREG) and Boolean Formulae:

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

Example:
```
sub r0,r1 ; Subtract r1 from r0
sbrs r0,7 ; Skip if bit 7 in r0 set
neg r0 ; Only executed if bit 7 in r0 not set
nop ; Continue (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false (no skip)
2 if condition is true (skip is executed)
SEC - Set Carry Flag

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

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

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

16 bit Opcode:

```
1001 0100 0000 1000
```

Status Register (SREG) and Boolean Formulae:

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

```
1001 0100 0101 1000
ITH S VN Z C
- - 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>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 Formulae:

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

I: 1 Global Interrupt flag set

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

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

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

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

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

16 bit Opcode:

```
1001 0100 0010 1000
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>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) \( \text{Rd} \leftarrow \$FF \)

**Syntax:**
(i) \text{SER Rd}

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

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

**16 bit Opcode:**

```
1110 1111 dddd 1111
```

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

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

**Example:**
clr r16 ; Clear r16
ers r17 ; Set r17
out $18,r16 ; Write zeros to Port B
nop ; Delay (do nothing)
out $18,r17 ; Write ones to Port B

**Words:** 1 (2 bytes)

**Cycles:** 1
SES - Set Signed Flag

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

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

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

16 bit Opcode:

```
1001 0100 0100 1000
```

Status Register (SREG) and Boolean Formulae:

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>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: Operands: Program Counter:
(i) SET None PC \( \leftarrow \) PC + 1

16 bit Opcode:
\[
\begin{array}{cccc}
1001 & 0100 & 0110 & 1000 \\
\end{array}
\]

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

T: \( 1 \)
T flag set

Example:
\[
\text{set } \quad ; \text{ Set T flag}
\]

Words: 1 (2 bytes)
Cycles: 1
SEV - Set Overflow Flag

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

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

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

16 bit Opcode:

```
1001 0100 0011 1000
```  

Status Register (SREG) and Boolean Formulae:

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

```
1001 0100 0001 1000
```

Status Register (SREG) and Boolean Formulae:

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

Z: 1
Zero flag set

Example:

```
add r2, r19 ; Add r19 to r2
sez ; Set zero flag
```

Words: 1 (2 bytes)
Cycles: 1
SLEEP

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

Operation:

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

16 bit Opcode:

```
1001 0101 100X 1000
```

Status Register (SREG) and Boolean Formulae:

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

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

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

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

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

Using the X pointer:

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

Syntax: Operands:

(i) ST X, Rr
(ii) ST X+, Rr
(iii) ST -X, Rr

Program Counter:

(i) PC ← PC + 1
(ii) PC ← PC + 1
(iii) PC ← PC + 1

16 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

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

Example:

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

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

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

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

Using the Y pointer:

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

Syntax: Operands: Program Counter:

| (i) ST \(Y, Rr\) | \(0 \leq r \leq 31\) | PC \(\leftarrow\) PC + 1 |
| (ii) ST \(Y +, Rr\) | \(0 \leq r \leq 31\) | PC \(\leftarrow\) PC + 1 |
| (iii) ST \(-Y, Rr\) | \(0 \leq r \leq 31\) | PC \(\leftarrow\) PC + 1 |
| (iv) STD \(Y + q, Rr\) | \(0 \leq r \leq 31, 0 \leq q \leq 63\) | PC \(\leftarrow\) PC + 1 |

16 bit Opcode:

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>001r</th>
<th>rrrr</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1000</td>
<td>001r</td>
<td>rrrr</td>
<td>1000</td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1001</td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>001r</td>
<td>rrrr</td>
<td>1010</td>
</tr>
<tr>
<td>(iv)</td>
<td>10r0</td>
<td>qq1r</td>
<td>rrrr</td>
<td>1qqq</td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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

Example:

```asm
clr r29 ; Clear Y high byte
ldi r28,020 ; Set Y low byte to 020
st Y+,r0 ; Store r0 in SRAM loc. 020(Y post inc)
st Y,r1 ; Store r1 in SRAM loc. 021
ldi r28,023 ; Set Y low byte to 023
st Y,r2 ; Store r2 in SRAM loc. 023
st -Y,r3 ; Store r3 in SRAM loc. 022(Y pre dec)
std Y+2,r4 ; Store r4 in SRAM loc. 024
```

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

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

The Z pointer register can either be left unchanged after the operation, or it can be incremented or decremented. These features are very suited for stack pointer usage of the Z pointer register, but because the Z pointer register can be used for indirect subroutine calls, indirect jumps and table lookup it is often more convenient to use the X or Y pointer as a dedicated stack pointer.

Using the Z pointer:

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

Syntax: Operands: Program Counter:

| (i) ST Z, Rr | 0 ≤ r ≤ 31 |
| (ii) ST Z+, Rr | 0 ≤ r ≤ 31 |
| (iii) ST -Z, Rr | 0 ≤ r ≤ 31 |
| (iv) STD Z+q, Rr | 0 ≤ r ≤ 31, 0 ≤ q ≤ 63 |

16 bit Opcode:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>1000</td>
<td>001</td>
<td>rrrr</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>(ii)</td>
<td>1001</td>
<td>001</td>
<td>rrrr</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>(iii)</td>
<td>1001</td>
<td>001</td>
<td>rrrr</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>(iv)</td>
<td>10q0</td>
<td>q1r</td>
<td>rrrr</td>
<td>0qqq</td>
<td></td>
</tr>
</tbody>
</table>

Status Register (SREG) and Boolean Formulae:

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

Example:

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

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

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

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

Syntax: \( \text{STS } k,Rr \)
Operands: \( 0 \leq r \leq 31, 0 \leq k \leq 65535 \)
Program Counter:
\( \text{PC} \leftarrow \text{PC} + 2 \)

32 bit Opcode:

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

Status Register (SREG) and Boolean Formulae:

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

Example:

- \( \text{lds } r2,$FF00 \); Load r2 with the contents of SRAM location $FF00
- \( \text{add } r2,r1 \); add r1 to r2
- \( \text{sts } $FF00,r2 \); Write back

Words: 2 (4 bytes)
Cycles: 3
SUB - Subtract without Carry

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

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

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

16 bit Opcode:

\[
\begin{array}{cccc}
0001 & 10 & d & d & r & r & r & r
\end{array}
\]

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

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

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

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

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

C:  \( \text{Rd}7 \cdot \text{Rr}7 \cdot \text{Rr}7 \cdot \text{R7} \cdot \text{R7} \cdot \text{Rd}7 \)
Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
sub     r13,r12 ; Subtract r12 from r13
brne    noteq ; Branch if r12<>r13
...
noteq:  nop ; Branch destination (do nothing)
```

Words:  1 (2 bytes)
Cycles:  1
SUBI - Subtract Immediate

Description:
Subtracts a register and a constant and places the result in the destination register Rd. This instruction is working on Register R16 to R31 and is very well suited for operations on the X, Y and Z pointers.

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

Syntax: Operands: Program Counter:
(i) SUBI Rd,K \( 16 \leq d \leq 31, \ 0 \leq K \leq 255 \) 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>0101</td>
<td>KKKK</td>
<td>dddd</td>
<td>KKKK</td>
</tr>
</tbody>
</table>

Status Register and Boolean Formulae:

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

H: \( R_3 \oplus K_3 + R_3 \oplus R_3 \)
Set if there was a borrow from bit 3; cleared otherwise

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

V: \( R_7 \oplus K_7 + R_7 \oplus R_7 \)
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: \( R_7 \oplus R_6 \oplus R_5 \oplus R_4 \oplus R_3 \oplus R_2 \oplus R_1 \oplus R_0 \)
Set if the result is $00; cleared otherwise.

C: \( R_7 \oplus K_7 + R_7 \oplus R_7 \)
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```assembly
subir22,$11 ; Subtract $11 from r22
brnenoteq ; Branch if r22<>$11
...
noteq:   nop ; Branch destination (do nothing)
```

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

Description:
Swaps high and low nibbles in a register.

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

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

16 bit Opcode:

\[
\begin{array}{cccc}
1001 & 010d & dddd & 0010 \\
\end{array}
\]

Status Register and Boolean Formulae:

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

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

Example:

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

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

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

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

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

16 bit Opcode:

```
0010 00dd dddd dddd
```

Status Register and Boolean Formulae:

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

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

V: 0
Cleared

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

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

R (Result) equals Rd.

Example:
```
tst r0 ; Test r0
breq zero ; Branch if r0=0
...
zero: nop ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1
WDR - Watchdog Reset

Description:
This instruction resets the Watchdog Timer. This instruction must be executed within a limited time given by the WD prescaler. See the Watchdog Timer hardware specification.

Operation:
(i) WD timer restart.

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

16 bit Opcode:

```
1001 0101 101X 1000
```

Status Register and Boolean Formulae:

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

Example:

```
wdr ; Reset watchdog timer
```

Words: 1 (2 bytes)
Cycles: 1