Grundlagen der Programmkonstruktion
Übungsblatt 1

Organisatorisches

Ablauf

Lösen Sie die Aufgaben auf diesem Übungsblatt. Kommen Sie mit diesen Lösungen in die Übungseinheit, für die Sie sich am 5.3. oder 8.3. angemeldet haben. Sollten Sie nach dem 8.3. noch nicht für eine Übungsgruppe angemeldet sein, informieren Sie uns bitte davon an

`programmkonstruktion@complang.tuwien.ac.at`

In der Übungseinheit sollen Sie in der Lage sein, die Lösung der gestellten Aufgaben zu präsentieren. Dabei steht im Vordergrund, wie Sie zu ihrer Lösung gelangen und weniger das korrekte Ergebnis.

Gruppenarbeiten

Die Aufgaben von Übungsblatt 2-6 sollen Sie in Gruppen erarbeiten und auch in Gruppen präsentieren. Diese Gruppen werden in der ersten Übungseinheit gebildet. Lösen Sie daher die Aufgaben auf diesem Übungsblatt selbstständig.

Fragen

Bei Fragen wenden Sie sich bitte zunächst an Ihre Kollegen oder an die Sprechstunden der Tutoren (genaueres dazu wird noch bekannt gegeben). Sollte in der Angabe etwas unklar sein, sehen Sie bitte zunächst auf der Homepage der Lehrveranstaltung nach, ob dort bereits ergänzende Anmerkungen zum Übungsteil stehen. Wichtige Ankündigungen zum Übungsteil werden auch in der Vorlesung und online bekannt gegeben.

1 Aufgabe 1: Binärzahlen

- Wandeln Sie ihre Matrikelnummer in eine Binärzahl, eine Octalzahl und eine Hexadezimalzahl um.
- Nehmen Sie danach die Bits 15..8 und 7..0 (d.h., die ersten 8 Stellen von rechts gelesen) der Binärzahl und verknüpfen Sie diese mit einem bitweisen XOR, AND und OR. (sollte ihre Matrikelnummer in Binärdarstellung 1001.1010.0001.1110.1101.1110 lauten, so sollen Sie 0001.1110 xor/and/or 1101.1110 berechnen)
- Wandeln Sie die Ergebnisse der vorigen Rechnung in eine Dezimalzahl um.

Beispiel Sie können die Binärdarstellung einer beliebigen Zahl (hier beispielsweise 295) folgendermaßen ermitteln:

Zahl	Division durch 2	Zahl modulo 2 (Rest bei Division durch 2)
295	295 / 2 = 147, 1 Rest	1 (Bit 0)
147	147 / 2 = 73, 1 Rest	1 (Bit 1)
73	73 / 2 = 36, 1 Rest	1 (Bit 2)
36	36 / 2 = 18, 0 Rest	0 (Bit 3)
18	18 / 2 = 9, 0 Rest	0 (Bit 4)
9	9 / 2 = 4, 1 Rest	1 (Bit 5)
4	4 / 2 = 2, 0 Rest	0 (Bit 6)
2	2 / 2 = 1, 0 Rest	0 (Bit 7)
1	1 / 2 = 0, 1 Rest	1 (Bit 8)
0	fertig.	

Die Binärdarstellung von 295 beträgt somit 1.0010.0111.

Diese Zahl können Sie mit folgender Rechnung wieder in eine Dezimalzahl umwandeln: $1*2^8+0*2^7+0*2^6+1*2^5+0*2^4+0*2^3+1*2^2+1*2^1+1*2^0 = 295$.

Für Zahlen zur Basis 8 (Octalzahlen) und zur Basis 16 ersetzen Sie 2 durch die entsprechende Basis.

2 Aufgabe 2: Assembler

Assemblersprache ist für den Computer lesbarer Maschinencode, der in eine für den Menschen besser verständliche Form gebracht wurde. Das folgende Beispiel ist solcher (vereinfachter) Assembler-Code (für x86-Prozessoren; die Zahl davor repräsentiert die Speicheradresse, an der der Befehl liegt):

```
0: mov ax 4
1: mov dx 1
2: cmp ax 0
3: je 7
4: mul dx ax
5: dec ax
6: jmp 2
7: ret
```

Der Code benutzt den Befehlszeiger (Program Counter, im Folgenden PC), zwei Register `ax` und `dx`, eine boolsche Variable (ein *Flag*) `zero` und die folgenden Befehle:

`mul` ***Register1 Register2*** Berechnet *Register1* * *Register2* (Multiplikation) und speichert das Ergebnis in *Register1* (äquivalent zu "*Register1* `*=` *Register2*").

`dec` ***Register*** Berechnet *Register* - 1 und speichert das Ergebnis in *Register* (äquivalent zu "*Register*`--`")

`cmp` ***Register Zahl*** Setzt `zero` auf `true`, wenn *Register* gleich *Zahl* ist, ansonsten auf `false` (äquivalent zu "`zero = (` *Register* `==` *Zahl*`)`")

`mov` ***Register Zahl*** Setzt den Wert von *Register* auf *Zahl* (äquivalent zu "*Register* `=` *Zahl*")

`je` ***Adresse*** Bedingter Sprung: Wenn `zero` auf `true` gesetzt ist, springe zu (= setze PC auf) *Adresse*, ansonsten gehe zum nächsten Befehl.

`jmp` ***Adresse*** Unbedingter Sprung: Springe zu (= setze PC auf) *Adresse*.

`ret` Beendet die ausgeührte Prozedur.

Nachdem ein Befehl ausgeführt wurde, wird PC um 1 erhöht und somit der nächste Befehl aufgeführt. Eine Ausnahme sind hier die Sprünge `je` (falls `zero` auf `true` gesetzt ist) und `jmp`, die PC auf eine neue Adresse setzen. D.h., wenn PC den Wert 3 hat, so wird der Befehl `je 7` ausgeführt. Sollte das `zero`-Flag gesetzt sein, so enthält danach der PC den Wert 7, ansonsten 4.

- Lassen Sie das Programm-Fragment mental laufen und erstellen Sie dabei eine Tabelle, in der Sie den Zustand des PC, der Register, des `zero`-Flags (sofern es von Bedeutung ist) und des nächsten Befehls angeben, bis Sie den Befehl `ret` ausführen. Sollte ein Wert irrelevant oder unbekannt sein, verwenden Sie "?" als Platzhalter.

Beispiel Die ersten zwei Zeilen ihrer Tabelle lauten folgendermaßen:

PC	`ax`	`dx`	`zero`	nächster Befehl
0	?	?	?	`mov ax 4`
1	4	?	?	`mov dx 1`
2	4	1	?	`cmp ax 0`
⋮	⋮	⋮	⋮	⋮

3 Aufgabe 3: Euklidischer Algorithmus

Für folgende Aufgabe teilen Sie ihre Matrikelnummer nun in zwei Teile: Die ersten vier Stellen werden im Folgenden als m, die letzten drei Stellen als n bezeichnet (d.h., wenn ihre Matrikelnummer 1234567 ist, so ist $m = 1234$ und $n = 567$).

Gegeben ist folgender in Java geschriebener Programm-Code:

```
int a = ...; // siehe Text
int b = ...; // siehe Text

while ( a != b ) {  // solange a ungleich b
    if ( a > b ) {  //   wenn a groesser als b
        a = a - b;  //     a wird ersetzt durch den Wert von a minus b
    } else {        //   ansonsten ( d.h. wenn a kleiner als b ist )
        b = b - a;  //     b wird ersetzt durch den Wert von b minus a
    }
}
```

Lassen Sie dieses Programm für folgende Werte ablaufen und erstellen Sie eine Tabelle mit allen lokalen Variablen. Sie sollen zumindest 10 Zeilen durchführen. Beurteilen Sie, wie das Programm weiter ablaufen wird.

1. $a = m$, $b = n$ (also z.B., $a = 1234, b = 567$).
2. $a = -m$, $b = n$ (also z.B., $a = -1234, b = 567$).
3. $a = -m$, $b = -n$ (also z.B., $a = -1234, b = -567$).
4. $a = m$, $b = -n$ (also z.B., $a = 1234, b = -567$).

Beispiel Sollte Ihre Matrikelnummer 1234567 sein, lauten die ersten vier Zeilen ihrer Tabelle für die erste Teilaufgabe folgendermaßen:

Schleifendurchlauf	a	b
0	1234	567
1	1234 - 567 = 667	567
2	667 - 567 = 100	567
3	100	567 - 100 = 467
⋮	⋮	⋮

Hinweis Sollten Sie den Programmcode in Java laufen lassen (empfohlen), beachten Sie, dass Dezimalzahlen keine führenden Nullen haben dürfen.