Aufgabe 2 (2+4+2 Punkte)

Geben Sie Ihre Lösung für diese Aufgabe in einer Datei `arithmetic.reti` ab.

Betrachten Sie folgendes Pico-C Programm mit Deklarations- und Anweisungsteil.

```
void main() {
  //Deklarationsteil
  int x;
  int y;
  const int z = 5;
  //Anweisungsteil
  y = 3;
  x = 2;
  x = (x + ((y * z) + 10)); // vollstaendig geklammerter Ausdruck
}
```

1. Schreiben Sie die Befehlsfolge der erweiterten RETI in die Datei `arithmetic.reti`, die den im obigen Programm dargestellten vollständig geklammerten Ausdruck auswertet und das Ergebnis oben auf den Stack schreibt.
2. Schreiben Sie in einen Kommentar am Ende der Datei `arithmetic.reti` die Einträge der Symboltabelle für den Deklarationsteil, wie sie sich in Ihrem geschriebenen Programm `arithmetic.reti` ergeben.
3. Angenommen Sie haben n Variablen $x_1, \ldots, x_n$, die durch (beliebige) binäre Operanden zu einem vollständig geklammerten Ausdruck verknüpft werden, sodass jede Variable exakt einmal vorkommt.
   (a) Wie sieht der vollständig geklammerte Ausdruck aus, der die maximale Anzahl an Teilergebnissen auf dem Stack erfordert? Wie viele Teilergebnisse sind das?
   (b) Wie sieht der vollständig geklammerte Ausdruck aus, der die minimale Anzahl an Teilergebnissen auf dem Stack erfordert? Wie viele Teilergebnisse sind das?

   Schreibe Sie Ihre Antwort in einen Kommentar am Ende der Datei `arithmetic.reti`.

Aufgabe 3 (6 Punkte)

Die Auswertung von Vergleichsoperatoren aus Pico-C erfolgt auf der RETI, indem die Differenz der linken und rechten Seite gebildet wird und diese (je nach Vergleichsoperator) mit 0 verglichen wird. Beispiel: $x \leq y$ wird zu $x - y \leq 0$ Dieses Vorgehen kann jedoch zu Überläufen führen, wenn die Ausdrücke auf beiden Seiten ein unterschiedliches Vorzeichen haben.

Geben Sie zur Behandlung dieses Problems innerhalb des folgenden Codegerüsts ein RETI-Programm in einer Datei `comparison.reti` an, das den logischen Ausdruck $x \leq y$ korrekt zu 0 bzw. 1 auf den Stack auswertet, auch wenn bei der Berechnung von $x - y$ ein Überlauf auftritt. Dazu soll entsprechend 1 oder 0 auf den Stack abgespeichert werden, sodass das Codegerüst darauffolgend das richtige Ergebnis vom Stack lesen kann. Die beiden zu vergleichenden Variablen x und y werden vom vorgegebenen Codegerüst vorher auf den Stack gespeichert.

```
# input: -2097152 2097151
SUBI SP 2
INT 2
STOREIN SP ACC 2 # Zahl 1
INT 2
STOREIN SP ACC 1 # Zahl 2
INT 3 # Breakpoint, springen sie mittels 'c' im Debug Mode ('-d') hierher
```

```
# Ihr Code kommt hierher
LOADIN SP ACC 1 # Ergebnis
INT 0
ADDI SP 1
JUMP 0
```

Testen Sie Ihr Programm mithilfe des RETI-Interpreters mittels des folgenden Aufrufs:

```
reti_interpreter -i ./isrs.reti ./comparison.reti -d -b -m -E
```

wobei Sie die Datei `isrs.reti` unter folgender URL: https://github.com/matthejue/RETI-Interpreter/blob/main/run/isrs.reti herunterladen können.

Die Kommandozeilenoption `-E` erlaubt es Ihnen Interrupt-Service Routinen direkt komplett auszuführen ohne in Sie hineinzuspringen. Mittels des Kommandos ‘s‘ ist es im Debug Mode weiterhin möglich in eine Interrupt-Service Routine hineinzuspringen, wenn der `PC` direkt auf einen `INT i`-Befehl zeigt. Wir wollten euch Studenten in den letzten Übungsblättern nicht mit zu vielen neuen Kommandos verwirren und haben daher erweiterte Funktionalitäten hinter der Option `-E` versteckt.