M3000

MOTOR AND MOTION CONTROLLER

INSTRUCTION SET

SYSTEM SEMICONDUCTOR inc.™

Patent Pending

M3000 Instruction Set Manual Change Log		
Date	Revision	Changes
08/07/2007	R080707	Initial Release

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1: ADDRESSING MODES

The Program and Data Addressing Modes

The M3000 Enhanced RISC microcontroller supports powerful and efficient addressing modes for access to the program memory and data memory (SRAM, Register file andI/O Memory). This section describes the different addressing modes supported by the M3000 architecture. In the following figures, OP means the operation code part of the instruction word. To simplify, not all figures show the exact location of the addressing bits.

Register Direct, Single Register Rd

The operand is contained in register Rd.

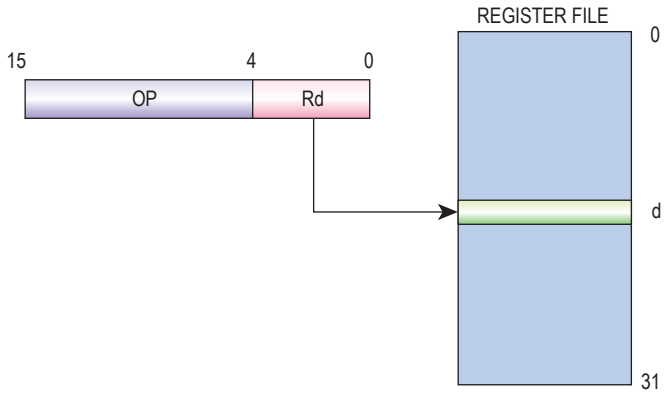


Figure 1.1: Direct Register Addressing, Single Register

Register Direct, Two Registers Rd and Rr

Operands are contained in register Rr and Rd. The result is stored in register Rd.

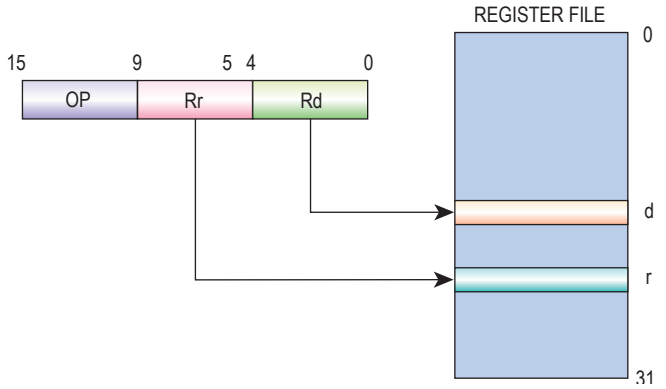


Figure 1.2: Direct Register Addressing, Two Registers

I/O Direct

The operand address is contained in 6 bits of the instruction word. Rr/Rd is the destination or source register address.

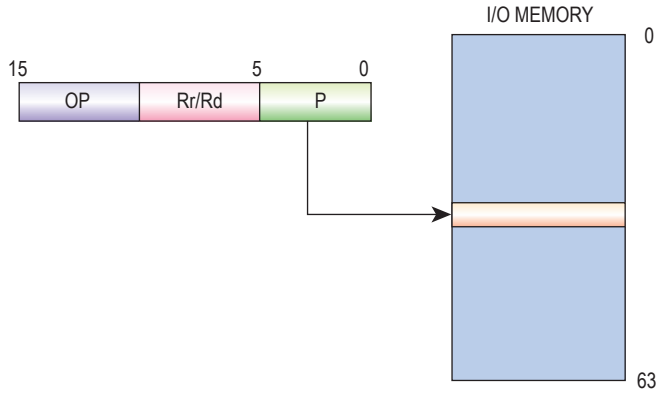


Figure 1.3: I/O Direct Addressing

Data Direct

A 16-bit Data Address is contained in the 16 LSBs of a two-word instruction. Rd/Rr specify the destination or source register.

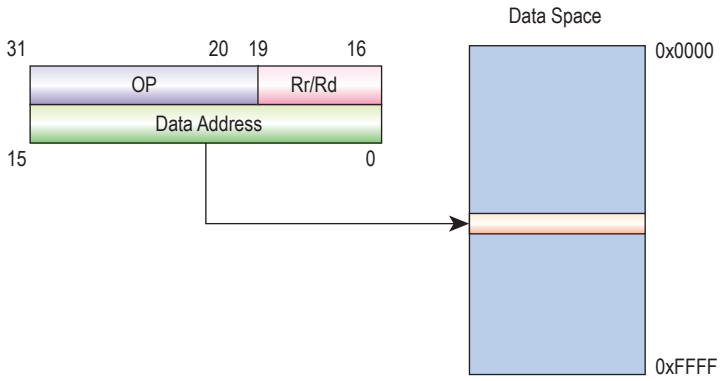


Figure 1.4: Direct Data Addressing

Data Indirect with Displacement

The operand address is the result of the Y or Z-register contents added to the address contained in 6 bits of the instruction word. Rr/Rd specify the destination or source register.

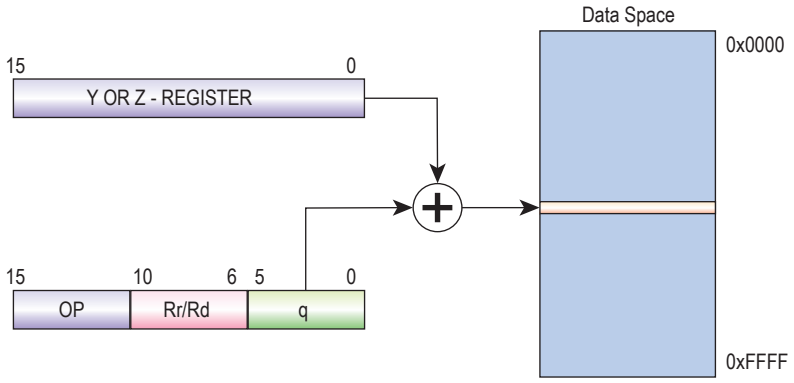


Figure 1. 5. Data Indirect with Displacement

Data Indirect Addressing

The operand address is the contents of the X, Y or the Z-register.

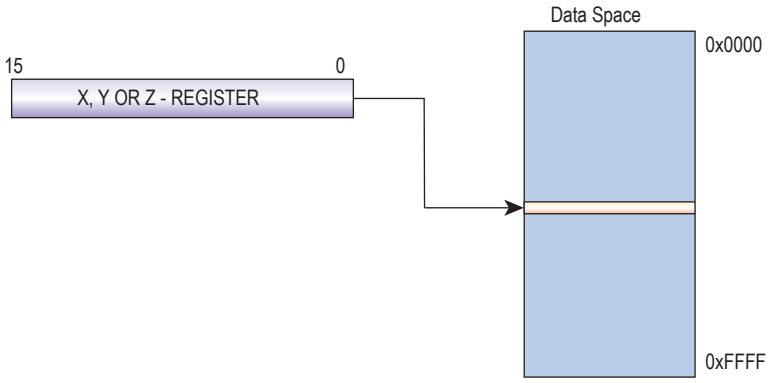


Figure 1. 6: Data Indirect Addressing

Data Indirect with Pre-decrement

The X, Y or the Z-register is decremented before the operation. The operand address is the decremented contents of the X,Y or the Z-register.

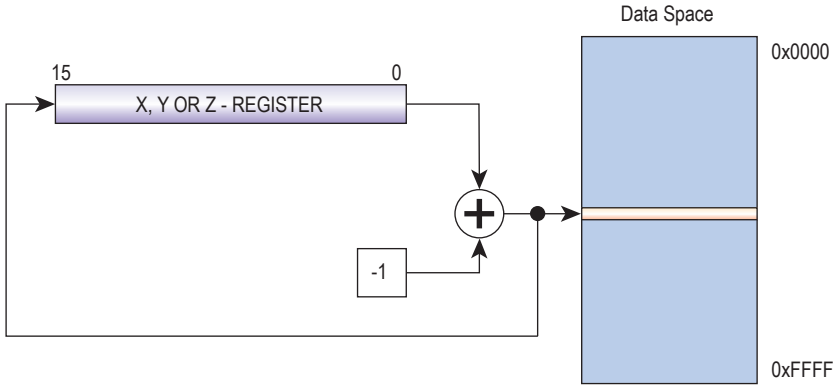


Figure 1.7: Data Indirect Addressing with Pre-decrement

Data Indirect with Post-increment

The X, Y or the Z-register is incremented after the operation. The operand address is the content of the X, Y or the Z-register prior to incrementing.

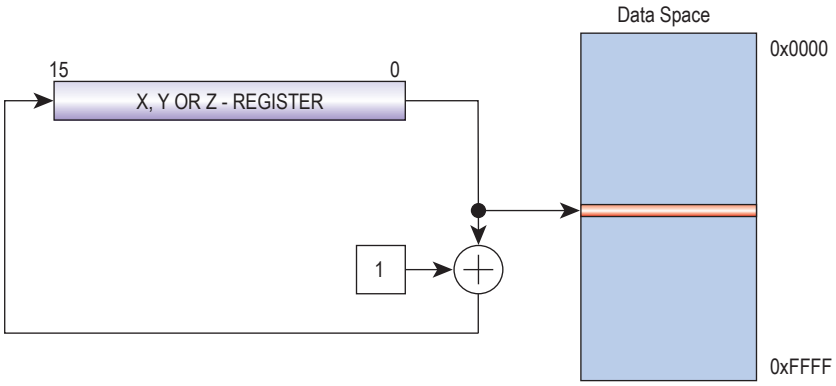


Figure 1.8: Data Indirect Addressing with Post-increment

Constant Addressing Using the LPM, ELPM, and SPM Instructions

Constant byte address is specified by the Z-register contents. The 15 MSBs select word address. For LPM, the LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). For SPM, the LSB should be cleared. If ELPM is used, the RAMPZ Register is used to extend the Z-register.

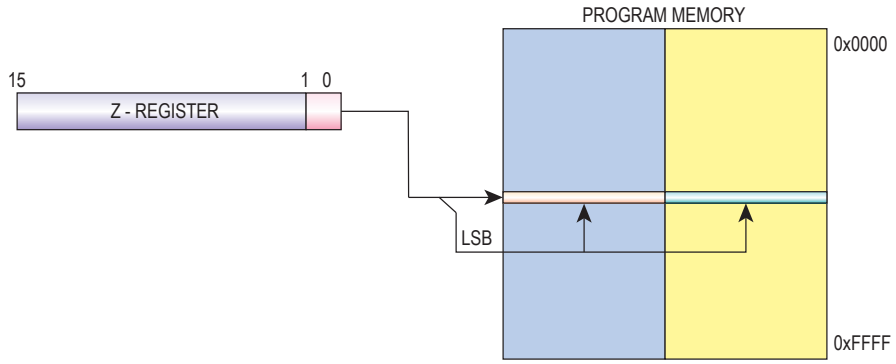


Figure 1.9: Program Memory Constant Addressing

Program Memory with Post-increment using the LPM Z+ and ELPM Z+ Instruction

Constant byte address is specified by the Z-register contents. The 15 MSBs select word address. The LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). If ELPM Z+ is used, the RAMPZ Register is used to extend the Z-register.

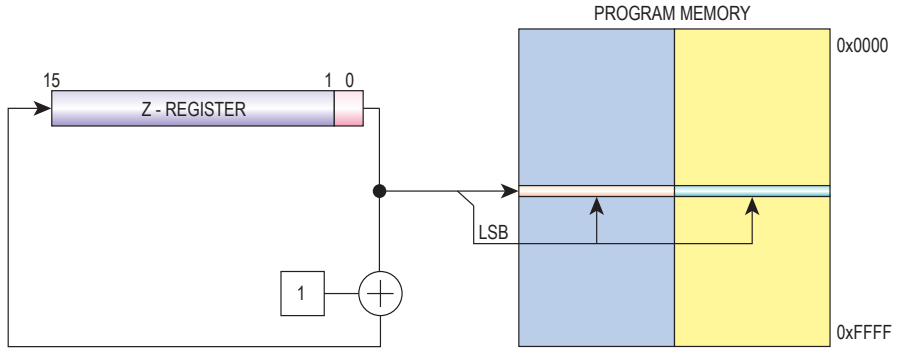


Figure 1.10: Program Memory Addressing with Post-increment

Direct Program Addressing, JMP and CALL

Program execution continues at the address immediate in the instruction words.

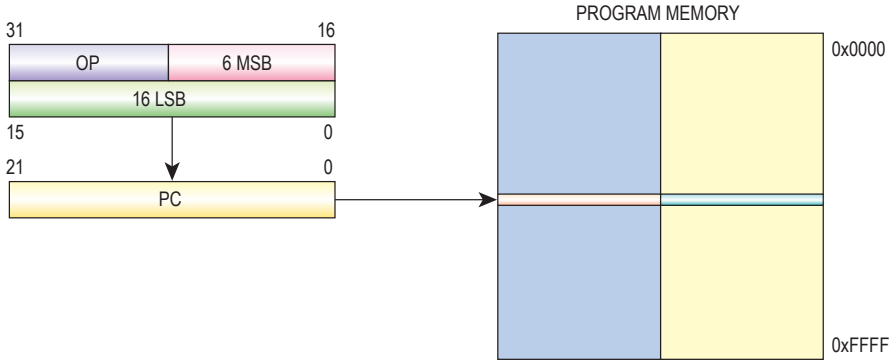


Figure 1.11: Direct Program Memory Addressing

Indirect Program Addressing, IJMP and ICALL

Program execution continues at address contained by the Z-register (i.e., the pc is loaded with the contents of the Z-register).

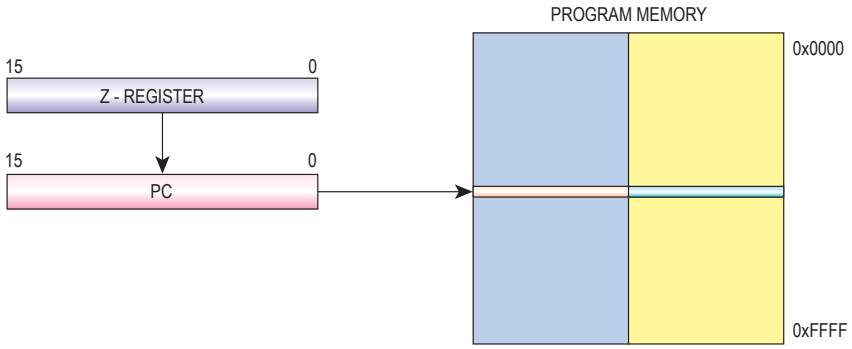


Figure 1.12: Indirect Program Memory Addressing

Relative Program Addressing, RJMP and RCALL

Program execution continues at address pc + k + 1. The relative address k is -2048 to 2047.

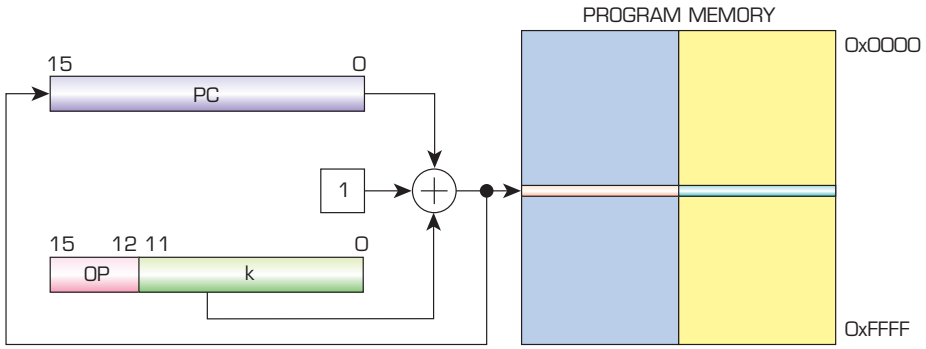


Figure 1.13: Relative Program Memory Addressing

SECTION 2:
INSTRUCTION SET NOMENCLATURE

Status Register	**SREG:**	Status register
	C:	Carry flag
	Z:	Zero flag
	N:	Negative flag
	V:	Two's complement overflow indicator
	S:	N ⊕ V, for signed tests
	H:	Half Carry flag
	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, Z=R31:R30)
	P:	I/O port address
	q:	Displacement for direct addressing (6-bit)
I/O Registers	**RAMPZ:**	Register concatenated with the Z register enabling indirect addressing of the whole Program Area on MCUs with more than 64K bytes of Program Code (ELPM instruction)
Stack	**STACK:**	Stack for return address and pushed registers
	SP:	Stack Pointer to STACK
Flags	**⇔:**	Flag affected by instruction
	0:	Flag cleared by instruction
	1:	Flag set by instruction
	-:	Flag not affected by instruction

SECTION 3:
COMPLETE INSTRUCTION SET SUMMARY

Arithmetic and Logic Instructions

	Arithmetic and Logic Instructions					
	Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**# Clock Note**
ADD	**ADD**	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1
	ADC	Rd, Rr	Add With Carry	Rd ← Rd +Rr + C	Z,C,N,V,S,H	1
	ADIW	Rd, K	Add Immediate to Word	Rd + 1:Rd ← Rd + 1:Rd + K	Z,C,N,V,S	2
SUBTRACT	**SUB**	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1
	SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1
	SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
	SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1
	SBIW	Rd, K	Subtract Immediate from Word	Rd + 1:Rd ← Rd + 1:Rd - K	Z,C,N,V,S	2
AND	**AND**	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V,S	1
	ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V,S	1
OR	**OR**	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1
	ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1
	EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1
MISCELLANEOUS	**COM**	Rd	One's Complement	Rd ← 0xFF - Rd	Z,C,N,V,S	1
	NEG	Rd	Two's Complement	Rd ← 0x00 - Rd	Z,C,N,V,S,H	1
	SBR	Rd, K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1
	CBR	Rd, K	Clear Bit(s) in Register	Rd ← Rd • [0xFF - K]	Z,N,V,S	1
	INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1
	DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1
	TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V,S	1
	CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1
	SER	Rd	Set Register	Rd ← 0xFF	None	1
MULTIPLY	**MUL**	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr [UU]	Z,C	2
	MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr [SS]	Z,C	2
	MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr [SU]	Z,C	2
	FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← [Rd x Rr]<<1 [UU]	Z,C	2
	FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← Rd x Rr<<1 [SS]	Z,C	2
	FMULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr<<1 [SU]	Z,C	2

Table 3.1: Arithmetic and Logic Instructions

MCU Control Instructions

MCU Control Instructions					
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**# Clock Note**
BREAK	—	Break	(See Specific Description for BREAK)	None	1
NOP	—	No Operation		None	1
SLEEP	—	Sleep	(See Specific Description for SLEEP)	None	1
WDR	—	Watchdog Reset	(See Specific Description for WDR)	None	1

Table 3.2: MCU Control Instructions

Branch Instructions

NOTE

[1] Note: Cycle times for data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface. For LD, ST, LDS, STS, PUSH, POP, add one cycle plus one cycle for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI, add five cycles plus three cycles for each wait state.

	Branch Instructions					
	Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**# Clock Note**
JUMP	**RJMP**	k	Relative Jump	PC ← PC + k + 1	None	2
	IJMP		Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2
	EIJMP		Extended Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	2
	JMP	k	Jump	PC ← k	None	2
CALL	**RCALL**	k	Relative Call Subroutine	PC ← PC + k + 1	None	3/4[1]
	ICALL		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	3/4[1]
	EICALL		Extended Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	4[1]
	CALL	k	Call Subroutine	PC ← k	None	4/5[1]
	RET		Subroutine Return	PC ← STACK	None	4/5[1]
	RETI		Interrupt Return	PC ← STACK	I	4/5[1]
COMPARE	**CPSE**	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
	CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,S,H	1
	CPC	Rd,Rr	Compare with Carry	Rd -Rr - C	Z,C,N,V,S,H	1
	CPI	Rd,k	Compare with Immediate	Rd - K	Z,C,N,V,S,H	1
SKIP	**SBRC**	Rd,b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
	SBRS	Rd,b	Skip if Bit in Register Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
	SBIC	A, b	Skip if Bit in I/O Register Cleared	if(I/O(A,b)=0) PC ← PC + 2 or 3	None	1/2/3
	SBIS	A, b	Skip if Bit in I/O Register Set	If(I/O(A,b)=1) PC ← PC + 2 or 3	None	1/2/3
BRANCH	**BRBS**	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC ← PC+k + 1	None	1/2
	BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC ← PC+k + 1	None	1/2
	BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
	BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
	BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
	BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
	BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
	BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
	BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
	BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
	BRGE	k	Branch if Greater or Equal, Signed	if (N ∨ V= 0) then PC ← PC + k + 1	None	1/2
	BRLT	k	Branch if Less Than, Signed	if (N ∨ V= 1) then PC ← PC + k + 1	None	1/2
	BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
	BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
	BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
	BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
	BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
	BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
	BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
	BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2

Table 3.3: Branch Instructions

Data Transfer Instructions

NOTE

(1) Note: Cycle times for data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface. For LD, ST, LDS, STS, PUSH, POP, add one cycle plus one cycle for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI, add five cycles plus three cycles for each wait state.

	Data Transfer Instructions					
	Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**# Clock Note**
COPY	**MOV**	Rd, Rr	Copy Register	Rd ← Rr	None	1
	MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1
LOAD DATA	**LDI**	Rd, K	Load Immediate	Rd ← K	None	1
	LDS	Rd, k	Load Direct from data space	Rd ← (k)	None	2(1)
	LD	Rd, X	Load Indirect	Rd ← (X)	None	2(1)
	LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X), X ← X + 1	None	2(1)
	LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X -1, Rd ← (X)	None	2(1)
	LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2(1)
	LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2(1)
	LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y -1, Rd ← (Y)	None	2(1)
	LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2(1)
	LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2(1)
	LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2(1)
	LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z -1, Rd ← (Z)	None	2(1)
	LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2(1)
STORE	**STS**	k, Rr	Store Direct to data space	(k) ← Rd	None	2(1)
	ST	X, Rr	Store Indirect	(X) ← Rr	None	2(1)
	ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2(1)
	ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X -1, (X) ← Rr	None	2(1)
	ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2(1)
	ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2(1)
	ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y -1, (Y) ← Rr	None	2(1)
	STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2(1)
	ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2(1)
	ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr, Z ← Z + 1	None	2(1)
	ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z -1, (Z) ← Rr	None	2(1)
	STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2(1)
LOAD PGM	**LPM**		Load Program Memory	R0 ← (Z)	None	3
	LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
	LPM	Rd, Z+	Load Program Memory and Post Increment	Rd ← (Z), Z ← Z + 1	None	3
	ELPM		Increment	R0 ← (RAMPZ:Z)	None	3
	ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z) ← Z + 1	None	3
	ELPM	Rd, Z+	Extended Load Program Memory and Post Increment	Rd ← (RAMPZ:Z)	None	3
MISC.	**SPM**		Store Program Memory	(Z) ← R1:R0	None	—
	IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1
	OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1
	PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
	POP	Rd	Pop Register From Stack	Rd ← STACK	None	2

Table 3.4: Data Transfer Instructions

Bit and Bit-Test Instructions

NOTE

(1) Note: Cycle times for data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface. For LD, ST, LDS, STS, PUSH, POP, add one cycle plus one cycle for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI in devices with 16-bit PC, add three cycles plus two cycles for each wait state. For CALL, ICALL, EICALL, RCALL, RET, RETI in devices with 22-bit PC, add five cycles plus three cycles for each wait state.

Bit and Bit-Test Instructions					
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**# Clock Note**
LSL	Rd	Logical Shift Left	Rd(n+1)←Rd(n),Rd(0)←0,C←Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n)←Rd(n+1),Rd(7)←0,C←Rd(0)	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)←Rd(n),C←Rd(7)	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)←Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ↔ Rd(7..4)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	2
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	2
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Two's Complement Overflow	V ← 1	V	1
CLV		Clear Two's Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1

Table 3.5: Bit and Bit-Test Instructions

Conditional Branch Summary

NOTE

Note: Interchange Rd and Rr in the operation before the test (i.e., CP Rd,Rr → CP Rr, Rd)

Conditional Branch Summary						
Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	**BRLT(1)**	Rd ≤ Rr	Z+(N ⊕ V) = 1	**BRGE***	Signed
Rd ≥ Rr	(N ⊕ V) = 0	**BRGE**	Rd < Rr	(N ⊕ V) = 1	**BRLT**	Signed
Rd = Rr	Z = 1	**BREQ**	Rd ≠ Rr	Z = 0	**BRNE**	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	**BRGE(1)**	Rd > Rr	Z•(N ⊕ V) = 0	**BRLT***	Signed
Rd < Rr	(N ⊕ V) = 1	**BRLT**	Rd ≥ Rr	(N ⊕ V) = 0	**BRGE**	Signed
Rd > Rr	C + Z = 0	**BRLO(1)**	Rd ≤ Rr	C + Z = 1	**BRLO/BRCS**	Unsigned
Rd ≥ Rr	C = 0	**BRSH/BRCC**	Rd < Rr	C = 1	**BRCS**	Unsigned
Rd = Rr	Z = 1	**BREQ**	Rd ≠ Rr	Z = 0	**BRNE**	Unsigned
Rd ≤ Rr	C + Z = 1	**BRSH(1)**	Rd > Rr	C + Z = 0	**BRLO***	Unsigned
Rd < Rr	C = 1	**BRLO/BRCS**	Rd ≥ Rr	C = 0	**BRSH/BRCC**	Unsigned
Carry	C = 1	**BRCS**	No carry	C = 0	**BRCC**	Simple
Negative	N = 1	**BRMI**	Positive	N = 0	**BRPL**	Simple
Overflow	V = 1	**BRVS**	No overflow	V = 0	**BRVC**	Simple
Zero	Z = 1	**BREQ**	Not zero	Z = 0	**BRNE**	Simple

Table 3.6: Conditional Branch Summary

SECTION 4:
INSTRUCTION SET DETAILS

ADC – Add with Carry

Mnemonic:	Function:		
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:	Rd ← Rd + Rr + C	**Syntax:**	ADC Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0001	11rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
	Set if there was a carry from bit 3; cleared otherwise.
S	$N \oplus V$, For signed tests.
V	$Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise.
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{Rd7} \bullet \overline{Rr6} \bullet \overline{Rr5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{Rd2} \bullet \overline{Rd1} \bullet \overline{Rd0}$
	Set if the result is 0x00; cleared otherwise.
C	$Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
	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

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

ADC – Add without Carry

Mnemonic:	Function:		
ADD	Add Without Carry		
Description:			
Adds two registers without the C flag and places the result in the destination register Rd.			
Operation:	Rd ← Rd + Rr	**Syntax:**	ADD Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0001	11rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
	Set if there was a carry from bit 3; cleared otherwise.
S	$N \oplus V$, For signed tests.
V	$Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise.
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise.
C	$Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
	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)
adc r28,r28 ; add r28 to itself (r28=r28+r28)
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

ADIW – Add Immediate to Word

Mnemonic:	Function:		
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:	Rd+1:Rd ← Rd+1:Rd + K	**Syntax:**	ADIW Rd+l:K
Operand:	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0110	KKdd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	⇔

S	N ⊕ V, For signed tests.
V	Rdh7 • R15
	Set if two's complement overflow resulted from the operation; cleared otherwise.
N	R15
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise.
C	$\overline{R15}$ • Rdh7
	Set if there was carry from the MSB of the result; cleared otherwise.

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

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

AND – Logical AND

Mnemonic:	Function:		
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:	Rd ← Rd • Rr	**Syntax:**	AND Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0010	00rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	N ⊕ V, For signed tests.
V	0
	Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
and  r2,r3     ; Bitwise and r2 and r3, result in r26
ldi r16,1      ; Set bitmask 0000 0001 in r16
and r2,r16  ; Isolate bit 0 in r2
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

ANDI – Logical AND with Immediate

Mnemonic:	Function:		
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:	Rd ← Rd • K	**Syntax:**	ANDI Rd,K
Operand:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0111	KKKK	dddd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	N ⊕ V, For signed tests.
V	0
	Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
andi        r17,0x0F ; Clear upper nibble of r17
andi        r18,0x10 ; Isolate bit 4 in r18
andi        r19,0xAA ; Clear odd bits of r19
```

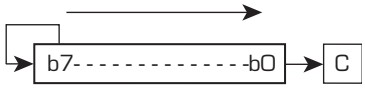
Words:	1 (2 Bytes)	Cycles:	1

ASR – Arithmetic Shift Right

Mnemonic:	Function:
ASR	Arithmetic Shift Right

Description:

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



Operation:	Rd(n) ← Rd(n+1), n=0..6	**Syntax:**	ASR Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	010d	dddd	0101

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	⇔

S	N ⊕ V, For signed tests.
V	N ⊕ C (For N and C after the shift)
	Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift).
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; 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:

```
ldi r16,0x10        ; Load decimal 16 into r16
asr r16             ; r16=r16 / 2
ldi r17,0xFC        ; Load -4 in r17
asr r17             ; r17=r17/2
```

Words:	1 (2 Bytes)	**Cycles:**	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

BCLR – Bit Clear in SREG

Mnemonic:	Function:		
BCLR	Bit Clear in SREG		
Description:			
Clears a single flag in SREG.			
Operation:	SREG(s) ← 0	**Syntax:**	BCLR s
Operand:	0 ≤ s ≤ 7	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1sss	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

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

Example:

```
bclr 0          ; Clear carry flag
bclr 7          ; Disable interrup
```

Words:	1 (2 Bytes)	Cycles:	1

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

Mnemonic:	Function:		
BLD	Bit Load from the T Flag in SREG to a Bit in Register		
Description:			
Copies the T flag in the SREG (status register) to bit b in register Rd.			
Operation:	Rd(b) ← T	**Syntax:**	BLD Rd,b
Operand:	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1111	100d	1sss	0b0b

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

BRBC – Branch if Bit in SREG is Cleared

Mnemonic:	Function:		
BRBC	Branch if Bit in SREG is Cleared		
Description:			
Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form.			
Operation:	If SREG(s) = 0 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRBC s,k
Operand:	0 ≤ s ≤ 7, -64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition false

16-Bit Opcode:

1111	01kk	kkkk	ksss

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRBS – Branch if Bit in SREG is Set

Mnemonic:	Function:		
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:	If SREG(s) = 1 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRBS s,k
Operand:	0 ≤ s ≤ 7, -64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	ksss

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

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

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

BRCC – Branch if Carry Cleared

Mnemonic:	Function:		
BRCC	Branch if Carry is Cleared		
Description:			
Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k.)			
Operation:	If C = 0 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRCC k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRCS – Branch if Carry Set

Mnemonic:	Function:		
BRCS	Branch if Carry is Set		
Description:			
Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 0,k.)			
Operation:	If C = 1 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRCS k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
cpi  r26,0x56  ; Compare r26 with 0x56
brcs carry  ; Branch if carry set
    ...
carry: nop  ; Branch destination (do nothing)
```

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BREAK – Break

Mnemonic:	Function:		
BREAK	Break		
Description:			
The BREAK instruction is used by the On-chip Debug system and is normally not used in the application software. When the BREAK instruction is executed, the AVR CPU is set in the Stopped Mode. This gives the On-chip Debugger access to internal resources.			
Operation:	On-Chip Debug System Break	**Syntax:**	BREAK
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:			
1001	0101	1001	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
None
```

Words:	1 (2 Bytes)	Cycles:	1

BREQ – Branch if Equal

Mnemonic:	Function:		
BREQ	Branch if Equal		
Description:			
Conditional relative branch. Tests the Zero flag (Z) and branches relatively to PC if Z is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 1,k.)			
Operation:	If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BREQ k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:			
1111	00kk	kkkk	k001

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

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

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

BRGE – Branch if Greater or Equal (Signed)

Mnemonic:	Function:		
BRGE	Branch if Greater or Equal (Signed)		
Description:			
Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 4,k.)			
Operation:	If Rd ≥ Rr (N ⊕ V = 0) then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRGE k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k100

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRHC – Branch if Half Carry Flag is Cleared

Mnemonic:	Function:		
BRHC	Branch if Half Carry Flag is Cleared		
Description:			
Conditional relative branch. Tests the Half Carry flag (H) and branches relatively to PC if H is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 5,k.)			
Operation:	If H = 0 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRHC k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k101

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRHS – Branch if Half Carry Flag is Set

Mnemonic:	Function:		
BRHS	Branch if Half Carry Flag is Set		
Description:			
Conditional relative branch. Tests the Half Carry flag (H) and branches relatively to PC if H is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 5,k.)			
Operation:	If H = 1 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRHS k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k101

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
brhs hset      ; Branch if half carry flag set
    ...
hset: nop   ; Branch destination (do nothing)
```

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRID – Branch if Global Interrupt is Disabled

Mnemonic:	Function:		
BRID	Branch if Global Interrupt is Disabled		
Description:			
Conditional relative branch. Tests the Global Interrupt flag (I) and branches relatively to PC if I is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 7,k.)			
Operation:	If I = 0 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRID k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k111

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

BRIE – Branch if Global Interrupt is Enabled

Mnemonic:	Function:
BRIE	Branch if Global Interrupt is Enabled

Description:

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

Operation:	If I = 1 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRIE k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k111

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	1 if condition is false 2 if condition is true

BRLO – Branch if Lower (Unsigned)

Mnemonic:	Function:
BRLO	Branch if Lower (Unsigned)

Description:

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

Operation:	If Rd < Rr (C = 1) then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRLO k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
eor r19,r19              ; Clear r19
loop: inc   r19          ; Increase r19
...
cpi r19,0x10             ; Compare r19 with 0x10
brlo loop                ; Branch if r19 < 0x10 (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)

Mnemonic:	Function:		
BRLT	Branch if Less Than (Signed)		
Description:			
Conditional relative branch. Tests the Signed flag (S) and branches relatively to PC if S is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was less than the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 4,k.)			
Operation:	If Rd < Rr (N ⊕ V = 1) then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRLT k
Operand:	-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) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRMI – Branch if Minus

Mnemonic:	Function:		
BRMI	Branch if Minus		
Description:			
Conditional relative branch. Tests the Negative flag (N) and branches relatively to PC if N is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 2,k.)			
Operation:	If N = 1 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRMI k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k010

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

BRNE – Branch if Not Equal

Mnemonic:	Function:
BRNE	Branch if Not Equal

Description:

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

Operation:	If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRNE k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k001

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

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

Mnemonic:	Function:
BRPL	Branch if Plus

Description:

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

Operation:	If N = 0 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRPL k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k010

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
subi r26,0x50        ; Subtract 0x50 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)

Mnemonic:	Function:		
BRSH	Branch if Same or Higher (Unsigned)		
Description:			
Conditional relative branch. Tests the Carry flag (C) and branches relatively to PC if C is cleared. If the instruction is executed immediately after execution of any of the instructions CP, CPI, SUB or SUBI the branch will occur if and only if the unsigned binary number represented in Rd was greater than or equal to the unsigned binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k.)			
Operation:	If Rd ≥ Rr (C = 0) then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRSH k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRTC – Branch if the T Flag is Cleared

Mnemonic:	Function:		
BRTC	Branch if the T Flag is Cleared		
Description:			
Conditional relative branch. Tests the T flag and branches relatively to PC if T is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 6,k.)			
Operation:	If T = 0 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRTC k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k110

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

BRTS – Branch if the T Flag is Set

Mnemonic:	Function:		
BRTS	Branch if the T Flag is Set		
Description:			
Conditional relative branch. Tests the T flag and branches relatively to PC if T is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 6,k.)			
Operation:	If T = 1 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRTS k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k110

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRVC – Branch if Overflow Cleared

Mnemonic:	Function:		
BRVC	Branch if Overflow Cleared		
Description:			
Conditional relative branch. Tests the Overflow flag (V) and branches relatively to PC if V is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 3,k.)			
Operation:	If V = 0 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRVC k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	01kk	kkkk	k011

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1 if condition is false 2 if condition is true

BRVS – Branch if Overflow Set

Mnemonic:	Function:
BRVS	Branch if Overflow Set

Description:

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

Operation:	If V = 1 then PC ← PC + k + 1, else PC ← PC + 1	**Syntax:**	BRVS k
Operand:	-64 ≤ k ≤ +63	**Program Counter:**	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-Bit Opcode:

1111	00kk	kkkk	k011

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	1 if condition is false 2 if condition is true

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

BSET – Bit Set in SREG

Mnemonic:	Function:		
BSET	Bit Set in SREG		
Description:			
Sets a single flag or bit in SREG.			
Operation:	SREG(s) ← 1	**Syntax:**	BSET s
Operand:	0 ≤ s ≤ 7	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0sss	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

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

Mnemonic:	Function:		
BST	Bit Store from Bit in Register to T Flag in SREG		
Description:			
Stores bit b from Rd to the T flag in SREG (status register).			
Operation:	T ← Rd(b)	**Syntax:**	BST Rd,b
Operand:	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1111	101d	dddd	0bbb

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	⇔	—	—	—	—	—	—

T	0 if s = 6; Unchanged otherwise 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

Mnemonic:	Function:		
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:	PC ← k	**Syntax:**	CALL k
Operand:	0 ≤ k < 4M	**Program Counter:**	PC ← k
		Stack:	STACK ← PC+2 SP ← SP-3 (3 bytes, 22 bits)

32-Bit Opcode:

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
mov r16,r0            ; Copy r0 to r16
call check            ; Call subroutine
nop                   ; Continue (do nothing)
...
check: cpi  r16,0x42  ; 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:	5

CBI – Clear Bit in I/O Register

Mnemonic:	Function:		
CBI	Clear Bit in I/O Register		
Description:			
Clears a specified bit in an I/O register. This instruction operates on the lower 32 I/O registers – addresses 0-31.			
Operation:	I/O(P,b) ← 0	**Syntax:**	CBI P,b
Operand:	0 ≤ P ≤ 31, 0 ≤ b ≤ 7	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	1000	pppp	pbbb

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
cbi 0x12,7     ; Clear bit 7 in Port D
```

Words:	1 (2 Bytes)	Cycles:	2

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

CBR – Clear Bits in Register

Mnemonic:	Function:		
CBR	Clear Bits in Register		
Description:			
Clears the specified bits in register Rd. Performs the logical AND between the contents of register Rd and the complement of the constant mask K. The result will be placed in register Rd.			
Operation:	Rd ← Rd • (0xFF - K)	**Syntax:**	CBR Rd,K
Operand:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0111	$\overline{KKKK}$	dddd	$\overline{KKKK}$

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	N ⊕ V, For signed tests.
V	0
	Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
cbr  r16,0xF0  ; Clear upper nibble of r16
cbr r18,1   ; Clear bit 0 in r18
```

Words:	1 (2 Bytes)	**Cycles:**	1

CLC – Clear Carry Flag

Mnemonic:	Function:		
CLC	Clear Carry Flag		
Description:			
Clears the Carry flag (C) in SREG (status register).			
Operation:	C ← 0	**Syntax:**	CLC
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1000	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	0

C	0
	Carry Flag Cleared

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	1

CLH – Clear Half Carry Flag

Mnemonic:	Function:		
CLH	Clear Half Carry Flag		
Description:			
Clears the Half Carry flag (H) in SREG (status register).			
Operation:	H ← 0	**Syntax:**	CLH
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1101	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	0	—	—	—	—	—

H	0
	Half Carry Flag Cleared

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

CLI – Clear Global Interrupt Flag

Mnemonic:	Function:		
CLI	Clear Global Interrupt Flag		
Description:			
Clears the Global Interrupt flag (I) in SREG (status register).			
Operation:	I ← 0	**Syntax:**	CLI
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1111	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
0	—	—	—	—	—	—	—

H	0
	Global Interrupt Flag Cleared

Example:

```
cli                 ; Disable interrupts
in  r11,0x16        ; Read port B
sei                 ; Enable interrupts
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

CLN – Clear Negative Flag

Mnemonic:	Function:		
CLN	Clear Negative Flag		
Description:			
Clears the Negative flag (N) in SREG (status register).			
Operation:	N ← 0	**Syntax:**	CLN
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1010	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	0	—	—

N	0
	Negative Flag Cleared

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

CLR – Clear Register

Mnemonic:	Function:		
CLR	Clear Register		
Description:			
Clears a register. This instruction performs an Exclusive OR between a register and itself. This will clear all bits in the register.			
Operation:	Rd ← Rd ⊕ Rd	**Syntax:**	CLR Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

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

0010	01dd	dddd	dddd

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	0	0	0	1	—

S	0
	Cleared
V	0
	Cleared
N	0
	Cleared
Z	1
	Set

R (Result) equals Rd after the operation.

Example:

```
clr r18                ; clear r18
loop: inc r18          ; increase r18
      ...
cpi r18,0x50           ; Compare r18 to 0x50
brne loop
```

Words:	1 (2 Bytes)	Cycles:	1

CLS – Clear Signed Flag

Mnemonic:	Function:		
CLS	Clear Signed Flag		
Description:			
Clears the Signed flag (S) in SREG (status register).			
Operation:	S ← 0	**Syntax:**	CLS
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1100	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	0	—	—	—	—

S	0
	Signed Flag Cleared

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

CLT – Clear T Flag

Mnemonic:	Function:		
CLT	Clear T Flag		
Description:			
Clears the T flag in SREG (status register).			
Operation:	T ← 0	**Syntax:**	CLT
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1110	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	0	—	—	—	—	—	—

T	0
	T Flag Cleared

Example:

```
clt            ;  Clear T flag
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

CLV – Clear Overflow Flag

Mnemonic:	Function:		
CLV	Clear Overflow Flag		
Description:			
Clears the Overflow flag in SREG (status register).			
Operation:	V ← 0	**Syntax:**	CLV
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1011	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	0	—	—	—

V	0
	Overflow Flag Cleared

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

CLZ – Clear Zero Flag

Mnemonic:	Function:		
CLZ	Clear Zero Flag		
Description:			
Clears the Zero flag in SREG (status register).			
Operation:	z ← 0	**Syntax:**	CLZ
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	1001	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	0	—

Z	0
	Overflow Flag Cleared

Example:

```
add  r2,r3     ; Add r3 to r2
clz            ; Clear zero
```

Words:	1 (2 Bytes)	Cycles:	1

COM – One's Complement

Mnemonic:	Function:
COM	Clear Bits in Register

Description:

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

Operation:	Rd ← 0xFF - Rd	**Syntax:**	COM Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	010d	dddd	0000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	1

S	N ⊕ V, For signed tests.
V	0
	Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; 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

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

CP – Compare

Mnemonic:	Function:
CP	Compare

Description:

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

Operation:	Rd - Rr	**Syntax:**	CP Rd,Rr
Operand:	$0 \le d \le 31$, $0 \le r \le 31$	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0001	01rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
	Set if there was a borrow from bit 3; cleared otherwise
S	$N \oplus V$, For signed tests.
V	$Rd7 \bullet \overline{Rd7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise.
C	$\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
	Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise

R (Result) after the operation.

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	1

CPC – Compare with Carry

Mnemonic:	Function:
CPC	Compare with Carry

Description:

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

Operation:	Rd - Rr - C	**Syntax:**	CPC Rd,Rr
Operand:	$0 \le d \le 31$, $0 \le r \le 31$	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	01rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
	Set if there was a borrow from bit 3; cleared otherwise
S	$N \oplus V$, For signed tests.
V	$Rd7 \bullet \overline{Rd7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
	Previous value remains unchanged when the result is zero; cleared otherwise
C	$\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
	Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise

R (Result) after the operation.

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

CPI – Compare with Immediate

Mnemonic:	Function:		
CPI	Compare with Immediate		
Description:			
This instruction performs a compare between register Rd and a constant. The register is not changed. All conditional branches can be used after this instruction.			
Operation:	Rd - K	**Syntax:**	CPI Rd,K
Operand:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0011	KKKK	dddd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
	Set if there was a borrow from bit 3; cleared otherwise
S	$N \oplus V$, For signed tests.
V	$Rd7 \bullet \overline{Rd7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Previous value remains unchanged when the result is zero; cleared otherwise
C	$\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
	Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) after the operation.

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

CPSE – Compare Skip if Equal

Mnemonic:	Function:		
CPSE	Compare Skip if Equal		
Description:			
This instruction performs a compare between two registers Rd and Rr, and skips the next instruction if Rd = Rr.			
Operation:	If Rd = Rr then PC ← PC + 2 (or 3) else PC ← PC + 1	**Syntax:**	CPSE Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-Bit Opcode:

0001	00rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

DEC – Decrement

Mnemonic:	Function:
DEC	Decrement

Description:

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

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

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

Operation:	Rd ← Rd - 1	**Syntax:**	DEC Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	010d	dddd	1010

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	—

S	$N \oplus V$
	For signed tests.
V	$\overline{R7} \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0$
	Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was 0x80 before the operation
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise

R (Result) equals Rd after the operation.

Example:

```
        ldi r17,0x10      ; Load constant in r17
loop:   add r1,r2         ; Add r2 to r1
        dec r17           ; Decrement r17
        brne loop         ; Branch if r17<>0
        nop               ; Continue (do nothing)
```

Words:	1 (2 Bytes)	**Cycles:**	1

EICALL - Extended Indirect Call to Subroutine

Mnemonic:	Function:		
EICALL	Extended Indirect Call to a Subroutine		
Description:			
Indirect call of a subroutine pointed to by the Z (16 bits) Pointer Register in the Register File and the EIND Register in the I/O space. This instruction allows for indirect calls to the entire Program memory space. The Stack Pointer uses a post-decrement scheme during EICALL.			
Operation:	PC(15:0) ← Z(15:0) PC(21:16) ← EIND	**Syntax:**	EICALL
Operand:	None	**Program Counter:**	Stack
		Stack:	STACK ← PC + 1 SP ← SP - 3(3 bytes, 22 bits)

16-Bit Opcode:

1001	0101	0001	1001

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
ldi r16,0x05            ; Set up EIND and Z pointer
out EIND,r16
ldi r30,0x00
ldi r31,0x10
eicall                  ; Call to 0x051000
```

Words:	1 (2 Bytes)	Cycles:	4

EIJMP - Extended Indirect Jump

Mnemonic:	Function:		
EIJMP	Extended Indirect Jump to an Address		
Description:			
Indirect jump to the address pointed to by the Z (16 bits) Pointer Register in the Register File and the EIND Register in the I/O space. This instruction allows for indirect jumps to the entire Program memory space.			
Operation:	PC(15:0) ← Z(15:0) PC(21:16) ← EIND	**Syntax:**	EICALL
Operand:	None	**Program Counter:**	See Operation
		Stack:	Not Affected

16-Bit Opcode:

1001	0100	0001	1001

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
ldi r16,0x05            ; Set up EIND and Z pointer
out EIND,r16
ldi r30,0x00
ldi r31,0x10
eijmp                   ; Jump to 0x051000
```

Words:	1 (2 Bytes)	Cycles:	2

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

ELPM – Extended Load Program Memory

Mnemonic:	Function:
ELPM	Extended Load Program Memory

Description:

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

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

The result of these combinations is undefined:

ELPM r30, Z+

ELPM r31, Z+

Operation:	(i) R0 ← (RAMPZ:Z) ii) Rd ← (RAMPZ:Z) (iii) Rd ← (RAMPZ:Z) (RAMPZ:Z) ← (RAMPZ: Z) + 1	**Comments:**	(i) RAMPZ:Z: Unchanged, RO implied destination register. (ii) RAMPZ:Z: Unchanged (iii) RAMPZ:Z: Post-incremented
		Syntax:	(i) ELPM (ii) ELPM Rd, Z (iii) ELPM Rd, Z+
Operand:	(i) None, RO Implied (ii) 0 ≤ d ≤ 31 (iii) 0 ≤ d ≤ 31	**Program Counter:**	(i) PC ← PC + 1 (ii) PC ← PC + 1 (iii) PC ← PC + 1

16-Bit Opcode:

(i)	1001	0101	1101	1000
(ii)	1001	000d	dddd	0110
(iii)	1001	000d	dddd	0111

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
ldi  ZL,byte3 (Table_1<<1)   ; Initialize Z Pointer
out RAMPZ,ZL
ldi ZH,byte2 (Table_1<<1)
ldi ZL,byte2 (Table_1<<1)
elpmr16,Z+   ; Load constant from program
             ; Memory pointed to by RAMPZ:Z (Z is r31:r30)
    ...
Table_1:
.dw 0´3738             ; 0´38 is addressed when ZLSB = 0
                       ; 0´38 is addressed when ZLSB = 0
```

Words:	1 (2 Bytes)	Cycles:	3

EOR – Exclusive OR

Mnemonic:	Function:		
EOR	Exclusive OR		
Description:			
Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.			
Operation:	Rd ← Rd ⊕ Rr	**Syntax:**	EOR Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0010	01rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	N ⊕ V
	For signed tests.
V	0
	Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise.
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise

R (Result) equals Rd after the operation.

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

FMUL – Fractional Multiply Unsigned

Mnemonic:	Function:
FMUL	Fractional Multiply Unsigned

Description:

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

Rd		Rr		R1	R0
Multiplicand	X	Multiplier	Æ	Product High	Product Low
8		8		16	

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

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

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

Operation:	R1:RO ← Rd × Rr [Unsigned (1.5) ← unsigned (1.7) × unsigned 1.7]	**Syntax:**	FMUL Rd,Rr
Operand:	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	0011	0ddd	1rrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	⇔	⇔

C	R16
	Set if bit 15 of the result before left shift is set; cleared otherwise
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise

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

Example:

See Example on following page

Words:	1 (2 Bytes)	**Cycles:**	2

Example:

```
;*****************************************************
;*DESCRIPTION
;*  Signed fractional multiply of two 16-bit numbers
;*  with 32-bit result.
;*USAGE
;*  r19:r18:r17:r16 = ( r23:r22 * r21:r20 ) << 1
;*****************************************************
fmuls16x16_32:
    clrr2
    fmulsr23, r21    ;((signed)ah * (signed)bh) << 1
    movwr19:r18, r1:r0
    fmulr22, r20           ;(al * bl) << 1
    adcr18, r2
    movwr17:r16, r1:r0
    fmulsur23, r20         ;((signed)ah * bl) << 1
    sbcr19, r2
    addr17, r0
    adcr18, r1
    adcr19, r2
    fmulsur21, r22         ;((signed)bh * al) << 1
    sbcr19, r2
    addr17, r0
    adcr18, r1
    adcr19, r2
```

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

FMULS - Fractional Multiply Signed

Mnemonic:	Function:
FMULS	Fractional Multiply Signed

Description:

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

Rd		Rr		R1	R0
Multiplicand	X	Multiplier	→	Product High	Product Low
8		8		16	

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

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

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

Operation:	R1:R0 ← Rd × Rr [Signed (1.5) ← signed (1.7) × signed 1.7]	**Syntax:**	FMULS Rd,Rr
Operand:	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	0011	1ddd	0rrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	⇔	⇔

C	R16
	Set if bit 15 of the result before left shift is set; cleared otherwise
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise

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

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	2

FMULSU - Fractional Multiply Signed with Unsigned

Mnemonic:	Function:
FMULSU	Fractional Multiply Signed with Unsigned

Description:

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

Rd		Rr		R1	R0
Multiplicand	X	Multiplier	→	Product High	Product Low
8		8		16	

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

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

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

Operation:	R1:R0 ← Rd × Rr [Signed (1.5) ← signed (1.7) × unsigned 1.7]	**Syntax:**	FMULSU Rd,Rr
Operand:	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	0011	1ddd	1rrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	⇔	⇔

C	R16
	Set if bit 15 of the result before left shift is set; cleared otherwise
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise

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

Example:

See Code Example on the following page

Words:	1 (2 Bytes)	**Cycles:**	2

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

```
;*****************************************************
;*DESCRIPTION
;*  Signed fractional multiply of two 16-bit numbers
;*  with 32-bit result.
;*USAGE
;*  r19:r18:r17:r16 = ( r23:r22 * r21:r20 ) << 1
;*****************************************************
fmuls16x16_32:
    clrr2
    fmulsr23, r21    ;((signed)ah * (signed)bh) << 1
    movwr19: r18, r1:r0
    fmulr22, r20           ;(al * bl) << 1
    adcr18, r2
    movwr17:r16, r1:r0
    fmulsur23, r20         ;((signed)ah * bl) << 1
    sbcr19, r2
    addr17, r0
    adcr18, r1
    adcr19, r2
    fmulsur21, r22         ;((signed)bh * al) << 1
    sbcr19, r2
    addr17, r0
    adcr18, r1
    adcr19, r2
```

ICALL – Indirect Call to Subroutine

Mnemonic:	Function:		
ICALL	Indirect Call to a 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:	PC(15-0) ← Z(15-0)	**Syntax:**	ICALL
Operand:	None	**Program Counter:**	See Operation
		Stack:	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

16-Bit Opcode:

1001	0101	xxxx	1001

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	4

IJMP – Indirect Jump

Mnemonic:	Function:
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:	PC(15-0) ← Z(15-0) PC(21-16) is unchanged	**Syntax:**	IJMP
Operand:	None	**Program Counter:**	See Operation
		Stack:	Not Affected

16-Bit Opcode:

1001	0100	XXXX	1001

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	2

IN – Load an I/O Location to Register

Mnemonic:	Function:
IN	Load I/O Location To Register

Description:

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

Operation:	Rd ← I/O(P)	**Syntax:**	IN Rd,P
Operand:	0 ≤ d ≤ 31, 0 ≤ P ≤ 63	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1011	0PPd	dddd	PPPP

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
in  r25,0x16        ; 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

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

INC – Increment

Mnemonic:	Function:
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:	Rd ← Rd + 1	**Syntax:**	INC Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	010d	dddd	0011

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	—

S	$N \oplus V$
	For signed tests
V	$R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Two's complement overflow occurs if and only if Rd was 0x7F before the operation
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise

R (Result) equals Rd after the operation.

Example:

```
clr r22                ; clear r22
loop: inc   r22        ; increment r22
    ...
cpi r22,0x4F           ; Compare r22 to 0x4f
brne loop              ; Branch if not equal
nop                    ; Continue (do nothing)
```

Words:	1 (2 Bytes)	**Cycles:**	1

JMP – Jump

Mnemonic:	Function:		
JMP	Jump		
Description:			
Jump to an address within the entire 4M (words) program memory. See also RJMP.			
Operation:	PC ← k	**Syntax:**	JMP k
Operand:	0 ≤ k < 4M	**Program Counter:**	PC ← k
		Stack:	Unchanged

32-Bit Opcode:

1001	010K	kkkk	110k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	2 (4 Bytes)	Cycles:	3

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

LD – Load Indirect from SRAM to Register Using Index X

Mnemonic:	Function:
LD	Load Indirect from SRAM to Register using Index X

Description:

Loads one byte indirect from SRAM, I/O location or register file to register. This memory location is pointed to by the X (16 bits) pointer register in the register file. Memory access is limited to the current SRAM, I/O location or register file page of 64K bytes.

The X pointer register can either be left unchanged by 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.

The results loaded by the following instructions are undefined:

ld r26, X+

ld r27, X+

ld r26, -X

ld r27, -X

Operation:	(i) Rd ← (X) (ii) Rd ← (X), X ← X + 1 (iii) X ← X - 1, Rd ← (X)	**Comments:**	(i) X: Unchanged (ii) X: Post-incremented (iii) X: Pre-decremented
		Syntax:	(i) LD Rd, X (ii) LD Rd, X+ (iii) LD Rd, -X
Operand:	(i) 0 ≤ d ≤ 31 (ii) 0 ≤ d ≤ 31 (iii) 0 ≤ d ≤ 31	**Program Counter:**	(i) PC ← PC + 1 (ii) PC ← PC + 1 (iii) PC ← PC + 1

16-Bit Opcode:

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

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r27              ; Clear X high byte
ldi r26,0x1F         ; Set X low byte to 0x1F
ld  r0,X+            ; Load r0 with memory loc. 0x1F-
                     ; R31(X post inc)
ld  r1,X             ; Load r1 with memory loc. 0x20-I/O loc.
                     ; 0x00
ldi r26,0x60         ; Set X low byte to 0x60
ld  r2,X             ; Load r2 with memory loc. 0x60-SRAM loc.
                     ; 0x60
ld  r3,-X            ; Load r3 with memory loc. 0x5F-I/O loc.
                     ; 0x3F(X pre dec)
```

Words:	1 (2 Bytes)	**Cycles:**	2

LD (LDD) – Load Indirect from SRAM to Register Using Index Y

Mnemonic:	Function:
LD(LDD)	Load Indirect from SRAM to Register using Index Y

Description:

Loads one byte indirect with or without displacement from SRAM, I/O location or register file to register. The memory location is pointed to by the Y (16 bits) pointer register in the register file. Memory access is limited to the current SRAM, I/O location or register file page of 64K bytes.

The Y pointer register can either be left unchanged by 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.

The results loaded by the following instructions are undefined:

ld r28, Y+

ld r29, Y+

ld r28, -Y

ld r29, -Y

Operation:	(i) Rd ← (Y) (ii) Rd ← (Y), Y ← Y + 1 (iii) Y ← Y - 1, Rd ← (Y) (iv) Rd ← (Y+q)	**Comments:**	(i) Y: Unchanged (ii) Y: Post-incremented (iii) Y: Pre-decremented (iv) Y: Unchanged q: Displacement
		Syntax:	(i) LD Rd, Y (ii) LD Rd, Y+ (iii) LD Rd, -Y (iv) LDD Rd, Y=q
Operand:	(i) 0 ≤ d ≤ 31 (ii) 0 ≤ d ≤ 31 (iii) 0 ≤ d ≤ 31 (iv) 0 ≤ d ≤ 31, 0 ≤ q ≤ 63	**Program Counter:**	(i) PC ← PC + 1 (ii) PC ← PC + 1 (iii) PC ← PC + 1 (iv) PC ← PC + 1

16-Bit Opcode:

(i)	1001	000d	dddd	1000
(ii)	1001	000d	dddd	1001
(iii)	1001	000d	dddd	1010
(iv)	10q0	qq0d	dddd	1qqq

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r29      ; Clear Y high byte
ldi r28,0x1F ; Set Y low byte to 0x1F
ld  r0,Y+    ; Load r0 with mem loc. 0x1F-R31 (Y post inc)
ld  r1,Y     ; Load r1 with mem loc. 0x20-I/O loc. 0x00
ldi r28,0x60           ; Set Y low byte to 0x60
ld  r2,Y     ; Load r2 with mem loc. 0x60-SRAM loc. 0x60
ld  r3,-Y    ; Load r3 with mem loc. 0x5F-I/O loc. 0x3F
             ; (Y pre dec)
ldd r4,Y+2   ; Load r4 with mem loc. 0x61-SRAM loc. 0x61
```

Words:	1 (2 Bytes)	Cycles:	2

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

LD (LDD) – Load Indirect from SRAM to Register Using Index Z

Mnemonic:	Function:
LD(LDD)	Load Indirect from SRAM to Register using Index Z

Description:

Loads one byte indirect with or without displacement from SRAM, I/O location or register file to register. The memory location is pointed to by the Z (16 bits) pointer register in the register file. Memory access is limited to the current SRAM, I/O location or register file page of 64K bytes.

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

The results loaded by the following instructions are undefined:

ld r30, Z+

ld r31, Z+

ld r30, -Z

ld r31, -Z

Operation:	(i) Rd ← (Z) (ii) Rd ← (Z), Z ← Z + 1 (iii) Z ← Z - 1, Rd ← (Z) (iv) Rd ← (Z+q)	**Comments:**	(i) Z: Unchanged (ii) Z: Post-incremented (iii) Z: Pre-decremented (iv) Z: Unchanged q: Displacement
		Syntax:	(i) LD Rd, Z (ii) LD Rd, Z+ (iii) LD Rd, -Z (iv) LDD Rd, Z+q
Operand:	(i) 0 ≤ d ≤ 31 (ii) 0 ≤ d ≤ 31 (iii) 0 ≤ d ≤ 31 (iv) 0 ≤ d ≤ 31, 0 ≤ q ≤ 63	**Program Counter:**	(i) PC ← PC + 1 (ii) PC ← PC + 1 (iii) PC ← PC + 1 (iv) PC ← PC + 1

16-Bit Opcode:

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

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r29      ; Clear Z high byte
ldi r28,0x1F ; Set Z low byte to 0x1F
ld  r0,Z+    ; Load r0 with mem loc. 0x1F-R31 (Z post inc)
ld  r1,Z     ; Load r1 with mem loc. 0x20-I/O loc. 0x00
ldi r28,0x60           ; Set Z low byte to 0x60
ld  r2,Z     ; Load r2 with mem loc. 0x60-SRAM loc. 0x60
ld  r3,-Z    ; Load r3 with mem loc. 0x5F-I/O loc. 0x3F
             ; (Z pre dec)
ldd r4,Z+2   ; Load r4 with mem loc. 0x61-SRAM loc. 0x61
```

Words:	1 (2 Bytes)	Cycles:	2

LDI – Load Immediate

Mnemonic:	Function:		
LDI	Load Immediate		
Description:			
Loads an 8-bit constant directly to register 16 to 31.			
Operation:	Rd ← K	**Syntax:**	LDI Rd,K
Operand:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	**Program Counter:**	PC ← PC + 1

32-Bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r31        ; Clear Z high byte
ldi r30,0xF0          ; Set Z low byte to 0xF0
lpm            ; Load constant from program
               ; memory pointed to by Z
```

Words:	1 (2 Bytes)	Cycles:	1

LDS – Load Direct from SRAM

Mnemonic:	Function:		
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:	Rd ← K	**Syntax:**	LDS Rd,K
Operand:	0 ≤ d ≤ 31, 0 ≤ k ≤ 65535	**Program Counter:**	PC ← PC + 2

32-Bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
lds r2,0xFF00          ; Load r2 with the contents of
                       ; SRAM location 0xFF00
add r2,r1              ; add r1 to r2
sts 0xFF00,r2          ; Write back
```

Words:	2 (4 Bytes)	Cycles:	3

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

LPM – Load Program Memory

Mnemonic:	Function:
LPM	Load Program Memory

Description:

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

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

The result of these combinations is undefined:

LPM r30, Z+

LPM r31, Z+

Operation:	(i) R0 ← (Z) (ii) Rd ← (Z) (iii) Rd ← (Z), Z ← Z + 1	**Comments:**	(i) Z: Unchanged, R0 implied destination register (ii) Z: Unchanged (iii) Z: Post-incremented
		Syntax:	(i) LPM (ii) LPM Rd, Z (iii) LPM Rd, Z+
Operand:	(i) None, R0 implied (ii) 0 ≤ d ≤ 31 (iii) 0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	3

LSL – Logical Shift Left

Mnemonic:	Function:		
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 unsigned value by two.			
Operation:	$C \leftarrow b7 \overset{\leftarrow}{\ldots} b0 \leftarrow 0$	**Syntax:**	LSL Rd
Operand:	$0 \le d \le 31$	**Program Counter:**	$PC \leftarrow PC + 1$

16-Bit Opcode:

0000	11dd	dddd	dddd

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

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	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise
C	Rd7
	Set if, before the shift, the MSB of Rd was set; cleared otherwise

R (Result) equals Rd after the operation.

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

LSR – Logical Shift Right

Mnemonic:	Function:		
LSR	Logical Shift Right		
Description:			
Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C flag of the SREG. This operation effectively divides an unsigned value by two. The C flag can be used to round the result.			
Operation:	$0 \rightarrow \overrightarrow{b7 \ldots b0} \rightarrow C$	**Syntax:**	LSR Rd
Operand:	$0 \le d \le 31$	**Program Counter:**	$PC \leftarrow PC + 1$

16-Bit Opcode:

1001	010d	dddd	0110

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	0	⇔	⇔

S	$N \oplus V$
	For signed tests
V	$N \oplus C$ (for N and C after the shift)
	Set if (N is set and C is clear) or (N is clear and C is set); Cleared otherwise (for values of N and C after the shift)
N	0
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise
C	Rd0
	Set if, before the shift, the LSB of Rd was set; cleared otherwise

R (Result) equals Rd after the operation.

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

MOV – Copy Register

Mnemonic:	Function:		
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:	Rd ← Rr	**Syntax:**	MOV Rd, Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

32-Bit Opcode:

0010	11rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
mov r16,r0  ; Copy r0 to r16
call check  ; Call subroutine
    ...
check: cpi  r16,0x11 ; Compare r16 to 0x11
    ...
ret                  ; Return from subroutine
```

Words:	1 (2 Bytes)	Cycles:	1

MOVW – Copy Register Word

Mnemonic:	Function:		
MOVW	Copy Register Word		
Description:			
This instruction makes a copy of one register pair into another register pair. The source register pair Rr+1:Rr is left unchanged, while the destination register pair Rd+1:Rd is loaded with a copy of Rr + 1:Rr.			
Operation:	Rd + 1:Rd ← Rr + 1:Rr	**Syntax:**	MOVW Rd+1:Rd,Rr+1Rr
Operand:	d ∈ {0,2,...,30}, r ∈ {0,2,...,30}	**Program Counter:**	PC ← PC + 1

32-Bit Opcode:

0000	0001	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
movw r17:16,r1:r0    ; Copy r1:r0 to r17:r16
call check           ; Call subroutine
    ...
check: cpi  r16,0x11 ; Compare r16 to 0x11
    ...
cpi r17,0x32         ; Compare r17 to 0x32
    ...
ret                  ; Return from subroutine
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

MUL – Multiply Unsigned

Mnemonic:	Function:
MUL	Multiply Unsigned

Description:

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

Rd		Rr		R1	R0
Multiplicand	X	Multiplier	→	Product High	Product Low
8		8		16	

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

Operation:	R1:R0 ← Rd × Rr (unsigned ← unsigned × signed)	**Syntax:**	MUL Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	11rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	⇔	⇔

C	R15
	Set if bit 15 of the result before left shift is set; cleared otherwise
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; Cleared otherwise

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

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	2

MULS – Multiply Signed

Mnemonic:	Function:
MULS	Multiply Signed

Description:

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

Rd		Rr		R1	R0
Multiplicand	X	Multiplier	→	Product High	Product Low
8		8		16	

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

Operation:	R1:R0 ← Rd × Rr (signed ← signed × signed)	**Syntax:**	MULS Rd,Rr
Operand:	16 ≤ d ≤ 31, 16 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	0010	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	⇔	⇔

C	R15
	Set if bit 15 of the result before left shift is set; cleared otherwise
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; Cleared otherwise

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

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	2

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

MULSU – Multiply Signed with Unsigned

Mnemonic:	Function:
MULSU	Multiply Signed with Unsigned

Description:

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

Rd		Rr		R1	R0
Multiplicand	X	Multiplier	→	Product High	Product Low
8		8		16	

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

Operation:	R1:R0 ← Rd × Rr (signed ← signed × unsigned)	**Syntax:**	MULSU Rd,Rr
Operand:	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	0001	0ddd	0rrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	⇔	⇔

C	R15
	Set if bit 15 of the result before left shift is set; cleared otherwise
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; Cleared otherwise

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

Example:

```
;*************************************************
;* Description
;* Signed multiply of two 16-bit numbers with
;* 32-bit result.
;* Usage
;* r19:r18:r17:r16 = r23:r22 * r21:r20
;*************************************************
muls16x16_32:
clrr2
mulsr23, r21;    (signed) ah * (signed) bh
movwr19:r18, r1:r0
mulr22, r20; al * bl
movwr17:r16, r1:r0
mulsur23, r20; (signed)ah * bl
sbcr19, r2
addr17, r0
adcr18, r1
adcr19, r2
mulsur21, r22; (signed)bh * bl
sbcr19, r2
addr17, r0
adcr18, r1
adcr19, r2
ret
```

Words:	1 (2 Bytes)	**Cycles:**	2

NEG – Two's Complement

Mnemonic:	Function:		
NEG	Two's Complement		
Description:			
Replaces the contents of register Rd with its two's complement; the value 0x80 is left unchanged			
Operation:	Rd ← 0x00 - Rd	**Syntax:**	NEG Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	010d	dddd	0001

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	R3 • Rd3
	Set if there was a borrow from bit 3; cleared otherwise
S	N ⊕ V
	For signed tests
V	$R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise.
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; 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 0x00

R (Result) equals Rd after the operation.

Example:

```
sub     r11,r0    ; Subtract r0 from r11
brpl    pos.      ; Branch if result positive
neg     r11       ; Take two's complement of r11
pos:    nop       ; Branch destination (do nothing)
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

NOP – No Operation

Mnemonic:	Function:		
NOP	No Operation		
Description:			
This instruction performs a single cycle No Operation.			
Operation:	No	**Syntax:**	NOP
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	0000	0000	0000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r16              ; Clear r16
ser r17              ; Set r17
out 0x18,r16         ; Write zeros to Port B
nop                  ; Wait (do nothing)
out 0x18,r17         ; Write ones to Port B
```

Words:	1 (2 Bytes)	Cycles:	1

OR – Logical OR

Mnemonic:	Function:		
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:	Rd ← Rd v Rr	**Syntax:**	OR Rd,Rr
Operand:	$0 \le d \le 31$, $0 \le r \le 31$	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0010	10rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	$N \oplus V$ - For signed tests
V	0 - Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; Cleared otherwise

R (Result) equals Rd after the operation.

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

ORI – Logical OR with Immediate

Mnemonic:	Function:		
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:	Rd ← Rd v K	**Syntax:**	ORI Rd, K
Operand:	0 ≤ d ≤ 31, 0 ≤ K ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0110	KKKK	dddd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	N ⊕ V - For signed tests
V	0 - Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; Cleared otherwise

R (Result) equals Rd after the operation.

Example:

```
ori     r16,0xF0      ; Set high nibble of r16
ori     r17,1         ; Set bit 0 of r17
```

Words:	1 (2 Bytes)	Cycles:	1

OUT – Store Register to I/O Location

Mnemonic:	Function:		
OUT	Store Register to I/O Location		
Description:			
Stores data from register Rr in the register file to I/O Space (Ports, Timers, Configuration registers etc.).			
Operation:	P ← Rr	**Syntax:**	OUT P,Rr
Operand:	0 ≤ r ≤ 31, 0 ≤ P ≤ 63	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1011	1PPr	rrrr	PPPP

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r16      ; Clear r16
ser r17      ; Set r17
out 0x18,r16          ; Write zeros to Port B
nop                   ; Wait (do nothing)
out 0x18,r17          ; Write ones to Port B
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

POP – Pop Register from Stack

Mnemonic:	Function:		
POP	POP Register from Stack		
Description:			
This instruction loads register Rd with a byte from the STACK.			
Operation:	Rd ← STACK	**Syntax:**	POP Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1
		Stack:	SP ← SP + 1

16-Bit Opcode:

1001	000d	dddd	1111

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

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

Mnemonic:	Function:		
PUSH	Push Register on Stack		
Description:			
This instruction stores the contents of register Rr on the STACK.			
Operation:	STACK ← Rr	**Syntax:**	PUSH Rr
Operand:	0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1
		Stack:	SP ← SP - 1

16-Bit Opcode:

1001	001d	dddd	1111

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

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

Mnemonic:	Function:		
RCALL	Relative Call to Subroutine		
Description:			
Stores data from register Rr in the register file to I/O Space (Ports, Timers, Configuration registers etc.).			
Operation:	PC ← PC + k + 1	**Syntax:**	RCALL k
Operand:	-2K ≤ k < 2K	**Program Counter:**	PC ← PC + k + 1
		Stack:	STACK ← PC + 1, SP ← SP-3

16-Bit Opcode:

1101	kkkk	kkkk	kkkk

Status Register (SREG) Boolean Formula:

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

Mnemonic:	Function:		
RET	Return From a Subroutine		
Description:			
Returns from subroutine. The return address is loaded from the STACK.			
Operation:	PC(21-0) ← STACK	**Syntax:**	RET
Operand:	None	**Program Counter:**	See Operation
		Stack:	SP ← SP+3

16-Bit Opcode:

1001	0101	0000	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

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

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

RETI – Return from Interrupt

Mnemonic:	Function:		
RETI	Return From Interrupt		
Description:			
Returns from interrupt. The return address is loaded from the STACK and the global interrupt flag is set. Note that the Status Register is not automatically stored when entering an interrupt routine and is not restored when returning from an interrupt routine. This must be handled by the application program. The Stack Pointer uses a pre-increment scheme during RETI.			
Operation:	PC(21-0) ← STACK	**Syntax:**	RETI
Operand:	None	**Program Counter:**	See Operation
		Stack:	SP ← SP+3

16-Bit Opcode:

1001	0101	0001	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
1	—	—	—	—	—	—	—

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

Mnemonic:	Function:		
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:	PC ← PC + k + 1	**Syntax:**	RJMP k
Operand:	-2K ≤ k < 2K	**Program Counter:**	PC ← PC + k + 1
		Stack:	Unchanged

16-Bit Opcode:

1100	kkkk	kkkk	kkkk

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
    cpi     r16,0x42        ; Compare r16 to 0x42
    brne    error     ; Branch if r16 <> 0x42
    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

Mnemonic:	Function:		
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:	$C \leftarrow b7 \ldots b0 \leftarrow C$	**Syntax:**	ROL Rd
Operand:	$0 \le d \le 31$	**Program Counter:**	$PC \leftarrow PC + 1$

16-Bit Opcode:

0001	11dd	dddd	dddd

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

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	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; 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:

```
rol r16      ; Rotate left
brcs         oneenc    ; Branch if carry set
    ...
oneenc:nop  ;Branch destination (do nothing)
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

ROR – Rotate Right Through Carry

Mnemonic:	Function:
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:	$C \rightarrow \overrightarrow{b7...b0} \rightarrow C$	**Syntax:**	ROR Rd
Operand:	$0 \le d \le 31$	**Program Counter:**	$PC \leftarrow PC + 1$

16-Bit Opcode:

0001	010d	dddd	0111

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	⇔

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	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; 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:

```
ror r15     ; Rotate left
brcc        zeroenc  ; Branch if carry set
    ...
oneenc:nop  ;Branch destination (do nothing)
```

Words:	1 (2 Bytes)	**Cycles:**	1

SBC – Subtract with Carry

Mnemonic:	Function:
SBC	Subtract with Carry

Description:

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

Operation:	Rd ← Rd - Rr - C	**Syntax:**	SBC Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0000	10rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
	Set if there was a borrow from bit 3; cleared otherwise
S	$N \oplus V$
	For signed tests
V	$Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
	Previous value remains unchanged when the result is zero; cleared otherwise
C	$\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
	Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of the Rd; cleared otherwise

R (Result) equals Rd after the operation.

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SBCI – Subtract Immediate with Carry

Mnemonic:	Function:		
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:	Rd ← Rd - K - C	**Syntax:**	SBCI Rd,k
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0100	KKKK	dddd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
	Set if there was a borrow from bit 3; cleared otherwise
S	$N \oplus V$
	For signed tests
V	$Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
	Previous value remains unchanged when the result is zero; cleared otherwise
C	$\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
	Set if the absolute value of the contents of Rr plus previous carry is larger than the absolute value of the Rd; cleared otherwise

R (Result) equals Rd after the operation.

Example:

```
                        ; Subtract 0x4F23 from r17:r16
subi r16,0x23           ; Subtract low byte
sbci r17,0x4F           ; Subtract with carry high byte
```

Words:	1 (2 Bytes)	Cycles:	1

SBI – Set Bit in I/O Register

Mnemonic:	Function:		
SBI	Set Bit in I/O Register		
Description:			
Sets a specified bit in an I/O register. This instruction operates on the lower 32 I/O registers – addresses 0-31.			
Operation:	I/O(P,b) ← 1	**Syntax:**	SBI P,b
Operand:	0 ≤ P ≤ 31, 0 ≤ b ≤ 7	**Program Counter:**	C ← PC + 1

16-Bit Opcode:

1001	1010	PPPP	Pbbb

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
out   0x1E,r0        ; Write EEPROM address
sbi   0x1C,0         ; Set read bit in EECR
in    r1,0x1D        ; Read EEPROM data
```

Words:	1 (2 Bytes)	Cycles:	2

SBIC – Skip if Bit in I/O Register is Cleared

Mnemonic:	Function:		
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:	If I/O(P,b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1	**Syntax:**	SBIC P,b
Operand:	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:

1001	1001	PPPP	Pbbb

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
e2wait: sbic0x1C,1   ; Skip next inst. if
                     ; EEWE cleared
        rjmpe2wait   ; 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)

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SBIS – Skip if Bit in I/O Register is Set

Mnemonic:	Function:		
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:	If I/O(P,b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1	**Syntax:**	SBIC P,b
Operand:	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:

1001	1011	PPPP	Pbbb

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
waitset: sbis 0x10,0        ; 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

Mnemonic:	Function:		
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:	Rd+1:Rd ← Rd+1:Rd - K	**Syntax:**	SBIW Rdl,K
Operand:	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0111	KKdd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	⇔	⇔	⇔	⇔

S	$N \oplus V$
	For signed tests
V	$Rdh7 \bullet \overline{R15}$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R15
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x0000; cleared otherwise
C	$R15 \bullet \overline{Rdh7}$
	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

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SBR – Set Bits in Register

Mnemonic:	Function:		
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:	Rd ← Rd v K	**Syntax:**	SBR Rd, K
Operand:	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0110	KKKK	dddd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	N ⊕ V - For signed tests
V	0 - Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; Cleared otherwise

R (Result) equals Rd after the operation.

Example:

```
sbr   r16,3          ; Set bits 0 and 1 in r16
sbr   r17,0xF0       ; Set 4 MSB in r17
```

Words:	1 (2 Bytes)	Cycles:	1

SBRC – Skip if Bit in Register is Cleared

Mnemonic:	Function:
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:	If Rr(b) = 0 then PC ← PC + 2 (or 3) else PC ← PC + 1	**Syntax:**	SBRC Rr,b
Operand:	0 ≤ r ≤ 31, 0 ≤ b ≤ 7	**Program Counter:**	PC ← PC + 1, Condition false, no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two work instruction

16-Bit Opcode:

1111	110r	rrrr	0bbb

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	**Cycles:**	1 if condition is false (no skip) 2 if condition is true (skip is executed) and the instruction skipped is 1 word 3 If condition is true (skip is executed) and the instruction skipped is 2 words

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SBRS – Skip if Bit in Register is Set

Mnemonic:	Function:		
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 cleared.			
Operation:	If Rr(b) = 1 then PC ← PC + 2 (or 3) else PC ← PC + 1	**Syntax:**	SBRS Rr,b
Operand:	0 ≤ r ≤ 31, 0 ≤ b ≤ 7	**Program Counter:**	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	111r	rrrr	0bbb

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

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

Mnemonic:	Function:		
SEC	Set Carry Flag		
Description:			
Sets the Carry flag (C) in SREG (status register).			
Operation:	C ← 1	**Syntax:**	SEC
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0000	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	1

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

Mnemonic:	Function:		
SEH	Set Half Carry Flag		
Description:			
Sets the Half Carry flag (C) in SREG (status register).			
Operation:	H ← 1	**Syntax:**	SEH
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0101	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	1	—	—	—	—	—

H	1 - Half Carry Flag Set

Example:

```
seh          ; Set half carry flag
```

Words:	1 (2 Bytes)	Cycles:	1

SEI – Set Global Interrupt Flag

Mnemonic:	Function:		
SEI	Set Global Interrupt Flag		
Description:			
Sets the Half Carry flag (C) in SREG (status register).			
Operation:	I ← 1	**Syntax:**	SEI
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0111	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
1	—	—	—	—	—	—	—

I	1 - Global Interrupt Flag Set

Example:

```
cli          ; Disable interrupts
in  r13,0x16          ; Read Port B
sei          ; Enable interrupts
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SEN – Set Negative Flag

Mnemonic:	Function:		
SEN	Set Negative Flag		
Description:			
Sets the Negative flag (N) in SREG (status register).			
Operation:	N ← 1	**Syntax:**	SEN
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0010	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	1	—	—

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

Mnemonic:	Function:		
SER	Set All Bits in Register		
Description:			
Load 0xFF directly into Register Rd			
Operation:	Rd ← 0xFF	**Syntax:**	SER
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1110	1111	dddd	1111

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr  r16    ; Clear r16
ser  r17            ; Set r17
out  0x18,r16       ; Write zeros to Port B
nop                 ; Delay (do nothing)
out  0x18,r17       ; Write ones to Port B
```

Words:	1 (2 Bytes)	Cycles:	1

SES – Set Signed Flag

Mnemonic:	Function:		
SES	Set Signed Flag		
Description:			
Sets the Signed flag (S) in SREG (status register).			
Operation:	S ← 1	**Syntax:**	SES
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0100	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	1	—	—	—	—

S	1 - Signed Flag Set

Example:

```
add r2,r19  ; Add r19 to r2
ses         ; Set Signed flag
```

Words:	1 (2 Bytes)	Cycles:	1

SET – Set T Flag

Mnemonic:	Function:		
SET	Set T Flag		
Description:			
Sets the T in SREG (status register).			
Operation:	T ← 1	**Syntax:**	SET
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0110	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	1	—	—	—	—	—	—

T	1 - T Flag Set

Example:

```
set         ; Set T flag
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SEV – Set Overflow Flag

Mnemonic:	Function:		
SEV	Set Overflow Flag		
Description:			
Sets the Overflow in SREG (status register).			
Operation:	V ← 1	**Syntax:**	SEV
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0011	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	1	—	—	—

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

Mnemonic:	Function:		
SEZ	Set Zero Flag		
Description:			
Sets the Zero Flag in SREG (status register).			
Operation:	Z ← 1	**Syntax:**	SEZ
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0100	0001	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	1	—

V	1 - Overflow Flag Set

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

SLEEP

Mnemonic:	Function:		
SLEEP	Sleep Mode		
Description:			
This instruction sets the circuit in sleep mode defined by the MCU control register. Refer to the device documentation for detailed description of SLEEP usage.			
Operation:	See Description	**Syntax:**	SLEEP
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:			
1001	0101	1000	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

SPM – Store Program Memory

Note: R1 determines the instruction high byte, and R0 determines the instruction low byte.

Mnemonic:	Function:		
SPM	Store Program Memory		
Description:			
To erase and /or write to FLASH (Code Space). Reference M3000 Manual Section 9: Instruction Memory Programming.			
Operation:	(ii) (RAMPZ:Z) ← R1:R0	Write Program Memory Word	
	(iii) (RAMPZ:Z) ← R1:R0	Write Temporary Page Buffer	
	(iv) (RAMPZ:Z) ← TEMP	Write Temporary Page Buffer to Program Memory	
Operand:	None	**Syntax:**	SPM
		Program Counter:	PC ← PC + 1

16-Bit Opcode:			
1001	0101	1110	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

See Following Page for Examples

Words:	1 (2 Bytes)	Cycles:	Depends on Operation

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

ST – Store Indirect from Register to Data Space Using Index X

Mnemonic:	Function:
ST	Store Indirect From Register to Data Space Using Index X

Description:

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

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

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

The results stored by the following instructions are undefined:

ST X+, r26
ST X+, r27
ST -X, r26
ST -X, r27

Operation:	(i) (X) ← Rr (ii) (X) ← Rr , X ← X+1 (iii) X ← X - 1, (X) ← Rr	X: Unchanged X: Post-incremented X: Pre-decremented	
Operand:	0 ≤ r ≤ 31 (For All)	**Syntax:**	(i) ST X, Rr (ii) ST X+, Rr (iii) ST -X, Rr
		Program Counter:	PC ← PC + 1 (For All)

16-Bit Opcode:

(i)	1001	001r	rrrr	1100
(ii)	1001	001r	rrrr	1101
(iii)	1001	001r	rrrr	1110

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r27              ; Clear X high byte
ldi r26,0x60         ; Set X low byte to 0x60
st X+,r0    ; Store r0 in data loc. 0x60 (X post inc)
st X,r1     ; Store r1 in data space location 0x61
ldi r26,0x63         ; Set X low byte to 0x63
st X,r2     ; Store r2 in data space location 0x63
st -X,r3    ; Store r3 in data loc. 0x62 (X pre dec)
```

Words:	1 (2 Bytes)	**Cycles:**	2

ST (STD) – Store Indirect from Register to Data Space Using Index Y

Mnemonic:	Function:
ST	Store Indirect From Register to Data Space Using Index X

Description:

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

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

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

The results stored by the following instructions are undefined:

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

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

Operand:	0 ≤ r ≤ 31 (For i - iii) (iiii) 0 ≤ r ≤ 31, 0 ≤ q ≤ 63	**Syntax:**	(i) ST Y, Rr (ii) ST Y+, Rr (iii) ST -Y, Rr (iiii) STD Y+q, Rr
		Program Counter:	PC ← PC + 1 (For All)

16-Bit Opcode:

(i)	1000	001r	rrrr	1000
(ii)	1001	001r	rrrr	1001
(iii)	1001	001r	rrrr	1010
(iiii)	10q0	qq1r	rrrr	1qqq

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r29              ; Clear Y high byte
ldi r28,0x60         ; Set Y low byte to 0x60
st Y+,r0    ; Store r0 in data loc. 0x60 (Y post inc)
st Y,r1     ; Store r1 in data space location 0x61
ldi r28,0x63         ; Set Y low byte to 0x63
st Y,r2     ; Store r2 in data space location 0x63
st -Y,r3    ; Store r3 in data loc. 0x62 (Y pre dec)
std Y+2,r4  ; Store r4 in in data space location 0x64
```

Words:	1 (2 Bytes)	**Cycles:**	2

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

ST (STD) – Store Indirect from Register to Data Space Using Index Z

Mnemonic:	Function:
ST(STD)	Store Indirect From Register to Data Space Using Index Z

Description:

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

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

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

The results stored by the following instructions are undefined:

ST Z+, r30
ST Z+, r31
ST -Z, r30
ST -Z, r31

Operation:	(i) (Z) ← Rr	Z: Unchanged
	(ii) (Z) ← Rr, Z ← Y+1	Z: Post incremented
	(iii) Z ← Z - 1, (Z) ← Rr	Z: Pre decremented
	(iiii) (Z+q) ← Rr	Z: Unchanged, q: Displacement

Operand:	0 ≤ r ≤ 31 (For i - iii) (iiii) 0 ≤ r ≤ 31, 0 ≤ q ≤ 63	**Syntax:**	(i)	ST Z, Rr
			(ii)	ST Z+, Rr
			(iii)	ST -Z, Rr
			(iiii)	STD Z+q, Rr
		Program Counter:	PC ← PC + 1 (For All)	

16-Bit Opcode:

(i)	1000	001r	rrrr	0000
(ii)	1001	001r	rrrr	0001
(iii)	1001	001r	rrrr	0010
(iiii)	10q0	qq1r	rrrr	0qqq

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
clr r29              ; Clear Y high byte
ldi r28,0x60         ; Set Y low byte to 0x60
st Y+,r0    ; Store r0 in data loc. 0x60 (Y post inc)
st Y,r1     ; Store r1 in data space location 0x61
ldi r28,0x63         ; Set Y low byte to 0x63
st Y,r2     ; Store r2 in data space location 0x63
st -Y,r3    ; Store r3 in data loc. 0x62 (Y pre dec)
std Y+2,r4  ; Store r4 in in data space location 0x64
```

STS – Store Direct to SRAM

Mnemonic:	Function:
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:	(k) ← Rr	**Syntax:**	STS k,Rr
Operand:	0 ≤ r ≤ 31, 0 ≤ k ≤ 65535	**Program Counter:**	PC ← PC + 2

16-Bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
lds r2,0xFF00        ; Load r2 with the contents of
                     ; SRAM location 0xFF00
add r2,r1            ; add r1 to r2
sts 0xFF00,r2        ; Write back
```

Words:	2 (4 Bytes)	**Cycles:**	3

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SUB – Subtract Without Carry

Mnemonic:	Function:		
SUB	Subtract Without Carry		
Description:			
Subtracts two registers and places the result in the destination register Rd.			
Operation:	Rd ← Rd - Rr	**Syntax:**	SUB Rd,Rr
Operand:	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0001	10rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
	Set if there was a borrow from bit 3; cleared otherwise
S	$N \oplus V$
	For signed tests
V	$Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise
C	$\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
	Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise

R (Result) 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

Mnemonic:	Function:		
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:	Rd ← Rd - K	**Syntax:**	SUBI Rd,K
Operand:	0 ≤ d ≤ 31, 0 ≤ K ≤ 255	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0101	KKKK	dddd	KKKK

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	⇔	⇔	⇔	⇔	⇔	⇔

H	$\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
	Set if there was a borrow from bit 3; cleared otherwise
S	$N \oplus V$
	For signed tests
V	$Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
	Set if two's complement overflow resulted from the operation; cleared otherwise
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; cleared otherwise
C	$\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
	Set if the absolute value of the contents of Rr is larger than the absolute value of Rd; cleared otherwise

R (Result) equals Rd after the operation

Example:

```
subi        r22,0x11        ; Subtract 0x11 from r22
brne        noteq     ; Branch if r22<>0x11
            ...
noteq:      nop       ; Branch destination (do nothing)
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

SWAP – Swap Nibbles

Mnemonic:	Function:		
SWAP	Swap Nibbles		
Description:			
Swaps high and low nibbles in a register.			
Operation:	R(7:4) ← Rd(3:0), R(3:0) ← Rd(7:4)	**Syntax:**	SWAP Rd
Operand:	0 ≤ d ≤ 31	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	01dd	dddd	0010

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

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

Words:	1 (2 Bytes)	Cycles:	1

TST – Test for Zero or Minus

Mnemonic:	Function:		
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:	Rd ← Rd - K	**Syntax:**	SUBI Rd,K
Operand:	0 ≤ d ≤ 31, 0 ≤ K ≤ 255	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

0010	00dd	dddd	dddd

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	⇔	0	⇔	⇔	—

S	N ⊕ V - For signed tests
V	0 - Cleared
N	R7
	Set if MSB of the result is set; cleared otherwise
Z	$\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
	Set if the result is 0x00; 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

Mnemonic:	Function:		
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:	WD timer restart.	**Syntax:**	WDR
Operand:	None	**Program Counter:**	PC ← PC + 1

16-Bit Opcode:

1001	0101	1010	1000

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
—	—	—	—	—	—	—	—

Example:

```
wdr          ; Reset watchdog timer
```

Words:	1 (2 Bytes)	Cycles:	1

Functional Grouping Color Coding

- Arithmetic and Logic Instructions
- MCU Control Instructions
- Branch Instructions
- Data Transfer Instructions
- Bit and Bit Test Instructions

WARRANTY

System Semiconductor (“SSI”), warrants only to the purchaser of the Product from SSI (the “Customer”) that the product purchased from SSI (the “Product”) will be free from defects and meets the applicable specifications at the time of sale. Customer’s exclusive remedy under this Limited Warranty shall be the repair or replacement, at Company’s sole option, of the Product, or any part of the Product, determined by SSI to be defective.

This Limited Warranty does not extend to any Product damaged by reason of alteration, accident, abuse, neglect or misuse or improper or inadequate handling; improper or inadequate wiring utilized or installed in connection with the Product; installation, operation or use of the Product not made in strict accordance with the specifications and written instructions provided by SSI; use of the Product for any purpose other than those for which it was designed; ordinary wear and tear; disasters or Acts of God; unauthorized attachments, alterations or modifications to the Product; the misuse or failure of any item or equipment connected to the Product not supplied by SSI; improper maintenance or repair of the Product; or any other reason or event not caused by SSI.

SSI HEREBY DISCLAIMS ALL OTHER WARRANTIES, WHETHER WRITTEN OR ORAL, EXPRESS OR IMPLIED BY LAW OR OTHERWISE, INCLUDING WITHOUT LIMITATION, **ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE**. CUSTOMER’S SOLE REMEDY FOR ANY DEFECTIVE PRODUCT WILL BE AS STATED ABOVE, AND IN NO EVENT WILL THE SSI BE LIABLE FOR INCIDENTAL, CONSEQUENTIAL, SPECIAL OR INDIRECT DAMAGES IN CONNECTION WITH THE PRODUCT.

This Limited Warranty shall be void if the Customer fails to comply with all of the terms set forth in this Limited Warranty. This Limited Warranty is the sole warranty offered by SSI with respect to the Product. SSI does not assume any other liability in connection with the sale of the Product. No representative of SSI is authorized to extend this Limited Warranty or to change it in any manner whatsoever. No warranty applies to any party other than the original Customer.

SSI and its directors, officers, employees, subsidiaries and affiliates shall not be liable for any damages arising from any loss of equipment, loss or distortion of data, loss of time, loss or destruction of software or other property, loss of production or profits, overhead costs, claims of third parties, labor or materials, penalties or liquidated damages or punitive damages, whatsoever, whether based upon breach of warranty, breach of contract, negligence, strict liability or any other legal theory, or other losses or expenses incurred by the Customer or any third party.

System Semiconductor’s general policy does not recommend the use of its products in life support or aircraft applications wherein a failure or malfunction of the product may directly threaten life or injury. Per System Semiconductor’s terms and conditions of sales, the user of System Semiconductor, products in life support or aircraft applications assumes all risks of such use and indemnifies System Semiconductor, against all damages.

M3000 Motor and Motion Controller Instruction Set Manual
P/N: SS-MAN-3000
Part 2 of 2

SYSTEM SEMICONDUCTOR inc.™

362 North Main Street
Marlborough CT 06447

Phone: 860-295-6170
Fax: 860-295-8318

info@systemsemi.com
www.systemsemi.com

Revision 080707