CKD

スカラロボット

KSL シリーズ

（ロボット言語マニュアル）

取扱説明書

SM-A20050

- 製品をご使用になる前に、本取扱説明書を必ずお読みください。
- 特に安全に関する記述は、注意深くお読みください。
- 本取扱説明書は必要なときにすぐ取出して読めるように、大切に保管してください。

CKD株式会社

はじめに

このたびは、当社の**スカラロボット「KSL シリーズ」**をお買求めいただきまして、誠にありがとうございます。
本取扱説明書は本製品の性能を十分に発揮させるために、取付、使用方法などの基本的な事項を記載したものです。よくお読みいただき、正しくご使用ください。
なお、本取扱説明書は紛失しないように、大切に保管してください。

本取扱説明書に記載の仕様、外観は、将来予告なく変更することがあります。

注意：

- この取扱説明書は産業用ロボットを実際にご使用になられる方のお手元に必ず届くようお取りはからいください。
- 産業用ロボットをご使用前にこの取扱説明書を必ずご覧くださいますようお願いいたします。
- お読みになった後は必ず保管してくださいますようお願いいたします。

本編では，ＫＳＬシリーズで使用するロボット言語、ＳＣＯＬ言語の命令語とプログラムの組み方について説明します。

ＳＣＯＬ言語（Symbolic Code Language for robot）は、ロボットの種々の機能に対応した命令語から構成されます。これらの命令語を使う事によりロボットの動作を簡潔にプログラムすることができます。

本編は、初めてロボットのプログラムを作成する方から、プログラミングに習熟した方まですべての方を対象に書いています。しかし、内容をロボット言語に限定しているため、コントローラＴＳシリーズの概要及び操作方法については、下記の各編を参照してください。

操 作 編

本編の構成は、次の通りです。

「１．ロボット言語の概要」

ここでは、ロボットの動作とロボット言語の関係、及びロボット言語の命令語の大まかな機能について説明しています。ロボット言語の概要を知る上でも、必ず一読するようお願いします。

「２．ロボット言語の書式」

ここでは、ロボット言語を用いてプログラムを作成する時の種々の規則について説明しています。プログラムの作成前に必ず一読するようお願いします。

「３．命令語の説明」

ここでは、ロボット言語の個々の命令語について、その詳細を説明しています。命令語は、アルファベット順に記載していますので、プログラム作成時に参照してください。

「４．プログラム例」

ここでは、ロボット言語を使用したプログラムの例について説明しています。ロボット言語を使用してプログラムを作成する時の参考にしてください。

「５．プログラミング上の注意事項」

ここでは、実際にプログラムを作成する上でのタイミング、禁止事項、及び注意事項について説明しています。プログラムの作成前に一読しておいてください。また、作成したプログラムがうまく動かない時にも本章を読み直してみてください。

目　　次

第１章

ロボット言語の概要

本章では、ロボットの動作とロボット言語の関係、及びロボット言語の命令語の機能について説明します。

1.1 ロボットの動作

ロボットは、人間が行う仕事を人間と同じように行います。例として、コンベア上を流れてくるワークに部品を取付ける作業を考えます。作業者は部品箱から部品を取出し、コンベア上を流れてくるワークに取付けていきます。この作業を人間の代わりにロボットが行う場合を考えると、図1.1 に示すような形になります。

図1.1でロボットは、パーツフィーダと呼ばれる部品供給装置から部品をつかみ、コンベア上を流れてくるワークの所定位置に部品を取付けます。この作業をロボットの動作だけで考えると、図1.2に示すような形になります。ロボットは部品をつかむ位置Ａの真上の位置Ｂから下降して、位置Ａで部品をつかみます。部品をつかむと位置Ｂまで上昇して、部品取付け位置Ｄの真上の位置Ｃまで移動します。位置Ｃからロボットはそのまま下降して、位置Ｄで部品をワークに取付けます。部品を取付け終わると、ロボットは位置Ｃを経由して位置Ｂに戻り、１回の作業を終了します。

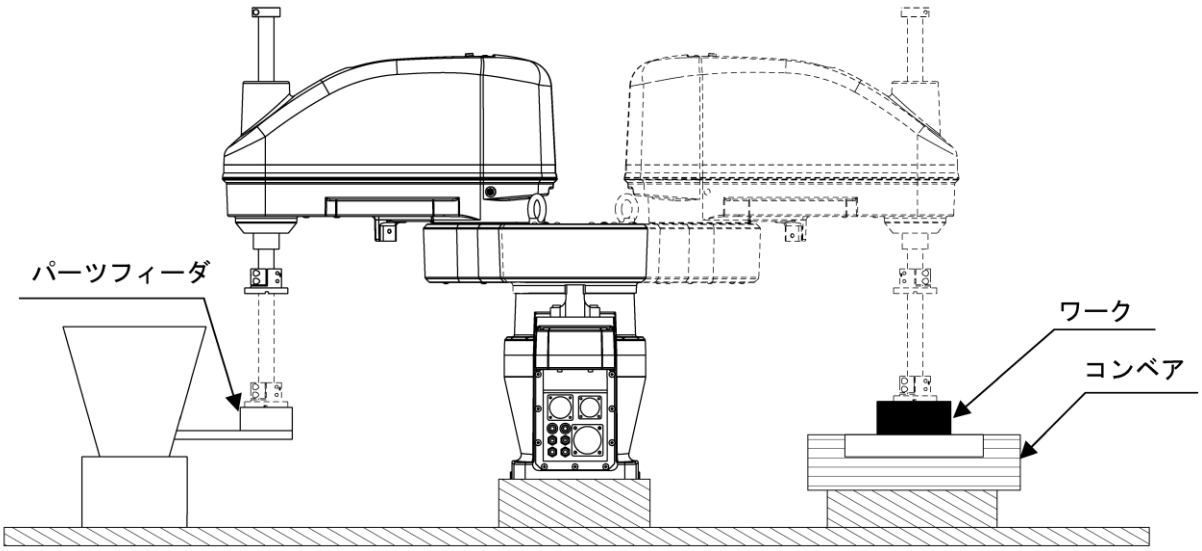


図 1.1 組立て作業

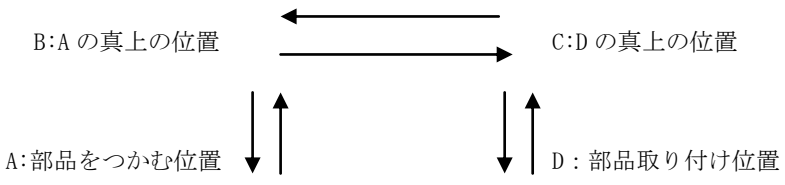


図 1.2 ロボットの動作

1.2 ロボット言語とは

ロボットは、組立て等の仕事を人間に代わって行います。しかし、ロボットに何を行うかを教えるのは人間の仕事です。ロボットは教えられた仕事を教えられた通りに行います。ロボットに仕事を教える事を教示（ティーチング）と呼びます。また、教示した内容に従ってロボットが動作することを再生（プレイバック）と呼びます。このように教示した内容どおりに動作するロボットを、プレイバックロボットと呼びます。

ロボットに仕事を教える方法として､作業の順番通りにロボットを動かす方法があります。塗装作業やスポット溶接に使われるロボットでは、人間が仕事を行う時と同じように、ロボットを動かしてロボットに仕事を教えます。ロボットは教えられた通りの順番で動作して、与えられた仕事を行います。

しかし、より複雑な仕事を行うためには周辺機器の状態に応じてロボットの動作を変えていかなければなりません。前節に示した組立て作業の例でも、流れてくるワークの種類によって取付ける部品を変えたり、取付けにミスがあった時、再度取付けを繰返すような作業が考えられます。このような作業では、ロボットはその時の周辺に応じて動作を行わなければなりません。そこで「どういう時はどうする」、「こういう時にはこうする」というようにロボットの動作を表す方法が考えられます。

ロボットの動作を記述する言語を、ロボット言語と呼びます。ロボットの行う作業はロボット言語を用いて表したものをプログラムと呼び、プログラムを作成する作業をプログラミングと呼びます。ＴＳシリーズでは、ロボット言語として独自のＳＣＯＬ言語（スコール： Symbolic Code Language for robot の略称）を用いています。前節の組立て作業の例をＳＣＯＬ言語を用いて表すと次のようになります。

```
ＰＲＯＧＲＡＭ　ＡＳＳＥＭＢＬＹ
  ＭＯＶＥ　Ｂ          位置Ｂへ移動する
  ＯＰＥＮ１            ロボットのハンド１を開く
  ＭＯＶＥ　Ａ          位置Ａへ移動する
  ＣＬＯＳＥ１          ロボットのハンド１を閉じる
  ＤＥＬＡＹ　０．５    ロボットが部品をつかむまで０．５秒間待つ
  ＭＯＶＥ　Ｂ          位置Ｂへ移動する
  ＭＯＶＥ　Ｃ          位置Ｃへ移動する
  ＭＯＶＥ　Ｄ          位置Ｄへ移動する
  ＯＰＥＮ１            ロボットのハンド１を開く
  ＤＥＬＡＹ　０．５    ロボットが部品をはなすまで０．５秒間待つ
  ＭＯＶＥ　Ｃ          位置Ｃへ移動する
  ＭＯＶＥ　Ｂ          位置Ｂへ移動する
ＥＮＤ
```

ＰＲＯＧＲＡＭ ～ ＥＮＤ までが１つのプログラムになり、このプログラムには、“ＡＳＳＥＭＢＬＹ”という名前が付けられています。”ＭＯＶＥ Ａ”は位置Ａにロボットを移動する事を表します。”ＯＰＥＮ１”、”ＣＬＯＳＥ１”はそれぞれロボットのハンド１を開く、閉じる事を表します。”ＤＥＬＡＹ ０．５”はロボットの動作を０．５秒間停止する事を表します。また位置Ａ，Ｂ，Ｃ，Ｄはプログラム作成とは別にロボットをその位置まで誘導して教示します。

この様に、ＳＣＯＬ言語ではロボットの動作に対応した命令語を作業を行う順番に並べる事で、ロボットの行う仕事を表す事ができます。

1.3 命令語の種類

ＳＣＯＬ言語によりロボットの動作を表わせる事を説明しましたが、本節ではＳＣＯＬ言語の命令語の概要を説明します。

ＳＣＯＬ言語には、“ＭＯＶＥ Ａ”のようにロボットを動かすものの他にも、外部への信号出力や、作業の繰返しを指定するもの等、様々な命令語が含まれます。表１．１にＳＣＯＬ言語の機能の一覧表を示します。

ＳＣＯＬ言語の命令語は、大きく６種類に分けられます。

(1) 動作制御命令

ロボットの本体を動かす命令語です。ロボットを一定時間停止したり、ロボットの動作を中断、再開する命令語も動作制御命令に含みます。特に、ロボット本体を動かす命令語を動作命令と呼びます。

(2) プログラム制御命令

外部信号の状態によりプログラムの実行を分岐したり、作業の繰返しを指定する等のように、プログラムの実行を制御する命令語です。

(3) 入出力制御命令

外部信号の読込み及び出力を行う命令語です。ハンドの開閉，通信チャンネルへのデータの入出力も入出力制御命令に含みます。

(4) 動作条件命令

ロボットが動作する時の姿勢、速度等の条件を指定する命令語です。

(5) 算術演算命令

平方根、三角関数等の演算を行う命令語です。

(6) 動作状態命令

ロボットの動きが１つの動作の何％まで動いたか等のように、ロボットの動作状態を参照する命令語です。プログラム中で使用するタイマーの設定、ロボットの運転モードの参照を行う命令語も含みます。

これらの命令語は他の命令語と組合わせて、例えば、ロボットが動作の７０％まで動いた時に外部に信号を出力したり、ある動作を開始してから指定時間以内に動作が終了しなければ異常と見なしてプログラム分岐したりするために用います。

これらの命令語を組合わせる事により、ロボットの目的に合った作業をプログラムする事ができます。

表 1.1 ＳＣＯＬ言語の機能

分　類	機　　能	命　　令　　語
動作制御命令	（１） ロボットを動かす。	ＭＯＶＥ，ＭＯＶＥＳ， ＭＯＶＥＣ，ＭＯＶＥＡ， ＭＯＶＥＩ，ＲＥＡＤＹ,ＭＯＶＥＪ
	（２）ロボットを一定時間停止する。	ＤＥＬＡＹ
	（３）ロボットのハンドを動かす。	ＯＰＥＮ１，ＯＰＥＮＩ１， ＯＰＥＮ２，ＯＰＥＮＩ２， ＣＬＯＳＥ１，ＣＬＯＳＥＩ１， ＣＬＯＳＥ２，ＣＬＯＳＥＩ２，
	（４）ロボットの動作を中断・再開する。	ＢＲＥＡＫ，ＰＡＵＳＥ， ＲＥＳＵＭＥ
プログラム制御命令	（１）外部信号、タイマー等の状態を監視する。	ＯＮ～ＤＯ～，ＩＧＮＯＲＥ， ＩＦ～ＴＨＥＮ～ＥＬＳＥ， ＷＡＩＴ，ＴＩＭＥＲ
	（２）プログラムの実行を制御する。	ＰＲＯＧＲＡＭ，ＥＮＤ，ＧＯＴＯ， ＲＣＹＣＬＥ，ＲＥＴＵＲＮ， ＦＯＲ～ＴＯ～ＳＴＥＰ～ＮＥＸＴ， ＳＴＯＰ，ＴＡＳＫ，ＫＩＬＬ， ＳＷＩＴＣＨ，ＴＩＤ，ＭＡＸＴＡＳＫ
	（３）プログラムの注釈文	ＲＥＭＡＲＫ
入出力制御命令	（１）外部信号の読込み、出力を行う。	ＤＩＮ，ＤＯＵＴ， ＰＵＬＯＵＴ，ＲＥＳＥＴ ＢＣＤＩＮ，ＢＣＤＯＵＴ
	（２）通信データの読込み、出力を行う。	ＰＲＩＮＴ，ＩＮＰＵＴ

分　類	機　　能	命　　令　　語
動作条件命令	（１）ロボットが動作する時の条件を設定する。	ＣＯＮＦＩＧ，ＡＣＣＵＲ， ＡＣＣＥＬ，ＤＥＣＥＬ， ＳＰＥＥＤ，ＰＡＳＳ， ＴＯＲＱＵＥ，ＧＡＩＮ， ＥＮＡＢＬＥ， ＳＥＴＧＡＩＮ， ＤＩＳＡＢＬＥ， ＮＯＷＡＩＴ， ＰＡＹＬＯＡＤ， ＦＲＥＥＬＯＡＤ， ＳＷＩＴＣＨ， ＭＯＶＥＳＹＮＣ
パレタイズ命令	（1）ライブラリロード （2）パレットの初期化 （３）パレットの指定位置に移動	ＬＯＡＤＬＩＢ ＩＮＩＴＰＬＴ ＭＯＶＥＰＬＴ
算術演算命令	（１）実数の演算を行う。 （２）位置データ、座標データの演算を行う。 （３）配列を扱う	ＳＩＮ，ＣＯＳ，ＴＡＮ， ＡＳＩＮ，ＡＣＯＳ， ＡＴＡＮ，ＡＴＡＮ２， ＳＱＲＴ，ＡＢＳ，ＳＧＮ， ＩＮＴ，ＲＥＡＬ，ＬＮ， ＭＯＤ，ＬＯＧ１０，ＥＸＰ， ＡＮＤ，ＯＲ，ＮＯＴ ＨＥＲＥ，ＤＥＳＴ， ＰＯＩＮＴ，ＴＲＡＮＳ ＤＩＭ，ＡＳ
動作状態命令	（１）ロボットの動作状態を調べる。 （２）システムの動作状態を調べる。 （３）ロボットの各座標系を与える。	ＭＯＴＩＯＮ，ＭＯＴＩＯＮＴ ＲＥＭＡＩＮ，ＲＥＭＡＩＮＴ ＭＯＤＥ，ＣＯＮＴ，ＣＹＣＬＥ， ＳＥＧＭＥＮＴ ＴＯＯＬ，ＢＡＳＥ，ＷＯＲＫ
その他	（１）変数を定義する （２）更新された変数値をプログラムファイルに戻す。 （３）電源遮断時にデータを保存する。	ＧＬＯＢＡＬ，ＤＡＴＡ，ＥＮＤ ＲＥＳＴＯＲＥ ＳＡＶＥＥＮＤ

第２章

ロボット言語の書式

第１章では、ロボット言語の概要について述べてきましたが、本章では、ロボット言語でプログラムを組む方法について説明します。

2.1 プログラムの構成

本節では、ＳＣＯＬ言語でのプログラムの構成の概要について説明します。

2.1.1 ファイル

ロボットが作業を行うためには、ロボット言語で書かれたプログラムとそのプログラム中で使用する位置のデータが必要です。

ＴＳシリーズでは、プログラムと位置データの組がそれぞれの作業に必要になり、１つの作業に対応したプログラムと位置データのまとまりをファイルと呼びます。プログラムの実行、編集は、このファイル単位で行います。

2.1.2 プログラム

ロボットが行う作業をロボット言語を使って表したものを、プログラムと呼びます。プログラムの中から別のプログラムを呼出して使用する事もできます。使用頻度の高い動作や、ある程度まとまった動作は１つのプログラムとして用意しておき、必要な時にそのプログラムを呼出す使い方ができます。このようなプログラムをサブプログラムと呼びます。これに対してサブプログラムを呼出すプログラムをメインプログラムと呼びます。

１つのファイルには、複数のプログラムを記述する事ができます。プログラムの実行時に何も指定が無ければ、ファイルの先頭に記述されたプログラムをメインプログラムとして実行します。サブプログラムを呼出す時には、呼出されるサブプログラムもメインプログラムと同じファイルになければなりません。また、１つのファイルに複数のプログラムを書いても、順番にすべてのファイルを実行していくわけではなく、プログラムの実行はメインプログラムの最後までが１つの実行単位となります。

また、プログラムはＴＡＳＫ命令を用いて同時に複数実行すること（マルチタスク実行）ができます。マルチタスク実行の詳細については本編２．８ プログラムの項を参照して下さい。

プログラムの編集は、コントローラのプログラムエディタ機能を用いてティーチペンダントから行います。エディタの使い方については、「操作編」を参照してください。

2.1.3 位置データ

プログラムにて使用する位置データは、その位置データを使用するプログラムと同じファイルに作成します。ファイル上に作成した位置データは、そのファイルに存在する全てのプログラムにて共通に使用します。異なるファイルの位置データを使用する事はできません。
位置データは、コントローラのデータエディタ機能を用いて教示します。データエディタの使い方については、「操作編」を参照してください。

2.2 使用できる文字

ＳＣＯＬ言語では、英数字と特殊文字が使用できます。

英数字　ＡＢＣＤＥＦＧＨＩＪＫＬＭＮＯＰＱＲＳＴＵＶＷＸＹＺ
ａｂｃｄｅｆｇｈｉｊｋｌｍｎｏｐｑｒｓｔｕｖｗｘｙｚ
１２３４５６７８９０

特殊記号 ” ´ （ ） ＋ － ＊ ／ , . ＜ ＞ ＝
！ ［ ］ ｛ ｝ ％ ＾ ＆ ？
スペース

これらの文字は、小文字を除いてすべてティーチペンダントから入力可能です。ティーチペンダントからは小文字は入力できません。プログラムを実行する時には、大文字と小文字は区別されません。記号の読み方は付録Ｅを参照してください 。

2.3 識別子

ＳＣＯＬ言語中で命令語、プログラム名、変数名、及びプログラムの分岐先を指定するラベル名はすべて識別子で表します。識別子は英字で始まる英数字で表します。長さには特に制限はありませんが、実行時には先頭の１０文字のみにて区別されます。大文字と小文字の区別は行いません。また、識別子の中に特殊記号やスペースを用いる事はできません。

識別子と識別子の区切りには特殊記号かスペースを用います。

例） ＴＯＳＨＩＢＡＲＯＢ
ｔｏｓｈｉｂａｒｏｂ
ＴＯＳＨＩＢＡＲＯＢＯＴ
上記の３つの識別子は、すべて“ＴＯＳＨＩＢＡＲＯＢ”と見なされます。

ＳＣＯＬ言語で既に使用されている識別子は、別の用途で使用することはできません。このような識別子には、ＳＣＯＬ言語の命令語の他に、システムで特別に使用するものや、今後の拡張用に予約されている識別子も含みます。付録ＢにＳＣＯＬ言語の予約語を示しますので参照してください。

同一名称の識別子を異なる意味で用いないでください。例えば、プログラムを作成する場合に、プログラム名と変数名に同じ識別子を用いるような使い方はしないでください。

2.4 変数と定数

変数と定数は、扱うデータの内容によって分けて考えます。それをデータ型と呼び、ＳＣＯＬ言語ではスカラ型（整数型、実数型、及び文字列）とベクトル型（位置型、座標型、及び負荷型）を使用できます。変数はまた、定義の方法によってグローバル変数と、オート変数に分けられます。全ての教示済みデータと、予約語ＧＬＯＢＡＬ～ＥＮＤで囲まれた部分で定義された変数をグローバル変数と呼び、これらはプログラム中のどこからでも参照や変更することが出来ます。またグローバル変数では、全てのデータ型に対して配列を宣言することも出来ます。グローバル変数や配列の説明は、
２．８．５を参照してください。

全てのデータはコントローラ内の作業領域に展開されますが明示的に初期化を設定していない配列変数を除いてグローバル変数はプログラムの実行を始める時に定義した値となっていることが保証されます。しかしプログラム中で変数に値を代入しても作業領域が変更されるだけで、コントローラの電源を落としたり、実行ファイルの選択をやりなおしたり、ファイルの編集作業を行った場合には、ファイル内に保存されている変数の初期値で作業領域を再設定する為、変更した内容は失われます。教示済みのデータに対する変更も同様です。ファイルのデータ自体を書き換えるには、プログラムでＲＥＳＴＯＲＥ命令を実行する必要があります。

2.4.1 スカラ型データ

スカラ型データには、整数型、実数型、及び文字列の３種類があります。スカラ型オート変数は宣言されたプログラムの中でのみ使用できます。よって、他のプログラムで同一名称の変数を使用していたとしてもその内容は一致しません。プログラムからプログラムへとデータを渡す場合は、スカラ型グローバル変数として定義し直すか、プログラムにて引数を宣言するようにします。詳細は、「２．８　プログラム」を参照してください。

(1) 整数型データ

(a) 定数

－２１４７４８３６４８　～　＋２１４７４８３６４７　の範囲の整数を扱うことができます。プログラム中で整数を定数として使用するときは、正の数の場合は直接数値を、負の数の場合は－の符号に続けて数値を記述します。ＩＮＰＵＴ命令で１１桁以上を入力するとエラーになります。

例）　　０
　　２３４
　－３９２０８
　　５９６３

(b) 変数

変数は識別子により区別されます。扱う数値は定数の場合と同じです。変数のデータ型は、その変数に最初に代入する数値のデータ型によって決まります。すなわち、最初に代入するデータが整数型の時はその変数は整数型となります。整数型の変数に実数を代入する場合には、実数の小数点以下を切捨てた値が整数値として代入されます。

変数にはプログラム全体で有効な変数（グローバル変数）とプログラムの一部で有効な一般の変数があります。グローバル変数はプログラムのどこからでも変更することができます。

(c) 論理値

プログラムの中で条件判断を行う場合には、論理値が用いられます。入力信号の状態を読込む命令語（ＤＩＮ）や、論理式は、実行結果として論理値を返します。論理値は真、偽の２つの値を持ちます。内部的には整数型のデータとして扱われ真は１に、偽は０になります。

注）　実際には式の値が０ならば偽，０以外ならば真と判断しています。

(2) 実数型データ

ＳＣＯＬ言語では、数値は特別な場合を除いて実数の形で扱います。

(a) 定数

数値の絶対値が　約５．８７×１０$^{-39}$（２$^{-127}$）～６．８０×１０^{38}((２^{23}－１）×２^{106}）の範囲の実数を扱うことができます。仮数部の有効数字は１０進数で約７桁です。

（２^{23}の精度です。）許される桁数は整数部９桁小数部３桁です。ＩＮＰＵＴ命令でこの桁数以上の数値が入力されるとエラーになります。

プログラム中に実数を使用する時は、正の数の場合は直接数値を、負の数の場合はーの符号に続けて数値を記述します。また、小数部が０の時の小数部は省略可能です。ただし、小数点を省略すると整数型のデータと見なされます。整数部は省略できませんので、数値の絶対値が１未満の時でも整数部には０を指定してください。

例）　　１２３４．５６７
　　－２８．１６
　　０．００９８５
　１２３４５６７．
　　－３６９．

小数点以下の数値を設定する場合には、実行結果の精度が限られるため、以下に示す最小設定単位を目安にしてください。

位置	（X，Y，Z　データ）	０．００１　ｍｍ	単位
角度	（C）	０．００１　度	単位
時間		０．０１　秒	単位
率	（速度、トルク等）	１　％	単位
質量		０．０１　ｋｇ	単位
慣性		０．０１　ｋｇ・ｍ	単位

(b) 変数

変数は識別子により区別されます。扱う数値は定数の場合と同じです。変数のデータ型は、その変数に最初に代入する数値のデータ型によって決まります。すなわち、最初に代入するデータが実数型の場合には、その変数は実数型となります。

(3) 文字列

文字列は、定数のみが使用可能で、１つ以上の文字を「”」で囲って表します。

例）　”ＳＣＯＬ　ＭＥＳＳＡＧＥ”

2.4.2 ベクトル型データ

スカラ型データは、１つの要素しか持たないのに対して、ベクトル型データは、１つのデータに複数の要素を持っています。ベクトル型データには位置型、座標型、及び負荷型の３種類があります。

ベクトル型のデータは１～５個の要素を{　}で囲んで表すことができます。位置型、座標型、および負荷型の他に、ＴＯＲＱＵＥやＧＡＩＮ命令でも、データは{　}で囲んだベクトル型のデータで指定します。

データエディタで教示したデータ等のベクトル型グローバル変数以外のベクトル型データは、コントローラの作業領域に一時的に作成され、ファイルの中には作成されません。このようなベクトル型変数は、宣言されたプログラムの中でのみ使用できます。よって他のプログラムで同一名称の変数を使用していたとしてもその内容は一致しません。プログラムからプログラムへとデータを渡す場合は、ベクトル型グローバル変数として定義し直すか、要素に分解して引数として受渡すようにします。引数については「2.8.2サブプログラム」を参照してください。

(1) 位置型データ

位置型データは、ロボットの位置データとして使用されます。位置型データの構成は、次の通りです。

（ Ｘ ， Ｙ ， Ｚ ， Ｃ ， Ｔ ， ＜姿勢＞ ）

Ｘ，Ｙ，Ｚ，Ｃ，Ｔ ： Ｘ、Ｙ、Ｚ、Ｃ、Ｔ の各座標値で、実数値をとります。
（ｍｍ，ｄｅｇ 単位）

＜姿勢＞ ： ０～２の整数値でロボットの姿勢を表します。
０・・・フリー（姿勢未定義）
１・・・左肩系
２・・・右肩系

(2) 座標型データ

座標型データは、ロボットの座標系を指定するために使用されます。座標型データの構成は、次の通りです。

（ Ｘ ， Ｙ ， Ｚ ， Ｃ ）

Ｘ，Ｙ，Ｚ，Ｃ ： Ｘ、Ｙ、Ｚ、Ｃ の各座標値で、実数値をとります。
（ｍｍ，ｄｅｇ 単位）

座標型データによりロボットの座標系は、図２．１に示すように変換されます。元の座標系Ｘ、Ｙ、Ｚは、座標型データ（ｘ，ｙ，ｚ，c）により、それぞれの座標軸に沿って、ｘ、ｙ、ｚの量だけ平行移動されます。平行移動された座標系の原点Ｏ´を中心にＺ軸の回りにｃだけ回転した座標系Ｘ´，Ｙ´，Ｚ´が新しい座標系になります。

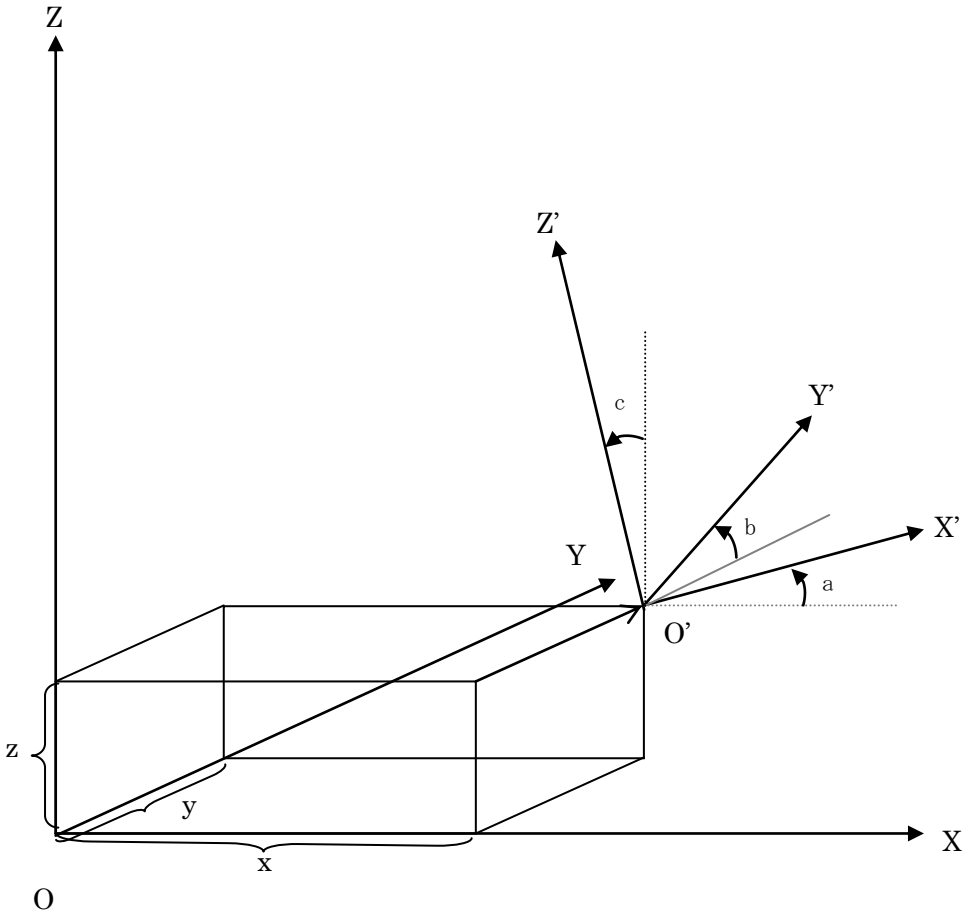


図 2.1 座標系の変換

(3) 負荷型データ

負荷型データは、ハンド部分（エンドエフェクタ部分）を含めた、ロボットの手先にかかる負荷を定義するものです。負荷型データの構成は、次の通りです。

{＜質量＞，＜重心オフセット＞}

＜質　　量＞ ： ロボットの手先にかかる負荷の質量です。（ｋｇ　単位）
＜重心オフセット＞ ： ロボットの手先にかかる負荷の重心が、手先のツ―ル中心からどれだけ離れているかの量です。（㎜　単位）

2.4.3 システム変数

ＳＣＯＬ言語では、ロボットが動作する時の条件や、システムの状態をプログラム上で指定、参照するために特別な変数を用意しています。これらの変数をまとめてシステム変数と呼びます。システム変数は、通常の変数と同じように代入や参照が可能です。しかし、代入によりシステム変数の設定を変えることは、ロボットの動作条件を変えることになりますので、十分に注意してください。システム変数の一覧表を、表２．１に示します。

表 2.1.　システム変数一覧表

名　前	内　　容	有効値	初期値	データ型
ＣＯＮＦＩＧ	ロボットの姿勢	０、１、２	０	整　数　型
ＡＣＣＵＲ	位置決め精度	０、１	１	整　数　型
ＡＣＣＥＬ	加速時の加速度	０～ｍａｘ％	１００	整　数　型
ＤＥＣＥＬ	減速時の加速度	０～ｍａｘ％	１００	整　数　型
ＳＰＥＥＤ	速度	０～ｍａｘ％	１００	整　数　型
ＰＡＳＳ	ショートカット動作のパラメータ	０～１００％	１００	整　数　型
ＴＯＲＱＵＥ	各軸最大トルク	０～ｍａｘ％	３００	ベクトル型
ＧＡＩＮ	各軸サーボゲイン	０、１	１	ベクトル型
ＴＯＯＬ	ツール座標系		０	座　標　型
ＢＡＳＥ	ベース座標系		０	座　標　型
ＷＯＲＫ	ワーク座標系		０	座　標　型
ＴＩＭＥＲ	タイマ	０～	－	実　数　型
ＥＲＲＯＲ	エラー	－		整　数　型
ＰＡＹＬＯＡＤ	ロボットの負荷	０～	０	負　荷　型
ＳＷＩＴＣＨ	マルチタスク	０、１	１	整　数　型
ＴＩＤ	タスク番号	１～	－	整　数　型
PLCDATAR1～8	簡易ＰＬＣ　Ｉ／Ｆ	０～６５５３５	０	整　数　型
PLCDATAW1～8	簡易ＰＬＣ　Ｉ／Ｆ	０～６５５３５	０	整　数　型

注）ｍａｘの値はシステム毎に設定されます。

動作制御に関するシステム変数の内容を更新した場合には、その効果はその後の動作から影響し、実行中の動作には影響しません。システム変数の効果は、システム変数を再設定するまで持続します。ただし、ＷＩＴＨ節を用いることにより１つの動作命令に対してだけ、一時的にシステム変数の値を設定することができます。

例）　ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＳＰＥＥＤ＝５０

また、システム変数への代入を行うときには有効範囲のチェックは行いません。実行時に設定値が有効範囲外の時には、システムは次の規則に従って値を設定します。

・最小値以下の値を設定しようとした場合には、最小値が設定されます。
・最大値以上の値を設定しようとした場合には、最大値が設定されます。

各システム変数の詳細な使い方は、３章を参照してください。

2.4.4 システム定数

ＳＣＯＬ言語では、プログラムを読み易くするために、表２．２に示すシステム定数を用意しています。これらの定数は、数値の代わりにプログラム中に記述する事ができます。しかし、プログラムが見づらくなるため、表２．２のコメントに示した箇所以外では使用しないでください。

表 2.2 システム定数一覧表

名前	値	コメント （使用する箇所）
ＦＲＥＥ ＬＥＦＴＹ ＲＩＧＨＴＹ	０ １ ２	システム変数 ＣＯＮＦＩＧ ＰＯＩＮＴ命令
ＣＯＡＲＳＥ ＦＩＮＥ	０ １	システム変数 ＡＣＣＵＲ
ＯＦＦ ＯＮ	０ １	システム変数 ＧＡＩＮ ＳＥＴＧＡＩＮ命令
ＰＡＩ	３．１４１５９３	円周率
ＣＯＮＴ ＣＹＣＬＥ ＳＥＧＭＥＮＴ	０ １ ２	ＭＯＤＥ命令

2.5 式

本節では、ＳＣＯＬ言語にて代入、演算、判定に使用する式について説明します。

ＳＣＯＬ言語で、式は単独で代入、演算を行う他に、命令の中に記述して使うこともできます。式の中には演算結果を変数に代入する演算式や、大小、真偽の判定を行う論理式があります。演算子としては、以下に示すものが使用できます。なお０／０と０＾０の実行結果は、エラーとならずそれぞれ－１，０になりますのでご注意ください。

表 2.3 演算子一覧表

種類	演算子	演算	例
算術	＾	べき乗	Ａ＾Ｂ （ＡのＢ乗）
演算子	－	負号	－Ａ
	＊，／	乗算、除算	Ａ＊Ｂ，Ａ／Ｂ
	＋，－	加算、減算	Ａ＋Ｂ，Ａ－Ｂ
	ＭＯＤ	剰余	Ａ ＭＯＤ Ｂ（ＡをＢで割った余り）
	＝	代入	Ａ＝Ｂ （Ｂの値をＡに代入）
関係	＝＝	等しい	Ａ＝＝Ｂ
演算子	＜＞，＞＜	等しくない	Ａ＜＞Ｂ，Ａ＞＜Ｂ
	＜	小さい	Ａ＜Ｂ
	＞	大きい	Ａ＞Ｂ
	＜＝，＝＜	小さいか等しい	Ａ＜＝Ｂ，Ａ＝＜Ｂ
	＞＝，＝＞	大きいか等しい	Ａ＞＝Ｂ，Ａ＝＞Ｂ
論理	ＡＮＤ	論理積	Ａ ＡＮＤ Ｂ
演算子	ＯＲ	論理和	Ａ ＯＲ Ｂ
	ＮＯＴ	否定	ＮＯＴ Ａ
関数	ＳＩＮ	正弦	ＳＩＮ（Ａ）
	ＣＯＳ	余弦	ＣＯＳ（Ａ）
	ＴＡＮ	正接	ＴＡＮ（Ａ）
	ＡＳＩＮ	逆正弦	ＡＳＩＮ（Ａ）
	ＡＣＯＳ	逆余弦	ＡＣＯＳ（Ａ）

種類	演算子	演算	例
関数	ＡＴＡＮ	逆正接	ＡＴＡＮ（Ａ）
	ＡＴＡＮ２	逆正接	ＡＴＡＮ２(Ａ，Ｂ)（A/B の逆正接）
	ＳＱＲＴ	平方根	ＳＱＲＴ（Ａ）
	ＡＢＳ	絶対値	ＡＢＳ（Ａ）
	ＳＧＮ	符号	ＳＧＮ（Ａ）　（Ａの符号を取出す）
	ＩＮＴ	整数化	ＩＮＴ（Ａ）
	ＲＥＡＬ	実数化	ＲＥＡＬ（Ａ）
	ＬＮ	自然対数	ＬＮ（Ａ）
	ＬＯＧ１０	常用対数	ＬＯＧ１０（Ａ）
	ＥＸＰ	ｅのべき乗	ＥＸＰ（Ａ）

式の中には、括弧（　）を使う事ができます。

2.5.1 演算式

ＳＣＯＬ言語で、演算は式の右辺の結果を左辺に代入します。式の中には変数、定数を使うことができます。

(1) 演算の優先順位

ＳＣＯＬ言語では、通常の算術演算と同様の優先順位で演算を行います。具体的に、以下の規則に従って処理します。

・括弧がある場合は、括弧の内側から処理します。

・負号、べき乗、乗法演算（＊，／）、加法演算（＋，－）の順番で処理します。

・優先度が同じ場合は、式の左側から右側へ向かって処理します。

例）　ａ＝ｂ＋ｃ＊ｄ／（ｅ－ｆ）－ｇ

処理の順番　１．ｅ－ｆを計算します。　ｅ－ｆ

２．ｃ＊ｄを計算します。　ｃ＊ｄ

３．ｃ＊ｄをｅ－ｆで割ります。　（ｃ＊ｄ）／（ｅ－ｆ）

４．ｂに上の結果を加えます。

ｂ＋（ｃ＊ｄ／（ｅ－ｆ））

５．上の結果からｇを引きます。

（ｂ＋ｃ＊ｄ／（ｅ－ｆ））－ｇ

表２．４に演算の優先順位の表を示します。

表２．４　演算の優先順位

優先度	演　　算	演　算　子	結合規則
高　い	括弧	（　）	左から右
	ベクトルの要素の指定	.	左から右
	負号，否定	－、　ＮＯＴ	右から左
	乗算、除算、剰余、べき乗	＊、／、ＭＯＤ、＾	左から右
	加算、減算	＋、－	左から右
	大小判別	＜、＞、＜＝、＞＝、＝＜、＝＞	左から右
	等号、不等号	＝＝、＜＞、＞＜	左から右
	論理積、論理和	ＡＮＤ、ＯＲ	左から右
低　い	代入	＝	右から左

注）　結合規則の説明

左から右　・・・　１＋２－３は（１＋２）－３と解釈，実行する。

右から左　・・・　ＮＯＴ－３はＮＯＴ（－３）と解釈，実行する。

(2)　スカラ型データの演算

スカラ型のデータは、算術演算子と組合わせて演算が可能です。式の中に１つでも実数型のデータがあると、演算結果は実数になります。また、次の各命令語を式の中で使用した場合にも、演算結果は実数になります。

ＳＩＮ，ＣＯＳ，ＴＡＮ，ＡＳＩＮ，ＡＣＯＳ，ＡＴＡＮ，ＡＴＡＮ２，

ＳＱＲＴ，ＲＥＡＬ，ＬＮ，ＬＯＧ１０，ＥＸＰ

左辺の変数のデータ型が整数型の場合には、演算結果は整数に変換されてから変数に代入します。実数から整数へ変換すると小数部は切捨てられます。また、整数から実数への変換を行うと、有効桁数が押えられますので注意してください。

データ型を明確にしたい時には、ＩＮＴ，ＲＥＡＬ命令を使用してください。

文字列に対する演算は行なえません。

ベクトル型の変数の要素をスカラ型のデータと組合わせて演算する事ができます。この場合は、要素指定子をベクトル型変数の後に付けて演算を行う要素を指定します。要素指定子としては、”．X”，”．Y”，”．Z”，”．C”，”．T”の５つが使用できます。また，”．1”，”．2”，”．3”，”．4”，”．5”のように何番めの要素かを数値指定することもできます。

例）　A＝ＰＯＩＮＴ１．Ｘ／２５
　　ＧＡＩＮ＝｛ＧＡＩＮ．１，ＧＡＩＮ．２，０，０，０｝

注）　ベクトル型の変数の要素を指定する時には参照のみ可能です。
　　指定した要素への代入は行えません。

(3) ベクトル型データの演算

ベクトル型のデータの演算では、要素同士の加減算を行うことができます。演算は同じ型の変数同志でのみ可能です。＜姿勢＞の要素は代入される変数の持っている値のままとなります。

例）　Ｐ１　：　（１０，２０，３０，４０，５０，ＲＩＧＨＴＹ）
　　Ｐ２　：　（－５，１０，－１５，２０，－２５，ＬＥＦＴＹ）
　　Ｈ１　：　（１００，５０，－５０，０）
　　Ｈ２　：　（１２，３４，５６，７８）

とした時、以下のプログラムを実行した結果は次の通りです。

　　Ｐ３＝Ｐ１－Ｐ２
　　Ｈ３＝Ｈ１－Ｈ２

実行後の結果

　　Ｐ３　：　（１５，１０，４５，２０，７５，ＲＩＧＨＴＹ）
　　Ｈ３　：　（８８，１６，－１０６，－７８）

注）　上記の例でＰ３の＜姿勢＞の項は不定です。

(4) ベクトル型データへの代入

ベクトル型データの各要素へ、定数、変数、及び演算結果を代入するには、次の方法があります。

(a) スカラ型データの並びをベクトル型データに変換する命令語を用いる。

スカラ型データの並びをベクトル型データに変換する命令語として、ＰＯＩＮＴ命令とＴＲＡＮＳ命令の２つがあります。これらの命令語を利用してベクトル型データへの演算が行なえます。ＰＯＩＮＴ命令は位置型のデータを生成し、ＴＲＡＮＳ命令は座標型のデータを生成します。ここで要素の記述を省略すると、該当する要素は０と見なされます。詳細は第３章を参照してください。

例）　Ｐ１＝ＰＯＩＮＴ（Ｐ２．Ｘ，Ｐ２．Ｙ，Ｐ２．Ｚ＋５０，０，０）

Ｈ１＝Ｈ２＋ＴＲＡＮＳ（１００，１００）

一般に、ベクトル型データの宣言は下記の様に行います。

位置型データ　ＰＯＩＮＴ　（Ｘ，Ｙ，Ｚ，Ｃ，Ｔ，＜姿勢＞）

座標型データ　ＴＲＡＮＳ　（Ｘ，Ｙ，Ｚ，Ｃ）

Ｘ，Ｙ，Ｚ，Ｃ，Ｔ　：　Ｘ，Ｙ，Ｚ，Ｃ，Ｔの各座標値で実数値をとります。
（ｍｍ，ｄｅｇ　単位）

＜姿勢＞　：　０～２の整数値でロボットの姿勢を表します。
０・・・フリー（姿勢未定義）
１・・・左肩系
２・・・右肩系

各要素の記述を省略すると、０と見なされます。

注１）　位置型、座標型データを宣言する場合には、データ型を明確にする為にＰＯＩＮＴ命令、ＴＲＡＮＳ命令を使用するようにしてください。

注２）　未教示の位置デ―タをロボット言語のプログラム中で使用すると、このような位置デ―タはコントロ―ラのメモリに一時的に作成されるため、プログラムがリセット状態になると消えてしまいます。また、このような位置デ―タは、デ―タを使用しているプログラムの中でのみ有効なので、サブプログラム中で使用するためには引数として受渡す必要があります。引数については「2.8.2 サブプログラム」を参照してください。

注３）　配列型データ（変数名［インデックス番号］の形式のもの）に対する代入、参照は配列型データのもとのデータ形式（スカラ型、ベクトル型）と同じ取り扱いになります

2.5.2 論理式

ＳＣＯＬ言語で、論理式はＩＦ、ＷＡＩＴ、ＯＮの各命令語と共に使用されます。関係演算子＜，＞，＜＝（または＝＜），＞＝（または＝＞），＜＞（または＞＜），＝＝の６種類を使用することができます。また、論理式同士を論理演算子（ＡＮＤ、ＯＲ、ＮＯＴ）を使って結合する事も可能です。データとしては、スカラ型の変数、定数及び演算式が指定できます。同値の判断を行う時には、＝でなくて＝＝を使います。実数の比較を行う場合には、０．００１以下の差は同値と見なします。

論理式の演算結果は真（１）か偽（０）になります。演算結果は整数型になります。

例１） ＩＦ Ｋ＝＝Ｋ２＊Ｋ３ ＴＨＥＮ Ｋ＝Ｋ２
ＯＮ ＭＯＴＩＯＮ＞＝５０ ＤＯ ＤＯＵＴ（１，２）

例２） 「ＩＦ Ｊ１ ＴＨＥＮ ＧＯＴＯ ＴＲＵＥ１ ＥＬＳＥ ＧＯＴＯ ＦＡＬＳＥ１」を実行すると、Ｊ１が整数型の０、もしくは実数型で｜Ｊ１｜≦０．００１ならば偽とみなしてＦＡＬＳＥ１へ分岐します。
Ｊ１が整数型の０以外、もしくは実数型で｜Ｊ１｜＞０．００１ならば真とみなしてＴＲＵＥ１へ分岐します。

2.6 ラベル

ＳＣＯＬ言語では、プログラム分岐を行う場合には分岐先をラベルを用いて指定します。分岐先のラベルは文の先頭にて指定します。文にラベルを付ける時は、識別子の後ろにコロン（：）を付けます。ＧＯＴＯ命令にてプログラムの分岐先を指定するときにはラベル名の後にコロンを付けません。

プログラムの分岐は、同一プログラム内のみで行なえます。あるプログラムから別のプログラムへの分岐は行えません。異なるプログラムで同じラベル名を使うことは差し障りありません。しかし、同一プログラムでラベル名を重複して用いることはできません。

例） ＬＯＯＰ１： ＭＯＶＥ Ｐ１
ＧＯＴＯ ＬＯＯＰ１

2.7 注釈

ＳＣＯＬ言語では、プログラムを読み易くするために、プログラム中に注釈文を書くことができます。注釈文としては、ティーチペンダントから入力できる任意の文字を使用することができます。注釈文は、次のいずれかの方法で記述します。

(1) ＲＥＭＡＲＫ文に書く

ＲＥＭＡＲＫ命令に続けて注釈を記述します。ＲＥＭＡＲＫ文は１行の終わり（［EXE］キー入力）まですべて注釈と解釈され、プログラムとしては実行されません。

ＲＥＭＡＲＫ文は単独の命令になるため、他の命令語に続けては書けません。

例） ＲＥＭＡＲＫ ＳＣＯＬ ＳＡＭＰＬＥ ＰＲＯＧＲＡＭ

(2) 記号”´”に続けて書く。

記号”´”に続けて書いた文も注釈文と見なされます。この方法を使うと、他の命令語に続けて注釈を入れることができます。記号”´”の後ろは、１行の終わり（［EXE］キー入力）まですべて注釈と解釈され、プログラムとしては実行されません。

例） ＭＯＶＥ Ｐ１ ´ＭＯＶＥＳ ＴＨＥ ＲＯＢＯＴ ＴＯ Ｐ１

´＊＊＊ ＳＣＯＬ ＣＯＭＭＥＮＴ ＳＡＭＰＬＥ ＊＊＊

2.8 プログラム

本節では、ＳＣＯＬ言語のプログラムについて説明します。

2.8.1 プログラムの宣言

ＳＣＯＬ言語の命令語を組合わせてプログラムを宣言する時は、次の形で記述します。

ＰＲＯＧＲＡＭ ＜プログラム名＞

プログラムの本体

ＥＮＤ

プログラムは、ＰＲＯＧＲＡＭ文からＥＮＤ文までで宣言します。ＰＲＯＧＲＡＭ文には、ＰＲＯＧＲＡＭに引き続いてプログラムの名称を＜プログラム名＞として記述します。＜プログラム名＞は識別子にて表します。プログラムの本体は、ＰＲＯＧＲＡＭ文とＥＮＤ文の間に記述します。

例）　ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ　　プログラム名”ＳＡＭＰＬＥ”
　　ＲＥＭＡＲＫ　ＳＡＭＰＬＥ　　注釈文
　　ＳＰＥＥＤ＝２０　　動作速度を最高速度の20% にする
　　ＭＯＶＥ　Ａ１　　位置Ａ１に移動する
　　ＤＥＬＡＹ　０．５　　０．５秒間待つ
　　ＭＯＶＥ　Ａ２　　位置Ａ２に移動する
　　ＤＥＬＡＹ　０．５　　０．５秒間待つ
　ＥＮＤ　　プログラムの終り

プログラムの本体は、ＳＣＯＬ言語の命令語を組合わせた文により構成されます。１つの文は［EXE］キーの入力までになります。１文の長さは１３０字以内になります。文の中には自由にスペースを入れる事ができます。また、記号”´”を用いれば文の中に注釈を書く事ができます。

注）　命令語、識別子の単語を構成する文字の間にスペースを入れる事はできません。

2.8.2 サブプログラム

サブプログラムは、そのプログラム名を書くだけで呼出す事ができます。

例）　メインプログラム

```
ＰＲＯＧＲＡＭ　ＭＡＩＮ
    ＲＥＭＡＲＫ　＊＊＊　ＳＡＭＰＬＥ　１　＊＊＊
    ＳＵＢ１
ＥＮＤ
```

サブプログラム

```
ＰＲＯＧＲＡＭ　ＳＵＢ１
    ＲＥＭＡＲＫ　＊＊＊　ＳＵＢＰＲＯＧＲＡＭ　Ｎｏ．１　＊＊＊
    サブプログラムの本体
    ＲＥＴＵＲＮ
ＥＮＤ
```

サブプログラムは、ＲＥＴＵＲＮ命令実行にてメインプログラムに戻ります。サブプログラム中にＲＥＴＵＲＮ命令が無いときには、サブプログラムのＥＮＤの前にＲＥＴＵＲＮ命令があるものとして処理されます。
サブプログラムとメインプログラムとの間でデータを引渡すには、引渡すデータを引数として指定する必要があります。サブプログラムにて引数を指定する時は、ＰＲＯＧＲＡＭ文に次のように書きます。

ＰＲＯＧＲＡＭ　＜プログラム名＞　（＜変数名＞）

サブプログラムのプログラム名に続けて、引数を（　）内に指定します。サブプログラム内では、指定した変数名に引数の値が入ります。複数の引数を使用する時は、（　）内に変数名を”，”で区切って記述します。引数の数は最大１０個です。
メインプログラムでは、サブプログラムを呼出す時に、サブプログラムのプログラム名に続けて引渡すデータを（　）内に指定します。（　）内に指定したデータは、サブプログラムの変数名に指定した順番で引渡されます。サブプログラム中でメインプログラムからの引数データに代入を行い、その内容を変更すると、その変数に対応するメインプログラム中の変数の内容も同時に変更されます。

例）　メインプログラム

```
ＰＲＯＧＲＡＭ　ＭＡＩＮ
    ＲＥＭＡＲＫ　＊＊＊　ＳＡＭＰＬＥ　２　＊＊＊
    Ｋ１＝１５
    Ｋ２＝２８
    ＳＵＢ２（Ｋ１，Ｋ２，Ｋ）
    ＰＲＩＮＴ　Ｋ
ＥＮＤ
```

サブプログラム

```
ＰＲＯＧＲＡＭ　ＳＵＢ２（Ｎ１，Ｎ２，Ｎ３）
    ＲＥＭＡＲＫ　＊＊＊　ＳＵＢＰＲＯＧＲＡＭ　ＮＯ．２　＊＊＊
    Ｎ３＝Ｎ１＋Ｎ２
    ＲＥＴＵＲＮ
ＥＮＤ
```

上記の例では、メインプログラムとサブプログラムの間で３つの引数を受け渡しています。メインプログラムのＫ１は、サブプログラムのＮ１に、メインプログラムのＫ２は、サブプログラムのＮ２にそれぞれ引き渡されます。サブプログラムで実行結果を、Ｎ３に代入するとメインプログラムの変数Ｋの値が更新されます。
このプログラムを実行すると、Ｋ１＝１５とＫ２＝２８の値をサブプログラムで加算して、結果Ｋ＝４３をメインプログラムでＰＲＩＮＴ命令によりティーチペンダントに表示します。

サブプログラムを使用する場合には、自分自身をサブプログラムとして呼出す使い方はできません。また、使用しているサブプログラムが同じファイル内に存在しない場合には、エラーとして扱われます。

注１） 引数として、式そのもの、位置データ等のベクトルデータ式の結果やベクトルデータの要素は指定できません。

注２） 引数として定数を使用した場合には、サブプログラムにて対応する変数への代入はできません。

注３） サブプログラムへの引数とする変数は、そのサブプログラムの実行前に何らかの値が代入されていなければなりません。

2.8.3 ライブラリ

ＳＣＯＬ言語でサブプログラムを使用する場合には、サブプログラムはメインプログラムと同じファイルに存在しなければなりません。しかし、サブプログラムをライブラリファイル）に作成すれば、メインプログラムからそのサブプログラムを使用する事ができるようになります。サブプログラムをライブラリファイルに作成するには、通常のプログラム入力と同じ方法で行ってください。プログラムの入力方法については、「操作編」を参照してください。

ライブラリには以下の２種類があります。

① システムライブラリ

プログラム実行時に常に読み込まれるライブラリです。

ファイル名は“ＳＣＯＬ．ＬＩＢ”で、ファイル名の変更は出来ませんが必要に応じて内容を追加、変更することが可能です。

ＯＰＥＮ１，ＣＬＯＳＥ１命令などもこのライブラリファイルに作成されたサブプログラムになっています。ロボットコントローラのシステムに標準として用意しているライブラリファイルの内容を、付録Ｃに示します。

システムライブラリにサブプログラムを新たに作成する場合は、現在のライブラリファイル（ＳＣＯＬ．ＬＩＢ）の一番最後に追加してください。

② ダイナミックリンクライブラリ

必要に応じてユーザーが読み込むライブラリファイルです。

ライブラリファイルの名称は、＿＿＿＿＿．ＬＩＢです。

ダイナミックリンクライブラリは、ユーザープログラムで読み込みを宣言する必要があります。以下のようにユーザープログラムのＧＬＯＢＡＬ部でライブラリファイルの読み込みを宣言します。

```
ＧＬＯＢＡＬ
      ＬＯＡＤＬＩＢ　ＰＡＬＬＥＴ．ＬＩＢ　←ライブラリの読み込み宣言
ＥＮＤ
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ１
   ～～～～～～
     省略
   ～～～～～～
ＥＮＤ
```

ダイナミックリンクライブラリは同時に５つまで読み込めます。

ＳＣＯＬ命令の中にもこのダイナミックリンクライブラリを用いているものがあります。その命令語を用いる場合にも、ＬＯＡＤＬＩＢ　ファイル名　命令で読み込みを宣言する必要があります。

ダイナミックリンクライブラリを使ったプログラムを実行する場合、コントローラはＳＣＯＬＬＩＢ．ＴＭＰ（ＳＣＯＬ．ＬＩＢ＋ダイナミックリンクライブラリ）という名称のテンポラリファイルを作成します。ユーザープログラム領域に空が無い場合には実行できない場合があります。

注意

ライブラリファイルに作成してあるサブプログラムと同一名称のサブプログラムが、選択したファイルに存在すると、自動運転時には、同一ファイル上にあるサブプログラムの方を実行します。

ダイナミックリンクライブラリー機能の動作

ダイナミックリンクライブラリーを必要とするプログラム（ＬＯＡＤＬＩＢ命令がある）をＳＥＬＥＣＴした場合、システムは以下の動作を行います。

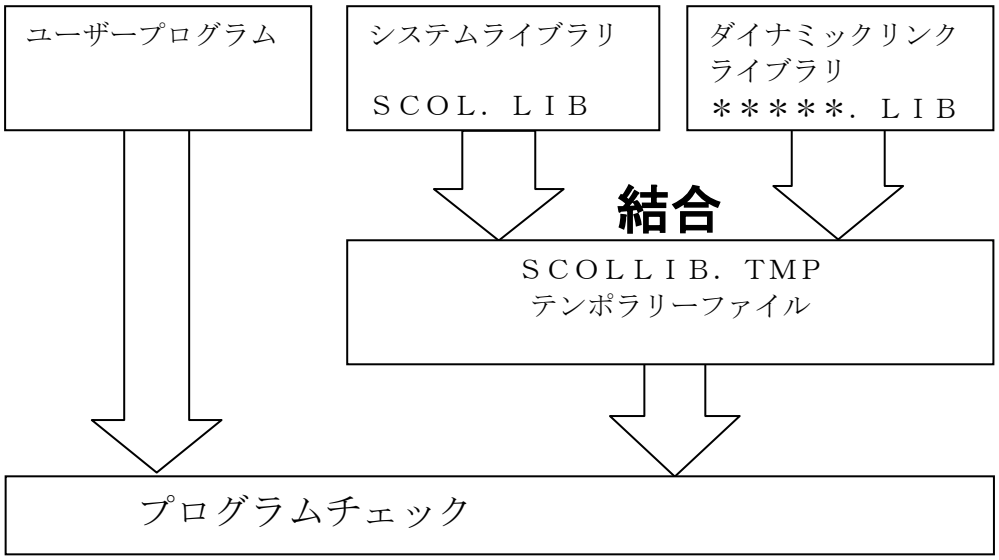
① ユーザープログラムを開く

② LOADLIB命令をチェックする（ファイル名称の確認）

③ SCOL.LIBとダイナミックリンクライブラリーを結合しSCOLLIB.TMPを作成する

④ プログラムのチェックを行い、問題が無ければSELECT完了

⑤ SCOLLIB.TMPは正常にSELECT完了した時点で自動的に削除します。
（ERRORが発生した場合にはSCOLLIB.TMPは削除されません）



何らかの原因でライブラリーにＥＲＲＯＲが発生した場合、システムは

“ＬＩＮＥ？？？：ＬＩＢ＞ＥＲＲＯＲ－＊＊＊”

と表示します。このＬＩＮＥ？？？で示される行数はＳＣＯＬＬＩＢ.ＴＭＰの行数を示します。

ＳＣＯＬＬＩＢ．ＴＭＰで内容を確認し、ライブラリーファイルを修正してください。

2.8.4 マルチタスク処理

本章ではＳＣＯＬ言語のマルチタスク機能について、関係する命令語やシステム変数とともに使い方を説明します。

図１、２にシングルタスク動作とマルチタスク動作のプログラム実行の様子を示します。図の番号はプログラムが実行される順番を示してあります。プログラムからプログラムへの切り替わり（タスク遷移）が具体的にどのようなタイミングで発生するかは後述します。

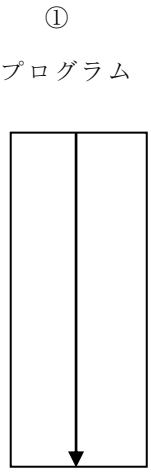


図1 シングルタスク動作

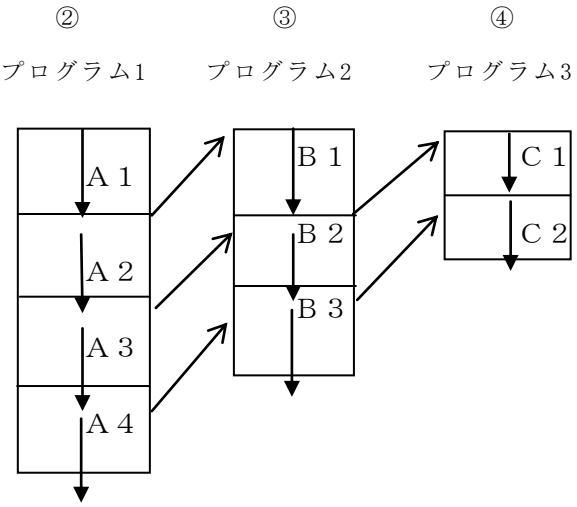


図2 マルチタスク動作

図１ではプログラムAの部分はプログラムの最初から最後まで連続的に実行されます。（シングルタスク動作、サブルーチンコールなしの場合）

マルチタスク命令が使用されていないプログラムはこの図１で示されるような動作となります。（サブルーチンコールなしの場合）

図２にマルチタスク命令が使用された場合のプログラム実行の様子を示します。

図２のようにマルチタスク動作は独立した複数のプログラムを時分割で切り替えながらあたかもプログラムが並列で動作しているように実現されています。下表にプログラムの実行順序を示します。

動作順序	動作するプログラム	
1	A1	プログラム１開始
2	B1	プログラム２開始
3	C1	プログラム３開始
4	A2	
5	B2	
6	C2	プログラム３が１サイクル終了
7	A3	
8	B3	プログラム２が1サイクル終了
9	C1	プログラム３開始
１０	A4	プログラム１が１サイクル終了
１１	B1	プログラム２開始
１２	C2	
１３	A1	プログラム１開始
:	:	

次にマルチタスクの起動について説明します。

マルチタスクとして扱うことのできるプログラムは引数のないプログラムブロックです。プログラムブロックとはＰＲＯＧＲＡＭ文とＥＮＤ文で囲まれたＳＣＯＬ言語のプログラム並びを言います。したがって引数のないサブルーチンはタスクとして扱うことができます。タスクには引数を渡すことはできません。

あるプログラムをタスクとして扱うには命令語“ＴＡＳＫ”を使用します。ＴＡＳＫは引数で示されるプログラムをタスクとして動作させる命令です。ＴＡＳＫ命令で起動されないかぎりプログラムはタスクとして動作しません。

ただし、プログラムファイルの先頭に記述されているプログラムブロック（ＰＲＯＧＲＡＭ～ＥＮＤまでの命令語並び）は例外で、ＴＡＳＫ命令無しでもタスクとして動作します。

図２でプログラム２をタスクとして動作させるためにはプログラム１の中でＴＡＳＫ(“ＰＲＯＧ２”)を実行する必要があります。（プログラム１はファイルの先頭に記述されており、ＴＡＳＫ命令無しでもタスクとして起動しているものとします）

またプログラム３をタスクとして動作させるためにはすでに起動しているタスクの中（この例ではプログラム１または２の中）で新たにＴＡＳＫ(“ＰＲＯＧ３”)を実行する必要があります。

一度起動したタスク、プログラムがリセットされるか、ＳＣＯＬ言語で明示的にタスク動作解除の命令が実行されるまで、タスクとして存在しつづけます。

次にタスク固有の番号(タスクＩＤ)について説明します。

ＴＡＳＫ命令により起動されて動作しているタスクにはそれぞれ固有の番号(タスクＩＤ)がついています。図２の例で説明すると、プログラム１は“１”、プログラム２は“２”、プログラム３は“３”です。このタスクIDは１から順番に始まりタスクが起動されるたび（つまりＴＡＳＫが実行されるたび）に１ずつ増えてゆきます。ＳＣＯＬ言語でタスクを管理する場合このタスクＩＤを使います。

タスクＩＤを得る方法は以下の例を参照して下さい。

例１：Ｉ１＝ＴＡＳＫ(“ＰＲＯＧ２”)

Ｉ１は整数型の任意の変数。ＰＲＯＧ２のタスクＩＤを得ることができます。ただし、この命令はプログラム１の中で実行されているので、この例ではプログラム２では自分のタスクＩＤを参照することはできません。

例２：Ｉ２＝ＴＩＤ

Ｉ２は整数型の任意の変数。システム変数ＴＩＤを参照すると自分のタスクＩＤを得ることができます。プログラム２の中でこの命令を実行すると、プログラム２の中で自分のタスクＩＤ(この場合は２)を参照することができます。

プログラム１の中でこの命令を実行すると、Ｉ２にはプログラム１のタスクＩＤ(この場合は１)が代入されます。

ほかのタスクから自分以外のタスクのタスクＩＤを参照する場合には、例１、２の変数をグローバル変数として定義しておくことで可能となります。

次にタスクの切り替えについて説明します。

図２に示されるように、システムは時分割でプログラム１から３を実行します。このときプログラムが切り替わるタイミングは次の３つの条件です。

（１） ＳＣＯＬ命令語ＳＷＩＴＣＨによって明示的にプログラム切り替えが指示された場合。

ＳＷＩＴＣＨ命令はＳＣＯＬ言語にてユーザーが明示的にタスク切り替えを行う場合に使用します。ＳＷＩＴＣＨ命令を使用するとシステムで定められたタスク切り替え条件が成立していない場合でも、タスクを切り替えることができます。

（２） ＳＣＯＬ命令語ＴＡＳＫによって新しいタスクが起動された場合。

ＴＡＳＫによって新しいタスクが起動された場合、プログラムの制御は新しく起動されたタスクに切り替わります。

（３） ＳＣＯＬ命令語ＫＩＬＬによってタスクが終了した場合。

ＫＩＬＬによってタスクを終了させたような場合にはプログラムの制御は次のタスクに切り替わります。

（４） システムで定められた特定の条件が成立して、システム側でプログラムの切り替えを行った場合。

システム側で定められたタスク切り替えの条件は以下の通りです。

（１） 一つのタスクでプログラムが１００ｍｓｅｃ以上実行されたとき。

（２） 移動命令用のデータエリアがフルの状態になったとき。移動命令は最大４つのデータを先読み可能ですが、この先読みのための内部エリアがＦＵＬＬになった場合にタスク切り替えが行われます。

（３） 外部機器との通信が必要な命令を実行したとき。

ＩＮＰＵＴ，ＰＲＩＮＴ，ＲＥＳＴＯＲＥ命令は、ＳＣＯＬプログラム実行部が単独で実行する訳ではなく、例えばオペレータのＴＰ操作やＲＡＭファイル操作という実行完了に時間のかかる命令です。このためＳＣＯＬプログラム実行部は何らかの応答待ちとタスク切替えが行われます。なお、システムによるタスク切り替えを抑止したい場合にはシステム変数ＳＷＩＴＣＨをＤＩＳＡＢＬＥに設定します。

（注） ステップ実行中、あるいはタスク切り替えを抑止した場合には現在実行しているプログラムのみが実行され、すでに起動している別のタスクのプログラムは動作しません。（シングルタスク動作になります）

マルチタスクプログラム作成上の注意

（１） 動作命令はメインタスクでのみ使用可能です。
サブタスクで動作命令を使用した場合はエラーになります。
（動作命令：ＭＯＶＥＩ，ＭＯＶＥＡ，ＭＯＶＥ，ＭＯＶＥＳ，ＭＯＶＥＣ，
ＭＯＶＥＪ，ＤＥＬＡＹ）

（２） 次のシステム変数はタスクごとに独立して持っていますので、それぞれのタスクで自由に設定、あるいは参照できます。ＮＯＷＡＩＴについてはユーザパラメータ[U02]により共通に持つか独立で持つかの切り替えをすることが出来ます。

ＴＩＭＥＲ，ＴＩＤ，ＮＯＷＡＩＴ

ＮＯＷＡＩＴは独立で使用し、サブタスクはＥＮＢＡＬＥ ＮＯＷＡＩＴで使用することをお勧めします。次のシステム変数はタスク間で共通であり、あるタスクで設定した値は他のタスクでも有効となります。したがって、複数のタスクで値を設定する場合、注意が必要です。

ＣＯＮＦＩＧ，ＡＣＣＵＲ，ＡＣＣＥＬ，ＤＥＣＥＬ，ＳＰＥＥＤ，ＰＡＳＳ，
ＴＯＲＱＵＥ，ＧＡＩＮ，ＴＯＯＬ，ＢＡＳＥ，ＷＯＲＫ，ＰＡＹＬＯＡＤ，
ＮＯＷＡＩＴ

（３） メインタスクで動作命令実行中、サブタスクで次のような動作の完了を待つ命令を実行すると、そのタスクはメインタスクで実行中の動作命令の完了まで待機状態となります。

１）ＤＩＳＡＢＬＥ ＮＯＷＡＩＴモードでの入出力命令の実行。
（入出力命令：ＩＮＰＵＴ，ＰＲＩＮＴ，ＤＩＮ，ＤＯＵＴ，ＢＣＤＩＮ，
ＢＣＤＯＵＴ，ＰＯＵＴ）
２）ＷＡＩＴ ＭＯＴＩＯＮ命令の実行

（４） あるタスクのＩＮＰＵＴ，ＰＲＩＮＴ命令実行中に、他タスクが同一命令を実行した場合、別の通信チャネルに対する命令であっても、そのタスクは先に実行中になったＩＮＰＵＴ，ＰＲＩＮＴ命令が終了するまで待機状態になります。

2.8.5 グローバル変数定義

プログラム全域で参照可能なグローバル変数を定義する場合、次の規則に従って記述します。

（１） グローバル変数宣言

グローバル変数を使用する場合、使用する変数の型と識別子（変数名）を定義する必要があります。

この定義は最初のＰＲＯＧＲＡＭ文より前で行われなければなりません。

たとえば、実数型の変数A、整数型の変数Bを定義する場合には以下のように宣言します。

```
GLOBAL
A=1.0(この値が変数の初期値となります)
B=２
END

PROGRAM
：
END
```

（２） 各型別グローバル変数宣言の方法

各型のグローバル変数を定義する際には以下のような記述にします。

整数型： A=１

実数型： B=1.0

配列型： DIM D(10) AS INT 10個の整数型の配列を定義 （注１）

DIM E(10,3) AS REAL 10×３の実数型の配列を定義

DIM F(5) AS POINT ５個の位置型の配列を定義

なお予約語ＧＬＯＢＡＬ～ＥＮＤで囲まれたグローバルブロックではスカラ型変数と配列型変数のみ記述しベクトル型変数は、データエディタで編集する予約語ＤＡＴＡ～ＥＮＤで囲まれたデータブロックに記述してください。

（３） 配列変数の初期値設定

配列以外のグローバル変数と同様にスカラ型配列変数の初期値はグローバルブロックでベクトル型配列変数の初期値はデータブロックに記述してください。

注１） 明示的に初期値が設定されていない配列型のグローバル変数の初期値は不定です。ユーザープログラムで初期化する必要があります。

2.8.6 配列型変数

スカラ型変数、ベクトル型変数は、一つの名称が一つのデータを表しています。しかし一つの名称が複数のデータを表すことが出来れば、プログラムが簡単になる場合があります。このため、配列変数を使用することが出来ます。

配列変数を扱うためには、ＤＩＭ命令により型と全体の要素数を決定しておかなければなりません。同一名称の配列変数は、個々の要素が異なる型を持つことは出来ません。
詳細はＤＩＭ命令の説明を参照ください。

ＤＩＭ命令により配列であることが宣言された変数は、プログラム内に以下のフォーマットで記述します。

変数名 (＜要素＞ [,＜要素＞] …)

変数名の後ろの要素番号は、１からＤＩＭ命令で指定する番号までの任意の値を指定できますが、変数名＋“(”＋要素＋“)”までが２０文字を超えると実行エラーになりますので注意してください。

ＳＣＯＬプログラム実行部では、変数名と、要素番号を元に複数個のデータ範囲から特定の一個を選択します。

例えば５×５個の計２５個の位置をＰという名称の配列変数として扱い、それぞれの位置に順番に移動するプログラムは、次のようになります。

```
GLOBAL
  DIM P'(5,5) AS POINT
END
PROGRAM SAMPLE
FOR I = 1 TO 5
  FOR J = 1 TO 5
      MOVE P(I,J)
  NEXT  J
NEXT I
END
DATA
 POINT P(1,1) = 650, 0, 100, 0, 0 / LEFTY
 POINT P(1,2) = 650, 0, 100, 0, 0 / LEFTY
                  ・
                  ・
                  ・
 POINT P(5,5) = 650, 0, 100, 0, 0 / LEFTY
END
```

ベクトル型配列変数のＸ要素やＹ要素を直接参照や代入することは出来ません。上記配列変数Ｐに対して、

```
PRINT P(1,1).X,CR
```

という命令を実行することは出来ません。この場合は、通常のベクトル型変数に値を複写した上で実行してください。

```
PP=P(1,1)
PRINT PP.X
```

また、誤ってＩ＝５、Ｊ＝６という状態でＰ（Ｉ，Ｊ）という位置に移動する命令を実行しようとすると、ＤＩＭ命令で指定された要素外への参照や代入はできないため、実行時にエラーとなります。なお配列の添え字(インデックス)の型は、整数型に限ります。実数型、ベクトル型のデータを使用すると、実行時にエラーとなります。

第３章

命令語の説明

本章ではＳＣＯＬ言語の個々の命令語について、その機能と使い方を説明します。
命令語はアルファベット順に説明しています。

3.1 命令語の記載書式

本章で命令語は、以下の書式で説明しています。

(1) 機能

命令語の機能を簡単に説明します。

(2) フォーマット

命令語の記載フォーマットについて説明します。説明中に使用している記号は、次の意味を持っています。

［ ］　括弧の中身が省略可能であることを示します。必要に応じて指定します。
＜ ＞　記載するデータの内容を示します。
｛ ｝　括弧内のデータの内、１つを選択使用します。
・・・　複数のデータを指定できることを示します。

上記の各記号は、説明の便宜上付けたもので、実際のプログラムには記入しないでください。
特に、上記の記号を使用する必要があるときには、その旨説明します。

(3) 文例

命令語を使用した文の例を示します。

(4) 解説・注意

命令語の解説と命令語を使用する上での注意事項、制約事項について説明します。

(5) サンプルプログラム

命令語を用いたプログラムの例を示して説明します。

また、命令語のフォーマット中で使用するデータの意味は、次の通りです。データとして式を用いる事もできます。

＜位置＞　位置型のデータを指定します。
＜軸＞　関節制御軸を指定します。データは整数型で１～５の範囲です。

＜絶対位置＞　各軸の絶対位置を指定します。（０．００１ｍｍもしくは０．００１度単位。）変数、式も使用可能です。

＜相対位置＞　各軸の移動量を相対位置で指定します。（０．００１ｍｍもしくは０．００１度単位。）　変数、式も使用可能です。

＜時間＞　０．０１秒単位で時間を指定します。変数、式も使用可能です。

＜論理式＞　論理式を指定します。

＜文＞　実行する文を指定します。通常のＳＣＯＬ言語の文であれば何を指定してもかまいません。

＜監視条件＞　“ＯＮ”文で監視する条件を指定します。

＜ラベル＞　プログラム分岐を行うラベルを指定します。

＜注釈＞　プログラムに付ける注釈文を示します。

＜変数＞　変数を指定します。

＜式＞　演算式を指定します。演算式には単独の変数も含みます。

＜信号名＞　入出力に用いる信号名を指定します。信号名は整数で与えられます。正の値は信号がオンの状態を、負の値は信号がオフの状態を示します。変数、式も使用可能です。

＜質量＞　ロボットの手先にかかる負荷の質量を指定します。

＜重心オフセット＞　ロボットの手先にかかる負荷の重心が、手先のツール中心からどれだけ離れているかの距離を指定します。

＜姿勢＞　ロボットの姿勢を整数値で指定します。０は未定義，１は左肩系，２は右肩系になります。

＜スイッチ＞　システムスイッチを指定します。システムスイッチとしては、次の２種類があります。

ＰＡＳＳ　ショートカット動作を指定するシステムスイッチです。

ＮＯＷＡＩＴ　動作命令の位置決め完了を待たずに、信号の入出力を行う事を指定するシステムスイッチです。

＜状態＞　ＲＥＳＥＴ命令でリセットする対象を指定します。

表３．１に各命令語を機能別に示します。

表 3.1　ＳＣＯＬ言語一覧

分　類	命　　令　　語	機　　能
動作制御命令	ＢＲＥＡＫ	動作の即時中断
	ＣＬＯＳＥ１，ＣＬＯＳＥ２	動作完了にてハンドを閉じる
	ＣＬＯＳＥＩ１，	ハンドを閉じる
	ＣＬＯＳＥＩ２	ハンドを閉じる
	ＤＥＬＡＹ	指定時間の停止
	ＭＯＶＥ	同期動作
	ＭＯＶＥＳ	直線補間動作
	ＭＯＶＥＣ	円弧補間動作
	ＭＯＶＥＡ	絶対単軸動作
	ＭＯＶＥＩ	相対単軸動作
	ＭＯＶＥＪ	アーチ動作
	ＯＰＥＮ１，ＯＰＥＮ２	動作完了にてハンドを開く
	ＯＰＥＮＩ１，ＯＰＥＮＩ２	ハンドを開く
	ＰＡＵＳＥ	動作の中断
	ＲＥＡＤＹ	機械原点への移動
	ＲＥＳＵＭＥ	中断した動作の再開
プログラム制御命令	ＦＯＲ～ＴＯ～ＳＴＥＰ～	動作の繰返し
	ＧＯＴＯ	無条件分岐
	ＧＯＴＯ（）	式の値による分岐
	ＩＧＮＯＲＥ	監視の解除
	ＩＦ～ＴＨＥＮ～ＥＬＳＥ～	条件判断
	ＮＥＸＴ	動作の繰返し
	ＯＮ～ＤＯ～	条件監視の登録
	ＰＲＯＧＲＡＭ	プログラム開始
	ＲＣＹＣＬＥ	サイクルリセット用ラベル
	ＲＥＴＵＲＮ	メインへ戻る
	ＳＴＯＰ	プログラム停止
	ＷＡＩＴ	条件成立を待つ

分　類	命　　令　　語	機　　能
プログラム制御命令	ＥＮＤ	プログラム終了
	ＫＩＬＬ	タスク停止
	ＭＡＸＴＡＳＫ	最大タスク数
	ＲＥＭＡＲＫ	注釈文
	ＳＷＩＴＣＨ	タスク切り替え
	ＴＡＳＫ	タスク起動
	ＴＩＤ	タスクＩＤ
入出力制御命令	ＢＣＤＩＮ	信号のＢＣＤ入力
	ＢＣＤＯＵＴ	信号のＢＣＤ出力
	ＣＲ	ＣＲコード出力
	ＤＩＮ	入力信号の読込み
	ＤＯＵＴ	信号の出力
	ＨＥＸＩＮ	信号のＨＥＸ入力
	ＨＥＸＯＵＴ	信号のＨＥＸ出力
	ＰＵＬＯＵＴ	信号のパルス出力
	ＲＥＳＥＴ	コントローラ状態のリセット
	ＰＲＩＮＴ	通信データ出力
	ＩＮＰＵＴ	通信データ入力
動作条件命令	ＡＣＣＥＬ	加速時の加速度
	ＡＣＣＵＲ	位置決め精度
	ＣＯＮＦＩＧ	姿勢
	ＤＥＣＥＬ	減速時の加速度
	ＤＩＳＡＢＬＥ	システムスイッチ切
	ＥＮＡＢＬＥ	システムスイッチ入
	ＦＲＥＥＬＯＡＤ	負荷データ解除
	ＧＡＩＮ	各軸ゲイン
	ＮＯＷＡＩＴ	動作の位置決め完了を待たない
	ＯＶＥＲＲＩＤＥ	速度オーバーライド
	ＰＡＳＳ	ショートカット動作のパラメータ
	ＰＡＹＬＯＡＤ	負荷データ設定
	ＳＭＯＯＴＨ（オプション）	スムース動作
	ＳＰＥＥＤ	速度
	ＭＯＶＥＳＹＮＣ	動作命令同期/非同期モードを指定
	ＳＷＩＴＣＨ	タスク切替えを禁止／許可
	ＳＬＯＷＤＯＷＮ	スローダウン
	ＳＬＷＳＰＤ	スローダウン速度
	ＷＩＴＨ	動作条件の指定

分　類	命　　令　　語	機　　能
算術演算	ＣＯＳ	余弦
命令	ＳＩＮ	正弦
	ＴＡＮ	正接
	ＡＢＳ	絶対値
	ＡＣＯＳ	逆余弦
	ＡＮＤ	論理積
	ＡＳＩＮ	逆正弦
	ＡＴＡＮ	逆正接
	ＡＴＡＮ２	逆正接
	ＤＥＳＴ	目標位置
	ＥＸＰ	ｅのべき乗
	ＨＥＲＥ	現在位置
	ＩＮＴ	整数化
	ＬＮ	自然対数
	ＬＯＧ１０	常用対数
	ＭＯＤ	剰余
	ＮＯＴ	否定
	ＯＲ	論理和
	ＰＯＩＮＴ	位置型データの生成
	ＲＥＡＬ	実数化
	ＳＧＮ	符号の取出し
	ＳＱＲＴ	平方根
	ＴＲＡＮＳ	座標型データの生成
動作状態	ＢＡＳＥ	ベース座標系
命令	ＭＯＤＥ	システムの運転モード
	ＭＯＴＩＯＮ	動作の実行量
	ＭＯＴＩＯＮＴ	動作の実行時間
	ＲＥＭＡＩＮ	動作の残量
	ＲＥＭＡＩＮＴ	動作の残り時間
	ＴＩＭＥＲ	タイマー
	ＴＯＯＬ	ツール座標系
	ＷＯＲＫ	ワーク座標系
データ定義	ＤＡＴＡ	データ定義開始
命令	ＤＩＭ～ＡＳ	配列変数定義
	ＧＬＯＢＡＬ	大域的変数定義
	ＲＥＳＴＯＲＥ	グローバル変数初期値をファイルに退避
	ＳＡＶＥＥＮＤ	電源遮断時データを保存

分　類	命　　令　　語	機　　能
パレタイズ	ＩＮＩＴＰＬＴ	パレットの初期化
命令	ＭＯＶＥＰＬＴ	パレットの指定位置に移動
位置データ	ＬＡＴＣＨ	位置ラッチ機能の有効／無効
ラッチ機能	ＬＡＴＣＨＴＲＧ１～８	検出エッジ方向
（ＴＳ３０００	ＬＡＴＣＨＳＩＧ１～８	信号状態
オプション）	ＬＡＴＣＨＰＳＮ１～８	ラッチした位置
システム	ＣＯＡＲＳＥ	位置決め粗
定数	ＣＯＭ０，ＴＰ	通信チャンネル（ﾃｨｰﾁﾍﾟﾝﾀﾞﾝﾄ）
	ＣＯＭ１	通信チャンネル１
	ＣＯＮＴ	連続運転モード
	ＣＹＣＬＥ	サイクル運転モード
	ＦＩＮＥ	位置決め精
	ＯＦＦ	各軸ゲインのオフ
	ＯＮ	各軸ゲインのオン
	ＰＡＩ	円周率
	ＳＥＧＭＥＮＴ	セグメント運転モード
簡易ＰＬＣ	ＰＬＣＤＡＴＡＲ１～８	簡易ＰＬＣ　Ｉ／Ｆ
	ＰＬＣＤＡＴＡＷ１～８	簡易ＰＬＣ　Ｉ／Ｆ
特殊変数	ＳＡＶＥＦ１～４	実数型変数（バックアップ）
	ＳＡＶＥＩ１～４	整数型変数（バックアップ）
記　号	^	べき乗
	－	負号
	＊，／	乗算、除算
	＋，－	加算、減算
	＝	代入
	＝＝	等しい
	＜＞，＞＜	等しくない
	＜	小さい
	＞	大きい
	＜＝，＝＜	小さいか等しい
	＞＝、＝＞	大きいか等しい
	’	注釈文
	.	ベクトルの要素の指定

3.2　命令語の説明

次ページ以降に、各命令語をアルファベット順に説明していきます。

ＡＢＳ

機能	数値の絶対値を与えます。
フォーマット	ＡＢＳ（＜式＞）
文例	ＡＫ＝ＡＢＳ（－２０．３４５） Ｋ　＝ＡＢＳ（Ｋ１） Ｊ１＝Ｋ－ＡＢＳ（Ｎ－２８．５）
解説・注意	（　）内の式の演算結果の絶対値を与えます。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
ABSSAMPLE(3,5,K)
PRINT TP,K,CR
END
PROGRAM  ABSSAMPLE(K1,K2,K)
K=ABS(K1-K2)          引数Ｋ１，Ｋ２の差を計算してその結果を引数Ｋとしてメイン
RETURN                プログラムに返します。
END
```

ＡＣＣＥＬ

機能

ロボットの加速時の加速度を指定します。

フォーマット

ＡＣＣＥＬ＝＜式＞

文例

ＡＣＣＥＬ＝８０
ＡＣＣＥＬ＝０．８＊ＡＣＣＥＬ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＡＣＣＥＬ＝９０

解説・注意

ロボットの加速時の加速度を指定するシステム変数です。
加速度は、標準の加速度に対するパーセンテージで指定します。
ＳＣＯＬ言語では、加速度を加速時と減速時で別々に指定します。減速時の加速度の指定には、ＤＥＣＥＬ命令を使用します。
ロボットに重量物を持たせる時、必要によって加速度を下げたい場合に使用します。この様な場合には、加速時の加速度に併せて減速時の加速度も変更してください。負荷に応じた加速度の設定値は「据付け・輸送編」を参照してください。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。
ロボットには、それぞれ加速度の上限が設定されています。上限値以上の数値を指定しても、加速度は上限値で抑えられます。
０以下の値は１と見なします。
本システム変数を参照すると、現在の加速時の加速度が参照できます 。
加速度の初期値は１００％になっています。

サンプルプログラム

```
PROGRAM  ACCELSAMPL
 FOR K=1 TO 100   加速度を１％から１００％まで、１％きざみで変えていきます。
ACCEL=K
DECEL=K
MOVE A1
MOVE A2
NEXT K
END
```

ＡＣＣＵＲ

機能

ロボットの位置決め精度を指定します。

フォーマット

ＡＣＣＵＲ＝＜式＞

文例

```
ＡＣＣＵＲ＝１
Ｎ＝ＡＣＣＵＲ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＡＣＣＵＲ＝ＣＯＡＲＳＥ
```

解説・注意

ロボットの位置決め精度を指定するシステム変数です。
位置決め精度は０で粗、１で精となります。
位置決め精度を粗にすれば、ロボットの位置決め完了を待たずに、次の命令の実行に移ります。正確な位置決めを待つ必要のない動作での位置決め精度を粗に設定すれば、ロボットのタクトタイムを短縮できます。
位置決め精度の指定には、システム定数のＦＩＮＥ，ＣＯＡＲＳＥを使用する事ができます。位置決め精度は、ＡＣＣＵＲ＝ＦＩＮＥで精に、ＡＣＣＵＲ＝ＣＯＡＲＳＥで粗に設定できます。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。
位置決め精度として、０以下，１以上を指定すると、それぞれ０，１が指定されたものとします。
本システム変数を参照すると､現在の位置決め精度が参照できます｡位置決め精度は、ＡＣＣＵＲの値が０で粗、１で精となります。
位置決め精度の初期値は、精になっています。

サンプルプログラム

```
PROGRAM  ACCURSMPL
 ACCUR=COARSE              Ａ３への移動のみ位置決め精度を精にして動作し、
 MOVE A1                   その他の動作では位置決め精度を粗にしてタクト
 MOVE A2                   タイムを早めます。
 MOVE A3 WITH ACCUR=FINE
 MOVE A4
 END
```

ＡＣＯＳ

機能	逆余弦の値を計算します。
フォーマット	ＡＣＯＳ（＜式＞）
文例	Ｋ＝ＡＣＯＳ（０．５７７） Ｊ１＝９０－ＡＣＯＳ（Ｘ／Ｌ）
解説・注意	（ ）内の逆余弦の値を計算します。 計算結果は度単位で与えられます。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM MAIN
ACOS2(2.0,1.0,K)
PRINT TP,K,CR
END
PROGRAM  ACOS2(L,X,K)
K=ACOS(X/L)          引数Ｌ、Ｘの値からＸ／Ｌの逆余弦を計算して、その結果を
RETURN               引数としてメインプログラムに返します。
END
```

ＡＮＤ

機能	論理積を計算します。
フォーマット	＜論理式＞ＡＮＤ＜論理式＞
文例	ＩＦ　ＤＩＮ（１）　ＡＮＤ　Ｋ＜＝３　ＴＨＥＮ　Ｊ＝０ ＷＡＩＴ　ＤＩＮ（５）　ＡＮＤ　ＴＩＭＥＲ＝＝０
解説・注意	ＡＮＤの右辺と左辺の論理式の論理積を計算します。 ＡＮＤの両辺の論理式が共に真ならば、結果が真になります。 本命令は論理式の中で使用します。
サンプル プログラム	PROGRAM ANDSAMPLE FOR K=1 TO 50 IF K==50 AND DIN(1) THEN J=1 ELSE J=0 PRINT TP,J,CR NEXT K END

ＡＳＩＮ

機能	逆正弦の値を計算します。
フォーマット	ＡＳＩＮ（＜式＞）
文例	Ｋ＝ＡＳＩＮ（０．５７７） Ｊ１＝９０－ＡＳＩＮ（Ｙ／Ｌ）
解説・注意	（　）内の逆正弦の値を計算します。 計算結果は度単位で与えられます。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM MAIN
ASIN2(5.0,2.0,K)
PRINT TP,K,CR
END
PROGRAM  ASIN2(L,Y,K)
K=ASIN(Y/L)          引数Ｌ、Ｙの値からＹ／Ｌの逆正弦を計算して、その結果
RETURN               を引数Ｋとしてメインプログラムに返します。
END
```

ＡＴＡＮ／ＡＴＡＮ２

機能

逆正接の値を計算します。

フォーマット

ＡＴＡＮ（＜式＞）
ＡＴＡＮ２（＜式＞，＜式＞）

文例

Ｋ＝ＡＴＡＮ（０．５７７）
Ｊ１＝９０－ＡＴＡＮ（Ｙ／Ｘ）
Ｎ＝ＡＴＡＮ２（０．３，０．５）
Ｄ＝ＡＴＡＮ２（１００／Ｋ，５０／Ｊ）
Ｌ３＝ＡＢＳ（１８０－ＡＴＡＮ２（Ａ１．Ｙ，Ａ１．Ｘ））

解説・注意

（　）内の逆正接の値を計算します。
ＡＴＡＮ２では、（　）内の前式の値を後式の値で割った結果の逆正接の値を計算します。
計算結果は度単位で与えられます。（なおＡＴＡＮ２（０，０）の実行結果はエラーにならず０になりますのでご注意ください。）
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM MAIN
ATANSAMPLE(5.0,3.0,K)
PRINT TP,K,CR
END
PROGRAM  ATANSAMPLE(X,Y,K)
K=ATAN(Y/X)          引数Ｘ、Ｙの値からＹ／Ｘの逆正接を計算して、その結果を
RETURN               引数Ｋとしてメインプログラムに返します。
END
```

```
PROGRAM MAIN
ATAN2SMPL(K)
PRINT TP,K,CR
END
PROGRAM  ATAN2SMPL(K)
K=ATAN2(A1.Y,A1.X)     教示点Ａ１のＸ、Ｙ成分からＹ／Ｘの
RETURN                 逆正接を計算して、その結果を引数Ｋとし
END                    てメインプログラムに返します。
```

ＢＡＳＥ

機能

ベース座標系を指定します。

フォーマット

ＢＡＳＥ

文例

```
ＢＡＳＥ＝ＴＲＡＮＳ（０，０，０，０）
ＢＡＳＥ１＝ＢＡＳＥ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＢＡＳＥ＝ＢＡＳＥ＋
                              ＴＲＡＮＳ（，，１００）
```

解説・注意

ベース座標系を指定するシステム変数です。
システム変数ＢＡＳＥは、通常の座標型データとして扱えます。
本システム変数を参照すると、現在のベース座標系の値が参照できます。
次のようにして、ベース座標系に直接座標値を指定する事ができます。

ＢＡＳＥ＝ＴＲＡＮＳ（Ｘ，Ｙ，Ｚ，Ｃ）
ＢＡＳＥ＝｛Ｘ，Ｙ，Ｚ，Ｃ｝

データ型を明確にする為に、ＴＲＡＮＳ命令を使用するようにしてください。
Ｘ，Ｙ，Ｚ，Ｃ　：Ｘ，Ｙ，Ｚ，Ｃの各座標値で実数値をとります。
（ｍｍ，ｄｅｇ　単位）
ベース座標系は、ワールド座標系のそれぞれの座標軸に沿って、Ｘ，Ｙ，Ｚの量だけ平行移動した座標軸を、新しいＺ軸の回りにＣだけ回転した座標系になります。
本命令は式の中で使用します。
プログラム中でベース座標系を変更すると、以降の全動作で教示した位置がずれてしまいますので、このような使い方はしないでください。

サンプルプログラム

```
PROGRAM  BASESAMPLE
MOVE A1
MOVE A2
BASE=BASE+TRANS(,,200)    ＢＡＳＥ命令によりベース座標系のＺ軸を２００ｍｍ
MOVE A1                   ずらして、以降の動作では教示した位置より２００ｍｍ
MOVE A2                   下方の点に動作します。
BASE=TRANS()
END
```

ＢＣＤＩＮ

機能

入力信号をＢＣＤコードとして読込みます。

フォーマット

ＢＣＤＩＮ（＜信号名＞，＜信号長＞）

文例

Ｋ＝ＢＣＤＩＮ（１，２）
Ｊ２＝ＢＣＤＩＮ（Ｎ，Ｎ＋２）
ＧＯＴＯ（ＢＣＤＩＮ（２０，１））Ｌ１，Ｌ２，Ｌ３

解説・注意

信号名から信号長の４倍分の入力信号を、ＢＣＤコードとして読込みます。
例えば、Ｋ＝ＢＣＤＩＮ（１，２）では、入力信号の１から８まで８点の状態をＢＣＤコードとして読込みます。
入力信号の番号が大きいものが、高位のビットに対応します。
入力信号は、信号の番号が小さいものから順に４ビットずつに区切って、１０進符号化します。
信号は、オン状態で１、オフ状態で０として符号化します。

例）　入力信号の１から１２までが下表の状態の場合、ＢＣＤＩＮ（１，３）の値は３２９になります。

入力信号の番号	12	11	10	9	8	7	6	5	4	3	2	1
入力信号の状態	オフ	オフ	オン	オン	オフ	オフ	オン	オフ	オン	オフ	オフ	オン
２進数表現	０	０	１	１	０	０	１	０	１	０	０	１
１０進数表現				3				2				9

＜信号名＞、＜信号長＞には、定数、変数、式、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は、式の中で使用します。

入力信号を４ビット単位で符号化する際に、９（２進数で１００１）を超えた場合には、その信号を２進数で読込んだ値で符号化します。
つまり２進数で１１１１という信号が入力された場合、１５という数値を読みます。

サンプル
プログラム

```
PROGRAM  BCDINSAMPL
   K=BCDIN(1,2)        入力信号１から８までの状態を符号化入力して、その値に
   SPEED=K             より動作速度を設定して動作します。
   MOVE A1
   MOVE A2
END
```

ＢＣＤＯＵＴ

機能

信号をＢＣＤコード化して出力します。

フォーマット

ＢＣＤＯＵＴ（＜信号名＞，＜信号長＞，＜式＞）

文例

ＢＣＤＯＵＴ（１，４，３）
ＢＣＤＯＵＴ（Ｎ，Ｎ＋４，Ｋ）

解説・注意

式の値をＢＣＤコード化して信号長で指定した桁数分を、信号名から信号長の４倍分の出力信号に出力します。
例えば、ＢＣＤＯＵＴ（１，１，３）では、１から４まで４点の出力信号に数値３をＢＣＤコード化して出力しますので、出力信号１、２がオン状態になります。
出力信号の番号が大きいものが高位のビットに対応します。
式の値は１０進数での１桁毎を、出力信号の番号が小さいものから４ビットずつ順番に対応して符号化します。
式の値が信号長で指定した桁数を超える場合には、上位の桁は無視します。信号は、オン状態を１、オフ状態を０として符号化します。
例）　ＢＣＤＯＵＴ（１，３，９５２）を実行すると出力信号は、下表のようになります。

１０進数表現				9				5				2			
２進数表現	1	0	0	1	0	1	0	1	0	0	1	0			
出力信号の番号	12	11	10	9	8	7	6	5	4	3	2	1			
出力信号の状態	オン	オフ	オフ	オン	オフ	オン	オフ	オン	オフ	オフ	オン	オフ			

＜信号名＞、＜信号長＞、＜式＞には、定数、変数、式、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。

ＢＣＤＯＵＴ命令に続けて同じ信号の出力処理を行うと、信号の出力は後から出力したものが有効となります。

同時性を保証できる信号範囲には制約がある事に注意願います。
ＤＯＵＴ1、ＤＯＵＴ１０１、ＤＯＵＴ３０１からの１６ｂｉｔ刻みの範囲へのＢＣＤＯＵＴの同時性は保証できますが、その境界を跨ぐＢＣＤＯＵＴの同時性は保証できません。

サンプル
プログラム

```
PROGRAM  BCDOUTSMPL
J=0.0
FOR K=1 TO 4
   J=2 ^(K-1)
   BCDOUT(1,1,J)
   TIMER=0.5
   WAIT TIMER==0
   BCDOUT(1,1,0)
   NEXT K
END
```

出力信号の１から４までを順番に、０．５秒間ずつ出力していきます。

ＢＲＥＡＫ

機能

ロボットの動作を即時中断します。

フォーマット

ＯＮ＜監視条件＞［｛ＢＲＥＡＫ｜ＰＡＵＳＥ｝］ＤＯ＜文＞

文例

ＯＮ　ＤＩＮ（１）　ＢＲＥＡＫ　ＤＯ　ＳＵＢ

解説・注意

ＯＮ命令で指定した監視条件の成立時にロボットの動作を即時に中止してＤＯに続く文を実行します。動作中のロボットは減速停止します。
詳細はＯＮ命令を参照してください。
ＢＲＥＡＫで中断した動作は、ＲＥＳＵＭＥ命令で再開する事ができます。
本命令を使用する事により、システムで異常があった場合にロボットの動作を即時中断して、異常処理を行う事ができます。

サンプルプログラム

```
PROGRAM  BREAKSMPL
  REMARK *** MAIN PROGRAM ***
  ON DIN(24) BREAK  DO BREAKSUB
  MOVE A1
  MOVE A2
  MOVE A3
  WAIT MOTION>=100
  IGNORE DIN(24)
END

PROGRAM  BREAKSUB
  REMARK *** SUBROUTINE ***
  WAIT  DIN(-24)
  RESUME
END
```

システムに異常が発生して入力信号の２４がオン状態になると、ロボットの動作を即時中断して異常処理のサブプログラム、ＢＲＥＡＫＳＵＢに処理が移ります。

異常処理のサブプログラムでは、異常の入力信号２４がオフ状態になるまで待ちます。
異常状態が解除すると、中断した動作から再開します。

ＣＬＯＳＥ１、ＣＬＯＳＥ２，ＣＬＯＳＥＩ１，ＣＬＯＳＥＩ２

機能

ロボットのハンドを閉じます。

フォーマット

ＣＬＯＳＥ１
ＣＬＯＳＥ２
ＣＬＯＳＥＩ１
ＣＬＯＳＥＩ２

文例

ＣＬＯＳＥ１
ＣＬＯＳＥＩ２

解説・注意

ロボットのハンドを閉じます。添字の１，２はそれぞれハンド１，２を表わします。
本命令を実行すると、コントローラはロボットのハンド制御用の出力信号の状態を変更してハンドを閉じます。
ＣＬＯＳＥ命令では、ロボットが現在実行中の動作を終了するのを待ってからロボットのハンドを閉じます。
ＣＬＯＳＥＩ命令では、命令実行にて即時にロボットのハンドを閉じます。
本命令は、コントローラのＲＡＭドライブに、ＳＣＯＬ．ＬＩＢという名称のファイルがないと実行できません。
ロボットのハンドに信号を出力してから、実際にハンドが閉じるまでには、多少の時間がかかりますので注意してください。
ロボットのハンドを開くには、

ＯＰＥＮ１，ＯＰＥＮ２，ＯＰＥＮＩ１，ＯＰＥＮＩ２

の４つの命令が用意されています。

本命令は、システムライブラリ（ＳＣＯＬ．ＬＩＢ）に書かれているプログラムを実行します。ロボットのハンドの仕様に合わせて、ＳＣＯＬ．ＬＩＢの内容を変更する必要があります。

サンプル
プログラム

```
PROGRAM  CLOSESMPL
OPENI1                  Ａ１への移動が終了してからハンド１を閉じます。
MOVE A1                 ＣＬＯＳＥ１命令を実行してから、ロボットのハンドが
CLOSE1                  完全に閉じるまで０．５秒間待ちます。
DELAY 0.5
MOVE A2
END


PROGRAM  CLOSEISMPL
ENABLE NOWAIT
OPENI1
DELAY 0.5
MOVE A1
CLOSEI1                 Ａ１に動きながら、ハンド１を閉じます。
DELAY 0.5
MOVE A2
END
```

ＣＯＡＲＳＥ

機能

位置決め精度を粗に指定するためのシステム定数です。

フォーマット

ＣＯＡＲＳＥ

文例

ＡＣＣＵＲ＝ＣＯＡＲＳＥ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＡＣＣＵＲ＝ＣＯＡＲＳＥ

解説・注意

ＡＣＣＵＲ命令と共に使用して、位置決め精度を粗に設定します。
本システム定数は、０という値を持っています。プログラムの式の中で定数０として使用する事もできますが、プログラムが見づらくなるためこのような使い方はしないでください。
システム定数への代入はできません。
位置決め精度については、ＡＣＣＵＲ命令を参照してください。

サンプルプログラム

```
PROGRAM  COARSESMPL
MOVE A1
ACCUR=COARSE              位置決め精度を粗にして動作します。
MOVE A2
MOVE A3
END
```

ＣＯＭ０、ＣＯＭ１

機能

ＰＲＩＮＴ、ＩＮＰＵＴ命令で使う通信チャネルを指定します。

フォーマット

```
ＰＲＩＮＴ［｛ＣＯＭ０｜ＣＯＭ１｜ＴＰ｝，］
    ｛＜文字列＞｜＜式＞｝［，｛＜文字列＞｜＜式＞｝］
                                        ・・・［，ＣＲ］
ＩＮＰＵＴ［｛ＣＯＭ０｜ＣＯＭ１｜ＴＰ｝，］
    ＜変数＞［，＜変数＞］・・・
```

文例

```
ＰＲＩＮＴ　ＣＯＭ０，”＊＊＊　ＩＮＰＵＴ　Ｎ　＊＊＊”
ＰＲＩＮＴ　ＣＯＭ１，Ｎ，Ｎ＊１０
ＩＮＰＵＴ　ＣＯＭ１，Ｋ
```

解説・注意

ＰＲＩＮＴ、ＩＮＰＵＴ命令にて、通信チャネルを指定するために使用します。
ＣＯＭ０はティーチペンダント専用の通信チャネルです。
ＣＯＭ１は、コントローラのコネクタＣＯＭ１通信チャネルに対応します。
ＰＲＩＮＴ、ＩＮＰＵＴ命令にて通信チャネルを指定しないと、ティーチペンダント専用の通信チャネルに対してデータの入出力を行います。
通信処理については、ＰＲＩＮＴ，ＩＮＰＵＴの各命令を参照してください。

サンプルプログラム

```
PROGRAM  COMSAMPLE
PRINT COM0,"*** INPUT N ***"
INPUT COM0,N
PRINT COM1,N,CR
END
```

ティーチペンダントから入力した数値を、通信チャネルの１番に出力します。

ＣＯＮＦＩＧ

機能

ロボットの姿勢を指定します。

フォーマット

ＣＯＮＦＩＧ＝＜式＞

文例

ＣＯＮＦＩＧ＝１
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＣＯＮＦＩＧ＝ＲＩＧＨＴＹ

解説・注意

ロボットの姿勢を指定するシステム変数です。
ロボットの姿勢により周辺機器とロボットが干渉するような場合に、ロボットの姿勢を指定します。
ロボットの姿勢は、０で未定義、１で左肩系、２で右肩系になります。
ロボットの姿勢の指定には、システム定数のＦＲＥＥ，ＬＥＦＴＹ、ＲＩＧＨＴＹを使用する事ができます。ロボットの姿勢は、ＣＯＮＦＩＧ＝ＦＲＥＥで未定義、ＣＯＮＦＩＧ＝ＬＥＦＴＹで左肩系に、ＣＯＮＦＩＧ＝ＲＩＧＨＴＹで右肩系に設定できます。
ロボットに教示した位置データには、ロボットの姿勢も含まれるため、姿勢が未定義の場合には、ロボットは教示された姿勢で動作します。特に、姿勢を指定する必要の無い場合には、姿勢は未定義としてください。ロボットの姿勢の初期値は、未定義になっています。
ロボットの姿勢は、動作命令を実行した時に変更されます。
直線、円弧補間（ＭＯＶＥＳ、ＭＯＶＥＣ命令を使用した動作）を行う場合には、ロボットの姿勢変更は不可のため、エラーとなります。
直角座標ロボットでは、ロボットの姿勢の指定は無視します。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本システム変数は参照しても、現在のロボットの姿勢が参照できません。
現在のロボットの姿勢は、ＨＥＲＥ命令によって取得できます。
Ｎ＝ＨＥＲＥ．６
等でＮに現在の姿勢の値が入ります。
スカラロボットではＨＥＲＥ．６の値が０で未定義、１で左肩系、２で右肩系となります。ロボットの原点位置が正確に設定されていないと、ロボットの姿勢を変えたとき、ロボットが教示した位置とずれた位置に動作します。このため、ロボットに位置を教示する時には、実際にロボットが動作する姿勢で行ってください。

サンプル
プログラム

```
PROGRAM  CONFIGSMPL
CONFIG=RIGHTY   Ａ３への移動のみ、ロボットの姿勢を左肩系
MOVE A1         にして動作します。その他の動作は、全て右
MOVE A2         肩系の姿勢になり ます。
MOVE A3 WITH CONFIG=LEFTY
MOVE A4
END
```

ＣＯＮＴ

機能

システムの運転モードを参照するためのシステム定数です。

フォーマット

ＣＯＮＴ

文例

ＩＦ　ＭＯＤＥ＜＞ＣＯＮＴ　ＴＨＥＮ　ＳＴＯＰ

解説・注意

ＭＯＤＥ命令と共に使用して、システムの運転モードを参照します。
ＭＯＤＥ＝＝ＣＯＮＴのとき、システムは連続運転モードで動作しています。
本システム定数は、０という値を持っています。プログラムの式の中で、定数０として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。
システムの運転モードについては、ＭＯＤＥ命令を参照してください。

サンプルプログラム

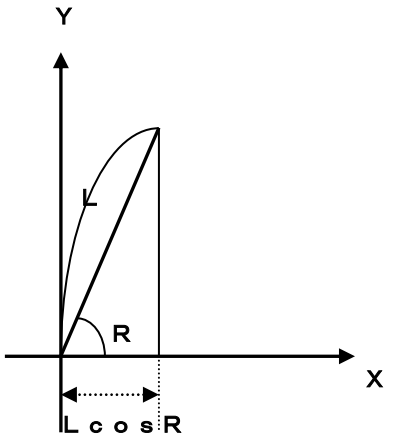
```
PROGRAM  CONTSAMPLE
IF MODE<>CONT THEN STOP        システムが連続運転モードで動作していなけ
MOVE A1                        れば、プログラムの実行を停止します。
MOVE A2
MOVE A3
END
```

ＣＯＳ

機能	余弦の値を計算します。
フォーマット	ＣＯＳ（＜式＞）
文例	Ｋ＝ＣＯＳ（６０） Ｊ１＝１－ＣＯＳ（１８０－Ｄ）
解説・注意	（　）内の余弦の値を計算します。 計算は度単位で行います。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 本命令は式の中で使用します。
サンプル プログラム	

```
PROGRAM  MAIN
COSSAMPLE(2,30,X)
PRINT TP,X,CR
END
PROGRAM  COSSAMPLE(L,R,X)
LOOP:
IF R>180 THEN R=R-360
IF R<-180 THEN R=R+360
IF R>180 OR R<-180 THEN GOTO LOOP
X=L*COS(R)
RETURN
END
```

Ｘ軸とＲの角度をなす線分Ｌ（長さＬ）にて、引数Ｌ，Ｒの値から線分ＬのＸ軸成分を求めて、その結果を引数Ｘとしてメインプログラムに返します。



ＣＲ

機能

通信チャンネルにＣＲ（キャリッジリターン）コードを出力します。

フォーマット

ＰＲＩＮＴ［｛ＣＯＭ０｜ＣＯＭ１｜ＴＰ｝，］｛＜文字列＞｜＜式＞｝
［，｛＜文字列＞｜＜式＞｝］　・・・　［，ＣＲ］

文例

ＰＲＩＮＴ　ＣＯＭ１，Ｋ，ＣＲ
ＰＲＩＮＴ　ＴＰ，ＣＲ

解説・注意

通信チャンネルにＣＲ（キャリッジリターン）コードを出力します。
通信チャネルとしては、ＣＯＭ０，ＣＯＭ１，ＴＰの中から１つを指定します。
ＣＯＭ０，およびＴＰはティーチペンダント専用の通信チャネルです。ＣＯＭ１コントローラのＣＯＭ１通信チャネルに対応します。
ＰＲＩＮＴ命令にて通信チャネルを指定しないと、ティーチペンダント専用の通信チャネルに対して、データの出力を行います。
ＰＲＩＮＴ命令の最後にＣＲを指定するとデータの最後にＣＲコード(０ＤＨ)が付けられます。
ＣＯＭ０（ＴＰ）へ出力した場合には、画面表示が改行されます。

サンプルプログラム

```
PROGRAM  COMSAMPLE
  PRINT TP,"*** INPUT N ***",CR      ＴＰに文字を表示し、改行します。
  INPUT TP,N
  PRINT TP,N,CR                      ＴＰにＮの値を表示し、改行します。
END
```

ＣＹＣＬＥ

機能

システムの運転モードを参照するためのシステム定数です。

フォーマット

ＣＹＣＬＥ

文例

ＩＦ　ＭＯＤＥ＜＞ＣＹＣＬＥ　ＴＨＥＮ　ＧＯＴＯ　ＬＯＯＰ

解説・注意

ＭＯＤＥ命令と共に使用して、システムの運転モードを参照します。
ＭＯＤＥ＝＝ＣＹＣＬＥのとき、システムはサイクル運転モードで動作しています。
本システム定数は、１という値を持っています。プログラムの式の中で、定数１として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。
システムの運転モードについては、ＭＯＤＥ命令を参照してください。

サンプルプログラム

```
PROGRAM  CYCLESMPL
 LOOP:                          システムがサイクル運転モードで動作してい
 MOVE A1                        なければ、プログラムの実行を先頭に戻して
 MOVE A2                        動作を繰返します。
 MOVE A3
 IF MODE<>CYCLE THEN GOTO LOOP
END
```

ＤＡＴＡ

機能	教示点のための位置や座標データを定義するデータブロックの始まりを示します。 データブロックの詳細は、5.3.5 データブロックを参照ください。
フォーマット	ＤＡＴＡ
文例	ＤＡＴＡ
解説・注意	データブロックはプログラムエディタではなく、データエディタにより編集されます。ただし、書式エラーの場合は、プログラムエディタで編集します。
サンプル プログラム	PROGRAM MAIN MOVE HOME END DATA POINT HOME = 650, 0, 100, 0, 0 /RIGHTY END

ＤＥＣＥＬ

機能

ロボットの減速時の加速度を指定します。

フォーマット

ＤＥＣＥＬ＝＜式＞

文例

```
ＤＥＣＥＬ＝８０
ＤＥＣＥＬ＝０．８＊ＤＥＣＥＬ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＤＥＣＥＬ＝９０
```

解説・注意

ロボットの減速時の加速度を指定するシステム変数です。
加速度は、標準の加速度に対するパーセンテージで指定します。
ＳＣＯＬ言語では、加速度を加速時と減速時で別々に指定します。加速時の加速度の指定には、ＡＣＣＥＬ命令を使用します。
ロボットに重量物を持たせる時、必要によって加速度を下げたい場合に使用します。この様な場合には、減速時の加速度に併せて加速時の加速度も変更してください。
負荷に応じた加速度の設定値は「据付け・輸送編」を参照してください。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。
ロボットには、それぞれ加速度の上限が設定されています。上限値以上の数値を指定しても、加速度は上限値で抑えられます。
０以下の値は１と見なします。
本システム変数を参照すると、現在の減速時の加速度が参照できます。
加速度の初期値は、１００％になっています。

サンプルプログラム

```
PROGRAM  DECELSMPL
FOR K=1 TO 100        加速度率を１％から１００％まで１％きざみで変えていきます。
DECEL=K
MOVE A1
MOVE A2
NEXT K
END
```

ＤＥＬＡＹ

機能

ロボットのアーム動作を指定時間停止します。

フォーマット

ＤＥＬＡＹ＜時間＞

文例

ＤＥＬＡＹ　０．５
ＤＥＬＡＹ　Ｔ＊０．２

解説・注意

ロボットのアーム動作を指定時間停止します。
＜時間＞は秒単位で指定しますが、実行結果の精度が限られるため、０．０１秒単位を目安に設定してください。
＜時間＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
ＤＥＬＡＹ命令の実行中にプログラムの停止操作を行うと、指定時間経過後、自動運転が停止します。また、ＤＥＬＡＹ命令の実行中にサーボオフ、非常停止操作等にて自動運転を中断した場合には、再起動時に再度ＤＥＬＡＹ命令をやり直します。
ＤＥＬＡＹ命令は動作制御命令の一種で、指定時間アームの動作を止める命令です。
ＤＥＬＡＹ命令の実行中はロボットのアーム動作は停止しますが、プログラムの実行は停止しません。プログラムの実行を遅らせたい場合にはＴＩＭＥＲとＷＡＩＴ命令を使用します。

サンプルプログラム

```
PROGRAM  DELAYSMPL
MOVE A1                 各動作の間に２秒間ずつ動作を停止します。
DELAY 2
MOVE A2
DELAY 2
MOVE A3
DELAY 2
END
```

ＤＥＳＴ

機能

現在の動作の目標位置を取出します。

フォーマット

ＤＥＳＴ

文例

Ａ１＝ＤＥＳＴ
Ｘ＝ＤＥＳＴ．Ｘ

解説・注意

現在の動作のワールド座標系での目標位置を取出します。
ＤＥＳＴは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが既に位置決め完了して停止している場合には、ＤＥＳＴの内容は現在位置に一致します。

サンプルプログラム

```
PROGRAM  DESTSAMPLE
AA=A                          Ａ１からＡ４まで移動中に入力信号１を監視して、
MOVE A                        信号がオンした時の目標位置をティーチペンダント
ON DIN(1) DO AA=DEST          に表示します。
MOVE A1
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.X,AA.Y,AA.Z
END
```

ＤＩＭ.....ＡＳ.....

機能	配列変数を定義します。
フォーマット	ＤＩＭ　＜配列変数＞（＜要素数＞，＜要素数＞，...）　ＡＳ　＜型名＞
文例	ＤＩＭ　Ａ（５）　ＡＳ　ＩＮＴ
解説・注意	配列変数の型と要素数を定義します。 配列変数は定義したファイル内のどこでも参照、代入可能なグローバル変数としてのみ定義できます。また配列のインデックスは１～要素数の値となります。つまり、文例では１から５です。 インデックス範囲外への初期値設定は、“SELECT”時にエラーとなります。また、インデックス範囲外への参照、代入はプログラム実行時にエラーとなります。 型名には，ＩＮＴ（整数型），ＲＥＡＬ（実数型），ＰＯＩＮＴ（位置型），ＴＲＡＮＳ（座標型），ＰＡＹＬＯＡＤ（負荷型）の５つが宣言できます。 INT 型、REAL型配列の初期値設定はグルーバルブロックにPOINT、TRANS、PAYLOAD型配列の初期値設定はデータブロックに記述してください。

サンプル
プログラム

```
GLOBAL
DIM ICHI(3) AS POINT  位置型の１次元配列を宣言
END
PROGRAM  DIMSAMPLE
ICHI(1) = P0
ICHI(2) = POINT(500.0, 0.0, 100.0, 0.0, 0.0, 0)
ICHI(3) = {500.0, -200.0, 100.0, 0.0, 0.0, 0}
FOR I＝1 TO 3
MOVE ICHI(I)
NEXT I
END
DATA
POINT P0       = 500.0, 200.0, 100.0, 0.0, 0.0 /LEFTY
POINT ICHI(1)  = 650.0, 0.0, 100.0, 0.0, 0.0 /LEFTY
END
```

ＤＩＮ

機能

入力信号の状態を読込みます。

フォーマット

ＤＩＮ（＜信号名＞［，＜信号名＞］・・・）

文例

```
ＩＦ　ＤＩＮ（１）　ＴＨＥＮ　ＧＯＴＯ　ＬＯＯＰ
ＷＡＩＴ　ＤＩＮ（１，－２，３）
ＯＮ　ＤＩＮ（Ｊ，Ｊ＋１，Ｊ＋２）　ＤＯ　ＲＥＴＵＲＮ
```

解説・注意

指定された入力信号の状態を読込みます。
本命令はＩＦ，ＷＡＩＴ，ＯＮの各命令と共に使用して、外部信号の条件判断に用います。
＜信号名＞には、読込む入力信号の番号を指定します。＜信号名＞の符号が、正ならば信号はオン状態、負ならばオフ状態を指定する事になります。＜信号名＞の指定は１０個までが有効になります。（１０個を超えた分は無視します。）
（　）内の信号の状態が、全て指定した通りならば条件が成立したものとして真（１）を返します。不成立の場合には偽（０）を返します。
＜信号名＞には、定数、変数、及び演算式を使用する事ができます。
ただし、ベクトル型のデータは使用できません。

サンプルプログラム

```
PROGRAM  DINSAMPLE
WAIT DIN(1)                    入力信号の１がオンになるまで待ちます。
MOVE A1
MOVE A2
MOVE A3
END
```

ＤＩＳＡＢＬＥ

機能

システムスイッチを無効にします。

フォーマット

ＤＩＳＡＢＬＥ　＜スイッチ＞［，＜スイッチ＞］・・・

文例

ＤＩＳＡＢＬＥ　ＰＡＳＳ
ＤＩＳＡＢＬＥ　ＰＡＳＳ，ＮＯＷＡＩＴ

解説・注意

ロボットの動作に関連したシステムスイッチを無効にします。
システムスイッチとしては、次の６種類があります。

（１）　ＰＡＳＳ

ショートカット動作の指定を行います。ショートカット動作では、現在の動作が位置決め完了する前に、次の移動を開始します。現在の動作から次の動作へ移るタイミングは、システム変数のＰＡＳＳにて指定します。
ショートカット動作を指定する事により、ロボットの動作時間を短縮する事ができます。ショートカット動作については、５章でも説明していますので、そちらを参照してください。
ＤＩＳＡＢＬＥ　ＰＡＳＳでショートカット動作は解除されます。初期状態ではＤＩＳＡＢＬＥ　ＰＡＳＳになっています。

（２）　ＮＯＷＡＩＴ

外部信号の入出力をロボットの位置決め完了を待って行うか、待たずに行うかを指定します。
信号出力のタイミングについては、５章にて説明していますので、そちらを参照してください。
ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴで外部信号の入出力は、ロボットの位置決め完了を待って行うようになります。
初期状態では、ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴになっています。

（３）　ＳＷＩＴＣＨ

マルチタスク動作時に、タスク切り替え動作を行うかどうかを決定します。
ＤＩＳＡＢＬＥ　ＳＷＩＴＣＨでタスク切換えは禁止されます。
初期状態では、ＥＮＡＢＬＥ　ＳＷＩＴＣＨになっています。

（４） ＭＯＶＥＳＹＮＣ

動作命令同期モードか、動作命令非同期モードかを指定します。

ＤＩＳＡＢＬＥ ＭＯＶＥＳＹＮＣ状態(動作命令非同期モード)では四つ先の動作命令直前まで実行した後、位置決め完了を待ちます。またシステム変数ＰＡＳＳがＥＮＡＢＬＥであれば、ショートカット(パス)動作が可能です。初期状態は、ユーザパラメータ[U03]によって設定されます。

（５） ＳＭＯＯＴＨ

スムース動作を指定します。スムース指定された動作命令は目標位置まで減速することなく移動し、続く動作命令はスムース指定の有無に関係なく加速せずに移動を開始します。

ＥＮＡＢＬＥ ＳＭＯＯＴＨでスムース動作を開始し、ＤＩＳＡＢＬＥ ＳＭＯＯＴＨでスムース動作を解除します。

初期状態では、ＤＩＳＡＢＬＥ ＳＭＯＯＴＨになっています。

補間動作命令（ＭＯＶＥＳ，ＭＯＶＥＣ）のみがスムース機能の制御対象となります。

（６） ＳＬＯＷＤＯＷＮ

スローダウン動作を指定します。スローダウン動作では、現在の動作の移動中に速度を変更（減速）することができます。

ＥＮＡＢＬＥ ＳＬＯＷＤＯＷＮ以降の動作命令は指定パラメータにしたがって、移動途中から速度を変更します。

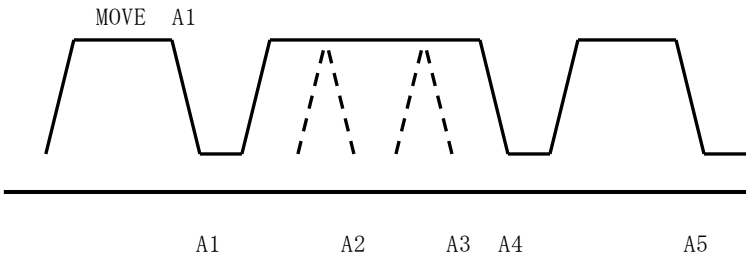
ＤＩＳＡＢＬＥ ＳＬＯＷＤＯＷＮでスローダウン動作を解除します。

初期状態では、ＤＩＳＡＢＬＥ ＳＬＯＷＤＯＷＮになっています。

システムスイッチを有効にするためには、ＥＮＡＢＬＥ命令を使用します。

サンプル
プログラム

```
PROGRAM  DISABLESPL
  MOVE A1              Ａ１からＡ４までをショートカット動作で移動します。
  PASS＝80
  ENABLE PASS          Ａ４以降の動作ではショートカット動作を解除します。
  MOVE A2
  MOVE A3
  DISABLE PASS
  MOVE A4
  MOVE A5
END
```



ＤＯ

機能	ＯＮ命令と組合わせて条件の監視を行います。
フォーマット	ＯＮ＜監視条件＞［｛ＢＲＥＡＫ｜ＰＡＵＳＥ｝］ＤＯ＜文＞
文例	ＯＮ　ＤＩＮ（１）　ＤＯ　ＲＥＴＵＲＮ ＯＮ　ＴＩＭＥＲ　ＤＯ　ＭＯＶＥ　Ａ１
解説・注意	＜監視条件＞の条件が成立したらＤＯに続く文を実行します。 条件の監視は、ロボットの動作中いかんにかかわらず行います。 ロボットが動作中でも、ＯＮ命令の処理はロボットの動作と並行して行います。 ＭＯＴＩＯＮ，ＭＯＴＩＯＮＴ，ＲＥＭＡＩＮ，ＲＥＭＡＩＮＴ命令を監視条件として使用した場合には後続の動作命令に対して条件の監視を行います。ＴＩＭＥＲ、ＥＲＲＯＲを使用した場合にはロボットの動作に関係なく条件の監視を行います。 ＤＩＮ命令などで、入力信号の監視を指定した場合には、システムスイッチＮＯＷＡＩＴの指定により、条件の監視を開始するタイミングが異なります。 ＥＮＡＢＬＥ　ＮＯＷＡＩＴの場合には、ロボットの動作に関係なく信号の監視を開始します。ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴの場合には実行中のロボットの動作が終了してから信号の監視を開始します。 ＤＯに続く文を実行するタイミングは監視条件が成立した時に実行中の命令の処理が終了してからになります。ただし、ＷＡＩＴ命令を実行中には、ＷＡＩＴの処理を中断してＤＯに続く文の処理を行います。ロボットが動作中の場合には、次の３種類の実行タイミングを指定できます。 ＢＲＥＡＫ　：　動作を即時中止してＤＯに続く文を実行します。 ＰＡＵＳＥ　：　実行中の動作の終了を待ってＤＯに続く文を実行します。但しアーム動作中は、サブプログラム呼出し命令、メインプログラムへの復帰命令、動作命令を除いて通常プログラム実行を続けます。これらの命令実行時には、アーム停止までプログラム実行が停止します。 指定なし：　動作と並行してＤＯに続く文を実行します。 ＤＯに続く文が動作命令の場合には必ずＢＲＥＡＫかＰＡＵＳＥを指定して下さい。

ＤＯに続く文（これをＤＯ節と言います）の実行が終了し、かつＤＯ節内で動作命令を実行した場合はそのアーム動作が完了すると、プログラムの実行はＯＮ命令の条件成立前の流れに従って再開します。（ＷＡＩＴ命令を中断した場合には，中断したＷＡＩＴ命令から実行を再開します。ただしＤＯに続く文にてラベルへのプログラム分岐を行った場合には、そのラベルを持つ文から実行します。
同時に監視できる条件は最大１０組までです。また、同一のＯＮ命令中で指定できる入力信号は４点までです。
同時に、複数の監視条件が成立した場合には、ＯＮ命令の実行された順番を優先順位にして、最も優先順位の高いＯＮ命令に対応するＤＯに続く文のみを実行して、その他のものは無視します。
ＯＮ命令による条件の監視は、他のＯＮ命令のＤＯに続く文を実行中は行いません。また、ＳＴＯＰ命令やエラーの発生により、プログラムの実行が停止している時にも条件監視は行いません。
システムのタイマーを監視条件として指定した場合には、条件のチェックは監視対象の状態が変わった時にのみ行います。外部信号、エラー条件、及び動作状態（動作の残り移動量等）の監視は、状態そのもので監視します。
ＯＮ文で指定した条件監視の解除は、ＩＧＮＯＲＥ命令で行います。条件監視は、条件が成立してＤＯに続く文を実行した時点にも解除します。

注１）　ＯＮ～ＤＯ命令は現状以下の使い方に限定されています。

・ＯＮ　ＴＩＭＥＲ　ＤＯ　＜文＞
　タイマーが０になったら＜文＞を実行します。

・ＯＮ　ＤＩＮ（　）　ＤＯ　＜文＞
　入力信号が指定状態になったら＜文＞を実行します。同時に監視できるのは４点までです。４点以上を指定すると４点を超えた分は無視します。

・ＯＮ　ＭＯＴＩＯＮ＞＝＜式＞　ＤＯ　＜文＞
　本命令の次に実行する動作命令の動作量が指定以上になったら＜文＞を実行します。ＭＯＴＩＯＮに関しては＞＝以外の比較演算子は使用できません。

・ＯＮ　ＭＯＴＩＯＮＴ＞＝＜式＞　ＤＯ　＜文＞
　本命令の次に実行する動作命令の動作時間が指定以上になったら＜文＞を実行します。ＭＯＴＩＯＮＴに関しては＞＝以外の比較演算子は使用できません。

・ＯＮ　ＲＥＭＡＩＮ＜＝＜式＞　ＤＯ　＜文＞

本命令の次に実行する動作命令の残り動作量が指定以下になったら＜文＞を実行します。ＲＥＭＡＩＮに関しては＜＝以外の比較演算子は使用できません。

・ＯＮ　ＲＥＭＡＩＮＴ＜＝＜式＞　ＤＯ　＜文＞

本命令の次に実行する動作命令の残り動作時間が指定以下になったら＜文＞を実行します。ＲＥＭＡＩＮＴに関しては＜＝以外の比較演算子は使用できません。

注２）　ＤＯ節に続く文の中ではタスク制御に関する以下の命令は使用できません。

ＴＡＳＫ，ＫＩＬＬ，ＳＷＩＴＣＨ

ＤＯ節以下でこれらの命令を使用した場合、その命令語は動作しません。また、サブタスクでＯＮ命令による条件の監視は行えませんので注意してください。

注３）　ＯＮ ＭＯＴＩＯＮ条件、ＯＮ ＭＯＴＩＯＮＴ条件、ＯＮ ＲＥＭＡＩＮ条件、ＯＮ ＲＥＭＡＩＮＴ条件の監視対象動作を停止したり、対象動作中に低速指令を発行するとＯＮ条件は解除されます。

サンプル
プログラム

```
PROGRAM MAIN
DOSAMPLE
MOVE P
END
PROGRAM  DOSAMPLE
ON DIN(1) PAUSE DO RETURN
MOVE A1                 入力信号の１がオンになったら、実行中の動作の完了
MOVE A2                 にてメインプログラムに戻ります。
MOVE A3
WAIT MOTION >=100
IGNORE DIN(1)
RETURN
END
```

ＤＯ節内の注意

ＯＮ～ＤＯ命令は、監視するＯＮ条件と条件が成立した場合に起動されるＤＯ節の組み合わせを登録します。このため次のＳＣＯＬプログラムで

```
 PROGRAM MAIN
 SIG=1
 ON DIN(1) DO INPUT SIG
 SUB
 IGNORE DIN(1)
 PRINT SIG
END
PROGRAM SUB
 MOVE P
 WAIT MOTION>=100
END
```

P への移動中に DIN(1)を ON すると、プログラムＳＵＢ内で変数ＳＩＧが定義されておらずＩＮＰＵＴ命令で取り込まれた変数を格納する場所がありませんので、ＤＯ節を実行することが出来ません。このような場合は変数ＳＩＧをグローバル変数として定義することで、該当ＤＯ節を正常に動作させることが出来ます。

```
GLOBAL
 SIG=0
END
PROGRAM MAIN
 SIG=1
 ON DIN(1) DO INPUT SIG
 SUB
 IGNORE DIN(1)
 PRINT SIG
END
PROGRAM SUB
 MOVE P
 WAIT MOTION>=100
END
```

なおＤＯ節内では、タスク切替条件が成立しても、あるいは“ＳＷＩＴＣＨ”命令を実行してもタスクは切替わりません。また“ＴＡＳＫ”命令や“ＫＩＬＬ”命令を実行しようとするとエラーとなります。

ＤＯＵＴ

機能

外部信号を出力します。

フォーマット

ＤＯＵＴ（＜信号名＞［，＜信号名＞］・・・）

文例

ＤＯＵＴ（１，２，－３）
ＤＯＵＴ（Ｊ，Ｊ＋１，Ｊ＋２）

解説・注意

指定された出力信号を出力します。
＜信号名＞には、出力する信号の番号を指定します。
＜信号名＞の符号が正ならば、信号はオン状態、負ならばオフ状態を指定する事になります。＜信号名＞の指定は１０個までが有効になります。
＜信号名＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
ＤＯＵＴ命令に続けて同じ信号の出力処理を行うと、信号の出力は保証されません。また、複数の信号出力を指定した場合、信号出力タイミングの同一性は保証されません。
ＤＯＵＴ(－４，３，－２，１)というロボットプログラムは、先頭から順にＤＯＵＴ（－４）、ＤＯＵＴ（３）、ＤＯＵＴ（－２）、ＤＯＵＴ（１）というロボットプログラムに分解して実行されます。つまりＤＯＵＴ(４，－３，２，－１)後のＤＯＵＴ(－４，３，－２，１)というロボットプログラムを実行すると数～数１００ｍｓ間隔程度で信号が
“１０１０”→”００１０”→”００１１”→”０００１”→”０１０１”
と変化して最終的に目的の値となります。

サンプルプログラム

```
PROGRAM  DOUTSAMPLE
FOR K=1 TO 16          出力信号の１から１６までを、順番に０．５秒間ずつ
DOUT(K)                オンします。
TIMER=0.5
WAIT TIMER==0
DOUT(-K)
NEXT K
END
```

ＥＬＳＥ

機能

ＩＦ～ＴＨＥＮ～命令と組合わせて条件判断に用います。

フォーマット

ＩＦ＜論理式＞ＴＨＥＮ＜文＞［ＥＬＳＥ＜文＞］

文例

ＩＦ　ＤＩＮ（１）　ＴＨＥＮ　Ｋ＝Ｋ＋１　ＥＬＳＥ　Ｋ＝０

解説・注意

ＩＦ文を用いた条件判断にて、条件が成立しなかった場合に実行する文を指定します。
ＩＦ命令にて、ＥＬＳＥ＜文＞は省略する事ができます。ＥＬＳＥ＜文＞を省略すると、条件が成立しなかった場合にＩＦ文の次の命令を実行します。
ＴＨＥＮ、ＥＬＳＥ命令に続く＜文＞には、ＰＲＯＧＲＡＭ，ＥＮＤ，ＩＦ，ＦＯＲ，ＮＥＸＴ及びＷＡＩＴ命令は使用できません。
条件判断については、ＩＦ文を参照してください。

サンプルプログラム

```
PROGRAM  ELSESAMPLE
IF DIN(1) THEN SPEED=100 ELSE SPEED=50
MOVE A1
MOVE A2
MOVE A3
END
```

入力信号１がオンの場合には１００％、オフの場合には５０％の速度で動作します。

ＥＮＡＢＬＥ

機能

システムスイッチを有効にします。

フォーマット

ＥＮＡＢＬＥ　＜スイッチ＞［，＜スイッチ＞］・・・

文例

ＥＮＡＢＬＥ　ＰＡＳＳ
ＥＮＡＢＬＥ　ＰＡＳＳ，ＮＯＷＡＩＴ

解説・注意

ロボットの動作に関連したシステムスイッチを有効にします。
システムスイッチとしては、次の６種類があります。

（１）　ＰＡＳＳ

ショートカット動作の指定を行います。ショートカット動作では、現在の動作が位置決め完了する前に次の移動を開始します。現在の動作から次の動作へ移るタイミングは、システム変数のＰＡＳＳにて指定します。
ショートカット動作を指定する事により、ロボットの動作時間を短縮する事ができます。ショートカット動作については、５章においても説明していますので、そちらを参照してください。
ＥＮＡＢＬＥ　ＰＡＳＳでショートカット動作を指定します。
初期状態では、ＤＩＳＡＢＬＥ　ＰＡＳＳになっています。

（２）　ＮＯＷＡＩＴ

外部信号の入出力をロボットの位置決め完了を待って行うか、待たずに行うかを指定します。
信号出力のタイミングについては、５章にて説明していますので、そちらを参照してださい。
ＥＮＡＢＬＥ　ＮＯＷＡＩＴで外部信号の入出力は、ロボットの位置決め完了を待たずに行うようになります。
初期状態では、ＤＩＳＡＢＬＥ　ＳＷＩＴＣＨになっています。

（３）　ＳＷＩＴＣＨ

マルチタスク動作時に、タスク切換え動作を行うかどうかを決定します。
ＤＩＳＡＢＬＥ　ＳＷＩＴＣＨでタスク切換えは禁止されます。
初期状態では、ＥＮＡＢＬＥ　ＳＷＩＴＣＨになっています。

（４）　ＭＯＶＥＳＹＮＣ

動作命令同期モードか、動作命令非同期モードかを指定します。

ＥＮＡＢＬＥ　ＭＯＶＥＳＹＮＣ状態(動作命令同期モード)では次の動作命令直前まで実行した後、位置決め完了を待ちます。初期状態は、ユーザパラメータ[U03]で設定されます。

なおこのモードでは､システム変数ＰＡＳＳの状態によらずショートカット(パス)動作できません。パスさせたい動作命令は、ＳＣＯＬプログラム上で予めシステム変数ＭＯＶＥＳＹＮＣをＤＩＳＡＢＬＥと設定してください。

（５）　ＳＭＯＯＴＨ

スムース動作を指定します｡スムース指定された動作命令は目標位置まで減速することなく移動し、続く動作命令はスムース指定の有無に関係なく加速せずに移動を開始します。

ＥＮＡＢＬＥ　ＳＭＯＯＴＨでスムース動作を開始し、ＤＩＳＡＢＬＥ　ＳＭＯＯＴＨでスムース動作を解除します。

初期状態では、ＤＩＳＡＢＬＥ　ＳＭＯＯＴＨになっています。

補間動作命令（ＭＯＶＥＳ，ＭＯＶＥＣ）のみがスムース機能の制御対象となります。

（６）　ＳＬＯＷＤＯＷＮ

スローダウン動作を指定します｡スローダウン動作では､現在の動作の移動中に速度を変更（減速）することができます。

ＥＮＡＢＬＥ　ＳＬＯＷＤＯＷＮ以降の動作命令は指定パラメータにしたがって、移動途中から速度を変更します。

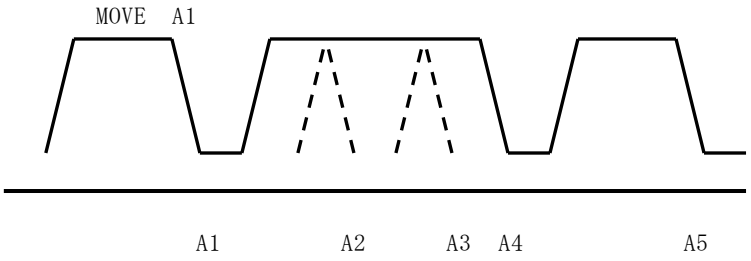
ＤＩＳＡＢＬＥ　ＳＬＯＷＤＯＷＮでスローダウン動作を解除します。

初期状態では、ＤＩＳＡＢＬＥ　ＳＬＯＷＤＯＷＮになっています。

システムスイッチを無効にするためには、ＤＩＳＡＢＬＥ命令を使用します。

サンプル
プログラム

```
PROGRAM  ENABLESMPL
  MOVE A1              Ａ１からＡ４までをショートカット動作で移動します。
  PASS＝80
  ENABLE PASS          Ａ４以降の動作ではショートカット動作を解除します。
  MOVE A2
  MOVE A3
  DISABLE PASS
  MOVE A4
  MOVE A5
END
```



ＥＮＤ

機能

プログラムの終わりを示します。

フォーマット

ＥＮＤ

文例

ＥＮＤ

解説・注意

プログラムの終わりを示します。
サイクル運転モードで実行中は、プログラム開始時に自動的に生成されるメインタスクのＥＮＤにてプログラムの実行を終了します。連続運転モードでは、プログラムの先頭に戻って実行を継続します。
プログラムをサブプログラムとして実行する場合に、ＥＮＤの前にＲＥＴＵＲＮ文が無くても、プログラムの処理はＥＮＤにてメインプログラムに戻ります。
プログラムは、ＰＲＯＧＲＡＭ文とＥＮＤ文の間に記述します。ＥＮＤ文のない場合は、エラーになります。
メインプログラムのＥＮＤ文実行後、内部の変数の値はクリアされます。

サンプルプログラム

```
PROGRAM  ENDSAMPLE
MOVE A1
MOVE A2
MOVE A3
END
```

ＰＲＯＧＲＡＭからＥＮＤまでが、１つのプログラムとして実行されます。

ＥＸＰ

機能

ｅのべき乗を計算します。

フォーマット

ＥＸＰ（＜式＞）

文例

Ｋ＝ＥＸＰ（３．５）
Ｊ１＝Ｎ＊ＥＸＰ（Ｌ－Ｋ）

解説・注意

ｅ（＝２．７１８２８・・・）のべき乗を計算します。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM MAIN
EXPSAMPLE(5,K)
PRINT TP,K,CR
END
PROGRAM  EXPSAMPLE(N,K)
K=EXP(N)                引数Ｎからｅの Ｎ乗を計算して、その結果を引数Ｋ
RETURN                  としてメインプログラムに返します。
END
```

ＦＩＮＥ

機能

位置決め精度を精に指定するためのシステム定数です。

フォーマット

ＦＩＮＥ

文例

ＡＣＣＵＲ＝ＦＩＮＥ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＡＣＣＵＲ＝ＦＩＮＥ

解説・注意

ＡＣＣＵＲ命令と共に使用して、位置決め精度を精に設定します。
本システム定数は、１という値を持っています。プログラムの式の中で、定数１として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。
位置決め精度については、ＡＣＣＵＲ命令を参照してください。

サンプルプログラム

```
PROGRAM  FINESAMPLE
ACCUR=FINE                    位置決め精度を精にして動作します。
MOVE A1
MOVE A2
MOVE A3
END
```

ＦＯＲ

機能

プログラムの繰返しを指定します。

フォーマット

ＦＯＲ＜変数＞＝＜式１＞ＴＯ＜式２＞［ＳＴＥＰ＜式３＞］
　　・・・・
ＮＥＸＴ［＜変数＞］

文例

```
ＦＯＲ　Ｋ＝１　ＴＯ　４
　　・・・・
ＮＥＸＴ　Ｋ
ＦＯＲ　Ｎ＝Ｋ１　ＴＯ　Ｋ１＋Ｋ２　ＳＴＥＰ　Ｋ３
　　・・・・
ＮＥＸＴ　Ｎ
```

解説・注意

プログラムの繰返しを指定します。

ＦＯＲ命令とＮＥＸＴ命令にはさまれたプログラムを、繰返し実行します。プログラムの実行は、ＦＯＲ文で指定した条件が満たされるまで繰返します。

ＦＯＲ文を実行すると、＜変数＞に＜式１＞の値が代入されます。ＮＥＸＴ文の実行にて、＜変数＞にはＳＴＥＰにて指定した＜式３＞の値が加えられます。この時に、プログラムの実行は、変数の値が＜式２＞の値を超えていればＮＥＸＴの次の命令に、超えていなければ対応するＦＯＲ文の次の文へと分岐します。

＜式１＞，＜式２＞，＜式３＞は、最初にＦＯＲ文を実行した時の値を用います。このため、これらの式の値がプログラムの繰返し中に変化しても、プログラムの繰返し回数は変化しません。

変数は、プログラムの繰返しを管理するために用いていますので、プログラムの繰返し中には代入等により値を変更しないでください。

＜式３＞の値が１の時には、ＳＴＥＰ以後を省略する事ができます。

＜式１＞，＜式２＞，＜式３＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。

対応するＮＥＸＴ文には、ＦＯＲ文で指定した変数を指定してください。

ＮＥＸＴ文で＜変数＞を指定しない場合には、最も近いＦＯＲ文（最後に実行したＦＯＲ文）との間でループを作ります。

ＦＯＲ～ＮＥＸＴのループの中で別のＦＯＲ命令を使用（ＦＯＲ命令をネストして使用）する場合には、127重以下になるようにしてください。ＦＯＲループのネスティングが127重を超えるとエラーになります。

注１）　ＦＯＲループの終了はＮＥＸＴ文で行います。従って少なくとも１回はループが実行されます。

注２）　＜変数＞，＜式１＞，＜式２＞，＜式３＞の値としては実数も使用できます。しかし実数には誤差があるため、ＦＯＲ命令のループ回数は整数になるようにしてください。＜変数＞，＜式１＞，＜式２＞，＜式３＞の値は実行時に同じデータ型に変換されて処理されます。データ型の変換は次のように行われます。

（１）　ＦＯＲ文の実行時に＜変数＞のデータ型が未定義の場合。

＜変数＞として使用する識別子が、ＦＯＲ命令実行時に初めてプログラム中に使用される場合には、ＦＯＲ命令実行時には＜変数＞のデータ型は未定義です。この場合には全てのデータは＜式１＞のデータ型に変換されて処理されます。

（２）　ＦＯＲ文の実行時に＜変数＞のデータ型が定義済みの場合。

＜変数＞として使用する識別子が、ＦＯＲ命令実行以前にプログラム中で使用されている場合には、＜変数＞のデータ型は最初に代入された時のデータ型になっています。この場合には全てのデータは＜変数＞のデータ型に変換されて処理されます。

例）
```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
  Ｊ１＝５
  ＦＯＲ　Ｊ１＝０．１　ＴＯ　０．９　ＳＴＥＰ　０．０１
    ＭＯＶＥ　Ａ１
    ＭＯＶＥ　Ａ２
  ＮＥＸＴ　Ｊ１
ＥＮＤ
```

上記のプログラムを実行するとＦＯＲ命令を実行する前にＪ１のデータ型が整数型に定義されているため、ＦＯＲ命令で使用する変数は全て整数型に変換されて処理されます。このため上記の例のＦＯＲ文は次に示すＦＯＲ文と同じ処理になります。

```
ＦＯＲ　Ｊ１＝０　ＴＯ　０　ＳＴＥＰ　０
```

従ってこの例ではループは１回しか行われません。

注３）　次に示すようなＦＯＲ文の使い方はエラーになります。

（１）
```
ＦＯＲ　Ｊ１＝　・・・　　　Ｊ２に対応するＮＥＸＴ
  ＦＯＲ　Ｊ２＝　・・・　　命令がありません。
  ・・・・
  ＮＥＸＴ　Ｊ１
```

（２）　ＦＯＲ　Ｋ＝　・・・
・・・
ＩＦ　ＤＩＮ(1) ＴＨＥＮ　ＧＯＴＯ　Ｌ１
・・・
ＮＥＸＴ　Ｋ
・・・
Ｌ１：

ループ内からループ外への分岐命令は不可です。同様に、ループ外からループ内への分岐命令も不可です。

サンプル
プログラム

```
PROGRAM  FORSAMPLE
FOR K=1 TO 100        Ａ１，Ａ２間の往復動作を、１００回繰返します。
MOVE A1
MOVE A2
NEXT K
END
```

ＦＲＥＥ

機能

ロボットの姿勢を未定義にするためのシステム定数です。

フォーマット

ＦＲＥＥ

文例

ＣＯＮＦＩＧ＝ＦＲＥＥ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＣＯＮＦＩＧ＝ＦＲＥＥ

解説・注意

ＣＯＮＦＩＧ命令と共に使用して、ロボットの姿勢を未定義に設定します。
本システム定数は、０という値を持っています。プログラムの式の中で定数０として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。ロボットの姿勢については、ＣＯＮＦＩＧ命令を参照してください。

サンプルプログラム

```
PROGRAM  FREESAMPLE
CONFIG=FREE                ロボットの姿勢を未定義として動作します。
MOVE A1
MOVE A2
END
```

ＦＲＥＥＬＯＡＤ

機能

ロボットの手先の負荷データを０にします。

フォーマット

ＦＲＥＥＬＯＡＤ

文例

ＦＲＥＥＬＯＡＤ

解説・注意

ロボットの手先の負荷データを、０に設定して動作します。
ＳＣＯＬ言語では、どんな負荷に対してもロボットが最適な状態で動作できるように、ロボットの手先にかかる負荷を負荷データとして設定できるようになっています。
ロボットの手先にかかる負荷は、システム変数のＰＡＹＬＯＡＤに設定します。この設定値により、コントローラは加減速等の制御定数を負荷に最適となるように計算してロボットを動作します。負荷データとしては、手先にかかる負荷の重量と慣性モーメントの値を指定します。
ＦＲＥＥＬＯＡＤ命令は、この負荷データを０にするための命令です。
ＦＲＥＥＬＯＡＤ命令を実行すると、負荷データの重量、慣性共に０になります。
本命令は、コントローラのＲＡＭドライブにＳＣＯＬ．ＬＩＢという名称のファイルがないと実行できません。

サンプルプログラム

```
PROGRAM  FREELOADSL
PAYLOAD = HAND
MOVE A1
CLOSE1
DELAY 0.5
MOVE A2 WITH PAYLOAD = HAND + MOTOR
OPEN1
DELAY 0.5
MOVE A3
FREELOAD                    負荷データを０に設定して動作します。
 MOVE A2
 MOVE A3
 END
```

ＧＡＩＮ

機能

各軸のゲイン（サーボ制御）のオン，オフを指定します。

フォーマット

ＧＡＩＮ＝｛＜式＞，＜式＞，＜式＞，＜式＞，＜式＞｝

文例

```
ＧＡＩＮ＝｛０，０，１，０，０｝
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＧＡＩＮ＝｛，，ＯＮ｝
```

解説・注意

各軸のゲイン（サーボ制御）のオン，オフを指定します。
ゲインをオフに指定すると、次に動作命令を実行した時にサーボ制御を切ります。サーボ制御を切った軸は、サーボフリー状態（位置決め制御を行わない状態）になるため、外力により自由に動かす事ができます。このため、ゲインをオフした軸についての位置決めのチェックは行いません。またゲインをオフした軸については、動作命令を実行しても動きません。
本命令は、例えば、はめ合い作業で、流れてくるワークに誤差がある場合に使用します。この場合、Ｚ軸以外の軸のゲインをオフして作業を行います。これにより、ロボットが水平面方向に自由に動ける状態になるため、多少の誤差に対してはＺ軸の動作時にロボットがワークの挿入口に倣って動き、作業を行う事ができます。
ＧＡＩＮ命令は、システム変数として扱われます。本システム変数は５軸分の値を持っています。ゲインのオン，オフの指定は、ＧＡＩＮに続けて｛　｝の中に各軸毎に行います。｛　｝内のデータは、５軸分を”，”で区切って指定し、左から順番に１～５軸に対応します。各軸に対応するデータは、０でゲインオフ、１でゲインオンの指定になります。｛　｝内のデータを省略した場合には、省略した軸に対するゲインの指定はオフと見なします。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。また、＜式＞としてシステム定数ＯＮ，ＯＦＦを使用する事もできます。ＯＮにてゲインのオンを、ＯＦＦにてゲインのオフを指定する事になります。
なお、ＧＡＩＮ命令の実行は必ずロボットの動作と同期をとる必要があります。すなわちサンプルプログラムに示しましたように、前の動作命令が完了した後、ＧＡＩＮ命令を実行してください。

同期をとらずに、ロボット動作中にＧＡＩＮ命令を実行すると正常に動作しない場合があります。なお、ロボットの動作に同期したゲイン指定としては
ＳＥＴＧＡＩＮ命令があります。
自動運転を終了して手動モードに切替えても、ゲインの状態は変化しせん。手動モードでオフしたゲインをオンするには、誘導モード下で対応する軸の誘導キーを押してください。
ゲインのオン、オフの値として、０以下，１以上を指定した時には、それぞれ０，１が指定されたものとします。

サンプル
プログラム

```
PROGRAM  GAINSAMPLE
MOVE A1                          Ａ２への動作時に、Ｚ軸（この場合第３軸）を
WAIT MOTION>=100                 除いた軸のゲインをオフします。
GAIN={OFF,OFF,ON,OFF,OFF}
MOVE A2
OPEN1
DELAY 0.5
MOVE A1
WAIT MOTION >=100
GAIN={ON,ON,ON,ON,ON}
READY
END

PROGRAM GMOVE
G=GAIN                           ３軸以外のゲインをオフにして動作し、動作後
GAIN={,,ON}                      各軸のゲインを以前の状態に戻します。
MOVE P1
WAIT MOTION >=100
GAIN=G
MOVE P2
END
```

ＧＬＯＢＡＬ

機能

大域的な変数領域を指定します。

フォーマット

ＧＬＯＢＡＬ

文例

```
ＧＬＯＢＡＬ
  Ａ＝１０
ＥＮＤ
```

解説・注意

プログラム中どこからでも参照、代入できる大域的な変数を定義します。配列以外の変数は型を特定するために代入形式での初期値設定が必要です。

配列変数は、ＤＩＭ命令による型要素数定義と初期値設定が分離されているため初期値をもたない物もあります。

大域的変数領域には以下の型が使用できます。

整数型，実数型，位置型，座標型，負荷型上記各変数と配列との組み合わせです。

ただし、整数型，実数型の変数定義と整数型配列，実数型配列の初期値設定はグローバルブロックに位置型，座標型，負荷型の変数定義と位置型配列，座標型配列，

負荷型配列の初期値設定はデータブロックに記述してください。

サンプルプログラム

```
GLOBAL
A=1
END
PROGRAM  TEST
A = A + 1
PRINT  “A=”,A,CR
END
```

ＧＯＴＯ

機能

プログラムの実行を指定したラベルを持つ文に移します。

フォーマット

ＧＯＴＯ＜ラベル＞

文例

```
ＩＦ　ＤＩＮ（1）　ＴＨＥＮ　ＧＯＴＯ　Ｌ1
ＧＯＴＯ　ＬＯＯＰ
ＧＯＴＯ　ＲＥＳＴＡＲＴ
```

解説・注意

プログラムの実行を指定したラベルを持つ文に移します。
プログラムの分岐先は、同一プログラム中の文に限ります。指定したラベルが同一プログラム中にない場合にはエラーになります。また、同一プログラム中に同一名称のラベルを持つ文が複数個ある場合には、分岐先は不定になります。
分岐先のラベルは文の先頭に指定します。ラベルには、識別子の後にコロン（：）を付けてください。

サンプルプログラム

```
PROGRAM  GOTOSAMPLE
MOVE A1                 プログラムを最後まで実行すると、プログラムの実行
LOOP:                   はラベルＬＯＯＰまで戻ります。
MOVE A2
MOVE A3
GOTO LOOP
END
```

ＧＯＴＯ（　）

機能	（　）内の式の値により、プログラムの実行を分岐します。
フォーマット	ＧＯＴＯ（＜式＞）＜ラベル＞［，＜ラベル＞］・・・
文例	ＧＯＴＯ（Ｋ）　ＬＡＢＥＬ１，ＬＡＢＥＬ２，ＬＡＢＥＬ３ ＧＯＴＯ（Ｎ１－Ｎ２）　Ｌ１，Ｌ２，Ｌ３，Ｌ４，Ｌ５
解説・注意	（　）内に指定した式の値により、プログラムの実行を分岐します。 （　）内に指定した式の値が、１の時は１番左側に指定したラベルへ、２の時は左から２番目に指定したラベルへというように、式の値に対応したラベルに分岐します。ラベルは１０個まで指定可能で、１０個を超えた分は無視します。 式の値が、指定したラベルの数よりも大きい場合、及び式の値が０以下の場合には、ＧＯＴＯ命令の次の文を実行します。式の値が実数の場合には、小数点以下を切捨てた値にて処理を行います。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 プログラムの分岐先は、同一プログラム中の文に限ります。指定したラベルが同一プログラム中にない場合には、エラーになります。また同一プログラム中に同一名称のラベルを持つ文が複数個ある場合には、分岐先は不定になります。 分岐先のラベルは、文の先頭に指定します。ラベルには、識別子の後にコロン（：）を付けてください。

サンプル
プログラム

```
PROGRAM MAIN
INPUT N
GOTOSAMPL2(N)
END
PROGRAM  GOTOSAMPL2(N)
GOTO(N)L1,L2,L3          引数Ｎの値によって、Ｎが１～３の場合は、それぞれ
RETURN                   Ａ１～Ａ３へと動作し、それ以外の値では、なにも行
L1:                      わずにメインプログラムに戻ります。
MOVE A1
RETURN
L2:
MOVE A2
RETURN
L3:
MOVE A3
RETURN
END
```

ＨＥＲＥ

機能

現在位置を取り出します。

フォーマット

ＨＥＲＥ

文例

```
ＭＯＶＥ　ＨＥＲＥ
Ａ１＝ＨＥＲＥ
Ｘ＝ＨＥＲＥ．Ｘ
```

解説・注意

ワールド座標系での現在位置を取出します。
ＨＥＲＥは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが動作中にＨＥＲＥ命令が実行された場合は、ＨＥＲＥ命令時点でのロボットへの指令位置が現在位置として取り出されます。

注）ＨＥＲＥ命令で取出される位置は、ロボットへの指令位置になります。ロボットの動作中は、ロボットの実際の現在位置は指令位置から遅れている事に注意してください。
現在のロボットの姿勢は、ＨＥＲＥ命令によって取得できます。

```
Ｎ＝ＨＥＲＥ．６
```

等でＮに現在の姿勢の値が入ります。
スカラロボットではＨＥＲＥ．６の値が０で未定義、１で左肩系、２で右肩系となります。

サンプルプログラム

```
PROGRAM  HERESAMPLE
AA=A                        Ａ１からＡ４まで移動中に入力信号１を監視して、
MOVE A                      信号がオンした時の現在位置を、ティーチペンダン
ON DIN(1) DO AA=HERE        トに表示します。
MOVE A1
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.X,AA.Y,AA.Z
END
```

ＨＥＸＩＮ

機能

入力信号をＨＥＸ(16進数)コードとして読込みます。

フォーマット

ＨＥＸＩＮ（＜信号名＞，＜信号長＞）

文例

Ｋ＝ＨＥＸＩＮ（１，２）
Ｊ２＝ＨＥＸＩＮ（Ｎ，Ｎ＋２）
ＧＯＴＯ（ＨＥＸＩＮ（２０，２））Ｌ１，Ｌ２，Ｌ３

解説・注意

信号名から信号長分の入力信号を、ＨＥＸコードとして読込みます。
例えば、Ｋ＝ＨＥＸＩＮ（１，２）では、入力信号の１から２まで２点の状態をＨＥＸコードとして読込みます。
信号長は１～３２の範囲で指定できます。入力信号の番号が大きいものが、高位のビットに対応します。信号は、オン状態で１、オフ状態で０として符号化します。
例）　入力信号の１から１２までが下表の状態の場合、ＨＥＸＩＮ（１，１２）の値は８０９(１６進数で３２９)になります。

入力信号の番号	12	11	10	9	8	7	6	5	4	3	2	1
入力信号の状態	オフ	オフ	オン	オン	オフ	オフ	オン	オフ	オン	オフ	オフ	オン
２進数表現	０	０	１	１	０	０	１	０	１	０	０	１
１６進数表現				3				2				9

＜信号名＞、＜信号長＞には、定数、変数、式、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は、式の中で使用します。

サンプルプログラム

```
PROGRAM  HEXINSAMPL
K=HEXIN(1,7)          入力信号１から７までの状態を符号化入力して、その値に
SPEED=K               より動作速度を設定して動作します。
MOVE A1
MOVE A2
END
```

ＨＥＸＯＵＴ

機能

信号をＨＥＸ(１６進数)コード化して出力します。

フォーマット

ＨＥＸＯＵＴ（＜信号名＞，＜信号長＞，＜式＞）

文例

ＨＥＸＯＵＴ（１，４，３）
ＨＥＸＯＵＴ（Ｎ，Ｎ＋４，Ｋ）

解説・注意

式の値をＨＥＸコード化して信号長で指定した桁数分を、信号名から信号長分の出力信号に出力します。

例えば、ＨＥＸＯＵＴ（１，４，３）では、１から４まで４点の出力信号に数値３をＨＥＸコード化して出力しますので、出力信号１、２がオン状態になります。

出力信号の番号が大きいものが高位のビットに対応します。

信号長は１～３２の範囲で指定できます。式の値が信号長で指定した桁数を超える場合には、上位の桁は無視します。番号が大きい信号が、高位のビットに対応します。

信号は、オン状態を１、オフ状態を０として符号化します。

例）　ＨＥＸＯＵＴ（１，１２，９５２）を実行すると出力信号は、下表のようになります。

１０進数表現												９５２
１６進数表現				３				Ｂ				８
２進数表現	０	０	１	１	１	１	０	１	１	０	０	０
出力信号の番号	12	11	10	9	8	7	6	5	4	3	2	1
出力信号の状態	オフ	オフ	オン	オン	オン	オン	オフ	オン	オン	オフ	オフ	オフ

＜信号名＞、＜信号長＞、＜式＞には、定数、変数、式、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。

ＨＥＸＯＵＴ命令に続けて同じ信号の出力処理を行うと、信号の出力は後から出力したものが有効となります。

同時性を保証できる信号範囲には制約がある事に注意願います。ＤＯＵＴ1、ＤＯＵＴ１０１、ＤＯＵＴ３０１からの１６ｂｉｔ刻みの範囲へのＨＥＸＯＵＴの同時性は保証できますが、その境界を跨ぐＨＥＸＯＵＴの同時性は保証できません。

サンプル
プログラム

```
PROGRAM  HEXOUTSMPL
FOR K=1 TO 4
J=2 ^(K-1)
HEXOUT(1,4,J)
TIMER=0.5
WAIT TIMER==0
HEXOUT(1,4,0)
NEXT K
END
```

出力信号の１から４までを順番に、０．５秒間ずつ出力していきます。

ＩＦ

機能

条件判断を行います。

フォーマット

ＩＦ＜論理式＞ＴＨＥＮ＜文＞［ＥＬＳＥ＜文＞］

文例

ＩＦ　ＤＩＮ（１）　ＴＨＥＮ　Ｋ＝Ｋ＋１　ＥＬＳＥ　Ｋ＝０

解説・注意

指定された論理式を判断して、条件が成立した場合にはＴＨＥＮに続く文を、条件が成立しなかった場合にはＥＬＳＥに続く文を実行します。
ＩＦ命令にてＥＬＳＥは省略する事ができます。ＥＬＳＥ＜文＞を省略すると、条件が成立しなかった場合にＩＦ文の次の命令を実行します。
ＴＨＥＮ，ＥＬＳＥに続く＜文＞には、ＰＲＯＧＲＡＭ，ＥＮＤ，ＩＦ，ＦＯＲ，ＮＥＸＴ及びＷＡＩＴ命令は使用できません。
ＩＦ～ＴＨＥＮ～ＥＬＳＥ～は１つの文になるため、同じ行に記述しなければなりません。（例えば、ＥＬＳＥ以降を行を変えて書くような事はできません。）

サンプルプログラム

```
PROGRAM  IFSAMPLE
IF DIN(1) THEN K=1 ELSE K=0
MOVE A1
MOVE A2
MOVE A3
PRINT TP,K,CR
END
```

入力信号１がオンの場合にはＫ＝１、入力信号１がオフの場合にはＫ＝０にします。

ＩＧＮＯＲＥ

機能

ＯＮ命令で指定した条件の監視を解除します。

フォーマット

ＩＧＮＯＲＥ　＜監視条件＞

文例

ＩＧＮＯＲＥ　ＤＩＮ（１）
ＩＧＮＯＲＥ　ＴＩＭＥＲ

解説・注意

ＯＮ命令にて指定した条件の監視を解除します。
＜監視条件＞には、条件の監視を指定したＯＮ文の論理式と同じ内容を記述します。
ただし、ＩＧＮＯＲＥできるＯＮ条件文は、ＩＧＮＯＲＥ命令で記述させている行の前に記述してください。またこの時同一のＯＮ条件が複数行に記述されてはいけません。
ＩＧＮＯＲＥ文の実行にて、条件の監視は解除します。
条件監視については、ＯＮ命令を参照してください。

サンプルプログラム

```
PROGRAM MAIN
IGNORESMPL
MOVE P
MOVE P2
END
PROGRAM  IGNORESMPL
ON DIN(1) PAUSE DO RETURN      入力信号１がオンになったら実行中の動作の
MOVE A1                        完了にてメインプログラムに戻ります。
MOVE A2                        Ａ１～Ａ２への移動終了にて、入力信号１の
WAIT MOTION >=100              監視を解除します。
IGNORE DIN(1)
RETURN
END
```

ＩＮＩＴＰＬＴ

機能

パレットの初期化

フォーマット

ＩＮＩＴＰＬＴ（＜パレット番号＞，＜ｉ＞，＜ｊ＞，＜ｋ＞）

文例

ＩＮＩＴＰＬＴ（１，５，４，３）

解説・注意

パレタイズ命令（ＭＯＶＥＰＬＴ）を実行する為、パレットを初期化します。

パレット番号：パレットに対し１から順に付けた番号（１以上の任意の整数）

ｉ　　：パレット原点からＩ点までの要素数（１以上の任意の整数）

ｊ　　：パレット原点からＪ点までの要素数（１以上の任意の整数）

ｋ　　：パレット原点からＫ点までの要素数（１以上の任意の整数）

ｉ，ｊ，ｋが０以下の値の時プログラムは”ERR!! ELEMENT IS TOO SMALL”とＴＰに表示しプログラムは停止します。

ＩＮＩＴＰＬＴ命令はダイナミックリンクライブラリで提供しています。

よって、命令を実行するにはＧＬＯＢＡＬエリアで**ライブラリの組み込み宣言およびグローバル変数宣言**をする必要があります。

詳細は付録Ｇ－１パレタイズライブラリを参照してください

サンプルプログラム

```
GLOBAL
LOADLIB PALLET.LIB              ライブラリの組み込み宣言
DIM PLTP(1,7)  AS POINT         グローバル変数宣言
END
PROGRAM SAMPLE
INITPLT(1,3,4,2)                PLTP(1,1)～PLTP(1,4)の教示点で３×４×２に初期化
MOVEPLT(1,1,0,0,0,0)    パレット１、要素番号１に移動
END
```

ＩＮＰＵＴ

機能

通信チャネルからデータを読込みます。

フォーマット

ＩＮＰＵＴ［｛ＣＯＭ０｜ＣＯＭ１｜ＴＰ｝，］＜変数＞［，＜変数＞］・・・

文例

ＩＮＰＵＴ　Ｋ１，Ｋ２，Ｋ３
ＩＮＰＵＴ　ＣＯＭ１，Ｋ

解説・注意

通信チャネルからデータを読込みます。
読込めるデータは、実数型と整数型のデータです。
通信チャネルとしては、ＣＯＭ０，ＣＯＭ１，ＴＰの中から１つを指定します。
ＣＯＭ０及びＴＰはティーチペンダント専用の通信チャネルです。ＣＯＭ１コントローラのＣＯＭ１通信チャネルに対応します。
ＩＮＰＵＴ命令にて通信チャネルを指定しないと、ティーチペンダント専用の通信チャネルに対して、データの読込みを行います。
ＩＮＰＵＴ命令を実行すると、通信チャネルからデータが読込まれるまで待ちます。読込まれたデータは、指定した変数に代入されます。読込んだデータの数が指定した変数の数より多い場合は，多い分は無視します。データの数が少ない場合には不足分のデータの入力を待ちます。
ティーチペンダントから入力を行う場合には、指定した個数の実数を”，”で区切って入力します。入力の終了は、［実行］キーによります。ティーチペンダント以外からデータを読込む場合は、通信の終了時に処理が行われます。データ通信については、「通信編」にて説明していますので、そちらを参照してください。
なおアーム動作中の停止操作後、本命令は実行されません。

サンプル
プログラム

```
PROGRAM  INPUTSMPL
PRINT COM0,"*** INPUT N1,N2,N3 ***"
INPUT COM0,N1,N2,N3
PRINT (N1+N2+N3)/3, CR
END
```

ティーチペンダントから３つの数値
Ｎ１～Ｎ３を読込み、その平均を
ティーチペンダントに表示します。

ＩＮＴ

機能

数値を整数に変換します。

フォーマット

ＩＮＴ（＜式＞）

文例

ＡＫ＝ＩＮＴ（－２０．３４５）
Ｎ＝ＩＮＴ（Ｋ）
Ｊ１＝Ｋ－ＩＮＴ（Ｎ－２８．５）

解説・注意

（ ）内の式の演算結果を整数に変換します。
実数は小数点以下を切捨てて、整数に変換します。
本命令は変数のデータ型を、特に整数型として指定したい場合に使用します。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
INTSAMPLE(2,30,K)
PRINT TP,K,CR
END
PROGRAM  INTSAMPL(L,R,K)
K=INT(L*COS(R))
RETURN
END
```

引数Ｌ，ＲからＬ＊ＣＯＳ（Ｒ）の値を計算して、その小数点以下を切捨てた整数値を引数Ｋとしてメインプログラムに返します。

ＫＩＬＬ

機能	マルチタスク動作を停止させます。
フォーマット	ＫＩＬＬ（＜式＞）
文例	ＫＩＬＬ（ＴＳＫＩＤ）
解説・注意	（　）内の式の演算結果で指定されるタスク番号を持つタスクの実行を停止します。プログラム開始時に自動的に生成されるメインタスクのタスクＩＤ（１に固定される）や存在しないタスクＩＤを指定した場合には、ＮＯＰ動作となります。ＫＩＬＬ命令によって停止したタスクはＴＡＳＫ命令によって再起動されるまで動作しません。その場合、タスク番号は異なる値となります。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。停止を指示されたタスクは、実行権が与えられた時点でシステムから削除されます。つまりシステム変数ＳＷＩＴＣＨがＤＩＳＡＢＬＥ状態ではタスク切り替えが発生しないため、他のタスクをＫＩＬＬできません。システムスイッチＳＷＩＴＣＨをＥＮＡＢＬＥ状態でＫＩＬＬ命令を実行してください。またステップ実行中、本命令は無効です。

サンプル
プログラム

```
GLOBAL
MAXTASK=2
K=0
END
PROGRAM MAIN
TID1=0
LOOP:                                                    客先入力信号１がＯＮの時に
IF DIN(1) AND TID1==0 THEN TID1=TASK(“SUB1”)             タスクを生成し、ＯＦＦの時は
IF DIN(-1) AND TID1<>0 THEN KILL(TID1)ELSE GOTO LOOP1    タスクを消滅させます。
TID1=0
LOOP1:
MOVEA 1,-90
MOVEA  1,90
GOTO LOOP
END
PROGRAM SUB1
K = K + 1
PRINT K,CR
END
```

ＬＡＴＣＨ（ＴＳ３０００オプション）

機能

専用入力ポート信号による位置ラッチ機能の有効無効を設定します。

フォーマット

ＬＡＴＣＨ

文例

ＤＩＳＡＢＬＥ　ＬＡＴＣＨ
ＥＮＡＢＬＥ　　ＬＡＴＣＨ

解説・注意

位置ラッチを行うための専用入力ポート信号の監視を行うか、行わないかを指定します。
システムスイッチのオン、オフには、ＥＮＡＢＬＥ、ＤＩＳＡＢＬＥ命令を使用します。
ＥＮＡＢＬＥ　ＬＡＴＣＨで専用入力ポート信号による位置ラッチ機能が有効になり、
ＤＩＳＡＢＬＥ　ＬＡＴＣＨで無効になります。
初期状態ではＤＩＳＡＢＬＥ　ＬＡＴＣＨになっています。
専用入力ポート信号の立ち上がりエッジ検出を指定し、専用ポート信号がＯＮ中に
ＥＮＡＢＬＥ　ＬＡＴＣＨとした場合、位置ラッチ機能の動作は保証されません。
専用入力ポート信号の立ち下がりエッジ検出を指定し、専用ポート信号がＯＦＦ中に
ＥＮＡＢＬＥ　ＬＡＴＣＨとした場合も同様です。　ただし、このような場合もエラーとはならず、処理を継続します。
ＤＩＮ命令で専用入力ポート信号の状態を確認してから、ＥＮＡＢＬＥするようにプログラムして下さい。　専用入力ポート信号は５３～５６に割り当てられています。
また、ＬＡＴＣＨＳＩＧ１，２の状態を確認してから、ラッチ位置を取得して下さい。
本機能を使用するためには拡張基板が必要になります。拡張基板のないシステムで
ＥＮＡＢＬＥ　ＬＡＴＣＨを実行した場合はエラーになります。
本機能はTSL3000に追加することは出来ません。

サンプルプログラム

```
PROGRAM LATCHSMP
 DISABLE NOWAIT
 MOVE A0
LATCHTRG1 = 1
IF DIN(49) THEN GOTO FINI
ENABLE LATCH
MOVES A1
WAIT MOTION >= 100
IF LATCHSIG1 == 1 THEN LP = LATCHPSN1 ELSE LP = HERE
DISABLE LATCH
FINI:
MOVE LP
END
```

ＬＡＴＣＨＴＲＧ１～８（ＴＳ３０００オプション）

機能

専用入力ポート信号による位置ラッチ機能の検出エッジ方向を設定します。

フォーマット

ＬＡＴＣＨＴＲＧ１＝｛ ０ ｜ １ ｝
ＬＡＴＣＨＴＲＧ３＝｛ ０ ｜ １ ｝

文例

```
ＬＡＴＣＨＴＲＧ１＝１
Ａ＝ＬＡＴＣＨＴＲＧ８
```

解説・注意

専用入力ポート信号による位置ラッチのトリガとなる検出エッジ方向を指定するシステム変数です。

ＬＡＴＣＨＴＲＧ１～８は専用入力ポートの検出エッジ方向を指定します。
エッジ方向の指定は０で立ち下がり（ＯＮからＯＦＦ）、１で立ち上がり（ＯＦＦからＯＮ）となります。
ＬＡＴＣＨＴＲＧ１～８には整数以外の数値は使用できません。 また、０以外の数値が指定された場合は１が指定されたものとします。
本システム変数を参照すると、現在の検出エッジ方向が参照できます。
検出エッジ方向の初期値は１（立ち上がりエッジ検出）です。

ＥＮＡＢＬＥ ＬＡＴＣＨ中に検出エッジ指定を変更した場合､位置ラッチ機能の動作は保証されません。 ただし、このような場合もエラーとはならず、処理を継続します。
プログラムの動作を明確にするために､ＤＩＳＡＢＬＥ ＬＡＴＣＨとしてから検出エッジ方向を変更して下さい。

本機能を使用するためには拡張基板が必要になります。ただし、拡張基板のないシステムでＬＡＴＣＨＴＲＧ１～８を指定した場合も動作に影響はしません。
本機能はTSL3000に追加することは出来ません。

サンプル
プログラム

```
PROGRAM LATCHSMP
DISABLE NOWAIT

MOVE A0
LATCHTRG1 = 1
IF DIN(49) THEN GOTO FINI
ENABLE LATCH
MOVES A1
WAIT MOTION >= 100
IF LATCHSIG1 == 1 THEN P1 = LATCHPSN1 ELSE P1 = HERE
DISABLE LATCH

LATCHTRG2 = 0
IF DIN(-50) THEN GOTO FINI
ENABLE LATCH
MOVES A2
WAIT MOTION >= 100
IF LATCHSIG2 == 1 THEN P2 = LATCHPSN2 ELSE P2 = HERE
DISABLE LATCH

FINI:
MOVE A0
END
```

ＬＡＴＣＨＳＩＧ１～８（ＴＳ３０００オプション）

機能

専用入力ポート信号による位置ラッチの状態を参照します。

フォーマット

ＬＡＴＣＨＳＩＧ１
ＬＡＴＣＨＳＩＧ５

文例

Ａ＝ＬＡＴＣＨＳＩＧ１
ＩＦ　ＬＡＴＣＨＳＩＧ２＝＝０　ＴＨＥＮ　ＧＯＴＯ　ＥＲＲ

解説・注意

専用入力ポート信号による位置ラッチが行われたかを参照します。
ＬＡＴＣＨＳＩＧ１～８は専用入力ポートの位置ラッチ状態を参照します。
位置ラッチが行われている場合は１を、行われいてない場合は０を返します。
ＤＩＳＡＢＬＥ　ＬＡＴＣＨ中は専用信号の状態に関係なく０になります。 また、立ち上がりエッジ検出中に専用信号がＯＦＦになると、ＬＡＴＣＨＳＩＧ１～８は０になります。　同様に、立ち下がりエッジ検出中に専用信号がＯＮになると、ＬＡＴＣＨＳＩＧ１～８は０になります。

ＬＡＴＣＨＳＩＧ１～８は参照のみ可能で、代入することはできません。
また、ＬＡＴＣＨＳＩＧ１～８はＯＮ条件として使用することはできません。
ＯＮ条件で信号を参照したい場合にはＤＩＮ命令で４９～５６を参照してください。

本機能を使用するためには拡張基板が必要になります。拡張入出力基板のないシステムでＬＡＴＣＨＳＩＧ１～８を参照した場合は常に０を返します。

注）ＬＡＴＣＨＳＩＧ～８で取り出されるラッチ状態は、現在のロボットの状態になります。　ロボットの実際の動作はＳＣＯＬプログラムの処理から遅れていることに注意して下さい。
ラッチ状態およびラッチ位置の取り出しは、ＷＡＩＴ　ＭＯＴＩＯＮ＞＝　１００の実行後に行うことをお奨めします。
本機能はTSL3000に追加することは出来ません。

サンプル
プログラム

```
PROGRAM LATCHSMP
DISABLE NOWAIT
MOVE A0
LATCHTRG1 = 1
IF DIN(49) THEN GOTO ERR
ENABLE LATCH
MOVES A1
WAIT MOTION >= 100
IF LATCHSIG1 == 0 THEN GOTO ERR
LP = LATCHPSN1
DISABLE LATCH
GOTO FINI
ERR:
PRINT "LATCH ERROR", CR
LP = HERE
FINI:
MOVE LP
END
```

ＬＡＴＣＨＰＳＮ１～８（ＴＳ３０００オプション）

機能

位置ラッチ機能によるラッチ位置を取り出します。

フォーマット

ＬＡＴＣＨＰＳＮ１
ＬＡＴＣＨＰＳＮ２

文例

Ｐ１＝ＬＡＴＣＨＰＳＮ１
ＡＸ＝ＬＡＴＣＨＰＳＮ２．Ｘ

解説・注意

専用入力ポート信号のエッジを検出した時のワールド座標系での位置を取り出します。
ＬＡＴＣＨＰＳＮ１～８は専用入力ポート１～８のエッジ検出位置をワールド座標系で取り出します。
検出エッジ方向はＬＡＴＣＨＴＲＧ１～８で指定して下さい。　エッジ検出処理はＥＮＡＢＬＥ　ＬＡＴＣＨで有効に、ＤＩＳＡＢＬＥ　ＬＡＴＣＨで無効になります。
ＬＡＴＣＨＰＳＮ１～８の参照は、対応するＬＡＴＣＨＳＩＧ１～８が１である時に行って下さい。　ＬＡＴＣＨＳＩＧ１～８が０である間もＬＡＴＣＨＰＳＮ１～８の参照は可能ですがその値は保証されません。

本機能を使用するためには拡張基板が必要になります。拡張基板のないシステムでＬＡＴＣＨＰＳＮ１～８を参照した場合はワールド座標系原点位置を示します。

本機能はTSL3000に追加することは出来ません。

注）　ＬＡＴＣＨＳＩＧ１～８で取り出されるラッチ状態は、現在のロボットの状態になります。　ロボットの実際の動作はＳＣＯＬプログラムの処理から遅れていることに注意して下さい。
ラッチ状態およびラッチ位置の取り出しは、ＷＡＩＴ　ＭＯＴＩＯＮ＞＝　１００の実行後に行うことをお奨めします。

サンプル
プログラム

```
PROGRAM LATCHSMP
DISABLE NOWAIT
MOVE A0
LATCHTRG1 = 1
IF DIN(49) THEN GOTO ERR
ENABLE LATCH
MOVES A1
WAIT MOTION >= 100
IF LATCHSIG1 == 0 THEN GOTO ERR
LP = LATCHPSN1
DISABLE LATCH
GOTO FINI
ERR:
PRINT "LATCH ERROR", CR
LP = HERE
FINI:
MOVE LP
END
```

ＬＥＦＴＹ

機能

ロボットの姿勢を左肩系にするためのシステム定数です。

フォーマット

ＬＥＦＴＹ

文例

ＣＯＮＦＩＧ＝ＬＥＦＴＹ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＣＯＮＦＩＧ＝ＬＥＦＴＹ

解説・注意

ＣＯＮＦＩＧ命令と共に使用して、ロボットの姿勢を左肩系に設定します。
本システム定数は、１という値を持っています。プログラムの式の中で定数１として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。
直角座標ロボットでは、ロボットの姿勢の指定は無視します。
ロボットの姿勢については、ＣＯＮＦＩＧ命令を参照してください。

サンプルプログラム

```
PROGRAM  LEFTYSMPL
CONFIG=LEFTY              ロボットの姿勢を左肩系として動作します。
MOVE A1
MOVE A2
END
```

ＬＮ

機能

自然対数の値を計算します。

フォーマット

ＬＮ（＜式＞）

文例

Ｋ＝ＬＮ（１００）
Ｊ１＝１－ＬＮ（５０－Ｄ）

解説・注意

（　）内の自然対数の値を計算します。なおＬＮ（０）の実行結果はエラーにならず０になりますのでご注意ください。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
LNSMPL(3,K)
PRINT TP,K,CR
END
PROGRAM  LNSMPL(N,K)
K=10 ^ N                引数Ｎで与えられた常用対数の値を自然対数の値に変
K=LN(K)                 換して、その結果を引数Ｋとしてメインプログラムに
RETURN                  返します。
END
```

ＬＯＡＤＬＩＢ

機能

ダイナミックリンクライブラリを読み込みます。

フォーマット

ＬＯＡＤＬＩＢ　ファイル名

文例

ＬＯＡＤＬＩＢ　ＰＡＬＬＥＴ．ＬＩＢ

解説・注意

ＬＯＡＤＬＩＢ命令は必ずグローバル部でライブラリの読み込みを宣言してください。
ライブラリのファイル名称は　＊＊＊＊＊＊＊＊．ＬＩＢとしてください。
１プログラムで読み込み可能なライブラリは５つまでです。

ダイナミックリンクライブラリについては　“2.8.3 ライブラリ”、“付録Ｇ”を参照してください。

サンプルプログラム

```
GLOBAL
LOADLIB PALLET.LIB              ライブラリの組み込み宣言
DIM PLTP(1,7) AS POINT
END
PROGRAM MAIN
INPUT N
GOTOSAMPL2(N)
END

PROGRAM  GOTOSAMPL2(N)          ライブラリファイルの例
GOTO(N)L1,L2,L3
RETURN
L1:
MOVE A1
RETURN
L2:
MOVE A2
RETURN
L3:
MOVE A3
RETURN
END
```

ＬＯＧ１０

機能	常用対数の値を計算します。
フォーマット	ＬＯＧ１０（＜式＞）
文例	Ｋ＝ＬＯＧ１０（１００） Ｊ１＝１－ＬＯＧ１０（５０－Ｄ）
解説・注意	（　）内の常用対数の値を計算します。（なおＬＯＧ１０（０）の実行結果はエラーにならず不定値になりますのでご注意ください。） ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 本命令は式の中で使用します。
サンプルプログラム	PROGRAM　MAIN LOG10SAMPLE(3,K) PRINT TP,K,CR END PROGRAM　LOG10SAMPLE(N,K) K=EXP(N)　　引数Ｎで与えられた自然対数の値を常用対数の値に変 K=LOG10(K)　　換して、その結果を引数Ｋとしてメインプログラムに RETURN　　返します。 END

ＭＡＸＴＡＳＫ

機能

マルチタスク機能を使用しているプログラムで、同時に存在できるタスクの最大個数を指定します。

フォーマット

ＭＡＸＴＡＳＫ

文例

ＭＡＸＴＡＳＫ＝４

解説・注意

この変数はグローバルブロックでのみ使用可能です。
この値は通常、“ＴＡＳＫ”命令の個数＋１の数を指定して下さい。最大は４です。
ＭＡＸＴＡＳＫへの代入が複数宣言されていれば、最後の値以外を無視します。
マルチタスク機能を使用しなければ、この変数を使用する必要はありません。この場合、１と見なしメインタスクのみの作業領域を確保します。
コントローラの作業領域は、この変数で等分して、各タスクに割り当てます。従って大きな値を指定しておくと、１つのタスクが使用できる作業領域が少なくなりサイズが大きなプログラムを実行できなくなりますので注意してください。

サンプルプログラム

```
GLOBAL
  A = 0
  MAXTASK=2              サブタスクを１つ使用するので値は２をセットします。
END
PROGRAM MAIN
ID = 0
ID = TASK("SUB")         タスク命令が重複で呼ばれないようにGOTO文でループ
LOOP:                     を組みます。
IF DIN(1) THEN A=1 ELSE A=0
GOTO LOOP
END
PROGRAM SUB
 ENABLE NOWAIT
 IF A==0 THEN PRINT "A",A,CR
END
```

ＭＯＤ

機能

剰余の値を計算します。

フォーマット

＜式＞　ＭＯＤ　＜式＞

文例

Ｎ＝Ｋ　ＭＯＤ　３

Ｊ＝Ｋ＋（Ｌ　ＭＯＤ　Ｍ）

解説・注意

左辺の式の値を右辺の式の値で割った余りを計算します。

＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。

本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
MODSAMPLE(5.0,3.0,K)
PRINT TP,K,CR
END
PROGRAM  MODSAMPLE(N1,N2,K)
K=N1 MOD N2              引数Ｎ１をＮ２で割った余りを、引数Ｋとしてメイ
RETURN                   ンプログラムに返します。
END
```

ＭＯＤＥ

機能

システムの運転モードを参照します。

フォーマット

ＭＯＤＥ

文例

ＩＦ　ＭＯＤＥ＜＞ＣＯＮＴ　ＴＨＥＮ　ＳＴＯＰ

解説・注意

システムの運転モードを参照します。
システムの運転モードは、ＭＯＤＥの値が０ならば連続運転モード，１ならばサイクル運転モード、２ならばセグメント運転モードになります。
システムの運転モードを参照する場合に、システム定数のＣＯＮＴ，ＣＹＣＬＥ、ＳＥＧＭＥＮＴを使用する事ができます。システムの運転モードは、
ＭＯＤＥ＝＝ＣＯＮＴで連続運転モード、ＭＯＤＥ＝＝ＣＹＣＬＥでサイクル運転モード、ＭＯＤＥ＝＝ＳＥＧＭＥＮＴでセグメント運転モードになります。
モニタコマンドではＭＯＤＥ　ＭＯＴＩＯＮにてセグメント運転モードを指定できます。

サンプルプログラム

```
PROGRAM  MODESAMPLE
MOVE A1                          システムが連続運転モードで動作していな
MOVE A2                          ければ、プログラムの実行を停止します。
IF MODE<>CONT THEN STOP
MOVE A3
END
```

ＭＯＴＩＯＮ

機能

動作の実行量を参照します。

フォーマット

ＭＯＴＩＯＮ

文例

Ｋ＝ＭＯＴＩＯＮ
ＯＮ　ＭＯＴＩＯＮ＞＝５０　ＤＯ　ＤＯＵＴ（１）

解説・注意

ロボットが、１つの動作の何％まで動いたかを参照します。
実行量は、ロボットが実行中の動作距離に対して、どれだけロボットが移動したかのパーセンテージで与えられます。実行量の算出には、その動作で最も移動量の大きい軸を基準にします。実行量は実数値として与えられます。
本命令とＯＮ命令を組合わせる事により、ロボットの動作途中で信号を出力する事ができます。本命令は式の中で使用します。

注）　ＭＯＴＩＯＮ命令で参照される移動量はロボットへの指令位置になります。実際のロボット現在位置は、ロボットの動作中は指令位置から遅れている事に注意してください。
比較演算子に＝＝は使用できませんので注意してください。
ＥＮＡＢＬＥ　ＰＡＳＳとの併用する場合、使い方によっては無限ﾙｰﾌﾟ動作を行うので注意してください。
下記例はＰ１からＰ２へのﾊﾟｽ軌道生成待ちが発生してしまうため、無限ﾙｰﾌﾟになります。
ＷＡＩＴ文に置き換えることにより、無限ﾙｰﾌﾟは回避可能です。

```
ENABLE PASS                                  ENABLE PASS
  PASS=50                                    PASS=50
  MOVE P1                          →         MOVE P1
LOOP1:                                       WAIT MOTION >= 95
  IF MOTION < 95 THEN GOTO LOOP1             MOVE P2
  MOVE P2
```

サンプルプログラム

```
PROGRAM  MOTIONSMPL
  ENABLE NOWAIT
  ON MOTION>=50 DO DOUT(1)
  MOVE A1
  ON MOTION>=80 DO DOUT(2)
  MOVE A2
END
```

Ａ１への移動が５０％以上進んだら信号１を出力し、Ａ２への移動が８０％以上進んだら信号２を出力します。

ＭＯＴＩＯＮＴ

機能

動作の実行時間を参照します。

フォーマット

ＭＯＴＩＯＮＴ

文例

Ｋ＝ＭＯＴＩＯＮＴ
ＯＮ　ＭＯＴＩＯＮＴ＞＝５　ＤＯ　ＤＯＵＴ（１）

解説・注意

ロボットが、１つの動作を開始してからの時間を参照します。
実行時間は、秒単位の実数値で与えられ、ロボットが位置決め完了すると０になります。
本命令とＯＮ命令を組合わせる事により、ロボットの動作途中で信号を出力することができます。また、ロボットの１動作当りの移動時間を監視して、規定時間以上かかった場合には、異常処理を行う事もできます。
本命令は式の中で使用します。
比較演算子に＝＝は使用できませんので注意してください。

注）　ＭＯＴＩＯＮ命令で参照される移動量はロボットへの指令位置になります。実際のロボット現在位置は、ロボットの動作中は指令位置から遅れている事に注意してください。
比較演算子に＝＝は使用できませんので注意してください。
ＥＮＡＢＬＥ　ＰＡＳＳとの併用する場合、使い方によっては無限ﾙｰﾌﾟ動作を行うので注意してください。
下記例はＰ１からＰ２へのﾊﾟｽ軌道生成待ちが発生してしまうため、無限ﾙｰﾌﾟになります。
ＷＡＩＴ文に置き換えることにより、無限ﾙｰﾌﾟは回避可能です。

```
ENABLE PASS                                  ENABLE PASS
  PASS=50                                    PASS=50
  MOVE P1                          →         MOVE P1
LOOP1:                                       WAIT MOTIONT >= 5
  IF MOTIONT < 5 THEN GOTO LOOP1             MOVE P2
  MOVE P2
```

サンプルプログラム

```
PROGRAM  MOTIONTSMPL
  ENABLE NOWAIT              Ａ１への動作に１０秒以上かかった場
ON MOTIONT>=10 DO DOUT(1)    合には、信号１を出力します。
  MOVE A1
  MOVE A2
END
```

ＭＯＶＥ

機能

ロボットを指定した位置へ動かします。

フォーマット

ＭＯＶＥ＜位置＞［ＷＩＴＨ節］

文例

ＭＯＶＥ　Ａ１
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＳＰＥＥＤ＝５０

解説・注意

ロボットを指定した位置へ同期動作で動かします。
ロボットの各関節軸は、同時に動き始めて同時に停止します。各関節軸の速度は、最も移動時間のかかる軸に合せて動作が終了するように計算されます。このような動作を同期動作（関節角補間）と呼びます。
＜位置＞には、位置型データを用いる事ができます。また、次のようにして＜位置＞として座標値を直接指定する事もできます。
なお、＜位置＞には座標型データや負荷型データを用いることはできません。

ＭＯＶＥ　ＰＯＩＮＴ（Ｘ，Ｙ，Ｚ，Ｃ，Ｔ，＜姿勢＞）
ＭＯＶＥ　｛Ｘ，Ｙ，Ｚ，Ｃ，Ｔ｝
　　ＷＩＴＨ　ＣＯＮＦＩＧ＝＜姿勢＞

（データ型を明確にするために、ＰＯＩＮＴ命令を使用するようにしてください。）

Ｘ，Ｙ，Ｚ，Ｃ，Ｔ　：　Ｘ，Ｙ，Ｚ，Ｃ，Ｔの各座標値を実数で指定します。
（ｍｍ，ｄｅｇ単位）

＜姿勢＞　：　０～２の整数値でロボットの姿勢を指定します。
（０：未定義，１：左肩系，２：右肩系）

各要素には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型の変数は使用できません。＜姿勢＞として０以下，２以上を指定すると、それぞれ０、２が指定されたものとして処理されます。各要素は省略する事もできますが、省略した要素は０と見なされます。

例）　MOVE POINT (100, 100, 0, 0, 0, RIGHTY)
　　MOVE POINT (100, 100): Z～＜姿勢＞まで全て０と見なします。

ティーチペンダントから位置データを教示すると、教示した時に指定したワーク座標系のデータも同時に記憶します。動作命令の実行時には、ワーク座標系はその位置データを教示した時のものに変わります。ベース，ツール座標系については、命令実行時の設定をそのまま使用します。

＜位置＞として座標値を直接指定した場合（ＤＥＳＴ，ＨＥＲＥ，ＰＯＩＮＴ命令等で位置データを合成した場合も含む）には、命令実行時のワーク座標系にて移動を行います。

ロボットが動作を行う時には、その時点でのシステム変数の設定値により、速度，加減速等の動作条件が決まります。１つの動作だけに関して動作条件を変更するためには、ＷＩＴＨ節を用いて動作条件を指定します。ＷＩＴＨ節の使い方は、ＷＩＴＨ命令を参照してください。

サンプル
プログラム

```
PROGRAM  MOVESAMPLE
MOVE A1                    Ａ１へ同期動作で移動します。
MOVE A2
END
```

ＭＯＶＥＡ

機能

ロボットの指定した関節軸を指定位置へ動かします。

フォーマット

ＭＯＶＥＡ＜軸＞，＜絶対位置＞［ＷＩＴＨ節］

文例

ＭＯＶＥＡ　１，６０
ＭＯＶＥＡ　３，０　ＷＩＴＨ　ＧＡＩＮ＝｛，，ＯＮ｝

解説・注意

ロボットの指定した関節軸を指定位置へ動かします。このような動作を絶対単軸動作と呼びます。
＜軸＞は、１～５までの整数値でロボットの関節軸を指定します。指定軸以外の軸は動作しません。
＜絶対位置＞には、ロボットの関節軸の原点からの位置を、回転軸は度単位で、直動軸はミリメートル単位で指定します。関節リミットで設定した範囲外を指定すると、関節リミットのすぐ内側まで動作します。
＜軸＞、＜絶対位置＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型の変数は使用できません。
＜軸＞として１～５以外、及び実装していない軸を指定すると無視され、ロボットは動作しません。
ロボットが動作を行う時には、その時点でのシステム変数の設定値により、速度、加減速等の動作条件が決まります。１つの動作だけに関して動作条件を変更するためには、ＷＩＴＨ節を用いて動作条件を指定します。ＷＩＴＨ節の使い方は、ＷＩＴＨ命令を参照してください。

サンプルプログラム

```
PROGRAM  MOVEASAMPL
MOVEA 1,0                    第１軸を０度の位置へ動かします。
MOVEA 2,90
END
```

ＭＯＶＥＣ

機能

ロボットの手先を指定した通過位置を通り、指定した位置へ円弧補間で動かします。

フォーマット

ＭＯＶＥＣ　＜通過位置＞　＜位置＞　［ＷＩＴＨ節］

文例

ＭＯＶＥＣ　Ａ１　Ａ２
ＭＯＶＥＣ　Ａ１　Ａ２　ＷＩＴＨ　ＳＰＥＥＤ＝１０

解説・注意

ロボットの手先は、現在位置と＜通過位置＞、＜位置＞を結ぶ円弧軌道を通るように動作します。また、手先の向きは現在位置から＜位置＞での向きに角速度一定で動作します。

＜通過位置＞、＜位置＞には位置型データを指定します。またＭＯＶＥ，ＭＯＶＥＳ命令と同様にＰＯＩＮＴ命令により座標値を直接指定することもできます。

なお、＜通過位置＞、＜位置＞には座標型データや負荷型データを用いることはできません。

［ＷＯＲＫ座標系］

ＷＯＲＫ座標系は＜通過位置＞、＜位置＞共に、教示したときのもので動作します。ＷＩＴＨ節によりＷＯＲＫ座標系を指定したときは、＜通過位置＞、＜位置＞共に指定したＷＯＲＫ座標系で動作します。また、＜通過位置＞、＜位置＞に座標値を直接指定したときは、命令実行時のＷＯＲＫ座標系で動作します。

［ＷＩＴＨ節］

ロボットが動作を行うときには、その時点でシステム変数の設定値により、速度、加速度などの動作条件が決まります。一つの動作だけに関して動作条件を変更するためには、ＷＩＴＨ節を使用して動作条件を指定します。ＷＩＴＨ節の使い方は、ＷＩＴＨ命令を参照してください。

［円弧補間中断後の再開時の制限］

“ＯＮ～ＢＲＥＡＫ～ＤＯ～”、フィードホールド、非常停止、故障等により円弧補間動作が中断した後、プログラムをリセットせずに実行を継続（“ＯＮ～ＢＲＥＡＫ～ＤＯ～”で動作を中断した時にはＲＥＳＵＭＥ命令により継続）すると中断した命令を再実行しますが、このとき、円弧補間動作は＜位置＞への直接補間動作で動くので注意してください。

［３点の位置関係の制限］

円弧を形成する３点の位置（現在位置，＜通過位置＞，＜位置＞）の内、３点または２点が一致していたり、極めて接近している場合は、所望の円弧と異なった動作を行う場合がありますので注意してください。またショートカット動作中で円弧補間を使用する場合は、ロボットの軌跡が円弧の接線方向につながるようにしてください。ロボットの軌跡が円弧の接線となす角度が鋭角になると、ショートカット動作の円弧補間の継目でロボットに急激な加速度が加わる場合があります。

［ツールオフセット］

オフセットのあるツールを使用して、ツールの向きを変えながら直線・円弧補間を行う場合には、ツール座標系を使用する工具に合せて設定しないと、所望の動作が得られない場合があります。ツール座標系は位置の教示時にも使用します。位置の教示を行う前にツール座標系が正しく設定されていることを確認してください。（ツール座標系の選択方法は取扱説明書・操作編「ツール座標選択」を参照してください。）このようにオフセットのあるツールを使用する場合には、原則として全ての位置をツール座標系を設定した上で教示してください。また、ロボット言語プログラムの先頭ではＴＯＯＬ命令を使用して、プログラム中で使用するツール座標系を確実に指定するようにしてください。

```
例）  PROGRAM MAIN
        TOOL=TOOL1        以降ツール座標系として“TOOL1 ”を使用します。
          …
        END
```

［５軸目追加（オプション）について］

５軸目追加（オプション）を行った場合、５軸目に関しては円弧補間動作を行えませんのでご注意ください。他の軸間では円弧補間動作を行います。

サンプルプログラム

```
PROGRAM  MOVECSAMPL
 MOVE A1
 MOVEC A2 A3              A1からA2を経て円弧補間で動作します。
END
```

ＭＯＶＥＩ

機能

ロボットの指定した関節軸を現在位置から指定量だけ動かします。

フォーマット

ＭＯＶＥＩ＜軸＞，＜相対位置＞［ＷＩＴＨ節］

文例

```
ＭＯＶＥＩ　１，６０
ＭＯＶＥＩ　３，１０　ＷＩＴＨ　ＳＰＥＥＤ＝５０
```

解説・注意

ロボットの指定した関節軸を指定量だけ動かします。このような動作を相対単軸動作と呼びます。

＜軸＞は１～５までの整数値で、ロボットの関節軸を指定します。指定軸以外の軸は動作しません。

＜相対位置＞には、ロボットの関節軸の現在位置からの移動量を、回転軸は度単位で、直動軸はミリメートル単位で指定します。移動が関節リミットで設定した範囲外になる場合には、関節リミットのすぐ内側まで動作します。

＜軸＞、＜相対位置＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型の変数は使用できません。

＜軸＞として１～５以外、及び実装していない軸を指定すると無視されロボットは動作しません。

ロボットが動作を行う時には、その時点でのシステム変数の設定値により、速度、加減速等の動作条件が決まります。１つの動作だけに関して動作条件を変更するためには、ＷＩＴＨ節を用いて動作条件を指定します。ＷＩＴＨ節の使い方は、ＷＩＴＨ命令を参照してください。

サンプルプログラム

```
PROGRAM  MOVEISAMPL
MOVEI 1,30          第１軸を現在位置から３０度動かします。
MOVEI 2,30          第２軸を現在位置から３０度動かします。
MOVEI 3,30          第３軸を現在位置から３０度動かします。
MOVEI 4,30          第４軸を現在位置から３０度動かします。
END
```

ＭＯＶＥＪ

機能

ロボットを指定した位置へアーチ動作で動かします。

フォーマット

ＭＯＶＥＪ　＜位置＞　＜アーチ定義＞

文例

ＭＯＶＥＪ　Ａ１　ＡＣ　ＷＩＴＨ　ＳＰＥＥＤ＝３０
ＭＯＶＥＪ　Ａ１　｛５０．０，　２０．０，　３０．０｝

解説・注意

ロボットを指定した位置へアーチ動作で動かします。

ロボットはＭＯＶＥ命令と同様にＰＴＰ動作で移動します。

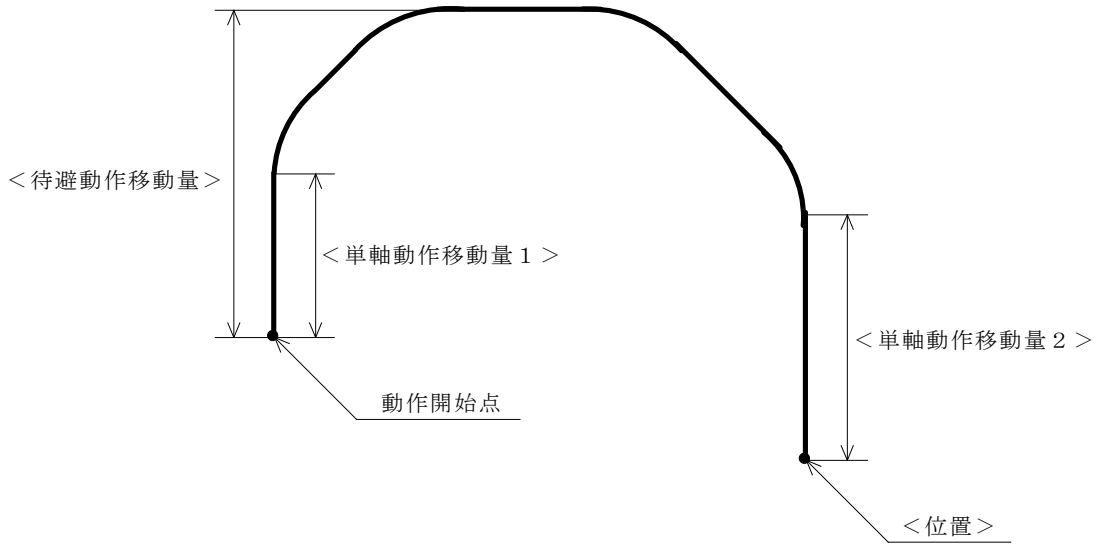
すなわち、アーチ動作における水平移動では直線軌道を通りません。

＜位置＞はＭＯＶＥＪ動作の最終的な目標位置を指定します。＜位置＞には位置型データを用いることができます。

＜アーチ定義＞はアーチ動作形状を数値で指定します。＜アーチ定義＞には位置型データを用います。＜アーチ定義＞では３つの要素のみが有効となります。（第４～６要素の数値は無視します）

＜アーチ定義＞＝｛＜待避動作移動量＞，＜単軸動作移動量１＞，＜単軸動作移動量２＞｝

なお、＜位置＞、＜アーチ定義＞には座標型データや負荷型データを用いることはできません。



＜待避動作移動量＞には動作開始位置からＺ軸方向最上位置までの距離(単位ｍｍ)を指定します。 ＜待避動作移動量＞は０．０以上の実数値で指定します。＜待避動作移動量＞に負の数値を指定した場合はエラーになります。

＜単軸動作移動量１＞には上方向移動動作における３軸のみの動作区間距離（単位ｍｍ）を指定します。 ＜単軸動作移動量１＞は０．０以上の実数値で指定します。 ＜単軸動作移動量１＞に負の数値を指定した場合はエラーになります。 ＜単軸動作移動量１＞が上方向移動量よりも大きい場合は、上方向移動量が指定されたものとして処理を行います。

＜単軸動作移動量２＞には下方向移動動作における３軸のみの動作区間距離（単位ｍｍ）を指定します。 ＜単軸動作移動量２＞は０．０以上の実数値で指定します。 ＜単軸動作移動量２＞に負の数値を指定した場合はエラーになります。 ＜単軸動作移動量２＞が下方向移動量よりも大きい場合は、下方向移動量が指定されたものとして処理を行います。

また次のようにして、＜位置＞、＜アーチ定義＞に数値を直接指定することもできます。

MOVEJ POINT(X,Y,Z,C,T,<姿勢>) POINT(Z1,Z2,Z3)
MOVEJ {X,Y,Z,C,T} {Z1,Z2,Z3} WITH CONFIG=<姿勢>
（データ型を明確にするために POINT 命令を使用するようにして下さい）

X,Y,Z,C,T ：Ｘ，Ｙ，Ｚ，Ｃ，Ｔの各座標値を実数で指定します。
（ｍｍ，ｄｅｇ単位）
<姿勢> ：０～２の整数値でロボットの姿勢を指定します。
（０：未定義，１：左肩系，２：右肩系）
Z1,Z2,Z3 ：アーチ形状を定義するための数値を実数で指定します。
（ｍｍ単位）

ＭＯＶＥＪ命令は上方向移動動作、水平方向移動動作、下方向移動動作を合成するもので、円弧軌道を描くものではありません。 また、Ｚ軸方向最上位置を優先するために上方向と下方向の移動動作は合成しません。 したがって、上下方向の移動量に対して水平方向の移動量が小さい場合は、単軸動作移動量が指定値よりも大きくなることがあります。

ＭＯＶＥＪ命令の動作開始点における姿勢と目標姿勢が異なる場合、水平方向移動動作において姿勢が変化します。

ＭＯＶＥＪ命令実行時にＭＯＴＩＯＮ命令が示す実行量は、ＭＯＶＥＪ命令の総移動時間に対する経過時間の百分率になります。 同様にＲＥＭＡＩＮ命令が示す残り実行量は、ＭＯＶＥＪ命令の総移動時間に対する残り時間の百分率になります。

ＭＯＶＥＪ命令をＢＲＥＡＫ命令で中断した場合、上方向への残り移動量、下方向への残り移動量、それぞれの残り単軸移動量、目標位置を保持し、継続起動を行う場合は、これらに基づきＭＯＶＥＪ命令を再生成します。　したがって、ＢＲＥＡＫによる中断中にロボットを手動誘導などで動かした場合､途中経路を保証することはできません。

ＭＯＶＥＪ命令はその他の動作命令（ＭＯＶＥＪ命令を含む）との間でショートカット動作する事はできません。

サンプル
プログラム

```
PROGRAM MOVEJSAMPL
P1   = POINT(300.0,  350.0,  0.0, 0.0, 0.0)
P2   = POINT(300.0, -350.0, 0.0, 0.0, 0.0)
ARCH = POINT(100.0,   40.0,   50.0)
MOVE  P1
MOVEJ P2 ARCH
END
```

ＭＯＶＥＰＬＴ

機能

ロボットをパレットの指定位置に動かします。

フォーマット

ＭＯＶＥＰＬＴ（＜パレット番号＞，＜要素番号＞，Ｘ，Ｙ，Ｚ，Ｃ）

文例

ＭＯＶＥＰＬＴ（１，１０，０，０，５０，０）

解説・注意

ロボットをパレット番号、要素番号で示される位置にＸ，Ｙ，Ｚ，Ｃのオフセットを加算した位置に動かします。オフセット値の０は省略できません。

ＭＯＶＥＰＬＴを実行する前には、その**パレットを初期化**（ＩＮＩＴＰＬＴ）**する必要**があります。

ＩＮＩＴＰＬＴ命令はダイナミックリンクライブラリで提供しています。

よって、命令を実行するにはＧＬＯＢＡＬエリアで**ライブラリの組み込み宣言およびグローバル変数宣言**をする必要があります。

要素番号が０以下、または最大要素数以上の値を指示した場合、ＴＰに下記メッセージを表示してプログラムは停止します。

要素番号＜１　の時　　“ERR !! ELEMENT NO. IS TOO SMALL”

要素番号＞INITPLTのｉ×ｊ×ｋの時　“ERR !! ELEMENT NO. IS TOO LARGE”

詳細は付録Ｇ－１パレタイズライブラリを参照してください

サンプルプログラム

```
GLOBAL
LOADLIB PALLET.LIB              ライブラリの組み込み宣言
DIM PLTP(1,7) AS POINT          グローバル変数宣言
END
PROGRAM SAMPLE
INITPLT(1,3,4,2)                PLTP(1,1)～PLTP(1,4)の教示点で３×４×２に初期化
MOVEPLT(1,1,0,0,50,0)           パレット１、要素番号１の５０㎜上に移動
OPEN1
MOVEPLT(1,1,0,0,0,0)            パレット１、要素番号１に移動
CLOSE1
MOVEPLT(1,1,0,0,50,0)           パレット１、要素番号１の５０㎜上に移動
END
```

ＭＯＶＥＳ

機能

ロボットを指定した位置へ直線補間で動かします。

フォーマット

ＭＯＶＥＳ＜位置＞［ＷＩＴＨ節］

文例

ＭＯＶＥＳ　Ａ１
ＭＯＶＥＳ　Ａ１　ＷＩＴＨ　ＧＡＩＮ＝｛，，ＯＮ｝

解説・注意

ロボットを指定した位置へ直線補間で動かします。
ロボットの手先は、現在位置から指定した位置までを結ぶ直線上を通るように動作します。このような動作を直線補間動作と呼びます。動作時には、ロボットの手先の線速は一定になります。（加減速時は除く）
＜位置＞には、位置型データを用いる事ができます。また、次のようにして、＜位置＞として座標値を直接指定する事もできます。
なお、＜位置＞には座標型データや負荷型データを用いることはできません。

ＭＯＶＥＳ　ＰＯＩＮＴ（Ｘ，Ｙ，Ｚ，Ｃ，Ｔ，＜姿勢＞）
ＭＯＶＥＳ｛Ｘ，Ｙ，Ｚ，Ｃ，Ｔ｝
　　ＷＩＴＨ　ＣＯＮＦＩＧ＝＜姿勢＞
（データ型を明確にするためにＰＯＩＮＴ命令を使用するようにしてください。）
Ｘ，Ｙ，Ｚ，Ｃ，Ｔ　：　Ｘ，Ｙ，Ｚ，Ｃ，Ｔの各座標値を実数で指定します。
　　（ｍｍ，ｄｅｇ単位）
＜姿勢＞　　：　０～２の整数値でロボットの姿勢を
　　指定します。
　　（０：未定義，１：左肩系，２：右肩系）
各要素には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型の変数は使用できません。＜姿勢＞として０以下，２以上を指定すると、それぞれ０，２が指定されたものとして処理されます。
各要素は、省略する事もできますが、省略した要素は０と見なされます。
例）　MOVES POINT(100,100,0,0,0,0)

ティーチペンダントから位置データを教示すると、教示した時に指定したワーク座標系のデータも同時に記憶します。動作命令の実行時には、ワーク座標系はその位置データを教示した時のものに変わります。ベース，ツール座標系については、命令実行時の設定をそのまま使用します。

＜位置＞として、座標値を直接指定した場合（ＤＥＳＴ，ＨＥＲＥ，ＰＯＩＮＴ命令等で位置データを合成した場合も含む）には、命令実行時のワーク座標系にて移動を行います。

ロボットが動作を行う時には、その時点でのシステム変数の設定値により、速度、加減速等の動作条件が決まります。１つの動作だけに関して動作条件を変更するためには、ＷＩＴＨ節を用いて動作条件を指定します。ＷＩＴＨ節の使い方は、ＷＩＴＨ命令を参照してください。

［５軸目追加（オプション）について］

５軸目追加（オプション）を行った場合、５軸目に関しては直線補間動作を行えませんのでご注意ください。他の軸間では直線補間動作を行います。

サンプル
プログラム

```
PROGRAM  MOVESSAMPL
MOVE A1
MOVES A2                              Ａ２へ直線補間で移動します。
END
```

ＭＯＶＥＳＹＮＣ

機能

動作命令同期モードか、動作命令非同期モードかを指定します。

フォーマット

ＭＯＶＥＳＹＮＣ

文例

ＤＩＳＡＢＬＥ ＭＯＶＥＳＹＮＣ

解説・注意

動作命令と信号入出力命令が複数個交互に並んでいるＳＣＯＬプログラムのＥＮＡＢＬＥ ＮＯＷＡＩＴ時の動作を考えます。ＥＮＡＢＬＥ ＭＯＶＥＳＹＮＣ状態(動作命令同期モード)では次の動作命令直前まで先実行してから位置決め完了を待ちます。このため2回目の信号入出力は2回目の動作命令開始直後に、３回目の信号入出力命令は３回目の動作命令開始直後に実行されます。但しこのモードではシステム変数ＰＡＳＳの状態によらず、ショートカット(パス)動作ができません。これに対してＤＩＳＡＢＬＥ ＭＯＶＥＳＹＮＣ状態(動作命令非同期モード)では最大４つ先の動作命令直前まで先実行してから位置決め完了を待ちます。このため２回目以降の信号入出力命令が１回目の動作中に実行されてしまうことがありますが、システム変数ＰＡＳＳをＥＮＡＢＬＥと設定することでパス動作が可能となります。ＳＣＯＬプログラム起動時の本システム変数の値は、ユーザパラメータ[U03]で設定します。

サンプルプログラム

```
PROGRAM  MAIN
ENABLE NOWAIT
ENABLE MOVESYNC
MOVEA 1,90
DOUT(1)                 Ｄ１は１軸＋９０度への移動中にＯＮする。
MOVEA 1,-90
DOUT(2)                 Ｄ２は１軸－９０度への移動中にＯＮする。
DISABLE MOVESYNC
MOVEA 2,90
DOUT(3)                 Ｄ３は２軸＋９０度への移動中にＯＮする。
MOVEA 2,-90
DOUT(4)                 Ｄ４は２軸（－９０度ではなく）＋９０度への移動中にＯＮする。
END
```

ＮＥＸＴ

機能

ＦＯＲ命令と組合わせてプログラムの繰返しを指定します。

フォーマット

ＮＥＸＴ［＜変数＞］

文例

```
ＮＥＸＴ　Ｋ
```

解説・注意

ＦＯＲ命令と組合わせて、プログラムの繰返しを指定します。
ＦＯＲ命令とＮＥＸＴ命令にはさまれたプログラムを、繰返し実行します。プログラムの実行は、ＦＯＲ文で指定した条件が満たされるまで繰返します。
＜変数＞には、対応するＦＯＲ文で指定した変数を指定します。
ＮＥＸＴ文で＜変数＞を指定しない場合には、最も近いＦＯＲ文（最後に実行したＦＯＲ文）との間でループを作ります。
プログラムの繰返し条件については、ＦＯＲ命令を参照してください。

サンプルプログラム

```
PROGRAM  NEXTSAMPLE
  FOR K=1 TO 100          Ａ１，Ａ２間の往復動作を、１００回繰返します。
    MOVE A1
    MOVE A2
  NEXT K
END
```

ＮＯＴ

機能	論理式の真偽の判定を逆にします。
フォーマット	ＮＯＴ＜論理式＞
文例	ＩＦ　ＮＯＴ　ＤＩＮ（１）　ＴＨＥＮ　ＳＴＯＰ
解説・注意	論理式の真偽の判定を逆にします。 本命令は、論理式の中で使用します。
サンプル プログラム	PROGRAM NOTSAMPLE IF NOT DIN(1) THEN DOUT(1)　　入力信号１がオフの場合、出力信号１をＯＮします。 END

ＮＯＷＡＩＴ

機能

動作の位置決め完了を待たないで、信号の入出力処理を行う指定をします。

フォーマット

ＮＯＷＡＩＴ

文例

ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴ
ＥＮＡＢＬＥ　ＮＯＷＡＩＴ

解説・注意

外部信号の入出力を、ロボットの位置決め完了を待って行うか、待たずに行うかを指定します。
信号出力のタイミングについては、５章にて説明していますので、そちらを参照してください。
システムスイッチのオン，オフには、ＥＮＡＢＬＥ，ＤＩＳＡＢＬＥ命令を使用します。ＥＮＡＢＬＥ　ＮＯＷＡＩＴで外部信号の入出力は、ロボットの位置決めを待たずに行い、ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴで位置決め完了を待って行うようになります。
初期状態では、ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴになっています。

サンプルプログラム

```
PROGRAM  NOWAITSMPL
ENABLE NOWAIT        外部信号の入出力を、ロボットの位置決め完了を待たず
MOVE A1              に行います。
DOUT(1)
MOVE A2
DOUT(2)
MOVE A3
END
```

ＯＦＦ

機能

各軸のゲイン（サーボ制御）のオフを指定するシステム定数です。

フォーマット

ＯＦＦ

文例

ＧＡＩＮ＝｛ＯＦＦ，ＯＦＦ，ＯＮ，ＯＦＦ，ＯＦＦ｝
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＧＡＩＮ＝｛，，ＯＦＦ｝

解説・注意

ＧＡＩＮ命令と共に使用して、各軸のゲイン（サーボ制御）のオフを指定するためのシステム定数です。
ゲインをオフに指定すると、次に動作命令を実行した時にサーボ制御を切ります。
サーボ制御を切った軸は、サーボフリー状態（位置決め制御を行わない状態）になります。
本システム定数は、０という値を持っています。プログラムの式の中で定数０として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。
ゲインについては、ＧＡＩＮ命令を参照してください。

サンプルプログラム

```
PROGRAM  OFFSAMPLE
MOVE A1                    Ａ２への動作時に、Ｚ軸（この場合第３軸）を
WAIT MOTION >=100          除いた軸のゲインをオフします。
GAIN={OFF,OFF,ON,OFF,OFF}
MOVE A2
OPEN1
DELAY 0.5
MOVE A1
WAIT MOTION >=100
GAIN={ON,ON,ON,ON,ON}
READY
END
```

ＯＮ

機能

各軸のゲイン（サーボ制御）のオンを指定するシステム定数です。

フォーマット

ＯＮ

文例

ＧＡＩＮ＝｛ＯＦＦ，ＯＦＦ，ＯＮ，ＯＦＦ，ＯＦＦ｝
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＧＡＩＮ＝｛，，ＯＮ｝

解説・注意

ＧＡＩＮ命令と共に使用して、各軸のゲイン（サーボ制御）のオンを指定するためのシステム定数です。
ゲインをオンに指定すると、次に動作命令を実行した時にサーボ制御をオンします。
サーボ制御を切った軸は、サーボフリー状態（位置決め制御を行わない状態）になります。
本システム定数は、１という値を持っています。プログラムの式の中で定数１として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。
ゲインについては、ＧＡＩＮ命令を参照してください。

サンプルプログラム

```
PROGRAM  ONSAMPLE
MOVE A1                          Ａ２への動作時に、Ｚ軸（この場合第３軸）を
WAIT MOTION >=100                除いた軸のゲインをオフし、プログラムの終わ
GAIN={OFF,OFF,ON,OFF,OFF}        りで、オフしたゲインを全軸オンします。
MOVE A2
OPEN1
DELAY 0.5
MOVE A1
WAIT MOTION >=100
GAIN={ON,ON,ON,ON,ON}
READY
END
```

ＯＮ

機能	条件の監視を行います。 （ゲインオン／オフの指定については、前ページを参照してください。）
フォーマット	ＯＮ＜監視条件＞［｛ＢＲＥＡＫ｜ＰＡＵＳＥ｝］ＤＯ＜文＞
文例	ＯＮ　ＤＩＮ（１）　ＤＯ　ＲＥＴＵＲＮ ＯＮ　ＴＩＭＥＲ　ＤＯ　ＭＯＶＥ　Ａ１
解説・注意	＜監視条件＞の条件が成立したら、ＤＯに続く文を実行します。 条件の監視は、ロボットの動作中いかんにかかわらず行います。 ロボットが動作中でも、ＯＮ命令の処理はロボットの動作と並行して行います。 ＭＯＴＩＯＮ，ＭＯＴＩＯＮＴ，ＲＥＭＡＩＮ，ＲＥＭＡＩＮＴ命令を監視条件として使用した場合には後続の動作命令に対して条件の監視を行います。ＴＩＭＥＲを使用した場合にはロボットの動作に関係なく条件の監視を行います。ＤＩＮ命令などで、入力信号の監視を指定した場合には、システムスイッチＮＯＷＡＩＴの指定により、条件の監視を開始するタイミングが異なります。 ＥＮＡＢＬＥ　ＮＯＷＡＩＴの場合には、ロボットの動作に関係なく信号の監視を開始します。ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴの場合には実行中のロボットの動作が終了してから信号の監視を開始します。 ＤＯに続く文を実行するタイミングは、監視条件が成立した時に実行中の命令の処理が終了してからになります。ただし、ＷＡＩＴ命令を実行中には、ＷＡＩＴの処理を中断してＤＯに続く文の処理を行います。 ロボットが動作中の場合には次の３種の実行タイミングを指定できます。 ＢＲＥＡＫ：　動作を即時中止してＤＯに続く文を実行します。 ＰＡＵＳＥ：　実行中の動作の終了を待ってＤＯに続く文を実行します。但しアーム動作中は、サブプログラム呼出し命令、メインプログラムへの復帰命令、動作命令を除いて通常プログラム実行を続けます。これらの命令実行時には、アーム停止までプログラム実行が停止します。 指定なし：　動作と並行してＤＯに続く文を実行します。ＤＯに続く文が動作命令の場合には必ずＢＲＥＡＫかＰＡＵＳＥを指定してください。

ＤＯに続く文の実行(これをＤＯ節と言います)が終了し、かつＤＯに続く文中で動作命令を実行した場合はそのアーム動作が完了すると、プログラムの実行はＯＮ命令の条件成立前の流れに従って再開します。（ＷＡＩＴ命令を中断した場合には中断したＷＡＩＴ命令から実行を再開します。）ただし、ＤＯに続く文にてラベルへのプログラム分岐を行った場合には、そのラベルを持つ文から実行します。
同時に監視できる条件は最大１０組までです。また、同一のＯＮ命令中で指定できる入力信号は４点までです。
同時に複数の監視条件が成立した場合には、ＯＮ命令の実行された順番を優先順位にして、最も優先順位の高いＯＮ命令に対応するＤＯに続く文のみを実行して、その他のものは無視します。
ＯＮ命令による条件の監視は、他のＯＮ命令のＤＯに続く文を実行中は行いません。また、ＳＴＯＰ命令やエラーの発生により、プログラムの実行が停止している時にも条件監視は行いません。
ＤＯに続く文にてサブプログラムを指定すれば，条件成立時にサブプログラムにて記述した複数の処理を行う事ができます。なお、ＤＯに続く文として実行中のサブプログラム内でＯＮ文を使用した場合、条件の監視はそのサブプログラムがＲＥＴＵＲＮした直後より有効となります。
システムのタイマを監視条件として指定した場合には、条件のチェックは監視対象の状態が変わった時にのみ行います。外部信号、エラー条件、及び動作状態（動作の残り移動量等）の監視は、状態そのもので監視します。
ＯＮ文で指定した条件監視の解除は、ＩＧＮＯＲＥ命令で行います。条件監視は、条件が成立してＤＯに続く文を実行した時点にも解除します。

注１）　ＯＮ～ＤＯ命令は現状以下の使い方に限定されています。

・ＯＮ　ＴＩＭＥＲ　ＤＯ　＜文＞
　タイマーが０になったら＜文＞を実行します。

・ＯＮ　ＤＩＮ（　）　ＤＯ　＜文＞
　入力信号が指定状態になったら＜文＞を実行します。同時に監視できるのは４点までです。４点以上を指定すると４点を超えた分は無視します。

・ＯＮ　ＭＯＴＩＯＮ＞＝＜式＞　ＤＯ　＜文＞
　本命令の次に実行する動作命令の動作量が指定以上になったら＜文＞を実行します。ＭＯＴＩＯＮに関しては＞＝以外の比較演算子は使用できません。

・ＯＮ　ＭＯＴＩＯＮＴ＞＝＜式＞　ＤＯ　＜文＞
　本命令の次に実行する動作命令の動作時間が指定以上になったら＜文＞を実行します。ＭＯＴＩＯＮＴに関しては＞＝以外の比較演算子は使用できません。

・ＯＮ　ＲＥＭＡＩＮ＜＝＜式＞　ＤＯ　＜文＞

本命令の次に実行する動作命令の残り動作量が指定以下になったら＜文＞を実行します。ＲＥＭＡＩＮに関しては＜＝以外の比較演算子は使用できません。

・ＯＮ　ＲＥＭＡＩＮＴ＜＝＜式＞　ＤＯ　＜文＞

本命令の次に実行する動作命令の残り動作時間が指定以下になったら＜文＞を実行します。ＲＥＭＡＩＮＴに関しては＜＝以外の比較演算子は使用できません

注２）　ＤＯ節に続く文の中ではタスク制御に関する以下の命令は使用できません。

ＴＡＳＫ，ＫＩＬＬ，ＳＷＩＴＣＨ

ＤＯ節以下でこれらの命令を使用した場合、その命令語は動作しません。また、サブタスクＯＮ命令による条件の監視は行えませんので注意してください。

注３）　ＯＮ ＭＯＴＩＯＮ条件、ＯＮ ＭＯＴＩＯＮＴ条件、ＯＮ ＲＥＭＡＩＮ条件、ＯＮ ＲＥＭＡＩＮＴ条件の監視対象動作を停止したり、対象動作中に低速指令を発行するとＯＮ条件は解除されます。

サンプル
プログラム

```
PROGRAM MAIN
ONSAMPLE
MOVE P
END
PROGRAM  ONSAMPLE
ON DIN(1) PAUSE DO RETURN      入力信号の１がオンになったら、実行中の動作
MOVE A1                        の完了にてメインプログラムに戻ります。
MOVE A2
MOVE A3
WAIT MOTION >=100
IGNORE DIN(1)
RETURN
END
```

ＤＯ節内の注意

ＯＮ～ＤＯ命令は、監視するＯＮ条件と条件が成立した場合に起動されるＤＯ節の組み合わせを登録します。このため次のＳＣＯＬプログラムで

```
 PROGRAM MAIN
 SIG=1
 ON DIN(1) DO INPUT SIG
 SUB
 IGNORE DIN(1)
 PRINT SIG
END
PROGRAM SUB
 MOVE P
 WAIT MOTION>=100
END
```

Ｐへの移動中にDIN(1)をONすると、プログラムＳＵＢ内で変数ＳＩＧが定義されておらずＩＮＰＵＴ命令で取り込まれた変数を格納する場所がありませんので、ＤＯ節を実行することが出来ません。このような場合は変数ＳＩＧをグローバル変数として定義することで、該当ＤＯ節を正常に動作させることが出来ます。

```
GLOBAL
 SIG=0
END
PROGRAM MAIN
 SIG=1
 ON DIN(1) DO INPUT SIG
 SUB
 IGNORE DIN(1)
 PRINT SIG
END
PROGRAM SUB
 MOVE P
 WAIT MOTION>=100
END
```

なおＤＯ節内では、タスク切替条件が成立しても、あるいは“ＳＷＩＴＣＨ”命令を実行してもタスクは切替わりません。また“ＴＡＳＫ”命令や“ＫＩＬＬ”命令を実行しようとするとエラーとなります。

ＯＮＧＡＩＮ

機能

各軸のゲイン（サーボ制御）のオンを指定します。

フォーマット

ＯＮＧＡＩＮ（＜整数＞，＜整数＞，＜整数＞，＜整数＞，＜整数＞）

文例

ＯＮＧＡＩＮ（０，　０，　１，　０，　０）

解説・注意

ロボットの各軸のゲイン（サーボ制御）をオンします。
ゲインについては、ＧＡＩＮ命令を参照してください。
ゲインをオフするためには、ＯＦＦＧＡＩＮ命令を使用します。
１～５軸の各関節軸に対応して５軸分の値を「，」で区切って指定します。値は１か０で指定し、１を指定した軸についてゲインがオンになります。０を指定した軸についてはそのままの状態になります。
ゲインのオン，オフ状態は、現在の動作が終了してから切替わります。
本命令は、コントローラのＲＡＭドライブにＳＣＯＬ．ＬＩＢという名称のファイルがないと実行できません。
本命令ではシステム定数のＯＮ，ＯＦＦ は使用できません。

サンプルプログラム

```
PROGRAM ONGAINSMPL
MOVE A1                          Ａ１への到着を待って、３軸以外の
OFFGAIN(1,1,0,1,1)               ゲインをオフします。
MOVE A2                          Ａ２への到着を待って、全ての軸の
ONGAIN(1,1,1,1,1)                ゲインをオンします。
MOVE A3
END
```

ＯＦＦＧＡＩＮ

機能

各軸のゲイン（サーボ制御）のオフを指定します。

フォーマット

ＯＦＦＧＡＩＮ（＜整数＞，＜整数＞，＜整数＞，＜整数＞，＜整数＞）

文例

ＯＦＦＧＡＩＮ（１，　１，　０，　１，　１）

解説・注意

ロボットの各軸のゲイン（サーボ制御）をオフします。
ゲインについては、ＧＡＩＮ命令を参照してください。
ゲインをオンするためには、ＯＮＧＡＩＮ命令を使用します。
１～５軸の各関節軸に対応して５軸分の値を「，」で区切って指定します。値は１か０で指定し、１を指定した軸についてゲインがオフになります。０を指定した軸についてはそのままの状態になります。
ゲインのオン，オフ状態は、現在の動作が終了してから切替わります。
本命令は、コントローラのＲＡＭドライブにＳＣＯＬ．ＬＩＢという名称のファイルがないと実行できません。
本命令ではシステム定数のＯＮ，ＯＦＦは使用できません。

サンプルプログラム

```
PROGRAM OFGAINSMPL
MOVE A1                          Ａ１への到着を待って、３軸以外の
OFFGAIN(1,1,0,1,1)               ゲインをオフします。
MOVE A2                          Ａ２への到着を待って、全ての軸の
ONGAIN(1,1,1,1,1)                ゲインをオンします。
MOVE A3
END
```

ＯＰＥＮ１，ＯＰＥＮ２，ＯＰＥＮＩ１，ＯＰＥＮＩ２

機能

ロボットのハンドを開きます。

フォーマット

ＯＰＥＮ１
ＯＰＥＮ２
ＯＰＥＮＩ１
ＯＰＥＮＩ２

文例

ＯＰＥＮ１
ＯＰＥＮＩ２

解説・注意

ロボットのハンドを開きます。添字の１，２はそれぞれハンド１，ハンド２を表わします。
本命令を実行すると、コントローラはロボットのハンド制御用の出力信号の状態を変更してハンドを開きます。
ＯＰＥＮ命令では、ロボットが現在実行中の動作を終了するのを待ってロボットのハンドを開きます。
ＯＰＥＮＩ命令では、命令実行にて即時にロボットのハンドを開きます。
本命令は、コントローラのＲＡＭドライブにＳＣＯＬ．ＬＩＢという名称のファイルがないと実行できません。
ロボットのハンドに信号を出力してから実際にハンドが開くまでには多少の時間がかかりますので注意してください。
ロボットのハンドを閉じるには、ＣＬＯＳＥ１，ＣＬＯＳＥ２，
ＣＬＯＳＥＩ１，ＣＬＯＳＥＩ２命令が用意されています。

本命令は、システムライブラリ（ＳＣＯＬ．ＬＩＢ）に書かれているプログラムを実行します。ロボットのハンドの仕様に合わせて、ＳＣＯＬ．ＬＩＢの内容を変更する必要があります。

サンプル
プログラム

```
PROGRAM  OPENSAMPLE
CLOSE1                  Ａ１の移動が終了してからハンドを開きます。
MOVE A1                 ＯＰＥＮ１命令を実行してからロボットのハンドが
OPEN1                   完全に開くまで０．５秒待ちます。
DELAY 0.5
MOVE A2
END
```

```
PROGRAM  OPENISMPLE
ENABLE NOWAIT
OPENI1                  Ａ１に動きながらハンド１を開きます。
DELAY 2
MOVE A1
CLOSEI1
DELAY 2
MOVE A2
END
```

ＯＲ

機能	論理和を計算します。
フォーマット	＜論理式＞ＯＲ＜論理式＞
文例	ＩＦ　ＤＩＮ（１）　ＯＲ　Ｋ＜＝３　ＴＨＥＮ　Ｊ＝０ ＷＡＩＴ　ＤＩＮ（５）　ＯＲ　ＴＩＭＥＲ＝＝０
解説・注意	ＯＲの右辺と左辺の論理式の論理和を計算します。 ＯＲの両辺の論理式の一方でも真ならば結果が真になります。 本命令は、論理式の中で使用します。
サンプル プログラム	PROGRAM　ORSAMPLE FOR K=1 TO 50 IF DIN(1) OR K==50 THEN J=1 ELSE J=0 PRINT TP,J,CR NEXT K END

ＰＡＩ

機能

円周率を与えるシステム定数です。

フォーマット

ＰＡＩ

文例

Ｒ＝Ｄ＊ＰＡＩ／１８０
Ｄ＝Ｎ＊ＰＡＩ＊２

解説・注意

円周率（＝３．１４１５９２６・・・）を与えるシステム定数です。
システム定数ＰＡＩは、式の中で円周率として自由に使う事ができます。
システム定数への代入はできません。

サンプルプログラム

```
PROGRAM  MAIN
PAISAMPLE(5,D)
PRINT TP,D,CR
END
PROGRAM  PAISAMPLE(R,D)
D=R/PAI*180                 引数の値をラジアンから度単位に変換して、その
RETURN                      結果を引数Ｄとしてメインプログラムに返します。
END
```

ＰＡＳＳ

機能

ショートカット動作を指定します。
（ショートカット動作のパラメータの設定については、次ページを参照してください。）

フォーマット

ＰＡＳＳ

文例

ＤＩＳＡＢＬＥ　ＰＡＳＳ
ＥＮＡＢＬＥ　ＰＡＳＳ

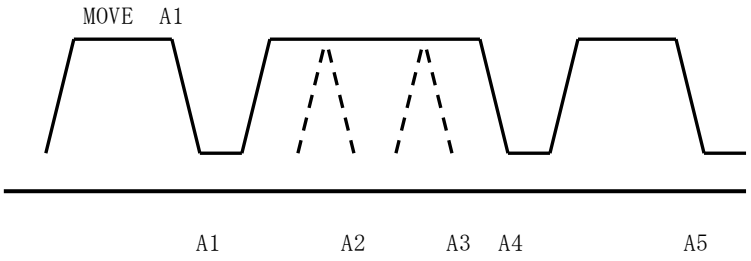
解説・注意

ショートカット動作を指定します。
ショートカット動作では、現在の動作が位置決め完了する前に次の移動を開始します。現在の動作から次の動作へ移るタイミングは、システム変数のＰＡＳＳにて指定します。ショートカット動作を指定する事により、ロボットの動作時間を短縮する事ができます。ショートカット動作については、５章にても説明していますので、そちらを参照してください。
システムスイッチのオン、オフには、ＥＮＡＢＬＥ，ＤＩＳＡＢＬＥ命令を使用します。ＥＮＡＢＬＥ　ＰＡＳＳでショートカット動作を開始し、
ＤＩＳＡＢＬＥ　ＰＡＳＳで、ショートカット動作は解除します。
初期状態では、ＤＩＳＡＢＬＥ　ＰＡＳＳになっています

サンプルプログラム

```
PROGRAM  PASSSAMPLE
  MOVE A1
  PASS＝80
  ENABLE PASS
  MOVE A2
  MOVE A3
  DISABLE PASS
  MOVE A4
  MOVE A5
END
```

Ａ１からＡ４までをショートカット動作で移動します。
Ａ４以降の動作ではショートカット動作を解除します。



ＰＡＳＳ

機能

ショートカット動作のパラメータを設定します。
（ショートカット動作の指定については、前ページを参照してください。）

フォーマット

ＰＡＳＳ＝＜式＞

文例

ＰＡＳＳ＝８０
ＰＡＳＳ＝ＰＡＳＳ＊０．８

解説・注意

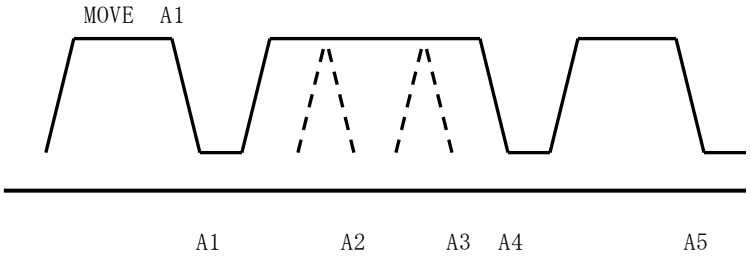
ショートカット動作のパラメータを指定するシステム定数です。
ショートカット動作では、現在の動作が位置決め完了する前に次の移動を開始します。現在の動作から次の動作へ移るタイミングを、システム変数のＰＡＳＳにて指定します。ショートカット動作のパラメータは、動作に対するロボットの移動量のパーセンテージで指定します。ロボットは、移動量がシステム変数ＰＡＳＳにて指定されたパーセンテージを超えたら、次の動作を現在の動作に重ねて実行します。
なお移動量とはロボットへの指令位置であり、ロボットと装置の干渉(衝突)により実際のロボットが動作できない場合でも指令位置が指定の移動量になると次の動作が開始されます。
ＰＡＳＳは、０～１００までの整数値で指定しますが、５０以下を指定した場合には５０％として処理されます。１００以上の数値を指定すると、１００が指定されたものとします。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本システム変数を参照すると、現在のショートカット動作のパラメータが参照できます。
ショートカット動作を指定する事により、ロボットの動作時間を短縮する事ができます。ショートカット動作については、５章においても説明していますので、そちらを参照してください。
システムスイッチのオン，オフには、ＥＮＡＢＬＥ，ＤＩＳＡＢＬＥ命令を使用します。ＥＮＡＢＬＥ　ＰＡＳＳでパス動作を開始し、ＤＩＳＡＢＬＥ
ＰＡＳＳでパス動作は解除します。
初期状態では、ＤＩＳＡＢＬＥ　ＰＡＳＳになっています。
ＰＡＳＳ動作を行う予定の動作命令間にＷＡＩＴ命令または入出力命令が指令された場合はＰＡＳＳ動作を行うことが出来ない場合がありますので注意してください。

サンプル
プログラム

```
PROGRAM  PASSSAMPLE
  MOVE A1
  PASS＝80
  ENABLE PASS
  MOVE A2
  MOVE A3
  DISABLE PASS
  MOVE A4
  MOVE A5
END
```

Ａ１からＡ４までをショートカット動作で移動します。

Ａ４以降の動作ではショートカット動作を解除します。



ＰＡＵＳＥ

機能

ロボットの動作終了を待ちます。

フォーマット

ＯＮ＜監視条件＞［｛ＢＲＥＡＫ｜ＰＡＵＳＥ｝］ＤＯ＜文＞

文例

ＯＮ　ＤＩＮ（１）　ＰＡＵＳＥ　ＤＯ　ＳＵＢ

解説・注意

ＯＮ文で指定した監視条件の成立時に、ロボットの現動作の終了を待ってＤＯに続く文を実行します。詳細はＯＮ命令を参照してください。

サンプルプログラム

```
PROGRAM  PAUSESMPL
ENABLE NOWAIT
REMARK *** MAIN PROGRAM ***
ON DIN(24) PAUSE DO STOP
MOVE A1
MOVE A2
MOVE A3
WAIT MOTION>=100
IGNORE DIN(24)
END
```

システムに異常が発生して入力信号の２４がオン状態になると、現動作の終了後に実行を停止します。

ＰＡＹＬＯＡＤ

機能

ロボットの手先の負荷データを指定します。

フォーマット

ＰＡＹＬＯＡＤ＝｛＜質量＞，＜重心オフセット＞｝

文例

```
ＰＡＹＬＯＡＤ＝｛１０，１０｝
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＰＡＹＬＯＡＤ＝ＭＯＴＯＲ
```

解説・注意

ロボットの手先の負荷データを指定するシステム変数です。
ＳＣＯＬ言語では、どんな負荷に対してもロボットが最適な状態で動作できるように、ロボットの手先にかかる負荷を負荷データとして設定できるようになっています。
ロボットの手先にかかる負荷は、システム変数のＰＡＹＬＯＡＤに設定します。この設定値により、コントローラは加減速等の制御定数を負荷に最適となるように計算してロボットを動作します。負荷データとしては、手先にかかる負荷の重量と慣性モーメントの値を指定します。
＜質量＞には、ロボットの手先にかかる負荷の重量を、キログラム単位で指定します。＜重心オフセット＞には、ロボットの手先にかかる負荷の重心が手先のツ―ル中心からどれだけ離れているかをミリメ―トル単位で指定します。
＜質量＞，＜重心オフセット＞には、定数、変数、及び演算式を使用する事ができます。また、｛＜質量＞，＜重心オフセット＞｝の代わりに負荷型データを指定する事もできます。負荷型データの設定は、次のようにして行います。負荷型データ同士の加減算も可能です。

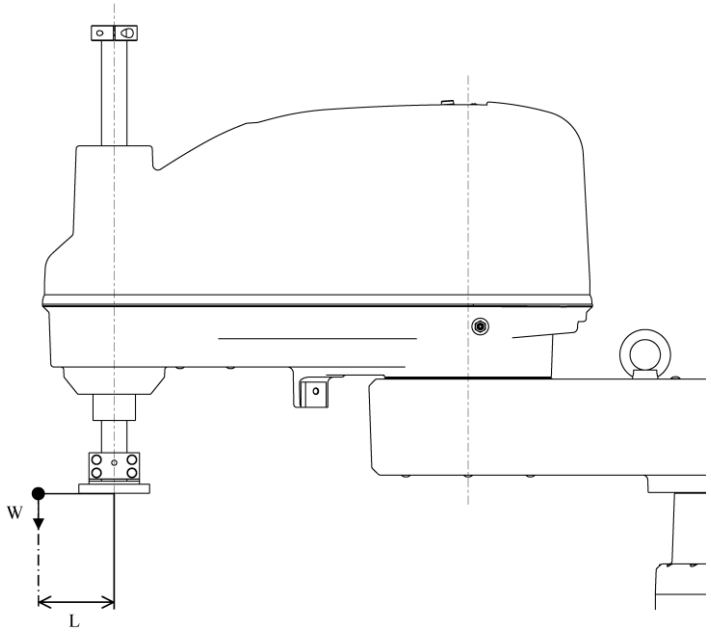
＜変数＞＝｛＜質量＞，＜重心オフセット＞｝
例）　ＭＯＴＯＲ＝｛５，１０｝
　　ＷＯＲＫＡ＝ＨＡＮＤ＋ＭＯＴＯＲ

負荷データを０にする命令として、ＦＲＥＥＬＯＡＤ命令があります。

サンプル
プログラム

```
PROGRAM PAYLOADSMP
PAYLOAD=HAND
MOVE A1
CLOSE1
DELAY 0.5
MOVE A2 WITH PAYLOAD=HAND+MOTOR
OPEN1
DELAY 0.5
MOVE A3
FREELOAD
MOVE A2
MOVE A3
END
```

負荷データをＨＡＮＤとしてＡ１まで動作し、物をつかんだ後は負荷データをＨＡＮＤ＋ＭＯＴＯＲとして動作します。



負荷データとして　　質　　量　W　　［単位　kg］
　　重心オフセット　L　　［単位　mm］
を入力します。

ＰＯＩＮＴ

機能

位置型データを生成します。

フォーマット

ＰＯＩＮＴ（＜式＞，＜式＞，＜式＞，＜式＞，＜式＞，＜姿勢＞）

文例

Ａ＝ＰＯＩＮＴ（１００，１００，０，０，０，０）
ＭＯＶＥ　ＰＯＩＮＴ（１００，１００）

解説・注意

位置型データを生成します。
＜式＞は、左から順番に、位置型データのＸ，Ｙ，Ｚ，Ｃ，Ｔ要素に対応します。
Ｘ，Ｙ，Ｚ，Ｃ，Ｔはミリメートル、もしくは度単位で指定します。
＜姿勢＞は、０～２の整数値でロボットの姿勢を指定します。ロボットの姿勢は０で未定義、１で左肩系、２で右肩系になります。ロボットの姿勢の指定には、システム定数のＦＲＥＥ，ＬＥＦＴＹ，ＲＩＧＨＴＹを使用する事ができます。ロボットの姿勢はＦＲＥＥで未定義、ＬＥＦＴＹで左肩系、ＲＩＧＨＴＹで右肩系に設定できます。ロボットの姿勢として、０以下，２以上を指定すると、それぞれ０，２が指定されたものとして処理されます。
＜式＞，＜姿勢＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。また、＜式＞，＜姿勢＞の記述は省略する事もできますが、省略した要素は０と見なされます。
本命令は、式の中で使用する事ができます。

サンプルプログラム

```
PROGRAM  POINTSMPL
MOVE POINT(500,500)
END
```

Ｘ＝５００，Ｙ＝５００，Ｚ，Ａ，
Ｂ，Ｃ，Ｔ＝０の位置へ動作します。

ＰＬＣＤＡＴＡＲ１～８

機能	ロボット内蔵の簡易ＰＬＣからデータを受取るシステム変数です。
フォーマット	ＰＬＣＤＡＴＡＲ１
文例	Ａ＝ＰＬＣＤＡＴＡＲ１ Ｂ＝ＰＬＣＤＡＴＡＲ５
解説・注意	ＰＬＣＤＡＴＡＲ１～８は読み込み専用のシステム変数です。 簡易ＰＬＣでセットした値を読み込むことができます。 （簡易ＰＬＣはオプション機能です。） 受取る値は０～６５５３５となります。（負の値および少数は扱えません）

SCOL言語名	データレジスタ
PLCDATAR1	D400
PLCDATAR2	D401
PLCDATAR3	D402
PLCDATAR4	D403
PLCDATAR5	D404
PLCDATAR6	D405
PLCDATAR7	D406
PLCDATAR8	D407

例）

100
（10 進数）

TCmini レジスタ

D400

PLCDATAR1
0x0064 10 進数（100）

ロボットプログラム

A=PLCDATAR1
結果 A=100

サンプル プログラム	PROGRAM PLCDIN A=PLCDATAR1 B=PLCDATAR2 END

ＰＬＣＤＡＴＡＷ１～８

機能

ロボット内蔵の簡易ＰＬＣに値を書き込むシステム変数です。

フォーマット

ＰＬＣＤＡＴＡＷ１＝＜式＞

文例

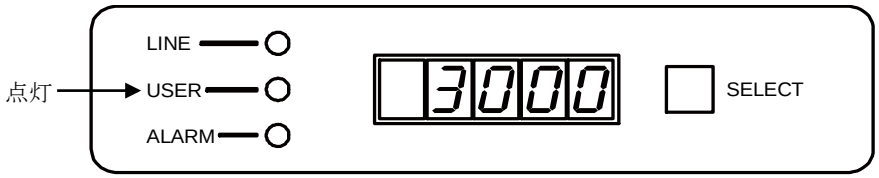
ＰＬＣＤＡＴＡＷ１＝１

ＰＬＣＤＡＴＡＷ５＝Ａ＋Ｂ

解説・注意

ＰＬＣＤＡＴＡＷ１～８は書込み専用のシステム変数です。
簡易ＰＬＣに値を受け渡すことができます。（簡易ＰＬＣはオプション機能です。）
受け渡す値は０～６５５３５となります。（負の値および少数は扱えません）

TS3000コントローラではＰＬＣＤＡＴＡＷ１に書き込まれた値が、コントローラ正面の７セグメント（ＵＳＥＲ）に表示されます。



簡易ＰＬＣ機能オプションが選択さている場合には、ＰＬＣＤＡＴＡＲ１～８の値をシーケンスプログラムで自由に扱うことができます。

SCOL言語名	データレジスタ
PLCDATAW1	D600
PLCDATAW2	D601
PLCDATAW3	D602
PLCDATAW4	D603
PLCDATAW5	D604
PLCDATAW6	D605
PLCDATAW7	D606
PLCDATAW8	D607

例）

ロボットプログラム

PLCDATAW1=20

⇩

TCmini レジスタ

D600
PLCDATAW1
0x0014 10 進数（20）

サンプルプログラム

```
PROGRAM  PLCDIN
A=10
PLCDATAW1=1
PLCDATAW2=A
END
```

ＰＲＩＮＴ

機能

通信チャネルにデータを出力します。

フォーマット

ＰＲＩＮＴ［｛ＣＯＭ０｜ＣＯＭ１｜ＴＰ｝，］｛＜文字列＞｜＜式＞｝
［，｛＜文字列＞｜＜式＞｝］　・・・　［，ＣＲ］

文例

ＰＲＩＮＴ　”Ｘ＝”，Ａ１．Ｘ
ＰＲＩＮＴ　ＣＯＭ１，Ｋ，ＣＲ

解説・注意

通信チャネルへデータを出力します。
通信チャネルとしては、ＣＯＭ０，ＣＯＭ１，ＴＰの中から１つを指定します。
ＣＯＭ０及びＴＰはティーチペンダント専用の通信チャネルです。ＣＯＭ１コントローラのＣＯＭ１通信チャネルに対応します。
ＰＲＩＮＴ命令にて通信チャネルを指定しないと、ティーチペンダント専用の通信チャネルに対して、データの出力を行います。
ＰＲＩＮＴ命令を実行すると、通信チャネルへデータを出力します。＜文字列＞データは、指定されたままの形で出力します。
＜式＞は、１２文字の固定長で右づめに出力します。式の値が整数型の場合は最大１０桁の１０進数で出力します。実数型の場合は整数部４桁、小数部３桁の最大７桁（小数点を含めて８桁）で出力します。数値の頭には符号が１文字分付きますが、＋の符号は省略します。１２文字中、数値を右詰めして、余った文字の部分にはスペースが詰められます。
データはすべてアスキーコードです。ＰＲＩＮＴ命令の最後にＣＲを指定するとデータの最後にキャリジリターンコードが付けられます。
ティーチペンダントに対してデータを出力すると、データをティーチペンダントに表示できます。
データ通信については、「通信編」にて説明していますので、そちらを参照してください。なおアーム動作中の停止操作後、本命令は実行されません。

サンプルプログラム

```
PROGRAM  PRINTSMPL
PRINT COM0,"*** INPUT N1,N2,N3 ***"
INPUT COM0,N1,N2,N3
PRINT (N1+N2+N3)/3, CR
END
```

ティーチペンダントから３つの数値Ｎ１～Ｎ３を読込み、その平均をティーチペンダントに表示します。

ＰＲＯＧＲＡＭ

機能

プログラムの始まりを示します。

フォーマット

ＰＲＯＧＲＡＭ＜プログラム名＞［（＜変数名＞，…）］

文例

```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
ＰＲＯＧＲＡＭ　ＭＡＩＮ
```

解説・注意

```
ＰＲＯＧＲＡＭ　ＳＵＢ１（Ｎ１，Ｎ２，Ｎ３）
```

プログラムの始まりを示します。
＜プログラム名＞には、プログラムの名前を識別子にて指定します。
プログラムは、ＰＲＯＧＲＡＭ文とＥＮＤ文の間に記述します。
サブプログラムを指定する場合には,必要に応じて(　)の中に引数を指定します。
サブプログラムと引数については「２．８　プログラム」を参照してください。

サンプルプログラム

```
PROGRAM  ENDSAMPLE
MOVE A1
MOVE A2
MOVE A3
END
```

ＰＲＯＧＲＡＭからＥＮＤまでが、１つのプログラムとして実行されます。

ＰＵＬＯＵＴ

機能

外部信号をパルス出力します。

フォーマット

ＰＵＬＯＵＴ（＜信号名＞［，＜信号名＞］・・・）

文例

ＰＵＬＯＵＴ（１，２，－３）
ＰＵＬＯＵＴ（Ｊ，Ｊ＋１，Ｊ＋２）

解説・注意

指定された出力信号を０．２秒幅のパルスで出力します。
＜信号名＞には、出力する信号の番号を指定します。＜信号名＞の符号が正ならば、信号はオフ・オン・オフと変化し、負ならば、オン・オフ・オンと変化します。信号名の符号が正の場合に既に信号がオン状態ならば、信号の状態は変化しません。同様に信号名の符号が負の場合に既に信号がオフ状態の時も、信号の状態は変化しません。＜信号名＞の指定は１０個までが有効になります。
＜信号名＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
信号のパルス出力は、パルス出力を指定した以外の信号の操作や、ロボットの動作と並行して行う事ができます。この場合には、ＥＮＡＢＬＥ　ＮＯＷＡＩＴを指定してください。ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴを指定すると、
ＰＵＬＯＵＴ命令に続く命令の処理は信号のパルス出力が終わるまで行われません。
ＥＮＡＢＬＥ　ＮＯＷＡＩＴを指定した場合に、ＰＵＬＯＵＴ命令に続けて同じ信号の出力処理を行うと、パルス出力は保証されません。

サンプルプログラム

```
PROGRAM  PULOUTSMPL
DISABLE NOWAIT          出力信号の１から１６までを順番に、0.2 秒間ずつオ
FOR K=1 TO 16           ンします。
PULOUT(K)
NEXT K
END
```

ＲＣＹＣＬＥ

機能

サイクルリセットを行うためのラベルです。

フォーマット

ＲＣＹＣＬＥ

文例

ＲＣＹＣＬＥ：

解説・注意

メインプログラムにおいて、最初のサイクルのみ先頭ステップからスタートし、第２サイクル以降はＲＣＹＣＬＥというラベルのついたステップから実行します。先頭ステップとＲＣＹＣＬＥの間に一度だけ実行したい命令、例えば毎サイクル増加するカウンタの初期化をプログラムすることができます。
こうすることで、パレットから部品を取出し（デパレタイズ）中のロボットが停止した場合でもそのパレットから取出し作業を継続することが可能となります。
なお、ＲＣＹＣＬＥは使用時には以下の注意事項があります。
記述できるのはメインプログラムで、しかも１つしか記述できません。
実行時に必ず1回はＲＣＹＣＬＥ：と記述した行を通過するようにプログラムしてください。
マルチタスクプログラムでは本機能は使用できません。サイクルリセット時にエラーとなります。
ＯＮ～ＤＯ命令においてＤＯ節実行中に本機能を使用することはできません。
サイクルリセット時にエラーとなります。
サイクルリセットの方法については取扱説明書「操作編」を参照してください

サンプルプログラム

```
PROGRAM RCYCLESMPL
COUNT=0
RCYCLE:
MOVE A1
MOVE A2
COUNT=COUNT+1
IF COUNT<=10 THEN GOTO RCYCLE
END
```

プログラムを途中で停止しても、変数ＣＯＵＮＴの値は保持されるため運転の続行が可能です。

ＲＥＡＤＹ

機能

ロボットを機械原点へ移動します。

フォーマット

ＲＥＡＤＹ

文例

ＲＥＡＤＹ

解説・注意

ロボットを各軸ごとに機械原点へ動かします。
本命令は、コントローラのＲＡＭドライブにＳＣＯＬ．ＬＩＢという名称のファイルがないと実行できません。
水平旋回型のロボットでは３軸→４軸→２軸→１軸→５軸の順番で動きます。３軸の機械原点は下方動作限付近になりますので注意してください。

サンプルプログラム

```
PROGRAM  READYSMPL
  READY                  機械原点へ動作します。
END
```

ＲＥＡＬ

機能

数値を実数に変換します。

フォーマット

ＲＥＡＬ（＜式＞）

文例

ＡＫ＝ＲＥＡＬ（－２０）
Ｎ＝ＲＥＡＬ（Ｋ）
Ｊ１＝Ｋ－ＲＥＡＬ（Ｎ－２８）

解説・注意

（　）内の式の演算結果を実数に変換します。
本命令は変数のデータ型を、特に実数型として指定したい場合に使用します。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は、式の中で使用します。

サンプルプログラム

```
PROGRAM  REALSAMPL
K=REAL(0.12345)              変数Ｋを実数型のデータとして定義します。
PRINT TP,K,CR
END
```

ＲＥＭＡＩＮ

機能

動作の残り移動量を参照します。

フォーマット

ＲＥＭＡＩＮ

文例

Ｋ＝ＲＥＭＡＩＮ
ＯＮ　ＲＥＭＡＩＮ＜＝５０　ＤＯ　ＤＯＵＴ（１）

解説・注意

ロボットの１つの動作の残りの移動量を参照します。
残り移動量は、ロボットが実行中の動作距離に対して、どれだけロボットの移動量が残っているかのパーセンテージで与えられます。移動量の算出には、その動作で最も移動量の大きい軸を基準にします。移動量は実数値として与えられます。
本命令とＯＮ命令を組合わせる事により、ロボットの動作途中で信号を出力する事ができます。
本命令は式の中で使用します。

注）ＲＥＭＡＩＮ命令で参照される移動量は、ロボットへの指令位置になります。実際のロボットの現在位置は、ロボットの動作中は指令位置から遅れている事に注意してください。
また、演算子＝＝は使用できませんので注意してください。

下記例はＰ１からＰ２へのﾊﾟｽ軌道生成待ちが発生してしまうため、無限ﾙｰﾌﾟになります。
ＷＡＩＴ文に置き換えることにより、無限ﾙｰﾌﾟは回避可能です。

```
ENABLE PASS                              ENABLE PASS
  PASS=50                                PASS=50
  MOVE P1                    →           MOVE P1
LOOP1:                                   WAIT REMAIN < 5
  IF REMAIN > 5 THEN GOTO LOOP1          MOVE P2
  MOVE P2
```

サンプルプログラム

```
PROGRAM  REMAINSMPL
ENABLE NOWAIT
ON REMAIN<=50 DO DOUT(1)
MOVE A1
ON REMAIN<=20 DO DOUT(2)
MOVE A2
END
```

Ａ１への移動が５０％以上進んだら信号１を出力し、Ａ２への移動が８０％以上進んだら（残量が20% 以下になったら）信号２を出力します。

ＲＥＭＡＩＮＴ

機能

動作の残り実行時間を参照します。

フォーマット

ＲＥＭＡＩＮＴ

文例

Ｋ＝ＲＥＭＡＩＮＴ
ＯＮ　ＲＥＭＡＩＮＴ＜＝１　ＤＯ　ＤＯＵＴ（１）

解説・注意

ロボットが１つの動作を終了するまでにかかる時間を参照します。
残り実行時間は、秒単位の実数値で与えられ、ロボットが位置決め完了すると０になります。
本命令とＯＮ命令を組合わせる事により、ロボットの動作途中で信号を出力することができます。
本命令は式の中で使用します。
また、演算子＝＝は使用できませんので注意してください。

下記例はＰ１からＰ２へのﾊﾟｽ軌道生成待ちが発生してしまうため、無限ﾙｰﾌﾟになります。
ＷＡＩＴ文に置き換えることにより、無限ﾙｰﾌﾟは回避可能です。

```
ENABLE PASS                              ENABLE PASS
  PASS=50                                PASS=50
  MOVE P1                    →           MOVE P1
LOOP1:                                   WAIT REMAINT < 1
  IF REMAINT > 1 THEN GOTO LOOP1         MOVE P2
  MOVE P2
```

サンプルプログラム

```
PROGRAM  REMAINTSMPL
ENABLE NOWAIT                    Ａ１への移動が、あと１秒以内になった時に
ON REMAINT<=1 DO DOUT(1)         信号１を出力します。
MOVE A1
MOVE A2
END
```

ＲＥＭＡＲＫ

機能

注釈文を記述します。

フォーマット

ＲＥＭＡＲＫ［＜注釈＞］

文例

ＲＥＭＡＲＫ　＊＊＊　ＳＣＯＬ　ＳＡＭＰＬＥ　＊＊＊

解説・注意

プログラムを見易くするための注釈文を記述します。
ＲＥＭＡＲＫ文は、全て注釈として解釈されます。
注釈文は実行されません。
“´”もＲＥＭＡＲＫ文と同じ意味で利用できます。
他の命令語に続けて注釈を記述する場合には、記号”´”を使用します。記号”´”に続けて記述した文字は、全て注釈と見なされます。

サンプルプログラム

```
PROGRAM  REMARKSMPL
REMARK ***SAMPLKE PROGRAM***        注釈文を記述します。
MOVE A1 'MOVES TO A1                “MOVES TO A1”を注釈にします。
MOVE A2
END
```

ＲＥＳＥＴ

機能

コントローラの状態をリセットします。

フォーマット

ＲＥＳＥＴ　＜状態＞［，＜状態＞］

文例

ＲＥＳＥＴ　ＤＯＵＴ
ＲＥＳＥＴ　ＲＥＳＵＭＥ

解説・注意

出力信号などのコントローラの状態をリセットします。
＜状態＞としては次の２つを指定できます。

（１）　ＤＯＵＴ
ユーザ用出力信号をすべてオフにします。

（２）　ＲＥＳＵＭＥ
ＯＮ命令にて、ＢＲＥＡＫを指定して、中断した動作をリセットします。
リセットを行った後はＢＲＥＡＫ命令で中断した動作をＲＥＳＵＭＥ命令で再開できなくなります。

サンプルプログラム

```
PROGRAM  RESETSMPL
RESET DOUT              ユーザ用出力信号を全てオフにします。
END
```

ＲＥＳＴＯＲＥ

機能	グローバル変数初期値を、更新します。
フォーマット	ＲＥＳＴＯＲＥ(“<変数>”)
文例	ＲＥＳＴＯＲＥ(“Ｉ”)
解説・注意	教示済み位置データの変更等をプログラムファイルに戻します。 指定できる変数は、初期値を持たない配列以外のグローバル変数のみです。 それ以外の変数をＲＥＳＴＯＲＥしようとすると、“ＳＥＬＥＣＴ”時にエラーとなります。 ＲＥＳＴＯＲＥ命令はプログラムファイルを上書きします。ＲＥＳＴＯＲＥ命令実行中にコントローラの電源をＯＦＦすると、ファイルの上書きが中断されるため、ファイルが消えてしまう場合があります。 ＲＥＳＴＯＲＥを含むプログラムを実行している場合には、必ずプログラムを停止(ＳＴＯＰ)させてから電源をＯＦＦしてください。 頻繁にＲＥＳＴＯＲＥ命令を使用するシステムでは、ＳＡＶＥＦ・ＳＡＶＥＩの使用を検討ください。

サンプルプログラム

```
GLOBAL
A=0
END
PROGRAM STORETEST
A=A+1
RESTORE(“A”)
PRINT ″A=″,A,CR
END
```

ＲＥＳＵＭＥ

機能

ロボットの動作を再開します。

フォーマット

ＲＥＳＵＭＥ

文例

ＲＥＳＵＭＥ

解説・注意

ＯＮ命令にてＢＲＥＡＫを指定して中断したロボットの動作を再開します。
動作の再開は、ロボットの現在位置からとなります。このため、円弧補間動作を再開すると、現在位置と経由点、目標位置の関係によっては軌跡が大きく変化する場合があります。
ＢＲＥＡＫ文が重なった場合には、最後に中断した動作のみ再開する事ができます。
ＲＥＳＥＴ　ＲＥＳＵＭＥを実行すると、中断した動作はリセットします。また、ＯＮ命令でＰＡＵＳＥが実行された場合も中断動作はリセットされます。
本命令は、ＤＯ節内でのみ有効です。ＤＯ節以外で実行した場合、本命令は無視されます。

サンプルプログラム

```
PROGRAM  RESUMESMPL
ENABLE NOWAIT
REMARK *** MAIN PROGRAM ***          システムに異常が発生して入力信号の２４
ON DIN(24) BREAK DO BREAKSUB         がオン状態になると、ロボットの動作を即
MOVE A1                              時中断して異常処理のサブプログラム、
MOVE A2                              ＢＲＥＡＫＳＵＢに処理が移ります。
MOVE A3
WAIT MOTION>=100
IGNORE DIN(24)
END

PROGRAM  BREAKSUB
REMARK *** SUBROUTINE ***            異常処理のサブプログラムでは、異常の入
WAIT DIN(-24)                        力信号２４がオフ状態になるまで待ちます。
RESUME                               異常状態が解除すると、中断した動作から
END                                  再開します。
```

ＲＥＴＵＲＮ

機能

サブプログラムからメインプログラムへと戻ります。

フォーマット

ＲＥＴＵＲＮ

文例

ＲＥＴＵＲＮ

解説・注意

サブプログラムからメインプログラムに戻ります。
サブプログラムでは、ＲＥＴＵＲＮ文を記述しなくても、ＥＮＤ文の実行にて制御はメインプログラムへ戻ります。
メインプログラムにＲＥＴＵＲＮ命令があると、エラーになります。

サンプルプログラム

```
PROGRAM  MAIN
RETURNSMPL(5,K)
PRINT TP,K,CR
END
PROGRAM  RETURNSMPL(N,K)
K=N*N                          引数Ｎの二乗を計算してその結果を引数Ｋとし
RETURN                         てメインプログラムに返します。
END
```

ＲＩＧＨＴＹ

機能

ロボットの姿勢を右肩系にするためのシステム定数です。

フォーマット

ＲＩＧＨＴＹ

文例

ＣＯＮＦＩＧ＝ＲＩＧＨＴＹ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＣＯＮＦＩＧ＝ＲＩＧＨＴＹ

解説・注意

ＣＯＮＦＩＧ命令と共に使用して、ロボットの姿勢を右肩系に設定します。
本システム定数は、２という値を持っています。プログラムの式の中で定数２として使用する事もできますが、プログラムが見づらくなるため、このような使い方はしないでください。
システム定数への代入はできません。
直角座標ロボットでは、ロボットの姿勢の指定は無視します。
ロボットの姿勢については、ＣＯＮＦＩＧ命令を参照してください。

サンプルプログラム

```
PROGRAM  RIGHTYSMPL
CONFIG=RIGHTY          ロボットの姿勢を右肩系として動作します。
MOVE A1
MOVE A2
END
```

ＳＡＶＥＥＮＤ

機能

ロボット停止時にバックアップメモリへ保存するグローバル変数を指定します。

フォーマット

ＳＡＶＥＥＮＤ：

文例

ＳＡＶＥＥＮＤ：

解説・注意

・ **ロボット停止時に,**指定された変数がバックアップメモリへ保存されます。

・ 保存したい変数は、グローバルブロックの先頭と“ＳＡＶＥＥＮＤ：”の間に記述します。

・ 次回電源ＯＮ時には、バックアップメモリに保存された値が復元されます。

・ **１０Ｋバイトまで保存可能です。**

・ 指定されたグローバル変数は、以下の停止条件で保存されます。

① サイクルストップ

② フィードホールド

③ ブレーク

④ 非常停止

⑤ サーボＯＦＦ

・ また以下の操作タイミングで、保存されている情報がクリアされます。

① プログラムセレクト

② プログラムリセット

③ 動作中（ステータスがｒｕｎｎｉｎｇ）での主電源断

＜注意＞

- **データを保存するには時間が掛かります（１Ｋバイトあたり１００ｍｓｅｃ）**
- **データ領域を大きくとる場合はロボットを停止させてから、１秒程度待った後、電源をＯＦＦしてください。**
- システム変数については一部を除き保存されます。（保存されない変数：メインプログラム以外のＴＩＭＥＲ変数及び、ＴＩＤ変数）
- グローバルブロックに記述できる変数の型については取扱説明書「言語編」「２．８．５ グローバル変数定義」で説明されている制限があります。この制限によりグローバルブロックに記述できない変数は、本機能で保存できません。

サンプル
プログラム

```
GLOBAL
I=0             ‘停止時に保存されます。
SAVEEND:
J=0             ‘停止時に保存されません。
END
PROGRAM MAIN
FOR K=1 TO 10
I=I+1
J=J+1
NEXT
PRINT “I=″,I,“J=″,J,CR
END
```

電源 OFF→ON 時のデータ復元までの流れ

① サイクル運転モードでプログラム「ＭＡＩＮ」を起動します。
② ＴＰに「Ｉ＝１０、Ｊ＝１０」と表示されます。
③ ＥＮＤ にてプログラム停止。⇒「Ｉ＝１０」がバックアップメモリへ保存されます。
④ （電源ＯＦＦ → 電源ＯＮ）
⑤ プログラム「ＭＡＩＮ」を起動します。
⑥ ＴＰに「Ｉ＝２０、Ｊ＝１０」と表示されます。

ＳＥＧＭＥＮＴ

機能

システムの運転モードを参照するためのシステム定数です。

フォーマット

ＳＥＧＭＥＮＴ

文例

ＩＦ　ＭＯＤＥ＝＝ＳＥＧＭＥＮＴ　ＴＨＥＮ　ＲＥＴＵＲＮ

解説・注意

ＭＯＤＥ命令と共に使用して、システムの運転モードを参照します。
ＭＯＤＥ＝＝ＳＥＧＭＥＮＴのとき、システムはセグメント運転モードで動作しています。
本システム定数は、２という値を持っています。プログラムの式の中で定数２として使用することもできますが、プログラムが見づらくなるためこのような使い方はしないでください。
システム定数への代入はできません。
モニタコマンドではＭＯＤＥ　ＭＯＴＩＯＮにてセグメント運転モードを指定できます。
システムの運転モードについては、ＭＯＤＥ命令を参照してください。

サンプルプログラム

```
PROGRAM  MAIN
SEGMENTSAMPLE
END
PROGRAM  SEGMENTSAMPLE
IF MODE==SEGMENT THEN RETURN        システムがセグメント運転モードで動作し
MOVE A1                             ていれば、このサブプログラムを実行せず
MOVE A2                             にメインプログラムに戻ります。
MOVE A3
END
```

ＳＥＴＧＡＩＮ

機能

各軸のゲイン（サーボ制御）のオン，オフを指定します。

フォーマット

ＳＥＴＧＡＩＮ（＜整数＞，＜整数＞，＜整数＞，＜整数＞，＜整数＞）

文例

ＳＥＴＧＡＩＮ（０，　０，　１，　０，　０）

解説・注意

ロボットの各軸のゲイン（サーボ制御）のオン，オフを現在実行中の動作命令の終了を待って設定します。

ゲインの設定が有効になるのは本命令後の動作命令を実行した時です。ゲインについては、ＧＡＩＮ命令を参照してください。

本命令は、コントローラのＲＡＭドライブにＳＣＯＬ．ＬＩＢという名称のファイルがないと実行できません。

本命令ではシステム定数のＯＮ、ＯＦＦ は使用できません。ゲインのＯＮ、ＯＦＦを行うために、ＯＮＧＡＩＮ，ＯＦＦＧＡＩＮの２つの命令をライブラリファイルに用意しています。

サンプルプログラム

```
PROGRAM SETGAINSMPL
MOVE A1                    ; Ａ１への到着を待って、３軸以外のゲイン
SETGAIN(0,0,1,0,0)         ; をオフします。
MOVE A2                    ; Ａ２への到着を待って、全ての軸のゲイン
SETGAIN(1,1,1,1,1)         ; をオンします。
MOVE A3
END
```

ＳＧＮ

機能	数値の符号を取出します。
フォーマット	ＳＧＮ（＜式＞）
文例	Ｓ＝ＳＧＮ（－２０．３４５） Ｎ＝ＡＢＳ（Ｋ）＊ＳＧＮ（Ｌ）
解説・注意	（　）内の式の演算結果の符号を取出します。 ＳＧＮの実行結果は、式の値が正ならば１、負ならば－１、０ならば０になります。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
SGNSAMPLE(5,3,K)
PRINT TP,K,CR
END
PROGRAM  SGNSAMPLE(K1,K2,K)
K=SGN(K1-K2)
RETURN
END
```

引数Ｋ１がＫ２よりも大きければ１を、同じならば０を、小さければ－１を引数Ｋとしてメインプログラムに返します。

ＳＩＮ

機能

正弦の値を計算します。

フォーマット

ＳＩＮ（＜式＞）

文例

Ｋ＝ＳＩＮ（６０）
Ｊ１＝１－ＳＩＮ（１８０－Ｄ）

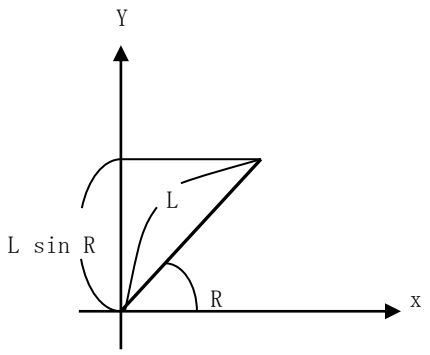
解説・注意

（　）内の正弦の値を計算します。
計算は度単位で行います。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
SINSAMPLE(2.0,60.0,Y)
PRINT TP,Y,CR
END
PROGRAM  SINSAMPLE(L,R,Y)
LOOP:
IF R>180 THEN R=R-360
IF R<-180 THEN R=R+360
IF R>180 OR R<-180 THEN GOTO LOOP
Y=L*SIN(R)
RETURN
END
```

Ｘ軸とＲの角度をなす線分Ｌ（長さＬ）にて、引数Ｌ、Ｒの値から線分ＬのＹ軸成分を求めて、その結果を引数Ｙとしてメインプログラムに返します。



ＳＭＯＯＴＨ

機能

スムース動作を指定します。
（スムース動作のパラメータの設定については、次ページを参照して下さい。）

フォーマット

ＳＭＯＯＴＨ

文例

ＤＩＳＡＢＬＥ　ＳＭＯＯＴＨ
ＥＮＡＢＬＥ　ＳＭＯＯＴＨ

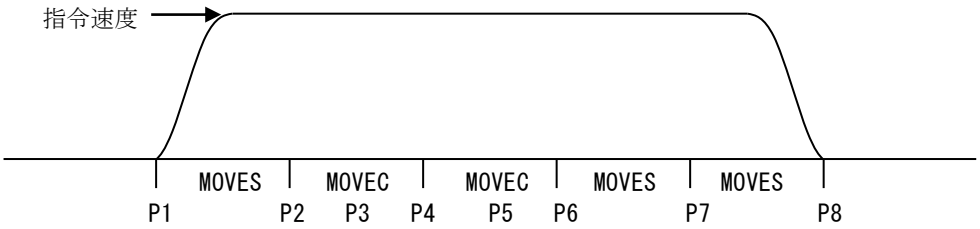
解説・注意

スムース動作を指定します。
スムース指定された動作命令は目標位置まで減速することなく移動し、続く動作命令はスムース指定の有無に関係なく加速せずに移動を開始します。
システムスイッチのオン、オフには、ＥＮＡＢＬＥ，ＤＩＳＡＢＬＥ命令を使用します。
ＥＮＡＢＬＥ　ＳＭＯＯＴＨでスムース動作を開始し、
ＤＩＳＡＢＬＥ　ＳＭＯＯＴＨでスムース動作を解除します。
初期状態では、ＤＩＳＡＢＬＥ　ＳＭＯＯＴＨになっています。
補間動作命令(ＭＯＶＥＳ,ＭＯＶＥＣ)のみスムース機能の制御対象となります。
ＥＮＡＢＬＥ　ＳＭＯＯＴＨ中は位置決め精度が自動的にＣＯＡＲＳＥになります。
スムース接続時の速度がその動作命令の指定速度に未到達であった場合、最大加速度で指定速度まで加速（減速）します。
スムース指定された動作命令のＣ軸移動量がＸＹＺ方向移動量に対して十分でない場合、動作命令がスムース指定された場合はエラーとなります。
Ｔ軸の移動を含む動作命令がスムース指定された場合はエラーとなります。
ＤＩＳＡＢＬＥ　ＳＭＯＯＴＨされた動作命令が減速停止に充分な移動量を持たない場合、減速停止時の速度制御は保証できません。
ＰＡＳＳ機能とＳＭＯＯＴＨ機能は併用できません。また、ＰＡＳＳ動作とＳＭＯＯＴＨ動作は接続することができません。
（サンプルプログラムＳＭＰＬ０４，ＳＭＰＬ０５参照）

サンプル
プログラム

```
PROGRAM SMPL01
MOVE  P01
ENABLE SMOOTH
MOVES P02
MOVEC P03 P04
MOVEC P05 P06
MOVES P07
DISABLE SMOOTH
MOVES P08
END
```

P02,P04,P06,P07ポイントをスムース接続し、P08で減速停止します。



（悪い例１）

```
PROGRAM SMPL02
  PASS=50
  ENABLE SMOOTH
  MOVES P01
  MOVES P02
  MOVES P03
  ENABLE PASS  ←アラーム発生　２－０３９“ＰＡＳＳ命令禁止”
  MOVES P04
  DISABLE SMOOTH
  MOVES P05
  MOVES P06
  MOVES P07
  DISABLE PASS
  MOVES P08
END
```

ＥＮＡＢＬＥ　ＳＭＯＯＴＨ中にＥＮＡＢＬＥ　ＰＡＳＳ指定した場合、アラームが発生し、ロボットは動作を停止します。

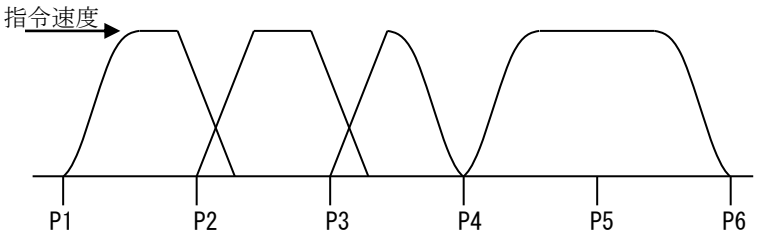
（悪い例２）

```
PROGRAM SMPL03
  PASS=50
  ENABLE PASS
  MOVES P01
  MOVES P02
  MOVES P03
  ENABLE SMOOTH  ←アラーム発生　２－０４０　“ＳＭＯＯＴＨ命令禁止”
  MOVES P04
  DISABLE PASS
  MOVES P05
  MOVES P06
  MOVES P07
  DISABLE SMOOTH
  MOVES P08
END
```

ＥＮＡＢＬＥ　ＰＡＳＳ中にＥＮＡＢＬＥ　ＳＭＯＯＴＨ指定した場合、アラームが発生し、ロボットは動作を停止します。

（ＰＡＳＳからＳＭＯＯＴＨへ移行する場合の例）

```
PROGRAM SMPL04
  PASS=50
  MOVE P01
  ENABLE PASS
  MOVES P02
  MOVES P03
  DISABLE PASS
  ENABLE SMOOTH
  MOVES P04
  MOVES P05
  DISABLE SMOOTH
  MOVES P06
END
```



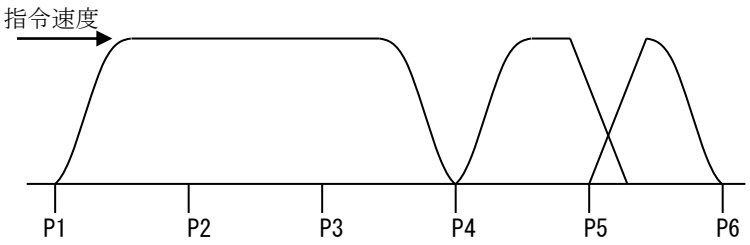
ＤＩＳＡＢＬＥ　ＰＡＳＳ時に減速停止をするため、Ｐ０４ポイントのＳＭＯＯＴＨ指定は無効になります。

すなわち、Ｐ０２，Ｐ０３ポイントをショートカット接続し、Ｐ０４ポイントで減速停止接続した後、Ｐ０５ポイントをスムース接続し、Ｐ０６ポイントで減速停止します。

このとき、Ｐ４ポイントでは、１－０１８“ＳＭＯＯＴＨ接続無効”アラームが発生します。

（ＳＭＯＯＴＨからＰＡＳＳへ移行する場合の例）

```
PROGRAM SMPL05
  PASS=50
  MOVE P01
  ENABLE SMOOTH
  MOVES P02
  MOVES P03
  DISABLE SMOOTH
  ENABLE PASS
  MOVES P04
  MOVES P05
  DISABLE PASS
  MOVES P06
END
```



ＤＩＳＡＢＬＥ　ＳＭＯＯＴＨ時に減速停止をするため、P04ポイントのＰＡＳＳ指定は無効になります。

すなわち、Ｐ０２，Ｐ０３ポイントをスムース接続し、Ｐ０４ポイントで減速停止接続した後、Ｐ０５ポイントをショートカット接続し、Ｐ０６ポイントで減速停止します。

このとき、Ｐ４ポイントでは、１－０１７“ＰＡＳＳ接続無効”アラームが発生します。

注意：　本機能の使用条件によっては機械の寿命に影響を与える場合がありますので使用にあたっては弊社と事前に打ち合わせをお願いします。

ＳＰＥＥＤ

機能

ロボットの動作速度を指定します。

フォーマット

ＳＰＥＥＤ＝＜式＞

文例

ＳＰＥＥＤ＝５０

ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＳＰＥＥＤ＝ＳＰＥＥＤ＊０．８

解説・注意

ロボットの動作速度を、最高速に対するパーセンテージで指定するためのシステム変数です。

ＳＰＥＥＤは、正の整数値で指定します。０以下の数値を指定すると、１が指定されたものとして処理されます。また１００以上の値を指定してもシステムに設定された最高速度におさえられます。

＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。

本システム変数を参照すると、現在の動作速度が参照できます。動作速度の初期値は100%になっています。

サンプルプログラム

```
PROGRAM SPEEDSMPL
SPEED=50                    動作速度を最高速の５０％で動作します。
MOVE A1                     Ａ３の動作では最高速で動作します。
MOVE A2
MOVE A3 WITH SPEED=100
MOVE A4
END
```

ＳＱＲＴ

機能

平方根の値を計算します。

フォーマット

ＳＱＲＴ（＜式＞）

Ｊ１＝ＳＱＲＴ（Ｌ１＾２＋Ｌ２＾２）

文例

Ｋ＝ＳＱＲＴ（６０）

Ｊ１＝１－ＳＩＮ（１８０－Ｄ）

解説・注意

（　）内の平方根の値を計算します。

＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。

＜式＞の値が負値の場合は、エラーになりますので注意してください。

本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
SQRTSAMPLE(3,5,L)
PRINT TP,L,CR
END
PROGRAM  SQRTSAMPLE(X,Y,L)
L=SQRT(X ^ 2+ Y ^ 2)
RETURN
END
```

引数Ｘ，Ｙからその長さを計算して、その結果を引数Ｌとして、メインプログラムに返します。

ＳＴＥＰ

機能

ＦＯＲ命令と組合わせて、プログラムの繰返しを指定します。

フォーマット

ＦＯＲ＜変数＞＝＜式１＞ＴＯ＜式２＞［ＳＴＥＰ＜式３＞］

文例

ＦＯＲ　Ｋ＝１　ＴＯ　４　ＳＴＥＰ　２
ＦＯＲ　Ｎ＝Ｋ１　ＴＯ　Ｋ１＋Ｋ２　ＳＴＥＰ　Ｋ３

解説・注意

プログラムの繰返しを指定します。
ＦＯＲ命令とＮＥＸＴ命令にはさまれたプログラムを、繰返し実行します。プログラムの実行は、ＦＯＲ文で指定した条件が満たされるまで繰返します。
ＦＯＲ文を実行すると、＜変数＞に＜式１＞の値が代入されます。ＮＥＸＴ文の実行にて、＜変数＞にはＳＴＥＰ文にて指定した＜式３＞の値が加えられます。この時にプログラムの実行は、変数の値が＜式２＞の値を超えていればＮＥＸＴの次の命令に、超えていなければ対応するＦＯＲ文の次の文へと分岐します。
＜式３＞の値は、最初にＦＯＲ文を実行した時の値が用いられます。このため、式の値がプログラムの繰返し中に変化しても、プログラムの繰返し回数は変化しません。
＜式３＞の値が１の時には、ＳＴＥＰ以後を省略する事ができます。
＜式３＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
プログラムの繰返し条件については、ＦＯＲ命令を参照してください。

サンプルプログラム

```
PROGRAM  STEPSAMPLE
FOR K=1 TO 100 STEP 2          ＡＩ，Ａ２間の往復動作を、動作速度を
MOVE A1 WITH SPEED=K           １～１００％まで２％きざみで変えながら
MOVE A2                        動作します。
NEXT K
END
```

ＳＴＯＰ

機能

プログラムの実行を停止します。

フォーマット

ＳＴＯＰ

文例

ＳＴＯＰ

解説・注意

ＳＴＯＰ命令実行にてプログラムの実行を停止します。
プログラムの実行は、システムの運転モードにかかわらず停止します。
実行を再開するためには、再度プログラムの起動の操作が必要です。
再起動すると、ロボットは続きのステップから実行を再開します。

サンプルプログラム

```
PROGRAM  STOPSAMPLE
ENABLE NOWAIT
ON DIN(10) DO STOP          ロボットの動作中に入力信号１０がオンしたら、
MOVE A1                     プログラムの実行を停止します。
MOVE A2
MOVE A3
END
```

ＳＷＩＴＣＨ

機能	マルチタスク時に、強制的に他のタスクに切替えます。
フォーマット	ＳＷＩＴＣＨ
文例	ＳＷＩＴＣＨ
解説・注意	シングルタスク時やシステム変数“ＳＷＩＴＣＨ”がＤＩＳＡＢＬＥ状態、及びステップ実行中は、本命令は無効です。

サンプル
プログラム

```
GLOBAL
MAXTASK=2
END
PROGRAM MAIN
ID = 0                          ２つのタスクで同一のI/Oを使用するので
ID = TASK("SUB1")               DIN(1)がOFFの状態ではどちらかのタスクに
LOOP:                           I/Oが占有されてしまいます。
IF DIN(1) THEN SWITCH           DIN(1)をONすることによって強制的にタスクが
PRINT " TASK1 ",CR              切り替え、占有が発生しないようにしています。
GOTO LOOP
END
PROGRAM SUB1
ENABLE NOWAIT
LOOP1:
IF DIN(1) THEN SWITCH
PRINT " TASK2",CR
GOTO LOOP1
END
```

ＳＷＩＴＣＨ

機能

タスク切替えを禁止／許可します。

フォーマット

ＳＷＩＴＣＨ

文例

ＥＮＡＢＬＥ ＳＷＩＴＣＨ
ＤＩＳＡＢＬＥ ＳＷＩＴＣＨ

解説・注意

本システム変数をＤＩＳＡＢＬＥ状態に切替えると、ＳＷＩＴＣＨ命令はもちろん、たとえシステムで決められたタスク切替条件が成立しても、タスクは切替わりません。本システム変数の初期状態は、ＥＮＡＢＬＥです。

サンプルプログラム

```
GLOBAL
MAXTASK=2
END
PROGRAM MAIN
ID = 0
ID = TASK("SUB1")
LOOP:                          メインタスクのループ開始時にタスク切り換え有効
IF DIN(1) THEN DISABLE SWITCH  ELSE ENABLE SWITCH     無効をDIN(1)の入力
MOVEA 1,90                     により選択します。タスク切り換えがDISABLEされると
MOVEA 1,-90                    サブタスクが動作を行わなくなるのでDOUTが変化しません。
GOTO LOOP
END
PROGRAM SUB1
ENABLE NOWAIT
DOUT (-1,-2)
TIMER=1
WAIT TIMER==0
DOUT (1)
TIMER=1
WAIT TIMER==0
DOUT (2)
TIMER=1
WAIT TIMER==0
END
```

ＴＡＮ

機能

正接の値を計算します。

フォーマット

ＴＡＮ（＜式＞）

文例

Ｋ＝ＴＡＮ（６０）
Ｊ１＝１－ＴＡＮ（１８０－Ｄ）

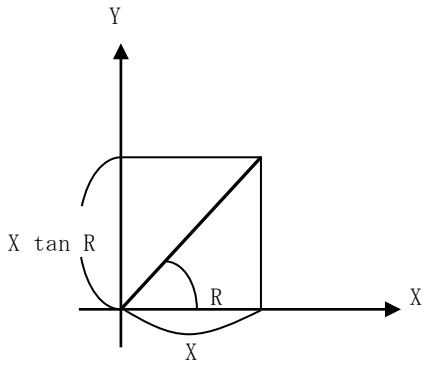
解説・注意

（　）内の正接の値を計算します。
計算は度単位で行います。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
本命令は式の中で使用します。

サンプルプログラム

```
PROGRAM  MAIN
TANSAMPLE(2,60,Y)
PRINT TP,Y,CR
END
PROGRAM  TANSAMPLE(X,R,Y)
LOOP:
IF R>180 THEN R=R-360
IF R<-180 THEN R=R+360
IF R>180 OR R<-180 THEN GOTO LOOP
Y=X*TAN(R)
RETURN
END
```

Ｘ軸とＲの角度をなす線分のＸ軸成分Ｘと角度Ｒの値から、線分のＹ軸成分を求めて、その結果を引数Ｙとしてメインプログラムに返します。



ＴＡＳＫ

機能	マルチタスクを実行します
フォーマット	ＴＡＳＫ（“＜プログラム名＞”）
文例	ＩＤ＝ＴＡＳＫ（“ＳＵＢ”）
解説・注意	（　）内で示されるプログラムをタスクとして起動します。 戻り値は起動されたサブタスクに固有の番号です。（タスクＩＤ）タスクＩＤはタスクの停止時の引数として使用します。サブタスクで入出力命令を用いる場合にはサブタスクでENBALE NOWAITを宣言してください。

サンプルプログラム

```
GLOBAL
ID=0
MAXTASK=2
END
PROGRAM MAIN
IF ID ==0 THEN ID= TASK(“SUB”)    ＳＵＢをタスクとして起動
LOOP:
MOVEA 1,90
MOVEA 1,-90
END
PROGRAM SUB
ENABLE NOWAIT
WAIT DIN (1)
DOUT (1)                          メインタスクでの動作とは非同期に
WAIT DIN (-1)                     信号入力に反応して信号を出力します
DOUT (-1)
END
```

ＴＨＥＮ

機能

ＩＦ文と組合わせて条件判断を行います。

フォーマット

ＩＦ＜論理式＞ＴＨＥＮ＜文＞［ＥＬＳＥ＜文＞］

文例

```
ＩＦ　ＤＩＮ（１）　ＴＨＥＮ　Ｋ＝Ｋ＋１　ＥＬＳＥ　Ｋ＝０
```

解説・注意

ＩＦ文を用いた条件判断にて条件が成立した場合に、実行する文を指定します。
条件が成立しなかった場合には、ＥＬＳＥに続く文を実行します。
ＥＬＳＥ＜文＞が省略されていると、条件が成立しなかった場合に、ＩＦ文の次の命令を実行します。
ＴＨＥＮ，ＥＬＳＥに続く＜文＞には、ＰＲＯＧＲＡＭ，ＥＮＤ，ＩＦ，ＦＯＲ，ＮＥＸＴ及びＷＡＩＴ命令は使用できません。
条件判断については、ＩＦ文を参照してください。

サンプルプログラム

```
PROGRAM  THENSAMPLE
IF DIN(1) THEN K=1 ELSE K=0
MOVE A1                          入力信号１がオンの場合にはＫ＝１、入力
MOVE A2                          信号１がオフの場合にはＫ＝０にします。
MOVE A3
PRINT "K=",K,CR
END
```

ＴＩＤ

機能

自タスクのタスクＩＤ（番号）を参照します。

フォーマット

ＴＩＤ

文例

ＭＡＩＮＩＤ＝ＴＩＤ

解説・注意

本システム変数は、ＴＡＳＫ命令で起動されるタスク毎に固有です。
本システム変数への書き込みは行えません。

サンプルプログラム

```
GLOBAL
MAXTASK=2
END
PROGRAM MAIN
 TASK(“SUB”)
LOOP:
 SWITCH
 PRINT “MAINID=”,TID,CR
GOTO LOOP
END
PROGRAM SUB
 ENABLE NOWAIT
 SWITCH
 PRINT “SUBID=”,TID,CR
END
```

ＴＩＤをメインタスクとサブタスクで表示しています。
それぞれの値が違うことがわかります。

ＴＩＭＥＲ

機能

システムで更新するタイマです。

フォーマット

ＴＩＭＥＲ

文例

ＴＩＭＥＲ＝２０
ＷＡＩＴ　ＴＩＭＥＲ＝＝０

解説・注意

ＳＣＯＬ言語のプログラム中で使用できるタイマです。
タイマの値は、システム変数ＴＩＭＥＲに秒単位の値で設定します。
０以下の値を設定すると正しく動作しません。
システム変数ＴＩＭＥＲの値は、設定と同時に減算していきます。値が０になるとそれ以上の減算は行いません。本システム変数を参照すると、タイマの残り時間を参照できます。

サンプルプログラム

```
PROGRAM  TIMERSMPL
TIMER=5                 タイマの値を５秒に設定し、この値が０になるまで
WAIT TIMER==0           待ちます。
PRINT TIMER,CR
END

PROGRAM  TIMERSMPL2
FOR J=1 TO 1000         １秒おきに”＊”をティーチペンダントに表示しま
PRINT "*",CR            す。（１０００回繰り返し）
TIMER=1
WAIT TIMER==0
NEXT J
END
```

ＴＯ

機能

ＦＯＲ命令と組合わせてプログラムの繰返しを指定します。

フォーマット

ＦＯＲ＜変数＞＝＜式１＞ＴＯ＜式２＞［ＳＴＥＰ＜式３＞］

文例

```
ＦＯＲ　Ｋ＝１　ＴＯ　４
ＦＯＲ　Ｎ＝Ｋ１　ＴＯ　Ｋ１＋Ｋ２　ＳＴＥＰ　Ｋ３
```

解説・注意

ＦＯＲ命令と組合わせてプログラムの繰返しの条件を指定します。
ＦＯＲ命令とＮＥＸＴ命令にはさまれたプログラムを繰返し実行します。プログラムの実行は、ＦＯＲ文で指定した条件が満たされるまで繰返します。
ＦＯＲ文を実行すると、＜変数＞に＜式１＞の値が代入されます。
ＮＥＸＴ文の実行にて、＜変数＞にはＳＴＥＰ文にて指定した＜式３＞の値が加えられます。この時に、プログラムの実行は、変数の値が＜式２＞の値を越えていればＮＥＸＴの次の命令に、超えていなければ対応するＦＯＲ文の次の文へと分岐します。
＜式１＞、＜式２＞、＜式３＞は、最初にＦＯＲ文を実行した時の値を用います。このため、これらの式の値がプログラムの繰返し中に変化しても、プログラムの繰返し回数は変化しません。
＜式１＞、＜式２＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
プログラムの繰返しについては、ＦＯＲ命令を参照してください。

サンプルプログラム

```
PROGRAM  TOSAMPLE
FOR K=1 TO 100          Ａ１，Ａ２間の往復動作を１００回繰返します。
MOVE A1
MOVE A2
NEXT K
END
```

ＴＯＯＬ

機能

ツール座標系を指定します。

フォーマット

ＴＯＯＬ

文例

```
ＴＯＯＬ＝ＴＲＡＮＳ（０，０，０，０）
ＴＯＯＬ１＝ＴＯＯＬ
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＴＯＯＬ＝　ＴＯＯＬ＋
                              ＴＲＡＮＳ（，，１００）
```

解説・注意

ツール座標系を指定するシステム変数です。
システム変数ＴＯＯＬは、通常の座標型データとして扱えます。
本システム変数を参照すると、現在のツール座標系の値が参照できます。
次のようにして、ツール座標系に直接座標値を指定する事ができます。

ＴＯＯＬ＝ＴＲＡＮＳ（Ｘ，Ｙ，Ｚ，Ｃ）
ＴＯＯＬ＝｛Ｘ，Ｙ，Ｚ，Ｃ｝

（データ型を明確にするためにＴＲＡＮＳ命令を使用するようにしてください。）

Ｘ，Ｙ，Ｚ，Ｃ　：Ｘ，Ｙ，Ｚ，Ｃの各座標値で、実数値をとります。
（ｍｍ，ｄｅｇ　単位）

ツール座標系は、メカニカルインタフェース座標系のそれぞれの座標軸に沿って、Ｘ，Ｙ，Ｚの量だけ平行移動した座標系を、新しいＺ軸の回りにＣだけ回転した座標系になります。
本命令は式の中で使用します。
プログラム中でツール座標系を変更すると、以降の全動作で教示した位置がずれてしまいますので注意してください。

サンプルプログラム

```
PROGRAM  TOOLSAMPLE
  MOVE A1                          ＴＯＯＬ命令によりツール座標系のＺ軸
  MOVE A2                          ５００ｍｍずらして、以降の動作では教示した
  TOOL=TOOL+TRANS(,,500)           位置より５００ｍｍ上方の点に動作します。
  MOVE A1
  MOVE A2
  TOOL=TRANS()
END
```

ＴＯＲＱＵＥ

機能

各軸のトルクの制限値を指定します。

フォーマット

ＴＯＲＱＵＥ＝｛＜式＞，＜式＞，＜式＞，＜式＞，＜式＞｝

文例

ＴＯＲＱＵＥ＝｛３００，３００，１００，３００，３００｝
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＴＯＲＱＵＥ＝｛Ｔ，Ｔ，Ｔ，Ｔ，Ｔ｝

解説・注意

各軸のトルクの制限値を指定します。
ＴＯＲＱＵＥは、システム変数として扱われます。本システム変数は５軸分の値を持っています。トルクは、ＴＯＲＱＵＥに続けて｛　｝の中に各軸に対する制限値を指定します。｛　｝内のデータは５軸分を”，”で区切って指定し、左から順番に１～５軸に対応します。トルクの制限値は、モータの定格トルクに対するパーセンテージにて指定します。何も制限しない場合には、モータは１００％を定格トルクとして最大トルクまで出すことができます。各軸に対応する要素を省略した場合には、その軸のトルクの制限値は０と見なします。
１００未満を指定すると脱調、モータロックエラーの検出レベルが下ります。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
トルクの制限値として、０以下を指定すると０と見なします。
トルクの制限値が低すぎると、ロボットが動作できずにエラーになる事があります。

サンプルプログラム

```
PROGRAM  TORQUESMPL
MOVE A1
TORQUE={100,100,100,100,100}
MOVE A2
OPEN1
DELAY 0.5
MOVE A1
TORQUE={200,200,200,200,200}
READY
END
```

Ａ２への動作時にＺ軸（この場合第３軸）のトルクを、定格トルクの１００％に制限します。

ＴＰ

機能

ＰＲＩＮＴ，ＩＮＰＵＴ命令で使う通信チャネルをティーチペンダントに指定します。

フォーマット

```
ＰＲＩＮＴ［｛ＣＯＭ０｜ＣＯＭ１｜ＴＰ｝，］
    ｛＜文字列＞｜＜式＞｝［，｛＜文字列＞｜＜式＞｝］
                                        ．．．［，ＣＲ］
ＩＮＰＵＴ［｛ＣＯＭ０｜ＣＯＭ１｜ＴＰ｝，］
   ＜変数＞［，＜変数＞］．．．
```

文例

```
ＰＲＩＮＴ　ＴＰ，”＊＊＊　ＩＮＰＵＴ　Ｎ　＊＊＊”，ＣＲ
ＰＲＩＮＴ　ＴＰ，Ｎ，Ｎ＊１０，ＣＲ
ＩＮＰＵＴ　ＴＰ，Ｋ
```

解説・注意

ＰＲＩＮＴ,ＩＮＰＵＴ命令にて通信チャネルをティーチペンダントに指定するために使用します。
通信チャネルでＴＰを指定すると、ティーチペンダントの通信チャネル（ＣＯＭ０）に対してデータの入出力を行います。
ＰＲＩＮＴ,ＩＮＰＵＴ命令にて通信チャネルを指定しない場合にもティーチペンダント専用の通信チャネルに対してデータの入出力を行います。
通信処理についてはＰＲＩＮＴ，ＩＮＰＵＴの各命令を参照してください。

サンプルプログラム

```
PROGRAM  TPSAMPLE
PRINT TP,"*** INPUT N ***",CR      ティーチペンダントから入力した数値を、
INPUT TP,N                         ティーチペンダントに出力します。
PRINT TP,N,CR
END
```

ＴＲＡＮＳ

機能

座標型データを生成します。

フォーマット

ＴＲＡＮＳ（＜式＞，＜式＞，＜式＞，＜式＞）

文例

Ａ＝ＴＲＡＮＳ（１００，１００，０，０）
ＷＯＲＫ＝ＷＯＲＫ＋ＴＲＡＮＳ（１００，１００）

解説・注意

座標型データを生成します。
＜式＞は、左から順番に座標型データのＸ，Ｙ，Ｚ，Ｃ要素に対応します。Ｘ，Ｙ，Ｚはミリメートル，Ｃは度単位で指定します。
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。また、＜式＞の記述は省略する事もできますが、省略した要素は０と見なされます。
本命令は、式の中で使用する事ができます。

サンプルプログラム

```
PROGRAM  TRANSSMPL
MOVE A1
MOVE A2
WORK=TRANS(,,300)                ワーク座標系の値をＺ＝３００
MOVE A1 WITH WORK=WORK1          Ｘ，Ｙ，Ａ，Ｂ，Ｃ＝０に設定します。
MOVE A2 WITH WORK=WORK1
END
```

ＷＡＩＴ

機能

条件が成立するまで待ちます。

フォーマット

ＷＡＩＴ＜論理式＞

文例

ＷＡＩＴ　ＤＩＮ（１）
ＷＡＩＴ　ＴＩＭＥＲ＝＝０
ＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００

解説・注意

＜論理式＞の条件が成立するまで、プログラムの実行を待ちます。
条件の監視は、ロボットの動作中いかんにかかわらず行います。

サンプルプログラム

```
PROGRAM  WAITSAMPLE
WAIT DIN(1)                    入力信号の１がオンになるまで待ちます。
MOVE A1
MOVE A2
MOVE A3
END
```

ＷＩＴＨ

機能

個々の動作命令に動作条件を指定します。

フォーマット

ＷＩＴＨ＜文＞［，＜文＞］・・・

文例

ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＳＰＥＥＤ＝５０
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＴＯＯＬ＝ＴＯＯＬ１，ＰＡＳＳ＝８０

解説・注意

個々の動作命令に対して動作条件を指定します。
ロボットが動作を行う時には、その時点でのシステム変数の設定値により、速度、加減速等の動作条件が決まります。１つの動作だけに関して、動作条件を変更するためには、ＷＩＴＨ節を用いて動作条件を指定します。ＷＩＴＨ節にて指定した動作条件は、ＷＩＴＨ節の指定された動作命令でのみ有効になります。動作命令実行の前後では、システム変数の値は変化しません。
＜文＞にて指定できる動作条件は、以下のものです。これらの動作条件は、複数の条件を”，”で区切って指定する事ができます。

動　作　条　件	システム変数名
ロボットの姿勢	ＣＯＮＦＩＧ
位置決め精度	ＡＣＣＵＲ
加速時の加速度	ＡＣＣＥＬ
減速時の加速度	ＤＥＣＥＬ
速度	ＳＰＥＥＤ
ショートカット動作のパラメータ	ＰＡＳＳ
各軸最大トルク	ＴＯＲＱＵＥ
各軸サーボゲイン	ＧＡＩＮ
ツール座標系	ＴＯＯＬ
ベース座標系	ＢＡＳＥ
ワーク座標系	ＷＯＲＫ
ロボットの負荷	ＰＡＹＬＯＡＤ

サンプル
プログラム

```
PROGRAM  WITHSAMPLE
SPEED=50
MOVE A1
MOVE A2 WITH SPEED=100
MOVE A3
MOVE A4
END
```

Ａ２への移動のみ１００％の速度で動作し、その他の移動は５０％の速度で動作します。

ＷＯＲＫ

機能

ワーク座標系を指定します。

フォーマット

ＷＯＲＫ

文例

```
ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＷＯＲＫ＝ＷＯＲＫ＋
                                ＴＲＡＮＳ（，，１００）
ＷＯＲＫ　ＷＫ１
```

解説・注意

ワーク座標系を指定するシステム変数です。
ワーク座標系は、ワールド座標系のそれぞれの座標軸に沿って、Ｘ，Ｙ，Ｚの量だけ平行移動した座標軸を、新しいＺ軸の回りにＣだけ回転した座標系になります。
本命令は式の中で使用します。
ティーチペンダントから位置データを教示すると、教示した時に指定したワーク座標系のデータも同時に記憶します。動作命令の実行時には、ワーク座標系はその位置データを教示した時のものに自動的に変わります。教示時に指定したワーク座標系と、異なる座標系でロボットを動作したい時には、ＷＩＴＨ節を用いてワーク座標系を指定するか、ワーク座標系のデータそのものを変更するようにしてください。
プログラム中でワーク座標系を変更すると、以降の動作で教示した位置がずれる事がありますので注意してください。

サンプル
プログラム

```
PROGRAM  WORKSAMPLE
MOVE A1
WK1=WORK+TRANS(,,300)
MOVE A1 WITH WORK=WK1
END

PROGRAM  WORKSAMPLE2
MOVE A2
WK1=WORK+TRANS(,,300)
MOVE A2
END
DATA
TRANS WK1=0,0,0,0
WORK WK1
POINT A2=500,0,100,0,0/FREE
END
```

ＷＯＲＫ命令によりワーク座標系のＺ軸を３００ｍｍずらして、以降の動作では教示した位置より３００ｍｍ上方の点に動作します。

ＣＵＲＯＶＲＤ

機能	現在設定されているオーバーライド値を参照します。
フォーマット	ＣＵＲＯＶＲＤ
文例	ＩＦ　ＣＵＲＯＶＲＤ＝＝１０　ＴＨＥＮ　ＯＶＥＲＲＩＤＥ　＝　２０
解説・注意	本システム変数を参照すると、現在設定されているオーバーライド値が参照できます。 本システム変数は参照のみで、値を代入することはできません。
サンプル プログラム	PROGRAM　CUROVRDSAMPLE 　IF CUROVRD <> 20 THEN OVERRIDE = 20 　MOVEA 1,90 END 現在設定されているオーバーライド値が２０ではないとき、オーバーライド値を ２０に設定して、第１軸を９０度の位置へ動かします。

ＩＮＰＳＴＳＴＰ
ＩＮＰＳＴＳＩＰ１
ＩＮＰＳＴＳＩＰ２
ＩＮＰＳＴＳＣＭ１
ＩＮＰＳＴＳＣＭ２

機能	各通信チャンネルにおけるＩＮＰＵＴ命令のタイムアウト監視結果を参照します。
フォーマット	ＩＮＰＳＴＳＴＰ ＩＮＰＳＴＳＩＰ１ ＩＮＰＳＴＳＩＰ２ ＩＮＰＳＴＳＣＭ１ ＩＮＰＳＴＳＣＭ２
文例	ＩＦ　ＩＮＰＳＴＳＴＰ＝＝１　ＴＨＥＮ　ＧＯＴＯ　ＥＲＲ
解説・注意	ＩＮＰＵＴ命令のタイムアウト監視結果を参照します。 ０：ＩＮＰＵＴ命令が正常に実行されました。 １：ＩＮＰＵＴ命令はタイムアウトで実行されました。

注）　タイムアウト監視結果は、ＩＮＰＵＴ命令に対するタイムアウトを設定していない場合、常に０になります。
タイムアウト監視結果を参照する場合は、ＩＮＰＵＴ命令に対するタイムアウト時間を設定してください。

ＩＮＰＵＴ命令の通信チャンネルに対応するタイムアウト監視結果言語は以下の通りになります。

通信チャンネル	タイムアウト時間言語	タイムアウト監視結果言語
ＴＰ	ＩＮＰＴＭＯＴＰ	ＩＮＰＳＴＳＴＰ
ＩＰ１	ＩＮＰＴＭＯＩＰ１	ＩＮＰＳＴＳＩＰ１
ＩＰ２	ＩＮＰＴＭＯＩＰ２	ＩＮＰＳＴＳＩＰ２
ＣＯＭ１	ＩＮＰＴＭＯＣＭ１	ＩＮＰＳＴＳＣＭ１
ＣＯＭ２	ＩＮＰＴＭＯＣＭ２	ＩＮＰＳＴＳＣＭ２

サンプル
プログラム

```
PROGRAM  INPTMOTPSAMPLE
 X1=0.0
 INPTMOIP1=0.1
 INPUT IP1,X1
 IF INPSTSIP1==1 THEN PRINT “NG”,CR ELSE PRINT “OK”,CR
END
```

INPUTタイムアウト時間を0.1秒に設定し、INPUT命令実行後、INPUTステータス情報が1の場合は、タイムアウトなのでティーチングペンダントにNGを表示。INPUTステータス情報が1以外の場合は、成功なのでティーチングペンダントにOKを表示する。

ＩＮＰＴＭＯＴＰ
ＩＮＰＴＭＯＩＰ１
ＩＮＰＴＭＯＩＰ２
ＩＮＰＴＭＯＣＭ１
ＩＮＰＴＭＯＣＭ２

機能

各通信チャンネルにおけるＩＮＰＵＴ命令に対するタイムアウト時間を秒単位で設定します。

フォーマット

ＩＮＰＴＭＯＴＰ
ＩＮＰＴＭＯＩＰ１
ＩＮＰＴＭＯＩＰ２
ＩＮＰＴＭＯＣＭ１
ＩＮＰＴＭＯＣＭ２

文例

ＩＮＰＴＭＯＴＰ＝１０．０
ＩＮＰＴＭＯＩＰ１＝０．０１

解説・注意

ＩＮＰＵＴ命令のタイムアウト時間を秒単位の値で設定します。
ＩＮＰＵＴ，ＴＰの場合、タイムアウト時間はＩＮＰＴＭＯＴＰに設定します。
ＩＮＰＵＴ命令実行後、設定したタイムアウト時間が経過するまでに通信チャンネルからデータが読み込めない場合、ＩＮＰＵＴ命令で指定した変数に０を代入し、次のステップに進みます。
ＩＮＰＵＴ命令で指定した通信チャンネルと、タイムアウトの通信チャンネルが一致しない場合、タイムアウト時間は監視されません。

注）０．００１以下の値を設定するとタイムアウト時間は監視されません。

ＩＮＰＵＴ命令の通信チャンネルに対応するタイムアウト時間言語は以下の通りになります。

通信チャンネル	タイムアウト時間言語	タイムアウト監視結果言語
ＴＰ	ＩＮＰＴＭＯＴＰ	ＩＮＰＳＴＳＴＰ
ＩＰ１	ＩＮＰＴＭＯＩＰ１	ＩＮＰＳＴＳＩＰ１
ＩＰ２	ＩＮＰＴＭＯＩＰ２	ＩＮＰＳＴＳＩＰ２
ＣＯＭ１	ＩＮＰＴＭＯＣＭ１	ＩＮＰＳＴＳＣＭ１
ＣＯＭ２	ＩＮＰＴＭＯＣＭ２	ＩＮＰＳＴＳＣＭ２

サンプル
プログラム

```
PROGRAM  INPTMOTPSAMPLE
 X1=0.0
 INPTMOIP1=0.1
 INPUT IP1,X1
 IF INPSTSIP1==1 THEN PRINT “NG”,CR ELSE PRINT “OK”,CR
END
```

INPUTタイムアウト時間を0.1秒に設定し、INPUT命令実行後、INPUTステータス情報が1の場合は、タイムアウトなのでティーチングペンダントにNGを表示。INPUTステータス情報が1以外の場合は、成功なのでティーチングペンダントにOKを表示する。

ＩＰＣＬＯＳＥ

機能

各通信チャンネルのクローズ処理を実行します。

フォーマット

ＩＰＣＬＯＳＥ(＜整数＞)

文例

ＩＰＣＬＯＳＥ(０)
ＩＰＣＬＯＳＥ(１)
ＩＰＣＬＯＳＥ(２)
ＩＰＣＬＯＳＥ(３)

解説・注意

各通信チャンネルのクローズ処理を実行します。
各通信チャンネルは以下の整数で指定します。
０：ＴＣＰ／ＩＰ０
１：ＴＣＰ／ＩＰ１
２：ＴＣＰ／ＩＰ２
３：ＴＣＰ／ＩＰ３
本命令ではシステム変数のＩＰ１、ＩＰ２は使用できません。
ＩＰＣＬＯＳＥ命令を実行する場合、通信チャンネルの状態がＣＬＯＳＥＤ状態以外のときに実行するようにしてください。
例）ＩＰＯＰＥＮ（１） を実行する場合、
ＩＰ１ＳＴＡＴＵＳ ＞ １（クローズ状態以外）を確認してください。
※ＩＰＣＬＯＳＥ実行後通信チャンネルの状態がＣＬＯＳＥＤ状態になっていないときは、断線などの場合がありますのでケーブルを確認してください。

サンプル
プログラム

```
PROGRAM  IPCLOSESAMPLE
 START:
  X1=0.0
  IF IP1STATUS>1 THEN IPCLOSE(1)
  IF IP1STATUS<>1 THEN GOTO START
  IF IP1STATUS==1 THEN IPOPEN(1)
  IF IP1STATUS<>2 AND IP1STATUS<>3 THEN GOTO START
 INP:
  IF IP1STATUS==5 THEN INPUT IP1,X1 ELSE GOTO INP
END
```

IP1の状態がnon、Closed状態でないときにIP1をCLOSEします。
IPCLOSE(1)実行後、IP1の状態がClosed状態でなければSTARTラベルに推移します。
IP1の状態がClosed状態のときIP1をOPENします。
IPOPEN(1)実行後、IP1の状態がListenまたはSyn_Sent状態でなければSTARTラベルに推移します。
IP1の状態がEstablished状態のときINPUT IP1を実行します。
IP1の状態がEstablished状態以外のときはINPラベルに推移します。
※各通信チャンネルの状態を参照するには、IP0STATUS、IP1STATUS、IP2STATUS、IP3STATUSをご覧ください。

ＩＰＯＰＥＮ

機能

各通信チャンネルのオープン処理を実行します。

フォーマット

ＩＰＯＰＥＮ(＜整数＞)

文例

ＩＰＯＰＥＮ(０)
ＩＰＯＰＥＮ(１)
ＩＰＯＰＥＮ(２)
ＩＰＯＰＥＮ(３)

解説・注意

各通信チャンネルのオープン処理を実行します。
各通信チャンネルは以下の整数で指定します。
０：ＴＣＰ／ＩＰ０
１：ＴＣＰ／ＩＰ１
２：ＴＣＰ／ＩＰ２
３：ＴＣＰ／ＩＰ３
本命令ではシステム変数のＩＰ１、ＩＰ２は使用できません。
ＩＰＯＰＥＮ命令を実行する場合、通信チャンネルの状態がＣＬＯＳＥＤ状態のときに実行するようにしてください。
例）ＩＰＯＰＥＮ（１） を実行する場合、
　　ＩＰ１ＳＴＡＴＵＳ＝＝１（クローズ状態）を確認してください。
※ＩＰＯＰＥＮ実行後通信チャンネルの状態がＬｉｓｔｅｎまたは
Ｓｙｎ＿Ｓｅｎｔ状態になっていないときは、断線などの場合がありますのでケーブルを確認してください。

サンプル
プログラム

```
PROGRAM  IPOPENSAMPLE
 START:
   X1=0.0
   IF IP1STATUS>1 THEN IPCLOSE(1)
   IF IP1STATUS<>1 THEN GOTO START
   IF IP1STATUS==1 THEN IPOPEN(1)
   IF IP1STATUS<>2 AND IP1STATUS<>3 THEN GOTO START
  INP:
   IF IP1STATUS==5 THEN INPUT IP1,X1 ELSE GOTO INP
END
```

IP1の状態がnon、Closed状態でないときにIP1をCLOSEします。
IPCLOSE(1)実行後、IP1の状態がClosed状態でなければSTARTラベルに推移します。
IP1の状態がClosed状態のときIP1をOPENします。
IPOPEN(1)実行後、IP1の状態がListenまたはSyn_Sent状態でなければSTARTラベルに推移します。
IP1の状態がEstablished状態のときINPUT IP1を実行します。
IP1の状態がEstablished状態以外のときはINPラベルに推移します。
※各通信チャンネルの状態を参照するには、IP0STATUS、IP1STATUS、IP2STATUS、IP3STATUSをご覧ください。

ＩＰ０ＳＴＡＴＵＳ
ＩＰ１ＳＴＡＴＵＳ
ＩＰ２ＳＴＡＴＵＳ
ＩＰ３ＳＴＡＴＵＳ

機能

各通信チャンネルにおける通信状態を参照します。

フォーマット

ＩＰ０ＳＴＡＴＵＳ
ＩＰ１ＳＴＡＴＵＳ
ＩＰ２ＳＴＡＴＵＳ
ＩＰ３ＳＴＡＴＵＳ

文例

ＩＦ　ＩＰ０ＳＴＡＴＵＳ＝＝０　ＴＨＥＮ　ＧＯＴＯ　ＥＲＲ

解説・注意

各通信チャンネルにおける通信状態を参照します。

０：ｎｏｎ	６：Ｆｉｎ　Ｗａｉｔ１
１：Ｃｌｏｓｅｄ	７：Ｆｉｎ　Ｗａｉｔ２
２：Ｌｉｓｔｅｎ	８：Ｃｌｏｓｅ　ｗａｉｔ
３：Ｓｙｎ　Ｓｅｎｔ	９：Ｃｌｏｓｉｎｇ
４：Ｓｙｎ　Ｒｅｃｉｖｅｄ	１０：Ｌａｓｔ　ＡＣＫ
５：Ｅｓｔａｂｌｉｓｈｅｄ	１１：Ｔｉｍｅ　Ｗａｉｔ

サンプルプログラム

```
PROGRAM  IPSTATUSSAMPLE
 STRT:
   X1=0.0
   IF IP1STATUS<>5 THEN GOTO ERR
   INPUT IP1,X1
   GOTO STRT
 ERR:
   PRINT “IP1 NG”,CR
END
```

IP1の通信状態がEstablishedのとき、INPUT命令を実行する。
Established以外のときはERRにジャンプしティーチングペンダントに“IP1 NG”表示する

ＯＶＥＲＲＩＤＥ

機能

ロボットのオーバライドを指定します。

フォーマット

ＯＶＥＲＲＩＤＥ＝＜式＞

文例

ＯＶＥＲＲＩＤＥ＝５０

解説・注意

ロボットの動作速度に対するオーバライドを指定するためのシステム変数です。
ＯＶＥＲＲＩＤＥは、正の整数値で指定します。０以下の数値を指定すると、１が指定されたものとして処理されます。また１００以上の値を指定しても１００％として処理します。
ＴＰ操作によってオーバライド（ＯＶＲＤ）が設定されている場合には、その設定値より大きな値を設定することはできません（命令は無視されます。）
＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。
ＯＶＥＲＲＩＤＥの初期値は１００％になっています。

サンプルプログラム

```
PROGRAM SAMPLE
SPEED=100
MOVE A1
ON MOTION>=25 DO OVERRIDE=50
MOVE A2
WAIT MOTION>=100
OVERRIDE=100
ON MOTION>=50 DO OVERRIDE=50
MOVE A3
WAIT MOTION>=100
OVERRIDE=100
ON MOTION>=75 DO OVERRIDE=50
MOVE A1
WAIT MOTION>=100
MOVE A2
END
```

ＰＬＣＳＬＲ０１～０８

機能

ロボット内蔵の簡易ＰＬＣからデータを受け取るシステム変数です。

フォーマット

ＰＬＣＳＬＲ０１

文例

Ａ＝ＰＬＣＳＬＲ０１
Ｂ＝ＰＬＣＳＬＲ０５

解説・注意

ＰＬＣＳＬＲ０１～０８は読み込み専用のシステム変数です。
簡易ＰＬＣでセットした値を読むことができます。
（簡易ＰＬＣはオプション機能です。）
受け取る値は　－２１４７４８３６４８～２１４７４８３６４７となります。

SCOL言語名		データレジスタ
PLCSLR01	L	D410
	H	D411
PLCSLR02	L	D412
	H	D413
PLCSLR03	L	D414
	H	D415
PLCSLR04	L	D416
	H	D417
PLCSLR05	L	D418
	H	D419
PLCSLR06	L	D41A
	H	D41B
PLCSLR07	L	D41C
	H	D41D
PLCSLR08	L	D41E
	H	D41F

例）

-10
（10 進数）

TCmini レジスタ

D411	D410
PLCSLR01H	PLCSLR01L
0xFFFF	0xFFF6
10 進数（-10）	

ロボットプログラム

A=PLCSLR01
結果 A=-10

サンプルプログラム

```
PROGRAM  PLCSLR
  A=PLCSLR01
  B=PLCSLR02
END
```

ＰＬＣＳＬＷ０１～０８

機能

ロボット内蔵の簡易ＰＬＣに値を書き込むシステム変数です。

フォーマット

ＰＬＣＳＬＷ０１＝＜式＞

文例

ＰＬＣＳＬＷ０１＝１
ＰＬＣＳＬＷ０５＝Ａ＋Ｂ

解説・注意

ＰＬＣＳＬＷ０１～０８は書込み専用のシステム変数です。
簡易ＰＬＣに値を受け渡すことができます。
（簡易ＰＬＣはオプション機能です。）
受け渡す値は　－２１４７４８３６４８～２１４７４８３６４７となります。
簡易ＰＬＣ機能オプションが選択されている場合には、
ＰＬＣＳＬＷ０１～０８の値をシーケンスプログラムで自由に
扱うことができます。

SCOL言語名		データレジスタ
PLCSLW01	L	D610
	H	D611
PLCSLW02	L	D612
	H	D613
PLCSLW03	L	D614
	H	D615
PLCSLW04	L	D616
	H	D617
PLCSLW05	L	D618
	H	D619
PLCSLW06	L	D61A
	H	D61B
PLCSLW07	L	D61C
	H	D61D
PLCSLW08	L	D61E
	H	D61F

例)

ロボットプログラム

PLCSLW01=70000

TCmini レジスタ

D611	D610
PLCSLW01H	PLCSLW01L
0x0001	0x1170
10 進数（70000）	

サンプルプログラム

```
PROGRAM  PLCSLW
  A=10
  PLCSLW01=1
  PLCSLW02=A
END
```

ＰＬＣＳＳＲ０１～０８

機能

ロボット内蔵の簡易ＰＬＣからデータを受け取るシステム変数です。

フォーマット

ＰＬＣＳＳＲ０１

文例

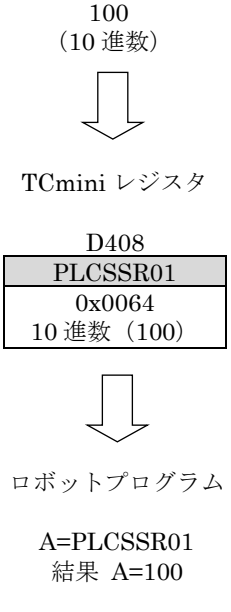
Ａ＝ＰＬＣＳＳＲ０１
Ｂ＝ＰＬＣＳＳＲ０５

解説・注意

ＰＬＣＳＳＲ０１～０８は読み込み専用のシステム変数です。
簡易ＰＬＣでセットした値を読むことができます。
（簡易ＰＬＣはオプション機能です。）
受け取る値は　－３２７６８～３２７６７となります。

SCOL言語名	データレジスタ
PLCSSR01	D408
PLCSSR02	D409
PLCSSR03	D40A
PLCSSR04	D40B
PLCSSR05	D40C
PLCSSR06	D40D
PLCSSR07	D40E
PLCSSR08	D40F

例）



サンプルプログラム

```
PROGRAM  PLCSSR
   A=PLCSSR01
   B=PLCSSR02
END
```

ＰＬＣＳＳＷ０１～０８

機能

ロボット内蔵の簡易ＰＬＣに値を書き込むシステム変数です。

フォーマット

ＰＬＣＳＳＷ０１＝＜式＞

文例

ＰＬＣＳＳＷ０１＝１
ＰＬＣＳＳＷ０５＝Ａ＋Ｂ

解説・注意

ＰＬＣＳＳＷ０１～０８は書込み専用のシステム変数です。
簡易ＰＬＣに値を受け渡すことができます。
（簡易ＰＬＣはオプション機能です。）
受け渡す値は　－３２７６８～３２７６７となります。
簡易ＰＬＣ機能オプションが選択されている場合には、
ＰＬＣＳＳＷ０１～０８の値をシーケンスプログラムで自由に
扱うことができます。

SCOL言語名	データレジスタ
PLCSSW01	D608
PLCSSW02	D609
PLCSSW03	D60A
PLCSSW04	D60B
PLCSSW05	D60C
PLCSSW06	D60D
PLCSSW07	D60E
PLCSSW08	D60F

例）

ロボットプログラム

PLCSSW01=-10

TCmini レジスタ

D608

PLCSSW01
0xFFF6 10 進数（-10）

サンプルプログラム

```
PROGRAM  PLCSSW
  A=10
  PLCSSW01=1
  PLCSSW02=A
END
```

ＰＳＮＣＭＤ

機能

位置指令値に基づくワールド座標系現在位置を取り出します。

フォーマット

ＰＳＮＣＭＤ

文例

Ａ１＝ＰＳＮＣＭＤ
Ｘ＝ＰＳＮＣＭＤ．Ｘ

解説・注意

ワールド座標系での現在位置を取出します。
ＰＳＮＣＭＤは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが動作中にＰＳＮＣＭＤ命令が実行された場合は、ＰＳＮＣＭＤ命令時点でのロボットへの指令位置が現在位置として取り出されます。

注）ＰＳＮＣＭＤ命令で取出される位置は、コンベア同期機能などの補正値を含んだロボットへの指令位置になります。ロボットの動作中は、ロボットの実際の現在位置は指令位置から遅れている事に注意してください。
現在のロボットの姿勢は、ＰＳＮＣＭＤ命令によって取得できます。
Ｎ＝ＰＳＮＣＭＤ．６
等でＮに現在の姿勢の値が入ります。
スカラロボットではＰＳＮＣＭＤ．６の値が０で未定義、１で左肩系、２で右肩系となります。

サンプルプログラム

```
PROGRAM  PSNCMDSAMPLE
AA=A                          Ａ１からＡ４まで移動中に入力信号１を監
MOVE A                        視して、信号がオンした時の位置指令値に基
ON DIN(1) DO AA=PSNCMD        づくワールド座標系現在位置を、ティーチペ
MOVE A1                       ンダントに表示します。
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.X,AA.Y,AA.Z
END
```

ＰＳＮＣＭＤＪ

機能

位置指令値に基づく関節軸現在位置を取り出します。

フォーマット

ＰＳＮＣＭＤＪ

文例

Ａ１＝ＰＳＮＣＭＤＪ
Ｘ＝ＰＳＮＣＭＤＪ．１

解説・注意

関節軸での現在位置を取出します。
ＰＳＮＣＭＤＪは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが動作中にＰＳＮＣＭＤＪ命令が実行された場合は、ＰＳＮＣＭＤＪ命令時点でのロボットへの指令位置が現在位置として取り出されます。

注）　ＰＳＮＣＭＤＪ命令で取出される位置は、コンベア同期機能などの補正値を含んだロボットへの指令位置になります。ロボットの動作中は、ロボットの実際の現在位置は指令位置から遅れている事に注意してください。

サンプルプログラム

```
PROGRAM  PSNCMDJSAMPLE
AA=A                          Ａ１からＡ４まで移動中に入力信号１を監
MOVE A                        視して、信号がオンした時の位置指令値に基
ON DIN(1) DO AA=PSNCMDJ       づく関節軸現在位置を、ティーチペンダト
MOVE A1                       ンに表示します。
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.1,AA.2,AA.3
END
```

ＰＳＮＣＭＤＷ

機能

位置指令値に基づくワーク座標系現在位置を取り出します。

フォーマット

ＰＳＮＣＭＤＷ

文例

Ａ１＝ＰＳＮＣＭＤＷ
Ｘ＝ＰＳＮＣＭＤＷ．Ｘ

解説・注意

ワーク座標系での現在位置を取出します。
ＰＳＮＣＭＤＷは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが動作中にＰＳＮＣＭＤＷ命令が実行された場合は、ＰＳＮＣＭＤＷ命令時点でのロボットへの指令位置が現在位置として取り出されます。

注）ＰＳＮＣＭＤＷ命令で取出される位置は、コンベア同期機能などの補正値を含んだロボットへの指令位置になります。ロボットの動作中は、ロボットの実際の現在位置は指令位置から遅れている事に注意してください。
現在のロボットの姿勢は、ＰＳＮＣＭＤＷ命令によって取得できます。
Ｎ＝ＰＳＮＣＭＤＷ．６
等でＮに現在の姿勢の値が入ります。
スカラロボットではＰＳＮＣＭＤＷ．６の値が０で未定義、１で左肩系、２で右肩系となります。

サンプルプログラム

```
PROGRAM  PSNCMDWSAMPLE
AA=A                          Ａ１からＡ４まで移動中に入力信号１を監
MOVE A                        視して、信号がオンした時の位置指令値に基
ON DIN(1) DO AA=PSNCMDW       づくワーク座標系現在位置を、ティーチペン
MOVE A1                       ダントに表示します。
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.X,AA.Y,AA.Z
END
```

ＰＳＮＦＢＫ

機能

位置フィードバックに基づくワールド座標系現在位置を取り出します。

フォーマット

ＰＳＮＦＢＫ

文例

Ａ１＝ＰＳＮＦＢＫ
Ｘ＝ＰＳＮＦＢＫ．Ｘ

解説・注意

ワールド座標系での現在位置を取出します。
ＰＳＮＦＢＫは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが動作中にＰＳＮＦＢＫ命令が実行された場合は、ＰＳＮＦＢＫ命令時点でのロボットへの指令位置が現在位置として取り出されます。

注）　ＰＳＮＦＢＫ命令で取出される位置は、ロボットへの指令位置になります。ロボットの動作中は、ロボットの実際の現在位置は指令位置から遅れている事に注意してください。
現在のロボットの姿勢は、ＰＳＮＦＢＫ命令によって取得できます。
　Ｎ＝ＰＳＮＦＢＫ．６
等でＮに現在の姿勢の値が入ります。
スカラロボットではＰＳＮＦＢＫ．６の値が０で未定義、１で左肩系、２で右肩系となります。

サンプルプログラム

```
PROGRAM  PSNFBKSAMPLE
AA=A                          Ａ１からＡ４まで移動中に入力信号１を監
MOVE A                        視して、信号がオンした時の位置フィードバ
ON DIN(1) DO AA=PSNFBK        ックに基づくワールド座標系現在位置を、
MOVE A1                       ティーチペンダントに表示します。
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.X,AA.Y,AA.Z
END
```

ＰＳＮＦＢＫＪ

機能

位置フィードバックに基づく関節軸現在位置を取り出します。

フォーマット

ＰＳＮＦＢＫＪ

文例

Ａ１＝ＰＳＮＦＢＫＪ
Ｘ＝ＰＳＮＦＢＫＪ．１

解説・注意

関節軸での現在位置を取出します。
ＰＳＮＦＢＫＪは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが動作中にＰＳＮＦＢＫＪ命令が実行された場合は、ＰＳＮＦＢＫＪ命令時点でのロボットへの指令位置が現在位置として取り出されます。

注）　ＰＳＮＦＢＫＪ命令で取出される位置は、ロボットへの指令位置になります。ロボットの動作中は、ロボットの実際の現在位置は指令位置から遅れている事に注意してください。

サンプルプログラム

```
PROGRAM  PSNFBKJSAMPLE
AA=A                        Ａ１からＡ４まで移動中に入力信号１を監
MOVE A                      視して、信号がオンした時の位置フィードバ
ON DIN(1) DO AA=PSNFBKJ     ックに基づく関節軸現在位置を、ティーチペ
MOVE A1                     ンダントに表示します。
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.1,AA.2,AA.3
END
```

ＰＳＮＦＢＫＷ

機能

位置フィードバックに基づくワーク座標系現在位置を取り出します。

フォーマット

ＰＳＮＦＢＫＷ

文例

Ａ１＝ＰＳＮＦＢＫＷ
Ｘ＝ＰＳＮＦＢＫＷ．Ｘ

解説・注意

ワーク座標系での現在位置を取出します。
ＰＳＮＦＢＫＷは、通常の位置型データと同様に使用する事ができますが、参照のみ可能で、代入は行なえません。
ロボットが動作中にＰＳＮＦＢＫＷ命令が実行された場合は、ＰＳＮＦＢＫＷ命令時点でのロボットへの指令位置が現在位置として取り出されます。

注）　ＰＳＮＦＢＫＷ命令で取出される位置は、ロボットへの指令位置になります。ロボットの動作中は、ロボットの実際の現在位置は指令位置から遅れている事に注意してください。
現在のロボットの姿勢は、ＰＳＮＦＢＫＷ命令によって取得できます。
　Ｎ＝ＰＳＮＦＢＫＷ．６
等でＮに現在の姿勢の値が入ります。
スカラロボットではＰＳＮＦＢＫＷ．６の値が０で未定義、１で左肩系、２で右肩系となります。

サンプルプログラム

```
PROGRAM  PSNFBKWSAMPLE
AA=A                          Ａ１からＡ４まで移動中に入力信号１を監
MOVE A                        視して、信号がオンした時の位置フィードバ
ON DIN(1) DO AA=PSNFBKW       ックに基づくワーク座標系現在位置を、
MOVE A1                      ティーチペンダントに表示します。
MOVE A2
MOVE A3
MOVE A4
IGNORE DIN(1)
PRINT "POSITION DATA=",AA.X,AA.Y,AA.Z
END
```

ＳＡＶＥＦ１～４

機能

主電源断時に、クリアされずに保持できる実数型のシステム変数です。

フォーマット

ＳＡＶＥＦ１　＝　＜式＞
ＳＡＶＥＦ２　＝　＜式＞
ＳＡＶＥＦ３　＝　＜式＞
ＳＡＶＥＦ４　＝　＜式＞

文例

ＳＡＶＥＦ　＝　１．０
Ａ　＝　ＳＡＶＥＦ１

解説・注意

・ **主電源断時に**指定された**実数型**のシステム変数がバックアップメモリへ保存されます。
・ 実数型変数はＳＡＶＥＦ１～４の4つを使用することができます。
・ 次回電源ＯＮ時に、バックアップメモリに保存された値が復元されます。
・ 実数型で扱える値は数値の絶対値が約５．８７×１０$^{(-39)}$（２$^{-127}$）～６．８０×１０^{38}（（２^{23}－１）×２^{106}）の範囲とします。
・ 実数型のシステム変数の値は、以下の操作タイミングでクリアされます。
① プログラムセレクト
② プログラムリセット

サンプルプログラム

```
PROGRAM SAVEFSAMPLE
FOR A=1 TO 100
PRINT “SAVEF1=”, SAVEF1,CR
DELAY 0.5
SAVEF1= SAVEF1+1.0
NEXT A
END
```

ＳＡＶＥＩ１～４

機能

主電源断時に、クリアされずに保持できる整数型のシステム変数です。

フォーマット

ＳＡＶＥＩ１　＝　＜式＞
ＳＡＶＥＩ２　＝　＜式＞
ＳＡＶＥＩ３　＝　＜式＞
ＳＡＶＥＩ４　＝　＜式＞

文例

ＳＡＶＥＩ１　＝　１
Ａ　＝　ＳＡＶＥＩ１

解説・注意

・ **主電源断時に**指定された**整数型**のシステム変数がバックアップメモリへ保存されます。
・ 整数型変数はＳＡＶＥＩ１～４の4つを使用することができます。
・ 次回電源ＯＮ時に、バックアップメモリに保存された値が復元されます。
・ 整数型で扱える値は-２１４７４８３６４８～＋２１４７４８３６４７の範囲とします。
・ 整数型のシステム変数の値は、以下の操作タイミングでクリアされます。
① プログラムセレクト
② プログラムリセット

サンプルプログラム

```
PROGRAM SAVEISAMPLE
FOR A= SAVEI1 TO 100
PRINT”A=”,A,CR
DELAY 0.5
SAVEI1 = A
NEXT A
END
```

ＳＬＯＷＤＯＷＮ

機能

スローダウン動作を指定します。

フォーマット

ＳＬＯＷＤＯＷＮ

文例

ＤＩＳＡＢＬＥ　ＳＬＯＷＤＯＷＮ
ＥＮＡＢＬＥ　ＳＬＯＷＤＯＷＮ

解説・注意

スローダウン動作を指定します。
スローダウン動作では、現在の動作の移動中に速度を変更（減速）することができます。
システムスイッチのオン、オフには、ＥＮＡＢＬＥ，ＤＩＳＡＢＬＥ命令を使用します。ＥＮＡＢＬＥ　ＳＬＯＷＤＯＷＮ以降の動作命令は指定パラメータにしたがって、移動途中から速度を変更します。
ＤＩＳＡＢＬＥ　ＳＬＯＷＤＯＷＮでスローダウン動作を解除します。
初期状態では、ＤＩＳＡＢＬＥ　ＳＬＯＷＤＯＷＮになっています。

サンプルプログラム

```
PROGRAM  SAMPLE
  MOVE A1
  SLOWDOWN=25
  SLWSPD=50
  ENABLE SLOWDOWN
  MOVE A2
  SLOWDOWN=50
  MOVE A3
  SLOWDOWN=75
  MOVE A1
  DISABLE SLOWDOWN
  MOVE A2
END
```

ＳＬＯＷＤＯＷＮ

機能

スローダウン動作のパラメータを設定します。

フォーマット

ＳＬＯＷＤＯＷＮ＝＜式＞

文例

ＳＬＯＷＤＯＷＮ＝８０

解説・注意

スローダウン動作のパラメータを指定するシステム定数です。
スローダウン動作では、現在の動作の移動中に速度を変更（減速）することができます。ＳＰＥＥＤ変数で指定された速度から減速開始するタイミングを、システム変数のＳＬＯＷＤＯＷＮにて指定します。スローダウン動作のパラメータは、動作に対するロボットの移動量のパーセンテージで指定します。ロボットは、移動量がシステム変数ＳＬＯＷＤＯＷＮにて指定されたパーセンテージを超えたとき、
ＳＬＷＳＰＤで指定された速度へ減速を開始します。

サンプルプログラム

```
PROGRAM  SAMPLE
  MOVE A1
  SLOWDOWN=25
  SLWSPD=50
  ENABLE SLOWDOWN
  MOVE A2
  SLOWDOWN=50
  MOVE A3
  SLOWDOWN=75
  MOVE A1
  DISABLE SLOWDOWN
  MOVE A2
END
```

ＳＬＷＳＰＤ

機能	スローダウン動作の低速側速度を指定します。
フォーマット	ＳＬＷＳＰＤ＝＜式＞
文例	ＳＬＷＳＰＤ＝２０
解説・注意	スローダウン動作の低速側速度を、最高速に対するパーセンテージで指定するためのシステム変数です。 ＳＬＷＳＰＤは、正の実数値で指定します。０以下の数値を指定すると、１が指定されたものとして処理されます。 なお、スローダウン動作で速度を上げることはできません。したがって、ＳＰＥＥＤ変数に設定された値よりも大きな値を指定した場合は無効になります。 ＜式＞には、定数、変数、及び演算式を使用する事ができます。ただし、ベクトル型のデータは使用できません。 ＳＬＷＳＰＤの初期値は１００％になっています。

サンプル
プログラム

```
PROGRAM SAMPLE
SLWSPD=50
SLOWDOWN=80
SLWSPD=30
MOVE A1
ENABLE PASS
ENABLE SLOWDOWN
MOVE A2
DISABLE PASS
DISABLE SLOWDOWN
MOVE A3
ENABLE SLOWDOWN
MOVE A4
DISABLE SLOWDOWN
END
```

第４章

プログラム例

本章では、ＳＣＯＬ言語を使用したプログラムの例を示します。ここに示すプログラムを実際のロボットに適応する場合には、そのロボットの可動範囲、周囲の状況に応じてプログラムを見直してください。

(1) 機械原点への移動プログラム

ロボットを機械原点(各関節軸の原点位置)まで移動するプログラムです。絶対単軸動作で、まずＺ軸（第３軸）から移動して、先端から根元の軸へと順々に動作します。

```
PROGRAM  ORIGIN
  MOVEA 3,0
  MOVEA 5,0
  MOVEA 4,0
  MOVEA 2,0
  MOVEA 1,0
END
```

(2) ロボットのウォーミングアッププログラム

実作業を始める前に、ウォーミングアップを行うプログラムです。各軸毎の動作で次第に速く動かします。

```
PROGRAM  WARMINGUP
  FOR K=1 TO 100
   SPEED=K
   MOVEA 3,100
   MOVEA 3,0
   MOVEA 5,50
   MOVEA 5,0
   MOVEA 4,50
   MOVEA 4,0
   MOVEA 2,50
   MOVEA 2,0
   MOVEA 1,50
   MOVEA 1,0
  NEXT K
END
```

(3) ロボットの動作

位置Ａ１，Ａ２の２点をロボットが移動するプログラムです。ロボットの動作速度は最高速度の２０％にしています。

(a) ＰＴＰ動作（ＭＯＶＥ命令）使用の場合

```
PROGRAM  MOVEA1A2
  SPEED=20
  MOVE A1
  MOVE A2
END
```

(b) 直線補間動作（ＭＯＶＥＳ命令）使用の場合

```
PROGRAM  MOVESA1A2
  SPEED=20
  MOVES A1
  MOVES A2
END
```

(4) 入出力信号

入力信号１～４がオンしたら対応する出力信号１～４をオンし､入力信号がオフしたら対応する出力信号をオフするプログラムです。

```
PROGRAM SAMPLE
  IF DIN(1) THEN DOUT(1) ELSE DOUT(-1)
  IF DIN(2) THEN DOUT(2) ELSE DOUT(-2)
  IF DIN(3) THEN DOUT(3) ELSE DOUT(-3)
  IF DIN(4) THEN DOUT(4) ELSE DOUT(-4)
END
```

(5) インタロック

Ａ１～Ａ４までの動作で､入力信号１がオフの間ロボットを停止するプログラムです。ロボットの動作速度は最高速度の２０％にしています。

(a) ＷＡＩＴ命令を使用した場合

```
PROGRAM  SAMPLE
  SPEED=20
  WAIT DIN(1)
  MOVE A1
  WAIT DIN(1)
  MOVE A2
```

```
  WAIT DIN(1)
  MOVE A3
  WAIT DIN(1)
  MOVE A4
END
```

(b) ＯＮ命令を使用して動作を中断し、入力信号１がオンになるまで待つ場合。

```
PROGRAM  SAMPLE
  SPEED=20
  ENABLE NOWAIT
  ON DIN(-1) BREAK DO SUB
  WAIT DIN(1)
  MOVE A1
  MOVE A2
  MOVE A3
  MOVE A4
END
PROGRAM SUB
  WAIT DIN(1)
  RESUME
  ON DIN(-1) BREAK DO SUB
END
```

(6) ピックアンドプレース

位置Ａ１からワークを持上げて位置Ａ２に置くプログラムです。ハンドの開閉は出力信号２０１を使用しています。ロボットの動作速度は最高速度の２０％にしています。

(a) Ａ１，Ａ２の位置の真上の位置を、Ａ３，Ａ４として教示した場合。

```
PROGRAM  SAMPLE
  SPEED=20
  DOUT(-201)
  MOVE A3
  MOVE A1
  DOUT(201)
  DELAY 0.5
  MOVE A3
  MOVE A4
  MOVE A2
  DOUT(-201)
```

```
    DELAY 0.5
    MOVE A4
END
```

(b) 「MOVE A1+POINT(0,0,20) 」の形で教示位置の上方の位置へ動作させる場合。

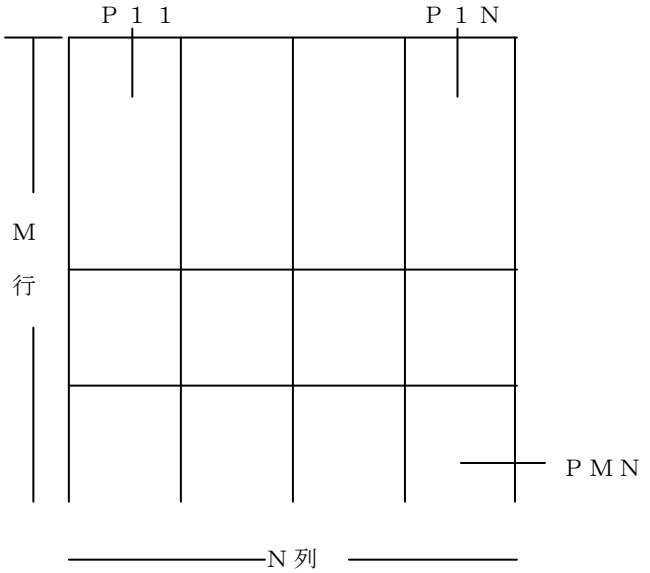
```
PROGRAM  SAMPLE
  SPEED=20
  DOUT(-201)
  MOVE A1+POINT(0,0,20)
  MOVE A1
  DOUT(201)
  DELAY 0.5
  MOVE A1+POINT(0,0,20)
  MOVE A2+POINT(0,0,20)
  MOVE A2
  DOUT(-201)
  DELAY 0.5
  MOVE A2+POINT(0,0,20)
END
```

(c) ショートカット動作で行う場合。

```
PROGRAM  SAMPLE
  SPEED=20
  DOUT(-201)
  ENABLE PASS
  PASS=80
  MOVE A1+POINT(0,0,50)
  DISABLE PASS
  MOVE A1
  DOUT(201)
  DELAY 0.5
  ENABLE PASS
  MOVE A1+POINT(0,0,50)
  MOVE A2+POINT(0,0,50)
  DISABLE PASS
  MOVE A2
  DOUT(-201)
  DELAY 0.5
  ENABLE PASS
  MOVE A2+POINT(0,0,50)
END
```

(7) パレタイズ

右図のようなパレットに部品を置いていくプログラムを考えます。



パレットのサイズ：

Ｍ行×Ｎ列

教示位置：

Ｐ１１：パレットの１行１列目の位置

Ｐ１Ｎ：パレットの１行Ｎ列目の位置

ＰＭＮ：パレットのＭ行Ｎ列目の位置

ＰＰ　：部品の取出し位置

Ｐ０　：待機位置

入出力信号の割付け：

ＤＩ１：パレット準備完了（パレットが作業位置にある時オン）

ＤＩ２：部品取出し準備完了（部品が取出し可能な状態にある時にオン）

ＤＯ１：ハンド閉（信号オンでハンド閉）

ＤＯ２：１パレット作業完了（１パレット全てに部品を挿入したらオン、パレットが払出されて、パレット準備完了がオフするとオフする。）

```
PROGRAM PALLET
 MOVE P0
 RESET DOUT
 M=10                                  パレットの行列数を定める。
 N=15
 RX=(P1N.X-P11.X)/(N-1)                １要素当りのシフト量を計算する。
 RY=(P1N.Y-P11.Y)/(N-1)
 LX=(PMN.X-P1N.X)/(M-1)
 LY=(PMN.Y-P1N.Y)/(M-1)
 PASS=80
 ENABLE PASS
 ENABLE NOWAIT
 L=0
 LOOPL:                                各行についての繰返し
   R=0
```

```
  LOOPR:                                          各列についての繰返し
    MOVE PP+POINT(0,0,50)
    WAIT DIN(2)
    MOVE  PP
    WAIT MOTION>=100
    DOUT(1)                                       部品を取出す
    DELAY 0.5
    MOVE PP+POINT(0,0,50)
    MOVE P11+POINT(R*RX+L*LX,R*RY+L*LY,100)
    WAIT DIN(1)
    MOVE P11+POINT(R*RX+L*LX,R*RY+L*LY)
    WAIT MOTION>=100
    DOUT(-1)                                      部品を置く
    DELAY 0.5
    MOVE P11+POINT(R*RX+L*LX,R*RY+L*LY,100)
  R=R+1
  IF R<=N-1 THEN GOTO LOOPR
  L=L+1
  IF L<=M-1 THEN GOTO LOOPL
  WAIT MOTION>=100
   DOUT(2)
  WAIT DIN(-2)
  MOVE P0
 END
```

(8) 挿入異常監視プログラムの組み方

部品などの挿入異常が発生したとき、それをロボット言語のプログラムで検出して異常処理を行うプログラムは次のようになります。

Ａ１：部品を挿入する位置

```
ACCUR=COARSE                                      位置決め精度を粗にする①。
MOVE A1+POINT(0,0,50)                             部品挿入点の上方50㎜の位置へ移動。
SETGAIN(0,0,1,0,0)                                Ｚ軸を除いた軸のサーボ制御を切る②。
TORQUE={300,300,50,300,300}                       Ｚ軸のトルクを50% に制限③。
MOVE A1                                           部品を挿入。
WAIT MOTION≧100
SETGAIN(0,0,0,0,0)                                全軸のサーボ制御を切る④。
SETGAIN(0,0,1,0,0)                                Ｚ軸のサーボ制御を生かす⑤。
MOVE HERE
```

```
HS=HERE.Z-A1.Z
IF HS<5 THEN GOTO OK
SETGAIN(1,1,1,1,1)
DELAY 0.1
TORQUE={300,300,300,300,300}
DELAY 0.1
MOVE A1+POINT(0,0,50)
```

現在位置と挿入位置の高さの差が5㎜未満なら挿入正常と見なす⑥。

全軸のサーボ制御を生かし、トルクを300%に戻し部品挿入点の上方50㎜の位置へ移動。

挿入異常時の処理

```
OK:
  SETGAIN(1,1,1,1,1)
  DELAY 0.1
  TORQUE={300,300,300,300,300}
  DELAY 0.1
  MOVE A1+POINT(0,0,50)
```

全軸のサーボ制御を生かし、トルクを300%に戻し部品挿入点の上方50㎜の位置へ移動。

挿入正常時の処理

プログラム解説中の丸数字に対するコメントは下記の通りです。

① 位置決め精度を粗にしておかないと、エラーを検出して自動運転が停止する場合があります。

② Ｚ軸を除くサーボ制御を切ることにより、ロボットがＸ－Ｙ平面上で自由に動けるようになります。これにより、ワーク、部品に面取りがしてあるような場合に、部品を面取りに沿って挿入することができます。

③ ＴＯＲＱＵＥ命令を使用してモータの発生するトルクを制限し、挿入異常が発生した時にワーク、ハンドに加わる力を抑えます。（通常は定格の300%になっています。）またトルクを100%未満に抑えることで、コントローラのエラー検出のレベルが下がります。このため、挿入異常を監視するような場合にはトルクは100%未満に設定する必要があります。
なおトルクの設定値を下げすぎるとロボットが動作するために必要なトルクが得られずにエラーになる場合があります。

④ ＨＥＲＥ命令で読込まれる位置はロボットの指令位置になります。ＨＥＲＥ命令でロボットの現在位置を読込むためには、一旦サーボ制御を切り、再度サーボ制御を生かす操作をしなければなりません。このためにＧＡＩＮ命令を使用して全軸のサーボ制御を切っています。

⑤ ④で切ったサーボ制御を生かします。

⑥ 現在位置と挿入位置との高さの差は、適当な値を設定します。

(9) ショートカット動作のプログラム例

ピックアンドプレースをショートカット動作で行うプログラム例を示します。

A：部品取出し位置　D：部品挿入位置　F：不良部品払出し位置

ロボットは部品取出し位置Aから部品を取出し、B，C点を経由して、部品挿入位置Dに部品を挿入します。部品に不良があった場合には、Dからさらに C，E点を経由して、不良部品払出し位置Fに部品を置きます。
動作はショートカット動作主体で行います。

本例では以下の信号を使用します。

入力信号
ＤＩ１：取出し準備完了 …… 取出す部品の準備が完了するとオンします。
ＤＩ２：挿入準備完了 …… 部品挿入位置の準備ができるとオンします。
ＤＩ３：部品不良 …… 把持した部品に不良があるとオンします。
ＤＩ４：不良部品払出し準備完了 … 不良部品払出し位置の準備ができるとオンします。

出力信号
ＤＯ１：部品取出し完了 …… 部品取出しが完了して､次の部品の準備を開始できる状態にあるとオンします。
ＤＯ２：部品挿入完了 …… 部品挿入が完了して､次の部品を挿入するための準備を開始できる状態になるとオンします。
ＤＯ３：不良部品払出し完了 …… 不良部品払出しが完了して､次の不良部品のための準備を開始できる状態になるとオンします。

本例のプログラムは次のようになります。

```
ＰＲＯＧＲＡＭ　ＰＩＣＫＰＬＡＣＥ
´＊　ＰＩＣＫ　ＡＮＤ　ＰＬＡＣＥ　ＳＡＭＰＬＥ　ＰＲＯＧＲＡＭ　＊
　ＩＮＰＵＴ：                          ´初期設定
　　ＯＰＥＮ１
　　ＤＯＵＴ（１，２，３）              ´部品取出し、部品挿入
　　ＥＮＡＢＬＥ　ＮＯＷＡＩＴ          ´不良部品払出し　完了
　　ＳＰＥＥＤ＝８０
　　ＰＡＳＳ＝８０
　　ＥＮＡＢＬＥ　ＰＡＳＳ
　　ＭＯＶＥ　Ｂ                        ´部品取出し位置上空へ

　ＰＩＣＫＵＰ：                        ´つかみ上げ処理
　　ＤＯＵＴ（－１）                    ´部品取出し開始
　　ＷＡＩＴ　ＤＩＮ（１）              ´取出し準備完了待ち
　　ＭＯＶＥ　Ａ                        ´下降
　　ＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００
　　ＣＬＯＳＥ１                        ´ハンド閉
　　ＤＥＬＡＹ　０．３
　　ＭＯＶＥ　Ｂ                        ´上昇

　ＰＬＡＣＥ：                          ´つかみおろし処理
　　ＭＯＶＥ　Ｃ                        ´部品挿入位置上空へ
　　ＤＯＵＴ（１，－２）                ´部品取出し完了、部品挿入開始
　　ＷＡＩＴ　ＤＩＮ（２）              ´挿入準備完了待ち
　　ＭＯＶＥ　Ｄ                        ´下降
　　ＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００
　　ＩＦ　ＤＩＮ（３）　ＴＨＥＮ　ＧＯＴＯ　ＥＲＲ　´部品不良判断
　　ＯＰＥＮ１                           ´ハンド開
　　ＤＥＬＡＹ　０．３
　　ＭＯＶＥ　Ｃ                        ´上昇
　　ＭＯＶＥ　Ｂ                        ´部品取出し位置上空へ
　　ＤＯＵＴ（２）                      ´部品挿入完了
　　ＩＦ　ＭＯＤＥ＝＝ＣＹＣＬＥ　ＴＨＥＮ　ＧＯＴＯ　ＣＹＣＬＥＥＮＤ
                                         ´モードのチェック
　　ＧＯＴＯ　ＰＩＣＫＵＰ
```

```
ＥＲＲ：                                  ´不良部品発生時処理
  ＭＯＶＥ　Ｃ                            ´部品挿入位置上空へ
  ＭＯＶＥ　Ｅ                            ´不良部品払出し位置上空へ
  ＤＯＵＴ（－３）                        ´不良部品払出し開始
  ＷＡＩＴ　ＤＩＮ（４）                  ´不良部品払出し準備完了待ち
  ＭＯＶＥ　Ｆ                            ´下降
  ＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００
  ＯＰＥＮ１                              ´ハンド開
  ＤＥＬＡＹ　０．３
  ＭＯＶＥ　Ｅ                            ´上昇
  ＭＯＶＥ　Ｂ                            ´部品取出し位置上空へ
  ＤＯＵＴ（３）                          ´不良部品払出し完了
  ＩＦ　ＭＯＤＥ＝＝ＣＹＣＬＥ　ＴＨＥＮ　ＧＯＴＯ　ＣＹＣＬＥＥＮＤ
                                          ´モードチェック
  ＧＯＴＯ　ＰＩＣＫＵＰ

ＣＹＣＬＥＥＮＤ：                        ´サイクル停止処理
  ＭＯＶＥ　Ｒ                            ´待機位置へ
ＥＮＤ
```

第５章

プログラミング上の注意事項

本章では、実際にプログラムを作成する上での、プログラムの実行タイミング、禁止事項、及び注意事項について説明します。

5.1 プログラムの実行タイミング

ロボット言語のプログラムは、先頭の命令から１つずつ実行します。通常、動作命令を実行すると、ロボットは次の命令が入出力命令の場合にはアームの位置決め完了を待ってから次の命令を行うため全体の動作時間が長くなってしまいます。

もし、ロボットの動作中に入出力を行うことができれば、動作時間を短縮することができます。このため、ＳＣＯＬ言語では、アームの動作と信号の入出力や通信の処理を並行して実行できるようになっています。

5.1.1 アームの動作と信号入出力のタイミング

信号の入出力を行う時に、アームの動作が完了するまで待つかどうかは、ロボット言語のプログラムにて指定します。信号出力の処理をアームの位置決め完了を待って行うかどうかは、システムスイッチＮＯＷＡＩＴによって指定します。

ＥＮＡＢＬＥ　ＮＯＷＡＩＴ　　アームの位置決め完了を待たない

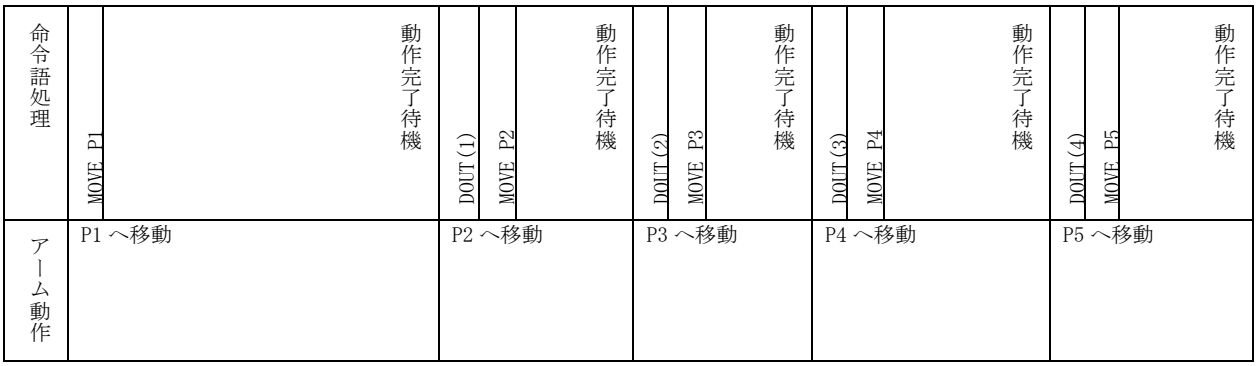
ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴ　　アームの位置決め完了を待つ

初期状態では、ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴの状態になっています。以下に、命令語の処理、アーム動作と信号入出力のタイミングについて、例を示して説明します。

例）　下記のプログラムを実行する場合を考えます。

```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
  ＭＯＶＥ　Ｐ１
  ＤＯＵＴ（1）
  ＭＯＶＥ　Ｐ２
  ＤＯＵＴ（２）
  ＭＯＶＥ　Ｐ３
  ＤＯＵＴ（３）
  ＭＯＶＥ　Ｐ４
  ＤＯＵＴ（４）
  ＭＯＶＥ　Ｐ５
  ＤＯＵＴ（５）
ＥＮＤ
```

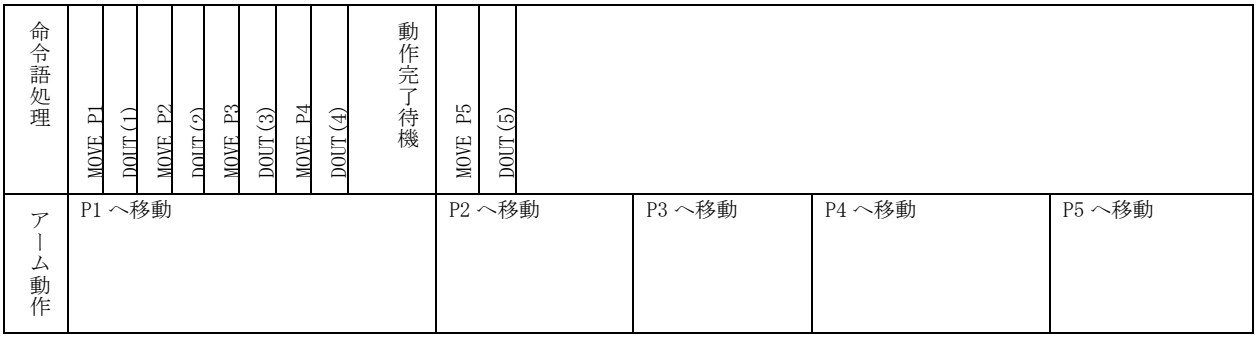
(1) ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴ　の時のタイミング



上図の様に、アーム動作完了後に信号が出力されます。

(注)ＡＣＣＵＲＥ＝ＣＯＡＲＳＥを指定すると、位置決め完了を待たずに続く命令が実行されるため、ロボットが完全に停止する前に信号が出力される場合があります。

(2) ＥＮＡＢＬＥ　ＮＯＷＡＩＴ　の時のタイミング



動作自体は､前の命令によるアーム動作が完了して次の命令によるアーム動作に移ります。しかしアーム動作中であっても、その動作命令の次に述された信号出力の処理を実行します。これを動作命令の先読みと呼び、本ロボットコントローラは最大四動作命令を先読みします(Ｐ１への動作中にＰ４への動作命令まで先読みし､Ｐ５への動作命令実行直前で動作完了を待ちます)。

5.1.2 アームの動作とプログラム実行の同期

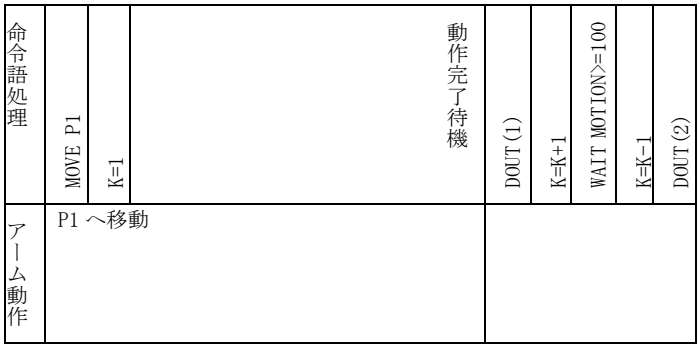
前節では、アーム動作中と信号入出力が並行実行していると説明しましたが、アーム動作とそれ以外のＳＣＯＬ命令を並行実行できるという説明が正解です。

このため通常のＳＣＯＬプログラムは､見かけ上はプログラム処理がアーム動作に先行する形となります。これは、アーム動作とプログラム実行を同期させるためには、特別な手順が必要であるということです。それがＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００命令です。この命令は、アームの位置決めが完了するまで終了しませんので、同期したい命令の前にＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００命令を記述します。以下に、命令語の処理、アーム動作と信号出力のタイミングについて、例を示して説明します。

例） 下記のプログラムを実行する場合を考えます。

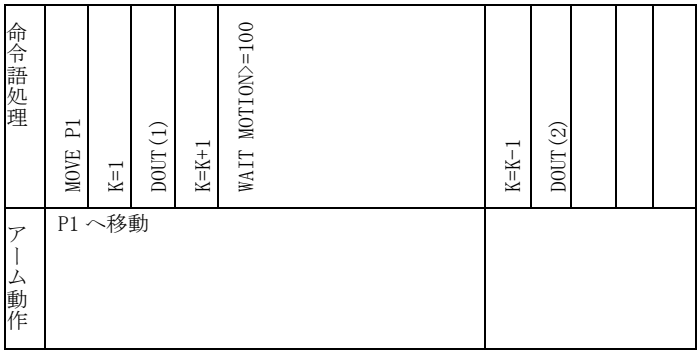
```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
  ＭＯＶＥ　Ｐ1
  Ｋ＝１
  ＤＯＵＴ（１）
  Ｋ＝Ｋ＋１
  ＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００
  Ｋ＝Ｋ－１
  ＤＯＵＴ（２）
ＥＮＤ
```

(1)ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴ　の時のタイミング



ＤＯＵＴ命令が、アーム動作が終了するまで待ちます。

(2)ＥＮＡＢＬＥ　ＮＯＷＡＩＴ　の時のタイミング



ＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００命令が、アーム動作が終了するまで待ちます。

(注)信号の入出力に限らず、以下の命令以外はロボットの動作中いかんにかかわらず処理が進みます。

ＤＩＮ　ＢＣＤＩＮ　ＩＮＰＵＴ　ＰＬＣＩＮＰＵＴ
ＤＯＵＴ　ＢＣＤＯＵＴ　ＰＵＬＯＵＴ　ＰＲＩＮＴ　ＰＬＣＰＲＩＮＴ

5.1.3 ＤＥＬＡＹ命令とＷＡＩＴ命令

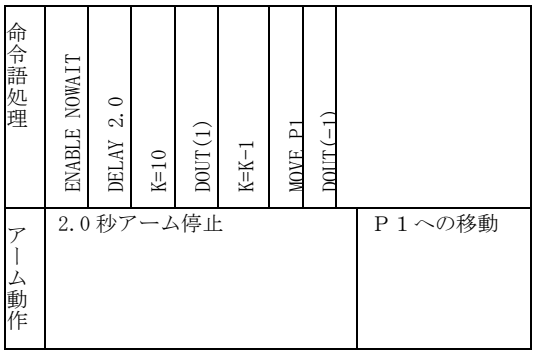
プログラム実行中にアーム動作を指定時間停止するためには、ＤＥＬＡＹ命令を用いる方法と、ＷＡＩＴ命令とＴＩＭＥＲ命令を組合わせて用いる方法があります。それぞれの方法には実行タイミングの違いがありますので、プログラムの作成時には十分注意してください。

(1) ＤＥＬＡＹ命令

ＤＥＬＡＹ命令は動作制御命令の一種で、アーム動作を指定時間停止します。しかし、ＳＣＯＬ言語のプログラムはアーム動作中にも並行して処理されるため、ＤＥＬＡＹ命令の実行中も、プログラムは実行されます。

例） 下記のプログラムを実行する場合を考えます。

```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
  ＥＮＡＢＬＥ　ＮＯＷＡＩＴ
  ＤＥＬＡＹ　２．０
  Ｋ＝１０
  ＤＯＵＴ（１）
  Ｋ＝Ｋ－１
  ＭＯＶＥ　Ｐ１
  ＤＯＵＴ（－１）
ＥＮＤ
```



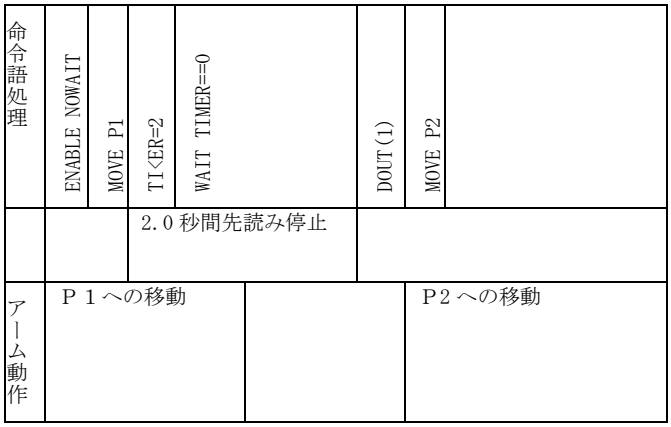
ＤＥＬＡＹ命令にてアーム動作が停止中にも、後続のプログラムは実行されるため、アーム停止中に、１番の信号を出力してしまいます。プログラムの実行も待つようにするためには、ＤＥＬＡＹ命令とＤＯＵＴ命令の間にＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００命令を追加します。

(2) ＷＡＩＴ命令

ＷＡＩＴ命令とＴＩＭＥＲ命令を組合わせてプログラムの実行を指定時間待たせることもできます。しかし（１）と同様に、ロボット言語のプログラムはロボットの動作中にも並行して処理されるために、実行タイミングには注意が必要です。

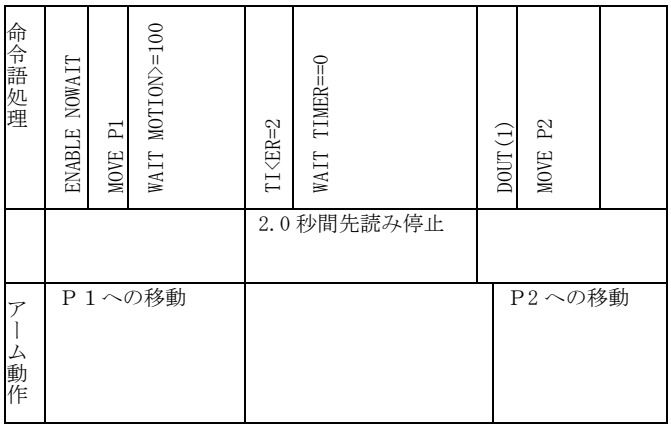
例） 下記のプログラムを実行する場合を考えます。

```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
　ＥＮＡＢＬＥ　ＮＯＷＡＩＴ
　ＭＯＶＥ　Ｐ１
　ＴＩＭＥＲ＝２
　ＷＡＩＴ　ＴＩＭＥＲ＝＝０
　ＤＯＵＴ（１）
　ＭＯＶＥ　Ｐ２
ＥＮＤ
```



ＷＡＩＴ命令によるプログラムの実行待は、ロボットのアーム動作中に終了します。ロボットのアーム動作も待つ必要がある場合には、次のようにＷＡＩＴ命令を挿入します。

```
ＭＯＶＥ　Ｐ１
ＷＡＩＴ　ＭＯＴＩＯＮ＞＝１００
ＴＩＭＥＲ＝２
ＷＡＩＴ　ＴＩＭＥＲ＝＝０
```



5.2 プログラミング上の禁止事項

本節では、プログラムを作成する上での制約事項、及び禁止事項について説明します。
個々の命令語に関する制約、禁止事項については、３章を参照してください。

5.2.1 変数

(1) 未定義変数の参照

変数のデータ型は、代入により初めて決定されるため、初めて用いる変数を参照するようなプログラムは作成しないようにしてください。プログラム中で代入前の変数を参照すると、変数の型と値は不定となります。プログラム上好ましくありませんのでおやめください。

例）
```
ＰＲＯＧＲＡＭ　　ＳＡＭＰＬＥ
    ＩＦ　Ｋ＜＞０　ＴＨＥＮ　ＧＯＴＯ　ＲＥＳＴＡＲＴ
    Ｋ＝１
    Ａ１＝Ａ
ＲＥＳＴＡＲＴ：
    ＦＯＲ　Ｎ＝１　ＴＯ　３
        ＭＯＶＥ　Ａ１
        Ａ１＝Ａ１＋ＰＯＩＮＴ（１００，１００）
    ＮＥＸＴ　Ｎ
ＥＮＤ
```

上記のプログラムを初めて実行した場合、ＩＦ文で参照している変数Ｋの値は、ＩＦ文の実行前にＫへの代入が行われていないため、不定となります。２回目以降はＫに１が代入されるため、Ｋ＝１として処理が行われます。

5.3 プログラミング上の注意事項

本節では、プログラミング上の注意事項と、プログラムを作成する上で参考になる、ＳＣＯＬ言語の内部構造の概要について説明します。

5.3.1 命令語の種類

ＳＣＯＬ言語の機能的な分類は、１章にて説明しましたが、ここでは、ＳＣＯＬ言語を内部処理的に分類して説明します。

ＳＣＯＬ言語の命令語は､内部的に基本命令語､関数､及びシステム変数に分類できます。基本命令語は、ＳＣＯＬ言語の基本となる命令語で、命令語の後ろのパラメータと組合わせて解釈実行します。関数は、ＳＣＯＬ言語をより使い易くするために付加えられたものです。

システム変数は、速度、座標系等システムにて直接参照する変数を、プログラム上から変更できるようにしたもので、通常の変数と同様に取扱う事ができます。以下に、各命令語について説明します。

(1) 基本命令語

ＳＣＯＬ言語の基本となる命令語です。（２）に示す関数と異なり、それ自身を式の中で使うことはできません。

基本命令語を、以下に示します。

ＭＯＶＥ	ＭＯＶＥＳ	ＭＯＶＥＣ	ＭＯＶＥＡ
ＭＯＶＥＩ	ＭＯＶＥＪ	ＤＥＬＡＹ	ＢＲＥＡＫ
ＰＡＵＳＥ	ＲＥＳＵＭＥ	ＯＮ～ＤＯ	ＩＧＮＯＲＥ
ＧＯＴＯ	ＲＥＴＵＲＮ	ＷＡＩＴ	ＳＴＯＰ
ＩＦ～ＴＨＥＮ～	ＥＬＳＥ	ＦＯＲ～ＮＥＸＴ	
ＰＲＩＮＴ	ＩＮＰＵＴ	ＥＮＡＢＬＥ	ＤＩＳＡＢＬＥ
ＰＲＯＧＲＡＭ	ＥＮＤ	ＲＥＳＥＴ	ＲＥＭＡＲＫ
ＤＩＭ～ＡＳ	ＴＡＳＫ	ＫＩＬＬ	ＳＷＩＴＣＨ
ＧＬＯＢＡＬ	ＭＡＸＴＡＳＫ		
ＲＥＳＴＯＲＥ	ＭＯＶＥＳＹＮＣ	ＳＡＶＥＥＮＤ	ＲＣＹＣＬＥ

(2) 関数

関数は基本命令語と較べて、以下の特徴を持っています。

・ＳＣＯＬ言語のサブルーチンプログラムと同様に、データの受渡しは引数で行います。引数は１０個まで指定できます。

関数は、組込み関数とシステムのライブラリと呼ばれるファイルに登録されているものに分類できます。ライブラリに登録されているものは、システムのＲＡＭドライブに“ＳＣＯＬ．ＬＩＢ”ファイルがなければ使う事はできません。

ＳＣＯＬ言語の関数を、以下に示します。

組込み関数

ＭＯＴＩＯＮ	ＭＯＴＩＯＮＴ	ＲＥＭＡＩＮ	ＲＥＭＡＩＮＴ
ＨＥＲＥ	ＤＥＳＴ	ＰＯＩＮＴ	ＴＲＡＮＳ
ＤＩＮ	ＤＯＵＴ	ＰＵＬＯＵＴ	ＢＣＤＩＮ
ＢＣＤＯＵＴ	ＳＩＮ	ＣＯＳ	ＴＡＮ
ＡＳＩＮ	ＡＣＯＳ	ＡＴＡＮ	ＡＴＡＮ２
ＳＱＲＴ	ＡＢＳ	ＳＧＮ	ＩＮＴ
ＲＥＡＬ	ＬＮ	ＬＯＧ１０	ＥＸＰ
ＭＯＤＥ			

組込み関数はそのものを式の中で使用することができます。

（ただし、ＤＯＵＴ，ＰＵＬＯＵＴ，ＢＣＤＯＵＴの信号出力用の命令語は除きます。）

ライブラリに登録されている関数

ＯＰＥＮ１	ＣＬＯＳＥ１	ＯＰＥＮＩ１	ＣＬＯＳＥＩ１
ＯＰＥＮ２	ＣＬＯＳＥ２	ＯＰＥＮＩ２	ＣＬＯＳＥＩ２
ＲＥＡＤＹ	ＯＮＧＡＩＮ	ＯＦＦＧＡＩＮ	ＦＲＥＥＬＯＡＤ
ＳＥＴＧＡＩＮ			

ライブラリに登録されている関数と同一名称のサブプログラムを新たに作成した場合には、新たに作成したサブプログラムの方を優先して処理します。

(3) システム変数

システム変数は、システムの状態を変更するもので、通常の変数と同様に扱います。

ＳＣＯＬ言語のシステム変数を、以下に示します。

ＣＯＮＦＩＧ	ＡＣＣＵＲ	ＡＣＣＥＬ	ＤＥＣＥＬ
ＳＰＥＥＤ	ＰＡＳＳ	ＴＯＲＱＵＥ	ＧＡＩＮ
ＴＯＯＬ	ＢＡＳＥ	ＷＯＲＫ	ＴＩＭＥＲ
ＥＲＲＯＲ	ＰＡＹＬＯＡＤ	ＴＩＤ	

5.3.2 ロボットの座標系

本項では、ＳＣＯＬ言語で扱うロボットの座標系について説明します。

(1) ロボットの座標系

ロボットはワールド座標系、ベース座標系、ワーク座標系、メカニカルインタフェース座標系、及びツール座標系の５つの座標系を持っています。これらの座標系の概要は、次の通りです。

(a) ワールド座標系（絶対座標系）

ロボットを設置した作業場所全体で唯一の座標系です。ロボットとは無関係に設定できます。ロボットの原点位置で設定した場合には、ワールド座標系とベース座標系は一致します。

(b) ベース座標系（機械座標系）

ロボット自身の座標系です。ロボットの構造から決まる固有の座標系になります。
必ずロボットの原点位置が原点になります。

(c) ワーク座標系（作業座標系）

ロボットが作業を行うワークに固有の座標系で、作業により決定します。

(d) メカニカルインタフェース座標系

ロボットに、エンドエフェクタを取付ける位置の座標系です。ロボット自身が動作する事により、この座標系も移動します。

(e) ツール座標系

ロボットのツール先端の座標系です。ロボット自身が動作する事により、この座標系も移動します。

これらの座標系は、ロボットを誘導する時、ロボットに位置を教示する時、及びロボットが動作する時に関係します。ロボットを誘導する時には、ワールド、ワーク、ツールの各座標系を指定することができます。ロボットは、指定された座標系に沿って動作します。ロボットに位置を教示する時、及びロボットが動作する時には、特に座標系を意識する必要はありません。普通の使い方では、ロボットは教えられた位置に動作します。

注意　座標系と付加軸、

スカラロボットの５軸目（Ｔ軸）、直角座標ロボットの４軸（Ｃ軸）、５軸目（Ｔ軸）は、付加軸であり座標系の影響を受けません。どの座標系を選択しても同じ値となります。

図5.1 に、ロボットの座標系を示します。

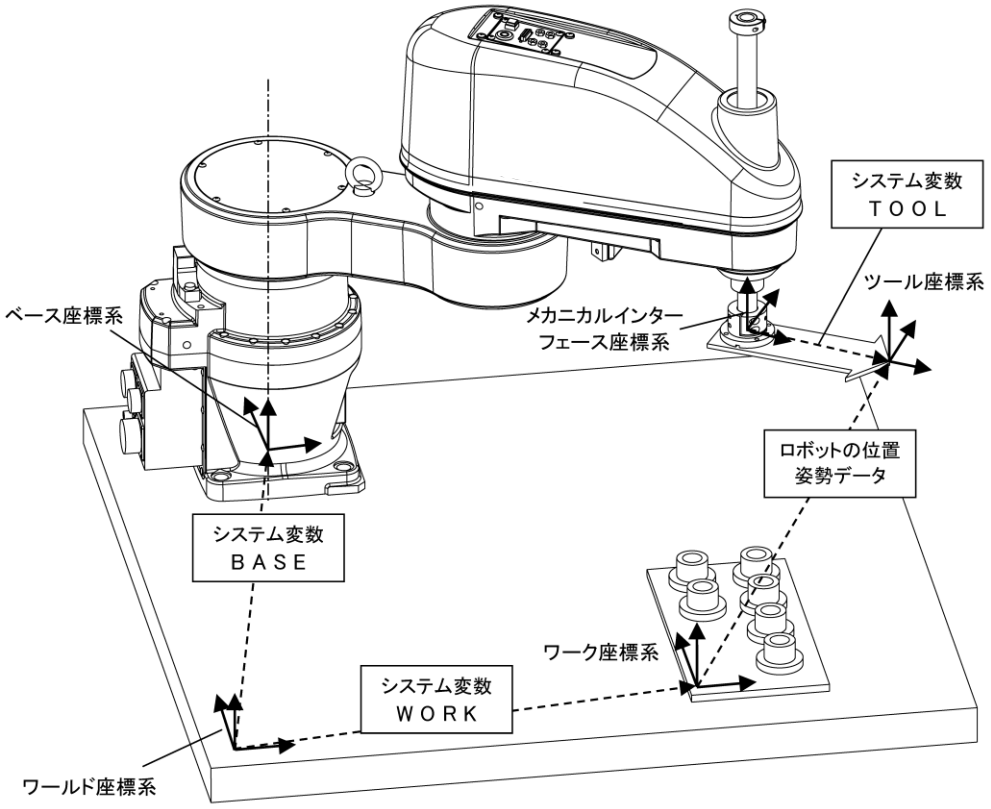


図 5. 1 ロボットの座標系

(2) 座標系とシステム変数

ベース、ワーク、ツールの３つの座標系については、それぞれに複数の座標系を設定して、作業によって座標系を選択して使用する事ができます。座標系の指定は、ＳＣＯＬ言語のプログラム中でも行なえます。座標系を指定する命令として、ＢＡＳＥ，ＷＯＲＫ，ＴＯＯＬの各システム変数が用意されています。これらのシステム変数は、通常の座標型データとして使用する事ができます。

各システム変数の意味は次の通りです。個々の座標系を教示する方法、及び座標系を選択する方法については、「操作編」を参照してください。

(a) システム変数ＢＡＳＥ

ワールド座標系から見たベース座標系を指定します。ロボットの原点とは別に、作業場所自体の座標系を基準にする場合や、ベース座標系と異なるワールド座標系に沿ってロボットを誘導する必要がある時に設定してください。ロボットの設置位置を変更した場合にも、ベース座標系を再設定すればわずかな再教示作業にて、以前と同様の作業を行う事ができます。システム変数ＢＡＳＥの値をすべて０にすると、ワールド座標系とベース座標系は一致します。特にワールド座標系を指定する必要のない場合にはシステム変数ＢＡＳＥの値はすべて０にしてください。また、一度設定したベース座標系は不用意に変更しないでください。

(b) システム変数ＷＯＲＫ

ワールド座標系から見たワーク座標系を指定します。システム変数ＷＯＲＫの値をすべて０にすると、ワールド座標系とワーク座標系は一致します。複数のワークに対して作業を行うときには、複数のワーク座標系を設定して、ワーク毎に使い分ける事ができます。その他、ワークの座標系に沿ってロボットを誘導する必要がある場合にも設定してください。特に、ワーク座標系を指定する必要のない時には、システム変数ＷＯＲＫの値はすべて０にしてください。

(c) システム変数ＴＯＯＬ

メカニカルインタフェース座標系から見たツール座標系を指定します。システム変数ＴＯＯＬの値をすべて０にすると、メカニカルインタフェース座標系とツール座標系は一致します。複数のツールを使用する場合には、複数のツール座標系を設定して、ツール毎に使い分ける事ができます。その他、ツールの座標系に沿ってロボットを誘導する必要がある場合にも設定してください。

プログラム上でこれらのシステム変数を操作すると、ロボットの動作する座標系が変わってしまうため、注意が必要です。

(3) 教示データと座標系

ロボットに対して位置を教示する時には、ロボットはワーク座標系から見たツール先端の位置を位置データとして記憶します。この時にロボットは、どのワーク座標系を使用したかという情報も一緒に記憶します。ロボットが動作する時には、現在のツール、ベース座標系を用いて、ロボットのツール先端が教示時に指定したワーク座標系での位置になるようにします。このため、教示時と動作時で指定した座標系が変更されていると、ロボットは教示した位置へ動作しません。

(4) プログラム上での座標系データの変更

以下には、プログラム上で座標系データを変更する使い方について簡単に説明します。
特に必要のないときには、プログラム中で座標系データを変更しないでください。

(a) ベース座標系の変更

プログラム中でベース座標系を変更する必要はありません。

(b) ワーク座標系の変更

ロボットに位置データを教示すると、どのワーク座標系を使用したかという情報も一緒に記憶します。プログラムの動作命令を実行すると、ワーク座標系の値は教示した時の値に書替えられます。

例） ワーク座標系データとして、ＷＯＲＫ１，ＷＯＲＫ２，ＷＯＲＫ３の３つを考えます。位置データとしてはＡ１，Ａ２，Ａ３の３点がそれぞれＷＯＲＫ１，ＷＯＲＫ２,ＷＯＲＫ３上で教示されている場合に次のプログラムを実行すると、ワーク座標系はプログラムの右に示したコメントのように変化します。

```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
  ＭＯＶＥ　Ａ１　　‘ワーク座標系はＷＯＲＫ１になります
  ＭＯＶＥ　Ａ２　　‘ワーク座標系はＷＯＲＫ２になります
  ＭＯＶＥ　Ａ３　　‘ワーク座標系はＷＯＲＫ３になります
ＥＮＤ
```

複数のワークに対して、ワーク座標系を変更する事により同じ作業を行うには、次の方法があります。

1) ワーク座標系自体を変更する方法

プログラム中でワーク座標系の座標系データ自身の値を変更して、異なるワークに対して同じ作業を行います。

例） ワーク座標系ＷＯＲＫ１にて教示された位置データＡ１，Ａ２，Ａ３において、ワーク座標系ＷＯＲＫ２にて作業を行う場合は、次のようになります。

```
ＰＲＯＧＲＡＭ　　ＳＡＭＰＬＥ
  ＤＵＭＭＹＷＯＲＫ＝ＷＯＲＫ１
  ＷＯＲＫ１＝ＷＯＲＫ２
  ＭＯＶＥ　Ａ１
  ＭＯＶＥ　Ａ２
  ＭＯＶＥ　Ａ３
  ＷＯＲＫ１＝ＤＵＭＭＹＷＯＲＫ
ＥＮＤ
```

この例では，Ａ１，Ａ２，Ａ３すべてＷＯＲＫ１をワーク座標系として動作しますが、ＷＯＲＫ１自身をＷＯＲＫ２に書替えて使用しているため、ロボットはＷＯＲＫ２のワーク座標系にて動作します。ＷＯＲＫ１にＷＯＲＫ２の値を代入すると、元のＷＯＲＫ１の値が無くなってしまうので、ＷＯＲＫ１の値は事前に他の変数（この例ではＤＵＭＭＹＷＯＲＫという変数）に保存して、作業の終了後に復帰しています。

2) ＷＩＴＨ節を使用する方法

動作命令にＷＩＴＨ節を用いて使用するワーク座標系を指定して動作します。

例） ワーク座標系ＷＯＲＫ１にて教示された位置データＡ１，Ａ２，Ａ３において、ワーク座標系ＷＯＲＫ２にて作業を行う場合は、次のようになります。

```
ＰＲＯＧＲＡＭ　　ＳＡＭＰＬＥ
 ＭＯＶＥ　Ａ２
 ＷＯＲＫ２＝ＷＯＲＫ＋ＴＲＡＮＳ（，，２０）
 ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＷＯＲＫ＝ＷＯＲＫ２
 ＭＯＶＥ　Ａ２　ＷＩＴＨ　ＷＯＲＫ＝ＷＯＲＫ２
 ＭＯＶＥ　Ａ３　ＷＩＴＨ　ＷＯＲＫ＝ＷＯＲＫ２
ＥＮＤ
```

ＷＩＴＨ節を用いて、教示時に用いたものと異なるワーク座標系を指定して処理する事が可能です。

3) 一般的な方法

一般的には、上記２つの方法を組合わせて使用する事をお勧めします。すなわち、ロボットの動作は、全て仮の変数をワーク座標系として使用するようにプログラムしておいて、ワークが変わる度に、仮の変数に実際に使用するワーク座標系を代入して動作します。

例） ワーク座標系ＷＯＲＫ１にて教示された位置データＡ１，Ａ２，Ａ３において、ワーク座標系ＷＯＲＫ２にて作業を行う場合は、次のようになります。

```
ＰＲＯＧＲＡＭ　　ＳＡＭＰＬＥ
  ＷＯＲＫ２＝ＷＯＲＫ＋ＴＲＡＮＳ（，，２０）
  ＤＵＭＭＹＷＯＲＫ＝ＷＯＲＫ２
  ＭＯＶＥ　Ａ１　ＷＩＴＨ　ＷＯＲＫ＝ＤＵＭＭＹＷＯＲＫ
  ＭＯＶＥ　Ａ２　ＷＩＴＨ　ＷＯＲＫ＝ＤＵＭＭＹＷＯＲＫ
  ＭＯＶＥ　Ａ３　ＷＩＴＨ　ＷＯＲＫ＝ＤＵＭＭＹＷＯＲＫ
ＥＮＤ
```

(c) ツール座標系の変更

複数のツールを取替えながら使用する場合には、ツール座標系を変更する事により対応できます。ロボットは、いつでも現在のツール座標系の値を用いて、ロボットのツール先端がワーク座標系の指定位置になるように動作します。このため、使用するツール座標系の値を間違えると、ロボットは思わぬ位置に移動してしまいます。

例） ２本のツール（ツールオフセットはＴＯＯＬ１とＴＯＯＬ２とする）を使用する場合は、次のようになります。

```
ＰＲＯＧＲＡＭ ＳＡＭＰＬＥ
ＴＯＯＬ１＝ＴＲＡＮＳ（，，１０）
ＴＯＯＬ２＝ＴＲＡＮＳ（，，３０）
ＴＯＯＬ＝ＴＯＯＬ１
ＭＯＶＥ Ａ１
ＴＯＯＬ＝ＴＯＯＬ２
ＭＯＶＥ Ａ２
ＥＮＤ
```

上記のプログラムでは、２本の異なるツールに対して、ツール先端は同じＡ１の位置へと動作します。（上記のプログラムでは、実際にツールを交換する動作は記載されていません。）

5.3.3 ショートカット動作

ＳＣＯＬ言語では、ロボットが動き始めてから位置決めが完了するまでが、１つの動作になります。通常は、１つの動作命令が１つの動作になります。これに対して、複数の動作命令で位置決めなしに連続的に目標点、またはその近傍を通過し、次の目標点に向かって移動する動作を指定する事ができます。この動作をショートカット動作と呼びます。ショートカット動作を使用すると、移動時間を短縮する事ができます。
ＭＯＶＥＳ，ＭＯＶＥＣ命令と、その他の動作命令の間でのショートカット動作は行えません。（ＭＯＶＥＳ命令とＭＯＶＥＣ命令の間のショートカット動作は可能です。）

(1) ショートカット動作の指定

ショートカット動作の開始，終了は、システムスイッチＰＡＳＳにより指定します。
ショートカット動作の開始の指定　ＥＮＡＢＬＥ　ＰＡＳＳ
ショートカット動作の終了の指定　ＤＩＳＡＢＬＥ　ＰＡＳＳ

ショートカット動作では、最大加速度を超えないように連続的に速度を変えながら動作します。ショートカット動作では、１つの動作命令に対して移動量が指定のパーセンテージを超えたら次の動作命令の移動を開始します。このパーセンテージは、システム変数のＰＡＳＳにより指定します。このパーセンテージをショートカット動作のパラメータと呼びます。ショートカット動作のパラメータは、０～１００までの整数値で指定しますが、５０以下を指定した時には、５０％として処理します。

例）

```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
 ＭＯＶＥ　Ａ１              Ａ１まで移動する
 ＰＡＳＳ＝８０              ショートカット動作のパラメータを８０％に設定
 ＥＮＡＢＬＥ　ＰＡＳＳ     ショートカット動作の開始を指定
 ＭＯＶＥ　Ａ２              動作の８０％終了にてＡ３への移動を開始する
 ＭＯＶＥ　Ａ３              動作の８０％終了にてＡ４への移動を開始する
 ＤＩＳＡＢＬＥ　ＰＡＳＳ   ショートカット動作の終了を指定
 ＭＯＶＥ　Ａ４              Ａ４まで移動する
ＥＮＤ
```

ショートカット動作のパラメータはショートカット動作の途中で変更する事ができます。

例）

```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
 ＭＯＶＥ　Ａ１              Ａ１まで移動する
 ＰＡＳＳ＝８０              ショートカット動作のパラメータを８０％に設定
 ＥＮＡＢＬＥ　ＰＡＳＳ     ショートカット動作の開始を指定
 ＭＯＶＥ　Ａ２              動作の８０％終了にてＡ３への移動を開始する
 ＭＯＶＥ　Ａ３　ＷＩＴＨ　ＰＡＳＳ＝６０
                             動作の６０％終了にてＡ４への移動を開始する
 ＭＯＶＥ　Ａ４              動作の８０％終了にてＡ５への移動を開始する
 ＰＡＳＳ＝９０              ショートカット動作のパラメータを９０％に設定
 ＭＯＶＥ　Ａ５              動作の９０％終了にてＡ６への移動を開始する
 ＤＩＳＡＢＬＥ　ＰＡＳＳ   ショートカット動作の終了を指定
 ＭＯＶＥ　Ａ６              Ａ６まで移動する
ＥＮＤ
```

(2) ショートカット動作が中断する命令

プログラムにてショートカット動作を指定しても、次のような命令を使うとショートカット動作が中断します。

ＷＡＩＴ命令

ＩＮＰＵＴ命令

ＰＲＩＮＴ命令

ＳＴＯＰ命令

ＢＲＥＡＫ，ＰＡＵＳＥ命令

ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴ 指定の時に次の各命令を使用した場合

ＤＯＵＴ命令，ＲＥＳＥＴ　ＤＯＵＴ命令，ＰＵＬＯＵＴ命令，ＤＩＮ命令，

ＢＣＤＩＮ命令，ＢＣＤＯＵＴ命令

その他にも、動作命令と動作命令の間に多くの命令がある場合及び個々の動作の移動量が小さい場合にも、ショートカット動作が中断する事があります。

またＭＯＶＥＳ，ＭＯＶＥＣ命令とその他の動作命令の間でショートカット動作を指定した場合にもショートカット動作は中断します。

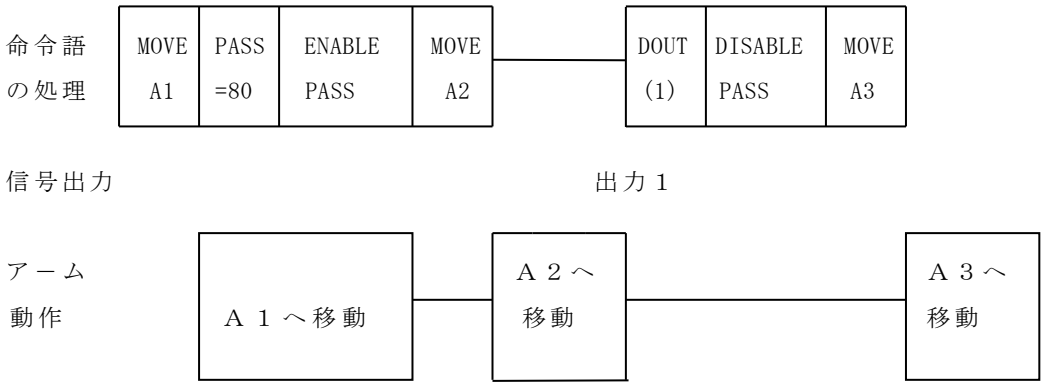
(3) ショートカット動作での信号出力のタイミング

ショートカット動作時のアームの動作と信号出力のタイミングについて、例を示して説明します。

例） 下記のプログラムを実行する場合を考えます。

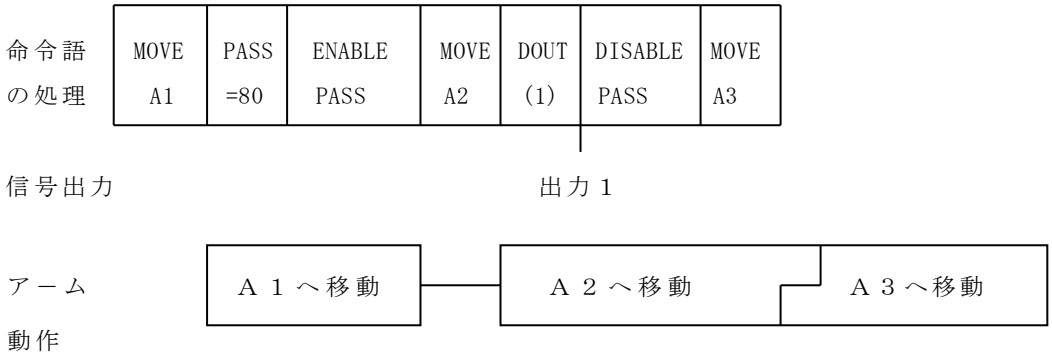
```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
  ＭＯＶＥ　Ａ１
  ＰＡＳＳ＝８０
  ＥＮＡＢＬＥ　ＰＡＳＳ
  ＭＯＶＥ　Ａ２
  ＤＯＵＴ（１）
  ＤＩＳＡＢＬＥ　ＰＡＳＳ
  ＭＯＶＥ　Ａ３
ＥＮＤ
```

(a) ＤＩＳＡＢＬＥ　ＮＯＷＡＩＴ　の場合のタイミング



信号出力の処理のためにショートカット動作が中断します。

(b) ＥＮＡＢＬＥ　ＮＯＷＡＩＴ　の場合のタイミング



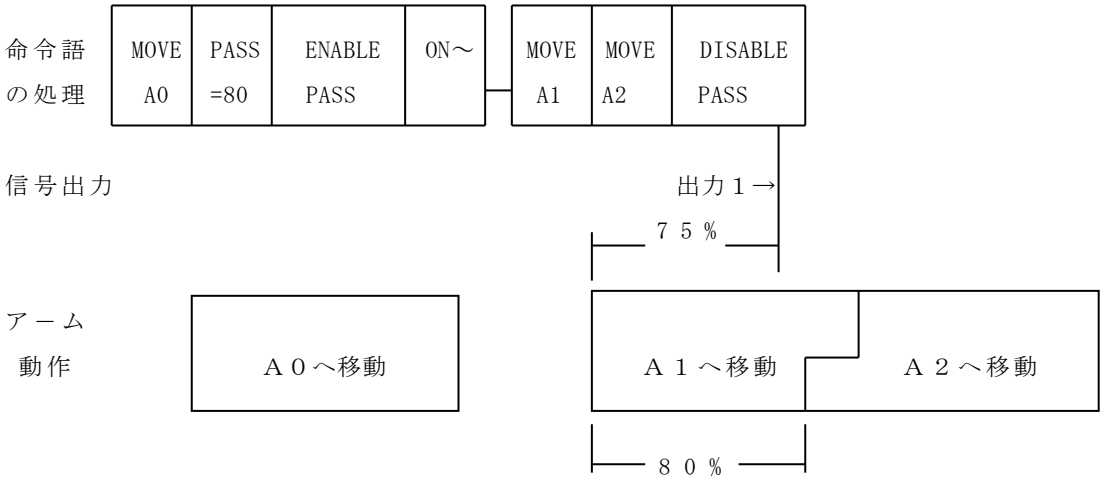
アーム動作中であっても信号出力の処理を行います。

(4) ショートカット動作でのロボット動作状態の参照

ショートカット動作を指定した場合に、ＭＯＴＩＯＮ，ＭＯＴＩＯＮＴ，ＲＥＭＡＩＮ，ＲＥＭＡＩＮＴ命令を用いてロボットの動作状態を参照すると、結果は１つの動作命令に対する値で与えられます。

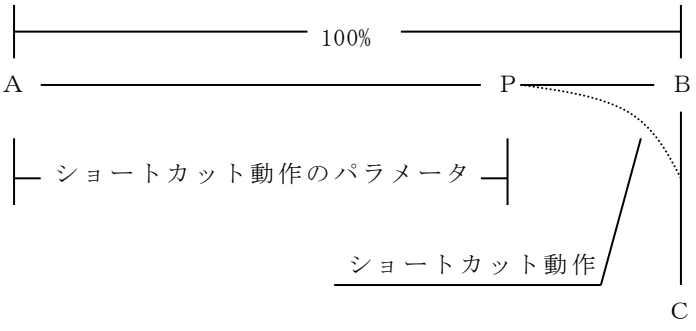
例）
```
ＰＲＯＧＲＡＭ　ＳＡＭＰＬＥ
    ＭＯＶＥ　Ａ０
    ＰＡＳＳ＝８０
    ＥＮＡＢＬＥ　ＰＡＳＳ
    ＯＮ　ＭＯＴＩＯＮ＞＝７５　ＤＯ　ＤＯＵＴ（１）
    ＭＯＶＥ　Ａ１
    ＭＯＶＥ　Ａ２
    ＤＩＳＡＢＬＥ　ＰＡＳＳ
ＥＮＤ
```

上記のプログラムで信号１は、Ａ０からＡ１への移動が７５％終了した時点で出力します。



(5) ショートカット動作での制限事項

ショートカット動作のパラメータは次のような意味を持ちます。



上の図で、Ａから移動を開始して、Ｂの近傍を通り、Ｃまで移動する動作を考えます。ショートカット動作では、ＡからＢまでの間に点Ｐを指定して、ロボットがＰまで移動したら、Ｃに向かっての移動を開始します。点Ｐの位置はロボットの移動距離に対するパーセンテージで指定します。この値がショートカット動作のパラメータになります。またこのパーセンテージをパス率と呼ぶこともあります。

前頁の例で、ショートカット動作のパラメータ（パス率）は次のように与えます。

（パス率）　＝　((ＡからＰまでの距離）／（ＡからＢまでの距離))＊100(%)

ショートカット機能では、ショートカット動作のパラメータを５０～１００％の間で指定します。５０％以下の値を指定しても５０％として処理します。

ただし以下に示すような理由で、指定したパラメータ通りにショートカット動作が行われない場合があります。

(a) 最大加速度による制限

ロボットにはその構造、使用している部品などの条件により、最大加速度が決められています。ロボットの加速度は、ロボットの動作速度を落したり、ロボット言語のＡＣＣＥＬ、ＤＥＣＥＬ命令を使用しても、変化します。ショートカット動作を行う場合には、指定された加速度を超えないように各関節軸の速度を計算します。

(b) 次に続く動作による制限

ショートカット動作では、次の動作の50%を移動するまでには、現在の動作が終了するように、現在移動中の動作に次の動作を重ねます。このため、次に続く動作が現在の動作よりも移動距離が短い場合、指定した位置よりも教示点に近付いた位置から、ショートカット動作を開始します。

これらの理由により、ショートカット動作を開始するタイミングは、ショートカット動作のパラメータ（パス率）を小さくしても、ある一定値以上には速くできません。

(6)ショートカット方向が同一ベクトル方向になる場合での注意事項

ショートカットする動作同士が同一方向になる場合、速度成分の重なりが多くなり、速度が指定した速度より速くなる場合があります。

5.3.4 ロボットの姿勢

スカラ型ロボットでは、ロボットのアームをまっすぐに伸ばした姿勢に対して、ロボット旋回中心から見て、第２アームが左側に曲がっている状態（肘が右へ出ている状態）を右肩系、右側に曲がっている状態（肘が左へ出ている状態）を左肩系の姿勢と呼びます。ワーク座標系にて指定した１つの位置（Ｘ，Ｙ）に対して、ロボットは右肩系と左肩系の２つの姿勢をとる事ができます。

(1) 教示時の姿勢と動作時の姿勢

ロボットに位置を教示すると、ロボットの姿勢も位置データとして記憶されます。
通常の動作では、ロボットは教示された姿勢で動作します。

(a) 動作時の姿勢の指定

動作時のロボットの姿勢は、ＣＯＮＦＩＧ命令によって指定します。ＣＯＮＦＩＧ命令の使い方は、単独で宣言する方法と動作命令にＷＩＴＨ節を用いて指定する方法があります。ＷＩＴＨ節を用いた場合には、１つの動作命令に対してのみ有効になります。
ＣＯＮＦＩＧ命令を単独で使用すると、以降の全ての動作に対してロボットの姿勢を指定する事になります。
ロボットの姿勢を教示時のままで動作したい場合には、ロボットの姿勢を未定義（ＣＯＮＦＩＧ＝０、もしくはＣＯＮＦＩＧ＝ＦＲＥＥ）に設定する必要があります。
ロボットの初期設定はこの状態になっています。
ロボットの姿勢は、ＣＯＮＦＩＧ＝１（もしくはＣＯＮＦＩＧ＝ＬＥＦＴＹ）にて左肩系に、ＣＯＮＦＩＧ＝２（もしくはＣＯＮＦＩＧ＝ＲＩＧＨＴＹ）にて右肩系に設定できます。
ロボットの姿勢を指定すると、以降の動作命令実行時にロボットの姿勢が指定した姿勢に変化します。
ロボットを教示時と異なる姿勢で動作させると、位置がずれる場合があります。ロボットに位置を教示する時には、ロボットが動作する時の姿勢で行ってください。

(b) ロボットの姿勢が変化しない場合

以下の動作命令の実行時には、姿勢の指定は無効になります。

1) 補間動作命令

直線補間、円弧補間を行う命令では、ロボットの姿勢変更は不可のためエラーとなります。補間動作命令は、次の各命令です。

ＭＯＶＥＳ命令，ＭＯＶＥＣ命令

2) 単軸動作命令

ロボットのある軸だけを指定して動かす命令では､ロボットの姿勢は指定できません。
単軸動作命令は、次の各命令です。

ＭＯＶＥＡ命令，ＭＯＶＥＩ命令

(2) プログラム中で合成された位置への動作

動作命令で座標値を直接指定した場合（ＭＯＶＥ　ＰＯＩＮＴ（１００，１００）等）には、ロボットは現在指定されている姿勢で動作します。姿勢の指定が未定義の場合には、現在のロボットの姿勢にて動作します。

また、位置データをプログラム中で合成した場合に、ロボットの姿勢は代入先の位置データが持っていた姿勢になります。代入先の位置データが代入時に初めて定義される時には、ロボットの姿勢は未定義になります。

5.3.5 データブロック

本項では、ロボットの位置データを、実際にどのような形でコントローラのファイルに保存するかについて説明します。本項の内容については、ティーチペンダントを使用してプログラミングを行う上で意識する必要はありません。ＳＣＯＬ言語のプログラム、位置データをコントローラ以外の計算機にて作成する場合に参照してください。なお、ファイルの内容は、すべてアスキーコードになります。

(1) データブロック

ロボットの位置データは、データブロックという単位でコントローラに記憶します。

コントローラのファイルには、複数のプログラムと１つのデータブロックが存在します。１つのファイルのデータブロックは必ず１つです。同一ファイル内のプログラムでは、データブロック内の位置データを共通で使用します。異なるファイルのデータブロックは参照できません。データブロックには、位置データ以外に、座標データ、負荷データも記憶します。

データブロックは、ファイル中に次の形で宣言します。

```
ＤＡＴＡ
（データの宣言）
・・・
ＥＮＤ
```

データブロックは、ＤＡＴＡからＥＮＤまでで宣言します。個々のデータは、ＤＡＴＡからＥＮＤまでの間に１つずつ宣言します。データブロックは、ファイルの最後に宣言します。データブロックをプログラム中に宣言する事はできません。

(2) データの宣言

位置データ、座標データ、及び負荷データは、コントローラのデータエディタ機能を用いて教示します。データエディタ機能により教示されたデータは、自動的に該当するファイルのデータブロックに記憶されます。データエディタを使用しないでプログラム中で定義したデータについては、データブロックには記憶されません。
これらのデータは、プログラム実行時に、コントローラ内のファイルとは別の領域に一時的に作成されます。

データブロック内のデータは、次の形で宣言します。

＜データ型＞ ＜識別子＞＝［＜要素＞,］・・・

＜データ型＞には、宣言するデータのデータ型を指定します。データ型の指定には、位置型、座標型、負荷型のそれぞれに対応して、ＰＯＩＮＴ，ＴＲＡＮＳ，ＰＡＹＬＯＡＤを指定します。＜識別子＞には、宣言するデータの名称を指定します。＜要素＞には、データの各要素を実数で指定します。各要素は省略可能で、省略した要素は０と見なします。

(a) 位置データの宣言

位置データの宣言は、次のようになります。

ＰＯＩＮＴ ＜識別子＞＝Ｘ，Ｙ，Ｚ，Ｃ，Ｔ／＜姿勢＞
Ｘ，Ｙ，Ｚ，Ｃ，Ｔ ：Ｘ，Ｙ，Ｚ，Ｃ，Ｔの各座標値を実数で指定します。
（ｍｍ，ｄｅｇ 単位）
＜姿勢＞ ：０～２の整数値でロボットの姿勢を指定します。
なし ・・・未定義
ＬＥＦＴＹ ・・・左肩系
ＲＩＧＨＴＹ ・・・右肩系

姿勢としては、システム定数のＦＲＥＥ，ＬＥＦＴＹ，ＲＩＧＨＴＹを使用する事もできます。

例） ＰＯＩＮＴ Ａ＝１００，２００，３０，４５，０
ＰＯＩＮＴ Ａ１＝４４４．４４，３３３．３３，，，／ＲＩＧＨＴＹ
ＰＯＩＮＴ ＺＥＲＯ＝

(b) 座標データの宣言

座標データの宣言は、次のようになります。

ＴＲＡＮＳ ＜識別子＞＝Ｘ，Ｙ，Ｚ，Ｃ

Ｘ，Ｙ，Ｚ，Ｃ，Ｔ ：Ｘ，Ｙ，Ｚ，Ｃの各座標値を実数で指定します。
（ｍｍ，ｄｅｇ 単位）

例） ＴＲＡＮＳ ＷＯＲＫ１＝１０，２０，３０，４５
ＴＲＡＮＳ ＴＯＯＬ２＝，，－２０，
ＴＲＡＮＳ ＺＥＲＯＷ＝

(c) 負荷データの宣言

負荷データの宣言は、次のようになります。

ＰＡＹＬＯＡＤ ＜識別子＞＝＜質量＞，＜重心オフセット＞
＜質量＞ ：ロボットの手先の負荷の質量を実数で指定します。（ｋｇ単位）
＜重心オフセット＞ ：ロボットの手先の負荷の重心オフセットを実数で指定します。
（ｍｍ単位）

例） ＰＡＹＬＯＡＤ ＨＡＮＤ１＝４．８，０．４８
ＰＡＹＬＯＡＤ ＨＡＮＤ２＝２，０．００４
ＰＡＹＬＯＡＤ ＨＡＮＤ０＝

(3) ワーク座標系の指定

ＳＣＯＬ言語の位置データは、ワーク座標系でのロボットのツール先端の座標で指定します。位置データには、それぞれどのワーク座標系を使用するかを指定するようになっています。位置データに対するワーク座標の指定には、ＷＯＲＫ文を使用してワーク座標系の名称を指定します。ＷＯＲＫ文に続けて宣言した位置データは、ＷＯＲＫ文で宣言したワーク座標系を使用する事になります。ワーク座標系の指定は、次にＷＯＲＫ文が宣言されるまで有効です。

例）　ＤＡＴＡ

```
  ＰＯＩＮＴ　Ａ００＝６５０，０，０，０，０
  ＰＯＩＮＴ　Ａ０１＝４００，４００，０，０／ＲＩＧＨＴＹ
  ＴＲＡＮＳ　ＷＯＲＫ１＝１００，－１００，０，０
  ＷＯＲＫ　ＷＯＲＫ１
    ＰＯＩＮＴ　Ａ１０＝０，０，０，０，０，０
    ＰＯＩＮＴ　Ａ１１＝２００，０，０，０，０，０
  ＴＲＡＮＳ　ＷＯＲＫ２＝－１００，－１００，０，０，０
  ＷＯＲＫ　ＷＯＲＫ２
    ＰＯＩＮＴ　Ａ２０＝２４６．８，６９．１，２３．５，１８．３，
                                          ／ＲＩＧＨＴＹ
    ＰＯＩＮＴ　Ａ２１＝０，０，－３０，０，０／ＬＥＦＴＹ
  ＥＮＤ
```

上記の例で、位置データＡ１０，Ａ１１はＷＯＲＫ１を、Ａ２０，Ａ２１はＷＯＲＫ２をそれぞれワーク座標系として動作します。また、位置データＡ００，Ａ０１については、データの前にワーク座標系の宣言がないため、ワークを｛０，０，０，０｝として動作します。

5.3.6 グローバルデータブロック

ＳＣＯＬ言語にて定義される変数には大域的データと一時的データが存在します。
本項では、プログラム全域で参照可能な大域的データ（グローバルデータ）について説明します。

データブロックで定義した変数もグローバルデータとして扱いますが、データブロックで定義可能なデータは位置型、座標型、負荷型に限定されます。
しかし、グローバルデータ宣言することにより、大域的な整数型、実数型、配列型のデータが使用可能となります。

(1) グローバルデータブロック

グローバルデータはグローバルデータブロックで宣言しますが、データブロックとは異なりプログラムの一部として扱います。したがって、グローバルデータブロックの編集はプログラムエディタで行います。コントローラのファイルには、複数のプログラムと１つのグローバルデータブロックが存在します。１つのファイルのグローバルデータブロックは必ず１つです。
同一ファイル内のプログラムではグローバルデータを共通で使用できますが、異なるファイルのグローバルデータは参照できません。グローバルデータブロックには整数データ、実数データ、配列データが記憶できます。

グローバルデータブロックは、ファイルの中に次の形で宣言します。

```
ＧＬＯＢＡＬ
    （データの宣言）
・・・
ＥＮＤ
```

グローバルデータブロックは、ＧＬＯＢＡＬからＥＮＤまでで宣言します。
個々のデータは、ＧＬＯＢＡＬからＥＮＤまでの間に１つずつ宣言します。
グローバルデータブロックはファイルの先頭に宣言します。
グローバルデータブロックをプログラム中に宣言することはできません。

(2) データの宣言

グローバルデータブロックはデータエディタで編集できません。

プログラムエディタでプログラムと同様に編集します。

グローバルデータ内のデータはプログラム内と同様に変数に対する代入文の形で宣言します。

(a) 整数データの宣言

整数型データの宣言は次のようになります。

＜識別子＞＝＜整定数＞

例）Ｎ１＝１

注） プログラム内で整数型グローバル変数に実数値を代入した場合、小数点以下が切捨てられますので注意して下さい。

(b) 実数データの宣言

実数型データの宣言は次のようになります。

＜識別子＞＝＜実定数＞

例）Ｒ１＝１．０

(c) 配列データの宣言

配列型データの宣言は次のようになります。

ＤＩＭ ＜識別子＞（ｉ，ｊ・・・） ＡＳ ＜変数型＞

例） ２×３×４要素の整数型３次元配列

ＤＩＭ ＩＤＡＴ（２，３，４） ＡＳ ＩＮＴ

４×３要素の実数型２次元配列

ＤＩＭ ＲＤＡＴ（１０，５０） ＡＳ ＲＥＡＬ

５要素の位置型１次元配列

ＤＩＭ ＰＤＡＴ（５） ＡＳ ＰＯＩＮＴ

注） ＤＩＭ命令では配列型グローバルデータの型と要素数を指定するだけで初期値は不定です。初期値の設定は通常のグローバルデータと同様に整数型，実数型はグローバルデータブロックに、位置型，座標型，負荷型データは、データブロックに設定してください。

5.3.7 ロボットの動作速度

(1) 各モードでの速度

ロボットの動作速度は各モードで以下の様になります。

モ ー ド		速 度
自動運転	ＰＴＰ	０～１００％
	直線円弧補間	０～１００％
テスト運転	ＰＴＰ	０～ ２５％
	直線円弧補間	０～ ２５％

注） 仕様書最高速度を１００％とする。直線円弧補間は１m/s を１００％とする。

(2) 自動運転時の速度

自動運転，テスト運転における動作速度は以下のようになります。

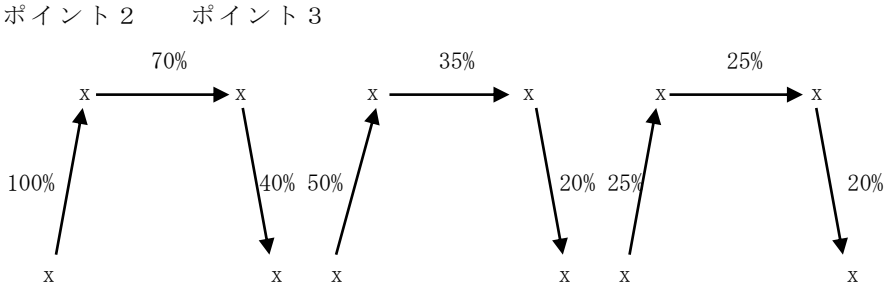
動作速度＝｛プログラム設定速度(1～100%) ×オーバライド速度(1～100%) ；

リミット速度を超えている動作はリミット速度｝

プログラム設定速度； ロボットプログラム´ＳＰＥＥＤ´命令で各動作の速度を設定します。

オーバーライド速度； 全ての動作を同じ割合で変化させます。
リミット速度 ； 設定されたリミットを超えた動作の速度のみを設定値に下げます。

プログラム設定 → オーバーライド50% → リミット25%



各動作の速度を指定できます。

全ての動作速度が指定の 50% になります。

ポイント１からポイント３までの動作速度が25% を超えているため、これを25% に下げます。

5.3.8 ロボットの加減速度

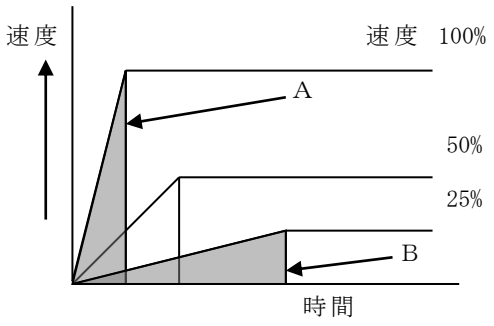
ロボットの動作加速度、減速度は以下の要因で変化します。

①動作モード： 直線補間（ＭＯＶＥＳ），ＰＴＰ（ＭＯＶＥ）

②速 度： ＳＰＥＥＤ命令，オーバーライド

③加減速命令： ＡＣＣＥＬ，ＤＥＣＥＬ

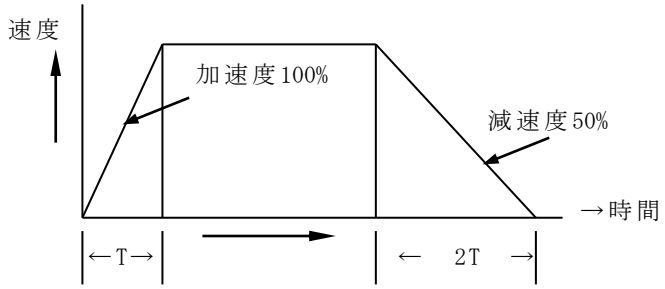
(1) 各動作モードでの最高加速度は機械強度、モータートルクから計算し、設定されています。また、スムーズな動作が得られるような速度カーブが使われています。



速度オーバーライドによる変化

速度の変化に対しては上図のように加速時間が速度の逆数に比例するように変化させます。これらによって加速中の移動量(図中の３角形Ａ，Ｂの面積)が速度に依らず一定となります。従ってオーバライド速度を変えてもロボットの動作経路はほとんど変わらず安全に調整運転を行えます。

(2) ＳＣＯＬ命令語によって加速度、減速度を別々に指定できます。
例えばハンドリングの際､減速度のみを下げてロボットを動作させ､停止時のハンド及びワークの振動を減少させます。



付録A 命令語一覧

動作制御命令

MOVE ＜位置＞ ［WITH節］	´同期動作
MOVES ＜位置＞ ［WITH節］	´直線補間動作
MOVEC ＜位置＞ ＜位置＞ ［WITH節］	´円弧補間動作
MOVEA ＜軸＞,＜絶対位置＞ ［WITH節］	´絶対単軸動作
MOVEI ＜軸＞,＜相対位置＞ ［WITH節］	´相対単軸動作
MOVEJ ＜位置＞ ＜アーチ定義＞	´相対単軸動作
READY	´機械原点への移動
DELAY ＜時間＞	´指定時間の停止
OPEN1, OPEN2	´ハンドを即時に開く
OPENI1, OPENI2	´ハンドを開く
CLOSE1, CLOSE2	´ハンドを即時閉じる
CLOSEI1, CLOSEI2	´ハンドを閉じる
RESUME	´中断した動作の再開

プログラム制御命令

PROGRAM ＜プログラム名＞	´プログラム開始
ON ＜監視条件＞ ［{BREAK｜PAUSE}］	´監視
DO ＜文＞	
IF ＜論理式＞ THEN ＜文＞ ［ELSE ＜文＞］	´条件判断
WAIT ＜論理式＞	´動作を待つ
IGNORE ＜監視条件＞	´監視の解除
GOTO ＜ラベル＞	´無条件分岐
GOTO(＜式＞) ＜ラベル＞ ［,＜ラベル＞］･･･	´式の値による分岐
RCYCLE	´サイクルリセット用ラベル
RETURN	´メインへ戻る
FOR ＜変数＞＝＜式＞TO＜式＞ ［STEP ＜式＞］	´動作の繰返し
NEXT ［＜変数＞］	´
STOP	´プログラム停止
REMARK ［＜注釈＞］	´注釈文
END	´プログラム終了
TASK(“プログラム名”)	´タスクプログラム開始
KILL(＜式＞)	´タスクプログラム終了
SWITCH	´タスクプログラム切り替え
LOADLIB ＜ファイル名＞	´ダイナミックライブラリの組み込み

入出力制御命令

DIN(＜信号名＞[, ＜信号名＞]･･･)	´入力信号の読込み
DOUT(＜信号名＞[, ＜信号名＞]･･･)	´信号の出力
PULOUT(＜信号名＞[, ＜信号名＞]･･･)	´信号のパルス出力
RESET ＜状態＞[, ＜状態＞]	´コントローラの状態のリセット
BCDIN(＜信号名＞, ＜信号長＞)	´信号のBCD入力
BCDOUT(＜信号名＞, ＜信号長＞, ＜式＞)	´信号のBCD出力
HEXIN(＜信号名＞, ＜信号長＞)	´信号のHEX入力
HEXOUT(＜信号名＞, ＜信号長＞, ＜式＞)	´信号のHEX出力
PRINT [{COM0｜COM1｜TP },] {＜文字列＞｜＜式＞} [, {＜文字列＞｜＜式＞}]･･･[, CR]	´通信データ出力
INPUT [{COM0｜COM1｜TP },] ＜変数＞, [＜変数＞]･･･	´通信データ入力

動作条件命令

CONFIG＝＜式＞	´姿勢
ACCUR＝＜式＞	´位置決め精度
ACCEL＝＜式＞	´加速時の加速度
DECEL＝＜式＞	´減速時の加速度
SPEED＝＜式＞	´速度
PASS＝＜式＞	´ショートカット動作のパラメータ
TORQUE＝{＜式＞, ＜式＞, ＜式＞, ＜式＞, ＜式＞}	´各軸トルク
GAIN＝{＜式＞, ＜式＞, ＜式＞, ＜式＞, ＜式＞}	´各軸ゲイン
SETGAIN(＜整数＞, ＜整数＞, ＜整数＞, ＜整数＞, ＜整数＞)	´動作に同期した各軸ゲイン
ENABLE ＜スイッチ＞[, ＜スイッチ＞]･･･	´システムスイッチ入
DISABLE ＜スイッチ＞[, ＜スイッチ＞]･･･	´システムスイッチ切
PAYLOAD＝{＜質量＞, ＜重心オフセット＞}	´負荷データ設定
FREELOAD	´負荷データ解除

パレタイズ命令

INITPLT(＜パレット番号＞, ＜i＞, ＜j＞, ＜k＞)	´パレットの初期化
MOVEPLT(＜パレット番号＞, ＜要素番号＞ , ＜X＞, ＜Y＞, ＜Z＞, ＜C＞)	´パレットの指定位置に移動

算術演算命令

SIN(＜式＞)	´正弦
COS(＜式＞)	´余弦
TAN(＜式＞)	´正接
ASIN(＜式＞)	´逆正弦
ACOS(＜式＞)	´逆余弦
ATAN(＜式＞)	´逆正接
ATAN2(＜式＞, ＜式＞)	´逆正接
SQRT(＜式＞)	´平方根
ABS(＜式＞)	´絶対値
SGN(＜式＞)	´符号の取出し
INT(＜式＞)	´整数化
REAL(＜式＞)	´実数化
LN(＜式＞)	´自然対数
＜式＞ MOD ＜式＞	´剰余
LOG10(＜式＞)	´常用対数
EXP(＜式＞)	´eのべき乗
＜論理式＞ AND ＜論理式＞	´論理積
＜論理式＞ OR ＜論理式＞	´論理和
NOT ＜論理式＞	´否定
HERE	´現在位置
DEST	´目標位置
POINT(＜式＞, ＜式＞, ＜式＞, ＜式＞, ＜式＞, ＜姿勢＞)	´位置型データの生成
TRANS(＜式＞, ＜式＞, ＜式＞, ＜式＞)	´座標型データの生成

動作状態命令

MOTION	´動作の実行量
MOTIONT	´動作の実行時間
REMAIN	´動作の残量
REMAINT	´動作の残り時間
TIMER	´タイマー
MODE	´システムの運転モード
TOOL	´ツール座標系
BASE	´ベース座標系
WORK	´ワーク座標系

付録B 予約語一覧

ABS	ACCEL	ACCUR	ACOS
ACTUAL	AND	AS	ASIN
ATAN	ATAN2	BASE	BCDIN
BCDOUT	BREAK	CLOSE1	CLOSE2
CLOSEI1	CLOSEI2	CNVCOUNT1	CNVCOUNT2
CNVSTAT	CNVTRIP1	CNVTRIP2	CNVVELOC1
CNVVELOC2	COARSE	COM0	COM1
CONFIG	CONT	CONV	COS
CR	CYCLE	DATA	DECEL
DELAY	DEST	DIM	DIN
DISABLE	DO	DOUT	ELSE
ENABLE	END	EXP	ERROR
FINE	FOR	FREE	FREELOAD
GAIN	GLOBAL	GOTO	GOTO ()
HERE	HEXIN	HEXOUT	HOST
IF	IGNORE	INITPLT	INPSTSTP
INPSTSIP1	INPSTSIP2	INPSTSCM1	INPSTSCM2
INPTMOTP	INPTMOIP1	INPTMOIP2	INPTMOCM1
INPTMOCM2	INPUT	INT	IP1
IP2	IPCLOSE	IPOPEN	IP0STATUS
IP1STATUS	IP2STATUS	IP3STATUS	KILL
LATCH	LATCHPSN1～8	LATCHSIG1～8	LATCHTRG1～8
LEFTY	LN	LOADLIB	LOG10
MAXTASK	MOD	MODE	MONEND
MONSTART	MONTRIP1	MONTRIP2	MOTION
MOTIONT	MOVE	MOVEA	MOVEC
MOVEI	MOVEJ	MOVEPLT	MOVES
MOVESI	MOVESYNC	MSPEED	NAME
NEXT	NOT	NOWAIT	OFF
OFFGAIN	ON	ONGAIN	OPEN1
OPEN2	OPENI1	OPENI2	OR
OVERRIDE	PAI	PASS	PAUSE
PAYLOAD	PLCDATAR1～8	PLCDATAW1～8	PLCINPUT
PLCPRINT	POINT	PRINT	PROGRAM
PRT	PSNCMD	PSNCMDJ	PSNCMDW
PSNFBK	PSNFBKJ	PSNFBKW	PULOUT
QUANTUM	RCYCLE	READY	REAL
REMAIN	REMAINT	REMAKE	REMARK
RESET	RESTORE	RESUME	RETURN
RIGHTY	SAVEEND	SAVEF1～4	SAVEI1～4

ＳＥＧＭＥＮＴ	ＳＥＴＧＡＩＮ	ＳＧＮ	ＳＩＮ
ＳＬＯＷＤＯＷＮ	ＳＬＷＳＰＤ	ＳＭＯＯＴＨ	ＳＰＥＥＤ
ＳＱＲＴ	ＳＴＥＰ	ＳＴＯＰ	ＳＷＩＴＣＨ
ＳＹＮＣ	ＴＡＮ	ＴＡＳＫ	ＴＨＥＮ
ＴＩＤ	ＴＩＭＥＲ	ＴＯ	ＴＯＯＬ
ＴＯＲＱＵＥ	ＴＰ	ＴＲＡＮＳ	ＵＮＳＹＮＣ
ＶＣＦＵＮＣ	ＶＣＭＡＲＫ	ＶＣＮＥＸＴ	ＶＣＰＯＳ
ＶＣＰＳＷＬＤ１～４	ＶＣＳＥＥＫ	ＶＣＳＥＬ１～３	ＶＣＶＴＹＰＥ１～４
ＶＣＷＳＴＡＴ１～３	ＷＡＩＴ	ＷＩＴＨ	ＷＯＲＫ
ＸＩＮ			

付録C　ライブラリファイル（SCOL．LIB）の内容

システムに標準で用意しているライブラリファイルの内容は、次のようになります。
なお、ライブラリファイルの内容はお客様ごとに、多少異なる場合があります。
ライブラリファイルを変更しても、SELECTを再度実行しないと変更内容が
現在選択中のプログラムに反映されませんのでご注意ください。

```
'Copyright(C) 2008 by TOSHIBA MACHINE CO.,LTD
'ALL RIGHTS RESERVED.
'TS3000 SCOL SUBPROGRAM LIBRALY
PROGRAM READY
MOVEA 3,0
MOVEA 4,0
MOVEA 2,0
MOVEA 1,0
MOVEA 5,0
END
PROGRAM OPEN1      'OPEN HAND-1
WAIT MOTION>=100
DOUT(203,-204)
END
PROGRAM CLOSE1     'CLOSE HAND-1
WAIT MOTION>=100
DOUT(-203,204)
END
PROGRAM OPENI1     'OPEN HAND-1 IMMEDIATE
DOUT(203,-204)
END
PROGRAM CLOSEI1    'CLOSE HAND-1 IMMEDIATE
DOUT(-203,204)
END
PROGRAM OPEN2      'OPEN HAND-2
WAIT MOTION>=100
DOUT(201,-202)
END
PROGRAM CLOSE2     'CLOSE HAND-2
WAIT MOTION>=100
DOUT(-201,202)
END
PROGRAM OPENI2     'OPEN HAND-2 IMMEDIATE
DOUT(201,-202)
```

```
END
PROGRAM CLOSEI2   'CLOSE HAND-2 IMMEDIATE
DOUT(-201,202)
END
PROGRAM FREELOAD 'FREE PAYLOAD
PAYLOAD={0,0}
END
PROGRAM SETGAIN(J_0,J_1,J_2,J_3,J_4) 'SET GAIN VARIABLE
WAIT MOTION>=100
GAIN={J_0,J_1,J_2,J_3,J_4}
MOVE HERE WITH PASS=100
MOVE HERE
END
PROGRAM ONGAIN(J_1,J_2,J_3,J_4,J_5)
J_6=0
J_7=0
J_8=0
J_9=0
J_0=0
IF J_1==0 THEN J_6=GAIN.1 ELSE J_6=1
IF J_2==0 THEN J_7=GAIN.2 ELSE J_7=1
IF J_3==0 THEN J_8=GAIN.3 ELSE J_8=1
IF J_4==0 THEN J_9=GAIN.4 ELSE J_9=1
IF J_5==0 THEN J_0=GAIN.5 ELSE J_0=1
WAIT MOTION>=100
GAIN={J_6,J_7,J_8,J_9,J_0}
MOVE HERE WITH PASS=100
MOVE HERE
RETURN
END
PROGRAM OFFGAIN(J_1,J_2,J_3,J_4,J_5)
J_6=0
J_7=0
J_8=0
J_9=0
J_0=0
IF J_1==0 THEN J_6=GAIN.1 ELSE J_6=0
IF J_2==0 THEN J_7=GAIN.2 ELSE J_7=0
IF J_3==0 THEN J_8=GAIN.3 ELSE J_8=0
IF J_4==0 THEN J_9=GAIN.4 ELSE J_9=0
IF J_5==0 THEN J_0=GAIN.5 ELSE J_0=0
```

```
WAIT MOTION>=100
GAIN={J_6,J_7,J_8,J_9,J_0}
MOVE HERE WITH PASS=100
MOVE HERE
RETURN
END
```

（余白）

付録D　算術演算命令の計算範囲

命　　令	引数X、Yの範囲	結果Zの範囲
SIN(X)	(＊)	－1≦Z≦1
COS(X)	(＊)	－1≦Z≦1
TAN(X)	(＊)	(＊)
ASIN(X)	－1≦X≦1	－90° ≦Z≦90°
ACOS(X)	－1≦X≦1	0≦Z≦180°
ATAN(X)	(＊)	－90° ＜Z＜90°
ATAN2(X, Y)	Y≠0	－180° ＜Z＜180°
SQRT(X)	X≧0	Z≧0
ABS(X)	(＊)	Z≧0
SGN(X)	(＊)	Z＝－1, 0, 1
INT(X)	(＊)	(＊)
REAL(X)	(＊)	(＊)
LN(X)	X＞0	(＊)
X　MOD　Y	Y≠0	(＊)
LOG10(X)	X＞0	(＊)
EXP(X)	(＊)	Z＞0
備　　考	(＊)は、コントローラが取扱う範囲で特に制約のないことを表わす。	

付録E　記号の読み方

ロボットで使用するキー・記号は以下のように読みます。(アルファベット、数字は省略)

[F1]～[F5]	:	ファンクションキー、エフ1～エフ6
[ESC]	:	エスケープキー
[INS]	:	挿入キー
[DEL]	:	削除キー
[BS]	:	バックスペースキー
[{]	:	左中括弧
[}]	:	右中括弧
[[]	:	左大括弧
[]]	:	右大括弧
[ERROR]	:	エラー表示キー
[UTILITY]	:	ユーティリティキー
[！]	:	感嘆符(エクスクラメーションマーク)
[;]	:	セミコロン
[:]	:	コロン
[´]	:	アポストロフィー
[%]	:	パーセント
[^]	:	アクサンシルコンフレックス
[＆]	:	アンパサンド
[”]	:	引用符(ダブルコーテーションマーク)
[(]	:	左括弧
[)]	:	右括弧
[Alt]	:	オルトキー(オルタネイティブキー)
[＋]	:	プラス
[－]	:	マイナス
[／]	:	スラッシュ(斜線)
[＊]	:	アスタリスク(星印)
[　]	:	スペース
[＜]	:	不等号(より小)
[＞]	:	不等号(より大)
[,]	:	コンマ
[.]	:	ピリオド
[？]	:	疑問符(クエスチョンマーク)

［＝］	：	等号
［EXE］	：	実行キー
［↑］	：	(上向き)カーソルキー
［↓］	：	(下向き)カーソルキー
［←］	：	(左向き)カーソルキー
［→］	：	(右向き)カーソルキー

付録Ｆ　コンパイルエラー、およびコンパイルワーニングメッセージ

以下にティーチペンダントに表示するオペレーションエラーメッセージの一覧表を示します。

コンパイルエラー一覧表

エラーＮｏ.	エラー内容
200	システムが実行可能状態にありません
201	作業用メモリが確保できません
202	コマンドが不正です
205	数値定数が異常です
206	文字列定数が異常です
207	使用不可能な文字が検出されました
208	代入式に誤りがあります
209	文法式に誤りがあります
210	文法に誤りがあります
211	[GLOBAL]変数が[GOTO]ラベルに使用されています
212	[GLOBAL]宣言内でベクトル変数の初期化は行えません
213	[PROGRAM]文が行の先頭にありません
214	[RETURN]命令の位置が異常です
215	[PROGRAM]文と[END]文の対応がとれていません
216	[PROGRAM]内で[DATA]宣言が行われています
217	[PROGRAM]内で[GLOBAL]宣言が行われています
218	[GLOBAL]文が行の先頭にありません
219	[GLOBAL]文と[END]文の対応がとれていません
220	[GLOBAL]内で[PROGRAM]宣言が行われています
221	[GLOBAL]内で[DATA]宣言が行われています
222	条件監視(ON ～ DO ～)の宣言が５０個を越えました
223	[GOTO]ラベルが二重定義されています
224	[DATA]内で[PROGRAM]宣言が行われています
225	[DATA]内で[GLOBAL]宣言が行われています
226	[GLOBAL]宣言内以外では使用できません
227	配列の次元数が正しくありません
228	[IF]文と[THEN]文及び[ELSE]文の対応がとれていません
229	[FOR]文と[NEXT]文の対応がとれていません
230	[FOR～NEXT]文が１２７重を超えました
231	予約語が多重定義されようとしました
232	監視条件が不正です
233	式に誤りがあります
234	演算子が不正です
235	[RCYCLE]ラベルは[MAIN]関数以外は使用できません
236	ベクトル変数は使用できません

エラーＮｏ.	エラー内容
237	ベクトル変数には使用できません
238	要素が多く指定されています
239	入出力命令は関数引数には使用できません
240	括弧の対応がとれていません
242	[RCYCLE]ラベルは[MAIN]関数以外は使用できません
243	[FOR]ループ内へのジャンプはできません
244	ラベルが行の先頭でありません
245	指定のラベルは存在しません
246	[PROGRAM]データがありません
247	[THEN]文又は[ELSE]文に不等式は使用できません
248	予約語に誤りがあります
249	関数の実引数と仮引数が一致しません
250	関数の引数は１０個以上指定できません
252	[GLOBAL]変数は関数名では使用できません
253	予約語は関数名では使用できません
254	関数宣言に誤りがあります
255	関数名は宣言済みです
256	指定の関数は、存在しません
257	[GLOBAL]変数名が未定義になっています
258	[GOTO]ラベルが変数名で使用されています
259	[PROGRAM]宣言内ではありません
260	[GLOBAL][DATA]では指定の予約語は使用できません
261	[GLOBAL][DATA]変数は代入又は再宣言できません
262	指定の変数又は定数は使用できません
263	論理演算子及び不等式は使用できません
264	共通しない型変数を使用しました
265	宣言中です[END]文が有りません
266	[GLOBAL]ブロック先頭以外では使用できません
267	バックアップ変数個数が多すぎます
268	配列変数の総数は 11,000 までしか宣言できません
269	POINT 教示点は 1,500 までしか設定できません
270	POINT 以外の教示点は 500 までしか設定できません
271	信号数が多すぎます
272	PASS と SMOOTH の二重定義はできません

※ エラーＮｏ.２０１の詳細は、次頁を参照して下さい。

エラーＮｏ．２０１のコンパイルエラーの詳細情報内容を下記に示します。

番号	内容	最大値	詳細説明
１	トークンコード変換バッファ	75500	
２	グローバル変数	500	グローバル変数定義数は最大 500 個
３	オート変数	600	オート変数定義数は最大 600 個
４	関数	128	関数は最大 128 個
５	ＧＯＴＯラベル	1000	GOTO ラベルは最大 1000 個
６	ＷＯＲＫ座標教示	50	WORK 座標系は最大 50 個
７	変数名格納バッファ	10000	
８	プログラム行数	6000	GLOBAL|PROGRAM|DATA ブロックの合計は最大 6000 行(空行も含む)
９	未定義変数情報テーブル	100	未使用
１０	配列変数	100	配列変数定義数は最大 100 個
１１	グローバル整数型変数宣言	100	整数型のグローバル変数定義数は最大 100 個
１２	グローバル実数型変数宣言	100	実数型のグローバル変数定義数は最大 100 個
１３	グローバル負荷型変数宣言	100	負荷型のグローバル変数定義数は最大 100 個
１４	グローバル座標系型変数宣言	100	座標系型のグローバル変数定義数は最大 100 個
１５	グローバル位置型変数宣言	1500	位置型のグローバル変数定義数は最大 1500 個
１６	オート整数型変数宣言	500	整数型のオート変数定義数は最大 500 個
１７	オート実数型変数宣言	500	実数型のオート変数定義数は最大 500 個
１８	オート負荷型変数宣言	100	負荷型のオート変数定義数は最大 100 個
１９	オート座標系型変数宣言	100	座標系型のオート変数定義数は最大 100 個
２０	オート位置型変数宣言	1000	位置型のオート変数定義数は最大 1000 個
２１	オート未定義変数宣言	100	
２２	関数呼先情報	300	関数コールは最大 300 回(平均して一関数は 6 ヶ所からコールされる)
２３	関数呼元情報	500	関数呼先情報(22)でリミットされる。
２４	関数引数情報	100	関数引数は最大 100 個(平均して一関数は引数 2 個持てる)
２５	ＧＯＴＯ呼元情報	1000	GOTO 命令は最大 1000 個
２６	オート整数型定数	2000	PROGRAM ブロック内で使用される整数型定数は最大 2000 個
２７	オート実数型定数	2000	PROGRAM ブロック内で使用される実数型定数は最大 2000 個
２８	オート負荷型定数	200	PROGRAM ブロック内で使用される負荷型定数は最大 100(=200÷2)個
２９	オート座標系型定数	100	PROGRAM ブロック内で使用される座標系型定数は最大 25(=100÷4)個
３０	オート位置型定数	2000	PROGRAM ブロック内で使用される位置型定数は最大 333(=2000÷6)個
３１	オート文字列情報	1000	PROGRAM ブロック内で使用される文字列情報は最大 1000 個
３２	グローバル整数型定数	100	GLOBAL、及び DATA ブロック内で使用される整数型定数は最大 100 個
３３	グローバル実数型定数	100	GLOBAL、及び DATA ブロック内で使用される実数型定数は最大 100 個
３４	グローバル負荷型定数	100	GLOBAL、及び DATA ブロック内で使用される負荷型定数は最大 50(=100÷2)個
３５	グローバル座標系型定数	100	GLOBAL、及び DATA ブロック内で使用される座標系型定数は最大 25(=100÷4)個
３６	グローバル位置型定数	10000	GLOBAL、及び DATA ブロック内で使用される位置型定数は最大 1666(=10000÷6)個
３７	グローバル整数型変数使用数	1000	整数型グローバル変数使用合計数は最大 1000 回
３８	グローバル実数型変数使用数	200	実数型グローバル変数使用合計数は最大 200 回
３９	グローバル負荷型変数使用数	100	負荷型グローバル変数使用合計数は最大 100 回
４０	グローバル座標系型変数使用数	100	座標系型グローバル変数使用合計数は最大 100 回
４１	グローバル位置型変数使用数	3000	位置型グローバル変数使用合計数は最大 3000 回
４２	オート整数型変数使用数	3001	整数型オート変数使用合計数は最大 3001 回
４３	オート実数型変数使用数	2001	実数型オート変数使用合計数は最大 2001 回
４４	オート負荷型変数使用数	100	負荷型オート変数使用合計数は最大 100 回
４５	オート座標系型変数使用数	100	座標系型オート変数使用合計数は最大 100 回
４６	オート位置型変数使用数	2000	位置型オート変数使用合計数は最大 2000 回
４７	オート未定義変数使用数	100	未使用
４８	ＧＯＴＯ呼元インデックス数	2000	GOTO 呼元情報数でリミットされる。
４９	関数呼元インデックス数	1000	関数呼備先情報(22)でリミットされる。
５０	関数引数インデックス数	200	関数引数情報(24)でリミットされる。
５１	配列変数使用数	200	配列変数は最大 200 個定義可能
５２	ＲＥＳＴＯＲＥ数	100	RESTORE 命令記述数は最大 100 回
５３	配列変数用数値格納数	17000	初期化配列変数の要素数合計は最大 17000(2×3 次元の位置型配列では 36=2×3×6 となる)
１００	インデックス情報数	5000	変数、定数、関数、ラベル等使用数合計は最大 5000 個
１０１	数値情報数	25000	変数、定数、関数、ラベル等定義数合計は 25000 個
１０２	コード生成数	399800	
２００	インタプリタ実行情報エリア		

※ ・ライブラリファイル(ＳＣＯＬ．ＬＩＢ)が存在する場合は、ライブラリファイルで使用されている数も加算されます。

・制限に関しては、次のＳＣＯＬプログラムの制限事項を参照して下さい。

SCOLプログラムの制限事項を下記に示します。

項目	制限数(1ファイル内)	備考
プログラム行数	5, 500	
関数	49	
関数の引数	10	
GOTOラベル数	999	但し、1関数内でのラベル宣言及び GOTO ラベル(同一 GOTO ラベルは複数あっても1個)は599個までです。
グローバル整数型変数	99	
グローバル実数型変数	99	
グローバル負荷型変数	48	
グローバル座標型変数	23	
グローバル位置型変数	499	
オート整数型変数	499	
オート実数型変数	499	
オート負荷型変数	99	
オート座標型変数	99	但し、1ファイル内での定数設定(プログラム内での定数設定)は24個までが可能です。
オート位置型変数	999	但し、1関数内では599個までが可能です。 なお、1ファイル内での定数設定(プログラム内での定数設定)は333個までが可能です。
オート変数及びラベル情報	599(1関数内)	1関数内におけるオート変数とGOTOラベル数の総数の制限値であり、1ファイル内の制限値ではありません。
配列変数	99	
配列変数の総要素数	11, 000	
配列変数の位置型データ(POINT)の教示点数	1, 200	
配列変数の位置型データ(POINT)以外の教示点数	500	
FOR～NEXTのネスト	127	
条件監視(ON～DO～)の同時設定数	10	
条件監視(ON～DO～)の宣言数	50	

注)1ファイルとは、SCOL．LIBファイルを含めます。

付録G　ダイナミックリンクライブラリ

付録G－1　パレタイズライブラリ

ライブラリ名	PALLET. LIB
機能	パレタイズ命令のライブラリです。 パレット原点、I点、J点、K点の、4点を教示することで最大(i×j×k) の3次元パレタイズが可能です
命令	ＩＮＩＴＰＬＴ（＜パレット番号＞，＜ｉ＞，＜ｊ＞，＜ｋ＞） パレット番号で示されたパレットをｉ×ｊ×ｋの３次元パレットとして初期化します。 **i**：パレット原点～Ｉ点間の要素数 **j**：パレット原点～Ｊ点間の要素数 **k**：パレット原点～Ｋ点間の要素数

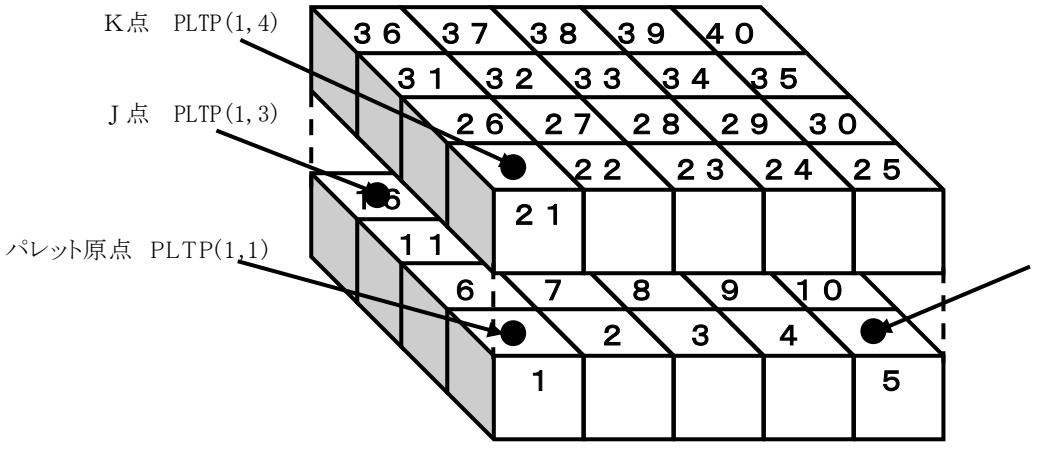
ＭＯＶＥＰＬＴ（＜パレット番号＞，＜要素番号＞，Ｘ，Ｙ，Ｚ，Ｃ）

パレット番号で示されたパレットの要素番号の位置にＸ、Ｙ、Ｚ、Ｃのオフセットを加算した位置に移動します。

Ｘ，Ｙ，Ｚ，Ｃのオフセット値は省略できません（オフセット無しの場合０を記述する）

用語の説明

例　パレット番号１、５×４×２の３次元パレット



パレット番号：パレット番号はそのプログラムで使うパレットに対し、1から順に与えます。

教示点　　：上図の原点、I点、J点、K点の4点がこのパレットの教示点です。

教示点名は<u>固定</u>で、<u>PLTP(＜パレット番号＞，1～4)です。</u>

要素番号　：パレットの構成要素に対し自動的に付加される番号で、上図の5×4×2のパレットでは、各要素に対し1～40の番号が与えられます。

パレタイズ命令では、パレット番号、要素番号を指定することで、目的の位置にロボットを移動させることが出来ます

ライブラリの組み込み

PALLET. LIBを使う為には、以下の①、②の命令が必要です。

① ユーザープログラムのGLOBAL部で、**ライブラリの組み込み**宣言が必要です。

LOADLIB PALLET. LIB

ライブラリの組み込み宣言

② ユーザープログラムのGLOBAL部で、ライブラリで使う**グローバル変数**の宣言が必要です。変数名は固定で、PLTPです。

DIM PLTP(＜パレット数＞, 7) AS POINT

パレット数は、1以上の任意の値、最大値は、そのプログラムの教示点数や使われている配列数により変化します。**7**は定数でライブラリで使う変数領域を確保します。

このグローバル変数は、PALLET. LIBと、教示点や計算値のやり取りに使います。

```
GLOBAL
    LOADLIB PALLET.LIB          ①ライブラリの組み込み
    DIM PLTP(2,7) AS POINT      ②グローバル変数の宣言
END
PROGRAM MAIN
・・・・・・・・・・
  省略
・・・・・・・・・・
END
```

解説注意

① パレット

パレットはX－Y平面に水平に置かれているものとする。

（パレットの傾斜には対応していません）

② 教示と有効データ

教示はＰＬＴＰ（ｎ、１）～ＰＬＴＰ（ｎ、４）の４点の教示を行います。

（ｎはパレット番号１～ｎ）

パレット原点 教示データは、X，Y，Z，C，T全ての座標が有効であり、このPLTP(n,1)のデータにシフト量を加算し、移動位置を算出する。

I 点 PLTP(n,2)
J 点 PLTP(n,3)
教示データは、X，Yのみ有効であり、X、Y方向のシフト量算出の為に用いる

K 点 PLTP(n,4) 教示データはZのみ有効でありパレット原点に対するZ方向のシフト量を算出する為に用いる。

1元パレットの場合には **J 点** PLTP(n,3)、**K 点** PLTP(n,4)示を省略可能
2次元パレットの場合には **K 点** PLTP(n,4)の教示を省略可能

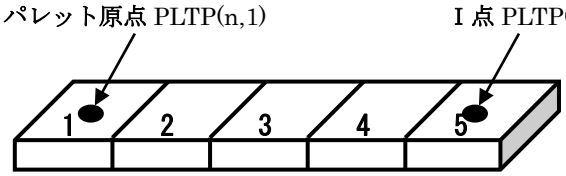
教示点名称は変更できない。

③ LOADLIB命令でPALLET.LIBを読み込む場合、PALLET.LIBで使われている変数名称（INITPLT****、MOVEPLT**** *は任意の文字）はユーザープログラムで変数名、教示点名として使用できません。

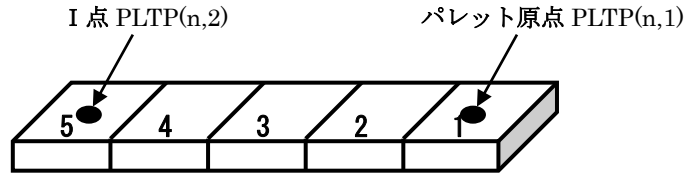
④ パレットの教示方法と要素番号

要素番号は、ＩＮＩＴＰＬＴ命令により、自動的に付けられますが、同じパレットでも**教示順序により番号が変化**します。

・１次元パレット

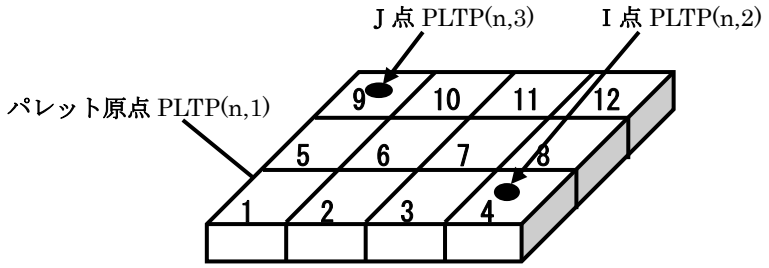


ＩＮＩＴＰＬＴ（ｎ，５，１，１）　　ｎは任意のパレット番号

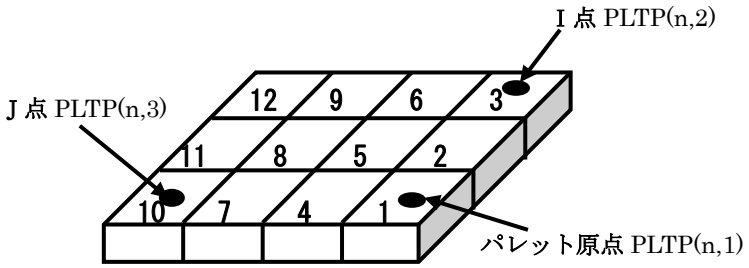


ＩＮＩＴＰＬＴ（ｎ，５，１，１）　　ｎは任意のパレット番号

・２次元パレット

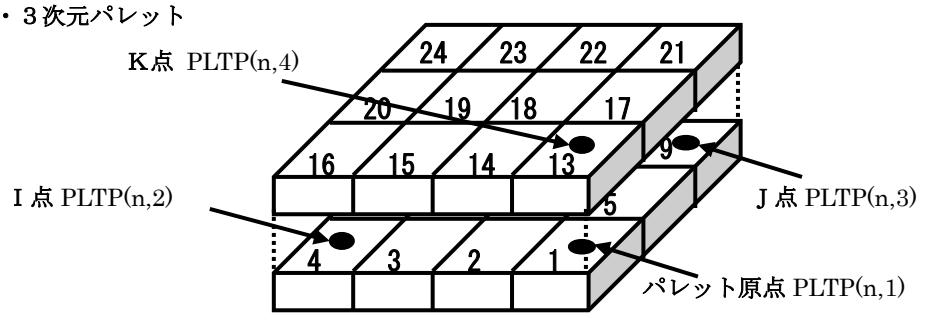


ＩＮＩＴＰＬＴ（ｎ，４，３，１）　　ｎは任意のパレット番号



ＩＮＩＴＰＬＴ（ｎ，３，４，１）　　ｎは任意のパレット番号

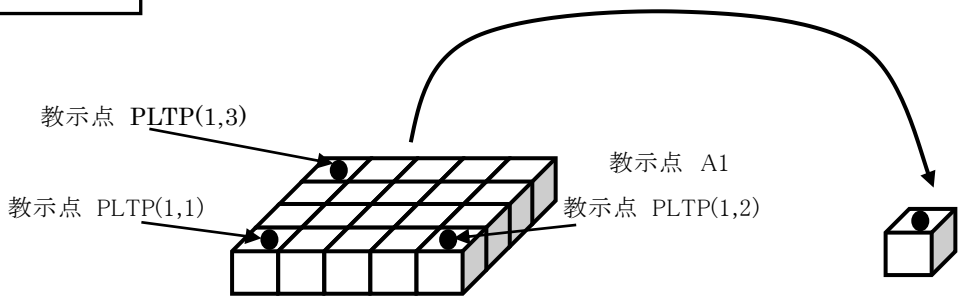
・３次元パレット



ＩＮＩＴＰＬＴ（ｎ，４，３，２）　　ｎは任意のパレット番号

サンプル
プログラム

① パレットからA1点に部品を供給する




```
GLOBAL
   LOADLIB PALLET.LIB
     DIM PLTP(1,7) AS POINT
END

PROGRAM PALLET
     INITPLT(1,5,4,1)
     OPEN1
     FOR I=1 TO 20 STEP 1
          MOVEPLT(1,I,0,0,50,0)
          MOVEPLT(1,I,0,0,0,0)
          CLOSE1
          MOVEPLT(1,I,0,0,50,0)
          MOVE A1+POINT(0,0,50)
          MOVE A1
          OPEN1
          MOVE A1+POINT(0,0,50)
     NEXT I
END

DATA
     POINT   A1         =  650.000,   -0.010,  187.140,    2.457,    0.000  / LEFTY
     POINT   PLTP(1,1)  =  203.346,  390.635,   94.252,   30.261,    0.000  / LEFTY
     POINT   PLTP(1,2)  =  357.548,  503.825,   94.252,   30.261,    0.000  / LEFTY
     POINT   PLTP(1,3)  =  337.299,  207.424,   94.252,   30.261,    0.000  / LEFTY
     POINT   PLTP(1,4)  =  337.299,  207.424,   94.252,   30.261,    0.000  / LEFTY
END
```

PALLET.LIB の内容

標準でシステムに付加されるPALLET.LIBの内容は以下の内容です。

```
'*********************************************************
'*      TS3000 Dynamic Link Liblary                      *
'*                                                       *
'*      file name   : PALET.LIB                          *
'*      function    : PALLETIZE                          *
'*      comannd     : INITPLT(PALLET_NO,I,J,K)           *
'*                  : MOVEOLT(PALLET_NO,POZITION_NO,X,Y,Z,C) *
'*                                                       *
'*      Copyright(C) 2008 by TOSHIBA MACHINE CO.,LTD     *
'*-------------------------------------------------------*
'*   08/08/28New                                         *
'*                                                       *
'*********************************************************
PROGRAM INITPLT (INITPLTNO,INITPLTI,INITPLTJ,INITPLTK)
'
      INITPLT1P = PLTP(INITPLTNO,1)
      INITPLT2P = PLTP(INITPLTNO,2)
      INITPLT3P = PLTP(INITPLTNO,3)
      INITPLT4P = PLTP(INITPLTNO,4)
      PLTP(INITPLTNO,5) = POINT(INITPLTI,INITPLTJ,INITPLTK)
INITPLT010:
      IF INITPLTI < 1 THEN GOTO INITPLTERR
      IF INITPLTI < 2 THEN GOTO INITPLT015
      INITPLT5I = INITPLTI - 1
      INITPLTXX = (INITPLT2P.X - INITPLT1P.X) / INITPLT5I
      INITPLTXY = (INITPLT2P.Y - INITPLT1P.Y) / INITPLT5I
      GOTO INITPLT020
INITPLT015:
      INITPLTXX = 0
      INITPLTXY = 0
INITPLT020:
      IF INITPLTJ < 1 THEN GOTO INITPLTERR
      IF INITPLTJ < 2 THEN GOTO INITPLT025
      INITPLT5J = INITPLTJ - 1
      INITPLTYX = (INITPLT3P.X - INITPLT1P.X) / INITPLT5J
      INITPLTYY = (INITPLT3P.Y - INITPLT1P.Y) / INITPLT5J
      GOTO INITPLT030
INITPLT025:
```

```
        INITPLTYX = 0
        INITPLTYY = 0
INITPLT030:
        IF INITPLTK < 1 THEN GOTO INITPLTERR
        IF INITPLTK < 2 THEN GOTO INITPLT035
        INITPLT5K = INITPLTK - 1
        INITPLTZZ   = (INITPLT4P.Z - INITPLT1P.Z) / INITPLT5K
        GOTO INITPLT040
INITPLT035:
        INITPLTZZ = 0
INITPLT040:
        PLTP(INITPLTNO,6) =   POINT(INITPLTXX,INITPLTXY,INITPLTZZ)
        PLTP(INITPLTNO,7) =   POINT(INITPLTYX,INITPLTYY,INITPLTZZ)
        GOTO INITPLTEND
INITPLTERR:
        PRINT "ERR !!   ELEMENT IS TOO SMALL."
        STOP
INITPLTEND:
END
'-----------------------------------------------------
PROGRAM MOVEPLT
(MOVEPLTNO,MOVEPLTPSN,MOVEPLTX,MOVEPLTY,MOVEPLTZ,MOVEPLTC)
        MOVEPLTI = 0
        MOVEPLTJ = 0
        MOVEPLTK = 0
        MOVEPLTPS1 = MOVEPLTPSN -1
        MOVEPLT1P = PLTP(MOVEPLTNO,1)
        MOVEPLT5P = PLTP(MOVEPLTNO,5)
        MOVEPLT6P = PLTP(MOVEPLTNO,6)
        MOVEPLT7P = PLTP(MOVEPLTNO,7)
        MOVEPLTA = MOVEPLT5P.X * MOVEPLT5P.Y
        MOVEPLTB = MOVEPLTPS1 MOD MOVEPLTA
        MOVEPLTMAX = MOVEPLTA *   MOVEPLT5P.Z
        IF 1 > MOVEPLTPSN THEN GOTO MOVEPLTER2
        IF MOVEPLTMAX < MOVEPLTPSN THEN GOTO MOVEPLTER3
        MOVEPLTI = MOVEPLTB MOD   MOVEPLT5P.X
        MOVEPLTJ = INT(MOVEPLTB / MOVEPLT5P.X)
        MOVEPLTK = INT(MOVEPLTPS1 / MOVEPLTA )
        MOVEPLTXXX = MOVEPLTI * MOVEPLT6P.X   + MOVEPLTJ * MOVEPLT7P.X + MOVEPLTX
        MOVEPLTYYY = MOVEPLTI * MOVEPLT6P.Y   + MOVEPLTJ * MOVEPLT7P.Y + MOVEPLTY
        MOVEPLTZZZ = MOVEPLTK * MOVEPLT6P.Z   + MOVEPLTZ
```

```
        MOVE MOVEPLT1P+ POINT(MOVEPLTXXX,MOVEPLTYYY,MOVEPLTZZZ,MOVEPLTC,0)
        GOTO MOVEPLTEND
MOVEPLTER2:
        PRINT "ERR !!   ELEMENT NO. IS TOO SMALL."
        STOP
MOVEPLTER3:
        PRINT "ERR !!   ELEMENT NO. IS TOO LARGE."
        STOP
MOVEPLTEND:
END
```

付録H　先読み停止を実行するSCOLプログラム言語について

先読み停止を実行する命令語を下記に示します。

・ PRINT
・ WAIT
・ ON ～ DO
・ IGNORE
・ DOUT
・ DIN
・ XIN （コンベアオプション）
・ BCDIN
・ BCDOUT
・ PULOUT
・ MOVE
・ MOVEA
・ MOVEI
・ MOVES
・ MOVEC
・ MOVEJ
・ DELAY
・ LATCH
・ SYNC （コンベアオプション）
・ UNSYNC （コンベアオプション）
・ STOP
・ RETURN
・ END