ARITHMETIK IN ZAHLENSYSTEMEN MIT GANZZAHLIGER BASIS (A318)

Referenten:

Jonas Nögel - Selim Mert Kaştan - Maxim Balajan

EINLEITUNG

- Konventionelles Zahlensystem: Basis 10, Ziffern 0 – 9
  - Nur eines von (unendlich) Vielen
  - Beispiele für andere Zahlensysteme: Binär- (Basis = 2), Oktal- (8), Hexadezimalsystem (16)
- Theoretisch können Basis und Alphabet beliebig gewählt werden
- Aber: Meist nur Unterstützung für genannte Systeme
- Idee: Implementierung einer Funktion, die mit jeglichem System zurecht kommt

EINLEITUNG

- Signatur:

```c
void arith_op_any_base(int base, const char* alph, const char* z1,
    const char* z2, char op, char* result)
```

- Parameter:
  - base: Basis des Zahlensystems
  - alph: Alphabet des Zahlensystems
  - z1 & z2: Operanden der durchzuführenden Operation
  - Op: Durchzuführende Operation (‘+‘, ‘-‘ oder ‘*‘)
  - Result: Buffer, an dem Ergebnis gespeichert werden soll

LÖSUNGSANSATZ

Inhalt:

- *Rahmenbedingungen*
- *Addition und Subtraktion*
- *Multiplikation*
- *Bewertung*

RAHMENBEDINGUNGEN

Erlaubte Eingaben:

- *Alphabet:*
  - Kein Vorkommen des Null-Byte und des Minus-Zeichen im Alphabet
  - Kein wiederholtes Aufkommen der selben Zeichen
- *Basis:*
  - $B = [-254, 254] \setminus \{-1, 0, 1\}$
  - Absoluter Wert der Basis gleich der Länge des Alphabetes
- *Zahlen:*
  - Nur Zeichen des Alphabets
  - Ausnahme bei positiven Basen: Minus-Zeichen am Anfang der Zahlen

ADDITION UND SUBTRAKTION

Umsetzung:

- ❿ Implementierung der schriftlichen Addition bzw. Subtraktion
- ❿ Umwandeln der Ziffern in numerische Werte zur Verrechnung und anschließendes Zurückwandeln
- Merken und anschließende Dazurechnen der Carry-Werte

	437		317
+	346	-	258
c:	010	c:	110
=	783	=	59

Abbildung 1: Schriftliche Addi-/Subtraktion mit Zahlen der Basis 10

ADDITION UND SUBTRAKTION

Sonderfälle:

- ❿ *Negative Basen:*
  - Verändern der VZ für alle Carry-Werte
- ❿ *Subtraktion bei pos. Basen:*
  - Subtraktion einer kleineren von einer größeren Zahl nur indirekt möglich

	50			380
+	50		-	527
c:	1100		c:	1010
=	1900		=	1873

Abbildung 2: Schriftliche Addi-/Subtraktion mit Zahlen der Basis -10

	53			104
-	104		-	53
c:	...11010		c:	100
=	...99849		=	-51

Abbildung 3: Subtraktion einer kleinern Zahl von einer größeren (Basis -10)

MULTIPLIKATION

Umsetzung:

- Erneutes Implementieren der schriftlichen Multiplikation
- Multiplizieren aller Ziffern einer Zahl mit zweiter Zahl und anschließende Addition aller Teilergebnisse
- Gleiche Regeln für Carry-Werte wie bei Addition und Subtraktion

	$356 \cdot 312$
+	106800
+	3560
+	712
c:	012000
=	111072

	$50 \cdot 50$
+	18500
+	0
c:	00000
=	18500

Abbildung 4: Schriftliche Multiplikation mit Zahlen der Basis ± 10

BEWERTUNG: LÖSUNGSANSATZ

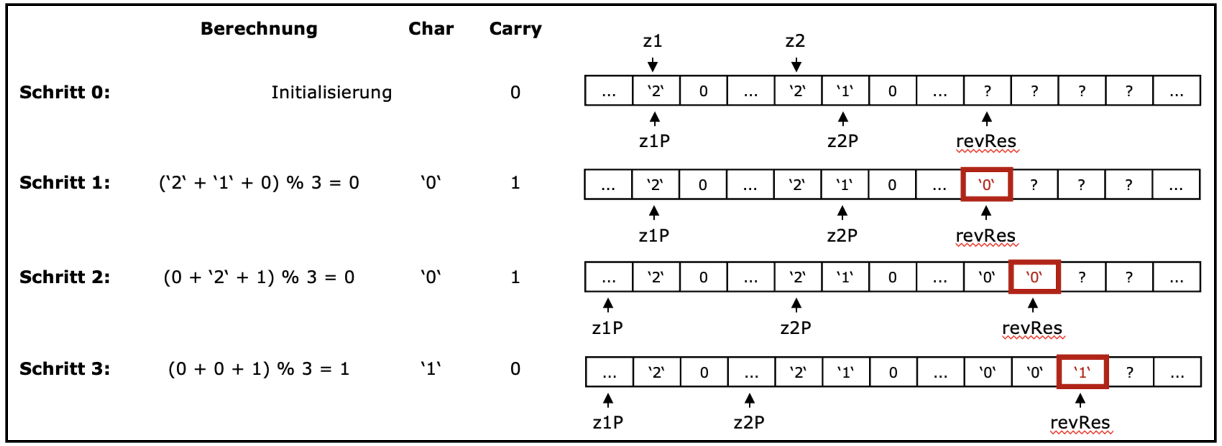
- Einfacher Algorithmus zur Umsetzung von arithmetischen Operationen
- Im Gegensatz zu einer direkten Konvertierung auf alle Zahlenbereiche anwendbar
- Relativ gute Laufzeit
  - **Ausnahme:** Multiplikation (Wiederholte Addition)
  - *Lösungsansatz:* Hybrid-Implementierung der direkten Konvertierung mit der bereits beschriebenen Implementierung

KORREKTHEIT

- Da Eingabewerte ganzzahlig sind, und Operation ‘+‘, ‘-‘ oder ‘*‘ ist, ist das Ergebnis ebenfalls ganzzahlig
- Korrektheit: Ausgegebenes und erwartetes Ergebnis stimmen komplett (inkl. Einerstelle) überein
  → Korrektheit impliziert Genauigkeit
- Rechenalgorithmen nicht kritisch, allerdings deren Umsetzung und Anwendung auf Strings beliebiger Länge

KORREKTHEIT

- Idee: „Abarbeiten“ der String-Zahlen durch zwei voneinander unabhängige Pointer
- Beispiel: Basis = 3, Alphabet =„012“, Operation: „2“ + „21“ = „100“



- Bei Multiplikation: z1 verändert sich erst, nachdem z2 die ganze Zahl abgelaufen ist

KORREKTHEIT

Referenzimplementierung: Konvertieren der Werte in Dezimal-system, und Berechnung durch Computer

- Problem: Darstellungsbereich der Datentypen ist begrenzt (z.B. int64_t auf $\pm 2^{63}-1$)
- Idee: Ähnlich zu Hauptimplementierung, Unterteilen der Zahlen in darstellbare Teile
- Beispiel: Basis 16, Länge der Teile: 2

  0x12AB + 0xAB34 = 0xBDDF

  Schritt 1: 0xAB + 0x34 = 171 + 52 = 223 = DF

  Schritt 2: 0x12 + 0xAB = 18 + 171 = 189 = BD
- Wie soll Länge der darstellbaren Teile gewählt werden?

Addition/Subtraktion: $\lfloor \log_{basis}(2^{63}-1) \rfloor$ Multiplikation: $\left\lfloor \frac{\log_{basis}(2^{63}-1)}{2} \right\rfloor$

KORREKTHEIT

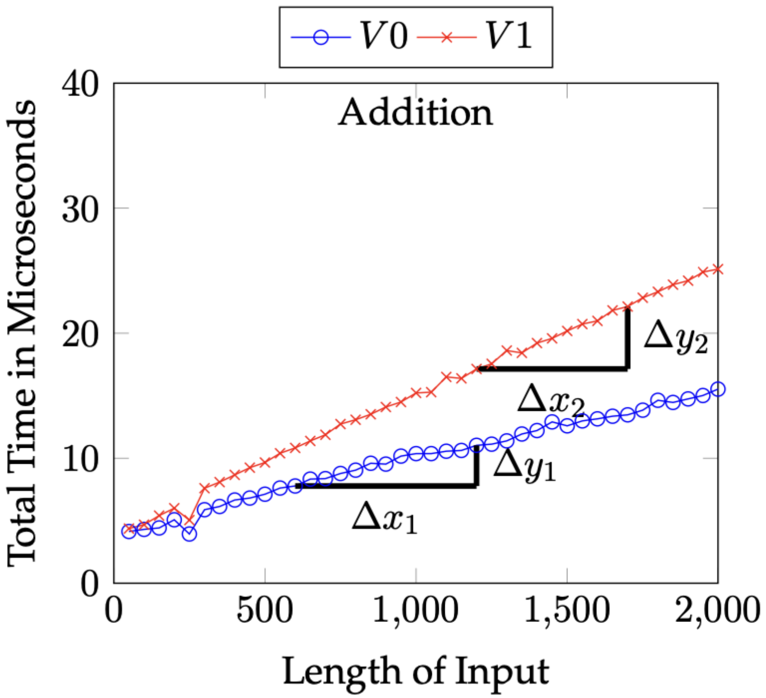
Testen der beiden Implementierungen im Hinblick auf

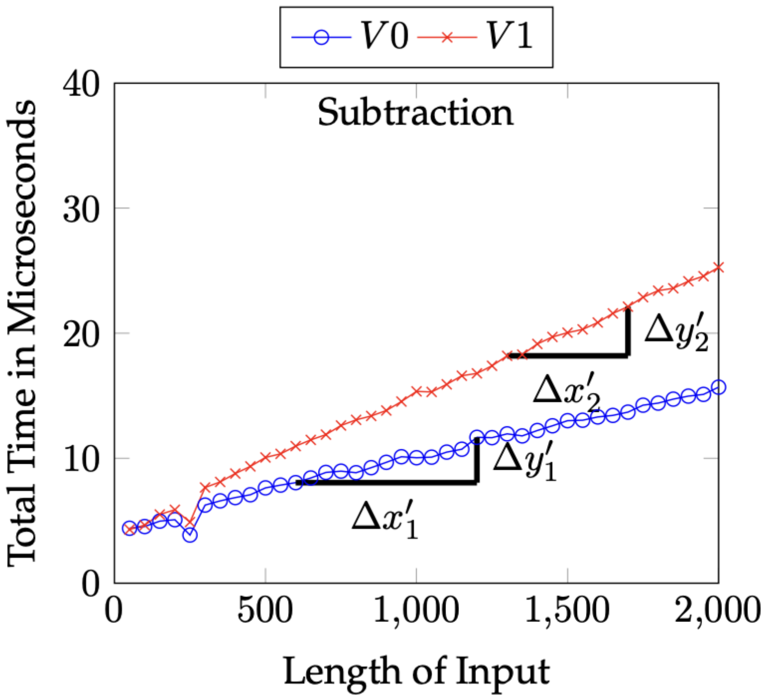
1. Generelle Korrektheit
   a) Basen ungleich 10
   b) Operanden verschiedener Länge
   c) Vorangestellte Nullen
2. Korrektheit bei sehr großen Werten ($>2^{63}$)

PERFORMANCE ANALYSE

- *Hardware:* MacBook Air M1
- *Compiler-Stufe:* O3
- *Basis:* 50
- 15 unterschiedliche Zahlenpaare 15-mal berechnet
- V0 für Hauptimplementierung ,
  V1 für Referenzimplementierung

ADDITION UND SUBTRAKTION





$$\Delta_1 = \frac{\Delta y_1}{\Delta x_1} = \frac{11.03 - 7.77}{1200 - 600} = \frac{3.26}{600} \approx 0.0054$$

$$\Delta_2 = \frac{\Delta y_2}{\Delta x_2} = \frac{22.13 - 17.14}{1700 - 1200} = \frac{4.99}{500} \approx 0.01$$

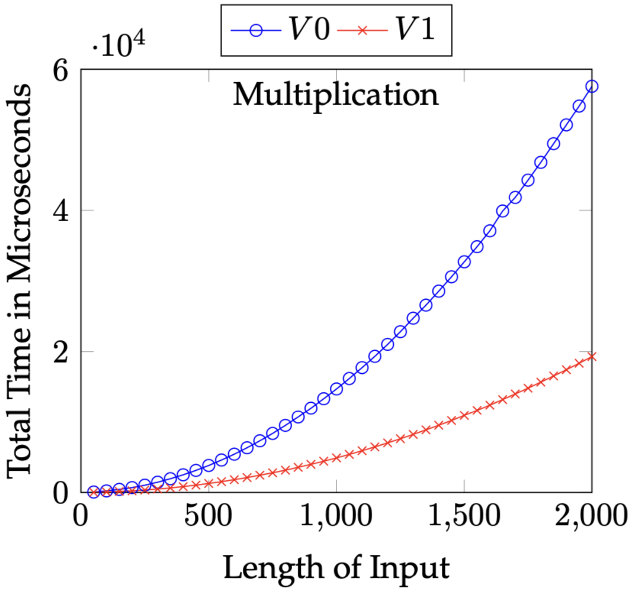
$$\Delta_1' = \frac{\Delta y_2'}{\Delta x_2'} = \frac{22.12 - 18.19}{1700 - 1300} = \frac{3.93}{400} \approx 0.01$$

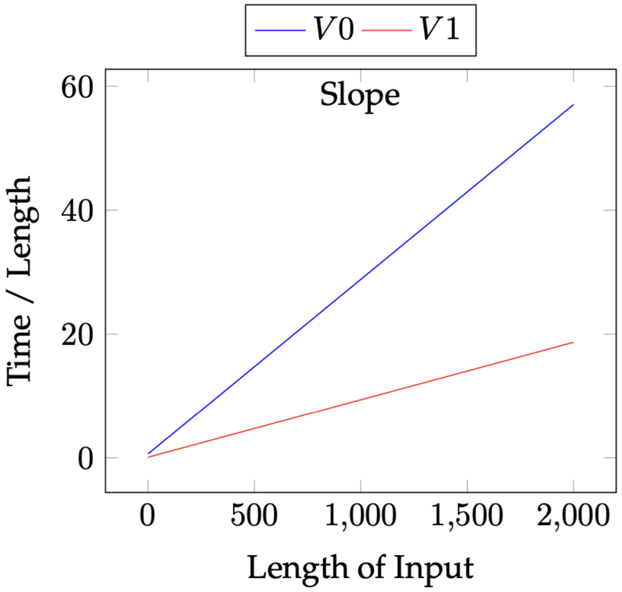
$$\Delta_2' = \frac{\Delta y_1'}{\Delta x_1'} = \frac{11.67 - 8.04}{1200 - 600} = \frac{3.63}{600} \approx 0.006$$

$$\Delta = \frac{\Delta_1}{\Delta_2} \approx 0.5 \qquad (5) \qquad \Delta' = \frac{\Delta_1'}{\Delta_2'} \approx 0.5$$

- Lineare Laufzeit
- O(n)
- V1 halb so schnell wie V0
  -> V0 bei Addi/-Subtraktion bevorzugt

MULTIPLIKATION





$$f_0(x) = 0.01411x^2 + 0.6264x - 6.411$$
$$f_1(x) = 0.0046464x^2 + 0.1054x + 13.646$$

$$f_0'(x) = 0.02822x + 0.6264$$

$$f_1'(x) = 0.0092928x + 0.1054$$

- Quadratische Laufzeit
- O(n^2)
- V1 effizienter als V0
  -> V1 bei Multiplikation bevorzugt

ZUSAMMENFASSUNG

- Beide Implementierung mit verschiedenen Ansätzen und unterschiedlichen Resultaten
- Korrekte Implementierung durch diverse Tests sichergestellt

  -> Mögliches Zusammenführen beider Implementierungen

  bei einer potentiellen Veröffentlichung

FRAGEN?

BACK UP

BACK UP

1. Testen der generellen Korrektheit

```
Jonas@MacBook-Pro-Jonas Implementierung % ./main -b'-'8 -a01234567 -o'+' 000123 0456 -V0
Result: 000123 + 0456 = 561
Jonas@MacBook-Pro-Jonas Implementierung % ./main -b'-'8 -a01234567 -o'+' 000123 0456 -V1
Result: 000123 + 0456 = 561
Jonas@MacBook-Pro-Jonas Implementierung % ./main -b'-'8 -a01234567 -o'-' 000123 0456 -V0
Result: 000123 - 0456 = 1665
Jonas@MacBook-Pro-Jonas Implementierung % ./main -b'-'8 -a01234567 -o'-' 000123 0456 -V1
Result: 000123 - 0456 = 1665
Jonas@MacBook-Pro-Jonas Implementierung % ./main -b'-'8 -a01234567 -o'*' 000123 0456 -V0
Result: 000123 * 0456 = 32112
Jonas@MacBook-Pro-Jonas Implementierung % ./main -b'-'8 -a01234567 -o'*' 000123 0456 -V1
Result: 000123 * 0456 = 32112
```

2. Testen der Korrektheit bei sehr großen Werten ($>2^{63}$)

```
Jonas@MBP-Jonas Implementierung % ./main -b16 -a0123456789ABCDEF -o'+' FEFEFEFEFEFEFEFEFE 010101010101010101 -V0
Result: FEFEFEFEFEFEFEFEFE + 010101010101010101 = FFFFFFFFFFFFFFFFFF
Jonas@MBP-Jonas Implementierung % ./main -b16 -a0123456789ABCDEF -o'+' FEFEFEFEFEFEFEFEFE 010101010101010101 -V1
Result: FEFEFEFEFEFEFEFEFE + 010101010101010101 = FFFFFFFFFFFFFFFFFF
Jonas@MBP-Jonas Implementierung % ./main -b'-'16 -a0123456789ABCDEF -o'-' FEFEFEFEFEFEFEFEFE 010101010101010101 -V0
Result: FEFEFEFEFEFEFEFEFE - 010101010101010101 = FDFDFDFDFDFDFDFDFD
Jonas@MBP-Jonas Implementierung % ./main -b'-'16 -a0123456789ABCDEF -o'-' FEFEFEFEFEFEFEFEFE 010101010101010101 -V1
Result: FEFEFEFEFEFEFEFEFE - 010101010101010101 = FDFDFDFDFDFDFDFDFD
Jonas@MBP-Jonas Implementierung % ./main -b16 -a0123456789ABCDEF -o'*' FEFEFEFEFEFEFEFEFE 010101010101010101 -V0
Result: FEFEFEFEFEFEFEFEFE * 010101010101010101 = FFFEFDFCFBFAF9F8F5F6F7F8F9FAFBFCFE
Jonas@MBP-Jonas Implementierung % ./main -b16 -a0123456789ABCDEF -o'*' FEFEFEFEFEFEFEFEFE 010101010101010101 -V1
Result: FEFEFEFEFEFEFEFEFE * 010101010101010101 = FFFEFDFCFBFAF9F8F5F6F7F8F9FAFBFCFE
```