7. Arithmetische Operatoren und Befehle zur Programmorganisation

7.1 Arithmetische und Vergleichs-Operatoren

Die in der Automatisierungstechnik erforderliche Datenverarbeitung erfordert Operatoren für Arithmetik und Vergleiche. IEC 61131-3 hält den Umfang der Operatoren gering. Als Folge sind für die Bearbeitung von BYTE, WORD oder DWORD bzw. INT und REAL keine speziellen, sich unterscheidenden Befehle vorgesehen.

Nachfolgender Auszug der CoDeSys AWL Referenzliste zeigt die wenigen Operatoren für Arithmetik und Vergleiche:

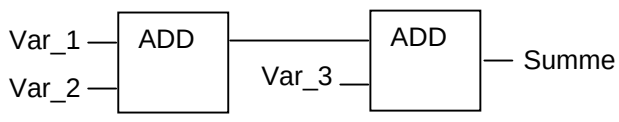
ADD	Addition von Akkumulator und Operand, das Ergebnis wird in den Akkumulator abgelegt.
SUB	Subtraktion von Akkumulator und Operand, das Ergebnis wird in den Akkumulator abgelegt.
MUL	Multiplikation von Akkumulator und Operand, das Ergebnis wird in den Akkumulator abgelegt.
DIV	Division von Akkumulator und Operand, das Ergebnis wird in den Akkumulator abgelegt.
GT	Überprüfen, ob der Akkumulator größer als (**g**reater **t**han)der Operand ist, das Ergebnis (BOOL) wird in den Akkumulator abgelegt.
GE	Überprüfen, ob der Akkumulator größer als oder gleich dem Operanden ist (**g**reater than or **e**qual), das Ergebnis (BOOL) wird in den Akkumulator abgelegt.
EQ	Überprüfen, ob der Akkumulator gleich dem Operanden ist (**eq**ual), das Ergebnis (BOOL) wird in den Akkumulator abgelegt.
NE	Überprüfen, ob der Akkumulator ungleich dem Operanden ist (**n**ot **e**qual), das Ergebnis (BOOL) wird in den Akkumulator abgelegt.
LE	Überprüfen, ob der Akkumulator kleiner als oder gleich dem Operanden ist (**l**ess than or **e**qual to), das Ergebnis (BOOL) wird in den Akkumulator abgelegt.
LT	Überprüfen, ob der Akkumulator kleiner als der Operand ist (**l**ess **t**han), das Ergebnis (BOOL) wird in den Akkumulator abgelegt.

Die **vier Grundrechenarten** Addition, Subtraktion, Multiplikation und Division sind für die Datentypen BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL und LREAL gleichermaßen zu nutzen.

Addition und Subtraktion kann auch mit Daten vom Typ **TIME** erfolgen.

Die Ergebnisse der Rechenoperationen werden im gleichen Register abgelegt, in welches auch die Operandengeladen werden.
Im System Simatic S7 wird hierbei mit den Akkumulatoren 1 und 2 gearbeitet, das Ergebnis steht stets im Akku 1.

Beispiel Addition:

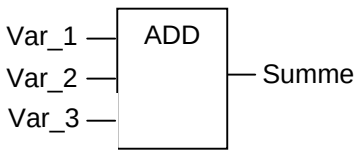



```
LD   Var_1
ADD  Var_2
ADD  Var_3
ST   Summe

oder

LD   Var_1
ADD  Var_2, Var_3
ST   Summe
```

identisch mit

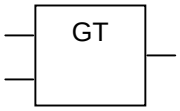


Es gibt sechs **Vergleichsoperationen**: Ihre Ergebnisse sind vom Typ BOOL. Das Ergebnis ist TRUE, wenn der Wert des ersten Operanden sich zum zweiten Operanden im Sinne der Operation verhält.

Die Operanden können vom Datentyp BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME und STRING sein.

Größer als (>)

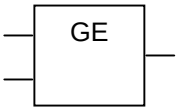
(**G**reater **T**han)




```
LD  Var_1
GT  Var_2
ST  Ergebnis
```

Größer oder gleich als (≥)

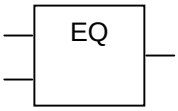
(**G**reater Than or **E**qual)




```
LD  Var_1
GE  Var_2
ST  Ergebnis
```

Gleich (=)

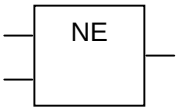
(EQual)




```
LD  Var_1
EQ  Var_2
ST  Ergebnis
```

Ungleich (<>)

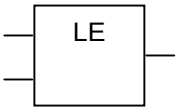
Not **E**qual)




```
LD  Var_1
NE  Var_2
ST  Ergebnis
```

Kleiner oder gleich als (≤)

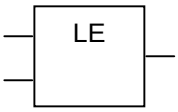
(**L**esser Than or **E**qual)




```
LD  Var_1
LE  Var_2
ST  Ergebnis
```

Kleiner als (≥)

(**L**esses **T**han)

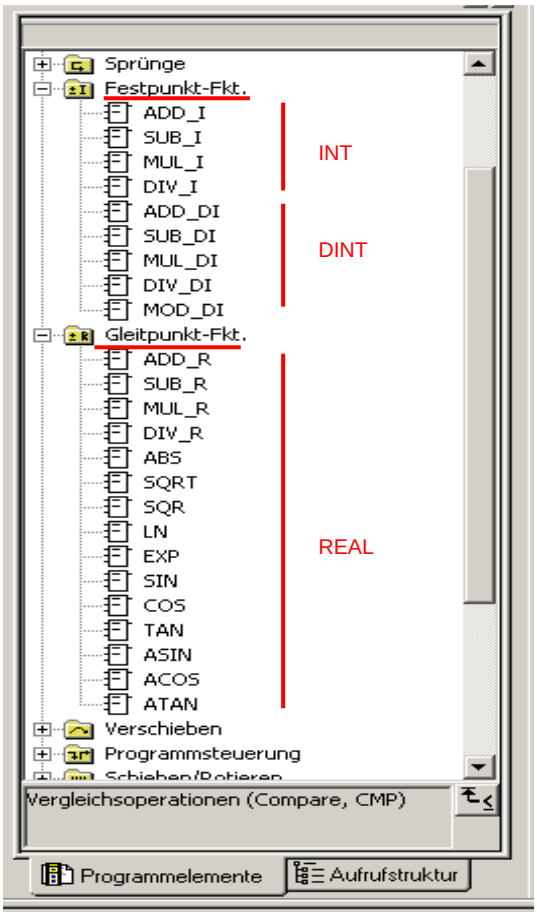



```
LD  Var_1
LE  Var_2
ST  Ergebnis
```

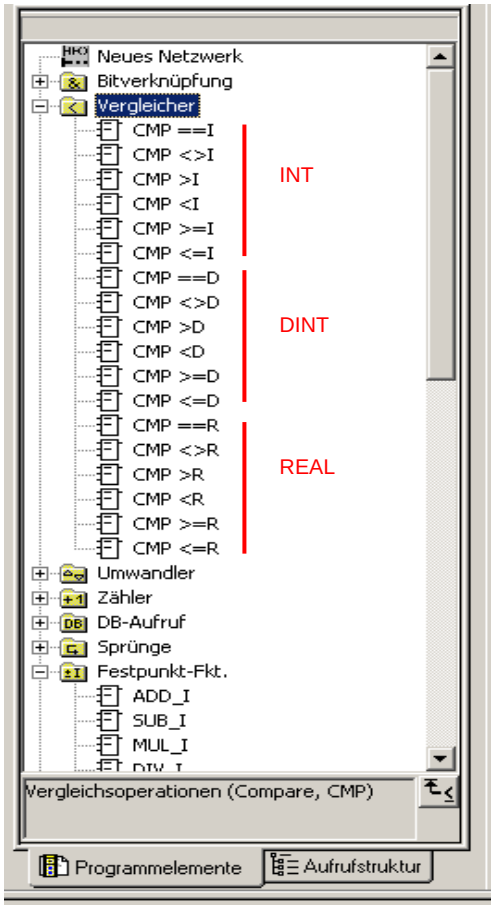
- **Zum Vergleich: Die Operationen in Step7**

Step7 verwendet bei den arithmetischen Operationen und auch bei den Vergleichsoperationen jeweils eigenständige Befehle für unterschiedliche Datenformate. Die Vergleichsoperationen sind auf Datentypen INT, DINT und REAL beschränkt (**Bild 7-1**).

Bei den Operationen für Gleitpunktzahlen gibt es auch Trigonometrische Operationen sowie Quadratwurzel, Logarithmus und Exponentialfunktion.



Step7 verwendet bei den arithmetischen Operationen getrennte Befehle für Festpunktzahlen (Datentyp INT und DINT) und Gleitpunkzahlen (Datentyp REAL).



Step7 lässt die gleichen sechs Vergleichs-operationen wie IEC 61131-3 zu,
verwendet aber für die drei zugelassenen Datentypen INT, DINT und

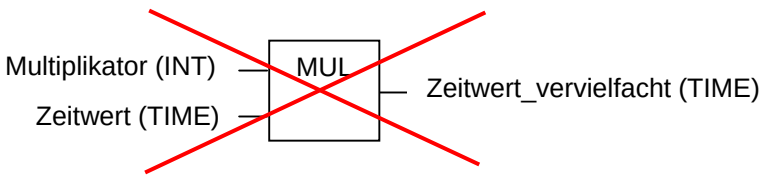
Bild 7-1: Step7 - Operationen für Arithmetik und Vergleich

7.2 Datenkonvertierungen

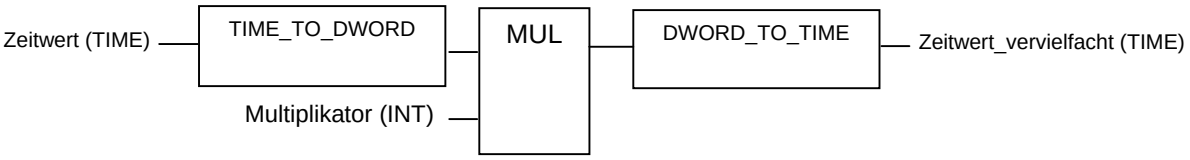
Quelle: Online-Hilfe des Systems CoDeSys:
(Beim Einsatz solcher sehr speziellen Operationen sollte stets diese Hilfe benutzt werden.)

Die Operationen für Arithmetik und Vergleich gelten für sehr viele Datentypen. Kommt es dabei dennoch zu Typkonflikten, so sind geeignete Konvertierungen durchzuführen. Dies schafft dem Programmierer viele Freiheiten!

Beispiel: Ein Zeitwert soll variabel vervielfacht werden. Die direkte Multiplikation einer Zeit (Datentyp TIME) mit einem Multiplikator vom Typ INT würde zu einem Typkonflikt führen und ist deshalb nicht erlaubt.



Deshalb muss der Datentyp TIME zunächst in einen Typ DWORD konvertiert werden, mit dem die Multiplikation möglich ist. Nach der Multiplikation kann dann umgekehrt das Produkt zurück in den Typ TIME gewandelt werden. Die Basis aller Zeit-Datentypen ist die Millisekunde und das 32-Bit-Doppelwort.



- **Grundsätzlich kann von jedem elementaren Typ zu jeden anderen elementaren Typ konvertiert werden.**

Es ist aber allgemein nicht sinnvoll, von einem "größeren" Typ auf einen "kleineren" Typ zu konvertieren (beispielsweise von INT nach BYTE oder von DINT nach WORD). Wenn man das tun will, muss man die Inhalte kennen und spezielle Typkonvertierungen anwenden.

Manche Konvertierungen sind zunächst nicht einzusehen. Mit einer Konvertierung TIME_TO_BOOL könnte man aber zum Beispiel abfragen, ob in eine Variable vom Typ TIME überhaupt ein konkreter Wert eingetragen wurde. In diesem Falle hätte die Boolesche Ausgangsvariable den Wert TRUE, sonst FALSE.

Nachfolgend wird das Wirken der wichtigsten Konvertierungen an Beispielen aufgezeigt:

Konvertierung	**Beispiel in AWL**	**Inhalt der Variablen, in welche das Ergebnis geschrieben wird**
Gruppe Konvertierung TIME_TO		
TIME_TO_DWORD	siehe o.a. Beispiel LD T#1min TIME_TO_DWORD ST Anzahl_Millisekunden	 60 000
TIME_TO_STRING	LD T#12ms TIME_TO_STRING ST Text_1	 ' T#12ms'
TOD_TO_SINT	LD TOD#00:00:00.30 TOD_TO_SINT ST Sekundenangabe	 30
Gruppe Konvertierung DATE_TO bzw. DATE_AND_TIME _ TO		

Es wird jeweils der interne Wert des Datums in Sekunden konvertiert. Wenn der Zieldatentyp ein kleinerer Wert ist, können Informationen verloren gehen. Wie auch bei anderen Konvertierungen ..._TO_STRING wird bei dieser Konvertierung der Eingangswert als Stringangabe '.....' geschrieben.

Beispiele:

DATE_TO_BOOL	LD D#1970-01-01 DATE_TO_BOOL ST Var_Bool1	 FALSE
	LD B#1985-03-12 DATE_TO_BOOL ST ar_Bool2	 TRUE

DATE_TO_INT	LD D#1970-01-15 DATE_TO_INT ST Var_INT	 29 952 (Dies ist die Anzahl Sekunden seit 01.01.1970)

Gruppe Konvertierung BOOL_TO

BOOL_TO_INT	LD TRUE BOOL_TO_INT ST Zahl	 1
BOOL_TO_TIME	LD TRUE BOOL_TO_TIMe ST Zeitwert	 1 ms
BOOL_TO_TOD	LD FALSE BOOL_TO_TOD ST Tageszeit	 TOD#00:00:00:.000
BOOL_TO_DATE	LD TRUE BOOL_TO_TOD ST Datum	 DT#1970-01-01-00:00:01
BOOL_TO_STRING	LD TRUE BOOL_TO_TOD ST Text	 ‘TRUE’

Gruppe Konvertierung TO_BOOL
Das Ergebnis ist TRUE, wenn der Operand ungleich 0 ist.
Das Ergebnis ist FALSE, wenn der Operand gleich 0 ist.
Beim Typ STRING ist das Ergebnis TRUE, wenn der Operand 'TRUE' ist, ansonsten ist das Ergebnis FALSE.

INT_TO_BOOL	LD 314 INT_TO_BOOL ST Var_BOOL	 TRUE
	LD 0 INT_TO_BOOL ST Var_BOOL	 FALSE
TIME_TO_ BOOL	LD T#2s TIME_TO_ BOOL ST Var_BOOL	 TRUE
STRING_TO_BOOL	LD ‘TRUE’ alternativ ‘FALSE’ STRING_TO_BOOL ST Var_BOOL	 FALSE alternativ TRUE
BYTE_TO_BOOL	LD 14 BYTE_TO_BOOL ST Var_BOOL	 TRUE

Gruppe Konvertierungen zwischen ganzzahligen Zahlentypen
Konvertierung von einem ganzzahligen Zahlentyp zu einem anderen Zahlentyp:
Bei der Typkonvertierung von größeren auf kleinere Typen können Informationen verloren gehen. Wenn die zu konvertierende Zahl die Bereichsgrenze überschreitet, dann werden die ersten Bytes der Zahl nicht berücksichtigt.

Beispiel:

INT_TO_SINT	LD 4223		4223 entspricht 16#107F
	INT_TO_SINT		
	ST Var_SINT	127	127 entspricht 16#7F

Beispiele:

REAL_TO_INT	LD 3.14	
	REAL_TO_INT	
	ST Var_INT	3 (entspricht Rundung
	LD –10.4	
	REAL_TO_INT	
	ST Var_INT	-10
REAL_TO_ STRING	LD 3.14	
	REAL_TO_STRING	
	ST Var_STRING	‘3.14’
REAL_TO_BOOL	LD 3.14	
	REAL_TO_BOOL	
	ST Var_BOOL	TRUE

Gruppe Konvertierung STRING_TO

STRING_TO_BOOL	LD ’3.14’	
	STRING_TO_BOOL	
	ST Var_BOOL	FALSE
	LD ’TRUE’	
	STRING_TO_BOOL	
	ST Var_BOOL	TRUE

Rundung von Gleitpunktzahlen (TRUNC)

Mit dem Befehl TRUNC wird der Betrag von REAL-Werten in INTEGER konvertiert. Bei der Konvertierung von größeren zu kleineren Typen können Teile der Information verloren gehen!

Konvertierungen	**Beispiel in AWL**	**Inhalt der Variablen, in welche Ergebnis geschrieben wird**
Beispiel:	LD 3.14	
	TRUNC	
	ST VAR_INT	3

7.3 Operationen zur Programmorganisation

Bei der Lösung von Problemen der Automatisierungstechnik kommt es häufig vor, dass nicht alle Anweisungen eines Programms zu jeder Zeit zu bearbeiten sind, sondern nur unter bestimmten Bedingungen. Dann kommen **zeit- oder ereignisgesteuerte Task** zum Einsatz.

In einem zyklisch bearbeiteten Programm müssen in diesem Fall dagegen entweder
- POE nicht aufgerufen oder
- bestimmte Programmteile einer POE übersprungen werden.

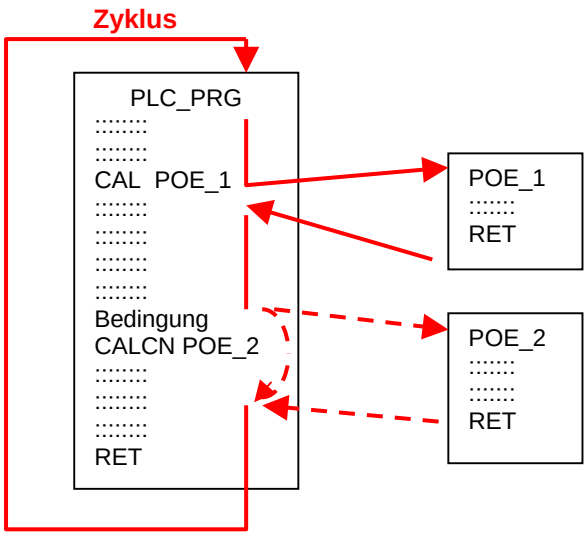
Für derartige Gestaltung eines Programms stehen die Operationen CAL, JMP und RET mit ihren Modifikatoren C und N zur Verfügung.

Modifikator	bei Befehlen	Wirkung
C	CAL, JMP, RET	Die Anweisung wird nur ausgeführt, wenn das Ergebnis der vorhergehenden Anweisung TRUE ist
N	CALC, JMPC, RETC	Die Anweisung wird nur ausgeführt, wenn das Ergebnis der vorhergehenden Anweisung FALSE ist

im einzelnen:

JMP	Unbedingter Sprung zur Sprungmarke
JMPC	Bedingter Sprung zur Sprungmarke nach einer Überprüfung, ob der Akkumulator TRUE ist (**C**onditional jump)
JMPCN	Bedingter Sprung zur Sprungmarke nach einer Überprüfung, ob der Akkumulator FALSE ist.
CAL	Aufruf eines PROGRAM (Bausteintyp "Programm") oder FUNCTION_BLOCK (Bausteintyp Funktionsblock)
CALC	Bedingter Aufruf eines PROGRAM (Bausteintyp "Programm") oder FUNCTION_BLOCK (Bausteintyp Funktionsblock), wenn der Akkumulator TRUE ist.
CALCN	Bedingter Aufruf eines PROGRAM (Bausteintyp "Programm") oder FUNCTION_BLOCK (Bausteintyp Funktionsblock), wenn der Akkumulator FALSE ist.
RET	Vorzeitige Beendigung des Programmbausteins und Rücksprung zum aufrufenden Programmbaustein.
RETC	Bedingte vorzeitige Beendigung des Programmbausteins und Rücksprung zum aufrufenden Programmbaustein, wenn der Akkumulator TRUE ist.
RETCN	Bedingte vorzeitige Beendigung des Programmbausteins und Rücksprung zum aufrufenden Programmbaustein, wenn der Akkumulator FALSE ist.

Beispiel: Nichtbearbeitung einzelner POE

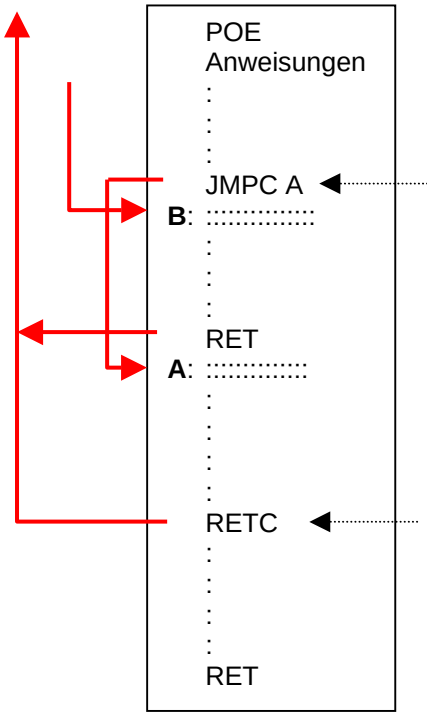


POE_1 wird immer bearbeitet.

POE_2 wird nur dann bearbeitet, wenn das Ergebnis der Anweisung vor CALCN POE_2 den Wert FALSE liefert.

Achtung!
Stets muss ein ablauffähiger Zyklus erhalten werden!

Beispiel: Anwendungen von Sprüngen und Marken, Nichtbearbeitung einzelner Anweisungen



Der Sprung zur Marke A wird ausgeführt, wenn das Ergebnis des vorhergehenden Ausdrucks TRUE ist.

JMP	Sprung wird immer ausgeführt
JMPCN	Sprung wird ausgeführt, wenn der Wert des vorhergehenden Ausdrucks FALSE ist

Der Absprung aus der POE wird ausgeführt, wenn das Ergebnis des vorhergehenden Ausdrucks TRUE ist.

RET	Absprung aus der POE wird immer ausgeführt
RETCN	Absprung aus der POE wird ausgeführt, wenn der Wert des vorhergehenden Ausdrucks FALSE ist

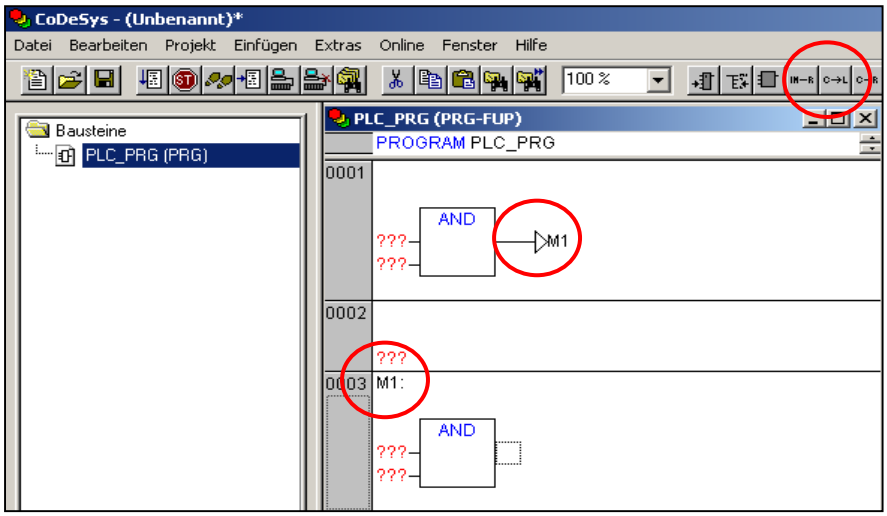


Bild 7-2: Darstellung von Sprüngen und Label in FUP

Beim Verwenden von Sprüngen und Aufrufen von POE ist stets zu beachten: Werden durch Sprünge Programmanweisungen nicht mehr bearbeitet und wurden in ihnen zuvor Variablen und insbesondere auf Ausgänge gelegte Variablen auf den Wert TRUE geschaltet, so bleiben diese Werte erhalten! Dies kann zu ernsthaften Problemen führen!!!