OMRON

マシンオートメーションコントローラ　NJ シリーズ

技術導入ガイド
IECプログラミング編

Technology
Introduction
Guide

SYSMAC
always in control

SBCA-405B

おことわり

（1）本ガイドの内容の一部または全部を無断で複写、複製、転載することを禁じます。
（2）本ガイドの内容に関しては、改良のため予告なしに仕様などを変更することがあります。
あらかじめご了承ください。
（3）本ガイドの内容に関しては、万全を期しておりますが、万一ご不審な点や誤りなどお気づき
のことがありましたら、お手数ですが弊社支店または営業所までご連絡ください。その際、巻末
記載のMan.No（マニュアルNo.）も併せてお知らせください。

はじめに

マシンオートメーションコントローラ NJ シリーズをご検討のお客様へ

マシンオートメーションコントローラ NJ シリーズは、お客さまの装置設計の生産性向上を第一に考え、IEC 61131-3 に準拠した言語仕様を搭載しています。データの変数化、変数に対応した命令の整備や、本格的な ST 言語を導入しました。
NJ シリーズ 技術導入ガイド IEC プログラミング編（以降、「本ガイド」と省略する場合があります）は、ＮＪ シリーズをご採用いただくことによる装置設計のメリットをまとめています。

なお、本ガイドには安全に関する注意事項を含め、使用する際の詳細内容については記載していません。必ず、本ガイドで使用する各機器のマニュアルや取扱説明書を入手し、「安全上のご注意」「安全上の要点」「使用上の注意」などの安全に関する注意事項を含め、使用する際の詳細内容を確認した上で使用してください。本ガイドは、次の方を対象に記述しています。

対象となる読者の方々

本ガイドは、次の方を対象に記述しています。
電気の知識（電気工事士あるいは同等の知識）を有する方で
- FA 機器の導入を担当される方
- FA システムを設計される方

対象となる製品

本ガイドは、以下の製品を対象にしています。
- マシンオートメーションコントローラ NJ シリーズ CPU ユニット
- オートメーションソフトウェア Sysmac Studio

ご承諾事項

マシンオートメーションコントローラ NJ シリーズ CPU ユニット

「当社商品」について特別の合意がない場合には、お客様のご購入先にかかわらず、本ご承諾事項記載の条件を適用いたします。

● 定義

本ご承諾事項中の用語の定義は次のとおりです。

- 「当社商品」:「当社」の FA システム機器、汎用制御機器、センシング機器、電子・機構部品
- 「カタログ等」:「当社商品」に関する、ベスト制御機器オムロン、電子・機構部品総合カタログ、その他のカタログ、仕様書、取扱説明書、マニュアル等であって電磁的方法で提供されるものも含みます。
- 「利用条件等」:「カタログ等」に記載の、「当社商品」の利用条件、定格、性能、動作環境、取り扱い方法、利用上の注意、禁止事項その他
- 「お客様用途」:「当社商品」のお客様におけるご利用方法であって、お客様が製造する部品、電子基板、機器、設備またはシステム等への「当社商品」の組み込み、または利用を含みます。
- 「適合性等」:「お客様用途」での「当社商品」の (a) 適合性、(b) 動作、(c) 第三者の知的財産の非侵害、(d) 法令の遵守および (e) 各種規格の遵守

● 記載事項のご注意

「カタログ等」の記載内容については次の点をご理解ください。

- 定格値および性能値は、単独試験における各条件のもとで得られた値であり、各定格値および性能値の複合条件のもとで得られる値を保証するものではありません。
- 参考データはご参考として提供するもので、その範囲で常に正常に動作することを保証するものではありません。
- 利用事例はご参考ですので、「当社」は「適合性等」について保証いたしかねます。
- 「当社」は、改善や当社都合等により、「当社商品」の生産を中止し、または「当社商品」の仕様を変更することがあります。

● ご利用にあたってのご注意

ご採用およびご利用に際しては次の点をご理解ください。

- 定格・性能ほか「利用条件等」を遵守しご利用ください。
- お客様ご自身にて「適合性等」をご確認いただき、「当社商品」のご利用の可否をご判断ください。「当社」は「適合性等」を一切保証いたしかねます。
- 「当社商品」がお客様のシステム全体の中で意図した用途に対して、適切に配電・設置されていることをお客様ご自身で、必ず事前に確認してください。
- 「当社商品」をご使用の際には、(i) 定格および性能に対し余裕のある「当社商品」のご利用、冗長設計などの安全設計、(ii)「当社商品」が故障しても、「お客様用途」の危険を最小にする安全設計、(iii) 利用者に危険を知らせるための、安全対策のシステム全体としての構築、(iv)「当社商品」および「お客様用途」の定期的な保守、の各事項を実施してください。
- 「当社」は DDoS 攻撃（分散型 DoS 攻撃）、コンピュータウイルスその他の技術的な有害プログラム、不正アクセスにより、「当社商品」、インストールされたソフトウェア、またはすべてのコンピュータ機器、コンピュータプログラム、ネットワーク、データベースが感染したとしても、そのことにより直接または間接的に生じた損失、損害その他の費用について一切責任を負わないものとします。

お客様ご自身にて、①アンチウイルス保護、②データ入出力、③紛失データの復元、④「当社商品」またはインストールされたソフトウェアに対するコンピュータウイルス感染防止、⑤「当社商品」に対する不正アクセス防止についての十分な措置を講じてください。

- 「当社商品」は、一般工業製品向けの汎用品として設計製造されています。従いまして、次に掲げる用途での使用は意図しておらず、お客様が「当社商品」をこれらの用途に使用される際には、「当社」は「当社商品」に対して一切保証をいたしません。ただし、次に掲げる用途であっても「当社」の意図した特別な商品用途の場合や特別の合意がある場合は除きます。
  - (a) 高い安全性が必要とされる用途（例：原子力制御設備、燃焼設備、航空・宇宙設備、鉄道設備、昇降設備、娯楽設備、医用機器、安全装置、その他生命・身体に危険が及びうる用途）
  - (b) 高い信頼性が必要な用途（例：ガス・水道・電気等の供給システム、24 時間連続運転システム、決済システムほか権利・財産を取扱う用途など）
  - (c) 厳しい条件または環境での用途（例：屋外に設置する設備、化学的汚染を被る設備、電磁的妨害を被る設備、振動・衝撃を受ける設備など）
  - (d) 「カタログ等」に記載のない条件や環境での用途
- 上記の (a) から (d) に記載されている他、「本カタログ等記載の商品」は自動車（二輪車含む。以下同じ）向けではありません。自動車に搭載する用途には利用しないで下さい。自動車搭載用商品については当社営業担当者にご相談ください。

● 保証条件

「当社商品」の保証条件は次のとおりです。

- 保証期間　ご購入後 1 年間といたします。
  （ただし「カタログ等」に別途記載がある場合を除きます。）
- 保証内容　故障した「当社商品」について、以下のいずれかを「当社」の任意の判断で実施します。
  - (a) 当社保守サービス拠点における故障した「当社商品」の無償修理
    （ただし、電子・機構部品については、修理対応は行いません。）
  - (b) 故障した「当社商品」と同数の代替品の無償提供
- 保証対象外　故障の原因が次のいずれかに該当する場合は、保証いたしません。
  - (a) 「当社商品」本来の使い方以外のご利用
  - (b) 「利用条件等」から外れたご利用
  - (c) 本ご承諾事項「ご利用にあたってのご注意」に反するご利用
  - (d) 「当社」以外による改造、修理による場合
  - (e) 「当社」以外の者によるソフトウェアプログラムによる場合
  - (f) 「当社」からの出荷時の科学・技術の水準では予見できなかった原因
  - (g) 上記のほか「当社」または「当社商品」以外の原因（天災等の不可抗力を含む）

● 責任の制限

本ご承諾事項に記載の保証が、「当社商品」に関する保証のすべてです。
「当社商品」に関連して生じた損害について、「当社」および「当社商品」の販売店は責任を負いません。

● 輸出管理

「当社商品」または技術資料を、輸出または非居住者に提供する場合は、安全保障貿易管理に関する日本および関係各国の法令・規制を遵守ください。お客様が法令・規則に違反する場合には、「当社商品」または技術資料をご提供できない場合があります。

オートメーションソフトウェア Sysmac Studio

● 保証内容

(1) 保証期間

本ソフトウェアの保証期間は、ご購入後またはご指定の場所に納入後 1 年といたします。

(2) 保証範囲

a) 本ソフトウェアの使用許諾を受けたお客様が、上記保証期間中にコンピュータ・プログラムの瑕疵（『Sysmac Studio Version 1 オペレーションマニュアル（SBCA-470)』との重要な不一致）を発見し、当社に返却した場合は、当社は瑕疵のないコンピュータ・プログラムを記録した媒体と交換いたします。もしくは、当社の選択により、当社ホームページより瑕疵のないコンピュータ・プログラムをダウンロードしていただく方法により提供いたします。また、当社の責任によるコンピュータ・プログラムの記録媒体の不良を発見し、当社に返却した場合、当社は、無償で、良品の媒体に記録したコンピュータ・プログラムと交換いたします。

b) 万一、当社がコンピュータ・プログラムの瑕疵を除去できないと判断した場合は、お客様が本ソフトウェア購入代金として支払った金額をお返しいたします。

● 責任の制限

(1) 前条に定める交換または購入代金の返金は、本ソフトウェアの保証責任のすべてを定めるものであり、当社は本ソフトウェアの瑕疵により発生した、お客様の直接的、間接的あるいは波及効果による損害等いかなる損害に対しても一切の責任を負いません。

(2) 当社は、本ソフトウェアを当社以外の第三者が変更、改造することにより生じた瑕疵につきましては、一切責任を負いません。

(3) 当社は、本ソフトウェアに基づき、当社以外の第三者が開発したソフトウェアおよびそれにより生じた結果について一切の責任を負いません。

● 本ソフトウェアの用途

本ソフトウェアを『Sysmac Studio Version 1 オペレーションマニュアル（SBCA-470）』に記載の用途以外の用途で使用しないでください。

● 仕様の変更

本ソフトウェアの仕様および付属品は改善またはその他の事由により、必要に応じて、変更される場合があります。

● サービスの範囲

本ソフトウェアの価格には、技術者派遣などのサービス費用は含まれておりません。

● 適用範囲

以上の内容は、日本国内での取引および使用を前提としております。日本国外での取引および使用に関しては、当社営業担当者までご相談ください。

注意事項

- 実際のシステム構築に際しては、システムを構成する各機器や装置の仕様をご確認のうえ、定格や性能に対し余裕を持った使い方をしてください。万一故障があっても危険を最小にする安全回路などの安全対策を講じてください。
- システムを安全に使用していただくため、システムを構成する各機器や装置のマニュアルや取扱説明書などを入手してください。「安全上のご注意」「安全上の要点」「使用上の注意」など安全に関する注意事項を含め、内容を確認のうえ使用してください。
- システムが適合すべき規格や法規、または規制については、お客様自身で確認してください。

商標

- Sysmac は、オムロン株式会社製 FA 機器製品の日本およびその他の国における商標または登録商標です。
- Windows、Windows 98、Windows XP、Windows Vista、Windows 7 は米国 Microsoft Corporation の米国およびその他の国における登録商標です。
- Intel、インテル、Intel ロゴ、インテル Atom は、米国およびその他の国におけるインテル コーポレーションの商標です。
- EtherCAT® は、ドイツのベッコフオートメーション株式会社がライセンスを供与した登録商標であり、特許取得済みの技術です。
- SD ロゴは、SD-3C,LLC の商標です。

その他、記載されている会社名と製品名などにつきましては、各社の登録商標または商標です。

ソフトウェアのライセンスと著作権

本製品にはサードパーティ製のソフトウェアが組み込まれています。このソフトウェアに関連するライセンスと著作権については、http://www.fa.omron.co.jp/nj_info_j/ をご覧ください。

関連マニュアル

関連する NJ シリーズのマニュアルは、下表のとおりです。併せてご覧ください。

マニュアル名称	Man.No.	形式	用途	内容
NJ シリーズ　CPU ユニット ユーザーズマニュアル ハードウェア編	SBCA-466	形 NJ501-□□□□ 形 NJ301-□□□□ 形 NJ101-□□□□	NJ シリーズ CPU ユニットの概要／設計／取付／保守などの基本的な仕様について知りたいとき。おもにハードウェアに関する情報。	NJ シリーズのシステム全体概要、および CPU ユニットに関して、以下の内容を説明します。 • 特長やシステム構成 • 概要 • 各部の名称と機能 • 一般仕様 • 設置と配線 • 保守点検
NJ/NX シリーズ　CPU ユニット ユーザーズマニュアル ソフトウェア編	SBCA-467	形 NX701-□□□□ 形 NX102-□□□□ 形 NX1P2-□□□□ 形 NJ501-□□□□ 形 NJ301-□□□□ 形 NJ101-□□□□	NJ/NX シリーズ CPU ユニットのプログラミング／システムの立ち上げについて知りたいとき。おもにソフトウェアに関する情報。	NJ/NX シリーズ CPU ユニットに関して、以下の内容を説明します。 • CPU ユニットの動作 • CPU ユニットの機能 • 初期設定 • IEC 61131-3 ベースの言語仕様とプログラミング
NJ/NX シリーズ　CPU ユニット ユーザーズマニュアル モーション制御編	SBCE-433	形 NX701-□□□□ 形 NX102-□□□□ 形 NX1P2-□□□□ 形 NJ501-□□□□ 形 NJ301-□□□□ 形 NJ101-□□□□	モーション制御の設定やプログラミングの考え方について知りたいとき。	モーション制御のための CPU ユニットの設定や動作、プログラミングの考え方について説明します。
NJ/NX シリーズ コマンドリファレンス マニュアル基本編	SBCA-468	形 NX701-□□□□ 形 NX102-□□□□ 形 NX1P2-□□□□ 形 NJ501-□□□□ 形 NJ301-□□□□ 形 NJ101-□□□□	オムロンが提供する命令仕様の詳細について知りたいとき。	各命令（IEC 61131-3 仕様）の詳細を説明します。
NJ/NX シリーズ コマンドリファレンス マニュアル モーション編	SBCE-434	形 NX701-□□□□ 形 NX102-□□□□ 形 NX1P2-□□□□ 形 NJ501-□□□□ 形 NJ301-□□□□ 形 NJ101-□□□□	モーション命令仕様の詳細について知りたいとき。	各モーション命令の詳細を説明します。
NJ/NX シリーズ トラブルシューティング マニュアル	SBCA-469	形 NX701-□□□□ 形 NX102-□□□□ 形 NX1P2-□□□□ 形 NJ501-□□□□ 形 NJ301-□□□□ 形 NJ101-□□□□	NJ/NX シリーズで検出する異常の詳細について知りたいとき。	NJ/NX シリーズ システムにて検出する異常管理の考え方と各異常項目について説明します。
Sysmac Studio Version 1 オペレーションマニュアル	SBCA-470	形 SYSMAC -SE2□□□	Sysmac Studio の操作方法、機能について知りたいとき。	Sysmac Studio の操作方法について説明します。
CX-Designer ユーザーズマニュアル	SBSA-532		プログラマブルターミナル NS シリーズの画面データを作成したいとき。	CX-Designer の操作方法について説明します。

マニュアル改訂履歴

マニュアル改訂記号は、表表紙・裏表紙に記載されている Man.No. の後尾に付記されます。

改訂記号	改訂年月	改訂理由・改訂ページ
A	2012 年 5 月	初版
B	2018 年 12 月	ご承諾事項の更新に伴う改訂、関連マニュアルの変更

目次

1

IEC 61131-3 プログラミングのすすめ

この章では、IEC 61131-3 プログラミングの背景と有用性、および NJ シリーズ CPU ユニットの特長について説明します。

1-1　IEC 61131-3 の概要と特長

日本国内市場の成熟にともない、機械装置産業は、新興国をはじめとする海外に成長の場を求めるようになりました。現在の輸出比率は 30% を超え、生産拠点自体を海外に持つことが当たり前となっています。機械メーカの競争の場も、国内からグローバルへと移ってきています。

近年の生産設備の高度化や多様化に加えて、グローバル競争での優位性を確保するために、機械装置の多機能化や高機能化および性能向上への要求はますます高まっています。
これを受けて、機械の制御をつかさどるプログラマブルロジックコントローラ（以降 PLC）のプログラムも更なる大規模化と複雑化が進んでいます。

PLC は、リレー回路を代替えする制御装置として開発されました。従来の PLC のプログラムは、シーケンス制御に適したラダー図言語での作成が主流でした。ラダー図言語は、リレー回路を記号化した言語です。
しかし、機械自体の多機能化や高機能化により、PLC のプログラムは、シーケンス制御だけでなく、以下のような多様な制御に対応しています。

- モーション制御
- 周辺機器と接続するためのインタフェース制御
- 情報処理など

多様な制御への対応が、PLC のプログラムの大規模化や複雑化のひとつの要因となっています。このため、プログラム全体に占めるシーケンス制御の比率が小さくなる傾向が強くなっています。

プログラムの大規模化や複雑化は、必然的に開発工数の増加を招きますが、機械設備のライフサイクルは短期化するばかりです。したがって、プログラム開発の生産性向上と品質向上の両立が、ますます深刻な課題となっています。

一方、グローバルでの競争は、プログラミングを担う開発設計技術者やメンテナンス技術者などの人材確保にも課題を投げかけています。
海外への生産拠点や開発拠点の進出には、現地での優秀な人材の確保や教育が不可欠となります。
国内の現場においても、習熟した優秀な技術者が次々リタイアしていくことで人材不足が深刻化しています。人材不足を補完するための教育コストも増加傾向にあります。

生産性向上と品質向上の両立には、効率的なプログラミング手法の導入が有効な手段と考えられます。
そして、グローバル観点での人材確保や教育コストの低減には、局所的、独自的な手法から国際的に標準化されたプログラミング手法の採用が効果的です。

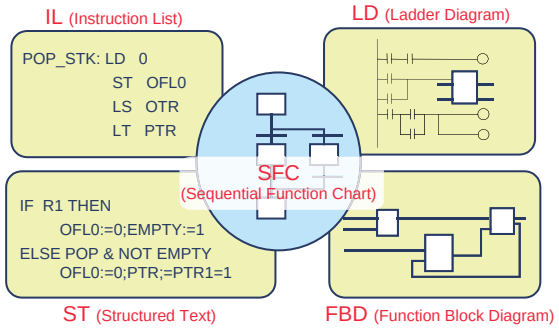
このような状況において、PLC の国際標準規格である IEC 61131-3 に準拠したプログラミング手法の導入が進んでいます。

IEC 61131-3 の概要

国際電気標準会議 (IEC) は、PLC に関する標準規格を規定しています。
- IEC 61131-1(JIS B 3501)：PLC の一般情報
- IEC 61131-2(JIS B 3502)：PLC 装置への要求事項および試験
- IEC 61131-3(JIS B 3503)：PLC のプログラム言語

PLC のプログラム言語の規格は、標準規格の第３章として 1993 年に第１版が、2002 年に改訂第２版が規定されています。(以降、IEC 61131-3)
この規格は日本では JIS 化され、JIS B 3503 として規定されています。

IEC 61131-3 で規定されている言語は以下の５種類です。
- IL(Instruction List)：アセンブラ言語相当のニモニックを規格化したテキスト言語
- LD(Ladder Diagram)：リレー回路を記号化したグラフィック言語
- ST(Structured Text)：高級言語の PASCAL に似た、演算や情報処理に適したテキスト言語
- FBD(Function Block Diagram)：機能をブロックに部品化したグラフィック言語
- SFC(Sequential Function Chart)：工程を状態遷移図のように記述できるグラフィック言語



IEC 61131-3 では、部品化の手段として「変数」、および「ファンクション（以降、FUN)」や「ファンクションブロック（以降、FB)」など「POU（Program Organization Units)」と呼ばれるプログラムの単位についても規定されており、プログラムの単位をもとにしたマルチタスクや、データの流れの「ソフトウェアモデル」がプログラムの構造として規定されています。

「POU」によって部品化するロジックを切り出し、物理的なメモリアドレスに固定化されない「変数」を用いることで、特定のハードウェアに縛られない、より可搬性の高いプログラムモジュールの作成が提唱されています。

NJ シリーズは、IEC 61131-3 に準拠しています。ラダー図言語 (LD) に加えて ST 言語 (ST) を使用でき、それぞれの言語で FB や FUN といった POU を記述できます。

IEC 61131-3 プログラミングの導入で問題解決

プログラミングの問題を解決するためには、プログラムの部品化や構造化が必要不可欠となります。
プログラムを構造化すると処理内容を把握しやすくなり、ソフトウェア品質の向上や保全性の向上を容易化することが可能です。
NJ シリーズでは、ユーザプログラムを POU（プログラム構成単位）という部品単位で作成します。複数の POU を組み合わせて使用することで、ユーザプログラム全体を構造化できます。

IEC 61131-3 プログラミングの導入により期待できる価値

- 可読性（Readability）
- 再利用性（Reusability）
- 多種類のプログラム言語（Variety of programming language:IL/LD/ST/SFC/FBD）
- 国際標準規格（Global standard）
- 教育コストの削減（Reducing education cost）

可読性

ユーザプログラムを構造的に作成することで、装置全体の工程フローと各処理の順序を把握しやすくなります。また、各信号にわかりやすい変数名を付けることで、処理の内容を読み取りやすくできます。

再利用性

変数によるプログラミングにより、メモリマップを意識することなくプログラムを作成できます。また、ユーザ定義 FB やユーザ定義の派生データ型を作成することで、プログラムを部品化できます。さらに、作成した FB をライブラリに保存することもできます。
これらにより、機器構成を変更したり他のプログラムに流用したりする場合でも、プログラムを容易に再利用できるため、解析工数や品質確保のための工数を削減できます。

多種類のプログラム言語

日本で広く普及しているラダー図言語はリレー回路と等価であるため、FA 電気技術者に受け入れられてきました。しかし、複雑な演算や情報処理の表現に適していないなどのデメリットもあります。一方、コンピュータプログラム用の高級言語に似た ST 言語は、演算や情報処理の表現に適しています。
このように、制御の目的や使用者の保有技術によって、最適なプログラム言語を選択できることが求められています。
IEC 61131-3 で規定されている言語から最適な言語を選択することで、プログラムを改善したり開発効率を向上したりすることが可能です。

国際標準規格

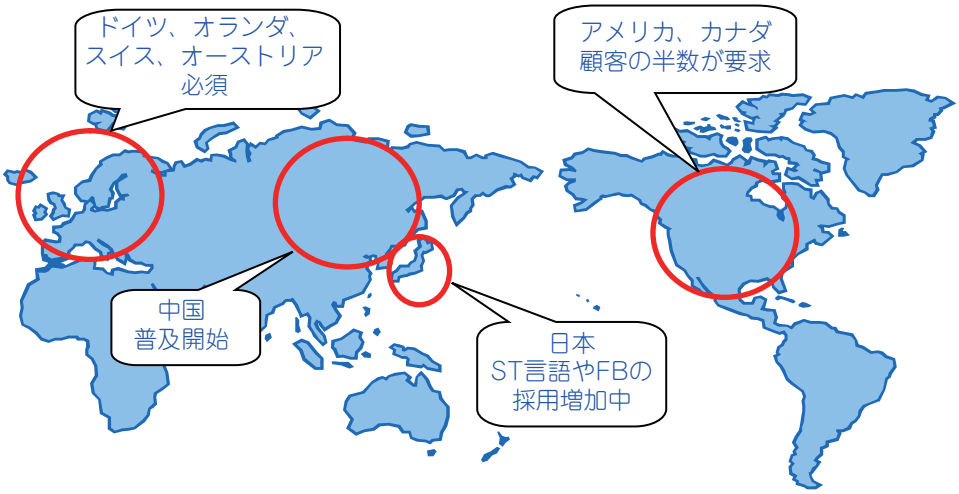
急速なグローバル化により、日本国内で生産された装置は、新興国を主とする海外に輸出されています。しかし、海外での日本製 PLC のシェアは、欧州製や北米製の PLC に及んでいません。たとえば、中国では欧州企業の影響により、国際標準規格に準拠した PLC が普及しつつあります。
IEC 611131-3 プログラミングを導入することで、国際的な競争力を高めることも可能となります。

教育コストの削減

欧米や中国でも標準規格となっている IEC 61131-3 プログラミングを導入することで、新たに必要となる教育コストを削減することが可能となります。IEC 61131-3 のプログラム言語は、コンピュータプログラム用のソフトウェアモデルをベースとしているため、プログラミングの経験を持つ若手技術者も少なくありません。

IEC 61131-3 の普及状況

IEC 61131-3 は国際標準規格ですが、現状はエリアごとに普及の度合いに差が生じています。



規格の発祥が欧州であったため、欧州各国での普及が先行してきました。
日本では、従来のプログラム資産やプログラミングスタイルを重視する傾向があり、欧州に比べると普及が遅れている状況です。
一方で、中国や韓国などの新興国では、欧州からの影響と過去のプログラム資産が少ないことから、生産性の高い国際標準手法として急速に浸透してきています。
米国は欧州と比べて普及度は低いものの、日本よりも普及が進んでいます。
このように、IEC 61131-3 は世界的な規模でみると、確実に普及が進んでいると言えます。

この状況を受け、オムロンを含めた国内外の PLC メーカから、規格に準拠したプログラミング環境とコントローラが提供され、製品数も増加しています。
IEC 61131-3 は世界レベルで今後ますます普及していくでしょう。
日本が国際的な競争に立ち向かっていくためには、このような状況を認識することが必要と考えられます。

1-2　NJ シリーズ CPU ユニットの特長

マシンオートメーションコントローラ NJ シリーズは、機械制御に必要な機能、高速性能と、産業用コントローラとしての安全性、信頼性、保守性とを両立した次世代コントローラです。
プログラマブルコントローラの機能に、モーション制御に必要な機能を付加した統合型コントローラとして、視覚センサ、モーション、I/O などを、高速な EtherCAT 上で同期して制御できます。

高速かつ高精度な制御を実現

インテル ®Atom™ プロセッサーによって、ユーザプログラムを高速に実行できます。ラダー図命令は最速 1.9ns、倍精度浮動小数点演算命令は最速 26ns で実行できます。
IEC 61131-3 に沿ったプログラムは、従来のラダー図言語だけの実行と比較して、コントローラへの負荷が大きくなる傾向がありますが、NJ シリーズは問題とならない実行性能を持っています。

国際標準規格 IEC 61131-3 準拠のプログラミング言語仕様

IEC 61131-3 に準拠した言語仕様を搭載しています。一部、オムロン独自の改良を加えています。
PLCopen に準拠したモーション制御命令をはじめ、IEC 規格に準拠した豊富な命令群（POU）が使用できます。ST 言語は、PLCopen CL（Conformity Level）認証取得済みです。

メモリマップを意識しない、変数によるプログラミング

パソコン上で高級言語の変数を使用するときと同様に、すべてのデータは変数によりアクセスします。変数を作成すると、CPU ユニットのメモリ上に変数が自動的に割り付けられるため、メモリマップを意識する必要はありません。このため、ハードウェア構成を変更しても、ソフトウェアの変更を最小限に抑えられます。

マルチタスク対応

I/O リフレッシュやユーザプログラム実行などの一連の処理をタスクに割り付け、タスクの実行条件と実行順序を指定することができます。複数のタスクを組み合わせることでアプリケーションに合わせた柔軟な制御が構築できます。

オートメーションソフトウェア Sysmac Studio

Sysmac Studio は、NJ シリーズをはじめとするマシンオートメーションコントローラ、および EtherCAT スレーブなどの設定、プログラミング、デバッグ、メンテナンスのための、統合開発環境を提供するソフトウェアです。

1-3　本ガイドについて

第 1 章では、日本の機械産業の現状に起因する PLC プログラミングの課題、その解決手段としての IEC 61131-3 国際標準規格、規格に準拠したオムロンの新しいマシンオートメーションコントローラ NJ シリーズの概要を説明してきました。

以降の章では、具体的な課題を題材とし、生産性と品質を向上するためのオムロン独自の仕様を含め、NJ シリーズでの IEC 61131-3 プログラミングの効用を説明しています。

本ガイドで説明しているさまざまな事例をお読みいただくことで、IEC 61131-3 に沿ったプログラミングの効用と、オムロンの課題解決に対する取り組みをご理解いただけるものと考えております。

2

NJ シリーズでのプログラミングの効用

本章では、NJ シリーズでの IEC 61131-3 に沿ったプログラミングの効用について説明します。

性能

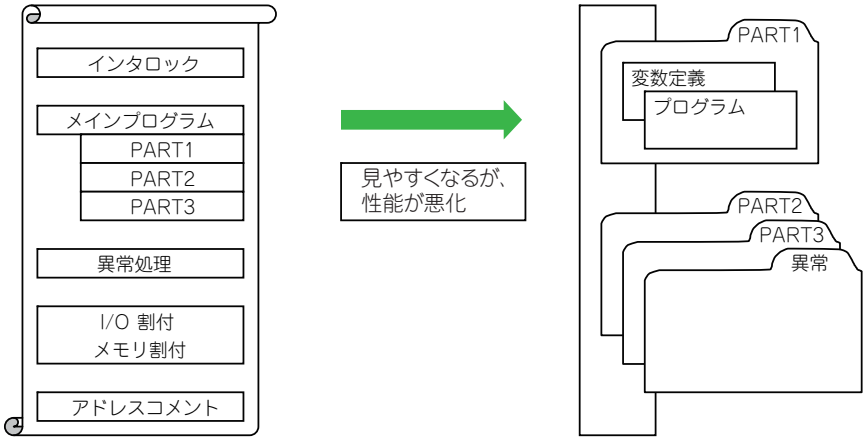
2-1　実行性能とプログラミング効率を両立

従来の問題点

生産設備への要求の高度化にともない、PLC を中心とする制御システムの大規模化や複雑化が加速しています。

プログラムを巻物状に記述する従来のラダー図言語では、ブロック化による構造化や他の技術者の理解が難しい点があります。そのため、大規模かつ複雑なプログラムの要求に対して、ラダー図言語によるプログラミングスタイルは、現実的でなくなりつつあります。

これらの問題を解決する手段として、IEC 61131-3 に対応した PLC では、変数を活用したプログラミングをベースに、ST 言語やファンクションブロックなどの機能が提供されています。
しかし、これらの機能を活用すると、従来の PLC ではプログラム実行性能が悪化するため、機能を十分には活用できない場合がありました。



NJ で解決！

NJ シリーズでは、プログラムのコンパイルから実行までのしくみを一新しました。
ST 言語を使用したり、ファンクションブロックを活用してプログラムを構造化したりした場合でも、ラダー図言語で、すべて展開して記述したプログラムとほぼ同等の実行性能を引き出せます。
このように、NJ シリーズでは、プログラミング手法によるプログラム実行性能への影響を最小限に抑えました。また、CPU 演算処理部にインテル ® Atom™ プロセッサーを搭載し、高速演算を実現しています。
オムロンの NJ シリーズなら、プログラムの実行性能を保ちながら可読性や再利用性にすぐれた仕様変更にも強いプログラムを作成することが可能です。

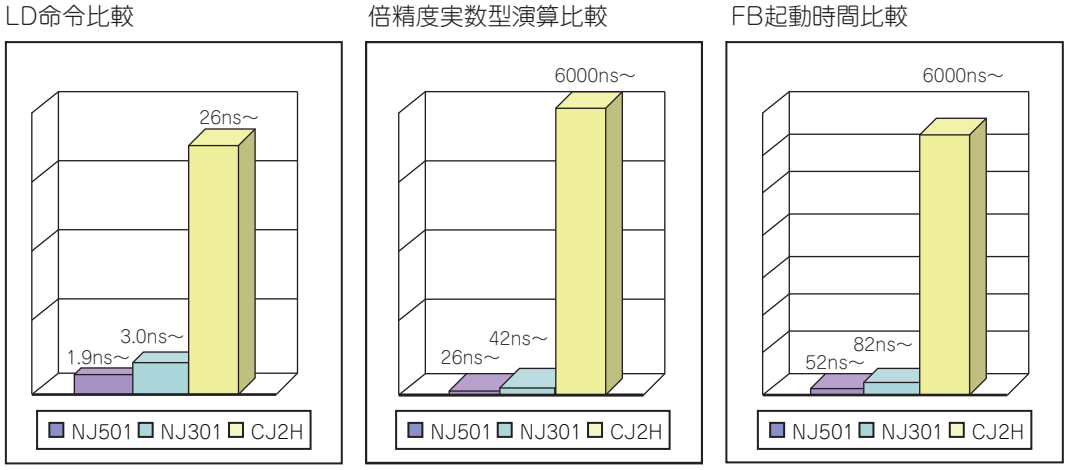
解説

プログラム実行性能向上

- プログラム実行方式
  NJ シリーズでは、プログラムを実行する前に機械語に変換するコンパイル方式を採用しています。プログラムの実行ごとに機械語に変換するインタープリタ方式に比べ、プログラム実行速度を向上することができます。

- 高速汎用プロセッサ
  演算処理を実行するプロセッサが性能の要となります。
  NJ シリーズでは、インテル® Atom™ プロセッサを搭載することで高速演算を実現しました。
  もちろん、従来のプログラマブルロジックコントローラ（PLC）の信頼性も継承しています。
  NJ シリーズは、高速処理性能と産業用コントローラとしての信頼性を両立しました。

これらの技術により、従来の PLC から大幅な性能向上を実現しています。



- ST 言語
  ST 言語は、PASCAL をベースに規定された構造化テキスト言語です。数値演算式の表現やネスティングした条件分岐など、ラダー図言語が苦手としている用途で高い効果を発揮します。
  以下に ST 言語の例を示します。

```
------------------------------------------------------
(* 消費電力を求める *)
(* 消費電力 = 抵抗 × 電流 ^2 *)
Power := Resistance * Current ** LREAL#10#2.0;
------------------------------------------------------
```

  ST 言語を使ったプログラミングの詳細は、「2-11 ST 言語の活用で複雑な演算の記述効率アップ」(P.2-33) を参照してください。

- ファンクションブロック
  ファンクションブロックとは、ST 言語またはラダー図言語でアルゴリズムを記述したプログラム部品です。
  プログラムの中で繰り返し使用する定型的なアルゴリズムを、ファンクションブロックとしてひとまとめにすることで、アルゴリズムを再利用することができます。
  ファンクションブロックの詳細は、「2-5 軸変数で簡単モーションプログラミング」(P.2-14) を参照してください。

2-2　変数プログラミングで再利用性アップ

従来の問題点

物理アドレスによるプログラミングでは、データ型や配列サイズを考慮してアドレスの割り当てを設計する必要がありました。そのため、プログラムを複製して再利用したり、データの型やサイズを変更したりすることに多くの工数が必要でした。

物理アドレス	コメント
D2000	ワーク1位置
D2001	ワーク1速度
D2002	ワーク2位置

ワーク1速度を32ビットデータに変更

物理アドレス	コメント
D2000	ワーク1位置
D2001	ワーク1速度(L)
D2002	ワーク1速度(H)
D2003	ワーク2位置

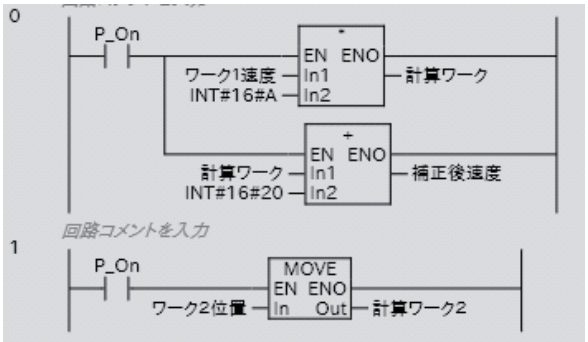
NJ で解決！

NJ シリーズでは、変数プログラミングをサポートしています。
変数プログラミングでは、データサイズを考慮しながらアドレスの割り当てを設計する必要はありません。また、プログラム内でファンクションブロックを複数回呼び出したり、他のプログラムで再利用したりする際にも、アドレスの修正は不要です。
そのため、特にデータ型指定や配列を使用したプログラムを、ストレスなく再利用することができます。

上記の例では、変数のデータ型を変更するだけで修正が済むため、アドレスの重複確認や、プログラムを変更する作業は不要です。

変数名	データ型
ワーク1位置	INT（整数型16ビット）
ワーク1速度	INT（整数型16ビット）
ワーク2位置	INT（整数型16ビット）

変数名	データ型
ワーク1位置	INT（整数型16ビット）
ワーク1速度	DINT（整数型32ビット）
ワーク2位置	INT（整数型16ビット）

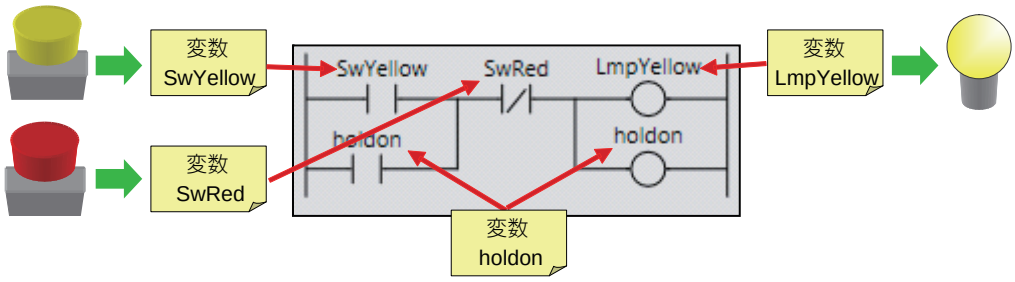


変数「ワーク1速度」のデータ型を変更するだけです。
プログラムを修正する必要はありません。

解説

- 変数

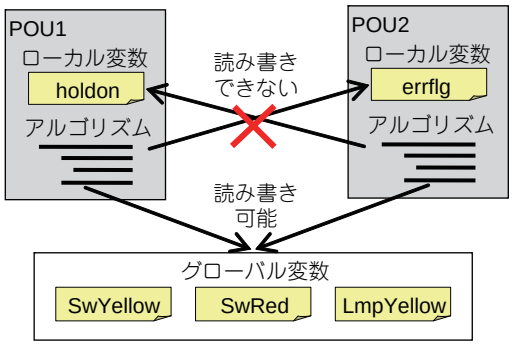
  変数とは、外部とやりとりした入出力データや、POU 内部処理時の一時データを格納する入れ物です。ローカル変数やグローバル変数などの種類、および名前やデータ型などの属性があります。
  NJ シリーズでは、外部との入出力情報のやりとりやデータの演算などを、すべて変数を介して処理するため、ハードウェアのメモリアドレスに依存しないプログラミングが可能です。



- ローカル変数とグローバル変数

  ローカル変数は、定義された POU 内だけで読み書きが可能な変数です。複数の POU で同じ変数名を使用しても、別々の変数として扱われます。
  そのため、他の POU で使用されている変数名との重複を気にすることなく、プログラムを作成することができます。また、同一 POU の複数回使用や、複数人でのプログラミングも容易です。
  グローバル変数は、すべての POU から読み書きが可能な変数です。

  たとえば上記プログラムでは、押しボタンスイッチとランプにアクセスするための変数をグローバル変数として定義しています。また、自己保持回路で使用する変数は、ローカル変数として定義しています。



- 変数のデータ型

  データ型とは、変数が表現する値の形式と範囲を特定する属性です。変数を定義する場合には、必ずデータ型を指定する必要があります。

  NJ シリーズでは、あらかじめ仕様が決まっている基本データ型として、以下の分類があります。他にもユーザが仕様を定義する派生データ型があります。

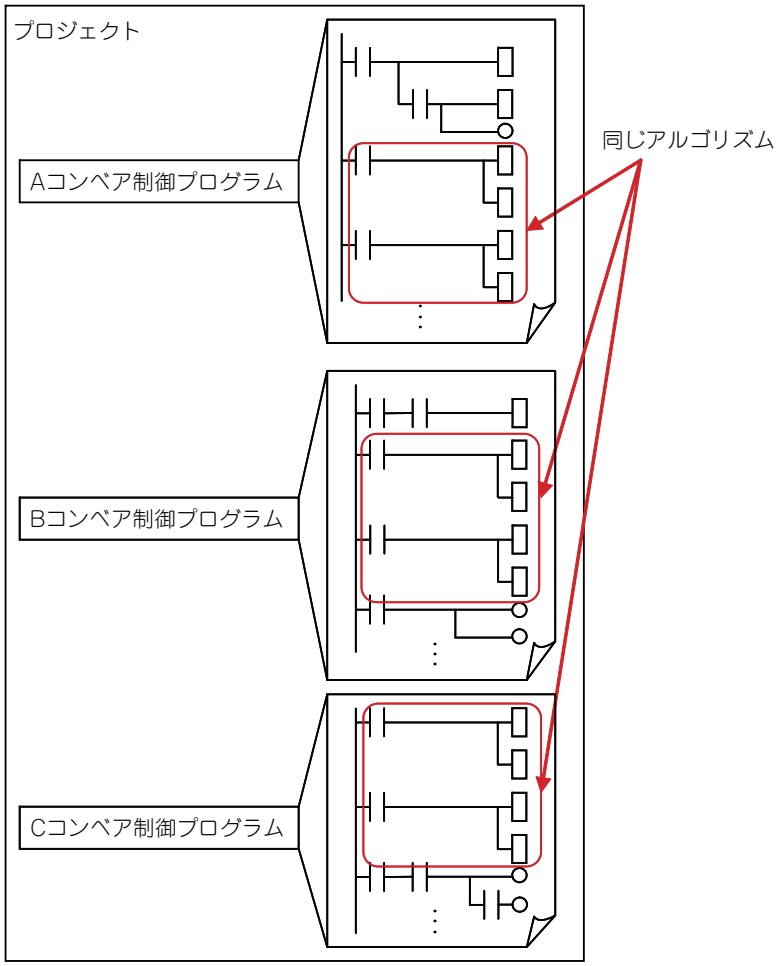
分　類	データ型	データサイズ	定　義
ブール型	BOOL	1 ビット	値が 0 または 1 のいずれかのデータ型です。
ビット列型	BYTE	8 ビット	ビットの列で値を表現したデータ型です。
	WORD	16 ビット	
	DWORD	32 ビット	
	LWORD	64 ビット	
整数型	SINT	8 ビット	値が整数値のデータ型です。
	INT	16 ビット	
	DINT	32 ビット	
	LINT	64 ビット	
	USINT	8 ビット	
	UINT	16 ビット	
	UDINT	32 ビット	
	ULINT	64 ビット	
実数型	REAL、	32 ビット	値が実数値のデータ型です。
	LREAL	64 ビット	
持続時間型	TIME	64 ビット	値が時間（日・時・分・秒・ミリ秒）のデータ型です。
時刻型	TIME_OF_DAY	64 ビット	値が時刻（時・分・秒）のデータ型です。
日付型	DATE	64 ビット	値が日付（年・月・日）のデータ型です。
日付時刻型	DATE_AND_TIME	64 ビット	値が日付時刻（年・月・日・時・分・秒・ミリ秒）のデータ型です。
文字列型	STRING	0 ～ 1986 バイト	値が文字列のデータ型です。

2-3　ファンクションブロックの活用で再利用性アップ

従来の問題点

1 つのプロジェクトで、同じアルゴリズムを繰り返し使用する場合、同じパターンの回路をコピー & ペーストし、アドレスなどを変更することが多くありました。
コピー & ペーストでアルゴリズムを流用する場合、以下の問題点があります。

- 流用手順が複雑
  アドレスの変更だけでなく、タイマ番号やカウンタ番号の変更等も必要となり、ミスが発生しやすい。
- プログラム修正工数が多い
  コピー元のアルゴリズムに誤りが見つかった場合、コピー先の全ての回路の修正が必要となる。
- アドレス重複に注意が必要
  コピー元や他の回路とアドレスが重複しないように、コピー先のアドレスを注意しながら変更しなければならない。
- プログラムの可読性が低い
  プログラムが巻物のように長くなり、各機能の区切りや、機能の入力と出力がすぐにわからないため、プログラム全体の処理を把握しにくい。

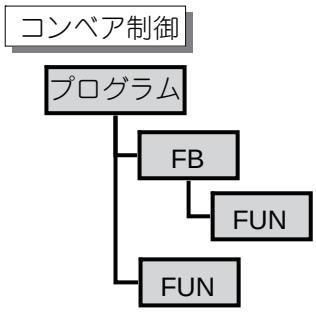


NJ で解決！

NJ シリーズでは、IEC 61131-3 で定義された、プログラムを構造化するしくみであるファンクションとファンクションブロックをサポートしています。
IEC 61131-3 では、POU（Program Organization Unit）と呼ばれる単位でアルゴリズムを作成します。POU には、以下の 3 種類があります。

POU の構成要素	説　明
プログラム	メインルーチンに相当する、アルゴリズムを記載する主となる要素です。 アルゴリズムに、すべての命令、ファンクション、ファンクションブロックを記述することができます。
ファンクションブロック（FB）	プログラムまたは別のファンクションブロックから呼び出されることによって起動します。 プログラムの FB を呼び出す箇所ごとに、FB 内の変数を格納しておく「インスタンス」を配置し、FB を呼び出すごとに変数の値を保持します。そのため、プログラム部品内で状態を保持する必要がある（カウンタ、立上り / 立下り微分など）場合は、FB を使用します。
ファンクション（FUN）	プログラム、ファンクションブロック、またはファンクションから呼び出されることによって起動します。 ファンクション内の変数を保持せず、同一入力に対し、常に同じ値を出力します。そのため、プログラム部品内で状態を保持する必要が無い（四則演算、三角関数など）場合は、ファンクションを使用すると、インスタンスを宣言せずにプログラムから呼び出すことができます。

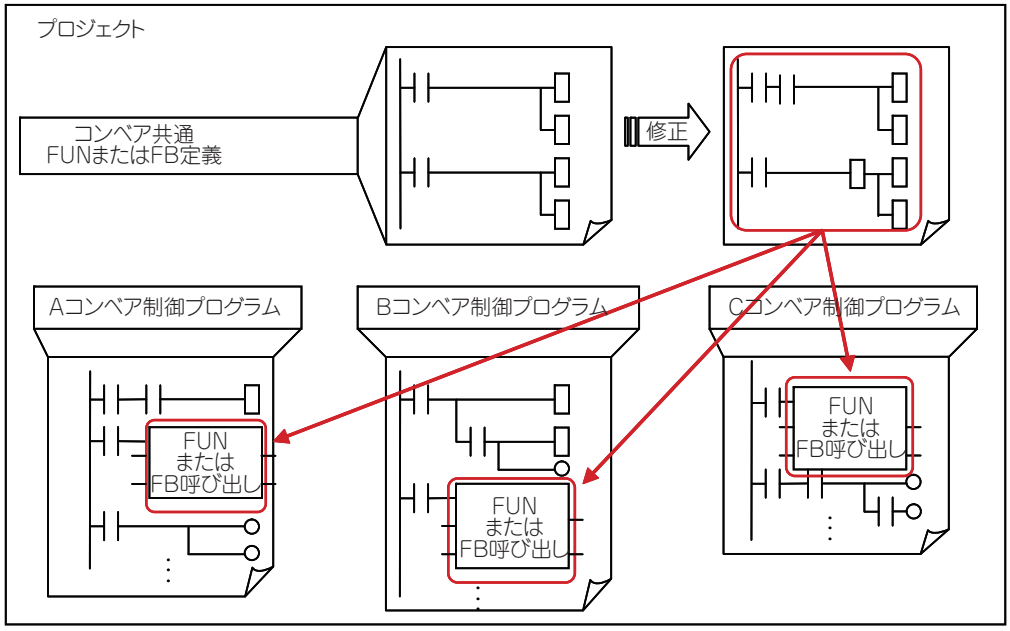
ある POU から他の POU を呼び出すことによって、プログラムを階層的に構成することができます。



プログラム内で繰り返し使用する定型的なアルゴリズムを、ファンクションブロックまたはファンクションとしてひとまとめにすることで、アルゴリズムを再利用することができます。

ファンクションやファンクションブロックを用いると、従来の問題点が解決します。

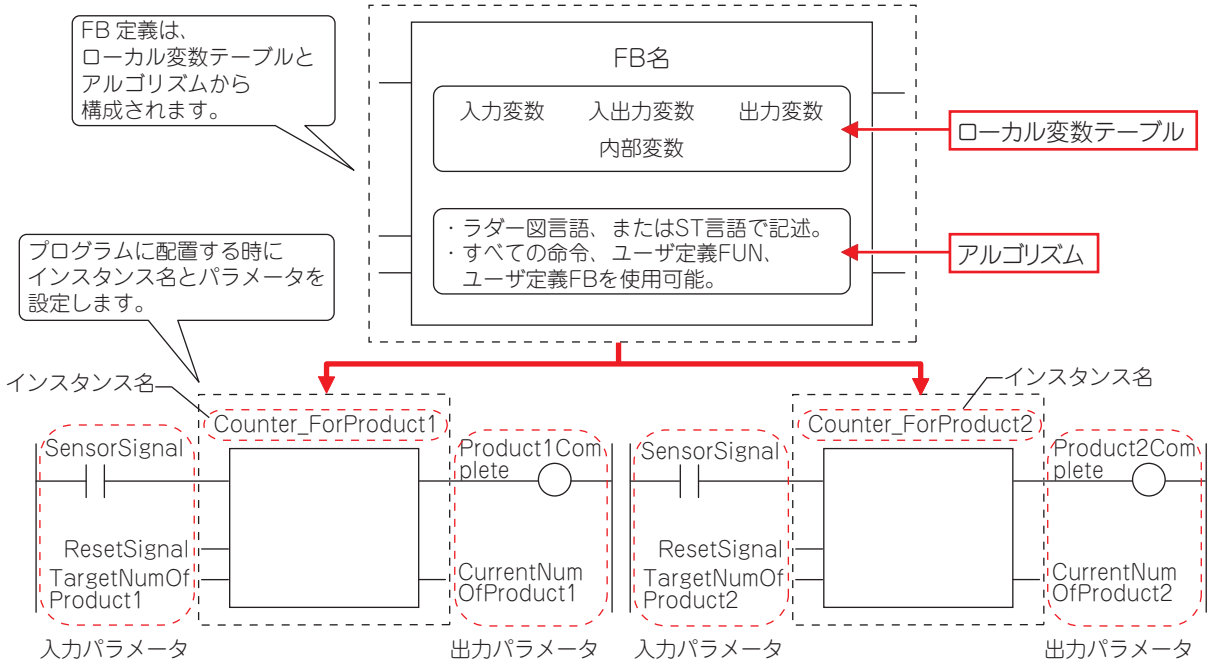
- 流用手順が簡単
  メモリアドレス、タイマ番号、カウンタ番号等の変更が不要で、ミスが発生しない。
- プログラム修正工数が少ない
  アルゴリズムに誤りが見つかった場合、修正箇所はファンクション定義またはファンクションブロック定義の 1 箇所で済みます。
- アドレスの変更が不要
  ファンクションやファンクションブロック内のアドレスは、インスタンスごとに独立しているため、アドレスの変更が不要です。アドレスが重複することはありません。
- プログラムの可読性が向上
  プログラム中の定型的な処理が部品化されているので、プログラムの階層構造や、処理内容を把握しやすくなります。短時間でプログラムの処理を把握できれば、保守の時間も短縮できます。
- ファンクションやファンクションブロック内アルゴリズムの理解が不要
  一度ファンクションやファンクションブロックを作成すれば、あとは、入力と出力の機能を知っているだけで流用できます。また、稼動実績のあるファンクションやファンクションブロックを流用することにより、総デバッグ工数を削減できます。



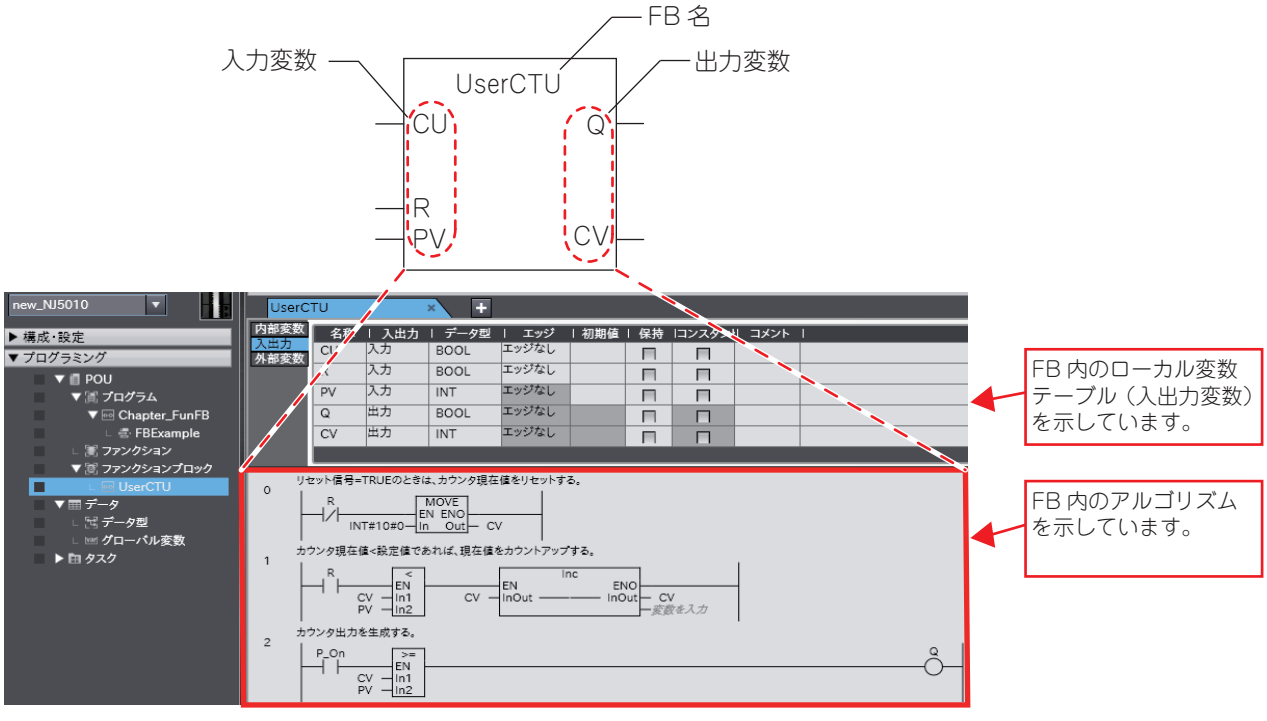
解説

- ファンクションブロック定義
  ファンクションブロック定義（以下 FB 定義）とは、入出力インタフェースおよびアルゴリズムを定義したものです。入出力インタフェースは、ローカル変数として定義します。

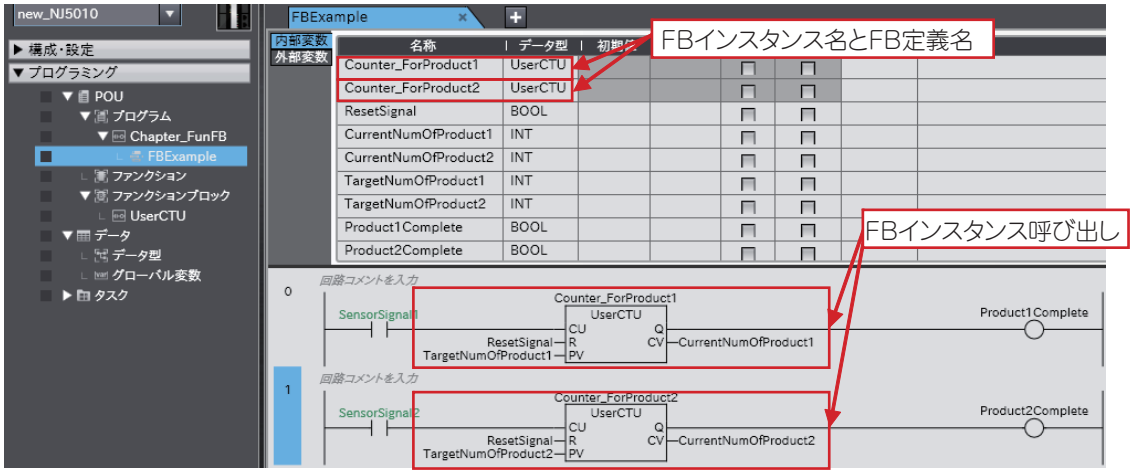
- FB インスタンス
  FB 定義をプログラムで使用する場合の呼び出し命令です。
  FBインスタンスをプログラムに配置した時点で、FB定義内で使用する変数のメモリが確保されます。
  使用する FB 内の変数は、FB インスタンスごとに別々のメモリに割り付けられて実行されるため、1 つの FB 定義を複数のインスタンスで呼び出しても問題は発生しません。



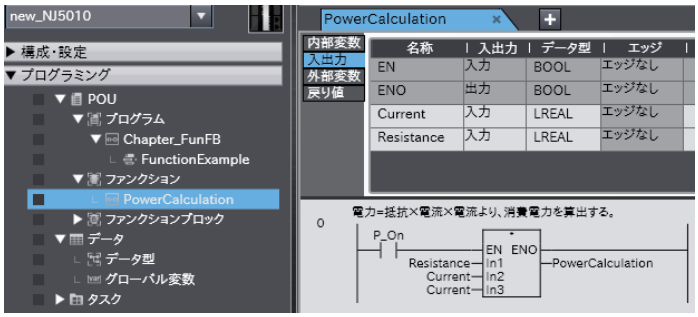
- FB 定義の記述例
  FB 定義内のアルゴリズムは、ラダー図言語または ST 言語を使用して、記述します。
  以下は、ラダー図言語での記述例です。



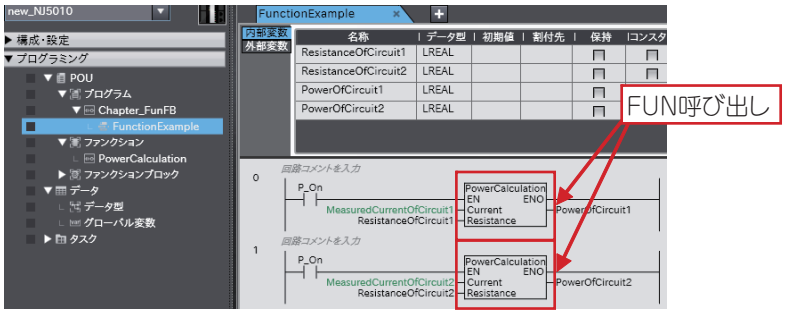
- FB インスタンスの呼び出し例
  FB インスタンスは、プログラムまたは別の FB から呼び出せます。
  以下は、ラダー図プログラムから呼び出した場合の記述例です。



- FUN 定義の記述例
  FUN 定義内のアルゴリズムは、ラダー図言語または ST 言語を使用して記述します。
  以下は、ラダー図言語での記述例です。



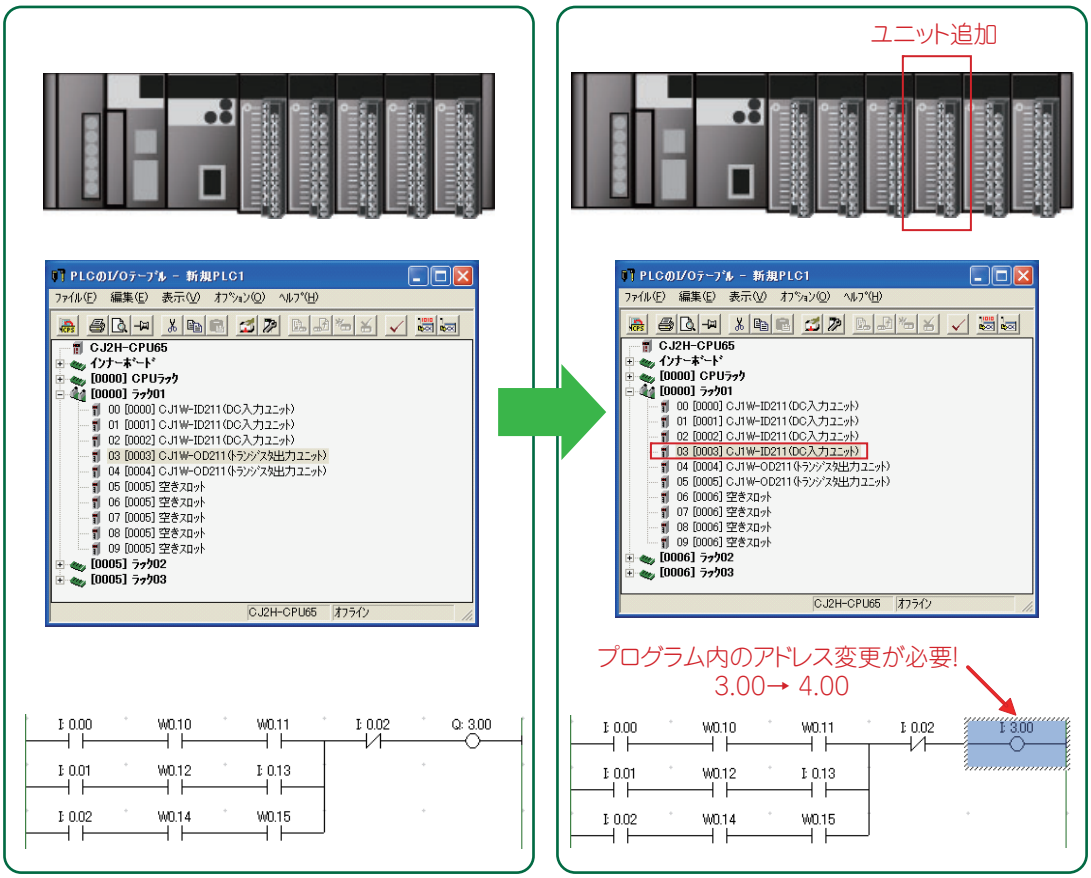
- FUN の呼び出し例
  FB インスタンスは、プログラム、FB または別の FUN から呼び出せます。
  以下は、ラダー図プログラムから呼び出した場合の記述例です。



2-4　機器構成を変更してもプログラム変更が不要

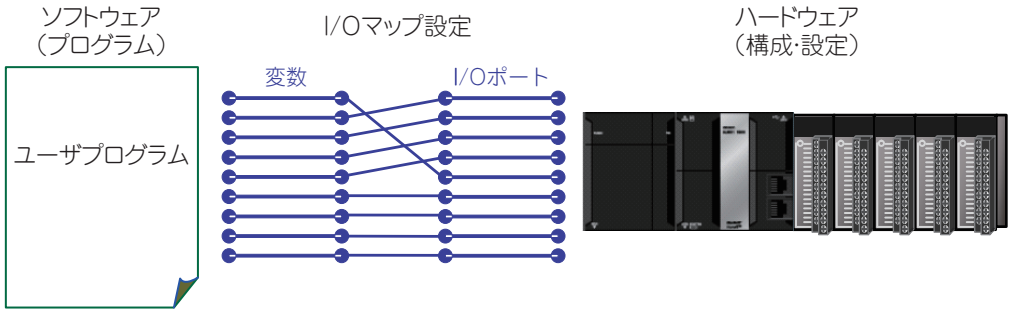
従来の問題点

従来の PLC システムでは、ユニットが割り付けられるメモリ空間上のアドレスを直接指定してプログラミングしていました。
ユニット構成の変更や、異なる装置にプログラムを流用する場合、アドレスの変更にともない、手作業でアドレスを置換したり、既存プログラムのアドレスを変更したりすることがミスの原因となっていました。
たとえば、既設装置に I/O ユニットを追加するような場合でも、アドレスの変更がプログラム全体に影響することもありました。

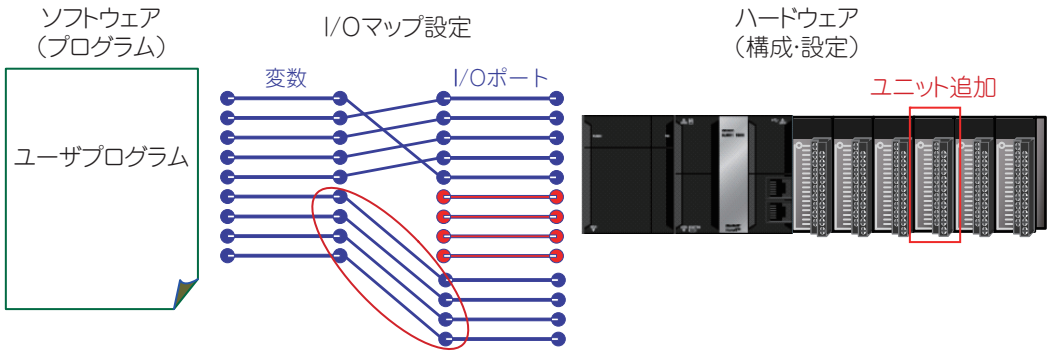


NJ で解決！

NJ シリーズでは、Sysmac Studio 上でユニット構成を作成することで、I/O ポートが自動的に生成されます。
I/O ポートとは、CPU ユニットがデバイス (スレーブ / ユニット) とデータをやりとりするための、論理的なインターフェースです。
NJ シリーズでは、I/O ポートに対して「デバイス変数」と呼ぶ変数を割り付け、プログラムからデバイス変数にアクセスします。
そのため、ハードウェア構成の決定を待たずに、独立してソフトウェアを設計できます。

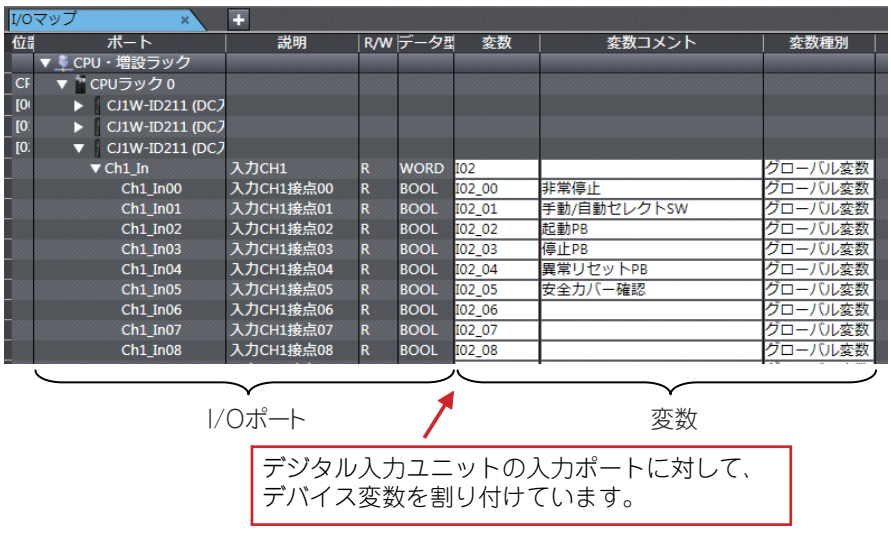


また、ユニット機種の変更があっても、システムで定義される I/O ポートが変更されるため、I/O マップ設定で I/O ポートと変数を再割り付けするだけです。プログラムの変更は、必要ありません。
ユニット追加の場合でも、既に I/O ポートに割り付けた変数や、既存のプログラムを修正する必要はありません。
そのため、手作業による修正が少なくなることでミスが低減し、確認工数や修正工数が削減され、開発コストの削減や、開発期間の短縮にもつながります。



変数やプログラムを修正する必要はありません。

● Sysmac Studio の I/O マップによる、I/O ポートへの変数の割り付け例



I/O マップ

位置	ポート	説明	R/W	データ型	変数	変数コメント	変数種別
	CPU・増設ラック						
CP	CPUラック 0						
[0	CJ1W-ID211 (DC入						
[0	CJ1W-ID211 (DC入						
[0	CJ1W-ID211 (DC入						
	Ch1_In	入力CH1	R	WORD	I02		グローバル変数
	Ch1_In00	入力CH1接点00	R	BOOL	I02_00	非常停止	グローバル変数
	Ch1_In01	入力CH1接点01	R	BOOL	I02_01	手動/自動セレクトSW	グローバル変数
	Ch1_In02	入力CH1接点02	R	BOOL	I02_02	起動PB	グローバル変数
	Ch1_In03	入力CH1接点03	R	BOOL	I02_03	停止PB	グローバル変数
	Ch1_In04	入力CH1接点04	R	BOOL	I02_04	異常リセットPB	グローバル変数
	Ch1_In05	入力CH1接点05	R	BOOL	I02_05	安全カバー確認	グローバル変数
	Ch1_In06	入力CH1接点06	R	BOOL	I02_06		グローバル変数
	Ch1_In07	入力CH1接点07	R	BOOL	I02_07		グローバル変数
	Ch1_In08	入力CH1接点08	R	BOOL	I02_08		グローバル変数

I/Oポート　　変数

デジタル入力ユニットの入力ポートに対して、
デバイス変数を割り付けています。

2-5　軸変数で簡単モーションプログラミング

従来の問題点

PLC プログラミングの中でも、モーションプログラミングは特に困難です。
従来の PLC では、モーションプログラムを作成するためには、膨大なアドレスマップから、目的の制御に必要な各種データのアドレスを探す必要があります。
アドレスによるモーションプログラムには以下の問題点があります。

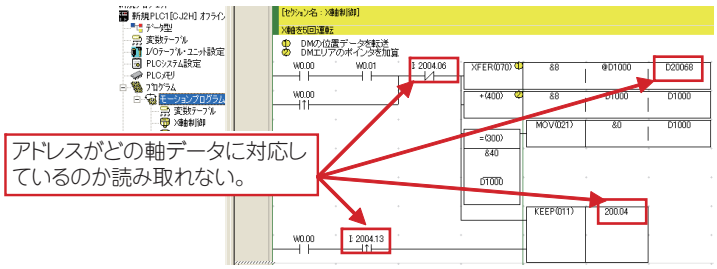
- プログラム作成時
  アドレスの番号と各軸データとの意味的な関連が無く、軸データが多いため、アドレス指定ミスを引き起こしやすい。
- デバッグ時や保守時
  回路の可読性が低く、プログラムの理解に時間を要する。
- プログラム流用時
  X 軸→ Y 軸のように軸を変更すると、アドレスマップも変わるため、流用に手間がかかる。

● 軸データのアドレス割り付け例

NC1□3：I＝m+32＝D20000＋100×号機No.+32
NC2□3：I＝m+60＝D20000＋100×号機No.+60
NC4□3：I＝m+116＝D20000＋100×号機No.+116

入出力	チャネル NC1□3 X軸	NC2□3 X軸	NC2□3 Y軸	NC4□3 X軸	NC4□3 Y軸	NC4□3 Z軸	NC4□3 U軸	分類	名称	機能	参照
PC→NC（出力）	I	I		I				データ転送用運転データ	書き込みチャネル数	PLCからNCユニットに書き込むデータのチャネル数を指定します。	5章 データの転送と保存
	I+1	I+1		I+1					書き込み元エリア	PLCからNCユニットに書き込むデータを用意するエリアを指定します。	
	I+2	I+2		I+2					書き込み元チャネル	PLCからNCユニットに書き込むデータの先頭チャネルを指定します。	
	I+3	I+3		I+3					書き込み先アドレス	データを書き込むNCユニットのアドレスを指定します。	
	I+4	I+4		I+4					読み出しチャネル数	NCユニットからPLCに読み出すデータのチャネル数を指定します。	
	I+5	I+5		I+5					読み出し元アドレス	データを読み出すNCユニットのアドレスを指定します。	
	I+6	I+6		I+6					読み出し先エリア	NCユニットからPLCに読み出したデータを出力するエリアを指定します。	
	I+7	I+7		I+7					読み出し先チャネル	NCユニットからPLCに読み出したデータを出力する先頭チャネルを指定します。	
	I+8	I+8	I+20	I+8	I+20	I+32	I+44	直接運転用運転データ	位置指令（下位）	直接運転、現在位置プリセットを行うときの位置を指定します。	7章 直接運転 6−6 現在位置プリセット
	I+9	I+9	I+21	I+9	I+21	I+33	I+45		位置指令（上位）		
	I+10	I+10	I+22	I+10	I+22	I+34	I+46		速度指令（下位）	直接運転、JOG、原点復帰を行うときの目標速度を指定します。	7章 直接運転 6−7 原点復帰 9−1 JOG動作
	I+11	I+11	I+23	I+11	I+23	I+35	I+47		速度指令（上位）		
	I+12	I+12	I+24	I+12	I+24	I+36	I+48		加速時間（下位）	直接運転、JOG、原点復帰を行うときの加速・減速時間を指定します。	
	I+13	I+13	I+25	I+13	I+25	I+37	I+49		加速時間（上位）		
	I+14	I+14	I+26	I+14	I+26	I+38	I+50		減速時間（下位）		
	I+15	I+15	I+27	I+15	I+27	I+39	I+51		減速時間（上位）		
	I+16	I+16	I+28	I+16	I+28	I+40	I+52	メモリ運転用	シーケンスNo.	メモリ運転の開始シーケンスNo.を指定します。	8章 メモリ運転
	I+17	I+17	I+29	I+17	I+29	I+41	I+53	特殊機能用	オーバーライド	オーバーライドの比率を指定します。	9−6 オーバーライド
	I+18	I+18	I+30	I+18	I+30	I+42	I+54	運転データ	ティーチングアドレス	ティーチングアドレスNo.を指定します。	9−2 ティーチング
	I+19	I+19	I+31	I+19	I+31	I+43	I+55		未使用	――	――
NC→PC（入力）	I+20	I+32	I+36	I+56	I+60	I+64	I+68	NCユニットステータス	現在位置（下位）	NCユニットで制御する各軸の現在位置を返します。−2,147,483,647～2,147,483,647パルスの位置を符号付き32ビットHex（16進）データの上位、下位を2チャネルで表します。現在位置：80000001～7FFFFFFF Hex（−2,147,483,647～2,147,483,647）	
	I+21	I+33	I+37	I+57	I+61	I+65	I+69		現在位置（上位）		
	I+22	I+34	I+38	I+58	I+62	I+66	I+70		シーケンスNo.	メモリ運転時、実行中のシーケンスNo.を返します。	8章 メモリ運転
	I+23	I+35	I+39	I+59	I+63	I+67	I+71		出力コード（注1）	メモリ運転時の出力コードを返します。	

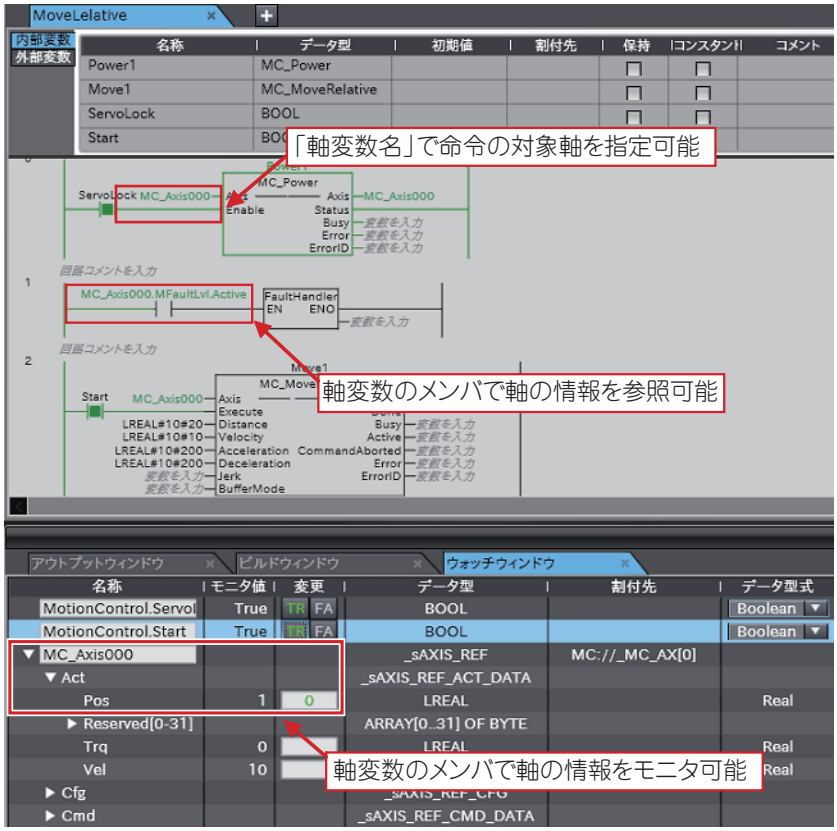
● プログラム例



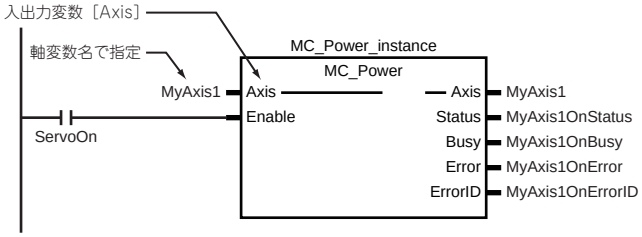
NJ で解決！

NJ では、各軸の現在位置、異常情報などをまとめて扱う「軸変数」というしくみをサポートしています。

- 1 つの軸を、1 つの軸変数（構造体変数）として扱うため、各軸の現在位置、異常情報を参照するとき、「軸変数名．メンバ名」で参照できます。アドレスマップから各種データのアドレスを探す必要がなくなります。
- モーション制御命令に軸変数を入力することで、制御対象の軸を指定できるため、簡単にプログラミングができます。



ユーザプログラムでは、モーション制御命令の入出力変数「Axis」に、軸変数名を指定します。

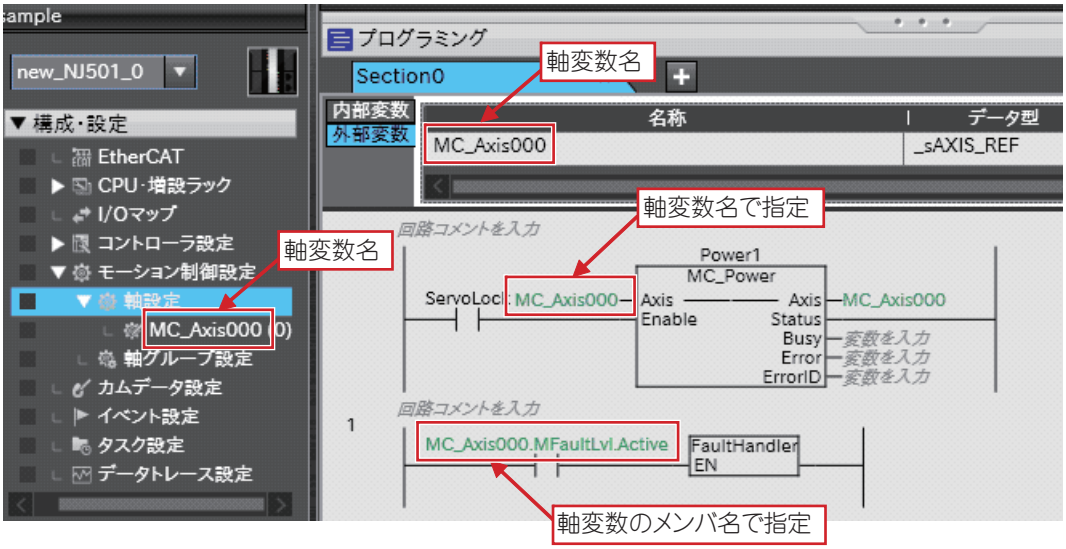


- プログラム作成
  軸の各種データの参照は、「軸変数名．メンバ名」で指定できるため、指定ミスを引き起こしにくい。
- デバッグ・保守時
  「軸変数名．メンバ名」という意味の名称を用いるため、回路の可読性が高くなり、プログラムの読解時間を短縮できる。
- プログラム流用時
  指定データは同じで、軸のみを変更する場合は、「軸変数名」を変更するだけで済むため、流用が簡単にできる。

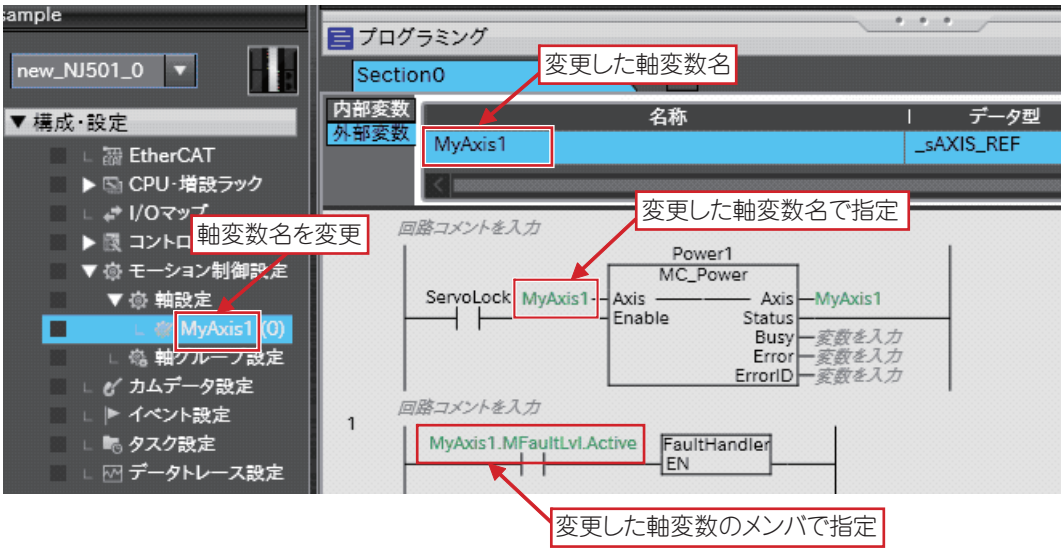
このように、軸変数を使用すると、可読性に優れ、軸数の増減に強いプログラムを作成することが可能です。

解説

- 軸変数
  軸の名称と同名の構造体型の変数です。SysmacStudio で軸を追加すると、自動で軸変数が作成されます。
  モーション制御命令を使用し、軸を制御する時や、軸の状態をモニタする時に使用します。



- 軸変数名の変更
  軸変数名は、わかりやすい名前に変更できます。下記は「MyAxis1」に変更した例です。



2-6　構造体型の活用でレシピ処理が簡単

従来の問題点

物理アドレスのプログラムで「レシピデータ」(*) を処理する場合、レシピデータの先頭物理アドレスを基準とするインデックスレジスタにより、製品 No. など複数の情報のオフセットを計算してアクセスする方法が一般的でした。

* ここでの「レシピデータ」とは、1 種類の製品が持つ複数の情報（製品 No.、長さ、幅、高さなど）の組み合わせの意味です。

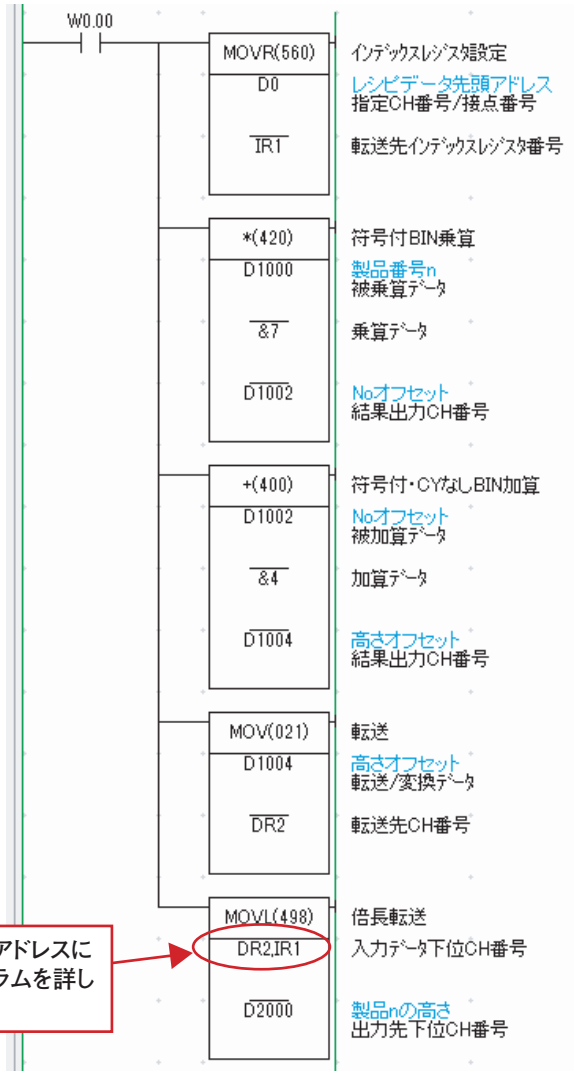
レシピデータ

物理アドレス	コメント
E0-E1	製品0の長さ
E2-E3	製品0の幅
E4-E5	製品0の高さ
E6	製品0のLotNo
E7-E8	製品1の長さ
E9-E10	製品1の幅
E11-E12	製品1の高さ
E13	製品1のLotNo
:	:

製品nの高さを抽出

物理アドレス	コメント
D2000	製品nの長さ

製品No**の”高さ”の格納先は、[製品 No**]×[製品1個のオフセット(7WORD)]+[製品の高さまでのオフセット(4WORD)]
この計算はプログラム作成が難しく、検証にも時間を要する。



インデックスレジスタがどのアドレスにアクセスしているか、プログラムを詳しく把握しないと分からない。

インデックスレジスタを使用したプログラムでは、下記のような問題があります。

- プログラムからどの物理アドレスにアクセスしているか分かりにくく、デバッグが困難
- インデックスの指定を誤ると、レシピデータの範囲外の物理アドレスにアクセスし、値を書き換えてしまう

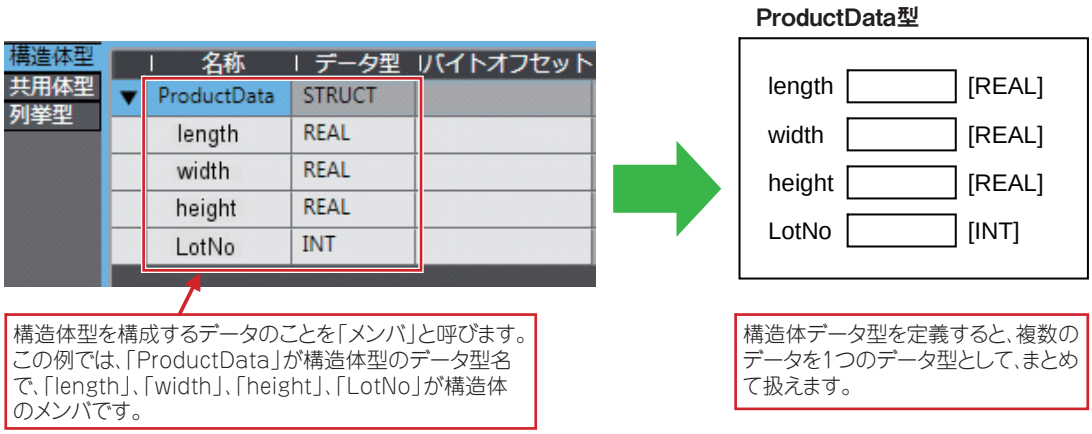
また、製品の情報（レシピデータのフィールド）の並び替えや追加が発生した場合、下記のようにプログラムの変更が容易でない問題もあります。

- インデックスレジスタを使用したプログラムは、解読が困難
- プログラム変更による影響の見極めが難しく、修正やデバッグに多くの時間が必要

NJ で解決！

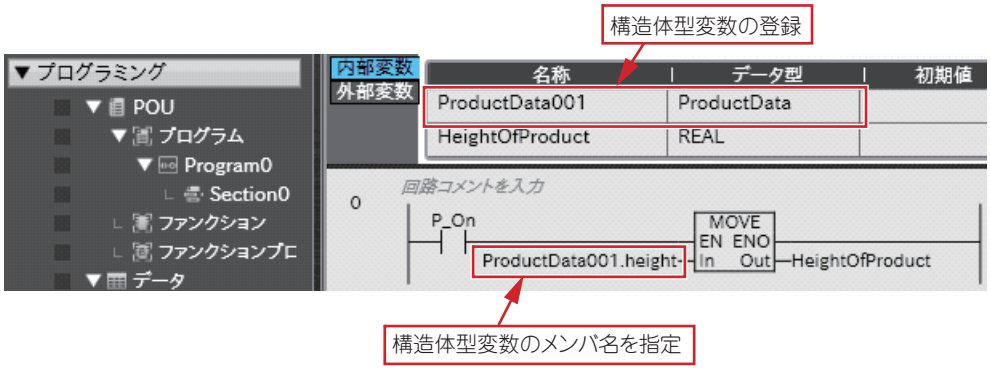
NJ シリーズのプログラミングは、構造体型をサポートしています。構造体型とは、複数の異なるデータ型を階層的にひとつにまとめたデータ型のことです。
レシピデータの例では、以下のように型定義、変数宣言をすると、＂構造体変数名．メンバ名＂で該当するレシピデータにアクセスでき、面倒なアドレス計算は一切不要です。そのため、製品レシピのフィールドの並びを変更してもプロジェクトの変更は型定義の 1 箇所で済みます。また、新たなフィールドを追加した場合、既存のプログラムは変更不要です。

● データ型定義の例

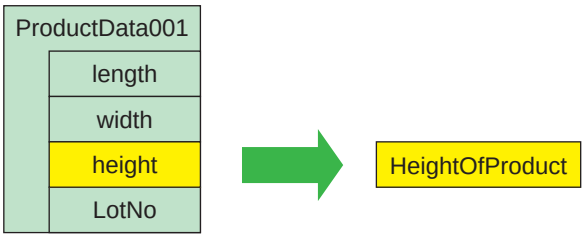


● プログラム例

構造体を使用する場合は、変数テーブルに登録して使用します。
構造体のメンバを指定する場合は、下記のように、［構造体変数名］．［メンバ名］を指定します。



このプログラム例では、構造体変数名「ProductData001」のメンバ名「height」の値を変数「HeightOfProduct」に転送します。



構造体型を配列として定義をすると、下記のようなデータ領域が確保されます。

Recipe	length	width	height	LotNo
Recipe[0]				
Recipe[1]				
Recipe[2]				
:				
Recipe[99]				

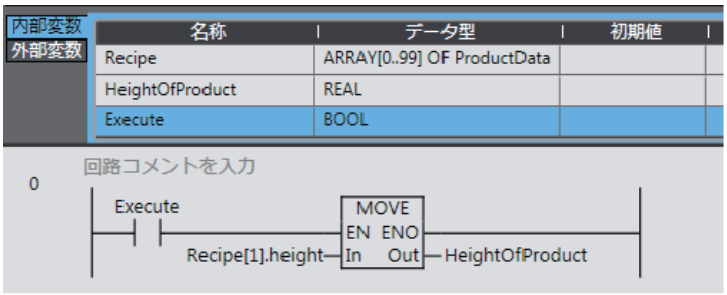
データ領域

- レシピデータから 1 要素をコピーする場合
  前ページのプログラムは、下記のように表現されます。
  - プログラムの考え方

Recipe	long	wide	height	LotNo
Recipe[0]				
Recipe[1]				
Recipe[2]				
:				
Recipe[99]				

コピー → HeightOfProduct

  - プログラム



- レシピデータをすべてコピーする場合
  レシピデータをすべてコピーするには、下記のようにします。
  - プログラムの考え方

Recipe[0] ～ [99]

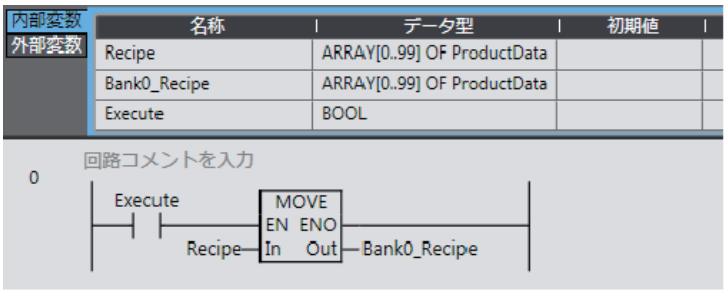
Recipe	length	LotNo
Recipe[0]		
Recipe[1]		
Recipe[2]		
:		
Recipe[99]		

コピー →

Bank0_Recipe[0] ～ [99]

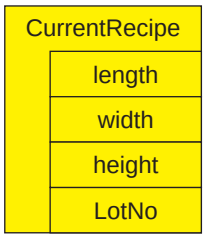
Recipe	length	LotNo
Recipe[0]		
Recipe[1]		
Recipe[2]		
:		
Recipe[99]		

  - プログラム

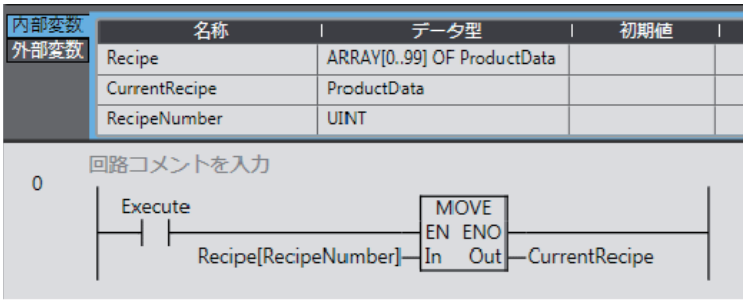


- レシピデータの 1 レコードをコピーする場合
  レシピデータの 1 レコード分をコピーするには、下記のようにします。
  - プログラムの考え方

Recipe	length	width	height	LotNo
Recipe[0]				
Recipe[1]				
Recipe[2]				
:				
Recipe[99]				



  - NJ シリーズのプログラム



解説

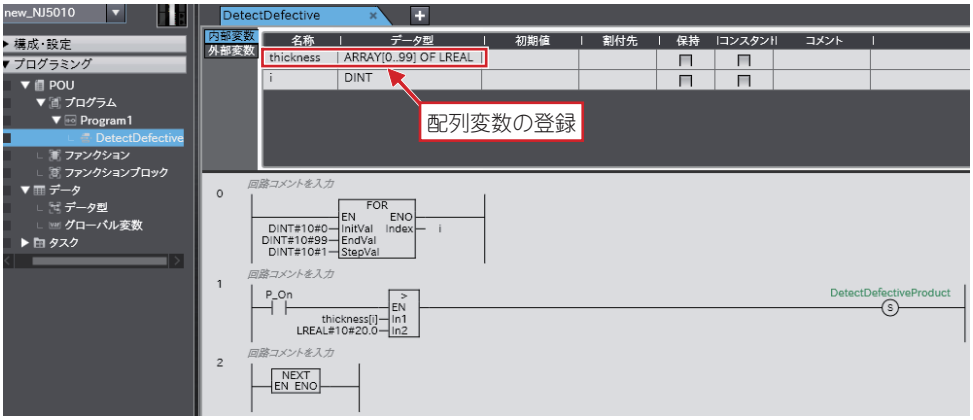
- 配列

  同じデータ型の要素を複数個まとめて、ひとつの変数にしたものです。配列を構成する各データのことを「要素」と呼びます。配列指定の要素は配列変数名に、先頭から数えた添え字（要素番号）を付けて記述します。

  - 例）基板の 100 ポイントの厚みを、ひとつの配列変数として扱います。

    配列変数名　　：thickness
    配列要素の開始番号　：0
    配列要素の終了番号　：99
    変数のデータ型　　：LREAL

要　素	記述方法
ポイント 0 の厚み	thickness[0]
ポイント 1 の厚み	thickness[1]
ポイント 2 の厚み	thickness[2]
ポイント 3 の厚み	thickness[3]
:	:
ポイント 99 の厚み	thickness[99]



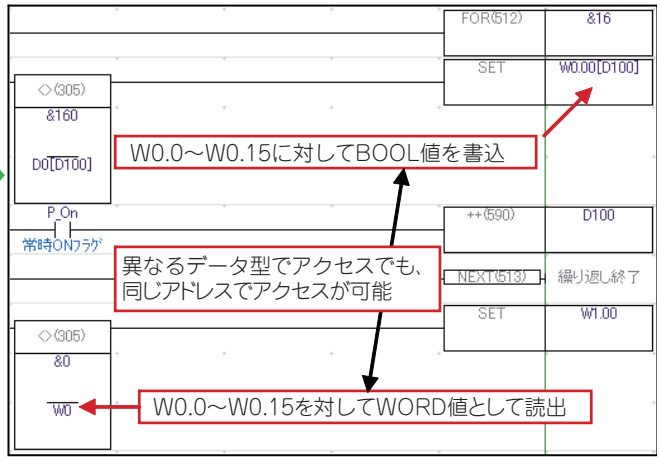
2-7　共用体型で同じデータを異なるデータ型としてアクセス

従来の問題点

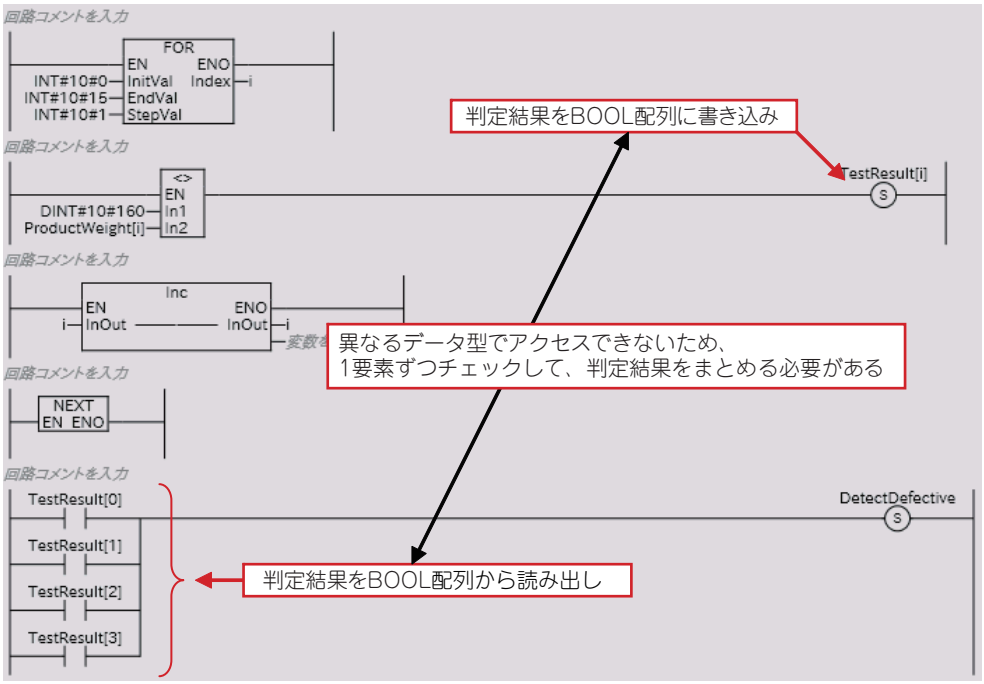
プログラムから、同一データに対して複数の異なるデータ型でアクセスしたい場合があります。
たとえば、16 個の製品検査データを規格値と比較し、合否結果を BOOL 型に格納する場合、製品全体での不具合品の有無は、BOOL 型を 16 個まとめて WORD 型でアクセスし、0 かどうかを調べるのが最も効率の良い方法です。

製品番号	製品重さ	規格値	検査 NG
0	160	160	FALSE
1	153	160	TRUE
:	:	:	:
15	160	160	FALSE

各製品の検査NGをBOOL型で格納し、
製品全体の検査NGをWORD型で判定する
プログラム例



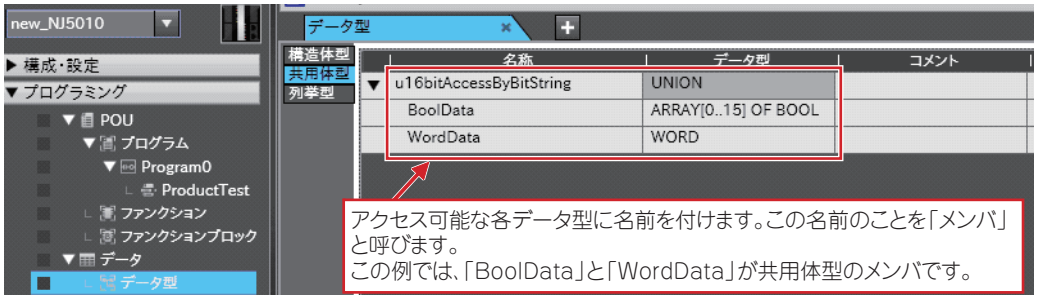
プログラム
変数プログラミングの場合、変数にデータ型が定義されるため、1 つの変数に対して 2 つ以上の異なるデータ型でアクセスすることができません。
たとえば、製品全体の不具合品の有無を調べるために、BOOL 型配列の TRUE/FALSE を 1 要素ずつチェックするなどのプログラムが増加するため、可読性の低下につながる場合がありました。



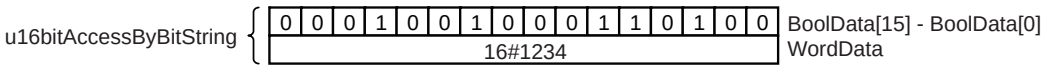
NJ で解決！

NJ シリーズのプログラミングは、共用体型をサポートしています。共用体型とは、同一のデータに対して、複数の異なるデータ型でアクセスできるようにしたデータ型です。

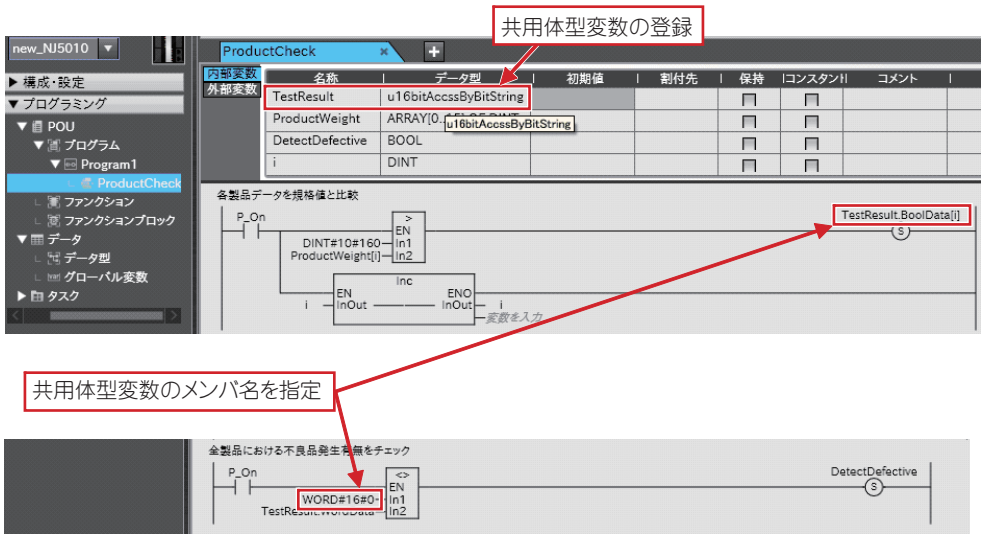
- データ型定義の例



上記のように共用体を型定義すると、この型を持つ変数は以下のように同一データを 1 ビット単位、あるいは 16 ビット単位の、いずれのデータ型でもアクセスすることができます。



- プログラム例



プログラム中では、「共用体変数名．メンバ名」でアクセスすることができます。
この共用体変数を使うと、たとえば、合否結果はそれぞれ製品番号に対応する要素の BOOL 型配列に格納されます。全製品における不良品の有無は WORD 型で参照できるため、簡潔なプログラムの記述が可能です。

解説

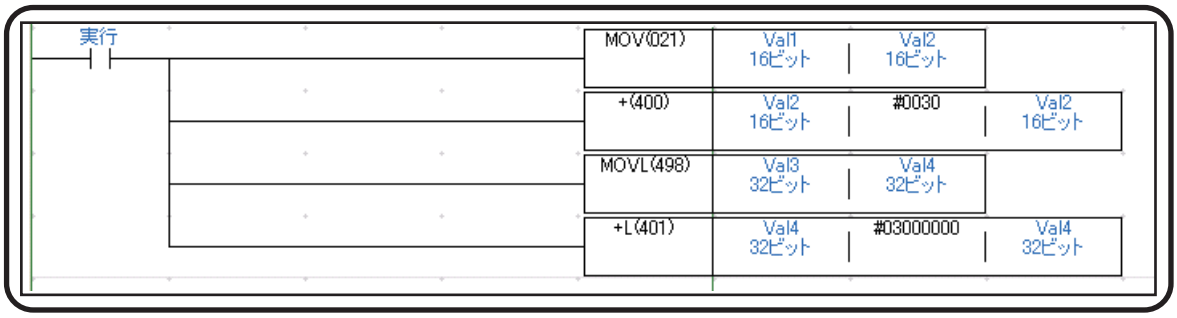
- 共用体型のメンバに指定できるデータ型

分類	データ型	データサイズ（ビット）	値の範囲
ブール型	BOOL	1	FALSE、TRUE
ビット列型	BYTE	8	BYTE#16#00 ～ FF
	WORD	16	WORD#16#0000 ～ FFFF
	DWORD	32	DWORD#16#00000000 ～ FFFFFFFF
	LWORD	64	LWORD#16#0000000000000000 ～ FFFFFFFFFFFFFFFF

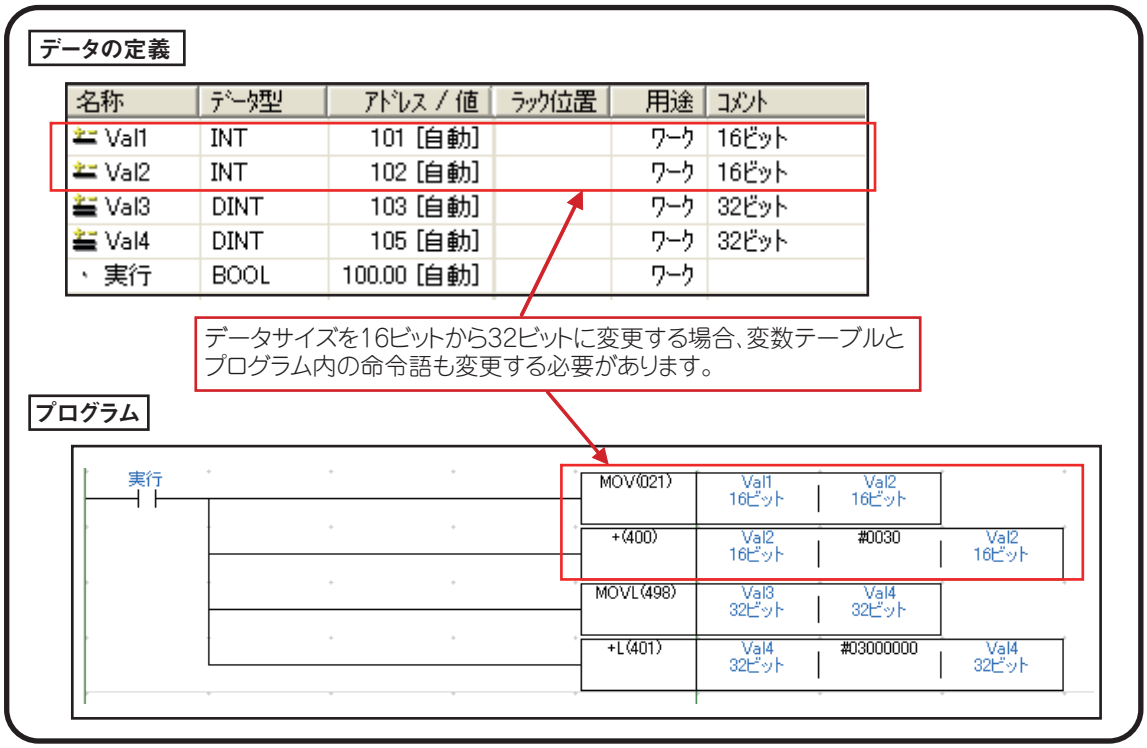
2-8　変数対応の命令語で型を意識せずプログラミング

従来の問題点

従来の PLC では、取り扱うデータのワード長によって複数の命令が用意されています。
たとえば、CJ シリーズでの転送命令は、16 ビットデータの場合は MOV、32 ビットデータの場合は MOVL になります。



このため、設計途中や改造でデータ型を変更しようとした場合、変数の型だけでなく、プログラムのコードも変更する必要がありました。
特に同じ変数を複数箇所で使用している場合、使用している箇所を検索し、命令を置換するため、多くの工数がかかりました。



NJ で解決！

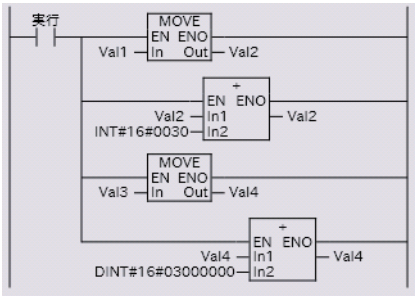
NJ シリーズでは、命令語が変数に対応しているので、取り扱う変数の型によらず、同じ命令を使用できます。たとえば、転送命令は MOVE です。
変数の型の変更は、変数テーブルで型を変えるだけです。プログラムの変更は不要なため、大幅な工数短縮が可能です。

変数テーブル

内部変数 / 外部変数

名称	データ型	初期値	割付先	保持	コンスタント	コメント
実行	BOOL			□	□	
Val1	INT			□	□	16ビット
Val2	INT			□	□	16ビット
Val3	DINT			□	□	32ビット
Val4	DINT			□	□	32ビット

データサイズを16ビットから32ビットに変更する場合、
変数テーブルを変更するだけです。



解説

- 使用可能なデータ型
  各命令で取り扱い可能なデータ型は、コマンドリファレンスで確認できます。
  例）MOVE 命令で使用可能なデータ型

変数

	名称	入 / 出力	内容	有効範囲	単位	初期値
In	転送元	入力	転送元	データ型に従う	−	(*)
Out	転送先	出力	転送先	データ型に従う	−	(*)

* 入力パラメータが省略された場合、初期値は適用されません。ビルド時に異常になります。

	ブール	ビット列				整数								実数		時刻、持続時間、日付、文字列				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	列挙型、配列全体、配列の 1 要素、構造体全体、構造体の 1 メンバも指定可能																			
Out	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	「In」が列挙型、配列の 1 要素、構造体全体、構造体の 1 メンバの場合、「In」と同じデータ型を指定 「In」が配列全体の場合、「In」と同じデータ型、大きさ、添え字の配列を指定																			

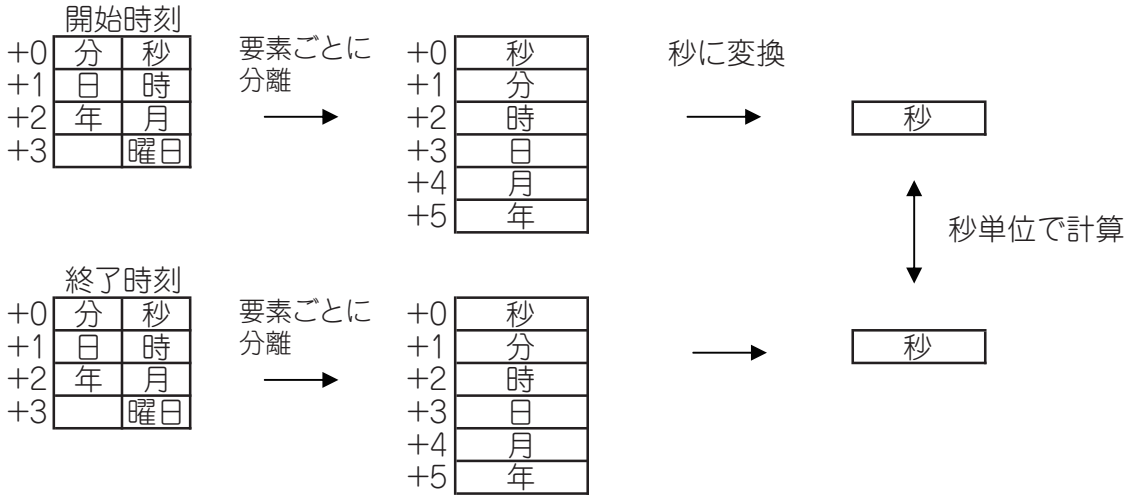
2-9　豊富な時間/時刻演算命令でカレンダ処理が簡単

従来の問題点

製品の生産情報や機械の稼働時間の管理などで、時間や時刻のデータを計算することがあります。
従来の PLC では、時間や時刻の命令数は少なく、扱えるデータ型も限られているため、データ型変換や演算命令との併用により処理が煩雑になり、プログラムの作成が大変でした。

● プログラムの考え方

たとえば、2 つの日付時刻の差を計算する回路の場合、従来の PLC では、要素ごとに分離し、さらに秒に変換して計算するため、プログラム行数が多くなります。



● ラダー図言語によるプログラム

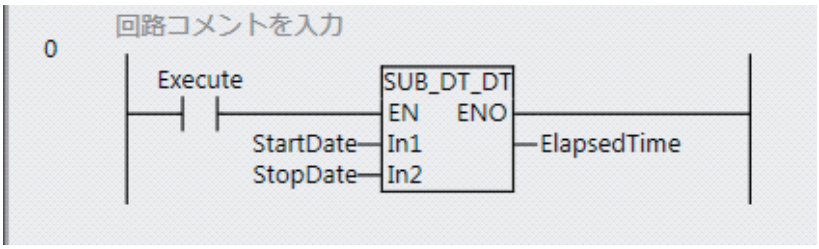
従来のラダー図言語では、約 100 行のプログラムが必要です。

NJ で解決！

NJ シリーズでは、変数で時刻や時間を直接データとして取り扱うことができます。
IEC 61131-3 で定義された時刻や時間のデータ型をすべてサポートしています。さらに、オムロン NJ シリーズ独自の豊富な命令を追加しました。
これにより、作成が大変だった時刻や時間処理のプログラムを大幅に効率化します。

● NJ シリーズのプログラム

ファンクション命令により、2 つの日付時刻の差を計算するプログラムを 1 行で表現することができます。



解説

- NJ シリーズで使用できる時間と時刻に関連するデータ型

分　類	データ型	定　義
持続時間型	TIME	値が時間（日・時・分・秒・ミリ秒）のデータ型です。
時刻型	TIME_OF_DAY	値が時刻（時・分・秒）のデータ型です。
日付型	DATE	値が日付（年・月・日）のデータ型です。
日付時刻型	DATE_AND_TIME	値が日付時刻（年・月・日・時・分・秒・ミリ秒）のデータ型です。

- IEC 61131-3 で定義された時間と時刻に関連する命令

命　令	名　称	説　明
ADD_TIME	時間加算	時間と時間を加算します。
SUB_TIME	時間減算	時間から時間を減算します。
MULTIME	時間乗算	時間を指定乗数で乗算します。
DIVTIME	時間除算	時間を指定除数で除算します。
ADD_TOD_TIME	時刻と時間の加算	時刻に時間を加算します。
SUB_TOD_TIME	時刻と時間の減算	時刻から時間を減算します。
SUB_TOD_TOD	時刻減算	時刻から時刻を減算します。
ADD_DT_TIME	日付時間と時間の加算	日付時刻に時間を加算します。
SUB_DATE_DATE	日付減算	日付から日付を減算します。
SUB_DT_DT	日付時刻減算	日付時刻から日付時刻を減算します。
SUB_DT_TIME	日付時刻と時間の減算	日付時刻から時間を減算します。
CONCAT_DATE_TOD	日付と時刻の結合	日付と時刻を結合します。
DT_TO_TOD	日付時刻からの時刻切出	日付時刻から時刻を切り出します。
DT_TO_DATE	日付時刻からの日付切出	日付時刻から日付を切り出します。

- IEC 61131-3 から拡張した NJ シリーズ独自の時間と時刻に関連する命令

命　令	名　称	説　明
DtToSec	日付時刻→秒変換	日付時刻を、1970年1月1日0時0分0秒からの秒数に変換します。
DateToSec	日付→秒変換	日付を、1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数に変換します。
TodToSec	時刻→秒変換	時刻を、0 時 0 分 0 秒からの秒数に変換します。
TimeToSec	時間→秒変換	時間を、秒数に変換します。
TimeToNanoSec	時間→ナノ秒変換	時間を、ナノ秒数に変換します。
SecToDt	秒→日付時刻変換	1970年1月1日0時0分0秒からの秒数を、日付時刻に変換します。
SecToDate	秒→日付変換	1970 年 1 月 1 日 0 時 0 分 0 秒からの秒数を、日付に変換します。
SecToTod	秒→時刻変換	0 時 0 分 0 秒からの秒数を、時刻に変換します。
SecToTime	秒→時間変換	秒数を、時間に変換します。
NanoSecToTime	ナノ秒→時間変換	ナノ秒数を、時間に変換します。
DtToDateStruct	時刻分解	日付時刻を、「年」「月」「日」「時」「分」「秒」「ナノ秒」に分解します。
DateStructToDt	時刻結合	「年」「月」「日」「時」「分」「秒」「ナノ秒」に分解された日付時刻を、結合します。
ChkLeapYear	うるう年判別	指定された年が、うるう年かどうかを判定します。
GetDaysOfMonth	月の日数取得	指定された月の、日数を取得します。
DaysToMonth	日数→月変換	1 月 1 日からの日数から、その日が何月かを算出します。
GetDayOfWeek	曜日取得	指定された年月日の、曜日を取得します。
GetWeekOfYear	週取得	指定された年月日が、その年の第何週であるかを算出します。
SetTime	時計補正	システム時刻を補正します。
GetTime	時刻取得	現在時刻を読み出します。

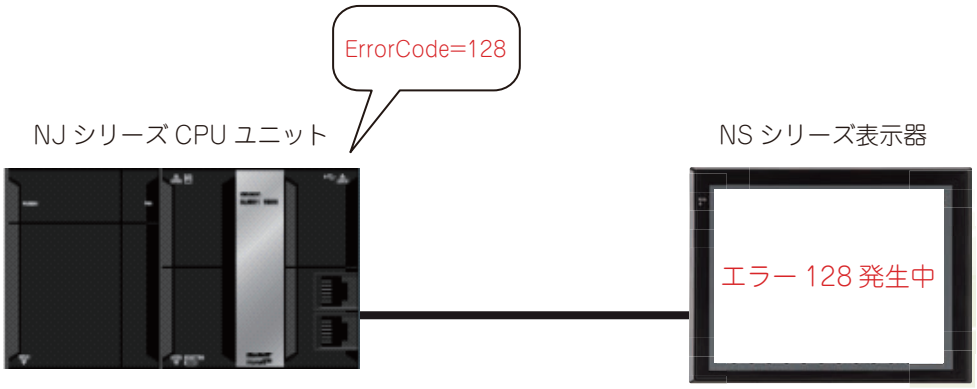
2-10 豊富な文字列命令で文字列処理が簡単

従来の問題点

バーコードリーダなど、外部接続機器からの文字列データ受信や、ID へのデータ書き込みなど、制御プログラムにおいて文字列データを扱うケースが増加しています。
従来の PLC は、メモリの単位がワード単位であることが一般的です。一方で、文字列データはバイト単位で扱うことが一般的なため、データの並べかえなど面倒な処理が必要でした。

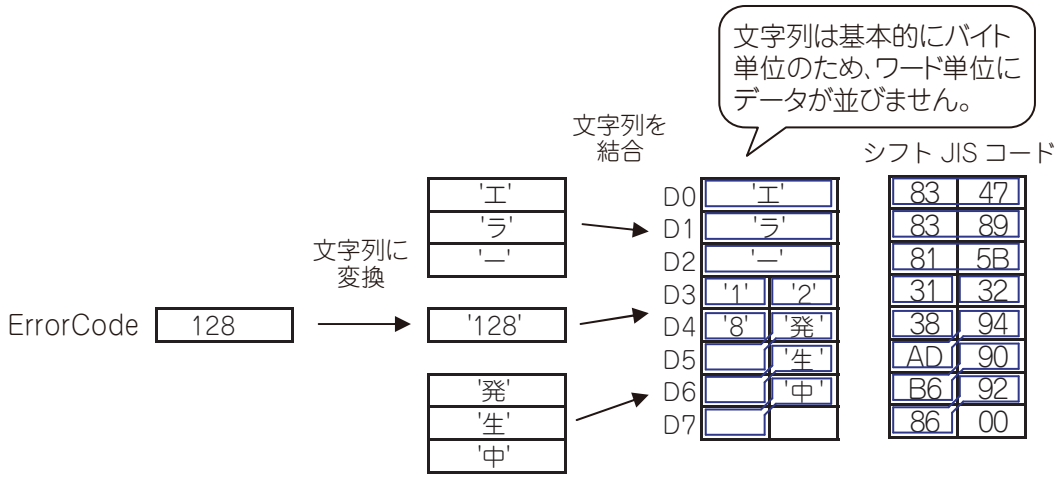
● 表示器へのテキスト表示

たとえば、下記のように表示器にエラーメッセージを表示するだけでも、データが格納されているアドレスの考慮が必要で、14 個の命令が必要になります。



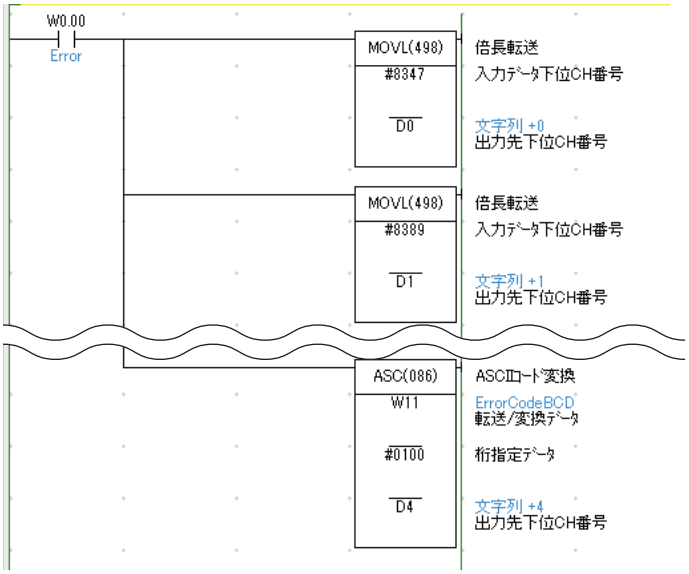
● プログラムの考え方

エラーコードのエラーメッセージを文字列に変換し、変換した文字列をシフト JIS コードで記述して、さらにバイト単位でメモリへ割り付ける必要があります。



● ラダー図言語によるプログラム

従来のラダー図言語では、約 14 行のプログラムが必要です。

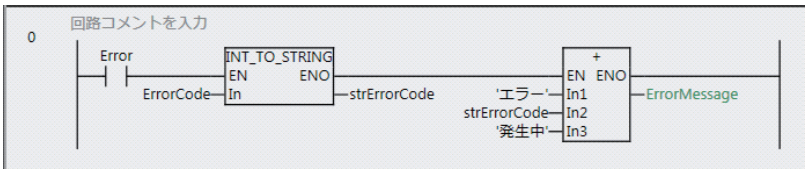


NJ で解決！

NJ シリーズでは、BYTE 型、STRING 型変数をサポートし、メモリ配置への考慮が一切不要となりました。また、豊富な文字列処理命令をサポートし、文字列を扱うことが容易になりました。

● NJ シリーズのプログラム

ファンクション命令により、エラーメッセージを表示するためのプログラムを 1 行で表現することができます。



解説

- IEC 61131-3 で定義された文字列に関連する命令

命　令	名　称	説　明
CONCAT	文字列・連結	2 ～ 5 個の文字列を連結します。
LEFT	文字列・左からの取出	文字列から、指定数の文字列を取り出します。 文字列の左 (先頭) から取り出します。
RIGHT	文字列・右からの取出	文字列から、指定数の文字列を取り出します。 文字列の右 (末尾) から取り出します。
MID	文字列･任意位置からの取出	文字列の任意位置から、指定数の文字列を取り出します。
FIND	文字列・検索	文字列の中の指定文字列の位置を検索します。
LEN	文字列・長さ検出	文字列の文字数を求めます。
REPLACE	文字列・置換	文字列の一部を、別の文字列で置換します。
DELETE	文字列・削除	文字列の一部または全部を削除します。
INSERT	文字列・挿入	文字列の中に、別の文字列を挿入します。

- IEC 61131-3 から拡張した NJ シリーズ独自の文字列に関連する命令

命　令	名　称	説　明
GetByteLen	文字列・バイト数取得	文字列のバイト数をカウントします。
ClearString	文字列・クリア	文字列をクリアします。
ToUCase	文字列・大文字変換	文字列中の半角アルファベットをすべて大文字に変換します。
ToLCase	文字列・小文字変換	文字列中の半角アルファベットをすべて小文字に変換します。
TrimL	文字列・左トリム	文字列中の先頭の空白を削除します。
TrimR	文字列・右トリム	文字列中の末尾の空白を削除します。
AddDelimiter	デリミタ付加文字列書込	構造体全体の値を、デリミタを付加した文字列に変換します。
SubDelimiter	文字列読出デリミタ削除	文字列からデリミタで区切られたデータを読み出し、構造体のメンバの値として格納します。
UTF8ToSJIS	文字コード変換 (UTF-8 → SJIS)	文字コード UTF-8 の文字列を、文字コード SJIS の BYTE 型配列に変換します。
SJISToUTF8	文字コード変換 (SJIS → UTF-8)	文字コード SJIS の BYTE 型配列を、文字コード UTF-8 の文字列に変換します。
NumToDecString	固定長 10 進文字列変換	整数を、固定長 10 進文字列表現に変換します。
NumToHexString	固定長 16 進文字列変換	整数を、固定長 16 進文字列表現に変換します。
HexStringToNum_**	16 進文字列→数値変換グループ	16 進文字列表現を、整数に変換します。
FixNumToString	固定小数点数→文字列変換	符号付き固定小数点数を、10 進文字列表現に変換します。
StringToFixNum	文字列→固定小数点数変換	10 進文字列を、符号付き固定小数点数表現に変換します。
DtToString	日時→文字列変換	日時を、文字列表現に変換します。
DateToString	日付→文字列変換	日付を、文字列表現に変換します。
TodToString	時刻→文字列変換	時刻を、文字列表現に変換します。
StringToAry	文字列→配列変換	文字列を、BYTE 型の配列に変換します。
AryToString	配列→文字列変換	BYTE 型配列を、文字列に変換します。
StringSum	サム値算出	文字列のサム値を算出します。
StringLRC	LRC 値算出 (文字列)	文字列の LRC 値（水平パリティ）を算出します。
StringCRCCCITT	CRC-CCITT 値算出 (文字列)	文字列の CRC-CCITT 値を、XMODEM 方式で算出します。
StringCRC16	CRC-16 値算出 (文字列)	文字列の CRC-16 値を、MODBUS 方式で算出します。

- 文字コードについて

  NJ シリーズの STRING 型は、文字コードとして UTF-8 を採用しています。

  UTF-8 は、ユニコードの一種で、下記特徴を持つコードです。

  - 1 文字は、1 バイトから 4 バイトで表現されます。
  - 英数アルファベットは、1 バイトで、ASCII コードと互換性があります。(00Hex ～ 7FHex)
  - ラテン文字、ギリシャ文字などは、2 バイト。
  - 東アジアの文字（漢字、カナ、中国語、韓国語等）は、基本的に 3 バイト (一部 4 バイト)

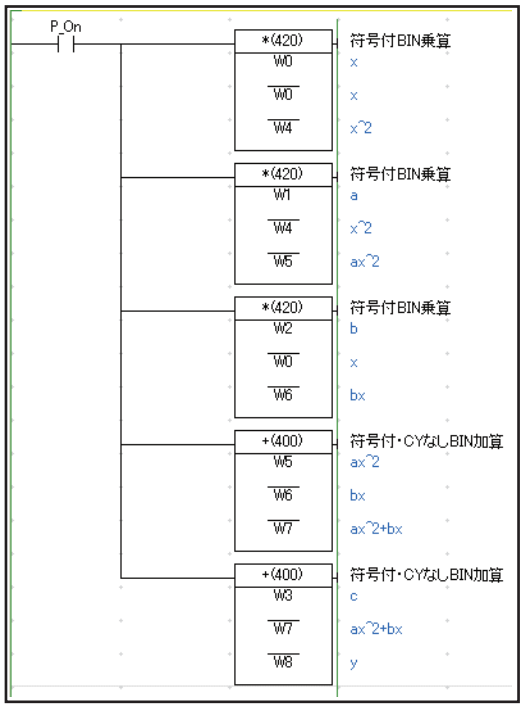
  NJ シリーズでは、日本語、英語、ドイツ語、フランス語、イタリア語、スペイン語、中国語 (簡体字)、中国語 (繁体字)、韓国語の 9 カ国語をサポートしています。

2-11　ST 言語の活用で複雑な演算の記述効率アップ

従来の問題点

制御プログラムの中には、座標の計算やスケーリング演算など、ラダー図言語では、記述しにくい処理があります。
さらに、一画面に表示できる行数が少ないため、プログラムの全体構造を視覚的に把握しづらいという問題があります。

● $y = ax^2 + bx + c$　の計算式の例



NJ で解決！

NJ シリーズでは、IEC 61131-3(JIS B3503) で規定された Structured Text(ST: ストラクチャードテキスト) 言語をサポートしています。ST 言語を使用すると、複雑な数式や条件式などを簡潔に表現することができるため、プログラミングの効率が向上するとともに、可読性も向上します。また、ST はテキストベースの言語のため、コピー＆ペーストにより、他社製ツールとのプログラム移植も容易に行えます。

● $y = ax^2 + bx + c$　の計算式の例

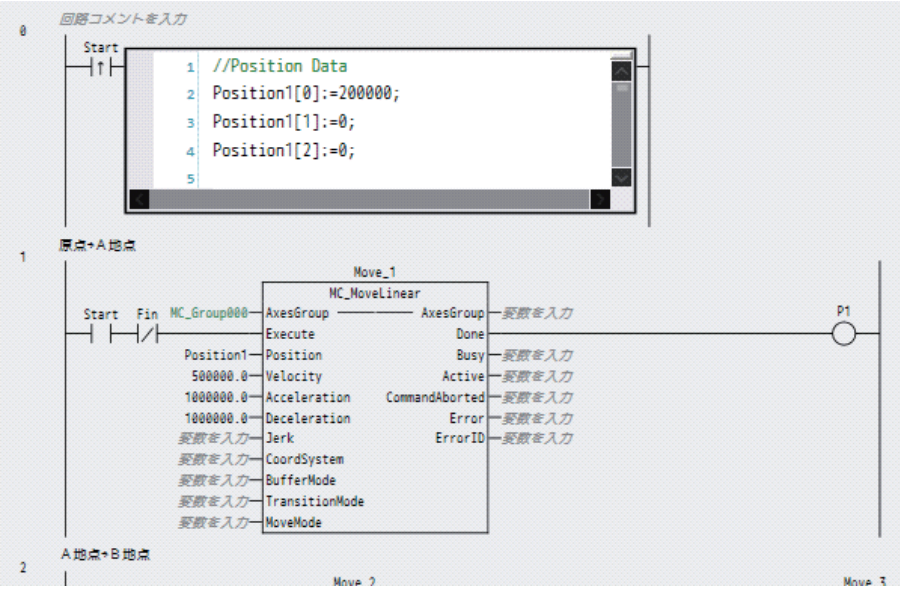
ST言語プログラム

```
1
2   y:=a*x**2 + b*x + c;
3
4
```

同じ処理の表示行数が「1/20」に!
プログラム作成スピードも視認性もUP!

● インライン ST 機能

制御プログラムの内容によっては、ラダー図言語の方が適している場合があります。この場合、インライン ST 機能を使用することで、ラダー図の中に ST 言語を混在して記述することができます。ST 言語とラダー図言語の両方のメリットを生かしたプログラミングが可能となります。

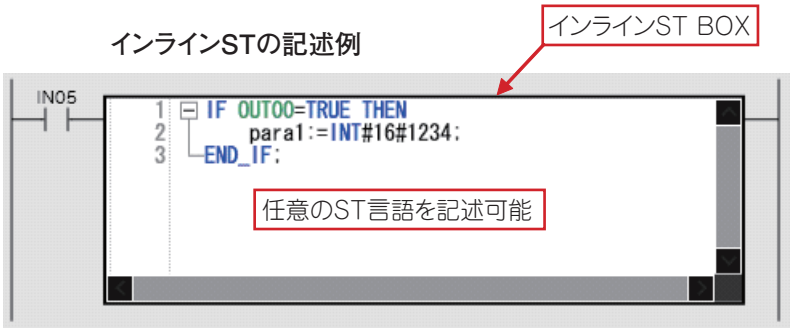


解説

- ST 言語
  ST 言語は、PASCAL をベースに規定された構造化テキスト言語です。数値演算式の表現やネスティングした条件分岐など、ラダー図言語が苦手としている用途で高い効果を発揮します。

- インライン ST
  NJ シリーズでは、ラダー図プログラム内の「インライン ST BOX」と呼ぶ、テキスト入力エリアに ST 言語を記述できます。
  これにより、数値演算処理や文字列操作処理を、ラダー図プログラム内で簡単に記述できます。

インラインSTの記述例



2-12 入力支援機能で快適プログラミング

従来の問題点

変数によるプログラミングにより、プログラムの可読性は向上しました。しかし、プログラミング時には、マニュアルやヘルプを参照しながら、正しいデータ型名や命令の引数を記述することに手間がかかりました。また、変数名の文字数が多い場合は、タイプミスすることもありました。

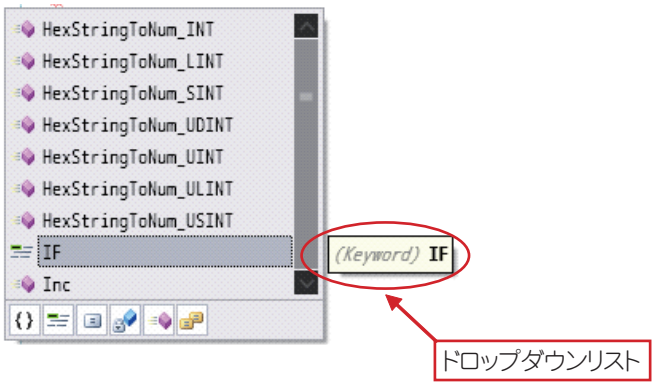
NJ で解決！

統合開発ツールの Sysmac Studio には、便利な入力支援機能が充実しています。

● コードスニペット機能

IF 文や CASE 文などの ST 構文を簡単に入力できます。

- ST エディタ上で、ST 言語の先頭文字を入力すると、変数候補がドロップダウンリストに表示されます。



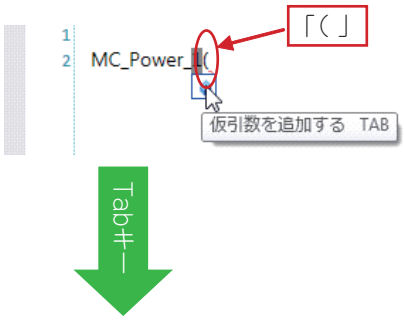
- ST 構文の直後にカーソルがある状態で、[Tab] キーを押すと、ST 構文を構成するキーワードが自動入力されます。

● ファンクション / ファンクションブロックのパラメータ入力支援機能

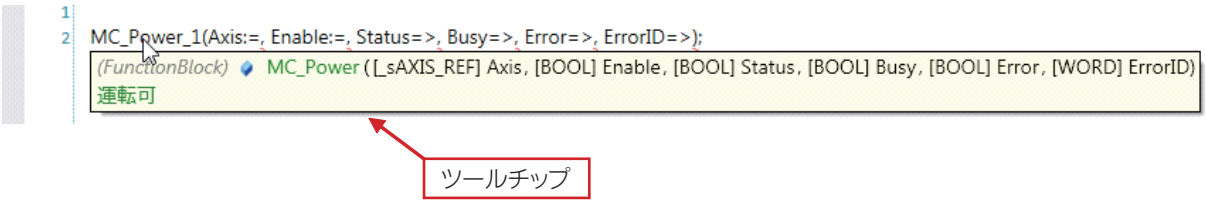
ファンクション / ファンクションブロックのパラメータを、簡単に入力できます。

- ST エディタ上でファンクション / ファンクションブロック名＋「(　」を入力し、[Tab] キーを押すと、ファンクション / ファンクションブロックの引数が自動入力されます。
  また、パラメータの説明もツールチップに表示されます。

FUN/FB名+「(」を入力し、[Tab]キーを
押します。

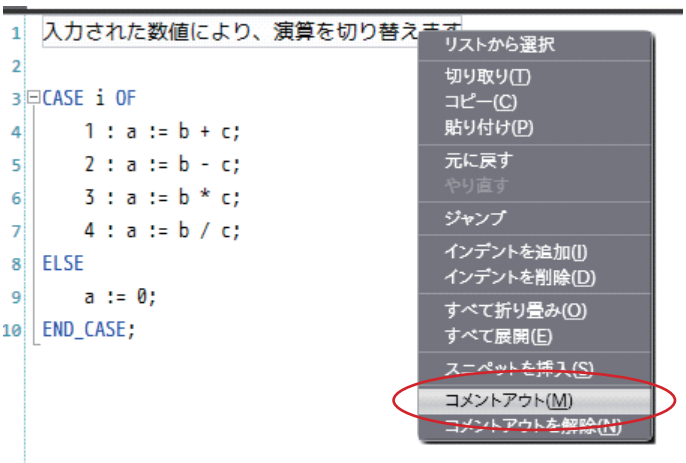


FUN/FBの仮引数が自動で入力されます。
必要箇所を入力･修正します。



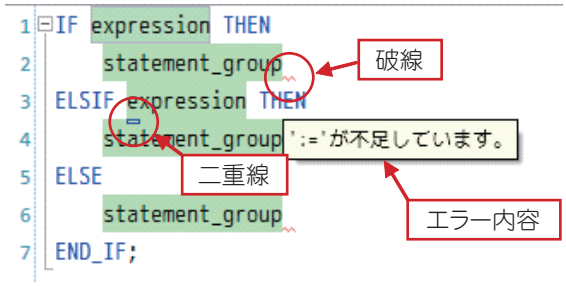
● コメントアウト機能

ST エディタ上で、ST 構文の文字列を選択し、右クリックして［コメントアウト］を選択することで、簡単にコメントアウトすることができます。デバッグ時に、回路の一部を一時的にスキップさせたいときに便利です。



● 自動文法エラーチェック機能

ST エディタ上で、ST 構文の文法にエラーがあった場合、自動的に赤色の波線が表示されます。カーソルを合わせると、エラー内容がツールチップに表示されるため、ビルドしなくても、すぐにエラー内容を確認することができます。
また、未登録の変数には、二重線が表示されます。



3

NS シリーズ表示器での画面設計の効用

本章では、NJ シリーズ CPU ユニットと NS シリーズ表示器を組み合わせて使用する場合の効用について説明します。

3-1　変数の活用で表示器画面の設計効率アップ

従来の問題点

アドレスによる表示器の画面設計は、コントローラ側でプログラミングするときのアドレス割り付けに依存するため、以下の問題がありました。

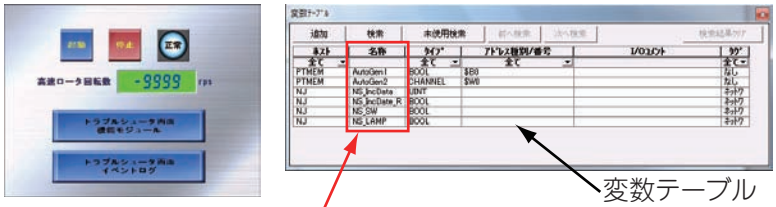
- コントローラ側のアドレスが決まらないと、作画に着手できない。
- 仮のアドレスを設定して作画しても、コントローラ側のアドレス決定後に変更する手間がかかる。

さらに、仕様変更などでアドレス割り付けに変更が発生すると、コントローラ側プログラムだけでなく表示器側の画面変更も必要となり、二重の修正と確認の作業が発生していました。

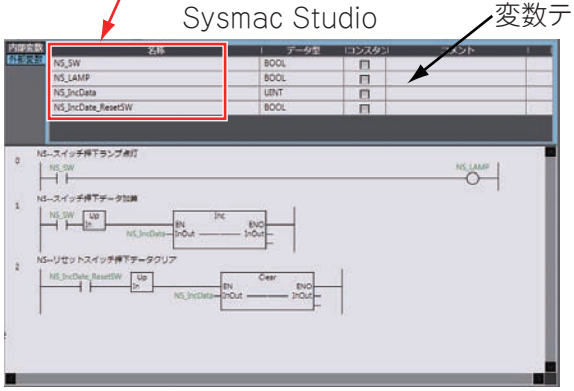
NJ で解決！

プログラマブルターミナル NS シリーズは、NJ シリーズやオムロン製コンポーネントとの親和性に優れた表示器です。
NS シリーズでは、NJ シリーズと同様に変数による画面設計をサポートしているため、アドレスではなく変数による作画が可能です。機器構成が決まらない場合でも、変数名を決めることができるので、画面設計に着手できます。また、コントローラ側の仕様変更などが発生しても変数名が変わらなければ、表示器側の画面データの修正は不要です。
画面設計には、NS シリーズ用の作画ツールである CX-Designer を使用します。

CX-Designerによる変数プログラミング



CX-DesignerにもSysmac Studioと同様に変数テーブルがあります。
コントローラ側の仕様変更により、プログラムが変更されても、表示器側で使用している変数に変更がなければ、画面データの修正は不要です。



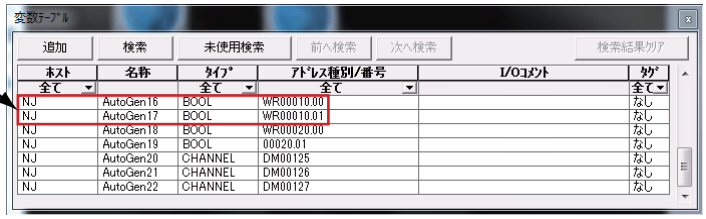
解説

「変数による画面設計」のさらなるメリット

- 「変数による画面設計」により可読性が向上します
  変数登録時に「変数」名に装置の機能名（例えば、RUN_SW、STOP_SW）を登録すると、スイッチやランプの動作が「変数」名で判断でき、画面データの可読性が大幅に向上します。
  従来のように「画面データ」と「ラダープログラム」を照らし合わせて「アドレス」からスイッチやランプの動作を確認する手間が省けます。

WR10.00/01に何を割り付けたの分からない。WR10.00/01をラダープログラムで確認する必要がある。

<FROM>　従来の物理アドレスの場合



1号機のRUNスイッチとSTOPスイッチであることが一目瞭然です!

<TO>　変数プログラミングの場合

- 変数による画面設計により再利用性が向上します
  アドレスによるタッチパネル画面の作画は、装置の仕様により、コントローラ機種や使用ユニット数が変更になるとアドレスの割り付けが変更になるため、画面修正が必要でした。
  機器構成に依存しない「変数による画面設計」により、画面の再利用性が高まるため、画面作画効率を大幅に向上できます。

「装置A」用の画面

「変数による画面設計」により、「装置B」や「装置C」にも流用できます。

●本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途に使用される際には、当社の意図した特別な商品用途の場合や特別の合意がある場合を除き、当社は当社商品に対して一切保証をいたしません。
●本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物（又は技術）に該当するものを輸出（又は非居住者に提供）する場合は同法に基づく輸出許可、承認（又は役務取引許可）が必要です。

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

●製品に関するお問い合わせ先
お客様相談室
フリー通話 **0120-919-066**（クイック／オムロン）
携帯電話・PHS・IP電話などではご利用いただけませんので、下記の電話番号へおかけください。
電話 **055-982-5015**（通話料がかかります）
■営業時間：8:00～21:00　■営業日：365日

●FAXやWebページでもお問い合わせいただけます。
FAX 055-982-5051 / www.fa.omron.co.jp

●その他のお問い合わせ
納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。
オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。

オムロン制御機器の最新情報をご覧いただけます。
www.fa.omron.co.jp
緊急時のご購入にもご利用ください。

オムロン商品のご用命は

Man. No. SBCA-405B　2018年12月現在