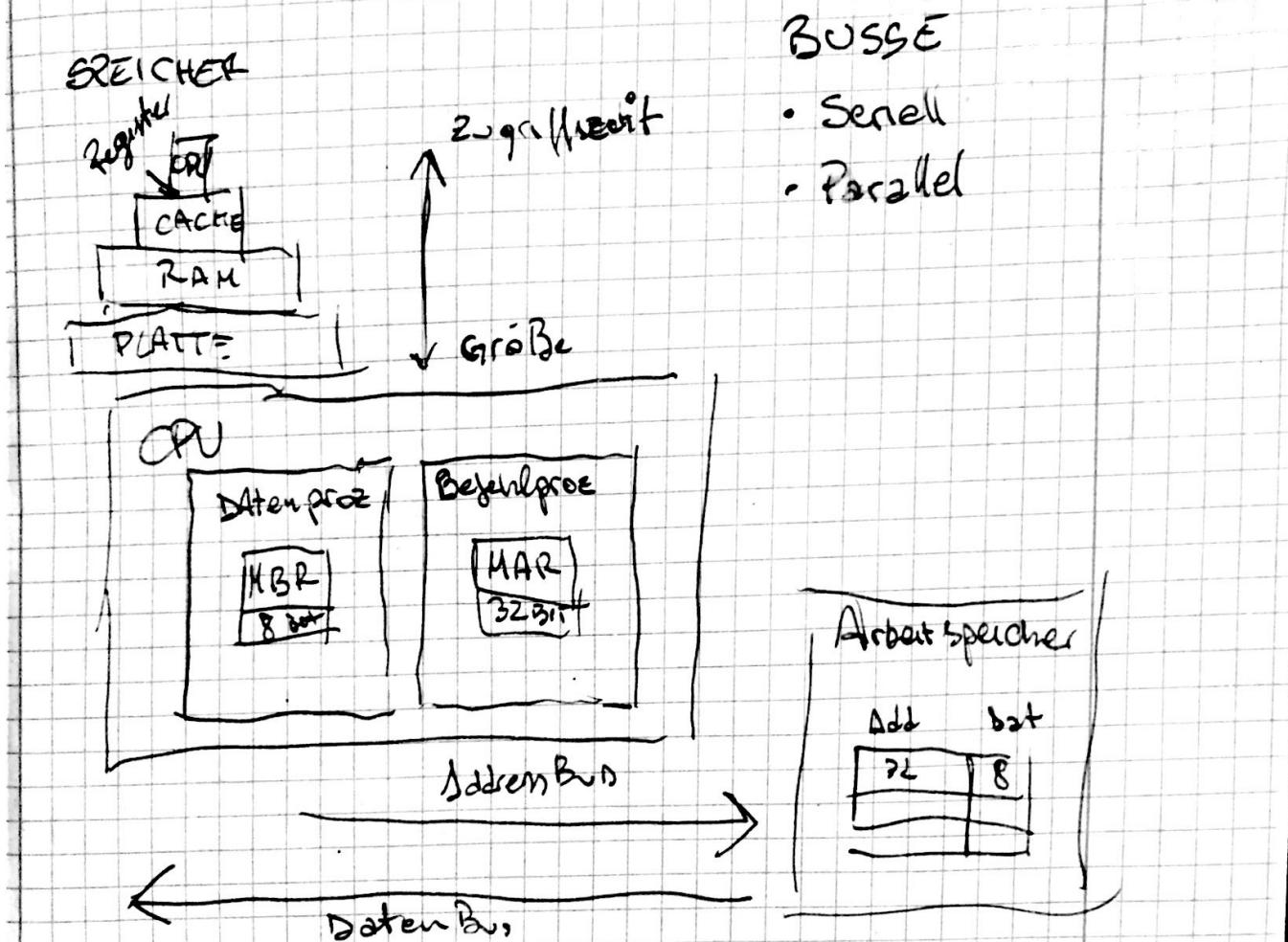


Von - Newman - Rechner

Komponenten

- CPU
 - Haupt Speicher
 - E/A Geräte (Platte, DVD...)
- Rechenwerk
- Memory Buffer
- Prozessor
- Multiplikationsregister
 - Link Register
 - Befehlregister
 - Speicheraddr. Register
 - Befehlzähler
 - Befehlsdecoder
 - Steuerwerk



Klassifikation

Preis

• PC

- Workstation
- Mainframe

Klassif.

Machine code
Setz

• CISC

• RISC

SISD Sing Instruction
Sing Data

SIMD Sing I mult. D

MISD Multis Sing D

MIMD Mult I Mult D

BOOLSCHE ALGEBRA

and OR not nand nor
 $\rightarrow \Rightarrow \rightarrow \Rightarrow \rightarrow \Rightarrow$

- Kommutativ
- Distributiv
- Assoziativ
- De Morgan'sche Regel

DECODER

n Eing. 2^n Ausg. nur ein Ausgang ist 1

	in1	in2	out0	out1	out2	out3
decod	0	0	1	0	0	0
	0	1	0	1	0	0
	1	0	0	0	1	0
	1	1	0	0	0	1

Encoder

2^n Eing n Ausg nur ein Eing ist 1

	in0	in1	in2	in3	o0	o1
encod	0	0	0	0	0	0
	0	1	0	0	1	0
	0	0	1	0	0	1
	0	0	0	1	1	1

Multiplexer (Selektor)

n Eing. $\Rightarrow \log_2 n$ Selektoren

SCHALTFAKTION $f: \mathbb{B}^n \rightarrow \mathbb{B}^m$ mit $n, m \in \mathbb{N}$
 \nwarrow Minterm \nearrow Maxterm $n \geq 1, m \geq 1$

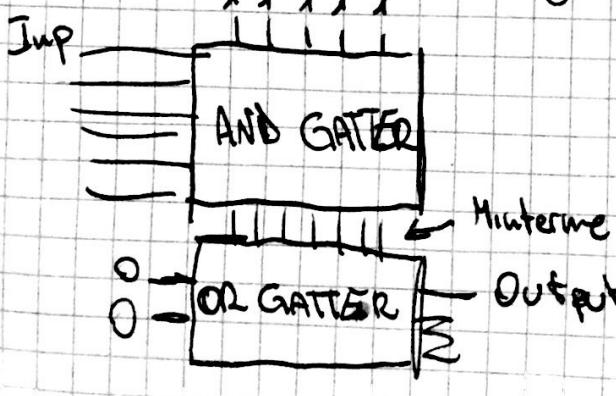
KNF: $(\bar{a} \vee b) \wedge (\bar{b} \vee c)$ \Rightarrow alle Nuller (

DNF: $(ab) + (\bar{b}c)$ \Rightarrow alle Einser (Einzeliger Index)
 \nwarrow Minterm

$f=0 \Rightarrow \exists DNF \quad f=1 \Rightarrow \exists KNF$

Funktion aus AND, OR, NOT ist funktional Vollständig

PLA (Programmierbare Logische Arrays) KOSTENRENUZIERUNG



$f: \mathbb{B}^r \rightarrow \mathbb{B}^s \Rightarrow r$ AND-Zeilen
 s OR-Zeilen

• Hardware Prog.

• Software Prog.

GAITERN & REALISIERUNG

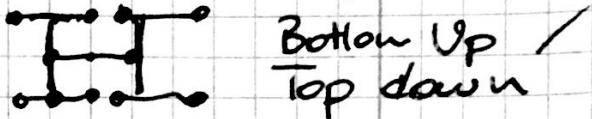
Silizium scheibe 8 cm / 26 µm Ø

$10^3 - 10^5$

- SSI Small scale Integration 10
- MSI Medium Scale Integration $^{10^2}$
- LSI Large scale I.
- VLSI Very Large Scale I. $> 10^5$

Gattern bestehen aus Kettenschaltungen auf unterschiedlichen Ebenen.

Es werden minterme ausgerechnet in einen Binärbaum der in "H" Form aufgebaut wird damit die Punkte auf dem Gitter bleiben (H-Baum)



OPTIMIERUNG VON SCHALTNETZEN

■ Karnaugh - Diagramm

Die Funktion wird minimiert

durch kürzen von unbedeutenden

Signalen. ES GIBT DON'T CARE ARGUMENTE

3	2	1	0	1	1	1	0
1	0	1	1	1	0	1	1
1	1	1	1	0	1	1	1
0	0	1	1	1	1	0	0

- Wahrheitstafel eintragen

- Einser markieren

in Zai 2x2 4x2 4x4 ...

- Minimale Inv. Blöcke

■ Quine - McCluskey Verfahren

Ist für größere Funktionen überschaubar und eignet sich für die Automation.

Schritt 1 : Bestimmung der Implikanten und dann Primimplikanten
Alle Minterme werden nach Anz. Nullen sortiert (1, 2, 3, 4)
Don't Cares werden gesucht und Zeilen zusammengefasst
und in eine Neue Tabelle geschrieben.
Wiederholen solange es geht.

Schritt 2 : Kostenminimale Auswahl der Primimplikanten.

Implikationsmatrix Aufstellen.

Zeilen wählen die Elemente aus anderen Zeilen enthalten.

Die gebrauchten Zeilen sind die Vötigen Minterme

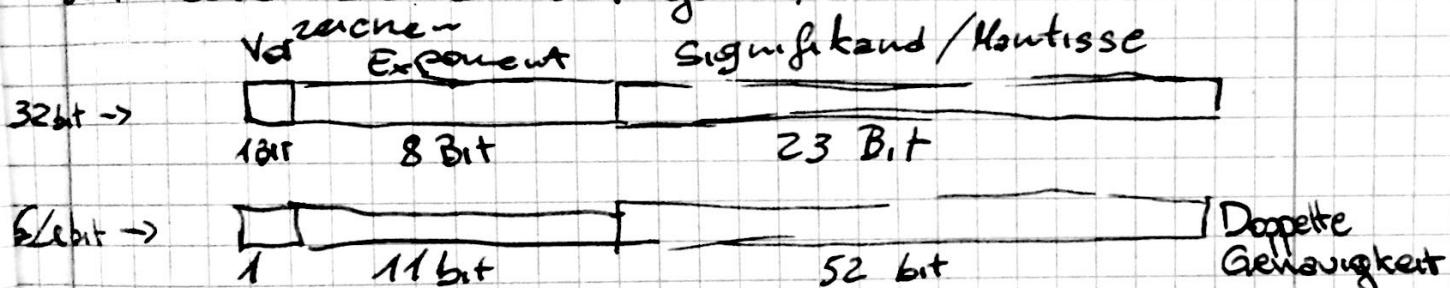
ARITHMETIK VON COMPUTERN

Vorzeichendarstellung bräuer ist problematisch. \Rightarrow Rear Trick.

- Einerkomplement : Verdrehen aller bits. \Rightarrow die Null hat eine doppelte Darstellung
 $-127 - 127$ $-2+6 = 1$ Fehler wenn über die Null addiert wird. dann es muss noch 1 dorthin werden.
 - Zweierkomplement : Einerkomplement + 1
 $-128 - 128$ \Rightarrow Alle bits stellenweise verdreht dann & Addieren.
 Es ermöglicht Subtraktion.

DARSTELLUNG REELLE ZAHLEN

wird durch IEEE-754 gelöst.



KONVERSION

11

10,25

Lichen O

$0,40 : 2 = 0,2$ $0,5 : 2 = 0,25$ $1 : 2 = 0,5$ $D = 10 \cdot 0,2$	wach kommt $0,2 \cdot 2 = 0,40$ $0,5 \cdot 2 = 1,00$ 1
---	---

spouwt 1010 01

1,01001-2³

$$127 + 3 = 130$$

130 | 010010_0_0

zurück

1 | 01111101 | 1010...00

Vote. -

$$E_{\text{ex}} \Rightarrow 0111\ 1101 = 125 \Rightarrow 125 - 127 = -2$$

Signifikanz 1010... -

$$\Rightarrow \frac{1}{2} + 0 \cdot \frac{1}{4} + 1 \frac{1}{8} + 0 \cdot \frac{1}{16} \dots$$

$$\Rightarrow 0,5 + 0,125 = 0,625$$

Formel

$$1 \cdot (-1) + (1 + 0,625) \cdot 2^{-2} = -0,40625$$

Darstellung von Speicherinhalten

ASCII: $8 \cdot 6 = 128$ Zeichen
 $0 - 1F \Rightarrow$ Sind Steuerzeichen

UNICODE: $2^{16} = 65.536$ Codepunkte

- Nicht alle Zeichen in Reihenfolge
Japanische Sortierung braucht eigene Tabellen
- Neue Wörter im Japanischen benötigen neue Codepunkte

Darstellung von ARRAYS Dimension des Feldes = ℓ Länge Array

Anordnung der Feldelemente wird für mehr dim. Speicherung in eine Matrix dargestellt und eine Formel für die Addressierung vom Speicher benötigt.

Mithilfe einer fiktiven Adresse ist die Berechnung der Speicheradressen erforderlich! Absolut Anfangs-Addr.

Fiktive Adresse = adr($a[0,0,0]$) Länge Array 3

Bsp. $Adr([7,8,3]) =$

$$= adr(a[0,0,0] + h[7,8,3] \cdot LOP)$$

$$= 1956 + 924 = 2880$$

$$AA = a[120] = 2100$$

$$s_1 = AA_1 - m_1 + 1 = 7$$

Gewicht 12
1 3 Ges: 8

$$s_2 = AA_2 - m_2 + 1 = 7$$

2 Ges: 3

$$s_3 = AA_3 - m_3 + 1 = 4$$

0

$$[((7 \cdot s_2) + 8) \cdot s_3 + 3] \cdot LOP$$

$$[((2 \cdot 7) + 8) \cdot 3 + 3] \cdot 3 -$$

Fehlererkennung und Korrektur

Falls Spannungsspitzen zu Übertragungsfehler führen werden Paritätsbits zur Fehlererkennung und Korr.

Speicherwort $\equiv m$ (Länge) $\exists n = m + r$

n : Gebrauchte Speich. m : Datenbits r : Paritätsbits

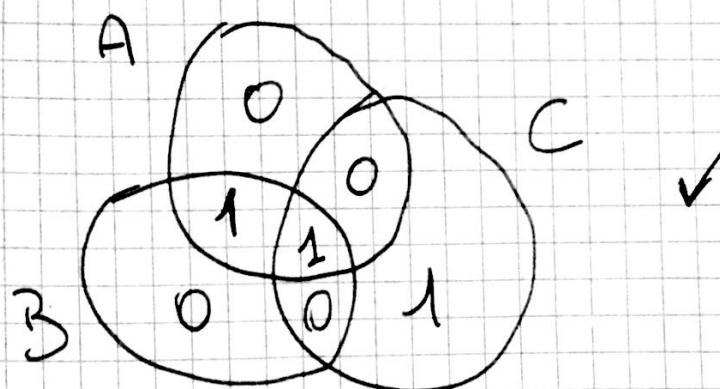
Hamming Abstand: Anzahl Bits, die unnd. geändert werden müssen, um von einem gültigen Bitcode zu einem anderen zu gelangen.

$$\Rightarrow \text{Minimales Abstand} = 2r + 1$$

$$\Rightarrow \text{Paritätbits auf Positionen } 2^0, 2^1, 2^2, 2^4, 2^5, 2^{10}$$

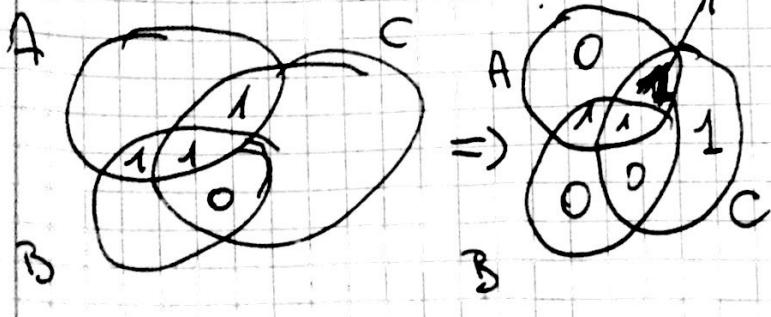
Einfache Fehlerkorrektur:

Codewort: $\begin{array}{c|ccccc} & 1 & 1 & 0 & 0 & 0 & 1 \\ \hline & 1 & 2 & 3 & 4 & 1 & 2 \\ & 1 & 1 & 0 & 0 & 0 & 1 \\ \hline & AB & ABC & AC & BC & A & B & C \end{array}$ können in ein
Daten m | r Parität Wenn-Diag darg.



Fehlererkennung

1 1 1 0 | 0 0 1 Fehler!



Planung Algorithmus: $m = 8$ Länge Speicherwort

1 1 0 1 1 1 1 1
P₁ P₂ P₃ P₄

P₁ prüft: 1, 3, 5, 7, 9, 11
P₂ prüft: 2, 3, 6, 7, 10, 11 } XOR
P₃ prüft: 4, 5, 6, 7, 12
P₄ prüft: 8, 9, 10, 11, 12 }

PRIMÄRE SPEICHER

Zugriffzeit ist sehr wichtig (\Rightarrow Bottleneck der VON-Neumann-Arch.)

CACHE: Schneller als Primärspeicher

Folgt das Lokalitätsprinzip: Es werden Informationen immer in benachbarten Zellen gespeichert

1. Temporale Lokalität: Zugriffe tendieren dazu sich zu wiederholen (z.B. Schleifen)

K: Zugriffsintervall
2. Spatiale Lokalität: Eine Zelle und ihre Nachbarn werden benutzt wegen sequentielle Abarbeitung

$$\text{Hit-Ratio} = h = \frac{k-1}{k}$$

$$\text{Miss Ratio: } e = 1 - h = \frac{1}{k} \quad c: \text{Zugriffzeit}$$

$$\Rightarrow \text{Mittlere Zugriffzeit: } \bar{t} = c + (1-h) \cdot m$$

Funktionsweise des Cache

direkte Abbildung Speicher \leftrightarrow Cache

$$\text{Cacheadresse} = (\text{BlockAdd. Speicher}) \% (\text{CacheGröße})$$

Bei direkter Abb. vom Speicher ist Mapping mehrdeutig \Rightarrow Wir fügen wir dem Cache ein TAG hinzu.

Bei 32bit Arch Wort = 1 Byte

Wird für 64 KB Cache

$$64 \text{ KB} = 64 \cdot 1024 \text{ Byte} = 2^6 \cdot 2^{10} \text{ Byte} = 2^{16} \text{ Byte}$$

$$2^{16} : 4 = 2^{16} : 2^2 = 2^{14} \text{ Wörter}$$

$$\Rightarrow 2^{14} \cdot (32 + 16 + 1) = 2^{14} \cdot 49 = 802816 \text{ Bit}$$

$\stackrel{!}{=} 98 \text{ KByte}$

CACHEEBENEN

Zum Verringern von Latenz und Vergrößerung der Bandbreite werden mehrere Cache Ebenen benutzt

Split Cache : Zwei unabhängige Caches \Rightarrow Doppelte Bandbreite

SIMM : Single Inline Memory Module

DIMM : Dual Inline Memory Module

ODIMM : Small Outline DIMM (Notebooks)