PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge
22. Januar 2003

Übungen zur „Technischen Informatik I“, WS 2002/03

Nr. 12 (letztes Blatt)
Besprechung bzw. Abgabe: 29.1. bis 31.1. in den Übungsgruppen

---

Die **Klausur** findet am Dienstag, dem 4. Februar 2003, von 14:30 Uhr bis 16:30 Uhr im Audimax des Hörsaalgebäudes, Biegenstraße 14, statt.
Hilfsmittel sind nicht erlaubt. Mitzubringen ist lediglich Schreibzeug.

A. Mündliche Aufgaben

56. Betrachten Sie eine Kellermaschine, in der folgende Befehle zur Verfügung stehen:

`ADD` `AND`
`SUB` `OR`
`MULT` `NOT`
`DIV`
	`PUSH`$<\mathit{adresse}>$
	`POP`$<\mathit{adresse}>$
		`JT`$<\mathit{label}>$
		`JF`$<\mathit{label}>$
		`COMP`

Die beiden oberen (bzw. bei `NOT` nur das oberste) Kellerelement(e) werden verknüpft und durch das Ergebnis ersetzt.	`PUSH` schreibt ein Speicherelement auf den Keller. `POP` nimmt das oberste Element vom Keller und speichert es.	`JT`/`JF` führt zu einem Sprung, falls das oberste Kellerelement `true`/`false` ist. Dieses wird dabei entfernt. `COMP` vergleicht die beiden oberen Kellerelemente und ersetzt diese durch `true`/`false` je nach Vergleichsergebnis.

Entwickeln Sie für die folgenden Anweisungen möglichst kurzen Stackcode:

(a) `Z:= (A - B/C) + D`

(b) `if A XOR B` $\neq$ `C then C := D`
`else goto E`

57. In imperativen Programmiersprachen existieren Verzweigungskommandos und Schleifenkonstrukte, die in Assembler nicht verfügbar sind. Im folgenden sollen Sie solche Konstrukte in Assemblercodestücke umsetzen, wobei eine Boolesche Variable im Akkumulatorregister `AX` simuliert wird. Es stehen die Sprungbefehle `jmp MARKE`, `jz MARKE` und `jnz MARKE` zur Verfügung. `LOOP`-Befehle sollen nicht verwendet werden. Beispielsweise wird die Alternative

`if <Bedingung> then <Anweisung>`

dargestellt durch

```
if     : <Code fuer Bedingungsauswertung mit Ergebnis in AX>
          OR AX, AX    ; Flags gemaess Inhalt von AX setzen
                       ; 0 entspricht False, Wert <> 0 bedeutet True
          jz weiter
then   : <Code fuer Anweisung>
weiter :
```

Setzen Sie die folgenden Pascal-Konstrukte entsprechend um.

(a) `while` <Bedingung> `do` <Anweisung>

(b) `for index :=` <Startwert> `to` <Endwert> `do` <Anweisung>

B. Hausaufgaben

Die Abgabe der Hausaufgaben ist in Gruppen bis zu 3 Personen erlaubt.

58. Ein-Befehlsrechner **4 Punkte**

Gegeben sei ein Rechner mit dem Namen SIC (Single Instruction Computer), der mit einem einzigen Befehl auskommt. Dieser Befehl heißt `sbn` (subtract and branch if negative). Er erhält drei Operanden, in denen jeweils Speicheradressen stehen:

```
sbn a, b, c         # a := a-b; if (a < 0) goto c
```

Das folgende Beispiel zeigt, wie eine Zahl von a nach b kopiert werden kann. Dabei steht `.+1` für die jeweils nachfolgende Adresse, so dass die Abarbeitung hier immer mit der nächsten Instruktion weitergeht, unabhängig davon, ob das Ergebnis negativ ist oder nicht. Die Adresse temp wird für Zwischenergebnisse verwendet.

```
start : sbn temp, temp, .+1        # temp := 0
        sbn temp, a, .+1           # temp := -a
        sbn b, b, .+1              # b := 0
        sbn b, temp, .+1           # b := -(-a)
```

(a) Schreiben Sie ein SIC-Programm zur Addition der Inhalte von `a` und `b`, das die Summe in `a` speichert und den Inhalt von `b` unverändert lässt.

(b) Schreiben Sie ein SIC-Programm zur Multiplikation zweier positiver Zahlen, die in `a` und `b` gespeichert sind. Das Produkt soll in `c` geschrieben werden und die Inhalte von `a` und `b` dürfen modifiziert werden. Unter der Adresse `one` steht der Wert 1 zur Verfügung.

59. Assemblerprogrammierung **4 Punkte**

Schreiben Sie ein `tasm`-Programm, das eine durch `<CR>` (carriage return) abgeschlossene Zeichenfolge von der Tastatur einliest und testet, ob sie die Form $0^n 1^n$ für eine beliebige natürliche Zahl $n \geq 1$ hat. Als Antwort soll auf dem Bildschirm der Buchstabe "j" oder "n" erscheinen.

Beispielsweise soll auf die Eingabe "000111" die Antwort "j" und auf die Eingaben "00111" oder "ab01" die Antwort "n" gegeben werden.

60. Assembler-Unterprogramm **4 Punkte**

Schreiben Sie ein Assembler-Unterprogramm, welches die maximale Komponente eines Vektors bestimmt.

Der Vektor soll aus 100 vorzeichenlosen Zahlen der Länge 1 Byte bestehen. Seine Anfangsadresse soll durch den symbolischen Namen VEKTOR bezeichnet sein. Das Ergebnis soll im Speicher unter der symbolischen Adresse MAXELEM abgelegt werden. Geben Sie die Deklaration der Speicherplätze an.

Nach dem Aufruf des Unterprogramms sollen alle Register dieselben Werte wie vor dem Aufruf enthalten. Versehen Sie Ihr Programm mit Kommentaren. Beachten Sie, dass zwei Speicheroperanden in Assemblerbefehlen nicht erlaubt sind.