

Streifzug Geschichte

mechanische zum
Hilfsmittel Rechnen
bis ca. 18. Jh.

mechanische zum seit
Aparate Rechnen
bis ca. 1760

ca. 600 v. Chr.: wahrscheinlich erstes Rechenhilfsmittel Abakus in China

1623: Rechenmaschine für 6-stellige + / - mit automatischem Zählerübergang Schickard
1624: unabhängig 8-stellige Rech.+/- auf + mit Komplement Tübinger Prof. Wilhelm Schickard
1673: Rechenmaschine mit Staffelwalzen 4 Grundrechenarten franz. Mathem. Blaise Pascal
Entwicklung Dualsystem Gottfried Wilhelm von Leibniz

1733: 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. Berkwerkingenieur Hermann Hollerith

21 mechanisch
22 wegen verhinderten Schaltzyklus Aufbau mit Relais

1941: elektromagnetischer Dualcode-Rechner (Z3) mit Daten und Programm auf einem 8-Kanal-Lochstreifen
0. Generation (erster funktionsfähige mit Dualsystem) Konrad Zuse
(bereits Rechenwerk, Programmwerk, Speicherwerk)

elektronische Rechenanlagen

seit 1944: erster Rechenautomat, bei dem Programm erst in Speicher geladen Math.- John von Neumann
(Speicherprogrammierbarer Rechner)

1945: erster Röhrenrechner (1700 Röhren, 150 kW) John P. Eckert, John W. Mauchly
1. Generation ~3,3 min eine Kaputt z.B. Tabellenberechnung für US Army

Datenverarbeitungsanlagen seit ca. 1955

Texte, Bilder verarbeiten

Rechnergenerationen

1. 1945-1956 Vakuumröhren 40.000 Operationen/sec
2. 1955-1964 Transistor 200.000
3. 1965-1971 Small and medium 1.000.000
scale integration
4. 1972-1977 Large scale integration 10.000.000) Abgrenzung schnell
5. 1978-?? Very large scale integration 100.000.000)

- Ab 1954 Entwicklung Programmiersprache Fortran
1955 Erster Transistorrechner
1957 Entwickl. Magnetplatten Speicher
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

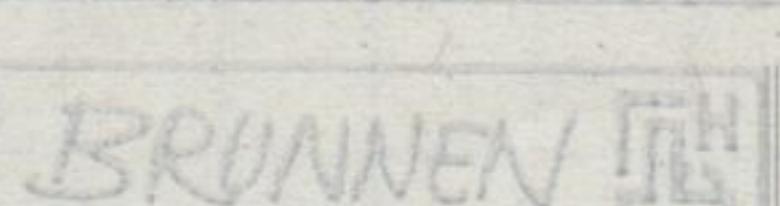
Ethik - Teilbereich Philosophie - Bewertung menschlichen Handelns

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

Kritik gabs vor langen

Joseph Weizenbaum, "Das Internet ist ein riesiger Misthaufen"

Antisoziale Systeme, Kl., Diskriminierung, Deepfakes



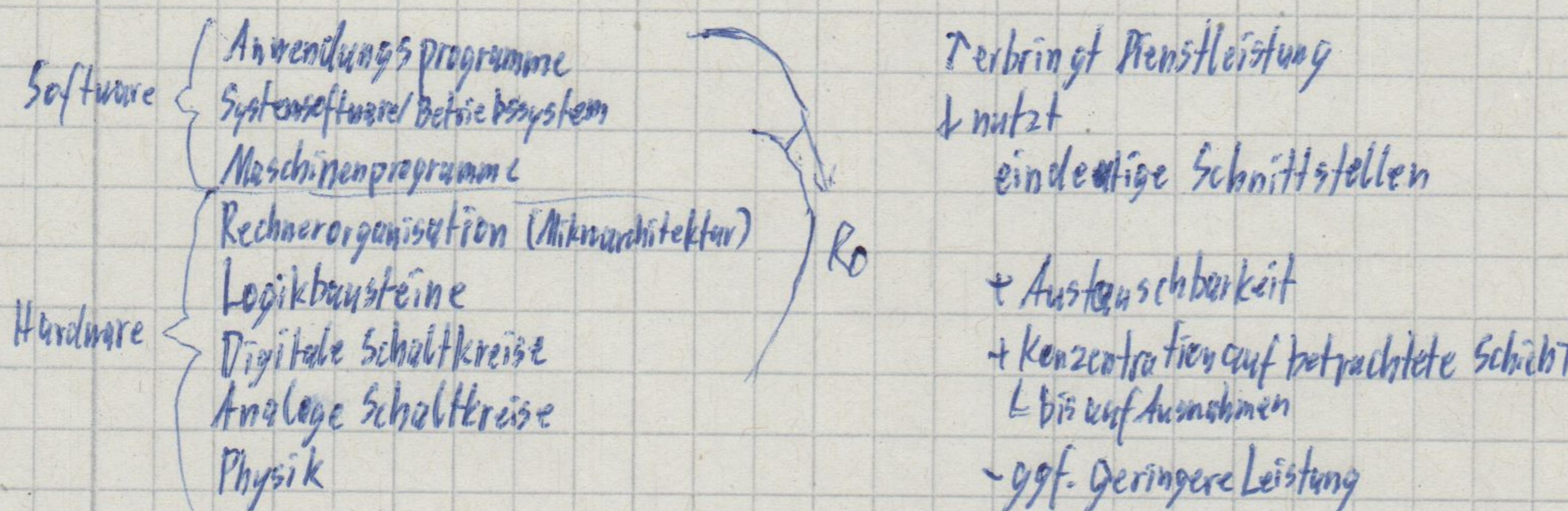
Gesellschaft für Informatik e.V. (GI) mit Leitlinien

moralischer Konflikt Gegenstand gemeinsamen Nachdenkens und Handelns

Ro

Abstraktion - Verstecken unnötiger Details

(ein) Schichtenmodell



Datenverarbeitungssystem ~ Computer ~ Rechner ~ Rechnersystem Syonym (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)

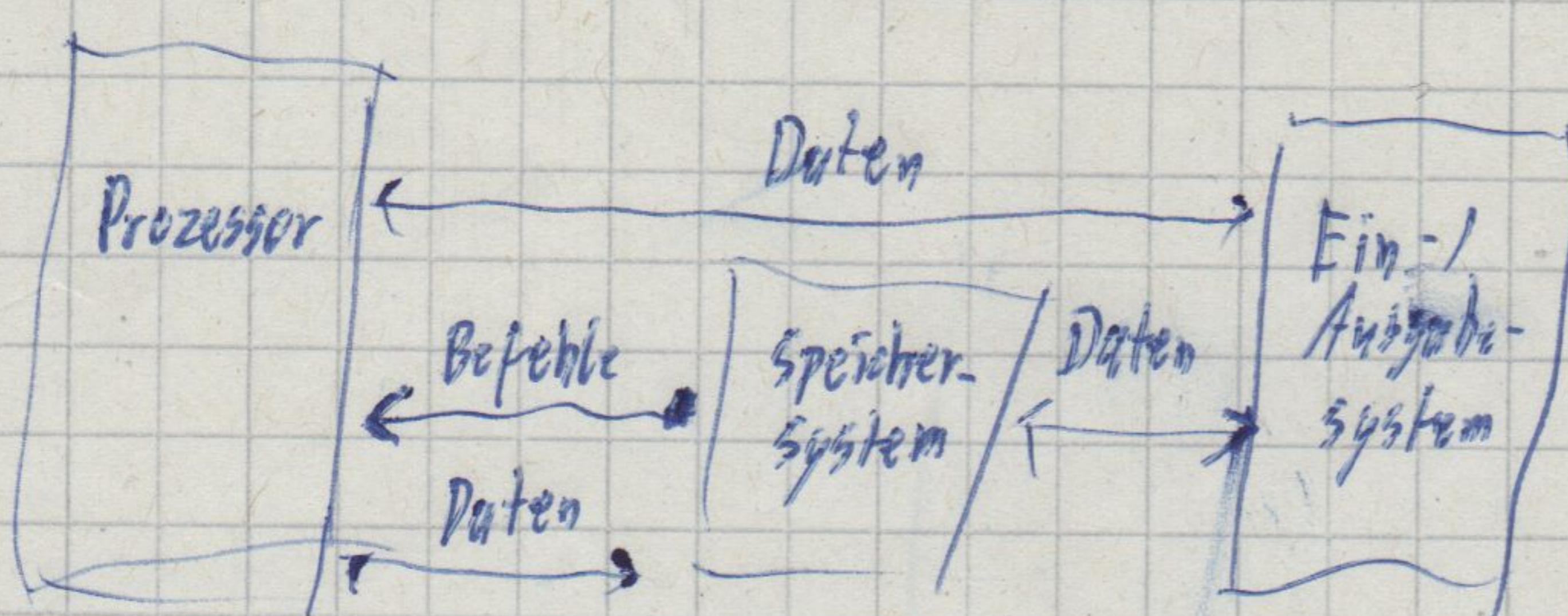
zurück fürbar Speichern " " (z.B. Ablegen, Wiederzuffinden, Löschen)

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

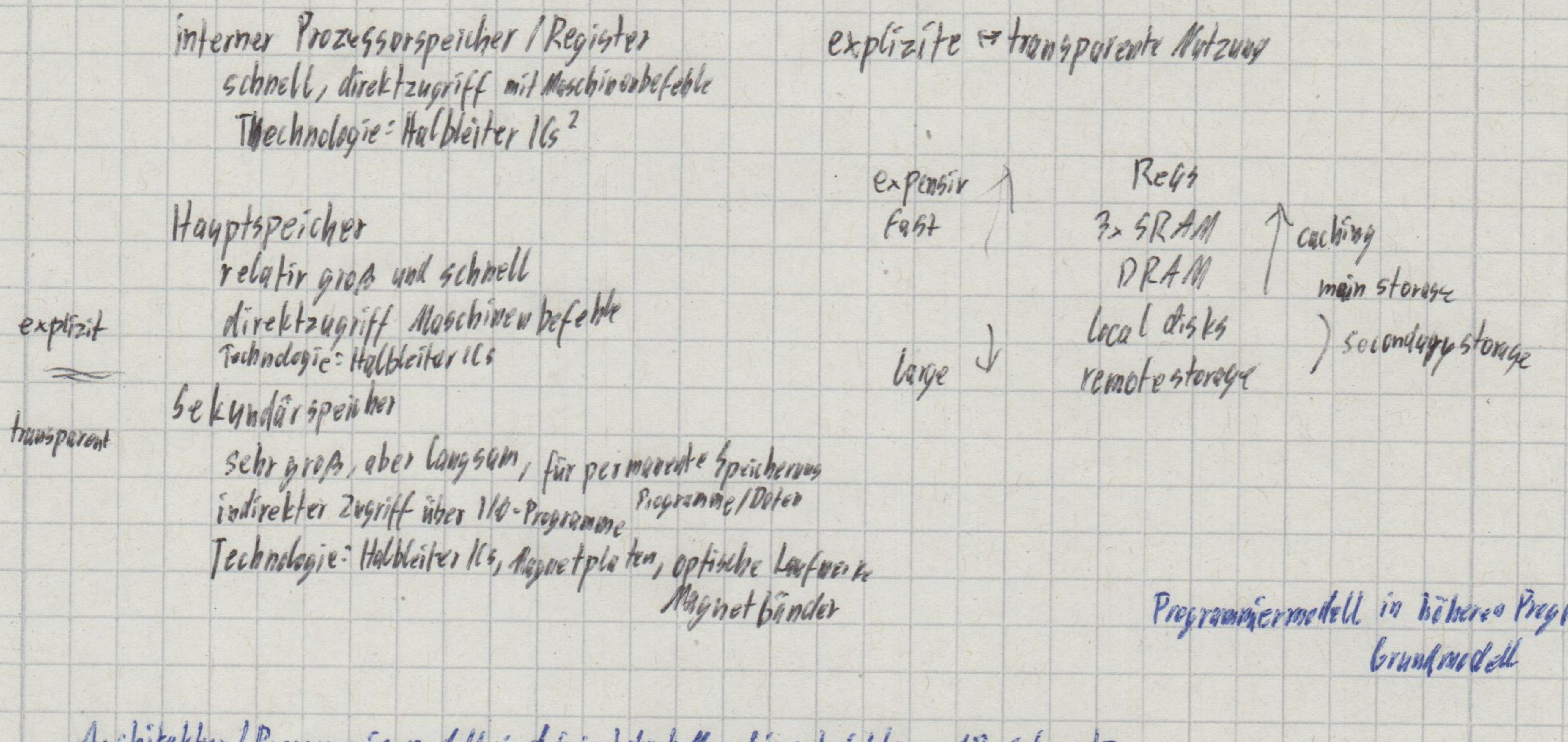
Kommunizieren " " (mit Benutzer, anderen Rechensystemen)

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

Komponenten



Speicherhierarchie



Architektur / Programmiermodelle: definiert durch Maschinebefehle und Registeratz
Programmierersicht statische Register zählen nicht

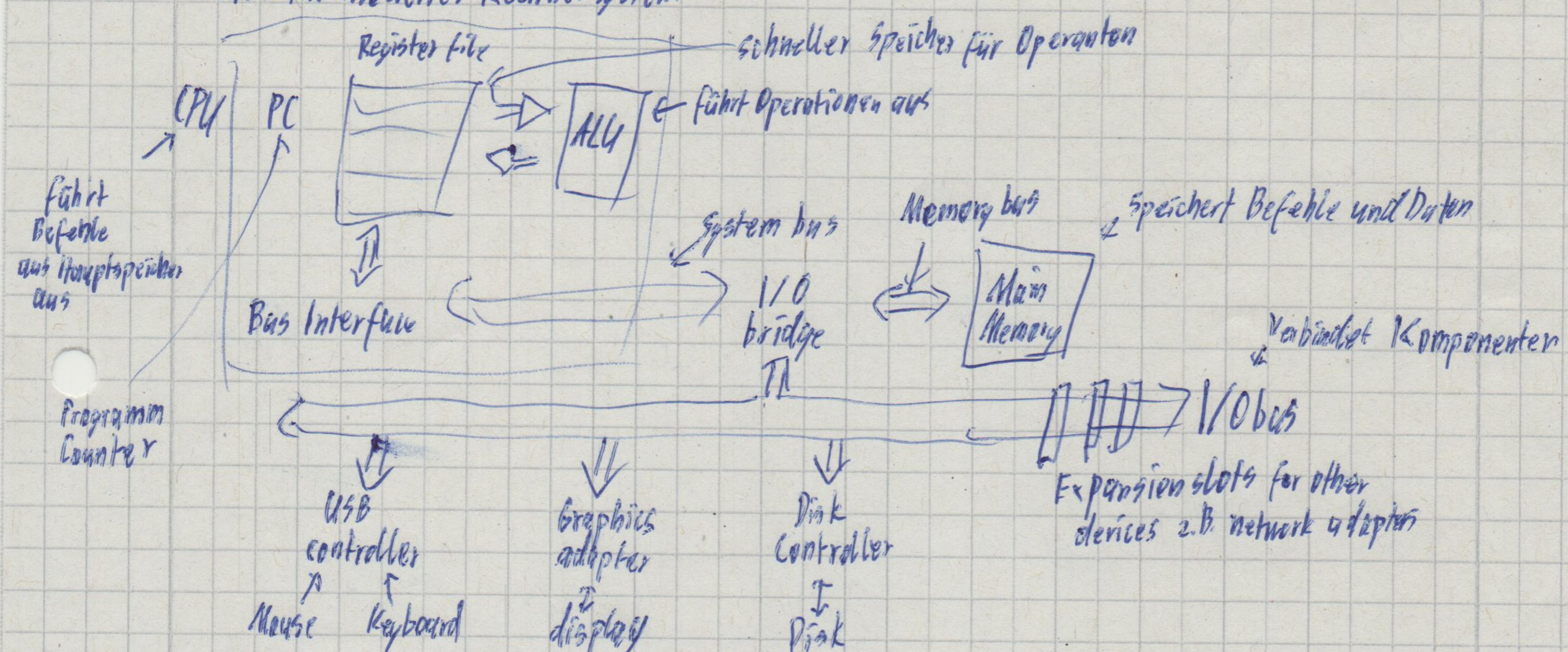
Mikroarchitektur: Hardwareimplementierung

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

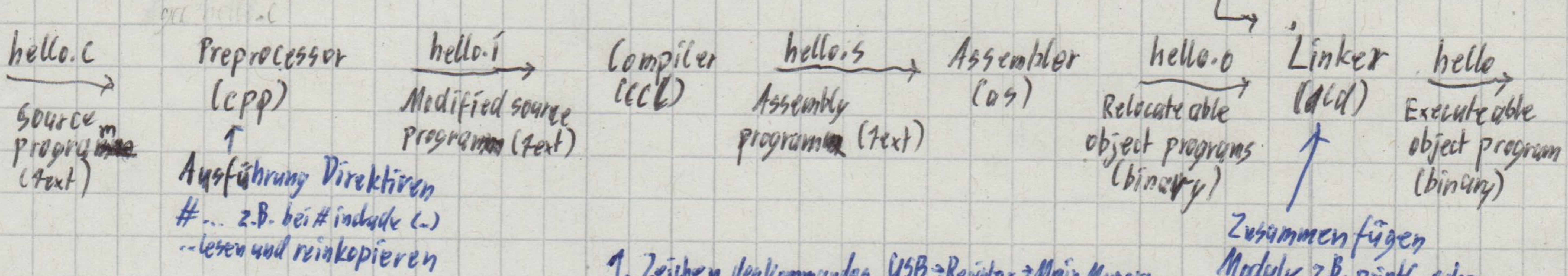
Vorlesung - ARM - Acorn/Advanced RISC Machines 1983 (heute Smartphones, Laptops, Supercomputer) (thick)

Raspberry Pi 4 ARM Cortex-A72

Struktur modernes Rechner system



Übersetzungsphasen



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 \rightarrow effizienteres Pipelining

Einteilung z.B. auch nach Anzahl Operanden in Maschinencode

z.B.

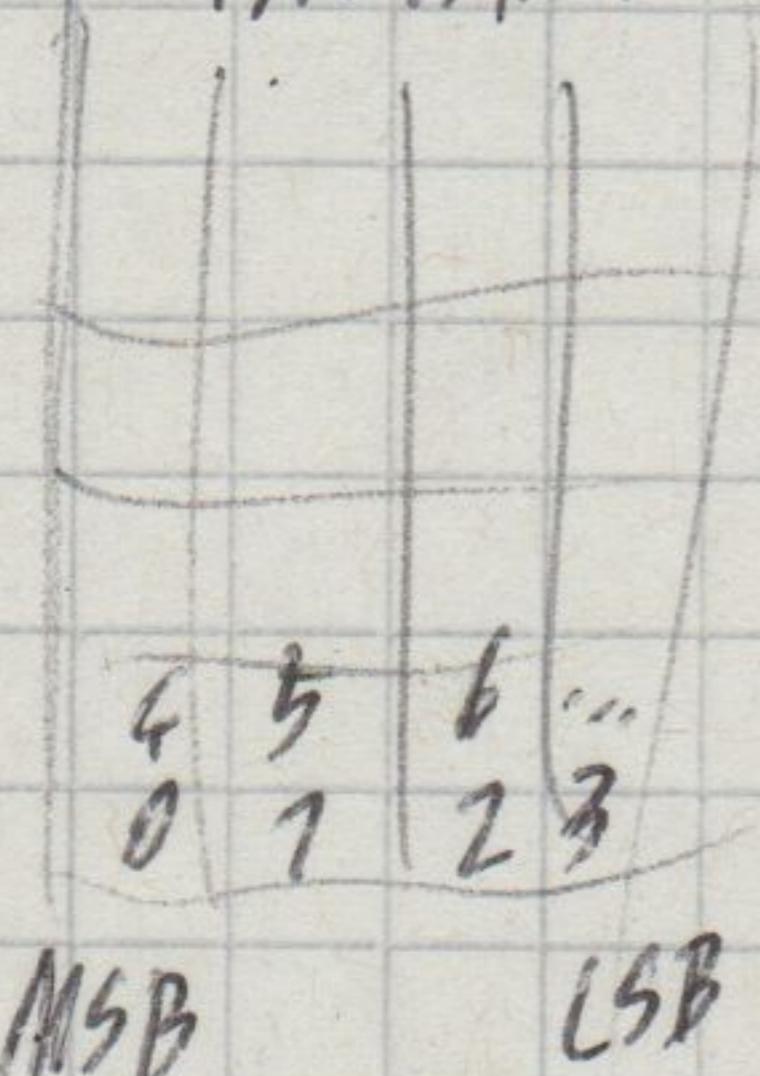
2-Adressmaschine (z.B. Intel Architektur)

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

Schemata für Bytenummerierung in Wort

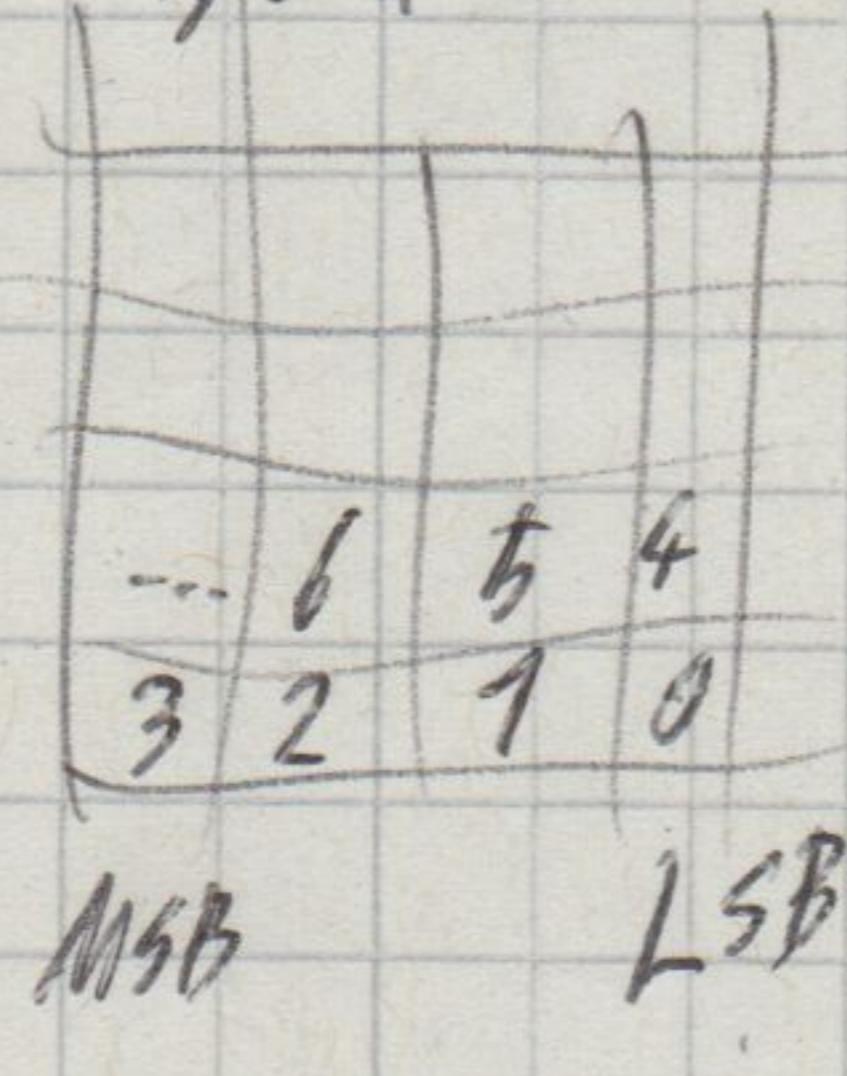
Big-Endian
(Motorola-Format)

Byte-Adressen



Little-Endian
(Intel-Format)

Byte-Adressen



ARM

↓

ARM 32-bit Wort,
byte-adressiert

\Rightarrow Werte Vielfaches von 4

ARM Registersatz

r0 Standard für Rückgabe

r1-r12

r13 - sp Stack Pointer

r14 - lr Link register (Rückkehradresse)

r15 - pc program counter

nur 13 freie Register \Rightarrow nur häufige Daten in Register, Rest Hauptspeicher

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

wichtigsten Statusflags

C - Carry flag

Z - Zero flag $\Rightarrow (r == 0)$

N - Negative flag $\Rightarrow (r < 0)$

V - Overflow flag

Assembler Befehle

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

[rBase] ↓
 Adresse / [rBase, H-]
 Adressarithmetik [rBase, rOffset]

Mehr Befehlsformate: + Flexibilität

- Hardware schwerer, langsamer

// str - store register to memory rZiel, Adresse

str
strh
strb

mov rZiel, #16bit

add r0, r1, /r2
add r0, r1, #12bit Zweierkomplement

Label-name: // Sprungmarke, Zielpunkt für Sprünge

// Sprünge

// unbedingt

b target

// Vergleichsbefehle

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

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

// Conditioncodes in ARM 64 Bit nur noch für Sprungentcheidung
z.B. beg, addeq, ... 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 (\geq) \geq für unsigned

LS C gesetzt (\leq) \leq für unsigned

GE N \Rightarrow V / N = V \geq für signed

LT N \neq V $<$ für signed

GT Z N = V \geq signed

LE Z N \neq V $<$ signed

.data
 var1: .word 5 // var1 im Speicher Wert 5 kann rolle 32bit sein
 adr-var1: .word var1 // Adresse von var1

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

main:

ldr r0, var1 // Ldt Wert
 ldr r0, adr-var1 // Ldt Adresse
 ldr r0, =var1
 ldr r0, [r0] // Ldt Wert

Sprachkonstrukte übersetzen

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

if (r0 == r1)
// if body \Rightarrow cmp r0, r1
else // else body
b L2
L1:
// if
L2:

while (r0 != 128) \Rightarrow WHILE:

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

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

a[i] \Rightarrow // r0 Basisadresse r1 = i
// frag 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

Für Aufruf Tiefe 1 manuelle Lösung
allgemeiner Register auf Stack sichern

bl schreibt Rückkehradresse in r74 (lr)

Stack

LIFO

zum temporären Speichern
dynamisch (dient sich aus, zieht sich zusammen)

bei ARM nach unten

r13 sp zeigt auf zuletzt abgelegtes Element

Pseudoinstruktionen für Stack

push Elr 3

pop Elr 3

Aufgerufener - callee

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

Fragen

lokale Variablen
lokaler/globaler Scope?

Unterprogramm sichert Register, die es verwendet will

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

Mehrfache Unterprogramme - Sicherung lr

z.B. //mor r10, lr
main: bl function
 mor lr, r10
 lr

typisches Unterprogramm

r9 bis r11 und r14 auf Stack sichern

Parameter in r4 bis r77 kopieren, weitere lok. Var. definieren
Implementierung Rechnung

Rückgabe in r0

Wiederherstellen r4 bis r77 und r14 Speicher freigeben
Rückprung

Speichernutzung
Programmgröße

Laufzeitanalyse bzw. Performance

Programm Profiling

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

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

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

(Mnemonics)

Assembler - Assemblerbefehl \rightarrow Maschinencode (Objektdateien) und symbolische Namen \mapsto Adressen
↳ Assembler
↳ Crossassembler: läuft auf X, generiert für Y
↳ Disassembler: MaschinenSprache \rightarrow Assembly (üblicherweise Verlust sym. 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 darin 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
 - Bei Aufruf 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) OS, version, type start program section headers
↳ .text: Maschinentabelle
↳ .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 (Anweisung)
↳ .strtab: Zeichentabelle für Symboltabelle und Debugging Symboltabelle

Linker/Binder: aus relocatable objekt files \rightarrow ein executable objekt file von Betriebssystem vorgegeben

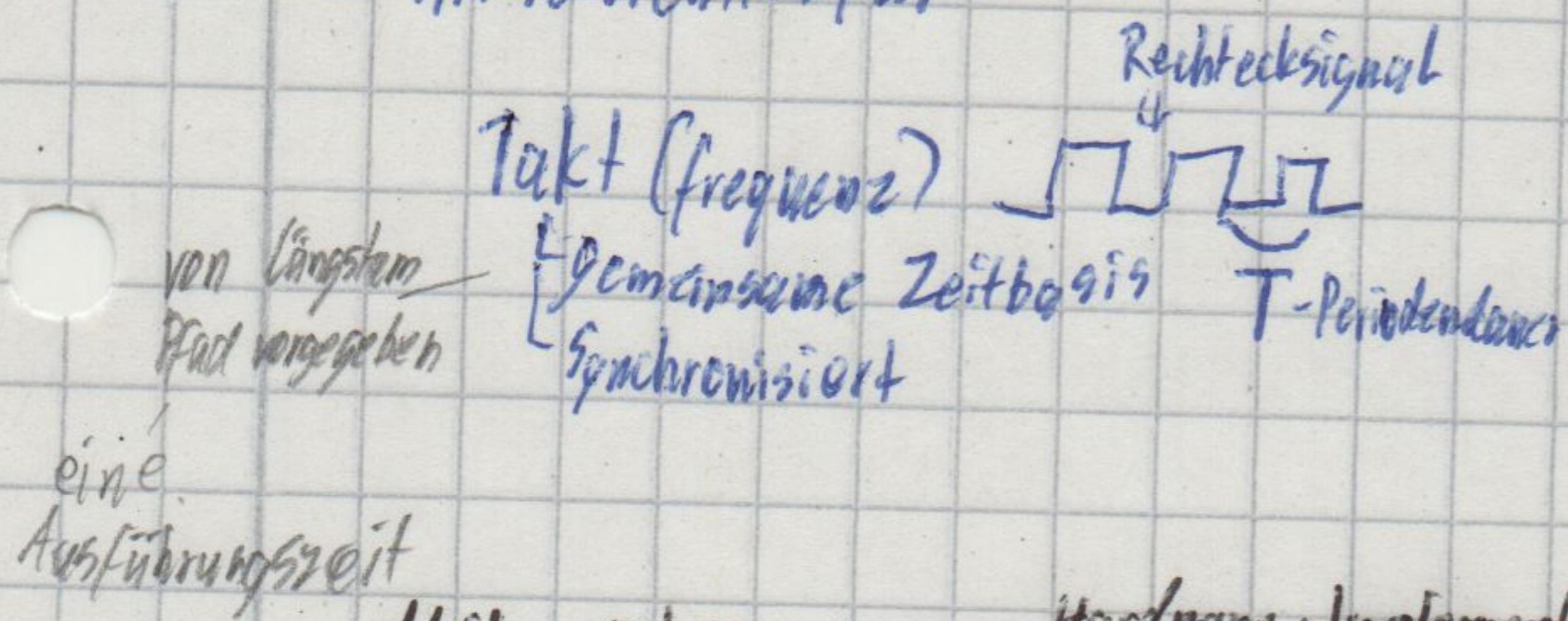
Loader/Lader: Systemprogramm: Programmmodul wird mit Startadresse in Hauptspeicher geladen

↳ absolutes Laden

↳ relatives

↳ dynamisches Laden zu Laufzeit

Mikroarchitektur



Einfaktimplementierung: 1:1 → braucht Harvard-Architektur

Mehr fakt implementierung: 1 Befehl in Teilschritte

Pipelined-Implementierung: 1:n und parallele Teilschritt ausführung

Mikroarchitektur ← Hardware-Implementierung → Architektur

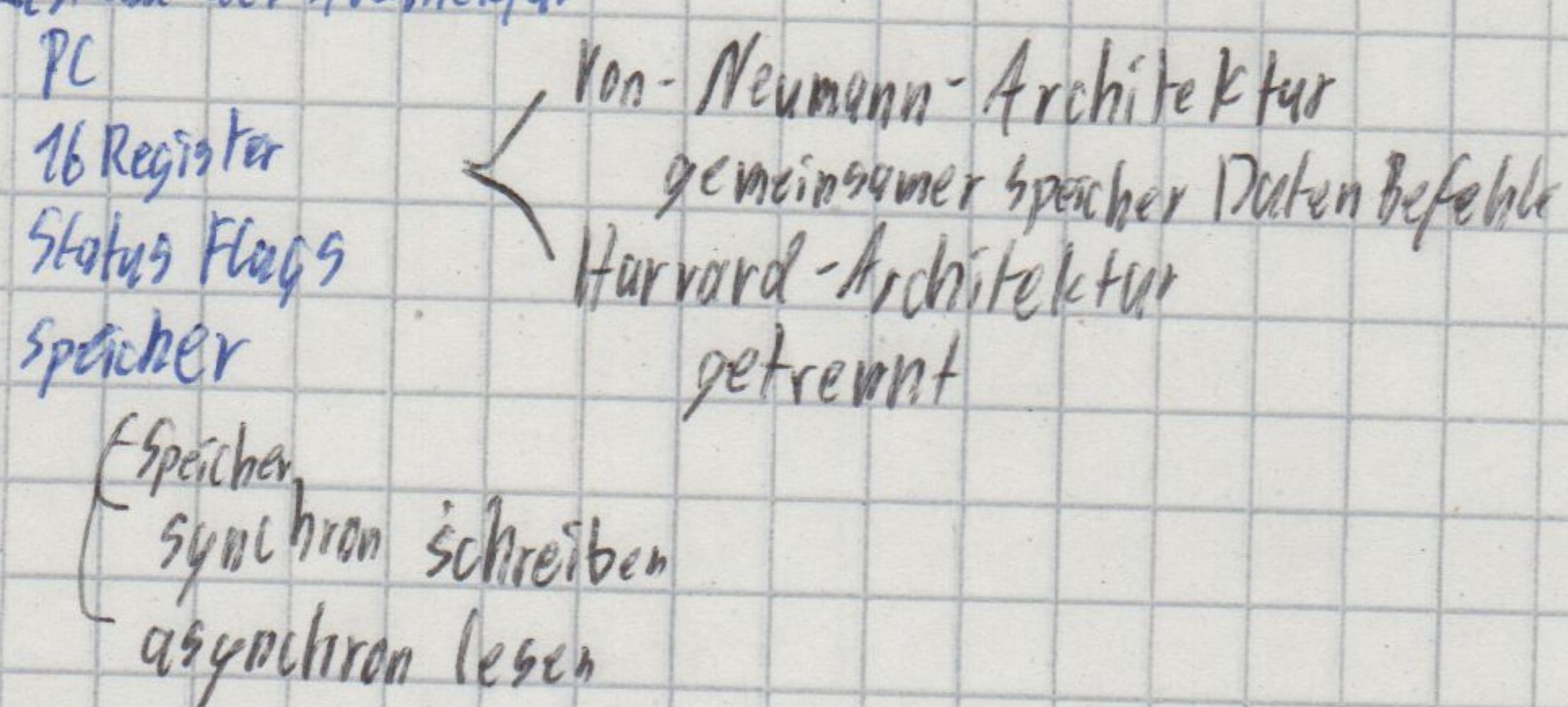
Datenpfad: verbindet funktionale Blöcke
Kontrollpfad: Steuersignale, Steuernetz

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)

Zustand der Architektur



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)

1. Befehlsholphase

2. Befehl dekodieren

Steuernetz

Lesen Quelloperander
(Erweiterung Intermediates)

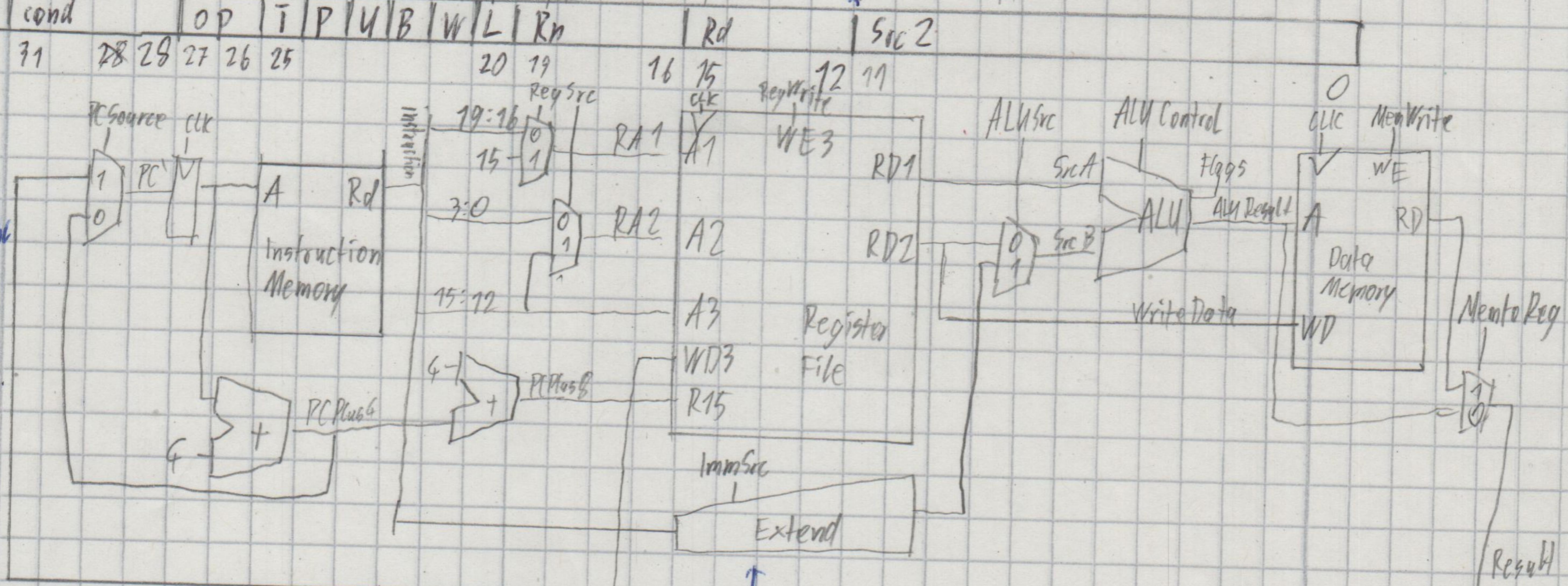
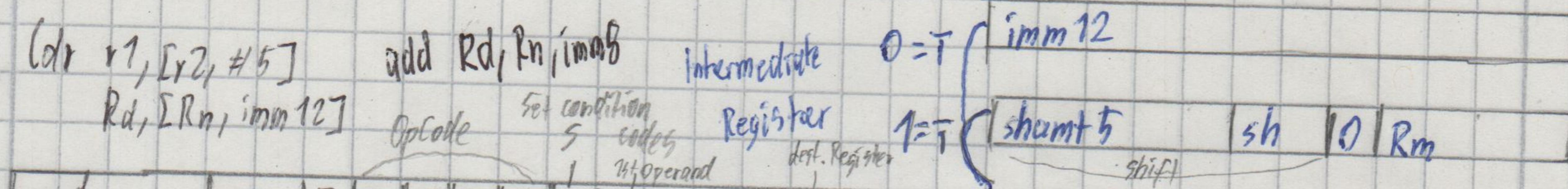
Wert/Speicher Berechnung

(Daten/Speicher/Speicher/Register)

Berechnung Adresse nächster Befehl

3. Befehlausführung

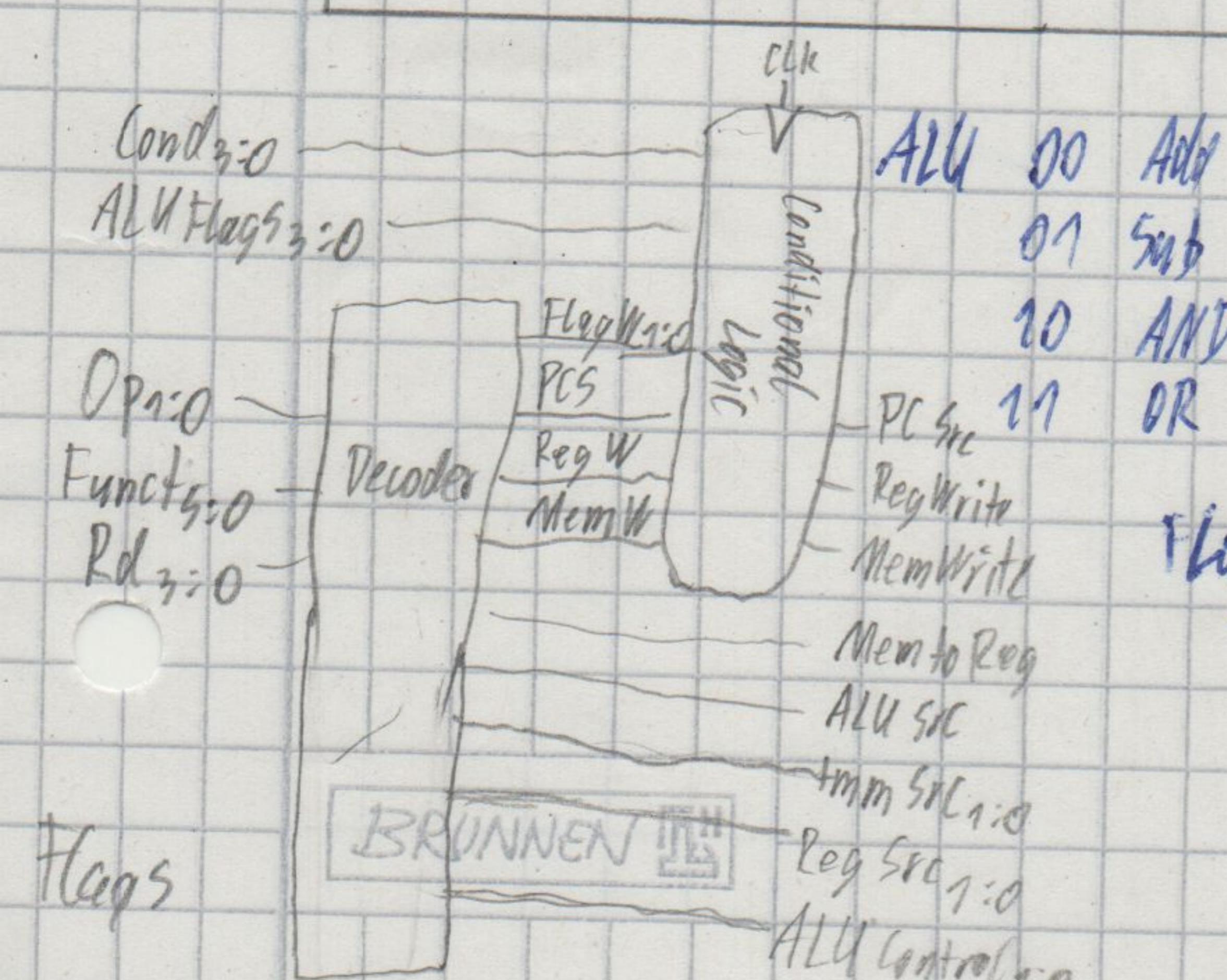
rotate kram



Cond 3:0	ALUFlags 3:0	ALU 00 Add	00 24b0, Inst 7:0
Op 1:0	Funct 5:0	01 Sub	01 20b0, Inst 11:0
Rd 3:0	Decoder	10 AND	10 b6Inst233, Inst 23:0 003 sign-extended
Flags		11 OR	multiplied by 4

Flag W1:0 → Flag W1=1 NZ sonst

Flag W0 = 1 CV sonst



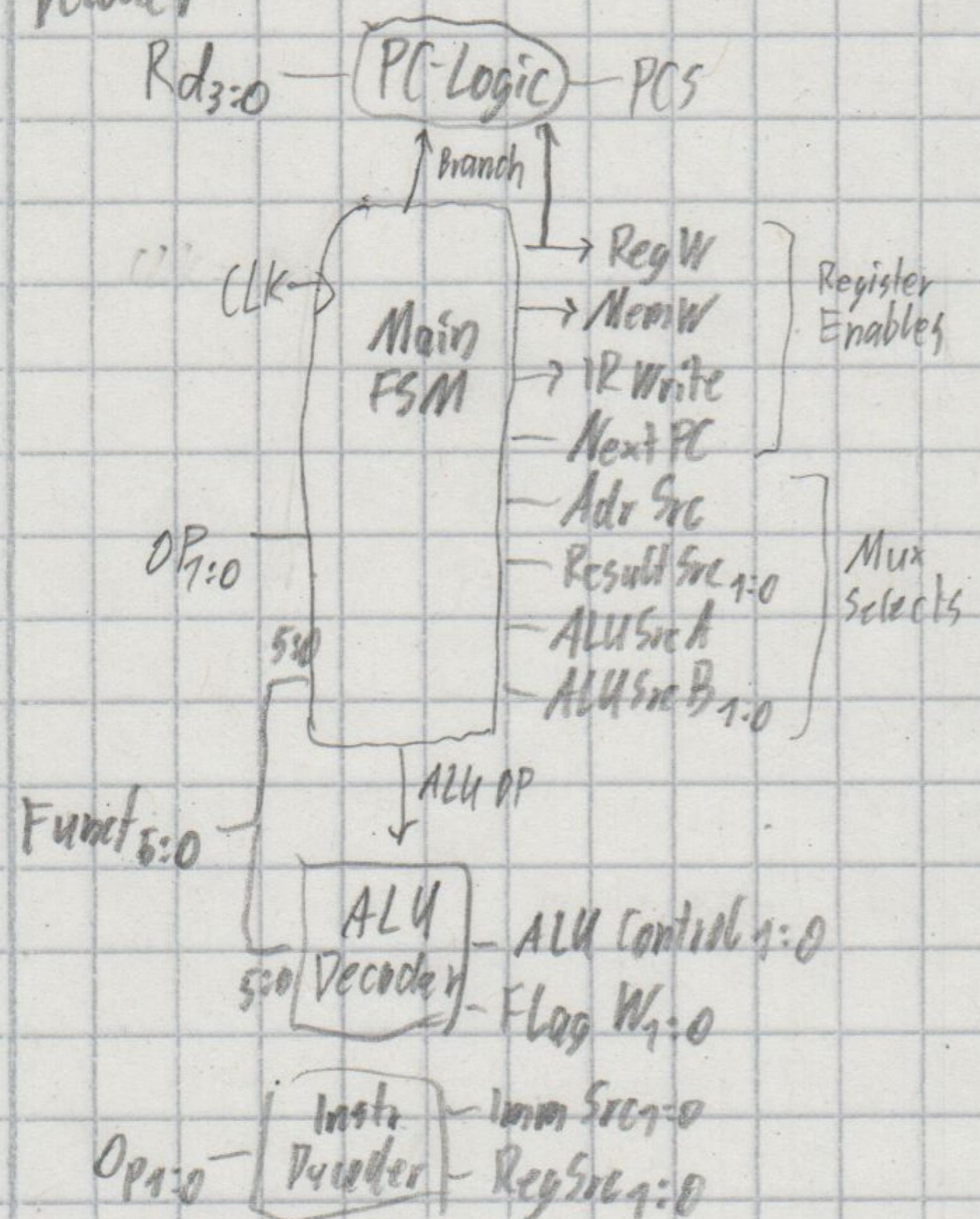
Jeder Befehl wird in Teilschritte zerlegt

Von-Neumann-Architektur

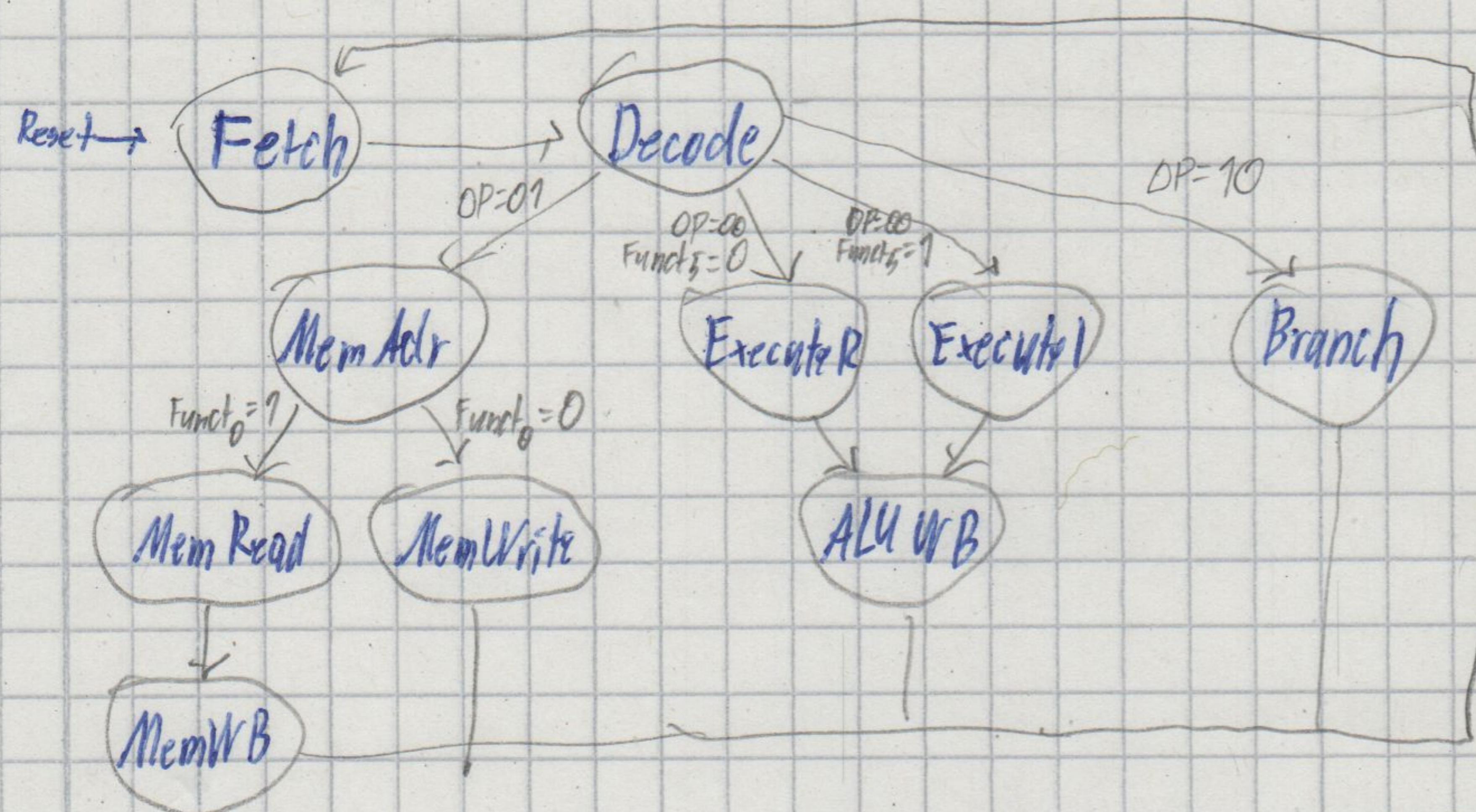
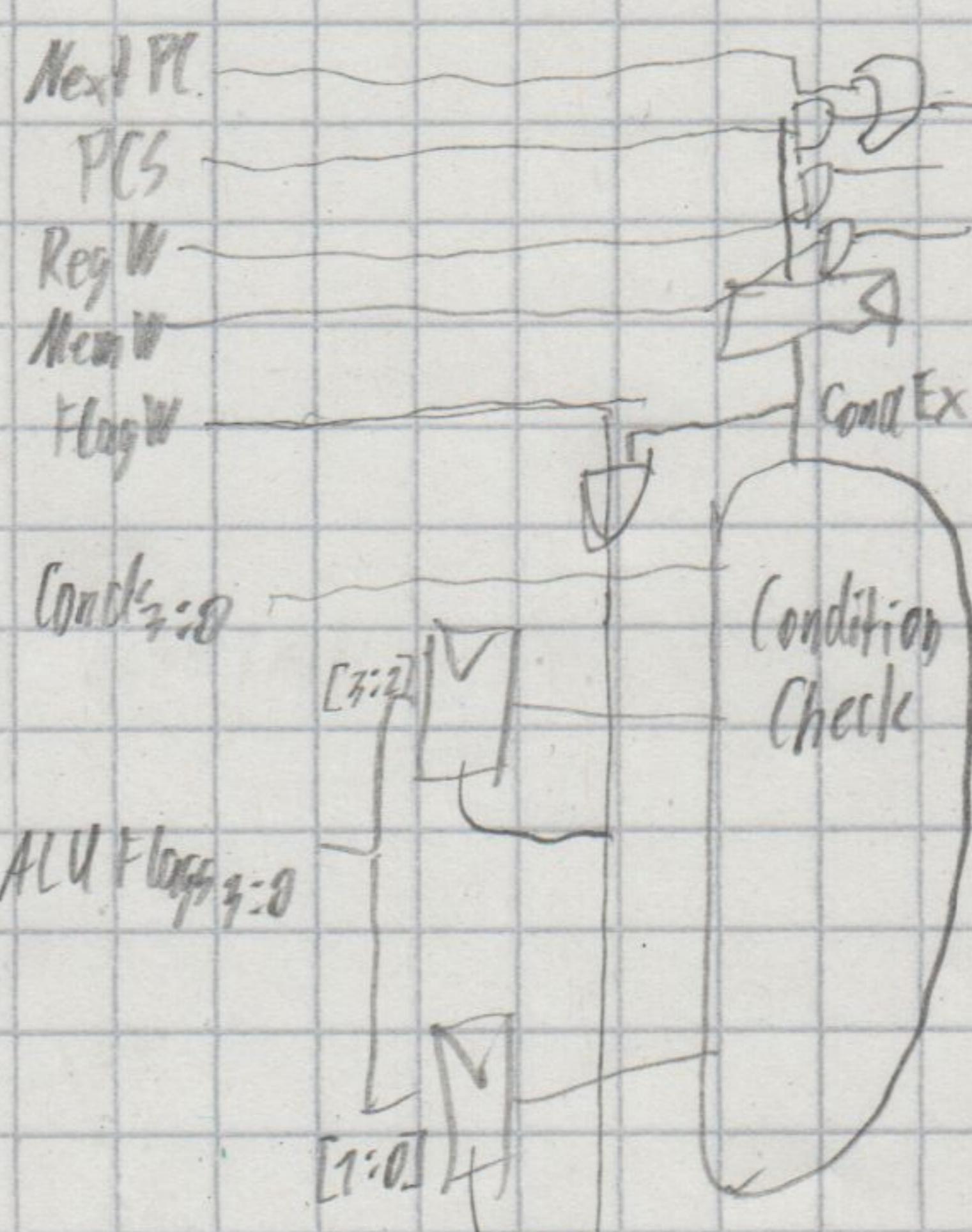
Mehraktivprozessor

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

Decoder

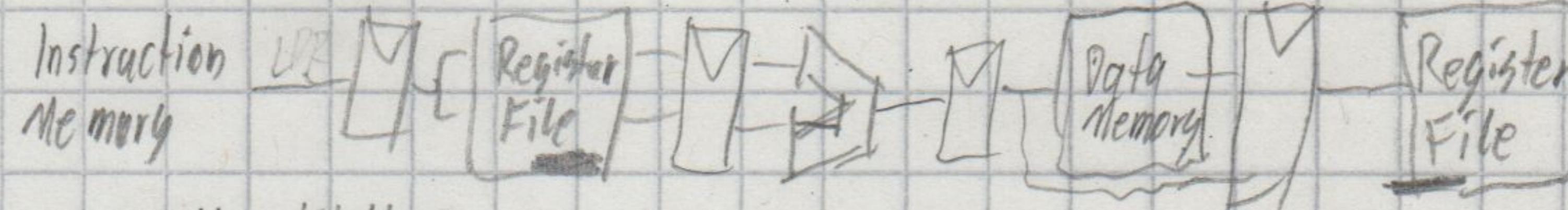


Conditional Logic



△ 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)

f. WAD muss auch durch Register gefabriert werden.

PC kann direkt ins Register File schreiben (da eins vergeben)

Hazards

Instruktion hängt von noch nicht verhandenem Ergebnis ab.

Hazard-Unit

Data Hazard: Neuer Wert noch nicht in Registerfeld

z.B. RAW - Read-after-Write

Lösungen

- Compilezeit
 - nops einplanen (no operations)
 - wie viel Zeit wird gebraucht
 - je nach Cache z.B.
- Laufzeit
 - forwarding
 - bypassing (Register durch abkürzung)
 - stalling - Prozessor anhalten bis Daten da
 - Befehlspipeline wiederholen

Control Hazard: unklar welche nächste Instruktion bei Verzweigungen

Lösung:

Flush wrongly started
third input of Registers

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

z.B. Taktfrequenz

Anzahl Prozessoren
Größe, Art des Speichers

Käufer, Designer

Preis \leftarrow Leistung
Verhältnis

Energiebedarf

Benutzer: Antwortzeit
Rechenzentrum: Durchsatz

Welche Aufgabe
repräsentativ?

Antwortzeit - reale Zeit Systemperformance

Ausführungszeit - CPU Zeit ETthreads Prozessorgleich

↳ System CPU time
User

Unix:
time-Befehl

CPI - cycles per instruction

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

MIPS - Million Instructions Per Second

früher Erhöhung Taktfrequenz

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

\Rightarrow Wärme schwer abzutransportieren

\Rightarrow Parallelrechner

Ausführungszeit = #Instructionen \cdot CPI \cdot $\frac{1}{IPC}$ Sekunden pro Takt

MFLOPS - Million Floating Point Operations per Second

Klassifikation von Flynn

Instruction- Data- Stream	SI	M1 zu einem Zeitpunkt
SD	SISD	Multiple Instructions
single Data	Von-Neumann Rechner	SIMD
MD	SiMD	MIMD
Multiple Data	Feld-, Vektor-Rechner	Multiprozessorsysteme

+ etabliert

- sehr grob

↳ Pipelining, Wortbreite, Schichten

Verbindungsstrukturen, Speicherorganisation

\Rightarrow SIMD Befehle, SSE Einheiten

↳ SIMD Extension

Streaming

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

	fraktion	bias
single	31 bit 8 bit 23 bit 1 0	127
double precision	63 bit 11 bit 52 bit 1 0	1023
extended	79 bit 15 bit 64 bit Mantisse 0	16383

s Charakteristik

Zahlendarstellung

integer
positiv
Dualcode

+/-
Vorzeichen & Betrag
Verschiebung
Komplement

real
Festkomma (Kommasstelle definiert)
Gleitkomma / halblogarithmisch
(Kommasstelle Bestandteil Zahl)
ANSI / IEEE 754

single
double precision
extended

fraktion

bias

Benchmarks - repräsentative Programme Instruktionssätze auf gängige optimiert Typen

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

Kernels - kurze isoliert zur Ausführung gebrauchte kritische Programmabschnitte

Toy Benchmarks - kleine einfache Programme z.B. Quicksort

Synthetische - speziell entwickelt soll einzelne Komponenten untersuchen
ein Befehl

Linux: cat /proc/cpuinfo

Taktfrequenz

Cache-Größe

Floating Point Unit

1989 SPEC - Standard Performance Evaluation Corporation

mebriger Hersteller zunächst reale Programme

Praxis: Anpassung aktueller Eigenschaften

Kostenpflichtig auch für GPU, Mailserver, Data

frei gzip, bzip, Linpack (für Supercomputer)

kein

Leistungsrandwert \rightarrow BogoMips \leftarrow bei boot ermittelt

BRUNNEN zum Kalibrieren interne Wurtschleife

Betriebssysteme & Ausnahmebehandlung

in Anfangen 1960: Programmierung durch Entwickler

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

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

Betriebssystem: Grundlage Betriebsart Anordnung von
Steuerung, Überwachung Programme

Aufgaben

Verarbeitung - 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
Prozesszustände - Prozessor - Wechsel per Software

↳ Maschinenzustand (stop, laden, tätig)

↳ Betriebszustand

Privilegierungszustand - Menge erlaubter Befehle

oft Anwenderzustand

Systemzustand - privilegiert privilegiert
voller Befehlsraum

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

↳ Programmbezogene

↳ durch Befehl, trifft Verursacher z.B. Exceptions

↳ synchron / intern

arithmetisch, Adressfehler, Rechte

↳ Systembezogen

Spezialbefehl zum Einleiten Systemaufrufe

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

↳ asynchron / extern

Prozessoraufrufe

Maschinenfehler

Anwendungssoftware

Dienstleistungssoftware

Systemumgebung
Systemgrenze

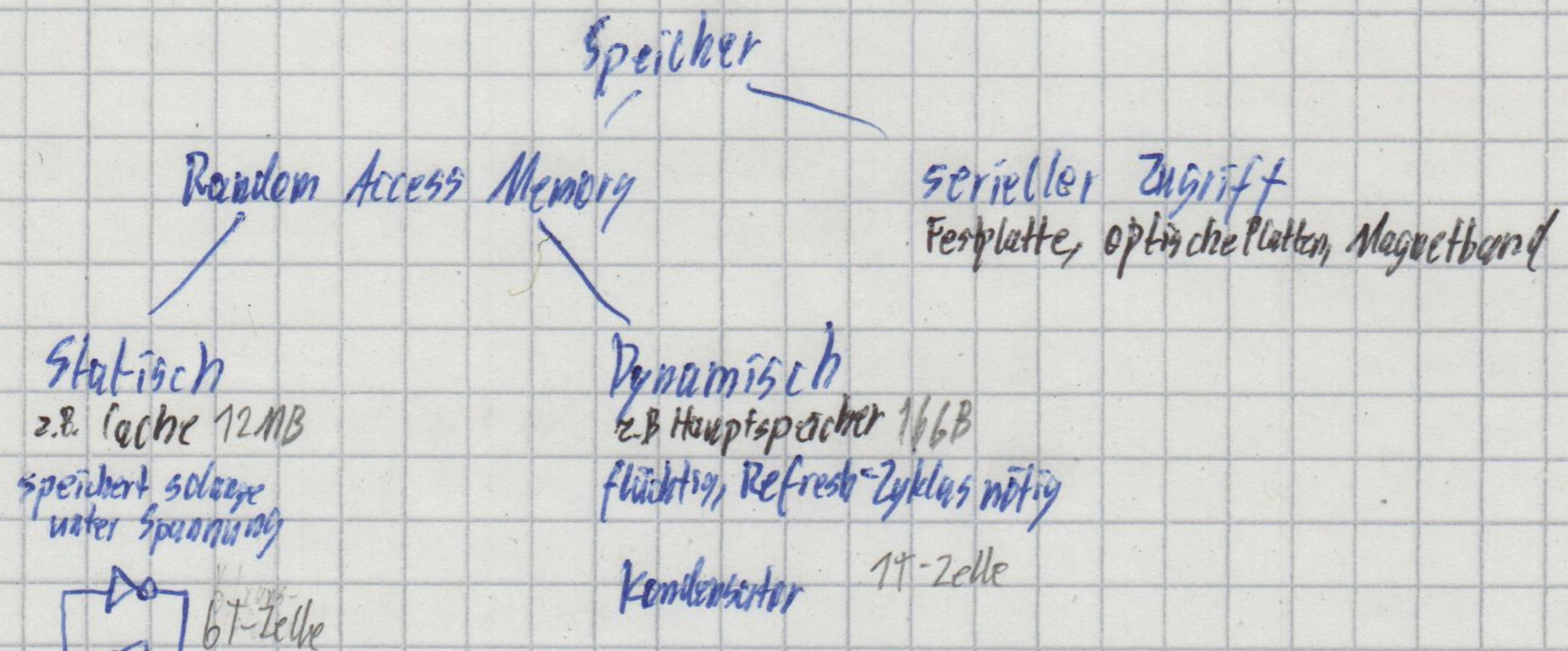
System-
software

Hardware

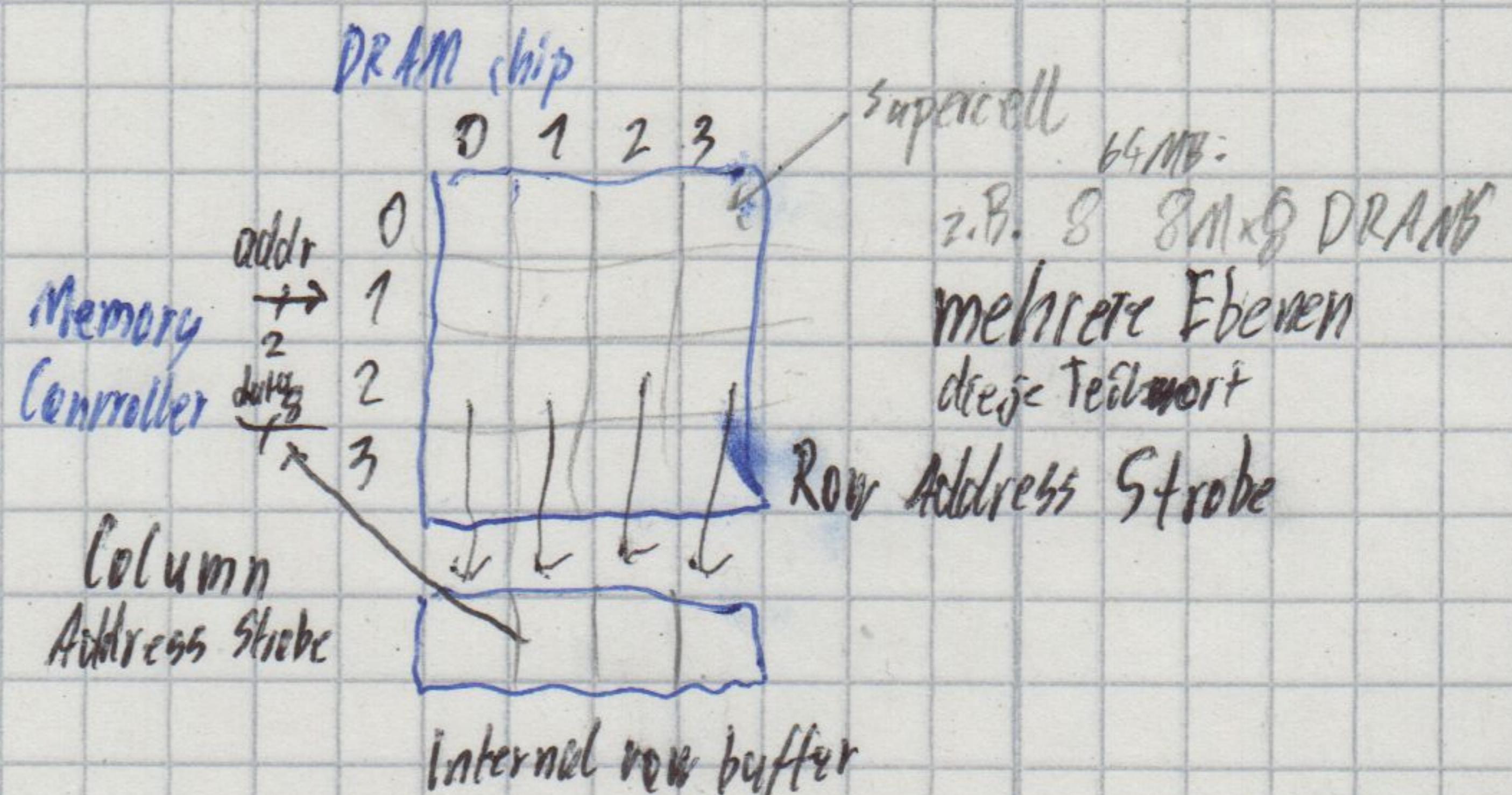
Zentraleinheit

Peripheriegeräte

auch BUSES protokoll relevant
 Speicherhierarchie Bandbreite - Breitsec
 $\frac{1 \text{ Bit}}{1 \text{ MB}}$ Zykluszeit - Min zwischen Zugriffen
 Kosten und Zugriffszeit = durchschnittliche Zeit
 Geschwindigkeit, Kapazität ein Wett aus Speicher
 Zugriffsvorhaben
 Änderbarkeit, Permanenz



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



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

Performance relevant \Rightarrow bei mehrdim. Array erst rechteste iterieren

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

Anfrage Block d
 Suche in Cache

z.B.	CPU Register	4/8 Byte	0
	L1 Cache	64Byte Block	1
	L2 Cache	64Byte Block	10
	L3 Cache	"	30
	...		
	Platten Cache	Disk sektor	100 000

ARM

Core 0

Regs

L1
d-cache
i-cache

L2 unified cache

L3 unified cache

Main Memory

Core 1

Regs

L1
d-cache
i-cache

L2 unified cache

Cache HIT

Cache Miss

hole d auf Ebene k
entl. Ersetzung

Zufallsersetzung
Least-recently used (LRU) Ersetzung

Cache
Memorys

Bus interface

system bus

→ ALU

Register File

→ Main Memory