

# 10. Die Adressierungsarten des MSP 430

- 10.1 Übersicht über die Adressierungsarten
- 10.2 Register-Operanden
- 10.3 Indexregister mit Distanz
- 10.4 Symbolische Adresse (relativ zum PC)
- 10.5 Absolute Adresse
- 10.6 Indirekte Adresse
- 10.7 Indirekte Adressierung mit Postinkrement
- 10.8 Konstante als Operand
- 10.9 Beispiele für die Programmierung mit dem Stackpointer

## 10.1 Übersicht über die Adressierungsarten

Unterschiedliche Adressierungsarten erlauben es, den Speicher in unterschiedlicher Art und Weise mit Hardwareunterstützung zu lesen und zu schreiben. Einadress-Maschinen haben oft nur zwei explizite Adressierungsbefehle, LOAD und STORE, die den Speicher benutzen; alle anderen Operationen können nur auf die Register zugreifen. Zweiadress-Maschinen erlauben in der Regel sowohl Operanden in Registern als auch Operanden im Speicher.

Der Motorola 68000, eine weit verbreitete Zweiadress-Maschine, hat viele mächtige Adressierungsarten, aber nicht alle können in allen Befehlen für alle Operanden verwendet werden.

Beim MSP430, ebenfalls eine Zweiadress-Maschine, können *alle* Adressierungsarten in *allen* Befehlen für *alle* Operanden verwendet werden.

# Tabelle der Adressierungsarten

As/Ad	Adressierungsart	Syntax	Beschreibung
00/0	Register	Rn	der Operand steht im Register
01/1	Indexregister mit Distanz	X(Rn)	(Rn + X) zeigt auf den Operanden
01/1	symbolisch	ADDR	(PC + X) zeigt auf den Operanden (Adressierung relativ zum Befehlzähler)
01/1	absolut	&ADDR	Das Speicherwort nach dem Befehl enthält die absolute Adresse des Op.
10/-	indirekt	@Rn	Rn wird als Zeiger auf den Operanden benutzt
11/-	indirekt mit Postinkrement	@Rn+	Rn wird als Zeiger auf den Operanden benutzt, danach wird Rn inkrementiert
11/-	immediate	#N	Das Wort nach dem Befehl enthält die Konstante N. Adressierungsart indirekt mit Postinkrement @PC+ wird benutzt

## 10.2 Register-Operanden

### Assembler-Code

```
MOV R10, R11
```

### Beschreibung

Kopiere den Inhalt von R10 nach R11; R10 bleibt unverändert

	Vorher	Nachher
R10	0A023h	0A023h
R11	0FA15h	0A023h
PC	PC <sub>old</sub>	PC <sub>old</sub> +2

## 10.3 Indexregister mit Distanz (1)

### Assembler-Code

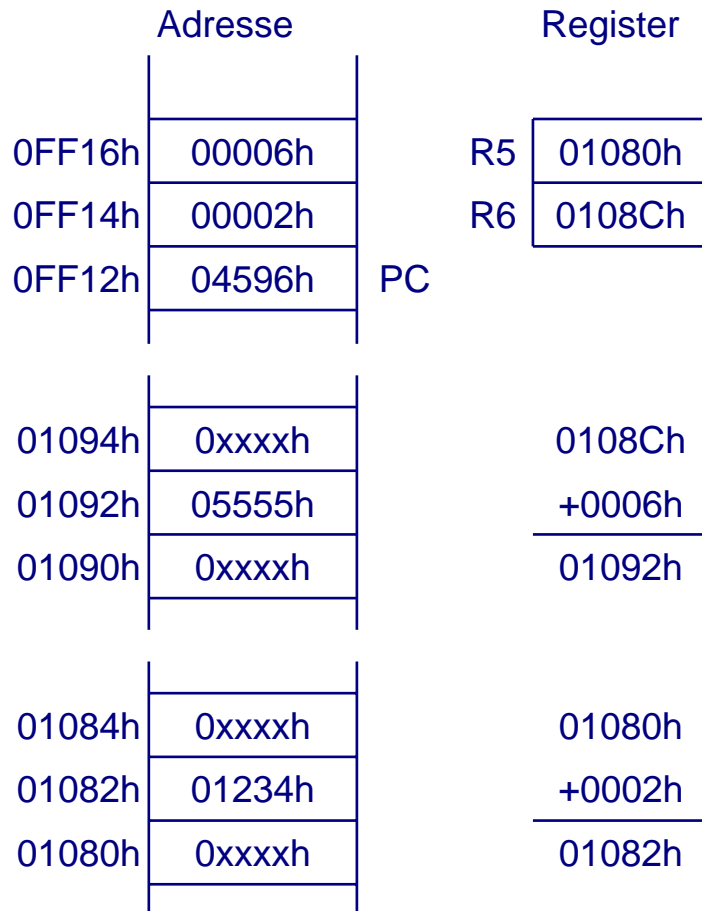
```
MOV 2(R5), 6(R6)
```

### Beschreibung

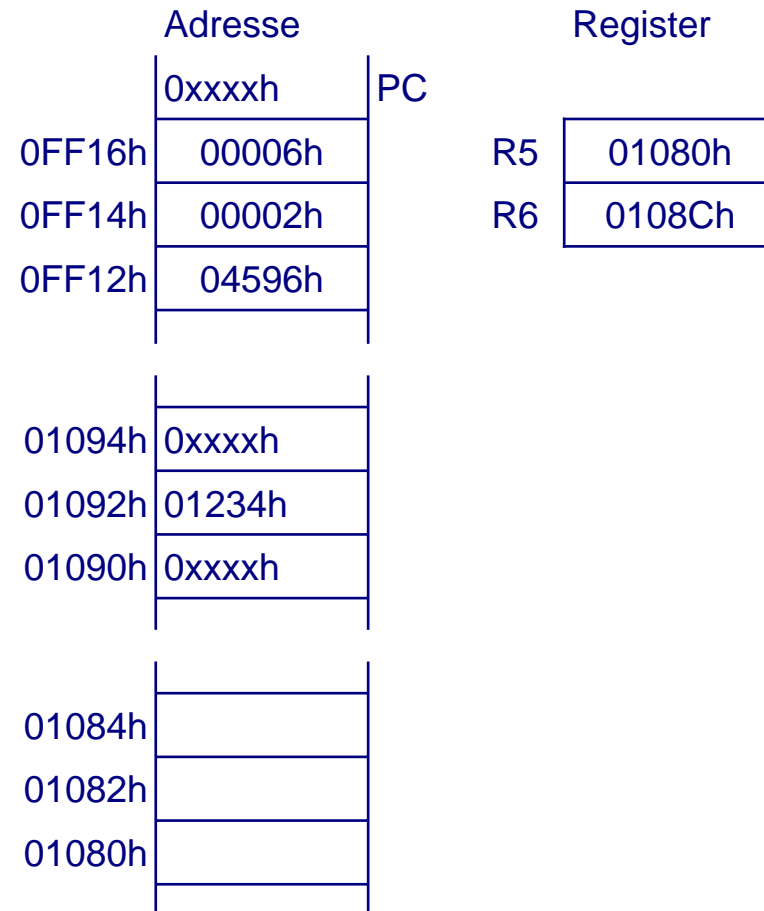
Kopiere den Inhalt von der Quelladresse (Inhalt von R5+2) zur Zieladresse (Inhalt von R6+6). Die Quell- und Zielregister (R5 und R6) werden nicht verändert.

# Indexregister mit Distanz: Beispiel

Vorher:



Nachher:



## 10.4 Symbolische Adresse (relativ zum PC)

### Assembler-Code

```
MOV EDE, TONI
```

entspricht:

```
MOV x(PC), y(PC)
```

### Beschreibung

Kopiere den Inhalt von der Quelladresse EDE (Inhalt von PC+x) an die Zieladresse TONI (Inhalt von PC+y). Die Speicherwörter nach der Instruktion enthalten die gewünschte *Distanz* (Offset, relative Adresse) zwischen dem PC und dem Quell- bzw. Zieloperanden. Der Assembler berechnet beim Assemblieren die Offsets x und y automatisch und fügt sie in den Befehl ein.

# Symbolische Adresse: Beispiel

**Vorher:**

	Adresse	
0FF16h	011FEh	
0FF14h	0F102h	
0FF12h	04090h	PC
0F018h	0xxxxh	
0F016h	0A123h	
0F014h	0xxxxh	
01116h	0xxxxh	
01114h	01234h	
01112h	0xxxxh	

Register

$$\begin{array}{r} 0FF14h \\ +0F102h \\ \hline 0F016h \end{array}$$

$$\begin{array}{r} 0FF16h \\ +011FEh \\ \hline 01114h \end{array}$$

**Nachher:**

	Adresse		Register
	0xxxxh		PC
0FF16h	011FEh		
0FF14h	0F102h		
0FF12h	04090h		
0F018h	0xxxxh		
0F016h	0A123h		
0F014h	0xxxxh		
01116h	0xxxxh		
01114h	0A123h		
01112h	0xxxxh		



## 10.5 Absolute Adresse

### Assembler-Code

```
MOV &EDE, &TONI
```

entspricht:

```
MOV x(0), y(0)
```

### Beschreibung

Kopiere den Inhalt von der Quelladresse EDE zur Zieladresse TONI. Die Speicherwörter nach der Instruktion enthalten die *absolute* Adresse des Quell- bzw. Zieloperanden.

# Absolute Adresse: Beispiel

Vorher:

	Adresse	Register
0FF16h	01114h	PC
0FF14h	0F016h	
0FF12h	04292h	
0F018h	0xxxxh	
0F016h	0A123h	
0F014h	0xxxxh	
01116h	0xxxxh	
01114h	01234h	
01112h	0xxxxh	

Nachher:

	Adresse	Register
	0xxxxh	PC
0FF16h	01114h	
0FF14h	0F016h	
0FF12h	04292h	
0F018h	0xxxxh	
0F016h	0A123h	
0F014h	0xxxxh	
01116h	0xxxxh	
01114h	0A123h	
01112h	0xxxxh	

## 10.6 Indirekte Adresse

### Assembler Code

```
MOV @R10, 0(R11)
```

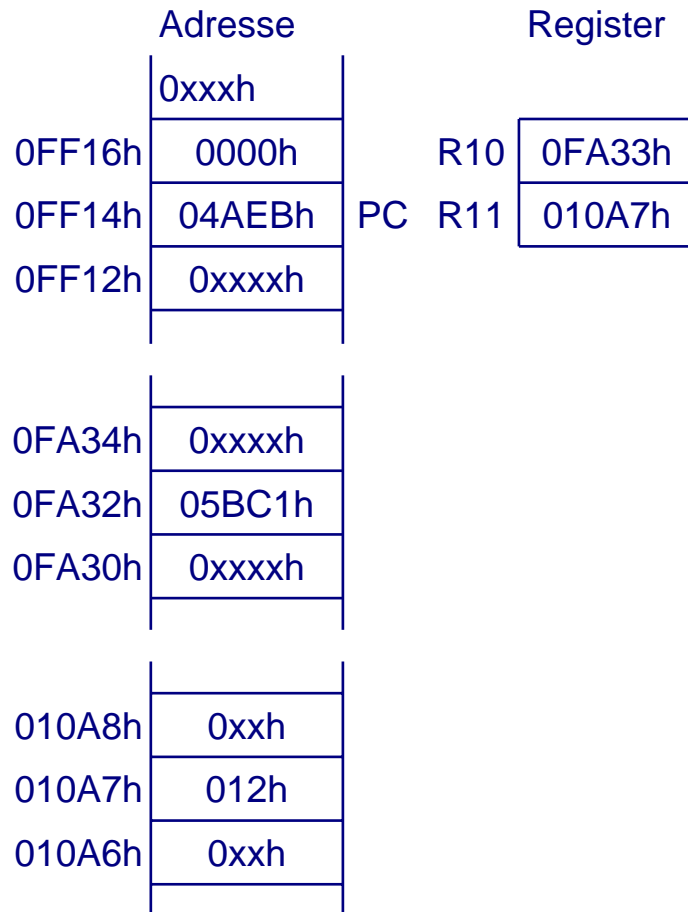
### Beschreibung

Kopiere den Inhalt von der Quelladresse (Inhalt von R10) zur Zieladresse (Inhalt von R11). Die Register werden nicht verändert.

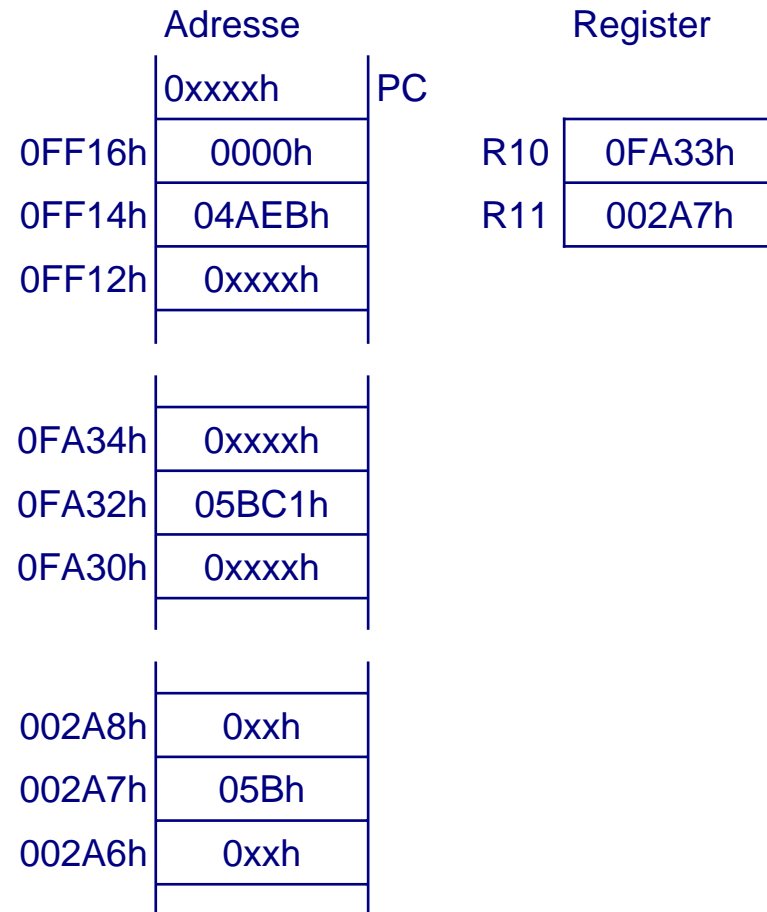
.

# Indirekte Adresse: Beispiel

**Vorher:**



**Nachher:**



## Register (indirekt)

@Rn            Register n enthält die *Adresse* des Operanden

**Beispiel:**    `mov #FFFE, R15 // lädt Register 15 mit Adresse FFFE`  
`mov #2100, @R15 // schreibt den Wert 2100 an die`  
`// Speicheradresse FFFE`

## Register (indexiert)

`offset (Rn)`            die Adresse des Operanden ist die Summe aus dem Inhalt von Rn und dem Offset

**Beispiel:**    `mov #FFFF, R15            // lädt Reg. 15 mit Adresse`  
`// (bzw. Zahl) FFFF`  
`mov #0123, -1(R15)       // schreibt 123 ab Adresse FFFE`

## 10.7 Indirekte Adressierung mit Postinkrement

### Assembler-Code

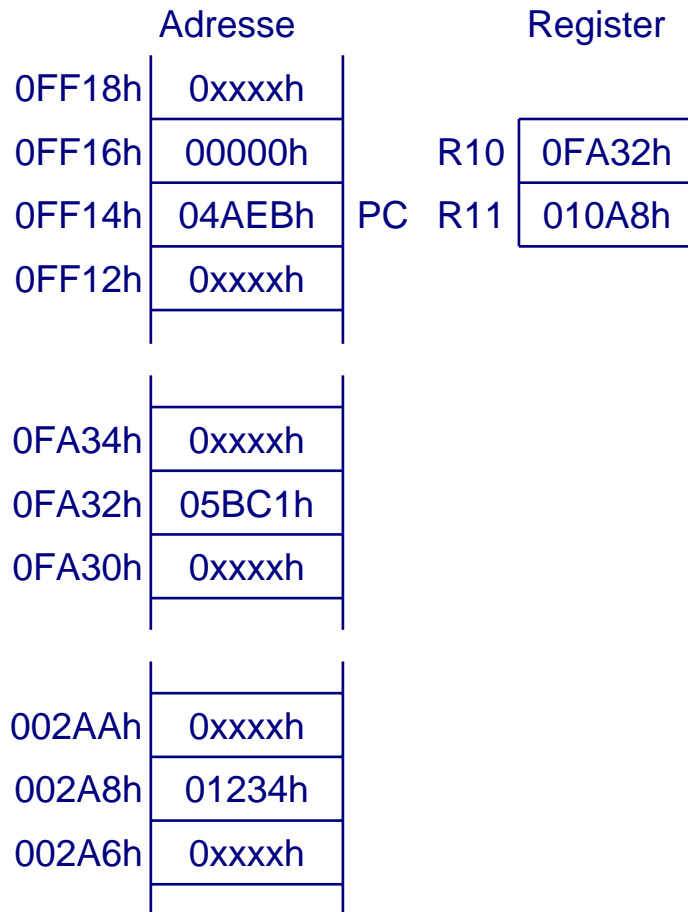
```
MOV @R10+, 0(R11)
```

### Beschreibung

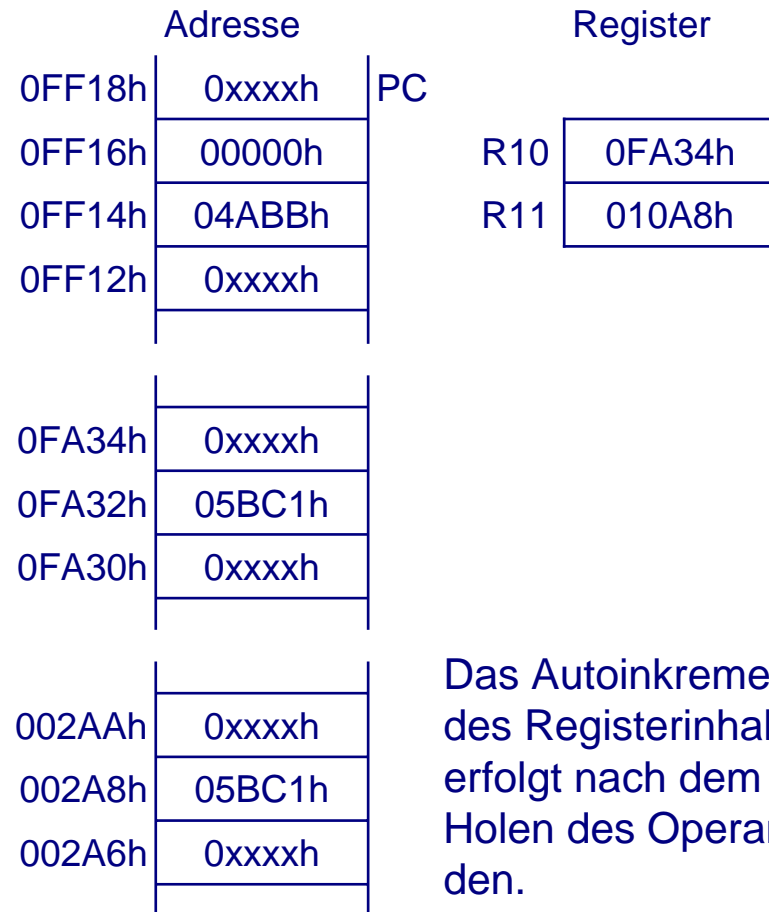
Kopiere den Inhalt von der Quelladresse (Inhalt von R10) zur Zieladresse (Inhalt von R11). Register R10 wird bei einer Byte-Operation um 1, bei einer Wort-Operation um 2 inkrementiert. Das Inkrementieren erfolgt nach der Operation; das Register (hier R10) zeigt auf die nächste Adresse im Speicher, ohne einen zusätzlichen Befehl zu erfordern. Dies ist zum Beispiel bei Verarbeitung von Arrays sehr nützlich.

# Indirekte Adressierung mit Postinkrement: Beispiel 1

**Vorher:**



**Nachher:**



Das Autoinkrement des Registerinhalts erfolgt nach dem Holen des Operanden.

## Indirekte Adressierung mit Postinkrement: Beispiel 2

```
mov #1234, &0200    // schreibt 1234 ab Speicherzelle 200
mov #0200, R7        // lädt Register 7 mit Wert (Adresse) 200
mov @R7+, R1         // lädt Register R1 mit 1234
                    // danach steht 202 in R7
```

Was bewirkt folgender Ausdruck? `add @R15+, -2(R15)`



## 10.8 Konstante als Operand

### Assembler-Code

```
MOV #45h, TONI
```

### Beschreibung

Kopiere die Konstante 45h, die sich im Wort nach der Instruktion befindet, an die Zieladresse TONI.

# Konstante als Operand: Beispiel

**Vorher:**

	Adresse	
0FF16h	01192h	
0FF14h	00045h	PC
0FF12h	040B0h	
010AAh	0xxxxh	
010A8h	01234h	
010A6h	0xxxxh	

Register

0FF16h  
+01192h  

---

010A8h

**Nachher:**

	Adresse	
0FF18h	0xxxxh	PC
0FF16h	01192hh	
0FF14h	00045h	
0FF12h	040B0h	
010AAh	0xxxxh	
010A8h	00045h	
010A6h	0xxxxh	

Register

## Die “Konstantenregister” R2 und R3

Register	Bits für Adressierungsart (As)	Konstante	Bemerkung
R2	00	-----	Registermodus
R2	01	(0)	absolute Adressierung
R2	10	0004h	+4, bitweise
R2	11	0008h	+8, bitweise
R3	00	00000h	0, wortweise
R3	01	00001h	+1
R3	10	00002h	+2, bitweise
R3	11	0FFFFh	-1, wortweise

### Vorteile dieser Art der Konstantengenerierung:

- keine speziellen Befehle erforderlich
- geringere Programmspeichieranforderungen, da kein zusätzliches Word für die 6 meistbenutzten Konstanten erforderlich ist.
- geringere Ausführungszeit für die Befehle, da kein separater Speicherzugriff zum Holen der Konstanten nötig ist.

# 10.9 Beispiele für die Programmierung mit dem SP

**Merke: Der Stackpointer ist R1**

```

MOV   R1, R4      ;SP -> R4
MOV   @R1, R5     ;D3 (TOS) ->R5
MOV   2 (R1), R6  ;D2 ->R5
MOV   R7, 0 (R1)  ;überschreibe TOS mit R7
MOV   R8, 4 (R1)  ;verändere D1
PUSH  R12         ;R12→0xxxh-6, SP an dieselbe Adresse
POP   R12        ;0xxxh-6 → R12, SP zeigt auf 0xxxh-4
MOV   @R1+, R5   ;D3 → R5 (POP vom Stack), bewirkt dasselbe wie POP
    
```

