

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/mangos zu finden.

13.75/14

19.75 / 20

6/6

Betriebssysteme WS22/23

Blatt 3

Darniel Augustin, Malte Pullich

11.11.2022

Nummer 1

a)

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

b)

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

c)

1	LOADI SP a	; Adresse a in den SP laden.
2	SHIFT-LOOP	; Code aus b)
3	LOADI DS 0	; Zugriff auf EPROM
4	LOAD DS s	; Zugriff auf SRAM
5	STOREIN SP IN1 0	; Befehl der UART in den SRAM an die
6		; 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
11		; != 0 naechsten Befehl ausfuehren
12	MOVE IN1 ACC	
13	JUMP!= -8	; Laden des naechsten Befehls
14	LOADI PC a	; Befehl von a in PC Laden/ausfuehren

Nummer 2

Speichern:

1	LOAD IN1 x	; x = Konstante des Speichers der
2		; adressiert werden soll
3	STOREIN IN1 S i	

Laden:

1	LOAD IN1 x	; x = Konstante des Speichers der
2		; adressiert werden soll
3	LOADIN IN1 D i	

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