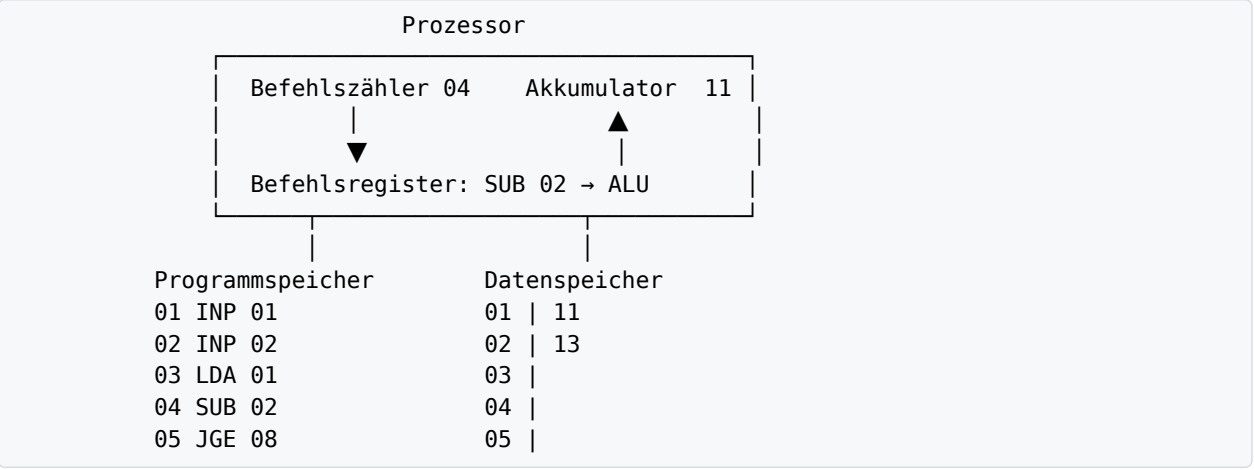
6.2 Die Registermaschine

ZUSATZ

Wir bauen uns eine Maschine mit dem Nötigsten: ein Programmspeicher, ein Datenspeicher, ein Akkumulator, ein Befehlsregister, ein Befehlszähler.



Die Zeichnung ist eine **Momentaufnahme mitten im Befehl** `04 SUB 02`: Der Befehl steht im Befehlsregister, der Operand wird gerade aus Datenzelle 02 geholt, im Akkumulator steht noch der alte Wert 11. Erst der vierte Schritt des Befehlszyklus schreibt $11 - 13 = -2$ hinein, der fünfte setzt den Befehlszähler auf 05.

Ein Befehl besteht immer aus drei Angaben: seiner **Befehlsadresse** im Programmspeicher, der **Operation** und einem **Adressteil**. Was der Adressteil bedeutet, hängt von der Operation ab: bei `LDA`, `STA`, `ADD` oder `SUB` eine Zelle des Datenspeichers, bei `LDK` die Zahl selbst, bei den Sprungbefehlen eine Adresse im *Programm*speicher, bei `OUT 00` den Akkumulator und bei `HLT 99` gar nichts.

6.2.1 Der Befehlssatz

Transportbefehle

Befehl	Wirkung
`LDA <Adresse>`	Lade den Inhalt der Adresse in den Akkumulator
`LDK <Zahl>`	Lade die Konstante in den Akkumulator
`STA <Adresse>`	Speichere den Akkumulatorinhalt in die Adresse

Verarbeitungsbefehle

Befehl	Wirkung
`ADD <Adresse>`	Addiere den Inhalt der Adresse zum Akkumulator
`SUB <Adresse>`	Subtrahiere den Inhalt der Adresse vom Akkumulator
`MUL <Adresse>`	Multipliziere den Akkumulator mit dem Inhalt der Adresse
`DIV <Adresse>`	Dividiere den Akkumulator durch den Inhalt der Adresse; der Rest fällt weg

Sprungbefehle

Befehl	Wirkung
`JMP <Adresse>`	Springe zur Adresse
`JEZ <Adresse>`	Springe, wenn Akkumulator $= 0$
`JNE <Adresse>`	Springe, wenn Akkumulator $\neq 0$
`JLZ <Adresse>`	Springe, wenn Akkumulator < 0
`JGZ <Adresse>`	Springe, wenn Akkumulator > 0
`JLE <Adresse>`	Springe, wenn Akkumulator ≤ 0
`JGE <Adresse>`	Springe, wenn Akkumulator ≥ 0

Sonstige Befehle

Befehl	Wirkung
`INP <Adresse>`	Lies eine Zahl ein und speichere sie in der Adresse
`OUT <Adresse>`	Gib den Inhalt der Adresse aus (Adresse 00 ist der Akkumulator)
`HLT 99`	Programmende

Stolperstein

`DIV` ist eine **Ganzzahldivision**: $7 : 2$ ergibt 3, der Rest 1 verfällt. Für unsere Programme rechnen wir nur mit nichtnegativen Zahlen — und das aus gutem Grund: Python rundet mit `//` immer nach unten (`-7 // 2` ergibt -4), Pascal schneidet mit `div` zur Null hin ab (`-7 div 2` ergibt -3). Für negative Operanden liefern die beiden Simulatoren weiter unten also verschiedene Ergebnisse. Das ist keine Schlamperei, sondern eine echte Festlegung, die jede Maschine treffen muss.

Eine solche maschinennahe Sprache heißt **Assemblersprache** (von *to assemble*, montieren): Die Befehle sind Bausteine, die zu einem Programm zusammengesetzt werden.

6.2.2 Beispiel: das Minimum zweier Zahlen

Gesucht ist $\min(a, b)$. Die beiden Zahlen sollen in den Datenzellen 01 und 02 liegen.

```
01 INP 01      ; a in Speicherzelle 01
02 INP 02      ; b in Speicherzelle 02
03 LDA 01      ; a in den Akkumulator
04 SUB 02      ; Akku = a - b
05 JGE 08      ; Sprung, wenn a - b >= 0, also b <= a
06 LDA 01      ; sonst: a ist kleiner
07 JMP 09      ; Sprung zur Ausgabe
08 LDA 02      ; b ist kleiner oder gleich
09 OUT 00      ; Ausgabe des Akkumulatorinhalts
10 HLT 99      ; Programmende
```

Verfolgen Sie den Ablauf für $a = 11$ und $b = 13$: In Zeile 04 steht -2 im Akkumulator, der Sprung in Zeile 05 findet nicht statt, Zeile 06 lädt $a = 11$, Zeile 07 überspringt Zeile 08, Zeile 09 gibt 11 aus. Richtig.

Merke

Die Maschine kennt kein `if`. Eine Verzweigung entsteht aus genau zwei Zutaten: einer Rechnung, die das Ergebnis der Bedingung ins Vorzeichen des Akkumulators bringt, und einem bedingten Sprung darauf. Genau das erzeugt Ihr Compiler aus jedem `if`.