Tag 24: Arithmetic Logic Unit

Untersuchung des Problems

Zunächst kann man einen Parser implementieren, der die Abläufe in der ALU simuliert. Diesen kann man probeweise anschließend mit dem als Puzzle Input gegebenen Programm und einigen 14 stelligen Modellnummern füttern, um die Funktionsweise zu testen.

Man wird sehr wahrscheinlich erkennen, dass der Wert des z-Registers - scheinbar unabhängig von der eingegebenen Modellnummer - immer größer wird.

Der Versuch, alle denkbaren Modellnummern in der so geschaffenen ALU durch das Programm prüfen zu lassen, schlägt (zumindest mit Java) fehl, da die Eingabemenge mit 10^{14} potentiellen Kandidaten dafür zu groß ist.

Reverse Engeneering

Man sollte sich also als nächstes den als Puzzle-Input gegebenen Code ansehen. Bei einer ersten Analyse fällt auf, dass die 14 Segmente, die jeweils von einem `inp w` Befehl eingeleitet werden, der die nächste Ziffer der Modellnummer einliest sich sehr ähnlich sind. Im wesentlichen gibt es zwei Arten von jeweils 18 Zeilen langen Befehlssegmenten[1)]:

```
  Variante A       Variante B

1  inp w            inp w
2  mul x 0          mul x 0
3  add x z          add x z
4  mod x 26         mod x 26
5  div z 1          div z 26
6  add x 11         add x -5  (ADD_TO_X)
7  eql x w          eql x w
8  eql x 0          eql x 0
9  mul y 0          mul y 0
10 add y 25         add y 25
11 mul y x          mul y x
12 add y 1          add y 1
13 mul z y          mul z y
14 mul y 0          mul y 0
15 add y w          add y w
16 add y 6          add y 12  (ADD_TO_Y)  Hier ist immer y=w+ADD_TO_Y
17 mul y x          mul y x
18 add z y          add z y
```

Betrachtung der Unterschiede und Auswirkungen

- In Zeile 5 taucht wahlweise `div z 1` oder `div z 26` auf. Ersteres verändert den wert von z nicht, letzteres dividiert z durch 26.
- In Zeile 6 wird mit `add x ADD_TO_X` ein Wert zu x addiert. Hier gibt es zwei Fälle:
  - (1) ADD_TO_X ist positiv und größer oder gleich 9: Dann ist in Zeile 7 `x + ADD_TO_X` niemals gleich w, da w eine Ziffer zwischen 0 und 9 ist. Das hat zur Folge, dass in Zeile 7 x immer auf 0 und in Zeile 8 x auf 1 gesetzt wird.
  - (2) ADD_TO_X ist negativ: Das tritt nur dann auf, wenn der Befehl in Zeile 5 `div z 26` lautet. Nun hängt es davon ab welcher Wert in x gespeichert ist, ob in Zeile 7 `x + ADD_TO_X` gleich w ist oder nicht - meist ist er das nicht, wenn doch ist x nach Zeile 8 0.
- In Zeile 16 unterscheiden sich die Anweisungen in der Zahl, die hier mit dem Befehl `add y ADD_TO_Y` zum Register y addiert wird.
- Zum genaueren Verständnis ist außerdem die Bedeutung von Zeile 13 interessant:
  - Wenn x=1 ist, wird der Wert von z hier mit 26 multipliziert.
  - Wenn in seltenen Fällen x=0ist , bleibt z unverändert. da die Zeile 10, in der y auf 25 gesetzt wird durch Zeile 11 "annuliert" wird, so dass nach Zeile 12 y=1 ist.

Interessant sind also vor allem die Fälle, bei denen in Zeile 5 `div z 26` steht, im weiteren verlauf in Zeile 6 x verkleinert wird und in Zeile 7 der auf diese Weise berechnete x-Wert gleich der Eingabeziffer w ist. Nur dann wird nämlich der Wert im Register z kleiner als zuvor, und dieser soll ja am Ende 0 sein.

Was passiert also?

Das Register z implementiert einen Stapel, af dem in jedem Schritt die Summe aus der Ziffer der Modellnummer und ADD_TO_Y abgelegt wird.

Standardfall

(`div z 1` und `add x <WERT>` mit `<WERT> > 0`)

```
Ziffer(w)     y              z
Start         0              0
3             3+ADD_TO_Y     3+ADD_TO_Y
2             2+ADD_TO_Y     (26*z1) + (2+ADD_TO_Y)
4             4+ADD_TO_Y     (26*z2) + (4+ADD_TO_Y)
....
```

Variante B

Wenn auf die obige Folge nun Variante B angewendet wird, erhält man in Zeile 4 den zuletzt abgelegten Wert im Register x zurück:

```
z3 = (26*z2) + (4+ADD_TO_Y) % 26 -> (4+ADD_TO_Y)
```

Außerdem wird z durch 26 dividiert, d.h. in z steht anschließend der vorige z-Wert:

```
z4 = z3  % 26 ->  (26*z2)
```

In x ist jetzt die vorige Ziffer der Modellnummer + das ADD_TO_Y aus dem vorigen Anweisungsblock gespeichert.

Jetzt kommt es darauf an, ob die Bedingung

```
Vorige Ziffer + ADD_TO_Y - ADD_TO_X = aktuelle Ziffer  (*)
```

erfüllt ist. Wenn ja, bleibt der Wert vom Stapel entfernt, wenn nein wird z in Zeile 13 wieder mit 26 multipliziert und der Wert wandert damit wieder auf den Stack.

(*) ist also die Bedingung, die erfüllt sein muss, damit der Stack kleiner wird.

Zusammenfassung

Im Puzzle-Input gibt es 7 Blöcke, die auf jeden Fall `(Ziffer+ADDTO_Y)` auf den Stack pushen. Außerdem gibt es 7 Blöcke, die bei erfüllen der Bedingung (*) einen Wert vom Stapel entfernen. Jeder dieser 7 Blöcke definiert eine Bedingung zwischen zwei der 14 Ziffern der Modellnummer - diese müssen also alle erfüllt sein, dann ist die Modellnummer gültig.

1)
für den mir vorliegenden und auf dieser Seite zur Verfügung gestellten Input