6. Übungsblatt:
Programmierpraktikum II (SS 2002)

Abgabe: 8. Juli 2002

Aufgabe 1: Vektoraddition

7 Punkte

Gesucht ist ein Programm, das zwei Vektoren der Länge n addiert. Gegeben sei die folgende Belegung des Speichers:

- In den Speicherzellen `6FFE` und `6FFF` steht die Vektorlänge n (setze voraus, dass $0 < n < 100$ immer erfüllt ist).
- Ab der Adresse 7000 befinden sich die n Wörter des Vektors A.
- Ab der Adresse 7200 befinden sich die n Wörter des Vektors B.

Schreibe ein Assembler-Programm, das die beiden Vektoren A und B wortweise addiert

$$\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

und den resultierenden Vektor C wortweise ab der Adresse 7400 abspeichert.

Bem.: Ausnahmsweise dürfen Überlauffehler, die bei der Addition zweier 16-Bit-Zahlen entstehen, ignoriert werden.

Aufgabe 2: Fibonacci-Zahlen

8 Punkte

Die Reihe der Fibonacci-Zahlen ist rekursiv durch die Gleichung

$$f(n) = f(n-1) + f(n-2)$$

definiert, wobei $f(1) = f(2) = 1$ ist. Die ersten Fibonacci-Zahlen sind also

$$1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Schreibe ein Assembler-Programm, das die ersten 40 Fibonacci-Zahlen berechnet und als Langwörter ab Zelle `$7000` in den Speicher schreibt. Eine Beispielausgabe könnte wie folgt aussehen:

```
00007000: 00 00 00 01 00 00 00 01 00 00 00 02 00 00 00 03
00007010: 00 00 00 05 00 00 00 08 00 00 00 0d 00 00 00 15
00007020: 00 00 00 22 00 00 00 37 00 00 00 59 00 00 00 90
00007030: 00 00 00 e9 00 00 01 79 00 00 02 62 00 00 03 db
...
```

Bem.: Wer sich jetzt wieder darüber wundert, dass die Zahlen im Speicher nicht mit der obigen Reihe der Fibonacci-Zahlen übereinstimmen, der sei nochmals daran erinnert, dass die Werte im Speicher als *Hexadezimalzahlen* dargestellt sind.

Assembler-Direktiven

Neben den Assembler-*Befehlen* wie `MOVE` oder `ADD`, die eigentlich nur leserliche Versionen der im Speicher abgelegten Maschinenbefehle sind, braucht man häufig auch noch sogenannte Assembler-*Direktiven* wie `ORG`, `DC` usw., auf die wir hier kurz eingehen wollen.

Im Gegensatz zu den Befehlen wenden sich die Direktiven direkt an das Assembler-Programm (und nicht an den Prozessor selbst). Sie werden also zur Assemblierzeit ausgeführt. Einige Beispiele:

- `ORG` (*origin*): Die Direktive `ORG n` weist dem Ortszähler (*location counter*) den Wert n zu. Auf diese Weise kann man z.B. angeben, an welche Stelle im Speicher der Assembler die nächsten Befehle oder Konstanten schreiben soll.
- `DC` (*define constant*): Diese Direktive wird in einer der Formen

  ```
  DC.B    d1, d2, ...
  DC.W    d1, d2, ...
  DC.L    d1, d2, ...
  ```

  verwendet und speichert die Konstanten `d1, d2, ...` ab der aktuellen Ortszähler-Adresse als Bytes, Worte oder Langworte.
- `END`: Diese Direktive ist selbsterklärend; sie markiert das Ende des Maschinenprogrammes.

Bsp. 1: Ein erstes, einfaches Beispiel für die Anwendung solcher Direktiven ist das Ablegen von Daten im Speicher, die vom Programm später benötigt werden. So führen die Direktiven

```
ORG     $7000
DC.W    $abcd, 50
```

dazu, dass im Speicher unter der Adresse `$7000` der Wert `$abcd` und unter der Adresse `$7002` der Wert `$0032` steht. Der aktuelle Stand des Ortszählers ist jetzt `$7004`. Es empfiehlt sich, bei Aufgabe 1 zur Initialisierung der Vektoren A und B ähnlich vorzugehen.

Bsp. 2: Mit diesen Befehlen machen nun auch die ersten vier Zeilen des Beispielprogrammes auf Übungsblatt 5 Sinn:

```
        ORG     $0          ;Setze Ortszaehler auf $0000.
        DC.L    $8000       ;Schreibe ($0000) = 00 00 80 00,
                            ;das ist der Wert des Stack Pointers
                            ;im Falle eines Resets.
        DC.L    START       ;Schreibe ($0004) = 00 00 20 00,
                            ;das ist der Wert des Program
                            ;Counters im Falle eines Resets.
        ORG     $2000       ;Setze Ortszaehler auf $2000.
START   ...                 ;Speichere Programm ab $2000.
```

Die Adressen `$0000` bzw. `$0004` sind also diejenigen, unter denen der M68000 im Falle eines Resets die Werte des Stack Pointers und des Program Counters sucht; siehe dazu im Referenzhandbuch den Anhang B1[1]. Normalerweise können diese ersten Zeilen für jedes Programm, das wir in diesem Kurs schreiben, übernommen werden.

Weiterführendes: Wer weitere Informationen zu diesen oder anderen Direktiven sucht, der sei an das Manual zum `68kasm` verwiesen, das sich im BSVC-Verzeichnis unter `doc/68kasm/68kasm.doc` findet.

[1] Aus dem gleichen Grund ist auf dem 6. PI-2-Übungsblatt, Aufgabe 2, die 2. Zeile des Beispielprogramms vermutlich falsch.