000.gif

Philipps-Universität Marburg

Technische Informatik I WS 2004/05

Dozentin: Prof. Dr. R. Loogen

Verwendete Zeichen und Symbole:

A	a	A	a		α	alpha	1
B	b	B	b		β	beta	2
C	c	C	c	Γ	γ	gamma	3
D	d	D	d	Δ	δ	delta	4
E	e	E	e		ε	epsilon	5
F	f	F	f		ζ	zeta	6
G	g	G	g		η	eta	7
H	h	H	h	Θ	θ, ϑ	theta	8
I	i	I	i		ι	iota	9
J	j	J	j		κ	kappa	
K	k	K	k	Λ	λ	lambda	
L	l	L	l		μ	mü	
M	m	M	m		ν	nü	
N	n	N	n		ξ	xi	
O	o	O	o		ο	omicron	
P	p	P	p	Π	π	pi	
Q	q	Q	q		ρ	rho	
R	r	R	r	Σ	σ, ς	sigma	
S	s	S	s		τ	tau	
T	t	T	t		υ	upsilon	
U	u	U	u	Φ	φ	phi	
V	v	V	v		χ	chi	
W	w	W	w	Ψ	ψ	psi	
X	x	X	x	Ω	ω	omega	
Y	y	Y	y				
Z	z	Z	z				
Ä	ä	Ä	ä	∀		für alle	
Ö	ö	Ö	ö	∃, ∃		es existiert	
Ü	ü	Ü	ü				
	ß		ß				

Vorlesungsmitschrift von Manuel Haim

http://www.mathematik.uni-marburg.de/~haim

Alphanumerische Codes
Textcodes
- ASCII = American Standard Code for Information Interchange
  7 Bits pro Zeichen → 128 Zeichen
- Unicode
  16 Bits → 65.536 Codepunkte

Grafikcodes
Kodierung von Bildpunkten
z.B. 4 Bits für 16 Graustufen
8 Bits für 256 Graustufen

Beispiel: Lauflängencodierung
Speicherung in 2 Bytes
1. Anzahl aufeinanderfolgender Bildpunkte mit gleicher Graustufe
2. Graustufe
⇒ Kompression bei vielen benachbarten gleichen Bildpunkten

CPI ≙ average number of clock cycles per instruction
bei RISC: CPI ≈ 1

Tafel

Abschätzung der Beschleunigung durch Fließbandverarbeitung:

Fließband mit r Stufen

Folge von m Instruktionen

$S_r = \frac{T_{seq}}{T_{pipe}}$, wobei $T_{seq} = m \cdot r$ Schritte

$T_{pipe} = r + \underbrace{m-1}_{m-1 \text{ restl. Instr.}}$

r: Anlaufphase bis Fertigstellung der ersten Instruktion

d.h. $S_r = \frac{m \cdot r}{\underbrace{r+m-1}_{\geq m}} \leq \frac{m \cdot r}{m} = r$

$\xrightarrow[m \to \infty]{} r$

IF - ID - OF - EX - WB

Fo48

Superskalar Architekturen: Parallelität verschiedener Einheiten

Fo50 (neu!)

Parallelrechner

Fo51

Adressraum \ Speicher	global	physikalisch verteilt
gemeinsam	SMP (Sym. multipr.) UMA-uniform memory access	DSM (distributed shared memory) NUMA-non-uniform memory access
verteilt		Multicomputer, DM (distr. mem.) (N)UCA (non-) uniform communication architecture

002.gif

Virtualisierung des Speichers

Fo 52

Tafel

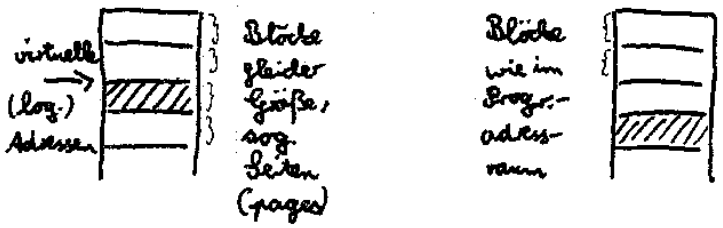
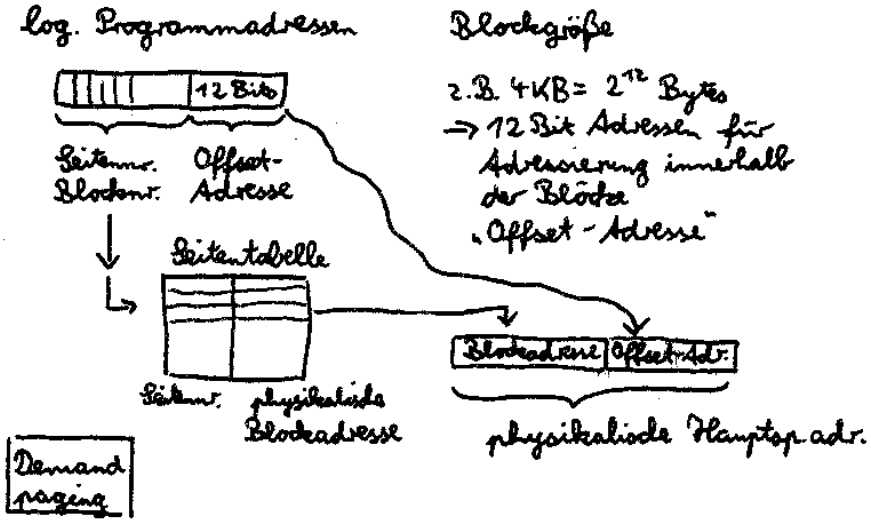


Abbildung von log. Programmadressen
auf physikalische Hauptspeicheradressen:



Lokalität

Fo 58

90/10 -Regel:
90% der Daten des Programms werden aus 10% des vom Programms insgesamt benötigten Speichers geladen

- Temporale Lokalität (zuletzt benutzte Daten)
- Räumliche Lokalität (benachbarte Speicherzelle)

27.10.2004

Parallel + (ODER)
Serie * (UND)

Schaltfunktion $f: \{0,1\}^n \rightarrow \{0,1\}$ beschreibt Gesamtschaltkreis

Mehrstellige Schaltfunktionen

Äquivalenz: Schaltfunktionen sind identisch

Gleichungen

Fo 71

Für beliebige Verbände muss der Nachweis von Gleichungen unter Verwendung der bekannten Gleichungen durch Gleichungsumformungen erfolgen. TAFEL

Bsp. $(x \underline{*} y) \underline{*} (x + y) \stackrel{!}{=} x * y$

$\stackrel{\text{Komm.}}{=} x * (y * (x + y))$

$\stackrel{\text{Komm.}}{=} x * (y * (y + x))$

$\stackrel{\text{Absorpt.}}{=} x * y$

alternative Umformung

$(x * y) \underline{*} (x \underline{+} y)$

$\stackrel{\text{Distr.}}{=} ((x * y) * x) + ((x * y) * y)$

$\stackrel{\text{Komm.}}{=} ((y * x) * x) + (x * (y * y))$

$\stackrel{\substack{\text{Assoz.} \\ \text{Idemp.}}}{=} (y * (x * x)) + (x * y)$

$\stackrel{\text{Idemp.}}{=} (y * x) + (x * y)$

$\stackrel{\text{Komm.}}{=} (x * y) + (x * y)$

$\stackrel{\text{Idemp.}}{=} x * y$

Dualitätsprinzip E 72

Eine induktiv aufgebaute Menge besteht aus TAFEL

- atomaren Elementen

und wird abgeschlossen unter gewissen Operationen.

Bsp. $\mathbb{N}$ 0 ist atomares Element von $\mathbb{N}$

Nachfolgefkt. ist Abschlussop.

d.h. für jedes $n \in \mathbb{N}$ ist auch $n+1 \in \mathbb{N}$ und kein anderes Element ist in $\mathbb{N}$.

S/P-Terme. atomare Elemente:
- Konstanten 0 und 1
- Variable x

Abschlussoperationen
- Parallelschaltung $t_1 + t_2$
- Serienschaltung $t_1 * t_2$

Beweisprinzipien für induktiv aufgebaute Mengen

① Vollständige Induktion (Spezialfall $\mathbb{N}$) (Spezialfall der strukturellen Induktion)

Der Beweis eines Satzes der Form

„ $E(n)$ gilt für $n \in \mathbb{N}$"

besteht aus zwei Schritten:

1. Induktionsanfang: Zeige: $E(0)$ ist wahr. Eigenschaft

2. Induktionsschluss:

Unter der Induktionsvoraussetzung, dass $E(n)$ für ein bel. $n \in \mathbb{N}$ gelte, zeigt man, dass auch $E(n+1)$ gilt.

Bsp. Zeige, dass für alle $n \in \mathbb{N}$ gilt:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Beweis durch vollst. Induktion:

Induktionsanfang: zz $\sum_{i=1}^{0} i \overset{!}{=} \frac{0(0+1)}{2}$! Behauptung

$\underbrace{\sum_{i=1}^{0} i}_{\text{leere Summe} = 0}$, $\frac{0(0+1)}{2} = 0$

Induktionsschluss:

Induktionsvoraussetzung: Sei $n \in \mathbb{N}$ mit $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Zeige: $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$

Es gilt: $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) \overset{\text{IVor.}}{=} \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$

$$= \frac{(n+1)(n+2)}{2}$$

$E(0)$ und $E(n) \Rightarrow E(n+1)$ für bel. n

② Strukturelle Induktion

Geg. induktiv aufgebaute Menge M.

zz Eigenschaft $E(m)$ gilt f.a. $m \in M$.

Induktionsanfang: Zeige, dass $E(m)$ für alle atomaren Elemente $m \in M$ gilt.

Induktionsschluss: Zeige unter der Induktionsvoraussetzung, dass $E(m_i)$ für Elemente $m_i \in M$ gelte, zeigt man, dass auch $E(f(m_1, \dots, m_{i_f}))$ für jede Operation f gilt.

005.gif

Bsp. M = Menge der S/P-Terme

$E(t): t = (t^d)^d$

Induktionsanfang:

- Konstanten: $(0^d)^d = (1)^d = 0$

  $(1^d)^d = 0^d = 1$

- Variablen: $(x^d)^d = x^d = x$

Induktionsschluss:

Induktionsvoraussetzung:

Seien u, v S/P-Terme mit: $(u^d)^d = u$, $(v^d)^d = v$

1. Zeige: $((u+v)^d)^d = u + v$

$((u+v)^d)^d = (u^d * v^d)^d = (u^d)^d + (v^d)^d$

$\overset{IV}{=} u + v$

2. Zeige $((u*v)^d)^d = u * v$

analog ✓

02.11.2004

TAFEL
zu Fo72

$\underline{2} := \{0,1\}$

Schaltfunktion $f: \underline{2}^n \to \underline{2}$

$n = 1$

x	0	1	x	?
0	0	1	0	1
1	0	1	1	0

monotone Fkt.en

nicht jede Schaltfunktion ist durch S/P-Schaltkreis realisierbar

nicht-monotone Fkt.

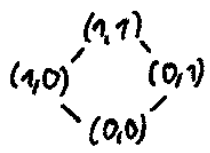
$0 \leq 1$

$(x_1, \dots, x_n) \leq (y_1, \dots, y_n)$ gdw. für alle $1 \leq i \leq n$ gilt: $x_i \leq y_i$

gdw = genau dann wenn

$(0,0) \leq (1,0) \leq (1,1)$

$(0,0) \leq (0,1) \leq (1,1)$



Satz: Durch S/P-Terme realisierbare Schaltfkt.en sind monoton

„Geschlossene Schaltkreise können durch das Schließen von Elementarschaltern nicht geöffnet werden."

Notation: Schreibe $f_t(x_1, \dots, x_n) = t$ für die Schaltfkt. zu Term t über Variable $x_1, \dots, x_n$.

Beweis: strukturelle Induktion über Aufbau der S/P-Terme

Induktionsanfang (atomare S/P-Terme)

a) Konstanten 0 und 1

$f_0(x_1, \dots, x_n) = 0$ für bel. $(x_1, \dots, x_n) \in \underline{2}^n$

$(x_1, \dots, x_n) \leq (y_1, \dots, y_n)$

$\Rightarrow f_0(x_1, \dots, x_n) = 0 = f_0(y_1, \dots, y_n)$ ✓

f_1 analog

b) Variable x_i, $f_{x_i}(x_1, \dots, x_n) = x_i$

$(x_1, \dots, x_n) \leq (y_1, \dots, y_n) \Rightarrow$ f.a. j: $x_j \leq y_j$

$\Rightarrow f_{x_i}(x_1, \dots, x_n) = x_i \leq y_i = f_{x_i}(y_1, \dots, y_n)$ ✓

Induktionsschluss

Induktionsvoraussetzung: Seien u und v S/P-Terme über $x_1, \dots, x_n$ mit monotonen Schaltfunktionen f_u und f_v.

Seien $(a_1, \dots, a_n), (b_1, \dots, b_n) \in \underline{2}^n$ mit $(a_1, \dots, a_n) \leq (b_1, \dots, b_n)$

Laut Induktionsvoraussetzung folgt:

$f_u(a_1, \dots, a_n) \leq f_u(b_1, \dots, b_n)$

$f_v(a_1, \dots, a_n) \leq f_v(b_1, \dots, b_n)$

Aus den Operationstafeln von + und * erkennt man, dass

$(c_1, c_2) \leq (d_1, d_2)$ ↷ $c_1 \leq d_1$, $c_2 \leq d_2$ ↷ $c_1 + c_2 \leq d_1 + d_2$, $c_1 * c_2 \leq d_1 * d_2$

Damit folgt:

$$f_{u+v}(a_1, \dots, a_n) = \underbrace{f_u(a_1, \dots, a_n)}_{\leq f_u(b_1, \dots, b_n)} + \underbrace{f_v(a_1, \dots, a_n)}_{\leq f_v(b_1, \dots, b_n)}$$

$f_{u+v}(a_1, \dots, a_n) \leq f_u(b_1, \dots, b_n) + f_v(b_1, \dots, b_n)$ ✓

(≤: Monotonie von +)

analog: $u * v$

TAFEL in FoBO

zur Schaltalgebra isomorphe Mengenalgebra

$(\underline{2}, +, *, ', 0, 1)$

Wähle $M = \{a\}$

Zugeh. Mengenalgebra:

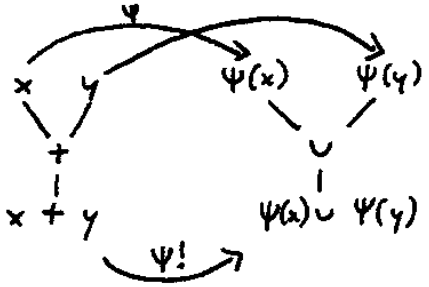
$(\{\underbrace{\emptyset, \{a\}}_{= P(\{a\})}\}, \cup, \cap, \overline{\ }, \emptyset, \{a\})$

$$\Psi: \begin{cases} \underline{2} \to \{\emptyset, \{a\}\} \\ 0 \mapsto \emptyset \\ 1 \mapsto \{a\} \end{cases}$$

zz: $\Psi(x+y) = \Psi(x) \cup \Psi(y)$

$\Psi(x*y) = \Psi(x) \cap \Psi(y)$

$\Psi(x') = \overline{\Psi(x)} = \{a\} \setminus \Psi(x)$



+	0	1
0	0	1
1	1	1

∪	∅	{a}
∅	∅	{a}
{a}	{a}	{a}

Eins-zu-Eins-Korrespondenz zwischen Werten in $\underline{2}$ und $P(\{a\})$ wird durch die Operationen erhalten.

x	x'
0	1
1	0

x	{a}\x
∅	{a}
{a}	∅

008.gif

$(\{f \mid f: \underline{2}^n \to \underline{2}\}, +, *, ', \underline{0}, \underline{1})$

↖ ↑ konstante Elemente

TAFEL zu Fo 85

Idempotenz $xx = x$

$x+x = x$

Komplementregel $xx' = 0$

$x+x' = 1$

→ Fo 89

03.11.2004

TAFEL zu Fo 88

Schaltalgebra - Kernfragen

1. Äquivalenz von Schaltkreisen / Termen?
   Wertetabellen
   Gleichungstransformationen
2. Gibt es eine Termdarstellung zu jeder Schaltfunktion?
   → Disjunktive Normalform
3. Welche Grundoperationen sind notwendig?

Bsp.

Index	x	y	z	Majorität		
0	0	0	0	0		
1	0	0	1	0		Minterme
2	0	1	0	0		
③	0	1	1	1	←	$x' y z$
4	1	0	0	0		
⑤	1	0	1	1	←	$x y' z$
⑥	1	1	0	1	←	$x y z'$
⑦	1	1	1	1	←	$x y z$

0 einschlägige Indices

Die Majoritätsfkt. kann durch die Mintermsumme

$x'yz + xy'z + xyz' + xyz$

dargestellt werden. Diese nennt man Disjunktive Normalform.

Beweis des DNF-Satzes

zz (i) Existenz der DNF

(ii) Eindeutigkeit der DNF

ad (i): Zeige f und $\sum_{i \in I} m_i$ liefern für jedes Argumenttupel $(j_1, \dots, j_n) \in \underline{2}^n$ denselben Wert.

Sei $j = (j_1, \dots, j_n)_2 = \sum_{i=1}^{n} j_i \cdot 2^{n-i}$

1. Fall: $f(j_1, \dots, j_n) = 1 \rightsquigarrow j \in I$

$\rightsquigarrow \sum_{i \in I} m_i(j_1, \dots, j_n) = 1$, da $m_j(j_1, \dots, j_n) = 1$

2. Fall: $f(j_1, \dots, j_n) = 0 \rightsquigarrow j \notin I$

$\rightsquigarrow \sum_{i \in I} m_i(j_1, \dots, j_n) = 0$, da für alle $i \neq j$ gilt $m_i(j_1, \dots, j_n) = 0$

ad (ii): Annahme, es gibt zwei Darstellungen von f als Summe von Mintermen

$$f = \sum_{i \in I} m_i = \sum_{j \in J} m_j$$

mit $I \neq J$ $\quad I, J \subseteq \{0, \dots, 2^{n-1}\}$

Dann existiert ein k mit o.B.d.A.

$k \in I$ und $k \notin J$. Sei $k = (k_1 \dots k_n)_2$.

$k \in I \rightsquigarrow \sum_{i \in I} m_i(k_1, \dots, k_n) = 1 = f(k_1, \dots, k_n)$

$k \notin J \rightsquigarrow \sum_{j \in J} m_j(k_1, \dots, k_n) = 0$ // ↯ Widerspruch

$\Rightarrow$ Die Annahme ist falsch.

zu E.o.B0 $\quad x'y'z'$

0 0 0	↦ 1	0
0 0 1	↦ 0	1
⋮	⋮	⋮
1 1 1	↦ 0	1

$(x'y'z')' = (x')' + (y')' + (z')'$

$= x + y + z$

(de Morgan)

010.gif

de Morgan: $(x + y)' = x' y'$

$$\boxed{x + y = (x' y')'}$$

$$x\ y = (x' + y')'$$

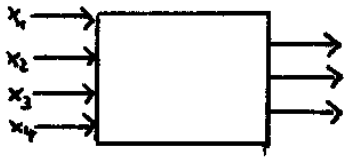
x NAND $y = (x\ y)'$

x NAND $x = (x\ x)' = x'$

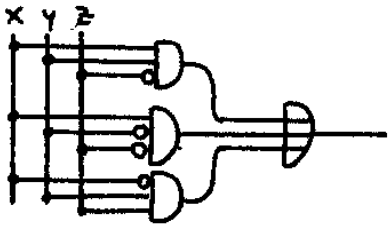
$NAND_3(x, y, z) = (x\ y\ z)'$

zu Fo97

Schaltfkt.



DNF $xyz' + xy'z' + x'yz$



zwei-stufige Schaltung

->112

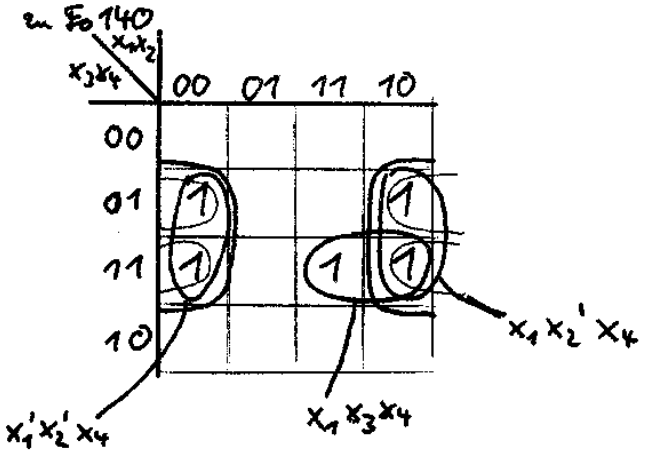
09.11.2004

zu Fo136

Kosten (Minterm) = Stelligkeit − 1

Kosten (DNF) = Anzahl einschlägige Indices * (Kosten (Minterm) + 1) − 1

zu Fo140



⇒ 5 Resolutionsmöglichkeiten

011.gif

3 Resolventen überdecken bereits alle Einsen:

$x_1 x_3 x_4 + x_1 x_2' x_4 + x_1' x_2' x_4$

$x_2' x_4$

$\hat{=}$ Viererblock von Einsen

zu Fo 141

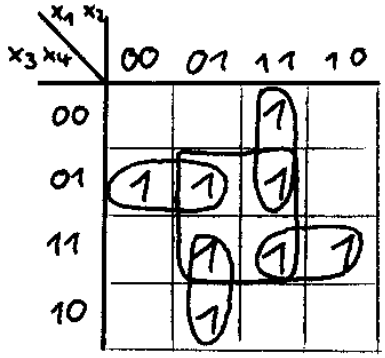
Rechteckige Einserblöcke im Karnaugh-Diagramm entsprechen verkürzten Monomen:

$2^r \times 2^s$ -Block $\hat{=}$ Monom mit $n-r-s$ Literalen

Viererblock: 2×2 -Block, 1×4 -Block, 4×1 -Block $\hat{=}$ Monom mit 2 Literalen

Achterblock: 2×4, 4×2 $\rightarrow$ Monom mit 1 Literal

Kostenminimale Überdeckung erhält man nicht immer durch maximale Blöcke



$\Rightarrow$ Minimalpolynom

$x_1' x_3' x_4$
$+ x_1 x_2 x_3'$
$+ x_1 x_3 x_4$
$+ x_1' x_2 x_3$

überdeckter Viererblock:

$x_2 x_4$

zu Fo 141

10.11.2004

$= \underbrace{(x_2' + x_1 x_3)}\, x_4$ Kosten 3

Dies ist keine Polynomdarstellung.

Das Minimalpolynom zu dieser Fkt. hat die Kosten 4.

012.gif

Primimplikanten von f: $x_2' x_4$, $x_1 x_3 x_4$

Implikanten:

- alle Minterme zu einschlägigen Indices, im Bsp. 5
- alle Zweierblöcke von Einsen, im Bsp. 5
- alle Viererblöcke, im Bsp. 1
- alle Achterblöcke, im Bsp. 0

zu Fo 145

<u>Satz</u>: Die Monome eines Minimalpolynoms sind Primimplikanten.

<u>Beweis:</u> Sei $f: \underline{2}^n \to \underline{2}$, $f \not\equiv \underline{0}$.

Sei $\sum_{i=1}^{k} a_i$ Minimalpolynom für f.

1.) Zeige: Alle a_i sind Implikanten von f.

Es gilt: $a_i(b_1, \dots, b_n) = 1 \rightsquigarrow f(b_1, \dots, b_n) = 1$

für alle $(b_1, \dots, b_n) \in \underline{2}^n$

Dies ist gleichbedeutend mit

$a_i(b_1, \dots, b_n) \leq f(b_1, \dots, b_n)$ für alle $(b_1, \dots, b_n) \in \underline{2}^n$ ✓

2.) Annahme: Es gäbe j mit: a_j ist kein Primimplikant von f.

Dann ex. ein Teilmonom a von a_j, das Implikant von f ist, d.h.

$a(b_1, \dots, b_n) \leq f(b_1, \dots, b_n)$ f.a. $(b_1, \dots, b_n) \in \underline{2}^n$.

Es gilt außerdem: a_j ist Implikant von a (Übungsaufgabe).

Weiter ist a kostengünstiger als a_j.

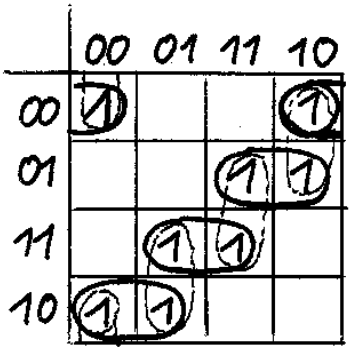
Die Summe $a_1 + \dots + a_{j-1} + a + a_{j+1} + \dots + a_k$

ist Polynom für f und kostengünstiger als $\sum_{i=1}^{k} a_i$ ↯ zur Kostenminimalität von $\sum_{i=1}^{k} a_i$

013.gif

Im Allgemeinen sind Minimalpolynome zu Schaltfkt. nicht eindeutig bestimmt.

Bsp. Schaltfkt. mit zwei Minimalpolynomen



zu Fo155

Bsp. zur Komplexität des Überdeckungsproblems

Betrachte zwei- und dreielementige Teilmengen der Menge $\{1, \ldots, 7\}$.

Es gilt: Es gibt $\binom{7}{2} = \frac{7 \cdot 6}{2} = 21$ zweielementige Teilmengen

$\binom{7}{3} = \frac{7 \cdot 6 \cdot 5}{1 \cdot 2 \cdot 3} = 35$ dreielementige Teilmengen

Dreielementige Teilmengen überdecken die in ihnen enthaltenen zweielementigen Teilmengen.

$\Rightarrow$ Überdeckungsmatrix hat Dimension $35 * 21$ ($= 735$ Einträge)

	{1,2}	{1,3}	{1,4}	{1,5}	...
{1,2,3}	1	1	0	0	...
{1,2,4}	1	0	1	0	...
{1,2,5}	1	0	0	1	...
⋮	⋮				

Diese Überdeckungsmatrix ist irredusibel.

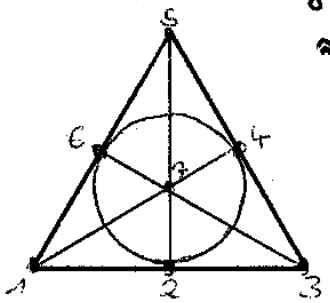
Bestimmung einer minimalen Überdeckung:

„brute force" - Algorithmus. Teste alle Teilmengen der Menge aller dreielementigen Teilmengen darauf, ob sie alle Zweiermengen überdecken:

d.h. es sind $2^{35} \approx 32$ Mrd. Auswahlen zu überprüfen.

Für eine gegebene Auswahl ist der Test, ob eine Überdeckung aller Objekte vorliegt, relativ einfach.

Lösung dieses Überdeckungsproblems:



≈ 7 Dreiermengen, die alle Zweiermengen überdecken

„endliche projektive Ebene der Ordnung 2"

→Fo157

zu Fo158

16.11.2004

Halb-addierer

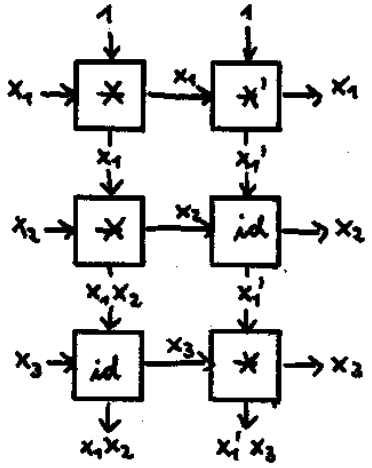
x	y	s	c „Überlauf"
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

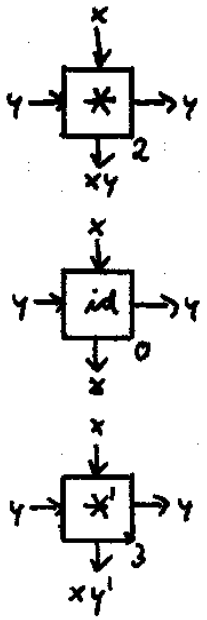
DNF für $s = x'y + xy'$

exklusives Oder ⊕
Antivalenz

zu Fo165 PLA-Realisierung einer Schaltfkt.

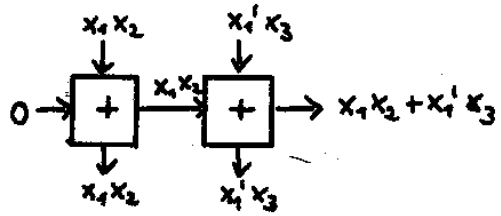
Bsp.: $f(x_1, x_2, x_3) = x_1x_2 + x_1'x_3$

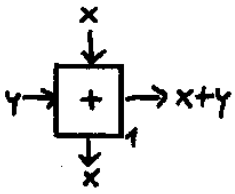




In den Spalten der UND-Ebene des PLAs werden die Monome der Schaltfkt. erzeugt.

015.gif





zu Fo 168

PLA-Matrix

$$\left(\begin{array}{cc} 2 & 3 \\ 2 & 0 \\ 0 & 2 \\ \hline 1 & 1 \end{array}\right)$$

zu Fo 167

Bsp. (analog Fo 170)

$u = y'z + xyz = (xy'z) + x'y'z + (xyz)$

$v = xz + xyz' = (xyz) + (xy'z) + xyz'$

alternative PLA-Matrix

$$\left(\begin{array}{cccc} 2 & 3 & 2 & 2 \\ 3 & 3 & 2 & 2 \\ 2 & 2 & 2 & 3 \\ \hline 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{array}\right)$$

Fo 172: dritte Spalte von links markiert (nicht von rechts), Inhalt der Adresse entsprechend 1010

016.gif

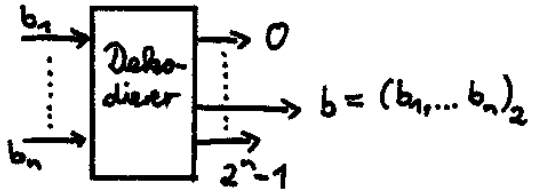
17.11.2004

Weitere Standardbausteine

Dekodierer und Kodierer

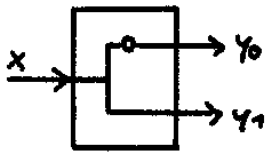
Dekodierer n Eingänge $\longrightarrow$ 2^n Ausgänge

$(b_1, \ldots, b_n) \mapsto (0, 0, \ldots, 0, \underset{\uparrow}{1}, 0, \ldots, 0)$
$(b_1 \ldots b_n)_2$



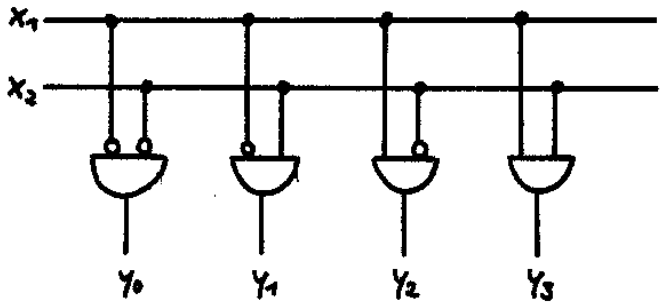
$n = 1$

x	y_0	y_1
0	1	0
1	0	1



$n = 2$

x_1	x_2	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

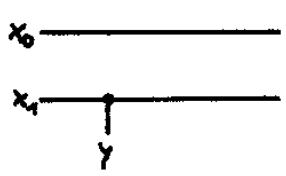


Kodierer 2^n Eingänge $\longrightarrow$ n Ausgänge

$(0, \ldots, 0, \underset{\uparrow}{1}, 0, \ldots, 0) \mapsto (b_1, \ldots, b_n)$
$b = (b_1 \ldots b_n)_2$

$n = 1$

x_1	x_0	y_0
0	1	0
1	0	1

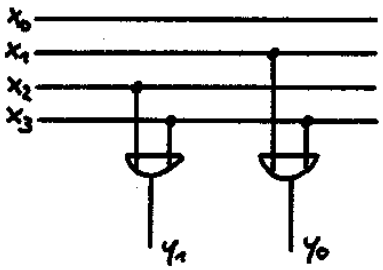


$n = 2$

x_3	x_2	x_1	x_0	y_1	y_0
1	0	0	0	1	1
0	1	0	0	1	0
0	0	1	0	0	1
0	0	0	1	0	0

$\rightsquigarrow$ $y_1 = x_3 + x_2$

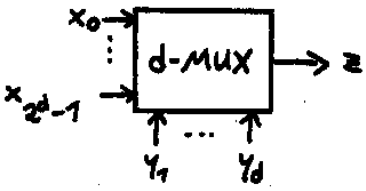
$y_0 = x_3 + x_1$

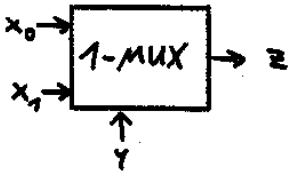


Multiplexer

2^d Dateneingänge
d Steuereingänge
} 1 Ausgang

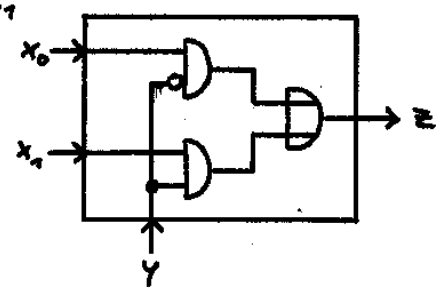
Die Steuereingänge bestimmen, welcher Dateneingang durchgeschaltet wird.



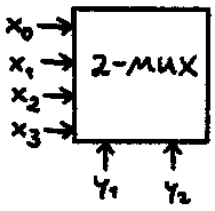


y	z
0	x_0
1	x_1

$\rightsquigarrow z = y' x_0 + y x_1$



018.gif



y_1	y_2	z
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

$\rightsquigarrow z = y_1' y_2' x_0 + y_1' y_2 x_1 + y_1 y_2' x_2 + y_1 y_2 x_3$

$\rightsquigarrow$ 2-stufige Schaltung mit

4 dreistelligen UND-Gattern und

1 vierstelligen ODER-Gatter

alternativ: 8 binäre UND-Gatter und

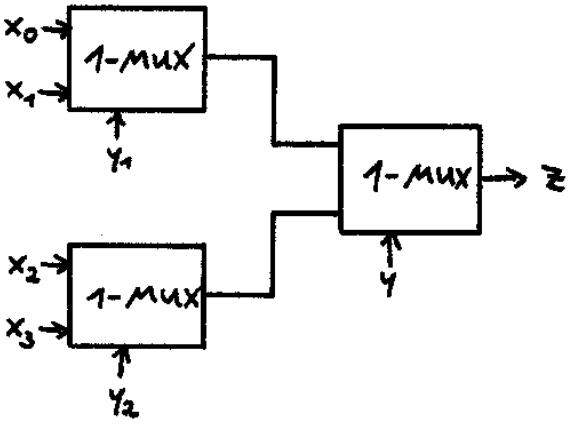
3 binäre ODER-Gatter

$\Rightarrow$ 11 binäre Gatter

Alternative Konstruktion von 2d-MUX aus verschiedenen d-MUX-Bausteinen.

zunächst: $d = 1$, d.h. Konstruktion von 2-MUX aus 1-MUX-Bausteinen

Ansatz: Aufteilung der Eingänge auf 2 1-MUX



9 binäre Gatter

019.gif

allgemein: $2d$-MUX hat

$2^{2d} = (2^d)^2$ Dateneingänge

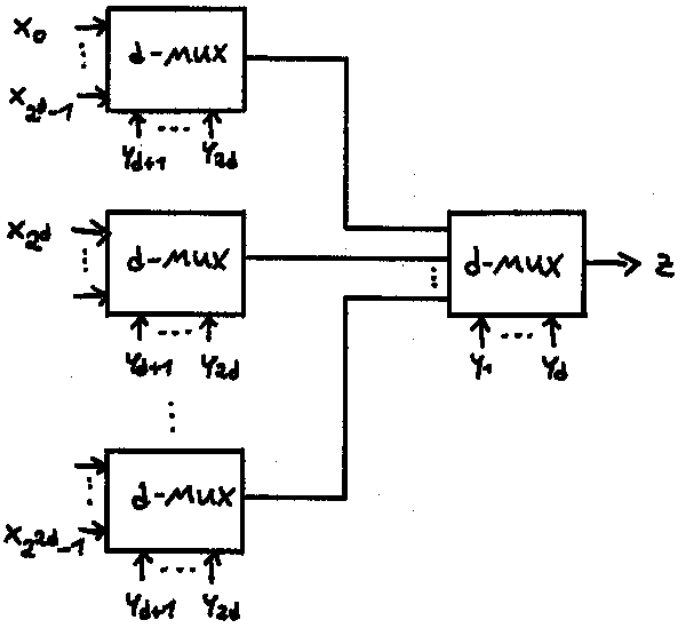
$2d$ Steuereingänge

d-MUXe haben

2^d Dateneingänge

d Steuereingänge

$\rightsquigarrow$ Benötige 2^d d-MUXe für Aufteilung der Dateneingänge



Vorselektion mit hinteren d Steuereingängen

$\rightsquigarrow$ insgesamt $2^d + 1$ d-MUX

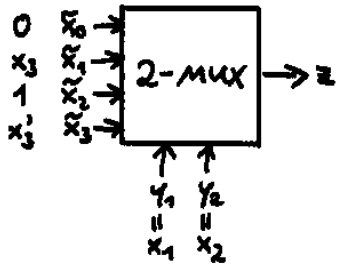
Auch Multiplexer können als Grundbausteine zur Realisierung beliebiger Boolescher Fkt.en verwendet werden.

Bsp.

y_1	y_2		
x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

x_1	x_2	f
0	0	0
0	1	x_3
1	0	1
1	1	x_3'

Realisierung von f mit 2-MUX



Idee: Wähle Teil der Argumente von f zur Steuerung des Multiplexers

Jede dreistellige Boolesche Fkt. kann in Abhängigkeit von x_1 und x_2 dargestellt werden durch die Abbildungen x_3, x_3', $\underline{0}$, $\underline{1}$

⇒ Zum Ausprobieren: siehe auch VL-Seite „Digital Simulator"

z.B. 74179

SR / $Q^{(n)}$	00	01	11	10
0			D	1
1	1		D	1

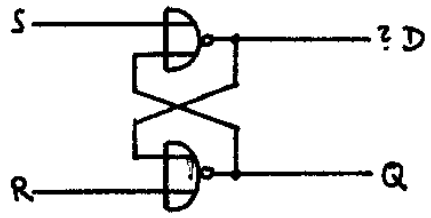
Minimalpolynom:

$$SR' + R'Q^{(n)}$$
$$= (R'(S + Q^{(n)}))''$$

021.gif

$= (R + (S + Q^{(n)})')'$

NOR-Darstellung

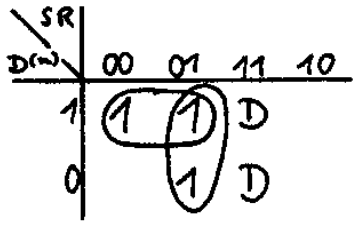


$D^{(n+1)} = (S + (R + D^{(n)})')'$

$= S'R + S'D^{(n)}$

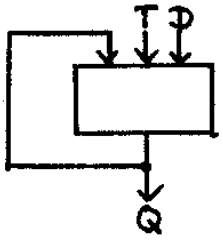
$\Rightarrow D^{(n+1)}$ ist dual zu $Q^{(n+1)}$

Für zulässige Eingaben gilt: $D^{(n+1)} = (Q^{(n+1)})'$



Speicherelement Early Latch — zu Fo 180 — 23.11.2004

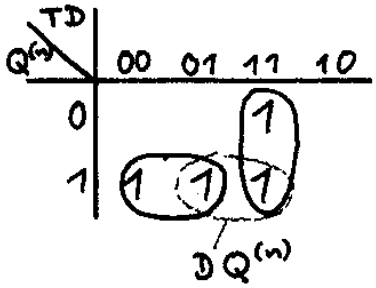
vereinfachtes Speicherelement mit nur einem Steuereingang D



$$Q^{(n+1)} = \begin{cases} Q^{(n)} & \text{falls } T = 0 \\ D & \text{falls } T = 1 \end{cases}$$

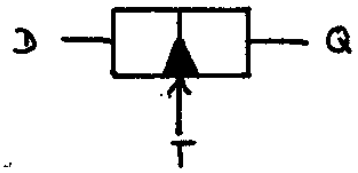
Solange T=1 ist, darf D sich nicht ändern.

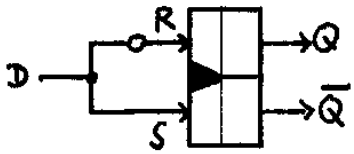
$Q^{(n+1)} = T' Q^{(n)} + T \cdot D$



Realisierung ohne Schaltungshazards

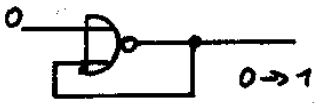
$Q^{(n+1)} = T' Q^{(n)} + TD + DQ^{(n)}$





Problem: Instabile Schaltungen

einfache Flimmerschaltung

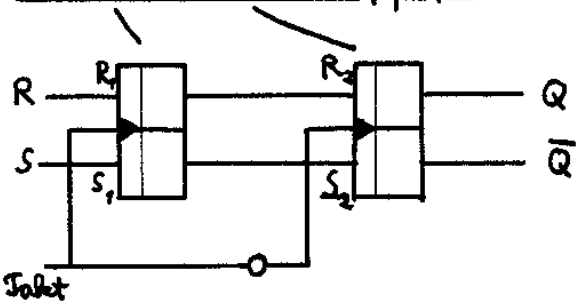


0→1→0→1→...

1→0→1→0→...

Auch bei Flipflops kann es zur Instabilität kommen, wenn die unzulässige Eingabe $S = R = 1$ mit $Q = \bar{Q} = 0$ gleichzeitig auf $S = R = 0$ zurückgesetzt wird.

Master-Slave-Flipflops



→ E 191

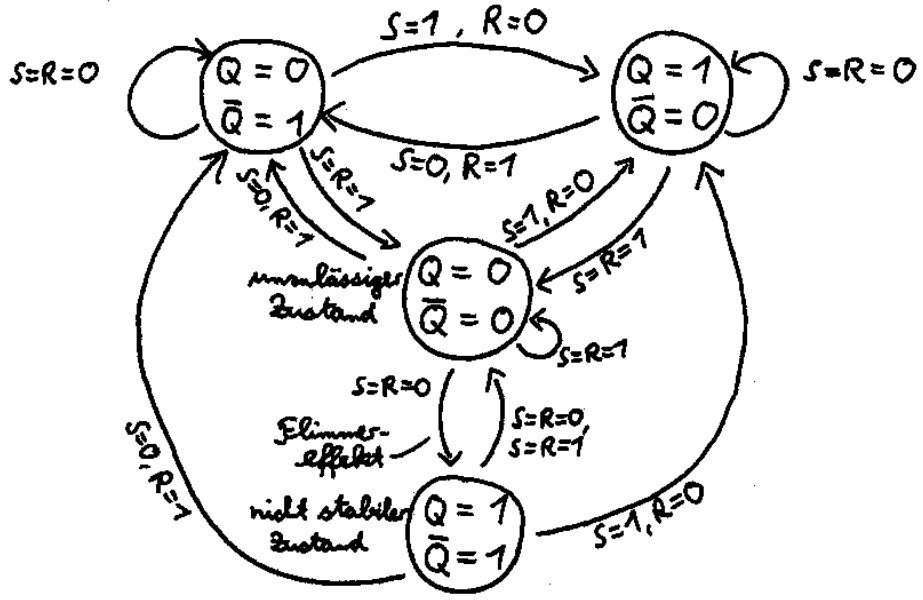
023.gif

24.11.2004

Beschreibung des Verhaltens von Schaltwerken mit endlichen Zustandsdiagrammen / Automaten (Finite State Machine, FSM)

Zustände $\hat{=}$ gespeicherte Werte

Bsp. Flipflop



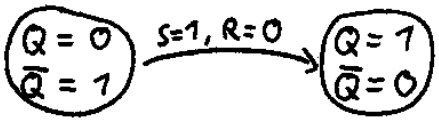
Flipflop als Schaltwerk

Eingaben $(S, R) \in \underline{2} \times \underline{2} = \Sigma$

Zustände $(Q, \bar{Q}) \in \underline{2} \times \underline{2} = Q$

Ausgaben $(Q, \bar{Q}) \in \underline{2} \times \underline{2} = \Gamma$

δ ist durch das obige Diagramm gegeben

z.B. 

$\hat{=}\ \delta\big(\underset{Q\ \bar{Q}}{(0,1)}, \underset{S\ R}{(1,0)}\big) = (1,0)$

klar: $\lambda\big((Q, \bar{Q}), (S,R)\big) = (Q, \bar{Q})$

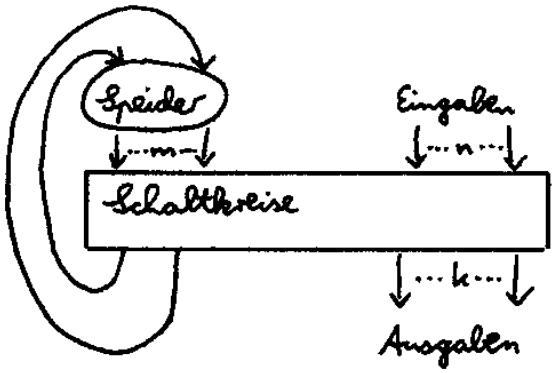
Ein endlicher (Mealy-) Automat ist ein Siebentupel

$\mathcal{A} = (Q, \Sigma, \Gamma, \delta, \lambda, q_0, F)$

mit endlicher Zustandsmenge Q,
endlichem Eingabealphabet Σ,
endlichem Ausgabealphabet Γ,
Transitionsfunktion $\delta: Q \times \Sigma \to Q$
Ausgabefunktion $\lambda: Q \times \Sigma \to \Gamma$
(falls $\lambda: Q \to \Gamma$ spricht man
von einem Moore-Automaten)
Anfangszustand $q_0 \in Q$
[Endzustandsmenge $F \subseteq Q$]

Für einen Schaltwerkautomaten gilt

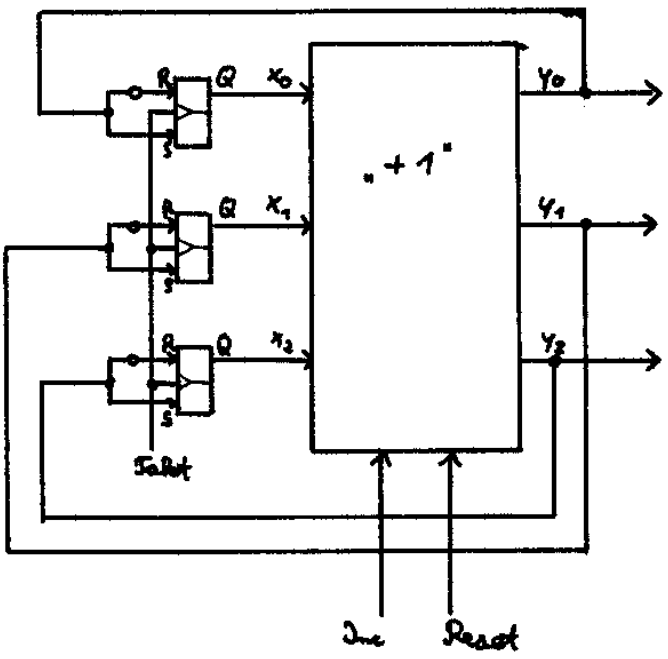
$Q = \underline{2}^m$ für ein $m \in \mathbb{N}$
$\Sigma = \underline{2}^n$ $n \in \mathbb{N}$
$\Gamma = \underline{2}^k$ $k \in \mathbb{N}$



025.gif

Bsp. 3 - Bit - Ringzähler

$000 \rightarrow 001 \rightarrow \ldots \rightarrow 111$



x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

$y_0 = x_0'$

$y_1 = x_1 \underline{\text{xor}} x_0$

$y_2 = x_2(x_1x_0)' + x_2' x_1 x_0$

$= x_2 \underline{\text{xor}} (x_1 x_0)$

Einarbeitung von Eingangen Inc und Reset

$y_0 = R \cdot 0 + R'(I \cdot x_0' + I' x_0)$

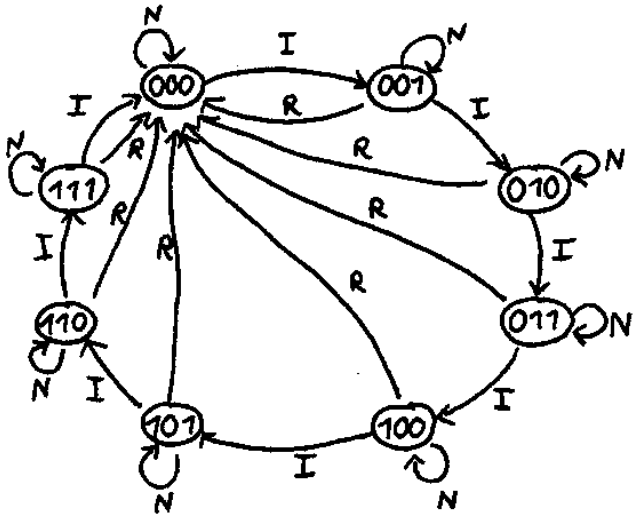
$= R'(I \underline{\text{xor}} x_0)$

$y_1 = R'(I(x_1 \underline{\text{xor}} x_0) + I' x_1)$

$y_2 = R'(I(x_2 \underline{\text{xor}} (x_1 x_0)) + I' x_2)$

026.gif

Zustandsdiagramm



N bedeutet: R=I=0

I steht für: R=0, I=1

R steht für: R=0, I∈{0,1}

zu Fo203 −3 als 4-Bit-Betrag-und-Vorzeichenzahl: 1011

8-Bit 1000 0011

3 mit 4 Bits 0011

8 Bits 0000 0011

Darstellung negativer Zahlen

Fragen: 1. Interpretation der Bitfolgen

2. Negieren von Zahlen „$x \to -x$"

3. Symmetrie des Zahlenbereichs

4. Zahlenverlängerung / Bereichserweiterung

5. Realisierung von Addition und Subtraktion

Betrag- und-Vorzeichen-Darstellung

1. $decode_{bv}(b_{n-1}, \ldots b_0) = \begin{cases} + \sum_{i=0}^{n-1} b_i 2^i & \text{falls } b_n = 0 \\ - \sum_{i=0}^{n-1} b_i 2^i & \text{falls } b_n = 1 \end{cases}$

$\uparrow$ Vorzeichen Betrag

$= (-1)^{b_n} \cdot \sum_{i=0}^{n-1} b_i \cdot 2^i$

2. Negieren durch Komplementieren des Vorzeichenbits
3. Zahlenbereich: $-(2^n-1) \ldots -0 \quad 0 \ldots (2^n-1)$
   symmetrischer Bereich, aber
   zwei Darstellungen für Null
4. Zahlenverlängerung durch Auffüllen mit Nullen
   nach dem Vorzeichenbit
5. schwierig!

028.gif

30. 11. 2004

Einerkomplementdarstellung

2. Ex 286

(a) Interpretation der Bitfolgen

$b_n\ b_{n-1} \dots b_0$

$b_n = 0$: $+\sum_{i=0}^{n-1} b_i 2^i$

$b_n = 1$: $-\sum_{i=0}^{n-1} b_i' \cdot 2^i \qquad (b_i' = 1 - b_i)$

$$= -\sum_{i=0}^{n-1} (1-b_i) \cdot 2^i$$

$$= -\left[\sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} b_i \cdot 2^i\right]$$

$$= -(2^n - 1) + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

$$\Rightarrow \text{decode}_1(b_n \dots b_0) = -b_n \cdot (2^n - 1) + \sum_{i=0}^{n-1} b_i 2^i$$

(b) Negieren: bitweises Komplementieren

(c) Zahlenbereich: $[-(2^n-1) \dots -0 \quad +0 \dots +(2^n-1)]$

-0: $11\dots1$ $\qquad$ $+0$: $00\dots0$

(d) Zahlenverlängerung / Bereichserweiterung:

Auffüllen mit dem Vorzeichenbit

negative $(n+1)$-Bit-Zahl: $1\ b_{n-1} \dots b_0$

$\downarrow \text{decode}_1$

$$-(2^n - 1) + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

$(m+1)$-Bit-Zahl mit $m > n$: $\underbrace{11 \dots 1}_{m-n}\ 1\ b_{n-1} \dots b_0$

$$-(2^m - 1) + \sum_{i=n}^{m-1} 1 \cdot 2^i + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

$$= -(2^m - 1) + 2^n \cdot \sum_{i=0}^{m-n-1} 2^i + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

$$= -(2^m - 1) + 2^n (2^{m-n} - 1) + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

029.gif

$$= -(2^n - 1) + 2^n - 2^n + \sum_{i=0}^{n-1} b_i 2^i$$

$$= -(2^n - 1) + \sum_{i=0}^{n-1} b_i 2^i$$

(c) Addition von Einerkomplementzahlen

$$(a_n\, a_{n-1} \ldots a_0) \quad + \quad (b_n\, b_{n-1} \ldots b_0) \;?$$

$$\downarrow \text{decode}_1 \qquad\qquad \downarrow \text{decode}_1$$

$$-a_n(2^n-1) + \sum_{i=0}^{n-1} a_i \cdot 2^i \quad + \quad (-b_n)(2^n-1) + \sum_{i=0}^{n-1} b_i 2^i$$

$$= -(a_n + b_n)(2^n-1) + \underbrace{\sum_{i=0}^{n-1} (a_i + b_i) \cdot 2^i}$$

Standardaddition

Summenziffern $s_i \in \{0,1\}$

Übertragsziffern $carry_i \in \{0,1\}$

$$= -(a_n + b_n)(2^n-1) + carry_{n-1} \cdot 2^n + \sum_{i=0}^{n-1} s_i \cdot 2^i$$

$$2^n = (2^n - 1) + 1$$

$$= -\underbrace{(a_n + b_n - carry_{n-1})}(2^n-1) + carry_{n-1} + \sum_{i=0}^{n-1} s_i \cdot 2^i$$

Vorzeichenbit für Summendarstellung

$carry_{n-1}$: end-around carry

Ein Überlauf erfolgt, wenn $a_n + b_n - carry_{n-1} \notin \{0,1\}$

d.h. wenn $a_n = b_n = 0$ und $carry_{n-1} = 1$ oder

$a_n = b_n = 1$ und $carry_{n-1} = 0$

Bsp.: $\begin{array}{r} -18 \\ +25 \\ \hline +7 \end{array}$ $\quad n = 7$

$+18 = (0\,0\,0\,1\;0\,0\,1\,0)_2$

$-18 = \text{decode}_1(1\,1\,1\,0\;1\,1\,0\,1)$

$+25 = (0\,0\,0\,1\;1\,0\,0\,1)_2$

schriftliche Addition

```
  1 1 1 0 1 1 0 1
  0 0 0 1 1 0 0 1
(1) 1 1 1 0 0 1
  0 0 0 0 0 1 1 0
              → 1   end-around carry
  ---------------
  0 0 0 0 0 1 1 1
```

Ausgangsübertrag c_i^{out} = Eingangsübertrag c_{i+1}^{in} zu Fo 207

zu ÜB Aufgabe 37: mit möglichst wenig Gattern gestalten!

zu Fo 217

Festkommazahlen

$X_{n-1} \dots X_0 , Y_1 \dots Y_m$

repräsentiert $\sum_{i=0}^{n-1} X_i \cdot 2^i + \sum_{j=1}^{m} Y_j \cdot 2^{-j}$

Insgesamt Darstellung von 2^{n+m} Zahlen

aus dem Bereich $[0 \dots (2^n - 2^{-m})]$

kleinste Zahl 0 $\quad 00\dots0,0\dots0$

größte Zahl $\quad \underbrace{1\dots1}_{2^n-1},\underbrace{1\dots1}_{2^{-m}\sum_{j=1}^{m} 2^{m-j}}$

$\underbrace{\qquad\qquad}_{=2^m-1}$

$= 2^n - 2^{-m}$

gleichmäßige Einteilung des Zahlenbereichs

in Intervalle der Größe 2^{-m}.

Addition / Subtraktion

Angleichung der Position des Kommas, evtl. Genauigkeitsverlust

Bsp. 16-Bit-Zahlen

1100 1001 0000,1010

1000 0100,0001 1101

zu Eo221

01.12.2004

Addition / Subtraktion von Gleitkommazahlen:

1. Exponenten anpassen
2. Mantissen addieren
3. Postnormalisieren

Bsp. Sei $a = 1,101 \cdot 2^3 = 13_{10}$

$b = 1,011 \cdot 2^1 = 2,75_{10}$

zu 1) Exponentenanpassung bei b

$0,0101 \cdot 2^3$

1 ↵ Rundung!

zu 2) Addition der Mantissen:

$$\begin{array}{r} 1,101 \\ 0,011 \\ \hline 10,000 \end{array} \cdot 2^3$$

zu 3) Postnormalisierung: $1,000 \cdot 2^4 = 16_{10}$

Multiplikation / Division

$$(-1)^{V_1} \cdot m_1 \cdot 2^{e_1} \cdot (-1)^{V_2} \cdot m_2 \cdot 2^{e_2}$$

$$= (-1)^{V_1 + V_2} \cdot m_1 \cdot m_2 \cdot 2^{e_1 + e_2}$$

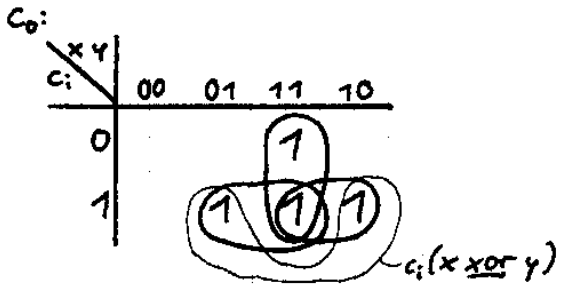
1.) Mantissen multiplizieren / dividieren

2.) Exponenten addieren / subtrahieren

3.) Postnormalisierung

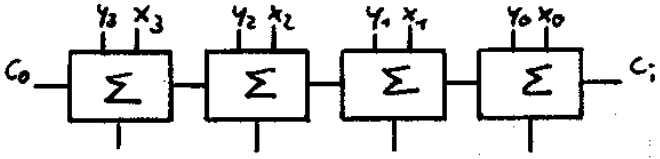
Volladdierer

x	y	c_i	c_o	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1



$$c_o = xy + c_i y + c_i x$$
$$= xy + c_i(x+y)$$
$$= xy + c_i(x \underline{\text{xor}} y)$$

zu Fo 227



$$c_o = x_3 y_3 + (x_3 + y_3) \cdot [x_2 y_2 + (x_2 + y_2) \cdot [x_1 y_1 + (x_1 + y_1) \cdot [x_0 y_0 + (x_0 + y_0) \cdot c_i]]]$$

Addierwerke

- Paralleladdierwerke (ripple-carry-adder)

  Berechnung in einem Takt (parallel),

  aber hohe Signalschaltzeiten wegen Carry-Propagation

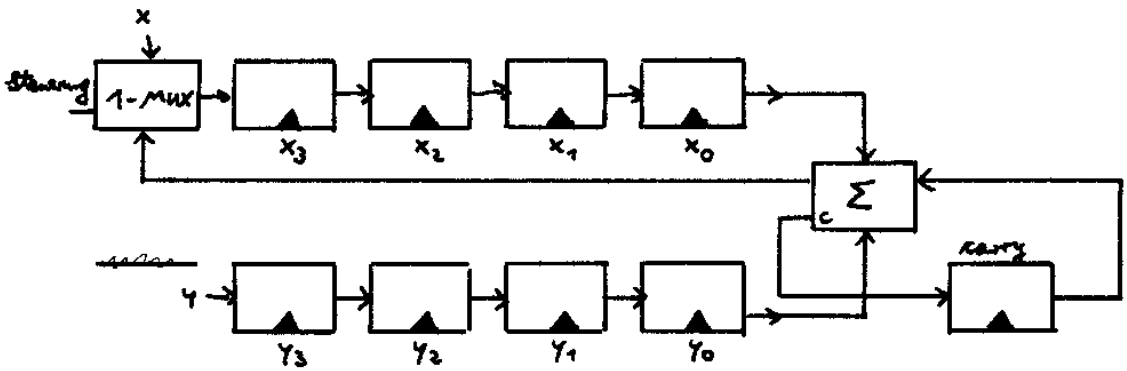
  Halbaddierer: zwei-stufig

  Volladdierer: vier-stufig

  ⇒ hoher Hardware-Aufwand, schnelle Schaltung

033.gif

- Serieller Addierer



wenig Hardware: nur 1 Volladdierer

n Takte, bis Ergebnis im Akkumulator vorliegt

- von Neumann Addierer (Fo 229)

getakteter carry-save-Addierer

Mischform aus Parallel- und Serienaddierer

n Halbaddierer, max (n+1) Takte, bis Ergebnis vorliegt

Bsp.	carry			Status
	0	a	1 1 1 1	1
		b	0 1 0 0	
	0	a	1 0 1 1	1
		b	1 0 0 0	
	(1)	a	(0 0 1 1)	0
		b	0 0 0 0	

⇒ Berechnung benötigte 2 Takte

Ergebnis: 1 0 0 1 1

Berechnung mit 4 Takten

carry			Status
0	a	1 0 1 1	1
	b	1 1 0 1	
1	a	0 1 1 0	1
	b	0 0 1 0	

carry			Status
1	a	0100	1
	b	0100	
1	a	0000	1
	b	1000	
1	a	1000	0
	b	0000	

Satz: Das von-Neumann-Addierwerk addiert zwei n-Bit-Summanden in durchschnittlich $\log_2 n$ Schritten.

Beweisskizze: Betrachte zwei zufällig ausgewählte Summanden.

Erwartung: Jeder Summand hat
$\frac{n}{2}$ Nullen und
$\frac{n}{2}$ Einsen, die
unabhängig voneinander verteilt sind.

$\Rightarrow$ Jede Kombination

a_i	b_i	S_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

kommt gleich häufig vor, also jeweils $\frac{n}{4}$ mal.

nächstes a-Register — nächstes b-Register

vor 1. Schritt: $\frac{n}{2}$ Einsen im b-Reg., $\frac{n}{2}$ Einsen im a-Reg.

vor 2. Schritt: $\frac{n}{4}$ Einsen im b-Reg., $\frac{n}{2}$ Einsen im a-Reg.

035.gif

Zufälligkeits- und Unabhängigkeitsannahme:

Die Hälfte der $\frac{n}{4}$ Einsen trifft auf 1 im a-Reg.

$\Rightarrow$ vor 3. Schritt: $\frac{n}{8}$ Einsen im b-Reg., $\frac{n}{2}$ Einsen im a-Reg.

$\vdots$

$\Rightarrow$ vor i-tem Schritt: $\frac{n}{2^i}$ Einsen im b-Reg.

Sobald $\frac{n}{2^i} < 1$ ist b-Reg $\equiv 0$, d.h. nach

$i > \log_2 n$ Schritten ist die Berechnung beendet.

07.12.2004

zu Fo 231

$$(2^n - 1)(2^n - 1) = \underline{\underline{2^{2n}}} - 2 \cdot 2^n + 1$$

zu Fo 239

Bsp.	Iteration	Multiplikand	Akkumulator / Multiplikator
		0100	0000 ¦ 0110
	1		Shift 0000 0 ¦ 011
	2		Add 0100
			Shift 001000 ¦ 01
	3		Add 0110
			Shift 00110000 ¦ 0
	4		Shift 00011000

zu Fo 242: altes Skript: letzten Schritt streichen (ist nämlich falsch)

zu Fo 241

Booth für Zweierkomplementzahlen:

Multiplikator $a = decode_2(a_n \dots a_0) =$

$$-a_n 2^n + \sum_{i=0}^{n-1} a_i 2^i$$

Multiplikand b

Ergänzung der Aktionstabelle um arithm. Operationen

a_i a_{i-1}	arithm. Op.	
0 0	—	} Addiere $(a_{i-1} - a_i) * b$
0 1	Addiere b	
1 0	Subtrahiere b	
1 1	—	

Rechtsshift des Zwischenproduktes

$\hat{=}$ Multiplikation mit 2 pro Iteration

$\Rightarrow$ Booth berechnet:

$$\sum_{i=0}^{n} (a_{i-1} - a_i) \cdot b \cdot 2^i$$

$$= b \left(\sum_{i=1}^{n} a_{i-1} 2^i - \sum_{i=0}^{n} a_i 2^i \right)$$

weil $a_{i-1} := 0$

$$= b \left(\sum_{i=0}^{n-1} a_i \cdot 2^{i+1} - \sum_{i=0}^{n} a_i 2^i \right)$$

$$= b \cdot \left(2 \cdot \sum_{i=0}^{n-1} a_i 2^i - \sum_{i=0}^{n-1} a_i 2^i - a_n 2^n \right)$$

$$= b \cdot \left(-a_n 2^n + \sum_{i=0}^{n-1} a_i 2^i \right)$$

$$= b \cdot a$$

Division — ganzzahlige Division

Dividend : Divisor = Quotient und Rest

d. h. Dividend = Quotient * Divisor + Rest

mit Rest < Divisor

Bsp. 1001010 : 1000 = 1001 ← Quotient

-1000

001010

- 1000

10 ← Rest

in Fo 245

formal nach Schema:

		Quotient	Iteration
Akku	01001010	0000	
Puffer	-10000000	←	
	1 10010 10		1
carry	+10000000		
	1 01001010	0000	
	⇒01000000		
	0 00001010		2
	⇒00100000	0001	
	1 11101010		3
	+00100000	0010	
	1 00001010		
	⇒00010000		
	1 11110010		4
	+00010000	0100	
	0 00010 10		
	⇒00001000		

0 00000010	1001	5
00000100		

Division und Restbildung bei negativen Zahlen

	Quotient	Rest
	-5 div 3	-5 mod 3
Pascal/Ada	-1	1
FORTRAN/Java	-1	-2
C	undef.	undef.
Modula-3/ Haskell	-2	1

Es sollte gelten

$$x = (x \text{ div } y) * y + (x \text{ mod } y)$$

$$-5 = (-5 \text{ div } 3) * 3 + (-5 \text{ mod } 3)$$

$$-1 * 3 + (-2)$$

$$-2 * 3 + 1$$

$$-1 * 3 + 1 \quad ?$$

sinnvollste Def.: $x \text{ div } y = \lfloor \frac{x}{y} \rfloor$ ← nächstkleinere ganze Zahl

$$x \text{ mod } y = x - \lfloor \frac{x}{y} \rfloor * y$$

Vorteile:
- x mod y ist immer positiv
- x div 2^k realisierbar mit arithmetischem k-Bit Rechtsshift
- x mod $2^k \hat{=}$ letzte k Bits von x

zu Fo 258 08. 12. 2004

„X <-> Y" Vertauschen von X und Y

SLL	logischer links-Shift
SLR	logischer rechts-Shift
SAL	arithmetischer links-Shift
SAR	arithmetischer rechts-Shift

zu Fo 14.12.2004

Bsp. Addiere Inhalt von Register R1
zu 16-Bit ab Adresse 5h

1. Befehl Speichermodus wartend

Phase 1:

Phase 2: z := 5

Phase 3: z -> MAR

2. Befehl Speichermodus lesend, Format: 2 Bytes

Phase 1: MDR -> y
R1 -> x

Phase 2: z := x + y (, cc)

Phase 3: z -> MDR

3. Befehl Speichermodus schreibend, Format: 2 Bytes

⋮

zu Fo 278

Interpreter-Schleife

while true do

L Lade Programmzähler und entsprechenden Befehl (Opcode)

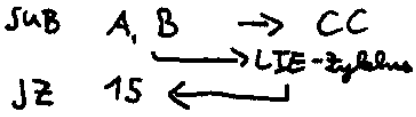
I Inkrementiere Programmzähler

E Führe geladenen Befehl aus

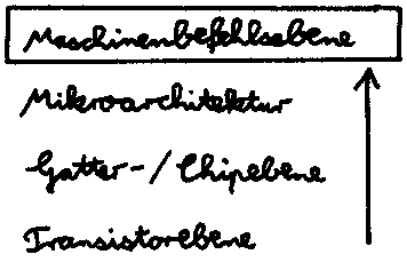
|

Zyklus

040.gif



15.12.2004



Befehlssatz eines Rechners

Aufbau eines Befehlswortes

FORMAT OPCODE ADRESSE 1 ... ADRESSE n

FORMAT bestimmt Befehlslänge
Adresskodierung
(Zahl/Art der Parameter)

OPCODE - auszuführende Operation
- Stelligkeit

ADRESSEN — Operandenadressen
- Quell- und Zieladressen
- ggf. Konstanten
- ggf. implizite Operanden
— Befehlsadressen bei Kontrollbefehlen

Bsp. Wortlänge 32 Bits
Registerzahl 32 → 5 Bits für Registeradresse
Speichergröße 64 KByte → 16 Bits für Speicheradresse

ALU-Befehl mit 3 Registeroperanden / 1 Reg., 1 Speicherop.

FORMAT	5 Bits	5	
OPCODE	6 Bits	6	
REG.-OPER.	15 Bits	5	Reg. Oper.
	26 Bits	16	Speicher-Op.
		32	

Darstellungsformen für Befehlsworte:

- Bitfolge
- Hexadezimal

} Objektcode

- mnemonische Darstellung „Assembler"
  (charakteristische Kürzel für jeden Befehl)

04.01.2005

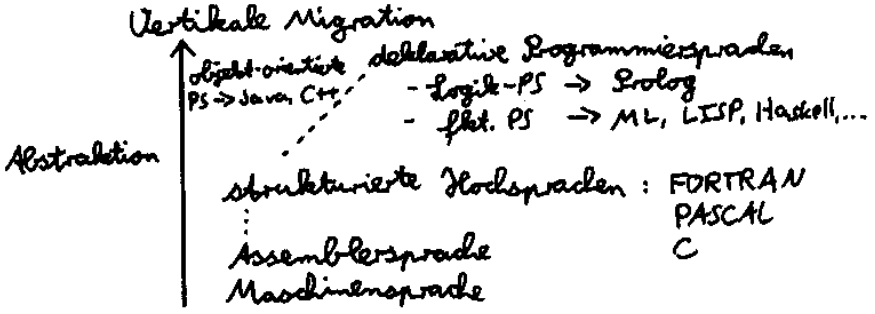
2. Leistungskontrolle morgen

9:15 - 9:45
Vorlesung 10:00 - 11:00

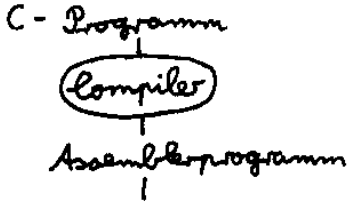
} HG 5 (wie gewohnt)

Stoffumfang: Kap. 4 - 6

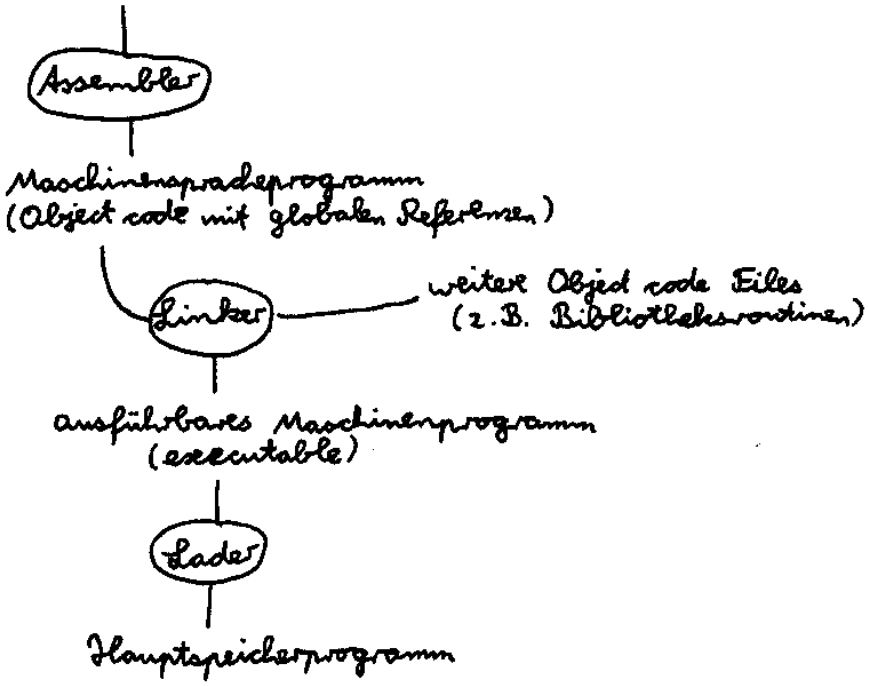
zu Fo 230



Übersetzung von Hochsprachenprogrammen



042.gif



Aufgaben des Assemblers

- Übersetzung von Pseudobefehlen,
  symbolische Namen für Befehlsadressen oder Daten
- Verwaltung von externen Referenzen in Symboltabellen
- Bereitstellung von Informationen für den Debugger

Aufgaben des Linkers

Zusammenfügen unabhängig assemblierter Object Files
zu Executable

- Platziere Code- und Datenmodule symbolisch im Speicher
- Übersetze Daten- und Instruktionslabel in
  relative Speicheradressen
- Zuordnung von internen und externen Referenzen

zu Fo 283

Ausführungsmodelle

Bsp. Realisierung von $A := B + C$,

wobei A, B, C Speicherplätze bezeichnen

Registermodell

a) mit Reg. / Speicher-Operanden

```
LOAD   R1, B
ADD    R1, C
STORE  R1, A
```

Zwei-Adress-Form

b) nur Register-Operanden

```
LOAD   R1, B
LOAD   R2, C
ADD    R1, R1, R2
STORE  R1, A
```

Drei-Adress-Form

→ LOAD/STORE - Architektur (RISC)

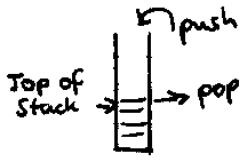
Akkumulator

```
LOAD   B
ADD    C
STORE  A
```

(Akkumulator ist impliziter Parameter)

Code ist kompakter als im Registermodell

Keller / Stack: nach LIFO (Last-In - First-Out) - Prinzip organisierte Datenstruktur



push : element × stack → stack

pop : stack → element × stack

Befehle für Kellermaschine

Keller ist jeweils impliziter Parameter

```
PUSH   adr   ≙ LOAD
POP    adr   ≙ STORE
```

ALU - Operationen werden jeweils auf den obersten Kellerelementen durchgeführt

ADD
↑
parameterloser Befehl

044.gif

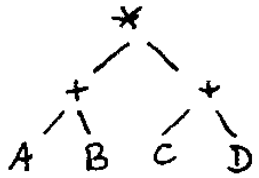
Realisierung von $A := B + C$

```
PUSH  B
PUSH  C
ADD
POP   A
```

Auswertung von arithmetischen Ausdrücken mit Stacks

Postfixnotation eines Ausdruckes liefert Stackcode

$(A + B) * (C + D)$ Infix



Präfixnotation
(Depth-first-left-to-right)

$* + A B + C D$

Postfix-Notation

$A B + C D + *$

PUSH A, PUSH B, ADD, PUSH C, PUSH D, ADD, MULT

	B		C	D	C+D	
A	A	A+B	A+B	A+B	A+B	(A+B)*(C+D)

ab Fo 287: komplett neue Folien (RISC-Prozessoren)

11.01.2005

zu Fo 303

$\$t_1$
$b_4\, b_3\, b_2\, b_1$ $\longrightarrow$ $\text{not}(\$t_1)$
$b_4'\, b_3'\, b_2'\, b_1'$

$$\sum_{i=1}^{4} b_i \cdot 2^{i-1}$$

$$\sum_{i=1}^{4} b_i' \cdot 2^{i-1}$$

$$= \sum_{i=1}^{4} (1 - b_i) \cdot 2^{i-1}$$

$$= \sum_{i=1}^{4} 2^{i-1} - \underbrace{\sum_{i=1}^{4} b_i\, 2^{i-1}}$$

$$2^4 - 1 - \$t_1$$

in MIPS: not (t_1) = $2^{32} - 1 - \$t_1$

$$\$t_1 + \$t_2 > \underbrace{2^{32}-1}_{\text{Überlauf}}$$

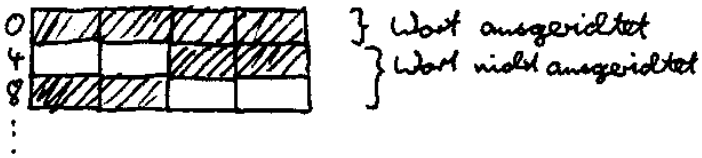
zu Fo

- .text <addr>
  ...
  ...
  ...
- .data
  ...
  ...
  ...

zu Fo 316

if $\hat{\$a0} < 1$ then return 1
else return fact ($a0 - 1) * $a0

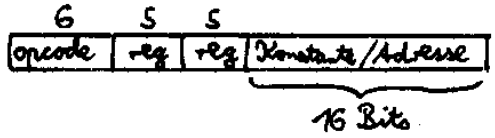
zu Fo 318



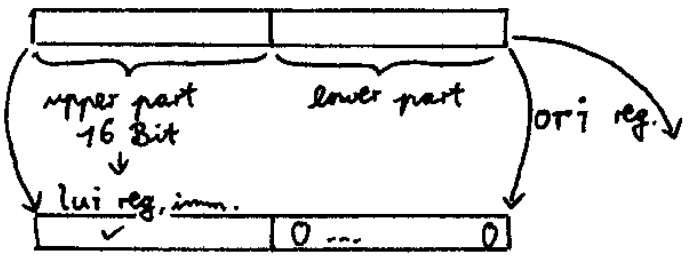
zu Fo 320 12.01.2005

Format von Lade-/Speicherbefehlen

bzw. Operationen mit immediate-Operanden

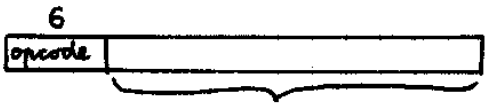


32-Bit-Konstante



zu Fo 321

Format von unbedingten Sprüngen



Wortadresse
≙ 28 Bit Byteadresse (zwei Nullen anhängen)

Fo 322: print_int ($s0) statt &s0

zu Fo 326

Kopieren von C-Strings

↑ Nullterminierung

source [i] → sink [i]

in C:

```
i = 0;
while (sink [i] = source[i]) != '\0'
    {i = i + 1}
```

Registerbelegung: $a0 ← sink
$a1 ← source
$s0 ← i
↑ Register muss gesichert werden

```
strcpy: addi  $sp , $sp , -4   } # $s0 sichern
        sw    $s0 ,  0($sp)
        li    $s0 , 0            # i=0

L1:     # sink[i] = source[i]
        add  $t1, $a1, $s0       # $t1 ← Adresse von source[i]
        lb   $t2, 0($t1)         # $t2 ← source[i]
        add  $t3, $a0, $s0       # $t3 ← Adresse von sink[i]
        sb   $t2, 0($t3)         # sink[i] ← $t2
        beq  $t2, $0 , L2        # falls source[i] = '\0'
                                 # goto L2
        addi $s0, $s0, 1         # i = i + 1
        j    L1

L2:     lw   $s0,  0($sp)        } # Restauriert $s0
        addi $sp,  $sp, 4
        jr   $ra
```

Statt den Array-Index in $s0 explizit zu verwalten und in jedem Schleifendurchlauf auf $a0 und $a1 aufzuaddieren, kann man auch die Basisregister $t1 und $t3 direkt inkrementieren.

```
strcpy2:  move  $t1, $a1
          move  $t3, $a0
L1:       lb    $t2, 0($t1)   } sink[i] ← source[i]
          sb    $t2, 0($t3)   }
          addi  $t1, $t1, 1
          addi  $t3, $t3, 1
          bne   $t2, $0, L1
          jr    $ra
```

zu Fo 328

100 ($t0)
↑ Offset ↑ Adresse

zu Fo 387 Programmzeile 24 und 30 vertauscht 19.01.2005
ori beq

(in Pipeline-Verarbeitung wird der unmittelbar nach dem Sprungbefehl folgende Befehl noch ausgeführt!)

zu Fo 371

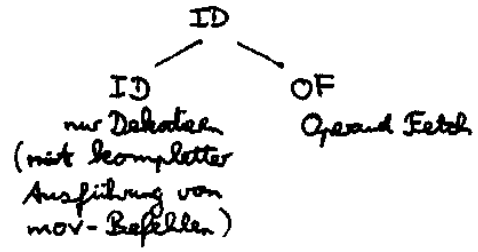
WAR - Hazard

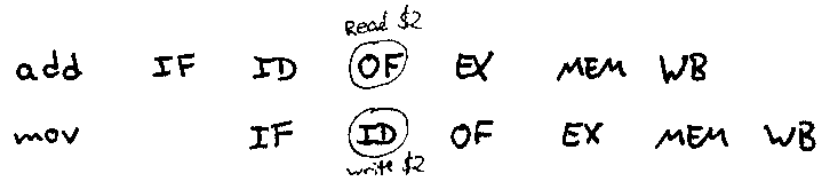
Bsp.
```
add $1, ($2)R, $3
mov ($2)W, $4
```

add	IF	(ID) Read $2	EX	MEM	WB	
mov		IF	ID	EX	MEM	(WB) Write $2

Bei MIPS-Pipeline tritt WAR-Hazard nicht auf.
Er ist nur möglich, falls in früher Phase geschrieben wird.

Modifikation der Pipeline:



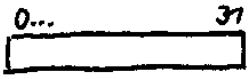


Falls vor dem Lesen geschrieben wird,
gibt es hier einen Konflikt.

25. 01. 2005

MIPS - Instruktionsverarbeitungsphasen

IF : IR ← Mem [PC]



NPC ← PC+4

ID: Dekodiere IR [0..5]

A ← Reg [$IR_{6..10}$]

B ← Reg [$IR_{11..15}$]

Imm ← extend ($IR_{16..31}$)

Vorzeichenexpansion

EX: 4 Fälle

- Speicherzugriff → Adressberechnung

  S ← A + Imm

- Register - Register - ALU - Operation

  S ← A op B, op aus $IR_{0..5}$ und $IR_{26..31}$

- Register - Imm - ALU - Op.

  $S \leftarrow A$ op Imm, op wie oben

- Sprung

  $S \leftarrow NPC + Imm$

  ggfs. Vergleich von A und B, um festzustellen, ob Sprung erfolgen soll

MEM: nur bei Lade-/Speicherbefehlen und Sprung aktiv

Laden: $M \leftarrow Mem[S]$

Speichern: $Mem[S] \leftarrow B$

Sprung: falls Sprung erfolgt, $NPC \leftarrow S$

alle ALU-Ops: $M \leftarrow S$

WB: $Reg(IR_{16..20}] \leftarrow M$ falls R-Format

$Reg[IR_{11..15}] \leftarrow M$ falls I-Format

$PC \leftarrow NPC$

Für die Fließbandverarbeitung ist eine Entkopplung dieser Phasen notwendig, dann spezielle Pufferregister (Pipeline Latches)

z.B. IF/ID.IR ... MEM/WB.IR

Hazards

- Strukturhazards

  z.B. Konflikt zwischen IF und MEM bei nur einem Speicherzugang

  Lösung in MIPS:

  getrennte Pufferspeicher (Cache)

  für Instruktionen und Daten

050.gif

- Datenhazards

- RAW-Hazards (Read After Write)

im MIPS:

Ⓐ ALU_OP reg, ..., ... } WRITE
... INSTR —, _reg... } READ
...

ALU_OP IF ID EX MEM WB
↑ Resultat liegt vor
↑ Resultat wird geschrieben

IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX ...
IF ID
↑ Resultat kann normal gelesen werden

↘ Für die 3 nachfolgenden Instruktionen ist eine Sonderbehandlung erforderlich.

Lösung: 1) Forwarding für die beiden nachfolgenden Instruktionen

für nächst. Instr.: ID/EX ALU_Input ← EX/MEM ALU_Output

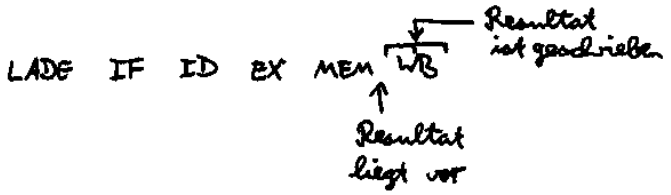
für zweite nächst. Ins: —› ——————— ← MEM/WB ALU_Output

2) Aufteilung der ID und WB-Stufe

in zwei Phasen [W|R]
↑ write WB
↑ read ID

↘ In WB geschriebene Werte können im selben Takt gelesen werden

Ⓑ LADE reg, ... } WRITE
... INSTR ..., _reg... } READ



IF ID (EX) MEM WB

Die direkt nachfolgende Instruktion kann geladenen Wert nicht mit ALU verarbeiten.

Lösung: Compiler Scheduling
„geeignete Anordnung der Instruktionen durch Compiler"

Für weite nachfolgende Instruktion ist Forwarding möglich.

- WAR -
- WAW - } Hazards, in MIPS nicht möglich

Kontroll-Hazards

bedingter Sprungbefehl

zu Fo 384 Bsp.

```
div.d   F0, F2, F4    RAW-Hazard
add.d   F10, F0, F8
sub.d   F6, F6, F14   ← unabhängig
```

↝ sub.d kann vor der Addition add.d ausgeführt werden.

```
div.d   F0, F2, F4    RAW-Hazard
add.d   F10, F0, F8   WAR-Hazard
sub.d   F8, F6, F14
```

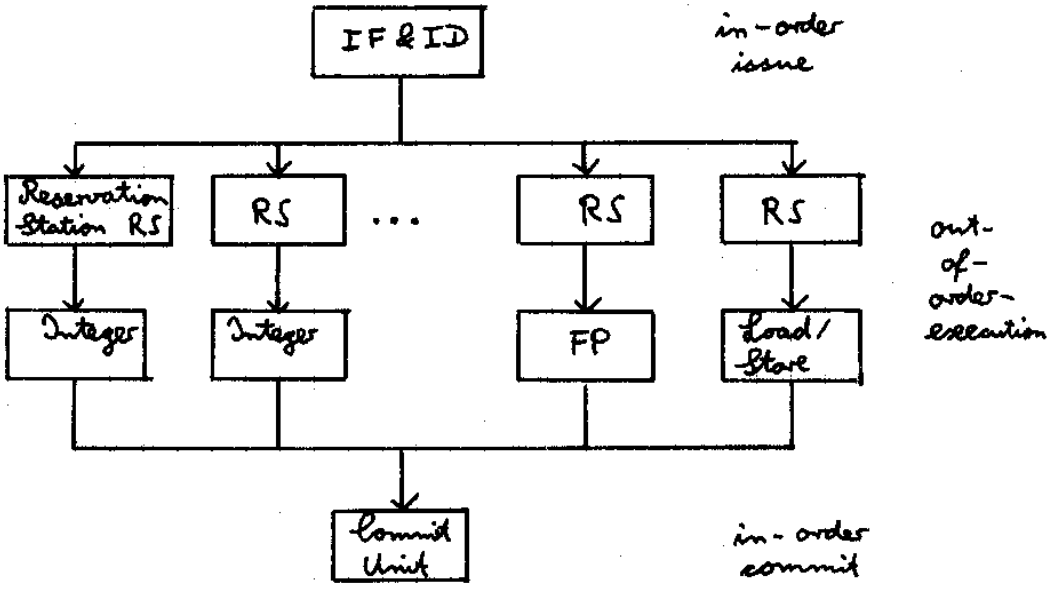
WAR-Hazard ↝ sub.d darf erst schreiben, wenn add.d F8 gelesen hat.

052.gif

26.01.2005

zu Fo 385

Aufbau einer Pipeline mit dynamischer Instruktionsanordnung

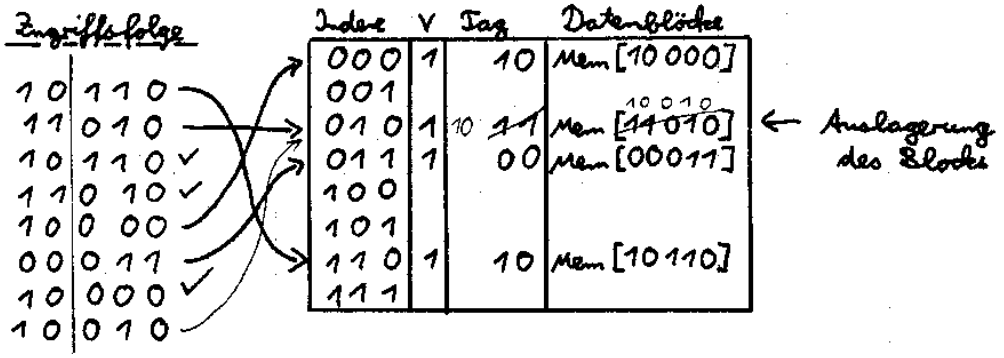


zu Fo 395

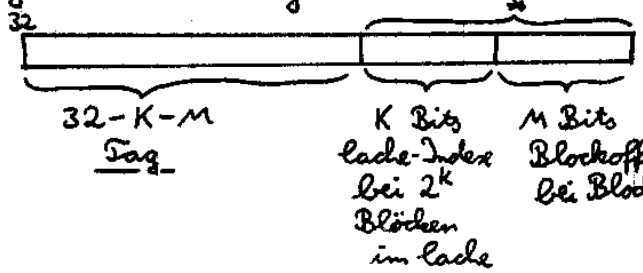
Bsp. Cache-Zugriffe bei direkter Abbildung

Speichergröße 32 Blöcke → 5 Bit-Blockadressen

Cache-Größe 8 Blöcke → 3 Bit Cache-Index



allg. Adressumsetzung bei direkter Abb.



* N bei Cache-Größe 2^N

053.gif

01.02.2005

Sondertutorium:
Mi 16⁰⁰h HS I Lahnberge

Di fällt VL aus!
Mi Klausureinsicht

zu Fo 404

Blockadresse

Speicheradr. | | k |

Tag | Index | Blockoffset

↝ Blöcke der Größe 2^k (= L)

Trefferquote $h = \frac{\#\text{erfolgreiche Cache-Zugriffe}}{\#\text{aller Speicherzugriffe}}$

Fehlschlagquote $1-h$ (h wie engl. „hit" = Treffer)

mittlere Zugriffszeit $t_M = h \cdot t_h + (1-h) \cdot t_f$

t_h: Trefferzeit; t_f: Fehlschlagzeit

$t = t_h + t_p$

t_p: Nachladezeit (penalty)

↝ $\boxed{t_M = t_h + (1-h) \cdot t_p}$

zu Fo 408

2:1 - Regel

Fehlschlagquote (Cache der Größe N mit direkter Abbildung)

↝ Fehlschlagquote (Cache der Größe $\frac{N}{2}$ mit 2-facher Mengenass.)

zu Fo 417

Compiler-kontrolliertes Prefetching

Idee: prefetch-Instruktion

↝ Vorausladen von Daten in Cache

- „faulting" prefetch" bedeutet, dass
  Seiten- oder Segmentfehler
  (Fehlschläge auf niedrigeren Hierarchie-Ebenen)
  ausgelöst werden können
- „non-faulting prefetch"
  ≙ no-op bei Fehlschlag auf
  nächstniedrigere Hierarchiestufe

wichtig: Cache muss nicht-blockierend sein,
d.h. während des Nachladens von Blöcken
müssen weitere Zugriffe möglich sein

Ziel: Einfügen von prefetch-Instr. für Daten,
die wahrscheinlich zu einem Fehlschlag führen.

Bsp. 8 KB Cache mit direkter Abb. und 16 Byte-Blöcken.
write back mit write allocate

Felder Double a [3, 100] } Zu Beginn
8B b [101, 3] } nicht im Cache

Zeilenweise Speicherung

```
for (i=0; i<3; i=i+1)
  for (j=0; j<100; j=j+1)
    a[i][j] = b[j][0] * b[j+1][0]
```

a: Cache-Block

a_{00}	a_{01}	a_{02}	a_{03}	a_{04} ...
a_{10}	a_{11}	a_{12} ...		

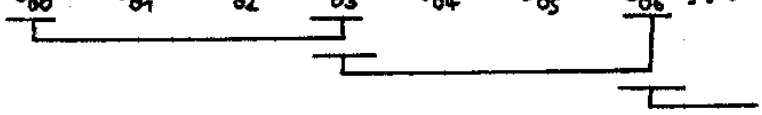
055.gif

→ räumliche Lokalität: gerade j-Werte → Fehlschlag

ungerade j-Werte → Treffer

⇒ # Fehlschläge $\frac{3*105}{2} = 150$

b: b_{00} b_{01} b_{02} b_{03} b_{04} b_{05} b_{06} ...



→ temporale Lokalität: 101 Fehlschläge

Annahme: Nachladezeit erfordert Prefetch jeweils 7 Iterationen im Voraus.

Vorgehensweise: Aufsplitten der Schleifen

Zunächst Prefetch von a und b

danach nur noch von a

```
for (j=0; j<100; j=j+1)
  {prefetch (b[j+7][0]);
   prefetch (a[0][j+7]);
   a[0][j] = b[j][0] * b[j+1][0]}
for (i=1; i<3; i=i+1)
   for (j=0; j<100; j=j+1)
     {prefetch (a[i][j+7]);
      a[i][j] = b[j][0] * b[j+1][0] }
```

Analyse: prefetch von a[i][7] ... a[i][99]
b[7][0] ... b[99][0]

⇒ verbleibende Fehlschläge $\underbrace{\frac{3*7}{2}}_{a} + \underbrace{7 + 1}_{b}^{?} = 19$

Reduktion der Fehlschläge von 251 auf 19
mit 400 zusätzlichen Prefetch-Instruktionen

zu Fo 431 02.02.2005

Bsp. Lokale vs. Globale Fehlschlagsquote

1000 Speicherzugriffe der CPU
40 Fehlschläge im L1 Cache
20 Fehlschläge im L2 Cache

Lokale L1-Fehlschlagsquote: 4%
Lokale L2-Fehlschlagsquote: 50%
Globale Fehlschlagsquote: 2%

PDF-Skript Version d
(02.02.2005)

-letzte Version-