MOTOROLA
MC 6809

Befehlssatz

Otto-von-Guericke-Universität Magdeburg

Unzulänglichkeiten des Modellrechners

- sehr kleiner Instruktionssatz
- keine Konstanten
- viele Speicherzugriffe
- keine Möglichkeit etwas in Registern zwischenzuspeichern
- nur eine Bedingung direkt abprüfbar
- nur eine Adressierungsart (direkt)
- einige Probleme nur mit selbst-modifizierendem Code lösbar
  z.B. Unterprogrammsprung, Zeiger, Indizierung
  (kann z.B. nicht in ROM abgelegt werden)
- Wortorientierter Speicher - keine unterschiedlich langen Befehle
- kleiner Adreßraum

- **wenige, einfache Befehle**
- **simples Speichermodell**
- **keine Unterstützung leistungsfähiger Adressierung**

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 Designziel 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

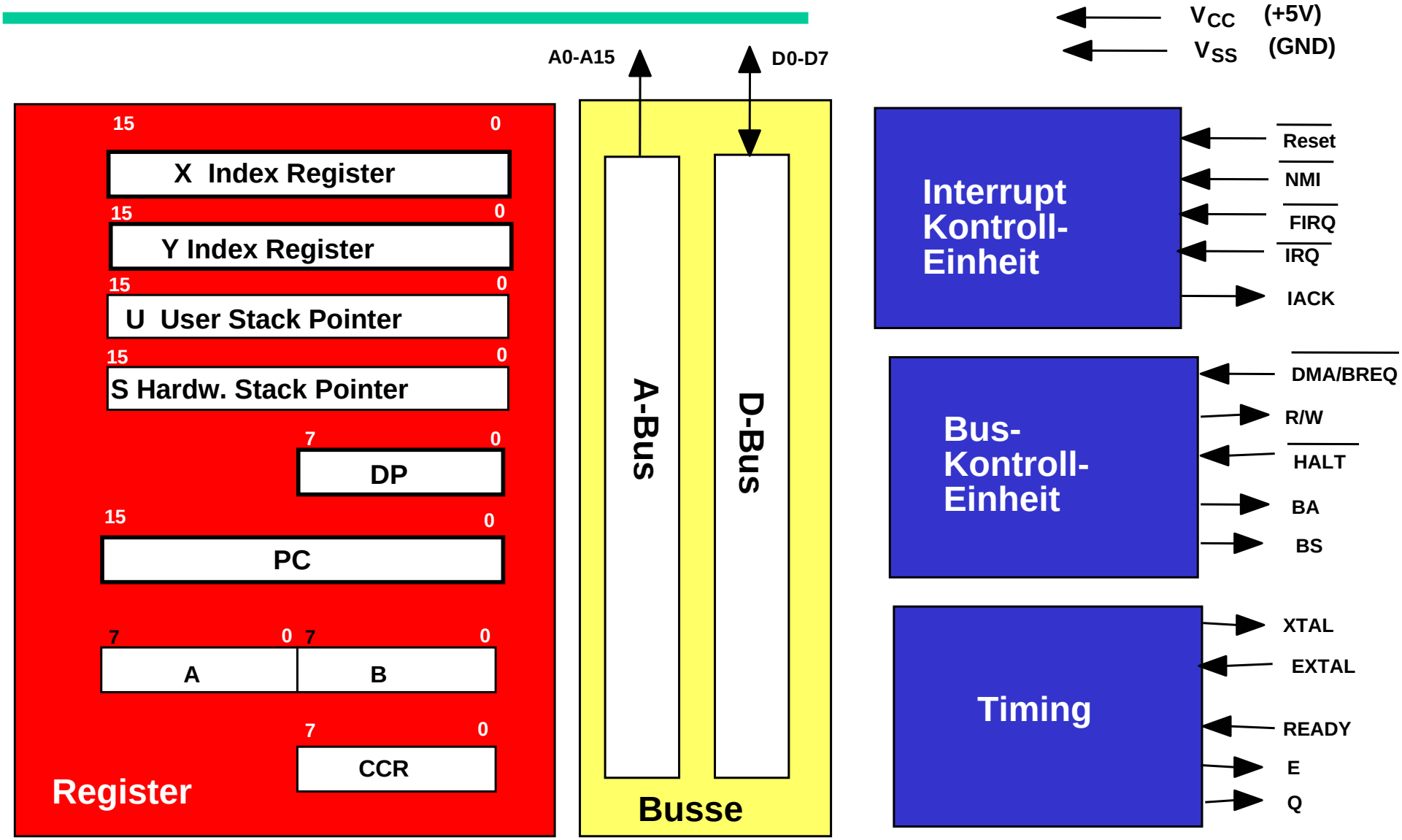
68020: ca. 100 Mnemonische Codes, 18 (Haupt-)Adressierungsarten

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 Motrola 6809



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

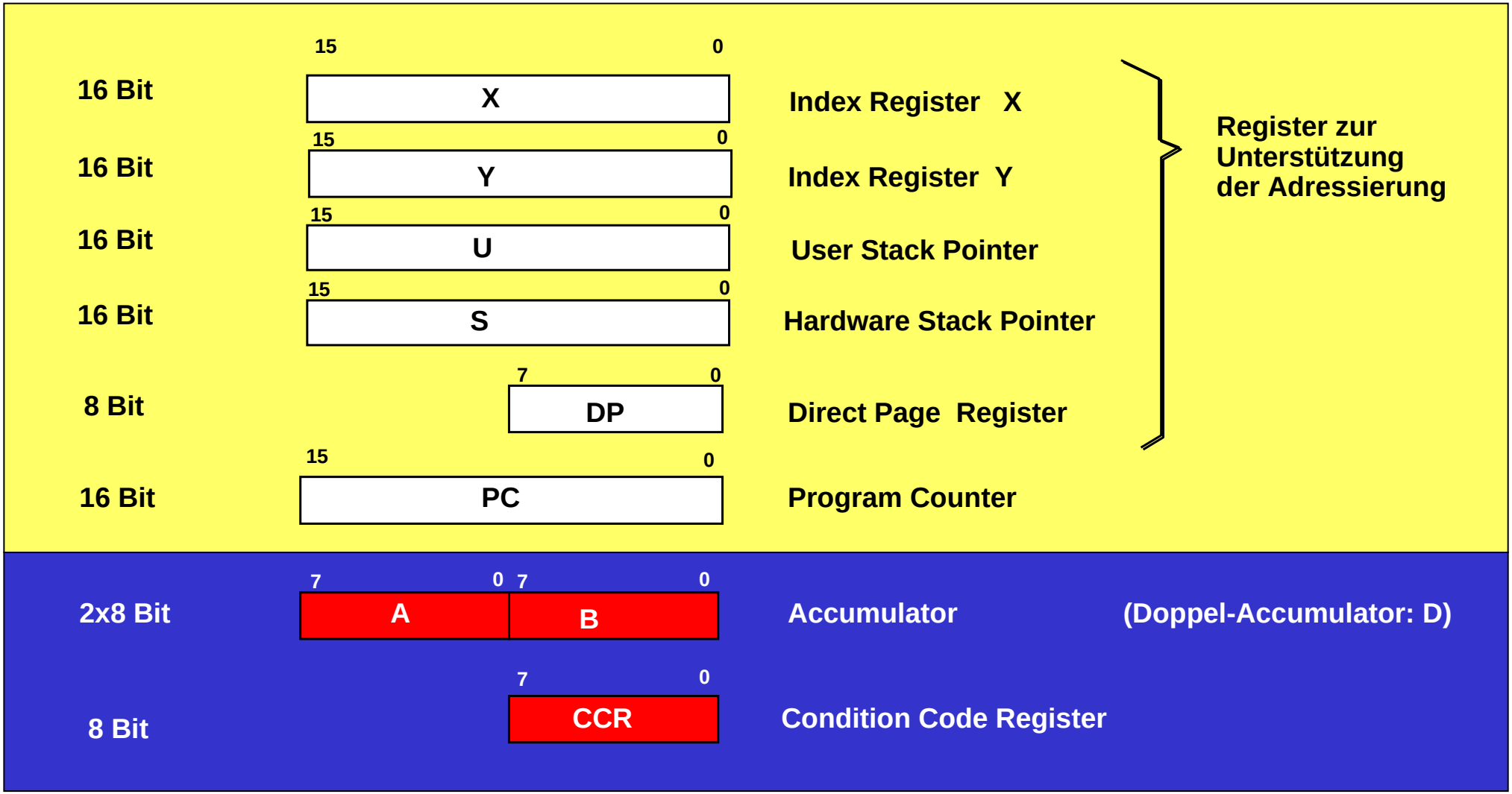
Programmiermodell für die Programmierung von Anwendungen:

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

Programmiermodell für die Programmierung von Systemprogrammen:

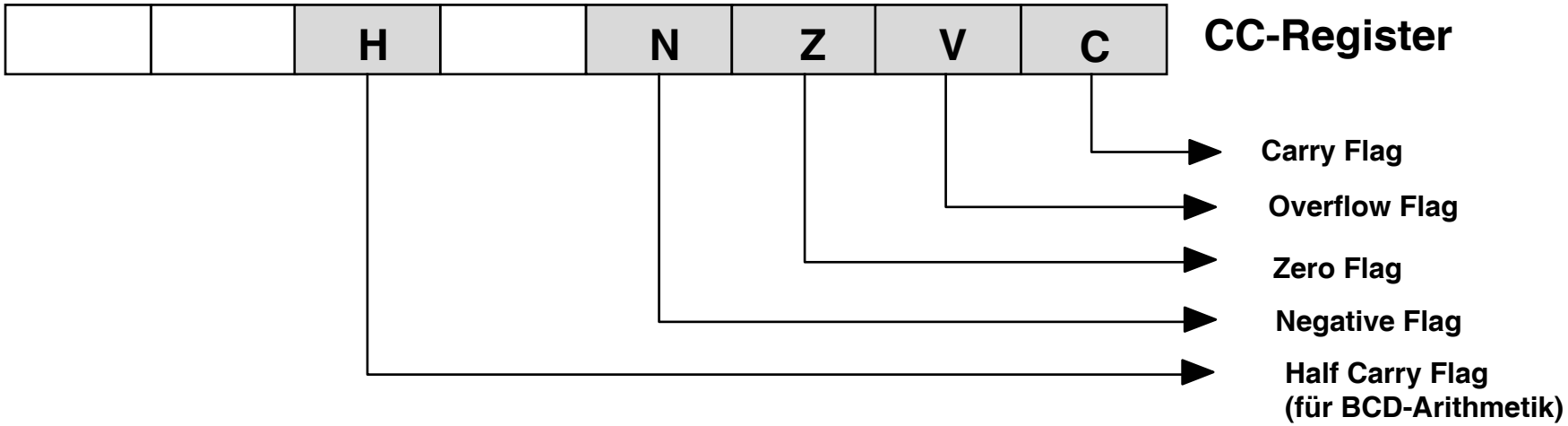
- Speicherverwaltung
- Zugriffsschutz
- Unterbrechungs- und Ausnahmebehandlung
- Multitasking
- Ein/Ausgabe
- Initialisierung des Prozessors
- Coprozessorschnittstellen und Protokolle
- Unterbrechnungs- und Ausnahmebehandlung

Programmiermodell des Registersatzes des Motorola 6809 Prozessors

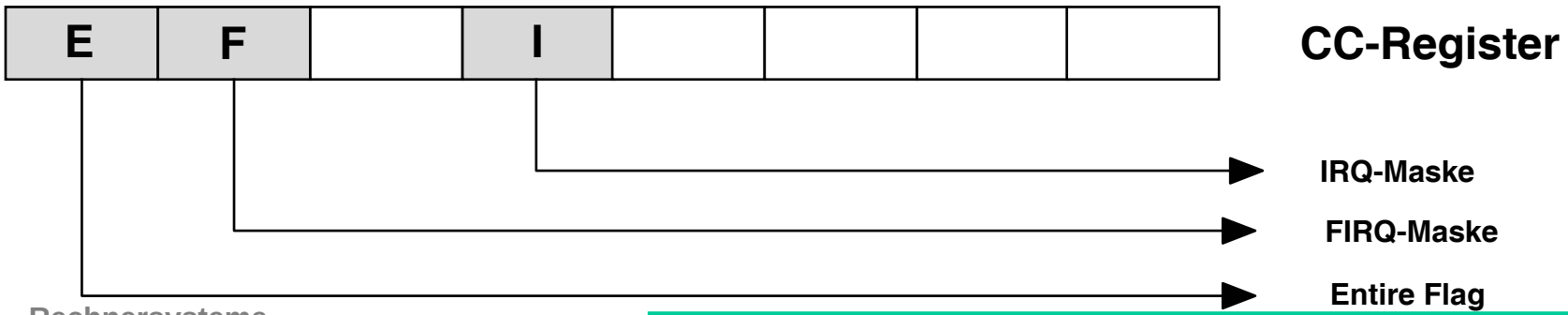


Bedingungs-Codes (Condition Codes) des 6809

Condition Codes für Arithmetische/Logische Operationen



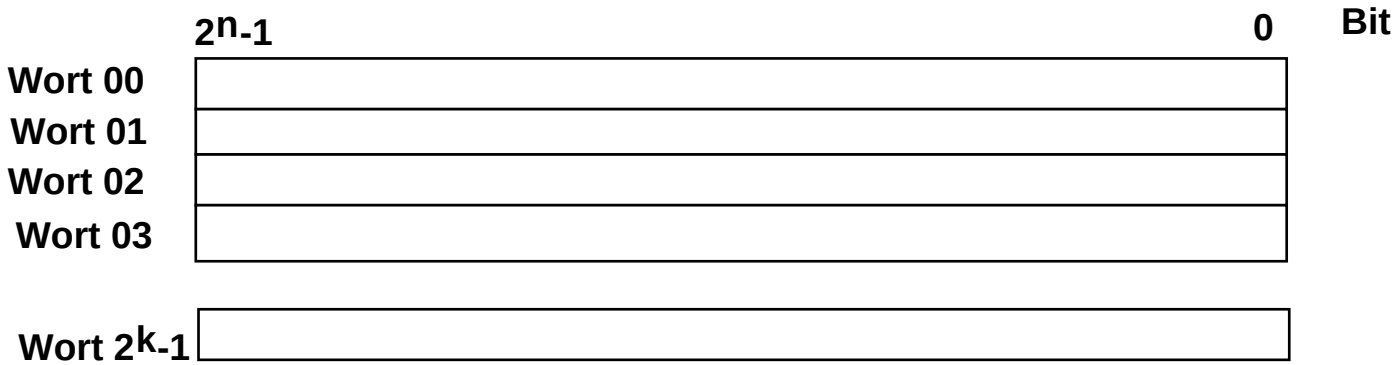
Condition Codes zur Unterbrechungsbehandlung (Interrupt)



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

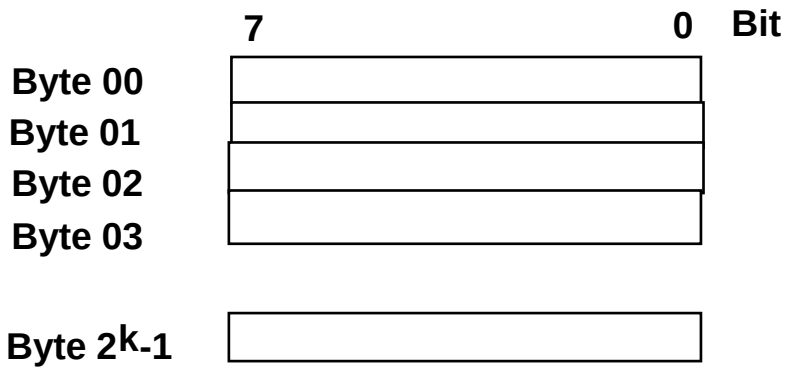
Speicherorganisation

Wort-orientierter Speicher



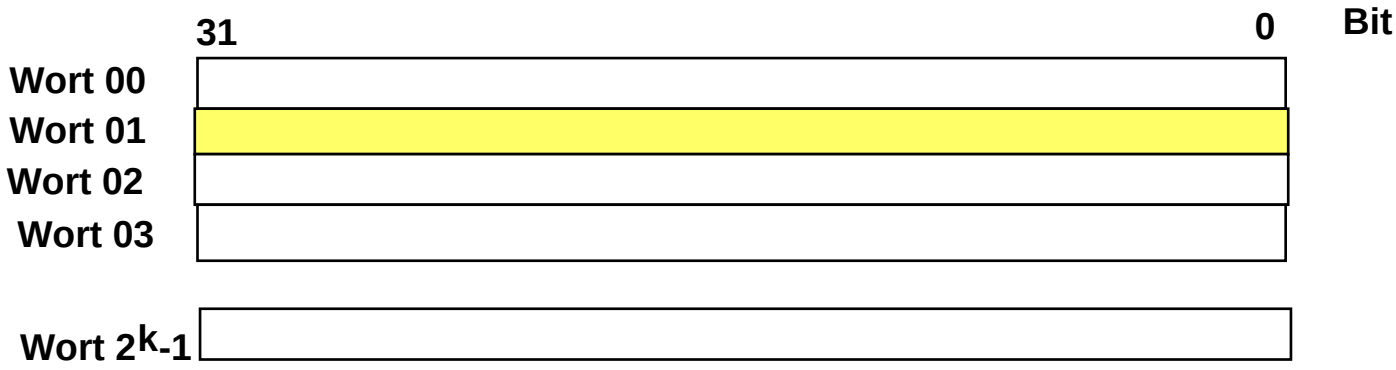
Byte-orientierter Speicher

Vorteil: Kompakteres Abspeichern verschieden langer Befehle und Operanden

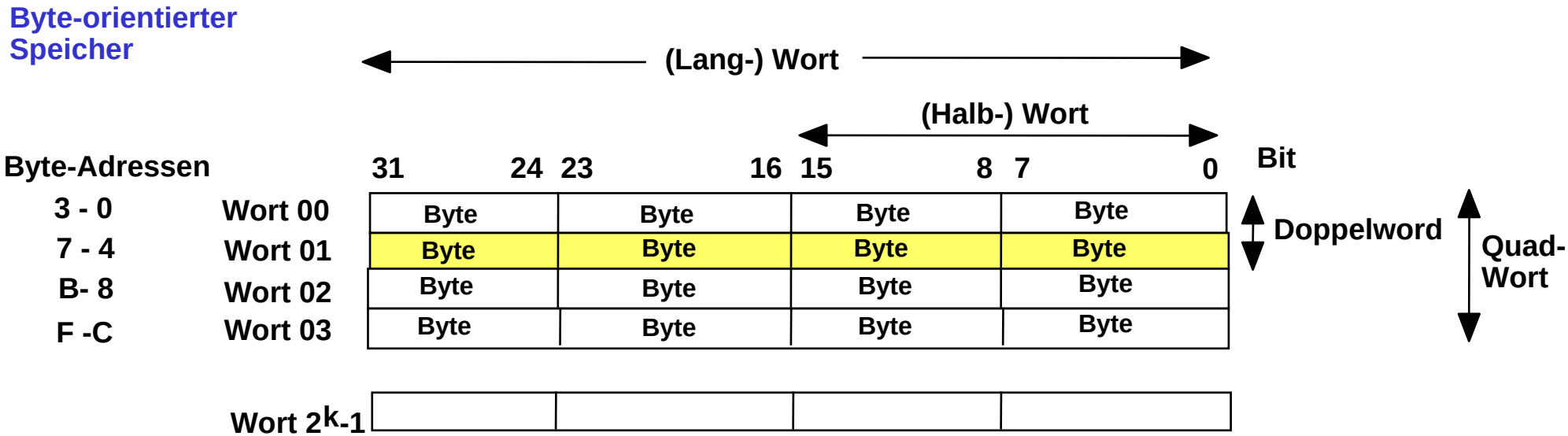


Speicherorganisation

Wort-orientierter Speicher

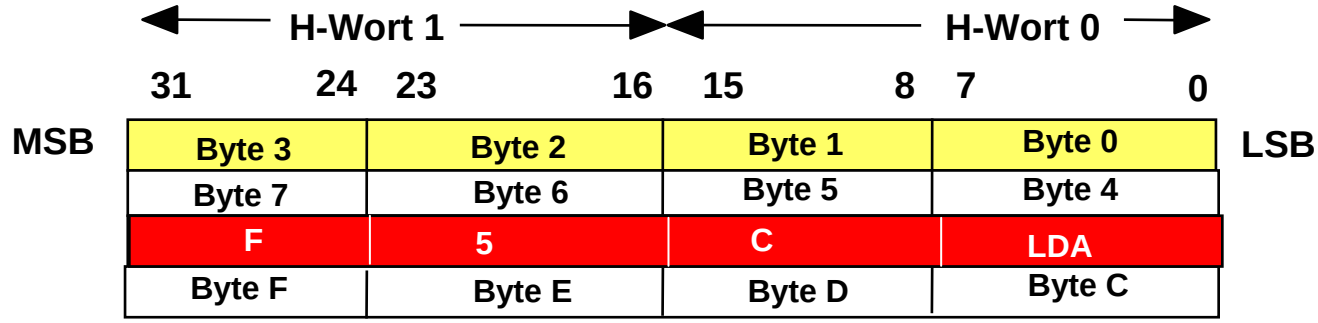


Byte-orientierter Speicher



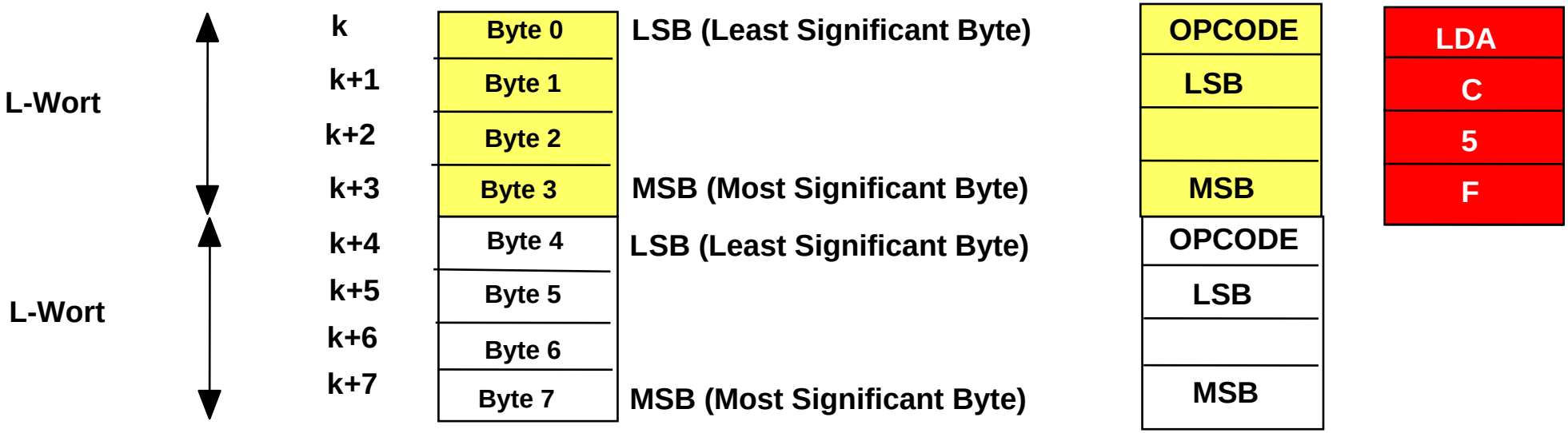
Little-Endian-Darstellung

Beisp. Intel 80x86



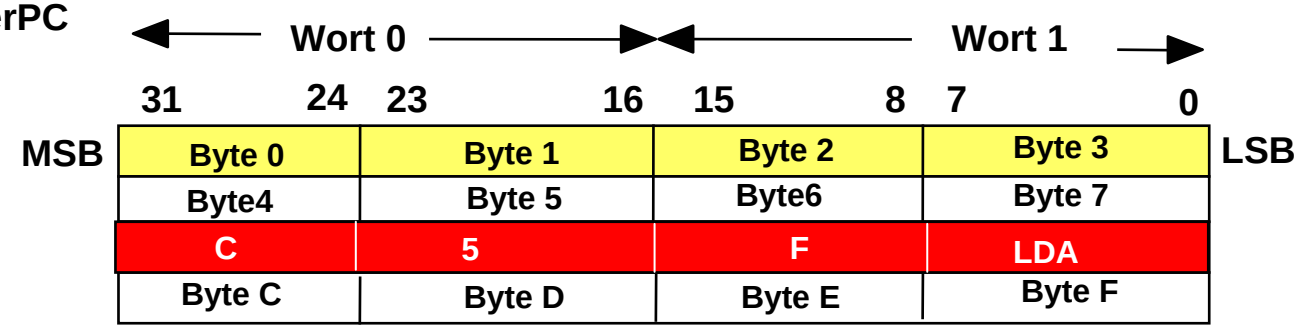
Beispiel:
LDA $F5C

Little-Endian-Darstellung



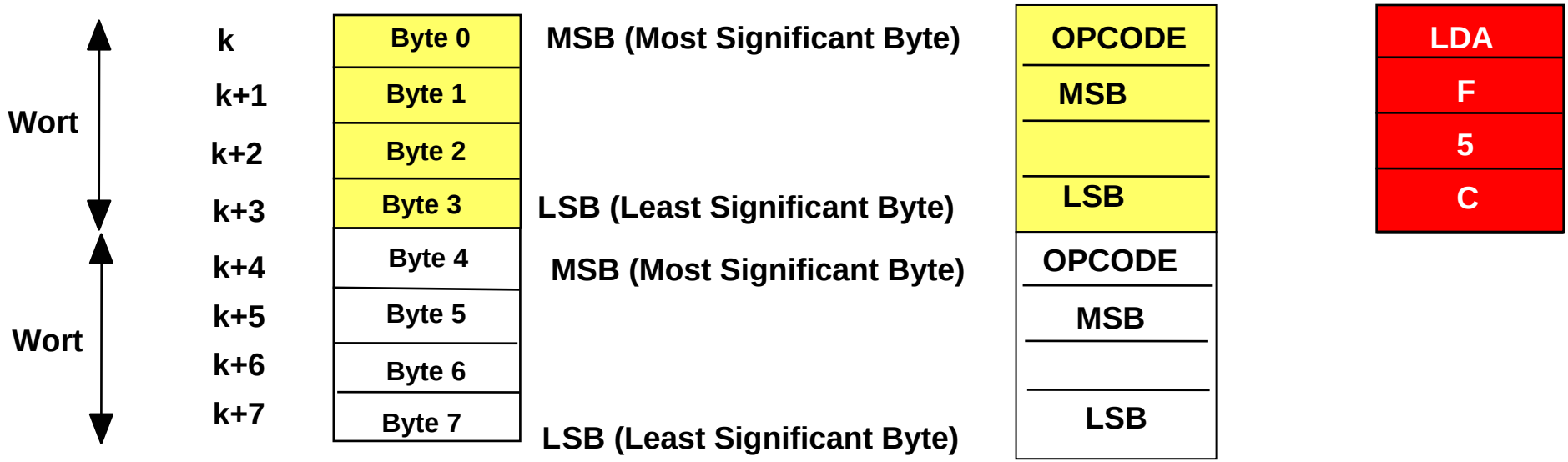
Big-Endian-Darstellung

Beisp. Motorola 680x, 68x00, SPARC, PowerPC

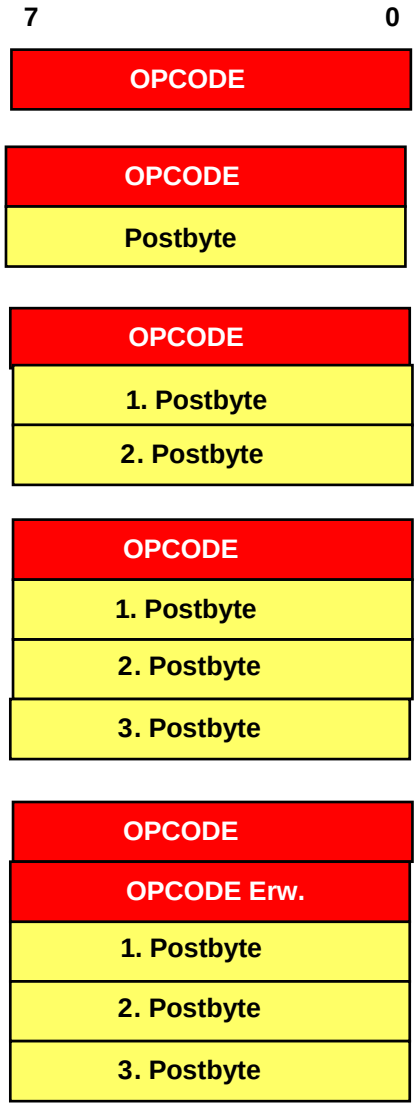


Beispiel:
LDA $F5C

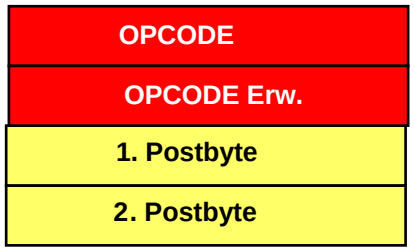
Big-Endian-Darstellung



6809 Befehlsformate:



oder



1-Byte -Befehle - der Operand, falls vorhanden wird implizit durch den OPCODE bestimmt z.B. INCA

2-Byte -Befehle - Bedeutung des Postbytes: 1. kurze Adresse
2. nähere Bestimmung der Adressierung mit Index

3-Byte -Befehle - Bedeutung der Postbytes: 1. lange Adresse
2. nähere Bestimmung der Adressierung mit Index

4-Byte -Befehle ohne OPCODE Erw.
Bedeutung der Postbytes: nähere Bestimmung der Adressierung mit Index

4-Byte -Befehle mit OPCODE Erw.
Bedeutung der Postbytes: lange Adresse

5-Byte -Befehle
Bedeutung der Postbytes: nähere Bestimmung der Adressierung mit Index

Der 6809 Befehlssatz

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

Datenmanipulation

Adreß manipulation

Kontrollflußmanipulation

Systembezogene Befehle

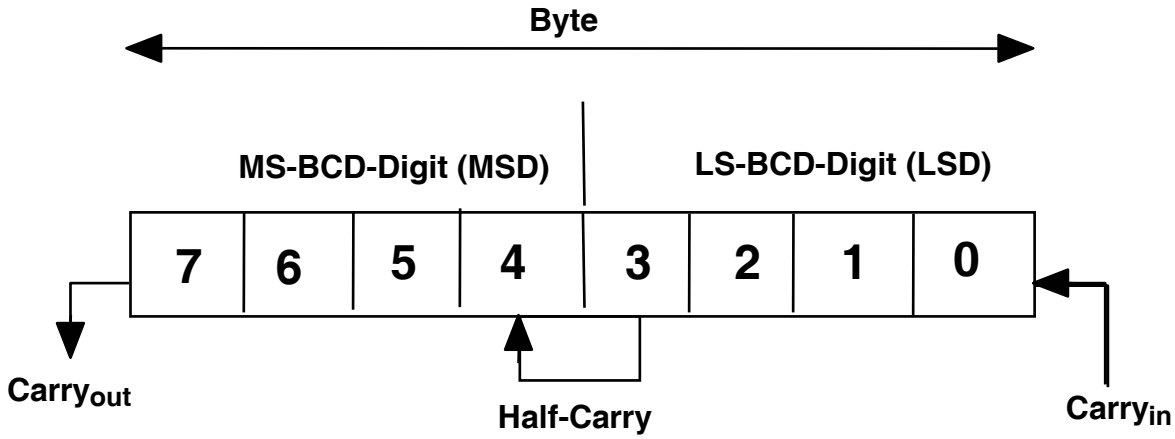
6809 Befehlssatz (1):

Arithmetische Befehle:

Addition:	**8**	**ADCA, ADCB**	**Add memory to Accumulator (A,B) with Carry**
		ADDA, ADDB	**Add memory to Accumulator (A,B)**
	16	**ADDD**	**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 ← A x B**
Vorz. Erw.:	**16**	**SEX**	**Sign Extend B Acc. into A Acc.**
Subtraktion:	**8**	**SBCA, SBCB**	**Subtract memory from Acc. (A,B) with Borrow**
		SBA	**Subtract Acc.B from Acc A**
	8/16	**SUBA, SUBB, SUBD**	**Subtract memory from Acc. A,B,D (without Borrow)**

Die Instruktion

DAA



Fall: $\mathbf{sum_{LSD}} \leq 9$ keine Korrektur erforderlich

2. Fall: $9 \leq \mathbf{sum_{LSD}} \leq 15$

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

3.Fall $16 \leq \mathbf{sum_{LSD}} \leq 18$

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

Korrektur durch Addition von +6

Bedingung für die Korrektur des LSD : (sum_{LSD} > 9) ODER (H=1)
Bedingung für die Korrektur des MSD : C_{out} =1 ODER (sum_{MSD} > 8) UND (sum_{LSD} > 9)

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

```
  0010 1001          29
+ 0011 0110        + 36
-------------------------------
  0101 1111    ≠     65
```

Bedingung für die Korrektur des LSD : (sum_{LSD} > 9) ODER (H=1)
Bedingung für die Korrektur des MSD : C_{out} =1 ODER (sum_{MSD} > 8) UND (sum_{LSD} > 9)

```
   0101 1111
+  0000 0110   +6
-----------------
  0110 0101     =    65
```

Half Carry

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)

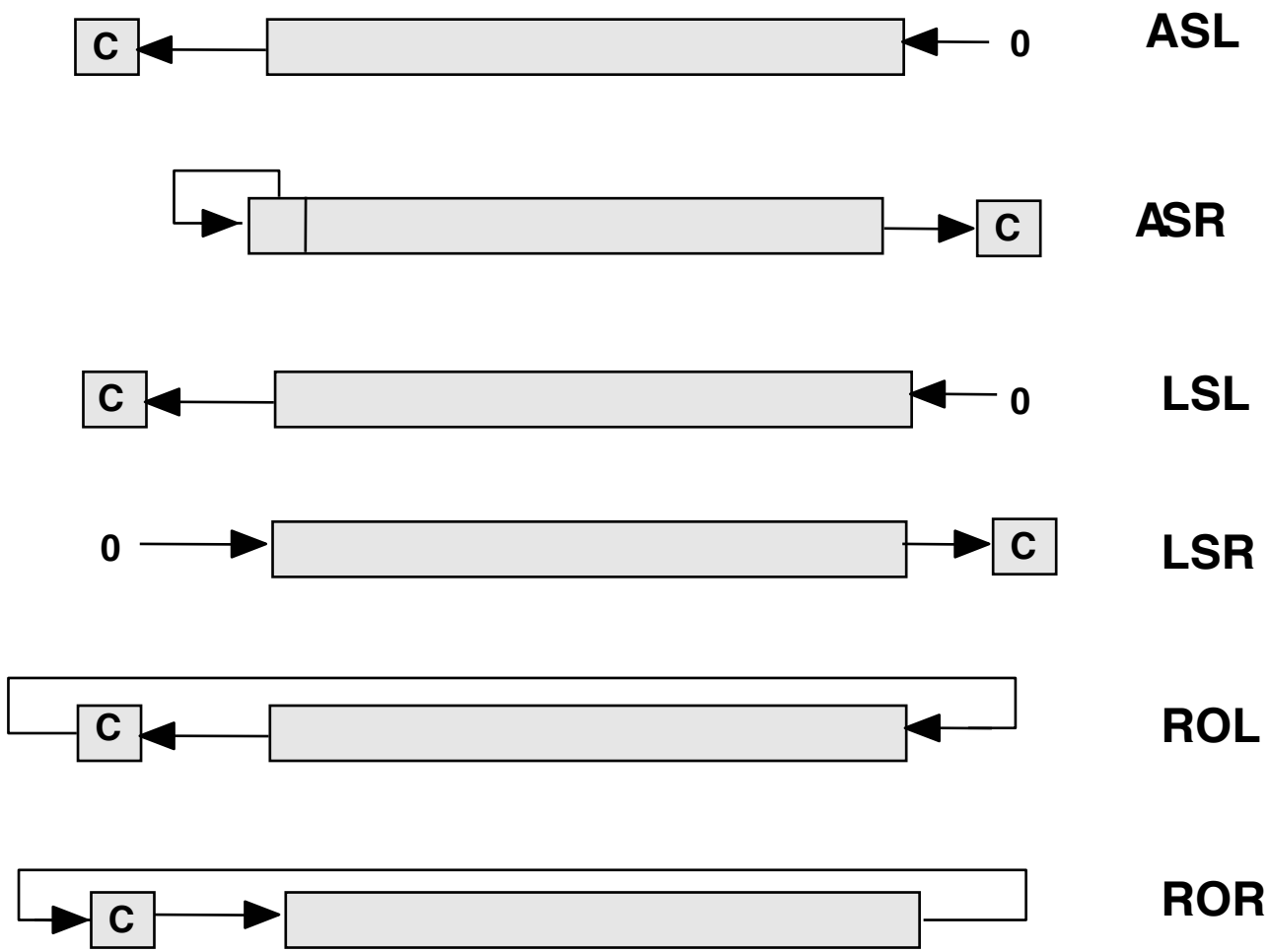
Shift Befehle:

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

Shift Befehle des 6809



6809 Befehlssatz (3):

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

Beispiele: Vergleichsbefehle CMPA mem

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

A > mem !

```
          0010 0101
          1110 1101        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)
mem = 67_{10} (0100 0011)

A < mem !

```
          0010 0111
          1011 1101        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

Beispiele: Vergleichsbefehle

A = $A3_{16}$ (1010 0011)
mem = 64_{16} (0110 0100)

A < mem !

```
      1010 0011
      1001 1100      2er-Komplement von 64
  1   1110 0100      Das Carry-Flag wird komplementiert
```

Folgende Belegung des CC-Registers wird erzeugt:

N	Z	V	C
0	0	1	0

6809 Befehlssatz (4):

Index Register Befehle:

Ladebefehle	**LDX, LDY**	**Load Index Register from memory (X,Y)**
	STX, STY	**Store Index Register to memory (X,Y)**
	LEAX, LEAY	**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**	**Load Stackpointer from memory (S,U)**
	STS, STU	**Store Stackpointer to memory (S,U)**
	LEAS, LEAU	**Load Effective Address into Stackpointer (S,U)**
Vergleichsbefehl	**CMPS, CMPU**	**Compare memory from Stackpointer (S,U)**
Stackoperationen	**PSHS**	**Push any register(s) onto system stack (except S)**
	PSHU	**Push any register(s) onto user stack (except U)**
	PULS	**Pull any register(s) from system stack (except S)**
	PULU	**Pull any register(s) from user stack (except U)**

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

6809 Befehlssatz (5): Branch Befehle

Unbedingte Sprünge:

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

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):

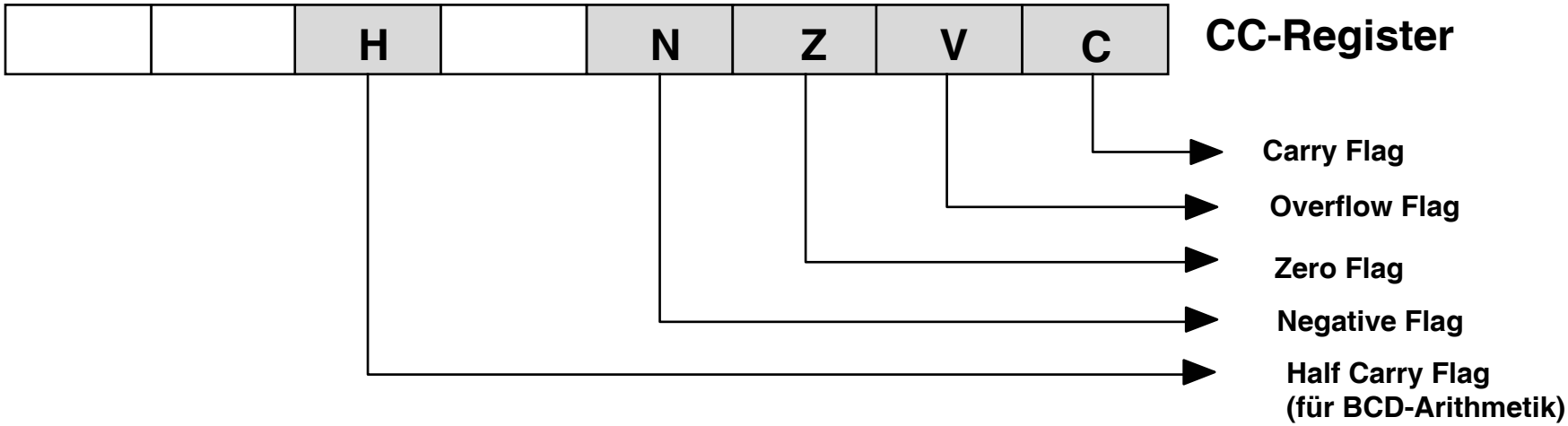
BGE, LBGE	Branch if greater than or equal
BLE, LBLE	Branch if less than or equal
BGT, LBGT	Branch if greater
BLT, LBLT	Branch if less than

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

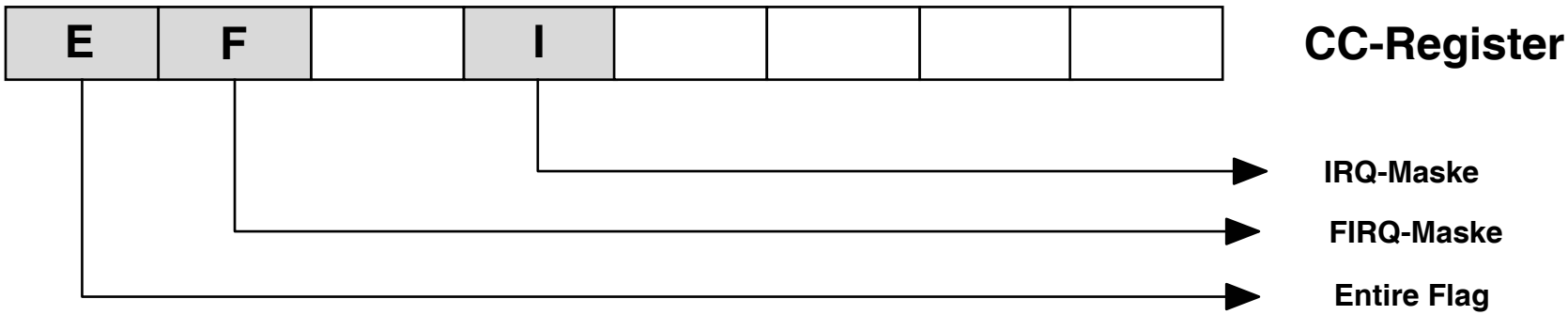
BHS, LBHS	Branch if higher or same
BLS, LBLS	Branch if lower or same
BHI, LBHI	Branch if higher
BLO, LBLO	Branch if lower

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 ⊕ V) = 1
a < b	x		BLT	N ⊕ V = 1
$a \geq b$	x		BGE	N ⊕ V = 0
a > b	x		BGT	Z + (N ⊕ 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)
mem = 19_{10} (0001 0011)

A > mem !

```
      0010 0101
      1110 1101      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)
mem = 67_{10} (0100 0011)

A < mem !

```
      0010 0111
      1011 1101      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 = $A3_{16}$ (1010 0011)
mem = 64_{16} (0110 0100)

A < mem !

```
        1010 0011
        1001 1100      2er-Komplement von 64
   1    1110 0100      Das Carry-Flag wird komplementiert
```

Folgende Belegung des CC-Registers wird erzeugt:

N	Z	V	C
0	0	1	0

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 Unterprogrammsprung- 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