1 Grundlagen

1.1 Wozu braucht man einen Assembler?

Anders als in BASIC oder anderen interpretativen Sprachen besteht ein ausführbares EXE- oder COM-Programm aus einer Aneinandereihung von Binärzahlen. Diese Zahlen lassen sich zum Beispiel mit DEBUG als Hexdump auf dem Bildschirm ausgeben:

D9 C3 00 43 47 33 45 89 90 90 90 ....

Die CPU des Rechners kann nun diese Codes als Steueranweisungen (Maschinenprogramm) verarbeiten. Für den Menschen ist die Darstellung jedoch alles andere als instruktiv. Er benötigt eine symbolische Schreibweise für die Befehle. Aufgabe des Assemblers ist es nun, ein Programm mit symbolischen Assembleranweisungen in einzelne Maschinencodes umzuformen und damit ein Maschinenprogramm in Form einer EXE- oder COM-Datei zu erzeugen. Die Assembleranweisungen werden per Editor in eine Textdatei mit der Extension .ASM abgelegt.

```
_   .MODEL TINY

    .STACK 200H

    .DATA             ; Daten vereinbaren

 Text  DB 'Hallo  $'

    .CODE             ; Programm Start

 Start:
    MOV  AH,09        ; Textausgabe
    MOV  DX,Offset Text
    INT 21
    MOV  AX,4C00      ; Exit
    INT 21

    END Start
```

Bild 1.1: Aufbau eines Assemblerprogramms

Der Assembler liest dann die Anweisungen der Quelldatei (.ASM) und erzeugt daraus eine COM- oder eine OBJ-Datei, je nach Anweisung des Nutzers. Bei der Übersetzung

führt der Assembler weiterhin noch eine Fehlerüberprüfung und eine Berechnung der Sprungadressen bei JMP- und CALL-Befehlen durch.

Besteht das Programm nur aus Assembleranweisungen und wird die Größe auf 64 KByte begrenzt, besteht bei vielen Assemblern die Möglichkeit direkt eine ausführbare COM-Datei zu erzeugen. Soll das Programm aber mit anderen Programmen oder -teilen zusammen gebunden werden, erzeugt der Assembler eine OBJ-Datei, die nur den übersetzten Code enthält. Diese OBJ-Dateien lassen sich dann durch ein spezielles Programm, welches als Linker bezeichnet wird, zu einer ausführbaren COM- oder EXE-Datei zusammenbinden. Microsoft liefert hierzu das Programm LINK.EXE beim Assembler mit. Bei Borland gibt es ein ähnliches Werkzeug (TLINK.EXE) für diese Aufgabe.

Um ein ausführbares Programm zu erhalten sind in der Regel folgende Schritte erforderlich:

- Erstellung des Quellprogramms mit einem Texteditor
- Übersetzung des Quellprogramms mit einem Assembler in eine OBJ- oder COM-Datei
- Falls OBJ-Dateien vorliegen sind diese mit einem Linker zu einer ausführbaren COM- oder EXE-Datei zu kombinieren

Weitere Einzelheiten lernen Sie in den folgenden Kapiteln kennen.

1.2 Das Hexadezimal-, Binär- und Dezimalsystem

Bei der Assemblerprogrammierung werden Zahlen und Konstante häufig im Hexadezimalsystem oder als Binärwerte verarbeitet. Die Umrechnung in Dezimalzahlen ist recht einfach. Die dezimale Zahl 123 setzt sich folgendermaßen zusammen:

```
1 * 100 + 2 * 10 + 3 * 1
```

Bei einer Binärzahl reduziert sich der Wertebereich für eine Ziffer auf 0 und 1. Der Wert 1011B läßt sich damit zu:

```
1 * 2**3 + 0 * 2**2 + 1 * 2**1 + 1 * 2**0
  8      +    0     +   2      +    1
              11 (dezimal)
```

darstellen. Jede Binärziffer ist mit der entsprechenden Wertigkeit (2**x) zu multiplizieren. Das Ergebnis der Addition ergibt den dezimalen Wert. Zur

Rückrechnung der Dezimalzahl in den Binärwert ist eine sukzessive Division durch 2 vorzunehmen:

```
11 : 2 = 5 Rest 1 (2 * 2**0)
 5 : 2 = 2 Rest 1 (2 * 2**1)
 2 : 2 = 1 Rest 0 (2 * 2**2)
 1 : 2 = 0 Rest 1 (2 * 2**3)

   11 = 1011B
```

Der Divisionsrest bestimmt die Ziffern der Binärzahl. Beim Hexadezimalsystem umfaßt der Bereich für eine Ziffer insgesamt 16 Werte:

Dezimal	Hexadezimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Tabelle 1.1: Umrechnung Hex-Dez

Der dezimale Wert 13 läßt sich somit als 0DH darstellen. Eine Umrechnung zwischen binären und hexadezimalen Werten ist recht einfach. Jeder Hexziffer entsprechen 4 binäre Stellen. Der Wert:

```
  3    7    F    D  H(exadezimal)
0011 0111 1111 1011 B(inär)
```

ist relativ einfach zwischen den Zahlensystemen umrechenbar. Eine Konvertierung zwischen Dezimal- und Hexadezimalsystem erfolgt durch Multiplikation oder Division der Ziffern mit dem Wert 16.

```
31FH = 3 * 16 **2 + 1 * 16**1 + 15 * 16 **0
     =     768    +    16     +    15
     = 799 (dezimal)
```

Die Rückwandlung einer Dezimalzahl in eine Hexzahl erfolgt durch Division mit 16:

```
69 : 16 = 4 Rest 5  (5 * 16 ** 0)
 4 : 16 = 0 Rest 4  (4 * 16 ** 1)
69      = 45H
```

Bezüglich der Darstellung der Zahlen durch den Prozessor (Byte, Wort, Integer, BCD, etc.) möchte ich auf die nachfolgenden Kapitel verweisen.

1.3 Die logischen Operatoren AND, OR, XOR und NOT

Mit diesen Operatoren lassen sich logische Verknüpfungen zwischen zwei Operanden durchführen. Sowohl der 80x86-Befehlssatz als auch die meisten Assembler kennen die logischen Befehle. Bei der AND-Verknüpfung wird jedes Bit der Operanden überprüft. Nur wenn beide Bits gesetzt sind, erhält das Ergebnis den Wert 1.

X1	X2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Tabelle 1.2: Die AND-Verknüpfung

Bei der OR-Verknüpfung wird das Ergebnis auf 1 gesetzt, falls eines der Eingangsbits den Wert 1 besitzt.

X1	X2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 1.3: Die OR-Verknüpfung

Bei der XOR-Verknüpfung wird ebenfalls jedes Bit der Operanden überprüft. Nur wenn beide Bits unterschiedliche Werte besitzen, erhält das Ergebnis den Wert 1.

X1	X2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 1.4: Die XOR-Verknüpfung

Bit der NOT-Operation wird einfach der Wert des Eingangsbits im Ergebnis invertiert weitergegeben.

Weitere Hinweise zu diesen Operationen finden sich in den nachfolgenden Kapiteln.

© Günter Born Einführung in die Assemblerprogrammierung