10.5/20

# Betriebssysteme

**Wintersemester 2022/23**
**Übungsblatt 05**

John Zeeh (4725202) Daniel Burkhardt (4139647)

5.5/6

## Aufgabe 1

### a)

SPckenpre $= NB \cdot (i_{24}\overline{i_{23}i_{22}} \cdot (\overline{i_{31}i_{30}} + \overline{i_{31}}i_{30} + i_{31}\overline{i_{30}}i_{29}i_{28}) + \overline{h_2}h_1\overline{h_0} + h_2h_1h_0) \cdot s_0\overline{s}_1 \cdot E$

-0.5

### b)

IVNckenpre $= NB \cdot (i_{31}i_{30}\overline{i_{26}}i_25 + \overline{h_2h_1}h_0) \cdot s_0\overline{s_1} \cdot E$

Folgefehler...

Normalbetrieb ist nur bei den normalen Befehl notwendig

## Aufgabe 2

5/8

### a)

Erstellen Sie die Einträge der Symboltabelle für den Deklarationsteil, nehmen Sie dabei an, dass *bds*= 128 (dezimal).

**Algorithm 1:** Deklarationsteil

```
1 st(x) = (var, int, 128);
2 st(y) = (var, int, 129);
3 st(z) = (const, int, 5);
```

### b)

Geben Sie die Befehlsfolge der erweiterten ReTI an, die den im obigen Programm dargestellten vollständig geklammerten Ausdruck auswertet und das Ergebnis oben auf den Stack schreibt. Stellen Sie außerdem nach jedem Teilergebnis den aktuellen Stand des Stacks graphisch dar.

-1 kein Stack

Wenn euch RETI direkt schreiben noch etwas schwär fällt, könnt ihr auch einfach die Pattern aus der Vorelsung verwenden



das Schreiben der Variablen auf den Stack gehört zu Aufgabe dazu

-0.5

in Scholls PicoC existieren Konstanten nur intern in der Symboltabelle und werden passend in die RETI-Befehle eingesetzt

in normalen C ja, aber in Scholls PicoC nein

```
1 code^aa(x)
2 code^aa(y)
3 code^aa(z);  // Ich ging jetzt mal davon aus, dass wir sie davor auf den Speicher
    schreiben?
4 LOADIN SP IN1 2 ;                    // Die Adresse von y in IN1
5 LOADIN SP ACC 0 ;                    // Wert von y in ACC
6 LOADIN SP IN1 1 ;                    // Wert von z in IN1
7 ADDI SP 2 ;    // Da y und z nicht mehr gebraucht werden, können wir sie wieder
    aus dem Speicher löschen
8 MUL ACC IN1 ;                  // Da z eine Konstante ist, können wir sie direkt
    multiplizieren. y * z im Akkumulator
9 ADDI ACC 10 ;                                  // y * z + 10 im Akkumulator
10 LOADIN SP IN1 1 ;                          // Die Adresse von x in IN1 speichern
11 ADDI SP 1 ;    // Da x nicht mehr gebraucht wird, können wir es wieder aus dem
    Speicher löschen
12 LOADIN IN2 IN1 ;    // Der Wert der Adresse, die in IN1 steht, wird in den IN2
    geladen
13 ADD ACC IN2 ;                          // x auf den Akkumulator draufaddieren
14 STOREIN SP ACC 0 ;    // Den Akkumulator an die Stelle speichern, wo x früher
    stand, also in den Inhalt von IN1
```

-0.5 in der Speicherzelle M[SP + 2] sollte nach codeaa bereits der richtige Wert stehen

da ist nichts mit Adressen. Die Symboltabelle gibts nur im Compiler. Hier in dieser Aufgabe seid ihr der Compiler

da ihr das Eregbnis nicht wieder auf den Stack speichert, könntest ihr es auch gleich lassen auf diesem immer wieder Platz zu machen, sondern nur einmal

-0.5 das wäre einmal zu viel

da ist nichts mit Adressen. Folgefehler

-0.5

## c)

Angenommen Sie haben n Variablen $x_1, ..., x_n$, die durch (beliebige) binäre Operanden zu einem vollständig geklammerten Ausdruck verknüpft werden, sodass jede Variable exakt einmal vorkommt

1. Wie sieht der vollständig geklammerte Ausdruck aus, der die maximale Anzahl an Teilergebnissen auf dem Stack erfordert? Wie viele Teilergebnisse sind das?

   Wir nehmen + als unseren binären Operanden.
   Sind die Variablen selbst Teilergebnisse? Falls ja, ist der längste Ausdruck:

   $$(x_1 + (x_2 + \cdots + (x_{n-1} + x_n \underbrace{)\ldots)}_{n-1 \text{ times}}),$$

   was in n Teilergebnissen resultieren würde. Ansonsten $(x_1 + x_2) + \cdots + (x_{n-1} + x_n)$ bzw. $x_1 + (x_2 + x_3) + \cdots + (x_{n-1} + x_n)$ für $n$ ungerade, was in $\lfloor n/2 \rfloor$ Teilergebnissen resultiert.

   etwas verwirrend ausgedrückt in der Aufgabenstellung, es war gemeint vieiel auf dem Stack steht

2. Wie sieht der vollständig geklammerte Ausdruck aus, der die minimale Anzahl an Teilergebnissen auf dem Stack erfordert? Wie viele Teilergebnisse sind das?

   Wir nehmen wieder + als unseren binären Operanden.
   Der kürzeste Ausdruck lautet

   $$\underbrace{(\ldots(}_{n-1 \text{ times}} x_1 + x_2) + \ldots x_{n-1}) + x_n),$$

   was in genau zwei Zwischenergebnis resultiert.

## Aufgabe 3

Die Auswertung von Vergleichsoperatoren aus Pico-C erfolgt auf der RETI, indem die Differenz der linken und rechten Seite gebildet wird und diese (je nach Vergleichsoperator) mit 0 verglichen wird. Dieses Vorgehen kann jedoch zu Überläufen führen, wenn die Ausdrücke auf beiden Seiten ein unterschiedliches Vorzeichen haben.
Geben Sie zur Behandlung dieses Problems expemplarisch ein RETI-Programm an, das den logischen Ausdruck $x \leq y <$ korrekt zu 0 bzw. 1 auswertet, auch wenn bei der Berechnung von $x - y$ ein Überlauf auftritt.
Nehmen Sie dazu an, dass die Integer-Variable $x$ an der Speicherstelle 10 und die Integer-Variable $y$ an der Speicherstelle 11 abgespeichert ist.