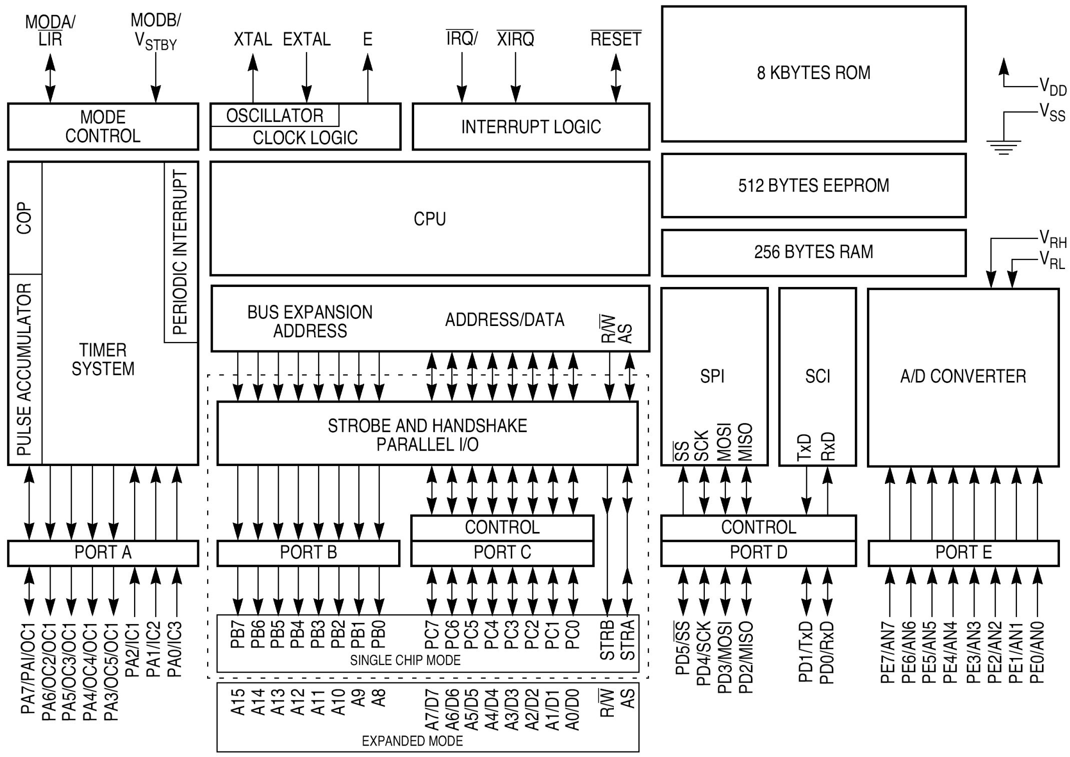
2. Der Microcontroller 68HC11

Der Microcontroller 68HC11 ist ein 1 Chip Rechner mit einer 8 Bit Verarbeitungsbreite. Beim MC68HC11A8 sind 8K ROM, 512 Byte EEPROM und 256 Byte RAM als Speicher auf dem Chip vorhanden. In einem speziellen Mode kann der 68HC11 auch externe Speicher adressieren. Der Adreßraum beträgt maximal 64K Byte. In ihm sind dann aber auch noch die E/A-Register und die Systemstatus- und Configurationsregister enthalten. Ein Blockschaltbild ist in Bild 2.1 dargestellt.



CIRCUITRY ENCLOSED BY DOTTED LINE IS EQUIVALENT TO MC68HC24.

Bild 2.1 Blockschaltbild des 68HC11

Folgende Configurationen werden von Motorola ebenfalls angeboten:

Part Number	EPROM	ROM	EEPROM	RAM	CONFIG[2]	Comments
MC68HC11A8	—	—	512	256	$0F	Family Built Around This Device
MC68HC11A1	—	—	512	256	$0D	'A8 with ROM Disabled
MC68HC11A0	—	—	—	256	$0C	'A8 with ROM and EEPROM Disabled
MC68HC811A8	—	—	8K + 512	256	$0F	EEPROM Emulator for 'A8
MC68HC11E9	—	12K	512	512	$0F	Four Input Capture/Bigger RAM 12K ROM
MC68HC11E1	—	—	512	512	$0D	'E9 with ROM Disabled
MC68HC11E0	—	—	—	512	$0C	'E9 with ROM and EEPROM Disabled
MC68HC811E2	—	—	2K[1]	256	$FF[3]	No ROM Part for Expanded Systems
MC68HC711E9	12K	—	512	512	$0F	One-Time Programmable Version of 'E9
MC68HC11D3	—	4K	—	192	N/A	Low-Cost 40-Pin Version
MC68HC711D9	4K	—	—	192	N/A	One-Time Programmable Version of 'D3
MC68HC11F1	—	—	512[1]	1K	$FF[3]	High-Performance Non-Multiplexed 6B-Pin
MC68HC11K4	—	24K	640	768	$FF	> 1 Mbyte memory space, PWM, C_S, 84-Pin
MC68HC711K4	24K	—	640	768	$FF	One-Time Programmable Version of 'K4
MC68HC11L6	—	16K	512	512	$0F	Like 'E9 with more ROM and more I/O, 64/68
MC68HC711L6	16K	—	512	512	$0F	One-Time Programmable Version of 'L4

1. The EEPROM is relocatable to the top of any 4 Kbyte memory page. Relocation is done with the upper four bits of the CONFIG register.
2. CONFIG register values in this table reflect the value programmed prior to shipment from Motorola.
3. At the time of this printing a change was being considered that would make this value $0F.

Bild 2.2 Configurationsvarianten des 68HC11

Abhängig von der Gehäuseart ist die Anzahl der Analogeingänge beim 48-Pin Gehäuse auf 4 reduziert.

2.1 Die Register und die Speicheraufteilung des 68HC11

Der 68HC11 besitzt 2 8 Bit Accumulatoren A und B, die als ein 16 Bit Accumulator D angesprochen werden können. Dabei bildet A den HIGH-Teil und B den LOW-Teil von Register D. Die 16 Bit Register IX und IY können als Accumulator für Additionen mit Register B (ABX, ABY) oder für die Division (FDIV, IDIF) verwendet werden. Hauptsächlich sind sie für die indexierte Adressierung des Speichers einzusetzen. Ein 16 Bit Stackpointer adressiert einen frei wählbaren Stackbereich im Speicher. Er wird von den Unterprogrammbefehlen, den PSH- und PUL- Befehlen sowie für die Interruptorganisation verwendet. Bei einer Interruptannahme werden alle CPU-Register gerettet. Mit dem WAI-Befehl kann dieses Retten vorbereitet werden, um die Interruptannahme noch zu verkürzen. Der RTI-Befehl muß eine Interruptbehandlung beenden und rettet alle Register in umgekehrter Reihenfolge zurück. Der 68HC11 schreibt bei allen Einkellervorgängen in den Stack zuerst das entsprechende Register auf die aktuelle Stackpointeradresse und dekrementiert erst danach den SP. Beim Auskellern wird zuerst der SP inkrementiert und dann das entsprechende Register aus dem Stack geladen. Zum Registersatz gehören dann noch der 16 Bit Befehlszähler (PC) und das 8 Bit Bedingungscoderegister (CCR). Die CPU-Register sind in Bild 2.3 dargestellt.

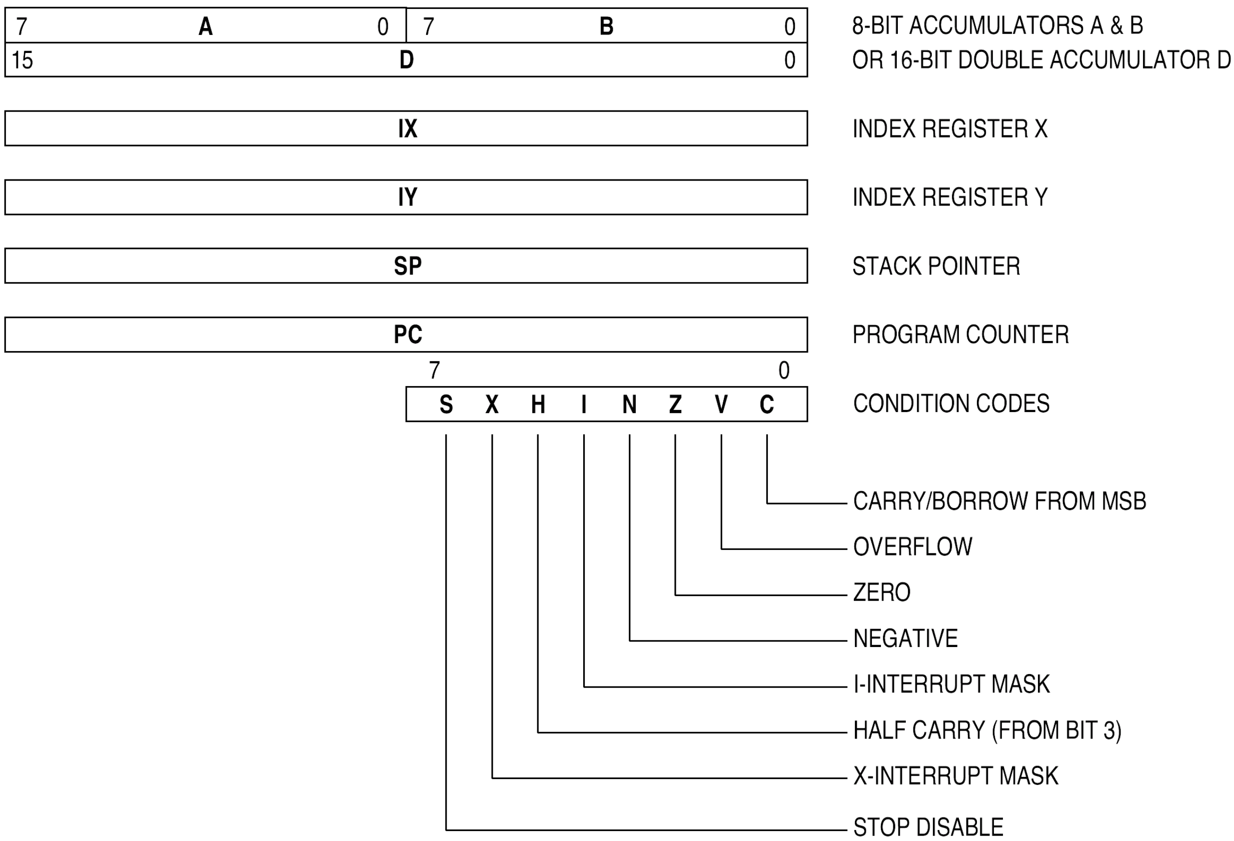


Bild 2.3 CPU-Register des 68HC11

Zur Steuerung aller Ein-/Ausgabefunktionen mit den Interfacemodulen des 68HC11, die im Bild 2.1 zu sehen sind, ist ein Registerblock in den 64K Speicherbereich eingeblendet. Seine Anfangsadresse ist nach RESET auf der Adresse 1000 Hex und kann in Schritten von 1000 Hex in dem 64K Adreßraum verschoben werden. Das gleiche gilt für die 256 Byte RAM. Sie befinden sich nach RESET auf der Adresse 0. Bei der Verschiebung der Anfangsadressen beider Speicherbereiche muß darauf geachtet werden, daß keine Überlappung untereinander oder mit den EEPROM- oder ROM-Bereichen entsteht. Die E/A-Register sind in Bild 2.4 und die Speicheraufteilung in Bild 2.5 dargestellt.

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Name	Description
$1000	Bit 7	–	–	–	–	–	–	Bit 0	PORTA	I/O Port A
$1001									Reserved	
$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB	PIOC	Parallel I/O Control Register
$1003	Bit 7	–	–	–	–	–	–	Bit 0	PORTC	I/O Port C
$1004	Bit 7	–	–	–	–	–	–	Bit 0	PORTB	Output Port B
$1005	Bit 7	–	–	–	–	–	–	Bit 0	PORTCL	Alternate Latched Port C
$1006									Reserved	
$1007	Bit 7	–	–	–	–	–	–	Bit 0	DDRC	Data Direction for Port C
$1008			Bit 5	–	–	–	–	Bit 0	PORTD	I/O Port D
$1009			Bit 5	–	–	–	–	Bit 0	DDRD	Data Direction for Port D
$100A	Bit 7	–	–	–	–	–	–	Bit 0	PORTE	Input Port E
$100B	FOC1	FOC2	FOC3	FOC4	FOC5				CFORC	Compare Force Register
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3				OC1M	OC1 Action Mask Register
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3				OC1D	OC1 Action Data Register
$100E	Bit 15	–	–	–	–	–	–	Bit 8	TCNT	Timer Counter Register
$100F	Bit 7	–	–	–	–	–	–	Bit 0		
$1010	Bit 15	–	–	–	–	–	–	Bit 8	TIC1	Input Capture 1 Register
$1011	Bit 7	–	–	–	–	–	–	Bit 0		
$1012	Bit 15	–	–	–	–	–	–	Bit 8	TIC2	Input Capture 2 Register
$1013	Bit 7	–	–	–	–	–	–	Bit 0		
$1014	Bit 15	–	–	–	–	–	–	Bit 8	TIC3	Input Capture 3 Register
$1015	Bit 7	–	–	–	–	–	–	Bit 0		
$1016	Bit 15	–	–	–	–	–	–	Bit 8	TOC1	Output Compare 1 Register
$1017	Bit 7	–	–	–	–	–	–	Bit 0		
$1018	Bit 15	–	–	–	–	–	–	Bit 8	TOC2	Output Compare 2 Register
$1019	Bit 7	–	–	–	–	–	–	Bit 0		
$101A	Bit 15	–	–	–	–	–	–	Bit 8	TOC3	Output Compare 3 Register
$101B	Bit 7	–	–	–	–	–	–	Bit 0		
$101C	Bit 15	–	–	–	–	–	–	Bit 8	TOC4	Output Compare 4 Register
$101D	Bit 7	–	–	–	–	–	–	Bit 0		
$101E	Bit 15	–	–	–	–	–	–	Bit 8	TCO5	Output Compare 5 Register
$101F	Bit 7	–	–	–	–	–	–	Bit 0		

Bild 2.4.1 E/A-Register des 68HC11

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	TCTL1	Timer Control Register 1
$1021			EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	TCTL2	Timer Control Register 2
$1022	OC1I	OC2I	OC3I	OC4I	OC5I	IC1I	IC2I	IC3I	TMSK1	Timer Interrupt Mask Register 1
$1023	OC1F	OC2F	OC3F	OC4F	OC5F	IC1F	IC2F	IC3F	TFLG1	Timer Interrupt Flag Register 1
$1024	TOI	RTII	PAOVI	PAII			PR1	PR0	TMSK2	Timer Interrupt Mask Register 2
$1025	TOF	RTIF	PAOVF	PAIF					TFLG2	Timer Interrupt Flag Register 2
$1026	DDRA7	PAEN	PAMOD	PEDGE			RTR1	RTR0	PACTL	Pulse Accumulator Control Register
$1027	Bit 7	–	–	–	–	–	–	Bit 0	PACNT	Pulse Accumulator Count Register
$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR	SPI Control Register
$1029	SPIF	WCOL		MODF					SPSR	SPI Status Register
$102A	Bit 7	–	–	–	–	–	–	Bit 0	SPDR	SPI Data Register
$102B	TCLR		SCP1	SCP0	RCKB	SCR2	SCR1	SCR0	BAUD	SCI Baud Rate Control
$102C	R8	T8		M	WAKE				SCCR1	SCI Control Register 1
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCCR2	SCI Control Register 2
$102E	TRDE	TC	RDRF	IDLE	OR	NF	FE		SCSR	SCI Status Register
$102F	Bit 7	–	–	–	–	–	–	Bit 0	SCDR	SCI Data (Read RDR, Write TDR)
$1030	CCF		SCAN	MULT	CD	CC	CB	CA	ADCTL	A/D Control Register
$1031	Bit 7	–	–	–	–	–	–	Bit 0	ADR1	A/D Result Register 1
$1032	Bit 7	–	–	–	–	–	–	Bit 0	ADR2	A/D Result Register 2
$1033	Bit 7	–	–	–	–	–	–	Bit 0	ADR3	A/D Result Register 3
$1034	Bit 7	–	–	–	–	–	–	Bit 0	ADR4	A/D Result Register 4
$1035 thru $1038									Reserved	
$1039	ADPU	CSEL	IRQE	DLY	CME		CR1	CR0	OPTION	System Configuration Options
$103A	Bit 7	–	–	–	–	–	–	Bit 0	COPRST	Arm/Reset COP Timer Circuitry
$103B	ODD	EVEN		BYTE	ROW	ERASE	EELAT	EEPGM	PPROG	EEPROM Program Control Register
$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	HPRIO	Highest Priority I-Bit Int and Misc
$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	INIT	RAM and I/O Mapping Register
$103E	TILOP		OCCR	CBYP	DISR	FCM	FCOP	TCON	TEST1	Factory TEST Control Register
$103F	–	–	–	–	NOSEC	NOCOP	ROMON	EEON	CONFIG	COP, ROM, and EEPROM Enables

Bild 2.4.2 E/A-Register des 68HC11

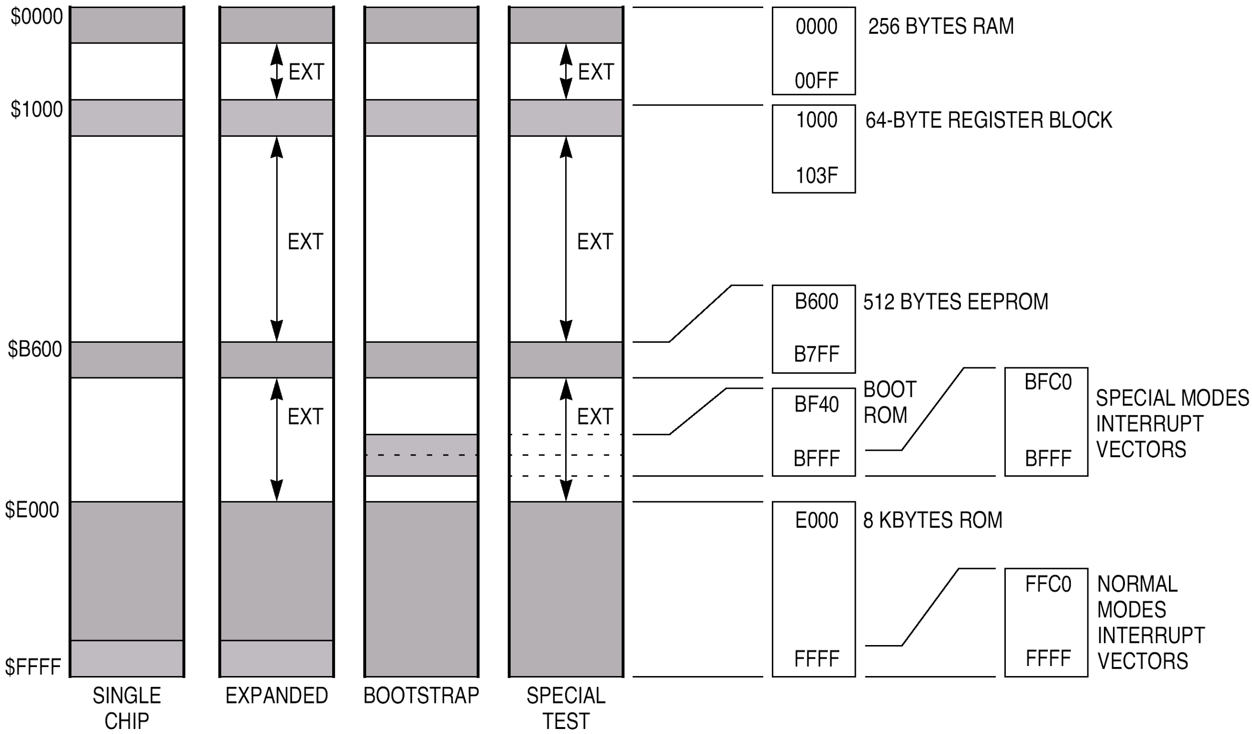


Bild 2.5 Speicheraufteilung des 68HC11

Die Auswahl der 4 Speichermodi erfolgt durch eine Beschaltung der Signale MODA und MODB, wie sie in der Tabelle im Bild 2.6 angegeben ist.

MODB	MODA	Mode Selected
1	0	Single Chip
1	1	Expanded Multiplexed
0	0	Special Bootstrap
0	1	Special Test

Bild 2.6 Beschaltung von MODA und MODB

Für kleine Anwendungen, die nur den EEPROM und RAM benötigen, ist der Special Bootstrap Mode von besonderer Bedeutung. Er gestattet das Starten von Programmen im RAM oder im EEPROM und realisiert auch das Laden von Daten in den RAM. Für diese Aufgaben ist ein Programm zuständig, das sich in einem kleinen ROM ab Adresse hex BF40 befindet. Dieses Programm wird immer nach RESET gestartet, wenn sich der 68HC11 im Special Bootstrap Mode befindet. Dazu liest die CPU den Vektor auf Adresse hex BFFE und startet das Programm auf der dort gefundenen Adresse. Der Inhalt dieses ROM ist in der folgenden Liste zu sehen:

```
        0000bf40        org     $bf40
bf40 8e00ff             lds     #$0ff            * RAM-Ende
bf43 ce1000             ldx     #REG             * Adresse Registerblock
bf46 1c2820             bset    $20,ospcr,x      *set bit5 auf $1028      PortD Out=OD
bf49 86a2               ldaa    #$a2
bf4b a72b               staa    obaud,x          * ($102b):= $a2          f/16=7812,5
bf4d 860c               ldaa    #$0c
bf4f  a72d              staa    osccr2,x         *($102d):= $0c           Rx Tx Freigabe
bf51 1c2d01             bset    1,osccr2,x       *set bit0 auf $102d      Sende Break
bf54 1e0801fc   w_0     brset   1,oportd,x,w_0   *warte auf bit0 in $1008 Empf. Break
bf58 1d2d01             bclr    1,osccr2,x       *res bit0 auf $102d      Break aus
bf5b 1f2e20fc   w_1     brclr   $20,oscsr,x,w_1  *warte auf bit5 in $102e Rx Daten ?
bf5f  a62f              ldaa    oscdr,x          *A:=($102f)              Daten lesen
bf61 2603               bne     no_stee          * RAM-Laden/Starten
```

```
bf63 7eb600              jmp     EEPROM              * start EEPROM

bf66 8155       no_stee  cmpa    #$55
bf68 271e                beq     str_ram             * RAM-Starten
bf6a 81ff                cmpa    #$ff
bf6c 2703                beq     m0                  * mit gleicher Baudrate
bf6e 1c2b33              bset    $33,obaud,x         * set bit 0,1,4,5 $102b      f/104=1201,9

     * Laden der empfangenen Byte in den RAM und START

bf71 18ce0000   m0       ldy     #0
bf75 1f2e20fc   w_2      brclr   $20,oscsr,x,w_2     *warte auf bit5 in $102e     Rx Daten ?
bf79 a62f                ldaa    oscdr,x             *A:=($102f)                  Daten lesen
bf7b 18a700              staa    0,y                 * speichern in RAM !!!
bf7e a72f                staa    oscdr,x             * Byte ECHO
bf80 1808                iny
bf82 188c0100            cpy     #$100               * RAM-ENDE ???
bf86 26ed                bne     w_2                 * Nein !!!
bf88 7e0000     str_ram  jmp     0                   * Start PG im RAM

        0000bfd4         org     $bfd4
bfd4 7498                dc.w    $7498

bfd6 00c4       int00    dc.w    $00c4               * SCI Serial System
bfd8 00c7       int01    dc.w    $00c7               * SPI Serial Transfer Complete
bfda 00ca       int02    dc.w    $00ca               * Pulse Accumulator Input Edge
bfdc 00cd       int03    dc.w    $00cd               * Pulse Accumulator Overflow
bfde 00d0       int04    dc.w    $00d0               * Timer Overflow
bfe0 00d3       int05    dc.w    $00d3               * Timer Output Compare 5
bfe2 00d6       int06    dc.w    $00d6               * Timer Output Compare 4
bfe4 00d9       int07    dc.w    $00d9               * Timer Output Compare 3
bfe6 00dc       int08    dc.w    $00dc               * Timer Output Compare 2
bfe8 00df       int09    dc.w    $00df               * Timer Output Compare 1
bfea 00e2       int10    dc.w    $00e2               * Timer Input Capture 3
bfec 00e5       int11    dc.w    $00e5               * Timer Input Capture 2
bfee 00e8       int12    dc.w    $00e8               * Timer Input Capture 1
bff0 00eb       int13    dc.w    $00eb               * Real Time Interrupt
bff2 00ee       int14    dc.w    $00ee               * ~IRQ (External Pin or Parallel I/O)
bff4 00f1       int15    dc.w    $00f1               * ~XIRQ Pin (Pseudo NMI)
bff6 00f4       int16    dc.w    $00f4               * SWI
bff8 00f7       int17    dc.w    $00f7               * illegal Op-Code Trap
bffa 00fa       int18    dc.w    $00fa               * COP Failure (Reset)
bffc 00fd       int19    dc.w    $00fd               * COP Clock Monitor Fail (Reset)

bffe bf40       reset    dc.w    $bf40               * Startvektor bei MODB=MODA=0
```

Im Special Bootstrap Mode werden auch alle Interruptvektoren in diesen ROM umgelenkt. Sie befinden sich ab Adresse hex BFD6 bis BFFC. Sie sind mit RAM Adressen im 3 Byte Abstand ab der Adresse hex 00C4 geladen, so daß der Anwender im RAM auf diese Adressen JMP-Befehle zu seiner Interruptbehandlung generieren kann. Damit ist in diesem Mode auch die Nutzung von Interrupts möglich. Für den Start des Programms im EEPROM müssen bei RESET die Signale TxD und RxD des SCI verbunden sein. Wenn dann durch das Bootprogramm auf der Adresse hex BF51 mit dem Befehl BSET das Senden eines Break ausgelöst wird, so muß am Port D0 = RxD eine 0 anliegen, und die Warteschleife auf hex BF5F Befehl BRSET wird verlassen. Durch das Senden von Break wird im Empfänger des SCI eine hex 00 empfangen. Beim Empfang von hex 00 springt das Bootprogramm die 1. Adresse des EEPROM (hex B600) an und startet damit ein dort geladenes Programm. Für das DOWN-Laden eines Programms in den RAM und Starten dieses Programms muß TxD über eine Pegelwandlung mit RxD und RxD des 68HC11 mit TxD einer RS232-Schnittstelle eines Hostrechners (IBM-PC) verbunden werden. Wenn dann der Hostrechner einen Break empfängt, kann er darauf mit dem Senden von zum Beispiel hex FF antworten. Da das Senden von Daten über asynchrone serielle Interface immer mit einem Startbit = 0 beginnt, wird auch in diesem Fall die Warteschleife auf der Adresse hex BF5F

Befehl BRSET verlassen. Danach verzweigt das Bootprogramm in eine Ladeschleife, in der alle weiteren empfangenen Bytes ab Adresse hex 0000 bis hex 00FF abgelegt werden. Sind alle 256 Bytes geladen, dann springt das Bootprogramm zu Adresse hex 0000 und startet damit das gerade empfangene Programm. Ein solches Programm kann dann zum Beispiel ein Programm zum Programmieren des EEPROM sein. Ein solches Programm speichert Daten, die vom seriellen Interface empfangen werden in den EEPROM und schickt die programmierten Daten zur Kontrolle an den Hostrechner zurück. Daraufhin kann dieser die Richtigkeit der programmierten Daten kontrollieren und das nächste Byte senden. Nach dem Senden eines Endekennzeichens durch den Hostrechner springt dann das Programm im RAM des 68HC11 zur Adresse hex B600 und startet damit das soeben in den EEPROM geschriebene Programm. Auf diesem Weg ist es möglich, ohne spezielle Hardware den Inhalt des EEPROM zu verändern.

2.2 Die Adressierungsarten des 68HC11

Für die Operandenadressierung stehen beim 68HC11 maximal 6 Möglichkeiten zur Verfügung.

- Der Operand steht in einem CPU-Register.

- Der Operand ist ein 8 oder 16 Bit Direktwert und steht unmittelbar in den Bytes des Befehlscodes.

- Der Operand steht in den unteren 256 Byte des Adreßraumes und wird durch einen 8 Bit Wert im Befehlscode adressiert. In der Assemblerbefehlszeile muß dann vor die symbolische Adresse ein "<" geschrieben werden. Wird es weggelassen, so wird vom Assembler der nachfolgende Mode benutzt.

- Der Operand steht an beliebiger Stelle des Adreßraumes und wird durch einen 16 Bit Wert im Befehlscode adressiert. Bei dieser Adressierungsart sind die Befehle grundsätzlich 1 Byte länger und brauchen 1 Takt mehr bei der Abarbeitung als bei der vorangegangenen Adressierungsart.

- Der Operand steht an beliebiger Stelle des Adreßraumes und wird durch den Inhalt des IX- Registers plus einem 8 Bit Offset (nur positiv von 0 bis 255), der im Operationscode steht, adressiert.

- Der Operand steht an beliebiger Stelle des Adreßraumes und wird durch den Inhalt des IY- Registers plus einem 8 Bit Offset (nur positiv von 0 bis 255), der im Operationscode steht, adressiert.

2.3 Die Befehle des 68HC11

Steht am Anfang einer Zeile in der nachfolgenden Befehlsliste ein "<" , so ist für diesen Befehl auch eine Adressierung mit einer 8 Bit Adresse möglich (siehe Adressierungsarten). Steht "<<" am Anfang der Zeile, so ist nur eine 8 Bit Adresse möglich. In der Assemblerquellzeile muß dann vor der symbolischen Adresse kein "<" geschrieben werden (siehe Bootprogramm BSET oder BRSET).

2.3.1 Arithmetikbefehle

2.3.1.1 Additionsbefehle

				(h,n,z,v,c	<--- opr)
	ABA			A	<--- A + B
	ABX			IX	<--- IX + (0^8 ## B)
				Flags werden nicht verändert	
	ABY			IY	<--- IY + (0^8 ## B)
				Flags werden nicht verändert	
	ADC	ADCA	#ii	A	<--- A + ii + c
<		ADCA	MA1	A	<--- A + M[MA1] + c
		ADCA	O1,X	A	<--- A + M[O1+IX] + c
		ADCA	O1,Y	A	<--- A + M[O1+IY] + c
		ADCB	#ii	B	<--- B + ii + c
		ADCB	weiter wie bei ADCA		
	ADD	ADDA	#ii	A	<--- A + ii
<		ADDA	MA1	A	<--- A + M[MA1]
		ADDA	O1,X	A	<--- A + M[O1+IX]
		ADDA	O1,Y	A	<--- A + M[O1+IY]

		ADDB	#ii	B	<--- B + ii
		ADDB	weiter wie bei	ADDA	
	ADDD			h wird nicht verändert	
		ADDD	#jjkk	D	<--- D + jjkk
<		ADDD	MA1	D	<--- D + (M[MA1] ## M[MA1+1])
		ADDD	O1,X	D	<--- D + (M[O1+IX] ## M[O1+IX+1])
		ADDD	O1,Y	D	<--- D + (M[O1+IY] ## M[O1+IY+1])

2.3.1.2 Subtraktionsbefehle (n,z,v,c <--- opr)

	NEG	NEGA		A	<--- 00 - A
		NEGB		B	<--- 00 - B
		NEG	MA1	M[MA1]	<--- 00 - M[MA1]
		NEG	O1,X	M[O1+IX]	<--- 00 - M[O1+IX]
		NEG	O1,Y	M[O1+IY]	<--- 00 - M[O1+IY]
	SBA			A	<--- A - B
	SBC	SBCA	#ii	A	<--- A - (ii + c)
<		SBCA	MA1	A	<--- A - (M[MA1] + c)
		SBCA	O1,X	A	<--- A - (M[O1+IX] + c)
		SBCA	O1,Y	A	<--- A - (M[O1+IY] + c)
		SBCB	#ii	B	<--- B - (ii + c)
		SBCB	weiter wie bei	SBCA	
	SUB	SUBA	#ii	A	<--- A - ii
<		SUBA	MA1	A	<--- A - M[MA1]
		SUBA	O1,X	A	<--- A - M[O1+IX]
		SUBA	O1,Y	A	<--- A - M[O1+IY]
		SUBB	#ii	B	<--- B - ii
		SUBB	weiter wie bei	SUBA	
	SUBD	SUBD	#jjkk	D	<--- D - jjkk
<		SUBD	MA1	D	<--- D - (M[MA1] ## M[MA1+1])
		SUBD	O1,X	D	<--- D - (M[O1+IX] ## M[O1+IX+1])
		SUBD	O1,Y	D	<--- D - (M[O1+IY] ## M[O1+IY+1])

2.3.1.3 Dezimalarithmetik (n,z,v,c <--- opr)

	DAA			A	<--- A + xx
					xx = 00 oder 06 oder 60 oder 66

2.3.1.4 Vergleichsbefehle

	CBA			n,z,v,c	<--- A - B
	CMP	CMPA	#ii	n,z,v,c	<--- A - ii
<		CMPA	MA1	n,z,v,c	<--- A - M[MA1]
		CMPA	O1,X	n,z,v,c	<--- A - M[O1+IX]
		CMPA	O1,Y	n,z,v,c	<--- A - M[O1+IY]
		CMPB	#ii	n,z,v,c	<--- B - ii
		CMPB	weiter wie bei	CMPA	
	CPD	CPD	#jjkk	n,z,v,c	<--- D - jjkk
<		CPD	MA1	n,z,v,c	<--- D - (M[MA1] ## M[MA1+1])
		CPD	O1,X	n,z,v,c	<--- D - (M[O1+IX] ## M[O1+IX+1])
		CPD	O1,Y	n,z,v,c	<--- D - (M[O1+IY] ## M[O1+IY+1])

	CPX	CPX	#jjkk	n,z,v,c	<--- IX - jjkk
<		CPX	MA1	n,z,v,c	<--- IX - (M[MA1] ## M[MA1+1])
		CPX	O1,X	n,z,v,c	<--- IX - (M[O1+IX] ## M[O1+IX+1])
		CPX	O1,Y	n,z,v,c	<--- IX - (M[O1+IY] ## M[O1+IY+1])
	CPY	CPY	#jjkk	n,z,v,c	<--- IY - jjkk
<		CPY	MA1	n,z,v,c	<--- IY - (M[MA1] ## M[MA1+1])
		CPY	O1,X	n,z,v,c	<--- IY - (M[O1+IX] ## M[O1+IX+1])
		CPY	O1,Y	n,z,v,c	<--- IY - (M[O1+IY] ## M[O1+IY+1])
	TST			Alle TST-Bef. setzen v,c mit 0	
		TSTA		n,z	<--- A - 00
		TSTB		n,z	<--- B - 00
		TST	MA1	n,z	<--- M[MA1] - 00
		TST	O1,X	n,z	<--- M[O1+IX] - 00
		TST	O1,Y	n,z	<--- M[O1+IY] - 00

2.3.1.5 Multiplikations- und Divisionsbefehle

MUL	D	<--- A * B
	C	<--- D_7 == B_7
FDIV	IX	<--- D / IX
	D	<--- Rest
	c=1,IX	<--- $ffff wenn IX == 0
	z	<--- 1 wenn D/IX == 0
	v=1,IX	<--- $ffff wenn IX <= D
IDIV	IX	<--- D / IX
	D	<--- Rest
	c=1,IX	<--- $ffff wenn IX == 0
	z	<--- 1 wenn D/IX == 0
	v	<--- 0

2.3.1.6 Increment- und Decrementbefehle

INC			(n,z,v	<--- opr)
	INCA		A	<--- A + 1
	INCB		B	<--- B + 1
	INC	MA1	M[MA1]	<--- M[MA1] + 1
	INC	O1,X	M[O1+IX]	<--- M[O1+IX] + 1
	INC	O1,Y	M[O1+IY]	<--- M[O1+IY] + 1
INS			SP	<--- SP + 1
			Flags werden nicht verändert	
INX			IX,z	<--- IX + 1
INY			IY,z	<--- IY + 1
DEC			(n,z,v	<--- opr)
	DECA		A	<--- A - 1
	DECB		B	<--- B - 1
	DEC	MA1	M[MA1]	<--- M[MA1] - 1
	DEC	O1,X	M[O1+IX]	<--- M[O1+IX] - 1
	DEC	O1,Y	M[O1+IY]	<--- M[O1+IY] - 1
DES			SP	<--- SP - 1
			Flags werden nicht verändert	
DEX			IX,z	<--- IX - 1

	DEY			IY,z	<--- IY - 1

2.3.2 Logikbefehle (v == 0 n,z <--- opr)

	AND	ANDA	#ii	A	<--- A & ii
<		ANDA	MA1	A	<--- A & M[MA1]
		ANDA	O1,X	A	<--- A & M[O1+IX]
		ANDA	O1,Y	A	<--- A & M[O1+IY]
		ANDB	#ii	B	<--- B & ii
		ANDB	weiter wie bei	ANDA	
	COM		c == 1	v == 0	n,z <--- opr
		COMA		A	<--- !A
		COMB		B	<--- !B
		COM	MA1	M[MA1]	<--- !M[MA1]
		COM	O1,X	M[O1+IX]	<--- !M[O1+IX]
		COM	O1,Y	M[O1+IY]	<--- !M[O1+IY]
	EOR	EORA	#ii	A	<--- A ^ ii
<		EORA	MA1	A	<--- A ^ M[MA1]
		EORA	O1,X	A	<--- A ^ M[O1+IX]
		EORA	O1,Y	A	<--- A ^ M[O1+IY]
		EORB	#ii	B	<--- B ^ ii
		EORB	weiter wie bei	EORA	
	ORA	ORAA	#ii	A	<--- A | ii
<		ORAA	MA1	A	<--- A | M[MA1]
		ORAA	O1,X	A	<--- A | M[O1+IX]
		ORAA	O1,Y	A	<--- A | M[O1+IY]
		ORAB	#ii	B	<--- B | ii
		ORAB	weiter wie bei	ORAA	

2.3.3 Schiebe- und Rotationsbefehle (n,z,v <--- opr)

ASL	ASLA		c	<<1A	<<1	00
	ASLB		c	<<1B	<<1	00
	ASL	MA1	c	<<1M[MA1]	<<1	00
	ASL	O1,X	c	<<1M[O1+IX]	<<1	00
	ASL	O1,Y	c	<<1M[O1+IY]	<<1	00
ASLD			c	<<1A <<1 B	<<1	00
ASR	ADRA		A_7	>>1A	>>1	c
	ASRB		B_7	>>1B	>>1	c
	ASR	MA1	$M[MA1]_7$	>>1M[MA1]	>>1	c
	ASR	O1,X	$M[O1+IX]_7$	>>1M[O1+IX]	>>1	c
	ASR	O1,Y	$M[O1+IY]_7$	>>1M[O1+IY]	>>1	c
LSL	LSLA		c	<<1A	<<1	00
	LSLB		c	<<1B	<<1	00
	LSL	MA1	c	<<1M[MA1]	<<1	00
	LSL	O1,X	c	<<1M[O1+IX]	<<1	00
	LSL	O1,Y	c	<<1M[O1+IY]	<<1	00
LSLD			c	<<1A <<1 B	<<1	00
LSR	LSRA		00	>>1A	>>1	c
	LSRB		00	>>1B	>>1	c
	LSR	MA1	00	>>1M[MA1]	>>1	c
	LSR	O1,X	00	>>1M[O1+IX]	>>1	c
	LSR	O1,Y	00	>>1M[O1+IY]	>>1	c

	LSRD			00	>>1A >>1 B	>>1	c
	ROL	ROLA		c	<<1A	<<1	c
		ROLB		c	<<1B	<<1	c
		ROL	MA1	c	<<1M[MA1]	<<1	c
		ROL	O1,X	c	<<1M[O1+IX]	<<1	c
		ROL	O1,Y	c	<<1M[O1+IY]	<<1	c
	ROR	RORA		c	>>1A	>>1	c
		RORB		c	>>1B	>>1	c
		ROR	MA1	c	>>1M[MA1]	>>1	c
		ROR	O1,X	c	>>1M[O1+IX]	>>1	c
		ROR	O1,Y	c	>>1M[O1+IY]	>>1	c

2.3.4 Bitbefehle

	BCLR			v == 0	n,z <--- opr	
<<		BCLR	mm,M1	M[M1]	<--- M[M1] & (!mm)	
		BCLR	mm,O1,X	M[O1+IX]	<--- M[O1+IX] & (!mm)	
		BCLR	mm,O1,Y	M[O1+IY]	<--- M[O1+IY] & (!mm)	
	BIT	BITA	#ii	n,z	<--- A & ii	
<		BITA	MA1	n,z	<--- A & M[MA1]	
		BITA	O1,X	n,z	<--- A & M[O1+IX]	
		BITA	O1,Y	n,z	<--- A & M[O1+IY]	
		BITB	#ii	n,z	<--- B & ii	
		BITB	weiter wie bei	BITA		
	BRCLR			Flags werden nicht verändert		
<<		BRCLR	mm,M1,rel	PC	<--- PC + 4 + rel	if M[M1] & mm == 00
		BRCLR	mm,O1,X,rel	PC	<--- PC + 4 + rel	if M[O1+IX] & mm == 00
		BRCLR	mm,O1,Y,rel	PC	<--- PC + 5 + rel	if M[O1+IY] & mm == 00
	BRSET			Flags werden nicht verändert		
<<		BRSET	mm,M1,rel	PC	<--- PC + 4 + rel	if M[M1] & mm == mm
		BRSET	mm,O1,X,rel	PC	<--- PC + 4 + rel	if M[O1+IX] & mm == mm
		BRSET	mm,O1,Y,rel	PC	<--- PC + 5 + rel	if M[O1+IY] & mm == mm
	BSET			v == 0	n,z <--- opr	
<<		BSET	mm,M1	M[M1]	<--- M[M1] | mm	
		BSET	mm,O1,X	M[O1+IX]	<--- M[O1+IX] | mm	
		BSET	mm,O1,Y	M[O1+IY]	<--- M[O1+IY] | mm	

2.3.5 Sprungbefehle (Flags werden außer bei SWI nicht verändert)

BCC	BCC	rel	PC	<--- PC + 2 + rel	if c==0
BCS	BCS	rel	PC	<--- PC + 2 + rel	if c==1
BEQ	BEQ	rel	PC	<--- PC + 2 + rel	if z==1
BGE	BGE	rel	PC	<--- PC + 2 + rel	if n^v==0
BGT	BGT	rel	PC	<--- PC + 2 + rel	if z|(n^v)==0
BHI	BHI	rel	PC	<--- PC + 2 + rel	if c|z==0
BHS	BHS	rel	PC	<--- PC + 2 + rel	if c==0
BLE	BLE	rel	PC	<--- PC + 2 + rel	if z|(n^v)==1
BLO	BLO	rel	PC	<--- PC + 2 + rel	if c==1

	BLS	BLS	rel	PC	<--- PC + 2 + rel	if c|z==1
	BLT	BLT	rel	PC	<--- PC + 2 + rel	if n^v==1
	BMI	BMI	rel	PC	<--- PC + 2 + rel	if n==1
	BNE	BNE	rel	PC	<--- PC + 2 + rel	if z==0
	BPL	BPL	rel	PC	<--- PC + 2 + rel	if n==0
	BRA	BRA	rel	PC	<--- PC + 2 + rel	
	BRCLR	siehe Bitbefehle				
	BRN	BRN	rel	PC	<--- PC + 2	
	BRSET	siehe Bitbefehle				
	BSR	BSR	rel	PC	<--- PC + 2	
				M[SP]	<--- PC_{LOW}	
				SP	<--- SP - 1	
				M[SP]	<--- PC_{HIGH}	
				SP	<--- SP - 1	
				PC	<--- PC + rel	
	BVC	BVC	rel	PC	<--- PC + 2 + rel	if v==0
	BVS	BVS	rel	PC	<--- PC + 2 + rel	if v==1
	JMP	JMP	MA1	PC	<--- MA1	
		JMP	O1,X	PC	<--- O1 + IX	
		JMP	O1,Y	PC	<--- O1 + IY	
<	JSR	JSR	M1	PC	<--- PC + 2	
				M[SP]	<--- PC_{LOW}	
				SP	<--- SP - 1	
				M[SP]	<--- PC_{HIGH}	
				SP	<--- SP - 1	
				PC	<--- 0^8 ## M1	
		JSR	MA1	PC	<--- PC + 3	
				M[SP]	<--- PC_{LOW}	
				SP	<--- SP - 1	
				M[SP]	<--- PC_{HIGH}	
				SP	<--- SP - 1	
				PC	<--- MA1	
		JSR	O1,X	PC	<--- PC + 2	
				M[SP]	<--- PC_{LOW}	
				SP	<--- SP - 1	
				M[SP]	<--- PC_{HIGH}	
				SP	<--- SP - 1	
				PC	<--- O1 + IX	
		JSR	O1,Y	PC	<--- PC + 3	
				M[SP]	<--- PC_{LOW}	
				SP	<--- SP - 1	
				M[SP]	<--- PC_{HIGH}	
				SP	<--- SP - 1	
				PC	<--- O1 + IY	

RTI			SP	<--- SP + 01
			CCR	<--- M[SP]
			SP	<--- SP + 01
			B	<--- M[SP]
			SP	<--- SP + 01
			A	<--- M[SP]
			SP	<--- SP + 01
			IX_{HIGH}	<--- M[SP]
			SP	<--- SP + 01
			IX_{LOW}	<--- M[SP]
			SP	<--- SP + 01
			IY_{HIGH}	<--- M[SP]
			SP	<--- SP + 01
			IY_{LOW}	<--- M[SP]
			SP	<--- SP + 01
			PC_{HIGH}	<--- M[SP]
			SP	<--- SP + 01
			PC_{LOW}	<--- M[SP]
RTS			SP	<--- SP + 01
			PC_{HIGH}	<--- M[SP]
			SP	<--- SP + 01
			PC_{LOW}	<--- M[SP]
SWI			PC	<--- PC + 01
			M[SP]	<--- PC_{LOW}
			SP	<--- SP - 01
			M[SP]	<--- PC_{HIGH}
			SP	<--- SP - 01
			M[SP]	<--- IY_{LOW}
			SP	<--- SP - 01
			M[SP]	<--- IY_{HIGH}
			SP	<--- SP - 01
			M[SP]	<--- IX_{LOW}
			SP	<--- SP - 01
			M[SP]	<--- IX_{HIGH}
			SP	<--- SP - 01
			M[SP]	<--- A
			SP	<--- SP - 01
			M[SP]	<--- B
			SP	<--- SP - 01
			M[SP]	<--- CCR
			i	<--- 1
			PC	<--- SWIvector

2.3.6 Transportbefehle

CLR			n,v,c == 0	z == 1
	CLRA		A	<--- 00
	CLRB		B	<--- 00
	CLR	MA1	M[MA1]	<--- 00
	CLR	O1,X	M[O1+IX]	<--- 00
	CLR	O1,Y	M[O1+IY]	<--- 00
LDA			v == 0	n,z <--- opr
	LDAA	#ii	A	<--- ii
<	LDAA	MA1	A	<--- M[MA1]
	LDAA	O1,X	A	<--- M[O1+IX]
	LDAA	O1,Y	A	<--- M[O1+IY]
	LDAB	#ii	B	<--- ii
	LDAB	weiter wie LDAA		

	LDD			v == 0	n,z <--- opr
		LDD	#jjkk	A ## B	<--- jj ## kk
<		LDD	MA1	A ## B	<--- M[MA1] ## M[MA1+1]
		LDD	O1,X	A ## B	<--- M[O1+IX] ## M[O1+IX+1]
		LDD	O1,Y	A ## B	<--- M[O1+IY] ## M[O1+IY+1]
	LDS			v == 0	n,z <--- opr
		LDS	#jjkk	SP	<--- jj ## kk
<		LDS	MA1	SP	<--- M[MA1] ## M[MA1+1]
		LDS	O1,X	SP	<--- M[O1+IX] ## M[O1+IX+1]
		LDS	O1,Y	SP	<--- M[O1+IY] ## M[O1+IY+1]
	LDX			v == 0	n,z <--- opr
		LDX	#jjkk	IX	<--- jj ## kk
<		LDX	MA1	IX	<--- M[MA1] ## M[MA1+1]
		LDX	O1,X	IX	<--- M[O1+IX] ## M[O1+IX+1]
		LDX	O1,Y	IX	<--- M[O1+IY] ## M[O1+IY+1]
	LDY			v == 0	n,z <--- opr
		LDY	#jjkk	IY	<--- jj ## kk
<		LDY	MA1	IY	<--- M[MA1] ## M[MA1+1]
		LDY	O1,X	IY	<--- M[O1+IX] ## M[O1+IX+1]
		LDY	O1,Y	IY	<--- M[O1+IY] ## M[O1+IY+1]
	PSH			Flags werden nicht verändert	
		PSHA		M[SP]	<--- A
				SP	<--- SP - 01
		PSHB		M[SP]	<--- B
				SP	<--- SP - 01
		PSHX		M[SP]	<--- IX_{LOW}
				SP	<--- SP - 01
				M[SP]	<--- IX_{HIGH}
				SP	<--- SP - 01
		PSHY		M[SP]	<--- IY_{LOW}
				SP	<--- SP - 01
				M[SP]	<--- IY_{HIGH}
				SP	<--- SP - 01
	PUL			Flags werden nicht verändert	
		PULA		SP	<--- SP + 01
				A	<--- M[SP]
		PULB		SP	<--- SP + 01
				B	<--- M[SP]
		PULX		SP	<--- SP + 01
				IX_{HIGH}	<--- M[SP]
				SP	<--- SP + 01
				IX_{LOW}	<--- M[SP]
		PULY		SP	<--- SP + 01
				IY_{HIGH}	<--- M[SP]
				SP	<--- SP + 01
				IY_{LOW}	<--- M[SP]

	STA			v == 0	n,z <--- opr
<		STAA	MA1	M[MA1]	<--- A
		STAA	O1,X	M[O1+IX]	<--- A
		STAA	O1,Y	M[O1+IY]	<--- A
		STAB	MA1	M[MA1]	<--- B
		STAB	weiter wie STAA		
	STD			v == 0	n,z <--- opr
<		STD	MA1	M[MA1]	<--- A
				M[MA1+1]	<--- B
		STD	O1,X	M[O1+IX]	<--- A
				M[O1+IX+1]	<--- B
		STD	O1,Y	M[O1+IY]	<--- A
				M[O1+IY+1]	<--- B
	STS			v == 0	n,z <--- opr
<		STS	MA1	M[MA1]	<--- SP_{HIGH}
				M[MA1+1]	<--- SP_{LOW}
		STS	O1,X	M[O1+IX]	<--- SP_{HIGH}
				M[O1+IX+1]	<--- SP_{LOW}
		STS	O1,Y	M[O1+IY]	<--- SP_{HIGH}
				M[O1+IY+1]	<--- SP_{LOW}
	STX			v == 0	n,z <--- opr
<		STX	MA1	M[MA1]	<--- IX_{HIGH}
				M[MA1+1]	<--- IX_{LOW}
		STX	O1,X	M[O1+IX]	<--- IX_{HIGH}
				M[O1+IX+1]	<--- IX_{LOW}
		STX	O1,Y	M[O1+IY]	<--- IX_{HIGH}
				M[O1+IY+1]	<--- IX_{LOW}
	STY			v == 0	n,z <--- opr
<		STY	MA1	M[MA1]	<--- IY_{HIGH}
				M[MA1+1]	<--- IY_{LOW}
		STY	O1,X	M[O1+IX]	<--- IY_{HIGH}
				M[O1+IX+1]	<--- IY_{LOW}
		STY	O1,Y	M[O1+IY]	<--- IY_{HIGH}
				M[O1+IY+1]	<--- IY_{LOW}
	TAB			v == 0	n,z <--- opr
				B	<--- A
	TAP	s ## x ## h ## i ## n ## z ## v ## c			<--- A
	TBA			v == 0	n,z <--- opr
				A	<--- B
	TPA			Flags werden nicht verändert	
				A	<--- s ## x ## h ## i ## n ## z ## v ## c
	TSX			Flags werden nicht verändert	
				IX	<--- SP + 01
	TSY			Flags werden nicht verändert	
				IY	<--- SP + 01
	TXS			Flags werden nicht verändert	
				SP	<--- IX - 01
	TYS			Flags werden nicht verändert	
				SP	<--- IY - 01

XGDX	Flags werden nicht verändert		
	IX	<-->	D
XGDY	Flags werden nicht verändert		
	IY	<-->	D

2.3.7 Steuerbefehle

CLC	c	<--- 0
CLI	i	<--- 0
CLV	v	<--- 0
NOP		
SEC	c	<--- 1
SEI	i	<--- 1
SEV	v	<--- 1
STOP	Standby Mode ein, wenn s-Bit ==0, sonst wie NOP	
WAI	PC	<--- PC + 01
	M[SP]	<--- PC_{LOW}
	SP	<--- SP - 01
	M[SP]	<--- PC_{HIGH}
	SP	<--- SP - 01
	M[SP]	<--- IY_{LOW}
	SP	<--- SP - 01
	M[SP]	<--- IY_{HIGH}
	SP	<--- SP - 01
	M[SP]	<--- IX_{LOW}
	SP	<--- SP - 01
	M[SP]	<--- IX_{HIGH}
	SP	<--- SP - 01
	M[SP]	<--- A
	SP	<--- SP - 01
	M[SP]	<--- B
	SP	<--- SP - 01
	M[SP]	<--- CCR