Arbeitsgruppe Eingebettete Systeme und Betriebssysteme

Vorlesung Rechnersysteme

Übungsblatt 9

ab 15. Juni 2010

Hinweis: Nutzen Sie bei der Erstellung der Programme den Easy68k. Bringen Sie das fertige Programm zur Demonstration auf einem USB-Stick mit.

Aufgabe 1

Entwickeln Sie aus der Vorlage `sinus_Vorlage.X68` auf der Übungsseite ein Assemblerprogramm, das für ganzzahlige positive Werte von 0–90 Grad den zugehörigen Sinuswert mithilfe einer Look-Up-Tabelle berechnet. Schätzen Sie die Größe des Interpolationsfehlers ab!

Anmerkung

Wie in der Vorlage ersichtlich, ist eine Look-Up-Tabelle zu verwenden. Dazu sind n Funktionswerte $f(x_k)$ mit $k < n$ in einem Array als Stützstellen der Funktion bereitgestellt. Die Zahl der Stützstellen über dem Intervall $0 \leq k \leq n$ hängt von der Auflösung r ab. Wenn also der Wert für ein x bestimmt werden soll, wird zunächst die Position der beiden benachbarten Stützstellen $(f(x_i), f(x_{i+1}))$ im Array bestimmt und der Punkt dann (linear) interpoliert. Die Position i ergibt sich als Ganzzahldivision des Eingabewertes x mit der Auflösung r.

$$i = \text{floor}(x/r)$$
$$f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \cdot (x - x_i)$$

Für unsere Fragestellung könnte man also ein Array mit den Einträgen für sin(x) erstellen. Wie in der Vorlage ersichtlich, bedienen wir uns dafür der ersten vier Nachkommawerte in `base`. Allerdings wird anhand der vorhergehenden Gleichnung für $f(x)$ deutlich, dass der erste Faktor ebenfalls eine Konstante ist. Um die Berechnungen zu reduzieren, wird auch hier eine Look-Up-Tabelle benutzt (`corr`). Die Auflösung r der Look-Up-Tabelle beträgt 6 Grad, für 90 Grad also 16 Einträge. Der Sinus von α berechnet sich also für unsere Fragestellung zu:

$$
\begin{aligned}
i &= \text{floor}(\alpha/r) \\
\text{rem} &= \text{remainder}(\alpha/r) \\
\sin(\alpha) &= \text{base}_i + \text{corr}_i \cdot \text{rem}
\end{aligned}
$$

Aufgabe 3

Erweitern Sie die vorgegebenen Assemblerdatei `matrixmult_Vorlage.X68` um ein Unterprogramm, das zwei $n \times n$-Matrizen multipliziert. Die Matrizen enthalten vorzeichenbehaftete Integerzahlen, die je in zwei Byte gespeichert werden.
Hinweise zur Programmierung:

- Ansatz: Konzentrieren Sie sich zunächst auf den konzeptionellen Algorithmus, brechen Sie ihn herunter auf Einzelschritte und gehen Sie erst dann über zur Implementierung.
- Die Speicherorte der Matrizen sind bereits vorgegeben.
- Speichern Sie das Ergebnis hinter dem symbolischen Namen MatrixC.
- Gestalten Sie Ihre Subroutine so, dass sie durch den Matrixparameter n (im Programm `N`) parametrisierbar ist.
- Vergessen Sie nicht, Ihren Programmabschnitt aussagekräftig zu kommentieren.

Testen Sie Ihr Programm nicht nur, aber auch mit der Multiplikation folgender Matrizen:

$$
\begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 3 \\ 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} 2 & -2 & 5 \\ 3 & 0 & -1 \\ 3 & 2 & 10 \end{pmatrix}
$$

Aufgabe 4

Gib für das in Quellcodeauszug 1 dargestellte Assemblerprogramm die verwendeten Kontrollstrukturen an (*jeweils Zeilennummer und Kontrollstruktur*) und stelle das Programm in C-Syntax dar!

Aufgabe 4

1. In der Vorlesung wurde die durchschnittliche Speichernutzung einer CISC und RISC Architektur gegenübergestellt (Folie 12). Welche Erkenntnisse lassen sich aus der Darstellung gewinnen und wie sind diese zu begründen?

```
1             ORG $1000
2   ARRAY     RMB     10          ; 10 Byte Array
3   MAIN      LDA     #10
4             PSHS    A
5             LEAX    ARRAY
6             PSHS    X
7             BSR     SORT
8             LEAS    2, S
9   END       BRA     END
10  SORT      PSHS    U
11            LEAU    , S
12            LEAS    -1, S
13            PSHS    X
14  L1        LDA     #1
15            STA     -1, U
16            LDX     2, U
17  L2        PSHS    A
18            LDA     , X+
19            CMPA    , X
20            BHS     L3
21            PSHS    X
22            LEAX    -1, X
23            PSHS    X
24            BSR     SWITCH
25            LEAS    1, S
26            PULS    X
27            LDA     #0
28            STA     -1, U
29  L3        PULS    A
30            INCA
31            CMPA    3, U
32            BNE     L2
33            LDA     -1, U
34            CMPA    #1
35            BNE     L1
36            LEAS    -2, U
37            PULS    X
38            LEAS    , U
39            PULS    U
40            RTS
41  SWITCH    PSHS    U
42            LEAU    , S
43            PSHS    X
44            PSHS    Y
45            LDX     2, U
46            LDY     3, U
47            LDA     , X
48            PSHS    A
49            LDA     , Y
50            STA     , X
51            PULS    A
52            STA     , Y
53            PULS    Y
54            PULS    X
55            PULS    U
56            RTS
57            END
```

Listing 1: Zu analysierendes Assemblerprogram

2. Warum benötigen RISC-Rechner mehr Register als CISC-Rechner?

3. Erklären Sie den Begriff Load and Store Architektur kurz und prägnant!

Aufgabe 5

Die Abarbeitung eines Programmes für einen (fiktiver) RISC Prozessor ABC123 und eines funktional equivalenten Programmes für einen 68020 (25 MHz) soll die gleiche CPU Zeit dauern. Mit welchem Takt muss der RISC-Prozessor laufen, um diesen Anspruch zu erfüllen? Die Programme wurden statistisch analysiert, dabei wurden die in der Tabelle dargestellten Werte ermittelt. Zur Berechnung kann für die Zyklen pro Instruktion der Mittelwert des angegeben Bereiches angenommen werden.

	68020	ABC123
IC	1.0	1.3
TD	40 ns	
CPI	5-7	1.3-1.7

Welche Erkenntnis läßt sich also aus der Angabe einer Taktfrequenz für einen Rechner ziehen?