Rechnerorganisation

Sommersemester 2023

Prof. Stefan Roth, Ph.D.

TECHNISCHE UNIVERSITÄT DARMSTADT

3. Aufgabenblatt

08.05.2023
Konzepte der maschinennahen Programmierung, C-Programmierung

Aufgabe 1: Theoriefragen

(a) Wie kann mit Maschinenbefehlen die Multiplikation mit 2 und die Division mit 2 effizient implementiert werden? Wie heißen diese Maschinenbefehle bei einer ARM-Architektur?

(b) Mittels welcher Flags werden Verzweigungen im Programmablauf realisiert?

Aufgabe 2: Implementierung von Hochsprachenkonstrukten in Assembler

Übersetzen Sie folgende Hochsprachenkonstruke in ARM-Assembler. Kommentieren Sie Ihren Assemblercode.

(a)
```
if (a == o)
  f = i + 1;
else
f = f - i;
```

Geben Sie das Assemblerprogramm ein und testen Sie das Programm mit den folgenden Werten:

- a = 1
- o = 2
- f = 5
- i = 1

Als Rahmen können Sie das Programm aus der 2. Übung verwenden. Schreiben Sie f als Rückgabewert in das Register r0.

(b)
```
int pow = 1;
int x = 0;
while (pow != 256) {
pow = pow * 2;
x = x + 1;}
```

Aufgabe 3: Makefiles

Hinweis: Makefiles sind nicht klausurrelevant, erlauben aber ein effizienteres Arbeiten bei den weiteren Übungen.

Damit die Befehle zum Kompilieren nicht jedes mal erneut eingegeben werden müssen, gibt es eine einfache Methode, die sich Makefiles nennt. Hier muss zum Assemblieren und Kompilieren nur ein einfacher Befehl in die Kommandozeile eingegeben werden. Diejenigen Befehle, die sonst jedes mal erneut einzeln eingegeben werden müssten, werden in eine Datei geschrieben.

Es funktioniert folgendermaßen:

1. Erstellen Sie eine Datei mit dem Namen $Makefile$
2. Schreiben Sie in die erste Zeile $all:$
3. Schreiben Sie in die darauf folgenden Zeilen die Befehle, die Sie ausführen wollen.
4. Führen Sie in der Shell den Befehl *make* aus, dadurch werden die einzelnen Befehle welche sich in dem Makefile befinden, ausgeführt.

Zur Erinnerung: die Befehle zum Assemblieren und Linken eines Assemblerprogramms `example.s` sind

```
arm-linux-gnueabihf-as -o example.o example.s
```

```
arm-linux-gnueabihf-gcc -o example example.o
```

Dies war nur eine kurze Einführung in Makefiles. Ausführlichere Tutorials gibt es in den Materialien auf Moodle und an verschiedenen Stellen im Internet.

Aufgabe 4: Einführung in C

Die Programmiersprache C wurde 1969 in den Bell Laboratories entwickelt. Wie so oft führte Unzufriedenheit mit einer Sprache, in diesem Fall B(CPL)[1] zur Entwicklung der Sprache C. Die Entwickler waren Dennis Ritchie und Ken Thompson. Letzterer war auch an der Entwicklung des Betriebssystems Unix beteiligt. In den 70er Jahren gab es eine Vielzahl von C-Dialekten. Die Lösung war die Schaffung eines C-Standards. Von Brian W. Kernighan und Dennis Ritchie wurde 1978 das Buch „The C Programming Language“ veröffentlicht. Daraus wurde der K & R Standard entwickelt, der dann im ANSI[2] C überführt wurde. Warum nutzt man heute noch die Sprache C?

- Die Sprache ist einfach aufgebaut – Grundlage für viele andere Sprachen (in Syntax und Semantik)
- Im Bereich der Eingebetteten Systeme und Unix/Linux hohe Verbreitung
- Maschinennahe Programmierung einfach möglich – z. B. Schreiben von Inline-Assembler-Code

Eine kurze Einführung in C sowie eine Literaturliste ist im Foliensatz Eine kurze Einführung in die Programmiersprache C zu finden.

Im Folgenden das Standardprogramm, welches in jeder Programmiersprache zum Testen verwendet wird.

```
#include<stdio.h>
int main(){
printf("Hello␣World\n");
return 0;
}
```

Zum Übersetzen eines C Programms soll im Folgenden der GCC Compiler benutzt werden. Die Übersetzung der Datei mit dem Namen hello.c wird durch folgende Befehle erreicht. Achtung: sh$ muss nicht eingegeben werden.

```
sh$ gcc -o hello hello.c        (compilieren)
sh$ ./hello                     (Ausführen des Executables)
```

Der optionale Parameter -o mit der folgenden Zeichenkette hello gibt den Namen der übersetzten (und ausführbaren) Datei an.

Gegeben ist folgendes C-Programm:

```
/* What-Do-I-Do */
#include<stdio.h>

int main(){

short int i = 1;
short int n = 32767;
for (i;i<=n;i++)
printf("i␣ist␣%i␣\n",i);
return 0;
}
```

[1] Basic Combined Programming Language
[2] American National Standard Institute

Was passiert bei der Ausführung des Programms?

Aufgabe 5: Fehlersuche

Übersetzen Sie das folgende C-Programm in ARM-Assembler und führen Sie es aus. Prüfen Sie anschließend dessen Rückgabe mit `echo $?`. Entspricht das ausgegebene Ergebnis Ihren Erwartungen? Worin könnten eventuelle Unstimmigkeiten begründet sein?

```
int main() {
   int a = 53;
   int b = a << 4;
   int c = b + 5;
   return c;
}
```

Aufgabe 6: Monte-Carlo-Simulation

Monte-Carlo-Simulationen können zur Berechnung des Inhalts regel- und unregelmäßiger Flächen und Körper verwendet werden. Die Algorithmen werden als Monte-Carlo-Algorithmen bezeichnet und gehören in die Klasse der randomisierten Algorithmen. Das Verfahren der Monte-Carlo-Simulation soll anhand der probabilistischen Bestimmung der Kreiszahl π verdeutlicht werden. Hierzu werden zufällige Punkte $\{(x, y)|x \in [-1, 1], y \in [-1, 1]\}$ generiert und es wird überprüft, ob diese innerhalb des Einheitskreises liegen. Die sich ergebende Wahrscheinlichkeitsverteilung erlaubt die Berechnung der Kreiszahl π nach folgender Formel.

$$\frac{\text{Treffer in Kreisflaeche}}{\text{generierte Punkte im Rechteck}} = \frac{r^2 \cdot \pi}{(2 \cdot r)^2} = \frac{\pi}{4}$$

a) Schreiben Sie ein C Programm, welches die Kreiszahl π berechnet und ausgibt. Die Funktion rand() liefert Zufallszahlen.[4]

b) Testen Sie Ihr Programm mit unterschiedlicher Anzahl von Punkten (im Bereich von ca. 100000 bis 100000000). Notieren Sie die Anzahl der Punkte und die Genauigkeit in einer Tabelle.

c) Die Berechnung von π kann auch auf ein Flächenintegral zurückgeführt werden.

$$\int_0^1 \frac{4}{1+x^2} dx$$

[4] Binden Sie falls notwendig die entsprechenden Librarys ein. Achten Sie bei rand() darauf vorher einen zufälligen Seed zu setzen.

Schreiben Sie ein C Programm, welches die Integration (z. B. Trapezintegration) ausführt und vergleichen Sie die Ergebnisse mit b).