


# 9. Assembler: Der Prozessor Motorola 68000

## 9.1 Beschreibung des Prozessors M 68000

## 9.2 Adressierungsarten des M 68000

	Programmierkurs II © Prof. Dr. W. Effelsberg	9. Assembler: Der Prozessor Motorola 68000	9 - 1
---	---	---	-------

# 9.1 Beschreibung des Prozessors M 68000

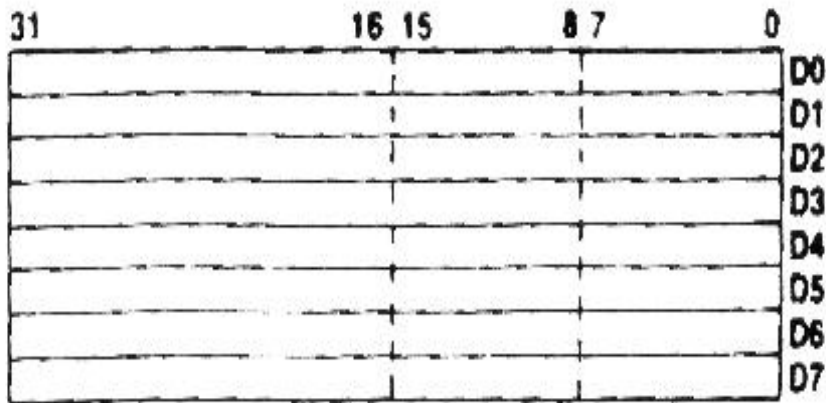
## Charakteristische Daten des Motorola 68000

- 56 Maschinenbefehle
- 14 Adressierungsarten
- Zweiadressmaschine mit
  - acht Datenregistern
  - neun Adress-/ Stackregistern
  - einem Befehlszähler (PC)
  - einem Statusregister
- CPU-intern 32- Bit-Rechnerorganisation
- Busbreite 16 Bit
- Taktfrequenz 4-12,5 MHz (je nach Modell)

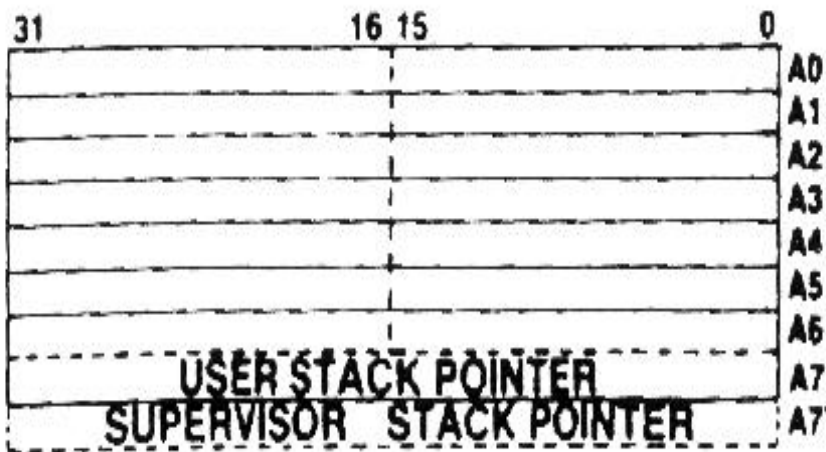
“Urvater“ einer sehr erfolgreichen Prozessor-Familie (68010, 68020, 68030, 68040)

	Programmierkurs II © Prof. Dr. W. Effelsberg	9. Assembler: Der Prozessor Motorola 68000	9 - 2
---	---	---	-------

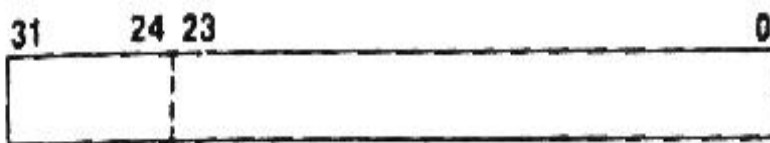
# Registermodell des M 68000



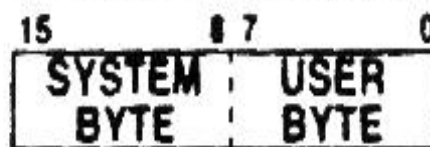
Acht  
Datenregister



Neun  
Adress-/Stack-  
Register

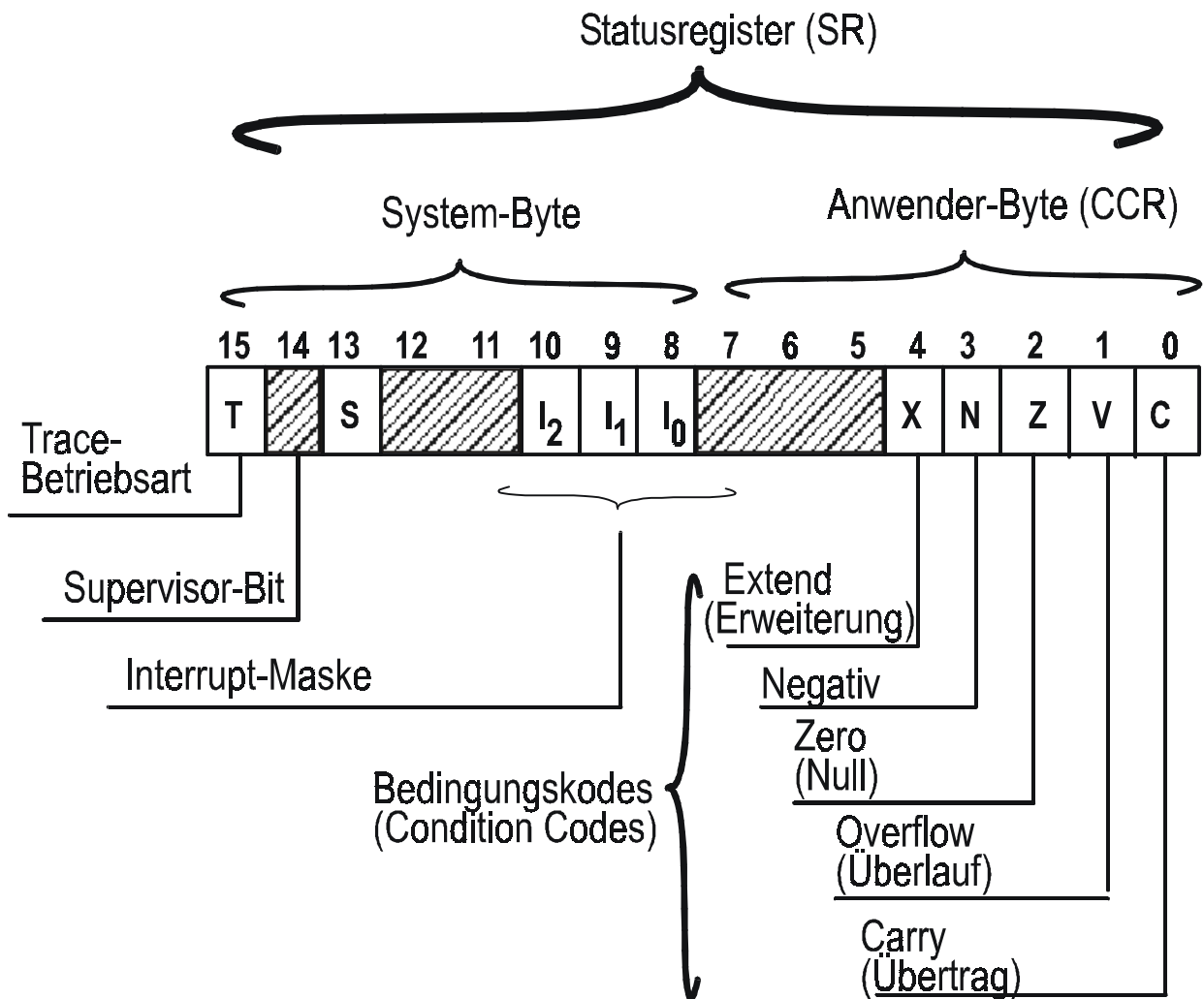


Programmzähler



Statusregister

# Statusregister



System - und User - (Anwender -) byte im Statusregister

## Die Flags im Status-Register SR (1)

Das **Carry-Flag C** wird immer dann auf Eins gesetzt, wenn bei einer arithmetischen Operation im M68000 (z. B. Addieren, Schieben) aus dem vordersten Bit des Zieloperanden (Bit 31 bei Langwort, Bit 15 bei Wort und Bit 7 bei Byte) eine Eins als Übertrag entsteht oder bei Subtraktion ein "Borgen" durchgeführt wird. Ein Beispiel für die Anwendung des Carry-Flags wäre die Addition von zwei Zahlen, die jeweils größer als 32 Bit sind.

Das **Overflow-Flag V** zeigt an, ob während der Durchführung einer arithmetischen Operation ein Überschreiten eines Zahlenbereichs aufgetreten ist, d. h. wenn z. B. bei Addition von zwei positiven Zahlen das Ergebnis eine negative Zahl im Zweierkomplement darstellt. Außerdem wird das Overflow-Flag benutzt, um bei einer Divisionsoperation anzuzeigen, dass der Dividend zu groß ist bzw. der Quotient größer als 16 Bit werden würde. Ist dieser Fall eingetreten, so wird das Overflow-Flag auf Eins gesetzt, ansonsten auf Null.

Das **Zero-Flag Z** wird auf Eins gesetzt, wenn bei einer Operation das Ergebnis gleich Null ist bzw. wenn bei einer Vergleichsoperation (CMP) beide Operanden gleich sind, ansonsten enthält es eine Null.

## Die Flags im Status-Register SR (2)

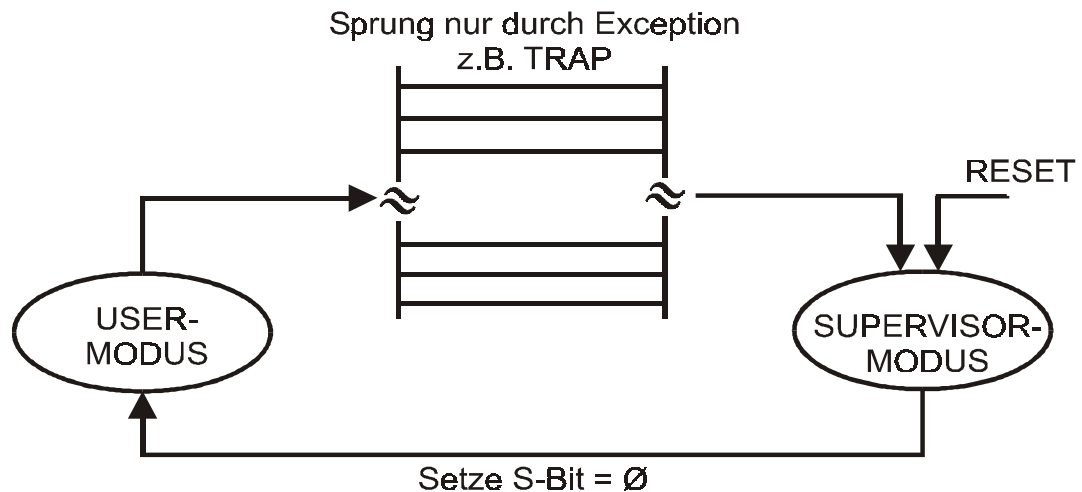
Das **Negativ-Flag N** steht nach der Ausführung einer Operation auf Eins, wenn das Ergebnis eine negative Zahl im Zweierkomplement darstellt, d.h. das höchstwertigste Bit im Ergebnis eine Eins ist. Ist dies nicht der Fall, so enthält dieses Flag eine Null.

Das **Extend-Flag X** ist eine Besonderheit des M68000. Generell lässt sich über dieses Flag sagen, dass es sich, wenn es gesetzt wird, genauso verhält wie das Carry-Flag. Bei einigen Befehlen jedoch wird das Carry-Flag verändert, das Extend-Flag nicht. Man kann das Extend-Flag dazu verwenden, den Inhalt des Carry-Flags über mehrere Operationen weg aufzubewahren. So wird das Extend-Flag z.B. bei Addition und Subtraktion genauso gesetzt wie das Carry-Flag, bei Vergleichsoperationen (CMP) jedoch nicht verändert. Für exakte Aussagen, wann das Extend-Flag verwendet wird und wann nicht, ist es am Besten, die einzelnen Befehlsbeschreibungen nachzulesen. Ein Beispiel für eine Anwendung des Extend-Flags im M68000 ist die Addition von zwei Zahlen, die jeweils größer als 32 Bit sind.

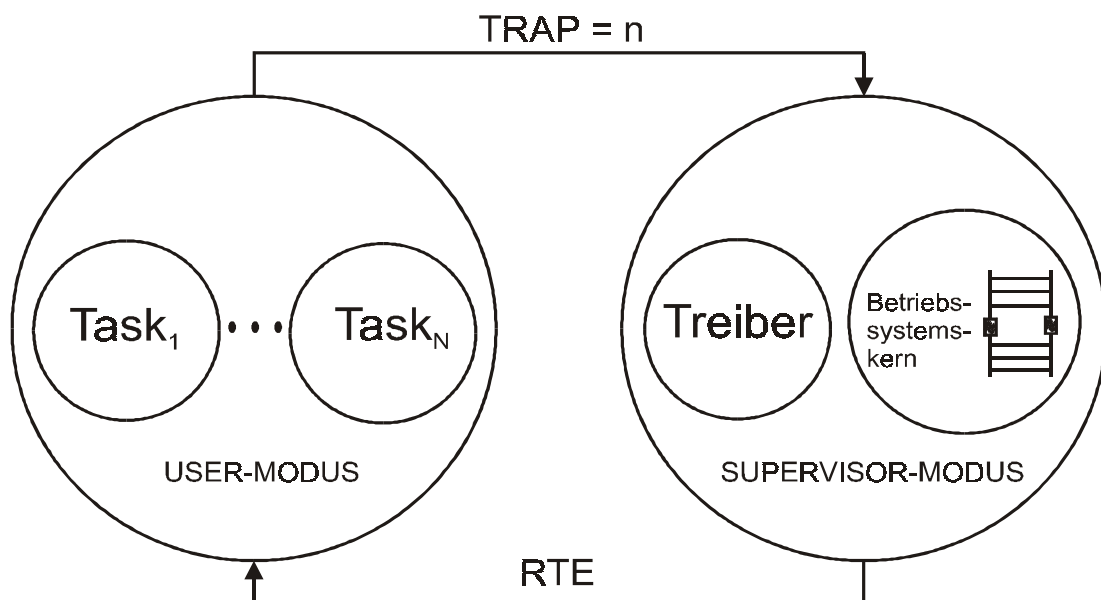


# User Mode und Supervisor Mode

## User / Supervisor-Modus-Übergang



## Typische Anwendung des User/Supervisor-Wechsels



# Supervisor Mode und privilegierte Befehle

Modus	S-Bit	Einschränkungen	
User	0	<b>Privilegierte Befehle, die nicht erlaubt sind</b>	
		<b>BEFEHL</b>	<b>BEDEUTUNG</b>
		RESET RYTE STOP ORI mit SR MOVE USP ANDI mit SR EORI mit SR MOVE EA zum SR	Rücksetzen externer Bausteine Return from Exception Programmausführung anhalten logisches ODER mit Statusregister MOVE zum User Stackpointer logisches UND mit Statusregister log. Exklusiv-ODER mit Statusreg. MOVE neuen Wert ins Statusreg.
Supervisor	1	Keine Einschränkungen - alle Befehle können ausgeführt werden.	

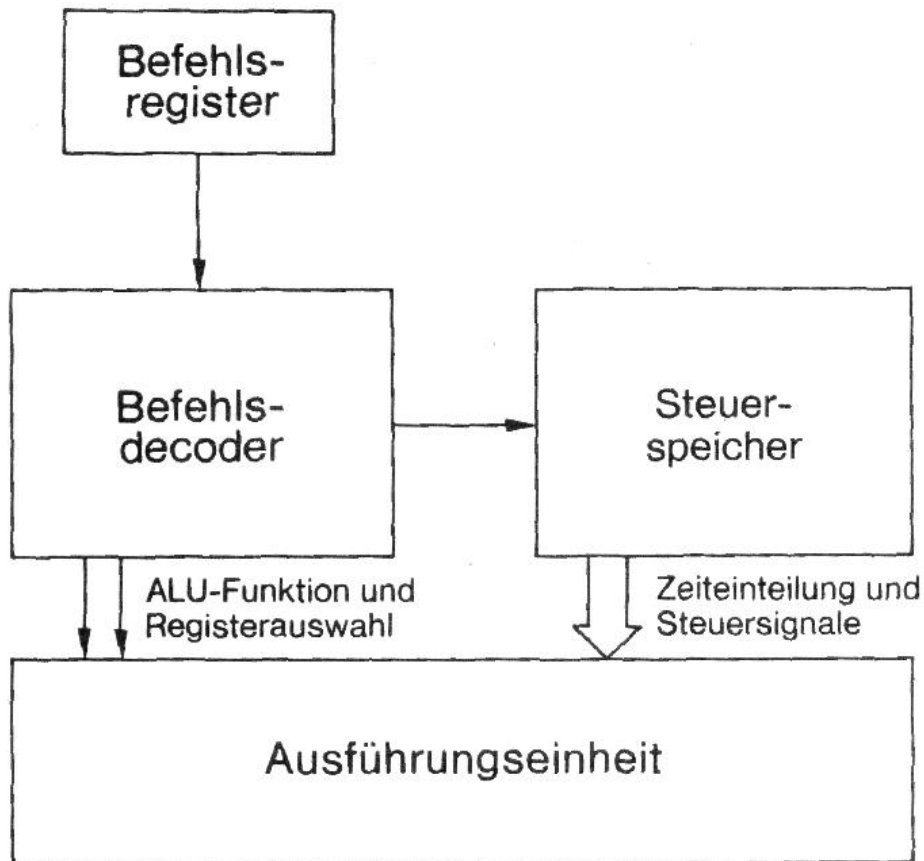
## Privilegierte Befehle



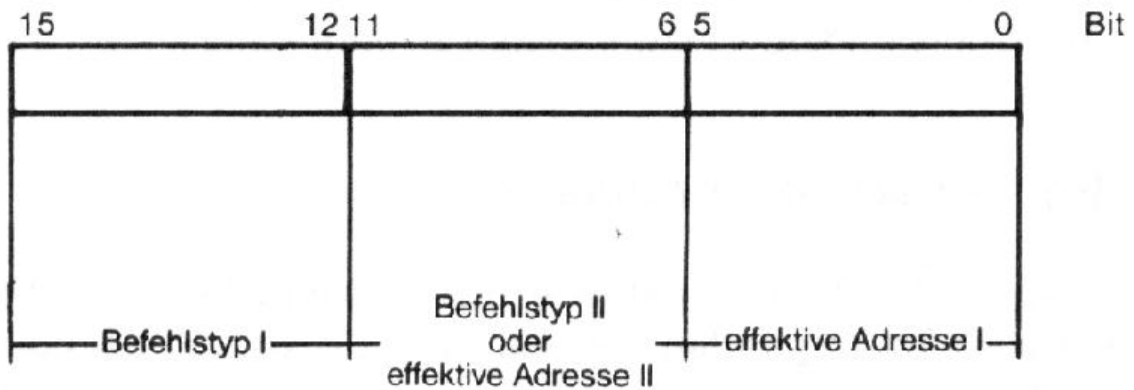
# Mikroprogramm-Steuerung

ca. 180 Steuersignale in der Ausführungseinheit  
(= Rechenwerk = ALU)

Speicher als mikroprogrammiertes ROM



## 9.2 Adressierungsarten des M 68000



Nr.	EA-Modus	EA-Reg.	Adressierungsart	Mnemonic
1	000	Reg.Nr.	Datenregister direkt	Dn
2	001	Reg.Nr.	Adressregister direkt	An
3	010	Reg.Nr.	Adressregister indirekt (ARI)	(An)
4	011	Reg.Nr.	ARI mit Postinkrement	(An)+
5	100	Reg.Nr.	ARI mit Predekrement	-(An)
6	101	Reg.Nr.	ARI mit Adressdistanz	d16(An)
7	110	Reg.Nr.	ARI mit Adressdistanz und Index	d8(An,Rx)
8	111	000	Absolut kurz	\$XXXX
9	111	001	Absolut lang	\$XXXXXXXX
10	111	010	PC relativ mit Adressdistanz	d16(PC)
11	111	011	PC rel. mit Adressdist. u. Index	d8(PC,Rx)
12	111	100	Konstante, Statusregister	#, SR, CCR
13	111	101	(nicht verwendet)	
14	111	110	(nicht verwendet)	
15	111	111	(nicht verwendet)	

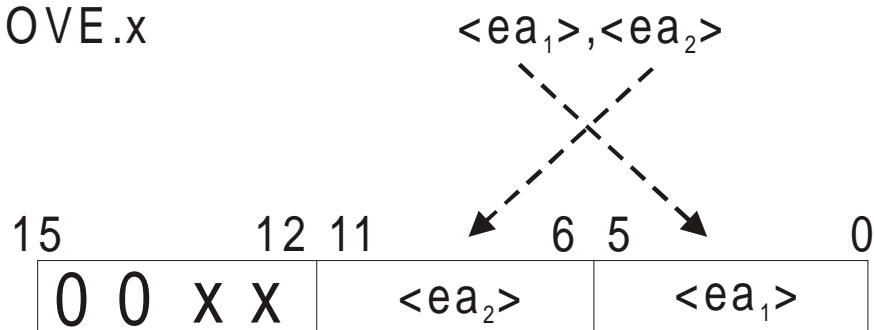
# Datenformate

Der M68000 verarbeitet folgende Datenlängen:

Name	Größe	Verarbeitung
Bit	1 Bit	Bitmanipulationsbefehle
BCD	4 Bit	BCD-Befehle (gepackt, d.h. zwei BCD-Zahlen in einem Byte)
Byte	8 Bit	versch. Befehle, externe Daten
Wort*	16 Bit	versch. Befehle, externe Daten
Langwort	32 Bit	versch. Befehle, externe Daten

\* Ein Wort ist die Standardverarbeitung, da der M68000 einen 16bit-Datenbus hat.  
Die Datenorganisation im Speicher zeigt Bild 2-10a.

MOVE.x



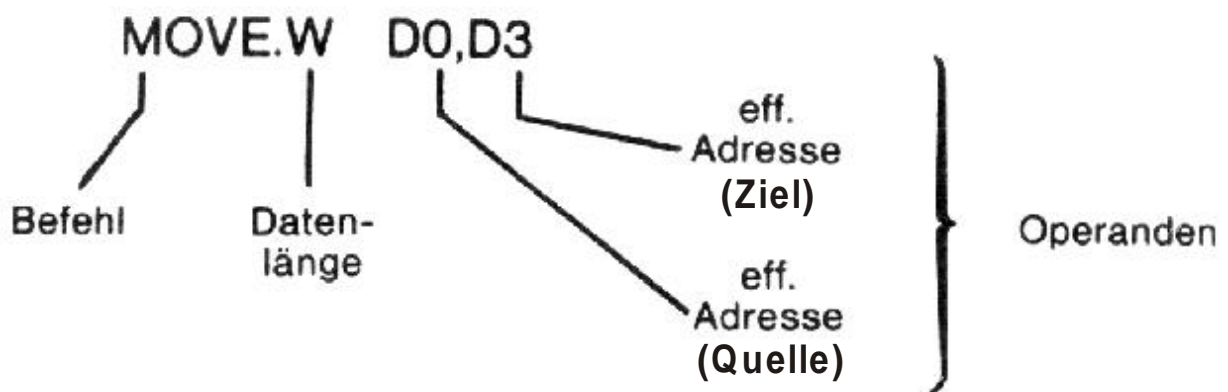
B - Byte                      8 Bit  
W - Wort                      16 Bit  
L - Langwort                  32 Bit

# Adressierungsart "Register direkt" (1)

Die Operandenadresse ist ein **Register** (Datenregister, Adressregister, Statusregister).

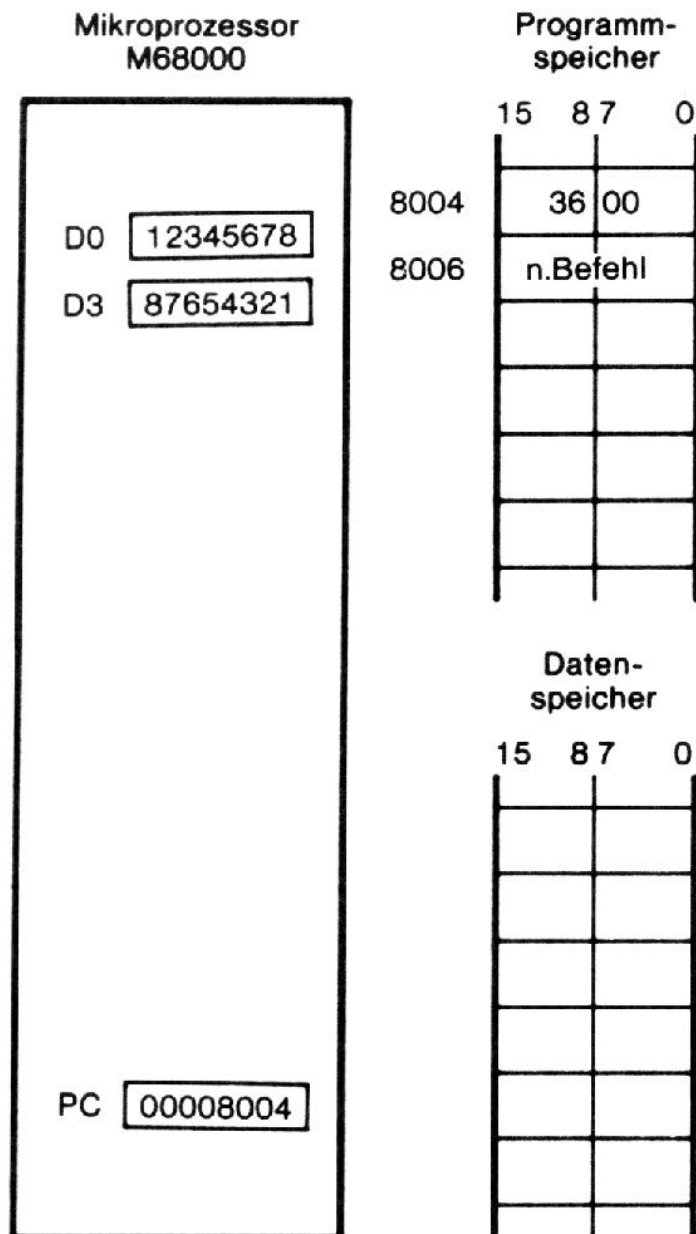
Beispiel:            MOVE    D0,D3

Gleichbedeutend mit:



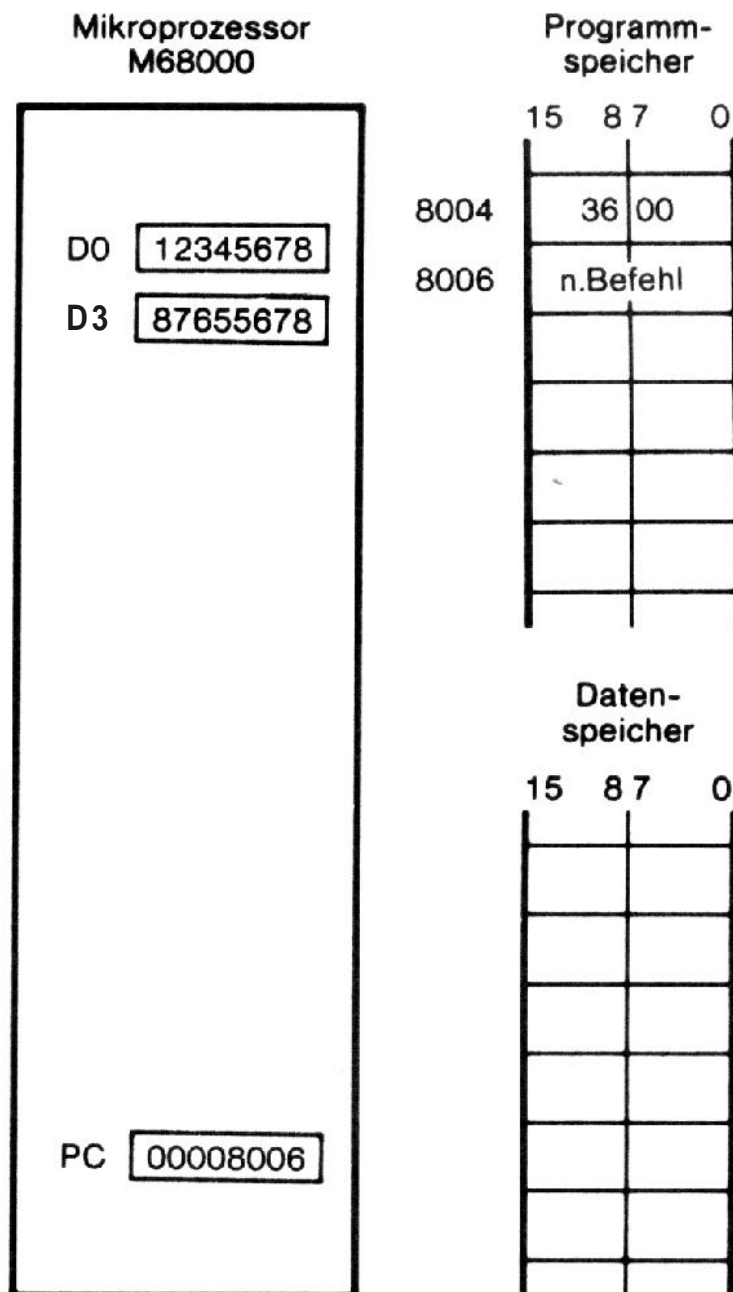
# Adressierungsart "Register direkt" (2)

## Vor Ausführung des Befehles



# Adressierungsart "Register direkt" (3)

Nach Ausführung des Befehles



# Absolute Adressierung (1)

Die Operandenadresse ist eine absolute Adresse im Speicher.

## Beispiel 1

MOVE \$1000,\$2000

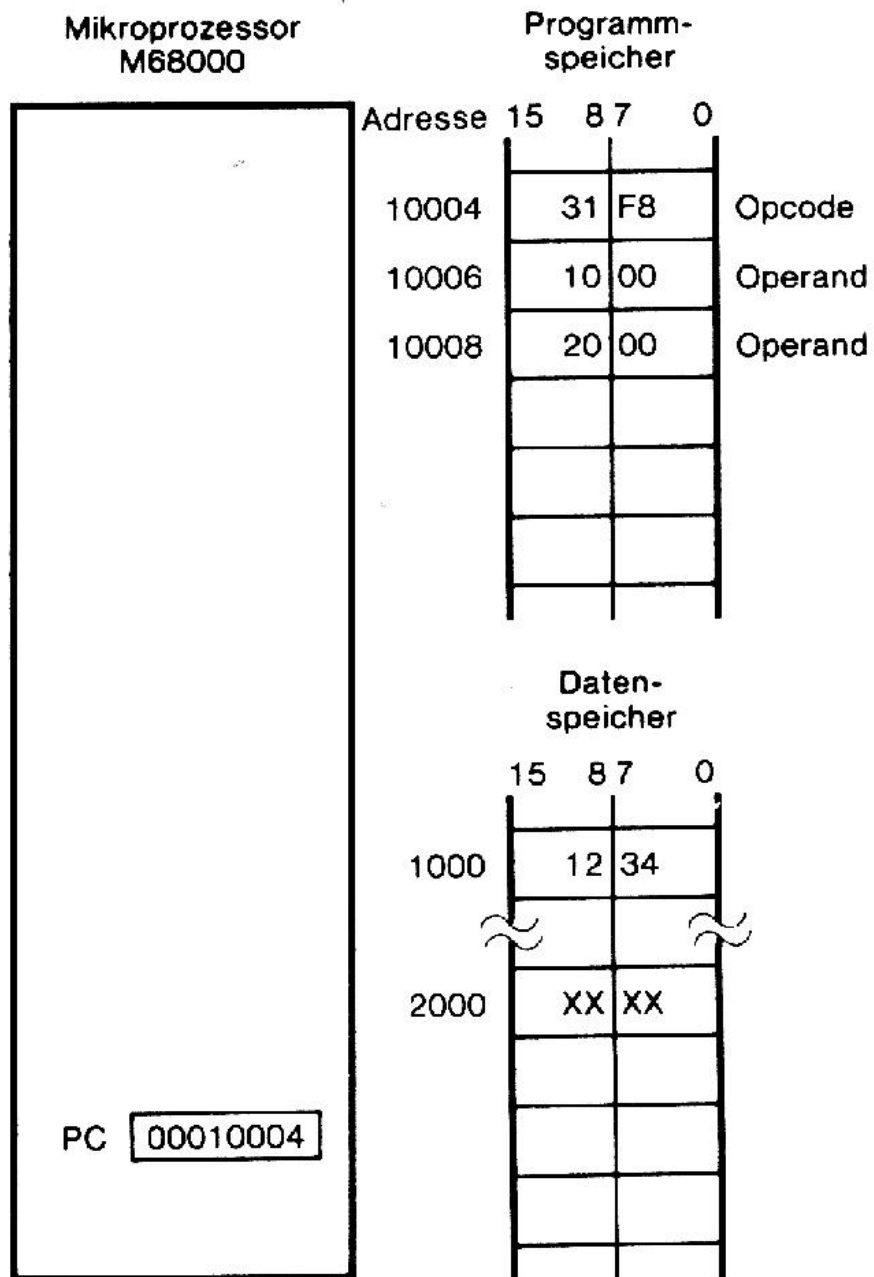
Gleichbedeutend mit:

MOVE.W \$1000,\$2000



# Absolute Adressierung (2)

## Vor Ausführung des Befehles

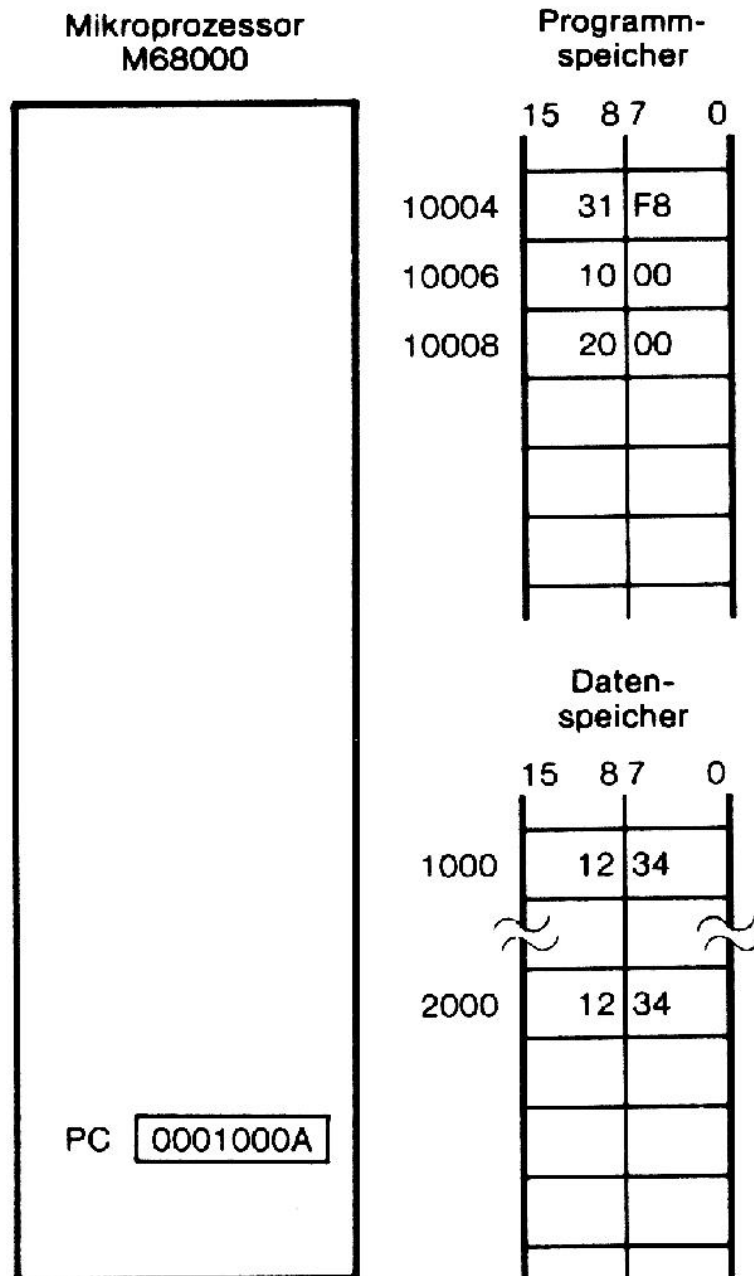


XXXX = undefinierter Wert



# Absolute Adressierung (3)

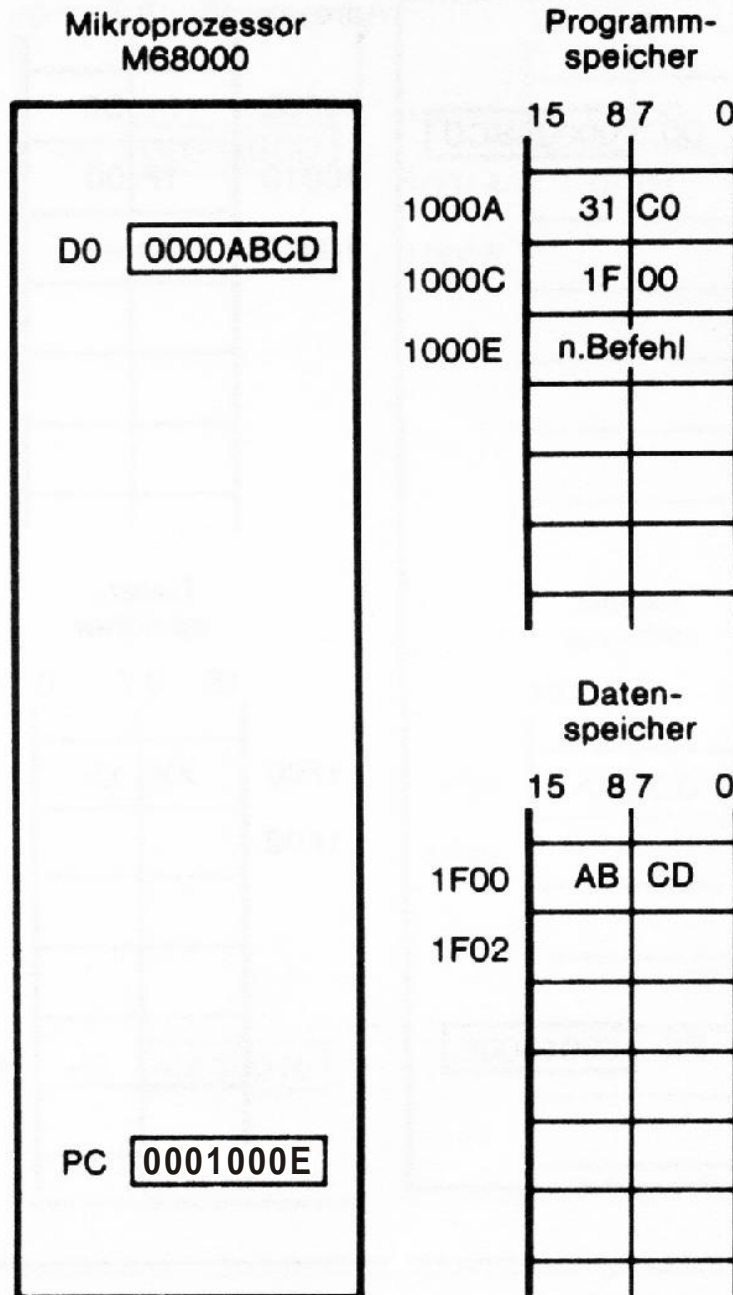
Nach Ausführung des Befehles





# Absolute Adressierung (5)

Nach Ausführung des Befehles



# Absolute Adressierung (6)

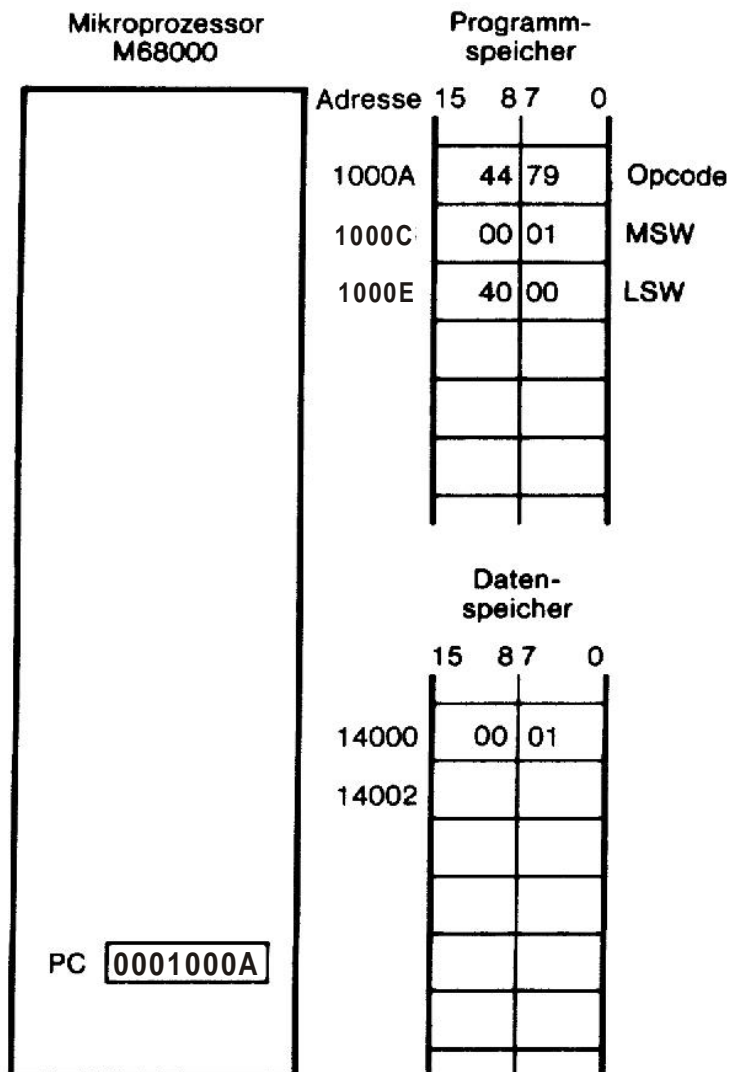
## Beispiel 3

NEG \$14000

Gleichbedeutend mit:

NEG.W \$14000

Vor Ausführung des Befehles

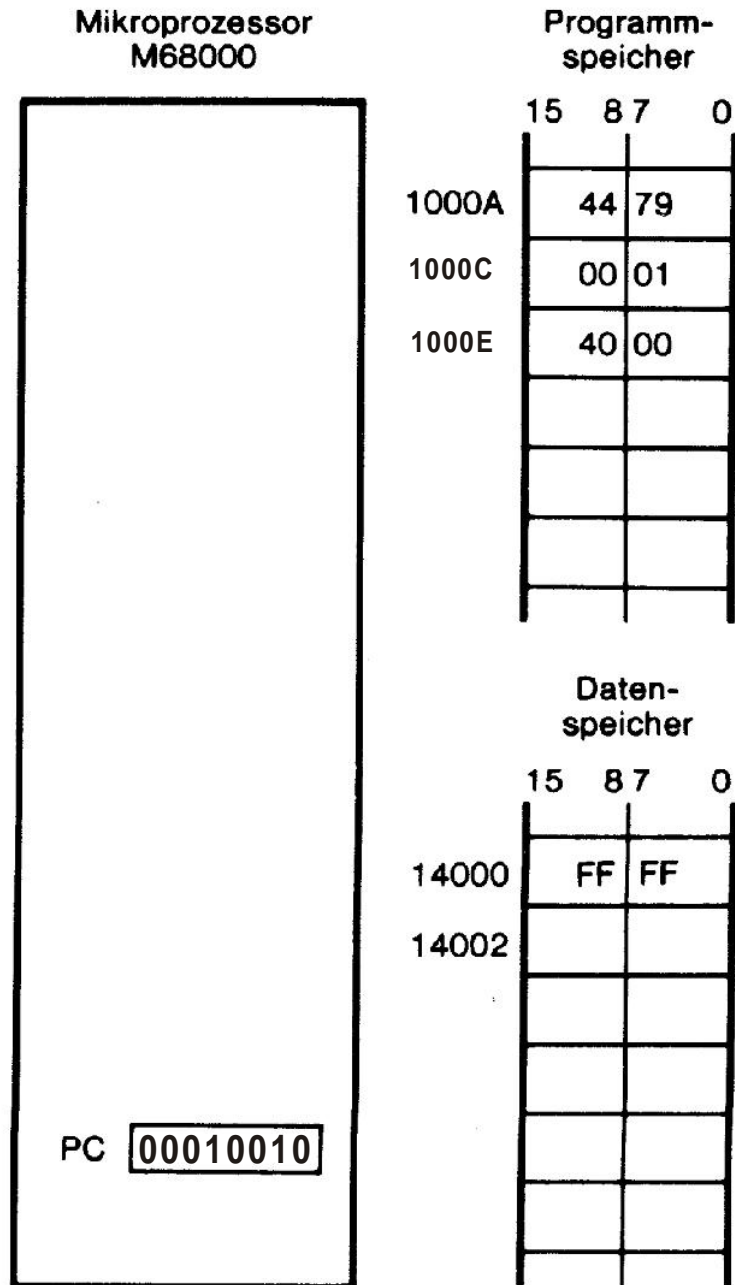


**MSW** = most significant word (höher wertiges Wort)

**LSW** = least significant word (nieder wertiges Wort)

# Absolute Adressierung (7)

Nach Ausführung des Befehles



# Konstanten-Adressierung (immediate addressing)

Der **Operand steht im Befehl** oder folgt unmittelbar dem Befehl im nächsten Wort oder den nächsten beiden Worten des Speichers.

Die Verarbeitung ist wesentlich schneller als bei absoluter Adressierung von Konstanten, die im Speicher abgelegt sind (nur eine "Fetch"-Operation).

Häufige Verwendung für kleine Ganzzahl-Konstanten (0, 1, -1).

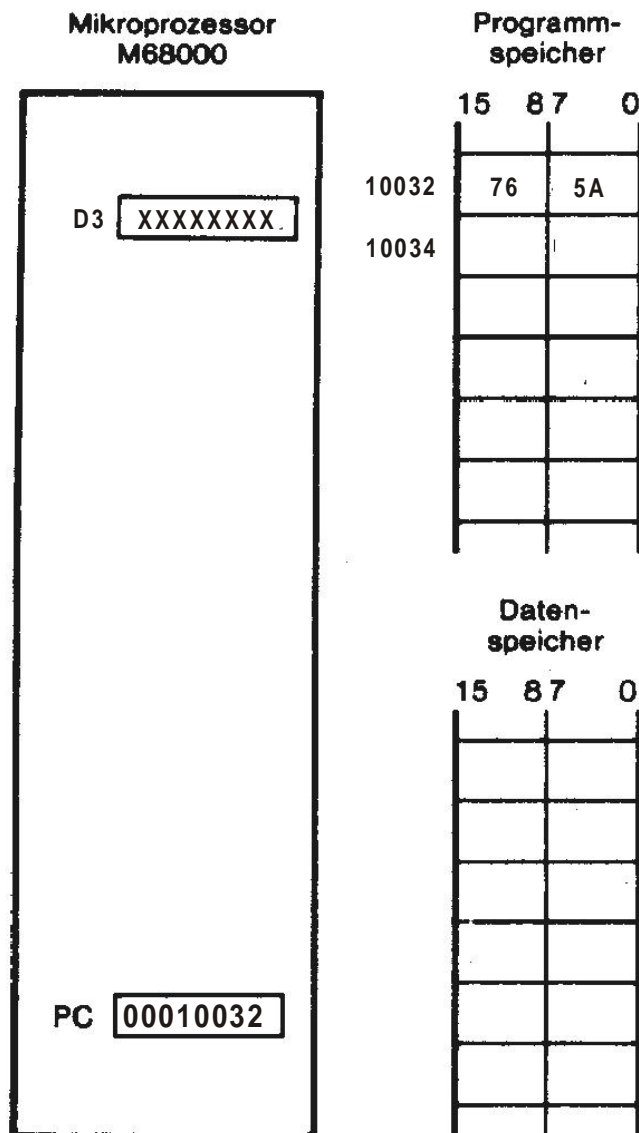


# Konstanten-Adressierung (1)

## Beispiel 1

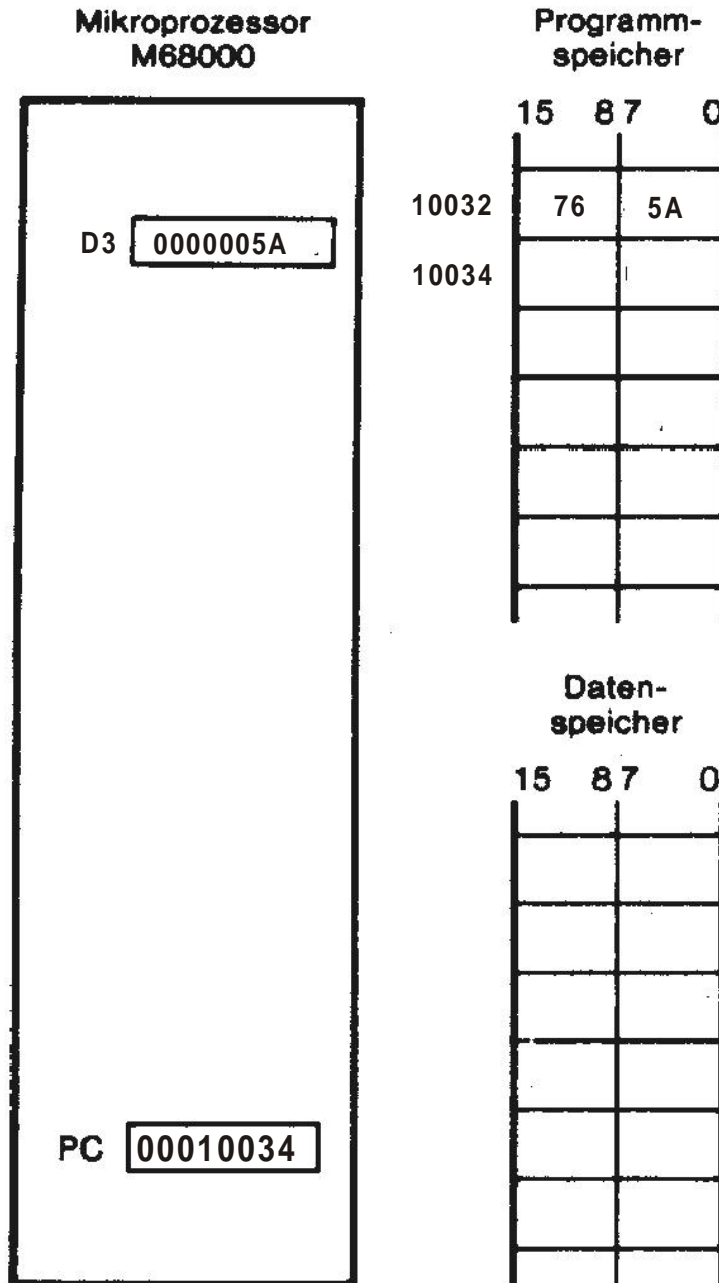
MOVEQ #\$5A,D3 (MOVEQ = MOVE QUICK)

### Vor Ausführung des Befehles



# Konstanten-Adressierung (2)

Nach Ausführung des Befehles

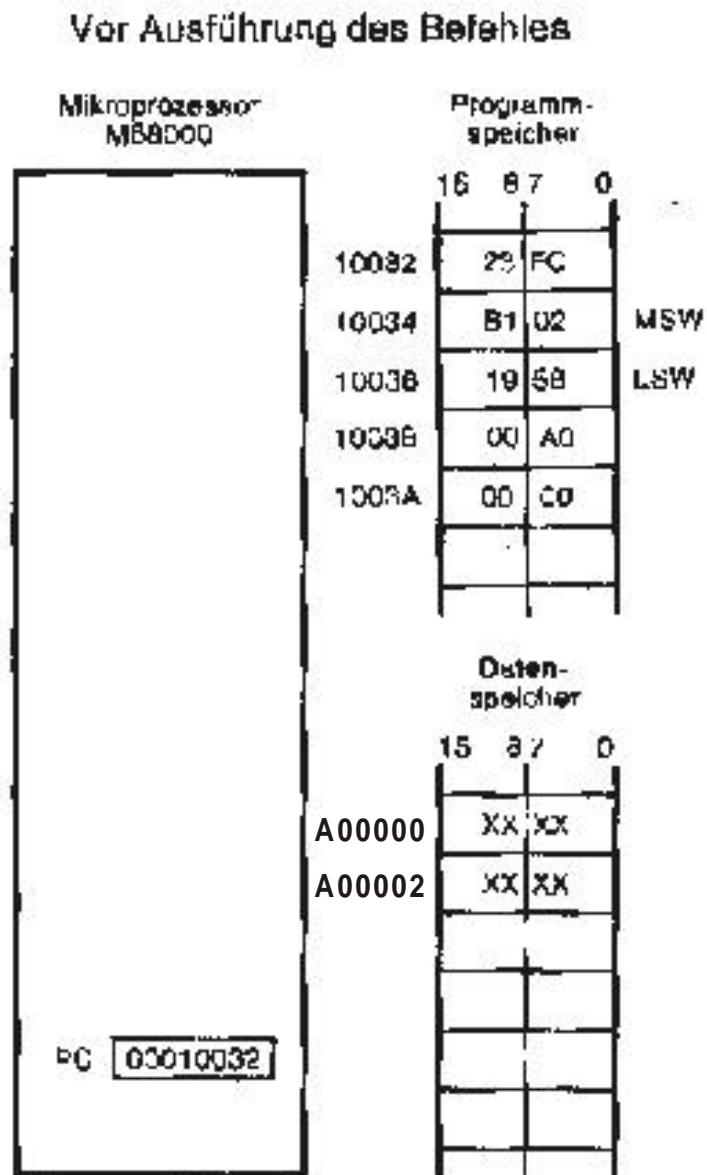




## Konstanten-Adressierung (3)

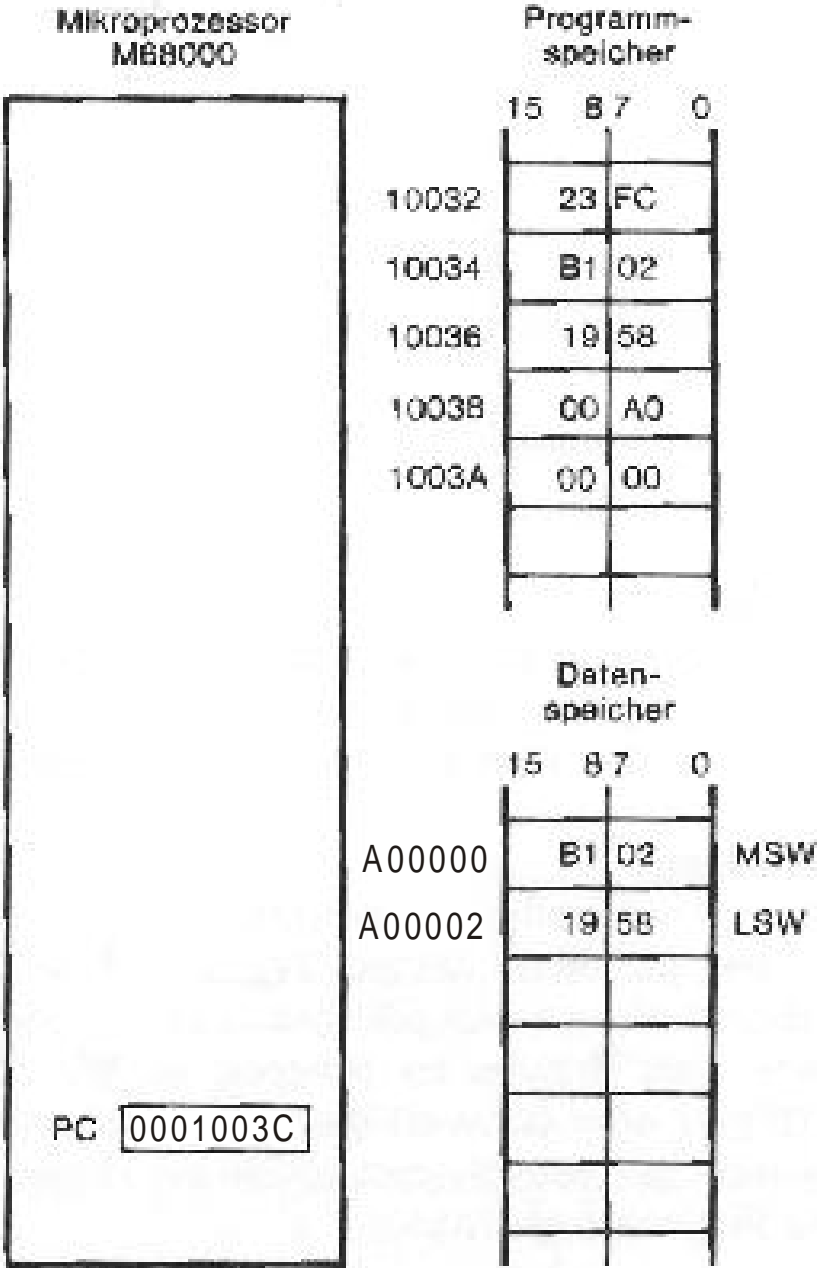
### Beispiel 2: Langwort- Konstante

**MOVE.L #\$B1021958, \$A00000**



# Konstanten-Adressierung (4)

Nach Ausführung des Befehles



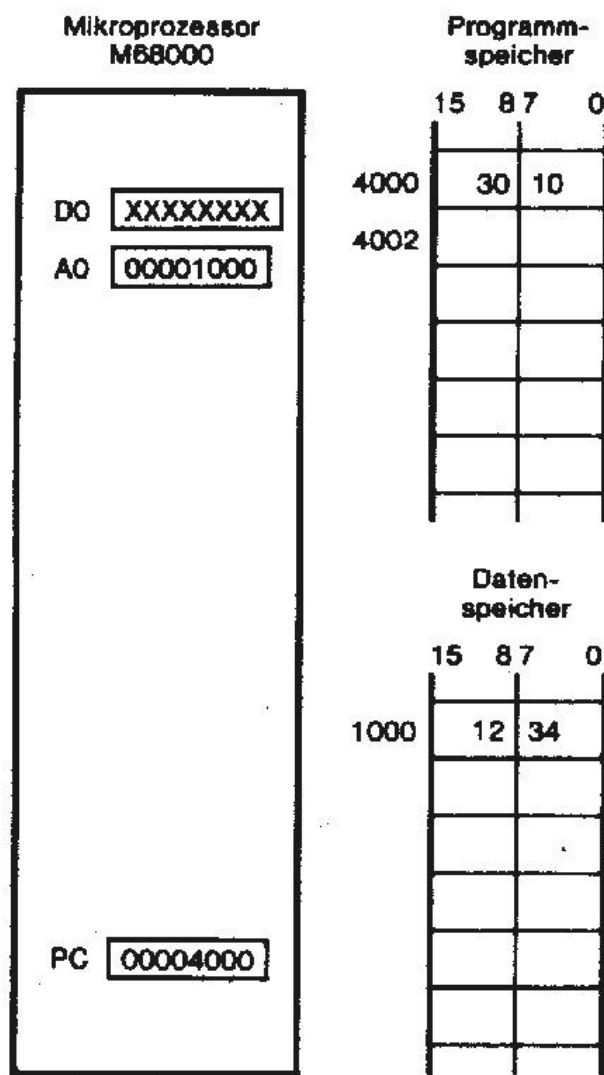
# Register-indirekte Adressierung (1)

Ein Adressregister enthält die Adresse des Operanden.

**Beispiel** MOVE (A0),D0

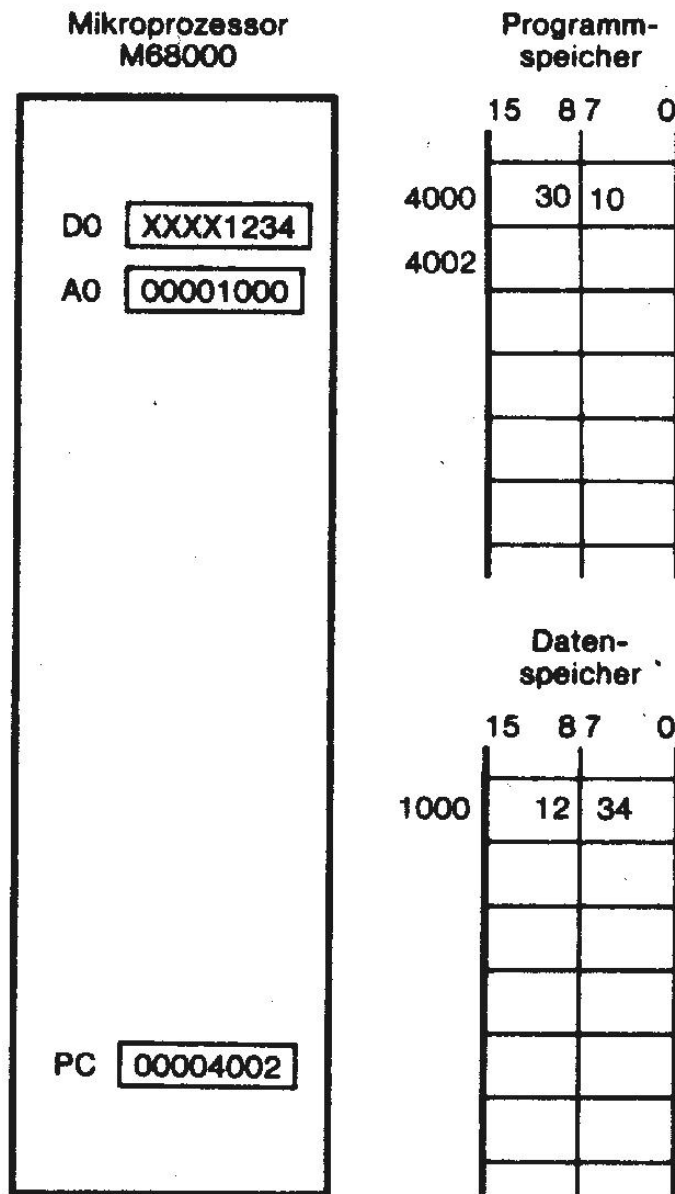
Gleichbedeutend mit MOVE.W (A0),D0

Vor Ausführung des Befehles



# Register-indirekte Adressierung (2)

Nach Ausführung des Befehles



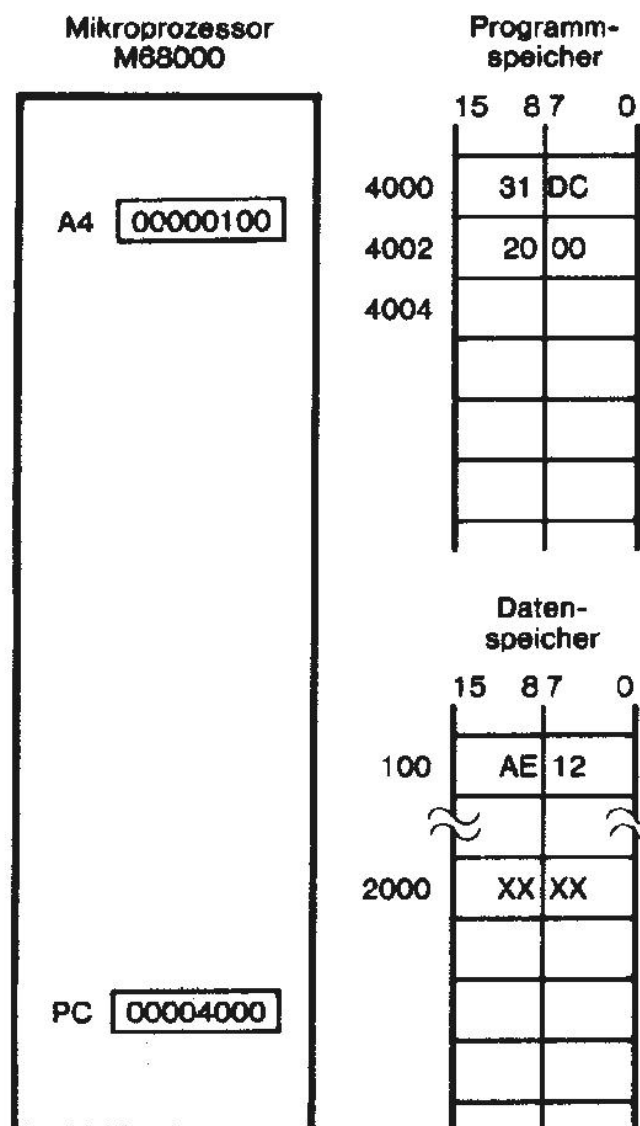
# Register-indirekte Adressierung (3)

## Beispiel 2: mit Postinkrement

MOVE (A4)+,\$2000

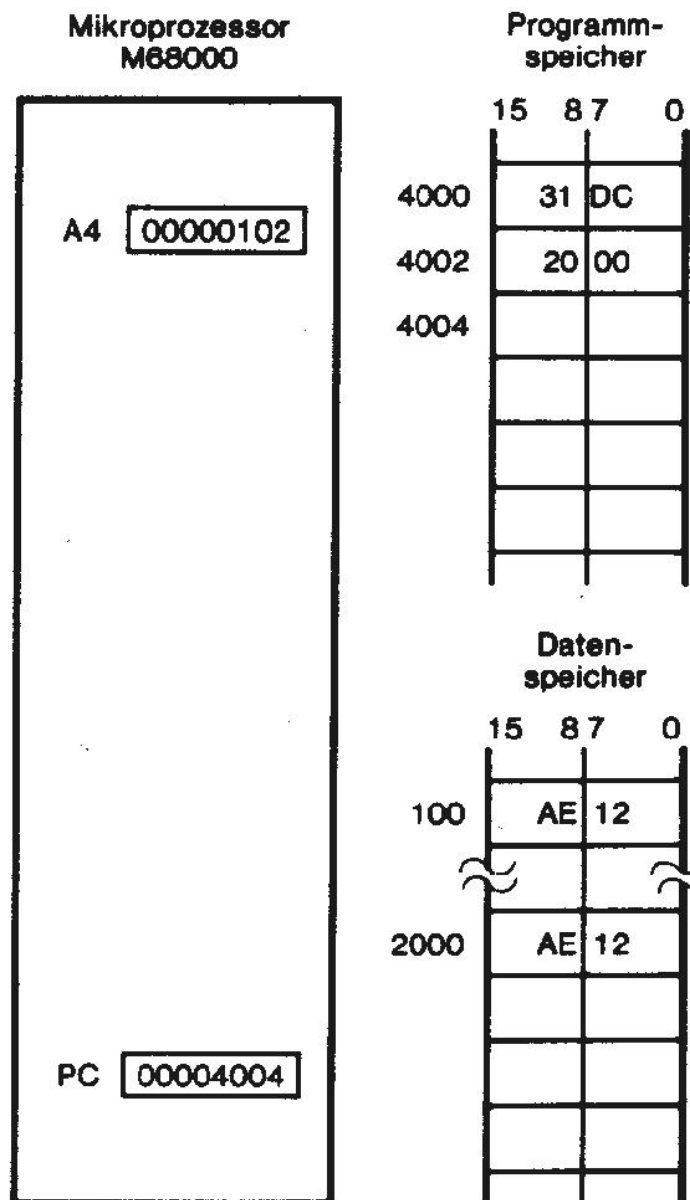
Gleichbedeutend mit: MOVE.W (A4)+,\$2000

### Vor Ausführung des Befehles



# Register-indirekte Adressierung (4)

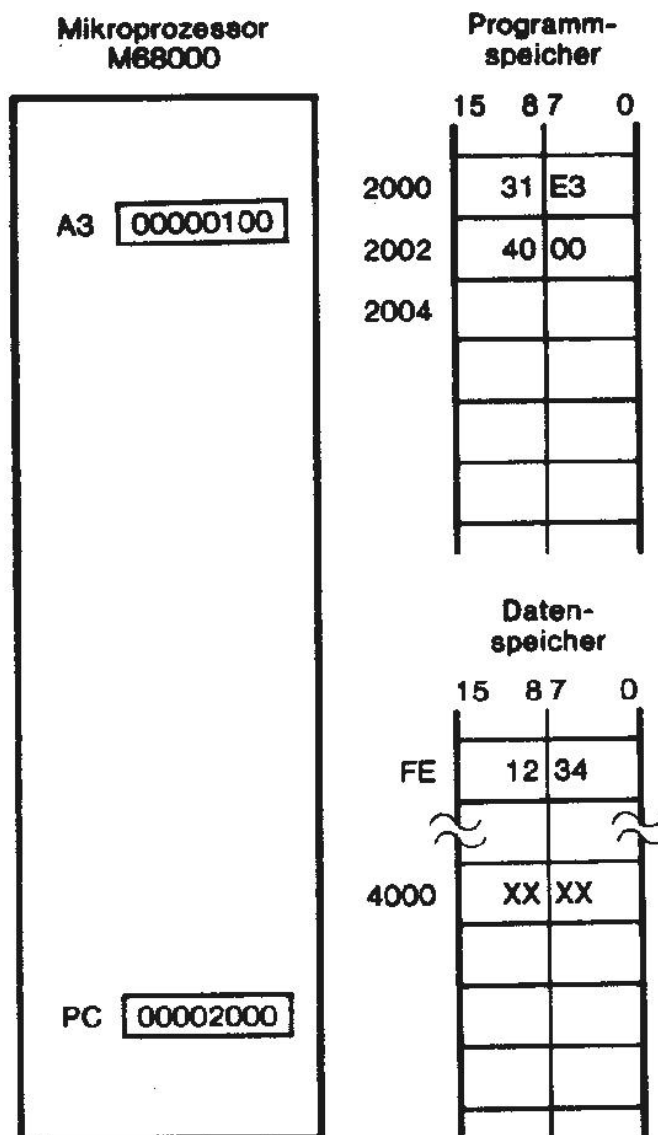
Nach Ausführung des Befehles



# Register-indirekte Adressierung (5)

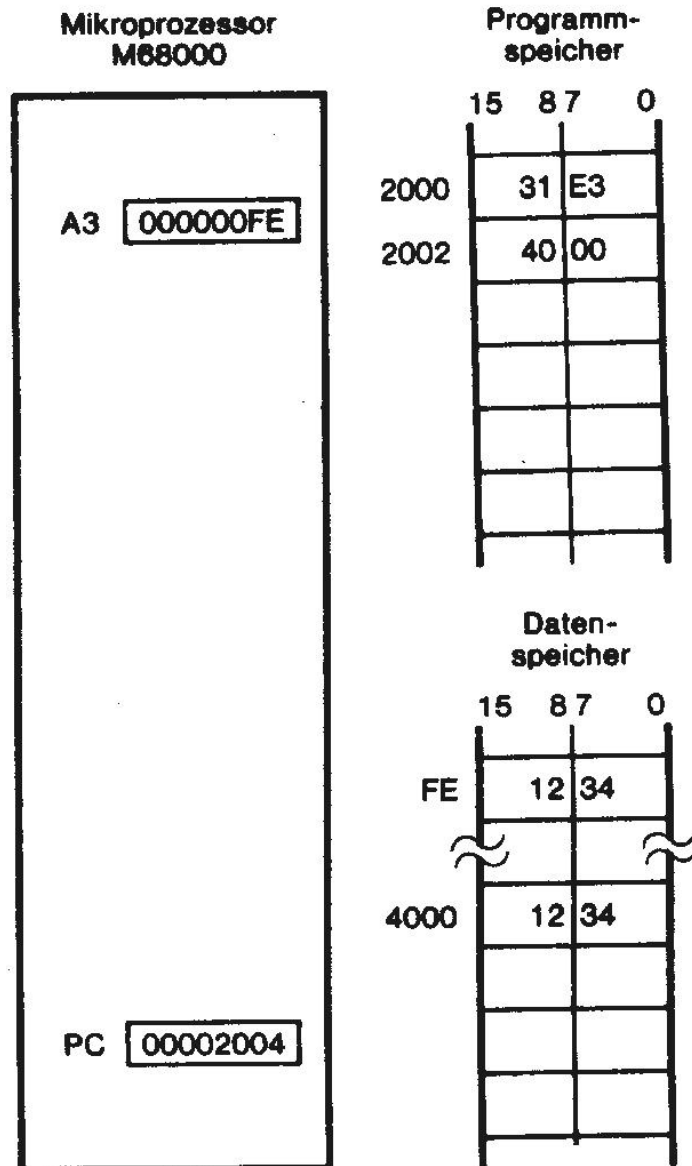
**Beispiel 3: mit Prädekrement** MOVE  $-(A3), \$4000$   
 Gleichbedeutend mit MOVE.W  $-(A3), \$4000$

## Vor Ausführung des Befehles



# Register-indirekte Adressierung (6)

## Nach Ausführung des Befehles



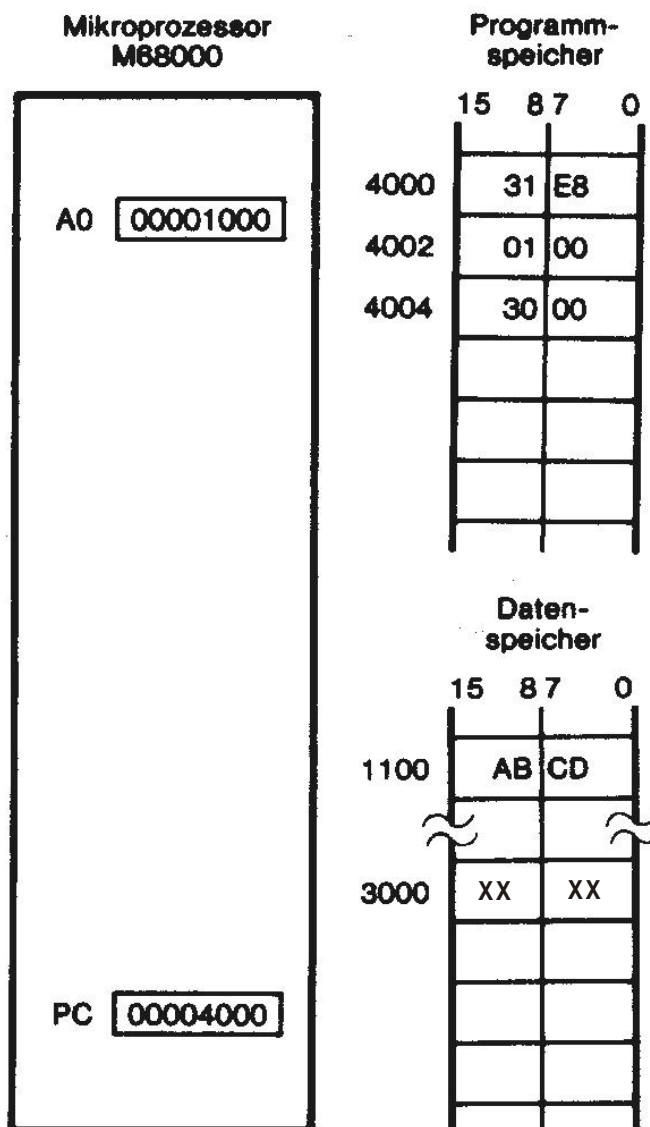


# Register-indirekte Adressierung (7)

## Beispiel 4: mit Adressdistanz

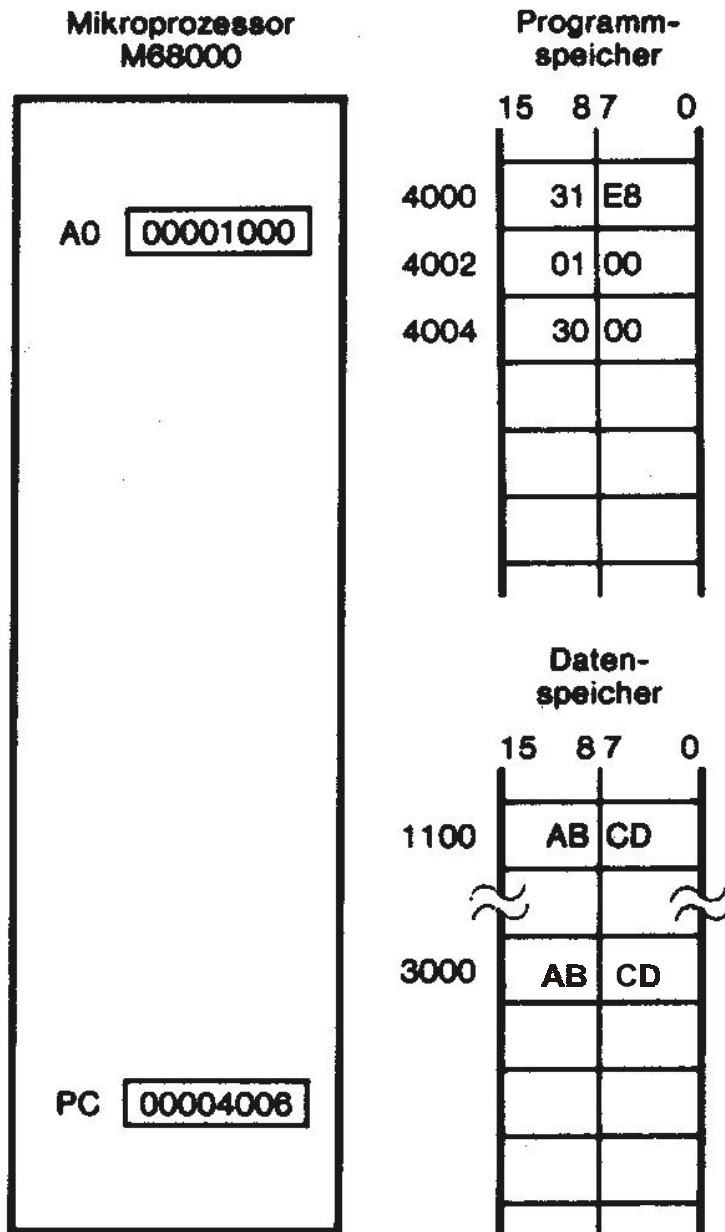
MOVE \$100(A0), \$3000  
 Gleichbedeutend mit MOVE.W 256(A0), \$3000

### Vor Ausführung des Befehles



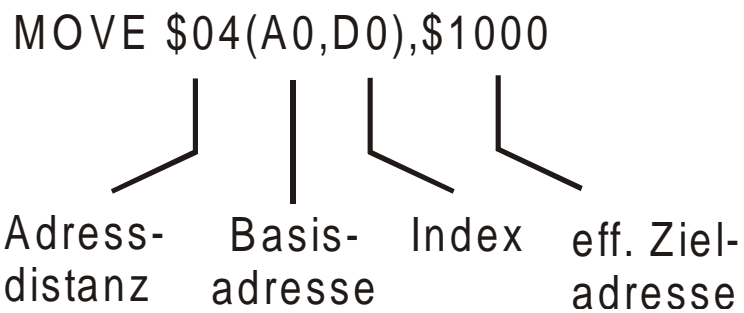
# Register-indirekte Adressierung (8)

## Nach Ausführung des Befehles



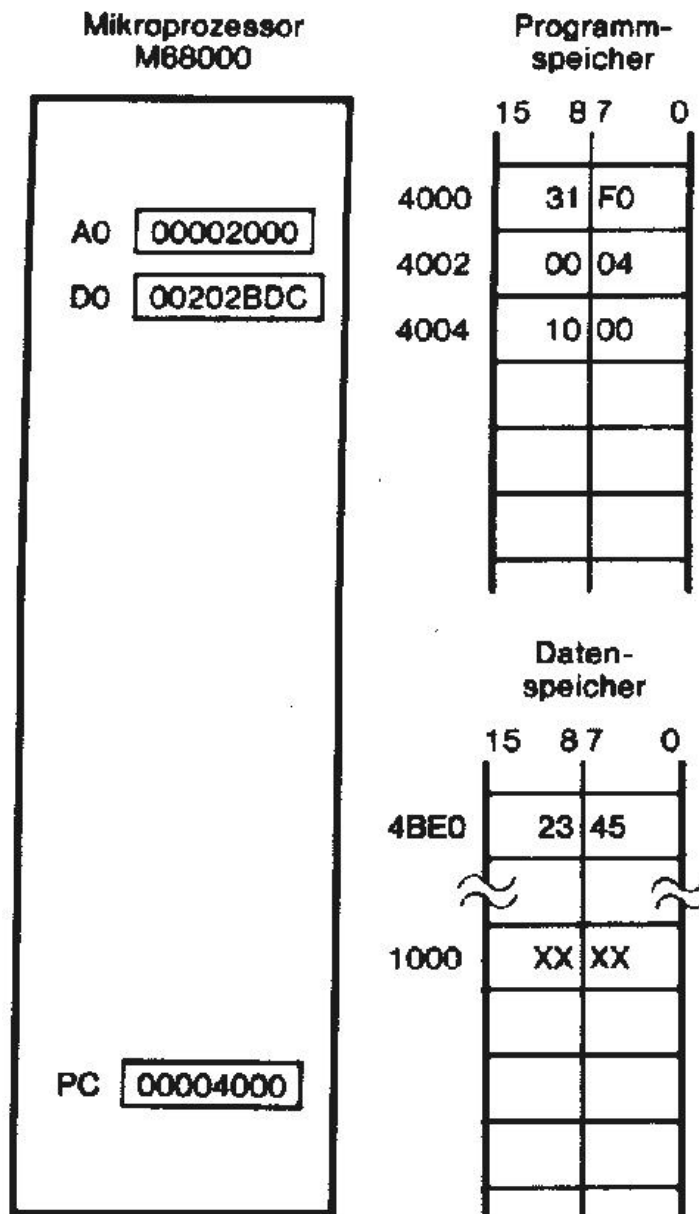
## Register-indirekte Adressierung (9)

### Beispiel 5: mit Index, mit oder ohne Adressdistanz



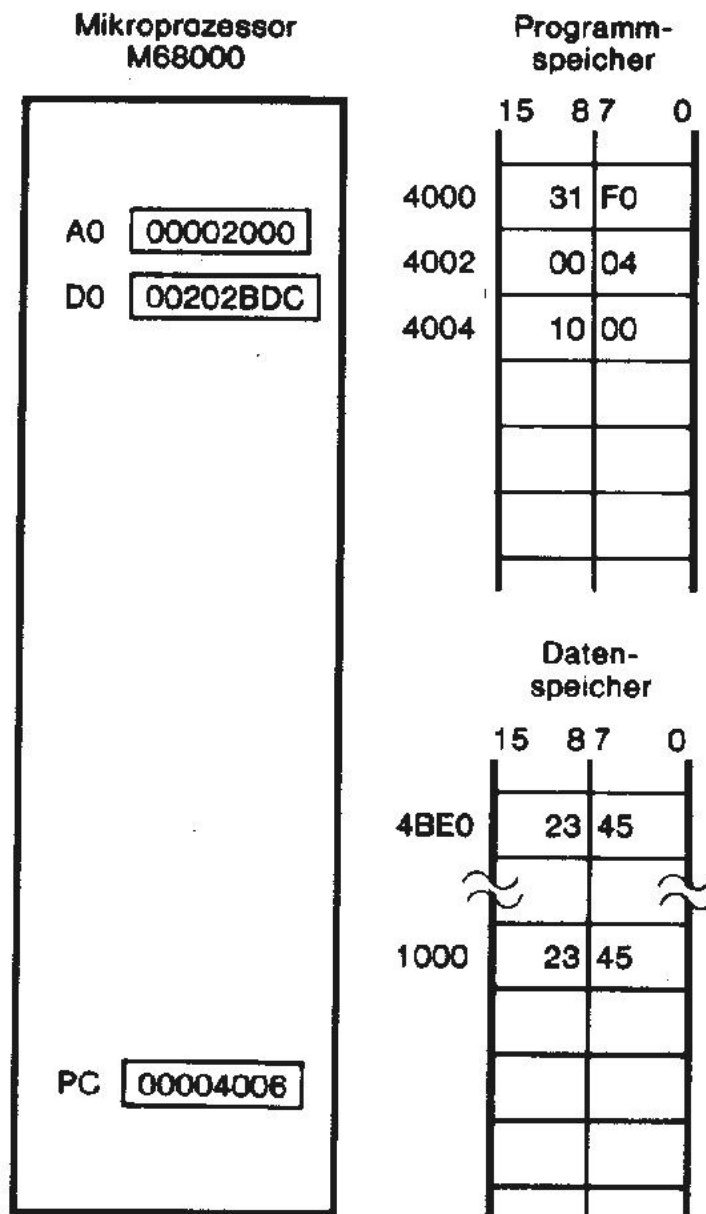
# Register-indirekte Adressierung (10)

## Vor Ausführung des Befehles



# Register-indirekte Adressierung (11)

## Nach Ausführung des Befehles



# Adressierung relativ zum Programmzähler

Der **Programmzähler (PC)** dient als **Basisregister**. Dazu kann eine positive oder negative Distanz angegeben werden sowie ein Indexregister. Dabei wird die Tatsache ausgenutzt, dass die zu verarbeitenden Daten häufig in der Nähe der Maschinenbefehle im Speicher stehen (modulare Programmierung!).

## Vorteil gegenüber absoluten Adressen

Im Mehrbenutzerbetrieb können Programme "reentrant" geschrieben werden: Derselbe Programmcode läuft korrekt an verschiedenen Speicheradressen, mit jeweils anderen Benutzerdaten.

## Vorteile gegenüber expliziter Nennung des Basisregisters

Die Verwaltung des Basisregisters bleibt dem Programmierer erspart. Die Befehle sind kürzer, da die Bitcodierung für das Basisregister entfällt.

	Programmierkurs II © Prof. Dr. W. Effelsberg	9. Assembler: Der Prozessor Motorola 68000	9 - 38
---	---	---	--------

# Adressierung relativ zum Programmzähler (1)

## 1. Beispiel

MOVE TEST(PC),D0

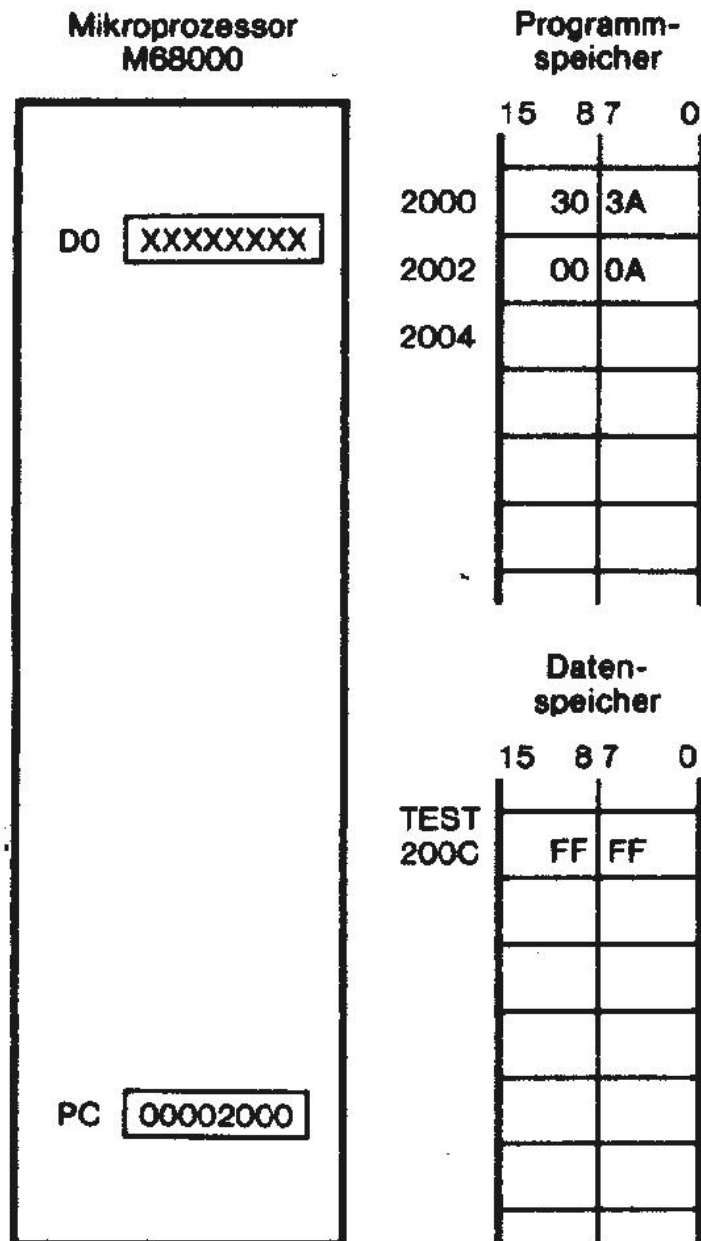
Gleichbedeutend mit:

MOVE.W TEST(PC),D0



# Adressierung relativ zum Programmzähler (2)

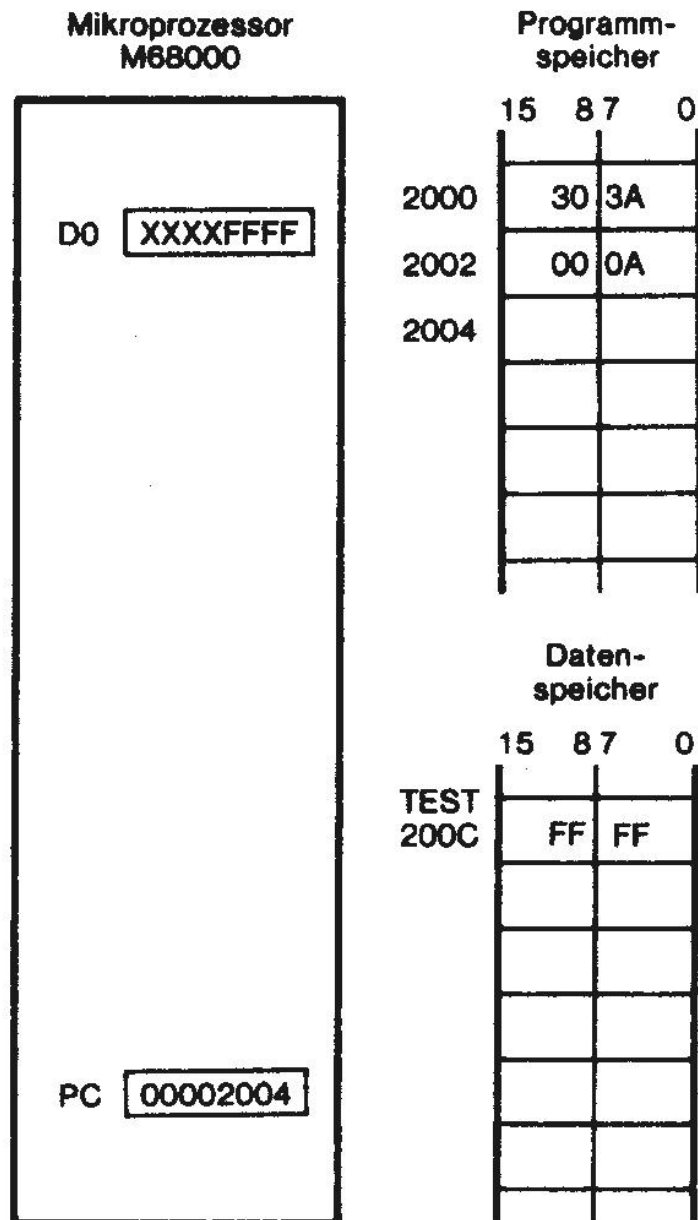
Vor Ausführung des Befehles





# Adressierung relativ zum Programmzähler (3)

Nach Ausführung des Befehles



# Programmbeispiele mit PC als Basisregister

## Programmbeispiel 1: Absolutes Programm

```
10          00000000          ORG.S $2000
20 002000 3038200C          MOVE TEST, D0
21 002004 4E71             NOP
22 002006 4E71             NOP
23 002008 4E71             NOP
24 00200A 4E71             NOP
25 00200C FFFF          TEST DC $FFFF
30                                     END
```

## Programmbeispiel 2: Verschiebbares Programm

```
10          ORG $2000
20 002000 303A000A          MOVE TEST(PC), D0
21 002004 4E71             NOP
22 002006 4E71             NOP
23 002008 4E71             NOP
24 00200A 4E71             NOP
25 00200C FFFF          TEST DC $FFFF
30                                     END
```

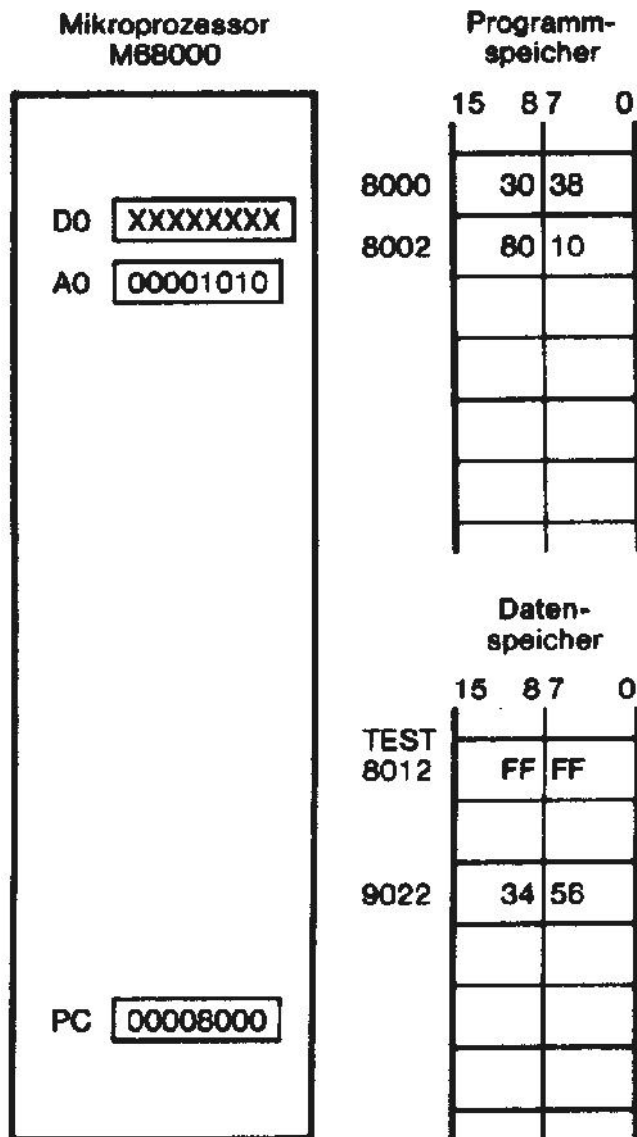
# Adressierung relativ zum Programmzähler (4)

**2. Beispiel:** Mit Indexregister und Distanz

MOVE     TEST(PC,A0),D0

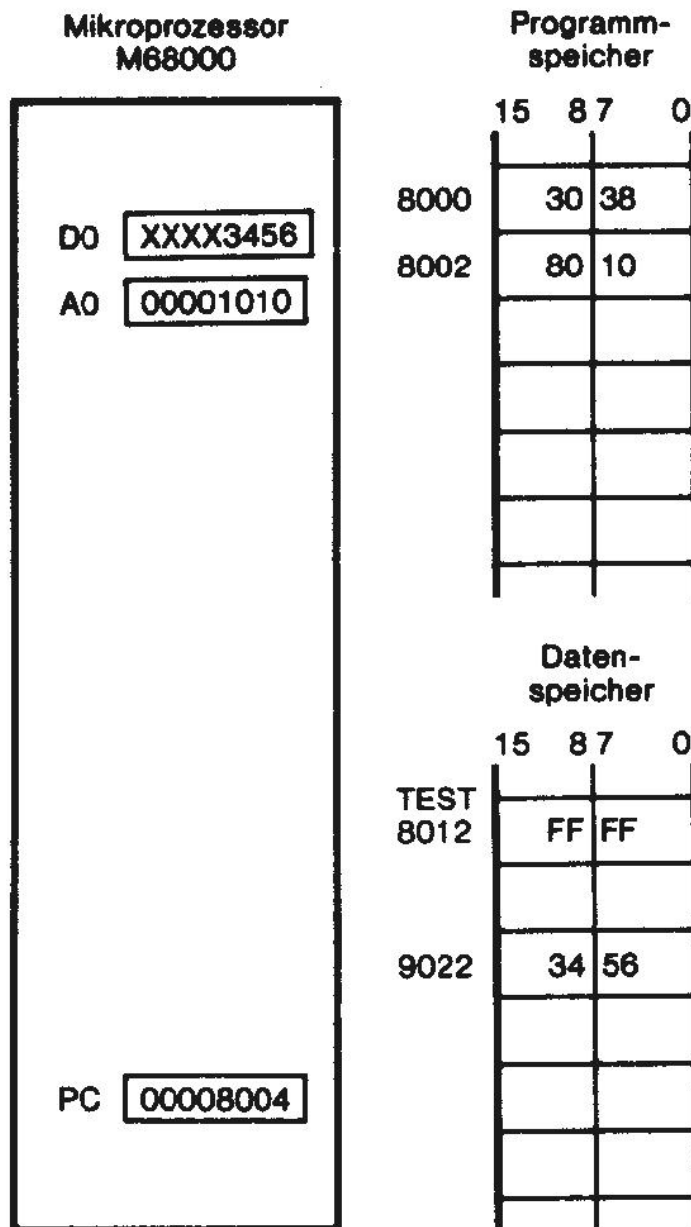
Gleichbedeutend mit     MOVE.W   TEST(PC,A0.W),D0

**Vor Ausführung des Befehles**



# Adressierung relativ zum Programmzähler (5)

Nach Ausführung des Befehles



## Zusammenfassung der Adressierungsarten

Adressierungsart	Mnemonic
Datenregister direkt	Dn
Adressregister direkt	An
Adressregister indirekt (ARI)	(An)
ARI mit Postinkrement	(An)+
ARI mit Prädekrement	-(An)
ARI mit Adressdistanz	d16(An)
ARI mit Adressdistanz und Index	d8(An,Rx)
Absolut kurz	\$XXXX
Absolut lang	\$XXXXXXXXXX
PC relativ mit Adressdistanz	d16(PC)
PC rel. Mit Adressdist. & Index	d8(PC,Rx)
Konstante, Statusregister	#, SR, CCR

