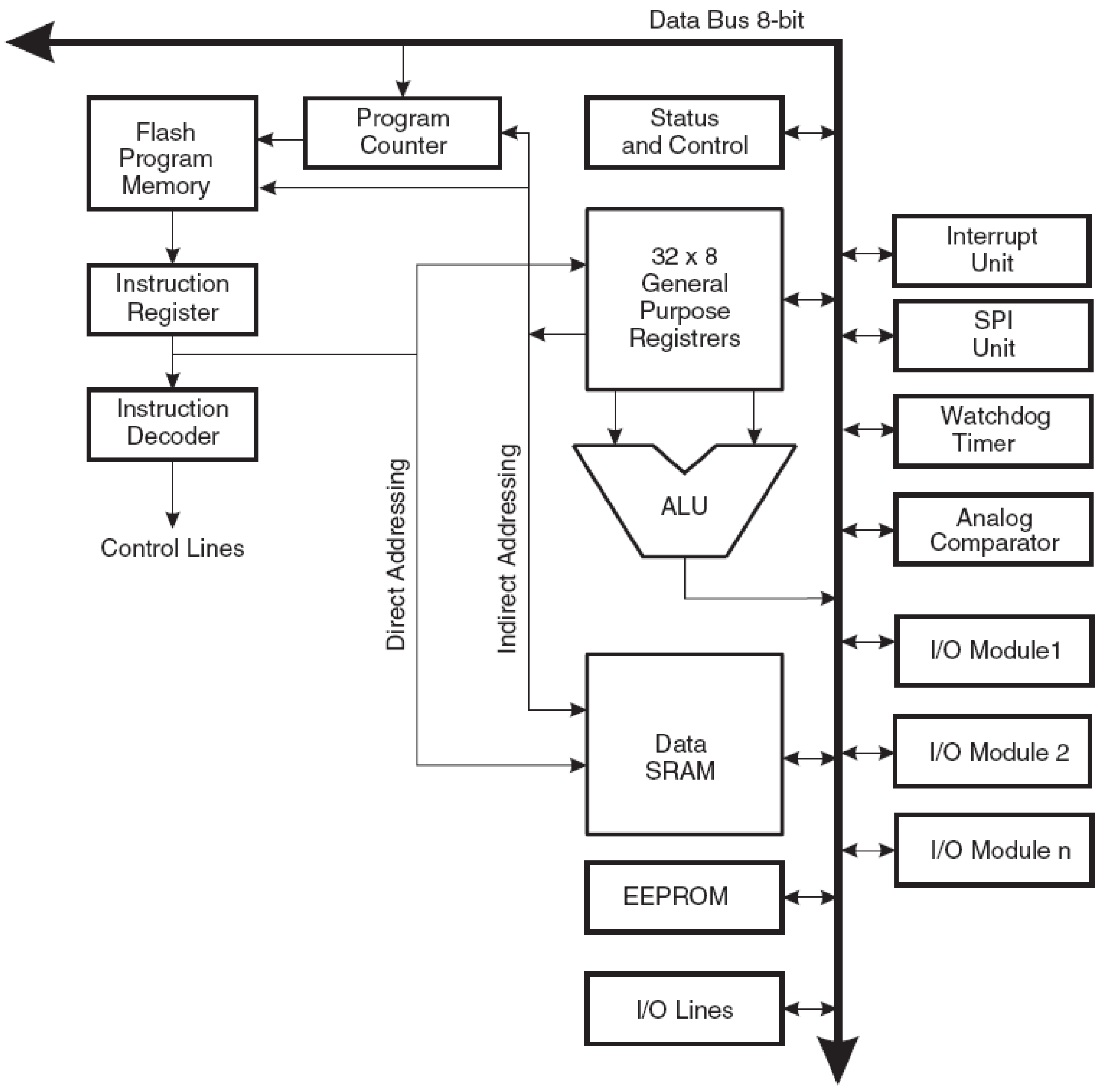
Block Diagram of the AVR Architecture



SREG – AVR Status Register

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Is useful in BCD arithmetic. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S: Sign Bit, S = N $\oplus$ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two’s Complement Overflow Flag V. See the “Instruction Set Description” for detailed information.

- **Bit 3 – V: Two’s Complement Overflow Flag**

The Two’s Complement Overflow Flag V supports two’s complement arithmetic. See the “Instruction Set Description” for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

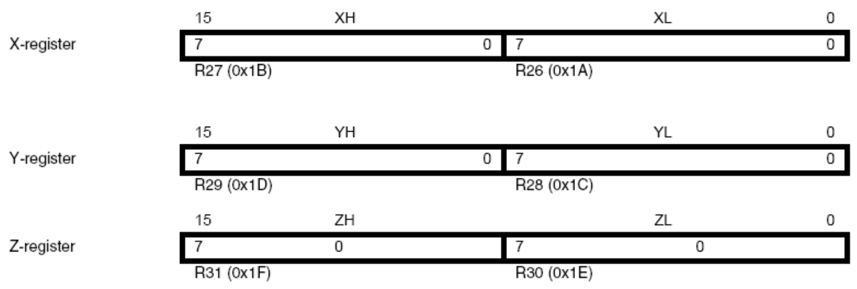
Figure 7-2. AVR CPU General Purpose Working Registers

	7 0	Addr.	
	R0	0x00	
	R1	0x01	
	R2	0x02	
	…		
	R13	0x0D	
General	R14	0x0E	
Purpose	R15	0x0F	
Working	R16	0x10	
Registers	R17	0x11	
	…		
	R26	0x1A	X-register Low Byte
	R27	0x1B	X-register High Byte
	R28	0x1C	Y-register Low Byte
	R29	0x1D	Y-register High Byte
	R30	0x1E	Z-register Low Byte
	R31	0x1F	Z-register High Byte

The X-register, Y-register, and Z-register

The registers R26...R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 7-3.

Figure 7-3. The X-, Y-, and Z-registers

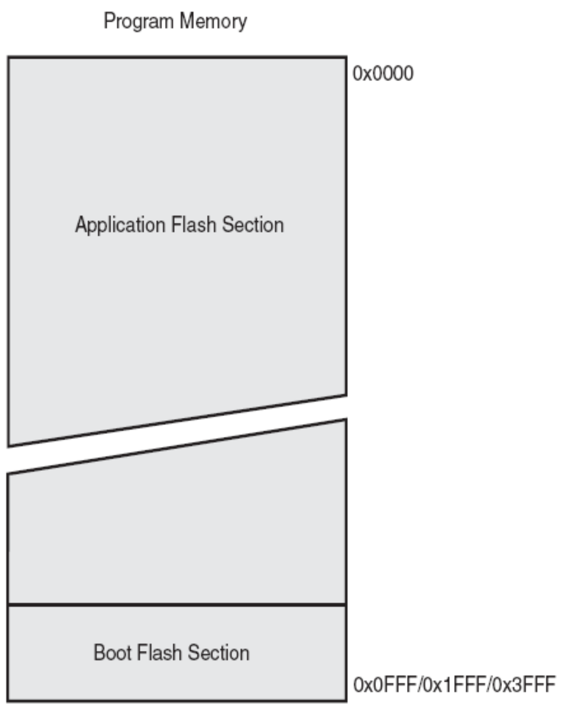


In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

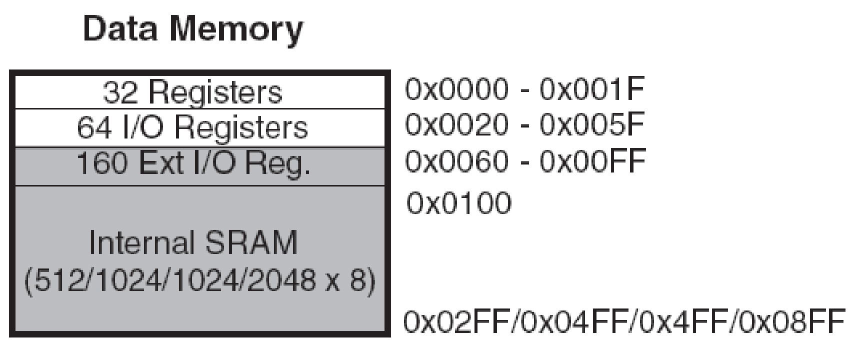
SPH and SPL – Stack Pointer High and Stack Pointer Low Register

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	**SP15**	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
0x3D (0x5D)	**SP7**	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

Program Memory Map ATmega88A, ATmega88PA, ATmega168A, ATmega168PA, ATmega328 and ATmega328P



Data Memory Map



Instruction Set Nomenclature

Status Register (SREG)

SREG:	Status Register
C:	Carry Flag
Z:	Zero Flag
N:	Negative Flag
V:	Two's complement overflow indicator
S:	$N \oplus V$, 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 data
k:	Constant address
b:	Bit in the Register File or I/O Register (3-bit)
s:	Bit in the Status Register (3-bit)
X,Y,Z:	Indirect Address Register
	(X=R27:R26, Y=R29:R28 and Z=R31:R30)
A:	I/O location address
q:	Displacement for direct addressing (6-bit)

I/O Registers

RAMPX, RAMPY, RAMPZ

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

RAMPD

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

EIND

Register concatenated with the Z-register enabling indirect jump and call to the whole program space on MCUs with more than 64K words (128K bytes) program space.

Stack

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

Flags

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

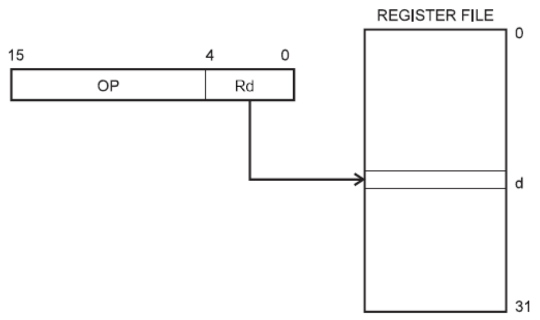
The Program and Data Addressing Modes

The AVR Enhanced RISC microcontroller supports powerful and efficient addressing modes for access to the Program memory (Flash) and Data memory (SRAM, Register file, I/O Memory, and Extended I/O Memory). This section describes the various addressing modes supported by the AVR 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. To generalize, the abstract terms RAMEND and FLASHEND have been used to represent the highest location in data and program space, respectively.

Note: Not all addressing modes are present in all devices. Refer to the device spesific instruction summary.

Register Direct, Single Register Rd

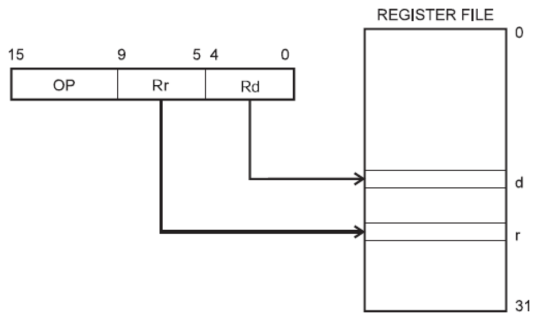
Figure 1. Direct Single Register Addressing



The operand is contained in register d (Rd).

Register Direct, Two Registers Rd and Rr

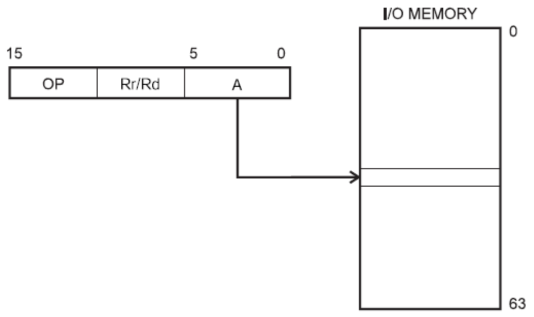
Figure 2. Direct Register Addressing, Two Registers



Operands are contained in register r (Rr) and d (Rd). The result is stored in register d (Rd).

I/O Direct

Figure 3. I/O Direct Addressing

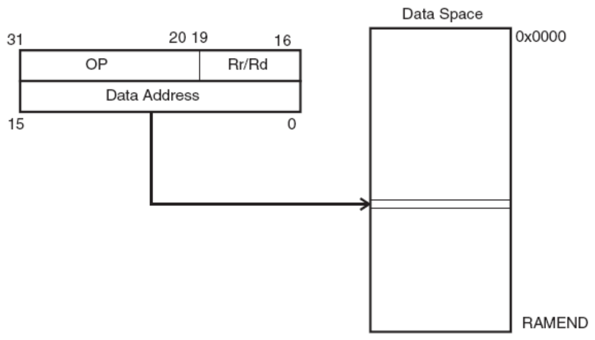


Operand address is contained in 6 bits of the instruction word. n is the destination or source register address.

Note: Some complex AVR Microcontrollers have more peripheral units than can be supported within the 64 locations reserved in the opcode for I/O direct addressing. The extended I/O memory from address 64 to 255 can only be reached by data addressing, not I/O addressing.

Data Direct

Figure 4. Direct Data Addressing



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

Data Indirect with Displacement

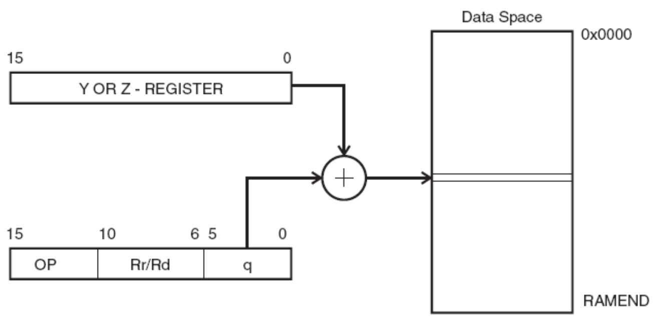


Figure 5. Data Indirect with Displacement

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

Data Indirect

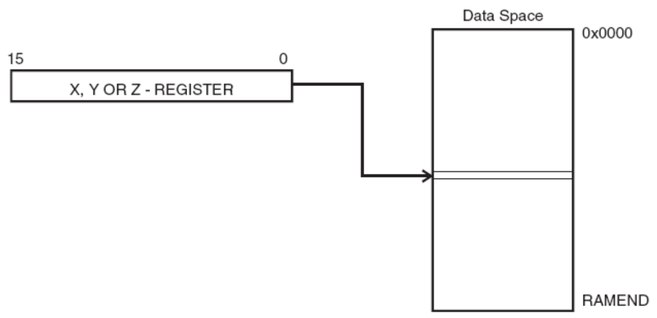


Figure 6. Data Indirect Addressing

Operand address is the contents of the X-, Y-, or the Z-register. In AVR devices without SRAM, Data Indirect Addressing is called Register Indirect Addressing. Register Indirect Addressing is a subset of Data Indirect Addressing since the data space form 0 to 31 is the Register File.

Data Indirect with Pre-decrement

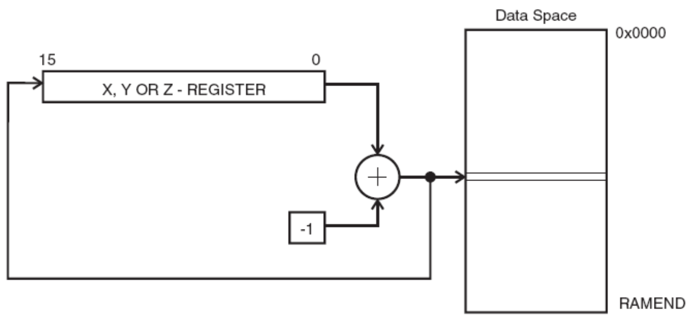


Figure 7. Data Indirect Addressing with Pre-decrement

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

Data Indirect with Post-increment

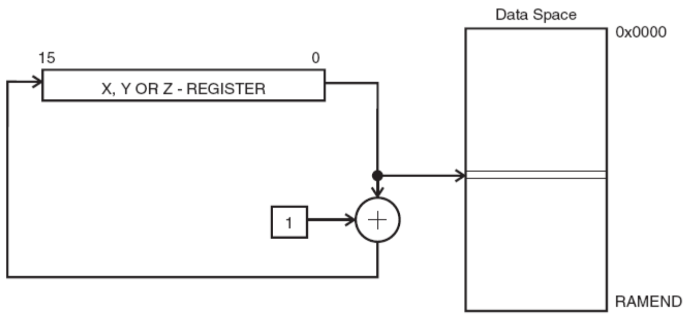
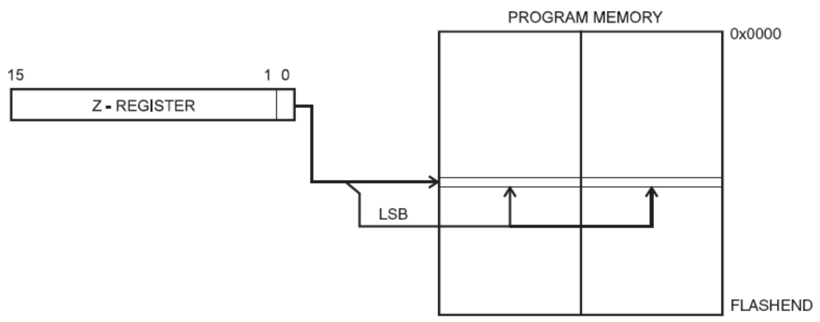


Figure 8. Data Indirect Addressing with Post-increment

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

Program Memory Constant Addressing using the LPM, ELPM, and SPM Instructions

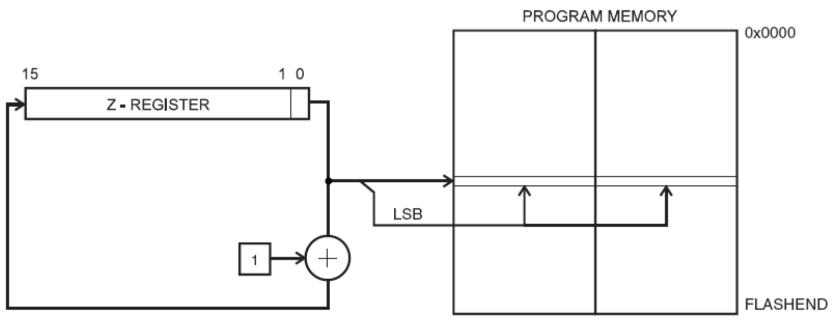
Figure 9. Program Memory Constant Addressing



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.

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

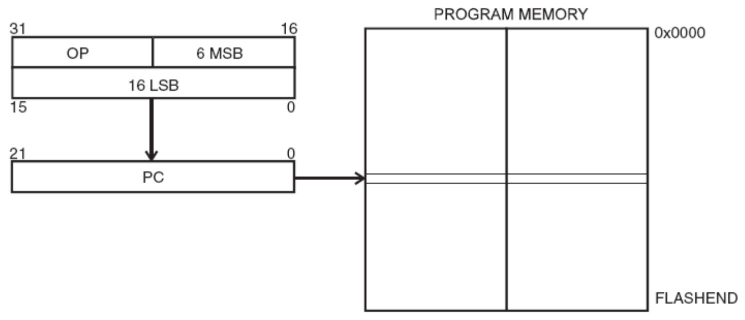
Figure 10. Program Memory Addressing with Post-increment



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.

Direct Program Addressing, JMP and CALL

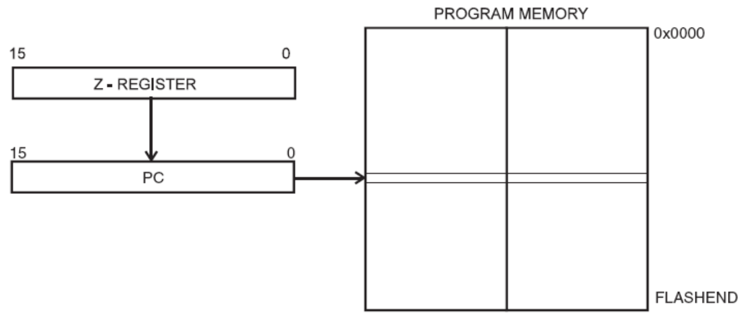
Figure 11. Direct Program Memory Addressing



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

Indirect Program Addressing, IJMP and ICALL

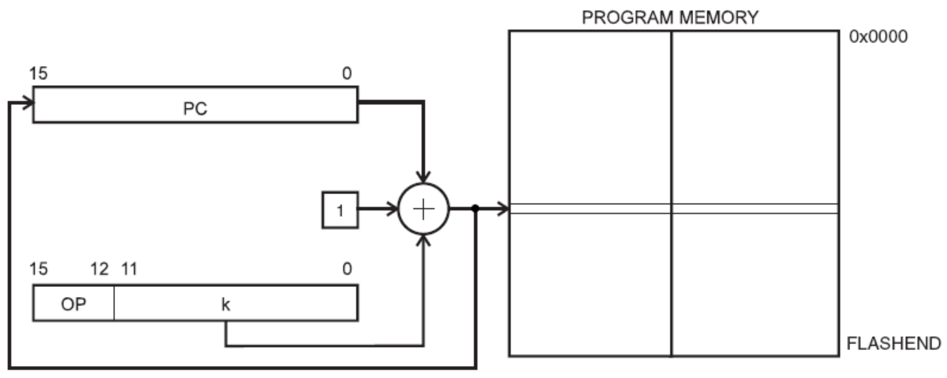
Figure 12. Indirect Program Memory Addressing



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

Relative Program Addressing, RJMP and RCALL

Figure 13. Relative Program Memory Addressing



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

Instruction Set Summary

Mnemonics	Operands	Description	Operation			Flags	#Clocks	#Clocks XMEGA
Arithmetic and Logic Instructions								
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(1)	Rd, K	Add Immediate to Word	Rd	←	Rd + 1:Rd + K	Z,C,N,V,S	2	
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(1)	Rd, K	Subtract Immediate from Word	Rd + 1:Rd	←	Rd + 1:Rd - K	Z,C,N,V,S	2	
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	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	
COM	Rd	One's Complement	Rd	←	$FF - Rd	Z,C,N,V,S	1	
NEG	Rd	Two's Complement	Rd	←	$00 - Rd	Z,C,N,V,S,H	1	
SBR	Rd,K	Set Bit(s) in Register	Rd	←	Rd v K	Z,N,V,S	1	
CBR	Rd,K	Clear Bit(s) in Register	Rd	←	Rd • ($FFh - 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	←	$FF	None	1	
MUL(1)	Rd,Rr	Multiply Unsigned	R1:R0	←	Rd x Rr (UU)	Z,C	2	
MULS(1)	Rd,Rr	Multiply Signed	R1:R0	←	Rd x Rr (SS)	Z,C	2	
MULSU(1)	Rd,Rr	Multiply Signed with Unsigned	R1:R0	←	Rd x Rr (SU)	Z,C	2	
FMUL(1)	Rd,Rr	Fractional Multiply Unsigned	R1:R0	←	Rd x Rr<<1 (UU)	Z,C	2	
FMULS(1)	Rd,Rr	Fractional Multiply Signed	R1:R0	←	Rd x Rr<<1 (SS)	Z,C	2	
FMULSU(1)	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0	←	Rd x Rr<<1 (SU)	Z,C	2	

DES	K	Data Encryption	if (H = 0) then R15:R0
			else if (H = 1) then R15:R0
				←
				←
					Encrypt(R15:R0, K)
					Decrypt(R15:R0, K)
								1/2

Branch Instructions								
RJMP	k	Relative Jump	PC	←	PC + k + 1	None	2	

IJMP(1)		Indirect Jump to (Z)	PC(15:0)
			PC(21:16)
				←
				←
					Z,
					0
						None	2	

EIJMP(1)		Extended Indirect Jump to (Z)	PC(15:0)
			PC(21:16)
				←
				←
					Z,
					EIND
						None	2	

JMP(1)	k	Jump	PC	←	k	None	3	

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3 / 4[3][5]	2 / 3[3]
ICALL[1]		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	3 / 4[3]	2 / 3[3]
EICALL[1]		Extended Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	4 [3]	3 [3]
CALL[1]	k	call Subroutine	PC ← k	None	4 / 5[3]	3 / 4[3]
RET		Subroutine Return	PC ← STACK	None	4 / 5[3]	
RETI		Interrupt Return	PC ← STACK	I	4 / 5[3]	
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	
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b) = 0) PC ← PC + 2 or 3	None	1 / 2 / 3	
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b) = 1) PC ← PC + 2 or 3	None	1 / 2 / 3	
SBIC	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	2 / 3 / 4
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	2 / 3 / 4
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	
Data Transfer Instructions						
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1	
MOVW[1]	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1	
LDI	Rd, K	Load Immediate	Rd ← K	None	1	
LDS[1]	Rd, k	Load Direct from data space	Rd ← (k)	None	1[5]/2[3]	2[3][4]
LD[2]	Rd, X	Load Indirect	Rd ← (X)	None	1[5]2[3]	1[3][4]

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
LD(2)	Rd, X+	Load Indirect and Post-Increment	Rd ← (X) X ← X + 1	None	2(3)	1(3)(4)
LD(2)	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, ← X - 1 Rd ← (X) ← (X)	None	2(3)/3(5)	2(3)(4)
LD(2)	Rd, Y	Load Indirect	Rd ← (Y) ← (Y)	None	1(5)/2(3)	1(3)(4)
LD(2)	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y) Y ← Y + 1	None	2(3)	1(3)(4)
LD(2)	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1 Rd ← (Y)	None	2(3)/3(5)	2(3)(4)
LDD(1)	Rd, Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2(3)	2(3)(4)
LD(2)	Rd, Z	Load Indirect	Rd ← (Z)	None	1(5)/2(3)	1(3)(4)
LD(2)	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2(3)	1(3)(4)
LD(2)	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2(3)/3(5)	2(3)(4)
LDD(1)	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2(3)	2(3)(4)
STS(1)	k, Rr	Store Direct to Data Space	(k) ← Rd	None	1(5)/2(3)	2(3)
ST(2)	X, Rr	Store Indirect	(X) ← Rr	None	1(5)/2(3)	1(3)
ST(2)	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	1(5)/2(3)	1(3)
ST(2)	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2(3)	2(3)
ST(2)	Y, Rr	Store Indirect	(Y) ← Rr	None	1(5)/2(3)	1(3)
ST(2)	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	1(5)/2(3)	1(3)
ST(2)	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2(3)	2(3)
STD(1)	Y+q, Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2(3)	2(3)
ST(2)	Z, Rr	Store Indirect	(Z) ← Rr	None	1(5)/2(3)	1(3)
ST(2)	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr Z ← Z + 1	None	1(5)/2(3)	1(3)
ST(2)	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1	None	2(3)	2(3)
STD(1)	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2(3)	2(3)
LPM(1)(2)		Load Program Memory	R0 ← (Z)	None	3	3
LPM(1)(2)	Rd, Z	Load Program Memory	Rd ← (Z)	None	3	3
LPM(1)(2)	Rd, Z+	Load Program Memory and Post-Increment	Rd ← (Z), Z ← Z + 1	None	3	3
ELPM(1)		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3	
ELPM(1)	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3	
ELPM(1)	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← (RAMPZ:Z), Z ← Z + 1	None	3	
SPM(1)		Store Program Memory	(RAMPZ:Z) ← R1:R0	None	-	-
SPM(1)	Z+	Store Program Memory and Post-Increment by 2	(RAMPZ:Z) ← R1:R0, Z ← Z + 2	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(1)	Rr	Push Register on Stack	STACK ← Rr	None	2	1(3)
POP(1)	Rd	Pop Register from Stack	Rd ← STACK	None	2	2(3)

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
XCH	Z, Rd	Exchange	(Z) ← Rd, Rd ← (Z)	None	1	
LAS	Z, Rd	Load and Set	(Z) ← Rd v (Z) Rd ← (Z)	None	1	
LAC	Z, Rd	Load and Clear	(Z) ← ($FF – Rd) • (Z) Rd ← (Z)	None	1	
LAT	Z, Rd	Load and Toggle	(Z) ← Rd ⊕ (Z) Rd ← (Z)	None	1	
Bit and Bit-test Instructions						
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	1(5)2	1
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	1(5)/2	1
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	
MCU Control Instructions						
BREAK(1)		Break	(See specific descr. for BREAK)	None	1	
NOP		No Operation		None	1	
SLEEP		Sleep	(see specific descr. for Sleep)	None	1	
WDR		Watchdog Reset	(see specific descr. for WDR)	None	1	

Notes:
1. This instruction is not available in all devices. Refer to the device specific instruction set summary.
2. Not all variants of this instruction are available in all devices. Refer to the device specific instruction set summary.
3. Cycle times for Data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface.
4. One extra cycle must be added when accessing Internal SRAM.
5. Number of clock cycles for Reduced Core tinyAVR.

SEZ – Set Zero Flag

Description:

Sets the Zero Flag (Z) in SREG (Status Register).

	Operation:
(i)	$Z \leftarrow 1$

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

16-bit Opcode:

1001	0100	0001	1000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	1	–

Z: 1
Zero Flag set

Example:

```
add   r2,r19    ; Add r19 to r2
sez             ; Set Zero Flag
```

Words: 1 (2 bytes)
Cycles: 1

INC – Increment

Description:

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

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

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

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

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

16-bit Opcode:

1001	010d	dddd	0011

Status Register and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	–

S: $N \oplus V$
For signed tests.

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if two's complement overflow resulted from the operation; cleared otherwise. Two's complement overflow occurs if and only if Rd was $7F before the operation.

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

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

R (Result) equals Rd after the operation.

Words: 1 (2 bytes)
Cycles: 1

MOV – Copy Register

Description:

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

	Operation:
(i)	$Rd \leftarrow Rr$

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

16-bit Opcode:

0010	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)

Cycles: 1

ADD – Add without Carry

Description:

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

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

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

16-bit Opcode:

0000	11rd	dddd	rrrr

Status Register (SREG) and 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 \$00; 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)
    add   r28,r28   ; Add r28 to itself (r28=r28+r28)
```

Words: 1 (2 bytes)

Cycles: 1

LDI – Load Immediate

Description:

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

Operation:

(i) $Rd \leftarrow K$

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

16-bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)

Cycles: 1

STS – Store Direct to Data Space

Description:

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

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

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

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

	Syntax:	**Operands:**	**Program Counter:**
(i)	STS k,Rr	$0 \le r \le 31$, $0 \le k \le 65535$	$PC \leftarrow PC + 2$

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 2 (4 bytes)
Cycles: 2

JMP – Jump

Description:

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

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

	Operation:
(i)	PC ← k

	Syntax:	**Operands:**	**Program Counter:**	**Stack:**
(i)	JMP k	$0 \leq k < 4M$	PC ← k	Unchanged

32-bit Opcode:

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and 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

LDS – Load Direct from Data Space

Description:

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

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

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

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

	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	$0 \le d \le 31$, $0 \le k \le 65535$	$PC \leftarrow PC + 2$

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 2 (4 bytes)

Cycles:	2
Cycles XMEGA:	2 If the LDS instruction is accessing internal SRAM, one extra cycle is inserted.

CLR – Clear Register

Description:

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

	Operation:
(i)	Rd ← Rd ⊕ Rd

	Syntax:	Operands:	Program Counter:
(i)	CLR Rd	$0 \le d \le 31$	PC ← PC + 1

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

0010	01dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	0	0	0	1	–

S: 0
Cleared

V: 0
Cleared

N: 0
Cleared

Z: 1
Set

R (Result) equals Rd after the operation.

Example:

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

Words: 1 (2 bytes)
Cycles: 1

CP – Compare

Description:

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

	Operation:
(i)	Rd - Rr

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

16-bit Opcode:

0001	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
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 \$00; 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

BRNE – Branch if Not Equal

Description:

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

Operation:

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

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

16-bit Opcode:

1111	01kk	kkkk	k001

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        eor     r27,r27    ; Clear r27
loop:   inc     r27        ; Increase r27
        ...
        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

CPI – Compare with Immediate

Description:

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

	Operation:
(i)	Rd - K

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

16-bit Opcode:

0011	KKKK	dddd	KKKK

Status Register (SREG) and Boolean 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: $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
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 \$00; 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

BRMI – Branch if Minus

Description:

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

Operation:

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

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

16-bit Opcode:

1111	00kk	kkkk	k010

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

ANDI – Logical AND with Immediate

Description:

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

	Operation:		
(i)	$Rd \leftarrow Rd \bullet K$		
	Syntax:	**Operands:**	**Program Counter:**
(i)	ANDI Rd,K	$16 \le d \le 31,\ 0 \le K \le 255$	$PC \leftarrow PC + 1$

16-bit Opcode:

0111	KKKK	dddd	KKKK

Status Register (SREG) and 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 \$00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

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

Words: 1 (2 bytes)
Cycles: 1

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

Operation:

(i) If Rd = Rr (Z = 1) then $PC \leftarrow PC + k + 1$, else $PC \leftarrow PC + 1$

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

16-bit Opcode:

1111	00kk	kkkk	k001

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

RJMP – Relative Jump

Description:

Relative jump to an address within PC - 2K +1 and PC + 2K (words). For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. See also JMP.

	Operation:
(i)	PC ← PC + k + 1

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

16-bit Opcode:

1100	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)
Cycles: 2

DEC – Decrement

Description:

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

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

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

	Operation:		
(i)	$Rd \leftarrow Rd - 1$		
	Syntax:	**Operands:**	**Program Counter:**
(i)	DEC Rd	$0 \le d \le 31$	$PC \leftarrow PC + 1$

16-bit Opcode:

1001	010d	dddd	1010

Status Register and 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 \$80 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 \$00; Cleared otherwise.

R (Result) equals Rd after the operation.

Example:

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

Words: 1 (2 bytes)

Cycles: 1

ST – Store Indirect From Register to Data Space using Index X

Description:

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

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

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

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

The result of these combinations is undefined:

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

Using the X-pointer:

	Operation:		**Comment:**
(i)	(X) ← Rr		X: Unchanged
(ii)	(X) ← Rr	X ← X+1	X: Post incremented
(iii)	X ← X - 1	(X) ← Rr	X: Pre decremented

	Syntax:	**Operands:**	**Program Counter:**
(i)	ST X, Rr	$0 \leq r \leq 31$	PC ← PC + 1
(ii)	ST X+, Rr	$0 \leq r \leq 31$	PC ← PC + 1
(iii)	ST -X, Rr	$0 \leq r \leq 31$	PC ← PC + 1

16-bit Opcode :

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

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)

Cycles: 2

SBRS – Skip if Bit in Register is Set

Description:

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

Operation:

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

	Syntax:	Operands:	Program Counter:
(i)	SBRS Rr,b	$0 \le r \le 31$, $0 \le b \le 7$	PC ← PC + 1, Condition false - no skip PC ← PC + 2, Skip a one word instruction PC ← PC + 3, Skip a two word instruction

16-bit Opcode:

1111	111r	rrrr	0bbb

Status Register (SREG) and 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) and the instruction skipped is 1 word
3 if condition is true (skip is executed) and the instruction skipped is 2 words

OUT – Store Register to I/O Location

Description:

Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers etc.).

Operation:

(i) I/O(A) ← Rr

	Syntax:	Operands:	Program Counter:
(i)	OUT A,Rr	$0 \le r \le 31$, $0 \le A \le 63$	PC ← PC + 1

16-bit Opcode:

1011	1AAr	rrrr	AAAA

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)
Cycles: 1

IN - Load an I/O Location to Register

Description:

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

	Operation:
(i)	Rd ← I/O(A)

	Syntax:	Operands:	Program Counter:
(i)	IN Rd,A	$0 \le d \le 31$, $0 \le A \le 63$	PC ← PC + 1

16-bit Opcode:

1011	0AAd	dddd	AAAA

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)
Cycles: 1

PUSH – Push Register on Stack

Description:

This instruction stores the contents of register Rr on the STACK. The Stack Pointer is post-decremented by 1 after the PUSH.

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

	Operation:
(i)	STACK ← Rr

	Syntax:	**Operands:**	**Program Counter:**	**Stack:**
(i)	PUSH Rr	$0 \le r \le 31$	PC ← PC + 1	SP ← SP - 1

16-bit Opcode:

1001	001d	dddd	1111

Status Register (SREG) and 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

POP – Pop Register from Stack

Description:

This instruction loads register Rd with a byte from the STACK. The Stack Pointer is pre-incremented by 1 before the POP.

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

	Operation:
(i)	Rd ← STACK

	Syntax:	**Operands:**	**Program Counter:**	**Stack:**
(i)	POP Rd	$0 \le d \le 31$	PC ← PC + 1	SP ← SP + 1

16-bit Opcode:

1001	000d	dddd	1111

Status Register (SREG) and 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

CALL – Long Call to a Subroutine

Description:

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

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

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

	Syntax:	**Operands:**	**Program Counter**	**Stack:**

(i)	CALL k	$0 \le k < 64K$	PC ← k	STACK ← PC+2
				SP ← SP-2, (2 bytes, 16 bits)

(ii)	CALL k	$0 \le k < 4M$	PC ← k	STACK ← PC+2
				SP ← SP-3 (3 bytes, 22 bits)

32-bit Opcode:

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words :	2 (4 bytes)
Cycles :	4, devices with 16 bit PC 5, devices with 22 bit PC

RET – Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment scheme during RET.

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

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

16-bit Opcode:

1001	0101	0000	1000

Status Register (SREG) and 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 devices with 16-bit PC
5 devices with 22-bit PC

LD (LDD) – Load Indirect From Data Space to Register using Index Z

Description:

Loads one byte indirect with or without displacement from the data space to a register. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. In some parts the Flash Memory has been mapped to the data space and can be read using this command. The EEPROM has a separate address space.

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

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

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

In the Reduced Core tinyAVR the LD instruction can be used to achieve the same operation as LPM since the program memory is mapped to the data memory space.

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

The result of these combinations is undefined:

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

Using the Z-pointer:

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

	Syntax:	**Operands:**	**Program Counter:**
(i)	LD Rd, Z	$0 \le d \le 31$	PC ← PC + 1
(ii)	LD Rd, Z+	$0 \le d \le 31$	PC ← PC + 1
(iii)	LD Rd, -Z	$0 \le d \le 31$	PC ← PC + 1
(iv)	LDD Rd, Z+q	$0 \le d \le 31$, $0 \le q \le 63$	PC ← PC + 1

16-bit Opcode:

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

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

Words: 1 (2 bytes)

Cycles:	(i) 1(2)
	(ii) 2
	(iii) 3(2)
Cycles XMEGA:	(i) 1(1)
	(ii) 1(1)
	(iii) 2(1)
	(iv) 2(1)

Notes:
1. IF the LD instruction is accessing internal SRAM, one extra cycle is inserted.
2. LD instruction can load data from program memory since the flash is memory mapped. Loading data from the data memory takes 1 clock cycle, and loading from the program memory takes 2 clock cycles. But if an interrupt occur (before the last clock cycle) no additional clock cycles is necessary when loading from the program memory. Hence, the instruction takes only 1 clock cycle to execute.

   LD instruction with pre-decrement can load data from program memory since the flash is memory mapped. Loading data from the data memory takes 2 clock cycles, and loading from the program memory takes 3 clock cycles. But if an interrupt occur (before the last clock cycle) no additional clock cycles is necessary when loading from the program memory. Hence, the instruction takes only 1 clock cycle to execute.

BRSH – Branch if Same or Higher (Unsigned)

Description:

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

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

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

16-bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

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

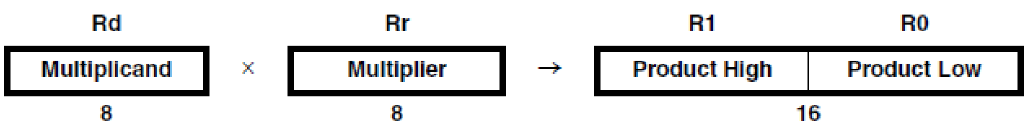
Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

MUL – Multiply Unsigned

Description:

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



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

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

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

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

16-bit Opcode:

1001	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C: R15
Set if bit 15 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 \$0000; cleared otherwise.

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

Example:

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

Words: 1 (2 bytes)
Cycles: 2