Befehl	Bemerkung	Beispiel	Zyklen
ABA	Addiert A mit B	ABA ; A=A+B	2
ABX	Addiert X mit B (Adressrechnung) B als vorzeichenlose Zahl	ABX ; X=X+B	3
ABY	Addiert Y mit B (Adressrechnung) B als vorzeichenlose Zahl	ABY ; Y=X+B	4
ADCA	Addition mit Übertrag	ADCA $21 ; A=A+$21+C	2-5
ADCB	Addition mit Übertrag	ADCB $21 ; B=B+$21+C	2-5
ADDA	Addiere zu A	ADDA $21 ; Addiert zu A was in $21 steht: A=A+$21	2-5
ADDB	Addiere zu B	ADDB $21 ; Addiert zu B was in $21 steht: B=B+$21	2-5
ADDD	Addiere zu D	ADDD $21 ; Addiert zu D was in $21 steht: D=D+$21	4-7
ANDA	UND-Verknüpfung mit A Anwendung: Löschen einzelner Bits	ANDA #$0F ; Löschen der 4 höherwertigen Bits von Akku A ANDA #%11101111 ; Rücksetzen des 5. Bits	2-5
ANDB	UND-Verknüpfung mit B Anwendung: Löschen einzelner Bits	ANDB #$0F ; Löschen der 4 höherwertigen Bits von Akku B ANDB #%11101111 ; Rücksetzen des 5. Bits	2-5
ASL für A,B,D	Arithm. Links schieben, wie LSL		adr: 6-7 A,B: 2 D:3
ASR ASR adr. für A,B,D	Arithmetisch nach Rechts schieben, VZ nachschieben, LSB in C-Flag Entspricht Div mit 2	; Akku A vorher: #%10101010 LSRA ; Akku A danach: #%11010101 C-Flag = 0	adr: 6-7 A,B: 2 D:3
BCC dest.	Springe zu dest, wenn C-Flag =0	BCC MARKE ; Sringt nach Marke wenn Carry-Flag=0	3
BCLR	wie BSET nur löschen anst.v. Setzen		6-8
BCS dest.	Springe zu dest, wenn C-Flag =1	BCS MARKE ; Sringt nach Marke wenn Carry-Flag=1	3
BEQ dest.	Springe zu dest, wenn Z-Flag =1 Entspr: wenn gleich 0, springe zu..	BQE MARKE ; Sringt nach Marke wenn Zero-Flag=1	3
BGE dest.	Springe zu dest, wenn OP1>=OP2 (signed)		3
BGT dest.	Springe zu dest, wenn OP1>OP2 (signed)		3
BHI dest.	Springe zu dest, wenn Op1>Op2 Natürlich (unsigned)	LDAA #$F8 ; Lade Akku A mit hex F8 CMPA #$12 ; Vergleiche Akku A (hex F8) mit hex 12 BHI true ; Springe zu „Funktion" true wenn wahr (Op1>Op2) BHS false ; Springe zu „Funktion" false w. falsch (Op1<=Op2)	3
BHS dest.	Springe zu dest, wenn Op1>=Op2 Natürlich (unsigned)	Siehe BHI	3
BITA	Test von Akku A auf 0, bzw VZ. Ohne A zu verändern	BITA $21 ; A und $21 Flags neu setzen, A bleibt unverändert	2-5
BITB	Wie BITA nur mit Akku B		2-5
BLE dest.	Springe zu dest, wenn OP1<=OP2 (signed)		3
BLO dest.	Springe zu dest, wenn Op1<Op2 Natürlich (unsigned)		3
BLS dest.	Springe zu dest, wenn Op1<=Op2 Natürlich (unsigned)		3
BLT dest.	Springe zu dest, wenn OP1<OP2 (signed)		3
BMI dest	Springe zu dest, wenn Operand negativ, N=1		3
BNE dest.	Springe zu dest, wenn Z-Flag =0 Entspr: wenn ungleich 0, springe zu	BNE MARKE ; Sringt nach Marke wenn Zero-Flag=0	3
BPL dest	Springe zu dest, wenn Operand positiv, N=0		3
BRA	Sprung (Relativ um X Befehle) X ∈ [-128,...,127] bei 68HC11	BRA 3 – (3 Befehle nach vorn) | BRA MRKE – springe zu Marke	3
BRN	Branch Never, macht nichts		3
BSET	Setzen von Bits in Speicherworten. 1. OP: adr. 2. OP:Maske (ohne #)	BSET $21 1	6-8
BSR dest.	Sprung (Relativ) Ass. übern. Berechn.		6-8

BVC dest.	Springe zu dest, wenn V-Flag =0		3
BVS dest.	Springe zu dest, wenn V-Flag =1		3
CBA	Wie CMPA, A wird mit B verglichen		2
CLC	Clear Carry Flag, C-Flag auf 0		2
CLI	Clear Interrupt Flag, I-Flag auf 0		2
CLR	Clear Akkumulator: A und B wird mit 0 beschrieben	CLR $FF – Adresse $FF wird auf 0 gesetzt	6-7
CLRA	Clear Akkumulator A: A wird mit 0 beschrieben, Schneller als LDAA #0		2
CLRB	Clear Akkumulator B: B wird mit 0 beschrieben, Schneller als LDAB #0		2
CLV	Clear overflow Flag,V-Flag auf 0		2
CMPA adr	Reg. Inhalt verglichen mit Speicher-Inhalt, oder Konstante. Flags: ZFlag=1, w. Beide Operanden gleich N-Flag=1,w. (Reg.)< Operand (konegativ und kein Überlauf) C-Flag=1, w. (Reg.)<OP. (nat. Zahl) V-Flag=1,w. Arithm. Überlauf		2-5
CMPB adr	Wie CMPA		2-5
COM	Wie COMA nur mit belibiger Adr.		6-7
COMA	invertieren von Akku A; 1er-Kompl.	COMA ; A=$\overline{A}$	2
COMB	Wie COMA nur mit Akku B		2
CP□ adr. f. D,X,Y	Wie CMPA		D,Y: 5-7 X: 4-7
DAA	Dezimalkorrektur von Akku A Für BCD-Arithmetik, C-Flag Übertrag	1. Zahl: Adr. $31-$32 | 2. Zahl: Adr. $33-$34 | Erg: Adr. $35-$36 LDAA $32 niederwertige 2 Ziffern ADDA $34 binär addieren DAA BCD-Justage STAA $36 LDAA $31 höherwertige 2 Ziffern ADCA $34 binär addieren mit Carry! DAA BCD-Justage STAA $35	2
DC.B	Define Constant Byte Reservierung v. Speicherplatz u. Belegung m. Byte-Werten	TAB DC.B $0D,$0A,$03 TEXT DC.B “Text\n\r“	
DC.L	Wie DC.B und DC.W Nur mit Typ Long-Word	XTAB DC.L $FF30FF30,$F000FF30	
DC.W	Define Constant Word Reservierung v. Speicherplatz u. Belegung m. Word-Werten	ATAB DC.W $FF30,$F000	
DEC	Dekrementieren eines Werts	DEC $21 ; $21=$21-1	6-7
DECA	dekrementieren von A	DECA ; A=A-1	2
DECB	Wie DECB		2
DES	dekrementieren von S (kein Flag)	DES ; S=S-1	3
DEX	X-- (nur Z-Flag beeinflusst)	DEX ; X=X-1	3
DEY	Wie DEX		4-7
DS.B	Reservieren von angegebener Anzahl an Speicher-Bytes Ohne Vorbelegen	PUFFER DS.B 100	
DS.L	Reservieren von angegebener Anzahl an Speicher-Longwords Ohne Vorbelegen	PUFFER DS.L 100	

DS.W	Reservieren von angegebener Anzahl an Speicher-Words Ohne Vorbelegen	PUFFER DS.W 100	
END	Quellcode zu ende (optional)		
EORA	Exklusiv-oder-Verknüpfung mit A Anwendung: Invertieren von Bits	EORA #%11110000 ; Invert. der 4 höherwertigen Bits von A	2-5
EORB	Wie EORA nur mit Akku B		2-5
EQU	Entspricht #define in C	ETX EQU 3	
FDB	Form Double Byte	Äquivalent mit DC.W	
FDIV	VZ. lose Division, Quot. Echter Bruch Für D>X: X=D/X Rest in D	FDIV; $1000 : $2000 = $.8000 R $0000	41
IDIV	VZ. lose Division, ganzzahliger Anteil Für D>X: X=D/X Rest in D	IDIV ; $8421 : $0004=$2108 R $0001	41
INC	increment	INC $21 ; $21=$21+1;	6-7
INCA	increment A (A++)	INCA ; A=A+1	2
INCB	increment B (B++)	INCB ; B=B+1	2
INCLUDE	Einfügen einer Quelldatei an Entsprechender Stelle	INCLUDE <HC11.h> INCLUDE "definitionen.a"	
INS	increment S (S++) (kein Flag)	INCS ; S=S+1	3
INX	increment X (nur Z-Flag beeinflusst)	INX ;X=X+1	3
INY	increment Y (nur Z-Flag beeinflusst)	INY ;Y=Y+1	4-7
JMP	Sprung (Absolut)	JMP MARKE – Springe nach Marke	3-4
JSR adr.	Springt zu Unterprogramm, welches mit RTS beendet werden muss	JSR uProg ; Springt zu uProg … ; evtl. weiterer Code uProg … ; Unterprogramm RTS ; Springt wieder zurück	5-7
LDAA	Lade Akku A (8bit)	LDAA #20 – Lade Akku A mit dem Wert 20	2-5
LDAB	Lade Akku B (8bit)	LDAB #20 – Lade Akku B mit dem Wert 20	2-5
LDD	Lade Akku D (16bit)	LDD #20 – Lade Akku D mit dem Wert 20 – D ist A und B zus.	3-6
LDS	Lade Register S (16bit)	LDY #20 – Lade Register S mit dem Wert 20 – S = Stackpointer	3-6
LDX	Lade Register X (16bit)	LDX #20 – Lade Register X mit dem Wert 20	3-6
LDY	Lade Register Y (16bit)	LDY #20 – Lade Register Y mit dem Wert 20	4-6
LIST	Fortsetzen des Listings nach einem vorher platzierten NOLIST-Befehl.		
LSL␣ LSL adr.	Nach Links schieben, | f. A,B,D 0 nachschieben, MSB in C-Flag	; Akku A vorher: #%01010101 LSLA ; Akku A danach: #%10101010 C-Flag = 0	A,B:2|D: 3 adr. 6-7
LSR␣ LSR adr.	Nach Rechts schieben, | f. A,B,D 0 nachschieben, LSB in C-Flag	; Akku A vorher: #%01010101 LSRA ; Akku A danach: #%00101010 C-Flag = 1	A,B:2|D: 3 adr. 6-7
MUL	VZ. lose Multiplikation D=A*B Z-Flag 1 wenn Erg.=0 sonst 1 C-Flag = Bit 7 von B		10
NEG␣	Negieren; 2er-Komplement	NEGA ;A negieren | NEGB; B negieren | NEG $21 ;$21 negieren	A,B: 2 adr.: 6-7
NOLIST	Unterdrücken der Erzeugung eines Programmlistings bis zum nächsten LIST-Befehl.		
NOP	No operation, macht nichts		
ORAA	ODER-Verknüpfung mit A Anwendung: Setzen einzelner Bits	ORAA #$F0 ; Setzen der 4 höherwertigen Bits von Akku A ORAA #%00010000 ; Setzen des 5. Bits	2-5
ORAB	Wie ORAA nur mit Akku B		2-5
ORG	Legt Adresse fest, ab der nachfolgende Code in Speicher Abgelegt wird.	ORG $8000 – Programm beginnt ab Speicherzelle $8000	
PHS␣	Schreibt Register auf Sack | f. A,B,X,Y		A,B: 3 X:4 | Y:5
PUL␣	Schreibt Stack in Register | f. A,B,X,Y		A,B: 4 X:5 | Y:6

ROL‾	Nach Links rotieren: | f. A,B C-Flag → LSB, MSB → C-Flag		A,B: 2 adr.: 6-7
ROR‾	Nach Rechts rotieren: | f. A,B C-Flag → MSB, LSB → C-Flag		A,B: 2 adr.: 6-7
RTI	Beendet Interrupt service routine		12
RTS	Beendet Unterprogramm, springt in aufrufendes Programm zurück	Siehe JSR	5
SBA	Subtaktion Reg A=RegA – RegB		2
SBCA	Subtraktion von A mit Übertrag	SUBA $21 ; A=A-$21+C	2-5
SBCB	Subtraktion von B mit Übertrag	SUBB $21 ; B=B-$21+C	2-5
SEC	Set Carry Flag, C-Flag auf 1		2
SEI	Set Interrupt Flag, I-Flag auf 1		2
SEV	Set overflow Flag, V-Flag auf 1		2
STAA	Register A in Memory speichern 8bit	STAA $49 | STAA VAR	3-5
STAB	Register B in Memory speichern 8bit	STAB $49 | STAB VAR	3-5
STD	Register D in Memory speichern (16-bit), in Adresse a werden Höchstwertigste 8bit gespeichert In a+1 niederwertigste 8bit		4-6
STOP	Abhängig von S-Flag: S=1: next instruction NOP S=0: halt clocks, enter standby and Wait for XIRQ or IRQ	Freigabe des STOP-Befehls: TPA ; Status Register nach Akku A ANDA #$7F ; Freigabe (S-Flag auf 0) TAP ; Akku A nach Statusregister	2
STS	Wie STD nur mit Register S		4-6
STX	Wie STD nur mit Register X		4-6
STY	Wie STD nur mit Register Y		5-6
SUBA	Subtraktion von A	SUBA $21 ; A=A-$21	2-5
SUBB	Subtraktion von B	SUBB $21 ; B=B-$21	2-5
SUBD	Subtraktion von D (16bit)		4-7
SWI	Software interrupt, Sprung in eine interrupt-service-routine (ISR) Startadresse ist interruptvektor Vorher „retten" der Register in Stack Beendigung der ISR mit RTI		14
TAB	Inhalt von A nach B kopieren, A & B danach gleichen Inhalt		2
TAP	Inhalt von A in P (Statusregister)		2
TBA	Inhalt von B nach A kopieren, A & B danach gleichen Inhalt		2
TPA	Inhalt von P (Statusregister) nach A kopieren		2
TST‾ TST adr.	Test auf 0 und VZ. | f. A,B ZFlag=1, wenn 0 | Nflag enth. MSB	TSTA ; Bei Inhalt #%10001000 ZFlag=0 und NFlag=1 TSTB ; Bei Inhalt #0 ZFlag=1 und NFlag=0	A,B: 2 adr.: 6-7
TSX	Inhalt von S nach X kopieren, S & X danach gleichen Inhalt		3
TSY	Inhalt von S nach Y kopieren, S & Y danach gleichen Inhalt		4
TXS	Inhalt von X nach S kopieren, S & X danach gleichen Inhalt		3
TYS	Inhalt von Y nach S kopieren, S & Y danach gleichen Inhalt		4
WAI	Wait for Interrupt, Ablage aller Register auf den Stack, warte auf I.		14+n
XGDX	X und D werden ausgetauscht		3
XGDY	Y und D werden ausgetauscht		4