Grundbegriffe der Informatik — Aufgabenblatt 8
Lösungsvorschläge

Matr.nr.:	
Nachname:	
Vorname:	
Tutorium Nr.:	Tutor*in:

Ausgabe: 10. Dezember 2021, 12:00 Uhr

Abgabe: 17. Dezember 2021, 12:30 Uhr
in dem Holzkasten neben Raum -119
im UG des Info-Gebäudes (50.34)

Lösungen werden nur korrigiert, wenn sie
- handschriftlich erstellt sind (Tablet-Ausdruck erlaubt) und
- mit dieser Seite als Deckblatt
- in der oberen **linken** Ecke zusammengeheftet

rechtzeitig abgegeben werden.

Abgaberegeln für Teilnehmer der Online-Tutorien:
- handschriftlich erstellt (lesbare Fotos akzeptiert)
- **rechtzeitig**, mit diesem Deckblatt in **genau einer** PDF-Datei
- direkt an den entsprechenden Tutor abgeben.

---

*Von Tutor*in auszufüllen:*

erreichte Punkte

Blatt 8:	/ 20
Blätter 7 – 8:	/ 40 [+3]

Gehen Sie in allen Aufgaben vom MIMA-Befehlssatz aus dem Skript (S. 91 - 94) aus. Beim Erstellen von eigenen Programmen dürfen Sie zusätzliche Adressen verwenden. Geben Sie diesen **aussagekräftige Namen**!

Aufgabe 8.1 (1,5 + 1 + 1,5 + 1 = 5 Punkte)

Rechts sehen Sie ein MIMA-Programm.

buffer und *k-buffer* sind zwei beliebige Adressen.

a) Berechnen Sie jeweils den Wert des *Akkumulators*, wenn das Programm, mit den jeweiligen Werten im *Akkumulator*, bzw. an Adresse *k* aufgerufen wird:

	Akku	*k*
i)	0111	0100
ii)	0110	0010
iii)	0000	0011

Anmerkung: Um Ihnen Arbeit zu sparen tun wir hier so, als ob die MIMA im 4-Bit-Zweierkomplement rechne.

b) Was berechnet das Programm allgemein für Start-*Akku*-werte ≥ 0 sowie *k*-Werte > 1?

```
Anfang:  STV   buffer
         LDV   k
         NOT
         STV   k-buffer
         LDC   1
         ADD   k-buffer
         ADD   buffer
         JMN   Ende
         JMP   Anfang
Ende:    ADD   k
         HALT
```

c) Bei den Berechnungen für a) ist Ihnen bestimmt aufgefallen, dass das Programm „ineffizient“ ist, weil bestimmte Befehle unnötig häufig wiederholt werden. Modifizieren Sie das Programm, sodass dies nicht mehr der Fall ist. Die Ergebnisse des Programms sollen dabei unverändert bleiben.

d) Das Programm „funktioniert nicht“ falls im *Akkumulator* zu Beginn Werte < 0 stehen. Modifizieren Sie das (Ursprungs-)Programm, sodass es für *Akku*-Werte < 0 analog wie für *Akku*-Werte ≥ 0 funktioniert.

Heben Sie bei c) und d) Ihre Änderungen zum Ursprungsprogramm (z.b.) farblich hervor. **Achten Sie auf lesbare Programme.**

Lösung 8.1

a) In der Folgenden Tabelle sind die Werte je einmal im Zweierkomplement sowie in Klammern in Dezimaldarstellung angegeben:

Start-*Akku*	Wert in k	End-*Akku*
0111 (7)	0100 (4)	0011 (3)
0110 (6)	0010 (2)	0000 (0)
0000 (0)	0011 (3)	0000 (0)

b) Das Programm berechnet x **mod** k^*, wobei $x \geq 0$ der Anfangswert im *Akku* und $k^* > 1$ der Wert an Adresse *k* ist.

```
            STV   buffer
            LDV   k
            NOT
            STV   k-buffer
            LDV   1
            ADD   k-buffer
c)          STV   k-buffer
            LDV   buffer
Anfang:     ADD   k-buffer
            JMN   Ende
            JMP   Anfang
Ende:       ADD   k
            HALT
```

```
            JMN   Ende
Anfang:     STV   buffer
            LDV   k
            NOT
            STV   k-buffer
            LDC   1
c)          ADD   k-buffer
            ADD   buffer
            JMN   Ende
            JMP   Anfang
Ende:       ADD   k
            JMN   Ende
            HALT
```

Aufgabe 8.2 (0,5 + 1,5 = 2 Punkte)

Irgendwie hat sich die Lieferung von Wunsch.de als Reinfall entpuppt. Statt „mega intelligenten maschinellen Assistenten“ (mimAs) wurden Ihnen „minimal machines“ (MIMAs) geliefert. Diese Geräte sind in der Praxis deutlich weniger hilfreich als es hyper-intelligente-Quanten-KIs gewesen wären. Insbesondere der sehr limitierte Befehlssatz macht Programmierung dafür etwas mühselig*. Glücklicherweise können Sie mit Ihrem jüngst erworbenen Verständnis von Prozessoren den Befehlssatz durch eigene kleine MIMA-Programme geschickt erweitern. Beenden Sie ihre Programme **nicht** mit HALT.
In nachfolgenden (Teil-)Aufgaben dieses Übungsblattes dürfen Sie die hier eingeführten Programme als zusätzliche MIMA-Befehle verwenden.

a) INC *addr*
   soll den Wert an Adresse *addr* inkrementieren, also um eins erhöhen. Das Ergebnis soll wieder in *addr* gespeichert werden.

b) SUBI *addr*
   soll den Wert, der sich an genau der Adresse befindet, die selbst in *addr* gespeichert ist, vom aktuellen *Akkumulator*-Inhalt subtrahieren. Das Ergebnis soll im *Akkumulator* stehen.
   Hinweis: $a - b$ ist immer noch $a + (-b)$ *// und LA immer noch wichtig ;-)*

*Dr. Meta hat entscheidend Wichtigeres zu tun, weshalb er diese Aufgabe natürlich Ihnen anvertraut.

Lösung 8.2

a) INC *addr*:

```
LDC   1
ADD   addr
STV   addr
```

b) SUBI *addr*:

```
STV   minuend
LDIV  addr
NOT
STV   inverser-subtrahend
INC   inverser-subtrahend
LDV   minuend
ADD   inverser-subtrahend
```

Aufgabe 8.3 (5 Punkte)

Während Sie **INC *addr*** und **SUBI *addr*** programmiert haben, hat jemand schnell noch einen Befehl **MOD *addr*** programmiert, mit dem der aktuell im *Akkumulator* befindliche Wert **mod** dem Wert an Adresse *addr* gerechnet werden kann. Das Ergebnis dieser Rechnung steht wieder im *Akkumulator*.

Mit Hilfe dieser zusätzlichen Befehle können Sie nun ein Programm schreiben, dass die B.I.R.D.-Daten automatisiert entschlüsselt. Die verschlüsselte Nachricht sowie das geeignete Passwort liegen als *Liste* von Zeichen jeweils in zusammenhängenden Adressbereichen. Die Zeichen sind dabei praktischerweise als Zahlenwert im Zweierkomplement gespeichert, sodass sie diese Werte einfach verrechnen können. Für ihr Programm kennen Sie 4 Adressen des Datenspeichers: *m-start*, *p-start*, bzw. *m-end*, *p-end*, an denen jeweils das erste, bzw. letzte Zeichen von Nachricht und Passwort gespeichert ist. Zum Entschlüsseln, sollen Sie die Zeichen jeweils einzeln verrechnen, sodass gilt: $e_i = (m_i - p_{i^*})$ **mod** k, wobei e_i, m_i, p_{i^*} je das i-te Zeichen der entschlüsselten Nachricht, der verschlüsselten Nachricht, bzw. des Passwortes* ist. Beachten Sie, dass das Passwort unter Umständen kürzer als die verschlüsselte Nachricht ist. In diesem Fall müssen Sie wiederholt über das Passwort iterieren (deshalb der *).

Schreiben Sie die entschlüsselte Nachricht wieder an die selben Adressen der ursprünglichen Nachricht, also *m-start* bis inkl. *m-end*. Der Wert k für die Modulorechnung liegt dem Progamm ebenfalls an Adresse *$k* vor. **Achten Sie auf ein lesbares Programm!** Nummerieren Sie die einzelnen Zeilen und beschreiben Sie außerdem kurz die Funktion einzelner Programmblöcke, sodass insgesamt Ihr gesamtes Programm beschrieben ist.

Hinweis: Orientieren Sie sich zur Verarbeitung einer Liste mittels indirekter Adressierung an den VL-Folien zum Aufsummieren von Elementen (Folie 61-63)

Lösung 8.3

(1)	LDC	*m-end*
(2)	STV	*m-end-addr*
(3)	LDC	*p-end*
(4)	STV	*p-end-addr*
(5)	LDC	*m-start*
(6)	STV	*m-ptr*
(7) *P-Reset:*	LDC	*p-start*
(8)	STV	*p-ptr*
(9) *Anfang:*	LDIV	*m-ptr*
(10)	SUBI	*p-ptr*
(11)	MOD	*$k*
(12)	STIV	*m-ptr*
(13)	LDV	*m-ptr*
(14)	EQL	*m-end-addr*
(15)	JMN	*Ende*
(16)	INC	*m-ptr*
(17)	LDV	*p-ptr*
(18)	EQL	*p-end-addr*
(19)	JMN	*P-Reset*
(20)	INC	*p-ptr*
(21)	JMP	*Anfang*
(22) *Ende:*	HALT	

(1-4) Je eine Variable (*m-ptr*, *p-ptr*) zur indirekten Adressierung der einzelnen Zeichen der Nachricht , bzw. des Passwortes angelegen. Beide verweisen initial auf das jeweils erste Zeichen.

(5-8) Einzelne Zeichen von Nachricht und Passwort verrechnen und Ergebnis speichern.

(9-12) Falls *m-ptr* bereits auf das letzte Zeichen der Nachricht verwiesen hat, wird zum *Ende* des Programms gesprungen, andernfalls wird *m-ptr* inkrementiert.

(13-16) Falls *p-ptr* auf das letzte Zeichen des Passwortes verwiesen hat, wird *p-ptr* wieder auf die Adresse des ersten Zeichens zurück gesetzt (recycling von Zeile 3-4), andernfalls inkrementiert.

(17) Sprung zu Zeile 5) um das nächste Zeichen zu entschlüsseln.

Aufgabe 8.4 (1 + 1 + 3 + 1 + 2 = 8 Punkte)

Gegeben eine kontextfreie Grammatik $G = (N, T, \mathtt{S}, P)$, mit:

- $N = \{\mathtt{S}, \mathtt{A}, \mathtt{B}, \mathtt{C}\}$
- $T = \{\mathtt{a}, \mathtt{b}\}$
- $P = \{\mathtt{S} \to \mathtt{aS}|\mathtt{bS}|\mathtt{aA}, \quad \mathtt{A} \to \mathtt{aA}|\mathtt{bA}|\mathtt{aB}, \quad \mathtt{B} \to \mathtt{aB}|\mathtt{bB}|\mathtt{aC}, \quad \mathtt{C} \to \mathtt{aC}|\mathtt{bC}|\varepsilon\}$

a) Geben Sie zwei verschiedene Ableitungsbäume für das Wort $w = abaaba$ an.

b) Geben Sie eine formale Definition der von G erzeugten Sprache $L(G)$ ohne Verwendung natürlicher Sprache an. (Teilpunkte bei Verwendung natürlicher Sprache).

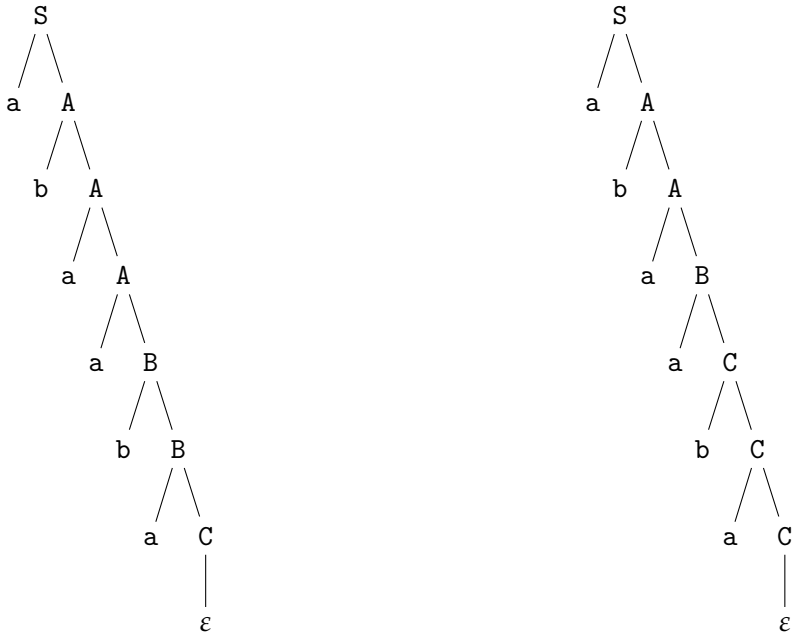
c) Sei $w = \mathtt{aabaa}$. Wie viele verschiedene Ableitungsbäume für die Erzeugung von w durch G gibt es? Beweisen Sie.
Hinweis: Argumentieren Sie über die Struktur der Produktionsmenge
Hinweis: Welche „vier Zwischenphasen" durchläuft ein Wort bei der Erzeugung durch G?

d) Welches $w \in (\{\mathtt{a}, \mathtt{b}\}^5 \cap L(G))$ hat die meisten verschiedenen Ableitungsbäume? (Angabe genügt)

e) Definieren Sie eine Grammatik $G' = (N', T', \mathtt{S}', P')$, mit $L(G') = L(G)$ und minimalem $|N'|$.

Lösung 8.4

a)



b) $L(G) = \{w \in \{\mathtt{a},\mathtt{b}\}^* \mid N_{\mathtt{a}}(w) \geq 3\}$

c) Es gibt vier verschiedene Ableitungsbäume für $w = \mathtt{aabaa}$.
Beweis:
Betrachtet man die Erzeugung von w durch G, dann sieht man, dass
$\mathtt{S} \Rightarrow^* w_{\mathtt{A}} \Rightarrow^* w_{\mathtt{B}} \Rightarrow^* w_{\mathtt{C}} \Rightarrow^* w$ gelten muss, wobei $w_{\mathtt{A}}, w_{\mathtt{B}}, w_{\mathtt{C}}$ jeweils „Zwischenworte“ sind, die als Nichtterminale ausschließlich ein `A`, `B`, bzw. `C` enthalten. Man kann die Erzeugung von w also nach dem jeweils „aktiven“ Nichtterminalsymbol in vier Zwischenphasen unterteilen, die der Reihe nach durchlaufen werden müssen. Um von einer Phase in die Nächste zu kommen, wird dabei stets ein „`a`“ erzeugt. Unterteilt man w also in die Teilworte, welche in der jeweiligen Phase erzeugt werden, erhält man folgende vier Möglichkeiten (eine je Zeile):

S	A	B	C
aa	ba	a	ε
a	aba	a	ε
a	a	baa	ε
a	a	ba	a

Hierbei steht in jeder Spalte das Teilwort, welches jeweils in der entsprechenden Phase (also aus dem entsprechenden Nichtterminal) erzeugt wurde. Dies sind die einzigen vier Möglichkeiten w zu erzeugen, da kein Phasenteilwort mit einem `b` enden kann (beim Übergang wird immer ein `a` erzeugt) und ausschließlich die letzte Phase ε erzeugen darf. Jede dieser Möglichkeiten erzeugt (trivialerweise) einen anderen Ableitungsbaum.

d) $w = \mathtt{aaaaa}$
Begründung (nicht gefordert):
aus c) folgt, dass `b`'s im Wort die Möglichkeiten für verschiedene Ableitungsbäume nur einschränken.

e) $G' = (N', \{\mathtt{a},\mathtt{b}\}, \mathtt{S'}, P')$, wobei:
- $N = \{\mathtt{S'}, \mathtt{X}\}$
- $P = \{\mathtt{S'} \to \mathtt{XaXaXaX}, \quad X \to \varepsilon \mid \mathtt{a} \mid \mathtt{b} \mid \mathtt{XX}\}$