Aufgabe 0

1 Lernziel

- Erste hardwarenahe Programmierung
- Beziehung zwischen Hochsprache und Maschinenebene

2 Aufgabenbeschreibung

Bei der Entwicklung von Betriebssystemen ist es unvermeidbar sich mit der Maschinenebene auseinander zu setzen. Teile des Systems, die sich nicht in einer Hochsprache umsetzen lassen, müssen in Maschinensprache implementiert werden. Als Vorbereitung darauf, wird in dieser Aufgabe ein Assembler-Programm erstellt.

3 Theoretische Aufgaben

3.1 Vorbereitung

Informiert euch über Assembler- und C-Programmierung. Dazu findet ihr einen Assembler-Crashkurs auf der Lehrstuhl-Webseite. Besorgt euch weiteres Material aus dem Internet, in der Bibliothek oder in den Übungen.

3.2 Kontrollfragen

Versucht, die folgenden Fragen zu beantworten **bevor** ihr mit der Implementierung beginnt.

1. Was sind Register?
2. Was ist ein Stack? Welche Operationen stellt er mindestens zur Verfügung?
3. Was ist der Stack-Pointer? In welchem Register wird er gespeichert?
4. Wie beeinflussen die Stack-Operationen den Stack-Pointer?
5. Was ist der Base-Pointer? In welchem Register wird er gespeichert?

6. Was ist ein Debugger?
7. Was ist ein Breakpoint?
8. Was sind Calling-Conventions? Wozu werden sie benötigt?
9. Wie sind die Calling-Conventions des gcc auf der x86 Architektur?
10. Worin besteht der Unterschied zwischen flüchtigen und nicht-flüchtigen Registern?
11. Was bedeutet das Schlüsselwort **extern** in der Programmiersprache C?

4 Praktische Aufgaben

4.1 Vorbereitung

4.1.1 Übersetzen und Linken

- Ladet die Vorgabe (main−nasm.asm) herunter.
- Übersetzt die Vorgabe mit nasm −f elf −o main.o main−nasm.asm
- Erzeugt eine ausführbare Datei mit gcc −m32 −o run main.o
- Startet die erzeugte Datei mit ./run

4.1.2 Debuggen

Mit dem einfachen Assembler-Programm ist es nicht möglich, Informationen z.B. auf der Konsole auszugeben. Um die Korrektheit des Programmes zu untersuchen oder Fehler zu finden, kann ein Debugger verwendet werden.

- Übersetzt und linkt die Vorgabe mit Debug-Symbolen. Dazu müssen die oben genannten Befehle um ein Flag erweitert werden. Findet das Flag mit Hilfe der Manpages beider Programme (man nasm und man gcc).
- Startet den Debugger mit ddd run
- Macht euch mit dem Debugger vertraut. Versucht einen Breakpoint zu setzen, das Programm zu starten und die Inhalte von Registern anzusehen.

4.2 Implementierung eigener Funktionen

Das zu implementierende Programm besteht aus zwei Teilen: dem Hauptprogramm und einer Funktion, welche die Anzahl ganzzahliger Teiler einer Zahl bestimmt. Der Rahmen des Haupt-

programmes ist bereits vorgegeben. Dort ist der Aufruf der Funktion zu implementieren. Es sind zwei Varianten der Funktion zu implementieren:

1. ohne Benutzung eines Basiszeigers, mit while-Schleife und folgender Signatur:
   `int getDivisorCount(unsigned int value)`

2. mit Benutzung eines Basiszeigers, mit for-Schleife und folgender Signatur:
   `void getDivisorCount(int value, int* result)`

Achtet hierbei auch auf die Datentypen beim Einlesen der Eingabe.

4.3 C-Anbindung

Schreibt ein C-Programm, welches die in Assembler implementierten Funktionen aufruft. Dieses soll die main-Funktion des Programms enthalten. Beim Start soll eine Zahl vom Benutzer eingegeben werden. Diese wird an die Assembler-Funktionen übergeben, welche die Anzahl der Teiler berechnen. Am Ende sollen die Ergebnisse beider Funktionen auf der Konsole ausgegeben werden. **Wichtig:** Das C-Programm soll lediglich die Ein- und Ausgabe, sowie den Aufruf der Assemblerfunktionen behandeln. Sämtliche Berechnungen sind in den Assembler-Funktionen zu implementieren.

Beachtet, dass die Assembler-Vorgabe bereits den Rumpf der main-Funktion enthielt. Der entsprechende Code muss deshalb entfernt werden.

Die Übersetzung von Assembler- und C-Code muss getrennt erfolgen. Der erste Teil des Übersetzens, bleibt erhalten: `nasm -f elf -o main.o main-nasm.asm` Für die C-Datei (main.c) benutzt folgenden Befehl: `gcc -m32 -c -o c.o main.c`. Die erstellten Object-Dateien (*.o) müssen nun noch zu einer einzigen ausführbaren Datei gelinkt werden: `gcc -m32 -o run main.o c.o`. Nun könnt ihr das Programm wieder mit `./run` starten.

4.4 Vorbereitung der Abnahme

Testet eure Implementierung ausführlich und vervollständigt Kommentare. Bereitet euch auf die Abnahme vor. Informationen zu deren Ablauf findet ihr auf der Lehrstuhl-Webseite.

Viel Erfolg!