

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

3. Aufgabenblatt mit Lösungen

Problem 1: Gleitkomma-Darstellung (2+2+2+2=8)

Stellen Sie die Zahlen $2/3$ und $-2/7$ als IEEE single unter Verwendung der

- „round-to-even“-Regel
- Rundung zum nächsten Gleitkommawert in Richtung 0
- Rundung zum nächsten Gleitkommawert in Richtung ∞
- Rundung zum nächsten Gleitkommawert in Richtung $-\infty$

dar.

Solution 1

IEEE-P 754 Single (Bitverteilung: 1 VZ, 8 Charakteristik, 23 Mantisse):

Tabelle 1: Zusammenfassung des 32-Bit-IEEE-Formats

Charakteristik	Zahlenwert
0	$(-1)^{VZ} 0, \text{Mantisse} * 2^{-126}$
1	$(-1)^{VZ} 1, \text{Mantisse} * 2^{-126}$
$2..2^8 - 2 = 254$	$(-1)^{VZ} 1, \text{Mantisse} * 2^{\text{Charakteristik}-127}$
254	$(-1)^{VZ} 1, \text{Mantisse} * 2^{-127}$
255	Mantisse = 0: $(-1)^{VZ} \infty$ overflow
255	Mantisse \neq 0: NaN (not a number)

$$\text{Bias/Exzess} = 2^{8-1} - 1 = 127$$

(i) $\frac{2}{3} = 0, \overline{10}_2 = 1, \overline{01}_2 * 2^{-1}$
VZ = 0

Exponent = -1 \Rightarrow Charakteristik = Exponent + Bias = $-1 + 2^{8-1} - 1 = 124_{10} = 01111100_2$
Mantisse = $01010101010101010101010(1) = (01)_{11}0(1)$

(ii) $-\frac{2}{7} = 0,0\overline{100}_2 = 1, \overline{001}_2 * 2^{-2}$
VZ = 1

Exponent = -3 \Rightarrow Charakteristik = Exponent + Bias = $-2 + 2^{8-1} - 1 = 123_{10} = 01111011_2$
Mantisse = $00100100100100100100(1) = (001)_{11}00(1)$

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

Tabelle 2: Mantissen nach Anwendung verschiedener Rundungsregeln

	VZ + Charakteristik	a) round-to-even	b) $\rightarrow 0$	c) $\rightarrow \infty$	d) $\rightarrow -\infty$
(i) $\frac{2}{3}$	0 01111100	(01) ₁₁ 0	(01) ₁₁ 0	(01) ₁₁ 1	(01) ₁₁ 0
(ii) $-\frac{2}{7}$	1 01111011	(001) ₇ 00	(001) ₇ 00	(001) ₇ 00	(001) ₇ 01

Problem 2: Carry-lookahead-Addierer (2+2+2=6)

- a) Lösen Sie die rekursive Berechnung von \ddot{u}_i dem Skript entsprechend in Und- und Oder-Verknüpfungen von g_i und p_i mit $i \leq 4$ auf.
- b) Berechnen Sie g_i und p_i mit $i \leq 4$ für $a_{4..1} = 0011$ und $b_{4..1} = 1101$.
- c) Berechnen Sie die ersten vier Ziffern s_i mit $i \leq 4$ der Summe von a und b aus b) entsprechend der Carry-lookahead-Addierer-Methode.

Solution 2

a)

$$\begin{aligned}
 \ddot{u}_0 &= 0 \\
 \ddot{u}_1 &= g_0 \vee p_0 \ddot{u}_0 \\
 \ddot{u}_2 &= g_1 \vee p_1 g_0 \vee p_1 p_0 \ddot{u}_0 \\
 \ddot{u}_3 &= g_2 \vee p_2 g_1 \vee p_2 p_1 g_0 \vee p_2 p_1 p_0 \ddot{u}_0 \\
 \\
 \ddot{u}_4 &= g_3 \vee p_3 \ddot{u}_3 \\
 &= g_3 \vee p_3 (g_2 \vee p_2 g_1 \vee p_2 p_1 g_0 \vee p_2 p_1 p_0 \ddot{u}_0) \\
 &= g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 g_0 \vee p_3 p_2 p_1 p_0 \ddot{u}_0
 \end{aligned}$$

- b) Es gilt $g_i = a_i b_i$ und $p_i = a_i \oplus b_i$. Damit ergeben sich $g_{0..3}$ und $p_{0..3}$ mit $a = 0101$ und $b = 1101$ folgendermaßen:

$$\begin{array}{l|l}
 g_0 = a_0 b_0 = 1 \wedge 1 = 1 & p_0 = a_0 \oplus b_0 = 1 \oplus 1 = 0 \\
 g_1 = a_1 b_1 = 0 \wedge 0 = 0 & p_1 = a_1 \oplus b_1 = 0 \oplus 0 = 0 \\
 g_2 = a_2 b_2 = 1 \wedge 1 = 1 & p_2 = a_2 \oplus b_2 = 1 \oplus 1 = 0 \\
 g_3 = a_3 b_3 = 0 \wedge 1 = 0 & p_3 = a_3 \oplus b_3 = 0 \oplus 1 = 1
 \end{array}$$

- c) Nach der Carry-lookahead-Addierer-Methode ergibt sich für die Addition von a und b das Ergebnis 0010 mit dem Übertrag 1:

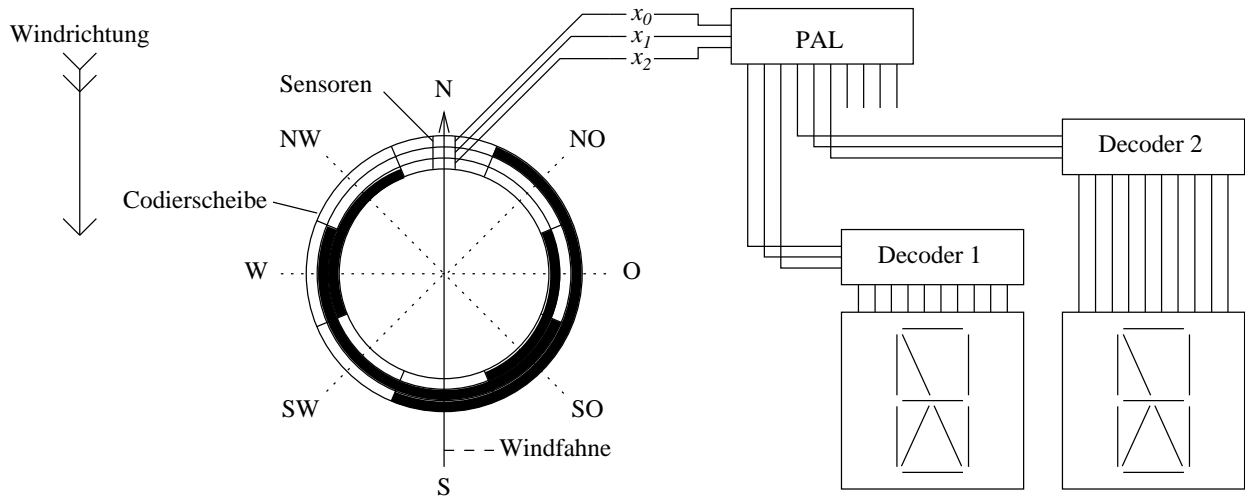
$$\begin{array}{l|l}
 s_0 = & p_0 \oplus \ddot{u}_0 = 0 \oplus 0 = & 0 \\
 s_1 = & p_1 \oplus \ddot{u}_1 = p_1 \oplus (g_0 \vee p_0 \ddot{u}_0) = 0 \oplus (1 \vee 0 \wedge 0) = 0 \oplus 1 = & 1 \\
 s_2 = & p_2 \oplus \ddot{u}_2 = p_2 \oplus (g_1 \vee p_1 g_0 \vee p_1 p_0 \ddot{u}_0) = 0 \oplus (0 \vee 0 \vee 0) = 0 \oplus 0 = & 0 \\
 s_3 = & p_3 \oplus \ddot{u}_3 = p_3 \oplus (g_2 \vee p_2 g_1 \vee p_2 p_1 g_0 \vee p_2 p_1 p_0 \ddot{u}_0) = 1 \oplus (1 \vee 0 \vee 0 \vee 0) = 1 \oplus 1 = & 0 \\
 \ddot{u}_4 = & g_3 \vee p_3 g_2 \vee p_3 p_2 g_1 \vee p_3 p_2 p_1 g_0 \vee p_3 p_2 p_1 p_0 \ddot{u}_0 = 0 \vee 1 \vee 0 \vee 0 \vee 0 \vee 0 = & 1
 \end{array}$$

Problem 3: Windmessung mittels PAL (4+4=8)

1. Konstruiere durch Programmierung eines PALs einen Decoder für die in Abbildung (a) dargestellte 10-Segment-Anzeige.

TI II

Sommersemester 2010
PD Dr. Katinka Wolter



Solution 3

1. Herleitung

$$\begin{aligned}
 s_0 &= x_1 \\
 s_1 &= \overline{x_0} \\
 s_2 &= (\overline{x_0} \wedge \overline{x_1}) \vee (\overline{x_0} \wedge x_1 \wedge x_2) \\
 &= \overline{x_0} \wedge (\overline{x_1} \vee (x_1 \wedge x_2)) \\
 s_3 &= x_1 \\
 s_4 &= \overline{x_0} \\
 s_5 &= s_2 \\
 s_6 &= \overline{x_0} \wedge \overline{x_1} \wedge x_2 \\
 s_7 &= (\overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2}) \vee (\overline{x_0} \wedge \overline{x_1} \wedge x_2) \\
 s_8 &= \overline{x_0} \wedge x_1 \wedge \overline{x_2} \\
 s_9 &= \overline{x_0} \wedge \overline{x_1} \wedge \overline{x_2}
 \end{aligned}$$

Die Kodierung $x_0 = 1; x_1 = 0; x_2 = 0$ wird so umgesetzt, dass $x_0 = 1; x_1, x_2$ beliebig. Es genügt nun für jeden Ausgang s_i dann $\overline{x_0} = 1$ zu setzen, falls dies nicht schon durch die oben aufgestellten Formeln der Fall war.

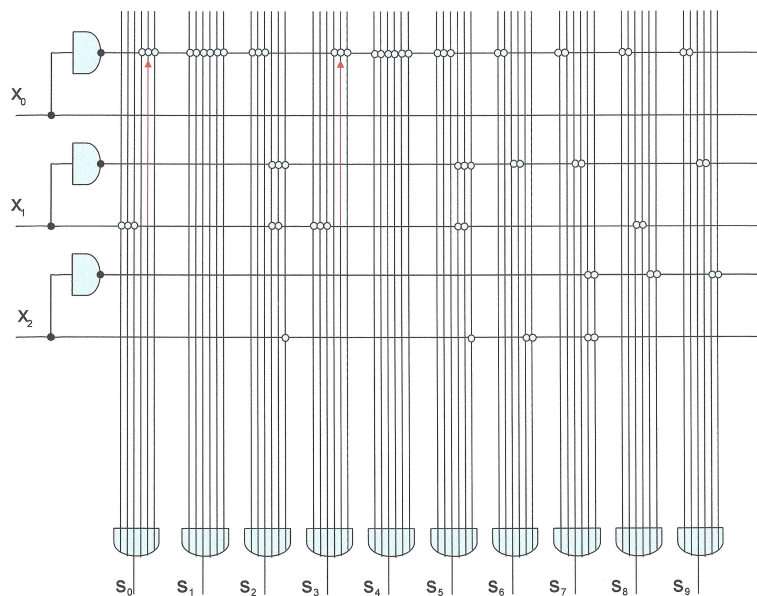
2. Herleitung

Eingänge				Anzeige 1			Anzeige 2		
x_0	x_1	x_2		d_{10}	d_{11}	d_{12}	d_{20}	d_{21}	d_{22}
0	0	0	SO	0	1	0	0	1	1
0	0	1	W	0	0	1	1	0	0
0	1	0	O	0	1	1	1	0	0
0	1	1	NW	0	0	0	0	0	1
1	0	0	S	0	1	0	1	0	0
1	0	1	SW	0	1	0	0	0	1
1	1	0	NO	0	0	0	0	1	1
1	1	1	N	0	0	0	1	0	0

Bilde Karnaugh Diagramme:

TI II

Sommersemester 2010
PD Dr. Katinka Wolter



PAL

Abbildung 1: Kodierer LCD Anzeige

	x ₂	\bar{x}_2		
x ₀	0	0	0	0
\bar{x}_0	0	0	0	0
	x ₁	\bar{x}_1	x ₁	

	x ₂	\bar{x}_2		
x ₀	0	1	1	0
\bar{x}_0	0	0	1	1
	x ₁	\bar{x}_1	x ₁	

	x ₂	\bar{x}_2		
x ₀	0	0	0	0
\bar{x}_0	0	1	0	1
	x ₁	\bar{x}_1	x ₁	

	x ₂	\bar{x}_2		
x ₀	1	0	1	0
\bar{x}_0	0	1	0	1
	x ₁	\bar{x}_1	x ₁	

	x ₂	\bar{x}_2		
x ₀	0	0	0	1
\bar{x}_0	0	0	1	0
	x ₁	\bar{x}_1	x ₁	

	x ₂	\bar{x}_2		
x ₀	0	1	0	1
\bar{x}_0	1	0	1	0
	x ₁	\bar{x}_1	x ₁	

$$d_{10} = 0$$

$$d_{11} = (\bar{x}_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (x_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (x_0 \vee x_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee x_2)$$

$$d_{12} = (\bar{x}_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee x_2) \wedge (\bar{x}_0 \vee x_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee x_1 \vee x_2) \wedge (x_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (x_0 \vee x_1 \vee x_2)$$

$$d_{20} = (x_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee x_1 \vee \bar{x}_2) \wedge (x_0 \vee x_1 \vee x_2) \wedge (\bar{x}_0 \vee \bar{x}_1 \vee x_2)$$

$$d_{21} = (\bar{x}_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee x_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee x_1 \vee x_2) \wedge (x_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (x_0 \vee x_1 \vee \bar{x}_2) \wedge (x_0 \vee \bar{x}_1 \vee x_2)$$

$$d_{22} = (\bar{x}_0 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (x_0 \vee x_1 \vee \bar{x}_2) \wedge (\bar{x}_0 \vee x_1 \vee x_2) \wedge (x_0 \vee \bar{x}_1 \vee x_2)$$

Problem 4: Halbaddierer und Volladdierer (3+3+4=10)

Ein NOR-Gatter ist ein Schaltnetz mit zwei Eingängen und einem Ausgang. Am Ausgang liegt genau dann eine 1 vor, wenn an keinem Eingang eine 1 anliegt.

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

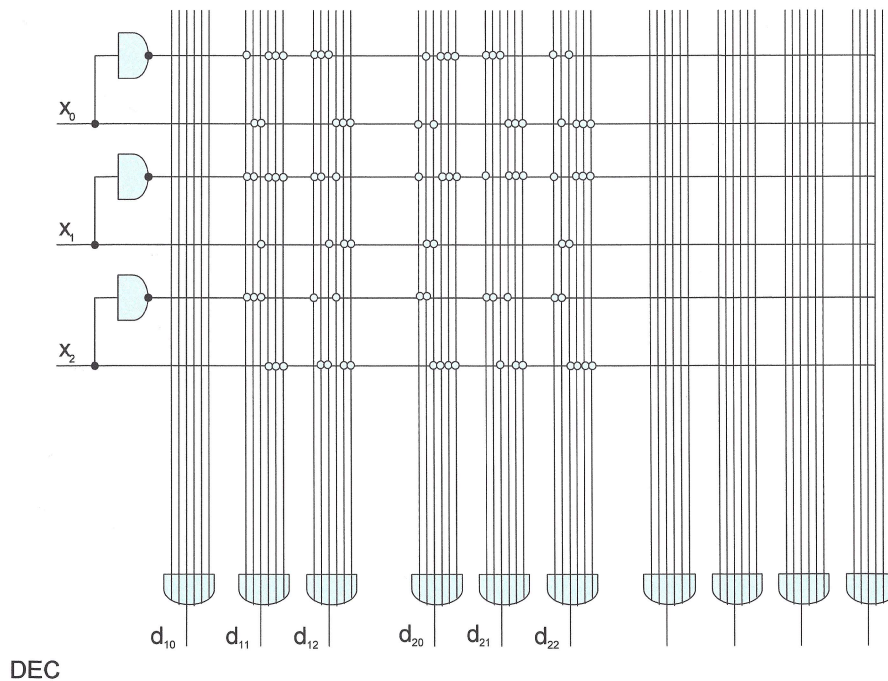


Abbildung 2: Dekodierer Windmesser

- Entwerfen Sie eine Schaltung für einen Halbaddierer (Resultat und Übertrag), der ausschließlich aus NOR-Gattern mit 2 Eingängen besteht (Herleitung!). Verwenden Sie nicht mehr als 5 NOR-Gatter.
- Wie sieht der aus NOR-Gattern bestehende Volladdierer aus (für die Halbaddierer jeweils ein Symbol verwenden)?
- Wir nehmen nun an, daß ein NOR-Gatter eine Schaltzeit von 20 psec hat. Wann liegt bei Ihrem Halbaddierer das Resultat vor? Wann der Übertrag? Wie sehen die Zeiten für Ihren Volladdierer aus?

Solution 4

- (a) Halbaddierer aus nicht mehr als 5 Nor-Gattern

	x	y	s	c
	0	0	0	0
Ausgaben des Halbaddierers:	0	1	1	0
	1	0	1	0
	1	1	0	1

Verwende De Morgan'sche Regel und die Darstellung der Negation:

x	$\neg x = \text{NOR}(x,x)$
0	1
1	0

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

Summe s und Übertrag c umgeschrieben:

$$\begin{aligned}c &= xy \\ &\equiv \neg\neg(xy) \\ &\equiv \neg(\neg x \vee \neg y) \\ &\equiv \neg(\neg(x \vee x) \vee \neg(y \vee y)) \\ s &= \neg xy \vee x\neg y \\ &\equiv \neg\neg(\neg xy \vee x\neg y) \\ &\equiv \neg((x \vee \neg y)(\neg x \vee y)) \\ &\equiv \neg(xy \vee \neg x\neg y) \\ &\equiv \neg(c \vee \neg(x \vee y))\end{aligned}$$

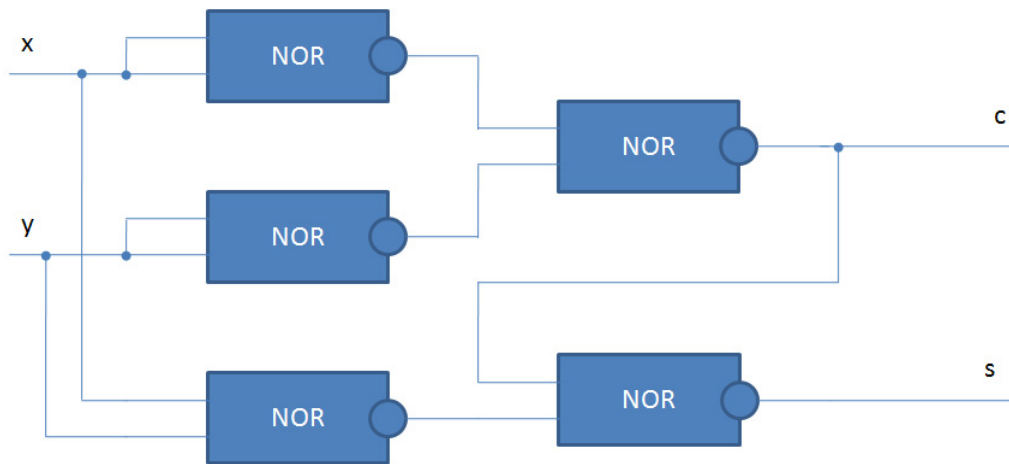


Abbildung 3: Halbaddierer

(b) Volladdierer aus NOR-Gattern

Ausgaben:

x	y	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ausgangsgleichungen

$$\begin{aligned}s &= x \oplus y \oplus c_{in} \\ c_{out} &= xc_{in} \vee y_{in} \vee xy\end{aligned}$$

Entweder langwieriges Umformen wie in a) oder Schaltung dem Skript entnehmen und das OR-Gatter durch $\text{NOR}(\text{NOR}(x,y), \text{NOR}(x,y))$ ersetzen:

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

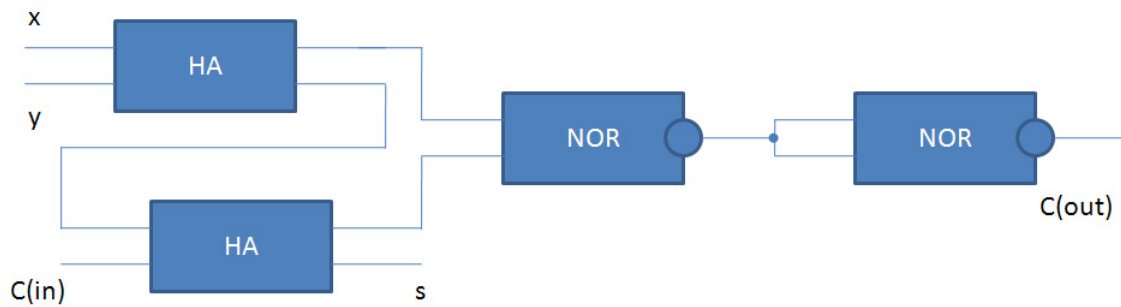


Abbildung 4: Volladdierer

- (c) Es habe ein NOR-Gatter eine Schaltzeit von 20 psek, dann ist für jede Ausgabe (Summe oder Übertrag) der längste Weg bzgl. der Anzahl der Gatter zu finden. Dieser gibt die Schaltzeit bis zur endgültigen Ausgabe an:
HA: Übertrag: 40 psek, Summe: 60 psek
VA: Übertrag: (Übertrag(HA) + Summe(HA) + 2 x Schaltzeit NOR-Gatter) psek = 140 psek,
Summe: 2*Summe(HA) = 120 psek

Problem 5: Assembler - Quadratwurzel (6)

Schreiben Sie ein MMIX-Assemblerprogramm, das die folgende Aufgabenstellung löst:

Das Programm soll den ganzzahligen Anteil x der Quadratwurzel einer gegebenen ganzen Zahl z berechnen, d.h. gesucht wir x mit $x = \max\{x \in \mathbb{Z} \mid x^2 \leq z\}$, bzw.

$$\text{result} = \lfloor \sqrt{\text{value}} \rfloor .$$

Verwenden Sie dabei die sich aus der Newton-Methode ergebene Intervallschachtelung:

```
a := value
b := 0
while |a - b| > 1
    a := [(a + b)/2]
    b := [value/a]
end_while
result := min{a, b}
```

Solution 5

MMIX-Assemblerprogramm, das ganzzahligen Anteil der Quadratwurzel einer ganzen Zahl berechnet nach der Newtonschen Intervallschachtelung:

Listing 1: sqrt.mms

```
1 a      IS $40
2 b      IS $41
3 value  IS $42
4 cmp    IS $10
5 result IS $43
6
7        LOC #100
```

TI II

Sommersemester 2010

PD Dr. Katinka Wolter

```
8
9 Main      SET value,25
10          SET a,value    // a := value
11          SET b,0        // b := 0
12          CMP cmp,a,1    // für a=1 Funktioniert der Algorithmus nicht
13          BZ  cmp,a_is_min
14  while    SUB cmp,a,b    // while |a-b| > 1, ganzzahliger Rest /= 0
15
16          BZ  cmp,end_while // Differenz = 0 => Abbruch
17          SUB cmp,cmp,1    // 0 <Differenz < 1 => Abbruch
18          BZ  cmp,end_while
19          ADD cmp,cmp,2    // -1 < Differenz < 0
20          BZ  cmp,end_while
21
22
23          ADD a,a,b        // a := (a+b)/2
24          DIV a,a,2
25          DIV b,value,a    // b := value/a
26          JMP while
27
28  end_while  CMP cmp,a,b    // result := min{a, b}
29            BN  cmp,a_is_min
30  b_is_min   SET result,b
31            JMP end
32  a_is_min   SET result,a
33
34  end       TRAP 0,Halt,0
```

Hier noch eine sehr schöne und komfortable alternative Lösung.

Thanks to Flutter,Höfken und Seifert

Listing 2: Alternative Lösung

```
1          LOC      Data_Segment
2          GREG     @
3
4  eintxt     BYTE   "Bitte_eine_Zahl_von_0_bis_9999999_eingeben.",0
5  austxt     BYTE   "Ganzzahliger_Anteil_der_Quadratwurzel_von_",0
6  aus2txt    BYTE   "_ist:",0
7  ausendtxt  BYTE   ".",10,0
8  zahltxt    BYTE   "freihaltener_Speicher_zur_Ausgabe_von_Zahlen",0
9  InBuf      BYTE   0
10  Args      OCTA   InBuf,8
11
12  a          IS     $1
13  b          IS     $2
14  result     IS     $3
15  tmp        IS     $4
16  value      IS     $5
17
18          LOC      #100
19
20  Main       PUSHJ  $4,zahlein
21            SET    value,$255
22
23  %ist die Eingabe 1 funktioniert der Algorithmus nicht
24  %abfangen des Fehlers
25            CMP    tmp,value,1
26            BNP   tmp,alterend
```

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

```
27
28         SET     a,value
29         SET     b,0
30
31         %result=a-b
32 while      SET     result,a
33         SUB     result,result,b
34         CMP     tmp,result,0
35         BNN    tmp,gr0
36
37         %wenn a-b<0 wird der Betrag genommen
38         SET     tmp,0
39         SUB     tmp,tmp,result
40         SET     result,tmp
41
42 gr0      CMP     result,result,1
43
44         %wenn der Betrag<1 springe aus Schleife raus
45         BNP    result,out
46
47         %a = |(a+b)/2|
48         ADD     a,a,b
49         DIV     a,a,2
50
51         %b=value/a
52         SET     tmp,value
53         DIV     b,tmp,a
54
55         %zurück zum Beginn der Schleife
56         JMP     while
57
58 %wenn a<b result=a
59 isa      SET     result,a
60         JMP     outtxt
61
62 %vergleich von a und b
63 out      CMP     result,a,b
64         BN     result,isa
65
66 %wenn b<=a result=b
67 isb      SET     result,b
68         JMP     outtxt
69
70 %wurzel aus 1 ist 1
71 alterend SET     result,1
72
73 %Ausgabe des Ergebnisses
74 outtxt   LDA     $255,austxt
75         TRAP    0,Fputs,StdOut
76         PUSHJ   $4,zahlaus
77         LDA     $255,aus2txt
78         TRAP    0,Fputs,StdOut
79         PUSHJ   $2,zahlaus
80         LDA     $255,ausendtxt
81         TRAP    0,Fputs,StdOut
82 endaus   TRAP    0,Halt,0
83
84
```

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

```
85 *****
86 *** Unterprozedur zur Ausgabe einer Zahl
87 *****
88 var          IS      $0
89 tmpaus       IS      $10
90 adr          IS      $11
91 len          IS      $12
92
93 %erste Schleife bestimmt, wie viele Stellen die Zahl hat (+1 Stelle)
94 zahlaus      SET     len,0
95              SET     adr,1
96 grloop       ADD     len,len,1          %solange 10^len <= var
97              MUL     adr,adr,10        %wird len um 1 erhöht
98              CMP     tmpaus,adr,var
99              BNP     tmpaus,grloop
100
101 %letzte stelle ist 0, damit die Ausgabe endet
102 anhang       LDA     adr,zahltxt
103              SET     tmpaus,0
104              STB     tmpaus,adr,len
105              SUB     len,len,1
106
107 %Schleife, die die Zahl zerlegt, in ASCII umwandelt und speichert
108 ausloop      LDA     adr,zahltxt
109              DIV     var,var,10
110              GET     tmpaus,rR
111              ADD     tmpaus,tmpaus,48   %zu ASCII
112              STB     tmpaus,adr,len     %speichern
113              SUB     len,len,1         %nächstes Byte
114              BN     len,ausgabe        %zu ausgabe wenn fertig
115              JMP     ausloop
116
117 %Ausgabe der Zahl
118 ausgabe      LDA     $255,zahltxt
119              TRAP    0,Fputs,StdOut
120              POP     0,0
121
122
123 *****
124 *** Unterprozedur zur Eingabe einer Zahl
125 *****
126 tmpein       IS      $10
127 einlen       IS      $11
128 stelle       IS      $12
129 adre         IS      $13
130
131 %Aufforderung zu Eingabe einer Zahl und einlesen
132 zahlein      LDA     $255,eintxt
133              TRAP    0,Fputs,StdOut
134              LDA     $255,Args
135              TRAP    0,Fgets,StdIn
136              SET     einlen,0
137
138 %Schleife zum feststellen wie lang die Zahl ist
139 lenloop      LDA     adre,InBuf
140              LDB     stelle,adre,einlen
141              CMP     tmpein,stelle,48
142              BN     tmpein,getz
```

```
143          CMP      tmpein,stelle,57
144          BP       tmpein,getz
145          ADD      einlen,einlen,1
146          JMP      lenloop
147
148 %Länge der Zahl wird dekrementiert, damit der Wert stimmt
149 getz      SUB      einlen,einlen,1
150          JMP      power
151
152 %ist der Exponent 0 ist 10^0 = 1
153 p0        SET      einlen,1
154          JMP      p0end
155
156 %einlen = 10^einlen
157 power     SET      tmpein,1
158          BZ       einlen,p0
159 ploop     MUL      tmpein,tmpein,10
160          SUB      einlen,einlen,1
161          BNZ      einlen,ploop
162          SET      einlen,tmpein
163 p0end     LDA      adre,InBuf
164
165 %Zahl wird Stelle für Stelle mit der entsprechenden 10er
166 %Potenz multipliziert und aufaddiert
167 getzloop  LDB      tmpein,adre,0
168          SUB      tmpein,tmpein,48
169          MUL      tmpein,tmpein,einlen
170          ADD      $0,$0,tmpein
171          DIV      einlen,einlen,10
172          ADD      adre,adre,1
173          CMP      tmpein,einlen,0
174          BNP      tmpein,einend
175          JMP      getzloop
176
177 %zwischen speichern der Zahl im Register 255
178 einend    SET      $255,$0
179          POP      0,0
```

Problem 6: MMIX – Unterprogramme (5+5=10)

Informieren Sie sich über die Erstellung von Unterprogrammen in MMIX.

- a) Implementieren Sie ein Unterprogramm für die Berechnung der Fakultätsfunktion $n! = 1 \cdot 2 \cdot \dots \cdot n$ und geben Sie ein Hauptprogramm an, vom dem das Unterprogramm aufgerufen wird.
- b) Die rekursive Berechnung des Binomialkoeffizienten $\binom{n}{k}$ kann mittels folgender Funktion durchgeführt werden:

```
function binom (n,k : integer) : integer;
begin
    if (k=0 OR k=n) then binom := 1
        else if k=1 then binom := n
            else binom := binom(n-1,k) + binom(n-1,k-1)
    end;
end;
```

TI II

Sommersemester 2010

PD Dr. Katinka Wolter

Schreiben Sie ein vollständiges MMIX Assemblerprogramm, welches mittels eines rekursiven Unterprogramms den Binomialkoeffizient $\binom{n}{k}$ berechnet.

Hinweis: Benutzen Sie PUSHJ und POP.

Solution 6

```
a)      LOC      #100

Main    SET      $0,2          % Hier werden irgendwelche Register
        SET      $1,0          % irgendwie belegt. Der Sinn ist die
        SET      $2,'H'       % Demonstration des Registerstacks:
        SET      $3,#A        % Alle Register unterhalb des bei
        SET      $4,$3        % PUSHJ übergebenen Registers werden
                                % vor dem Unterprogramm verborgen.

        SET      $6,4          % Die Register oberhalb des bei PUSHJ
                                % übergebenen Registers werden vom
                                % Unterprogramm gesehen und können
                                % zur Argumentübergabe verwendet
                                % werden.

        PUSHJ   $5,factorial:begin % Hier wird das Unterprogramm
                                % aufgerufen.

        SET      $0,$5        % Der Hauptrückgabewert steht immer in
                                % dem Register, dass bei PUSHJ übergeben
                                % wurde. In diesem Beispiel steht in Register
                                % $5 der Rückgabewert des Unterprogramms. Wenn
                                % es mehr als einen Rückgabewert gibt, stehen
                                % diese in den darüberliegenden Registern
                                % (hier nicht der Fall).

        TRAP    0,Halt,0

        PREFIX factorial:

n        IS      $0          % Diese Register sind nicht
fac      IS      $0          % dieselben wie im Hauptprogramm.
j_addr  IS      $1          % In jedem rekursiven Aufruf werden
                                % Register verborgen und umbenannt!

fac_1   IS      $2
n_1     IS      $3

begin   BNN     n,rec        % Fehler werden behandelt. Eine negative
                                % Eingabe ist unzulässig.
        SET      fac,0        % Bei einer fehlerhaften Eingabe wird -1
        SUB     fac,fac,1     % zurückgegeben. Nicht ideal, aber besser
        POP     1,0          % als nichts...

rec     BP      n,next        % Das ist der Rekursionsanker.

        SET     fac,1
```

TI II

Sommersemester 2010

PD Dr. Katinka Wolter

```

        POP      1,0                % Es wird nur ein Wert zurückgegeben,
                                     % das Ergebnis in result.

next    GET      j_addr,:rJ        % Da wir rekursiv factorial aufrufen,
                                     % muss jedes mal die aktuelle Rücksprung-
                                     % adresse gespeichert werden.

        SUB      n_1,n,1
        PUSHJ   fac_1,rec          % Hier erfolgt ein rekursiver Aufruf.
                                     % f_1 grenzt die zu schützenden Register
                                     % (n, j_addr) von den Argumenten (n_1) für
                                     % den rekursiven Aufruf ab. Hier wird auch
                                     % das Zwischenergebnis aus dem rekursiven
                                     % Aufruf abgelegt.

        MUL     fac,n,fac_1
        PUT     :rJ,j_addr        % POP springt an die Adresse, die im
        POP     1,0                % Spezialregister rJ steht, daher schreiben
                                     % wir die Rücksprangadresse in dieses
                                     % Register zurück.

b)      LOC      Data_Segment
        GREG    @
n_out   BYTE    "N >= 0: ",0
k_out   BYTE    "K >= 0: ",0
result  BYTE    "result: ",0
errmsg  BYTE    "Es muss gelten N >= K >= 0 !",0

bufSiz  IS      100
buffer  BYTE    0
        LOC     buffer+bufSiz
bufArg  OCTA    buffer,bufSiz

        LOC     #100

n       IS      $0
k       IS      $1

Main    LDA     $255,n_out          % Einlesen von n
        TRAP   0,Fputs,StdOut
        LDA     $255,bufArg
        TRAP   0,Fgets,StdIn
        LDA     $4,buffer          % In $4, $5 Argumente von atoi
        SET    $5,$255
        PUSHJ  $2,atoi:begin
        SET    n,$2

        LDA     $255,k_out        % Einlesen von k
        TRAP   0,Fputs,StdOut
        LDA     $255,bufArg
        TRAP   0,Fgets,StdIn
        LDA     $4,buffer          % In $4, $5 Argumente von atoi
        SET    $5,$255
        PUSHJ  $2,atoi:begin
```

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

```

SET      k,$2

SET      $3,n          % Berechnung des Binomialkoeffizienten
SET      $4,k          % In $3, $4 Argumente von binomial
PUSHJ    $2,binomial:begin
BN       $2,error      % Wenn -1 dann falsche Eingabe!

SET      $5,$2         % In $5,$6 Argumente von itoa
LDA      $6,bufArg
PUSHJ    $3,itoa:begin
LDA      $255,result
TRAP     0,Fputs,StdOut % Ausgabe
LDA      $255,buffer
TRAP     0,Fputs,StdOut
SET      $2,$2
TRAP     0,Halt,0

error    LDA      $255,errmsg      % Fehlermeldung
         TRAP     0,Fputs,StdOut
         TRAP     0,Halt,0

PREFIX   :atoi:          % einfache asciiToInteger-Funktion,
result   IS      $0        % die alle Non-Digits ignoriert,
string   IS      $1        % d.h. nur positive Zahlen werden
strlen   IS      $2        % gelesen.
digit    IS      $3
deci     IS      $4
c        IS      $5

begin    SET      result,0
         SET      deci,1
0H       SUB      strlen,strlen,1
         LDB      digit,string,strlen
         CMP      c,digit,'0'
         BN       c,0B
         CMP      c,digit,'9'
         BP       c,0B
         SUB      digit,digit,'0'
         MUL      digit,digit,deci
         ADD      result,result,digit
         BZ       strlen,end
         MUL      deci,deci,10
         JMP      0B
end      POP      1,0

PREFIX   :itoa:          % Einfache integerToAscii-Funktion,
result   IS      $0        % die nur mit positiven Zahlen arbeiten
zero     IS      $0        % kann.
number   IS      $1
bufArg   IS      $2
buffer   IS      $2

```

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

```

bufSiz IS    $3
length IS    $4
digit  IS    $5
c      IS    $6
ntemp  IS    $6

begin  LDO    bufSiz,bufArg,8
      LDO    buffer,bufArg,0
      SET    length,0
      SET    result,0
      SET    ntemp,number

0H     ADD    length,length,1
      DIV    ntemp,ntemp,10
      PBNZ   ntemp,0B
      CMP    c,length,bufSiz
      BNN    c,end
      STB    zero,buffer,length
      SET    result,1
1H     SUB    length,length,1
      BN     length,end
      DIV    number,number,10
      GET    digit,:rR
      ADD    digit,digit,'0'
      STB    digit,buffer,length
      JMP    1B
end     POP    1,0

      PREFIX :binomial:           % Berechnet den Binomialkoeffizienten

bin     IS    $0                   % Ergebnis von binomial
n       IS    $0
k       IS    $1
j_addr IS    $2
c       IS    $3
bin_1   IS    $3                   % Ergebnis von rek. Aufruf
n_1     IS    $4                   % Argumente fuer Unterprogramm
k_1     IS    $5

begin   BN    n,error              % Eingabefehler werden behandelt.
      BN    k,error              % Bei Fehlern wird -1 zurückgegeben.
      CMP   c,k,n
      BNP   c,rec
error   SET   bin,0
      SUB   bin,bin,1
      POP   1,0

rec     BZ    k,base1              % k == 0
      CMP   c,k,n                  % oder
      BZ    c,base1              % k == n

```

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

```
CMP      c,k,1          % k == 1
BZ       c,baseN

GET      j_addr,:rJ    % Die Rücksprungadresse muss gesichert
SUB      n,n,1         % werden.
SET      n_1,n         % In $4, $5 Argumente für rekursiven
SET      k_1,k         % Aufruf
PUSHJ   bin_1,rec      % rekursiv binom(n-1, k)
SET      n_1,n
SUB      k_1,k,1
SET      bin,bin_1
PUSHJ   bin_1,rec      % rekursiv binom(n-1, k-1)
ADD      bin,bin,bin_1
PUT      :rJ,j_addr    % Die Rücksprungadresse muss wieder
                        % geladen werden.
POP      1,0           % Es wird ein Wert zurückgegeben.

base1   SET      bin,1
baseN   POP      1,0
```