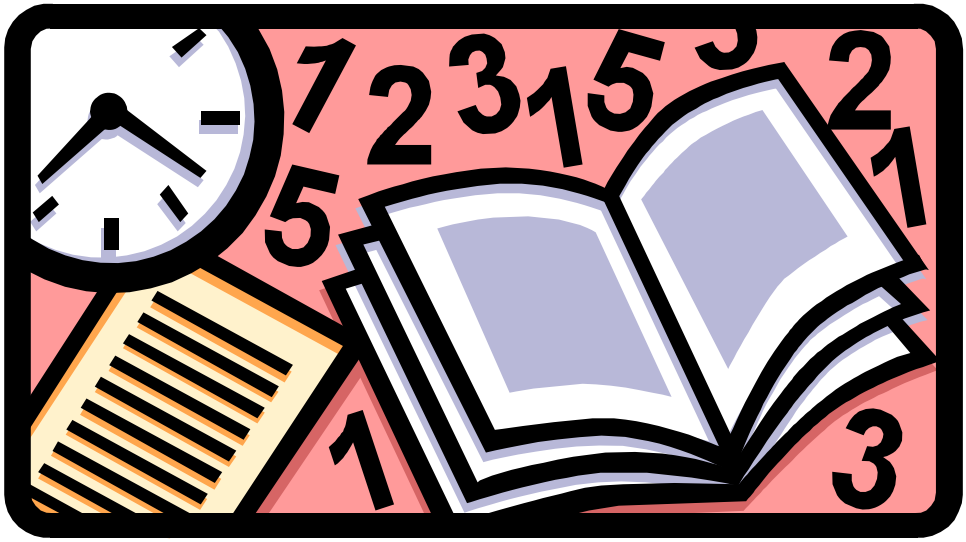
Assembler I - Grundlagen 1. Teil

Assembler – Unterprogramme und Makros



1. Unterprogramme

In den bisher gezeigten Programmen wurde schon deutlich, dass manche Programmteile sich öfters wiederholen oder mittels Schleifen mehrmals durchlaufen werden.

Bei größeren Programmen bedeutet dies zusätzliche Schreibarbeit. Außerdem werden solche Programme schnell unübersichtlich, wenn nicht mehr klar ist in welcher Reihenfolge und in welchen Fällen welche Sprünge ausgeführt werden. Selbst der ursprüngliche Programmierer tut sich irgendwann schwer da noch durchzublicken. Es kommt der Zeitpunkt an dem der Programmier im Programmcode nur noch planlos umherhüpft und sich in den endlosen Weiten des Programmcodes verliert.

Man nennt diesen Programmierstiel dann "Spaghetti – Code".

1.1 Aufbau und Struktur von Unterprogrammen

Aus diesem Grund ist man auf die Idee gekommen, dass man die Programmteile welche öfters durchlaufen werden einfach in ein eigenes Programm packt.

Dieses Unterprogramm wird dann einfach im Hauptprogramm an der benötigten Stelle aufgerufen. Das Hauptprogramm wartet dann solange, bis das Unterprogramm seine Aufgabe durchgeführt hat und fährt danach mit der Ausführung des restlichen Programmcodes fort.

Wie startet man nun ein solches Unterprogramm aus dem Hauptprogramm heraus ?

Die geschieht mit dem Befehl: CALL (→ rufen)

Nun muss man natürlich auch noch angeben welches Unterprogramm (Namen) man auf*ruft*. Den Namen kann man sich selbst auswählen. Der Befehl CALL darf nicht verändert werden.

Der Aufruf im Hauptprogramm sieht daher so aus: CALL PROGRAMMNAME

Dieses Unterprogramm befinden sich meist am Ende des Hauptprogramms. Der Assembler muss nun aber auch erkennen können an welcher Stelle ein bestimmtes Unterprogramm beginnt und an welcher Stelle es endet.

Ein Unterprogramm hat folgendes Grundgerüst:

```
PROGRAMMNAME  PROC   [NEAR][FAR]     ← Anfang des Unterprogramms

              .
              .
        Anweisungen
              .
              .
                    RET
PROGRAMMNAME  ENDP                    ← Ende des Unterprogramms
```

Die Ausdrücke NEAR und FAR sind grün, weil wir uns für **einen** Ausdruck von beiden entscheiden müssen. (Die eckigen Klammern werden nicht geschrieben !)

Es wird der Ausdruck NEAR gewählt, wenn sich das Unterprogramm mit dem Namen PROGRAMMNAME im gleichen Segment wie sein Aufruf (CALL PROGRAMMNAME) befindet.

Die RET – Anweisung zeigt dem Assembler an, an welcher Stelle wieder ins Hauptprogramm zurück gesprungen wird.

Die PROGRAMMNAME ENDP – Anweisung markiert das Ende des Unterprogramms.

1.1.1 NEAR - Unterprogramme

Das Programm HALLO.EXE soll nun mittels eines Unterprogramms erstellt werden.
Wir schreiben den unteren Programmcode in eine Datei UPROG1.ASM

```
DATEN SEGMENT

      MELDUNG  DB  "HALLO WELT !","$"

DATEN ENDS


STAPEL SEGMENT BYTE STACK

            DW 128  DUP(0)

STAPEL ENDS


CODE SEGMENT

   ASSUME CS:CODE,DS:DATEN,ES:NOTHING,SS:STAPEL

START:    MOV AX, DATEN
          MOV DS, AX

          CALL PRINT

          MOV AH, 4CH
          INT 21H


PRINT  PROC  NEAR

        MOV AH, 09H
        MOV DX, OFFSET MELDUNG
        INT 21H

       RET
PRINT ENDP


      CODE ENDS
END START
```

Ja, ok…. Dieses Programm würde niemand mittels eines Unterprogramms schreiben. Es eignet sich aber besonders um die Verwendung eines Unterprogramms zu zeigen.

Nachdem dieses Programm assembliert und gelinkt ist werden wir uns den Maschinencode mittels DEBUG anschauen.

Damit ist es möglich uns ein Bild von der Wirkung der einzelnen Befehle zu machen.

Wir geben am DOS – Prompt ein:

```
C:>\ASM>DEBUG UPROG1.EXE
-u
```

…und wir sehen unser Programm im Maschinencode...

```
1EB8:0000 B8A71E      MOV    AX,1EA7      1.  START
1EB8:0003 8ED8        MOV    DS,AX        2.
1EB8:0005 E80400      CALL   000C         3.
1EB8:0008 B44C        MOV    AH,4C        8.
1EB8:000A CD21        INT    21           9.  ENDE
1EB8:000C B409        MOV    AH,09        4.
1EB8:000E BA0000      MOV    DX,0000      5.
1EB8:0011 CD21        INT    21           6.
1EB8:0013 C3          RET                 7.
1EB8:0014 83CA02      OR     DX,+02       10.
1EB8:0017 3C3F        CMP    AL,3F        11.
1EB8:0019 7503        JNZ    001E         .
1EB8:001B 83CA04      OR     DX,+04       .
1EB8:001E 0AC0        OR     AL,AL
-
```

Auch hier erkennt DEBUG nicht an welcher Stelle unser Programm endet. Der rote Teil ist nicht mehr Bestandteil unseres Programms. Es ist zu erkennen, dass das Unterprogramm im selben Segment steht wie das restliche Programm. Es ist also wirklich ein NEAR Sprung.

Am rechten Rand steht die Reihenfolge in der die einzelnen Programmzeilen ausgeführt werden. Es ist zu erkennen, dass der CALL Befehl eigentlich ein Sprungbefehl ist. Die OFFSET – Adresse hinter dem Sprungbefehl gibt das Sprungziel an. Die OFFSET – Adresse 000C hat bei uns den "Namen" PRINT.

Durch den RET – Befehl wird wieder zurück gesprungen.

1.1.2 FAR – Unterprogramme

Es kommt relativ oft vor, dass Routinen mehrmals in einem Programm verwendet werden. So zum Beispiel Routinen für das Ausgeben von Text, das Öffnen von Dateien, das Beenden von Programmen usw.....

Das Hauptprogramm und die Standartroutinen werden in getrennte Dateien geschrieben. Auf diese Weise kann man die einmal geschriebenen Standartroutinen immer wieder in jedes neue Programm einbinden (spart viel Tipperei ☺).

Hinweis: Die Dateien DATEI_2.ASM und SUB_PROG.ASM müssen nicht vollständig neu abgetippt werden. Wenn man DATEI_1.ASM hat kann man durch kleine Abänderungen und durch das Abspeichern unter einem anderen Namen die oben genannten Dateien leicht erstellen.

Ein Beispiel:

Wir schreiben zunächst eine Datei DATEI_1.ASM

```
EXTRN PRINT:FAR
EXTRN MSDOS:FAR

DATEN SEGMENT

      MELDUNG  DB  "Meldung aus Datei 1!","$"

DATEN ENDS

STAPEL SEGMENT BYTE STACK

            DW 128 DUP(0)

STAPEL ENDS

CODE SEGMENT

      ASSUME CS:CODE,DS:DATEN,ES:NOTHING,SS:STAPEL

START:  MOV AX, DATEN
      MOV DS, AX
      MOV DX, OFFSET MELDUNG

      CALL PRINT

      CALL MSDOS

CODE ENDS
      END START
```

Danach wird eine Datei DATEI_2.ASM erstellt.

```
EXTRN PRINT:FAR
EXTRN MSDOS:FAR

DATEN SEGMENT

      MELDUNG  DB  "Meldung aus Datei 2!","$"

DATEN ENDS

STAPEL SEGMENT BYTE STACK

            DW 128 DUP(0)

STAPEL ENDS

CODE SEGMENT

      ASSUME CS:CODE,DS:DATEN,ES:NOTHING,SS:STAPEL

START:  MOV AX, DATEN
      MOV DS, AX
      MOV DX, OFFSET MELDUNG

      CALL PRINT

      CALL MSDOS

CODE ENDS
      END START
```

Zum Schluss eine Datei SUB_PROG.ASM

```
PUBLIC PRINT
PUBLIC MSDOS

DATEN SEGMENT



DATEN ENDS

STAPEL SEGMENT BYTE STACK

            DW 128 DUP(0)

STAPEL ENDS


CODE SEGMENT

      ASSUME CS:CODE,DS:DATEN,ES:NOTHING,SS:STAPEL

START:  PRINT PROC FAR

      MOV AH, 09H
      INT 21H

      RET
      PRINT ENDP

      MSDOS PROC FAR

      MOV AH, 4CH
      INT 21H

      RET
      MSDOS ENDP

CODE ENDS
      END START
```

Nun wird jede Datei einzeln mit TASM assembliert:

```
C:\asm>TASM DATEI_1.ASM
Turbo Assembler  Version 1.01  Copyright (c) 1988, 1989 Borland International

Assembling file:   DATEI_1.ASM
Error messages:    None
Warning messages:  None
Remaining memory:  411k


C:\asm>TASM DATEI_2.ASM
Turbo Assembler  Version 1.01  Copyright (c) 1988, 1989 Borland International

Assembling file:   DATEI_2.ASM
Error messages:    None
Warning messages:  None
Remaining memory:  411k

C:\asm>TASM SUB_PROG.ASM
Turbo Assembler  Version 1.01  Copyright (c) 1988, 1989 Borland International

Assembling file:   SUB_PROG.ASM
Error messages:    None
Warning messages:  None
Remaining memory:  41
```

```
C:\asm>_
```

Dann können die einzelnen Hauptprogramme DATEI_1.OBJ und DATEI_2.OBJ jeweils mit der Unterprogrammdatei SUB_PROG.OBJ zusammengebunden werden.

Dies geschieht mittels:

```
C:\asm>TLINK DATEI_1.OBJ  SUB_PROG.OBJ
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International

C:\asm>TLINK DATEI_2.OBJ  SUB_PROG.OBJ
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International

C:\asm>_
```

Durch Eingabe von:

```
C:\asm>Datei_1
```

Sehen wir:
Meldung aus Datei 1!

Durch Eingabe von:

```
C:\asm>Datei_2
```

Sehen wir:
Meldung aus Datei 2!

Es tauchen nun in den Dateien DATEI_1.ASM und DATEI_2.ASM die Ausdrücke EXTRN und PUBLIC auf.

In der Datei DATEI_1.ASM werden die Unterprogramme PRINT und MSDOS aufgerufen. Der Assembler kann aber diesmal diese Unterprogramme in der Datei DATEI_1.ASM nicht finden ! Wenn nun nicht die Hinweise EXTRN PRINT:FAR und EXTRN MSDOS:FAR in der Datei wären, würde der Assembler eine Fehlermeldung ausgeben. Mit diesen Hinweisen wird dem Assembler mitgeteilt, dass diese Unterprogramme mit den Namen PRINT und MSDOS eben außerhalb (extern) und im wahrsten Sinne weit weg (far) von unserer Datei DATEI_1.ASM (nämlich in der Datei SUB_PROG.ASM) existieren.

Der Assembler lässt also das Programm DATEI_1.OBJ lediglich zwei "Ösen" auswerfen, an denen dann der Linker (TLINK) die Unterprogramme einklinken kann. Dazu muss der Assembler aber die Datei SUB_PROG.ASM zwei "Haken" auslegen lassen. Damit der Assembler dies veranlasst braucht es die Ausdrücke PUBLIC PRINT und PUBLIC MSDOS.

Das mit den Haken und Ösen wollen wir nun etwas genauer anschauen. Wir geben ein:

```
C:\asm>DEBUG DATEI_1.EXE
-u
1EC9:0000 B8A71E          MOV     AX,1EA7
1EC9:0003 8ED8            MOV     DS,AX
1EC9:0005 BA0000          MOV     DX,0000
1EC9:0008 9A0000CB1E      CALL    1ECB:0000
1EC9:000D 9A0500CB1E      CALL    1ECB:0005
1EC9:0012 0000        ADD     [BX+SI],AL
1EC9:0014 0000        ADD     [BX+SI],AL
1EC9:0016 0000        ADD     [BX+SI],AL
1EC9:0018 0000        ADD     [BX+SI],AL
1EC9:001A 0000        ADD     [BX+SI],AL
1EC9:001C 0000        ADD     [BX+SI],AL
1EC9:001E 0000        ADD     [BX+SI],AL
```

Zunächst erkennen wir hier (bei anderen Rechnern werden sich andere Segmentadressen ergeben), dass in den ersten zwei Zeilen die Adresse des Datensegments (1EA7) ins DS Register geschoben wird. Für DX ist in der dritten Zeile die Offset – Adresse 0000 zu entnehmen. Dies bedeutet, dass unsere Zeichenkette an Adresse 1EA7 : 0000 beginnt.

Wir suchen nun den String "Meldung aus Datei 1!" im Arbeitsspeicher auf. Dazu geben wir ein:

```
-E 1EA7:0000
```

und sehen zunächst das M von Meldung. Durch wiederholtes Drücken der Leerzeichentaste schauen wir uns die ganze Zeichenkette an.

```
-E 1EA7:0000
1EA7:0000  4D.   65.   6C.   64.   75.   6E.   67.   20.
1EA7:0008  61.   75.   73.   20.   44.   61.   74.   65.
1EA7:0010  69.   20.   31.   21.   24.
```

Die 24 ist wieder das $ - Zeichen, welches das Ende des Strings kennzeichnet. Als nächstes wollen wir uns das Unterprogramm anschauen, welches den String ausgibt. Dazu geben wir ein:

```
-u 1ECB:0000
```

Wir sehen:

```
1ECB:0000 B409        MOV    AH,09
1ECB:0002 CD21        INT    21
1ECB:0004 CB          RETF
1ECB:0005 B44C        MOV    AH,4C
1ECB:0007 CD21        INT    21
1ECB:0009 CB          RETF
```

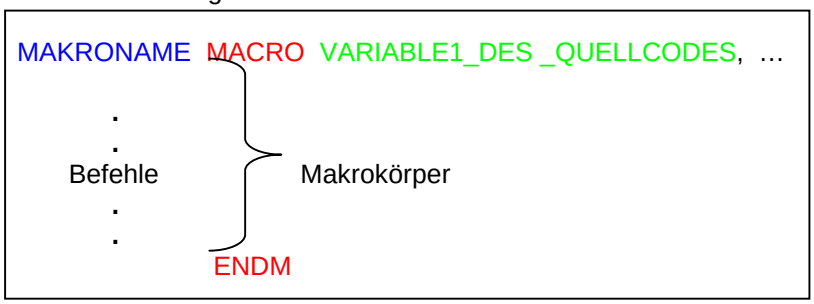
Wir sehen nun auch an Adresse 1ECB:0005 den Beginn des Unterprogramms, welches nach Ablauf unseres Gesamtprogramms die Kontrolle an das Betriebssystem zurück gibt. Ebenfalls sehen wir nun auch, dass wir ein Unterprogramm vom Typ FAR benötigen. Der CALL sitzt im Segment 1EC9. Das aufgerufene Unterprogramm sitzt im Segment 1ECB. Beide Segmente sind unterschiedlich !

2. Makros

Makros haben einen ähnlichen Aufbau wie Unterprogramme. Die Wirkung im Quellcode ist jedoch eine völlig andere. Ein Unterprogramm wird nur einmal in den Arbeitsspeicher geschrieben. Dann wird von verschiedenen Stellen des Hauptprogramms zu diesem Unterprogramm gesprungen.

Der Assembler ersetzt beim Assemblieren den Aufruf des Makros einfach durch den Inhalt des Makro ! Dies bedeutet, dass durch einen Makroaufruf kein Sprung entsteht. Wenn wir zum Beispiel ein Makro dreimal hintereinander aufrufen wird der Inhalt dieses Makro im Quellcode lediglich dreimal hintereinander gehängt.

Ein Mako hat folgenden Aufbau:



Wie veranlasst man nun den Assembler den Makrokörper in den Quellcode einzufügen ?

Dies gelingt mit dem Ausdruck:

```
MAKRONAME  VARIABLE1_DES_MAKRO, …
```

Neu sind die Zusätze "VARIABLE1_DES_MAKRO, …" und " VARIABLE1_DES _QUELLCODES,..." . Mit diesem Zusatz erfährt der Assembler, dass die beiden unterschiedlichen Ausdrücke das gleiche meinen und austauschbar sind.

Wie der Name Variable schon vermuten lässt, werden über diese Ausdrücke Zahlenwerte ausgetauscht. Es können mehrere Variablen hintereinander geschrieben werden.

Ein Beispiel:

Wir erstellen zuerst eine Makro – Bibliothek. Zu diesem Zweck erstellen wir mit dem DOS- Editor die Datei MACRO.BIB und schreiben folgende Makros :

```
PRINT MACRO STRING

MOV AH, 09H
MOV DX, OFFSET STRING
INT 21H

ENDM



MSDOS MACRO

MOV AH, 4CH
INT 21H

ENDM
```

Der Assembler muss nun diese Datei nach den Makros durchsuchen, die im Hauptprogramm verwendet werden. Damit er das tut, muss im Hauptprogramm auf diese Datei hingewiesen werden. Dies geschieht mit dem Ausdruck INCLUDE.

Jetzt wird das Hauptprogramm DATEI_3.ASM erstellt.

```
INCLUDE MACRO.BIB

DATEN SEGMENT

      MELDUNG  DB  "Meldung aus Datei 3!","$"

DATEN ENDS

STAPEL SEGMENT BYTE STACK

            DW 128 DUP(0)

STAPEL ENDS

CODE SEGMENT

      ASSUME CS:CODE,DS:DATEN,ES:NOTHING,SS:STAPEL

START:  MOV AX, DATEN
      MOV DS, AX

      PRINT MELDUNG

     MSDOS

CODE ENDS
      END START
```

Der Assembler setzt hier die beiden Ausdrücke MELDUNG und STRING gleich. Es wird die Offset – Adresse der Zeichenkette MELDUNG übergeben.

Zum Assemblieren geben wir ein:

C:\asm>TASM DATEI_3.ASM

Wir sehen:
Turbo Assembler Version 1.01 Copyright (c) 1988, 1989 Borland International

Assembling file: DATEI_3.ASM
Error messages: None
Warning messages: None
Remaining memory: 411k

Dann wird eingegeben:

C:\asm>TLINK DATEI_3.OBJ

Wir sehen:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
C:\asm>_

Wir schauen uns das Programm mit DEBUG an.

Wir geben ein:

```
C:\asm>DEBUG DATEI_3.EXE
-u
```

Wir sehen:

```
1EB9:0000 B8A71E      MOV     AX,1EA7
1EB9:0003 8ED8        MOV     DS,AX
1EB9:0005 B409        MOV     AH,09
1EB9:0007 BA0000      MOV     DX,0000
1EB9:000A CD21        INT     21
1EB9:000C B44C        MOV     AH,4C
1EB9:000E CD21        INT     21
```

Man kann hier nicht erkennen, dass dieses Programm mittels Makros erstellt wurde.

Fazit: Programme werden mittels Unterprogrammen geschrieben, wenn sie möglichst klein sein sollen.
Für den Fall, dass jedoch die Geschwindigkeit der Programme im Vordergrund steht, bedient man sich lieber Makros.