Kapitel 7

Benutzung der JMP- und CALL-Befehle

(für MASM-, ASM86- und ASM286-Programmierer)

7.1 Unbedingte Sprünge im Bereich von -128 bis +127 Byte

Die 8086/80186/80286-CPUs kennen zwei unterschiedliche JMP-Befehle, um Labels innerhalb des augenblicklichen Code-Segments zu erreichen. Beide bestehen aus einem Einzel-Byte-Opcode, gefolgt von einem Displacement-Feld. Dieses Feld spezifiziert den Wert, der zum Instruction-Pointer IP addiert wird.

Der eine JMP-Befehl ist durch **zwei** Bytes im Displacement-Feld gekennzeichnet. Mit ihm sind sogenannte **NEAR-Labels**, also Sprungziele im Bereich von ±32K Bytes ab der augenblicklichen Position zu erreichen.

Der andere JMP-Befehl ist durch **ein** Byte im Displacement-Feld gekennzeichnet, das von den CPUs intern vorzeichenbehaftet auf zwei Bytes erweitert wird (sign extension). Mit diesem Befehl sind sogenannte SHORT-Labels, also Sprungziele im Bereich von -128 bis +127 Bytes ab der augenblicklichen Position zu erreichen. Der **SHORT JMP** ist gegenüber dem allgemeinen **NEAR JMP** optimiert, da er ein Byte aus dem Befehlscode entfernt.

Beispiel 1:

In der angegebenen Quelldatei ist der Label RUECK_WAERTS **vor** dem Befehl `JMP RUECK_WAERTS` codiert. Er liegt weniger als 129 Bytes vom Sprungziel entfernt.

```
LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_60
                             2
                             3     ASSUME CS:CODE_PROG
                             4
----                         5     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6
0000                         7     START:          ;
                             8                     ;
0000 8BD0                    9     RUECK_WAERTS:   MOV     DX,AX
                            10                     ;
                            11                     ;
                            12                     ;
0002 EBFC                   13                     JMP     RUECK_WAERTS
                            14                     ;
----                        15     CODE_PROG ENDS
                            16
                            17             END     START
```

Für diesen **Rückwärtssprung** optimiert der Assembler den Code des JMP-Befehls und generiert automatisch einen Zwei-Byte **SHORT JMP**.

Dies ist möglich, da der Assembler den Abstand zwischen dem Sprungziel und der Position des JMP-Befehls kennt.

Beispiel 2:

In der angegebenen Quelldatei ist der Label RUECK_WAERTS **vor** dem Befehl JMP RUECK_WAERTS codiert. Er liegt außerhalb des 128-Byte-Bereiches.

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_61
                             2
                             3     ASSUME CS:CODE_PROG
                             4
----                         5     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6
0000                         7     START:          ;
                             8                     ;
0000 8BD0                    9     RUECK_WAERTS:   MOV     DX,AX
                            10                     ;
0002 (129                   11                     DW      129 DUP (?)         ;Diese Definition dient nur
     ????
     )
                            12                                                 ;Demonstrationszwecken
                            13                     ;
0104 E9F9FE                 14                     JMP     RUECK_WAERTS
                            15                     ;
----                        16     CODE_PROG ENDS
                            17
                            18             END     START
```

Für diesen **Rücksprungbefehl** generiert der Assembler einen Drei-Byte **NEAR-JMP**.

Beispiel 3:

In der angegebenen Quelldatei ist der Label VOR_WAERTS **nach** dem Befehl JMP VOR_WAERTS codiert.

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_62
                             2
                             3     ASSUME CS:CODE_PROG
                             4
----                         5     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6
0000                         7     START:          ;
                             8                     ;
0000 EB0190                  9     SPRUNG:         JMP     VOR_WAERTS
                            10                     ;
                            11                     ;
                            12                     ;
0003 8BD0                   13     VOR_WAERTS:     MOV     DX,AX
                            14                     ;
----                        15     CODE_PROG ENDS
                            16
                            17             END     START
```

Der Assembler kennt in diesem Fall den Abstand zwischen der Position des JMP-Befehls und des Sprungziels **nicht**.

Um zu garantieren, daß das Sprungziel erreicht wird, reserviert der Assembler Platz für den längeren **NEAR-JMP**-Befehl. D.h. für den Befehl `JMP VOR_WAERTS` generiert der Assembler einen Drei-Byte **NEAR-JMP**; und dies sogar, wenn der Label VOR_WAERTS im Bereich eines **SHORT-JMP** liegt.

7.1.1 Der SHORT-Operator

Um einen **Vorwärtssprung** zu optimieren, ist dem Assembler mitzuteilen, daß der Code zwischen dem Vorwärtssprung und seinem Ziel-Label kleiner als 128 Bytes ist. Dies ist möglich, wenn im JMP-Befehl der sogenannte **SHORT-Operator** benützt wird.

Beispiel 4:

```
LOC  OBJ                  LINE      SOURCE
                             1              NAME    BEISPIEL_63
                             2
                             3      ASSUME CS:CODE_PROG
                             4
----                         5      CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6
0000                         7      START:          ;
                             8                      ;
0000 EB00                    9      SPRUNG:         JMP     SHORT VOR_WAERTS
                            10                      ;
                            11                      ;
                            12                      ;
0002 8BD0                   13      VOR_WAERTS:     MOV     DX,AX
                            14                      ;
----                        15      CODE_PROG ENDS
                            16
17              END     START
```

In der angegebenen Quelldatei ist im JMP-Befehl der SHORT-Operator verwendet.

Wenn der Assembler in einem JMP-Befehl den SHORT-Operator sieht, versucht er immer den Zwei-Byte SHORT-JMP zu generieren. Findet er den Ziel-Label außerhalb des 127-Byte-Bereiches, wird er eine Fehlermeldung ausgeben. Darin teilt er mit, daß er den Label nicht finden kann. Der Anwenderprogrammierer wird daraufhin in der Quelldatei den SHORT-Operator entfernen.

Der SHORT-Operator ist also immer dann anzuwenden, wenn ein JMP-Befehl optimiert werden soll. Dies setzt allerdings voraus, daß das Sprungsziel im SHORT-JMP-Bereich liegt.

7.2 Bedingte Sprünge außerhalb des Bereiches von -128 bis +127 Bytes

Die 8086/80186/80286 bedingten Sprungbefehle (JC, JZ, JBE usw.) sind Zwei-Byte-Befehle. Das erste Byte entspricht dem Opcode des Befehls, während das zweite Byte eine vorzeichenbehaftete Zahl kennzeichnet. Dieses Offset-Byte wird zum Instruction-Pointer IP addiert, wenn der Sprung ausgeführt wird.

Da bedingte Sprungbefehle in einem Assembler-Programm oft vorkommen, reduzieren diese Zwei-Byte-Befehle zwar die Speicherbelegung, sie sind aber auf einen Bereich von -128 bis +127 Bytes begrenzt. D.h. das Sprungziel muß innerhalb von -128 bis +127 Bytes ab der Position des Sprungbefehls liegen.

Gelegentlich kommt es vor, daß bedingte Sprungbefehle ausgeführt werden müssen, deren Ziel-Label außerhalb des Bereichs von -128 bis +127 Bytes liegen.

Um dies zu realisieren, stehen zwei Techniken zur Verfügung:

- Die **gerichtete-Sprung-Methode** (vektored-Jump) und
- die **Jump-around-Jump-Methode**

7.2.1 Die gerichtete-Sprung-Methode

Bei dieser Methode wird zunächst der bedingte Sprung zu einem Zwischen-Label im -128 bis +127 Byte-Bereich durchgeführt ①. Dieser Zwischen-Label kennzeichnet einen JMP-Befehl, mit dessen Hilfe dann die gewünschte Position außerhalb des -128 bis +127 Byte-Bereiches zu erreichen ist ②.

Beispiel:

```
LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_64
                             2
                             3     ASSUME CS:CODE_PROG
                             4
----                         5     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6
                             7     EXTRN NACH_IRGENDWO:NEAR
                             8
0000                         9     START:          ;
                            10                     ;          ②
0000                        11     GEWUENSCHTES_ZIEL:
0000 8BD0                   12                     MOV     DX,AX
0002 (500                   13                     DW      500 DUP (?)     ;Diese Definition dient nur
     ????
     )
                            14                                             ;Demonstrationszwecken
                            15                     ;
                            16                     ;
03EA E90000         E       17                     JMP     NACH_IRGENDWO
                            18                     ;
                            19                     ;
03ED E910FC                 20     ZWISCHEN_ZIEL:  JMP     GEWUENSCHTES_ZIEL ②
03F0 (100                   21                     DB      100 DUP (?)     ;Diese Definition dient nur
     ??
     )
                            22                                             ;Demonstrationszwecken
                            23                     ;                         ①
                            24                     ;
0454 7297                   25                     JC      ZWISCHEN_ZIEL
                            26                     ;
----                        27     CODE_PROG ENDS
                            28
                            29             END     START
```

Die gerichtete-Sprung-Methode setzt voraus, daß **gefahrlos** im -128 bis +127 Byte-Bereich des unbedingten Sprungbefehls ein JMP-Befehl codiert werden kann.

So ist im Beispiel vor dem Zwischen-Label ZWISCHEN_ZIEL ein anderer JMP-Befehl (`JMP NACH_IRGENDWO`) codiert. Die gleiche Schutzfunktion würde an dieser Stelle auch ein RET-Befehl erfüllen.

7.2.2 Die Jump-around-Jump-Methode

Bei dieser Methode wird ebenfalls ein JMP-Befehl benützt, um den Bereich eines bedingten Sprungbefehls zu erweitern. Jedoch folgt in diesem Fall der JMP-Befehl unmittelbar dem bedingten Sprungbefehl ① und dreht damit den Sinn des bedingten Sprungbefehls um.

Beispiel:

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_65
                             2
                             3     ASSUME CS:CODE_PROG
                             4
----                         5     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6
0000                         7     START:          ;
                             8                     ;
0000                         9     GEWUENSCHTES_ZIEL:
0000 8BD0                   10                     MOV     DX,AX
0002 (500                   11                     DW      500 DUP (?)     ;Diese Definition dient nur
     ????
     )
                            12                                             ;Demonstrationszwecken
                            13                     ;
                            14                     ;
03EA 7503                   15                     JNZ     WEITER
03EC E911FC                 16                     JMP     GEWUENSCHTES_ZIEL
03EF 8BCA                   17     WEITER:         MOV     CX,DX
                            18                     ;
                            19                     ;
----                        20     CODE_PROG ENDS
                            21
                            22             END     START
```

Im angegebenen Beispiel ist der Befehl JZ GEWUENSCHTES_ZIEL nachgebildet, wobei der Label GEWUENSCHTES_ZIEL außerhalb des -128 bis +127 Byte-Bereiches liegt.

Solange die Originalbedingung (Ergebnis einer Operation = 0) erfüllt ist, wird mit Hilfe des Befehls JMP GEWUENSCHTES_ZIEL der Code wiederholt. Ist die Originalbedingung nicht erfüllt, wird der JMP-Befehl übersprungen (Jump-around-Jump). Die Fortsetzung des Programms erfolgt beim Label WEITER. Die Jump-around-Jump-Methode hat den Vorteil, daß eine geschützte Position für den JMP-Befehl nicht vorhanden sein muß.

7.3 JMP- und CALL-Operanden (direkt)

JMP- und CALL-Befehle können **direkt** und **indirekt** ausgeführt werden. Dabei ist der Operand eines **direkten** JMP- oder CALL-Befehls durch einen Label identifiziert.

Format:

```
[Name:]    JMP oder CALL    Label    [;Kommentar]
```

- **Name** ist ein metasprachlicher Ausdruck, der den Offset des JMP- oder CALL-Befehls im augenblicklichen Code-Segment kennzeichnet. Die eckigen Klammern geben an, daß der Name frei gewährt werden kann.
- **Label** ist der metasprachliche Ausdruck für das Sprungziel eines JMP-Befehls oder das Verzweigungsziel eines CALL-Befehls. Das Ziel wird also direkt im Befehl angegeben.

Beispiel:

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_66
                             2
                             3     ASSUME CS:CODE_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6                    DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
----                        11     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            12
0000 B8----          R      13     START:          MOV     AX,STACK_PROG          ;SS Register initialisieren
0003 8ED0                   14                     MOV     SS,AX
0005 BC0A00          R      15                     MOV     SP,OFFSET TOP_STACK    ;SP Register initialisieren
                            16                             ;
                            17                             ;
0008 EB0490                 18     INTRA_1:        JMP     ZIEL                   ;Sprung nach CS:ZIEL
                            19                                                    ;Intrasegment direkt
000B E80000                 20     INTRA_2:        CALL    ZIEL                   ;Verzweige nach CS:ZIEL
                            21                                                    ;Intrasegment direkt
                            22                             ;
                            23                             ;
000E 8AC1                   24     ZIEL:           MOV     AL,CL
                            25                             ;
                            26                             ;
                            27
----                        28     CODE_PROG ENDS
                            29
                            30             END     START
```

Die im Programmlisting angegebenen Befehle JMP ZIEL und CALL ZIEL bewirken, daß das Programm beim Label CS:ZIEL fortgesetzt wird. In beiden Fällen wird nur der Instruction-Pointer IP umgeladen (Instruction-Pointer relativ). D.h. die direkten Programmverzweigungen erfolgen im augenblicklichen Codesegment.

7.4 JMP- und CALL-Operanden (indirekt)

Der Operand eines **indirekten** JMP- oder CALL-Befehls ist durch einen WORD- oder DWORD-Pointer zum Sprungziel bzw. Verzweigungsziel identifiziert. Dabei besteht ein WORD-Pointer aus einem 16-Bit-Offset und ein DWORD-Pointer aus einem 16-Bit-Offset, gefolgt von einem 16-Bit-Segment-Basiswert. Um diese Pointer zu definieren, werden die DW- und DD-Anweisungen benützt.

Format:

```
[Name:]    JMP oder CALL    Adreß-Ausdruck    [;Kommentar]
```

Im angegebenen Format kennzeichnet **Adreß-Ausdruck** ein Register, eine WORD-Variable, eine DWORD-Variable oder den Ausdruck [BX].

Ein **Register** enthält den Offset des Sprung- bzw. Verzweigungsziels im augenblicklichen Codesegment.

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_67
                             2
                             3     ASSUME CS:CODE_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6                    DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
----                        11     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            12
0000 B8----          R      13     START:          MOV     AX,STACK_PROG          ;SS Register initialisieren
0003 8ED0                   14                     MOV     SS,AX
0005 BC0A00          R      15                     MOV     SP,OFFSET TOP_STACK    ;SP Register initialisieren
                            16                             ;
0008 BB100090        R      17                     MOV     BX,OFFSET ZIEL
                            18                             ;
                            19                             ;
000C FFE3                   20     INTRA_1:        JMP     BX                     ;Sprung nach CS:(BX)
                            21                                                    ;Intrasegment indirekt
                            22                             ;
000E FFD3                   23     INTRA_2:        CALL    BX                     ;Verzweige nach CS:(BX)
                            24                                                    ;Intrasegment indirekt
                            25                             ;
0010 8AC1                   26     ZIEL:           MOV     AL,CL
                            27                             ;
                            28                             ;
                            29
----                        30     CODE_PROG ENDS
                            31
                            32             END     START
```

Im angegebenen Beispiel enthält das Register BX den Offset des Labels ZIEL. Die Befehle `JMP BX` bzw. `CALL BX` bewirken, daß das Programm beim Label CS:(BX) = CS:ZIEL fortgesetzt wird. In beiden Fällen erfolgen die indirekten Programmverzweigungen im augenblicklichen Codesegment (Intrasegment indirekt).

Eine **WORD-Variable** enthält den Offset des Sprung- bzw. Verzweigungsziels im augenblicklichen Codesegment.

Beispiel:

```
LOC  OBJ                  LINE     SOURCE

                             1           NAME    BEISPIEL_68
                             2
                             3     ASSUME CS:CODE_PROG,DS:DATEN_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6                    DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
----                        11     DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                            12
0000 1500            R      13     NEAR_LABEL      DW      ZIEL                    ;Offset von ZIEL in diesem
                            14                                                     ;Segment
----                        15     DATEN_PROG ENDS
                            16
----                        17     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            18
0000 B8----          R      19     START:          MOV     AX,DATEN_PROG           ;DS Register initialisieren
0003 8ED8                   20                     MOV     DS,AX
0005 B8----          R      21                     MOV     AX,STACK_PROG           ;SS Register initialisieren
0008 8ED0                   22                     MOV     SS,AX
000A BC0A00          R      23                     MOV     SP,OFFSET TOP_STACK     ;SP Register initialisieren
                            24                             ;
                            25                             ;
000D FF260000        R      26     INTRA_1:        JMP     NEAR_LABEL              ;Sprung nach CS:(NEAR_LABEL)
                            27                                                     ;Intrasegment indirekt
                            28                             ;
0011 FF160000        R      29     INTRA_2:        CALL    NEAR_LABEL              ;Verzweige n. CS:(NEAR_LABEL)
                            30                                                     ;Intrasegment indirekt
                            31                             ;
0015 8AC1                   32     ZIEL:           MOV     AL,CL
                            33                             ;
                            34                             ;
                            35
----                        36     CODE_PROG ENDS
                            37
                            38             END     START
```

Die Befehle `JMP NEAR_LABEL` und `CALL NEAR_LABEL` bewirken, daß das Programm beim Label CS:(NEAR_LABEL) = CS:ZIEL fortgesetzt wird.

Der Operand NEAR_LABEL ist ein WORD-Pointer, der den Offset des Sprung- bzw. Verzweigungsziels ZIEL kennzeichnet.

Bei beiden Befehlen wird der Offset von ZIEL in den Instruction-Pointer IP geladen. D.h. diese indirekte Programmverzweigung erfolgt im augenblicklichen Codesegment (Intrasegment indirekt).

Eine DWORD-Variable enthält einen Offset, gefolgt von einem Segment-Basiswert und zeigt damit auf ein Sprung- bzw. Verzweigungsziel in einem beliebigen Segment.

Beispiel:

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_69
                             2
                             3     ASSUME CS:CODE_PROG,DS:DATEN_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6                  DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
----                        11     DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                            12
0000 0000----       R       13     FAR_LABEL       DD      ZIEL                    ;Offset von ZIEL und
                            14                                                     ;Basiswert von CODE_PROG_1
----                        15     DATEN_PROG ENDS
                            16
----                        17     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            18
0000 B8----         R       19     START:          MOV     AX,DATEN_PROG           ;DS Register initialisieren
0003 8ED8                   20                     MOV     DS,AX
0005 B8----         R       21                     MOV     AX,STACK_PROG           ;SS Register initialisieren
0008 8ED0                   22                     MOV     SS,AX
000A BC0A00         R       23                     MOV     SP,OFFSET TOP_STACK     ;SP Register initialisieren
                            24                             ;
                            25                             ;
000D FF2E0000       R       26     INTER_1:        JMP     FAR_LABEL               ;Sprung nach CS:(FAR_LABEL)
                            27                                                     ;Intersegment indirekt
                            28                             ;
0011 FF1E0000       R       29     INTER_2:        CALL    FAR_LABEL               ;Verzweige n. CS:(FAR_LABEL)
                            30                                                     ;Intersegment indirekt
                            31                             ;
----                        32     CODE_PROG ENDS
                            33
                            34     ASSUME CS:CODE_PROG_1
                            35
----                        36     CODE_PROG_1 SEGMENT BYTE PUBLIC 'ROM_REGION'
                            37                             ;
0000 8BCA                   38     ZIEL:           MOV     CX,DX
                            39                             ;
                            40                             ;
----                        41     CODE_PROG_1 ENDS
                            42
                            43             END     START
```

Die im Programmlisting angegebenen Befehle JMP FAR_LABEL und CALL FAR_LABEL bewirken, daß das Programm beim Label CS: (FAR_LABEL) = CS:ZIEL fortgesetzt wird.

Der Operand FAR_LABEL ist ein DWORD-Pointer, der den Offset und den Segment-Basiswert des Sprung- bzw. Verzweigungsziels ZIEL kennzeichnet.

Bei beiden Befehlen wird der Basiswert des Segments CODE_PROG_1 in das Register CS und der Offset von ZIEL (relativ zu CODE_PROG_1) in den Instruction-Pointer IP geladen. D.h. diese indirekte Programmverzweigung erfolgt in ein anderes Codesegment (Intersegment indirekt).

Die Anwendung von Intersegment-indirekt JMP- bzw. CALL-Befehlen sind eine mögliche Methode den Selektor CS umzuladen. Der Ausdruck **[BX]** wird als Pointer auf eine WORD-Variable (Offset) oder eine DWORD-Variable (Offset und Segment-Basiswert) behandelt. Wobei diese Variablen zum Ziel eines JMP- oder CALL-Befehls zeigen.

Beispiel:

```
LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_70
                             2
                             3     ASSUME CS:CODE_PROG,DS:DATEN_PROG,SS:STACK_PROG
                             4
----                         5     DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                             6
0000 1B00           R        7     NEAR_LABEL      DW      ZIEL_1              ;Offset von ZIEL_1 im
                             8                                                 ;Segment CODE_PROG
0002 0000----       R        9     FAR_LABEL       DD      ZIEL_2              ;Offset von ZIEL_2 und
                            10                                                 ;Basiswert von CODE_PROG_1
                            11
----                        12     DATEN_PROG ENDS
                            13
----                        14     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                     15                     DW      5 DUP (?)
     ????
     )
000A                        16     TOP_STACK LABEL WORD
                            17
----                        18     STACK_PROG ENDS
                            19
----                        20     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            21
0000 B8----         R       22     START:          MOV     AX,DATEN_PROG       ;DS Register initialisieren
0003 8ED8                   23                     MOV     DS,AX
0005 B8----         R       24                     MOV     AX,STACK_PROG       ;SS Register initialisieren
0008 8ED0                   25                     MOV     SS,AX
000A BC0A00         R       26                     MOV     SP,OFFSET TOP_STACK ;SP Register initialisieren
                            27                     ;
000D BB0000         R       28                     MOV     BX,OFFSET NEAR_LABEL
                            29                     ;
                            30                     ;
0010 FF27                   31     INTRA_1:        JMP     WORD PTR[BX]        ;Sprung nach CS:[BX]
                            32                                                 ;Intrasegment indirekt
                            33                     ;
0012 FF17                   34     INTRA_2:        CALL    WORD PTR[BX]        ;Verzweige nach CS:[BX]
                            35                                                 ;Intrasegment indirekt
                            36                     ;
                            37                     ;
0014 BB0200         R       38                     MOV     BX,OFFSET FAR_LABEL
```

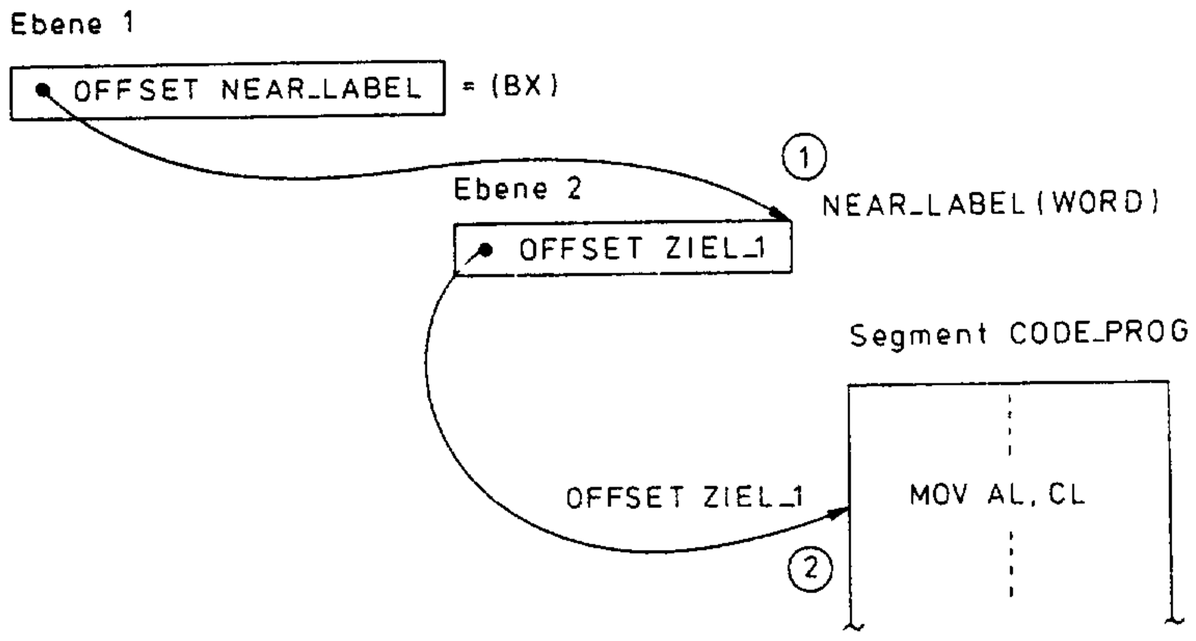
```
LOC  OBJ                LINE     SOURCE

                          39                          ;
                          40                          ;
0017 FF2F                 41     INTER_1:     JMP     DWORD PTR[BX]      ;Sprung nach CS:[BX]
                          42                                             ;Intersegment indirekt
                          43                          ;
0019 FF1F                 44     INTER_2:     CALL    DWORD PTR[BX]      ;Verzweige nach CS:[BX]
                          45                                             ;Intersegment indirekt
                          46                          ;
                          47                          ;
001B 8AC1                 48     ZIEL_1:      MOV     AL,CL
                          49                          ;
                          50                          ;
                          51
----                      52     CODE_PROG ENDS
                          53
                          54     ASSUME CS:CODE_PROG_1
                          55
----                      56     CODE_PROG_1 SEGMENT BYTE PUBLIC 'ROM_REGION'
                          57                          ;
                          58                          ;
0000 8BCA                 59     ZIEL_2:      MOV     CX,DX
                          60                          ;
                          61                          ;
                          62
----                      63     CODE_PROG_1 ENDS
                          64
                          65             END     START
```

Bei den im Beispiel verwendeten JMP- und CALL-Befehlen sind die Typ-Override-Operatoren WORD PTR und DWORD PTR verwendet. Sie kennzeichnen [BX] als Zeiger zur WORD-Variablen NEAR_LABEL bzw. zur DWORD-Variablen FAR_LABEL.

Diese Variablen zeigen ihrerseits zum Sprung-/Verzweigungsziel ZIEL_1 im CODE_PROG-Segment bzw. zum Sprung-/Verzweigungsziel ZIEL_2 im CODE_PROG_1-Segment.

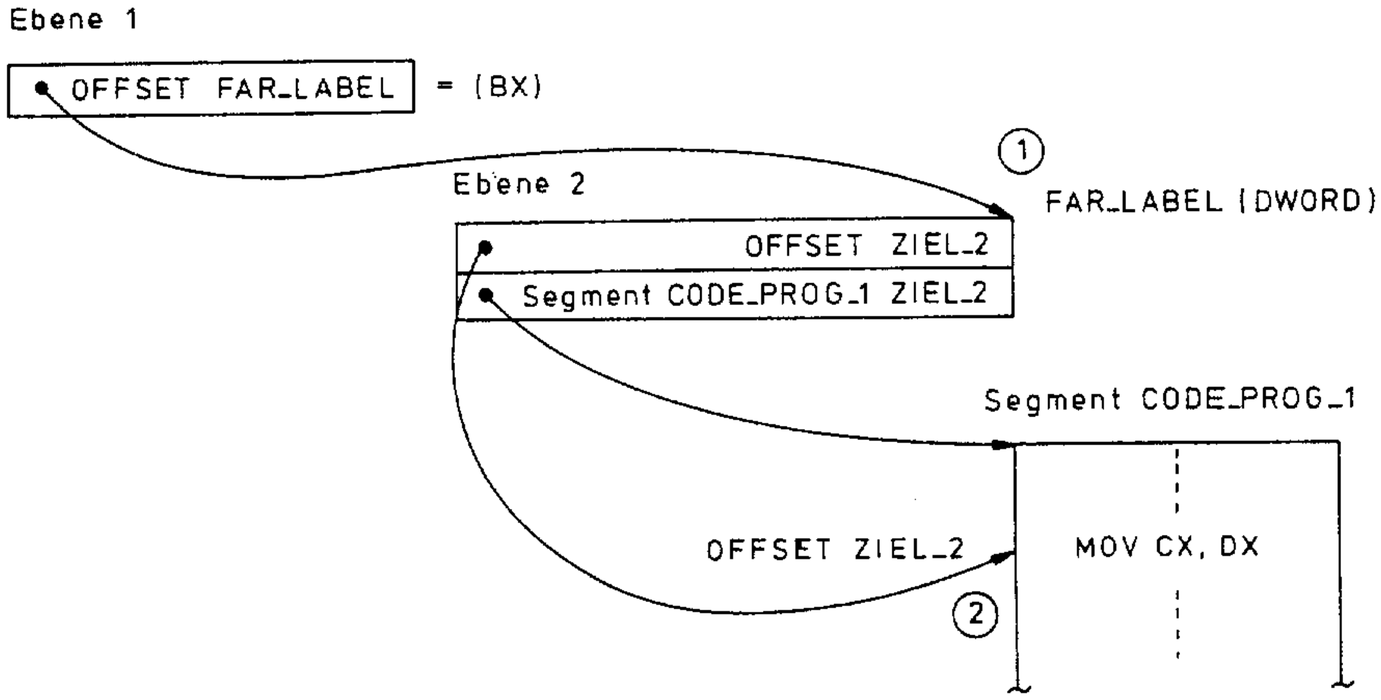
Die beiden Ebenen der Indirektion in `JMP/CALL WORD PTR [BX]` und in `JMP/CALL DWORD PTR [BX]` sind in folgenden Bildern schematisch dargestellt.

1. `JMP/CALL WORD PTR [BX]`



In Ebene 1 zeigt [BX] zur WORD-Variablen NEAR_LABEL in Ebene 2 ①. In dieser Ebene kennzeichnet NEAR_LABEL den Offset des Sprung-/Verzweigungsziels ZIEL_1. In Ebene 2 zeigt OFFSET ZIEL_1 zum nächsten auszuführenden Befehl im augenblicklichen Segment CODE_PROG ②. D.h. es wird nur der Instruction-Pointer IP umgeladen (Intrasegment-indirekt).

2. JMP/CALL DWORD PTR [BX]



In Ebene 1 zeigt [BX] zur DWORD-Variablen FAR_LABEL in Ebene 2 ①. In dieser Ebene kennzeichnet FAR_LABEL den Offset und den Segment-Basiswert des Sprung-/Verzweigungsziels ZIEL_2.

In Ebene 2 zeigt CODE_PROG_1: ZIEL_2 zum nächsten auszuführenden Befehl ②. In diesem Fall werden sowohl der Instruction-Pointer IP als auch der Selektor CS umgeladen (Intersegment-indirekt).

7.5 Labels als Operanden

Im Kapitel "Segmentierung aus der Sicht des Assembler-Programmierers" wurde bereits die LABEL-Direktive für **Variablen** benützt.

Es soll nun die LABEL-Direktive verwendet werden, um Namen für die Positionen von Befehlen zu definieren.

Syntax:

```
Symbolname    LABEL    Typ
```

Dabei ist Symbolname ein metasprachlicher Ausdruck mit dem folgende Attribute verbunden sind:

- Segment – das augenblicklich bearbeitete Codesegment
- Offset – der Abstand im augenblicklichen Codesegment
- Typ – der Operand von LABEL

Als Typ-Attribute sind für Befehle

- NEAR und
- FAR möglich.

NEAR- und FAR-Labels können nur in JMP- und CALL-Befehlen, aber nicht in MOV- oder anderen Datenmanipulations-Befehlen benützt werden. Das bedeutet, daß Labels vom Typ NEAR oder FAR nur erlaubt sind, wenn das augenblickliche Segment durch das CS-Register (Codesegment) adressiert wird.

In diesem Fall ist eine ASSUME-Direktive der Form `ASSUME CS: Name` erforderlich. Dabei ist "Name" entweder der Name des augenblicklichen Codesegments oder der Name einer GROUP, in der das augenblickliche Codesegment enthalten ist.

7.5.1 Adressierbarkeit von Labels

Werden in JMP- und CALL-Befehlen Labels als Operanden benützt, stellen Sie für den Assembler einen einfacheren Fall als bei Variablen dar. Der Grund liegt darin, daß nur das Segment-Register CS relevant ist. Die Adressierbarkeit eines Labels hängt davon ab, wie er deklariert ist und wie er benützt wird.

Deklaration: Ist ein Ziel-Label als NEAR oder FAR deklariert?

Benutzung: Werden die JMP- bzw. CALL-Befehle und ihre Ziele unter der gleichen Direktive `ASSUME CS: Name` assembliert?

Die folgende Tabelle zeigt im Einzelnen die Assemblierungsergebnisse.

	Ziel-Label als NEAR deklariert	**Ziel-Label als FAR deklariert**
JMP/CALL assembliert unter der gleichen ASSUME CS: Name-Direktive	**NEAR** JMP **NEAR** CALL	**FAR** JMP **FAR** CALL
JMP/CALL assembliert unter unterschiedlichen ASSUME CS: Name-Direktiven	... Fehler ...	**FAR** JMP **FAR** Call

In Anlehnung an diese Tabelle sei darauf hingewiesen, daß NEAR JMP/CALL-Befehle ein 2-Byte-Displacement (16 Bit Offset) benützen, so daß 64 KBytes in einem Segment adressierbar sind.

Dagegen benützen FAR JMP/CALL-Befehle ein 4-Byte-Displacement (einen 16-Bit-Offset und eine 16-Bit-Paragraph-Number). Damit sind insgesamt 1 MBytes adressierbar.

Der Assembler benützt die Information `ASSUME CS: Name` um sicher zu gehen, daß das Ziel für JMP- bzw. CALL-Befehle erreicht werden kann. Ist die Information `ASSUME CS:Name` beim Ziel eine andere als bei einem JMP- bzw. CALL-Befehl **und** ist das Ziel als FAR deklariert, generiert der Assembler automatisch FAR JMP/CALL-Befehle (4-Byte-Displacement).

Die folgenden Programmierbeispiele zeigen die Anwendung von NEAR- und FAR JMP/CALL-Befehlen.

7.5.2 Beispiel 1: NEAR JMP/CALL-Befehle

```
LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_71
                             2
                             3     ASSUME CS:CODE_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6             DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
----                        11     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            12     EXTRN ZIEL_1:NEAR,ZIEL_2:NEAR
                            13
0000 B8----         R       14     START:  MOV     AX,STACK_PROG           ;SS Register initialisieren
0003 8ED0                   15             MOV     SS,AX
0005 BC0A00         R       16             MOV     SP,OFFSET TOP_STACK     ;SP Register initialisieren
                            17                     ;
0008 E90000         E       18     EX_1:   JMP     ZIEL_1                  ;Sprungziel ist vom Typ NEAR
                            19                                             ;(2 Byte Displacement)
000B E80000         E       20     EX_2:   CALL    ZIEL_2                  ;Verzweigungsziel ist vom
                            21                                             ;Type NEAR
                            22                                             ;(2 Byte Displacement)
                            23                     ;
                            24                     ;
                            25
----                        26     CODE_PROG ENDS
                            27
                            28             END     START
```

```
LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_72
                             2
                             3     ASSUME CS:CODE_PROG
                             4
----                         5     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6     PUBLIC ZIEL_1,ZIEL_2
                             7                     ;
                             8                     ;
0000                         9     ZIEL_1 LABEL NEAR
0000 8BCA                   10             MOV     CX,DX
                            11                     ;
                            12                     ;
0002                        13     ZIEL_2 LABEL NEAR
0002 03C2                   14             ADD     AX,DX
                            15                     ;
                            16                     ;
----                        17     CODE_PROG ENDS
                            18
                            19             END
```

Es existieren zwei Quelldateien, in denen die Module BEISPIEL_71 und BEISPIEL_72 gespeichert sind. Im Modul BEISPIEL_71 wird mit `JMP ZIEL_1` und `CALL ZIEL_2` auf externe Labels verzweigt, die im Modul BEISPIEL_72 vom Typ NEAR deklariert sind.

Da sowohl für die JMP/CALL-Befehle, als auch für die Ziele ZIEL_1 und ZIEL_2 die gleiche Information `ASSUME CS: CODE_PROG` gilt, generiert der Assembler NEAR JMP/CALL-Befehle (2-Byte-Displacements). D.h. in

beiden Fällen wird nur der Instruction-Pointer IP umgeladen, also der Sprung bzw. die Verzweigung "Intrasegment-direkt" ausgeführt.

7.5.3 Beispiel 2: FAR JMP/CALL-Befehle

```
LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_73
                             2
                             3     ASSUME CS:CODE_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6             DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
                            11     EXTRN ZIEL_1:FAR,ZIEL_2:FAR           ;Da die Ziele in einem anderen
                            12                                           ;Segment liegen, ist die EXTRN An-
                            13                                           ;weisung ausserhalb von CODE_PROG
                            14                                           ;anzugeben
                            15
----                        16     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            17
0000 B8----          R      18     START:  MOV     AX,STACK_PROG         ;SS Register initialisieren
0003 8ED0                   19             MOV     SS,AX
0005 BC0A00          R      20             MOV     SP,OFFSET TOP_STACK   ;SP Register initialisieren
                            21                     ;
0008 EA0000----      E      22     EX_1:   JMP     ZIEL_1                ;Sprungziel ist vom Typ FAR
                            23                                           ;(4 Byte Displacement)
000D 9A0000----      E      24     EX_2:   CALL    ZIEL_2                ;Verzweigungsziel ist vom
                            25                                           ;Type FAR
                            26                                           ;(4 Byte Displacement)
                            27                     ;
                            28                     ;
                            29
----                        30     CODE_PROG ENDS
                            31
                            32             END     START


LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_74
                             2
                             3     ASSUME CS:CODE_PROG_1
                             4
----                         5     CODE_PROG_1 SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6     PUBLIC ZIEL_1,ZIEL_2
                             7                     ;
                             8                     ;
0000                         9     ZIEL_1 LABEL FAR
0000 8BCA                   10             MOV     CX,DX
                            11                     ;
                            12                     ;
0002                        13     ZIEL_2 LABEL FAR
0002 03C2                   14             ADD     AX,DX
                            15                     ;
                            16                     ;
----                        17     CODE_PROG_1 ENDS
                            18
                            19             END
```

Es existieren zwei Quelldatein, in denen die Module BEISPIEL_73 und BEISPIEL_74 gespeichert sind. Im Modul BEISPIEL_73 wird JMP ZIEL_1

und CALL ZIEL_2 auf externe Labels verzweigt, die im Modul BEISPIEL_74 vom Typ FAR deklariert sind.

Für die JMP/CALL-Befehle gilt die Information ASSUME CS: CODE_PROG. Für die Ziele ZIEL_1 und ZIEL_2 gilt hingegen die Information ASSUME CS: CODE_PROG_1. Die JMP/CALL-Befehle und ihre Ziele liegen also in verschiedenen Segmenten.

Der Assembler generiert daher FAR JMP/CALL-Befehle (4-Byte-Displacements). In beiden Fällen wird sowohl der Instruction-Pointer IP, als auch das Segment-Register CS umgeladen. D.h. der Sprung bzw. die Verzweigung wird "Intersegment direkt" ausgeführt.

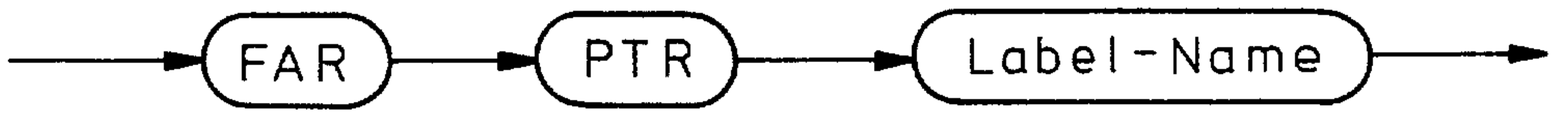
7.5.4 Typ-Overrides für Labels

In einem Codesegment ist jeder Label implizit durch das Typ-Attribut NEAR gekennzeichnet. Es kann nun vorkommen, daß in einem Modul für die JMP/CALL-Befehle und ihre Ziele unterschiedliche Informationen ASSUME CS: Name existieren.

D.h. in einem Modul sind mindestens zwei oder mehrere Codesegmente enthalten, wobei die JMP/CALL-Befehle und ihre Ziele in unterschiedlichen Segmenten liegen.

Um die Ziele zu erreichen, ist dem Assembler mitzuteilen, daß sowohl der Instruction-Pointer, als auch das Segment-Register CS umgeladen werden muß. Dies ist möglich, wenn die Ziel-Labels explizit mit dem Typ Attribut FAR "überschrieben" werden. In Assembler steht hierfür der bereits beschriebene Typ-Override-Operator zur Verfügung.

Syntax:



Der Ausdruck FAR PTR Label-Name besagt folgendes: Benütze das Segment und den Offset des Labels mit dem Namen **Label-Name**, aber vom explizit angegebenen Typ FAR.

Beispiel: FAR PTR

```
LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_75
                             2
                             3     ASSUME CS:CODE_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RZM_REGION'
0000 (5                      6             DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
----                        11     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            12
0000 B8----         R       13     START:  MOV     AX,STACK_PROG           ;SS Register initialisieren
0003 8ED0                   14             MOV     SS,AX
0005 BC0A00         R       15             MOV     SP,OFFSET TOP_STACK     ;SP Register initialisieren
                            16                     ;
0008 EA0000----     R       17     FAR_1:  JMP     FAR PTR ZIEL_1          ;Fuer die JMP/CALL Befehle
                            18                                             ;gilt ASSUME CS:CODE_PROG
                            19                     ;
000D 9A0200----     R       20     FAR_2:  CALL    FAR PTR ZIEL_2
                            21                     ;
                            22                     ;
----                        23     CODE_PROG ENDS
                            24
                            25     ASSUME CS:CODE_PROG_1
                            26
----                        27     CODE_PROG_1 SEGMENT BYTE PUBLIC 'ROM_REGION'
                            28                     ;
                            29                     ;
0000 8BCA                   30     ZIEL_1: MOV     CX,DX                   ;Fuer ZIEL_1 und ZIEL_2
                            31                     ;                       ;gilt ASSUME CS:CODE_PROG_1
                            32                     ;
0002 8BC2                   33     ZIEL_2: MOV     AX,DX
                            34                     ;
                            35                     ;
----                        36     CODE_PROG_1 ENDS
                            37
                            38             END     START
```

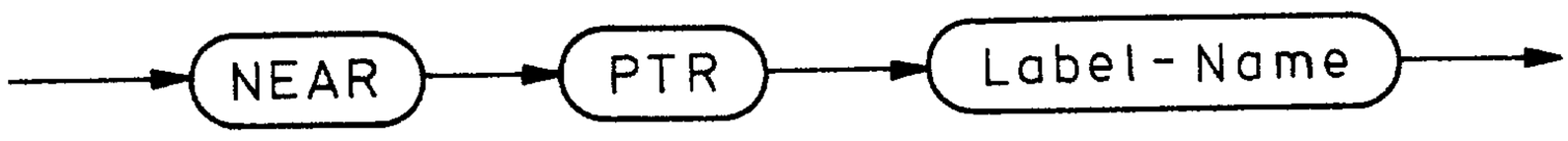
In der angegebenen Quelldatei ist der Modul BEISPIEL_75 gespeichert. Er enthält die zwei Code-Segmente CODE_PROG und CODE_PROG_1. Für die JMP/CALL-Befehle ist `ASSUME CS: CODE_PROG` wirksam. Für die Ziele ZIEL_1 und ZIEL_2 gilt `ASSUME CS: CODE_PROG_1`.

Die Typ-Override-Operatoren in `JMP FAR PTR ZIEL_1` und `CALL FAR PTR ZIEL_2` teilen dem Assembler mit, daß zum Erreichen der Ziele sowohl der Instruction-Pointer IP als auch das Segment-Register CS umgeladen werden muß (4-Byte-Displacement). In beiden Fällen wird der Sprung bzw. die Verzweigung "Intersegment direkt" durchgeführt.

Gelegentlich kommt es vor, daß in einem Modul Labels vom Typ FAR und JMP/CALL-Befehle existieren, für die die gleiche Information `ASSUME CS: Name` gilt. D.h. sowohl die Ziele, als auch die JMP/CALL-Befehle liegen im gleichen Segment.

Da die Ziele vom Typ FAR sind, generiert der Assembler FAR JMP/CALL-Befehle. Es wird, obwohl die Ziele und die JMP/CALL-Befehle im gleichen Segment liegen, der Instruction-Pointer IP als auch das Segmentregister CS umgeladen. Das CS-Register wird dabei mit seinem vorherigen Wert initialisiert. Um dies zu vermeiden, muß dem Assembler mitgeteilt werden, daß nur der Instruction-Pointer IP umzuladen ist. Dies ist möglich, wenn die Ziel-Labels explizit mit dem Typ-Attribut NEAR "überschrieben" werden. In Assembler steht hierfür der Typ-Override-Operator NEAR PTR zur Verfügung.

Syntax:



Der Ausdruck **NEAR PTR Label-Name** besagt folgendes: Benütze das Segment und den Offset des Labels mit dem Namen **Label-Name**, aber vom explizit angegebenen Typ NEAR.

Beispiel 1: NEAR PTR

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_76
                             2
                             3     ASSUME CS:CODE_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6            DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
                            11     EXTRN ZIEL_1:FAR,ZIEL_2:FAR          ;Da die Ziele in einem anderen
                            12                                          ;Segment liegen, ist die EXTRN An-
                            13                                          ;weisung ausserhalb von CODE_PROG
                            14                                          ;anzugeben
                            15
----                        16     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            17
0000 B8----         R       18     START:  MOV     AX,STACK_PROG       ;SS Register initialisieren
0003 8ED0                   19             MOV     SS,AX
0005 BC0A00         R       20             MOV     SP,OFFSET TOP_STACK ;SP Register initialisieren
                            21             ;
0008 EA0000----     E       22     EX_1:   JMP     ZIEL_1              ;Sprungziel ist vom Type FAR
                            23                                         ;(4 Byte Displacement)
000D 9A0000----     E       24     EX_2:   CALL    ZIEL_2              ;Verzweigungsziel ist vom
                            25                                         ;Typ FAR
                            26                                         ;(4 Byte Displacement)
                            27             ;
                            28             ;
                            29
----                        30     CODE_PROG ENDS
                            31
                            32             END     START
```

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_77
                             2
                             3     ASSUME CS:CODE_PROG_1
                             4
----                         5     CODE_PROG_1 SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6     PUBLIC ZIEL_1,ZIEL_2
                             7                    ;
                             8                    ;
0000                         9     ZIEL_1 LABEL FAR
0000 8BCA                   10            MOV     CX,DX
0002 (129                   11                    DW      129 DUP (?)          ;Diese Definition dient nur
     ????
     )
                            12                                                 ;zu Demonstrationszwecken
                            13                    ;
0104                        14     ZIEL_2 LABEL FAR
0104 03C2                   15            ADD     AX,DX
                            16                    ;
                            17                    ;
0106 E9F7FE                 18            JMP     NEAR PTR ZIEL_1              ;Sprungziel ist vom Typ NEAR
                            19                                                 ;(2 Byte Displacement)
                            20                    ;
0109 E8F8FF                 21            CALL    NEAR PTR ZIEL_2              ;Verzweigungsziel ist vom
                            22                                                 ;Typ NEAR
                            23                                                 ;(2 Byte Displacement)
                            24                    ;
                            25                    ;
----                        26     CODE_PROG_1 ENDS
                            27
                            28            END
```

Es existieren zwei Quelldateien, in denen die Module BEISPIEL_76 und BEISPIEL_77 gespeichert sind. Im Modul BEISPIEL_77 sind die Ziel-Labels ZIEL_1 und ZIEL_2 vom Typ FAR vereinbart. Im CODE_PROG_1-Segment werden sie mit NEAR JMP/CALL-Befehlen (2 Byte Displacements) nur erreicht, wenn ihr Typ mit dem Typ-Override-Operator NEAR PTR "überschrieben" wird. So laden die Befehle `JMP NEAR PTR ZIEL_1` und `CALL NEAR PTR ZIEL_2` nur den Instruction-Pointer IP um; der Sprung bzw. die Verzweigung wird damit "Intrasegment direkt" durchgeführt.

Beispiel 2: NEAR PTR

```
LOC  OBJ                  LINE     SOURCE

                             1            NAME    BEISPIEL_78
                             2
                             3     ASSUME CS:CODE_PROG,SS:STACK_PROG
                             4
----                         5     STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
0000 (5                      6            DW      5 DUP (?)
     ????
     )
000A                         7     TOP_STACK LABEL WORD
                             8
----                         9     STACK_PROG ENDS
                            10
                            11     EXTRN ZIEL_1:FAR,ZIEL_2:FAR                ;Da die Ziele in einem anderen
                            12                                                ;Segment liegen, ist die EXTRN An-
                            13                                                ;weisung ausserhalb von CODE_PROG
                            14                                                ;anzugeben
                            15
```

```
LOC  OBJ                  LINE     SOURCE

----                        16     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            17
0000 B8----         R       18     START:  MOV     AX,STACK_PROG           ;SS Register initialisieren
0003 8ED0                   19             MOV     SS,AX
0005 BC0A00         R       20             MOV     SP,OFFSET TOP_STACK     ;SP Register initialisieren
                            21             ;
0008 EA0000----     E       22     EX_1:   JMP     ZIEL_1                  ;Sprungziel ist vom Typ FAR
                            23                                             ;(4 Byte Displacement)
000D 9A0000----     E       24     EX_2:   CALL    ZIEL_2                  ;Verzweigungsziel ist vom
                            25                                             ;Typ FAR
                            26                                             ;(4 Byte Displacement)
                            27             ;
                            28             ;
                            29
----                        30     CODE_PROG ENDS
                            31
                            32             END     START


LOC  OBJ                  LINE     SOURCE

                             1             NAME    BEISPIEL_79
                             2
                             3     ASSUME CS:CODE_PROG_1
                             4
----                         5     CODE_PROG_1 SEGMENT BYTE PUBLIC 'ROM_REGION'
                             6     PUBLIC ZIEL_1,ZIEL_2
                             7                     ;
                             8                     ;
0000                         9     ZIEL_1 LABEL FAR
0000 8BCA                   10     NEAR_1: MOV     CX,DX
                            11                     ;
                            12                     ;
0002                        13     ZIEL_2 LABEL FAR
0002 03C2                   14     NEAR_2: ADD     AX,DX
                            15                     ;
                            16                     ;
0004 EBFA                   17             JMP     NEAR PTR ZIEL_1         ;Sprungziel ist vom Typ NEAR
                            18                                             ;(2 Byte Displacement)
                            19                     ;
0006 E8F9FF                 20             CALL    NEAR PTR ZIEL_2         ;Verzweigungsziel ist vom
                            21                                             ;Typ NEAR
                            22                                             ;(2 Byte Displacement)
                            23                     ;
0009 EBF5                   24             JMP     NEAR_1                  ;Sprungziel ist vom Typ NEAR
                            25                                             ;Der Befehl ist optimiert
                            26                                             ;(1 Byte Displacement)
                            27                     ;
000B E8F4FF                 28             CALL    NEAR_2                  ;Verzweigungsziel ist vom
                            29                                             ;Typ NEAR
                            30                                             ;(2 Byte Displacement)
                            31                     ;
----                        32     CODE_PROG_1 ENDS
                            33
                            34             END
```

Es existieren zwei Quelldatein, in denen die Module BEISPIEL_78 und BEISPIEL_79 gespeichert sind. Im Modul BEISPIEL_79 markieren jeweils zwei Labels unterschiedlichen Typs die Befehle `MOV CX, DX` und `ADD AX, DX`.. So sind die Labels ZIEL_1 und ZIEL_2 explizit vom Typ FAR vereinbart, während die Labels NEAR_1 und NEAR_2 implizit vom Typ NEAR existieren. Dadurch ist es möglich, die gleichen Ziele entweder "Intrasegment direkt" mit `JMP/CALL NEAR_1/NEAR_2` oder mit `JMP/CALL NEAR PTR ZIEL_1/ZIEL_2` zu erreichen.