

17.5/20

Betriebssysteme

Übungsblatt 5

Micha Erkel

Felix Ruh

5.5/6

die Zustände der Kontrollogik und die Befehle des Normlabetriebs sollen nicht gleichzeitig auftreten

Aufgabe 1

-0.5

a) Für $SPcken_{pre}$ muss gelten:

$$([E \cdot \overline{s_1} \cdot s_0] \cdot [I_{31} \cdot I_{30} \cdot I_{24} \cdot \overline{I_{23}} \cdot \overline{I_{22}} + I_{31} \cdot \overline{I_{30}} \cdot I_{29} \cdot I_{28} \cdot I_{24} \cdot \overline{I_{23}} \cdot \overline{I_{22}} + \overline{I_{31}} \cdot \overline{I_{30}} \cdot I_{24} \cdot \overline{I_{23}} \cdot \overline{I_{22}}] \cdot [NB] + \overline{h_2} \cdot h_1 \cdot \overline{h_0} + h_2 \cdot h_1 \cdot h_0)$$

b) Für $IVNcken_{pre}$ muss gelten:

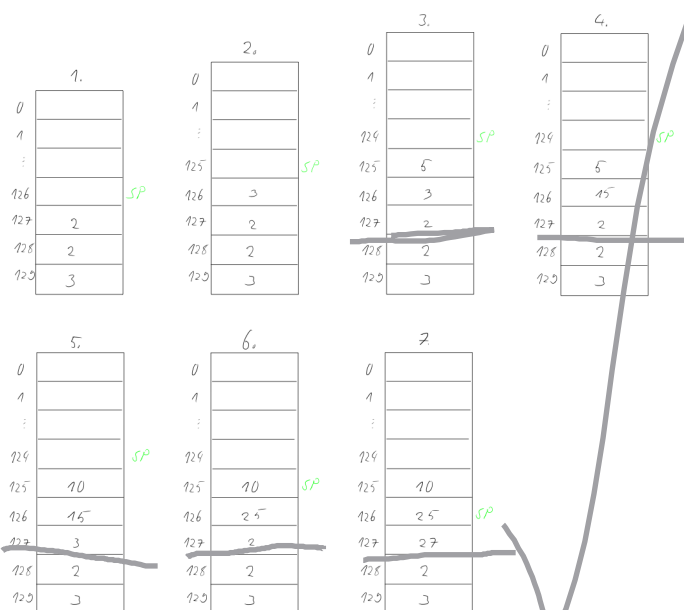
$$([E \cdot \overline{s_1} \cdot s_0] \cdot [I_{31} \cdot I_{30} \cdot \overline{I_{26}} \cdot I_{25} \cdot NB + \overline{h_2} \cdot \overline{h_1} \cdot h_0])$$

8/8

Aufgabe 2

- a) $st(x) = (var, int, 128)$
 $st(y) = (var, int, 129)$
 $st(z) = (const, int, 5)$

b) Die Schritte auf dem Stack dargestellt:



Der Code:

SUBI SP 1	Setzte den SP auf die nächste freie Zelle.
LOAD ACC 128	Lade x in den ACC.
STOREIN SP ACC 1	Speichere den ACC in der letzten freien Zelle.
SUBI SP 1	Setzte den SP auf die nächste freie Zelle.
LOAD ACC 129	Lade y in den ACC.
STOREIN SP ACC 1	Speichere den ACC in der letzten freien Zelle.
SUBI SP 1	Setzte den SP auf die nächste freie Zelle.
LOADI ACC encode(5)	Lade 5 in den ACC.
STOREIN SP ACC 1	Speichere den ACC in der letzten freien Zelle.
LOADIN SP ACC 2	Lade y in den ACC.
LOADIN SP IN2 1	Lade z das IN2
MUL ACC IN2	Multipliziere y mit z.
STOREIN SP ACC 2	Speichere das Ergebnis in der Speicherzelle von y.
ADDI SP 1	
SUBI SP 1	
LOADI ACC encode(10)	Lade 10 in den ACC.
STOREIN SP ACC 1	Speichere den ACC in der letzten freien Zelle.
LOADIN SP ACC 2	Lade y' (das Ergebnis der Multiplikation) in den ACC.
LOADIN SP IN2 1	Lade 10 in das IN2.
ADD ACC IN2	Addieren y' und 10
STOREIN SP ACC 2	Speichere das Ergebnis in der Speicherzelle von y'.
ADDI SP 1	Senke den SP um 1.
LOADIN SP ACC 2	Lade x in den ACC.
LOADIN SP IN2 1	Lade y' (das Ergebnis der beiden vorherigen Operationen) in das IN2.
ADD ACC IN2	Addiere y' und IN2
STOREIN SP ACC 2	Speichere das Ergebnis in der Zelle von x
ADDI SP 1	Senke den SP um 1.

ADDI SP 1

SUBI SP 1

aber korrekt, da es sich ausgleicht XD

schreibt einfach 10, encode ist für den Compiler

hier seid ihr der Compiler

- c) 1. $(x_1 \text{ op } (x_2 \text{ op } (x_3 \text{ op } (\dots (x_{n-1} \text{ op } x_n) \dots))))$
→ So müssten n Teilergebnisse auf den Stack geschrieben werden.
2. $((\dots(((x_1 \text{ op } x_2) \text{ op } x_3) \text{ op } x_4) \text{ op } \dots) \text{ op } x_n)$
→ Auf diese Weise sind immer nur 2 Teilergebnisse auf dem Stack.

Aufgabe 3

Unter der Annahme, dass es sich bei den Zahlen um das 2er Komplement handelt vergleicht dieser Code die beiden Zahlen:

LOADI ACC 1	Lade 00...01 in den ACC.
MULI ACC 2 ¹⁶	Verschiebe die 1 um 16 Stellen, auf Position 16, nach links.
MULI ACC 2 ¹⁵	Verschiebe die 1 um 15 Stellen, auf Position 31, nach links.
STORE ACC 12	Speichere 2 ³¹ in Zelle 12.
LOAD ACC 10	Lade x in das ACC.
LOAD IN1 11	Lade y in das IN1.
AND ACC 12	Vergleich von x mit 1000... - falls x negativ ist, wird erste Stelle 1 sein.
AND IN1 12	Vergleich von y mit 1000... - falls y negativ ist, wird erste Stelle 1 sein.
SUB ACC IN1	Subtrahiere die Ergebnisse von AND von einander.
<i>JUMP</i> ₌ 6	Springe falls beide Vorzeichen gleich sind.
<i>JUMP</i> _{<} 3	Springe falls y negativ und x positiv ist.
gebe 1 aus	Falls y positiv und x negativ ist.
JUMP 10	Beende das Programm.
gebe 0 aus	
JUMP 8	Beende das Programm.
LOAD ACC 10	Lade x in das ACC.
LOAD IN1 11	Lade y in das IN1.
SUB ACC IN1	Subtrahiere y von x.
<i>JUMP</i> _{>} 3	Jump falls x größer ist wie y.
gebe 1 aus	
JUMP 2	Beende das Programm.
gebe 0 aus	
JUMP 0	Beendet das Programm.

Ich hab den RETI-Code aller Studenten die Aufgabe 3 bearbeitet haben (Aufgabe 3 war diesmal die beliebteste Aufgabe) mithilfe des im PicoC-Compilers <https://github.com/matthejue/PicoC-Compiler/releases> eingebauten RETI-Interpreters ausgeführt, genauer mittels des Befehls `picoc_compiler -b -p c.reti -S -P 2 -D 15`. Ich habe versucht den Code von euch Studenten lauffähig zu machen, sodass dieser die Aufgabenstellung erfüllt.
 Die Datei <fruit>.in enthält Eingaben für CALL INPUT REG. Die Datei <fruit>.out enthält die Ausgaben der CALL PRINT REG bei der Ausführung. Die Datei <fruit>.out_expected enthält die erwarteten Ausgaben.
 Eure Korrektur ist unter https://github.com/matthejue/Abgaben_Blatt_3/tree/main/Blatt5/apfelsine.reti zu finden.