KSP Aufgabe 3

1. Implementieren Sie die neuen Instruktionen zum Testen von Zahlen (`eq`, `ne`, `lt`, `le`, `gt`, `ge`), die die numerischen Vergleiche (`==`, `!=`, `<`, `⇐`, `>`, `>=`) repräsentieren. Die Auswirkungen auf den Stack können Sie wieder in der Spalte Stack Layout der VM Instruktionen sehen. Das Ergebnis eines Vergleichs, ein Boole'scher Wert, wird durch die ganze Zahl `0` für `false` bzw. `1` für `true` dargestellt. Natürlich gibt's einen Assembler , der auch die neuen Instruktionen assemblieren kann.

Table 1. VM Instruktionen

Instruktion	Opcode	Stack Layout
`eq`	17	`... n1 n2 -> ... n1==n2`
`ne`	18	`... n1 n2 -> ... n1!=n2`
`lt`	19	`... n1 n2 -> ... n1<n2`
`le`	20	`... n1 n2 -> ... n1<=n2`
`gt`	21	`... n1 n2 -> ... n1>n2`
`ge`	22	`... n1 n2 -> ... n1>=n2`
`jmp <target>`	23	`... -> ...`
`brf <target>`	24	`... b -> ...`
`brt <target>`	25	`... b -> ...`

1. Nehmen Sie sich dann den unbedingten Sprung `jmp` sowie die beiden bedingten Sprünge (*branch on false* `brf` und *branch on true* `brt`) vor. Der Immediate-Wert in diesen Instruktionen gibt das Sprungziel an. Die bedingten Sprünge prüfen das oberste Stack-Element, um zu entscheiden, ob gesprungen wird. Wenn nicht gesprungen wird, kommt die nächste Instruktion zur Ausführung. Die Auswirkungen auf den Stack können Sie wieder in der Spalte Stack Layout in VM Instruktionen sehen. Der Assembler hat eine neue Kommandozeilenoption `--map`; was kann man eigentlich mit der anfangen?

Teilaufgabe: Interaktiver Debugger

Nun kommt die erste etwas größere Aufgabe: **ein interaktiver Debugger** für Ihre VM. Die Instruktionen werden komplizierter, die auszuführenden Programme umfangreicher - man wünscht sich, mehr über den inneren Zustand der VM zu erfahren, speziell wenn Fehler auftreten.

a) Im Gegensatz zu den früheren Aufgaben soll ab jetzt das Programm nach dem Laden nicht mehr automatisch gelistet werden, sondern einfach nur ausgeführt werden. Wenn man aber die VM mit dem Kommandozeilenschalter `--debug` startet, soll sich nach dem Laden des Programms der interaktive Debugger melden und auf Kommandos warten, die er dann ausführt.

b) Schreiben Sie eine kurze, aber exakte Spezifikation, welche Kommandos Ihr Debugger verstehen soll. Als kleiner Hinweis für diejenigen, die sich wenig unter einem solchen Teil vorstellen können: Was wünscht man sich, um den Programmverlauf verfolgen oder vielleicht sogar beeinflussen zu können? Lassen Sie doch einfach mal dieses Beispielprogramm 1 laden und stellen Sie sich vor, es würde nicht das gewünschte Ergebnis (den größten gemeinsamen Teiler zweier positiver Zahlen)

liefern. Natürlich können Sie das auch ganz praktisch ausprobieren, indem Sie irgendeine Instruktion (z.B. das zweite `brf`) in eine andere Instruktion (z.B. `brt`) umändern, oder noch besser: von Ihrem anderen Grupenmitglied umändern lassen, ohne dass Sie wissen, was da passiert ist. Was braucht man dann für Hilfsmittel, um den Fehler zu lokalisieren?

Als Minimalmenge müssen die Kommandos: *Anzeigen des Stacks*, *Anzeigen der statischen Daten*, *Listen des Programms*, *Ausführen des nächsten Befehls*, *Weiterlaufen ohne Anhalten* und *Verlassen der VM* aufgenommen werden.

Sehr zweckmäßig ist auch das *Setzen eines Breakpoints* (wenn das Programm bei der Ausführung später dort vorbeikommt, hält es an und der Debugger übernimmt die Kontrolle). Anregungen können Sie sich natürlich wie immer auch von der Referenzimplementierung holen: njvm

c) Implementieren Sie nun die von Ihnen vorgesehen Kommandos des Debuggers. Prüfen Sie mit dem Beispielprogramm 1 und Beispielprogramm 2 sowie anderen, von Ihnen geschriebenen Testprogrammen, ob der Debugger das leistet, was Sie sich von ihm erwarten. Wahrscheinlich fallen Ihnen noch Verbesserungen ein - dann gehen Sie zurück zu b).