

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 Simulation einer UART da und stehen für das Empfangs- und Statusregister und die darin enthalten Zahlen werden sobald auf die entsprechenden Register zugegriffen wird gepopt. Eure Korrektur ist unter https://github.com/matthejue/Abgaben_Blatt_3/tree/main/Blatt3/pflaumen zu finden.

Benke Hargitai 5370932

Lukas Seyfried 5343019

13.25/14

4/6

17.25/20

Aufgabenblatt 03

Abgabe: 11.11.2022

Aufgabe 1

```
; POLLING LOOP:
LOADI IN1 0 ; IN1 auf 0 setzen (hier kann spaeter Inhalt aus R1 addiert werden).
LOADI DS 0 ; Zugriff auf Daten im EPROM.
LOAD DS r ; Konstante 010...0 in DS laden --> Zugriff auf UART.
LOAD ACC 2 ; Statusregister R2 in Akkumulator laden.

ANDI ACC 2 ; Bitmaske, die alle Bits außer dem vorletzten auf 0 setzt.
JUMP= -2 ; Ist das vorletzte Bit ebenfalls 0, dann springe zurück
; und lade neue Daten.

LOAD IN1 1 ; b1 = 1, Schreibvorgang von R1 nach IN1.
OPLUS ACC 2 ; ACC = 0...0010 0+ R2, b1 wird geflippt.
STORE ACC 2 ; Speichern des neuen Status in R2.

; INPUTCOMMAND:
LOADI IN2 4 ; Benutze IN2 als Schleifenzähler.
MULI IN1 2^8 ; Linkshift durch Multiplikation, vor POLLING damit
; der Linksshift nur 3 mal auf die Eingabe wirkt.

POLLING ; Code aus Teil a).
SUBI IN2 1 ; Anpassung des Schleifenzählers.
MOVE IN2 ACC ; Kopiervorgang von IN2 nach ACC, da JUMP den Akkumulator vergleicht.
JUMP> -4 ; Schleife bis IN2 = 0

; WRITECODE:
LOADI DS 0 ; Zugriff auf Daten im EPROM.
LOAD DS s ; Konstante 100...0 in DS laden --> Zugriff auf SRAM.
LOADI CS a ; Speichern der Startadresse in Register.

INPUTCOMMAND ; Code aus Teil b), setzt DS wieder auf UART.

LOADI DS 0 ; Zugriff auf Daten im EPROM.
LOAD ACC t ; lädt Kodierung von "LOADI PC 0" in den Akkumulator.
LOAD DS s ; Konstante 100...0 in DS laden --> Zugriff auf SRAM.

STOREIN CS IN1 0 ; Kopiervorgang von IN1 in die nächste Speicheradresse
; nach dem bisher eingelesenen Code.
ADDI CS 1 ; Anpassung des CS.

OPLUS ACC IN1 ; Vergleich des letzten geschriebenen Befehls mit "LOADI PC 0".
JUMP> -7 ; Sprung zum Einlesen des nächsten Befehls.

LOADI DS 0 ; Zugriff auf Daten im EPROM.
LOAD ACC s ; Konstante 100...0 (SRAM) in ACC laden.
ADDI ACC a ; Volle Adresse des ersten eingelesenen Befehls
; in den Akkumulator geschrieben.
MOVE ACC PC ; Ausführung des eingelesenen Programs.
```

Aufgabe 2

Man kann statt LOAD und STORE Befehle den Befehl INT i und RTI benutzen:

Z.B. statt

```
LOADI DS 0 ; Zugriff auf Daten im EPROM.  
LOAD DS r ; Konstante 010...0 in DS laden --> Zugriff auf UART.  
...  
STORE ACC 2 ; Speichern des neuen Status in R2.
```

4/6
-2 da es etwas gecheated
ist

schreiben wir

```
INT r ; Wir springen gleich ins UART.  
...  
STORE ACC 2 ; Speichern des neuen Status in R2.  
RTI
```

würde ich zählen lassen, man könnte so
eine ISR definieren.

Kommentar: es stimmt sehr wahrscheinlich nicht, aber ich habe das Vorlesungsmaterial bzgl. INT nicht verstanden:(

INT i ist einfach nur ein Softwareinterrupt. i ist ein Index in der Interruptvektortabelle in der die Anfangsadressen der Interruptserviceroutinen stehen. Man springt nach dem sichern des PC's dann in diese ISR und führt diese aus, bis der Befehl RTI auftaucht.