3. Sequentielle Schaltkreise

3.1 Vergleich kombinatorische $\Longleftrightarrow$ sequentielle Schaltkreise

3.2 Binäre Speicherelemente

3.2.1 RS Flipflop

3.2.2 Getaktetes RS Flipflop

3.2.3 D Flipflop

3.2.4 JK Flipflop

3.2.5 Master Slave JK Flipflop

3.3 Schaltungen mit Flipflops

3.3.1 Zähler und Teiler

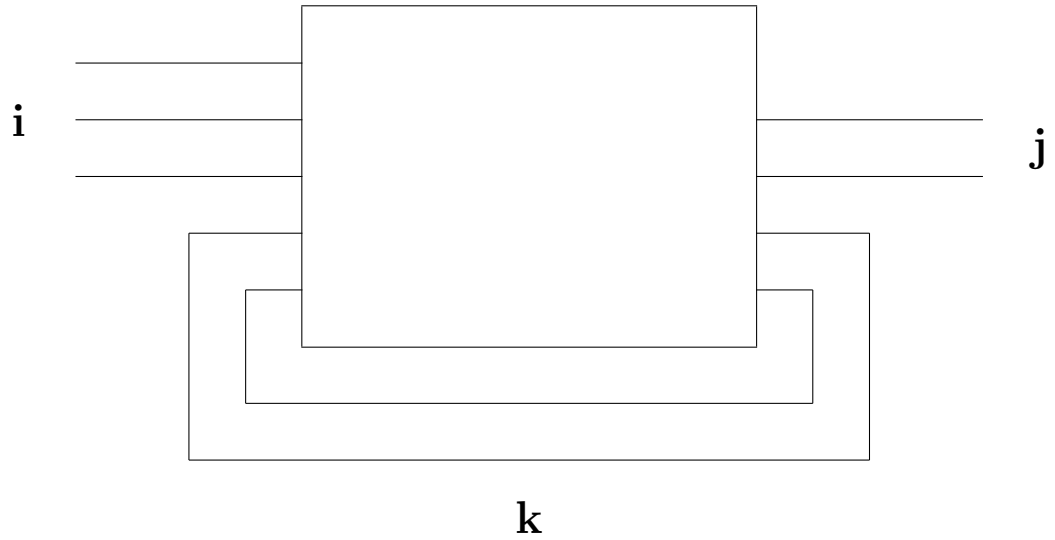
3.3.2 Register

Vergleich von kombinatorischem Schaltkreis mit sequentiellem Schaltkreis

Kombinatorischer Schaltkreis

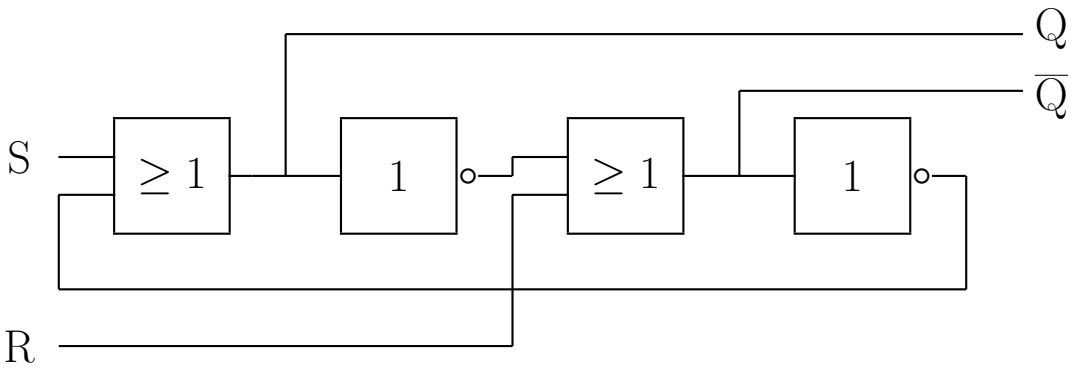
Der Zustand der Ausgänge j wird eindeutig von den Zuständen der Eingänge i bestimmt.

Sequentieller Schaltkreis



Der Zustand der Ausgänge j wird von den Zuständen der Eingänge i und den Rückführungen k bestimmt. Die Rückführungen implementieren die Funktionälitat eines Gedächtnisses.

RS-Flipflop



Wahrheitstabelle

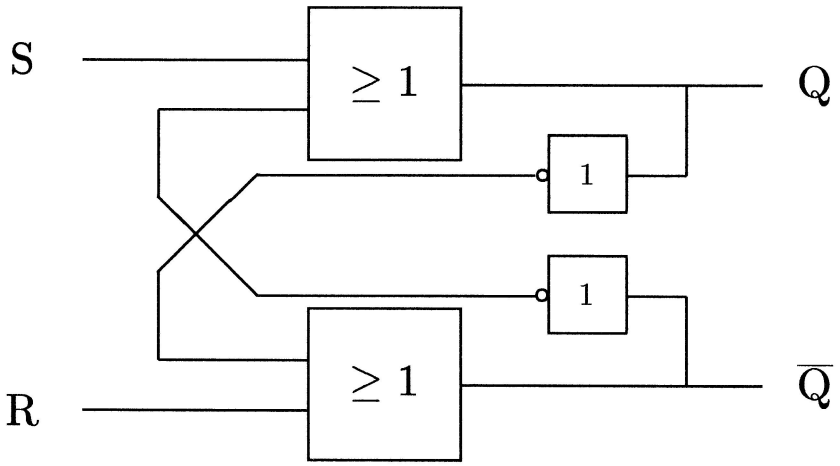
Q_v	S	R	Q_n	$\overline{Q}_n$
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

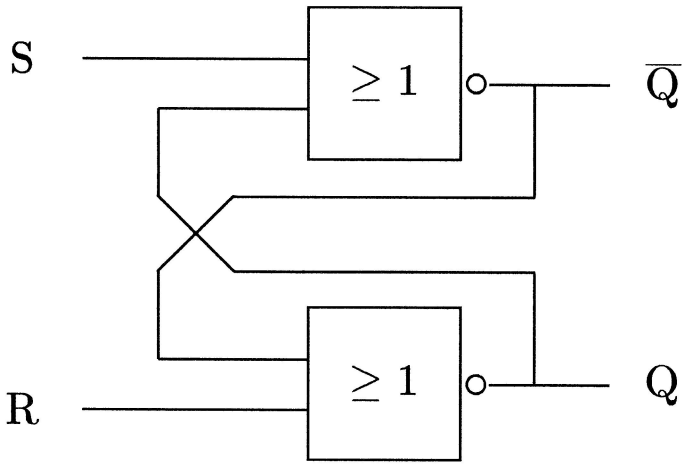
Komprimierte Wahrheitstabelle

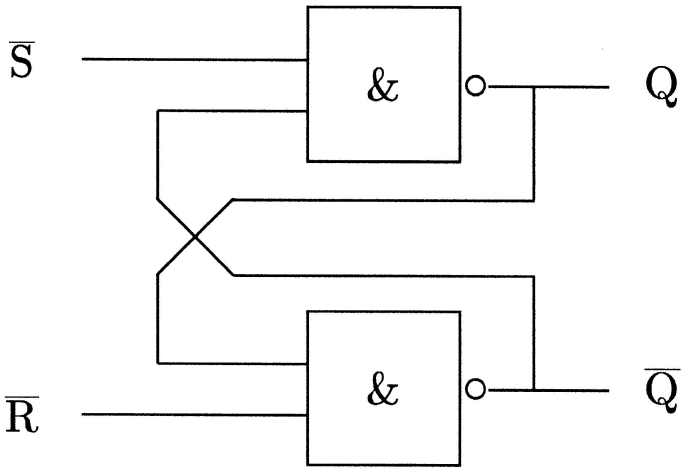
Q_v	S	R	Q_n
Q_v	0	0	Q_v
X	1	0	1
X	0	1	0

Nebenbedingung: $R \wedge S = 0$

RS-Flipflops

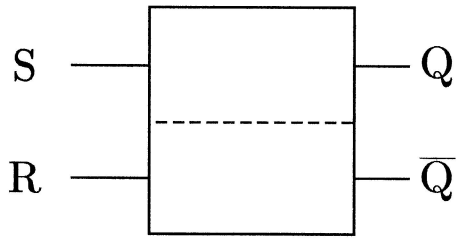




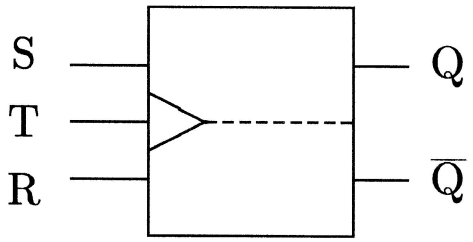


Schaltsymbole von Flipflops

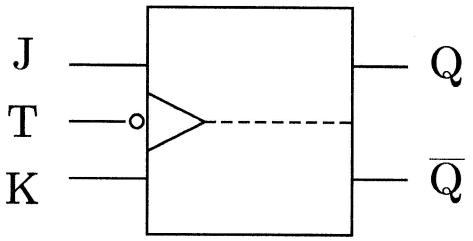
RS-Flipflop



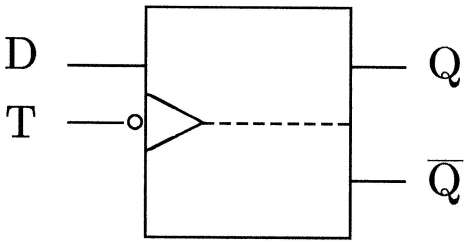
Getaktetes RS-Flipflop



JK-Flipflop

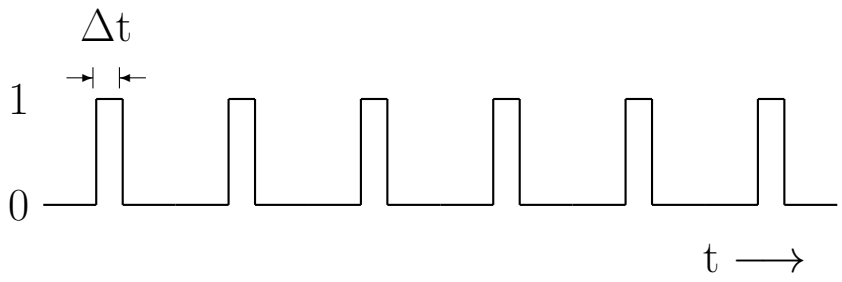


D-Flipflop

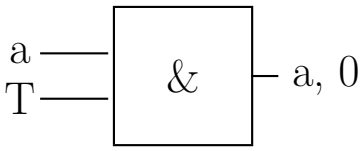


Getaktetes RS-Flipflop

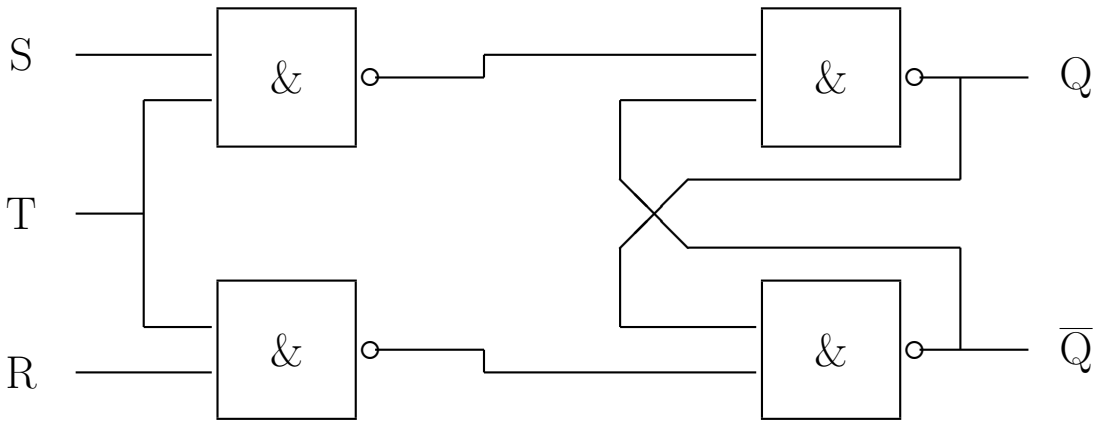
Takt



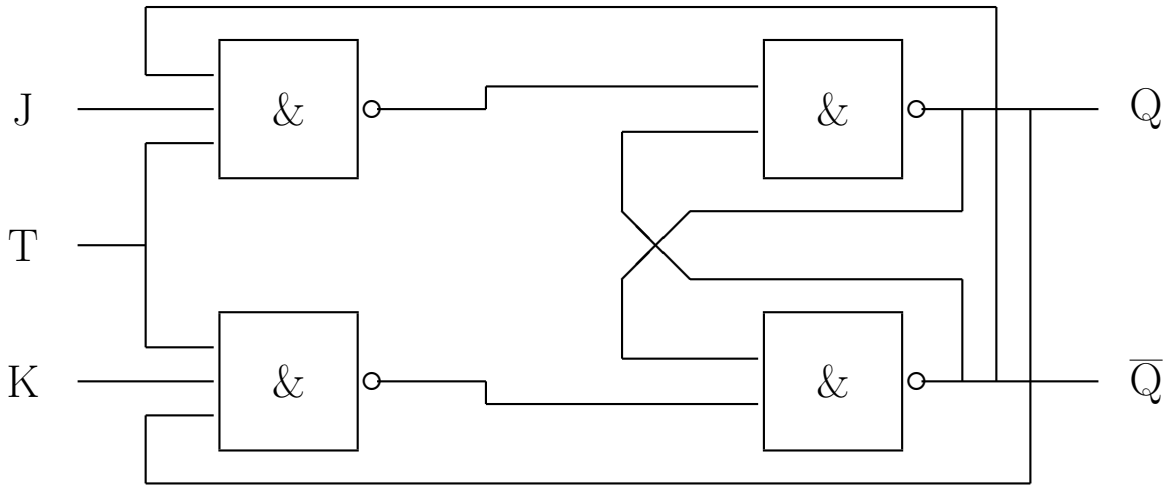
Getakteter Eingang



Getaktetes RS-Flipflop



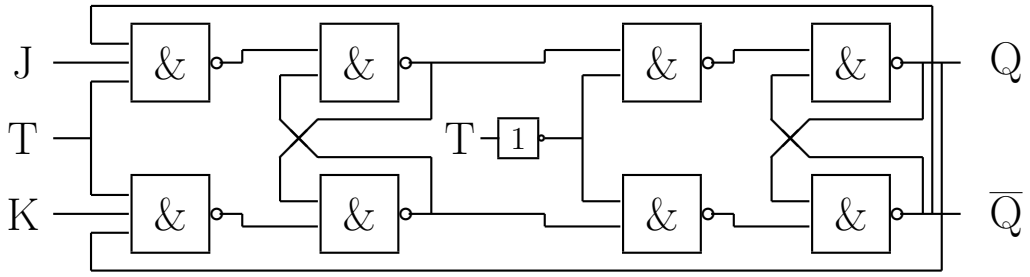
JK-Flipflop



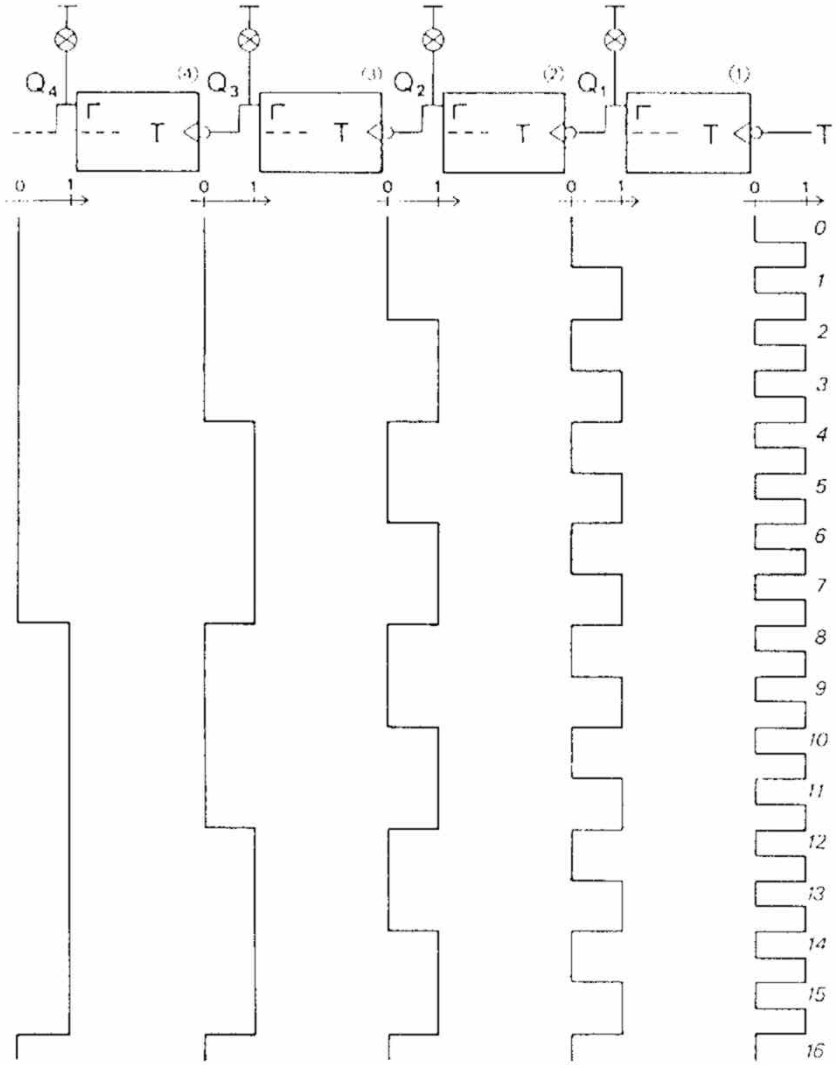
Komprimierte Wahrheitstabelle

J	K	Q_n
0	0	Q_{n-1}
0	1	0
1	0	1
1	1	$\overline{Q}_{n-1}$

Master Slave JK-Flipflop



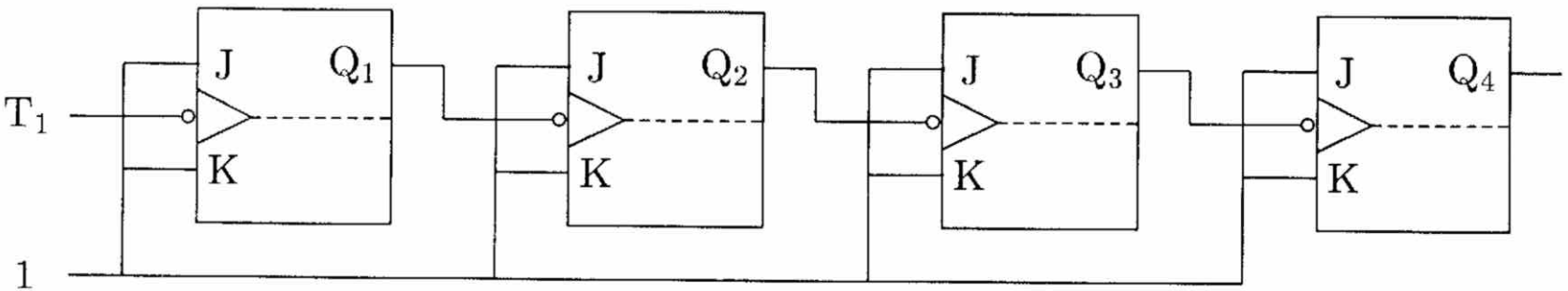
Master Slave



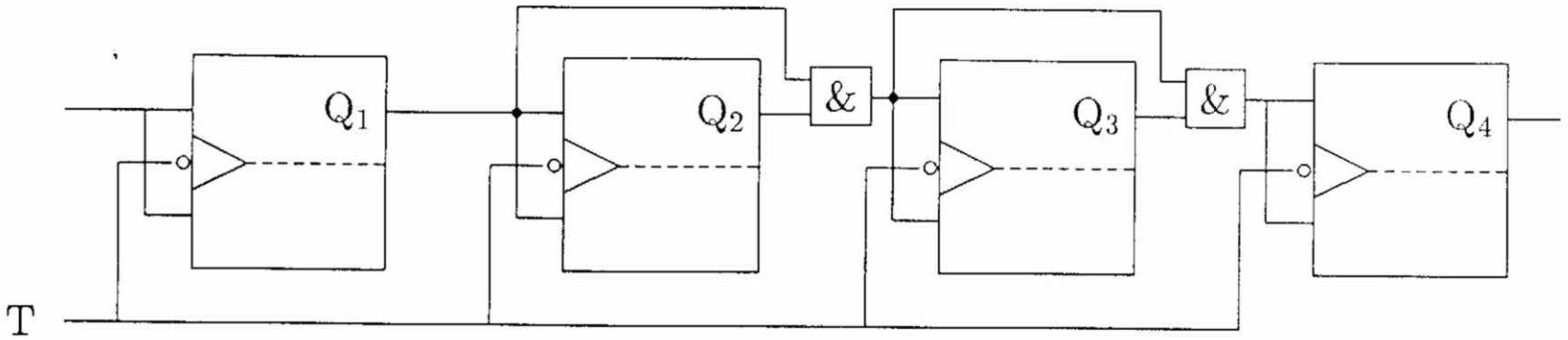
Zähler mit Impulsdiagramm
(Verzögerungen vernachlässigt).

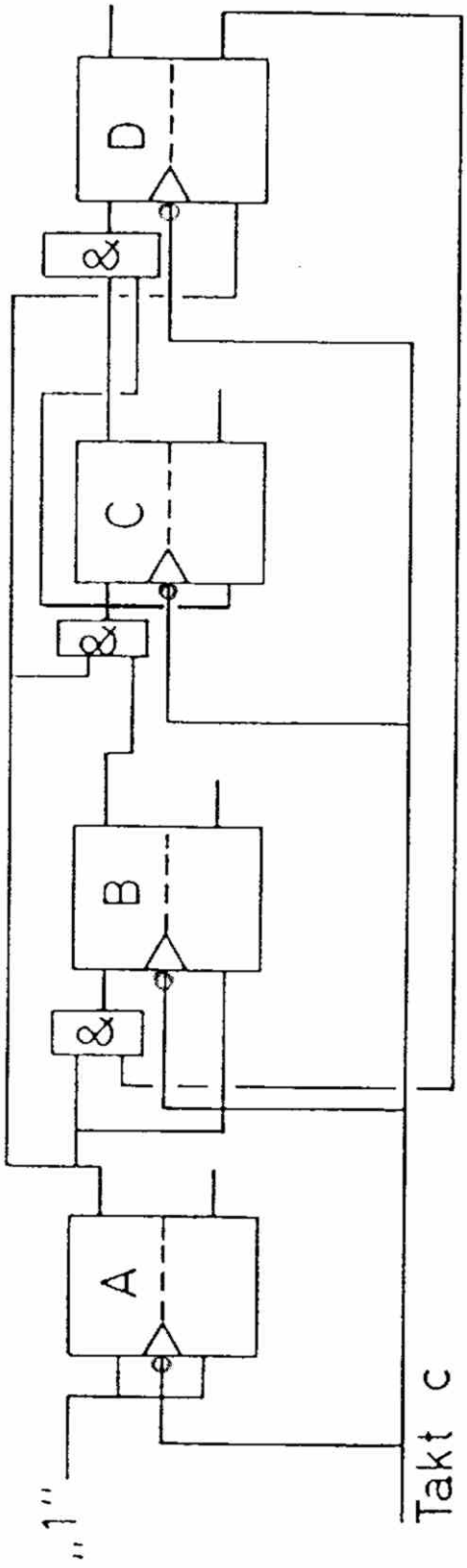
Q_4 8	Q_3 4	Q_2 2	Q_1 1	Anzahl der Takte
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Asynchroner Zähler aus JK-Flipflops



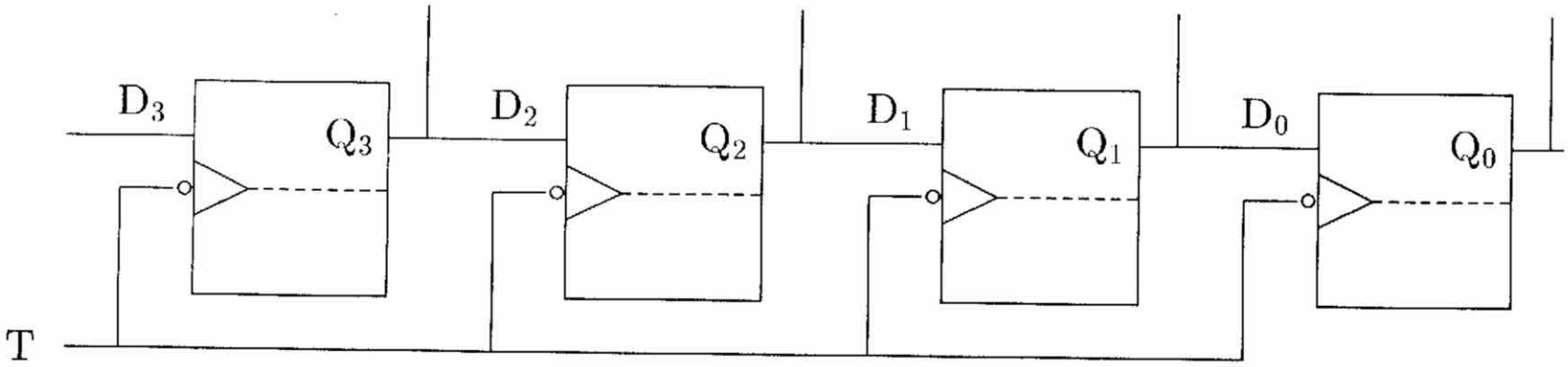
Synchroner Zähler





SYNCHRONER ZÄHLER BIS 9

Schieberegister aus D-Flipflops



4. Struktur und Arbeitsweise eines Rechners

4.1 Grundlegende Komponenten und Operationen

4.1.1 Register

4.1.2 Bus

4.1.3 Einfache Operationen

4.1.4 Steuerung

4.1.5 Hauptspeicher

4.2 Grundlegende Architekturen

4.2.1 von Neumann-Struktur eines Rechners

4.2.2 Memory mapped IO

4.3 Einige Mikroprozessoren

4.3.1 Intel 4004

4.3.2 Rockwell 6502

4.3.3 ARM

4.4 Architekturen: CISC und RISC

4.4.1 Vergleich CISC und RISC

4.4.2 Pipeline

4.4.3 Hazards: Probleme bei der Pipelineverarbeitung

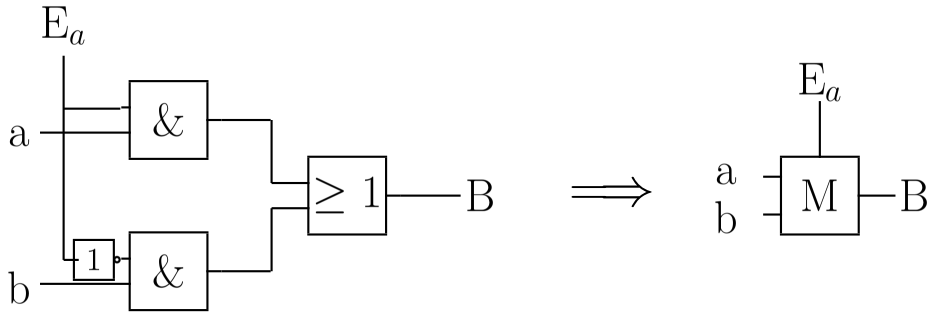
4.4.4 Alpha 21x64

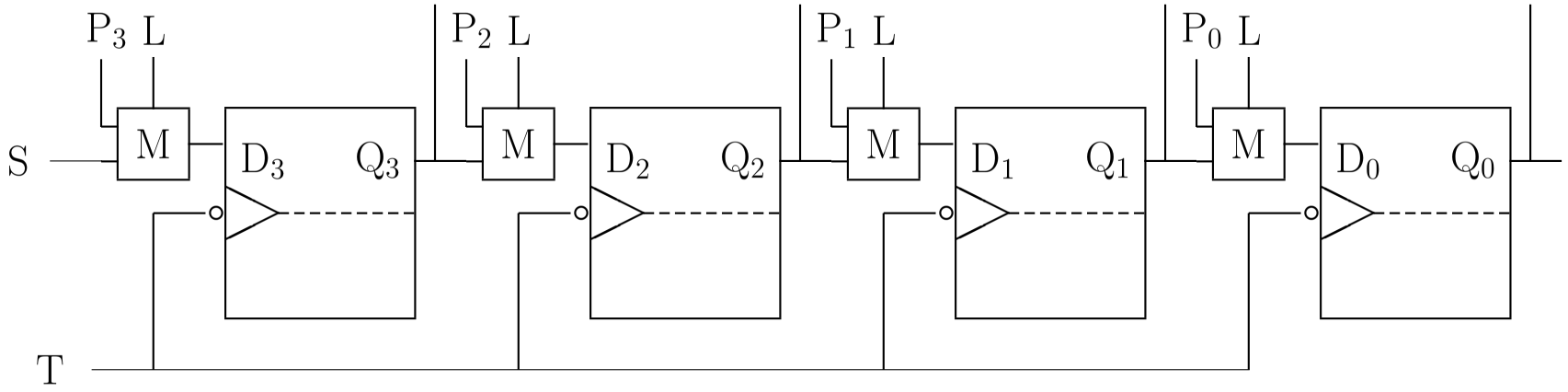
4.5 Neue Architekturen

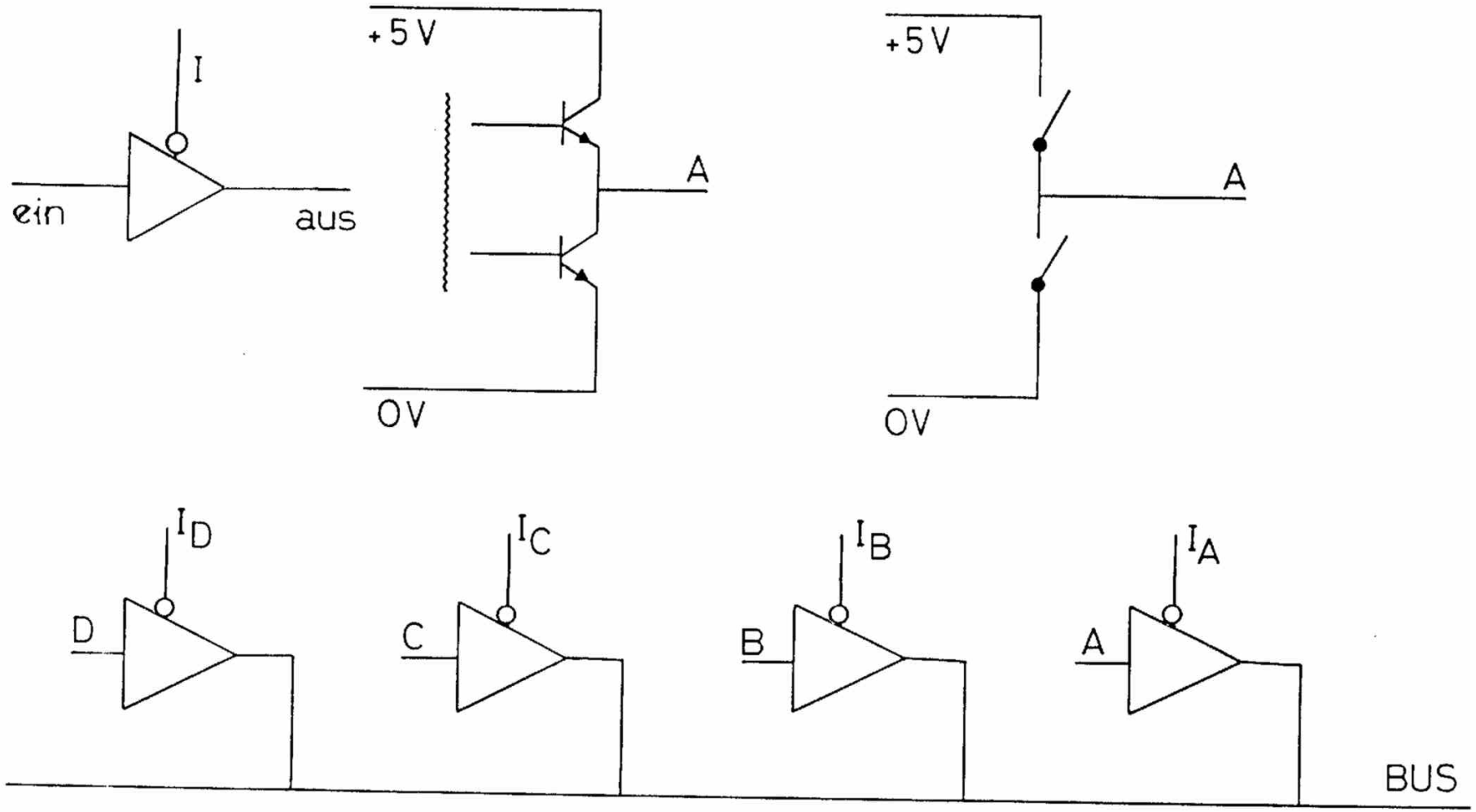
4.5.1 EPIC

4.5.2 MPP mit GPUs

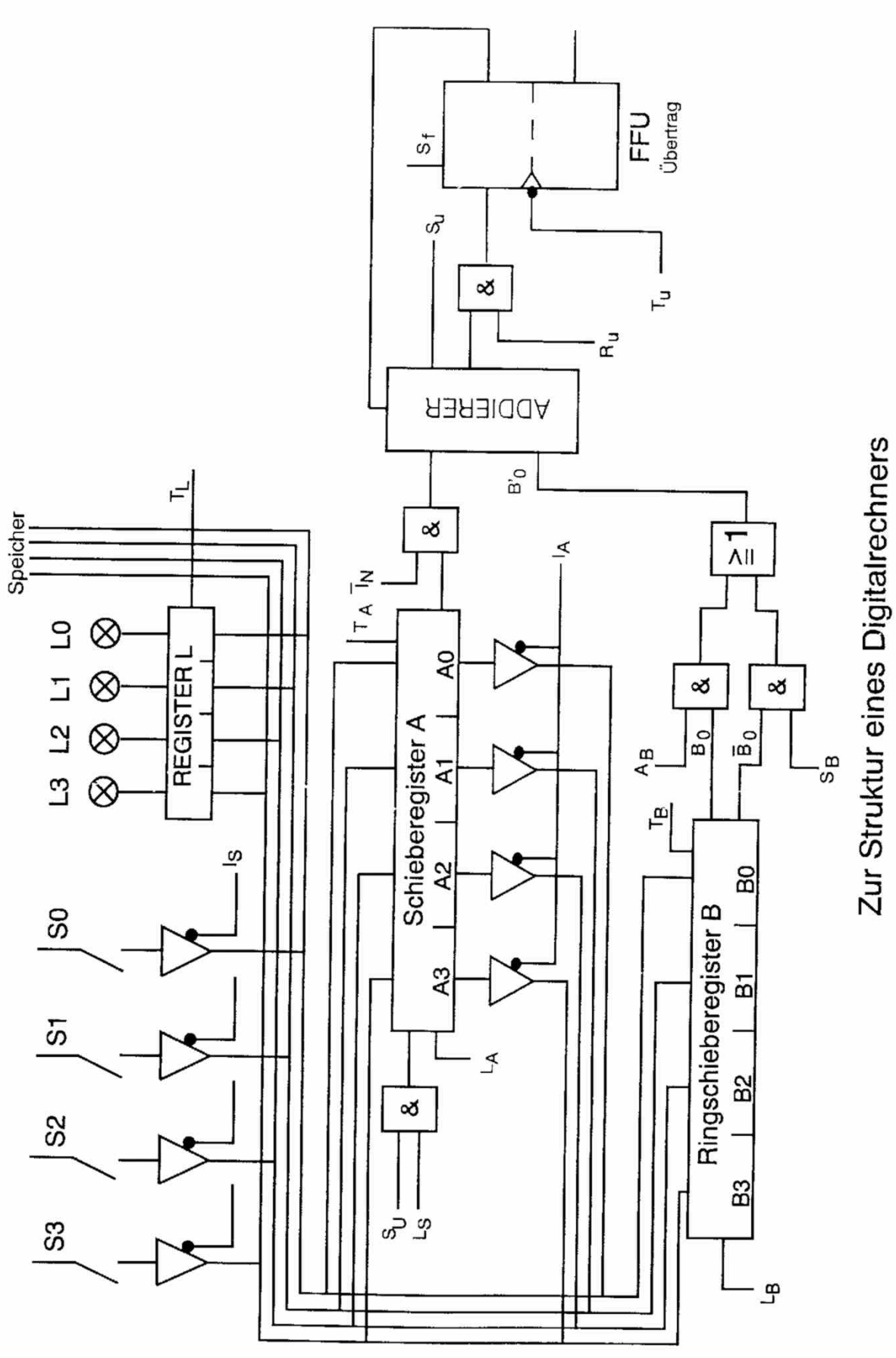
Schieberegister mit parallelem Laden







TRISTATE - TREIBER

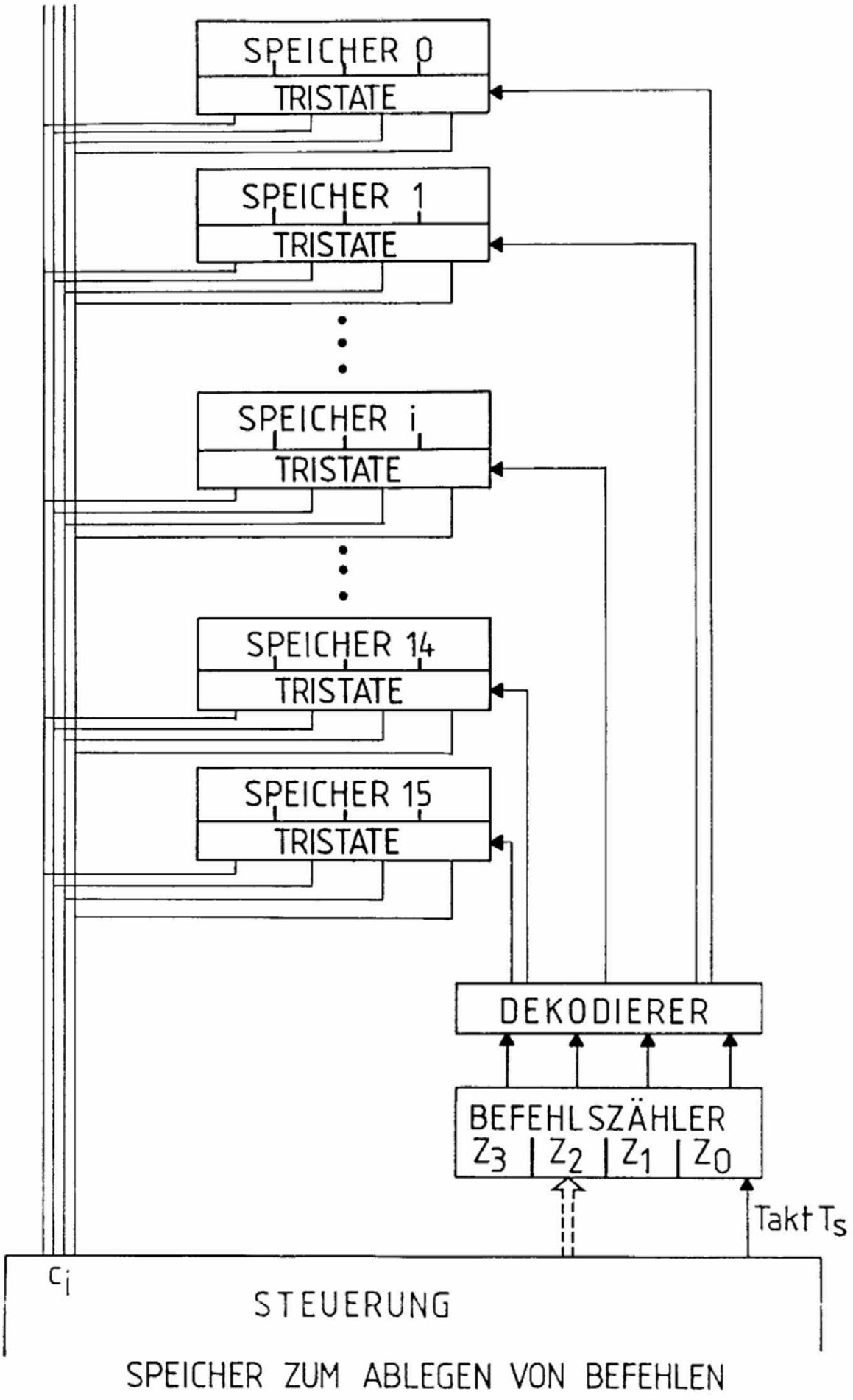


Zur Struktur eines Digitalrechners

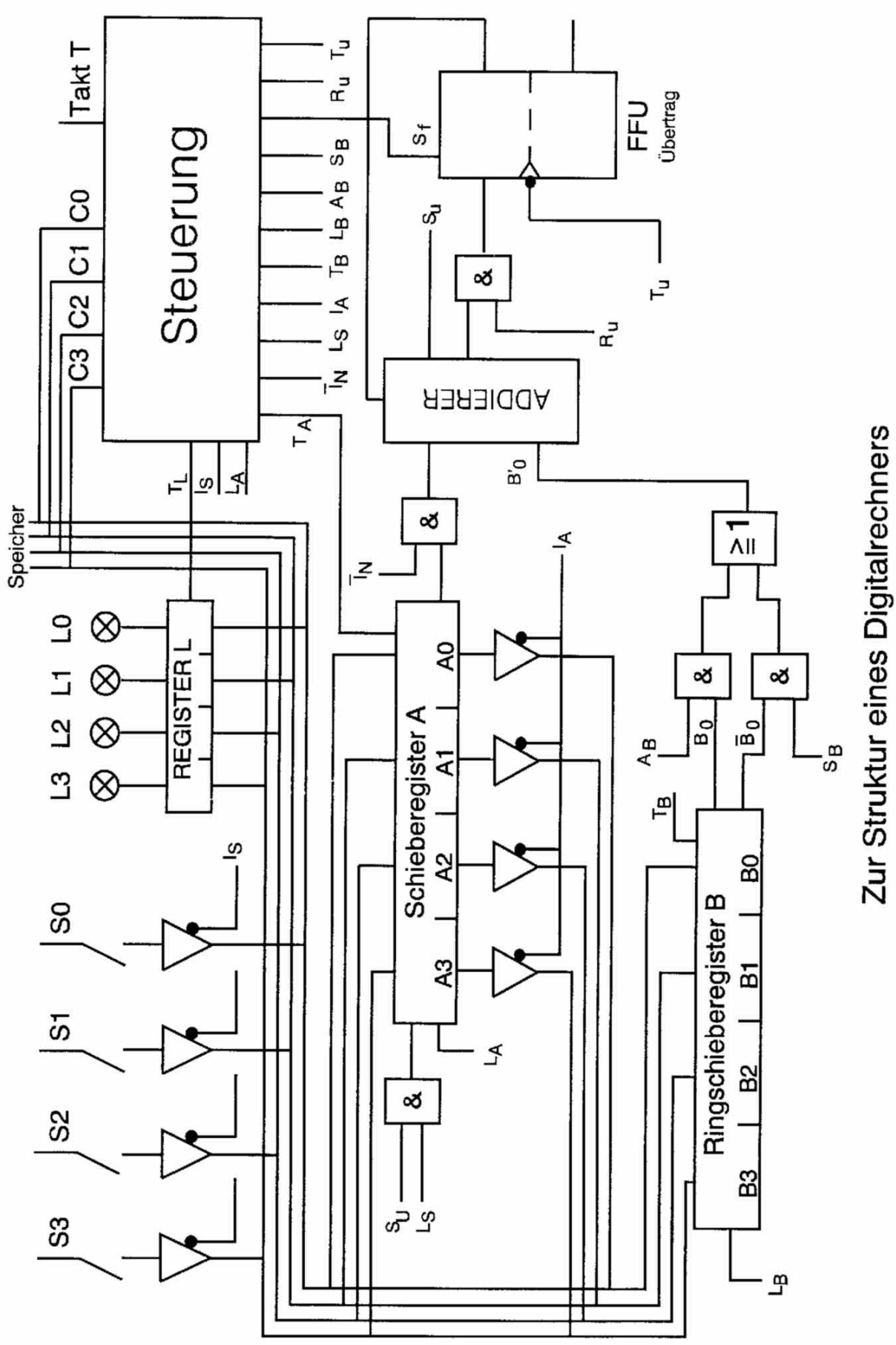
Wahrheitstabelle für Steuerwerk

Operationskode und Steuersignale

Funktion	Mnemonic	$c_2c_1c_0$	$\overline{I_S}$	L_A	L_S	$\overline{I_A}$	A_B	S_B	R_U	S_F	T_A	T_B	T_U	T_L	L_B	$\overline{I_N}$
$S \rightarrow A$	IN	0 0 0	1	1							1					
$A \rightarrow L$	OUT	0 0 1				1								1		
$A \rightarrow B$	MOV	0 1 0				1						1			1	
$A + B \rightarrow A$	ADD	0 1 1			1		1		1		1	1	1			1
$A - B \rightarrow A$	SUB	1 0 0			1			1	1	1	1	1	1			1
$A + 1 \rightarrow A$	INC	1 0 1			1				1	1	1		1			1
$0 \rightarrow A, FF_U$	CLR	1 1 0									1		1			
$-B \rightarrow A$	NEG	1 1 1			1			1	1	1	1	1	1			

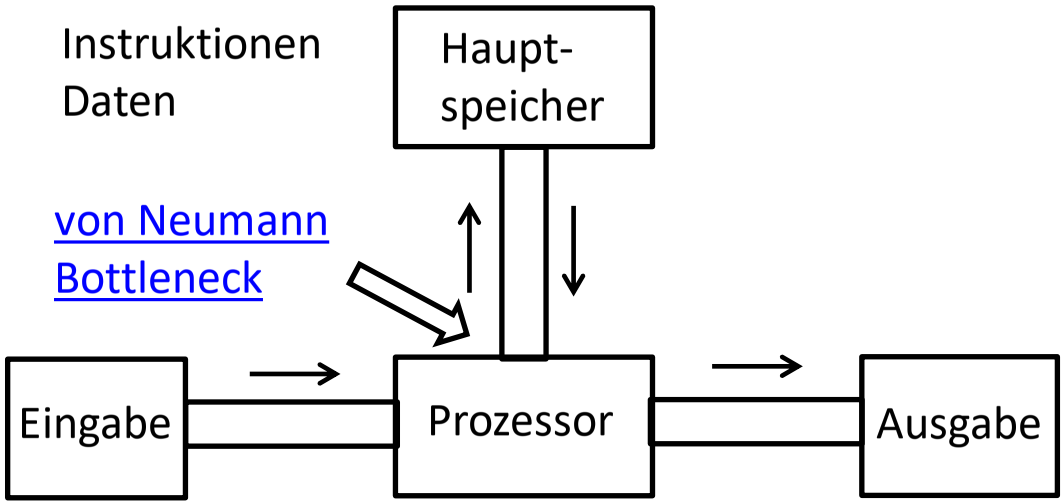


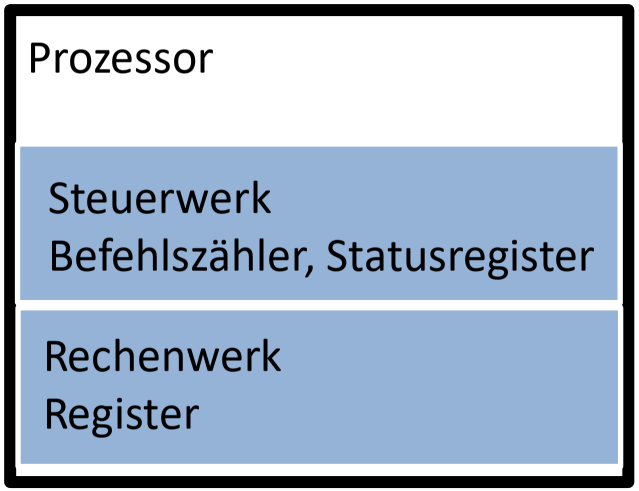
SPEICHER ZUM ABLEGEN VON BEFEHLEN



Zur Struktur eines Digitalrechners

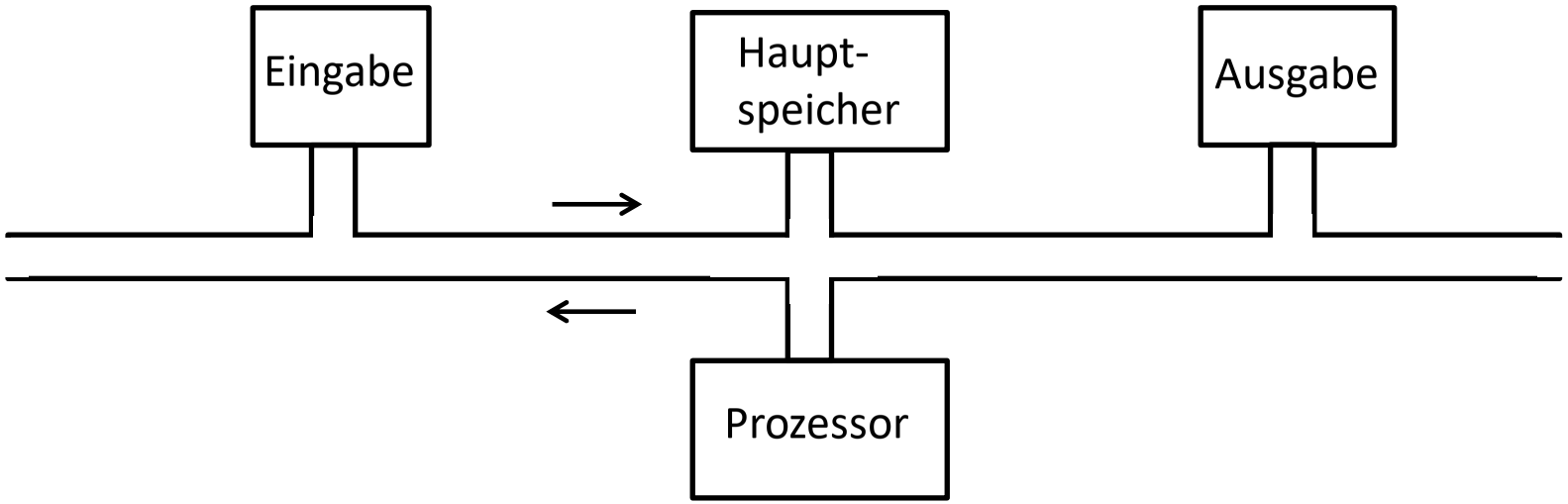
von Neumann Architektur



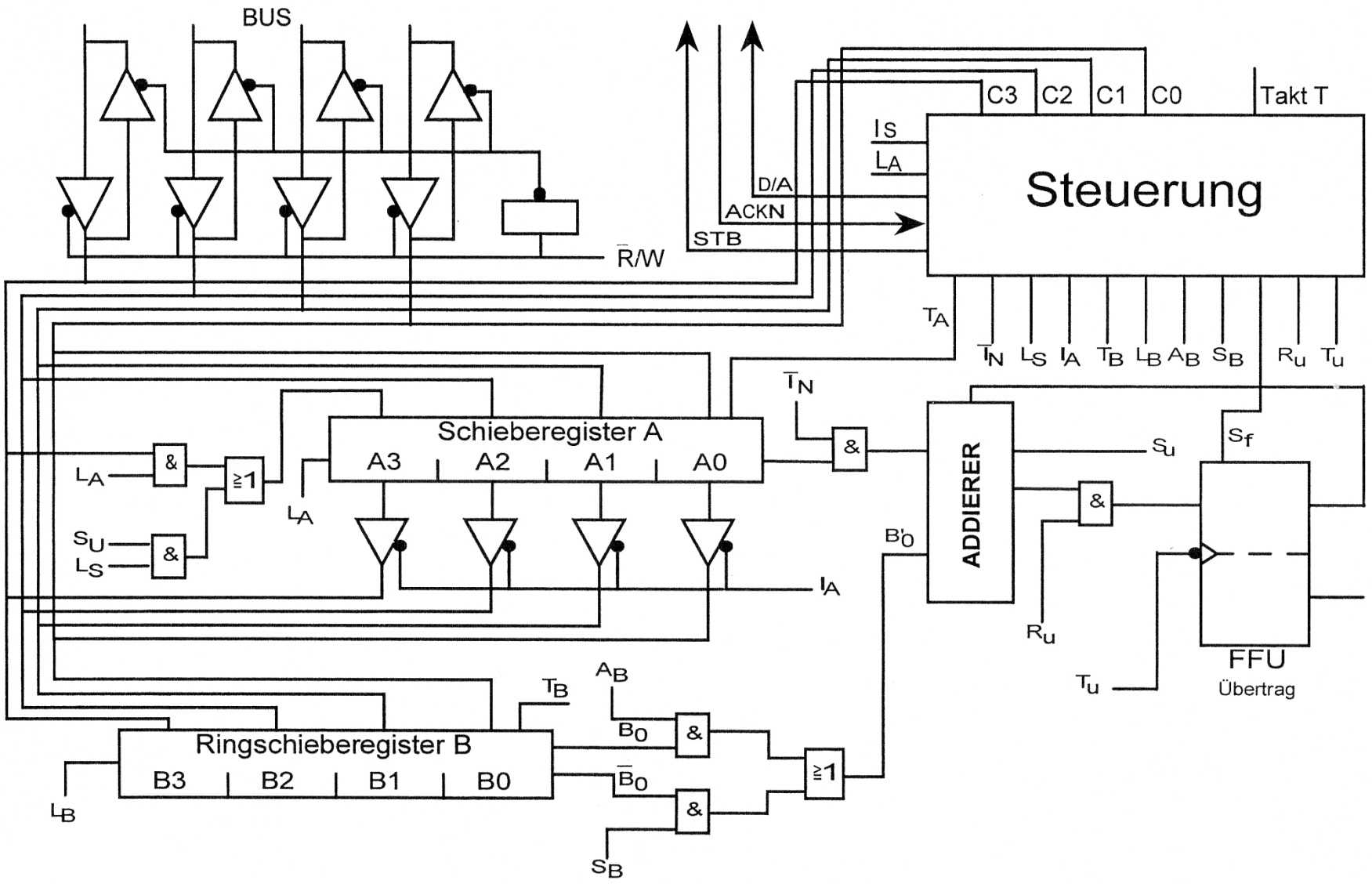


Unibus Architektur

(Unibus mapping, Memory-mapped IO)



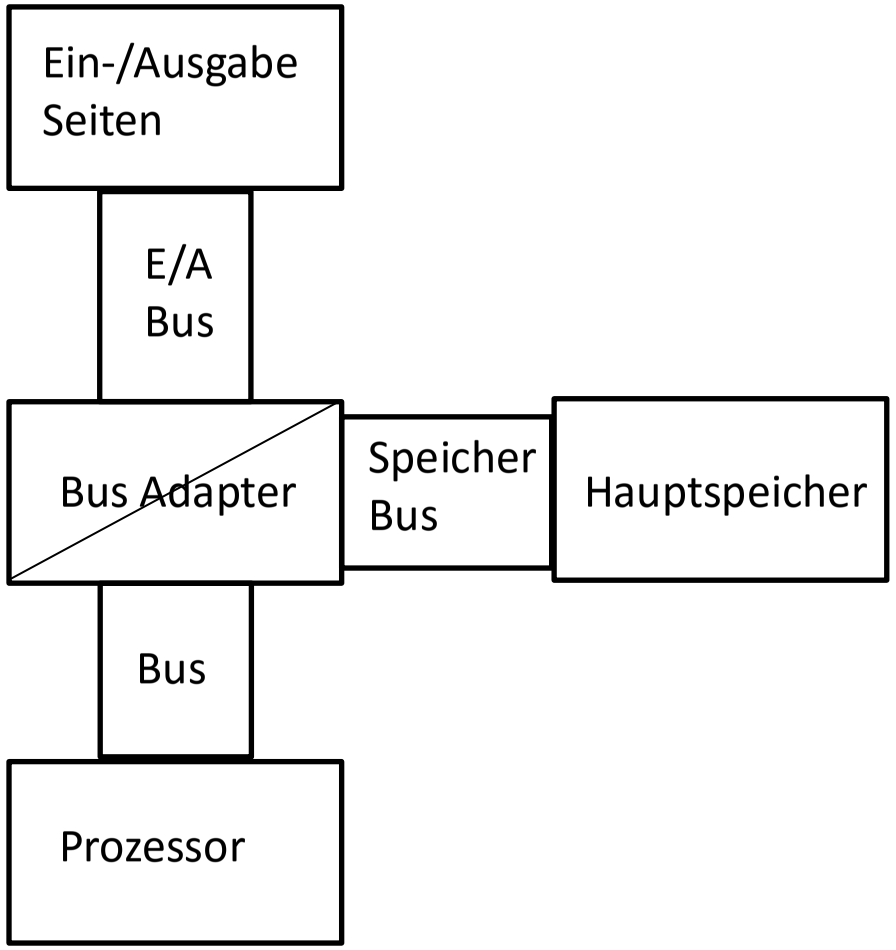
Memory-mapped IO: Adressierung der Register der Ein-/Ausgabe wie Hauptspeicher
Keine Befehle für Ein-/Ausgaben

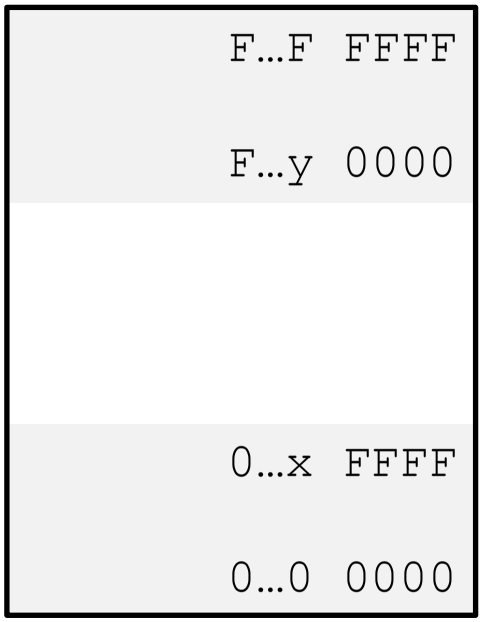


Zur Struktur eines Digitalrechners

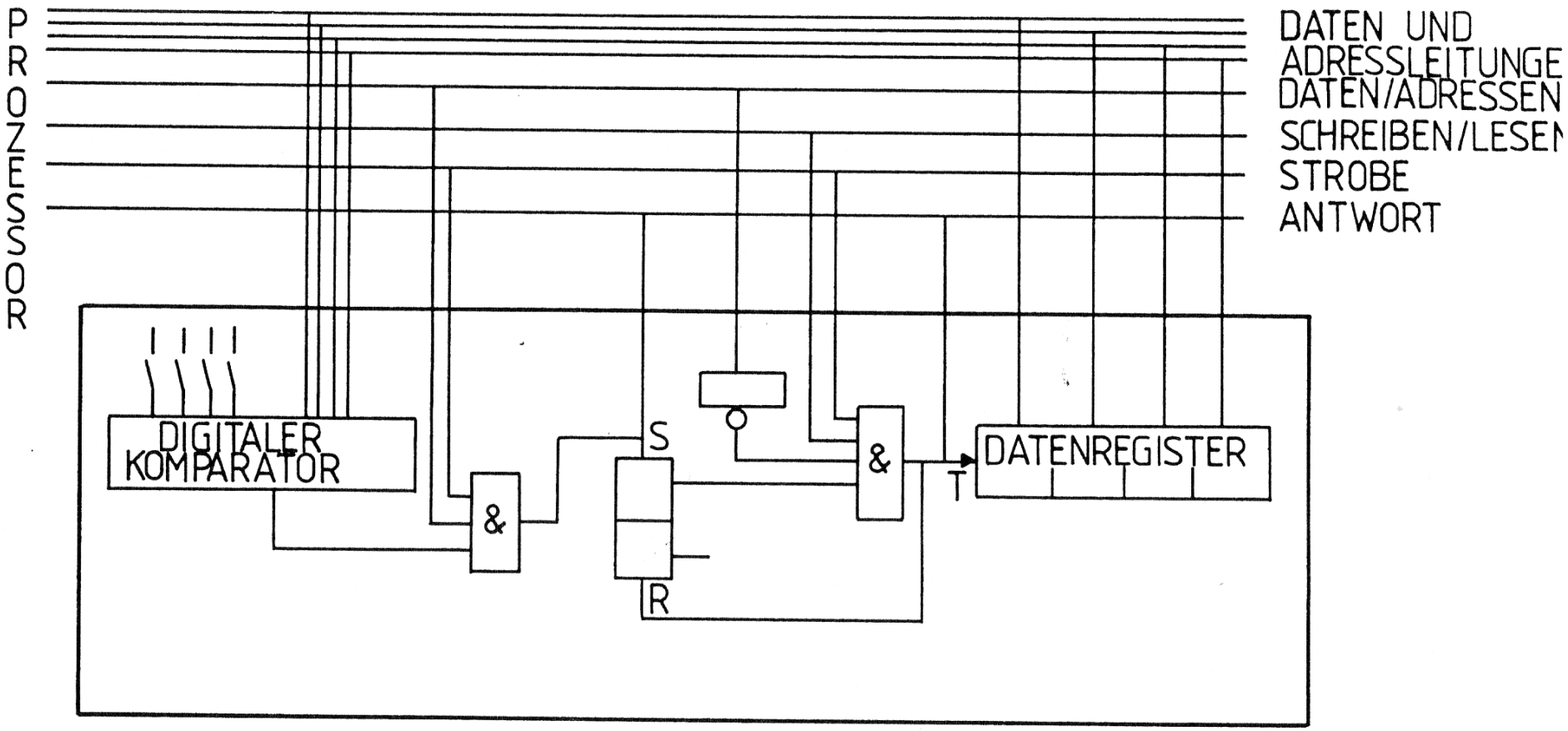
0595IS3B.DRW

Memory-mapped IO



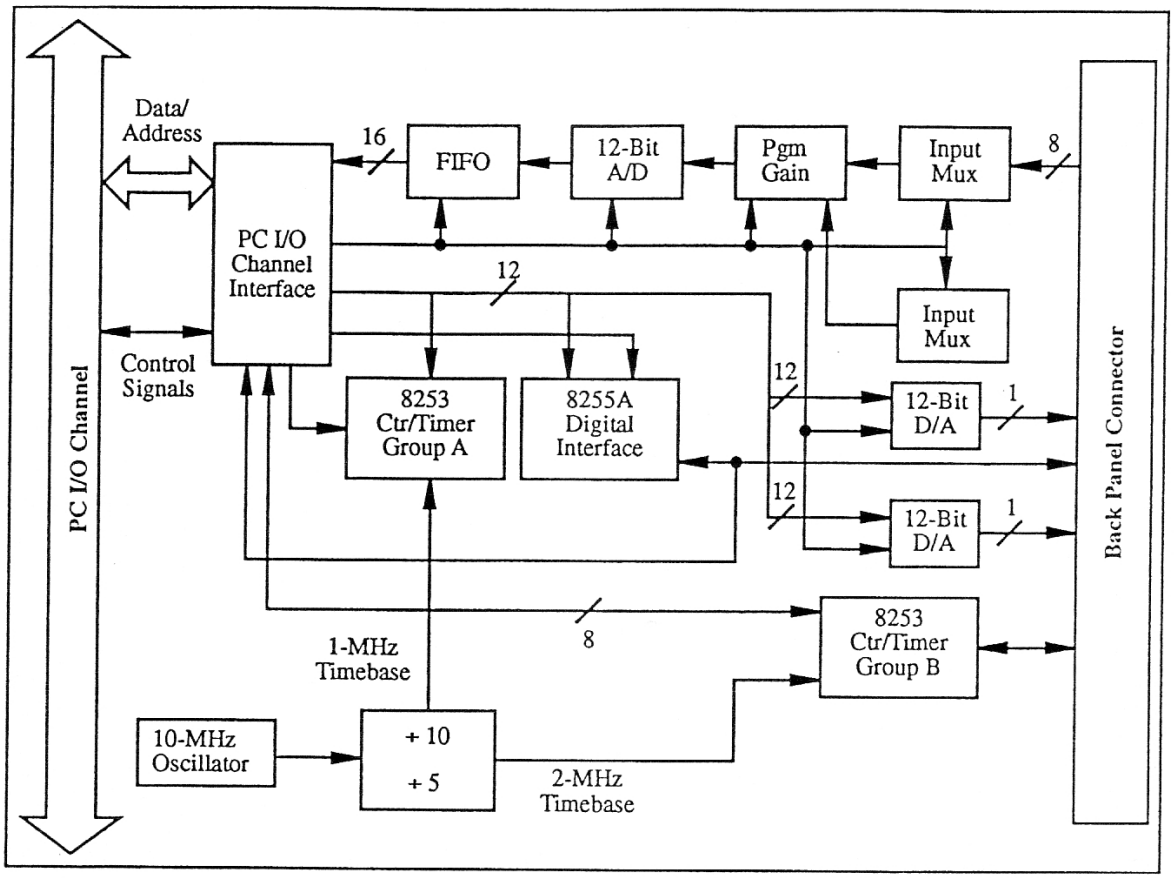


Logischer Adressraum



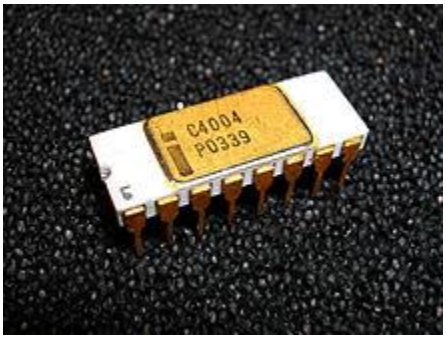
DATENTRANSFER ZUM INTERFACE

Multifunktionsschnittstellenkarte LabPC+

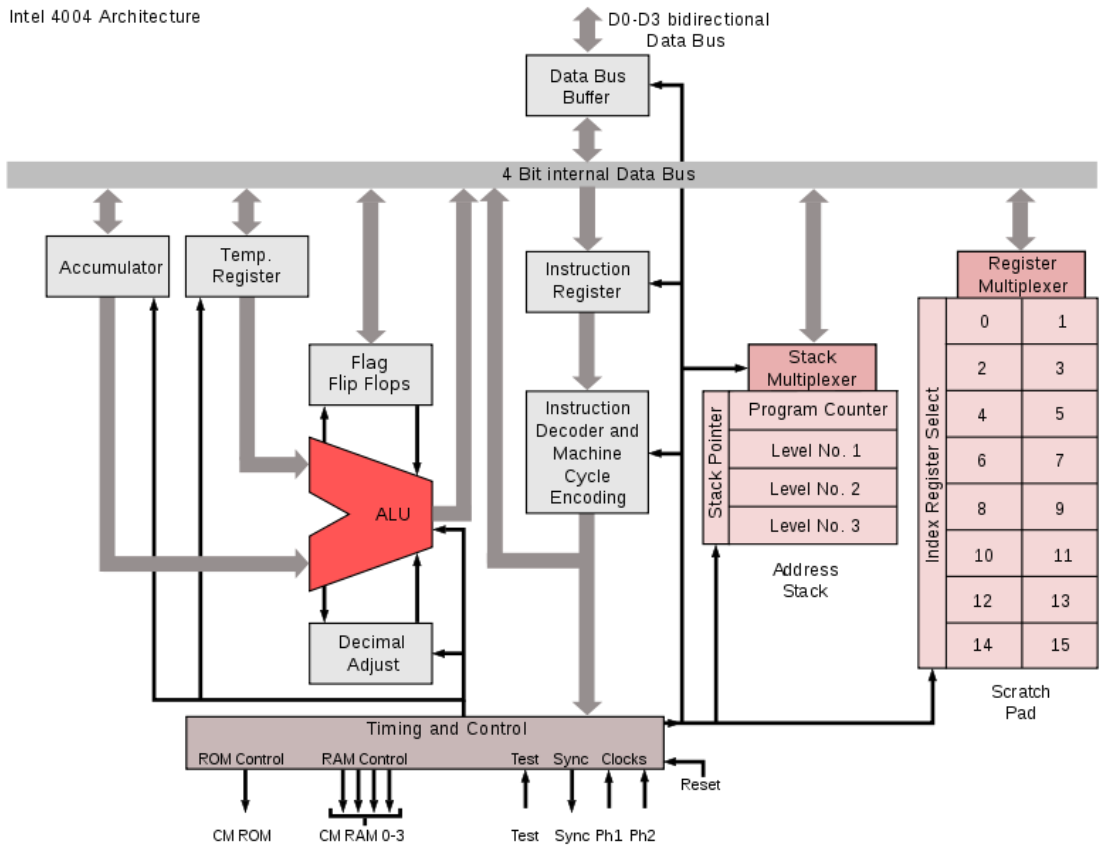


Intel 4004

Der Intel 4004 ist ein 4-Bit-Mikroprozessor des Mikrochipherstellers Intel, der am 15. November 1971 auf den Markt kam. Er gilt als der erste Ein-Chip-Mikroprozessor, der in Serie produziert

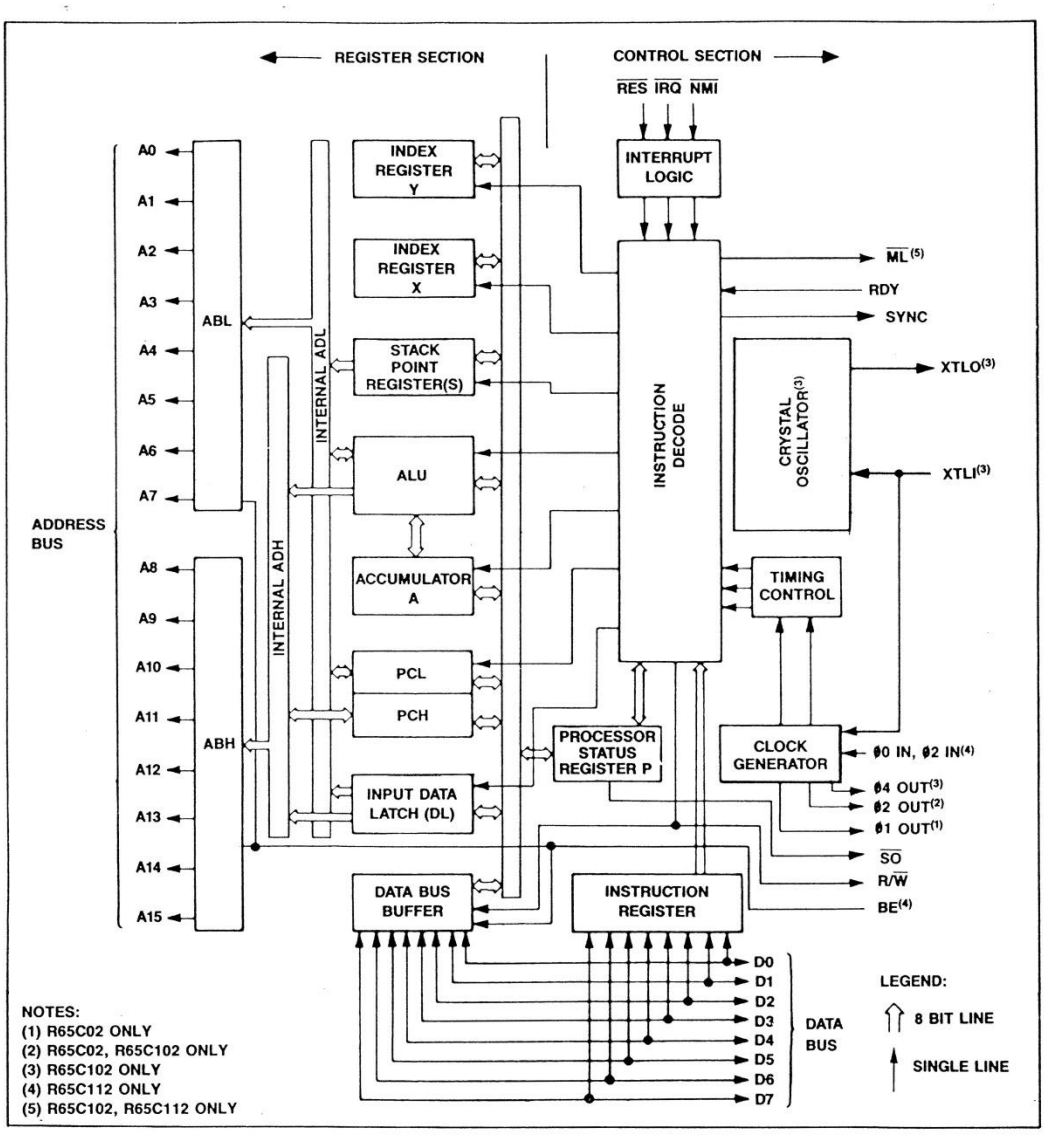


und am freien Markt vertrieben wurde. Meist wird er auch als erster Mikroprozessor überhaupt bezeichnet, was aber nicht richtig ist, da bei Texas Instruments bereits 1968 ein Mikroprozessor als Auftragsarbeit entwickelt wurde, der aber nie in Serie ging.



Rockwell 6502 (1975)

- Taktrate 1- 3 MHz
- 8 Bit Register
  ausser PC (16 Bit) → 16 Bit Adressen 64k Byte Adressraum
- 56 Instruktionstypen; 151 Instruktionen
- Instruktionen 1 – 3 Byte, erstes Byte immer Op-Code
- Ausführungsdauer: 1 Byte Befehl 2 µs
  3 Byte Befehl 7 µs



R65C00 Internal Architecture

INSTRUCTION SET

Table 3 lists the instruction set for the CMOS CPU family in alphabetic order according to mnemonic. Table 4 lists the hexadecimal codes for each of the instructions that are new to the CMOS family and were not available in the NMOS R6502 device family. Table 5 lists those instructions that were available on the NMOS family, but have been assigned new addressing modes in the CMOS CPU family.

Table 3. Alphabetic Listing of Instruction Set

Mnemonic	Function	Mnemonic	Function
(2) ADC	Add Memory to Accumulator with Carry	NOP	No Operation
(2) AND	"AND" Memory with Accumulator		
ASL	Shift Left One Bit (Memory or Accumulator)	(2) ORA	"OR" Memory with Accumlator
(1) BBR	Branch on Bit Reset	PHA	Push Accumulator on Stack
(1) BBS	Branch on Bit Set	PHP	Push Processor Status on Stack
BCC	Branch on Carry Clear	(1) PHX	Push X Register on Stack
BCS	Branch on Carry Set	(1) PHY	Push Y Register on Stack
BEQ	Branch on Result Zero	PLA	Pull Accumulator from Stack
(2) BIT	Test Bits in Memory with Accumulator	PLP	Pull Processor Status from Stack
BMI	Branch on Result Minus	(1) PLX	Pull X Register from Stack
BNE	Branch on Result not Zero	(1) PLY	Pull Y Register from Stack
BPL	Branch on Result Plus		
(1) BRA	Branch Always	(1) RMB	Reset Memory Bit
BRK	Force Break	ROL	Rotate One Bit Left (Memory or Accumulator)
BVC	Branch on Overflow Clear	ROR	Rotate One Bit Right (Memory or Accumulator)
BVS	Branch on Overflow Set	RTI	Return from Interrupt
		RTS	Return from Subroutine
CLC	Clear Carry Flag		
CLD	Clear Decimal Mode	SBC	Subtract Memory from Accumulator with Borrow
CLI	Clear Interrupt Disable Bit	SEC	Set Carry Flag
CLV	Clear Overflow Flag	SED	Set Decimal Mode
(2) CMP	Compare Memory and Accumulator	SEI	Set Interrupt Disable Status
CPX	Compare Memory and Index X	(1) SMB	Set Memory Bit
CPY	Compare Memory and Index Y	(2) STA	Store Accumulator in Memory
		STX	Store Index X in Memory
(2) DEC	Decrement Memory by One	STY	Store Index Y in Memory
DEX	Decrement Index X by One	(1) STZ	Store Zero
DEY	Decrement Index Y by One		
		TAX	Transfer Accumulator to Index X
(2) EOR	"Exclusive-OR" Memory with Accumulator	TAY	Transfer Accumulator to Index Y
		(1) TRB	Test and Reset Bits
(2) INC	Increment Memory by One	(1) TSB	Test and Set Bits
INX	Increment Index X by One	TSX	Transfer Stack Pointer to Index X
INY	Increment Index Y by One	TXA	Transfer Index X to Accumulator
		TXS	Transfer Index X to Stack Register
(2) JMP	Jump to New Location	TYA	Transfer Index Y to Accumulator
JSR	Jump to New Location Saving Return Address		
(2) LDA	Load Accumulator with Memory		
LDX	Load Index X with Memory		
LDY	Load Index Y with Memory		
LSR	Shift One Bit Right (Memory or Accumulator)		

Notes:
(1) Instruction not available on the NMOS family.
(2) R6502 instruction with additional addressing mode(s).

Beispiele von Adressierungsmodi

Immediate Addressing 2 Byte

LDA #$3F # immediate, $ hexadezimal

Lade die Konstante $3F_H$ in den Akkumulator

Absolute Addressing 3Byte

LDA $12BA

Lade den Inhalt von Speicheradresse $12BA_H$ in den Akkumulator

Absolute Indexed Addressing 3Byte

LDA $12BA, X

Lade den Inhalt von Speicheradresse, gegeben durch $12BA_H$ + Inhalt von Indexregister X, in den Akkumulator

ARM-Architektur

Die ARM-Architektur ist ein 1983 vom britischen Computerunternehmen Acorn entwickeltes 32-Bit-Chip-Design. Das Unternehmen begann die Entwicklung eines leistungsfähigen Prozessors für einen Nachfolger seines bis dahin auf dem **6502** basierenden Computer. **ARM** steht für Advanced RISC Machines.

Das Unternehmen ARM Limited stellt keine eigenen Elektronikchips her, sondern vergibt unterschiedliche Lizenzen an Halbleiterhersteller. Die Vielzahl dieser Lizenznehmer und verschiedene Vorteile der Architektur (z. B. geringer Energiebedarf) führten dazu, dass ARM-Chips im Embedded-Bereich die meistgenutzte Architektur sind. Fast alle derzeitigen Smartphone und Tablet-Computer haben beispielsweise einen oder mehrere lizenzierte ARM-Prozessoren.

Nachdem der ARM2 1989 zum ARM3 (mit Cache und höherer Taktfrequenz) weiterentwickelt worden war und immer mehr Unternehmen Interesse an diesen Prozessoren bekundet hatten, gründete Acorn im Jahre 1990 zusammen mit Apple und VLSI Technology das Unternehmen Advanced RISC Machines Ltd. mit Sitz in Großbritannien, welches später in ARM Ltd. umbenannt wurde.

Übersicht

Architektur	Familie	Erscheinungsjahr	Takt
ARMv1	ARM1	1985	4 MHz
ARMv2	ARM2, ARM3	1986, 1989	8–25 MHz
ARMv3	ARM6, ARM7	1991, 1993	12–40 MHz
ARMv4	ARM7TDMI, ARM8, StrongARM ARM9TDMI	1995, 1997	16,8–72 MHz, 203–206 MHz 180 MHz
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale	2002	104–369 MHz 133–1250 MHz
ARMv6	ARM11, ARM Cortex-M0, ARM Cortex-M1	2002 ?	400–772 MHz 170 MHz
ARMv7	ARM Cortex-M1, ARM Cortex-M2, ARM Cortex-M3, ARM Cortex-M4 ARM Cortex-A, ARM Cortex-R	? 2004 2005 ?	bis 200 MHz[1] ? bis 2 GHz ?
ARMv8	ARM Cortex-A (A53, A57, A72, A35, A73)[15] ARM Cortex-R (nur AArch32) ARM Cortex-M (nur AArch32)	2012 ? ?	1,2–3,0[16] GHz ? ?

Quelle: Wikipedia: ARM-Architektur

Vergleich CISC ⇔ RISC

Complex **I**nstruction **S**et **C**omputer
Vertreter: 6502, PDP11, VAX11
Oft Implementierung des Orthogonal Instruction Set

Anfang ’70-iger: Semantische Lücke; die meisten komplexen Instruktionen werden von Compilern nicht oder nur wendig genutzt.

Reduced **I**nstruction **S**et **C**omputer auch Load/Store Architektur genannt
Vertreter: Alpha

Unterscheiden sich nicht durch die Zahl der Instruktion sondern durch die Pipeline Verarbeitung der Befehle. Die Pipeline hat eine fixe Anzahl von Schritten. Dadurch sind komplexe Adressierungsmodi ausser bei Load oder Store Operationen nicht möglich.

Beispiel CISC

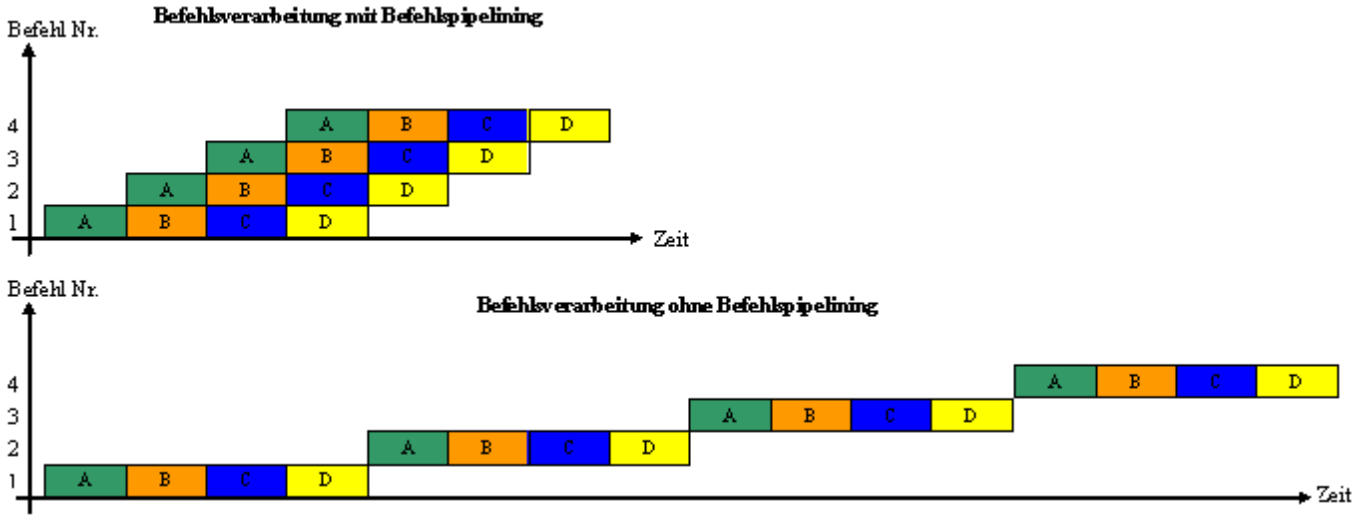
ADD	adr1, adr2, adr3	einfache Adressierung
ADD	@(adr1), @(adr2), @(adr3)	komplexe Adressierung

Beispiel RISC

Es gibt separate Load und Store Befehle, die auch komplexe Addressierungsmodi unterstützen. Die anderen Operationen finden ausschliesslich in/zwischen den Registern statt

```
LOA     @(adr1),  R1
LOA     @(adr2),  R2
ADD     R1, R2, R1
STO     R1, @(adr3)
```

Befehlspipelining



A –IF, Instruction Fetch

B –ID, Instruction Decoding

C –EX, Execution

D –WB, Write Back

➔ Wenn jede Phase des Befehls mit einem Taktzyklus durchgeführt wird, wird effektiv ein Befehl pro Takt ausgeführt.

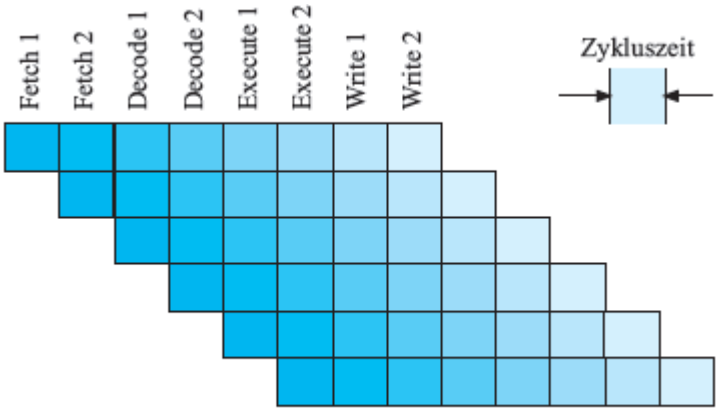
➔ Falls die Phasen mehrere Taktzyklen zur Ausführung benötigen, bestimmt die längste Phase die effektive Ausführungszeit.

Quelle Wikipedia

Beschleunigung der Befehlsbearbeitung

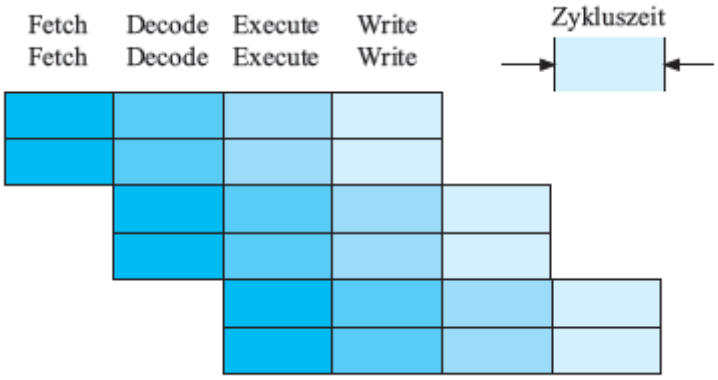
Superpipelining:

- Aufteilen jeder Phase in Teilschritte mit einem Taktzyklus.
- Trotz einer grösseren Zahl von (Teil-)Phasen wird pro Takt effektiv ein Befehl ausgeführt!



Superskalare Architektur:

- Mehrere parallele Pipelines



Quelle: D.W. Hoffmann

Hazards:

Probleme, derart dass effektiv weniger als ein Befehl pro Taktzyklus ausgeführt wird.

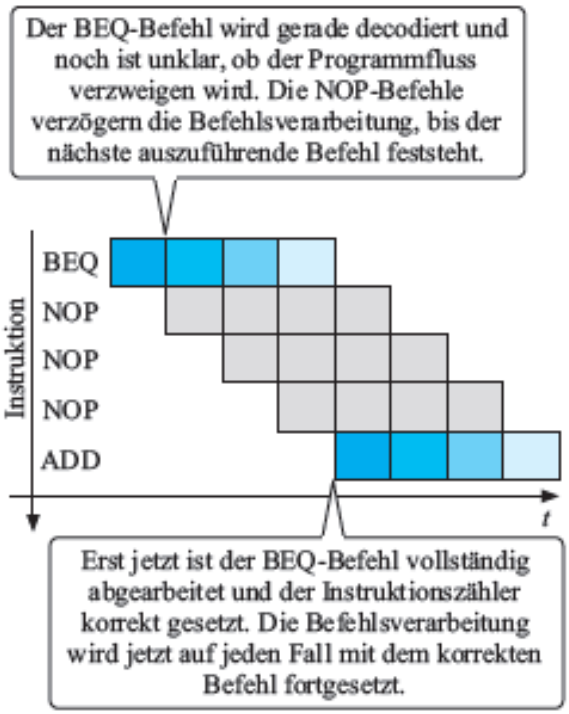
Control Hazard

Probleme bei bedingten Sprüngen

```
start: BEQ R1, R2, else: // if (R1 == R2)
ADD R1, #1;              // R1 = R1 + 1;
JMP end:                 // else
else: ADD R2, #1;        // R2 = R2 + 1;
end: ...
```

Lösen des Problems durch Einschieben von leeren Operationen NOP

```
start: BEQ R1, R2, else:
NOP
NOP
NOP
ADD R1, #1;
JMP end:
NOP
NOP
NOP
else: ADD R2, #1;
end: ...
```

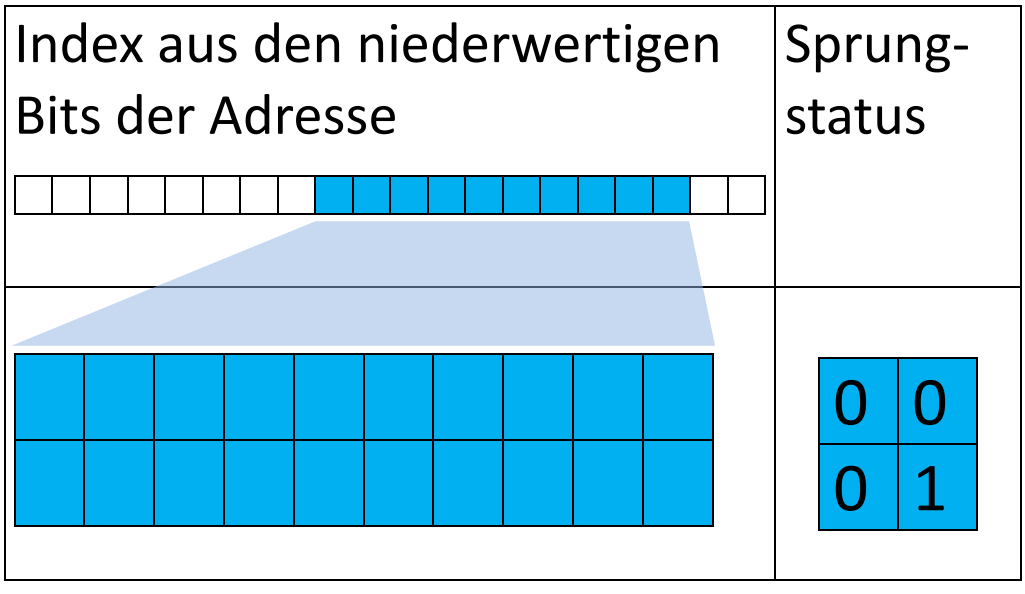


Quelle: D.W. Hoffmann

Lösen des Problems durch Spekulative Befehlsausführung

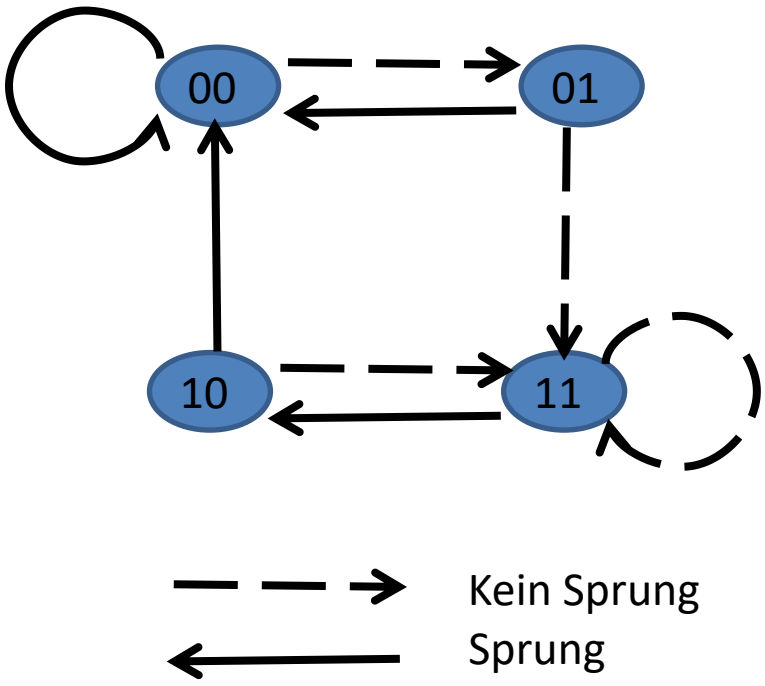
- Nach bedingtem Sprung Einfüllen der Befehle des wahrscheinlicheren Zweiges
  ➔ Trifft die Vorhersage nicht zu, muss die Pipeline entleert und der Prozessorstatus zurückgesetzt werden
- Statische Vorhersage
  ➔ schlechte Trefferquote << 85%
- Dynamische Vorhersage
  Erhöhung der Trefferquote (bis 98%) durch Aufzeichnen des Sprungverhaltens in Branch History Table (BHT)
  ➔ Reaktion auf den ersten Wechsel des Sprung-verhaltens ist zu ineffizient, daher Aufzeichnung zweier (oder mehr) Wechsel

BHT mit 2 Bit Prediction



Sprungstati und Übergänge

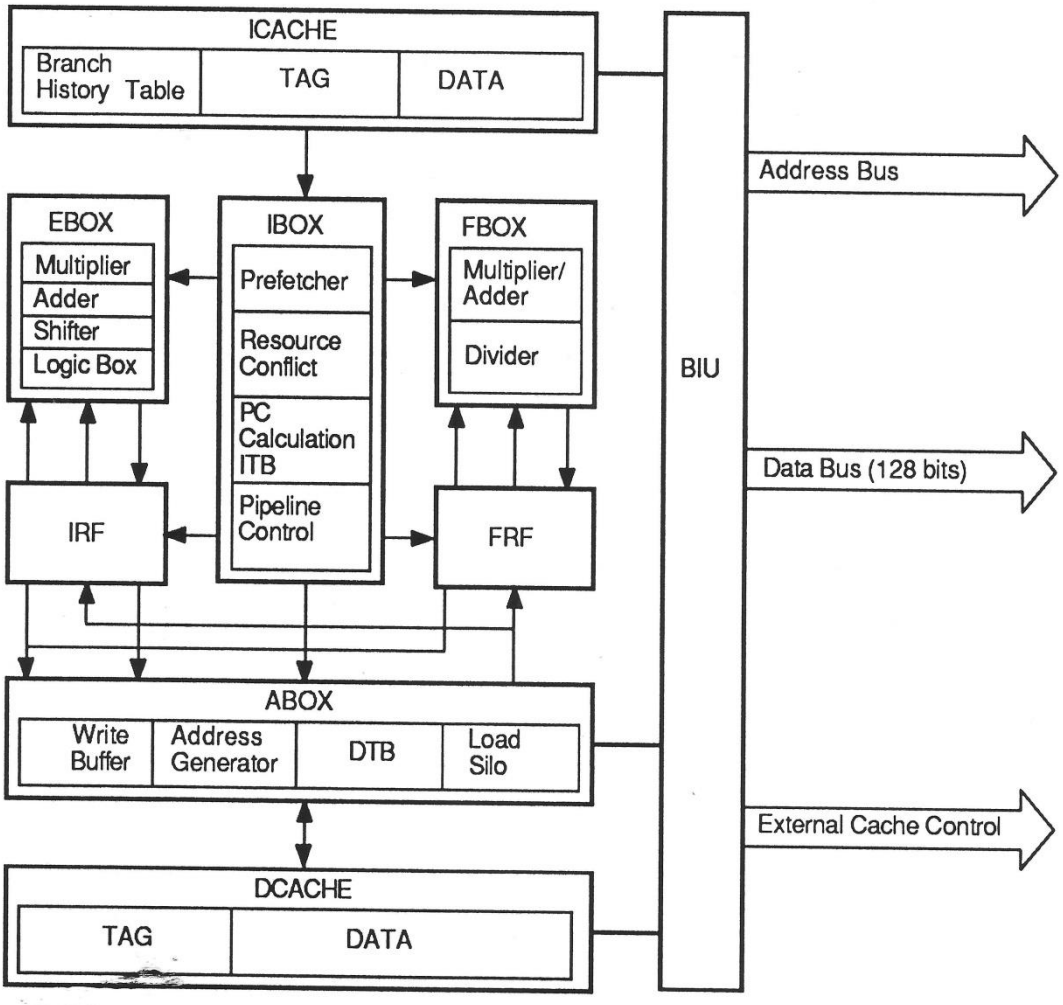
Sprung-status	Bedeutung	Vorhersage springen?
00	Sprung 2 oder mehrere Male ausgeführt	ja
01	Sprung zum ersten Mal nicht ausgeführt	ja
10	Sprung zum ersten Mal wieder ausgeführt	nein
11	Sprung 2 oder mehrere Male nicht ausgeführt	nein



Alpha Prozessor

64-Bit-RISC-Mikroprozessor
Serie von Prozessoren mit Namen Alpha 21x64
der Firma Digital

erster Vertreter 21064 (1992)



Alpha 21064 Floor Plan

(Leiterbahnen 0.75 µm, 1,7 ∗ 10^6 Transistoren, Takt 200 MHz)

Entwicklung der RISC Prozessoren

Beispiel Alpha 21064 → 21164 → 21264 →…

CMOS6 Technologie: Leiterbahnen 0.35 μm, 6 Schichten
3 cm^2 Fläche
$15{,}2 * 10^6$ Transistoren

- weiteres Parallelisieren der Instruktionsbearbeitung:
  mehr Pipelines
  4 Befehle pro Zyklus → Superskalar
  4 Integer Pipelines
  2 Floatingpoint Pipelines

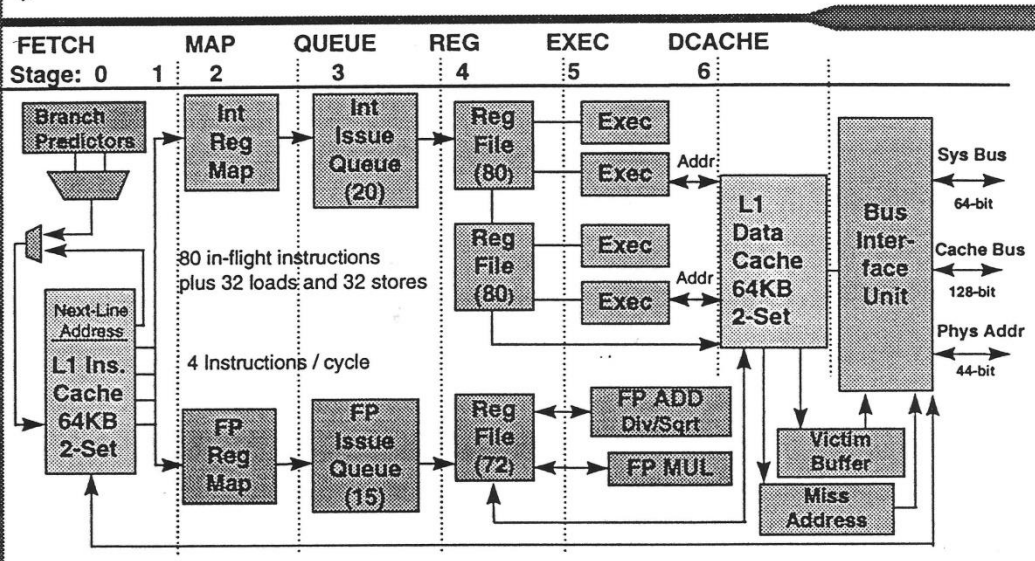
- effizienteres Nutzen der Pipelines
  aufwendigere Branch Prediction
  Out-of-order Execution

- Takt 500 MHz: 4 * 500 MHz → 2000 MIPS
  Später Takt 1250 MHz: 4 * 1250 MHz → 5000 MIPS
  Erhöhung der Geschwindigkeit durch Reduktion der Grösse:
  0.35 μm → 0.25 μm Leiterbahnen

Problem:

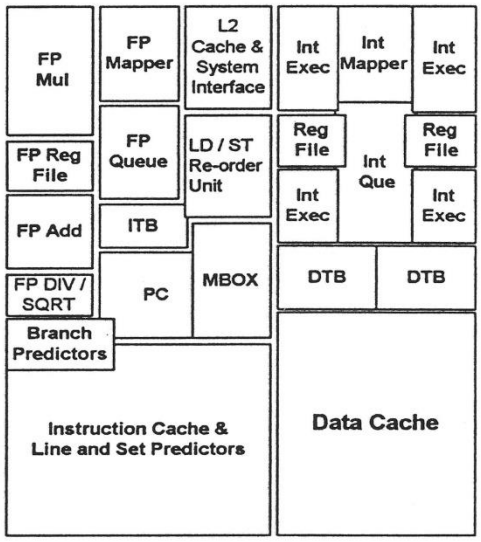
Verteilung der Instruktionen auf Pipelines, Branch Prediction und Out-of-order Execution verursachen einen enormen Aufwand an Logik

Alpha 21264: Block Diagram



digital

Alpha 21264 Floor Plan



digital

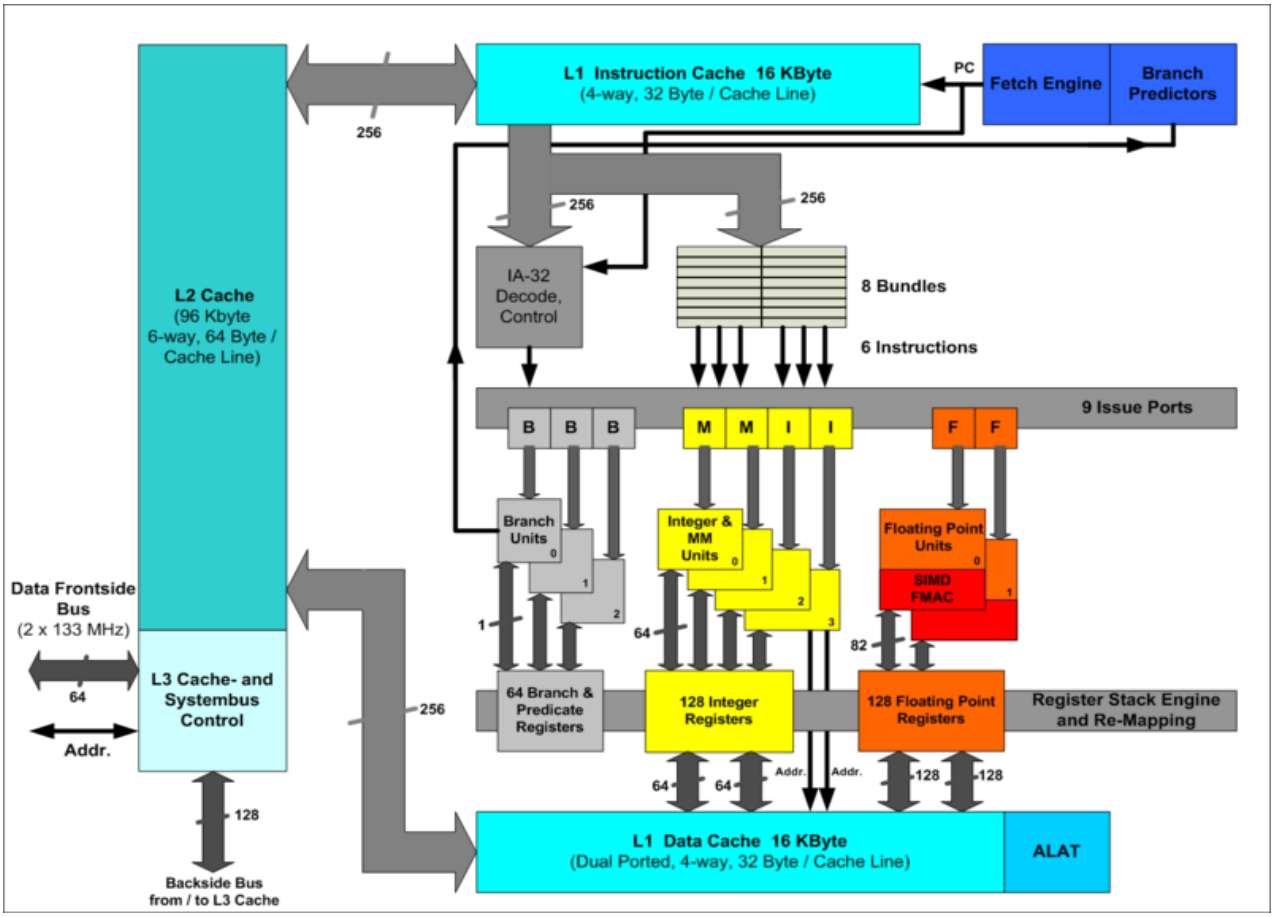
Und wie geht es weiter?

Explicitly **P**arallel **I**nstruction **C**omputing (EPIC)

Vertreter: Itanium

Prozessor speziell konzipiert für die Parallelverarbeitung.
Die Entscheidung über die Parallelisierung wird schon vom Compiler vorgenommen, damit Vereinfachung des Prozessors im Vergleich zur superskalaren Architektur.

VLIW: Bundles je 128 Bit von je 3 Instruktionen



Massively **P**arallel **P**rocessing (MPP)

Häufig implementiert in Vektor-Prozessoren

Architektur ist **S**ingle **I**nstruction **M**ultiple **D**ata (SIMD)
Beispiel: die Addition zweier Vektoren A(i) und B(i)
und Speichern der Summe in Vektor C(i);
Ausführung jeweils paarweise für jedes i

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
int i = threadIdx.x;
C[i] = A[i] + B[i];
}
int main()
{
...
// Kernel invocation with N threads
VecAdd<<<1, N>>>(A, B, C);
...
}
```

Solche vektorisierbare Operationen werden in Grafik-Prozessoren (GPUs) ausgeführt

Speziell angepasste GPUs werden daher für das MPP im High Performance Computing eingesetzt

Einsatz bei Simulationen und Berechnungen wie Computational Fluid Dynamics, Quantum Chemistry, Machine Vision und AI Anwendungen mit Deep Learning

NVIDIA Pascal GP100 GPU

GP100 GPU umfasst:

- 6 Graphics Processing Clusters (GPCs),
- 6 * 5 Texture Processing Clusters (TPCs),
- 6 * 10 Streaming Multiprocessors (SMs),
- 8 512-bit memory controllers (4096 bits total) mit
- 8 * 512 kB L2 Cache.

15,3 Mia. Transistoren

.

Quelle, NVIDIA: pascal-architecture-whitepaper-v1.2

Pascal Streaming Multiprocessor

- 2 Blocks mit je 32 CUDA Cores
- Die Zahl der Cores quantifiziert den Vektorisierungsgrad der entsprechenden Pipeline
- Jeder Core führt einen eigenen Thread von Instruktionen aus (**S**ingle **I**nstruction **M**ultiple **T**hread) mit seperatem PC und Status
- Gruppen von Cores (Warps) führen jeweils den selben Thread von Instruktionen aus aber mit verschiedenen Daten
- Ausführung von 64 FP-Operationen mit einfacher Genauigkeit (32 Bit)
  oder 32 FP-Operationen mit doppelter Genauigkeit (64 Bit)
  oder 64 Paare von FP-Operationen mit halber Genauigkeit (64 Bit)

Die gesamte GP100 GPU kann also theoretisch 60 * 64 = 3840 FP Operationen parallel ausführen! Abhängigkeiten der Befehle und Daten in den verschiedenen Pipelines kann zu starken Reduktion des Durchsatzes führen.

NVIDIA Ampere GA100 GPU

GA100 GPU umfasst:

- 8 Graphics Processing Clusters (GPCs),
- 8 * 8 Texture Processing Clusters (TPCs),
- 8 * 16 Streaming Multiprocessors (SMs),
- 8 * 16 * 4 Tensor Cores,
- 12 * 512-bit Memory Controllers mit
- 40960 kB L2 Cache.

54,2 Mia. Transistoren

Quelle, NVIDIA: nvidia-ampere-architecture-whitepaper

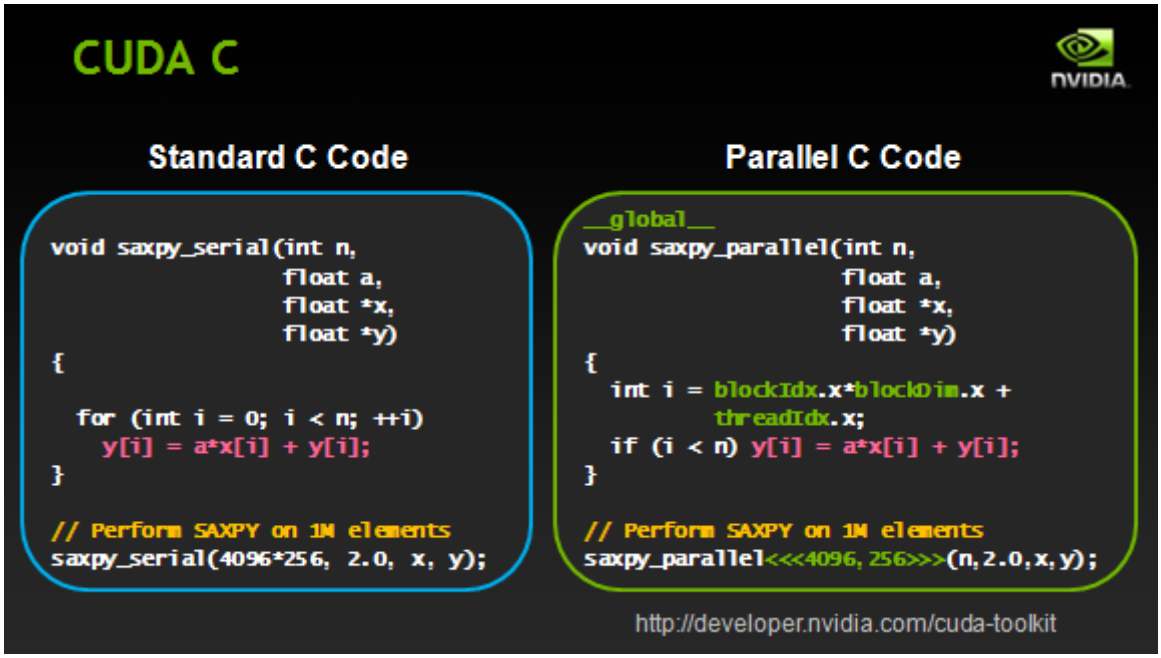
Ampere GA100 Streaming Multiprocessor

- 4 * 16 FP32 CUDA Cores/SM, 8192 FP32 CUDA Cores per full GPU
  Verwendung für Standard HPC Anwendungen
- 4 Tensor Cores
  Eingeführt für Matrixoperationen vom Typ $\boldsymbol{D} = \boldsymbol{A} * \boldsymbol{B} + \boldsymbol{C}$
  Verwendung für Neuronale Netzwerke bei z.B. Deep Learning

GPU Features	NVIDIA Tesla P100	NVIDIA Tesla V100	NVIDIA A100
GPU Codename	GP100	GV100	GA100
GPU Architecture	NVIDIA Pascal	NVIDIA Volta	NVIDIA Ampere
GPU Board Form Factor	SXM	SXM2	SXM4
SMs	56	80	108
TPCs	28	40	54
FP32 Cores / SM	64	64	64
FP32 Cores / GPU	3584	5120	6912
FP64 Cores / SM (excl. Tensor)	32	32	32
FP64 Cores / GPU (excl. Tensor)	1792	2560	3456
INT32 Cores / SM	NA	64	64
INT32 Cores / GPU	NA	5120	6912
Tensor Cores / SM	NA	8	4[2]
Tensor Cores / GPU	NA	640	432
GPU Boost Clock	1480 MHz	1530 MHz	1410 MHz
Peak FP16 Tensor TFLOPS with FP16 Accumulate[1]	NA	125	312/624[3]
Peak BF16 Tensor TFLOPS with FP32 Accumulate[1]	NA	NA	312/624[3]
Peak TF32 Tensor TFLOPS[1]	NA	NA	156/312[3]
Peak FP64 Tensor TFLOPS[1]	NA	NA	19.5
Peak INT8 Tensor TOPS[1]	NA	NA	624/1248[3]
Peak INT4 Tensor TOPS[1]	NA	NA	1248/2496[3]
Peak FP16 TFLOPS[1] (non-Tensor)	21.2	31.4	78
Peak BF16 TFLOPS[1] (non-Tensor)	NA	NA	39
Peak FP32 TFLOPS[1] (non-Tensor)	10.6	15.7	19.5
Peak FP64 TFLOPS[1] (non-Tensor)	5.3	7.8	9.7
Peak INT32 TOPS[1,4]	NA	15.7	19.5
…			
TDP	300 Watts	300 Watts	400 Watts
Transistors	15.3 billion	21.1 billion	54.2 billion
GPU Die Size	610 mm²	815 mm²	826 mm2
TSMC Manufacturing Process	16 nm FinFET+	12 nm FFN	7 nm N7

1. Peak rates are based on GPU Boost Clock.
2. Four Tensor Cores in an A100 SM have 2x the raw FMA computational power of eight Tensor Cores in a GV100 SM.
3. Effective TOPS / TFLOPS using the new Sparsity Feature
4. TOPS = IMAD-based integer math

Note: Because the A100 Tensor Core GPU is designed to be installed in high-performance servers and data center racks to power AI and HPC compute workloads, it does not include display connectors, NVIDIA RT Cores for ray tracing acceleration, or an NVENC encoder.

Programmierung mit erweiterter Standard-Programmiersprache
Beispiel C++: CUDA C

Spezielle Sprachelemente steuern die Parallelisierung
Braucht umfangreiches Wissen über die CUDA Architektur