CSC 230 - Summer 2018 - AVR Instruction Quick Reference

Instruction	Description	Operation	Flags
Arithmetic Operations			
`ADC Rd, Rr`	Add with carry	Rd ← Rd + Rr + C	ZCNVSH
`ADD Rd, Rr`	Add without carry	Rd ← Rd + Rr	ZCNVSH
`DEC Rd`	Decrement (no carry)	Rd ← Rd - 1	ZNVS
`INC Rd`	Increment (no carry)	Rd ← Rd + 1	ZNVS
`NEG Rd`	Negate	Rd ← –Rd	ZCNVSH
`SBC Rd, Rr`	Subtract with carry	Rd ← Rd - Rr - C	ZCNVSH
`SUB Rd, Rr`	Subtract without carry	Rd ← Rd - Rr	ZCNVSH
`SUBI Rd, K`	Subtract (immediate)	Rd ← Rd - K	ZCNVSH
Wide Arithmetic Operations (Rd must be R24, R26, R28 or R30)			
`ADIW Rd, K`	Add immed. (wide)	Rd+1:Rd ← Rd+1:Rd + K	ZCNVS
`SBIW Rd, K`	Subtract immed. (wide)	Rd+1:Rd ← Rd+1:Rd - K	ZCNVSH
Bitwise Operations			
`AND Rd, Rr`	Bitwise AND	Rd ← Rd `&` Rr	ZNVS
`ANDI Rd, K`	AND (immediate)	Rd ← Rd `&` K	ZNVS
`ASR Rd`	Arithmetic shift right	Rd ← ((`signed`)Rd)`>>`1	ZCNV
`COM Rd`	Bitwise complement	Rd ← ∼Rd	ZCNVS
`EOR Rd, Rr`	Bitwise XOR	Rd ← Rd ⊕ Rr	ZNVS
`LSL Rd`	Logical shift left	Rd ← Rd `<<` 1	ZCNVH
`LSR Rd`	Logical shift right	Rd ← ((`unsigned`)Rd)`>>`1	ZCNV
`OR Rd, Rr`	Bitwise OR	Rd ← Rd `|` Rr	ZNVS
`ORI Rd, K`	OR (immediate)	Rd ← Rd `|` K	ZNVS
`ROL Rd`	Rotate left with carry	Rd ← (Rd`<<`1)`|`C	ZCNVH
`ROR Rd`	Rotate right with carry	Rd ← (Rd`>>`1)`|`(C`<<`7)	ZCNV
`SWAP Rd`	Swap half-bytes	Rd(3:0) ↔ Rd(7:4)	-
Misc. Operations			
`CLR Rd`	Clear register	Rd ← 0	ZNVS
`MOV Rd, Rr`	Copy register	Rd ← Rr	-
`MOVW Rd, Rr`	Copy register pair	Rd+1:Rd ← Rr+1:Rr	-
`NOP`	No operation	No effect	-
`POP Rd`	Pop stack into register	Rd ← STACK	-
`PUSH Rr`	Push register onto stack	STACK ← Rr	-
`SER Rd`	Set to 11 . . . 1	Rd ← 0xFF	-

Instruction	Description	Operation	Flags
Comparisons			
`CP Rd, Rr`	Compare	Set flags based on Rd - Rr	ZCNVSH
`CPC Rd, Rr`	Compare with carry	Set flags based on Rd - Rr - C	ZCNVSH
`CPI Rd, K`	Compare (immediate)	Set flags based on Rd - K	ZCNVSH
`CPSE Rd, Rr`	Compare & skip if equal	Skip next inst. if Rd = Rr	-
`TST Rd`	Test if zero	Set Z if Rd = 0	ZNVS
Unconditional Branch and Skip Instructions			
`IJMP`	Indirect jump to (Z)	PC ← Z	-
`JMP K`	Absolute Jump	PC ← K	-
`RJMP K`	Relative jump	PC ← PC + K + 1	-
`SBRC Rd, b`	Skip if bit cleared	Skip next inst. if Rd(b) = 0	-
`SBRS Rd, b`	Skip if bit set	Skip next inst. if Rd(b) = 1	-
Call and Return Instructions			
All CALL instructions push PC onto the stack before changing PC.			
`CALL K`	Call subroutine	PC ← K	-
`ICALL`	Call (indirect)	PC ← Z (R31:R30)	-
`RCALL K`	Call (relative)	PC ← PC + K + 1	-
`RET`	Subroutine return	PC ← STACK	-
`RETI`	Interrupt return	PC ← STACK	I

Operand Registers

R0 - R31: General Purpose 8-bit registers
Instructions with immediate operands (and SER) can only use R16 - R31,
and some multiplication instructions can only use R16 - R23.
X: 16-bit pair of R27:R26 (XH:XL)
Y: 16-bit pair of R29:R28 (YH:YL)
Z: 16-bit pair of R31:R30 (ZH:ZL)

Status Register (SREG)

C: Carry Flag	(Set/clear with SEC/CLC instructions)
Z: Zero Flag	(Set/clear with SEZ/CLZ instructions)
N: Negative Flag	(Set/clear with SEN/CLN instructions)
V: Two's-Complement Overflow Flag	(Set/clear with SEV/CLV instructions)
H: Halfbyte Carry Flag	(Set/clear with SEH/CLH instructions)
I: Global Interrupt Enable Flag	(Set/clear with SEI/CLI instructions)

ver. 2018.5.13.BB

CSC 230 - Summer 2018 - AVR Instruction Quick Reference

Instruction	Description	Condition
Conditional Branches		
`BRNE K`	Branch if not equal	Z = 0
`BREQ K`	Branch if equal	Z = 1
`BRCC K`	Branch if carry cleared	C = 0
`BRCS K`	Branch if carry set	C = 1
`BRSH K`	Branch if same or higher	C = 0
`BRLO K`	Branch if lower	C = 1
`BRPL K`	Branch if non-negative	N = 0
`BRMI K`	Branch if negative	N = 1
`BRGE K`	Branch if greater or equal (signed)	N $\oplus$ V = 0
`BRLT K`	Branch if less than (signed)	N $\oplus$ V = 1
`BRHC K`	Branch if half carry cleared	H = 0
`BRHS K`	Branch if half carry set	H = 1
`BRVC K`	Branch if overflow flag cleared	V = 0
`BRVS K`	Branch if overflow flag set	V = 1
`BRID K`	Branch if interrupt disabled	I = 0
`BRIE K`	Branch if interrupt enabled	I = 1
Conditional branch instructions take a relative 7-bit signed offset K as an operand, and, if the branch condition is met, set PC $\leftarrow$ PC + K + 1.		
Conditional branch instructions do not modify any flags.		
Load and store instructions have various encodings for the different addressing modes. In the table on the right, values in brackets refer to the contents of memory at that address.		
The `LPM` instruction is used to read from program memory. `LPM Rd, Z` and `LPM Rd, Z+` are the only encodings available, and behave like `LD Rd, Z` and `LD Rd, Z+`, respectively.		

Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex	Dec
0000	0x0	0	0100	0x4	4	1000	0x8	8	1100	0xC	12
0001	0x1	1	0101	0x5	5	1001	0x9	9	1101	0xD	13
0010	0x2	2	0110	0x6	6	1010	0xA	10	1110	0xE	14
0011	0x3	3	0111	0x7	7	1011	0xB	11	1111	0xF	15

Instruction	Description	Operation
Load and Store Instructions		
Immediate Operand		
The value K is an 8-bit constant. Rd must be in the range R16 - R31.		
`LDI Rd, K`	Load immediate	Rd $\leftarrow$ K
Direct from data memory		
The value K is a 16-bit address.		
`LDS Rd, K`	Load direct	Rd $\leftarrow$ (K)
`STS K, Rr`	Store direct	(K) $\leftarrow$ Rr
Indirect via register X, Y or Z		
The examples below use X, but Y or Z can also be used.		
`LD Rd, X`	Load indirect via X	Rd $\leftarrow$ (X)
`ST X, Rr`	Store indirect via X	(X) $\leftarrow$ Rr
Indirect with pre-decrement		
In each of the encodings below, the register (X, Y or Z) is decremented **before** the load/store operation. The examples below use X, but Y or Z can also be used.		
`LD Rd, -X`	Load (pre-decrement)	X $\leftarrow$ X - 1, Rd $\leftarrow$ (X)
`ST -X, Rr`	Store (pre-decrement)	X $\leftarrow$ X - 1, (X) $\leftarrow$ Rr
Indirect with post-increment		
In each of the encodings below, the register (X, Y or Z) is incremented **after** the load/store operation. The examples below use X, but Y or Z can also be used.		
`LD Rd, X+`	Load (post-increment)	Rd $\leftarrow$ (X), X $\leftarrow$ X + 1
`ST X+, Rr`	Store (post-increment)	(X) $\leftarrow$ Rr, X $\leftarrow$ X + 1
Indirect with displacement		
Only Y and Z can be used for these instructions. The value q is a 6-bit constant.		
`LDD Rd,Y+q`	Load (displacement)	Rd $\leftarrow$ (Y+q)
`STD Y+q,Rr`	Store (displacement)	(Y+q) $\leftarrow$ Rr

ver. 2018.5.13.BB