Lösung 4

Aufgabe 20

Die Aufgabe ist nach Kochrezept "Elementweises Modifizieren eines Arrays am Ort" lösbar. Da auf jede Zahl mehrfach lesend zugegriffen werden muß, legt man in TEMP1 zunächst eine Kopie der Zahl an. Aus dieser Kopie heraus setzt man stückweise mittels Rotation und Maskierung in TEMP2 die gewünschte Struktur zusammen (& = Konjunktion, $\vee$ = Disjunktion, RR2 = Rechtsrotieren um 2 Bits, RL2 = Linksrotieren um 2 Bits):

1. Schritt: TEMP2 := RR2(TEMP1)&MASKE1

B1	**B0**	0	0	0	0	0	0	0	0

2. Schritt: TEMP2 := TEMP2 $\vee$ RL2(TEMP1)&MASKE2

B1	**B0**	0	0	0	0	0	0	**B9**	**B8**

3. Schritt: TEMP2 := TEMP2 $\vee$ TEMP1&MASKE3

B1	**B0**	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B9**	**B8**

Die **fett** markierten Zeilen sind aufgabenspezifisch.

```
; TAUSCH.LC1

M1:     LDA ANZ    ; A := <ANZ>
        LDB EINS   ; B := 1
        SUB        ; A := A - 1
        MOV ANZ    ; <ANZ> := A
        JPS M99    ; IF SF = 1 GOTO M99
        ;
M10:    LDA ZAHL   ; A := <ZAHL>
        ;
        MOV TEMP1  ; <TEMP1> := A
        RRA 2      ; A := A1,A0,A9,...,A2
        LDB MASKE1 ; B := 1100000000
        AND        ; A := A & 1100000000
        MOV TEMP2  ; <TEMP2> := A
        LDA TEMP1  ; A := <TEMP1>
        RLA 2      ; A := A7,...,A0,A9,A8
        LDB MASKE2 ; B := 0000000011
        AND        ; A := A & 0000000011
        LDB TEMP2  ; B := <TEMP2>
        ADD        ; A := A + B
        MOV TEMP2  ; <TEMP2> := A
        LDA TEMP1  ; A := <TEMP1>
        LDB MASKE3 ; B := 0011111100
        AND        ; A := A & 0011111100
        LDB TEMP2  ; B := <TEMP2>
        ADD        ; A := A + B
M20:    MOV ZAHL   ; <ZAHL> := A
```

```
        LDB EINS   ; B := 1
        LDA M10    ; A := <M10>
        ADD        ; A := A + 1
        MOV M10    ; <M10> := A
        LDA M20    ; A := <M20>
        ADD        ; A := A + 1
        MOV M20    ; <M20 := A
        JMP M1     ; GOTO M1
M99:    HLT        ; HALT
        ;
EINS:   DEF 1      ; KONSTANTE 1
        ;
TEMP1:  DEF 0      ; ZWISCHENSPEICHER 1
TEMP2:  DEF 0      ; ZWISCHENSPEICHER 2
MASKE1: DEF -256   ; MASKE 1100000000
MASKE2: DEF 3      ; MASKE 0000000011
MASKE3: DEF 252    ; MASKE 0011111100
        ;
ANZ:    DEF 5      ; ANZAHL = 5
ZAHL:   DEF 511    ; 0111111111 -> 1111111101,  511 ->   -3
        DEF -3     ; 1111111101 -> 0111111111,   -3 ->  511
        DEF -256   ; 1100000000 -> 0000000011, -256 ->    3
        DEF 3      ; 0000000011 -> 1100000000,    3 -> -256
        DEF -1     ; 1111111111 -> 1111111111,   -1 ->   -1
```

Aufgabe 21

Aus dem gegebenen Quelltext

```
DEF -510
DEF -8
DEF 1
DEF 73
DEF 448
DEF -440
DEF 256
DEF 129
DEF -64
DEF 1
```

muß zunächst der Inhalt der damit belegten Speicherzellen in binärer Form gefunden werden:

```
0  DEF -510  ———>  0  1000000010        s. Anmerkung
1  DEF -8    ———>  1  1111111000
2  DEF 1     ———>  2  0000000001
3  DEF 73    ———>  3  0001001001
4  DEF 448   ———>  4  0111000000
5  DEF -440  ———>  5  1001001000
6  DEF 256   ———>  6  0100000000
7  DEF 129   ———>  7  0010000001
8  DEF -64   ———>  8  1111000000
9  DEF 1     ———>  9  0000000001
```

Anmerkung

Die Binärdarstellung negativer Zahlen kann auf zwei unterschiedlichen Wegen berechnet werden:

1. Man ermittelt zunächst die Binärdarstellung der betragsgleichen positiven Zahl und bildet anschließend

davon das Zweierkomplement. Für die Zahl -510 ergibt sich

$510 = 1*256 + 15*16 + 14*1 = 1FE_{16} = 0111111110_2$
$-510 = 1000000010_2$

2. Man erinnert sich daran, daß im Zahlenraum der vorzeichenbehafteten 10-Bit-Zahlen eine negative Zahl z genauso kodiert wird wie die vorzeichenlose 10-Bit-Zahl $z' = 1024 - |z|$. Für die Zahl -510 ergibt sich

$1024 - |-510| = 1024 - 510 = 514$
$514 = 2*256 + 0*16 + 2*1 = 202_{16} = 1000000010_2$

Um herauszufinden, wie der LC1 dieses Bitmuster interpretiert, sind grundsätzliche Kenntnisse der Abläufe im LC1 erforderlich. Der PC wird hardwaremäßig mit dem Wert 0 initialisiert. Danach wird die erste Fetchphase gestartet. Folglich interpretiert der LC1 jedes Bitmuster auf Adresse 0 als Befehl:

0 1000000010 ——> 0 1000_000010 ——> JMP 2

Auf Adresse 0 steht ein unbedingter Sprung zur Adresse 2. Dort muß offensichtlich wieder ein Befehl stehen. Als was der Inhalt der Speicherzelle 1 interpretiert wird, kann zur Zeit noch nicht gesagt werden.

1 1111111000 ——> ???

2 0000000001 ——> 2 0000_000001 ——> LDA 1

Der Ladebefehl liest den Inhalt der Speicherzelle 1. Es ist zu vermuten, daß dort Daten stehen. Es ist aber noch nicht ausgeschlossen, daß auf Adresse 1 ein Befehl steht, der modifiziert wird.

Der Ladebefehl inkrementiert den PC. Auf Adresse 3 steht wieder ein Befehl:

3 0001001001 ——> 3 0001_001001 ——> LDB 9

Der Ladebefehl inkrementiert den PC. Auf Adresse 4 steht wieder ein Befehl:

4 0111000000 ——> 4 0111_000000 ——> NOT

Der Negationsbefehl inkrementiert den PC. Auf Adresse 5 steht wieder ein Befehl:

5 1001001000 ——> 5 1001_001000 ——> JPS 8

Der bedingte Sprungbefehl inkrementiert den PC oder überschreibt ihn mit dem Wert 8. Auf den Adressen 6 und 8 stehen Befehle:

6 0100000000 ——> 6 0100_000000 ——> ADD

Der Additionsbefehl inkrementiert den PC. Auf Adresse 7 steht wieder ein Befehl:

```
7  0010000001  ———>  7  0010_000001   ———>  MOV 1
8  1111000000  ———>  8  1111_000000   ———>  HLT
```

Die Adresse 9 wird durch den PC nicht erreicht. Dort stehen Daten:

```
9  0000000001  ———>  9  00_0000_0001  ———>  DEF 1
```

Abschließend kann festgestellt werden, daß die Adresse 1 durch den PC ebenfalls nicht erreicht wird. Dort stehen ebenfalls Daten:

```
                   s. Aufgabenstellung
1  1111111000  ————————————————————————————>  DEF -8
```

Damit ergibt sich insgesamt:

```
0  JMP 2
1  DEF -8
2  LDA 1
3  LDB 9
4  NOT
5  JPS 8
6  ADD
7  MOV 1
8  HLT
9  DEF 1
```

Mit etwas Phantasie folgt daraus schließlich

```
       JMP M1   ; GOTO M1
WERT:  DEF -8   ; Z
M1:    LDA WERT ; A := <WERT>
       LDB EINS ; B := 1
       NOT      ; A := /A
       JPS M99  ; IF A < 0 GOTO M99
       ADD      ; A := A + 1
       MOV WERT ; <WERT> := A
M99:   HLT      ; HALT
EINS:  DEF 1    ; KONSTANTE 1
```

Für den Fall, daß <WERT> negativ ist, wird <WERT> mit der betragsgleichen positiven Zahl |<WERT>| überschreiben, andernfalls wird <WERT> unverändert gelassen. Das Programm realisiert also die Betragsbildung <WERT> := |<WERT>|.