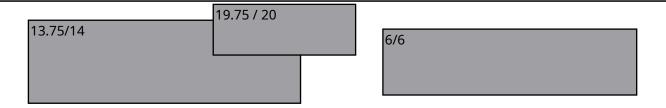
Diesmal sieht die Korrektur etwas anders aus als sonst. Ich hab den RETI-Code aller Studenten mithilfe des im PicoC-Compilers https://github.com/matthejue/PicoC-Compiler/releases eingebauten RETI-Interpreters ausgeführt, genauer mittels des Befehls `picoc\_compiler -b -p c.reti -S -P 2 -D 15`. Ich habe 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/mangos zu finden.



# Betriebssysteme WS22/23 Blatt 3

Darniel Augustin, Malte Pullich 11.11.2022

# Nummer 1

a)

```
LOADI IN1 0; IN1 auf 0 setzen (hier kann spaeter Inhalt
                 aus R1 addiert werden).
   LOADI DS 0
                  Zugriff auf Daten im EPROM
3
  LOAD DS r
                 Konstante 010...0 in DS laden
                ; —> Zugriff auf UART
5
   LOAD ACC 2
                ; Statusregister R2 in Akkumulator laden.
7
   ANDI ACC 2
                  Bitmaske um b1 gesondert zu ueberpruefen.
   JUMP = -2
                 Wenn b1 = 0 wird das Statusregister erneut
8
                  geladen.
10
   ADD IN1 1
                 Indexregister += R1
   LOAD ACC 2
                 R2 Laden
11
                ; b1 = 1 \text{ setzen}
12
   ORI ACC 2
   OPLUS ACC 2; b1 wird so auf jeden Fall auf 0 geflippt
13
                 Also mit ORI und XORI b1 = 0 setzen.
14
  STORE ACC 2; R2 mit b1 = 0 wieder speichern.
15
```

**b**)

```
Benutze IN2 als Schleifenzaehler.
  LOADI IN2 4
  LOADI IN1 0
                     IN1 auf 0 setzen
                    ; Linksshift, * 2^8 setzt 8 bits weiter.
   MULII IN1 256
   POLLING-LOOP
                      Code aus Teil a) au er Zeile 1.
                      Schleifenzaehler aufrufen.
  MOVE IN2 ACC
5
   SUBI ACC 1
                      Schleifenzaehler eins runter zaehlen.
6
   MOVE ACC IN2
7
                      Schleifenzaehler speichern.
8
   JUMP > -5
                      Solange die Schleifenabbruchbedingung
9
                      (Nicht mehr als 4 Schritte) nicht
                    ; nicht erfuellt ist an Anfang springen.
10
```

**c**)

```
LOADI SP a
                          Adresse a in den SP laden.
1
  SHIFT-LOOP
                          Code aus b)
3
   LOADI DS 0
                          Zugriff auf EPROM
  LOAD DS s
                          Zugriff auf SRAM
   STOREIN SP IN1 0
                          Befehl der UART in den SRAM an die
                          Stelle der Adresse in SP Speichern
7
   ADDI SP 1
                          SP += 1
8
   LOAD DS 0
9
   OPLUS IN1 t
                          Erwarteter Befehl und aktueller
10
                          Befehl XOR wenn das Ergebnis
                          != 0 naechsten Befehl ausfuehren
11
12
   MOVE IN1 ACC
   JUMP! = -8
13
                          Laden des naechsten Befehls
14
   LOADI PC a
                          Befehl von a in PC Laden/ausfuehren
```

# Nummer 2

### Speichern:

## Laden:

```
LOAD IN1 x ; x = Konstante des Speichers der ; adressiert werden soll LOADIN IN1 D i
```

Wir speichern zwei relative Konstanten für den Zugriff auf SRAM und UART ab und addieren diese auf die jeweilige Zieladresse.