

# Streifzug Geschichte

- mechanische zum ca. 600 v. Chr.: wahrscheinlich erstes Rechenhilfsmittel Abakus in China  
Hilfsmittel Rechnen bis ca. 18. Jh.
- mechanische zum seit 1623: Rechenmaschine für 6-stellige + / - mit automatischem Zehnerübertrag Tübinger Prof. Wilhelm Schickard  
Aparate Rechnen 1624: unabhängig 8-stellige Rech.+/-, auf + mit Komplement Franz Mathem. Blaise Pascal  
bis ca. 1760 1673: Rechenmaschine mit Staffelzügen 4 Grundrechenarten Gottfried Wilhelm von Leibniz  
Entwicklung Dualsystem
- 1933: Idee digitaler analytischer Rechenautomat mit Programmsteuerung Problem bei englische Mathem. Charles Babbage  
1886: elektromagnetische Sortier- und Zählmaschine Ausarbeitung Lochkartenfertigung für Realisierung (Ada Lovelace)  
Volksszählung 1890 USA amerik. Berkverkäufer Hermann Hollerith
- 21 mechanisch  
22 wegen verdeckten Schaltgliedern Aufbau mit Relais
- 1941: elektromagnetischer Dualcode-Rechner (Z3) mit Daten und Programm auf einem 8-Kanal-Lochstreifen  
D. Generation (erster funktionsfähige mit Dualsystem) Konrad Zuse  
(bereits Rechenwerk, Programmwerk, Speicherwerk)
- elektronische seit 1944: erster Rechenautomat, bei dem Programm erst in Speicher geladen Math.- John von Neumann  
Rechenanlagen (Speicherprogrammierbarer Rechner)
- 1945: erster Röhrenrechner (1700 Röhren, 150 kW) John P. Eckert, John W. Mauchly  
1. Generation ~3,3 min eine Karte z.B. Tabellenberechnung für US Army
- ### Rechnergenerationen
- Datenverarbeitungsanlagen seit ca. 1955  
Texte Bilder verarbeiten
- Informationsverarbeitungssysteme seit 1968  
Rechnerberat-ICL
- |              |                                    |                                 |
|--------------|------------------------------------|---------------------------------|
| 1. 1945-1956 | Vakuumröhren                       | 40.000 Operation/sec            |
| 2. 1955-1964 | Transistor                         | 200.000                         |
| 3. 1965-1971 | Small and medium scale integration | 1.000.000                       |
| 4. 1972-1977 | Large scale integration            | 10.000.000 ) Abgrenzung schnell |
| 5. 1978-???  | Very Large scale integrated        | 100.000.000                     |

- Ab 1954 Entwicklung Programmiersprache Fortran  
1955 Erster Transistorrechner  
1957 Entwickl. Magnetplattenrechner  
erste Betriebssysteme für Großrechner  
1968 Erster Taschenrechner  
1971 Erster Mikroprozessor (Intel 4004)  
1981 Erster IBM PC  
heute Smartphone, ...

Moore'sches Gesetz (1965) - Exp. Wachstum  
Gordon Moore, Mitgründer Intel

## Ethik - Teilbereich Philosophie - Bewertung menschlichen Handelns

z.B. Entwicklung für Anwendung Krieg  
Dual-Use Problematik (zivile, militärische Zwecke)

Kritik gabs nie so tiefgehend  
Joseph Weizenbaum „Das Interact ist ein riesiger Mistbefall“  
Anticaine Systeme, KI, Diktatiori im Raum Deepfakes

BRUNNEN

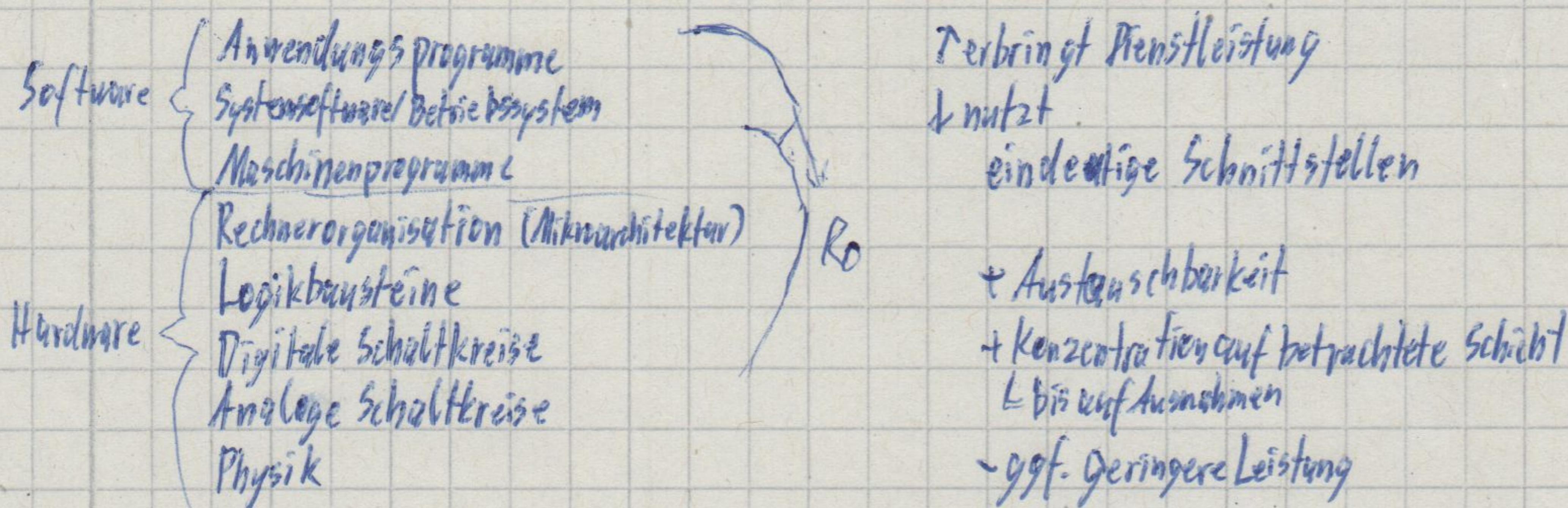
Gesellschaft für Informatik e.V. (GI) mit Leitlinien

moralischer Konflikt Gegenstand gemeinsamen Nachdenkens und Handelns

Ro

Abstraktion - Verstecken unnötiger Details

## (ein) Schichtenmodell



Datenverarbeitungssystem ~ Computer ~ Rechner ~ Rechnersystem Synonym (veraltet Rechenanlage, Rechenautomat, Informationsverarbeitungssystem)

eine Funktionseinheit zur Verarbeitung und Aufbewahrung von Daten. Verarbeitung umfasst die Durchführung mathematisch umformender, übertragender und speichernder Operationen

↳ Grundfunktionen: Verarbeiten von Daten (z.B. Rechnen, logische Verknüpfungen)

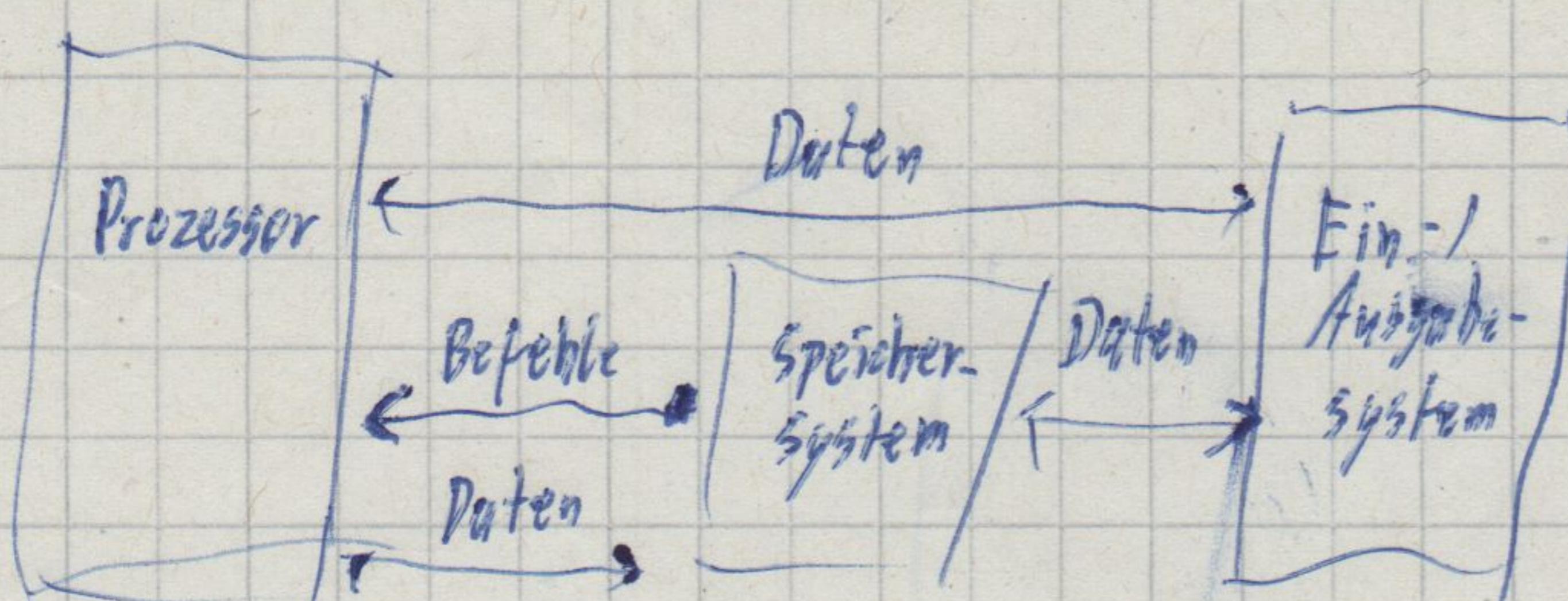
zurück fürbar Speichern " " (z.B. Ablegen, Wieder auffinden, Löschen)

Umformen " " (z.B. Sortieren, Packen, Entpacken)

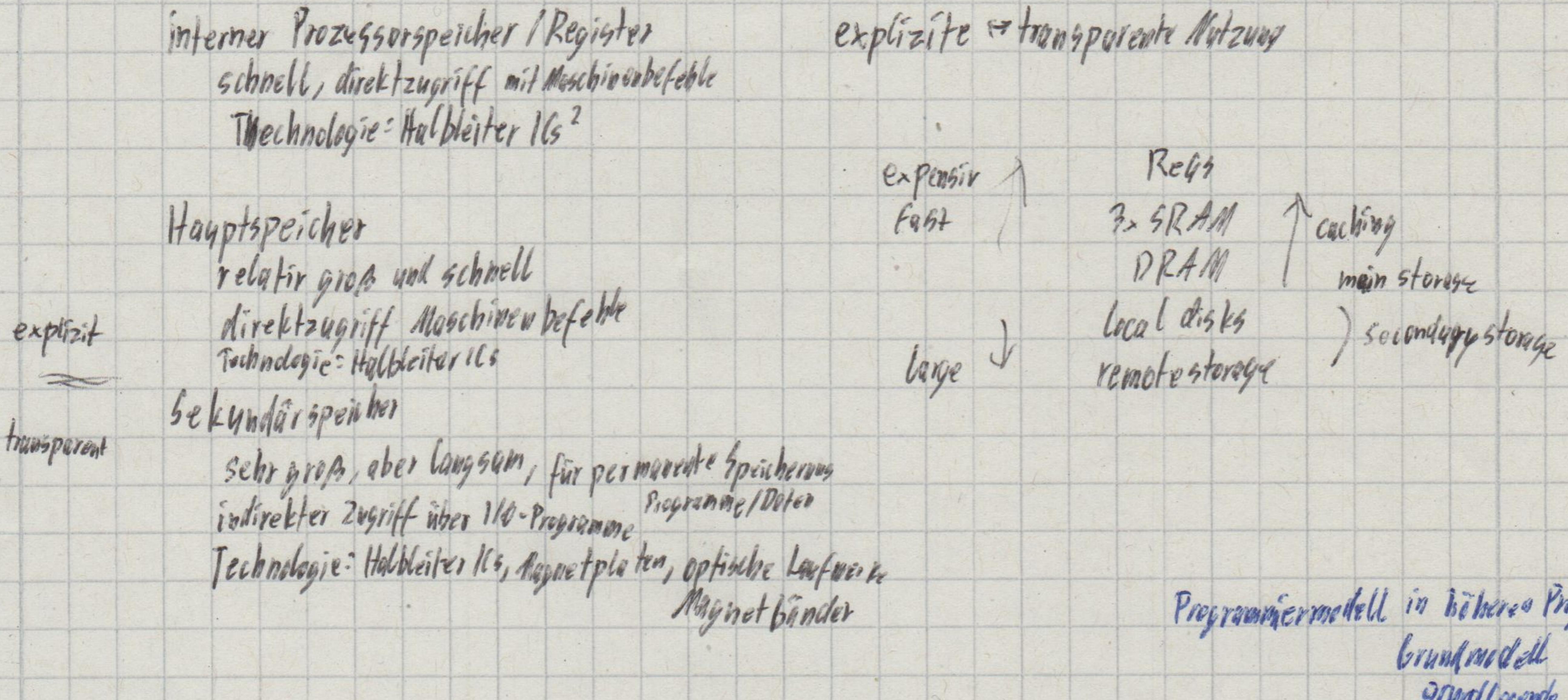
Kommunizieren " " (mit Benutzer, anderen Rechensystemen)

Abgrenzung z.B. Taschenrechner, Messgeräte: ladbares Programm, bzw. endliche Folge Maschinenbefehle

## Komponenten



# Speicherhierarchie



Programmiermodell in höherer Programmiersprache  
Grundmodell  
grundsätzliche Eigenschaften

Architektur / Programmiermodelle: definiert durch Maschinbefehle und Registeranzahl  
Programmierersicht: Statusregister zählen nicht

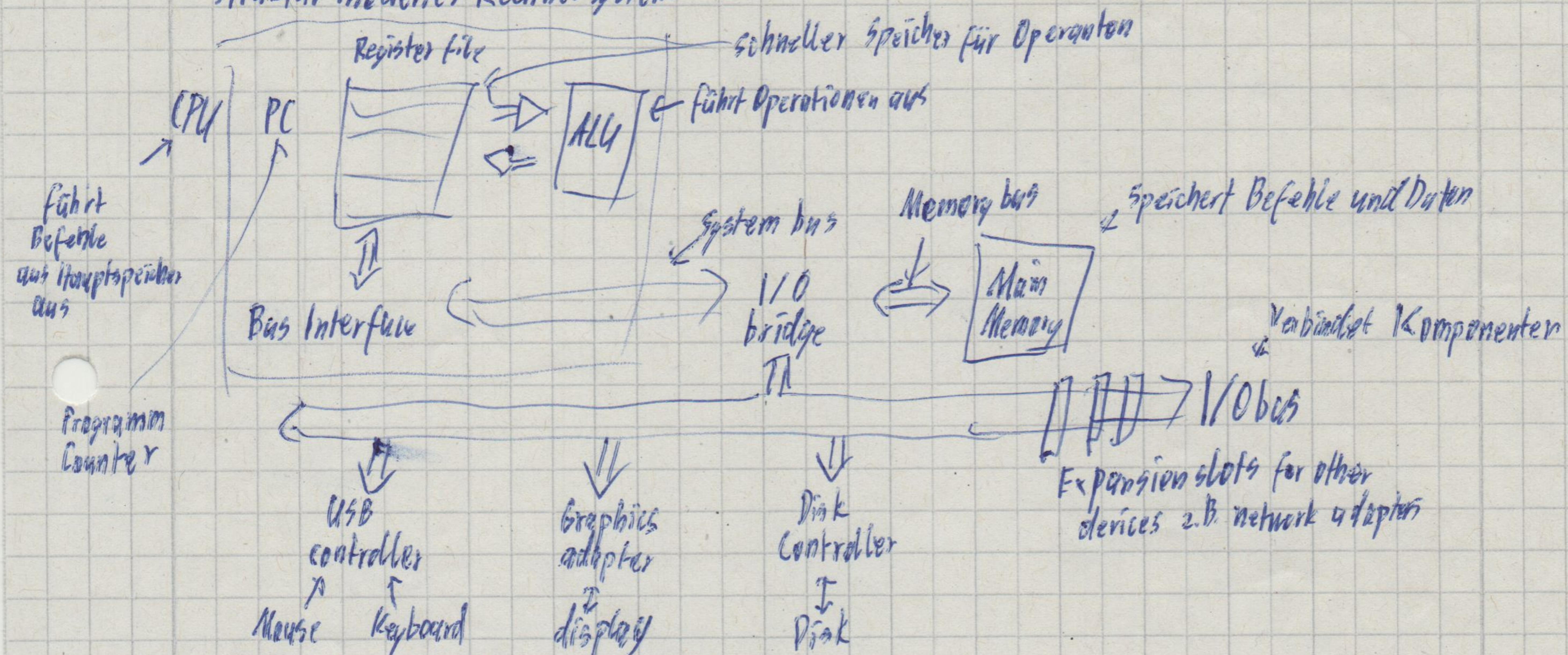
Mikroarchitektur: Hardwareimplementierung

Paradigma: Denkmuster / Beispiel z.B. Assembler primitives Paradigma

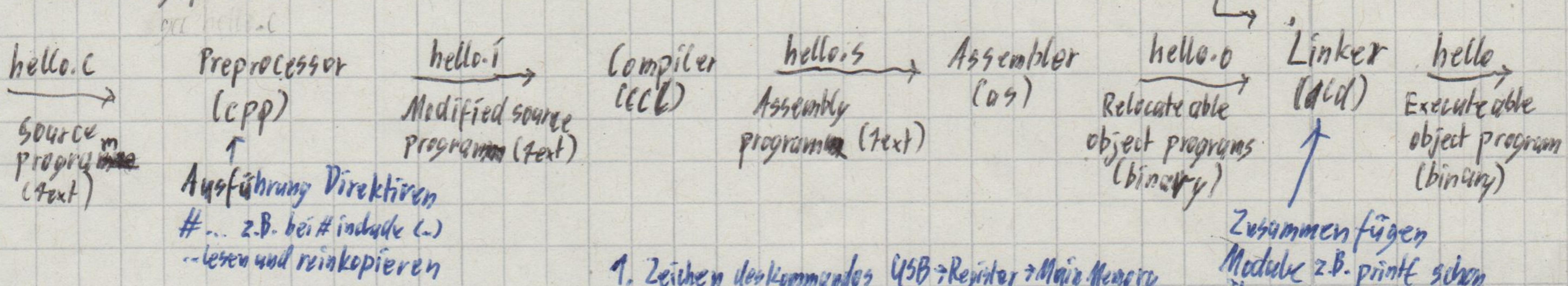
Vorlesung - ARM - Acorn Advanced RISC Machines 1985 (Smartphones, Laptops, Supercomputer) (heute)

Raspberry Pi 4 ARM Cortex-A72

## Struktur modernes Rechner system



## Übersetzungsphasen



Starten der Ausführung mit Shell (Kommandozeileninterpret)

- Zeichen des Kommandos USB → Register → Main Memory
- Befehle und Daten Disk → Main Memory mit DMA
- Ausführen Main Memory → CPU → Direct Memory Access

Zusammenfügen  
Module z.B. print-f0 schon  
übersetzt in Bibliothek  
u.a. Auflösung Referenzen

BRUNNEN

gcc hello.c übersetzt  
gcc -S hello.c generiert Assembler

# Befehle eines Rechnersystems

z.B. Implementierung Multiplikation

komplexe

komplexe Befehle, die Funktion vollständig ausführen

reduzierte Befehle, die Funktion als Befehlsfolge hintereinander ausführen

z.B. Multiplikation Register und Speicher

- CISC-Maschinen

- RISC-Maschinen

CISC

z.B. Intel Architektur

auch Speicheradressen als Operanden

RISC auch Load/Store-Architektur

z.B. ARM Architektur

→ Befehle weitgehend gleiche Ausführungszeit → effizienteres Pipelining

Einteilung z.B. auch nach Anzahl Operanden in Maschinencode

z.B.

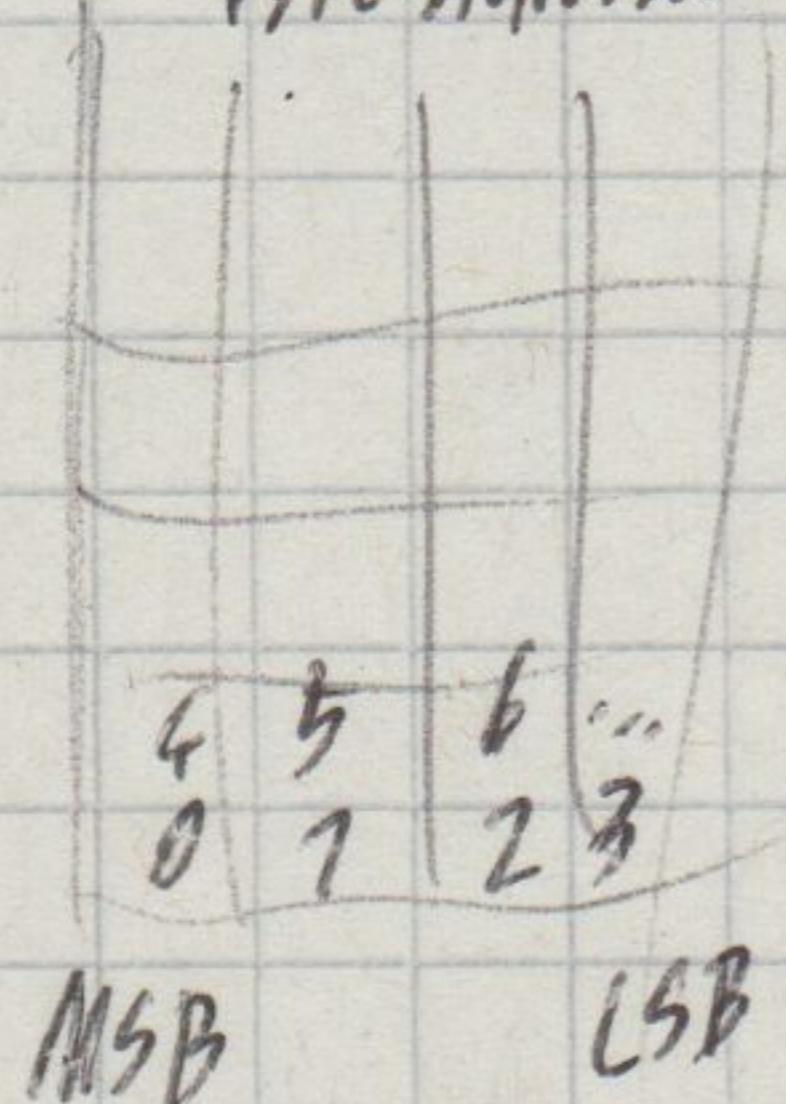
2-Adressmaschine (z.B. Intel Architektur)

3-Adressmaschine (z.B. ARM® Architektur)

Schemata für Byteummerierung in Wort

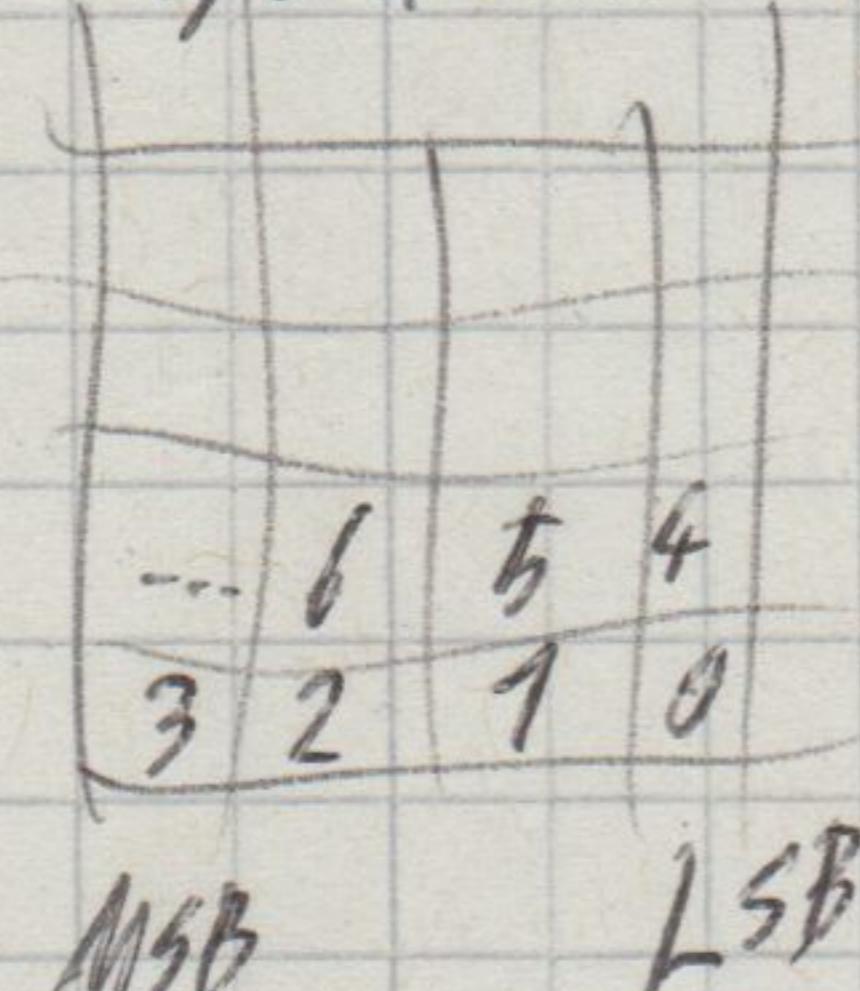
Big-Endian  
(Motorola-Format)

Byte-Adressen



Little-Endian  
(Intel-Format)

Byte-Adressen



ARM 32-bit Wort,

byte-adressiert

⇒ Werte Vielfaches von 4

ARM Registersatz

r0 Standard für Rückgaben

r1-r12

r13 - sp Stack Pointer

r14 - lr Link register (Rückkehradresse)

r15 - pc program counter

r0-r3 Parameter

r4-r11 sicher

r12 temporär

nur r3 freie Register ⇒ nur häufige Daten in Register, Rest Hauptspeicher

CPSR - Current Program Status Register enthält u.a. Statusflags

wichtigsten Statusflags

C - Carry flag

Z - Zero flag ⇒ (Z=0)

N - Negative flag ⇒ (Z<0)

V - Overflow flag

## Assembler Befehle

//ldr - load register from memory  
 ldr  
 ldrh - halfword  
 ldrb - byte

[rBase] ↓ offset  
 Adresse / [rBase, #...]  
 Adressarithmetik [rBase, rOffset]

Mehr Befehlsoptionen + Flexibilität

- Hardware sauberer, langsamer

//str - store register to memory rZiel, Adresse

str

strh

strb

mr rZiel, #16bit

operand

add r0, r1, r2

~~add r0, r1, #12bit Zweierkomplement~~

// Conditioncodes in ARM 64 Bit nur noch für Sprunganweisung  
 z.B. beq, bldeq, ... Befehlsuffixe

eq Z gesetzt

ne Z nicht gesetzt

CS/HS (gesetzt) ( $\geq$  für unsigned)

CC/LC (ungeSETzt) ( $<$  für unsigned)

MI N gesetzt negativ

PL N ungeSETzt positiv oder null

VS V gesetzt overflow

VC V ungeSETzt kein overflow

HI C gesetzt C=0  $\geq$  für unsigned

LS C gesetzt C=1  $c$ = für unsigned

GE N  $\leftrightarrow$  V / N = V  $\geq$  für signed

LT N + V  $<$  für signed

GT Z N = V  $\geq$  signed

LE Z N  $\neq$  V  $c$  = signed

Label-name: // Sprungmarke, Zielfür Sprünge

// Sprünge

// unbedingt!

b target

// Vergleichsbefehle

cmp r0, r1 // Subtraktion r0-r1 und setzt flags

tsr r0, operand // bitreines und setzt flags (N, Z, C)

status  
 // discards  
 // ergebnis

.data

var1: .word 5 // var1 im Speicher Wert 5 kann rolle 32bit sein

adr-var1: .word var1 // Adresse von var1

.asciz

skip

.global main // Definition Einstiegspunkt (muss nicht main)

main:

ldr r0, var1 // Lädt Wert

ldr r0, adr-var1 // Lädt Adresse

ldr r0, =var1

ldr r0, [r0] // Lädt Wert

## Sprachkonstrukte übersetzen

if ( $r0 == r1$ )  $\Rightarrow$  cmp r0, r1  
... bne L1  
...  
L1:  
...

if ( $r0 == r1$ )  $\Rightarrow$  cmp r0, r1  
// if body  
else // else body  
b L2  
NL1:  
// if  
L2:

while ( $r0 != 128$ )  $\Rightarrow$  WHILE:

// while cmp r0, #128  
bge DONE  
// while  
b WHILE  
DONE:

for ( $i=0; i < 10; i++$ )  $\Rightarrow$  mcr r0, #0  
// for FOR:  
cmp r0, #10  
bge DONE  
// for  
add r0, r0, #1  
b FOR  
DONE:

a[i]  $\Rightarrow$  // r0 Basisadresse r1 = i  
↑  
String von Wörtern lsl r2, r1, #2  
ldr r3, [r0, r2]

(TP)

Unterprogramme - Teilprogramm soll in Hauptprogramm ausgeführt werden

Makrotechnik - TP bekommt Makroname

TP wird bei jeder Erwähnung reinkopiert

+ effiziente Ausführung (keine Sprünge)

- Programm kann groß werden

- keine dynamischen Aufrufe

Unterprogrammtechnik - Teilprogramm (Unterprogramm, UP) nur einmal in Code durch Sprungmarke gekennzeichnet

Aufrufer - caller

übergibt Argumente (lokale Param.)  
springt

Aufgerufener - callee

führt Funktion aus  
gibt Ergebnis zurück  
! darf keine Speicherstellen des Callers überschreiben!

Fragen

lokale Variablen

lokaler/globaler Scope?

Für Aufruf Tiefe 1 manuelle Lösung  
allgemeiner Register auf Stack sichern

pl schreibt Rückkehradresse in r7c (lr)

Stack

LIFO

zum temporären Speichern  
dynamisch (dehnt sich aus, zieht sich zusammen)  
bei ARM nach unten  
r13 sp zeigt auf zuletzt abgelegtes Element

Pseudoinstruktionen für Stack

push & lr  
pop & lr

Unterprogramm sichert Register, die es verwendet will

z.B.	// Speichern	// Register
sub sp, sp, #12	ldr r4, [sp]	ldr r4, [sp]
Speicher reservieren	str r9, [sp, #8]	ldr r8, [sp, #4]
	str r8, [sp, #4]	ldr r9, [sp, #8]
	str r4, [sp]	str r4, [sp, #12]
		// Rückgabe
		speicher freigeben

Mehrfache Unterprogramme - Sicherung

z.B. smov r10, lr  
main: bl function  
mov lr, r10  
: lr

typisches Unterprogramm

r9 bis r11 und r14 auf Stack sichern  
Parameter in r4 bis r7/11 kopieren, weitere lok.Var. definition  
Implementierung Rechnung  
Rückgabe in r0

Wiederherstellen r4 bis r11 und r14 Speicher freigeben  
Rücksprung

- Speicher Nutzung  
- Programmgröße

Laufzeitanalyse bzw. Performance

Programm Profiling

gcc -pg -o hello hello.c  
./hello  
gprof hello gmon.out > analysis.txt

% time	cumulative seconds	self seconds	calls	self % calls	total % call	name
						func1
						func2
						main

↳ hilft z.B. bei Loop Unrolling, Parallelisierung, Threads

## (Mnemonics)

Assembler - Assemblerbefehle  $\rightarrow$  Maschinencode (Objektdateien) und symbolische Namen  $\rightarrow$  Adressen  
↳ Assembler  
↳ Crossassembler: läuft auf X, generiert für Y  
↳ Disassembler: MaschinenSprache  $\rightarrow$  Assembler (üblicherweise Verlust symb. Namen, Kommentare)  
↳ objdump -S hello.o Analyse Werkzeuge: gcc Toolchain; IDA; Ghidra; Radare2

Funktionsweise:

1. Schritt: 1-ter Lauf (Phase): Finden Marken, Zuordnen Maschinenadressen (Befehle dazu zählen)  
2-ter Lauf (Phase): Übersetzen in äquivalente Operatoren, Registerbezeichner, ...

2. Schritt: Erzeugen .o- Datei

- ↳ 1. Fall: absolute Adressen, 1 Objektdatei
  - + direkt lediglich - Speicherort vorher bekannt, nicht verschiebbar
- ↳ 2. Fall: relative Adressen (ggf. mehrere Segmente als Input, mehrere Objektdateien
  - + zu einzelnen Objektdateien
- Beispiele Linkers (Binders) und Loaders (Loaders) nötig

## Objekt-Programme

standard -  
↳ Executable - direkt in Speicher kopier und ausführbar  
↳ Shared: Spezialfall relocatable in Speicher laden, dynamisch mit anderen ausführbar  
↳ Relocatable: kann mit anderen zu executable zusammengefügt werden

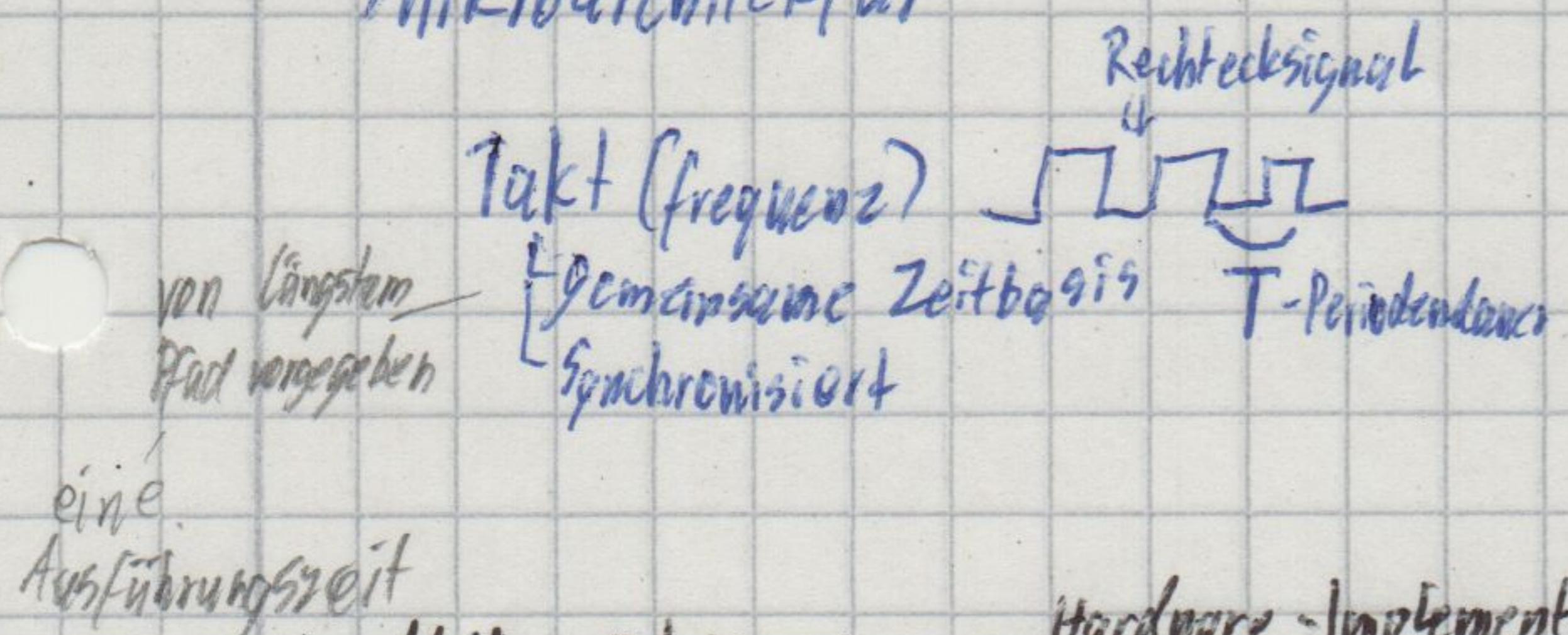
readelf -a hello.o  
↳ ELF 16 Byte  
↳ header: Wort-Größe, Byte-Ordering, ... (z.B. ARM, IA 32) sh, revision, type start program section headers  
↳ text: Maschinenbefehle  
↳ rodata: muss nur gelesen werden read only  
↳ data: init global variables  
↳ bss: nicht init global vars  
↳ symtab: Symboltabelle mit Infos  
↳ rel.data: Relocation Infos für globale vars  
↳ debug: Debugging Symboltabelle (nur bei C-Compiler mit -g langsam)  
↳ line: Zuordnung Maschinencode  $\rightarrow$  C-Anweisung  
↳ strtab: Zeichentabelle für Symboltabelle und Debugging Symboltabelle

Linker/Binder: aus relocatable object files  $\rightarrow$  ein executable object file - von Betriebssystem vorgegeben

Loader/Lader: Systemprogramm: Programmmodul wird mit Startadresse in Hauptspeicher geladen

- ↳ absolutes Laden
- ↳ relatives
- ↳ dynamisches Laden zu Laufzeit

# Mikroarchitektur



Eintaktimplementierung: 1:1 → braucht Harvard-Architektur

Mehrtaktimplementierung: 1 Befehl in Teilschritte

Pipelined-Implementierung: 1:n und parallele Teilschritt ausführungs

## Mikroarchitektur ← Hardware-Implementierung → Architektur

Datenpfad: verbindet funktionale Blöcke  
Kontrollpfad: Steuersignale, Steuernetz

IPC - instructions per cycle

ILP - Pipelining für Instruktionslevel Parallelismus

Sprung Vorhersage

out-of-order execution (dynamic instruction scheduling)  
multi-issue systems (mehrere Operationen pro Zyklus)

ISA - instruction set architecture (Menge Befehle)

RISC -

CISC -

SIMD - single instruction multiple data (vector processing)

VLIW - very long instruction word (static multi-issue)

superscalar processor (dynamic multi-issue)

## Zustand der Architektur

PC

16 Register

Status Flags

Speicher

Speicher  
synchron schreiben  
asynchron lesen

Kon-Neumann-Architektur  
gemeinsamer Speicher Daten/Befehle

Harvard-Architektur  
getrennt

## 1. Befehlsholphase

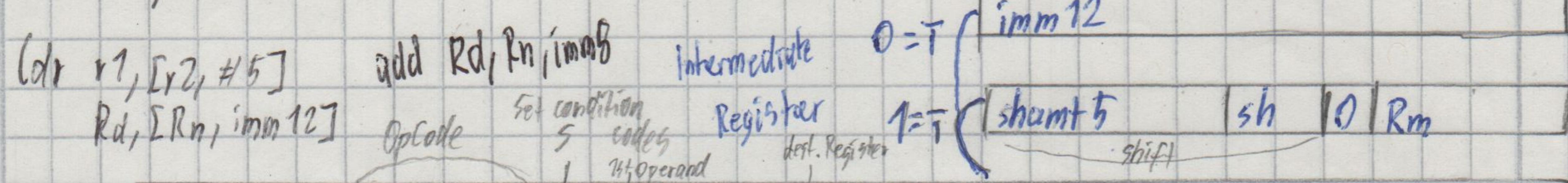
## 2. Befehl dekodieren

Lesen Quelloperanden  
(Erweiterung Intermediates)

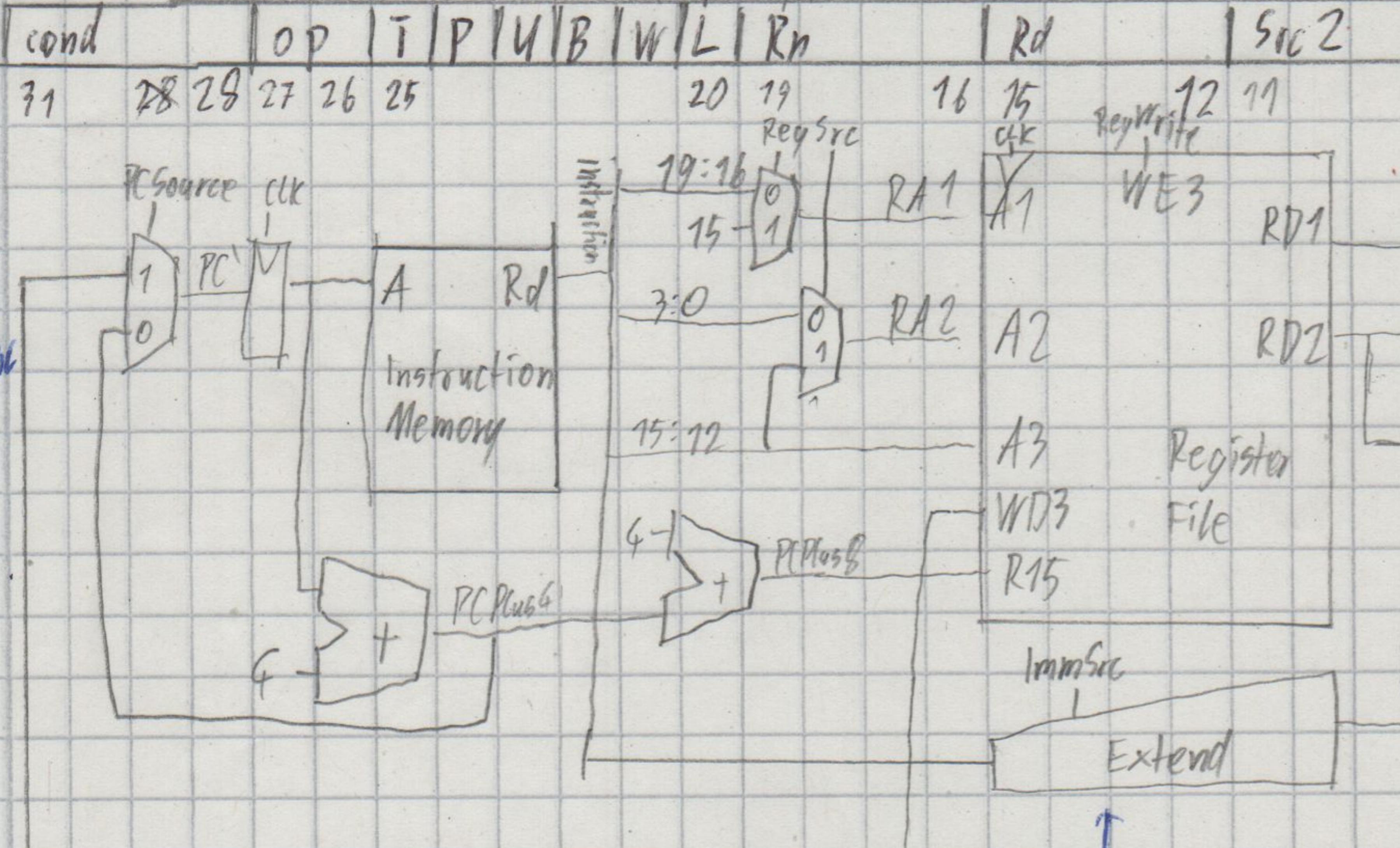
## 3. Befehlausführung

Wert/Speicher Berechnung  
(Laden/Speichern Speicher/Register)

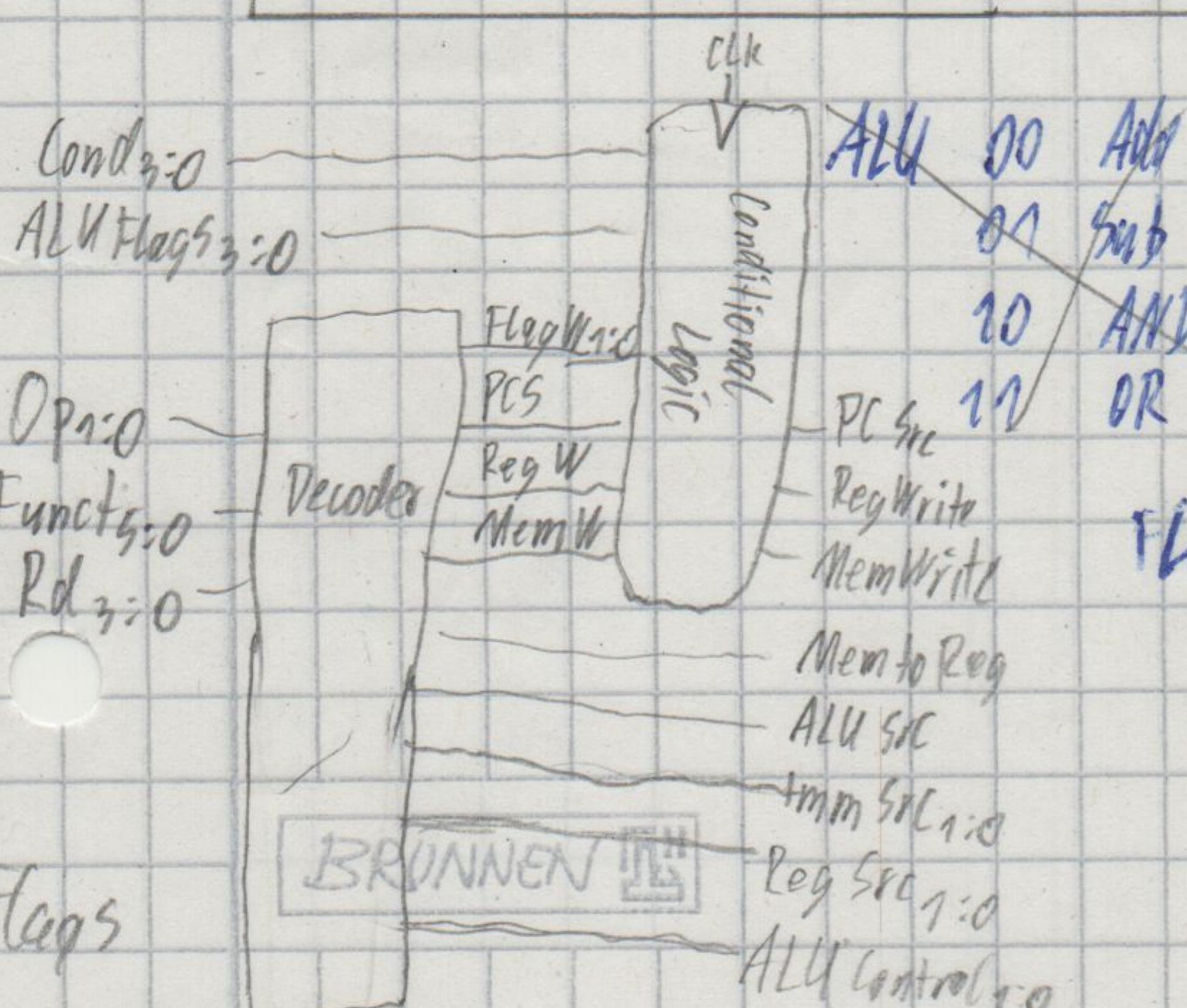
Berechnung Adresse nächster Befehl



Eintakt  
prozessor  
einfach  
-Taktsequenz  
durch  
-3 Adressen 1 ALU  
2 Speicher



00 24b0, Inst 2:0  
01 20b0, Inst 11:0  
10 6fInst 23:3, Inst 23:0 003 sign-extended  
multiplexed by 4



Flag W1:0 ⇒ Flag W1=1 NZ saved

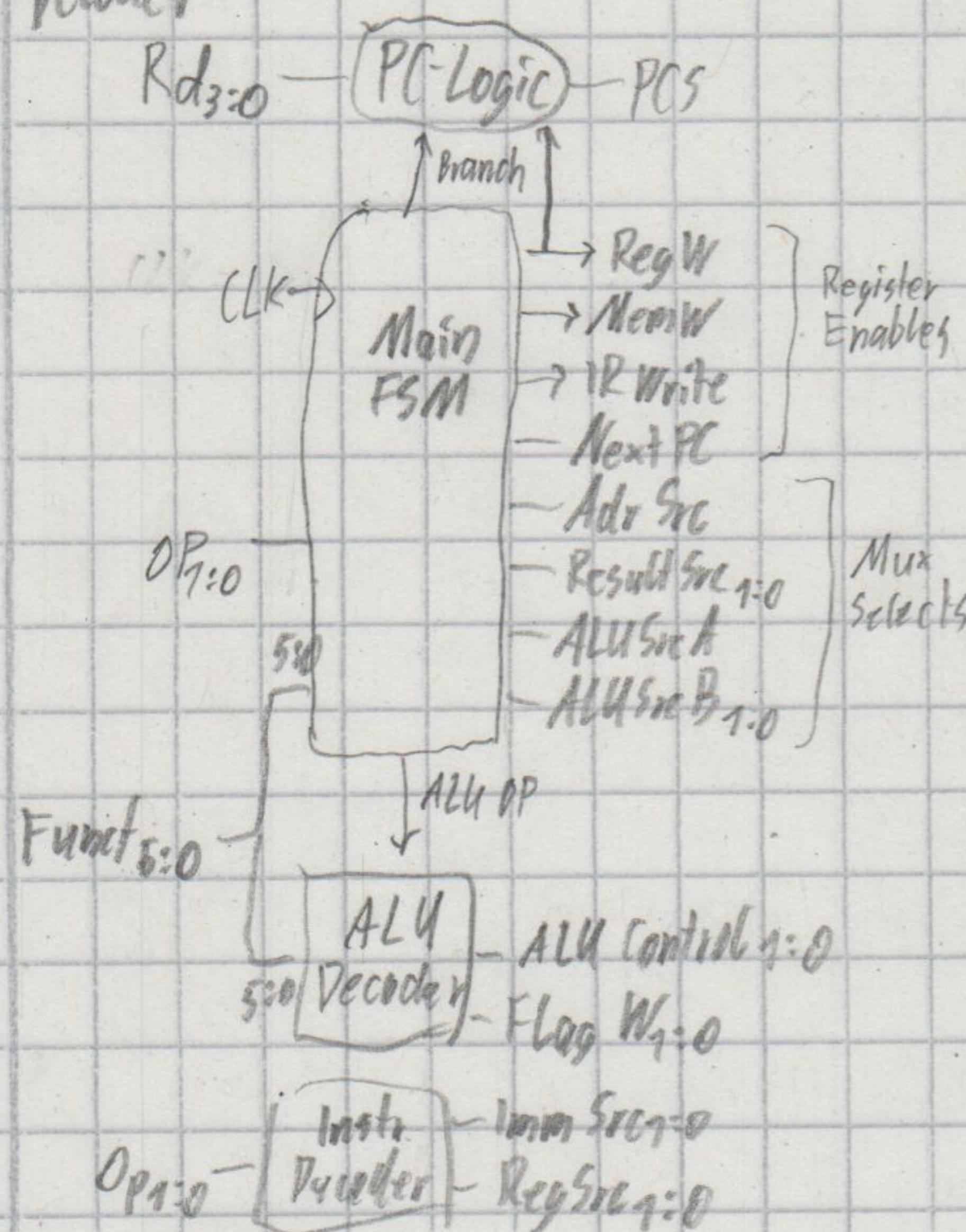
Flag W0 = 1 CV saved

Jeder Befehl wird in Teilschritte zerlegt  
Von-Neumann-Architektur

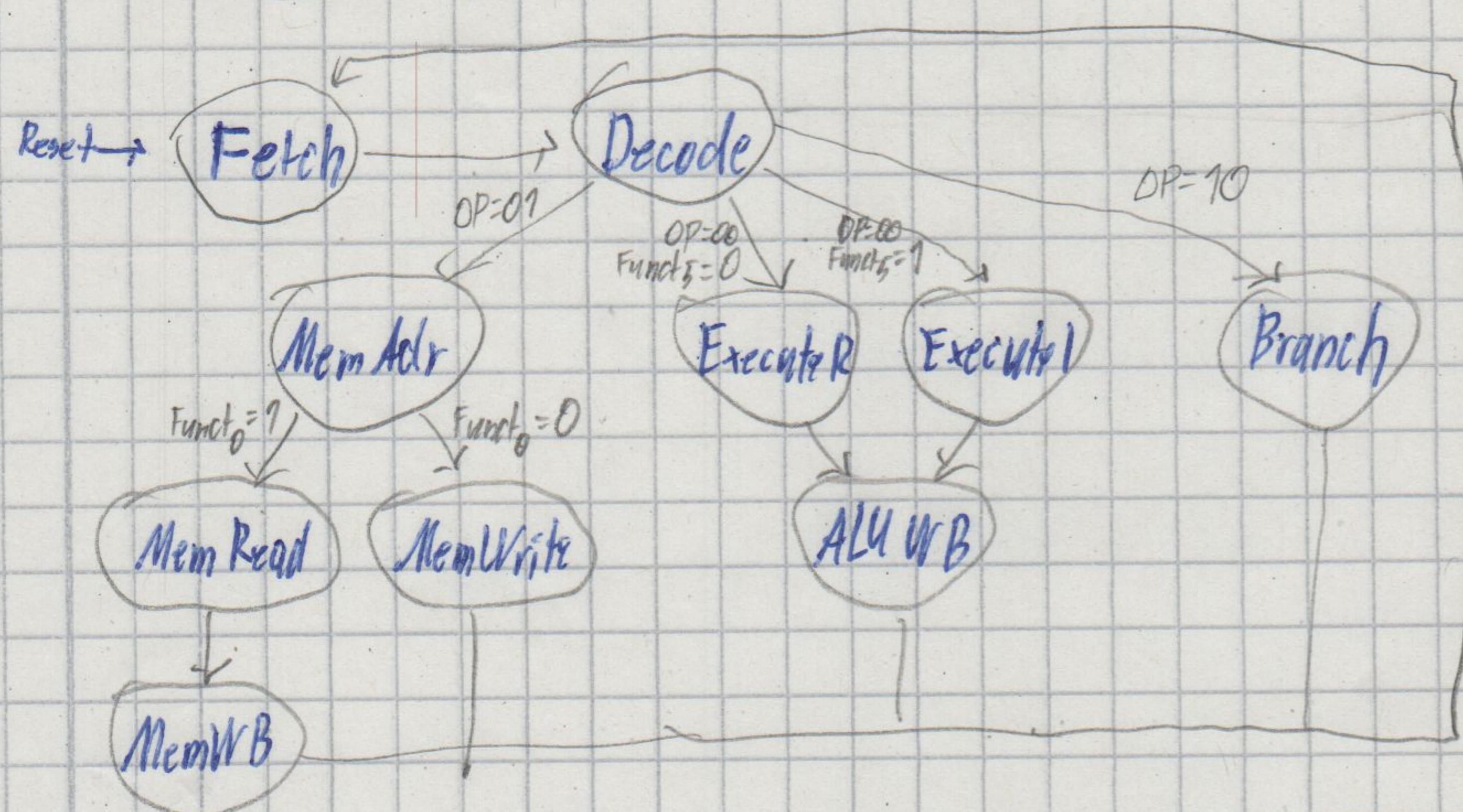
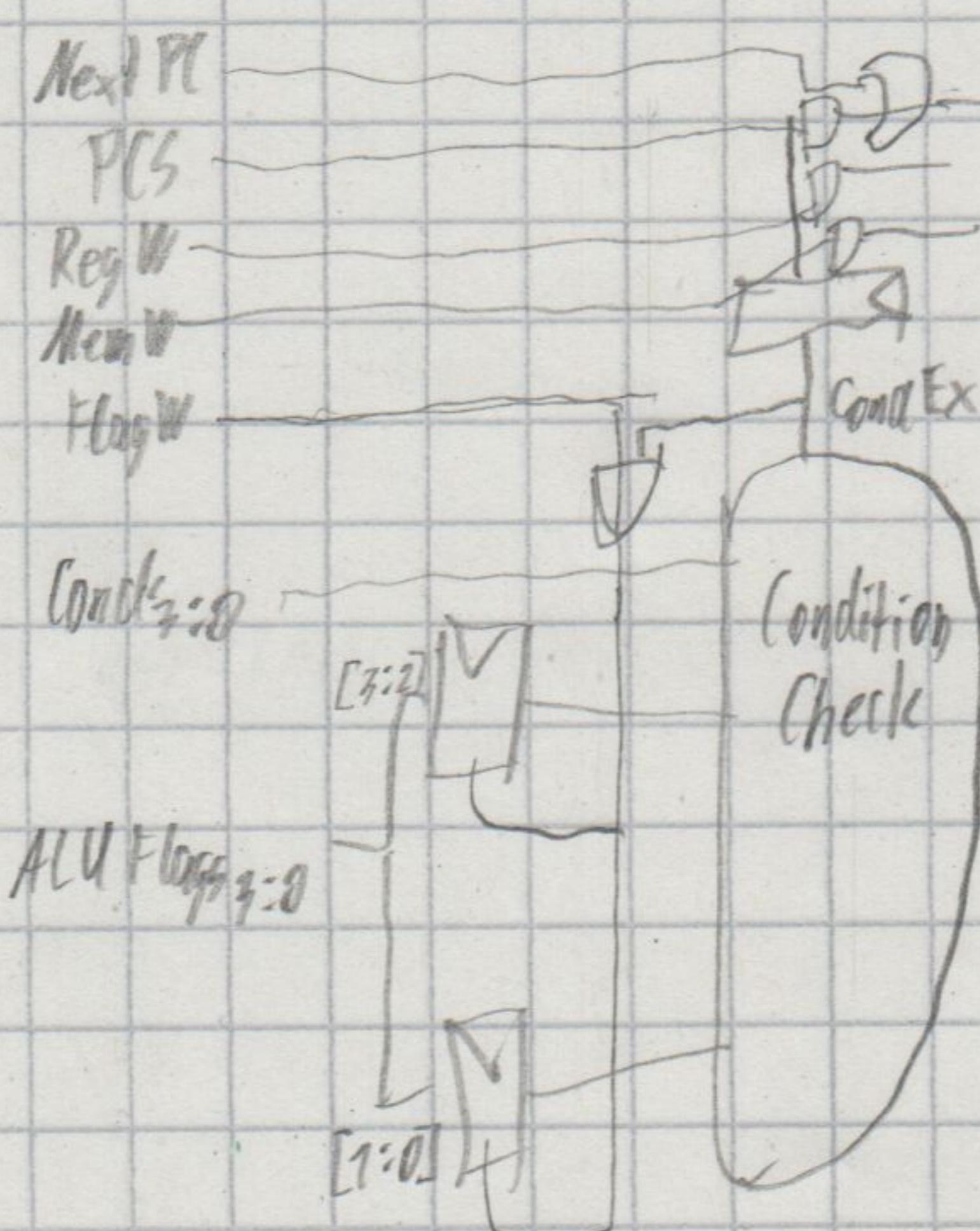
### Mehrtaaktprozessor

- + höhere Taktsequenz
- + einfache Instruktionen schneller
- + bessere Wiederverwendung Hardware
- aufwendigere Ablaufsteuerung

### Decoder



### conditional logic



△ See Ausdruck Prozessor

## Pipeline-Prozessor



### Phasen Befehlausführung

- Instruction Fetch
- Instruction Decode, Read Register
- Execute ALU
- Memory Read/Write
- Write Register

nicht architektonische Register zwischen Stufen (auch mit Steuersignalen)  
WABD muss auch durch Register geführt werden  
PC kann direkt ins Register File (sieben +8 da eins vergrbt)

## Hazards Abhängigkeiten zwischen Pipelinestufen

Instruction hängt von noch nicht verhandeltem Ergebnis ab.

### Hazard-Unit

Data Hazard: Neuer Wert noch nicht in Registerfeld

z.B. RAW - Read-after-Write

#### Lösungen

Compilezeit

scheduling (nop einplanen (no operations))

wie viel Zeit wird gebraucht

in-order

reordering (jedoch Cache z.B.)

out-of-order + schneller

- Siliziumfläche

Laufzeit Forwarding

geht nicht immer (bypassing Register durch abkürzung)

Stalling - Prozessor anhalten bis Daten der Befehlphase wiederholen

Control Hazard: unklar welche nächste Instruktion bei Verzweigungen

#### Lösung:

Flush wrongly started  
flush input of Registers

Analyse Rechenleistung dient Leistungsbewertung, abhängig von Hardware (Prozessor, Speicher) Befriebssystem

z.B. Taktfrequenz

Anzahl Prozessoren  
Größe, Art des Speichers

Käufer, Designer

Preis  $\leftrightarrow$  Leistung  
Verhältnis

Energiebedarf

Benutzer: Antwortzeit  
Rechenzentrum: Durchsatz

Welche Aufgabe  
repräsentativ?

Antwortzeit - reale Zeit Systemperformance

Ausführungszeit - CPU Zeit  $\times$  threads Prozessorvergleich

$\downarrow$  System CPU time  
User

Max:  
time-  
Befehl

CPI - cycles per instruction

IPC =  $\frac{1}{CPI}$  instructions per cycle

MIPS - Million Instructions Per Second

früher Erhöhung Taktfrequenz

$\Rightarrow P = V^2 \cdot f \cdot C_L$  Leistungsumsatz

$\Rightarrow$  Wärme schwer abtransportieren

$\Rightarrow$  Parallelrechner

Ausführungszeit = Max:  $\cdot CPI \cdot \frac{1}{T_k}$  Sekunden pro tick

MFLOPS - Million Floating Point Operations per second

## Klassifikation von Flynn

Instruction- Data- Stream	SI	MI zu einem Zeitpunkt
SD	SISD	SIMD
single Data	Von-Neumann Rechner	$\downarrow$
MD	SIMD	MIMD
Multiple Data	Feld-, Vektor-rechner	Multiprozessorsysteme

+ etabliert

- sehr grob

$\downarrow$  Pipelining, Wortbreite, ~~Cache~~

Verbindungsstrukturen, Speicherorganisationen

$\Rightarrow$  SIMD Befehle, SSE Einheiten

$\downarrow$  SIMD Extension

Streaming

c	Mantisse	Normalisiert	0 < e < max	beliebig
0	0	Nicht normalisiert	0	$\neq 0$
0	0	Null	0	0
max	max	Unendlich	max	0
$\neq 0$	0	keine Zahl	max	$\neq 0$

Zahlendarstellung

real	f Festkomma (Kommasstelle definiert)	f Gleitkomma / halb logarithmisch
$\downarrow$ Kommasstelle Bestimmt die Zahl	ANSI / IEEE 754	$\downarrow$ Kommasstelle Bestimmt die Zahl
single	31 bit	fraction
double precision	11 bit	bias
extended	15 bit	127
	64 bit Mantisse	1023
	0	16383

s Charakteristik c

aber häufig Compiler,

Benchmarks - repräsentative Programme Instruktionssätze auf gängig optimiert

typen

Real Programme z.B. C-Compiler, LATEX, SPICE

Kernels - kurze isoliert zur Ausführung gebundene kritische Programmabschnitte

Toy Benchmarks - kleine einfache Programme z.B. Quicksort

Synthetische - speziell entwickelt soll einzelne Komponenten untersuchen  
in Bezug auf Instructions

1989 SPEC - Standard Performance Evaluation Corporation  
mehrerer Hersteller zunächst reale Programme

Prax. Anpassung aktueller Eigenschaften

Kostenpflichtig auch für GPU, Mailserver, Java

frei gzip, bzip, Linpack (für Supercomputer)

Linux: cat /proc/cpuinfo

Taktfrequenz

Cache-Größe

Floating Point Unit

kein Leistungsmaßnahmen  $\rightarrow$  BogoMips  $\leftarrow$  bei boot ermittelt

BRUNNEN zum Kalibrieren interne Würtschleife

# Betriebssysteme & Ausnahmebehandlung

in Anfangen 1940: Programmierung durch Entwickler

Hardware machte mehr Probleme war viel teurer  
wurde stärker teurer, Benutzer schrieben eigene

⇒ Standardprogramme Steuerung Rechnerkomponenten (minitry, supervisory, executive, system)  
Auftragsarbeiten operating

Betriebssystem: Grundlage Betriebsart Ablösung von  
Steuerung, Überwachung Programme

## Aufgaben

Veredelung = Hardware

I/O-Schnittstellen

Benutzer - "

Unterbrechungsbehandlung

Organisation, Steuerung, Protokollierung des Ablaufs

Langfristige Datenhaltung

Einhalten Qualität (Leistung, Verfügbarkeit, Sicherheit)

Anforderungen: viel Parallel und verfügbar (Optimal ausgenutzt)  
↳ zunehmend komplexer

## Betriebs

Prozessenzustände - Prozessor - Wechsel per Software

Maschinenzustand (stop, laden, tätig)

Betriebszustand

Privilegierungszustand - Menge erlaubter Befehle

oft Innenzustand

Systemzustand - privilegiert privilegiert  
voller Befehlszustand

Unterbrechungen auf Wunsch NACA 1956 mit UNIVAC, IBM 1958

Peripheriegeräte können nach Beendigung Auftrag Prozessor Unterbrechen

⇒ parallele Arbeit möglich

Signal, dass Befehlszyklus abbricht und an spezieller Stelle fortführt

Ereignisursache in Software oder Hardware

Programmbezogen

durch Befehl, trifft Verursacher z.B. Exzeptions

⇒ synchron / intern

arithmetisch, Adressfehler, Rechte

Systembezogen

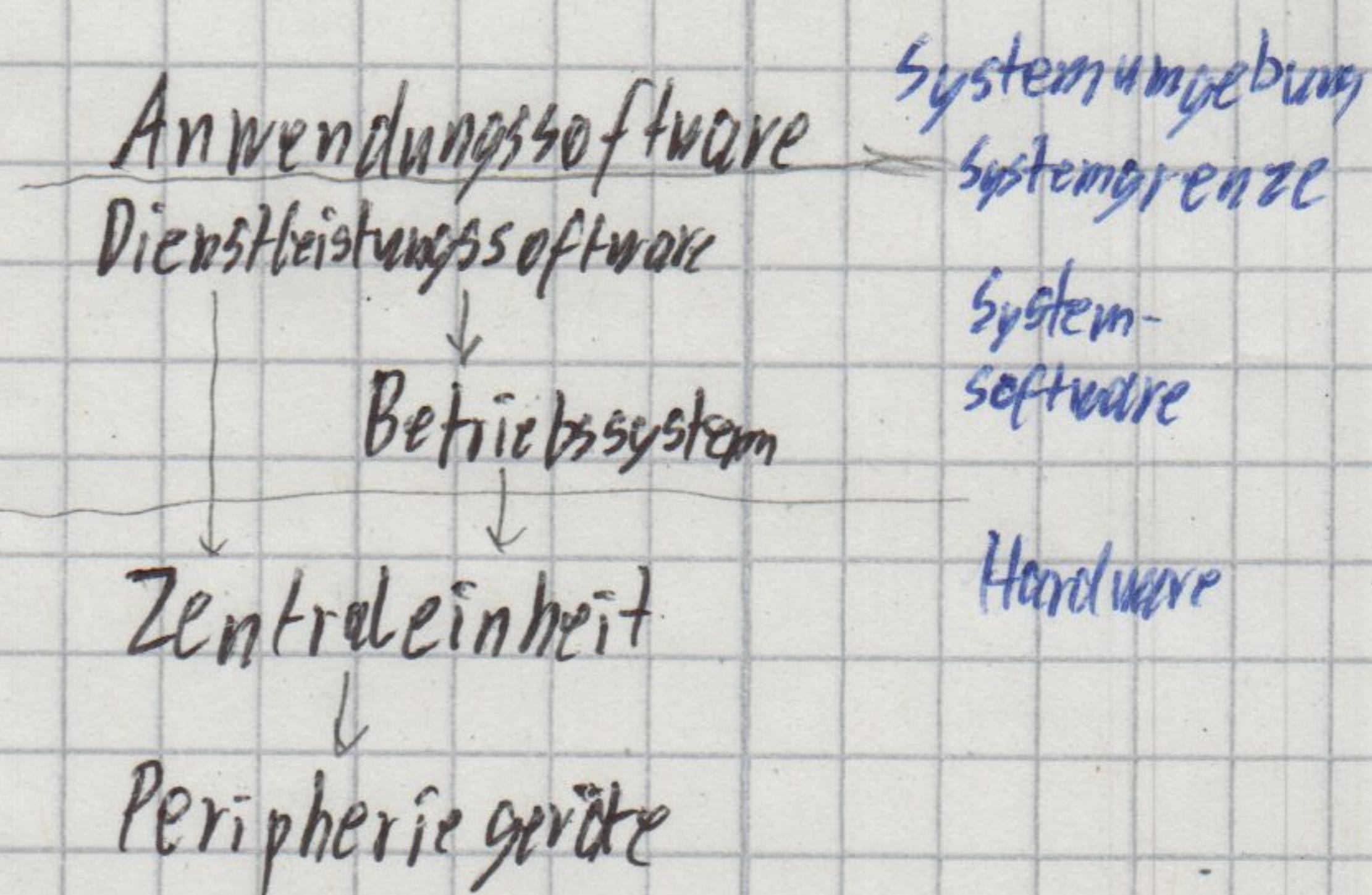
Spezialbefehl zum Einleiten Systemaufrufe

durch Gerät, trifft zufällig laufendes Programm z.B. Zeit, I/O-Unterbrechung

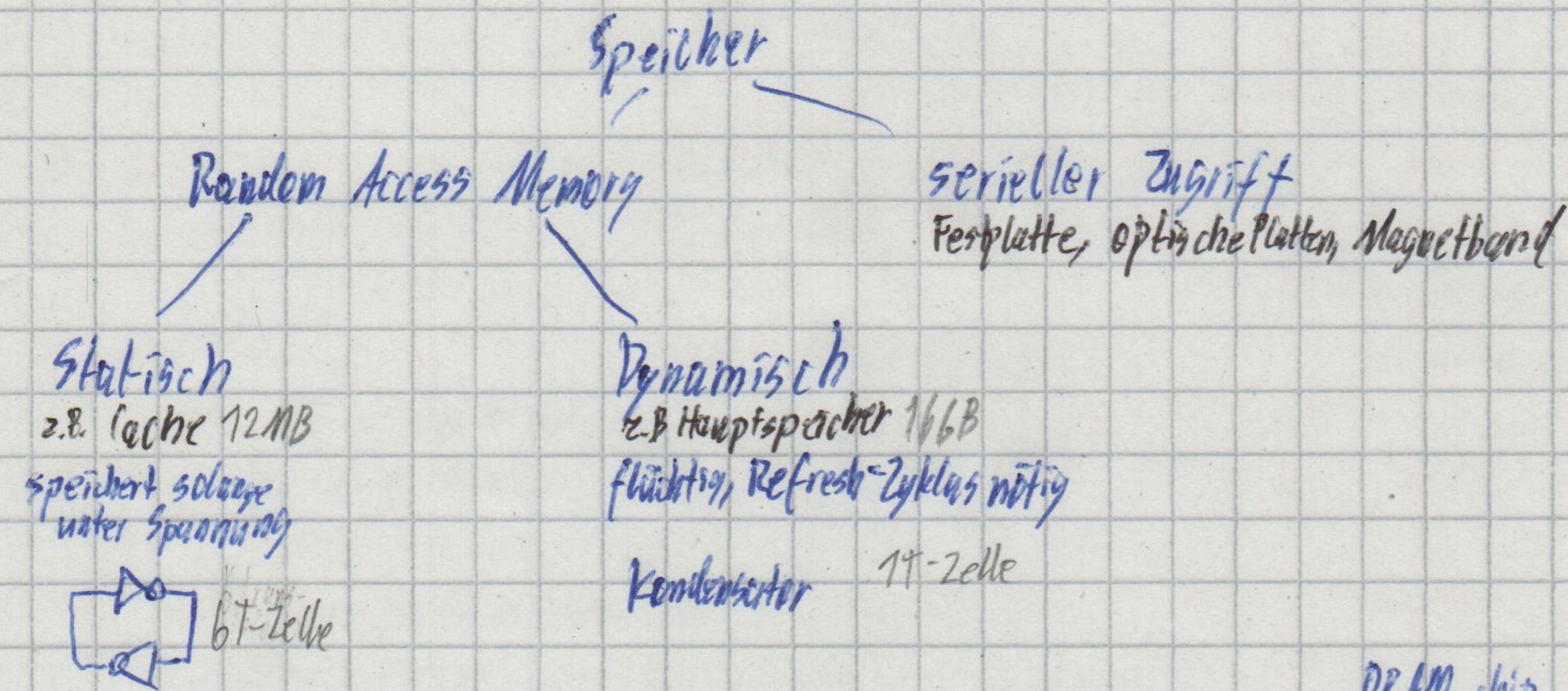
⇒ asynchron / extern

Prozessoranrufe

Maschinenfehler



auch BUSES protokoll relevant  
 Speicherhierarchie Bandbreite - Bgfssec  
 $\frac{1}{8} \text{bit}$  zykluszeit - Min zwischen Zugriffen  
 $\frac{1}{16} \text{bit}$  Kosten und Zugriffszeit = durchschnittliche Zeit  
 Geschwindigkeit, Kapazität ein Wort aus Speicher  
 Zugriffsvorfahren  
 Anforderbarkeit, Permanenz

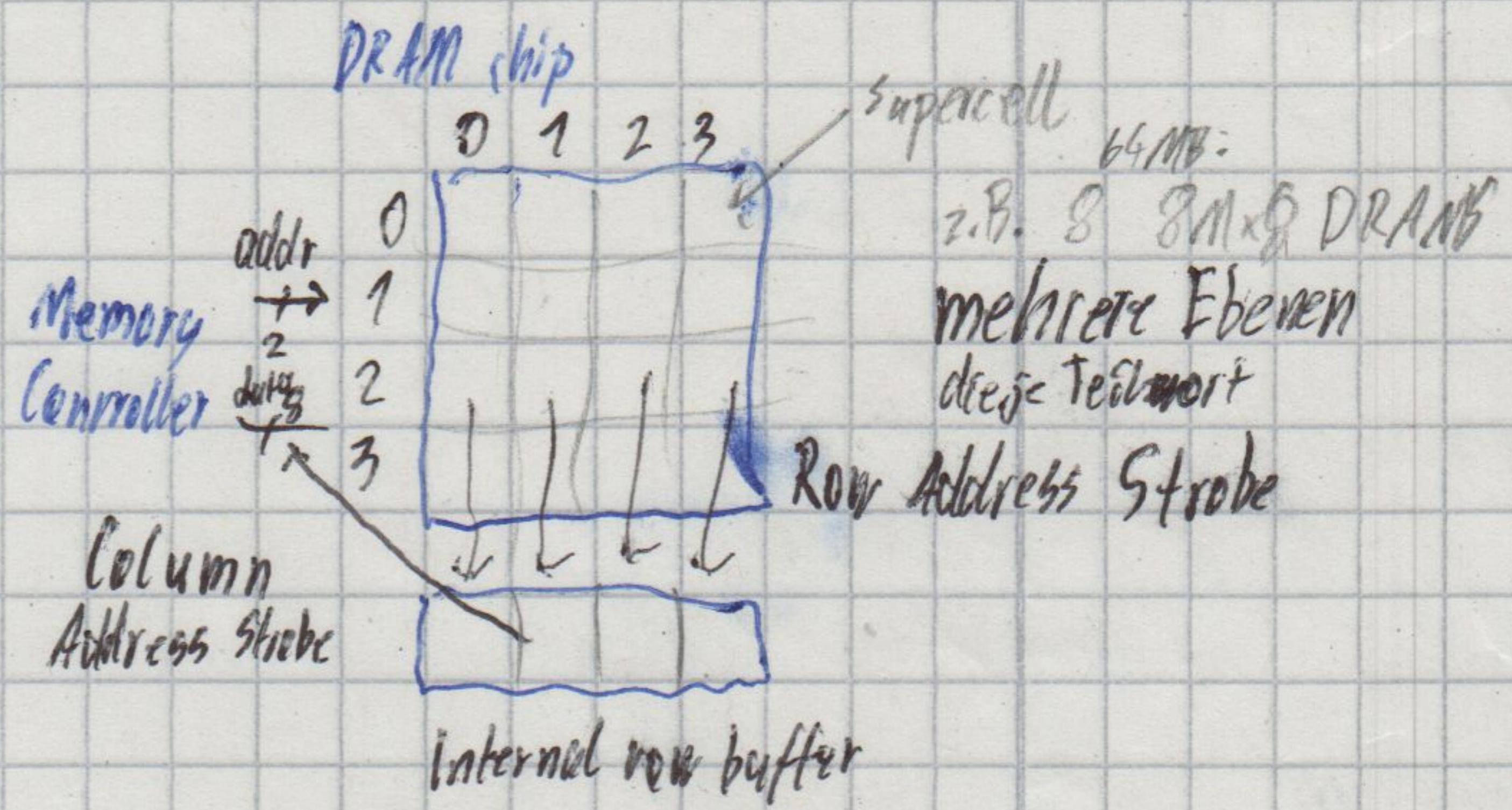


Speicherorganisation  
 Anzahl adressierbarer Plätze  $\times$  Breite (in Bit)  
 $\log_2(\text{Adressen})$  Adresseneingänge      Daten ein/ausgänge  
 chip select, output, write enable

der Daten  
 der Befehle → Zuverlässigkeit auf Basis vorheriger Zugriffe  
 Lokalitätsprinzip - meist nur Zugriff auf geringen Teil  
 zeitliche Lokalität - z.B. Schleifen      Adressraum  
 Raumliche Lokalität - z.B. Matrizen

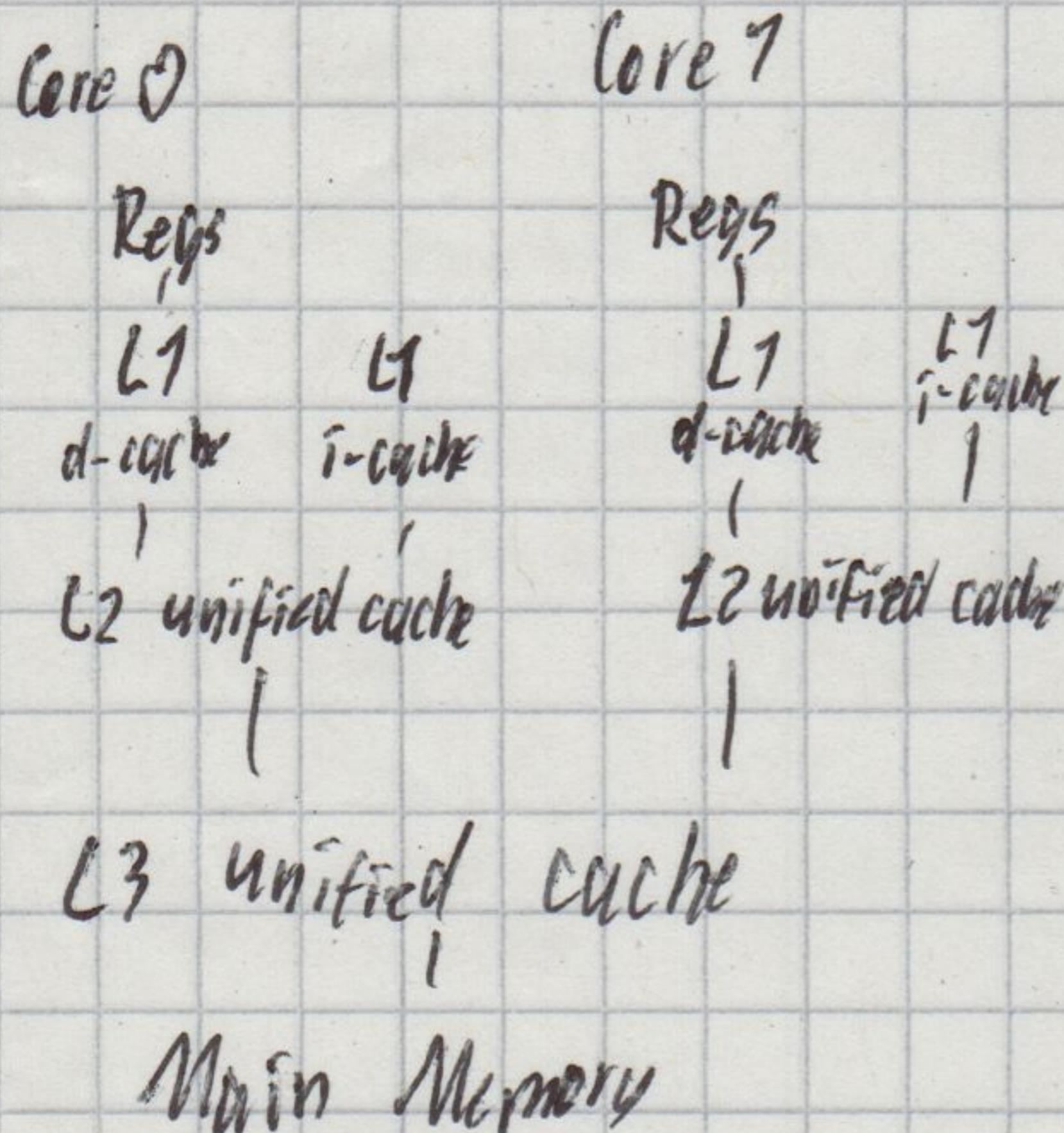
Performance relevant  $\Rightarrow$  bei mehrdim. Array erst rechteste iterieren

Cachen Level k Cache für k+1  
 copied in block-sized transfer units



z.B.	CPU Register	4/8 Byte	0
	L1 Cache	64Byte Block	1
	L2 Cache	64Byte Block	10
	L3 Cache	"	70
	Platten Cache	Disk Sektör	100 000

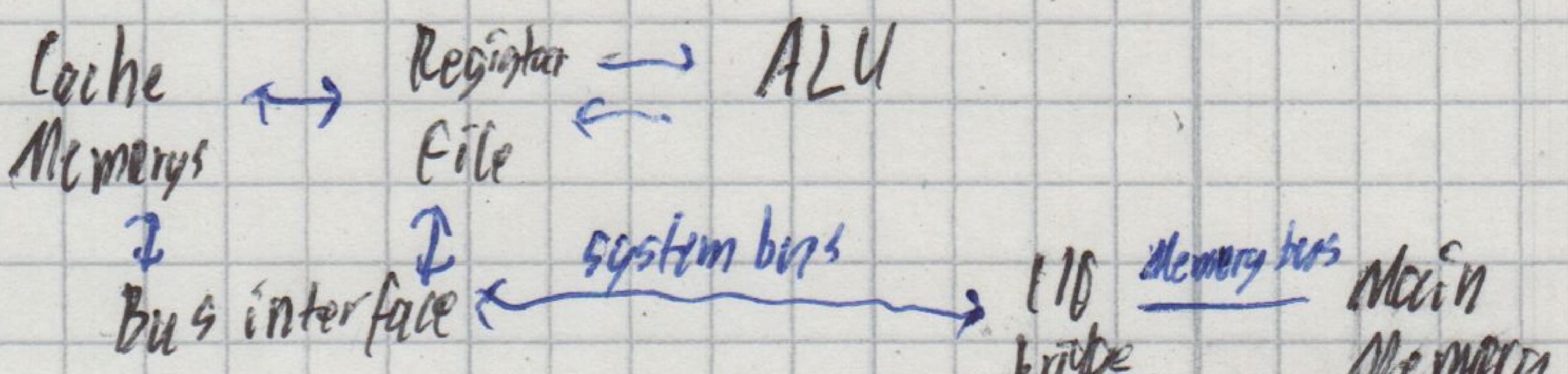
ARM



Anfrage Block d  
 Suche in Cache

Cache Hit      Cache Miss  
 hole d auf Ebene k  
 ent. Ersetzung

↑ Zufallsersetzung  
 ↓ Least-recently used (LRU) Ersetzung



# Glossar

Schichtmodell Computer

Datenverarbeitungssystem - Funktionseinheit: Verarbeitung, Aufbereitung

Komponenten

Speicherhierarchie

Explizit zugreifbar Programmierbar

Implizit für Maschinenprogramm transparent verwendbar

Architektur/Programmiermodell - (Maschinenbefehle, Registersatz, Statusregister)

Mikroarchitektur - Hardwareimplementierung

Paradigma - Denkmuster, Beispiel

CISC

RISC + ähnliche Ausführungszeit → Pipelining

Big/little höchstwertiges an niedrigster/höchster Stelle

Makrotechnik Makroname - effiziente Ausführung

Unterprogrammtechnik Sprungname, Aufruf

Stack - LIFO, dynamisch, sp

common subexpression elimination

