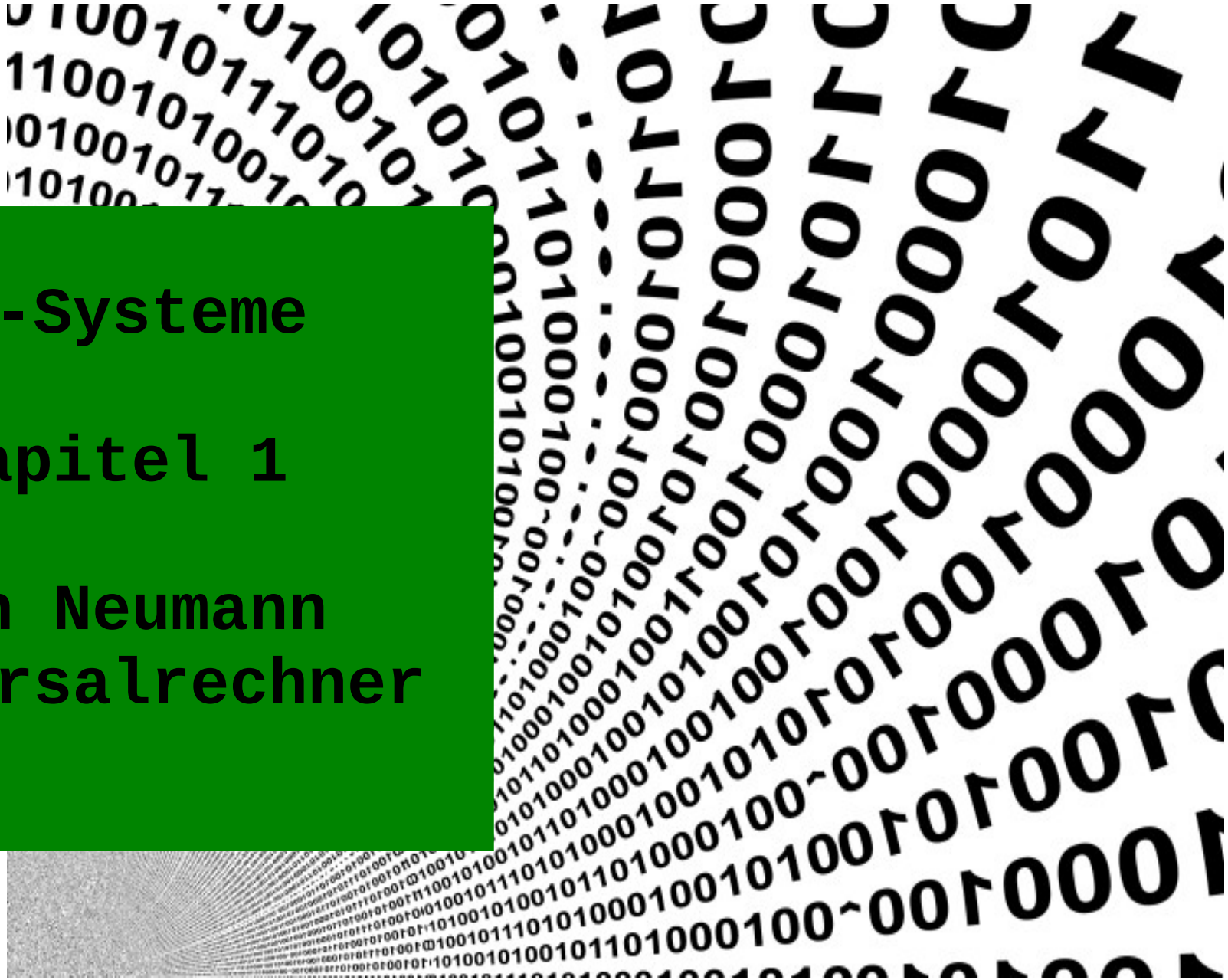


DS-Systeme

Kapitel 1

von Neumann Universalrechner

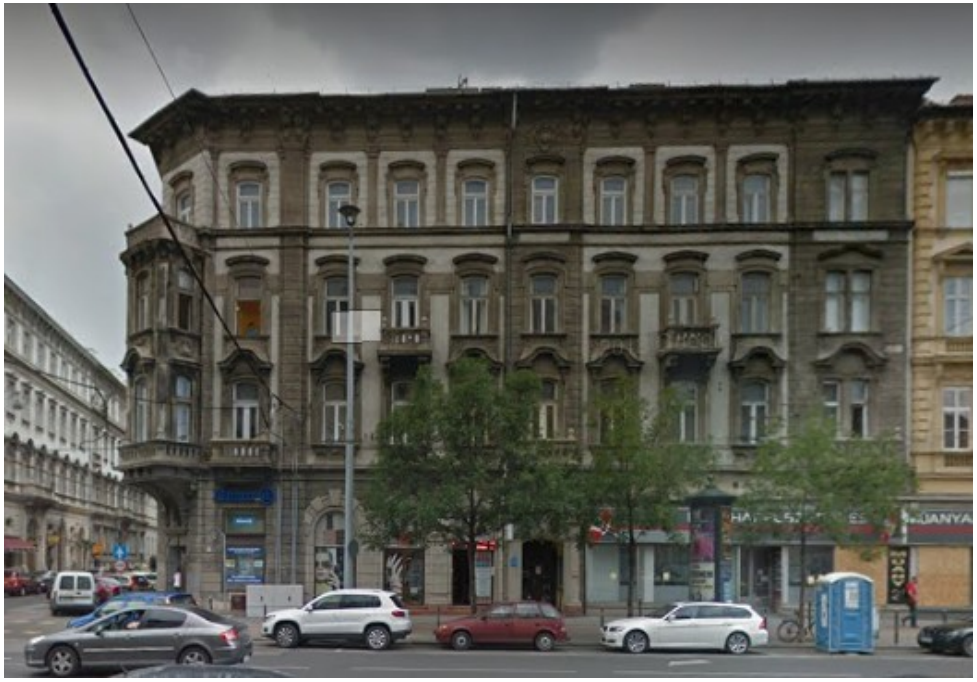
Inhaltsverzeichnis

Einführung in die

Von-Neumann-Architektur

John von Neumann

Institute for Advanced Study,
Princeton, (N.J.).



Geburtshaus in Budapest / Ungarn

János Lajos Neumann von Margitta
(genannt **John von Neumann**)

geb. 28.12.1903 in Budapest, Österreich-Ungarn
gest. 08.02.1957 in Washington, D.C.

Vater Jurist und Bankier, Mutter Hausfrau

1914 Besuch des Gymnasiums
dort bereits als mathematisches Wunderkind angesehen

1921 Studium der Chemiewissenschaften in Berlin
1923 Wechsel zur ETH Zürich
gleichzeitig Studium der Mathematik in Budapest

1925
- Doktorarbeit: „Axiomatisierung der Mengenlehre“
  Doktortitel im Alter von 22 Jahren
- Nach dem Studium Erhalt des Rockefeller-Stipendiums
- Dozent an der Friedrich-Wilhelms-Universtät in Berlin
- Studium und Zusammenarbeit bei und mit *David Hilbert*, Max Born und *Lothar Nordheim* in Göttingen
- Verfassung der Schrift „Die Grundlagen der Quantenmechanik“

1933 Professor für Mathematik am *Institute for Advanced Study* der Universität Princeton / New Jersey. Einer seiner Kollegen dort war Albert Einstein.

1940 Mitglied des wissenschaftlichen Beraterstabs des *Ballistic Research Laboratory*, einer Versuchsanlage der Armee in Aberdeen Proving Ground in Maryland

1943 Beratender Mathematiker beim Manhattan-Projekt

1944 Installation eines IBM-Rechners auf Los Alamos.
Bau der ENIAC (Electronic Numerical Integrator and Computer) ohne John von Neuman in Pennsylvania.
Sept. 1944 Beitritt von Neumanns zur ENIAC-Gruppe.

1945 Konzipierung eines eigenen Computers am Institute for Advanced Studies in Princeton
Im 100-seitigen Paper „First Draft of a Report on the EDVAC“ wird zum ersten Mal die **von Neumann-Architektur** veröffentlicht.

1953 Vorsitz des Air Force Strategic Missiles Evaluation Committees

1955 5-Jahres-Vertrag als Commissioner bei der AEC (United States Atomic Energy Commission), einem der höchsten Posten in den USA.

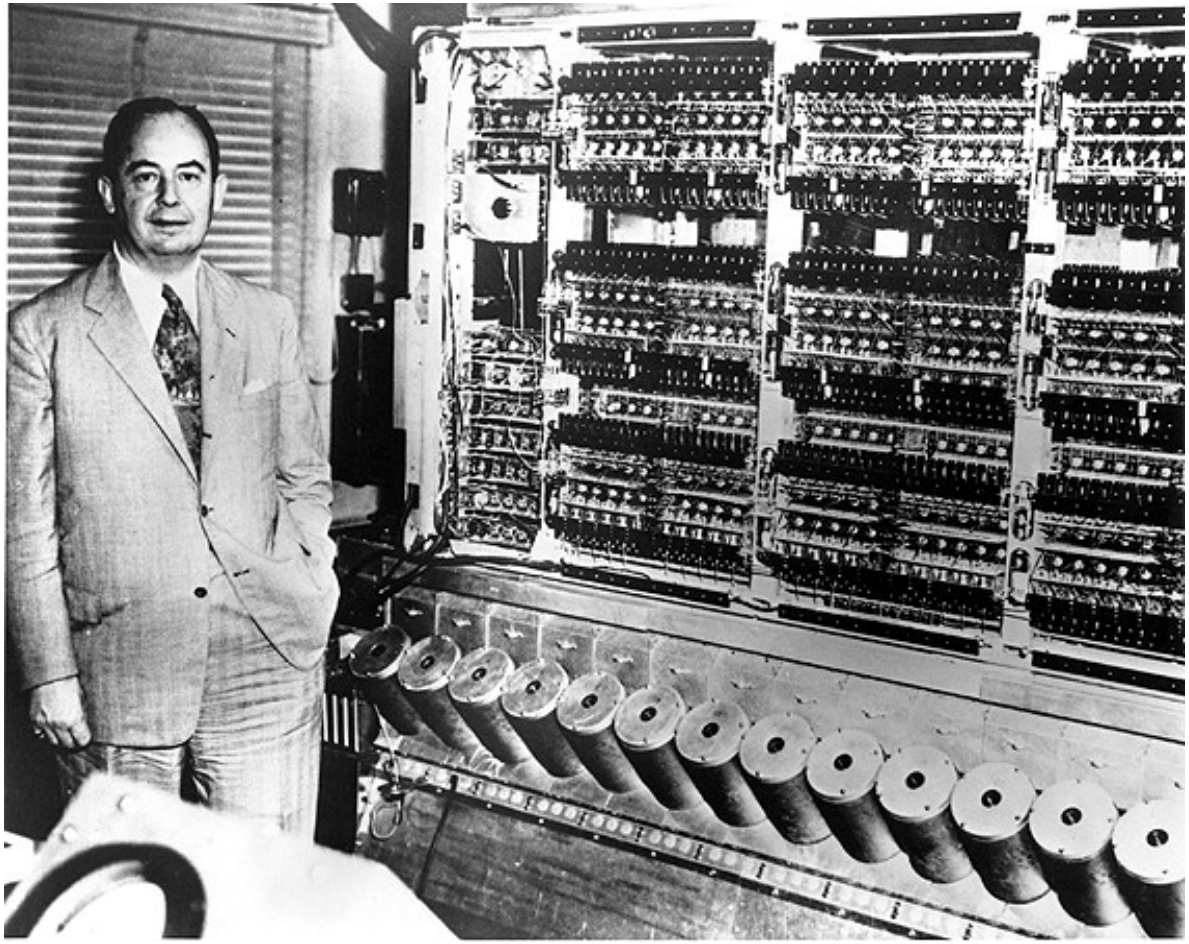
John von Neumann 1946

1956

Erkrankung an Krebs (vermutlich wegen der Teilnahme an einem Atomtest)

1957

Krankenhausaufenthalt bis zu seinem Tod am 8. Februar in Washington D.C.



John von Neumann steht neben der **IAS**-Maschine (**IAS** = **I**nstitute for **A**dvanced **S**tudy), die er auf der Grundlage seiner Beratungsarbeit für den Electronic Discrete Variable Automatic Computer (EDVAC), den ersten Computer mit gespeicherten Programmen in den USA, entwickelt hat.

Die Verdienste von Neumanns beruhen insbesondere auf der Mathematisierung und Verwissenschaftlichung der Rechenmaschinen.

Die Erfindung des ersten programmierbaren Computers

Konrad Ernst Otto Zuse

geb.: 22.6.1910 in Deutsch-Wilmersdorf (Berlin)
gest.: 18.12.1995 in Hünfeld /Landkreis Fulda

Deutscher Erfinder, Bauingenieur und Unternehmer (Zuse AG), Erbauer des ersten funktionsfähigen Computers der Welt (Z3)

Z3 (1941)

Erster funktionstüchtiger, vollautomatischer, frei programmierbarer und in binärer Gleitkommarechnung arbeitender Recher.

Anzahl Relais:	ca. 2500
Taktfrequenz:	5 Hertz
Addition:	ca. 0.8 Sekunden
Multiplikation:	ca. 2 Sekunden

Vergleich mit modernen Computern:

Z3 - Anzahl Relais: ca. 2500

Ein moderner Prozessor kann Milliarden von Transistoren haben. Zum Vergleich: Der erste kommerziell erhältliche Mikroprozessor, der Intel 4004 von 1971, hatte nur 2.300 Transistoren.

Z3 – Taktfrequenz: 5 Hz

Ein moderner Prozessor kann Taktfrequenzen von mehreren Gigahertz (GHz) haben.

Z3 - Addition: ca. 0.8 Sekunden, Multiplikation:ca. 2 Sekunden

Moderne Prozessoren können Hunderte von Milliarden von Gleitkommaoperationen pro Sekunde durchführen.

Z3 – Speicherkapazität: 512 Byte (als Zwischenspeicher, keine Festplatte)

Ein modernes Notebook kann mehrere Terabyte (TB) an Speicherplatz haben.

Z3 – Größe: wie ein kleiner Raum, Gewicht: mehrere Tonnen

Ein modernes Notebook ist wesentlich kleiner und wiegt in der Regel 1.5 - 2.5 kg.

Konzept des Universalrechners

- Alle Daten (Programmbefehle, Adressen, Werte) sind **binär** codiert. Mögliche Zustände innerhalb einer Speicherzelle werden mit 0 oder 1 bezeichnet.

- Programm zur Lösung eines Problems und Daten sind im gleichen Speicher abgelegt.

- Ein Programm besteht aus Arbeitsschritten (= Befehle). Die Befehle eines Programms sind in aufeinanderfolgenden Speicherzellen abgelegt. Über die Nummer (= Adresse) einer Speicherzelle kann deren Inhalt abgerufen oder verändert werden.

- Innerhalb eines Befehls kann höchstens ein Datum bearbeitet, d.h. neu berechnet werden – Prinzip **SISD** (**S**ingle **I**nstruction **S**ingle **D**ata)

  Es gibt folgende Befehlsarten:

  Arithmetische Befehle (+, -, *, /)

  Logische/relationale Befehle (∧, ∨, ¬, <, >)

  Transportbefehle z.B. - vom Speicher zum Rechenwerk
  - für Ein- / Ausgabe

Konzept des Universalrechners

Sprungbefehle zur Abweichung von der vorgesehenen Programmreihenfolge.

Befehle zur Programmablaufsteuerung

- Programmzähler initialisieren (mit Startadresse des Programms)
- Programmende

Übertragung von Daten, Befehlen, Adressen und Kontrollinformationen zwischen den einzelnen Komponenten erfolgt über das Bussystem:

- Datenbus - Bidirektionale* Übertragung von Daten

- Adressbus - Unidirektionale Übertragung von Adressen

- Steuerbus - Uni- bzw. bidirektionale Übertragung von Steuersignalen zwischen Steuerwerk und den übrigen Funktionseinheiten.

  Signalisierung, für welche Funktionseinheiten die jeweils auf Adress- und Datenbus anliegenden Daten bestimmt sind.

  Festlegung der Zugriffsart Schreiben oder Lesen

* Bidirektionale Übertragung bedeutet gleichzeitige Übertragung in beide Richtungen.

Konzept des Universalrechners

Alle Komponenten der CPU werden durch das **Steuerwerk** beeinflusst:

- Laden des nächsten Befehls aus dem Speicher
- Dekodierung der Befehle
- Steuerung der Befehlsausführung
- Laden und Ablage von Daten aus bzw. in den Speicher

Die Struktur des VNR's (von Neumann-Rechner = Universalrechner) war unabhängig von dem zu lösenden Problem, da vorher Programme noch hardwaremäßig verschaltet oder über Lochstreifen schrittweise eingelesen und sofort sequentiell verarbeitet wurden (z.B. Z1 – 1936).

Nachteile der von Neumann – Architektur

- Im Speicher lassen sich Befehle und Daten anhand des Bitmusters nicht unterscheiden.
- Im Speicher lassen sich variable und konstante Daten nicht unterscheiden.
- Bei falscher Adressierung können Speicherinhalte verändert werden, die nicht geändert werden dürfen, wie z.B. Befehle und Konstanten (Eine Bitänderung bei einem Befehl erzeugt einen ganz anderen Befehl!)
- Da Daten und Befehle im Speicher gehalten werden, wird die Verbindung und Datenübertragung zwischen CPU und Speicher über den Systembus zum Von-Neumann-Flaschenhals:
  - Jeglicher Datenverkehr von und zur CPU wird über den internen Bus abgewickelt, dessen Transfergeschwindigkeit langsamer ist als die Verarbeitungsgeschwindigkeit der CPU.

Hardware-Komponenten der von Neumann-Architektur

CPU – **C**entral **P**rocessing **U**nit (Zentraleinheit) = **ALU + Control Unit**

ALU - **A**rithmetik **L**ogical **U**nit (Rechenwerk):
Ausführung arithmetischer (+, -, * , /) und logischer bzw. relationaler Operationen (∧, ∨, ¬, …)

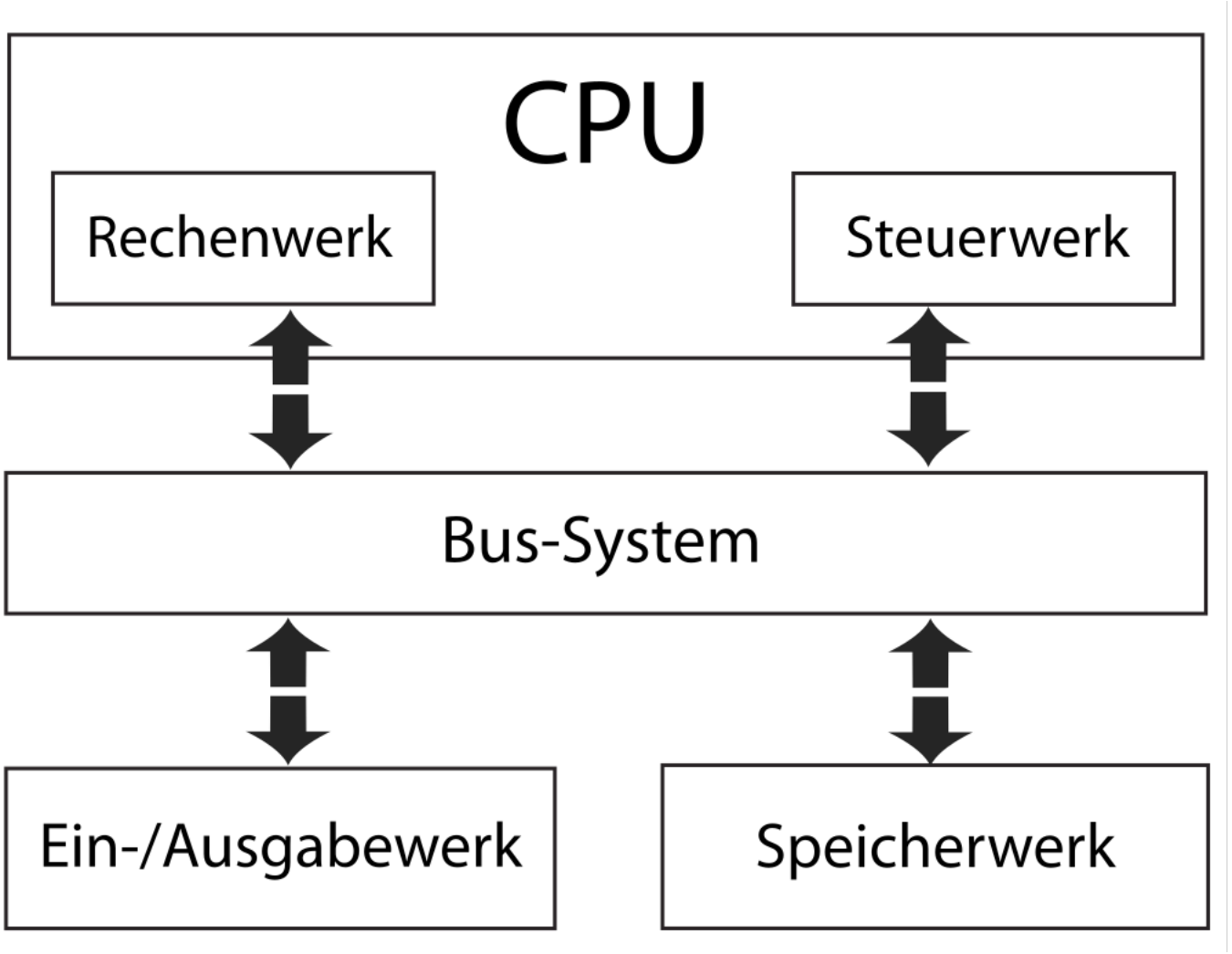
Control Unit – Steuerwerk / Leitwerk
Steuerung des Programmablaufs:
- Interpretiation der Anweisungen eines Programms
- entsprechende Verschaltung von Datenquelle, -senke und notwendigen ALU-Komponenten
- Regelung der Befehlsabfolge

Memory - Speicherwerk:
Speicherung von Programm und Daten

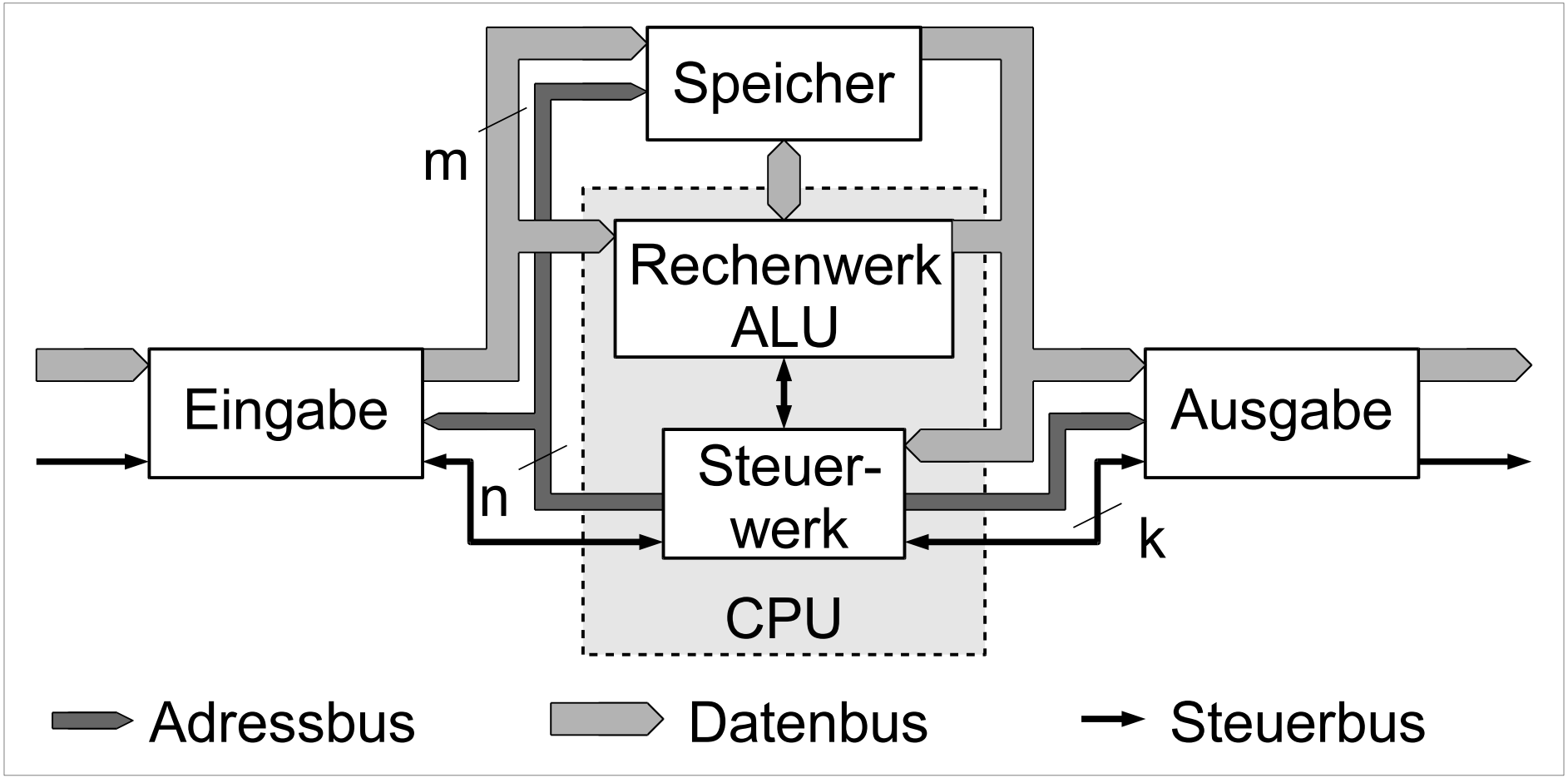
I/O Unit – Ein- / Ausgabewerk
Eingabe von Programmen --> Daten in den Speicher
Ausgabe von Ergebnissen --> Daten vom Speicher nach außen

BUS – Bussystem
dient zur Kommunikation zwischen den einzelnen Komponenten (Steuerbus, Adressbus, Datenbus)

Hardware-Komponenten der von Neumann-Architektur



Schematischer Aufbau eines Von-Neumann-Rechners mit Bussystem



Der Befehlszyklus

Beim Start eines Programms wird der Befehlszähler **PC** (program counter) bzw. **IP** (instruction pointer) in der **CPU** auf die Anfangsadresse des Programms gesetzt.

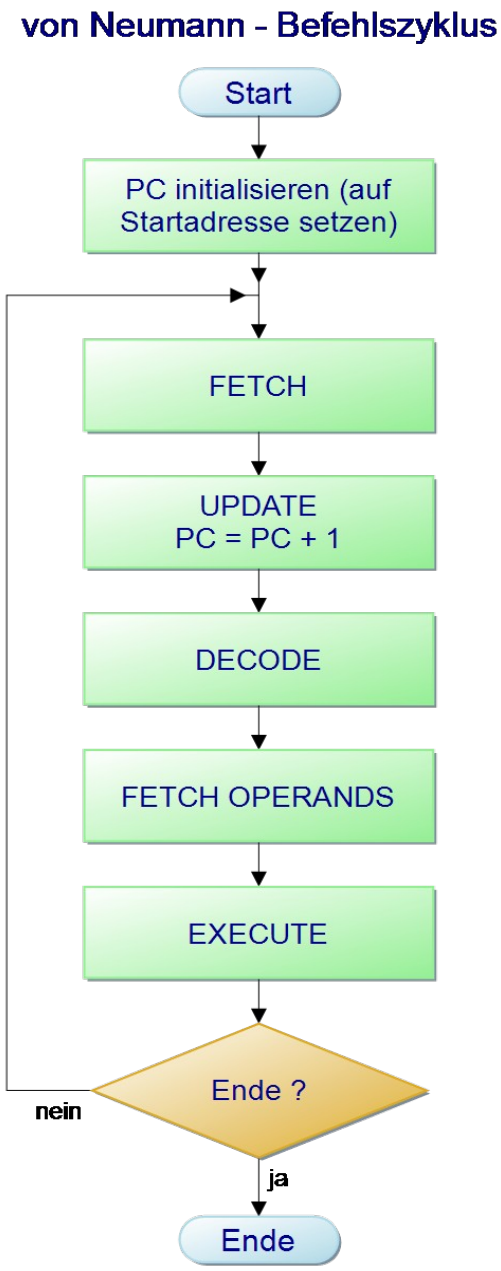
a. Fetch-Phase (Ladephase)
Den durch PC adressierten Befehl ins Befehlsregister laden. Ein Befehl kann aus einem Operations- und einem Adressteil bestehen.

b. Update-Phase
Befehlszähler aktualisieren – in den meisten Fällen durch Inkrementation.

c. Decode-Phase
Entschlüsselung bzw. Zerlegung des Befehls in Operations- und Adressteil im Dekodierer (Teil des Steuerwerks)

d. Fetch Operands
Laden von Operanden aus dem Speicher in das Datenregister (nicht immer Teil des Befehlszyklus, da nicht jeder Befehl Operanden aus dem Speicher benötigt).

e. Execution-Phase (Ausführungsphase)
Ausführung des Befehls



Beispiel:

```c
#include <stdio.h>

int main()
{
    int a = 27;
    int b = 35;

    int c = a + b;

    // printf("%d\n", c);

    return 0;
}
```

Ergebnis:

62

CPU

PC	500
CIR	**LDD** 700
AC	
MAR	500
MDR	**LDD** 700

PC wird mit Adresse 500 initialisiert

1. Zyklus

1. Inhalt vom **PC** wird ins **MAR** kopiert
2. zugehöriger Befehl wird geholt und ins **MDR** kopiert
3. Befehl wird vom **MDR** nach **CIR** kopiert

Speicher

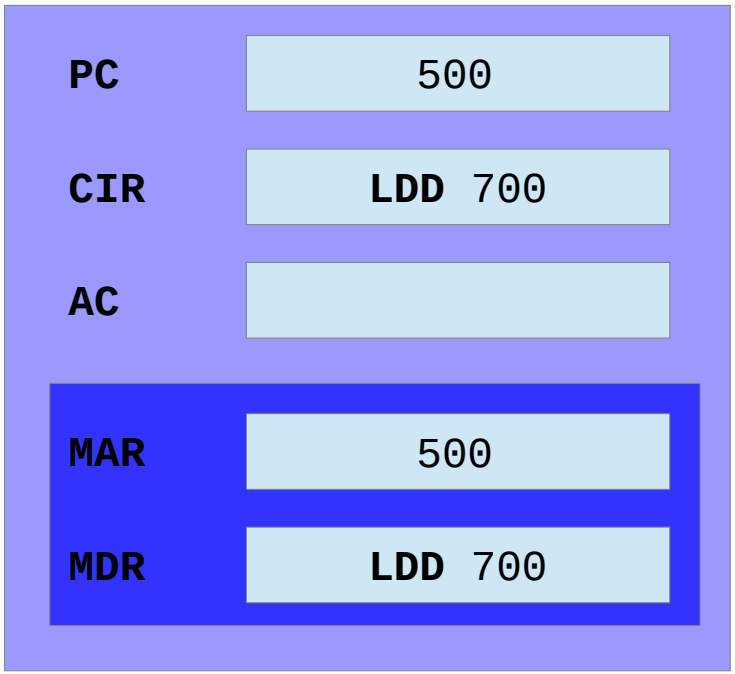
0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU



Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	501
CIR	**LDD** 700
AC	
MAR	500
MDR	**LDD** 700

4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

5. Steuerwerk dekodiert den Inhalt vom **CIR**, d.h. der Operand des Befehls wird ins **MAR** geladen

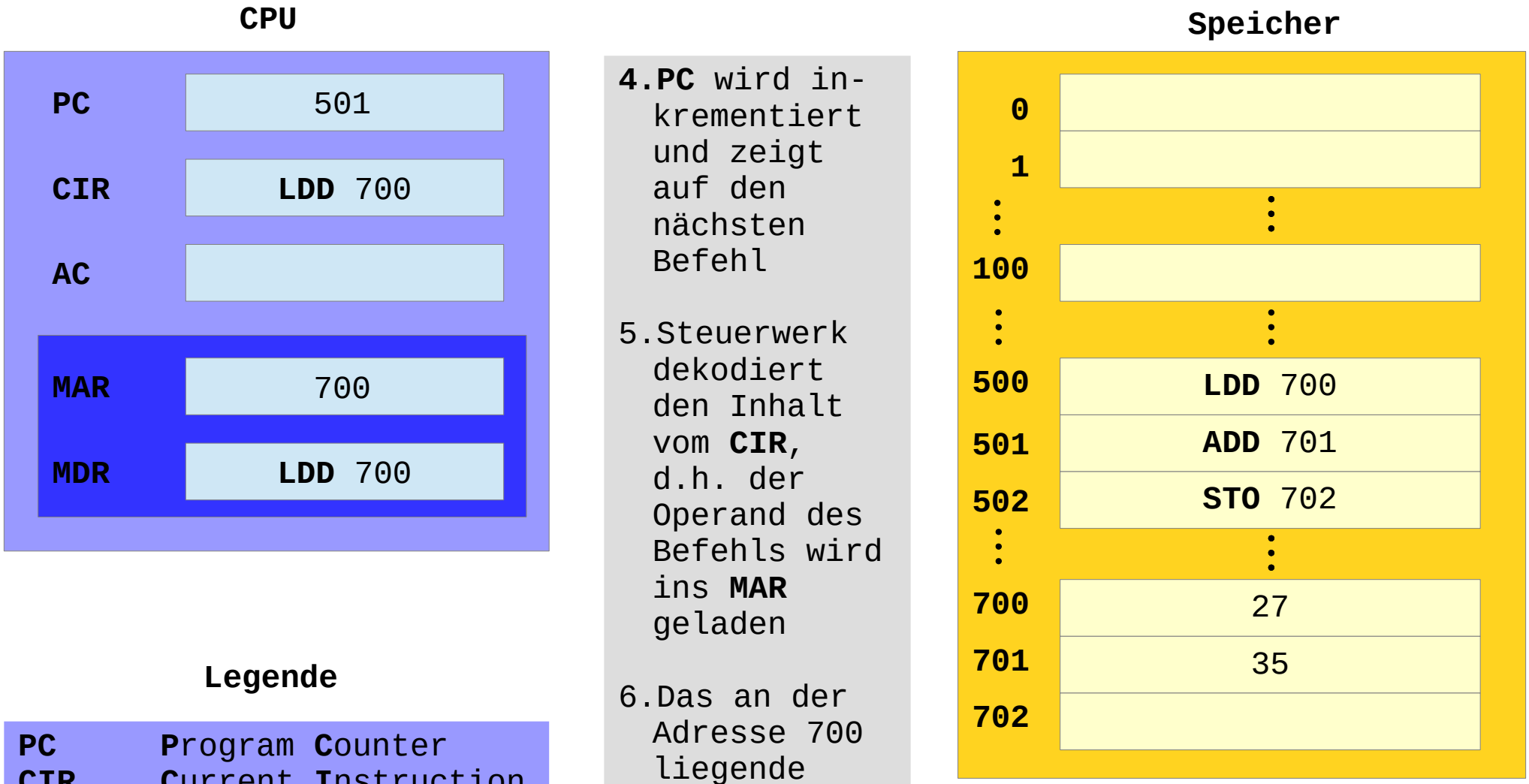
Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**



4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

5. Steuerwerk dekodiert den Inhalt vom **CIR**, d.h. der Operand des Befehls wird ins **MAR** geladen

6. Das an der Adresse 700 liegende Datum wird ins **MDR** kopiert

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

…Wert, der an Adresse **A** liegt
…chere ihn im **AC**
…den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**

STO C Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	501
CIR	**LDD** 700
AC	27
MAR	700
MDR	27

vorheriger Inhalt vom **MDR** muss unter Umständen in den **AC** kopiert werden

7. Inhalt vom **MDR** wird in den **AC** (Zwischenergebnis) kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	501
CIR	**LDD** 700
AC	27
MAR	700
MDR	27

2. Zyklus

1. Inhalt vom **PC** wird ins **MAR** kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	501
CIR	**LDD** 700
AC	27
MAR	501
MDR	27

2. Zyklus

1. Inhalt vom **PC** wird ins **MAR** kopiert

2. zugehöriger Befehl wird geholt und ins **MDR** kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	501
CIR	**LDD** 700
AC	27
MAR	501
MDR	**ADD** 701

2. Zyklus

1. Inhalt vom **PC** wird ins **MAR** kopiert
2. zugehöriger Befehl wird geholt und ins **MDR** kopiert
3. Befehl wird vom **MDR** ins **CIR** kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	501
CIR	**ADD** 701
AC	27
MAR	501
MDR	**ADD** 701

2. Zyklus

1. Inhalt vom **PC** wird ins **MAR** kopiert
2. zugehöriger Befehl wird geholt und ins **MDR** kopiert
3. Befehl wird vom **MDR** ins **CIR** kopiert

Speicher

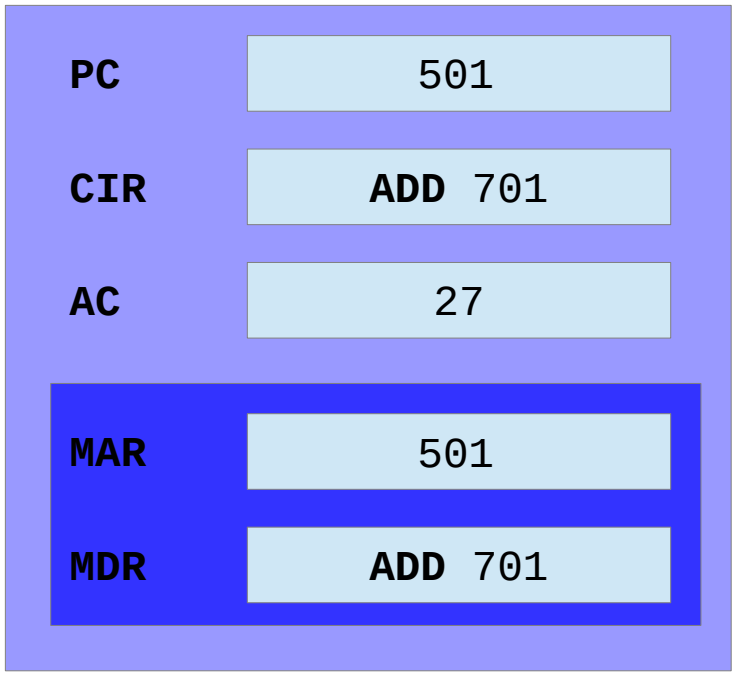
0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

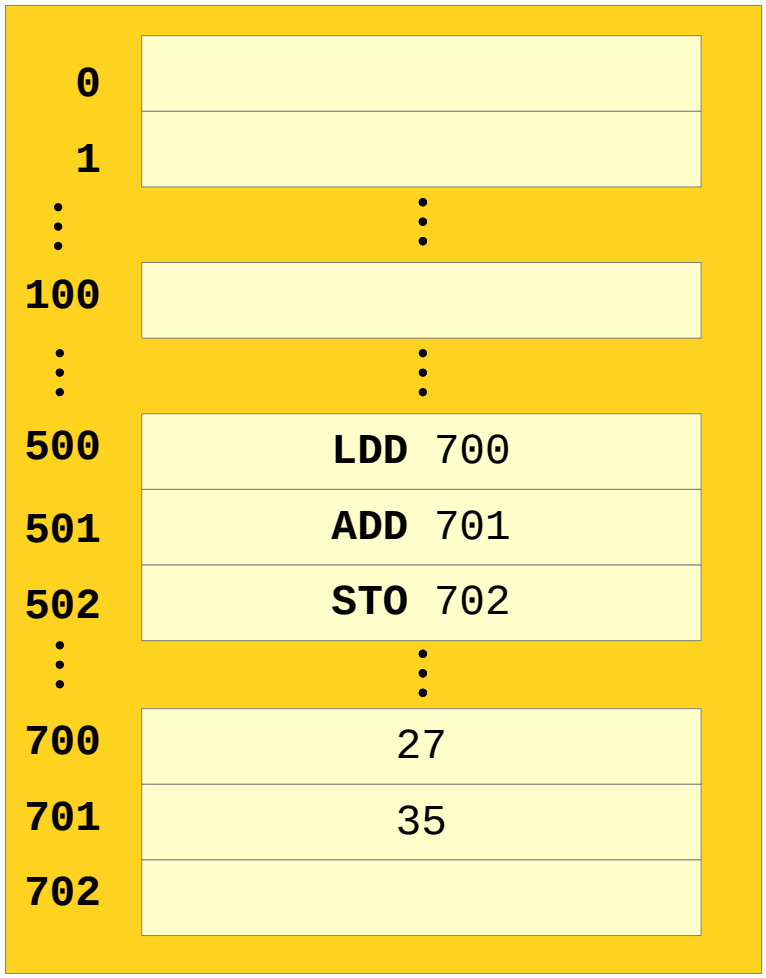
LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU



4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

Speicher



Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	502
CIR	**ADD** 701
AC	27
MAR	501
MDR	**ADD** 701

4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

5. Steuerwerk dekodiert Inhalt vom **CIR**, d.h. der Operand des Befehls wird ins **MAR** geladen

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	502
CIR	**ADD** 701
AC	27
MAR	701
MDR	**ADD** 701

4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

5. Steuerwerk dekodiert Inhalt vom **CIR**, d.h. der Operand des Befehls wird ins **MAR** geladen

6. das an der Adresse 701 liegende Datum wird ins **MDR** kopiert

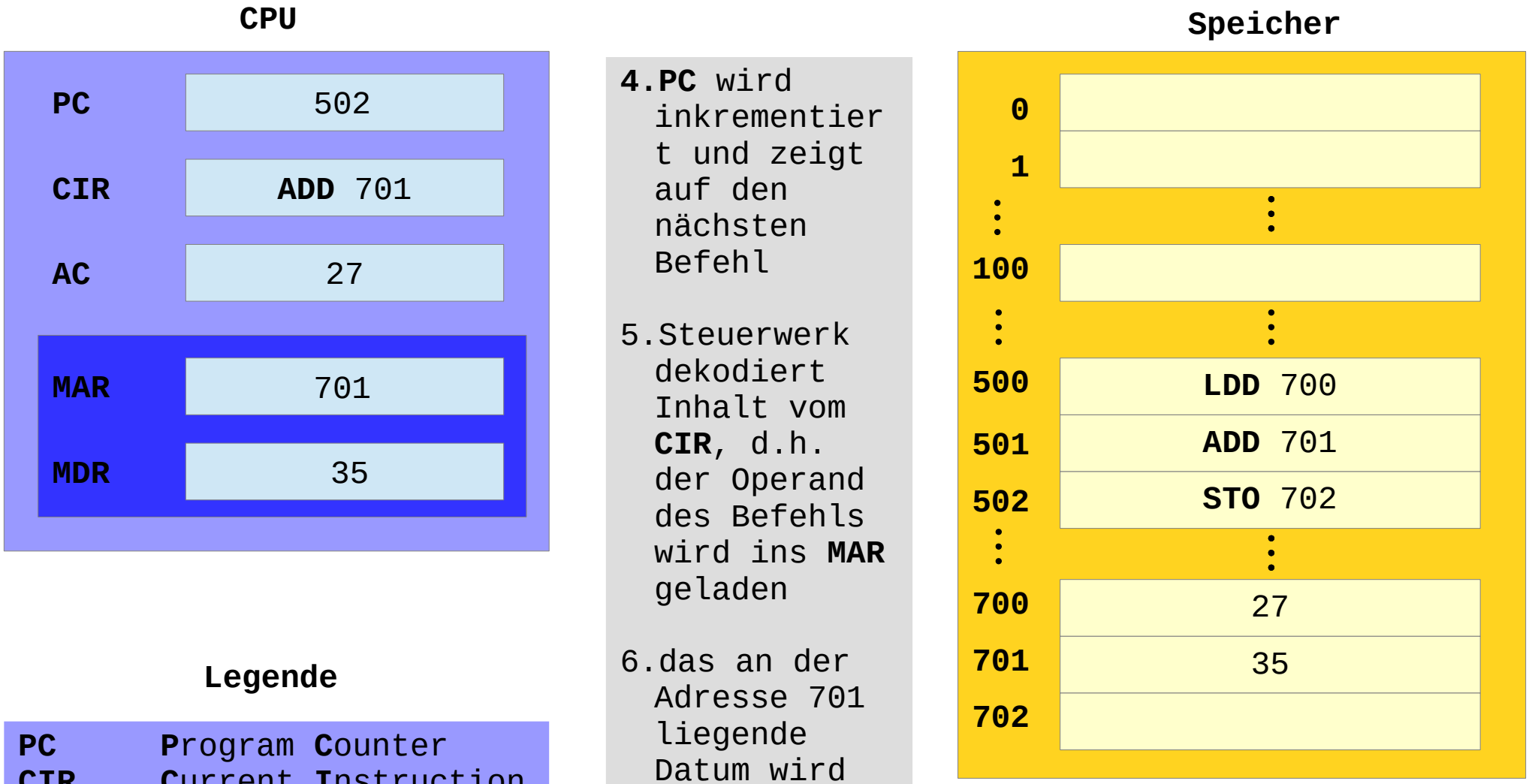
Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

	Wert, der an Adresse **A** liegt chere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**



4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

5. Steuerwerk dekodiert Inhalt vom **CIR**, d.h. der Operand des Befehls wird ins **MAR** geladen

6. das an der Adresse 701 liegende Datum wird ins **MDR** kopiert

Legende

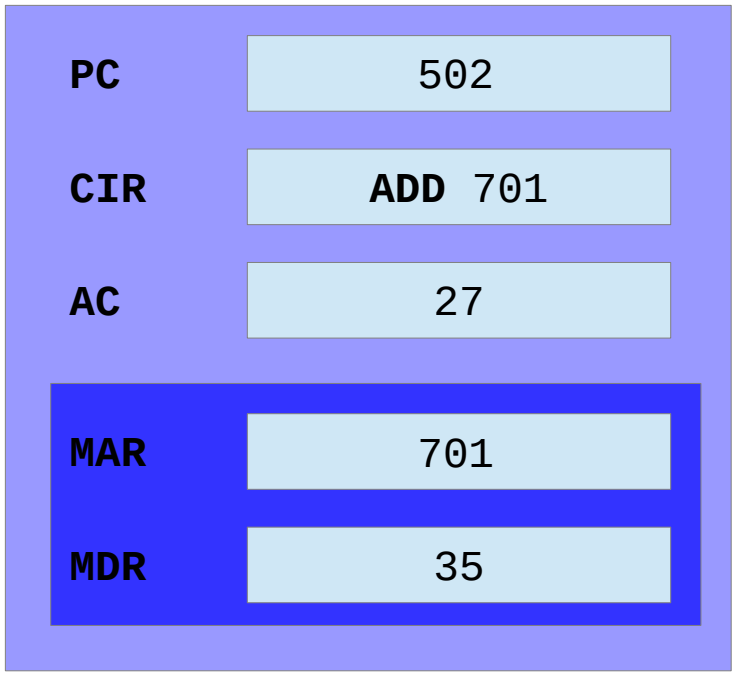
PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

Wert, der an Adresse **A** liegt
chere ihn im **AC**

ADD B Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**

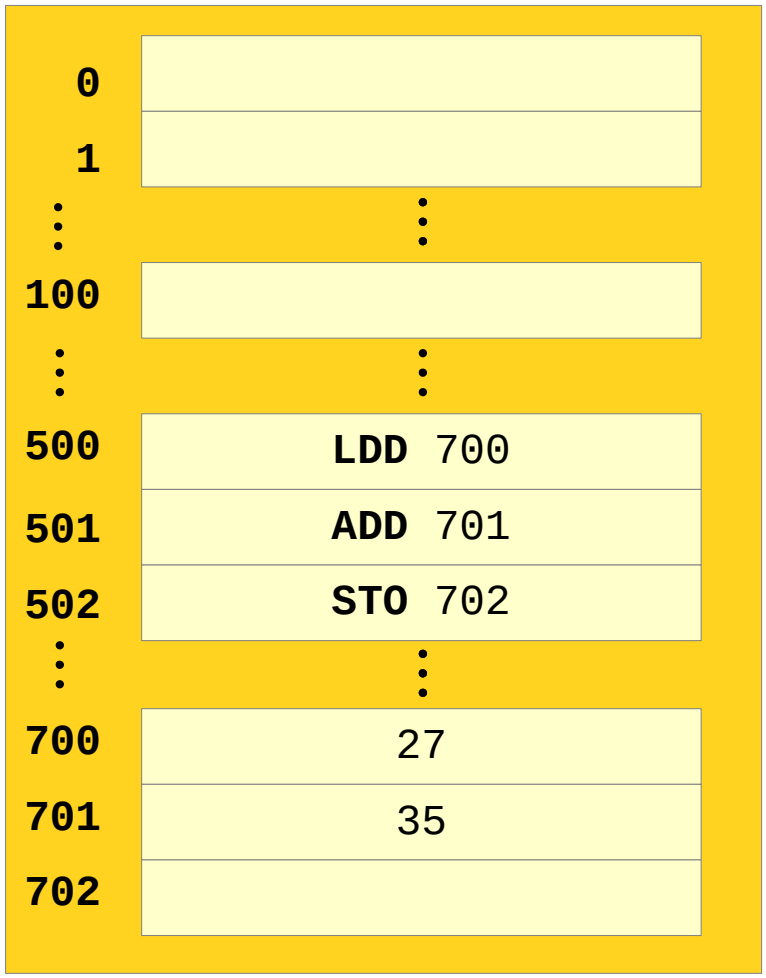
STO C Speichere den Wert aus dem **AC** in Adresse **C**

CPU



7. Inhalt vom **MDR** wird zum Inhalt vom **AC** addiert

Speicher



Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	502
CIR	**ADD** 701
AC	62
MAR	701
MDR	35

3. Zyklus

1. Inhalt vom **PC** wird ins **MAR** kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	502
CIR	**ADD** 701
AC	62
MAR	502
MDR	35

3. Zyklus

1. Inhalt vom **PC** wird ins **MAR** kopiert

2. zugehöriger Befehl wird geholt und ins **MDR** kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	502
CIR	**ADD** 701
AC	62
MAR	502
MDR	**STO** 702

3. Zyklus

1. Inhalt von **PC** wird nach **MAR** kopiert
2. zugehöriger Befehl wird geholt und in **MDR** kopiert
3. Befehl wird vom **MDR** nach **CIR** kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	502
CIR	**STO** 702
AC	62
MAR	502
MDR	**STO** 702

4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	503
CIR	**STO** 702
AC	62
MAR	502
MDR	**STO** 702

4. **PC** wird inkrementiert und zeigt auf den nächsten Befehl

5. Steuerwerk dekodiert Inhalt von **CIR**, d.h. der Operand des Befehls wird ins **MAR** geladen

Speicher

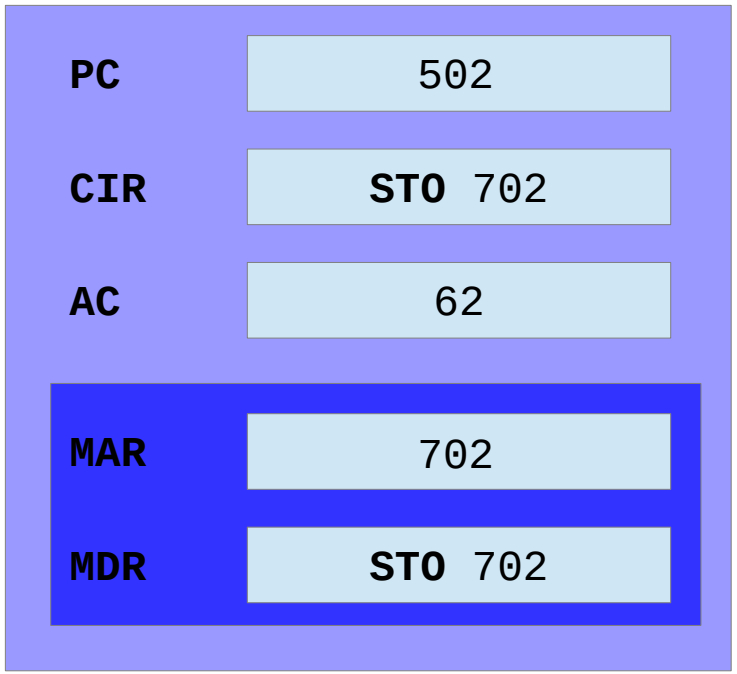
0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU



6. Inhalt vom **AC** wird ins **MDR** kopiert

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	503
CIR	**STO** 702
AC	62
MAR	702
MDR	62

6. Inhalt vom **AC** wird ins **MDR** kopiert

7. Inhalt vom **MDR** wird an der Adresse gespeichert, die im **MAR** angegeben ist

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**

CPU

PC	503
CIR	**STO** 702
AC	62
MAR	702
MDR	62

6. Inhalt vom **AC** wird ins **MDR** kopiert

7. Inhalt vom **MDR** wird an der Adresse gespeichert, die im **MAR** angegeben ist

Speicher

0	
1	
⋮	⋮
100	
⋮	⋮
500	**LDD** 700
501	**ADD** 701
502	**STO** 702
⋮	⋮
700	27
701	35
702	62

Legende

PC	**P**rogram **C**ounter
CIR	**C**urrent **I**nstruction **R**egister
AC	**A**ccumulator
MAR	**M**emory **A**ddress **R**egister
MDR	**M**emory **D**ata **R**egister

LDD A	Lade den Wert, der an Adresse **A** liegt und speichere ihn im **AC**
ADD B	Addiere den Wert, der an Adresse **B** liegt zu dem Inhalt vom **AC**
STO C	Speichere den Wert aus dem **AC** in Adresse **C**