Kapitel 6

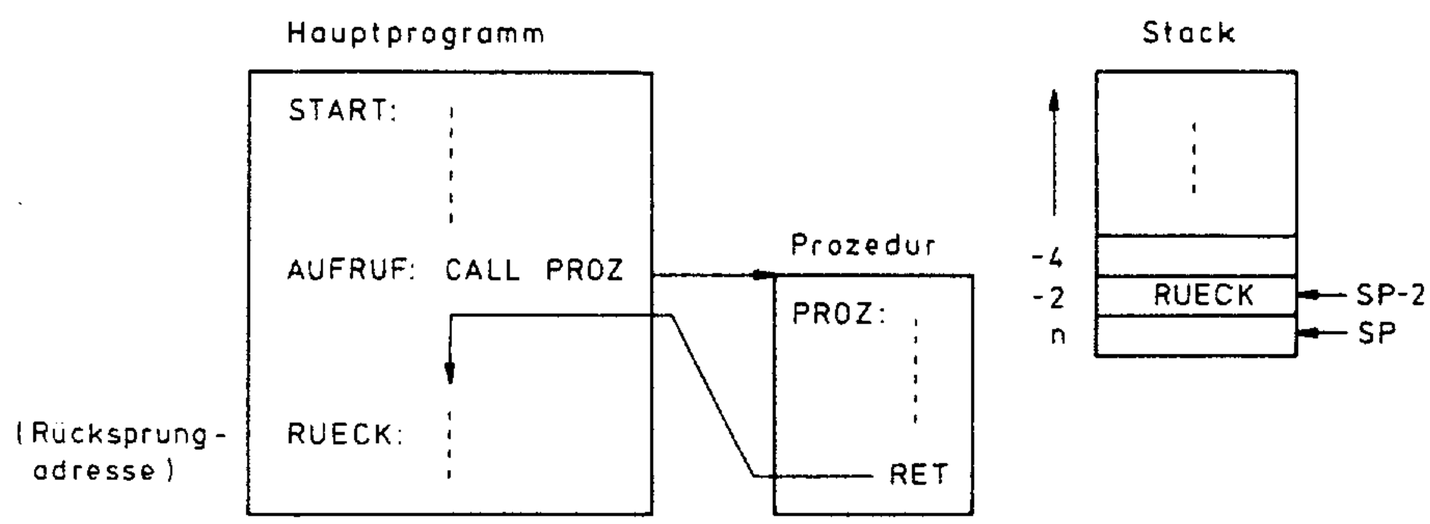
Prozeduren

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

Eine Prozedur ist eine Sequenz von Befehlen, die einen oder mehrere RET-Befehle enthält. Sie wird durch einen CALL-Befehl aktiviert. Oft ist eine Prozedur als funktioneller Block aufgebaut. Sie wird mit einem definierten Satz von Eingabedaten, den sogenannten **Parametern** versorgt. Die Prozedur wird diese Eingabedaten bearbeiten und als Ergebnis einen Satz von Ausgabedaten liefern.

Die Position, ab der die Ausführung einer Prozedur beginnt, ist mit einem Label markiert. Ein CALL-Befehl, der diesen Label als seinen Operanden benutzt, wird zunächst die Rücksprungadresse (Adresse des Befehls nach dem CALL-Befehl) im Stack ablegen und bewirken, daß anschließend der Code in der Prozedur ab dem Label ausgeführt wird.

Der Code in der Prozedur wird solange ausgeführt, bis ein **RET**-Befehl erreicht ist. Danach wird das Programm an der Stelle fortgesetzt, die durch die Rücksprungadresse im Stack spezifiziert ist.



In Assembler werden die **PROC**- und **ENDP**-Direktiven benützt, um die Ein- und Austrittsstellen der Prozedur zu markieren. Die PROC-Anweisung gehört an den Anfang und die ENDP-Anweisung an das Ende der Prozedur.

Die Prozedur- und END-Prozedur-Anweisung:

```
Prozedur-Name    PROC  NEAR  FAR
                 .
                 .
                 (Prozedurkoerper)
                 .
                 .
Prozedur-Name    ENDP
```

Der **Prozedur-Name** ist ein Label (vom Anwenderprogrammierer zu wählen) und kennzeichnet die Eintrittsstelle in die Prozedur. Der Typ des Labels ist NEAR oder FAR und steht rechts neben dem Schlüsselwort PROC. Ist kein Typ angegeben, setzt der Assembler NEAR voraus.

Den Typ des Labels **Prozedur-Name** benutzt der Assembler, um zu bestimmen, welchen CALL-Befehl er generieren soll.

- Wenn FAR angegeben ist, wird die lange Form des CALL-Befehls verwendet. Das bedeutet, daß auch eine Zwei-Wort-Rücksprungadresse (CS und IP) im Stack abgelegt wird.
- Wenn NEAR angegeben ist, wird **nur** der IP-Wert geändert. Das bedeutet, daß auch nur eine Ein-Wort-Rücksprungadresse (IP) im Stack abgelegt wird.

Da zwei Arten von Rücksprungadressen existieren, muß es auch zwei Arten von RET-Befehlen geben.

- Der RET-Befehl für eine FAR-Prozedur lädt zwei Worte aus dem Stack in die CS- und IP-Register einer 8086/80186/80286-CPU.
- Der RET-Befehl für eine NEAR-Prozedur lädt nur ein Wort aus dem Stack in das IP-Register einer 8086/80186/80286-CPU.

Beide Typen von RET-Befehlen sind in der RET-Mnemonik spezifiziert. Aufgrund des Prozedurtypes, der im PROC/ENDP-Statement angegeben ist, entscheidet der Assembler, welchen RET-Befehl er generieren wird. Die folgende einfache Prozedur wird benützt, um in DX den Mittelwert der Werte in AX und BX zu laden.

```
LOC  OBJ                  LINE     SOURCE

                            1              NAME    BEISPIEL_51
                            2
                            3      ASSUME CS:CODE_PROG
                            4
----                        5      CODE_PROG SEGMENT
                            6
0000                        7      MITTEL_WERT PROC NEAR
0000 8BD0                   8              MOV     DX,AX        ;(DX) = erste Zahl
0002 03D3                   9              ADD     DX,BX        ;(DX) = erste+zweite Zahl
0004 D1DA                  10              RCR     DX,1         ;(DX) = (AX+BX)/2
                           11                                   ;       (erste+zweite Zahl)/2
0006 C3                    12              RET
                           13
                           14      MITTEL_WERT ENDP
                           15
----                       16      CODE_PROG ENDS
                           17
                           18              END
```

Die Eintrittsstelle in die angegebene Prozedur ist durch den Label MITTEL_WERT gekennzeichnet. Da MITTEL_WERT den Typ NEAR hat, wird ein irgendwo im Programm angegebener Befehl `CALL MITTEL_WERT` einen CALL-Befehl generieren, der nur IP ändert.

Da der RET-Befehl innerhalb des PROC/ENDP-Paares einer NEAR-Prozedur angegeben ist, wird ein RET-Befehl generiert, der aus dem Stack nur ein Wort in das IP-Register lädt. Das PROC/ENDP-Paar wird also benützt, um für die Eintrittsstelle einer Prozedur den Typ eines Labels festzulegen und um zu bestimmen, welcher RET-Befehl generiert werden soll.

6.1 Übergabe von Parametern an eine Prozedur

Die Werte, die als Eingabedaten an eine Prozedur übergeben werden, heißen Parameter. Als Parameter können z.B. numerische Werte oder Zeigerwerte an eine Prozedur übergeben werden.

Es existieren mehrere Methoden der Parameterübergabe:

1. Eine Methode ist, Übergabewerte in Variablen der Datenregion zu speichern. Von dort können dann die Werte von der Prozedur übernommen werden.

```
LOC  OBJ                    LINE     SOURCE

                              1              NAME    BEISPIEL_52
                              2
                              3      ASSUME CS:CODE_PROG,DS:DATEN_PROG
                              4
----                          5      DATEN_PROG SEGMENT 'RAM_REGION'
                              6
0000 6400                     7      ZAHLEN_PAAR     DW      100,28          ;Feld mit zwei Zahlen
0002 1C00
                              8
----                          9      DATEN_PROG ENDS
                             10
----                         11      CODE_PROG SEGMENT 'ROM_REGION'
                             12
0000 B8----          R       13      START:  MOV     AX,DATEN_PROG           ;DS Register initialisieren
0003 8ED8                    14              MOV     DS,AX
0005 E80100                  15              CALL    MITTEL_WERT             ;Prozeduraufruf
0008 F4                      16              HLT                             ;Prozessorhalt
                             17
0009                         18      MITTEL_WERT PROC NEAR
                             19
0009 8B160000                20              MOV     DX,ZAHLEN_PAAR          ;(DX) = erste Zahl
000D 8B1E0200                21              MOV     BX,ZAHLEN_PAAR+2        ;(BX) = zweite Zahl
0011 03D3                    22              ADD     DX,BX                   ;(DX) = (erste Zahl+zweite Zahl)
0013 D1DA                    23              RCR     DX,1                    ;(DX) = Mittelwert der beiden
                             24                                              ;       Zahlen
0015 C3                      25      FERTIG: RET
                             26
                             27      MITTEL_WERT ENDP
```

```
LOC  OBJ                    LINE     SOURCE

                              28
----                          29     CODE_PROG ENDS
                              30
                              31             END     START
```

Im angegebenen Programmbeispiel soll in der Prozedur MITTEL_WERT der Mittelwert zweier Zahlen gebildet werden. Die Zahlen sind im DATEN_PROG Segment definiert. Sie sollen als Eingabeparameter an die Prozedur MITTEL_WERT übergeben werden. Die Prozedur wird mit `CALL MITTEL_WERT` aufgerufen. Sie übernimmt die beiden Zahlen von der Variablen ZAHLEN_PAAR im DATEN_PROG Segment mit `MOV DX, ZAHLEN_PAAR` und `MOV BX,ZAHLEN_PAAR+2` und berechnet anschließend den Mittelwert.

2. Eine andere Methode ist, Übergabewerte in Registern zu speichern.

```
LOC  OBJ                    LINE     SOURCE

                              1             NAME    BEISPIEL_53
                              2
                              3     ASSUME CS:CODE_PROG,DS:DATEN_PROG
                              4
----                          5     DATEN_PROG SEGMENT 'RAM_REGION'
                              6
0000 6400                     7     ZAHLEN_PAAR     DW      100,28          ;Feld mit zwei Zahlen
0002 1C00
                              8
----                          9     DATEN_PROG ENDS
                             10
----                         11     CODE_PROG SEGMENT 'ROM_REGION'
                             12
0000 B8----          R       13     START:  MOV     AX,DATEN_PROG           ;DS Register initialisieren
0003 8ED8                    14             MOV     DS,AX
0005 A10000                  15             MOV     AX,ZAHLEN_PAAR          ;(AX) = erste Zahl
0008 8B1E0200                16             MOV     BX,ZAHLEN_PAAR+2        ;(BX) = zweite Zahl
000C E80100                  17             CALL    MITTEL_WERT             ;Prozeduraufruf
000F F4                      18             HLT                             ;Prozessorhalt
                             19
0010                         20     MITTEL_WERT PROC NEAR
                             21
0010 8BD0                    22             MOV     DX,AX                   ;(DX) = erste Zahl
0012 03D3                    23             ADD     DX,BX                   ;(DX) = (erste Zahl+zweite Zahl)
0014 D1DA                    24             RCR     DX,1                    ;(DX) = Mittelwert der beiden
                             25                                             ;       Zahlen
0016 C3                      26     FERTIG: RET
                             27
                             28     MITTEL_WERT ENDP
                             29
----                         30     CODE_PROG ENDS
                             31
                             32             END     START
```

Im angegebenen Programmbeispiel soll in der Prozedur MITTEL_WERT der Mittelwert zweier Zahlen gebildet werden. Die Zahlen werden zunächst in den Registern AX und BX mit `MOV AX, ZAHLEN_PAAR` und `MOV BX, ZAHLEN_PAAR+2` zwischengespeichert. Die Registerwerte AX und BX sollen als Eingabeparameter an die Prozedur MITTEL_WERT übergeben

werden. Sie übernimmt diese Parameter mit `MOV DX, AX` und `ADD DX, BX`, und berechnet anschließend den Mittelwert.

6.2 Rückgewinnung eines Wertes von der Prozedur

Als Wirkung einer Prozedur wird entweder eine Datenstruktur verändert, oder ein Wert geliefert.

Eine Prozedur kann eine Variable oder ein Register benützen, um einen Wert dem aufrufenden Programm zu übergeben. Stellt eine Prozedur einen **einzelnen** Wert zur Verfügung, wird sie als **Funktion** betrachtet.

Im angegebenen Modul BEISPIEL_54 ist die Prozedur MITTEL_WERT_BERECHNUNG eine Funktion. Sie bildet aus den Werten in AX und BX den Mittelwert und stellt ihn im Register DX zur Verfügung. Das aufrufende Programm übernimmt den Mittelwert von DX und legt ihn mit `MOV MITTEL_WERT, DX` in der Variablen MITTEL_WERT ab ①.

```
LOC  OBJ                  LINE     SOURCE

                            1              NAME    BEISPIEL_54
                            2
                            3      ASSUME CS:CODE_PROG,DS:DATEN_PROG
                            4
----                        5      DATEN_PROG SEGMENT 'RAM_REGION'
                            6
0000 6400                   7      ZAHLEN_PAAR     DW      100,28          ;Feld mit zwei Zahlen
0002 1C00
0004 (1                     8      MITTEL_WERT     DW      1 DUP (?)       ;Mittelwert der beiden Zahlen
     ????
     )
                            9
----                       10      DATEN_PROG ENDS
                           11
----                       12      CODE_PROG SEGMENT 'ROM_REGION'
                           13
0000 B8----          R     14      START:  MOV     AX,DATEN_PROG           ;DS Register initialisieren
0003 8ED8                  15              MOV     DS,AX
0005 A10000                16              MOV     AX,ZAHLEN_PAAR          ;(AX) = erste Zahl
0008 8B1E0200              17              MOV     BX,ZAHLEN_PAAR+2        ;(BX) = zweite Zahl
000C E80500                18              CALL    MITTEL_WERT_BERECHNUNG  ;Prozeduraufruf
000F 89160400              19              MOV     MITTEL_WERT,DX          ;Mittelwert speichern   ①
0013 F4                    20              HLT                             ;Prozessorhalt
                           21
0014                       22      MITTEL_WERT_BERECHNUNG PROC NEAR
                           23
0014 8BD0                  24              MOV     DX,AX                   ;(DX) = erste Zahl
0016 03D3                  25              ADD     DX,BX                   ;(DX) = (erste Zahl+zweite Zahl)
0018 D1DA                  26              RCR     DX,1                    ;(DX) = Mittelwert der beiden
                           27                                              ;       Zahlen
001A C3                    28      FERTIG: RET
                           29
                           30      MITTEL_WERT_BERECHNUNG ENDP
                           31
----                       32      CODE_PROG ENDS
                           33
                           34              END     START
```

6.3 Programmbeispiel

Das angegebene Programmbeispiel ist in zwei Module BEISPIEL_55 und BEISPIEL_56 unterteilt. Es zeigt an einer **konkreten** Aufgabe die Anwendung von GROUPS, PUBLICS, EXTRNs und einer Prozedur.

```
LOC  OBJ                  LINE     SOURCE

                            1              NAME    BEISPIEL_55
                            2
                            3      DATEN_GROUP GROUP DATEN_PROG,STACK_PROG
                            4
                            5      ASSUME CS:CODE_PROG,DS:DATEN_GROUP
                            6
----                        7      DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                            8
0000 (4                     9      MITTEL_WERTE    DW      4 DUP (?)              ;Feld, das die Mittelwerte
     ????
     )
                           10                                                     ;von Zahlenpaaren aus einem
                           11                                                     ;externen Modul speichert
                           12      EXTRN ZAHLEN_PAAR:WORD,ZAEHLER:ABS
                           13
----                       14      DATEN_PROG ENDS
                           15
----                       16      STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
                           17
0000 (5                    18                      DW      5 DUP (?)
     ????
     )
000A                       19      TOP_STACK LABEL WORD
                           20
----                       21      STACK_PROG ENDS
                           22
----                       23      CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                           24
                           25      EXTRN MITTEL_WERT_BERECHNUNG:NEAR
                           26
0000 B8----        R       27      START:  MOV     AX,DATEN_GROUP                 ;DS Register und
0003 8ED8                  28              MOV     DS,AX                          ;SS Register mit gemeinsamer
0005 8ED0                  29              MOV     SS,AX                          ;Basis initialisieren
0007 BC0A00        R       30              MOV     SP,OFFSET DATEN_GROUP:TOP_STACK ;SP Register initialisieren
000A B90000        E       31              MOV     CX,ZAEHLER                     ;Anzahl  Mittelwertbildungen
                           32                                                     ;festlegen
000D BE0000                33              MOV     SI,0                           ;Indexwert=0 fuer das externe
                           34                                                     ;Feld, das die Zahlenpaare
                           35                                                     ;enthaelt
0010 8BFE                  36              MOV     DI,SI                          ;Indexwert=0 fuer das externe
                           37                                                     ;Feld, das die Mittelwerte
                           38                                                     ;enthaelt
0012 8B840000      E       39      MITTEL: MOV     AX,ZAHLEN_PAAR[SI]             ;(AX) = erste Zahl
0016 8B9C0200      E       40              MOV     BX,ZAHLEN_PAAR[SI+2]           ;(BX) = zweite Zahl
001A E80000        E       41              CALL    MITTEL_WERT_BERECHNUNG         ;Prozeduraufruf
001D 89950000      R       42              MOV     MITTEL_WERTE[DI],DX            ;Mittelwert speichern
0021 83C604                43              ADD     SI,4                           ;Index zur ersten Zahl des
                           44                                                     ;nae. Zahlenpaares bilden
0024 83C702                45              ADD     DI,2                           ;Index zum naechsten Wort des
                           46                                                     ;Mittelwertes bilden
0027 E2E9                  47              LOOP    MITTEL                         ;Wiederhole bis die Mittel-
                           48                                                     ;werte aller Zahlenpaare
                           49                                                     ;gebildet sind
0029 F4                    50              HLT                                    ;Prozessorhalt
                           51
----                       52      CODE_PROG ENDS
                           53
                           54              END     START
```

Das Programm berechnet die Mittelwerte von vier Zahlenpaaren, die im Modul BEISPIEL_56 im Feld ZAHLEN_PAAR definiert sind. Die Mittelwerte werden im Modul BEISPIEL_55 im Feld MITTEL_WERTE gespeichert.

```
LOC  OBJ                LINE     SOURCE

                          1              NAME    BEISPIEL_56
                          2
                          3      DATEN_GROUP GROUP DATEN_PROG,STACK_PROG
                          4
                          5      ASSUME CS:CODE_PROG,DS:DATEN_GROUP
                          6
 ----                     7      DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                          8
 0000 7000                9      ZAHLEN_PAAR     DW      112,56          ;Ein Feld von vier Zahlenpaaren
 0002 3800
 0004 8F0D               10                      DW      3471,29
 0006 1D00
 0008 8301               11                      DW      387,6123
 000A EB17
 000C 1300               12                      DW      19,8
 000E 0800
   0004                  13      ZAEHLER EQU 4                           ;Die Konstante gibt die Anzahl der
                         14                                              ;Zahlenpaare an
                         15      PUBLIC ZAHLEN_PAAR,ZAEHLER
                         16
 ----                    17      DATEN_PROG ENDS
                         18
 ----                    19      STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
 0000 (2                 20                      DW      2 DUP (?)       ;Addiere zwei Worte zur Stacklaenge
      ????
      )
                         21
 ----                    22      STACK_PROG ENDS
                         23
 ----                    24      CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                         25
                         26      PUBLIC MITTEL_WERT_BERECHNUNG
                         27
 0000                    28      MITTEL_WERT_BERECHNUNG PROC NEAR
                         29
 0000 8BD0               30              MOV     DX,AX                   ;(DX) = erste Zahl
 0002 03D3               31              ADD     DX,BX                   ;(DX) = erste+zweite Zahl
 0004 D1DA               32              RCR     DX,1                    ;(DX) = Mittelwert der beiden
                         33                                              ;       Zahlen
 0006 C3                 34              RET
                         35
                         36      MITTEL_WERT_BERECHNUNG ENDP
                         37
 ----                    38      CODE_PROG ENDS
                         39
                         40              END
```

Im Modul BEISPIEL_55 werden die Zahlenpaare vom **externen** Feld ZAHLEN_PAAR gelesen. Damit vier Mittelwertbildungen durchgeführt werden, ist ein ZAEHLER mitgeführt, dessen Wert vom Typ ABS **extern** durch eine EQU-Anweisung definiert ist.

Die eigentliche Mittelwertberechnung erfolgt in der Prozedur MITTEL_WERT_BERECHNUNG. Da sie im gleichen physikalischen Segment liegt wie das aufrufende Programm, ist ihr Typ als **NEAR** vereinbart.

Die logischen Segmente DATEN_PROG und STACK_PROG sind zur GROUP DATEN_GROUP kombiniert. Da in DS die Basis der GROUP geladen ist, werden die Variablen relativ dazu adressiert. Es wäre also **inkorrekt** dem Assembler `ASSUME DS:DATEN_PROG` mitzuteilen. Da die Code-Region in keine Group eingebunden ist, zeigt CS zur Basis von CODE_PROG, daher ist `ASSUME CS: CODE_PROG` korrekt. Zusätzlich ist `ASSUME SS: NOTHING` wirksam.

6.4 HLL-Methoden (High Level Language) der Parameter-Übergabe

Wie bereits erwähnt, gehört zum Konzept der modularen Programmierung, daß Teile eines Programms in Assembler und andere Teile in einer höheren Programmiersprache, wie z.B. C geschrieben werden. In der C-Sprache werden alle Parameter für eine Prozedur an den Stack übergeben. Am Beispiel des folgenden C-Programm-Fragments soll dies verdeutlicht werden:

```
int   wpara;
char  bpara;
int   *ppara;
beispiel()
{
void drei_parameter();                    /* Funktionsdeklaration */
drei_parameter (wpara,bpara,ppara);       /* Funktionsaufruf */
}
```

Wie zu sehen ist, ruft das C-Programm die Funktion drei_parameter auf, die als externe Assembler-FAR-Prozedur in einem anderen Modul existieren soll. Dabei übergibt das C-Programm an die Assembler-Prozedur den **Word**-Parameter wpara (Typ **int**) den **Byte**-Parameter bpara (Typ **char**) und den Zeiger (*ppara), der auf eine Integer-Variable zeigt. Dabei soll *ppara ein sogenannter **Long-Pointer**, also ein kompletter Basis:Offset-Zeiger, sein.

Es sei darauf hingewiesen, daß der Byte-Parameter bpara (Typ char) als **Word** an den Stack übergeben wird. Dabei wird der char-Wert im Low-Teil des Words gespeichert, während der High-Teil undefiniert oder sign extended (FFH oder 00H) ist.

Wenn das angegebene C-Programm z. B. mit

```
>iC86  DAT.C  SMALL  VARPARAMS
```

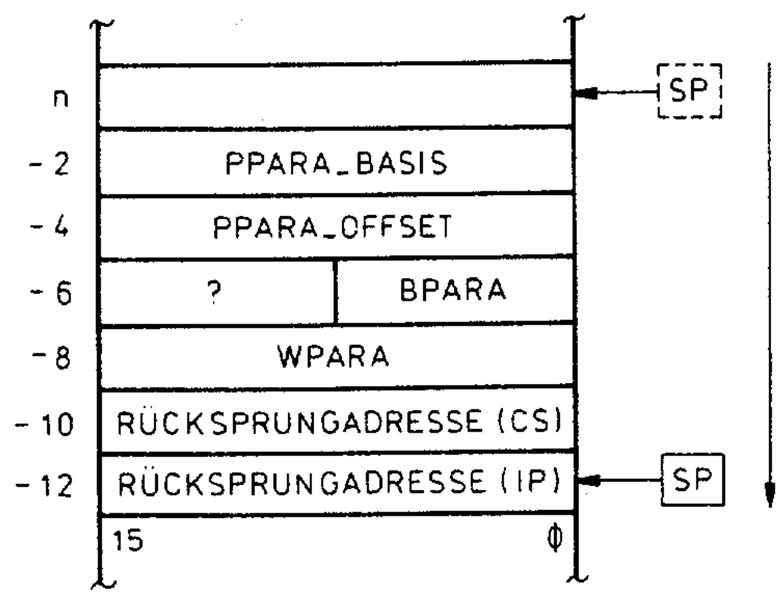
LARGE

- VARPARAMS → **Variable** Anzahl von Funktions-Parametern. Sie werden von **rechts** nach **links** an den CPU-Stack übergeben
- SMALL → Übersetzungsmodell ~~SMALL~~ LARGE wählen
- DAT.C → DAT.C enthält das C-Quellprogramm
- iC86 → Aufruf des INTEL-C-Compilers

compiliert wird, dann setzt der C-Compiler einen Code ab, der im wesentlichen folgendermaßen aussieht:

```
    :
PUSH   ppara_basis          ;Basis-Komponente des Long-Pointer-
                            ;Parameters *ppara
PUSH   ppara_offset         ;Offset-Komponente des Long-Pointer-
                            ;Parameters *ppara
PUSH   bpara                ;Byte-Parameter bpara (Typ char)
PUSH   wpara                ;Word-Parameter wpara (Typ int)
CALL   drei_parameter       ;Aufruf der Assembler-FAR-Prozedur
ADD    SP,08H               ;Parameter aus dem Stack entfernen
    :
```

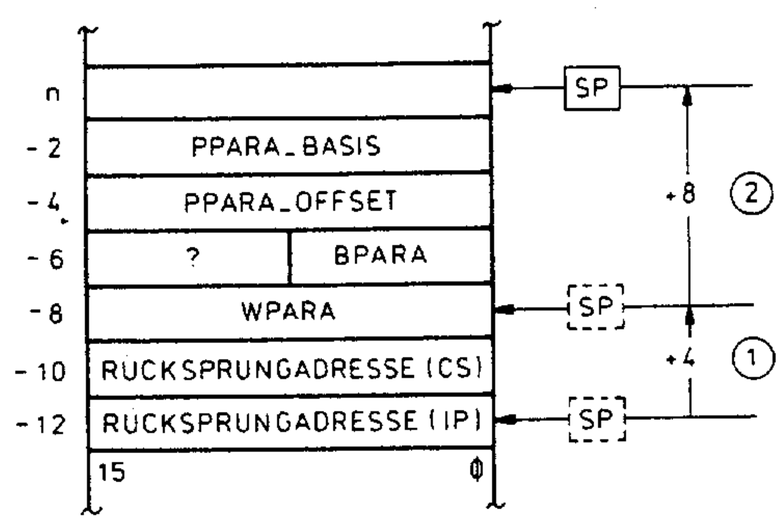
Das zugehörige Stack-Layout zeigt das folgende Bild. Dabei entspricht die Stack-Belegung dem sogenannten **LARGE**-Modell, das manchmal zur Kombination von C- und Assembler-Modulen benützt wird. Bei diesem Modell hat jedes Programm-Modul ein eigenes Code- und Daten-Segment. Dies ist auch der Grund, warum der C-Compiler für *ppara einen kompletten 32-Bit-Zeiger (Basis:Offset) liefert.



Wie zu sehen ist, werden die Parameter von rechts nach links an den Stack übergeben.
Weil sich nach dem LARGE-Modell der Code der Assembler-FAR-Prozedur drei_parameter in einem **anderen** Segment wie der Code des C-Programms aufhält, rettet der Prozessor als Rücksprung-Adresse einen kompletten 32-Bit-Zeiger CS:IP in den Stack, bevor er nach drei_parameter verzweigt.
Führt der Prozessor die FAR-Prozedur drei_parameter aus, trifft er irgendwann auf den RET-Befehl. Dies zeigt das folgende Assembler-Programm-Fragment:

```
drei_parameter PROC    FAR
                 .
                 .
                 .
               RET
```

Der RET-Befehl lädt die komplette 32-Bit-Rücksprung-Adresse in die CPU, inkrementiert den Stack-Pointer SP mit 4 ① und bewirkt, daß der Prozessor im C-Programm beim Befehl `ADD SP,08H` die Programmausführung fortsetzt.



Wird nun `ADD SP,08H` ausgeführt, bedeutet dies, daß der Stack-Pointer auf eine Position unmittelbar jenseits der Parameter gestellt wird ②. Dies bedeutet, daß die Parameter aus dem Stack "entfernt" werden.
Wenn sich der Code der Assembler-Prozedur drei_parameter im gleichen Segment wie der Code des C-Programms aufhält und wenn sich zusätzlich die Variablen des Assembler-Programms im gleichen Daten-Segment wie die Variablen des C-Programms befinden, liegt das sogenannte **SMALL**-Modell vor. Auch dieses Modell wird manchmal zur Kombination von C-Modulen und Assembler-Modulen benützt. In diesem Modell werden Zeiger auf Variablen und Labels als sogenannte **Short-Pointer**, das sind reine 16-Bit-Offsetwerte, an den Stack übergeben. Aus diesem Grund sind alle Prozeduren vom Typ NEAR.

Wenn das obige C-Programm mit

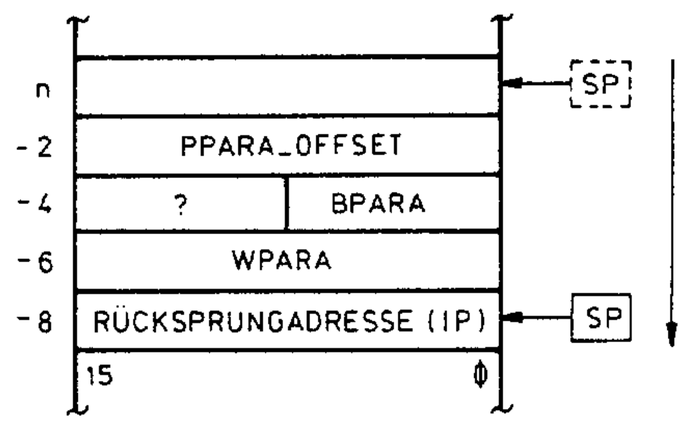
```
>iC86  DAT.C  SMALL  VARPARAMS
```

- VARPARAMS → **Variable** Anzahl von Funktions-Parametern. Sie werden von **rechts** nach **links** an den CPU-Stack übergeben
- SMALL → Übersetzungsmodell SMALL wählen
- DAT.C → DAT.C enthält das C-Quellprogramm
- iC86 → Aufruf des INTEL-C-Compilers

im SMALL-Modell übersetzt wird, dann liefert der C-Compiler einen Code, der im wesentlichen folgendermaßen aussieht:

```
        .
        .
        .
PUSH    ppara_offset        ;Offset-Komponente des Short-Pointer-
                            ;Parameters *ppara
PUSH    bpara               ;Byte-Parameter bpara (Typ char)
PUSH    wpara               ;Word-Parameter wpara (Typ int)
CALL    drei_parameter      ;Aufruf der Assembler-NEAR-Prozedur
ADD     SP,06H              ;Parameter aus dem Stack entfernen
        .
        .
        .
```

Das zugehörige Stack-Layout zeigt das folgende Bild:



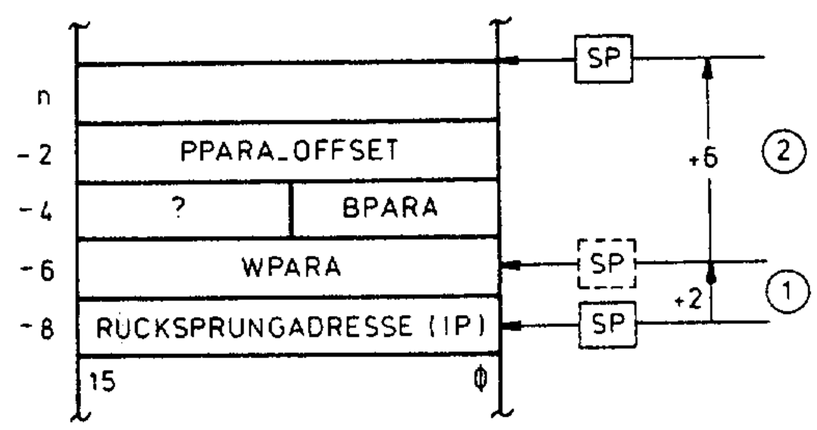
Wie zu sehen ist, werden auch hier gemäß die Parameter von rechts nach links an den Stack übergeben. Dabei wird nur die Offset-Komponente von *ppara gespeichert.

Weil sich der Code der Assembler-NEAR-Prozedur drei_parameter im gleichen Segment wie der Code des C-Programms aufhält, rettet der Prozessor als Rücksprung-Adresse nur den 16-Bit-Zeiger IP in den Stack, bevor er nach drei_parameter verzweigt.

Führt der Prozessor die NEAR-Prozedur drei_parameter aus, trifft er irgendwann auf den RET-Befehl. Dies zeigt das folgende Assembler-Programm-Fragment:

```
drei_parameter PROC    NEAR
                 .
               RET
```

Der RET-Befehl lädt die 16-Bit-Rücksprung-Adresse in die CPU, inkrementiert den Stack-Pointer SP mit 2 ① und bewirkt, daß der Prozessor im C-Programm beim Befehl `ADD SP,06H` die Programmausführung fortsetzt.



Wird nun `ADD SP,06` ausgeführt, bedeutet dies, daß der Stack-Pointer auf eine Position unmittelbar jenseits der Parameter gestellt wird ②. Dies bedeutet, daß die Parameter aus dem Stack "entfernt" werden.

Wenn ein Anwender die C-Compiler iC86, iC286 von INTEL benützt, kann er die Standard-Konventionen der Parameterübergabe, wie sie bei den herkömmlichen Programmiersprachen wie PL/M, PASCAL und FORTRAN üblich ist, mit dem Schlüsselwort aktivieren. Dabei bewirkt alien, daß die Parameter-Übergabe an den Stack in der angegebenen Reihenfolge von **links** nach **rechts** erfolgt. Am Beispiel des bereits benützten C-Programms soll die Anwendung und Wirkung von alien gezeigt werden.

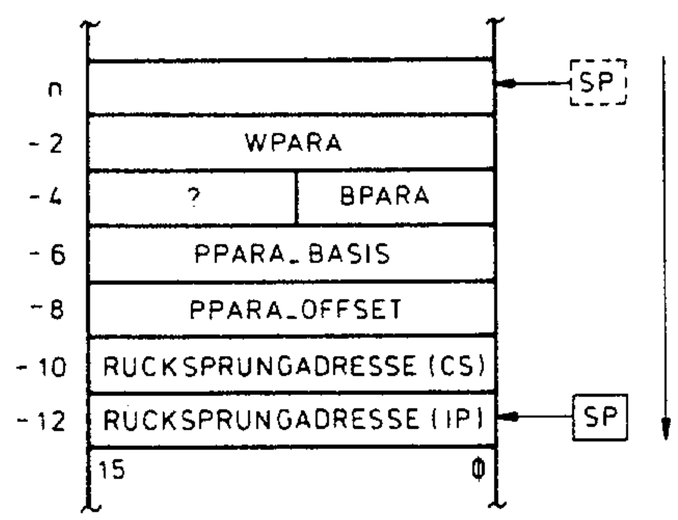
```
int   wpara;
char  bpara;
int   *ppara;
beispiel()
{
alien void drei_parameter ();                /* Funktionsdeklaration */
drei_parameter (wpara,bpara,ppara);          /* Funktionsaufruf */
}
```

Wie zu sehen ist, muß alien **vor** die Funktionsdeklaration plaziert werden. Das Schlüsselwort bewirkt, daß abhängig vom gewählten Übersetzungsmodell (SMALL oder LARGE) der C-Compiler im wesentlichen folgenden Code liefert:

C-Code (LARGE-Modell):

```
    .
    .
    .
PUSH   wpara                ;Word-Parameter wpara (Typ int)
PUSH   bpara                ;Byte-Parameter bpara (Typ char)
PUSH   ppara_basis          ;Basis-Komponente des Long-Pointer-
                            ;Parameters *ppara
PUSH   ppara_offset         ;Offset-Komponente des Long-Pointer-
                            ;Parameters *ppara
CALL   drei_parameter       ;Aufruf der Assembler-FAR-Prozedur
    .
    .
    .
```

Zugehöriges Stack-Layout (LARGE-Modell):



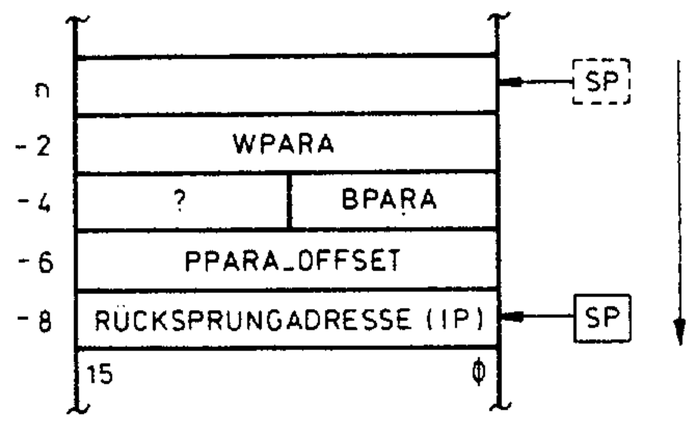
Wie im C-Code zu sehen ist, setzt der C-Compiler wegen `alien` den Befehl `ADD SP,08H` **nicht** ab. Dies bedeutet, daß jetzt die FAR-Prozedur drei_parameter die übergebenen Parameter aus dem Stack entfernen muß.

C-Code (SMALL-Modell):

```
        .
        .
        .
PUSH    wpara                   ;Word-Parameter wpara (Typ int)
PUSH    bpara                   ;Byte-Parameter bpara (Typ char)
PUSH    ppara_offset            ;Offset-Komponente des Short-Pointer-
                                ;Parameters *ppara
CALL    drei_parameter          ;Aufruf der Assembler-NEAR-Prozedur
        .
        .
        .
```

Auch hier ist im C-Code erkennbar, daß der Compiler wegen `alien` den Befehl `ADD SP,06H` **nicht** liefert, so daß in diesem Fall die NEAR-Prozedur drei_parameter die übergebenen Parameter aus dem Stack entfernen muß.

Zugehöriges Stack-Layout (SMALL-Modell):



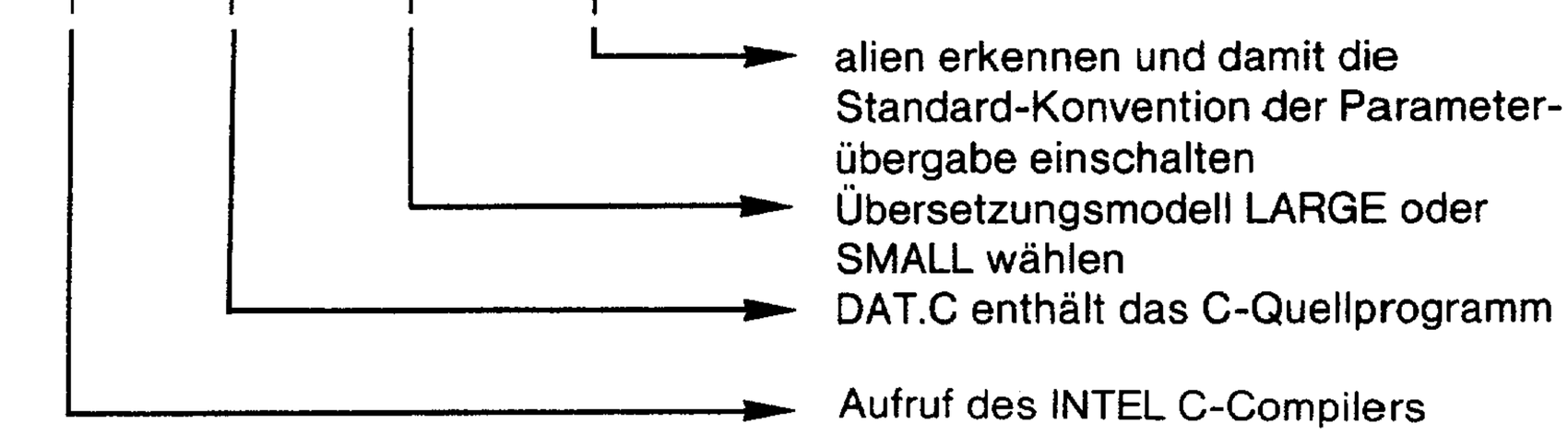
Welche Möglichkeiten der Assembler hat, die nach der Standard-Konvention übergebenen Parameter aus dem Stack zu entfernen, wird in den folgenden Abschnitten erläutert.

Damit der INTEL ANSI-C-Compiler iC86 das Attribut alien im C-Quellprogramm erkennen kann, muß im Compiler-Aufruf der Steuerparameter extend (klein oder groß geschrieben) angegeben werden. Der Aufruf von iC86 sieht dann folgendermaßen aus:

```
>iC86 DAT.C LARGE EXTEND
```

oder

```
>iC86 DAT.C SMALL EXTEND
```

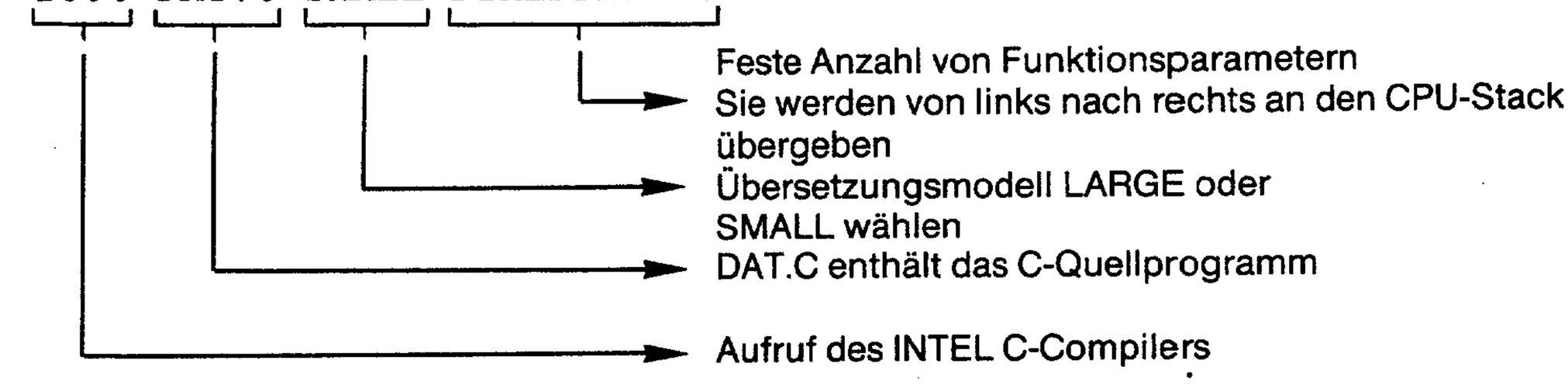


iC86 bietet neben alien eine weitere Möglichkeit, die Standard-Konvention der Parameter-Übergabe zu aktivieren. Dabei deklariert der C-Programmierer die aufzurufende externe NEAR- oder FAR-Prozedur wie **gewöhnlich,** also **ohne** das Attribut alien. Wenn nun die Parameter von **links** nach **rechts** an den Stack übergeben werden sollen, dann kann dies im Compiler-Aufruf durch den Steuerparameter **fixedparams** (klein oder groß geschrieben) angegeben werden. Der Aufruf von iC86 sieht dann folgendermaßen aus.

```
>iC86 DAT.C LARGE FIXEDPARAMS
```

oder

```
>iC86 DAT.C SMALL FIXEDPARAMS
```



Hinweis: Von den beiden Spezifikationen fixedparams und varparams ist fixedparams voreingestellt (default) und kann deshalb auch weggelassen werden.

6.5 Rückgewinnung der Parameter vom Stack

Ein Assembler-Programm, das von einem C-Programm aufgerufen wird, kann auf die in den Stack übergebenen Parameter in folgender Weise zugreifen:

- Mit Hilfe von POP-Befehlen wird jeder Parameter entweder in ein Register oder in eine lokale Variable zurückgeschrieben. Wird diese Methode angewendet, ist zunächst die Rücksprungadresse zwischenzeitlich z.B. in zwei CPU-Registern aufzubewahren.
- Anschließend können mittels POP-Befehlen die Parameter vom Stack gelesen werden. Danach ist die aufbewahrte Rücksprungadresse in den Stack zurückzuschreiben.

Die angegebene sogenannte **Pop-Methode** erlaubt nun einen 'normalen' RET-Befehl (ohne Operand), da vor seiner Ausführung die Parameter bereits aus dem Stack 'entfernt' wurden.

Das angegebene Programmierbeispiel zeigt die Anwendung der beschriebenen Technik. Es sei vorausgesetzt, daß die in Assembler geschriebene Prozedur drei_parameter von C aus nach der fixedparams-Konvention mit drei_parameter (wpara, bpara, ppara) aufgerufen wird.

```
LOC  OBJ                    LINE     SOURCE

                               1             NAME    BEISPIEL_57
                               2
                               3     ASSUME CS:CODE_PROG,DS:DATEN_PROG
                               4
----                           5     DATEN_PROG SEGMENT 'RAM_REGION'
                               6
0000 (1                        7     POINTER_PARA    DD      1 DUP (?)
     ????????
     )
                               8
----                           9     DATEN_PROG ENDS
                              10
----                          11     STACK_PROG SEGMENT 'RAM_REGION'
                              12
0000 (10                      13                     DW      10 DUP (?)
     ????
     )
                              14
0014                          15     TOP_STACK LABEL WORD
                              16
----                          17     STACK_PROG ENDS
                              18
----                          19     CODE_PROG SEGMENT 'ROM_REGION'
                              20
0000 B8----          R        21     START:  MOV     AX,DATEN_PROG          ;DS Register initialisieren
0003 8ED8                     22             MOV     DS,AX
0005 B8----          R        23             MOV     AX,STACK_PROG          ;SS Register initialisieren
0008 8ED0                     24             MOV     SS,AX
000A BC1400                   25             MOV     SP,OFFSET TOP_STACK    ;SP Register initialisieren
                              26
```

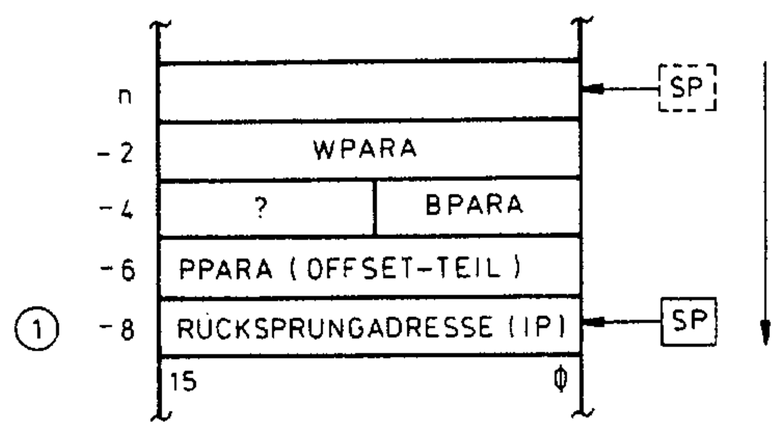
```
LOC  OBJ                LINE    SOURCE

000D                     27     DREI_PARAMETER PROC FAR
                         28
000D 5E                  29             POP     SI                      ;(SI) = Ruecksprungadresse (IP)
000E 5F                  30             POP     DI                      ;(DI) = Ruecksprungadresse (CS)
000F 8F060000            31             POP     WORD PTR POINTER_PARA   ;POINTER_PARA bekommt
                         32                                             ;Pointer Parameter (Offset Teil)
0013 8F060200            33             POP     WORD PTR POINTER_PARA+2 ;POINTER_PARA+2 bekommt
                         34                                             ;Pointer Parameter (Basis Teil)
0017 58                  35             POP     AX                      ;(AL) = Byte Parameter
0018 59                  36             POP     CX                      ;(CX) = Word Parameter
0019 57                  37             PUSH    DI                      ;Schreibe Ruecksprungadresse
001A 56                  38             PUSH    SI                      ;zurueck in den Stack
                         39                     ;
                         40                     ;
                         41                     ;
                         42             ;(Prozedurkoerper)
                         43                     ;
                         44                     ;
                         45                     ;
001B CB                  46             RET                             ;'Normaler' RET
                         47
                         48     DREI_PARAMETER ENDP
                         49
----                     50     CODE_PROG ENDS
                         51
                         52             END     START
```

Die ersten beiden Befehle in der Prozedur retten die Rücksprungadresse in die SI- und DI-Register. Der Long-Pointer-Parameter ppara wird anschließend in die Double-Word-Variable POINTER_PARA geschrieben.

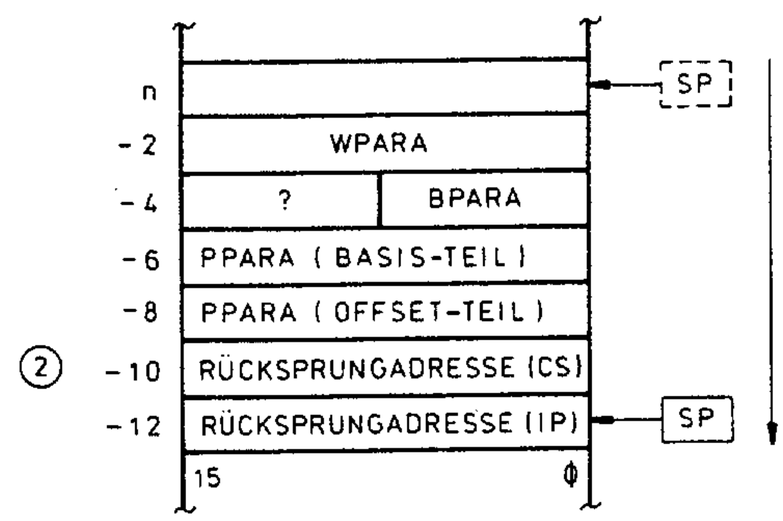
Danach wird das Wort, das bpara enthält nach AX (bpara nach AL) und wpara nach CX übertragen. Schießlich wird vor der Ausführung der eigentlichen Prozedur (Prozedurkörper) die Rücksprungadresse in den Stack zurückgeschrieben.

Eine andere Methode, Parameter vom Stack zurückzubekommen, beruht auf der Idee, die Parameter **BP-relativ** zu adressieren. Bei dieser Technik werden die im Stack hinterlegten Parameter als Datenstruktur betrachtet. Sie sind adressierbar, wenn der Zeiger SS:BP zum Anfang dieser Datenstruktur zeigt und ihre Offsets relativ zu BP benützt werden.

Zur Erläuterung dieser sogenannten **BP-Methode** soll die Assembler-Prozedur drei_parameter existieren. Sie wird mit der C-Anweisung drei_parameter (wpara, bpara, ppara) nach Standard-Konvention aufgerufen. Ist die Prozedur vom Typ **NEAR**, ergibt sich nach dem Aufruf eine Stackbelegung, die folgendem Bild entspricht:



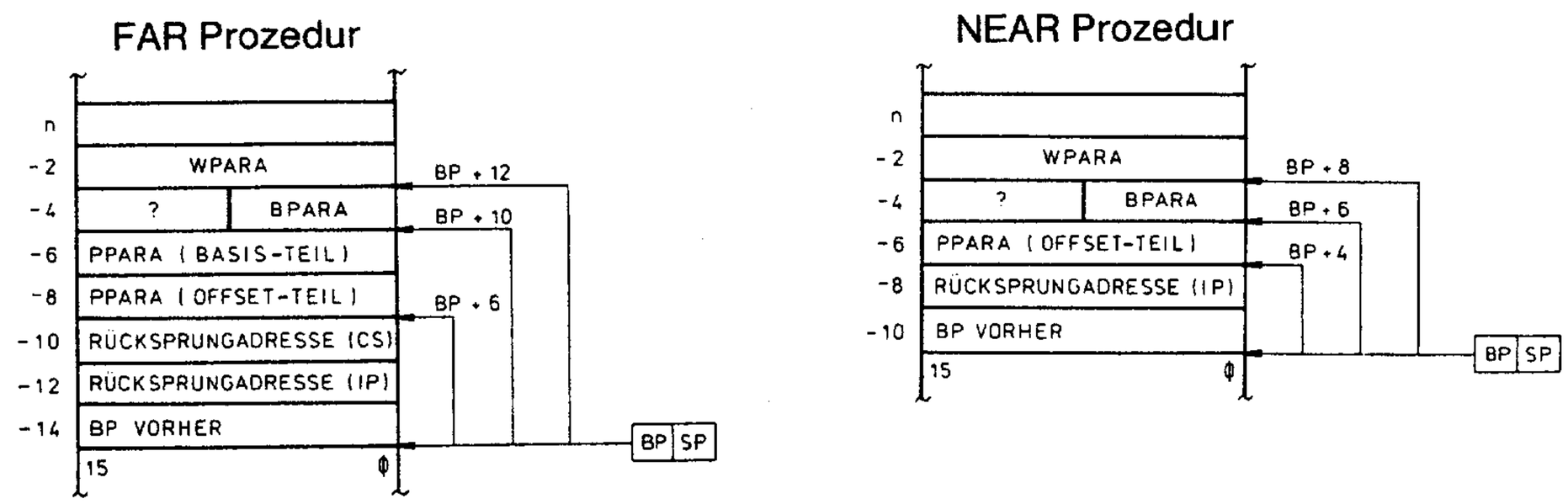
Als Rücksprungadresse wird nur IP im Stack hinterlegt ①. Ist die Prozedur vom Typ **FAR** wird neben IP auch CS im Stack hinterlegt ②, so daß sich nach dem Aufruf folgende Belegung im Stack ergibt.



Im **Prolog** von drei_parameter (die ersten beiden Befehle) wird zunächst mit `PUSH BP` das BP-Register in den Stack gerettet und anschließend mit `MOV BP, SP` das BP-Register mit dem augenblicklichen Stackpointer geladen.

```
DREI_PARAMETER PROC FAR      oder      DREI_PARAMETER PROC NEAR

Prolog { PUSH BP                       Prolog { PUSH BP
         MOV BP, SP                             MOV BP, SP
```

Danach zeigen sowohl SP, als auch BP auf den letzten Eintrag (BP vorher) im Stack



Wie nun die Parameter im Stack zu erreichen sind, zeigen folgende Code-Fragmente.

```
DREI_PARAMETER PROC FAR          oder      DREI_PARAMETER PROC NEAR
     .                                          .
     .                                          .
MOV AL, [BP+10] ;Param. bpara              MOV AL, [BP+6] ;Param. bpara
LES BX, [BP+6]  ;Param. ppara              MOV BX, [BP+4] ;Param. ppara
SUB CX, [BP+12] ;Param. wpara              SUB CX, [BP+8] ;Param. wpara
     .                                          .
     .                                          .
```

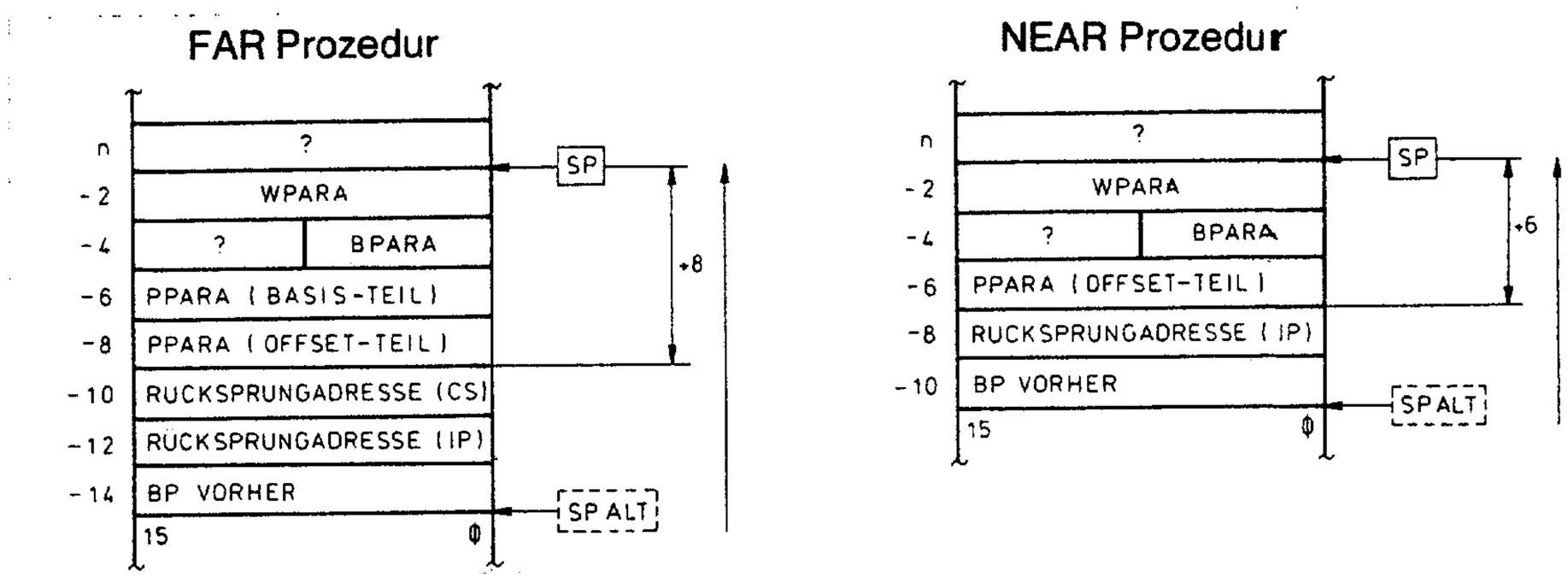
Wie die Befehle zeigen, sind die Parameter adressierbar, wenn ihre Offsets relativ zu BP benützt werden.

Im **Epilog** von DREI_PARAMETER (die letzten beiden Befehle) wird zunächst mit `POP BP` das BP-Register mit seinem vorherigen Wert geladen und anschließend werden mit `RET 8` in der FAR-Prozedur bzw. `RET 6` in der NEAR-Prozedur die Parameter aus dem Stack "entfernt".

Die Befehle `RET 8` bzw. `RET 6` bewirken in der fixedparams-Konvention das gleiche, wie die Befehle `ADD SP, 08H` bzw. `ADD SP, 06H`, die in der **vorparams**-Konvention der vom Compiler automatisch abgesetzt werden. Dies bedeutet, daß der Prozessor die RET-Operanden 8 bzw. 6 zum Stackpointer addiert und somit die Parameter aus dem Stack "entfernt". Der Wert im SP-Register entspricht wieder dem Wert vor dem Aufruf der Prozedur, so daß nun die Parameter jenseits des Stackpointers liegen.

```
DREI_PARAMETER PROC FAR      oder      DREI_PARAMETER PROC NEAR
                .                                .
                .                                .
                .                                .
        Epilog { POP BP                  Epilog { POP BP
                 RET 8                            RET 6
DREI_PARAMETER ENDP                      DREI_PARAMETER_ENDP
```

Das angegebene Programmlisting zeigt vollständig die **BP-Methode** am Beispiel der FAR Prozedur DREI_PARAMETER.




```
LOC  OBJ                    LINE     SOURCE

                              1              NAME    BEISPIEL_58
                              2
                              3      ASSUME CS:CODE_PROG
                              4
----                          5      STACK_PROG SEGMENT 'RAM_REGION'
                              6
0000 (20                      7              DW      20 DUP (?)
     ????
     )
0028                          8      TOP_STACK LABEL WORD
                              9
----                         10      STACK_PROG ENDS
                             11
----                         12      CODE_PROG SEGMENT 'ROM_REGION'
                             13
0000 B8----          R       14      START:  MOV     AX,STACK_PROG          ;SS Register initialisieren
0003 8ED0                    15              MOV     SS,AX
0005 BC2800                  16              MOV     SP,OFFSET TOP_STACK    ;SP Register initialisieren
                             17
0008                         18      DREI_PARAMETER PROC FAR
                             19
0008 55                      20              PUSH    BP                     ;Rette BP des aufrufenden Programms
0009 8BEC                    21              MOV     BP,SP                  ;Stelle BP an die Spitze des Stack
                             22                      ;
                             23                      ;
                             24              ;(Prozedurkoerper)
                             25                      ;
                             26                      ;
000B 8A460A                  27              MOV     AL,[BP+10]             ;(AL) = Byte Parameter BPARA
000E C45E06                  28              LES     BX,DWORD PTR[BP+6]     ;Lade ES:BX mit Long Pointer
                             29                                             ;(ES) = PPARA (Basis Teil)
                             30                                             ;(BX) = PPARA (Offset Teil)
0011 2B4E0C                  31              SUB     CX,[BP+12]             ;(CX) = (CX)-Word Parameter WPARA
                             32                      ;
                             33                      ;
                             34              ;(Prozedurkoerper)
                             35                      ;
                             36                      ;
0014 5D                      37              POP     BP                     ;Lade BP mit dem alten Wert
0015 CA0800                  38              RET     8                      ;'Entferne' 8 Bytes beim Rueck-
                             39                                             ;sprung zum aufrufenden Programm
                             40
                             41      DREI_PARAMETER ENDP
                             42
----                         43      CODE_PROG ENDS
                             44
                             45              END     START
```

Die Methoden zur Rückgewinnung der Parameter vom Stack können sinngemäß auch für die Parameter benützt werden, die nach vorparams-Konvention von rechts nach links an den Stack übergeben wurden.

6.6 Auswahlkriterien der Methode bei Parameterzugriffen

Steht eine Assembler-Prozedur zur Verfügung, die durch eine C-Anweisung aufgerufen wird, muß für die Parameterzugriffe entweder die **POP-Methode** oder die BP-Methode gewählt werden. Welche Methode gewählt wird, ist abhängig von der Natur der Prozedur.

- Die POP-Methode kann am effektivsten sein, wenn **alle** Parameter in die CPU-Register übertragen werden, da Registerzugriffe wesentlich schneller als Speicherzugriffe sind.
- Aus diesem Grund ist es sinnvoll, zunächst die POP-Methode für kurze Prozeduren und einige wenige Parameter zu betrachten.

Es gibt aber Faktoren, die die POP-Methode uneffektiv machen können. Sind z.B. viele Parameter vom Stack zu lesen; dies geschieht mit einer Sequenz von POP-Befehlen am Kopf der Prozedur, sind einige Register, da sie Parameter enthalten, nicht mehr benützbar. Die POP-Methode sollte also nicht verwendet werden, wenn der Registerraum für Zwischenergebnisse z.B. bei fortgesetzten Berechnungen benötigt wird.

Der große Vorteil der BP-Methode gegenüber der POP-Methode ist, daß die Parameter unverändert im Stack bleiben können, so daß beliebig oft zugegriffen werden kann. Die POP-Methode ist am effektivsten, wenn auf jeden Parameter nur **einmal** zugegriffen und durch anschließende Codesequenzen **nichts** verändert wird.

6.7 Definition von Symbolen für Parameter im Stack

Für die im Stack abgelegten Parameter können mit Hilfe der EQU-Direktive Namen vergeben werden. So benützt die im Modul BEISPIEL_59 angegebene FAR-Prozedur DREI_PARAMETER die **BP-Methode**, um den Byte-Parameter BPARA, den Word-Parameter WPARA und Long-Pointer-Parameter PPARA zu adressieren. Für diese Parameter können gemäß folgender Definitionen Symbole vergeben werden.

```
BPARA  EQU   [BP+10]
WPARA  EQU   [BP+12]
PPARA  EQU   DWORD PTR [BP+6]
```

Diese Symbole können nun verwendet werden, um die im Stack hinterlegten Parameter zu adressieren.

```
LOC  OBJ                    LINE     SOURCE

                               1             NAME    BEISPIEL_59
                               2
                               3     ASSUME CS:CODE_PROG
                               4
----                           5     STACK_PROG SEGMENT 'RAM_REGION'
                               6
0000 (20                       7             DW      20 DUP (?)
     ????
     )
0028                           8     TOP_STACK LABEL WORD
                               9
----                          10     STACK_PROG ENDS
                              11
----                          12     CODE_PROG SEGMENT 'ROM_REGION'
                              13
0000 B8----          R        14     START:  MOV     AX,STACK_PROG           ;SS Register initialisieren
0003 8ED0                     15             MOV     SS,AX
0005 BC2800                   16             MOV     SP,OFFSET TOP_STACK     ;SP Register initialisieren
                              17
0008                          18     DREI_PARAMETER PROC FAR
                              19
  000A[]                      20     BPARA   EQU     [BP+10]
  000C[]                      21     WPARA   EQU     [BP+12]
  0006[]                      22     PPARA   EQU     DWORD PTR[BP+6]
                              23
0008 55                       24             PUSH    BP                      ;Rette BP des aufrufenden Programms
0009 8BEC                     25             MOV     BP,SP                   ;Stelle BP an die Spitze des Stack
                              26                     ;
                              27                     ;
                              28             ;(Prozedurkoerper)
                              29                     ;
                              30                     ;
000B 8A460A                   31             MOV     AL,BPARA                ;(AL) = Byte Parameter BPARA
000E C45E06                   32             LES     BX,PPARA                ;Lade ES:BX mit Long Pointer
                              33                                             ;(ES) = PPARA (Basis Teil)
                              34                                             ;(BX) = PPARA (Offset Teil)
0011 2B4E0C                   35             SUB     CX,WPARA                ;(CX) = (CX)-Word Parameter WPARA
                              36                     ;
                              37                     ;
                              38             ;(Prozedurkoerper)
                              39                     ;
                              40                     ;
0014 5D                       41             POP     BP                      ;Lade BP mit dem alten Wert
0015 CA0800                   42             RET     8                       ;'Entferne' 8 Bytes beim Rueck-
                              43                                             ;sprung zum aufrufenden Programm
                              44
                              45     DREI_PARAMETER ENDP
                              46
----                          47     CODE_PROG ENDS
                              48
                              49             END     START
```

Da nun für die Parameter Namen existieren, ist der Programmcode leichter zu verstehen, zumal die vergebenen Symbole den Typ der Parameter in gewisser Weise beschreiben.

In jedem Fall ist ein Code bequemer zu lesen, wenn z.B. statt des Befehls `MOV AL, [BP+10]` mit anonymem Zugriff der selbstdokumentierende Befehl `MOV AL, BPARA` benützt wird.

6.8 Anwendungsbeispiel 1

Gegeben sind zwei Quelldateien, in denen die Module BEISPIEL_59_1 und BEISPIEL_59_2 gespeichert sind. Der Code zeigt verschiedene Techniken, die typischerweise bei Assembler-Prozeduren benützt werden.

Der Hauptmodul BEISPIEL_59_1 enthält ein Programm, das die externe Prozedur UMWANDLUNG **zweimal** aufruft. Die Prozedur UMWANDLUNG ist im Modul BEISPIEL_59_2 gespeichert.

Im DATEN_PROG-Segment des Hauptmoduls sind zwei Felder FELD_1 und FELD_2 unterschiedlicher Länge definiert.

Vor jedem Aufruf der Prozedur werden zwei Parameter an den Stack übergeben.

1. Aufruf: Parameter 1 – die Anzahl der Feldelemente von FELD_1
Parameter 2 – den Offset des 1ten Elements in FELD_1

2. Aufruf: Parameter 1 – die Anzahl der Feldelemente von FELD_2
Parameter 2 – den Offset des 1ten Elements in FELD_2.

```
LOC  OBJ                    LINE     SOURCE

                              1              NAME    BEISPIEL_59_1
                              2
                              3      ASSUME CS:CODE_PROG,DS:DATEN_PROG,SS:STACK_PROG
                              4
----                          5      STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
                              6
0000 (10                      7              DW      10 DUP (?)
     ????
     )
0014                          8      TOP_STACK LABEL WORD
                              9
----                         10      STACK_PROG ENDS
                             11
----                         12      DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                             13
0000 00                      14      FELD_1  DB      0,1,2,3,4,5,6,7,8,9            ;Feld mit 10 Elementen
0001 01
0002 02
0003 03
0004 04
0005 05
0006 06
0007 07
0008 08
0009 09
000A 00                      15      FELD_2  DB      0,1,2,3,4                      ;Feld mit 5 Elementen
000B 01
000C 02
000D 03
000E 04
                             16
----                         17      DATEN_PROG ENDS
                             18
```

```
LOC  OBJ                  LINE     SOURCE

----                        19     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                            20
                            21     EXTRN UMWANDLUNG:FAR
                            22
0000 B8----          R      23     START:  MOV     AX,DATEN_PROG                  ;DS Register initialisieren
0003 8ED8                   24             MOV     DS,AX
0005 B8----          R      25             MOV     AX,STACK_PROG                  ;SS Register initialisieren
0008 8ED0                   26             MOV     SS,AX
000A BC1400          R      27             MOV     SP,OFFSET TOP_STACK            ;SP Register initialisieren
                            28
                            29     ;Aufruf der Prozedur UMWANDLUNG mit einer Feldlaenge von 10 Bytes************
                            30
000D B80A00                 31             MOV     AX,SIZE FELD_1                 ;AX = Anzahl der Elemente von
                            32                                                    ;FELD_1 (Feldlaenge 10 Bytes)
0010 50                     33             PUSH    AX                             ;Parameter1 in den Stack
                            34                                                    ;schreiben
0011 B80000          R      35             MOV     AX,OFFSET FELD_1               ;AX = Position des 1ten
                            36                                                    ;Elements von FELD_1
0014 50                     37             PUSH    AX                             ;Parameter2 in den Stack
                            38                                                    ;schreiben
0015 9A0000----      E      39             CALL    UMWANDLUNG
                            40
                            41     ;Aufruf der Prozedur UMWANDLUNG mit einer Feldlaenge von 5 Bytes*************
                            42
001A B80500                 43             MOV     AX,SIZE FELD_2                 ;AX = Anzahl der Elemente von
                            44                                                    ;FELD_2 (Feldlaenge 5 Bytes)
001D 50                     45             PUSH    AX                             ;Parameter1 in den Stack
                            46                                                    ;schreiben
001E B80A00          R      47             MOV     AX,OFFSET FELD_2               ;AX = Position des 1ten
                            48                                                    ;Elements von FELD_2
0021 50                     49             PUSH    AX                             ;Parameter2 in den Stack
                            50                                                    ;schreiben
0022 9A0000----      E      51             CALL    UMWANDLUNG
                            52
----                        53     CODE_PROG ENDS
                            54
                            55             END     START
```

6.8.1 Der SIZE-Operator

Damit vor jedem Aufruf der Prozedur UMWANDLUNG eine unterschiedliche Feldlänge an den Stack übergeben werden kann, stellt der Assembler den **SIZE**-Operator zur Verfügung.

Der SIZE-Operator liefert die Anzahl der **Bytes**, die einer Variablen zugewiesen wurden.

Syntax:

```
SIZE   Variable
```

Einige Beispiele:

```
LOC  OBJ                    LINE     SOURCE

                              1              NAME    BEISPIEL_SIZE
                              2
                              3     ASSUME CS:CODE_PROG,DS:DATEN_PROG
                              4
----                          5     DATEN_PROG SEGMENT
                              6
0000 (150                     7     EIN_WORD_FELD   DW      150 DUP (?)            ;SIZE = 300
     ????
     )
012C 01                       8     EIN_BYTE_FELD   DB      1,2,3,4,5,6,7          ;SIZE = 7
012D 02
012E 03
012F 04
0130 05
0131 06
0132 07
0133 07                       9     INIT_SIZE       DB      SIZE EIN_BYTE_FELD     ;INIT_SIZE ist mit 7
                             10                                                    ;initialisiert
                             11
----                         12     DATEN_PROG ENDS
                             13
----                         14     CODE_PROG SEGMENT
                             15
0000 B8----          R       16     START:  MOV     AX,DATEN_PROG                  ;DS Register initialisieren
0003 8ED8                    17             MOV     DS,AX
0005 B82C01                  18             MOV     AX,SIZE EIN_WORD_FELD          ;AX = 300
                             19
----                         20     CODE_PROG ENDS
                             21
                             22             END     START
```

Wird der SIZE-Operator auf die Feldvariablen FELD_1 und FELD_2 im Anwendungsbeispiel 1 angewendet, dann liefert der Ausdruck `SIZE FELD_1` den Wert 10 und der Ausdruck `SIZE FELD_2` den Wert 5. Damit sind die unterschiedlichen Längen beider Felder bestimmt. Sie werden vor dem Prozeduraufruf mit `PUSH AX` an den Stack übergeben.

6.8.2 Der LENGTH-Operator

Der LENGTH-Operator liefert die Anzahl der Dateneinheiten, die einer Variablen zugewiesen wurden.

Syntax:

```
LENGTH   Variable
```

Dabei entspricht die Dateneinheit dem Typ der Variablen, also BYTE, WORD oder DWORD. D.h. der LENGTH-Operator liefert die Anzahl von Bytes, Words oder Double Words, in einer Variablen.

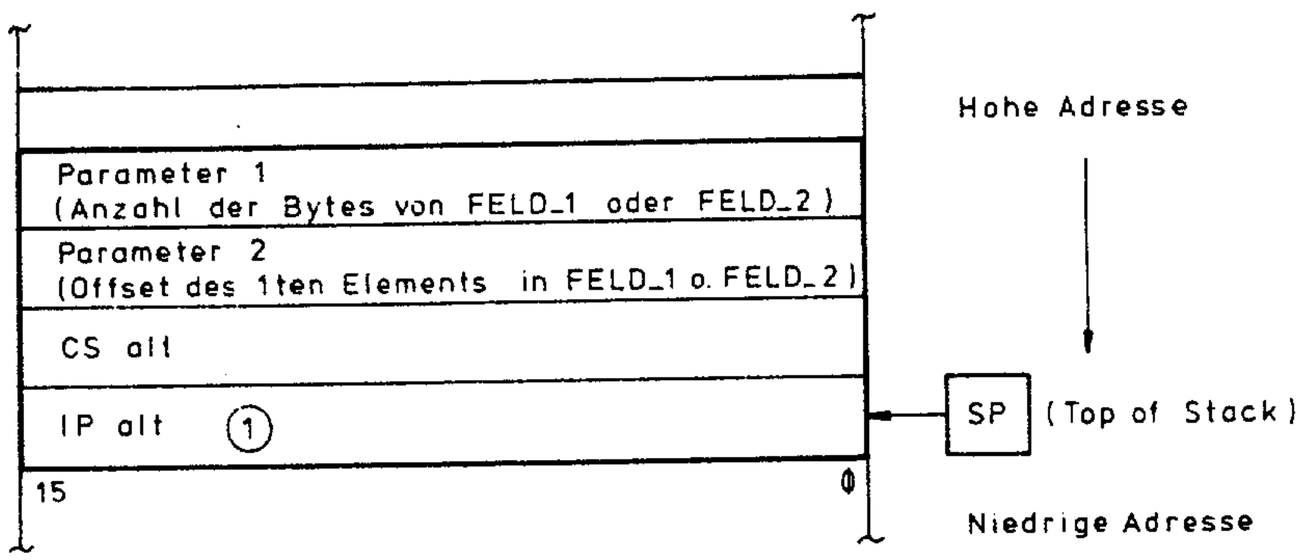
Einige Beispiele:

```
LOC  OBJ                    LINE     SOURCE
                              1              NAME    BEISPIEL_LENGTH
                              2
                              3      ASSUME CS:CODE_PROG,DS:DATEN_PROG
                              4
----                          5      DATEN_PROG SEGMENT
                              6
0000 (150                     7      EIN_WORD_FELD   DW      150 DUP (?)            ;LENGTH = 150
     ????
     )
012C 01                       8      EIN_BYTE_FELD   DB      1,2,3,4,5,6,7          ;LENGTH = 7
012D 02
012E 03
012F 04
0130 05
0131 06
0132 07
0133 96                       9      INIT_LENGTH     DB      LENGTH EIN_WORD_FELD   ;INIT_LENGTH ist mit 150
                             10                                                    ;initialisiert
                             11
----                         12      DATEN_PROG ENDS
                             13
----                         14      CODE_PROG SEGMENT
                             15
0000 B8----          R       16      START:  MOV     AX,DATEN_PROG                 ;DS Register initialisieren
0003 8ED8                    17              MOV     DS,AX
0005 B89600                  18              MOV     AX,LENGTH EIN_WORD_FELD       ;AX = 150
                             19
----                         20      CODE_PROG ENDS
                             21
                             22              END     START
```

Wird der LENGTH-Operator auf die Feldvariablen FELD_1 und FELD_2 im Anwendungsbeispiel 1 angewendet, dann liefert der Ausdruck `LENGTH FELD_1` den Wert 10 und der Ausdruck `LENGTH FELD_2` den Wert 5. Da die Feldvariablen FELD_1 und FELD_2 vom Typ BYTE sind, liefern in diesem Fall die LENGTH- und SIZE-Operatoren die gleichen Werte.

Hinweis: Der LENGTH-Operator ist nützlich, um einen Zähler für Programmwiederholungen zu setzen. Als Anwendung ist dafür z.B. der Zugriff auf die Elemente eines Feldes denkbar.

Nachdem mit `CALL UMWANDLUNG` die Prozedur aufgerufen worden ist, ergibt sich folgende Stackbelegung:



Da die externe Prozedur vom Typ FAR vereinbart ist, muß das aufrufende Programm einen "Intersegment-CALL" benützen, um die Prozedur zu aktivieren. Dies bedeutet, daß sowohl CS als auch IP in den Stack gerettet werden ①.

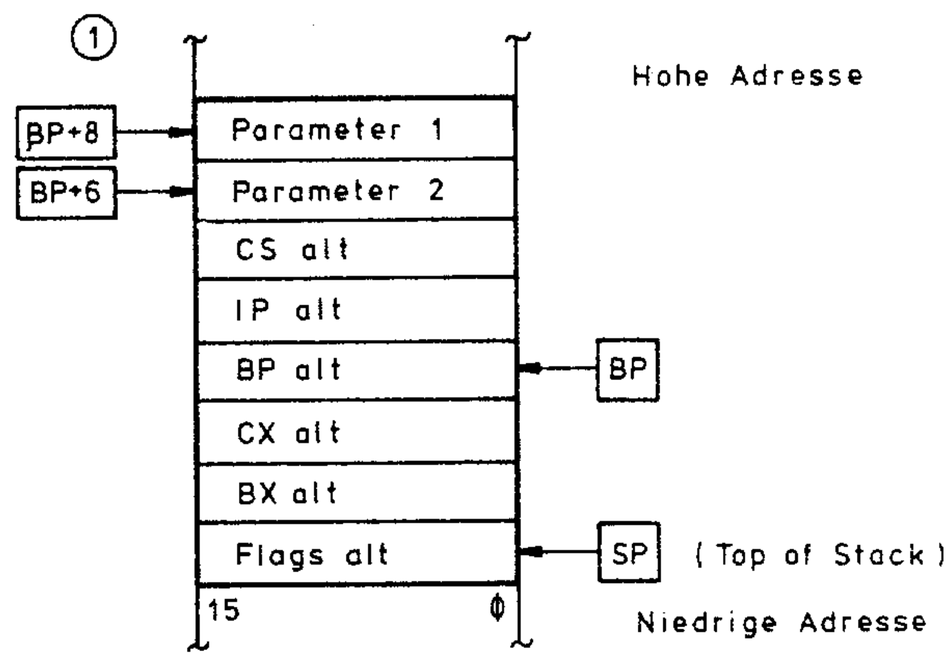
Die Prozedur UMWANDLUNG ist im Modul BEISPIEL_59_2 gespeichert. Sie ist in vier Sektionen unterteilt.

```
LOC  OBJ                LINE     SOURCE

                         1             NAME    BEISPIEL_59_2
                         2
                         3     ASSUME CS:CODE_PROG_1
                         4
----                     5     CODE_PROG_1 SEGMENT BYTE PUBLIC 'ROM_REGION'
                         6
                         7     PUBLIC UMWANDLUNG
                         8
0000                     9     UMWANDLUNG      PROC FAR
                        10
  0008[]                11     FELDLAENGE      EQU [BP+8]
  0006[]                12     ELEMENT_1       EQU [BP+6]
                        13
                        14     ;Prozedur Prolog***********************
                        15
0000 55                 16                     PUSH    BP                  ;Rette Register BP
0001 8BEC               17                     MOV     BP,SP               ;Setze Base Pointer
0003 51                 18                     PUSH    CX                  ;Rette Register und Flags
                        19                                                 ;des aufrufenden Programms
0004 53                 20                     PUSH    BX
0005 9C                 21                     PUSHF
                        22     ;Ende des Prolog***********************
                        23
                        24
                        25     ;Prozedurkoerper***********************
0006 8B4E08             26                     MOV     CX,FELDLAENGE       ;CX = Feldlaenge
0009 8B5E06             27                     MOV     BX,ELEMENT_1        ;Offset des 1ten Feldelements
000C 800730             28     ASCII:          ADD     BYTE PTR[BX],30H    ;Wandle Feldelement in
                        29                                                 ;ASCII um
000F 43                 30                     INC     BX                  ;Bilde Feldindizes
0010 E2FA               31                     LOOP    ASCII               ;Wiederhole bis alle
                        32                                                 ;Feldelemente umgewandelt sind
                        33     ;Ende Prozedurkoerper******************
                        34     ;
                        35     ;
                        36     ;Prozedur Epilog***********************
0012 9D                 37                     POPF                        ;Schreibe Register und Flags
0013 5B                 38                     POP     BX                  ;zurueck an das aufrufende
0014 59                 39                     POP     CX                  ;Programm
0015 5D                 40                     POP     BP
                        41     ;Ende Epilog***************************
                        42
0016 CA0400             43                     RET     4                   ;Entferne 2 Parameter
                        44
                        45     UMWANDLUNG ENDP
                        46
----                    47     CODE_PROG_1 ENDS
                        48
                        49             END
```

Im **Prozedur-Prolog** wird das BP-Register so gesetzt, daß es benützt werden kann, um die Daten im Stack zu adressieren. BP ist dabei automatisch als Base-Register für die Adressierung von Daten im Stack spezifiziert.

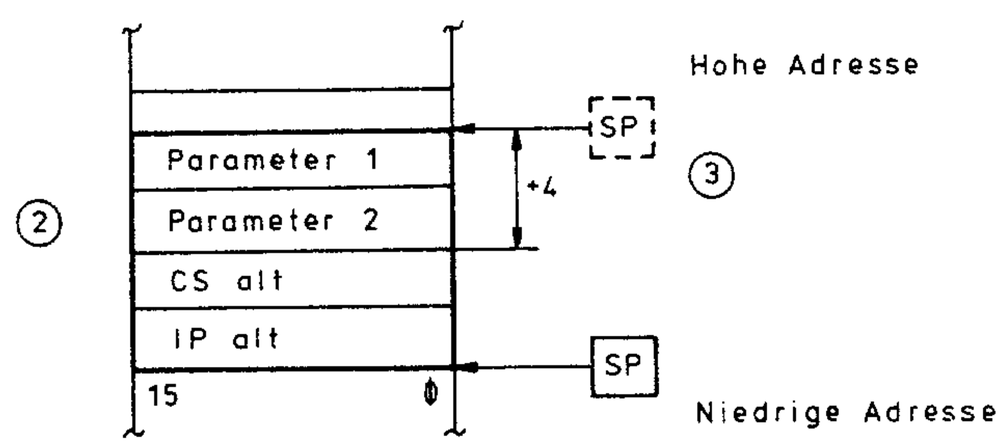
Als nächstes wird im Prolog der "Maschinenstatus" gerettet. Dies sind notwendigerweise die Register CX, BX und die Flags, da ihre Zustände im Prozedurkörper geändert werden.

Am Ende des Prozedurprologs ergibt sich folgendes Stack-Layout:



Im **Prozedurkörper** wird mit `MOV CX, [BP+8]` und `MOV BX, [BP+6]` auf die an den Stack übergebenen Parameter 1 (Feldlänge) und Parameter 2 (Position des 1ten Feldelements) "anonym" zugegriffen ①. Dabei wurden für die Pointer [BP+8] und [BP+6] die Symbole FELDLAENGE und ELEMENT_1 vergeben, um Parameter 1 und Parameter 2 auch mit Namen zu kennzeichnen. Als weiterer Schritt im Prozedurkörper werden die ungepackten Dezimalzahlen in FELD_1 oder FELD_2 **komplett** in ASCII-Zeichen umgewandelt.

Der **Prozedur-Epilog** dreht die Aktivitäten des Prolog um. Der gerettete "Maschinenstatus" wird an das aufrufende Programm zurückgeschrieben. Am Ende ergibt sich folgendes Stack-Layout ②:



In der vierten Sektion der Prozedur werden mit `RET 4` zunächst IP und CS an das aufrufende Programm zurückgeschrieben und anschließend Parameter 1 und Parameter 2 aus dem Stack "entfernt" ③. Der Stack ist nun wieder im gleichen Zustand wie **vor** der Parameterübergabe.

6.9 Anwendungsbeispiel 2

Das angegebene Beispiel enthält die NEAR-Prozedur MAXIMUM, in der eine Assembler-Datenstruktur definiert ist. Diese Struktur wird benützt, um den Stack zu adressieren.

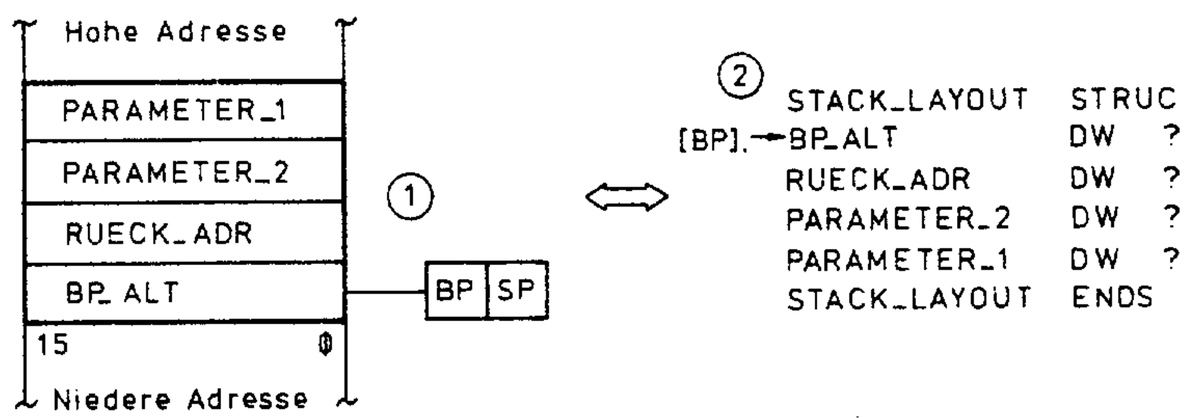
Um die Strukturelemente zu erreichen, wird BP als Basisregister spezifiziert: `[BP].Struktur-Element`. Danach stellt der Offset eines Strukturelements den relativen Abstand zum Anfang der Datenstruktur dar.

```
LOC  OBJ                  LINE     SOURCE

                           1              NAME    BEISPIEL_59_3
                           2
                           3     ASSUME CS:CODE_PROG,DS:DATEN_PROG
                           4
----                       5     DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                           6
0000 ????                  7     PARAMETER1      DW      ?
0002 ????                  8     PARAMETER2      DW      ?
0004 ????                  9     MAXIMALER_WERT  DW      ?
                          10
----                      11     DATEN_PROG ENDS
                          12
----                      13     CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                          14
0000 FF360000       R     15     START:          PUSH    PARAMETER1      ;Aufruf der Prozedur MAXIMUM
0004 FF360200       R     16                     PUSH    PARAMETER2      ;mit der angegebenen
0008 E80000               17                     CALL    MAXIMUM         ;Befehlssequenz
                          18
000B                      19     MAXIMUM         PROC
                          20
                          21     ;Definition des Stack Layout als Datenstruktur***************
                          22
----                      23     STACK_LAYOUT STRUC
                          24
0000                      25     BP_ALT          DW      ?                       ;Basis der Datenstruktur
0002                      26     RUECK_ADR       DW      ?                       ;Rueckspringadresse IP
0004                      27     PARAMETER_2     DW      ?                       ;2.  Parameter (PARAMETER2)
0006                      28     PARAMETER_1     DW      ?                       ;1. Parameter (PARAMETER1)
                          29
----                      30     STACK_LAYOUT ENDS
                          31
                          32     ;Prozedur Prolog*********************************************
000B 55                   33                     PUSH    BP                      ;Rette BP alt
000C 8BEC                 34                     MOV     BP,SP                   ;Markiere BP alt
                          35
                          36     ;Prozedurkoerper*********************************************
000E 8B4606               37                     MOV     AX,[BP].PARAMETER_1     ;Wenn der erste Parameter
0011 3B4604               38                     CMP     AX,[BP].PARAMETER_2     ;groesser als der zweite ist,
0014 7F03                 39                     JG      MAXIMAL                 ;dann liefere den ersten
0016 8B4604               40                     MOV     AX,[BP].PARAMETER_2     ;ansonsten den zweiten
0019 A30400         R     41     MAXIMAL:        MOV     MAXIMALER_WERT,AX       ;Speichere maximalen Wert
                          42
                          43     ;Prozedur Epilog*********************************************
                          44
001C 5D                   45                     POP     BP                      ;Schreibe BP und SP
                          46                                                     ;zurueck
001D C20400               47                     RET     4                       ;'Entferne' die Parameter
                          48
                          49     MAXIMUM ENDP
                          50
----                      51     CODE_PROG ENDS
                          52
                          53             END     START
```

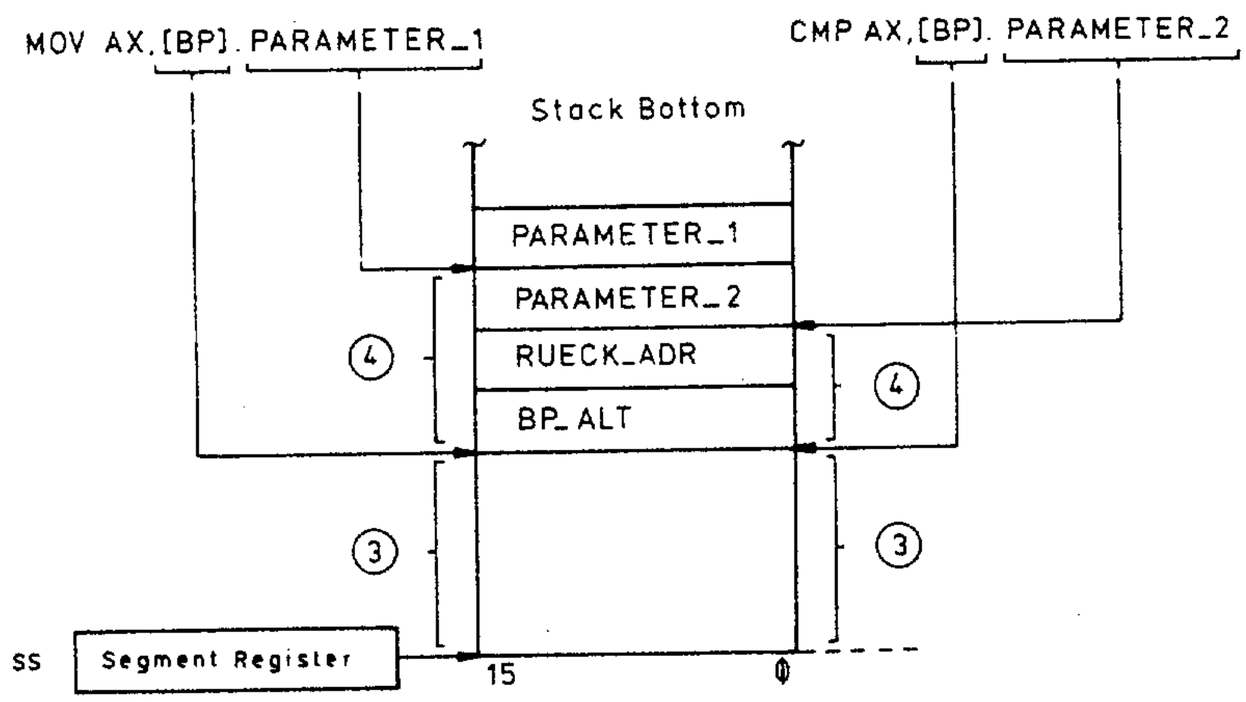
Nachdem das aufrufende Programm mit PUSH PARAMETER1/PARAMETER2 zwei Word-Parameter an den Stack übergeben hat, ermittelt der Prozedurkörper die größere der beiden Zahlen. Zu diesem Zweck werden die beiden Parameter mit [BP].PARAMETER_1 und [BP].PARAMETER_2 adressiert und miteinander verglichen. Die größere Zahl wird in MAXIMALER_WERT gespeichert.

Am Ende des Prozedur-Prolog ergibt sich eine Stackbelegung gemäß ①. Dann kennzeichnet das BP-Register den Anfang der Stackeintragungen.

Diese Eintragungen sind identisch mit den Strukturelementen in der Datenstruktur STACK_LAYOUT. Damit kann das BP-Register benützt werden, um mit [BP]. den Anfang der Datenstruktur zu markieren ②.



Das angegebene Bild zeigt die Analyse der Befehle MOV AX, [BP]. PARAMETER_1 und CMP AX, [BP]. PARAMETER_2.

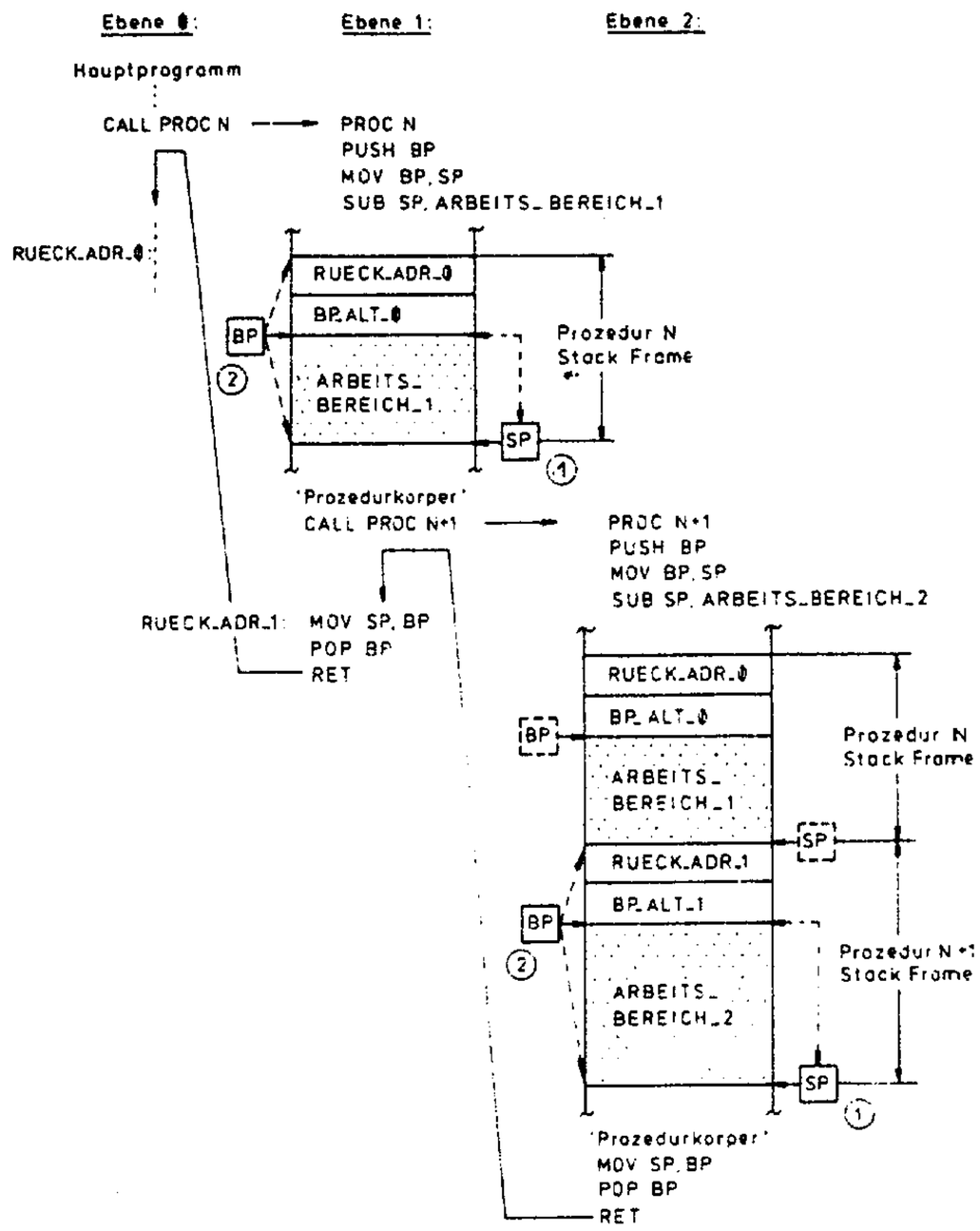


Wegen [BP]. markiert [BP] den Anfang der Datenstruktur STACK_LAYOUT bzw. den Anfang der Eintragungen im Stack. Dabei stellt BP den relativen Abstand zur Stacksegment-Basis dar ③.

Die zu adressierenden Parameter PARAMETER_1 und PARAMETER_2 stellen die relativen Abstandswerte zum Struktur-Basisregister BP dar ④.

6.10 Das BP-Register als Stack-Frame-Pointer zur Adressierung lokaler Variablen

Die Funktion des BP-Registers als Stack-Frame-Pointer soll an folgendem Bild erläutert werden:



Es existieren zwei Prozeduren PROC N und PROC N+1, in denen **lokale** Variablen bearbeitet werden. Auf lokale Variablen kann **nur** in der augenblicklichen Prozedur zugegriffen werden, d. h. sie werden gegen den Zugriff externer Programmteile abgegrenzt.

Zu diesem Zweck wird SP so verschoben, daß im Stack genügend Raum zur Bearbeitung lokaler Variablen entsteht. Die Größe des Arbeitsbereichs kann dabei abhängig von der Aufgabe in der jeweiligen Prozedur **dynamisch** erfolgen.

Gemäß der gezeigten Abbildung wird in PROC N mit `SUB SP, ARBEITS_BEREICH_1` ein anderer lokaler Arbeitsbereich festgelegt, als in PROC N+1 mit `SUB SP, ARBEITS_BEREICH_2` ①. In beiden Fällen kennzeichnet SP den Anfang (Top of Stack) ① und BP das Ende des Arbeitsbereichs ②.

Liegen geschachtelte (nested) Prozeduren wie im Beispiel vor, kennzeichnet SP außerdem den Anfang der zur augenblicklichen Prozedur gehörenden Stackbelegung (Stack-Frame). D.h. SP ist nur im Stack-Frame der augenblicklichen Prozedurebene (im Beispiel Ebene 2) aktuell.

Innerhalb des aktuellen Stack-Frame kann der lokale Arbeitsbereich mit BP erreicht werden (z.B. mit `MOV [BP-4], AL`) ②. Damit wird BP als Stack-Frame-Pointer benützt.

Durch diese Technik sind die Stack-Frames der verschiedenen Prozedurebenen und damit ihre lokalen Arbeitsbereiche gegeneinander abgegrenzt.

6.11 Anwendungsbeispiel 3

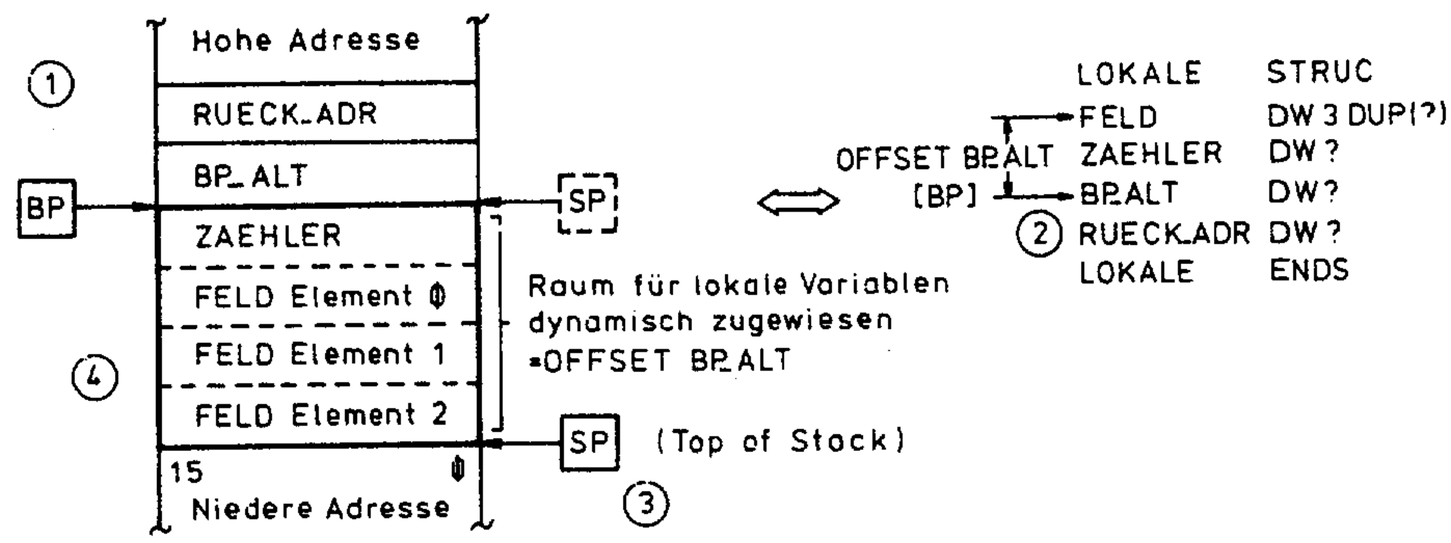
Das angegebene Programmbeispiel zeigt die Umwandlung lokaler ungepackter Dezimalzahlen in ASCII-Zeichen.

```
LOC  OBJ                  LINE     SOURCE

                           1              NAME    BEISPIEL_59_4
                           2
                           3      ASSUME CS:CODE_PROG,DS:DATEN_PROG,SS:STACK_PROG
                           4
----                       5      DATEN_PROG SEGMENT WORD PUBLIC 'RAM_REGION'
                           6
0000 00                    7      FELD_1  DB      0,1,2                   ;Feld mit 3 ungepackten Dezimalzahlen
0001 01
0002 02
                           8
----                       9      DATEN_PROG ENDS
                          10
----                      11      STACK_PROG SEGMENT WORD STACK 'RAM_REGION'
                          12
0000 (10                  13              DW      10 DUP (?)
     ????
     )
0014                      14      TOP_STACK LABEL WORD
                          15
----                      16      STACK_PROG ENDS
                          17
----                      18      CODE_PROG SEGMENT BYTE PUBLIC 'ROM_REGION'
                          19
0000 B8----          R    20      START:  MOV     AX,DATEN_PROG           ;DS Register initialisieren
0003 8ED8                 21              MOV     DS,AX
0005 B8----          R    22              MOV     AX,STACK_PROG           ;SS Register initialisieren
0008 8ED0                 23              MOV     SS,AX
000A BC1400          R    24              MOV     SP,OFFSET TOP_STACK     ;SP Register initialisieren
000D E80000               25              CALL    UMWANDLUNG              ;Prozeduraufruf
                          26              ;
                          27
0010                      28      UMWANDLUNG      PROC
                          29
                          30      ;Definition des Stack Layout als Datenstruktur***************
                          31
----                      32      LOKALE STRUC
                          33
```

```
LOC  OBJ                LINE    SOURCE

0000                     34     FELD            DW      3 DUP (?)       ;Lokales Feld mit 3 Elementen
0006                     35     ZAEHLER         DW      ?               ;Lokaler Feldlaengenzaehler (Wert=3)
0008                     36     BP_ALT          DW      ?
000A                     37     RUCK_ADR        DW      ?
                         38
----                     39     LOKALE ENDS
                         40
 -0008[]                 41     DYNAM_STACK     EQU     [BP-OFFSET BP_ALT]
                         42
                         43     ;Prozedur Prolog*********************************************
0010 55                  44             PUSH    BP                      ;Rette BP alt
0011 8BEC                45             MOV     BP,SP                   ;Markiere BP alt
0013 83EC08              46             SUB     SP,OFFSET BP_ALT        ;Stelle SP an den Anfang der
                         47                                             ;lokalen Variablen
                         48
                         49     ;Prozedurkoerper*********************************************
0016 B80300              50             MOV     AX,LENGTH FELD_1
0019 8946FE              51     LAENG:  MOV     [BP-2],AX               ;[BP-2] = 0003H (Feldlaengenzaehler)
001C A00000        R     52             MOV     AL,FELD_1
001F 8846FC              53     DEZ_0:  MOV     [BP-4],AL               ;[BP-4] = 0tes Feldelement (Wert=00H)
0022 A00100        R     54             MOV     AL,FELD_1+1
0025 8846FA              55     DEZ_1:  MOV     [BP-6],AL               ;[BP-6] = 1tes Feldelement (Wert=01H)
0028 A00200        R     56             MOV     AL,FELD_1+2
002B 8846F8              57     DEZ_2:  MOV     [BP-8],AL               ;[BP-8] = 2tes Feldelement (Wert=02H)
                         58
                         59
002E BE0000              60             MOV     SI,0                    ;Feldindex = 0 setzen
0031 8B4EFE              61             MOV     CX,DYNAM_STACK.ZAEHLER  ;CX = lokaler Zaehlerwert
                         62                                             ;(Feldlaenge)
0034 8342F830            63     ASCII:  ADD     DYNAM_STACK.FELD[SI],30H;Lokalen Dezimalwert in ASCII
                         64                                             ;umwandeln
0038 83C602              65             ADD     SI,2                    ;Bilde Feldindizes
003B E2F7                66             LOOP    ASCII                   ;Wiederhole bis alle Feldelemente
                         67                                             ;umgewandelt sind
                         68
                         69
                         70
003D 8A46FC              71     ASC_0:  MOV     AL,[BP-4]               ;AL = 0tes Feldelement (Wert=30H)
0040 8A66FA              72     ASC_1:  MOV     AH,[BP-6]               ;AH = 1tes Feldelement (Wert=31H)
0043 8A4EF8              73     ASC_2:  MOV     CL,[BP-8]               ;CL = 2tes Feldelement (Wert=32H)
                         74             ;
                         75             ;Feldelemente weiterverarbeiten
                         76             ;
                         77     ;Prozedur Epilog*********************************************
0046 8BE5                78             MOV     SP,BP                   ;Stelle SP an das Ende der
                         79                                             ;lokalen Variablen
0048 5D                  80             POP     BP                      ;Schreibe BP zurueck
0049 C3                  81             RET
                         82
                         83     UMWANDLUNG ENDP
                         84
----                     85     CODE_PROG ENDS
                         86
                         87             END     START
```

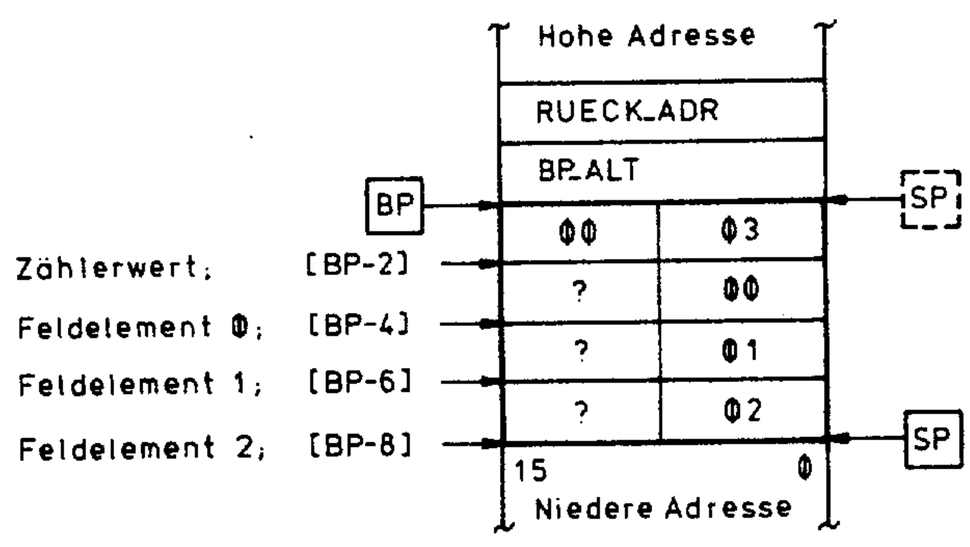
Nachdem die NEAR-Prozedur UMWANDLUNG mit `CALL UMWANDLUNG` aufgerufen worden ist, ergibt sich am Ende des Prozedur-Prologs folgende Stackbelegung ①.

Das Stack-Layout ist identisch mit der Datenstruktur LOKALE. [BP]. markiert dabei das Ende der Datenstruktur ②.

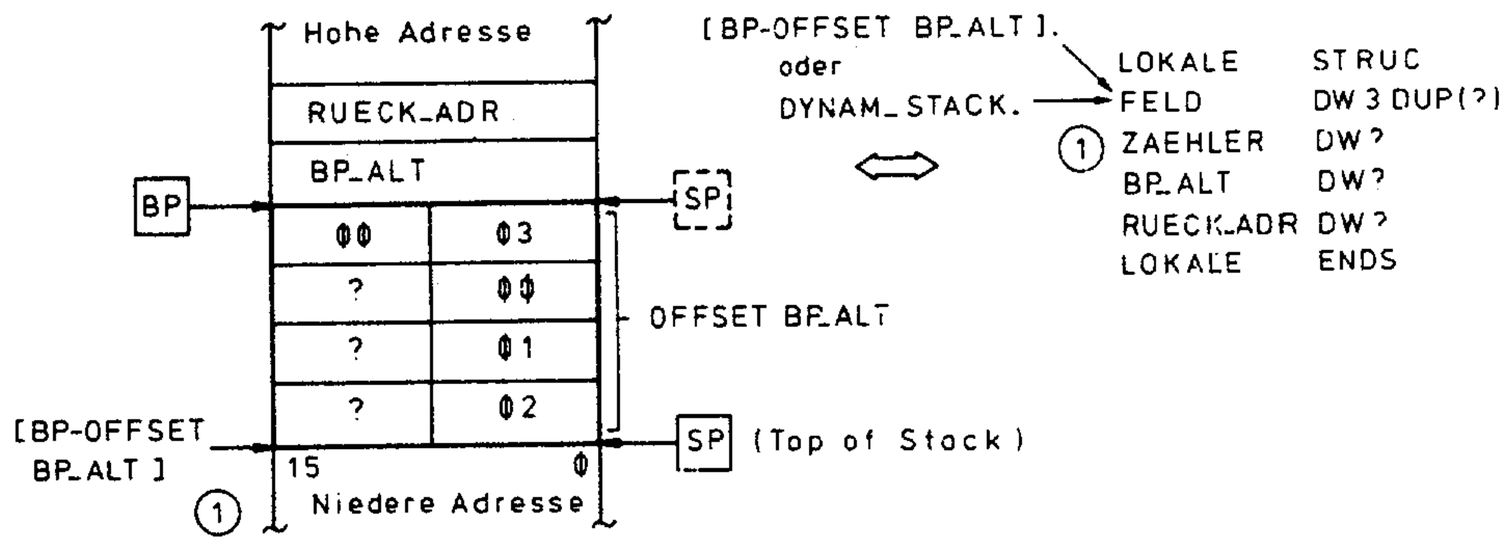


Der lokale Arbeitsraum hat in jeder Prozedur seinen individuellen Bereich. Um diesen Bereich zu optimieren, kann die Datenstruktur benützt werden. So legt der Befehl `SUB SP, OFFSET BP_ALT` den Anfang des notwendigen Arbeitsbereichs für jede Prozedur individuell (dynamisch) fest ③. Für das Anwendungsbeispiel sind dies 3 Worte für 3 lokale Feldelemente und 1 Wort für den lokalen Zahlenwert ④.

Im Prozedurkörper wird zunächst der Arbeitsbereich mit den lokalen Variablen geladen. Zu diesem Zweck wird die BP-Methode benützt. Sind die Variablen geladen, ergibt sich folgendes Stack-Layout.

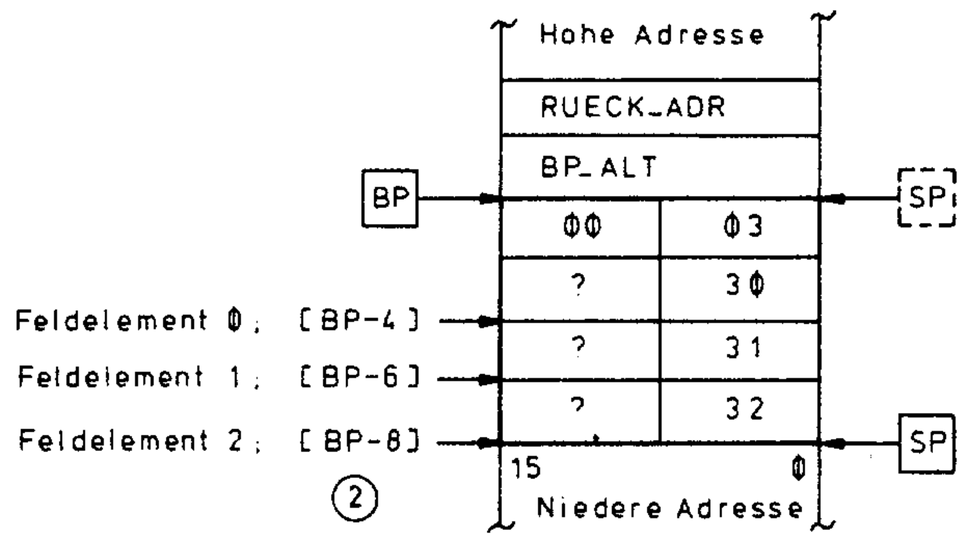


Da BP das Ende der Datenstruktur im Stack markiert, wird mit `DYNAM_STACK EQU [BP-OFFSET BP_ALT]` ein Pointer definiert, der den **Anfang** (= Top of Stack) der Datenstruktur kennzeichnet ①.



Durch die Definition von DYNAM_STACK als Strukturzeiger kann nun auf die Strukturelemente und damit auf die lokalen Variablen im Stack zugegriffen werden.

So initialisiert der Befehl `MOV CX, DYNAM_STACK.ZAEHLER` das CX-Register und der Befehl `ADD DYNAM_STACK.FELD [SI], 30H` wandelt die Feldelemente in ASCII-Zeichen um. Sind alle Feldelemente umgewandelt, ergibt sich folgendes Stack-Layout:



Die Wandelergebnisse werden nun nach der BP-Methode mit `MOV AL, [BP-4]`, `MOV AH, [BP-6]` und `MOV CL, [BP-8]` zur Weiterverarbeitung in die entsprechenden CPU-Register geladen ②.