15/20

5.5/6

SP als Destination soll ja bei allen Operationen möglich sein

1. $SP_{ckenpre} = NB \cdot$ Normalbetrieb

$(E \cdot s_0 \cdot \overline{s_1}) \cdot$ P_1 von execute

$(\overline{I_{31}} \cdot \overline{I_{30}} \cdot \overline{I_{27}} \cdot \overline{I_{26}} \cdot \overline{I_{25}} +$ Inkrementieren bei Compute

warum das Funktionsfeld?

$\overline{I_{31}} \cdot \overline{I_{30}} \cdot \overline{I_{27}} \cdot \overline{I_{26}} \cdot I_{25} +$ Dekrementieren bei Compute

meint ihr Inkrementieren und Dekrementieren wegen hs2 und hs7? Die müssen ja sowieso schon abgedeckt sein, weil ihr alle Compute Befehl mit Desination SP erlauben müsst

$I_{31} \cdot \overline{I_{30}} \cdot I_{29} \cdot I_{28} +$ Move

$\overline{I_{31}} \cdot I_{30}) \cdot$ -0.5 Load

$I_{24} \cdot \overline{I_{23}} \cdot \overline{I_{22}} +$ SP als Destination

$(h_0 \cdot h_1 \cdot h_2 + \overline{h_0} \cdot h_1 \cdot \overline{h_2})$ h_{s2} oder h_{s7} bei Interrupt

$IVN_{ckenpre} = (E \cdot s_0 \cdot \overline{s_1}) \cdot$ P_1 von execute

$(NB \cdot I_{31} \cdot I_{30} \cdot \overline{I_{26}} \cdot I_{25} +$ Syscall im Normalbetrieb

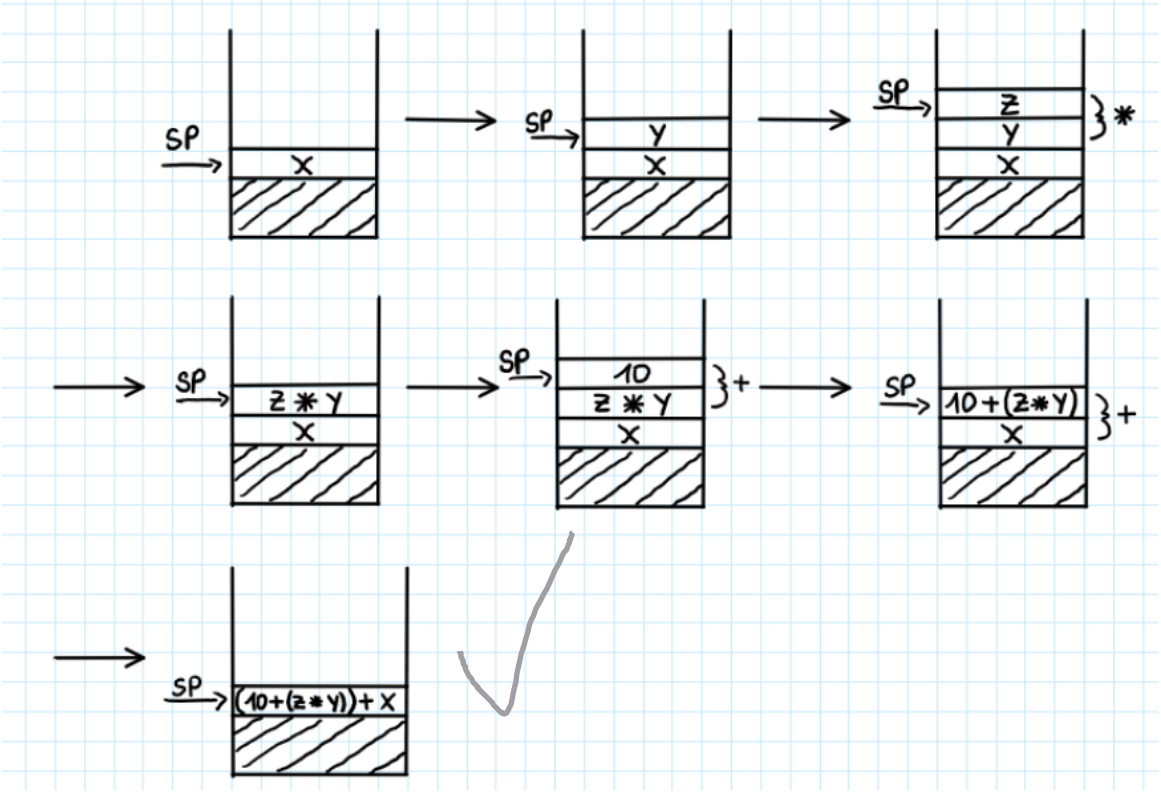
$h_0 \overline{h_1} \overline{h_2})$ h_{s1} bei Interrupt

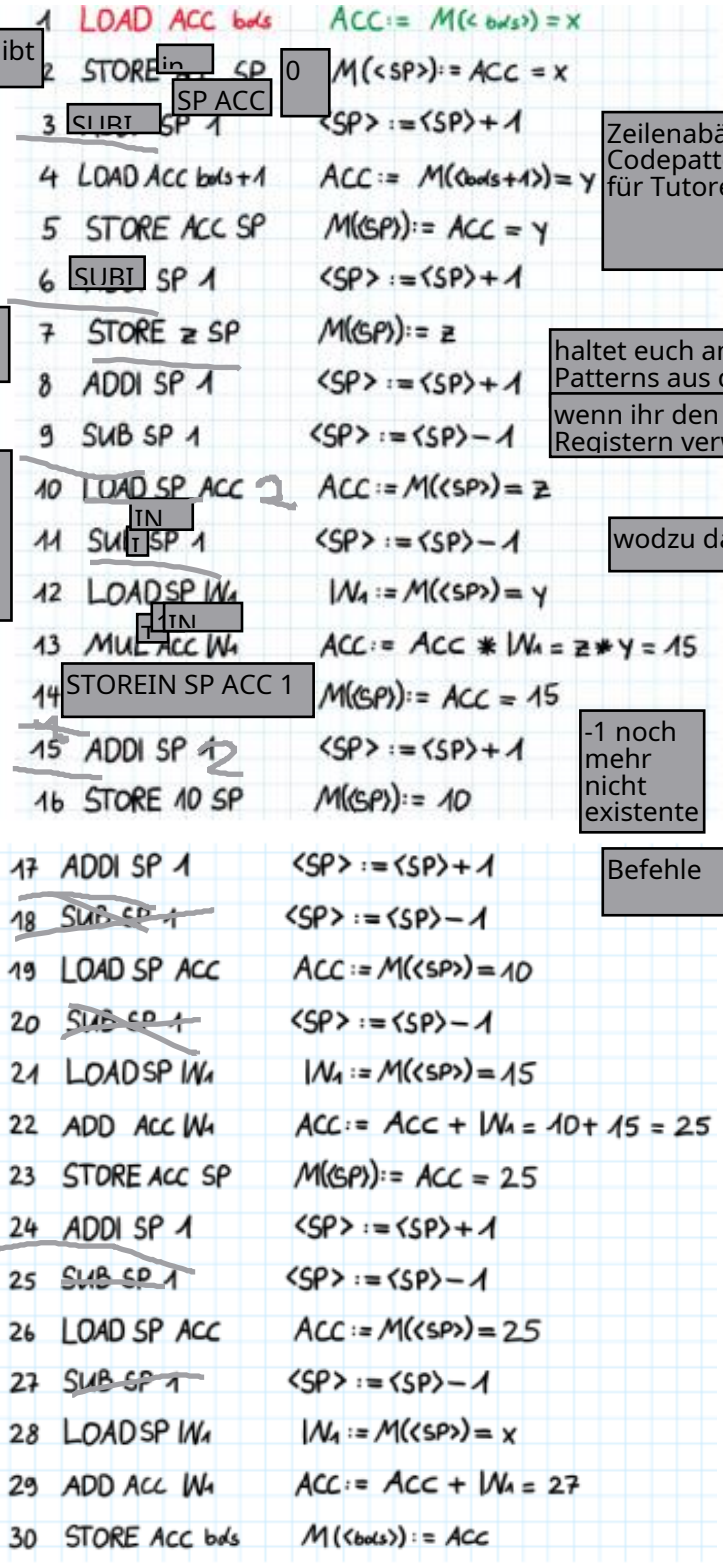
üblicherweise schreibt man es aber in der Reihenfolge h2h1h0 auf, quasi big endian auf Bit bezogen, ihr macht es aber little endian auf Bit bezogen

2

4.5/8

$x = (x + ((y \cdot z) + 10))$





den Befehl gibt es nicht

-0.5

SP ACC

Zeilenabände zwischen den einzelnen Codepattern machen es leichter lesbar für Tutoren ^_^

den Befehl gibt es nicht

haltet euch am besten an die Patterns aus der Vorlesung

wenn ihr den Stack anstelle von Registern verwendet

den Befehl gibt es nicht

den Befehl gibt es ebeffalls nicht

wodzu das SUBI SP 1?

-2 keine Symboltabelle

-1 noch mehr nicht existente

den Befehl gibt es nicht

Befehle

ich schreibt es ab jetzt nicht jedes mal dazu

verwendet vielleicht mal den PicoC-Compiler mit seinem RETI-Interpreter um zu verstehen, wie der Stack funktionier und wann man ihn verkleinern oder vergrößern sollte

c

1.) Maximale Anzahl an Teilergebnissen: n

$(x_1 \circ (x_2 \circ (\dots (x_{n-1} \circ x_n) \dots))$

2.) Minimale Anzahl an Teilergebnissen: 2

$(\dots (x_n \circ x_{n-1}) \circ x_{n-2}) \dots) \circ x_1)$

Erika, Javier Betriebssysteme, Übung 5

Wenn ihr sowas getippt abgebt, könnte ich es viel genauer mithilfe des RETI-Interpreters bewerten.

5/6

3

Überlauf nur bei unterschiedlichen Vorzeichen,
ansonsten kann $x \leq y$ über $x - y \leq 0$ geprüft werden

```
1   LOAD ACC 11      ACC = y
2   LOAD IN1 10      IN1 = x
3   ANDI ACC 10...0₂ ACC = y ∧ 10...0₂   ist y negativ
4   ANDI IN1 10...0₂ IN1 = x ∧ 10...0₂   ist x negativ
5   OPLUS ACC IN1    ACC = ACC - IN1
6   Jump=3           PC = PC+3
```

wird ausgeführt, wenn beide selben Vorzeichen haben

```
7   Jump<6           PC = PC+6
```

wird ausgeführt, wenn $ACC - IN1 < 0$, also $y < 0$ und $x > 0$ ist, wegen $ACC = 10...0_2$ und $IN1 = 0$

```
8   Jump 7           PC = PC+7
```

wird ausgeführt, wenn $ACC - IN_1 > 0$, also $y > 0$ und $x < 0$ ist, wegen $ACC = 0$ und $IN_1 = 10...0_2$

```
9   LOAD ACC 10      ACC = x
10  LOAD IN1 11      IN1 = y
11  SUB ACC IN1      ACC = ACC - IN1
12  Jump>            PC = PC+3
13  LOADI ACC 1      ACC = 1
14  Jump 2           PC = PC+2
15  LOADI ACC 0      ACC = 0
16  END
```

dieser Teil wird ausgeführt, wenn Vorzeichen gleich sind, also kann über $x - y \leq 0$ geprüft werden

(JUMP 0)

funktioniert nicht, da Immedaites nur 22 Bit sind. Ich geh ab jetzt von der Annahme aus, dass es funktioniert.

daher OPLUS ACC

OPLUS

das ganze funktioniert leider nicht aufgrund einer Sache an die man nicht denkt: die Negation von 100000 ist 100000

genauer 100000 -> 11111 + 1 -> 100000, also bei 0 - 10...000 wird 10...000 rauskommen, daher muss mit AND etwas nachgeholfen werden

falsche Relation, da die Negation von <= die Relation > ist und nciht >=
-0.5

>

ist das eine Relation oder eine 2? ein JUMP 2 wäre hier jedenfalls korrekt

die Grundsätzliche Idee stimmt, aber das ganze funktioniert so nicht, da es das ÜBerlaufproblem bei der Negation von 10..0^30 gitb, welches die Einsatz von SUB ACC IN1 nicht erlaubt

einfach direkten JUMP, da ihr, sobald ihr ausschließen könnt, dass die Vorzeichen gleich sind, bereits wenn das Vorzeichen der ersten Zahl negativ ist, wisst, dass die anderen Zahl positviv sein muss und vice versa

-0.5

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.
Da eure Korrektur nicht getippt war, konnte ich sie nicht nicht mit dem RETI-Interpreter testen, aber ihr könnt unter https://github.com/matthejue/Abgaben_Blatt_3/tree/main/Blatt5/eigene_loesung.reti meine eigenen Lösung und die lauffähig gemachten Lösungen anderer Studenten mal inspizieren.