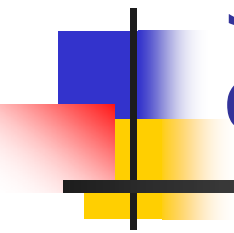


3. Mikroalgorithmen und Rechenwerke für die Grundrechenarten

Technische Grundlagen der Informatik 2
(Rechnertechnologie 2)
SS 2006

Andreas Koch
FG Eingebettete Systeme und ihre Anwendungen

Auf Basis von Material von

Rolf Hoffmann
FG Rechnerarchitektur
Technische Universität Darmstadt

3.1 Addition

- **Addition von Dualzahlen**
  - Meist ziffernweise unter Berücksichtigung der Überträge
  - Addition der Ziffern durch **Halbaddierer** (HA) und **Volladdierer** (VA)
  - Addierschaltnetze durch Hintereinanderschaltung von HA/VA
  - Beschleunigung durch Carry-Look-Ahead

- Beispiele für die Addition von Dualzahlen

```
  olllolol = 117
  ololollo =  86
  lll l          Überträge
  lloololl = 203

  llllllll = 255
  oooooool =   1
 looooooo = 256 (Überlauf)

  llllllll = 255
  llllllll = 255
 lllllllo = 510 (Überlauf)
```

Dieses Bit wird als Carry(-Out)

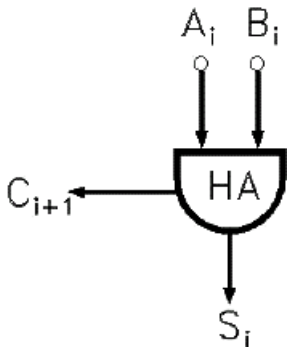
bezeichnet. Es wird l, wenn das Ergebnis nicht mehr mit n Stellen darstellbar ist

Halbaddierer (half adder)

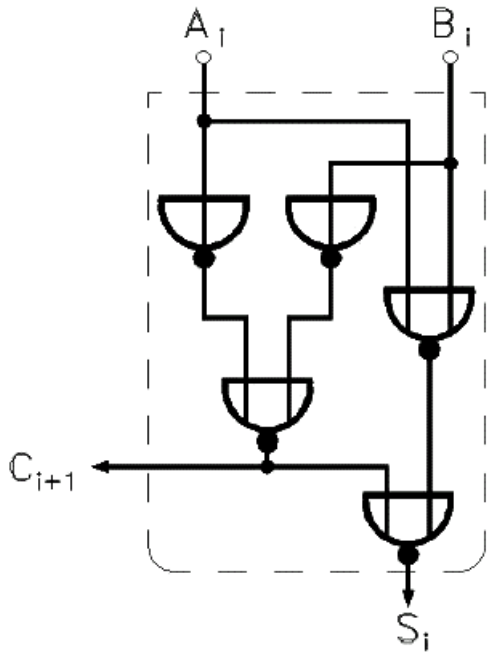
- Bildet die Summe von **zwei** Dualziffern A_i und B_i
- Summe kann die Werte 0,1,2 annehmen
- Summe wird codiert durch das Summenbit S_i und das Übertragsbit C_{i+1}
- $C_{i+1}\_ S_i = A_i + B_i$
- $S_i = A_i \oplus B_i = A_i \,.\, \sim B_i \; v \; \sim A_i . B_i$
- $C_{i+1} = A_i \,.\, B_i$

A_i B_i	C_{i+1} S_i	Summe
o o	o o	0
o l	o l	1
l o	o l	1
l l	l o	2

Halbaddierer



Schaltzeichen



Eine mögliche Realisierung

mit NOR-Gattern

Volladdierer

- Bildet die Summe von **drei** Dualziffern A_i , B_i und C_i
- Summe kann die Werte 0,1,2,3 annehmen
- Summe wird codiert durch das Summenbit S_i und das Übertragsbit C_{i+1}
- $C_{i+1}\_S_i = A_i + B_i + C_i$

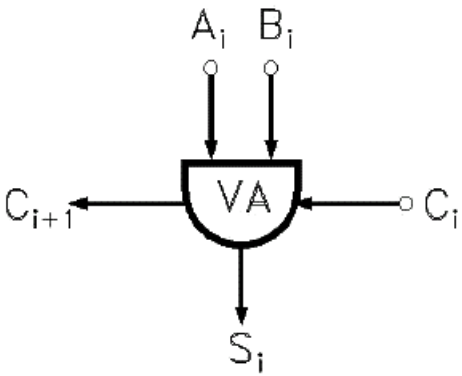
- $S_i = A_i \oplus B_i \oplus C_i$
- $C_{i+1} = A_i.B_i \vee A_i.C_i \vee B_i.C_i$

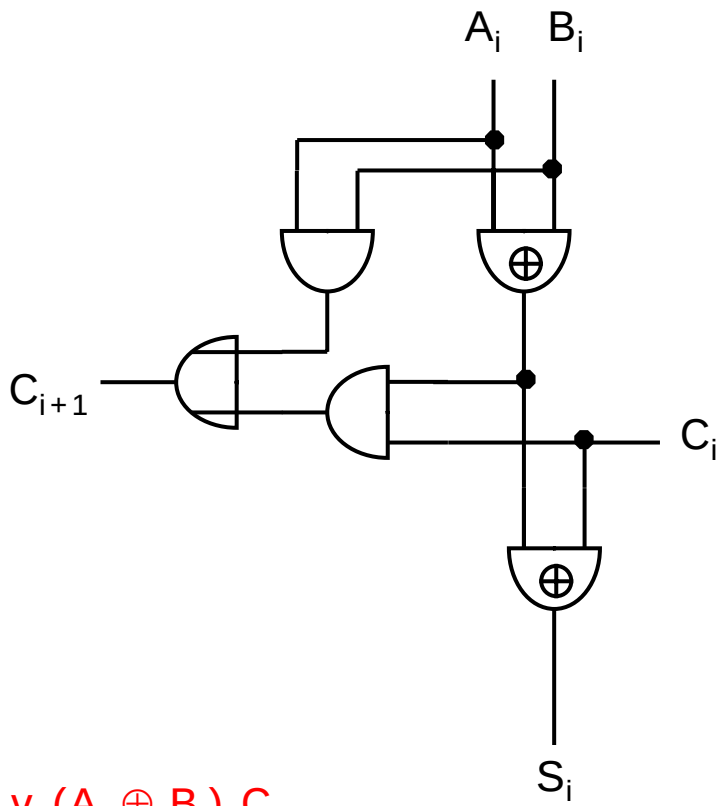
  $C_{i+1} = A_i.B_i \vee (A_i \vee B_i).C_i$

  $C_{i+1} = A_i.B_i \vee (A_i \oplus B_i).C_i$

A_i	B_i	C_i	C_{i+1}	S_i	Summe
o	o	o	o	o	0
o	o	l	o	l	1
o	l	o	o	l	1
o	l	l	l	o	2
l	o	o	o	l	1
l	o	l	l	o	2
l	l	o	l	o	2
l	l	l	l	l	3

Volladdierer

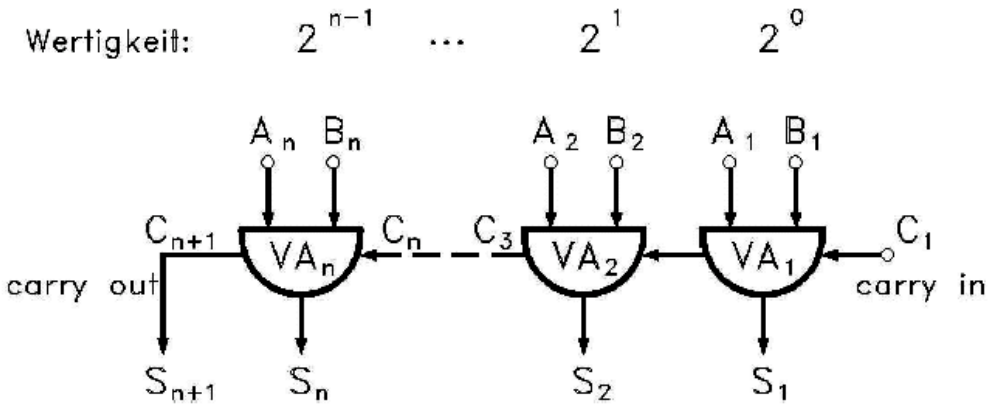




$$C_{i+1} = A_i.B_i \ \text{v} \ (A_i \oplus B_i).C_i$$

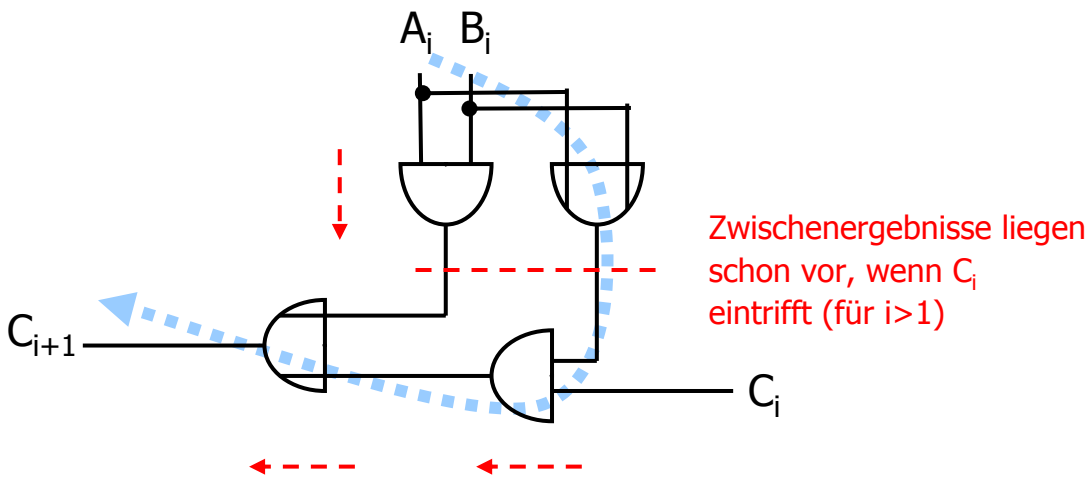
Addierer mit Übertragsweiterleitung

- engl. **Carry-Ripple-Adder**
- Additionsschaltnetz, das die Addition für zwei Dualzahlen mit n Bits durchführen kann.
- Hintereinanderschaltung von n Volladdierern, Verbindung über das Carry
- **Carry-Out** wird 1, wenn die Summe nicht mehr durch n Bits darstellbar ist (Überlauf bei unsigned number)
- **Carry-In**
  - = o bei normaler Addition
  - = l bei der Subtraktion
  - = altes Carry-Out bei der Doppelwort-Addition

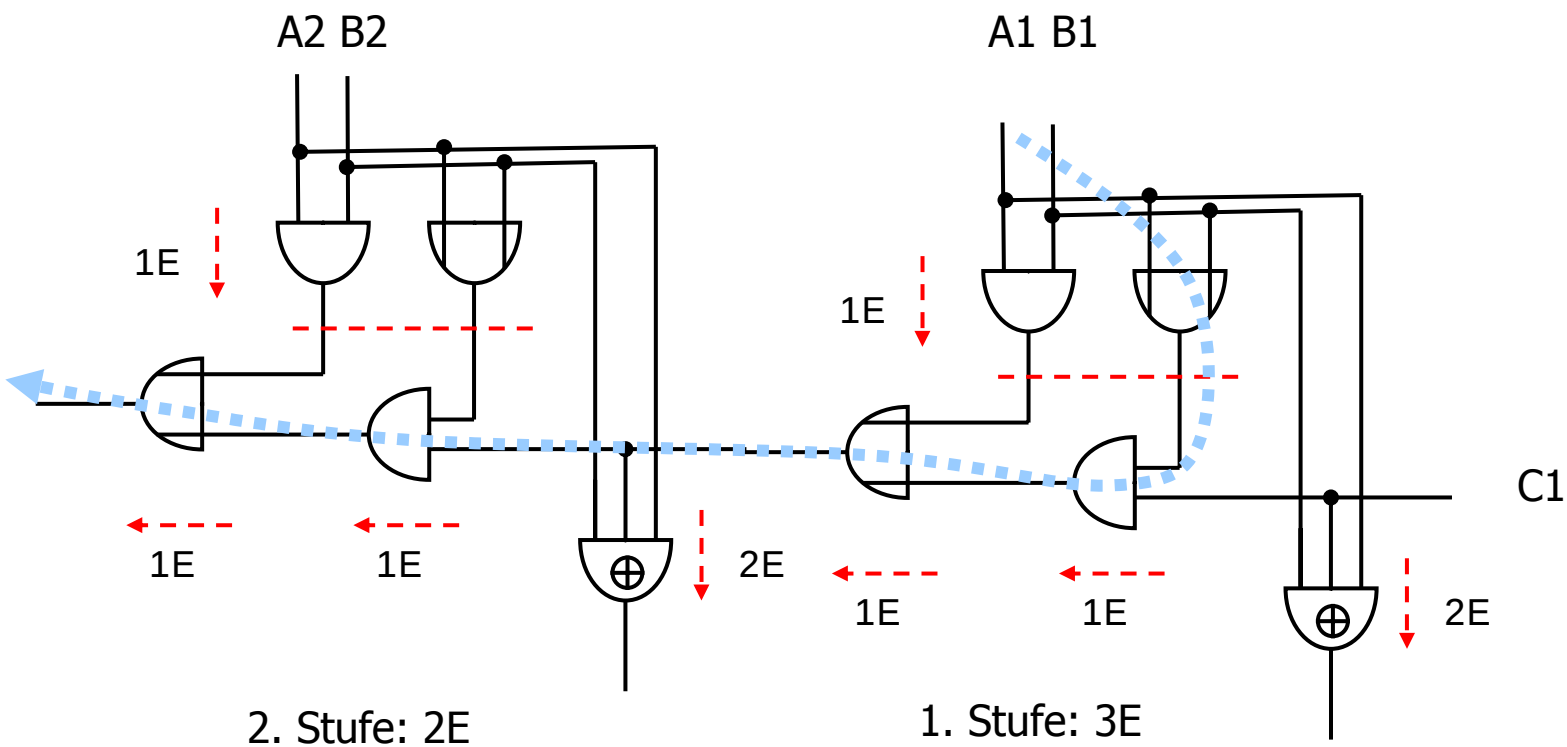


Rechenzeit / Verzögerungszeit

- Zu betrachten ist der längste Pfad
- Annahme: Jedes Gatter verursacht die Verzögerung 1E (eine Einheit)
- $C_{i+1} = A_i.B_i \ v \ A_i.C_i \ v \ B_i.C_i = A_i.B_i \ v \ (A_i \ v \ B_i).C_i$
  - T_1=3E für die erste Stufe, -- zusätzlich t_i=2E für alle weiteren
  - T_i = 3E + 2E(i-1) -- Rechenzeit für das Carry i
- Die Rechenzeit steigt linear mit der Stellenzahl n



Berechnung der Rechenzeiten



Addierer mit Übertragsvorausberechnung

- engl. **Carry-Look-Ahead**
- **Ziel**: Beschleunigung der Addition durch schnellere Berechnung der Überträge
- **Ansatz**: Gleichungen für C_i ineinander einsetzen und ausmultiplizieren, so daß im Extremfall die Logik nur noch zweistufig ist. (hier dreistufig)
- **C_2** = $\underbrace{A_1.B_1}_{G_1}$ v $(\underbrace{A_1 \text{ v } B_1}_{P_1}).C_1$ //T_2 = 3E für den ersten Übertrag
- $C_3 = A_2.B_2$ v $(A_2 \text{ v } B_2)$.**C_2** // T_3 = 3E +2E für zweiten Übertrag bei Weiterverwendung von C2

$$C_3 = \underbrace{A_2.B_2}_{G_2} \vee \underbrace{(A_2 \vee B_2)}_{P_2}.\underbrace{A_1.B_1}_{G_1} \vee \underbrace{(A_2 \vee B_2)}_{P_2}.\underbrace{(A_1 \vee B_1)}_{P_1}.C_1$$

Hilfsfunktionen
G=Carry Generate
(neuer Übertrag)
P=Carry Propagate
(Weiterleiten)

$C_3 = G_2$ v $P_2.G_1$ v $P_2.P_1.C_1$

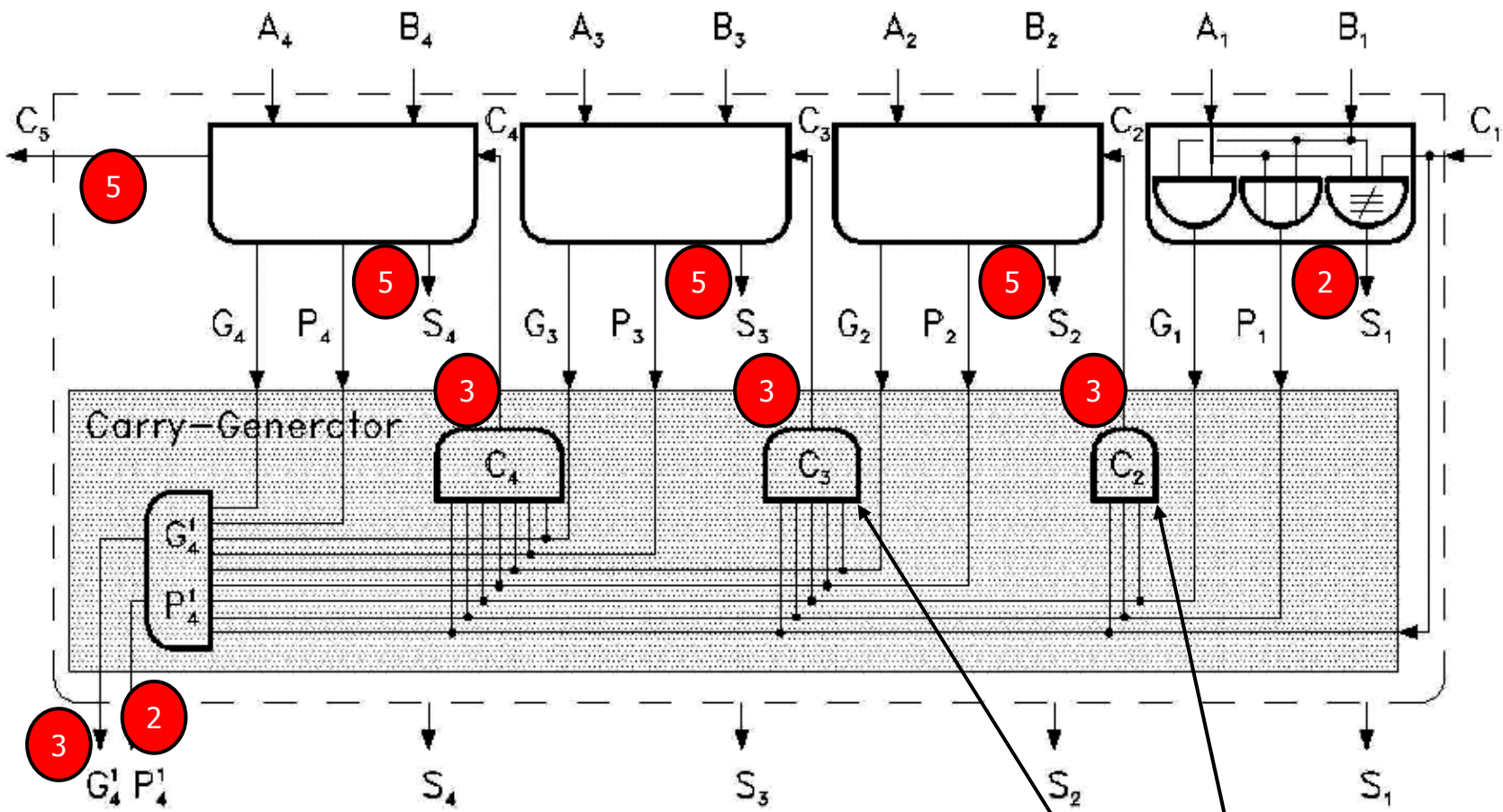
T_3 = 3E für zweiten Übertrag nach dem Einsetzen von C2, geht jetzt schneller

Rechenzeit

- 4-Bit-Addierer mit Übertragvorausberechnung
  - Annahme: Die Berechnung des Summenbits 2E dauert (3-fach EXOR-Gatter).
  - Da die Berechnung von S_4 5E dauert, braucht der Übertrag C_5 nicht schneller berechnet zu werden, es kann die vereinfachte Funktion (3.6) verwendet werden.

	Gesamtrechenzeit	
$C_2 = G_1 \vee P_1 C_1$	$3\,E$	(3.3)
$C_3 = G_2 \vee P_2 G_1 \vee P_2 P_1 C_1$	$3\,E$	(3.4)
$C_4 = G_3 \vee P_3 G_2 \vee P_3 P_2 G_1 \vee P_3 P_2 P_1 C_1$	$3\,E$	(3.5)
$C_5 = G_4 \vee P_4 C_4$	$\mathbf{5\,E}$	(3.6)
$S_1 = A_1 \not\equiv B_1 \not\equiv C_1$	$2\,E$(für ex.oder)	(3.7)
$S_2 = A_2 \not\equiv B_2 \not\equiv C_2$	$5\,E$	(3.8)
$S_3 = A_3 \not\equiv B_3 \not\equiv C_3$	$5\,E$	(3.9)
$S_4 = A_4 \not\equiv B_4 \not\equiv C_4$	$5\,E$	(3.10)

4-Bit-Addierer mit Übertragsvorausberechnung



Realisierungsaufwand

- **Carry-Ripple**
  - $C_{i+1}=G_i \vee P_i.C_i$
  - Anzahl der Logik-Eingänge R:
    R = 2(für G_i) + 2(für P_i) + 2(für AND) + 2(für OR) = 8
- für n Stufen: $R_{ges} = 8n = \mathbf{O(n)}$

- **Carry-Look-Ahead**
  - 1. Stufe: $R^1=8$
  - 2. Stufe: $R^2=12$
  - 3. Stufe: $R^3=17$
  - 4. Stufe: $R^4=23$
- allgemein für k-te Stufe
  $R^k = 8 + 4 + 5 + 6 +...$
  $= 8 + 3(k-1) + (k(k-1)/2)$
- Gesamt-Eingänge
  R_{ges}=Summe R^k k=1 bis n
- für n=1,2,3,...:
- R_{ges}=8, 20, 37, 60, ...

$$C_2 = G_1 \vee P_1 C_1$$
$$C_3 = G_2 \vee P_2 G_1 \vee P_2 P_1 C_1$$
$$C_4 = G_3 \vee P_3 G_2 \vee P_3 P_2 G_1 \vee P_3 P_2 P_1 C_1$$

4-Bit-Carry-Generator

- berechnet die Überträge voraus
- berechnet **zusätzlich für die nächst höhere Ebene**
  - Generate G^1_4 und Propagate P^1_4

$$C_2 = G_1 \vee P_1 C_1$$
$$C_3 = G_2 \vee P_2 G_1 \vee P_2 P_1 C_1$$
$$C_4 = G_3 \vee P_3 G_2 \vee P_3 P_2 G_1 \vee P_3 P_2 P_1 C_1$$

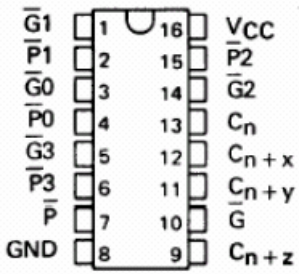
$$G^1_4 = G_4 \vee P_4 G_3 \vee P_4 P_3 G_2 \vee P_4 P_3 P_2 G_1$$ Übertragserzeugung 1. Ordnung

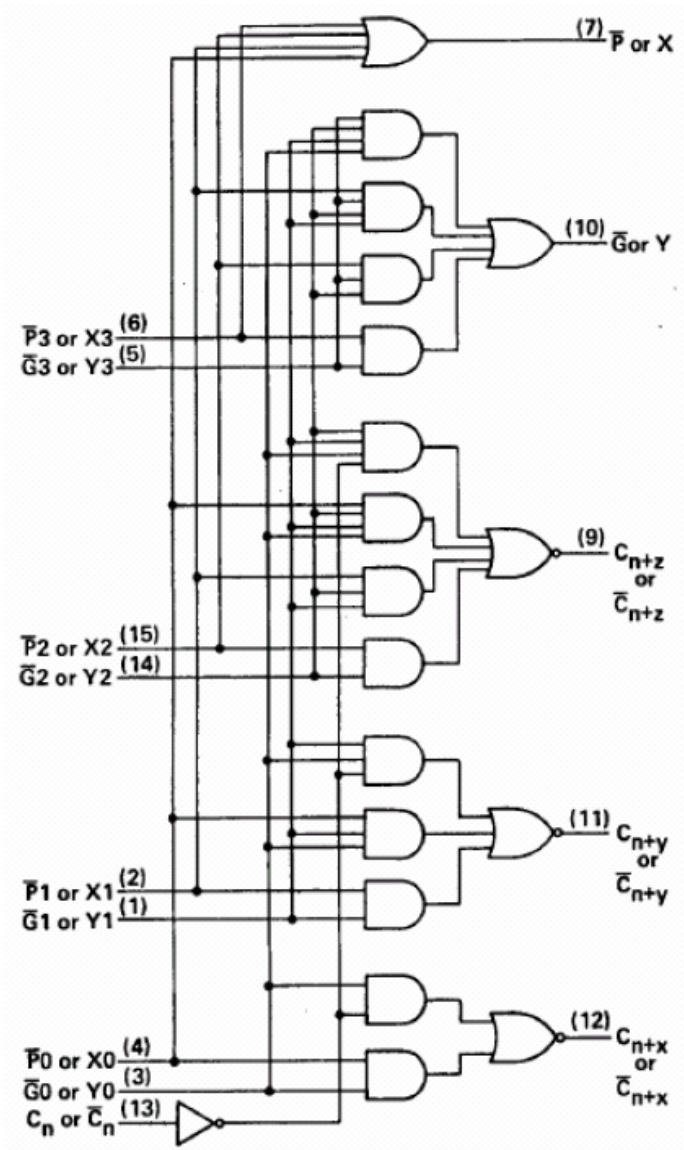
$$P^1_4 = P_4 P_3 P_2 P_1$$ Übertragsweiterleitung 1. Ordnung

Carry-Look-Ahead-Baustein

TEXAS INSTRUMENTS

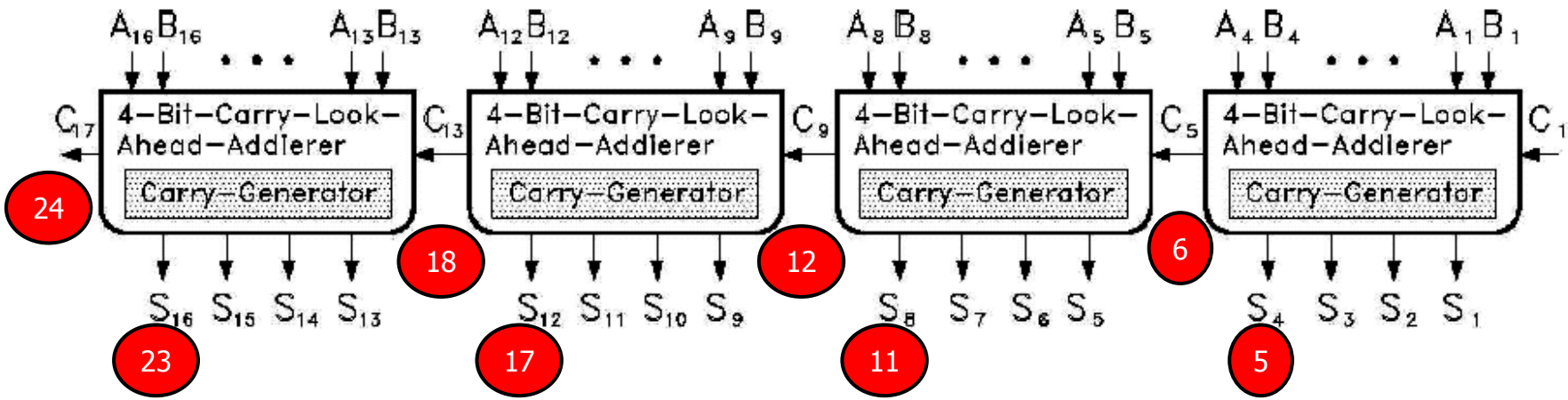
SN54S182 . . . J OR W PACKAGE
SN74S182 . . . D OR N PACKAGE
(TOP VIEW)





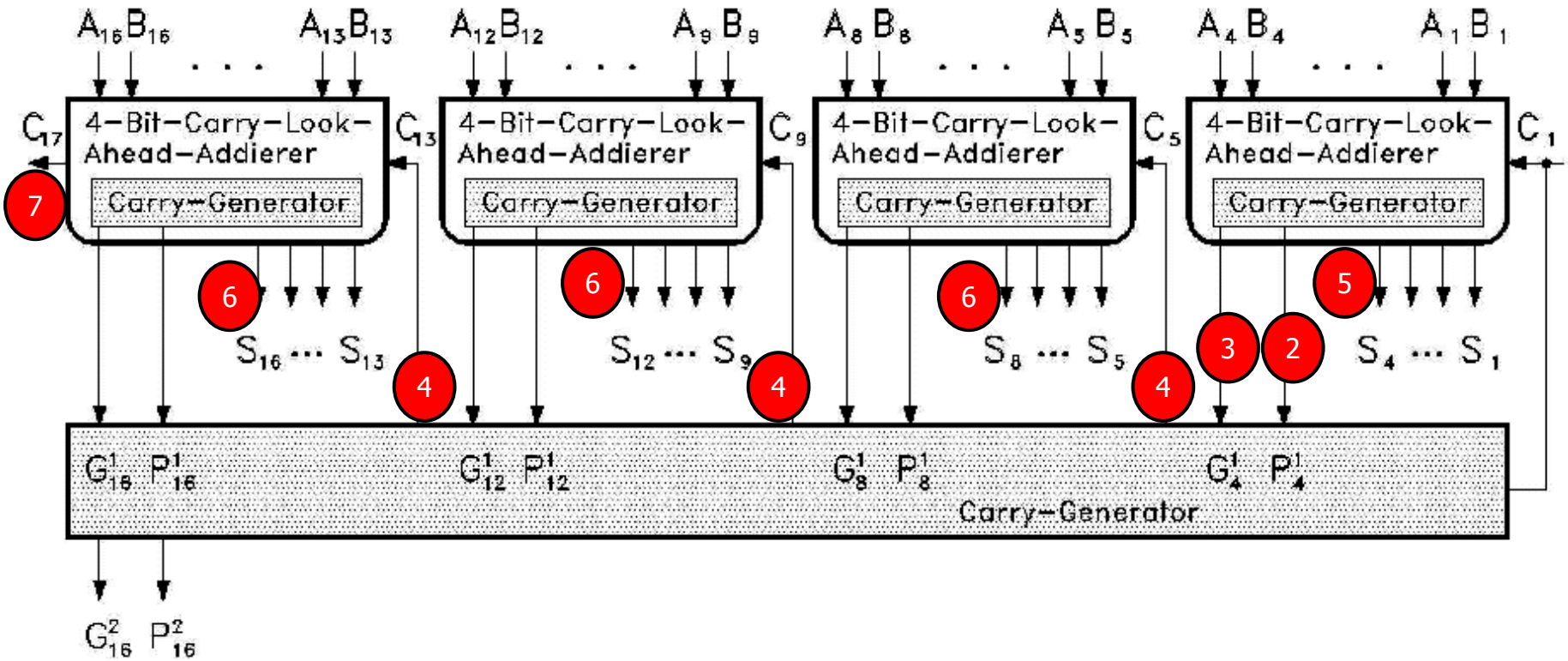
16-Bit-Addierer

- 16-Bit-Addierer bestehend aus vier 4-Bit-Addierern mit Übertragsvorausberechnung
- Zwischen den Stufen Carry-Ripple-Technik



Vergleich: Ripple-Carry 1*3 + 15*2 = 33

16-Bit-Addierer mit zwei Carry-Generator-Ebenen



3.1.2 Subtraktion allgemein
A) Subtraktion positiver Dualzahlen

- **1. Möglichkeit**: Verwendung von Vollsubtrahierern
- C_{i+1} $D_i = A_i - B_i - C_i$
- Differenz kann die Werte 1, 0, -1, -2 annehmen
- Borgebit (borrow) C_{i+1} hat die Wertigkeit $-(2^i)$
- $D_i = A_i \oplus B_i \oplus C_i$
- $C_{i+1} = \sim A_i.B_i \vee \sim A_i.C_i \vee B_i.C_i$
- Entsteht ein Carry-Out, so war B>A (Für Vergleichszwecke genügt es, nur das Carry-Out zu berechnen)

A_i	B_i	C_i	C_{i+1}	D_i	Differenz
o	o	o	o	o	0
o	o	l	l	l	-1
o	l	o	l	l	-1
o	l	l	l	o	-2
l	o	o	o	l	1
l	o	l	o	o	0
l	l	o	o	o	0
l	l	l	l	l	-1

```
 olol=5
-oolo=2
- l    borrow
 ooll=3
```

```
  olol=5
- olll=7
- ll    borrow
-l      borrow-out = -16
  lllo  =14
(-16 + 14 = -2)
```

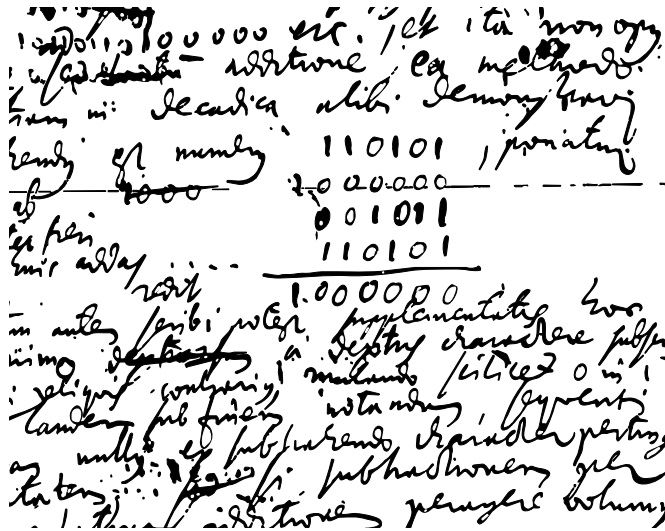
Die Subtraktion ergibt in diesem Beispiel ein Ergebnis, das nicht darstellbar ist (negative Zahlen). Bereichsunterschreitung.

Erkennbar am Carry-out=borrow-out=1. (→ (7>5))

(Das Ergebnis mit n+1 Stellen wäre interpretiert als 2K-Zahl korrekt)

Leibniz: Supplementzahlen zur Subtraktion

... Es ist auch zu beachten, daß für eine zu subtrahierende Zahl ihr Supplement auf looooo usw. eingesetzt werden kann, und dann braucht man keine Subtraktion sondern nur die Addition, nach dem Verfahren, das ich auch für das dekadische System anderswo dargelegt habe.



Beispiel: Zu subtrahieren ist die Zahl: 110101

Angenommen, sie sei zu subtrahieren von: 1000000

dann ergibt das offenbar: 001011

Denn wenn man dazu addiert: 110101

ergibt es wieder: 1000000

Man kann aber sofort solche Supplementzahlen hinschreiben, wenn man unter der ersten Ziffer rechts die ursprüngliche, bei den übrigen aber jeweils die gegenteilige (Ziffer des Subtrahenden) einsetzt, also 0 in 1 verwandelt. Schließlich ist am Ende zu beachten, daß von der folgenden Spalte, bis zu der keine Ziffer des Subtrahenden hinreicht, eine Einheit zu subtrahieren ist, wenn wir die Subtraktion durch bloße Addition mit Hilfe von Supplementen durchführen wollen.

Subtraktion positiver Dualzahlen

- **2. Möglichkeit:** Normaler Addierer, Addition des 2-Komplements von B
- Es gilt B + ~B = 2^n -1 →
  -B = ~B + 1 - 2^n
- A – B = (A + ~B + 1) –2^n
  - Wenn (A + ~B + 1) einen Übertrag 2^n erzeugt, so wird er auf Null gesetzt, was der geforderten Korrektur um –2^n entspricht. Das Ergebnis ist positiv.
- Wenn (A + ~B + 1) keinen Übertrag 2^n erzeugt, so ist ebenfalls die Korrektur –2^n anzubringen, was dem Setzen des Übertrags (Borrowbit) auf 1 entspricht. Das Ergebnis ist als negativ zu werten. (In diesem Fall war B>A, was für Vergleichszwecke benutzt werden kann)

```
5=  olol   -00lo =-2
    llol   negiert
    oool   +1
   looll   -2^n ergibt
   oooll   =3
```

```
5=  olol - olll =7
    looo
    oool
   olllo  negativ
  (llllo)
 (-16 +14 = -2)
```

Ergebnis ist nicht mehr darstellbar (negativ geworden).

Durch die Negation des Carry-Outs ergibt sich das gleiche Ergebnis wie bei der 1. Möglichkeit der Subtraktion (Verwendung eines Vollsubtrahierers).

Ergebnis im 2K mit n+1 Stellen wäre dann korrekt.

B) Subtraktion von 2K-Zahlen

- Es soll **A – B im 2K** gebildet werden.
- Entspricht der **Addition des Komplements** A + B´ .
- Verwendet wird ein Addierer für 2K-Zahlen
  - Wie noch gezeigt wird, kann dafür ein normaler Addierer für Dualzahlen benutzt werden.
- Das **Komplement wird durch Negieren aller Bits von B und Setzen des Carry-In-Eingangs auf l** bewerkstelligt.

3.1.4 Addition von Zweikomplementzahlen

- **Gegeben**: 2K-Zahlen X[n] und Y[n]
  - Schreibweise X[n]=X_n.. X_1
- **Gesucht**: Summe S[n] oder S[n+1] bei Überlauf
- **Lösungsweg**: Verwendung der Rückabbildungsgleichung **$x=X[n]-X_n 2^n$**
- Betrachtung aller möglichen Fälle für X_n , Y_n , S_n , S_{n+1}

$$s = x + y = X[n] - X_n * 2^n + Y[n] - Y_n * 2^n$$

$$s = S[n] - S_n * 2^n$$
$$s = S[n+1] - S_{n+1} * 2^{n+1}$$

$$\begin{aligned} S[n] &= X[n] + Y[n] \quad \text{„kein Überlauf“} \\ &\quad + (S_n - X_n - Y_n) * 2^n \\ S[n+1] &= X[n] + Y[n] \quad \text{„Überlauf“} \\ &\quad + (2S_{n+1} - X_n - Y_n) * 2^n \, . \end{aligned}$$

6 verschiedene Fälle

Fall	X_n	Y_n	S_{n+1}	S_n	Summe	
a1	o	o	(o)	o	$S[n] = X + Y$	
a2	o	o	o	l	$S[n+1] = X + Y$	„Überlauf“
b1	l	l	(l)	l	$S[n] = X + Y - 2^n$	
b2	l	l	l	o	$S[n+1] = X + Y$	„Überlauf“
c1	o	l	(l)	o	$S[n] = X + Y - 2^n$	
c2	o	l	(o)	l	$S[n] = X + Y$	

- Zusammenfassung der Fälle
  - 1. S[n] = X + Y bilden und den Übertrag S_{n+1} merken. Die Überlaufbedingung berechnen.
  - 2. Wenn kein Überlauf aufgetreten ist (OV=o), dann ist die Summe gleich S[n], ansonsten gleich S[n + 1].

2K-Additionsbeispiele

			C	OV
a1)	+2	o.o l o		
	+2	o.o l o		
	+4	o.l o o	o	o

			C	OV
a2)	+7	o.l l l		
	+7	o.l l l		
	+14	o.l l l o	o	l

bei 4 Bit: –2

			C	OV
b1)	-1	l.l l l		
	-1	l.l l l		
	-2	l l.l l o	l	o

			C	OV
b2)	-7	l.o o l		
	-7	l.o o l		
	-14	l.o o l o	l	l

bei 4 Bit: +2

			C	OV
c1)	+2	o.o l o		
	-1	l.l l l		
	+1	l o.o o l	l	o

			C	OV
c2)	+1	o.o o l		
	-7	l.o o l		
	-6	o l.o l o	o	o

In den Fällen b1 und c1 (kein OV) wird C=Carry-Out abgezogen (ignoriert)

In den Fällen a2 und b2 (OV) gibt C das Vorzeichen von S[n+1] an

Überlaufbedingung (Overflow)

- Carry-Out $C_n = S_{n+1}$
- Der Überlauf kann auf drei verschiedene Weisen berechnet werden:
  - 1. Wenn die Vorzeichen von X_n und Y_n gleich sind und das Ergebnis-Vorzeichen S_n sich davon unterscheidet
    - $OV = (X_n Y_n S_n = ool) \vee (X_n Y_n S_n = llo)$
  - 2. Wenn die Vorzeichen X_n und Y_n gleich und die beiden höchstwertigen Summenbits $S_{n+1} S_n$ ungleich sind
    - **$OV = (X_n \equiv Y_n) \cdot (S_{n+1} \oplus S_n)$**
  - 3. Wenn Carry-Out und der vorhergehende Übertrag ungleich sind
    - **$OV = (C_{n+1} \oplus C_n)$**

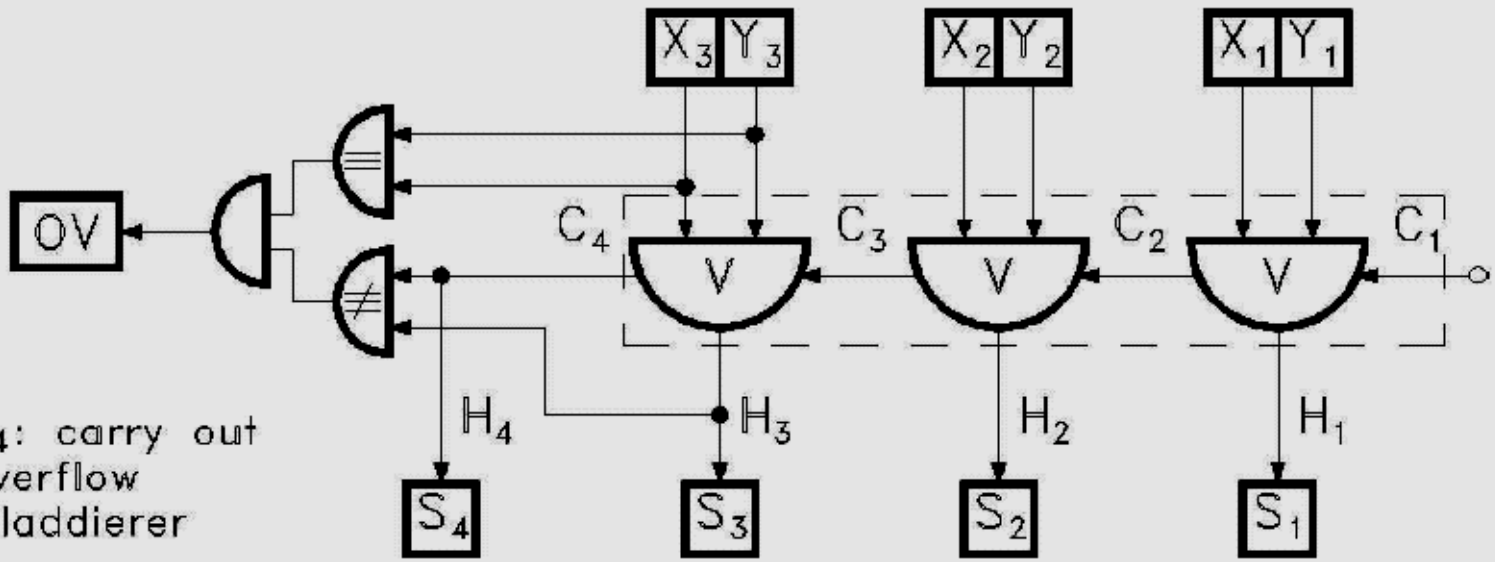
Rechenwerk für 2K-Zahlen


```
register X[n], Y[n], S[n+1], OV;
signal H[n+1];
perm H == X++Y pend "Additionsschaltnetz"
on clock

[1]   S←H, OV←(X_n ≡ Y_n).(H_n+1 ≢ H_n), next 2

[2]   "if OV=o then Summe = S[n]
              else Summe = S[n+1] fi"

noc
```

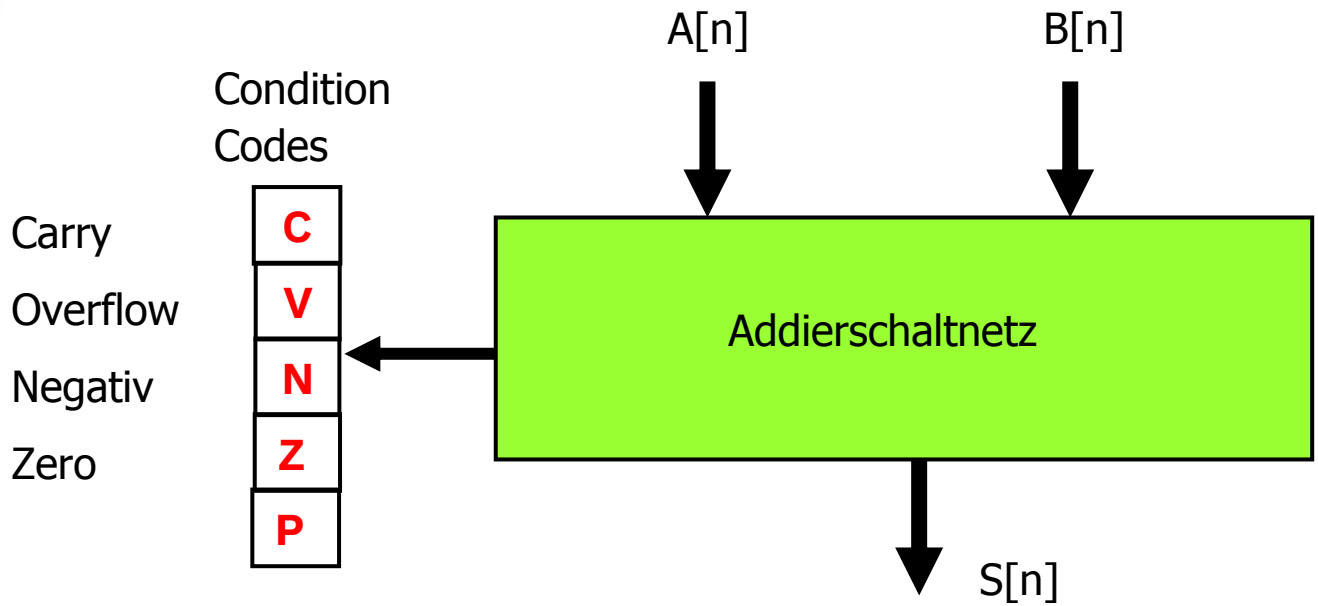


Vergleich unsigned / signed Addition

- Ein Addierschaltnetz kann positive Dualzahlen (unsigned) addieren
  - Wenn Carry-Out=1, dann Ergebnis nicht mehr mit n Stellen darstellbar (Überlauf).

- Das Addierschaltnetz für positive Dualzahlen kann auch 2K-Zahlen (signed) addieren
  - Überlaufbedingung OV muß jetzt anders (vorher genannte Formeln) berechnet werden.
  - Wenn OV=1, dann Ergebnis nicht mehr durch n Stellen darstellbar. Carry-Out gibt in diesem Falle die zusätzliche Stelle S_{n+1} an.

Typisches Rechenwerk für Dualzahlen und 2K-Zahlen



- **C = Carry-Out**, zeigt Überlauf für vorzeichenlose Dualzahlen an oder ist gleich S_{n+1}, wenn Überlauf bei der 2K-Addition aufgetreten ist.
- **V = Overflow** bei der 2K-Addition *("Vorzeichen kaputt")*.
- **Negativ N=S_n**, zeigt eine negative 2K-Zahl an, wenn kein Überlauf aufgetreten ist.
- **Zero (Z=1)** = (S=0), zeigt an, dass die Summe Null geworden ist, wenn kein Überlauf aufgetreten ist.
- **P = Parity** wird gesetzt, wenn die Anzahl der Einsen in S ungerade ist.

Vergleich von Zahlen

- Der Vergleich von Dualzahlen und 2K-Zahlen wird oft (z.B. Motorola 68000) wie folgt durchgeführt:
  - **1. Subtraktion A – B = A + ~B +1**
    - Berechnen und Speichern der Condition Codes
  - **2. Auswerten der Condition Codes**
    - z. B. durch einen bedingten Sprungbefehl

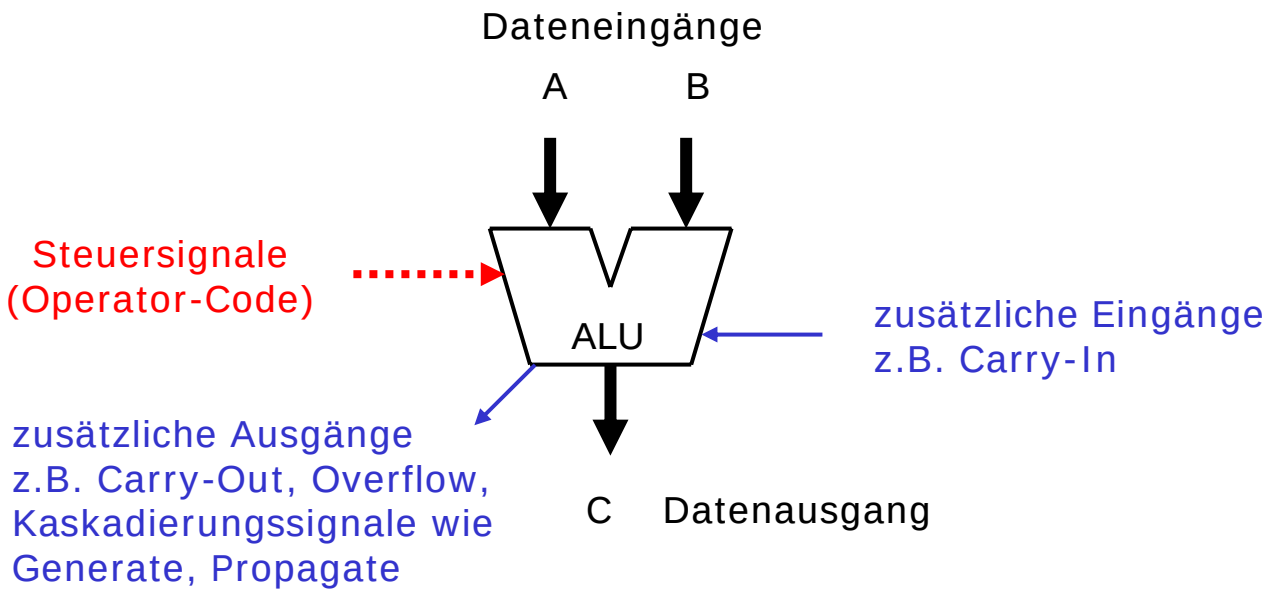
Auswertung der Condition Codes

Befehle

BGT	greater than (signed) >	Z = 0 and N = V
BGE	greater or equal (signed) ≥	N = V
BLE	less or equal (signed) ≤	Z = 1 or N ≠ V
BLT	less than (signed) <	N ≠ V
BPL	plus	N = 0
BMI	minus	N = 1
BHI	higher (unsigned) >	Z = 0 and C = 0
BHS = BCC	higher or same (unsigned) ≥	C = 0
BLS	lower or same (unsigned) ≤	Z = 1 or C = 1
BLO = BCS	lower (unsigned) <	C = 1
BEQ	equal =	Z = 1
BNE	not equal ≠	Z = 0
BVS	overflow set	V = 1
BVC	overflow clear	V = 0
BCS	carry set	C = 1
BCC	carry clear	C = 0

ALU = Arithmetische u. Logische Unit

- engl. **A**rithmetical and **L**ogical **U**nit
- Darunter versteht man (in seiner einfachen Form) ein Schaltnetz, das Operationen wie **+, –, and, or** berechnen kann.



ALU ´181 - Funktionen

Dieser ALU-Baustein kann 16 logische Funktionen (Mode Control=l) oder 16 arithmetische Funktionen (Mode Control=o) ausführen.

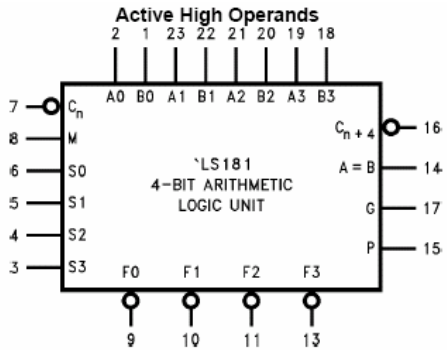
FAIRCHILD SEMICONDUCTOR™

DM74LS181
4-Bit Arithmetic Logic Unit

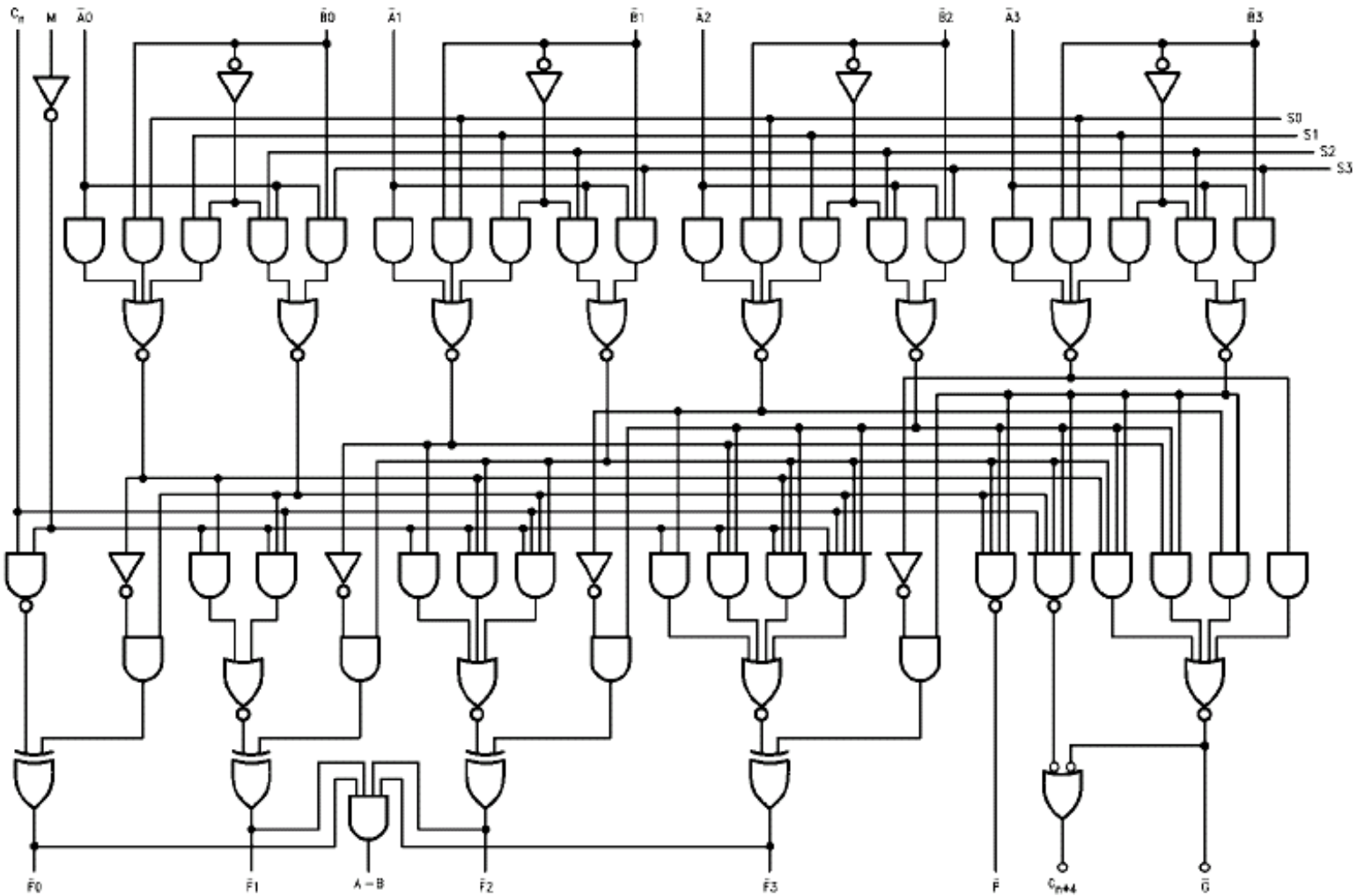
Mode Select Inputs				Active LOW Operands & F_n Outputs		Active HIGH Operands & F_n Outputs	
				Logic	Arithmetic (Note 2)	Logic	Arithmetic (Note 2)
S3	S2	S1	S0	(M = H)	(M = L) (C_n = L)	(M = H)	(M = L) (C_n = H)
L	L	L	L	$\overline{A}$	A minus 1	$\overline{A}$	A
L	L	L	H	$\overline{AB}$	AB minus 1	$\overline{A}+\overline{B}$	A + B
L	L	H	L	$\overline{A}+B$	$A\overline{B}$ minus 1	$\overline{A}B$	$A+\overline{B}$
L	L	H	H	Logic 1	minus 1	Logic 0	minus 1
L	H	L	L	$\overline{A}+\overline{B}$	A plus (A + $\overline{B}$)	$\overline{AB}$	A plus $A\overline{B}$
L	H	L	H	$\overline{B}$	AB plus (A + $\overline{B}$)	$\overline{B}$	(A + B) plus $A\overline{B}$
L	H	H	L	$\overline{A}\oplus\overline{B}$	A minus B minus 1	$A\oplus B$	A minus B minus 1
L	H	H	H	$A+\overline{B}$	$A+\overline{B}$	$A\overline{B}$	AB minus 1
H	L	L	L	$\overline{A}B$	A plus (A + B)	$\overline{A}+B$	A plus AB
H	L	L	H	$A\oplus B$	A plus B	$\overline{A}\oplus\overline{B}$	A plus B
H	L	H	L	B	$A\overline{B}$ plus (A + B)	B	(A + $\overline{B}$) plus AB
H	L	H	H	A + B	A + B	AB	AB minus 1
H	H	L	L	Logic 0	A plus A (Note 1)	Logic 1	A plus A (Note 1)
H	H	L	H	$A\overline{B}$	AB plus A	$A+\overline{B}$	(A + B) plus A
H	H	H	L	AB	$A\overline{B}$ minus A	A + B	(A + $\overline{B}$) plus A
H	H	H	H	A	A	A	A minus 1

Note 1: Each bit is shifted to the next most significant position.

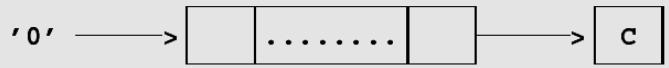
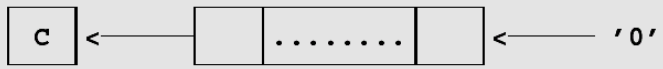
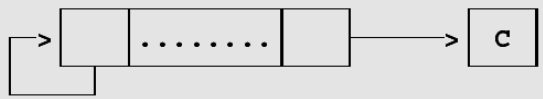
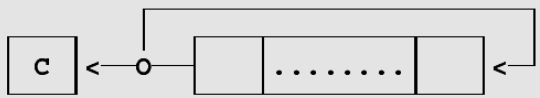
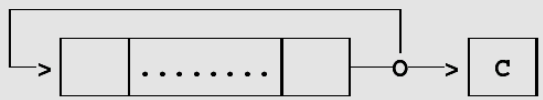
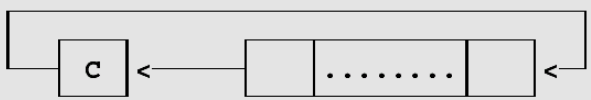
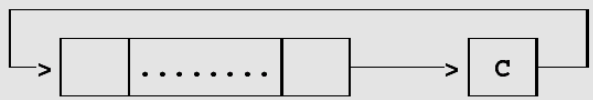
Note 2: Arithmetic operations expressed in 2s complement notation.



ALU ´181 - Logik



Shift-Operationen

SHL	Shift left	
SHR	Shift right	
SAL	Shift arithmetic left	
SAR	Shift arithmetic right	
ROL	Rotate left	
ROR	Rotate right	
RCL	Rotate left through carry	
RCR	Rotate right through carry	

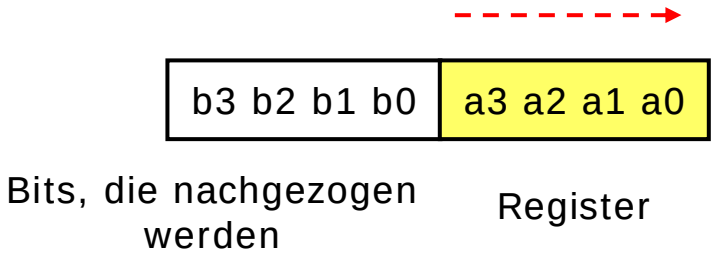
Typische Schiebebefehle in Mikroprozessoren (Intel 80x86)

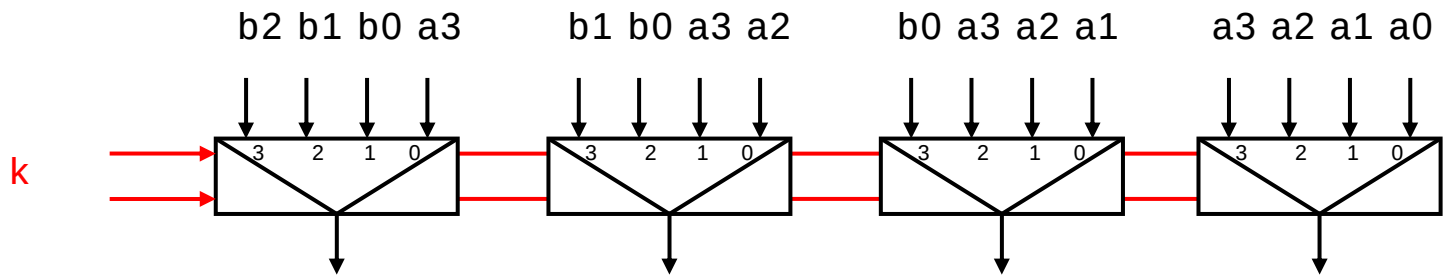
C dient hier zum Retten bzw. Wiedereinspeisen des herausgeschobenen Bits.

http://www.tu-chemnitz.de/informatik/ce/Lehre/DIGT/Nau17.pdf

Barrelshifter, Ringshifter

- Shift um k Stellen in einem Zeitschritt
- Eine Realisierungsmöglichkeit mit Multiplexern:





3.1.6 Doppelwort-Addition

- **Gegeben**: Addition mit der Wortlänge n
- **Gesucht**: Addition mit der Wortlänge 2n
  - X = X2_X1 plus Y = Y2_Y 1

- **Lösung**:
  - boole OV, (X2, X1, Y2, Y1)[n], (S2, S1)[n+1];
  - S1 := X1 ++ Y1; "$S1_{n+1}$=CARRY1, ++ Addition mit Übertrag"
  - S2 := X2 ++ Y2 + $S1_{n+1}$; "$S2_{n+1}$=CARRY2"
  - OV := ($X2_n \equiv Y2_n$).($S2_{n+1} \oplus S2_n$)
  - if OV = o then *Summe* = S2[n]_S1[n]
    else *Summe* = S2[n+1]_S1[n] fi

3.2 Multiplikation Leibniz

Ich gehe nun zur Multiplikation über. Hier ist es wiederum klar, daß man sich nichts Leichteres vorstellen kann. Denn man braucht keine Pythagoreische Tafel), und diese Multiplikation ist die einzige, die keine andere als bereits bekannt voraussetzt. Man schreibt nämlich nur die Zahl oder an ihrer Stelle 0.

```
          1 0 1 1 1 0 1
                  1 1 1 0
          ---------------
          0 0 0 0 0 0 0
        1 0 1 1 1 0 1
      1 0 1 1 1 0 1
    1 0 1 1 1 0 1
  ---------------
1 0 1 0 0 0 1 0 1 1 0
```

Pythagoreische Tafel

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Leibniz: Multiplikationsmaschine

Diese Art Kalkül könnte auch mit einer Maschine ausgeführt werden [ohne Räder] . Auf folgende Weise sicherlich sehr leicht und ohne Aufwand: [mit einer Dose - einer Dose mit Löchern - diese - an den Stellen, die entsprechen - wo . . . ist - deren Löcher geöffnet und geschlossen werden können - es sollen die den. . . entsprechenden geöffnet werden – mit einer kleinen Maschine mit zweizähnigen Rädern, welche -]

Eine Büchse soll so mit Löchern versehen sein, daß diese geöffnet und geschlossen werden können. Sie sei offen an den Stellen, die jeweils 1 entsprechen, und bleibe geschlossen an denen, die 0 entsprechen. Durch die offenen Stellen lasse sie kleine Würfel oder Kugeln in Rinnen fallen, durch die anderen nichts. Sie werde so bewegt und von Spalte zu Spalte verschoben, wie die Multiplikation es erfordert. Die Rinnen sollen die Spalten darstellen, und kein Kügelchen soll aus einer Rinne in eine andere gelangen können. es sei denn. nachdem die Maschine in Bewegung gesetzt ist. Dann fließen alle Kügelchen in die nächste Rinne, wobei immer eines weggenommen wird, welches in ein leeres Loch fällt [fallend dieses ausfüllt], sofern es allein die Tür passieren willl). Denn die Sache kann so eingerichtet werden, daß notwendig immer zwei zusammen herauskommen, sonst sollen sie nicht herauskommen.

3.2.1 Multiplikation von Dualzahlen

- P[n+m] = X[n] * Y [m]
- Zur Vereinfachung n=m: X=X[n], Y=Y[n]
- mit $Y=Y_n \ldots Y_1$

$$P = X * Y = X * Y_1 + 2^1 * X * Y_2 + \cdots + 2^{n-1} * X * Y_n$$

Abkürzung $\widehat{X} = X * 2^n$

$$P = (\cdots(\underbrace{(\widehat{X} * Y_1) * 2^{-1}}_{P^1} + \widehat{X} * Y_2) * 2^{-1} + \cdots + \widehat{X} * Y_n) * 2^{-1}$$

P^2

P^n

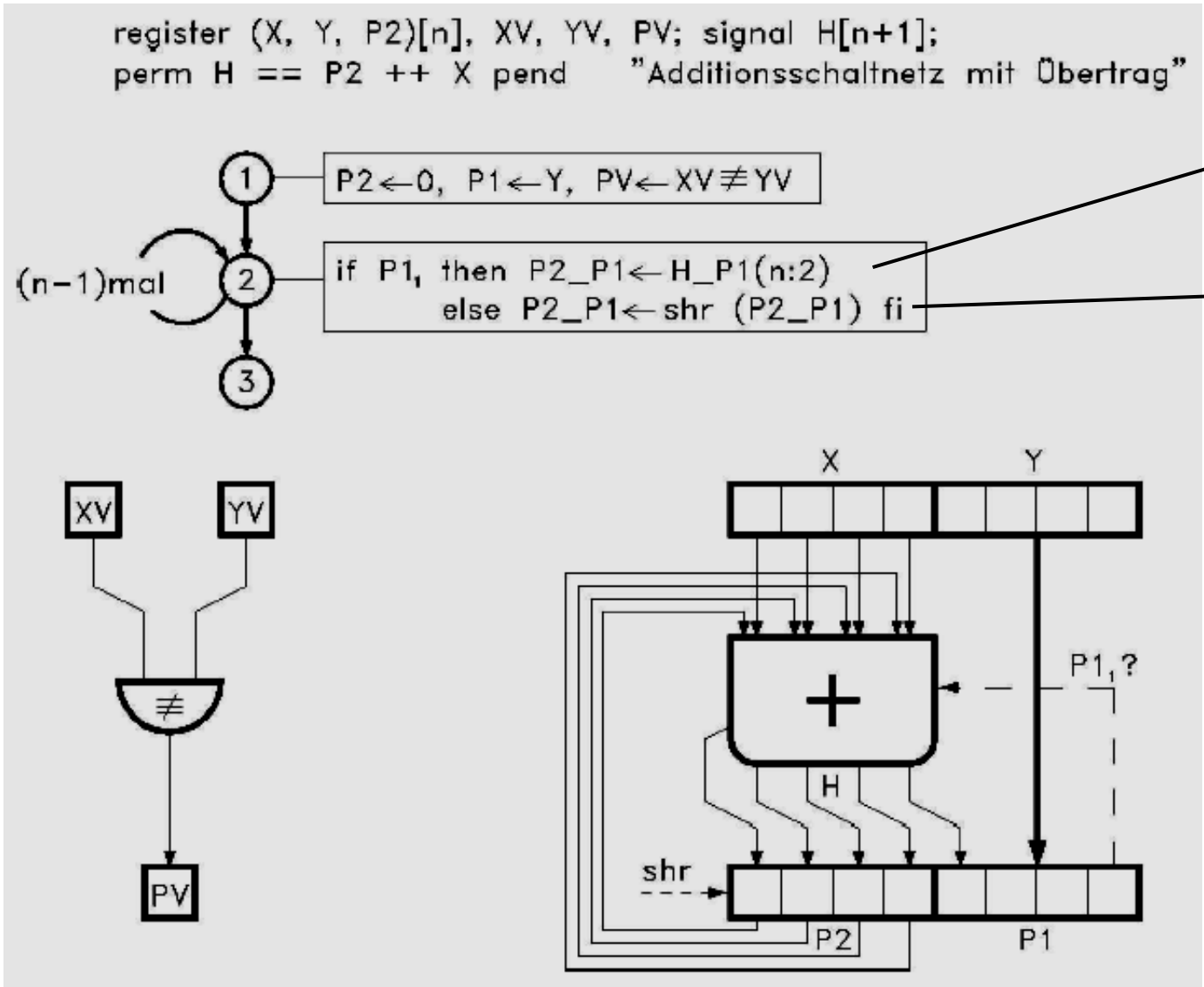
1. Umwandlung in Hornerschema
2. Rekursives Gleichungssystem
3. Algorithmus

```
boole X[n], Y[n], P[2n];
P:=0;
for i:=1 to n do
   P:=(P+X_oo…o * Y_i)/2 od
```

$$\begin{aligned} P^0 &= 0 \\ P^1 &= (P^0 + \widehat{X} * Y_1) * 2^{-1} \\ P^2 &= (P^1 + \widehat{X} * Y_2) * 2^{-1} \\ &\vdots \\ P^n &= (P^{n-1} + \widehat{X} * Y_n) * 2^{-1}. \end{aligned}$$

Letzter Shift kann entfallen mit der Abkürzung $X = \widehat{X} * 2^{n-1}$

Serien-paralleles Multiplikationswerk



Addition und Shift

Nur Shift

Beispiel zum Rechenwerk

Zahlenbeispiel: $(+15) * (-9) = -135$

$V_x = 0 \quad |X| = 1111$

$V_y = 1 \quad |Y| = 1001$

Zustand						
1			P2_P1	←	0000 1001	Vp ← 1
2	H = 0	1111	P2_P1	←	0111 1100	
2			P2_P1	←	0011 1110	
2			P2_P1	←	0001 1111	
2	H = 1	0000	P2_P1	←	1000 0111	„gleich − 135“
3						

3.2.2 Multiplikation von Zweikomplementzahlen, Grundsätzl. Methode

- **Gesucht**:
  P[2n] = X[n] mal$_{2k}$ Y[n]
  ←→ p=x*y

1. Rückabbildung einsetzen
2. Gleichsetzen
3. Vorzeichenbedingung

$$p = x * y = (X - X_n * 2^n) * (Y - Y_n * 2^n) \quad \text{und}$$
$$p = P[2n] - P_{2n} * 2^{2n}.$$

$$P[2n] = (X - X_n * 2^n) * (Y - Y_n * 2^n) + P_{2n} * 2^{2n}$$

$$P_{2n} = X_n \not\equiv Y_n = \overline{X_n} Y_n \vee X_n \overline{Y_n} = \overline{X_n} Y_n + X_n \overline{Y_n}$$

4. Einsetzen ergibt

$$P[2n] = X * Y - X_n * 2^n Y - Y_n * 2^n X$$
$$+ \underbrace{X_n Y_n 2^{2n} + \overline{X_n} Y_n 2^{2n}}_{=Y_n 2^{2n}} + \underbrace{X_n \overline{Y_n} 2^{2n} + \overbrace{X_n Y_n 2^{2n}}^{=0} - X_n Y_n 2^{2n}}_{=X_n 2^{2n}}.$$

Für den Fall $X_n = Y_n = 1$ muß 2^{2n} abgezogen werden, indem das Übertragsbit in die Stelle 2^{n+1} mit der Wertigkeit 2^{2n} nicht beachtet wird $\pmod{2^{2n}}$. Damit erhält man allgemein:

$$P[2n] = (X * Y + X_n * 2^n(2^n - Y) + Y_n * 2^n(2^n - X)) \bmod 2^{2n}$$

4 Fälle

1. $X_n = \mathtt{o}\,, Y_n = \mathtt{o}$: $P[2n] = X * Y$
   x und y sind positiv: es ist keine Korrektur notwendig.

2. $X_n = \mathtt{o}\,, Y_n = \mathtt{1}$: $P[2n] = X * Y + 2^n(2^n - X)$
   x ist positiv, y negativ: das Komplement von X, verschoben um n Stellen nach links, muß addiert werden.

3. $X_n = \mathtt{1}\,, Y_n = \mathtt{o}$: $P[2n] = X * Y + 2^n(2^n - Y)$
   x ist negativ, y positiv: das Komplement von Y, verschoben um n Stellen nach links, muß addiert werden.

4. $X_n = \mathtt{1}\,, Y_n = \mathtt{1}$:
   $P[2n] = X * Y + 2^n(2^n - Y) + 2^n(2^n - X) - [2^{2n}]$
   x und y sind negativ: die Komplemente von X und Y, verschoben um n Stellen nach links, müssen addiert werden. Der entstehende Übertrag wird nicht beachtet.

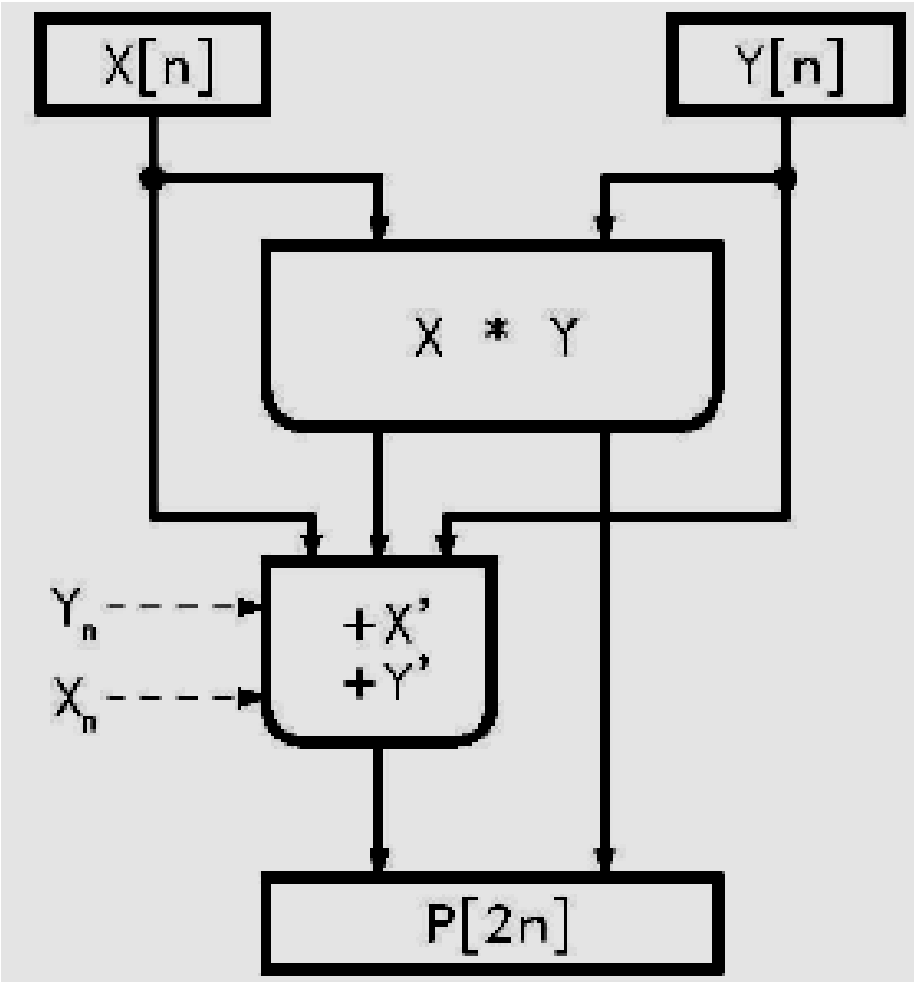
Zahlenbeispiel für den 4. Fall


```
X[5] = l o o l l , x = -13
Y[5] = l o l l o , y = -10

l o o l l * l o l l o
          o o o o o
        l o o l l
      l o o l l
    o o o o o
  l o o l l
  l l o l o o o l o
 o l l o l              +2^5(2^5 - X)
 o l o l o              +2^5(2^5 - Y)
l o o l o o o o o l o
```

wird abgezogen (ignoriert)

Rechenwerk, Prinzipielle Lösung



3.2.3 Parallele Multiplikation

- Möglichkeiten
  - 1. Entnehme Ergebnis aus einer Tabelle im Speicher
    - schnell, aber Tabellengröße $R = 2n * 2^{2n}$ Bits
  - 2. Minimiere die Funktion und speichere sie in einer Schaltmatrix (PLA, UND/ODER-Matrix)
    - Aufwand steigt auch signifikant mit n an.
  - 3. Zerlege die Funktion in eine Hintereinanderschaltung von einfachen Funktion, Schaltkettenrealisierung
    - Ein Ansatz: Benutze einen serien-parallelen Algorithmus (wie vorher behandelt) und wandle den zeitlich-seriellen Ablauf in einen räumlich seriellen.
    - Ergebnis ist eine paralleles Schaltnetz (Sicht von außen), das intern als Schaltkette arbeitet und die Zwischenergebnisse asynchron weitergibt.

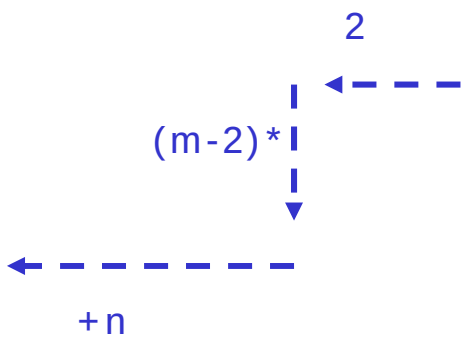
Schaltkette für Multiplikation von Dualzahlen

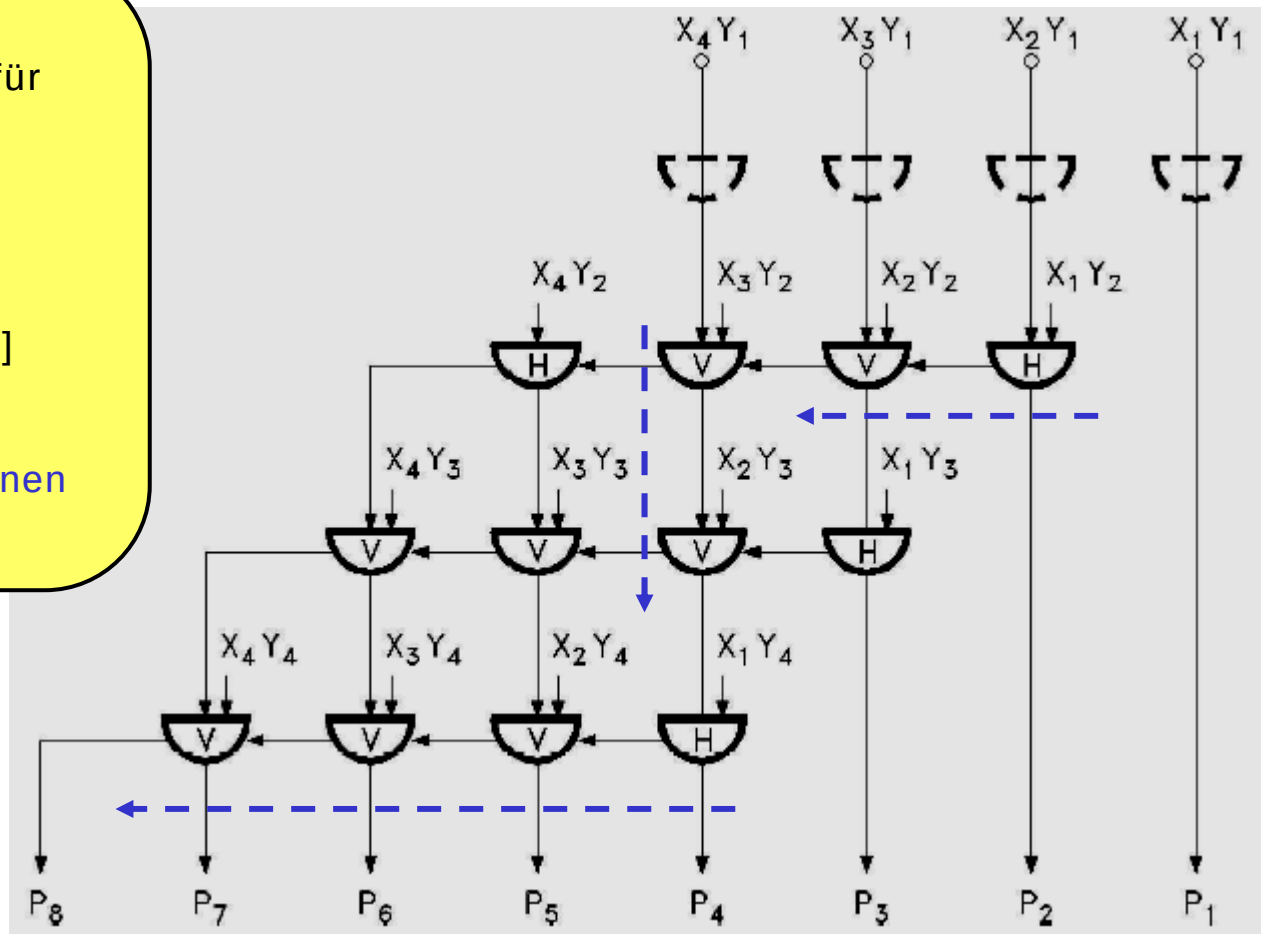
Umwandlung des serien-parallelen Rechenwerks für Dualzahlen

Nachteil: Carry-Ripple-Technik

Rechenzeit für X[n]*Y[m]
t = [n + (m-2)*2]*T

mit T = Laufzeit durch einen V oder H





3.3 Division, 3.3.1 Division von Dualzahlen

- Beschränkung der Betrachtung auf ganzzahlige Division von Dualzahlen
- **Rest = Dividend – Quotient ∗ Divisor**
- Quotient = (Dividend – Rest) / Divisor
- Drei Methoden
  - **Vergleichsmethode**
    - Wie oft paßt der Divisor in den höherwertigen Teil des Dividenden → Quotientenstelle Qi und Abziehen von D ∗ Qi .
    - Erfordert Vergleicher und Subtrahierer.
  - **Methode mit Rückstellen des Restes**
    - Vergleich wird durch Test-Subtraktion ersetzt.
    - erfordert Subtrahierer und Addierer
  - **Methode ohne Rückstellen des Restes**
    - Nach bestimmten Schema wird entweder subtrahiert oder addiert.
    - erfordert Subtrahierer und Addierer
    - geht schneller

Überlauf-Test

- Gesucht

$$r = p - q * d \quad \text{mit} \quad 0 \leq r < d$$

- Betrachtet werden positive Dualzahlen MSB = o
- **Überlauf** kann auftreten, wenn die vorgesehen (n-1) Stellen für den Quotienten nicht ausreichen, **q darf nicht größer als $2^{n-1} - 1$ werden.**
  - Der um (n-1) Stellen verschobene Divisor muß größer als p sein, damit kein Überlauf auftritt.
  - Vor Beginn wird diese Bedingung getestet, entweder durch einen Vergleicher oder eine Testsubtraktion

Dividend

Divisor

$$\begin{aligned} p = P[2n-1] &= \circ\, P_{2n-2} \ldots P_1 \\ q = Q[n] &= \circ\, Q_{n-1} \ldots Q_1 \\ d = D[n] &= \circ\, D_{n-1} \ldots D_1 \\ r = R[n] &= \circ\, R_{n-1} \ldots R_1 \end{aligned}$$

$$\begin{aligned} 0 &\leq q \leq 2^{n-1} - 1 \\ 0 &\leq q * d + r \leq (2^{n-1} - 1) * d + r < 2^{n-1} * d \\ 0 &\leq p \leq 2^{n-1} * d. \end{aligned}$$

$$(p - 2^{n-1} * d < 0)$$

Entwicklung der Vergleichsmethode

1. Umwandlung Hornerschema
2. Schrittweise $Q_i * D$ abziehen, aber Q_i noch unbekannt
3. Rekursionsschema

- Falls Divisor in den höherwertigen Teil des Dividenden paßt ($D \leq 2R^i$), dann ist Q_{i-1}=l und die durchzuführende Subtraktion $R^{i-1} = 2R^i - D$ ergibt den neuen (Zwischen-)Rest $R^{i-1} \geq 0$.
- Erforderlich: Vergleicher

$$
\begin{aligned}
R &= P - (Q_n Q_{n-1} \cdots Q_1) * D \\
R &= P - Q_n * D * 2^{n-1} - Q_{n-1} * D * 2^{n-2} - \cdots Q_1 * D \\
R &= ((P 2^{-(n-1)} - Q_n * D) 2 - Q_{n-1} * D) 2 - \cdots Q_1 * D
\end{aligned}
$$

$$
\begin{aligned}
R^n &= P * 2^{-(n-1)} - Q_n * D \\
R^{n-1} &= R^n * 2 - Q_{n-1} * D \\
&\vdots \\
R^1 &= R^2 * 2 - Q_1 * D \,.
\end{aligned}
$$

nur Abziehen falls Rest ≥ 0

Vergleichsmethode, Beispiel

34 ÷ 5 = 6 Rest 4

```
  0100010 ÷ 0101    = 0110, Rest 0100
(-0101)          nicht möglich ⇒ 0
   1000
  -0101          passt ⇒ 1
    0111
  - 0101         passt ⇒ 1
     0100
   (-0101)      nicht möglich ⇒ 0

     0100        ⇒ Rest
```

Methode mit Rückstellen des Restes

$R^{i-1} := R^i * 2 - Q_{i-1} * D$ kann wie folgt umgeformt werden:

$R^{i-1} := R^i * 2 - D$;
if $R^{i-1} < 0$ then $Q^{i-1} := 0$; $R^{i-1} := R^{i-1} + D$; „Rückstellen“
else $Q^{i-1} := 1$; fi .

Dabei wird in jedem Schritt zuerst der Divisor probeweise subtrahiert; wird der Teilrest R^{i-1} negativ, dann war die Subtraktion unberechtigt ($Q_{i-1} = 0$), und der Teilrest wird durch Addition von D zurückgestellt.

Vorteil: Kein separates Vergleichsschaltnetz erforderlich

Methode ohne Rückstellen des Restes

- Umformung ergibt neues Rekursionsschema
  1. Im ersten Schritt wird getestet, ob die Bedingung $p < d * 2^{n-1}$ erfüllt ist. Überlauf, falls die Subtraktion mit D kein negatives Ergebnis liefert.
  2. Schritt i : Wenn der vorhergehende Teilrest negativ ist, dann wird D addiert, sonst subtrahiert
     1. Wenn der neue Teilrest negativ ist, dann ist $Q_i = o$, sonst ist $Q_i = l$.
  3. Korrekturschritt am Ende, damit Rest positiv
     - if $R^1<0$ then $R:=R^1+D$

Umformung

$$R = P - D * 2^{n-1} + (1 - 2Q_n)D * 2^{n-2} + \cdots + (1 - 2Q_2)D + (1 - Q_1)D\,.$$

Rekursionsschema

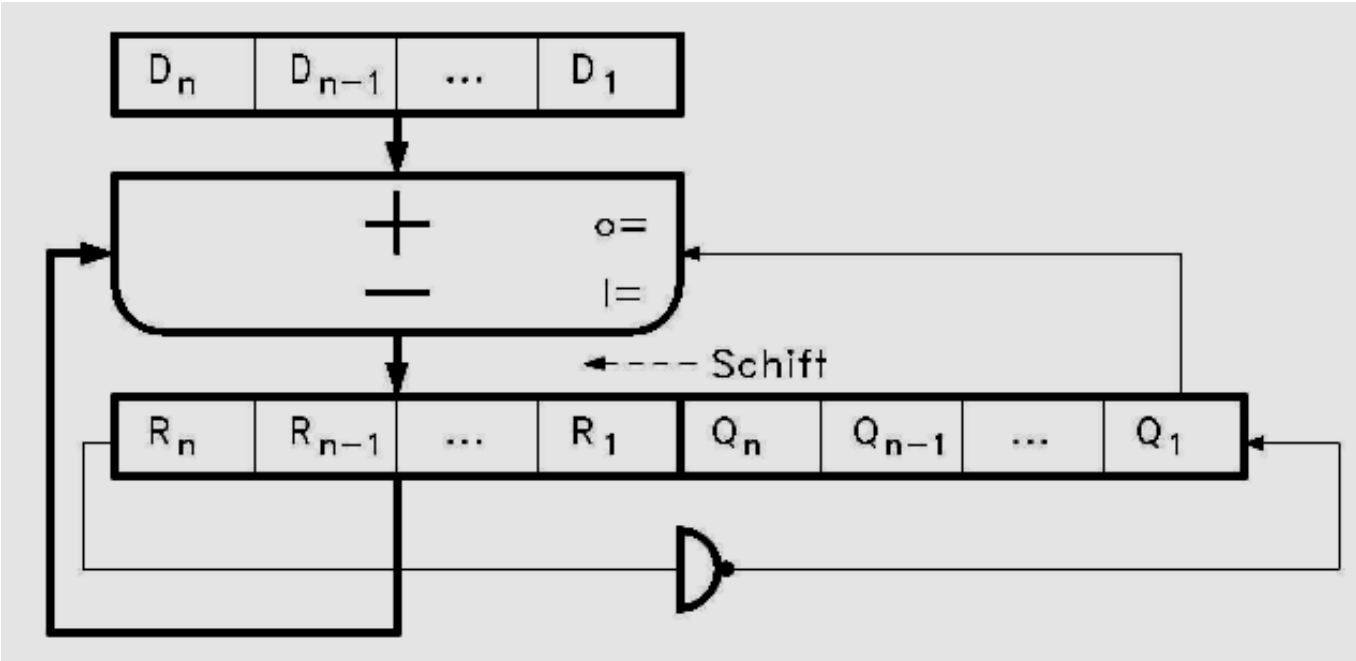
$$\begin{aligned} R^n &= P * 2^{-(n-1)} - D && \rightarrow Q_n = (R_n \geq 0) \\ & && \text{if } Q_n = 1 \text{ then „Überlauf“} \\ R^{n-1} &= R^n * 2 + (1 - 2 * Q_n) * D && \rightarrow Q_{n-1} = (R_{n-1} \geq 0) \\ &\vdots \\ R^1 &= R^2 * 2 + (1 - 2 * Q_2) * D && \rightarrow Q_1 = (R_1 \geq 0) \\ R &= R^1 + (1 - Q_1) * D && . \end{aligned}$$

Ohne Rückstellen, Beispiel

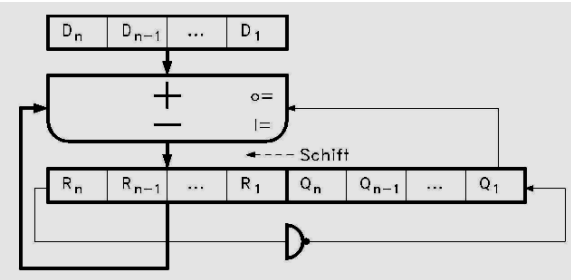
34 ÷ 5 = 6 Rest 4

```
   mit D = 0101, D' = 1011
   0 1 0 0 0 1 0  ÷  0 1 0 1  =  0 1 1 0,  Rest 0 1 0 0
+  1 0 1 1                    +D', Test auf Overflow im ersten Schritt
   1 1 1 1 0                  negativ  ==>   Quotientenbit 0, kein Overflow
+    0 1 0 1                  +D
 1 0 0 0 1 1 1                positiv  ==>      Quotientenbit 1
     + 1 0 1 1                +D'
     1 0 0 1 0 0              positiv  ==>         Quotientenbit 1
       + 1 0 1 1              +D'
         1 1 1 1              Rest negativ  ==>            Quotientenbit 0
       + 0 1 0 1              +D, damit Rest positiv wird
       1 0 1 0 0
```

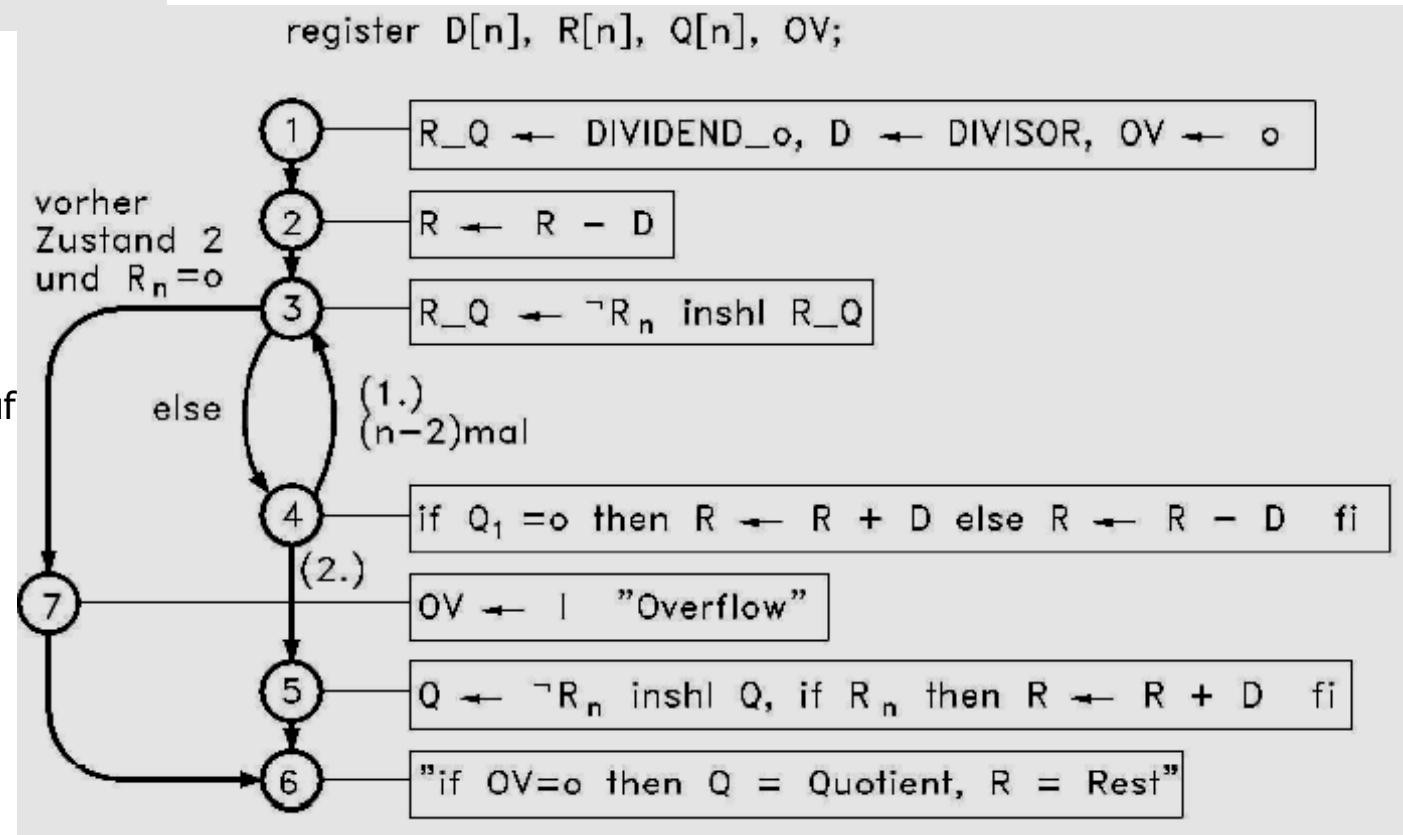
Dividierwerk



Mikroalgorithmus / Steueralgorithmus



Divisionsüberlauf



Zahlenbeispiel

34 ÷ 5 = 6 Rest 4