Übungen zu
Grundlagen der Informatik für Studierende des Maschinenbaus
im Sommersemester 2001
(Dipl.-Inf. Jörg Barner)

Grundlage für die Aufgaben dieses Übungsblattes ist die im Lehrbuch von Rembold angegebene hypothetische Assemblersprache.

Aufgabe 4: Maschinensprache
Interpretieren Sie die folgenden Maschinenbefehle:

a) 0001000100001010

b) 0101010001100100

Lösung:

	Opcode	Doppelw.	MOD	Adresse/Wert	Mnemot. Bez.
a)	0001	0	001	00001010	`LAD, @266`
b)	0101	0	100	01100100	`MUL, #100`

Aufgabe 5: Adressierung
Erläutern Sie kurz die wesentlichen Eigenschaften der in der Vorlesung vorgestellten Adressierungsarten.

Lösung:

Sofort-Operanden (immediate): Der Operand wird als Konstante direkt im Befehl angegeben. (z.B. `ADD, #2`)

Direkte Adressierung (direct): Im Befehl wird eine Speicheradresse angegeben, an der sich der Operand befindet.

Relative Adressierung (relative): Der Operand befindet sich im Speicher an der Adresse

$$\text{Adresse des nächsten Befehls} + \textit{offset}$$

wobei *offset* die im Befehl enthaltene Adressangabe ist.

Indirekte Adressierung (indirect): Die Adresse des Operanden befindet sich wiederum im Speicher, und zwar an der im Befehl (direkt) angegebenen Adresse.

Indexregister (index with displacement): Die Adresse des Operanden ergibt sich aus

$$\text{Inhalt Indexregister} + \textit{offset}$$

wobei *offset* die im Befehl enthaltene Adressangabe ist.

Ist ähnlich zur indirekten Adressierung, mit dem Unterschied daß sich die indirekte Adreßangabe in einem Register und nicht im Speicher befindet, und daß zusätzlich ein Offset addiert wird.

Aufgabe 6: Assemblersprache
Betrachten Sie einen 10-dimensionalen Vektor. Die Elemente des Vektors sind in den aufeinanderfolgenden Adressen 2000 bis 2009 gespeichert.

a) Überlegen Sie sich, wie ein Programm in der hypothetischen Assemblersprache aussieht, welches das Skalarprodukt des Vektors mit sich selbst berechnet und *keine Sprungbefehle* enthält. Als Adressierungsart ist nur direkte Adressierung erlaubt. Das Ergebnis der Berechnung soll an der Adresse 2010 abgelegt werden.

b) Schreiben Sie ein Programm zur Berechnung des Skalarproduktes des Vektors mit sich selbst, indem Sie Sprungbefehle und indirekte Adressierung verwenden. Welche Vorteile ergeben sich?

c) Formulieren Sie die Berechnung des Skalarproduktes des Vektors mit sich selbst unter Verwendung eines Indexregisters.

Lösung:

a)
```
VEKTOR EQUIVALENT 2000
SPROD  EQUIVALENT 2010

    LAD, @VEKTOR
    MUL, @VEKTOR        ; Akku := erstes Element quadriert
    SPI, @SPROD         ; speichern

    LAD, @VEKTOR+1
    MUL, @VEKTOR+1      ; Akku := zweites Element quadriert
    ADD, @SPROD         ; plus bisheriges Ergebnis
    SPI, @SPROD         ; speichern
...

    LAD, @VEKTOR+9
    MUL, @VEKTOR+9      ; Akku := letztes Element quadriert
    ADD, @SPROD         ; plus bisheriges Ergebnis
    SPi, @SPROD         ; speichern
```

Die Befehle zur Quadrierung und zum Addieren müssen für jedes Element des Vektors wiederholt hingeschrieben werden!

b)
```
VEKTOR EQUIVALENT 2000
SPROD  EQUIVALENT 2010
ZEIGER EQUIVALENT 2011  ; Zeiger auf zu bearbeitendes Element

    LAD, #0
    SPI, @SPROD         ; SPROD auf 0 initialisieren
    LAD, #VEKTOR
    SPI, @ZEIGER        ; ZEIGER auf erstes Element initialisieren

SCHLEIFE:
    LAD, (ZEIGER)       ; Laden des aktuellen Elements
    MUL, (ZEIGER)       ; mit sich selbst multiplizieren
    ADD, @SPROD         ; bisherige Summe addieren
    SPI, @SPROD         ; speichern

    LAD, @ZEIGER
    ADD, #1
    SPI, @ZEIGER        ; ZEIGER um 1 erhoehen
    SUB, #VEKTOR+10     ; Akku := ZEIGER - VEKTOR - 10
    JLZ, @SCHLEIFE      ; wiederhole Schleife, wenn Akku < 0
                        ; <=> ZEIGER - VEKTOR < 10
```

Die Länge des Programms ist nun unabhängig von der Länge des Vektors! Es können also nun beliebig große Vektoren bearbeitet werden, ohne das Programm immer weiter verlängern zu müssen.

Da die zu bearbeitende Adresse für jeden Schleifendurchlauf verschieden ist, kann sie nicht direkt angegeben werden, sondern es muß die indirekte Adressierung verwendet werden.

c)
```
VEKTOR EQUIVALENT 2000
SPROD  EQUIVALENT 2010

    LAD, #0
    SPI, @SPROD         ; SPROD auf 0 initialisieren
    CLI                 ; Indexregister auf 0 setzen

SCHLEIFE:
    LAD, %VEKTOR        ; Laden des aktuellen Elements
                        ; (Adr. VEKTOR + Indexregister)
    MUL, %VEKTOR        ; mit sich selbst multiplizieren
    ADD, @SPROD         ; bisherige Summe addieren
    SPI, @SPROD         ; speichern

    INK                 ; Index um 1 erhoehen
    LIA                 ; Index in den Akku holen
    SUB, #10            ; Akku := Index - 10
```

```
    JLZ, @SCHLEIFE      ; wiederhole Schleife, wenn Akku < 0
                        ; <=> Index < 10
```

Unterschied zu vorher ist nur, daß sich der Zeiger auf das aktuelle Element nun im Indexregister und nicht mehr im Speicher befindet. Dies ermöglicht die Ausnutzung der Index-Adressierung und spezieller Index-Befehle (z.B. `CLI, INK`), was die Ausführung des Programms beschleunigt.

Äquivalent wäre es, direkt die Adresse des Elements im Indexregister zu halten, d.h. es läuft von `VEKTOR` bis `VEKTOR * 9`.
