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 lauffähig 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/granatapfel zu finden.

7.75/14 also 7.75/20

Betriebssysteme

Wintersemester 2022/23
Übungsblatt 03

John Zeeh (4725202) Daniel Burkhardt (4130647)

Aufgabe 1

Algorithm 1: Aufgabe a)

```
1 LOADI IN1 0 ;     // IN1 auf 0 setzen (später kann Inhalt aus R1 addiert werden
2 LOADI DS 0 ;                                    // Zugriff auf Daten im EPROM
3 LOAD DS r ;             // Konstante 010...0 in DS laden --> Zugriff auf UART
4 LOAD ACC 2 ;                        // Statusregister R2 in Akkumulator laden
5 ADDIN IN1 1 ;                                      // Prüfe, ob b1=1 im R2
6 JUMP= -2 ;                         // Falls b1 nicht 1, springe zu LOAD ACC 2
7 ADD IN1 1 ;                                          // Daten in IN1 laden
8 LOAD ACC 2 ;                                     // Lade 0 in Akkumulator
9 STORE ACC 2 ;                                          // Setze b1 auf 0
```

Algorithm 2: Aufgabe b)

```
1 LOADI IN2 4 ;                                // Benutze IN2 als Schleifenzaehler
2 POLLING-LOOP ;                                          // Code aus Teil a)
3 ADD IN1 0 ;     // Setze Instruktionsregister 1 auf 0, sodass Befehl ausführbar
4 JUMP> − 3; // Springe zu PC=1
```