Zahlendarstellung

Addition & Subtraktion

Benjamin Tröster

Hochschule für Technik und Wirtschaft Berlin

15. Dezember 2021

Fahrplan

Addition und Subtraktion

Subtraktion

Gleitkomma-Addition

Arithmetisch-logische Einheit

Addition und Subtraktion

- Schaltungen zur Addition von Festkomma-Dualzahlen:
  - Grundlage für die Durchführung aller arithmetischen Verknüpfungen
  - Denn:
    - Subtraktion entspricht der Addition mit negativen Zahl
    - $X - Y = X + (-Y)$
- Multiplikation und Division lassen sich ebenfalls auf die Addition zurückführen
  - Bei Gleitkommazahlen:
    - Mantisse und Exponent werden separat verarbeitet
    - Hierbei bildet die Addition von Festkomma-Dualzahlen die Grundlage
- Grundtypen von Addierern sind wichtig

Halbaddierer

- Bei der Addition zweier Dualzahlen:
  - Entstehen Summe und Übertrag als Ergebnis
- Funktionstabelle, Eingangswerte a, b und Summe s, sowie Übertrag ü/ c

a	b	s	ü/c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

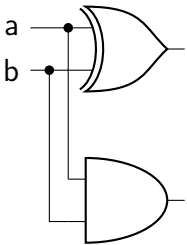
- Man nennt dies einen Halbaddierer

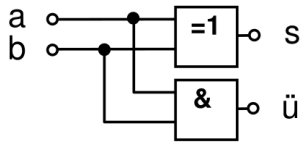
Halbaddierer

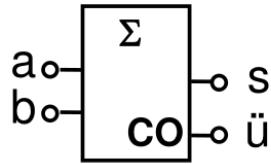
- Gleichung:

$$s = a \wedge b \vee a \wedge b = a \oplus b$$
$$c = a \wedge b$$

- Als Schaltbild und Schaltsymbol:







Mehrstellige Dualzahlen

- Zusätzlicher Eingang für den Übertrag der vorhergehenden Stellen ist nötig

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	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

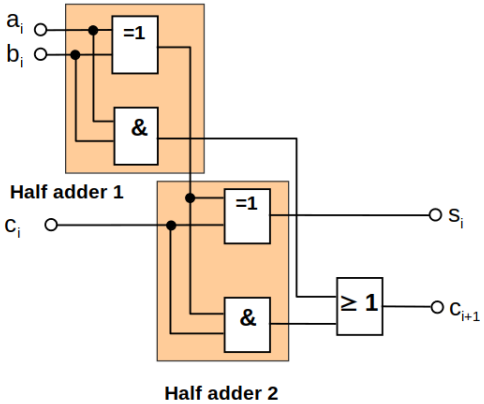
- Dies nennt man einen Volladdierer

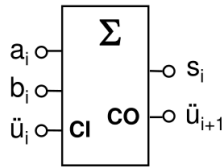
Gleichungen, Schaltnetz und Schaltsymbol

- Ausgangsgleichungen:

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i \wedge c_i \vee b_i \wedge c_i \vee a_i \wedge b_i = (a_i \oplus b_i) \wedge c_i \vee a_i \wedge b_i$$

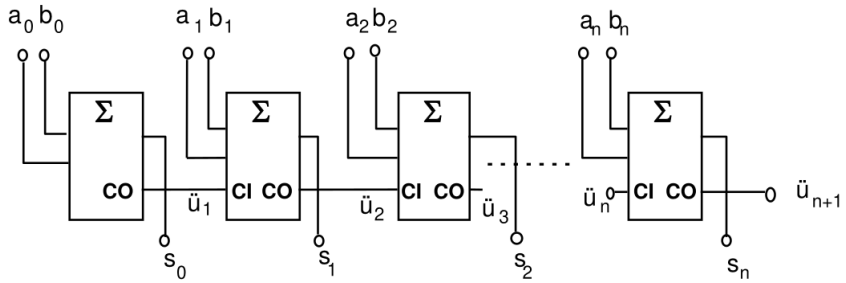




Volladdierer

Carry-ripple-Addierer

- Addieren zweier Dualzahlen mit mehreren Stellen
- Einfachste Lösung:
  - Für jede Stelle einen Volladdierer vorsehen und den Übertrag der Stelle i im Volladdierer der Stelle $i + 1$ berücksichtigen
  - Die Stelle geringster Wertigkeit (LSB, least significant bit) kann mit einem Halbaddierer realisiert werden



Probleme

- Ergebnis der Addition einer Stelle ist erst dann gültig, wenn der Übertrag aus der vorhergehenden Stelle berechnet ist
- Ungünstiger Fall: Das Durchlaufen durch alle Stufen muss abgewartet werden (Carry-ripple-Addierer, to ripple = rieseln)
- Die Stabilisierungsdauer ist proportional zur Anzahl der Stellen
- Man nennt den Carry-ripple-Addierer auch Asynchroner Parallel-Addierer, da er bit-parallel addiert, d.h. alle Bits der Operanden gleichzeitig benutzt

Carry-lookahead-Addierer

- Um den Nachteil der großen Additionszeit des Carry-ripple-Addierers zu vermeiden:
  - Alle Überträge direkt aus den Eingangsvariablen bestimmen (Carry-Lookahead)
- Es gilt:

$$c_{i+1} = a_i \wedge b_i(a_i \oplus b_i) \wedge c_i \qquad = g_i \vee p_i c_i$$
$$s_i = (a_i \oplus b_i) \oplus c_i \qquad = p_i \oplus c_i$$

  mit
  - $g_i = a_i \wedge b_i$ (generate carry, erzeuge Übertrag) und
  - $p_i = (a_i \oplus b_i)$ (propagate carry, leite Übertrag weiter)
- g_i und p_i können direkt aus den Eingangsvariablen erzeugt werden

Berechnung der Überträge aus den Eingangsvariablen

- Die rekursive Berechnung der Übertrage c_i kann aufgelöst werden, indem sukzessive die Ausdrücke für die Berechnung des Übertrags in den vorhergehenden Stellen eingesetzt werden

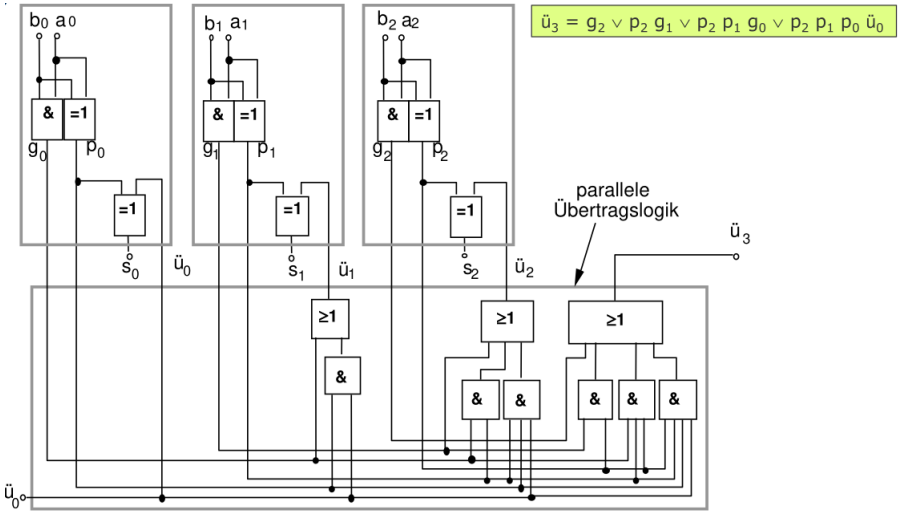
$$c_{i+1} = g_i \vee p_i \wedge c_i$$

- Man erhält

$$\begin{aligned}
c_1 &= g_0 \vee p_0 \wedge c_0 \\
c_2 &= g_1 \vee p_1 \wedge g_0 \vee p_1 \wedge p_0 \wedge c_0 \\
c_3 &= g_2 \vee p_2 \wedge g_1 \vee p_2 \wedge p_1 \wedge g_0 \vee p_2 \wedge p_1 \wedge p_0 \wedge c_0 \\
&\dots \\
c_n &= g_{n-1} \vee p_{n-1} \dots \wedge c_0
\end{aligned}$$

- Die Additionszeit wird damit weitgehend unabhängig von der Stellenzahl, weil die Berechnung des Übertrags in allen Stufen sofort (vorausschauend) beginnen kann. Deshalb wird dieser Addierer als Carry-lookahead-Addierer bezeichnet

Schaltbild: 3-Bit-Carry-lookahead-Addierer

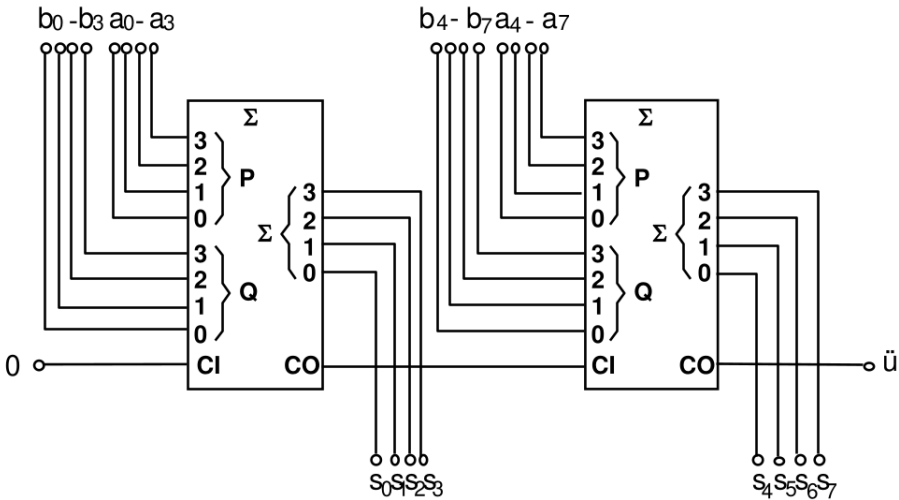


Carry-lookahead-Addierer

- Problem:
  - Größe des Hardware-Aufwands steigt mit steigender Stellenzahl stark an.
- Lösungen:
  - kleinere Carry-lookahead-Addierer mit paralleler Übertragserzeugung, die seriell kaskadiert werden
  - Blocküberträge der kleineren Blöcke parallel verarbeiten
  - $\Rightarrow$ Hierarchie von Carry-lookahead-Addierern

Carry-lookahead-Addierer Kaskadierung

Kaskadierung zweier 4-Bit Carry-lookahead-Addierer zur Addition von 8-Bit-Zahlen



Subtraktion

- Subtraktion durch Addition des Zweierkomplements
- Zweierkomplement: bit-weise Komplementierung der Zahl und anschließende Addition von 1

$$X - Y = X + (\neg Y + 1) = X + \neg Y + 1$$

- Man beachte:
  - Wir nehmen an, wenn beide Eingabezahlen X, Y im Zweierkomplement-Form gegeben sind
  - $\rightarrow$ Am Ausgang entsteht wieder eine Zahl in Zweierkomplement-Form

Subtraktion

- Die beiden Additionen können mit einem Addierer vorgenommen werden, indem man den Übertragseingang ausnutzt

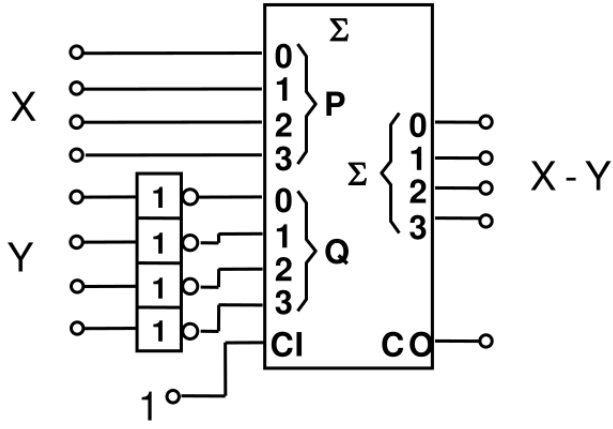
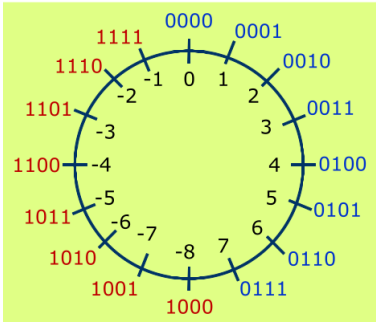


Abbildung: Subtraktion von Zweierkomplementzahlen

Sonderfälle

- Bei der Addition lassen sich 3 Sonderfälle unterscheiden
  1. Beide Summanden sind positiv
     - die Vorzeichenbits beider Zahlen sind 0
     - das Ergebnis muss positiv sein
     - Das Ergebnis ist nur dann korrekt, wenn sein Vorzeichenbit gleich 0 ist, ansonsten wurde der Zahlenbereich überschritten
     - Man kann sich diese Situation anhand des Zahlenkreises klarmachen




```
  5        0 1 0 1
+ 2      +   0 0 1 0
-----    -----------
= 7        0 0 0 0 0
           -------
           0 1 1 1
```

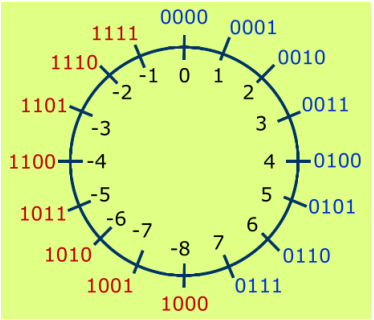
Überträge gleich
Ergebnis korrekt

```
  5        0 1 0 1
+ 6      +   0 1 1 0
-----    -----------
= 11       0 1 0 0 0
           -------
           1 0 1 1
```

Überträge ungleich
Überlauf
Ergebnis falsch

Sonderfall 2

2. Beide Summanden sind negativ
   - Die Vorzeichenbits beider Zahlen haben den Wert 1
   - Das Ergebnis muss negativ sein
   - Das Ergebnis ist nur dann korrekt, wenn das Vorzeichenbit des Ergebnisses 1 ist
   - Die beiden vordersten Überträge müssen den gleichen Wert haben




```
 - 5          1 0 1 1
+ (- 2)      + 1 1 1 0
-------      -------
= - 7        1 1 1 0
             -------
             1 0 0 1
```

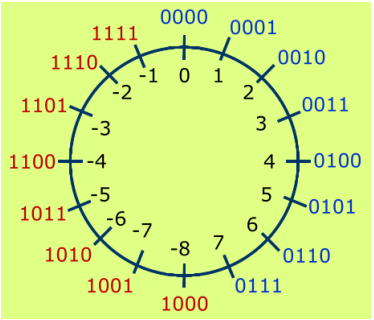
Überträge gleich
Ergebnis korrekt

```
 - 5          1 0 1 1
+ (- 6)      + 1 0 1 0
-------      -------
= - 11       1 0 1 0
             -------
             0 1 0 1
```

Überträge ungleich
Überlauf
Ergebnis falsch

Sonderfall 3

3. Beide Summanden haben unterschiedliche Vorzeichen
   - Das Ergebnis ist auf jeden Fall korrekt, das Vorzeichen hängt davon ab, ob Subtrahend oder Minuend betragsmäßig größer ist
   - Der Übertrag aus der vordersten Stelle ist zu streichen




```
   5        0 1 0 1          - 5        1 0 1 1
+ (- 6)     1 0 1 0        + 6        0 1 1 0
-------     0 0 0 0 0      -------    1 1 1 0 1 0
= - 1       1 1 1 1        = 1        0 0 0 1
```

Überträge gleich Ergebnis korrekt

Überträge gleich Ergebnis korrekt

Überlauferkennung

- Allgemeine Überlauferkennung bei dualer Addition:
  - korrekte Addition: beide Überträge sind gleich
  - Überlauf: beide Überträge sind ungleich
- Realisierung z.B. durch ein Antivalenzgatter (XOR)

Zusatzbetrachtung: Gleitkomma-Addition

- Addition von zwei Gleitkommazahlen a_1 und a_2

$$a_1 = s_1 \cdot b^{e_1} \qquad a_2 = s_2 \cdot b^{e_2}$$

- Beispiel:
  - $a_1 = 3,21 \cdot 10^2$
  - $a_2 = 8,43 \cdot 10^{-1}$
- Gerechnet wird mit zwei zusätzlichen Stellen in der Mantisse (Guard und Round) sowie dem Sticky-Bit

Gleitkomma-Addition

1. Exponentenangleichung
   - Gleitkommazahlen können nur addiert werden, wenn die Exponenten gleich sind
   - Schritt 1: Wenn $e_1 < e_2$, dann vertausche die Operanden, so dass gilt: $d = e_1 - e_2 \geq 0$
   - Schritt 2: Verschiebe die Mantisse s_2 um d Stellen nach rechts
     - Wenn $d > 2$, setze das Sticky Bit, falls die $d - 2$ herausgeschobenen Stellen einen Wert $\neq 0$ ergeben
   - Im Beispiel: $d = 2 - (-1) = 3$

$3,2100 \cdot 10^2$

$0,0084 \cdot 10^2$ Sticky-Bit gesetzt, da die 3 aus 8,43 herausgeschoben wird

Gleitkomma-Addition

2. Mantissenaddition
   - Addiere die beiden Mantissen
   - Im Beispiel:

$$\begin{array}{r} 3,2100 \cdot 10^2 \\ 0,0084 \cdot 10^2 \\ \hline 3,2184 \cdot 10^2 \end{array}$$

3. Normalisierung
   - Normalisiere die entstandene Summe durch Verschieben der Mantisse und Korrektur des Exponenten

Gleitkomma-Addition

4. Rundung
   - Runde unter Berücksichtigung der Stellen g, r und des Sticky-Bits, sowie der gegebenen Rundungsart (meist „round-to-even“)
   - Im Beispiel:

$$\begin{array}{r} 3,2100 \cdot 10^2 \\ 0,0084 \cdot 10^2 \\ \hline 3,2184 \cdot 10^2 \end{array}$$

5. Ergebnis wird gerundet zu: $3,22 \cdot 10^2$

Beispiel 1: $32 - 2.25 = 30$ mit 4 Genauigkeit 4, $q = 2$

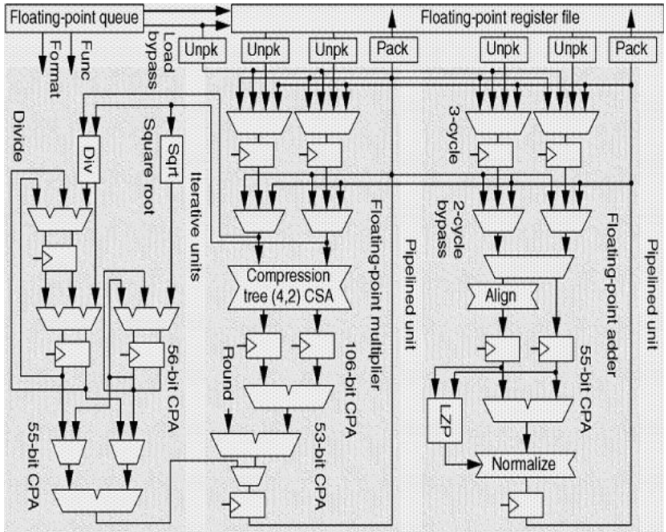
$1.000 \cdot 2^5 - 1.001 \cdot 2^1$
Unendliche Genauigkeit

1.000	$0000 \cdot 2^5$	
-0.000	$1001 \cdot 2^5$	anpassen
0.111	$0111 \cdot 2^5$	prez. 29.75
1.110	$1110 \cdot 2^4$	norm.
1.111	$\cdot 2^4$	runden

Nutzung von Guard, Round, Sticky Bits
Plus 4 Stellen Genauigkeit

$\overbrace{1.000}$	$000 \cdot 2^5$	
-0.000	$101 \cdot 2^5$	
0.111	$011 \cdot 2^5$	
1.110	$11 \cdot 2^4$	norm.
1.111	$\cdot 2^4$	runden

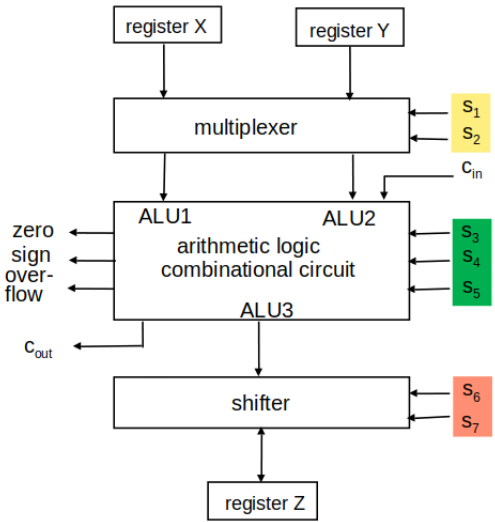
MIPS R10000 Floating-point Unit



Arithmetisch-logische Einheit

- Arithmetisch-logische Einheit (ALU, arithmetic logic unit):
  - Rechenwerk, der funktionale Kern eines Digitalrechners zur Durchführung logischer und arithmetischer Verknüpfungen
- Eingangsdaten der ALU:
  - Daten und Steuersignalen vom Prozessor
- Ausgangsdaten der ALU:
  - Ergebnisse und Statussignale an den Prozessor
- Oft können die in einen Prozessor integrierten ALUs nur Festkommazahlen verarbeiten. Die Gleitkommaoperationen werden dann entweder von einer Gleitkommaeinheit ausgeführt oder per Software in eine Folge von Festkommabefehlen umgewandelt

Schema einer einfachen ALU



s_1	s_2	ALU1	ALU2
0	0	X	Y
0	1	X	0
1	0	Y	0
1	1	Y	X

s_3	s_4	s_5	ALU3
0	0	0	ALU1 + ALU2 + c_{in}
0	0	1	ALU1 – ALU2 – Not(c_{in})
0	1	0	ALU2 – ALU1 – Not(c_{in})
0	1	1	ALU1 $\veebar$ ALU2
1	0	0	ALU1 $\wedge$ ALU2
1	0	1	Not(ALU1) $\wedge$ ALU2
1	1	0	ALU1 $\oplus$ ALU2
1	1	1	ALU1 $\leftrightarrow$ ALU2

s_6	s_7	Z
0	0	ALU3
0	1	ALU3 $\div$ 2
1	0	ALU3 $\times$ 2
1	1	store Z

Bestandteile der ALU

- Registersatz
- Multiplexerschaltnetz
- Arithmetisch logisches Schaltnetz zur Durchführung arithmetisch logischer Operationen
- Schiebeschaltnetz
- Eingänge:
  - Datenworte X und $Y4$
  - Steuersignale $s_1 \ldots s_7$ zur Festlegung der ALU-Operation
- Ausgänge:
  - Statussignale zero, sign und overflow
  - Hiermit kann das Steuerwerk bestimmte ALU-Zustände erkennen und darauf entsprechend reagieren

Beispiele

- Einerkomplement von Y um ein Bit nach links verschoben in Z ablegen
  - Steuersignale: $s_1 \dots s_7 = 1011110$
  - 10 : $ALU1 = Y$
  - 111: $ALU3 = ALU1 \leftrightarrow ALU2$
  - 10: $Z = ALU3 \cdot 2$
- Ist $X > Y$?
  - Statussignal „sign“ bei der Operation $Y - X$
  - Steuersignale: $s_1 \dots s7 = 0001000$ und $c_{in} = 1$
    - 00 : $ALU1 = X$ und $ALU2 = Y$
    - 010: $ALU3 = ALU2 - ALU1 - not(c_{in})$
    - 00 : $Z = ALU3$

Quellen I