Kapitel 13
ReTI II - Ein einfacher Rechner Grobentwurf

ReTI-II = Rechner Technische Informatik II

Vorgehen Kap. 13 und folgende

- 13.1 Abstrakte RE-TI-II-Maschine
- 13.2 Übergang zur realen Maschine
- 13.3 Spezifikation der Datenpfade zur Befehlsdurchführung

- Kap. 14 Timing ReTI-II
  - grobe zeitliche Planung (idealisierte Timing-Diagramme)
  - Spezifikation und Realisierung der Kontroll-Logik
  - Detaillierte Timing-Analyse → Zykluszeit
- Kap. 15 Ein/Ausgabe ReTI-II

Vorgehen Kap. 13

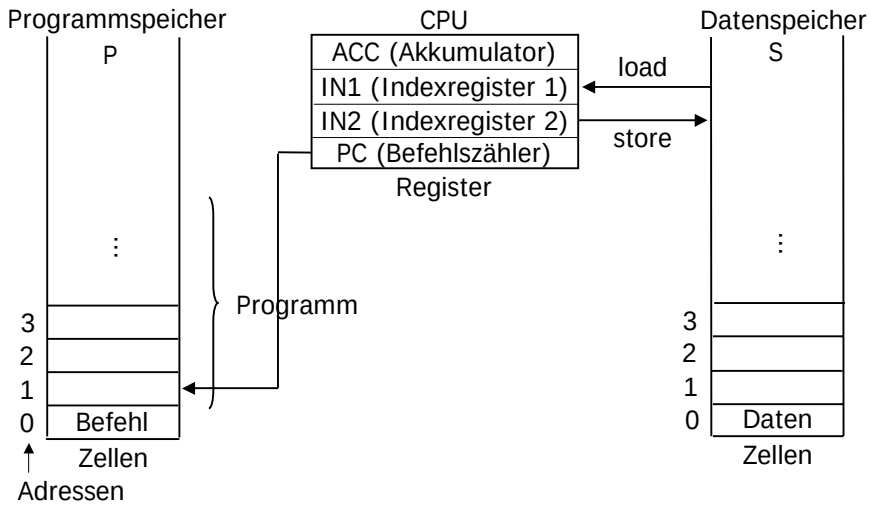
- 13.1 Abstrakte RE-TI-II-Maschine
  - 2 unendlich große Speicher
  - Datenspeicher für Daten
  - Programmspeicher für Programme
  - Speicherzellen des Datenspeichers für beliebig große ganze Zahlen
  - Speicherzellen des Programmspeichers für Befehle
    (Befehlsparameter = beliebig große ganze Zahlen)
  - Zentraleinheit zum Ausführen von Operationen

Vorgehen Kap. 13

- 13.2 Übergang zur realen Maschine
  - alles endlich
  - Festlegung von Formaten / Kodierung für Daten und Befehle
- 13.3 Datenpfade zur Befehlsdurchführung

13.1
Die abstrakte RE-TI-II-Maschine

Aufbau der RE-TI-II



Aufbau (ff)

3 Teile:

- Zentraleinheit = CPU (central processing unit)
- Programmspeicher P
- Datenspeicher S

Speicher bestehen aus unendlich vielen Zellen, nummeriert durch *Adressen* $i \in \mathbf{N}_0$

P(i) = Inhalt von Zelle i des Programmspeichers

S(i) = Inhalt von Zelle i des Datenspeichers

Aufbau (ff)

CPU mit 4 für Benutzer sichtbaren Registern:

- PC = Befehlszähler (program counter)
- ACC = Akkumulator
- IN1, IN2 = Indexregister 1, 2

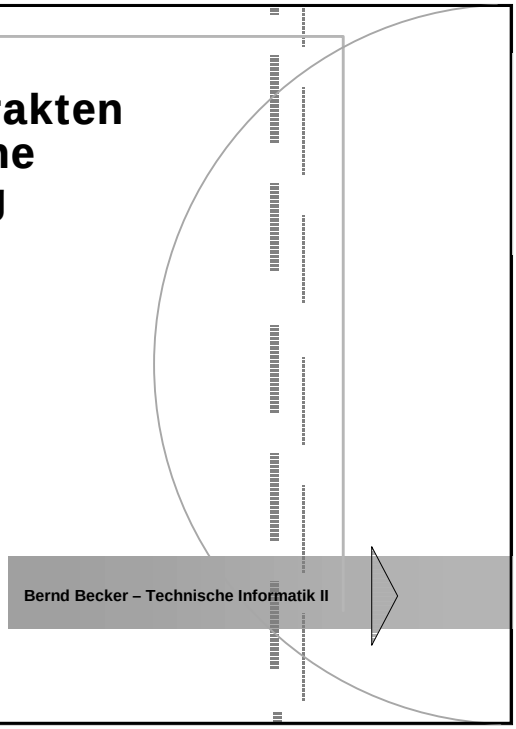
Programmablauf

- Programme bzw. Daten stehen beim Start der Maschine in P bzw. S
- Programm beginnt bei Zelle 0 von P
- Inhalt von P wird nicht geändert
- Maschine arbeitet in Schritten t = 1, 2, ...

  In jedem Schritt t:
  - Ausführung eines Befehls: P(PC) wird als Befehl interpretiert und in Schritt t ausgeführt
  - PC erhält neuen Wert (abhängig von Befehl)

  Bei Programmstart: PC = 0

Befehle der abstrakten RE-TI-II-Maschine und ihre Wirkung



(a) Load / Store

Transport von Daten zwischen ACC und Datenspeicher:

LOAD i :
Lädt Inhalt S(i) von Speicherzelle i in Akkumulator ACC
und erhöht PC um 1
STORE i :
Speichert den Inhalt von ACC in S(i)
und erhöht PC um 1

(a) Load / Store (ff)

Befehl	Wirkung	
LOAD i	ACC := S(i)	PC := PC + 1
STORE i	S(i) := ACC	PC := PC + 1

Beispiel:

Programm, das Inhalte von Speicherzelle S(0) (=x) und S(1) (=y) vertauscht:

Programmspeicherzelle 0 — Kommentar

```
0    LOAD 0 ;     ACC := S(0) = x
1    STORE 2 ;    S(2) := ACC = x
2    LOAD 1 ;     ACC := S(1) = y
3    STORE 0 ;    S(0) := ACC = y
4    LOAD 2 ;     ACC := S(2) = x
5    STORE 1 ;    S(1) := ACC = x
```


(b) Compute - Befehle

Verknüpfe Inhalt von ACC mit S(i) und speichere das Ergebnis in ACC

Befehl	Wirkung	
ADD i	ACC := ACC + S(i)	PC := PC + 1
SUB i	ACC := ACC – S(i)	PC := PC + 1

(c) Immediate - Befehle

Interpretiere Parameter i direkt als Konstante

Befehl	Wirkung	
LOADI i	ACC := i	PC := PC + 1
ADDI i	ACC := ACC + i	PC := PC + 1
SUBI i	ACC := ACC – i	PC := PC + 1

ADD, SUB = Compute *memory* – Befehle

ADDI, SUBI = Compute *immediate* – Befehle

Beides zusammen ergibt die Compute - Befehle

(d) Indexregister - Befehle

Befehl	Wirkung	
LOADINj i	ACC := S(INj + i)	PC := PC + 1 (j ∈ {1,2})
STOREINj i	S(INj + i) := ACC	PC := PC + 1 (j ∈ {1,2})
MOVE S D	D := S	PC := PC + 1 (D ∈ {ACC, IN1, IN2} S ∈ {ACC, IN1, IN2})
MOVE S PC	PC := S	(S ∈ {ACC, IN1, IN2})

Beispiel:

S(0) = x, S(1) = y

Kopiere y in Zelle S(x):

```
0   LOAD 0 ;          ACC := S(0) = x
1   MOVE ACC IN1 ;    IN1 := ACC = x
2   LOAD 1 ;          ACC := S(1) = y
3   STOREIN1 0;       S(x) = S(IN1 + 0) := ACC = y
```

(e) Sprung - Befehle

Manipulation des Befehlszählers

JUMP für unbedingte Sprünge,
JUMP$_c$ mit c ∈ { <, =, >, ≤, ≠, ≥ } für bedingte Sprünge

Mit bedingten Sprüngen kann man *Programmschleifen* und *bedingte Anweisungen* realisieren!

(e) Sprung - Befehle (ff)

Befehl	Wirkung
JUMP i	PC := PC + i (i ∈ **Z**)
JUMP$_c$ i	PC := { PC + i, falls ACC c 0 PC + 1, sonst (i ∈ **Z**, c ∈ {<, =, >, ≤, ≠, ≥})