ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

INSTITUT FÜR INFORMATIK

Dr. Jürgen Ruf
Dipl.-Inf. Ilia Polian

Freiburg, 4. Juni 2002

4. Übungsblatt zur Vorlesung

Rechnerarchitektur

Aufgabe 1

Das Skalarprodukt zweier Vektoren der Dimension 4 soll ausgerechnet werden. Zu zwei Vektoren (a_1, a_2, a_3, a_4) und (b_1, b_2, b_3, b_4) ist der Wert $c := a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 + a_4 \cdot b_4$ gesucht.
Es stehen verschiedene Rechner zur Verfügung, um diese Berechnung durchzuführen. Die Werte a_i stehen linear angeordnet (nacheinander) im Speicher, die Werte b_i ebenfalls. Zur Vereinfachung wird davon ausgegangen, daß der Speicher Byte-adressiert ist und daß alle Zahlen ein Byte groß sind (d. h. die Adresse von b_3 ist genau die Adresse von b_1 plus zwei). Im folgenden werden die einzelnen Rechner und relevante Teile ihres Befehlssatzes vorgestellt. Schreiben Sie jeweils ein Programm in der Maschinensprache, welches das Skalarprodukt berechnet.

a) Der Rechner hat 15 frei benutzbare Register `Reg1..Reg15`. Die Befehle `ADD` und `MUL` haben 3 Argumente:
`ADD A B C` # A := B + C
`MUL A B C` # A := B · C
Zulässige Adressierungsarten sind Register-Adressierung (`Reg`*i* ist für `A`, `B` und `C` zulässig), indirekte Register-Adressierung (`(Reg`*i*`)` ist ebenfalls für `A`, `B` und `C` zulässig) und absolute Zahlenwerte (`#Zahl` ist für `B` und `C` zulässig).

   Im Register `Reg1` steht die Adresse von a_1 im Speicher, im Register `Reg2` steht die Adresse von b_1 im Speicher. Im Register `Reg3` steht die Adresse im Speicher, an die der Wert c geschrieben werden soll.

b) Wie Punkt (a), aber eine weitere Adressierungsart ist bei allen Argumenten zulässig: Displacement `d(Reg`*i*`)` (Inhalt von Speicher an der Adresse `Reg`*i* `+ d`, wobei `d` eine Zahl ist).

c) Wie Punkt (b), aber eine weitere Adressierungsart ist bei allen Argumenten zulässig: Inkrement `(Reg`*i*`)+`. Dabei wird beim ersten Zugriff der Speicher an der Adresse referenziert, die im `Reg`*i* steht, beim zweiten an der Adresse `Reg`*i* `+ 1`, beim dritten an der Adresse `Reg`*i* `+ 2`, ...

d) Wie im Punkt (c), jedoch arbeiten die Befehle `ADD` und `MUL` nur auf Registern. Für den Speicherzugriff gibt es die Befehle
`LOAD A B`
`STORE A B`
`A` muß ein Register sein, `B` muß eine Speicheradresse sein (Register indirekt, Displacement, Inkrement). `LOAD` lädt den Speicherwert aus Adresse `B` in das Register `A`, `STORE` speichert den Inhalt des Registers `A` an der durch `B` gegebene Speicheradresse.

e) Die Maschine hat nur ein Register `ACC`. Alle Befehle haben nur ein Argument mit folgender Semantik:

```
ADD A      # ACC := ACC + A
MUL A      # ACC := ACC · A
LOAD A      # ACC := A
STORE A      # Speicher[A] := ACC
```

Alle bisher verwendeten Adressierungsarten sind erlaubt. Die Adressen von a_1, b_1 und c sind explizit gegeben als `Addr1`, `Addr2` und `Addr3`.

f) Es ist ein *Stack* vorhanden. Es gibt die beiden Funktionen für den Speicherzugriff:

```
PUSH A      # Wert an der Adresse A im Speicher auf den Stack legen
POP A       # Wert vom Stack nehmen und in den Speicher an der Adresse A schreiben.
```

Die Funktionen `ADD` und `MUL` haben keine Parameter. Sie nehmen die zwei obersten Werte vom Stack, berechnen die Summe bzw. Produkt und legen das Ergebnis auf den Stack (die Größe von Stack wird somit um 1 erniedrigt).

Bei `PUSH` und `POP` sind alle bisher verwendeten Adressierungsarten erlaubt. Die Adressen von a_1, b_1 und c sind explizit gegeben als `Addr1`, `Addr2` und `Addr3`.

Aufgabe 2

Klassifizieren Sie die Rechnertypen aus Aufgabe 1.

Aufgabe 3

Überlegen Sie sich mögliche Kodierungen für den Befehl `ADD` in den verschiedenen Rechnertypen aus Aufgabe 1 (vergessen Sie dabei nicht die evtl. verschiedenen Adressierungsarten). Gehen Sie davon aus, daß es neben `ADD` weitere 15 Befehle gibt.

Aufgabe 4

Ein *Stack-Register* unterstützt folgende zwei Operationen: `PUSH Reg` (nimmt einen Wert aus einem „normalen“ Register `Reg` und legt ihn auf den Stack) und `POP Reg` (entfernt den obersten Wert vom Stack und speichert ihn in dem Register `Reg`).

a) Kann man ein solches Schema mit „gewöhnlichen“ Registern implementieren oder muß man Einschränkungen in Kauf nehmen?

b) Schreiben sie Mikroprogramme (s. erste Aufgabe) für `PUSH` und `POP`. Diese können auf beliebige Register und Hauptspeicher zugreifen, soweit notwendig.

c) Was muß man beachten, wenn man neben dem Stack auch direkten Zugriff auf den Hauptspeicher haben will?

d) Was muß man beachten, wenn man mehrere Stack-Register implementieren will?

Besprechung am Donnerstag, den 20.6.2002.