
# Einführung Assembler

## Was und warum?

Das Hauptelement eines Computers ist der Mikroprozessor. Die Aufgabedes Mikroprozessors ist es, Daten zu manipulieren, also zu verändern.

Über ein **Leitungssystem (Bus)** kann der Prozessor Daten mit Speicher- und Peripheriebausteinen austauschen. Fur die Verarbeitung der Daten verfügt er über einige interne Speicherplätze, die sogenannten **Register** .

Jedes Programm, das auf einem Computer ausgeführt wird, wird in viele kleine Einzelschritte zerlegt, die der Prozessor dann ausführt, um Daten mit anderen Teilen des Rechners auszutauschen, zu manipulieren und wieder auszugeben. Wenn wir in einer "höheren" Prögrammiersprache wie Java, C++ oder PHP programmieren, übernehmen Compiler und Interpreter die Übersetzung unserer Programme in diese kleinen Einzelschritte die der Prozessor verstehen kann.

Ein Prozessor verfügt über eine gegebene Menge an Aktionen, die er ausführen kann[1)], den **Befehlssatz**. Die Befehle des Befehlssatzes heißen **Maschinenbefehle**. Es gibt Maschinenbefehle für den Datenaustausch mit Speicherzellen, für das Ansprechen von Peripheriegeräten, für den Transport zwischen Registern, für Veränderung von Daten und für vieles mehr.

Maschinenbefehle sind letztlich nur binäre Bitmuster aus Nullen und Einsen, z.B.:

```
00010101 00000000 00101010 10001011 11011000 ...
```

Das kann man nun platzsparend Hexadezimal schreiben[2)]

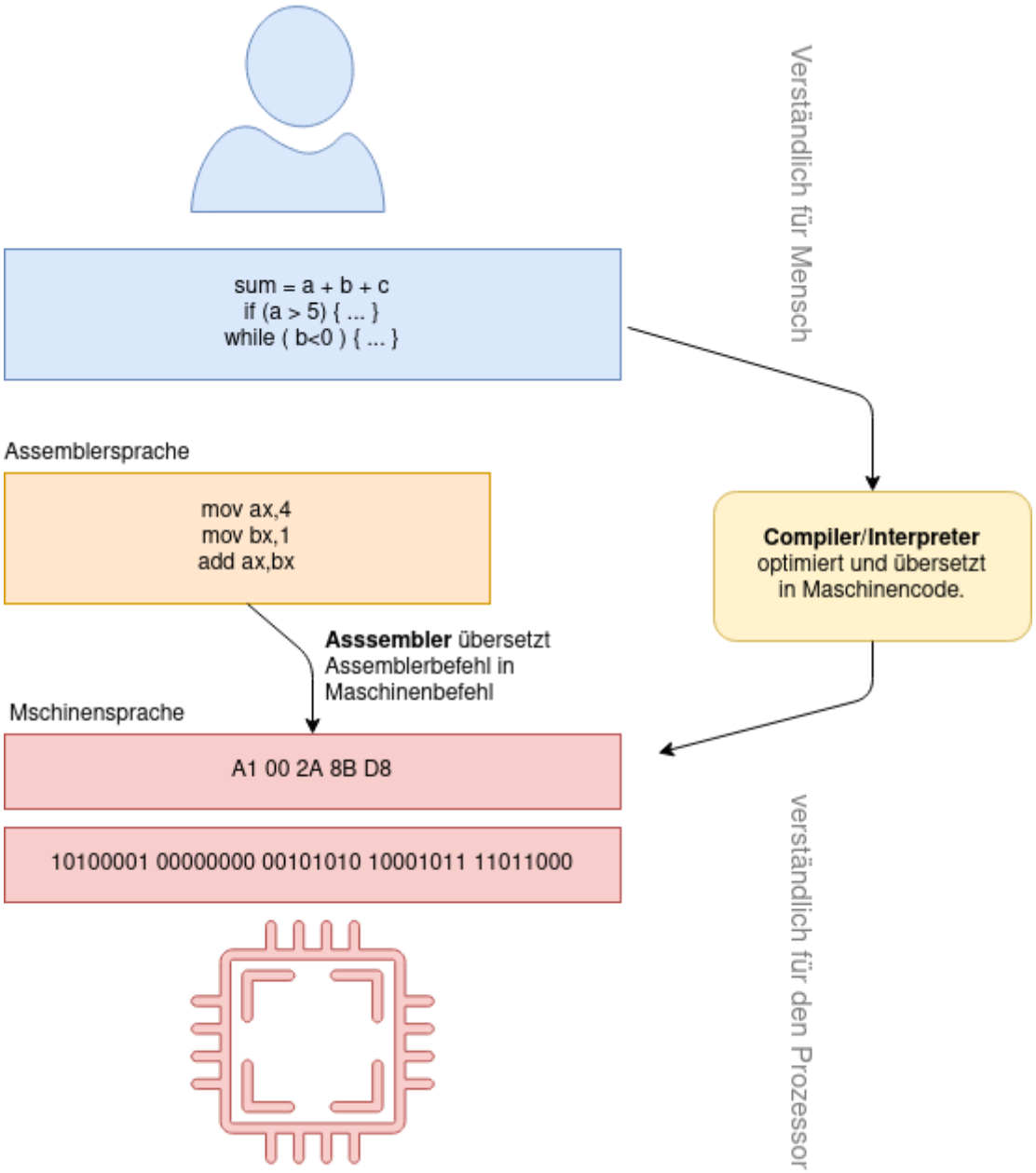
```
A1 00 2A 8B D8 ...
```

Das ändert jedoch nichts am Umstand, dass die Darstellung für uns Menschen wenig intuitiv ist.

Es gibt jedoch Anwendungsfälle in denen man sehr nah an der Hardware programmieren möchte oder muss - wann immer man Kontrolle darüber erlangen möchte, was der Prozessor genau macht.

Die **Assemblersprache** ist eine fast vollständige 1:1 Repräsentation der Maschinenbefehle des Prozessors, der **Assembler** kann Assemblerbefehle also direkt in Maschinensprache übersetzen - auf diese Weise ist es auch für uns Menschen möglich, dem Prozessor direkte Anweisungen zu geben. Assemblerbefehle bestehen meist aus 3 Buchstaben, den sogenannten **Mnemonics**.

Die folgende Grafik veranschaulicht die Situation:



# Anders denken...

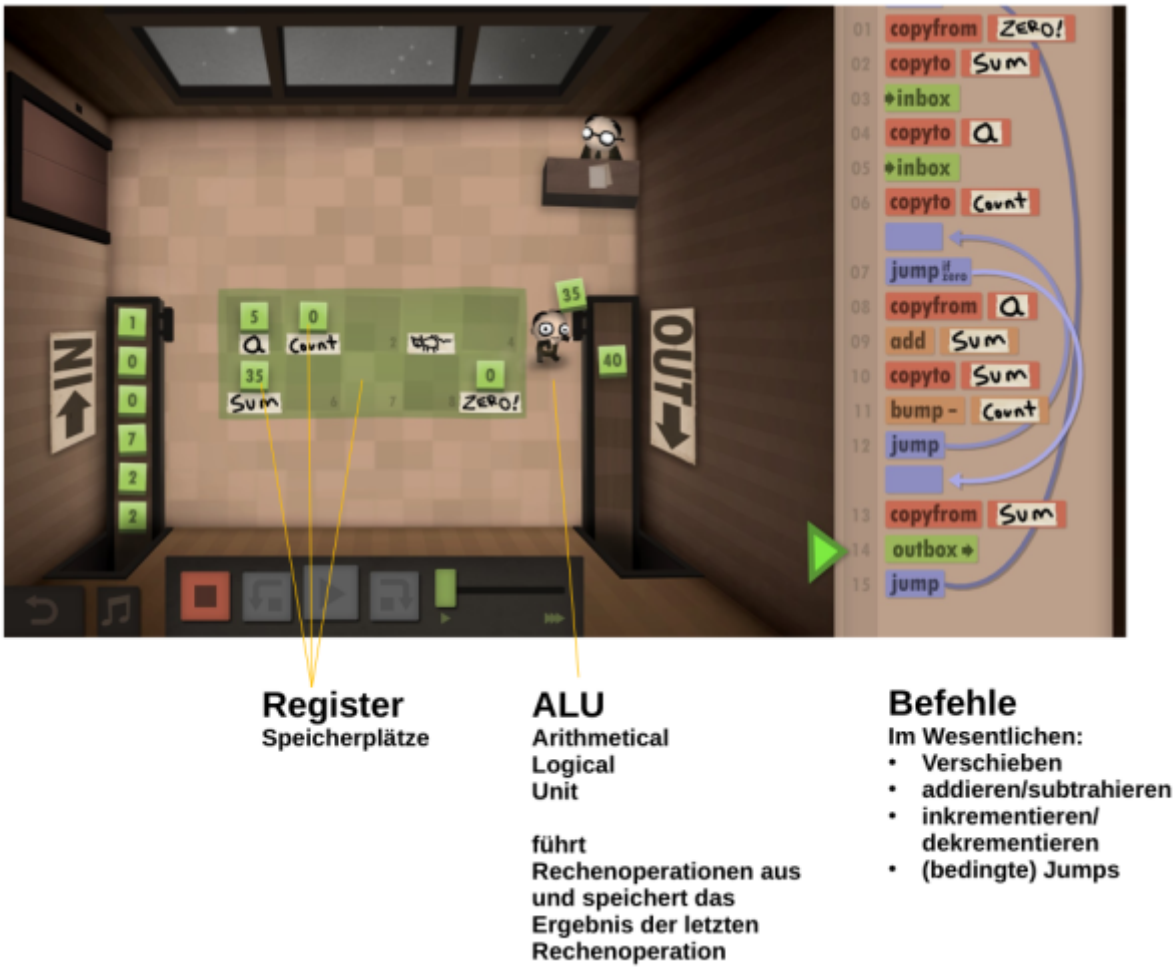
Mit Hilfe von Assemblerbefehlen kann ein Ausdruck wie

```
sum = a + b + c
```

nicht direkt dargestellt werden, da sich die zur Verfügung stehenden Befehle daran orientieren, welche Register der Prozessor hat und welche Operationen er unterstützt. Das Vorgehen bei der Lösung von Problemen wird dadurch sehr kleinschrittig, die zur Verfügung stehenden Befehle sind sehr beschränkt:

```
mov eax,[a] ;Schreibe den Inhalt der Speicherzelle a ins Register eax
add eax,[b] ;Addiere den Inhalt der Speicherzelle b zum Inhalt des Registers
eax (eax ist jetzt a+b)
add eax,[c] ;Addiere den Inhalt der Speicherzelle c zum Inhalt des Registers
eax (eax ist jetzt a+b+c)
```

Um uns an eine solche Problemlösestrategie zu gewöhnen, kann man ein Spiel spielen: https://tomorrowcorporation.com/humanresourcemachine (Gibt es auch für iOS und Android, ca. 5-10EUR).



# Dateien

| | | |
|---|---|---|
| arch.png | 34.8 KiB | 22.07.2021 09:44 |
| assembler_einstieg.odp | 3.4 MiB | 20.07.2021 12:59 |
| assembler_einstieg.pdf | 463.7 KiB | 20.07.2021 12:59 |
| compile_assemble.png | 43.9 KiB | 22.07.2021 08:36 |
| hrm.png | 429.4 KiB | 22.07.2021 08:53 |
| prozessor.jpg | 63.4 KiB | 22.07.2021 08:09 |

1)

welche das genau sind, hängt von der Prozessorarchitekur ab

2)

Erinnere dich: 4 Bit sind eine Hexadezimalzahl