



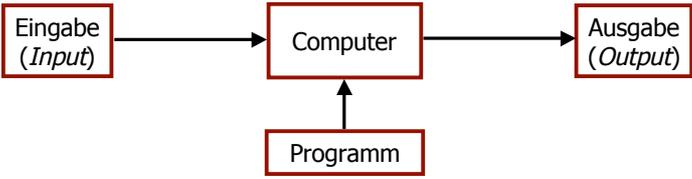
Informatik I

Aufbau und Funktionsweise eines Computers, abstrakte Maschinenmodelle

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de



Arbeitsweise eines Computers



```
graph LR; Input["Eingabe (Input)"] --> Computer["Computer"]; Computer --> Output["Ausgabe (Output)"]; Programm["Programm"] --> Computer;
```

- Computer
 - Programm legt die Bearbeitung der Eingaben fest
 - kann fast beliebige Aufgaben übernehmen
 - ⇒ **Universalrechner**

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 2

Internet

- weltweite Vernetzung praktisch aller Computer ("Das Internet ist der Computer")
- ist in Wahrheit ein Netz von Netzen

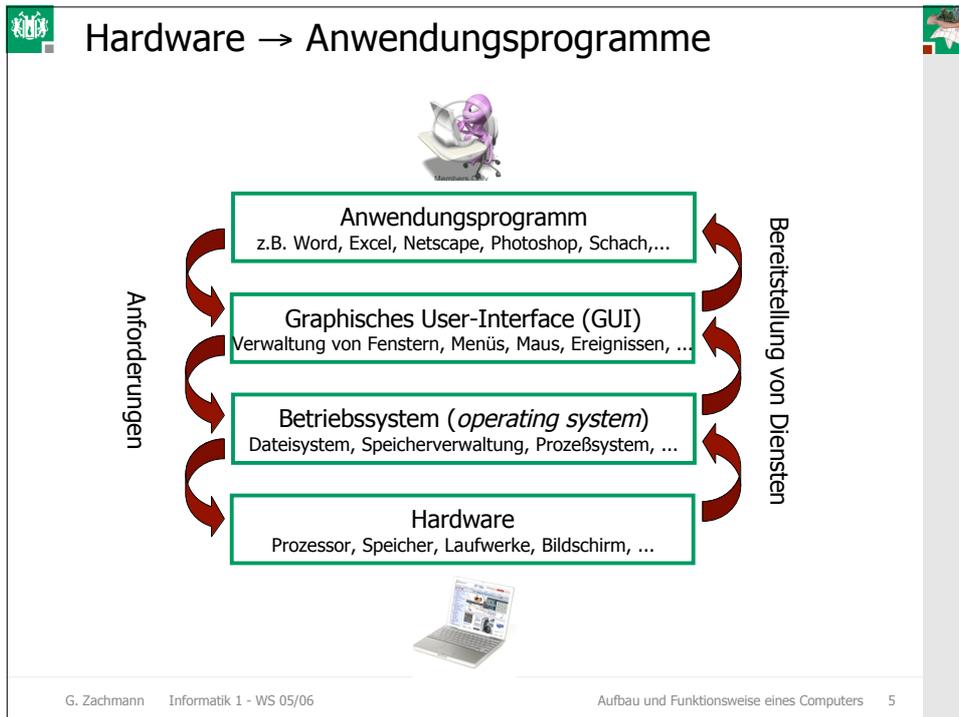
Das Diagramm zeigt die hierarchische Struktur des Internets. Ein zentraler Knoten 'Internet Service Provider (ISP)' ist mit vier weiteren 'ISP'-Knoten verbunden. Diese sind weiter untereinander vernetzt. Rechts sind Laptop-Symbole dargestellt, die mit den ISPs verbunden sind, was die Vernetzung von Computern über das Internet zeigt.

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 3

Zwischen Hardware und User

- Weiter Weg von der Hardware bis zum User
- Hardwareebene
 - CPU (*Central Processing Unit*) kann nicht viel mehr als
 - Speicherinhalte in Register auf der CPU laden
 - Registerinhalte in den Speicher ablegen
 - Registerinhalte logisch oder arithmetisch verknüpfen
 - Registerinhalte mit Peripheriegeräten austauschen
- Benutzerebene
 - User möchte z.B.
 - Briefe editieren und drucken
 - Photos bearbeiten
 - Computerspiele ausführen

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 4



Speicher

- Kleinste Speichereinheit hat **2 Zustände**
 - **1 Bit**
 - Zustände werden i.A. mit 0 und 1 bezeichnet
- Mit 2 Speichereinheiten $2^2=4$ Zustände darstellbar
- Mit 8 Bit sind $2^8=256$ Zustände darstellbar
 - **8 Bit = 1 Byte**
 - Heutzutage sind Bytes die kleinsten *adressierbaren* Speichereinheiten
 - Kleinere Einheiten müssen aus einem Byte extrahiert werden
- Viele Daten benötigen **4 Byte = 1 Wort**

Bit 1	Bit 0	
0	0	Zustand 0
0	1	Zustand 1
1	0	Zustand 2
1	1	Zustand 3

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 6



Kilo, Mega, Giga

- In der Informatik wird mit *kilo* meist $1024=2^{10}$ gemeint
 - 1 kByte = 1024 Byte
 - Mit Mega $1024 \cdot 1024 = 2^{20}$
 - 1 MByte = 1024 kByte
 - Mit Giga $1024 \cdot 1024 \cdot 1024 = 2^{30}$
 - 1 GByte = 1024 MByte
 - Entsprechend **kBit, MBit**
 - Manchmal auch kB für kByte und kb für kBit (entsprechend MB, Mb, GB, Gb)
- Widerspricht eigentlich dem normierten Sprachgebrauch, in dem **k** immer **1000**, und **M** immer **1000000** bezeichnen muß
 - Festplattenhersteller benutzen diesen normierten Sprachgebrauch
 - Damit scheint die Speicherkapazität etwa höher ☺

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 7

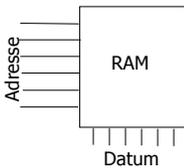


- Weitere Einheiten:
 - Wort (*word*) = 4 Byte
- Heutige Rechner können meist **32 Bit** oder **64 Bit** auf einmal verarbeiten
 - Pentium noch 32 Bit
 - Itanium-2 Prozessor schon mit 64 Bit
 - Spiele-Konsolen schon ab 1998 64-Bit CPUs

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 8

Adressen

- Jedes Byte im Speicher hat eine Nummer = **Adresse**
- Mit **Speicheradressen** von **32 Bit** können 2^{32} Byte = $4 \cdot 2^{30}$ Byte = **4 Gbyte** = 4096 MByte **adressiert** werden
 - 64 Bit ergeben 2^{64} Byte = 2^{34} Gbyte $\approx 10^{11}$ Gbyte adressiert werden
- Wenn ein Wort aus den Bytes mit den Adressen $n, n+1, n+2, n+3$ besteht, dann ist n die **Adresse** des Worts
 - In einem Speichermodul sind die Werte von n , die durch 4 teilbar sind, die natürlichen Grenzen für Worte
 - An solchen Stellen beginnende Worte sind an den **Wortgrenzen** (*word boundary*) **ausgerichtet** (*aligned*)



The diagram shows a rectangular box labeled 'RAM'. On the left side, there are five horizontal lines pointing towards the box, labeled 'Adresse'. On the bottom side, there are five vertical lines pointing away from the box, labeled 'Datum'.

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 9

Famous last words

Bill Gates in 1981:
"640 K ought to be enough for anybody",



Thomas Watson, chairman of IBM in 1943:
"I think there is a world market for five computers only".



G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 10

Big endian, Little endian

- Ein Wort fängt immer mit dem am weitesten links stehenden Byte an
 - in dem die Bits mit den höchsten Nummern (=Wertigkeiten) stehen
- Es endet mit dem am weitesten rechts stehenden
 - in dem die niederwertigsten Bits stehen
- Frage:** Beginnt die Zählung n , $n+1$, $n+2$, $n+3$ der Bytes links oder rechts?
 - Auf diese Frage gibt es **beide Antworten!**

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 11

- Sowohl *Big Endian* als auch *Little Endian* werden benutzt
 - SUN SPARC, IBM Mainframes, und SGI sind *Big Endian*
 - Die Intel Familie ist *Little Endian*
- Dieser Unterschied macht dann große Probleme, wenn ein Wort byteweise zwischen *verschiedenen* Computern übermittelt wird
 - Das "NUXI"-Problem

Little Endian

Big Endian

Beispiel: Repräsentation von 1025

00000000 00000000 00000100 00000001

Address	Big-Endian representation of 1025	Little-Endian representation of 1025
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 12

Organisation der Hardware

- Architektur eines einfachen Computersystems mit Bus

- CPU = Central Processing Unit

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 13

Architektur eines PC Systems mit mehreren Bussen an Brücken

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 14

Prozessor und Programmausführung

- CPU = Steuerwerk + ALU + FPU + Register + Cache
 - Steuerwerk = holt aus Speicher Befehle und interpretiert sie, sorgt für den richtigen Datenfluß in der CPU,
 - ALU (*arithmetic-logic unit*) = "einfache" Operationen (1+2)
 - FPU (*floating-point unit*) = Operationen auf reellen Zahlen
 - Register = "Variablen" (typ. ca. 64 Stück)
 - Cache = sehr schneller Zwischenspeicher
- Befehle sind in **Maschinsprache**
 - CISC = complex instruction set computer
 - RISC = reduced instruction set computer

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 15

Chip Technik

- CPU in **VLSI Technik**
 - VLSI=very large scale integration
- Transistoren als Grundstruktur
 - Größenordnung 50-100 nm (Nanometer)
- **Moore's Law:** Anzahl Transistoren/Chip verdoppelt sich in jeweils 18 Monaten (eigtl. 2 Jahre)
 - D.h., exponentielles Wachstum!
 - Pentium II: 7 Mill. Transistoren
 - Itanium-2: 220 Mill. Transistoren
 - Phys. Grenze (1 Transistor besteht aus wenigen Atome) um 2020
 - Was kommt dann????

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 16

Ein paar Takte zur Geschwindigkeit ...

- Chips sind „getaktet“: Pro Takt wird eine einfache Operation ausgeführt
 - heute ≤ 4 GHz: 1 Takt = 0.25 ns
 - Lichtstrecke bei 0.25 ns = 7,5 cm
 - Im Abstand von 75 cm „beobachten“ (z.B. per Radio) wir am Chip also die vergangenen Zustände von vor 10 Takten!
- Komplexe Operationen brauchen mehrere Takte (z.B. Mult, externe Operanden holen)
- Partielle Lösung: *pipelining*
 - Neues Problem: bei Sprüngen alles umsonst (*pipeline invalidation*)
 - Lösung hierzu: viele Register, schnelle Zwischenspeicher (Caches)

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 17

Leistungssteigerung der CPUs

- MFLOPS = Million Floating Point Operations Per Second
 - 1 MFLOP = 10^6 Gleitkommaoperationen pro Sekunde

	MFLOPS	Transistoren/Chip
1950	0,001	-
1960	0,1	10
1970	1	100
1980	50	1 000
1990	150	1 000 000
2000	>1000	> 50 000 000

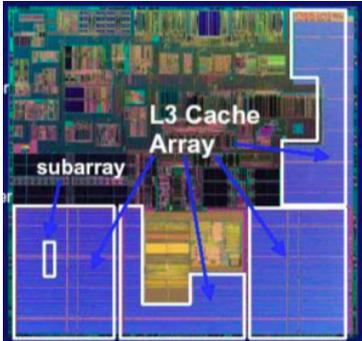
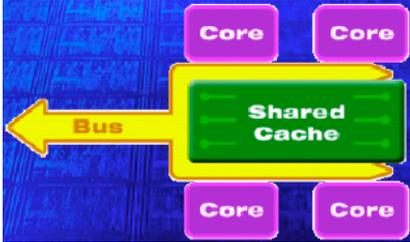
- NB: TeraFLOP-Rennen (früher GFLOP-Rennen) im Super-Computer-Bereich
 - Siehe top500.org
 - BlueGene (IBM), Altix (SGI), EarthSimulator (NEC),

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 18

Trend

- Itanium-2 (2002):
 - 221 Mio Transistoren
 - 3MB L3-Cache
 - 6.4 GByte/sec IO

- Mehrere "Cores" um Cache herum anordnen

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 19

■ Dallas Morning News, Nov 1990



G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 20

▪ 80386, 20 MHz	150:1	▪ Pentium IV, 3 GHz
▪ 1 Mbyte RAM	1024:1	▪ 1 Gbyte RAM
▪ 40 MB Platte	5000:1	▪ 200 GB Platte
▪ 2400 bit/sec (Baud)	416:1	▪ 1 Mbit/sec (DSL)
▪ DOS		▪ DOS / WinNT
▪ VGA (640x480)	ca. 4:1	▪ 1280x1024

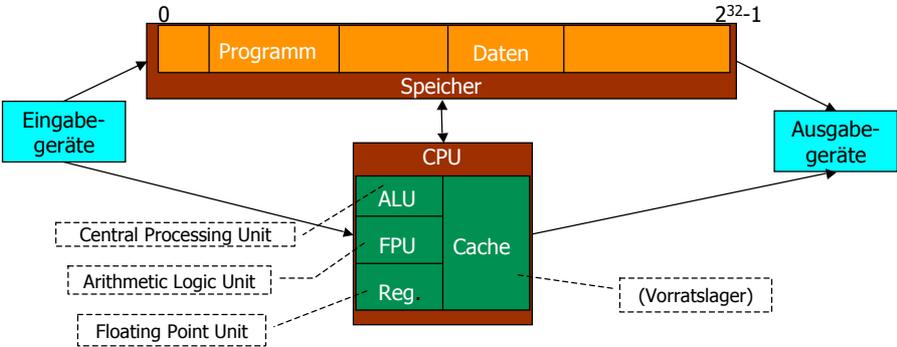


"Moore's Wall"

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 21

Von Neumann Architektur

- Daten und Programm **gemeinsam** im Hauptspeicher
- Fundamentaler Instruktionszyklus
- Spezielles Register mit Befehlszähler (*program counter = PC*) enthält Adresse des aktuellen Maschinenbefehls



The diagram illustrates the Von Neumann architecture. At the top, a horizontal bar represents memory (Speicher) from address 0 to 2³²-1, divided into 'Programm' and 'Daten' sections. Below this is the CPU, which contains an ALU, FPU, and Register (Reg.). To the right of the CPU is a Cache, labeled as '(Vorratslager)'. On the left, 'Eingabegeräte' (input devices) and on the right, 'Ausgabegeräte' (output devices) are shown. Arrows indicate the flow of data and instructions between these components. Dashed boxes on the left group the ALU, FPU, and Reg. as the 'Central Processing Unit' and 'Arithmetic Logic Unit', while the Cache is labeled as '(Vorratslager)'.

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 22

Fundamentaler Instruktionszyklus

- **Fetch:** Hole den Befehl, dessen Adresse im Befehlszähler steht, aus dem Speicher in das Befehlsregister
- **Increment:** Inkrementiere den Befehlszähler, damit er auf die nächste auszuführende Instruktion weist
- **Decode:** Dekodiere die Instruktion, damit klar wird, was zu tun ist
- **Fetch operands:** Falls nötig, hole die Operanden aus den im Befehl bezeichneten Stellen im Speicher
- **Execute:** Führe die Instruktion aus, ggf. durch die ALU. (Bei einem Sprung wird neuer Wert in das Befehlsregister geschrieben.)
- **Loop:** Gehe zum ersten Schritt (**Fetch**)

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 23

Beispiel: Der TOY-Computer



The image shows the TOY computer interface with the following components:

- Buttons:** Load (blue), Look (blue), Step (green), Run (green), and Reset (red).
- ADDR (Address):** A row of four lights, with the second and third lights illuminated.
- OUTPUT:** A green seven-segment display showing the number 8888.
- DATA:** A row of eight lights, with the first and second lights illuminated.

Quelle: <http://www.cs.princeton.edu/introcs/50machine/>
(courtesy Robert Sedgewick and Kevin Wayne.)

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 24

Was ist in der Box

- **Schalter**
 - Eingabe der Daten und Programme
- **Lämpchen**
 - Zeigen Daten an
- **Register**
 - schnellste Form des Speichers
 - Verwendung als Variablen während der Berechnung
 - 16 Register, jedes speichert 16 Bit
 - Register 0 ist immer 0
- **Program counter (PC)**
 - ein extra 8-Bit Register
 - behält nächste auszuführende Anweisung im Auge
- **Datenspeicher**
 - speichert Daten und Programme
 - 256 "Wörter"
 - ein TOY-Wort hat 16 Bit
 - Adresse 256 (= FF) für I/O
- **Arithmetic-logic unit (ALU)**
 - verarbeitet Daten

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 27

Daten und Programme sind binär codiert

- **Jedes Bit hat 2 Zustände:**
 - Der Schalter ist ON oder OFF.
 - Spannung oder keine Spannung.
 - 1 oder 0.
 - Wahr oder falsch.
- **Wie stellt man Zahlen dar?**
 - Verwendung binärer Codierung.
 - Bsp: $6375_{10} = 0001100011100111_2$

Dec	Bin	Dec	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1	
			$+2^{12}$	$+2^{11}$				$+2^7$	$+2^6$	$+2^5$				$+2^2$	$+2^1$	$+2^0$
			= 4096	+2048				+128	+64	+32				+4	+2	+1

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 28

Shorthand-Notation

- Verwendung hexadezimaler (Basis 16) Zahlendarstellung.
 - Binärer Code, 4 Bits auf einmal.
 - Bsp:

$$6375_{10} = 0001100011100111_2$$

$$= 18E7_{16}$$

Dec	Bin	Hex	Dec	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
1				8				E				7			

$6375_{10} = 1 \times 16^3$	$+ 8 \times 16^2$	$+ 14 \times 16^1$	$+ 7 \times 16^0$
$= 4096$	$+ 2048$	$+ 224$	$+ 7$

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 29

Der Core-Dump

- Kompletter Inhalt der Maschine (Speicher + Register) zu bestimmter Zeit.
 - Aufzeichnung was das Programm getan hat.
 - Legt komplett fest, was Maschine tun wird.

Registers			
R0	R1	R2	R3
0000	0000	0000	0000
R4	R5	R6	R7
0000	0000	0000	0000
R8	R9	RA	RB
0000	0000	0000	0000
RC	RD	RE	RF
0000	0000	0000	0000

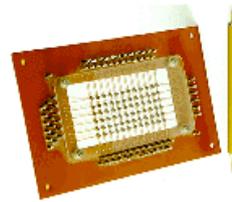
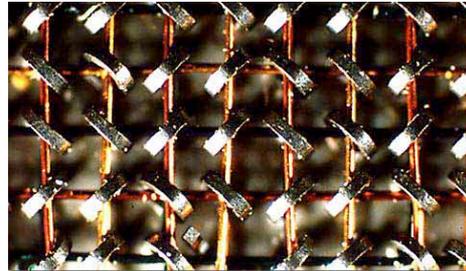
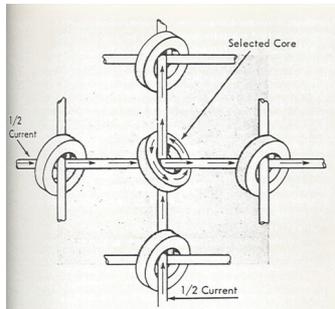
PC
10

Main Memory							
00:	0008	0005	0000	0000	0000	0000	0000
08:	0000	0000	0000	0000	0000	0000	0000
10:	8A00	8B01	1CAB	9C02	0000	0000	0000
18:	0000	0000	0000	0000	0000	0000	0000
20:	0000	0000	0000	0000	0000	0000	0000
28:	0000	0000	0000	0000	0000	0000	0000
.							
.							
E8:	0000	0000	0000	0000	0000	0000	0000
F0:	0000	0000	0000	0000	0000	0000	0000
F8:	0000	0000	0000	0000	0000	0000	0000

Nächste Anweisung →
 Daten →
 Programm →
 Variablen →

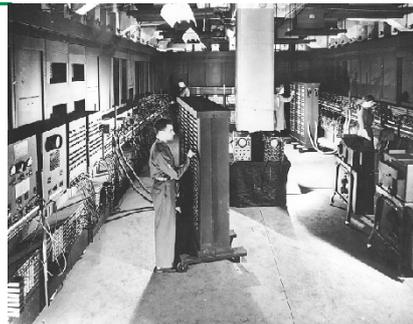
G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 30

Warum heißt es "core"?



Eins dieser berühmten Zitate ("Er könnte nicht mehr daneben liegen") von 1950:
Howard Aiken, leitender Designer des Mark I, während einer Unterhaltung mit John Curtiss, vom National Council of Investigation of the USA:

"Wir werden niemals genug Probleme haben, um genug Arbeit zu haben, damit 2 Computer daran arbeiten könnten."



Ein Beispielprogramm

- Ein Beispielprogramm.
 - Addition $8 + 5 = D$ (hex)

RA	RB	RC
0000	0000	0000

PC
10

Register

00:	0008	8	add.toy
01:	0005	5	
02:	0000	0	
10:	8A00	RA ← mem[00]	
11:	8B01	RB ← mem[01]	
12:	1CAB	RC ← RA + RB	
13:	9C02	mem[02] ← RC	
14:	0000	halt	

Datenspeicher

Wegen PC = 10, interpretiert die Maschine 8A00 als einen Maschinenbefehl.

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 33

Load

- Opcode 8 [opcode = operation code]
 - Lade Inhalt einer Speicherstelle in ein Register.
 - 8A00 = "Lade Inhalt von Zelle 00 in Register A".

RA	RB	RC
0000	0000	0000

pc
10

Register

00:	0008	8	add.toy
01:	0005	5	
02:	0000	0	
10:	8A00	RA ← mem[00]	
11:	8B01	RB ← mem[01]	
12:	1CAB	RC ← RA + RB	
13:	9C02	mem[02] ← RC	
14:	0000	halt	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
8 ₁₆				A ₁₆				00 ₁₆							
opcode				dest d				addr							

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 34

▪ 8B01 = "Lade Inhalt von Zelle 01 in Register B".

RA	RB	RC
0008	0000	0000

Register

pc
11

00:	0008	8	add.toy
01:	0005	5	
02:	0000	0	
10:	8A00	RA	← mem[00]
11:	8B01	RB	← mem[01]
12:	1CAB	RC	← RA + RB
13:	9C02	mem[02]	← RC
14:	0000	halt	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
8 ₁₆				B ₁₆				0 ₁₆							
opcode				dest d				addr							

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 35

Add

▪ Opcode 1

- Addiere Inhalt von 2 Registern und speichere in Ziel-Register
- 1CAB = "Addiere Inhalt von A und B und gebe Ergebnis in Register C".

RA	RB	RC
0008	0005	0000

Register

PC
12

00:	0008	8	add.toy
01:	0005	5	
02:	0000	0	
10:	8A00	RA	← mem[00]
11:	8B01	RB	← mem[01]
12:	1CAB	RC	← RA + RB
13:	9C02	mem[02]	← RC
14:	0000	halt	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	1	0	1	0	1	1
1 ₁₆				C ₁₆				A ₁₆				B ₁₆			
opcode				dest d				source s				source t			

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 36

Store

- opcode 9
 - Speichere Inhalt eines Registers in einer Speicherzelle.
 - 9C02 = "Speichere Inhalt von Register A in Speicherzelle 02."

RA	RB	RC
0008	0005	000D

PC
13

Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0
9 ₁₆				C ₁₆				02 ₁₆							
opcode				dest d				addr							

00:	0008	8	add.toy
01:	0005	5	
02:	0000	0	
10:	8A00	RA	← mem[00]
11:	8B01	RB	← mem[01]
12:	1CAB	RC	← RA + RB
13:	9C02	mem[02]	← RC
14:	0000	halt	

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 37

Halt

- Opcode 0
 - Maschine anhalten.

RA	RB	RC
0008	0005	000D

pc
14

Register

00:	0008	8	add.toy
01:	0005	5	
02:	0000	0	
10:	8A00	RA	← mem[00]
11:	8B01	RB	← mem[01]
12:	1CAB	RC	← RA + RB
13:	9C02	mem[02]	← RC
14:	0000	halt	

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 38

Programm und Daten

- Programm:
 - Sequenz von Anweisungen.
- 16 Anweisungstypen:
 - 16-Bit Wort (interpretiert auf eine Weise).
 - Ändert den Inhalt von Registern, Datenspeicher und PC in spezifischer, wohl-definierter Weise.
- Daten:
 - 16-Bit Wort (interpretiert auf eine andere Weise).
- Program counter (PC):
 - speichert Adressen der "nächsten Anweisung."

Instructions	
0:	halt
1:	add
2:	subtract
3:	and
4:	xor
5:	shift left
6:	shift right
7:	load address
8:	load
9:	store
A:	load indirect
B:	store indirect
C:	branch zero
D:	branch positive
E:	jump register
F:	jump and link

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 39

TOY Instruction Set Architecture

- TOY instruction set architecture (*ISA*).
 - Definiert Aufbau und genaues Verhalten jeder Instruktion
 - 16 Register, 256 Wörter des Hauptspeichers, 16-Bit Wörter.
 - 16 Anweisungen.
- Jede Anweisung besteht aus 16 Bits.
 - Bits 12-15 codieren einen von 16 Anweisungsarten oder *opcodes*.
 - Bits 8-11 codieren Zielregister *d*.
 - Bits 0-7 codieren, ja nach Opcode:
 - Format 1: Quellenregister *s* und *t*
 - Format 2: 8-Bit Speicheradresse oder Konstante

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0
Format 1	opcode				dest d				source s				source t			
Format 2	opcode				dest d				addr							

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 40

Bedienung des TOY-Computers

- Wie man ein Programm eingibt:
 1. Setze 8 Speicheradressenschalter.
 2. Setze 16 Datenschalter.
 3. Drücke LOAD.



The image shows the front panel of the TOY computer. It features four buttons at the top: 'Load' (blue), 'Look' (blue), 'Step' (green), and 'Run' (green), along with a 'Reset' button (red). Below these are two rows of switches. The top row is labeled 'ADDR' and has 8 switches, with the first three illuminated. The bottom row is labeled 'DATA' and has 16 switches. To the right of the switches is a green 7-segment display labeled 'OUTPUT' showing the number '8888'.

- Wie man das Resultat abruft:
 - Setze 8 Speicheradressenschalter.
 - Drücke LOOK: Inhalt des adressierten Wortes erscheint an Lämpchen

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 41

Benutzung der TOY Maschine: Run

- Um das Programm zu starten:
 - Setze 8 Speicheradressenschalter auf die Adresse der ersten Anweisung
 - Drücke LOOK um den PC auf erste Anweisung zu setzen.
 - Drücke RUN Button
 - führt Fetch-Execute-Zyklus bis zum halt opcode durch
- Fetch-Execute-Zyklus
 - FETCH: Anweisung vom Speicher holen.
 - EXECUTE: PC erhöhen, Daten zu oder aus Speicher und Registern bewegen, Berechnungen durchführen.



The diagram shows a circular flow with two arrows. A green arrow at the top points clockwise and is labeled 'Fetch'. A brown arrow at the bottom points clockwise and is labeled 'Execute'.

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 42

Programmieren der TOY-Maschine

- Um die volle Mächtigkeit zu haben, brauchen wir Schleifen und Bedingungen in der TOY-Maschine
- Idee:
 - PC benutzen, um Programmfluß zu kontrollieren.
 - Ändere PC abhängig von Bedingung in einem Register
- **Springen wenn 0.** (opcode C)
 - ändert PC, abhängig vom Wert einiger Register.
 - zum Implementieren benutzt: `for`, `while`, `if-else`.
- **Springen wenn positiv.** (opcode D)
 - Analog.

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 43

Beispiel: Multiplikation

- keine direkte Unterstützung in der TOY Hardware.
- Lade Zahlen `a` und `b`, und speicher `c = a × b`.
- Brute-force Algorithmus:
 - Initialisierung `c = 0`
 - `b` zu `c` addieren, `a` mal

```

int a = 3;
int b = 9;
int c = 0;

while ( a != 0 )
{
    c = c + b;
    a = a - 1;
}
Java
```

ingorierte Probleme: Overflow, langsam, negative Zahlen.

G. Zachmann Informatik 1 - WS 05/06 Aufbau und Funktionsweise eines Computers 44

Multiplizieren

0A: 0003 3 ← Eingabe
 0B: 0009 9 ← Eingabe
 0C: 0000 0 ← Ausgabe

0D: 0000 0 ← Konstanten
 0E: 0001 1 ← Konstanten

10: 8A0A RA ← mem[0A] a
 11: 8B0B RB ← mem[0B] b
 12: 8C0D RC ← mem[0D] c = 0

13: 810E R1 ← mem[0E] always 1

14: CA18 if (RA == 0) PC ← 18 while (a != 0) {
 15: 1CCB RC ← RC + RB c = c + b
 16: 2AA1 RA ← RA - R1 a = a - 1
 17: C014 PC ← 14 }
 18: 9C0C mem[0C] ← RC
 19: 0000 halt

multiply.toy

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 45

Step-By-Step Trace

		R1	RA	RB	RC
10: 8A0A	RA ← mem[0A]		0003		
11: 8B0B	RB ← mem[0B]			0009	
12: 8C0D	RC ← mem[0D]				0000
13: 810E	R1 ← mem[0E]	0001			
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0009
16: 2AA1	RA ← RA - R1		0002		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0012
16: 2AA1	RA ← RA - R1		0001		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				001B
16: 2AA1	RA ← RA - R1		0000		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
18: 9C0C	mem[0C] ← RC				
19: 0000	halt				

G. Zachmann Informatik 1 - WS 05/06
Aufbau und Funktionsweise eines Computers 46