Regeln für arithmetische Operationen

Dieses Kapitel behandelt folgende Themen:

- Initialisierung von Feldern
- Kompatibilitätsregeln zur Datenübertragung
- Abschneiden und Runden von Feldwerten
- Format/Länge von Ergebnisfeldern bei arithmetischen Operationen
- Arithmetische Operationen mit Gleitkomma-Zahlen
- Arithmetische Operationen mit Datum und Zeit
- Formatwahl im Hinblick auf die Verarbeitungszeit
- Genauigkeit von Ergebnissen bei arithmetischen Operationen
- Fehlerbedingungen bei arithmetischen Operationen
- Verarbeitung von Arrays

Initialisierung von Feldern

Ein Feld (Datenbankfeld oder Benutzervariable), das in einer arithmetischen Operation als Operand verwendet werden soll, muss mit einem der folgenden Formate definiert werden:

Format	
N	Numerisch ungepackt
P	Numerisch gepackt
I	Integer (Ganzzahl)
F	Floating Point (Gleitkomma)
D	Datum
T	Zeit

Anmerkung:
Reporting Mode: Ein Feld, das in einer arithmetischen Operation als Operand verwendet werden soll, muss vorher definiert werden. Benutzervariablen oder Datenbankfelder, die in einer arithmetischen Operation als Ergebnisfeld verwendet werden, müssen nicht vorher definiert werden.

Sobald ein Programm zur Ausführung aufgerufen wird, werden alle im `DEFINE DATA`-Bereich definierten Benutzervariablen und Datenbankfelder mit den entsprechenden Leer- bzw. Nullwerten initialisiert.

Kompatibilitätsregeln zur Datenübertragung

Die Datenübertragung erfolgt mit einem `MOVE`- oder `COMPUTE`-Statement. Die folgende Tabelle fasst die Kompatibilitätsregeln zur Datenübertragung der Formate zusammen, die ein Operand annehmen kann.

Ausgangsfeld	**Zielfeld**												
	N oder P	**A**	**U**	**B*n* (*n*<5)**	**B*n* (*n*>4)**	**I**	**L**	**C**	**D**	**T**	**F**	**G**	**O**
N oder P	Y	[2]	[14]	[3]	-	Y	-	-	-	Y	Y	-	-
A	-	Y	[13]	[1]	[1]	-	-	-	-	-	-	-	-
U	-	[11]	Y	[12]	[12]	-	-	-	-	-	-	-	-
B*n* (*n*<5)	[4]	[2]	[14]	[5]	[5]	Y	-	-	-	Y	Y	-	-
B*n* (*n*>4)	-	[6]	[15]	[5]	[5]	-	-	-	-	-	-	-	-
I	Y	[2]	[14]	[3]	-	Y	-	-	-	Y	Y	-	-
L	-	[9]	[16]	-	-	-	Y	-	-	-	-	-	-
C	-	-	-	-	-	-	-	Y	-	-	-	-	-
D	Y	[9]	[16]	Y	-	Y	-	-	Y	[7]	Y	-	-
T	Y	[9]	[16]	Y	-	Y	-	-	[8]	Y	Y	-	-
F	Y	[9] [10]	[10] [16]	[3]	-	Y	-	-	-	Y	Y	-	-
G	-	-	-	-	-	-	-	-	-	-	-	Y	-
O	-	-	-	-	-	-	-	-	-	-	-	-	Y

Legende:

Y	Übertragung möglich.
-	Übertragung nicht möglich.
[ ]	Übertragung möglich. Die Ziffern in eckigen Klammern [ ] beziehen sich auf die entsprechende Regel für die Übertragung (siehe unten).

Umsetzung von Daten in ein anderes Format

Bei der Umsetzung von Werten in ein anderes Format gelten folgende Regeln:

1. **Von Alphanumerisch (A) in Binär (B):**
   Der Wert wird Byte für Byte von links nach rechts übertragen. Je nach Länge des Zielfeldes und der Anzahl der Bytes wird der Wert entweder abgeschnitten oder der Rest des Feldes mit Leerzeichen aufgefüllt.

2. **Von Numerisch (N), Gepackt (P), Ganzzahl (I) und Binär (B) mit 1-4 Bytes Länge in Alphanumerisch:**
   Der Wert wird in ungepacktes Format umgesetzt und linksbündig in das Zielfeld übertragen, wobei vorangestellte Nullen weggelassen werden und der Rest des Feldes mit Leerzeichen aufgefüllt wird. Bei negativen numerischen Werten wird das Vorzeichen in die Hexadezimalnotation `Dx` umgesetzt.

Ein Komma (Dezimalpunkt) im Ausgangswert wird nicht berücksichtigt, und alle Stellen vor und nach dem Komma werden als ganze Zahl interpretiert.

3. **Von Numerisch (N), Gepackt (P), Ganzzahl (I), Gleitkomma (F) in Binär (B) mit 1-4 Bytes Länge:**
   Der Wert wird in binäres Format umgesetzt (4 Bytes). Ein Komma (Dezimalpunkt) wird ignoriert, die Stellen vor und nach dem Komma werden als ganze Zahl behandelt. Je nach Vorzeichen ist die Binärzahl entweder positiv oder das Zweierkomplement des Wertes.

4. **Von Binär (B) mit 1-4 Bytes Länge in Numerisch (N):**
   Der Wert wird umgesetzt und rechtsbündig übertragen, der Rest des Feldes wird mit Nullen aufgefüllt. Binäre Werte von 1 bis 3 Bytes Länge werden immer als positiv interpretiert. Bei 4 Byte langen binären Werten bestimmt das erste (linke) Bit das Vorzeichen: 1 = negativ, 0 = positiv. Ein Komma (Dezimalpunkt) im Zielfeld wird nicht berücksichtigt, und alle Stellen vor und nach dem Komma werden als ganze Zahl interpretiert.

5. **Von Binär (B) in Binär (B):**
   Der Wert wird Byte für Byte von rechts nach links übertragen, und der Rest des Feldes wird mit Nullen aufgefüllt.

6. **Von Binär (B) mit mehr als 4 Bytes in Alphanumerisch (A):**
   Der Wert wird Byte für Byte von links nach rechts übertragen. Je nach Länge des Zielfeldes und der Anzahl der Bytes wird der Wert entweder abgeschnitten oder der Rest des Feldes mit Leerzeichen aufgefüllt.

7. **Von Datum (D) in Zeit (T):**
   Das Datum wird in Zeit umgesetzt, und zwar ausgehend von der Zeit `00:00:00:0`.

8. **Von Zeit (T) in Datum (D):**
   Die Zeitkomponente wird abgeschnitten und nur die Datumskomponente des Zeitfeldes wird in das Datumsfeld übertragen.

9. **Von Logisch (L), Datum (D), Zeit (T), Gleitkomma (F) in Alphanumerisch (A):**
   Der Wert wird in Anzeigeform umgesetzt und linksbündig übertragen.

10. **Gleitkomma (F):**
    Wird ein Gleitkomma-Wert in ein alphanumerisches oder Unicode-Feld übertragen, das zu kurz ist, wird die Mantisse entsprechend gekürzt.

11. **Von Unicode (U) in Alphanumerisch (A):**
    Der Unicode-Wert wird anhand der Library ICU (International Components for Unicode) entsprechend der voreingestellten (Default-)Codepage (Wert der Systemvariablen `*CODEPAGE`) in alphanumerische Zeichencodes umgesetzt. Je nach Länge des Zielfeldes und der Anzahl der Bytes wird das Ergebnis entweder abgeschnitten oder der Rest des Feldes mit Leerzeichen aufgefüllt.

12. **Von Unicode (U) in Binär (B):**
    Der Wert wird Code Unit für Code Unit von links nach rechts verschoben. Je nach Länge des Zielfeldes und der Anzahl der Bytes wird das Ergebnis entweder abgeschnitten oder der Rest des Feldes mit Leerzeichen aufgefüllt. Die Länge des binären Zielfeldes muss geradezahlig sein.

13. **Von Alphanumerisch (A) in Unicode (U):**
    Der alphanumerische Wert wird unter Benutzung der Library ICU (International Components for Unicode) von der voreingestellten (Default-)Codepage in einen Unicode-Wert umgesetzt. Je nach Länge

des Zielfeldes und der Anzahl der Code Units wird das Ergebnis entweder abgeschnitten oder der Rest des Feldes mit Leerzeichen aufgefüllt.

14. **Von N, P, I und Binär (Länge 1–4) in Unicode (U):**
    Der Wert wird in ungepacktes Format konvertiert, aus dem dann ein alphanumerischer Wert durch die Unterdrückung von führenden Nullen erhalten werden kann. Bei negativen numerischen Werten wird das Vorzeichen in die hexadezimale Notation `Dx` umgesetzt. Ein Dezimalpunkt im numerischen Wert wird ignoriert. Alle Ziffern vor und nach dem Dezimalpunkt werden als ein Ganzzahlwert (Integer) behandelt. Der Ergebniswert wird von alphanumerisch in Unicode umgesetzt. Je nach Länge des Zielfeldes und der Anzahl der Code Units wird das Ergebnis entweder abgeschnitten oder der Rest des Feldes mit Leerzeichen aufgefüllt.

15. **Von Binär (B) mit mehr als 4 Bytes in Unicode (U):**
    Der Wert wird Byte für Byte von links nach rechts verschoben. Je nach Länge des Zielfeldes und der Anzahl der Bytes wird das Ergebnis entweder abgeschnitten oder der Rest des Feldes mit Leerzeichen aufgefüllt. Die Länge des binären Ausgangsfeldes muss geradzahlig sein.

16. **Von L, D, T, F in U:**
    Die Werte werden in ein alphanumerisches Anzeigeformat konvertiert. Der Ergebniswert wird von alphanumerisch in Unicode umgesetzt und linksbündig ausgerichtet.

Wenn das Ausgangs- und Zielformat identisch sind, kann je nach der Länge und der Anzahl der Bytes (Format A und B) oder Code-Einheiten (Format U) das Ergebnis abgeschnitten oder mit Leerzeichen (Format A und U) oder führenden binären Nullen (Format B) aufgefüllt werden.

Siehe auch *Dynamische Variablen benutzen*.

Abschneiden und Runden von Feldwerten

Die folgenden Regeln gelten für das Abschneiden und Runden von Feldwerten:

- Numerische Felder: vorangestellte Stellen dürfen nur abgeschnitten werden, falls ihr Wert Null ist. Stellen nach einem ausdrücklich angegebenen oder implizierten Komma (Dezimalpunkt) dürfen abgeschnitten werden.

- Alphanumerische Felder: Nachfolgende Stellen dürfen abgeschnitten werden.

- Bei Verwendung der Option `ROUNDED` wird die letzte Stelle im Feld aufgerundet, falls die erste abgeschnittene Stelle größer/gleich 5 ist. Zur Ergebnisgenauigkeit einer Division siehe auch Abschnitt *Genauigkeit von Ergebnissen bei arithmetischen Operationen*.

Format/Länge von Ergebnisfeldern bei arithmetischen Operationen

Die folgende Tabelle zeigt Format/Länge von Ergebnisfeldern bei arithmetischen Operationen:

	I1	I2	I4	N oder P	F4	F8
I1	I1	I2	I4	P*	F4	F8
I2	I2	I2	I4	P*	F4	F8
I4	I4	I4	I4	P*	F4	F8
N oder P	P*	P*	P*	P*	F4	F8
F4	F4	F4	F4	F4	F4	F8
F8	F8	F8	F8	F8	F8	F8

Auf Großrechnern wird Format/Länge F8 anstatt F4 für eine verbesserte Ergebnisgenauigkeit einer arithmetischen Operation benutzt.

P* ergibt sich aus der ganzzahligen Länge und Genauigkeit der einzelnen Operanden je nach Operation (siehe Abschnitt *Genauigkeit von Ergebnissen bei arithmetischen Operationen*.

Für Format I gelten die folgenden dezimalen Ganzzahl-Längen und möglichen Werte:

Format/Länge	Dezimale Ganzzahl-Länge	Mögliche Werte
`I1`	`3`	`-128` bis `127`
`I2`	`5`	`-32768` bis `32767`
`I4`	`10`	`-2147483648` bis `2147483647`

Arithmetische Operationen mit Gleitkomma-Zahlen

Folgende Themen werden behandelt:

- Einige allgemeine Hinweise
- Genauigkeit von Gleitkomma-Zahlen
- Konvertierung in Gleitkomma-Darstellung
- Plattform-abhängige Unterschiede

Einige allgemeine Hinweise

Gleitkomma-Zahlen (Format F) werden ebenso wie Ganzzahlen (Format I) als Summe von Zweierpotenzen dargestellt, wohingegen ungepackte und gepackte Zahlen (Formate N und P) als Summe von Zehnerpotenzen dargestellt werden.

Bei ungepackten oder gepackten Zahlen ist die Position des Dezimalkommas fest. Bei Gleitkomma-Zahlen dagegen ist (wie der Name schon andeutet) die Position des Dezimalkommas "gleitend", d.h. seine Position ist nicht fest, sondern hängt vom tatsächlichen Wert der Zahl ab.

Gleitkomma-Zahlen sind unverzichtbar bei der Berechnung trigonometrischer und mathematischer Funktionen wie etwa Sinus oder Logarithmus.

Genauigkeit von Gleitkomma-Zahlen

Die Genauigkeit von Gleitkomma-Zahlen an sich ist begrenzt:

- Bei einer Variablen mit Format/Länge F4 ist die Genauigkeit auf etwa 7 Stellen begrenzt.
- Bei einer Variablen mit Format/Länge F8 ist die Genauigkeit auf 15 Stellen begrenzt.

Werte mit einer größeren Anzahl signifikanter Stellen lassen sich nicht exakt als Gleitkomma-Zahlen darstellen. Unabhängig von der Zahl zusätzlicher Vor- oder Nachkommastellen kann eine Gleitkomma-Zahl nur die ersten 7 bzw. 15 Stellen abdecken.

Eine Ganzzahl lässt sich nur exakt in einer Variablen mit Format/Länge F4 darstellen, wenn ihr absoluter Wert nicht größer als $2^{23}-1$ ist.

Konvertierung in Gleitkomma-Darstellung

Wenn ein alphanumerischer, ungepackter numerischer oder gepackter numerischer Wert in Gleitkomma-Format umgesetzt wird (zum Beispiel bei einer Zuweisung), muss auch die Darstellungsform geändert werden, d.h. eine Summe von Zehnerpotenzen muss in eine Summe von Zweierpotenzen konvertiert werden.

Folglich lassen sich nur Zahlen, die als endliche Summe von Zweierpotenzen darstellbar sind, exakt darstellen; alle anderen Zahlen lassen sich nur näherungsweise darstellen.

Beispiele:

Diese Zahl hat eine exakte Gleitkomma-Darstellung:

$$1.25 = 2^{0} + 2^{-2}$$

Diese Zahl ist eine periodische Gleitkomma-Zahl ohne exakte Darstellung:

$$1.2 = 2^{0} + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-8} + 2^{-11} + 2^{-12} + \ldots$$

Daher kann die Konvertierung von alphanumerischen, ungepackt numerischen oder gepackt numerischen Werten in Gleitkomma-Werte, und umgekehrt, zu kleineren Fehlern führen.

Plattform-abhängige Unterschiede

Aufgrund der unterschiedlichen Hardware-Architektur ist die Darstellung von Gleitkomma- Zahlen auf Großrechnern anders als auf anderen Plattformen. Dies erklärt, warum dieselbe Anwendung bei Gleitkomma-Berechnungen auf verschiedenen Plattformen andere Ergebnisse liefert.

Der mögliche Wertebereich auf einem Großrechner ist (ca.):

- $\pm 1.17 * 10^{-38}$ to $\pm 3.40 * 10^{38}$ für F4-Variablen,
- $\pm 2.22 * 10^{-308}$ to $\pm 1.79 * 10^{308}$ für F8-Variablen.

Anmerkung:
Die von Ihrem Taschenrechner verwendete Darstellung kann sich ebenfalls von der Ihres Computers unterscheiden,und die Ergebnisse für die gleiche Berechnung können daher auch hier unterschiedlich sein.

Arithmetische Operationen mit Datum und Zeit

Mit Feldern der Formate D (Datum) und T (Time = Zeit) sind nur Addition, Subtraktion, Multiplikation und Division erlaubt. Multiplikation und Division sind nur bei Zwischenergebnissen von Addition und Subtraktion zulässig.

Datums-/Zeitwerte können addiert bzw. voneinander subtrahiert werden; oder Ganzzahl-Werte (ohne Nachkommastellen) können zu/von Datums-/Zeitwerten addiert/subtrahiert werden. Solche ganzzahligen Werte können in Feldern der Formate N, P, I, D oder T enthalten sein.

Die Zwischenergebnisse einer solchen Addition oder Subtraktion können als Multiplikand oder Dividend in einer nachfolgenden Operation verwendet werden.

Von ganzzahligen Werten, die zu einem Datumswert addiert oder von einem Datumswert subtrahiert werden, wird angenommen, dass es sich um Tage handelt. Von ganzzahligen Werten, die zu einem Zeitwert addiert oder von einem Zeitwert subtrahiert werden, wird angenommen, dass es sich um Zehntelsekunden handelt.

Bei arithmetischen Operationen mit Datum und Zeit gelten gewisse Einschränkungen, und zwar aufgrund von Naturals interner Behandlung von Datums- und Zeitarithmetik, wie im folgenden erläutert.

Intern behandelt Natural eine arithmetische Operation mit Datums- bzw. Zeitvariablen wie folgt:

COMPUTE *result-field = operand1 +/- operand2*

Das obige Statement wird aufgelöst als:

1. `intermediate-result = operand1 +/- operand2`
2. `result-field = intermediate-result`

Das heißt, zunächst berechnet Natural das Ergebnis der Addition/Subtraktion, und erst danach weist es das Ergebnis dem Ergebnisfeld zu.

Komplexere arithmetische Operationen werden nach dem gleichen Muster aufgelöst:

COMPUTE *result-field = operand1 +/- operand2 +/- operand3 +/- operand4*

Das obige Statement wird aufgelöst als:

1. `intermediate-result1 = operand1 +/- operand2`
2. `intermediate-result2 = intermediate-result1 +/- operand3`
3. `intermediate-result3 = intermediate-result2 +/- operand4`
4. `result-field = intermediate-result3`

Die Auflösung bei der Multiplikation und Division ist ähnlich wie die Auflösung bei der Addition und Subtraktion.

Das interne Format eines solchen Zwischenergebnisses (*intermediate-result*) hängt vom Format der einzelnen Operanden ab, wie die folgenden Tabellen zeigen.

Addition

Die folgende Tabelle zeigt das Format vom Zwischenergebnis einer Addition (*intermediate-result = operand1 + operand2*):

Format von *operand1*	**Format von *operand2***	**Format von *intermediate-result***
D	D	Di
D	T	T
D	Di, Ti, N, P, I	D
T	D, T, Di, Ti, N, P, I	T
Di, Ti, N, P, I	D	D
Di, Ti, N, P, I	T	T
Di, N, P, I	Di	Di
Ti, N, P, I	Ti	Ti
Di	Ti, N, P, I	Di
Ti	Di, N, P, I	Ti

Subtraktion

Die folgende Tabelle zeigt das Format des Zwischenergebnisses einer Subtraktion (*intermediate-result = operand1 - operand2*):

Format von *operand1*	**Format von *operand2***	**Format von *intermediate-result***
D	D	Di
D	T	Ti
D	Di, Ti, N, P, I	D
T	D, T	Ti
T	Di, Ti, N, P, I	T
Di, N, P, I	D	Di
Di, N, P, I	T	Ti
Di	Di, Ti, N, P, I	Di
Ti	D, T, Di, Ti, N, P, I	Ti
N, P, I	Di, Ti	P12

Multiplikation oder Division

Die folgende Tabelle zeigt das Format des Zwischenergebnisses einer Multiplikation: (*intermediate-result = operand1 * operand2*) oder Division (*intermediate-result = operand1 / operand2*):

Format von *operand1*	Format von *operand2*	Format von *intermediate-result*
D	D, Di, Ti, N, P, I	Di
D	T	Ti
T	D, T, Di, Ti, N, P, I	Ti
Di	T	Ti
Di	D, Di, Ti, N, P, I	Di
Ti	D	Di
Ti	Di, T, Ti, N, P, I	Ti
N, P, I	D, Di	Di
N, P, I	T, Ti	Ti

Interne Zuweisungen

Di ist ein Wert im internen Datumsformat; Ti ist ein Wert im internen Zeitformat; solche Werte können zwar in weiteren arithmetischen Datums-/Zeitoperationen verwendet werden, aber sie können keinem Ergebnisfeld vom Format D zugewiesen werden (siehe Zuweisungstabelle unten).

Bei komplexen arithmetischen Operationen, bei denen ein Zwischenergebnis im internen Format Di bzw. Ti als Operand für eine weitere Addition/Subtraktion/Multiplikation/Division verwendet wird, wird davon ausgegangen, dass es Format D bzw. T hat.

Die folgende Tabelle zeigt, welche Zwischenergebnisse intern welchen Ergebnisfeldern zugewiesen werden können (*result-field = intermediate-result*).

Format von *result-field*	Format von *intermediate-result*	Zuweisung möglich
D	D, T	ja
D	Di, Ti, N, P, I	nein
T	D, T, Di, Ti, N, P, I	ja
N, P, I	D, T, Di, Ti, N, P, I	ja

Ein Ergebnisfeld vom Format D oder T darf keinen negativen Wert enthalten.

Beispiele 1 und 2 (ungültig):

```
COMPUTE DATE1 (D) = DATE2 (D) + DATE3 (D)
COMPUTE DATE1 (D) = DATE2 (D) - DATE3 (D)
```

Diese Operationen sind nicht möglich, da das Zwischenergebnis der Addition bzw. Subtraktion Format Di hätte, und ein Wert vom Format D*i* keinem Ergebnisfeld vom Format D zugewiesen werden kann.

Beispiele 3 und 4 (ungültig):

```
COMPUTE DATE1 (D) = TIME2 (T) - TIME3 (T)
COMPUTE DATE1 (D) = DATE2 (D) - TIME3 (T)
```

Diese Operationen sind nicht möglich, da das Zwischenergebnis der Addition bzw. Subtraktion Format T*i* hätte, und ein Wert vom Format T*i* keinem Ergebnisfeld vom Format D zugewiesen werden kann.

Beispiel 5 (gültig):

```
COMPUTE DATE1 (D) = DATE2 (D) - DATE3 (D) + TIME3 (T)
```

Diese Operation ist möglich. Zunächst wird `DATE3` von `DATE2` subtrahiert, woraus sich ein Zwischenergebnis vom Format D*i* ergibt; dann wird dieses Zwischenergebnis zu `TIME3` hinzuaddiert, woraus sich ein Zwischenergebnis vom Format T ergibt; und schließlich wird dieses zweite Zwischenergebnis dem Ergebnisfeld `DATE1` zugewiesen.

Beispiele 6 und 7 (ungültig):

```
COMPUTE DATE1 (D) = DATE2 (D) + DATE3 (D) * 2
COMPUTE TIME1 (T) = TIME2 (T) - TIME3 (T) / 3
```

Diese Operationen sind nicht möglich, da die versuchte Multiplikation bzw. Division mit Datums-/Zeitfeldern und nicht mit Zwischenergebnissen durchgeführt wird.

Beispiel 8 (gültig):

```
COMPUTE DATE1 (D) = DATE2 (D) + (DATE3(D) - DATE4 (D)) * 2
```

Diese Operation ist möglich. Zunächst wird `DATE4` von `DATE3` subtrahiert, woraus sich ein Zwischenergebnis vom Format D*i* ergibt; dann wird dieses Zwischenergebnis mit 2 multipliziert, woraus sich ein Zwischenergebnis vom Format D*i* ergibt; dieses Zwischenergebnis wird zu `DATE2` addiert, woraus sich ein Zwischenergebnis vom Format D ergibt; und schließlich wird dieses dritte Zwischenergebnis dem Ergebnisfeld `DATE1` zugewiesen.

Wenn Sie einen Format-T-Wert einem Format-D-Feld zuweisen, müssen Sie dafür sorgen, dass der Zeitwert eine gültige Datumskomponente enthält.

Formatwahl im Hinblick auf die Verarbeitungszeit

Bei arithmetischen Operationen hat die Wahl der richtigen Feldformate starken Einfluss auf die Verarbeitungszeit:

Bei kaufmännischen Berechnungen empfiehlt es sich, nur Felder mit dem Format I (Integer) zu verwenden.

Bei wissenschaftlichen Berechnungen empfiehlt es sich, nur Felder mit dem Format F (Gleitkomma-Format) zu verwenden.

Werden die numerischen Formate N und P mit dem Format F vermischt, erfolgt intern eine Umsetzung in Format F; diese Umsetzung führt zu einer nicht unbeträchtlichen CPU-Beanspruchung. Daher sollte es möglichst vermieden werden, bei arithmetischen Operationen unterschiedliche Formate zu verwenden.

Genauigkeit von Ergebnissen bei arithmetischen Operationen

Operation	Stellen vor dem Komma	Stellen nach dem Komma
Addition/Subtraktion	Fi + 1 oder Si + 1 (das jeweils größere)	Fd oder Sd (das jeweils größere)
Multiplikation	Fi + Si + 2	Fd + Sd (höchstens 7)
Division	Fi + Sd	(siehe unten)
Potenzierung	15 - Fd (Siehe *Ausnahme* unten)	Fd
Quadratwurzel	Fi	Fd

Dabei ist:

F	Erster Operand
S	Zweiter Operand
R	Ergebnis
i	Stellen vor dem Komma (Dezimalpunkt)
d	Stellen nach dem Komma (Dezimalpunkt)

Ausnahme:

Wenn die Hochzahl eine oder mehrere Stellen hinter dem Komma (Dezimalpunkt) aufweist, wird die Potenzierung intern im Gleitkomma-Format ausgeführt und das Ergebnis hat ebenfalls Gleitkomma-Format. Weitere Informationen siehe Abschnitt *Arithmetische Operationen mit Gleitkomma-Zahlen*.

Nachkommastellen bei Divisionsergebnissen

Die Genauigkeit des Ergebnisses einer Division hängt davon ab, ob ein Ergebnisfeld vorhanden ist oder nicht:

- Ist ein Ergebnisfeld vorhanden, ist die Genauigkeit: `Rd` oder `Fd` (das jeweils größere) *.
- Ist kein Ergebnisfeld vorhanden, ist die Genauigkeit: `Fd` oder `Sd` (das jeweils größere) *.

* Bei Verwendung der `ROUNDED`-Option erhöht sich die Ergebnisgenauigkeit intern um eine Stelle, bevor das Ergebnis tatsächlich gerundet wird.

Ein Ergebnisfeld ist vorhanden (bzw. wird als vorhanden angenommen) in einem `COMPUTE`- und `DIVIDE`-Statement sowie in einer logischen Bedingung, in der die Division hinter dem Vergleichsoperator steht (z.B.: `IF #A = #B / #C THEN ...`).

Ein Ergebnisfeld ist nicht vorhanden (bzw. wird als nicht vorhanden angenommen) in einer logischen Bedingung, in der die Division vor dem Vergleichsoperator steht (z.B.: `IF #B / #C = #A THEN ...`).

Ausnahme:

Wenn Dividend und Divisor Ganzzahlen sind und mindestens eine davon eine Variable ist, dann ist auch das Divisionsergebnis eine Ganzzahl (unabhängig von der Genauigkeit des Ergebnisfeldes sowie der Verwendung der `ROUNDED`-Option).

Genauigkeit von Ergebnissen bei arithmetischen Ausdrücken

Die Genauigkeit von arithmetischen Ausdrücken, zum Beispiel `#A / (#B * #C) + #D * (#E - #F + #G)`, wird von der Auswertung der Ergebnisse von arithmetischen Operationen in ihrer Verarbeitungsreihenfolge hergeleitet. Weitere Informationen zu arithmetischen Ausdrücken siehe *arithmetic-expression* in der Beschreibung des `COMPUTE`-Statements.

Fehlerbedingungen bei arithmetischen Operationen

Bei einer Addition, Subtraktion, Multiplikation oder Division erhalten Sie einen Fehler, wenn das Ergebnis (insgesamt, d.h. vor und nach dem Komma) mehr als 31 Stellen hat.

Bei einer Potenzierung erhalten Sie unter einer der folgenden Bedingungen einen Fehler:

- wenn die Basis gepacktes Format hat und entweder das Ergebnis mehr als 16 Stellen oder ein Zwischenergebnis mehr als 15 Stellen hat;
- wenn die Basis Gleitkomma-Format hat und das Ergebnis größer ist als ca. $7 * 10^{75}$.

Verarbeitung von Arrays

Grundsätzlich gelten folgende Regeln:

- Alle Skalar-Operationen können auf Array-Elemente angewandt werden, die aus einer einzigen Ausprägung bestehen.
- Wenn eine Variable mit einem konstanten Wert definiert ist (z.B. `#FIELD (I2) CONSTANT <8>`), dann wird der Wert der Variablen bei der Kompilierung zugewiesen, und die Variable wird als Konstante behandelt. Falls eine solche Variable in einem Array-Index verwendet wird, bedeutet dies, dass die betreffende Dimension eine *bestimmte* Anzahl von Ausprägungen hat.
- Bei einer Zuweisung bzw. einem Vergleich zwischen zwei Arrays mit unterschiedlich vielen Dimensionen wird angenommen, dass die "fehlende" Dimension in dem Array mit weniger Dimensionen (1:1) ist.

  Beispiel: Wenn das Array `#ARRAY1 (1:2)` dem Array `#ARRAY2 (1:2,1:2)` zugewiesen wird, wird für `#ARRAY1` angenommen, dass es `#ARRAY1 (1:1,1:2)` ist.

Folgende Themen werden behandelt:

- Definition von Array-Dimensionen
- Zuweisungen bei Arrays

- Vergleiche mit Arrays
- Arithmetische Operationen mit Arrays

Definition von Array-Dimensionen

Die erste, zweite und dritte Dimension eines Arrays werden wie folgt definiert:

Dimensionen	Eigenschaften
3	#a3 (3. Dim., 2. Dim., 1. Dim.)
2	#a2 (2. Dim., 1. Dim.)
1	#a1 (1. Dim.)

Zuweisungen bei Arrays

Wenn Sie einen Array-Bereich einem anderen Array-Bereich zuweisen, erfolgt die Zuweisung Element für Element.

Beispiel:

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:5) INIT <10,20,30,40,50>
END-DEFINE
*
MOVE #ARRAY(2:4) TO #ARRAY(3:5)
/* is identical to
/* MOVE #ARRAY(2) TO #ARRAY(3)
/* MOVE #ARRAY(3) TO #ARRAY(4)
/* MOVE #ARRAY(4) TO #ARRAY(5)
/*
/* #ARRAY contains  10,20,20,20,20
```

Wenn Sie eine einzelne Ausprägung einem Array-Bereich zuweisen, wird jedes Element des Bereiches mit dem Wert der einzelnen Ausprägung gefüllt. (Bei einer mathematischen Funktion wird jedes Element des Bereiches mit dem Ergebnis der Funktion gefüllt.)

Bevor eine Zuweisung ausgeführt wird, werden die einzelnen Dimensionen der betroffenen Arrays miteinander verglichen, um zu prüfen, ob sie eine der unten aufgeführten Bedingungen erfüllen.

Die Dimensionen werden dabei unabhängig voneinander verglichen; d.h. die 1. Dimension des einen Arrays wird mit der 1. Dimension des anderen Arrays verglichen, die 2. Dimension des einen Arrays wird mit der 2. Dimension des anderen Arrays verglichen, und die 3. Dimension des einen Arrays wird mit der 3. Dimension des anderen Arrays verglichen.

Die Zuweisung von Werten eines Arrays an ein anderes Array ist nur unter einer der folgenden Bedingungen erlaubt:

- Die zwei verglichenen Dimensionen haben die gleiche Anzahl von Ausprägungen.
- Die zwei verglichenen Dimensionen haben beide eine unbestimmte Anzahl von Ausprägungen.

- Die Dimension, die einer anderen Dimension zugewiesen wird, besteht aus einer einzelnen Ausprägung.

Beispiel für Array-Zuweisungen:

Das folgende Programm zeigt, welche Zuweisungen zwischen Arrays möglich sind.

```
DEFINE DATA LOCAL
1 A1   (N1/1:8)
1 B1   (N1/1:8)
1 A2   (N1/1:8,1:8)
1 B2   (N1/1:8,1:8)
1 A3   (N1/1:8,1:8,1:8)
1 I    (I2)          INIT <4>
1 J    (I2)          INIT <8>
1 K    (I2)          CONST <8>
END-DEFINE
*
COMPUTE A1(1:3) = B1(6:8)                                  /* allowed
COMPUTE A1(1:I) = B1(1:I)                                  /* allowed
COMPUTE A1(*)   = B1(1:8)                                  /* allowed
COMPUTE A1(2:3) = B1(I:I+1)                                /* allowed
COMPUTE A1(1)   = B1(I)                                    /* allowed
COMPUTE A1(1:I) = B1(3)                                    /* allowed
COMPUTE A1(I:J) = B1(I+2)                                  /* allowed
COMPUTE A1(1:I) = B1(5:J)                                  /* allowed
COMPUTE A1(1:I) = B1(2)                                    /* allowed
COMPUTE A1(1:2) = B1(1:J)                                  /* NOT ALLOWED (NAT0631)
COMPUTE A1(*)   = B1(1:J)                                  /* NOT ALLOWED (NAT0631)
COMPUTE A1(*)   = B1(1:K)                                  /* allowed
COMPUTE A1(1:J) = B1(1:K)                                  /* NOT ALLOWED (NAT0631)
*
COMPUTE A1(*)       = B2(1,*)                              /* allowed
COMPUTE A1(1:3)     = B2(1,I:I+2)                          /* allowed
COMPUTE A1(1:3)     = B2(1:3,1)                            /* NOT ALLOWED (NAT0631)
*
COMPUTE A2(1,1:3)   = B1(6:8)                              /* allowed
COMPUTE A2(*,1:I)   = B1(5:J)                              /* allowed
COMPUTE A2(*,1)     = B1(*)                                /* NOT ALLOWED (NAT0631)
COMPUTE A2(1:I,1)   = B1(1:J)                              /* NOT ALLOWED (NAT0631)
COMPUTE A2(1:I,1:J) = B1(1:J)                              /* allowed
*
COMPUTE A2(1,I)     = B2(1,1)                              /* allowed
COMPUTE A2(1:I,1)   = B2(1:I,2)                            /* allowed
COMPUTE A2(1:2,1:8) = B2(I:I+1,*)                          /* allowed
*
COMPUTE A3(1,1,1:I)   = B1(1)                              /* allowed
COMPUTE A3(1,1,1:J)   = B1(*)                              /* NOT ALLOWED (NAT0631)
 COMPUTE A3(1,1,1:I)   = B1(1:I)                           /* allowed
COMPUTE A3(1,1:2,1:I) = B2(1,1:I)                          /* allowed
COMPUTE A3(1,1,1:I)   = B2(1:2,1:I)                        /* NOT ALLOWED (NAT0631)
END
```

Vergleiche mit Arrays

Grundsätzlich gilt Folgendes: Wenn mehrdimensionale Arrays miteinander verglichen werden, werden die einzelnen Dimensionen unabhängig voneinander behandelt; d.h. die 1. Dimension des einen Arrays wird mit der 1. Dimension des anderen Arrays verglichen, die 2. Dimension des einen Arrays wird mit der 2. Dimension des anderen Arrays verglichen, und die 3. Dimension des einen Arrays wird mit der 3. Dimension des anderen Arrays verglichen.

Der Vergleich zweier Array-Dimensionen ist nur unter einer der folgenden Bedingungen erlaubt:

- Die zwei verglichenen Dimensionen haben die gleiche Anzahl von Ausprägungen.
- Die zwei verglichenen Dimensionen haben beide eine unbestimmte Anzahl von Ausprägungen.
- Alle Dimensionen des einen Arrays bestehen jeweils aus einer einzelnen Ausprägung.

Beispiel für Array-Vergleiche:

Das folgende Programm zeigt, welche Vergleiche zwischen Arrays möglich sind.

```
DEFINE DATA LOCAL
1 A3  (N1/1:8,1:8,1:8)
1 A2  (N1/1:8,1:8)

1 A1  (N1/1:8)
1 I   (I2)   INIT <4>
1 J   (I2)   INIT <8>
1 K   (I2)   CONST <8>
END-DEFINE
*
IF A2(1,1)     = A1(1)              THEN IGNORE END-IF /* allowed
IF A2(1,1)     = A1(I)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(1)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(I)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(*)              THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(I -3:I+4)       THEN IGNORE END-IF /* allowed
IF A2(1,5:J)   = A1(1:I)            THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A1(1:I)            THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
IF A2(1,*)     = A1(1:K)            THEN IGNORE END-IF /* allowed
*
IF A2(1,1)     = A2(1,1)            THEN IGNORE END-IF /* allowed
IF A2(1,1)     = A2(1,I)            THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A2(1,1:8)          THEN IGNORE END-IF /* allowed
IF A2(1,*)     = A2(I,I -3:I+4)     THEN IGNORE END-IF /* allowed
IF A2(1,1:I)   = A2(1,I+1:J)        THEN IGNORE END-IF /* allowed
IF A2(1,1:I)   = A2(1,I:I+1)        THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
IF A2(*,1)     = A2(1,*)            THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
IF A2(1,1:I)   = A1(2,1:K)          THEN IGNORE END-IF /* NOT ALLOWED(NAT0629)
*
IF A3(1,1,*)   = A2(1,*)            THEN IGNORE END-IF /* allowed
IF A3(1,1,*)   = A2(1,I -3:I+4)     THEN IGNORE END-IF /* allowed
IF A3(1,*,I:J) = A2(*,1:I+1)        THEN IGNORE END-IF /* allowed
IF A3(1,*,I:J) = A2(*,I:J)          THEN IGNORE END-IF /* allowed
END
```

Wenn Sie zwei Array-Bereiche miteinander vergleichen, beachten Sie bitte, dass die folgenden zwei Ausdrücke zu unterschiedlichen Ergebnissen führen:

```
#ARRAY1(*) NOT EQUAL #ARRAY2(*)
NOT #ARRAY1(*) = #ARRAY2(*)
```

Beispiel:

- **Bedingung A:**

  ```
  IF #ARRAY1(1:2) NOT EQUAL #ARRAY2(1:2)
  ```

Dies entspricht:

```
IF (#ARRAY1(1) NOT EQUAL #ARRAY2(1)) AND (#ARRAY1(2) NOT EQUAL #ARRAY2(2))
```

Bedingung A ist also erfüllt, wenn die erste Ausprägung von #ARRAY1 ungleich der ersten Ausprägung von #ARRAY2 ist und die zweite Ausprägung von #ARRAY1 ungleich der zweiten Ausprägung von #ARRAY2 ist.

- **Bedingung B:**

```
IF NOT #ARRAY1(1:2) = #ARRAY2(1:2)
```

Dies entspricht:

```
IF NOT (#ARRAY1(1)= #ARRAY2(1) AND #ARRAY1(2) = #ARRAY2(2))
```

Dies wiederum entspricht:

```
IF (#ARRAY1(1) NOT EQUAL #ARRAY2(1)) OR (#ARRAY1(2) NOT EQUAL #ARRAY2(2))
```

Bedingung B ist also erfüllt, wenn *entweder* die erste Ausprägung von #ARRAY1 ungleich der ersten Ausprägung von #ARRAY2 ist *oder* die zweite Ausprägung von #ARRAY1 ungleich der zweiten Ausprägung von #ARRAY2 ist.

Arithmetische Operationen mit Arrays

Eine allgemeine Regel zu Arrays lautet, dass die Anzahl der Ausprägungen der entsprechenden Dimensionen gleich sein muss.

Das folgende Beispiel veranschaulicht diese Regel:

```
#c(2:3,2:4) := #a(3:4,1:3) + #b(3:5)
```

Mit anderen Worten:

Array	Dimension	Anzahl der Ausprägungen	Bereich
#c	2.	2	2:3
#c	1.	3	2:4
#a	2.	2	3:4
#a	1.	3	1:3
#b	1.	3	3:5

Die Operation wird Element für Element durchgeführt

Anmerkung:
Eine arithmetische Operation einer unterschiedlichen Anzahl von Dimensionen ist zulässig.

Für das obige Beispiel werden die folgenden Operationen ausgeführt:

```
#c(2,2) := #a(3,1) + #b(3)

#c(2,3) := #a(3,2) + #b(4)

#c(2,4) := #a(3,3) + #b(5)

#c(3,2) := #a(4,1) + #b(3)

#c(3,3) := #a(4,2) + #b(4)

#c(3,4) := #a(4,3) + #b(5)
```

In arithmetischen Operationen (in `COMPUTE`-, `ADD`- oder `MULTIPLY`-Statements) können Array-Bereiche auf folgende Arten verwendet werden. In den Beispielen 1 - 4 muss die Anzahl der Ausprägungen der entsprechenden Dimensionen gleich sein.

1. *`range + range = range`*.

   Die Addition wird Element für Element ausgeführt.

2. *`range * range = range`*.

   Die Multiplikation wird Element für Element ausgeführt.

3. *`scalar + range = range`*.

   Der Skalarwert wird zu jedem Element des Bereichs addiert.

4. *`range * scalar = range`*.

   Jedes Element des Bereichs wird mit dem Skalarwert multipliziert.

5. *`range + scalar = scalar`*.

   Jedes Element des Bereichs wird zum Skalarwert addiert und das Ergebnis im Skalar ausgegeben.

6. *`scalar * range = scalar2`*.

   Der Skalarwert wird mit jedem Element des Arrays multipliziert und das Ergebnis in *`scalar2`* ausgegeben.

Weil, wie aus den obigen Beispielen hervorgeht, bei arithmetischen Operationen Zwischenergebnisse erzeugt werden, wird das Ergebnis von sich überlappenden Indexbereichen Element für Element in einem Zwischenergebnis-Array berechnet, und schließlich wird das Zwischenergebnis-Array dem Ergebnisfeld zugewiesen.

Beispiel:

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:5) INIT <10,20,30,40,50>
END-DEFINE

#ARRAY(3:5) := #ARRAY(2:4) + 1

/* A temporary array for the
/* intermediate result values is
```

```
/* generated implicitly: #temp(1:3).
/* The following operations are
/* performed internally:
/* #temp(1) := #ARRAY(2) + 1
/* #temp(2) := #ARRAY(3) + 1
/* #temp(3) := #ARRAY(4) + 1
/* #ARRAY(3:5) := #temp(1:3)
/*
/* #ARRAY contains  10,20,21,31,41
```