GPRs	Kodierung	Beschreibung
r0	000	General Purpose Register
r1	001	General Purpose Register
r2	010	General Purpose Register
r3	011	General Purpose Register
r4	100	General Purpose Register
r5	101	General Purpose Register
r6	110	General Purpose Register. Aber: bei P=1 in CSR werden die 4 LSBs in PR geschrieben
r7	111	General Purpose Register. Aber: r7 bildet bei load/store die 8 LSBs der Adresse

Special Register

PC	11 Bit	Programm Counter
IR	16 Bit	Instruction Register
SCR	8 Bit	Status and Control Register (8 Steuer/Zustandsbits)
SP	10 Bit	Stack Pointer (für Stackzugriffe)
PR	4 Bit	Page Register (4 MSBs für 12 Bit Speicherzugriffe)

Status and Control Register

C (Carry)	0	Statusbit: Carry Bit der letzten 8 Bit Operation gesetzt
V (oVerflow)	1	Statusbit: Letzte Operation hat ein Overflow ausgelöst (signed)
N (Negative)	2	Statusbit: Vorzeichenbit der letzten Operation
S (Signed)	3	Statusbit: Vorzeichenbit des korrekten Ergebnisses der letzten Operation
Z (Zero)	4	Statusbis: Letztes Ergebnis ist 0
P (write Page select)	5	Steuerbit: Schreibe r6 4 LSBs in PR im nächsten Takt. Bit wird automatisch cleared
	6	
	7	

Maske für Cond	Zugriff auf einzelne Statusbits bzw. Vergleichsoperatoren				
000	C		S=any	aus SCR	
001	V		S=any	aus SCR	
010	N		S=any	aus SCR	
011	S		S=any	aus SCR	
100	Z		S=any	aus SCR	
101	==	inv: <>	S=any	Z==1 unsigned und signed	
110	<	inv: >=	S=0	C==1	siehe ATMega ISA S.21
111	>	inv: <=	S=0	C or Z == 0	siehe ATMega ISA S.21
110	<	inv: >=	S=1	S == 1	siehe ATMega ISA S.21
111	>	inv: <=	S=1	S and Z == 0	siehe ATMega ISA S.21

Speicher	Adressbreite	Datenbreite	Gesamtspeicher
Programmspeicher	11	16 Bit	4 kB
Datenspeicher	12	8 Bit	4 kB

Datenspeicher Adressierung über 8 Bit im Register / im Befehl und weitere 4 Bit im Page-Register. Das Page-Register wird über das P-Bit im CR aktiviert. Page-Register: 4 LSBs von r6
Programmspeicher Adressierung über 10 Bit im PC. Bei jpa 10 Bit direkt im Befehl, bei brbs, brbc und bra wird ein relativer Offset von 5 Bit (max +-32) angegeben

Addressierung

Programmspeicher:

A: 11 Bit PC im normalen Zugriffverfahren bei Programmablauf
B: 11 Bit direkt im Befehl kodiert (jpr, jsr). Diese werden in den PC geladen

C: 5 Bit Offset im Befehl kodiert, der zum aktuellen PC addiert wird (signed)

Datenspeicher:

A: 8 Bit direkt im Befehl kodiert (st/ld). Diese bilden die 8 LSBs, PR bildet die 4 MSBs

B: 8 Bit aus r6 bilden die 8 LSBs, PR bildet die 4 MSBs (bei stz/ldz)

Befehlsgruppe	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Arithmetisc-logische Befehle	**Opcode**					**Rd**			**X**	**X**	**X**	**X**	**X**	**Rs**			Rd: Zielregister	Rs: Quellregister	! Zielregister ist auch Operand 1
Schiebe- und Rotationsbefehle	**Opcode**					**Rd**			**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	Rd: Zielregister		! Zielregister ist auch Operand
Datenbewegungsbefehle																			
Register-Register	**Opcode**					**Rd**			**X**	**X**	**X**	**X**	**X**	**Rs**			Rd: Zielregister	Rs: Quellregister	! psh/pll haben nur Rd oder Rs
Register-Speicher indirekt	**Opcode**					**Rd**			**Offset**					**Rs**			Rd: Zielregister	Rs: Quellregister	! stz/ldz nutzen r7 als Adressangabe. Entweder Rd oder Rs
Register-Speicher direkt	**Opcode**					**Rd/Rs**			**Address**								Rs/Rd: Ziel-/Quellr	Address: Adresse	! st/ld entweder Rd oder Rs. Diese sind dann an den entsprechenden Stellen
Immediate	**Opcode**					**Rd**			**Data**								Rd: Zielregister	Data: Daten	! Direktes Laden des 8 Bit Datenworts in Rd
Programmsteuerbefehle																			
Absolute Sprungbefehle	**Opcode**					**Address**												Address: Zieladresse	! Address ist eine absolute Adressangabe im Programmnspeicher
Relative Sprungbefehle	**Opcode**					**S**	**X**	**X**	**Offset**					**Cond**			Offset: Sprungweite	Cond: Sprungbedingu	! Cond ist ein optionaler Parameter nur bei brbs, brbc. Nicht bra
Systemsteuerbefehle	**Opcode**					**Rd**			**X**	**X**	**X**	**X**	**X**	**Rs**			Rd: Zielregister	Rs: Quellregister	! Rd und Rs optional je nach Befehl (hlt, lcr, stcr)

Mnemonik	Maschinenbefehl	Beschreibung (intern)	Beschreibung (Text)
	Opcode **Operands**		
nop	**0 0 0 0 0** **X X X X X X X X X X X**	---	1 Wartetakt ohne Operation
add	**0 0 0 0 1** **Rd** **X X X X X** **Rs**	Rd <= Rd + Rs	Addition der Registerinhalte Rd + Rs
addc	**0 0 0 1 0** **Rd** **X X X X X** **Rs**	Rd <= Rd + Rs + C	Addition der Registerinhalte Rd + Rs mit vorangehendem Übertrag
sub	**0 0 0 1 1** **Rd** **X X X X X** **Rs**	Rd <= Rd - Rs	Subtraktion der Registerinhalte Rd - Rs
subc	**0 0 1 0 0** **Rd** **X X X X X** **Rs**	Rd <= Rd - Rs - C	Subtraktion der Registerinhalte Rd - Rs mit vorangehendem Übertrag
inc	**0 0 1 0 1** **Rd** **X X X X X X X X**	Rd <= Rd + 1	Inkrementieren des Registerinhalts Rd um 1
dec	**0 0 1 1 0** **Rd** **X X X X X X X X**	Rd <= Rd - 1	Dekrementieren des Registerinhalts Rd um 1
and	**0 0 1 1 1** **Rd** **X X X X X** **Rs**	Rd <= Rd AND Rs	UND-Verknüpfung der Registerinhalte Rd AND Rs
or	**0 1 0 0 0** **Rd** **X X X X X** **Rs**	Rd <= Rd OR Rs	ODER-Verknüpfung der Registerinhalte Rd OR Rs
xor	**0 1 0 0 1** **Rd** **X X X X X** **Rs**	Rd <= Rd XOR Rs	XOR-Verknüpfung der Registerinhalte Rd XOR Rs
not	**0 1 0 1 0** **Rd** **X X X X X X X X**	Rd <= NOR Rd	Logische Invertierung des Registerinhalts Rd
sll	**0 1 0 1 1** **Rd** **X X X X X X X X**	Rd <= Rd << 1	Logisches Verschieben des Inhalts von Rd um 1 Bitstelle nach links
slr	**0 1 1 0 0** **Rd** **X X X X X X X X**	Rd <= Rd >> 1	Logisches Verschieben des Inhalts von Rd um 1 Bitstelle nach rechts
mov	**0 1 1 0 1** **Rd** **X X X X X** **Rs**	Rd <= Rs	Kopieren des Registerinhalts von Rs nach Rd
psh	**0 1 1 1 0** **X X X** **X X X X X** **Rs**	MEM[SP] <= Rs; SP <= SP + 1	Speichern des Registerinhalts von Rs auf dem Stack
pll	**0 1 1 1 1** **Rd** **X X X X X X X X**	Rd <= MEM[SP - 1]; SP <= SP - 1	Speichern des ersten Stackeintrags im Register Rd
stz	**1 0 0 0 0** **X X X** **Offset** **Rs**	MEM[r7 + Offset] <= Rs	Speichern des Inhalts von Rs im Speicher an der Adresse in r7 + Offset
ldz	**1 0 0 0 1** **Rd** **Offset** **X X X**	Rd <= MEM[r7 + Offset]	Laden des Inhalts von der Adresse in r7 + Offset des Speichers in Rd
st	**1 0 0 1 0** **Address** **Rs**	MEM[Address] <= Rs	Speichern des Inhalts von Rs im Speicher an der angegebenen Adresse
ld	**1 0 0 1 1** **Rd** **Address**	Rd <= MEM[Address]	Laden des Inhalts von der angegebenen Adresse im Speicher in Rd
ldi	**1 0 1 0 0** **Rd** **Data**	Rd <= Data	Laden des angegebenen Wertes in Rd
jpa	**1 0 1 0 1** **Address**	PC <= Address	Springen zur absolut angegebenen Adresse im Programm
bra	**1 0 1 1 0** **X X X** **Offset** **X X X**	PC <= PC + Offset	Springen um den relativ angegebenen Offset im Programm
brs	**1 0 1 1 1** **S X X** **Offset** **Cond**	PC <= PC + Offset IF Cond=1	Springen um den relativ angegebenen Offset im Programm, wenn Cond true
brc	**1 1 0 0 0** **S X X** **Offset** **Cond**	PC <= PC + Offset IF Cond=0	Springen um den relativ angegebenen Offset im Programm, wenn Cond false
call	**1 1 0 0 1** **Address**	MEM[SP,+1] <= PC, SP <= SP + 2	Aufrufen der Subroutine an der angegebenen Adresse
ret	**1 1 0 1 0** **X X X X X X X X X X X**	PC <= MEM[SP - 1,2], SP <= SP - 2	Zurückkehren von der Subroutine ins Hauptprogramm
res2	**1 1 1 0 1** **X X X X X X X X X X X**	nop	freier Befehl (aktuell nop)
res3	**1 1 1 1 0** **X X X X X X X X X X X**	nop	freier Befehl (aktuell nop)
lcr	**1 1 0 1 1** **Rd** **X X X X X X X X**	CR <= Rs	Lade den Inhalt des Control Registers in Rd. C, V, N, S, Z: write protected
stcr	**1 1 1 0 0** **X X X X X X X X** **Rs**	Rd <= CR	Speichere den Inhalt von Rs im Control Register
hlt	**1 1 1 1 1** **X X X X X X X X X X X**	CLK off	Gesamten Prozessor und Programmausführung anhalten