Prof. Dr. Rolf Drechsler, drechsler@informatik.uni-bremen.de, MZH 4330
Christina Plump, cplump@uni-bremen.de, MZH 4206

2. Übungsblatt zur Vorlesung

Technische Informatik 1

Aufgabe 1 (2 Punkte)

Als Zwischenform von Compilern und Interpretern kann der sogenannte *Bytecode* verstanden werden. Hierbei wird eine Hochsprache von einem Compiler in eine Zwischenansprache gebracht und kann dann von einem passenden Interpreter verarbeitet werden. Beschreibt die Vor- und Nachteile, die ein solcher (*hybrider*) Ansatz im Gegensatz zur reinen Verwendung eines Compilers oder Interpreters hat und gebt ein Beispielszenario an, bei dem Ihr Bytecode verwenden würdet.

Aufgabe 2 (3 Punkte)

Ihr wurdet damit beauftragt, für das neueste Smart-Home Hub den Prozessor zu entwerfen. Hierzu müsst Ihr Euch vor dem Entwurf entscheiden, ob Ihr einen RISC oder einen CISC verwendet. Nennt kurz die Eigenschaften von RISC und CISC und begründet dann in einer kurzen Stellungnahme, für welches der beiden Modelle Ihr Euch entscheidet. Eure Entscheidung soll am Ende eindeutig auf die eine oder die andere Architektur fallen und nachvollziehbar sein.

Aufgabe 3 (2 + 2 Punkte)

Für ein spezielles Einsatzgebiet soll ein neuer Rechner entworfen werden. Die Rechengenauigkeit aller Operationen soll 16 Bit betragen. Aufgrund der vorkommenden Operationen werden 95 Instruktionen benötigt. Der Rechner hat 28 Register bei einem Adressraum von maximal 68K. Es wird eine Load/Store-Architektur verwendet: Befehle, die auf den Speicher zugreifen, haben zwei Register als Operanden (Befehl 1: `Rdest := Mem[Rsrc]`, `Befehl 2: Mem[Rsrc] := Rdest`). Kein Befehl hat mehr als drei Operanden, mindestens zwei Operanden sind Register, der Dritte kann eine 8-Bit-Konstante oder ein Register sein.

a) Wie breit müssen die Register mindestens sein?

b) Wie viele Bits werden benötigt, um eine Instruktion (Befehlswort + Operanden) zu kodieren?

Begründet Eure Überlegungen.

Aufgabe 4 (3 + 2 Punkte)

a) Im Befehlssatz von RISC-V gibt es keine eigenen Befehle für `NOT` (d.h. eine bitweise `NOT`-Operation), `MOV` (Datentransport von Register zu Register), und `ROTL` (eine *logische* Rotation nach links).

Die Funktionalität dieser Befehle ist wie folgt gegeben:

- `NOT Rdest, Rsrc` ⇔ `Rdest := ~Rsrc`
- `MOV Rdest, Rsrc` ⇔ `Rdest := Rsrc`
- `ROTL Rdest, Rsrc, imm` ⇔ `Rdest := Rsrc[31-imm,30-imm,...,0,31,30,...,imm]`

Gebt jeweils ein kurze Folge von RISC-V-Befehlen an, mit der die gewünschte Funktionalität dennoch realisiert werden kann.

b) Betrachtet die RISC-V-Befehle `jal` (unbedingter Sprung) und `beq` (bedingter Sprung). Bei diesen wird der Sprung relativ zum Program Counter durchgeführt, d.h. der Wert des angegebenen Offsets wird zum aktuellen Wert des Program Counters hinzuaddiert. Wie groß ist der Speicherbereich (Adressraum), der sich auf diese Weise jeweils mit einem Sprung erreichen lässt? Bei welchem der beiden Befehle ist dieser Bereich größer?

Aufgabe 5 (6 Punkte)

Beim *Collatz-Problem* betrachtet man Zahlenfolgen, die wie folgt definiert sind:

$$a_0 \in \mathbb{N}^+, \qquad a_n := \begin{cases} \frac{a_{n-1}}{2} & a_{n-1} \text{ gerade} \\ 3 \cdot a_{n-1} + 1 & a_{n-1} \text{ ungerade} \end{cases} \text{(für } n > 0)$$

Die bislang unbewiesene Vermutung (auch $(3n+1)$-Vermutung genannt) lautet, dass diese Folgen für beliebiges $a_0 \in \mathbb{N}^+$ in den Zyklus $4, 2, 1$ münden. Startet man z. B. mit $a_0 = 7$, ergibt sich die Zahlenfolge

$$(a_0, a_1, \ldots) = (7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4 \ldots)$$

Wir wollen nun zu einem gegebenen $a_0 \in \mathbb{N}^+$ den Zykluseintritts-Index n^* bestimmen, für den zum ersten Mal $a_{n^*} = 4$ gilt (formal: $n^* := \min\{n \in \mathbb{N}^+ | a_n = 4\}$). Dafür haben wir folgendes Programm geschrieben:

```
int collatz(int a_0) {
    int a_n=a_0;
    int nStar=0;
    while (a_n != 4) {
        if (a_n % 2 == 0) {
            a_n=a_n / 2;
        }
        else {
            a_n=3*a_n+1;
        }
        nStar=nStar+1;
    }
    return nStar;
}
```

Gebt ein RISC-V Programm an, das den Zykluseintritts-Index n^* bestimmt. Euer Programm soll nur Befehle aus RV32I nutzen, d.h. insbesondere keine Multiplikation, keine Division und keine Modulo-Operation benutzen.

Geht bei Eurer Implementierung davon aus, dass der Parameter a_0 zu Beginn in dem Register $a0$ zur Verfügung steht, und speichert **das Ergebnis im Register** $a1$. Euer Programm sollte nicht mehr als 10.000 Zyklen brauchen, um zu terminieren. Testet Euer Programm.

Hinweis: Quellcode, der nicht im *Ripes*-Simulator assembliert oder nicht ausreichend dokumentiert ist, wird mit 0 Punkten bewertet! Das Assembler-Programm ist als separate ASCII-Textdatei (keine Umlaute!) mit der Endung `.s` an Eure Tutoren zu senden. Hierbei gilt die gleiche Frist wie für den restlichen Übungszettel.

Ripes findet Ihr hier: `https://github.com/mortbopet/Ripes`

Eine Kurzübersicht über den RISC-V-Befehlssatz ist z. B. unter folgender URL zu finden:

`https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf`

Abgabe: bis Donnerstag, den 04.05.2023, 08:15 Uhr per e-Mail an den Tutor und Christina Plump mit folgendem Betreff: [TI-Abgabe] <Tutorium> - <Gruppe>: Blatt <Blatt>