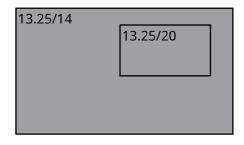
Diesmal sieht die Korrektur etwas anders aus als sonst. Ich hab den RETI-Code aller Studenten mithilfe des im PicoC-Compilers https://github.com/matthejue/PicoC-Compiler/releases eingebauten RETI-Interpreters ausgeführt, genauer mittels des Befehls `picoc_compiler -b -p c.reti -S -P 2 -D 15`. Ich habe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



Betriebssysteme

Übungsblatt 3

Mircea Sergiu Negrea 5168463

10. November 2022

Aufgabe 1

a)

```
LOADI IN1 0
                       // IN1 auf 0 setzen (hier kann später Inhalt aus R1 addiert werden).
LOADI DS 0
                       // Zugriff auf Daten im EPROM
LOADI DS r
                       // Konstante 010...0 in DS laden --> Zugriff auf UART
LOAD ACC 2
                       // Statusregister R2 in Akkumulator laden.
ANDI ACC 2
                       // b1 aus R2 durch Nullen aller anderen Bits isolieren.
                       // Falls das isolierte b1 = 1, dann ist das erwartete Ergebnis von SUBI 0.
SUBI ACC 2
JUMP_{!=} [-3]
                       // b1 != 0, d.h. b1 != 1 - den b1-Check wiederholen.
ADD IN1 O
                       // Daten sind da - auf genulltes IN1 addieren.
LOAD ACC 2
                       // Statusregister R2 erneut laden, da alte in ACC befindliche modifiziert.
ANDI ACC 253
                       // 253 = <11111101>_2 - alle bits in R2 übernehmen aber 0 auf b1 erzwingen.
b)
LOADI IN2 4
                       // Benutze IN2 als Schleifenzähler
POLLING-LOOP
                       // Daten abholen und in die ersten 8 bits von IN1 laden
MULTI IN1 256
                       // 2^8 = 256 --> Acht mal linksshiften
SUBI IN2 1
                       // Schleifenzähler dekrementieren
JUMP_{notequal} [-3] // Wiederholen bis Schleifenzähler = 0
c)
LOADI SP 0
                       // SP als SRAM-Adressoffset benutzen (<a> + SP)
LOADI IN2 4
                       // Benutze IN2 als Schleifenzähler
                      // Daten abholen und in die ersten 8 bits von IN1 laden
POLLING-LOOP
                       // 2^8 = 256 --> Acht mal linksshiften
MULTI IN1 256
                       // Schleifenzähler dekrementieren
SUBI IN2 1
JUMP_{notequal} [-3]
                       // Wiederholen bis Schleifenzähler = 0
LOADI DS 0
                       // Wechsle auf EPROM
LOAD DS s
                       // Wechsle auf SRAM
STOREIN a IN1 SP
                       // Speichere Inhalt von IN1 in M(a + SP) des SRAMs
ADDI SP 1
                       // Inkrementiere Adressoffset
LOADI DS 0
                       // Wechsle auf EPROM
LOAD ACC t
                       // Hole die Konstante zum Prüfen auf Programmende
LOAD DS s
                       // Wechsle auf SRAM
SUB ACC IN1
                       // Wenn in IN1 die Konstante für Programmende, dann ist ACC = 0 nach Subtraktion
JUMP_{notequal} [-12] // Wenn aktuelle Codezeile nicht Programmende
                       // dann springe zurück auf POLLING-LOOP
JUMP a
                       // Ende des Benutzerprogramms - springe auf Benutzerprogramm
```