Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Sprachtechnologie und Compiler WS 2009/2010 http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Sebastian Buchwald sebastian.buchwald@kit.edu

Übungsblatt 11 | Ausgabe: 14.1.2010 | Besprechung: 21.1.2010

Aufgabe 1: *Zugriff auf nichtlokale Stackdaten*

1.1 *Zugriffslinks*

- Welches Problem beim Zugriff auf Daten lösen Zugriffslinks?
- Warum tritt das Problem in C und Java nicht auf?
- Kennen Sie Programmiersprachen in denen das Problem auftritt?

1.2 *Activation Records*

Nehmen Sie an C unterstütze verschachtelte Funktionen [1]. Betrachten Sie folgendes Programm:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    int fib0(int n) {
        int fib1(int n) {
            int fib2(int n) {
                return fib1(n-1) + fib1(n-2);
            }

            if (n >= 4)
                return fib2(n);
            return fib0(n-1) + fib0(n-2);
        }

        if (n >= 2)
            return fib1(n);
        return 1;
    }

    int v = 4;
    if (argc > 1)
        v = atoi(argv[1]);
    printf("fib(%d) = %d\n", v, fib0(v));
    return 0;
}
```

- Nehmen Sie an, dass das Programm ohne Parameter gestartet wurde. Zeigen Sie den Stack der Activation Records, zum Zeitpunkt, wenn der Aufruf von fib0 mit Parameter 1 zurückkehrt. Geben Sie den Zugriffslink in jedem der Aktivierungseinträge auf dem Stack an.

1.3 *Displays*

[1] In der Tat, besitzt der gcc-Compiler eine solche Erweiterung. Das Beispiel ist dort ohne Änderungen lauffähig.

- Angenommen wir implementieren die Funktionen aus der letzten Aufgabe mit Displays. Geben Sie das Display zu dem Zeitpunkt an, in dem der Aufruf von fib0 mit Parameter 1 zurückkehrt. Kennzeichnen Sie außerdem den gespeicherten Displayeintrag zu diesem Zeitpunkt in jedem der Activation Records auf dem Stack.

Aufgabe 2: *Implementierung*

Die RISC-Architektur aus dem letzten Übungsblatt wird um 2 Register erweitert und eine Aufrufkonvention eingeführt:

- Das Register IP „Instruction-Pointer" ist der Befehlszeiger. Er zeigt auf den gerade Ausgeführten Befehl.
- Das Register LR „Link-Register" enthält nach Aufrufkonvention die Rücksprungadresse bei einem Funktionsaufruf.
- Das Register SP „Stack-Pointer" zeigt auf die Spitze des Stacks. Der Stack wächst nach Konvention nach oben also von niedrigen zu hohen Speicheradressen.
- Das Register GPT „General-Purpose Temporary" ist reserviert für Hilfsinstruktionen des Assemblers (siehe unten).
- Die Parameterübergabe soll bei Funktionsaufrufen komplett über den Stack erfolgen.
- Für den Rückgabewert wird das Register GP1 verwendet.
- Eine Funktion darf alle Register beliebig modifizeren; GP1, GP2, ... sind also Caller-Save.

Wiederholung der Instruktionen:

mov R1, R2	Kopiert den Inhalt von Register R2 in Register R1
jmp label	Springt an die Stelle "label"
jmp R1	Springt an die Stelle im Programm die im Register R1 steht
bz R1, label	springe an die Stelle "label" falls die Register R1 den Wert 0 hat
bnz R1, label	Springe an die Stelle "label" falls das Register R1 einen Wert ungleich 0 hat
slt R1, R2, R3	Setzt Register R1 auf 1, falls der Inhalt von Register R2 kleiner als der von R3 ist, sonst auf 0
seq R1, R2, R3	Setzt Register R1 auf 1, falls der Inhalt der Register R2 und R3 gleich ist, sonst auf 0
add R1, R2, R3	Addiert den Inhalt von R2 und R3 und schreibt das Ergebnis in Register R1
mul R1, R2, R3	Multipliziert den Inhalt von R2 und R3 und schreibt das Ergebnis in Register R1
shl R1, R2, R3	Die Bits in Register R2 werden um den Wert im Register R3 nach links verschoben, das Ergebnis wird in R1 geschrieben
ld R1, R2	Lädt 4 Byte aus dem Hauptspeicher von Adresse R2 und speichert das Ergebnis in Register R1
st R1, R2	Schreibt 4 Byte des Wertes von R2 in den Hauptspeicher an Adresse R1
ldb R1, R2	Lädt 1 Byte aus dem Hauptspeicher von Adresse R2 und speichert das Ergebnis in Register R1
stb R1, R2	Schreibt 1 Byte des Wertes von R2 in den Hauptspeicher an Adresse R1
label:	Assembler Pseudobefehl um die Programmstelle mit dem Namen "label" zu versehen.
.long C	Assembler Pseudobefehl der den Wert der Konstante C ins Programm einsetzt (für C kann auch das Label einer Programmstelle angegeben werden)

2.1 *Pseudo-Instruktionen*

Es werden einige Hilfsinstruktionen eingeführt um das Programmieren zu erleichtern:

push R1	Schreibt den Wert von Register R1 auf den Stack und inkrementiert SP um 4.
pop R1	Dekrementiert SP um 4, liest den Wert an der Stackspitze und schreibt ihn in R1.
call label	Ruft Funktion bei label auf.
return	Kehrt aus Funktion zurück.

Wie könnte eine Implementierung der Hilfsfunktionen aussehen?

2.2 *Assembler, Zugriffslinks*

Übersetzen Sie folgendes Programm in Assembler:

```
typedef int (*function_pointer)(int parameter);
void a(int x) {
    int b(function_pointer f) {
        /* ... */ int res = f(x); /* ... */
        return res;
    }
    void c(int y) {
        int d(int z) {
            /* ... */ int res = z * 42; /* ... */
            return res;
        }
        /* ... */
        b(d);
        /* ... */
    }

    c(1);
}

int main(void)
{
    a(1);
    return 0;
}
```