Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

5. Aufgabenblatt mit Lösungsvorschlag

29.11.2010

Hinweis: Der Schnelltest und die Aufgaben sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind in der Kalenderwoche 49 (06.12. bis 10.12.) bei den Tutoren in **physikalischer Form** (handschriftlich oder gedruckt) abzugeben. Bei allen Abgaben ist der Name des Tutors und die Übungsgruppe deutlich anzugeben. Bei Teamabgaben wird nur eine Lösung eingereicht, die alle Namen der Teammitglieder enthält.

Schicken Sie Ihre Lösungen von Programmieraufgaben *zusätzlich* zur schriftlichen Abgabe per E-Mail an Ihren Tutor. Kommentieren Sie Ihren Quellcode.

Aufgabe 1: Schnelltest

Fragen	***Antworten***
1. *Welche Variable ist ein optimaler Indikator für die Performanz eines beliebigen Programms? - An dieser Stelle ist keine der Antworten korrekt.*	☐ *Anzahl der Taktzyklen für Programmausführung.* ☐ *Anzahl der Instruktionen des Programms.* ☐ *Durchschnittliche Anzahl der Taktzyklen pro Instruktion.* ☐ *Durchschnittliche Anzahl der Taktzyklen pro Sekunde.*
2. *Welche Aussagen zu Bewertungskriterien für Prozessoren sind richtig?*	■ *Die CPU-Ausführungszeit ergibt sich aus den verwendeten Operationen und der Taktung der CPU.* ☐ *Die CPU-Ausführungszeit ist die gemessene Zeit, welche die CPU real zum Ausführen der Codezeilen eines Programms benötigt.* ■ *Die CPU-Systemzeit ist die Zeit, welche die CPU für Betriebssystemaufgaben verwendet.* ☐ *Die Benutzerantwortzeit ist ein geeignetes Maß für CPU-Performanz.* ☐ *Rechner mit hohen Durchsatz haben auch kürzere Ausführungszeiten als Rechner mit niedrigem Durchsatz.*
3. *Welche Elemente einer Programmausführung fließen nicht in die user-CPU-time mit ein?*	■ *Festplattenzugriffe* ■ *Verzögerung durch Cache-Misses* ■ *I/O-Verzögerung durch Hardware* ☐ *CPU-Ausführungszeit*
4. *Welche Elemente einer Programmausführung fließen in die Benutzerantwortzeit mit ein?*	■ *I/O-Verzögerung durch Hardware* ■ *Verzögerung durch Cache-Misses* ■ *CPU-Ausführungszeit* ■ *Festplattenzugriffe*

Aufgabe 2: Gleitkommazahlen

Gegeben sei ein Rechner, der mit 8-Bit Gleitkommazahlen arbeitet. Diese 8 Bit teilen sich auf in 1 Bit für das Vorzeichen, 3 Bit für den Exponenten und 4 Bit für die Mantisse:

V	E	E	E	M	M	M	M

$$r = -1^V \cdot (1 + M) \cdot 2^{E-\text{Bias}}$$

a) *Wie groß ist der Bias, der für die Darstellung des Exponenten verwendet wird?*

Lösungsvorschlag:

Allgemein ergibt sich der Bias für einen n-Bit Exponenten als $2^{n-1} - 1$. Da in diesem Beispiel 3 Bit für den Exponenten zur Verfügung stehen, beträgt der Bias also $2^{3-1} - 1 = 3$.

b) *Wandeln Sie die Zahl $6,25$ in die beschriebene Darstellung um.*

Lösungsvorschlag:

$6,25_{10} = 110,01_2 = 1,1001_2 \cdot 2^2$. Somit besteht die Mantisse aus 1001 und der Exponent mit Bias aus $2 + 3 = 5_{10} = 101_2$, das Vorzeichen ist 0. Insgesamt lautet die 8-Bit Gleitkommazahl 0 101 1001.

c) *Wandeln Sie die Zahl $-0,78125$ in die beschriebene Darstellung um.*

Lösungsvorschlag:

$-0,78125_{10} = -1,5625_{10} \cdot 2^{-1} = -1,1001_2 \cdot 2^{-1}$. Für die Mantisse ergibt sich wie zuvor 1001, aber der Exponent mit Bias beträgt $-1+3 = 2_{10} = 010_2$, das Vorzeichen ist 1, da negativ. Damit ergibt sich die Lösung als 1 010 1001.

d) *Multiplizieren Sie die beiden Repräsentation (Darstellung der 8 Bit Gleitkommazahl) von $6,25$ und $-0,78125$. Geben Sie das Ergebnis in Gleitkommadarstellung an und wandeln Sie es in die Dezimalzahldarstellung um. Hinweise zur Multiplikation von Gleitkommazahlen sind durch die Suche in Büchern, Datenbanken (ULB) und dem Internet zu finden.*

Lösungsvorschlag:

Es ergibt sich $V_3 = 1$. Das Produkt aus $M_1 + 1$ und $M_2 + 1$ ist $10,01110001_2$, wovon nur die ersten fünf Stellen betrachtet werden (4 Bits Mantissenbreite + implizite 1). Nach Multiplikation mit 2 und entfernen der impliziten 1 ergibt sich $M_3 = 0,0011_2$ und $E_3 = 5$, als Gleitkommazahl 1 101 0011.

Als Dezimalzahl ist dies nach der Formel $-1^V \cdot (1 + M) \cdot 2^{E-\text{Bias}} = -1^1 \cdot 1,0011_2 \cdot 2^{5-3} = -4,75$.

Aufgabe 3: Gleitkommazahl nach IEEE 754

Gegeben ist eine Gleitpunktzahl G im IEEE 754 Standard (einfache Genauigkeit) in Hexadezimaldarstellung.

$$G = C080\,0000_{16}$$

Welchen dezimalen Wert repäsentiert die die Darstellung von G?

Lösungsvorschlag:

Die Zahl in Binärschreibweise:

$$G = 1100\,0000\,1000\,0000\,0000\,0000\,0000\,0000_2 .$$

Das erste Bit ist gesetzt, daher handelt es sich um eine negative Zahl ($S = 1$). Es gilt $E = 1000\,0001_2 = 129$ und $M = 0,000\,0000\,0000\,0000\,0000\,0000_2 = 0$. Daher gilt: $G = (-1)^S \cdot (1 + M) \cdot 2^{(127-E)} = (-1) \cdot 1,0 \cdot 2^2 = -4,0$

Aufgabe 4: Leistungsbewertung

Folgende Größen sind im Zusammenhang mit der Leistungsbewertung gegeben:

T	*CPU-Zeit in Sekunden*
f	*Taktfrequenz in Zyklen pro Sekunde*
t	*Taktzykluszeit in Sekunden pro Zyklus*
MIPS	*Millionen von Instruktionen pro Sekunde*
CPI	*mittlere Anzahl Zyklen pro Instruktion*
B	*Anzahl Instruktionen*
Z	*CPU-Zeit in Zyklen*

a) *Leiten Sie die Formeln der gesuchten Größen aus den angegebenen Größen her.*

1. *gesucht: T, gegeben: Z, t*
2. *gesucht: CPI, gegeben: Z, MIPS, T*
3. *gesucht: T, gegeben: f, CPI, B*
4. *gesucht: B, gegeben: T, MIPS*
5. *gesucht: Z, gegeben: T, CPI, MIPS*

Lösungsvorschlag:

- gesucht: T gegeben: Z, t $\Rightarrow$ T = Z * t
- gesucht: CPI, gegeben: Z, MIPS, T $\Rightarrow$ CPI = Z / (T * MIPS * 10^6)
- gesucht: T, gegeben: f, CPI, B $\Rightarrow$ T = B * CPI / f
- gesucht: B, gegeben: T, MIPS $\Rightarrow$ B = T * MIPS * 10^6
- gesucht: Z, gegeben: T, CPI, MIPS $\Rightarrow$ Z = T *CPI * MIPS * 10^6

b) *Zwei unterschiedliche Prozessoren mit jeweils 2.4 GHz und 2.233 GHz Taktfrequenz bearbeiten eine Berechnung, die insgesamt aus $9 \cdot 10^7$ Instruktionen besteht. Der erste Rechner benötigt 14 Zyklen pro Instruktion, der zweite Rechner 12 Zyklen pro Instruktion. Welcher Rechner ist schneller fertig?*

Lösungsvorschlag:

$T = B \cdot \text{CPI}/f$. Also $T_1 = \frac{9 \cdot 10^7 \cdot 14}{2.4 \cdot 10^9 Hz} \approx 525$ ms und $T_2 = \frac{9 \cdot 10^7 \cdot 12}{2.233 \cdot 10^9 Hz} \approx 484$ ms.

c) *Die Operationen einer komplexen Berechnung benötigen zusammengenommen $7.7 \cdot 10^7$ Rechenzyklen. Wie lange dauert die Ausführung auf einem Prozessor mit 3.2 GHz Taktfrequenz?*

Lösungsvorschlag:

$T = Z/f = \frac{7.7 \cdot 10^7}{3.2 \cdot 10^9 Hz} \approx 24$ ms

d) *Ein Rechner, der eine Leistung von 2.3 MIPS hat, benötigt für eine komplexe Berechnung 1350 ms. Wie viele Instruktionen werden dabei verarbeitet?*

Lösungsvorschlag:

$B = T \cdot \text{MIPS} \cdot 10^6 = 1.35\text{s} \cdot 2.3/\text{s} \cdot 10^6 = 3\,105\,000$ Instruktionen.

e) *Obiger Rechner hat eine Taktfrequenz von 750 MHz. Wie hoch ist die mittlere Anzahl an Rechenzyklen pro Instruktion?*

Lösungsvorschlag:

$T = B \cdot \text{CPI}/f \rightarrow \text{CPI} = T \cdot f\,/B = \frac{1.35s \cdot 7{,}5 \cdot 10^8 Hz}{3105000} \approx 326$

Aufgabe 5: Performanzvergleich

Betrachtet werden zwei Hardware-Implementierungen M1 und M2 eines Befehlssatzes mit vier Befehlsklassen Add, Jmp, Mul und Shft. Die Taktfrequenz für M1 ist 1.133 *GHz und für M2* 750 *MHz. Desweiteren sind die Daten aus den folgenden Tabelle gegeben.*

Befehlsklasse	*CPI für M1*	*CPI für M2*
Add	*2*	*1*
Jmp	*2*	*2*
Mul	*5*	*3*
Shft	*1*	*1*

Befehlsklasse	*Häufigkeit*
Add	*45%*
Jmp	*15%*
Mul	*25%*
Shft	*15%*

a) *Ein Programm P1 weist die in der Tabelle angebene Verteilung der Befehlshäufigkeiten auf. Wie groß ist die CPI-Rate von M1 und M2 bei der Ausführung von P1?*

Lösungsvorschlag:

CPI für M1 bei Ausführung von P1 $= \frac{45}{100} \cdot 2 + \frac{15}{100} \cdot 2 + \frac{25}{100} \cdot 5 + \frac{15}{100} \cdot 1 = 2{,}6$
CPI für M2 bei Ausführung von P1 $= \frac{45}{100} \cdot 1 + \frac{15}{100} \cdot 2 + \frac{25}{100} \cdot 3 + \frac{15}{100} \cdot 1 = 1{,}65$

b) *Wie ist die relative Performanz zwischen M1 und M2 bei Ausführung von P1?*

Lösungsvorschlag:

n sei die Anzahl der Befehle von P1.
CPU-Zeit für M1 zur Ausführung von P1 $= \frac{n \cdot 2{,}6}{1{,}133 \cdot 10^9} s \approx n \cdot 2{,}29 \cdot 10^{-9} s$
CPU-Zeit für M1 zur Ausführung von P1 $= \frac{n \cdot 1{,}65}{7{,}5 \cdot 10^8} s \approx n \cdot 2{,}2 \cdot 10^{-9} s$
M1 ist um 4.09% langsamer als M2.

c) *Das Programm P2 enthält jeweils gleich viele Befehle aus den vier Befehlsklassen. Wie hoch ist die mittlere Anzahl der Zyklen pro Befehl (CPI) bei Ausführung von P2 auf M1 und M2?*

Lösungsvorschlag:

CPI für M1 bei Ausführung von P2 $= \frac{2+2+5+1}{4} = 2{,}5$

CPI für M2 bei Ausführung von P2 $= \frac{1+2+3+1}{4} = 1{,}75$

d) *Wie ist die relative Performanz zwischen M1 und M2 bei Ausführung von P2?*

Lösungsvorschlag:

n sei die Anzahl der Befehle von P2.

CPU-Zeit für M1 zur Ausführung von P2 $= \frac{n \cdot 2{,}5}{1{,}133 \cdot 10^9} s \approx n \cdot 2{,}21 \cdot 10^{-9} s$

CPU-Zeit für M2 zur Ausführung von P2 $= \frac{n \cdot 1{,}75}{7{,}5 \cdot 10^8} s \approx n \cdot 2{,}33 \cdot 10^{-9} s$

M1 ist um 5.15% schneller als M2.

Hausaufgabe 1: Nachbildung einer FPU mit Integerarithmetik (6 Punkte)

*In dieser Aufgaben sollen Sie die bisher nativ genutzte FPU (Floating Point Unit) softwaretechnisch mittels IA32-Assemblercode emulieren. Das bedeutet, dass Sie die Grundrechenarten +, * für IEEE 754 single precision in IA32-Assemblercode implementieren sollen. Die Spezialfälle (vgl. Vorlesung 11, Folie 42) müssen nicht berücksichtigt zu werden.*

Es steht Ihnen für diese Aufgabe ein Rahmencode zur Verfügung. **Nutzen Sie diesen Rahmencode.** *Er enthält eine Ausgabe für die Gleitkommazahlen und alle Unterprogramme, die Sie schreiben sollen. Natürlich dürfen für diese Aufgabe keine FPU-Befehle genutzt werden. Weiterhin sollten Sie darauf achten die vorgegebenen Konventionen der Unterprogrammtechnik einzuhalten. Kommentieren Sie ihren Code ausführlich!*

a) *Vervollständigen Sie zuerst die Unterprogramme* `sign`, `exponent`, `significand`, *um die* `fltprint`*-Methode testen zu können. Die Unterprogramme sollen folgende Funktionen übernehmen:*

- *sign: Vorzeichen der Gleitkommazahl extrahieren*
- *exponent: Exponenet der Gleitkommazahl extrahieren*
- *significand: Mantisse der Gleitkommazahl extrahieren*

Danach sollten Sie in der Lage sein die `fltprint`*-Methode zu nutzen und ein float auf der Konsole auszugeben.*

b) *Implementieren Sie nun die Unterprogramme für Addition und Multiplikation. Nutzen Sie an dieser Stelle falls notwendig die zur Verfügung stehenden Unterprogramme* `sign`, `exponent`, `significand`.

Lösungsvorschlag:

```
.data
fltout: .string "Wert: %f\n"
a:      .float 2.76
b:      .float -3.12
c:      .float 1.2

.text
.globl main
main:
    # a * b
    movl a, %eax
    movl b, %ebx
    call fltmul
```

```
    call fltprint

    # b + c
    movl b, %eax
    movl c, %ebx
    call fltadd
    call fltprint

    # c + a
    movl c, %eax
    movl a, %ebx
    call fltadd
    call fltprint

    # Exit
    movl $1, %eax
    int $0x80

/*
    Addition zweier float Werte
    Eingabe: %eax, %ebx
    Ausgabe: %eax

    Notizen: %ecx enthaelt dauerhaft den 1. float-Wert (bis auf Ende)
             %ebx enthaelt dauerhaft den 2. float-Wert
             %esi verwendet um 1. Extraktion zu speichern
             %edi Zwischenergebniss der Addition
             %edx enthaelt Differenz der Exponenten
*/
fltadd:
    pushl %ebx
    pushl %ecx
    pushl %edx
    pushl %esi
    pushl %edi

    movl %eax, %ecx         # Sicherung des 1. floats
    call exponent           # Exponenten extrahieren
    movl %eax, %esi
    movl %ebx, %eax
    call exponent
    movl %eax, %edi
    cmpl %edi, %esi         # Beide Exponenten vergleichen
    jge  fasizeok           # 1. float sollte groesser sein
    xchgl %ecx, %ebx        # falls nicht, vertauschen
    xchgl %edi, %esi
fasizeok:
    movl %esi, %edx
    subl %edi, %edx         # Differenz der Exponenten merken
    movl %ecx, %eax         # Vorzeichen extrahieren
    call sign
    movl %eax, %edi         # Resultierendes Vorzeichen ist vom 1. float
    movl %ebx, %eax
    call sign
    xorl %edi, %eax         # Unterschied der Vorzeichen
    pushl %eax              # Auf den Stack legen fuer spaeter
    shll $8, %edi           # Exponent von frueher an Ergebnis haengen
    orl  %esi, %edi
```

```
    movl %ecx, %eax         # Mantissen extrahieren
    call significand
    orl  $0x800000, %eax    # und beide um implizite 1, erweitern
    movl %eax, %esi
    movl %ebx, %eax
    call significand
    orl  $0x800000, %eax
    movl %edx, %ecx
    shrl %cl, %eax          # 2. float anpassen fuer Addition
    popl %ecx               # Unterschied der Vorzeichen wiederholen
    jecxz fasame            # Falls gleich = Addition, ansonsten Subtraktion
    subl %eax, %esi         # Subtraktion
    movl %esi, %eax
fasubnorm:
    cmpl $0x800000, %eax    # Bei Subtraktion mehrmaliges
    jge  faend              #   Normalisieren moeglich
    shll $1, %eax
    decl %edi
    jmp  fasubnorm
fasame:
    addl %esi, %eax         # Addition
    cmpl $0x1000000, %eax   # Einfaches Normalisieren falls noetig
    jl   faend
    shrl $1, %eax
    incl %edi

faend:
    shll $23, %edi          # Mantisse an Ergebnis anhaengen
    andl $0x7FFFFF, %eax    # Vorher aber die implizite 1, entfernen
    orl  %edi, %eax

    popl %edi
    popl %esi
    popl %edx
    popl %ecx
    popl %ebx
    ret

/*
    Multiplikation zweier float Werte
    Eingabe: %eax, %ebx
    Ausgabe: %eax

    Notizen: %ecx enthaelt dauerhaft den 1. float-Wert
             %ebx enthaelt dauerhaft den 2. float-Wert
             %esi verwendet um 1. Extraktion zu speichern
             %edi Zwischenergebniss der Multiplikation
*/
fltmul:
    pushl %ebx
    pushl %ecx
    pushl %edx
    pushl %esi
    pushl %edi

    movl %eax, %ecx         # Sicherung des 1. floats
    call sign               # Vorzeichen extrahieren
```

```
    movl %eax, %esi
    movl %ebx, %eax
    call sign
    xorl %esi, %eax         # Resultierendes Vorzeichen errechnen
    movl %eax, %edi         # In Ergebnis uebernehmen

    movl %ecx, %eax         # Exponenten extrahieren
    call exponent
    movl %eax, %esi
    movl %ebx, %eax
    call exponent
    addl %esi, %eax         # Exponenten addieren
    subl $127, %eax         # Ein Bias ist zuviel daher subtrahieren
    shll $8, %edi           # An Ergebnis anhaengen
    orl  %eax, %edi

    movl %ecx, %eax         # Mantissen extrahieren
    call significand
    orl  $0x800000, %eax    # und beide um implizite 1, erweitern
    movl %eax, %esi
    movl %ebx, %eax
    call significand
    orl  $0x800000, %eax
    mull %esi               # Multiplikation der Mantissen
    shrl $23, %eax          # 23 Stellen sind zuviel (Rundung)
    shll $9, %edx           #   und zusaetzlich haengt ein Teil in %edx
    orl  %edx, %eax         #   nun steht alles in %eax

    cmpl $0x1000000, %eax   # Mantisse normalisieren
    jl   fmnorm
    shrl $1, %eax
    incl %edi
fmnorm:
    shll $23, %edi          # An Ergebnis anhaengen
    andl $0x7FFFFF, %eax    # Vorher aber die implizite 1, entfernen
    orl  %edi, %eax

    popl %edi
    popl %esi
    popl %edx
    popl %ecx
    popl %ebx
    ret

/*
    Ausgabe einer float Zahl via printf
    Wichtig: printf ist leider nur double kompatibel,
        daher muss float auf double erweitert werden
        auch wenn %f genutzt wird!

    Eingabe: %eax
    Ausgabe: ---
*/
fltprint:
    pusha

    call extend             # Wegen printf auf double erweitern
    pushl %edx
```

```
    pushl %eax
    pushl $fltout
    call printf
    addl $12, %esp

    popa
    ret

/*
    Erweitert ein float in double
    Eingabe: %eax
    Ausgabe: %edx:%eax

    Notizen: %ebx enthaelt dauerhaft den float-Wert
             %ecx:%edx Zwischenergebniss der Erweiterung
             %edx enthaelt Differenz der Exponenten
*/
extend:
    pushl %ebx
    pushl %ecx

    movl %eax, %ebx         # Sicherung des floats
    call sign               # Vorzeichen extrahieren
    movl %eax, %ecx         # ecx:edx wird double enthalten
    movl %ebx, %eax
    call exponent           # Exponent extrahieren
    addl $896, %eax         # Exponent erweitern -127 +1023
    shll $11, %ecx
    orl  %eax, %ecx         # An ecx anhaengen
    movl %ebx, %eax
    call significand        # Mantisse extrahieren
    movl %eax, %edx
    shrl $3, %eax           # Nur 20 Bit passen noch dran
    shll $20, %ecx
    orl  %eax, %ecx
    shll $29, %edx          # Mantisse erweitern
    movl %edx, %eax         # Ausgabe setzen
    movl %ecx, %edx

    popl %ecx
    popl %ebx
    ret

/*
    Vorzeichen des floats extrahieren
    Ein-/Ausgabe: %eax
*/
sign:
    andl $0x80000000, %eax
    rol $1, %eax
    ret

/*
    Exponent des floats extrahieren
    Ein-/Ausgabe: %eax
*/
exponent:
    andl $0x7F800000, %eax
```

```
    shll $1, %eax
    rol $8, %eax
    ret

/*
    Mantisse des floats extrahieren
    Ein-/Ausgabe: %eax
*/
significand:
    andl $0x7FFFFF, %eax
    ret
```

Hausaufgabe 2: Arithmetischer Ausdruck mit Gleitkommazahlen (4 Punkte)

Gegeben ist der folgende arithmetische Ausdruck:

((a - x) + b) / ((c - d) * (e * f - (g / h + i)))

a) *Stellen Sie diesen Ausdruck als binären Baum dar. Ergänzen Sie zunächst im Ausdruck Klammern, die einzelne Terme zusammenfassen. Beachten Sie dabei die Berechnungsfolge.* **Hinweis:** *Operatoren gleicher Priorität sollen von links nach rechts ausgewertet werden.*

b) *Geben Sie die LR-Postorder-Reihenfolge der Knoten an.*

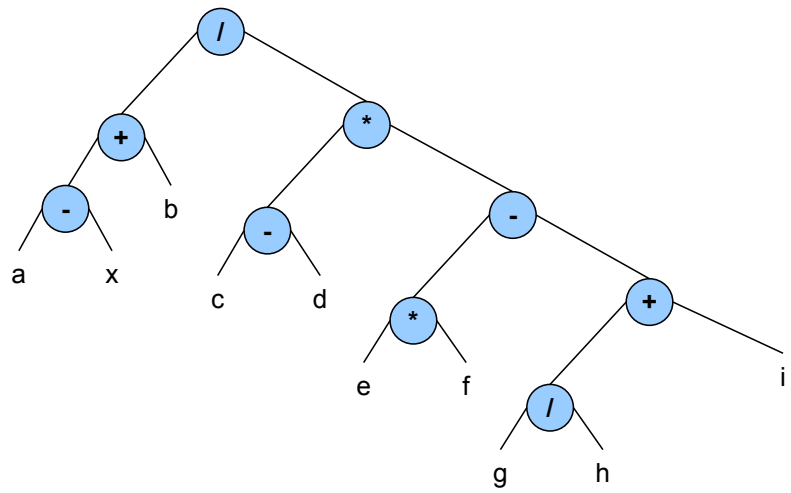
c) *Schreiben Sie ein IA32-Assemblerprogramm (ATT- oder Intel-Syntax) welches die LR-Postorder-Darstellung in Maschinenbefehle abbildet. Dabei sollen 32-Bit Gleitkommazahlen und die x87 FPU verwendet werden. Halten Sie Zwischenergebnisse ausschließlich auf dem FPU-Stack. Fehlersituationen wie Überlauf und Division durch Null sollen* ***nicht*** *berücksichtigt werden. Das Ergebnis soll am Ende in* **st(0)** *stehen und ausgegeben werden. Es dürfen keine General Purpose Register (%eax, usw.) verwendet werden (außer für die Ausgabe). Die Variablen dürfen nicht verändert werden und es dürfen, außer den Eingabewerte a, b, c, d, e, f, g, h, i, x, keine zusätzlichen Variablen im* **.DATA***-Bereich vereinbart werden. Testen Sie Ihr Programm mit folgenden Werten.*

a = 2.3, b = 0.7, c = 4.7, d = 1.7, e = 3.0, f = 2.0, g = 9.0, h = 3.0, i = 1.0, x = 1.0

Hinweis*: Eine ausführliche Beschreibung der x87 FPU finden Sie unter* `https://ics.ra.informatik.tu-darmstadt.de/svn/Material/Referenz/Intel/253665.pdf`*, Abschnitt 8-1.*

Lösungsvorschlag:

Baum:



LR-Postorder-Reihenfolge: **a x - b + c d - e f * g h / i + - * /**

IA32-Assemblerprogramm:

```
.data
intout:  .string "Ergebnis␣%f\n"
 a:      .float 2.3
 b:      .float 0.7
 c:      .float 4.7
 d:      .float 1.7
 e:      .float 3.0
 f:      .float 2.0
 g:      .float 9.0
 h:      .float 3.0
 i:      .float 1.0
 x:      .float 1.0

.text
.globl main
main:
         fld  a
         fsub x # a - x
         fadd b # (a - x) + b
         fld  c # Ergebnis auf Stack und c laden
         fsub d # c - d
         fld e  # Ergebnis auf Stack und e laden
         fmul f # e * f
         fld g  # Ergebnis auf Stack und g laden
         fdiv h # g / h
         fadd i # g / h + i
         fxch %st(1),%st(0) # Operanden tauschen
         fsubp %st(1) # -
         fmulp %st(1) # *
         fxch %st(1),%st(0) # Operanden tauschen
         fdiv %st(1) # /
         fstpl (%esp)
         pushl $intout
         call printf
# Exit
movl $1, %eax
int $0x80
```