18/20

Name ist manchmal ganz pratkisch, damit ich nicht im Übungsportal die Namen nachschauen muss.

Betriebssysteme, Übung 3

Diesmal sieht die Korrektur etwas anders aus als sonst. Ich hab den RETI-Code aller Studenten 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 laufähig zu machen, sodass dieser die Aufgabenstellung erfüllt. Alle Korrekturanmerkungen sind in der `c.reti`-Datei als Kommentare zu finden. Die Dateien `c.uart_r` und `c.uart_s` sind zur Simualation einer UART da und stehen für das Empfangs- und Statusregister und die darin enthalten Zahlen werden sobald auf die entsprechendedn Register zugegriffen wird gepopt.
Eure Korrektur ist unter https://github.com/matthejue/Abgaben_Blatt_3/tree/main/Blatt3/mandarinen zu finden

12.5/14

```
3   LOADI IN1 0     // IN1 auf 0 setzen (hier kann später Inhalt aus R1 addiert werden)
4   LOADI DS 0      // Zugriff auf Daten im EPROM
5   LOAD DS r       // Konstante 010...0 in DS laden --> Zugriff auf UART
6   LOAD ACC 2      // Statusregister R2 in Akkumulator laden
7   ANDI ACC 2      // Prüft, ob b1 auf 1 gesetzt wurde (2 hat die Binärcodierung
                       102, also wird b0 nicht beachtet)
8   JUMP= -2        // Wenn b1=0, wird Statusregister neu geladen
9   LOAD IN1 1      // Lädt Daten in R1 auf IN1
```

b.)

```
1   LOADI SP a      // Im Stack-Pointer wird angezeigt, wohin der nächste Befehl ins
                       Hauptspeicher gespeichert werden muss
2   LOADI ACC 4     // Benutze ACC als Scheifenzähler
POLLING-LOOP
10  MULI IN1 256    // Durch die Multiplikation von IN1 mit 28, also 256,
                       werden die Bits um 8 Stellen nach links verschoben
11  SUBI IN2 1      // Setzt die Scheifenzähler um 1 runter
12  MOVE IN2 ACC    // Schiebt den Schleifenzähler ins ACC
13  STORE 2 0       // Setzt b1 wieder auf 0
14  JUMP≠ -8        // Wenn ACC, also auch der Schleifenzähler, nicht den Wert 0 hat,
                       wird Vorgang wiederholt ab 6 (also wo der Statusregister geladen
                       wird)
```

c.)

```
15   LOADI DS 0          // Zugriff auf Daten im EPROM
16   LOAD DS t           // Konstante 011100…0 in DS laden (Kodierung des Befehls LOADI PC
                            0)
17    MOVE IN1 ACC       // Schiebt Befehl im Register IN1 ins ACC

18   OPLUS ACC DS        // Prüft, ob Binärcodierung des Befehls identisch sind mit der des
                            Befehls LOADI PC 0 (alle Bits werden auf 0 gesetzt, wenn das der
                            Fall ist)
19   JUMP= 5             // Wenn alle Bits des ACC auf 0 gesetzt sind, wird auf 24 gesprungen
20   LOAD DS s           // Konstante 100...0 in DS laden --> Zugriff auf SRAM
21   STORE IN1 SP        // Befehl wird nun im Hauptspeicher gespeichert, der Stack-Pointer
                            gibt Adresse an
22   ADDI SP 1           // Stack-Pointer wird um 1 erhöht, um nächsten Befehl in
                            nächste Speicheradresse zu speichern
23   JUMP -21            // Wiederholt gesamten Vorgang, bis von der UART der Befehl
                            LOADI PC 0 erscheint
24   LOADI PC 0          // Beendet das Programm
```

Nr. 2

Es wird angenommen, dass im Datensegmentregister der Wert 0^{32} steht und im EPROM die Konstanten 010^{30} und 10^{31}. Weiterhin nehmen wir an, an der Adresse x des EPROMs steht die Konstante 010^{30} und an der Adresse y die Konstante 10^{31}. In IN2 werden die Konstanten geladen, in IN1 steht weiterhin der Inhalt von R1.
Die Konstante des Speichers, auf das zugegriffen werden soll, wird in IN2 geladen. Anschließend wird diese Konstante auf die Zieladresse, welches im SP steht, dazuaddiert. Es kann dabei kann Überlauf entstehen, da die Zieladresse 22-Bit codiert ist.
Für das Schreiben kann folgender Code verwendet werden:

LOAD IN2 v

STOREIN IN2 IN1 0

v ϵ {x, y}

5.5/6 richtige Idee

-0.5 den Befehl gitb es nicht

LOADIN usw.