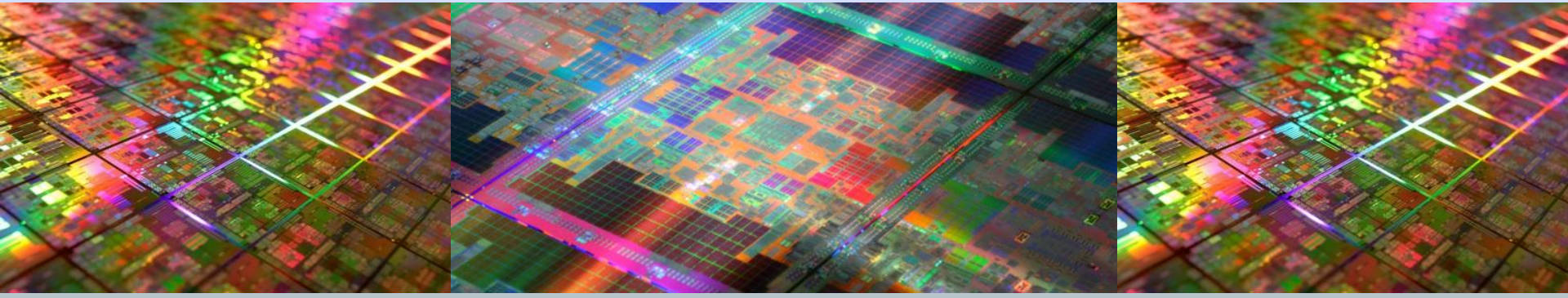
Rechnerarithmetik

Technische Grundlagen der Informatik für Wirtschaftsinformatik

Stefan Podlipnig

TU Wien



Lernziele

- Verständnis für die Realisierung von Rechenwerken
  - Für Operationen auf ganzzahlige Operanden
  - Implementierung als Schaltnetz
- Verständnis für die Einschränkungen bei der Zahlendarstellung in Rechnern

DIGITALE GRUNDLAGEN

Allgemein

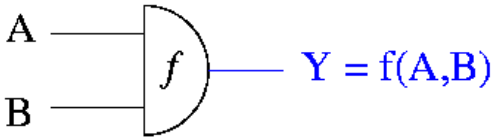
- Boolesche Algebra
  - „Boolesche Algebra“ wie in der Vorlesung Formale Modellierung
  - Boolesche Algebra dient als
    - Beschreibung von Schaltungen
    - Hilfsmittel zur Berechnung binärer Schaltnetze und Schaltwerke

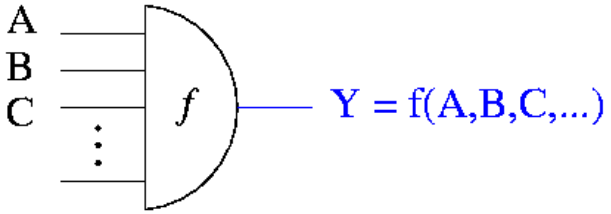
Formale Modellierung

- Elementare Funktionen auf {0, 1}
  - Werden in der technischen Informatik durch Gatter realisiert
  - Komplexe Funktionen durch eine geeignete Verschaltung von mehreren Gattern

Gatter

- Gatter (*gate*)
  - Ist ein elektronischer Schalter, der binäre Werte verknüpft
  - Wird aus einfachen elektronischen Bauteilen (Transistoren, Dioden, Widerstände) zusammengesetzt
  - Ist eine „Black Box“ mit einem, zwei oder mehreren Eingängen A, B, C, … ∈ {0,1} und genau einem Ausgang Y ∈ {0,1} zur Realisierung einer Funktion Y = *f*{A, B, C,…}

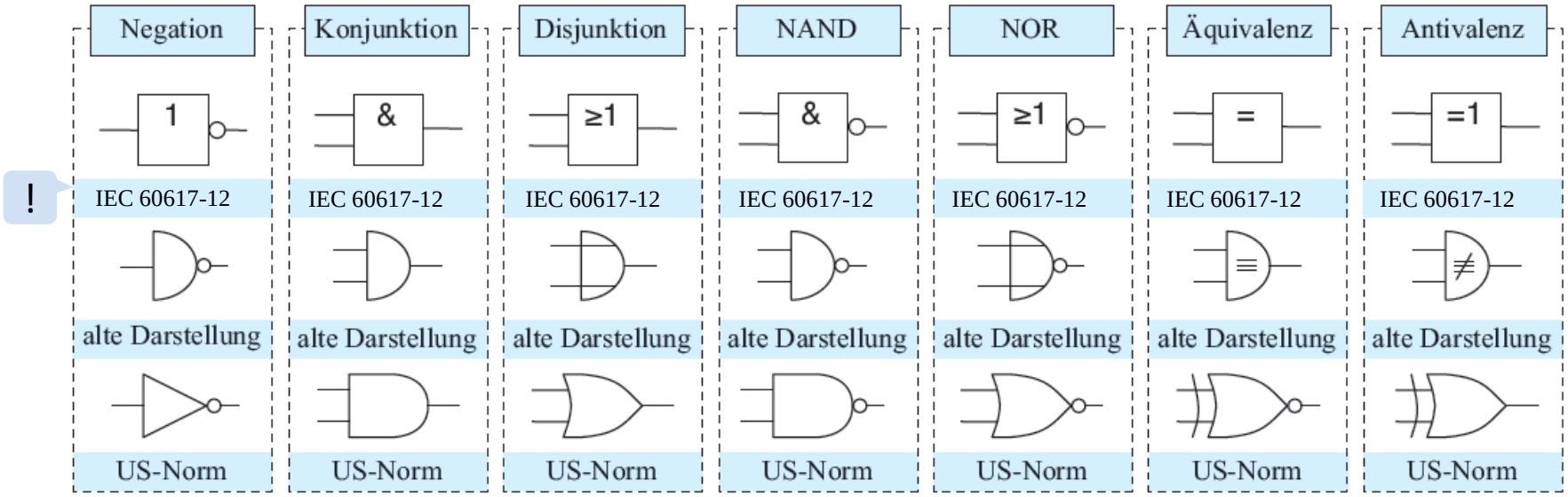




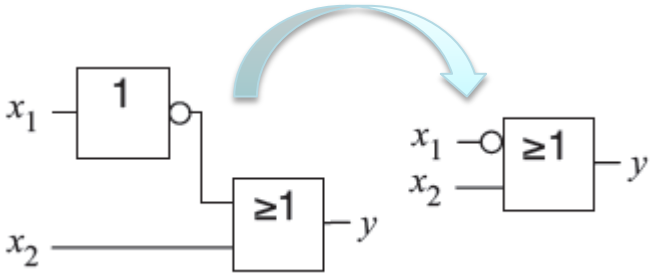
A	B	Y
0	0	
0	1	
1	0	
1	1	

A	B	C	…	Y
0	0	0		
0	0	1		
⋮	⋮	⋮		
1	1	1		

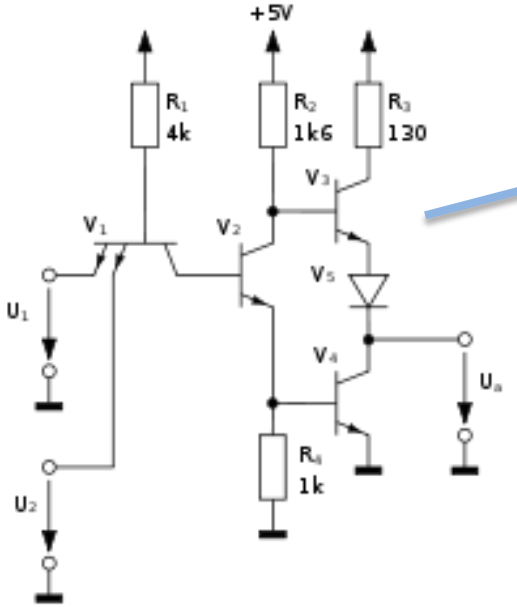
Gatter – Symbole

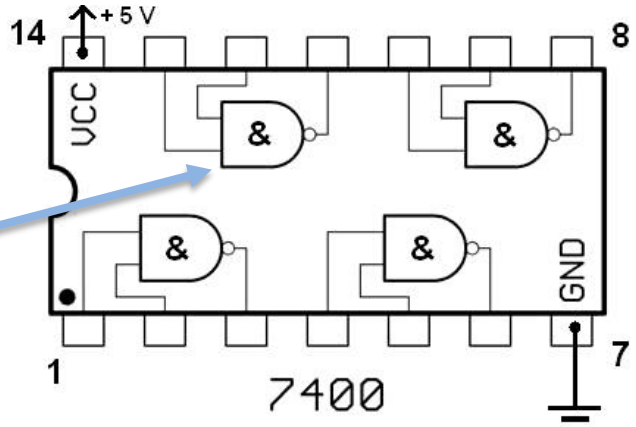


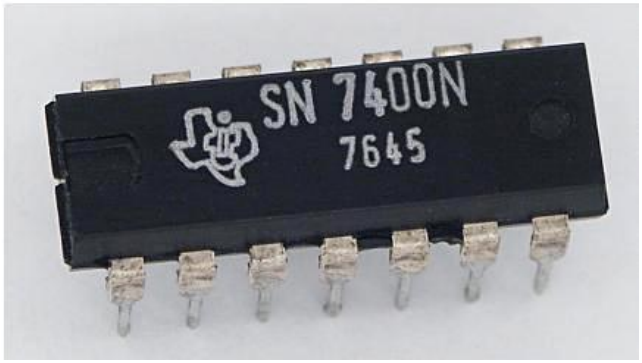
- Verkürzte Schreibweise (Beispiel)



Überblick


+5V
R1
4k
R2
1k6
R3
130
V1
V2
V3
V5
V4
U1
U2
Ua
R4
1k


14
+5 V
8
VCC
&
&
&
&
GND
1
7
7400

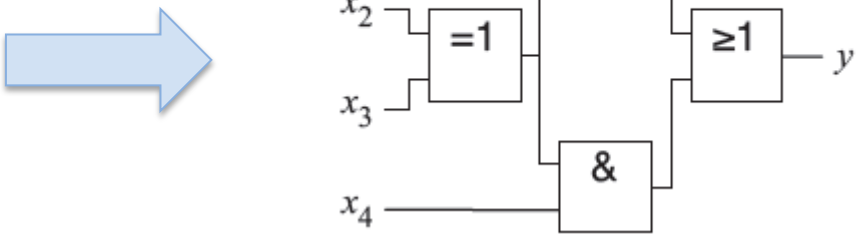

SN 7400N
7645

Beispiel

- Wahrheitstabelle und mögliche Schaltung (Schaltnetz)

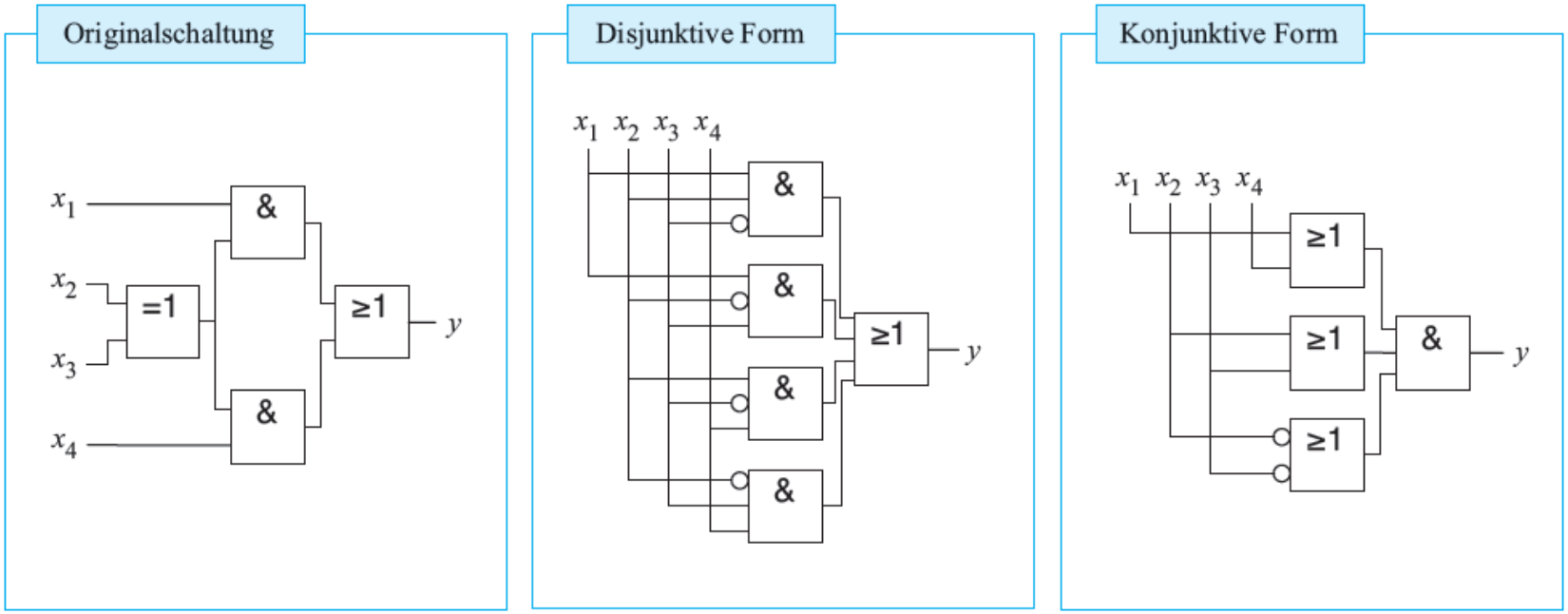
	x_4	x_3	x_2	x_1	y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

Schaltnetz: *Eine Funktionseinheit zum Verarbeiten von Schaltvariablen, deren Wert am Ausgang zu irgendeinem Zeitpunkt nur vom Wert am Eingang zu diesem Zeitpunkt abhängt.“*
[DIN 44300, ISO/IEC 2382]



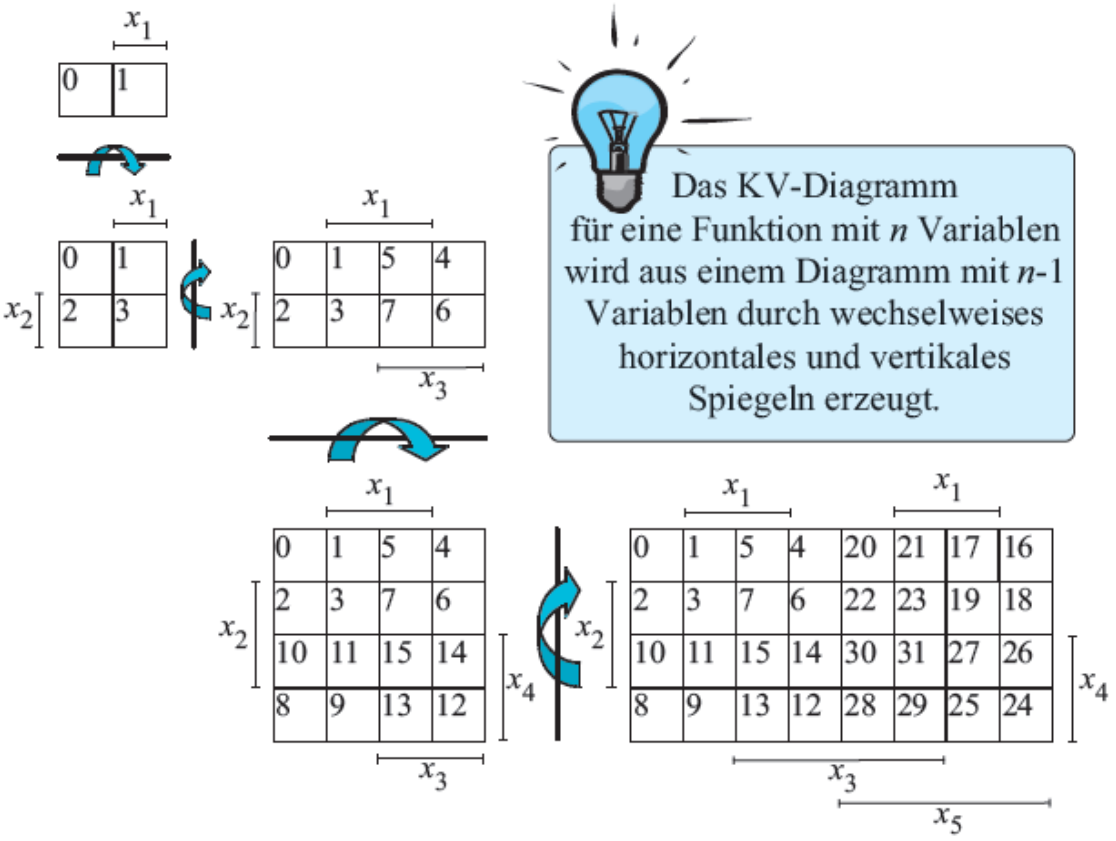
Transformation

- Auf der Logikebene besitzt jede boolesche Funktion mehr als eine Darstellung
  - Jede für sich führt zu einer ganz unterschiedlichen Implementierung auf der Hardware-Ebene
- Beispiel: Originalschaltung, disjunktive Form, konjunktive Form



Karnaugh-Veitch-Diagramme (1)

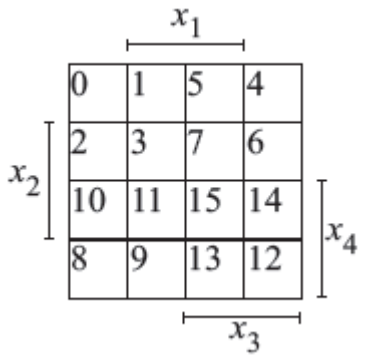
- Karnaugh-Veitch-Diagramme (KV-Diagramme) sind eine spezielle grafische Darstellung für boolesche Funktionen
  - Daraus lässt sich eine minimierte zweistufige Schaltungsdarstellung auf einfache Weise ableiten

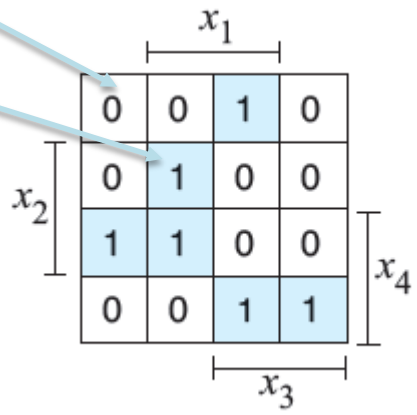


Karnaugh-Veitch-Diagramme (2)

- Beispiel von vorher:

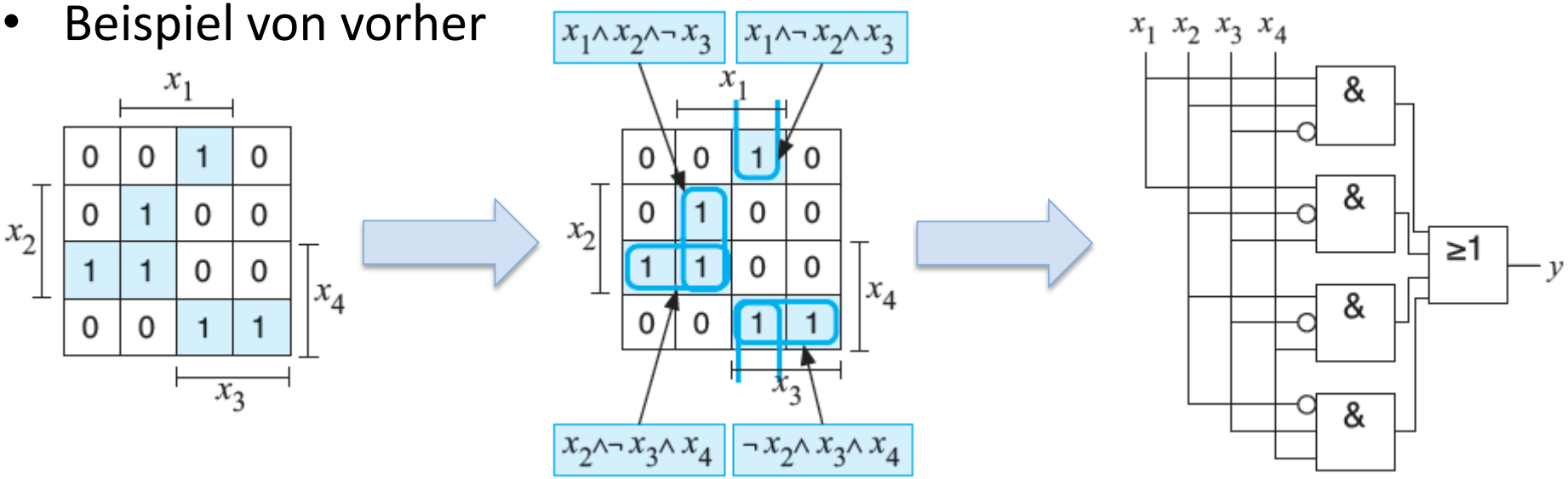
	x_4	x_3	x_2	x_1	y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0





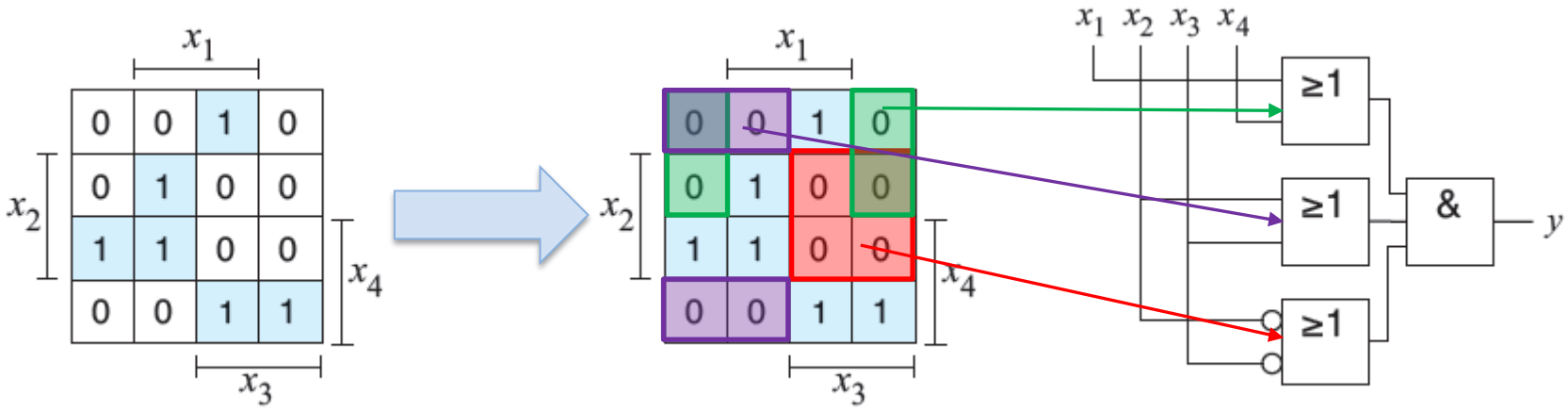
Karnaugh-Veitch-Diagramme (3)

- Im KV-Diagramm sucht man möglichst große 1er-Blöcke
  - Einschränkungen
    - Größe muss eine Zweierpotenz sein (2^0, 2^1, 2^2, …)
    - Blöcke müssen eine Rechteckform aufweisen
    - Anzahl der Blöcke sollte minimal sein (größere Blöcke bei Alternativen wählen)
- Weitere Besonderheiten
  - Blöcke dürfen sich teilweise überlappen
  - Blöcke können sich über die Ränder des Diagramms erstrecken
- Beispiel von vorher



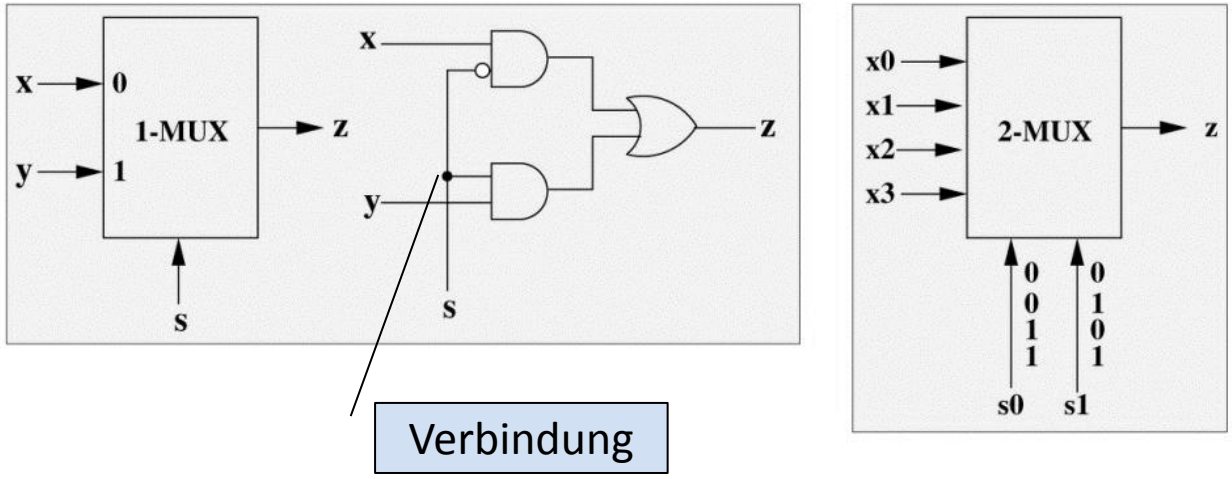
Karnaugh-Veitch-Diagramme (4)

- Inverse Blockbildung
  - Konjunktive Minimalform bestimmen
  - Dazu müssen möglichst große 0er-Blöcke gebildet werden
  - Das Ergebnis muss aber negiert ausgelesen werden
- Beispiel von vorher



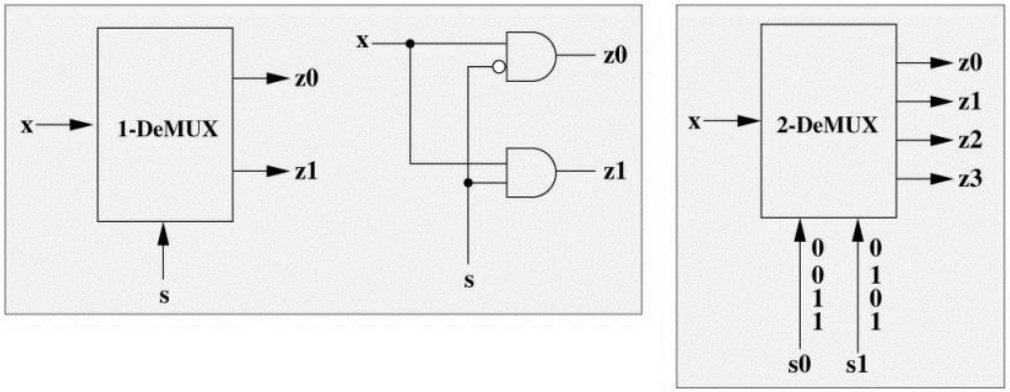
Spezielle Schaltnetze – Multiplexer

- Multiplexer
  - n Steuerleitungen $s_{n-1}, \ldots, s_1, s_0$
  - $k = 2^n$ Eingänge $x_0, x_1, \ldots, x_{k-1}$
  - Ein Ausgang
  - Ausgang erhält den Wert der ausgewählten Eingangsleitung
- Benötigt z.B. zur Auswahl einer Datenquelle



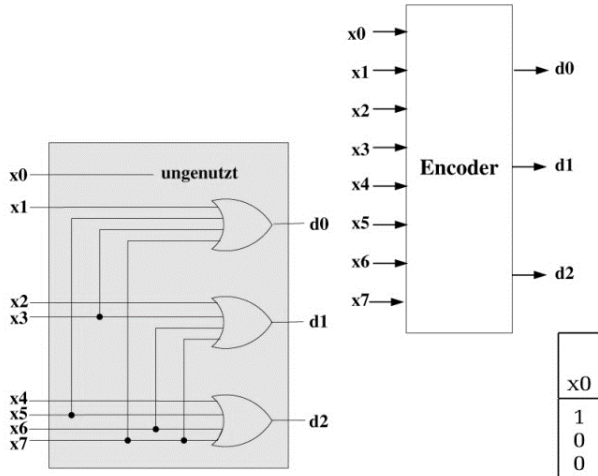
Spezielle Schaltnetze – Demultiplexer

- Demultiplexer
  - n Steuerleitungen $s_{n-1}, \ldots, s_1, s_0$
  - Ein Eingang
  - $k = 2^n$ Ausgänge $z_0, z_1, \ldots, z_{k-1}$
  - Ausgewählter Ausgang erhält den Wert der Eingangsleitung
- Benötigt z.B. zur Auswahl einer Datensenke (z.B. Speicherzeile)



Spezielle Schaltnetze – Encoder

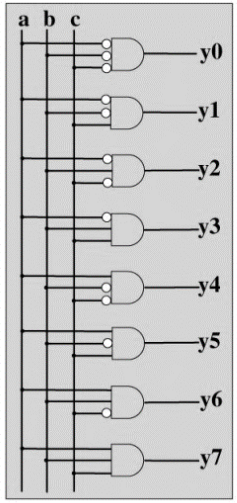
- Encoder
  - n Ausgänge
  - 2^n Eingänge
  - Nur genau eine Eingangsleitung darf auf 1 sein: $x_i = 1$, $x_{j \neq i} = 0$
  - Bitmuster an den Ausgängen entspricht Nummer der Eingangsleitung, die auf 1 gesetzt ist
- Benötigt z.B. zur Kodierung von Tasten einer Tastatur

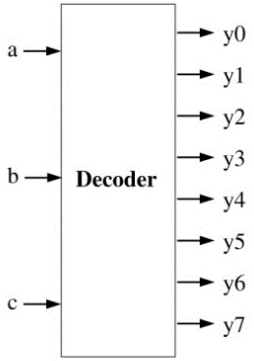


Eingänge								Ausgänge		
x0	x1	x2	x3	x4	x5	x6	x7	d2	d1	d0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Spezielle Schaltnetze – Decoder

- Decoder
  - *n* Eingänge
  - 2^n Ausgänge
  - Bitmuster an den Eingängen bestimmt die Ausgangsleitung, die auf 1 gesetzt wird
- Benötigt z.B. zur Dekodierung von Adressen oder Instruktionen



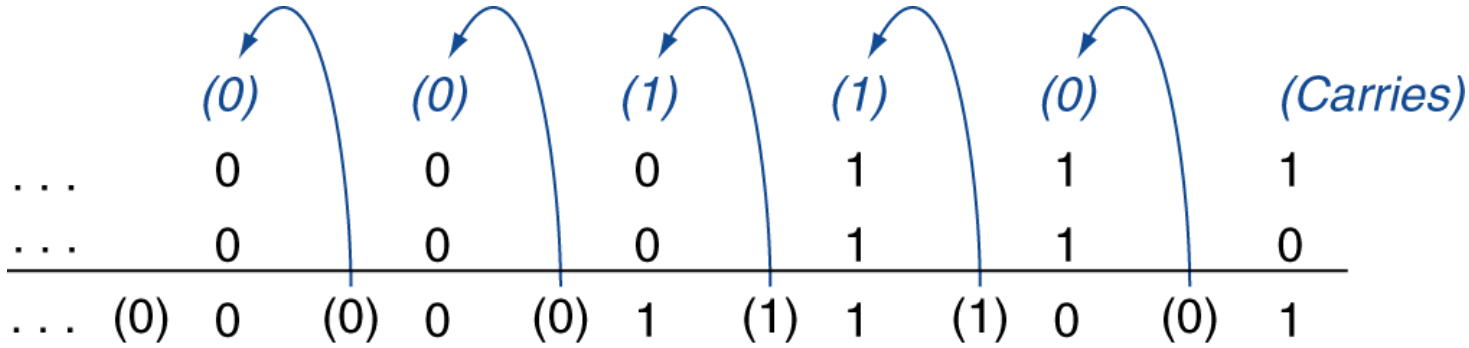


Eingänge			Ausgänge							
a	b	c	y0	y1	y2	y3	y4	y5	y6	y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

OPERATIONEN AUF GANZZAHLEN

Addition von Ganzzahlen

- Beispiel: 7 + 6



- Überlauf (bei Zweierkomplement) tritt auf, wenn die Hardware das Ergebnis nicht mehr darstellen kann
  - Ein Operand ≥ 0, ein Operand < 0 → Überlauf nicht möglich
  - Beide Operanden ≥ 0 → Überlauf, wenn das Vorzeichenbit des Ergebnisses 1 ist
  - Beide Operanden < 0 → Überlauf, wenn das Vorzeichenbit des Ergebnisses 0 ist

Subtraktion von Ganzzahlen

- Addiere die Negation des zweiten Operanden
- Beispiel: 7 – 6 = 7 + (–6)
  - +7: 0000 0000 … 0000 0111
    –6: 1111 1111 … 1111 1010
    +1: 0000 0000 … 0000 0001
- Überlauf bei einer Subtraktion
  - Gleiche Vorzeichen bei Operanden → Überlauf nicht möglich
  - Ein Operand ≥ 0 wird von einem Operanden < 0 subtrahiert → Überlauf, wenn das Vorzeichenbit des Ergebnisses 0 ist
  - Ein Operand < 0 wird von einem Operanden ≥ 0 subtrahiert → Überlauf, wenn das Vorzeichenbit des Ergebnisses 1 ist

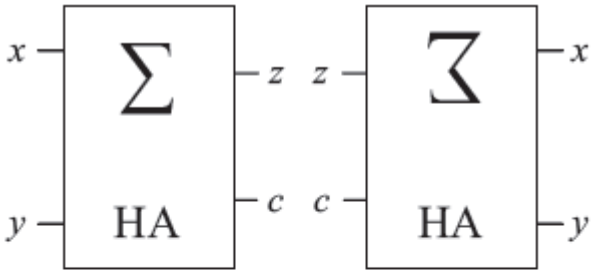
Überlauf – Zusammenfassung

- Überlauf bei MIPS
  - Wenn das Ergebnis nicht mehr korrekt als 32-Bit Zahl dargestellt werden kann
  - Vorzeichenbit enthält ein Bit des Wertes und nicht mehr das korrekte Vorzeichen
  - Addition von Operanden mit unterschiedlichen Vorzeichen bzw. Subtraktion von Operanden mit gleichen Vorzeichen führt nie zu einem Überlauf
    - Möglicher Übertrag an der höchsten Stelle kann ignoriert werden
- Überlaufbedingungen

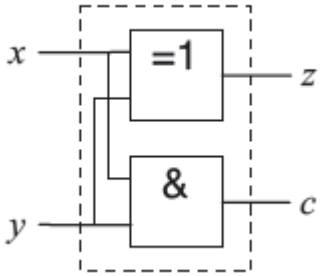
Operation	Operand A	Operand B	Überlauf, wenn Ergebnis
A + B	≥ 0	≥ 0	< 0
A + B	< 0	< 0	≥ 0
A - B	≥ 0	< 0	< 0
A – B	< 0	≥ 0	≥ 0

Addition – Halbaddierer

- Addition zweier Binärziffern *x* und *y* erfordert einen Halbaddierer, der zwei Ergebnisse ermittelt
  - Summe: Im folgenden als *s* oder *z* bezeichnet
  - Carry: Im folgenden als *c* bezeichnet
- Schaltsymbol, Wahrheitstafel und Schaltung

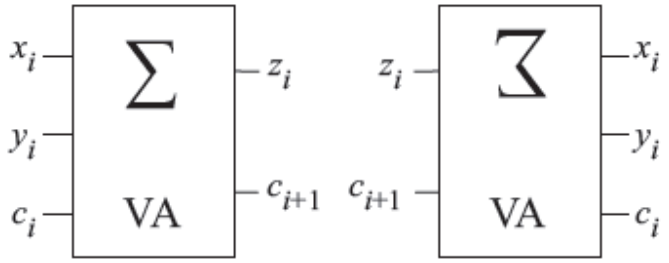


x	*y*	*z*	*c*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

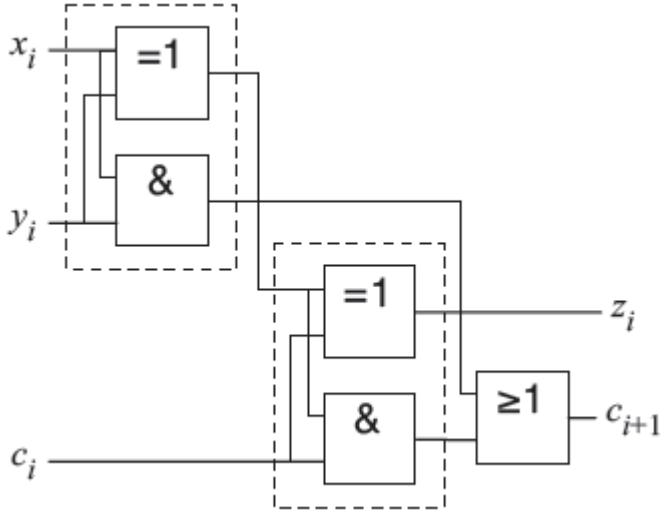


Addition – Volladdierer

- Addition von x_i, y_i und c_i an den Bitpositionen i = 1, ... , n–1 erfordert einen Volladdierer, der die Summe s_i (bzw. z_i) und den Übertrag c_{i+1} bestimmt
- Schaltsymbol, Wahrheitstafel und Schaltung

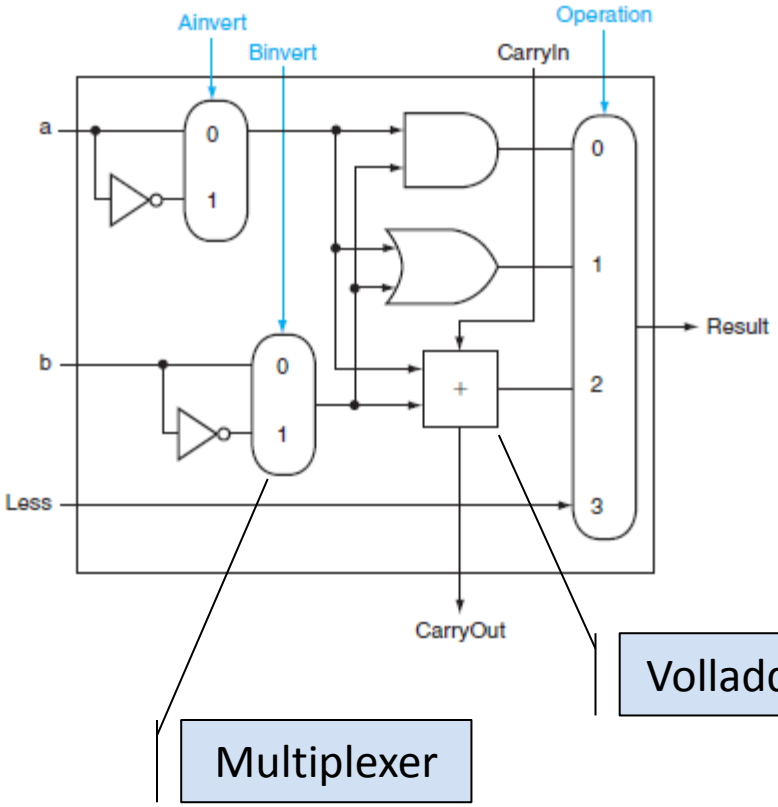


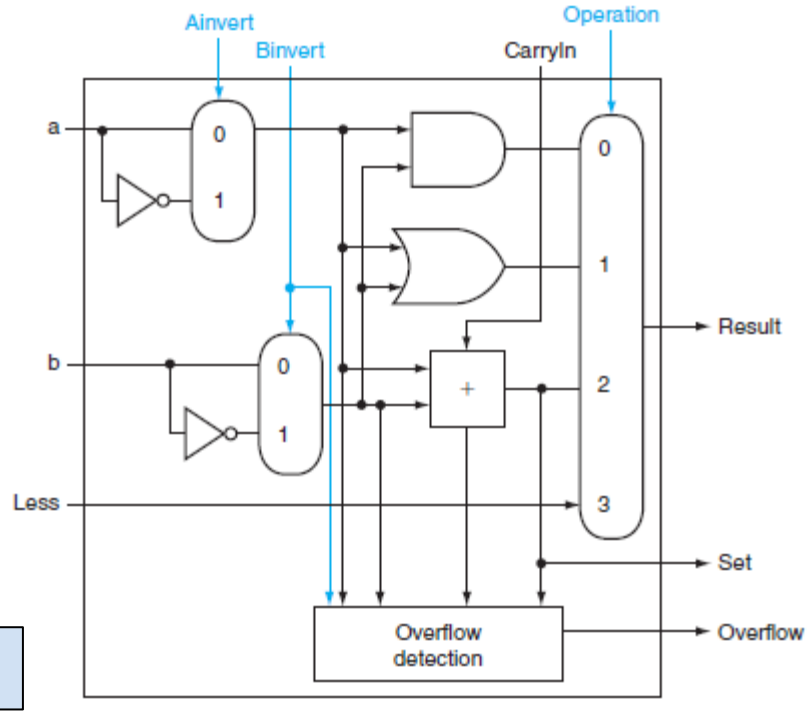
x_i	y_i	c_i	z_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



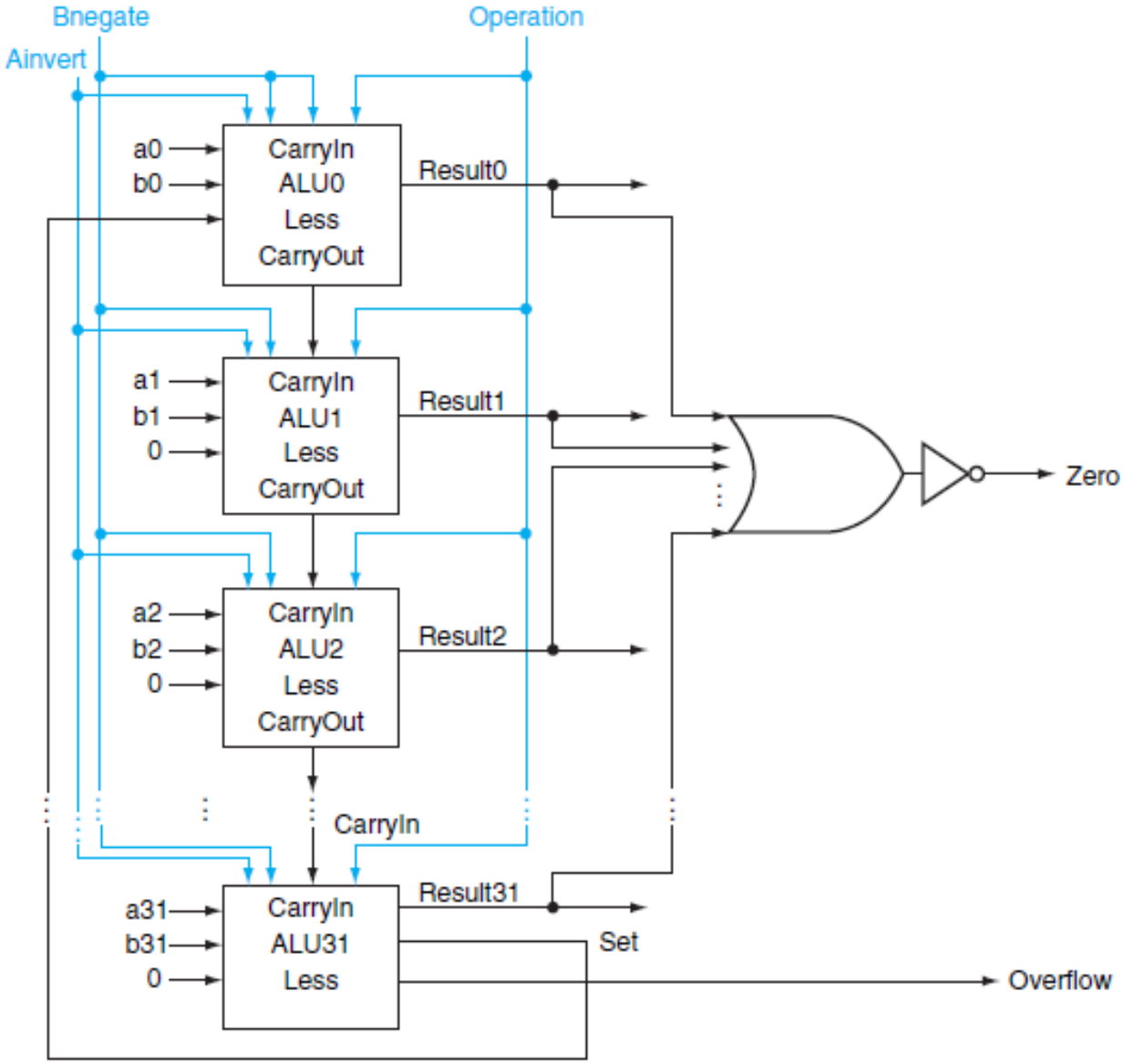
Einfache 1-Bit MIPS ALU

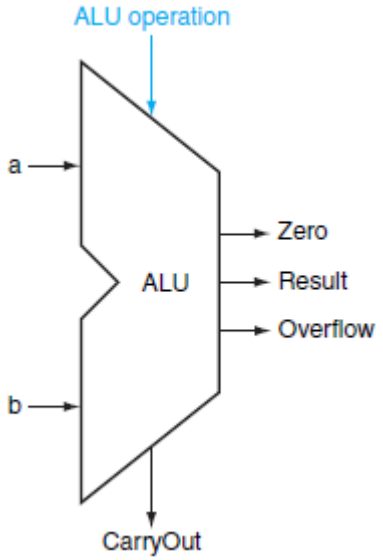
- Unterstützt and, or, add, slt
  - Rechts ALU für höchstwertiges Bit einer n-Bit Zahl





32-Bit ALU nach dem Ripple-Carry-Prinzip





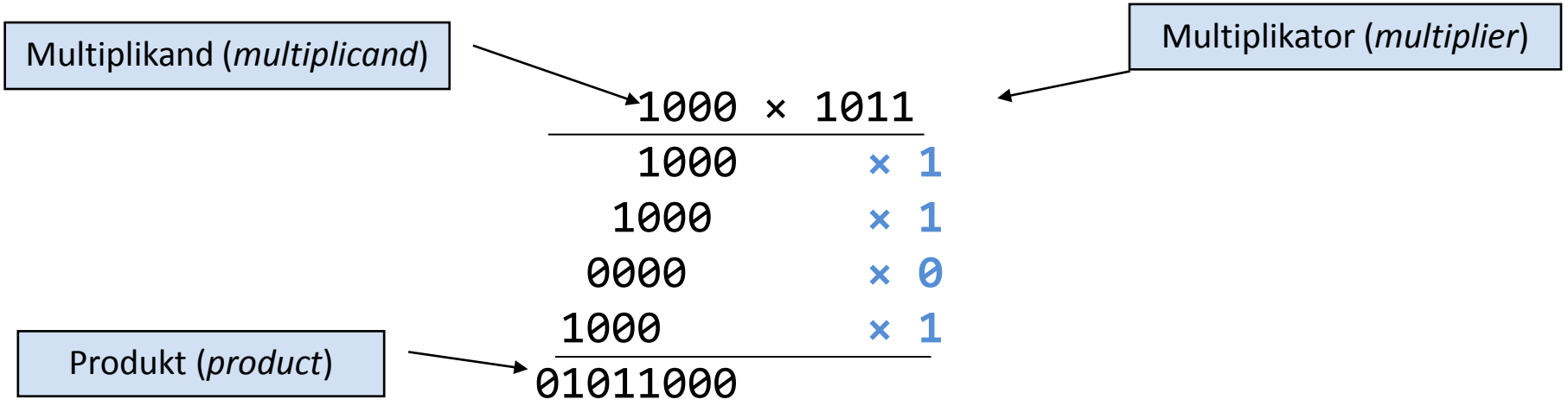
32-Bit ALU (Erklärung)

- Operanden a und b haben jeweils 32 Bit
  - Daher werden 32 1-Bit ALUs benötigt
- Steuersignale für 1-Bit ALU an Stelle i
  - Ainvert
    - 0: a_i wird unverändert übernommen
    - 1: a_i wird invertiert
  - Bnegate
    - 0: b_i wird unverändert übernommen, CarryIn von ALU0 ist 0
    - 1: b_i wird invertiert, CarryIn von ALU0 ist 1 (realisiert Zweierkomplement von b!)
  - Operation (2 Bit)
    - 00 = and, 01 = or, 10 = add, 11 = slt
- Anordnung der vier Steuerleitungen
  - Ainvert, Bnegate, Operation-Bit1, Operation-Bit2
  - Beispiel (slt)
    - Ainvert = 0, Bnegate = 1, Operation = 11

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

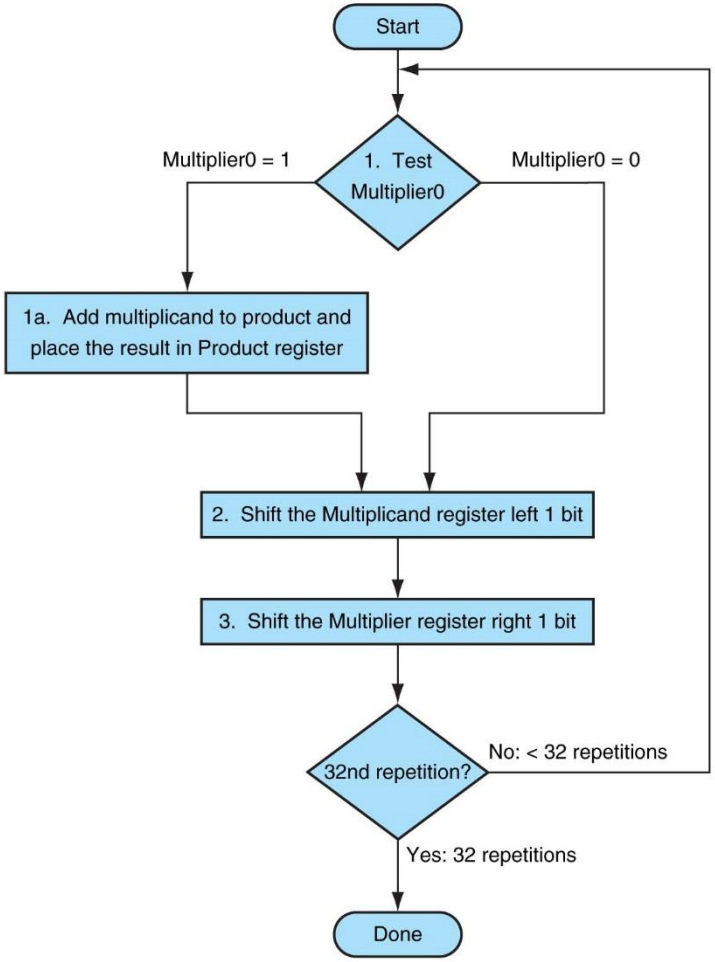
Multiplikation

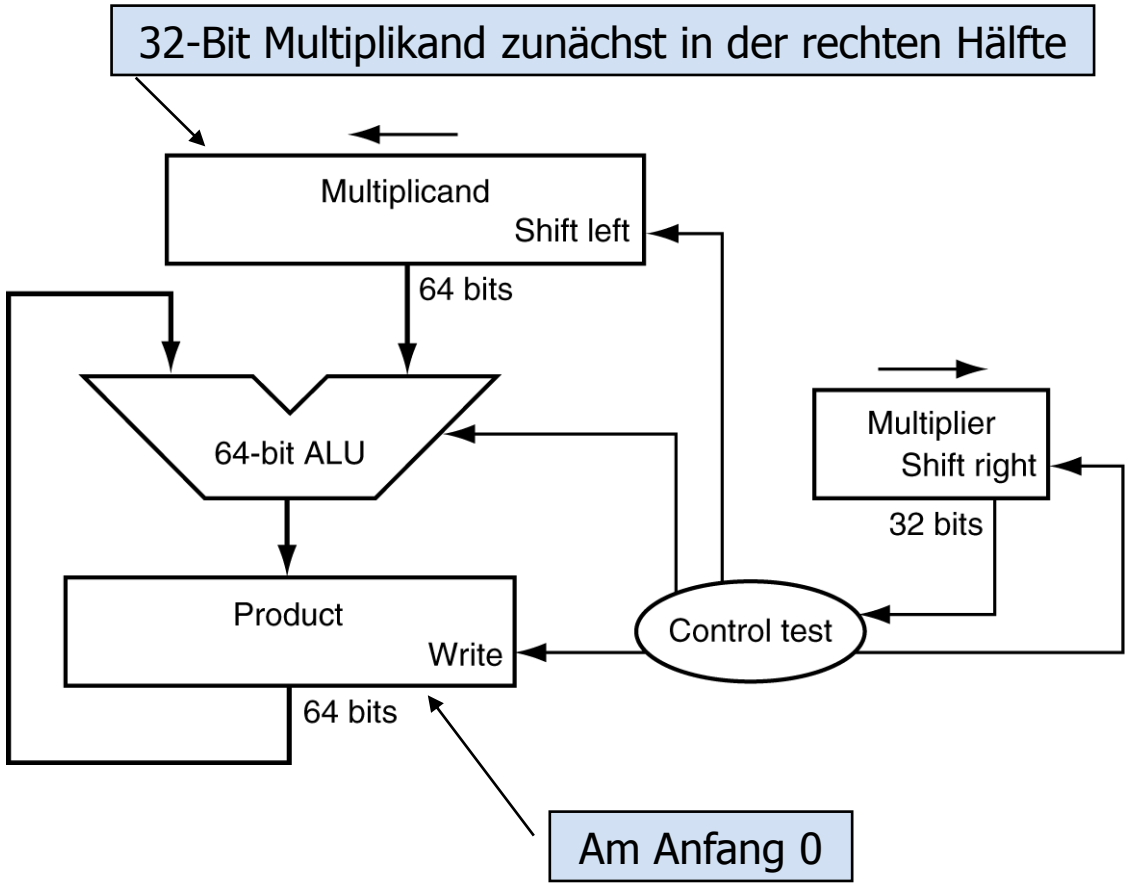
- Algorithmus zur Multiplikation zweier Binärzahlen a und b (beide größer gleich 0) entspricht in etwa der handschriftlichen Multiplikation zweier Dezimalzahlen



- Produkt zweier n-Bit Binärzahlen hat 2n Bitstellen
- Algorithmus ist zurückführbar auf wiederholte bedingte Additionen und Schiebeoperationen

Hardware für Multiplikation





Beispiel (0010 x 0011)

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⟹ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⟹ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⟹ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⟹ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

Multiplikation in MIPS

- In MIPS werden zwei 32-Bit Zahlen aus zwei Registern multipliziert

```
mult reg1,reg2   # vorzeichenbehaftete Multiplikation
multu reg1,reg2  # vorzeichenlose Multiplikation
```

  - Ergebnis hat eine Wortbreite von 64 Bit und befindet sich in den Spezialregistern Hi und Lo, die jeweils 32 Bit breit sind
- Weiterverarbeitung des Produktes erfolgt durch

```
mfhi reg     # Laden der höheren 32 Bit des Produktes
mflo reg     # Laden der niedrigen 32 Bit des Produktes
```

  - Lade Hi/Lo ins **reg**
  - Hi-Wert testen um Überlauf bei Produkt zu erkennen
- Der Assembler realisiert zusätzlich den Pseudobefehl

```
mul reg3,reg1,reg2 # vorzeichenbehaftete Multiplikation
```

  - Ohne Überprüfung, ob das Ergebnis in 32 Bit dargestellt werden kann

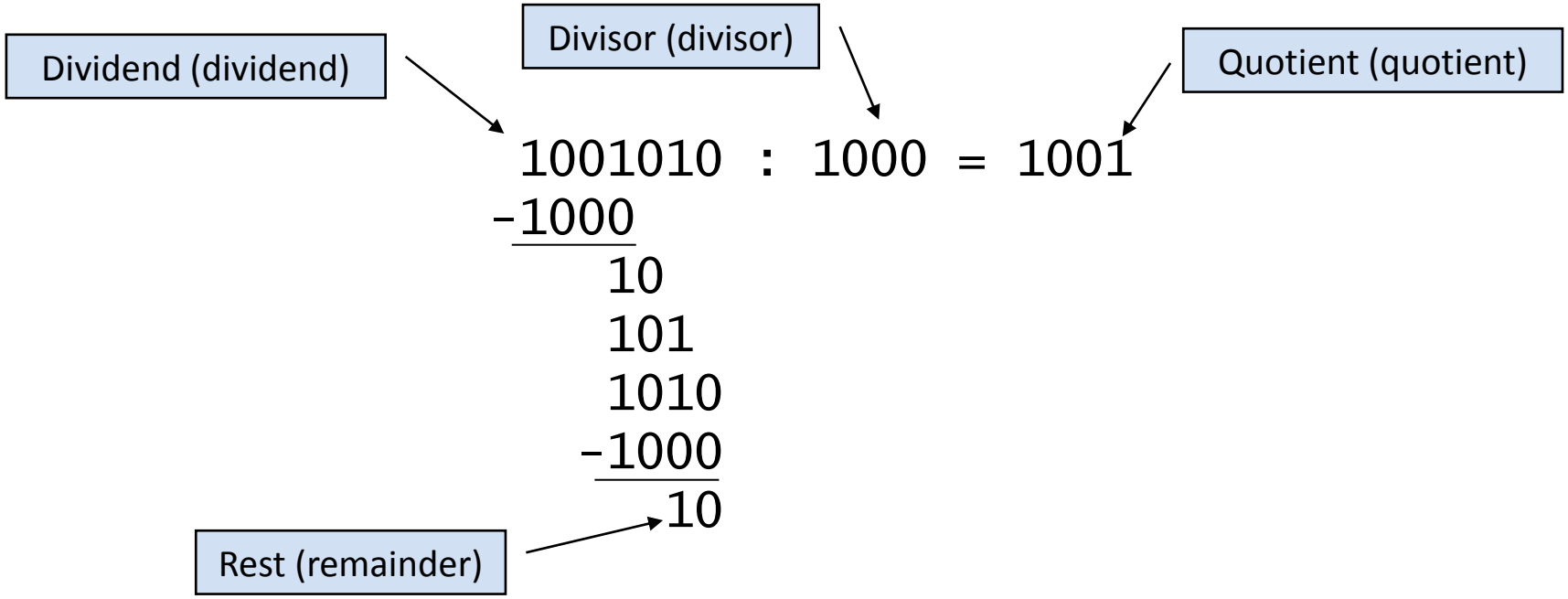
Division

- Umkehrung der Multiplikation
  - Berechnung von q = a / b durch wiederholte bedingte Subtraktionen und Schiebeoperationen
- In jedem Schritt wird Divisor b testweise vom Dividenden a subtrahiert

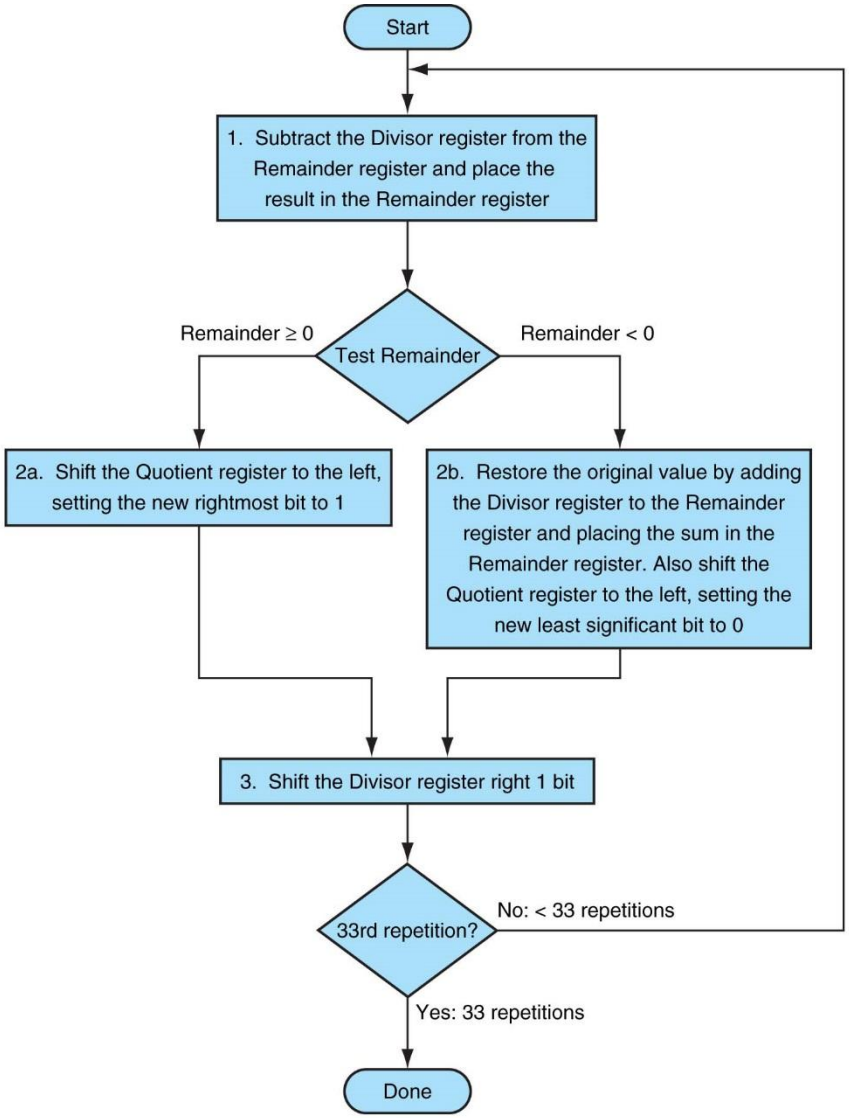
  $q_i = 1$, falls $a - b > 0$

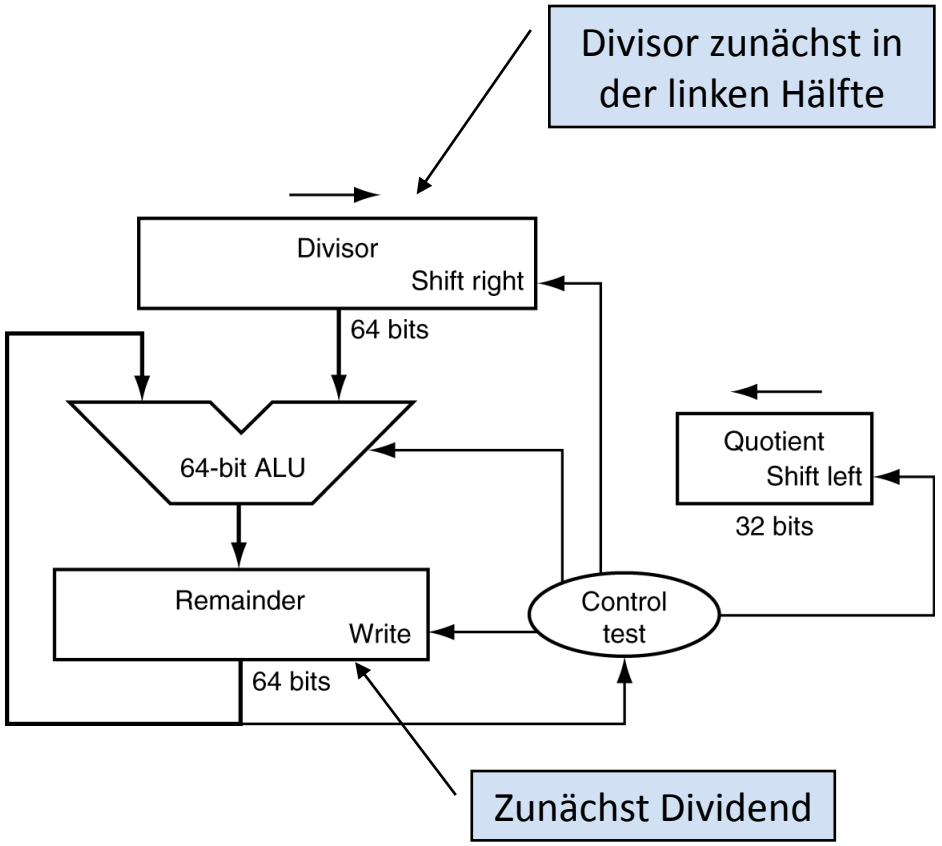
  $q_i = 0$ und Korrektur durch $a = a + b$, falls $a - b < 0$
  - Dieses Verfahren wird auch als „Restoring“- Division bezeichnet
- Division mit Vorzeichen
  - Division mit absoluten Werten
  - Vorzeichenanpassung bei Quotienten und Rest

Beispiel


Dividend (dividend)
Divisor (divisor)
Quotient (quotient)
1001010 : 1000 = 1001
-1000
10
101
1010
-1000
10
Rest (remainder)

Hardware für Division





Beispiel (0111:0010)

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Division in MIPS

- In MIPS werden zwei 32-Bit Zahlen aus zwei Registern dividiert

```
div reg1,reg2          # vorzeichenbehaftete Division
divu reg1,reg2         # vorzeichenlose Division
```

  - Quotient befindet sich nach Ausführung der Division im Spezialregister Lo, der Rest im Spezialregister Hi
- Keine Überprüfung auf Division durch 0
- Weiterverarbeitung erfolgt wie bei der Multiplikation mit den Befehlen **`mfhi`** und **`mflo`**
- Der Assembler realisiert zusätzlich den Pseudobefehl

```
div reg3,reg1,reg2     # Quotient nach reg3
```

ZUSAMMENFASSUNG

Zusammenfassung

- Digitale Grundlagen
  - Gatter
  - Schaltnetze
  - Karnaugh-Veitch-Diagramme
  - Spezielle Schaltnetze
- Arithmetik
  - Überlauf
  - Halbaddierer, Volladdierer
  - ALU
  - Multiplikation, Division

Literatur

- Grundliteratur
  - D. A. Patterson, J. L. Hennessy: **Computer Organization and Design**, 5. Auflage, Morgan Kaufmann, 2013 – Kapitel 3
  - H. Herold, B. Lurz, J. Wohlrab, **Grundlagen der Informatik**, 2. Auflage, Pearson Studium, 2012 – Kapitel 13
  - D. W. Hoffmann: **Grundlagen der Technischen Informatik**, 4. Auflage, Hanser, 2014 – Kapitel 5 - 7