

Konvertierung von STEP7-SCL nach STEP7-AWL

Dr.-Ing. Rolf Geisler

Die Programmierung von Steuerungen in strukturiertem Text nach IEC1131 hat dank der kompakten Formulierung des Quellcodes gegenüber der althergebrachten AWL deutliche Vorteile. So dürfte ST in der Programmierergemeinde inzwischen wohl eine der favorisierten Sprachen sein. Ungeachtet dessen bestehen viele Maschinenbetreiber nach wie vor auf Software in AWL, im Extremfall sogar KOP oder FUP. Als Argument wird häufig angeführt, dass das Wartungspersonal mit dem Strukturierten Text nicht vertraut sei und daher im Falle einer Störung keine eigenständige Fehlersuche durchführen könne. Ein dankbares Feld für die Weiterbildung ...

Sofern die Programmierung mit CoDeSys erfolgt, ist die Konvertierung von Bausteinen nach AWL kein Problem. Anders bei SIMATIC S7. Mit den Bordmitteln von STEP7 ist es bekanntermaßen nicht möglich, in SCL geschriebene Bausteine nach AWL zu konvertieren und sie mit dem STEP7-AWL-Editor weiter zu bearbeiten.

Der vorliegende Beitrag beschreibt eine Möglichkeit, SCL doch nach AWL zu konvertieren, wobei die konvertierten Bausteine tatsächlich mit dem STEP7-AWL-Editor geöffnet und weiterbearbeitet werden können. Das beschriebene Verfahren nutzt das frei verfügbare SPS-Entwicklungssystem CoDeSys der Firma 3S – Smart Software Solutions GmbH. Es ist allerdings eine umfangreiche manuelle Nachbearbeitung des konvertierten Codes erforderlich.

1 Übersicht

Die Konvertierung von STEP7-SCL nach STEP7-AWL erfolgt in den folgenden Schritten:

1. Import des STEP7-SCL-Projektes in ein CoDeSys-Projekt
2. Umwandeln der Bausteine in Anweisungsliste. Dabei entsteht die AWL-Struktur (Befehlssequenzen, Sprünge und Sprungmarken) in IEC1131-Notation
3. Re-Export der konvertierten Bausteine nach STEP7, jetzt in AWL
4. Manuelle Nachbearbeitung und Übersetzen der STEP7-AWL-Bausteine

2 Import eines STEP7-SCL-Projektes nach CoDeSys

2.1 Aufbau eines STEP7-SCL-Bausteins

Ein STEP7-SCL-Baustein hat immer den folgenden Aufbau:

Bausteinbeginn	FUNCTION xyz: Funktionsergebnistyp oder FUNCTION_BLOCK xyz oder ORGANIZATION_BLOCK xyz
Deklarationsteil	VAR_INPUT ... END_VAR VAR_OUTPUT ... END_VAR VAR_IN_OUT

	... END_VAR VAR ... END_VAR VAR_TEMP ... END_VAR
Einleitung Codeteil	BEGIN
Codeteil	...
Bausteinende	END_FUNCTION oder END_FUNCTION_BLOCK oder END_ORGANIZATION_BLOCK

Der Deklarationsteil muss nicht alle oben aufgeführten Segmente enthalten. Ihre Reihenfolge ist unter STEP7 fest vorgegeben, spielt jedoch unter CoDeSys keine Rolle.

2.2 Konvertierungsfreundliche SCL-Quellen

CoDeSys verschiebt ungünstig platzierte Kommentare bei der Umwandlung nach AWL an eine andere Stelle.

Zeilenkommentare

Zeilenkommentare, die in SCL im Anschluß an Programmtext auf die gleiche Zeile geschrieben wurden, beim Konvertieren nach AWL vor den Anweisungsblock, in den diese SCL-Anweisung umgewandelt wird, oder auf seine letzte Zeile geschrieben. Dies kann die Verständlichkeit des konvertierten Programmcodes beeinträchtigen.

<u>SCL-Code:</u> <pre>(* Kommentar 1 *) IF NOT ixE1 (* Zeilenkommentar 1 *) THEN Var2 := C1; END_IF; IF BoolVar3 (* Zeilenkommentar 2 *) THEN Var2 := C2; END_IF;</pre>	<u>AW-Code:</u> <pre>(* Kommentar 1 *) LD ixE1 JMPC else12_0 (* Zeilenkommentar 1 *) LD C1 ST Var2 else12_0: end12_0: LDN BoolVar3 JMPC else13_0 (* Zeilenkommentar 2 *) LD C2 ST Next else13_0: end13_0:</pre>
<u>SCL-Code:</u> <pre>tmpInfeed := Data[ix] (* 0 .. 10V *) / UmrFaktor (* ==> Kraft *) * NominalLoad / AnzahlKanaele; (* kN *)</pre>	<u>AWL-Code:</u> <pre>(* Eingangsspannung 0 .. 10V ==> Kraft *) LD Data[ix] DIV UmrFaktor MUL NominalLoad DIV AnzahlKanaele ST tmpInfeed (* Umrechnung in kN</pre>

Da Zeilenkommentare, wie gesehen, bei der Umwandlung in AWL ihre Position wechseln, und zudem STEP7-AWL als Zeilenkommentar im laufenden Quelltext nur maximal 79 Zeichen pro Kommentar zulässt, sollte man Zeilenkommentare grundsätzlich vermeiden.

Besser ist es in jedem Falle, einen ausführlichen, gegebenenfalls mehrzeiligen Kommentar vor eine SCL-Befehlssequenz einzufügen. Dieser bleibt auch nach der Konversion in AWL als „Blocküberschrift“ erhalten.

Kommentare innerhalb von Bausteinaufrufen

Kommentare innerhalb von Bausteinaufrufen, sowohl Kommentarzeilen innerhalb der VAR_INPUT-Liste als auch Kommentare zu einzelnen VAR_INPUTs, werden vor den Bausteinaufruf geschoben.

SCL-Code:	AW-Code:
<pre> (* Kommentar 1 *) SomeFB (ixE1 := ..., ixE2 := ..., (* Kommentar 2 *) ixE3 := ..., ixE4 := ..., (* Kommentar 3 *) ixE5 := ..., ixE6 := ..., (* Kommentar 4 *) ixE7 := ..., ixE8 := ..., (* Kommentar 5 *) ixE9 := ...); </pre>	<pre> (* Kommentar 1 Kommentar 2 Kommentar 3 Kommentar 4 Kommentar 5 *) CAL SomeFB(ixE1 := ..., ixE2 := ..., ixE3 := ..., ixE4 := ..., ixE5 := ..., ixE6 := ..., ixE7 := ..., ixE8 := ..., ixE9 := ...); </pre>

Konvertierungsfreundliche SCL-Quellen vermeiden Kommentare innerhalb von Bausteinaufrufen. Wichtige Informationen werden vor den Bausteinaufruf auf separate Kommentarzeilen geschrieben.

Besser ist es, Zeilenkommentare zu vermeiden und anstatt dessen Kommentierungen auf separaten Zeilen zu schreiben. Diese Kommentare dürfen auch mehrzeilig sein. Sie bleiben bei der Konvertierung an ihrer Ursprungsposition stehen. Dies erleichtert auch die Navigation im Programm.

2.3 Import nach CoDeSys

Der Import eines Projektes von STEP7 nach CoDeSys erfolgt am zweckmäßigsten über die Windows-Zwischenablage.

Dazu legt man zunächst ein neues CoDeSys-Projekt an. Für jeden Baustein des Quellprojektes legt man danach im Projektbrowser ein neues Objekt an

- Sprache des Bausteins: ST
- Bausteintyp, Bausteinname und gegebenenfalls Funktionsergebnis identisch zur STEP7-Vorlage

öffnet dieses und holt den STEP7-Baustein in das CoDeSys-Projekt herein.

Der Quellbaustein kann entweder direkt vom STEP7-Editor oder auch von einer exportierten STEP7-SCL-Quelle (diese kann mit einem normalen Texteditor geöffnet werden) abgeholt werden.

Beim Import ist zu beachten

1. dass der Bausteinbeginn, Einleitung Codeteil und Bausteinende von CoDeSys verwaltet werden und daher nicht im Quelltext stehen. Sie sind nach dem Import auszukommentieren.

Man könnte die betreffenden Zeilen auch löschen. Wenn das Projekt aber nach STEP7 re-exportiert werden soll, werden sie dort wieder benötigt. Sie sind durch Ent-kommentieren leicht wiederherstellbar.

2. dass STEP7-OB's im CoDeSys zu PROGRAMs werden
3. dass das Quellprojekt auf alle Variablen, Bausteine und Datentypen nur unter ihrem symbolischen Namen zugreifen darf

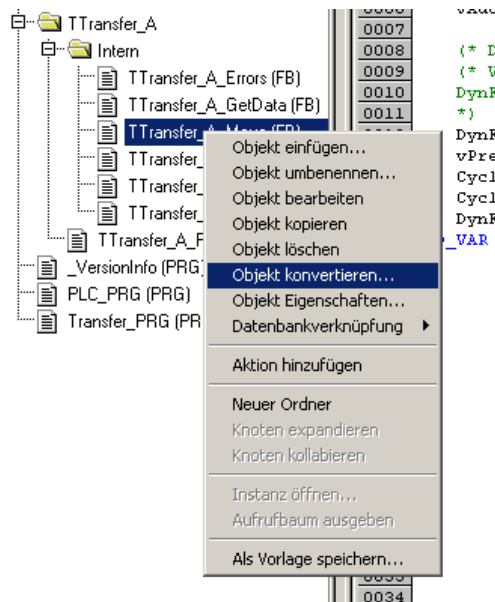
4. dass die Variablenliste des Bausteins unter CoDeSys im Deklarationseditor (dem oberen Teil des Arbeitsblattes) abgelegt wird, während der Programmcode in den Bausteinrumpf (den unteren Teil des Arbeitsblattes) gehört
5. dass CoDeSys keine *VAR_TEMP* kennt. Variablen aus der temporären Variablen-tabelle werden zu normalen *VAR*'s
6. dass Kommentare in CoDeSys immer mit *(** eingeleitet und mit **)* beendet werden. Mehrfach geschachtelte Kommentare sind möglich. Das unter STEP7 zulässige *//* ist in CoDeSys verboten
7. dass in STEP7 deklarierte *UDT*'s unter ihrem symbolischen Namen ebenfalls nach CoDeSys übertragen werden müssen. Für sie werden im CoDeSys-Projektbaum auf dem Tabellenreiter *Datentypen* Arbeitsblätter eingerichtet
8. dass alle vom Quellprojekt verwendeten globalen Variablen ebenfalls nach CoDeSys übertragen werden müssen. Sie werden im CoDeSys-Projektbaum auf dem Tabellenreiter *Ressourcen* im Arbeitsblatt *Globale Variablen* eingetragen. Weitere Arbeitsblätter mit globalen Variablen können hier beliebig hinzugefügt werden

Wenn die Absicht besteht, dieses Projekt im CoDeSys-Simulator zu testen, ist es erforderlich, den OB1 sowie die Alarm-OB's des Quellprojektes in die CoDeSys-Taskverwaltung einzubinden. Für das reine Übersetzen der Bausteine ist die Einbindung in die CoDeSys-Taskverwaltung nicht notwendig.

Abschließend übersetzt man das Projekt unter Nutzung des betreffenden Menüpunktes aus dem *Projekt*-Menü. Es dürfen keine Übersetzungsfehler, wie fehlende Variablendeklarationen, Datentypen usw. mehr auftreten.

3 Umwandeln der Bausteine in Anweisungsliste

CoDeSys stellt für die Konvertierung nach AWL den Eintrag *Objekt konvertieren ...* im Kontextmenü eines markierten Bausteines zur Verfügung.



Im nachfolgenden Dialog wählt man die Sprache AWL.



Man kann zwar einen anderen Bausteinnamen angeben, in welchen das Ergebnis der Umwandlung abgelegt wird, verliert dabei aber die Übersetzbarkeit des Umwandlergebnisses. Daher sollte man den Bausteinnamen beibehalten. Der neue AWL-Baustein überschreibt dann den alten ST-Quellcode.

Die automatisch generierte AWL bildet den Programmablauf der SCL-Vorlage vollständig auf eine Anweisungsliste ab. Wo erforderlich, werden weitere Einträge in die Variablenliste des Bausteins erzeugt. Insbesondere definiert CoDeSys alle Sprünge mit Sprungbefehl und Sprungziel.

Man sollte jedoch schon jetzt alle Sprungmarken auf das STEP7-AWL-Format (max. 4 Zeichen) bringen. Der CoDeSys-Editor ist für diese Aufgabe besser geeignet als der STEP7-Editor.

Vor dem Konvertieren des nächsten Bausteins muss das Projekt erneut übersetzt werden.

4 Re-Export der konvertierten Bausteine nach STEP7-AWL

Um die konvertierten Bausteine mit STEP7-AWL öffnen zu können, müssen sie in sog. AWL-Quellen umgewandelt werden.

AWL-Quellen sind reine ASCII-Dateien mit dem Inhalt eines oder mehrerer Bausteine. Zum Export des CoDeSys-AWL-Projektes öffnet man also einen beliebigen Texteditor und kopiert den Bausteininhalt über die Windows-Zwischenablage in ein leeres Dokument. Danach sichert man dieses Dokument unter einem beliebigen Namen und der Dateierweiterung AWL.

Hinweise:

- *Es empfiehlt sich, nur jeweils einen Baustein in eine Datei zu schreiben.*
- *Als Dateiname bietet sich der symbolische Bausteinname an. Dies erleichtert die Navigation beim Import in STEP7.*

Beim Re-Export ist zu beachten

1. dass der Bausteinbeginn, Einleitung Codeteil und das Bausteinende wieder ent-kommentiert werden
2. dass CoDeSys-PROGRAM's in STEP7 zu OB's werden

Nachdem alle Bausteine in einzelne Dateien exportiert wurden, stehen sie für den Import in ein STEP7-Projekt bereit. Dieser Import erfolgt in den Ordner Quellen. Im Kontextmenü findet sich dazu der Eintrag *Quelle importieren*.

5 Nachbearbeitung

Die von CoDeSys produzierte AWL entspricht vollständig der IEC1131. Leider trifft das für STEP7-AWL nicht zu. Die gravierendsten Abweichungen sind

- unterschiedliche Befehle, bei STEP7-AWL zusätzlich verkompliziert durch Unterscheidungen hinsichtlich des Datentyps der an einer Operation beteiligten Variablen.
- Restriktionen hinsichtlich der Wahl von Sprungmarkenbezeichnern
- Unterschiedliches Handling von Kommentaren im Quelltext
- Unterschiedliches Handling der Formalparameterliste an Bausteinaufrufen

Daher ist eine Nachbearbeitung der konvertierten Bausteine unerlässlich. Diese erfolgt am günstigsten in STEP7, da man dort die Quellen immer wieder bis zur Fehlerfreiheit übersetzen und später im Simulator testen kann.

Die Nachbearbeitung der AWL-Quellen sollte etwa in folgender Reihenfolge erfolgen:

1. Korrektur der Sprungmarkenbezeichner
2. Überarbeitung der Kommentare
3. Herstellen der Netzwerkstruktur
4. Korrektur der Befehle
5. Korrektur der Aufrufe unterlagerter FB's und Funktionen

5.1 Korrektur der Sprungmarkenbezeichner

CoDeSys generiert bei der Umwandlung in AWL Sprungmarken nach dem Muster *SchlüsselwortNr_0*, z.B. *else5_0*. STEP7-AWL hingegen erlaubt nur maximal 4 Zeichen pro Sprungmarkenbezeichner (das erste Zeichen muss ein Buchstabe sein). Daher müssen alle CoDeSys-Sprungmarken durch Zeichenketten, die die STEP7-Einschränkungen befolgen, ersetzt werden.

Der STEP7-AWL-Editor ist für diese Aufgabe nicht besonders gut geeignet, da sein Suchen-Ersetzen-Dialog keine Memory-Funktion für Such- und Ersatzbegriffe besitzt. Man muss bei der Bearbeitung mehrerer Bausteine die immer wiederkehrenden CoDeSys-Sprungmarkenbezeichner jedes Mal neu eintippen.

Der Austausch der Sprungmarken geht mit dem CoDeSys-Editor deutlich schneller von der Hand. Daher sollte man die Sprungmarken unmittelbar nach der Generierung der AWL noch vor dem Re-Export nach STEP7 korrigieren.

5.2 Überarbeitung der Kommentare

In CoDeSys werden Kommentare grundsätzlich von den Zeichenfolgen (* und *) eingeschlossen. Zeilenübergreifende Kommentare sind möglich. STEP7-AWL dagegen verwendet grundsätzlich die Steuerzeichen //, um einen Kommentar einzuleiten. Zeilenübergreifende Kommentare sind nicht möglich; Kommentarfortsetzungen müssen auf jeder Zeile mit einem neuen Kommentarsteuerzeichen eingeleitet werden.

Die CoDeSys-Kommentarkennungen sind daher durch die STEP7-AWL-Kommentarkennungen zu ersetzen. Die Zeichenkette // ist dabei, sofern es sich um zeilenübergreifende Kommentare handelt, auf jeder Zeile einzufügen.

Zeilenkommentare sind von CoDeSys verschoben worden. Für eine bessere Lesbarkeit der AWL-Übersetzung sollten sie an die Ursprungsstelle zurückgeschoben werden.

Besser ist es jedoch, Zeilenkommentare zu vermeiden und anstatt dessen Kommentare auf separate Zeilen zu schreiben.

→ [konvertierungsfreundliche SCL-Quellen](#)

5.3 Herstellen der Netzwerkstruktur

CoDeSys erzeugt beim Konvertieren nach AWL einen fortlaufenden Baustein, ohne die von Siemens bekannte Gliederung in Netzwerke. Daher muss in den umgewandelten Quelltext nach dem Schlüsselwort *BEGIN* mindestens ein Netzwerkanfang eingefügt werden.

```
BEGIN
NETWORK
TITLE=
```

NETWORK weist den STEP7-AWL-Editor an, ein neues Netzwerk zu öffnen. *TITLE=* wird von STEP7-AWL als Netzwerkkommentar interpretiert.

Um eine bessere Übersichtlichkeit des Programms zu erreichen, sollte der Baustein in mehrere Netzwerke unterteilt werden. Jedem Netzwerk wird dabei eine möglichst aussagekräftige Überschrift, auch mehrzeilig, gegeben.

Programmierer, die nichts anderes als STEP7-AWL kennen, werden dankbar dafür sein. Es gibt dabei kein Problem mit Sprüngen. Die Sprungdistanz darf in STEP7-AWL $\pm 32k$ Worte sein, das Sprungziel darf auch in einem anderen Netzwerk liegen.

5.4 Korrektur der Befehle

Die von CoDeSys produzierte AWL entspricht vollständig der IEC1131. Leider trifft das für STEP7-AWL nicht zu. Am schwerwiegendsten sind unterschiedliche Befehle, bei STEP7-AWL zusätzlich verkompliziert durch Unterscheidungen hinsichtlich des Datentyps der an einer Operation beteiligten Variablen. Daher macht die Nachbearbeitung hinsichtlich der Befehle wohl den meisten Aufwand.

Die (wahrscheinlich unvollständige) Tabelle im Anhang zeigt, welche CoDeSys-Befehle durch welche STEP7-AWL-Befehle bzw. Befehlssequenzen zu ersetzen sind.

5.5 Korrektur der Aufrufe unterlagerter Bausteine

Die Notierung von Bausteinaufrufen ist in CoDeSys und STEP7-AWL stark unterschiedlich. Dennoch kann man die CoDeSys-Notierung relativ leicht in die STEP7-AWL-Form überführen. Es ist eine reine Fleißaufgabe.

Die Hauptunterschiede bestehen darin, dass

- CoDeSys diejenigen Eingaben des aufgerufenen Bausteins, die in SCL als Verknüpfung oder Rechenergebnis direkt am Formalparameter geschrieben sind, nach der Berechnung in AWL direkt in der Eingangsvariablen der aufgerufenen Instanz speichert. Am Bausteinaufruf selbst tauchen diese vorab bereits generierten Eingabevariablen nicht mehr auf.

STEP7-AWL lässt dieses Verfahren (wahrscheinlich) nicht zu. Das Verknüpfungs- oder Berechnungsergebnis muss in Zwischenvariablen gespeichert werden, die am Bausteinaufruf übergeben explizit werden.

Beispiel :

<u>SCL-Code:</u>			
<pre>EineBausteinInstanz (ixE1 := E1, ixE2 := E2 AND NOT E3, ixE3 := E4, ixE4 := E5, ixE5 := E6, ixE6 := E7 OR E8, ixE7 := NOT E9);</pre>			
<u>CoDeSys AWL:</u>		<u>STEP7-AWL:</u>	
LD	E2	VAR	
ANDN	E3	tmpE2 : BOOL;	
ST	EineBausteinInstanz.ixE2	tmpE6 : BOOL;	
LD	E7	tmpE7 : BOOL;	
OR	E8	END_VAR	
ST	EineBausteinInstanz.ixE6		
LDN	E9	...	
ST	EineBausteinInstanz.ixE7	U	E2
CAL	EineBausteinInstanz	UN	E3
	(ixE1 := E1,	=	tmpE2
	ixE3 := E4,	U	E7
	ixE4 := E5,	O	E8

<pre>ixE5 := E6)</pre>	<pre>= tmpE6 UN E9 = tmpE7 CALL EineBausteininstanz (ixE1 := E1, ixE2 := tmpE2, ixE3 := tmpE3, ixE3 := E4, ixE4 := E5, ixE5 := E6, ixE6 := tmpE6, ixE7 := tmpE7)</pre>
------------------------	---

- Dass CoDeSys die Ausgaben eines aufgerufenen Bausteines abholt durch Laden der Ausgangsvariablen der aufgerufenen Instanz.

STEP7-AWL dagegen gibt die Ausgänge in der gleichen Formalparameterliste zurück, in welche auch die Eingänge eingetragen werden. Sofern Ausgänge beim Abholen noch manipuliert werden, müssen die Bausteinrückgaben bei STEP7-AWL auf einer Zwischenvariablen geparkt werden.

Beispiel:

<u>SCL-Code:</u>	<u>CoDeSys AWL:</u>	<u>STEP7-AWL:</u>
<pre>EineBausteinInstanz (ixE1 := E1, ixE2 := E2 AND NOT E3, ixE3 := E4, ixE4 := E5, ixE5 := E6, ixE6 := E7 OR E8, ixE7 := NOT E9); A1 := EineBausteinInstanz.qxA1; A2 := EineBausteinInstanz.qxA2; A3 := EineBausteinInstanz.qxA3; A4 := SHL (EineBausteinInstanz.qbA4, 4);</pre>	<pre>LD E2 ANDN E3 ST EineBausteinInstanz.ixE2 LD E7 OR E8 ST EineBausteinInstanz.ixE6 LDN E9 ST EineBausteinInstanz.ixE7 CAL EineBausteinInstanz (ixE1 := E1, ixE3 := E4, ixE4 := E5, ixE5 := E6) LD EineBausteinInstanz.qxA1 ST A1 LD EineBausteinInstanz.qxA2 ST A2 LD EineBausteinInstanz.qxA3 ST A3 LD EineBausteinInstanz.qbA4 SHL 4 ST A4</pre>	<pre>VAR tmpE2 : BOOL; tmpE6 : BOOL; tmpE7 : BOOL; tmpA4 : WORD; END_VAR ... U E2 UN E3 = tmpE2 U E7 O E8 = tmpE6 UN E9 = tmpE7 CALL EineBausteininstanz (ixE1 := E1, ixE2 := tmpE2, ixE3 := tmpE3, ixE3 := E4, ixE4 := E5, ixE5 := E6, ixE6 := tmpE6, ixE7 := tmpE7, qxA1 := A1, qxA2 := A2, qxA3 := A3, qbA4 := tmpA4); l tmpA4 SLW 4</pre>

	T	A4
--	---	----

5.6 Übersetzen der AWL-Quellen im STEP7-Entwicklungssystem

Das Übersetzen der AWL-Quellen erfolgt analog dem Übersetzen von SCL-Bausteinen. Der einzige Unterschied besteht darin, dass jede AWL-Quelle manuell gebunden werden muss. Make-Dateien wie für SCL stehen nicht zur Verfügung.

Die Bausteine sind erst nach dem Übersetzen ladbar auf die SPS.

Anhang

Tabelle Umsetzung Befehlssatz CoDeSys nach STEP7-AWL

Arg steht für einen beliebigen Operanden.

CoDeSys-Befehl	Operanden- typ	Ersetzender STEP7- AWL-Befehl	Bemerkungen
Bausteinaufrufe			
CAL Arg	FB, FUN	CALL Arg	
Ladebefehle			
LD Arg	BOOL	U #Arg	
	BYTE	L #Arg	
	WORD		
	INT		
	DWORD		
LDN Arg	DINT		
	REAL		
	BOOL	UN #Arg	
	BYTE	L #Arg	Für INT möglicherweise anstelle von <i>INVI</i> auch Befehl zur Bildung des Zweierkomplements
	WORD	INVI	
INT			
DWORD	L #Arg	Für DINT möglicherweise anstelle von <i>INVD</i> auch Befehl zur Bildung des Zweierkomplements	
DINT	INVD		
Verknüpfungen			
AND Arg	BOOL	U #Arg	
	BYTE	L #Arg	
	WORD	UW	
ANDN Arg	DWORD	L #Arg	
		UD	
AND(Arg)	BOOL	U(U #Arg ...)	Wenn <i>AND(</i> eine eingeschachtelte Boolesche Verknüpfung einleitet
	BYTE	U(L #Arg ...)	Wenn <i>AND(</i> die Bewertung einer numerischen Variablen einleitet
ANDN Arg	WORD		
	DWORD	L #Arg	
		INVI	
OR Arg		L #Arg	
		INVD	
OR(Arg ...)	BOOL	O #Arg	
	BYTE	L #Arg	
	WORD	OW	
OR(Arg ...)	DWORD	L #Arg	
		OD	
OR(Arg ...)	BOOL	O(U #Arg ...)	Wenn <i>OR(</i> eine eingeschachtelte Boolesche Verknüpfung einleitet
	BYTE	O(L #Arg ...)	Wenn <i>OR(</i> die Bewertung einer numerischen Variablen einleitet
	WORD		
	DWORD		

CoDeSys-Befehl	Operanden- typ	Ersetzender STEP7- AWL-Befehl	Bemerkungen
ORN Arg	BOOL	ON #Arg	
	BYTE WORD	L INVI OW	
	DWORD	L INVD OD	
XOR Arg	BOOL	X #Arg	
	BYTE WORD	L XOW	
	DWORD	L XOD	
Speicherbefehle			
ST Arg	BOOL	= #Arg	
	BYTE WORD DWORD INT DINT REAL	T #Arg	
Negation und Arithmetik			
NOT Arg	BYTE WORD	L INVI	#Arg
	DWORD	L INVD	#Arg
ADD Arg	INT	L +I	#Arg
	DINT	L +D	#Arg
	REAL	L +R	#Arg
SUB Arg	INT	L -I	#Arg
	DINT	L -D	#Arg
	REAL	L -R	#Arg
MUL Arg	INT	L *I	#Arg
	DINT	L *D	#Arg
	REAL	L *R	#Arg
DIV Arg	INT	L /I	#Arg
	DINT	L /D	#Arg
	REAL	L /R	#Arg
Schiebebefehle			
SHL Arg	BYTE WORD	SLW #Arg	
	DWORD	SLD #Arg	
SHR Arg	BYTE WORD	SRW #Arg	

CoDeSys-Befehl	Operanden- typ	Ersetzender STEP7- AWL-Befehl	Bemerkungen
	DWORD	SRD #Arg	
Formatumwandlungen			
BOOL_TO_DWORD			Nicht unterstützt. Selbstdefinierte Funktion BOOL_TO_DWORD notwendig
DWORD_TO_BOOL		L L#0 <>D	Achtung: diese Befehlsfolge zerstört den Inhalt von Akku2
BYTE_TO_DWORD			Nicht direkt unterstützt. Möglicherweise kann Akku1 als BYTE geschrieben und als DWORD gelesen werden.
WORD_TO_INT			Immanent. Akku1 kann als WORD geschrieben und als INT gelesen werden.
WORD_TO_DWORD			Nicht direkt unterstützt. Möglicherweise kann Akku1 als WORD geschrieben und als DWORD gelesen werden.
INT_TO_WORD			Immanent. Akku1 kann als INT geschrieben und als WORD gelesen werden.
INT_TO_DWORD			Nicht direkt unterstützt. Möglicherweise kann Akku1 als INT geschrieben und als DWORD gelesen werden.
INT_TO_DINT		ITD	
INT_TO_REAL		ITD DTR	
DINT_TO_INT			Nicht unterstützt. Beschreiben von Akku1 mit DINT und Auslesen im 16-Bit-Format nur zulässig im Wertebereich 0 .. 7FFFh
DINT_TO_REAL		DTR	
DINT_TO_TIME			Nicht unterstützt. Selbstdefinierte Funktion DINT_TO_TIME notwendig
REAL_TO_DINT		RND	
REAL_TO_INT		RND ... ???	Unklar. Siehe DINT_TO_INT
LREAL_TO_INT		RND ... ???	LREAL_TO_INT wird von CoDeSys gleichbedeutend zu REAL_TO_INT gebraucht
Vergleiche			
LT Arg	INT	L #Arg <I	
	DINT	L #Arg <D	
	REAL	L #Arg <R	
LE Arg	INT	L #Arg <=I	
	DINT	L #Arg <=D	
	REAL	L #Arg <=R	

CoDeSys-Befehl		Operanden- typ	Ersetzender STEP7- AWL-Befehl	Bemerkungen
EQ	Arg	INT	L ==I	#Arg
		DINT	L ==D	#Arg
		REAL	L ==R	#Arg
NE	Arg	INT	L <>I	#Arg
		DINT	L <>D	#Arg
		REAL	L <>R	#Arg
GE	Arg	INT	L >=I	#Arg
		DINT	L >=D	#Arg
		REAL	L >=R	#Arg
GT	Arg	INT	L >I	#Arg
		DINT	L >D	#Arg
		REAL	L >R	#Arg
Sprungbefehle				
JMP			SPA	
JMPC			SPB	
Programmorganisation				
RET			BEA	