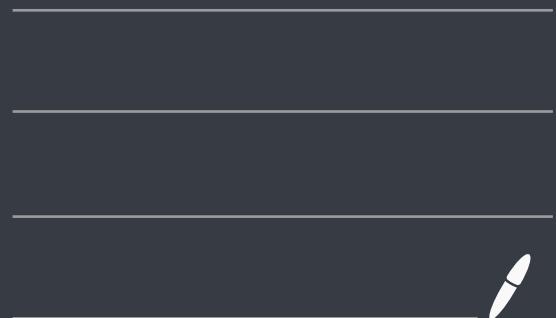


Betriebssysteme



Themen dieser VO:

verstehen, was das bedeutet

./selfie -c selfie.c -m 3 -c selfie.c -y 3 -c selfie.c

Mit irgendeinem Compiler
(gcc) gebootstrapiertes (kompiliertes)
Programm "selfie" (ELF x86 binary)

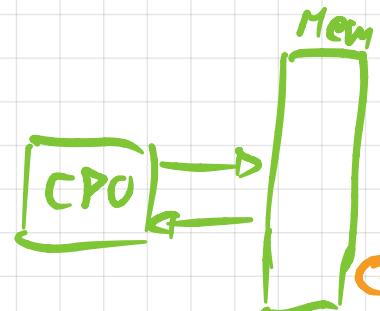
technisch gesehen eine Funktion
 $f: \{0,1\}^* \rightarrow \{0,1\}^*$, die Bitsequenzen, die
Text repräsentieren, zu Bitsequenzen,
die x86-CPU's ausführen können,
abbildet.

Text-Bitsequenzen (Code)
sind bloß Abstraktions-
ebenen. Der Compiler legt
die Bedeutung der zu
übersetzenden Sprache fest.

↪ ähnlich wie Chinesisch \leftrightarrow Deutsch
übersetzen.

was heißt "ausführen
können"?

↪ Von Neumann Modell:



↪ Code wird von
Platte geladen

Selfie kompiliert C* zu RISC-U

Selfie ist in der Sprache geschrieben, die er kompiliert

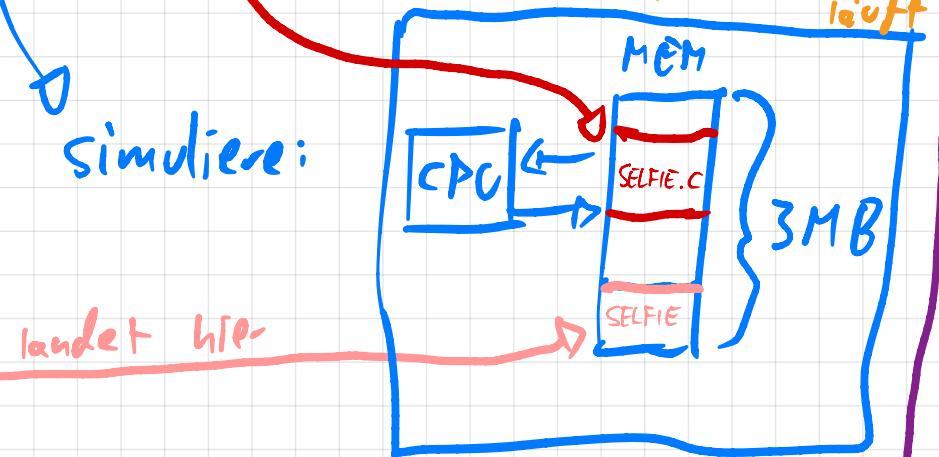
↳ Selfie kann sich selbst kompilieren.

`./selfie -c selfie.c` `-m 3 -c selfie.c`

hieraus wird
"anonyme Binary" erstellt,
die auf RISC-U Maschine
ausführbar wäre (z.B. Selfie's
RISC-U Emulator)

↳ dieses Ausführen passiert hier

Hypervisor – führt auch
Instruktionen aus,
interpretiert aber nicht,
sondern führt den
`-y 3 -c selfie.c` ^{Code auf}
Maschine aus,
auf der der Hypervisor selber
läuft



führt auf Hypervisor &
nochmal Selfie aus just for the
lolz

Analogie zu OS:

./selfie -c selfie.c -m 3 -c selfie.c -y 3 -c selfie.c

"Hardware"

"Software"

aka OS.

Wir sollen experimentieren: "was passt, wenn...?" \rightarrow Hypothese aufstellen und prüfen.

Anderen Dudes den Shit erklären. Code angucken etc.

[Loc]

Weitere Randnotiz: [Travis] "C I" - continuous integration. Bei jeder Änderung der Software wird auf irgendeinem Server direkt alles getestet.

Review

1. Encoding

- Dezimalzahlen binär kodieren können (und dekodieren)
- Base-8, Base-16 ebenso
- $42_{10} = 101010_2$, aber woher kommt die Bedeutung / warum ist 101010 42 ? \rightarrow Wir geben sie ihr. Sind nur Symbole, könnte ja alles bedeuten

Bei uns kommt die Bedeutung davon, was die Maschine damit anstellt, wenn wir ihr die Sequenz geben

Maschine hat keine Semantik, wenn wir sie "einfrieren", nur wenn sie läuft, ist die Semantik "in Hardware"

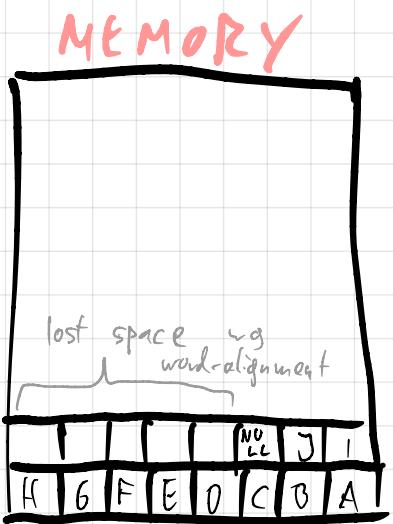
↗ kann immer nur von 8 byte auf 8 bit reeden

byte-addressed/word-aligned Speicher

Wie sind Strings im Speicher abgelegt?

Wir haben ASCII-Strings → 8 bit/Symbol.

Eine Speicheradresse hält 8 bytes → 8 Zeichen von LSB zu MSB geordnet. Strings sind null-terminiert.



Selfie line 4972: Opcode Encoding
zB

addi \$rd, \$rs1, \$rs2

bedeutet semantisch:

64-bit addition
mit Overflow

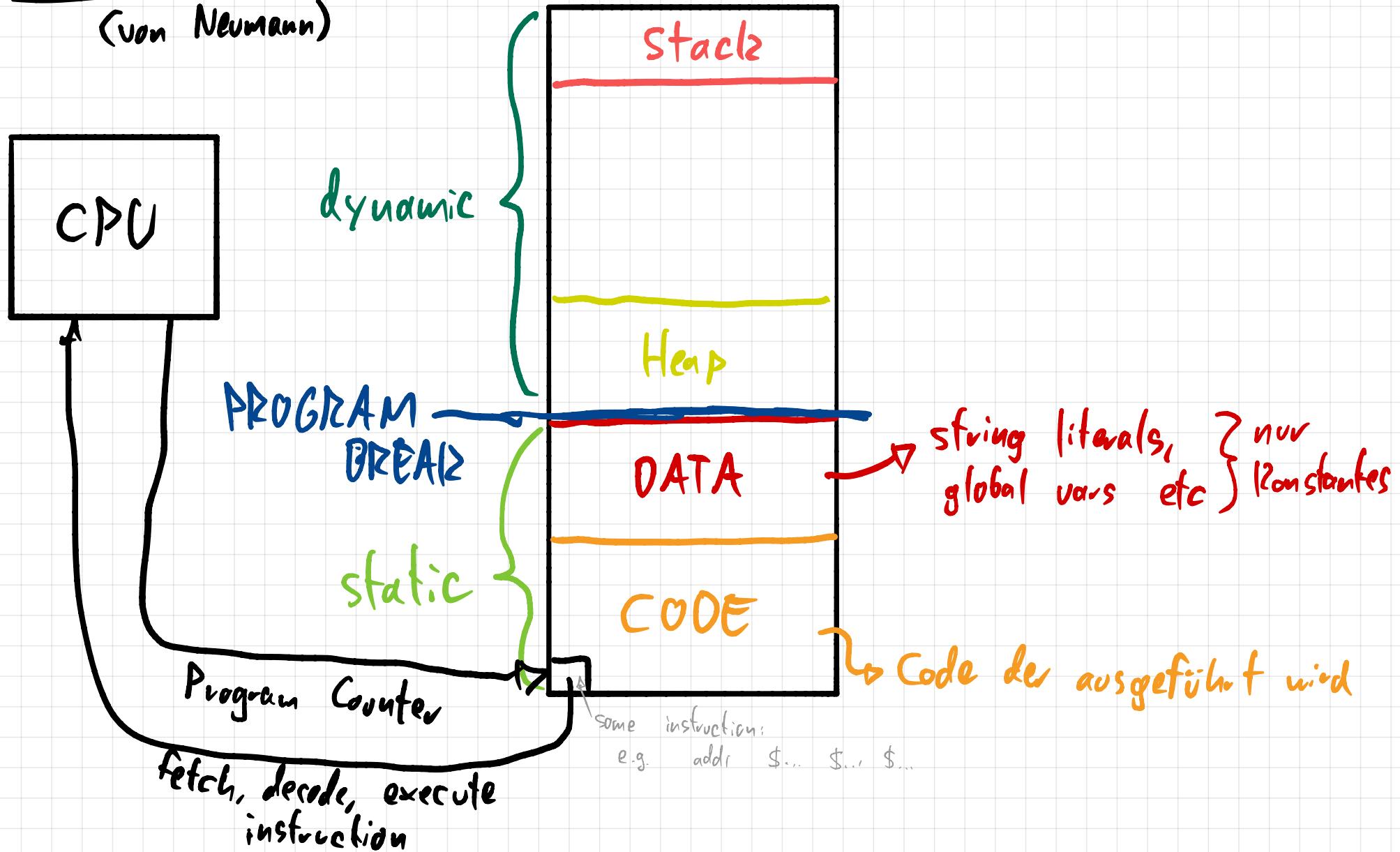
$$\$rd = \$rs1 + \$rs2$$

NUR

Bei Division und Remainder macht es 1 Unterschied,
ob man Zahlen als signed oder unsigned betrachtet

2. Speichersmodell

(von Neumann)



3. Compiler

oder auch via Automat
wie in FL. ↗

Definition
reguläre Grammatik:
alles, was in einen
EBNF-Ausdruck passt
↗ bzw. in eine
Produktionsregel
passt.

Scanner - wandelt das zu Tokens.
[lexer]

(Symbolsequenzen)
Tokens sind reguläre Ausdrücke
(z.B. digit etc., ist EBNF)

selfie nutzt "recursive descent" parser: d.h. parst rekursiv,
+ "single-pass": generiert Code

Parser - hat einen Stack (kann "zählen"), macht;

Syntaxanalyse

→ wenn passt, ist Programm
syntaktisch korrekt.

bis hierhin

O Semantik:

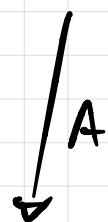
nur, ob Input passende Form hat.

Selfie Code Notiz:

compile - Prefix für Methoden,
zum Kompilieren
do_ für Instruktionen

4. Virtualization

Jeder Prozess auf einer Maschine soll glauben, er läge die Maschine für sich:



Maschine für jeden Prozess emulieren

Zusammenfassung:

Systeme setzen sich aus vielen trivialen Komponenten zusammen, aber das Zusammenspiel ist fucking insane.

context switches

Komplexester Shit

(→ erfordert Isolierung ... via Segmentation z.B. oder Paging)

page: virtuelles Konzept
vs
page frame: physisch
+
page table: mappt
virtuell → physisch

→ Lerne RISC-V Instruktionen, C# lesen und schreiben

können