

## Speicherorganisation und Adressierungsarten

Otto-von-Guericke-Universität Magdeburg



# Speicherorganisation

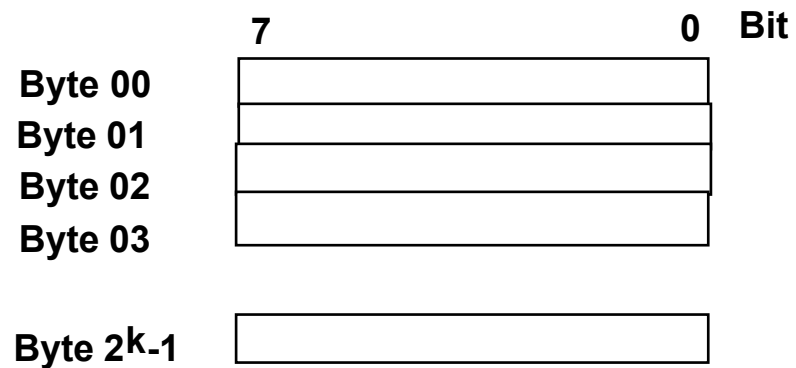
---

## Wort-orientierter Speicher



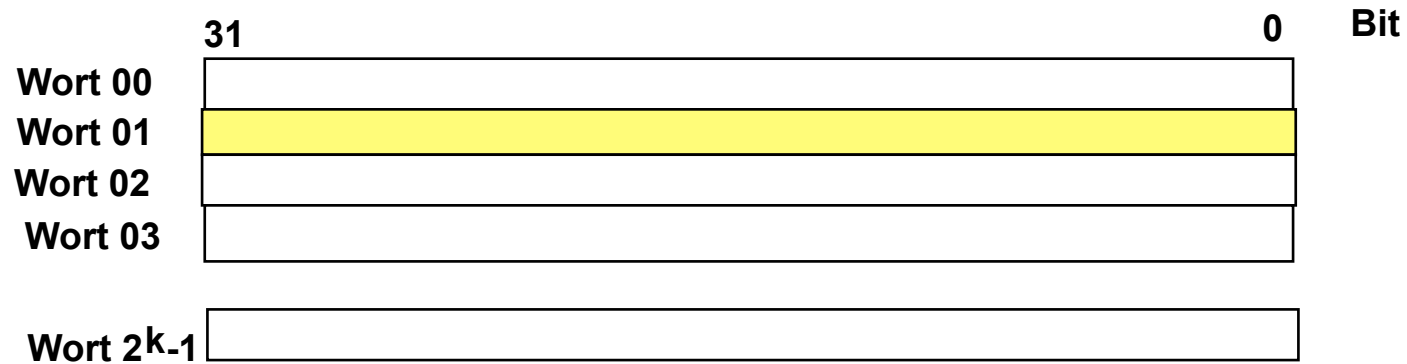
## Byte-orientierter Speicher

**Vorteil: Kompakteres Abspeichern verschieden langer Befehle und Operanden**

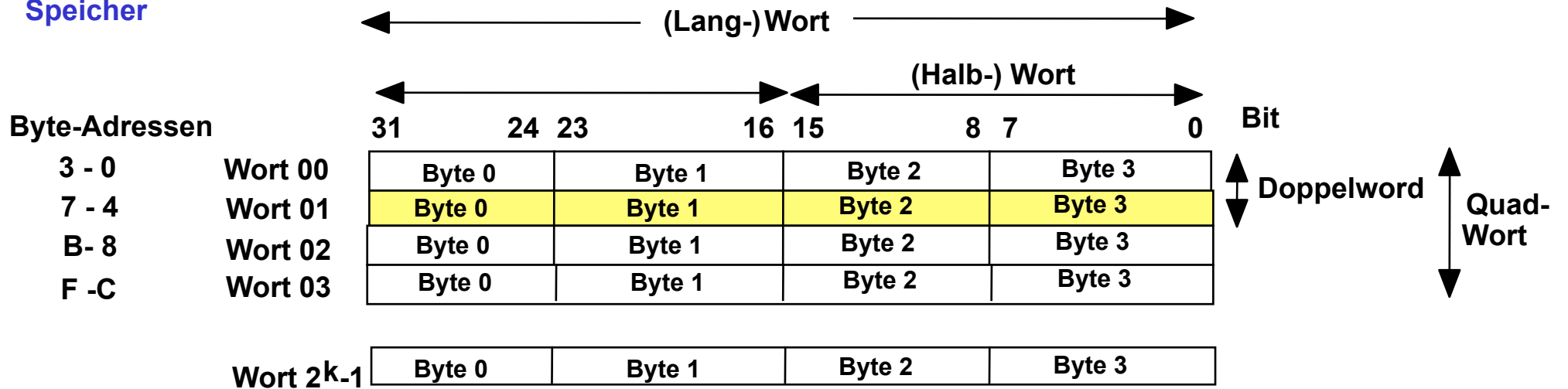


# Speicherorganisation

## Wort-orientierter Speicher

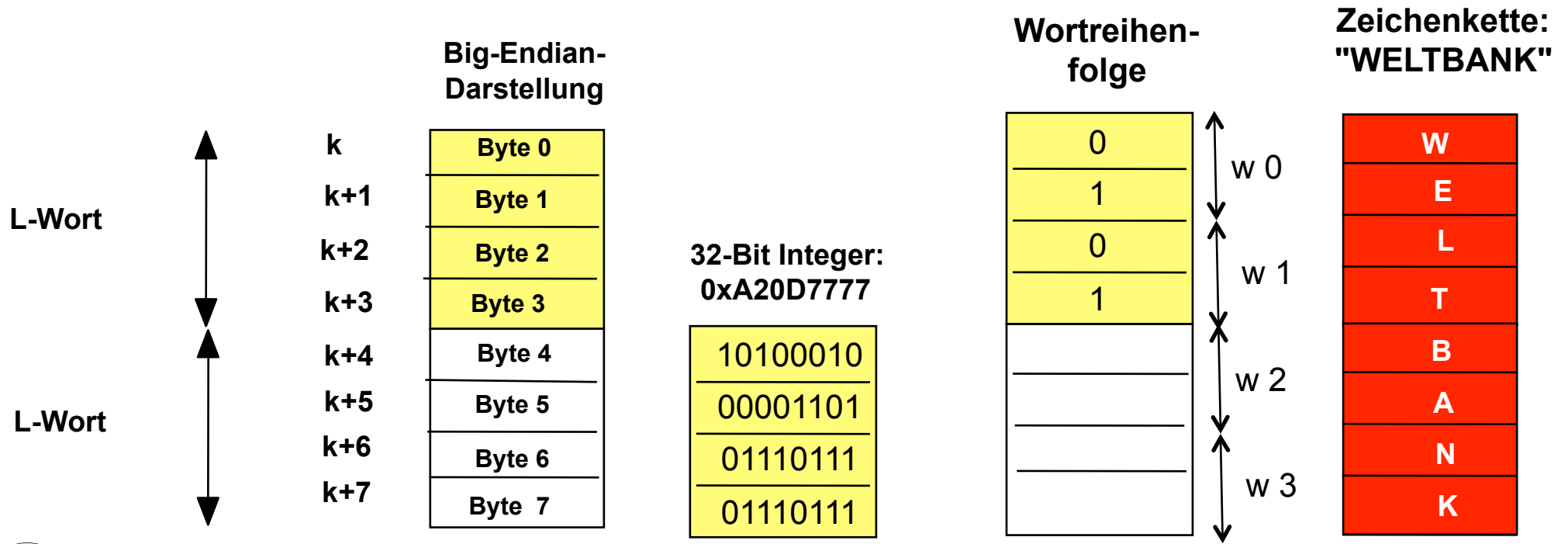
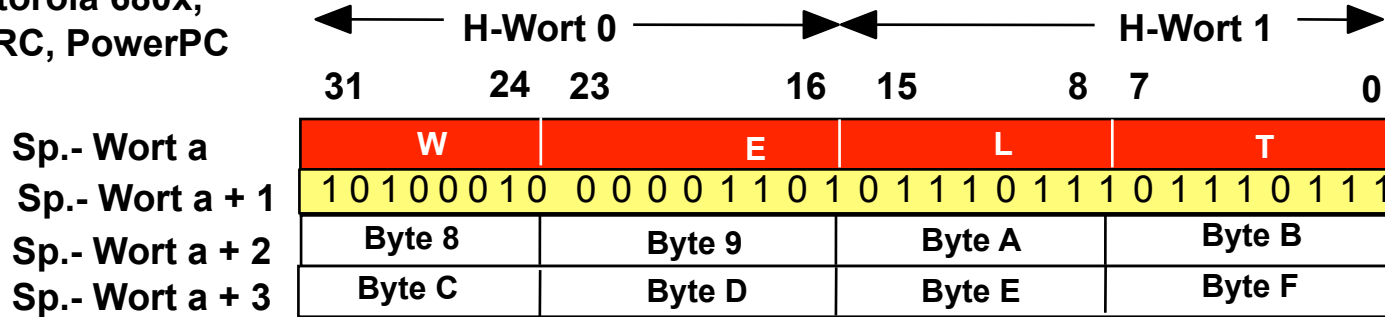


## Byte-orientierter Speicher



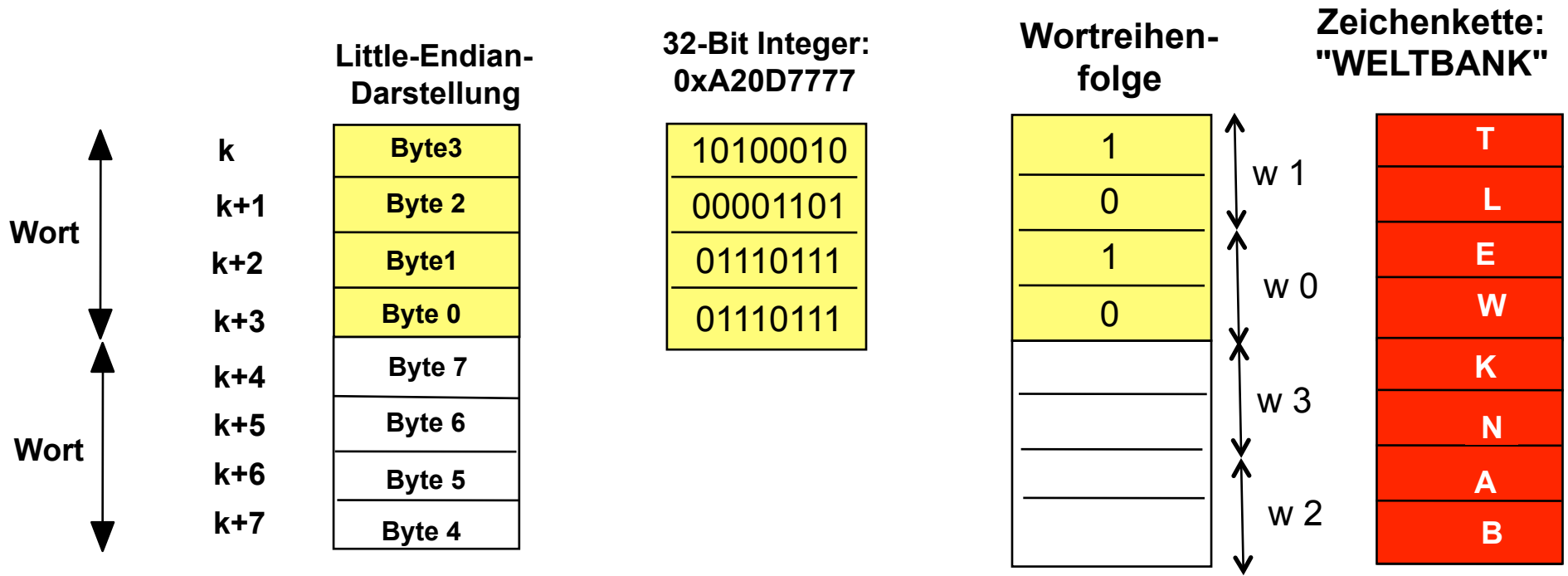
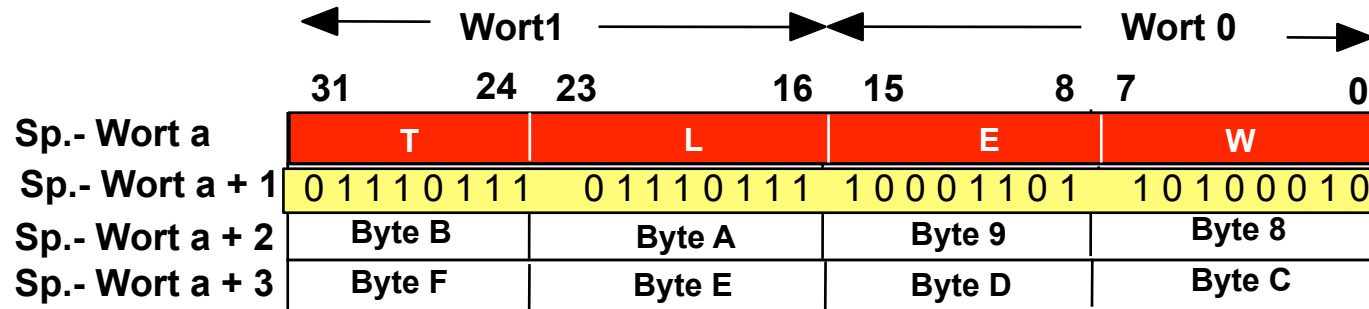
# Big-Endian-Darstellung

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



# Little-Endian-Darstellung

Beisp. Intel 80x86, ARM



# Little-/Big-Endian-Darstellung

Zeichenkette: HELLO\_World

## Big - Endian

BYTE-NR	0	1	2	3	0	1	2	3	0	1	2	3
	H	E	L	L	O	_	W	O	R	L	D	!

## Little - Endian

BYTE-NR	3	2	1	0	3	2	1	0	3	2	1	0
	L	L	E	H	O	W	_	O	!	D	L	R

Für 16-Bit Datentypendes MC 6809 wird die Big-Endian Darstellung verwendet

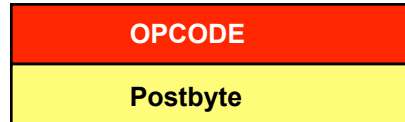


# 6809 Befehlsformate:

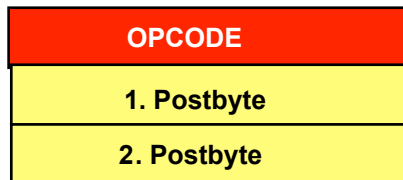
7 0



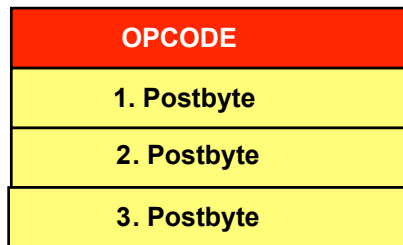
**1-Byte -Befehle** - der Operand, falls vorhanden wird implizit durch den OPCODE bestimmt z.B. INCA



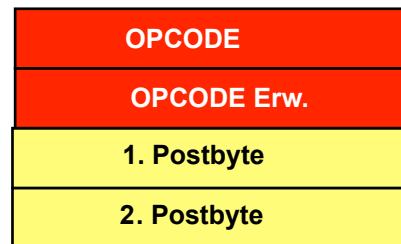
**2-Byte -Befehle** - Bedeutung des Postbytes: 1. kurze Adresse  
2. nähere Bestimmung der Adressierung mit Index



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

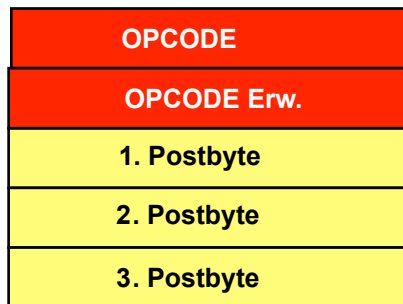


oder



**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



# Ziele beim Entwurf eines Adressierungsmechanismus

- **großer Adreßraum**
- **kleines Adreßfeld in der Instruktion**
- **Komfort für den Programmierer**
- **positionsunabhängiger (verschieblicher) Code**

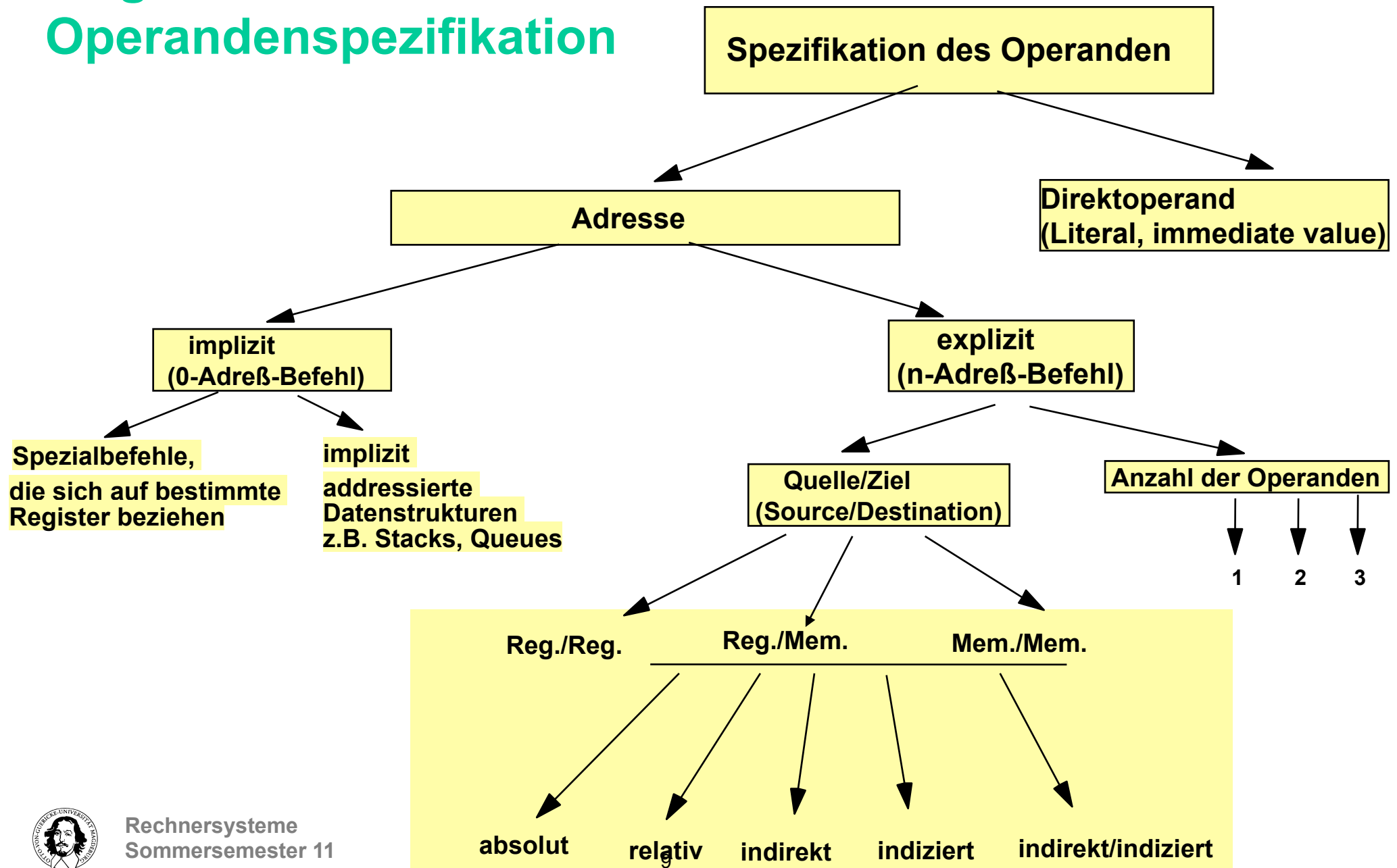
**Lösung des Zielkonflikts zwischen Größe des AR und Größe des AF:  
Berücksichtigung von Lokalität der Berechnungen**

**Vereinfachung der Adressierung einfacher Datenstrukturen (Listen, Arrays)**





# Möglichkeiten der Operandenspezifikation



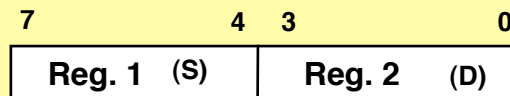
# 6809 Register Adressierung

Befehle: TFR, EXG,

Speicherplatz

m	OPCODE
m+1	Reg. Spec.
m+2	
m+3	
m+4	

Postbyte Definition bei TFR und EXG:

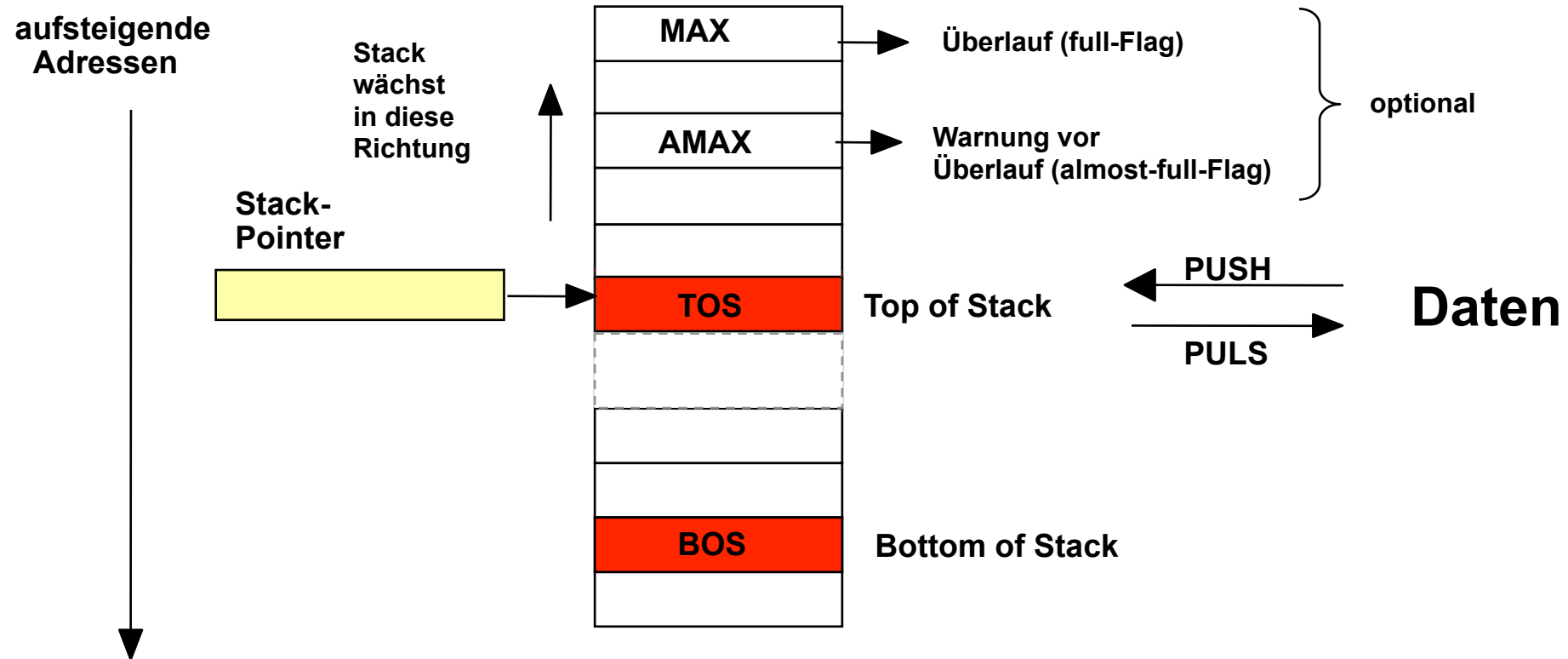


S: Source  
D: Destination

Register	Code (Bin)	Hex
D	0000	0
X	0001	1
Y	0010	2
U	0011	3
S	0100	4
PC	0101	5
A	1000	8
B	1001	9
CCR	1010	A
DP	1011	B



# Stack (Stapel, Keller, LIFO-Queue)



## PUSH:

$SP = MAX$ : Exception 'full'

$SP \leftarrow SP - 1$

$Mem[SP] \leftarrow Register$

$SP > AMAX$ : Signal 'almost full'

## PULS:

$SP = BOS$ : Exception 'empty'

$Register \leftarrow Mem[SP]$

$SP \leftarrow SP + 1$



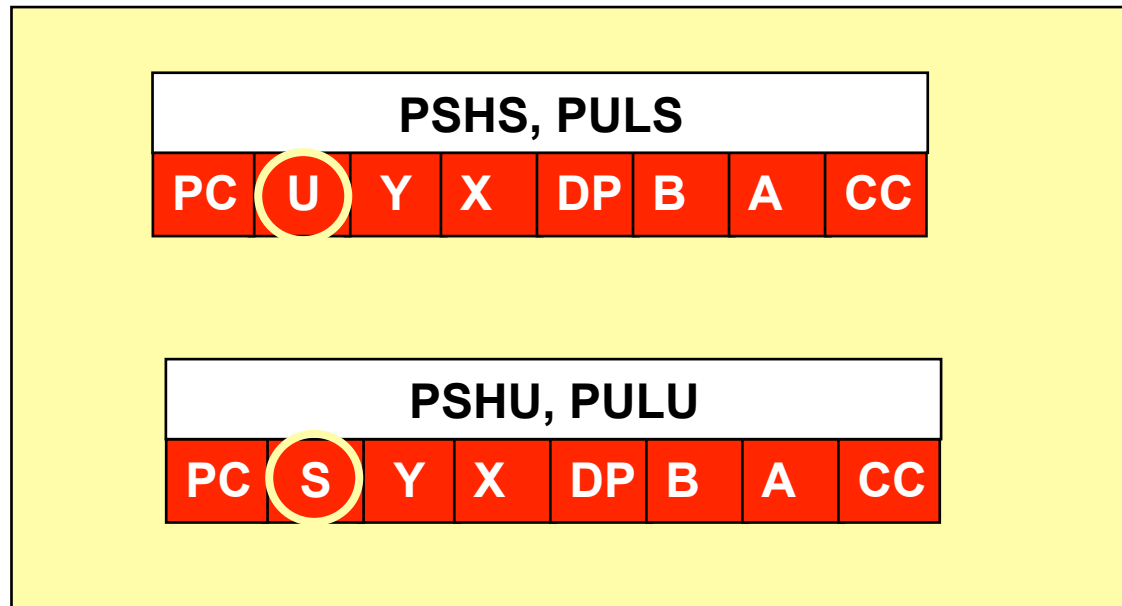
# 6809 Stack Adressierung

**Befehle: PSHU, PSHS, PULU, PULS**

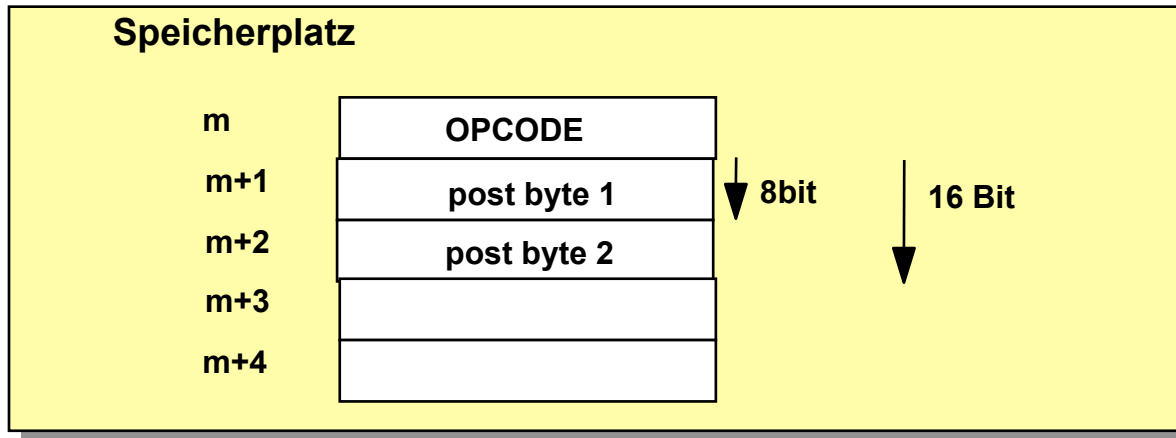
Speicherplatz

m	OPCODE
m+1	Register List.
m+2	
m+3	
m+4	

Post-Byte Definition bei  
PSHx und PULx



# 6809 Direktoperanden (unmittelbare Adressierung, engl.: immediate)



Beisp.: ADDA #\$95

m	8B
m+1	95

PC = PC+2

ADDD #\$1295

m	C3
m+1	12
m+2	95

PC = PC+3

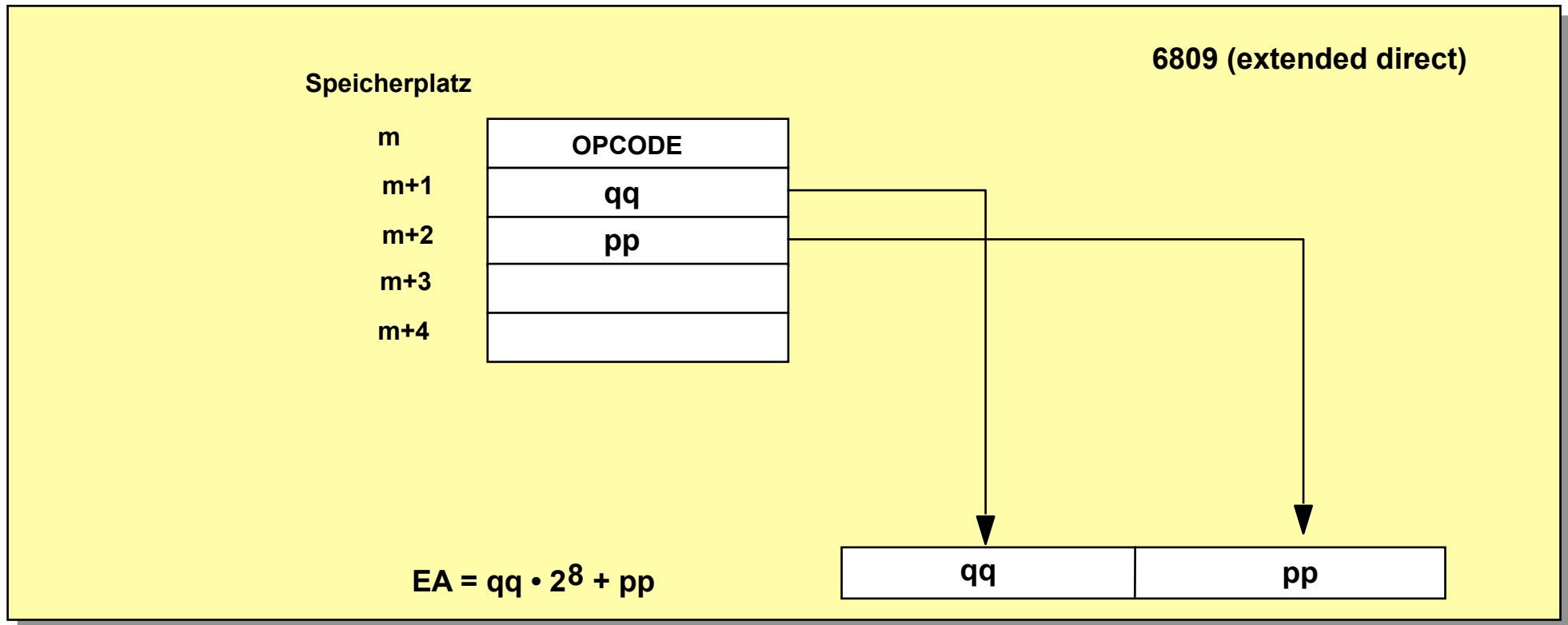
LDS #\$ABCD

m	10
m+1	CE
m+2	AB
m+3	CD

PC = PC+4



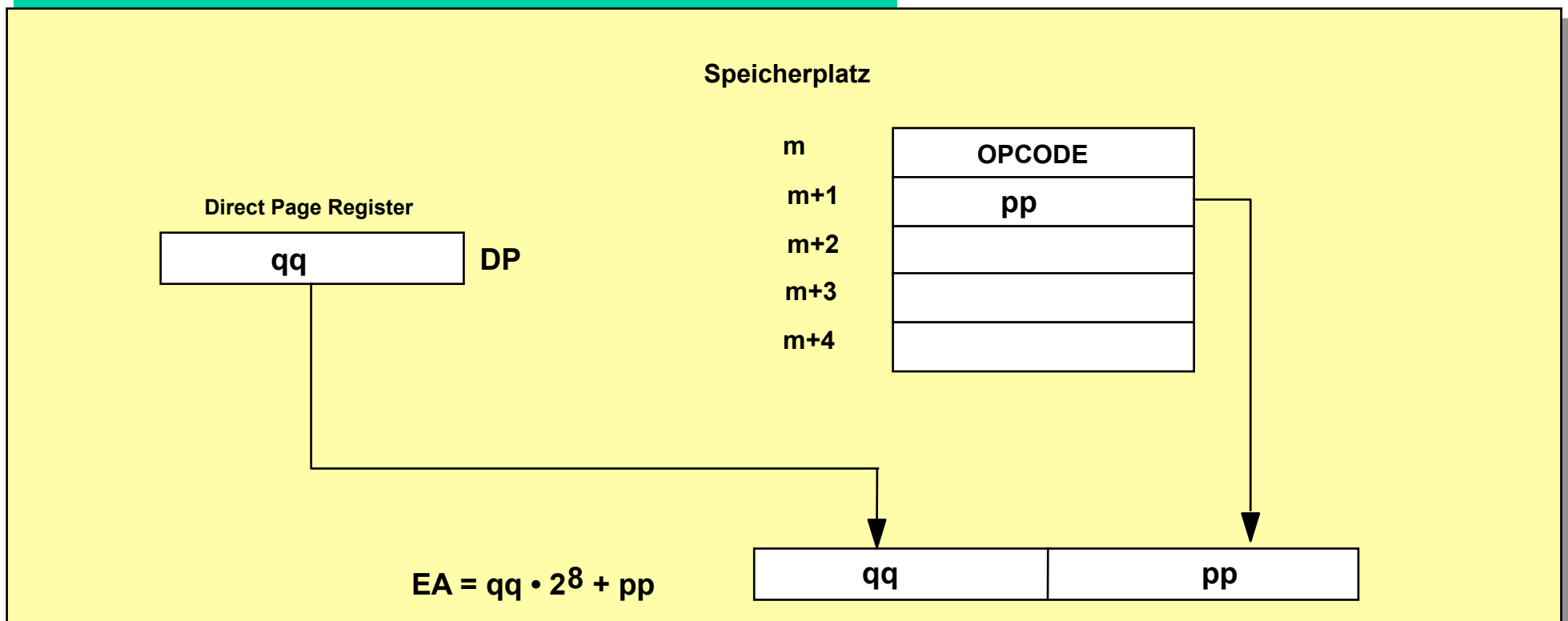
# Absolute (Direkte) Adressierung



Die **effektive Adresse EA** wird durch Konkatination der beiden Post Bytes nach dem OPCODE gebildet. Das erste Post Byte (qq) gibt das höherwertige Byte, das zweite Postbyte (pp) das niederwertige Byte der EA an.



# 6809 (base page direct) Adressierung über Basisregister

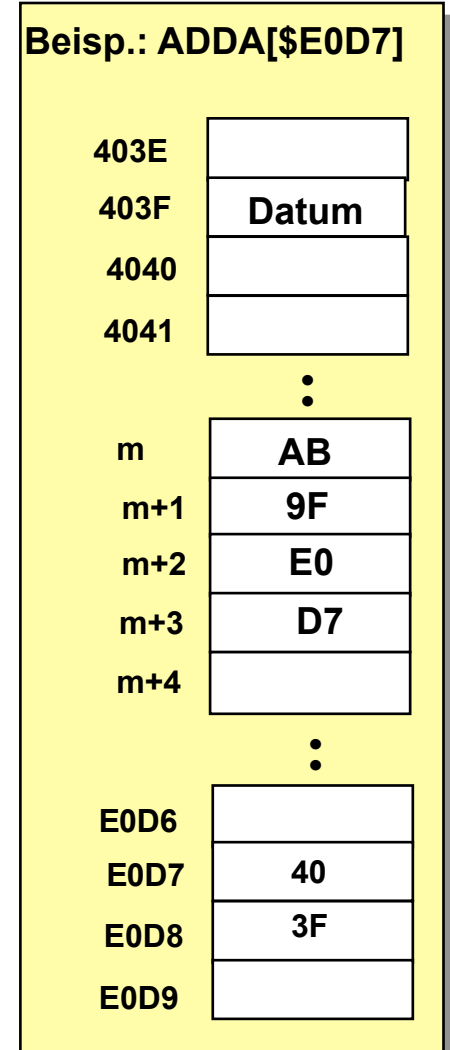
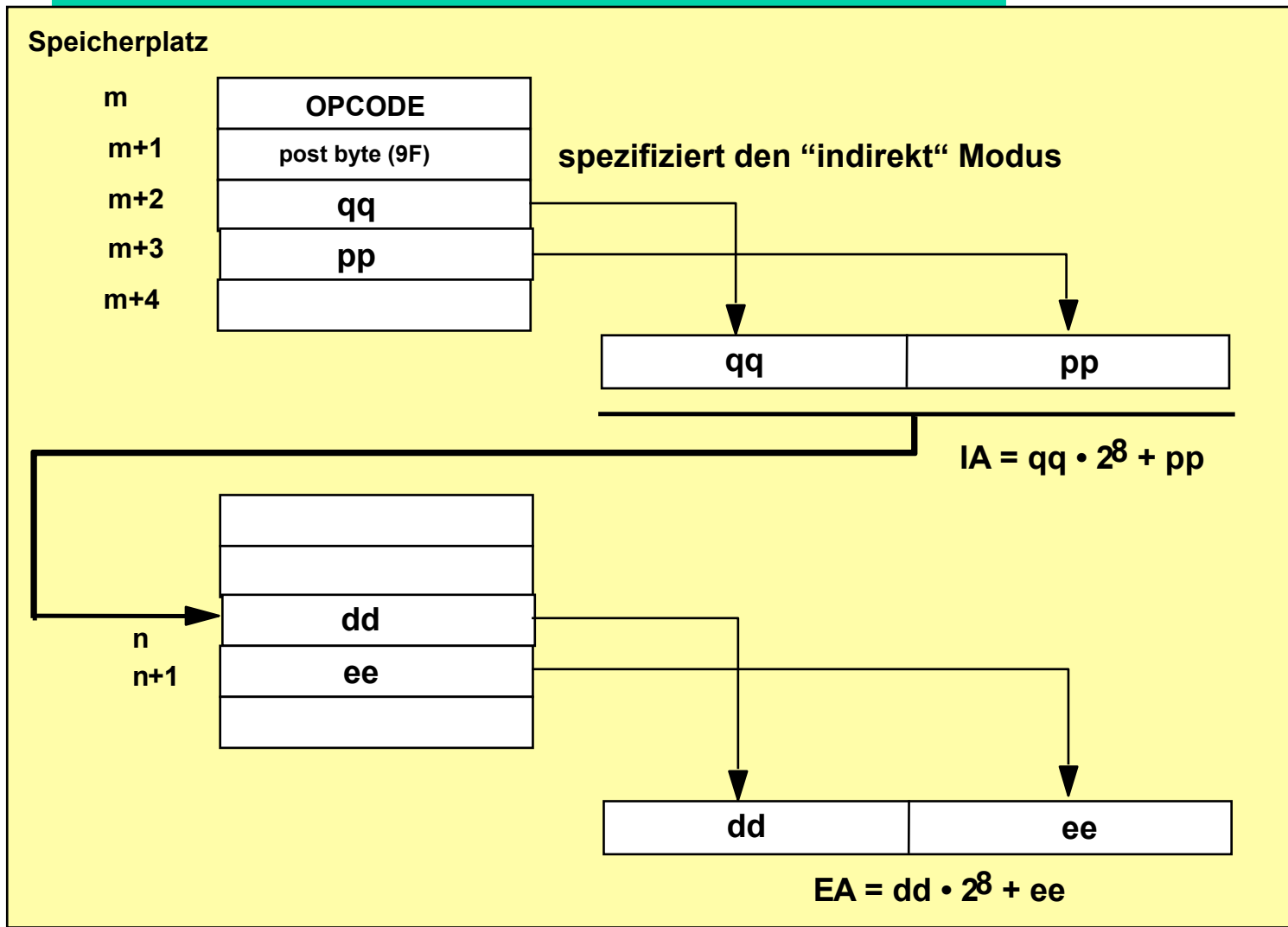


Die effektive Adresse EA wird durch Konkatination einer kurzen Adresse nach dem OPCODE mit dem Inhalt des Direct Page Registers (DP) gebildet. Die kurze Adresse gibt das niederwertige Byte, der Inhalt des DP das höherwertige Byte der EA an.

Das DP kann dynamisch durch einen Transferbefehle z.B. TFR A, DP geladen werden oder durch eine Assemblerdirective SETDP gesetzt werden.



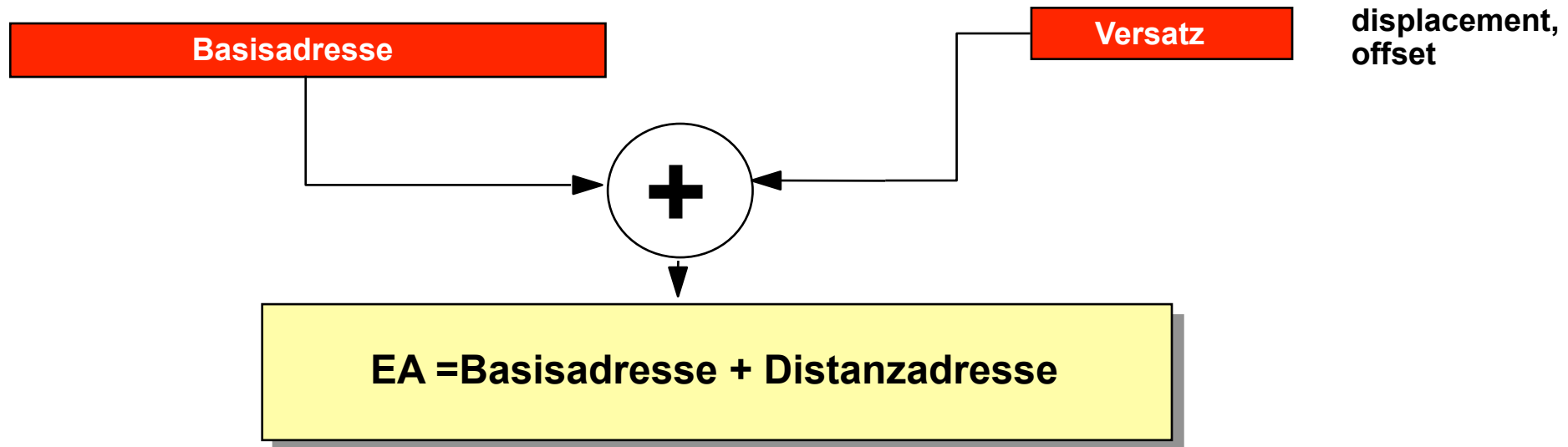
# 6809 (extended indirect) Indirekte Adressierung



Die dem OPCODE folgende Adresse gibt nicht an, wo sich die Daten für den Befehl befinden, sondern die Adresse, die auf die Daten zeigt (Unterstützung von Zeigern).



# Indizierte Adressierung (Indexed Addressing)



## Wo steht die Basisadresse ?

- Register: Indexregister
- Speicher: indirekte Adressierung

## Wo steht der Versatz ?

- Speicher (Literal)
- Register
- Autoincrement /Autodecrement



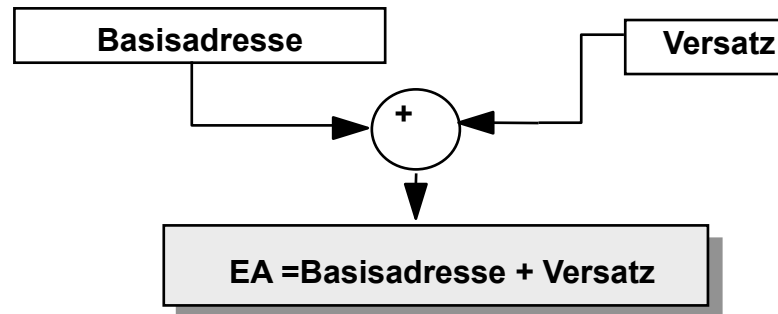
# Indizierte Adressierung beim MC 6809:

---

- **Zero offset**
- **5-Bit offset**
- **Long (16 Bit) constant offset**
- **Accumulator offset** (Versatz kann verändert werden)
- **Autoincrement**
- **Program Counter Relativ**
- **Relative Branches**
- **Indexed Indirect**



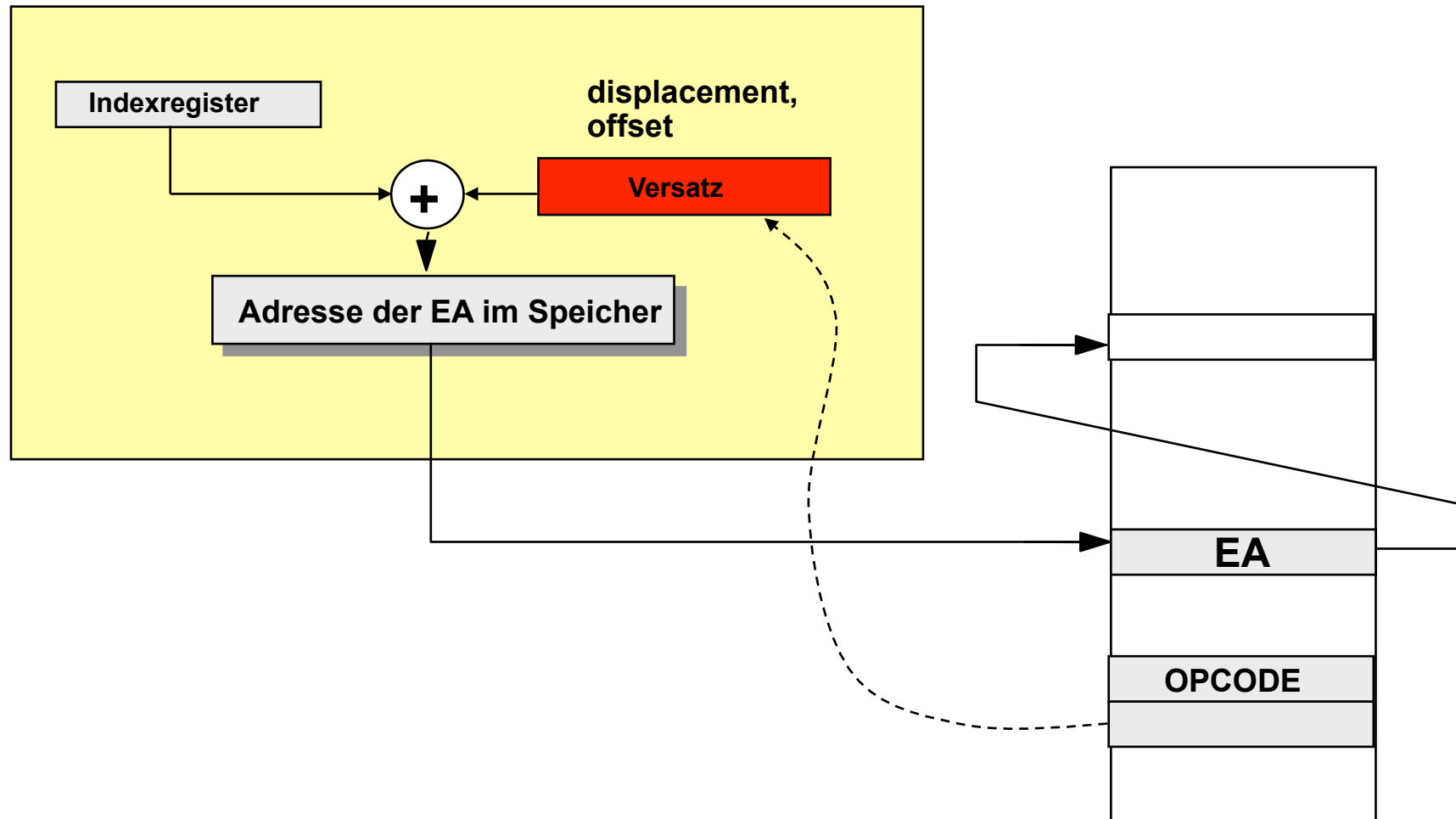
# Grundformen der indizierten Adressierung

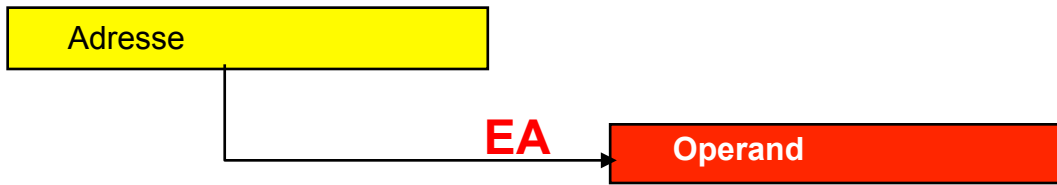


Basisadresse	Versatz	Bezeichnung
(Index-) Register X, Y, U, S, PC	Literal (kurz 5 Bit) Speicher (lang 8/16 Bit)	Adressierung mit konstanter Distanzadresse vom Indexregister "Constant Offset Mode"
(Index-) Register X, Y, U, S	Register A, B, D	Adressierung mit Distanzadresse aus dem Accumulator A, B oder D "Accumulator Offset"
(Index-) Register X, Y, U, S	fest $\pm 1, \pm 2$	Autoincrement / Autodecrement <u>POST</u> increment / <u>PRE</u> decrement

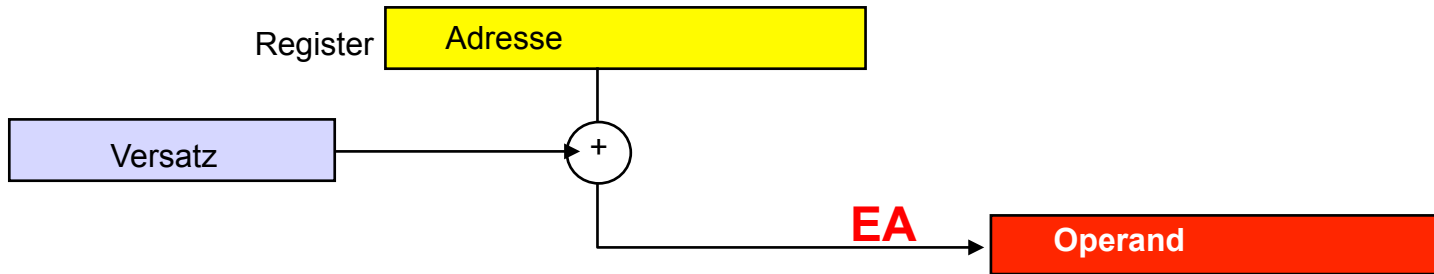


# Grundform der indiziert indirekten Adressierung

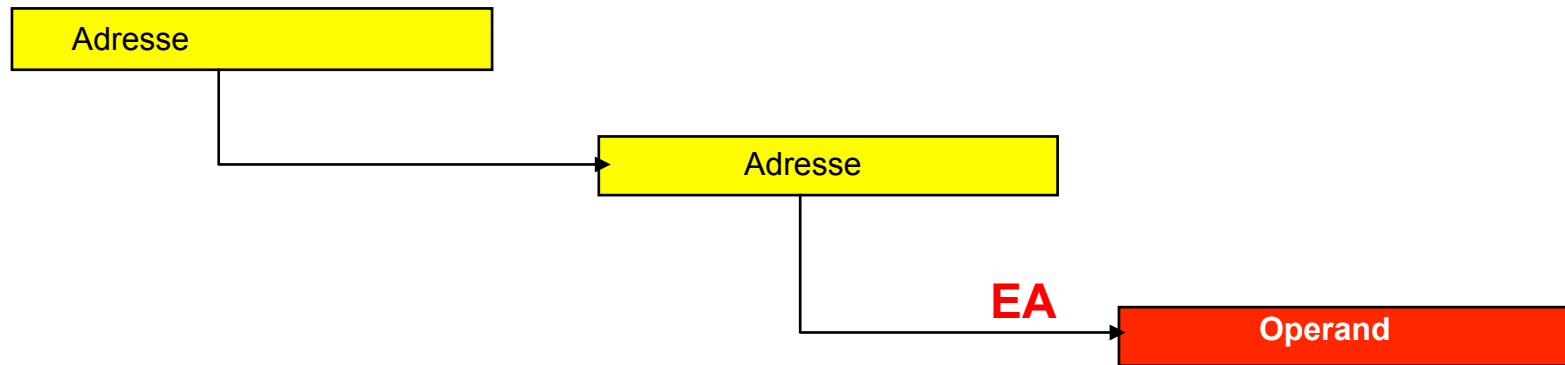




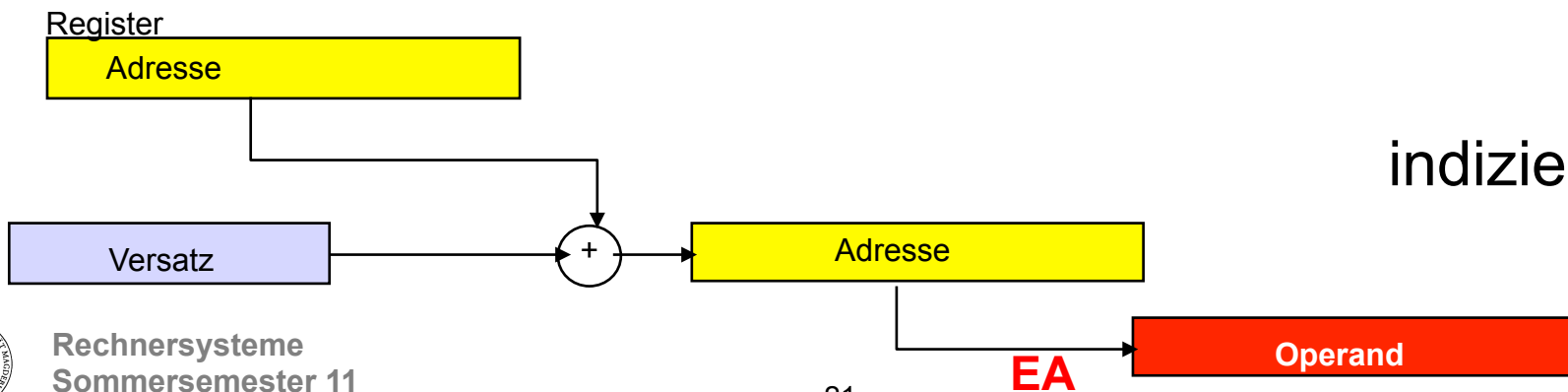
direkt



indiziert



indirekt



indiziert indirekt

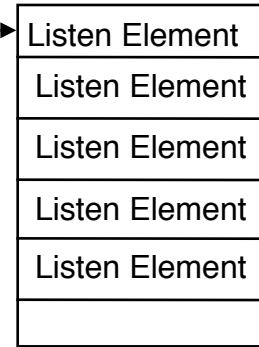


```

LDX HEAD_OF_LIST
COMP CMPX End_1st
      BHI Fin
      LDA ,X+
      CMPA #77
      BNE COMP
      BSR EF
Fin   BSR NF
      .
End_1st FDB $ABCD

```

**Index Reg.**



Indiziert mit (Post-) Autoincrement

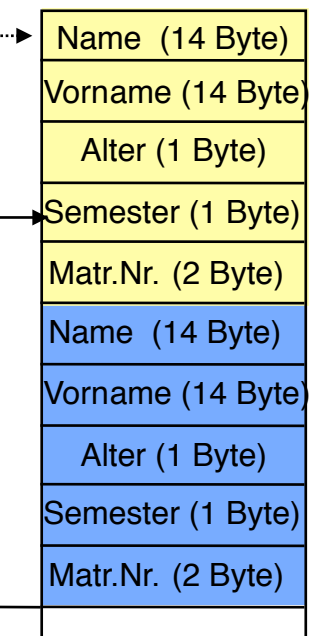
Branch to "element found"  
Branch to "element not found"

```

LDX HEAD_OF_LIST
LDA Sem_O
COMP LDB A,X
      CMPB S_zahl
      BHI OUTPUT
      LEAX RL,X
      CMPX End_1st
      BHI Fin
      BRA COMP
      .
      .
      .
RL   FCB 32
Sem_O FCB 29
S_zahl FCB 12
End_1st FDB $ABCD

```

**Index Reg.**



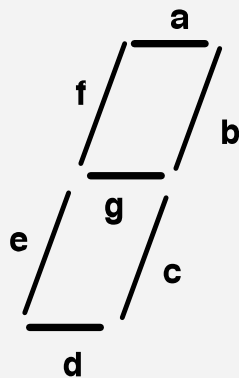
Indiziert mit Register A Offset  
Falls Semesterzahl >12:  
Ausgabe: Name, Vorname  
Addieren der Record-Länge\*  
Ende der Liste erreicht?

Versatz

\* da es keinen Additionsbefehl für das  
Indexregister X gibt, wird LEAX verwendet.

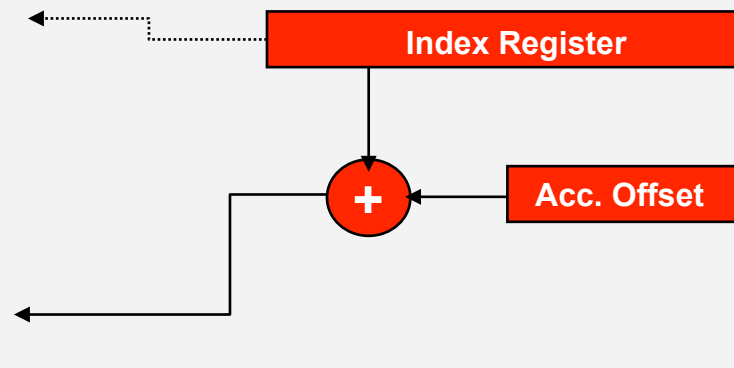


SP. Adr	OPC	OP	SP.M	MNE.	OP	Kommentar
A000	3F			CLRB		setzt den Fehlercode : löschen des Displays
A001	96	A0		LDA	\$A0	Wert in den Acc. A holen
A003	81	09		CMPA	#9	ist der Wert eine Ziffer 0...9 ?
A005	22	05		BHI	DONE	wenn A > 9 bleibt der Fehlercode gesetzt
A007	8E	AF00		LDX	#SSG	Lade X mit dem Beginn der Conversionsliste "SSG"
A00A	E6	86		LDB	A, X	Lade 7-Segment Muster in Acc A
A00C	D7	.....	DONE	STB	MMSEG	Abspeichern auf ein Ausgabegerät
A010	39			RTS		Return from Subroutine
AF00		SSG	FCB	\$3F, \$06, \$5B, \$4F, \$66		
AF05			FCB	\$6D, \$7D, \$07, \$7F, \$6F		
.....		MMSEG	.....	.....		Adresse des Ausgabegeräts



**7-Segment Code:**

0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F



7	6	5	4	3	2	1	0
0	g	f	e	d	c	b	a

**Zuordnung von Segmenten zu Bitpositionen im Ausgabewort**



# 6809 Indizierte Adressierung (Indexed Addressing)

Modi für indizierte Adressierung:

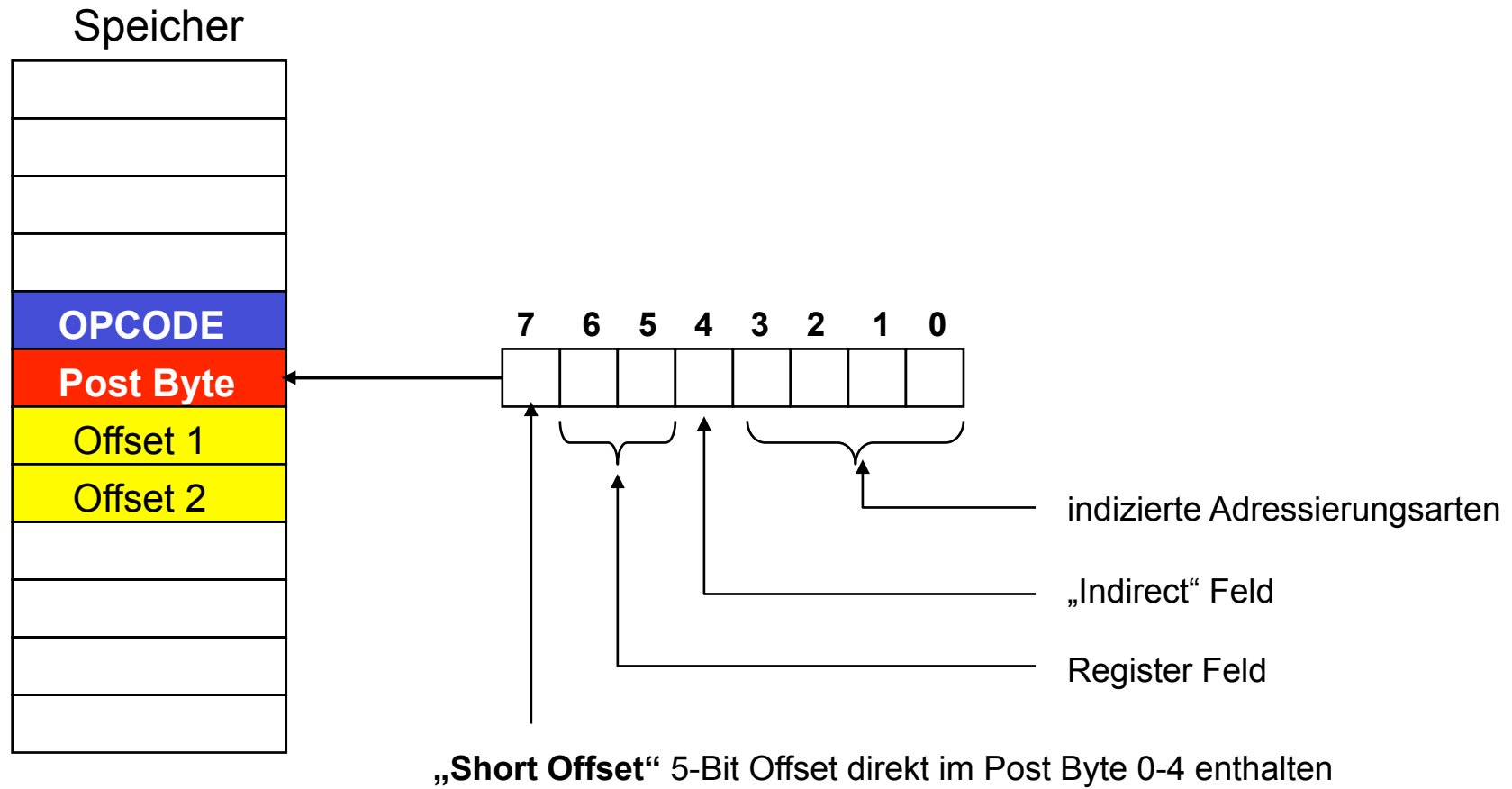
Konstanter Versatz	Langform (zusätzliches Byte)	lange Form(16 Bit)
	Kurzform (Spez. im Post Byte)	kurze Form (8 Bit)
	Programmzähler Offset	Versatz = 0 Versatz ist 5-Bit lang 8/16 Bit
Accumulator Offset	A, B und D	
Auto Increment	um 1 oder 2	
Auto Decrement	um 1 oder 2	
Extended Indirect		

Assembler Notation:

Notation	Bedeutung
,R offset ,R label ,PCR	das Register "R" (X, Y, U, S, PC) ist als Indexregister spezifiziert offset wird zum Indexregister "R" addiert Programmzähler-relative Adressierung – den Versatz zwischen PC und Label wird in der Assemblierungsphase berechnet und als konstanter Versatz im Modus: "constant PC-Offset" eingesetzt
,R+(+)	Autoincrement um 1 (bzw. 2) des Registers "R" (X, Y, U, S, PC) (Postincrement)
, -(-)R	Autodecrement um 1 (bzw 2) des Registers "R" (X, Y, U, S, PC) (Predecrement)
[.....]	indizierte Adresse ist eine indirekte Adresse



# Codierung der Indizierten Adressierungsarten



# Belegung des Post Bytes

Bit Number								Addressing Mode
7	6	5	4	3	2	1	0	
0	R	R	X	X	X	X	X	5-Bit Offset
1	R	R	0	0	0	0	0	Autoincrement by 1
1	R	R	i	0	0	0	1	Autoincrement by 2
1	R	R	0	0	0	1	0	Autodecrement by 1
1	R	R	i	0	0	1	1	Autoincrement by 2
1	R	R	i	0	1	0	0	Zero Offset
1	R	R	i	0	1	0	1	Acc. B Offset
1	R	R	i	0	1	1	0	Acc. A Offset
1	R	R	i	1	0	0	0	8-Bit Offset
1	R	R	i	1	0	0	1	16-Bit Offset
1	R	R	i	1	0	1	1	Acc. D Offset
1	X	X	i	1	1	0	0	PC 8-Bit Offset
1	X	X	i	1	1	0	1	PC 16-Bit Offset
1	R	R	1	1	1	1	1	Extended Indirect

Addressing Mode

Indirect Field:  
i=1 for indirect  
i=0 for direct  
sign bit when bit 7 =0

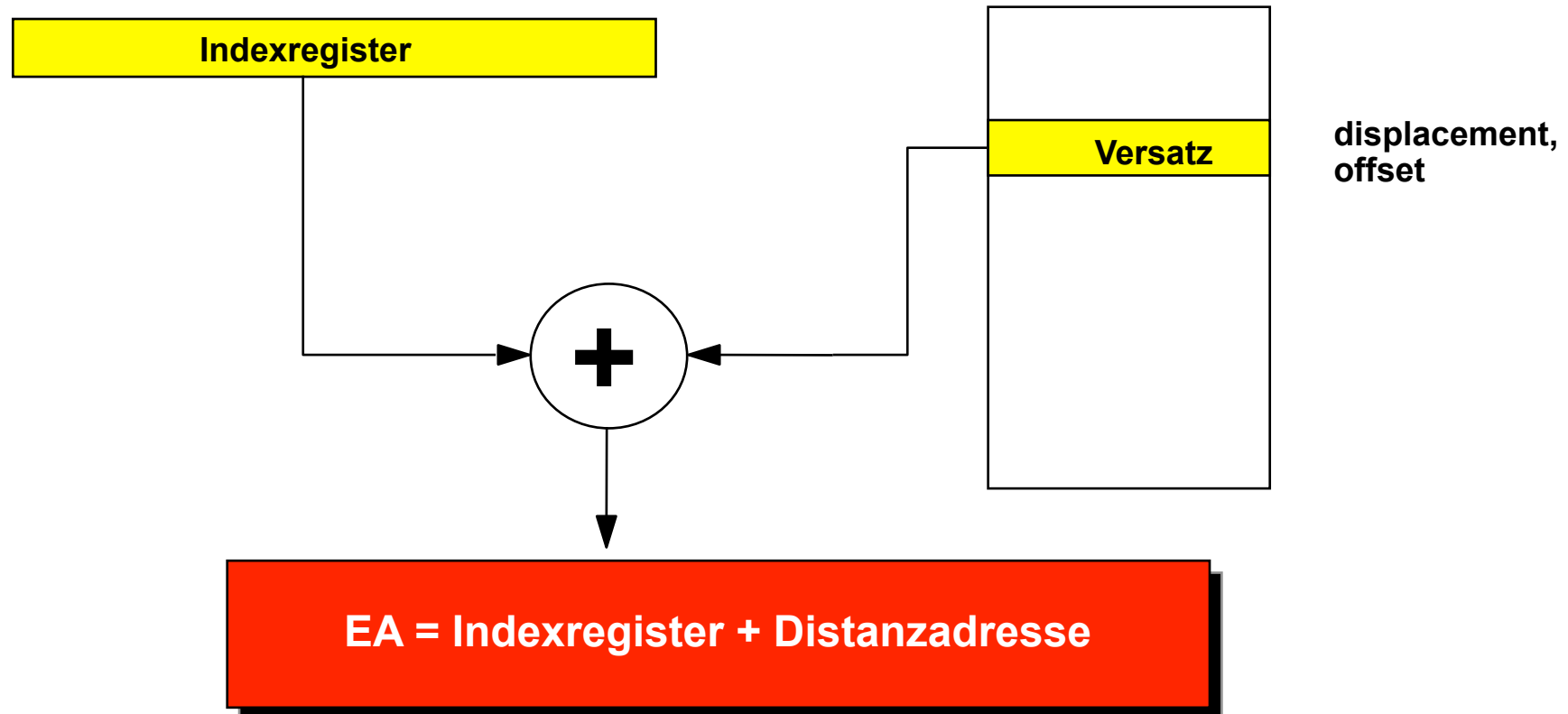
Register Field:

00 R = X  
01 R = Y  
10 R = U  
11 R = S



# Erweiterungen für die Indizierte Adressierung

## Basisform

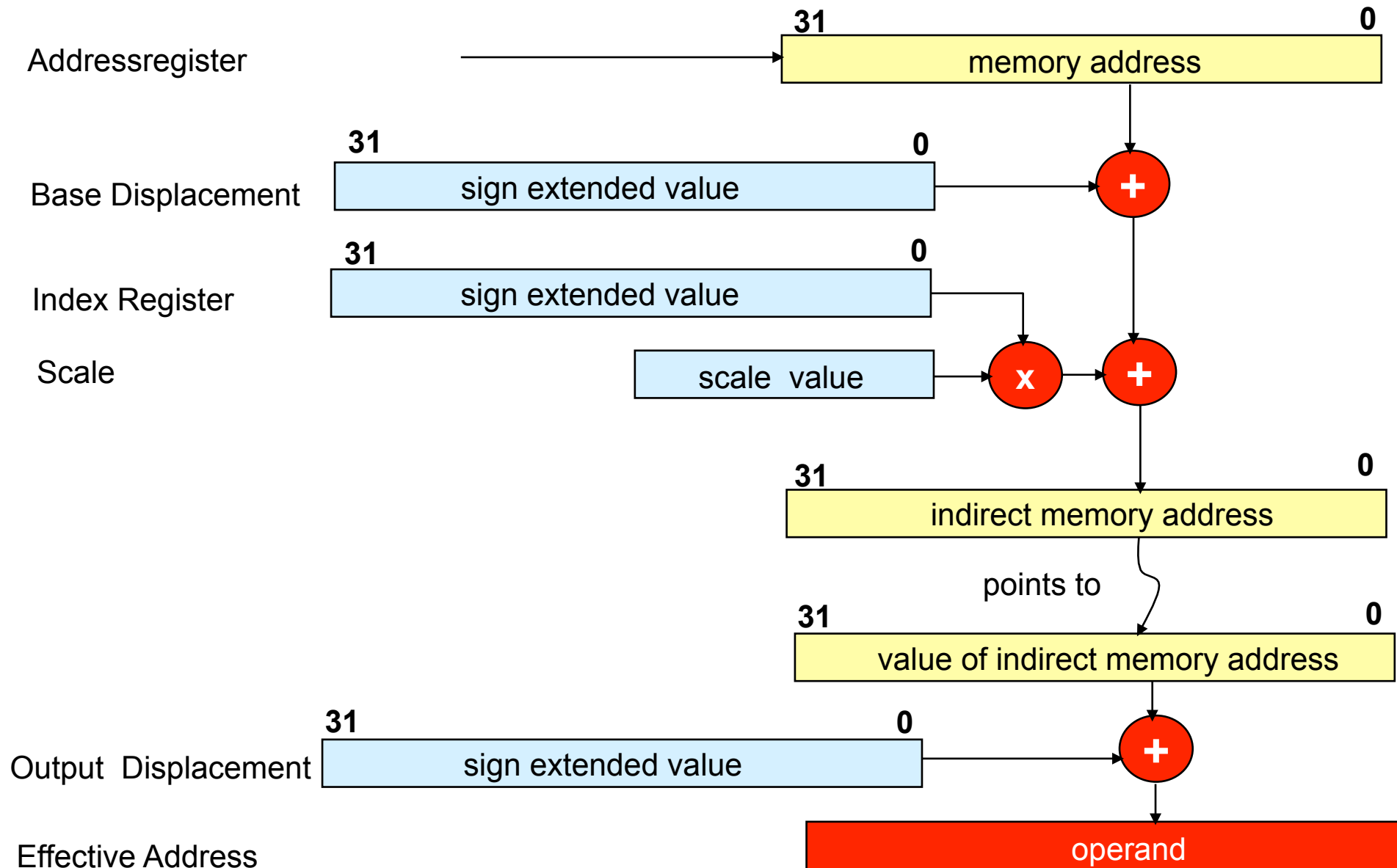


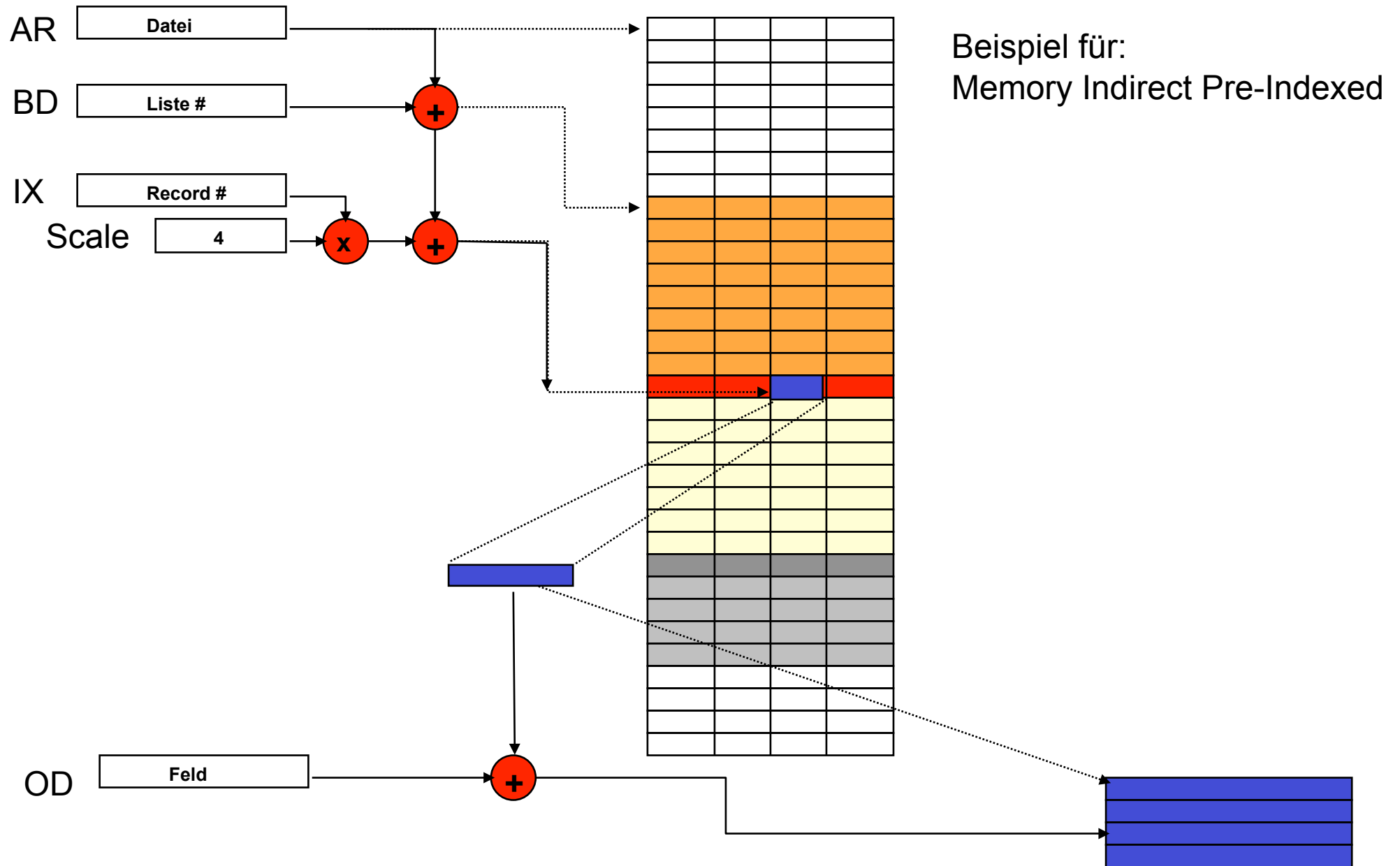
## Erweiterungen:

- skaliertes Index
- zusätzliche feste Distanzadressen
- Indirektion



## Indiziert, indizierter, am ... Memory indirect pre-indexed Adressierungsart bei MC 68020





# Architektur Varianten

---

## **Stack orientiert:**

Burroughs, Java VM

## **Speicher orientiert:**

IBM 360 series, Accumulator-Architekturen, frühe Mikroprozessoren

## **Register orientiert:**

DEC Alpha, ARM, Sparc, MIPS

## **Hybrid:**

DEC VAX, Motorola 68K, IBM PowerPC, Intel x 86, Pentium II



# Architektur Varianten

---

## ➔ Instruktionen mit 0 Operanden (implizite Adressierung):

Hit, NOP,

in Stack Architekturen auch Push, Pop und arithmetische und logische Befehle wie Add, Sub, Mult,...

## ➔ Instruktionen mit 1 Operanden:

Not, Increment, Decrement, Jmp, Bra,...

Bei Accumulatorarchitekturen auch alle zweistelligen arithm./log. Befehle

## ➔ Instruktionen mit 2 Operanden:

Datentransferbefehle wie Mov, Load, Store,

zweistellige arithm./logische Operationen, ein Operand wird vom Resultat überschrieben.

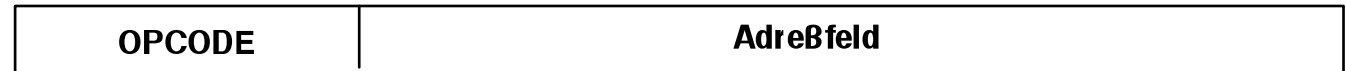
## ➔ Instruktionen mit 3 Operanden:

zweistellige Operationen: Ziel ← Quelle1 op Quelle2



## Ein-Adreß-Befehl

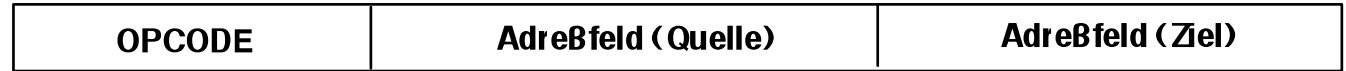
Beisp.:  
Motorola 6809



## Zwei-Adreß-Befehl

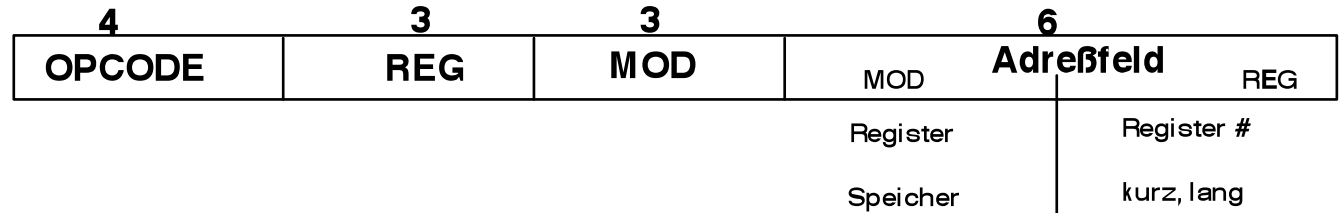
**Befehls-  
formate:**

680x0,  
Intel 386,486



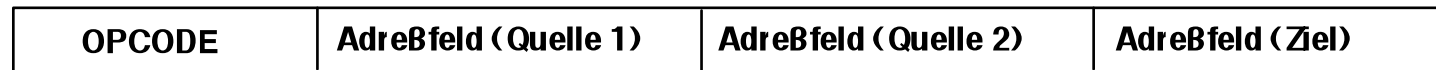
2.ter Quelloperand steht an  
Zieladresse und wird durch  
Operation überschrieben

Beisp.:  
Motorola  
680x0

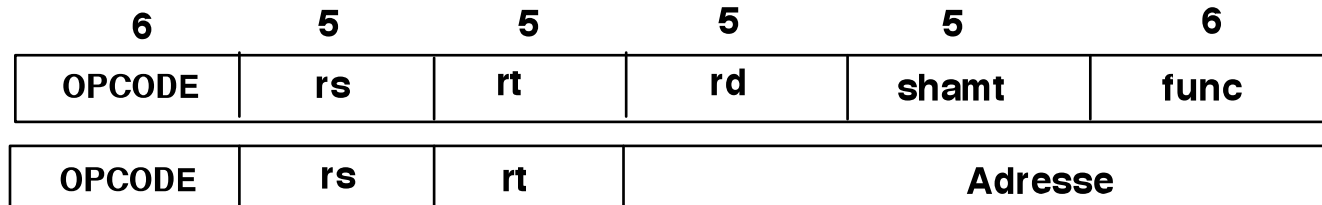


## Drei-Adreß-Befehl

MIPS,  
Alpha,  
SPARC



Beisp.  
MIPS



rs: erstes Quellreg.  
rt: zweites Quellreg.  
rd: Zielreg.  
shamt : shift amount  
func: funktionale Erw.  
des OPCODE



# Lernziele

---

- **Aufbau eines realen einfachen Prozessors**
  - **Befehlssatz**
    - **Standardbefehle (arithm./log., Shift, Transfer, System)**
    - **Sprünge und Unterprogrammaufruf**
    - **Behandlung von Bedingungen**
    - **Spezifikation von bedingten Verzweigungen**
  - **Adressierungsarten**
    - **Zielkonflikte bei der Adressierung**
    - **gängige Adressierungsarten**
- **einfache Programme in Assembler unter Ausnutzung der Befehls- und Adressierungsmöglichkeiten.**

