

Maschinennahe Programmierung

- Maschinencode ist sehr aufwändig von Hand zu schreiben
- Maschinennahe Programmierung ist manchmal notwendig (bspw. bei Firmware/Betriebssystem)
- Assembler-Sprache erlaubt hardwarenahes Programmieren in besser lesbarer Form
 - Befehlsvorrat und Syntax abhängig von der verwendeten Prozessorrchitektur

- Die meisten Befehle entsprechen direkt einem zugehörigen Maschinenbefehl
- Verwendung von leicht(er) merkbaren Kürzeln für die Befehle (Mnemonics)
 - Beispiel (x86-Assembler): `mov eax, [ebx]`

- Simulator für eine einfache Registermaschine
 - Alle Operationen werden auf dem **Akkumulator** (spezielles Prozessorregister) ausgeführt
 - Die Ergebnisse landen wieder im Akkumulator
- Beispielprogramm:

```
LOADI 20    # Zahl (nicht Adresse!) 20 in den Akkumulator laden
STORE 100   # Wert im Akkumulator an Adresse 100 speichern
LOADI 30    # Zahl 30 in den Akkumulator laden
STORE 101   # Wert im Akkumulator an Adresse 101 speichern
LOAD 100    # Wert an Adresse 100 in Akkumulator laden
ADD 101     # Wert an Adresse 101 zum Akkumulatorinhalt addieren
STORE 102   # Wert im Akkumulator an Adresse 102 speichern
HOLD       # Prozessor anhalten
```

- Lade die Minimaschine [herunter](#)
- Implementiere und teste das Beispielprogramm von der letzten Folie
- Schaue Dir die Befehlsübersicht auf der Webseite der Minimaschine an

Aufgabe 2

- Erstelle ein Programm, das drei Zahlen addiert und die Summe in den Speicher schreibt
 - Die Zahlen können mittels **LOADI** geladen werden
- Erstelle ein Programm für den Rechenausdruck $Z_1 - Z_2 + 2Z_3$, wobei Z_1, Z_2, Z_3 für drei verschiedene, selbst gewählte natürliche Zahlen stehen
- Erstelle ein Programm für den Rechenausdruck $Z_1 - 2 \cdot (Z_2 - Z_3)$

Hinweis: Vor dem Testen eines neuen Programms sollte man zunächst den Speicherinhalt löschen

- Am Anfang einer Zeile kann ein Label definiert werden
 - Zeichenfolge gefolgt von einem Doppelpunkt
- Ein solches Label kann statt einer Adresse verwendet werden:

```
LOAD Wert1
```

```
ADD Wert2
```

```
STORE Ergebnis
```

```
Wert1: WORD 10
```

```
Wert2: WORD 20
```

```
Ergebnis: WORD 0
```

- Schreibe das Programm zur Berechnung von $Z_1 - 2 \cdot (Z_2 - Z_3)$ neu
 - Verwende analog zum Beispiel auf der vorherigen Folie Labels für die Zahlen Z_1, Z_2, Z_3

- Schreibe ein Programm, das die größere von zwei Zahlen ermittelt und im Speicher ablegt

- Schreibe ein Programm, das die Zahlen von 1 bis 20 nacheinander an eine beliebige, freie Adresse im Speicher schreibt
 - Wie lassen sich Schleifen in Assembler realisieren?
- Zusatzaufgabe: Modifiziere das Programm so, dass die Zahlen in aufeinander folgende Speicherzellen geschrieben werden

STORE 100 Wert im Akkumulator an Adresse 100 speichern

STORE (100) Wert im Akkumulator an die Adresse speichern, die an
Adresse 100 im Speicher steht

Funktioniert bei allen Befehlen, die eine Adresse als Parameter erwarten

- Schreibe ein Programm, das die Summe aller Zahlen von 1 bis n mittels einer geeigneten “Schleife” berechnet
- Schreibe ein Programm, das die Quersumme einer natürlichen Zahl berechnet