5. Instruction Set Summary

Several updates of the AVR CPU during its lifetime has resulted in different flavors of the instruction set, especially for the timing of the instructions. Machine code level of compatibility is intact for all CPU versions with very few exceptions related to the Reduced Core (AVRrc), though not all instructions are included in the instruction set for all devices. The table below contains the major versions of the AVR 8-bit CPUs. In addition to the different versions, there are differences depending on the size of the device memory map. Typically these differences are handled by a C/EC++ compiler, but users that are porting code should be aware that the code execution can vary slightly in the number of clock cycles.

Table 5-1. Versions of AVR® 8-bit CPU

Name	Description
AVR	Original instruction set from 1995
AVRe	AVR instruction set extended with the Move Word (MOVW) instruction, and the Load Program Memory (LPM) instruction has been enhanced. Same timing as AVR.
AVRe+	AVRe instruction set extended with the Multiply (xMULxx) instructions, and if applicable with the extended range instructions EICALL, EIJMP and ELPM. Same timing as AVR and AVRe. Thus, tables listing number of clock cycles do not distiguish between AVRe and AVRe+, and use AVRe to represent both.
AVRxm	AVRe+ instruction set extended with the Read Modify Write (RMW) and Data Encryption Standard (DES) instructions. SPM extended to include SPM Z+2. Significantly different timing compared to AVR, AVRe, AVRe+.
AVRxt	A combination of AVRe+ and AVRxm. Available instructions are the same as AVRe+, but the timing has been improved compared to AVR, AVRe, AVRe+ and AVRxm.
AVRrc	AVRrc has only 16 registers in its register file (R31-R16), and the instruction set is reduced. The timing is significantly different compared to the AVR, AVRe, AVRe+, AVRxm and AVRxt. Refer to the instruction set summary for further details.

Table 5-2. Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1	1	1	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1	1	1	1
ADIW	Rd, K	Add Immediate to Word	R[d + 1]:Rd ← R[d + 1]:Rd + K	Z,C,N,V,S	2	2	2	N/A
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1	1	1	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1	1	1	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1	1	1	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1	1	1	1
SBIW	Rd, K	Subtract Immediate from Word	R[d + 1]:Rd ← R[d + 1]:Rd - K	Z,C,N,V,S	2	2	2	N/A
AND	Rd, Rr	Logical AND	Rd ← Rd ∧ Rr	Z,N,V,S	1	1	1	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd ∧ K	Z,N,V,S	1	1	1	1
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1	1	1	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1	1	1	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1	1	1	1

.....continued

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
COM	Rd	One's Complement	Rd ← 0xFF - Rd	Z,C,N,V,S	1	1	1	1
NEG	Rd	Two's Complement	Rd ← 0x00 - Rd	Z,C,N,V,S,H	1	1	1	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1	1	1	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd ∧ (0xFFh - K)	Z,N,V,S	1	1	1	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1	1	1	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1	1	1	1
TST	Rd	Test for Zero or Minus	Rd ← Rd ∧ Rd	Z,N,V,S	1	1	1	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1	1	1	1
SER	Rd	Set Register	Rd ← 0xFF	None	1	1	1	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd x Rr (UU)	Z,C	2	2	2	N/A
MULS	Rd,Rr	Multiply Signed	R1:R0 ← Rd x Rr (SS)	Z,C	2	2	2	N/A
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr (SU)	Z,C	2	2	2	N/A
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 ← Rd x Rr<<1 (UU)	Z,C	2	2	2	N/A
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 ← Rd x Rr<<1 (SS)	Z,C	2	2	2	N/A
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← Rd x Rr<<1 (SU)	Z,C	2	2	2	N/A
DES	K	Data Encryption	if (H == 0), R15:R0 ← Encrypt(R15:R0, K) if (H == 1), R15:R0 ← Decrypt(R15:R0, K)		N/A	1 / 2	N/A	N/A

Table 5-3. Change of Flow Instructions

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2	2	2	2
IJMP		Indirect Jump to (Z)	PC(15:0) ← Z PC(21:16) ← 0	None	2	2	2	2
EIJMP		Extended Indirect Jump to (Z)	PC(15:0) ← Z PC(21:16) ← EIND	None	2	2	2	N/A
JMP	k	Jump	PC ← k	None	3	3	3	N/A
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3 / 4(1)	2 / 3(1)	2 / 3	3
ICALL		Indirect Call to (Z)	PC(15:0) ← Z PC(21:16) ← 0	None	3 / 4(1)	2 / 3(1)	2 / 3	3
EICALL		Extended Indirect Call to (Z)	PC(15:0) ← Z PC(21:16) ← EIND	None	4(1)	3(1)	3	N/A
CALL	k	Call Subroutine	PC ← k	None	4 / 5(1)	3/ 4(1)	3 /4	N/A
RET		Subroutine Return	PC ← STACK	None	4 / 5(1)	4 / 5(1)	4 / 5	6
RETI		Interrupt Return	PC ← STACK	I	4 / 5(1)	4 / 5(1)	4 / 5	6
CPSE	Rd,Rr	Compare, skip if Equal	if (Rd == Rr) PC ← PC + 2 or 3	None	1 / 2 / 3	1 / 2 / 3	1 / 2 / 3	1 / 2
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,S,H	1	1	1	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,S,H	1	1	1	1

.....continued

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,S,H	1	1	1	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b) == 0) PC ← PC + 2 or 3	None	1 / 2 / 3	1 / 2 / 3	1 / 2 / 3	1 / 2
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b) == 1) PC ← PC + 2 or 3	None	1 / 2 / 3	1 / 2 / 3	1 / 2 / 3	1 / 2
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	1 / 2 / 3	1 / 2
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	1 / 2 / 3	1 / 2
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BREQ	k	Branch if Equal	if (Z == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRNE	k	Branch if Not Equal	if (Z == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRCS	k	Branch if Carry Set	if (C == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRCC	k	Branch if Carry Cleared	if (C == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRSH	k	Branch if Same or Higher	if (C == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRLO	k	Branch if Lower	if (C == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRMI	k	Branch if Minus	if (N == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRPL	k	Branch if Plus	if (N == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (S == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 /2
BRLT	k	Branch if Less Than, Signed	if (S == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H == 1) then PC ← PC + k + 1	None	1 / 2	1 /2	1 / 2	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRTS	k	Branch if T Bit Set	if (T == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRTC	k	Branch if T Bit Cleared	if (T == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I == 1) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I == 0) then PC ← PC + k + 1	None	1 / 2	1 / 2	1 / 2	1 / 2

Table 5-4. Data Transfer Instructions

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1	1	1	1
MOVW	Rd, Rr	Copy Register Pair	R[d + 1]:Rd ← R[r + 1]:Rr	None	1	1	1	N/A
LDI	Rd, K	Load Immediate	Rd ← K	None	1	1	1	1
LDS	Rd, k	Load Direct from Data Space	Rd ← DS(k)	None	2(1)	3(1)(3)	3(2)	2
LD	Rd, X	Load Indirect	Rd ← DS(X)	None	2(1)	2(1)(3)	2(2)	1 / 2
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← DS(X) X ← X + 1	None	2(1)	2(1)(3)	2(2)	2 / 3
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1 Rd ← DS(X)	None	2(1)	3(1)(3)	2(2)	2 / 3

.....continued

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
LD	Rd, Y	Load Indirect	Rd ← DS(Y)	None	2(1)	2(1)(3)	2(2)	1 / 2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← DS(Y) Y ← Y + 1	None	2(1)	2(1)(3)	2(2)	2 / 3
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1 Rd ← DS(Y)	None	2(1)	3(1)(3)	2(2)	2 / 3
LDD	Rd, Y+q	Load Indirect with Displacement	Rd ← DS(Y + q)	None	2(1)	3(1)(3)	2(2)	N/A
LD	Rd, Z	Load Indirect	Rd ← DS(Z)	None	2(1)	2(1)(3)	2(2)	1 / 2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← DS(Z) Z ← Z+1	None	2(1)	2(1)(3)	2(2)	2 / 3
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1 Rd ← DS(Z)	None	2(1)	3(1)(3)	2(2)	2 / 3
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← DS(Z + q)	None	2(1)	3(1)(3)	2(2)	N/A
STS	k, Rr	Store Direct to Data Space	DS(k) ← Rd	None	2(1)	2(1)	2(2)	1
ST	X, Rr	Store Indirect	DS(X) ← Rr	None	2(1)	1(1)	1(2)	1
ST	X+, Rr	Store Indirect and Post-Increment	DS(X) ← Rr X ← X + 1	None	2(1)	1(1)	1(2)	1
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1 DS(X) ← Rr	None	2(1)	2(1)	1(2)	2
ST	Y, Rr	Store Indirect	DS(Y) ← Rr	None	2(1)	1(1)	1(2)	1
ST	Y+, Rr	Store Indirect and Post-Increment	DS(Y) ← Rr Y ← Y + 1	None	2(1)	1(1)	1(2)	1
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1 DS(Y) ← Rr	None	2(1)	2(1)	1(2)	2
STD	Y+q, Rr	Store Indirect with Displacement	DS(Y + q) ← Rr	None	2(1)	2(1)	1(2)	N/A
ST	Z, Rr	Store Indirect	DS(Z) ← Rr	None	2(1)	1(1)	1(2)	1
ST	Z+, Rr	Store Indirect and Post-Increment	DS(Z) ← Rr Z ← Z + 1	None	2(1)	1(1)	1(2)	1
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1 DS(Z) ← Rr	None	2(1)	2(1)	1(2)	2
STD	Z+q,Rr	Store Indirect with Displacement	DS(Z + q) ← Rr	None	2(1)	2(1)	1(2)	N/A
LPM		Load Program Memory	R0 ← PS(Z)	None	3	3	3	N/A
LPM	Rd, Z	Load Program Memory	Rd ← PS(Z)	None	3	3	3	N/A
LPM	Rd, Z+	Load Program Memory and Post-Increment	Rd ← PS(Z) Z ← Z + 1	None	3	3	3	N/A
ELPM		Extended Load Program Memory	R0 ← PS(RAMPZ:Z)	None	3	3	3	N/A
ELPM	Rd, Z	Extended Load Program Memory	Rd ← PS(RAMPZ:Z)	None	3	3	3	N/A
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← PS(RAMPZ:Z) (RAMPZ:Z) ← (RAMPZ:Z) + 1	None	3	3	3	N/A
SPM		Store Program Memory	PS(RAMPZ:Z) ← R1:R0	None	-(4)	-(4)	-(4)	N/A
SPM	Z+	Store Program Memory and Post-Increment by 2	PS(RAMPZ:Z) ← R1:R0 Z ← Z + 2	None	N/A	-(4)	-(4)	N/A

.....continued

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1	1	1	1
OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1	1	1	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2	1(1)	1	1
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2	2(1)	2	3
XCH	Z, Rd	Exchange	DS(Z) ↔ Rd	None	N/A	2	N/A	N/A
LAS	Z, Rd	Load and Set	DS(Z) ← Rd v DS(Z) Rd ← DS(Z)	None	N/A	2	N/A	N/A
LAC	Z, Rd	Load and Clear	DS(Z) ← (0xFF – Rd) ∧ DS(Z) Rd ← DS(Z)	None	N/A	2	N/A	N/A
LAT	Z, Rd	Load and Toggle	DS(Z) ← Rd ⊕ DS(Z) Rd ← DS(Z)	None	N/A	2	N/A	N/A

Table 5-5. Bit and Bit-Test Instructions

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
LSL	Rd	Logical Shift Left	C ← Rd(7) Rd(n+1) ← Rd(n), n=6...0 Rd(0) ← 0	Z,C,N,V,H	1	1	1	1
LSR	Rd	Logical Shift Right	C ← Rd(0) Rd(n) ← Rd(n+1), n=0...6 Rd(7) ← 0	Z,C,N,V	1	1	1	1
ROL	Rd	Rotate Left Through Carry	temp ← C C ← Rd(7) Rd(n+1) ← Rd(n), n=6...0 Rd(0) ← temp	Z,C,N,V,H	1	1	1	1
ROR	Rd	Rotate Right Through Carry	temp ← C C ← Rd(0) Rd(n) ← Rd(n+1), n=0...6 Rd(7) ← temp	Z,C,N,V	1	1	1	1
ASR	Rd	Arithmetic Shift Right	C ← Rd(0) Rd(n) ← Rd(n+1), n=0..6 Rd(7) ← Rd(7)	Z,C,N,V	1	1	1	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ↔ Rd(7..4)	None	1	1	1	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	2	1	1	1
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	2	1	1	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1	1	1	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1	1	1	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1	1	1	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1	1	1	1
SEC		Set Carry	C ← 1	C	1	1	1	1
CLC		Clear Carry	C ← 0	C	1	1	1	1
SEN		Set Negative Flag	N ← 1	N	1	1	1	1

.....continued

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
CLN		Clear Negative Flag	N ← 0	N	1	1	1	1
SEZ		Set Zero Flag	Z ← 1	Z	1	1	1	1
CLZ		Clear Zero Flag	Z ← 0	Z	1	1	1	1
SEI		Global Interrupt Enable	I ← 1	I	1	1	1	1
CLI		Global Interrupt Disable	I ← 0	I	1	1	1	1
SES		Set Sign Bit	S ← 1	S	1	1	1	1
CLS		Clear Sign Bit	S ← 0	S	1	1	1	1
SEV		Set Two's Complement Overflow	V ← 1	V	1	1	1	1
CLV		Clear Two's Complement Overflow	V ← 0	V	1	1	1	1
SET		Set T in SREG	T ← 1	T	1	1	1	1
CLT		Clear T in SREG	T ← 0	T	1	1	1	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1	1	1	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1	1	1	1

Table 5-6. MCU Control Instructions

Mnemonic	Operands	Description	Operation	Flags	#Clocks AVRe	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
BREAK		Break	See the debug interface description	None	1	1	1	1
NOP		No Operation		None	1	1	1	1
SLEEP		Sleep	See the power management and sleep description	None	1	1	1	1
WDR		Watchdog Reset	See the Watchdog Controller description	None	1	1	1	1

Notes:

1. Cycle times for data memory access assume internal RAM access and are not valid for accessing external RAM.
2. Cycle time for data memory access assumes internal RAM access, and are not valid for access to NVM. A minimum of one extra cycle must be added when accessing NVM. The additional time varies dependent on the NVM module implementation. See the NVMCTRL section in the specific devices data sheet for more information.
3. If the LD instruction is accessing I/O Registers, one cycle can be deducted.
4. Varies with the programming time of the device.