

Kapitel 1 – Grundlagen

1. Mathematische Grundlagen

2. **Beispielrechner ReTI**

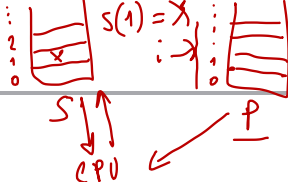
Albert-Ludwigs-Universität Freiburg

Prof. Dr. Christoph Scholl

Institut für Informatik

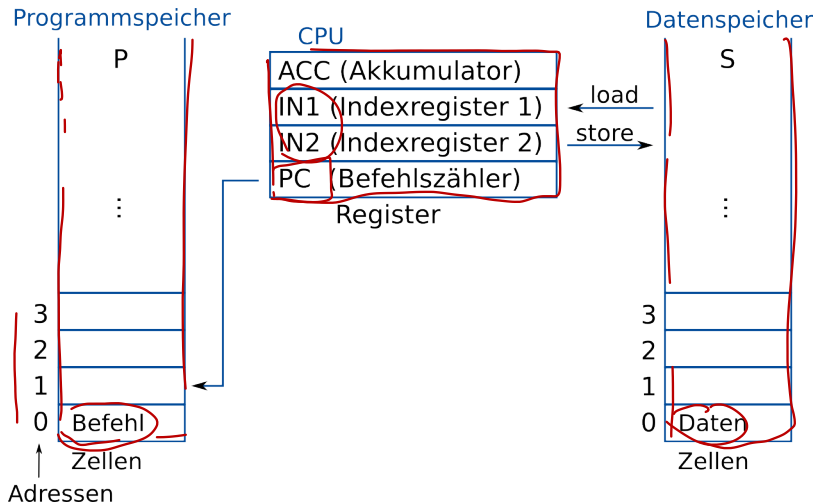
WS 2015/16

- Ursprünglich eingeführt in [Keller, Paul] unter dem Namen ReSa.
- Hier wird ReTI zunächst abstrakt eingeführt.
 - Alle Speicher bestehen aus unendlich vielen Speicherzellen, die beliebig große ganze Zahlen aufnehmen können.
- Später wird die tatsächliche Implementierung von ReTI unter realistischen Annahmen thematisiert.



- Zwei unendlich große Speicher
 - **Datenspeicher S** für Daten (beliebig große Zahlen).
 - $S(i)$ = Inhalt von Zelle i des Datenspeichers, $i \in \mathbb{N}$ **Adresse**.
 - **Programmspeicher P** für Maschinenbefehle.
 - Lade-/Speicher-, Rechen, Sprungbefehle - siehe später.
 - $P(i)$ = Inhalt von Zelle i des Programmspeichers.
- **Zentraleinheit CPU** (Central Processing Unit)
 - Vier für Benutzer sichtbare Register.
 - PC = Befehlszähler (Program Counter). *An Anfang $PC = 0$*
 - ACC = Akkumulator. *$A+B=C$
ARC wo auch \downarrow ACC*
 - $IN1$, $IN2$ = Indexregister 1 und 2.

Aufbau von ReTI



Programmablauf

- Programme bzw. Daten stehen beim Start der Maschine in P bzw. S . ANNAHME

- Programm beginnt bei Zelle 0 von P .

- Inhalt von P wird nicht geändert.

- Maschine arbeitet in Schritten $t = 1, 2, \dots$

In jedem Schritt t :

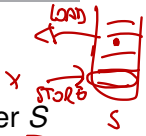

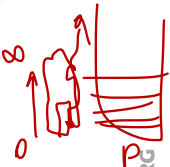
- Ausführung eines Befehls: $P(PC)$ wird als Befehl interpretiert und in Schritt t ausgeführt.

- PC erhält neuen Wert (abhängig von Befehl). $PC := PC + 1$

- Bei Programmstart ist $PC = 0$.



ReTI-Befehle und ihre Wirkung

- 1 ■ **Load/Store**: Laden von Werten aus dem Datenspeicher S bzw. Schreiben von Werten in S.

- 2 ■ **Compute**: Berechnungen (hier zunächst Addition und Subtraktion).
 - Mit Werten im Datenspeicher S. 
 - Mit Absolutwerten (Immediate).
- 3 ■ **Indexregister**: Indirekte Speicheradressierung (siehe unten).
- 4 ■ **Sprungbefehle**: Bedingte und unbedingte Sprünge.


Load/Store



$PC = 2$

$PC = 2 + 1 = \}$

LOAD 2

Transport von Daten zwischen ACC und Datenspeicher.

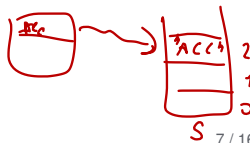
■ LOAD i :

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.

STORE 2



Load/Store: Übersicht

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

ASSEMBLY

LOW-LEVEL PROGRAMMIEREN

Beispielprogramm

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



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;	$S(1) := ACC = x$

$2 \leftarrow 1$

ACC	$S(0)$	$S(1)$	$S(2)$
	x	y	-
x	x	y	-
x	x	y	x
y	x	y	x
y	y	y	x
x	y	y	x
x	y	x	x

$S(0) = y$

$S(1) = x$

Compute-Befehle

Verknüpfe den Inhalt von ACC mit S(i) (oder mit einer Konstante) und speichere das Ergebnis in ACC ab.

- ADD, SUB = Compute *memory*-Befehle
- ADDI, SUBI = Compute *immediate*-Befehle
- Beides zusammen ergibt die **Compute-Befehle**.

Bei Compute memory: Interpretiere Parameter i direkt als Speicheradresse.

Befehl	Wirkung
<u>ADD</u> i	$ACC := \underline{ACC} + \underline{S(i)} \quad \underline{PC} := PC + 1$
<u>SUB</u> i	$ACC := \underline{ACC} - \underline{S(i)} \quad \underline{PC} := PC + 1$

Immediate-Befehle

LOAD i | $ACC := S(i)$ $PC := PC + 1$

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	$ACC := ACC - i$ $PC := PC + 1$

STORE i $i := ACC$

- Anmerkung: **ADDI** und **SUBI** sind Compute Befehle.
LOADI ist den Load-/Store-Befehlen zuzuordnen.

STORE 6 6 := ACC ?? Keinen Sinn!

Indexregister-Befehle

ACC
PC

LOADIN1 i | ACC := S(IN1 + 1) Bsp. LOADIN1 2 →
ACC := S(IN1 + 2)

IN1 = 2
IN2

LOADIN1 i

LOADIN2 i

Befehl	Wirkung = S(4)	
LOADINj i	$ACC := S(INj + i)$ ($j \in \{1, 2\}$)	$PC := PC + 1$
<u>STOREINj i</u>	$S(INj + i) := ACC$ ($j \in \{1, 2\}$)	$PC := PC + 1$
<u>MOVE S D</u>	$D := S$ ($D \in \{ACC, IN1, IN2\}$, $S \in \{ACC, IN1, IN2, PC\}$)	$PC := PC + 1$
<u>MOVE S PC</u>	<u>PC := S</u> ($S \in \{ACC, IN1, IN2\}$)	---



S = Source = Quelle

D = Destination = Ziel

Beispielprogramm für Indexregister-Befehle

$S(0) = x$, $S(1) = y$
Kopiere y in Zelle $S(x)$:

0	LOAD 0;	$ACC := S(0) = x$
1	MOVE ACC IN1;	$IN1 := ACC = x$
2	LOAD <u>1</u> ;	$ACC := S(1) = y$
3	STOREIN1 <u>0</u> ;	$S(x) = S(IN1 + 0) := ACC = y$

Handwritten diagram illustrating memory state and operations:

Registers: ACC, IN1, S(0), S(1)
Values: x, x, x, y
Operations: $S(x) = y$

Sprung-Befehle

Manipulation des Befehlszählers.

- JUMP für *unbedingte* Sprünge,
- JUMP_c mit ^{condition} $c \in \{<, =, >, \leq, \neq, \geq\}$ für *bedingte* Sprünge.
- Mit bedingten Sprüngen kann man *Programmschleifen* und *bedingte Anweisungen* realisieren!

Befehl	Wirkung
<u>JUMP</u> <u>i</u>	$PC := PC + i \quad (i \in \mathbb{Z})$
<u>JUMP_c</u> <u>i</u>	$PC := \begin{cases} PC + i, & \text{falls } ACC \text{ } c \text{ } 0 \\ PC + 1, & \text{sonst} \end{cases}$ $(i \in \mathbb{Z}, c \in \{<, =, >, \leq, \neq, \geq\})$

JUMP 7
wenn $ACC < 0$ dann
 $PC := PC + 7$
sonst
 $PC := PC + 1$

↑
Relative Sprünge

Absolute S. →
MOV S PC

Beispielprogramm

$S(0) = x; S(1) = y, y \geq 0$

Polare Programmiersprache:

prod
b
a

0 x	S(2)
2 y-1	S(1)
x	S(0)

prod := 0	0	LOADI 0;	ACC := 0	S(2) wird auf 0 gesetzt
	1	STORE 2;	S(2) := 0	
b := b - 1	2	LOAD 1;	ACC := S(1)	S(1) wird um 1 verringert
	3	SUBI 1;	ACC := ACC - 1	
	4	STORE 1;	S(1) := ACC	
	5	JUMP < 5;	PC := PC + 5, falls ACC < 0	
while (b > 0) do {	6	LOAD 2;	ACC := S(2)	(sonst PC := PC + 1, wie immer)
	7	ADD 0;	ACC := ACC + S(0)	
prod := prod + a;	8	STORE 2;	S(2) := ACC	S(2) wird um x (S(0)) erhöht
	9	JUMP -7;	PC := PC - 7	
b := b - 1;	10	JUMP 0		
}				

Das Programm rechnet so lange bis in S(2) x · y steht und bleibt dann in P(10).

- Mathematik erlaubt es uns, reale Zusammenhänge formal zu fassen und allgemeingültige Folgerungen aus ihnen herzuleiten.
- Rechner ReTI wird uns im weiteren Verlauf der Vorlesung als Illustrator und Anwendungsbeispiel für die vorgestellten Konzepte dienen.

