11. Unterprogrammtechnik

11.1 Sprung und Rücksprung

11.2 Retten der Register

11.3 Parameter-Übergabe

11.1 Sprung und Rücksprung

BSR

Verzweige in ein Unterprogramm (“branch to subroutine“)

Assembler-Syntax

BSR Marke

Operation:

<PC> -> Stack

<PC>+d -> <PC>

Welche Flags werden verändert? Keine

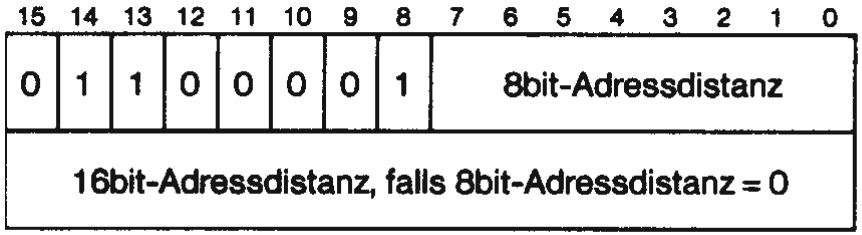
Operandengröße

Hierbei handelt es sich nicht um eine Operandengröße im herkömmlichen Sinn, sondern um die Angabe, wie groß die Adressdistanz d werden kann. Sie kann (wie üblich) 8 oder 16 Bit lang sein.

BSR

Befehlsbeschreibung

Der BSR-Befehl dient zum gezielten Springen in ein Unterprogramm mit Ablegen der Rücksprungadresse auf dem Stack. Führt der M 68000 einen BSR-Befehl aus, so wird als erstes die Adresse des auf den BSR-Befehl folgenden Befehls als Langwort auf dem Stack abgelegt. Damit ist die Adresse für die Rückkehr aus dem Unterprogramm fest gehalten. Danach springt der M 68000 zum Unterprogramm, d. h., er verzweigt zur Anfangsadresse des Unterprogramms. Die Adressdistanz d, die dabei auf den Programmzähler aufaddiert wird, ist eine Zweierkomplementzahl. Sie gibt (wie üblich) die Anzahl der Bytes an, die zwischen dem aktuellen Befehl und der Anfangsadresse des Unterprogramms liegen.



RTS / RTR (1)

Rückkehr aus dem Unterprogramm (“return from subroutine“)

Assembler-Syntax
RTS

Operation
PC vom Stack laden: (A7)+ -> <PC>

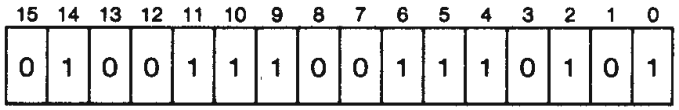
Welche Flags werden verändert? Keine

Operandengröße
Keine, da der RTS-Befehl keinen Operanden hat.

Befehlsbeschreibung
Der RTS-Befehl dient zum gezielten Rücksprung aus einem Unterprogramm. Bei Anspringen des Unterprogramms mit einem BSR- oder JSR-Befehl wird die Rücksprungadresse automatisch auf dem Stack abgelegt. Der RTS-Befehl holt diese Adresse dann wieder vom Stack und lädt sie in den Befehlszähler (PC). Der alte PC-Inhalt wird dabei natürlich überschrieben. Die Programmausführung wird im rufenden Programm mit dem Befehl, der nach dem Sprungbefehl in das Unterprogramm steht, fortgesetzt.

RTS / RTR (2)

Befehlsformat



= $4E75

RTR

Wie RTS, aber mit Rückholen des CCR vom Stack.

Programmbeispiel zu BSR/RTS (1)

Dieses Programm dient zum Umwandeln von Werten in ASCII in die hexadezimale Darstellung per Programm. Das Programm erwartet bei Start die Anzahl der zu wandelnden ASCII-Werte (geradzahlig) im Datenregister D2 und die Anfangsadresse für die Ablage der gewandelten Hexwerte im Speicher im Adressregister A2. Das Programm liest die zu wandelnden ASCII-Werte von einer ACIA-Schnittstelle an Adresse $3000 ein.

Programmbeispiel zu BSR/RTS (2)

```
13
14                00001000                ORG       $1000
15  00001000      48E7C000                MOVEM.L   D0/D1, -(A7)   Verwendete Register
                                                                     retten
16  00001004      4280                    CLR.L     D0
17  00001006      04420001                SUBI.W    #1, D2
18  0000100A      103900003000  LOOP      MOVE.B    ACIA, D0       1. ASCII Wert laden
19  00001010      6100001E                BSR       WANDEL         Wandeln unteres
                                                                     Halbbyte
20  00001014      1200                    MOVE.B    D0,D1          Unteres Halbbyte
                                                                     retten
21  00001016      103900003000            MOVE.B    ACIA, D0       2. ASCII Wert laden
22  0000101C      61000012                BSR       WANDEL         Wandeln oberes
                                                                     Halbbyte
23  00001020      E909                    LSL.B     #4,D1          Oberes Halbbyte an
                                                                     richtige
                                                                     Stelle verschieben
24                *
25  00001022      D200                    ADD.B     D0, D1         Oberes und unters
                                                                     Halbbyte
                                                                     zusammenfassen
26                *
27  00001024      14C1                    MOVE.B    D1, (A2)+      Gewandeltes Byte
                                                                     ablegen
28  00001026      51CAFFE2                DBRA      D2, LOOP
29  0000102A      4CDF0003                MOVEM.L   (A7)+,D0/D1    Register zurückholen
30  0000102E      4E75                    RTS
31
```

Zeilen 18, 21: Jeder MOVE.B-Befehl liest ein neues ASCII-Zeichen von der ACIA-Schnittstelle ein.

Programmbeispiel zu BSR/RTS (3)

```
32  00001030      0C000039      WANDEL    CMPI.B    #$39, D0       Unterprogramm
                                                                   Wandeln, Abfrage,
33  00001034      6E00000A                BGT       TR             ob größer 9.
                                                                   Falls  nein,
34  00001038      04000030                SUBI.B    #$30, D0       $30 von ASCII Wert
                                                                   subtrahieren
35  0000103C      60000006                BRA       ENDE
36  00001040      04000037      TR        SUBI.B    #$37, D0       falls ja $37 von ASCII
                                                                   Wert subtrahieren
38  00001044      4E75          ENDE      RTS
39                00003000      ACIA      EQU       $3000
40                                        END
```

```
***** Total Errors       0--
***** Total Warnings     0--
```

Zeile 36: Man bedenke: Der Eingabedatenstrom enthält nur die ASCII-Zeichen ‘0‘ - ‘9‘ und ‘A‘ - ‘F‘.

11.2 Retten der Register

LINK

Rette den Stackpointer und lege einen neuen Stackbereich an

Assembler-Syntax

LINK An, #Adressdistanz

Operation

<An> -> -(A7)

<A7> -> <An>

<A7>+ #Adressdistanz -> <A7>

Welche Flags werden verändert? Keine

Operandenlänge

Beim LINK-Befehl wird der 32-Bit-Inhalt des spezifizierten Adressregisters benötigt, und die Adressdistanz (sie muss negativ sein) ist eine 16-Bit-Zahl im Zweierkomplement. Die 16-Bit-Adressdistanz wird vor ihrer Verarbeitung vorzeichenrichtig auf 32 Bits erweitert.

LINK (1)

Befehlsbeschreibung

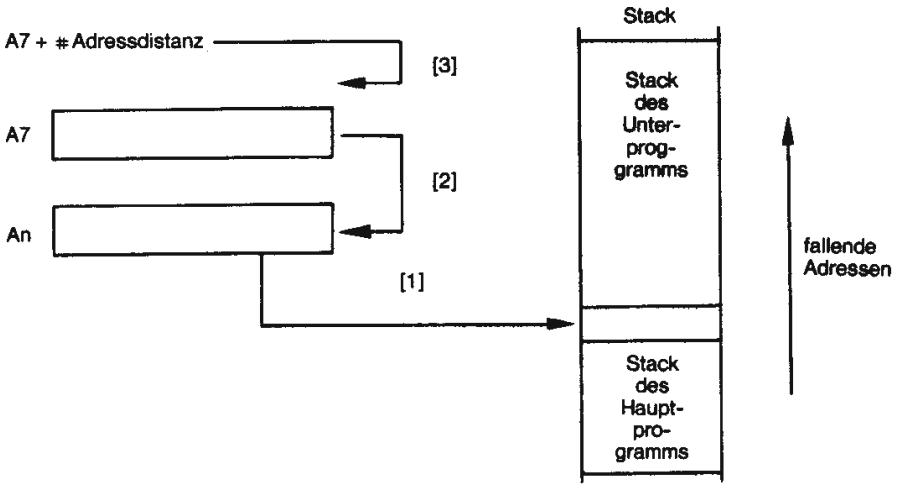
Der LINK-Befehl dient zum Anlegen eines eigenen Stackbereichs für ein Unterprogramm.

Bei Ausführung des LINK-Befehls wird zunächst der Inhalt des im Befehl spezifizierten Adressregisters auf dem Stack abgelegt. Danach wird der Stackpointer selbst in das eben gerettete Adressregister geladen. Zum Schluss wird auf den Stackpointer die Adressdistanz aufaddiert. Damit hat man einen eigenen Stackbereich erzeugt und kann diesen z. B. für ein Unterprogramm nutzen.

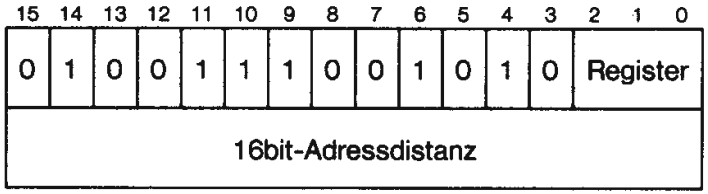
Mit dem LINK-Befehl kann man sich also insbesondere für ein Unterprogramm einen eigenen Stackbereich schaffen. Im Normalfall steht der LINK-Befehl als erster Befehl des Unterprogramms, gleich anschließend folgt ein MOVEM-Befehl.

Da der Stackpointer des Hauptprogramms in einem der Adressregister steht, kann man, falls die Daten des Hauptprogramms auf dem Stack abgelegt sind, auch vom Unterprogramm aus auf diese zugreifen.

LINK (2)



Opcode



UNLK (1)

Gegenstück zum LINK-Befehl. Holt den Stackpointer zurück.

Assembler-Syntax

UNLK An

Operation

Inhalt von An -> <A7>

(A7)+ -> <An>

Welche Flags werden verändert? Keine

Operandengröße

Da beim UNLK-Befehl Adressen in Adressregister geladen werden, arbeitet der Befehl nur mit 32-Bit-Operandengröße.

UNLK (2)

Befehlsbeschreibung

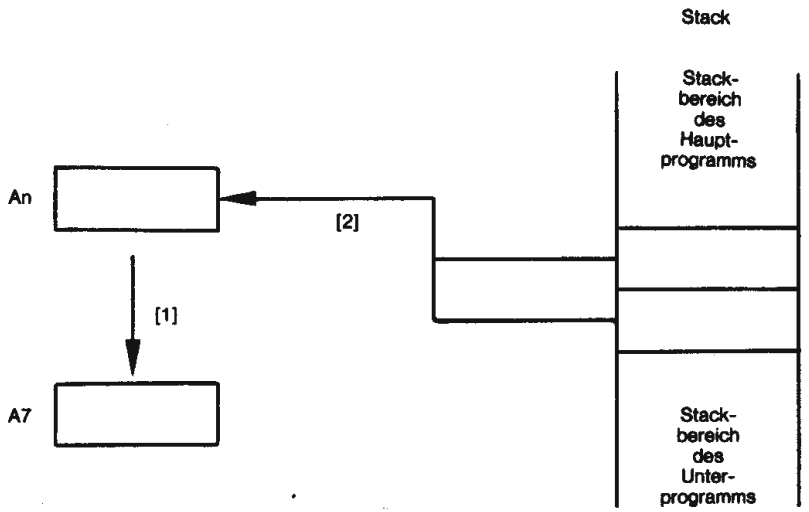
(siehe auch LINK-Befehl)

Der UNLK-Befehl dient zum Abbauen des eigenen Stackbereichs eines Unterprogramms bei Rücksprung in das Hauptprogramm. Der UNLK-Befehl ist das Gegenstück zum LINK-Befehl. Bei Ausführung des UNLK-Befehls wird zunächst der Stackpointer des Hauptprogramms, den der LINK-Befehl in ein Adressregister An abgelegt hatte, in den Stackpointer A7 zurückgeholt. Anschließend wird der vorherige Inhalt des dazu benötigten Adressregisters An vom Stack des Hauptprogramms in das Adressregister An zurückgespeichert. Damit ist der Stackbereich und der alte Registerinhalt von An für das Hauptprogramm wieder vorhanden.

Anwendung

In Verbindung mit dem LINK-Befehl dient der UNLK-Befehl zum Aufbauen bzw. Abbauen eigener Stackbereiche für Unterprogramme. Der UNLK-Befehl wird vor allem eingesetzt in Compilern für höhere Programmiersprachen. Wie man sieht, ist der M 68000 sehr gut für höhere Programmiersprachen geeignet.

UNLK (3)



Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

Beispiel für LINK und UNLK

Unterprogramm mit eigenem Stackbereich

```
Befehlsadr.     Befehl

$1000           MOVE.L     D2,DS
$1002           JSR        $2000
$1006           MOVE.W     D4,$2560
.               .
.               .
.               .
$2000           LINK       A2,-$30   Aufbauen des eigenen Stackbereichs
$2004           MOVEM.L    D0-D7/A0/A1/A3-A6, -(A7)
.               .
.               .
.               .
$2100           MOVEM.L    (A7)+, DO-D7/A0/A1/A3-A6
$2104           UNLK A2    Abbauen des eigenen Stackbereichs
$2106           RTS
```


11.3 Parameter-Übergabe

Drei Methoden

1. Übergabe in Registern
2. Übergabe der Adresse eines Versorgungsblocks
3. Wert- und Ergebnisübergabe über den Laufzeitkeller (Stack)

Erste Methode: Übergabe in Registern

Für jedes Unterprogramm wird verabredet, dass bestimmte Daten- oder Adressregister beim Aufruf die Werte der Eingabeparameter und transienten Parameter enthalten und dass bei Rückkehr aus dem Unterprogramm dieselben oder andere Register die dann gültigen Werte der transienten Parameter und der Ergebnisparameter enthalten.

Dieses Verfahren eignet sich offensichtlich nur für Parameter, deren Umfang ein Register nicht übersteigt (zum Beispiel nicht für Felder (arrays)). Es kann außerdem zu einem erheblichen Verbrauch von Registern führen, die dann bei der Ausführung des Unterprogramms fehlen. Andererseits ist dies das Verfahren mit dem geringsten organisatorischen Aufwand. Die Methode wird daher insbesondere für häufig aufgerufene (kleine) Standard-Unterprogramme benutzt.

Zweite Methode: Adressübergabe (Versorgungsblock-Technik)

Hier wird in einem Adressregister die Anfangsadresse eines zusammen hängenden Speicherbereichs übergeben, der die Parameter enthält. Das Unterprogramm greift auf die im Versorgungsblock vorhandenen Eingabewerte zu und legt dort auch seine Ergebnisse ab.

Das Verfahren ist offensichtlich nur geeignet, wenn man wenige Parameter hat, die zudem zusammen hängende Speicherbereiche belegen. Dies ist üblicherweise bei Systemaufrufen der Fall, so dass dies die bevorzugte Übergabetechnik für Aufrufe an das Betriebssystem ist.

Übrigens lässt sich das Verfahren auch auf Parameter in nicht zusammenhängenden Speicherbereichen anwenden, indem man in den Versorgungsblock nicht die Werte, sondern die Adressen der Parameter aufnimmt und dann doppelt indirekt adressiert.

Dritte Methode: Wert- und Ergebnisübergabe über den Stack

Die Werte der Eingabe- und transienten Parameter werden vor dem Unterprogrammaufruf auf dem Stack abgelegt; für Ergebnisparameter wird entsprechender Platz reserviert. Nach der Rückkehr aus dem Unterprogramm entnimmt man dem Stack die Werte der transienten und der Ergebnisparameter.

Bei Parametern, die sehr umfangreich sind und daher nicht kopiert werden sollten (wie z. B. Felder), kann man statt des Wertes des Parameters auch seine Adresse auf dem Stack ablegen.

Diese Technik ist bei der Implementierung höherer Programmiersprachen weit verbreitet. Die Assembler-Anweisungen für die Parameterübergabe werden vom Compiler generiert.

Beispiel zur dritten Methode

```
00002000  =00007FFF   MAXINT   EQU    $7FFF
00002000  =00000008   PAR      EQU    8
00002000  =0000000A   ERG      EQU    10
00002000
00002000  4E56 0000   FAC      LINK   A6,#0
00002004  302E 0008            MOVE.W PAR(A6),D0
00002008  0C40 0001            CMPI.W #1,D0
0000200C  6700 001C            BEQ    ENDE
00002010
00002010  5340                 SUBQ.W #1,D0
00002012  554F                 SUBQ   #2,SP    reserviere Platz
00002014            *                          für Ergebnis
00002014  3F00                 MOVE.W D0,-(SP)
00002016
00002016  61E8                 BSR    FAC
00002018  544F                 ADDQ   #2,SP
0000201A
0000201A  302E 0008            MOVE.W PAR(A6),D0
0000201E  C1DF                 MULS   (SP)+,D0
00002020  0C80 00007FFF        CMPI.L #MAXINT,D0
00002026  6E00 000A            BGT    UEBERL
0000202A  3D40 000A   ENDE     MOVE.W D0,ERG(A6)
0000202E  4E5E                 UNLK   A6
00002030  4E75                 RTS
00002032              UEBERL   ...
```