Übungsaufgaben

Anmerkung

Allen Beispielen soll noch hinzugefügt sein, dass wertvolle Hinweise, also die Tipps und Tricks die der schnellen maschinellen Multiplikation zu Grunde liegen, neben dem Stoff zur Vorlesung, in Kapitel 6.8 bis 6.10 des Skripts hinterlegt wurden. Beispielsweise die zu Beginn von Kap. 6.8 gemachte Einleitung auf das, was im folgenden Stoff zur Multiplikation zu berücksichtigen ist, die Vorzeichen Beachtung, die Verringerung der Anzahl der Operationen auf die im Beispiel auf Seite 111 Skript hingewiesen wird, der Trick, auf den Seite 112 bzgl. Bitgruppenbildung hinweist. Und ferner den Hinweis, auf den Sinn der Vorzeichenerweiterung im Boothalgorithmus und die Zwischenergebnisse, wie auf Seite 113 unten beschrieben.
Wie im Carry Save Beispiel unten gezeigt, finden Sie noch einmal methodische, technische Erläuterungen in Kap. 6.9 zur schnellen Multiplikation.
Berücksichtigen Sie diese Dinge, werden Ihnen diese Beispiele noch klarer. Viel Erfolg!

Prinzip Multiplikation

- Vorgehensweise entsprechend dem Algorithmus der schriftlichen Multiplikation

Beispiel 10 • 29 = 290 :

```
 001010 • 011101
          001010
         000000
        001010
       001010
      001010
     001010
    000100100010
```

Multiplikation mit negativen Zahlen

(+a) • (+b) → (+a) • (+b)
(+a) • (-b) → (-b) • (+a)
(-a) • (+b) → (-a) • (+b)
(-a) • (-b) → (+a) • (+b)

positive Zahlen werden wie beschrieben multipliziert

treten zwei negative zahlen auf, so ist das Produkt positiv. Dazu werden beide über das 2er-Komplement in positive Zahlen umgewandelt

tritt eine negative Zahl auf, so wird sie als linker Multiplikator genutzt

Algorithmus zur Multiplikation mit negativen Zahlen:

- links: negativer Faktor, rechts: positiver Faktor
- ermitteln der Stellenzahl des Produktes
- Bilden der Teilprodukte:
  - wenn aktuelles Bit im rechten Faktor 1, dann übernimm linken Faktor (Multiplikation mit 1)
  - Vorzeichenausdehnung bis zur höchsten Stelle entspr. Stellenzahl des Produktes
  - wenn aktuelles Bit im rechten Faktor 0, dann ist dieses Teilprodukt 0
- Addition der Teilprodukte. Ein Überlauf ist durch die negativen Vorzeichen bestimmt und wird ignoriert.

Beispiel:

(-22) • (+17) = - 374

```
 101010 • 010001
 ------------------
    111111101010
    000000000000
    0000000000
    000000000
    11101010
    0000000
 ------------------
   1111010001010
```

Booth Multiplikation

Multiplizieren Sie $+85_{10}$ und -65_{10} binär (8-Bit-Darstellung) mit Hilfe des Booth-Algorithmus!

Lösung:

1. Binärdarstellung mit Zweierkomplement:

85 binär: 01010101
-85 binär (ZK): 10101011
65 binär: 01000001
-65 binär (ZK): 10111111

2. Booth-Codierung

```
   1  0  1  1  1  1  1  1 (0)
  -1 +1  0  0  0  0  0 -1
```

3a. Multiplikation mit dem "Schulschema"

```
0  1  0  1  0  1  0  1  *  -1 +1  0  0  0  0  0 -1
-----------------------------------------------------
    1  1  1  1  1  1  1  1  1  0  1  0  1  0  1  1
    0  0  0  1  0  1  0  1  0  1
    1  1  0  1  0  1  0  1  1
-----------------------------------------------------
(1) 1  1  1  0  1  0  1  0  0  1  1  0  1  0  1  1
```

Probe:
ZK: 0001010110010101
dezimal: 5525_{10}

85 * (-65) = -5525

3b. Multiplikation mit Tabellenmethode

00	nichts
01	addiere A auf linke Hälfte Produktregister
10	subtrahiere A von linker Hälfte Produktregister bzw. addiere -A auf linke Hälfte Produktregister
11	nichts

Schritt	Operation		B	Q
	Init	00000000	10111111	0
1	-A	00000000 **10101011** 10101011	1011111**1** 10111111	**0**
	ASR	11010101	11011111	1
2	nichts	11010101	1101111**1**	**1**
	ASR	11101010	11101111	1
3	nichts	11101010	1110111**1**	**1**
	ASR	11110101	01110111	1
4	nichts	11110101	0111011**1**	**1**
	ASR	11111010	10111011	1
5	nichts	11111010	1011101**1**	**1**
	ASR	11111101	01011101	1
6	nichts	11111101	0101110**1**	**1**
	ASR	11111110	10101110	1
7	+A	11111110 **01010101** 01010011	1010111**0** 10101110	**1**
	ASR	00101001	11010111	0
8	-A	00101001 **10101011** 11010100	1101011**1** 11010111	**0**
	ASR	11101010	01101011	1

Ergebnis: 1110 1010 0110 1011

Booth-Algorithmus mit Bit Pair Recoding

Er dient zur weiteren Beschleunigung der Multiplikation.

Idee: Zur Verringerung der Teilsummanden wird vorzugsweise bei größeren 1er-Blöcken ausgenutzt, dass folgender Zusammenhang gilt:

$$a \cdot b = a \cdot (b+1) - a$$

Dazu wird zu einem größeren 1er-Block in einem Faktor ein Betrag hinzuaddiert, der im Teilprodukt wieder abgezogen wird. Das hat zur Folge, dass bei dem durch Addition entstandenen Faktor weniger 1en vorkommen.

Beispiel:

7	0	1	1	1
8	1	0	0	0
-1	0	0	0	-1
	+1	0	0	-1

Algorithmus zum Recoding:

- Ausgangspunkt ist LSB
- erfolgt ein Wechsel von 0 auf 1, so ist unter der 1 „-1“ zu notieren
- erfolgt ein Wechsel von 1 auf 0, so ist unter der 0 „+1“ zu notieren
- ist das LSB 1, so erfolgt ein Wechsel von einer virtuellen 0 zu einer 1 und es ist unter das LSB „-1“ zu notieren

Beispiel:

```
  ┌─┐       ┌─┐
0 0 1 1 1 1 0 0
0 +1 0 0 0 -1 0 0
```

Algorithmus zum Multiplizieren:

- rechter Faktor ist der recodierte Faktor
- tritt eine 1 oder 0 auf, so ist wie bekannt zu multiplizieren
- tritt eine „–1“ auf, so
  - bilde das 2er-Komplement des linken Faktors
  - multipliziere mit 1
  - führe Vorzeichenausdehnung aus wie bei Multiplikation mit negativem Faktor
- Der Algorithmus ist für positive und negative Faktoren anwendbar

Beispiel: (+30) • **(-9) = -270**

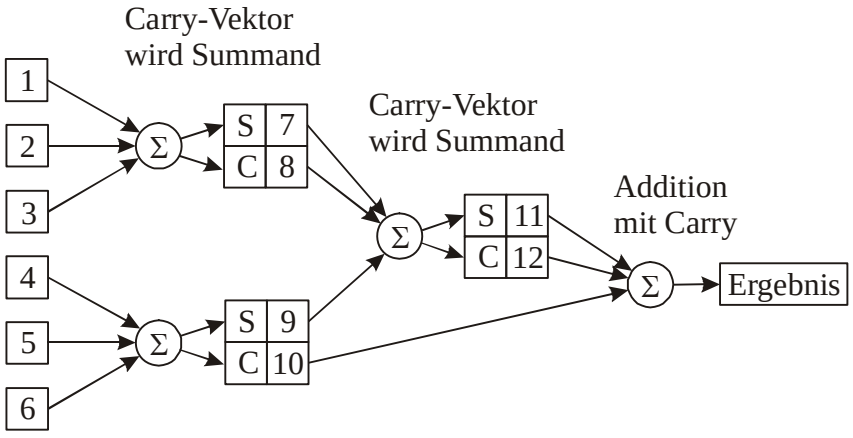
```
0 1 1 1 1 0 • 1 1 0 1 1 1
0 1 1 1 1 0 • 0 -1 +1 0 0 –1
    1 1 1 1 1 1 1 0 0 0 1 0
    0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 1 1 1 1 0
    1 1 1 0 0 0 1 0
    0 0 0 0 0 0 0
    1 1 1 0 1 1 1 1 0 0 1 0
```

Carry-Save-Algorithmus

Problem und Lösung : Bei einer vollständigen Addition mit Übertrag kann das Ergebnis erst ermittelt werden, wenn die Überträge in Abhängigkeit aller niederwertigeren Bits berechnet worden sind. Das tritt beim Ripple-Carry-Adder besonders in Erscheinung. Der Carry-Save-Algorithmus macht sich dabei den Umstand zunutze, dass der Übertrag bei einer Addition zweier Bits zu einer zusätzlichen Addition im nächst höherwertigen Bit führt.. Da der Carry-Vektor aber eigentlich nur ein zusätzlicher Summand ist, kann er in eine weitere Addition als Summand einfließen. Dadurch wird bei Berechnung der Teilsumme die zeitaufwendige Auswertung des Carry-Vektors eingespart. Die neuen Summanden werden hier durch eine zweistufige Logik erzeugt. Drei Summanden werden jeweils zu zwei neuen zusammengefasst, somit erfolgt die Reduktion von Summanden bei jedem Schritt um jeweils ein Drittel. Nur die letzte Addition muss unter Einbeziehung des Carry-Vektors erfolgen. Dazu kann vorteilhaft ein Carry Look Ahead Adder genutzt werden.

Beispiel: 55 • 59 = 3245

		110111 • 111011	
		110111	(1)
		110111	(2)
		000000	(3)
		110111	(4)
		110111	(5)
		110111	(6)
(1)+(2)+(3)	S	**01011001**	(7)
	C	**00100110**	(8)
(4)+(5)+(6)	S	**10000101**	(9)
	C	**01111110**	(10)
(7)+(8)+(9)	S	**10000111101**	(11)
	C	**00001001000**	(12)
(10)+(11)+(12)	Σ	**110010101101**	



Zur Addition der Teilprodukte können Volladdierer genutzt werden.