Grundbegriffe der Informatik — Aufgabenblatt 7
Lösungsvorschläge

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

Ausgabe: Freitag, 9.12.2022, 14:30 Uhr

Abgabe: Freitag, 16.12.2022, 12:30 Uhr
Online, oder 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 Tutorien mit Online-Abgabe:
- handschriftlich erstellt (Scans und lesbare Fotos akzeptiert)
- **rechtzeitig**, mit diesem Deckblatt in **genau einer** PDF-Datei
- in ILIAS unter "Tutorien" im Ordner des richtigen Tutoriums abgeben.

---

*Von Tutor*in auszufüllen:*

erreichte Punkte

Blatt 7:	/ 20 (+4)
Blätter 7 – 7:	/ 20 (+4)
Blätter 1 – 7:	/ 144 (+4)

Aufgabe 7.1 (3 + 3 = 6 Punkte)
In dieser Aufgabe sollen Sie mit kleinen Programmen mit der Programmierung der MIMA etwas warm werden. Es bezeichnen in dieser Aufgabe $x, y, z \in Z_2^{20}$ drei verschiedene Speicherstellen, die außerhalb des Speicherbereichts, in dem das Programm steht, liegen. Sie können die drei Speicherstellen zum Zwischenspeichern von Ergebnissen in Ihren Programmen verwenden.

a) Sie haben die MIMA-Befehl **RAR** kennengelernt, die die Bits im Akkumulator um eine Stelle zyklisch nach rechts rotiert. Formal beschrieben setzt sie damit folgende Funktionalität

$$Akku \leftarrow \text{rar}(Akku) \text{ mit } \text{rar}(wx) = xw \quad (\text{mit } x \in Z_2,\ w \in Z_2^{23})$$

um. Schreiben Sie ein MIMA-Programm, das die Umkehrfunktion ral : $Z_2^{24} \to Z_2^{24}$ von rar realisiert, also den Wert des Akkumulator um eine Stelle zyklisch nach links rotiert und ihn zurück in den Akkumulator schreibt.

b) Der Programmablauf in der MIMA kann durch bedingte Sprünge gesteuert werden. In der MIMA funktionieren bedingte Sprünge, indem überprüft wird, ob der Akkumulator *Akku* eine negative Zahl enthält (also das höchst-wertige Bit gesetzt ist).
Es gibt einen MIMA-Befehl **EQL** zur Prüfung der Gleichheit zweier Werte. Ziel dieser Teilaufgabe ist es, ein Programm zu schreiben, dass einen bedingten Sprung ausführt, wenn der Wert an der Speicherstelle *a* (echt) kleiner ist als der Wert an der Speicherstelle *b* (jeweils im Zweierkomplement interpretiert).

   i) Schreiben Sie ein Programm, das zwei positive Zahlen, die an den Adressen *a* und *b* hinterlegt sind, wie folgt vergleicht: Das Programm fährt genau dann mit der Ausführung an der Speicherstelle *a_less_b* fort, wenn die an *a* hinterlegte Zahl echt kleiner ist als die an *b* hinterlegte Zahl. Andernfalls fährt das Programm an der Speicherstelle *a_geq_b* fort.

   ii) Funktioniert ihr Programm auch, wenn beliebige Zahlen aus $\mathbb{K}_{24} = \{x \in \mathbb{Z} \mid -2^{23} \leq x \leq 2^{23} - 1\}$in Zweierkomplementdarstellung an den Adressen *a* und *b* hinterlegt sind? Wenn ja, begründen Sie. Wenn nein, geben Sie ein Gegenbeispiel an.

Lösung 7.1
a)

```
                STV   x
                JMN   is_negative
                JMP   fin
is_negative:    ADD   x
                STV   x
                LDC   1
fin:            ADD   x
```

b) i) Sei $m \in \text{Val}^{\text{Adr}}$ der Speicher vor Ausführung des Programms. Für $m(a), m(b) \in \mathbb{K}_{24} \cap \mathbb{N}_+$ gilt $m(a) < m(b)$ genau dann, wenn $m(a) \ominus m(b) < 0$, wobei $\ominus$ die

Subraktion im Zweierkomplement beschreibt.

LDV *b*
NOT
STV *x*
LDC *1*
ADD *x*
ADD *a*
JMN *a_less_b*
JMP *a_geq_b*

ii) Für $m(a), m(b) \in \mathbb{K}_{24}$ gilt nicht mehr, dass $m(a) < m(b)$ genau dann, wenn $m(a) \ominus m(b) < 0$. Zum Beispiel ist $m(a) \ominus m(b) = 1$ für $m(a) = \texttt{0x800000}$ (also die kleinste darstellbare Zahl in $\mathbb{K}_{24}$) und $m(b) = \texttt{0x7fffff}$ (also die größte darstellbare Zahl in $\mathbb{K}_{24}$) obwohl offensichtlich $m(a) < m(b)$ gilt.

Aufgabe 7.2 (2 + 3 + 1 = 6 Punkte)
In dieser Aufgabe geht es darum, die Funktionsweise eines gegebenenen MIMA-Programm zu verstehen. Betrachten Sie dazu das Programm in Abbildung **??**. Die Konstanten *NULL*, *item-ptr*, *next-ptr*, und *x*, bezeichnen darin MIMA-Adressen, die außerhalb des Speicherbereichs des Programms liegen, *START*, *add-next* und *was-last-element* sind Sprungziel-Adressen im Speicherbereich des Programms

a) Geben Sie die Werte an den Speicherstellen *item-ptr*, *next-ptr* und *x* nach der einmaligen Ausführung des Programms vom Start bis inklusive Zeile 16 an. Geben Sie hierfür Ausdrücke an, die die Werte an diesen Adressen in Abhängigkeit vom Speicherzustand $m \in \text{Val}^{\text{Adr}}$ vor der Ausführung des Programmabschnittes beschreiben. Gehen Sie davon aus, dass der bedingte Sprung in Zeile 11 nicht durchgeführt wird.

```
START:          LDC   1
                ADD   item-ptr
                STV   next-ptr
                LDC   0
                STV   x
                STV   NULL
add-next:       LDIV  item-ptr
                ADD   x
                STV   x
                LDIV  next-ptr
                EQL   NULL
                JMN   was-last-element          (11)
                LDIV  next-ptr
                STV   item-ptr
                LDC   1
                ADD   item-ptr
                STV   next-ptr                  (16)
                JMP   add-next
was-last-element: LDV x
                HALT
```

Abbildung 1: Das zu analysierende Programm für Aufgabe 7.**??**

b) Sei nun der folgende Speicherausschnitt gegeben:

Bezeichner	Adresse	Wert
NULL	4	12
x	5	154
next-ptr	6	29
	7	33
	8	5
	9	14
	10	0
	11	29
	12	3
	13	0
	14	17
	15	12
item-ptr	16	8

Das Programm wird nun auf einer MIMA-Maschine ausgeführt, wobei zu Beginn die Adresse *START* des Programmbeginns im IAR steht. Die Ausführung endet, wenn sie den **HALT**-Befehl erreicht.

Das Programm beschreibt die Speicheradresse x mehrfach. Geben Sie die Folge von Werten an, die durch den Ablauf des Programmes an die Addresse x geschrieben werden.

c) Beschreiben Sie informell, was das Programm berechnet.
Hinweis: Können Sie eine Datenstruktur, die Sie bereits in der Vorlesung „Programmieren“ kennengelernt haben, wiedererkennen?

Lösung 7.2

a) Der Speicherinhalt nach Ausführung des Programms vom Start bis Zeile 16 an den Adressen *item-ptr*, *next-ptr* und x lässt sich in Abhängigkeit vom Speicher m vor Ausführung des Programms wie folgt beschreiben:

Ort	Wert
item-ptr	$m(m(\textit{item-ptr})+1)$
next-ptr	$m(m(\textit{item-ptr})+1)+1$
x	$m(m(\textit{item-ptr}))$

b) Die Wert nach jedem Sprung lauten:

Schreibzugriff	Wert an Adresse x
1	0
2	5
3	22
4	25

c) Das Programm bildet die Summe aller Elemente einer (nicht-leeren) einfach verketteten Liste und hinterlegt diese nach Programmausführung im Akkumulator. (*Nicht verlangt*: Dabei stehen die Elemente und die Adresse des nächsten Listenelements jeweils an direkt aufeinanderfolgenden Stellen im Speicher).

Nun sollen Sie eigene kleine Mima Programme entwerfen und implementieren. Verwenden Sie zum Prüfen Ihrer Programme den Mima-Simulator *Mima Flux*, der in der Vorlesung vorgestellt wurde.[1]

Bitte geben Sie die Lösungen für die beiden folgenden Aufgaben als Eingabedateien für das Programm *Mima Flux*, die von den Tutor:innen in diesem Programm geladen und getestet werden können.

Assemblerprogramme sind sind meist eher kryptisch und schwerer zu verstehen. Machen Sie sich und Ihrem:r Tutor:in eine Gefallen und verteilen Sie großzügig erklärende Kommentare über das Programm. Unzureichende Dokumentation (in Kommentaren) führt zu Punktabzug.

Aufgabe 7.3 (8 Punkte)

[1] Mehr Information auf der Ilias-Seite. Verfügbar unter: `https://github.com/mattulbrich/mimaflux`

Sie haben auf dem vierten Aufgabenblatt die *Spiegelung*sabbildung $\bullet^R : A^* \to A^*$ für ein Alphabet A kennengelernt. Zur Erinnerung:

$$|w^R| = |w|$$
$$w^R(i) = w(|w| - i - 1) \quad \text{für alle } 0 \leq i < |w| \ .$$

Schreiben Sie ein Programm, dass ein Wort $w \in Z_2^{24}$ aus dem Speicher liest und dessen Spiegelung $w^R \in Z_2^{24}$ an eine andere Speicherstelle schreibt.

Geben Sie hierzu eine Datei im Eingabeformat von *Mima Flux* ab. Das Programm soll seine Eingabe von einer Speicherstelle mit dem symbolischen Namen *in* lesen und seine Ausgabe an die Speicherstelle mit dem symbolischen Namen *out* schreiben und dann anhalten.

Lösung 7.3

```
1
2  ; Aufgabe: Schreiben Sie ein Programm, dass das inverse Wort, das an <in> steht
3  ; an <out> schreibt.
4
5
6  in:  DS 0xf00005
7  out: DS
8
9  i: DS
10
11 START:
12   LDC 0
13   STV i
14   STV out
15
16 next:
17   ; double out
18   LDV out
19   ADD out
20   STV out
21
22   ; add lowest bit of in to out
23   LDC 1
24   AND in
25   ADD out
26   STV out
27
28   ; increment counter
29   LDC 1
30   ADD i
31   STV i
32   LDC 24
33   EQL i
34   JMN ende
35
36   ; eines weiter gehen
37   LDV in
38   RAR
```

```
39    STV in
40
41    JMP next
42
43  ende:
44    ; div in by two
45    LDV in
46    RAR
47    STV in
48    HALT
```

Aufgabe 7.4 (4 Bonus[2]-Punkte)
Sie haben auf dem letzten Übungsblatt die Funktion *to_string* kennengelernt, die einen 0-terminierten Speicherbereich als Zeichenkette interpretiert. In dieser Bonus-Aufgabe sollen Sie ein Programm schreiben, dass die Reihenfolge der Zeichen in einem 0-terminierten Speicherbereich umkehrt. Ihr Programm soll folgende Spezifikation erfüllen:

> Die Eingabe des Programmes steht in *array*. Die Speicherstelle mit dem symbolischen Namen *array* enthalte zu Beginn als Wert die Adresse a des Beginns eines Speicherbereichs. Die Adresse b des Endes des Speicherbereichs ist die erste Adresse mit $b \geq a$, an der Wert 0 steht.
>
> Nach Ausführung des Programmes sollen die Werte zwischen den Speicherstellen a (einschließlich) und b ausschließlich in umgekehrter Reihenfolge im Speicher stehen.

Sie dürfen annehmen, dass es eine Speicherstelle jenseits von a gibt, die den Wert 0 enthält.

Lösung 7.4

```
1
2   ; Reverse a 0-terminated array
3
4
5   * = 0x10
6   array:
7     DS 1
8     DS 2
9     DS 3
10    DS 0
11
12  * = 0x0
13  ; variables:
14  ref0:  DS
15  ref1:  DS
16  tmp:   DS
17
18  ; PROGRAM address:
19  * = 0x20
```

[2]Die Punkte dieser Aufgabe zählen nicht zur Gesamtpunktzahl, die über das Bestehen des Übungsscheines entscheidet.

```
20
21 START:         ; The IAR will be initially set to
22                 ; the value of the label "START".
23
24   LDC array
25   STV ref0
26   STV ref1
27
28 ;;; Find the zero
29
30 find0:
31   LDIV ref1
32   STV tmp
33   LDC 0
34   EQL tmp
35   JMN step2
36
37   LDC 1
38   ADD ref1
39   STV ref1
40   JMP find0
41
42 step2:
43   LDC 0
44   NOT
45   ADD ref1
46   STV ref1
47
48 ;;; Now reverse the string
49
50 reverse:
51   LDV ref1
52   NOT
53   STV tmp
54   LDC 1
55   ADD tmp
56   ADD ref0
57   NOT
58   JMN end
59
60   ;; swap
61   LDIV ref0
62   STV tmp
63   LDIV ref1
64   STIV ref0
65   LDV tmp
66   STIV ref1
67
68   ;; ref0 ++
69   LDC 1
70   ADD ref0
71   STV ref0
72
```

```
73      ;; ref1 --
74      LDC 0
75      NOT
76      ADD ref1
77      STV ref1
78      JMP reverse
79
80  end: HALT
```