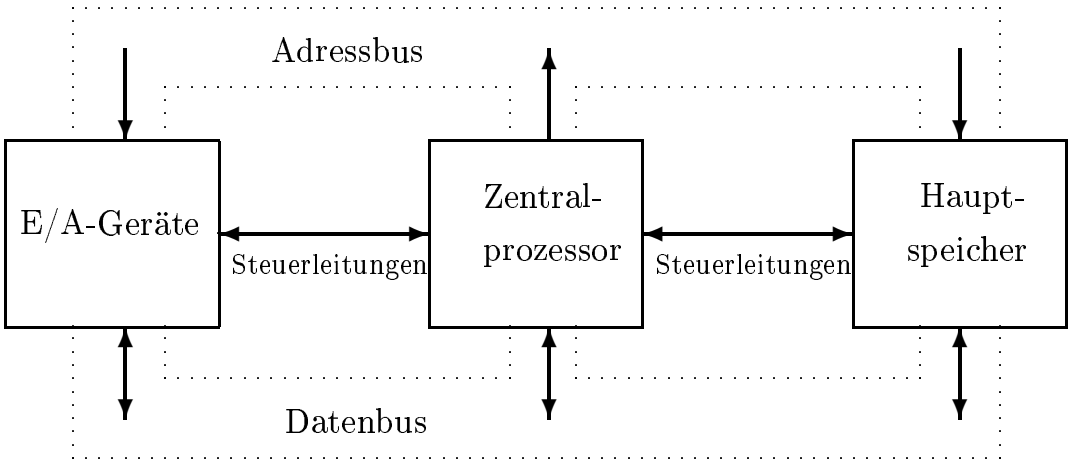
Informatik II für Verkehrsingenieure
AM_0 (Kapitel 14.1 + 14.2)

Janis Voigtländer

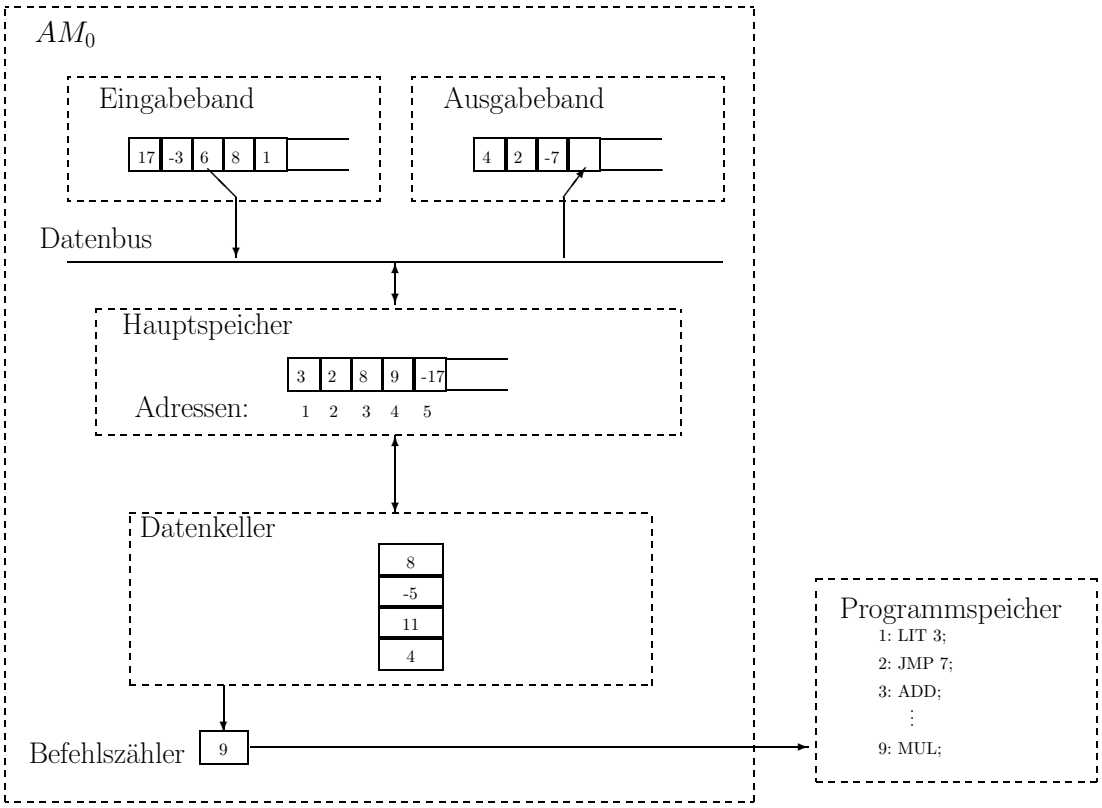
Technische Universität Dresden

Sommersemester 2007

Von-Neumann-Rechner



Abstrakte Maschine AM_0



Modellierung von C-Features

```
int main()
{ int i,n,s;
  scanf("%i",&n);
  i=1;
  s=0;
  while (i<=n)
   { s=s+i*i;
     i=i+1;
   }
  printf("%d",s);
  return 0;
}
```

Wir brauchen: Speicherplätze, Ein- und Ausgabe, Auswertung von Ausdrücken, Kontrollfluss

Modellierung des Hauptspeichers

- statt benannter Speicherplätze (Variablen) lediglich numerische Adressen
- einzig möglicher Speicherinhalt: ganze Zahl (`int`)
- konkreter Hauptspeicherinhalt modelliert als partielle Abbildung von Adressen auf Inhalte
- mathematisch: HS $= \{h \mid h : \mathbb{N} \rightharpoonup \mathbb{Z}\}$
- Notation: $h = [1/3,2/2,3/5]$
- Update: $h[n/d]$;
  $h[n/d](n') = d$, falls $n' = n$, sonst $h[n/d](n') = h(n')$
- Beispiele für $h = [1/3,2/2,3/5]$:
  $h[5/7] = [1/3,2/2,3/5,5/7]$ und
  $h[2/7] = [1/3,2/7,3/5]$
- Befehle zur Kommunikation mit:
  - Ein- und Ausgabe
  - „Berechnungseinheit“

Modellierung von Ein- und Ausgabe

- lediglich Ein- und Ausgabe ganzer Zahlen (`int`)
- Abstraktion von interaktiver Ein- und Ausgabe;
  stattdessen: Eingabeband und Ausgabeband
- Bandinhalte modelliert als endliche Listen
- mathematisch: $\underline{\text{Inp}} = \underline{\text{Out}} = \mathbb{Z}^*$
- Notation: $inp = 1.13.5$ oder $out = \varepsilon$
- READ n: Einlesen erster Position von inp und
  entsprechendes Update der Position n des HS
- WRITE n: Ausgabe des HS-Inhalts an Position n
  am Ende von out

Auswertung von Ausdrücken

Problem:

- Wir wollen strukturierte Ausdrücke wie `(3+5)*(7-2)` oder `j>2*i+1` auswerten.
- Dies soll jedoch in „linearisierter Form“ erfolgen, insbesondere eine Operation immer nur auf zwei Operanden wirken.
- Folglich müssen Zwischenergebnisse berechnet und in geeigneter Weise vorrätig gehalten werden.

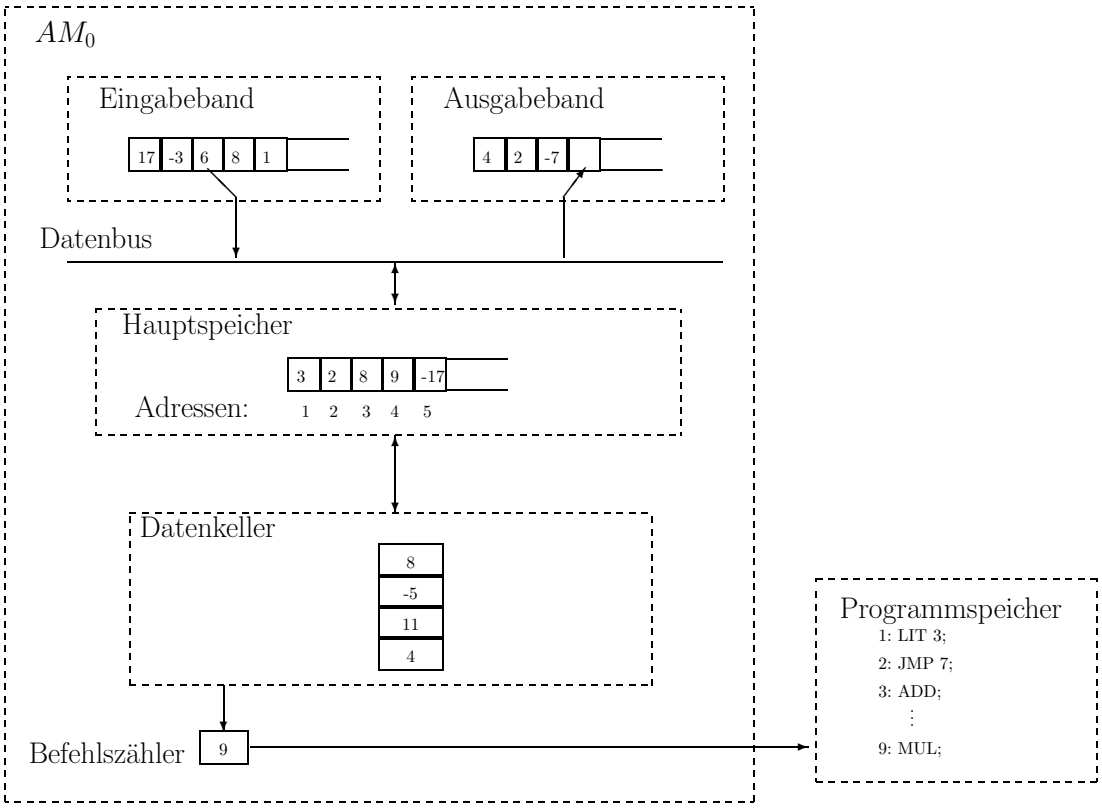
Lösung:

- Einführung eines „Datenkellers“
- Ablage und Entnahme jeweils nur an Kellerspitze
- mathematisch: DK = $\mathbb{Z}^*$
- Notation: $d = 8{:}7{:}2$
- LIT z: Ablage einer Konstante
- LOAD n: Ablage des HS-Inhalts an Position n
- ADD, MUL, ..., EQ, NE ,LT, ...: Berechnungen
- STORE n: Entnahme und entsprechendes Update der Position n des HS

Kontrollfluss — Abarbeitungsreihenfolge

- Durchnummerierung aller Befehle in einem Programm
- aktuelle Position im Befehlszähler gespeichert ($m \in \mathrm{BZ} = \mathbb{N}$)
- normalerweise einfache Erhöhung nach jeder Befehlsabarbeitung
- JMP n: Sprung an Position n
- JMC n: bedingter Sprung an Position n; nur wenn Spitze des Datenkellers gleich 0

Abstrakte Maschine AM_0



Formale Modellierung

- $\mathsf{AM}_0 = \mathsf{BZ} \times \mathsf{DK} \times \mathsf{HS} \times \underline{\mathsf{Inp}} \times \underline{\mathsf{Out}}$ mit:

BZ	$= \mathbb{N}$	Befehlszähler
DK	$= \mathbb{Z}^*$	Datenkeller
HS	$= \{h \mid h : \mathbb{N} \rightharpoonup \mathbb{Z}\}$	Hauptspeicher
$\underline{\mathsf{Inp}}$	$= \mathbb{Z}^*$	Eingabeband
$\underline{\mathsf{Out}}$	$= \mathbb{Z}^*$	Ausgabeband

- Maschine ist jederzeit in einem Zustand $s = (m, d, h, inp, out) \in \mathsf{AM}_0$
- Ein Programm ist eine partielle, endlich definierte Abbildung P von $\mathbb{N}$ auf die Menge Γ aller Befehle.
- Notation: 1: P(1);
  2: P(2);
  3: P(3);
  ...
- Programmabarbeitung besteht aus wiederholter Anwendung von Befehlen auf einen Zustand.

Befehlssemantik (I)

$\mathcal{C}[\![.]\!]\colon \Gamma \longrightarrow (\mathsf{AM}_0 \dashrightarrow \mathsf{AM}_0)$

$\mathcal{C}[\![\mathsf{READ}\ n]\!](m, d, h, inp, out) =$
wenn $inp = first(inp).rest(inp)$ mit $first(inp) \in \mathbb{Z}, rest(inp) \in \mathbb{Z}^*$,
dann $(m+1, d, h[n/first(inp)], rest(inp), out)$

$\mathcal{C}[\![\mathsf{WRITE}\ n]\!](m, d, h, inp, out) =$
wenn $h(n) \in \mathbb{Z}$, dann $(m+1, d, h, inp, out.h(n))$

$\mathcal{C}[\![\mathsf{LOAD}\ n]\!](m, d, h, inp, out) =$
wenn $h(n) \in \mathbb{Z}$, dann $(m+1, h(n) : d, h, inp, out)$

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

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

Befehlssemantik (II)

$\mathcal{C}[\![\mathsf{ADD}]\!](m, d, h, inp, out) =$
wenn $d = d.1 : d.2 : d'$, dann $(m+1, (d.2 + d.1) : d', h, inp, out)$

für MUL, SUB, DIV und MOD analog

$\mathcal{C}[\![\mathsf{LT}]\!](m, d, h, inp, out) =$
wenn $d = d.1 : d.2 : d'$, dann $(m+1, b : d', h, inp, out)$,
wobei $b = 1$, falls $d.2 < d.1$, sonst $b = 0$

für EQ, NE, GT, LE und GE analog

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

$\mathcal{C}[\![\mathsf{JMC}\ e]\!](m, d, h, inp, out) =$
wenn $d = 0 : d'$, dann (e, d', h, inp, out);
wenn $d = 1 : d'$, dann $(m+1, d', h, inp, out)$

Programmsemantik — am Beispiel

```
 1: READ 2;        8: LE;          15: STORE 3;
 2: LIT 1;         9: JMC 21;      16: LOAD 1;
 3: STORE 1;      10: LOAD 3;      17: LIT 1;
 4: LIT 0;        11: LOAD 1;      18: ADD;
 5: STORE 3;      12: LOAD 1;      19: STORE 1;
 6: LOAD 1;       13: MUL;         20: JMP 6;
 7: LOAD 2;       14: ADD;         21: WRITE 3;
```

(1 , ε , [] , 2 , ε)

Programmsemantik — am Beispiel

```
 1: READ 2;        8: LE;          15: STORE 3;
 2: LIT 1;         9: JMC 21;      16: LOAD 1;
 3: STORE 1;      10: LOAD 3;      17: LIT 1;
 4: LIT 0;        11: LOAD 1;      18: ADD;
 5: STORE 3;      12: LOAD 1;      19: STORE 1;
 6: LOAD 1;       13: MUL;         20: JMP 6;
 7: LOAD 2;       14: ADD;         21: WRITE 3;
```

(7 , 3 , [1/3,2/2,3/5] , ε , ε)
(8 , 2:3 , [1/3,2/2,3/5] , ε , ε)
(9 , 0 , [1/3,2/2,3/5] , ε , ε)
(21 , ε , [1/3,2/2,3/5] , ε , ε)
(22 , ε , [1/3,2/2,3/5] , ε , 5)

Wiederholung — AM_0

$AM_0 = BZ \times DK \times HS \times \underline{Inp} \times \underline{Out}$ mit:

BZ	$= \mathbb{N}$	Befehlszähler
DK	$= \mathbb{Z}^*$	Datenkeller
HS	$= \{h \mid h : \mathbb{N} \dashrightarrow \mathbb{Z}\}$	Hauptspeicher
$\underline{Inp}$	$= \mathbb{Z}^*$	Eingabeband
$\underline{Out}$	$= \mathbb{Z}^*$	Ausgabeband

- READ n: Lesen von Eingabeband in Hauptspeicher
- WRITE n: Ausgabe aus Hauptspeicher auf Ausgabeband
- LOAD n: Ablage aus Hauptspeicher auf Datenkeller
- STORE n: Entnahme aus Datenkeller in Hauptspeicher
- LIT z: Ablage einer Konstante auf Datenkeller
- ADD, MUL, SUB, DIV, MOD, LT, EQ, NE, GT, LE, GE: Berechnungen und Vergleiche (auf Datenkeller)
- JMP n: Sprung
- JMC n: Sprung abhängig von Datenkeller

Programmsemantik — formal

(zur Erinnerung: Befehlssemantik $\mathcal{C}[\![.]\!]: \Gamma \longrightarrow (AM_0 \dashrightarrow AM_0)$)

Sei $Prog_0$ die Menge aller Programme.

$\mathcal{I}[\![.]\!] : Prog_0 \longrightarrow (AM_0 \dashrightarrow AM_0)$

$$\mathcal{I}[\![P]\!](m, d, h, inp, out) = \begin{cases} \mathcal{I}[\![P]\!](\mathcal{C}[\![P(m)]\!](m, d, h, inp, out)), & \text{falls } m \in def(P) \\ (m, d, h, inp, out), & \text{falls } m \notin def(P) \end{cases}$$

$\mathcal{P}[\![.]\!] : Prog_0 \longrightarrow (\underline{Inp} \dashrightarrow \underline{Out})$

$\mathcal{P}[\![P]\!](inp) = proj_5^{(5)}(\mathcal{I}[\![P]\!](1, \varepsilon, [], inp, \varepsilon))$

Weiteres Beispiel

Aufgabe:
- Einlesen einer Folge (Ende: 0) vom Eingabeband
- Ausgabe der Gesamtsumme auf Ausgabeband

```
1: LIT 0;        6: NE;        11: STORE 2;
2: STORE 2;      7: JMC 13;    12: JMP 3;
3: READ 1;       8: LOAD 2;    13: WRITE 2;
4: LOAD 1;       9: LOAD 1;
5: LIT 0;       10: ADD;
```

(1 , ε , [] , 3.4.2.0 , ε)

Weiteres Beispiel

Aufgabe:
- Einlesen einer Folge (Ende: 0) vom Eingabeband
- Ausgabe der Gesamtsumme auf Ausgabeband

```
1: LIT 0;        6: NE;        11: STORE 2;
2: STORE 2;      7: JMC 13;    12: JMP 3;
3: READ 1;       8: LOAD 2;    13: WRITE 2;
4: LOAD 1;       9: LOAD 1;
5: LIT 0;       10: ADD;
```

(12 , ε , [1/2,2/9] , 0 , ε)
(3 , ε , [1/2,2/9] , 0 , ε)
(4 , ε , [1/0,2/9] , ε , ε)
(5 , 0 , [1/0,2/9] , ε , ε)
(6 , 0:0 , [1/0,2/9] , ε , ε)
(7 , 0 , [1/0,2/9] , ε , ε)
(13 , ε , [1/0,2/9] , ε , ε)
(14 , ε , [1/0,2/9] , ε , 9)

Optimierung

Idee: statt Ablage im HS, Zwischensumme auf DK lassen

1: LIT 0;	5: NE;	9: JMP 2;
2: READ 1;	6: JMP 10;	10: STORE 1;
3: LOAD 1;	7: LOAD 1;	11: WRITE 1;
4: LIT 0;	8: ADD;	

(1 , ε , [] , 3.4.2.0 , ε)

Optimierung

Idee: statt Ablage im HS, Zwischensumme auf DK lassen

1: LIT 0;	5: NE;	9: JMP 2;
2: READ 1;	6: JMP 10;	10: STORE 1;
3: LOAD 1;	7: LOAD 1;	11: WRITE 1;
4: LIT 0;	8: ADD;	

(8 , 2:7 , [1/2] , 0 , ε)
(9 , 9 , [1/2] , 0 , ε)
(2 , 9 , [1/2] , 0 , ε)
(3 , 9 , [1/0] , ε , ε)
(4 , 0:9 , [1/0] , ε , ε)
(5 , 0:0:9 , [1/0] , ε , ε)
(6 , 0:9 , [1/0] , ε , ε)
(10 , 9 , [1/0] , ε , ε)
(11 , ε , [1/9] , ε , ε)
(12 , ε , [1/9] , ε , 9)

Erweiterung

Aufgabe: ▶ statt Summe, nun Durchschnitt zu berechnen

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

(1 , ε , [] , 3.4.2.0 , ε)

Erweiterung

Aufgabe: ▶ statt Summe, nun Durchschnitt zu berechnen

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

```
( 7 , 0:0:9 , [1/0,2/3] , ε , ε )
( 8 , 0:9 , [1/0,2/3] , ε , ε )
( 16 , 9 , [1/0,2/3] , ε , ε )
( 17 , 3:9 , [1/0,2/3] , ε , ε )
( 18 , 3 , [1/0,2/3] , ε , ε )
( 19 , ε , [1/3,2/3] , ε , ε )
( 20 , ε , [1/3,2/3] , ε , 3 )
```