



MOTOROLA

MC 6809

Befehlssatz

Otto-von-Guericke-Universität Magdeburg



Unzulänglichkeiten des Modellrechners

- sehr kleiner Instruktionssatz
 - keine Konstanten
 - viele Speicherzugriffe
 - nur eine Bedingung direkt abprüfbar
 - nur eine Adressierungsart (direkt)
- **wenige, einfache Befehle**
 - **simples Speichermodell**
 - **kleiner Adressraum**
 - **keine Unterstützung leistungsfähiger Adressierungsarten**
- einige Probleme nur mit selbst-modifizierendem Code lösbar
z.B. Unterprogrammprung, Zeiger, Indizierung
(kann nicht in ROM abgelegt werden)
 - kleiner, wortorientierter Speicher



Warum MC 6809?

überschaubarer aber repräsentativer Befehlssatz

alle wesentliche (Speicher-) Adressierungsarten

orthogonale Architektur

gute Basis, um Erweiterungen einzuführen



Orthogonalität einer Architektur:

Grad der Unabhängigkeit einzelner Komponenten oder Eigenschaften

Beisp.:

Befehlssatz orthogonal zu Adressierungsarten bedeutet, daß Befehle und Adressierungsarten unabhängig voneinander spezifiziert, und daher beliebig kombiniert werden können.

Register orthogonal zu Befehlssatz bedeutet, daß alle Befehle auf alle Register angewandt werden können

VAX: Orthogonalität war das Entwurfsziel z. B. der DEC VAX. Sie hat einige hundert Befehle, die, wenn man sie mit den verfügbaren Datentypen, den Adressierungsarten (22), und der Spezifikation der Anzahl von Operanden kombiniert, einige hunderttausend verschiedene Operationen ergibt.

6809: 59 Mnemonische Codes, 268 Opcodes, 10 (Haupt-)Adressierungsarten , 24 Unterarten (Indexed), 1464 unterschiedliche Operationen

68020: ca. 100 Mnemonische Codes, 18 (Haupt-)Adressierungsarten

zum Vergleich:

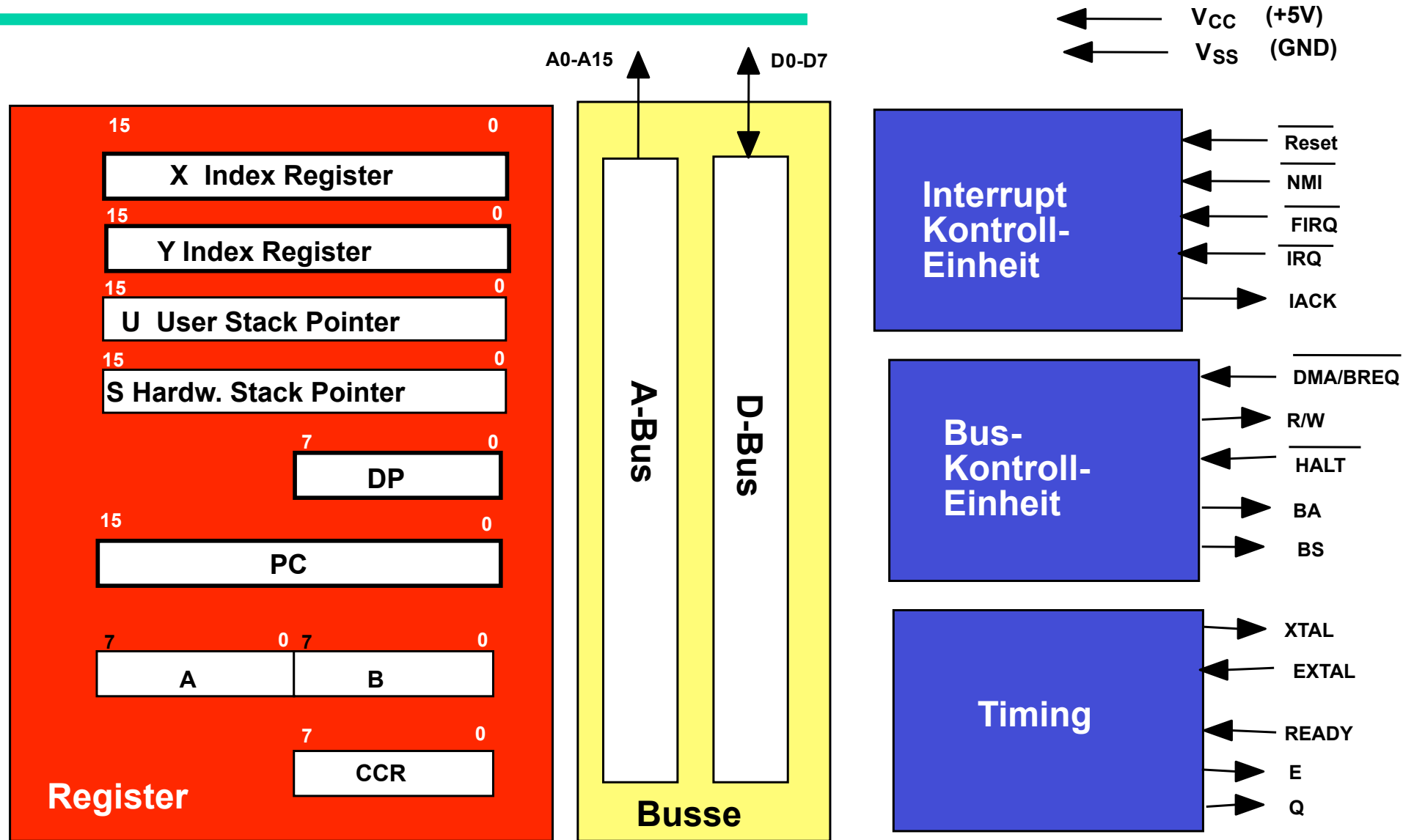
- Der RISC-Prozessor MIPS hat 4 Adressierungsarten
- SPARC hat 4 Adressierungsarten für Daten (3 Mem, 1 Reg.) und jeweils 1 Adressierungsart für Sprünge und den Unterprogrammaufruf

6809 allgemeine Eigenschaften:

- **Eingeschränkte 16 Bit Architektur bei 8 Bit Datenbus, 16 Bit Adreßbus**
- **16 Bit Adressen, d.h. 64 kB adressierbarer Speicher**
- **2 8-Bit allgemeine Datenregister, die als ein 16-Bit Register verwendet werden können**
- **2 16-Bit Indexregister**
- **2 16-Bit Stackpointer**
- **59 Basisbefehle, 268 Operationscodes**
- **10 Basis-Adressierungsarten**
- **Unterstützung positionsunabhängigen Codes**
- **Leistungsfähiger Instruktionssatz (für einen 8-Bit Prozessor)**



Komponenten des Motorola 6809



Programmiermodell: beschreibt die Architektur eines Rechners auf der für seine Programmierung relevanten Abstraktionsebene

Programmiermodell für die Programmierung von Anwendungen:

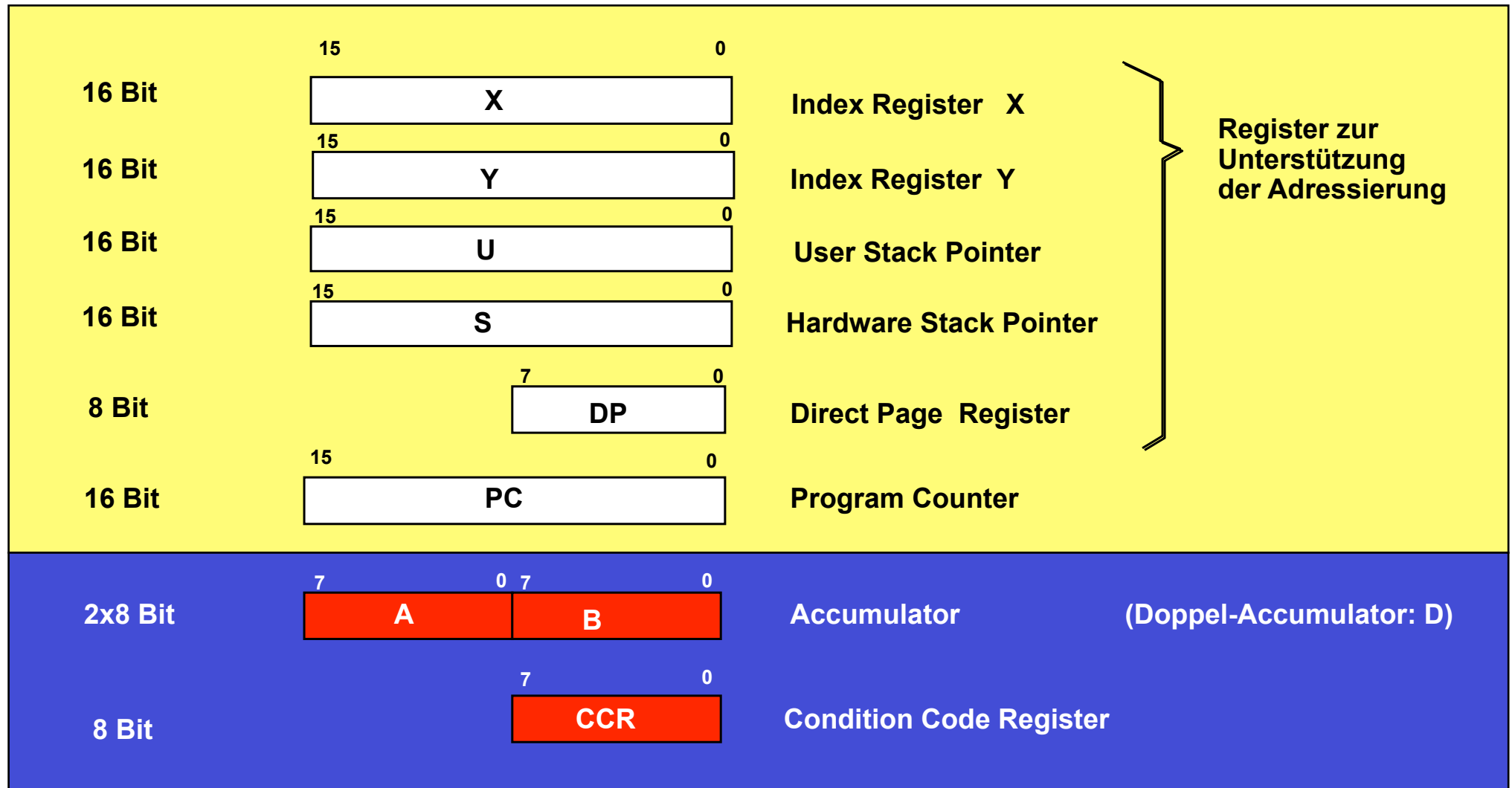
- Speicherorganisation
- Registersatz
- Befehlssatz
- Befehlsformat
- Auswahl der Operanden (Adressierung)
- vom Rechner unterstützte Datentypen

Programmiermodell für die Programmierung von Systemprogrammen:

- Speicherverwaltung
- Zugriffsschutz
- Unterbrechungs- und Ausnahmebehandlung
- Multitasking
- Ein/Ausgabe
- Initialisierung des Prozessors
- Coprozessorschnittstellen und Protokolle
- Unterbrechungs- und Ausnahmebehandlung

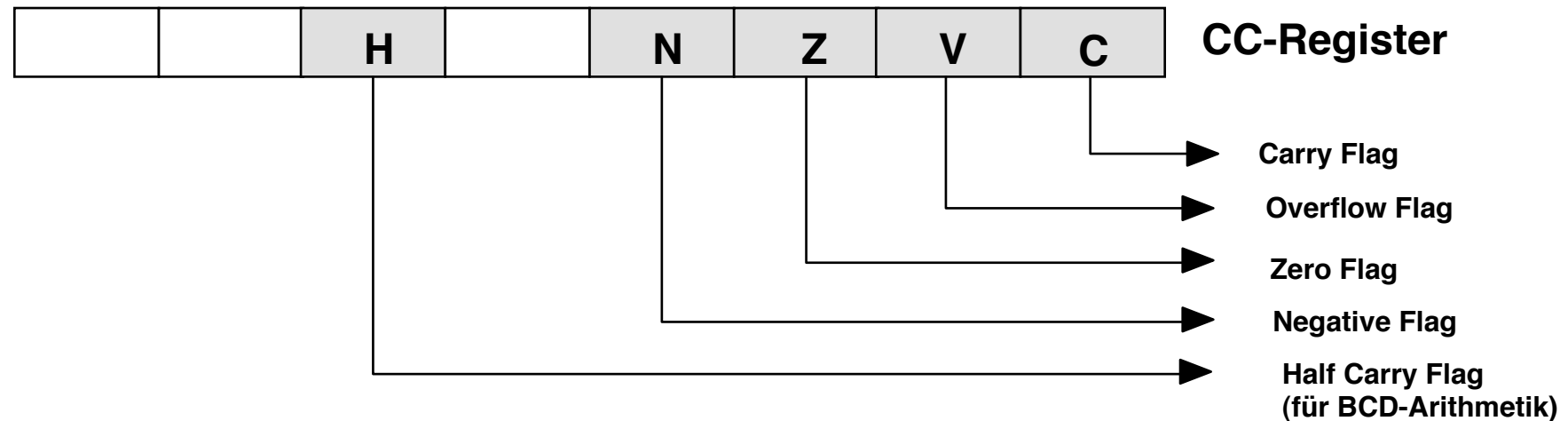


Programmiermodell des Registersatzes des Motorola 6809 Prozessors

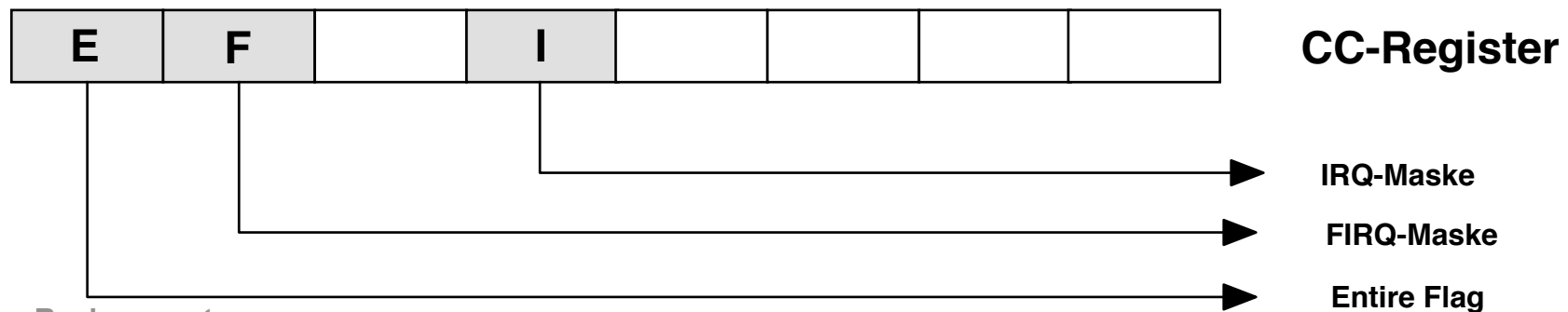


Prozessor-Status: Die "Condition Codes" des 6809

Condition Codes für Arithmetische/Logische Operationen



Condition Codes zur Unterbrechungsbehandlung (Interrupt)



-
- **Befehlssatz**
 - **Speicherorganisation**
 - **Adressierungsarten**



Der 6809 Befehlssatz

- **Arithmetische Befehle**
- **Logische Befehle**
- **Shift Befehle**
- **Daten Transfer Befehle**
- **Masken- und Vergleichsbefehle**
- **Stackpointer Befehle**
- **Index Register Befehle**
- **Sprungbefehle**
- **“Verschiedenes“**

Datenmanipulation

Adreß manipulation

Kontrollflußmanipulation

Systembezogene Befehle



6809 Befehlssatz (1):

Arithmetische Befehle:

Addition:	8	ADCA, ADCB	Add memory to Accumulator (A,B) with Carry
	16	ADDA, ADDB ADDD	Add memory to Accumulator (A,B) Add memory to Double Accumulator (D)
BCD-Arithm.:	8	DAA	Decimal Adjust Accumulator (A)
Increment:	8	INC, INCA, INCB	Increment memory location bzw. Acc. (A,B)
Decrement:	8	DEC, DECA, DECB	Decrement memory location bzw. Acc. (A,B)
Multiplikation:	8	MUL	unsigned multiply : $D \leftarrow A \times B$
Vorz. Erw.:	16	SEX	Sign Extend B Acc. into A Acc.
Subtraktion:	8	SBCA, SBCB SBA	Subtract memory from Acc. (A,B) with Borrow Subtract Acc.B from Acc A
	8/16	SUBA, SUBB, SUBD	Subtract memory from Acc. A,B,D (without Borrow)



6809 Befehlssatz (2):

Logische Befehle:

8	ANDA, ANDB	And memory to Accumulator (A,B)
8	COM, COMA, COMB	Complement memory, Accumulator (A,B)
8	EORA, EORB	Exclusiv OR memory with Accumulator (A,B)
8	NEG, NEGA, NEGB	Negate memory or Acc. (A,B) (2's complement)
8	ORA, ORB	OR memory with Accumulator (A,B)

Shift Befehle:

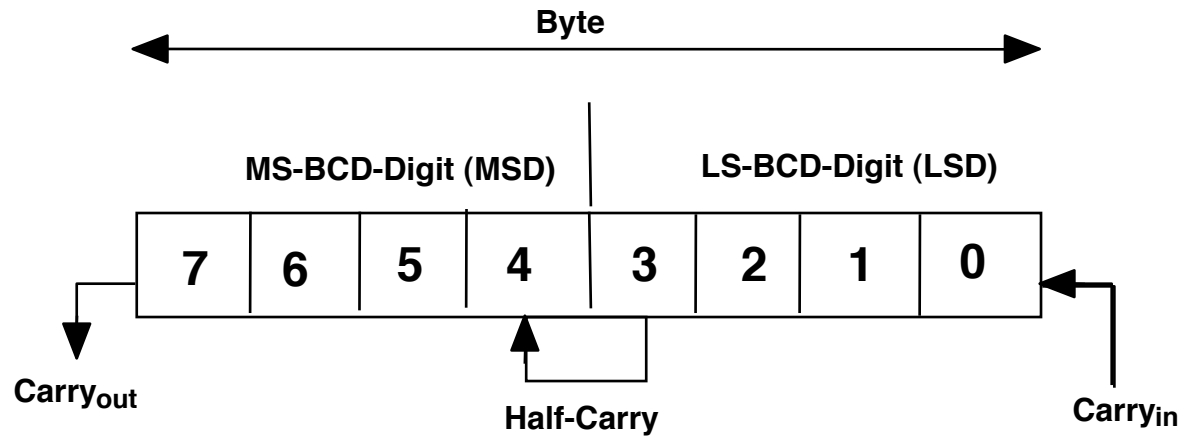
8	ASL, ASLA, ASLB	Arithmetic Shift Left memory or Accumulator (A,B)
8	ASR, ASRA, ASRB	Arithmetic Shift Right memory or Accumulator (A,B)
8	LSL, LSLA, LSLB	Logical Shift Left memory or Accumulator (A,B)
8	LSR, LSRA, LSRB	Logical Shift Right memory or Accumulator (A,B)
8	ROL, ROLA, ROLB	Rotate Left memory or Accumulator (A,B)
8	ROR, RORA, RORB	Rotate Right memory or Accumulator (A,B)

Daten Transfer:

8	EXG R1, R2	Exchange R1 with R2 (R1, R2 : A,B, CC, DP)
16	EXG R1, R2	Exchange R1 with R2 (R1, R2 :D, X, Y, U, S, PC)
8	LDA, LDB	Load Accumulator (A,B) from memory
16	LDx (x: D, X, Y, U, S)	Load Register from memory
8	STA, STB	Store Accumulator (A,B) to memory
16	STx (x: D, X, Y, U, S)	Store Register to memory
8	TFR, R1,R2	Transfer R1 to R2 (R1, R2 : A,B, CC, DP)
16	TFR, D,R	Transfer D to X, Y, S, U, PC
16	TFR, R, D	Transfer X, Y, S, U, PC to D

Die Instruktion

DAA



1. Fall: $\text{sum}_{\text{LSD}} \leq 9$

keine Korrektur erforderlich

2. Fall: $9 < \text{sum}_{\text{LSD}} \leq 15$

Half-Carry = 0

10	→	0	(10 + 6) mod 16	Half Carry = 1
11	→	1	(11 + 6) mod 16	Half Carry = 1
12	→	2	(12 + 6) mod 16	Half Carry = 1
.....				
15	→	5	(15 + 6) mod 16	Half Carry = 1

Korrektur durch Addition von +6

3. Fall $15 < \text{sum}_{\text{LSD}} \leq 18$

Half-Carry = 1

16	→	6	(0 + 6) mod 16	Half Carry = 0
17	→	7	(1 + 6) mod 16	Half Carry = 0
18	→	8	(2 + 6) mod 16	Half Carry = 0

Bedingung für die Korrektur des LSD : $(\text{sum}_{\text{LSD}} > 9)$ ODER $(H=1)$

Bedingung für die Korrektur des MSD : $C_{\text{out}} = 1$ ODER $(\text{sum}_{\text{MSD}} > 8)$ UND $(\text{sum}_{\text{LSD}} > 9)$

Beispiel: Binäre Addition zweier BCD-Zahlen mit Korrektur durch DAA

$$\begin{array}{r} 0010\ 1001 \quad 29 \\ + 0011\ 0110 \quad + 36 \\ \hline 0101\ 1111 \quad \neq 65 \end{array}$$

Bedingung für die Korrektur des LSD : $(\text{sum}_{\text{LSD}} > 9)$ ODER $(H=1)$

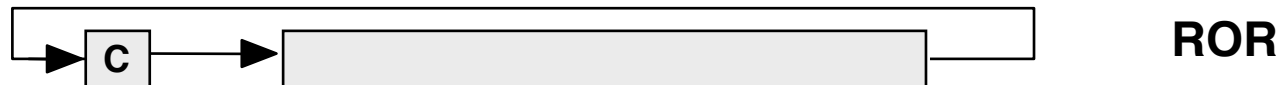
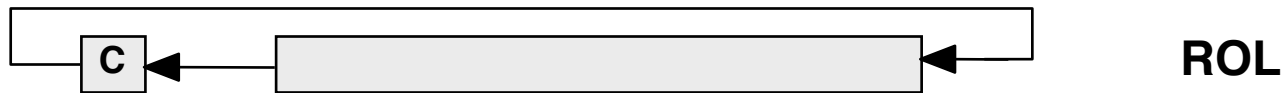
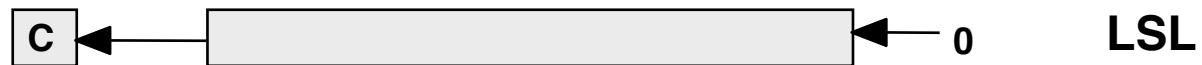
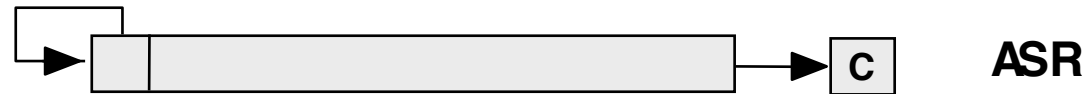
Bedingung für die Korrektur des MSD : $C_{\text{out}}=1$ ODER $(\text{sum}_{\text{MSD}} > 8)$ UND $(\text{sum}_{\text{LSD}} > 9)$

$$\begin{array}{r} 0101\ 1111 \\ + 0000\ 0110 \quad +6 \\ \hline 0110\ 0101 \quad = 65 \end{array}$$

↖
Half Carry



Shift Befehle des 6809



6809 Befehlssatz (3):

Index Register Befehle:

Ladebefehle	LDX, LDY STX, STY LEAX, LEAY	Load Index Register from memory (X,Y) Store Index Register to memory (X,Y) Load Effective Address into Index Reg. (X,Y)
Vergleichsbefehl	CMPX, CMPY	Compare memory from Index Reg. (X,Y)
Adreßberechnung	ABX	Add Acc.B to Index Reg X (unsigned)

Stackpointer Befehle:

Ladebefehle	LDS, LDU STS, STU LEAS, LEAU	Load Stackpointer from memory (S,U) Store Stackpointer to memory (S,U) Load Effective Address into Stackpointer (S,U)
Vergleichsbefehl	CMPS, CMPU	Compare memory from Stackpointer (S,U)
Stackoperationen	PSHS PSHU PULS PULU	Push any register(s) onto system stack (except S) Push any register(s) onto user stack (except U) Pull any register(s) from system stack (except S) Pull any register(s) from user stack (except U)

Transferbefehle	EXG R1, R2 TFR R1, R2	Exchange D, X, Y, S, U, PC with D, X, Y, S, U, PC Transfer D, X, Y, S, U, PC to D, X, Y, S, U, PC
-----------------	--------------------------	--

6809 Befehlssatz (4):

Masken- und Vergleichsbefehle:

8	BITA, BITB	Bit test memory with Accumulator (A,B)
8	CMPA, CMPB	Compare memory with Accumulator (A,B)
8	TST, TSTA, TSTB	Test memory location or Acc. (A,B)
16	CMPx x=D, S, U, X, Y,	Compare memory with Register x

Löschbefehl:

8	CLR, CLRA, CLRB	Clear memory bzw. Accumulator (A,B)
---	-----------------	-------------------------------------



Beispiel für ein „Klingelbrett“:

```
loop    JRS    wait n sec
        LDA    bellboard
        BITA   #$FF
        BEQ    loop
out     JSR
        BRA    loop
```

die Maske "1111 1111" testet, ob irgend eine Klingel gedrückt wurde
falls Z=1, d.h. im "bellboard" befindet sich keine "1", dann zurück nach loop
Sprung zur Ausgabe von A



Beispiele: Vergleichsbefehle CMPA mem

A = 37_{10} (0010 0101) A > mem !
mem = 19_{10} (0001 0011)

	0010 0101	
	<u>1110 1101</u>	2er-Komplement von 19
1	0001 0010	

Folgende Belegung des CC-Registers wird erzeugt:

N	Z	V	C
0	0	0	0

Das Carry-Flag wird komplementiert

A = 39_{10} (0010 0111) A < mem !
mem = 67_{10} (0100 0011)

	0010 0111	
	<u>1011 1101</u>	2er-Komplement von 67
0	1110 0100	

Folgende Belegung des CC-Registers wird erzeugt:

N	Z	V	C
1	0	0	1

Das Carry-Flag wird komplementiert



Beispiele: Vergleichsbefehle

$A = A3_{16} (1010\ 0011_2, -93_{10})$
 $mem = 64_{16} (0110\ 0100_2, 100_{10})$

$A < mem !$

1010 0011
1001 1100
1 0011 1111

2er-Komplement von 64

N Z V C
0 0 1 0

Das Carry-Flag wird komplementiert



6809 Befehlssatz (5): Branch Befehle

Unbedingte Sprünge:

BRA, LBRA	Branch always	unbedingter Sprung
BRN, LBRN	Branch never	NOP
BSR, LBSR	Branch to Subroutine	Unterprogramm sprung

Bedingte Sprünge:

BCC, LBCC	Branch if Carry Clear
BCS, LBCS	Branch if Carry Set
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BVC, LBVC	Branch if overflow clear
BVS, LBVS	Branch if overflow set

Bedingte Sprünge nach einer Vergleichsoperation:

Bedingte Sprünge, die das Vorzeichen interpretieren (signed):

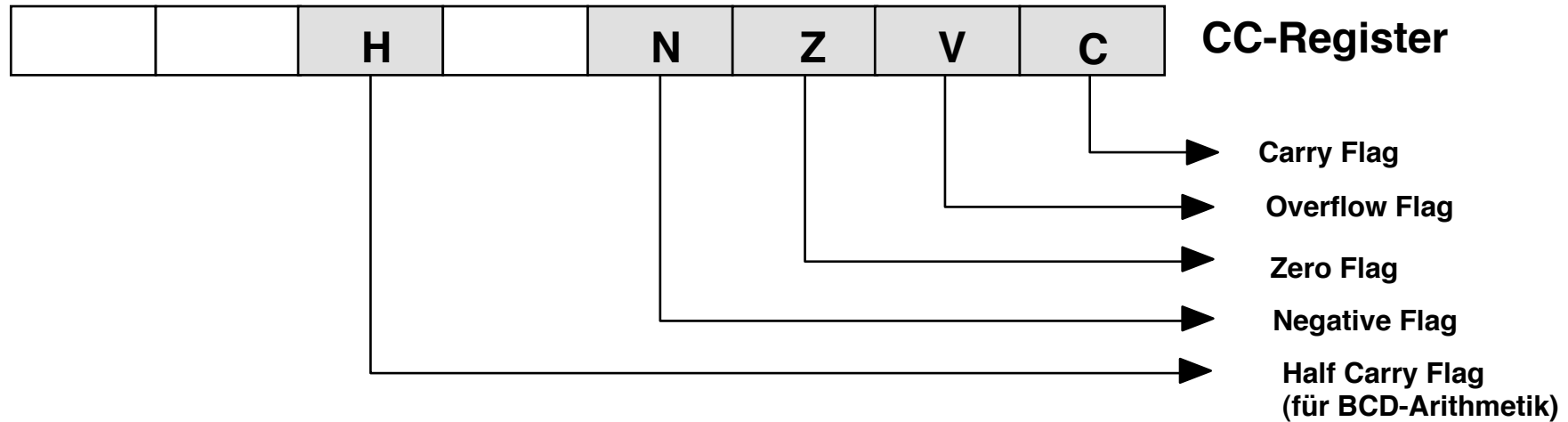
BGE, LBGE	Branch if greater than or equal
BLE, LBLE	Branch if less than or equal
BGT, LBGT	Branch if greater
BLT, LBLT	Branch if less than

Bedingte Sprünge, die das Vorzeichen nicht interpretieren (unsigned):

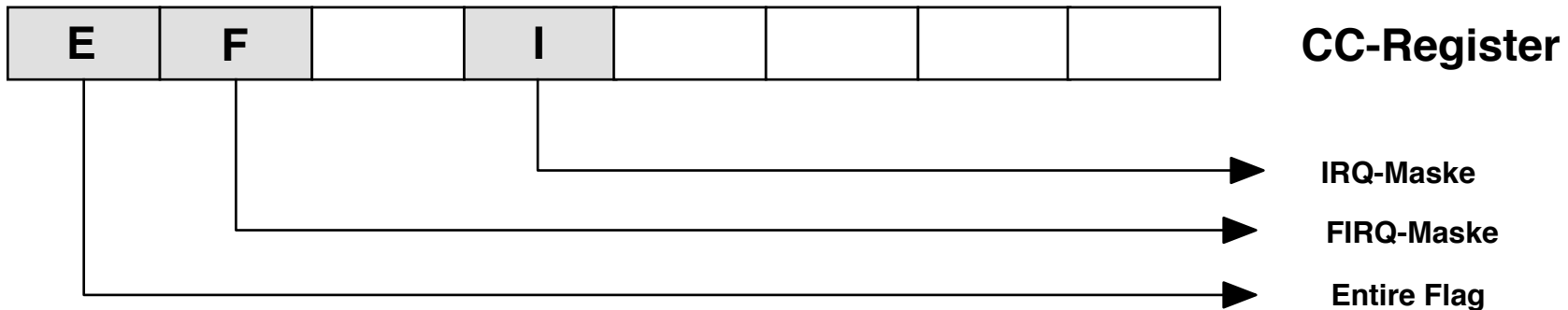
BHS, LBHS	Branch if higher or same
BLS, LBLs	Branch if lower or same
BHI, LBHI	Branch if higher
BLO, LBLO	Branch if lower

Bedingungs-Codes (Condition Codes) des 6809

Condition Codes für Arithmetische/Logische Operationen



Condition Codes zur Unterbrechungsbehandlung (Interrupt)



Bedingte Sprünge im 6809 und die entsprechende Belegung der Condition Codes

Bedingung	mit Vorz.	ohne Vorz.	Branch Mnemonic Bxx	Flags
$a \neq b$	x	x	BNE	$Z = 0$
$a = b$	x	x	BEQ	$Z = 1$
$a \leq b$	x		BLE	$Z + (N \oplus V) = 1$
$a < b$	x		BLT	$N \oplus V = 1$
$a \geq b$	x		BGE	$N \oplus V = 0$
$a > b$	x		BGT	$Z + (N \oplus V) = 0$
$a \leq b$		x	BLS	$Z + C = 1$
$a < b$		x	BLO	$C = 1$
$a \geq b$		x	BHS	$C = 0$
$a > b$		x	BHI	$Z + C = 0$
			BCC	$C = 0$
			BCS	$C = 1$
			BEQ	$Z = 1$
			BNE	$Z = 0$
			BMI	$N = 1$
			BPL	$N = 0$
			BVC	$V = 0$
			BVS	$V = 1$

Beispiele: Bedingte Sprünge nach Vergleichsbefehl

$A = 37_{10}$ (0010 0101) $A > \text{mem} !$
 $\text{mem} = 19_{10}$ (0001 0011)

	0010 0101	
	<u>1110 1101</u>	2er-Komplement von 19
1	0001 0010	Das Carry-Flag wird komplementiert

Folgende Belegung des CC-Registers wird erzeugt:

N	Z	V	C
0	0	0	0

$A = 39_{10}$ (0010 0111) $A < \text{mem} !$
 $\text{mem} = 67_{10}$ (0100 0011)

	0010 0111	
	<u>1011 1101</u>	2er-Komplement von 67
0	1110 0100	Das Carry-Flag wird komplementiert

Folgende Belegung des CC-Registers wird erzeugt:

N	Z	V	C
1	0	0	1

BNE	Z = 0
BEQ	Z = 1
BLE	Z + (N ⊕ V) = 1
BLT	N ⊕ V = 1
BGE	N ⊕ V = 0
BGT	Z + (N ⊕ V) = 0
BLS	Z + C = 1
BLO	C = 1
BHS	C = 0
BHI	Z + C = 0



Beispiele: Bedingte Sprünge nach Vergleichsbefehl

A = A3₁₆ (1010 0011₂, -93₁₀) A < mem !
 mem = 64₁₆ (0110 0100₂, 100₁₀)

	1010 0011	
	<u>1001 1100</u>	
1	0011 1111	2er-Komplement von 64

N	Z	V	C
0	0	1	0

Das Carry-Flag wird komplementiert

BNE	Z = 0
BEQ	Z = 1
BLE	Z + (N ⊕ V) = 1
BLT	N ⊕ V = 1
BGE	N ⊕ V = 0
BGT	Z + (N ⊕ V) = 0
BLS	Z + C = 1
BLO	C = 1
BHS	C = 0
BHI	Z + C = 0



6809 Befehlssatz (6) :

Befehle, die sich auf das Condition Code Register beziehen:

ANDCC	And CC with Postbyte (immediate)
CWAI	And CC and wait for Interrupt
ORCC	OR CC with Postbyte (immediate)

Sprung- und Unterprogramm sprung- bezogene Befehle:

JMP	Jump	im Gegensatz zu Branch ist JMP ein absoluter Sprung
JSR	Jump to Subroutine	
RTS	Return from Subroutine	

Interrupt- bezogene Befehle:

RTI	Return from Interrupt
SWI, SWI2, SWI3	Software interrupt
SYNC	Synchronize with interrupt line

