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/kokosnuesse zu finden.

11.5/14 und somit 11.5/20

Betriebssysteme Blatt 3

Baran Güner, bg160 Tobias Hangel, th151

11. November 2022

Aufgabe 1

a)

PC	Befehl	Kommentar
1	LOADI IN1 0	IN1 auf 0 setzen (hier kann später Inhalt aus R1 addiert werden).
2	LOADI DS 0	Zugriff auf Daten im EPROM
3	LOAD DS r	Konstante 010...0 in DS laden $\rightarrow$ Zugriff auf UART
4	LOAD ACC 2	Statusregister R2 in Akkumulator laden.
5	SUBI ACC $010^{27}010$	Wenn b1 1 ist, wird das Ergebnis 0 oder 1. Sonst wird es negativ.
6	$\text{Jump}_{<}$ -2	Wenn b1 0 ist, wird der PC um 2 zurück gesetzt.
7	ADD IN1 1	Der Inhalt von R1 wird auf IN1(0) addiert.
8	SUBI 2 10	In R2 wird b1 bon 1 auf 0 gesetzt.
9	JUMP -8	PC wird um 8 zurück gesetzt.

b)

Hier ist davoon auszugehen, dass im POLLING-LOOP der Code insofern abgeändert ist, als dass der erste Befehl IN1 nicht mehr auf 0 setzt, da sonst in jedem Loop alles gelöscht wird.

PC	Befehl	Kommentar
1	LOADI IN2 4	Benutze IN2 als Schleifenzähler
2	LOADI IN1 0	IN1 auf 0 setzen (für Teil c), da sonst alte Befehle neue abändern)
3	POLLING-LOOP	Code aus Teil a)
4	MULI IN1 100.000.000	IN1 wird mit 100 Mio. multipliziert, damit die ersten 8 bit frei sind
5	SUBI IN2 1	Schleifenzähler wird um 1 reduziert
6	LOAD ACC IN2	IN2 wird für den folgenden JUMP-Befehl in den ACC geladen
7	$\text{JUMP}_{>}$-4	Springt zum Befehl 3 zurück wenn der ACC größer als 0 ist.

c)

PC	Befehl	Kommentar
1	DATA-LOOP	Code aus Teil b)(und a)) wird ausgeführt, neuer Befehl befindet sich in IN1
2	LOADI DS 0	Zugriff auf Daten im EPROM
3	LOAD DS s	Konstante 10^{31} in DS laden $->$ Zugriff auf SRAM
4	STORE IN1 a	Speichert den Befehl aus IN1 in M(a) ab
5	LOAD ACC a	Der neuste Befehl wird in den ACC geladen
6	ADDI a 1	a wird für den nächsten Befehl um 1 erhöht.
7	LOADI DS 0	Zugriff auf Daten im EPROM
8	SUB ACC t	Der ACC wird um den LOADI PC 0 Befehl(01110000000...0) reduziert
9	$JUMP_{=}2$	Der JUMP findet nur dann statt, wenn der ACC vorher LOADI PC 0 war
10	JUMP -9	Falls der Befehl nicht LOADI PC 0 war wird der nächste Befehl geladen
11	LOAD DS s	Konstante 10^{31} in DS laden $->$ Zugriff auf SRAM
12		"Sprung zu a"(Keine Ahnung was das heißen soll)