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/melonen zu finden.

13.75/14	13.75/20

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       // Satusregister R2 in ACC laden.

ANDI ACC 2;      //  Prüfen, ob b_1 = 1

JUMP= -2;        // Wenn b_1 = 0, dann wieder zurück springen zu R2 in ACC laden

ADD IN1 1;       // Inhalt zum R1 auf IN1 addieren

LOAD ACC 2;      // Statusregister R2 in ACC laden

ANDI ACC 253;  // R2 mit 253 (11111101) verunden, sodass nur b_1 auf 0 gesetzt wird

STORE ACC 2;     //  R2-Inhalt mit zurückgesetzten  b_1 Bit wieder in R2 laden.
```

b)

Schleife läuft 4x durch aber mit richtigen Werten wird nur 3x die Bits verschoben, damit die ersten 8-Bit am Ende die vordersten im 32-Bit Befehl sind. Der LOADI IN1 0; Befehl darf in der a) nicht ausgeführt werden.

```
LOADI IN2 4;          // Benutzer IN2 als Schleifenzähler

LOADI IN1 0;          // Für den Anfang 0 in IN1 laden

MULI IN1 256;         // Bit shiften um 8 Bit

POLLING-LOOP;         // Code aus Teil a)

SUBI IN2 1;           // Den Schleifenzähler um 1 verringern

MOVE IN2 ACC;         // Den Wert von IN2 in ACC laden für den Vergleich

JUMP> -4;             // Wenn Schleifenzähler > 0, dann wieder in POLLING-LOOP
```

c)

Angenommen ich hab eine Kostante t2 = 1000111111...1 (Geflippte Bits von t)

```
LOADI  SP a;          // Adresse a auf StackPointer
Programm_B            // 32-Bit Befehl in IN1
LOADI DS 0;           // Zugriff auf Daten im EPROM
LOAD DS s;            // Zugriff auf Daten im SRAM
STOREIN SP IN1 0;     // aktuellen IN1 Befehl in Adresse a
ADDI SP 1;            // SP um 1 erhören für die nächste Adresse
LOADI DS 0;           // Zugriff auf EPROM
MOVE IN1 ACC;         // Wert von IN1 in ACC
ANDI ACC t2;          // Wert vom ACC verunden mit t2
JUMP!= -8;            // Falls ungleich 0, dann Programm wiederholen
LOADI PC 0;           // Falls 0, also der Beenden-Befehl ist drin, diesen ausführen.
```