PROGRAMMIERUNG

ÜBUNG 10: C_0 UND ABSTRAKTE MASCHINE AM_0

Eric Kunze

eric.kunze@tu-dresden.de

INHALT

Implementierung von C_0 und abstrakte Maschine AM_0

C_0 **UND** AM_0

- **Ziel:** Implementierung einer einfachen Programmiersprache $C_1 \subset C$

C_0 UND AM_0

- **Ziel:** Implementierung einer einfachen Programmiersprache $C_1 \subset C$
- **Hier:** zunächst Einschränkung auf $C_0 \subset C_1$
  - genau eine main-Funktion
  - Zugriff auf `stdio` durch `#include`
  - einzig zugelassende Datenstruktur: `int`, Konstanten
  - Kontrollstrukturen: Ein-/Ausgabebefehle, Zuweisungen, Sequenzen, Verzweigungen, bedingte Schleifen

C_0 **UND** AM_0

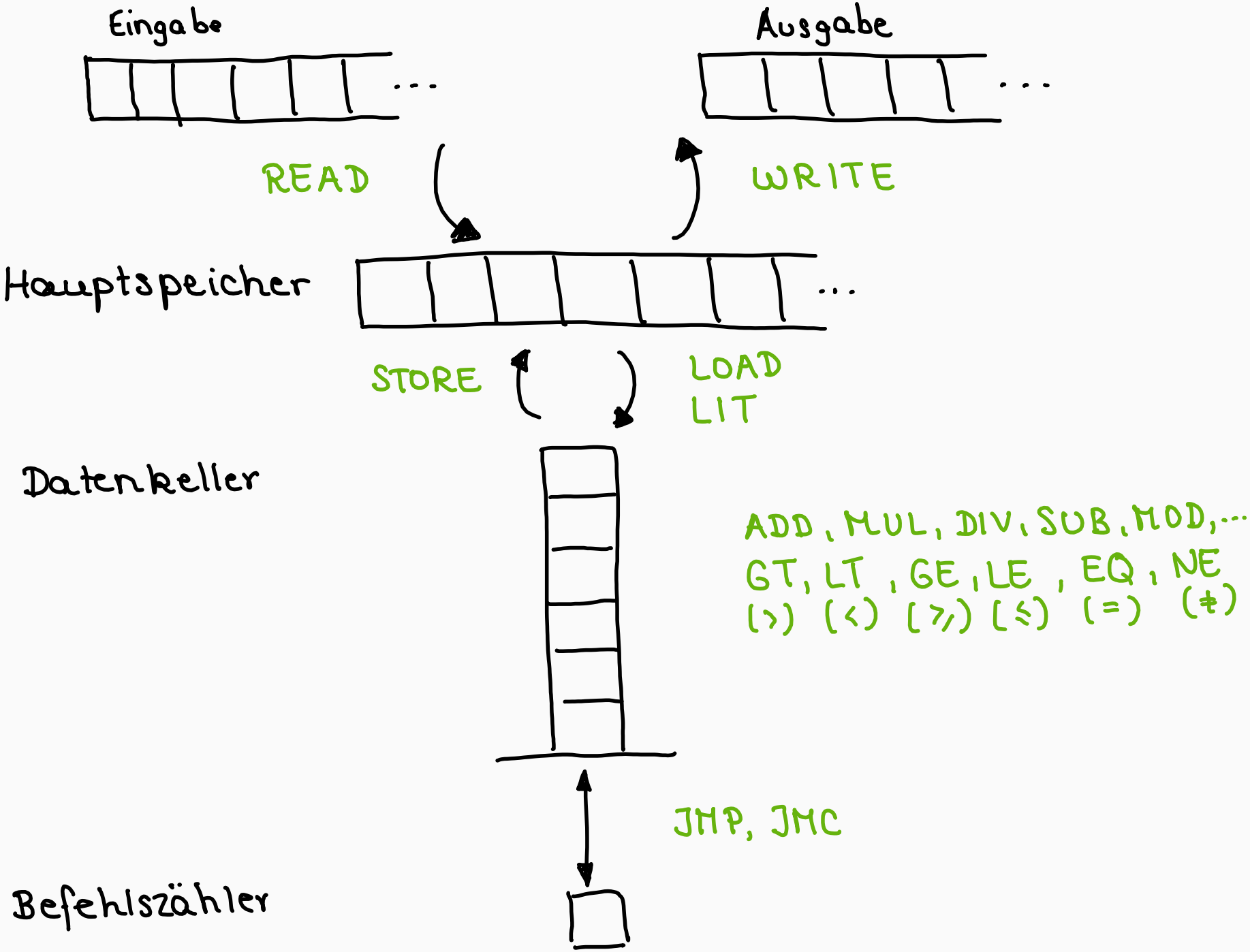
- **Ziel:** Implementierung einer einfachen Programmiersprache $C_1 \subset C$
- **Hier:** zunächst Einschränkung auf $C_0 \subset C_1$
  - genau eine main-Funktion
  - Zugriff auf `stdio` durch `#include`
  - einzig zugelassende Datenstruktur: `int`, Konstanten
  - Kontrollstrukturen: Ein-/Ausgabebefehle, Zuweisungen, Sequenzen, Verzweigungen, bedingte Schleifen
- **Implementierung** durch
  - Syntax von C_0
  - Befehle und Semantik einer abstrakten Maschine AM_0
  - Übersetzer $C_0 \leftrightarrow AM_0$

BEFEHLE UND SEMANTIK DER AM_0

Wir bauen eine abstrakte Maschine AM_0, die unsere Berechnungen ausführen kann. Wir benötigen dafür:

- ein Ein- und Ausgabeband,
- einen Datenkeller,
- einen Hauptspeicher und
- einen Befehlszähler

Nun müssen aber auch Aktionen ausgeführt werden, wie zum Beispiel das Einlesen vom Eingabeband in den Hauptspeicher. Dafür gibt es folgende Befehle:


Eingabe
Ausgabe
READ
WRITE
Hauptspeicher
STORE
LOAD
LIT
Datenkeller
ADD, MUL, DIV, SUB, MOD, ...
GT, LT, GE, LE, EQ, NE
(>) (<) (≥) (≤) (=) (≠)
JMP, JMC
Befehlszähler

SEMANTIK DER BEFEHLE

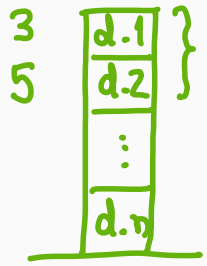
Den Zustand der abstrakten Maschine beschreiben wir durch die Zustände der 5 Komponenten, also als 5-Tupel

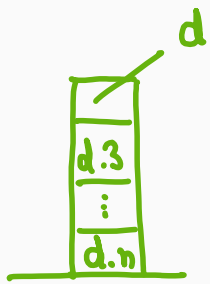
$$(m, d, h, inp, out)$$
$$= (\text{Befehlszähler}, \text{Datenkeller}, \text{Hauptspeicher}, \text{Input}, \text{Output})$$

Jeder Befehl verändert den Zustand der Maschine – er verändert also die Einträge in diesem Tupel.

5 - 3

LOAD "5"
LOAD "3"





$$\mathcal{C}[\![\text{SUB}]\!](m, d, h, inp, out) :=$$
$$\text{if } d = d.1 : d.2 : \cdots : d.n$$
$$\text{then } (m + 1, (d.2 - d.1) : d.3 : \cdots : d.n, inp, out)$$

h

SEMANTIK DER BEFEHLE

formation der AM_0 dieser Befehl bewirkt. Dabei schreiben wir statt $\mathcal{C}[\![.]\!](\gamma)$ nun $\mathcal{C}[\![\gamma]\!]$.

- $\mathcal{C}[\![\text{ADD}]\!](m, d, h, inp, out) :=$
  if $d = d.1 : d.2 : d.3 : \ldots : d.n$ mit $n \geq 2$ then $(m+1, (d.2 + d.1) : d.3 : \ldots : d.n, h, inp, out)$

  für MUL analog

  $\mathcal{C}[\![\text{SUB}]\!](m, d, h, inp, out) :=$
  if $d = d.1 : d.2 : d.3 : \ldots : d.n$ mit $n \geq 2$ then $(m+1, (d.2 - d.1) : d.3 : \ldots : d.n, h, inp, out)$

  für DIV und MOD analog

  $\mathcal{C}[\![\text{LT}]\!](m, d, h, inp, out) :=$
  if $d = d.1 : d.2 : d.3 : \ldots : d.n$ mit $n \geq 2$ then $(m+1, b : d.3 : \ldots : d.n, h, inp, out)$
  wobei $b = 1$, falls $d.2 < d.1$, und $b = 0$, falls $d.2 \geq d.1$,
  d. h. für den Wert *true* (bzw. *false*) wird 1 (bzw. 0) abgelegt

  für EQ, NE, GT, LE und GE analog

- $\mathcal{C}[\![\text{LOAD } n]\!](m, d, h, inp, out) :=$
  if $h(n) \in \mathbb{Z}$ then $(m+1, h(n) : d, h, inp, out)$

  $\mathcal{C}[\![\text{LIT } z]\!](m, d, h, inp, out) := (m+1, z : d, h, inp, out)$

  $\mathcal{C}[\![\text{STORE } n]\!](m, d, h, inp, out) :=$
  if $d = d.1 : d'$ then $(m+1, d', h[n/d.1], inp, out)$

  wobei $h[n/d.1](k) = \begin{cases} d.1 & \text{falls } k = n \\ h(k) & \text{sonst} \end{cases}$

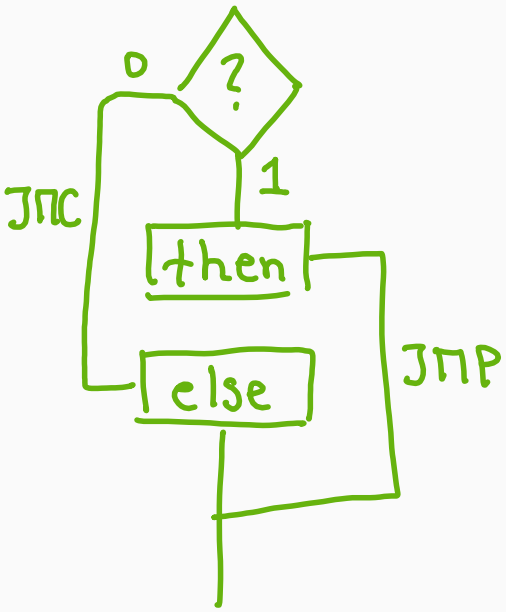
- $\mathcal{C}[\![\text{JMP } e]\!](m, d, h, inp, out) := (e, d, h, inp, out)$

  $\mathcal{C}[\![\text{JMC } e]\!](m, d, h, inp, out) :=$
  if $d = 0 : d.2 : \ldots : d.n$ mit $n \geq 1$ then $(e, d.2 : \ldots : d.n, h, inp, out)$
  if $d = 1 : d.2 : \ldots : d.n$ mit $n \geq 1$ then $(m+1, d.2 : \ldots : d.n, h, inp, out)$

  Es wird also zum Befehl mit der Nummer e gesprungen, wenn das oberste Kellerelement gleich 0 ist; die 0 repräsentiert den Wert *false*. Wenn das oberste Kellerelement gleich 1 ist (und damit den Wert *true* repräsentiert), dann wird der Befehlszähler um 1 inkrementiert.

- $\mathcal{C}[\![\text{READ } n]\!](m, d, h, inp, out) :=$
  if $inp = first(inp).rest(inp)$ then $(m+1, d, h[n/first(inp)], rest(inp), out)$
  wobei für jedes $n \in \mathbb{Z}$ und $w \in \mathbb{Z}^*$ gilt: $first(n : w) = n$ und $rest(n : w) = w$

ÜBERSETZUNG VON `IF - THEN - ELSE`



$sttrans(\texttt{if (}exp\texttt{)}\ stat_1\ \texttt{else}\ stat_2, tab, a) :=$

	$boolexptrans(exp, tab)$	Bedingung
	`JMC` $a.1$;	
	$sttrans(stat_1, tab, a.2)$	then
	`JMP` $a.3$;	
$a.1$:	$sttrans(stat_2, tab, a.4)$	else
$a.3$:		

für alle $exp \in \mathrm{W}(\langle \mathrm{BoolExpression} \rangle)$, $stat_1, stat_2 \in \mathrm{W}(\langle \mathrm{Statement} \rangle)$, $tab \in \mathrm{Tab}$ und $a \in \mathbb{N}^*$.

AUFGABE 1

Wir betrachten das C_0-Programm *Max*:

```
1  #include <stdio.h>
2
3  int main( ) {
4    int a, b, max;
5    scanf("%i", &a);
6    scanf("%i", &a);
7    if (a > b)
8      max = a;
9    else max = b;
10   printf("%d", max);
11   return 0;
12 }
```

(a) Berechnen Sie schrittweise das baumstrukturierte Programm $bMax_0 = \text{trans}(Max)$ mit Hilfe der in der Vorlesung angegebenen Übersetzungsfunktionen. Dokumentieren Sie dabei jeden rekursiven Funktionsaufruf.

AUFGABE 1 – TEIL (A)

Baumstrukturierte Adressen:

```
        READ 1;
        READ 2;
        LOAD 1;
        LOAD 2;
        GT;
        JMC 1.3.1;
        LOAD 1;
        STORE 3;
        JMP 1.3.3;
1.3.1   LOAD 2;
        STORE 3;
1.3.3   WRITE 3;
```

AUFGABE 1 – TEIL (A)

Baumstrukturierte Adressen:

```
      READ 1;
      READ 2;
      LOAD 1;
      LOAD 2;
      GT;
      JMC 1.3.1;
      LOAD 1;
      STORE 3;
      JMP 1.3.3;
1.3.1 LOAD 2;
      STORE 3;
1.3.3 WRITE 3;
```

Linearisierte Adressen:

```
1  READ 1;
2  READ 2;
3  LOAD 1;
4  LOAD 2;
5  GT;
6  JMC 10;
7  LOAD 1;
8  STORE 3;
9  JMP 12;
10 LOAD 2;
11 STORE 3;
12 WRITE 3;
```

AUFGABE 1 – TEIL (B)

Ablauf der abstrakten Maschine:

	BZ	,	DK	,	HS	,	Inp	,	Out	
(	1	,	ε	,	[ ]	,	5:7	,	ε	)
(	2	,	ε	,	[1/5]	,	7	,	ε	)
(	3	,	ε	,	[1/5, 2/7]	,	ε	,	ε	)
(	4	,	5	,	[1/5, 2/7]	,	ε	,	ε	)
(	5	,	7:5	,	[1/5, 2/7]	,	ε	,	ε	)
(	6	,	0	,	[1/5, 2/7]	,	ε	,	ε	)
(	10	,	ε	,	[1/5, 2/7]	,	ε	,	ε	)
(	11	,	7	,	[1/5, 2/7]	,	ε	,	ε	)
(	12	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	ε	)
(	13	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	7	)

AUFGABE 1 – TEIL (B)

Ablauf der abstrakten Maschine:

	BZ	,	DK	,	HS	,	Inp	,	Out	
(	1	,	ε	,	[ ]	,	5:7	,	ε	)
(	2	,	ε	,	[1/5]	,	7	,	ε	)
(	3	,	ε	,	[1/5, 2/7]	,	ε	,	ε	)
(	4	,	5	,	[1/5, 2/7]	,	ε	,	ε	)
(	5	,	7:5	,	[1/5, 2/7]	,	ε	,	ε	)
(	6	,	0	,	[1/5, 2/7]	,	ε	,	ε	)
(	10	,	ε	,	[1/5, 2/7]	,	ε	,	ε	)
(	11	,	7	,	[1/5, 2/7]	,	ε	,	ε	)
(	12	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	ε	)
(	13	,	ε	,	[1/5, 2/7, 3/7]	,	ε	,	7	)

$$\mathcal{P}[\![Max_0]\!](5:7) = proj_5^{(5)}\left(\mathcal{I}[\![Max_0]\!](1, \varepsilon, [], 5:7, \varepsilon)\right) = 7$$

AUFGABE 2 – TEIL (A)

```
1  #include <stdio.h>
2
3  int main() {
4    int x1, x2;
5    scanf("%i", &x1);
6    scanf("%i", &x2);
7    while (x1 > 0){
8      x1 = x2 - x1;
9      if (x2 > x1)
10       x2 = x2 / 2;
11   }
12   printf("%d", x1);
13   return 0;
14 }
```

Übersetzen Sie das Programm mittels trans in AM_0-Code mit linearen Adressen. Geben Sie nur das Endergebnis der Übersetzung (keine Zwischenschritte) an!

AUFGABE 2 – TEIL (A)

```
1  READ 1;
2  READ 2;
3  LOAD 1;
4  LIT 0;
5  GT;
6  JMC 20;
7  LOAD 2;
8  LOAD 1;
9  SUB;
10 STORE 1;
11 LOAD 2;
12 LOAD 1;
13 GT;
14 JMC 19;
15 LOAD 2;
16 LIT 2;
17 DIV;
18 STORE 2;
19 JMP 3;
20 WRITE 1;
```

AUFGABE 2 – TEIL (B)

```
3  LOAD 2;
4  LIT 5;
5  LT;
6  JMC 14;
7  LOAD 1;
8  LOAD 2;
9  LIT 2;
10 MUL;
11 ADD;
12 STORE 2;
13 JMP 3;
14 WRITE 1;
```

Erstellen Sie ein Ablaufprotokoll für dieses Programmfragment, bis die AM_0 terminiert. Die Startkonfiguration ist $(7, \varepsilon, [1/3, 2/1], \varepsilon, \varepsilon)$.

AUFGABE 2 – TEIL (B)

Ablauf der abstrakten Maschine:

BZ	DK	HS	Inp	Out
7	ε	[1/3, 2/1]	ε	ε
8	3	[1/3, 2/1]	ε	ε
9	1:3	[1/3, 2/1]	ε	ε
10	2:1:3	[1/3, 2/1]	ε	ε
11	2:3	[1/3, 2/1]	ε	ε
12	5	[1/3, 2/1]	ε	ε
13	ε	[1/3, 2/5]	ε	ε
3	ε	[1/3, 2/5]	ε	ε
4	5	[1/3, 2/5]	ε	ε
5	5:5	[1/3, 2/5]	ε	ε
6	0	[1/3, 2/5]	ε	ε
14	ε	[1/3, 2/5]	ε	ε
15	ε	[1/3, 2/5]	ε	3