デンソーロボット

# プログラミングマニュアルⅠ
# 基礎知識とコマンド

Copyright © 2009-2011 DENSO WAVE INCORPORATED
All rights reserved.

この取扱説明書の著作権は、株式会社デンソーウェーブにあります。

本書に掲載されている会社名や製品は、一般に各社の商標または登録商標です。

仕様は予告なく変更することがあります。

# はじめに

デンソーロボットをお買い上げいただき、誠にありがとうございます。

この製品は当社の技術を結集した、高速・高精度でかつ高度な機能を備えた「組立て用ロボット」です。

ご使用にあたっては、本書をよく読み理解のうえ、安全で効率的な運用をお願いします。

## 本書が扱う対象製品

■RC7M 型コントローラ搭載

- 垂直多関節型ロボット　V＊シリーズ
- 水平多関節型ロボット　H＊-G シリーズ
- 直角座標型ロボット　XYC-4G シリーズ
- 組込型ロボット　XR-G シリーズ

■オプション品

- 視覚装置　μVision シリーズ

**注**：ロボットコントローラのバージョンはコントローラの上面に貼られている「コントローラ設定表」のソフトウェアVer. 欄に記されています。
またティーチングペンダントからは、[基本画面]-[F6 設定]-[F6 保守.]-[F2 バージョン]で表示されるROMバージョン欄から確認できます。

## お願い

ご使用の前に、「安全にご使用いただくために」をお読みいただき、正しく安全にデンソーロボットをお使いください。

**本書の構成**

本書の構成は、以下のようになっております。

## 第１部　プログラムデザイン

### 第１章　プログラム例

ロボット言語PACのプログラム例を載せてあります。コマンドの使い方を理解するための資料としてご利用ください。

### 第２章　プログラムの動作機構

プログラムを作成する際に必要となる、PAC言語によるプログラムの、動作上のきまりについて説明します。

### 第３章　ロボットの動作の種類

ロボットの動作は、その基準とする位置の取り方や、目標位置に到達したことを判定する方法によって、いくつかの種類があります。こうしたロボットの動作の種類について説明します。

### 第４章　速度・加速度・減速度の指定

速度、加速度、減速度の意味と設定について説明します。

### 第５章　視覚制御

プログラムを作成する際に必要となる、視覚に関する用語について説明します。

# 第２部　コマンドリファレンス

## 第 6 章　コマンドの表記方法と分類

ロボット言語PACのコマンドの表記方法とコマンド一覧を記載しています。

個別のコマンドについて、すばやく説明を見つけたいときは、この章のコマンド一覧表からコマンドの説明ページを見つけることができます。

## 第 7 章　PAC 言語の構成要素

この章では、PAC言語を構成する要素（識別子、変数、定数、演算子、式、コマンドなど）のきまりについて説明します。

## 第 8 章　PAC 言語の文法

PAC言語によってプログラムを書く場合の、文法上のきまりについて説明します。

## 第 9 章～第 21 章

ロボット言語PACの各コマンドについて説明しています。機能別にまとめてあります。

特定のコマンドについて、すばやく説明を見つけたいときは、第8章のコマンド一覧表からコマンドの説明ページを見つけることができます。

## 第 22 章　付録

各種のコードやパラメータなど、プログラムの作成と運用にあたって必要となる情報をまとめてあります。

# 目次

**アルファベット順コマンド一覧（総合目次の後に掲載してあります。）**

**機能別コマンド一覧（アルファベット順コマンド一覧の後に掲載してあります。）**

## 第１部 プログラムの基礎知識

### 第 1 章 プログラム例

1.1 モデルケースアプリケーション ..... 1-1
1.2 プログラムフロー ..... 1-2
1.3 プログラムリスト ..... 1-3

### 第 2 章 プログラムの動作機構

2.1 プログラムの呼び出しとサブルーチン ..... 2-1
2.1.1 サブルーチンの呼び出し ..... 2-2
2.1.2 プログラムの呼び出し ..... 2-3
2.1.3 プログラムの再帰呼び出し ..... 2-4
2.1.4 フォルダー構造のプログラムの呼び出し方 [RC7_Ver.2.2 以降] ..... 2-5
2.2 プログラムの起動 ..... 2-7
2.2.1 ティーチングペンダント・ミニペンダントからの起動 ..... 2-7
2.2.2 外部機器からの起動 ..... 2-7
2.3 マルチタスク ..... 2-8
2.3.1 優先順位 ..... 2-8
2.3.2 タスク間通信 ..... 2-8
2.4 シリアル通信 ..... 2-10
2.4.1 回線番号 ..... 2-10
2.4.2 通信コマンド ..... 2-11
2.4.3 通信バッファのクリア ..... 2-11
2.4.4 通信ポートコマンド ..... 2-11
2.4.5 サンプル・アプリケーション ..... 2-12
2.5 ライブラリ ..... 2-16
2.5.1 プログラムバンク ..... 2-16
2.5.2 パレタイジングライブラリ ..... 2-17
2.6 TP 簡易操作盤 [Ver.1.5 以降] ..... 2-28
2.6.1 操作盤画面作成のプログラム例 ..... 2-30

### 第 3 章 ロボットの動作の種類

3.1 絶対動作と相対動作 ..... 3-1
3.1.1 絶対動作 ..... 3-1
3.1.2 相対動作 ..... 3-1
3.1.3 絶対動作と相対動作の動作例 ..... 3-1

3.2 到達位置の確認方法 ... 3-3
3.2.1 パス動作 ... 3-3
3.2.2 エンド動作 ... 3-3
3.2.3 エンコーダ値確認動作 ... 3-3
3.2.4 パス動作、エンド動作、エンコーダ値確認動作の動作例 ... 3-4
3.2.5 パス動作、エンド動作、エンコーダ値確認動作の実行時間の違い ... 3-5
3.2.6 パス動作しない場合 ... 3-6
3.2.7 パス動作の効果が小さくなる場合 ... 3-7
3.2.8 加速度がパス動作の経路に影響する場合 ... 3-8
3.2.9 パス開始変位 ... 3-9
3.2.10 アーチモーション機能 [Ver.1.9 以降、4 軸ロボットのみ] ... 3-11
3.3 補間制御 ... 3-12
3.3.1 PTP 制御 ... 3-12
3.3.2 CP 制御 ... 3-13
3.3.3 円弧補間制御 ... 3-13
3.3.4 自由曲線補間 [Ver2.3 以降] ... 3-14
3.3.5 高軌跡制御機能 [Ver.2.61 以降] ... 3-19
3.3.6 特異点回避機能［Ver2.61 以降、6 軸専用］ ... 3-20
3.4 動作命令の後に出力コマンドがある場合 ... 3-24
3.5 力制限機能 ... 3-25
3.5.1 概要 ... 3-25
3.5.2 各軸電流制限機能 [V1.2 以降] ... 3-25
3.5.3 先端力制限機能 [V1.4 以降] ... 3-27
3.5.4 衝突検出機能 ... 3-33

**第 4 章 速度・加速度・減速度 の指定**

4.1 外部速度・内部速度 ... 4-1
4.2 速度指定 ... 4-2
4.3 外部加速度・外部減速度・内部加速度・内部減速度 ... 4-3
4.4 加速度・減速度の設定 ... 4-4
4.5 速度・加速度設定例 ... 4-5
4.6 最適可搬質量設定機能 ... 4-8
4.6.1 モード 0 ... 4-8
4.6.2 モード 1 ... 4-9
4.6.3 モード 2 ... 4-10
4.6.4 モード 3 ... 4-11
4.6.5 設定が必要な使用条件 ... 4-11
4.6.6 設定にあたって ... 4-12
4.7 「使用条件」における最適可搬質量設定機能 ... 4-13
4.7.1 外部負荷条件値(先端負荷質量、負荷重心位置)と外部モード設定方法 ... 4-13
4.7.2 内部負荷条件値(先端負荷質量､負荷重心位置)と内部モード設定方法 ... 4-17
4.7.3 ロボットの設置条件設定方法 ... 4-18
4.7.4 最適可搬質量初期化設定の設定方法 [V1.4 以降] ... 4-21

**第 5 章 視覚制御**

5.1 視覚制御 ... 5-1
5.1.1 視覚制御の用語 ... 5-1
5.1.2 視覚命令を使うにあたっての注意点（視覚ボード） ... 5-10
5.1.3 従来の μVision-15 との互換性 ... 5-11

# 第２部 コマンドリファレンス

## 第 6 章 コマンドの表記方法と分類


6.1 コマンド解説の表記方法 ..... 6-1
6.2 コマンド一覧表 ..... 6-2
6.2.1 アルファベット順コマンド一覧 ..... 6-2
6.2.2 機能別コマンド一覧 ..... 6-2


## 第 7 章 PAC 言語の構成要素


7.1 新しいロボット言語 PAC ..... 7-1
7.2 ロボット言語 PAC と従来の言語との関係 ..... 7-2
7.3 言語要素 ..... 7-3
7.4 名前 ..... 7-3
7.5 識別子 ..... 7-4
7.5.1 変数 ..... 7-4
7.5.2 関数 ..... 7-8
7.5.3 ラベル ..... 7-8
7.5.4 プログラム ..... 7-9
7.6 データ型 ..... 7-10
7.6.1 文字列 ..... 7-10
7.6.2 数値 ..... 7-10
7.6.3 ベクトル ..... 7-10
7.6.4 ポーズ ..... 7-10
7.6.5 I/O（ON/OFF） ..... 7-10
7.7 データ型の変換 ..... 7-11
7.7.1 数値 ..... 7-11
7.7.2 文字と数値 ..... 7-11
7.7.3 ポーズ型データ ..... 7-11
7.8 定数 ..... 7-12
7.8.1 数値定数 ..... 7-12
7.8.2 文字列定数 ..... 7-14
7.8.3 ベクトル型定数 ..... 7-14
7.8.4 ポーズ定数 ..... 7-14
7.9 式と演算子 ..... 7-16
7.9.1 代入演算子 ..... 7-16
7.9.2 算術演算子 ..... 7-16
7.9.3 関係演算子 ..... 7-17
7.9.4 論理演算子 ..... 7-18
7.9.5 文字列演算子 ..... 7-18
7.9.6 ベクトル演算 ..... 7-19
7.9.7 ポジション演算 ..... 7-19
7.9.8 ジョイント演算 ..... 7-20
7.9.9 同次変換行列演算 ..... 7-20
7.9.10 算術演算子、論理演算子、関係演算子の優先順位 [Ver.1.5 以降] ..... 7-20
7.10 PAC 言語における単位の取り扱い ..... 7-22

## 第 8 章 PAC 言語の文法


8.1 文と行 ..... 8-1
8.2 プログラム名と宣言 ..... 8-2
8.3 ラベル ..... 8-3
8.4 文字セット ..... 8-4
8.5 予約語 ..... 8-5
8.6 宣言文 ..... 8-6
8.6.1 型宣言 ..... 8-6
8.6.2 関数／プログラム宣言 ..... 8-9
8.6.3 ユーザ座標系の宣言 ..... 8-9
8.6.4 フォルダー変数宣言／EXTERN 変数宣言 [RC7_Ver.2.2 以降] ..... 8-9
8.7 代入文 ..... 8-10
8.7.1 数値代入文 ..... 8-10
8.7.2 文字列代入文 ..... 8-10
8.7.3 ベクトル代入文 ..... 8-10
8.7.4 ポーズ代入文 ..... 8-11
8.8 フロー制御文 ..... 8-13
8.8.1 無条件分岐 ..... 8-13
8.8.2 条件分岐 ..... 8-13
8.8.3 選択 ..... 8-13
8.8.4 繰り返し ..... 8-14
8.8.5 定義済み処理呼び出し ..... 8-15
8.9 ロボット制御文 ..... 8-17
8.9.1 動作制御文 ..... 8-17
8.9.2 停止制御文 ..... 8-17
8.9.3 速度制御文 ..... 8-18
8.9.4 時間制御文 ..... 8-18
8.9.5 座標変換文 ..... 8-18
8.10 入出力制御文 ..... 8-19
8.10.1 DI/DO 制御文 ..... 8-19
8.10.2 RS232C 制御文 ..... 8-19
8.10.3 ティーチングペンダント制御文 ..... 8-19
8.11 マルチタスク制御文 ..... 8-20
8.11.1 タスク制御文 ..... 8-20
8.11.2 セマフォ制御文 ..... 8-20
8.11.3 専用セマフォ制御文 ..... 8-20
8.12 時刻・日付制御 ..... 8-21
8.13 エラー制御 ..... 8-22
8.13.1 エラー制御コマンド ..... 8-22
8.13.2 エラー格納機能［Ver.1.98 以降］ ..... 8-22
8.14 システム情報 ..... 8-25
8.15 プリプロセッサ ..... 8-26
8.16 値による呼び出しと参照による呼び出し ..... 8-27
8.16.1 値による呼び出し ..... 8-27
8.16.2 参照による呼び出し ..... 8-28
8.17 視覚制御 ..... 8-29
8.17.1 画像入出力 ..... 8-29
8.17.2 ウィンドウ設定 ..... 8-29
8.17.3 描画 ..... 8-30
8.17.4 画像処理 ..... 8-30
8.17.5 コード認識 ..... 8-31
8.17.6 ラベリング ..... 8-31
8.17.7 サーチ機能 ..... 8-31

8.17.8 結果取得 ..... 8-32
8.17.9 視覚 CAL ..... 8-32

**第 9 章 宣言文**

9.1 プログラム名 ..... 9-1
9.2 干渉エリア座標 ..... 9-2
9.3 ユーザ関数 ..... 9-6
9.4 ホーム座標 ..... 9-7
9.5 ツール座標 ..... 9-8
9.6 ワーク座標 ..... 9-9
9.7 ローカル変数 ..... 9-11
9.8 配列 ..... 9-19
9.9 フォルダー機能 ..... 9-20

**第 10 章 代入文**

10.1 変数 ..... 10-1
10.2 ベクトル ..... 10-2
10.3 形態 ..... 10-5
10.4 リンク角 ..... 10-6
10.5 姿勢 ..... 10-7
10.6 回転成分 ..... 10-8
10.7 軸成分 ..... 10-12

**第 11 章 フロー制御文**

11.1 プログラムの停止 ..... 11-1
11.2 呼び出し ..... 11-4
11.3 繰り返し ..... 11-9
11.4 条件分岐 ..... 11-17
11.5 無条件分岐 ..... 11-21
11.6 コメント ..... 11-23

**第 12 章 ロボット制御文**

12.1 動作制御 ..... 12-1
12.2 形態制御 ..... 12-35
12.3 停止制御 ..... 12-41
12.4 速度制御 ..... 12-45
12.5 時間制御 ..... 12-60
12.6 座標変換 ..... 12-63
12.7 干渉チェック ..... 12-67
12.8 サーボ内部データ ..... 12-71
12.9 モータ電源 ..... 12-73
12.10 CAL ..... 12-74
12.11 特殊制御 ..... 12-75

**第 13 章 入出力制御文**

13.1 I/O ポート ..... 13-1
13.2 RS232C および Ethernet ポート ..... 13-10
13.3 シリアルバイナリ通信（RS-232C および Ethernet ポート） ..... 13-16
13.4 ティーチングペンダント ..... 13-23
13.5 TP 簡易操作盤 [Ver.1.5 以降] ..... 13-26
13.5.1 操作盤画面作成のプログラム例 ..... 13-37

## 第 14 章 マルチタスク制御文

14.1 タスク制御 ..... 14-1
14.2 セマフォ ..... 14-15
14.3 アームセマフォ ..... 14-22
14.4 特権タスク ..... 14-30

## 第 15 章 関数

15.1 算術関数 ..... 15-1
15.2 三角関数 ..... 15-12
15.3 角度変換 ..... 15-19
15.4 速度変換 ..... 15-22
15.5 時間関数 ..... 15-24
15.6 ベクトル ..... 15-25
15.7 ポーズデータ型変換 ..... 15-29
15.8 距離抽出 ..... 15-37
15.9 形態成分 ..... 15-38
15.10 角度成分 ..... 15-39
15.11 軸成分 ..... 15-40
15.12 回転成分 ..... 15-43
15.13 姿勢成分 ..... 15-47
15.14 位置関数 ..... 15-48
15.15 文字列関数 ..... 15-52

## 第 16 章 定数

16.1 組み込み定数 ..... 16-1

## 第 17 章 時刻／日付制御

17.1 時刻／日付 ..... 17-1

## 第 18 章 エラー制御

18.1 エラー情報 ..... 18-1

## 第 19 章 システム情報

19.1 システム ..... 19-1
19.2 ログ ..... 19-4
19.3 動作モード ..... 19-13

## 第 20 章 プリプロセッサ

20.1 記号定数・マクロ定義 ..... 20-1
20.2 ファイル取り込み ..... 20-4
20.3 最適化 ..... 20-6

## 第 21 章　視覚制御(ロボットコントローラ:オプション)

21.1 視覚命令を使うにあたっての注意点（視覚ボード） ..... 21-1
21.2 従来のμVision-15 との互換性 ..... 21-1
21.3 画像入出力 ..... 21-3
21.4 ウィンドウ設定 ..... 21-13
21.5 描画 ..... 21-23

21.6 画像処理 ..... 21-43
21.7 コード認識 ..... 21-67
21.8 ラベリング ..... 21-70
21.9 サーチ機能 ..... 21-79
21.10 結果取得 ..... 21-97

**第 22 章 付録**

22.1 文字コード表 ..... 22-1
22.2 ロボット形態 ..... 22-2
22.3 環境設定値 ..... 22-13
22.4 使用条件パラメータ ..... 22-14
22.5 予約語一覧 ..... 22-24
22.6 旧言語コマンド対応表(VS) ..... 22-26
22.7 バージョン対応表 ..... 22-31
22.8 設定パラメータ表 ..... 22-32

# アルファベット順コマンド一覧

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |


| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| **#** | | | | | |
| #define | プログラム中の指定した定数またはマクロ名を、指定文字列で置き換えます。 | ◎ | ◎ | ◎ | 20-1 |
| #error | #error コマンドを実行すると、強制的にコンパイルエラーとします。 | ◎ | ◎ | ◎ | 20-3 |
| #include | プリプロセッサプログラムを取り込みます。 | ◎ | ◎ | ◎ | 20-4 |
| #pragma optimize | プログラムごとに行う最適化を指定します。 | ◎ | ◎ | ◎ | 20-6 |
| #undef | #define で定義されている記号定数またはマクロの定義を無効にします。 | ◎ | ◎ | ◎ | 20-2 |
| **A** | | | | | |
| ABS | 数式の値の絶対値を得ます。 | ◎ | ◎ | ◎ | 15-1 |
| ACCEL | 現在取得しているアームグループ軸の内部加速度、内部減速度を指定します。 | ◎ | ◎ | | 12-48 |
| ACOS | 逆余弦 ( アークコサイン ) を得ます。 | ◎ | ◎ | ◎ | 15-12 |
| APPROACH | ツール座標系指定の絶対動作を行ないます。 | ○ | ○ | | 12-1 |
| AREA | 干渉チェックを行なうエリアを宣言します。 | ○ | ○ | | 9-2 |
| AREAPOS | 干渉チェックの行なわれる領域の中心位置と直方体の方向をポジション型で返します。 | ◎ | ◎ | | 15-48 |
| AREASIZE | 干渉チェックの行なわれる領域を定義する、直方体の大きさ（各辺の長さ）をベクトル型で返します。 | ◎ | ◎ | | 15-49 |
| ARRIVE | 動作命令の全移動距離に対する動作割合を設定する事によって、ロボットが設定した動作割合に到達するまでプログラムを待機させます。 | ◎ | V1.2 | | 12-29 |
| ASC | 文字コードへ変換します。 | ◎ | ◎ | ◎ | 15-52 |
| ASIN | 逆正弦 ( アークサイン ) を得ます。 | ◎ | ◎ | ◎ | 15-13 |
| ATN | 逆正接 ( アークタンジェント ) を得ます。 | ◎ | ◎ | ◎ | 15-14 |
| ATN2 | 数式 1 を数式 2 で除算した逆正接 ( アークタンジェント ) を得ます。 | ◎ | ◎ | ◎ | 15-15 |
| AVEC | アプローチベクトルを抽出します。 | ◎ | ◎ | | 15-25 |
| **B** | | | | | |
| BIN$ | 数式の値を 2 進数の文字列へ変換します。 | ◎ | ◎ | ◎ | 15-53 |
| BLOB | ラベリングを実行します。 | ◎ | ◎ | ◎ | 21-70 |
| BLOBCOPY | 対象ラベル番号をコピーします。 | ◎ | ◎ | ◎ | 21-77 |
| BLOBLABEL | 指定座標のラベル番号を取得します。 | ◎ | ◎ | ◎ | 21-75 |
| BLOBMEASURE | 対象ラベル番号の特徴計測を行ないます。 | ◎ | ◎ | ◎ | 21-73 |
| BUZZER | ブザーを鳴らします。 | ◎ | ◎ | | 13-24 |
| **C** | | | | | |
| CALL | プログラムを呼び出し、実行します。 | ◎ | ◎ | ◎ | 11-4 |
| CAMIN | カメラからの映像を画像メモリ（処理画面）に格納します。 | ◎ | ◎ | ◎ | 21-3 |
| CAMLEVEL | カメラ映像の入力レベルを設定します。 | ◎ | ◎ | ◎ | 21-6 |
| CAMMODE | カメラ映像を格納する際の機能を設定します。 | ◎ | ◎ | ◎ | 21-4 |
| change_bCap | ボタンのキャプション設定を行ないます。 | V1.5 | V1.5 | | 13-34 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| change_pCap | ページのキャプション設定を行ないます。 | V1.5 | V1.5 | | 13-35 |
| CHANGETOOL | ツール座標系を変更します。 | ◎ | ◎ | | 12-63 |
| CHANGEWORK | ユーザ座標系を変更します。 | ◎ | ◎ | | 12-64 |
| CHGEXTMODE | 外部自動モード切替を行ないます。 | V1.98 | V1.98 | | 19-13 |
| CHGINTMODE | 内部自動モード切替を行ないます。 | V1.98 | V1.98 | | 19-14 |
| CHR$ | ASCII コードを文字に変換します。 | ◎ | ◎ | ◎ | 15-54 |
| CLEARLOG | サーボ制御ログの記録を初期化します。 | ◎ | ◎ | | 19-5 |
| CLEARTRACELOG | トレースログデータをクリアします。 | V2.7 | V2.7 | | 19-9 |
| CLEARVARLOG | 変数ログデータをクリアします。 | V2.7 | V2.7 | | 19-12 |
| CLRERR | エラーをクリアします。 | V1.98 | V1.98 | | 18-3 |
| CLRSPLINEPOINT | 自由曲線動作の通過点をすべて消去します。 | V2.3 | V2.3 | | 12-33 |
| com_discom | RS-232C ポートをバイナリ通信終了のため開放します。(COM ポート開放) Ethernet クライアントポート使用時にクライアントポートのクローズを おこない接続を切断します。 | V1.5 | V1.5 | V1.9 | 13-21 |
| com_encom | RS-232C ポートまたは Ethernet クライアントポートを使ってバイナリ通信行うための準備をします。RS-232C ポートの場合は、バイナリ通信のためにポートを占有します。Ethernet クライアントポートを使う場合はクライアントポートのオープンを行いサーバとの間で接続を確立します。 | V1.5 | V1.5 | V1.9 | 13-20 |
| com_state | RS-232C および Ethernet ポートの状態を取得します。 | V1.5 | V1.5 | V1.9 | 13-22 |
| CONTINUERUN | コンティニュー起動を行います。 | V1.98 | V1.98 | | 14-10 |
| COS | 余弦 ( コサイン ) を得ます。 | ◎ | ◎ | ◎ | 15-16 |
| CREATESEM | セマフォを生成します。 | ◎ | ◎ | | 14-15 |
| CURACC | 現在取得しているアームグループ軸の、内部加速度を取得します。 | ◎ | ◎ | | 12-52 |
| CURDEC | 現在取得しているアームグループ軸の、内部減速度を取得します。 | ◎ | ◎ | | 12-54 |
| CURERRSTATUS | エラー発生状態を返します。(特権タスク専用命令) | V2.2 | V2.2 | | 14-34 |
| CUREXJ | 付加軸の現在角度を F 型で取得します。 | V1.5 | V1.5 | | 12-24 |
| CUREXTACC | 外部加速度の現在値を得ます。 | V1.4 | V1.4 | | 12-58 |
| CUREXTDEC | 外部減速度の現在値を得ます。 | V1.4 | V1.4 | | 12-58 |
| CUREXTSPD | 外部速度の現在値を得ます。 | V1.4 | V1.4 | | 12-59 |
| CURFIG | ロボット形態の現在値を得ます。 | ◎ | ◎ | | 12-35 |
| CURJACC | 現在取得しているアームグループ軸の、内部軸加速度を取得します。 | ◎ | ◎ | | 12-53 |
| CURJDEC | 現在取得しているアームグループ軸の、内部軸減速度を取得します。 | ◎ | ◎ | | 12-55 |
| CURJNT | ロボットの現在角度を J 型で得ます。 | ○ | ○ | | 12-21 |
| CURJSPD | 現在取得しているアームグループ軸の、内部移動軸速度を取得します。 | ◎ | ◎ | | 12-56 |
| CUROPTMODE | 動作モードを取得します。 | V1.98 | V1.98 | | 19-15 |
| CURPOS | ツール座標系での現在位置を P 型データで得ます。 | ○ | ○ | | 12-22 |
| CURSPD | 現在取得しているアームグループ軸の、内部移動速度を取得します。 | ◎ | ◎ | | 12-57 |
| CURTOOL | 現在設定されている TOOL 番号を得ます。 | V1.4 | V1.4 | | 12-65 |
| CURTRN | ツール座標系での現在位置を T 型データで得ます。 | ◎ | ◎ | | 12-23 |
| CURWORK | 現在設定されている WORK 番号を得ます。 | V1.4 | V1.4 | | 12-66 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| **D** | | | | | |
| DATE$ | 現在の日付を得ます。 | ◎ | ◎ | | 17-1 |
| DEADMANSTATE | デッドマンスイッチの状態を取得します。 0：OFF、 1：ON | V2.2 | V2.2 | | 14-13 |
| DECEL | 現在取得しているアームグループ軸の内部減速度を指定します。 | ◎ | ◎ | | 12-50 |
| DEF FN | ユーザ定義関数を宣言します。 | ◎ | ◎ | ◎ | 9-6 |
| DEFDBL | D 型変数（倍精度実数型変数）を宣言します。倍精度実数の範囲は -1.7D+308 ～ 1.7D+308 です。 | ◎ | ◎ | ◎ | 9-12 |
| DEFEND | タスクを保護します。 | ◎ | ◎ | | 14-5 |
| DEFINT | I 型変数（整数型変数）を宣言します。整数の範囲は -2147483648 ～ 2147483647 です。 | ◎ | ◎ | ◎ | 9-11 |
| DEFIO | 入出力ポートに対応する IO 変数（I/O 変数）を宣言します。 | ◎ | ◎ | ◎ | 9-17 |
| DEFJNT | J 型変数（ジョイント型変数）を宣言します。 | ○ | ○ | | 9-15 |
| DEFPOS | P 型変数（ポジション型変数）を宣言します。 | ○ | ○ | | 9-14 |
| DEFSNG | F 型変数（単精度実数型変数）を宣言します。単精度実数の範囲は -3.4E+38 ～ 3.4E+38 です。 | ◎ | ◎ | ◎ | 9-11 |
| DEFSTR | S 型変数（文字列型変数）を宣言します。文字列の長さは、243 文字までです。 | ◎ | ◎ | ◎ | 9-12 |
| DEFTRN | T 型変数（同次変換型変数）を宣言します。 | ◎ | ◎ | | 9-16 |
| DEFVEC | V 型変数（ベクトル型変数）を宣言します。 | ◎ | ◎ | | 9-13 |
| DEGRAD | 単位をラジアンに変換します。 | ◎ | ◎ | ◎ | 15-19 |
| DELAY | 指定した時間の間、プログラムの進行を停止します。 | ◎ | ◎ | ◎ | 12-60 |
| DELETESEM | セマフォを削除します。 | ◎ | ◎ | | 14-18 |
| DEPART | ツール座標系指定で相対動作を行ないます。 | ○ | ○ | | 12-4 |
| DESTEXJ | 付加軸の、現在の動作命令で指定された目標位置を F 型で取得します。停止時は、現在の位置（指令値）を取得します。 | V1.5 | V1.5 | | 12-28 |
| DESTJNT | 現在の動作命令目標位置を J 型で取得します。ロボット停止時は、現在の位置（指令値）を取得します。 | ○ | ○ | | 12-25 |
| DESTPOS | 現在の動作命令目標位置を P 型で取得します。ロボット停止時は、現在の位置（指令値）を取得します。 | ○ | ○ | | 12-26 |
| DESTTRN | 現在の動作命令目標位置を T 型で取得します。ロボット停止時は、現在の位置（指令値）を取得します。 | ◎ | ◎ | | 12-27 |
| DETECT ON/OFF | ハンド入力の立上り及び立下りを検知し、その時のロボットの位置を変数に格納します。 | V3.2 | V3.2 | | 13-8 |
| DIM | 配列を宣言します。 | ◎ | ◎ | ◎ | 9-19 |
| disp_page | ページの表示を行ないます。 | V1.5 | V1.5 | | 13-36 |
| DIST | 2 点間の距離を返します。 | V1.8 | V1.8 | | 15-37 |
| DO...LOOP | 判定反復（繰り返し）を行ないます。 | ◎ | ◎ | ◎ | 11-9 |
| DRAW | ワーク作業座標系指定で相対動作を行ないます。 | ◎ | ◎ | | 12-6 |
| DRIVE | 各軸の相対動作を行ないます。 | ◎ | ◎ | | 12-8 |
| DRIVEA | 各軸の絶対動作を行ないます。 | ◎ | ◎ | | 12-10 |
| **E** | | | | | |
| END | プログラムによる動作の終了を宣言します。 | ◎ | ◎ | ◎ | 11-1 |
| ERRMSG$ | エラーメッセージを与えます。 | ◎ | ◎ | ◎ | 18-1 |
| EXECAL | CAL 実行 | V1.98 | V1.98 | | 12-74 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| EXIT DO | DO ～ LOOP からの強制脱出を行ないます。 | ◎ | ◎ | ◎ | 11-11 |
| EXIT FOR | FOR ～ NEXT からの強制脱出を行ないます。 | ◎ | ◎ | ◎ | 11-14 |
| EXP | 自然対数を基数とする指数関数を得ます。 | ◎ | ◎ | ◎ | 15-2 |
| EXTERN | 他のプログラムで定義されている FOLDER 変数を参照することを宣言します。 | V2.2 | V2.2 | | 9-23 |
| EXTSPEED | 外部速度設定 | V1.98 | V1.98 | | 12-59 |
| **F** | | | | | |
| FALSE | ブール値の偽 (0) の値を与えます。 | ◎ | ◎ | ◎ | 16-4 |
| FIG | 形態を抽出します。 | ○ | ○ | | 15-38 |
| FIGAPRL | CP 動作可能なアプローチ位置と基準位置の形態を計算します。 | ○ | ○ | | 12-37 |
| FIGAPRP | PTP 動作可能なアプローチ位置と基準位置の形態を計算します。 | ○ | ○ | | 12-39 |
| FLUSH | 入力バッファをクリアします。 | ◎ | ◎ | ◎ | 13-15 |
| FLUSHSEM | セマフォ待ちタスクを解放します。 | ◎ | ◎ | | 14-19 |
| FOLDER | 外部プログラムからアクセス可能なローカル変数を宣言します。 | V2.2 | V2.2 | | 9-20 |
| FOR...NEXT | FOR ～ NEXT までの区間中にある一連の命令を繰り返して実行します。 | ◎ | ◎ | ◎ | 11-12 |
| **G** | | | | | |
| GETENV | システムの環境設定値を取得します。 | ◎ | ◎ | ◎ | 19-1 |
| GETERR | エラー格納機能で宣言されたバッファからエラーコードを取得します。 | V1.98 | V1.98 | | 18-2 |
| GETERRLVL | エラーコードのレベルを与えます。 | V1.98 | V1.98 | | 18-4 |
| GetJntData | 指定軸のサーボ内部データを取得します。(Ver. 1.5 以降対応 ) | | | | 12-72 |
| GETLANGUAGE | 現在の言語設定を取得します。 | V2.2 | V2.2 | | 19-3 |
| GETSPLINEPOINT | 登録済の自由曲線動作の通過点を取り出します。 | V2.3 | V2.3 | | 12-34 |
| GetSrvData | ロボット軸のサーボ内部データを取得します。(Ver. 1.5 以降対応 ) | | | | 12-71 |
| GIVEARM | 現在取得しているアームグループを解放します。アームグループ 0 （ロボットのみ）の場合、ロボット制御権を開放します。 | ◎ | ◎ | | 14-27 |
| GIVESEM | セマフォ待ちタスクを解放します。 | ◎ | ◎ | | 14-20 |
| GIVEVIS | 視覚処理権を解放します。 | ◎ | ◎ | | 14-29 |
| GOHOME | HOME 文で定義したポジション（原点）へ移動します。 | ◎ | ◎ | | 12-12 |
| GOSUB | サブルーチンを呼び出します。 | ◎ | ◎ | ◎ | 11-6 |
| GOTO | プログラムの分岐を無条件で行ないます。 | ◎ | ◎ | ◎ | 11-21 |
| **H** | | | | | |
| HALT | プログラムの実行を停止します。 | ◎ | ◎ | | 12-42 |
| HEX$ | 10 進数から 16 進数へ変換した値を文字列として得ます。 | ◎ | ◎ | ◎ | 15-57 |
| HOLD | プログラムの実行を一時停止します。 | ◎ | ◎ | | 12-41 |
| HOME | 任意の座標をホームポジションとして宣言します。 | ○ | ○ | | 9-7 |
| **I** | | | | | |
| IF...END IF | IF ～ END IF 間の条件式の条件判断を行ないます。 | ◎ | ◎ | ◎ | 11-17 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| IF...THEN...ELSE | 論理式の条件判断を行ないます。 | ◎ | ◎ | ◎ | 11-18 |
| IN | I/O 変数で示される I/O ポートからデータを読み込みます。 | ◎ | ◎ | ◎ | 13-1 |
| INIT | 特権タスクパラメータの状態により、モータ ON、CAL、スピード設定を行います。 | V1.7 | V1.7 | | 14-30 |
| INITWAITERR | エラー検知を初期化します。(特権タスク専用命令) | V2.2 | V2.2 | | 14-32 |
| INPUT | RS232C および Ethernet ポートからのデータを得ます。 | ◎ | ◎ | ◎ | 13-10 |
| INPUTB | 1 バイトデータを RS-232C および Ethernet ポートから入力します。 | V1.5 | V1.5 | V1.9 | 13-17 |
| INT | 指定された値を超えない最大の整数値を得ます。 | ◎ | ◎ | ◎ | 15-3 |
| INTERRUPT ON/OFF | ロボットの動作を中断します。 | ◎ | ◎ | | 12-43 |
| IOBLOCK ON/OFF | 動作命令を実行中に、I/O 命令や演算命令などの非動作命令を並列に実行します。 | ◎ | ◎ | | 13-3 |
| **J** | | | | | |
| J2P | ジョイント型からポジション型に変換します。 | ○ | ○ | | 15-29 |
| J2T | ジョイント型から同次変換型に変換します。 | ○ | ○ | | 15-30 |
| JACCEL | 現在取得しているアームグループ軸の、内部軸加速度、内部軸減速度を指定します。 | ◎ | ◎ | | 12-49 |
| JDECEL | 現在取得しているアームグループ軸の内部軸減速度を指定します。 | ◎ | ◎ | | 12-51 |
| JOINT | ジョイント型座標から角度を抽出します。 | ○ | ○ | | 15-39 |
| JSPEED | 現在取得しているアームグループ軸の内部軸速度を指定します。 | ◎ | ◎ | | 12-47 |
| **K** | | | | | |
| KILL | タスクを強制終了します。 | ◎ | ◎ | | 14-3 |
| KILLALL | 特権タスク以外の全てのタスクを強制終了します。(プログラムリセットに相当) | V1.98 | V1.98 | | 14-9 |
| **L** | | | | | |
| LEFT$ | 左部分文字列を抽出します。 | ◎ | ◎ | ◎ | 15-58 |
| LEN | 文字列の長さをバイト数で得ます。 | ◎ | ◎ | ◎ | 15-59 |
| LET | 変数に値を代入します。 | ○ | ○ | ○ | 10-1 |
| LETA | 同次変換型のアプローチベクトルへ代入します。 | | ◎ | | 10-2 |
| LETENV | システムの環境設定値を設定します。 | ◎ | ◎ | ◎ | 19-2 |
| LETF | ポジション型または同次変換型の形態成分へ代入します。 | ◎ | ◎ | | 10-5 |
| LETJ | ジョイント型の指定リンク角へ代入します。 | ◎ | ◎ | | 10-6 |
| LETO | 同次変換型のオリエントベクトルへ代入します。 | ◎ | ◎ | | 10-3 |
| LETP | ポジション型または同次変換型の位置ベクトルへ代入します。 | ◎ | ◎ | | 10-4 |
| LETR | ポジション型の 3 つの回転成分へ代入します。 | | ◎ | | 10-7 |
| LETRX | ポジション型の X 軸回転成分へ代入します。 | | ◎ | | 10-8 |
| LETRY | ポジション型の Y 軸回転成分へ代入します。 | | ◎ | | 10-9 |
| LETRZ | ポジション型の Z 軸回転成分へ代入します。 | | ◎ | | 10-10 |
| LETT | ポジション型の T 成分へ代入します。 | ◎ | | | 10-11 |
| LETX | ベクトル型／ポジション型／同次変換型の X 軸成分へ代入します。 | ◎ | ◎ | | 10-12 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| LETY | ベクトル型／ポジション型／同次変換型の Y 軸成分へ代入します。 | ◎ | ◎ | | 10-13 |
| LETZ | ベクトル型／ポジション型／同次変換型の Z 軸成分へ代入します。 | ◎ | ◎ | | 10-14 |
| LINEINPUT | RS232C および Ethernet ポートからデリミタまでのデータを読み込み、文字列型変数に代入します。 | ◎ | ◎ | ◎ | 13-12 |
| LINPUTB | 複数バイトデータを RS-232C および Ethernet ポートから入力します。 | V1.5 | V1.5 | V1.9 | 13-19 |
| LOCKSTATE | マシンロックの状態を取得します。 0：OFF、 1：ON | V2.2 | V2.2 | | 14-12 |
| LOG | 自然対数を得ます。 | ◎ | ◎ | ◎ | 15-4 |
| LOG10 | 常用対数を得ます。 | ◎ | ◎ | ◎ | 15-5 |
| LPRINTB | 複数バイトデータを RS-232C および Ethernet ポートに出力します。 | V1.5 | V1.5 | V1.9 | 13-18 |
| **M** | | | | | |
| MAGNITUDE | ベクトルの大きさを得ます。 | ◎ | ◎ | | 15-28 |
| MAX | 最大値を抽出します。 | ◎ | ◎ | ◎ | 15-7 |
| MID$ | 文字列から指定した文字数分の文字列を取り出します。 | ◎ | ◎ | ◎ | 15-60 |
| MIN | 最小値を抽出します。 | ◎ | ◎ | ◎ | 15-8 |
| MOTOR {ON \| OFF} | モータ電源 ON/OFF を行ないます。 | V1.98 | V1.98 | | 12-73 |
| MOVE | ロボットを指定座標へ移動します。EX オプション ( 付加軸相対動作 ) または EXA オプション（付加軸絶対動作）を付けることにより、ロボットと付加軸を同期 ( 同時発着 ) で動作させることができます。 | ○ | ○ | | 12-13 |
| MPS | 指定した速度値（mm/sec）を CP 動作時の最大内部速度に対する割合（%）に変換します。 | ◎ | ◎ | | 15-22 |
| **N** | | | | | |
| NORMTRN | 同次変換型の正規化計算を行います。 | V1.8 | V1.8 | | 15-36 |
| **O** | | | | | |
| OFF | OFF(0) の値を与えます。 | ◎ | ◎ | ◎ | 16-1 |
| ON | ON(1) の値を与えます。 | ◎ | ◎ | ◎ | 16-2 |
| ON...GOSUB | 式の値に応じて、対応するサブルーチンの呼び出しを行ないます。 | ◎ | ◎ | ◎ | 11-7 |
| ON...GOTO | 式の値による無条件分岐を行ないます。 | ◎ | ◎ | ◎ | 11-22 |
| ORD | 文字コードへ変換します。 | ◎ | ◎ | ◎ | 15-61 |
| OUT | I/O 変数で示される I/O ポートにデータを出力します。 | ◎ | ◎ | ◎ | 13-2 |
| OVEC | オリエントベクトルを抽出します。 | ◎ | ◎ | | 15-26 |
| **P** | | | | | |
| P2J | ポジション型からジョイント型に変換します。 | ○ | ○ | | 15-31 |
| P2T | ポジション型から同次変換型に変換します。 | ○ | ○ | | 15-32 |
| PI | π の値を与えます。 | ◎ | ◎ | ◎ | 16-3 |
| POSCLR | 軸の現在位置を強制的に 0mm または 0 度にします。 | V1.5 | V1.5 | | 12-31 |
| POSRX | X 軸回転成分を抽出します。 | | ◎ | | 15-43 |
| POSRY | Y 軸回転成分を抽出します。 | | ◎ | | 15-44 |
| POSRZ | Z 軸回転成分を抽出します。 | | ◎ | | 15-45 |
| POST | T 軸回転成分を抽出します。 | ◎ | | | 15-46 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| POSTUREAREA | 高感度位置姿勢検知機能の設定を宣言します。 | V3.0 | V3.0 | | 9-4 |
| POSX | X 成分を抽出します。 | ○ | ○ | | 15-40 |
| POSY | Y 成分を抽出します。 | ○ | ○ | | 15-41 |
| POSZ | Z 成分を抽出します。 | ○ | ○ | | 15-42 |
| POW | べき乗を求めます。 | ◎ | ◎ | ◎ | 15-6 |
| PRINT | RS232C および Ethernet ポートからデータを出力します。 | ◎ | ◎ | ◎ | 13-13 |
| PRINTB | １バイトデータを RS-232C および Ethernet ポートに出力します。 | V1.5 | V1.5 | V1.9 | 13-16 |
| PRINTDBG | デバッグウィンドウにデータを出力します。 | ◎ | ◎ | | 13-24 |
| PRINTLBL | ユーザ定義ボタンにラベル（キャプション）を設定します。 | ◎ | ◎ | | 13-25 |
| PRINTMSG | メッセージをキャプションとアイコンを付けて、ティーチングペンダントのカラー液晶に表示します。 | ◎ | ◎ | | 13-23 |
| PRINTWARNING | ペンダント警告メッセージ表示部分へメッセージを表示します。 | V2.2 | V2.2 | | 13-25 |
| PROGRAM | プログラム名を宣言します。 | ◎ | ◎ | ◎ | 9-1 |
| PVEC | 位置ベクトルを抽出します。 | ○ | ○ | | 15-27 |
| **R** | | | | | |
| RAD | ラジアンで与えた数値を角度に変換します。 | ◎ | ◎ | ◎ | 15-20 |
| RADDEG | 単位を度に変換します。 | ◎ | ◎ | ◎ | 15-21 |
| REM | コメントの記述をします。 | ◎ | ◎ | ◎ | 11-23 |
| REPEAT...UNTIL | 後判定反復を行ないます。 | ◎ | ◎ | ◎ | 11-15 |
| RESET | I/O ポートを OFF にします。 | ◎ | ◎ | ◎ | 13-7 |
| RESETAREA | 干渉チェックを初期化します。 | ◎ | ◎ | | 12-68 |
| RESETPOSTUREAREA | 高感度位置姿勢検知機能を無効にします。 | V3.0 | V3.0 | | 12-70 |
| RETURN | サブルーチンから復帰します。 | ◎ | ◎ | ◎ | 11-8 |
| RIGHT$ | 文字列の右部分を抽出します。 | ◎ | ◎ | ◎ | 15-62 |
| RND | 0 以上 1 以下の乱数を発生します。 | ◎ | ◎ | ◎ | 15-9 |
| ROBOTSTOP | ロボット停止を行います。 | V1.98 | V1.98 | | 14-11 |
| ROTATE | 指定した軸回りの回転動作を行ないます。 | ○ | ○ | | 12-17 |
| ROTATEH | アプローチベクトルを軸とした、回転動作を行ないます。 | ◎ | ◎ | | 12-19 |
| RPM | 指定した軸番号の回転数 (rpm) を PTP 動作時の最大内部速度に対する割合（%）に変換します。 | V2.8 | V2.8 | | 15-23 |
| RUN | 別プログラムを並列起動します。 | ◎ | ◎ | | 14-1 |
| RVEC | 姿勢を抽出します。 | | ◎ | | 15-47 |
| **S** | | | | | |
| SEC | 秒の単位で与えた数値をミリ秒に変換します。 | ◎ | ◎ | ◎ | 15-24 |
| SELECT CASE | 複数条件判断を行ないます。 | ◎ | ◎ | ◎ | 11-19 |
| SEMIDSTATE | 指定したセマフォ ID が有効か無効かを返します。 | V2.2 | V2.2 | | 14-14 |
| SET | I/O ポートを ON にします。 | ◎ | ◎ | ◎ | 13-5 |
| set_button | ボタンパラメータの設定を行ないます。 | V1.5 | V1.5 | | 13-28 |
| set_page | ページパラメータの設定を行ないます。 | V1.5 | V1.5 | | 13-32 |
| SETAREA | 干渉チェックを行なうエリアを選択します。 | ◎ | ◎ | | 12-67 |
| SETERR | ユーザ定義エラーを I 型変数領域にセットします。 | V1.98 | V1.98 | | 18-1 |
| SETOCCUPATIONTIME | 特権タスクの占有時間を変更します。 | V2.0 | V2.0 | | 14-31 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| SETPOSTUREAREA | 高感度位置姿勢検知機能を有効にします。 | V3.0 | V3.0 | | 12-69 |
| SETSPLINEPOINT | 自由曲線動作の通過点を登録します。 | V2.3 | V2.3 | | 12-32 |
| SGN | 符号を調べます。 | ◎ | ◎ | ◎ | 15-10 |
| SHCIRCLE | 円をサーチします。 | ◎ | ◎ | ◎ | 21-95 |
| SHCLRMODEL | 登録モデルを消去します。 | ◎ | ◎ | ◎ | 21-84 |
| SHCOPYMODEL | 登録モデルをコピーします。 | ◎ | ◎ | ◎ | 21-83 |
| SHCORNER | コーナーをサーチします。 | ◎ | ◎ | ◎ | 21-91 |
| SHDEFCIRCLE | 円サーチの条件を設定します。 | ◎ | ◎ | ◎ | 21-94 |
| SHDEFCORNER | コーナーサーチの条件を設定します。 | ◎ | ◎ | ◎ | 21-90 |
| SHDEFMODEL | サーチモデルの登録をします。 | ◎ | ◎ | ◎ | 21-79 |
| SHDISPMODEL | 登録モデルを画面に表示します。 | ◎ | ◎ | ◎ | 21-85 |
| SHMODEL | モデルをサーチします。 | ◎ | ◎ | ◎ | 21-86 |
| SHREFMODEL | 登録モデルデータを参照します。 | ◎ | ◎ | ◎ | 21-82 |
| SIN | 正弦 ( サイン ) を得ます。 | ◎ | ◎ | ◎ | 15-17 |
| SPEED | 現在取得しているアームグループ軸の内部移動速度を指定します。 | ◎ | ◎ | | 12-45 |
| SPRINTF$ | 式を指定したフォーマットに変換し、文字列として返します。 | ◎ | ◎ | ◎ | 15-55 |
| SQR | 平方根を得ます。 | ◎ | ◎ | ◎ | 15-11 |
| ST_aspACLD | 内部負荷条件値を変更します。負荷条件値は、先端負荷質量 (g)、負荷重心位置（mm）（注 1）ですべて指定します。 | V1.9 | V1.9 | | 12-75 |
| ST_aspChange | 最適可搬質量設定モードの内部モードを選択します。 | V1.9 | V1.9 | | 12-76 |
| ST_OffSrvLock | 指定した軸のサーボロックを解除します。（4 軸ロボット専用） | V1.9 | | | 12-87 |
| ST_OnSrvLock | 指定した軸をサーボロック状態にします。（4 軸ロボット専用） | V1.9 | | | 12-86 |
| ST_ResetCompControl | 力制限機能を無効にします。( ６軸専用命令 ) | | V1.9 | | 12-90 |
| ST_ResetCompEralw | 力制限時のツール端の位置、姿勢偏差許容値を初期化します。( ６軸専用命令 ) | | V1.9 | | 12-103 |
| ST_ResetCompJLimit | 力制限時の電流制限値を初期化します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-99 |
| ST_ResetCompRate | 力制限時の柔らかさの割合を初期化します。( ６軸専用命令 ) | | V1.9 | | 12-95 |
| ST_ResetCompVMode | 力制限時の速度制御モードを無効にします。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-101 |
| ST_ResetCurLmt | 指定した軸のモータ電流制限を解除します。 | V1.9 | V1.9 | | 12-83 |
| ST_ResetDampRate | 力制限時の粘性割合を初期化します。( ６軸専用命令 ) | | V1.9 | | 12-105 |
| ST_ResetEralw | 指定した軸の偏差許容値をデフォルト値に戻します。 | V1.9 | V1.9 | | 12-85 |
| ST_ResetFrcAssist | 力制限時のオフセット力を初期化します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-97 |
| ST_ResetFrcLimit | 力制限割合を初期化します。( ６軸専用命令 ) | | V1.9 | | 12-93 |
| ST_ResetGravity | 重力バランスを無効にします。 | V1.9 | V1.9 | | 12-78 |
| ST_ResetGrvOffset | 重力補償値の補正を初期化します。 | V1.9 | V1.9 | | 12-80 |
| ST_ResetZBalance | 重力補償値の補正を初期化します。(4 軸専用) | V1.9 | | | 12-107 |
| ST_SetCompControl | 力制限機能を有効にします。( ６軸専用命令 ) | | V1.9 | | 12-88 |
| ST_SetCompEralw | 力制限時のツール端の位置、姿勢偏差許容値を設定します。( ６軸専用命令 ) | | V1.9 | | 12-102 |
| ST_SetCompFControl | 力制限機能を有効にします。( ６軸専用命令 ) | | V1.9 | | 12-89 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| ST_SetCompJLimit | 力制限時の電流制限値を設定します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-98 |
| ST_SetCompRate | 力制限時の柔らかさの割合を設定します。( ６軸専用命令 ) | | V1.9 | | 12-94 |
| ST_SetCompVMode | 力制限時の速度制御モードを設定します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-100 |
| ST_SetCurLmt | 指定した軸のモータ電流値を制限します。 | V1.9 | V1.9 | | 12-81 |
| ST_SetDampRate | 力制限時の粘性割合を設定します。( ６軸専用命令 ) | | V1.9 | | 12-104 |
| ST_SetEralw | 指定した軸の偏差許容値を変更します。 | V1.9 | V1.9 | | 12-84 |
| ST_SetFrcAssist | 力制限時のオフセット力を設定します。( 力制限特殊機能ステートメント )( ６軸専用命令 ) | | V1.9 | | 12-96 |
| ST_SetFrcCoord | 力制限設定座標系を選択します。( ６軸専用命令応 ) | | V1.9 | | 12-91 |
| ST_SetFrcLimit | 力制限割合を設定します。( ６軸専用命令 ) | | V1.9 | | 12-92 |
| ST_SetGravity | 各関節の静荷重（重力トルク）を補正し、重力バランスを設定します。 | V1.9 | V1.9 | | 12-77 |
| ST_SetGrvOffset | 各関節の重力トルクより重力補償値を補正します。 | V1.9 | V1.9 | | 12-79 |
| ST_SetZBalance | Z 軸、T 軸の重力補償値を設定します。(4 軸専用) | V1.9 | | | 12-106 |
| STARTLOG | サーボ制御ログの記録を開始します。 | ◎ | ◎ | | 19-4 |
| STARTTRACELOG | トレースログの記録を開始します。 | V2.7 | V2.7 | | 19-7 |
| STARTVARLOG | 変数ログの記録を開始します。 | V2.7 | V2.7 | | 19-10 |
| STATUS | プログラムの状態を得ます。 | ◎ | ◎ | | 14-6 |
| STOP | プログラムの実行を終了します。 | ◎ | ◎ | ◎ | 11-2 |
| STOPEND | 連続起動または CYCLE オプション付きで起動されたプログラムをサイクル停止させるステートメントです。このステートメントを含むプログラムを、サイクル起動した場合は、このステートメントを実行しても、動作に影響しません。 | ◎ | ◎ | | 11-3 |
| STOPLOG | サーボ制御ログの記録を停止します。 | ◎ | ◎ | | 19-6 |
| STOPTRACELOG | トレースログの記録を終了します。 | V2.7 | V2.7 | | 19-8 |
| STOPVARLOG | 変数ログの記録を終了します。 | V2.7 | V2.7 | | 19-11 |
| STR$ | 数値から文字列に変換します。 | ◎ | ◎ | ◎ | 15-64 |
| STRPOS | 文字列の位置を得ます。 | ◎ | ◎ | ◎ | 15-63 |
| SUSPEND | タスクを一時停止します。 | ◎ | ◎ | | 14-4 |
| SUSPENDALL | 特権タスク以外の全てのプログラムを停止します。 | V1.98 | V1.98 | | 14-8 |
| SYSSTATE | コントローラのステータスを取得します。 | V1.98 | V1.98 | | 19-16 |
| **T** | | | | | |
| T2J | 同次変換型からジョイント型に変換します。 | ◎ | ◎ | | 15-33 |
| T2P | 同次変換型からポジション型に変換します。 | ◎ | ◎ | | 15-34 |
| TAKEARM | アームグループを取得します。アームグループ NO を省略した場合、アームグループ 0 （ロボットのみ）の制御権を取得します。取得時、内部速度・加速度・減速度を 100 に設定します。取得するアームグループがロボット軸を含む場合は、ツール座標、ワーク座標を原点に戻します。 | ◎ | ◎ | | 14-22 |
| TAKEARMSTATE | アームグループ制御権の取得状態を返します。 0：未取得、1：取得済み | V2.2 | V2.2 | | 14-12 |
| TAKESEM | 指定したセマフォ ID を持つセマフォを取得します。 | ◎ | ◎ | | 14-21 |
| TAKEVIS | 視覚処理権を取得します。 | ◎ | ◎ | | 14-28 |
| TAN | 正接 ( タンジェント ) を得ます。 | ◎ | ◎ | ◎ | 15-18 |
| TIME$ | 現在の時刻を得ます。 | ◎ | ◎ | | 17-2 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| TIMER | 経過時間を得ます。 | ◎ | ◎ | | 17-3 |
| TINV | 同次変換型の逆行列を計算します。 | ◎ | ◎ | | 15-35 |
| TOOL | ツール座標系を宣言します。 | ○ | ○ | | 9-8 |
| TOOLPOS | ツール座標系をポジション型で返します。 | ◎ | ◎ | | 15-50 |
| TRUE | ブール値の真 (1) の値を与えます。 | ◎ | ◎ | ◎ | 16-5 |
| **V** | | | | | |
| VAL | 文字列から数値へ変換します。 | ◎ | ◎ | ◎ | 15-65 |
| VER$ | 各モジュールのバージョンを取得します。 | ◎ | ◎ | ◎ | 19-3 |
| VISBINA | 画面を 2 値化処理します。 | ◎ | ◎ | ◎ | 21-49 |
| VISBINAR | 画面を 2 値化表示します。 | ◎ | ◎ | ◎ | 21-51 |
| VISBRIGHT | 描画する際の輝度値を指定します。 | ◎ | ◎ | ◎ | 21-25 |
| VISCAMOUT | カメラからの映像をモニタに表示します。 | ◎ | ◎ | ◎ | 21-7 |
| VISCIRCLE | 画面に円を描画します。 | ◎ | ◎ | ◎ | 21-33 |
| VISCLS | モードで指定した画面を指定輝度で塗りつぶし（クリア）します。 | ◎ | ◎ | ◎ | 21-26 |
| VISCOPY | 画面をコピーします。 | ◎ | ◎ | ◎ | 21-56 |
| VISCROSS | 画面に十字マークを描画します。 | ◎ | ◎ | ◎ | 21-37 |
| VISDEFCHAR | 文字のサイズと表示方法を指定します。 | ◎ | ◎ | ◎ | 21-40 |
| VISDEFTABLE | カメラ映像の取り込み、画像出力の際のルックアップテーブルデータを設定します。 | ◎ | ◎ | ◎ | 21-10 |
| VISEDGE | ウィンドウ内のエッジを計測します。 | ◎ | ◎ | ◎ | 21-62 |
| VISELLIPSE | 画面に楕円を描画します。 | ◎ | ◎ | ◎ | 21-34 |
| VISFILTER | 画像にフィルタ処理を行ないます。 | ◎ | ◎ | ◎ | 21-52 |
| VISGETNUM | 画像処理結果を格納メモリから取得します。 | ◎ | ◎ | ◎ | 21-97 |
| VISGETP | 格納メモリ（処理画面）から指定座標の輝度を取得します。 | ◎ | ◎ | ◎ | 21-44 |
| VISGETSTR | コード認識結果を取得します。 | ◎ | ◎ | ◎ | 21-98 |
| VISHIST | 画面のヒストグラム（輝度分布）を得ます。 | ◎ | ◎ | ◎ | 21-45 |
| VISLEVEL | ヒストグラム結果に基づき 2 値化レベルを求めます。 | ◎ | ◎ | ◎ | 21-47 |
| VISLINE | 画面に直線を描画します。 | ◎ | ◎ | ◎ | 21-29 |
| VISLOC | 文字の表示位置を指定します。 | ◎ | ◎ | ◎ | 21-38 |
| VISMASK | 画像間演算をします。 | ◎ | ◎ | ◎ | 21-54 |
| VISMEASURE | ウィンドウ内の特徴（面積、重心、主軸角）を計測します。 | ◎ | ◎ | ◎ | 21-57 |
| VISOVERLAY | 描画画面の情報をモニタに表示します。 | ◎ | ◎ | ◎ | 21-9 |
| VISPLNOUT | 格納メモリの画像をモニタに表示します。 | ◎ | ◎ | ◎ | 21-8 |
| VISPOSX | 画像処理結果（X 座標）を格納メモリから取得します。 | ◎ | ◎ | ◎ | 21-99 |
| VISPOSY | 画像処理結果（Y 座標）を格納メモリから取得します。 | ◎ | ◎ | ◎ | 21-100 |
| VISPRINT | 画面に文字・数字を表示します。 | ◎ | ◎ | ◎ | 21-42 |
| VISPROJ | ウィンドウ内の投影データを計測します。 | ◎ | ◎ | ◎ | 21-60 |
| VISPTP | 画面に 2 点間を結ぶ直線を描画します。 | ◎ | ◎ | ◎ | 21-30 |
| VISPUTP | 画面に点を描画します。 | ◎ | ◎ | ◎ | 21-28 |
| VISREADQR | QR コードを読み取ります。 | ◎ | ◎ | ◎ | 21-67 |
| VISRECT | 画面に矩形を描画します。 | ◎ | ◎ | ◎ | 21-31 |
| VISREFCAL | CAL（視覚－ロボット座標変換）データを取得します。 | ◎ | ◎ | ◎ | 21-102 |
| VISREFHIST | ヒストグラム結果を読み出します。 | ◎ | ◎ | ◎ | 21-46 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|
| VISREFTABLE | ルックアップテーブルのデータを参照します。 | ◎ | ◎ | ◎ | 21-12 |
| VISSCREEN | 描画する画面を指定します。 | ◎ | ◎ | ◎ | 21-23 |
| VISSECT | 画面に扇を描画します。 | ◎ | ◎ | ◎ | 21-36 |
| VISSTATUS | 各命令の処理結果をモニタします。 | ◎ | ◎ | ◎ | 21-101 |
| VISWORKPLN | 処理対象の格納メモリ（処理画面）を指定します。 | ◎ | ◎ | ◎ | 21-43 |
| **W** | | | | | |
| WAIT | 条件プログラム停止を行ないます。 | ◎ | ◎ | | 12-62 |
| WAITERROR | エラー発生を検知します。(特権タスク専用命令) | V2.2 | V2.2 | | 14-33 |
| WHILE...WEND | 前判定反復を行ないます。 | ◎ | ◎ | ◎ | 11-16 |
| WINDCLR | 設定したウィンドウ情報を消去します。 | ◎ | ◎ | ◎ | 21-18 |
| WINDCOPY | ウィンドウのデータをコピーします。 | ◎ | ◎ | ◎ | 21-19 |
| WINDDISP | 設定したウィンドウを描画します。 | ◎ | ◎ | ◎ | 21-22 |
| WINDMAKE | 画像処理する範囲を指定します。 | ◎ | ◎ | ◎ | 21-13 |
| WINDREF | ウィンドウの情報を得ます。 | ◎ | ◎ | ◎ | 21-21 |
| WORK | ユーザ座標系を宣言します。 | ○ | ○ | | 9-9 |
| WORKATTRIBUTE | 指定したワーク座標系の属性値を取得します | V3.2 | V3.2 | | 15-51 |
| WORKPOS | ユーザ座標系をポジション型で返します。 | ◎ | ◎ | | 15-51 |
| WRITE | RS232C および Ethernet ポートからデータを出力します。 | ◎ | ◎ | ◎ | 13-14 |

# 機能別コマンド一覧

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| **宣言文** | | | | | | |
| プログラム名 | PROGRAM | プログラム名を宣言します。 | ◎ | ◎ | ◎ | 9-1 |
| 干渉エリア座標 | AREA | 干渉チェックを行なうエリアを宣言します。 | ○ | ○ | | 9-2 |
| | POSTUREAREA | 高感度位置姿勢検知機能の設定を宣言します。 | V3.0 | V3.0 | | 9-4 |
| ユーザ関数 | DEF FN | ユーザ定義関数を宣言します。 | ◎ | ◎ | ◎ | 9-6 |
| ホーム座標 | HOME | 任意の座標をホームポジションとして宣言します。 | ○ | ○ | | 9-7 |
| ツール座標 | TOOL | ツール座標系を宣言します。 | ○ | ○ | | 9-8 |
| ワーク座標 | WORK | ユーザ座標系を宣言します。 | ○ | ○ | | 9-9 |
| ローカル変数 | DEFINT | I 型変数（整数型変数）を宣言します。整数の範囲は -2147483648 ～ 2147483647 です。 | ◎ | ◎ | ◎ | 9-11 |
| | DEFSNG | F 型変数（単精度実数型変数）を宣言します。単精度実数の範囲は -3.4E+38 ～ 3.4E+38 です。 | ◎ | ◎ | ◎ | 9-11 |
| | DEFDBL | D 型変数（倍精度実数型変数）を宣言します。倍精度実数の範囲は -1.7D+308 ～ 1.7D+308 です。 | ◎ | ◎ | ◎ | 9-12 |
| | DEFSTR | S 型変数（文字列型変数）を宣言します。文字列の長さは、243 文字までです。 | ◎ | ◎ | ◎ | 9-12 |
| | DEFVEC | V 型変数（ベクトル型変数）を宣言します。 | ◎ | ◎ | | 9-13 |
| | DEFPOS | P 型変数（ポジション型変数）を宣言します。 | ○ | ○ | | 9-14 |
| | DEFJNT | J 型変数（ジョイント型変数）を宣言します。 | ○ | ○ | | 9-15 |
| | DEFTRN | T 型変数（同次変換型変数）を宣言します。 | ◎ | ◎ | | 9-16 |
| | DEFIO | 入出力ポートに対応する IO 変数（I/O 変数）を宣言します。 | ◎ | ◎ | ◎ | 9-17 |
| 配列 | DIM | 配列を宣言します。 | ◎ | ◎ | ◎ | 9-19 |
| フォルダー機能 | FOLDER | 外部プログラムからアクセス可能なローカル変数を宣言します。 | V2.2 | V2.2 | | 9-20 |
| | EXTERN | 他のプログラムで定義されている FOLDER 変数を参照することを宣言します。 | V2.2 | V2.2 | | 9-23 |
| **代入文** | | | | | | |
| 変数 | LET | 変数に値を代入します。 | ○ | ○ | ○ | 10-1 |
| ベクトル | LETA | 同次変換型のアプローチベクトルへ代入します。 | | ◎ | | 10-2 |
| | LETO | 同次変換型のオリエントベクトルへ代入します。 | ◎ | ◎ | | 10-3 |
| | LETP | ポジション型または同次変換型の位置ベクトルへ代入します。 | ◎ | ◎ | | 10-4 |
| 形態 | LETF | ポジション型または同次変換型の形態成分へ代入します。 | ◎ | ◎ | | 10-5 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| リンク角 | LETJ | ジョイント型の指定リンク角へ代入します。 | ◎ | ◎ | | 10-6 |
| 姿勢 | LETR | ポジション型の 3 つの回転成分へ代入します。 | | ◎ | | 10-7 |
| 回転成分 | LETRX | ポジション型の X 軸回転成分へ代入します。 | | ◎ | | 10-8 |
| | LETRY | ポジション型の Y 軸回転成分へ代入します。 | | ◎ | | 10-9 |
| | LETRZ | ポジション型の Z 軸回転成分へ代入します。 | | ◎ | | 10-10 |
| | LETT | ポジション型の T 成分へ代入します。 | ◎ | | | 10-11 |
| 軸成分 | LETX | ベクトル型／ポジション型／同次変換型の X 軸成分へ代入します。 | ◎ | ◎ | | 10-12 |
| | LETY | ベクトル型／ポジション型／同次変換型の Y 軸成分へ代入します。 | ◎ | ◎ | | 10-13 |
| | LETZ | ベクトル型／ポジション型／同次変換型の Z 軸成分へ代入します。 | ◎ | ◎ | | 10-14 |
| **フロー制御文** | | | | | | |
| プログラムの停止 | END | プログラムによる動作の終了を宣言します。 | ◎ | ◎ | ◎ | 11-1 |
| | STOP | プログラムの実行を終了します。 | ◎ | ◎ | ◎ | 11-2 |
| | STOPEND | 連続起動または CYCLE オプション付きで起動されたプログラムをサイクル停止させるステートメントです。このステートメントを含むプログラムを、サイクル起動した場合は、このステートメントを実行しても、動作に影響しません。 | ◎ | ◎ | | 11-3 |
| 呼び出し | CALL | プログラムを呼び出し、実行します。 | ◎ | ◎ | ◎ | 11-4 |
| | GOSUB | サブルーチンを呼び出します。 | ◎ | ◎ | ◎ | 11-6 |
| | ON...GOSUB | 式の値に応じて、対応するサブルーチンの呼び出しを行ないます。 | ◎ | ◎ | ◎ | 11-7 |
| | RETURN | サブルーチンから復帰します。 | ◎ | ◎ | ◎ | 11-8 |
| 繰り返し | DO...LOOP | 判定反復（繰り返し）を行ないます。 | ◎ | ◎ | ◎ | 11-9 |
| | EXIT DO | DO ～ LOOP からの強制脱出を行ないます。 | ◎ | ◎ | ◎ | 11-11 |
| | FOR...NEXT | FOR ～ NEXT までの区間中にある一連の命令を繰り返して実行します。 | ◎ | ◎ | ◎ | 11-12 |
| | EXIT FOR | FOR ～ NEXT からの強制脱出を行ないます。 | ◎ | ◎ | ◎ | 11-14 |
| | REPEAT...UNTIL | 後判定反復を行ないます。 | ◎ | ◎ | ◎ | 11-15 |
| | WHILE...WEND | 前判定反復を行ないます。 | ◎ | ◎ | ◎ | 11-16 |
| 条件分岐 | IF...END IF | IF ～ END IF 間の条件式の条件判断を行ないます。 | ◎ | ◎ | ◎ | 11-17 |
| | IF...THEN...ELSE | 論理式の条件判断を行ないます。 | ◎ | ◎ | ◎ | 11-18 |
| | SELECT CASE | 複数条件判断を行ないます。 | ◎ | ◎ | ◎ | 11-19 |
| 無条件分岐 | GOTO | プログラムの分岐を無条件で行ないます。 | ◎ | ◎ | ◎ | 11-21 |
| | ON...GOTO | 式の値による無条件分岐を行ないます。 | ◎ | ◎ | ◎ | 11-22 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| コメント | REM | コメントの記述をします。 | ◎ | ◎ | ◎ | 11-23 |
| **ロボット制御文** | | | | | | |
| 動作制御 | APPROACH | ツール座標系指定の絶対動作を行ないます。 | ○ | ○ | | 12-1 |
| | DEPART | ツール座標系指定で相対動作を行ないます。 | ○ | ○ | | 12-4 |
| | DRAW | ワーク作業座標系指定で相対動作を行ないます。 | ◎ | ◎ | | 12-6 |
| | DRIVE | 各軸の相対動作を行ないます。 | ◎ | ◎ | | 12-8 |
| | DRIVEA | 各軸の絶対動作を行ないます。 | ◎ | ◎ | | 12-10 |
| | GOHOME | HOME 文で定義したポジション（原点）へ移動します。 | ◎ | ◎ | | 12-12 |
| | MOVE | ロボットを指定座標へ移動します。EX オプション ( 付加軸相対動作 ) または EXA オプション（付加軸絶対動作）を付けることにより、ロボットと付加軸を同期 ( 同時発着 ) で動作させることができます。 | ○ | ○ | | 12-13 |
| | ROTATE | 指定した軸回りの回転動作を行ないます。 | ○ | ○ | | 12-17 |
| | ROTATEH | アプローチベクトルを軸とした、回転動作を行ないます。 | ◎ | ◎ | | 12-19 |
| | CURJNT | ロボットの現在角度を J 型で得ます。 | ○ | ○ | | 12-21 |
| | CURPOS | ツール座標系での現在位置を P 型データで得ます。 | ○ | ○ | | 12-22 |
| | CURTRN | ツール座標系での現在位置を T 型データで得ます。 | ◎ | ◎ | | 12-23 |
| | CUREXJ | 付加軸の現在角度を F 型で取得します。 | V1.5 | V1.5 | | 12-24 |
| | DESTJNT | 現在の動作命令目標位置を J 型で取得します。ロボット停止時は、現在の位置（指令値）を取得します。 | ○ | ○ | | 12-25 |
| | DESTPOS | 現在の動作命令目標位置を P 型で取得します。ロボット停止時は、現在の位置（指令値）を取得します。 | ○ | ○ | | 12-26 |
| | DESTTRN | 現在の動作命令目標位置を T 型で取得します。ロボット停止時は、現在の位置（指令値）を取得します。 | ◎ | ◎ | | 12-27 |
| | DESTEXJ | 付加軸の、現在の動作命令で指定された目標位置を F 型で取得します。停止時は、現在の位置（指令値）を取得します。 | V1.5 | V1.5 | | 12-28 |
| | ARRIVE | 動作命令の全移動距離に対する動作割合を設定する事によって、ロボットが設定した動作割合に到達するまでプログラムを待機させます。 | ◎ | V1.2 | | 12-29 |
| | POSCLR | 軸の現在位置を強制的に 0mm または 0 度にします。 | V1.5 | V1.5 | | 12-31 |
| | SETSPLINEPOINT | 自由曲線動作の通過点を登録します。 | V2.3 | V2.3 | | 12-32 |
| | CLRSPLINEPOINT | 自由曲線動作の通過点をすべて消去します。 | V2.3 | V2.3 | | 12-33 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | GETSPLINEPOINT | 登録済の自由曲線動作の通過点を取り出します。 | V2.3 | V2.3 | | 12-34 |
| 形態制御 | CURFIG | ロボット形態の現在値を得ます。 | ◎ | ◎ | | 12-35 |
| | FIGAPRL | CP 動作可能なアプローチ位置と基準位置の形態を計算します。 | ○ | ○ | | 12-37 |
| | FIGAPRP | PTP 動作可能なアプローチ位置と基準位置の形態を計算します。 | ○ | ○ | | 12-39 |
| 停止制御 | HOLD | プログラムの実行を一時停止します。 | ◎ | ◎ | | 12-41 |
| | HALT | プログラムの実行を停止します。 | ◎ | ◎ | | 12-42 |
| | INTERRUPT ON/OFF | ロボットの動作を中断します。 | ◎ | ◎ | | 12-43 |
| 速度制御 | SPEED | 現在取得しているアームグループ軸の内部移動速度を指定します。 | ◎ | ◎ | | 12-45 |
| | JSPEED | 現在取得しているアームグループ軸の内部軸速度を指定します。 | ◎ | ◎ | | 12-47 |
| | ACCEL | 現在取得しているアームグループ軸の内部加速度、内部減速度を指定します。 | ◎ | ◎ | | 12-48 |
| | JACCEL | 現在取得しているアームグループ軸の、内部軸加速度、内部軸減速度を指定します。 | ◎ | ◎ | | 12-49 |
| | DECEL | 現在取得しているアームグループ軸の内部減速度を指定します。 | ◎ | ◎ | | 12-50 |
| | JDECEL | 現在取得しているアームグループ軸の内部軸減速度を指定します。 | ◎ | ◎ | | 12-51 |
| | CURACC | 現在取得しているアームグループ軸の、内部加速度を取得します。 | ◎ | ◎ | | 12-52 |
| | CURJACC | 現在取得しているアームグループ軸の、内部軸加速度を取得します。 | ◎ | ◎ | | 12-53 |
| | CURDEC | 現在取得しているアームグループ軸の、内部減速度を取得します。 | ◎ | ◎ | | 12-54 |
| | CURJDEC | 現在取得しているアームグループ軸の、内部軸減速度を取得します。 | ◎ | ◎ | | 12-55 |
| | CURJSPD | 現在取得しているアームグループ軸の、内部移動軸速度を取得します。 | ◎ | ◎ | | 12-56 |
| | CURSPD | 現在取得しているアームグループ軸の、内部移動速度を取得します。 | ◎ | ◎ | | 12-57 |
| | CUREXTACC | 外部加速度の現在値を得ます。 | V1.4 | V1.4 | | 12-58 |
| | CUREXTDEC | 外部減速度の現在値を得ます。 | V1.4 | V1.4 | | 12-58 |
| | CUREXTSPD | 外部速度の現在値を得ます。 | V1.4 | V1.4 | | 12-59 |
| | EXTSPEED | 外部速度設定 | V1.98 | V1.98 | | 12-59 |
| 時間制御 | DELAY | 指定した時間の間、プログラムの進行を停止します。 | ◎ | ◎ | ◎ | 12-60 |
| | WAIT | 条件プログラム停止を行ないます。 | ◎ | ◎ | | 12-62 |
| 座標変換 | CHANGETOOL | ツール座標系を変更します。 | ◎ | ◎ | | 12-63 |
| | CHANGEWORK | ユーザ座標系を変更します。 | ◎ | ◎ | | 12-64 |
| | CURTOOL | 現在設定されている TOOL 番号を得ます。 | V1.4 | V1.4 | | 12-65 |
| | CURWORK | 現在設定されている WORK 番号を得ます。 | V1.4 | V1.4 | | 12-66 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| 干渉チェック | SETAREA | 干渉チェックを行なうエリアを選択します。 | ◎ | ◎ | | 12-67 |
| | RESETAREA | 干渉チェックを初期化します。 | ◎ | ◎ | | 12-68 |
| | SETPOSTUREAREA | 高感度位置姿勢検知機能を有効にします。 | V3.0 | V3.0 | | 12-69 |
| | RESETPOSTUREAREA | 高感度位置姿勢検知機能を無効にします。 | V3.0 | V3.0 | | 12-70 |
| サーボ内部データ | GetSrvData | ロボット軸のサーボ内部データを取得します。(Ver. 1.5 以降対応 ) | | | | 12-71 |
| | GetJntData | 指定軸のサーボ内部データを取得します。(Ver. 1.5 以降対応 ) | | | | 12-72 |
| モータ電源 | MOTOR {ON \| OFF} | モータ電源 ON/OFF を行ないます。 | V1.98 | V1.98 | | 12-73 |
| CAL | EXECAL | CAL 実行 | V1.98 | V1.98 | | 12-74 |
| 特殊制御 | ST_aspACLD | 内部負荷条件値を変更します。負荷条件値は、先端負荷質量（g)、負荷重心位置（mm）（注 1）ですべて指定します。 | V1.9 | V1.9 | | 12-75 |
| | ST_aspChange | 最適可搬質量設定モードの内部モードを選択します。 | V1.9 | V1.9 | | 12-76 |
| | ST_SetGravity | 各関節の静荷重（重力トルク）を補正し、重力バランスを設定します。 | V1.9 | V1.9 | | 12-77 |
| | ST_ResetGravity | 重力バランスを無効にします。 | V1.9 | V1.9 | | 12-78 |
| | ST_SetGrvOffset | 各関節の重力トルクより重力補償値を補正します。 | V1.9 | V1.9 | | 12-79 |
| | ST_ResetGrvOffset | 重力補償値の補正を初期化します。 | V1.9 | V1.9 | | 12-80 |
| | ST_SetCurLmt | 指定した軸のモータ電流値を制限します。 | V1.9 | V1.9 | | 12-81 |
| | ST_ResetCurLmt | 指定した軸のモータ電流制限を解除します。 | V1.9 | V1.9 | | 12-83 |
| | ST_SetEralw | 指定した軸の偏差許容値を変更します。 | V1.9 | V1.9 | | 12-84 |
| | ST_ResetEralw | 指定した軸の偏差許容値をデフォルト値に戻します。 | V1.9 | V1.9 | | 12-85 |
| | ST_OnSrvLock | 指定した軸をサーボロック状態にします。（4 軸ロボット専用） | V1.9 | | | 12-86 |
| | ST_OffSrvLock | 指定した軸のサーボロックを解除します。（4 軸ロボット専用） | V1.9 | | | 12-87 |
| | ST_SetCompControl | 力制限機能を有効にします。( 6 軸専用命令 ) | | V1.9 | | 12-88 |
| | ST_SetCompFControl | 力制限機能を有効にします。( 6 軸専用命令 ) | | V1.9 | | 12-89 |
| | ST_ResetCompControl | 力制限機能を無効にします。( 6 軸専用命令 ) | | V1.9 | | 12-90 |
| | ST_SetFrcCoord | 力制限設定座標系を選択します。( 6 軸専用命令応 ) | | V1.9 | | 12-91 |
| | ST_SetFrcLimit | 力制限割合を設定します。( 6 軸専用命令 ) | | V1.9 | | 12-92 |
| | ST_ResetFrcLimit | 力制限割合を初期化します。( 6 軸専用命令 ) | | V1.9 | | 12-93 |
| | ST_SetCompRate | 力制限時の柔らかさの割合を設定します。( 6 軸専用命令 ) | | V1.9 | | 12-94 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | ST_ResetCompRate | 力制限時の柔らかさの割合を初期化します。( ６軸専用命令 ) | | V1.9 | | 12-95 |
| | ST_SetFrcAssist | 力制限時のオフセット力を設定します。( 力制限特殊機能ステートメント )( ６軸専用命令 ) | | V1.9 | | 12-96 |
| | ST_ResetFrcAssist | 力制限時のオフセット力を初期化します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-97 |
| | ST_SetCompJLimit | 力制限時の電流制限値を設定します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-98 |
| | ST_ResetCompJLimit | 力制限時の電流制限値を初期化します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-99 |
| | ST_SetCompVMode | 力制限時の速度制御モードを設定します。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-100 |
| | ST_ResetCompVMode | 力制限時の速度制御モードを無効にします。( 力制限特殊機能 ) ( ６軸専用命令 ) | | V1.9 | | 12-101 |
| | ST_SetCompEralw | 力制限時のツール端の位置、姿勢偏差許容値を設定します。( ６軸専用命令 ) | | V1.9 | | 12-102 |
| | ST_ResetCompEralw | 力制限時のツール端の位置、姿勢偏差許容値を初期化します。( ６軸専用命令 ) | | V1.9 | | 12-103 |
| | ST_SetDampRate | 力制限時の粘性割合を設定します。( ６軸専用命令 ) | | V1.9 | | 12-104 |
| | ST_ResetDampRate | 力制限時の粘性割合を初期化します。( ６軸専用命令 ) | | V1.9 | | 12-105 |
| | ST_SetZBalance | Z 軸、T 軸の重力補償値を設定します。(4 軸専用) | V1.9 | | | 12-106 |
| | ST_ResetZBalance | 重力補償値の補正を初期化します。(4 軸専用) | V1.9 | | | 12-107 |
| **入出力制御文** | | | | | | |
| I/O ポート | IN | I/O 変数で示される I/O ポートからデータを読み込みます。 | ◎ | ◎ | ◎ | 13-1 |
| | OUT | I/O 変数で示される I/O ポートにデータを出力します。 | ◎ | ◎ | ◎ | 13-2 |
| | IOBLOCK ON/OFF | 動作命令を実行中に、I/O 命令や演算命令などの非動作命令を並列に実行します。 | ◎ | ◎ | | 13-3 |
| | SET | I/O ポートを ON にします。 | ◎ | ◎ | ◎ | 13-5 |
| | RESET | I/O ポートを OFF にします。 | ◎ | ◎ | ◎ | 13-7 |
| | DETECT ON/OFF | ハンド入力の立上り及び立下りを検知し、その時のロボットの位置を変数に格納します。 | V3.2 | V3.2 | | 13-8 |
| RS232C および Ethernet ポート | INPUT | RS232C および Ethernet ポートからのデータを得ます。 | ◎ | ◎ | ◎ | 13-10 |
| | LINEINPUT | RS232C および Ethernet ポートからデリミタまでのデータを読み込み、文字列型変数に代入します。 | ◎ | ◎ | ◎ | 13-12 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | PRINT | RS232C および Ethernet ポートからデータを出力します。 | ◎ | ◎ | ◎ | 13-13 |
| | WRITE | RS232C および Ethernet ポートからデータを出力します。 | ◎ | ◎ | ◎ | 13-14 |
| | FLUSH | 入力バッファをクリアします。 | ◎ | ◎ | ◎ | 13-15 |
| シリアルバイナリ通信（RS-232C および Ethernet ポート） | PRINTB | １バイトデータを RS-232C および Ethernet ポートに出力します。 | V1.5 | V1.5 | V1.9 | 13-16 |
| | INPUTB | １バイトデータを RS-232C および Ethernet ポートから入力します。 | V1.5 | V1.5 | V1.9 | 13-17 |
| | LPRINTB | 複数バイトデータを RS-232C および Ethernet ポートに出力します。 | V1.5 | V1.5 | V1.9 | 13-18 |
| | LINPUTB | 複数バイトデータを RS-232C および Ethernet ポートから入力します。 | V1.5 | V1.5 | V1.9 | 13-19 |
| | com_encom | RS-232C ポートまたは Ethernet クライアントポートを使ってバイナリ通信行うための準備をします。RS-232C ポートの場合は、バイナリ通信のためにポートを占有します。Ethernet クライアントポートを使う場合はクライアントポートのオープンを行いサーバとの間で接続を確立します。 | V1.5 | V1.5 | V1.9 | 13-20 |
| | com_discom | RS-232C ポートをバイナリ通信終了のため開放します。（COM ポート開放）Ethernet クライアントポート使用時にクライアントポートのクローズを おこない接続を切断します。 | V1.5 | V1.5 | V1.9 | 13-21 |
| | com_state | RS-232C および Ethernet ポートの状態を取得します。 | V1.5 | V1.5 | V1.9 | 13-22 |
| ティーチングペンダント | PRINTMSG | メッセージをキャプションとアイコンを付けて、ティーチングペンダントのカラー液晶に表示します。 | ◎ | ◎ | | 13-23 |
| | PRINTDBG | デバッグウィンドウにデータを出力します。 | ◎ | ◎ | | 13-24 |
| | BUZZER | ブザーを鳴らします。 | ◎ | ◎ | | 13-24 |
| | PRINTWARNING | ペンダント警告メッセージ表示部分へメッセージを表示します。 | V2.2 | V2.2 | | 13-25 |
| | PRINTLBL | ユーザ定義ボタンにラベル（キャプション）を設定します。 | ◎ | ◎ | | 13-25 |
| TP 簡易操作盤 | set_button | ボタンパラメータの設定を行ないます。 | V1.5 | V1.5 | | 13-28 |
| | set_page | ページパラメータの設定を行ないます。 | V1.5 | V1.5 | | 13-32 |
| | change_bCap | ボタンのキャプション設定を行ないます。 | V1.5 | V1.5 | | 13-34 |
| | change_pCap | ページのキャプション設定を行ないます。 | V1.5 | V1.5 | | 13-35 |
| | disp_page | ページの表示を行ないます。 | V1.5 | V1.5 | | 13-36 |
| **マルチタスク制御文** | | | | | | |
| タスク制御 | RUN | 別プログラムを並列起動します。 | ◎ | ◎ | | 14-1 |
| | KILL | タスクを強制終了します。 | ◎ | ◎ | | 14-3 |
| | SUSPEND | タスクを一時停止します。 | ◎ | ◎ | | 14-4 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | DEFEND | タスクを保護します。 | ◎ | ◎ | | 14-5 |
| | STATUS | プログラムの状態を得ます。 | ◎ | ◎ | | 14-6 |
| | SUSPENDALL | 特権タスク以外の全てのプログラムを停止します。 | V1.98 | V1.98 | | 14-8 |
| | KILLALL | 特権タスク以外の全てのタスクを強制終了します。(プログラムリセットに相当) | V1.98 | V1.98 | | 14-9 |
| | CONTINUERUN | コンティニュー起動を行います。 | V1.98 | V1.98 | | 14-10 |
| | ROBOTSTOP | ロボット停止を行います。 | V1.98 | V1.98 | | 14-11 |
| | TAKEARMSTATE | アームグループ制御権の取得状態を返します。 0：未取得、 1：取得済み | V2.2 | V2.2 | | 14-12 |
| | LOCKSTATE | マシンロックの状態を取得します。 0：OFF、 1：ON | V2.2 | V2.2 | | 14-12 |
| | DEADMANSTATE | デッドマンスイッチの状態を取得します。 0：OFF、 1：ON | V2.2 | V2.2 | | 14-13 |
| | SEMIDSTATE | 指定したセマフォ ID が有効か無効かを返します。 | V2.2 | V2.2 | | 14-14 |
| セマフォ | CREATESEM | セマフォを生成します。 | ◎ | ◎ | | 14-15 |
| | DELETESEM | セマフォを削除します。 | ◎ | ◎ | | 14-18 |
| | FLUSHSEM | セマフォ待ちタスクを解放します。 | ◎ | ◎ | | 14-19 |
| | GIVESEM | セマフォ待ちタスクを解放します。 | ◎ | ◎ | | 14-20 |
| | TAKESEM | 指定したセマフォ ID を持つセマフォを取得します。 | ◎ | ◎ | | 14-21 |
| アームセマフォ | TAKEARM | アームグループを取得します。アームグループ NO を省略した場合、アームグループ 0 (ロボットのみ) の制御権を取得します。取得時、内部速度・加速度・減速度を 100 に設定します。取得するアームグループがロボット軸を含む場合は、ツール座標、ワーク座標を原点に戻します。 | ◎ | ◎ | | 14-22 |
| | GIVEARM | 現在取得しているアームグループを解放します。アームグループ 0 (ロボットのみ) の場合、ロボット制御権を開放します。 | ◎ | ◎ | | 14-27 |
| | TAKEVIS | 視覚処理権を取得します。 | ◎ | ◎ | | 14-28 |
| | GIVEVIS | 視覚処理権を解放します。 | ◎ | ◎ | | 14-29 |
| 特権タスク | INIT | 特権タスクパラメータの状態により、モータ ON、CAL、スピード設定を行います。 | V1.7 | V1.7 | | 14-30 |
| | SETOCCUPATIONTIME | 特権タスクの占有時間を変更します。 | V2.0 | V2.0 | | 14-31 |
| | INITWAITERR | エラー検知を初期化します。(特権タスク専用命令) | V2.2 | V2.2 | | 14-32 |
| | WAITERROR | エラー発生を検知します。(特権タスク専用命令) | V2.2 | V2.2 | | 14-33 |
| | CURERRSTATUS | エラー発生状態を返します。(特権タスク専用命令) | V2.2 | V2.2 | | 14-34 |
| **関数** | | | | | | |
| 算術関数 | ABS | 数式の値の絶対値を得ます。 | ◎ | ◎ | ◎ | 15-1 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | EXP | 自然対数を基数とする指数関数を得ます。 | ◎ | ◎ | ◎ | 15-2 |
| | INT | 指定された値を超えない最大の整数値を得ます。 | ◎ | ◎ | ◎ | 15-3 |
| | LOG | 自然対数を得ます。 | ◎ | ◎ | ◎ | 15-4 |
| | LOG10 | 常用対数を得ます。 | ◎ | ◎ | ◎ | 15-5 |
| | POW | べき乗を求めます。 | ◎ | ◎ | ◎ | 15-6 |
| | MAX | 最大値を抽出します。 | ◎ | ◎ | ◎ | 15-7 |
| | MIN | 最小値を抽出します。 | ◎ | ◎ | ◎ | 15-8 |
| | RND | 0 以上 1 以下の乱数を発生します。 | ◎ | ◎ | ◎ | 15-9 |
| | SGN | 符号を調べます。 | ◎ | ◎ | ◎ | 15-10 |
| | SQR | 平方根を得ます。 | ◎ | ◎ | ◎ | 15-11 |
| 三角関数 | ACOS | 逆余弦 ( アークコサイン ) を得ます。 | ◎ | ◎ | ◎ | 15-12 |
| | ASIN | 逆正弦 ( アークサイン ) を得ます。 | ◎ | ◎ | ◎ | 15-13 |
| | ATN | 逆正接 ( アークタンジェント ) を得ます。 | ◎ | ◎ | ◎ | 15-14 |
| | ATN2 | 数式 1 を数式 2 で除算した逆正接 ( アークタンジェント ) を得ます。 | ◎ | ◎ | ◎ | 15-15 |
| | COS | 余弦 ( コサイン ) を得ます。 | ◎ | ◎ | ◎ | 15-16 |
| | SIN | 正弦 ( サイン ) を得ます。 | ◎ | ◎ | ◎ | 15-17 |
| | TAN | 正接 ( タンジェント ) を得ます。 | ◎ | ◎ | ◎ | 15-18 |
| 角度変換 | DEGRAD | 単位をラジアンに変換します。 | ◎ | ◎ | ◎ | 15-19 |
| | RAD | ラジアンで与えた数値を角度に変換します。 | ◎ | ◎ | ◎ | 15-20 |
| | RADDEG | 単位を度に変換します。 | ◎ | ◎ | ◎ | 15-21 |
| 速度変換 | MPS | 指定した速度値（mm/sec）を CP 動作時の最大内部速度に対する割合（%）に変換します。 | ◎ | ◎ | | 15-22 |
| | RPM | 指定した軸番号の回転数 (rpm) を PTP 動作時の最大内部速度に対する割合（%）に変換します。 | V2.8 | V2.8 | | 15-23 |
| 時間関数 | SEC | 秒の単位で与えた数値をミリ秒に変換します。 | ◎ | ◎ | ◎ | 15-24 |
| ベクトル | AVEC | アプローチベクトルを抽出します。 | ◎ | ◎ | | 15-25 |
| | OVEC | オリエントベクトルを抽出します。 | ◎ | ◎ | | 15-26 |
| | PVEC | 位置ベクトルを抽出します。 | ○ | ○ | | 15-27 |
| | MAGNITUDE | ベクトルの大きさを得ます。 | ◎ | ◎ | | 15-28 |
| ポーズデータ型変換 | J2P | ジョイント型からポジション型に変換します。 | ○ | ○ | | 15-29 |
| | J2T | ジョイント型から同次変換型に変換します。 | ○ | ○ | | 15-30 |
| | P2J | ポジション型からジョイント型に変換します。 | ○ | ○ | | 15-31 |
| | P2T | ポジション型から同次変換型に変換します。 | ○ | ○ | | 15-32 |
| | T2J | 同次変換型からジョイント型に変換します。 | ◎ | ◎ | | 15-33 |
| | T2P | 同次変換型からポジション型に変換します。 | ◎ | ◎ | | 15-34 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | TINV | 同次変換型の逆行列を計算します。 | ◎ | ◎ | | 15-35 |
| | NORMTRN | 同次変換型の正規化計算を行います。 | V1.8 | V1.8 | | 15-36 |
| 距離抽出 | DIST | 2 点間の距離を返します。 | V1.8 | V1.8 | | 15-37 |
| 形態成分 | FIG | 形態を抽出します。 | ○ | ○ | | 15-38 |
| 角度成分 | JOINT | ジョイント型座標から角度を抽出します。 | ○ | ○ | | 15-39 |
| 軸成分 | POSX | X 成分を抽出します。 | ○ | ○ | | 15-40 |
| | POSY | Y 成分を抽出します。 | ○ | ○ | | 15-41 |
| | POSZ | Z 成分を抽出します。 | ○ | ○ | | 15-42 |
| 回転成分 | POSRX | X 軸回転成分を抽出します。 | | ◎ | | 15-43 |
| | POSRY | Y 軸回転成分を抽出します。 | | ◎ | | 15-44 |
| | POSRZ | Z 軸回転成分を抽出します。 | | ◎ | | 15-45 |
| | POST | T 軸回転成分を抽出します。 | ◎ | | | 15-46 |
| 姿勢成分 | RVEC | 姿勢を抽出します。 | | ◎ | | 15-47 |
| 位置関数 | AREAPOS | 干渉チェックの行なわれる領域の中心位置と直方体の方向をポジション型で返します。 | ◎ | ◎ | | 15-48 |
| | AREASIZE | 干渉チェックの行なわれる領域を定義する、直方体の大きさ（各辺の長さ）をベクトル型で返します。 | ◎ | ◎ | | 15-49 |
| | TOOLPOS | ツール座標系をポジション型で返します。 | ◎ | ◎ | | 15-50 |
| | WORKPOS | ユーザ座標系をポジション型で返します。 | ◎ | ◎ | | 15-51 |
| | WORKATTRIBUTE | 指定したワーク座標系の属性値を取得します | V3.2 | V3.2 | | 15-51 |
| 文字列関数 | ASC | 文字コードへ変換します。 | ◎ | ◎ | ◎ | 15-52 |
| | BIN$ | 数式の値を 2 進数の文字列へ変換します。 | ◎ | ◎ | ◎ | 15-53 |
| | CHR$ | ASCII コードを文字に変換します。 | ◎ | ◎ | ◎ | 15-54 |
| | SPRINTF$ | 式を指定したフォーマットに変換し、文字列として返します。 | ◎ | ◎ | ◎ | 15-55 |
| | HEX$ | 10 進数から 16 進数へ変換した値を文字列として得ます。 | ◎ | ◎ | ◎ | 15-57 |
| | LEFT$ | 左部分文字列を抽出します。 | ◎ | ◎ | ◎ | 15-58 |
| | LEN | 文字列の長さをバイト数で得ます。 | ◎ | ◎ | ◎ | 15-59 |
| | MID$ | 文字列から指定した文字数分の文字列を取り出します。 | ◎ | ◎ | ◎ | 15-60 |
| | ORD | 文字コードへ変換します。 | ◎ | ◎ | ◎ | 15-61 |
| | RIGHT$ | 文字列の右部分を抽出します。 | ◎ | ◎ | ◎ | 15-62 |
| | STRPOS | 文字列の位置を得ます。 | ◎ | ◎ | ◎ | 15-63 |
| | STR$ | 数値から文字列に変換します。 | ◎ | ◎ | ◎ | 15-64 |
| | VAL | 文字列から数値へ変換します。 | ◎ | ◎ | ◎ | 15-65 |
| **定数** | | | | | | |
| 組み込み定数 | OFF | OFF(0) の値を与えます。 | ◎ | ◎ | ◎ | 16-1 |
| | ON | ON(1) の値を与えます。 | ◎ | ◎ | ◎ | 16-2 |
| | PI | π の値を与えます。 | ◎ | ◎ | ◎ | 16-3 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | FALSE | ブール値の偽 (0) の値を与えます。 | ◎ | ◎ | ◎ | 16-4 |
| | TRUE | ブール値の真 (1) の値を与えます。 | ◎ | ◎ | ◎ | 16-5 |
| **時刻／日付制御** | | | | | | |
| 時刻／日付 | DATE$ | 現在の日付を得ます。 | ◎ | ◎ | | 17-1 |
| | TIME$ | 現在の時刻を得ます。 | ◎ | ◎ | | 17-2 |
| | TIMER | 経過時間を得ます。 | ◎ | ◎ | | 17-3 |
| **エラー制御** | | | | | | |
| エラー情報 | ERRMSG$ | エラーメッセージを与えます。 | ◎ | ◎ | ◎ | 18-1 |
| | SETERR | ユーザ定義エラーを I 型変数領域にセットします。 | V1.98 | V1.98 | | 18-1 |
| | GETERR | エラー格納機能で宣言されたバッファからエラーコードを取得します。 | V1.98 | V1.98 | | 18-2 |
| | CLRERR | エラーをクリアします。 | V1.98 | V1.98 | | 18-3 |
| | GETERRLVL | エラーコードのレベルを与えます。 | V1.98 | V1.98 | | 18-4 |
| **システム情報** | | | | | | |
| システム | GETENV | システムの環境設定値を取得します。 | ◎ | ◎ | ◎ | 19-1 |
| | LETENV | システムの環境設定値を設定します。 | ◎ | ◎ | ◎ | 19-2 |
| | VER$ | 各モジュールのバージョンを取得します。 | ◎ | ◎ | ◎ | 19-3 |
| | GETLANGUAGE | 現在の言語設定を取得します。 | V2.2 | V2.2 | | 19-3 |
| ログ | STARTLOG | サーボ制御ログの記録を開始します。 | ◎ | ◎ | | 19-4 |
| | CLEARLOG | サーボ制御ログの記録を初期化します。 | ◎ | ◎ | | 19-5 |
| | STOPLOG | サーボ制御ログの記録を停止します。 | ◎ | ◎ | | 19-6 |
| | STARTTRACELOG | トレースログの記録を開始します。 | V2.7 | V2.7 | | 19-7 |
| | STOPTRACELOG | トレースログの記録を終了します。 | V2.7 | V2.7 | | 19-8 |
| | CLEARTRACELOG | トレースログデータをクリアします。 | V2.7 | V2.7 | | 19-9 |
| | STARTVARLOG | 変数ログの記録を開始します。 | V2.7 | V2.7 | | 19-10 |
| | STOPVARLOG | 変数ログの記録を終了します。 | V2.7 | V2.7 | | 19-11 |
| | CLEARVARLOG | 変数ログデータをクリアします。 | V2.7 | V2.7 | | 19-12 |
| 動作モード | CHGEXTMODE | 外部自動モード切替を行ないます。 | V1.98 | V1.98 | | 19-13 |
| | CHGINTMODE | 内部自動モード切替を行ないます。 | V1.98 | V1.98 | | 19-14 |
| | CUROPTMODE | 動作モードを取得します。 | V1.98 | V1.98 | | 19-15 |
| | SYSSTATE | コントローラのステータスを取得します。 | V1.98 | V1.98 | | 19-16 |
| **プリプロセッサ** | | | | | | |
| 記号定数・マクロ定義 | #define | プログラム中の指定した定数またはマクロ名を、指定文字列で置き換えます。 | ◎ | ◎ | ◎ | 20-1 |
| | #undef | #define で定義されている記号定数またはマクロの定義を無効にします。 | ◎ | ◎ | ◎ | 20-2 |
| | #error | #error コマンドを実行すると、強制的にコンパイルエラーとします。 | ◎ | ◎ | ◎ | 20-3 |
| ファイル取り込み | #include | プリプロセッサプログラムを取り込みます。 | ◎ | ◎ | ◎ | 20-4 |
| 最適化 | #pragma optimize | プログラムごとに行う最適化を指定します。 | ◎ | ◎ | ◎ | 20-6 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

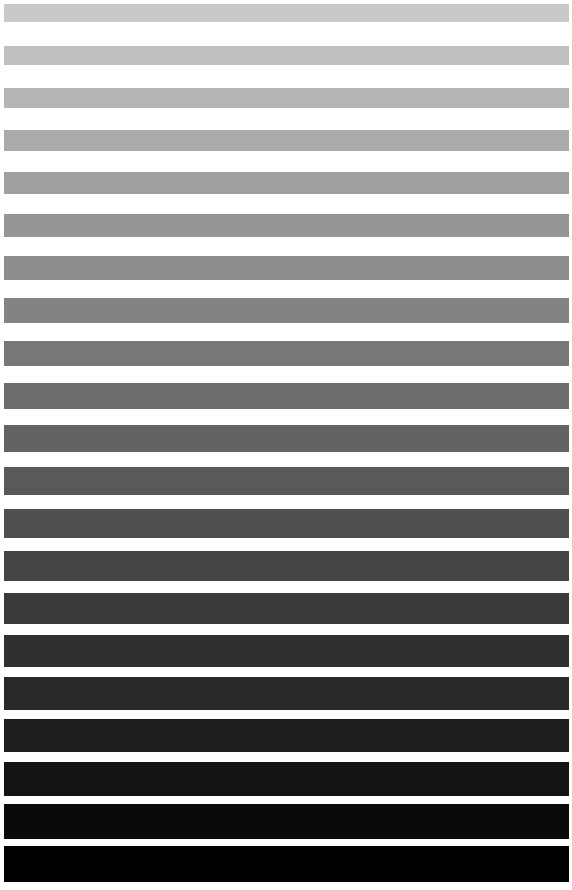
| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| **視覚制御 ( ロボットコントローラ : オプション )** | | | | | | |
| 画像入出力 | CAMIN | カメラからの映像を画像メモリ（処理画面）に格納します。 | ◎ | ◎ | ◎ | 21-3 |
| | CAMMODE | カメラ映像を格納する際の機能を設定します。 | ◎ | ◎ | ◎ | 21-4 |
| | CAMLEVEL | カメラ映像の入力レベルを設定します。 | ◎ | ◎ | ◎ | 21-6 |
| | VISCAMOUT | カメラからの映像をモニタに表示します。 | ◎ | ◎ | ◎ | 21-7 |
| | VISPLNOUT | 格納メモリの画像をモニタに表示します。 | ◎ | ◎ | ◎ | 21-8 |
| | VISOVERLAY | 描画画面の情報をモニタに表示します。 | ◎ | ◎ | ◎ | 21-9 |
| | VISDEFTABLE | カメラ映像の取り込み、画像出力の際のルックアップテーブルデータを設定します。 | ◎ | ◎ | ◎ | 21-10 |
| | VISREFTABLE | ルックアップテーブルのデータを参照します。 | ◎ | ◎ | ◎ | 21-12 |
| ウィンドウ設定 | WINDMAKE | 画像処理する範囲を指定します。 | ◎ | ◎ | ◎ | 21-13 |
| | WINDCLR | 設定したウィンドウ情報を消去します。 | ◎ | ◎ | ◎ | 21-18 |
| | WINDCOPY | ウィンドウのデータをコピーします。 | ◎ | ◎ | ◎ | 21-19 |
| | WINDREF | ウィンドウの情報を得ます。 | ◎ | ◎ | ◎ | 21-21 |
| | WINDDISP | 設定したウィンドウを描画します。 | ◎ | ◎ | ◎ | 21-22 |
| 描画 | VISSCREEN | 描画する画面を指定します。 | ◎ | ◎ | ◎ | 21-23 |
| | VISBRIGHT | 描画する際の輝度値を指定します。 | ◎ | ◎ | ◎ | 21-25 |
| | VISCLS | モードで指定した画面を指定輝度で塗りつぶし（クリア）します。 | ◎ | ◎ | ◎ | 21-26 |
| | VISPUTP | 画面に点を描画します。 | ◎ | ◎ | ◎ | 21-28 |
| | VISLINE | 画面に直線を描画します。 | ◎ | ◎ | ◎ | 21-29 |
| | VISPTP | 画面に 2 点間を結ぶ直線を描画します。 | ◎ | ◎ | ◎ | 21-30 |
| | VISRECT | 画面に矩形を描画します。 | ◎ | ◎ | ◎ | 21-31 |
| | VISCIRCLE | 画面に円を描画します。 | ◎ | ◎ | ◎ | 21-33 |
| | VISELLIPSE | 画面に楕円を描画します。 | ◎ | ◎ | ◎ | 21-34 |
| | VISSECT | 画面に扇を描画します。 | ◎ | ◎ | ◎ | 21-36 |
| | VISCROSS | 画面に十字マークを描画します。 | ◎ | ◎ | ◎ | 21-37 |
| | VISLOC | 文字の表示位置を指定します。 | ◎ | ◎ | ◎ | 21-38 |
| | VISDEFCHAR | 文字のサイズと表示方法を指定します。 | ◎ | ◎ | ◎ | 21-40 |
| | VISPRINT | 画面に文字・数字を表示します。 | ◎ | ◎ | ◎ | 21-42 |
| 画像処理 | VISWORKPLN | 処理対象の格納メモリ（処理画面）を指定します。 | ◎ | ◎ | ◎ | 21-43 |
| | VISGETP | 格納メモリ（処理画面）から指定座標の輝度を取得します。 | ◎ | ◎ | ◎ | 21-44 |
| | VISHIST | 画面のヒストグラム（輝度分布）を得ます。 | ◎ | ◎ | ◎ | 21-45 |
| | VISREFHIST | ヒストグラム結果を読み出します。 | ◎ | ◎ | ◎ | 21-46 |

| 4 軸 | 6 軸 | 視覚装置 | |
|---|---|---|---|
| ◎ | ◎ | ◎ | 全ロボットおよび視覚装置で使用可能 |
| ○ | ○ | ○ | 全ロボットで使用可能、ただし 4 軸ロボット、6 軸ロボット、または視覚装置で仕様が異なる。 |
| ◎ | V1.2 | | 4 軸ロボットとバージョン 1.2 以降の 6 軸ロボットで使用可能 |

| 機能区分 | コマンド | 機能 | 4 軸 | 6 軸 | 視覚装置 | 説明ページ |
|---|---|---|---|---|---|---|
| | VISLEVEL | ヒストグラム結果に基づき 2 値化レベルを求めます。 | ◎ | ◎ | ◎ | 21-47 |
| | VISBINA | 画面を 2 値化処理します。 | ◎ | ◎ | ◎ | 21-49 |
| | VISBINAR | 画面を 2 値化表示します。 | ◎ | ◎ | ◎ | 21-51 |
| | VISFILTER | 画像にフィルタ処理を行ないます。 | ◎ | ◎ | ◎ | 21-52 |
| | VISMASK | 画像間演算をします。 | ◎ | ◎ | ◎ | 21-54 |
| | VISCOPY | 画面をコピーします。 | ◎ | ◎ | ◎ | 21-56 |
| | VISMEASURE | ウィンドウ内の特徴（面積、重心、主軸角）を計測します。 | ◎ | ◎ | ◎ | 21-57 |
| | VISPROJ | ウィンドウ内の投影データを計測します。 | ◎ | ◎ | ◎ | 21-60 |
| | VISEDGE | ウィンドウ内のエッジを計測します。 | ◎ | ◎ | ◎ | 21-62 |
| コード認識 | VISREADQR | QR コードを読み取ります。 | ◎ | ◎ | ◎ | 21-67 |
| ラベリング | BLOB | ラベリングを実行します。 | ◎ | ◎ | ◎ | 21-70 |
| | BLOBMEASURE | 対象ラベル番号の特徴計測を行ないます。 | ◎ | ◎ | ◎ | 21-73 |
| | BLOBLABEL | 指定座標のラベル番号を取得します。 | ◎ | ◎ | ◎ | 21-75 |
| | BLOBCOPY | 対象ラベル番号をコピーします。 | ◎ | ◎ | ◎ | 21-77 |
| サーチ機能 | SHDEFMODEL | サーチモデルの登録をします。 | ◎ | ◎ | ◎ | 21-79 |
| | SHREFMODEL | 登録モデルデータを参照します。 | ◎ | ◎ | ◎ | 21-82 |
| | SHCOPYMODEL | 登録モデルをコピーします。 | ◎ | ◎ | ◎ | 21-83 |
| | SHCLRMODEL | 登録モデルを消去します。 | ◎ | ◎ | ◎ | 21-84 |
| | SHDISPMODEL | 登録モデルを画面に表示します。 | ◎ | ◎ | ◎ | 21-85 |
| | SHMODEL | モデルをサーチします。 | ◎ | ◎ | ◎ | 21-86 |
| | SHDEFCORNER | コーナーサーチの条件を設定します。 | ◎ | ◎ | ◎ | 21-90 |
| | SHCORNER | コーナーをサーチします。 | ◎ | ◎ | ◎ | 21-91 |
| | SHDEFCIRCLE | 円サーチの条件を設定します。 | ◎ | ◎ | ◎ | 21-94 |
| | SHCIRCLE | 円をサーチします。 | ◎ | ◎ | ◎ | 21-95 |
| 結果取得 | VISGETNUM | 画像処理結果を格納メモリから取得します。 | ◎ | ◎ | ◎ | 21-97 |
| | VISGETSTR | コード認識結果を取得します。 | ◎ | ◎ | ◎ | 21-98 |
| | VISPOSX | 画像処理結果（X 座標）を格納メモリから取得します。 | ◎ | ◎ | ◎ | 21-99 |
| | VISPOSY | 画像処理結果（Y 座標）を格納メモリから取得します。 | ◎ | ◎ | ◎ | 21-100 |
| | VISSTATUS | 各命令の処理結果をモニタします。 | ◎ | ◎ | ◎ | 21-101 |
| | VISREFCAL | CAL（視覚－ロボット座標変換）データを取得します。 | ◎ | ◎ | ◎ | 21-102 |

# 第１部
# プログラムの基礎知識

# 第 1章
# プログラム例



この章では、簡単なアプリケーションの例を用いて、各コマンドの使用例を示します。

# 第１章　プログラム例

## 1.1　モデルケースアプリケーション

最初にモデルケースとして図1-1に示すようなアプリケーションを取り上げ、プログラム例を示します。このプログラムを読みながら、プログラムに必要な知識を身につけてください。

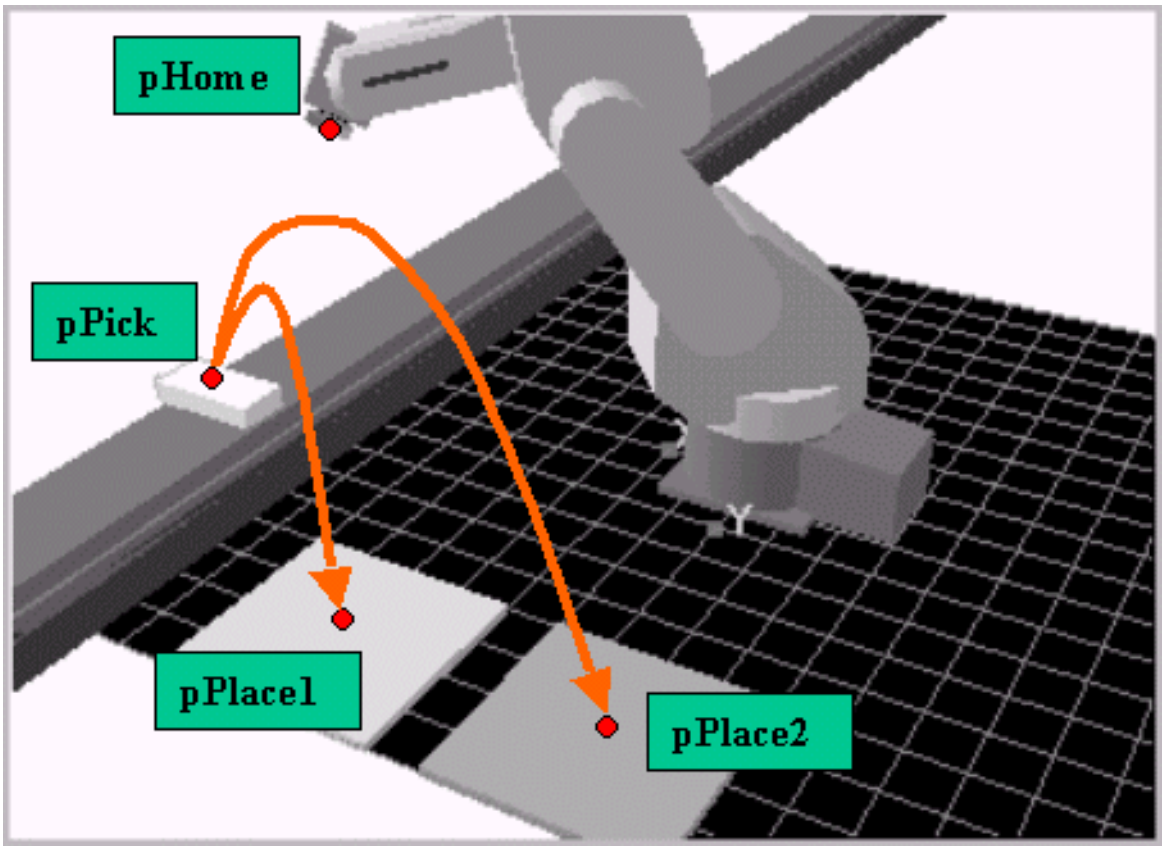



図1-1　モデルケースアプリケーション

このモデルケースでは、pPickに置かれた対象物のQRコードを読み取り、QRコードによってpPlace1、またはpPlace2のどちらに運ぶかを判断して、掴んで、それぞれの場所に運んで、置きます。pHomeは、ホームポジション、すなわち基点となる位置です。

注意：プログラム作成のための操作方法については、立上げハンドブック　第11章　プログラムの作成を参照してください。同じモデルケースアプリケーションを用いて、プログラム作成の操作手順について説明しています。

# 1.2　プログラムフロー

モデルケースのためのプログラムフローを、図1-2に示します。

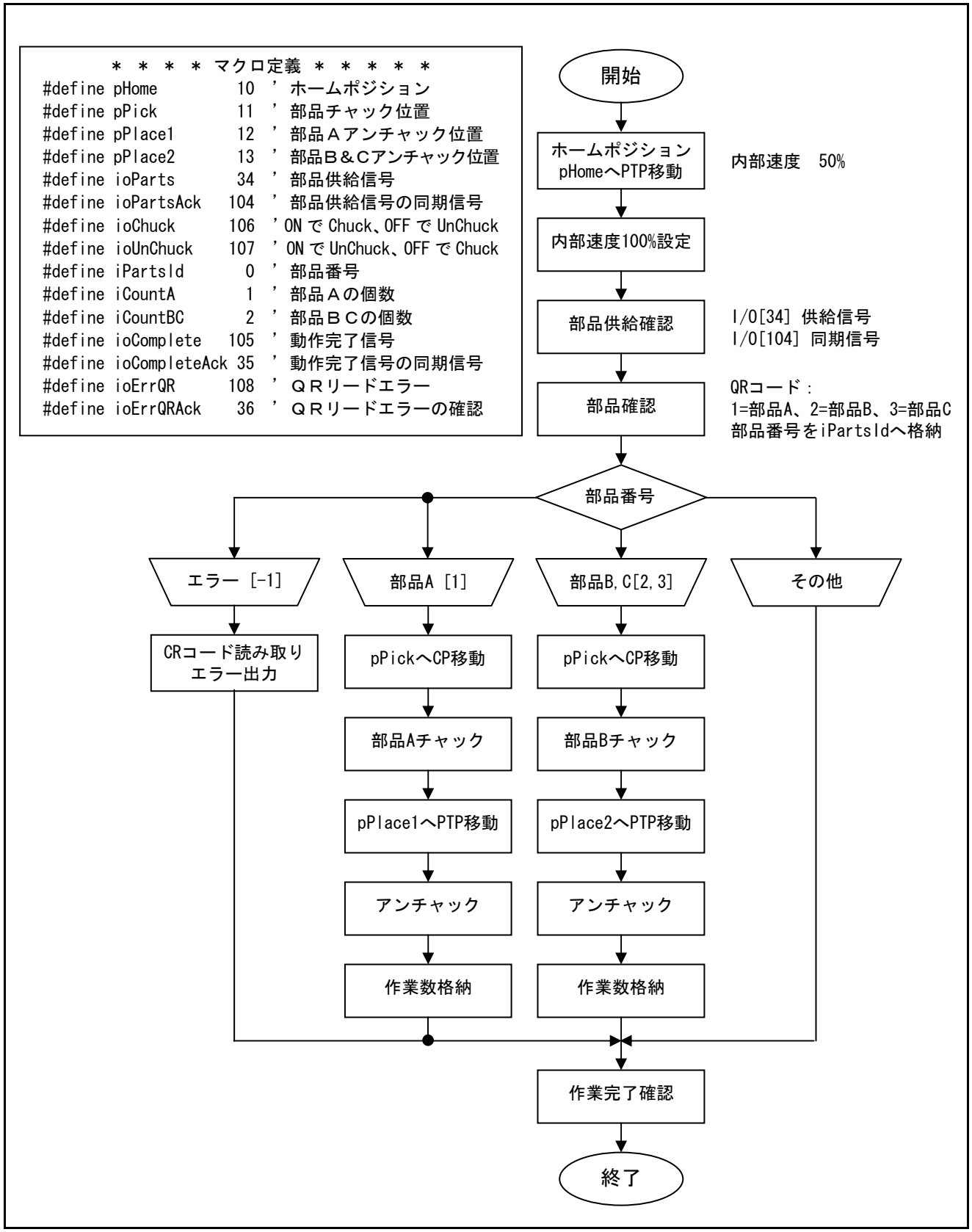



図1-2　モデルケースのプログラムフロー

# 1.3　プログラムリスト

モデルケースのためのプログラムのリストを掲げます。
プログラムは、「PRO1」、「PRO2」、「dioSetAndWait」、「dioWaitAndSet」の4本です。
「PRO1」は、メインプログラムです。
「PRO2」は、QRコードの読み取りに関わるサブプログラムです。
「dioSetAndWait」と「dioWaitAndSet」は、部品供給を確認するためのI/Oを操作するサブプログラムです。この2つは、WINCAPSⅢのプログラムバンクに収録されています。

プログラムリスト「PRO1」

```
'!TITLE "ピック＆プレース"
#INCLUDE  "dio_tab.h"                          'DIOマクロ定義ファイルの読み込み
#INCLUDE  "var_tab.h"                          '変数マクロ定義ファイルの読み込み
#DEFINE   appLen 100                           'アプローチ長の定義
            マクロ定義appLenと100は同義となります。

PROGRAM pro1  プログラム宣言(プログラム名を宣言します。詳しくはp.9-1を参照)
   TAKEARM  TAKEARM 文(ロボットを動作させるのに必要
            な文です。詳しくは p.14-19 を参照)    'アームセマフォ取得
   MOVE P, P[pHome], S=50                      'ホームポジションへ内部速度を50%でPTP移動
              マクロ P[10]あるいは P10 と同じ意味です。マ
              クロについては p.20-1 を参照
   SPEED 100                                   '内部速度を100%に
   CALL dioWaitAndSet(ioParts, ioPartsAck)     '部品供給確認
              プログラム呼び出し(p.2-1 と p.2-3 を参照)

   CALL pro2                                   'QRコードの読み取り
   SELECT CASE I[iPartsId]  SELECT CASE 文(I[〇〇]を条
                マクロ      件にした分岐命令です。
                            p.11-19 を参照
     CASE -1  I[〇〇]が-1 のとき、以下が実行されます。  'QRコードの読み取り失敗時処理
       CALL dioSetAndWait(ioErrQR, ioErrQRAck)  'エラー出力
     CASE 1
       GOSUB *PlacePartsA                      '部品Aの処理
          サブルーチン呼び出し(p.2-2を参照)
     CASE 2, 3
       GOSUB *PlacePartsBC                     '部品B、Cの処理
   END SELECT
   CALL dioSetAndWait(ioComplete,ioCompleteAck) ' 動作完了信号出力
   GIVEARM  TAKEARM で宣言してロボットアームの占有  ' アームセマフォ解放
            使用権を開放します。
END
```

次ページへ続く

プログラムリスト「PRO1」（続き）

```
' ===== 部品チャック =====
*ChuckItem: *○○:でラベルを指定します。後にCALL *ChuckItemで呼び出されます。
   SET IO[ioChuck] IO[106]と同意
                マクロ
   SET 命令(IO を ON します。p.13-5 参照)

   RESET IO[ioUnChuck] IO[106]と同意
                マクロ
   RESET 命令(IO を OFF します。p.13-7 参照)

   RETURN RETURN文(*ChuckItemとの間でサブルーチンを構成します。)


' ===== 部品アンチャック =====
*UnchuckItem: ラベル
   RESET IO[ioChuck]
   SET IO[ioUnChuck]
   RETURN リターン文


' ===== 部品Aの処理 =====
*PlacePartsA: ラベル名(サブルーチン名の宣言)
   APPROACH P, P[pPick], appLen                    'チャック点から100mm離れた位置へCP移動
   APPROACH 命令      マクロ P[11]    4 行目で定義されたマクロ。
   (P[pPick1]の       と同意          100 と定義。このように定義
   直上へロボットを                   することで 1 ヶ所でアプロー
   移動させます。)                    チ長を変更できます。

   MOVE L, P[pPick], S=80                          'チャック点へ内部速度を80%でCP移動
     MOVE命令(p.12-14参照)
   GOSUB *ChuckItem                                '部品チャック
   DEPART L, appLen                                '現在位置から100mm離れた位置へ移動
     DEPART 命令      4 行目で定義されたマクロ。100 と定義。
     (p.12-4 参照)    このように定義することで 1 ヶ所でア
                      プローチ長を変更できます。

   APPROACH P, P[pPlace1], appLen                  'アンチャック点から100mm離れた位置へCP移動
     APPROACH 命令          4 行目で定義されたマクロ。
      (P[pPick1]の          100 と定義。このように定義
     直上へロボットを       することで 1 ヶ所でアプロー
     移動させます。         チ長を変更できます。

   MOVE P, P[pPlace1], S=50                        'アンチャック点へ内部速度を50%でCP移動
     MOVE命令(p.12-14参照)
   GOSUB *UnchuckItem                              '部品アンチャック
   DEPART L, appLen                                '現在位置から100mm離れた位置へCP移動
     DEPART 命令      4 行目で定義されたマクロ。100 と定義。こ
     (p.12-4 参照)    のように定義することで 1 ヶ所でアプロー
                      チ長を変更できます。
   I[iCountA] = I[iCountA] + 1                     '部品Aの個数をカウント
                  I[iCountA]を1増加させます。
   RETURN
```

次ページへ続く

プログラムリスト「PRO1」(続き)

```
' ===== 部品BCの処理 =====
*PlacePartsBC: ラベル名 (サブルーチン名の宣言)
   APPROACH P, P[pPick], appLen                    'チャック点から100mm離れた位置へCP移動
     APPROACH 命令 (P[pPick1]の直上へロボットを移動させます。
                                  4行目で定義されたマクロ。100と定義。このように定義することで1ヶ所でアプローチ長を変更できます。
   MOVE L, P[pPick], S=90                          'チャック点へ内部速度を90%でCP移動
   GOSUB *ChuckItem                                '部品チャック
   DEPART L, appLen                                '現在位置から100mm離れた位置へ移動
     DEPART 命令 (p.12-4参照)  4行目で定義されたマクロ。100と定義。このように定義することで1ヶ所でアプローチ長を変更できます。
   APPROACH P, P[pPlace2], appLen                  'アンチャック点から100mm離れた位置へCP移動
     APPROACH 命令 (P[pPick1]の直上へロボットを移動させます。(p.12-1参照)
                                  4行目で定義されたマクロ。100と定義。このように定義することで1ヶ所でアプローチ長を変更できます。
   MOVE L, P[pPlace2], S=80                        'アンチャック点へ内部速度を80%でCP移動
     MOVE命令(p.12-14参照)
   GOSUB *UnchuckItem                              '部品アンチャック
   DEPART L, appLen                                '現在位置から100mm離れた位置へCP移動
     DEPART 命令 (p.12-4参照)  4行目で定義されたマクロ。100と定義。このように定義することで1ヶ所でアプローチ長を変更できます。
   I[iCountBC] = I[iCountBC] + 1                   '部品B、Cの個数をカウント
RETURN
```

次ページへ続く

プログラムリスト「PRO2」

```
'!TITLE "QRコード読み取り"
#INCLUDE  "var_tab.h"                          '変数マクロ定義ファイルの読み込み

' 品番をI[iPartsId]に格納する
PROGRAM pro2
  DIM len1 AS INTEGER ローカル変数宣言'len'を整数型ローカル変数として使用可能とします。
  TAKEVIS 視覚装置の占有を宣言する。視覚装置の使用に必要 '視覚セマフォ取得
  VISSCREEN 1,0,1                              '描画画面0を描画先に指定
    p.21-24参照
  VISCLS 0                                     '描画画面0をクリア
    p.21-27参照
  VISOVERLAY 1                                 '描画画面0を表示
    p.21-9参照
  CAMIN  1,0,0                                 'カメラ映像を処理画面0に取り込み
    p.21-3参照
  VISPLNOUT  0,1                               '処理画面0を表示
    p.21-8参照
  VISREADQR  1,WINDREF(1, 2),WINDREF(1, 3),0,0,1'QRコードの読み取り
    p.21-64参照
  WINDDISP  1                                  'ウィンドウを描画
    p.21-23参照
  VISLOC  10  , 10                             '表示位置を指定
    p.21-37参照
  IF VISSTATUS(0) = 0 THEN                     '計測結果のOKの場合
    len1 = VISGETNUM(0, 0)                     '計測文字数の取得
    VISPRINT  VISGETSTR(1, len1)               '計測結果の表示
    I[iPartsId] = VAL(VISGETSTR(1, len1))      '品番を格納
  ELSE
    VISPRINT  "コード認識できません"           'エラーの表示
    I[iPartsId] = -1                           'エラーのときは-1を格納
  ENDIF
  IF…ELSE… 条件分岐を行います。p.11-17を参照
  GIVEVIS 視覚装置の占有を解除します。         '視覚セマフォ解放
END
```

注意：QRコード読み取りの処理ウィンドウ1はすでに定義されているものとします。

プログラムリスト「WAIT → SET」（ライブラリ）

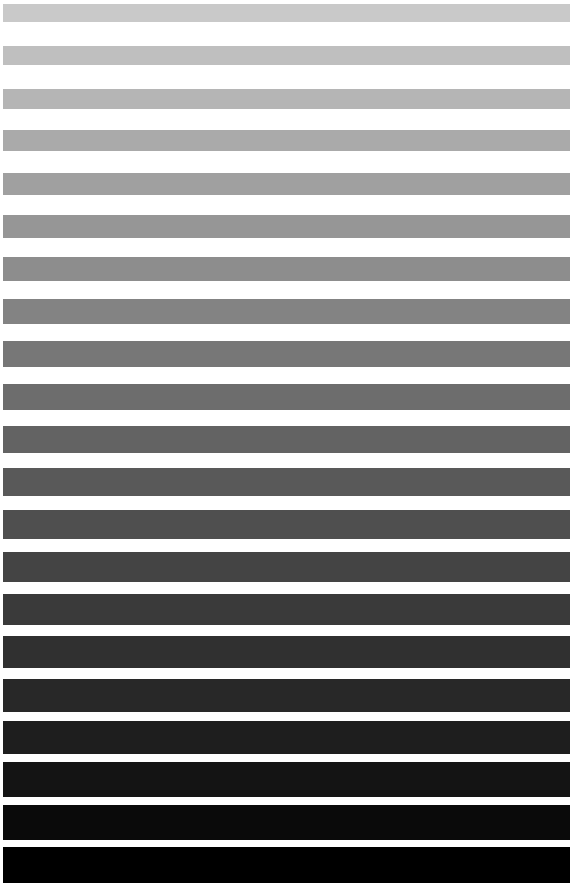
```
'!TITLE "WAIT → SET"
PROGRAM dioWaitAndSet(waitIndex%, ackIndex%)
                      引数宣言    引数宣言p.9-1参照
  RESET IO[ackIndex] RESET文p.13-7参照          '同期信号OFF
  WAIT IO[waitIndex] = ON WAIT文 IO[]がONになるまで待ちます。p.12-64参照  '同期信号ON待ち
  SET IO[ackIndex] SET文p.13-5参照              '同期信号ON

END
```

プログラムリスト「SET → WAIT」（ライブラリ）

```
'!TITLE "SET → WAIT"
PROGRAM dioSetAndWait(ackIndex%, waitIndex%)
                      引数宣言    引数宣言p.9-1参照
  SET IO[ackIndex] SET文p.13-5参照              '同期信号ON
  WAIT IO[waitIndex] = ON WAIT文 IO[]がONになるまで待ちます。p.12-64参照  '同期信号ON待ち
  RESET IO[ackIndex] RESET文p.13-7参照          '同期信号OFF
END
```

# 第 2 章
# プログラムの動作機構



プログラムを作成する際に必要となる、PAC 言語によるプログラムの、動作上のきまりについて説明します。

# 第２章　プログラムの動作機構

## 2.1　プログラムの呼び出しとサブルーチン

プログラムの中で、特定の動作を繰り返す部分を抜き出しておいて、必要に応じて呼び出すことができます。
同じプログラムファイルの中に、この部分を置く方法を、サブルーチンと呼びます。別のファイルに独立させて、別のプログラムとしておいて、呼び出すことをプログラムの呼び出しといいます。
サブルーチンは、呼び出す側のプログラムと同じファイルの中になくてはなりません。
独立した別のファイルになっているプログラムを呼び出す場合は、一つのプログラムを、いろいろなプログラムから呼び出して、共通に利用することができます。
一連の作業を、サブルーチンや別のプログラムにして、一つにまとめることにより、何度も同じ内容のプログラムを記述しなくて済みます。記述間違いや、プログラム作成時間の削減、またプログラムの読みやすさの向上などにも効果的です。

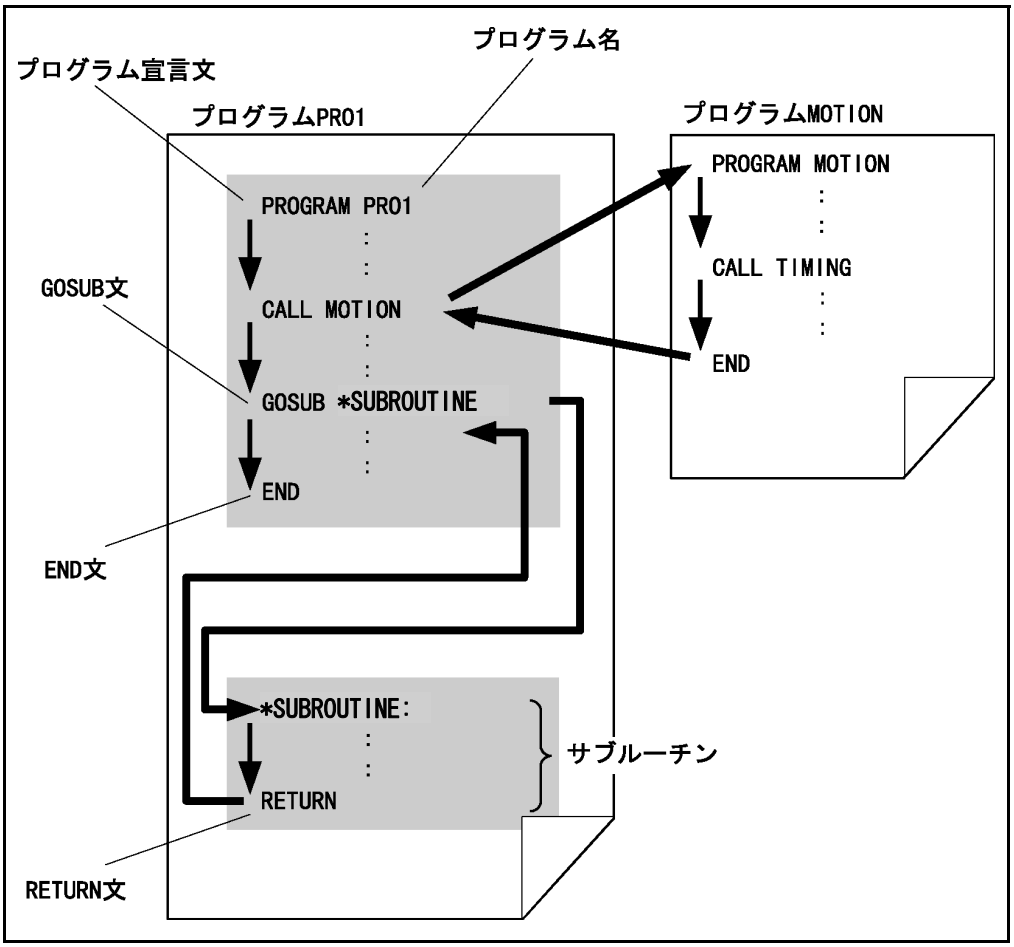



図2-1　プログラムとサブルーチンの呼び出し方の違い

## 2.1.1 サブルーチンの呼び出し

一つのプログラムの中で、同じ処理を、異なった複数の場所で使いたい場合に、その処理をサブルーチンとして記述し、これを複数の場所から呼び出して使用します。

サブルーチンは、呼び出すプログラムと同じファイルの中に書かれている必要があります。

GOSUB文でサブルーチンを呼び出すと、制御はサブルーチンに移ります。サブルーチンの最後の行にあるRETURN文を実行すると、サブルーチンを呼び出したプログラムの元の行の次へ制御が戻ります。

サブルーチンから、別のサブルーチンを呼び出すこともできます。

サブルーチンは、呼び出し元と同じファイルの中にあるので、ローカル変数も共用していますから、呼び出す際に変数の受け渡しを考えなくても良いのが特徴です。ただし、他のファイルから呼び出して使うことはできませんし、直接ティーチングペンダントや外部装置から起動することもできません。

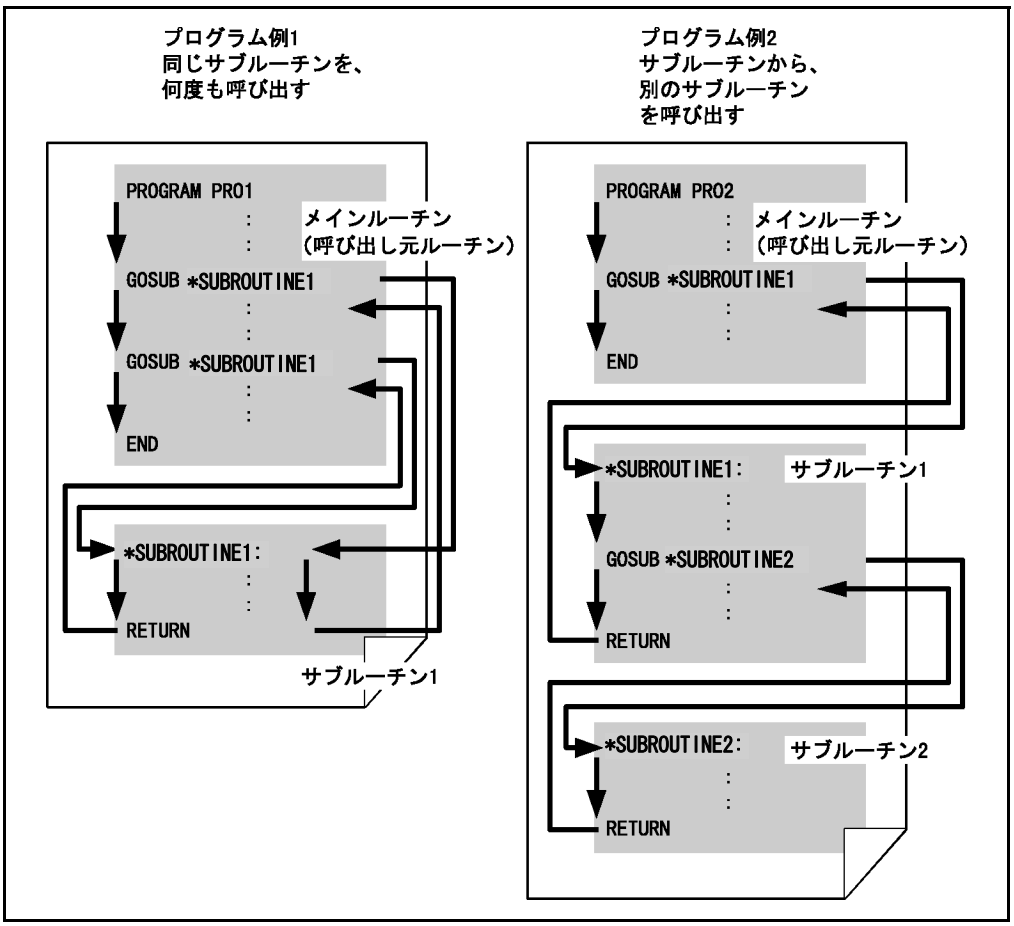



図2-2 サブルーチンの呼び出し

## 2.1.2　プログラムの呼び出し

主に実行するプログラムとは別に、プログラムを作成し、これをサブルーチンのように呼び出して使用することができます。
プログラムを呼び出すには、CALL文でプログラム名を指定して行ないます。
CALL文でプログラムを呼び出すと、制御は呼び出したプログラムに移ります。
呼び出したプログラムの最後の行にあるEND文を実行すると、呼び出し元のプログラムの次の行へ制御が戻ります。
呼び出されたプログラムもまた、別のプログラムを呼び出すことができます。ただし、呼び出されたプログラムが、呼び出し元のプログラムを呼び出すことはできません。
一つのプログラムを、複数のプログラムが呼び出せるので、汎用性のあるプログラムは、開発効率を向上させます。
プログラムの呼び出しでは、グローバル変数のみ共通で使用できます。ローカル変数は共有していないので、必要に応じて引数として渡します。「8.16　値による呼び出しと参照による呼び出し」を参照してください。

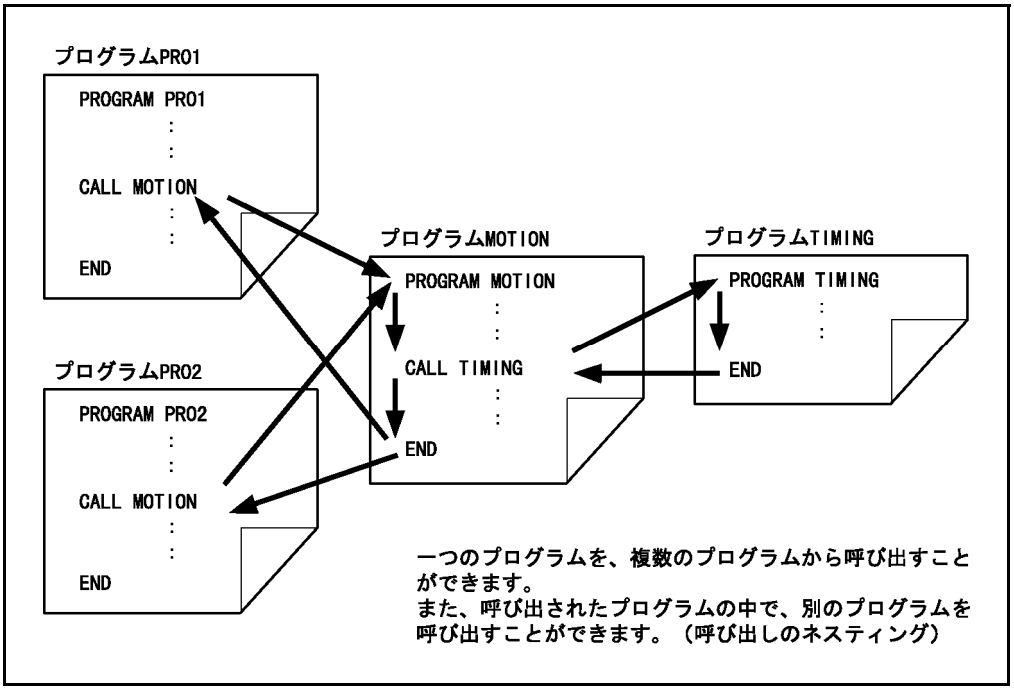



図2-3　プログラムの呼び出し構造

## 2.1.3 プログラムの再帰呼び出し

プログラムを呼び出す際に、CALL文で指定するプログラム名として、呼び出すプログラム自身を指定することができます。これを再帰呼び出しといいます。

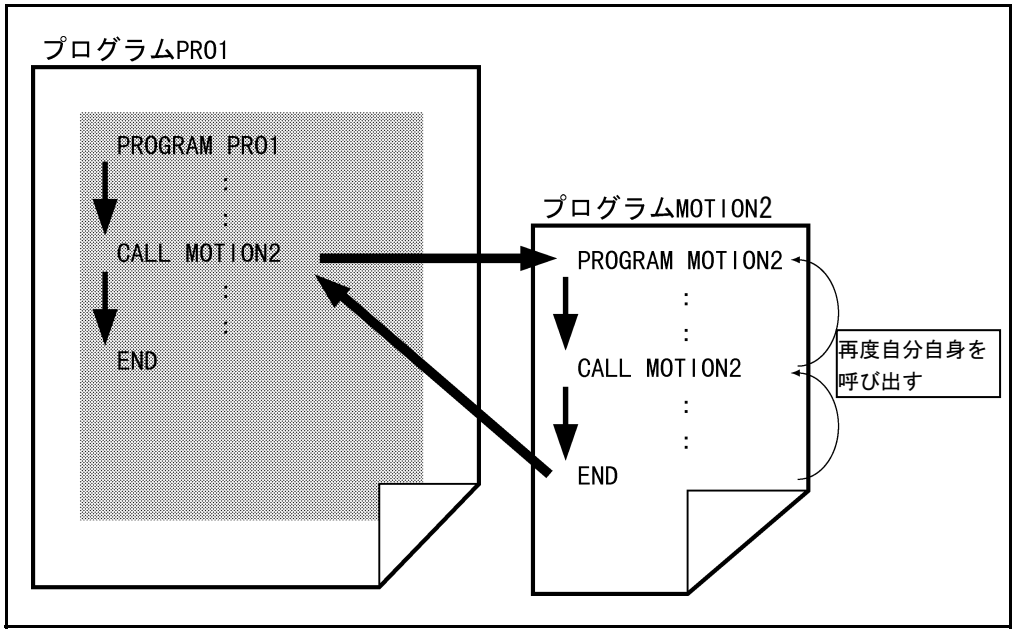



図2-4　プログラムの再帰呼び出し構造

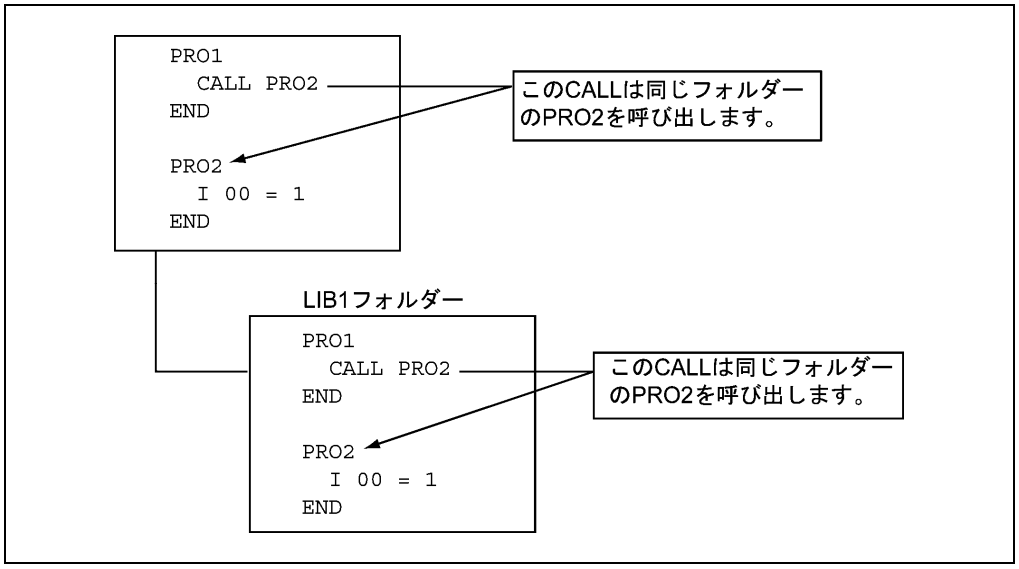
注意：PACプログラムの変数は、プログラムごとに静的な場所に確保されるため、再帰呼び出しを使用する場合には、変数の使用方法に十分注意してください。ローカル変数により初期値が設定されている場合には、呼び出されるごとに値が初期値に戻ります。

## 2.1.4　フォルダー構造のプログラムの呼び出し方　[RC7_Ver.2.2 以降]

### 2.1.4.1　CALL、RUN ステートメント

#### [ 1 ]プログラム名のみのとき

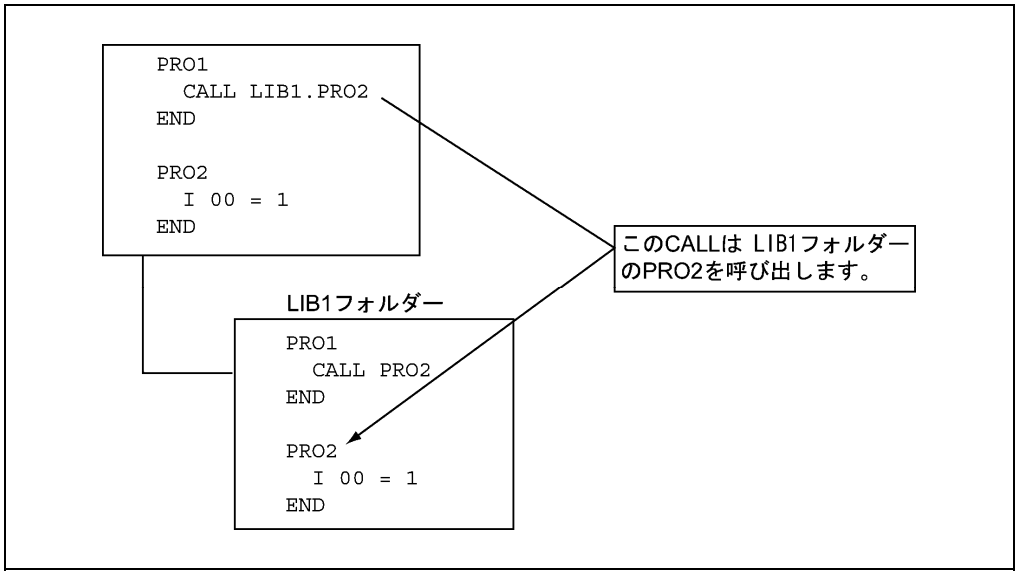
呼び出すPACがあるフォルダー内のプログラムを呼び出します。




#### [ 2 ]フォルダー名を含めたプログラム名のとき

・プログラムは下記のように指定します。
　　CALL(フォルダー名).(プログラム名)

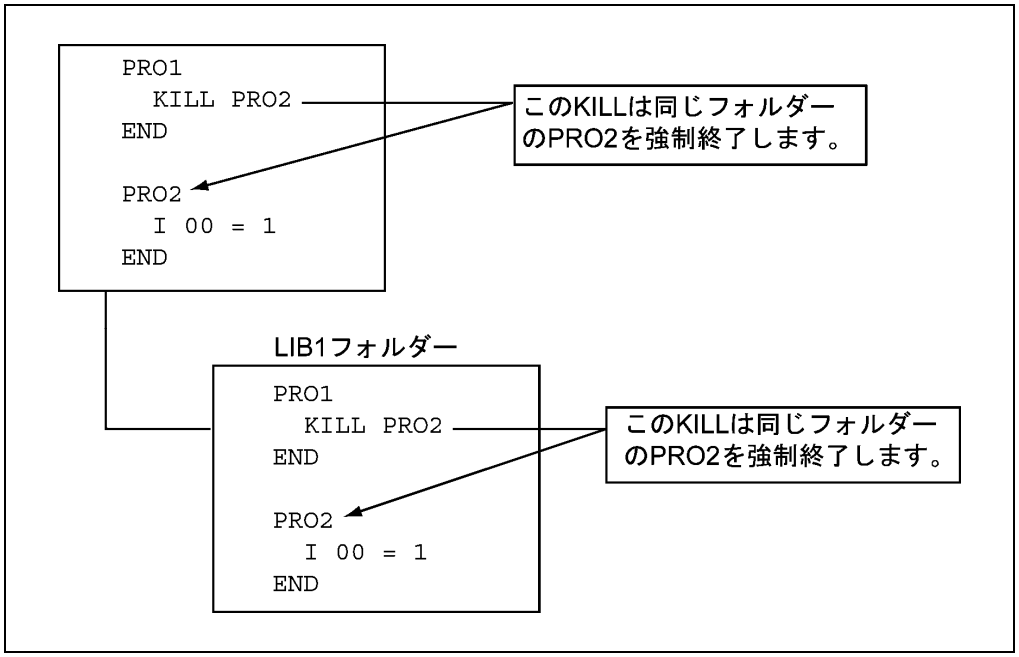
・呼び出せるフォルダーの階層は１つ下のフォルダーまでです。
・CALL(フォルダー名).(フォルダー名).(プログラム名)はコンパイルエラーになります。



## 2.1.4.2 KILL、SUSPEND、STATUS ステートメント

### [ 1 ]プログラム名のみのとき

呼び出すPACがあるフォルダー内のプログラムを**処理**します。



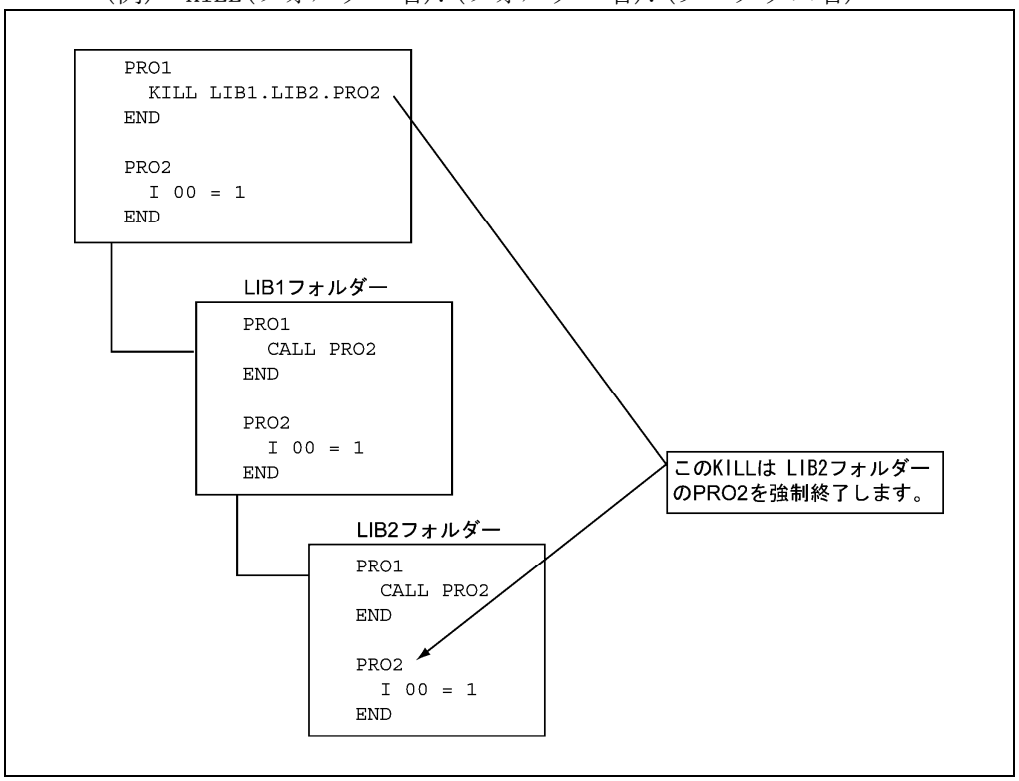

### [ 2 ]フォルダー名を含めたプログラム名のとき

・プログラムは下記のように指定します。

KILL(フォルダー名).(プログラム名)

・複数のフォルダーの指定が可能です。

(例) KILL(フォルダー名).(フォルダー名).(プログラム名)



## 2.2　プログラムの起動

プログラムを起動するのには、

●ティーチングペンダントの操作により起動する
●ミニペンダントの操作により起動する
●外部機器からI/Oを通じて起動する
という３つの方法があります。
起動する方法により、起動できるプログラムに、次に説明するような制限があります。
起動されたプログラムが、別のプログラムを呼び出す方法については、「2.1 プログラムの呼び出しとサブルーチン」に説明されています。

### 2.2.1　ティーチングペンダント・ミニペンダントからの起動

ティーチングペンダント・ミニペンダントからプログラムを起動するには、ティーチチェックモードまたは、内部自動モードで行ないます。
起動できるプログラムは、引数を持たないプログラムに限られます。プログラム名はどんな名前でも起動できます。

### 2.2.2　外部機器からの起動

外部自動モードの場合には、外部I/Oからの入力により、プログラムを起動することができます。
起動できるプログラムは、「PRO<番号>」の形式のプログラム名のプログラムに限られます。
外部から起動できるプログラムは、標準モードの場合PR00～PR032767、互換モードの場合PR00～PR0127に限られます。

**注意：「PRO<番号>」の形式のプログラム名を持つプログラムは、引数を持つことができません。**

# 2.3 マルチタスク

PAC言語のプログラムは、同時に複数のプログラムの進行管理を行なうことができます。それぞれのプログラムは、独自の動作プロセスを形成し、これをタスクと呼びます。
複数のプログラムを同時に進行管理することにより、複数のタスクが存在することとなります。これを、マルチタスクと呼びます。

## 2.3.1 優先順位

RUNコマンドによってタスクを起動する際には、優先順位を指定します。これにより、すべてのタスクは、それぞれ決まった優先順位を持っています。
デンソーロボットは、プログラムを実行している間、一定時間ごとにタスクの優先順位を監視し、待ち状態のタスクの中から、優先順位の高いものを実行するように、タスクを切り替えます。
これにより、優先順位の高い順にプログラムは実行され、優先順位が同じであれば、一定時間ごとに実行を交代していくように管理されます。タスクの交代は、短い時間間隔で監視されるので、見かけ上は同時に複数のプログラムが実行されているように見えます。

## 2.3.2 タスク間通信

マルチタスクでロボットを制御する場合、タスクどうしを同期したり、あるいは同時には動かないように排他制御を行なう必要が出ることがあります。いずれの場合も、タスクどうしが合図を交換するために、タスク間で通信する必要が出てきます。
タスクどうしが通信をするには、セマフォを使う方法が一般的ですが、デンソーロボットではI/Oを使う方法もあります。

### 2.3.2.1 セマフォ

セマフォは、必要に応じて32個まで作ることができます。セマフォの使用に先立ち、CREATESEMコマンドでセマフォを生成すると、セマフォIDを取得できます。このセマフォIDによって、複数のセマフォの中から特定のセマフォを指定できるようになります。
同期制御または排他制御を行なう場合、他のタスクの指令を待つ側のタスクはTAKESEMコマンドを実行して、指令を出す側のタスクがGIVESEMコマンドを実行するのを待ちます。指令を出す側のタスクは、準備ができたらGIVESEMコマンドを実行して、セマフォを待っているタスクに対して処理の実行を許可します。
一つのGIVESEMコマンドは、一つのセマフォ待ちタスクに対してのみ有効です。
複数のタスクが、同じセマフォIDのセマフォを待っている場合に、どのタスクから実行するかは、先着順または優先順位順の2つのキューイング（実行待ち）方式から選択できます。CREATESEMコマンドでキューイング方式を指定します。

次の2つのプログラム、「MOTION1」と「TIMING1」は、セマフォを使った同期制御の例です。
プログラム「TIMING1」を起動すると、プログラム「MOTION1」を並行して起動します。「MOTION1」は、「TIMING1」が生成するタイミングをセマフォで受け取り、MOVE コマンドを実行します。

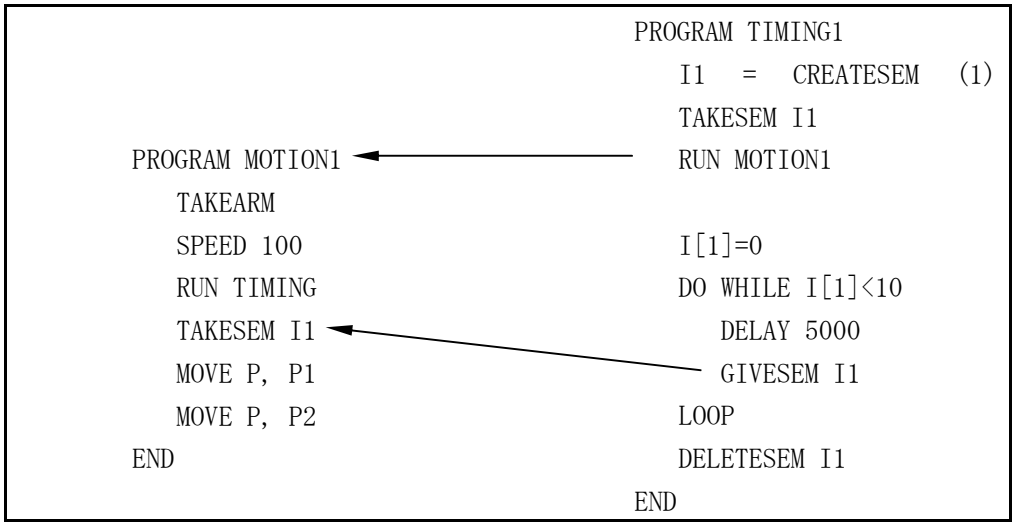



**図2-5　セマフォによる同期制御の例**

## 2.3.2.2　I/O

I/Oを使って、タスク間通信をすることができます。
次の2つのプログラム、「MOTION2」と「TIMING2」は、I/Oを使った同期制御の例です。P2-6「2.3.2.1　セマフォ」で説明している「セマフォによる同期制御の例」と同じ動作をします。
I/Oでタスク間通信を行なう場合には、キューイングは先着順でしか行なえません。
プログラム「TIMING2」を起動すると、「TIMING2」が生成するタイミングを「MOTION2」は IO[118]の状態によって受け取り、MOVE コマンドを実行します。そして、「TIMING2」のループが完了するまで繰り返します。

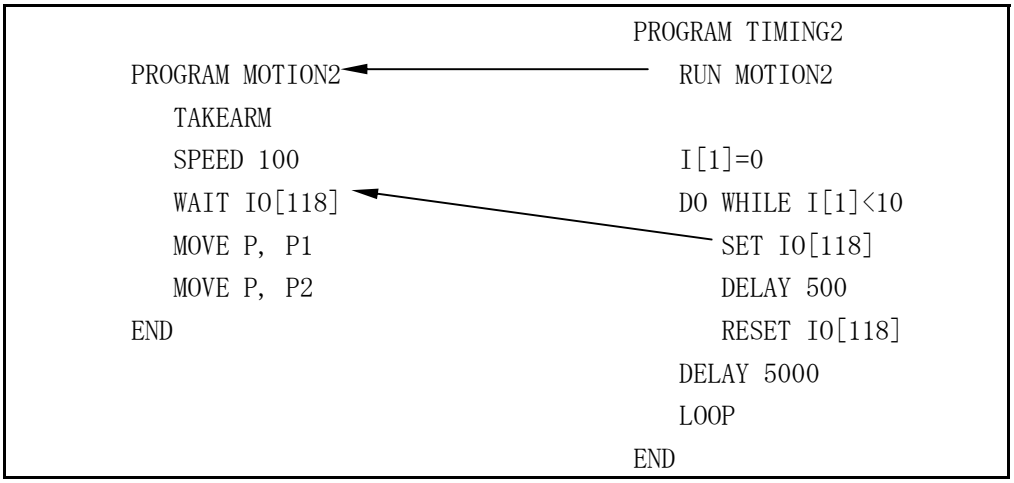



**図2-6　I/Oによる同期制御の例**

# 2.4 シリアル通信

## 2.4.1 回線番号

ロボットコントローラの回線番号とチャンネル番号の対応は以下のとおりです。

ロボットコントローラの回線番号

| 回線番号 | チャンネル番号 | 用途 |
|---|---|---|
| -1 (注1) | - | μVision ボードRS-232C ポート |
| 0 | ch1 | ティーチングペンダント接続ポート |
| 1 | ch2 | 汎用ポート |
| 2 | ch3 | (予約) |
| 3 | ch4 | (予約) |
| 4 ～ 7 | ch5 ～ ch8 | Ethernetサーバポート |
| 8 ～ 15 | ch9 ～ ch16 | Ethernetクライアントポート(注2) |

注1： 次のコマンドにのみ使用可能です。

・FLUSH
・printb
・inputb
・lprintb
・linputb
・com_state

また、μVision ボードRS-232C ポートを使用する場合は、RS-232C の通信設定を次の設定にしてください。

転送速度 ：9600bps
ビット長 ：８
パリティビット：なし
ストップビット：１
フロー制御 ：なし

注2： Ethernetクライアントポートを使用するときは、[7.2 シリアルバイナリ通信]の com_encom文、com_discom文を使用してポートのオープン、クローズを行なう必要があります。

注3： ロボットコントローラのクライアント（Client)、サーバ（Server）の設定は操作ガイド「5.7項」を参照してください。

## 2.4.2 通信コマンド

RS232CおよびEthernetを介して通信するPAC言語のコマンドには、次のものがあります。送受信するデータが2種類あり、種類別にコマンドが用意されています。各コマンドの説明は「13.2項」、「13.3項」を参照してください。

| 送受信データ | コマンド |
|---|---|
| 文字コードデータ（ASCII コードデータ） | PRINT 文(出力) |
| | WRITE 文(出力) |
| | INPUT 文(入力) |
| | LINEINPUT 文(行入力) |
| バイナリデータ | printb(1 バイトデータ出力) |
| | inputb(1 バイトデータ入力) |
| | lprintb (複数バイトデータ出力) |
| | linputb (複数バイトデータ入力) |

PRINT文とWRITE文は出力形式が異なります。WRITE文の形式で出力された文字列は、INPUT文で正しく入力できます。

## 2.4.3 通信バッファのクリア

FLUSH文によって、RS232CおよびEthernetの通信バッファを空にします。
通信を開始する前に必ず実施してください。

・FLUSH :「13.2項」を参照。

## 2.4.4 通信ポートコマンド

RS232CおよびEthernetの各ポートの設定および状態取得のコマンドは次のものがあります。各コマンドはRS232CポートかEthernetポートかによって機能が異なります。
コマンドの説明は「13.3項」を参照してください。

| コマンド | 機能 | |
|---|---|---|
| | RS232C | Ethernet |
| com_encom | バイナリデータ送受信するための設定 | クライアントポートを介してのデータ送受信するための設定 |
| com_discom | | |
| com_state | 各データ送受信時のポートの状態取得 | |

## 2.4.5 サンプル・アプリケーション

この項では、シリアル通信を利用する簡単なアプリケーションを想定して、プログラムの実例を掲げます。実際のアプリケーションへの応用のための参考としてください。

この例のアプリケーションでは、ロボットコントローラは、動作回数をカウントして、一定時間ごとに動作回数のデータを送信します。パソコン側では、ロボットコントローラの出す動作回数のデータを受信します。ロボットコントローラ側のプログラムは、PAC言語で書かれます。パソコン側の例としては、Visual Basic を使用します。

「2.4.4.1 ロボットコントローラのプログラム」および「2.4.4.2 パソコン側のプログラム」に示すプログラムを、ロボットコントローラとパソコンにそれぞれ用意し、走らせることで、ロボットの動作回数がRS232C回線を通じて送信され、パソコンに受信されます。

### 2.4.5.1 ロボットコントローラのプログラム

ロボットコントローラ側のプログラム例を、図2-7と図2-8に示します。

この2つのプログラムは、マルチタスクで動作するようになっています。

プログラム「PRO1」は、カウンタや座標の初期化を行なった後「PRO2」を並行起動します。さらに、「PRO1」はロボットを「P2」と「P1」の間を往復運動させます。その間に「PRO2」は、2秒（2000ミリ秒）ごとに、変数 I[1] に収められたロボットの動作回数をRS232Cポート（ch2）から送信します。

```
'!TITLE “ロボット動作”
PROGRAM PRO1
    I1=0                                ’カウントのクリア
    P1=(370,400,750,-90,90,0,5)         ’座標の設定
    P2=(510,400,750,-90,90,0,5)         ’座標の設定
    RUN PRO2                            ’タスクを平行して起動
    TAKEARM                             ’ロボット制御権の取得
    DO
       MOVE P,P2                        ’P2 点に移動
       DELAY 1000                       ’１秒待ちます。
       MOVE P,P1                        ’P1 点に移動
       I1=I1+1                          ’移動数をカウント
    LOOP                                ’ループします。
END
```

図 2-7 ロボットコントローラのプログラム「PRO1」“ロボット動作”

```
'!TITLE“送信”
PROGRAM PRO2
    DO
       PRINT #2,I1
       DELAY 2000
    LOOP
END
```

図 2-8　ロボットコントローラのプログラム「PRO2」“送信”

### 2.4.5.2　パソコン側のプログラム

パソコン側のプログラムを以下に示します。
プログラムリストの始めの方は、通信を行なうためのプログラムモジュールを定義しています。リストの最後に、使用例として、各モジュールを利用したデータ受信のためのプログラムがあります。

**注意：このプログラムは、文化オリエント株式会社製の通信コントロール（PDQComm）を用いて、マイクロソフト株式会社のVisual Basic によって書かれています。したがって、Visual Basic および PDQComm のインストールしてあるパソコンでご使用ください。**

```
'    モジュール定義文
'
'    通信ポートおよび通信速度の設定
'    PortNo%      :通信ポートの指定     1 or 2
'    ComInitString$:通信速度などの設定     “9600,N,8,1”など
'
Private Sub CommOpen(PortNo%,ComInitString$)
    '----------------------------------------
    ' 通信速度などの設定
    '----------------------------------------
    PDQComm1.CommPort = PortNo%
    PDQComm1.Settings= ComInitString$       '“9600,N,8,1”
    '----------------------------------------
    ' ハンドシェークなし
    ' 入力タイムアウト値設定（ミリ秒単位）
    ' 受信バッファサイズの設定
    ' 送信バッファサイズの設定
    '----------------------------------------
    PDQComm1.Handshaking = 0
    PDQComm1.InTimeout = 50
    PDQComm1.InBufferSize = 2048
    PDQComm1.OutBufferSize = 2048

    '----------------------------------------
    ' ポートをオープン
    '----------------------------------------
```

次ページへ続く

```
    PDQComm1.PortOpen = True
End Sub


'
'   通信ポートを閉じます。
'
'
Private Sub CommClose()
    '----------------------------------------
    ' ポートをクローズする
    '----------------------------------------
    PDQComm1.PortOpen = False
End Sub

'
'   通信ポートに与えられた文字列をダブルクォーテーションで囲み、
'   最後にキャリッジリターンを付加し送信します。
'   SendText$:   送信したい文字列
'
Private Sub CommWrite(SendText$)
    '----------------------------------------
    ' 文字列と制御コード ich(CR+LF)を送信する場合
    '----------------------------------------
    PDQComm1.Output = Chr(&H22) &SendText$ & Chr(&H22) & Chr(13) &
Chr(10)

    '----------------------------------------
    ' 出力バッファが空になるまで待つ
    '----------------------------------------
    Do
        DoEvents
    Loop Until PDQComm1.OutBufferCount = 0
End Sub

'
'   通信ポートから受信された文字列をキャリッジリターンが来るま
'   で、取得し文字列を返します。
'
Private Function CommRead()

    '----------------------------------------
    ' データを読み出す処理
    '----------------------------------------
    InString$ = ""
    Do While 1
        DoEvents
```

次ページへ続く

```
            '----------------------------------------
            '受信データを読み出す
            '----------------------------------------
            If PDQComm1.InBufferCount > 0 Then
                    InString$ = InString$ +PDQComm1.Input
                    If InStr(1, InString$,vbCr, vbBinaryCompare)
<> 0 Then Exit Do
            End If
    Loop
    CommRead = InString$

End Function

使用サンプル例
Private Sub mnuCh1_Click()
    '  ポート１を使用し、ボーレートを 19200 にします。
    CommOpen 1, "19200,N,8,1"
    '  コントローラから移動カウント数を受信します。
    ReadBuf$ = CommRead
    '  使用されたポートを閉じます。
    CommClose
End Sub
```

**図 2-9　パソコン側のプログラム**

# 2.5 ライブラリ

汎用性のあるプログラムを、部品のように集めて、用途に合わせて利用するためのものを、プログラムのライブラリと呼びます。PAC言語では、プログラムの中から別のプログラムを呼び出すことができるので、ライブラリの中にあるプログラムを利用したり、作ったプログラムをライブラリに登録して、より効率的なプログラム開発を行なうことができます。

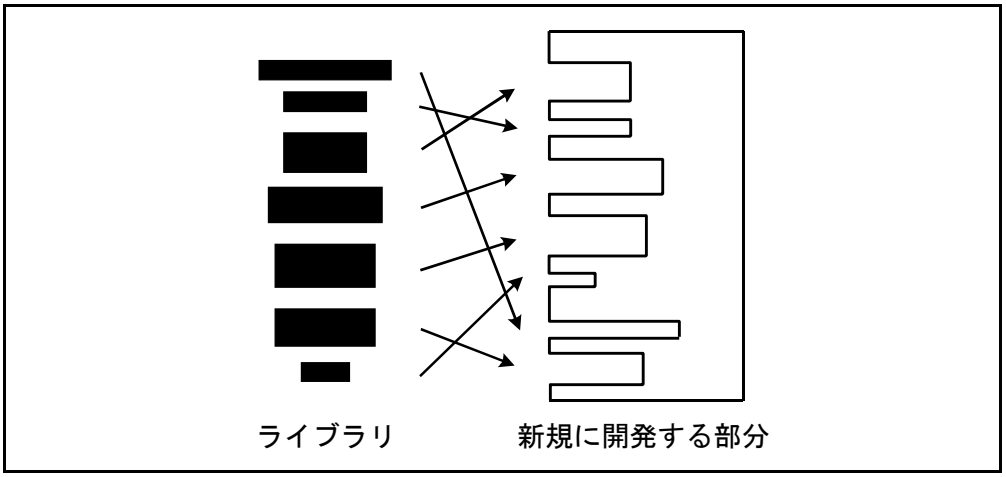



図2-10 ライブラリを用いたプログラム開発のイメージ

## 2.5.1 プログラムバンク

WINCAPSⅢにはライブラリを使うためのプログラムバンクが用意されています。プログラムバンクは、プログラムをライブラリとして登録したり、あるいは登録されているプログラムをプロジェクトに追加するためのツールです。
プログラムバンクの操作方法については、WINCAPSⅢガイド「プログラムバンク」を参照してください。

### 2.5.1.1 プログラムライブラリの分類

プログラムバンクにあらかじめ収録されているプログラムはカテゴリによって分類されています。
ユーザによって新たにライブラリに登録するプログラムは新しいカテゴリを作成して登録してください。詳しくはプログラミングマニュアル2を参照ください。

## 2.5.2　パレタイジングライブラリ

PAC言語には、パレタイジングのための専用命令はありませんが、ライブラリとして用意されているプログラムを使用することで、パレタイジング動作を実現できます。
パレタイジングのためのライブラリは、プログラムバンクを使ってプロジェクトに追加することができます。

### 2.5.2.1　パレタイジング

図2-12に示すような仕切りのあるパレットに、部品などの投入や取り出しを順次行なうことを、パレタイジングといいます。
パレットの仕切りの数、四隅の位置をティーチングするだけで、パレタイジング動作を行なえるように、パレタイジング用のライブラリプログラムは作られています。
パレタイジングのプログラムは、動作のために呼び出すごとに、パレットの取り出し位置を順番に変えていきます。

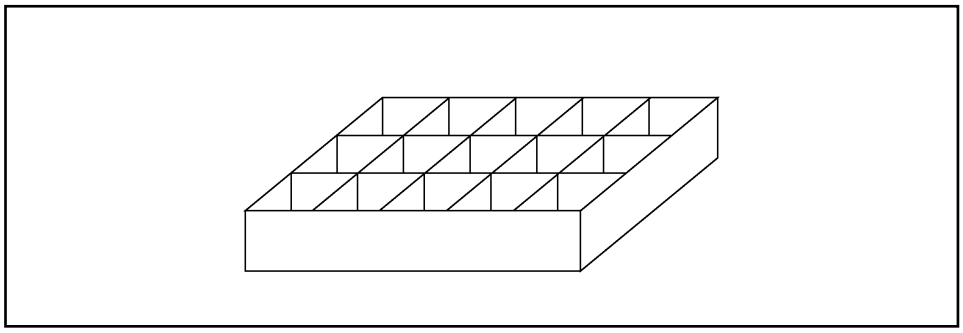


図 2-12　仕切りのあるパレット

## [ 1 ] パレタイジングのパラメータ

パレタイジングを行なうために必要なパラメータを、図2-13、図2-14、図2-15、表2-1に示します。
PAC言語では、これらのパラメータを変数の値として保持しています。

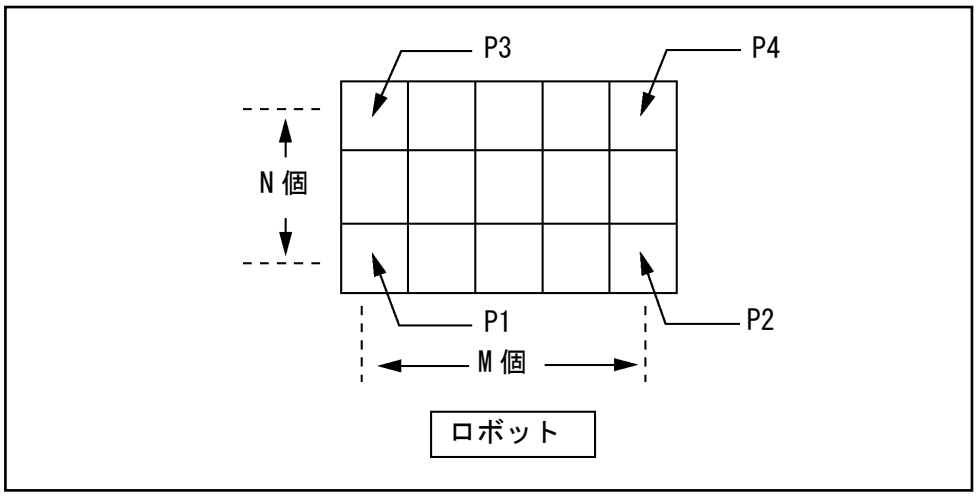



図 2-13　パレットの上視図

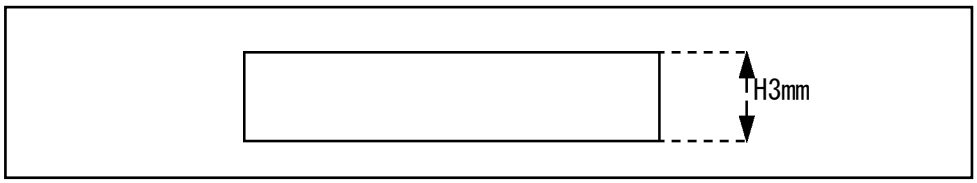



図 2-14　パレットの横視図

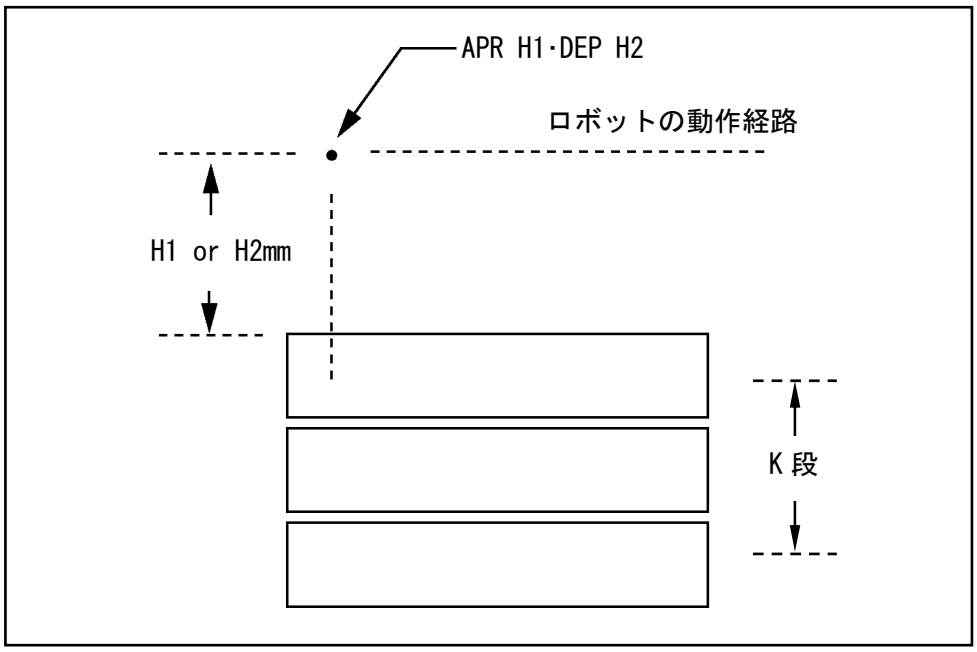



図 2-15　パレットの段積図

表 2-3　パレタイジングに必要なパラメータ

| 記号 | 名称 | 意味 | 単位 |
|---|---|---|---|
| | パレタイジング番号 | パレタイジングのインデックス番号 | なし（整数） |
| N | 横分割数 | P1 から P3 方向への分割数 | 個（整数） |
| M | 縦分割数 | P1 から P2 方向への分割数 | 個（整数） |
| K | 段積数 | パレットの段積数 | 個（整数） |
| H1 | アプローチ長 | ロボットがパレットに近づくときのアプローチ長 | mm(単精度実数) |
| H2 | デパート長 | ロボットがパレットから離れるときのデパート長 | mm(単精度実数) |
| H3 | パレット高さ | パレットの 1 段の高さ | mm(単精度実数) |
| | ただし、H1 と H2 は、次の条件を満たさなければなりません。<br>H1　＞　{H3×（K-1)} +5<br>H2　＞　{H3×（K-1)} +5 | | |
| P1<br>P2<br>P3<br>P4 | 図 2-13 に示すパレット四隅の点。各点の相対位置関係は、入れ替えができません。<br>またロボットの姿勢は、P1 の位置をティーチングしたときの姿勢が、すべての点で維持されます。 | | |

N　横分割数
パレットの横方向の分割数を示します。
図2-13では3列になっています。

M　縦分割数
パレットの縦方向の分割数を示します。
図2-13では5列になっています。

K　段積数
パレットの段積数を示します。
図2-15では3段になっています。

H1　アプローチ長
ロボットがパレットに近づくときのアプローチ長を示します。
パレタイジングプログラムを呼び出すたびに、同じアプローチ長を使用します。

H2　デパート長
ロボットがパレットから離れるときのデパート長を示します。
パレタイジングプログラムを呼び出すたびに、同じデパート長を使用します。

H3　パレット高さ
パレットの1段の高さを示します。
パレットが次第に積み上がっていく場合には、プラスの値を入力します。
パレットが次第に減っていく場合には、マイナスの値を入力します。
パレットの段数が変化しない場合には、0を入力します。

注意：H1とH2は、次の条件を満たさなければなりません。

H1 ＞ {H3×（K-1）} +5

H2 ＞ {H3×（K-1）} +5

この条件を満たさない場合には初期化時にエラーが表示されます。ロボットがパレットに衝突しないように、この条件があります。パレットの段数が最も多いときよりもさらに 5㎜高い点をアプローチ、デパートの位置にするためです。段積みが増減しても、図2-16に示すように、アプローチ点、デパート点は同じです。

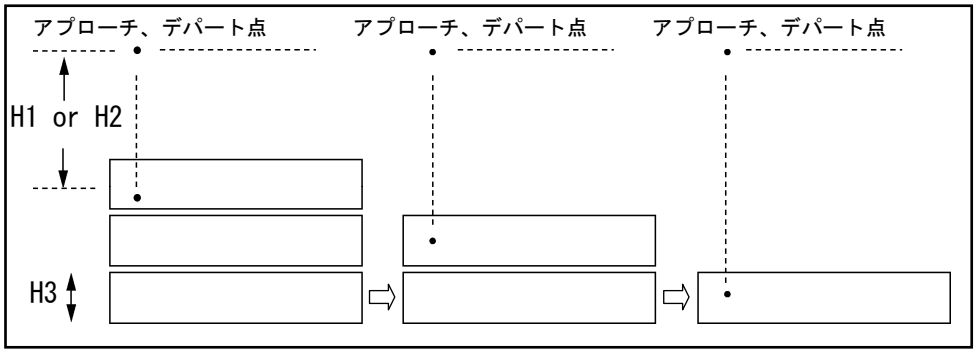



図 2-16　段積みの変化とアプローチ点、デパート点

P1、P2、P3、P4　四隅の点

パレットの四隅の部品位置を示します。各点と実行順序の関係は、図2-17のとおりです。

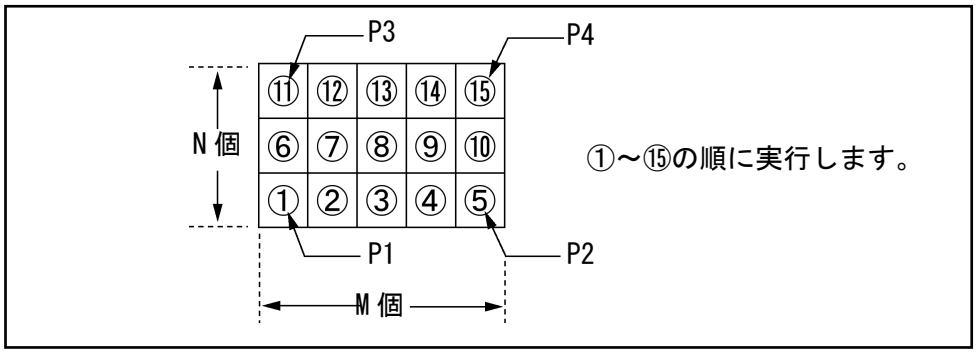



図 2-17　パレタイジングの順番

## [2] パラメータ値の設定

パレタイジングにおける、縦横の分割数や段積数など、パラメータ値の設定は、ライブラリプログラムの「pltInitialize」を呼び出すことによって行ないます。

「システムプロジェクトの新規作成」を行なうときに、[プロジェクトの新規作成] ダイアログボックスの [設備タイプ] の欄で、[1-パレタイジング] を選択すると、ライブラリの中に、「パレタイジング初期化テンプレート1」というタイトルのプログラム「PRO2」が自動的に登録されます。このプログラムは、「pltInitialize」を呼び出すようになっていますから、CALL 文の引数の値を、適当な値に変更して使います。

```
'!TITLE “パレタイジング 初期化テンプレート 1”
'!AUTHOR “デンソーロボット技術部”

PROGRAM PRO2
    CALL pltInitialize(0,4,3,1,50,50,50,52,53,54,55)    '適当な名前に変更して下さい。
END
```

**図 2-18　ライブラリプログラム「パレタイジング初期化テンプレート 1」**

```
CALL pltInitialize(0, 3, 5, 3, 50, 50, 10, 52, 53, 54, 55)のパラメータの説明
  1 番目... パレタイジングプログラム番号 (0)
  2 番目... 横分割数 (3)
  3 番目... 縦分割数 (5)
  4 番目... 段積数 (3)
  5 番目... アプローチ長 (50mm)
  6 番目... デパート長 (50mm)
  7 番目... パレットの高さ (10mm)
  8 番目... P1 位置 (P52)  ┐
  9 番目... P2 位置 (P53)  ├ P 型変数の番号のみ指定します。
 10 番目... P3 位置 (P54)  ┤
 11 番目... P4 位置 (P55)  ┘
```

**図 2-19　「pltInitialize」のパラメータの説明**

このパラメータの意味は、WINCAPSⅢの「プログラムバンク」というツールでも見ることができます。

## [ 3 ] パレタイジングカウンタ

パレタイジングでは、動作中にパレットの仕切り数を数え、変数にカウント値を保持します。
カウンタは、横方向（N)、縦方向（M)、高さ方向（K)、トータル（cnt）の4つがあります。これらのカウンタは、パレタイジング動作を制御する中核プログラム、「pltKernl」の中で定義されています。
ライブラリプログラムの「pltMove」は、パレットの位置に対して一つの作業を完了するごとに、トータルカウンタの値を1つ加算し、他のカウンタの値も調整します。
ライブラリプログラムの「pltDecCnt」は、呼び出すことで、トータルカウンタの値を1つ減算し、各カウンタの値を減算調整します。
パレタイジングプログラムは、デフォルト設定では30個まで作成可能です。したがって、パレタイジングカウンタも31セットあります。パレタイジングプログラム数を変更する方法については、p.2-25「[ 1 ] パレタイジングプログラムのカスタマイズ」を参照してください。

### カウント規則

パレタイジングカウンタは、「pltMove」が実行されるたびに、トータルカウンタの値を1つ加算し、他のカウンタの値を調整します。これにより、次のパレット位置が保証されます。
トータルカウンタが一つ加算されると、縦方向カウンタ（M）の位置が次へ1つ移動します。縦方向カウンタ（M）の位置が端まで来て最大値になると、横方向カウンタ（N）の位置が次へ1つ移動し、縦方向カウンタ（M）の位置は設定された最小値になります。横方向カウンタ（N）の位置が端まで来て最大値になると、高さ方向カウンタ（K）の位置が次へ1つ移動し、縦方向カウンタ（M）と横方向カウンタ（N）は、設定された最小値になります。
パレタイジングプログラムを、途中でプログラム停止して再開すると、カウンタ変数の値はカウントアップされているので次の位置へ動きます。
電源を切っても、パレタイジングカウンタの内容は保持されます。電源再投入後に、パレタイジングカウンタを初期化しなければ、前回終了時の続きからパレタイジング動作を行ないます。

**注意：コンパイル後、実行プログラムのロードにて変数の値が初期化されます。**

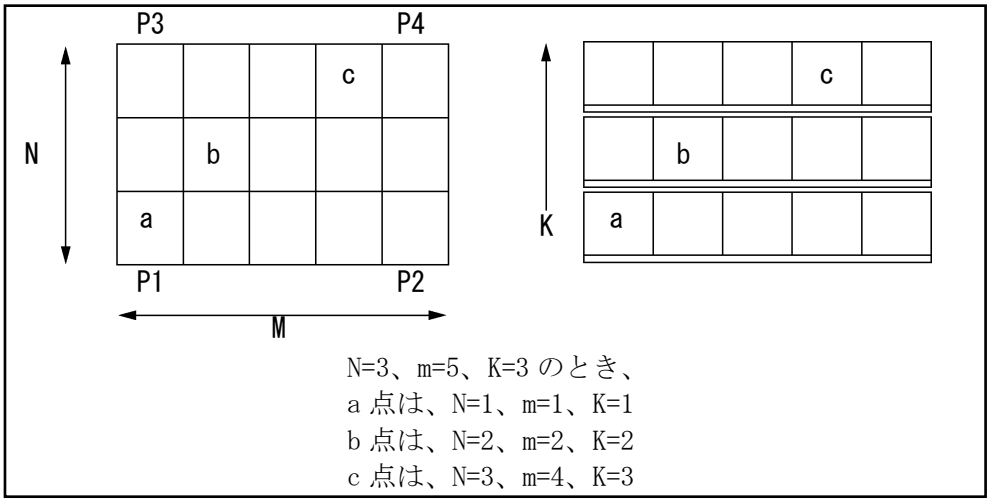



**図 2-20　パレタイジング位置とカウンタの関係**

## カウンタの初期化

パレットの入れ替えや、パレットのすべてのます目を使用しない場合に、カウンタを初期化します。

カウンタの初期化では、すべてのパレタイジングカウンタに「1」を代入します。

ライブラリプログラムの「pltResetAll」を使用すると、すべてのパレタイジングカウンタを一度に初期化できます。

たとえば、パレット番号1のすべてのカウンタを初期化するには、次のように記述します。

```
CALL   pltResetAll(1)
```

各パレタイジングカウンタを個別に初期化するには、「pltLetN1」、「pltLetM1」、「pltLetK1」、「pltLetCnt」を使います。

たとえば、パレット番号1のNカウンタを初期化するには、次のように記述します。

```
CALL   pltLetN1(1, 1)
```

**注意：第 2 引数がカウンタに設定する値です。1 以外の値に設定することもできます。**

## パレタイジングプログラムの終了信号

パレタイジングプログラムは、1段終了および全段終了すると、1段終了フラグおよび全段終了フラグをセットします。

1段終了フラグの状態を取得するには、ライブラリプログラム「pltGetPLT1END」を使います。全段終了フラグの状態を取得するには、ライブラリプログラム「pltGetPLTEND」を使います。

1段終了フラグをリセット（0）するには、ライブラリプログラム「pltResetPLT1END」を使います。全段終了フラグをリセット（0）するには、ライブラリプログラム「pltResetPLTEND」を使います。

## 2.5.2.2 パレタイジングプログラム

実際のパレタイジングプログラムは、そのアプリケーションでの特殊な状況によってさまざまですが、ライブラリを使ってプログラムを組み立てる場合の標準的な手順を、タイトル「パレタイジングテンプレート2」のプログラム名「PRO1」に用意してあります。
この「パレタイジングテンプレート2」を下敷きにして、各アプリケーションの必要事項を書き足し、プログラム開発に役立ててください。
図2-21に、プログラム名「PRO1」の「パレタイジングテンプレート2」を示します。この例では、以下のような設備を想定しています。

- パレタイジングポイントはP50～P55とします。
- 作成プログラムは、定位置P50に移動し、パレタイジングプログラム0番を実行して、組み付け位置P51に移動し、アンチャック動作を行ない、段終了を確認し、終了信号が出力されていれば必要に応じてパレット入れ替え動作を行ない、ワークがなくなると終了します。

```
' !TITLE “パレタイジングテンプレート 2”
' !AUTHOR “デンソーロボット技術部”
#DEFINE pltIndex    0                       'パレタイジングプログラム番号
                                            '0～30 までの任意の数値を選択可。
#DEFINE ChuckNG        40                   '取り出し検査 IO の番号
                                            '任意の数値を選択可。

PROGRAM PRO1                                '適当な名前に変更してください。
  TAKEARM                                   'アームセマフォの取得
  MOVE P, P50                               '定位置 P50 へ移動。
                                            'パレタイジングされた P50 に移動。
  IF IO[ChuckNG] = ON THEN                  '前回の取り出し検査。
    CALL pltDecCnt(pltIndex)                '総カウンタを-1 する。
  END IF
  CALL pltMove(pltIndex)                    'パレタイジング番号 0 実行。
  MOVE P, P51                               '組み付け位置 P51 へ移動。
                                            'パレタイジングされた P51 に移動。
  '<--- アンチャック動作などを挿入 --->
  CALL pltGetPLT1END(pltIndex,0)            'I[0]に 1 段終了信号取得
                                            '2 個目の引数“0”は次の行で判定
                                            'される“I”の行列番号
  IF I[0] THEN                              'ON のとき(つまり<>0 のとき)
    '<--- パレットチェンジ動作などを挿入 --->
    CALL pltResetPLT1END(pltIndex)          '1 段終了信号クリア
  END IF
  GIVEARM                                   'アームセマフォの開放
END
```

図 2-21　プログラム名「PRO1」、「パレタイジングテンプレート 2」

## [1] パレタイジングプログラムのカスタマイズ

**Q.** 実行するパレタイジングプログラムを１に変更するには？

**A.** 「PRO1」および「PRO2」の「pltIndex」及び「0」の値を「1」に変更します。

**Q.** パレタイジングの分割を横５段、縦３段に変更するには？

**A.** 「pltInitialize」の 2番目、3番目のパラメータ「3」と「5」をそれぞれ「5」と「3」に変更してください。

☆ CALL pltInitialize(0, 5, 3, 3, 50, 50, 10, 52, 53, 54, 55)

**Q.** 使用できるパレタイジングプログラム数を変更するには？

**A.** 「pltKernel」プログラムの「#DEFINE mcPaltMax 31」の「31」を任意の数値に変更することにより、パレタイジングプログラム数を変更できます。

**Q.** パレタイジングの回避ポイントを設定するには？

**A.** アプローチ方向に対して、X = 10、Y = 10、Z = -10 の回避ポイントを追加する場合、「paltmove1.pac」ファイルの　P[mcNextPos]点に、並進偏差分「(10, 10, -10, 0, 0, 0)H」を加算した点への移動コマンドを追加してください。

```
☆ MOVE P, P[mcNextPos] + (10, 10, -10, 0, 0, 0)H
   'X = 10、Y = 10、Z = -10 の回避ポイントに移動
```

**Q.** N1、M1、K1 を取得するには？

**A.** プログラムライブラリ「pltGetK1」、「pltGetM1」、「pltGetN1」を追加してください。

```
☆ call pltGetK1(Index, iValue)
☆ call pltGetM1(Index, iValue)
☆ call pltGetN1(Index, iValue)
   'Index ... パレタイジングプログラム番号。
   'iValue... K1, M1, N１の値が返されるＩ型変数の番号。
```

例）

```
#define pltIndex    0       'パレタイジングプログラム番号
Program Exmp
DefInt  Index, iValue
Index = pltIndex  'Index レジスタにパレタイジングプロ
                  'グラム番号を挿入。
call pltGetK1(Index, iValue)    'iValue にて指定したＩ
                                '型変数に K1 の値が返
                                'されます。
call pltGetM1(Index, iValue)    'iValue にて指定しＩ
                                '型変数に M1 の値が返
                                'されます。
call pltGetN1(Index, iValue)    'iValue にて指定したＩ
                                '型変数に N1 の値が返
                                'されます。
End
```

Q. N1, M1, K1 を変更するには？

A. プログラムライブラリ“pltLetK1”、“pltLetM1”、“pltLetN1”を追加してください。

☆ call pltLetK1(Index, iValue)
☆ call pltLetM1(Index, iValue)
☆ call pltLetN1(Index, iValue)
'Index ... パレタイジングプログラム番号。
'iValue... この値が K1, M1, N1 に入力されます。

例）
```
#define pltIndex    0              'パレタイジングプログラム番
                                   '号
Program Exmp
DefInt  Index, iValue
Index  = pltIndex          'Index レジスタにパレタイジ
                           'ングプログラム番号を挿入。
iValue = 10        'K1, M1, N1 に入れる値を入力します。
call pltLetK1(Index, iValue)      'K1 に iValue の値が
                                  '入力されます。
call pltLetM1(Index, iValue)      'M1 に iValue の値が
                                  '入力されます。
call pltLetN1(Index, iValue)      'N1 に iValue の値が
                                  '入力されます。
End
```

Q. 総カウンタを取得するには？

A. プログラムライブラリ“pltGetCnt”を追加してください。

☆ call pltGetCnt(Index, iValue)
'Index ... パレタイジングプログラム番号。
'iValue... 総カウンタの値が返される I 型変数の番号。

例）
```
#define pltIndex    0              'パレタイジングプログラム番
                                   '号
Program Exmp
DefInt  Index, iValue
Index = pltIndex           'Index レジスタにパレタイジン
                           'グプログラム番号を挿入。
call pltGetCnt(Index, iValue)     'iValue に総カウンタ
                                  '値が返されます。
End
```

Q．総カウンタを変更するには？

A．プログラムライブラリ「pltLetCnt」を追加してください。

☆ call pltLetCnt(Index, iValue)

'Index ... パレタイジングプログラム番号。

'iValue... この値が総カウンタに入力されます。

例）#define pltIndex　0　　'パレタイジングプログラム番
'号

```
Program Exmp
DefInt  Index, iValue
Index  = pltIndex        'Index レジスタにパレタイジン
                         'グプログラム番号を挿入。
iValue = 0        ' 総カウンタに入れる値を入力します。
call pltLetCnt(Index, iValue)     '総カウンタに iValue
                                  'の値が入力されます。
End
```

Q．総カウンタはどのタイミングで加算されているのですか？

A．「pltKernel(Index, -15, mcNextPos, ndErr)」を実行されたときに加算されます。したがって、プログラムライブラリの「pltGetNextPos」、「pltMove」、「pltMove0」が実行されると総カウンタが加算されます。

☆ call pltMove(Index)

☆ call pltMove0

☆ call pltGetNextPos(Index, NextPos)

'Index ... パレタイジングプログラム番号。

'NextPos .. 次位置のポジションが入力されるＰ型変数の番号。

## 2.6 TP 簡易操作盤 [Ver.1.5 以降]

任意の大きさ・位置・色のボタンの作成、画面への張り込みをPAC言語にて実現します。またボタン、ページ設定PACプログラムを一度動作させることで操作盤を有効とします。なお一度設定するとクリア操作を行わない限り操作盤は動作し続けます。

### 使用できるボタン・画面

ボタン・・・500個まで
画面・・・・50画面まで

### 操作盤画面作成のためのコマンド

| | |
|---|---|
| set_button | ボタンパラメータの設定（属性により表示ON/OFFも兼ねる） |
| set_page | ページパラメータの設定（属性により表示ON/OFFも兼ねる） |
| change_bCap | ボタンキャプション変更 |
| change_pCap | ページキャプション変更 |
| disp_page | ページ表示 |

### コマンドにより設定されるパラメータ

**ボタンパラメータ**

- インデックス
- 表示位置（左上x座標、左上y座標、右下x座標、右下y座標）
- ボタンタイプ（タッチスイッチ[形状のバリエーション有り]、数値表示ボックス、数値入力ボックス、デジスイッチ、ランプ[形状のバリエーション有り]、ページ切り替えボタン）
- ステータス（予約）
- 背面色
- 文字色
- 有効フラグ
- 表示フラグ
- キャプション
- 変数タイプ
- 変数番号
- IO番号
- ページ番号
- 変更フラグ（予約）
- 結果（予約）

**ページパラメータ**

インデックス、画面タイプ（固定）、ステータス（予約）、背面色、文字色、有効フラグ、表示フラグ、キャプション、変更フラグ（予約）、結果（予約）

## TP 簡易操作盤をプログラムする

操作盤画面の表示を行なうためには以下の操作を行なってください。

(1)　ボタンパラメータの設定

set_button命令を使って、ボタンのパラメータを決定します。これをすべてのパラメータについて実行します。

(例)　番号１のボタンの背景色(7)を黒(0)に設定する。

set_button 1、7、0

(2)　ページパラメータの設定

set_page命令を使って、ページのパラメータを決定します。これをすべてのパラメータについて実行します。

(例)　2ページ目のページの背景色(3)を赤(4)に設定する。

set_page 2、3、4

(3)　ボタンキャプションの設定

change_bCap命令を使って、ボタンのキャプションを決定します。

(例)　番号2のボタンのキャプションを“準備”に設定する。

change_bCap　2、“準備”

(4)　ページキャプションの設定

change_pCap命令を使って、ページのキャプションを決定します。

(例)　3ページ目のキャプションを“画面3”に設定する。

change_pCap　3、“画面3”

(5)　ページの表示

disp_page命令を使って、ページを表示します。

(6)　操作盤画面の選択

ティーチングペンダントの基本画面から[拡張画面]－[操作盤]を選択し、操作盤を表示します。

(画面例)



## 2.6.1 操作盤画面作成のプログラム例

全体の命令を用いた操作盤画面作成のプログラム例を次に示します。

```
'!TITLE "<ﾀｲﾄﾙ>"
PROGRAM BUTTON3D_VAL_3
'色の定義
#define BLACK          0          '黒
#define BLUE           1          '青
#define GREEN          2          '緑
#define CYAN           3          'シアン
#define RED            4          '赤
#define MAGENTA        5          'マジェンタ
#define BROWN          6          '茶
#define LIGHTGLAY      7          '明るい灰色
#define GLAY           8          '灰色
#define LIGHTBLUE      9          '明るい青
#define LIGHTGREEN     10         '明るい緑
#define LIGHTCYAN      11         '明るいシアン
#define LIGHTRED       12         '明るい赤
#define LIGHTMAGENTA   13         '明るいマジェンタ
#define YELLOW         14         '黄
#define WHITE          15         '白

'ボタンの定義
#define LABEL          1          'ラベル
#define LINE           2          '線
#define 2DBUTTON_V     17         '2Dボタン(変数変更)
#define 3DBUTTON_V     18         '3Dボタン(変数変更)
#define 3DBUTTON_V2    19         '3Dボタン2(変数変更)
#define CIRCLE         20         '円(変数変更)
#define 2DLED          33         '2DLED(ランプ)
#define CIRCLELED      34         '円LED(ランプ)
#define 3DBUTTON_IO    35         '3Dボタン(IO変更)

'ページパラメータ
#define P_BGCOLOR      3          'ページ背景色
#define P_CHARCOLOR    4          'ページ文字色
#define P_USESTATE     5          '使用可能状態
#define P_VISSTATE     6          '可視状態
```

```
'ボタンパラメータ
#define X_UPPERLEFT_P    1          '左上Ｘ座標
#define Y_UPPERLEFT_P    2          '左上Ｙ座標
#define X_LOWERRIGHT_P   3          '右下Ｘ座標
#define Y_LOWERRIGHT_P   4          '右下Ｙ座標
#define B_KIND           5          'ボタン種類
#define B_BGCOLOR        7          'ボタン背景色
#define B_FGCOLOR        8          'ボタン文字色
#define B_USESTATUS      9          'ボタン使用可能状態
#define B_VISSTATUS      10         'ボタン可視状態
#define B_VALUEKIND      11         'ボタン変数種類(固定)
#define B_VALUE_NO       12         'ボタン変数番号
#define B_IO_NO          13         'ボタンI/O番号
#define B_DISP_PNO       14         'ボタンの表示ﾍﾟｰｼﾞ番号

'状態定義
#define ON               1          '使用する
#define OFF              0          '使用しない
#define I_VAL            1          '整数型

'ボタンアドレス
#define IO_PB_ADRS       170        '操作盤ボタン先頭I/O番号

defint btn_no,minx,maxx,miny,maxy,loopcnt
defint enable,visible,var_type,var_index
defint panel_no,io_no,io_adrs,btn_adrs
defstr panel_cap
defstr bcap0,bcap1,bcap2,bcap3,bcap4,bcap5,bcap6,bcap7
       panel_no = 3

       panel_cap = "段取画面設定(画面3)"
       loopcnt = 0

       change_pCap panel_no,panel_cap      'ページのタイトル設定
       set_page panel_no,P_BGCOLOR,GLAY    'ページの背景色設定
       set_page panel_no,P_USESTATE,ON     'ページ使用の設定
       set_page panel_no,P_VISSTATE,ON     'ページ可視の設定


'各パラメータの初期化
       btn_adrs = 30
       io_no = IO_PB_ADRS
       reset io[128 to 133]
       enable = ON
       visible = ON
```

```
var_type = I_VAL
var_index = 1

bcap0 = "段取り"+chr$(10)+"データ１"
bcap1 = "段取り"+chr$(10)+"データ２"
bcap2 = "段取り"+chr$(10)+"データ３"
bcap3 = "段取り"+chr$(10)+"データ４"
bcap4 = "段取り"+chr$(10)+"データ５"
bcap5 = "段取り"+chr$(10)+"データ６"
bcap6 = "画面０"+chr$(10)+"へ戻る"
bcap7 = "ここを"+chr$(10)+"タッチ"

while loopcnt < 6        '6回ループ
        btn_no = btn_adrs + loopcnt
        minx = 10 + ((loopcnt mod 6)*100)
        miny = 50 + ((loopcnt / 6)*100)
        maxx = 100 + ((loopcnt mod 6)*100)
        maxy = 120 + ((loopcnt / 6)*100)
        io_adrs = IO_PB_ADRS + loopcnt

'ここは共通処理
        set_button (btn_no),(1),(minx)
        set_button (btn_no),(2),(miny)
        set_button (btn_no),(3),(maxx)
        set_button (btn_no),(4),(maxy)
        set_button (btn_no),(9),(enable)
        set_button (btn_no),(10),(visible)
        set_button (btn_no),(11),(var_type)
        set_button (btn_no),(12),(var_index)
        set_button (btn_no),(14),(panel_no)


'ラベル表示
        select case loopcnt
        case 0
                set_button btn_no,B_KIND,LABEL          'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs       'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,GLAY        '背景色設定
                set_button btn_no,B_FGCOLOR,RED         '前面色設定
                change_bCap btn_no,bcap0                'ボタン名設定

        case 1
                set_button btn_no,B_KIND,LABEL          'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs       'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,GLAY        '背景色設定
```

```
                set_button btn_no,B_FGCOLOR,BLACK          '前面色設定
                change_bCap btn_no,bcap1                   'ボタン名設定

            case 2
                set_button btn_no,B_KIND,LABEL             'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs          'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,GLAY           '背景色設定
                set_button btn_no,B_FGCOLOR,WHITE          '前面色設定
                change_bCap btn_no,bcap2                   'ボタン名設定

            case 3
                set_button btn_no,B_KIND,LABEL             'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs          'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,GLAY           '背景色設定
                set_button btn_no,B_FGCOLOR,YELLOW         '前面色設定
                change_bCap btn_no,bcap3                   'ボタン名設定

            case 4
                set_button btn_no,B_KIND,LABEL             'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs          'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,GLAY           '背景色設定
                set_button btn_no,B_FGCOLOR,LIGHTMAGENTA   '前面色設定
                change_bCap btn_no,bcap4                   'ボタン名設定


            case 5
                set_button btn_no,B_KIND,LABEL             'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs          'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,GLAY           '背景色設定
                set_button btn_no,B_FGCOLOR,LIGHTBLUE      '前面色設定
                change_bCap btn_no,bcap5                   'ボタン名設定
            end select
            loopcnt = loopcnt + 1
        wend

'下段表示
        loopcnt = 0
        while loopcnt < 6                  '6回ループ
            btn_no = btn_adrs + 10 + loopcnt
            minx = 10 + ((loopcnt mod 6)*100)
            miny = 120 + ((loopcnt / 6)*100)
            maxx = 100 + ((loopcnt mod 6)*100)
            maxy = 190 + ((loopcnt / 6)*100)
            io_adrs = IO_PB_ADRS+6+loopcnt
            var_index = var_index + 1
```

```
'ここは共通処理
                set_button (btn_no),(1),(minx)
                set_button (btn_no),(2),(miny)
                set_button (btn_no),(3),(maxx)
                set_button (btn_no),(4),(maxy)
                set_button (btn_no),(9),(enable)
                set_button (btn_no),(10),(visible)
                set_button (btn_no),(11),(var_type)
                set_button (btn_no),(12),(var_index)
                set_button (btn_no),(14),(panel_no)


'ラベル表示
        select case loopcnt
        case 0
'               set_button btn_no,B_KIND,3DBUTTON_V      'ボタンタイプ設定
                set_button btn_no,B_KIND,3DBUTTON_V2     'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs        'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,MAGENTA      '背景色設定
                set_button btn_no,B_FGCOLOR,RED          '前面色設定
                var_index = loopcnt
                set_button btn_no,B_VALUE_NO,var_index
                change_bCap btn_no,bcap7

        case 1
'               set_button btn_no,B_KIND,3DBUTTON_V      'ボタンタイプ設定
                set_button btn_no,B_KIND,3DBUTTON_V2     'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs        'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,LIGHTGLAY    '背景色設定
                set_button btn_no,B_FGCOLOR,BLACK        '前面色設定
                var_index = loopcnt
                set_button btn_no,B_VALUE_NO,var_index
                change_bCap btn_no,bcap7

        case 2
'               set_button btn_no,B_KIND,3DBUTTON_V      'ボタンタイプ設定
                set_button btn_no,B_KIND,3DBUTTON_V2     'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs        'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,BLUE         '背景色設定
                set_button btn_no,B_FGCOLOR,WHITE        '前面色設定
                var_index = loopcnt
                set_button btn_no,B_VALUE_NO,var_index
                change_bCap btn_no,bcap7
```

```
                case 3
'                       set_button btn_no,B_KIND,3DBUTTON_V      'ボタンタイプ設定
                        set_button btn_no,B_KIND,3DBUTTON_V2     'ボタンタイプ設定
                        set_button btn_no,B_IO_NO,io_adrs        'I/Oアドレス設定
                        set_button btn_no,B_BGCOLOR,GREEN        '背景色設定
                        set_button btn_no,B_FGCOLOR,YELLOW       '前面色設定
                        var_index = loopcnt
                        set_button btn_no,B_VALUE_NO,var_index
                        change_bCap btn_no,bcap7

                case 4
'                       set_button btn_no,B_KIND,3DBUTTON_V      'ボタンタイプ設定
                        set_button btn_no,B_KIND,3DBUTTON_V2     'ボタンタイプ設定
                        set_button btn_no,B_IO_NO,io_adrs        'I/Oアドレス設定
                        set_button btn_no,B_BGCOLOR,CYAN         '背景色設定
                        set_button btn_no,B_FGCOLOR,LIGHTMAGENTA '前面色設定
                        var_index = loopcnt
                        set_button btn_no,B_VALUE_NO,var_index
                        change_bCap btn_no,bcap7

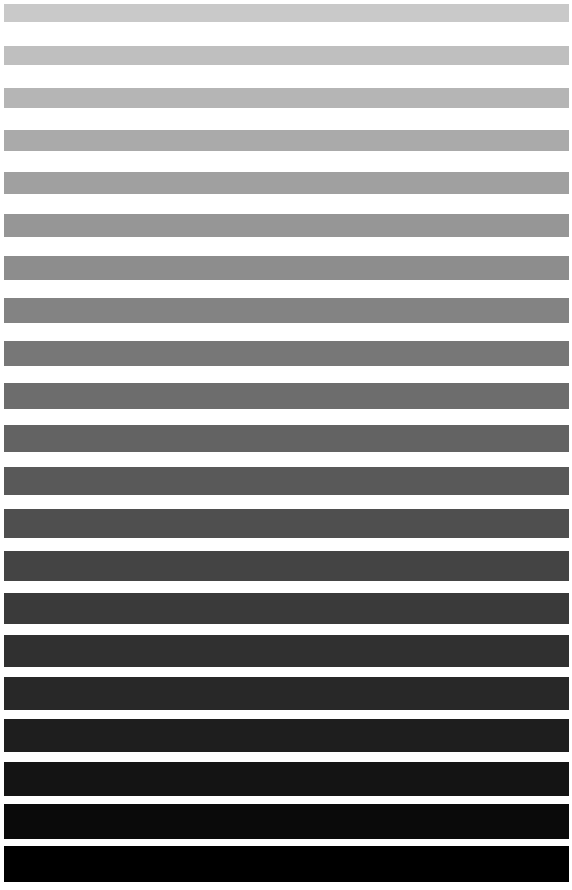

                case 5
'                       set_button btn_no,B_KIND,3DBUTTON_V      'ボタンタイプ設定
                        set_button btn_no,B_KIND,3DBUTTON_V2     'ボタンタイプ設定
                        set_button btn_no,B_IO_NO,io_adrs        'I/Oアドレス設定
                        set_button btn_no,B_BGCOLOR,GLAY         '背景色設定
                        set_button btn_no,B_FGCOLOR,LIGHTBLUE    '前面色設定
                        var_index = loopcnt
                        set_button btn_no,B_VALUE_NO,var_index
                        change_bCap btn_no,bcap7

                end select
                loopcnt = loopcnt + 1
        wend
'画面０へ戻る３Ｄボタン作成
                io_adrs = 128
                minx = 510
                maxx = 600
                miny = 250
                maxy = 320
                btn_no = 92
                set_button (btn_no),(1),(minx)
                set_button (btn_no),(2),(miny)
                set_button (btn_no),(3),(maxx)
                set_button (btn_no),(4),(maxy)
```

```
                set_button (btn_no),(9),(enable)
                set_button (btn_no),(10),(visible)
                set_button (btn_no),(11),(var_type)
                set_button (btn_no),(12),(var_index)
                set_button (btn_no),(14),(panel_no)
                set_button btn_no,B_KIND,3DBUTTON_IO    'ボタンタイプ設定
                set_button btn_no,B_IO_NO,io_adrs       'I/Oアドレス設定
                set_button btn_no,B_BGCOLOR,BLUE        '背景色設定
                set_button btn_no,B_FGCOLOR,WHITE       '前面色設定
                change_bCap btn_no,bcap6                'ボタン名設定

        disp_page panel_no                              '指定画面の表示
END
```

# 第 3 章
# ロボットの動作の種類



ロボットの動作は、その基準とする位置の取り方や、目標位置に到達したことを判定する方法によって、いくつかの種類があります。この章では、こうしたロボットの動作の種類について説明します。

# 第３章　ロボットの動作の種類

## 3.1　絶対動作と相対動作

### 3.1.1　絶対動作

ティーチングされた動作位置へ移動する動作を、絶対動作といいます。
絶対動作は、その直前の動作に影響されることなく、必ずティーチングされた位置へ移動します。
絶対動作を行なうコマンドは、次のとおりです。

APPROACH、MOVE、GOHOME、DRIVEA

### 3.1.2　相対動作

現在位置から、ティーチングされた移動量だけ移動する動作のことを、相対動作といいます。
相対動作は、その直前の動作コマンドを実行した結果の現在位置を基準とするので、直前の動作コマンドの影響を受けることになります。
相対動作を行なうコマンドは、次のとおりです。

DEPART、DRAW、DRIVE、ROTATE、ROTATEH

### 3.1.3　絶対動作と相対動作の動作例

現在位置P1より、点P2を通り、点P3へ移動するプログラムの例を2種類示します。
「MOVEMENT1」は、絶対動作だけで記述されています。
「MOVEMENT2」は、絶対動作と相対動作によって記述されています。どちらのプログラムも、実行すると図3-3に示すとおり、同じ動きをします。

```
PROGRAM MOVEMENT1
    TAKEARM
    MOVE L, P2
    MOVE L, P3
END
```

図3-1　絶対動作のプログラム例

```
PROGRAM MOVEMENT2
    TAKEARM
    MOVE L, P2
    DRAW L, V3   'V3はP2とP3の相対距離
END
```

図3-2　相対動作のプログラム例

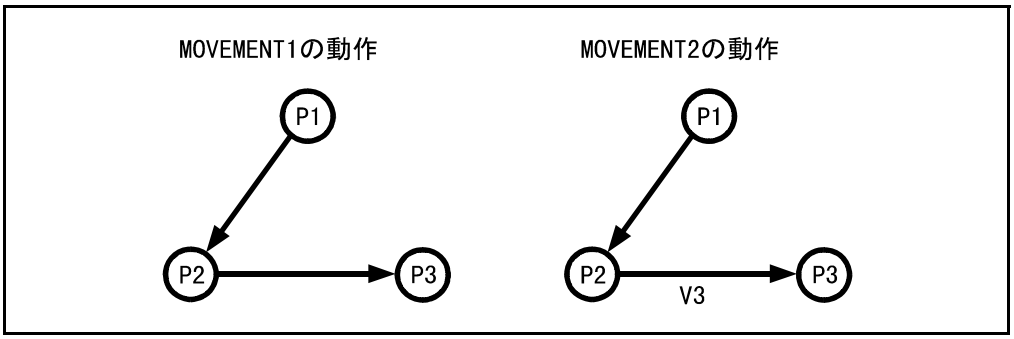



図3-3　2つのプログラムの動作例

ここで、「MOVEMENT1」と「MOVEMENT2」の最初の動作命令「MOVE P, P2」を削除すると、動作は図3-4に示すとおり、異なった動きになります。
「MOVEMENT1」では絶対動作により、点P3へ移動しますが、「MOVEMENT2」では、現在位置P1からV3だけ相対移動します。

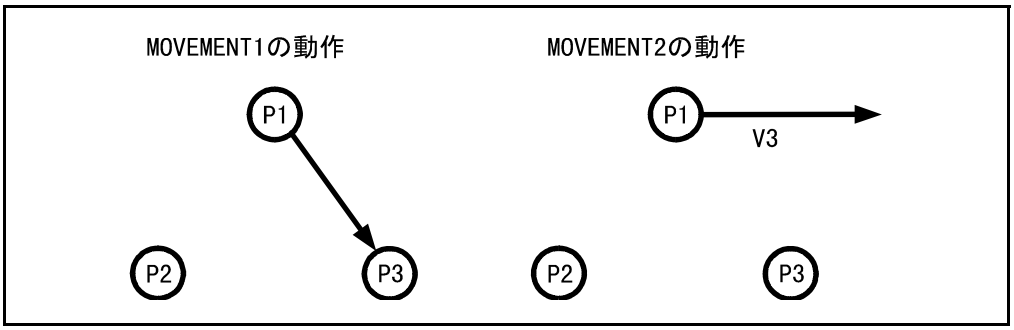



図3-4　2つのプログラムの動作例（削除後）

注意：相対動作は、現在の位置から指定された相対距離分の動作をします。したがって、INTERRUPT ON/OFF 命令（12.3 項 参照）にて動作命令をスキップさせた直後に相対動作させる場合、割り込み信号が ON されるタイミングにより動作終了位置が変化します。動作終了位置を固定する場合は、絶対動作を使用してください。

## 3.2　到達位置の確認方法

ロボットアームが、一つの動作から次の動作へと移る場合に、初めの動作の完了を判断する方法が3種類あります。それぞれ、パス動作、エンド動作、エンコーダ値確認動作といいます。各動作について、以下に説明します。

### 3.2.1　パス動作

ティーチングされた動作位置、または相対位置の近傍を通過する動作をパス動作といいます。
すべての動作コマンドで、パス開始変位量を「@P」または「@数値(数値＞0)」と指定することにより、パス動作を行なうことができます。

### 3.2.2　エンド動作

ティーチングされた動作位置、または相対位置へ到達する動作のうち、サーボ系への指令値が目標位置に一致したときに、目標位置に到達したと判断するものを、エンド動作といいます。
すべての動作コマンドで、パス開始変位量を「@0」と指定することにより、エンド動作を行なうことができます。

### 3.2.3　エンコーダ値確認動作

ティーチングされた動作位置、または相対位置へ到達する動作のうち、エンコーダ値が目標位置に対し、指定パルス(初期値20)以内に入ったときに、目標位置に到達したと判断するものを、エンコーダ値確認動作といいます。指定パルス数は各軸ごとにプログラム中で変更できます。
すべての動作コマンドで、パス開始変位量を「@E」と指定することにより、エンコーダ値確認動作を行なうことができます。

**注意①：エンコーダ値が目標位置に対し、指定パルス以上ずれていると、到達したことにはならず、次へ進めなくなります。指定パルスを小さくした場合や負荷が大きい場合などは、サーボ系の偏差量を考慮してください。**
**②：マシンロック運転中は「@E」指定された動作はすべて「@0」で動作します。したがって、表示されるプログラムの実行時間が、実際に動作させた場合よりも短くなります。**
**③：到達位置の確認方法を記述しない場合は、パス開始変位量にエンド動作「@0」が自動的に設定されます。**

## 3.2.4　パス動作、エンド動作、エンコーダ値確認動作の動作例

現在位置P1より、点P2を通り、点P3へ移動するプログラムの例を3種類示します。それぞれ、パス動作、エンド動作、エンコーダ値確認動作のプログラム例です。

```
PROGRAM PASS_MOVE
    TAKEARM
    MOVE P, @P P2
    MOVE P, @0 P3
END
```

図3-5　パス動作のプログラム例

```
PROGRAM END_MOVE
    TAKEARM
    MOVE L, @0 P2
    MOVE L, @0 P3
END
```

図3-6　エンド動作のプログラム例

```
PROGRAM ENCODER_MOVE
    TAKEARM
    MOVE L, @E P2
    MOVE L, @0 P3
END
```

図3-7　エンコーダ値確認動作のプログラム例

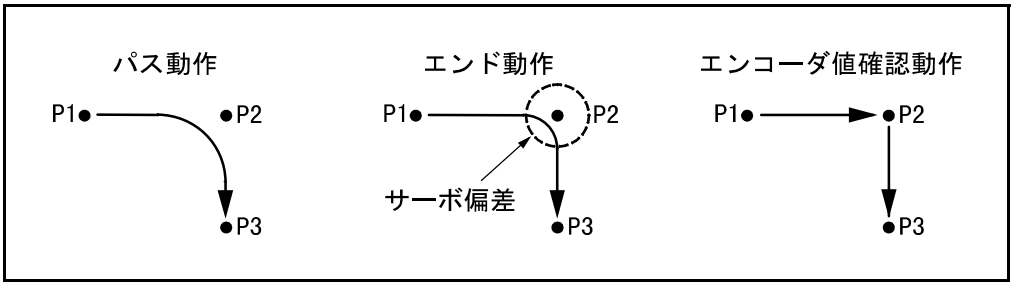



図3-8　パス動作、エンド動作、エンコーダ値確認動作の動作例

## 3.2.5　パス動作、エンド動作、エンコーダ値確認動作の実行時間の違い

3種類の動作の実行時間は、もっとも早いのがパス動作、次にエンド動作、もっとも時間がかかるのがエンコーダ値確認動作という順になります。
パス動作は図に示すように、P2到達前の減速時間中に次の加速動作を開始します。したがって、減速と加速をそれぞれ行なうエンド動作よりも動作時間が短くなります。
エンコーダ値確認動作では、目標位置到達を厳密にエンコーダ値によって確認するので、エンド動作よりもサーボ偏差を解消する時間分余計に時間がかかることになります。

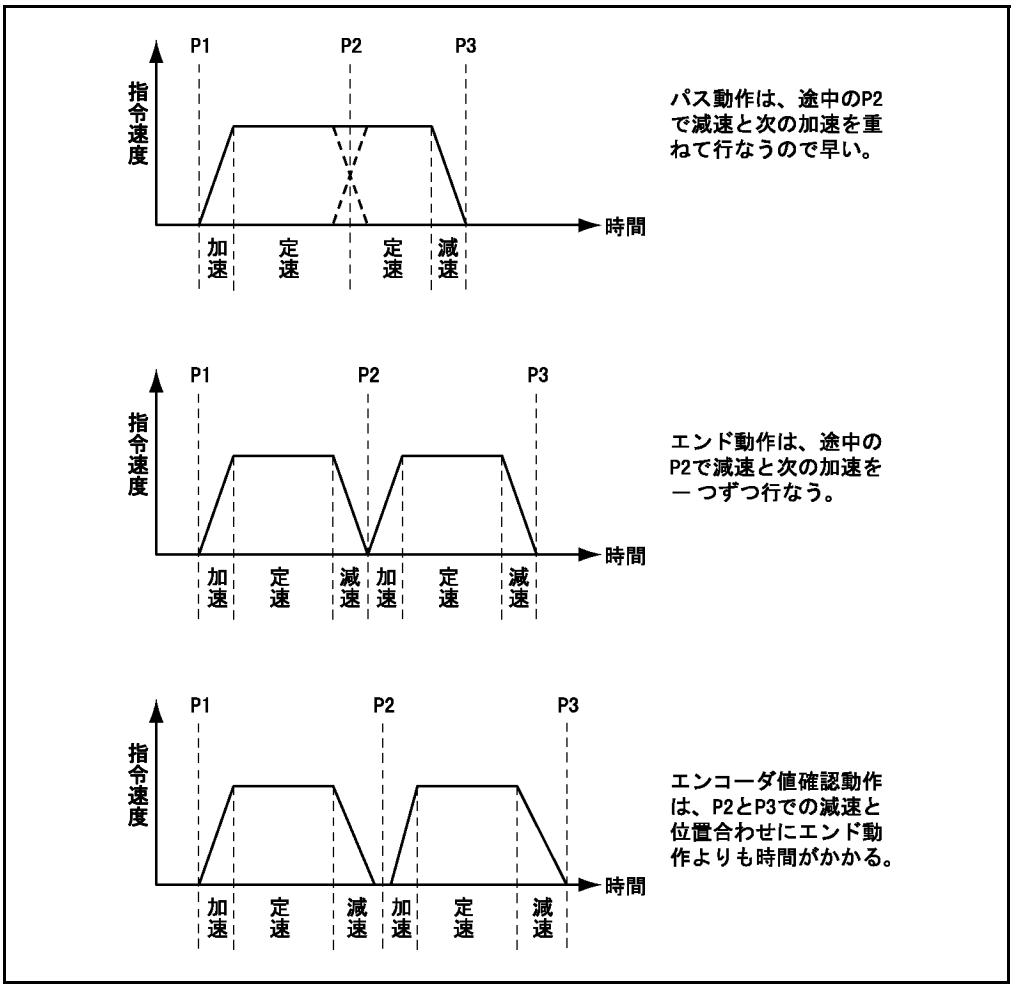



図3-9　パス動作、エンド動作、エンコーダ値確認動作の実行時間

## 3.2.6　パス動作しない場合

次の場合には、パス動作を指定しても、エンド動作で動きます。

### 3.2.6.1　メインプログラムの最後にパス動作コマンドがある場合

メインプログラムの最後に、実行されるパス動作コマンドは、エンド動作コマンドとして実行されます。図3-10のような場合、最後のP3にはエンド動作で到達します。

```
PROGRAM PRO10
    TAKEARM
    MOVE L, @P P1
    MOVE L, @P P2
    MOVE L, @P P3
END
```

図3-10　メインプログラムの最後にパス動作コマンドがあるプログラム例

### 3.2.6.2　パス動作コマンド直後に GIVEARM がある場合

GIVEARM（p.14-27参照）を実行すると、ロボットの動作が完全に停止するのを待ちます。したがって、パス動作コマンドの直後にGIVEARMを実行すると、パス動作しません。

## 3.2.7　パス動作の効果が小さくなる場合

### 3.2.7.1　パス動作後に非動作命令がある場合

パス動作コマンドと次の動作コマンドの間に、非動作コマンドがあると、パス動作の実行時間短縮効果が小さくなります。非動作コマンドとは、ロボット本体の動作を伴わないコマンドのことをいいます。
図3-12に、パス動作コマンドと次の動作コマンドとの間に、非動作コマンドがある例を示します。この例のような場合、図3-13に示すように、パス動作コマンドの減速時間中に非動作コマンドを実行するため、パス動作の実行時間短縮効果が小さくなります。

```
PROGRAM PRO13
    MOVE L, @P P2       'パス動作コマンド
    SET IO1             '非動作コマンド
    RESET IO2           '非動作コマンド
      :
    MOVE L, @0 P3       'エンド動作コマンド
END
```

図3-12　パス動作コマンドと次の動作コマンドの間に、
非動作コマンドがある例

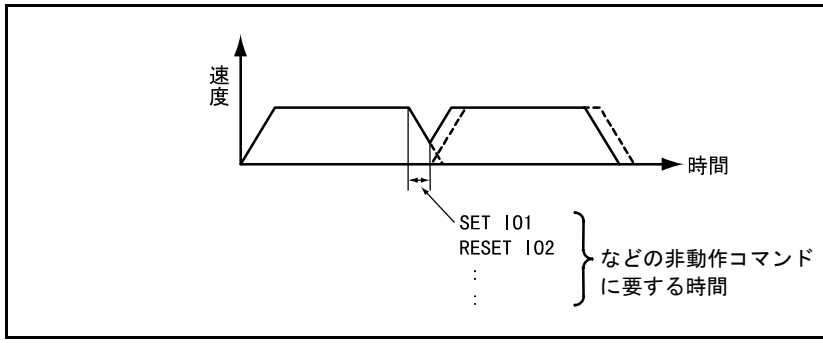



図3-13　パス動作の実行時間短縮効果が小さくなる例

### 3.2.7.2　パス後の軌道が短い場合

パス後の軌道が短く速度パターンが三角パターンになる場合、パス開始位置は、パス後の軌道の加速終了時にパス動作が終了するように遅れます。したがって、図3-14のようにパス前軌道の減速度が小さい場合、特にパス効果が小さくなります。

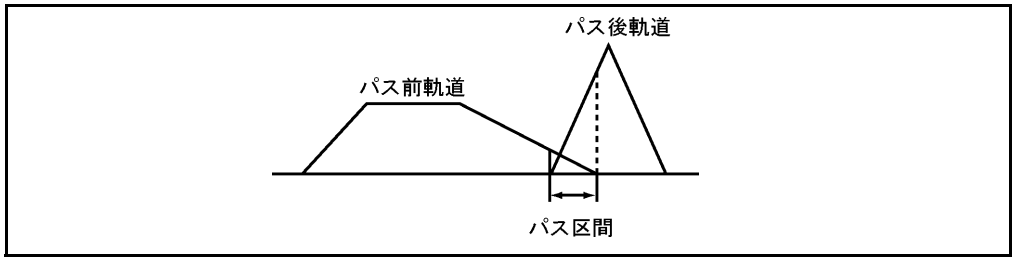



図 3-14　パス後軌道が短く三角パターンになる場合

## 3.2.8　加速度がパス動作の経路に影響する場合

デンソーシリーズロボットは、速度を設定すると自動的に、速度の二乗を100分の1した加速度と減速度が設定されます。
SPEEDコマンドを使用した場合も、同様に加速度と減速度が設定されます。
ロボットが自動的に設定した加速度を使用し、パス動作を行なう場合、動作経路は常に一定です。図3-15に、速度を60%と80%に設定した例を示します。

⚠注意：加速度を設定するときは、パス動作の経路変化に伴う衝突などの危険がないことを確認してください。

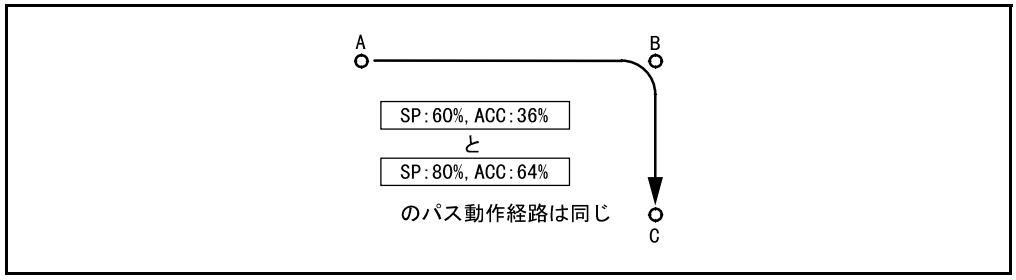



図3-15　加速度を自動設定した例

任意に加速度を設定すると、動作経路が変化します。図3-16では、速度60%の設定に対し、加速度と減速度の両方を100%にした場合と、10%にした場合を示します。

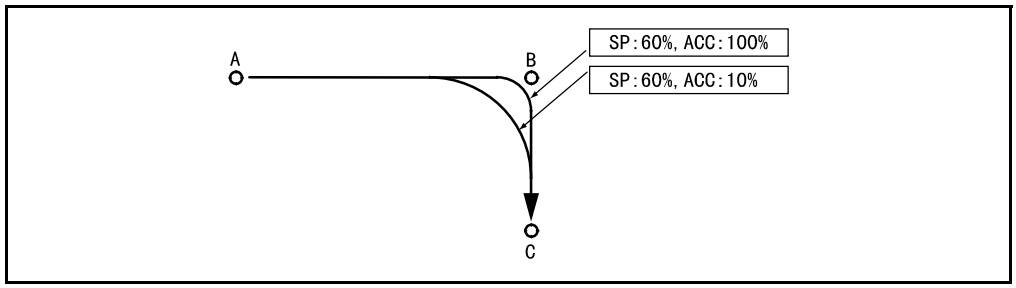



図3-16　加速度を任意設定した例

⚠注意：パス動作の減速途中で、瞬時停止を行ない、次に動作を再開すると、次のステップの指令値に動きます。速度と加速度、減速度が低いときは、注意してください。

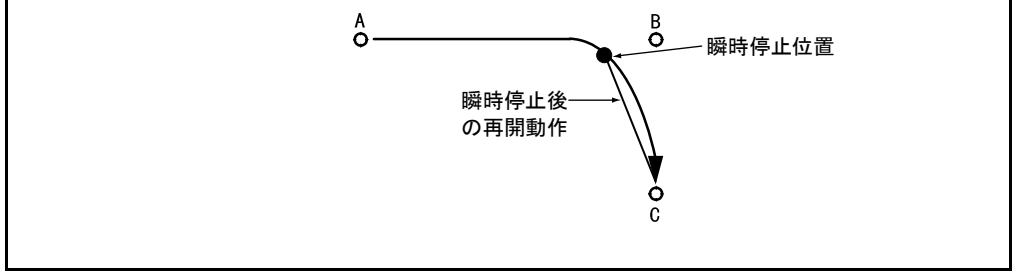



図3-17　瞬時停止後の再開動作

## 3.2.9　パス開始変位

パス開始変位を「@P」で指定すると、それまでの動作の減速開始と同時に次の動作を開始します。したがって、図 3-18 に示すように、パス開始位置とパス通過点 B との距離 L は、速度・加速度により変化します。

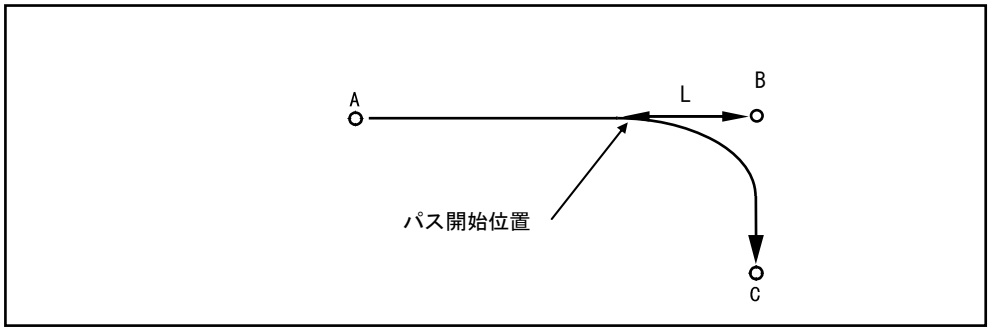



図 3-18　パス開始位置

パス開始変位を「@数値」で指定すると、数値=L（mm）の位置からパス動作を開始します。

「@数値」で指定する値が、減速移動距離を超える場合は、図3-19に示すように、減速開始位置が修正されます。

したがって、「＠数値」で指定した場合、減速度が低下し、移動時間が長くなる場合があります。

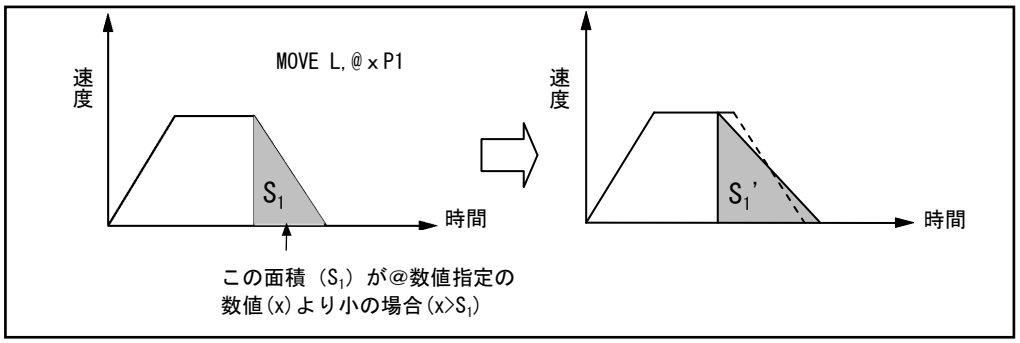



図 3-19　減速開始位置

また、「@数値」で指定する値が、移動距離の 1/2 を超える場合は、図 3-20 に示すように、減速開始位置は移動距離の 1/2 に固定されます。

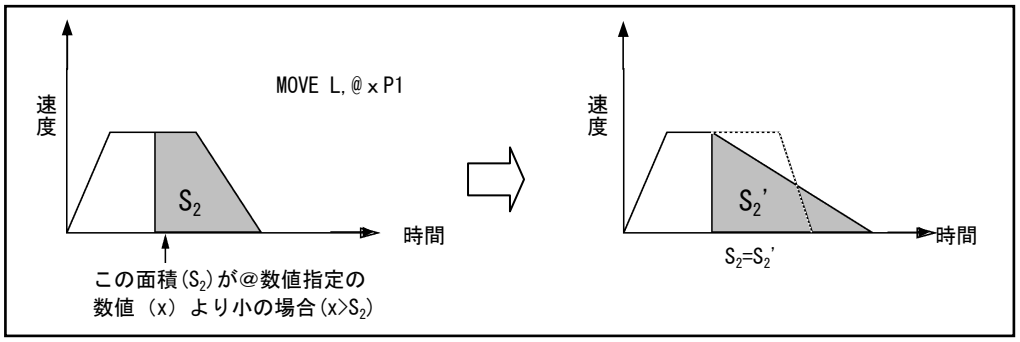



図 3-20　「@数値」が移動距離の 1/2 を超える場合

したがって、X>S2 の場合、X を変えてもパス動作は変化しません。
また、X>S2 の場合、「パス開始変位量を再設定して下さい」のワーニングがティーチングペンダントに表示されます。

減速開始位置が変化する場合、次の軌道の加速時間も修正されます。図 3-21 に示すように、@P 指定の場合に比べ、動作時間が長くなる場合があります。

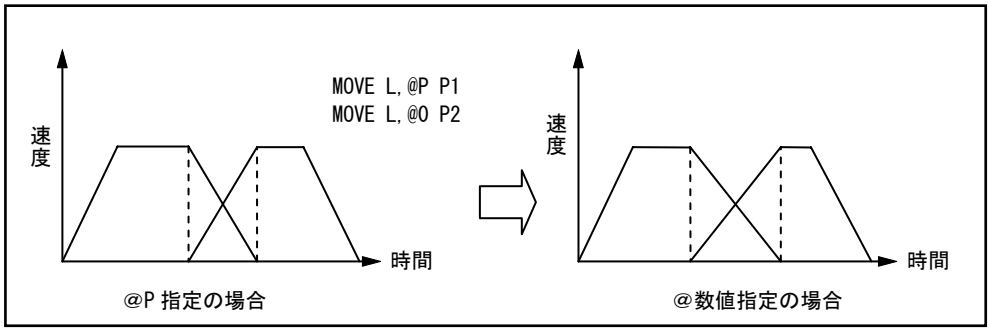


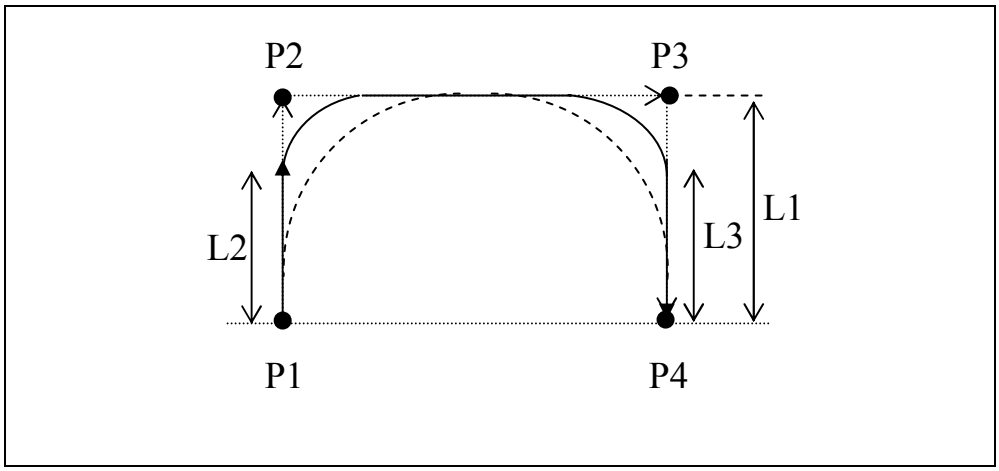

図 3-21　減速開始位置が変化する場合の動作時間

注意：次の場合、「@数値」で指定しても、数値=L（㎜）の位置より短い位置からパス動作を開始します。
(1)パス後軌道が短く、速度パターンが三角パターンになる場合
「3.2.7　パス動作の効果が小さくなる場合」と同様に、パス動作開始位置が変動します。特に、低速動作から高速動作になる際にパス動作させると、パス動作開始位置が変動しやすくなります。パス動作時は、パス動作前後で内部速度、加速度をなるべく一定にしてご使用ください。

補足説明
パス開始変位の数値指定では変数を指定することができます(Ver.3.0以降)。
I型変数で指定します。変数に-1を指定すると"@P"と同じパス動作をします。
-2を指定するとエンコーダ値確認動作"@E"となります。
その他の負の値を指定した場合はエラーとなります。

## 3.2.10　アーチモーション機能［Ver.1.9 以降、4 軸ロボットのみ］

アーチモーション機能を使うと、効率のよいPick&Place（Ｐ＆Ｐ)動作をさせることができます。




上図のように現在位置（Ｐ１）から目標位置（Ｐ４）へ動作するＰ＆Ｐ動作は、通常、経由するＰ２，Ｐ３点を指定して実現します。Ｐ２、Ｐ３付近でパス動作させることで動作時間を短縮させることができます。
アーチモーションでは、パス動作と異なりＺ方向の動作中のどの位置からでも次動作に移る事ができますので、さらに効率のよい動作が可能です。

また、Ｐ４に対するＺ軸の動作量（Ｌ１）、パスの開始位置（Ｐ１からの距離Ｌ２）、パスの完了位置（Ｐ４からの距離Ｌ３）を指定するだけですので、プログラム作成も簡単です。

ただし、アーチモーションでは位置制御を行なっていません。動作速度の変動により動作経路が変わりますので、動作の際は周辺設備との干渉に注意してください。なお、アーチモーション中の動作は全てＰＴＰ補間となります。

アーチモーション機能に使用するライブラリ：
ArchMove, SetArchParam

# 3.3　補間制御

ロボットアームの先端が移動するとき、その経路は一通りではありません。各軸の動きが相まって、いろいろな経路をつくることができます。直線や円弧になるように制御することもできます。以下に、動作経路の種類に応じた制御方法について説明します。
次のコマンドで、補間方法を指定します。

補間方法を指定するコマンド：
　APPROACH、DEPART、DRAW、MOVE

## 3.3.1　PTP 制御

PTP（Point to Point）とは、点から点への移動を意味します。移動する経路はロボットの姿勢に依存し、直線運動をするとは限りません。
図3-22にPTP制御による動作例を示します。
動作制御コマンドで補間方法を指定するときに、「P」の指定をするとPTP動作を行ないます。
PTP動作目標位置にP型、T型変数を指定した場合、ロボット形態を指定すると、指定したロボット形態となるよう移動します。ロボット形態を指定しない場合は、現在のロボット形態と同一の形態になります。

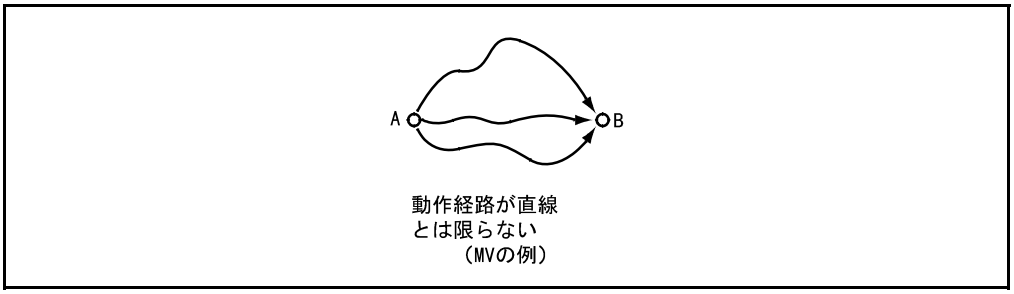



図3-22　PTP制御動作

## 3.3.2 CP 制御

CP制御は、動作目標位置に達する経路を、直線になるように補間制御します。
図3-23にCP制御による動作例を示します。
動作制御コマンドで補間方法を指定するときに、「L」の指定をするとCP動作を行ないます。

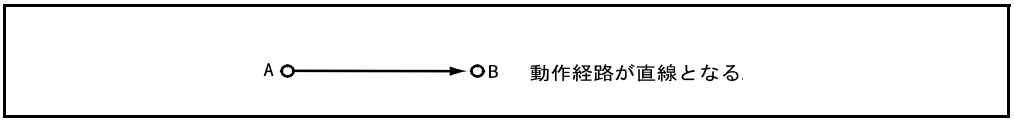



図3-23　CP制御動作

・CP制御時は、原則として、現在のロボット形態と異なる形態へは移動できません。異なる形態を指定した場合は、「607F　ロボット形態不一致」エラーが発生します。ただし、移動可能な場合は、エラーは発生しません。
・CP制御時のロボット形態は、現在の姿勢に近い形態が選択されます。したがって、P型、T型変数でロボット形態を指定しても、指定した形態になるとは限りません。指定した形態と異なる場合は、「601C　形態を変更して下さい」のワーニングが発生します。
・プログラムでの最初の動作命令をCP制御で行なうと、ロボットの位置によって動作できない場合があります。プログラムの最初の動作命令にはPTP制御を使用することをおすすめします。

## 3.3.3 円弧補間制御

円弧補間制御は、動作目標位置に達する経路を、円弧を描くように補間制御します。
図3-24に円弧補間制御による動作例を示します。
動作制御コマンドで補間方法を指定するときに、「C」の指定をすると円弧補間動作を行ないます。

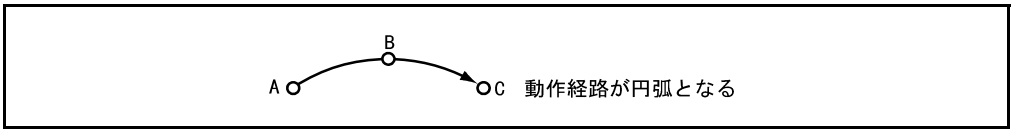


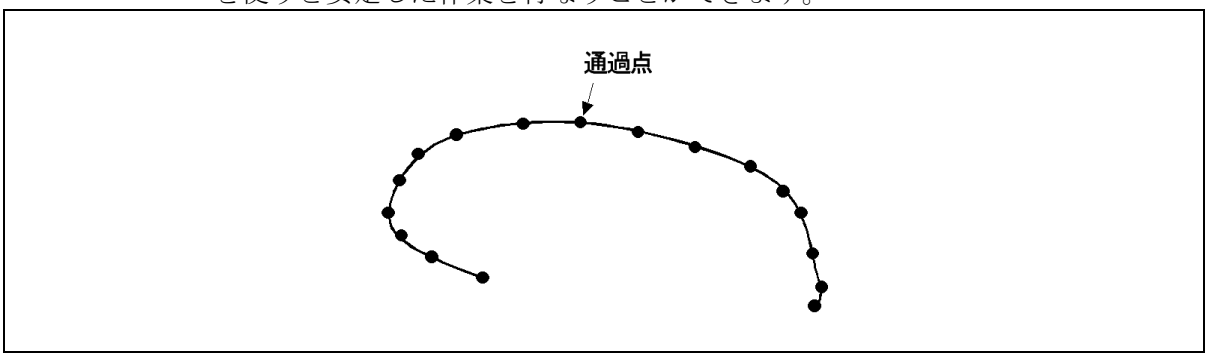

図3-24　円弧補間制御動作

・円弧補間制御時は、CP制御時と同様、原則として、現在のロボット形態と異なる形態へは移動できません。異なる形態を指定した場合は、「607F　ロボット形態不一致」エラーが発生します。ただし、移動可能な場合は、エラーは発生しません。
・円弧補間制御時のロボット形態は、現在の姿勢に近い形態が選択されます。したがって、P型、T型変数でロボット形態を指定しても、指定した形態になるとは限りません。指定した形態と異なる場合は、「601C　形態を変更して下さい」のワーニングが発生します。
・プログラムでの最初の動作命令を円弧補間制御で行なうと、ロボットの位置によって動作できない場合があります。プログラムの最初の動作命令にはPTP制御を使用することをおすすめします。

## 3.3.4　自由曲線補間　[Ver2.3 以降]

### 3.3.4.1　概要

下図に示すように、指定された点列（通過点）を通過する自由曲線上を一定速度で動作します。動作開始時は加速し、動作終了時は減速します。
シール塗布、パッキン貼り、バリ取りなどの作業を行なう場合、自由曲線補間を使うと安定した作業を行なうことができます。

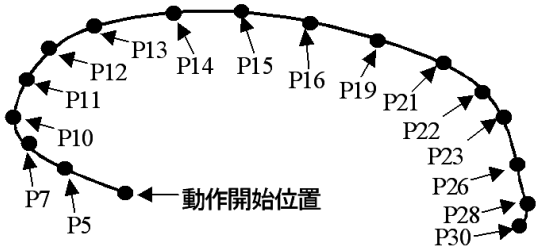



自由曲線補間

### 3.3.4.2　作業ステップ

### (1) 通過点の教示

通過点を教示します。　通過点は、P型、J型のどちらかの変数に教示してください。T型変数に教示した位置データは、「(2)　通過点登録」時にT2P命令によりP型に変換する必要があります。
通過点の番号とP型、J型変数の番号の対応をわかり易くするために、下図に示す対応表を作成してください。




| 通過点 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 変数 | P5 | P7 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P21 | P22 | P23 | P26 | P28 | P30 |

通過点の対応表例

## (2) 通過点登録

自由曲線軌道番号を指定し通過点を登録します。通過点は、自由曲線軌道番号毎に専用命令（SETSPLINEPOINT）を用いて登録します。
例1のように通過の順番に従って通過点を登録するプログラムを作成してください。詳細は、SETSPLINEPOINT命令、CLRSPLINEPOINT命令を参照ください。自由曲線の軌道番号は１としています。

```
PROGRAM PASSPOINT1
CLRSPLINEPOINT 1
SETSPLINEPOINT 1, P5
SETSPLINEPOINT 1, P7
SETSPLINEPOINT 1, P10
        ⋮
SETSPLINEPOINT 1, P28
SETSPLINEPOINT 1, P30
END
```

**例1　通過点登録プログラム**

## (3) 自由曲線補間の実行

軌道番号を指定し自由曲線補間を実行します。自由曲線の動作命令は、MOVE S<自由曲線の軌道番号>になります。詳細は、MOVE命令を参照ください。通過点は自由曲線補間命令の前に登録してください。動作直前に登録する場合は、例2になります。

```
PROGRAM PRO1
TakeArm
    ⋮
CALL PASSPOINT1 '通過点を登録する。
MOVE S, 1 '軌道番号1の自由曲線動作を実行
    ⋮
END
```

**例2　自由曲線動作の直前で通過を登録する場合**

TakeArm命令実行時、SETSPLINEPOINTで登録した通過点は、すべて消去されます。通過点を初期化プログラム等の別プログラムで登録する場合、自由曲線動作プログラムのTakeArm命令で登録した通過点を消去しないように、xdSPLClrTakeArm(0)を実行してください。例3は、初期化プログラム中で通過点を登録するプログラム例です。詳細は、ライブラリxdSPLClrTakeArmを参照ください。

```
PROGRAM INITIAL
    ⋮
CALL PASSPOINT1 '通過点を登録する。
CALL xdSPLClrTakeArm(0) 'TakeArm時の通過点クリアを無効化
    ⋮
END
```

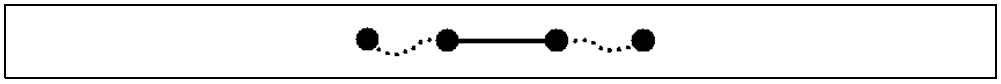
例3　初期化プログラムで通過点を登録する場合

### (4) 通過点の調整

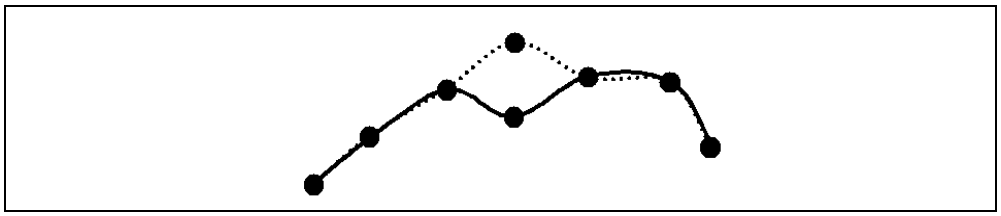
ティーチチェックモードで自由曲線の経路を確認ください。自由曲線動作もステップ戻し機能を使用できます。瞬時停止、ステップ戻し、ステップ送りにて経路を確認してください。経路が理想曲線とずれる場合は、通過点間隔を短くするために通過点を挿入してください。
速度を上げた場合、軌道が曲線の内側にずれる場合があります。その場合、通過点を理想曲線の外側に教示し、調整してください。

## 3.3.4.3 自由曲線の特徴

(1) 通過点が1点の場合、直線動作になります。
(2) 下図のように通過点４点を直線上に教示した場合、中間の2点間は直線になります。

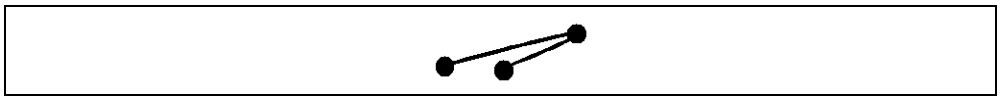


(3) 下図のように通過点を変更した場合、変更した点の前後2点間の経路が変わります。



(4) 自由曲線の経路が理想曲線とずれる場合は、通過点の間隔を短くしてください。

(5) 下図に示すように同一点を往復したり、鋭角に曲がる通過点を指定する場合、通過点で速度が低下し、エラー「676* 指令加速度制限オーバ」、「60D0 最適可搬質量実行異常」が発生する場合があります。この場合、速度を下げて調整ください。

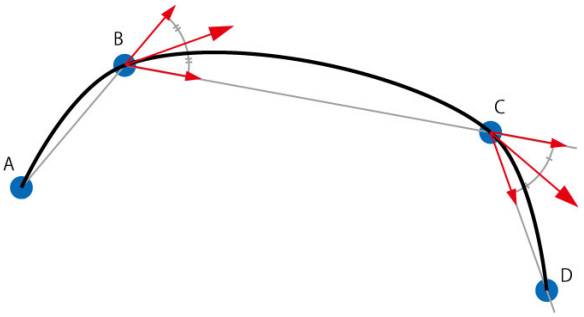


(6)動作に同期してI/O入出力等の非動作命令を実行する場合、下例のように通過点待ち命令（ライブラリ）を使用してください。図2にてP11（4番目の通過点）を通過した際にI/O出力する場合、以下のプログラムになります。詳細は、ライブラリxdWAITSPLINEを参照してください。

```
PROGRAM  PRO1
TakeArm
      ⋮
CALL PASSPOINT1  ‘通過点を登録する。
MOVE S, 1, NEXT  ‘軌道番号1の自由曲線動作を実行
CALL xdWAITSPLINE(4,1) ‘4番目の通過点を通過待ち
SET IO[240]
      ⋮
END
```

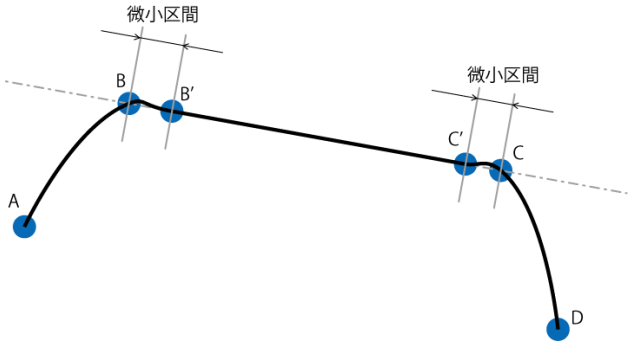
**■自由曲線補間の特性（軌跡生成概略）**

自由曲線補間は、区間前2点を結んだ方向と、次点を結んだ方向の中間の方向に傾きを持った曲線でスタートし、次点を結んだ方向と区間後２点を結んだ方向の中間の方向に傾きを持った曲線で終了します。




**■直線区間とする場合**

４点を直線上に配置する必要があります。
直線開始位置と直線区間終了位置の直線区間（右図青点の小区間の間隔）は、微小としてティーチングを行ってください。




**〈ポイント〉**

４点を直線上にのせることで、中間の２点間（B’-C’点間）は完全に直線になりますが、B-B’間、C’-C間は、前後のポイントの影響を受けるため、直線からズレます。
前後のポイントの影響は、区間距離を短くすることで低減できるため、微小区間にすれば、ズレを低減できます。

### 3.3.4.4 自由曲線補間を使用するときの注意点

(1) 通過点は、軌道番号毎に最大200点まで登録可能です。200点を超えて登録した場合は、エラー「685A 自由曲線登録データ数オーバ」が発生します。

(2) 軌道番号は、最大20まで指定可能です。20を超えて指定した場合は、エラー「685B 自由曲線番号指定異常」が発生します。

(3) 自由曲線動作後に通過点を変更した場合、通過点変更前の自由曲線のステップ戻し動作はできません。エラー「737D　これ以上戻れません」が発生します。

(4) 通過点間の移動距離が短く姿勢変化が大きい場合、エラー「608*　指令速度制限オーバ」、「60D0　最適可搬質量実行異常」が発生する場合があります。この場合、速度を落としてください。

(5) コンベアトラッキング中に自由曲線動作を実行できません。

(6) 自由曲線使用時は、復電機能を使用できません。

(7) 1つの自由曲線上をパス動作で繰返し動作させたい場合は、その自由曲線を2つの軌道番号に登録し、動作毎に軌道番号を切り替えてください。同じ軌道番号の自由曲線をパス動作させるとエラーが発生する場合があります。

## 3.3.5 高軌跡制御機能 [Ver.2.61 以降]

### 3.3.5.1 概要

高軌跡制御機能とは、ロボットの動作軌跡の精度を向上させることができる機能です。特に高速時の円弧動作や自由曲線補間動作においては動作軌跡の精度が向上します。シール塗布などの作業を行う場合に、高軌跡制御機能を使用すると安定した作業を行うことができます。

### 3.3.5.2 高軌跡制御機能使用方法

ライブラリ機能を使用し、高軌跡動作ライブラリ（SetHighPathAccuracy、ResetHighPathAccuracy）を実行することで機能を有効、無効に設定します。なお、ライブラリの詳細については、PACライブラリ項を参照ください。

```
PROGRAM  PRO1                  プログラム説明
TAKEARM                        ’ アームグループの制御権を取得します。
MOVE  P, @E P1                 ’ P1 へ PTP 動作で移動します。
CALL  SetHighPathAccuracy      ’ 高軌跡制御機能を有効にします。
MOVE S, 5                      ’ 軌跡番号 5 を自由曲線動作で高軌跡移動します。
CALL  ResetHighPathAccuracy    ’ 高軌跡制御機能を無効にします。
MOVE  P, @0 P5                 ’ P4 から P5 へ PTP 動作で移動します。
END                            ’ プログラム終了します。
```

高軌跡動作のプログラム例

### 3.3.5.3 高軌跡制御機能を使用する時の注意点

(1) 動作軌跡を向上させたい動作のみ､高軌跡制御機能を有効にしてください。
(2) 動作条件､負荷条件によっては振動しながらの動作になる場合があります。その場合、速度や加速度を下げたり、機能を無効にしたりして下さい。
(3) コントローラ電源オフまたはモーターオフで高軌跡制御機能は無効になります。
(4) 付加軸オプションの軸（7, 8軸）には対応していません。
(5) 力制限機能有効時には高軌跡制御機能を有効にすることはできません。エラーが発生します。
(6) 動作中（パス動作中など）に高軌跡制御機能を有効にすることはできません。
(7) 負荷設定(先端負荷質量、先端負荷重心位置など)を正確に設定してください。正確に設定されてない場合、精度が向上しない原因となります。

## 3.3.6　特異点回避機能［Ver2.61 以降、6 軸専用］

### 3.3.6.1　機能の概要

特異点とはベース座標で表される位置について、ロボットのとる姿勢が無数に存在してしまう点のことを言います。腕、ひじ、手首の形態において、各々の境界に特異点は存在します。手首の形態の場合では、5軸が0度の時、4軸と6軸は平行になり、姿勢が一意に決まらない状態になります。(操作ガイド 4.座標系と形態　参照)

CP動作で特異点やその近くを通過する際、形態を維持するために特定の軸が瞬時に急動作しようとします。そのため速度オーバやソフトリミットオーバのエラーになってしまいます。

この特異点によるエラーを回避する機能を「特異点回避機能」と言います。ただし、もっとも頻繁に起こる手首の特異点(5軸が0度の時)のみを回避します。[Ver.3.2]より前のバージョンでは、パス動作中に特異点を回避できない場合がありました。[Ver.3.2以降]ではパス動作中でも特異点を回避することができます。本機能では、本来動作開始時の形態を保持して動作するところを動作途中で手首形態を変更して動作するため、この間のロボットの軌道は本来の軌道からずれて動作いたします。動作条件によってはロボットの姿勢が大きく変化する事もあります

### 3.3.6.2　機能の使用方法

特異点回避機能を有効にするには下記の2つの方法があります。

また、[Ver.3.2以降]では「特異点回避拡張設定」を有効にすることにより、パス動作中も特異点を回避することが出来ます。ただし、この機能を有効にするには、特異点回避機能有効設定を有効にしておく必要があります。この機能をはティーチングペンダントでのみ設定可能です。

プログラムコマンドによる設定

プログラム内で「SetSingularAvoid」を記述することにより特異点回避機能の有効/無効を設定します(プログラミングマニュアルⅡ SetSingularAvoid 参照)。コントローラ電源オフやモータオフでリセットされ、「無効」になります。

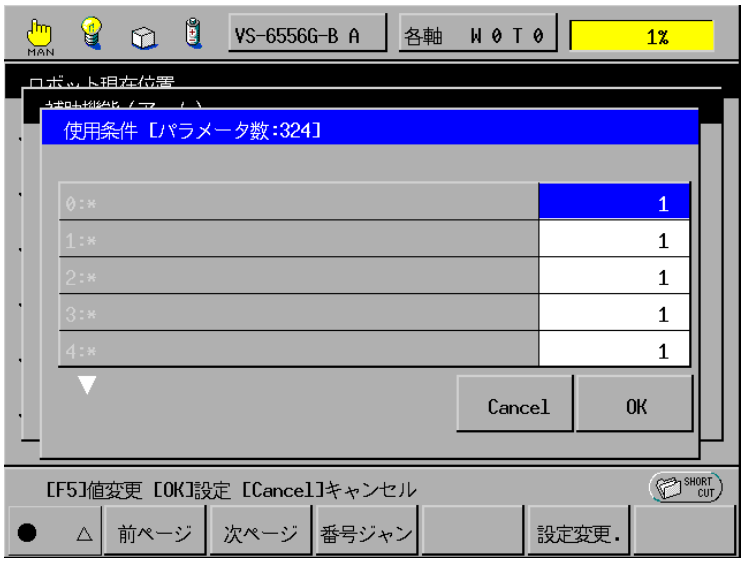
ティーチングペンダントによる設定

全てのCP動作に対して特異点回避機能及び、特異点回避拡張設定 (Ver.3.2以降) を有効/無効にできます。設定はコントローラ電源オフ後も保持されます。

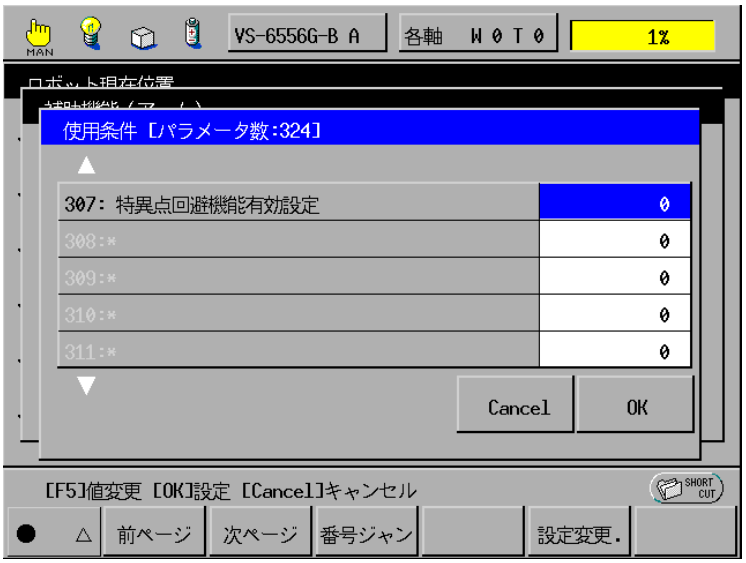
### 3.3.6.3　ティーチングペンダントによる設定方法

操作経路：基本画面－[F2 アーム]－[F6 補助機能] －[F7 使用条件]

(1) ティーチングペンダントを上記経路の通りに操作すると[使用条件]画面が現れます。



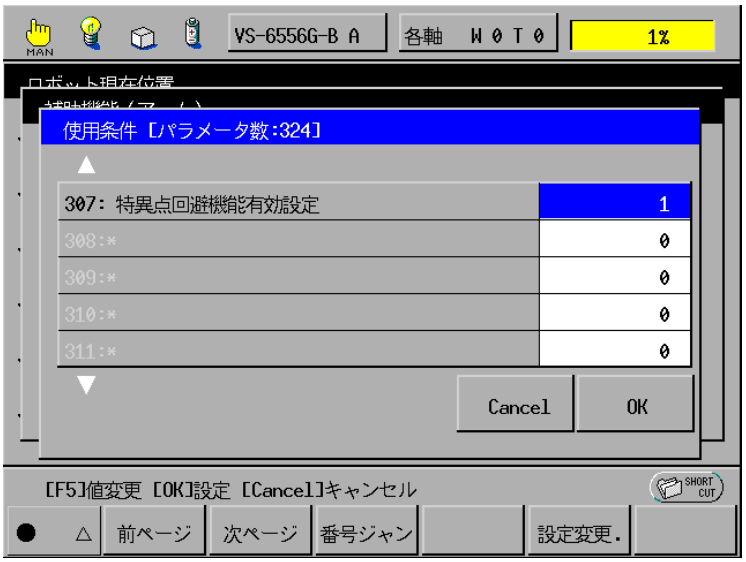

(2) [F3 番号ジャン]で307へジャンプすると[特異点回避機能有効設定]が表示されます。



(3) [F5設定変更. ]を押し、テンキー画面で設定を変更し[OK]を押します。
有効：1　　無効：0




(4) 使用条件のパラメータが変わったことを確認して[OK]を押し、終了します。




⚠注意：ペンダントから機能を有効にした場合は全ての CP 動作が特異点回避の対象となります。本機能は動作条件によっては本来の軌道からはずれて動作し、ロボットの姿勢が大きく変化する場合があります。ライブラリによる設定を用いて個々の動作が周辺設備との干渉が生じない事を確認の上で使用して下さい。

### 3.3.6.4　特異点近傍警告

実際に動作中に特異点回避機能が働いた場合、以下の操作で特異点近傍を通過すると警告音とワーニングメッセージを発生します。
・手動モード、ティーチチェックモードでの変数移動（CP動作のみ）。
・ティーチチェックモードでのプログラム動作（CP動作のみ）
自動モードでは警告は行われません。

### 3.3.6.5　特異点回避機能使用時の注意点

(1) 本機能は手動モードでの各軸動作、XY動作及びツール動作では使用できません。プログラムでの動作命令及び変数移動に対して使用できますが、PTP動作では無効となります。
(2) 本機能は手首形態の特異点（5軸0度）の近傍を通過する際に回避動作を行います。回避動作中は本来動作すべき軌道と異なり、ロボットの姿勢が大きく変化する場合があります。特異点の性質上、手先位置・姿勢をずらさずに制御する事はできませんので、軌道からのずれを許容できない動作では本機能は使用できません。また本機能使用時、軌道からのずれによる周辺設備との干渉に注意して下さい。
(3) 本機能は、腕・ひじの形態に対する特異点には適用されません。手首形態の特異点（5軸0度）に対してのみ有効です。
(4) 4軸及び5軸ロボットでは本機能は無効となります。
(5) 5軸0度となる位置へはCP補間での移動はできません。5軸0度近傍位置へのCP補間での移動は、特異点を通過する場合のみ回避動作を行います。
(6) 動作条件（ツールの長さ、動作速度、動作方向や動作位置・姿勢）によっては、動作中にエラーが発生して動作できない場合があります。この時速度を下げる、あるいは動作位置や姿勢を変える事によって動作できる場合もあります。

# 3.4　動作命令の後に出力コマンドがある場合

動作命令実行時に、ロボットコントローラの指令する位置に対して、ロボットの現在位置には遅れがあります。
エンド動作の場合、ロボットコントローラでは、指令位置が動作命令の目標位置に達すると、次のコマンドの実行に移ります。そのため、動作コマンドに続いて出力コマンドがあると、ロボットが目標位置に到達する前に、出力コマンドが実行されることがあります。
目標位置に到達してから、次のコマンドを実行するためには、次のような方法があります。

●DELAYコマンドによる待ち時間を設ける。(図3-25にプログラム例)
●エンド動作ではなく、エンコーダ値確認動作にする。(図3-26にプログラム例)

**注意：エンコーダ値確認動作では、負荷が重い場合には、目標位置にエンコーダ値が完全に一致できない場合があります。**

```
PROGRAM PRO13
      :
      :
   MOVE L, @0 P3      'エンド動作コマンド
   DELAY 500          '0.5秒待つ
   SET IO1            'I/O1をON
   RESET IO2          'I/O2をOFF
      :
      :
END
```

**図3-25　DELAYコマンドによる待ち時間を設けるプログラム例**

```
PROGRAM PRO13
      :
      :
   MOVE L, @E P3      'エンコーダ値確認動作コマンド
   SET IO1            'I/O1をON
   RESET IO2          'I/O2をOFF
      :
      :
END
```

**図3-26　エンコーダ値確認動作を利用するプログラム例**

# 3.5　力制限機能

## 3.5.1　概要

力制限機能は、ロボットにソフトウエアにて柔らさを設定する機能です。
ロボットのハンドリング時の位置ずれを吸収したり、ロボットやワークに過大な力が加わるのを防止する事が可能です。本機能は、各関節毎に柔らかさを設定する機能（電流制限機能）とロボット先端の座標系の各要素毎に柔らかさを設定する機能（先端力制限機能）があります。

**注意：本機能は PAC ライブラリを使用した事例で説明していますが、Ver.1.9 以降は「12.10 特殊制御」コマンドも使用可能です。**

## 3.5.2　各軸電流制限機能 [V1.2 以降]

ロボットの各軸毎に柔らかさを設定する機能です。各軸モータのトルク(電流値)を制限する事で柔らかさを実現します。主にロボットやワークに過大な力が加わるのを防止したり、過負荷や過電流エラーにて停止するのを防止する為に使用します。

### 3.5.2.1　電流制限機能使用方法

ライブラリ機能を使用し、電流制限ライブラリ(SetCurLmt, ResetCurLmt)を実行する事で機能を有効、無効に設定します。ただし、以下の設定をしてください。なお､ライブラリの詳細については､ＰＡＣライブラリ項を参照ください。

(1) 先端負荷設定（V*-Dシリーズ必須）

先端負荷質量と負荷重心位置を正確に設定してください。使用条件中の先端負荷質量[g]、負荷重心位置X6[mm]、負荷重心位置Y6[mm]、負荷重心位置Z6[mm]に使用するハンドやワークに対応した値を入力してください。ただし、ワークチャック、アンチャック等で質量変化する場合は、aspACLDを使用して設定を変更してください。

詳細は、「4.7「使用条件」における最適可搬質量設定機能」を参照ください。

**注意： 正確に設定されない場合､電流制限設定時にロボットが重力方向へ落下する場合があります。**

(2) 重力補償設定（V*-Dシリーズ必須）

力制限時は、必ず重力補償を有効にしてください。有効でない場合は、エラー「665a 電流制限設定できません」が発生します。重力補償を有効化するには、ペンダントで使用条件の重力補償有効無効設定を１に設定してください。設定方法は、以下のとおりです。

①ペンダントの基本画面で［F2 アーム］→［F6 補助機能］→［F7 使用条件]を押してください。使用条件パラメータ設定画面が表示されます。

②[F3 番号ジャン]を押し、24番を指定してください。重力補償有効無効設定行が表示されます。

③重力補償有効無効設定に1を入力し、OKを押してください。
重力補償有効無効設定を１に設定した場合、重力補償は、キャリブレーション終了状態で常に有効になります。
また、SetGravity、ResetGravityにて重力補償を有効、無効に設定できますが、有効、無効切り替え時にロボットが微小動作する場合があります。
ペンダントで使用条件の重力補償有効無効設定を1に設定し、初期化プログラム中でSetGravityを一度CALLして頂くことを推奨します。

(3) 重力補償補正（V*-Dシリーズのみ必要時）
先端負荷質量設定値と実際の負荷質量のずれにより、重力補償値による重力バランスにずれが生じる場合があります。この場合、重力を受ける軸の電流制限設定値を小さく設定するとロボットが重力方向に落下する場合があります。この場合もロボットの落下を防止する為、ロボット停止状態のトルクと重力補償トルクとを比較し、誤差分を補正する機能が重力補償補正機能です。重力を受ける軸の電流制限値を30%以下に設定する場合は、必ず重力補償補正を実施してください。
ロボット停止状態でSetGrvOffsetを実行すると補正値を計算します。
ただし、重力補償補正値は、**SetGrvOffsetを実行した姿勢の補正値です。補正姿勢から大きく姿勢が変化する場合は、再度SetGrvOffsetを実行してください。**
また、**ロボットが接触等で外力を受けた状態でSetGrvOffsetを実行すると、補正値に誤差が生じます。外力（重力を除く）を受けない状態でSetGrvOffsetを実行してください。**

(4) 偏差許容値設定（必要時）
電流制限中に外力を受けると関節が力方向に回転します。よって、関節角度の偏差が増加し、エラー「612* J*偏差過大」が発生する場合があります。この場合、SetEralwにて偏差許容値を設定してください。**偏差チェック機能は安全機能です。必要以上に範囲を広げないでください。**

(5) ライブラリ実行手順
■ 電流制限有効設定時
①ペンダントにて使用条件中の重力補償有効無効設定を１に設定する。(V*-Dシリーズのみ)
② 初期化プログラムにて下記追加　(V*-Dシリーズのみ)
```
CALL SetGravity
```
③ 電流制限有効化 ( )内は必要時
動作終了を待つ。
```
( CALL SetGrvOffset )     '重力補償補正値を計算する。
                                (V*-Dシリーズのみ)
CALL SetCurLmt(n1,m1)  'n1軸の電流値を定格のm1(%)に設定する。
( CALL SetEralw(n2,m2) )  'n2軸の偏差許容値をm2(度)に設定する。
```
■ 電流制限無効設定時
```
CALL ResetCurLmt(0)  '全軸電流制限をリセット,偏差許容値初期化
( CALL ResetGrvOffset )  '重力補償補正値を初期化
```

## 3.5.3　先端力制限機能［V1.4 以降］

**ロボット先端座標系の各要素毎に柔らかさを設定する機能（コンプライアンス機能）です。**先端の力制限値から各軸のモータトルクを制御する事で先端の柔らかさを実現します。座標系は、ベース座標、ツール座標、ワーク座標を選択できます。ロボットを、外力に対し特定方向に倣わせる場合や、突き当て動作による高さチェック等に使用します。

### 3.5.3.1　先端力制限機能使用可能設定方法

本機能は、拡張機能です。よって、**ペンダントにて力制限機能を使用可能に設定する必要があります。**拡張機能は、一度使用可能に設定すれば、コントローラ電源OFF後も使用可能状態を保ちますので、再設定は不要です。力制限機能使用可能設定方法を以下に示します。

(1) ペンダントの機能拡張の暗証番号入力画面を表示します。

操作経路：基本画面-[F6　設定]-[F7　オプション]-[F8　機能拡張]-[F5　機能追加]




(2) 暗証番号「6519」を入力し、OKを押します。システムメッセージが表示されるので確認し、OKを押すと追加された機能（コンプライアンス機能）が表示されます。



### 3.5.3.2　先端力制限機能使用方法

ライブラリ機能を使用し、力制限ライブラリ(SetCompControl, ResetCompControl)を実行する事で機能を有効、無効に設定します。ただし、以下の設定をしてください。なお、ライブラリの詳細については、ＰＡＣライブラリ項を参照ください。

(1) 先端負荷設定（必須）

3.5.2.1 電流制限機能使用方法を参照ください。

(2) 重力補償設定（必須）

3.5.2.1 電流制限機能使用方法を参照ください。

(3) 力制限パラメータの設定（必須）

力制限パラメータは、以下の 5 種類あります。

① 力制限座標系選択パラメータ

力制限座標系をベース座標、ツール座標、ワーク座標から選択します。力制限ライブラリ(SetFrcCood)にて設定します。

② 力制限割合

① にて設定された座標系の各成分(Ｘ方向、Ｙ方向、Ｚ方向、Ｘ周り、Ｙ周り、Ｚ周り)の力制限割合を設定します。力制限ライブラリ(SetFrcLimit)にて設定します。SetFrcLimit設定時、③ 柔らかさ割合、④ 粘性割合も同一割合となります。

③ 柔らかさ割合　(必要時)

① にて設定された座標系の各成分(Ｘ方向、Ｙ方向、Ｚ方向、Ｘ周り、Ｙ周り、Ｚ周り)の柔らかさを設定します。力制限ライブラリ(SetCompRate)にて設定します。SetCompRate設定時、④ 粘性割合も同一割合となります。

④ 粘性割合　(必要時)

① にて設定された座標系の各成分(Ｘ方向、Ｙ方向、Ｚ方向、Ｘ周り、Ｙ周り、Ｚ周り)の粘性割合を設定します。力制限ライブラリ(SetDampRate)にて設定します。先端力制限特殊機能である速度制御モード時以外、使用する必要はほとんどありません。ただし、SetFrcLimit,SetCompRateにて設定した柔らかさ割合より小さい値を設定しない様にお願いします。

⑤ 力制限時、位置、姿勢偏差許容値（必要時）

① にて設定された座標系の各成分(Ｘ方向、Ｙ方向、Ｚ方向、Ｘ周り、Ｙ周り、Ｚ周り)の位置、姿勢偏差許容値を設定します。力制限ライブラリ(SetCompEralw)にて設定します。初期値は、位置偏差が100(mm)、姿勢偏差が30(度)です。**偏差チェック機能は安全機能です。必要以上に範囲を広げないでください。**

(4) 重力補償補正（必要時）

SetCompControl時にロボット停止状態のトルクと重力補償トルクとを比較し、誤差分を補正します。しかし、ロボットが接触等で外力を受けた状態でSetCompControlを実行した場合、補正値に誤差が生じます。**外力（重力を除く）を受けない状態でSetCompControlを実行してください。外力を受けた状態で力制限有効にする場合は、SetGrvOffsetとSetCompFControlを使用し、以下の様にライブラリ実行してください。**

(SetCompControl = SetGrvOffset + SetCompFControl)
ロボット停止状態で外部との接触(干渉)なしの状態で
**CALL SetGrvOffset**
ロボットを外部接触(干渉)状態へ移動
**CALL CompFControl**

(1) ライブラリ実行手順

■力制限有効設定時

① ペンダントにて使用条件中の重力補償有効無効設定を１に設定する。

② 初期化プログラムにて下記追加

**CALL SetGravity**

③ 力制限パラメータ設定と有効化　( ) 内は必要時

**CALL SetFrcCoord(n)**　　' 力制限座標系をｎに設定する。

**CALL SetFrcLimit(fx,fy,fz,mx,my,mz)**'力制限(fx,fy,fz,mx,my,mz)[%]に設定する。

( CALL SetCompRate(cx,cy,cz,crx,cry,crz) )
　　'柔らかさ設定値を(cx,cy,cz,crx,cry,crz)[%]に設定する。

( CALL SetDumpRate(dx,dy,dz,drx,dry,drz) )
　　'粘性設定値を(dx,dy,dz,drx,dry,drz)[%]に設定する。

**CALL SetCompControl**　　'力制限を有効にする。

( CALL SetCompEralw(ex,ey,rz,erx,ery,erz) )'位置偏設定値を(ex,ey,ez) (mm)、姿勢偏差設定値を(erx,ery,erz) (度)とする。

( CALL SetEralw(n2,m2) )'n1軸の偏差許容値をm1(度)に設定する。

■力制限無効設定時　( ) 内は必要時

**CALL ResetCompControl**　　'力制限無効化設定

**CALL ResetCompEralw**　　'位置偏差設定値を初期化

( CALL ResetEralw(0) )　　'角度偏差設定値を初期化

### 3.5.3.3　先端力制限機能特殊ライブラリ使用方法

力制限特殊ライブラリを使用する事により、力制限機能をより有効にお使い頂けます。ただし、**本機能は、ロボットの制御データを直接変更するため、設定値によっては、ロボットが異常動作する場合があります。偏差許容値を小さく設定し、調整してください。**力制限特殊ライブラリにて以下のパラメータを設定できます。

(1) 力制限時のオフセット力

力制限時に摩擦等により力を加えてもロボットが倣い動作しない場合、**倣い方向に一定のオフセット力を加える事で倣い易くします。**力制限特殊機能ライブラリ(SetFrcAssis)にて、各方向のオフセット量を設定します。電源立ち上げ時の初期値は0です。

(2) 力制限時の電流制限設定値

力制限時は、ロボット先端の位置偏差に応じたトルクをモータに与える事で先端力を制限します。よって、従来の各軸毎の角度偏差に応じた制御はされず（電流制限値=0）ロボット動作が振動的になる場合があります。その場合、**力制限割合を調整する代わりに力制限時の電流制限設定値を調整する事で、よりなめらかな制限動作が可能になります。**特定方向の先端力制限を力制限ではなく、電流制限設定値で調整する場合、特定方向の力制限設定値を０にし、制限方向動作時に回転する軸の電流制限値を調整します。力制限特殊機能ライブラリ(SetCompJLimit)にて,各軸の電流制限値を設定します。電源立ち上げ時の初期値は0です。

(3) 力制限方法選択パラメータ

力制限時に先端の位置偏差ではなく速度偏差に応じた制御をする事で、**突き当て動作時の接触動作の安定性を向上します。**力制限特殊ライブラリ(SetCompVMode)にて速度制御モードに設定し、SetFrcLimitとSetDampRateにて制御方向を設定してください。

(例) Ｚ方向の速度制御モード設定

```
CALL SetFrcCoord(0)
CALL SetCompVMode
CALL SetFrcLimit(100,100,0,100,100,100)
CALL SetDampRate(100,100,100,100,100,100)
CALL SetCompControl
```

### 3.5.3.4　安全に関する注意事項

(1) 異常検出機能

力制限機能によって力を制限した場合、ロボットは、外力により力制限方向に動作する場合があります。また、制限方向以外の外力が加わった場合も、制限方向の分力により、動作する場合があります。

よって、ロボット調整時の安全を考慮し、以下の異常検出機能が追加されています。

① 力制限時位置偏差過大異常(60F8)

ロボットツール端の位置偏差が許容値を超えた場合に発生します。偏差許容値の初期値は、X,Y,Z方向が100(mm)、X,Y,Z周りが30(度)です。SetCompEralwにて偏差許容値を変更する事が可能です。しかし、偏差許容値を広げた場合、異常検出されなくなる為、必要異常に許容値を広げないでください。

② 力制限指令値異常(60FB)

ロボットツール端に異常な力を検出した場合に発生します。力制限割合を高く設定した方向にロボットを突き当て動作させると、このエラーが発生します。エラー発生時、力制限方向や力制限パラメータが正しく設定されているか確認ください。

③ 力制限時の指令速度制限値を超えました(60FC)
力制限中は、高速動作できません。外部速度と内部速度の積が50%を超えた状態で動作命令を実行すると、このエラーが発生します。エラー発生時、内部速度の設定値を下げてください。

④ 力制限時は、ＰＴＰ動作できません(60FD)
力制限中は、PTP動作（3.3.1 PTP制御参照）を禁止しています。力制限中にPTP動作を実行した場合、本エラーが発生します。CP動作に変更して頂くか、力制限を無効化した後、動作命令を実行してください。また、力制限中にコンティニュー起動した場合や、パス再開モードで再起動した場合も本エラーが発生します。

⑤ 力制限中の現位置がJ*ソフトリミットオーバ(66D*)
力制限中は、外力によりロボットが動作し、ソフトリミットを超えた位置へ移動する場合はあります。この場合、本エラーが発生します。手動モードでソフトリミットオーバが発生した場合、モータ電源はOFFしませんが、力制限中に本エラーが発生した場合は、モータ電源がOFFします。

(2) 安全に調整して頂く為に
力制限機能は、パラメータ設定の誤り等により、ロボットが重力方向に落下したり、外力を受け、想定した方向と異なる方向に動作する場合があります。
調整時の安全性を確保する為、角度偏差許容値、位置偏差許容値を必要以上に広げないでください。また、力制限方向以外の偏差許容値は、SetCompEralwにて初期値より小さい値(X,Y,Z方向10(mm)、X,Y,Z周り5(度)程度）に設定し、調整実施してください。

(3) 力制限機能のパラメータ調整
力制限時の制限力は、力制限割合の値で調整してください。ただし、力制限値と力制限割合との関係は、ロボットの姿勢、方向により変化します。力制限姿勢、方向毎に調整してください。
ロボットは、各関節の静止摩擦力より、外力による関節トルクが大となった場合、力方向に倣い動作します。静止摩擦力が外力による関節トルクより大の場合、ロボットは、力が釣り合った状態で停止します。
力制限中にロボットを動作させた場合、姿勢変化により摩擦力が変化する事で、摩擦力と制限力の釣り合い状態が変化し、ロボットが力制限方向（指令値方向）に動作する場合があります。力制限中にロボットを動作させる場合、力制限方向へのロボットの動作に注意してください。

(4) 位置教示
ロボットの位置は、変数の位置取り込み機能により教示します。変数位置取り込み時の位置は、ロボットの現在の指令位置です。力制限機能により、ロボットを直接押して移動し、位置取り込み（ダイレクト教示）する場合、ロボットの指令位置と実位置が一致しない為、教示位置とロボットの実位置とのずれが生じます。力制限を使って位置教示する場合は、以下の要領で作業してください。

① 教示番号用のI型変数を決める。I10を使用すると仮定する。
② 以下のプログラムを作成する。
P型変数に教示する場合

```
Program DirectTeach
    P(I10) = CURPOS
end
```

J型変数に教示する場合

```
Program DirectTeach
    J(I10) = CURJNT
end
```

③ 教示位置番号をI10に入力し、教示位置移動後、DirectTeachを実行する。

(5) ツール座標、ワーク座標設定
SetFrcCoordにて力制限座標をツール座標に設定し、力制限実行した場合、**力制限中にChangetoolにてツール座標を変更しても、力制限方向は変化しません。しかし、ロボット動作に関するツール座標は変化します。**よって、力制限座標系と動作座標系が一致しない状態になります。ワーク座標を設定した場合も同様です。
ツール座標のX,Y,Zの値を大きく設定し、力制限実行した場合、**ロボットの姿勢変化に対する制御感度が高くなり、ロボットが発振しエラー停止する場合があります。**その場合、SetFrcLimit、SetCompRateにて100に設定されたパラメータを100から徐々に下げて調整してください。

(6) コンティニュー機能
力制限中は、コンティニュー動作できません。力制限中にモータＯＦＦし、コンティニュー起動待ち状態でモータONするとエラー「66EF 力制限は解除されました」が発生します。また、力制限中に瞬時停止しコンティニュー起動すると、エラー「66FD 力制限中はPTP動作できません」が発生します。

(7) 復電機能
力制限中にコントローラ電源が落ちた場合、復電機能は無効になります。

(8) 力制限中のロボットの動作精度
力制限中に動作命令を実行した場合、ロボットの動作軌跡精度は低下します。特に力制限方向は大きく位置ずれを生じる場合があります。精度が必要な動作は、力制限機能を無効にしてください。
力制限中は、ロボットの停止精度が劣化します。力制限中にエンコーダ値確認動作(3.2.3エンコーダ値確認動作参照)を実行した場合、長時間停止したり、エラー「6651 チェック命令タイムオーバ」が発生する場合があります。停止精度が必要な動作は、力制限機能を無効にしてください。

## 3.5.4　衝突検出機能

### 機能の概要

ロボットおよび手先のツールと、周辺機器との衝突を、特別なセンサを使用せず検出し、ロボットを緊急停止させることで、衝突による被害を低減する機能です。手動操作時、自動運転時ともに使用することができます。

> ⚠**注意**：本機能で、衝突による被害を完全に回避することはできません。また、人に対する安全を保証するものではありません。

対応バージョン

各ロボットは下記のコントローラバージョンで機能します。

下記コントローラバージョンより古いプロジェクトデータを使用する場合は、「旧バージョンのプロジェクトデータについて」を参照してください。

VS-G シリーズ：Ver.2.61 以降

VS-***シリーズ：Ver.3.20 以降

VM-G、VP-G シリーズ：Ver.2.70 以降

HM-G、HS-G シリーズ：Ver.2.80 以降

注意：衝突検出機能は旧エンジンボードでは機能しません。新/旧エンジンボードの違いは「RC7M コントローラエンジンボード変更のご案内」を参照ください。

## 3.5.4.1　機能の使用方法

### 有効・無効設定

ライブラリまたはティーチングペンダントを使って衝突検出機能の有効・無効設定を各軸毎に行うことができます。ただし、付加軸を有効にすることはできません。ティーチングペンダントで行った設定はコントローラ電源 OFF 後も保持されます。ライブラリでの設定はコントローラ電源 OFF 後、元の設定に戻ります。初期値は「無効」です。

#### [1] ライブラリによる設定

ライブラリ SetCollisionJnt、ResetCollisionJnt を PAC プログラムに記述することでプログラム単位、動作単位で機能の有効・無効を設定できます。詳細は、プログラミングマニュアルⅡパックライブラリを参照してください。

#### [2] ティーチングペンダントによる設定

以下のティーチングペンダント操作により、機能の有効・無効を設定できます。ロボット操作中に設定はできません。

**操作経路：　基本画面－[F2 アーム]－[F6 補助機能]－[F2 衝突検出]**

(1)　ティーチングペンダントの画面上で、衝突検出したい軸の有効欄をタッチすると機能が有効になり、表示が黒色から緑色に変わります。再度タッチすると無効になり、表示が黒色に戻ります。




(2)　上の画面で、[F６　全軸選択]を押すと全軸有効になります。
また、[F４　全軸未選択]を押すと全軸無効になります。

## 外力最大値

ロボットを手動・自動で動作させた時に、ロボットが感知した外力の最大値を表示します。この値を参考にして検出レベルを設定してください。外力最大値は、モータ ON 状態の衝突検出有効設定軸のみ計算され、無効設定の軸では 0 が表示されます。

また、プログラム単位、動作単位での外力最大値を確認したいときは、動作前に現在の外力最大値をクリアします。

### [1] 外力最大値の表示方法

以下のティーチングペンダント操作により、外力最大値が表示できます。機能が有効な軸に対して外力最大値が表示されます。

**操作経路：　基本画面－[F2 アーム]－[F6 補助機能] －[F2 衝突検出]**




### [2] 外力最大値のクリア方法

衝突検出設定画面で、[F3 外力クリア]を押すと確認メッセージが表示されますので、[OK]を押します。手動・自動両方の外力最大値がクリアされます。また、コントローラ電源を OFF しても外力最大値がクリアされます。



## 検出レベルの設定

ライブラリまたはティーチングペンダントを使って衝突検出の感度（検出レベル）を各軸毎に設定することができます。入力範囲は 1～500 で、値を大きくすると感度が鈍り、小さくすると鋭くなります。

デフォルト値は 100 で、ロボットが定格負荷にて最大動作しても誤検出なく衝突を検出できる値です。

ティーチングペンダントで行った設定はコントローラ電源 OFF 後も保持されます。ライブラリでの設定はコントローラ電源 OFF 後、元の設定に戻ります。

> ⚠**注意**：検出レベルは誤検出防止のため、表示された外力最大値に余　を持たせて設定してください。目安としては外力最大値の1.2倍以上の値を設定してください。

### [1] ライブラリによる設定

ライブラリ SetCollisionLevel を PAC プログラムに記述することでプログラム単位、動作単位で検出レベルを設定できます。詳細は、プログラミングマニュアルⅡPAC ライブラリを参照してください。

### [2] ティーチングペンダントによる設定

以下のティーチングペンダント操作により、検出レベルを設定できます。ロボット動作中のレベル設定はできません。

**操作経路：基本画面－[F2 アーム]－[F6 補助機能] －[F2 衝突検出]**

(1) ティーチングペンダントの画面上で、手動時レベル、自動時レベルの該当欄をタッチします。



(2)　[F５ 値変更]を押し、テンキーで数値を入力し、[OK]を押します。




(3)　変更された値が表示されます。



## 衝突検出警告

衝突検出機能が有効の状態で、ロボットが検出した外力最大値が、設定された検出レベルを超えた時、ロボットは緊急停止し、ペンダントに衝突を知らせる警告が表示されます。
衝突を検出した軸に対して警告が表示されます。[OK]を押すと警告をクリアできます。




衝突検出設定画面を表示させると、外力最大値が検出レベルを超えていることが分かります。




| | |
|---|---|
| 注意: | ロボットまたはツールが設備等に接触した状態で　　停止した場合、警告表示をクリアし、そのままモータ電源をONすると、再び衝突を検出し、モータ電源をONできないことがあります。ロボットを　避するためにモータ電源をONしたい場合は、衝突検出機能を「無効」にするか、検出レベルを大きくしてからモータ電源をONしてください。 |

### 衝突検出機能使用時の注意点

（1）先端負荷が正しく設定されていないと衝突を誤検出する場合があります。ロボットの外力検出感度にも影響しますので、先端負荷は正しく設定してご使用ください。
（2）本機能は力制限機能と同時に使用することはできません。力制限機能を OFF した状態でご使用ください。
（3）付加軸では本機能は無効となります。
（4）メカの経年変化（グリス量の変化、ギアの磨耗など）により、ロボットが検出する外力が大きくなり衝突を誤検出する場合があります。その場合は検出レベルを上げてください。
（5）ロボットプログラム中で本機能の有効・無効の切り替えを指定する場合、ロボット軸の動作終了を待って切り替えを行います。ロボット軸が動作している間の切り替えはできません。
（6）エアバランス機能を搭載したロボットはエアバランスを正しく調整してください。エアバランス調整の不備は検出値に影響があり、誤検出となる場合があります。

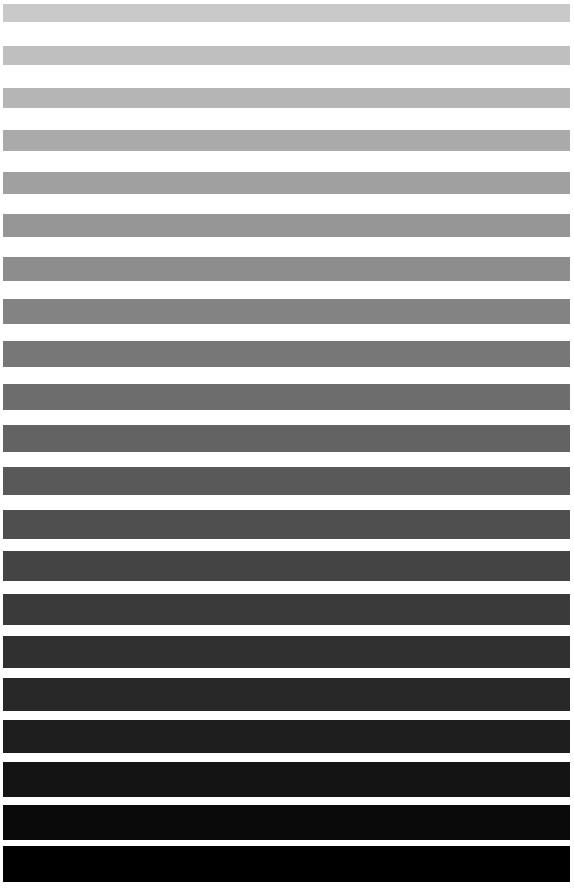
### 旧バージョンのプロジェクトデータについて

下記バージョンより古いプロジェクトデータを使用する場合は、次の手順により、プロジェクトデータを再構築する必要があります。

VS-G シリーズ：Ver.2.61 以降
VS-***シリーズ：Ver.3.20 以降
VM-G、VP-G シリーズ：Ver.2.70 以降
HM-G、HS-G シリーズ：Ver.2.80 以降

(1) コントローラ(上記バージョン以降のバージョン)に古いプロジェクトデータを転送します。
(2) 使用条件パラメータ、付加軸設定、ソフトウェアリミット、コンベアトラッキングの各設定値を控えておきます。
(3) WINCAPS3(上記バージョン以降のバージョン)にて新規プロジェクトを作成します。
(4) コントローラに新規プロジェクトのアームパラメータのみを送信します。
キャルセット値も送信しない様に注意してください。
(5) (2)で控えた設定をコントローラに反映します。
(6) コントローラのデータをすべて新規プロジェクトに受信します。

# 第 4 章
# 速度・加速度・減速度の指定



ロボットを動作させるには、速度、加速度、減速度を、ロボットの最大能力に対して、それぞれどれくらいの割合で動作させるかを設定する必要があります。
この章では、速度、加速度、減速度の意味と設定について説明します。

# 第 4 章　速度・加速度・減速度の指定

## 4.1　外部速度・内部速度

ロボットの速度には、外部速度と内部速度があります。
外部速度とは、プログラムの実行に先立って、ティーチングペンダントまたは外部機器より設定する速度のことをいいます。
内部速度とは、プログラムの中のコマンドによってのみ設定される速度のことをいいます。

# 4.2　速度指定

ロボットが動作する際の実際の速度は、外部速度と内部速度の積によって決まります。
たとえば、
　外部速度：70%
　内部速度：30%
という設定の場合、
　実際の速度 = 最高速度×0.7×0.3
となりますから、最高速度の21%が自動モードにおける実際の速度ということになります。

> ⚠ **注：手動モード、ティーチチェックモードでは、自動モード時の１０%の速度で動作します。**

プログラムの中でSPEEDコマンドによって内部速度を変更すると、内部加速度、内部減速度も自動的に設定されます。内部速度の二乗を100で割った値が、内部加速度と内部減速度となります。
JSPEEDコマンドは、内部軸速度を変更します。内部軸速度を変更すると、内部軸加速度、内部軸減速度も自動的に設定されます。内部軸速度の二乗を100で割った値が、内部軸加速度と内部軸減速度となります。

## 4.3　外部加速度・外部減速度・内部加速度・内部減速度

加速度と減速度には、それぞれ外部加速度・外部減速度および内部加速度・内部減速度があります。
外部加速度と外部減速度は、プログラムの実行に先立って、ティーチングペンダントまたは外部機器より設定する加速度のことをいいます。
内部加速度と内部減速度は、プログラムの中でコマンドによって設定される加速度と減速度のことをいいます。

# 4.4　加速度・減速度の設定

ロボットが動作する際の実際の加速度は、外部加速度と内部加速度の積によって決まります。減速度も同じです。
たとえば、
　外部減速度：70%
　内部減速度：30%
という設定の場合、
　実際の減速度 = 最高減速度×0.7×0.3
となりますから、最高減速度の21%が自動モードにおける実際の減速度ということになります。
加速度と減速度を設定するコマンドは、表4-1に示すとおりです。

> ⚠ 注：手動モード、ティーチチェックモードでは、自動モード時の１０%の加速度・減速度で動作します。

表4-1　加速度・減速度の設定コマンド

| コマンド | 機　能 |
|---|---|
| SPEED | 内部速度設定 |
| ACCEL | 内部加速度設定 |
| DECEL | 内部減速度設定 |
| JSPEED | 内部軸速度設定 |
| JACCEL | 内部軸加速度設定 |
| JDECEL | 内部軸減速度設定 |

# 4.5　速度・加速度設定例

図4-1に示すプログラムは、内部速度の設定をしません。
このようなプログラムを、外部速度80%の設定で実行すると、同図に示すような値になります。

```
PROGRAM PRO1
TAKEARM
   MOVE P, P1
END

実 速 度 = 外部速度×内部速度
         = 80%×100%
         = 80%

実加速度 = 外部加速度×内部加速度
         = 64%×100%
         = 64%

実減速度 = 外部減速度×内部減速度
         = 64%×100%
         = 64%
```

**図4-1　内部速度を設定しない例**

図4-2に示すプログラムは、内部速度を設定します。
このようなプログラムを、外部速度80%の設定で実行すると、同図に示すような値になります。

```
PROGRAM PRO2
TAKEARM
   SPEED 50          ’内部速度50%
   MOVE P, P1
END

実 速 度 = 外部速度×内部速度
         = 80%×50%
         = 40%

実加速度 = 外部加速度×内部加速度
         = 64%×25%
         = 16%

実減速度 = 外部減速度×内部減速度
         = 64%×25%
         = 16%
```

図4-2　内部速度を設定する例

図4-3に示すプログラムは、内部加速度と内部減速度を設定します。
このようなプログラムを、外部速度80%の設定で実行すると、同図に示すような値になります。

```
PROGRAM PRO1
TAKEARM
   ACCEL 50            'internal acceleration 50%
   DECEL 25            'internal deceleration 25%
   MOVE P, P1
END
```

実 速 度 = 外部速度×内部速度
= 80%×100%
= 80%

実加速度 = 外部加速度×内部加速度
= 64%×50%
= 32%

実減速度 = 外部減速度×内部減速度
= 64%×25%
= 16%

**図4-3　内部加速度と内部減速度を設定する例**

⚠ **注意：加速度、減速度を設定するときは、パス動作の経路変化に伴う衝突などの危険がないことを確認してください。**
**パス動作中に動作オプションやコマンドによって速度、加速度を変更した場合、経路が変化しますので、衝突などの危険がありますのでご注意ください。**

# 4.6　最適可搬質量設定機能

ロボットの先端負荷や姿勢に応じて、最適な速度、加速度を設定する機能です。
表 4-2 に示す 4 つのモードを選択できます。

表 4-2　最適可搬質量のモード

| モード | 設定条件 | 設定内容 | |
|---|---|---|---|
| | | PTP 動作 | CP 動作 |
| 0 | 負荷条件値 | 最高加速度 | 最高加速度 |
| 1 | 負荷条件値<br>ロボットの姿勢 | 最高速度、加速度 | モード 0 と同じ |
| 2 | | モード 0 と同じ | 最高速度、加速度 |
| 3 | | モード 1 と同じ | モード 2 と同じ |

## 4.6.1　モード 0

工場出荷時のデフォルトのモードです。ロボットの負荷条件値により、PTP 動作、CP 動作時の最高加速度を設定します。
設計時のサイクルタイムの計算は該当のロボット概要書の「位置決め時間」の資料を参照してください。

## 4.6.2　モード１

ロボットの負荷条件値と、動作中のロボットの姿勢により、PTP 動作時の 1、2、3 軸の最高速度と加速度を設定します。PTP 動作時の 4、5、6 軸と CP 動作は、モード 0 と同様です。

### 4.6.2.1　モード１を使用する場合

PTP 動作の動作時間短縮が必要な場合、モード 1 に設定してください。
モード 1 設定時の動作時間は、マシンロック運転にてご確認ください。

### 4.6.2.2　モード１使用時の注意事項

動作中に過負荷エラーや偏差過大エラーが発生する場合があります。負荷率については、調整時にティーチングペンダントの過負荷予想値（操作ガイド「5.3 ロボットの現在位置の表示、[F2 アーム]-［F6 補助機能.]-［F10 過負荷予想]」参照）または、WINCAPSⅢのログ機能を利用して、確認してください。
過負荷エラーが発生する場合は、タイマや内部速度、加速度を設定し、モータ負荷を調整してください。
偏差過大エラーが発生する場合は、速度、加速度を調整してください。

動作速度により、パス軌道が 20mm 程度変化する場合があります。したがって、障害物近傍のパス動作は、障害物に干渉する可能性があるため、モード 0 で実行してください。

## 4.6.3　モード 2

ロボットの負荷条件値と動作中のロボットの姿勢により、CP 動作時の最高速度と加速度を設定します。PTP 動作時は、モード 0 と同様です。

### 4.6.3.1　モード 2 を使用する場合

以下の 2 つの場合、モード 2 に設定してください。

・CP 動作時の動作時間短縮が必要な場合
動作中に速度を自動的に低減する場合がありますので、動作時間は、マシンロック運転にてご確認ください。

・モード 0 またはモード 1 を使用すると、CP 動作中に指令速度制限オーバエラー(6081～6086)が発生する場合、モード 0 またはモード 1 設定時、CP 動作時の軌道が特異点近傍（操作ガイド「4.1.3　腕・ひじ・手首の形態について［ 2 ］形態の境界」参照）や、動作範囲リミット近傍を通過する場合、指令速度制限オーバエラーが発生し、停止することがあります。この場合、モード 2 に設定すると、自動的に指令速度制限内に速度を低減し、エラーを発生せずに動作させることができます。

### 4.6.3.2　モード２使用時の注意事項

・動作中に過負荷エラーが発生する場合があります。調整時にティーチングペンダントの過負荷予想値（操作ガイド「5.3 ロボットの現在位置の表示、[F2 アーム]-［F6 補助機能.]-［F10 過負荷予想]」参照）または、WINCAPSⅢのログ機能を利用して負荷率を確認してください(WINCAPSⅢガイド「ログ機能」参照)。過負荷エラーが発生する場合は、タイマや内部速度、加速度を設定し、モータ負荷を調整してください。

・動作速度により、パス軌道が 20mm 程度変化する場合があります。したがって、障害物近傍のパス動作は、障害物に干渉する可能性があるため、モード 0 で実行してください。

・CP 動作時に、等速移動区間において、速度が変化する可能性があるため、等速移動を必要とする作業は、モード 0 かモード 1 で実行してください。

・CP 動作時に指令加速度制限オーバエラー(6761～6766)、偏差過大エラー(6111～6116）が発生する場合があります。エラーが発生した場合は、内部速度、内部加速度にて速度、加速度を調整してください。また、高速動作時最大 5mm 程度の軌跡ずれが発生する場合があります。動作近傍に障害物がある場合は、速度を下げて使用してください。

・特異点近傍（操作ガイド「4.1.3　腕・ひじ・手首の形態について［ 2 ］形態の境界」参照）で動作速度が低下中に瞬時停止させた場合、瞬時停止時間が延びる場合があります。ただし、瞬時停止距離は変わりません。

### 4.6.4　モード3

PTP 動作時はモード 1、CP 動作時はモード 2 と同様に動作します。「4.6.2　モード 1」、「4.6.3　モード 2」を参照してください。

### 4.6.5　設定が必要な使用条件

ロボットの第 6 軸に取り付く、負荷（ツールとワーク）の質量と重心位置が判明したら、先端負荷質量と重心位置を設定してください。先端負荷質量と重心位置は、ティーチングペンダントまたは WINCAPSⅢの使用条件にて設定します(「4.6 最適可搬質量設定機能」参照)。また、作業中のツール、ワークの変更に対応するため、プログラム中でも変更できます。プログラムによる変更は、「4.7.2 内部負荷条件値(先端負荷質量、負荷重心位置)と内部モード設定方法」を参照してください。

## 4.6.6　設定にあたって

この機能で設定する負荷条件値（先端負荷質量、重心位置）とモードには、ティーチングペンダント、WINCAPSⅢから設定する外部負荷条件値、外部モードと、プログラム内で設定する内部負荷条件値、内部モードがあります。

**外部負荷条件値、外部モードを設定すると、内部負荷条件値、内部モードも同じ値になります。**

**ロボットは、内部負荷条件と内部モードから最高速度、加速度を設定し、動作します。**

現在の内部負荷条件値と、内部モードは、ティーチングペンダントにて確認できます（「4.7「使用条件」における最適可搬質量設定機能」参照）。

**注意①: 負荷条件値は、負荷に応じた正しい値を必ず設定してください。**
**設定値が正しくないと、動作中に過電流エラーや偏差過大エラー、過負荷エラーなどが発生します。またロボット故障の原因にもなり、異常発生時、ロボット停止入力時などに停止距離が伸び、周辺設備に衝突する危険があります。**

**②: ロボットの設置条件を正しく設定してください。設定は、ティーチングペンダント、WINCAPSⅢにて設定します。**
**設定値が正しくないと、動作中に過電流エラーや偏差過大エラー、過負荷エラーなどが発生します。またロボット故障の原因にもなり、異常発生時、ロボット停止入力時などに停止距離が伸び、周辺設備に衝突する危険があります。**

**■コントローラ電源立ち上げ時のモードについて**

コントローラ電源立ち上げ時の設定を、最適可搬質量設定モードの「モード0」に初期化するか、現在の設定モードを保持するかが選択できます。
（「4.7.4　最適可搬質量初期化設定の設定方法」参照）

# 4.7　「使用条件」における最適可搬質量設定機能

## 4.7.1　外部負荷条件値(先端負荷質量、負荷重心位置)と外部モード設定方法

### ■ティーチングペンダントによる設定

操作経路：基本画面―[F2 アーム]―[F6 補助機能.]―[F7 使用条件.]

ティーチングペンダントを操作して、上記の操作経路を経ると、[使用条件(パラメータ数：)]画面が現れます。現在の内部負荷条件値と内部モードなどが表示されます。(操作ガイド「2.8 負荷質量、負荷重心、最適可搬質量に関する基本パラメータの設定 (TP/WC) ■ティーチングペンダントを使って」を参照してください。)




この[使用条件 (パラメータ数：)]画面で、以下の項目を選択し、[F5　設定変更.]を押すと、テンキー形式の[パラメータ変更]画面が表示され、各パラメータ値を変更できます。

設定項目：
[最適可搬質量設定モード]
[先端負荷質量 (g)]
[負荷重心位置X (mm)]
[負荷重心位置Y (mm)]

[負荷重心位置Z (mm)]、
　但し、＜４軸ロボットのVer.1.9以降＞は[負荷イナーシャ(kgcm$^2$)]

最適可搬質量設定モードは 0～3 で、それ以外の数値を入力すると「入力された値が範囲外です。」が発生します。

先端負荷質量は、ロボット毎に設定された許容値を超えた値を入力すると「入力された値が範囲外です。」が発生します。

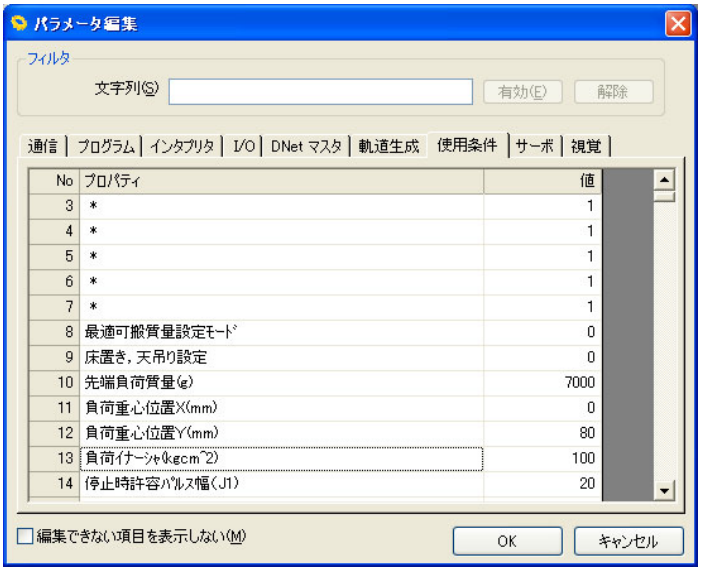
負荷重心位置は、許容範囲を満たすように入力してください。ロボット毎に設定された許容範囲を満たさない場合は、「入力された値が範囲外です。」が発生します。



## ■　WINCAPSⅢによる設定

プログラミング支援ツールWINCAPSⅢによって、外部負荷条件値（先端負荷質量、重心位置）と外部モードの設定を行なう方法について説明します。（操作ガイド「2.8 負荷質量、負荷重心、最適可搬質量に関する基本パラメータの設定（TP/WC）■WINCAPSⅢを使って」および、WINCAPSⅢガイド「パラメータの確認」を参照してください。）

WINCAPSⅢの［プロジェクト(P)］メニューから［パラメータ(T)］を選択すると、［パラメータ編集］ウィンドウが表示されます。
［パラメータ編集］ウィンドウの［使用条件］タブをクリックすると、使用条件パラメータの一覧が表示されます。

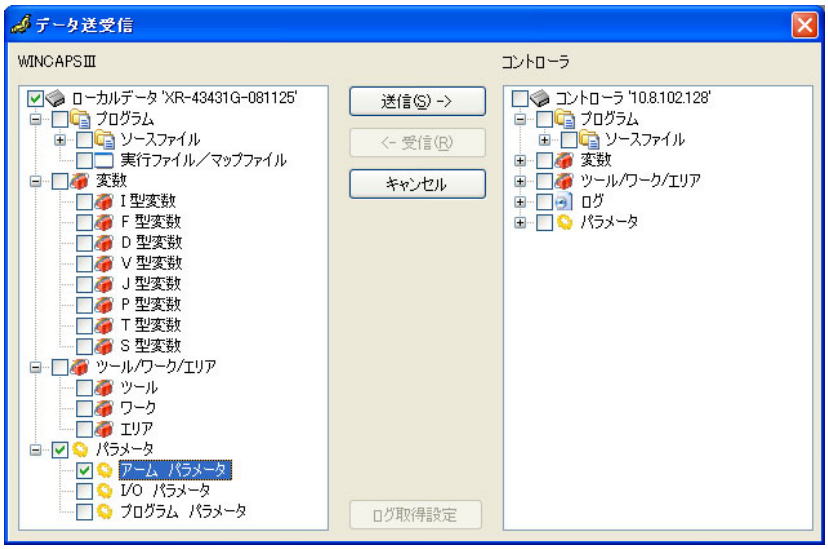



この画面で、以下の項目の設定値欄をダブルクリックすると、各パラメータ値を入力変更できます。

設定項目：
［最適可搬質量設定モード］
［先端負荷質量（g)］
［負荷重心位置X（mm)］
［負荷重心位置Y（mm)］

［負荷重心位置Z（mm)］
　但し、＜４軸ロボットのVer.1.9以降＞は［負荷イナーシャ(kgcm²)］

各パラメータ値の設定ができたら、ロボットコントローラにデータを転送します。
まず、ティーチングペンダントの[MOTOR]で、モータ電源をOFFにします。
WINCAPSⅢの[通信(N) ]の［データ送受信(T)...］をクリックして［データ送受信］ダイアログボックスを表示させ、送信の操作を行います。
データ送信については、(WINCAPSⅢガイド「4.7 ロボットコントローラとの通信」を参照してください。




注意①：［データ送受信］ダイアログボックスにある［<-受信(R)］をクリックすると、コントローラのデータを受信できます。この場合、外部負荷条件値と外部モードが受信されます。プログラム中で変更した内部負荷条件値と内部モードは、WINCAPSⅢ側に転送されません。
②：負荷重心位置は、ツール0座標系で入力してください（図4-9参照）。単位はmmです。ツール0座標系の原点は、6軸フランジ中心、Y成分はフランジ中心からΦ6H7ピン穴方向（オリエントベクトル方向)、Z成分はフランジ中心を通りフランジ面に垂直な方向（アプローチベクトル方向)、X成分は、オリエントベクトルをY軸、アプローチベクトルをZ軸としたときの、右手座標系におけるX軸方向（ノーマルベクトル方向）になります（図4-10参照)。

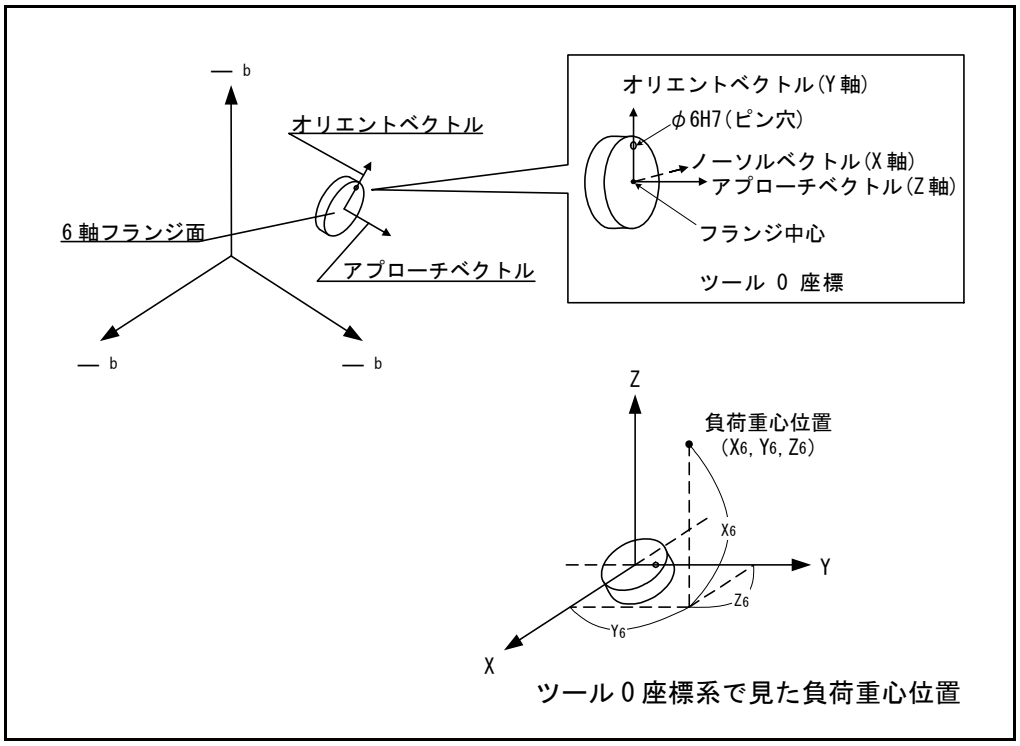



図4-9　負荷重心位置

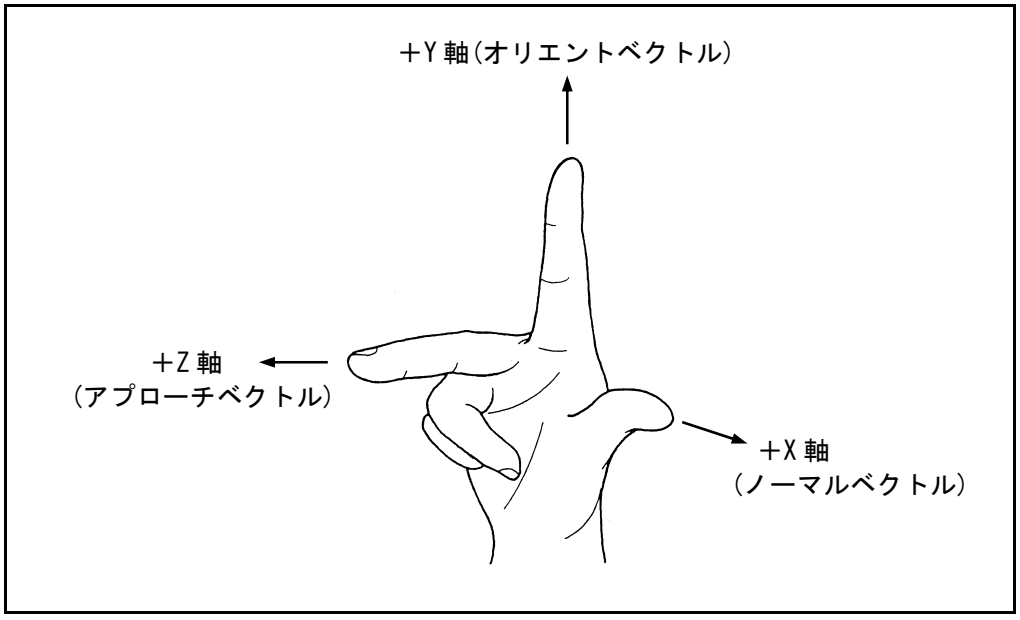



図4-10　右手座標系

## 4.7.2　内部負荷条件値(先端負荷質量、負荷重心位置)と内部モード設定方法

### 4.7.2.1　内部負荷条件値設定方法

従来言語ライブラリ aspACLD を実行することで設定します。詳しくは、「22.1 従来言語、aspACLD」を参照してください。

### 4.7.2.2　内部モード設定方法

従来言語ライブラリ aspChange を実行することで設定します。詳しくは、「22.1 従来言語、aspChange」を参照してください。

## 4.7.3　ロボットの設置条件設定方法

6軸ロボットの場合、床置きにて使う場合と、天吊りにて使う場合の設定をします。床置きは「0」、天吊りは「1」に設定します。工場出荷時は、「0」（床置き）に設定されています。天吊りで使用する場合には、設定を変更してください。

### ■　ティーチングペンダントによる設定

**操作経路：基本画面―[F2 アーム]―[F6 補助機能.]―[F7 使用条件.]**

ティーチングペンダントを操作して、上記の操作経路を経ると、[使用条件（パラメータ数：)] 画面が現れます。現在の内部負荷条件値と内部モードなどが表示されます。（操作ガイド「2.9 ロボット設置条件の設定（TP/WC）■ティーチングペンダントを使って」を参照してください。



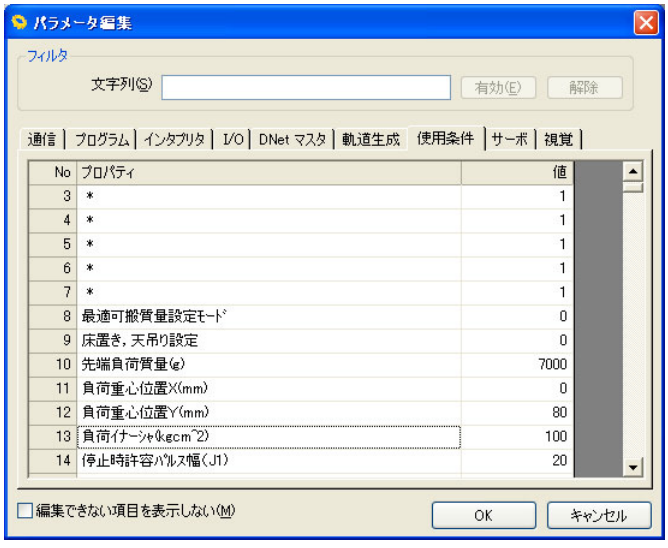

この［使用条件（パラメータ数：)］画面で、［床置き、天吊り設定］の項目を選択し、［F5 設定変更.］を押すと、テンキー形式の［パラメータ変更］画面が表示され、各パラメータ値を変更できます。
設置条件は「0」および「1」に設定してください。それ以外の値を入力すると「入力された値が範囲外です。」が発生します。




注意：ティーチングペンダントで設定した設置条件は、WINCAPSⅢへ転送してください。
転送方法は、p.4-18の注意①を参考にしてください。

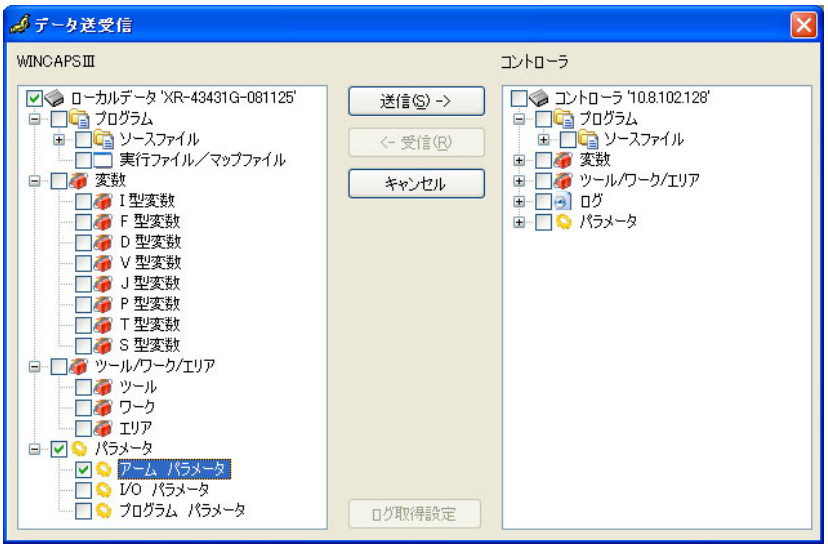
## ■ WINCAPSⅢによる設定

プログラミング支援ツールWINCAPSⅢによって、床置き、天吊りでロボットを使う設定方法について説明します。(操作ガイド「2.9 ロボット設置条件の設定 (TP/WC) ■WINCAPSⅢを使って」および、WINCAPSⅢガイド「パラメータの確認」を参照してください。)
WINCAPSⅢの［プロジェクト(P)］メニューから［パラメータ(T)］を選択すると、［パラメータ編集］ウィンドウが表示されます。
［パラメータ編集］ウィンドウの［使用条件］タブをクリックすると、使用条件パラメータの一覧が表示されます。



この画面で、[床置き、天吊り設定] の設定値欄をダブルクリックすると、パラメータ値を入力変更できます。

パラメータ値の設定ができたら、ロボットコントローラにデータを転送します。
まず、ティーチングペンダントの[MOTOR]で、モータ電源をOFFにします。
WINCAPSⅢの[通信(N) ]の [データ送受信(T)...] をクリックして [データ送受信] ダイアログボックスを表示させ、送信の操作を行います。
データ送信については、(WINCAPSⅢガイド「4.7 ロボットコントローラとの通信」を参照してください。




注意：[データの送受信] ダイアログボックスにある [<-受信] をクリックすると、コントローラのデータを受信できます。この場合、外部負荷条件値と外部モードが受信されます。プログラム中で変更した内部負荷条件値と内部モードは、WINCAPSⅢ側に転送されません。

## 4.7.4　最適可搬質量初期化設定の設定方法［V1.4 以降］

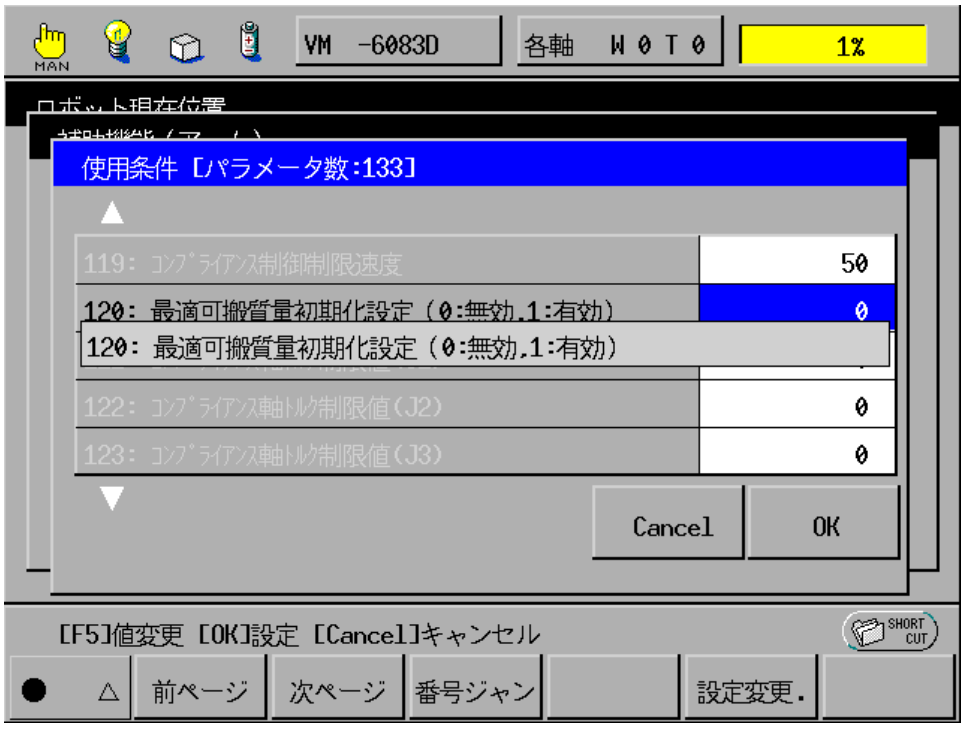
コントローラ電源ON時に最適可搬質量設定モードをモード0に初期化したい場合、または現在設定されている値のままにしたい場合の設定をする。

| 設定値 | 設定値の意味 |
|---|---|
| 0 | コントローラ電源投入時に最適可搬質量設定モードをモード０に初期化する。（工場出荷時） |
| 1 | コントローラ電源を投入しても最適可搬質量設定モードを初期化しない。（現在の値を保持する） |

### ■ティーチングペンダントによる設定

操作経路：基本画面―［F2 アーム］―［F6 補助機能］―［F7 使用条件］

ティーチングペンダントを操作して、上記の操作経路を経ると、［使用条件（パラメータ数：）］画面が現れます。現在の内部負荷条件値と内部モードなどが表示されます。




この［使用条件（パラメータ数：）］画面で、「最適可搬質量初期化設定」の項目を選択し［F5　設定変更.］を押すと、テンキー形式の［パラメータ変更］画面が表示され、各パラメータ値を変更できます。

0：無効→コントローラ電源投入時初期化

1：有効→コントローラ電源を投入しても初期化しない。
（現在の値を保持する）

最適可搬質量初期化設定は0～1で、それ以外の数値を入力すると「入力された値が範囲外です。」が発生します。

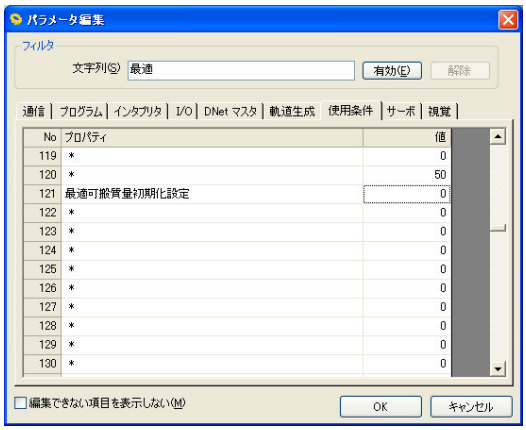



注意：ティーチングペンダントで設定した最適可搬質量初期化設定はWINCAPSⅢへ転送してください。転送方法はP4-18の注意②を参考にしてください。

## ■WINCAPSⅢによる設定

プログラミング支援ツールWINCAPSⅢによって、最適可搬質量初期化設定の設定を行う方法について説明します。

WINCAPSⅢの［プロジェクト(P)］メニューから［パラメータ(T)］を選択すると、［パラメータ編集］ウィンドウが表示されます。
［パラメータ編集］ウィンドウの［使用条件］タブをクリックすると、使用条件パラメータの一覧が表示されます。

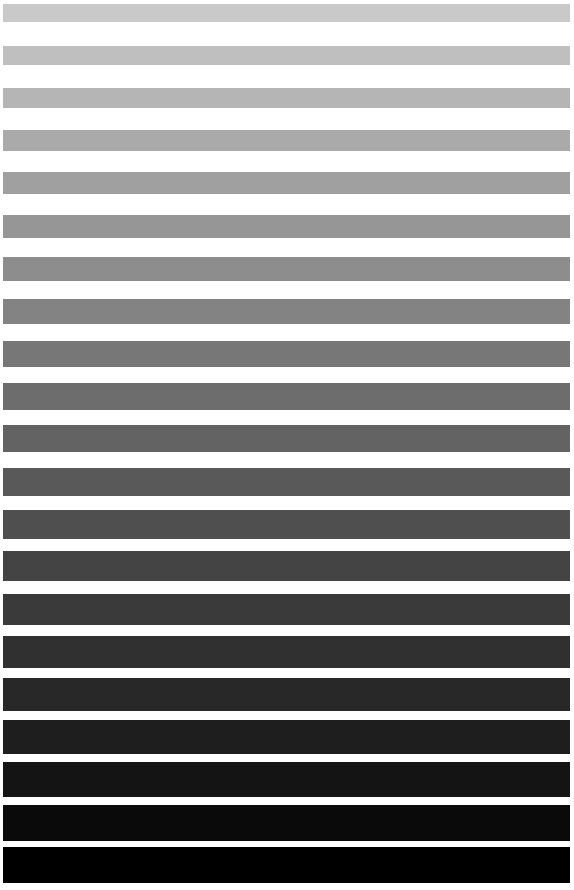



この画面で、最適可搬質量初期化設定の設定項目をダブルクリックすると、パラメータ値を入力変更できます。

各パラメータ値の設定ができたら、ロボットコントローラにデータを転送します。
まず、ティーチングペンダントの［MOTOR］で、モータ電源をＯＦＦにします。
WINCAPSⅢの[通信(N) ]の［データ送受信(T)...］をクリックして［データ送受信］ダイアログボックスを表示させ、送信の操作を行います。
データ送信については、(WINCAPSⅢガイド「4.7 ロボットコントローラとの通信」を参照してください。

**注意：［データの送受信］ダイアログボックスにある［<-受信］をクリックすると、コントローラのデータを受信できます。プログラム中で変更した最適可搬質量初期化設定はWINCAPSⅢ側に転送されません。**

# 第 5 章
# 視覚制御



プログラムを作成する際に必要となる、視覚に関する用語について説明します。

# 第５章　視覚制御

## 5.1　視覚制御

オプションでロボットコントローラに内蔵される、μVisionボードを利用するためのコマンドについて説明します。
視覚装置を利用するプログラムの作成・入力を行なうときにお読みください。

### 5.1.1　視覚制御の用語

#### 5.1.1.1　画素

μVisionボードの内部では、画素データを一つ一つの点の集まりとして取り扱っています。その一つの点のことを「画素（PIXEL)」と呼びます。
μVisionボードが扱う画素数は、格納メモリ（処理画面）で横512画素×縦480画素、描画専用メモリ（描画画面）で横624画素×縦480画素です。

#### 5.1.1.2　輝度

μVisionボードで処理する画像データの各画素は、明るさを表す数値を持っています。この数値のことを輝度といいます。輝度は0～255の256段階の数値をとります。0に近いほど暗く、255に近いほど明るくなります。

#### 5.1.1.3　ウィンドウ

μVisionボードで画像処理をする範囲として、図5-1に示すように、ウィンドウを設定します。ウィンドウは、μVisionボード内部にウィンドウ番号ごとにその大きさが記憶されます。ウィンドウを編集する方法にはWINCAPSⅢの視覚機能による方法と、ユーザプログラムによる方法の2つがあります。ユーザプログラムにより編集したウィンドウ情報は、電源を切ると失われてしまうので、注意してください。

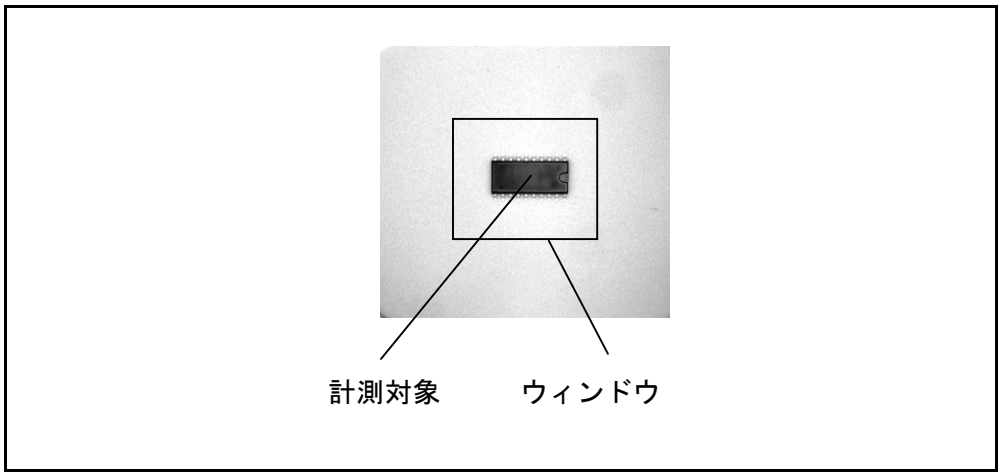



図5-1　画像処理ウィンドウ

## 5.1.1.4　面積・重心・主軸角

### ［１］　面積

カメラから取り込んだ画像データを2値化し、ウィンドウで指定した範囲内の白(1)と黒(0)のそれぞれの画素数を、面積としてカウントします。μVisionボードでは、ウィンドウ内の各画素の輝度値を変更せずに、リアルタイムに2値化して面積をカウントするので、事前に画像データを2値化する必要はありません。

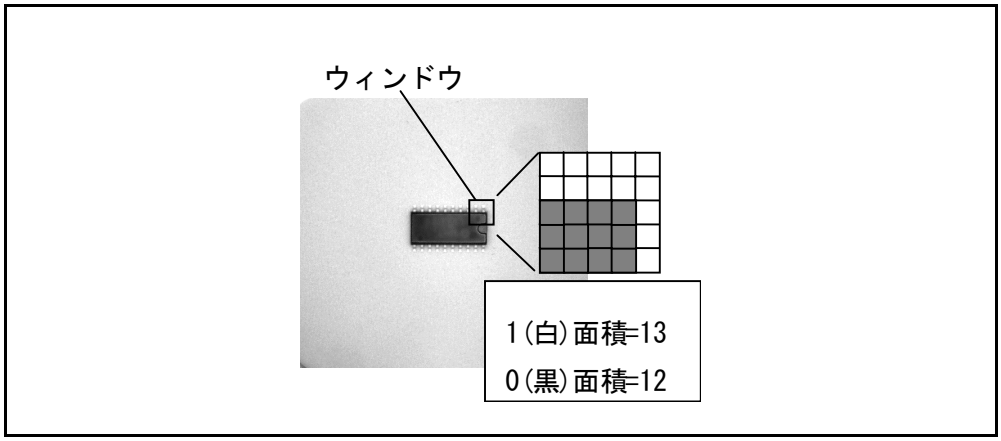



図5-2　面積

### ［２］　重心

カメラから取り込んだ画像データの中では、対象物は平面になります。平面上で、対象物の重さの釣り合う点を、重心といいます。
μVisionボードでは、ウィンドウで指定した範囲内の白(1)と黒(0)の画素から、重心を求めています。重心はX座標値とY座標値で表されます。

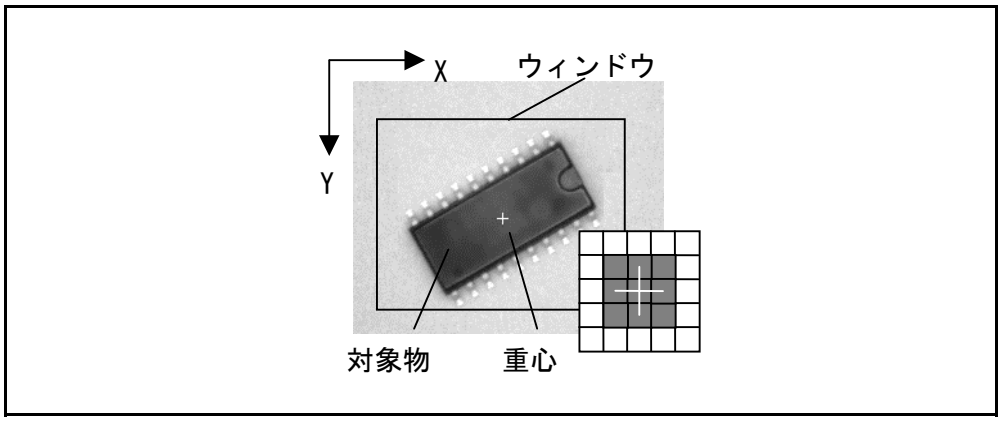



図5-3　重心

## ［３］ 主軸角

カメラから取り込んだ画像データの中では、対象物は平面になります。この平面の対象物を回転させたときに、最も良く回る長手方向の軸を主軸の長軸といい、この軸に直交する軸を主軸の短軸といいます。
μVisionボードでは、水平軸(X軸)から主軸の長軸までの角度(Θ)を、主軸角と定義しています。主軸角は、画像データを2値化した後に、ウィンドウで指定した範囲の白(1)または黒(0)の対象物に対して求めます。

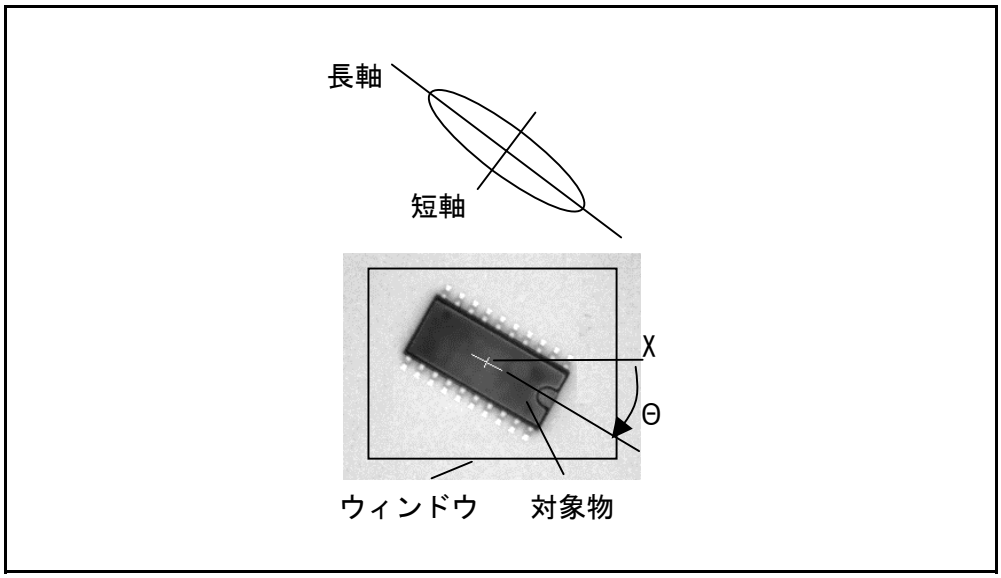



図5-4　主軸角

注意：正方形および真円の主軸角は特定することができませんので注意してください。

## 5.1.1.5　2 値化

### ［ 1 ］　2 値化

カメラからμVisionボードに取り込んだ画像データは、各画素ごとに256階調の輝度を持っています。2値化とは、各画素の輝度を、しきい値を境にして白(1)と黒(0)に書き直すことをいいます。このしきい値を、2値化レベルといいます。μVisionボードは2値化レベルを2値下限と2値上限の2つの値で指定します。2値下限より大きな値から2値上限までを白(1)、それ以外を黒(0)に2値化します。

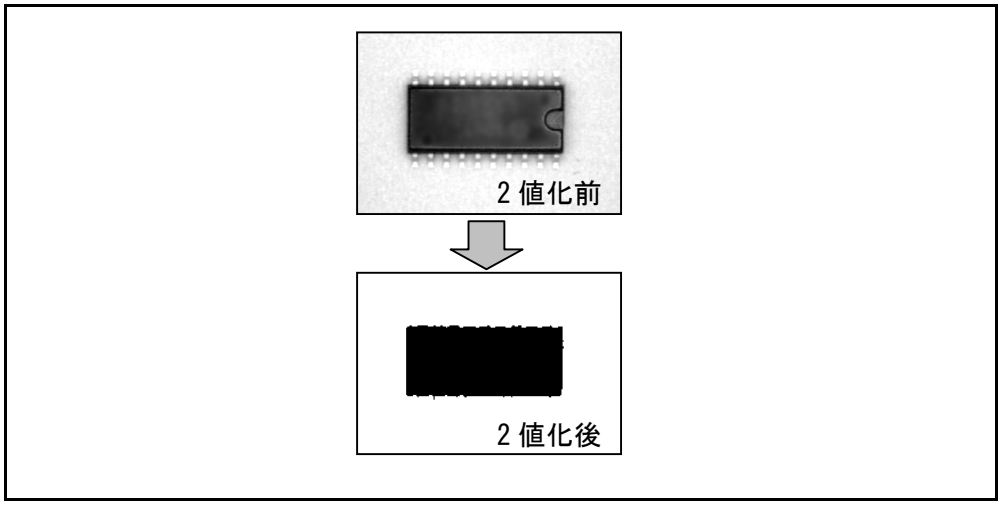



図5-5　2値化

### ［ 2 ］　ヒストグラム

ヒストグラムとは、カメラから取り込んだ画像データのウィンドウで指定した範囲について、輝度値の出現頻度をカウントしたものです。ヒストグラムをグラフ表示すると、輝度値の分布状況が分かりやすくなり、画像データを2値化する際の2値化レベルの決定が容易になります。また、μVisionボードでは、2値化レベルを自動決定する機能を提供していますが、これらはヒストグラムを利用して、2値化レベルを決定しています。図5-6は、2値化前の画像データのヒストグラムであり、対象と背景が2つに別れている様子が分かります。図5-7は、異なった2値化レベルで2値化を行なった結果です。下限(2値下限)と上限(2値上限)を調整することで、2値化結果が異なることが分かります。

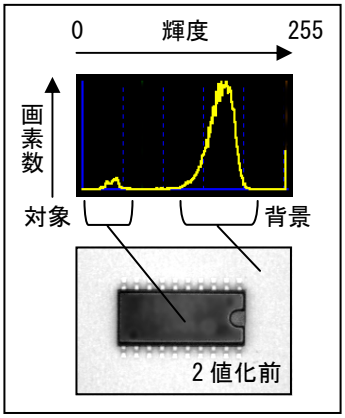



図5-6　ヒストグラム(2値化前)

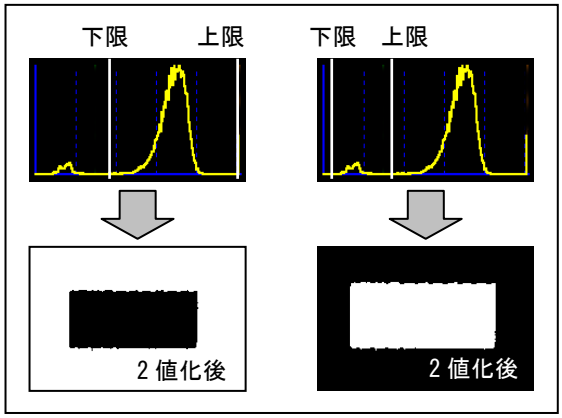



図5-7　ヒストグラム(2種類の2値化)

## ［３］ ２値化レベル検出

### モード法

画像のヒストグラムが、2つのピーク（対称図形と背景に対応する）を持つ二峰性の分布になる場合、この2つの山の間にある、谷の部分に2値化レベル（t）を設定する方法をモード法といいます。

μVisionボードでは、輝度値0から255への方向でヒストグラムの山・谷を検出するため、図5-8の場合は、tの点を谷と見て2値化レベルとします。

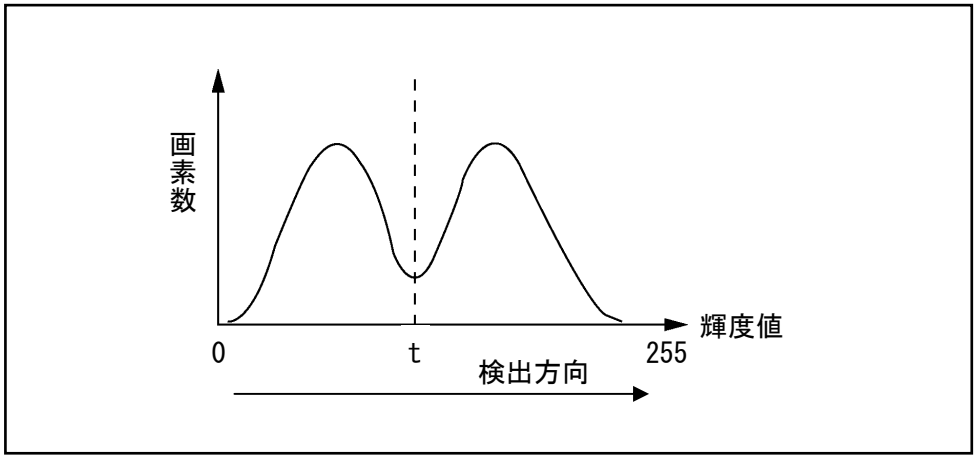



図5-8　モード法による2値化レベル

### 判別分析法

画像ヒストグラムにおいて、2値化レベルKで黒部分と白部分の2つのクラスに分割したとき、この2つのクラス間の分離が最も良くなるように、統計手法により2値化レベルKを決める方法を判別分析法といいます。

ヒストグラムが、2つのピークを持つ二峰性の分布でない画像には、前出のモード法は適していません。そのような場合には、判別分析法により最適なしきい値を得ることができます。

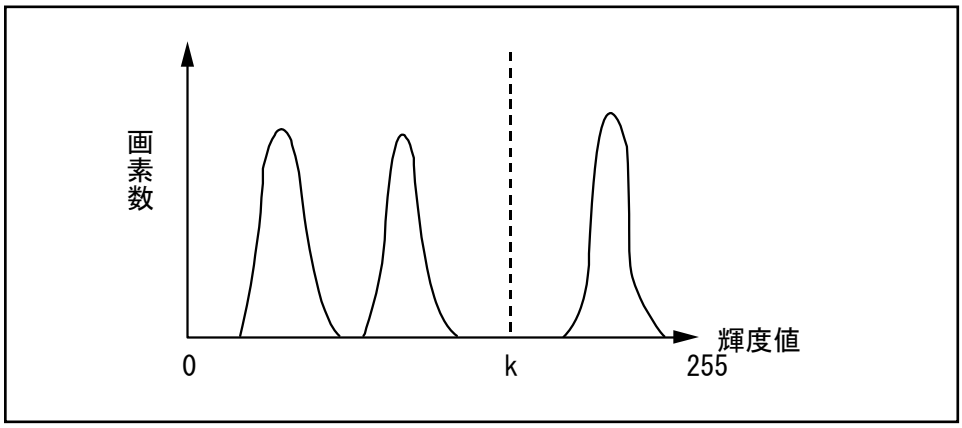



図5-9　判別分析法による2値化レベル

### Pタイル法

あらかじめ対象の白もしくは黒の面積がSxと分かっている場合に、ヒストグラムを利用し、対象の面積がSxに一致する2値化レベルを検出する方法です。
対象が白の場合には、ヒストグラムの明るい輝度値の画素から加算して面積を求め、ちょうどSxになる輝度を2値化レベルとします。
対象が黒の場合は、暗い輝度値の画素から加算して同様に2値化レベルを求めます。
あらかじめ対象の面積が分かっている場合に有効な方法です。

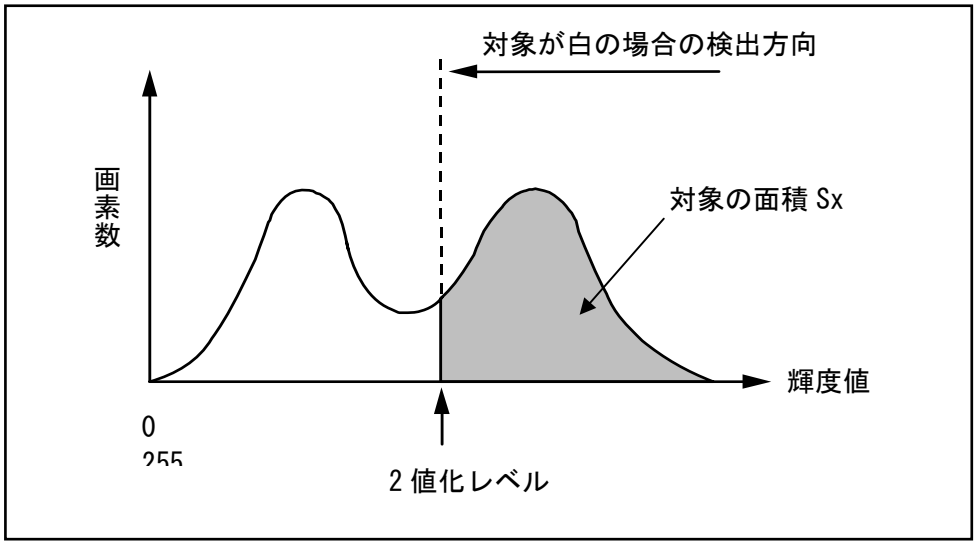



図5-10　Pタイル法による2値化レベル

### 5.1.1.6　輝度積分値

カメラから取り込んだ画像のうち、ウィンドウで指定した範囲のすべての画素の輝度値を集計することを輝度積分といいます。集計した結果の値を輝度積分値といいます。

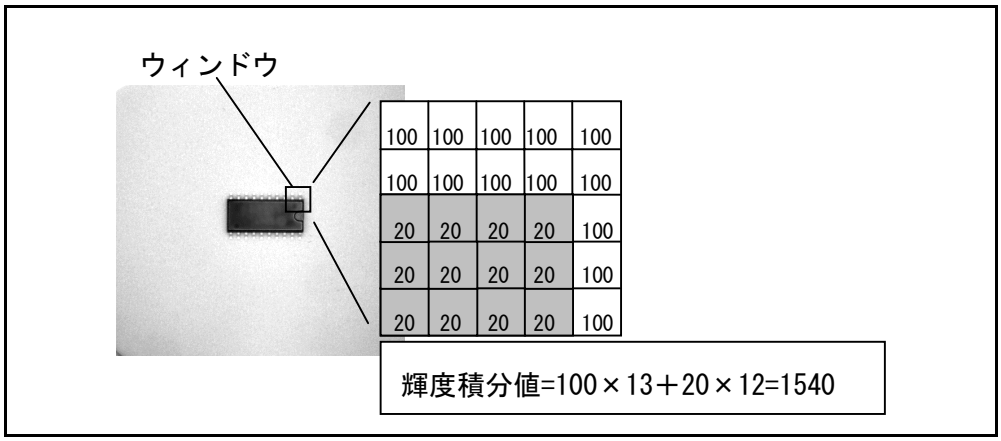



図5-11　輝度積分値

### 5.1.1.7　エッジ

指定したウィンドウ内の対象物について、暗から明(黒→白)、明から暗(白→黒)へ変化している点、つまり輝度の変化点をエッジといいます。

μVision ボードで検出するエッジの位置は、指定したレベル値をウィンドウ内の輝度値または面積値が満足する箇所です。レベル値の指定には、絶対値と差分値があり、対象物の状況により使い分けます。絶対値は、指定した箇所を、輝度値または面積値が通過する位置を検出します。差分値は、輝度値または面積値の変化量が、指定した値よりも大きくなった位置を検出します。

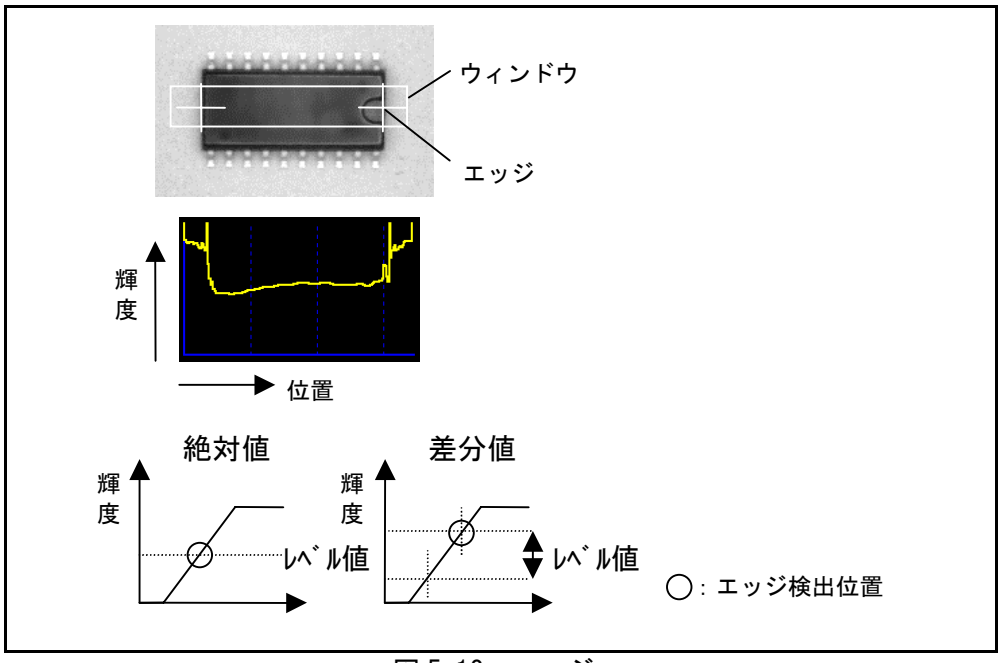



図 5-12　エッジ

## 5.1.1.8　ラベリング

カメラから取り込んだ画像データを2値化し、白(1)または黒(0)の画素の連結領域に、順に番号を付ける処理を、ラベリングといいます。

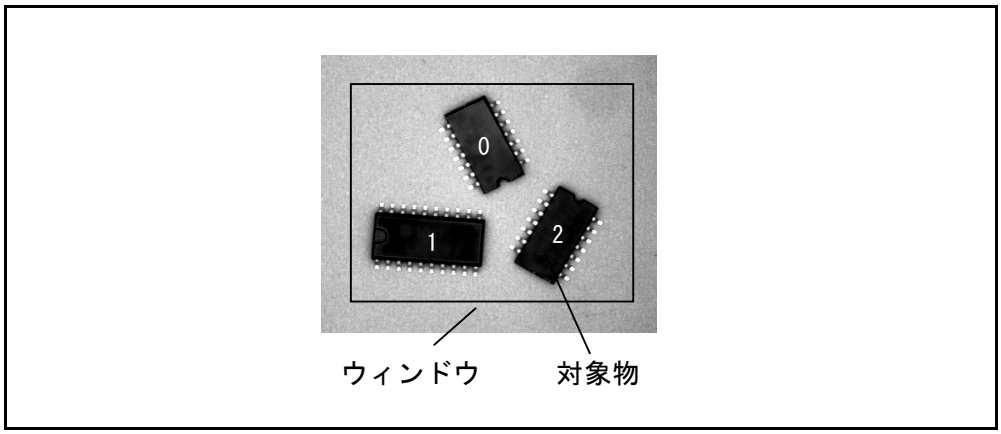



図 5-13　ラベリング

ラベリングにより、ウィンドウで指定した範囲に存在する複数の対象物を、個々に区別して扱えるようになります。
μVision ボードでは、ラベリングを行なうと、個々の対象物の特徴として、面積、重心、主軸角、フィレ形状、周囲長が求まります。フィレ形状とは、対象物に外接する長方形のことです。

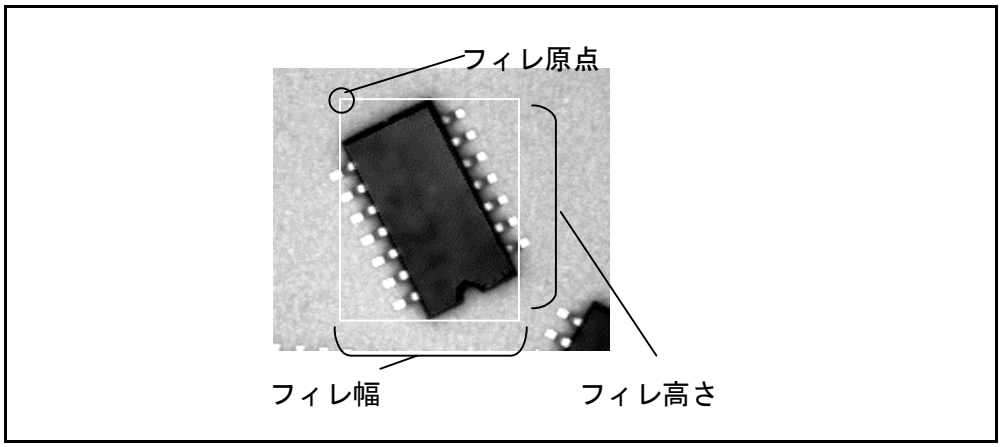



図 5-14　フィレ形状

また、周囲長とは対象物の外形を構成する画素をカウントしたものです。

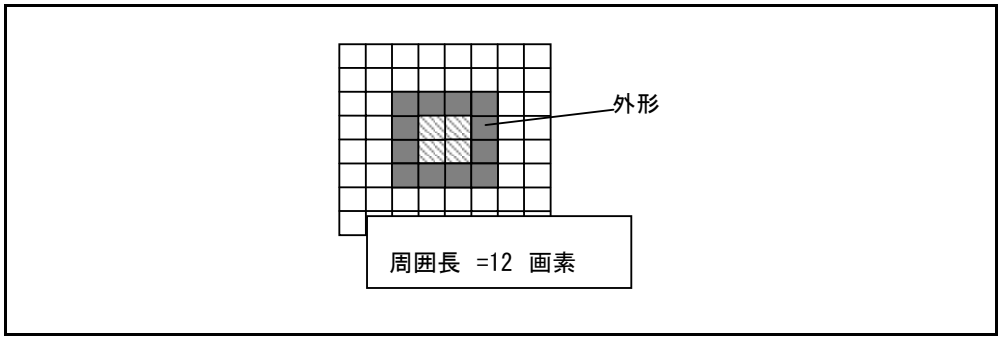



図5-15　周囲長

## 5.1.1.9　サーチ

サーチとは、あらかじめ登録された、標準的な画像データ(サーチモデル)を計測対象画像の探索範囲内(ウィンドウ範囲内)で動かして、一致した場所を探すことをいいます。

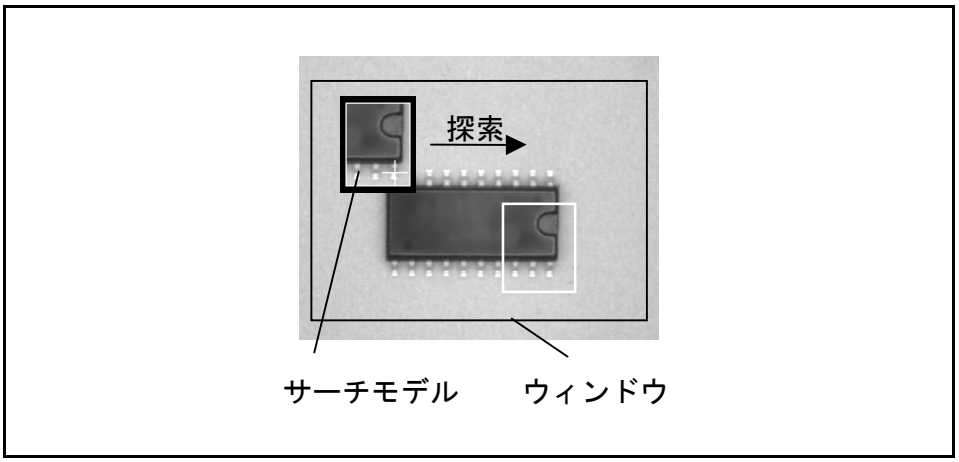



図5-16　サーチ

μVisionボードでは、標準的な画像データをサーチモデルといい、画像データと基準座標(OX,OY)により構成されます。

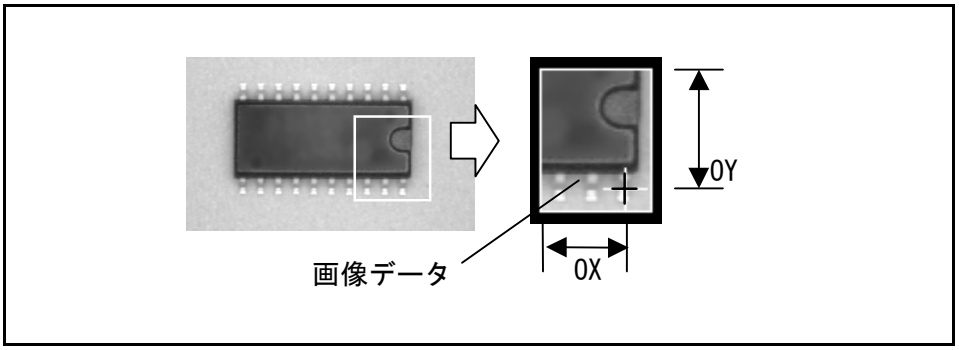



図5-17　サーチモデル

また、サーチモデルと計測対象画像の、一致する度合を表す数値を「一致度」といいます。指定した値よりも大きな一致度が得られた場合に、サーチモデルと計測対象画像の一致した場所の座標を得ることができます。座標の検出精度は、ピクセルでは最小単位が1画素になり、サブピクセルでは1画素以下の精度で、結果を得ることができます。サブピクセルにより計測を行なった場合は、ピクセルによる計測よりも、計測時間が長くなります。

## 5.1.2視覚命令を使うにあたっての注意点（視覚ボード）

ロボットコントローラにおいて視覚命令を使用する際は、μVisionボード（オプション）が必要です。
視覚命令は視覚処理権取得命令（TAKEVIS）を先立って実行する必要があります。μVision-21では必要ありません。

```
(例)  TAKEVIS
      CAMIN 1
      VISPLNOUT 0
      :
```

命令実行時にエラーとなった場合、カメラ入力命令以外の命令では次に視覚命令を実行した時点でエラーメッセージが表示されます。

```
(例)  WINDMAKE R,1,100,10,0,2
      VISPRJ 1,100,100,1,128   ←  この時点で“ウインドウ形状異常”
                                   エラー
      WINDDISP 1               ←  次の命令実行でエラーメッセージを
                                   表示します。
      :
```

WINCAPSIIIから視覚ボードに対し通信を行っている際に、視覚処理権取得命令（TAKEVIS）を実行した場合はエラーになります。

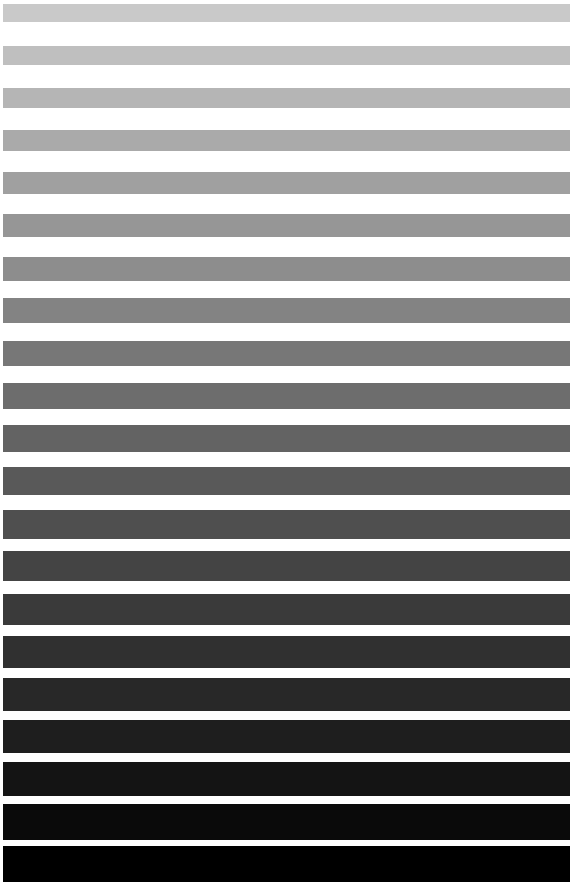
## 5.1.3従来のμVision-15との互換性

従来のμVision-15の機能は外部出力（パラレルI/O、RS232C）関連以外の機能は継承しています。ただし、命令の書式などに変更がありますので、従来のプログラムを参考に編集し直す場合、下記の対応表をご参照ください

| | μVision-15 | μVisionボード | 説明 |
|---|---|---|---|
| 画像入出力 | VIN | CAMIN | カメラ入力 |
| | AOUT | VISCAMOUT | カメラ画像の表示 |
| | VOUT | VISPLNOUT | 処理画面の表示 |
| | | VISOVERLAY | オーバレイ設定 |
| ウィンドウ設定 | WNDALN | WINDMAKE | ウィンドウの設定 |
| | WNDCIR | | |
| | WNDDPT | | |
| | WNDELP | | |
| | WNDRCT | | |
| | WNDSCT | | |
| | HWIND | | |
| | CLRWND | WINDCLR | ウィンドウ設定データの消去 |
| | WNDCPY | WINDCOPY | ウィンドウ設定データのコピー |
| | WDISP | WINDDISP | ウィンドウの表示 |
| 描画 | SCREEN | VISSCREEN | 描画条件の設定 |
| | CLS | VISCLS | 画面の消去 |
| | PUTP | VISPUTP | 点の描画 |
| | LINE | VISLINE | 直線（長さ、角度指定）の描画 |
| | LNDPT | VISPTP | 直線（2点指定）の描画 |
| | RECT | VISRECT | 矩形の描画 |
| | CIRCLE | VISCIRCLE | 円の描画 |
| | ELIPSE | VISELLIPSE | 楕円の描画 |
| | SECT | VISSECT | 扇の描画 |
| | PAINT | VISRECT | 矩形の描画 |
| | | VISCIRCLE | 円の描画 |
| | | VISELLIPSE | 楕円の描画 |
| | CROSS | VISCROSS | 十字マークの描画 |
| | CRSALN | | |
| | LOC | VISLOC | 文字表示位置指定 |
| | PRINT | VISDEFCHAR | 描画文字の設定 |
| | | VISPRINT | 文字の描画 |
| 画像処理 | GETP | VISGETP | 指定座標の輝度取得 |
| | HIST | VISHIST | ヒストグラム実行 |
| | HCLR | | |
| | | VISREFHIST | ヒストグラム結果の取得 |
| | LEVEL | VISLEVEL | 2値化レベル計算 |
| | BINA | VISBINA | 2値化処理 |
| | RBINA | VISBINAR | 2値化表示 |
| | AREA | VISMEASURE | 特徴(面積、重心、主軸角等)計算 |
| | BIGT | | |
| | CENTR | | |
| | MOMENT | | |
| | STRT | | |
| | PROJ | VISPROJ | 投影処理 |
| | EDGE | VISEDGE | エッジ計測 |
| サーチ機能 | CORN | SHCORNER | コーナーのサーチ |
| | CIRC | SHCIRCLE | 円のサーチ |

# 第２部
# コマンドリファレンス

# 第 6 章
# コマンドの表記方法と分類



デンソーロボットで使用されるロボット言語PACのコマンドの表記方法とコマンド一覧を記載します。
個別のコマンドについての情報を手早く探し出すには、コマンド一覧表をご利用ください。

# 第６章　コマンドの表記方法と分類

## 6.1　コマンド解説の表記方法

第9章以降に、PACの各コマンドについて解説しています。ここでは、その表記方法について説明します。



### 9.4　ホーム座標

**HOME**(1)（ステートメント）(2)【SLIM 準拠】

(3) **機能**　　任意の座標をホームポジションとして宣言します。

(4) **書式**　　HOME 〈ポジション型〉

(5) **説明**　　〈ポジション型〉で指定した任意の座標をホームポジションとして宣言します。
ホームポジションは各プログラムごとに定義されます。
HOME 文を複数宣言した場合は、最新の宣言をホームポジションとします。

(6) **関連項目**　　GOHOME

(7) **用例**

```
6 軸        DIM lp1 As Point
            HOME (350, 0, 450, 0, 0, 180 )    '(350, 0, 450, 0, 0, 180 )の座標をホーム
                                              'ポジションとして宣言します。
            HOME lp1                          'lp1 の座標をホームポジションとして宣言
                                              'します。
```

**コマンド解説の表記方法**

(1) コマンドの名称：コマンドの名称が、はじめに記されます。

(2) コマンドの種別：コマンドの名称に続いて、コマンドの種別(ステートメント、関数、システム変数、組み込み定数)が記されます。SLIMの規格に準拠しているコマンドには、【SLIM準拠】と記します。

(3) 機能：コマンドの機能について説明します。

(4) 書式：文法上の記述方法について説明します。コマンドが引数を持つ場合は、引数を“<”と“>”でくくって表記します。複数の引数がある場合には、区切り記号“,”が用いられます。
省略可能な要素は“[”と“]”でくくって表記します。
“[<a>[,<b>]]”のように“[”と“]”でくくられた“[,<b>]”は、<a>が省略された場合には省略する必要があることを表します。
複数の要素のうち、どれか一つだけを必要とする場合には、複数の要素を“{”と“}”でくくって表記し、各引数を“|”で区切って表記します。

(5) 説明：コマンドの使い方について説明します。

(6) 関連項目：関連する他のコマンドや説明箇所を記します。

(7) 用例：コマンドの使用例を説明します。

## 6.2　コマンド一覧表

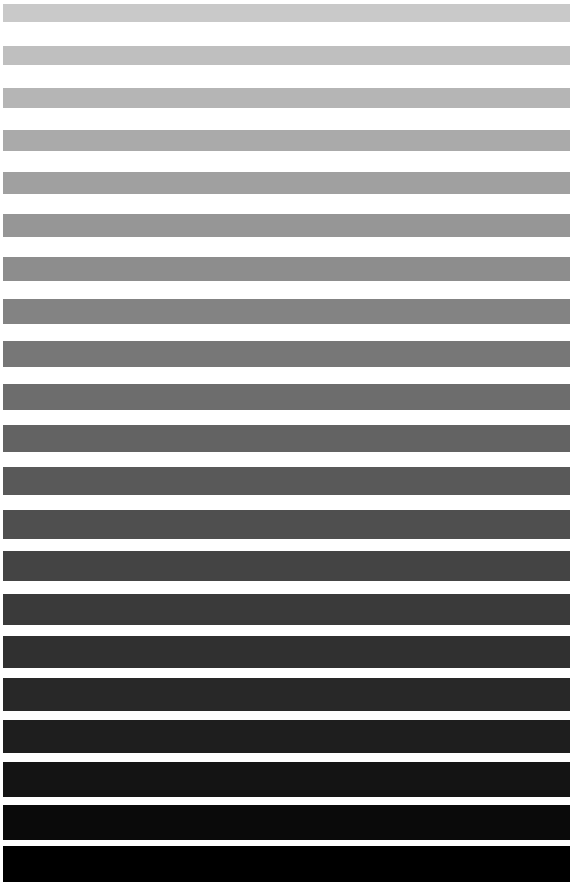
### 6.2.1　アルファベット順コマンド一覧

目次の「アルファベット順コマンド一覧」を参照してください。

### 6.2.2　機能別コマンド一覧

目次の「機能別コマンド一覧」を参照してください。

# 第 7 章
# PAC 言語の構成要素



この章では、PAC言語を構成する要素について説明します。

# 第７章　PAC言語の構成要素

## 7.1　新しいロボット言語PAC

ロボットの動作や作業を記述するためのプログラミング言語を、ロボット言語といいます。
デンソーロボットで使用されるロボット言語は、PAC(Programming language for Assembly Cell)と名付けられています。PACは、ロボット制御プログラムの開発とメンテナンスを、従来以上に効率化するために、新しく開発されました。主な特徴を以下に示します。

・JISで規定された、産業用ロボット言語SLIMの上位互換言語である。
・構造化プログラミング言語であるため読みやすく、開発とメンテナンスが容易である。
・ロボットだけでなく、視覚装置の制御も、PACで統一的に記述できる。
・マルチタスク機能により、プログラムの効率的な処理ができる。
・割り込み処理機能により、エラーの発生や時刻などによる例外的な処理を、効率よく記述できる。

# 7.2　ロボット言語 PAC と従来の言語との関係

従来、ロボット言語は、ロボットによって、あるいはメーカによって、互換性のないことが多くありました。ロボットごとに異なった言語を操るのはユーザにとって負担になるので、標準化をはかるために、JISで産業用ロボットプログラム言語SLIMが制定されました。

SLIMは、主に組立作業を記述することを目的に規格化(JIS B 8439)されたもので、BASIC言語を基本として、ロボット特有のデータタイプやロボット動作命令などを追加したものです。したがって、SLIM以前のロボット言語に慣れた人でも、比較的違和感なくSLIM言語を習得できるという利点があります。

デンソーロボットで使用されるロボット言語PACは、BASIC言語の上位互換である構造化BASIC言語を基本にしていますので、SLIMの利点はそのままに引き継いだうえで、さらに構造化言語やマルチタスク機能などの数々の特徴を備えています。ロボット特有の命令に関しては、仕様がSLIMと同じですから、PACはSLIM言語の上位互換にもなっています。したがって、PACは「従来のロボット言語に慣れた人でも、比較的違和感なく習得できる」というSLIMの利点を活かしつつ、高度な要求にも応えられるロボット言語になっています。

## 7.3　言語要素

PAC言語を構成する要素には、以下のものがあります。

●識別子……構成要素を識別するための名前
●変　数……データを一時的に記憶するもの
●定　数……一定した値を持つデータ
●演算子……2つの値の間で計算をする記号
●式……値を求めるための構成要素の組み合わせ
●コマンド……処理を実行するようPAC言語に組み込まれた命令

また、変数や定数のデータにはいくつかの型があります。
この章では、構成要素およびデータ型について説明します。
コマンドについては、第9章～第22章で詳しく説明しています。

## 7.4　名前

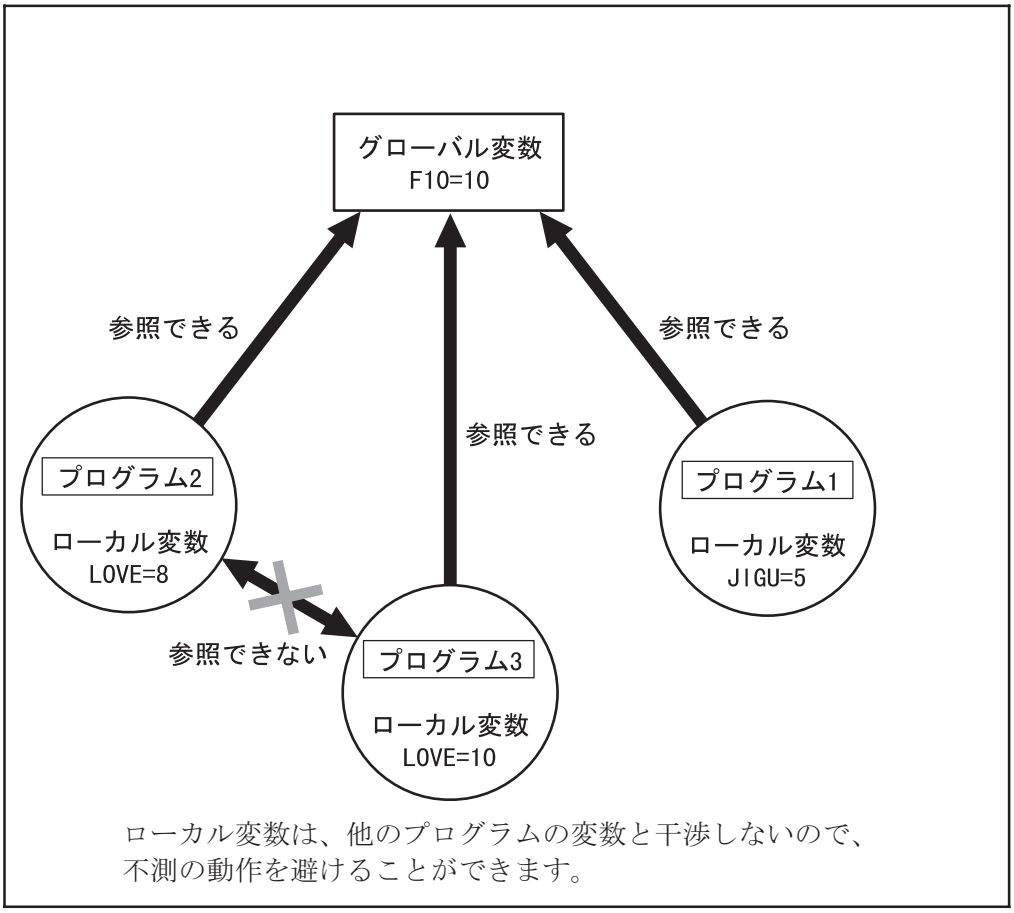
PAC言語には、プログラムの中にあるさまざまな要素を識別するための規則があります。この章では、この規則について説明します。なお、コマンド、変数、関数、ラベル、プログラムを表す名前は、次の規則に従います。

●名前は、文字（半角アルファベット。大文字と小文字の区別はしない）または、決められた記号で始まらなければなりません。
●名前には、文字、数字、アンダースコアが使用できます。
●名前の先頭文字はアルファベットでなければなりません。
●ピリオド、スラッシュ、バックスラッシュ、スペース、コロン、セミコロン、シングルクォート、ダブルクォーテーション、アスタリスクは使用できません。
●+、-、*、/、(、)などの、演算子として使われる文字は使用できません。
●名前を他の語と見分けるために、スペース文字を名前と他の語との間に置きます。
●名前に使用できる文字数は、最大64文字です。

# 7.5　識別子

## 7.5.1　変数

変数は、プログラム中で使うデータを一時的にしまっておくものです。グローバル変数とローカル変数、システム変数があります。
グローバル変数は、どのプログラム（タスク）からも共通して使用できる変数です。
ローカル変数は、一つのプログラムの中でのみ有効な変数です。同時に実行される他のプログラムの中に、同じ名前のローカル変数があっても、それぞれが属するプログラムの中で働き、相互に影響することはありません。




グローバル変数とローカル変数

## [1] グローバル変数

グローバル変数の名前は、型を表すアルファベット（I、F、D、S、V、P、J、T、IO）の後ろに整数式を付けて表します。I/O変数だけはアルファベットが2文字（IO）になります。
たとえば、F0001、F1、F[1]はすべて、同じ単精度実数型変数を表します。
変数名は、システムで予約されていますから、宣言をすることなく使えます。
グローバル変数には、下記に示す型が使えます。

- I型：整数型 （範囲：-2147483648～+2147483647）
  例）I0001、I1、I[1]
- F型：単精度実数型 （-3.402823E+38～3.402823E+38）
  例）F0001、F1、F[1]
- D型：倍精度実数型 （-1.7976931348623157D+308～1.7976931348623157D+308）
  例）D0001、D1、D[1]
- S型：文字列型（最大で243文字）
  例）S0001、S1、S[1]
- V型：ベクトル型 （X,Y,Z）
  例）V0001、V1、V[1]
- P型：ポジション型（X,Y,Z,RX,RY,RZ,FIG） （6軸）
  例）P0001、P1、P[1]
- J型：ジョイント型（J1,J2,J3,J4,J5,J6） （6軸）
  例）J0001、J1、J[1]
- T型：同次変換型 （Px,Py,Pz,Ox,Oy,Oz,Ax,Ay,Az,FIG）
  例）T0001、T1、T[1]
- IO型：I/O型
  例）IO0001、IO1、IO[1]

**注意：視覚装置μVision-21ではＶ型、Ｐ型、Ｊ型、Ｔ型は使用できません。**

### グローバル変数の間接参照

グローバル変数を指定する際に、変数の番号を数式で指定することを、変数の間接参照といいます。
間接参照をする場合は、変数の番号を[ ]で括った数式で表します。

用例：

```
I1 = I[5*3]     'I15の値をI1に代入します。
F1 = F[I1]      'I1の値を番号とするF型変数の値を、F1に代入します。
D1 = D[I1+1]    'I1の値に1を加えた値を番号とするD型変数の値を、D1に
                '代入します。
S1 = S[I7]      'I7の値を番号とするS型変数の値を、S1に代入します。
V1 = V[I1]      'I1の値を番号とするV型変数の値を、V1に代入します。
MOVE L, P[I5]   'I5の値を番号とするP型変数の位置へ移動します。
J1 = J[I5*3]    'I5の3倍の値を番号とするJ型変数の値を、J1に代入します。
T1 = T[I1*3]    'I1の3倍の値を番号とするT型変数の値を、T1に代入します。
SET  IO[I1*5]   'I1の5倍の値を番号とするビット型ポートをONします。
```

## [2] ローカル変数

ローカル変数には、グローバル変数と同様に、下記の型の変数が使えます。

- I型：整数型（範囲：-2147483648～+2147483647）
- F型：単精度実数型（-3.402823E+38～3.402823E+38）
- D型：倍精度実数型（-1.7976931348623157D+308～1.7976931348623157D+308）
- S型：文字列型（最大で243文字）
- V型：ベクトル型（X,Y,Z）
- P型：ポジション型（X,Y,Z,RX,RY,RZ,FIG）　（6軸）
- J型：ジョイント型（J1,J2,J3,J4,J5,J6）　（6軸）
- T型：同次変換型（Px,Py,Pz,Ox,Oy,Oz,Ax,Ay,Az,FIG）
- IO型：I/O型

ローカル変数は、型宣言命令を使用して、型宣言をした後に使用できます。
数値型と文字列型のローカル変数は、型宣言文字を使用して型宣言を行なうこともできます。型宣言命令および型宣言文字については、P8-6「8.6　宣言文」を参照してください。

注意①：型宣言をせずに変数を使用すると、単精度実数型変数として機能しますが、型宣言なしでの変数の使用は、プログラムミスの原因ともなります。可能な限り、型宣言を行なうように心掛けてください。

②：プログラミング支援ツールWINCAPSⅢのデフォルト設定では、「明示的な型宣言が必ず必要」の設定になっています。この設定では、型宣言を省略すると、コンパイル時にエラーが出るようになっています。特に理由がない限り、この設定のままご使用ください。

③：プログラム中で、I/O型以外のローカル変数を参照する場合には、参照する以前になんらかの値をローカル変数に代入しておくこと（変数の初期化）が必要です。ローカル変数を初期化せずに参照すると、実行時エラーが発生します。このような場合は、あらかじめ、プログラム中に変数への代入文を記述するか、配列変数以外ではプログラムの変数宣言部に、 DEFINT A = 1 などと記述することにより、変数を初期化できます。

④：ローカル変数を持つプログラムが、CALL コマンドにより呼び出され、実行中・待機中・一時停止中の状態にあるとき、そのプログラムを再起動すると、最初の呼び出しによるタスクと、再起動によるタスクの 2 つのタスクが存在することになります。別のタスクではあっても同じプログラムではローカル変数の独立が確保できないので、予期せぬ誤動作をすることがあります。
この誤動作を防止するために、CALL コマンドによってプログラムを呼び出しているときは、そのプログラム単体での起動を行なわないようにしてください。

⑤：視覚装置μVision-21 ではＶ型、Ｐ型、Ｊ型、Ｔ型は使用できません。

### [3] システム変数

システム変数は、システムの状況を調べるための変数です。変数名は、システムの予約語になっているので、変数名を宣言する必要はありません。
システム変数には以下のものがあります。

| | | | | | | |
|---|---|---|---|---|---|---|
| CURJNT | CURPOS | CURTRN | CURFIG | CURACC | CURDEC | CURJDEC |
| CURJSPD | CURSPD | DESTJNT | DESTPOS | DESTTRN | DATE$ | TIME$ |
| TIMER | ERL | ERR | | | | |

## 7.5.2　関数

関数は、あらかじめ決められた演算方法により、指定された値（引数）に対する演算結果を得るものです。関数によっては引数を持たないものもあります。
PAC言語の関数には、ポーズデータおよびベクトルデータを扱う関数、数値データを扱う算術関数、文字列データを扱う関数、ユーザ定義関数の4種類があります。
関数コマンドについては、第15章「関数」を参照してください。

## 7.5.3　ラベル

ラベルは、プログラム中での文の位置を表します。分岐先の位置をラベルで表すことができます。
ラベルの名前を意味のある名前にしておくと、プログラムがわかりやすくなります。

## 7.5.4　プログラム

プログラムの名前を使って、プログラムを指定し、プログラムの中から、別のプログラムを呼び出すことができます。
プログラムの名前は、プログラムの先頭でPROGRAMコマンドで宣言します。
プログラムの名前については、「8.2　プログラム名と宣言」で詳しく説明しています。

プログラム例：プログラム名を使ったプログラムの呼び出し

プログラムを呼び出す側のプログラム

```
PROGRAM PRO1
    IF IO[138] = ON THEN
        CALL MOTION
    ELSE
        DELAY 200
    END IF
END
```

呼び出される側の「MOTION」プログラム

```
PROGRAM MOTION
    TAKEARM
    SPEED 100
    MOVE P, P1
    DELAY 200
    MOVE P, P2
END
```

# 7.6　データ型

PAC言語で扱うデータには、文字列、数値、位置、ベクトル、I/Oの型があります。以下に、これらのデータ型について説明します。

## 7.6.1　文字列

文字列（String）型データは、S型とも呼びます。
最大で243文字までの文字列を含むことができます。

## 7.6.2　数値

数値型データには、次の3種類があります。
　I型：整数型　（範囲：-2147483648～+2147483647）
　F型：単精度実数型　（-3.402823E+38～3.402823E+38）
　D型：倍精度実数型　（-1.79769313486231D+308～1.79769313486231D+308）

## 7.6.3　ベクトル

ベクトル型データは、V型とも呼びます。
X、Y、Z成分からなる3つの単精度実数パラメータより構成されます。
　V型：ベクトル型　　(X,Y,Z)

## 7.6.4　ポーズ

ポーズ型データには、次の3種類があります。
　P型：ポジション型（X,Y,Z,RX,RY,RZ,FIG）
　J型：ジョイント型（J1,J2,J3,J4,J5,J6）
　T型：同次変換型　　(Px,Py,Pz,Ox,Oy,Oz,Ax,Ay,Az,FIG)

## 7.6.5　I/O（ON/OFF）

I/O型のデータは、I/Oポートの状態（ONまたはOFF）を値として持ちます。

# 7.7　データ型の変換

異なるデータ型の間で、型を変換することが可能です。

## 7.7.1　数値

数値データは、次の規則に従って型変換を行なうことができます。

●数値データを、違った型の数値変数に代入すると、数値は変数の型に合わせて変換されます。

●型の違う数値どうしで演算を行なうと、精度の高い方の型に合わせて演算します。

●論理演算では、数値は整数に変換して演算し、結果は整数になります。

●実数が整数に変換される場合には、その実数の値を超えない最大の整数に丸められます。

●倍精度実数が単精度実数に代入された場合、結果は有効数字7桁に丸めたものになります。

## 7.7.2　文字と数値

PAC言語には、文字、文字列、文字コード、数値を変換するために、以下のようなコマンドがあります。

文字と数値の変換コマンド

| 変換前＼変換後 | 文字コードへ変換 | 文字列へ変換 | 数　値へ変換 | 2進表記文字列へ変換 | 16進表記文字列へ変換 |
|---|---|---|---|---|---|
| 文字コード | | CHR$ | − | − | − |
| 文字 | ASC | | VAL$ | − | − |
| 数値 | − | STR$ | | BIN$ | HEX$ |

## 7.7.3　ポーズ型データ

ポーズ型データは、次の関数を用いて型変換を行なうことができます。

ポーズ型データには、型変換実行時のツール、ワーク座標系が反映されます。

ポーズ型データの変換コマンド

| 変換前＼変換後 | P型へ変換 | J型へ変換 | T型へ変換 |
|---|---|---|---|
| P型 | | P2J | P2T |
| J型 | J2P | | J2T |
| T型 | T2P | T2J | |

用例：J0 = P2J (P0)
　　T0 = P2T (P0)
　　P0 = J2P (J0)
　　T0 = J2T (J0)
　　P0 = T2P (T0)
　　J0 = T2J (T0)

# 7.8　定数

定数は、固定した値を持つ式です。
PAC言語の定数は次のように分類できます。

| | |
|---|---|
| 1 数値データ | I型：整数型定数 |
| | F型：単精度実数型定数 |
| | D型：倍精度実数型定数 |
| 2 文字列データ | S型：文字列型定数（最大で243文字） |
| 3 ベクトルデータ | V型：ベクトル型定数（X,Y,Z） |
| 4 ポーズデータ | P型：ポジション型定数（X,Y,Z,RX,RY,RZ,FIG）(6軸の場合) |
| | J型：ジョイント型定数（J1,J2,J3,J4,J5,J6）（6軸の場合） |
| | T型：同次変換型定数（Px,Py,Pz,Ox,Oy,Oz,Ax,Ay,Az,FIG） |
| | （V型、P型、J型、T型の各要素は単精度実数です） |

以下に、各型の定数について説明します。

## 7.8.1　数値定数

### [１]　整数型定数

-2147483648 から +2147483647 の範囲の整数による定数です。
整数型定数には、10進、2進、16進の3種類の表記形式があります。

#### 10進形式

10進数で表現される整数型定数です。
-2147483648 から +2147483647 の範囲内の実数に、整数型後置子「%」を付けると、小数点以下は切り捨てられ、整数型の定数とみなされます。

例：32767
　　-125
　　+10
　　3256.21% → 整数型後置子があるので、3256とみなされます。

#### 2進形式

2進数で表現される整数型定数です。
数値の先頭に「&B」を付け、0または1の並びで表します。
2進形式で整数型定数の数値の範囲を表記すると、
&B0～&B11111111111111111111111111111111
（32ビットの範囲内）となります。

例：&B110
　　&B0011

### 16進形式

16進数で表現される整数型定数です。
数値の先頭に「&H」を付け、0からFまでの並びで表します。
16進形式で整数型定数の数値の範囲を表記すると、
&H0～&HFFFFFFFF （32ビットの範囲内）となります。
例：&H100
　　&H3D5A

## [２] 単精度実数型定数

7桁の有効桁精度を持つ実数型の定数です。
値の範囲は-3.402823E+38～3.402823E+38です。
単精度実数型定数には、次の3つの表記形式があります。
●最後に、単精度実数型後置子、「!」を伴った数
●Eを使った指数形式
●上記指定がなく、7桁以下の実数
例：1256.3
　　35.78!
　　-9.345E-06

## [３] 倍精度実数型定数

15桁の有効桁精度を持つ実数型の定数です。
値の範囲は-1.79769313486231D+308～1.79769313486231D+308です。
倍精度実数型定数には、次の3つの表記形式があります。
●最後に、倍精度実数型後置子、「#」を伴った数
●Dを使った指数形式
●上記指定がなく、7桁を超え15桁以下の実数
例：1256.325468
　　35.78#
　　-9.345D-06

**注意：単精度実数型定数と倍精度実数型定数の有効桁数と内部表現により、誤差が生じる場合があります。**
**例**
F1=10.0025
D1=10.0025
**この場合、内部的には、1.0002499580383…E+1のように表現されており、単精度実数型に入れる場合、有効桁数の次の桁（8桁目）を四捨五入することにより、10.00250と正しく入ります。**
**倍精度実数型に入れる場合、有効桁数の次の桁（16桁目）を四捨五入することにより、10.002499580383…のような値が入ります。**

## 7.8.2　文字列定数

文字列型定数は、文字列を表す定数です。
文字列は、前後をダブルクォーテーション「”」で囲んで表記します。
文字列の長さは、243文字以内でなければなりません。

例：“PAC”

## 7.8.3　ベクトル型定数

ベクトル型定数は、X、Y、Z成分からなる、ベクトル表現の定数です。
X、Y、Zの各成分は、単精度実数で表されます。

例：ベクトル型変数V1に、ベクトル型定数を代入します。
　　V[1] = (1,0,0)

## 7.8.4　ポーズ定数

### [１]　ポジション型定数

ポジション型定数は、位置（X,Y,Z)、姿勢（RX,RY,RZ)、形態（FIG）の3つから構成されており、合計7つの単精度実数パラメータより構成されています。
位置X,Y,Zはmm単位で表現され、また、姿勢（RX,RY,RZ）は回転角度（°）で、形態（FIG）は番号（不定は-1）で表現されます。形態（FIG）の番号については、操作ガイドの「腕・ひじ・手首の形態について」と「腕の形態について」を参照してください。
姿勢は、X軸、Y軸、Z軸に対する回転角度を示し、それぞれ右ねじ方向を正回転としており、その範囲は-180° ～ +180° に限定されています。また、姿勢は軸の回転順序に依存するため、その回転順序は、X軸、Y軸、Z軸の順に回転するものと規定されています。

例：位置型変数P1に、ポジション型定数を代入します。
　　P[1] = (100,200,300,10,20,30,0)
　　‘X=100,Y=200,Z=300,RX=10,RY=20,RZ=30,FIG=0

## [2] ジョイント型定数

ジョイント型定数は、1軸から6軸までの各軸値によって構成されています。各軸値は、単精度実数で表されます。単位は角度（°）です。

例：軸型変数J1に、ジョイント型定数を代入します。
J[0] = (10,20,30,40,50,60) '1～6軸 :10°,20°,30°,40°,50°,60°

## [3] 同次変換型定数

同次変換型定数は、位置ベクトル、オリエントベクトル、アプローチベクトルの3つのベクトル、および形態で表されるポーズ定数です。
3つのベクトルは、それぞれ3つの単精度実数型パラメータで表されます。したがって、同次変換型定数は、10個の単精度実数型パラメータで表されます。
位置ベクトルの3つの要素の単位はmmです。

**注意：オリエントベクトル、アプローチベクトルはともに大きさ1の単位ベクトルであり、かつ互いに直交していなくてはなりません。もし、そうでないベクトルを指定し、動作命令でそれを使用した場合には、システムはアプローチベクトルを優先し、オリエントベクトルを自動的に直交させてから動作させます。**

例：同次変換型変数T1に、同次変換型定数を代入します。
T[1] = (10,20,30,1,0,0,0,1,0,4)

# 7.9　式と演算子

値を返すものを式といいます。定数や変数のように単独で値を持つ式と、演算子によって結合された複数の要素からなる式があります。PAC言語のすべてのデータ型の値が、式によって表されます。
この節の以下の部分で説明する演算子を用いると、式の中で演算を行なうことができます。

## 7.9.1　代入演算子

変数への代入は、代入文で、代入演算子「=」を使って行ないます。代入演算子の右側の値が、左側へ代入されます。
代入文では、データの種類によって次のコマンドを用います。

LET、LETA、LETF、LETJ、LETO、LETP、LETR、LETRX、LETRY、LETRZ、LETX、LETY、LETZ

数値変数の代入に用いるLETコマンドだけは、コマンドを省略して記述できるので、次の2つの式は同じ意味になります。

```
例：LET I1 = 5      'I1に5を代入します。
    I1 = 5          'I1に5を代入します。
```

## 7.9.2　算術演算子

算術演算では、下表に示す演算子を使います。
演算は、表に示す優先順位に従って行なわれます。

算術演算子

| 算術演算子 | 演算内容 | 演算の優先順位 |
|---|---|---|
| ^ | 指数（べき乗）演算 | 高 |
| - | 負符号 | ↕ |
| *、/ | 乗算、除算 | ↕ |
| MOD | 剰余 | ↕ |
| +、- | 加算、減算 | 低 |

注意①：0による除算を行なうと実行時にエラーとなります。
②：整数の加算・乗算で桁あふれが発生すると、あふれた部分は無視されます。エラーにはなりませんので注意してください。
例：I1 = 2147483647 + 1　'2147483647 は整数型変数の取り得る最大値
上の式を実行すると、結果は -2147483648 となります。
-2147483648 は、整数型変数の取り得る最小値です。

③：実数の加算・乗算で桁あふれが発生した場合や、整数型変数で記録できない値を実数型変数から代入しようとした場合には、エラーとなります。
④：整数どうしの除算では、その結果の値を超えない最大の整数に丸められます。
⑤：整数の剰余の符号は、下表のように決定されます。

算術演算子

| 除　数 / 被除数 | + | 0 | − |
|---|---|---|---|
| + | + | エラー | + |
| 0 | 0 | エラー | 0 |
| − | − | エラー | + |

## 7.9.3　関係演算子

関係演算子は、2つの数値を比較する場合に用います。結果はブール値（真「1」、偽「0」）で得られ、フロー制御文の条件式などで用います。

関係演算子

| 関係演算子 | 演算内容 |
|---|---|
| = | 等しい |
| =. | ほぼ等しい（近似比較） |
| <> | 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 |

備考：近似比較演算子（=.）の比較精度の設定は、次の手順で設定できます。
TP：[F1 プログラム]-[F6 補助機能]-[PRJ設定.]　No.12
WC：[プロジェクト]-[パラメータ]　プログラムタブ　No.13

## 7.9.4　論理演算子

論理演算子は、ビット演算を行ないます。
整数型以外の数値は、整数型に変換されてから演算されます。

| 注意：整数型の有効範囲を越えた値を指定した場合は実行時エラーになります。 |
|---|

論理演算子

| 論理演算子 | 演算内容 |
|---|---|
| NOT | 否定 |
| AND | 論理積 |
| OR | 論理和 |
| XOR | 排他的論理和 |

例：ビット演算

```
I1 = &B1100  XOR  &B0101
```

この場合の結果は、&B1001 となります。

注意：NOT 演算子について

NOT 演算子はビット演算を行ないますので式の評価結果を否定する場合には使用できません。

式の評価結果を否定できない例：

```
if NOT (I1=I2) then ...
```

I1=I2 が（真）のとき NOT では（偽）にできません。（真になります。）

式の評価結果を否定したい場合は以下の書式のように記述してください。

式の評価結果を否定できる例：

```
if (I1=I2) = FALSE then ...
```

## 7.9.5　文字列演算子

文字列は、文字列演算子「+」によって連結することができます。

例：A$ =“ABC”+“DEF”　　’A$は“ABCDEF”になります。

## 7.9.6 ベクトル演算

ベクトル演算では、下表に示す演算子を使います。
演算は、表に示す優先順位に従って行なわれます。

ベクトル演算子

| ベクトル演算子 | 演算内容 | 演算の優先順位 |
|---|---|---|
| .、*、x | 内積、スカラ倍、外積 | 高 |
| +、− | 加算、減算 | 低 |

例1：V1とV2の内積の算出
F1 = V1 . V2

例2：V1の2倍を算出
V2 = V1 * 2

例3：V1とV2の外積V3の算出
V3 = V1 x V2

**注意：外積演算子の x はアルファベットのエックスなので、その前後に空白を必ず入れてください。**

## 7.9.7 ポジション演算

ポジション型のデータに対して行なう演算です。
並進、回転偏差の演算には、「+」のみ使用できます。

例1：偏差の基準点P0から、ロボット座標系で(dx, dy, dz)だけ離れた点、P1の算出。(演算しない要素は、0を記述してください。)
P1 = P0 + (dx, dy, dz, 0, 0, 0)

例2：現在ポジションから、ツール座標系で(df, dg, dh)だけ離れた点、P2の算出。(演算しない要素は、0を記述してください。)
P2 = * + (df, dg, dh, 0, 0, 0)H

例3：基準点 P0 の姿勢を、ロボット座標系の X 軸周り、Y 軸周り、Z 軸周りにそれぞれ Rx（度)、Ry（度)、Rz（度）だけ回転させた点、P3 の算出。(演算しない要素は、0 を記述してください。)
P3 = P0 + (0, 0, 0, Rx, Ry, Rz)

例4：現在ポジションから、ツール座標系で（df, dg, dh）だけ離れ、ノーマルベクトル周りに Rx（度)、オリエントベクトル周りに Ry（度)、アプローチベクトル周りに Rz（度）回転した点、P4 の算出。(演算しない要素は、0 を記述してください。)
P4 = * + (df, dg, dh, Rx, Ry, Rz)H

**注意：偏差に、ポジション型変数は使用できません。**

## 7.9.8　ジョイント演算

ジョイント型のデータに対して行なう演算です。
並進偏差の演算には、「+」のみ使用できます。

例：偏差の基準点J0から、ロボット座標系で(d1, d2, d3, d4, d5, d6)だけ離れた点、J1の算出。(演算しない軸は、0を記述してください。)
J1 = J0 + (d1, d2, d3, d4, d5, d6)

注意：偏差に、ジョイント型変数は使用できません。

## 7.9.9　同次変換行列演算

同次変換行列に対して行なう演算です.
同次変換行列の積は、演算子「*」を使用します。

例：T1とT2の積、T3の算出。
T3 = T1 * T2

このとき、T3の形態はT2の形態に設定されます。

## 7.9.10　算術演算子、論理演算子、関係演算子の優先順位　[Ver.1.5以降]

算術演算子、論理演算子、関係演算子が１行に複数記述可能です。
演算子同士の優先順位は下記を参照してください。

| 演算子 | 演算内容 | 優先順位 |
|---|---|---|
| ^ | 指数（べき乗） | 高 |
| － | 負符号 | ↑ |
| ＊、／、 .、 x | 乗算、除算、内積、外積 | |
| ＭＯＤ | 剰余 | |
| ＋、－ | 加算、減算 | |
| ＮＯＴ | 否定 | |
| ＡＮＤ | 論理積 | |
| ＯＲ | 論理和 | |
| ＸＯＲ | 排他的論理和 | ↓ |
| ＝、＝.、＜＞、＜、＞、＜＝、＞＝ | 関係演算子 | 低 |

優先順位が同じ場合は式の左から右に評価されます。（ ）で閉じた場合は括弧内の演算が先に処理されます。(優先順位が高くなる)

**優先順位の例**

1. if IO128 = ON and IO129 = OFF then は

(1)

(2)

(3)

の順番で処理されます。

2. if (IO128 = ON) and (IO129 = OFF) then は

(1) (2)

(3)

の順番で処理されます。

**書式の例**

```
if (IO128＝ON) and (IO129＝OFF) then
        move  p,p 1
endif
```

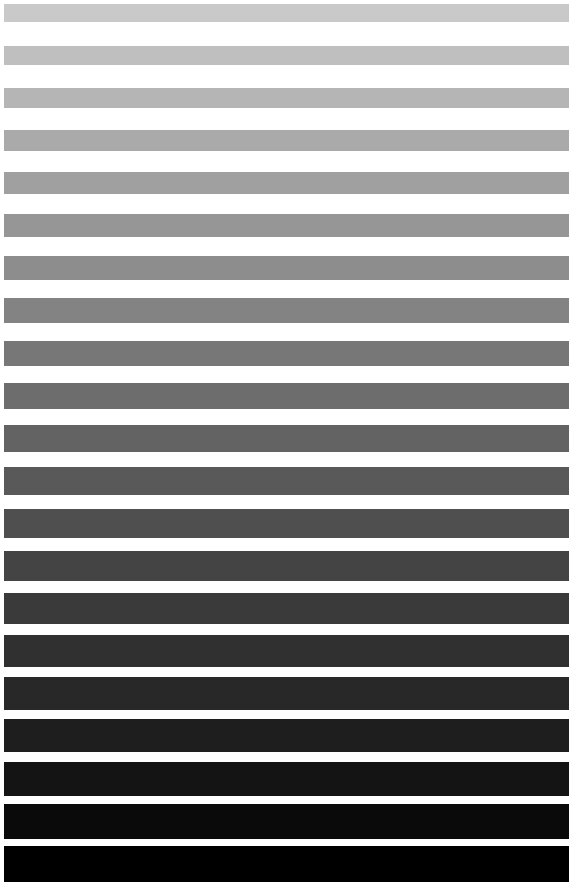
# 7.10　PAC 言語における単位の取り扱い

PAC言語における各物理量の表現単位は、下表に示すとおりです。

PAC言語における物理量の表現単位

| 物理量 | 単　位 |
|---|---|
| 長　さ | ミリメートル（mm） |
| 角　度 | 度（DEGREE） |
| 時　間 | ミリ秒（msec.） |
| 速　度 | %（最高速度に対する比率） |
| 加速度 | %（最大加速度に対する比率） |
| 減速度 | %（最大減速度に対する比率） |

# 第 8 章
# PAC 言語の文法



PAC言語によってプログラムを書く場合のきまりについて説明します。

# 第８章　PAC 言語の文法

## 8.1　文と行

PAC言語のプログラムは、複数の行から構成されます。
任意の行に、一つの文を記述できます。
1行の長さは、255バイトまでです。
文は、PAC言語による処理を記述する最小の単位であり、一つのコマンドで構成されます。
コマンドは、コマンドの名称と、コマンドに与える情報（パラメータ）から構成されます。

# 8.2　プログラム名と宣言

プログラム名や変数など、プログラムの実行時に必要な項目は、実行に先立って宣言します。
特にプログラム名の宣言を行なう場合は、プログラムの最初の有効行で宣言しなければなりません。この文をPROGRAM宣言文といいます。
PROGRAM宣言文は、省略することも可能です。省略すると、プログラムのファイル名がプログラム名として使われます。たとえば、ファイル名がPRO1.PACである場合、PROGRAM宣言文で別の名前を付けないと、PRO1というプログラム名になります。
PROGRAM宣言文を省略すると、プログラムを呼び出すときに引数を渡すことができません。また、プログラム編集中にプログラム名がわかりにくくなります。特別な理由がない限り、PROGRAM宣言文を省略しないようにしてください。
プログラムの呼び出しについては、「2.1　プログラムの呼び出しとサブルーチン」を参照してください。

注意①：ティーチングペンダントからの操作で起動できるプログラムは、引数を持たないプログラムに限られます。また、外部機器からの呼び出しで起動できるプログラムは、プログラム名が「PRO<番号>」の形式のものに限られます。ティーチングペンダントや外部機器からの呼び出しで起動できないプログラムは、他のプログラムから呼び出すことができます。
②：「PRO<番号>」の形式のプログラム名を持つプログラムを呼び出すときには、引数を渡すことができません。
③：プログラム名の先頭 3 文字を「PRO」とした場合、必ず番号を付けなければなりません。
<番号>は数字として解釈しますので、PRO01 や PRO0001 のように記述しても PRO1 として扱われます。

# 8.3 ラベル

プログラムの分岐先や、プログラム中での文の位置を表すために、ラベルを使うことができます。
ラベルを使用するには、次の規則があります。

●ラベルの名前は、アスタリスク（＊）で始まります。
●ラベルの名前の2文字目は、任意のアルファベットでなくてはなりません。
●ラベルの名前の3文字目以後は、英字と数字の任意の組み合わせが使えます。
●ラベルの名前の最後の文字は、コロン（：）でなければなりません。
●予約語を、ラベル名に使用することはできません。
●参照されるラベル名は、行の最初になければなりません。
●参照されるラベル名に、同じものが複数ある場合には、エラーとなります。
●ラベルを参照できる範囲は、そのラベルが存在するプログラムの中だけです。

```
PROGRAM WITH_LABEL
   IF IO138=1  THEN  *ACTION  ELSE  *NOACTION
   *ACTION:          SPEED 100
                     MOVE P, P1
                     DELAY 200
                     MOVE P, P2
   *NOACTION:        DELAY 200
END
```

ラベルを使ったプログラム例

# 8.4　文字セット

PAC言語で使用できる文字は、英文字（アルファベット）、数字、特殊記号です。
英文字は、大文字と小文字の区別はしません。
特殊記号は、算術演算子（ +, -, *, / ）のほかに、次のようなものがあります。

●コンマ（ , ）
　パラメータが並ぶ場合の区切りに使用します。
●セミコロン（ ; ）
　コマンドの引数にパラメータが並ぶ場合の区切りに使用します。
●シングルクォート（ ’ ）
　REMコマンドの代用として使用します。
●ダブルクォーテーション（ ” ）
　任意の文字列の前後を囲むことで、文字列データの指定に使用します。
●アスタリスク（ * ）
　ラベル名の先頭で使用します。
　インターフェイス座標系の現在位置を示します。
●スペース
　命令の名前の前後に必ず必要です。

注意：REM文のコメントと文字列定数には、漢字コード（Shift JIS）を含むことができます。

## 8.5　予約語

コマンドの名前や演算子など、PAC言語が処理を行なう上で、使用方法を決めているものを予約語といいます。
予約語を使用するには、予約語を識別するために、語の前、後ろ、あるいはその両方に、文法上規定された特殊文字（スペース文字、”、#、:）を挿入する必要があります。
PAC言語の予約語は、付録５「予約語一覧」に掲げてあります。
変数名やラベルに、予約語そのものを使うことはできませんが、予約語を含んだ名前は使用できます。たとえば、「HOME」という語は予約語ですが、「HOME1」という名前を、変数名またはラベルなどに使用することは可能です。

# 8.6　宣言文

変数や定数、関数など、名前や型を決めておく必要のあるものは、プログラム中で使用する前に、宣言文で定義を宣言します。
宣言文は、大きく分けて、次の3種類があります。

●型宣言　　：変数および定数の型を宣言します。
●関数／プログラム宣言：関数やプログラムを宣言します。
●ユーザ座標系宣言　　：ユーザ座標系の宣言をします。

## 8.6.1　型宣言

### [1]　型宣言文字（後置子）

決められた後置子を使って、変数の型を宣言することができます。後置子は次のとおりです。

型宣言文字（後置子）

| 種　類 | 後置子 | 使用例 |
|---|---|---|
| 整数型後置子 | % | A% |
| 単精度実数型後置子 | ! | A! |
| 倍精度実数型後置子 | # | A# |
| 文字列型後置子 | $ | A$ |

注意：プログラムの中では、変数の型は最初に宣言された型で扱われます。プログラムの途中で、宣言と異なる型の後置子を使っても、型の識別は行ないません。
A、A%、A!、A#、A$は、すべて同一の変数として認識され、最初に使用した型で登録されます。したがって、はじめに使用した型と異なる型の後置子を使用するとエラーになります。

## [2] 型宣言命令

次の型宣言のコマンドを使って、変数の型を宣言することができます。

**型宣言命令**

| 種　類 | コマンド | 使用例 |
|---|---|---|
| 整数型 | DEFINT | DEFINT AA,AB |
| 単精度実数型 | DEFSNG | DEFSNG BA,BB |
| 倍精度実数型 | DEFDBL | DEFDBL CA,CB |
| 文字列型 | DEFSTR | DEFSTR DA,DB |
| ベクトル型 | DEFVEC | DEFVEC EA,EB |
| ポジション型 | DEFPOS | DEFPOS FA,FB |
| ジョイント型 | DEFJNT | DEFJNT GA,GB |
| 同次変換型 | DEFTRN | DEFTRN HA,HB |

これらのコマンドでは、型を宣言するのと同時に、変数の初期化をすることができます。
これらのコマンドによる宣言文の中では、変数名に後置子を付けることはできません。

使用例：

```
DEFINT AA = 1       'AAを整数型とし1を代入します。
DEFSNG BB(10)       'BBを要素数10の単精度実数型にします。
```

悪い例：

```
DEFINT AB%          '後置子を使っているのでエラーになります。
```

## [３] 配列宣言

配列宣言のための宣言文では、変数名に後置子を付けることで、I/O変数以外のすべての型の配列を作ることができます。
ただし、配列の初期化は、宣言時にはできません。
配列の添字は0以上でなくてはなりません。配列の次元は3次元までです。
配列の要素の総数は32767を上限とします。

配列宣言

| 種　類 | コマンド | 使用例 |
|---|---|---|
| 配列宣言 | DIM | DIM AA(10,10) |

配列宣言の例：

```
DIM CC(3,3,3) AS INTEGER  'CCを整数型の3次元配列とし
                          'ます。
```

上記の例は、DEFINTコマンドを使って、次のように表現することもできます。

```
DEFINT CC(3,3,3)      'CCを整数型の3次元配列とします。
```

**注意：DEF???文とDIM文との違い**

**DIM文は、配列でない場合にも使用できます。たとえば、**

```
DEFINT  CC
```

**の代わりに、**

```
DIM  CC  AS  INTEGER
```

**というようになります。DIM文は、DEF???文よりも汎用的ですが、型宣言と同時に変数を初期化することはできません。たとえば、次のような表現はDIM文ではできません。**

```
DEFINT  CC = 1    'CCを整数型に宣言し初期値を1とします。
```

**宣言と同時に初期化をするのでなければ、すべての型や配列を同様に扱えるので、DIM文を使うことを推奨します。**

## [４] I/O 変数宣言

変数名と特定のI/Oポートを対応付けます。

I/O変数宣言

| 種　類 | コマンド | 使用例 |
|---|---|---|
| I/O変数宣言 | DEFIO | DEFIO PORT = BYTE,104 |

## 8.6.2 関数／プログラム宣言

関数またはプログラム名を宣言するコマンドは次のとおりです。

関数／プログラム宣言

| 種　類 | コマンド | 使用例 |
|---|---|---|
| 関数宣言 | DEF FN | DEF FN AREA(R) = PI * R * R |
| プログラム宣言 | PROGRAM | PROGRAM PRO1 |

## 8.6.3 ユーザ座標系の宣言

次に示すコマンドは、ユーザ座標系を宣言します。

ユーザ座標系の宣言

| 種　類 | コマンド | 使用例 |
|---|---|---|
| 干渉エリアの宣言 | AREA | AREA 1,P0,V0,15,6 |
| ツール座標系宣言 | TOOL | TOOL 1,P0 |
| ワーク座標系宣言 | WORK | WORK 1,P0 |

## 8.6.4 フォルダー変数宣言／EXTERN 変数宣言　[RC7_Ver.2.2 以降]

FOLDERステートメントでローカル変数を宣言すると、ローカル変数を外部のプログラムからEXTERNステートメントを使用して参照可能となります。
フォルダー変数とEXTERN変数の宣言をするコマンドを以下に示します。

フォルダー機能の宣言

| 種　類 | コマンド |
|---|---|
| フォルダー変数宣言 | FOLDER DEFINT |
| | FOLDER DEFSNG |
| | FOLDER DEFDBL |
| | FOLDER DEFSTR |
| | FOLDER DEFVEC |
| | FOLDER DEFPOS |
| | FOLDER DEFJNT |
| | FOLDER DEFTRN |
| | FOLDER DIM |
| EXTERN変数宣言 | EXTERN DEFINT |
| | EXTERN DEFSNG |
| | EXTERN DEFDBL |
| | EXTERN DEFSTR |
| | EXTERN DEFVEC |
| | EXTERN DEFPOS |
| | EXTERN DEFJNT |
| | EXTERN DEFTRN |
| | EXTERN DIM |

# 8.7　代入文

代入文は、各型の変数に値を設定します。
数値代入文、文字列代入文、ベクトル代入文、ポーズ代入文の4つに大別されます。

注意：すべての代入文でLETコマンドは省略することができます。

## 8.7.1　数値代入文

数値代入文では、数値変数に値を代入します。

例：LET I[1] = 10　　'I[1]に10を代入します。
　　D[2] = 3.14　　'D[2]に3.14を代入します。

## 8.7.2　文字列代入文

文字列代入文は、文字列型変数に文字列を代入します。

例：S[2] = “DENSO”　'S[2]に“DENSO”を代入します。

　　LET SS$ = S[2]　'SS$にS[2]の値を代入します。

## 8.7.3　ベクトル代入文

ベクトル代入文は、ベクトル型変数に値を代入します。次の4つのコマンドが使えます。

ベクトル代入文

| 種　類 | コマンド | 使用例 |
|---|---|---|
| ベクトル型代入 | LET | LET V[2] = (1,2,3) |
| X成分代入 | LETX | LETX V[2] = F1 |
| Y成分代入 | LETY | LETY V[2] = F1 |
| Z成分代入 | LETZ | LETZ V[2] = F1 |

## 8.7.4 ポーズ代入文

ポーズ代入文には、ポジション代入文、ジョイント代入文、同次変換代入文、ホームポジション代入文の4つがあります。

### [1] ポジション代入文

ポジション代入文は、ポジション型変数に値を代入します。次のコマンドが使えます。

ポジション代入文

| 種　類 | コマンド | 使用例 |
|---|---|---|
| ポジション型代入 | LET | LET P[3] = (10,10,10,0,0,0) |
| 位置ベクトル代入 | LETP | LETP P3 = V1 |
| 回転ベクトル代入 | LETR | LETR P3 = V2 |
| X成分代入 | LETX | LETX P[3] = F1 |
| Y成分代入 | LETY | LETY P[3] = F1 |
| Z成分代入 | LETZ | LETZ P[3] = F1 |
| RX成分代入 | LETRX | LETRX P[3] = F1 |
| RY成分代入 | LETRY | LETRY P[3] = F1 |
| RZ成分代入 | LETRZ | LETRZ P[3] = F1 |
| 形態成分代入 | LETF | LETF P[3] = I1 |

### [2] ジョイント代入文

ジョイント代入文は、ジョイント型変数に値を代入します。次のコマンドが使えます。

ジョイント代入文

| 種　類 | コマンド | 使用例 |
|---|---|---|
| ジョイント型代入 | LET | LET J[4] = (1,2,3,4,5,6) 6軸 |
| 軸成分代入 | LETJ | LETJ 1,J[4] = F1 |

## [３] 同次変換代入文

同次変換代入文は、同次変換型変数に値を代入します。次のコマンドが使えます。

同次変換代入文

| 種　類 | コマンド | 使用例 |
|---|---|---|
| 同次変換型代入 | LET | LET T[x]=(1,2,3,4,5,6,7,8,9,4) |
| 位置ベクトル代入 | LETP | LETP T[x] = V1 |
| オリエントベクトル代入 | LETO | LETO T[x] = V[1] |
| アプローチベクトル代入 | LETA | LETA T[x] = V2 |
| 形態成分代入 | LETF | LETF T[x] = I1 |

## [４] ホームポジション代入文

ホームポジション代入文は、任意の座標値をホームポジションとします。

ホームポジション代入文

| 種　類 | コマンド | 使用例 |
|---|---|---|
| ホームポジション代入 | HOME | HOME *またはHOME CURPOS<br>現在位置をホームポジションにします。 |

# 8.8　フロー制御文

プログラムの各文の実行順序を制御するために、フロー制御文を使います。
フロー制御では、プログラム中の文の位置を指し示すために、ラベルを使用します。
フロー制御は、無条件分岐、条件分岐、選択、繰り返し、定義済み処理呼び出しの5つに大別されます。

## 8.8.1　無条件分岐

プログラムの実行を、任意の場所へ移す場合に使用します。GOTOコマンドに続けて、次に実行したい文のラベルを記述します。

例：GOTO　*LABEL1

## 8.8.2　条件分岐

IF～THEN～ELSE文またはIF～ENDIF文を使い、指定した条件が成立するかどうかにより、分岐先が決められます。
IFの直後に記述した関係式の値が真（TRUE(1)）であればTHEN以降を、そうでなければELSE以降を実行します。

## 8.8.3　選択

指定した式の値によって、実行する処理を選びます。3つのコマンドがあります。

SELECT CASE文では、SELECT行のCASE以降に算術式を置き、この算術式を満たす値を持つCASE行以降を、次のCASE行またはEND SELECT行まで実行します。いずれのCASE行も等しくない場合は、CASE ELSE以降をEND SELECT行まで実行します。

ON～GOSUB文では、ON以降に算術式を置き、算術式の値に従って、サブルーチンへ実行を移します。

ON～GOTO文では、ON以降に算術式を置き、算術式の値に従って、ラベル名へ実行を移します。

## 8.8.4　繰り返し

指定した条件によって、繰り返すかどうかを制御します。4つのコマンドがあります。

FOR～NEXT文では、FOR行のFOR以降に繰り返し条件を置き、この条件が満たされるまで、対応するNEXT行までの間の処理を繰り返します。

DO～LOOP文では、WHILEまたはUNTIL以降に関係式を置き，これが満たされている間（WHILEの場合）あるいは、満たされるまで（UNTILの場合)、DO行からLOOP行までの処理を繰り返します。

WHILE～WEND文では、WHILE以降に関係式を置き、これが満たされている間、WHILE行からWEND行までの処理を繰り返します。DO WHILE～LOOP文と同じ効果を持ちます。

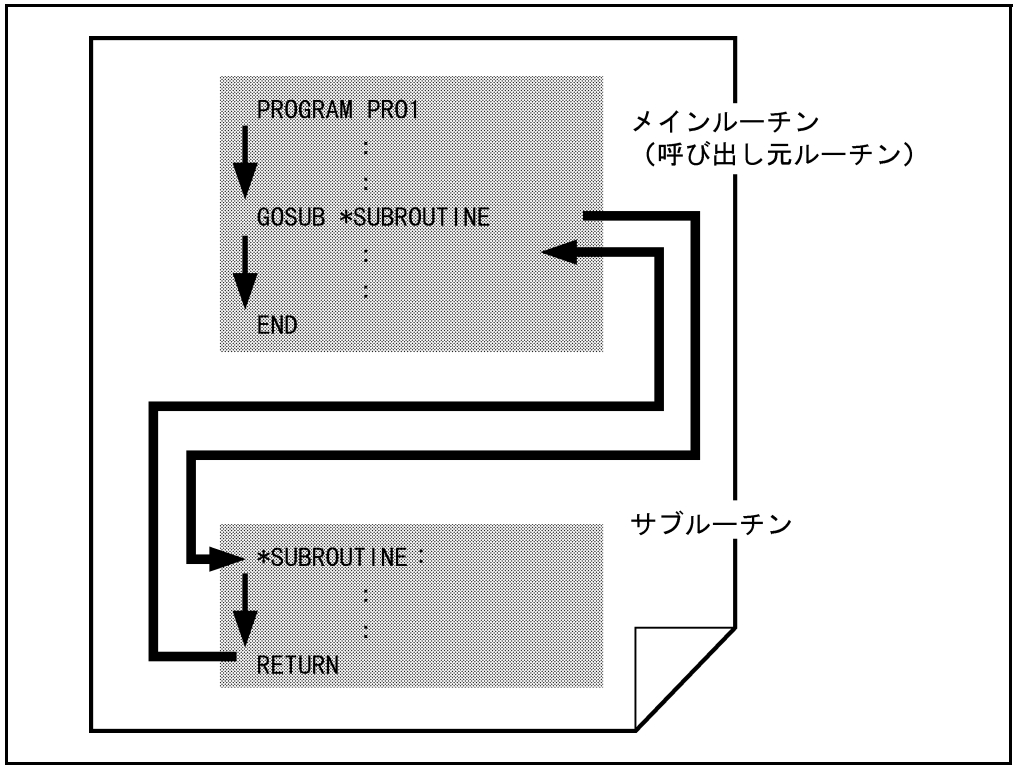
REPEAT～UNTIL文では、UNTIL以降に関係式を置き、これが満たされるまで、REPEAT行からUNTIL行までの処理を繰り返します。DO～LOOP UNTIL文と同じ効果を持ちます。

## 8.8.5　定義済み処理呼び出し

プログラムの中で、特定の動作を繰り返す部分を、別に書き出しておいて、必要に応じて呼び出すことができます。これを、定義済み処理呼び出しといいます。
定義済み処理呼び出しには、サブルーチンの呼び出しと、プログラムの呼び出しの2種類があります。

### [1]　サブルーチン

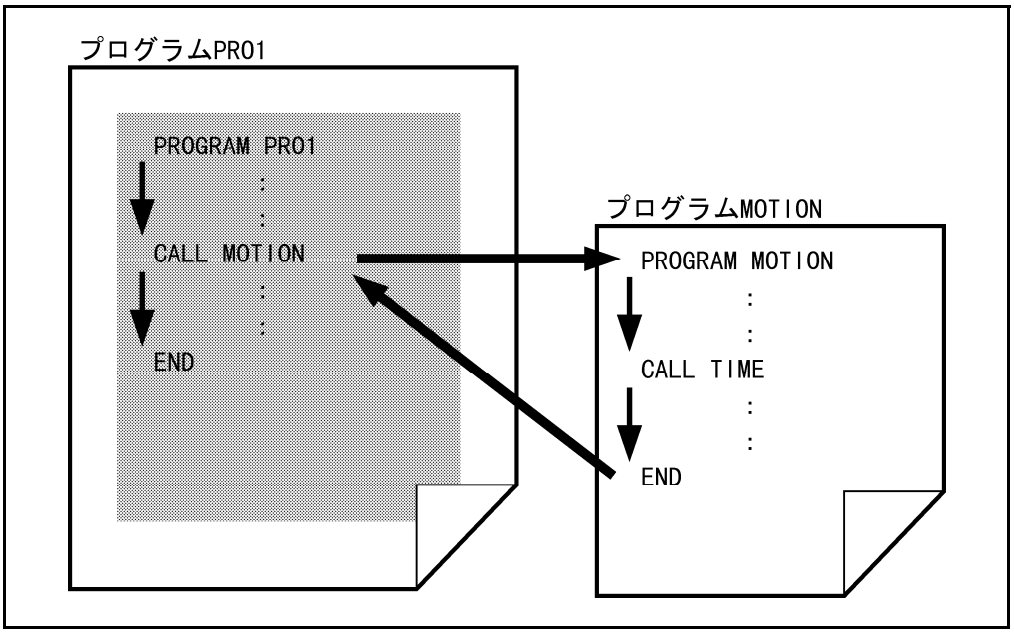
サブルーチンを呼び出すには、GOSUB文で飛び先を、ラベルで指定します。
サブルーチンは、呼び出すプログラムと同じファイルの中に書かれている必要があります。サブルーチンの最後の行にはRETURN文があり、一連の処理を実行した後にRETURN文を実行することにより、サブルーチンを呼び出したプログラムの元の行の次へ、制御を戻します。




サブルーチンの呼び出し構造

## [2] プログラム

プログラムを呼び出すには、CALL文でプログラム名を指定して行ないます。再帰呼び出しも行なえます。
詳しくは、「2.1　プログラムの呼び出しとサブルーチン」を参照してください。




プログラムの呼び出し構造

# 8.9　ロボット制御文

ロボット制御文には大きく分けて、動作制御文、形態制御文、停止制御文、速度制御文、時間制御文、座標変換文があります。

## 8.9.1　動作制御文

ロボットの動作を制御する文を、動作制御文といいます。
ロボットアームの動作制御文、ハンド制御文、モータ制御文があります。

### [1]　ロボットアームの制御

ロボットアームの動作を制御する文は、目標ポーズが、P型の場合、J型の場合、T型の場合、P/J/T型以外のデータの場合の4種類があります。

#### 目標ポーズがP、J、T型データ

動作制御コマンド（目標ポーズがP、J、T型データ）

| 動作の種類 | コマンド |
|---|---|
| 一般移動 | MOVE |
| アプローチ移動 | APPROACH |
| デパート移動 | DEPART |

#### 目標ポーズがP、J、T型以外のデータ

動作制御コマンド（目標ポーズがP、J、T型以外のデータ）

| 動作の種類 | コマンド |
|---|---|
| 回転移動 | ROTATE |
| アプローチ方向回転移動 | ROTATEH |
| 並進移動 | DRAW |
| 各軸相対動作 | DRIVE |
| 各軸絶対動作 | DRIVEA |
| ホームポジション移動 | GOHOME |

## 8.9.2　停止制御文

停止制御文は、プログラムの実行を停止または終了させます。

停止制御コマンド

| 動作の種類 | コマンド | 備　考 |
|---|---|---|
| 実行一時停止 | HOLD | ステップ停止 |
| 実行停止 | HALT | 瞬時停止。自タスクをSUSPENDします |
| 実行終了 | STOP | 自タスクをKILLします |
| 動作の中断 | INTERRUPT | 割り込みにより瞬時停止します |

## 8.9.3　速度制御文

速度制御文は、アームの移動速度と加速度、減速度を設定します。

速度制御コマンド

| 動作の種類 | コマンド |
|---|---|
| 手先移動速度指定 | SPEED |
| 軸移動速度指定 | JSPEED |
| 手先加速度指定 | ACCEL |
| 軸加速度指定 | JACCEL |
| 手先減速度指定 | DECEL |
| 軸減速度指定 | JDECEL |

## 8.9.4　時間制御文

時間制御文は、時間によるロボットの動作制御をします。

時間制御コマンド

| 動作の種類 | コマンド |
|---|---|
| 指定時間停止 | DELAY |
| 条件停止 | WAIT |

## 8.9.5　座標変換文

座標変換文は、座標系の切り替えをします。

座標変換コマンド

| 動作の種類 | コマンド |
|---|---|
| ツール座標系切り替え | CHANGETOOL |
| ワーク座標系切り替え | CHANGEWORK |
| 干渉チェックエリアの有効化 | SETAREA |
| 干渉チェックエリアの無効化 | RESETAREA |

# 8.10　入出力制御文

入出力制御文には、DI/DO制御文、RS232C制御文、ティーチングペンダント制御文の3つがあります。

## 8.10.1　DI/DO 制御文

DI/DO制御文は、I/Oポートの入出力を制御します。

DI/DO制御コマンド

| 動作の種類 | コマンド |
|---|---|
| データ読み込み | IN |
| データ書き込み | OUT |
| I/Oポート ON | SET |
| I/Oポート OFF | RESET |
| 非動作命令並列処理 | IOBLOCK |

## 8.10.2　RS232C 制御文

RS232C制御文は、RS232CおよびEthernetポートの入出力を制御します。

RS232C制御コマンド

| 動作の種類 | コマンド |
|---|---|
| データ書き込み | PRINT |
| データ読み込み | INPUT |
| データ書き込み | WRITE |
| バッファクリア | FLUSH |

## 8.10.3　ティーチングペンダント制御文

ティーチングペンダント制御文は、ティーチングペンダントの入出力の設定をします。

ティーチングペンダント制御コマンド

| 動作の種類 | コマンド |
|---|---|
| デバッグ画面出力 | PRINTDBG |
| 各個操作ボタン定義 | PRINTLBL |
| メッセージ画面出力 | PRINTMSG |
| ブザー鳴動 | BUZZER |
| TP簡易操作盤定義［Ver.1.5以降］ | set_button<br>set_page<br>change_bCap<br>change_pCap<br>disp_page |

# 8.11　マルチタスク制御文

マルチタスク制御文には、タスク制御文とセマフォ制御文があります。

## 8.11.1　タスク制御文

タスク制御文は、そのタスク制御文が属するタスク以外のタスクを制御します。

タスク制御コマンド

| 動作の種類 | コマンド |
|---|---|
| タスクの生成・起動 | RUN |
| タスクの中断 | SUSPEND |
| タスクの削除 | KILL |
| タスクの保護 | DEFEND |

## 8.11.2　セマフォ制御文

セマフォ制御文は、セマフォに関わる制御を行ないます。

セマフォ制御コマンド

| 動作の種類 | コマンド |
|---|---|
| セマフォの生成 | CREATESEM |
| セマフォの削除 | DELETESEM |
| セマフォの解放 | GIVESEM |
| セマフォの取得 | TAKESEM |
| セマフォ待ちタスクの解放 | FLUSHSEM |

## 8.11.3　専用セマフォ制御文

専用セマフォ制御文

| 動作の種類 | コマンド |
|---|---|
| アームセマフォの解放 | GIVEARM |
| アームセマフォの取得 | TAKEARM |
| 視覚セマフォの解放 | GIVEVIS |
| 視覚セマフォの取得 | TAKEVIS |

# 8.12　時刻・日付制御

時刻日付制御文は、時刻と日付、および経過時間を取得することと、時刻による割り込みの制御を行ないます。

**時刻日付制御コマンド**

| 動作の種類 | コマンド |
|---|---|
| 現在の日付を取得 | DATE$ |
| 現在の時刻を取得 | TIME$ |
| 経過時間を取得 | TIMER |

# 8.13　エラー制御

## 8.13.1　エラー制御コマンド

エラー制御文は、エラーによる割り込みを制御します。

エラー制御コマンド

| 動作の種類 | コマンド |
|---|---|
| エラー割り込み定義 | ON ERROR GOTO |
| エラーコード | ERR |
| エラー行 | ERL |
| エラー復帰 | RESUME |

## 8.13.2　エラー格納機能 [Ver.1.98 以降]

### 8.13.2.1　エラー格納機能とは

エラー発生時、そのエラーコードをI型変数領域（リングバッファとして使用）に格納する機能です。
Ver. 1.98で追加されたSETERR命令、GETERR命令を使用することで、リングバッファに書き込むエラーおよびリングバッファから読み出すエラーを定義できます。
エラー格納機能を使用するには、ティーチングペンダントから拡張機能を使って機能追加し、設定をしておく必要があります。

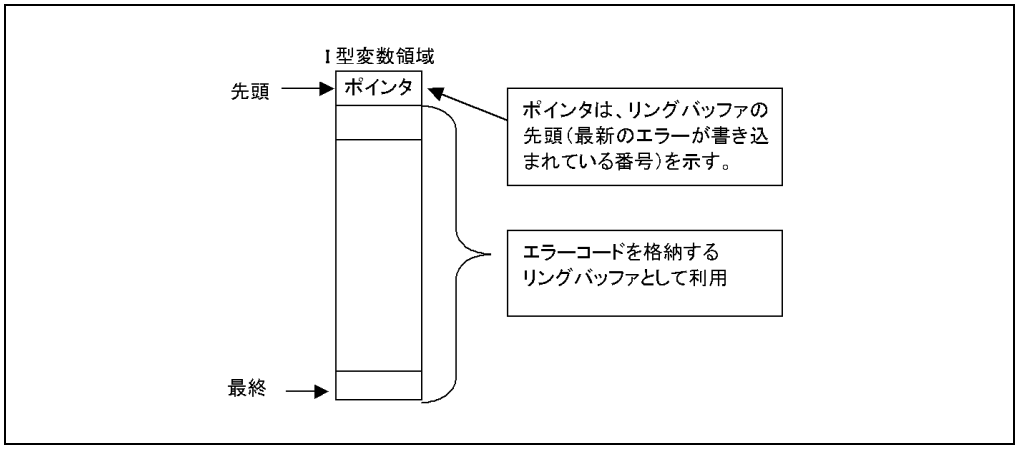
### 8.13.2.2　エラー格納機能の設定項目とリングバッファについて

#### 設定項目

・機能使用／未使用の設定
・エラーを格納する領域として使うI型変数の先頭番号と最終番号の設定

#### リングバッファについて

リングバッファにおけるポインタ位置、Ｉ型変数の先頭番号と最終番号については下図を参照してください。
注：Ｉ型変数領域の先頭にはポインタが格納されるため、実際のリングバッファは、Ｉ型変数領域の先頭番号+1から始まります。



## 8.13.2.3　エラー格納機能の設定方法

(1) ティーチングペンダントの機能拡張画面を表示します。
**操作経路：[F6 設定]－[F7 オプション]－[F8 機能拡張]**

(2) [F5 機能追加]を押し、暗証番号入力画面に「3237」を入力します。



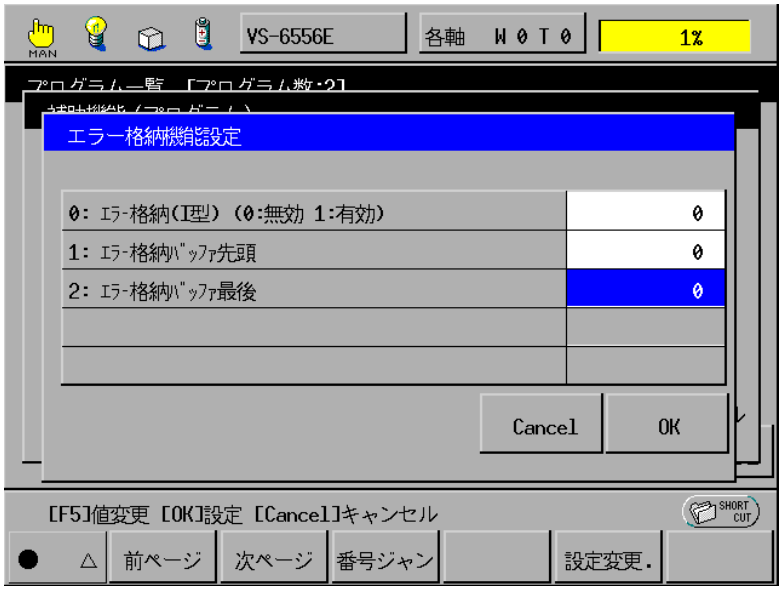

(3) [OK]を押すと、エラー格納機能が追加されます。



(4) 補助設定画面を表示させて、[F11 エラー格納]ボタンを押します。
**操作経路：[F1 プログラム]－[F6 補助機能]－[F11 エラー格納]**




(5) エラー格納機能設定画面が表されます。




ここで、エラー格納機能の設定を行ないます。

| 項目 | 設定内容 |
|---|---|
| 0:エラー格納 | エラー格納機能を使用するならば"1"を設定 |
| 1:エラー格納ﾊﾞｯﾌｧ先頭 | エラー格納領域の先頭となるI型変数番号を設定 |
| 2:エラー格納ﾊﾞｯﾌｧ最後 | エラー格納領域の最後となるI型変数番号を設定 |

## 8.13.2.4　エラー格納機能関連のコマンド

SETERR、GETERR、CLRERR、GETERRLVLについては、「第18章 エラー制御」を参照してください。

# 8.14　システム情報

システム情報は、次に示すコマンドによって得られます。

システム情報コマンド

| 動作の種類 | コマンド |
|---|---|
| システムの環境設定値の取得 | GETENV |
| システムの環境設定値の代入 | LETENV |
| ROMバージョン情報の取得 | VER$ |
| プログラムの状態を取得 | STATUS |

# 8.15　プリプロセッサ

プリプロセッサ文は、プログラムを実行形式に翻訳（コンパイル）する際に、文字列の置き換え、またはファイルの取り込みを制御します。

プリプロセッサコマンド

| 動作の種類 | コマンド |
|---|---|
| 定数、マクロ名を文字列に置換 | #define |
| #define文のキャンセル | #undef |
| ファイルの取り込み | #include |
| 擬似的なエラーの発生 | #error |
| プログラムの最適化を指定 | #pragma optimize |

# 8.16　値による呼び出しと参照による呼び出し

プログラムから別のプログラムを呼び出すときに、プログラムにデータを渡したい場合があります。この場合、呼び出すプログラム名に続けて引数リストを付けることにより、データを渡すことができます。引数の渡し方には、直接に値を渡す「値による呼び出し」と、変数を渡す「参照による呼び出し」があります。

プログラムを呼び出す命令には、CALLとRUNがあります。CALLコマンドでは、両方の呼び出し方法が使えますが、RUNコマンドでは、「値による呼び出し」しか使えません。

プログラムに渡す引数の型は、呼び出される側のプログラムで記述した引数の型に合わせなければなりません。したがって、定数を渡すときは、型宣言文字を付ける必要があります。また、呼び出される側のプログラムの中では、引数はローカル変数として宣言されるため、引数名に型宣言文字を付ける必要があります。

## 8.16.1　値による呼び出し

定数と数式と文字式は、必ず値で渡します。変数もカッコ（ ）で囲めば式とみなされ、値で渡すことができます。

例：プログラム PROGRAM SUB1(AA#)を呼び出す方法

●変数を値で渡す場合　　CALL SUB1((D1))
●定数を渡す場合　　CALL SUB1(10#)
●数式を渡す場合　　CALL SUB1(D1 + D2)

**注意：① 変数を値で渡す場合、配列全体の値による呼び出しを行うことはできません。配列全体の値による呼び出しを行なった場合は、配列全体の参照による呼び出しとして処理されます。**
**② 数式を渡す場合、式の結果の型に注意します。呼び出される側のプログラムの引数の型と同じ型にならなければなりません。**

## 8.16.2　参照による呼び出し

グローバル変数およびローカル変数を、引数として渡せます。
引数として配列全体を指定したいときは、配列名にカッコ（ ）を付けます。
グローバル変数、ローカル配列の１要素は、参照で渡すことはできません。グローバル変数、ローカル配列の１要素を参照で渡したい場合は、いったんローカル変数に値を代入し、そのローカル変数を介して参照呼び出しを行なってください。

例1：プログラム PROGRAM SUB1(AA#)を呼び出す場合

●ローカル変数を渡す場合1（DEFDBLでDDが宣言済みのとき）
```
CALL SUB1(DD#)
```
●ローカル変数を渡す場合2（DEFDBLでDAが宣言済みのとき）
```
CALL SUB1(DA)
```

**注意：ローカル変数は、あらかじめ値を代入しておく必要があります。次の場合は、値で渡すことになります。**
```
CALL SUB1(D1)
```

例2：プログラム PROGRAM SUB2(BB%(10))を呼び出す場合

●配列全体を渡す場合（DIMでAB%(10)が宣言済みのとき）
```
CALL SUB2(AB%())
```

# 8.17　視覚制御

## 8.17.1　画像入出力

次に示すコマンドは、カメラの映像とメモリ内の画像データの入出力を制御します。

画像入出力制御コマンド

| 動作の種類 | コマンド |
|---|---|
| カメラ映像をメモリに格納 | CAMIN |
| カメラ映像格納の機能設定 | CAMMODE |
| カメラ映像入力レベルの設定 | CAMLEVEL |
| カメラ映像をモニタに表示 | VISCAMOUT |
| メモリの画像をモニタに表示 | VISPLNOUT |
| 描画画面情報をモニタに表示 | VISOVERLAY |
| ルックアップテーブルのデータ設定 | VISDEFTABLE |
| ルックアップテーブルのデータ参照 | VISREFTABLE |

## 8.17.2　ウィンドウ設定

次に示すコマンドは、ウィンドウ（画像処理を行なう範囲）の編集を行ないます。

ウィンドウ設定コマンド

| 動作の種類 | コマンド |
|---|---|
| ウィンドウ情報の設定 | WINDMAKE |
| ウィンドウ情報のクリア | WINDCLR |
| ウィンドウ情報のコピー | WINDCOPY |
| ウィンドウ情報の参照 | WIMDREF |
| ウィンドウの描画 | WINDDISP |

## 8.17.3　描画

次に示すコマンドは、格納メモリ(処理画面)、オーバレイメモリ(描画専用画面)への描画動作を制御します。

描画コマンド

| 動作の種類 | コマンド |
|---|---|
| 描画先画面の指定 | VISSCREEN |
| 描画時の輝度指定 | VISBRIGHT |
| 画面の消去 | VISCLS |
| 点の描画 | VISPUTP |
| 長さ、角度指定による直線描画 | VISLINE |
| 2点を結ぶ直線の描画 | VISPTP |
| 矩形の描画 | VISRECT |
| 円の描画 | VISCIRCLE |
| 楕円の描画 | VISELLIPSE |
| 扇形の描画 | VISSECT |
| 十字マークの描画 | VISCROSS |
| 文字の表示位置指定 | VISLOC |
| 描画文字の設定 | VISDEFCHAR |
| 文字の表示 | VISPRINT |

## 8.17.4　画像処理

次に示すコマンドは、画像データの処理を行ないます。

画像処理コマンド

| 動作の種類 | コマンド |
|---|---|
| 処理対象画面の指定 | VISWORKPLN |
| 指定座標の輝度取得 | VISGETP |
| ヒストグラムの計測 | VISHIST |
| ヒストグラム結果の参照 | VISREFHIST |
| 2値化レベルの算出 | VISLEVEL |
| 2値化処理 | VISBINA |
| 2値化表示 | VISBINAR |
| フィルタ処理 | VISFILTER |
| 画像間演算 | VISMASK |
| 画面のコピー | VISCOPY |
| 面積、重心、主軸角の計測 | VISMEASURE |
| 投影データの計測 | VISPROJ |
| エッジの計測 | VISEDGE |

## 8.17.5 コード認識

次に示すコマンドは、QRコードの読み取りを行ないます。

コード認識コマンド

| 動作の種類 | コマンド |
|---|---|
| QRコードの読み取り | VISREADQR |

## 8.17.6 ラベリング

次に示すコマンドは、ラベルの処理を行ないます。

ラベリングコマンド

| 動作の種類 | コマンド |
|---|---|
| ラベリングの実行 | BLOB |
| ラベル番号別特徴計測 | BLOBMEASURE |
| ラベル番号の取得 | BLOBLABEL |
| ラベル画像のコピー | BLOBCOPY |

## 8.17.7 サーチ機能

次に示すコマンドは、画像モデルの登録と検索を行ないます。

サーチ機能コマンド

| 動作の種類 | コマンド |
|---|---|
| サーチモデルの登録 | SHDEFMODEL |
| 登録モデルの情報取得 | SHREFMODEL |
| 登録モデルのコピー | SHCOPYMODEL |
| 登録モデルの消去 | SHCLRMODEL |
| 登録モデルの表示 | SHDISPMODEL |
| モデルの検索 | SHMODEL |
| コーナー検索条件設定 | SHDEFCORNER |
| コーナー検索 | SHCORNER |
| 円検索条件設定 | SHDEFCIRCLE |
| 円の検索 | SHCIRCLE |

## 8.17.8　結果取得

次に示すコマンドは、画像処理を行なった結果の内容について、各種情報の取得を行ないます。

結果取得コマンド

| 動作の種類 | コマンド |
|---|---|
| 画像処理結果の取得 | VISGETNUM |
| コード認識結果の取得 | VISGETSTR |
| 画像処理結果X座標の取得 | VISPOSX |
| 画像処理結果Y座標の取得 | VISPOSY |
| 処理結果ステータスの取得 | VISSTATUS |

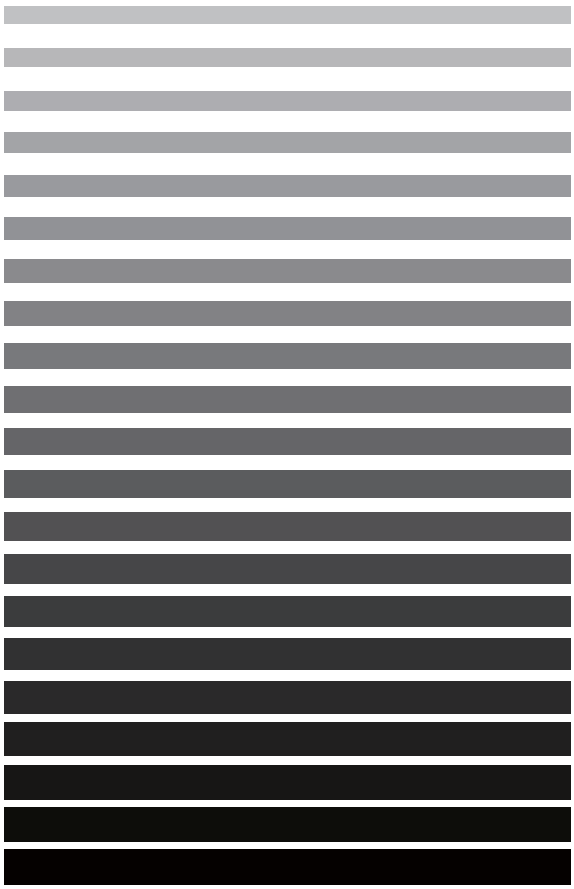
## 8.17.9　視覚 CAL

次に示すコマンドは､CAL(視覚-ロボット座標変換)データを取得します。

視覚CALコマンド

| 動作の種類 | コマンド |
|---|---|
| 座標変換データの取得 | VISREFCAL |

# 第 9 章
# 宣言文



プログラムの中で、変数や関数を使用する場合は、あらかじめ宣言文で定義をしておく必要があります。この章で説明する宣言文のコマンドは、そのために使います。
システム変数と組み込み関数は、宣言することなくプログラム中で使用できます。

# 第 9 章　宣言文

## 9.1　プログラム名

## PROGRAM（ステートメント）

**機　能**

プログラム名を宣言します。

**書　式**

PROGRAM < プログラム名 > [(< 引数 >[,< 引数 >…])]

**説　明**

< プログラム名 > で指定した文字列を、プログラム名として宣言します。

プログラム名の宣言は、プログラムの先頭の行で行ないます。先頭にプログラム名の宣言がない場合には、ファイル名がプログラム名になります。

プログラム名が、PRO< 番号 > という形式の名前になっている場合は、引数を付けることはできません。

プログラム名の先頭の文字はアルファベットでなければなりません。プログラム名は 64 文字までです。

< 引数 > は、呼び出し側から渡される引数データになります。CALL ステートメントから変数が引数として渡された場合には、PROGRAM ステートメント内で < 引数 > で指定した変数の値を変更すると、呼び出し側で < 引数 > に指定した変数の値も変わります。

< 引数 > の型は、CALL ステートメントの引数の型と同じでなければなりません。

型は、後置子または AS 表現を用いて以下のように行ないます。

```
PROGRAM  SUB0(aa%,bb!)
PROGRAM  SUB0(aa AS INTEGER,bb AS SINGLE)
```

< 引数 > に配列変数を使用するときには、呼び出し側の DIM ステートメントで定義した配列の添字の最大値と、添字の数を入力します。

< 引数 > の最大個数は、32 個です。

外部より起動可能なプログラムは PRO< 番号 > のものに限られます。

**関連項目**

CALL

**用　例**

```
PROGRAM PRO1                    'PRO1 をプログラム名として宣言します。
PROGRAM SUB2( la, lb%, lc# )    'la, lb%, lc# の引数を持たせた SUB2 をプログラムと
                                ' して宣言します。
PROGRAM SUB3( lb%( 12, 5 ) )    'SUB3 をプログラムとして宣言し、2 次元配列 lb% で引
                                ' 数を受け取ります。この例では、配列の添字は 12, 5 で
                                ' す。
PROGRAM SUB0( la%, lb! )
PROGRAM SUB0( la As Integer, lb As Single )
```

# 9.2　干渉エリア座標

## AREA（ステートメント）

**機　能**

干渉チェックを行なうエリアを宣言します。

**書　式**

AREA < エリア番号 >,< ポジション >,< ベクトル >,<I/O 番号 >,< 干渉位置格納ポジション型変数番号 >[,< エラー検出設定 >]

**説　明**

干渉チェックエリアを宣言します。

< エリア番号 > で宣言できるエリア番号は、0 ～ 31 までの 32ヶ所です。

< ポジション > は、干渉チェックエリアの中心位置と角度を指定するポジションです。

< ベクトル > は、干渉チェックエリアの範囲を指定するベクトルです。

エリアの一辺の長さは、< ベクトル > の各成分の 2 倍の長さとなります。

<I/O 番号 > には、干渉チェックエリアで干渉が起こったときに SET する I/O 番号を与えます。I/O の状態は、RESETAREA を実行するか、その I/O を RESET するまで保持されます。

[Ver.1.8 以降 ] では、10104、IO[104] と IO 変数表記が可能になりました。また、-1 を設定することにより IO 出力を行わない設定が可能になりました。

< 干渉位置格納ポジション型変数番号 > には、干渉した位置を格納するポジション型変数の番号を与えます。

[Ver.1.8 以降 ] では、P55、P[55] と P 型変数表記が可能になりました。また、-1 を設定すると P 型変数への位置の取り込みを行わない設定が可能になりました。

< エラー検出設定 > には､位置干渉検出のエラー条件を設定します｡干渉時のエラー設定は以下のとおりです。

0: エリア領域内部に干渉時、エラー出力無し
1: エリア領域内部に干渉時、エラー出力
2: エリア領域内部に干渉時、エラー出力（手動に切換えて操作可）
3: エリア領域外部に干渉時、エラー出力無し
4: エリア領域外部に干渉時、エラー出力
5: エリア領域外部に干渉時、エラー出力（手動に切換えて操作可）

干渉のチェックは、ツール座標系の原点と干渉チェックエリアに指定した直方体を比較します。干渉チェックエリアの内側にツール座標原点を検出すると、干渉と判定されます。

干渉位置格納ポジション型変数番号には、干渉が検出されたとき (I/O が SET されたとき ) の、ユーザ座標系におけるツール座標原点位置が格納されます。一般には直方体の面上の点になりますが、SETAREA 実行時にツール座標原点が直方体内部にある場合は、その位置となります。

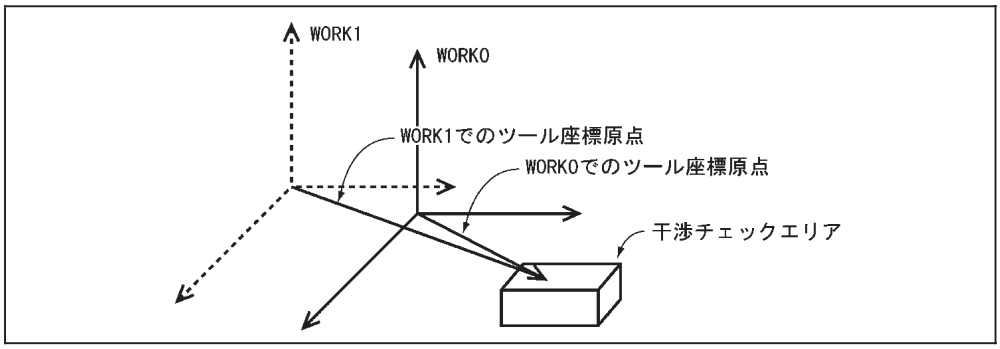
WINCAPS Ⅲ、ティーチングペンダントでも設定が可能です。

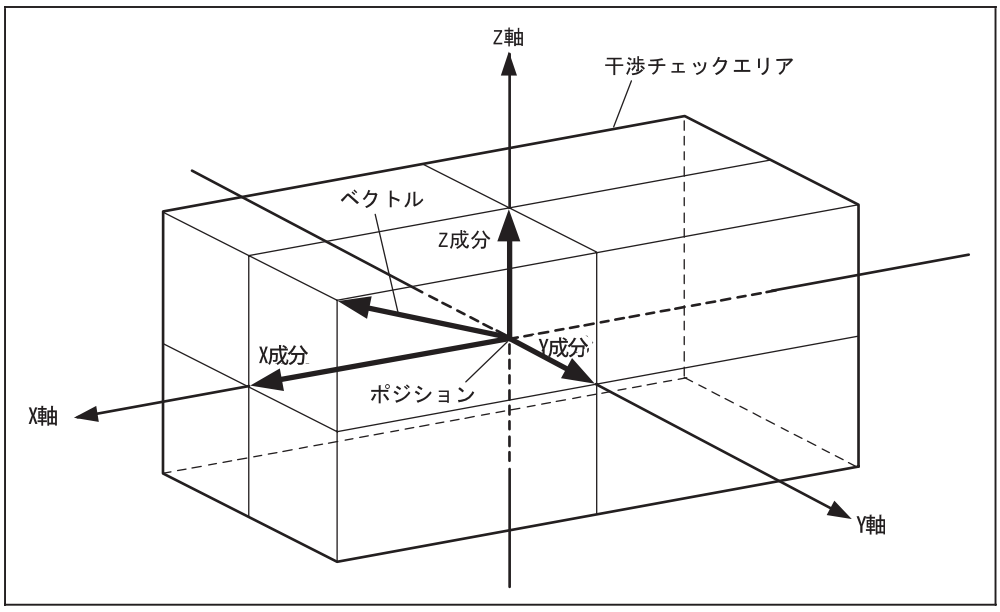
**関連項目**

SETAREA、RESETAREA、AREAPOS、AREASIZE

注意事項

- エリアの中心位置は、常に WORK0 が基準になっています。
- ユーザ座標系を変更しても、干渉チェックエリアの位置は変わりません。




用　例

**共通**

```
AREA 2, P50, V10, 104, 55     '2 番に P50, V10 で規定されるエリアを定義します。
SETAREA 2                     '2 番のエリアチェックを有効にします。
RESETAREA 2                   '2 番のエリアチェックを無効にします。
```

**6 軸**

```
AREA 2, P50+(100, 100, 0, 10, 0, 0 ), V10, 104, 55
                              '2 番に P50+(100, 100, 0, 10, 0, 0 ), V50 で規定さ
                              ' れるエリアを定義します。
SETAREA 2                     '2 番のエリアチェックを有効にします。
RESETAREA 2                   '2 番のエリアチェックを無効にします。
```

**4 軸**

```
AREA 2, P50+(100, 100, 0, 10 ), V10, 104, 55
                              '2 番に P50+(100, 100, 0, 10 ), V50 で規定されるエ
                              ' リアを定義します。
SETAREA 2                     '2 番のエリアチェックを有効にします。
RESETAREA 2                   '2 番のエリアチェックを無効にします。
```

# POSTUREAREA（ステートメント）【Ver.3.0 以降】

## 機　能

高感度位置姿勢検知機能の設定を宣言します。

## 書　式

POSTUREAREA< 位置 >,< 範囲 ( ベクトル )>,<I/O 番号 >,< 検知位置格納ポジション型変数番号 >[ ,< 検知時エラー設定 >][ ,TAR_POSTURE(< 姿勢 > , < 誤差範囲 >)][ ,TAR_EXJOINT（< 軸番号 > , < 位置 > , < 誤差範囲 >）]

**付加軸を 2 軸以上指定する場合は、**

POSTUREAREA< 位置 >,< 範囲 ( ベクトル )>,<I/O 番号 >,< 検知位置格納ポジション型変数番号 >[ ,< 検知時エラー設定 >][ ,TAR_POSTURE(< 姿勢 > , < 誤差範囲 >)][ ,TAR_EXJOINT（（< 軸番号 > ,< 位置 > ,< 誤差範囲 >）,（< 軸番号 > ,< 位置 > ,< 誤差範囲 >）[ ,（< 軸番号 > ,< 位置 > ,< 誤差範囲 >）・・・]）

## 説　明

高感度位置姿勢検知機能の位置 ( 範囲 )、姿勢、付加軸角度を宣言します。

指定した位置、姿勢、付加軸角度が全て範囲内であるとき位置と姿勢を同時にチェックできますのでカメラのシャッタートリガとして使用できます。

高感度位置姿勢検知機能は 1ms 間隔で検知チェックを行います。

< 位置 > : 位置をポジション型変数で指定します。詳細は干渉エリアコマンド (AREA) を参照ください。

< 範囲 ( ベクトル )> : 検知と判断する範囲を指定します。中心から端点までのベクトルを使用してエリアの大きさを指定します。詳細は干渉エリアコマンド (AREA) を参照ください。

<I/O 番号 > : 検知時、ON にする I/O 番号を指定します。I/O を使わない場合は -1 を入力してください。

< 検知位置格納ポジション型変数番号 > : 検知をしたときの現在位置をポジション型変数に格納できます。ポジション型変数を指定してください。変数への格納をしない場合は -1 を入力してください。

< 検知時エラー設定 > : 検知時のエラー出し方を設定します。

0 : 検知エリアに入ったとき、エラー出力なし。
1 : 検知エリアに入ったとき、エラー出力あり。
2 : 検知エリアに入ったとき、エラー出力なし。（手動に切り替えて操作可）
3 : 検知エリアから出たとき、エラー出力なし。
4 : 検知エリアから出たとき、エラー出力あり。
5 : 検知エリアから出たとき、エラー出力なし。（手動に切り替えて操作可）

TAR_POSTURE(< 姿勢 > , < 誤差範囲 >) : 検知姿勢を指定します ( 省略可 )。

< 姿勢 > : 検知する姿勢を指定します。ポジション型変数で指定する場合は位置の要素は無視されます。数値指定の場合は (Rx, Ry, Rz) で指定します。4 軸ロボットの場合は (Rz) もしくは (0, 0, Rz) で指定します。

< 誤差範囲 > : 検知する姿勢の範囲を指定します。< 姿勢 > からの誤差量を角度で指定します。-1 を指定すると姿勢をチェックしません。

TAR_EXJOINT（< 軸番号 > , < 位置 > , < 誤差範囲 >）: 検知する付加軸角度を指定します ( 省略可 )。

< 軸番号 > : 検知する付加軸の軸番号を指定します。

< 位置 > : 付加軸の検知する位置を指定します。

< 誤差範囲 > : 検知する位置の許容誤差を指定します。-1 を指定すると付加軸位置をチェックしません。

## 関連項目

SETPOSTUREAREA、RESETPOSTUREAREA

**注意事項**

- 高感度位置姿勢検知機能はオプション機能です。機能ライセンスを購入してください。
- 設定できるエリアは 1 つです。
- プログラムでのみ設定可能です。TP および WINCAPS では設定できません。
- 電源 OFF すると設定の保持はされません。
- モータ OFF 時は機能しませんがマシンロック時は 8ms で検知可能です。

**用　例**

```
POSTUREAREA, P1,V0,24,-1,0, TAR_POSTURE ( V2 , 1 ), TAR_EXJOINT ( ( 7 , 30 , 4 ),
( 8 , 90 , 1) )
```

## 9.3　ユーザ関数

# DEF FN（ステートメント）

**機　能**

ユーザ定義関数を宣言します。

**書　式**

DEF FN < 関数名 >[< 後置子 >] = < 定数 >

DEF FN < 関数名 >[< 後置子 >](< 引数 >[,< 引数 >…]) = < 数式 >

**説　明**

FN で始まる < 関数名 > をユーザ定義関数として宣言します。

< 引数 > には、< 数式 > の中で使用する変数名を指定します。

< 後置子 > は省略可能ですが、これを付けることによって、同時に変数の型を宣言することができます。後置子には次のものが使用できます。

整数型後置子：%
単精度型後置子：!
倍精度型後置子：#
文字列型後置子：$

後置子を省略すると、単精度実数型とみなされます。

変数名が同じで、型の違う変数を宣言することはできません。

**用　例**

```
DEF FND$ = "DENSO"                          'FND$ をユーザ定義関数として宣言します。
DEF FNLAP#(radius) = 2 * PI * radius        'FNLAP#(radius) を倍精度実数型のユーザ定義関
                                            ' 数として宣言します。
DEF FNAREA(radius) = PI * POW(radius, 2)    'FNAREA(radius) を単精度実数型のユーザ定義関
                                            ' 数として宣言します。
PRINT #1, FND$                              '"DENSO" を ch1 から出力します。
PRINT #2, HANKEI
PRINT #1, FNLAP#(HANKEI)                    '(2 * PI * HANKEI) の値を ch1 から出力します。
PRINT #2, FNAREA(HANKEI)                    '(PI * POW(HANKEI, 2) の値を ch2 から出力しま
                                            ' す。
```

# 9.4　ホーム座標

## HOME（ステートメント）

**機　能**

任意の座標をホームポジションとして宣言します。

**書　式**

HOME < ポジション型 >

**説　明**

< ポジション型 > で指定した任意の座標をホームポジションとして宣言します。

ホームポジションは各プログラムごとに定義されます。

HOME 文を複数宣言した場合は、最新の宣言をホームポジションとします。

**関連項目**

GOHOME

**用　例**

**6 軸**

```
DIM lp1 As position
HOME (350, 0, 450, 0, 0, 180 ) '(350, 0, 450, 0, 0, 180 ) の座標をホームポ
                               ' ジションとして宣言します。
HOME lp1                       'lp1 の座標をホームポジションとして宣言しま
                               ' す。
```

**4 軸**

```
DIM lp1 As position
HOME (200, 300, 300, 45, 0 )   '(200, 300, 300, 45, 0 ) の座標をホームポジ
                               ' ションとして宣言します。
HOME lp1                       'lp1 の座標をホームポジションとして宣言しま
                               ' す。
```

# 9.5　ツール座標

## TOOL（ステートメント）

**機　能**

ツール座標系を宣言します。

**書　式**

TOOL < ツール座標系番号 >,< ポジション型 >

**説　明**

<ポジション型> で示されるポジションを、<ツール座標系番号> で示されるツール座標系として宣言します。
ツール座標系番号には、1 ～ 63 が指定できます。

**関連項目**

CHANGETOOL、TOOLPOS

**注意事項**

このコマンドにより変更される値は、電源が入っている限り値を保持します。電源が切れても値を保持したい場合は、システムパラメータの保存（操作ガイド P5-181 参照）を行ってください。

**用　例**

6 軸

```
DIM lp1 As position
TOOL 1, lp1                          'lp1 を、ツール座標系番号 1 のツール座標系として宣
                                     ' 言します。
TOOL 2, (100, 100, 50, 0, 90, 0 )
```

4 軸

```
DIM lp1 As position
TOOL 1, lp1                          'lp1 を、ツール座標系番号 1 のツール座標系として宣
                                     ' 言します。
TOOL 2, (100, 100, 50, 0 )
```

# 9.6　ワーク座標

# WORK（ステートメント）

**機　能**

ユーザ座標系を宣言します。

**書　式**

WORK < ユーザ座標系番号 >,< ポジション型 >[,< 属性 >]

**説　明**

< ポジション型 > で示されるポジション型変数に代入したユーザ座標を、< ユーザ座標系番号 > で示されるユーザ座標系として宣言します。

ユーザ座標系番号には、1 ～ 7 が使用できます。

< 属性 > ではワーク座標に属性を付加します。(0:標準、1:固定工具) 省略時は 0 が入力されます。(Ver.3.0 以降)

**関連項目**

CHANGEWORK、WORKPOS

**注意事項**

- ワーク座標の X，Y，Z 要素にロボットの作業エリアを超える値を設定した場合、数値処理の桁落ちにより、ロボットの位置ずれが発生する場合があります。ワーク座標の X，Y，Z 要素はロボットの作業エリア内の値を用いてください。
- このコマンドにより変更される値は、電源が入っている限り値を保持します。電源が切れても値を保持したい場合は使用条件の「TOOL・WORK 復帰条件」を設定変更してください。
- 現在選択されている番号のワーク座標系の値を「WORK」コマンドで変更しても、ロボットのワーク座標系は変更されません。「CHANGEWORK」で再度選択してください。

**用　例**

**6 軸**

```
DEFINT li1, li2
WORK 1, P1                              ' ユーザ座標系番号 1 に、ポジション型変数番号 2 に代
                                        ' 入した座標を、ユーザ座標系として宣言します。
WORK li1, P[li2]                        'li1 で示されるユーザ座標系番号に、li2 で示される
                                        ' ポジション型変数に代入した座標を、ユーザ座標系
                                        ' として宣言します。
WORK li1, (100, 100, 50, 0, 0, 90)
WORK li1, P[li2], 1                     ' ワーク座標系 li1 に P[li2] を固定工具の
                                        ' ワーク座標として設定します。
WORK li1, (100, 100, 50, 0, 0, 90) , 1 ' ワーク座標系 li1 に固定工具の
                                        ' ワーク座標を設定します。
```

**4 軸**

```
DEFINT li1, li2
WORK 1, P1                          ' ユーザ座標系番号 1 に、ポジション型変数番号 2 に代
                                    ' 入した座標を、ユーザ座標系として宣言します。
WORK li1, P[li2]                    'li1 で示されるユーザ座標系番号に、li2 で示される
                                    ' ポジション型変数に代入した座標を、ユーザ座標系
                                    ' として宣言します。
WORK li1, (100, 100, 50, 0)
WORK li1, P[li2], 1                 ' ワーク座標系 li1 に P[li2] を固定工具の
                                    ' ワーク座標として設定します。
WORK li1, (100, 100, 50, 90) , 1    ' ワーク座標系 li1 に固定工具のワーク座標を設定します。
```

## 9.7　ローカル変数

# DEFINT（ステートメント）

**機　能**

I 型変数（整数型変数）を宣言します。整数の範囲は -2147483648 ～ 2147483647 です。

**書　式**

DEFINT < 変数名 >[=< 定数 >][,< 変数名 >[=< 定数 >]…]

**説　明**

< 変数名 > で指定した変数を整数型の変数として宣言します。< 変数名 > に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を宣言できます。

**関連項目**

DEFDBL、DEFSNG、DEFSTR

**用　例**

```
DEFINT lix, liy, liz     'lix, liy, liz を I 型変数として宣言します。
DEFINT lix = 1           'lix を I 型変数として宣言し、初期値 =1 を与えます。
```

# DEFSNG（ステートメント）

**機　能**

F 型変数（単精度実数型変数）を宣言します。単精度実数の範囲は -3.4E+38 ～ 3.4E+38 です。

**書　式**

DEFSNG < 変数名 >[=< 定数 >][,< 変数名 >[=< 定数 >]…]

**説　明**

< 変数名 > で指定した変数を単精度実数型として宣言します。< 変数名 > に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を指定できます。

**関連項目**

DEFDBL、DEFINT、DEFSTR

**用　例**

```
DEFSNG lfx, lfy, lfz     'lfx, lfy, lfz を F 型変数として宣言します。
DEFSNG lfx = 1.0         'lfx を F 型変数として宣言し、初期値 =1.0 を
                         ' 与えます。
```

# DEFDBL（ステートメント）

**機　能**

D 型変数（倍精度実数型変数）を宣言します。倍精度実数の範囲は -1.7D+308 ～ 1.7D+308 です。

**書　式**

DEFDBL < 変数名 >[=< 定数 >][,< 変数名 >[=< 定数 >]…]

**説　明**

< 変数名 > で指定した変数を倍精度実数型として宣言します。< 変数名 > に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を指定できます。

**関連項目**

DEFINT、DEFSNG、DEFSTR

**用　例**

```
DEFDBL ldx, ldy, ldz     'ldx, ldy, ldz を D 型変数として宣言します。
DEFDBL ldx = 1.0         'ldx を D 型変数として宣言し、初期値 =1.0 を
                         ' 与えます。
```

# DEFSTR（ステートメント）

**機　能**

S 型変数（文字列型変数）を宣言します。文字列の長さは、243 文字までです。

**書　式**

DEFSTR < 変数名 >[=< 定数 >][,< 変数名 >[=< 定数 >]…]

**説　明**

< 変数名 > で指定した変数を文字列型として宣言します。< 変数名 > に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を指定できます。

**関連項目**

DEFDBL、DEFINT、DEFSNG

**用　例**

```
DEFSTR lsx, lsy, lsz     'lsx, lsy, lsz を S 型変数として宣言します。
DEFSTR lsx = "DENSO"     'lsx を S 型変数として宣言し、初期値 ="DENSO" を与えま
                         ' す。
```

# DEFVEC（ステートメント）

**機　能**

V 型変数（ベクトル型変数）を宣言します。

**書　式**

DEFVEC < 変数名 >[=< ベクトル型定数 >][,< 変数名 >[=< ベクトル型定数 >]…]

**説　明**

< 変数名 > で指定した変数をベクトル型として宣言します。< 変数名 > に続けて等号とベクトル型定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を指定できます。

**関連項目**

DEFJNT、DEFPOS、DEFTRN

**用　例**

```
DEFVEC lvx, lvy, lvz        'lvx, lvy, lvz を V 型変数として宣言します。
DEFVEC lvx = (10, 10, 5)    'lvx を V 型変数として宣言し、初期値 =(10, 10, 5) を与
                            ' えます。
```

# DEFPOS（ステートメント）

**機　能**

P 型変数（ポジション型変数）を宣言します。

**書　式**

DEFPOS < 変数名 >[=< ポジション型定数 >][,< 変数名 >[=< ポジション型定数 >]…]

**説　明**

< 変数名 > で指定した変数をポジション型として宣言します。< 変数名 > に続けて等号とポジション型定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を指定できます。

**関連項目**

DEFJNT、DEFTRN、DEFVEC

**用　例**

6 軸

```
DEFPOS lpx, lpy, lpz                    'lpx, lpy, lpz を P 型変数として宣言します。
DEFPOS lpx = (10, 10, 5, 0, 9, 0, 1)    'lpx を P 型変数として宣言し、初期値 =(10, 10, 5, 0, 9,
                                        '0, 1) を与えます。
```

4 軸

```
DEFPOS lpx, lpy, lpz                    'lpx, lpy, lpz を P 型変数として宣言します。
DEFPOS lpx = (100, 100, 300, 45, 0)     'lpx を P 型変数として宣言し、初期値 =(100, 100, 300,
                                        '45, 0) を与えます。
```

# DEFJNT（ステートメント）

**機　能**

J 型変数（ジョイント型変数）を宣言します。

**書　式**

DEFJNT < 変数名 >[=< ジョイント型定数 >][,< 変数名 >[=< ジョイント型定数 >]…]

**説　明**

< 変数名 > で指定した変数をジョイント型として宣言します。< 変数名 > に続けて等号とジョイント型定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を指定できます。

**関連項目**

DEFPOS、DEFTRN、DEFVEC

**用　例**

6 軸

```
DEFJNT ljx, ljy, ljz                ’ljx, ljy, ljz を J 型変数として宣言します。
DEFJNT ljx = (10, 10, 5, 0, 9, 0)   ’ljx を J 型変数として宣言し、初期値 =(10, 10, 5, 0, 9,
                                    ’0) を与えます。
```

4 軸

```
DEFJNT ljx, ljy, ljz                ’ljx, ljy, ljz を J 型変数として宣言します。
DEFJNT ljx = (10, 10, 5, 0)         ’ljx を J 型変数として宣言し、初期値 =(10, 10, 5, 0)
                                    ’ を与えます。
```

# DEFTRN（ステートメント）

**機　能**

T 型変数（同次変換型変数）を宣言します。

**書　式**

DEFTRN < 変数名 >[=< 同次変換型定数 >][,< 変数名 >[=< 同次変換型定数 >]…]

**説　明**

< 変数名 > で指定した変数を同次変換型として宣言します。< 変数名 > に続けて等号と同次変換型定数を書くことにより、宣言と同時に初期化を行なえます。

"," で区切ることにより、一度に複数の変数名を指定できます。

**関連項目**

DEFJNT、DEFPOS、DEFVEC

**用　例**

```
DEFTRN ltx, lty, ltz      'ltx, lty, ltz を T 型変数として宣言します。
DEFTRN ltx = (10, 10, 5, 20, 20, 10, 30, 30, 15)
                          'ltx を T 型変数として宣言し、初期値 =(10, 10, 5, 20, 20,
                          '10, 30, 30, 15) を与えます。
```

# DEFIO（ステートメント）

**機　能**

入出力ポートに対応する IO 変数（I/O 変数）を宣言します。

**書　式**

DEFIO < 変数名 > = <I/O 変数の型 >,< ポートアドレス >[,< マスク情報 >]

**説　明**

< 変数名 > で指定した変数を I/O 変数として宣言します。

<I/O 変数の型 >

I/O 変数の型を選択します。I/O 変数の型は、BIT、BYTE、WORD、INTEGER、SINGLE があります。BIT 型は 1 ビット、BYTE 型は 8 ビット、WORD 型は 16 ビット、INTEGER 型は 32 ビット、SINGLE 型は 32 ビットの範囲を指定します。

（SINGLE 型は Ver.3.2 未満では使用できません）

< ポートアドレス >

入出力ポートでの開始番号を指定します。

整数型定数で指定します。

2 つの整数型定数の加減算式による指定も可能です。

< マスク情報 >

入力ポートの場合、入力データとマスク情報の AND をとります。

出力ポートの場合、出力データとマスク情報の AND をとり出力しますが、マスクがセットされていないビットの出力状態は変化しません。

SINGLE ではマスク情報は無効です。マスク情報を記述するとコンパイルエラーとなります。

**関連項目**

IN、OUT、SET、RESET

**注意事項**

- WORD,INTEGER の場合、最上位ビットにあたるポートは符号ビットとなります。
  SINGLE の表現方法は、IEEE 754 形式です。
  数値の範囲と、各種ビットに対するポート番号は下記のとおりとなります。

| | |
|---|---|
| WORD | 数値範囲　　：-32768 ～ 32767<br>最上位ビットポート番号：開始ポートアドレス + 15 |
| INTEGER | 数値範囲　　：-2147483648 ～ 2147483647<br>最上位ビットポート番号：開始ポートアドレス + 31 |
| SINGLE | 数値範囲　　：-3.4E+38 ～ 3.4E+38<br>符号部ポート番号　：開始ポートアドレス + 31<br>指数部ポート番号　：開始ポートアドレス + 30 ～開始ポートアドレス + 23<br>仮数部ポート番号　：開始ポートアドレス + 22 ～開始ポートアドレス |

- WINCAPSIII では、SINGLE 型の値の表示（ウォッチなど）に対応していません。SINGLE 型の値の参照はティーチングペンダントで行ってください。
- WORD 型の数値範囲は、IN コマンドで値を取得する時の範囲です。OUT コマンドで出力する時にこの範囲を超える数値を指定した場合、整数部分の下位 16 ビットを出力します。

用　例

```
#DEFINE IO_OFFSET 256                ’"IO_OFFSET" を 256 として定義
DEFIO samp1 = BIT, 1                 ’samp1 をポート 1 から始まる BIT 型 I/O 変数として宣
                                     ’ 言します。samp1 の返値は、ポート 1 の状態を表す、0
                                     ’ または 1 の 1 ビット整数となります。
DEFIO samp2 = BYTE, 10, &B00010000   ’samp2 をポート 10 から始まる BYTE 型 I/O 変数のマス
                                     ’ ク情報を付けて宣言します。samp2 の返値は、ポート 10
                                     ’ の状態を表す、0 または 16 の 8 ビット整数となります。
DEFIO samp3 = WORD, 15               ’samp3 をポート 15 から始まる WORD 型 I/O 変数として
                                     ’ 宣言します。samp3 の返値は、ポート 15 から 30 の状態
                                     ’ を表す、0 ～ &Hffff の 16 ビット整数となります。
DEFIO samp4 = INTEGER, 1             ’samp4 をポート 1 から始まる INTEGER 型 I/O 変数とし
                                     ’ て宣言します。samp4 の返値は、ポート 1 から 32 の状
                                     ’ 態を表す、0 ～ &Hffffffff の 32 ビット整数となります。
DEFIO samp5 = BYTE, IO_OFFSET + 10   ’samp5 を 10 + IO_OFFSET の計算結果 ( 用例では 266)
                                     ’ のポート番号 から始まる BYTE 型 I/O 変数として
                                     ’ 宣言します。samp5 の返値は、用例ではポート 266
                                     ’ から 273 の状 態を表す、0 ～ &Hff の 8 ビット整数
                                     ’ となります
DEFIO samp6 = SINGLE, 128            ’samp6 をポート 128 から始まる SINGLE 型 I/O 変数として
                                     ’ 宣言します。
                                     ’samp6 の戻値は、ポート 128 から 159 の状態を表す、
                                     ’-3.4E+38 ～ 3.4E+38 の 32 ビット単精度実数となります。
```

# 9.8　配列

## DIM（ステートメント）

**機　能**

配列を宣言します。

**書　式**

DIM < 変数名 >[< 後置子 >] [(< 要素数 >[,< 要素数 >[,< 要素数 >]])][AS< 変数型 >][,< 変数名 >[< 後置子 >]…]

**説　明**

< 変数名 > で指定した変数を配列変数として宣言します。

< 後置子 > を付けることによって、同時に変数の型を宣言することができます。

後置子には次のものが使用できます。

整数型後置子：%

単精度実数型後置子：!

倍精度実数型後置子：#

文字列型後置子：$

後置子を省略すると、単精度実数型とみなされます。

< 要素数 > には、配列の要素の最大値を指定します。要素数は 1 以上でなければなりません。

要素の総数は 32767 以下でなければなりません。

配列の次元は 3 次元までです。

配列で使用可能な添字の範囲は 0 ～（要素数 - 1）となります。

AS< 変数型 > を付けることによって変数の型を宣言することができます。

AS 表現と後置子は同時に使用できません。

AS 表現における < 変数型 > には次のものが使用できます。

| 変数型 | 識別子 | 変数型 | 識別子 |
|---|---|---|---|
| 長整数型 | INTEGER | ベクトル型 | VECTOR |
| 単精度実数型 | SINGLE | ポジション型 | POSITION |
| 倍精度実数型 | DOUBLE | ジョイント型 | JOINT |
| 文字列型 | STRING | 同次変換型 | TRANS |

**用　例**

```
DIM samp1(5)              'samp1 を大きさ (5) の単精度実数型の配列変数として宣
                          ' 言します。
DIM samp2(10, 10)         'samp2 を大きさ (10, 10) の単精度実数型の配列変数と
                          ' して宣言します。
DIM samp3(20, 5, 10)      'samp3 を大きさ (20, 5, 10) の単精度実数型の配列変数
                          ' として宣言します。
DIM samp4%(3, 3, 3)       'samp4 を大きさ (3, 3, 3) の整数型の配列変数として宣
                          ' 言します。
DIM samp5!(4, 3)          'samp5 を大きさ (4, 3) の単精度実数型の配列変数とし
                          ' て宣言します。
DIM samp6#(3)             'samp6 を大きさ (3) の倍精度実数型の配列変数として
                          ' 宣言します。
```

# 9.9　フォルダー機能

## FOLDER（ステートメント）【RC7_Ver.2.2 以降】

**機　能**

外部プログラムからアクセス可能なローカル変数を宣言します。

**書　式**

FOLDER DEFINT < 変数名 >[=< 定数 >] [,< 変数名 > [=< 定数 >]…]
FOLDER DEFSNG < 変数名 >[=< 定数 >] [,< 変数名 > [=< 定数 >]…]
FOLDER DEFDBL < 変数名 >[=< 定数 >] [,< 変数名 > [=< 定数 >]…]
FOLDER DEFSTR < 変数名 >[=< 定数 >] [,< 変数名 > [=< 定数 >]…]
FOLDER DEFVEC < 変数名 >[=< ベクトル定数 >] [,< 変数名 > [=< ベクトル定数 >]…]
FOLDER DEFPOS < 変数名 >[=< ポジション定数 >] [,< 変数名 > [=< ポジション定数 >]…]
FOLDER DEFJNT < 変数名 >[=< ジョイント定数 >] [,< 変数名 > [=< ジョイント定数 >]…]
FOLDER DEFTRN < 変数名 >[=< 同次変換型定数 >] [,< 変数名 > [=< 同次変換型定数 >]…]
FOLDER DIM < 変数名 > < 後置子 >[(< 要素数 >[,< 要素数 >[,< 要素数 >])][AS< 変数型 >][,< 変数名 >…]

**説　明**

FOLDER ステートメントでローカル変数を宣言すると、ローカル変数を外部のプログラムから EXTERN ステートメントを使用して参照可能となります。

- プログラム内で同一変数名は定義不可

| リスト 1 | 説明 |
|---|---|
| PRO1<br>  FOLDER DEFINT AA<br>  FOLDER DEFSNG AA<br>  DEFINT AA<br>END | リスト 1 では AA の宣言 3 行とも変数 2 重定義エラーになります。 |

- フォルダー内で、同一フォルダー変数名は定義不可

| リスト 2<br>(PRO1 と PRO2 は同一フォルダー内 ) | 説明 |
|---|---|
| PRO1<br>  FOLDER DEFINT AA<br>END<br><br>PRO2<br>  FOLDER DEFSNG AA<br>END | リスト 2 は同一フォルダー内で AA フォルダー変数を別のプログラムで定義しているのでエラーになります。 |

| リスト 3<br>(PRO1 と PRO2 は別のフォルダー ) | 説明 |
|---|---|
| PRO1<br>  FOLDER DEFINT AA<br>END | リスト 3 は別々のフォルダーで同じ AA フォルダー変数を定義可能（異なる変数となる） |
| PRO2<br>  FOLDER DEFINT AA<br>END | |

• 別プログラムでフォルダー変数名とローカル変数名が同一の時は宣言可能

| リスト４<br>(PRO1 と PRO2 は同一フォルダー内 ) | 説明 |
|---|---|
| PRO1<br>  FOLDER DEFINT AA<br>END<br><br>PRO2<br>  DEFINT AA<br>END | PRO1 の AA と PRO2 の AA は別変数になります。 |

ペンダントでプログラム編集や、WINCAPS Ⅲから＜マップ／実行プログラム＞をコントローラに送信すると FOLDER 変数はクリアされます。

**関連項目**

EXTERN

**用　例**

1) 同一フォルダー内に下記 3 つのプログラムが存在する場合
   - PRO1 を実行後に PRO3 を実行すると "STOP"( プログラム終了 ) します。
   - PRO2 を実行後に PRO3 を実行すると "HOLD "STOP""( プログラムを一時停止 ) します。
   - 3 つのプログラムで宣言されている AA 変数は 3 つとも同じ値になります。

```
PRO1
   FOLDER DEFINT AA      'FOLDER変数を宣言
   AA=1                  'AA変数に 1 を代入
END
PRO2
   EXTERN DEFINT AA      'EXTERN変数を宣言
   AA=2                  'AA変数に2を代入
END
PRO3
   EXTERN DEFINT AA      'EXTERN変数を宣言
   SELECT CASE AA        'AAの値がCASE文と一致した場合そのコマンド
                         'が実行されます。
   CASE 1                'AAが 1 の場合
      STOP               'プログラムを終了します。
   CASE 2                'AAが2の場合
      HOLD "STOP"        'プログラム実行を一時停止します。
   END SELECT            '複数条件判断文の終了を宣言します。
END
```

2) 下記のフォルダー構造にプログラムが存在する場合
注 1 : I100 に 33 が代入されます。
注 2 : I100 に 55 が代入されます。

```
PRO1
   FOLDER DEFINT  AA          'FOLDER変数を宣言
   CALL  LIB1.PRO1            'LIB1フォルダーのPRO1をCALLする
   I100 = AA                  'I100にAAを代入する　(注1)
   CALL  LIB2.PRO1            'LIB2フォルダーのPRO1をCALLする
   I100 = AA                  'I100にAAを代入する　(注2)
END
```

LIB1 フォルダー

```
PRO1
   EXTERN DEFINT  AA          'EXTERN変数を宣言
   AA  = 33                   'AAに33を代入する
END
```

LIB2 フォルダー

```
PRO1
   EXTERN DEFINT  AA          'EXTERN変数を宣言
   AA  = 55                   'AAに55を代入する
END
```

3) FOLDER 変数で初期値を宣言した場合
注 : PRO1 を実行しないと初期値は代入されません。

- PRO1 を実行後に PRO3 を実行すると I1 には 3 が入り、I2 にも 3 が入る。
- PRO2 を実行後に PRO1 を実行すると I1 には 3 が入り、I2 には 0 が入る。

```
PRO1
   FOLDER DEFINT AA＝3
   I1=AA
END

PRO2
   EXTERN DEFINT AA
   I2=AA
END
```

# EXTERN（ステートメント）【RC7_Ver.2.2 以降】

**機　能**

他のプログラムで定義されている FOLDER 変数を参照することを宣言します。

**書　式**

EXTERN DEFINT < 変数名 >[,< 変数名 >…]
EXTERN DEFSNG < 変数名 >[,< 変数名 >…]
EXTERN DEFDBL < 変数名 >[,< 変数名 >…]
EXTERN DEFSTR < 変数名 >[,< 変数名 >…]
EXTERN DEFVEC < 変数名 >[,< 変数名 >…]
EXTERN DEFPOS < 変数名 >[,< 変数名 >…]
EXTERN DEFJNT < 変数名 >[,< 変数名 >…]
EXTERN DEFTRN < 変数名 >[,< 変数名 >…]
EXTERN DIM < 変数名 > < 後置子 >[(< 要素数 >[,< 要素数 >[,< 要素数 >]])][AS< 変数型 >][,< 変数名 >…]

**説　明**

他のプログラムで定義されている FOLDER 変数を参照することを宣言します。

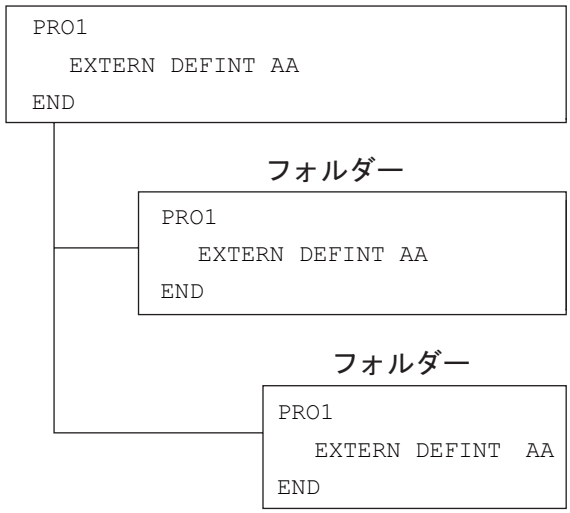
**FOLDER 変数探索方法**

1) EXTERN が宣言されている PAC が存在するフォルダーに FOLDER 変数が存在するか探索する。
   - FOLDER 変数が存在する場合は探索終了。
   - 存在しない場合は 2) へ進む。
2) 1 つ上のフォルダーに移り FOLDER 変数が存在するか探索する。
   - FOLDER 変数が存在する場合は探索終了。
   - 存在しない場合は (2) を繰り返す。
   - フォルダーのトップを探索しても FOLDER 変数が存在しない場合はコンパイルエラーになる。

| リスト 1 | 説明 |
|---|---|
| PRO1<br>  EXTERN DEFINT AA<br>  DEFINT AA<br>END | リスト 1 は同じ変数名が PAC プログラムの中で宣言されているのでコンパイルエラーになります。 |

| リスト 2 | 説明 |
|---|---|
| PRO1<br>  EXTERN DEFINT AA<br>  EXTERN DEFINT AA<br>END | リスト 2 は同じ EXTERN 変数名が同一の PAC ファイルに宣言されているのでコンパイルエラーになります。 |

| リスト 3 | 説明 |
|---|---|
| PRO1<br>  EXTERN DEFINT AA<br>END<br><br>PRO2<br>  EXTERN DEFINT AA<br>END | リスト 3 で別の PAC ファイルに同じ EXTERN 変数が宣言されている場合は宣言可能です。 |

| リスト 4 | 説明 |
| --- | --- |
|  | リスト 4 で別のフォルダーで同じ EXTERN 宣言されている場合は宣言可能です。 |

ペンダントでプログラム編集や、WINCAPS Ⅲから＜マップ／実行プログラム＞をコントローラに送信すると EXTERN 変数はクリアされます。

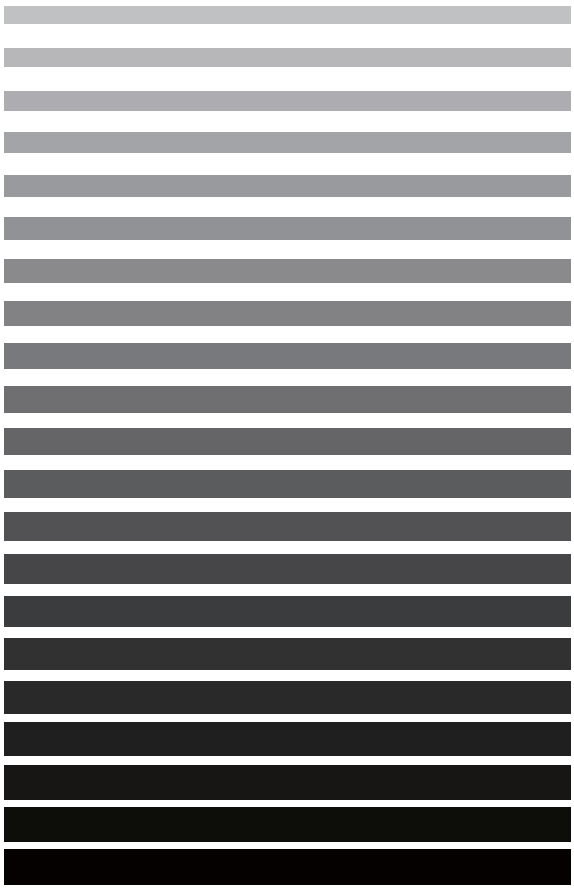
**関連項目**

FOLDER

**用　例**

FOLDER に記載の用例を参照。

# 第 10 章
# 代入文



変数に値を代入するときには、代入文のコマンドを使います。扱う値の型や内容によって、コマンドを使い分けます。

# 第 10 章　代入文

## 10.1　変数

### LET（ステートメント）【SLIM 準拠】

**機　能**

変数に値を代入します。

**書　式**

[LET] < 変数名 > = < 演算式 >

**説　明**

[LET] は省略可能です。

基本的には、< 変数名 > と < 演算式 > の型は、同じでなければなりません。

ただし、変数名と演算式の型が違う場合は以下のように変換されます。

- < 変数名 > の変数の型に変換されます。
  実数が整数に変換される場合は、小数点以下は切り捨てられます。
  倍精度実数変数が単精度実数変数に変換される場合は、値は有効数字 7 桁に丸めたものになります。
- 演算は、精度の高い方の型の精度で演算されます。

**用　例**

```
DEFINT li1, li2
LET li1 = li2 + 1                    '(li2 + 1) の値を li1 に代入します。
li1 = li2 + 1                        '(li2 + 1) の値を li1 に代入します ( 上記命令文と同
                                     ' じ意味です )。
```

6 軸

```
DEFPOS lp1, lp2
DEFJNT lj1, lj2
lp1 = lp2 + (10, 10, 10, 0, 0, 0)    '(lp2 + (10, 10, 10, 0, 0, 0) ) の値を lp1 に代入
                                     ' します。
lj1 = lj2 + (10, 20, 30, 40, 0, 0)   '(lj2 + (10, 20, 30, 40, 0, 0) ) の値を lj1 に代
                                     ' 入します。
```

4 軸

```
DEFPOS lp1, lp2
DEFJNT lj1, lj2
lp1 = lp2 + (10, 10, 10, 20)         '(lp2 + (10, 10, 10, 20) ) の値を lp1 に代入しま
                                     ' す。
lj1 = lj2 + (10, 20, 30, 40)         '(lj2 + (10, 20, 30, 40) ) の値を lj1 に代入しま
                                     ' す。
```

## 10.2　ベクトル

# LETA（ステートメント）

**機　能**

同次変換型のアプローチベクトルへ代入します。

**書　式**

LETA < 同次変換型変数 > = < ベクトル型 >

**説　明**

LETA で始まる代入文で、式の右辺はベクトルです。この代入の実行によって、< 同次変換型変数 > の内のアプローチベクトルが < ベクトル型 > に変更されます。

**関連項目**

LETO、LETP

**用　例**

```
DEFTRN lt1, lt2, lt3
DEFVEC lv1, lv2
LETA lt1 = AVEC(lt3)        'lt3 のアプローチベクトルを lt1 のアプローチベクト
                            ' ルに代入します。
LETA lt2 = lv1 x lv2        '(lv1 x lv2) の値を lt2 のアプローチベクトルに代入
                            ' します。
```

# LETO（ステートメント）

**機　能**

同次変換型のオリエントベクトルへ代入します。

**書　式**

LETO＜同次変換型変数＞＝＜ベクトル型＞

**説　明**

LETO で始まる代入文で、式の右辺はベクトルです。この代入の実行によって、＜同次変換型変数＞の内のオリエントベクトルが＜ベクトル型＞に変更されます。

**関連項目**

LETA、LETP

**用　例**

```
DEFTRN lt1, lt2, lt3
DEFVEC lv1, lv2
LETO lt1 = OVEC(lt3)        'lt3 のオリエントベクトルを lt1 のオリエントベクト
                            ' ルに代入します。
LETO lt2 = lv1 x lv2        '(lv1 x lv2) の値を lt2 のオリエントベクトルに代入
                            ' します。
```

# LETP（ステートメント）

**機　能**

ポジション型または同次変換型の位置ベクトルへ代入します。

**書　式**

LETP {< ポジション型変数 >|< 同次変換型変数 >} = < ベクトル型 >

**説　明**

LETP で始まる代入文で、式の右辺はベクトルです。この代入の実行によって、< ポジション型変数 > または < 同次変換型変数 > の内の位置ベクトルが < ベクトル型 > に変更されます。

**関連項目**

LETO、LETA

**用　例**

```
DEFTRN lt1, lt2
LETP lt1 = PVEC(lt2)          'lt2 の位置ベクトルを lt1 の位置ベクトルに代入しま
                              ' す。
```

## 10.3　形態

# LETF（ステートメント）

**機　能**

ポジション型または同次変換型の形態成分へ代入します。

**書　式**

LETF {< ポジション型変数 >|< 同次変換型変数 >} = < 形態 >

**説　明**

LETF で始まる代入文で、式の右辺は < 形態 > です。この代入の実行によって、< ポジション型変数または同次変換型変数 > の内の形態成分が < 形態 > の値に変更されます。

**関連項目**

CURFIG、FIG、ロボット形態 ( 付録 2)

**用　例**

```
DEFPOS lp1, lp2
LETF lp1 = 1                    'lp1 の形態を LEFTY-ABOVE-FLIP に設定します。
LETF lp2 = CURFIG               'lp2 の形態を現在の姿勢に設定します。
```

# 10.4　リンク角

## LETJ（ステートメント）

**機　能**

ジョイント型の指定リンク角へ代入します。

**書　式**

LETJ＜軸番号＞,＜ジョイント型変数＞＝＜数式＞

**説　明**

LETJ＜軸番号＞で始まる代入文で、式の右辺は＜数式＞です。この代入の実行によって、＜ジョイント型変数＞の内の＜軸番号＞で指定したリンク角が＜数式＞の値に変更されます。

**関連項目**

JOINT

**用　例**

```
DEFJNT lj1, lj2, lj3
DEFSNG lf1, lf2
LETJ 1, lj1 = JOINT(2, lj3)  'lj3 の 2 軸のリンク角を lj1 の 1 軸に代入します。
LETJ 3, lj2 = lf1 - lf2      '(lf1 - lf2) の値を lj2 の 3 軸に代入します。
```

## 10.5　姿勢

# LETR（ステートメント）

**機　能**

ポジション型の 3 つの回転成分へ代入します。

**書　式**

LETR＜ポジション型変数＞=＜ベクトル型＞

**説　明**

LETR で始まる代入文で、式の右辺はベクトルです。この代入の実行によって、＜ポジション型変数＞の内の姿勢が＜ベクトル型＞に変更されます。

**関連項目**

LETP、LETRX、LETRY、LETRZ

**用　例**

```
DEFPOS lp1, lp2
LETR lp1 = RVEC(lp2)          'lp2 の姿勢を lp1 の姿勢に代入します。
```

## 10.6　回転成分

# LETRX（ステートメント）

**機　能**

ポジション型の X 軸回転成分へ代入します。

**書　式**

LETRX < ポジション型変数 > = <X 軸回転角 >

**説　明**

LETRX で始まる代入文で、式の右辺は X 軸回転角です。この代入の実行によって、< ポジション型変数 > の内の X 軸回転成分が、<X 軸回転角 > の値に変更されます。

**関連項目**

LETRY、LETRZ、LETR

**用　例**

```
DEFPOS lp1, lp2, lp3
DEFSNG lf1, lf2
LETRX lp1 = POSRX(lp3)        'lp3 の X 軸回転成分を lp1 の X 軸回転成分に代入しま
                              ' す。
LETRX lp2 = lf1 - lf2         '(lf1 - lf2) の値を lp2 の X 軸回転成分に代入します。
```

# LETRY（ステートメント）

**機　能**

ポジション型の Y 軸回転成分へ代入します。

**書　式**

LETRY < ポジション型変数 > = <Y 軸回転角 >

**説　明**

LETRY で始まる代入文で、式の右辺は Y 軸回転角です。この代入の実行によって、< ポジション型変数 > の内の Y 軸回転成分が、<Y 軸回転角 > の値に変更されます。

**関連項目**

LETRX、LETRZ、LETR

**用　例**

```
DEFPOS lp1, lp2, lp3
DEFSNG lf1, lf2
LETRY lp1 = POSRY(lp3)      'lp3 の Y 軸回転成分を lp1 の Y 軸回転成分に代入しま
                            ' す。
LETRY lp2 = lf1 - lf2       '(lf1 - lf2) の値を lp2 の Y 軸回転成分に代入します。
```

# LETRZ（ステートメント）

**機　能**

ポジション型の Z 軸回転成分へ代入します。

**書　式**

LETRZ < ポジション型変数 > = <Z 軸回転角 >

**説　明**

LETRZ で始まる代入文で、式の右辺は Z 軸回転角です。この代入の実行によって、
< ポジション型変数 > の内の Z 軸回転成分が、<Z 軸回転角 > の値に変更されます。

**関連項目**

LETRX、LETRY、LETR

**用　例**

```
DEFPOS lp1, lp2, lp3
DEFSNG lf1, lf2
LETRZ lp1 = POSRZ(lp3)        'lp3 の Z 軸回転成分を lp1 の Z 軸回転成分に代入しま
                              ' す。
LETRZ lp2 = lf1 - lf2         '(lf1 - lf2) の値を lp2 の Z 軸回転成分に代入します。
```

# LETT（ステートメント）

**機　能**

ポジション型の T 成分へ代入します。

**書　式**

LETT < ポジション型変数 > = <T 軸回転角 >

**説　明**

LETT で始まる代入文で、式の右辺は T 軸回転角です。この代入の実行によって、
< ポジション型変数 > の内の T 軸回転成分が、<T 軸回転角 > の値に変更されます。

**用　例**

```
DEFPOS lp1
DEFSNG lf1, lf2
LETT lp1 = lf1 - lf2          '(lf1 - lf2) の値を lp1 の T 軸回転成分に代入します。
```

## 10.7　軸成分

# LETX（ステートメント）【SLIM 準拠】

**機　能**

ベクトル型／ポジション型／同次変換型の X 軸成分へ代入します。

**書　式**

LETX {< ベクトル型変数 >|< ポジション型変数 >|< 同次変換型変数 >}=<X 軸成分 >

**説　明**

LETX で始まる代入文で、式の右辺は X 軸成分です。この代入の実行によって、

< ベクトル型変数 > または < ポジション型変数 > または < 同次変換型変数 > の内の X 軸成分が、<X 軸成分 > の値に変更されます。

**関連項目**

LETP、LETY、LETZ

**用　例**

```
DEFPOS lp1, lp2
DEFVEC lv1, lv2
LETX lv1 = POSX(lv2)          'lv2 の X 軸成分を lv1 の X 軸成分に代入します。
LETX lp2 = POSX(lp2)          'lp2 の X 軸成分を lp1 の X 軸成分に代入します。
```

# LETY（ステートメント）【SLIM 準拠】

**機　能**

ベクトル型／ポジション型／同次変換型の Y 軸成分へ代入します。

**書　式**

LETY {< ベクトル型変数 >|< ポジション型変数 >|< 同次変換型変数 >}=<Y 軸成分 >

**説　明**

LETY で始まる代入文で、式の右辺は Y 軸成分です。この代入の実行によって、
< ベクトル型変数 > または < ポジション型変数 > または < 同次変換型変数 > の内の Y 軸成分が、<Y 軸成分 > の値に変更されます。

**関連項目**

LETP、LETX、LETZ

**用　例**

```
DEFPOS lp1, lp2
DEFVEC lv1, lv2
LETY lv1 = POSY(lv2)          'lv2 の Y 軸成分を lv1 の Y 軸成分に代入します。
LETY lp2 = POSY(lp2)          'lp2 の Y 軸成分を lp1 の Y 軸成分に代入します。
```

# LETZ（ステートメント）【SLIM 準拠】

**機　能**

ベクトル型／ポジション型／同次変換型の Z 軸成分へ代入します。

**書　式**

LETZ {< ベクトル型変数 >|< ポジション型変数 >|< 同次変換型変数 >}=<Z 軸成分 >

**説　明**

LETZ で始まる代入文で、式の右辺は Z 軸成分です。この代入の実行によって、
< ベクトル型変数 > または < ポジション型変数 > または < 同次変換型変数 > の内の Z 軸成分が、<Z 軸成分 > の値に変更されます。

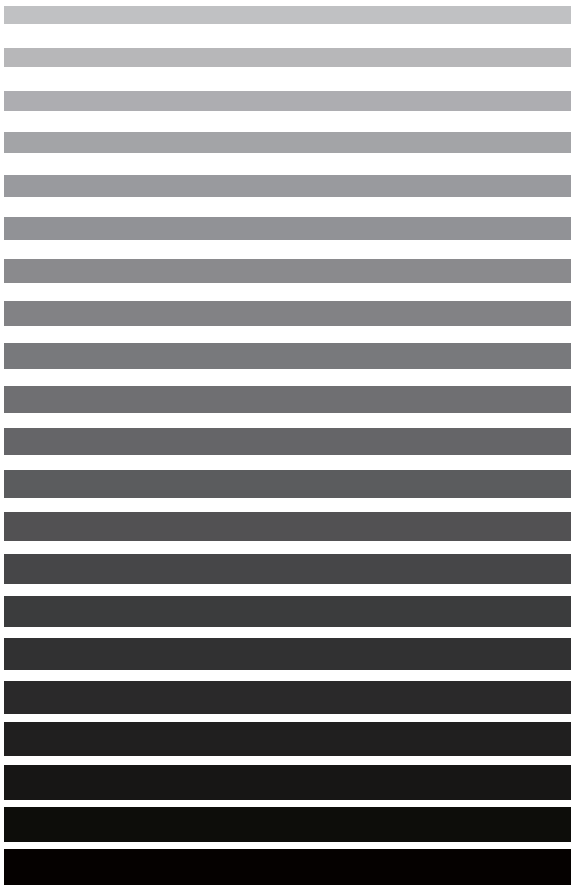
**関連項目**

LETP、LETX、LETY

**用　例**

```
DEFPOS lp1, lp2
DEFVEC lv1, lv2
LETZ lv1 = POSZ(lv2)          'lv2 の Z 軸成分を lv1 の Z 軸成分に代入します。
LETZ lp2 = POSZ(lp2)          'lp2 の Z 軸成分を lp1 の Z 軸成分に代入します。
```

# 第 11 章
# フロー制御文



状況を判断して、次の手順を選ぶなど、プログラムの流れを状況に応じて変える場合には、フロー制御文を用います。状況を判定する方法や、次の手順への進み方にはいくつも種類があります。

# 第 11 章　フロー制御文

## 11.1　プログラムの停止

### END（ステートメント）【SLIM 準拠】

**機　能**

プログラムによる動作の終了を宣言します。

**書　式**

END

**説　明**

このコマンドを実行すると、プログラムによる動作を終了します。

プログラムファイルの終了を意味するものではありません。

サブルーチンを使用する場合、メインルーチンの後ろにサブルーチンを集めて書く場合がありますが、メインルーチンとサブルーチンの区分けを、この END 命令により行ないます。

サブルーチンを使用しない場合、END 命令を省略すると最終行に END があるものとして処理をします。

**関連項目**

PROGRAM

**用　例**

```
DEFSNG lf1, lf2, lf3
PROGRAM prog1(lf1, lf2, lf3)
END                                    'prog1 を終了します。
```

# STOP（ステートメント）【SLIM 準拠】

**機　能**

プログラムの実行を終了します。

**書　式**

STOP

**説　明**

プログラムの実行中に STOP 文に出会うと、そこでプログラムの実行を終了します。

**関連項目**

DELAY、HALT、HOLD、STATUS

**用　例**

REM 複数条件判断を行ないます。

```
SELECT CASE Index     'Index の値が CASE 文の値と一致した場合、そのコマンドが実行されま
                      ' す。
  CASE 0              'Index が 0 の場合。
    STOP              ' プログラムを終了します。
  CASE 1              'Index が 1 の場合。
    HALT "STOP"       ' プログラム実行を一時停止します。
  CASE 2              'Index が 2 の場合。
    HOLD "STOP"       ' プログラム実行を一時停止します。
  CASE 3              'Index が 3 の場合。
    STOPEND           ' 連続起動されたプログラムをサイクル停止します。
  CASE 4              'Index が 4 の場合。
    ON li1 + li2 GOSUB *samp1, *samp2, *samp3
                                                'li1 + li2 の値と同じ順位に書かれたラベル
                                                ' 名のサブルーチンを呼び出します。
  CASE 5              'Index が 5 の場合。
    ON li1 + li2 GOTO *samp1, *samp2, *samp3
                                                'li1 + li2 の値と同じ順位に書かれたラベル
                                                ' へジャンプします。
  CASE 6              'Index が 6 の場合。
    END               ' プログラムによる動作の終了を宣言します。
END SELECT            ' 複数条件判断文の終了を宣言します。
```

# STOPEND（ステートメント）

**機　能**

連続起動またはCYCLEオプション付きで起動されたプログラムをサイクル停止させるステートメントです。このステートメントを含むプログラムを、サイクル起動した場合は、このステートメントを実行しても、動作に影響しません。

**書　式**

STOPEND

**説　明**

プログラムの実行中にSTOPEND文に出会うと、実行中のプログラムをサイクル停止します。

**関連項目**

STOP、HALT、END

**用　例**

REM 複数条件判断を行ないます。

```
SELECT CASE Index      'Indexの値がCASE文の値と一致した場合、そのコマンドが
                       ' 実行されます。
  CASE 0               'Indexが0の場合。
    STOP               ' プログラムを終了します。
  CASE 1               'Indexが1の場合。
    HALT "STOP"        ' プログラム実行を一時停止します。
  CASE 2               'Indexが2の場合。
    HOLD "STOP"        ' プログラム実行を一時停止します。
  CASE 3               'Indexが3の場合。
    STOPEND            ' 連続起動されたプログラムをサイクル停止します。
  CASE 4               'Indexが4の場合。
    ON li1 + li2 GOSUB *samp1, *samp2, *samp3
                                          'li1＋li2の値と同じ順位に書かれた
                                          ' ラベル名のサブルーチンを呼び出します。
  CASE 5               'Indexが5の場合。
    ON li1 + li2 GOTO *samp1, *samp2, *samp3
                                          'li1＋li2の値と同じ順位に書かれた
                                          ' ラベルへジャンプします。
  CASE 6               'Indexが6の場合。
    END                ' プログラムによる動作の終了を宣言します。
END SELECT             ' 複数条件判断文の終了を宣言します。
```

## 11.2　呼び出し

# CALL（ステートメント）

**機　能**

プログラムを呼び出し、実行します。

**書　式**

CALL < プログラム名 > [(< 引数 >[,< 引数 >…])]

**説　明**

CALL ステートメントは、< プログラム名 > で指定したプログラムを呼び出し、制御を移します。

CALL されたプログラム内の END 文は、サブルーチン内の RETURN 文と同じ意味を持ち、その呼び出し元へ制御を戻します。

< 引数 > は、< プログラム名 > で指定したプログラムへ渡す引数を指定します。

< 引数 > は、定数または式を用いる「値による呼び出し」と、変数を渡す「参照による呼び出し」の、いずれかで渡すことができます。詳しくは、「8.16　値による呼び出しと参照による呼び出し」を参照してください。

**1. 値による呼び出し**

定数と数式と文字式は値で渡します。変数もカッコ（ ）で囲めば式とみなされ、値で渡すことができます。

番号付き変数のうち整数型、単精度実数型、倍精度実数型、文字列型変数はカッコ（ ）で囲めば、値で渡すことができます。

例：プログラム PROGRAM SUB1(AA#) を呼び出す方法

- 変数を値で渡す場合　CALL SUB1((D1))
- 定数を渡す場合　　CALL SUB1(10#)
- 数式を渡す場合　　CALL SUB1(D1 + D2)

**注意**

**数式を渡す場合、式の結果の型に注意します。呼び出される側のプログラムの引数の型と同じ型にならなければいけません。**

2. 参照による呼び出し

参照による呼び出しでは、変数を引数として渡すことができます。

＜引数＞として配列全体を指定したいときは、配列名にカッコ（ ）を付けます。

グローバル変数、ローカル配列の 1 要素は、参照で渡すことはできません。

グローバル変数、ローカル配列の 1 要素を参照で渡したい場合は、いったんローカル変数に値を代入し、そのローカル変数を介して参照呼び出しを行なってください。

参照による呼び出しで渡した変数は、呼び出された側のプログラムの中で、その内容を変更されると、元のプログラムに戻った場合でも、その変更は有効となります。

例 1：プログラム PROGRAM SUB1(AA#) を呼び出す場合

- ローカル変数を渡す場合 1　CALL SUB1(DD#)
- ローカル変数を渡す場合 2　CALL SUB1(DA)

（DEFDBL で DA が宣言済みのとき）

**注意**

**ローカル変数はあらかじめ、値を代入しておく必要があります。**

例 2：プログラム PROGRAM SUB2(BB%(10)) を呼び出す場合

- 配列全体を渡す場合　CALL SUB2(AB%())

（DIM で AB%(10) が宣言済みのとき）

**注意**

- **変数に値が代入されていない場合は、実行時エラーとなります。**
- **プログラム名が、PRO< 数字 > の場合は、引数を渡すことはできません。**
- **呼び出す側と呼び出される側の引数の数を合わせてください。**

**関連項目**

PROGRAM

**用　例**

```
DEFDBL ld1, ld2
CALL SUB1((ld1))              ’変数を値で渡す場合。
CALL SUB1(10#)                ’定数を渡す場合。
CALL SUB1(ld1 + ld2)          ’数式を渡す場合。
```

# GOSUB（ステートメント）【SLIM 準拠】

**機　能**

サブルーチンを呼び出します。

**書　式**

GOSUB < ラベル名 >

**説　明**

指定した < ラベル名 > が示すサブルーチンを呼び出します。
一つのサブルーチンの中から、他のサブルーチンを呼び出すこと ( サブルーチンの多重化 ) もできます。

**関連項目**

GOTO、RETURN

**注意事項**

GOSUB（ON ～ GOSUB）で呼び出されたサブルーチンから呼び出し元に戻る場合は、RETURN 文を使用する必要があります。

**用　例**

```
DIM li1 As Integer
IF li1 = 0 THEN              'li1 が 0 の場合。
  STOP                       ' プログラムの実行を終了します。
ELSEIF li1 = 1 THEN          'li1 が 1 の場合。
  GOTO *samp1                '*samp1 のラベルへジャンプします。
  GO TO *samp2               '*samp2 のラベルへジャンプします。
ELSEIF li1 = 2 THEN          'li1 が 2 の場合。
  GOSUB *samp3               ' ラベル名 *samp3 のサブルーチンを呼び出します。
ELSE                         'li1 がその他の場合。
  RETURN                     ' 元のプログラムに戻ります。
END IF                       'IF 文の終了を宣言します。
```

# ON...GOSUB（ステートメント）【SLIM 準拠】

**機　能**

式の値に応じて、対応するサブルーチンの呼び出しを行ないます。

**書　式**

ON < 式 > GOSUB < ラベル名 >[,< ラベル名 >]…

**説　明**

< 式 > の値と同じ順位に書かれたラベルへジャンプして、プログラムの実行を継続します。このときの順位は、左から 1 、2、… と数えます。

< 式 > の値が実数のときは、その値を超えない最大の整数に丸められた後処理します。

**注意**

**式の結果が順位を超えていた場合は、何も実行しません。**

**関連項目**

ON...GOTO、RETURN、SELECT CASE

**注意事項**

GOSUB（ON...GOSUB）で呼び出されたサブルーチンから呼び出し元に戻る場合は、RETURN 文を使用する必要があります。

**用　例**

REM 複数条件判断を行ないます。

```
SELECT CASE Index     'Index の値が CASE 文の値と一致した場合、そのコマンドが
                      ' 実行されます。
  CASE 0              'Index が 0 の場合。
    STOP              ' プログラムを終了します。
  CASE 1              'Index が 1 の場合。
    HALT "STOP"       ' プログラム実行を一時停止します。
  CASE 2              'Index が 2 の場合。
    HOLD "STOP"       ' プログラム実行を一時停止します。
  CASE 3              'Index が 3 の場合。
    STOPEND           ' 連続起動されたプログラムをサイクル停止します。
  CASE 4              'Index が 4 の場合。
    ON li1 + li2 GOSUB *samp1, *samp2, *samp3
                                          'li1 + li2 の値と同じ順位に書かれた
                                          ' ラベル名のサブルーチンを呼び出します。
  CASE 5              'Index が 5 の場合。
    ON li1 + li2 GOTO *samp1, *samp2, *samp3
                                          'li1 + li2 の値と同じ順位に書かれた
                                          ' ラベルへジャンプします。
  CASE 6              'Index が 6 の場合。
    END               ' プログラムによる動作の終了を宣言します。
END SELECT            ' 複数条件判断文の終了を宣言します。
```

# RETURN（ステートメント）【SLIM 準拠】

**機　能**

サブルーチンから復帰します。

**書　式**

RETURN

**説　明**

GOSUB 文によって制御が移ったサブルーチンの実行を終了し、元のプログラムに戻ります。

**関連項目**

GOSUB

**用　例**

```
DIM li1 As Integer
IF li1 = 0 THEN                 'li1 が 0 の場合。
  STOP                          ' プログラムの実行を終了します。
ELSEIF li1 = 1 THEN             'li1 が 1 の場合。
  GOTO *samp1                   '*samp1 のラベルへジャンプします。
  GO TO *samp2                  '*samp2 のラベルへジャンプします。
ELSEIF li1 = 2 THEN             'li1 が 2 の場合。
  GOSUB *samp3                  ' ラベル名 *samp3 のサブルーチンを呼び出します。
ELSE                            'li1 がその他の場合。
  RETURN                        ' 元のプログラムに戻ります。
END IF                          'IF 文の終了を宣言します。
```

## 11.3　繰り返し

# DO...LOOP（ステートメント）

**機　能**

判定反復（繰り返し）を行ないます。

**書　式**

DO [{WHILE|UNTIL}[< 条件式 >]]
　:
LOOP
または
DO
　:
LOOP [{WHILE|UNTIL}[< 条件式 >]]

**説　明**

DO WHILE と DO UNTIL は、前判定反復です。

LOOP WHILE と LOOP UNTIL は、後判定反復です。

WHILE 文は、条件式が満たされている間 ( 真 (0 以外 )) は繰り返し、UNTIL 文は、条件式が満たされるまで繰り返します。

< 条件式 > の右辺を省略すると値が真 (0 以外 ) かどうか判定します。

< 条件式 > を省略すると条件式を真 (0 以外 ) とみなし、WHILE 文では無限ループになり、UNTIL 文では前判定時には未処理で、後判定時にはループを一度だけ実行します。

DO WHILE ～ LOOP と同じ機能で、WHILE ～ WEND という構文もありますが、DO WHILE ～ LOOP で統一した方がよいでしょう。

DO ～ LOOP UNTIL と同じ機能で、REPEAT ～ UNTIL という構文もありますが、DO ～ LOOP UNTIL で統一した方がよいでしょう。

WHILE および UNTIL は省略が可能ですが、この場合は無限ループとなります。

**関連項目**

EXIT DO、WHILE...WEND、REPEAT...UNTIL、FOR...NEXT

用　例

```
DEFINT li1, li2, li3, li4, li5, li6, li7, li8, li9
DO WHILE li1 > li2                  ' 前判定反復を行ないます。
  IF li1 = 4 THEN EXIT DO           'li1 = 4 の場合、DO ～ LOOP から脱出します。
  FOR li3 = 0 TO 5                  'FOR ～ NEXT の処理を 5 回繰り返します。
    FOR li4 = li5 TO li6            'FOR ～ NEXT の処理を li5 の値を 1 ずつ加算し、li6 にな
                                    ' るまで繰り返します。
      FOR li7 = 1 TO li8 STEP 2     'FOR ～ NEXT の処理を 1 から 2 ずつ加算し、li8 になるま
                                    ' で繰り返します。
        IF li2 = 2 THEN EXIT FOR    'li2 = 0 の場合、FOR ～ NEXT から脱出します。
        DO WHILE li2 < li9          ' 前判定反復を行ないます。
          GOSUB *samp2
          li9 = li9 + 1
        LOOP                        'li2 < li9 になるまで GOSUB *samp2 文を呼びます。
      NEXT li7                      ' 繰り返します。
    NEXT                            ' 繰り返します。
  NEXT                              ' 繰り返します。
  li9 = 0
  DO                                ' 後判定反復を行ないます。
    GOSUB *samp2
    li9 = li9 + 1
  LOOP UNTIL li9 < 5                'li9 < 5 になるまで GOSUB *samp2 文を呼びます。
LOOP                                ' 繰り返します。
```

# EXIT DO（ステートメント）

**機　能**

DO ～ LOOP からの強制脱出を行ないます。

**書　式**

EXIT DO

**説　明**

DO ～ LOOP から強制的に脱出し、DO ～ LOOP 文の次の命令に移ります。

**関連項目**

DO...LOOP

**用　例**

```
DEFINT li1, li2, li3, li4, li5, li6, li7, li8, li9
DO WHILE li1 > li2                  ' 前判定反復を行ないます。
  IF li1 = 4 THEN EXIT DO           'li1 = 4 の場合、DO ～ LOOP から脱出します。
  FOR li3 = 0 TO 5                  'FOR ～ NEXT の処理を 5 回繰り返します。
    FOR li4 = li5 TO li6            'FOR ～ NEXT の処理を li5 の値を 1 ずつ加算し、li6 にな
                                    ' るまで繰り返します。
      FOR li7 = 1 TO li8 STEP 2     'FOR ～ NEXT の処理を 1 から 2 ずつ加算し、li8 になるま
                                    ' で繰り返します。
        IF li2 = 2 THEN EXIT FOR    'li2 = 0 の場合、FOR ～ NEXT から脱出します。
        DO WHILE li2 < li9          ' 前判定反復を行ないます。
          GOSUB *samp2
          li9 = li9 + 1
        LOOP                        'li2 < li9 になるまで GOSUB *samp2 文を呼びます。
      NEXT li7                      ' 繰り返します。
    NEXT                            ' 繰り返します。
  NEXT                              ' 繰り返します。
  li9 = 0
  DO                                ' 後判定反復を行ないます。
    GOSUB *samp2
    li9 = li9 + 1
  LOOP UNTIL li9 < 5                'li9 < 5 になるまで GOSUB *samp2 文を呼びます。
LOOP                                ' 繰り返します。
```

# FOR...NEXT（ステートメント）【SLIM 準拠】

**機　能**

FOR ～ NEXT までの区間中にある一連の命令を繰り返して実行します。

**書　式**

FOR < 変数名 > = < 初期値 > TO < 最終値 > [STEP < 増分 >]
　:
NEXT [< 変数名 >]

**説　明**

FOR ～ NEXT までの区間中にある一連の命令を、FOR 行で指定した条件に従って繰り返して実行します。
< 初期値 > は、< 変数名 > で指定した変数の初期値を設定します。
< 最終値 > は、< 変数名 > で指定した変数の最終値を設定します。
< 増分 > は、初期値と最終値との間の増分を設定します。STEP を省略すると増分は 1 とみなされます。
次の場合には、FOR ～ NEXT は実行されずに、NEXT の次へ実行が移ります。

- < 増分 > が正の値で、< 初期値 > が、< 終値 > より大きい場合。
- < 増分 > が負の値で、< 初期値 > が、< 終値 > より小さい場合。

ただし、< 変数 > には < 初期値 > が代入されます。
１つの FOR ～ NEXT の中にはもう１つの FOR ～ NEXT を置くことができます ( 入れ子構造、ネスト構造等と呼ぶ )。
この場合、それぞれの < 変数名 > には別のものを使わなければなりません。
また、このとき、１つの FOR ～ NEXT は完全に他の FOR ～ NEXT の内部になければなりません。

**注意**

FOR と NEXT は必ず１対１に対応していなければなりません。
また、FOR ～ NEXT ループ内へ外部から GOTO などでジャンプして入ってきたり、逆にループ内から外部へジャンプしたりするようなプログラムは、その動作が保証されなくなります。増分を 0 にした場合は無限ループになります。

**関連項目**

DO...LOOP、EXITFOR

用　例

```
DEFINT li1, li2, li3, li4, li5, li6, li7, li8, li9
DO WHILE li1 > li2                  ' 前判定反復を行ないます。
  IF li1 = 4 THEN EXIT DO           'li1 = 4 の場合、DO ～ LOOP から脱出します。
  FOR li3 = 0 TO 5                  'FOR ～ NEXT の処理を 5 回繰り返します。
    FOR li4 = li5 TO li6            'FOR ～ NEXT の処理を li5 の値を 1 ずつ加算し、li6 にな
                                    ' るまで繰り返します。
      FOR li7 = 1 TO li8 STEP 2     'FOR ～ NEXT の処理を 1 から 2 ずつ加算し、li8 になるま
                                    ' で繰り返します。
        IF li2 = 2 THEN EXIT FOR    'li2 = 0 の場合、FOR ～ NEXT から脱出します。
        DO WHILE li2 < li9          ' 前判定反復を行ないます。
          GOSUB *samp2
          li9 = li9 + 1
        LOOP                        'li2 < li9 になるまで GOSUB *samp2 文を呼びます。
      NEXT li7                      ' 繰り返します。
    NEXT                            ' 繰り返します。
  NEXT                              ' 繰り返します。
  li9 = 0
  DO                                ' 後判定反復を行ないます。
    GOSUB *samp2
    li9 = li9 + 1
  LOOP UNTIL li9 < 5                'li9 < 5 になるまで GOSUB *samp2 文を呼びます。
LOOP                                ' 繰り返します。
```

# EXIT FOR（ステートメント）

**機　能**

FOR ～ NEXT からの強制脱出を行ないます。

**書　式**

EXIT FOR

**説　明**

FOR ～ NEXT から強制的に脱出し，FOR ～ NEXT 文の次の命令に移ります。

**関連項目**

FOR...NEXT

**用　例**

```
DEFINT li1, li2, li3, li4, li5, li6, li7, li8, li9
DO WHILE li1 > li2                ' 前判定反復を行ないます。
  IF li1 = 4 THEN EXIT DO         'li1 = 4 の場合、DO ～ LOOP から脱出します。
  FOR li3 = 0 TO 5                'FOR ～ NEXT の処理を 5 回繰り返します。
    FOR li4 = li5 TO li6          'FOR ～ NEXT の処理を li5 の値を 1 ずつ加算し、li6 にな
                                  ' るまで繰り返します。
      FOR li7 = 1 TO li8 STEP 2   'FOR ～ NEXT の処理を 1 から 2 ずつ加算し、li8 になるま
                                  ' で繰り返します。
        IF li2 = 2 THEN EXIT FOR  'li2 = 0 の場合、FOR ～ NEXT から脱出します。
        DO WHILE li2 < li9        ' 前判定反復を行ないます。
          GOSUB *samp2
          li9 = li9 + 1
        LOOP                      'li2 < li9 になるまで GOSUB *samp2 文を呼びます。
      NEXT li7                    ' 繰り返します。
    NEXT                          ' 繰り返します。
  NEXT                            ' 繰り返します。
  li9 = 0
  DO                              ' 後判定反復を行ないます。
    GOSUB *samp2
    li9 = li9 + 1
  LOOP UNTIL li9 < 5              'li9 < 5 になるまで GOSUB *samp2 文を呼びます。
LOOP                              ' 繰り返します。
```

# REPEAT...UNTIL（ステートメント）

**機　能**

後判定反復を行ないます。

**書　式**

REPEAT

　:

UNTIL [< 条件式 >]

**説　明**

< 条件式 > が満たされるまで、REPEAT 文と UNTIL 文の間を繰り返します。

< 条件式 > を省略すると、そこは真 (0 以外 ) とみなし、ループを一度だけ実行します。

REPEAT ～ UNTIL と同じ機能で、DO ～ LOOP UNTIL という構文があり、DO ～ LOOP UNTIL に統一した方がよいでしょう。

**関連項目**

DO...LOOP、WHILE...WEND

**用　例**

```
DEFINT li1, li2, li3, li4, li5, li6, li7, li8, li9
DO WHILE li1 > li2                  ' 前判定反復を行ないます。
  IF li1 = 4 THEN EXIT DO           'li1 = 4 の場合、DO ～ LOOP から脱出します。
  FOR li3 = 0 TO 5                  'FOR ～ NEXT の処理を 5 回繰り返します。
    FOR li4 = li5 TO li6            'FOR ～ NEXT の処理を li5 の値を 1 ずつ加算し、li6 にな
                                    ' るまで繰り返します。
      FOR li7 = 1 TO li8 STEP 2     'FOR ～ NEXT の処理を 1 から 2 ずつ加算し、li8 になるま
                                    ' で繰り返します。
        IF li2 = 2 THEN EXIT FOR    'li2 = 0 の場合、FOR ～ NEXT から脱出します。
        WHILE li2 < li9             ' 前判定反復を行ないます。
          GOSUB *samp2
          li9 = li9 + 1
        WEND                        'li2 < li9 になるまで GOSUB *samp2 文を呼びます。
      NEXT li7                      ' 繰り返します。
    NEXT                            ' 繰り返します。
  NEXT                              ' 繰り返します。
  li9 = 0
  REPEAT                            ' 後判定反復を行ないます。
    GOSUB *samp2
    li9 = li9 + 1
  UNTIL li9 < 5                     'li9 < 5 になるまで GOSUB *samp2 文を呼びます。
LOOP                                ' 繰り返します。
```

# WHILE...WEND（ステートメント）

**機　能**

前判定反復を行ないます。

**書　式**

WHILE [< 条件式 >]
　:
WEND

**説　明**

< 条件式 > が満たされている間、WHILE 文と WEND 文の間を繰り返します。

< 条件式 > を省略すると、そこは真 (0 以外 ) とみなし、無限にループを繰り返します。

GOTO 文などで WHILE 文と WEND 文の間に分岐すると、WEND 文まで普通に実行し、WHILE 文に戻り、それからはこの構文に普通に入ったときのように実行します。

WHILE ～ WEND と同じ機能で、DO WHILE ～ LOOP という構文があり、DO WHILE ～ LOOP に統一した方がよいでしょう。

**関連項目**

DO...LOOP、REPEAT...UNTIL

**用　例**

```
DEFINT li1, li2, li3, li4, li5, li6, li7, li8, li9
DO WHILE li1 > li2                  ' 前判定反復を行ないます。
  IF li1 = 4 THEN EXIT DO           'li1 = 4 の場合、DO ～ LOOP から脱出します。
  FOR li3 = 0 TO 5                  'FOR ～ NEXT の処理を 5 回繰り返します。
    FOR li4 = li5 TO li6            'FOR ～ NEXT の処理を li5 の値を 1 ずつ加算し、li6 にな
                                    ' るまで繰り返します。
      FOR li7 = 1 TO li8 STEP 2     'FOR ～ NEXT の処理を 1 から 2 ずつ加算し、li8 になるま
                                    ' で繰り返します。
        IF li2 = 2 THEN EXIT FOR    'li2 = 0 の場合、FOR ～ NEXT から脱出します。
        WHILE li2 < li9             ' 前判定反復を行ないます。
          GOSUB *samp2
          li9 = li9 + 1
        WEND                        'li2 < li9 になるまで GOSUB *samp2 文を呼びます。
      NEXT li7                      ' 繰り返します。
    NEXT                            ' 繰り返します。
  NEXT                              ' 繰り返します。
  li9 = 0
  REPEAT                            ' 後判定反復を行ないます。
    GOSUB *samp2
    li9 = li9 + 1
  UNTIL li9 < 5                     'li9 < 5 になるまで GOSUB *samp2 文を呼びます。
LOOP                                ' 繰り返します。
```

## 11.4　条件分岐

# IF...END IF（ステートメント）

**機　能**

IF ～ END IF 間の条件式の条件判断を行ないます。

**書　式**

IF < 条件式 > THEN
  :
[ELSEIF < 条件式 > THEN]
  :
[ELSE]
  :
END IF

**説　明**

< 条件式 > の条件によってプログラムの実行を制御します。

IF 文の < 条件式 > が真 (0 以外 ) ならば IF ～ ELSEIF 文の間の文を実行し、< 条件式 > が偽 (0) ならば ELSEIF 文の < 条件式 > を判断します。同様に、ELSEIF ～ ELSE、ELSE ～ END IF と実行していきます。

**関連項目**

IF...THEN...ELSE

**用　例**

```
DIM li1 As Integer
IF li1 = 0 THEN              'li1 が 0 の場合。
  STOP                       ' プログラムの実行を終了します。
ELSEIF li1 = 1 THEN          'li1 が 1 の場合。
  GOTO *samp1                '*samp1 のラベルへジャンプします。
  GO TO *samp2               '*samp2 のラベルへジャンプします。
ELSEIF li1 = 2 THEN          'li1 が 2 の場合。
  GOSUB *samp3               ' ラベル名 *samp3 のサブルーチンを呼び出します。
ELSE                         'li1 がその他の場合。
  RETURN                     ' 元のプログラムに戻ります。
END IF                       'IF 文の終了を宣言します。
```

# IF...THEN...ELSE（ステートメント）【SLIM 準拠】

**機　能**

論理式の条件判断を行ないます。

**書　式**

IF < 条件式 > THEN {< 文 >|< ラベル名 >} [ELSE {< 文 >|< ラベル名 >}]

**説　明**

< 条件式 > の条件によってプログラムの実行を制御します。

< 条件式 > が真 (0 以外 ) ならば THEN 以降を実行し、< 条件式 > が偽 (0) ならば ELSE 以降が実行されます。

**関連項目**

IF...END IF

**用　例**

```
IF i1 = 0 THEN STOP ELSE GOSUB *samp1  'il が 0 の場合プログラムの実行を終了し、il がその
                                       ' 他の場合、ラベル名 *samp1 のサブルーチンを
                                       ' 呼び出します。
  i1 = i1 + 1                          'il を加算します。
END                                    ' プログラムの終わりを定義します。
*samp1:                                ' サブルーチンのラベルを定義します。
  i0 = 0                               'i0 に 0 を代入します。
RETURN                                 ' 元のプログラムに戻ります。
```

# SELECT CASE（ステートメント）

**機　能**

複数条件判断を行ないます。

**書　式**

SELECT CASE < 式 >
  CASE < 項目 >[,< 項目 >...]
    :
  [CASE ELSE]
END SELECT

**説　明**

< 式 > の値が CASE 文の < 項目 > に一致する場合、その CASE 以降の一連の命令を実行します。

< 式 > には、数式または文字列を指定できます。

< 項目 > では、変数、定数、式および条件式を指定することができます。

条件式は、次のように指定できます。

- < 数式 1> TO < 数式 2>
  < 式 > の結果が < 数式 1> 以上、< 数式 2> 以下であるかどうか調べます。
  文字列の場合は使用できません。
- IS < 比較演算子 >< 数式 >
  < 式 > の結果と < 数式 > の値を比較します。
  文字列の場合、< 比較演算子 > は "=" のみとなります。

CASE ELSE 文は、すべての CASE 文が一致しなかった場合に実行されます。

また、CASE ELSE 文は、END SELECT 文の前に置かなければなりません。

**関連項目**

IF...END IF

用　例

REM 複数条件判断を行ないます。

```
SELECT CASE Index              'Index の値が CASE 文の値と一致した場合、そのコマンドが実行されま
                               ' す。
 CASE 0                        'Index が 0 の場合。
  STOP                         ' プログラムを終了します。
 CASE 1                        'Index が 1 の場合。
  HALT "STOP"                  ' プログラム実行を一時停止します。
 CASE 2                        'Index が 2 の場合。
  HOLD "STOP"                  ' プログラム実行を一時停止します。
 CASE 3                        'Index が 3 の場合。
  STOPEND                      ' 連続起動されたプログラムをサイクル停止します。
 CASE 4                        'Index が 4 の場合。
  ON li1 + li2 GOSUB *samp1, *samp2, *samp3
                                                    'li1 + li2 の値と同じ順位に書かれたラベル
                                                    ' 名のサブルーチンを呼び出します。
 CASE 5                        'Index が 5 の場合。
  ON li1 + li2 GOTO *samp1, *samp2, *samp3
                                                    'li1 + li2 の値と同じ順位に書かれたラベル
                                                    ' へジャンプします。
 CASE 6 TO 8                   'Index が 6 ～ 8 の場合。
  PRINTDBG " 予約 "            ' デバッグウィンドウにメッセージを出力します。
 CASE IS ≧ 9                   'Index が 9 以上の場合。
  END                          ' プログラムによる動作の終了を宣言します。
END SELECT                     ' 複数条件判断文の終了を宣言します。
```

## 11.5　無条件分岐

# GOTO（ステートメント）【SLIM 準拠】

**機　能**

プログラムの分岐を無条件で行ないます。

**書　式**

{GOTO|GO TO}< ラベル名 >

**説　明**

< ラベル名 > で指定したラベルへジャンプして、ジャンプ先から実行を継続します。

GOTO の代わりに GO TO を使うこともできます。

**関連項目**

GOSUB

**用　例**

```
DIM li1 As Integer
IF li1 = 0 THEN                 'li1 が 0 の場合。
  STOP                          ' プログラムの実行を終了します。
ELSEIF li1 = 1 THEN             'li1 が 1 の場合。
  GOTO *samp1                   '*samp1 のラベルへジャンプします。
  GO TO *samp2                  '*samp2 のラベルへジャンプします。
ELSEIF li1 = 2 THEN             'li1 が 2 の場合。
  GOSUB *samp3                  ' ラベル名 *samp3 のサブルーチンを呼び出します。
ELSE                            'li1 がその他の場合。
  RETURN                        ' 元のプログラムに戻ります。
END IF                          'IF 文の終了を宣言します。
```

# ON...GOTO（ステートメント）【SLIM 準拠】

**機　能**

式の値による無条件分岐を行ないます。

**書　式**

ON < 式 > GOTO < ラベル名 > [,< ラベル名 >]…

**説　明**

< 式 > の値と同じ順位に書かれた < ラベル名 > へジャンプして、プログラムの実行を継続します。このときの順位は、左から 1、2、… と数えます。

< 式 > の値は、整数に変換されて処理します。

**注意**

**式の結果が順位を超えていた場合は、何も実行しません。**

**関連項目**

ON...GOSUB

**用　例**

REM 複数条件判断を行ないます。

```
SELECT CASE Index     'Index の値が CASE 文の値と一致した場合、そのコマンドが
                      ' 実行されます。
  CASE 0              'Index が 0 の場合。
    STOP              ' プログラムを終了します。
  CASE 1              'Index が 1 の場合。
    HALT "STOP"       ' プログラム実行を一時停止します。
  CASE 2              'Index が 2 の場合。
    HOLD "STOP"       ' プログラム実行を一時停止します。
  CASE 3              'Index が 3 の場合。
    STOPEND           ' 連続起動されたプログラムをサイクル停止します。
  CASE 4              'Index が 4 の場合。
    ON li1 + li2 GOSUB *samp1, *samp2, *samp3
                                          'li1 + li2 の値と同じ順位に書かれた
                                          ' ラベル名のサブルーチンを呼び出します。
  CASE 5              'Index が 5 の場合。
    ON li1 + li2 GOTO *samp1, *samp2, *samp3
                                          'li1 + li2 の値と同じ順位に書かれた
                                          ' ラベルへジャンプします。
  CASE 6              'Index が 6 の場合。
    END               ' プログラムによる動作の終了を宣言します。
END SELECT            ' 複数条件判断文の終了を宣言します。
```

# 11.6　コメント

## REM（ステートメント）【SLIM 準拠】

**機　能**

コメントの記述をします。

**書　式**

{REM|'}[< コメント >]

**説　明**

プログラムに < コメント > を入れるときに付けます。

REM に続く文字列は、プログラムの実行にまったく影響を与えません。

REM の代わりにシングルクォート (') を使うこともできます。

**用　例**

REM 複数条件判断を行ないます。

```
SELECT CASE Index              'Index の値が CASE 文の値と一致した場合、そのコマンドが
                               ' 実行されます。
  CASE 0                       'Index が 0 の場合。
    STOP                       ' プログラムを終了します。
  CASE 1                       'Index が 1 の場合。
    HALT "STOP"                ' プログラム実行を一時停止します。
  CASE 2                       'Index が 2 の場合。
    HOLD "STOP"                ' プログラム実行を一時停止します。
  CASE 3                       'Index が 3 の場合。
    STOPEND                    ' 連続起動されたプログラムをサイクル停止します。
  CASE 4                       'Index が 4 の場合。
    END                        ' プログラムによる動作の終了を宣言します。
END SELECT                     ' 複数条件判断文の終了を宣言します。
```

# 第 12 章
# ロボット制御文

この章では、ロボットの動作を制御するためのコマンド、ロボット制御文を説明します。

# 第 12 章　ロボット制御文

## 12.1　動作制御

# APPROACH（ステートメント）

機　能

ツール座標系指定の絶対動作を行ないます。

書　式

APPROACH < 補間方法 >, < 基準位置 >,[@< パス開始変位 > ]< アプローチ長 >[,< 動作オプション >][,NEXT]

説　明

< 基準位置 > はポジション型、ジョイント型、同次変換型が使用できます。

**6 軸**

< 基準位置 > からツール座標系の -Z 軸方向へ < アプローチ長 > 分離れた位置へ移動します。

**4 軸**

< 基準位置 > からベース座標系の +Z 軸方向へ < アプローチ長 > 分離れた位置へ移動します。

< 補間方法 > には P(PTP と表記も可)、L どちらかの選択ができます。

| 補間方法 | 意味 |
|---|---|
| P（または PTP と表記） | PTP 制御で移動します。 |
| L | CP 制御で移動します。 |

< パス開始変位 > の値は目標位置を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

省略するとデフォルト値 @0 として処理します。

@0 とすると、エンド動作で動きます。

@P とすると、パス動作で動きます。

@E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

< 動作オプション > には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。

ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードのときは、NEXT オプションは無効になります。

補足説明

APPROACH ステートメントは、MOVE ステートメントによって書き換えることができます。

APPROACH < 補間方法 >,[< パス開始変位 > ]< 基準位置 >,< アプローチ長 > [,< 動作オプション >][,NEXT]

上の APPROACH ステートメントを、MOVE ステートメントで書くと、下のようになります。

6 軸

MOVE < 補間方法 >,[< パス開始変位 > ]< 基準位置 >+(0, 0, -< アプローチ長 >)H[,< 動作オプション >][,NEXT]

例：APPROACH  P, P3, 100　　'MOVE P, P3+(0, 0, -100)H と同じ

例：APPROACH  P, P3, 100　　'MOVE P, P3+(0, 0, -100)H と同じ

4 軸

MOVE <補間方法>,[<パス開始変位> ]<基準位置>+(0, 0, <アプローチ長>),[,<動作オプション>][,NEXT]

例：APPROACH  P, P3, 100　　'MOVE P, P3+(0, 0, 100) と同じ

関連項目

DEPART、SPEED

注意事項

- 基準位置から求めたアプローチ位置がロボットの動作範囲外となることがあります。その場合、エラー 6070 番台 (J* ソフトリミットオーバ、可動範囲外、特異点です ) が発生します。
- アプローチ位置の形態は、基準位置の形態になります。したがって、アプローチ位置が動作範囲外となり、「エラー 667*　ソフトリミットオーバ、可動範囲外 2」が発生する場合があります。この場合は、FIGAPRL、FIGAPRP（「FIGAPRL （P.12-37)」、「FIGAPRP （P.12-39)」参照）を使用して、アプローチ位置の形態を計算するか、備考に示すように MOVE 命令に置き換えて、さらに LETF（10.3 項 参照）にて形態を変更してください。
- CP 動作において、現在とっている形態（操作ガイド「4.1.3　腕・ひじ・手首の形態について」参照）と基準位置の形態が異なる場合、エラー 607F（ロボット形態不一致）が発生します。ただし、ロボット形態が変化する動作が可能な場合は、エラーは発生しません。
- CP 動作において、特異点（操作ガイド「4.1.3　腕・ひじ・手首の形態について [ 2 ] 形態の境界」参照）近傍を通るとき、エラー 6080 番台（指令速度制限オーバ）を発生し、停止することがあります。この場合、スピードを落とすか、最適可搬質量設定モード（p. 4-8 「4.6 最適可搬質量設定機能」参照）を 2 または 3 にして使用してください。それでもエラーが発生する場合は、特異点近傍の軌道を回避してください。
- CP 動作において、アプローチ位置の形態と基準位置の形態が一致せず、ワーニング 601C（形態を変更してください）が発生する場合があります。動作終了時の形態へ基準位置の形態を変更いただくようお願いします（ただし、ワーニングが発生しても動作には影響しません)。

用　例

```
DEFSNG lf1, lf2
DEFPOS lp1, lp2, lp3
```

6 軸

```
APPROACH P, (740, 0, 480, 180, 0, 180, 5), 70
                          '(740, 0, 480, 180, 0, 180) でロボット形態 5 の位置から -Zm 方向に
                          '70mm 離れた位置へ (PTP 制御 ) 移動します。
APPROACH L, lp1, lf1, SPEED = 100
                          'lp1 の位置から -Zm 方向に lf1 離れた位置へ (CP 制御 , 内部速度 =
                          '100%) 移動します。
APPROACH P, lp2, @P lf2, S = 50
                          'lp2 の位置から -Zm 方向に lf2 離れた位置へ (PTP 制御 , 内部速度 =
                          '50%) パス動作します。
APPROACH L, lp3, 80       'lp3 の位置から -Zm 方向に 80mm 離れた位置へ (CP 制御 ) 移動します。
```

4 軸

```
APPROACH P, (100, 200, 300, 45, 1), 70
                          '(100, 200, 300, 45, 1) でロボット形態 1 の位置から +Zb 方向に 70mm
                          ' 離れた位置へ (PTP 制御 ) 移動します。
APPROACH L, lp1, lf1, SPEED = 100
                          'lp1 の位置から +Zb 方向に lf1 離れた位置へ (CP 制御 , 内部速度 =
                          '100%) 移動します。
APPROACH P, lp2, @P lf2, S = 50
                          'lp2 の位置から +Zb 方向に lf2 離れた位置へ (PTP 制御 , 内部速度 =
                          '50%) パス動作します。
APPROACH L, lp3, 80       'lp3 の位置から +Zb 方向に 80mm 離れた位置へ (CP 制御 ) 移動します。
```

# DEPART（ステートメント）

機　能

ツール座標系指定で相対動作を行ないます。

書　式

DEPART < 補間方法 >,[@< パス開始変位 > ]< デパート長 >[,< 動作オプション >][,NEXT]

説　明

6 軸

現在位置からツール座標系の -Z 軸方向へ < デパート長 > 分移動します。

4 軸

現在位置からベース座標系の +Z 軸方向へ < デパート長 > 分移動します。

< 補間方法 > には P(PTP と表記も可)、L どちらかの選択ができます。

| 補間方法 | 意味 |
|---|---|
| P（または PTP と表記） | PTP 制御で移動します。 |
| L | CP 制御で移動します。 |

< パス開始変位 > の値は目標位置を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

省略するとデフォルト値 @0 として処理します。

@0 とすると、エンド動作で動きます。

@P とすると、パス動作で動きます。

@E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

< 動作オプション > には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。

ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードでは、NEXT オプションは無効になります。

補足説明

DEPART ステートメントは、MOVE ステートメントによって書き換えることができます。

DEPART < 補間方法 >,[< パス開始変位量 > ]< デパート長 >[,< 動作オプション >][,NEXT]

上の DEPART ステートメントを、MOVE ステートメントで書くと、下のようになります。

6 軸

MOVE < 補間方法 >,[< パス開始変位量 > ]< 現在位置 >+(0, 0, -< デパート長 >)H[,< 動作オプション >][,NEXT]

例 1：DEPART P, 70　　'MOVE P, P0+(0, 0, -70)H と同じ　　; P0 は現在位置

例 2：DEPART P, @P 70　　'MOVE P, @P P0+(0, 0, -70)H と同じ　　; P0 は現在位置

4 軸

MOVE <補間方法>,[<パス開始変位量> ]<現在位置>+(0, 0, <デパート長>)[,<動作オプション>][,NEXT]

例 1：DEPART P, 70　　'MOVE P, P0+(0, 0, 70) と同じ　　; P0 は現在位置

例 2：DEPART P, @P 70　　'MOVE P, @P P0+(0, 0, 70) と同じ　　; P0 は現在位置

関連項目

APPROACH、SPEED

注意事項

DEPART 動作位置の形態は、動作開始時の形態になります。したがって、DEPART 動作位置が動作範囲外となり、「エラー 667*　ソフトリミットオーバ、可動範囲外 2」が発生する場合があります。この場合は、備考に示すように MOVE 命令に置き換えて、さらに LETF（「10.3 形態」参照）にて形態を変更してください。

用　例

```
DEFSNG lf1, lf2
```

6 軸

```
DEPART P, 70                  ' 現在位置から -Zm 方向に 70mm 離れた位置へ (PTP 制御 )
                              ' 移動します。
DEPART L, lf1, SPEED = 100    ' 現在位置から -Zm 方向に lf1 離れた位置へ (CP 制御 ,
                              'S = 100) 移動します。
DEPART P, lf2, S = 50         ' 現在位置から -Zm 方向に lf2 離れた位置へ (PTP 制御 ,
                              'S = 50) 移動します。
DEPART L, 80                  ' 現在位置から -Zm 方向に 80mm 離れた位置へ (CP 制御 )
                              ' 移動します。
```

4 軸

```
DEPART P, 70                  ' 現在位置から Zb 方向に 70mm 離れた位置へ (PTP 制御 )
                              ' 移動します。
DEPART L, lf1, SPEED = 100    ' 現在位置から Zb 方向に lf1 離れた位置へ (CP 制御 ,
                              'S = 100) 移動します。
DEPART P, lf2, S = 50         ' 現在位置から Zb 方向に lf2 離れた位置へ (PTP 制御 ,
                              'S = 50) 移動します。
DEPART L, 80                  ' 現在位置から Zb 方向に 80mm 離れた位置へ (CP 制御 )
                              ' 移動します。
```

# DRAW（ステートメント）

機　能

ワーク作業座標系指定で相対動作を行ないます。

書　式

DRAW < 補間方法 >,[@< パス開始変位量 > ]< 並進移動量 >[,< 動作オプション >][,NEXT]

説　明

現在位置から < 並進移動量 > 分移動します。

< 補間方法 > には P(PTP と表記も可)、L どちらかの選択ができます。

| 補間方法 | 意味 |
|---|---|
| P（または PTP と表記） | PTP 制御で移動します。 |
| L | CP 制御で移動します。 |

< パス開始変位量 > の値は目標位置を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

省略するとデフォルト値 @0 として処理します。

@0 とすると、エンド動作で動きます。

@P とすると、パス動作で動きます。

@E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

< 動作オプション > には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。

ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードでは、NEXT オプションは無効となります。

補足説明

DRAW ステートメントは、MOVE ステートメントによって書き換えることができます。

DRAW < 補間方法 >,[< パス開始変位 > ]< 並進移動量 >[,< 動作オプション >] [,NEXT]

上の DRAW ステートメントを、MOVE ステートメントで書くと、下のようになります。

MOVE < 補間方法 >,[< パス開始変位 > ]< 現在位置 >+< 並進移動量 >

例：DRAW L, (50, 10, 50)　　'MOVE L, P0+(50, 10, 50) と同じ

**関連項目**

SPEED、TOOL、CHANGEWORK

**注意事項**

DRAW 動作位置の形態は、動作開始時の形態になります。したがって、DRAW 動作位置が動作範囲外となり、「エラー 667*　ソフトリミットオーバ、可動範囲外 2」が発生する場合があります。この場合は、備考に示すように MOVE 命令に置き換えて、さらに LETF（10.3 項 参照）にて形態を変更してください。

**用　例**

```
DEFVEC lv1, lv2
DRAW L, (50, 10, 50)          ' 現在位置から (X = 50, Y = 10, Z = 50) 離れた位置へ
                              ' (CP 制御 ) 移動します。
DRAW L, lv1, SPEED = 90       ' 現在位置から lv1 離れた位置へ (CP 制御 , S = 90) 移動
                              ' します。
DRAW L, lv2, S = 50           ' 現在位置から lv2 離れた位置へ (CP 制御 , S = 50) 移動
                              ' します。
```

# DRIVE（ステートメント）【SLIM 準拠】

**機　能**

各軸の相対動作を行ないます。

**書　式**

DRIVE［＜@パス開始変位＞］（＜軸番号＞，＜相対移動量＞）［,（＜軸番号＞，＜相対移動量＞）…］［,＜動作オプション＞］［，NEXT］

**説　明**

＜軸番号＞で指定した軸を、＜相対移動量＞で指定した角度 (DEG) へ移動させます。＜相対移動量＞が正の場合は、指定軸を＋方向へ、負の場合は－方向へ動作させます。

このコマンドを実行するためには、動作させる軸を含むアームグループの取得が必要です。同じ軸を複数指定した場合は、後に指定したものが有効になります。

< @パス開始変位 > の値は目標位置を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

- 省略するとデフォルト値 @0 として処理します。
- @0 とすると、エンド動作で動きます。
- @P とすると、パス動作で動きます。
- @E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

＜動作オプション＞には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。

ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードでは、NEXT オプションは無効になります。

**関連項目**

DRIVEA、MOVE の EX または EXA オプション

( 付加軸を動かすコマンドは MOVE の EX または EXA オプション、DRIVE、DRIVEA のみです。)

**注意事項**

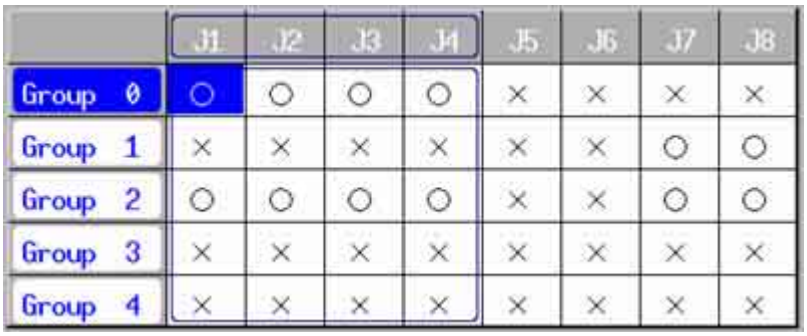
付加軸を指定し、＜@パス開始変位＞オプションに数値を入力した場合、数値に関わらず、パス動作で動作します。

用　例

**例 1**：

```
DEFINT li1, li2, li3
DEFSNG lf1, lf2, lf3
DRIVE (li1, 30)                    ' 現在位置から li1 を 30 度 (deg) 移動します。
DRIVE (li1, lf1)                   ' 現在位置から li1 の値の軸を lf1 値分移動します。
DRIVE (li1, 0.78RAD), (li2, lf2), (li3, lf3)
                                   ' 現在位置から li1 を 0.78(rad)、li2 を lf2
                                   ' が示す値分、li3 を lf3 の値分移動します。
```

**例 2（付加軸の例）**：



| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 0 | ○ | ○ | ○ | ○ | × | × | × | × |
| Group 1 | × | × | × | × | × | × | ○ | ○ |
| Group 2 | ○ | ○ | ○ | ○ | × | × | ○ | ○ |
| Group 3 | × | × | × | × | × | × | × | × |
| Group 4 | × | × | × | × | × | × | × | × |

```
PROGRAM PRO1
TAKEARM 1                          ' アームグループ 1 は 7 軸 8 軸を含んでいる。
DRIVE (7,30),(8,50)                '7 軸 8 軸を動作させる。
END
```

7 軸 8 軸を動作させるためには上の例ではアームグループ 1 またはアームグループ 2 を取得している必要があります。

# DRIVEA（ステートメント）

**機　能**

各軸の絶対動作を行ないます。

**書　式**

DRIVEA［＜@パス開始変位＞］（＜軸番号＞，＜軸座標＞）［，（＜軸番号＞，＜軸座標＞）…］［，＜動作オプション＞］［，NEXT］

**説　明**

＜軸番号＞で指定した軸を、＜軸座標＞で指定した角度 (DEG) へ移動させます。

このコマンドを実行するためには、動作させる軸を含むアームグループの取得が必要です。同じ軸を複数指定した場合は、後に指定したものが有効になります。

< パス開始変位 > の値は目標位置を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

- 省略するとデフォルト値 @0 として処理します。
- @0 とすると、エンド動作で動きます。
- @P とすると、パス動作で動きます。
- @E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

< 動作オプション > には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。

ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードでは、NEXT オプションは無効になります。

**関連項目**

DRIVE、MOVE の EX または EXA オプション

( 付加軸を動かすコマンドは MOVE の EX または EXA オプション、DRIVE、DRIVEA のみです。)

**注意事項**

付加軸を指定し、＜@パス開始変位＞オプションに数値を入力した場合、数値に関わらず、パス動作で動作します。

用　例

**例 1**：

```
DEFINT li1, li2, li3
DEFSNG lf1, lf2, lf3
DRIVEA (li1, 30)                    'li1 を 30 度 (deg) 移動します。
DRIVEA (li1, lf1)                   ' 現在位置から li1 の値の軸を lf1 の値に移動します。
DRIVEA @P (li1, 0.78RAD), (li2, lf2), (li3, lf3)
                                    ' 現在位置から li1 を 0.78(rad)、li2 を lf2 が示す値分、
                                    'li3 を lf3 の値に移動します。
```

**例 2（付加軸の例）**：

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 0 | ○ | ○ | ○ | ○ | × | × | × | × |
| Group 1 | × | × | × | × | × | × | ○ | ○ |
| Group 2 | ○ | ○ | ○ | ○ | × | × | ○ | ○ |
| Group 3 | × | × | × | × | × | × | × | × |
| Group 4 | × | × | × | × | × | × | × | × |

```
PROGRAM PRO1
TAKEARM 1                           ' アームグループ 1 は 7 軸 8 軸を含んでいる。
DRIVE (7,30),(8,50)                 '7 軸 8 軸を動作させる。
END
```

7 軸 8 軸を動作させるためには上の例ではアームグループ 1 またはアームグループ 2 を取得している必要があります。

# GOHOME（ステートメント）【SLIM 準拠】

**機　能**

HOME 文で定義したポジション（原点）へ移動します。

**書　式**

GOHOME

**説　明**

ロボットを、現在位置からホームポジションへ PTP 制御で移動します。
ホームポジションは HOME 文で宣言します。
HOME を設定せずに実行した場合、エラーとなります。

**関連項目**

HOME

**用　例**

```
GOHOME                          ’現在位置からホームポジションへ移動します。
```

# MOVE（ステートメント）【SLIM 準拠】

## 機　能

ロボットを指定座標へ移動します。

EX オプション ( 付加軸相対動作 ) または EXA オプション（付加軸絶対動作）を付けることにより、ロボットと付加軸を同期 ( 同時発着 ) で動作させることができます。

## 書　式

- ＜補間方法＞が自由曲線以外の場合
  MOVE ＜補間方法＞,［@＜パス開始変位＞］＜ポーズ＞［＜ EX または EXA オプション＞］[,［@＜パス開始変位＞］＜ポーズ＞［＜ EX または EXA オプション＞］…］[,＜動作オプション＞］[, NEXT］
- ＜補間方法＞が自由曲線の場合【Ver.2.3 以降】
  MOVE S,［@＜パス開始変位＞］＜軌道番号＞［＜ EX または EXA オプション＞］[,＜動作オプション＞］[, NEXT］

## 説　明

現在位置から指定座標＜ポーズ＞へ移動します。

＜ポーズ＞はポジション型（P 型）、ジョイント型（J 型）、同次変換型（T 型）が使用できます。

- ポジション型は、変数、番号付き変数によるポーズ列、定数および、現在位置（*,CURPOS）が使用できます。
- ジョイント型は、変数、番号付き変数によるポーズ列および、現在角度（CURJNT）が使用できます。
- 同次変換型は、変数、番号付き変数によるポーズ列が使用できます。

ポーズ列の表し方：P[3 TO 6]　　P[3] から P[6] まで。

＜補間方法＞には P、L、C、S の 4 種類の選択ができます。

| 補間方法 | 意味 |
|---|---|
| P（または PTP と表記） | 現在位置から指定座標へ PTP 制御で移動します。 |
| L | 現在位置から指定座標へ CP 制御で移動します。 |
| C | 現在位置から経由ポーズを通り、目的ポーズへ円弧補間をして移動します。<br>姿勢は、現在位置の姿勢から目的ポーズの姿勢へ補間動作します。(経由ポーズの姿勢は無視されます)<br>円弧補間では、経由ポーズ、目的ポーズの 2 点の指示が必要です。<br>(C にはポーズ列は使えません )<br>経由ポーズにパス開始変位を指定しても動作は変化しません。<br>円弧補間動作終了時のパスは、MOVE C, P1, @P P2 のように指定します。 |
| S　自由曲線<br>【Ver.2.3 以降】 | 現在位置から、SETSPLINEPOINT で登録した通過点を通り、最終の通過点へ移動します。軌道は滑らかな曲線になります。ツール端の速度は加減速時を除き、曲線上を一定速度で動作します。<br>姿勢は、通過点毎にパス動作で通過します。 |

＜軌道番号＞は自由曲線の軌道番号です。　SETSPLINEPOINT で＜軸番号＞毎に登録された通過点を通る自由曲線になります。最大 20 まで指定できます。

＜パス開始変位＞の値は指定座標 ( ポーズ ) を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

- 省略するとデフォルト値 @0 として処理します。
- @0 とすると、エンド動作で動きます。

- @P とすると、パス動作で動きます。
- @E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

< 動作オプション > には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。　ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードでは、NEXT オプションは無効になります。

MOVE P,P[3 TO 5] または、MOVE P,P3,P6,P9 のような表現は、それぞれ

MOVE P,P[3 TO 5] →　MOVE P,P3,NEXT
MOVE P,P4,NEXT
MOVE P,P5

MOVE P,P3,P6,P9 →　MOVE P,P3,NEXT
MOVE P,P6,NEXT
MOVE P,P9

と表現したのと同じ意味になります。（処理効率は左側の表現の方が良い）

従って、

- IOBLOCK ON と OFF の間に、これらの表現があったとしても、最終ポーズ（P5 とか P9）の動作が開始されるまで、次の非動作命令は実行されません。「IOBLOCK ON/OFF （P.13-3）」を参照してください。
- NEXT オプションを付けた場合も最終ポーズ（P5, P9）の動作が開始されるまで、次の非動作命令は実行されません。

EX または EXA オプションは下記書式で指定します。

＜ EX または EXA オプション＞の書式

EX（(＜軸番号＞，＜相対移動量＞)［,（＜軸番号＞，＜相対移動量＞）…］)

EXA（(＜軸番号＞，＜軸座標＞)［,（＜軸番号＞，＜軸座標＞）…］)

＜軸番号＞には付加軸のみを指定でき、ロボット軸は指定できません。

**関連項目**

SPEED、DRIVE、DRIVEA

( 付加軸を動かすコマンドは MOVE の EX または EXA オプション、DRIVE、DRIVEA のみです。)

**注意事項**

- ポジション型、同次変換型にてポーズ指定した場合、指定されたポーズがロボットの動作範囲外となることがあります。その場合、エラー 6070 番台 (J* ソフトリミットオーバ、可動範囲外、特異点です ) が発生します。特に 16 ～ 31 の形態を指定した場合はご注意ください。
- CP 動作、円弧補間動作において、現在とっている形態（操作ガイド「4.1.3　腕・ひじ・手首の形態について」参照）と指定座標の形態が異なる場合、エラー 607F（ロボット形態不一致）が発生します。ただし、ロボット形態が変化する動作が可能な場合は、エラーは発生しません。

- CP 動作、円弧補間動作、自由曲線動作において、特異点（操作ガイド「4.1.3　腕・ひじ・手首の形態について [ 2 ] 形態の境界」参照）近傍を通るとき、エラー 6080 番台（指令速度制限オーバ）を発生し、停止することがあります。この場合、スピードを落とすか、最適可搬質量設定モード（p. 4-8「4.6 最適可搬質量設定機能」参照）を 2 または 3 にして使用してください。それでもエラーが発生する場合は、特異点近傍の軌道を回避してください。
- CP 動作において、動作終了時の形態と指定されたポーズの形態が一致せず、ワーニング 601C（形態を変更してください）が発生する場合があります。動作終了時の形態で再度、教示いただくようお願いします（ただし、ワーニングが発生しても動作には影響しません）。
- 円弧補間動作時のパス動作、エンコーダ値確認動作は、目的ポーズにパス開始変位量を指定してください。経由ポーズにパス開始変位量を指定しても、動作は変化しません。
- 円弧補間動作は、経由ポーズの姿勢が無視されます。したがって、ツール端が経由ポーズを通過しない場合があります。ツール端が経由ポーズを通過するようにするには、ツール定義にてツール端にツール座標を設定してください。
- 円弧補間動作にて、前動作がパスの場合、パス動作中に瞬時停止をかけて再起動すると、「エラー 60de 指定した回転動作と異なる動作をします」が発生する場合があります。
- 円弧補間動作において、現在のポーズと目的ポーズが一致している場合、動作しません。また、現在のポーズと経由ポーズが一致している場合、経由ポーズと目的ポーズが一致している場合、目的ポーズへ向けて CP 動作となります。
- 自由曲線動作において、SETSPLINEPOINT　で登録した通過点の形態が現在の形態と異なる場合エラー「607F ロボット形態不一致」が発生します。
- 自由曲線動作において、通過点間の移動距離が短く、姿勢変化が大きい場合、エラー 6080 番台「指令速度制限オーバ」を発生し停止することがあります。この場合、スピードを落とすか、最適可搬質量設定モードを 2 または 3 にしてください。
- 1 つの自由曲線上をパス動作で繰返し動作させたい場合は、その自由曲線を 2 つの軌道番号に登録し、動作毎に軌道番号を切り替えてください。同じ軌道番号の自由曲線をパス動作させるとエラーが発生する場合があります。

**付加軸の注意事項**

- 無限回転の設定になっている軸は EXA オプション ( 絶対動作命令 ) は実行できません。
- ポーズ列を扱う場合は、注意が必要です。
  例：MOVE P,P[3 TO 5] EX((7,30)) は
  MOVE P,P3,NEXT
  MOVE P,P4,NEXT
  MOVE P,P5 EX((7,30))
  と記述したのと同じになります。
  よって 7 軸はロボットが P5 の座標へ移動するのと同期 ( 同時発着 ) します。

用　例

例 1：

```
DIM li1 As Integer
6 軸
MOVE P, (740, 0, 480, 180, 0, 180, 5), NEXT    '(740, 0, 480, 180, 0, 180) でロボット形態 5 の
                                               ' 座標へ (PTP 制御 ) 移動します。
                                               ' 動作開始後、次命令を実行します。
4 軸
MOVE P, (100, 200, 300, 45, 1), NEXT           '(740, 0, 480, 180, 0, 180) でロボット形態 1 の
                                               ' 座標へ (PTP 制御 ) 移動します。
                                               ' 動作開始後、次命令を実行します。
MOVE L, lp1, SPEED = 100                       'lp1 の座標へ (CP 制御 , 内部速度 =100%)
                                               ' 移動します。
MOVE P, @30 lp2, lp3, S = li1                  'lp2(@30)、lp3 の座標へ順に
                                               '(PTP 制御 , 内部速度 = li1) 動作します。
MOVE L, @20 lp4, @50 lp5, @100 lp6             'lp4(@20)、lp5(@50), lp6(@100) の
                                               ' 座標へ順に (CP 制御 ) 動作します。
MOVE L, @P P[6 TO 15], lp7                     'P[6] から P[15] まで順にパス動作で
                                               ' 移動し、lp7 の座標へ (CP 制御 ) 動作
                                               ' します。
MOVE C, lp1, @p lp2                            'lp1 を通り lp2 へ移動する円弧補間
                                               ' 動作をします。lp2 にてパス動作し、
                                               ' 次の制御に移ります。
```

例 2（付加軸の例）：

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 0 | ○ | ○ | ○ | ○ | × | × | × | × |
| Group 1 | × | × | × | × | × | × | ○ | ○ |
| Group 2 | ○ | ○ | ○ | ○ | × | × | ○ | ○ |
| Group 3 | × | × | × | × | × | × | × | × |
| Group 4 | × | × | × | × | × | × | × | × |

```
PROGRAM PRO1
TAKEARM 2                          ' ロボット軸と付加軸を含むアームグループ 2 を取得
MOVE P,P0 EX((7,30),(8,10))        ' ロボットの P0 への移動と、7 軸、8 軸の相対移動
                                   ' が同時発着します。
MOVE P,P1 EXA((7,30))              ' ロボットの P1 への移動と、
                                   '7 軸の絶対移動が同時発着します。
END
```

例 3（自由曲線動作の例）【Ver.2.3 以降】：

```
MOVE S, @P 2, S=10, NEXT           ' 軌道番号 2 の自由曲線を内部速度 10%で
                                   ' パス動作します。動作開始後、
                                   ' 次命令を実行します。
```

# ROTATE（ステートメント）【SLIM 準拠】

## 機　能

指定した軸回りの回転動作を行ないます。

## 書　式

ROTATE < 回転面 >,[@< パス開始変位 >]< 相対回転角 > [,[< 回転中心点 >] [,< 回転オプション >] [,< 動作オプション >][,NEXT]]

回転面の書式

### 6 軸

{{XY|YZ|ZX}|{XYH|YZH|ZXH}|( ベクトル型 , ベクトル型 , ベクトル型 )}

### 4 軸

{{XY}|{XYH}}

## 説　明

< 回転面 > に垂直に作られた軸を中心に < 回転角度 > 分、手先を回転します。回転軸は回転面に対し 2 種類ありますが、アプローチとのなす角が小さい方が選択されます。

< 回転面 > は、以下の 7 種類の中から選択することができます。

回転面の書式

### 6 軸

XY|YZ|ZX は、回転面がワーク座標の XY,YZ,ZX 平面に平行である場合に指定します。

XYH|YZH|ZXH は、回転面がツール座標の XY,YZ,ZX 平面に平行である場合に指定します。

( ベクトル型 , ベクトル型 , ベクトル型 ) は、3 点の XYZ 座標からベース座標基準の平面を作ります。

### 4 軸

XY は、回転面がワーク座標の XY 平面に平行である場合に指定します。

XYH は、回転面がツール座標の XY 平面に平行である場合に指定します。

< パス開始変位 > の値は指定座標 ( ポーズ ) を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

省略するとデフォルト値 @0 として処理します。

@0 とすると、エンド動作で動きます。

@P とすると、パス動作で動きます。

@E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

< 相対回転角 > の単位は度 (DEG) で、符号は右ねじ回りがプラス (+) になります。

### 6 軸

< 回転中心点 > は回転面が {XY|YZ|ZX} のときと（ベクトル型，ベクトル型，ベクトル型）のときは、ワーク座標上の点を指定し、回転面が {XYH|YZH|ZXH} のときは、ツール座標上の点を指定します。

< 回転中心点 > を省略した場合は、回転面が {XY|YZ|ZX} のときは、(0,0,0) が回転中心になり、回転面が ( ベクトル型 , ベクトル型 , ベクトル型 ) のときは、一つ目のベクトルが回転中心になります。ただし、回転面 {XYH|YZH|ZXH} を指定した場合、(0,0,0) を回転中心に動作できません。必ず回転中心点を指定してください。

**4 軸**

< 回転中心点 > は回転面が｛XY｝のときは、ワーク座標上の点を指定し、回転面が｛XYH｝のときは、ツール座標上の点を指定します。

< 回転中心点 > を省略した場合は、回転面が {XY} のときは、(0,0,0) が回転中心になります。ただし、回転面 {XYH} を指定した場合、(0,0,0) を回転中心に動作できません。必ず回転中心点を指定してください。

< 回転オプション > は pose=1、pose=2 で指定します。

pose=1 の場合、ロボットの姿勢は回転中心に対し一定になるように変化します。回転角度は ±540 福 ‹ でとなります。pose=2 の場合、ロボットの姿勢は現在の姿勢を保持します。オプション指定しない場合、pose=2 と同様な動作になります。

< 動作オプション > には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。

ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードのときは、NEXT オプションは無効になります。

**関連項目**

ROTATEH

**注意事項**

パス動作中に瞬時停止をかけ、再起動すると、「エラー 60de　指定した回転動作と異なる動作をします」が発生することがあります。

**用　例**

```
        ROTATE XY, 45, V1              'V1 の点を通る XY 平面に垂直な軸回りに、姿勢固定で
                                       '45 度回転します。
6 軸のみ ROTATE (V2, V3, V4), F1       'V2 の点を通る（V2，V3，V4）の作る平面に垂直な軸回に
                                       'F1 の値分回転します。
        ROTATE XYH, 43, V1, S = 100    ' ツール座標 V1 を通るアプローチベクトル回りに 43 度
                                       ' の回転動作を移動速度 100% で行ないます。
```

# ROTATEH（ステートメント）

**機　能**

アプローチベクトルを軸とした、回転動作を行ないます。

**書　式**

ROTATEH [@< パス開始変位 > ]< アプローチベクトル回りの相対回転角 >[,< 動作オプション >][,NEXT]

**説　明**

アプローチベクトルを軸として < アプローチベクトル回りの相対回転角 > 分、手先を回転します。

< アプローチベクトル回りの相対回転角 > の単位は度 (DEG) です。

ただし、回転角は「-180° ＜ 回転角 ＜ 180° 」で指定してください。

< パス開始変位 > の値は指定座標 ( ポーズ ) を中心とした球の半径で、動作指令値がその中に入ると次の制御へ移ります。mm 単位で指定します。パス開始タイミングを変えるための目安となる数値であり、アーム先端がその中に入ったときに次の制御へ移るわけではありません。

省略するとデフォルト値 @0 として処理します。

@0 とすると、エンド動作で動きます。

@P とすると、パス動作で動きます。

@E とすると、エンコーダ値によって目標位置への到達を確認し、次の動作に移ります。

< 動作オプション > には SPEED、ACCEL、DECEL があります。

| 動作オプション | 意味 |
|---|---|
| SPEED（または S と表記） | 移動速度を指定します。意味は SPEED 文と同じです。 |
| ACCEL | 加速度を指定します。意味は ACCEL 文と同じです。ただし、減速度の指定はできません。減速度の指定は DECEL 文を使ってください。 |
| DECEL | 減速度を指定します。意味は DECEL 文と同じです。 |

<NEXT オプション > を付けると、ロボットの動作完了を待たずに次の非動作命令に続きます。ただし、以下の命令は、ロボットの動作完了（パス開始）まで実行待ちになります。

ロボット動作命令（CHANGETOOL、CHANGEWORK、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL）、最適可搬質量設定ライブラリ（aspACLD、aspChange）、アーム動作ライブラリ（mvSetPulseWidth など）

また、動作オプションと併用した場合、NEXT オプションは無効になります。<NEXT オプション > を付けた場合、ステップ停止を実行すると、次の動作命令実行待ちの場合は、その動作を終了後に停止します。したがって、停止までの移動が長くなりますので注意してください。

また、ティーチチェックモードのときは、NEXT オプションは無効になります。

**補足説明**

ROTATEH ステートメントは、MOVE ステートメントによって書き換えることができます。ROTATEH, [< パス開始変位 >] < 回転角 > [,< 動作オプション >] [,NEXT] の ROTATE ステートメントを MOVE ステートメントで書くと、以下のようになります。

MOVE L, [< パス開始変位 >] < 現在位置 >+(0，0，0，0，0，< 回転角 >) H [, < 動作オプション >] [,NEXT]

例：

ROTATEH @P，F1 'MOVE L，@P PO+(0，0，0，0，0，F1) H と同じ：PO は現在位置

**関連項目**

ROTATE

用　例

```
DEFSNG FF1=50                ' 相対回転角を 50 度とします。
TAKEARM                      ' ロボット制御権を取得します。
MOVE P,P1                    'P1 点へ PTP 移動します。
CHANGETOOL 1                 'TOOL の 1 を有効にします。
ROTATEH @50 FF1              ' パス開始変位を 50、相対回転角を FF1 とします。
CHANGETOOL 0                 'TOOL を 0 にします。
```

# CURJNT（システム変数）

**機　能**

ロボットの現在角度を J 型で得ます。

**書　式**

CURJNT

**説　明**

ロボットの現在の各軸エンコーダで検出した関節角度が J 型データの形式で格納されています。
ロボットが動作中の場合、この命令を実行した時点での値が取得されます。
目標ポーズを取得する場合は、DESTJNT を使ってください。

**関連項目**

DESTJNT、CURPOS、CURTRN

**注意事項**

マシンロック運転中は各軸のエンコーダ値で検出した関節角度ではなく、仮想的な関節角度（指令角度）となります。

**用　例**

```
DIM lj1 As Joint
lj1 = CURJNT                    ' 現在の関節角度を lj1 に代入します。
```

6 軸

```
lj1 = CURJNT + (10, 10, 0, 0, 0, 10)
                                ' 現在の関節角度 + (10, 10, 0, 0, 0, 10) を lj1 に
                                ' 代入します。
```

4 軸

```
lj1 = CURJNT + (10, 10, 0, 0)
                                ' 現在の関節角度 + (10, 10, 0, 0) を lj1 に代入します。
```

# CURPOS（システム変数）【SLIM 準拠】

**機　能**

ツール座標系での現在位置を P 型データで得ます。

**書　式**

{CURPOS|*}

**説　明**

各軸エンコーダの検出値から検出される現在のツール座標系での位置が P 型データの形式で格納されています。

ロボットが動作中の場合、この命令を実行した時点での値が取得されます。

目標ポーズを取得する場合は、DESTPOS を使ってください。

**関連項目**

DESTPOS、CURJNT、CURTRN

**注意事項**

マシンロック運転中は各軸のエンコーダ値で検出した関節角度ではなく、仮想的な関節角度（指令角度）となります。

**用　例**

```
DEFPOS lp1, lp2
lp1 = CURPOS                    ' 現在のツール座標系での位置を lp1 に代入します。
lp2 = *                         ' 現在のツール座標系での位置を lp2 に代入します。
```

6 軸

```
lp1 = CURPOS + (100, 200, 0, 10, 10, 0)
                                ' 現在のツール座標系での位置 + (100, 200, 0, 10, 10,
                                '0) を lp1 に代入します。
```

4 軸

```
lp1 = CURPOS + (100, 200, 0, 10)
                                ' 現在のツール座標系での位置 + (100, 200, 0, 10) を
                                'lp1 に代入します。
```

# CURTRN（システム変数）【SLIM 準拠】

**機　能**

ツール座標系での現在位置を T 型データで得ます。

**書　式**

CURTRN

**説　明**

各軸エンコーダの検出値から検出される現在のツール座標系での位置が T 型データの形式で格納されています。

ロボットが動作中の場合、この命令を実行した時点での値が取得されます。

目標ポーズを取得する場合は、DESTTRN を使ってください。

**関連項目**

DESTTRN、CURJNT、CURPOS

**注意事項**

マシンロック運転中は各軸のエンコーダ値で検出した関節角度ではなく、仮想的な関節角度（指令角度）となります。

**用　例**

```
DEFTRN l+1, l+2
l+1 = CURTRN                    ' 現在のツール座標系での位置を l+1 に代入します。
l+2 = *                         ' 現在のツール座標系での位置を l+2 に代入します。
```

# CUREXJ（ステートメント）

**機　能**

付加軸の現在角度を F 型で取得します。

**書　式**

CUREXJ( ＜軸番号＞ )

**説　明**

＜軸番号＞で指定した付加軸のエンコーダで検出した角度を F 型変数に書き込みます。指定した付加軸が動作中の場合、この命令を実行した時点での値が取得されます。

目標位置を取得する場合は、DESTEXJ を使ってください。

**関連項目**

CURJNT、CURPOS、CURTRN、DESTEXJ

**注意事項**

マシンロック運転中はエンコーダ値で検出した角度でなく、仮想的な角度（指令角度）となります。

**用　例**

```
PROGRAM PRO1
DIM  lf1  AS  SINGLE
TAKEARM  1
DRIVEA(7,100)                  '7 軸を 100 度の位置へ動かします。
lf1 = CUREXJ(7)                '7 軸の現在位置を lf1 に代入します。
END
```

# DESTJNT（システム変数）

**機　能**

現在の動作命令目標位置を J 型で取得します。

ロボット停止時は、現在の位置（指令値）を取得します。

**書　式**

DESTJNT

**説　明**

直前の動作命令の目標位置が、J 型データの形式で格納されています。システム変数で取得できるので、ローカル変数やグローバル変数などで、他のプログラムとデータの受け渡しを行なう必要がなくなります。

各軸エンコーダの検出値から検出された位置を取得する場合は、CURJNT を使ってください。

**関連項目**

CURJNT、DESTPOS、DESTTRN

**注意事項**

動作停止命令（「INTERRUPT ON/OFF （P.12-43）」参照）が入力されて動作が停止したときは、停止位置が取り込まれます。

例：

```
INTERRUPT ON
MOVE P, J1　←動作中に割り込み信号 ON
INTERRUPT OFF
J2=DESTJNT
```

J1=J2 にはなりません。J2 は停止位置になります。

**用　例**

```
DEFJNT lj1, lj2
MOVE P, @P lj1, NEXT
```

**6 軸**

```
lj2 = DESTJNT+(100, 0, 0, 0, 0, 0)　’この場合 DESTJNT = lj1 です。
```

**4 軸**

```
lj2 = DESTJNT+(100, 0, 0, 0)　　’この場合 DESTJNT = lj1 です。
```

# DESTPOS（システム変数）

**機　能**

現在の動作命令目標位置を P 型で取得します。
ロボット停止時は、現在の位置（指令値）を取得します。

**書　式**

DESTPOS

**説　明**

直前の動作命令の目標位置が、P 型データの形式で格納されています。システム変数で取得できるので、ローカル変数やグローバル変数などで、他のプログラムとデータの受け渡しを行なう必要がなくなります。
各軸エンコーダの検出値から検出された位置を取得する場合は、CURPOS を使ってください。

**関連項目**

CURPOS、DESTJNT、DESTTRN

**注意事項**

動作停止命令（「INTERRUPT ON/OFF （P.12-43）」参照）が入力されて動作が停止したときは、停止位置が取り込まれます。
例：

```
INTERRUPT ON
MOVE P, P1        ←動作中に割り込み信号 ON
INTERRUPT OFF
P2=DESTPOS
```

P1=P2 にはなりません。P2 は停止位置になります。

**用　例**

```
DEFPOS lp1, lp2
MOVE P, @P lp1, NEXT
```

6 軸

```
lp2 = DESTPOS+(100, 10, 10, 0, 0, 0)  ’この場合 DESTPOS = lp1 です。
```

4 軸

```
lp2 = DESTPOS+(100, 10, 10, 0)        ’この場合 DESTPOS = lp1 です。
```

# DESTTRN（システム変数）

**機　能**

現在の動作命令目標位置を T 型で取得します。

ロボット停止時は、現在の位置（指令値）を取得します。

**書　式**

DESTTRN

**説　明**

直前の動作命令の目標位置が、T 型データの形式で格納されています。システム変数で取得できるので、ローカル変数やグローバル変数などで、他のプログラムとデータの受け渡しを行なう必要がなくなります。

各軸エンコーダの検出値から検出された位置を取得する場合は、CURTRN を使ってください。

**関連項目**

CURTRN、DESTJNT、DESTPOS

**注意事項**

動作停止命令（「INTERRUPT ON/OFF （P.12-43)」参照）が入力されて動作が停止したときは、停止位置が取り込まれます。

例：

```
INTERRUPT ON
MOVE P, T1          ←動作中に割り込み信号 ON
INTERRUPT OFF
T2=DESTTRN
```

T1=T2 にはなりません。T2 は停止位置になります。

**用　例**

```
DEFPOS lp1, lp2
MOVE P, @P lp1, NEXT
lp2 = DESTTRN+(100, 10, 10) ’この場合 DESTTRN = lp1 です。
```

# DESTEXJ（ステートメント）

**機　能**

付加軸の、現在の動作命令で指定された目標位置を F 型で取得します。停止時は、現在の位置（指令値）を取得します。

**書　式**

DESTEXJ( ＜軸番号＞ )

**説　明**

＜軸番号＞で指定した付加軸の、直前の動作命令で指定された目標位置を F 型変数に書き込みます。

各軸エンコーダの検出値から検出された位置を取得する場合は、CUREXJ を使ってください。

**関連項目**

CUREXJ、DESTJNT、DESTPOS、DESTTRN

**注意事項**

動作停止命令（「INTERRUPT ON/OFF （P.12-43)」参照）が入力されて動作が停止した時は、停止位置が取り込まれます。

例：

```
INTERRUPT ON
DRIVEA (7,100)      ' 動作中に割り込み信号 ON、7 軸は停止して次に進む。
INTERRUPT OFF
F1= DESTEXJ(7)      'F1 は 100 にはなりません。停止位置が F1 に代入されます。
```

**用　例**

```
PROGRAM PRO1
DIM  lf1  AS  SINGLE
TAKEARM  1
DRIVEA(7,100),NEXT          '7 軸を 100 度の位置へ動かします。
                            'NEXT で動作完了前に次行に進みます。
lf1 = DESTEXJ(7)            ' 直前の動作命令の目標位置である 100 を
                            'lf1 に代入します。
END
```

# ARRIVE（ステートメント）【Ver.1.2 以降】

**機　能**

動作命令の全移動距離に対する動作割合を設定する事によって、ロボットが設定した動作割合に到達するまでプログラムを待機させます。

**書　式**

ARRIVE ＜動作割合＞

**説　明**

通常の動作命令では動作完了あるいはパス開始まで次ステップの命令を実行することができませんが、IOBLOCK 指定や NEXT オプション指定の動作命令では動作途中で次ステップの命令を実行する事が可能です。この場合、ARRIVE 命令によってロボットが設定した動作割合に到達するまで次ステップの命令を待機させる事ができます。

＜動作割合＞は動作命令の全移動距離に対する割合を示し、１～99（単位は%）の範囲で、数値、あるいは整数型（I 型)、実数型（F 型）変数にて指定します。

この機能はロボットに対してのみ有効です。よってDRIVE、DRIVEAで付加軸のみを動作させる場合、ARRIVE機能は働きません。

**注意事項**

- ARRIVE 命令はアームセマフォを取得した（TAKEARM した）タスクにおいて、直前の動作命令に対する動作割合を設定します。
- アームセマフォを取得した後、動作命令を実行する前に ARRIVE 命令を実行した場合はエラー 648C になります。
- アームセマフォを取得せずに ARRIVE 命令を実行した場合はエラー 21F7 になります。
- ARRIVE 命令実行中に瞬時停止した場合、再起動時に瞬時停止した位置から 10mm 以上ずれている場合はエラー 6486 になります。
- ARRIVE 命令実行中に瞬時停止した場合、再起動時に動作命令が実行されず＜動作割合＞に満たない場合はエラー 6489 になります。
- ＜動作割合＞はロボット移動距離に対する割合を示します。しかし、＜動作割合＞が一定であっても速度・加速度等の動作条件によって ARRIVE が終了するロボット位置が変動する場合があります。また＜動作割合＞を多少変動させても、ARRIVE が終了するロボット位置が変わらない場合もあります。ARRIVE 命令にてロボットと周辺機器との同期をとる場合は、タイミングをご確認ください。
- 電流制限機能と併用する場合は、電流制限によってロボット位置が＜動作割合＞を通過できなくなる可能性があります。この時、プログラムは ARRIVE 命令が完了するのを無限に待ち続けていますので、プログラム停止を実行して復帰させてください。
- INTERRUPT と併用する場合、INTERRUPT ON 後の動作命令は割り込みスキップ信号によって実行されなくなりますが、ARRIVE 命令は実行されますので、割り込みスキップ信号のタイミングによっては、意図した動作命令とは異なる動作命令に対して ARRIVE 命令が実行されることがあります。また割り込みスキップ信号のタイミングにより、ロボット位置が＜動作割合＞を通過できなくなる可能性があります。この時、プログラムは ARRIVE 命令が完了するのを無限に待ち続けていますので、プログラム停止を実行して復帰させてください。
- ティーチチェックでは NEXT オプション、IOBLOCK 命令は無効となりますので、ARRVIE 命令は意味を持ちません。

- ARRVIE 命令の対象となる動作命令は、＜動作オプション＞を使用できません。
  ＜動作オプション＞を使用した場合、ARRVIE 命令は無効になります。
  ARRVIE 命令の対象となる動作命令の内部速度・外部加減速度を使用する場合は次の例のようにしてください。
  ＜例＞

```
PROGRAM PRO1
TAKEARM
SPEED 10          ' 内部 SP 10
MOVE P,P1,NEXT
ARRIVE 50
SET IO [240]      ' 動作割合が 50%になったら、I/O を ON
SPEED 100         ' 内部スピードを元の値に戻す
END
```

用　例

**例 1**：

```
PROGRAM PRO1
TAKEARM
MOVE P, P1, NEXT
ARRIVE 50                    ' 動作割合が 50% になったら
SET IO [240]                 'IO [240] を ON します。
ARRIVE I1                    ' 動作割合が I1% になったら
RESET IO [240]               'IO [240] を OFF します。
END
```

**例 2（付加軸の例）**：
4 軸ロボットで次のようなアームグループが設定されている場合。

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | ○ | ○ | ○ | ○ | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1
DRIVE(1,30),(7,30),NEXT
ARRIVE 50 ←  付加軸との同期動作もロボット軸を含んでいるので、ARRIVE は機能する。
SET IO [240]                 ' IO [240] を ON します。
MOVE P,P0 EX((7,30)),NEXT
ARRIVE 50 ←  直前の動作命令が、ロボット軸を含んでいるため ARRIVE は機能する。
RESET IO [240]               ' IO [240] を OFF します。
DRIVE(7,30),NEXT
 ARRIVE 50 ←  直前の動作命令が、付加軸のみであるため、ARRIVE は機能せず次に進む。
SET IO [240]                 ' IO [240] を ON します。
END
```

# POSCLR（ステートメント）【Ver.1.5 以降】

**機　能**

軸の現在位置を強制的に 0mm または 0 度にします。

**書　式**

POSCLR ＜軸番号＞

**説　明**

＜軸番号＞で指定した軸の角度を、強制的に 0mm または 0 度にします。無限回転の設定になっている軸のみ実行可能です。

同じ方向に回転させつづけて、現在位置の値が大きくなり扱い難い場合、また現在値の値がマイナスの大きな値に飛んでしまった場合 ( 同じ方向に回転させ続けた場合、位置が飛ぶことがある ) などに用います。

このコマンドを実行するためには、位置をクリアする軸を含んだアームグループの取得が必要です。

**注意事項**

- ロボット軸は実行できません。（付加軸のみ）
- POSCLR 実行時はロボットが完全に停止するのを待ちます。したがって、POSCLR の前にパス動作指定をしてもパス動作しません。
- ステップ戻し機能で POSCLR 実行以前には、戻すことはできません。

**用　例**

```
PROGRAM PRO1
TAKEARM 1
DRIVEA(7,100)                ’7 軸を 100 度の位置へ動かします。
POSCLR 7                     ’7 軸の現在位置を強制的に 0 度とします。
END
```

# SETSPLINEPOINT（ステートメント）【Ver.2.3 以降】

**機　能**

自由曲線動作の通過点を登録します。

**書　式**

SETSPLINEPOINT ＜自由曲線の軌道番号＞、＜通過点＞

**説　明**

＜自由曲線の軌道番号＞で指定した自由曲線軌道の通過点として、＜通過点＞にて指定したポイントを登録します。　＜通過点＞は P 型、J 型変数になります。

**関連項目**

CLRSPLINEPOINT、GETSPLINEPOINT、MOVE

**注意事項**

- 1 つの自由曲線で登録可能な通過点の数は、最大で 200 点です。それ以上登録した場合、エラー「685A 自由曲線登録データ数オーバ」が発生します。
- 通過点未登録の自由曲線動作を実行した場合、エラー「685B 自由曲線番号指定異常」が発生します。
- 軌道番号は、20 まで指定できます。20 を超えた番号を指定した場合、エラー「685B 自由曲線番号指定異常」が発生します。
- 指定した通過点が動作範囲外の場合、エラー 6070 番台（J* ソフトリミットオーバ、可動範囲外）が発生します。
- 通過点教示時にツール座標、ワーク座標を設定した場合、本コマンド実行の前に CHANGETOOL コマンド、CHANGEWORK コマンドを実行し、教示時と同じツール座標、ワーク座標に変更してください。

**用　例**

```
PROGRAM PRO1
TAKEARM
CLRSPLINEPOINT 5              ' 軌道番号 5 番の自由曲線で通過点をすべてクリアする。
SETSPLINEPOINT 5, P4          ' 軌道番号 5 番の自由曲線で P4 を 1 番目の通過とする。
SETSPLINEPOINT 5, P1          ' 軌道番号 5 番の自由曲線で P1 を 2 番目の通過点とする。
SETSPLINEPOINT 5, J5          ' 軌道番号 5 番の自由曲線で J5 を 3 番目の通過点をする。
MOVE S, 5                     'P4,P1 を通過し J5 へ移動する自由曲線動作を実行する。
```

# CLRSPLINEPOINT（ステートメント）【Ver.2.3 以降】

**機　能**

自由曲線動作の通過点をすべて消去します。

**書　式**

CLRSPLINEPOINT ＜自由曲線の軌道番号＞

**説　明**

＜自由曲線の軌道番号＞で指定した自由曲線軌道にて、SETSPLINEPOINT で登録した通過点をすべて消去します。

**関連項目**

SETSPLINEPOINT、GETSPLINEPOINT

**注意事項**

CLRSPLINEPOINT を実行すると通過点未登録になります。通過点未登録の自由曲線動作を実行した場合、エラー「685B 自由曲線番号指定異常」が発生します。

**用　例**

```
PROGRAM PRO1
TAKEARM
CLRSPLINEPOINT 5            ' 軌道番号 5 番の自由曲線で通過点をすべてクリアする。
SETSPLINEPOINT 5, P4        ' 軌道番号 5 番の自由曲線で P4 を 1 番目の通過とする。
SETSPLINEPOINT 5, P1        ' 軌道番号 5 番の自由曲線で P1 を 2 番目の通過点とする。
SETSPLINEPOINT 5, J5        ' 軌道番号 5 番の自由曲線で J5 を 3 番目の通過点をする。
MOVE S, 5                   'P4,P1 を通過し J5 へ移動する自由曲線動作を実行する。
```

# GETSPLINEPOINT（ステートメント）【Ver.2.3 以降】

**機　能**

登録済の自由曲線動作の通過点を取り出します。

**書　式**

＜通過点＞＝ GETSPLINEPOINT (＜自由曲線の軌道番号＞、＜通過点番号＞)

**説　明**

＜自由曲線の軌道番号＞で指定した自由曲線軌道にて、＜通過点番号＞で指定した通過点のポイントを＜通過点＞に出力します。＜通過点＞は P 型変数になります。

**関連項目**

CLRSPLINEPOINT、SETSPLINEPOINT

**注意事項**

未登録の通過点番号を指定した場合、エラー「685C 自由曲線ポイント指定異常」が発生します。

**用　例**

```
PROGRAM PRO1
TAKEARM
CLRSPLINEPOINT 5              ’軌道番号 5 番の自由曲線で通過点をすべてクリアする。
SETSPLINEPOINT 5, P4          ’軌道番号 5 番の自由曲線で P4 を 1 番目の通過とする。
SETSPLINEPOINT 5, P1          ’軌道番号 5 番の自由曲線で P1 を 2 番目の通過点とする。
SETSPLINEPOINT 5, J5          ’軌道番号 5 番の自由曲線で J5 を 3 番目の通過点をする。
P10=GETSPLINEPOINT(5,2)       ’軌道番号 5 番の自由曲線の 2 番目の通過点を P10 にセットする。
                              ’（P10 に P1 のデータが入ります）
```

## 12.2　形態制御

# CURFIG（システム変数）

**機　能**

ロボット形態の現在値を得ます。

**書　式**

CURFIG

**説　明**

現在のロボットの形態が整数で格納されています。整数値と形態の関係は下表のとおりです。

6 軸

| 値 | 形　態 |
|---|---|
| 0 | SINGLE4―SINGLE6―FLIP―ABOVE―RIGHTY |
| 1 | SINGLE4―SINGLE6―FLIP―ABOVE―LEFTY |
| 2 | SINGLE4―SINGLE6―FLIP―BELOW―RIGHTY |
| 3 | SINGLE4―SINGLE6―FLIP―BELOW―LEFTY |
| 4 | SINGLE4―SINGLE6―NONFLIP―ABOVE―RIGHTY |
| 5 | SINGLE4―SINGLE6―NONFLIP―ABOVE―LEFTY |
| 6 | SINGLE4―SINGLE6―NONFLIP―BELOW―RIGHTY |
| 7 | SINGLE4―SINGLE6―NONFLIP―BELOW―LEFTY |
| 8 | SINGLE4―DOUBLE6―FLIP―ABOVE―RIGHTY |
| 9 | SINGLE4―DOUBLE6―FLIP―ABOVE―LEFTY |
| 10 | SINGLE4―DOUBLE6―FLIP―BELOW―RIGHTY |
| 11 | SINGLE4―DOUBLE6―FLIP―BELOW―LEFTY |
| 12 | SINGLE4―DOUBLE6―NONFLIP―ABOVE―RIGHTY |
| 13 | SINGLE4―DOUBLE6―NONFLIP―ABOVE―LEFTY |
| 14 | SINGLE4―DOUBLE6―NONFLIP―BELOW―RIGHTY |
| 15 | SINGLE4―DOUBLE6―NONFLIP―BELOW―LEFTY |
| 16 | DOUBLE4―SINGLE6―FLIP―ABOVE―RIGHTY |
| 17 | DOUBLE4―SINGLE6―FLIP―ABOVE―LEFTY |
| 18 | DOUBLE4―SINGLE6―FLIP―BELOW―RIGHTY |
| 19 | DOUBLE4―SINGLE6―FLIP―BELOW―LEFTY |
| 20 | DOUBLE4―SINGLE6―NONFLIP―ABOVE―RIGHTY |
| 21 | DOUBLE4―SINGLE6―NONFLIP―ABOVE―LEFTY |
| 22 | DOUBLE4―SINGLE6―NONFLIP―BELOW―RIGHTY |
| 23 | DOUBLE4―SINGLE6―NONFLIP―BELOW―LEFTY |
| 24 | DOUBLE4―DOUBLE6―FLIP―ABOVE―RIGHTY |
| 25 | DOUBLE4―DOUBLE6―FLIP―ABOVE―LEFTY |
| 26 | DOUBLE4―DOUBLE6―FLIP―BELOW―RIGHTY |
| 27 | DOUBLE4―DOUBLE6―FLIP―BELOW―LEFTY |
| 28 | DOUBLE4―DOUBLE6―NONFLIP―ABOVE―RIGHTY |
| 29 | DOUBLE4―DOUBLE6―NONFLIP―ABOVE―LEFTY |

| 値 | 形　態 |
|---|---|
| 30 | DOUBLE4―DOUBLE6―NONFLIP―BELOW―RIGHTY |
| 31 | DOUBLE4―DOUBLE6―NONFLIP―BELOW―LEFTY |

4 軸

| 値 | 形　態 |
|---|---|
| 0 | SINGLE―RIGHTY |
| 1 | SINGLE―LEFTY |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | DOUBLE―RIGHTY |
| 9 | DOUBLE―LEFTY |
| 10 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | |
| 16 | |
| 17 | |
| 18 | |
| 19 | |
| 20 | |
| 21 | |
| 22 | |
| 23 | |
| 24 | |
| 25 | |
| 26 | |
| 27 | |
| 28 | |
| 29 | |
| 30 | |
| 31 | |

関連項目

LETF、ロボット形態 ( 付録 2)

用　例

```
DIM li1 As Integer
li1 = CURFIG                    ’現在のロボットの形態を整数型変数 li1 に代入します。
```

# FIGAPRL（関数）

**機　能**

CP 動作可能なアプローチ位置と基準位置の形態を計算します。

**書　式**

FIGAPRL( ＜基準位置＞ , ＜アプローチ長＞ )

**説　明**

＜基準位置＞は、ポジション型のみ使用できます。

**6 軸**

＜基準位置＞からハンド座標系の -Z 軸方向へ＜アプローチ長＞分離れた位置（アプローチ位置）から＜基準位置＞へ CP 動作可能な＜基準位置＞の形態を計算します。

アプローチ位置から＜基準位置＞へ CP 動作可能な形態は、以下の順位で移動可能となるまで自動探査します。

1) ＜基準位置＞の形態（＜基準位置＞の形態が不定の場合は現在の形態）
2) 手首形態（フリップ―ノンフリップ）を入れ替える。
3) 手首形態ノンフリップでひじ形態（アバッブ―ビロー）を入れ替える。
4) 手首形態をフリップにする。
5) ひじ形態アバッブ、手首形態ノンフリップで腕形態（レフティー―ライティー）を入れ替える。
6) 手首形態をフリップにする。
7) ひじ形態ビロー、手首形態ノンフリップにする。
8) 手首形態をフリップにする。

8 番目の探索でも移動不可能な場合は、エラー 6070 番台（J* ソフトリミット　　オーバ）を発生します。この場合は、現在位置またはアプローチ長を変えるなどの処置を行なってください。

**4 軸**

＜基準位置＞からベース座標系の +Z 軸方向へ＜アプローチ長＞分離れた位置（アプローチ位置）から＜基準位置＞へ CP 動作可能な＜基準位置＞の形態を計算します。

アプローチ位置から＜基準位置＞へ CP 動作可能な形態は、以下の順位で移動可能となるまで自動探査します。

1) ＜基準位置＞の形態（＜基準位置＞の形態が不定の場合は現在の形態）

**関連項目**

FIGAPRP、APPROACH

注意事項

- FIGAPRL は、アプローチ後の移動動作が CP 動作の場合に使用してください。PTP 動作の場合は FIGAPRP を使用してください。
- FIGAPRL で使用する＜基準位置＞、アプローチ長は、APPROACH 動作時に指定する＜基準位置＞、アプローチ長と同一にしてください。＜基準位置＞、アプローチ長が異なる場合は、アプローチ後の CP 動作時に 6070 番台のエラーが発生する場合があります。
- FIGAPRL にて形態を求めた場合でも、アプローチ後の CP 動作でエラー 6080 番台（指令速度制限オーバ）を発生し、停止することがあります。この場合、スピードを落とすか、最適可搬質量設定モード（p. 4-8「4.6 最適可搬質量設定機能」参照）を、2 または 3 にしてください。それでもエラーが発生する場合は、アプローチ長を調整してください。

用　例

一般

```
I1=FIGAPRL(P1, 100.0)
LETF  P1=I1
APPROACH  P, P1, 100.0
MOVE  L, P1
```

6 軸

```
I1=FIGAPRL(P1 + (100, 200, 0, 0, 10, 0),
100)
LETF  P1=I1
APPROACH  P, P1, 100.0
MOVE  L, P1
```

4 軸

```
I1=FIGAPRL(P1 + (100, 200, 0, 0), 100)
LETF  P1=I1
APPROACH  P, P1, 100.0
MOVE  L, P1
```

# FIGAPRP（関数）

**機　能**

PTP 動作可能なアプローチ位置と基準位置の形態を計算します。

**書　式**

FIGAPRP( ＜基準位置＞ , ＜アプローチ長＞ )

**説　明**

＜基準位置＞は、ポジション型のみ使用できます。

**6 軸**

＜基準位置＞からハンド座標系の -Z 軸方向へ＜アプローチ長＞分離れた位置（アプローチ位置）から＜基準位置＞へ PTP 動作可能な＜基準位置＞の形態を計算します。

アプローチ位置から＜基準位置＞へ PTP 動作可能な形態は、以下の順位で移動可能となるまで自動探査します。

1) ＜基準位置＞の形態（＜基準位置＞の形態が不定の場合は現在の形態）
2) 手首形態（フリップ－ノンフリップ）を入れ替え、アプローチ位置から
　＜基準位置＞への 4 軸の移動角度が ±90 度以内なら動作可能とする。

4 軸の移動角度が ±90 度を超えたり、移動不可能な場合は、エラー 6070 番台（J* ソフトリミットオーバ）を発生します。この場合は、現在位置またはアプローチ長を変えるなどの処置を行なってください。

**4 軸**

＜基準位置＞からベース座標系の +Z 軸方向へ＜アプローチ長＞分離れた位置（アプローチ位置）から＜基準位置＞へ PTP 動作可能な＜基準位置＞の形態を計算します。

アプローチ位置から＜基準位置＞へ PTP 動作可能な形態は、以下の順位で移動可能となるまで自動探査します。

1) ＜基準位置＞の形態（＜基準位置＞の形態が不定の場合は現在の形態）

**関連項目**

FIGAPRL、APPROACH

**注意事項**

- FIGAPRP は、アプローチ後の移動動作が PTP 動作の場合に使用してください。CP 動作の場合は FIGAPRL を使用してください。
- FIGAPRP で使用する＜基準位置＞、アプローチ長は、APPROACH 動作時に指定する＜基準位置＞、アプローチ長と同一にしてください。＜基準位置＞、アプローチ長が異なる場合は、アプローチ後の PTP 動作時に 6070 番台のエラーが発生する場合があります。

用　例

```
I1=FIGAPRL(P1, 100.0)
LETF  P1=I1
APPROACH  P, P1, 100.0
MOVE  L, P1
```

6 軸

```
I1 = FIGAPRP (P1 + (100, 200, 0, 0, 0, 10), 100, 0)
```

4 軸

```
I1 = FIGAPRP (P1 + (100, 200, 0, 0), 100, 0)
```

## 12.3　停止制御

# HOLD（ステートメント）【SLIM 準拠】

**機　能**

プログラムの実行を一時停止します。

**書　式**

HOLD＜メッセージ＞

**説　明**

プログラムの実行を一時停止し、ステップ停止状態にして＜メッセージ＞に指定した内容をティーチングペンダントへ出力します。＜メッセージ＞の内容に関しては、PRINTMSG 文を参照してください。

再起動すると、プログラムは HOLD 文の次の文から実行します

メッセージの文字数は半角で 60 文字です。

**関連項目**

DELAY、HALT、STOP

**注意事項**

メッセージとして F 型変数、D 型変数を使用する場合、小数点以下 4 桁までしか表示しません。

**用　例**

```
DEFINT li1, li2
HOLD li1 + li2              ' プログラムを一時停止し、ステップ停止状態にして
                            '(li1 + li2) の値をティーチングペンダントへ出力し
                            ' ます。
HOLD "stop"                 ' プログラムを一時停止し、ステップ停止状態にして文
                            ' 字列 "stop" をティーチングペンダントへ出力します。
```

# HALT（ステートメント）【SLIM 準拠】

**機　能**

プログラムの実行を停止します。

**書　式**

HALT < メッセージ >

**説　明**

プログラムの実行を停止し、< メッセージ > に指定した内容をティーチングペンダントへ出力します。< メッセージ > の内容に関しては、PRINTMSG 文を参照してください。

再起動すると、プログラムはその先頭から実行します。

メッセージの文字数は半角で 60 文字です。

**関連項目**

DELAY、HOLD、STOP

**注意事項**

メッセージとして F 型変数、D 型変数を使用する場合、小数点以下 4 桁までしか表示しません。

**用　例**

```
DEFINT li1, li2
HALT li1 + li2                ' プログラムを停止し、(li1 + li2) の値をティーチング
                              ' ペンダントへ出力します。
HALT "end"                    ' プログラムを停止し、文字列 "end" をティーチングペン
                              ' ダントへ出 ' 力します。
```

# INTERRUPT ON/OFF（ステートメント）

**機　能**

ロボットの動作を中断します。

**書　式**

INTERRUPT {ON|OFF}

**説　明**

INTERRUPT ON と INTERRUPT OFF は対に用いられ、それに囲まれた範囲内では、動作命令実行中に割り込みスキップ信号が ON した場合、動作命令を中断後、次のステップに進みます。

INTERRUPT コマンドを実行するためには、タスクがアームグループを取得している必要があります。

INTERRUPT ON 文を実行して割り込み可能な状態にしておかないと、割り込みスキップ信号を ON しても、実行中の動作命令は終了しません。

プログラムが停止状態になる、またはGIVEARM命令を実行した場合、自動的にINTERRUPTはOFFされます。

**注意事項**

- 割り込みスキップ直後に相対動作命令を実行すると、停止した位置からの相対動作となります。INTERRUPT で囲んだ範囲の動作命令は、絶対動作命令を使用してください。
- INTERRUPT ON 中は、パス動作を行ないません。
  パス動作を指定した場合は、すべてエンド動作で実行されます。
- INTERRUPT で囲まれた範囲の動作命令は、割り込みスキップ信号入力で、すべて動作を停止します。マルチタスクで動作中は注意が必要です。
  例

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 0 | ○ | ○ | ○ | ○ | × | × | × | × |
| Group 1 | × | × | × | × | × | × | ○ | ○ |
| Group 2 | ○ | ○ | ○ | ○ | × | × | ○ | ○ |
| Group 3 | × | × | × | × | × | × | × | × |
| Group 4 | × | × | × | × | × | × | × | × |

```
PRO1
TAKEARM  0
INTERRUPT ON
MOVE P,P0          ’動作命令
INTERRUPT OFF
END
```

```
PRO2
TAKEARM  1
INTERRUPT ON
DRIVE(7,30)        ’動作命令
INTERRUPT OFF
END
```

PRO1、PRO2 が同時に動作命令実行中に、割り込みスキップ信号が ON した場合、PRO1 の MOVE 命令、PRO2 の DRIVE 命令のどちらも動作を中断し、次のステップに進みます。

用　例

例 1：

```
DIM lp1 As Position
INTERRUPT ON                    ' 専用 I/O ポートの割り込み信号が ON されたときに、実
                                ' 行中の動作命令を中断後、次のステップに進みます。
MOVE P, lp1
INTERRUPT OFF
```

例 2（付加軸の例）：

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | × | × | × | × | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1                       ' アームグループ 1 を取得 (7 軸、8 軸を含むアームグループ)
INTERRUPT ON
DRIVE (7,100),(8,30)            'INTERRUPT 命令で囲まれた範囲の動作命令実行中に、
                                ' 割り込みスキップ信号が ON した場合、動作命令を中断後、
                                ' 次のステップに進みます。
INTERRUPT OFF
END
```

## 12.4　速度制御

# SPEED（ステートメント）【SLIM 準拠】

**機　能**

現在取得しているアームグループ軸の内部移動速度を指定します。

**書　式**

SPEED ＜移動速度＞

**説　明**

＜移動速度＞には、現在取得しているアームグループ軸の、最大内部移動速度の比率 (%) を 0.1 ～ 100 の実数で指定しします。

**注意**

**0.1 以下の値を指定した場合、0 より大きいときはエラーになりませんが、実際の速度と異なることがあります。**

実際の速度値は ( 外部 × 内部 ÷100) です。

SPEED を設定すると JSPEED 、ACCEL、DECEL、JACCEL、JDECEL の値も自動的に設定されます。

| | CP 制御 | PTP 制御 |
|---|---|---|
| 速度設定 | SPEED | JSPEED |
| 加速度設定 | ACCEL | JACCEL |
| 減速度設定 | DECEL | JDECEL |
| 現在速度取得 | CURSPD | CURJSPD |
| 現在加速度取得 | CURACC | CURJACC |
| 現在減速度取得 | CURDEC | CURJDEC |

例：SPEED 50 の時次の値に設定されます。

| | | |
|---|---|---|
| JSPEED | 50 | (SPEED と同じ値 ) |
| ACCEL | 25 | (SPEED ＊ SPEED÷100) |
| JACCEL | 25 | (SPEED ＊ SPEED÷100) |
| DECEL | 25 | (SPEED ＊ SPEED÷100) |
| JDECEL | 25 | (SPEED ＊ SPEED÷100) |

アームグループを設定せずに速度を設定しようとした場合は、エラーとなります。

**関連項目**

ACCEL、DECEL、JSPEED、MPS

用　例

例 1：

```
DIM li1 As Integer
SPEED 100                    ’手先の移動速度を 100 に設定します。
SPEED li1/100                ’手先の移動速度を (li1/100) の値に設定します。
```

例 2（付加軸の例）：

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | × | × | × | × | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1                    ’アームグループ 1 を取得（7 軸、8 軸を含むアームグループ）
SPEED 100                    ’アームグループの軸 (7 軸、8 軸 ) の速度が設定されます。
END
```

# JSPEED（ステートメント）

**機　能**

現在取得しているアームグループ軸の内部軸速度を指定します。

**書　式**

JSPEED ＜軸速度＞

**説　明**

＜軸速度＞には、現在取得しているアームグループ軸の、最大内部軸速度の比率 (%) を 0.1 ～ 100 の実数で指定します。

**注意**

**0.1 以下の値を指定した場合、0 より大きいときはエラーになりませんが、実際の速度と異なることがあります。**

実際の速度値は ( 外部 × 内部 ÷100) です。

JSPEED を設定すると JACCEL、JDECEL の値も自動的に設定されます。

例：JSPEED50 の時次の値に設定されます。

JACCEL 25 (JSPEED ＊ JSPEED÷100)
JDECEL 25 (JSPEED ＊ JSPEED÷100)

アームグループを設定せずに軸速度を設定しようとした場合は、エラーとなります。

**関連項目**

CURJSPD、JACCEL、JDECEL、SPEED

**用　例**

**例 1：**

```
DIM li1 As Integer
JSPEED 100                    ' 軸の移動速度を 100 に設定します。
JSPEED li1/100                ' 軸の移動速度を (li1/100) の値に設定します。
```

**例 2（付加軸の例）：**

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | × | × | × | × | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1                     ' アームグループ 1 を取得（7 軸、8 軸を含むアームグループ）
JSPEED 100                    ' アームグループの軸 (7 軸、8 軸 ) の軸速度のみが設定されます。
END
```

# ACCEL（ステートメント）【SLIM 準拠】

**機　能**

現在取得しているアームグループ軸の内部加速度、内部減速度を指定します。

**書　式**

ACCEL ＜加速度＞ ［, ＜減速度＞］

**説　明**

＜加速度＞、＜減速度＞には、現在取得しているアームグループ軸の、最大内部加速度、減速度の比率 (%) を 0.0001 ～ 100 の実数で指定します。

**注意**

**0.0001 以下の値を指定した場合、0 より大きいときはエラーになりませんが、実際の加速度と異なることがあります。**

実際の加減速度値は ( 外部 × 内部 ÷100) です。

加減速値は速度を変更すると、(SPEED$^2$÷100) の式の値に自動的に変更されます。

付加軸の場合、ACCEL を設定すると JACCEL( 軸加速度 ) も同じ値に設定されます。

例：ACCEL50 を設定すると、JACCEL50( 同じ値 ) も設定されます。

アームグループを設定せずに加速度を設定しようとした場合は、エラーとなります。

**関連項目**

DECEL、SPEED

**用　例**

**例 1：**

```
DEFINT li1, li2
ACCEL 100                    ' 加速度を 100 に設定します。
ACCEL 50, 25                 ' 加速度を 50, 減速度を 25 に設定します。
ACCEL li1/100, li2/100       ' 加速度を (li1/100), 減速度を (li2/100) の値に設定し
                             ' ます。
```

**例 2（付加軸の例）：**

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | × | × | × | × | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1                    ' アームグループ 1 を取得（7 軸、8 軸を含むアームグループ）
ACCEL 100                    ' アームグループの軸 (7 軸、8 軸 ) の加速度が設定されます。
END
```

# JACCEL（ステートメント）【SLIM 準拠】

**機　能**

現在取得しているアームグループ軸の、内部軸加速度、内部軸減速度を指定します。

**書　式**

JACCEL ＜軸加速度＞［, ＜軸減速度＞］

**説　明**

＜軸加速度＞、＜軸減速度＞には、現在取得しているアームグループ軸の、最大内部軸加速度、軸減速度の比率 (%) を 0.0001 ～ 100 の実数で指定します。

**注意**

**0.0001 以下の値を指定した場合、0 より大きいときはエラーになりませんが、実際の速度と異なることがあります。**

実際の加減速度値は ( 外部 × 内部 ÷100) です。

軸加減速値は軸速度を変更すると、(JSPEED$^2$÷100) の式の値に自動的に変更されます。

アームグループを設定せずに内部軸加速度を設定しようとした場合は、エラーとなります。

**関連項目**

JDECEL、JSPEED、SPEED

**用　例**

**例 1：**

```
DEFINT li1, li2
JACCEL 100                  ' 軸加速度を 100 に設定します。
JACCEL 50, 25               ' 軸加速度を 50, 軸減速度を 25 に設定します。
JACCEL li1/100, li2/100     ' 軸加速度を (li1/100), 軸減速度を (li2/100) の値に設
                            ' 定します。
```

**例 2（付加軸の例）：**

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | × | × | × | × | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1                   ' アームグループ 1 を取得（7 軸、8 軸を含むアームグループ）
JACCEL 100                  ' アームグループの軸 (7 軸、8 軸 ) の軸加速度が設定されます。
END
```

# DECEL（ステートメント）【SLIM 準拠】

**機　能**

現在取得しているアームグループ軸の内部減速度を指定します。

**書　式**

DECEL ＜減速度＞

**説　明**

＜減速度＞には、現在取得しているアームグループ軸の、最大内部減速度の比率 (%) を 0.0001 ～ 100 の実数で指定します。

**注意**

**0.0001 以下の値を指定した場合、0 より大きいときはエラーになりませんが、実際の減速度と異なることがあります。**

実際の減速度値は ( 外部 × 内部 ÷100) です。

減速値は速度を変更すると、(SPEED$^2$÷100) の式の値に自動的に変更されます。

付加軸の場合、DECEL を設定すると JDECEL( 軸加速度 ) も同じ値に設定されます。

例：DECEL 50 を設定すると、JDECEL50( 同じ値 ) も設定されます。

アームグループを設定せずに減速度を設定しようとした場合は、エラーとなります。

**関連項目**

ACCEL、SPEED

**用　例**

**例 1：**

```
DIM li1 As Integer
DECEL 100                    ' 減速度を 100 に設定します。
DECEL li1/100                ' 減速度を (li1/100) の値に設定します。
```

**例 2（付加軸の例）：**

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | × | × | × | × | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1                    ' アームグループ 1 を取得（7 軸、8 軸を含むアームグループ）
DECEL 100                    ' アームグループの軸 (7 軸、8 軸 ) の減速度が設定されます。
END
```

# JDECEL（ステートメント）【SLIM 準拠】

**機　能**

現在取得しているアームグループ軸の内部軸減速度を指定します。

**書　式**

JDECEL ＜軸減速度＞

**説　明**

＜軸減速度＞には、現在取得しているアームグループ軸の、最大内部軸減速度の比率 (%) を 0.0001 ～ 100 の実数で指定します。

**注意**

**0.0001 以下の値を指定した場合、0 より大きいときはエラーになりませんが、実際の速度と異なることがあります。**

実際の減速度値は ( 外部 × 内部 ÷100) です。

軸減速値は軸速度を変更すると、(JSPEED$^2$÷100) の式の値に自動的に変更されます。

アームグループを設定せずに内部軸減速度を設定しようとした場合は、エラーとなります。

**関連項目**

JACCEL、JSPEED、SPEED

**用　例**

**例 1：**

```
DIM li1 As Integer
JDECEL 100                    ' 軸減速度を 100 に設定します。
JDECEL li1/100                ' 軸減速度を (li1/100) の値に設定します。
```

**例 2（付加軸の例）：**

| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 1 | × | × | × | × | × | × | ○ | ○ |

```
PROGRAM PRO1
TAKEARM 1                     ' アームグループ 1 を取得（7 軸、8 軸を含むアームグループ）
JDECEL 100                    ' アームグループの軸 (7 軸、8 軸 ) の軸減速度が設定されます。
END
```

# CURACC（システム変数）

**機　能**

現在取得しているアームグループ軸の、内部加速度を取得します。

**書　式**

CURACC

**説　明**

アームグループを取得していないタスクで CURACC コマンドを実行した場合、100 を返します。

**関連項目**

ACCEL、DECEL、CURDEC、CURJACC、CURJDEC、CURJSPD、CURSPD

**用　例**

例 1：

```
DIM lf1 As Single
lf1 = CURACC                    ' 加速度の現在値を lf1 に代入します。
ACCEL CURACC * 0.5              ' 加速度を現在の 50% にします。
```

例 2（付加軸の例）：

```
PROGRAM PRO1
TAKEARM 1                       ' アームグループ 1 を取得
ACCEL 70                        ' アームグループ 1 の加速度を 70 と設定
I0 = CURACC                     ' アームグループ 1 の加速度 70 を I0 に代入します。
END
```

# CURJACC（システム変数）

**機　能**

現在取得しているアームグループ軸の、内部軸加速度を取得します。

**書　式**

CURJACC

**説　明**

アームグループを取得していないタスクで CURJACC コマンドを実行した場合、100 を返します。

**関連項目**

JACCEL、JDECEL、JSPEED、SPEED、CURDEC、CURJDEC、CURJSPD、CURSPD

**用　例**

例 1：

```
DIM lf1 As Single
lf1 = CURJACC                  ' 軸加速度の現在値を lf1 に代入します。
JACCEL CURJACC * 0.5           ' 軸加速度を現在の 50% にします。
```

例 2（付加軸の例）：

```
PROGRAM PRO1
TAKEARM 1                      ' アームグループ 1 を取得
JACCEL 70                      ' アームグループ 1 の軸加速度を 70 と設定
I0 = CURJACC                   ' アームグループ 1 の軸加速度 70 を I0 に代入します。
END
```

# CURDEC（システム変数）

**機　能**

現在取得しているアームグループ軸の、内部減速度を取得します。

**書　式**

CURDEC

**説　明**

アームグループを取得していないタスクで CURDEC コマンドを実行した場合、100 を返します。

**関連項目**

ACCEL、DECEL、CURACC、CURJACC、CURJDEC、CURJSPD、CURSPD

**用　例**

例 1：

```
DIM lf1 As Single
lf1 = CURDEC                    ' 減速度の現在値を lf1 に代入します。
DECEL CURDEC * 0.5              ' 減速度を現在の 50% にします。
```

例 2（付加軸の例）：

```
PROGRAM PRO1
TAKEARM 1                       ' アームグループ 1 を取得
DECEL 70                        ' アームグループ 1 の減速度を 70 と設定
I0 = CURDEC                     ' アームグループ 1 の減速度 70 を I0 に代入します。
END
```

# CURJDEC（システム変数）

**機　能**

現在取得しているアームグループ軸の、内部軸減速度を取得します。

**書　式**

CURJDEC

**説　明**

アームグループを取得していないタスクで CURJDEC コマンドを実行した場合、100 を返します。

**関連項目**

JACCEL、JDECEL、JSPEED、SPEED、CURACC、CURDEC、CURJACC、CURJSPD、CURSPD

**用　例**

**例 1：**

```
DIM lf1 As Single
lf1 = CURDEC                    ' 減速度の現在値を lf1 に代入します。
DECEL CURDEC * 0.5              ' 減速度を現在の 50% にします。
```

**例 2（付加軸の例）：**

```
PROGRAM PRO1
TAKEARM 1                       ' アームグループ 1 を取得
JDECEL 70                       ' アームグループ 1 の軸減速度を 70 と設定
I0 = CURJDEC                    ' アームグループ 1 の軸減速度 70 を I0 に代入します。
END
```

# CURJSPD（システム変数）

**機　能**

現在取得しているアームグループ軸の、内部移動軸速度を取得します。

**書　式**

CURJSPD

**説　明**

アームグループを取得していないタスクで CURJSPD コマンドを実行した場合、100 を返します。

**関連項目**

JSPEED、CURACC、CURDEC、CURJACC、CURJDEC、CURSPD

**用　例**

例 1：

```
DIM lf1 As Single
lf1 = CURJSPD                  ' 軸の移動速度の現在値を lf1 に代入します。
JSPEED CURJSPD * 0.5           ' 軸の移動速度を現在の 50% にします。
```

例 2（付加軸の例）：

```
PROGRAM PRO1
TAKEARM 1                      ' アームグループ 1 を取得
JSPEED 70                      ' アームグループ 1 の軸速度を 70 と設定
I0 = CURJSPD                   ' アームグループ 1 の軸速度 70 を I0 に代入します。
END
```

# CURSPD（システム変数）

**機　能**

現在取得しているアームグループ軸の、内部移動速度を取得します。

**書　式**

CURSPD

**説　明**

アームグループを取得していないタスクで CURSPD コマンドを実行した場合、100 を返します。

**関連項目**

CURACC、CURDEC、CURJSPD、CURJACC、CURJDEC

**用　例**

**例 1：**

```
DIM lf1 As Single
lf1 = CURJSPD                ’軸の移動速度の現在値を lf1 に代入します。
JSPEED CURJSPD * 0.5         ’軸の移動速度を現在の 50% にします。
```

**例 2（付加軸の例）：**

```
PROGRAM PRO1
TAKEARM 1                    ’アームグループ 1 を取得
SPEED 70                     ’アームグループ 1 の速度を 70 と設定
I0 = CURSPD                  ’アームグループ 1 の速度 70 を I0 に代入します。
END
```

# CUREXTACC（システム変数）【Ver.1.4 以降】

**機　能**

外部加速度の現在値を得ます。

**書　式**

CUREXTACC

**説　明**

現在設定されている外部加速度が格納されます。

**関連項目**

ACCEL、DECEL、CUREXTDEC

**用　例**

```
DIM 1f1 As Single
1f1 = CUREXTACC
ACCEL CUREXTACC * 0.5
```

# CUREXTDEC（システム変数）【Ver.1.4 以降】

**機　能**

外部減速度の現在値を得ます。

**書　式**

CUREXTDEC

**説　明**

現在設定されている外部減速度が格納されます。

**関連項目**

ACCEL、DECEL、CUREXTACC

**用　例**

```
DIM 1f1 As Single
1f1 = CUREXTDEC
DECEL CUREXTDEC * 0.5
```

# CUREXTSPD（システム変数）【Ver.1.4 以降】

**機　能**

外部速度の現在値を得ます。

**書　式**

CUREXTSPD

**説　明**

現在設定されている外部速度が格納されます。

**関連項目**

SPEED

**用　例**

```
DIM 1f1 As Single
1f1 = CUREXTSPD
SPEED CUREXTSPD * 0.5
```

# EXTSPEED（ステートメント）【Ver.1.98 以降】

**機　能**

外部速度設定

**書　式**

EXTSPEED　＜外部速度設定値＞

**説　明**

外部速度を設定します。

**補足説明**

**このコマンドにより外部速度設定がプログラムからも可能になりました。**

**関連項目**

SPEED

**用　例**

```
PROGRAM  PRO1
-------------------------
EXTSPEED 100                  ' 外部速度設定値を 100 にする
-------------------------
END
```

## 12.5　時間制御

# DELAY（ステートメント）【SLIM 準拠】

**機　能**

指定した時間の間、プログラムの進行を停止します。

**書　式**

DELAY < 遅延時間 >

**説　明**

< 遅延時間 > で指定された時間が経過するまで待ちます。

< 遅延時間 > の単位は ms ですが、実際の遅延時間は 1/60s 単位で増減します。また、多数のタスクを同時に走らせた場合には、遅延時間が指定値より延びることがあります。

**関連項目**

HOLD、WAIT

**注意事項**

- 遅延時間を使用する際には、命令実行中に瞬時停止を行ない再起動すると、一時停止中も遅延時間が経過することに注意してください。
- 実際の遅延時間には 16ms 程度の誤差があります。

**実際の遅延時間の計算方法**

x：遅延時間設定値、y：実際の遅延時間

$y_1 = \dfrac{x \times 60 + 500}{1000}$　’小数点以下は切り捨てます。また、0 の場合は 1 にする

$y = \dfrac{y_1 \times 1000}{60}$

実際の遅延時間は、最小 $y - \dfrac{1000}{60}$ ms から最大 y ms になります。

例）遅延時間設定値に 30ms を設定したとき

$y_1 = \dfrac{30 \times 60 + 500}{1000} = 2.3$　’小数点以下を切り捨てるので 2 になる

$y = \dfrac{2 \times 1000}{60} = 33.33$

実際の遅延時間は、$\dfrac{1000}{60}$ ms から $\dfrac{2000}{60}$ ms になります。

- 指定の時間以上必ず遅延させたい場合の設定値の算出方法は以下の通りです。

  x：必ず遅延させたい時間、y：設定値

  $x_1 = \dfrac{x}{\left(\dfrac{1000}{60}\right)}$　　‘小数点以下は切り捨ててください。

  $y = x_1 \times \dfrac{1000}{60} + 25$　‘小数点以下は切り上げてください。

  例：30ms 以上遅延が必要な場合

  $x_1 = \dfrac{30}{\left(\dfrac{1000}{60}\right)} = 1.8$　　‘小数点以下を切り捨てて 1 にする。

  $y = 1 \times \dfrac{1000}{60} + 25 = 41.666$　‘小数点以下を切り上げて 42 にする。

  設定値を 42 (DELAY 42) にすると実際の遅延時間は 30ms 以上必ず確保できます。

**用　例**

```
DIM li1 As Integer
DELAY 100                    '100ms(0.1s) の時間が経過するまで待ちます。
DELAY li1 + 10               '(li1 + 10) の値の時間が経過するまで待ちます。
```

# WAIT（ステートメント）【SLIM 準拠】

**機　能**

条件プログラム停止を行ないます。

**書　式**

WAIT < 条件式 > [,< タイムアウト時間 >][,< 格納変数 >]

**説　明**

< 条件式 > が成立するまでプログラムの実行を停止します。

< タイムアウト時間 > を設定すると、指定時間を経過した後に WAIT 文の実行を終了し、次のコマンドに制御が移ります。これによって、無限停止を防止できます。

< タイムアウト時間 > の単位は ms です。

< 条件式 > または < タイムアウト時間 > の条件成立を監視するために再評価する間隔は、タスクの優先度に依存します。

[Ver.1.8 以降 ] では、< 格納変数 > を設定すると条件式が成立して、WAIT ステートメントを抜けたとき指定した変数に TRUE(1) が入り、タイムアウトで抜けたときは指定した変数に FALSE(0) が入ります。

**関連項目**

DELAY

**注意事項**

タイムアウト時間を使用する際には、命令実行中に瞬時停止を行ない再起動すると、一時停止中も指定時間が経過することに注意してください。

**用　例**

```
DEFINT li1, li2, li3, li4, li5
WAIT li1 = 1                    'li1 = 1 が成立するまで待ちます。
WAIT li2 = 0, 2000              'li2 = 0 が成立するまで待ち、2 秒経
                                ' 過しても成立しなければ次の文へ移
                                ' ります。
WAIT li3 = li4, li5             'li3 = li4 が成立するまで待ち、li5
                                ' の値の時間が経過しても成立しなけ
                                ' れば次の文へ移ります。
WAIT IO[10] = ON                'IO の 10 番が ON になるまで待ちます。
```

[Ver.1.8 以降 ]

```
WAIT li3 = li4, li5, li6        'li3 = li4 が成立するまで待ち、li5
                                ' の値の時間が経過しても成立しなけ
                                ' れば次の文へ移ります。このとき、
                                'li5 の値の時間までに li3=li4 が成立
                                ' したら li6=TRUE が入り、条件式が成
                                ' 立せずにタイムアウトしたら
                                'li6=FALSE が入ります。
```

## 12.6　座標変換

# CHANGETOOL（ステートメント）

**機　能**

ツール座標系を変更します。

**書　式**

CHANGETOOL < ツール座標系番号 >

**説　明**

ツール座標系を < ツール座標系番号 > に指定した座標系に変更します。
この宣言は、次に CHANGETOOL 文が実行されるまで有効です。
< ツール座標系番号 > は、0 ～ 63 まで有効です。
ツール 0 はメカニカルインターフェイス座標系です。

**関連項目**

TOOL、TOOLPOS

**注意事項**

TAKEARM を実行すると TOOL0 が設定されます。

**用　例**

```
DIM li1 As Integer
DIM lp1 As Position
Li1 = 1
Lp1 = (10, 10, 5, 0, 9, 0, 1)
TOOL 1, lp1                    ' ツール座標系番号 1 に、ポジション型変数 lp1 で示さ
                               ' れるツール座標系を宣言します。
CHANGETOOL li1                 'li1 で示されるツール座標系番号のツール座標系に変
                               ' 更します。
```

# CHANGEWORK（ステートメント）

**機　能**

ユーザ座標系を変更します。

**書　式**

CHANGEWORK < ユーザ座標系番号 >

**説　明**

ユーザ座標系を < ユーザ座標系番号 > に指定した座標系に変更します。

この宣言は、次に CHANGEWORK 文が実行されるまで有効です。

< ユーザ座標系番号 > は、0 ～ 7 まで有効です。

ユーザ 0 はベース座標系です。

**関連項目**

WORK、WORKPOS

**注意事項**

- TAKEARM を実行すると WORK0 が設定されます。(ただし、サブルーチンはその限りではありません。)
- 現在選択されている番号のワーク座標系の値を「WORK」コマンドで変更しても、ロボットのワーク座標系は変更されません。「CHANGEWORK」で再度選択してください。

**用　例**

```
DIM li1 As Integer
DIM lp1 As Position
Li1 = 1
Lp1 = (10, 10, 5, 0, 9, 0, 1)
WORK 1, lp1                  ' ユーザ座標系番号 1 に、ポジション型変数 lp1 で示さ
                             ' れるユーザ座標系を宣言します。
CHANGEWORK li1               'li1 で示されるユーザ座標系番号のユーザ座標系に変
                             ' 更します。
```

# CURTOOL（システム変数）【Ver.1.4 以降】

**機　能**

現在設定されている TOOL 番号を得ます。

**書　式**

CURTOOL

**説　明**

現在設定されている TOOL 番号が格納されます。

**関連項目**

TOOL、CHANGETOOL、TOOLPOS

**用　例**

```
DIM li1 As Integer
li1 = CURTOOL
TOOL CURTOOL+I1, P1
CHANGETOOL CURTOOL+I1
P1 = TOOLPOS (CURTOOL)
```

# CURWORK（システム変数）【Ver.1.4 以降】

**機　能**

現在設定されている WORK 番号を得ます。

**書　式**

CURWORK

**説　明**

現在設定されている WORK 番号が格納されます。

**関連項目**

WORK、CHANGEWORK、WORKPOS

**用　例**

```
DIM li1 As Integer
li1 = CURWORK
WORK CURWORK+I1, P1
CHANGEWORK CURWORK+I1
P1 = WORKPOS (CURWORK)
```

## 12.7　干渉チェック

# SETAREA（ステートメント）

**機　能**

干渉チェックを行なうエリアを選択します。

**書　式**

SETAREA < 干渉チェックエリア番号 >

**説　明**

< 干渉チェックエリア番号 > は、あらかじめ定義しておいた干渉チェックエリアを選択します。有効なエリア番号は、0 ～ 31 です。AREA コマンドによって事前に宣言されていなければなりません。

SETAREA コマンドを実行後、RESETAREA コマンドが実行されるまでの間、干渉チェックが行なわれます。いったん干渉が検出されると、RESETAREA コマンドを実行し、再度 SETAREA するまでは、次の干渉を検出しません。

干渉チェックは、SETAREA コマンドが実行されていれば、0 ～ 31 までのエリア番号で表される、最大 32 個のエリアについて同時に行なえます。

なお、SETAREA コマンドは、AREA コマンドで宣言した指定 I/O を初期化しません。入出力制御文によって I/O を直接 SET している場合などは、RESETAREA コマンドで初期化する必要があります。

**関連項目**

AREA、RESETAREA、AREAPOS、AREASIZE

**用　例**

```
DIM lp1 As Position
DIM lv1 As Vector
Lp1 = (10, 10, 5, 0, 9, 0, 1)
Lv1 = (50, 10, 50)
AREA 2, lp1, lv1, 104, 1     '2 番のエリアに、lp1 で示されるポジションで lv1 で示
                             ' されるエリアに I/O 番号 104 を宣言します。
SETAREA 2                    '2 番のエリアチェックを有効にします。
RESETAREA 2                  '2 番のエリアチェックを初期化します。
```

# RESETAREA（ステートメント）

**機　能**

干渉チェックを初期化します。

**書　式**

RESETAREA < 初期化エリア番号 >

**説　明**

干渉を検出したときに SET した I/O を RESET し、干渉チェックを無効化します。

入出力制御文によって I/O を直接 RESET した場合、再度干渉が起きても I/O は SET されません。再度干渉チェックを行なうには、RESETAREA コマンドを実行して初期化し、再び SETAREA コマンドを実行します。

有効なエリア番号は、0 ～ 31 です。

**関連項目**

AREA、SETAREA、AREAPOS、AREASIZE

**注意事項**

マルチタスクで動作しているため、SETAREA を実行してすぐに RESET 命令があると、エリアをチェックするタイミングによっては RESET の方が先に実行されてしまうことがあります。

**用　例**

```
DIM lp1 As Position
DIM lv1 As Vector
Lp1 = (10, 10, 5, 0, 9, 0, 1)
Lv1 = (50, 10, 50)
AREA 2, lp1, lv1, 104, 1     '2 番のエリアに、lp1 で示されるポジションで lv1 で示
                             ' されるエリアに I/O 番号 104 を宣言します。
SETAREA 2                    '2 番のエリアチェックを有効にします。
RESETAREA 2                  '2 番のエリアチェックを初期化します。
```

# SETPOSTUREAREA（ステートメント）【Ver.3.0 以降】

**機　能**

高感度位置姿勢検知機能を有効にします。

**書　式**

SETPOSTUREAREA

**説　明**

POSTUREAREA で設定した高感度位置姿勢検知機能を有効にします。

SETPOSTUREAREA コマンド実行後、RESETPOSTUREAREA コマンドが実行されるまでの間、高感度位置姿勢検知が行われます。

SETPOSTUREAREA コマンドでは POSTUREAREA で宣言した I/O を初期化しません。

**関連項目**

POSTUREAREA、RESETPOSTUREAREA

**注意事項**

高感度位置姿勢検知機能はオプション機能です。機能ライセンスを購入してください。

**用　例**

```
POSTUREAREA, P1,V0,24,-1,0, TAR_POSTURE ( V2 , 1 ), TAR_EXJOINT ( ( 7 , 30 , 4 ),
( 8 , 90 , 1) )
SETPOSTUREAREA
```

# RESETPOSTUREAREA（ステートメント）【Ver.3.0 以降】

**機　能**

高感度位置姿勢検知機能を無効にします。

**書　式**

RESETPOSTUREAREA

**説　明**

SETPOSTUREAREA で有効にした高感度位置姿勢検知機能を無効にします。

SETPOSTUREAREA コマンド実行後、RESETPOSTUREAREA コマンドが実行されるまでの間、高感度位置姿勢検知が行われます。

RESETPOSTUREAREA コマンドでは POSTUREAREA で宣言した I/O を OFF にします。

**関連項目**

POSTUREAREA、SETPOSTUREAREA

**注意事項**

高感度位置姿勢検知機能はオプション機能です。機能ライセンスを購入してください。

**用　例**

```
POSTUREAREA, P1,V0,24,-1,0, TAR_POSTURE ( V2 , 1 ), TAR_EXJOINT ( ( 7 , 30 , 4 ),
( 8 , 90 , 1) )
SETPOSTUREAREA
MOVE P,P1
RESETPOSTUREAREA
```

# 12.8　サーボ内部データ

## GetSrvData（システム変数）【Ver.1.5 以降】

**機　能**

ロボット軸のサーボ内部データを取得します。(Ver. 1.5 以降対応 )

**書　式**

< サーボ内部データ > = GetSrvData(< データ番号 >)

**説　明**

< データ番号 > で指定したサーボ内部データを取得します。

< サーボ内部データ > は、ジョイント型のデータで、ロボット軸のデータがセットされます。データ番号により以下のデータが取得されます。

| < データ番号 > | < サーボ内部データ > |
|---|---|
| 1 | モータ速度現在値 [rpm] |
| 2 | モータ角度偏差 [Pulse] |
| 4 | モータ電流絶対値 [ 定格比 %] |
| 5 | モータトルク指令値 ( 重力補償分は除く ) [ 定格比 %] |
| 8 | 各軸位置、角度指令値 [mm or deg] |
| 17 | ツール端速度 ( ワーク座標系 , 位置 3 成分のみ )[mm/s] |
| 18 | ツール端偏差 ( ワーク座標系 , 位置 3 成分のみ )[mm] |
| 19 | ツール端速度 ( ツール座標系 , 位置 3 成分のみ )[mm/s] |
| 20 | ツール端偏差 ( ツール座標系 , 位置 3 成分のみ )[mm] |

**関連項目**

GetJntData

**注意事項**

- 上記以外のデータ番号は、予約番号です。30 まで指定してもエラーになりませんが、指定しないでください。
- サーボ単軸モニタ実行中にサーボデータを取得すると、取得時間が大幅に伸びる場合があります。サーボ単軸モニタ機能と併用する場合はご注意ください。
- < データ番号 > を変更すると、変更時間がかかる場合があります。< データ番号 > の切替えは最小限にしてください。
- ロボット制御権を取得 (TAKEARM) したタスクにて実行ください。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
defjnt vel
defsng absv,xvel,yvel,zvel
vel=GetSrvData(17)                          ' ツール端速度を取得する。
xvel=JOINT(1,vel)                           ' ワーク座標 X 成分抽出
yvel=JOINT(2,vel)                           ' ワーク座標 Y 成分抽出
zvel=JOINT(3,vel)                           ' ワーク座標 Z 成分抽出
absv = SQR(xvel*xvel+yvel*yvel+zvel*zvel)   ' ツール端合成速度計算
```

# GetJntData（システム変数）

**機　能**

指定軸のサーボ内部データを取得します。(Ver. 1.5 以降対応 )

**書　式**

＜指定軸サーボ内部データ＞= GetJntData(＜データ番号＞、＜軸番号＞)

**説　明**

＜データ番号＞と＜軸番号＞で指定した指定軸のサーボ内部データを取得します。

＜指定軸サーボ内部データ＞は、F 型 ( 単精度実数型 ) のデータで、指定軸のデータがセットされます。データ番号により以下のデータが取得されます。

| ＜データ番号＞ | ＜サーボ内部データ＞ |
|---|---|
| 1 | モータ速度現在値 [rpm] |
| 2 | モータ角度偏差 [Pulse] |
| 4 | モータ電流絶対値 [ 定格比 %] |
| 5 | モータトルク指令値 ( 重力補償分は除く ) [ 定格比 %] |
| 8 | 各軸位置、角度指令値 [mm or deg] |

**関連項目**

GetSrvData

**注意事項**

- 上記以外のデータ番号は、予約番号です。30 まで指定してもエラーになりませんが、指定しないでください。
- サーボ単軸モニタ実行中にサーボデータを取得すると、取得時間が大幅に伸びる場合があります。サーボ単軸モニタ機能と併用する場合はご注意ください。
- ＜データ番号＞を変更すると、変更時間がかかる場合があります。＜データ番号＞の切替えは最小限にしてください。
- 制御権を取得 (TAKEARM) したタスクにて実行ください。制御権未取得の場合は、エラー「セマフォを取得できません」が発生します。

**用　例**

```
defsng vel
vel=GetJntData(1,7)          ’７軸のモータ速度を取得します。
```

## 12.9　モータ電源

# MOTOR ｛ON | OFF｝（ステートメント）【Ver.1.98 以降】

**機　能**

モータ電源 ON/OFF を行ないます。

**書　式**

MOTOR ｛ON | OFF｝[, <MOTOR オプション >]

**説　明**

モータ電源の ON、OFF を行ないます。

<MOTOR オプション > で「RESULT」を指定すると、MOTER コマンド実行時に内部でエラーが発生してもプログラムを停止させません（ただし MOTOR OFF コマンドは安全上、エラーメッセージを出し、プログラムが停止します。）。そのときの内部エラーのエラーコードを RESULT=( 変数名 ) で指定した変数に格納します。(v2.6 以降のコントローラに限る。)

**関連項目**

INIT

**注意事項**

MOTOR OFF 命令は、ペンダントからのモータ OFF キーと全く同じ動作をします。ロボット動作、およびプログラム実行を停止し、コンティ停の状態となります。

使用条件の 296 : モータコマンド設定を 0 → 1 に変更した場合、MOTOR OFF 命令でプログラム実行中にモータ OFF することができますが、プログラムは停止しません。ただし、MOTOR OFF 命令をロボット動作中に実行することはできません。誤ってロボット動作中に実行するとエラーを発生してプログラムを停止します。

MOTOR ON 命令は、モータコマンド設定を変更しても動作は変わりません。

**用　例**

```
PROGRAM  PRO1
------------
MOTOR  ON                    ' モータ電源 ON
------------
MOTOR  OFF                   ' モータ電源 OFF
------------
MOTOR ON, RESULT = I1        ' モータ電源を ON します。
                             'MOTOR ON 失敗時、変数 I1 にエラーコードを格納し、
                             ' 次の行に移ります。
------------
MOTOR OFF ,RESULT = I2       ' モータ電源を OFF します。
                             'MOTOR OFF 失敗時、変数 I2 にエラーコードを
                             ' 格納します。
                             ' エラーメッセージを表示し、プログラムは停止します。
------------
END
```

# 12.10 CAL

## EXECAL（ステートメント）【Ver.1.98 以降】

**機　能**

CAL 実行

**書　式**

EXECAL

**説　明**

CAL を実行します。

**関連項目**

INIT

**注意事項**

"E シリーズ "　ロボットのように、CAL の必要ないロボットでは何も行ないません。

**用　例**

```
PROGRAM  PRO1
------------
EXECAL                          'CAL を実行する
------------
END
```

# 12.11 特殊制御

従来、サーボ関係の PAC ライブラリとして使用していたものをコマンド（ステートメント）として追加し、使いやすさの向上をはかりました。

## ST_aspACLD（ステートメント）【Ver.1.9 以降】

**機　能**

内部負荷条件値を変更します。負荷条件値は、先端負荷質量（g）、負荷重心位置（mm）（注 1）ですべて指定します。

**書　式**

ST_aspACLD < 先端負荷質量 >, < 先端負荷重心位置 X 座標 >, < 先端負荷重心位置 Y 座標 >, < 先端負荷重心位置 Z 座標 >（注 1）

注 1： 4 軸ロボットの Ver.1.9 以降の場合

ST_aspACLD < 先端負荷質量 >, < 先端負荷重心位置 X 座標 >, < 先端負荷重心位置 Y 座標 >, < 先端負荷イナーシャ >

**説　明**

先端負荷質量は、ロボットの先端軸に取り付く負荷（ツール + ワーク）の質量です。単位は（g）で指定します。

負荷重心位置は、負荷の重心位置をツール 0 座標系で指定します。単位は（mm）です。

負荷イナーシャは、負荷の重心位置を原点とした Z 軸まわりのイナーシャです。単位は $kgcm^2$ です。

6 軸ロボットの例で説明すると、ツール 0 座標系の原点は、6 軸フランジ中心、Y 成分はフランジ中心から Φ5H7（Φ6H7）ピン穴方向（オリエントベクトル方向）、Z 成分は、フランジ中心を通りフランジ面に垂直な方向（アプローチベクトル方向）、X 成分は、オリエントベクトルを Y 軸、アプローチベクトルを Z 軸としたときの右手座標系における X 軸方向（ノーマルベクトル方向）になります。プログラミングマニュアル「4.7「使用条件」における最適可搬質量設定機能」を参照してください。

先端負荷質量、負荷重心位置 X6、負荷重心位置 Y6、負荷重心位置 Z6（負荷イナーシャ）の 4 つの値の中の 1 つだけ変更する場合でも、4 つの値をすべて記述してください。

負荷条件値の切り替えには、約 0.1 秒の実行時間がかかります。頻繁に負荷条件を切り替えると動作時間遅れの要因になります。また、パス動作中に負荷条件値を切り替える場合、エンド動作になりますので、障害物近傍のパス動作中にモードを変更しないようご注意ください。

**関連項目**

プログラミングマニュアル「4.7「使用条件」における最適可搬質量設定機能」

**注意事項**

- 先端負荷質量はロボット毎に設定された範囲内の整数で指定してください。それ以外の数値を指定すると、「60d2　先端負荷設定値が許容値を超えました」が発生します。
- 先端負荷重心位置は、ロボット毎に設定された範囲を満たすように入力してください。範囲を満たさない場合、「60d2 先端負荷設定値が許容値を超えました」が発生します。
- 内部負荷条件値は、外部負荷条件値に対し、以下の範囲に設定してください。範囲を満たさない場合、「60d2 先端負荷設定値が許容値を超えました」が発生します。

  0.5× 外部負荷条件値≦内部負荷条件値≦外部負荷条件値

**用　例**

```
ST_aspACLD 8500,-50,100,80    ' 内部先端負荷条件値を先端負荷質量 8500(g)、
                              ' 負荷重 ' 心位置 X 成分 -50（mm）、Y 成分 100
                              '（mm）、Z 成分 80（mm）に設定します。
```

# ST_aspChange（ステートメント）【Ver.1.9 以降】

**機　能**

最適可搬質量設定モードの内部モードを選択します。

**書　式**

ST_aspChange < モード >

**説　明**

最適可搬質量設定モードを切り替えます。

< 設定値 >

0 → 無効
1 → PTP のみ有効
2 → CP のみ有効
3 → PTP、CP ともに有効

モード切り替えには、約 0.1 秒の実行時間がかかります。頻繁にモードを切り替えると動作遅れの要因になります。また、パス動作中にモードを切り替える場合、エンド動作になります。障害物近傍のパス動作中にモードを変更しないようにしてください。

**関連項目**

プログラミングマニュアル「4.7「使用条件」における最適可搬質量設定機能」

**注意事項**

< モード > は 0 ～ 3 の整数で指定してください。それ以外の数値を指定すると、「6003　有効な数値範囲を超えた」が発生します。

**用　例**

```
ST_aspChange 1                ' 最適可搬質量設定モードの内部モードを 1 にします。
```

# ST_SetGravity（ステートメント）【Ver.1.9 以降】

**機　能**

各関節の静荷重（重力トルク）を補正し、重力バランスを設定します。

**書　式**

ST_SetGravity

**説　明**

ロボットの各関節は、重力により、下向きの静荷重（重力トルク）を受けます。重力トルクは、ツールやワークの質量、重心位置、ロボットの姿勢により変化します。電流制限を設定し、モータのトルクを制限した場合、制限トルクが重力トルク以下になれば重力方向に動作してしまいます。その為、制限トルクから重力トルク分を補正し、電流制限設定時でも重力方向に動作しない様に重力バランスを設定する機能です。

**関連項目**

ST_SetCurLmt、ST_ResetGravity、ST_SetGrvOffset

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 先端負荷質量と負荷重心位置を正確に設定して下さい。正確に設定されていない状態で、電流制限設定値を小さく (30 以下程度 ) 設定すると重力落下する場合がありますのでご注意ください。先端負荷質量と負荷重心位置設定は、p. 4-13「4.7「使用条件」における最適可搬質量設定機能」を参照ください。
- 先端負荷質量と負荷重心位置が正確に分からない場合、正確に設定しても電流制限設定時に重力落下する場合は、重力補償補正機能 (ST_SetGrvOffset) を使用し、重力補償値の補正する事が可能です。重力補償補正ライブラリを参照ください。
- 使用条件の重力補償有効無効設定をペンダントで 1 に設定した場合、重力補償が有効になります。ペンダントで有効設定した場合、コントローラ電源を立ち上げ、キャリブレーション実行直後から、重力補償が有効になります。

**用　例**

```
ST_SetGravity                ' 重力補償を有効。
Delay 100                    ' 重力補償が反映されるのを待つ。
ST_SetCurLmt 2,30            '2 軸の電流制限値を 30%に設定する。
```

# ST_ResetGravity（ステートメント）【Ver.1.9 以降】

**機　能**

重力バランスを無効にします。

**書　式**

ST_ResetGravity

**説　明**

重力バランスを無効にします。

**関連項目**

ST_ResetCurLmt、ST_SetEralw

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 電流制限中に命令実行した場合、エラー「665b 重力補償を無効にできません」が発生します。電流制限を解除した後、再度実行してください。
- 使用条件の重力補償有効無効設定をペンダントで 0 に設定した場合でも、重力補償が無効になります。ただし、(2) 同様、電流制限中は、設定できません。

**用　例**

```
ST_ResetGravity
```

# ST_SetGrvOffset（ステートメント）【Ver.1.9 以降】

**機　能**

各関節の重力トルクより重力補償値を補正します。

**書　式**

ST_SetGrvOffset

**説　明**

ロボットの各関節は、重力により、下向きの静荷重（重力トルク）を受けます。重力補償ステートメント (ST_SetGravity) により重力バランスを設定できますが、先端負荷質量設定値と実際の負荷質量とのずれ等により、重力バランスにずれが生じる場合があります。本機能は、ロボットが停止した状態で重力トルクを推定し、重力補償値を補正します。

**関連項目**

ST_SetCurLmt、ST_SetGravity、ST_ResetGrvOffset

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- モータ電源が ON でロボットが停止した状態で実行してください。モータ電源 OFF 状態で実行した場合、エラー「6006 モータ電源がオフです」が発生します。ロボットが動作中に実行した場合、エラー「600B ロボット動作中です」が発生します。直前の動作コマンドのパス開始変位の指定は @E にすることをお薦めします（@0 でも「エラー 600B ロボット動作中」発生する可能性があります)。
- 重力補償補正を実行した姿勢から大きく変化した場合は、補正にずれが生じます。その場合は再度補正を実施して下さい。
- 使用条件の電流制限リセット設定値が１、３、５、７以外の場合は、モータ電源投入時に補正値が 0 にリセットされます。

**用　例**

```
ST_SetGrvOffset
```

# ST_ResetGrvOffset（ステートメント）【Ver.1.9 以降】

**機　能**

重力補償値の補正を初期化します。

**書　式**

ST_ResetGrvOffset

**説　明**

重力補償値の補正を初期化します。

**関連項目**

ST_SetGrvOffset

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- ロボットが停止した状態で実行してください。ロボットが動作中に実行した場合、エラー「600B ロボット動作中です」が発生します。モータ電源 OFF 中でも実行可能です。

**用　例**

```
ST_ResetGrvOffset
```

# ST_SetCurLmt（ステートメント）【Ver.1.9 以降】

**機　能**

指定した軸のモータ電流値を制限します。

**書　式**

ST_SetCurLmt < 軸番号 >、< 設定値 >

**説　明**

< 軸番号 > で指定した軸のモータ電流値（トルク）を設定値に制限します。挿入作業、突き当て作業時にワークに加わる力を制限したい場合に使用します。

< 設定値 > は、モータ定格電流値が 100 になります。各軸の許容値を超える設定をした場合は、許容値に制限されます。

1 以上の値を設定してください。0 以下の値を設定した場合は、エラー 6003「有効な数値範囲を越えました」が発生します。

**関連項目**

ST_ResetCurLmt、ST_SetGravity、ST_SetGrvOffset、ST_SetEralw

**注意事項**

- 電流制限設定中は、モータ電流が制限される為、最高速度、最高加速度で動作できません。電流制限は、必要なステップのみ使用してください。また、電流制限を使用する際は、加速度を下げてください。
- 電流制限にて推力を制限しても高速でワークが衝突すると、ワークとハンドと軸の慣性で衝撃力が発生します。電流制限はワークが接触する直前から SET し、かつ速度を下げてください。
- 電流制限は、ロボット停止状態で SET してください。パス動作中に SET すると、エラーが発生する事があります。
- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 使用条件の電流制限リセット設定値が１、３、５、７以外の場合は、モータ電源投入時に電流制限がリセットされます。
  モータ電源投入直後から電流制限を有効にする場合は、電流制限リセット設定値の最下位ビットを 1 に設定してください。

**6 軸**

- 電流制限設定する際は、必ず重力補償機能を有効にしてください。重力補償機能が無効の状態で電流制限を設定すると、エラー「665a 電流制限設定できません」が発生します。重力補償は、ST_SetGravity を参照ください。
- 先端負荷質量と負荷重心位置を正確に設定して下さい。正確に設定されていない状態で、設定値を小さく (30 以下程度 ) 設定すると重力落下する場合がありますのでご注意ください。先端負荷質量と負荷重心位置設定は、プログラミングマニュアル「4.7「使用条件」における最適可搬質量設定機能」を参照ください。
- 電流制限リセット設定値の最下位ビットが 1 に設定されている場合、モータ電源投入直後に重力落下する場合があります。モータ電源 OFF 状態で電流制限を RESET (ST_ResetCurLmt を実行 ) し、モータ電源を入れてください。
- 先端負荷質量と負荷重心位置が正確に分からない場合、正確に設定しても重力落下する場合は、重力補償補正機能 (ST_SetGrvOffset) を使用し、重力補償値の補正を行ってください。

4 軸 HS-E

エアバランスの無い 4 軸ロボットの Z 軸、T 軸に電流制限を設定する場合、制限値を小さくすると Z 軸が落下したり、T 軸が回転する場合があります。Z 軸、T 軸重力補償値設定機能 (ST_SetZBalance) を実行した後、電流制限を実行してください。

用　例

6 軸

```
ST_SetGravity                ' 重力補償を有効にする。
ST_SetGrvOffset              ' 重力補償値を補正する。
ST_SetEralw 2, 20            ' ２軸の偏差許容値を 20 度に設定する。
ST_SetCurLmt 2, 30           ' ２軸の電流制限値を 30% に設定する。
```

4 軸

```
ST_SetEralw 2, 20            ' ２軸の偏差許容値を 20 度に設定する。
ST_SetCurLmt 2, 30           ' ２軸の電流制限値を 30% に設定する。
```

4 軸 HS-E

```
ST_SetZBalance               'Z 軸重力補償値を設定する。
ST_SetEralw 3, 100           'Z 軸の偏差許容値を 100mm に設定する。
ST_SetEralw 4, 30            'T 軸の偏差許容値を 30 度に設定する。
ST_SetCurLmt 3, 10           'Z 軸の電流制限値を 10% に設定する。
ST_SetCurLmt 4, 10           'T 軸の電流制限値を 10% に設定する。
```

# ST_ResetCurLmt（ステートメント）【Ver.1.9 以降】

**機　能**

指定した軸のモータ電流制限を解除します。

**書　式**

ST_ResetCurLmt < 軸番号 >

**説　明**

< 軸番号 > で指定した軸のモータ電流制限を解除します。解除した後、電流制限値と偏差許容値はデフォルト値に戻ります。

[Ver 1.4 以前 ]

軸番号に 0 を指定すると、全軸の電流制限を解除します。

[Ver 1.5 以降 ]

軸番号に 0 を指定すると、ST_ResetCurLmt 実行タスクのセマフォ取得中の軸の全部の電流制限を解除します。

**関連項目**

ST_SetCurLmt、ST_ResetEralw

**注意事項**

- 電流制限解除時に偏差除去処理を実施します。外力にて角度偏差がある場合、設定速度、加速度により偏差除去処理時間が変化します。偏差除去処理時間を短縮するには、速度、加速度を高く設定してください。
- モータ電源 OFF 状態でも実行可能です。モータ電源 OFF 状態で、電流制限を解除する場合は、アームセマフォ取得中のタスクを終了した後、以下のプログラムを実行してください。
  PRO999
  TAKEARM
  ST_ResetCurLmt 0
  END

[Ver. 1.4 以前 ]

ロボット制御権を取得 (TAKEARM) したタスクにて実行してください。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

[Ver. 1.5 以降 ]

制御権を取得 (TAKEARM) したタスクにて実行してください。制御権未取得の軸を指定した場合は、エラー「27D*　* 軸セマフォを取得できません」が発生します。

**用　例**

```
ST_ResetCurLmt 0              ’全軸の電流制限を解除
ST_ResetGrvOffset             ’重力補償補正値をリセット
```

# ST_SetEralw（ステートメント）【Ver.1.9 以降】

**機　能**

指定した軸の偏差許容値を変更します。

**書　式**

ST_SetEralw < 軸番号 >、< 設定値 >

**説　明**

< 軸番号 > で指定した軸の偏差許容値を設定します。電流制限時に外力にて角度偏差が生じる場合に使用します。

<設定値>は、回転関節の場合はアーム関節角度(度)で、直動関節の場合はアーム関節変位(mm)で指定します。

偏差許容値とは、安全の為サーボ偏差が一定値以上になると発生するエラー「611* J* 偏差過大」の許容範囲です。電流制限を使用して組立てを行う場合、外力によりサーボ偏差が発生し、このエラーが発生する事があります。それを回避するため、一時的に偏差許容値を大きくする場合に使用します。また、電流制限調整中、重力落下を素早く検出するため、偏差許容値を小さく設定する場合にも使用します。

**関連項目**

ST_SetCurLmt、ST_ResetEralw

**注意事項**

ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
ST_SetEralw 2, 20              ’２軸の偏差許容値を 20 度に設定する。
```

# ST_ResetEralw（ステートメント）【Ver.1.9 以降】

**機　能**

指定した軸の偏差許容値をデフォルト値に戻します。

**書　式**

ST_ResetEralw < 軸番号 >

**説　明**

< 軸番号 > で指定した軸の偏差許容値をデフォルト値に戻します。

[Ver. 1.4 以前 ]

軸番号に 0 を指定すると、全軸の偏差許容値をデフォルト値に戻します。

[Ver. 1.5 以降 ]

軸番号 0 を指定すると、ST_ResetEralw 実行タスクのセマフォ取得中の軸の全部の偏差許容値をデフォルト値に戻します。

**関連項目**

ST_ResetCurLmt、ST_SetEralw

**注意事項**

電流制限解除命令 (ST_ResetCurLmt) 実行時も同様に偏差許容値がデフォルト値になります。

[Ver. 1.4 以前 ]

ロボット制御権を取得 (TAKEARM) したタスクにて実行してください。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

[Ver. 1.5 以降 ]

制御権を取得 (TAKEARM) したタスクにて実行してください。制御権未取得の軸を指定した場合は、エラー「27D*　* 軸セマフォを取得できません」が発生します。

**用　例**

```
ST_ResetEralw 0                    ’全軸の偏差許容値をデフォルト値に戻す。
```

# ST_OnSrvLock（ステートメント）【Ver.1.9 以降】

**機　能**

指定した軸をサーボロック状態にします。（４軸ロボット専用）

**書　式**

ST_OnSrvLock ＜指定軸＞

**説　明**

従来言語の ON SVLOCK 命令と同等な機能を提供します。

サーボロックとはロボットのアームが制御され、その位置が保たれている状態をいいます。

**関連項目**

ST_OffSrvLock

**注意事項**

- サーボロックは、ロボット停止状態で SET してください。パス動作中に SET すると、エラーが発生する事があります。
- ロボット制御件を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御件未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
ST_OnSrvLock 1                ’１軸のサーボロック
ST_OnSrvLock 0                ’全軸のサーボロック
```

# ST_OffSrvLock（ステートメント）【Ver.1.9 以降】

**機　能**

指定した軸のサーボロックを解除します。（４軸ロボット専用）

**書　式**

ST_OffSrvLock < 指定軸 >

**説　明**

従来言語の OFF SVLOCK 命令と同等な機能を提供します。

サーボロックとはロボットのアームが制御され、その位置が保たれている状態をいいます。サーボロックを解除するとロボットのアームは位置が保たれないため、外力が加わると位置がずれます。

**関連項目**

ST_OnSrvLock

**注意事項**

- サーボロックが解除状態にある軸は、動作コマンドを実行できません。
- サーボロック解除は、ロボット停止状態で SET してください。パス動作中に SET すると、エラーが発生する事があります。
- ロボット制御件を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御件未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 使用条件の "25: 電流制限リセット " 設定値の２ bit 目が " ０ "（初期値）の場合は、モータ電源入時にサーボロック解除がリセット ( サーボロック ) されます。モータ電源投入直後からサーボロック解除を有効にする場合は、電流制限リセット設定値に " ＋２ " を設定して下さい。
  注：使用条件 "25：電流制限リセット " 設定例

| | PWM | SVLock | Curlmt | |
|---|---|---|---|---|
| | ＊ | ＊ | ＊ | |
| SrvLock のみ有効の場合 | 0 | 1 | 0 | ＝２ |
| 全て有効の場合 | 1 | 1 | 1 | ＝７ |

**用　例**

```
ST_OnSrvLock 1                ’１軸のサーボロック
```

# ST_SetCompControl（ステートメント）【Ver.1.9 以降】

**機　能**

力制限機能を有効にします。( 6 軸専用命令 )

**書　式**

ST_SetCompControl

**説　明**

力制限機能を有効にします。ST_SetFrcLimit,ST_SetCompRate,ST_SetFrcCoord 等で設定された力制限条件を有効にします。

**関連項目**

ST_SetFrcLimit、ST_SetCompRate、ST_SetFrcCoord、ST_ResetCompControl
ST_SetCompFControl

**注意事項**

- 重力補償無効時、電流制限有効時に本ステートメントを実行した場合、エラー「60f5 力制限実行できません」が発生します。電流制限を無効、重力補償を有効にした後、再度、実行してください。電流制限無効化は、ST_ResetCurLmt、重力補償有効化は、ST_SetGravity を参照ください。
- モータ電源 OFF 状態で実行しても、力制限は有効になりません。また、力制限中にモータ電源 OFF した場合は、力制限は無効になります。
- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- ロボット停止状態で実行してください。パス動作中に本ステートメントを実行した場合は、エンド動作となります。また、ロボット動作中に本ステートメントを実行し、エラー「600b ロボット動作中です」が発生する場合、Delay 命令等でロボット停止を待ってから実行してください。
- 力制限中にロボットが外力等で動作する場合、エラー「611* 偏差過大」が発生する場合があります。エラー発生する場合は、偏差許容値を変更してください。偏差許容値変更は、ST_SetEralw を参照ください。
- ロボットが接触状態などロボットに力が加わった状態で本ステートメントを実行しないでください。力が加わった状態で力制限機能を有効にする場合は、ST_SetCompFControl を使用してください。
- ST_SetCompControl 実行後、ロボットの姿勢が大きく変化した場合は、重力補償の補正値に誤差が生じ、ロボットが重力方向に動作する場合があります。力制限中に姿勢が大きく変化する場合は、途中で ST_ResetCompControl にて力制限を無効にした後、再度 ST_SetCompControl にて力制限を有効にしてください。

**用　例**

```
ST_SetFrcCoord 1                          ' 力制限座標系を設定する。
ST_SetFrcLimit 100,0,100,100,100,100      ' 力制限割合を設定する。
ST_SetCompControl                         ' 力制限機能を有効にする。
ST_SetEralw 1,90                          ' 偏差許容値を設定する。
```

# ST_SetCompFControl（ステートメント）【Ver.1.9 以降】

**機　能**

力制限機能を有効にします。( ６軸専用命令 )

**書　式**

ST_SetCompFControl

**説　明**

ST_SetCompControl と同様、力制限機能を有効にします。ただし、ST_SetCompControl にて実行される重力補償補正処理は実行されません。

**関連項目**

ST_SetCompControl

**注意事項**

- 重力補償無効時、電流制限有効時に本ステートメントを実行した場合、エラー「60f5 力制限実行できません」が発生します。電流制限を無効、重力補償を有効にした後、再度、実行してください。
- モータ電源 OFF 状態で実行しても、力制限は有効になりません。また、力制限中にモータ電源 OFF した場合は、力制限は無効になります。
- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- ロボット停止状態で実行してください。パス動作中に本ステートメントを実行した場合は、エンド動作となります。また、ロボット動作中に本ステートメントを実行し、エラー「600b ロボット動作中です」が発生する場合、Delay 命令等でロボット停止を待ってから実行してください。
- 先端負荷設定を正確に設定してください。先端負荷設定値と実際の先端負荷が異なる場合、重力方向に落下する場合があります。また、ST_SetGrvOffset にて重力補償補正を実行して頂くと重力落下を防止できます。

**用　例**

```
ST_SetGrvOffset                          ' 重力補償補正値を計算する。
ST_SetFrcCoord 1                         ' 力制限座標系を設定する。
ST_SetFrcLimit 100,0,100,100,100,100     ' 力制限割合を設定する。
STSetCompFControl                        ' 力制限を有効化する。
```

# ST_ResetCompControl（ステートメント）【Ver.1.9 以降】

**機　能**

力制限機能を無効にします。( 6 軸専用命令 )

**書　式**

ST_ResetCompControl

**説　明**

力制限機能を無効にします。

**関連項目**

ST_SetCompControl

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- ロボット停止状態で実行してください。パス動作中に本ステートメントを実行した場合は、エンド動作となります。また、ロボット動作中に本ステートメントを実行し、ロボットが急停止したり、エラー「612* 過電流」が発生する場合、< 設定値 > を 1 に設定するか、Delay 命令等でロボット停止を待ってから実行してください。
- 本ステートメント実行中に瞬時停止し、ステップバックやプログラムリセット操作した場合、エラー「60f9 力制限無効操作異常です」が発生します。
- 力制限有効時に、ST_SetEralw にて偏差許容値を変更した場合、偏差許容値は、初期値に戻ります。しかし、ST_ResetCompControl 実行時にエラーが発生した場合、初期値に戻らない場合がありますので、力制限無効後、ST_ResetEralw にて偏差許容値を初期値に戻してください。
- エラー「608* J* 指令速度限界オーバ」が発生する場合があります。その場合、ST_ResetCompControl 実行前に ST_aspChange にて最適可搬質量 " モードを 2 又は 3 に変更し、実行後に最適可搬質量設定 " モードを元の値に戻してください。

**用　例**

```
ST_ResetCompControl          ' 力制限機能を無効化する。
ST_ResetEralw(0)             ' 偏差許容値を初期化する。
```

# ST_SetFrcCoord（ステートメント）【Ver.1.9 以降】

**機　能**

力制限設定座標系を選択します。( ６軸専用命令応 )

**書　式**

ST_SetFrcCoord < 設定値 >

**説　明**

ST_SetFrcLimit,ST_SetCompRate にて設定する力制限値の座標系を選択します。

設定値が 0 の場合、ロボットのベース座標を選択し、設定値が１の場合は、ツール座標、設定値が２の場合は、ワーク座標を選択します。

**関連項目**

ST_SetFrcLimit、ST_SetCompRate、ST_SetFrcCoord、ST_ResetCompControl

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 力制限中は、実行できません。力制限中に本ステートメントを実行すると、エラー「60fa 力制限中です」が発生します。
- < 設定値 > を 1 とし、ツール座標を選択した場合、ツール座標は、力制限有功設定 (ST_SetCompControl 実行 ) 時のツール座標となります。力制限中に changetool 命令にてツール座標を変更しても、力制限座標は変化しません。
- < 設定値 > を 2 とし、ワーク座標を選択した場合、ワーク座標は、力制限有功設定 (ST_SetCompControl 実行 ) 時のワーク座標となります。力制限中に changework 命令にてワーク座標を変更しても、力制限座標は変化しません。
- コントローラ電源立ち上げ直後は、設定値は初期化され、０（ベース座標）となります。

**用　例**

```
ST_SetFrcCoord 1                ’力制限座標系をツール座標に設定する。
Changetool 2                    ’ツール座標をツール２に設定する。
ST_SetFrcLimit 100,0,100,100,100,100
                                ’力制限割合を設定する。
ST_SetCompControl               ’ツール２座標系の Y 方向の力制限割合を 0% に設定し、
                                ’力機能を有効にする。
```

# ST_SetFrcLimit（ステートメント）【Ver.1.9 以降】

**機　能**

力制限割合を設定します。( ６軸専用命令 )

**書　式**

ST_SetFrcLimit　<X 方向制限割合 >,<Y 方向制限割合 >,<Z 方向制限割合 >,<X 回り制限割合 >,<Y 回り制限割合 >,<Z 回り制限割合 >

**説　明**

ST_SetFrcCoord にて設定された座標系の X 軸方向、Y 軸方向、Z 軸方向、X 軸回り、Y 軸回り、Z 軸回りの力制限割合を設定します。

設定値は 0 ～ 100 で設定します。小数点２桁まで有効です。

**関連項目**

ST_ResetFrcLimit、ST_SetFrcCoord、ST_SetCompControl

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 力制限中は、実行できません。力制限中に本ステートメントを実行すると、エラー「60fa 力制限中です」が発生します。
- コントローラ電源立ち上げ直後は、設定値は初期化され、X 軸方向 ,Y 軸方向 ,Z 軸方向 ,X 軸回り ,Y 軸回り ,Z 軸回りすべて 100 となります。

**用　例**

```
ST_SetFrcCoord 1              ’力制限座標系をツール座標に設定する。
ST_SetFrcLimit 100,0,100,100,100,100
                              ’力制限割合を設定する。
ST_SetCompControl             ’ツール座標系の Y 方向の力制限割合を 0% に設定し、力
                              ’機能を有効にする。
```

# ST_ResetFrcLimit（ステートメント）【Ver.1.9 以降】

**機　能**

力制限割合を初期化します。( ６軸専用命令 )

**書　式**

ST_ResetFrcLimit

**説　明**

力制限割合を初期化します。X軸方向,Y軸方向,Z軸方向,X軸回り,Y軸回り,Z軸回りすべて100%となります。

**関連項目**

ST_SetFrcLimit

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 力制限中は、実行できません。力制限中に本ステートメントを実行すると、エラー「60fa 力制限中です」が発生します。

**用　例**

```
ST_ResetCompControl          ’力制限を無効にします。
ST_ResetFrcLimit             ’力制限割合を初期化します。
```

# ST_SetCompRate（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の柔らかさの割合を設定します。( ６軸専用命令 )

**書　式**

ST_SetCompRate <X 方向柔らかさ >,<Y 方向柔らかさ >,<Z 方向柔らかさ >,<X 回り柔らかさ >,<Y 回り柔らかさ >,<Z 回り柔らかさ >

**説　明**

ST_SetFrcCoord にて設定された座標系の X 軸方向、Y 軸方向、Z 軸方向、X 軸回り、Y 軸回り、Z 軸回りの柔らかさの割合を設定します。

設定値は 0 ～ 100 の範囲で設定します。0 の時最も柔らかくなります。小数点２桁まで有効です。

**関連項目**

ST_ResetCompRate、ST_SetFrcCoord、ST_SetCompControl

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 力制限中は、実行できません。力制限中に本ステートメントを実行すると、エラー「60fa 力制限中です」が発生します。
- コントローラ電源立ち上げ直後は、設定値は初期化され、X 軸方向 ,Y 軸方向 ,Z 軸方向 ,X 軸回り ,Y 軸回り ,Z 軸回りすべて 100 となります。

**用　例**

```
ST_SetFrcCoord1              ’力制限座標系をツール座標に設定する。
ST_SetCompRate 100,0,100,100,100,100
                             ’柔らかさ割合を設定する。
ST_SetCompControl            ’ツール座標系の Y 方向の柔らかさ割合を 0% に設定し、
                             ’力機能を有効にする。
```

# ST_ResetCompRate（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の柔らかさの割合を初期化します。( ６軸専用命令 )

**書　式**

ST_ResetCompRate

**説　明**

X 軸方向、Y 軸方向、Z 軸方向、X 軸回り、Y 軸回り、Z 軸回りの柔らかさの割合を初期化し、すべて 100 とします。

**関連項目**

ST_SetCompRate

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 力制限中は、実行できません。力制限中に本ステートメントを実行すると、エラー「60fa 力制限中です」が発生します。

**用　例**

```
ST_ResetCompControl          ’力機能を無効にする。
ST_ResetCompRate             ’柔らかさ割合を初期化する。
```

# ST_SetFrcAssist（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時のオフセット力を設定します。( 力制限特殊機能ステートメント )( ６軸専用命令 )

**書　式**

SetFrcAssist　<X 方向オフセット力 >,<Y 方向オフセット力 >,<Z 方向オフセット力 >,<X 回りオフセットモーメント >,<Y 回りオフセットモーメント >,<Z 回りオフセットモーメント >

**説　明**

SetFrcCoord にて設定された座標系の X 軸方向、Y 軸方向、Z 軸方向、X 軸回り、Y 軸回り、Z 軸回りのオフセット力、モーメントを設定します。力制限最大値の 10% が設定最大値となります。

力設定値の単位は [N] です。モーメント設定値の単位は [Nm] です。小数点 1 桁まで有効です。

**関連項目**

ST_ResetFrcAssist、ST_SetFrcCoord、ST_SetCompControl

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- オフセット力、モーメントを加える方向にロボットが動作する場合があります。動作する場合は、設定値を下げてください。
- コントローラ電源立ち上げ直後は、設定値は初期化され、X 軸方向 ,Y 軸方向 ,Z 軸方向 ,X 軸回り ,Y 軸回り ,Z 軸回りすべて 0 となります。

**用　例**

```
ST_SetFrcCoord 1              ’力制限座標系をツール座標に設定する。
ST_SetFrcAssist -30,0,0,0,0,0
                              ’-X 方向に 30[N] のオフセット力を設定する。
ST_SetFrcLimit 0,100,100,100,100,100
                              ’力制限割合を設定する。
CALL SetCompControl           ’力機能を有効にする。X 方向は０％に力制限され、-X
                              ’方向に 30[N] の力を加える。
```

# ST_ResetFrcAssist（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時のオフセット力を初期化します。( 力制限特殊機能 )
( ６軸専用命令 )

**書　式**

ST_ResetFrcAssist

**説　明**

ST_SetFrcCoord にて設定された座標系の X 軸方向、Y 軸方向、Z 軸方向、X 軸回り、Y 軸回り、Z 軸回りのオフセット力、モーメントを 0 にします。

**関連項目**

ST_SetFrcAssist

**注意事項**

ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
ST_ResetCompControl          ' 力制限を無効にする。
ST_ResetFrcAssist            ' オフセット力、モーメントを 0 に設定する。
```

# ST_SetCompJLimit（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の電流制限値を設定します。( 力制限特殊機能 ) ( ６軸専用命令 )

**書　式**

ST_SetCompJLimit <J1 電流制限値 >,< J2 電流制限値 >,< J3 電流制限値 >,<J4 電流制限値 >,< J5 電流制限値 >,< J6 電流制限値 >

**説　明**

力制限時の電流制限値を設定します。モータ定格電流値が 100 になります。ST_SetFrcLimit にて全方向を 0 に設定した場合、モータ電流は、設定値以下に制限されます。

設定値は 0 ～ 100 の範囲で設定します。

**関連項目**

ST_ResetCompJLimit、ST_SetCompControl

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- コントローラ電源立ち上げ直後は、設定値は初期化され、電流制限値は、すべて 0 となります。
- ST_SetCompJLimit に比較的大きな値を設定した場合、ロボットが発振的になりエラーが発生する場合があります。その場合、力制限の柔らかさを ST_SetCompRate,ST_SetDumpRate にて調整してください。

**用　例**

```
ST_SetFrcCoord 1                          ' 力制限座標系をツール座標に設定する。
ST_SetCompJLimit 30,0,0,0,0,0             'J1 の電流制限値を 30% に設定する。
ST_SetFrcLimit 100,0,100,100,100,100      ' 力制限割合を設定する。
ST_SetCompControl                         ' 力機能を有効にする。
```

# ST_ResetCompJLimit（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の電流制限値を初期化します。( 力制限特殊機能 )

( ６軸専用命令 )

**書　式**

ST_ResetCompJLimit

**説　明**

力制限時の電流制限値を初期化し、全軸 0 に設定ます。

**関連項目**

ST_SetCompJLimit、ST_SetCompControl

**注意事項**

ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
ST_ResetCompControl          ’力機能を無効にする。
ST_ResetCompJLimit           ’電流制限値を初期化する。
```

# ST_SetCompVMode（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の速度制御モードを設定します。( 力制限特殊機能 )
( ６軸専用命令 )

**書　式**

ST_SetCompVMode

**説　明**

ST_SetCompControl 実行時の力制限速度制御モードを有効にします。

**関連項目**

ST_ResetCompVMode

**注意事項**

ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
ST_SetCompVMode              ’力制限速度制御モードを有効にする。
ST_SetCompControl            ’力制限を有効にする。
```

# ST_ResetCompVMode（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の速度制御モードを無効にします。( 力制限特殊機能 )

( ６軸専用命令 )

**書　式**

ST_ResetCompVMode

**説　明**

ST_SetCompControl 実行時の力制限速度制御モードを無効にします。

**関連項目**

ST_SetCompVMode

**注意事項**

ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
ST_ResetCompVMode          ’力制限速度制御モードを無効にする。
ST_SetCompControl          ’力制限を有効にする。
```

# ST_SetCompEralw（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時のツール端の位置、姿勢偏差許容値を設定します。( 6 軸専用命令 )

**書　式**

ST_SetCompEralw <X 方向偏差許容値 >,<Y 方向偏差許容値 >,<Z 方向偏差許容値 >,<X 回り偏差許容値 >,<Y 回り偏差許容値 >,<Z 回り偏差許容値 >

**説　明**

力制限時のツール端の位置、姿勢偏差許容値を設定します。X,Y,Z 方向の偏差許容値の単位は、(mm)、X,Y,Z 回りの偏差許容値の単位は、( 度 ) です。小数点 1 桁まで有効です。

**関連項目**

ST_ResetCompEralw、ST_SetFrcCoord

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- コントローラ電源立ち上げ直後の偏差許容値の初期値は、X,Y,Z 方向が 100(mm)、X,Y,Z 回りが 30( 度 ) です。X,Y,Z 周りは、最大値が 175( 度 ) です。最大値を超える設定をした場合、エラー「6003 有効な数値範囲を超えた」が発生します。
- ツール端の位置、姿勢偏差が設定した許容値を超えた場合、エラー「60f8　力制限時位置偏差過大異常」が発生します。
- 偏差設定座標系は、ST_SetFrcCoord にて設定された座標系となります。例えば ST_SetFrcCoord(1) と設定した場合は、SetCompEralw の設定座標系は、ツール座標系となります。

**用　例**

```
ST_SetFrcCoord 2                        ' 力制限座標系をワーク座標にします。
ST_SetFrcLimit 100,0,100,100,100,100    ' ワーク座標の Y 方向の力制限割合を 0% にします。
ST_SetCompEralw 10,150,10,5,5,5         ' ワーク座標 X,Z 方向の偏差許容値を 10(mm),
                                        'Y 方向の偏差許容値を 150(mm),X,Y,Z 回りの
                                        ' 偏差許容値を 5( 度 ) にします。
```

# ST_ResetCompEralw（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時のツール端の位置、姿勢偏差許容値を初期化します。( ６軸専用命令 )

**書　式**

ST_ResetCompEralw

**説　明**

力制限時のツール端の位置、姿勢偏差許容値を初期化します。X,Y,Z 方向の偏差許容値の初期値は、100(mm)、X,Y,Z 回りの偏差許容値の初期値は、30( 度 ) です。

**関連項目**

ST_SetCompEralw

**注意事項**

ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。

**用　例**

```
ST_ResetCompEralw
```

# ST_SetDampRate（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の粘性割合を設定します。( ６軸専用命令 )

**書　式**

ST_SetDampRate <X 方向粘性割合 >,<Y 方向粘性割合 >,<Z 方向粘性割合 >,<X 回り粘性割合 >,<Y 回り粘性割合 >,<Z 回り粘性割合 >

**説　明**

ST_SetFrcCoord にて設定された座標系の X 軸方向、Y 軸方向、Z 軸方向、X 軸回り、Y 軸回り、Z 軸回りの粘性割合を設定します。

設定値は 0 ～ 100 の範囲で設定します。0 の時最も粘性が小さくなります。小数点２桁まで有効です。

**関連項目**

ST_ResetDampRate、ST_SetFrcCoord、ST_SetCompRate

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 力制限中は、実行できません。力制限中に本ステートメントを実行すると、エラー「60fa 力制限中です」が発生します。
- コントローラ電源立ち上げ直後は、設定値は初期化され、X 軸方向 ,Y 軸方向 ,Z 軸方向 ,X 軸回り ,Y 軸回り ,Z 軸回りすべて１００となります。
- ST_SetCompRate にて設定した柔らかさ割合より小さい値を設定しないでください。柔らかさ割合より小さい値を設定した場合、ロボットが振動し、エラー停止する場合があります。
- ST_SetDampRate 実行後、ST_SetCompRate を設定すると、粘性割合は、ST_SetCompRate の設定値となります。

**用　例**

```
ST_SetFrcCoord 1                    ’力制限座標系をツール座標に設定する。
ST_SetCompRate 100,0,100,100,100,100
                                    ’柔らかさ割合を設定する。
ST_SetDampRate 100,20,100,100,100,100
                                    ’粘性割合を設定する。
ST_SetCompControl                   ’ツール座標系の Y 方向の柔らかさ割合を 0%、粘性割合
                                    ’を 20% に設定し、力機能を有効にする。
```

# ST_ResetDampRate（ステートメント）【Ver.1.9 以降】

**機　能**

力制限時の粘性割合を初期化します。( ６軸専用命令 )

**書　式**

ST_ResetDampRate

**説　明**

Ｘ軸方向、Ｙ軸方向、Ｚ軸方向、Ｘ軸回り、Ｙ軸回り、Ｚ軸回りの粘性割合を初期化し、すべて１００にします。

**関連項目**

ST_SetDampRate

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- 力制限中は、実行できません。力制限中に本ステートメントを実行すると、エラー「60fa 力制限中です」が発生します。

**用　例**

```
ST_ResetDampRate                ’粘性割合を初期化する。
```

# ST_SetZBalance（ステートメント）【Ver.1.9 以降】

**機　能**

Z 軸、T 軸の重力補償値を設定します。（4 軸専用）

**書　式**

ST_SetZBalance

**説　明**

エアバランスのない 4 軸ロボットの Z 軸・T 軸は、重力により、下向きの静荷重（重力トルク）を受けており、コントローラ電源投入時には初期の重力補償値が設定されています。電流制限設定時、制限値を重力トルク以下に設定すると Z 軸が落下または上昇し、T 軸が回転します。（4 軸専用）　本機能は、ロボットが停止した状態で重力トルクを推定し、重力補償値を設定します。本機能を使用することにより、電流制限値を小さく設定した場合の Z 軸の落下または上昇及び、T 軸の回転を防止します。

**関連項目**

ST_ResetZBalance、ST_SetCurLmt

**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- モータ電源が ON でロボットが停止した状態で実行してください。モータ電源 OFF 状態で実行した場合、エラー「6006 モータ電源がオフです」が発生します。ロボットが動作中に実行した場合、エラー「600B ロボット動作中です」が発生します。直前の動作コマンドのパス開始変位の指定は @E にすることをお薦めします（@0 でも「エラー 600B ロボット動作中」発生する可能性があります）。
- 重力補償値設定を実行した後、ワーク重量が変化した場合は、設定値にずれが生じます。その場合は再度本命令を実行してください。

**用　例**

```
ST_SetZBalance
```

# ST_ResetZBalance（ステートメント）【Ver.1.9 以降】

**機　能**

重力補償値の補正を初期化します。(4 軸専用)

**書　式**

ST_ResetZBalance

**説　明**

4 軸ロボットの Z 軸、T 軸の重力補償値の設定を初期化します。

**関連項目**

ST_SetZBalance

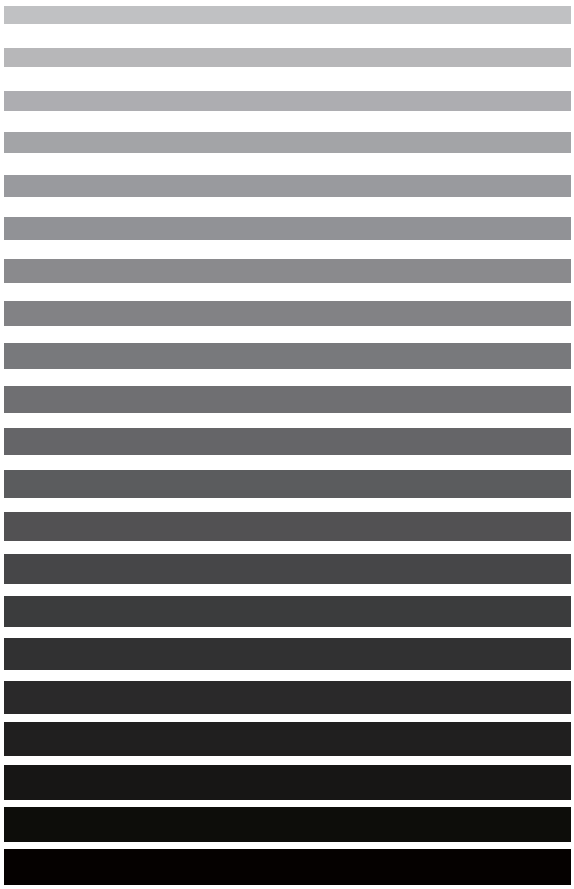
**注意事項**

- ロボット制御権を取得 (TAKEARM) したタスクにて実行下さい。ロボット制御権未取得の場合は、エラー「21F7 アームセマフォを取得できません」が発生します。
- ロボットが停止した状態で実行してください。ロボットが動作中に実行した場合、エラー「600B ロボット動作中です」が発生します。モータ電源 OFF 中でも実行可能です。

**用　例**

```
ST_ResetZBalance
```

# 第 13 章
# 入出力制御文



この章では、各種の I/O の入出力を制御するコマンドについて説明します。

# 第 13 章　入出力制御文

## 13.1　I/O ポート

### IN（ステートメント）【SLIM 準拠】

**機　能**

I/O 変数で示される I/O ポートからデータを読み込みます。

**書　式**

IN < 算術変数名 > = <I/O 変数 >

**説　明**

<I/O 変数 > で示されるポートのデータを < 算術変数名 > で指定した変数に代入します。

<I/O 変数 > は、DEFIO 文で宣言された変数か、または I/O 型変数です。

**関連項目**

OUT、DEFIO

**注意事項**

SINGLE で宣言した I/O 型変数の内容が、単精度実数として表現できない場合はエラー 777F（実数変換失敗）が発生します。

**用　例**

```
DEFINT Li1, Li2
DEFIO samp1 = INTEGER, 220   'samp1 をポート 220 から始まる INTEGER 型 I/O 変数と
                             ' して宣言します。
IN Li1 = samp1               'samp1 のデータを Li1 へ代入します。
IN Li2 = IO[240]             ' ポート 240 のデータを Li2 へ代入します。
OUT samp1 = Li1              'Li1 のデータを samp1 で宣言したポートから出力しま
                             ' す。
OUT IO[240] = Li2            'Li2 のデータをポート 240 から出力します。
```

# OUT（ステートメント）【SLIM 準拠】

**機　能**

I/O 変数で示される I/O ポートにデータを出力します。

**書　式**

OUT <I/O 変数 > = < 出力データ >

**説　明**

<I/O 変数 > で示されるポートアドレスに < 出力データ > の値を出力します。

<I/O 変数 > は、DEFIO 文で宣言された変数か、または I/O 型変数です。

**関連項目**

IN、DEFIO

**用　例**

```
DEFINT Li1, Li2
DEFIO samp1 = INTEGER, 220   'samp1 をポート 220 から始まる INTEGER 型 I/O 変数と
                             ' して宣言します。
IN Li1 = samp1               'samp1 のデータを Li1 へ代入します。
IN Li2 = IO[240]             ' ポート 240 のデータを Li2 へ代入します。
OUT samp1 = Li1              'Li1 のデータを samp1 で宣言したポートから出力しま
                             ' す。
OUT IO[240] = Li2            'Li2 のデータをポート 240 から出力します。
```

# IOBLOCK ON/OFF（ステートメント）【SLIM 準拠】

**機　能**

動作命令を実行中に、I/O 命令や演算命令などの非動作命令を並列に実行します。

**書　式**

IOBLOCK {ON|OFF}

**説　明**

IOBLOCK ON と IOBLOCK OFF は対に用いられ、それに囲まれた範囲内では、ロボットが動作命令を実行中に、その動作命令に引き続く複数の I/O 命令や演算命令などの非動作命令が並列に実行されます。

ロボット動作命令が現れると、一度並列実行は打ち切られ、現在実行中の動作命令が完了（パス動作時はパス開始）するのを待って、次の動作命令を実行します。次の動作命令のあとに、非動作命令が続く場合、これらも動作命令と並列に実行されます。IOBLOCK 内の動作命令実行中に、IOBLOCK OFF が実行されると、動作命令の実行を待って、次ステップへ進みます。

**注意事項**

- 以下の場合、並列処理を行ないません。
  いずれの場合も動作命令が完了（パス動作指定時はパス動作開始）するまで、次ステップの実行は待たされます。
  - 動作命令に動作オプションを付加した場合。
  - CHANGEWORK，CHANGETOOL，SPEED，JSPEED，ACCEL，JACCEL を実行する場合。
  - 従来言語ライブラリ「aspChange」、「aspACLD」、アーム動作ライブラリ「mvSetPulseWidth」、「mvSetTimeOut」、「mvReverseFlip」、「mvResetPulseWidth」、「mvResetTimeOut」を実行する場合。
- ロボット動作中にステップ停止・瞬時停止などをかけ、その後再開した場合には、非動作コマンドとの並列動作を行なわない場合もあります。
- IOBLOCK 内で動作命令が 2 回続くとき、最初の動作実行中にステップ停止させた場合、次の動作が終了してからステップ停止します。最初の動作実行中に、瞬時停止を行ない次にステップ起動させた場合も、次の動作が終了してからステップ停止します。

  例 :

  ```
  IOBLOCK ON
  MOVE P，J0      ← J0 へ動作中にステップ停止させても、次の動作 MOVE P，J1 は動作されます。
                    その後、ステップ停止します。
  MOVE P，J1
  SET IO[1]
  IOBLOCK OFF
  ```

- IOBLOCK の有効範囲は、定義したプログラムでのみ有効であり、サブプログラムでは有効でありません。以下のようなプログラムがある場合、SUB1 では PRO1 で定義した IOBLOCK は有効ではありません。IOBLOCK の有効範囲はプログラムごとに独立です。

  例 :

  ```
  PROGRAM PRO1
  TAKEARM
  DEFPOS lp1，lp2
  IOBLOCK ON
  CALL SUB1
  MOVE P，lp1，lp2
  SET IO[1]
  ```

```
SET IO[2]
IOBLOCK OFF
      :
      :
```

- ティーチチェックモードでは IOBLOCK は無効になります。

用　例

```
TAKEARM                          ’ロボット制御権を取得します
IOBLOCK ON                       ’次の動作命令と並列に I/O 命令が実行されます。
MOVE P (902.7, 0, 415.3, 180, 50, 180, 1)
                                 ’（902.7，0，415.3，180，50，180，1）の座標へ（PTP
                                 ’制御）移動します。
SET IO[240]                      ’BIT 型ポート 240 を ON にします。
SET IO[241], 40                  ’BIT 型ポート 241 を 40ms 間 ON にします。
SET IO[SOL1]                     ’I/O 変数 SOL1 で示されるポートを ON にします。
SET IO[104 TO 110]               ’BIT 型ポート 104 ～ 110 を ON にします。
IF IO[242] THEN
  RESET IO[240]                  ’BIT 型ポート 240 を OFF にします。
  RESET IO[SOL1]                 ’I/O 変数 SOL1 で示されるポートを OFF にします。
  RESET IO[104 TO 110]           ’BIT 型ポート 104 ～ 110 を OFF にします。
ENDIF
IOBLOCK OFF
```

# SET（ステートメント）【SLIM 準拠】

**機　能**

I/O ポートを ON にします。

**書　式**

SET <I/O 変数 >[,< 出力時間 >]

**説　明**

<I/O 変数 > で指定されたポートを ON にします。

< 出力時間 > を指定するとパルス出力します。（出力時間の単位は ms です。）

< 出力時間 > を指定すると、その時間が経過するまで次の命令へは進みません。また、出力時間の設定値は最低出力時間であり、実際の出力時間は、タスクのプライオリティなどに影響され変化します。

**関連項目**

RESET、DEFIO

**注意事項**

- 出力時間指定を行う場合、動作中の別プログラムの存在、ティーチングペンダント操作状況、周辺機器との通信などにより出力時間が長くなる場合があります。
- 出力時間指定を行う場合、コントローラの基準時間単位が 16.7ms であるため、指定時間が ±16.7ms ずれることがあります。
- 時間指定 SET を使用する場合には以下の 2 点にご注意ください。
  - 時間指定 SET によりポートが ON 中に別タスクにより同じポートを RESET すると、その時点からそのポートは OFF 状態となります。（時間指定 SET が無効になる。）
  - 時間指定 SET によりポートが ON 中に別タスクにより同じポートを RESET し続けて SET すると、指定時間経過後に、指定ポートが OFF します。（時間指定 SET が有効になる。）
- 出力時間を使用する際には、命令実行中に瞬時停止を行い再起動すると、一時停止中も出力時間が経過することに注意してください。
- 汎用出力、ハンド出力、内部 IO ポート以外の番号を指定するとエラー 21FB( システムエリア書込みエラー ) が発生します。特に、ポート番号を指定する際、算術式を用いる場合にはご注意ください。
- SINGLE で宣言した I/O 型変数には使用できません。実行した場合、エラー 777F（実数変換失敗）が発生します。

用　例

```
TAKEARM                          ’ロボット制御権を取得します
IOBLOCK ON                       ’次の動作命令と並列に I/O 命令が実行されます。
MOVE P (902.7, 0, 415.3, 180, 50, 180, 1)
                                 ’（902.7，0，415.3，180，50，180，1）の座標へ
                                 ’（PTP 制御）移動します。
SET IO[240]                      ’BIT 型ポート 240 を ON にします。
SET IO[241], 40                  ’BIT 型ポート 241 を 40ms 間 ON にします。
SET IO[SOL1]                     ’I/O 変数 SOL1 で示されるポートを ON にします。
SET IO[104 TO 110]               ’BIT 型ポート 104 ～ 110 を ON にします。
IF IO[242] THEN
  RESET IO[240]                  ’BIT 型ポート 240 を OFF にします。
  RESET IO[SOL1]                 ’I/O 変数 SOL1 で示されるポートを OFF にします。
  RESET IO[104 TO 110]           ’BIT 型ポート 104 ～ 110 を OFF にします。
ENDIF
IOBLOCK OFF

TAKEARM                          ’ロボット制御権を取得します
SET IO[I1 * 5]                   ’I1 の値 *5 の数値のポートを ON にします。
SET IO[27-24]                    ’27-24=3 のポートを ON にします。
```

# RESET（ステートメント）

**機　能**

I/O ポートを OFF にします。

**書　式**

RESET <I/O 変数 >

**説　明**

<I/O 変数 > で指定されたポートを OFF にします。

**関連項目**

SET、DEFIO

**用　例**

```
TAKEARM                          ' ロボット制御権を取得します
IOBLOCK ON                       ' 次の動作命令と並列に I/O 命令が実行されます。
MOVE P (902.7, 0, 415.3, 180, 50, 180, 1)
                                 '（902.7，0，415.3，180，50，180，1）の座標へ（PTP
                                 ' 制御）移動します。
SET IO[240]                      'BIT 型ポート 240 を ON にします。
SET IO[241], 40                  'BIT 型ポート 241 を 40ms 間 ON にします。
SET IO[SOL1]                     'I/O 変数 SOL1 で示されるポートを ON にします。
SET IO[104 TO 110]               'BIT 型ポート 104 ～ 110 を ON にします。
IF IO[242] THEN
  RESET IO[240]                  'BIT 型ポート 240 を OFF にします。
  RESET IO[SOL1]                 'I/O 変数 SOL1 で示されるポートを OFF にします。
  RESET IO[104 TO 110]           'BIT 型ポート 104 ～ 110 を OFF にします。
ENDIF
IOBLOCK OFF
```

# DETECT ON/OFF（ステートメント）【Ver.3.2 以降】

機　能

ハンド入力の立上り及び立下りを検知し、その時のロボットの位置を変数に格納します。

書　式

- DETECT ON の書式
  DETECT ON < ハンド入力ポート番号 >, < ROB >, < 格納先先頭変数 >, < 最大格納数 >, < 検知回数格納先変数 > ［, < 検知方向 >］
- DETECT OFF の書式
  DETECT OFF < 検知結果格納先変数 > ［, < ハンド入力ポート番号 >］

説　明

DETECT ON を実行してから DETECT OFF を実行するまでの間、< ハンド入力ポート番号 > で指定されたハンド入力の立上り（OFF から ON）、または立下り（ON から OFF）を検知し、< 格納先先頭変数 > で指定された変数に格納します。

DETECT は拡張機能です。暗証番号「1312」で機能拡張した後にご使用ください。機能拡張については、操作ガイドの「拡張機能の追加」をご参照ください。

DETECT ON

< ハンド入力ポート番号 > には検知するハンド入力のポート番号（48~55）を指定します。

< ROB > には、" ROB " を指定してください。

< 格納先先頭変数 > にはロボットの位置を格納する変数を指定します。複数回検知した場合、その度に変数番号に 1 を足して、その番号の変数へ格納します。

P 型及び J 型のグローバル変数を指定することができます。変数番号だけを記述した場合は P 型グローバル変数となります。たとえば 1 を記述した場合、P1 を指定したことになります。

< 最大格納数 > には、検知した時の位置の最大格納数を指定します。

< 検知回数格納先変数 > には、検知した回数を格納する変数を指定します。I 型グローバル変数を指定することができます。

< 検知方向 > には、立上りと立下りのどれで検知するかを指定します。

- 0 : 立上りを検知
- 1 : 立下りを検知
- 2 : 立上りと立下りの両方を検知

省略時は 0（立上り）が指定されます。

DETECT OFF

< 検知結果格納先変数 > には検知結果を格納する変数を指定します。I 型グローバル変数を指定することができます。

- 0 : 正常終了
- -1 : 検知回数と格納した位置の数が一致しない
- -2 : 検知回数が < 最大格納数 > より多い
- -3 : 上記 -1 と -2 の両方

< ハンド入力ポート番号 > には検知を終了するハンド入力のポート番号（48~55）を指定します。省略した場合は、DETECT ON 時に指定したハンド入力全てを終了します。ただし、他のプログラムの DETECT ON で指定したハンド入力の検知は終了しません。

DETECT OFF を実行しなくても、END が実行された時点で、そのプログラム内の DETECT ON で指定したハンド入力の検知を全て終了します。

同一プログラム内で DETECT ON を複数実行する場合、同じハンド入力ポート番号、同じ検知方向を指定すると、最後に実行された DETECT ON の設定が有効になります。

例

```
DETECT ON, 50, ROB, P10, 30, I5, 0
DETECT ON, 50, ROB, P20, 10, I1, 0        ' この行の設定が有効になる
```

同じハンド入力ポート番号を指定しても、検知方向が異なれば全て別々の設定となります。

例

```
DETECT ON, 50, ROB, P10, 10, I1, 0        ' 立上り時の位置を P10 から入れる
DETECT ON, 50, ROB, P20, 10, I2, 1        ' 立下り時の位置を P20 から入れる
DETECT ON, 50, ROB, P30, 10, I3, 2        ' 立上り、下り時の位置を P30 から入れる
```

異なるプログラムで同じハンド入力ポート番号、同じ検知方向を指定した時には、エラー「77FB I/O 番号が重複しています」が発生します。

**注意事項**

ハンド入力ポートのデジタルフィルタ時間（デジタル信号処理）は通常 3msec ですが、DETECT コマンドを使用すると 50μsec へ切替わります。リレー接点入力など、チャタリングが発生する機器を使用すると、正しく機能しない場合がありますのでご注意ください。

**用　例**

```
PROGRAM PRO1
 TAKEARM
 MOVE P, P1
 DETECT ON, 48, ROB, 0, 5, I10, 0     ' ハンド入力 48 の立上がり時に位置を P0 から
                                      ' 最大 5 個格納する。
                                      ' 実際に検知した回数を I10 に格納する。
 DETECT ON, 49, ROB, 10, 5, I11, 1    ' ハンド入力 49 の立下がり時に位置を
                                      'P10 から最大 5 個格納する。
                                      ' 実際に検知した回数を I11 に格納する。
 MOVE L, P2
 DETECT OFF, I0                       ' ハンド入力 48 と 49 の検知を終了し
                                      ' 検知結果を I0 に格納する。
 IF I0 <> 0 THEN                      ' 検知結果が " 正常終了 " ではなかった場合
  STOP                                ' プログラムを停止する。
 ELSE                                 ' 検知結果が " 正常終了 " であった場合
  MOVE L, P3                          'P3 へ移動する
 ENDIF
END
```

# 13.2　RS232C および Ethernet ポート

# INPUT（ステートメント）

**機　能**

RS232C および Ethernet ポートからのデータを得ます。

**書　式**

INPUT [#< 回線番号 >,]< 変数名 >[,WTIME=< タイムアウト時間 > [,RVAL=< 格納変数 >]]

[,< 変数名 >[,WTIME=< タイムアウト時間 > [,RVAL=< 格納変数 >]]…]

**説　明**

RS232C および Ethernet ポートに受信したデータを、< 変数名 > で指定した変数に格納します。

< 回線番号 > には、使用する RS232C および Ethernet ポートの回線番号を指定します。< 回線番号 > を省略するとデフォルト値の ch2 となります。ch1 はティーチングペンダント用に使用するため指定できません。(「2.4.1　回線番号」参照 )　複数の情報を、それと同数の変数で受ける場合は、個々の変数間にカンマ (,) を付けます。

デリミタ（CR または CR+LF）の直前までのすべてを、指定した変数に格納します。

デリミタは、変数には読み込まれません。

ボーレートは、システムパラメータで指定します。

＜タイムアウト時間＞　［RC7_Ver2.2 以降］

変数毎にタイムアウト時間を指定できます。入力データがない場合、指定時間を経過した後、現在の変数に対しての入力待ちを終了し、次に制御が移ります。単位は ms ですが、実際の遅延時間は 1/60s 単位で増減します。

＜格納変数＞　［RC7_Ver2.2 以降］

＜タイムアウト時間＞の設定と対で使います。入力データを受けて正常に抜けた場合 TRUE(1) が入り、タイムアウトで抜けたときは指定した変数に FALSE(0) が入ります。

**注意**

- **データの受信に先立って、受信データの入力バッファに残っているデータをクリアするために、FLUSH コマンドを実行してください。**
- **入力したデータが、代入する変数の型が表せる値の最大値を超えている場合、結果はその型の最大値となります。**

**関連項目**

FLUSH、PRINT、WRITE

用　例

例 1：

```
DIM li1 As Integer
defint li
DEFSTR ls1, ls2, ls3, ls4
INPUT #1, ls1                    'ch2 からのデータを ls1 に書きます。
INPUT #1, li1, ls2, ls3          'ch2 からのデータを li1, ls2, ls3 に書きます。
INPUT ls4                        'ch2 からのデータを ls4 に書きます。
INPUT ls4 ,WTIME=100,RVAL=li     ' 入力がない場合 100msec 時間後次のコマン
                                 ' ドへ処理が移ります。li にはタイムアウト
                                 ' で抜けたことを表す 0 が入ります。
```

注意

次の用例 2、用例 3 は、文字列情報がカンマ（,）で区切られ、その区切られた数と同じ要素数の変数で受けることができるので余分な文字列演算の必要はありません。

例 2：外部から、"505.425,-125.325,400.238" を送った場合（6 軸ロボットの例）

```
DIM lv1 AS VECTOR
INPUT #1,lv1
LETP P11=lv1                     ' 受信した x,y,z を P11 へ代入
LETR P11=RVEC(P10)               ' 姿勢データに基準位置 P10 の値を代入
LETF P11=FIG(P10)                ' 形態データに基準位置 P10 の値を代入
MOVE P,@E P11,S=100
```

例 3：外部から、"505.425,-125.325,400.238,180,0,180,5" を送った場合（6 軸ロボットの例）

```
INPUT #1,P11
MOVE P,@E P11,S=100
```

# LINEINPUT（ステートメント）

**機　能**

RS232C および Ethernet ポートからデリミタまでのデータを読み込み、文字列型変数に代入します。

**書　式**

LINEINPUT [#< 回線番号 >,] < 文字列型変数名 >[,WTIME=< タイムアウト時間 > [,RVAL=< 格納変数 >]]

**説　明**

RS232C および Ethernet ポートに受信したデータから、デリミタ（CR または CR+LF）の直前までのすべての文字を、< 文字列型変数名 > で指定した変数に格納します。デリミタは、変数には読み込まれません。

< 回線番号 > には、使用する RS232C および Ethernet ポートの回線番号を指定します。< 回線番号 > を省略するとデフォルト値のch2となります。ch1はティーチングペンダント用に使用するため指定できません。（「2.4.1 回線番号」参照）

< 文字列変数名 > には、文字列型変数の名前を指定します。

＜タイムアウト時間＞　［RC7_Ver2.2 以降］

指定時間内にデリミタの入力がない場合、lineinput 文の実行を終了し、次のコマンドに制御が移ります。単位は ms ですが、実際の遅延時間は 1/60s 単位で増減します。

＜格納変数＞　［RC7_Ver2.2 以降］

＜タイムアウト時間＞の設定と対で使います。入力データを受けて正常に抜けた場合 TRUE(1) が入り、タイムアウトで抜けたときは指定した変数に FALSE(0) が入ります。

**用　例**

```
DEFSTR ls1,ls2,ls3,ls4
DEFINT li
LINEINPUT #0, ls1              ' ポート 1 から文字列を受信し、ls1 へ代入します。
LINEINPUT #1, ls2              ' ポート 2 から文字列を受信し、ls2 へ代入します。
LINEINPUT ls3                  ' ポート 2 から文字列を受信し、ls3 へ代入します。
LINEINPUT #1,ls4,WTIME=100,RVAL=li
                               ' デリミタまでの入力がない場合、100msec
                               ' 時間後次のコマンドへ処理が移ります。
                               'li にはタイムアウトで抜けたことを表す
                               '0 が入ります。
```

# PRINT（ステートメント）

**機　能**

RS232C および Ethernet ポートからデータを出力します。

**書　式**

PRINT [#< 回線番号 >,]< メッセージ >[< セパレータ >< メッセージ >…][< セパレータ >]

**説　明**

RS232C および Ethernet ポートから、< メッセージ > の値を外部へ出力します。

< 回線番号 > には、使用する RS232C および Ethernet ポートの回線番号を指定します。< 回線番号 > を省略するとデフォルト値の ch2 となります。ch1 はティーチングペンダント用に使用するため指定できません。

（「2.4.1　回線番号」参照）

< セパレータ > には、カンマ (,) とセミコロン (;) が使用できます。

カンマ (,) の場合は、< メッセージ > と < メッセージ > の間に、スペース文字を入れます。

< メッセージ > の後にカンマ (,) を指定し、その後に < メッセージ > がなかった場合は、最後の < メッセージ > の後にデリミタ（CR または CR+LF）を送出して、出力を完了します。

セミコロン (;) の場合は、< メッセージ > との間にはスペース文字を入れません。< メッセージ > の後にセミコロン (;) を指定し、その後に < メッセージ > がなかった場合は、最後の < メッセージ > の後にデリミタ（CR または CR+LF）を送出して、出力を完了します。

ポーズ型データを出力した場合は、各要素をスペース文字で区切って、出力されます。

**関連項目**

WRITE

**用　例**

```
DEFINT li1, li2
DEFSTR ls1, ls2
PRINT #1, li1, ls1; ls2      'li1, ls1, ls2 の値を ch2 から出力します。(li1 と ls1
                             ' の間はスペースが空き、ls1 と ls2 の間はスペースが空
                             ' きません。)
PRINT li2                    'li2 の値を ch2 から出力します。
```

# WRITE（ステートメント）

**機　能**

RS232C および Ethernet ポートからデータを出力します。

**書　式**

WRITE [#< 回線番号 >,]< メッセージ >[,< メッセージ >…]

**説　明**

RS232C および Ethernet ポートから、< メッセージ > の値を外部へ出力します。

< 回線番号 > には、使用する RS232C および Ethernet ポートの回線番号を指定します。< 回線番号 > を省略するとデフォルト値のch2となります。ch1はティーチングペンダント用に使用するため指定できません。（「2.4.1 回線番号」参照）

複数の < メッセージ > を書き出す場合は、カンマ (,) で区切ります。

PRINT 文とは、以下の点が異なります。

- 文字列のデータは、二重引用符 (") で囲まれて、出力されます。
- 出力情報の終わりに、デリミタ（CR または CR+LF）が付きます。
- < メッセージ > を区切っているカンマ (,) は、そのまま出力されます。
- ポーズ型データを出力した場合は、各要素をカンマ (,) で区切って、出力されます。

**関連項目**

PRINT

**用　例**

```
DEFINT li1, li2
DEFSTR ls1, ls2
WRITE #1, li1, ls1, ls2      'li1, ls1, ls2 の値を ch2 から出力します。
WRITE li2                    'li2 の値を ch2 から出力します。
```

# FLUSH（ステートメント）

**機　能**

入力バッファをクリアします。

**書　式**

FLUSH [#< 回線番号 >]

**説　明**

RS232C および Ethernet ポートの入力バッファをクリアします。

< 回線番号 > には、入力バッファをクリアする RS232C および Ethernet ポートの回線番号を指定します。< 回線番号 > を省略するとデフォルト値の ch2 となります。ch1 はティーチングペンダント用に使用するため指定できません。（「2.4.1　回線番号」参照）

データの受信に先立って、受信データの入力バッファに残っているデータをクリアするために、FLUSH コマンドを実行してください。

**関連項目**

INPUT

**用　例**

```
FLUSH #1                    'ch2 の回線の入力バッファをクリアします。
FLUSH                       'ch2 の回線の入力バッファをクリアします。
```

# 13.3　シリアルバイナリ通信（RS-232C および Ethernet ポート）

## PRINTB【Ver.1.5 以降】

**機　能**

1 バイトデータを RS-232C および Ethernet ポートに出力します。

**書　式**

PRINTB # ＜ポート番号＞ , ＜出力データ格納 I 変数＞

＜ポート番号＞

出力ポート番号
( 1 ：コントローラ RS-232C ポート、-1 ： μ Vision ボード RS-232C ポート
4 ～ 7 : Ethernet サーバポート、8 ～ 15 : Ethernet クライアントポート )

＜出力データ格納 I 変数＞

出力データが格納された I 型変数

**説　明**

＜出力データ格納 I 変数＞で指定された I 型変数の下位バイトを、指定ポートに出力します。

**用　例**

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample
     .
     .
     .
     PRINTB #1,I10          'I10 に格納されたデータの下位バイトを RS-232C ポートに
                            ' 出力
     .
     .
     .
END
```

# INPUTB【Ver.1.5 以降】

**機　能**

１バイトデータを RS-232C および Ethernet ポートから入力します。

**書　式**

INPUTB [#＜ポート番号＞,]＜入力データ格納I変数＞[,WTIME=<タイムアウト時間>[,RVAL=<格納変数>]]

＜ポート番号＞

入力ポート番号
( 1 ：コントローラ RS-232C ポート、-1 ： $\mu$ Vision ボード RS-232C ポート
4 ～ 7 : Ethernet サーバポート、8 ～ 15 : Ethernet クライアントポート )
省略時　1 : コントローラ RS-232C ポート

＜入力データ格納 I 変数＞

入力データが格納される I 型変数の番号

＜タイムアウト時間＞　[RC7_Ver2.2 以降 ]

入力データがない場合、指定時間を経過した後に inputb 文の実行を終了し、次のコマンドに制御が移ります。単位は ms ですが、実際の遅延時間は 1/60s 単位で増減します。

＜格納変数＞　[RC7_Ver2.2 以降 ]

＜タイムアウト時間＞の設定と対で使います。入力データを受けて正常に抜けた場合 TRUE(1) が入り、タイムアウトで抜けたときは指定した変数に FALSE(0) が入ります。

**説　明**

＜入力データ格納 I 変数＞で指定された I 型変数に、ポートから入力したデータを格納します。

**注意**

**タイムアウト時間を設定せずにこのコマンドでデータ入力を行なう場合、データがポートに存在しないと無限待ちとなります。データの有無の確認には com_state コマンドを使用してください。**

**用　例**

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample
    defint li
    .
    .
    .
    INPUTB #1,I10                 'I10 に RS-232C ポートから入力したデータを格納
    INPUTB                        ' 入力がない場合 100msec 時間後次のコマン
    #1,I10,WTIME=100,RVAL=li      ' ドへ処理が移ります。li にはタイムアウト
                                  ' で抜けたことを表す 0 が入ります。
    .
    .
    .
END
```

# LPRINTB【Ver.1.5 以降】

**機　能**

複数バイトデータを RS-232C および Ethernet ポートに出力します。

**書　式**

LPRINTB # ＜ポート番号＞ , ＜格納配列先頭要素＞ , ＜出力バイト数＞

＜ポート番号＞

出力ポート番号
(1：コントローラ RS-232C ポート、-1：μVision ボード RS-232C ポート
4 ～ 7：Ethernet サーバポート、8 ～ 15：Ethernet クライアントポート )

＜格納配列先頭要素＞

出力データが格納された配列の先頭要素

＜出力バイト数＞

出力するデータのバイト数

**説　明**

出力データが格納された配列の指定要素から指定出力バイト数分だけ指定ポートにデータを出力します。

**用　例**

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample
      .
      .
      .
      LPRINTB #1,I64, 30          'I64 から I93 までのデータの下位バイトを連続して
                                  'RS-232C ポートに出力
      .
      .
      .
END
```

# LINPUTB【Ver.1.5 以降】

**機　能**

複数バイトデータを RS-232C および Ethernet ポートから入力します。

**書　式**

LINPUTB [# ＜ポート番号＞ ,] ＜格納配列先頭要素＞ , ＜入力バイト数＞ [,WTIME=< タイムアウト時間 >[,RVAL=< 格納変数 >]]

＜ポート番号＞

入力ポート番号
( 1 ：コントローラ RS-232C ポート、-1 ： μVision ボード RS-232C ポート
4 ～ 7 : Ethernet サーバポート、8 ～ 15 : Ethernet クライアントポート )
省略時　1 : コントローラ RS-232C ポート

＜格納配列先頭要素＞

入力データが格納される配列の先頭要素

＜入力バイト数＞

入力するデータのバイト数

＜タイムアウト時間＞　[RC7_Ver2.2 以降 ]

入力バイト数分の入力データがない場合、指定時間を経過した後に linputb 文の実行を終了し、次のコマンドに制御が移ります。単位は ms ですが、実際の遅延時間は 1/60s 単位で増減します。

＜格納変数＞　[RC7_Ver2.2 以降 ]

＜タイムアウト時間＞の設定と対で使います。入力データを受けて正常に抜けた場合 TRUE(1) が入り、タイムアウトで抜けたときは指定した変数に FALSE(0) が入ります。

**説　明**

配列の指定要素から、指定入力バイト数分だけ指定ポートからデータを入力します。

**注意**

**タイムアウト時間を設定せずに、このコマンドでデータ入力を行う場合、＜入力バイト数＞分のデータがポートに存在しない場合、無限待ちとなります。データの有無の確認には com_state コマンドを使用してください。**

**用　例**

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample
    defint li
    .
    .
    LINPUTB #1, I64, 30                  'I64 から I93 までにデータを連続して RS-232C
                                         ' ポートから入力
LINPUTB #1,I64, 30,WTIME=100,RVAL=li     ' ＜入力バイト数＞分の入力がない場合、100msec
                                         ' 時間後次のコマンドへ処理が移ります。li には
                                         ' タイムアウトで抜けたことを表す 0 が入ります。
    .
    .
END
```

# com_encom【Ver.1.5 以降】

**機　能**

RS-232C ポートまたは Ethernet クライアントポートを使ってバイナリ通信行うための準備をします。

RS-232C ポートの場合は、バイナリ通信のためにポートを占有します。

Ethernet クライアントポートを使う場合はクライアントポートのオープンを行いサーバとの間で接続を確立します。

**書　式**

com_encom # ＜ポート番号＞

＜ポート番号＞

出力ポート番号
1：コントローラ RS-232C ポート
8 ～ 15：Ethernet クライアントポート

**説　明**

バイナリ通信中に WINCAPS Ⅲなどで PC ←→コントローラ間通信を行う際、通信データが交じり合わないようにします。

バイナリ通信が終了したら、com_discom コマンドで通信ポートを開放する必要があります。

**注意**

- このコマンドを実行したあとで、WINCAPS Ⅲなどでデータを送信してもバイナリ通信データとして取り扱われます。RS-232C ポートを閉じてしまうことではありませんのでご注意ください。
- Ethernet 接続では、このコマンドを実行した後でも WINCAPS Ⅲとのデータ通信に影響はありません。

**注意事項**

com_discom,com_encom を短い間隔で行うと、com_encom でクライアントオープン失敗エラーが発生することがあります。上記用例のように回線未確立を確認するプログラムを使用するようにしてください。

For loop の中で com_discom、com_encom を使用しないでください。

**用　例**

<EtherNet ポート使用時の場合 >

```
PROGRAM sample
DEFINT Flg

com_state #8,Flg              ’ポートの状態を取得
IF Flg=-1 THEN                ’回線未確立なら
com_encom #8                  ’ポートオープン
ENDIF

END
```

# com_discom【Ver.1.5 以降】

**機　能**

RS-232C ポートをバイナリ通信終了のため開放します。（COM ポート開放）
Ethernet クライアントポート使用時にクライアントポートのクローズを
おこない接続を切断します。

**書　式**

com_discom # ＜ポート番号＞

**説　明**

バイナリ通信を行なうために占有されていた通信ポートを、他の用途のために開放します。

**注意**

**このコマンドを実行するとデータの交じり合いを防ぐため、RS-232C ポートを一度クリアします。**

**注意**

**Ethernet 使用時この処理を行うと未送信データは送信完了を待たずに切断する為、受信側において受信待ちし続けることがあります。**

**注意事項**

com_discom,com_encom を短い間隔で行うと、com_encom でクライアントオープン失敗エラーが発生することがあります。com_encom の用例のように回線未確立を確認するプログラムを使用するようにしてください。
For loop の中で com_discom、com_encom を使用しないでください。

**用　例**

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample
      .
      .
      .
      com_discom #1                    ' ポートのバイナリ通信からの解放
      .
      .
      .
END
```

# com_state【Ver.1.5 以降】

**機　能**

RS-232C および Ethernet ポートの状態を取得します。

**書　式**

com_state # ＜ポート番号＞ , ＜状態格納 I 型変数＞

**説　明**

RS-232C および Ethernet 通信バッファの残りデータ数を取得します。ただし通信ポートエラー発生時は -1 を返します。

Ethernet 使用時は該当ポートの接続が確立していないときも -1 を返します。

**用　例**

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample
      .
      .
      .
      com_state #1,I280          'I280 に通信バッファに溜まっているデータ数を
                                 ' 取得する
      .
      .
      .
END
```

## 13.4　ティーチングペンダント

# PRINTMSG（ステートメント）

**機　能**

メッセージをキャプションとアイコンを付けて、ティーチングペンダントのカラー液晶に表示します。

**書　式**

PRINTMSG＜メッセージ文字列＞,＜アイコンタイプ＞,＜キャプション文字列＞

**説　明**

引数のそれぞれで指定したメッセージを、キャプションとアイコンを付けて、ティーチングペンダントのカラー液晶に表示します。

| アイコンタイプ | アイコン |
|---|---|
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |

＜メッセージ文字列＞は最大 60 文字、＜キャプション文字列＞は最大 40 文字までです。

**関連項目**

PRINTDBG、PRINTLBL

**用　例**

```
PRINTMSG "Hello World !", 1, " メッセージ "
                              '"Hello World !" をキャプションとアイコンを付けて
                              ' 表示します。
```

# PRINTDBG（ステートメント）

**機　能**

デバッグウィンドウにデータを出力します。

**書　式**

PRINTDBG < メッセージ >[< セパレータ >< メッセージ >…][< セパレータ >]

**説　明**

ペンダントのデバッグウィンドウに、< メッセージ > の値を出力します。
< セパレータ > には、カンマ (,) とセミコロン (;) が使用できます。
カンマ (,) の場合は、カンマと後ろの < メッセージ > の間に、スペース文字を入れます。
セミコロン (;) の場合は、< メッセージ > との間にはスペース文字を入れません。
ポーズ型データを出力した場合は、各要素をスペース文字で区切って表示します。
< メッセージ > の最後に < セパレータ > がない場合は改行を行ないます。

**関連項目**

PRINTMSG、PRINTLBL

**用　例**

```
PRINTDBG "DEBUG"                    ' デバッグウィンドウに "DEBUG" を出力します。
```

# BUZZER（ステートメント）

**機　能**

ブザーを鳴らします。

**書　式**

BUZZER < 鳴動時間 >

**説　明**

< 鳴動時間 > で指定した時間、ティーチングペンダントのブザーを鳴らします。
< 鳴動時間 > の単位は、msec です。

**用　例**

```
BUZZER 3000                         '3 秒間ブザーを鳴らします。
```

# PRINTWARNING（ステートメント）【RC7_Ver.2.2 以降】

**機　能**

ペンダント警告メッセージ表示部分へメッセージを表示します。

**書　式**

PRINTWARNING ＜表示メッセージ＞ ［, ＜メッセージ色＞］

**説　明**

ペンダント警告メッセージ表示部分へメッセージを表示します。

表示メッセージの最大長は半角で 55 文字までです。それ以上は切り捨てられます。

メッセージ色

0：黒　1：青　2：緑　3：赤　その他：黒　（省略した場合も黒です。）

メッセージのみの表示でありエラーログには記録されません。

ペンダントでのみの機能です。

**用　例**

```
PRINTWARNING "Hello World!",1
```

# PRINTLBL（ステートメント）

**機　能**

ユーザ定義ボタンにラベル（キャプション）を設定します。

**書　式**

PRINTLBL < パネル番号 >,< ボタン番号 >,< キャプション文字列 >

**説　明**

ティーチングペンダントの各個操作パネルのユーザ定義ボタンにラベル（キャプション）を設定します。

< パネル番号 > には 0 ～ 6 までの番号が指定できます。

< ボタン番号 > には 0 ～ 11 までの番号が指定できます。

< キャプション文字列 > には半角 6 文字までの文字列を指定します。

**関連項目**

PRINTDBG、PRINTMSG

**用　例**

```
PRINTLBL 3, 1, "E_STOP"        '3 番目のパネルの 1 番目のボタンのラベルを "E_ STOP"
                               ' に設定します。
```

# 13.5　TP 簡易操作盤【Ver.1.5 以降】

任意の大きさ・位置・色のボタンの作成、画面への張り込みを PAC 言語にて実現します。またボタン、ページ設定 PAC プログラムを一度動作させることで操作盤を有効とします。なお一度設定するとクリア操作を行わない限り操作盤は動作し続けます。

### 使用できるボタン・画面

- ボタン・・・５００個まで
- 画面・・・・５０画面まで

### 操作盤画面作成のためのコマンド

| | |
|---|---|
| set_button | ボタンパラメータの設定（属性により表示 ON/OFF も兼ねる） |
| set_page | ページパラメータの設定（属性により表示 ON/OFF も兼ねる） |
| change_bCap | ボタンキャプション変更 |
| change_pCap | ページキャプション変更 |
| disp_page | ページ表示 |

### コマンドにより設定されるパラメータ

| | |
|---|---|
| ボタンパラメータ | • インデックス<br>• 表示位置（左上 x 座標、左上 y 座標、右下 x 座標、右下 y 座標）<br>• ボタンタイプ（タッチスイッチ [ 形状のバリエーション有り ]、数値表示ボックス、数値入力ボックス、デジスイッチ、ランプ [ 形状のバリエーション有り ]、ページ切り替えボタン）<br>• ステータス（予約）<br>• 背面色<br>• 文字色<br>• 有効フラグ<br>• 表示フラグ<br>• キャプション<br>• 変数タイプ<br>• 変数番号<br>• IO 番号<br>• ページ番号<br>• 変更フラグ（予約）<br>• 結果（予約） |
| ページパラメータ | インデックス、画面タイプ（固定）、ステータス（予約）、背面色、文字色、有効フラグ、表示フラグ、キャプション、変更フラグ（予約）、結果（予約） |

### TP 簡易操作盤をプログラムする

操作盤画面の表示を行なうためには以下の操作を行なってください。

1) ボタンパラメータの設定
   set_button 命令を使って、ボタンのパラメータを決定します。これをすべてのパラメータについて実行します。
   ( 例 ) 番号 1 のボタンの背景色 (7) を黒 (0) に設定する。
   set_button １、７、０
2) ページパラメータの設定
   set_page 命令を使って、ページのパラメータを決定します。これをすべてのパラメータについて実行します。
   ( 例 ) 2 ページ目のページの背景色 (3) を赤 (4) に設定する。
   set_page ２、３、４
3) ボタンキャプションの設定
   change_bCap 命令を使って、ボタンのキャプションを決定します。
   ( 例 ) 番号２のボタンのキャプションを " 準備 " に設定する。
   change_bCap ２、" 準備 "
4) ページキャプションの設定
   change_pCap 命令を使って、ページのキャプションを決定します。
   ( 例 ) 3 ページ目のキャプションを " 画面３ " に設定する。
   change_pCap ３、" 画面３ "
5) ページの表示
   disp_page 命令を使って、ページを表示します。
6) 操作盤画面の選択
   ティーチングペンダントの基本画面から [F9: 操作盤 ] を選択し、操作盤を表示します。
   ( 画面例 )



# set_button【Ver.1.5 以降】

機　能

ボタンパラメータの設定を行ないます。

書　式

set_button ＜ボタン番号＞ , ＜パラメータ種類＞ , ＜変更パラメータ値＞

＜ボタン番号＞

操作盤に配置されるボタンの全体の中での番号

＜パラメータ種類＞

色、位置などボタンの属性（下表を参照）

＜変更パラメータ値＞

ボタンそれぞれがもつ変更パラメータ値（下表を参照）

| ＜パラメータ種類＞ | 意味 | ＜変更パラメータ値＞（注 1） |
|---|---|---|
| 1 | 左上 X 座標 | ０～６４０、ドット単位 |
| 2 | 左上 Y 座標 | ０～３５０、ドット単位 |
| 3 | 右下 X 座標 | ０～６４０、ドット単位 |
| 4 | 右下 Y 座標 | ０～３５０、ドット単位 |
| 5 | ボタン種類 | ０：なにもなし<br>１：ラベル<br>２：線<br>１７：２Ｄボタン（変数変更）<br>１８：３Ｄボタン（変数変更）<br>１９：３Ｄボタン（変数変更）<br>２０：円（変数変更）<br>３３：２ＤＬＥＤ（ランプ）<br>３４：円ＬＥＤ（ランプ）<br>３５：３Ｄボタン（ＩＯ変更）<br>３６：予約（使用しないでください。）<br>３７：予約（使用しないでください。）<br>３８：ボックス |
| 6 | ボタンステータス | ０：文字センタリング<br>１：文字左つめ<br>２：文字右つめ |

| ＜パラメータ種類＞ | 意味 | ＜変更パラメータ値＞（注 1） |
|---|---|---|
| 7 | 背景色 | ０：黒<br>１：青<br>２：緑<br>３：シアン<br>４：赤<br>５：マジェンタ<br>６：茶<br>７：明るい灰色<br>８：灰色<br>９：明るい青<br>１０：明るい緑<br>１１：明るいシアン<br>１２：明るい赤<br>１３：明るいマジェンタ<br>１４：黄<br>１５：白 |
| 8 | 文字色 | ０：黒<br>１：青<br>２：緑<br>３：シアン<br>４：赤<br>５：マジェンタ<br>６：茶<br>７：明るい灰色<br>８：灰色<br>９：明るい青<br>１０：明るい緑<br>１１：明るいシアン<br>１３：明るいマジェンタ<br>１４：黄<br>１５：白 |
| 9 | 使用可能状態 | ０：使用せず<br>１：使用する |
| 10 | 可視状態 | ０：見せない<br>１：見せる |
| 11 | 変数種類（注 2） | １：整数<br>２：実数<br>３：倍精度実数<br>４：文字列（３２文字まで） |
| 12 | 変数番号 | 変数変更ボタンにより変更される変数番号 |
| 13 | I/O 番号 | IO 変更ボタンにより変更される変数番号 (128~511) |
| 14 | 表示ページ番号 | ボタンが表示されるページ番号 |

注１：変更パラメータ値に整数以外の値を指定するとエラー " データタグ異常 " が発生します。整数値で入力してください。

注２：数値入力ボタンで実数、倍精度実数を使用する場合、実数では１３文字、倍精度実数では２２文字の領域を必要としますのでボタン領域をそれに合わせて設定してください。

説　明

＜パラメータ種類＞の値を＜変更パラメータ値＞に変えて、＜ボタン番号＞で指定した番号のボタンの仕様を変更します。

**ボタン種類別サンプル例**

ボタン種類ごとのサンプル例を示します。

①ラベル

②線

③２ D ボタン（変数変更）

④３ D ボタン ( 変数変更 )

⑤３ D ボタン ( 変数変更 )

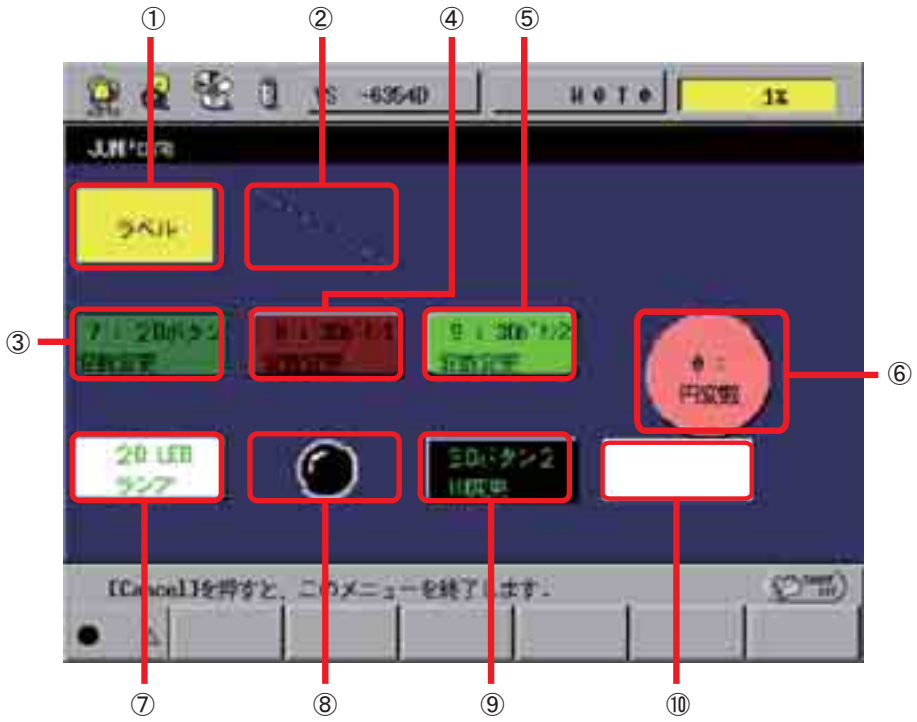
⑥円

⑦２ DLED

⑧円 LED

⑨３ D ボタン（IO 変更）

⑩ボックス



用　例

```
'!TITLE " <タイトル> "
PROGRAM sample1
     .
     .
     .
     set_button (btn_no),(1),(minx)
     set_button (btn_no),(2),(miny)
     set_button (btn_no),(3),(maxx)
     set_button (btn_no),(4),(maxy)
     .
     .
     .
END
```

# set_page【Ver.1.5 以降】

機　能

ページパラメータの設定を行ないます。

書　式

set_page ＜ページ番号＞ , ＜パラメータ種類＞ , ＜変更パラメータ値＞

＜ページ番号＞

操作盤に配置されるページの全体の中での番号

＜パラメータ種類＞

色、位置などページの属性（下表を参照）

＜変更パラメータ値＞

ページそれぞれがもつ変更パラメータ値（下表を参照）

| ＜パラメータ種類＞ | 意味 | ＜変更パラメータ値＞ |
|---|---|---|
| 1 | ページ種類 | 0 固定 |
| 2 | ボタンステータス | 未使用 |
| 3 | 背景色 | ０：黒<br>１：青<br>２：緑<br>３：シアン<br>４：赤<br>５：マジェンタ<br>６：茶<br>７：明るい灰色<br>８：灰色<br>９：明るい青<br>１０：明るい緑<br>１１：明るいシアン<br>１２：明るい赤<br>１３：明るいマジェンタ<br>１４：黄<br>１５：白 |

| ＜パラメータ種類＞ | 意味 | ＜変更パラメータ値＞ |
|---|---|---|
| 4 | 文字色<br>（Ver. 1.5 では未使用） | ０：黒<br>１：青<br>２：緑<br>３：シアン<br>４：赤<br>５：マジェンタ<br>６：茶<br>７：明るい灰色<br>８：灰色<br>９：明るい青<br>１０：明るい緑<br>１１：明るいシアン<br>１２：明るい赤<br>１３：明るいマジェンタ<br>１４：黄<br>１５：白 |
| 5 | 使用可能状態 | ０：使用せず<br>１：使用する |
| 6 | 可視状態 | ０：見せない<br>１：見せる |

**説　明**

＜パラメータ種類＞の値を＜変更パラメータ値＞に変えて、＜ページ番号＞で指定した番号のページの仕様を変更します。

**用　例**

```
'!TITLE“＜タイトル＞“
PROGRAM sample2
     .
     .
     .
     set_page panel_no,P_BGCOLOR,GLAY          'ページの背景色設定
     set_page panel_no,P_USESTATE,ON           'ページ使用の設定
     set_page panel_no,P_VISSTATE,ON           'ページ可視の設定
     .
     .
     .
END
```

# change_bCap 【Ver.1.5 以降】

## 機　能

ボタンのキャプション設定を行ないます。

## 書　式

change_bCap ＜ボタン番号＞ , ＜指定キャプション＞

＜ボタン番号＞

　　操作盤に配置されるボタンの全体の中での番号

＜指定キャプション＞

　　ボタンの中心に表示される文字列

( 図示例 )




ボタンキャプション例

## 説　明

＜ボタン番号＞で指定した番号のボタンの中心に表示される文字列を＜指定キャプション＞変更します。

## 用　例

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample3
      .
      .
      .
     bcap4 = " ワーク切断 "
     btn_no = 3
      .
      .
      .
     change_bCap btn_no,bcap4
      .
      .
      .
END
```

# change_pCap 【Ver.1.5 以降】

## 機　能

ページのキャプション設定を行ないます。

## 書　式

change_pCap ＜ページ番号＞ , ＜指定キャプション＞

＜ページ番号＞

　　操作盤に配置されるページの全体の中での番号

＜指定キャプション＞

　　ページのタイトルバーに表示される文字列

（表示例）




## 説　明

ページ番号で指定した番号のページのタイトルバーに表示される文字列を変更します。

## 用　例

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample4
      .
      .
      .
     pcap4 = " ワーク切断 "
     page_no = 3
      .
      .
      .
     change_pCap page_no,pcap4
      .
      .
      .
END
```

# disp_page【Ver.1.5 以降】

**機　能**

ページの表示を行ないます。

**書　式**

disp_page ＜ページ番号＞

＜ページ番号＞

操作盤に配置されるページの全体の中での番号

**説　明**

ページ番号で指定した番号のページを操作盤画面に表示します。

**用　例**

```
'!TITLE " ＜タイトル＞ "
PROGRAM sample3
      .
      .
      .
      'disp_page panel_no          ' 指定画面の表示
      .
      .
      .
END
```

## 13.5.1 操作盤画面作成のプログラム例

全体の命令を用いた操作盤画面作成のプログラム例を次に示します。

```
'!TITLE " <タイトル> "
PROGRAM BUTTON3D_VAL_3
' 色の定義
#define BLACK                    0          ' 黒
#define BLUE                     1          ' 青
#define GREEN                    2          ' 緑
#define CYAN                     3          ' シアン
#define RED                      4          ' 赤
#define MAGENTA                  5          ' マジェンタ
#define BROWN                    6          ' 茶
#define LIGHTGLAY                7          ' 明るい灰色
#define GLAY                     8          ' 灰色
#define LIGHTBLUE                9          ' 明るい青
#define LIGHTGREEN               10         ' 明るい緑
#define LIGHTCYAN                11         ' 明るいシアン
#define LIGHTRED                 12         ' 明るい赤
#define LIGHTMAGENTA             13         ' 明るいマジェンタ
#define YELLOW                   14         ' 黄
#define WHITE                    15         ' 白

' ボタンの定義
#define LABEL                    1          ' ラベル
#define LINE                     2          ' 線
#define 2DBUTTON_V               17         '2D ボタン ( 変数変更 )
#define 3DBUTTON_V               18         '3D ボタン ( 変数変更 )
#define 3DBUTTON_V2              19         '3D ボタン 2( 変数変更 )
#define CIRCLE                   20         ' 円 ( 変数変更 )
#define 2DLED                    33         '2DLED( ランプ )
#define CIRCLELED                34         ' 円 LED( ランプ )
#define 3DBUTTON_IO              35         '3D ボタン (IO 変更 )

' ページパラメータ
#define P_BGCOLOR                3          ' ページ背景色
#define P_CHARCOLOR              4          ' ページ文字色
#define P_USESTATE               5          ' 使用可能状態
#define P_VISSTATE               6          ' 可視状態

' ボタンパラメータ
#define X_UPPERLEFT_P            1          ' 左上 X 座標
#define Y_UPPERLEFT_P            2          ' 左上 Y 座標
#define X_LOWERRIGHT_P           3          ' 右下 X 座標
#define Y_LOWERRIGHT_P           4          ' 右下 Y 座標
#define B_KIND                   5          ' ボタン種類
#define B_BGCOLOR                7          ' ボタン背景色
#define B_FGCOLOR                8          ' ボタン文字色
#define B_USESTATUS              9          ' ボタン使用可能状態
#define B_VISSTATUS              10         ' ボタン可視状態
#define B_VALUEKIND              11         ' ボタン変数種類 ( 固定 )
#define B_VALUE_NO               12         ' ボタン変数番号
#define B_IO_NO                  13         ' ボタン I/O 番号
```

```
#define B_DISP_PNO                    14          ' ボタンの表示ページ番号

' 状態定義
#define ON                            1           ' 使用する
#define OFF                           0           ' 使用しない
#define I_VAL                         1           ' 整数型

' ボタンアドレス
#define IO_PB_ADRS                    170         ' 操作盤ボタン先頭 I/O 番号


defint btn_no,minx,maxx,miny,maxy,loopcnt
defint enable,visible,var_type,var_index
defint panel_no,io_no,io_adrs,btn_adrs
defstr panel_cap
defstr bcap0,bcap1,bcap2,bcap3,bcap4,bcap5,bcap6,bcap7
   panel_no = 3

   panel_cap = " 段取画面設定（画面 3）"
   loopcnt = 0

   change_pCap panel_no,panel_cap                ' ページのタイトル設定
   set_page panel_no,P_BGCOLOR,GLAY              ' ページの背景色設定
   set_page panel_no,P_USESTATE,ON               ' ページ使用の設定
   set_page panel_no,P_VISSTATE,ON               ' ページ可視の設定

' 各パラメータの初期化
   btn_adrs = 30
   io_no = IO_PB_ADRS
   reset io[128 to 133]
   enable = ON
   visible = ON
   var_type = I_VAL
   var_index = 1

   bcap0 = " 段取り "+chr$(10)+" データ 1"
   bcap1 = " 段取り "+chr$(10)+" データ 2"
   bcap2 = " 段取り "+chr$(10)+" データ 3"
   bcap3 = " 段取り "+chr$(10)+" データ 4"
   bcap4 = " 段取り "+chr$(10)+" データ 5"
   bcap5 = " 段取り "+chr$(10)+" データ 6"
   bcap6 = " 画面 0"+chr$(10)+" へ戻る "
   bcap7 = " ここを "+chr$(10)+" タッチ "

   while loopcnt < 6                             '6 回ループ
     btn_no = btn_adrs + loopcnt
     minx = 10 + ((loopcnt mod 6)*100)
     miny = 50 + ((loopcnt / 6)*100)
     maxx = 100 + ((loopcnt mod 6)*100)
     maxy = 120 + ((loopcnt / 6)*100)
     io_adrs = IO_PB_ADRS + loopcnt

' ここは共通処理
```

```
    set_button (btn_no),(1),(minx)
    set_button (btn_no),(2),(miny)
    set_button (btn_no),(3),(maxx)
    set_button (btn_no),(4),(maxy)
    set_button (btn_no),(9),(enable)
    set_button (btn_no),(10),(visible)
    set_button (btn_no),(11),(var_type)
    set_button (btn_no),(12),(var_index)
    set_button (btn_no),(14),(panel_no)

'ラベル表示
    select case loopcnt
    case 0
      set_button btn_no,B_KIND,LABEL                ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs             'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GLAY              ' 背景色設定
      set_button btn_no,B_FGCOLOR,RED               ' 前面色設定
      change_bCap btn_no,bcap0                      ' ボタン名設定

    case 1
      set_button btn_no,B_KIND,LABEL                ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs             'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GLAY              ' 背景色設定
      set_button btn_no,B_FGCOLOR,BLACK             ' 前面色設定
      change_bCap btn_no,bcap1                      ' ボタン名設定

    case 2
      set_button btn_no,B_KIND,LABEL                ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs             'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GLAY              ' 背景色設定
      set_button btn_no,B_FGCOLOR,WHITE             ' 前面色設定
      change_bCap btn_no,bcap2                      ' ボタン名設定

    case 3
      set_button btn_no,B_KIND,LABEL                ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs             'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GLAY              ' 背景色設定
      set_button btn_no,B_FGCOLOR,YELLOW            ' 前面色設定
      change_bCap btn_no,bcap3                      ' ボタン名設定

    case 4
      set_button btn_no,B_KIND,LABEL                ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs             'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GLAY              ' 背景色設定
      set_button btn_no,B_FGCOLOR,LIGHTMAGENTA      ' 前面色設定
      change_bCap btn_no,bcap4                      ' ボタン名設定

    case 5
      set_button btn_no,B_KIND,LABEL                ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs             'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GLAY              ' 背景色設定
      set_button btn_no,B_FGCOLOR,LIGHTBLUE         ' 前面色設定
      change_bCap btn_no,bcap5                      ' ボタン名設定
```

```
    end select
    loopcnt = loopcnt + 1
  wend

'下段表示
  loopcnt = 0
  while loopcnt < 6                                    '6 回ループ
    btn_no = btn_adrs + 10 + loopcnt
    minx = 10 + ((loopcnt mod 6)*100)
    miny = 120 + ((loopcnt / 6)*100)
    maxx = 100 + ((loopcnt mod 6)*100)
    maxy = 190 + ((loopcnt / 6)*100)
    io_adrs = IO_PB_ADRS+6+loopcnt
    var_index = var_index + 1

'ここは共通処理
    set_button (btn_no),(1),(minx)
    set_button (btn_no),(2),(miny)
    set_button (btn_no),(3),(maxx)
    set_button (btn_no),(4),(maxy)
    set_button (btn_no),(9),(enable)
    set_button (btn_no),(10),(visible)
    set_button (btn_no),(11),(var_type)
    set_button (btn_no),(12),(var_index)
    set_button (btn_no),(14),(panel_no)

'ラベル表示
    select case loopcnt
    case 0
      set_button btn_no,B_KIND,3DBUTTON_V              'ボタンタイプ設定
      set_button btn_no,B_KIND,3DBUTTON_V2             'ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs                'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,MAGENTA              '背景色設定
      set_button btn_no,B_FGCOLOR,RED                  '前面色設定
      var_index = loopcnt
      set_button btn_no,B_VALUE_NO,var_index
      change_bCap btn_no,bcap7

    case 1
      set_button btn_no,B_KIND,3DBUTTON_V              'ボタンタイプ設定
      set_button btn_no,B_KIND,3DBUTTON_V2             'ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs                'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,LIGHTGLAY            '背景色設定
      set_button btn_no,B_FGCOLOR,BLACK                '前面色設定
      var_index = loopcnt
      set_button btn_no,B_VALUE_NO,var_index
      change_bCap btn_no,bcap7

    case 2
      set_button btn_no,B_KIND,3DBUTTON_V              'ボタンタイプ設定
      set_button btn_no,B_KIND,3DBUTTON_V2             'ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs                'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,BLUE                 '背景色設定
```

```
      set_button btn_no,B_FGCOLOR,WHITE              ' 前面色設定
      var_index = loopcnt
      set_button btn_no,B_VALUE_NO,var_index
      change_bCap btn_no,bcap7

    case 3
      set_button btn_no,B_KIND,3DBUTTON_V            ' ボタンタイプ設定
      set_button btn_no,B_KIND,3DBUTTON_V2           ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs              'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GREEN              ' 背景色設定
      set_button btn_no,B_FGCOLOR,YELLOW             ' 前面色設定
      var_index = loopcnt
      set_button btn_no,B_VALUE_NO,var_index
      change_bCap btn_no,bcap7

    case 4
      set_button btn_no,B_KIND,3DBUTTON_V            ' ボタンタイプ設定
      set_button btn_no,B_KIND,3DBUTTON_V2           ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs              'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,CYAN               ' 背景色設定
      set_button btn_no,B_FGCOLOR,LIGHTMAGENTA       ' 前面色設定
      var_index = loopcnt
      set_button btn_no,B_VALUE_NO,var_index
      change_bCap btn_no,bcap7

    case 5
      set_button btn_no,B_KIND,3DBUTTON_V            ' ボタンタイプ設定
      set_button btn_no,B_KIND,3DBUTTON_V2           ' ボタンタイプ設定
      set_button btn_no,B_IO_NO,io_adrs              'I/O アドレス設定
      set_button btn_no,B_BGCOLOR,GLAY               ' 背景色設定
      set_button btn_no,B_FGCOLOR,LIGHTBLUE          ' 前面色設定
      var_index = loopcnt
      set_button btn_no,B_VALUE_NO,var_index
      change_bCap btn_no,bcap7
    end select
    loopcnt = loopcnt + 1
  wend

' 画面０へ戻る３ Ｄ ボタン作成
  io_adrs = 128
  minx = 510
  maxx = 600
  miny = 250
  maxy = 320
  btn_no = 92
  set_button (btn_no),(1),(minx)
  set_button (btn_no),(2),(miny)
  set_button (btn_no),(3),(maxx)
  set_button (btn_no),(4),(maxy)
  set_button (btn_no),(9),(enable)
  set_button (btn_no),(10),(visible)
  set_button (btn_no),(11),(var_type)
  set_button (btn_no),(12),(var_index)
```

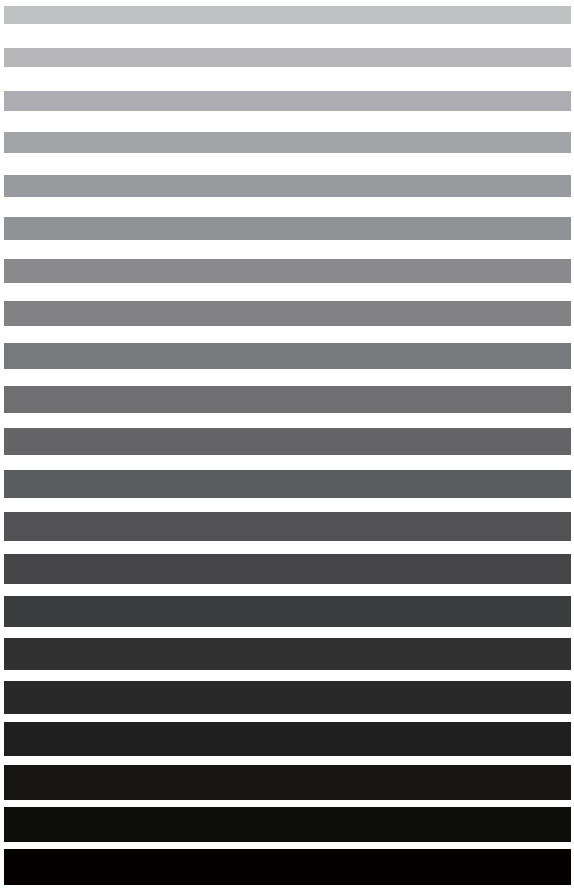
```
   set_button (btn_no),(14),(panel_no)
   set_button btn_no,B_KIND,3DBUTTON_IO          ' ボタンタイプ設定
   set_button btn_no,B_IO_NO,io_adrs             'I/O アドレス設定
   set_button btn_no,B_BGCOLOR,BLUE              ' 背景色設定
   set_button btn_no,B_FGCOLOR,WHITE             ' 前面色設定
   change_bCap btn_no,bcap6                      ' ボタン名設定

   disp_page panel_no                            ' 指定画面の表示
END
```

前述のプログラム例で作成された TP 簡易操作盤



# 第 14 章
# マルチタスク制御文



マルチタスク制御は、PAC の特徴の一つです。この章では、マルチタスク制御のためのコマンドを説明します。

# 第 14 章　マルチタスク制御文

## 14.1　タスク制御

### RUN（ステートメント）

**機　能**

別プログラムを並列起動します。

**書　式**

RUN < プログラム名 > [(< 引数 >[,< 引数 >…])][,<RUN オプション >]

**説　明**

現在実行中のプログラムから、< プログラム名 > に指定したプログラムを並列に起動します。自プログラムを RUN することはできません。

< 引数 >

< 引数 > には、値渡しのみ使用できます。参照渡しを指定した場合は自動的に値渡しになりますが、ローカル配列は使用することができません。

<RUN オプション >

<RUN オプション > には、次の 3 つのオプションが指定できます。

PRIORITY（もしくは P）

プログラムの優先度を指定します。省略するとデフォルト値 128 として処置します。

数値が小さいほど優先度が高くなります。設定範囲は、102 ～ 255 の間です。

**注意**

**特権タスクの優先順位は変更できません。**

CYCLE（もしくは C）

スイッチング周期（プログラムを繰り返し起動する場合のサイクル毎の間隔）を指定します。単位は、msec です。設定範囲は、1 ～ 2147483647 の間です。

引数を伴うプログラムは、CYCLE オプションを付けて起動できません。

RESULT（v2.6 以降のコントローラ）

RUN コマンド実行時に、内部エラーが発生してもプログラムを停止させません。RESULT=( 変数名 ) で指定した変数に内部エラーのエラーコードを格納します。

**関連項目**

CALL、GOSUB

**注意事項**

- SUSPEND 命令にて動作停止中のタスクを再度 RUN させる場合は、動作が停止してから RUN するようにしてください。動作停止完了前に RUN をかけ再動作させると、指令速度制限オーバなどのエラーになります。
- CYCLE オプションを使用する際は、命令実行中に瞬時停止を行ない再起動すると、一時停止中もサイクル同期経過が進むので注意してください。
- ティーチチェックモードでは RUN 命令は実行されません。

用　例

```
DEFINT Li1 = 1, Li2 = 2, Li3 = 3
RUN samp1, C = 1000          'samp1 を並列に（C = 1000）起動します。
RUN samp2 (Li1)              'samp2 に Li1 の引数を付けて並列に起動します。
RUN samp3 (Li1, Li2), PRIORITY = 129
                             'samp3 に Li1，Li2 の引数を付けて並列に（P = 129）
                             ' 起動します。
RUN samp4 (Li1, Li2), PRIORITY = 150
                             'samp4 に Li1，Li2 の引数を付けて並列に（P = 150）
                             ' 起動します。
RUN samp5 (Li1, Li2, Li3), PRIORITY = 120
                             'samp5 に Li1，Li2，Li3 の引数を付けて並列に（P = 120）
                             ' 起動します。
RUN samp6, RESULT = I1       'samp6 を起動させます。
                             'RUN 失敗時、変数 I1 にエラーコードを格納し次の行に
                             ' 移ります。
```

# KILL（ステートメント）

**機　能**

タスクを強制終了します。

**書　式**

KILL＜プログラム名＞

**説　明**

＜プログラム名＞で指定したタスク ( プログラム ) の実行を強制終了します。

自プログラムを KILL することはできません。自プログラムを KILL するとエラーが発生します。自プログラムを強制終了するには、STOP 命令を使ってください。

**関連項目**

SUSPEND、STOP

**注意事項**

アームセマフォを取得中のタスクを強制終了し、別タスクにてアームセマフォを取得する場合、「アームセマフォ取得失敗」エラーが発生する場合があります。その場合、別タスクがアームセマフォを取得する前にタイマなどを入れてください。

**用　例**

```
RUN samp1                    'samp1 を並列に起動します。
   .
   .
   .
KILL samp1                   'samp1 を終了します。
```

# SUSPEND（ステートメント）

**機　能**

タスクを一時停止します。

**書　式**

SUSPEND < プログラム名 >

**説　明**

指定したタスクを一時停止します。

自プログラムを SUSPEND することはできません。自プログラムを一時停止するには、HOLD 命令を使ってください。

**関連項目**

KILL、HOLD

**注意事項**

SUSPEND 命令にて動作停止中のタスクを再度 RUN させる場合は、動作が停止してから RUN するようにしてください。動作停止完了前に RUN をかけ再動作させると、指令速度制限オーバなどのエラーが発生します。

**用　例**

```
SUSPEND samp1                    'samp1 のタスクの実行を一時停止します。
```

# DEFEND（ステートメント）

**機　能**

タスクを保護します。

**書　式**

DEFEND {ON|OFF}

**説　明**

通常、プログラムタスクは、一定期間ごとに自分よりもプライオリティの高い、または同等のプログラムタスクに、実行権を明け渡します。

他のプログラムタスクに実行権を渡さずに、一気に処理したい場合には、DEFEND コマンドを使います。DEFEND ON 実行後は、DEFEND OFF が実行されるまで、実行権が確保されます。ただし、動作命令と、DELAY、WAIT、SET IO の時間指定の各コマンドを実行すると、実行権は他のプログラムタスクに移ります。

**注意**

- DEFEND ON で保護する区間はできるだけ短く設定してください。DEFEND ON 実行後に、無限ループ状態になると、他のプログラムタスクに実行権が移らなくなります。
- DEFEND ON で保護されていても、次の場合には、自動的に保護状態は解除されます。
  - END コマンドを実行した場合（呼び出したプログラムの最後にある END コマンドは除く）
  - KILL コマンドを実行した場合
  - ティーチングペンダントまたは I/O によって、ロボットコントローラが初期化された場合

**用　例**

```
DEFEND ON                    ’自タスクを保護します。
SET IO[100]                  ’保護されているため、以下の 3 命令は必ず連続して実
                             ’行されます。
SET IO[102]
SET IO[104]
DEFEND OFF                   ’自タスクの保護を解除します。
```

# STATUS（関数）

**機　能**

プログラムの状態を得ます。

**書　式**

STATUS(< プログラム名 >)

**説　明**

< プログラム名 > で指定したプログラムの状態が整数で格納されています。

| 値 | 状　態 | |
|---|---|---|
| 1 | running | 実行中 |
| 2 | stopping | 停止中 |
| 3 | suspend | 一時停止中 |
| 4 | delay | 遅延中 |
| 5 | pending | 待機中 |
| 6 | step stopped | ステップ停止中 |

**関連項目**

DELAY、HALT、HOLD、STOP

**注意事項**

- 自分自身の STATUS を取得することはできません。
- プログラムが実行中であることを示す状態には 1:running の他に 4:delay、5:pending も該当し、実行するコマンドの処理状況によって変化します。プログラムが実行中かどうかを判断する場合は用例のプログラムを参考にしてください。
- CALL コマンドで呼び出されているプログラムの状態を得るには、呼び出している方のプログラムのプログラム名を指定してください。例えば

```
PROGRUM PRO1
      :
  CALL SUB1
      :
```

というプログラムがあり、プログラム "SUB1" が呼び出されている最中のプログラム "SUB1" の状態を得るには "PRO1" を指定してください。"SUB1" を指定すると、正しい状態が得られません。

```
I0 = STATUS(PRO1)
```

用　例

```
DIM li1 As Integer
li1 = STATUS(samp1)          'samp1 のプログラムの状態を整数で li1 に代入します。

'====== プログラムの状態別処理 ======

DEFINT iX
iX = STATUS(PRO1)
SELECT CASE iX
  CASE 2                     ' 停止中
                             ' 停止中の処理
  CASE 3,6                   ' 一時停止中、ステップ停止中
                             ' 一時停止中、ステップ停止中の処理
  CASE ELSE                  ' 実行中、遅延中、待機中
                             ' 実行中の処理
END SELECT
```

# SUSPENDALL（ステートメント）【Ver.1.98 以降】

**機　能**

特権タスク以外の全てのプログラムを停止します。

**書　式**

SUSPENDALL

**説　明**

特権タスク以外の全タスクを停止させ、停止中のステップから動作させることのできる " コンティ停 " 状態となり、ロボット運転中出力を OFF します。

**関連項目**

SUSPEND、KILLALL、ROBOTSTOP、CONTINUERUN

**注意事項**

SUSPENDALL 命令を実行してプログラム停止後、RUN 命令にてプログラムの続きを実行可能です。

また CONTINUERUN 命令にてコンティニュー起動可能です。

動作停止中のタスクを再度 RUN させる場合は、動作が停止してから RUN させるようにしてください。動作完了前に RUN をかけ再動作させると、指令速度制限オーバなどのエラーが発生します。

SUSPENDALL 命令実行後（ロボット停止入力等、TP、外部からの停止関連入力全てと同様に）は約 0.5 秒間プログラム起動できません。

**用　例**

```
PROGRAM TSR1
------------
SUSPENDALL                    ’全タスク瞬時停止　コンティ停状態へ
------------
CONTINUERUN                   ’コンティニュー起動
END
```

# KILLALL（ステートメント）【Ver.1.98 以降】

**機　能**

特権タスク以外の全てのタスクを強制終了します。（プログラムリセットに相当）

**書　式**

KILLALL

**説　明**

ロボットは現在実行中の特権タスク以外の全タスクを強制終了させ、ロボット運転中出力を OFF します。

**関連項目**

KILL、SUSPENDALL、ROBOTSTOP

**注意事項**

停止させたタスクの続きを動作させることはできません。KILLALL 命令実行後（ロボット停止入力等、TP、外部からの停止関連入力全てと同様に）は約 0.5 秒間プログラム起動できません。

**用　例**

```
PROGRAM TSR1
------------
KILLALL                         ’全タスク停止　プログラムリセット状態
------------
END
```

# CONTINUERUN（ステートメント）【Ver.1.98 以降】

**機　能**

コンティニュー起動を行います。

**書　式**

CONTINUERUN

**説　明**

コンティ停状態にあるタスクを引き続き全てのタスクを停止されたステップの途中から再起動します。

**関連項目**

SUSPENDALL、ROBOTSTOP

**注意事項**

コンティニュースタート許可状態のとき、特権タスクからのみ実行可能です。

**用　例**

```
PROGRAM TSR1
------------
SUSPENDALL                    ’全タスク瞬時停止　コンティ停状態へ
------------
CONTINUERUN                   ’コンティニュー起動
------------
END
```

# ROBOTSTOP（ステートメント）【Ver.1.98 以降】

**機　能**

ロボット停止を行います。

**書　式**

ROBOTSTOP

**説　明**

特権タスク以外の全タスクを停止させ、停止中のステップから動作させることのできる " コンティ停 " 状態となり、ロボット運転中出力を OFF します。またモータ電源を OFF させます。

**補足説明**

"SUSPENDALL" コマンドとの相違点はモータ電源が OFF され、それに伴った外部出力が変化することです。ROBOTSTOP 命令実行後（ロボット停止入力等、TP、外部からの停止関連入力全てと同様に）は約 0.5 秒間プログラム起動できません。

またこのコマンドは、ティーチングペンダントの設定で、[F1 プログラム ] ー [F6 補助機能 ] ー [F7 コンティ設定 ] の操作で表示される画面のパラメータ 0: コンティニュー（0：無効　1：コンティニュー）が無効である場合、プログラムは、コンティニュー停止せず停止状態となります。

**関連項目**

SUSPENDALL、CONTINUERUN

**用　例**

```
PROGRAM TSR1
------------
ROBOTSTOP                    ’ロボット停止
------------
END
```

# TAKEARMSTATE（関数）【RC7_Ver.2.2 以降】

**機　能**

アームグループ制御権の取得状態を返します。
0：未取得、 1：取得済み

**書　式**

TAKEARMSTATE (<アームグループ No>)

**説　明**

現在の指定アームグループ No 制御権の取得状態を返します。
＜アームグループ No ＞
状態を取得したいアームグループ No を指定します。

**用　例**

```
PROGRAM PRO1
------------
I1 = TAKEARMSTATE (1)          'アームグループ 1 の取得状態を取得
------------
END
```

# LOCKSTATE（関数）【RC7_Ver.2.2 以降】

**機　能**

マシンロックの状態を取得します。
0：OFF、 1：ON

**書　式**

LOCKSTATE

**説　明**

現在のマシンロックの状態を取得します。

**用　例**

```
PROGRAM PRO1
------------
I1 = LOCKSTATE                 'マシンロックの状態を取得
------------
END
```

# DEADMANSTATE（関数）【RC7_Ver.2.2 以降】

**機　能**

デッドマンスイッチの状態を取得します。

0：OFF、 1：ON

**書　式**

DEADMANSTATE

**説　明**

現在のデッドマンスイッチの状態を取得します。

**用　例**

```
PROGRAM PRO1
------------
I1 = DEADMANSTATE              'デッドマン SW の状態を取得
------------
END
```

# SEMIDSTATE（関数）【RC7_Ver.2.2 以降】

**機　能**

指定したセマフォ ID が有効か無効かを返します。

**書　式**

SEMIDSTATE(< セマフォ ID>)

**説　明**

指定したセマフォ ID の「有効 (1) / 無効 (0)」を返します。
CREATESEM 実行時セマフォ ID は有効になり、DELETESEM 実行時セマフォ ID は無効になります。

**関連項目**

CREATESEM、DELETESEM

**用　例**

PRO1、PRO2 で使用するセマフォを TSR1 で作成する。
SEMIDSTATE を使ってセマフォ ID が有効でない時のみセマフォを作成するようにする。

```
'!TITLE " セマフォ作成 "
PROGRAM TSR1
  FOLDER DEFINT SEMID                  'SEMID が有効 (1) でない時のみセマフォ作成
  IF SEMIDSTATE (SEMID) <> 1
    SEMID = CREATESEM (0)
  ENDIF
END

'!TITLE "I0 への書込みをセマフォで制御 "
PROGRAM PRO1
  EXTERN DEFINT SEMID
  TAKESEM SEMID
    I0=1
  GIVESEM SEMID
END

'!TITLE "I0 への書込みをセマフォで制御 "
PROGRAM PRO2
  EXTERN DEFINT SEMID
  TAKESEM SEMID
    I0=1
  GIVESEM SEMID
END
```

## 14.2　セマフォ

セマフォは、複数のタスクを同期させたり ( 同期制御)、同時には動かないようにしたり ( 排他制御 ) する場合に、タスク間の通信 ( 合図の連絡 ) に利用します。

セマフォを利用するには、まず CREATESEM コマンドでセマフォを生成し、セマフォ ID を取得します。セマフォ ID によって、複数のセマフォの中から特定のセマフォを指定できるようになります。

同期または排他制御を行う場合、他のタスクの指令を待つ側のタスクは TAKESEM コマンドを実行して、指令を出す側のタスクが GIVESEM コマンドを実行するのを待ちます。指令を出す側のタスクは、準備ができたら GIVESEM コマンドを実行して、セマフォを待っているタスクに対して処理の実行を許可します。

一つの GIVESEM コマンドは、一つのセマフォ待ちタスクに対してのみ有効です。

複数のタスクが、同じセマフォ ID のセマフォを待っている場合に、どのタスクから実行するかは、先着順またはプライオリティ順の 2 つのキューイング（実行待ち）方式から選択できます。

# CREATESEM（関数）

**機　能**

セマフォを生成します。

**書　式**

CREATESEM(< 数式 >)

**説　明**

セマフォを生成し、セマフォ ID を取得します。

セマフォ ID がないと、他のセマフォ関連のコマンドは使用できませんので、セマフォの使用に先立ってこの CREATESEM コマンドを実行しなくてはなりません。

引数でタスクのキューイング（実行待ち）方式を指定します。複数のタスクが同じセマフォを待っている場合の実行順を決めます。

キューイング方式は次の 2 種類があります。

| 引 数 | |
|---|---|
| 0 | 先着順 |
| 0 以外 | タスクのプライオリティ順 |

先着順は、TAKESEM コマンドを実行した順です。

タスクのプライオリティは、RUN コマンドのオプション (PRIORITY) で指定されたものです。

CREATESEM を実行すると、その時点でセマフォが存在する状態になりますので、GIVESEM を実行しなくても、TAKESEM を実行できます。

セマフォは 32 個生成することができます。

**関連項目**

DELETESEM、FLUSHSEM、GIVESEM、TAKESEM

注意事項

CREATESEM 命令使用上の注意：

- 間違った使用方法により発生する現象
  CREATESEM 命令によりすでに生成されたセマフォ ID を持つプログラムが動作中・一時停止待機中の場合に
  - SEMCREATE によりセマフォ ID 格納変数を書き換える、
    あるいは
  - その変数の書き換えをティーチングペンダントなどから故意に行うとセマフォID を持つプログラムがセマフォを獲得できずに待機中のままになります。
- 例
  PRO1 を起動し PRO2 の動作中に STOP キーにより瞬時停止を行います。その後 PRO1 を再起動すると、i1 に格納されたセマフォ ID が変更されるため、PRO3 は無限にセマフォを待ち続けます。

```
PROGRAM PRO1
 i1=CREATESEM(0)
 RUN PRO2
 RUN PRO3
END

PROGRAM PRO2
 TAKESEM i1
 .
 .
 .
 GIVESEM i1
 .
 .
 .
END

PROGRAM PRO3
 TAKESEM i1
 .
 .
 .
 GIVESEM i1
END
```

- 発生時の対処方法
  この状態から抜け出すには、無限待ちになっているプログラム（例では PRO3）を停止してから、呼び出し元のプログラム（例では PRO1）を起動してください。
- 正しく使用するために守っていただくこと
  待ち状態にあるセマフォの ID を書き換えないように注意してください。
  具体的には、以下の 2 点をお守りください。
  - 同一変数番号を使用するセマフォは明示的に削除されない限り、生成を一度だけにする。
  - セマフォ ID を管理する変数は他の用途で利用しない。

用　例

```
DEFINT Li1, Li2, Li3 = 1
Li1 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li1 にセマフォ ID を得ます。
Li2 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li2 にセマフォ ID を得ます。
TAKESEM Li1                   'Li1 で指定されるセマフォを取得します。
TALESEM Li2, 100              'Li2 で指定されるセマフォを取得します。ただし、100ms
                              ' でタイムアウトします。
RUN samp1
GIVESEM Li1                   'Li1 で指定されるセマフォを持つ一つのタスクを待ち
                              ' 状態から解放します。
FLUSHSEM Li2                  'Li2 で指定されるセマフォを持つすべてのタスクを待
                              ' ち状態から解放します。
DELETESEM Li1                 'Li1 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
DELETESEM Li2                 'Li2 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
```

# DELETESEM（ステートメント）

**機　能**

セマフォを削除します。

**書　式**

DELETESEM＜セマフォ ID>

**説　明**

＜セマフォ ID> で指定したセマフォ ID を持つセマフォを削除します。

**関連項目**

CREATESEM、FLUSHSEM、GIVESEM、TAKESEM

**用　例**

```
DEFINT Li1, Li2, Li3 = 1
Li1 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li1 にセマフォ ID を得ます。
Li2 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li2 にセマフォ ID を得ます。
TAKESEM Li1                   'Li1 で指定されるセマフォを取得します。
TALESEM Li2, 100              'Li2 で指定されるセマフォを取得します。ただし、100ms
                              ' でタイムアウトします。
RUN samp1
GIVESEM Li1                   'Li1 で指定されるセマフォを持つ一つのタスクを待ち
                              ' 状態から解放します。
FLUSHSEM Li2                  'Li2 で指定されるセマフォを持つすべてのタスクを待
                              ' ち状態から解放します。
DELETESEM Li1                 'Li1 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
DELETESEM Li2                 'Li2 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
```

# FLUSHSEM（ステートメント）

**機　能**

セマフォ待ちタスクを解放します。

**書　式**

FLUSHSEM＜セマフォ ID>

**説　明**

＜セマフォ ID> で指定したセマフォを待っているすべてのタスクに対し、処理の再開を許可します。

**関連項目**

CREATESEM、DELETESEM、GIVESEM、TAKESEM

**用　例**

```
DEFINT Li1, Li2, Li3 = 1
Li1 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li1 にセマフォ ID を得ます。
Li2 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li2 にセマフォ ID を得ます。
TAKESEM Li1                   'Li1 で指定されるセマフォを取得します。
TALESEM Li2, 100              'Li2 で指定されるセマフォを取得します。ただし、100ms
                              ' でタイムアウトします。
RUN samp1
GIVESEM Li1                   'Li1 で指定されるセマフォを持つ一つのタスクを待ち
                              ' 状態から解放します。
FLUSHSEM Li2                  'Li2 で指定されるセマフォを持つすべてのタスクを待
                              ' ち状態から解放します。
DELETESEM Li1                 'Li1 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
DELETESEM Li2                 'Li2 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
```

# GIVESEM（ステートメント）

**機　能**

セマフォ待ちタスクを解放します。

**書　式**

GIVESEM < セマフォ ID>

**説　明**

< セマフォ ID> で指定したセマフォを解放します。

解放時、< セマフォ ID> で指定したセマフォを待っている一つのタスクがあれば、処理の再開を許可します。複数のセマフォ待ちタスクが存在する場合には、セマフォ生成時に CREATESEM コマンドで指定したキューイング方式で、実行順位が決められます。

**関連項目**

CREATESEM、DELETESEM、FLUSHSEM、TAKESEM

**用　例**

```
DEFINT Li1, Li2, Li3 = 1
Li1 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li1 にセマフォ ID を得ます。
Li2 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li2 にセマフォ ID を得ます。
TAKESEM Li1                   'Li1 で指定されるセマフォを取得します。
TALESEM Li2, 100              'Li2 で指定されるセマフォを取得します。ただし、100ms
                              ' でタイムアウトします。
RUN samp1
GIVESEM Li1                   'Li1 で指定されるセマフォを持つ一つのタスクを待ち
                              ' 状態から解放します。
FLUSHSEM Li2                  'Li2 で指定されるセマフォを持つすべてのタスクを待
                              ' ち状態から解放します。
DELETESEM Li1                 'Li1 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
DELETESEM Li2                 'Li2 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
```

# TAKESEM（ステートメント）

**機　能**

指定したセマフォ ID を持つセマフォを取得します。

**書　式**

TAKESEM < セマフォ ID>[,< タイムアウト時間 >]

**説　明**

< セマフォ ID> で指定したセマフォを取得します。

取得時に他のタスクがセマフォを取得している場合は、そのセマフォが解放されるのを待ちます。

オプションの < タイムアウト時間 > で、待ち時間の限度を ms の単位で指定できます。指定した時間の間にセマフォを取得できなかったときは、エラーになります。

**関連項目**

CREATESEM、DELETESEM、FLUSHSEM、GIVESEM

**注意事項**

タイムアウト時間を使用する際には、命令実行中に瞬時停止を行ない再起動すると、一時停止中も待ち時間は経過しますので注意してください。

**用　例**

```
DEFINT Li1, Li2, Li3 = 1
Li1 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li1 にセマフォ ID を得ます。
Li2 = CREATESEM(Li3)          'Li3 で指定したキューイング方式でセマフォを生成し、
                              'Li2 にセマフォ ID を得ます。
TAKESEM Li1                   'Li1 で指定されるセマフォを取得します。
TALESEM Li2, 100              'Li2 で指定されるセマフォを取得します。ただし、100ms
                              ' でタイムアウトします。
RUN samp1
GIVESEM Li1                   'Li1 で指定されるセマフォを持つ一つのタスクを待ち
                              ' 状態から解放します。
FLUSHSEM Li2                  'Li2 で指定されるセマフォを持つすべてのタスクを待
                              ' ち状態から解放します。
DELETESEM Li1                 'Li1 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
DELETESEM Li2                 'Li2 で指定されるセマフォ ID を持つセマフォを削除し
                              ' ます。
```

## 14.3　アームセマフォ

# TAKEARM（ステートメント）

機　能

アームグループを取得します。アームグループ NO を省略した場合、アームグループ 0（ロボットのみ）の制御権を取得します。

取得時、内部速度・加速度・減速度を 100 に設定します。

取得するアームグループがロボット軸を含む場合は、ツール座標、ワーク座標を原点に戻します。

書　式

TAKEARM［＜アームグループ NO ＞］［＜ KEEP ＝初期化設定値＞］

説　明

制御権をすでに持っているタスク、あるいはそのタスクから呼ばれたサブルーチンの中で TAKEARM コマンドを実行した場合には、そのまま処理を続行します。

＜アームグループ NO ＞ [V1.5 以降 ]

アームグループを取得します。アームグループ NO を省略した場合、アームグループ 0（ロボットのみ）の制御権を取得します。

<KEEP = 初期化設定値 > [V1.4 以降 ]

初期化設定値　0：ツール座標、ワーク座標を原点に戻し、内部速度、加速度、減速度を 100 に設定します。

初期化設定値

1：ツール座標、ワーク座標、内部速度、加速度、減速度を現在設定されている値のままにします。

<KEEP = 初期化設定値＞省略時は KEEP = 0（ツール座標、ワーク座標を原点に戻し、内部速度、加速度、減速度を 100）に設定します。

（1）アームグループ 0（ロボットのみ）の場合の説明

ロボット制御権を取得していないタスクが、下表のロボット動作命令を実行しようとした場合は、エラーとなります。これらの動作命令を実行するプログラムでは、必ず TAKEARM コマンドで、制御権を取得してください。

制御権を必要とするロボット動作命令

| 種　類 | コマンド |
|---|---|
| 宣言文 | HOME, TOOL, WORK |
| ロボット制御文 | APPROACH, DEPART, DRAW, DRIVE, DRIVEA, GOHOME, MOVE, ROTATEH, ROTATE, SPEED, JSPEED, ACCEL, JACCEL, DECEL, JDECEL, CHANGETOOL, CHANGEWORK, LETENV,<br>最適可搬質量設定ライブラリ、アーム動作ライブラリ |

**注意**

- 次の場合には、自動的にロボット制御権を解放します。
  - END コマンドを実行した場合
    （呼び出したプログラムの最後にある END コマンドは除く）
  - KILL コマンドを実行した場合
  - HALT コマンドを実行した場合
  - STOP コマンドを実行した場合
  - ティーチングペンダントまたは I/O によって、ロボットコントローラが初期化された場合
- TAKEARM コマンドを実行すると、次の処理が自動的に実行されます。
  - TOOL 定義を TOOL0 に初期化、ワーク座標をロボットのベース座標に設定
  - 内部速度、内部加速度、内部減速度を 100 に設定
- プログラム一時停止中は解放しません。

（2）付加軸関連の説明

アームグループを取得していないタスクが、下表の動作命令を実行しようとした場合、エラーとなります。これらの動作命令を実行する場合は、必ず TAKEARM コマンドで制御権を取得してください。

アームグループを必要とするコマンド

| コマンド | 必要なアームグループ |
|---|---|
| HOME、TOOL、WORK、APPROACH、DEPART、DRAW、GOHOME、MOVE、ROTATEH、ROTATE、CHANGETOOL、CHANGEWORK、DRIVE、DRIVEA、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL、INTERRUPT、LETENV、最適可搬質量設定ライブラリ、アーム動作ライブラリ | ロボット軸を含むアームグループが必要 |
| DRIVE、DRIVEA、SPEED、JSPEED、ACCEL、JACCEL、DECEL、JDECEL、INTERRUPT、LETENV、POSCLR | アームグループが必要（ロボット軸を含まなくてもよい） |

**注意**

- DRIVE、DRIVEA は、動作させる軸を含むアームグループが必要です。
  例　DRIVE(7、10)
  7 軸を動かすには、7 軸を含むアームグループの取得が必要。
- MOVE 命令の EX(EXA) オプションを使う場合は、ロボット軸と EX(EXA) で動かす付加軸を含むアームグループの取得が必要です。
  例　MOVE P,P0 EX((7,10))
  この場合、ロボット軸と 7 軸を含むアームグループの取得が必要です。
- 速度設定のコマンドは、現在取得しているアームグループの軸のみ変更します。
- LETENV 命令は、変更するパラメータが影響する軸を含むアームグループの取得が必要です。

**関連項目**

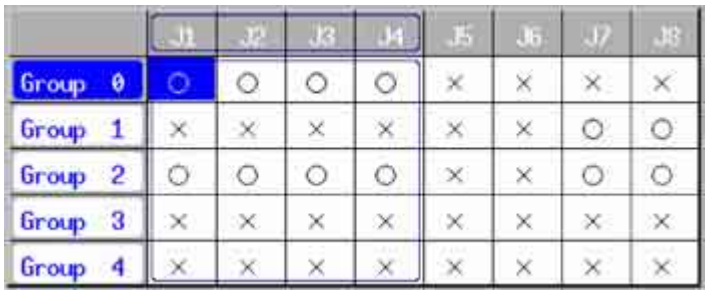
GIVEARM、TAKEVIS、GIVEVIS

**注意事項**

- 一つのプログラムで 2 つのグループを取得することはできません。
  ただし同じアームグループの場合は再取得することができます。

```
例 :  TAKEARM 0
      MOVE P, P0
      TAKEARM 0                ' グループ 0 取得後でも、グループ 0 は再取得可能。
      TAKEARM 1                ' グループ 0 取得後に、グループ 1 を取得しているので
                               ' エラー
```

- TAKEARM の KEEP オプションを 1 にしてグループを取得した場合は、速度は初期化されず、以前のアームグループの速度で動作します。
  例 : 次のようなアームグループが設定されており、下記の 3 つのプログラムがある場合。



| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 0 | ○ | ○ | ○ | ○ | × | × | × | × |
| Group 1 | × | × | × | × | × | × | ○ | ○ |
| Group 2 | ○ | ○ | ○ | ○ | × | × | ○ | ○ |
| Group 3 | × | × | × | × | × | × | × | × |
| Group 4 | × | × | × | × | × | × | × | × |

```
PROGRAM PRO1          PROGRAM PRO2          PROGRAM PRO3
TAKEARM 1             TAKEARM 2             TAKEARM 1 keep=1
SPEED 10              SPEED 20              DRIVE(7,10)
DRIVE(7,10)           DRIVE(7,10)           END
END                   END
```

まず PRO1 を動作させます。7 軸は速度 10 で動きます。
次に PRO2 を動作させます。7 軸は速度 20 で動きます。
最後に PRO3 を動作させた場合、KEEP 初期化設定値が 1 であるので、7 軸は以前のアームグループ 1 の速度 10 で動きます。

**用　例**

**例 1 :**

```
PROGRAM PRO1                   '
 TAKEARM                       ' ロボット動作を行うプログラムの先頭で TAKEARM を
                               ' 実行します。
 MOVE P, P1
 MOVE P, P2
 GIVEARM 1                     ' 終了時に GIVEARM でロボット制御権を解放します。直
                               ' 後の END で自動的に解放するので、必ずしも実行する
                               ' 必要はありません。
END                            '
```

**例 2 :**

```
PROGRAM PRO2                   '
 MOVE P, P3                    '× : TAKEARM を実行せずに MOVE 命令を実行すると
                               ' エラーになります。
END                            '
```

例３：

```
PROGRAM PRO3                    '
 TAKEARM                        '
 MOVE P, P4                     '
 CALL SUB1                      '
END                             '
PROGRAM SUB1                    '
 MOVE P, P5                     'PRO3 で TAKEARM 実行済みなので、エラーになりません。
END                             '
```

例４：

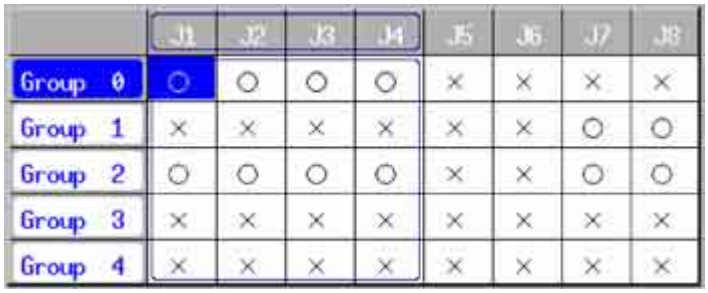
```
PROGRAM PRO4                    '
 TAKEARM                        '
 SPEED 50                       '
 CHANGEWORK 3                   '
 CHANGETOOL 1                   '
 MOVE P, P5                     '
 CALL PRO5                      '
END                             '
PROGRAM PRO5                    '
 TAKEARM                        'PRO4 のサブルーチンとして呼ばれているので、PRO4
                                ' と重複して TAKEARM してもエラーになりません。
 MOVE P, P6                     ' ただし、ツール座標、ワーク座標は 0、内部速度は 100
                                ' になります。
END                             '
```

例５：

```
PROGRAM PRO6                    '
 TAKEARM                        '
 RUN PRO7                       '
 MOVE P, P7                     '
END                             '
PROGRAM PRO7                    '
 TAKEARM                        '×：すでにロボット制御権を持つ PRO6 とは別のタス
                                ' クとして起動された PRO7 で、ロボット実行権を取得
                                ' しようとすると、エラーになります。
 MOVE P, P7                     '
END                             '
```

**例 6：付加軸有りの例**

次のようなアームグループが設定されている場合のプログラム例。(4 軸ロボットの例)



| | J1 | J2 | J3 | J4 | J5 | J6 | J7 | J8 |
|---|---|---|---|---|---|---|---|---|
| Group 0 | ○ | ○ | ○ | ○ | × | × | × | × |
| Group 1 | × | × | × | × | × | × | ○ | ○ |
| Group 2 | ○ | ○ | ○ | ○ | × | × | ○ | ○ |
| Group 3 | × | × | × | × | × | × | × | × |
| Group 4 | × | × | × | × | × | × | × | × |

```
PROGRAM PRO1
TAKEARM 1                    ' アームグループ 1 を取得。(7 軸を含むアームグループ)
DRIVEA(7,100)                '7 軸を 100 度の位置へ動かします。
END

PROGRAM PRO2
TAKEARM 2                    ' アームグループ 2 を取得。(ロボット軸、7 軸を含むアーム
                             ' グループ)
MOVE P,P0 EX((7,10))         ' ロボット軸と付加軸を同時に動作させます。
DRIVEA(7,100)                '7 軸を 100 度の位置へ動かします。
END
```

# GIVEARM（ステートメント）

**機　能**

現在取得しているアームグループを解放します。

アームグループ０（ロボットのみ）の場合、ロボット制御権を開放します。

**書　式**

GIVEARM

**説　明**

現在取得しているアームグループを解放します。これにより別のタスクがアームグループの制御権を取得できるようになります。

アームグループを取得していないタスクが GIVEARM コマンドを実行しようとした場合は、エラーとなります。

**注意**

**次の場合には、自動的にロボット制御権が解放されます。したがって、GIVEARM コマンドを省略してもかまわない場合があります。**

- **END コマンドを実行した場合（呼び出したプログラムの最後にある END コマンドは除く）**
- **KILL コマンドを実行した場合**
- **ティーチングペンダントまたは I/O によって、ロボットコントローラが初期化された場合**

**関連項目**

TAKEARM、TAKEVIS、GIVEVIS

**注意事項**

GIVEARM 実行時は、現在取得しているアームグループの軸が完全に停止するのを待ちます。したがって、GIVEARM の前にパス動作指定をしてもパス動作しません。

**用　例**

**例１：**

```
TAKEARM                 ’ロボット動作を行うプログラムの先頭で TAKEARM を実行します。
MOVE P, lp1             ’
MOVE P, lp2             ’
GIVEARM                 ’1 終了時に GIVEARM でロボット制御権を解放します。
                        ’直後の END で自動的に解放するので、必ずしも実行する
                        ’必要はありません。
```

**例２（付加軸有りの例）：**

```
PROGRAM PRO1
TAKEARM 1               ’アームグループ１を取得。
GIVEARM                 ’現在取得しているアームグループ１を解放します。
END

PROGRAM PRO2
TAKEARM 2               ’アームグループ２を取得。
END                     ’プログラム停止時、現在取得しているアームグループ
                        ’２を解放します。
```

# TAKEVIS（ステートメント）

**機　能**

視覚処理権を取得します。

**書　式**

TAKEVIS

**説　明**

ロボットコントローラにオプションで内蔵される、μVISION ボードでの視覚処理権を取得します。μVISION ボードが内蔵されていない状態で、TAKEVIS コマンドを実行するとエラーになります。

視覚処理権を別のタスクが取得しており、処理権を取得できなかった場合には、エラーとなります。

視覚処理権を取得していないタスクが、視覚命令を実行しようとした場合は、エラーとなります。視覚命令を実行するプログラムでは、視覚命令に先立って、TAKEVIS コマンドで視覚処理権を取得してください。

視覚処理権を持っているタスク、あるいはそのタスクから呼ばれたサブルーチンの中で TAKEVIS コマンドを実行した場合には、そのまま処理を続行します。

**注意**

次の場合には、自動的に視覚処理権が解放されます。

- END コマンドを実行した場合（呼び出したプログラムの最後にある END コマンドは除く）
- KILL コマンドを実行した場合
- ティーチングペンダントまたは I/O によって、ロボットコントローラが初期化された場合

**関連項目**

TAKEARM、GIVEARM、GIVEVIS

**用　例**

```
DEFINT Li1, Li2
TAKEVIS                                  ' 視覚処理権を取得します。
VISSCREEN 1, 0, 1
VISCLS
FOR Li1 = 0 TO 255
 Li2 = VISREFTABLE(1, Li1)               ' テーブル１番のデータを取得します。
 VISLOC 10, 10                           ' 表示位置の設定
 VISDEFCHAR 1, 1, 2                      ' 表示文字の設定
 VISPRINT "データ" ; Li1 ; "=" ; Li2     ' 表示
NEXT Li1
GIVEVIS                                  ' 視覚処理権の解放
```

# GIVEVIS（ステートメント）

**機　能**

視覚処理権を解放します。

**書　式**

GIVEVIS

**説　明**

ロボットコントローラにオプションで内蔵される、μVISION ボードでの視覚処理権を解放します。これにより、別のタスクが視覚処理権を取得できるようになります。μVISION ボードが内蔵されていない状態で、GIVEVIS コマンドを実行するとエラーになります。

視覚処理権を、別のタスクがすでに取得しているにもかかわらず、GIVEVIS コマンドを実行しようとした場合には、エラーとなります。

**注意**

**次の場合には、自動的にロボット制御権が解放されます。したがって、GIVEVIS コマンドを省略してもかまわない場合があります。**

- **END コマンドを実行した場合（呼び出したプログラムの最後にある END コマンドは除く）**
- **KILL コマンドを実行した場合**
- **ティーチングペンダントまたは I/O によって、ロボットコントローラが初期化された場合**

**関連項目**

TAKEARM、GIVEARM、TAKEVIS

**用　例**

```
DEFINT Li1, Li2
TAKEVIS                                  ' 視覚処理権を取得します。
VISSCREEN 1, 0, 1
VISCLS
FOR Li1 = 0 TO 255
 Li2 = VISREFTABLE(1, Li1)               ' テーブル１番のデータを取得します。
 VISLOC 10, 10                           ' 表示位置の設定
 VISDEFCHAR 1, 1, 2                      ' 表示文字の設定
 VISPRINT "データ " ; Li1 ; "=" ; Li2    ' 表示
NEXT Li1
GIVEVIS                                  ' 視覚処理権の解放
```

## 14.4　特権タスク

# INIT【Ver.1.7 以降】

**機　能**

特権タスクパラメータの状態により、モータ ON、CAL、スピード設定を行います。

**書　式**

INIT　[, <INIT オプション >]

**説　明**

- 特権タスク使用／未使用が [ 未使用 ] になっている場合は何もしません。
- 特権タスク使用／未使用が [ 使用 ] になっている場合は以下の処理を実行します。

  **INIT 命令実行モードが［(モータ ON+CAL)：非実行］の場合**

  － INIT 起動スピードが 10 であれば、コントローラの外部スピードを 10 にします。

  － INIT 起動スピードが 100 であれば、コントローラの外部スピードを 100 にします。

  **INIT 命令実行モードが［(モータ ON+CAL)：実行］の場合**

  － INIT 起動スピードが 10 であれば、外部スピードを 10、モータ ON および CAL を実行します。

  － INIT 起動スピードが 100 であれば、外部スピードを 100、モータ ON
- <INIT オプション > で「RESULT」を記述すると、INIT コマンド実行時に内部エラーが発生しても、特権タスクを停止させません。RESULT=（変数名）で発生したエラーのエラーコードを指定した変数に格納します。（v 2.6 以降のコントローラに限る。）

**注意事項**

- 無限ループでINIT命令のみを行うようなプログラムとロボット動作プログラムを同時に走らせないようにしてください。
- INIT 実行中は動作中プログラムの表示が待機中になることがありますので、注意してください。
- 複数の特権タスクから同時に INIT 命令を実行しないでください。

**用　例**

```
'!TITLE " 初期化 "
PROGRAM TSR1
  INIT, RESULT = I1          ' モータ ON、CAL、スピード設定を行います。
                             'INIT 失敗時、変数 I1 にエラーコードを格納し、次の
                             ' 行に移ります。
END
```

# SETOCCUPATIONTIME（ステートメント）【RC7_Ver.2.0 以降】

**機　能**

特権タスクの占有時間を変更します。

**書　式**

SETOCCUPATIONTIME < 占有時間 >

**説　明**

- 特権タスク使用 / 未使用が [ 未使用 ] になっている場合は何もしません。
- 特権タスク使用 / 未使用が [ 使用 ] になっている場合は以下の処理を実行します。
  < 占有時間 > には、0,2,4,6,8 のいずれかが設定可能です。それぞれの設定値は、
  0: 0ms/8ms
  2: 2ms/8ms
  4: 4ms/8ms
  6: 6ms/8ms
  8: 8ms/8ms
  を意味します。設定は特権タスク動作中でも可能であり、即反映されます。

**注意事項**

0m/8ms に設定した場合、PAC プログラムの空いている時間を使って特権タスクは動作します。空いている時間がない場合特権タスクは動作しません。

8ms/8ms に設定した場合、特権タスク動作中に、PAC プログラムは動作しません。

PAC プログラムと特権タスクを同時に走らすには、特権タスクに、DELAY またはセマフォ待ちなどの処理を入れる必要があります。

**用　例**

下記のような 3 つのプログラムがある場合、通常は、PRO1 と PRO2 の実行を優先させたいが、ロボット動作の時のみは、PRO2 の実行を多少犠牲にしても特権タスクの動作を優先させたい場合などに使用します。

```
PROGRAM TSR1                    ’特権タスクは何らかの動作範囲チェックプログラム
  -------------
  -------------
END
PROGRAM PRO1
  TAKEARM
  -------------
  SETOCCUPATIONTIME 4           ’特権タスクの占有時間を 4msec/8msec にする
  MOVE P,P0                     ’ロボット動作
  SETOCCUPATIONTIME 2           ’特権タスクの占有時間をデフォルトの 2msec/8msec に戻す
  -------------
END
PROGRAM PRO2
  DO WHILE 1
  -------------
  LOOP
END
```

# INITWAITERR（ステートメント）【RC7_Ver.2.2 以降】

**機　能**

エラー検知を初期化します。（特権タスク専用命令）

**書　式**

INITWAITERR

**説　明**

エラー検知命令（WAITERROR）を使用してエラー検知を行う場合にエラー検知を初期化します。

この命令を実行するとそれまでに検知しているエラーをクリアします。

エラー検知命令を使用する場合にこの命令にてエラー検知を初期化しておかないと、正常にエラー検知できなくなる場合があります。

**関連項目**

WAITERROR、CURERRSTATUS

**用　例**

```
PROGRAM TSR1
-------------
INITWAITERR                    ’エラー検知の初期化
-------------
  .
  .
  .
WHILE 1                        ’繰り返し開始
  -------------
  I1 = WAITERROR               ’エラー検知
  I2 = CURERRSTATUS            ’現在のエラー発生状態取得
  -------------
WEND                           ’繰り返し終了
-------------
END
```

# WAITERROR（関数）【RC7_Ver.2.2 以降】

**機　能**

エラー発生を検知します。（特権タスク専用命令）

**書　式**

WAITERROR

**説　明**

検知したエラーのエラーコードを返します。

この命令を実行するとコントローラがエラー発生するまで待ち、エラーが発生した場合に発生したエラーのエラーコードを返します。

検知するエラーは発生した順に検知します、エラー検知開始時はエラー検知初期化命令（INITWAITERR）を使用してエラー検知しておかないと、正常にエラー検知できなくなる場合があります。

連続してエラーが発せした場合、検知できるエラーは最後に発生したエラーから 32 個前までのエラーを古い順に検知します。

**関連項目**

INITWAITERR、CURERRSTATUS

**用　例**

```
PROGRAM TSR1
-------------
INITWAITERR                    ’エラー検知の初期化
-------------
  .
  .
  .
WHILE 1                        ’繰り返し開始
  -------------
  I1 = WAITERROR               ’エラー検知
  I2 = CURERRSTATUS            ’現在のエラー発生状態取得
  -------------
WEND                           ’繰り返し終了
-------------
END
```

# CURERRSTATUS（関数）【RC7_Ver.2.2 以降】

**機　能**

エラー発生状態を返します。（特権タスク専用命令）

**書　式**

CURERRSTATUS

**説　明**

現在のエラー発生状態、「エラー発生中 (1) / エラーなし (0)」を返します。
この命令は、エラー検知を行う場合に現在エラーが発生しているかを判断する場合に使用します。

**関連項目**

INITWAITERR、WAITERROR

**用　例**

```
PROGRAM TSR1
-------------
INITWAITERR                    ’エラー検知の初期化
-------------
  .
  .
  .
WHILE 1                        ’繰り返し開始
  -------------
  I1 = WAITERROR               ’エラー検知
  I2 = CURERRSTATUS            ’現在のエラー発生状態取得
  -------------
WEND                           ’繰り返し終了
-------------
END
```

# 第 15 章
# 関数

この章では、PAC に用意されている各種の関数について説明します。

# 第 15 章　関数

## 15.1　算術関数

### ABS（関数）【SLIM 準拠】

**機　能**

数式の値の絶対値を得ます。

**書　式**

ABS(< 数式 >)

**説　明**

< 数式 > で指定した値の絶対値を得ます。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**補足説明**

**数値の絶対値とは、その数値から符号を取り除いたものです。たとえば、ABS(-1) と ABS(1) は、どちらも 1 を返します。**

**関連項目**

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4
lf1 = ABS(-2)                ’-2 の絶対値を f1 に代入します。
lf2 = ABS(lf3 / lf4)         ’(lf3 / lf4) の絶対値を lf2 に代入します。
```

# EXP（関数）【SLIM 準拠】

**機　能**

自然対数を基数とする指数関数を得ます。

**書　式**

EXP(< 数式 >)

**説　明**

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**補足説明**

- 引数 number の値が 709.782712893 を超えると、オーバフローエラーが発生します。定数 e の値は約 2.718282 です。
- Exp 関数は Log 関数の逆関数であり、逆対数と呼ばれることもあります。

**関連項目**

LOG、LOG10

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4
lf1 = EXP(2)                    ’自然対数が底の 2 乗を lf1 に代入します。
lf2 = EXP(lf3 / lf4)            ’自然対数が底の (lf3 / lf4) 乗を lf2 に代入します。
```

# INT（関数）【SLIM 準拠】

**機　能**

指定された値を超えない最大の整数値を得ます。

**書　式**

INT(< 数式 >)

**説　明**

＜数式＞の値を超えない最大の整数を返します。

オーバフローした場合には、＜数式＞と同符号の整数の最大値になります。

**関連項目**

**用　例**

```
DIM li1 As Integer
li1 = INT(123.456)          ' 小数点以下を切り捨て、整数変数 li1 に代入します。
                            '123.456->123
```

# LOG（関数）【SLIM 準拠】

**機　能**

自然対数を得ます。

**書　式**

LOG(< 数式 >)

**説　明**

< 数式 > で指定した値の自然対数を得ます。

< 数値 > には 0 よりも大きい値を指定します。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**補足説明**

自然対数とは、定数 e を底とする対数です。定数 e の値は約 2.718282 です。任意の数値 x の n を底とする対数は、次に示すように x の自然対数を n の自然対数で割ることによって得られます。

Lognx=Logex / Logen

**関連項目**

EXP、LOG10

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4
lf1 = LOG(2)                  '2 の自然対数を lf1 に代入します。
lf2 = LOG(lf3 / lf4)          '(lf3 / lf4) の自然対数を lf2 に代入します。
```

# LOG10（関数）【SLIM 準拠】

**機　能**

常用対数を得ます。

**書　式**

LOG10(< 数式 >)

**説　明**

< 数式 > で指定した値の常用対数を得ます。常用対数の底は 10 です。

< 数値 > には 0 よりも大きい値を指定します。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**関連項目**

EXP、LOG

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4
lf1 = LOG10(2)              '2 の常用対数を lf1 に代入します。
lf2 = LOG10(lf3 / lf4)      '(lf3 / lf4) の常用対数を lf2 に代入します。
```

# POW（関数）【SLIM 準拠】

**機　能**

べき乗を求めます。

**書　式**

POW(< 底 >,< 指数 >)

**説　明**

< 底 > の < 指数 > で指定した値のべき乗を得ます。

< 底 >、< 指数 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**関連項目**

EXP

**注意事項**

べき乗の計算で第 1 引数が負の場合は、ドメインエラーが発生します。

< 底 >、< 指数 > に倍精度実数が含まれる場合は、精度は 15 桁までしか保証されません。

< 底 >、< 指数 > に倍精度実数が含まれない場合は、精度は 7 桁までしか保証されません。

**用　例**

```
DEFSNG lf1, lf2
DEFINT li1, li2
li1 = POW(5, 2)              '5 の 2 乗を li1 に代入します。
lf1 = POW(lf2, li2)          'lf2 の li2 乗を lf1 に代入します。
```

# MAX（関数）【SLIM 準拠】

**機　能**

最大値を抽出します。

**書　式**

MAX(< 数式 >,< 数式 >[,< 数式 >…])

**説　明**

任意個数の < 数式 > から最大値を抽出します。

数式の個数は最大 32 個です。

**補足説明**

Min 関数および Max 関数を使用すると、指定した集計方法またはグループのフィールドの最小値と最大値を求めることができます。たとえば、運送料の最高額と最低額を調べる場合にこの関数を使用します。集計方法の指定がない場合は、テーブル全体が対象となります。

**関連項目**

MIN

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5, lf6
DEFINT li1, li2, li3, li4, li5, li6
li1 = MAX(1, 2)                        '(1, 2) の内の最大値を li1 に代入します。
li2 = MAX(li3, li4, li5, li6)          '(li3, li4, li5, li6) の内の最大値を li2
                                       ' に代入します。
lf1 = MAX(lf2, lf3, lf4, lf5, lf6)     '(lf2, lf3, lf4, lf5, lf6) の内の最大値
                                       ' を lf1 に代入します。
```

# MIN（関数）【SLIM 準拠】

**機　能**

最小値を抽出します。

**書　式**

MIN(< 数式 >,< 数式 >[,< 数式 >…])

**説　明**

任意個数の < 数式 > から最小値を抽出します。

数式の個数は最大 32 個です。

**補足説明**

Min 関数および Max 関数を使用すると、指定した集計方法またはグループのフィールドの最小値と最大値を求めることができます。たとえば、運送料の最高額と最低額を調べる場合にこの関数を使用します。集計方法の指定がない場合は、テーブル全体が対象となります。

**関連項目**

MAX

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5, lf6
DEFINT li1, li2, li3, li4, li5, li6
li1 = MIN(1, 2)                        '(1, 2) の内の最小値を li1 に代入します。
li2 = MIN(li3, li4, li5, li6)          '(li3, li4, li5, li6) の内の最小値を li2 に
                                       ' 代入します。
lf1 = MIN(lf2, lf3, lf4, lf5, lf6)     '(lf2, lf3, lf4, lf5, lf6) の内の最小値を
                                       'lf1 に代入します。
```

# RND（関数）【SLIM 準拠】

**機　能**

0 以上 1 以下の乱数を発生します。

**書　式**

RND (< 数式 >)

**説　明**

< 数式 > の値により、下表に示すように処理します。

< 数式 > は整数値を取ります。実数値を入れた場合、切り捨てが行なわれ整数に変換されます。

| 数式の値 | 処　　理 |
|---|---|
| 数式 < 0 | < 数式 > を乱数のシード値とします。シード値が同じであれば、RND 関数の戻り値は常に同一値です。 |
| 数式 = 0 | 直前に発生した乱数を発生します。ロボットコントローラの電源投入後、一度も乱数を発生していない場合は、0 を返します。 |
| 数式 > 0 | 乱数系列の次の乱数を発生します。 |

**補足説明**

Rnd 関数は 0 以上、1 未満の範囲の値を返します。引数の値によって、Rnd 関数が返す乱数が決まります。初期シード値が変わらない限り、一連の Rnd 関数が返す乱数系列は同じになります。これは、連続する各 Rnd 関数が乱数系列の中の直前の乱数をシード値として、次の乱数をそれぞれ生成するためです。

**注意**

乱数のシード値が同一の場合、その後得られる乱数系列は同一の系列となります。毎回違った系列を得たい場合には、シード値として TIMER 関数の戻り値などを使ってください。

**用　例**

```
DIM array(10) As Single
array(0) = RND(-TIMER)        ' 乱数のシード値を TIMER 関数で与えます。
FOR I1 = 1 TO 9
  array(I1) = RND(1)          ' 乱数を得ます。
NEXT I1
```

# SGN（関数）

**機　能**

符号を調べます。

**書　式**

SGN (< 数式 >)

**説　明**

< 数式 > の符号を調べ、以下の数値を返します。

| | |
|---|---|
| 正の場合 | 1 |
| 0 の場合 | 0 |
| 負の場合 | -1 |

**用　例**

```
DEFSNG lf1, lf2
lf2 = SGN(lf1)                    ' 実数型変数 lf1 の符号を返します。
```

# SQR（関数）【SLIM 準拠】

**機　能**

平方根を得ます。

**書　式**

SQR(< 数式 >)

**説　明**

< 数式 > の値に対する平方根 ( スクエアルート ) を得ます。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

< 数式 > は 0 以上の値でなければなりません。

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5, lf6, lf7
lf1 = SQR(2)                          '2 の平方根を lf1 に代入します。
lf2 = SQR(lf4)                        'lf4 の平方根を lf2 に代入します。
lf3 = SQR(lf5 + lf6) * lf7            '(lf5 + lf6) の平方根に lf7 を掛けた値を lf3 に代入
                                      ' します。
```

# 15.2　三角関数

## ACOS（関数）【SLIM 準拠】

**機　能**

逆余弦 ( アークコサイン ) を得ます。

**書　式**

ACOS(< 数式 >)

**説　明**

< 数式 > の値に対する逆余弦値を得ます。

得られる値は度で、0 から 180 までの範囲です。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**関連項目**

SIN、TAN、ASIN、ATN、ATN2

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5, lf6, lf7
lf1 = ACOS(0.5)                  '0.5 の逆余弦値を lf1 に代入します。
lf2 = ACOS(lf4 / 2)              '(lf4 / 2) の逆余弦値を lf2 に代入します。
lf3 = ACOS(lf5 / lf6) * lf7      '(lf5 / lf6) の逆余弦値に lf7 を掛けた値を
                                 'lf3 に代入します。
```

# ASIN（関数）【SLIM 準拠】

**機　能**

逆正弦 ( アークサイン ) を得ます。

**書　式**

ASIN(< 数式 >)

**説　明**

< 数式 > の値に対する逆正弦値を得ます。

得られる値は度で、-90 から 90 までの範囲です。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**関連項目**

SIN、COS、TAN、ACOS、ATN、ATN2

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5, lf6, lf7
lf1 = ASIN(0.5)                  '0.5 の逆正弦値を lf1 に代入します。
lf2 = ASIN(lf4 / 2)              '(lf4 / 2) の逆正弦値を lf2 に代入します。
lf3 = ASIN(lf5 / lf6) * lf7      '(lf5 / lf6) の逆正弦値に lf7 を掛けた値を lf3 に
                                 ' 代入します。
```

# ATN（関数）【SLIM 準拠】

**機　能**

逆正接 ( アークタンジェント ) を得ます。

**書　式**

ATN(< 数式 >)

**説　明**

< 数式 > の値に対する逆正接値を得ます。

得られる値は度で、-90 から 90 までの範囲です。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

**補足説明**

- Atn 関数は、直角三角形の 2 辺の比を引数 (number) として受け取り、対応する角度を返します。ここでいう 2 辺とは、直角をはさむ 2 つの辺を指します。2 辺の比は、求める角の反対側の辺 ( 対辺 ) の長さをもう一方の辺 ( 底辺、つまり求める角に隣接する側の辺 ) の長さで割った値です。戻り値は、-π/2 ～ π/2 の範囲の値 ( 単位はラジアン ) になります。角度の単位を度からラジアンに変換するには、度に π/180 を掛けます。ラジアンから度に変換するには、ラジアンに 180/π を掛けます。
- Atn 関数は Tan 関数の逆三角関数です。Tan 関数は、引数として角度を受け取り、その角度を含む直角三角形の直角をはさむ 2 辺の比を返します。Atn 関数と、タンジェントの逆数であるコタンジェント (1/ タンジェント ) の違いに気を付けてください。

**関連項目**

SIN、COS、TAN、ASIN、ACOS、ATN2

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5, lf6, lf7
lf1 = ATN(0.5)                  '0.5 の逆正接値を lf1 に代入します。
lf2 = ATN(lf4 / 2)              '(lf4 / 2) の逆正接値を lf2 に代入します。
lf3 = ATN(lf5 / lf6) * lf7      '(lf5 / lf6) の逆正接値に lf7 を掛けた値を lf3 に代
                                ' 入します。
```

# ATN2（関数）【SLIM 準拠】

**機　能**

数式 1 を数式 2 で除算した逆正接 ( アークタンジェント ) を得ます。

**書　式**

ATN2(< 数式 1>,< 数式 2>)

**説　明**

< 数式 1> を < 数式 2> で除算した値に対する逆正接値を得ます。

得られる値は度で、-180 から 180 までの範囲です。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

ATN2 の数値範囲は以下のようになります。

| | 第一象限 | 第二象限 | 第三象限 | 第四象限 |
|---|---|---|---|---|
| 下限値 | 0 | 90 | -180 | -90 |
| 上限値 | 90 | 180 | 90 | - 0 |

**関連項目**

SIN、COS、TAN、ASIN、ACOS、ATN

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5, lf6, lf7, lf8
lf1 = ATN2(1, 2)                  '(1, 2) の逆正接値を lf1 に代入します。
lf2 = ATN2(lf4, lf5)              '(lf4, lf5) の逆正接値を lf2 に代入します。
lf3 = ATN2(lf6, lf7) * lf8        '(lf6, lf7) の逆正接値に lf8 を掛けた値を lf3 に代入
                                  ' します。
```

# COS（関数）【SLIM 準拠】

**機　能**

余弦 ( コサイン ) を得ます。

**書　式**

COS(< 数式 >)

**説　明**

< 数式 > の値に対する余弦値を得ます。

< 数式 > の単位は度で指定します。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

引数をラジアン (RAD) で入力する場合は、定数の後に RAD を付けます。

**補足説明**

Cos 関数は、引数として角度を受け取り、その角度を含む直角三角形の 2 辺の比を返します。ここでいう 2 辺とは、引数 number に指定した角をはさむ 2 つの辺を指します。2 辺の比は、短い方の辺 ( 底辺 ) の長さを、もう一方の辺 ( 斜辺 ) の長さで割った値です。戻り値は、-1 ～ 1 の範囲の値になります。角度の単位を度からラジアンに変換するには、度に π/180 を掛けます。ラジアンから度に変換するには、ラジアンに 180/π を掛けます。

**関連項目**

SIN、TAN、ASIN、ACOS、ATN、ATN2

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5
lf1 = COS(0.78525RAD)          '0.78525（ラジアン）の余弦値を lf1 に代入します。
lf2 = COS(lf4)                 'lf4 の余弦値を lf2 に代入します。
lf3 = COS(45) * lf5            '45 度の余弦値に lf5 を掛けた値を lf3 に代入します。
```

# SIN（関数）【SLIM 準拠】

**機　能**

正弦 ( サイン ) を得ます。

**書　式**

SIN(< 数式 >)

**説　明**

< 数式 > の値に対する正弦値を得ます。

< 数式 > の単位は度で指定します。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

引数をラジアン (RAD) で入力する場合は、定数の後に RAD を付けます。

**補足説明**

Sin 関数は、引数として角度を受け取り、その角度を含む直角三角形の 2 辺の比を返します。ここでいう 2 辺とは、指定した角の反対側の辺 ( 対辺 ) と斜辺を指します。2 辺の比は、対辺の長さを斜辺の長さで割った値です。戻り値は、-1 ～ 1 の範囲の値になります。角度の単位を度からラジアンに変換するには、度に π/180 を掛けます。ラジアンから度に変換するには、ラジアンに 180/π を掛けます。

**関連項目**

COS、TAN、ASIN、ACOS、ATN、ATN2

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5
lf1 = SIN(0.78525RAD)        '0.78525（ラジアン）の正弦値を lf1 に代入します。
lf2 = SIN(lf4)               'lf4 の正弦値を lf2 に代入します。
lf3 = SIN(45) * lf5          '45 度の正弦値に lf5 を掛けた値を lf3 に代入します。
```

# TAN（関数）【SLIM 準拠】

**機　能**

正接 ( タンジェント ) を得ます。

**書　式**

TAN(< 数式 >)

**説　明**

< 数式 > の値に対する正接値を得ます。

< 数式 > の単位は度で指定します。

< 数式 > に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

引数をラジアン (RAD) で入力する場合は、定数の後に RAD を付けます。

**補足説明**

Tan 関数は、引数として角度を受け取り、その角度を含む直角三角形の 2 辺の比を返します。ここでいう 2 辺とは、直角をはさむ 2 つの辺を指します。2 辺の比は、指定した角の反対側の辺 ( 対辺 ) の長さを、もう一方の辺 ( 底辺、つまり指定した角に隣接する側の辺 ) の長さで割った値です。角度の単位を度からラジアンに変換するには、度に π/180 を掛けます。ラジアンから度に変換するには、ラジアンに 180/π を掛けます。

**関連項目**

SIN、COS、ASIN、ACOS、ATN、ATN2

**用　例**

```
DEFSNG lf1, lf2, lf3, lf4, lf5
lf1 = TAN(0.78525RAD)          '0.78525（ラジアン）の正接値を lf1 に代入します。
lf2 = TAN(lf4)                 'lf4 の正接値を lf2 に代入します。
lf3 = TAN(45) * lf5            '45 度の正接値に lf5 を掛けた値を lf3 に代入します。
```

## 15.3　角度変換

# DEGRAD（関数）【SLIM 準拠】

**機　能**

単位をラジアンに変換します。

**書　式**

DEGRAD(< 数式 >)

**説　明**

< 数式 > に指定した値を度からラジアンに変換します。

**関連項目**

RADDEG

**用　例**

```
DEFSNG lf1, lf2, lf3
lf1 = DEGRAD(90)              '90 をラジアンに変換した値を lf1 に代入します。
lf2 = SIN(DEGRAD(lf3 + 20))   '(lf3 + 20) をラジアンに変換した値の正弦値を lf2 に
                              ' 代入します。
```

# RAD（関数）

**機　能**

ラジアンで与えた数値を角度に変換します。

**書　式**

＜数値＞RAD

**説　明**

ラジアンで与えられる＜数値＞を角度に変換し演算します。
＜数値＞には定数を指定します。

**関連項目**

RADDEG

**注意事項**

＜数値＞には変数や予約語（PI など）を使用することはできません。
変数や予約語を使用する場合は、RADDEG をご使用ください。

**用　例**

```
DIM lf1 As SINGLE
lf1 = 2RAD                    '2 ラジアンを角度に変換し lf1 に代入します。
```

# RADDEG（関数）【SLIM 準拠】

**機　能**

単位を度に変換します。

**書　式**

RADDEG(< 数式 >)

**説　明**

< 数式 > に指定した値をラジアンから度に変換します。

**関連項目**

RAD、DEGRAD

**用　例**

```
DEFSNG lf1, lf2
lf1 = RADDEG(1.5705)          '1.5705 を度に変換した値を lf1 に代入します。
lf2 = RADDEG( PI / 2)         '(π / 2) を度に変換した値の正弦値を lf2 に代入しま
                              ' す。
```

# 15.4　速度変換

## MPS（関数）

**機　能**

指定した速度値（mm/sec）を CP 動作時の最大内部速度に対する割合（%）に変換します。

**書　式**

MPS(< 数式 >)

**説　明**

速度指定のコマンド「SPEED」と合わせて使用することにより、速度指定を速度値 (mm/sec) で指定できます。
指定する速度値の単位は、mm/sec です。

**関連項目**

SPEED

**注意事項**

内部速度を変換するコマンドですので実際の動きは外部速度指定の影響を受けます。
例：SPEED MPS（50) と指定し、外部速度 50%の場合、実際の速度は 25mm/sec となります。

**用　例**

```
SPEED MPS(50)                    ' 内部移動速度を 50 (mm/sec) にします。
```

# RPM（関数）【Ver.2.8 以降】

**機　能**

指定した軸番号の回転数 (rpm) を PTP 動作時の最大内部速度に対する割合（%）に変換します。

**書　式**

RPM（< 軸番号 >, < 数式 >）

**説　明**

指定した軸番号の回転数 (rpm) を PTP 動作時の最大内部速度に対する割合（%）に変換します。
単位は、rpm です。

**関連項目**

SPEED

**注意事項**

内部速度を変換するコマンドですので実際の動きは外部速度指定の影響を受けます。
例：SPEED RPM（7,50) と指定し、外部速度 50%の場合、7 軸の動きは 25rpm となります。
付加軸の場合、回転軸を指定すると指定する回転数は出力段の回転数を指します。
直動軸を指定すると入力段の回転数 ( モータの回転数 ) となります。

**用　例**

```
SPEED RPM(7, 50)                    '7 軸の内部速度を 50rpm にします。
```

## 15.5　時間関数

# SEC（関数）

**機　能**

秒の単位で与えた数値をミリ秒に変換します。

**書　式**

＜数値＞SEC

**説　明**

秒で与えられる＜数値＞をミリ秒に変換し演算します。ミリ秒で設定する命令などで使用します。

**用　例**

```
DELAY(10SEC)                    '10 秒間経過するまで待ちます。
```

## 15.6　ベクトル

# AVEC（関数）

**機　能**

アプローチベクトルを抽出します。

**書　式**

AVEC(< 同次変換型 >)

**説　明**

同次変換型座標からアプローチベクトルを抽出します。

**関連項目**

OVEC、PVEC

**用　例**

```
DEFTRN lt1, lt2, lt3
DEFVEC lv1, lv2
lv1 = AVEC(lt1)              'lt1 のアプローチベクトルを lv1 に代入します。
lv2 = AVEC(lt2 * lt3)        '(lt2 * lt3) の値のアプローチベクトルを lv2 に代入
                             ' します。
```

# OVEC（関数）

**機　能**

オリエントベクトルを抽出します。

**書　式**

OVEC(< 同次変換型 >)

**説　明**

同次変換型座標からオリエントベクトルを抽出します。

**関連項目**

AVEC、PVEC

**用　例**

```
DEFTRN lt1, lt2, lt3
DEFVEC lv1, lv2
lv1 = OVEC(lt1)              'lt1 のオリエントベクトルを lv1 に代入します。
lv2 = OVEC(lt2 * lt3)        '(lt2 * lt3) の値のオリエントベクトルを lv2 に代入
                             ' します。
```

# PVEC（関数）

**機　能**

位置ベクトルを抽出します。

**書　式**

PVEC({< 同次変換型 >|< ポジション型 >})

**説　明**

同次変換型座標または、ポジション型座標から、位置ベクトルを抽出します。

**関連項目**

AVEC、OVEC、RVEC

**用　例**

```
DIM lt1 As Trans
DIM lv1 As Vector
DIM lp1 As Position
lv1 = PVEC(lt1)                 'lt1 の位置ベクトルを lv1 に代入します。
```

6 軸

```
lv1 = PVEC(lp1 + (100, 200, 0, 0, 0, 0) )
                                'lp1 + (100, 200, 0, 0, 0, 0) の位置ベクトルを lv1
                                ' に代入します。
```

4 軸

```
lv1 = PVEC(lp1 + (100, 200, 0, 0) )
                                'lp1 + (100, 200, 0, 0) の位置ベクトルを lv1 に
                                ' 代入します。
```

# MAGNITUDE（関数）

**機　能**

ベクトルの大きさを得ます。

**書　式**

MAGNITUDE(< ベクトル型 >)

**説　明**

ベクトル型座標の値に対するベクトルの大きさを得ます。

**関連項目**

DIST

**用　例**

```
DEFSNG lf1, lf2
DIM lv1 As Vector
lf1 = MAGNITUDE((10, 10, 10))   '(10, 10, 10) のベクトルの大きさを lf1 に代入します。
lf2 = MAGNITUDE(lv1)            'lv1 のベクトルの大きさを lf2 に代入します。
```

# 15.7　ポーズデータ型変換

## J2P（関数）

**機　能**

ジョイント型からポジション型に変換します。

**書　式**

J2P（< ジョイント型 >[ ,<J2P オプション >])

**説　明**

< ジョイント型 > に指定したジョイント型のデータをポジション型のデータに変換します。得られる値は、そのとき設定されているツール、ワーク座標系を反映したものとなります。

<J2P オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。I 型変数を指定します。<J2P オプション > を指定すると J2P コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**関連項目**

J2T、P2J、P2T、T2J、T2P

**用　例**

```
DEFPOS lp1, lp2
DIM lj1 As Joint
lp1 = J2P(lj1)                    'lj1 をポジション型に変換したデータを lp1 に代入し
                                  ' 次の行ステップに移ります。
lp1 = J2P(lj1,li1)                'J2P コマンド失敗時、変数 li1 にエラーコードを格納し
                                  ' 次の行ステップに移ります。
```

6 軸

```
lp2 = J2P((0, 0, 90, 0, 0, 0))    '(0, 0, 90, 0, 0, 0) をポジション型に変換したデー
                                  ' タを lp2 に代入します。
```

4 軸

```
lp2 = J2P((0, 0, 300, 0))         '(0, 0, 300, 0) をポジション型に変換したデータを lp2
                                  ' に代入します。
```

# J2T（関数）

**機　能**

ジョイント型から同次変換型に変換します。

**書　式**

J2T(< ジョイント型 >[ ,<J2T オプション >])

**説　明**

< ジョイント型 > に指定したジョイント型のデータを同次変換型のデータに変換します。得られる値は、そのとき設定されているツール、ワーク座標系を反映したものとなります。

<J2T オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。I 型変数を指定します。<J2T オプション > を指定すると J2T コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**関連項目**

J2P、P2J、P2T、T2J、T2P

**用　例**

```
DEFTRN lt1, lt2
DIM lj1 As Joint
lt1 = J2T(lj1)                    'lj1 を同次変換型に変換したデータを lt1 に代入しま
                                  ' す。
lt1 = J2T(lj1,li1)                'J2T コマンド失敗時、変数 li1 にエラーコードを格納し
                                  ' 次の行ステップに移ります。
```

6 軸

```
lt2 = J2T((0, 0, 90, 0, 0, 0))    '(0, 0, 90, 0, 0, 0) を同次変換型に変換したデータ
                                  ' を lt2 に代入します。
```

4 軸

```
lt2 = J2T((0, 0, 300, 0))         '(0, 0, 300, 0) を同次変換型に変換したデータを lt2
                                  ' に代入します。
```

# P2J（関数）

**機　能**

ポジション型からジョイント型に変換します。

**書　式**

P2J(< ポジション型 >[ ,<P2J オプション >])

**説　明**

< ポジション型 > に指定したデータをジョイント型に変換します。

ポジション型データの FIG の値が -1( 未定 ) の場合は、現在の形態で変換します。得られる値は、そのとき設定されているツール、ワーク座標系を反映したものとなります。

<P2J オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。I 型変数を指定します。<P2J オプション > を指定すると P2J コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**関連項目**

J2P、J2T、P2T、T2J、T2P、CURFIG

**用　例**

```
DEFJNT lj1, lj2
DIM lp1 As position
lj1 = P2J(lp1)                    'lp1 を現在の形態でジョイント型に変換したデータを
                                  'lj1 に代入します。
lj1 = P2J(lp1,li1)                'P2J コマンド失敗時、変数 li1 にエラーコードを格納し
                                  ' 次の行ステップに移ります。
```

**6 軸**

```
lj2 = P2J((0, 0, 90, 0, 0, 180))  '(0, 0, 90, 0, 0, 180) を現在の形態でジョイント型
                                  ' に変換したデータを lj2 に代入します。
```

**4 軸**

```
lj2 = P2J((100, 100, 300, 45))    '(100, 100, 300, 45) を現在の形態でジョイント型に
                                  ' 変換したデータを lj2 に代入します。
```

# P2T（関数）

**機　能**

ポジション型から同次変換型に変換します。

**書　式**

P2T(< ポジション型 >[ ,<P2T オプション >])

**説　明**

< ポジション型 > に指定したデータを同次変換型に変換します。得られる値は、そのとき設定されているツール、ワーク座標系を反映したものとなります。

<P2T オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。I 型変数を指定します。<P2T オプション > を指定すると P2T コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**関連項目**

J2P、J2T、P2J、T2J、T2P

**用　例**

```
DEFTRN lt1, lt2
DIM lp1 As position
lt1 = P2T(lp1)                          'lp1 を同次変換型に変換したデータを lt1 に代入しま
                                        ' す。
lt1 = P2T(lp1,li1)                      'P2T コマンド失敗時、変数 li1 にエラーコードを格納し
                                        ' 次の行ステップに移ります。
```

6 軸

```
lt2 = P2T((350, 0, 450, 0, 0, 180)) '(350, 0, 450, 0, 0, 180) を同次変換型に変換したデ
                                        'ータを lt2 に代入します。
```

4 軸

```
lt2 = P2T((100, 100, 300, 45))          '(100, 100, 300, 45) を同次変換型に変換したデータ
                                        ' を lt2 に代入します。
```

# T2J（関数）

**機　能**

同次変換型からジョイント型に変換します。

**書　式**

T2J(< 同次変換型 >[ ,<T2J オプション >])

**説　明**

< 同次変換型 > に指定したデータをジョイント型に変換します。

同次変換型データの FIG の値が -1（未定）の場合は、現在の形態で変換されます。

得られる値は、そのとき設定されているツール、ワーク座標系を反映したものとなります。

<T2J オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。I 型変数を指定します。<T2J オプション > を指定すると T2J コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**関連項目**

J2P、J2T、P2J、P2T、T2P、CURFIG

**用　例**

```
DEFJNT lj1, lj2, lj3
DEFTRN lt1, lt2
DIM li1 As Integer
lj1 = T2J(lt1)                  'lt1 を現在の形態でジョイント型に変換したデータを
                                'lj1 に代入します。
lj2 = T2J(lt2)                  'lt2 を 0 の形態でジョイント型に変換したデータを lj2
                                ' に代入します。
lj3 = T2J((350, 0, 450, 0, 1, 0, 0, 0, -1))
                                ' 同次変換型データ (350, 0, 450, 0, 1, 0, 0, 0, -1)
                                ' をジョイント型に変換したデータを lj3 に代入します。
lj1 = T2J(lt1,li1)              'T2J コマンド失敗時、変数 li1 にエラーコードを格納し
                                ' 次の行ステップに移ります。
```

# T2P（関数）

**機　能**

同次変換型からポジション型に変換します。

**書　式**

T2P(< 同次変換型 >[ ,<T2P オプション >])

**説　明**

< 同次変換型 > に指定したデータをポジション型に変換します。得られる値は、そのとき設定されているツール、ワーク座標系を反映したものとなります。

<T2P オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。I 型変数を指定します。<T2P オプション > を指定すると T2P コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**関連項目**

J2P、J2T、P2J、P2T、T2J

**用　例**

```
DEFPOS lp1, lp2
DIM lt1 As Trans
lp1 = T2P(lt1)                  'lt1 をポジション型に変換したデータを lp1 に代入し
                                ' ます。
lp2 = T2P(350, 0, 450, 0, 1, 0, 0, 0, -1))
                                ' 同次変換型データ (350, 0, 450, 0, 1, 0, 0, 0, -1)
                                ' をポジション型に変換したデータを lp2 に代入します。
lp1 = T2P(lt1,li1)              'T2P コマンド失敗時、変数 li1 にエラーコードを格納し
                                ' 次の行ステップに移ります。
```

# TINV（関数）

**機　能**

同次変換型の逆行列を計算します。

**書　式**

TINV(< 同次変換型 >[ ,<TINV オプション >])

**説　明**

< 同次変換型 > に指定したデータを逆変換します。

<TINV オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。 I 型変数を指定します。<TINV オプション > を指定すると TINV コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**用　例**

```
DEFTRN lt1, lt2, Lt3
lt1 = TINV((350, 0, 450, 0, 1, 0, 0, 0, -1))
                                ’(350, 0, 450, 0, 1, 0, 0, 0, -1) の逆行列を lt1 に
                                ’ 代入します。
lt2 = TINV(lt3)                 ’lt3 の逆行列を lt2 に代入します。
lt2 = TINV(lt1,li1)             ’TINV コマンド失敗時、変数 li1 にエラーコードを格納し
                                ’ 次の行ステップに移ります。
```

# NORMTRN（関数）【Ver.1.8 以降】

**機　能**

同次変換型の正規化計算を行います。

**書　式**

NORMTRN(< 同次変換型 >[ ,<NORMTRN オプション >])

**説　明**

< 同次変換型 > に指定したデータを正規化します。

正規化 … アプローチベクトルを基準としてオリエントベクトルを直交化し、アプローチ、オリエントベクトルを単位ベクトル化します。

<NORMTRN オプション > は省略可能な Result 格納変数です（Ver.3.0 以降）。 I 型変数を指定します。<NORMTRN オプション > を指定すると NORMTRN コマンド実行時に内部でエラーが発生してもプログラムを停止させず、内部エラーのエラーコードを指定した変数に格納します。

**用　例**

```
DEFTRN lt1, lt2
lt1 = NORMTRN((350, 0, 450, 0, 1, 0, 0, 0, -1))

lt1 = NORMTRN(lt2)
lt2 = NORMTRN(lt1,li1)        'NORMTRN コマンド失敗時、変数 li1 にエラーコードを格納し
                              ' 次の行ステップに移ります。
```

# 15.8　距離抽出

## DIST（関数）【SLIM 準拠】

**機　能**

2 点間の距離を返します。

**書　式**

DIST(< ポジション型 1>,< ポジション型 2>)

**説　明**

< ポジション型 1> と < ポジション型 2> との距離を得ます。

**用　例**

```
DEFSNG lf1, lf2
DEFPOS lp1, lp2, lp3
```

**6 軸**

```
lf1 = DIST((350, 0, 450, 0, 0, 180), lp1)
                              '(350, 0, 450, 0, 0, 180) と lp1 間の距離を lf1 に代
                              ' 入します。
```

**4 軸**

```
lf1 = DIST((350, 0, 300, 45), lp1)
                              '(350, 0, 300, 45) と lp1 間の距離を lf1 に代入しま
                              ' す。

lf2 = DIST(lp2, lp3)          'lp2 と lp3 間の距離を lf2 に代入します。
```

# 15.9　形態成分

## FIG（関数）

**機　能**

形態を抽出します。

**書　式**

FIG({< ポジション型 >|< 同次変換型 >})

**説　明**

ポジション型、同次変換型から形態を抽出します。

**関連項目**

CURFIG、LETF、ロボット形態 ( 付録 2)

**用　例**

```
DIM li1 As Integer
DIM lp1 As position
li1 = FIG(lp1)                                  'lp1 の形態を li1 に代入します。
```

6 軸

```
li1 = FIG(lp1 + (0, 0, 100, 0, 0, 0) )          'lp1 + (0, 0, 100, 0, 0, 0) の形態を li1 に
                                                ' 代入します。
```

4 軸

```
li1 = FIG(lp1 + (0, 0, 100, 0) )                'lp1 + (0, 0, 100, 0) の形態を li1 に代入し
                                                ' ます。
```

## 15.10 角度成分

# JOINT（関数）

**機　能**

ジョイント型座標から角度を抽出します。

**書　式**

JOINT(< 軸番号 >,< ジョイント型 >)

**説　明**

<ジョイント型>で指定したジョイント型座標から、<軸番号>で指定した一つの軸の関節角度を抽出します。

**関連項目**

LETJ

**用　例**

```
DEFJNT lj1, lj2
DEFSNG lf1, lf2
DIM li1 As Integer
lf1 = JOINT(1, lj1)          'lj1 の 1 軸の関節角度を lf1 に代入します。
lf2 = JOINT(li1, lj2)        'lj2 の li1 の値の軸の関節角度を lf2 に代入します。
```

6 軸

```
lf1 = JOINT(1, lj1 + (10, 10, 0, 0, 0, 0) )
                             'lj1 + (10, 10, 0, 0, 0, 0) の 1 軸の関節角度を lf1
                             ' に代入します。
```

4 軸

```
lf1 = JOINT(1, lj1 + (10, 10, 0, 0) )
                             'lj1 + (10, 10, 0, 0) の 1 軸の関節角度を lf1 に代入
                             ' します。
```

# 15.11 軸成分

## POSX（関数）【SLIM 準拠】

**機　能**

X 成分を抽出します。

**書　式**

POSX({< ポジション型 >|< ベクトル型 >})

**説　明**

ポジション型またはベクトル型の座標から、X 成分を抽出します。

**関連項目**

POSY、POSZ

**用　例**

```
DIM lf1 As Single
DIM lp1 As Position
lf1 = POSX(lp1)                    'lp1 の X 成分を lf1 に代入します。
```

6 軸

```
lf1 = POSX(lp1 + (100, 100, 100, 0, 0, 0) )
                                   'lp1 + (100, 100, 100, 0. 0, 0) の X 成分を lf1 に代
                                   ' 入します。
```

4 軸

```
lf2 = POSX(lp1 + (100, 100, 100, 0) )
                                   'lp1 + (100, 100, 100, 0) の X 成分を lf1 に代入しま
                                   ' す。
```

# POSY（関数）【SLIM 準拠】

**機　能**

Y 成分を抽出します。

**書　式**

POSY({< ポジション型 >|< ベクトル型 >})

**説　明**

ポジション型またはベクトル型の座標から、Y 成分を抽出します。

**関連項目**

POSX、POSZ

**用　例**

```
DIM lf1 As Single
DIM lp1 As Position
lf1 = POSY(lp1)                    'lp1 の Y 成分を lf1 に代入します。
```

6 軸

```
lf1 = POSY(lp1 + (100, 100, 100, 0, 0, 0) )
                                   'lp1 + (100, 100, 100, 0, 0, 0) の Y 成分を lf1 に代
                                   ' 入します。
```

4 軸

```
lf1 = POSY(lp1 + (100, 100, 100, 0) )
                                   'lp1 + (100, 100, 100, 0) の Y 成分を lf1 に代入しま
                                   ' す。
```

# POSZ（関数）【SLIM 準拠】

**機　能**

Z 成分を抽出します。

**書　式**

POSZ({< ポジション型 >|< ベクトル型 >})

**説　明**

ポジション型またはベクトル型の座標から、Z 成分を抽出します。

**関連項目**

POSX、POSY

**用　例**

```
DIM lf1 As Single
DIM lp1 As Position
lf1 = POSZ(lp1)                    'lp1 の Z 成分を lf1 に代入します。
```

6 軸

```
lf1 = POSZ(lp1 + (100, 100, 100, 0, 0, 0) )
                                   'lp1 + (100, 100, 100, 0, 0, 0) の Z 成分を lf1 に代
                                   ' 入します。
```

4 軸

```
lf1 = POSZ(lp1 + (100, 100, 100, 0) )
                                   'lp1 + (100, 100, 100, 0) の Z 成分を lf1 に代入しま
                                   ' す。
```

## 15.12 回転成分

# POSRX（関数）

**機　能**

X 軸回転成分を抽出します。

**書　式**

POSRX(< ポジション型 >)

**説　明**

< ポジション型 > で指定したポジション型の座標から、X 軸回転成分を抽出します。

**関連項目**

POSRY、POSRZ

**用　例**

```
DIM lf1 As Single
DIM lp1 As Position
lf1 = POSRX(lp1)              'lp1 の X 軸回転成分を lf1 に代入します。
```

# POSRY（関数）

**機　能**

Y 軸回転成分を抽出します。

**書　式**

POSRY(< ポジション型 >)

**説　明**

< ポジション型 > で指定したポジション型の座標から、Y 軸回転成分を抽出します。

**関連項目**

POSRX、POSRZ

**用　例**

```
DIM lf1 As Single
DIM lp1 As Position
lf1 = POSRY(lp1)                    'lp1 の Y 軸回転成分を lf1 に代入します。
```

# POSRZ（関数）

**機　能**

Z 軸回転成分を抽出します。

**書　式**

POSRZ(< ポジション型 >)

**説　明**

< ポジション型 > で指定したポジション型の座標から、Z 軸回転成分を抽出します。

**関連項目**

POSRX、POSRY

**用　例**

```
DIM lf1 As Single
DIM lp1 As Position
lf1 = POSRZ(lp1)                ’lp1 の Z 軸回転成分を lf1 に代入します。
```

# POST（関数）

**機　能**

T 軸回転成分を抽出します。

**書　式**

POST(< ポジション型 >)

**説　明**

< ポジション型 > で指定したポジション型の座標から、T 軸回転成分を抽出します。

**関連項目**

**用　例**

```
DIM lf1 As Single
DIM lp1 As Position
lf1 = POST(lp1)                ’lp1 の T 軸回転成分を lf1 に代入します。
```

## 15.13 姿勢成分

# RVEC（関数）

**機　能**

姿勢を抽出します。

**書　式**

RVEC(< ポジション型 >)

**説　明**

ポジション型の座標から、姿勢を抽出します。

**関連項目**

PVEC

**用　例**

```
DIM lp1 As position
DIM lv1 As Vector
lv1 = RVEC(lp1)                    'lp1 の姿勢を lv1 に代入します。
```

## 15.14 位置関数

# AREAPOS（関数）

**機　能**

干渉チェックの行なわれる領域の中心位置と直方体の方向をポジション型で返します。

**書　式**

AREAPOS(< エリア番号 >)

**説　明**

AREAコマンドで宣言された<エリア番号>を指定して、現在の設定値（中心位置）をポジション型で返します。

< エリア番号 > エリア番号 (0 ～ 31)

**関連項目**

AREA、SETAREA、RESETAREA、AREASIZE

**用　例**

```
DIM lp1 As Position
lp1 = AREAPOS(1)              ' 干渉チェックの行なわれる領域の中心位置と立方体の
                              ' 方向を lp1 に代入します。
```

# AREASIZE（関数）

**機　能**

干渉チェックの行なわれる領域を定義する、直方体の大きさ（各辺の長さ）をベクトル型で返します。

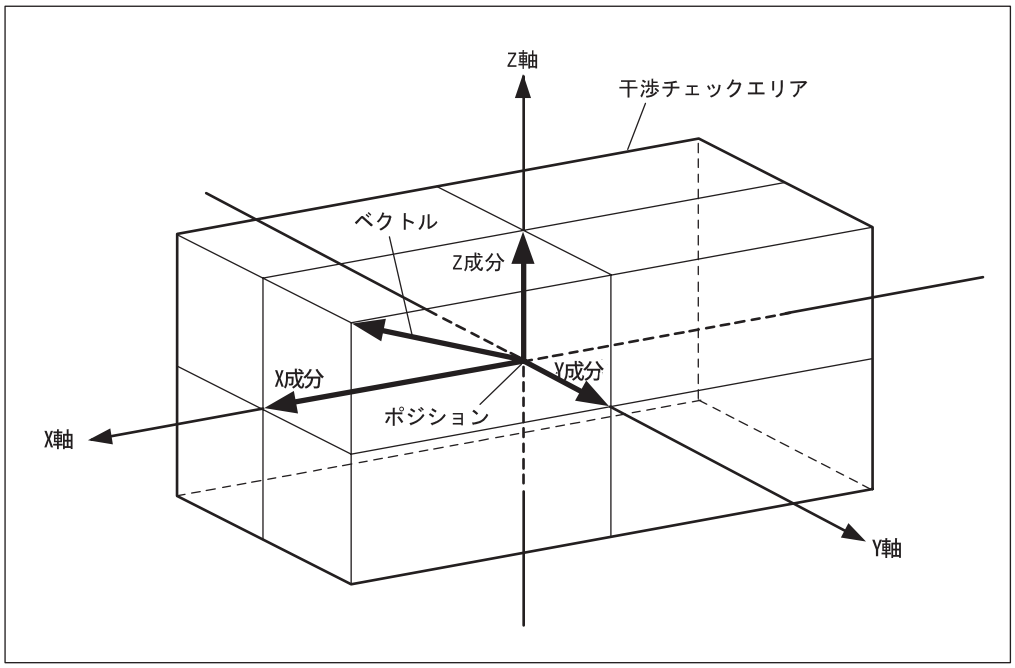
**書　式**

AREASIZE(<エリア番号>)

**説　明**

AREA コマンドで宣言された＜エリア番号＞を指定することで、その領域を定義する直方体の大きさ（各辺の長さ）をベクトル型で返します。

直方体の各辺の長さは、ベクトルの X、Y、Z 各成分の長さの倍です。




**関連項目**

AREA、SETAREA、RESETAREA、AREAPOS

**用　例**

```
DIM lv1 As Vector
lv1 = AREASIZE(1)          ’干渉チェックの行なわれる領域を定義する直方体の大
                           ’きさを lv1 に代入します。
```

# TOOLPOS（関数）

**機　能**

ツール座標系をポジション型で返します。

**書　式**

TOOLPOS(< ツール座標系番号 >)

**説　明**

TOOL コマンドで宣言された < ツール座標系番号 > を指定することで、そのツール座標系の設定値をポジション型で返します。この場合、ポジションデータの FIG 成分には、-1（不定）が入ります。

< ツール座標系番号 > ツール座標系番号 (1 ～ 63)

**関連項目**

TOOL、CHANGETOOL

**用　例**

```
DIM lp1 As Position
lp1 = TOOLPOS(1)                ' ツール座標系を lp1 に代入します。
```

# WORKPOS（関数）

**機　能**

ユーザ座標系をポジション型で返します。

**書　式**

WORKPOS(< ユーザ座標系番号 >)

**説　明**

WORK コマンドで宣言された < ユーザ座標系番号 > を指定することで、そのワーク座標系の設定値をポジション型で返します。この場合、ポジションデータの FIG 成分には、-1（不定）が入ります。

< ユーザ座標系番号 > ユーザ座標系番号 (1 ～ 7)

**関連項目**

WORK、CHANGEWORK

**用　例**

```
DIM lp1 As Position
lp1 = WORKPOS(1)                ' ユーザ座標系を lp1 に代入します。
```

# WORKATTRIBUTE（関数）【Ver.3.2 以降】

**機　能**

指定したワーク座標系の属性値を取得します

**書　式**

WORKATTRIBUTE(< ワーク座標系番号 >)

**説　明**

< ワーク座標系番号 > で指定したワーク座標の属性値（0：標準、1：固定工具（外部 TCP））を取得します。（Ver.3.2 以降）

外部 TCP 機能については、操作ガイドの「外部 TCP 機能 ( ワーク座標原点を中心に回転動作する )」を参照してください。

**関連項目**

WORK、CHANGEWORK

**用　例**

```
I55=WORKATTRIBUTE(5)            ' ワーク座標 5 の属性値を I55 に取得します。
```

# 15.15 文字列関数

## ASC（関数）

**機　能**

文字コードへ変換します。

**書　式**

ASC(< 文字列 >)

**説　明**

< 文字列 > の最初の文字をキャラクタコードに変換します。

**注意**

この関数は文字列の最初の 1 バイトの値を返すだけです。したがって、最初の文字が漢字であっても、Shift-JIS コードの頭の 1 バイトが返されます。

**関連項目**

CHR$、文字コード表（付録 1）

**用　例**

```
DIM li1 As Integer
li1 = ASC( "ABCDEFGH" )        'A のキャラクタコード 65(&H41) を li1 に代入します。
```

# BIN$（関数）【SLIM 準拠】

**機　能**

数式の値を 2 進数の文字列へ変換します。

**書　式**

BIN$(< 数式 >)

**説　明**

< 数式 > に指定した値を 2 進数に変換し、その値を文字列に変換します。

**関連項目**

CHR$、HEX$、STR$、VAL

**用　例**

```
DIM li1 As Integer
DEFSTR ls1, ls2
ls1 = BIN$(20)              '20 を 2 進数に変換し ls1 に代入します。
ls2 = BIN$(li1)             'li1 を 2 進数に変換し ls2 に代入します。
```

# CHR$（関数）【SLIM 準拠】

**機　能**

ASCII コードを文字に変換します。

**書　式**

CHR$(< 数式 >)

**説　明**

< 数式 > に指定した値のキャラクタコードを持つ文字を得ます。

**関連項目**

BIN$、HEX$、STR$、VAL、文字コード表 ( 付録 1)

**用　例**

```
DEFSTR ls1, ls2
ls1 = CHR$(49)              '49 のキャラクタコードを持つ文字を ls1 に代入します。
ls2 = CHR$(&H4E)            '&H4E のキャラクタコードを持つ文字を ls2 に代入します。
```

# SPRINTF$（関数）

機　能

式を指定したフォーマットに変換し、文字列として返します。

書　式

SPRINTF$（< フォーマット >,< 式 >）

説　明

< フォーマット > には、そのまま出力される文字列と、< 式 > の内容を変換して（文字列の一部として）出力するための変換指定文字列が含まれます。

変換指定文字列の先頭は % で、以下に文字（列）が続きます。

変換指定文字列の意味を修飾する 0 個以上のフラグ（順不同）。

- 文字列を書き出すフィールドの最小幅（オプション）。変換された値の文字数がフィールドの最小幅より小さい場合は、フィールドの最小幅を満たすように、左側（下記の左揃えフラグを指定した場合は右側）が空白（デフォルト）で埋められます。フィールドの幅の値は、アスタリスク (*) (後述) または整数値です。
- 精度（オプション）。精度によって指定されるのは、d、i、o、x および X 変換指定記号による変換で表示される最小桁数、e、E および f 変換指定記号による変換で表示される小数点以下の桁数、g および G 変換指定記号による変換で表示される最大有効桁数、そして s 変換指定記号による変換で文字列から書き込まれる最大文字数です。精度の値は、ピリオド（.）と、それに続くアスタリスク（*）（後述）または整数値（オプション）です。ピリオドのみを指定すると、精度は 0 に設定されます。精度を他の変換指定記号と一緒に指定した場合の動作は未定義です。
- 変換指定記号
  前述のように、フィールド幅と精度にはアスタリスク（*）を指定できます。この場合には、整数型の引数によって幅または精度の値が指定されます。フィールド幅または精度、あるいはその両方の値を指定する引数は、変換される引数（存在する場合のみ）の前に（フィールド幅を精度より前に）指定しなければなりません。フィールドの幅に負の値を指定すると、- 変換指定記号の後に正のフィールド幅を指定したのと同じになります。精度に負の値を指定すると、精度を省略したのと同じになります。
  フラグの意味は次のとおりです。

  -　：変換結果をフィールド内で左揃えにします（このフラグを指定しなかった場合は右揃えになります）。

  +　：符号付きの変換では、常に数字の先頭に + または - の符号が付きます（このフラグを指定しなかった場合は、負の値の場合にのみ - の符号が付きます）。

  スペース文字　：符号付き変換で先頭文字が符号でない場合、もしくは符号付き変換によって何も文字を生じなかった場合には、結果の先頭にスペース文字が付きます。スペース文字と + フラグを同時に指定すると、スペース文字は無視されます。

  #　：変換は「置換形式」によって行なわれます。o 変換指定記号による変換では、結果の先頭桁が 0 になるように精度が変更されます。x（または X）変換指定記号による変換では、0 以外の結果の先頭には「0x」（または 0X）が付けられます。e、E、f、g または G 変換指定記号による変換では、必ず結果に（小数位の数字がなくても）小数点文字が付けられます。（通常、これらの変換では、小数位の数字がある場合にのみ小数点文字が付けられます。）g および G 変換指定記号による変換では、末尾の 0 が削除されません。他の変換での動作は未定義です。

  0　：d、i、o、x、X、e、E、f、g および G 変換指定記号による変換では、先頭（符号または底数の後）に、0 を埋め込むことで最小フィールド幅を確保します。スペース文字による埋め込みは行なわれません。0 フラグと - フラグを同時に指定した場合、0 フラグは無視されます。d、i、o、x、および X の変換指定記号によって、精度の変換を指示してある場合は、0 フラグは無視されます。その他の変換指定記号とともに用いられた場合の動作については規定しません。

変換指定記号の意味は次のとおりです。

d、i　：整数型引数の値を「[-］dddd」の形式で符号付き 10 進数の文字列に変換します。指定されている精度により、表示される最小桁数が決定します。変換後の値が最小桁数に満たない場合には、先頭が 0 で埋められます。デフォルトの精度は 1 です。精度 0 で 0 の値を変換すると、文字は出力しません。

o、x、X　：整数型の引数を、「dddd」の形式で符号なしの 8 進数（o)、または 16 進数（x または X）の文字列に変換します。x 変換指定記号では小文字（abcdef)、X 変換指定記号では大文字（ABCDEF）が使用されます。指定されている精度により、表示される最小桁数が決定します。変換後の値が最小桁数に満たない場合には、先頭が 0 で埋められます。デフォルトの精度は 1 です。精度 0 で 0 の値を変換すると、文字は出力しません。

f　：実数型の引数を「[-］ddd.ddd」の形式で 10 進数表示の文字列に変換します。指定されている精度により、小数点以下の桁数が決定されます。精度が指定されていなければ、小数点以下の桁数は 6 桁になります。精度が 0 で＃フラグが指定されていないときは、小数点以下の文字は表示されません。小数点以下の文字が表示される場合には、最低でもその前の 1 桁が表示されます。値は、適切な桁数に丸められます。

e、E　：実数型の引数を「[-］d.ddde+/-dd」の形式の文字列に変換します。小数点文字の前には 1 桁（引数が 0 でなければ 0 以外）が表示されます。小数点以下の桁数は、精度（precision）によって決定されます。精度が指定されていなければ、小数点以下は 6 桁になります。精度が 0 で＃フラグが指定されていないときは、小数点以下の文字は出力されません。値は、適切な桁数に丸められます。E 変換指定記号を使用すると、指数が e ではなく E を用いて表示されます。指数は常に 2 桁以上となります。値が 0 であれば、指数部も 0 になります。

g、G　：実数型の引数を f または e(G の場合は E) の形式に変換します。精度により、有効桁数が決まります。精度が 0 である場合、有効桁数は 1 になります。使用される形式は、変換される値によって決まります。e（または E）の形式の変換は変換結果の指数部が -4 より小さいか、もしくは精度と等しいか精度より大きい場合のみ用います。小数点以下の末尾にある 0 は削除されます。小数点文字は小数点以下の数字がある場合にのみ表示されます。

c　：整数型の引数を 1 バイトの ASCII 文字に変換し、変換後の文字を出力します。

S　：システム予約

%　：% の文字を出力します。引数は変換されません。変換の完全な指定は %% です。

変換指定が正しくない場合の動作については規定していません。

変換結果がフィールド幅より大きい場合には、変換結果が切り詰められることはあっても、変換結果に合わせてフィールドが拡張されることはありません。

**関連項目**

HEX$、STR$

**用　例**

```
S1 = SPRINTF$("% d",123)          'S1 に " 123" が代入されます。
S2 = SPRINTF$("%-6.3f", 1.23)     'S2 に "1.230 " が代入されます。
S3 = SPRINTF$("%e",123.456#)      'S3 に "1.234560e+002" が代入されます。
S4 = SPRINTF$("%g",12345678#)     'S4 に "1.23457e+007" が代入されます。
S5 = SPRINTF$("%#x",128)          'S5 に "0x80" が代入されます。
```

# HEX$（関数）【SLIM 準拠】

**機　能**

10 進数から 16 進数へ変換した値を文字列として得ます。

**書　式**

HEX$(< 数式 >)

**説　明**

< 数式 > に指定した値を 10 進数から 16 進数に変換し、その値を文字列に変換します。

**補足説明**

**引数が整数でない場合、変換の前にその値を超えない整数に丸められます。**

**関連項目**

BIN$、CHR$、STR$、VAL

**用　例**

```
DIM li1 As Integer
DEFSTR ls1, ls2
ls1 = HEX$(20)              '20 を 16 進数に変換し ls1 に代入します。
ls2 = HEX$(li1)             'li1 を 16 進数に変換し ls2 に代入します。
```

# LEFT$（関数）【SLIM 準拠】

**機　能**

左部分文字列を抽出します。

**書　式**

LEFT$(< 文字列 >,< 桁数 >)

**説　明**

< 文字列 > の左側から < 桁数 > 分の文字列を抽出します。

文字列に NULL 値が含まれる場合は、NULL 値を返します。

< 桁数 > に文字列の長さ以上の値を指定した場合は、文字列全体を返します。

**注意**

**< 桁数 > はバイト数として扱われます。したがって、漢字コードは、2 バイトと数えてください。**

**関連項目**

LEN、MID$、RIGHT$

**用　例**

```
DIM li1 As Integer
DEFSTR ls1, ls2
ls1 = LEFT$( "abcdefg", 3 )  ' 文字列 "abcdefg" の左から 3 文字の文字列 "abc" を ls1
                             ' に代入します。
ls2 = LEFT$( ls1, li1 )      'ls1 の左から li1 の文字数分の値の文字列を ls2 に代
                             ' 入します。
```

# LEN（関数）【SLIM 準拠】

**機　能**

文字列の長さをバイト数で得ます。

**書　式**

LEN(< 文字列 >)

**説　明**

< 文字列 > で指定される、文字列の長さの合計バイト数を得ます。

**関連項目**

MID$、LEFT$、RIGHT$

**用　例**

```
DEFINT li1, li2
DIM ls1 As String
li1 = LEN( "abcdefg" )        ' 文字列 "abcdefg" の文字数 7 を li1 に代入します。
li2 = LEN( ls1 )              'ls1 のバイト数を li2 に代入します。
```

# MID$（関数）【SLIM 準拠】

**機　能**

文字列から指定した文字数分の文字列を取り出します。

**書　式**

MID$(< 文字列 >,< 開始位置 >[,< 桁数 >])

**説　明**

< 文字列 > の < 開始位置 > の文字から < 桁数 > 分の文字列を抽出します。

文字列に NULL 値が含まれる場合は、NULL 値を返します。

< 桁数 > に文字列の長さ以上の値を指定した場合は、文字列全体を返します。

< 桁数 > を省略した場合、または文字列内に指定した桁数より少ない文字数しかない場合は、開始位置から後ろのすべての文字が返されます。

< 開始位置 > に文字列の長さ以上の値を指定した場合、文字列全体を返します。

**注意**

**< 桁数 > はバイト数として扱われます。したがって、漢字コードは、2 バイトと数えてください。**

**関連項目**

LEN、LEFT$、RIGHT$

**用　例**

```
DEFSTR ls1, ls2
ls1 = MID$( "abcdefg", 2, 3 )     ' 文字列 "bcd" を ls1 に代入します。
ls2 = MID$( ls1, 2, 2 )           'ls1 の 2 番目から 2 桁までを ls2 に代入します。
```

# ORD（関数）【SLIM 準拠】

**機　能**

文字コードへ変換します。

**書　式**

ORD(< 文字列 >)

**説　明**

< 文字列 > の最初の文字をキャラクタコードに変換します。

**注意**

**この関数は文字列の最初の 1 バイトの値を返すだけです。したがって、最初の文字が漢字であっても、Shift-JIS コードの頭の 1 バイトが返されます。また、文字列が " " の場合、0 を返します。**

**関連項目**

CHR$、文字コード表 ( 付録 1)

**用　例**

```
DIM li1 As Integer
li1 = ORD( "ABCDEFGH" )        'A のキャラクタコード 65(&H41) を li1 に代入します。
```

# RIGHT$（関数）【SLIM 準拠】

**機　能**

文字列の右部分を抽出します。

**書　式**

RIGHT$(< 文字列 >,< 桁数 >)

**説　明**

< 文字列 > の右側から < 桁数 > 分の文字列を抽出します。

文字列に NULL 値が含まれる場合は、NULL 値を返します。

< 桁数 > に文字列の長さ以上の値を指定した場合は、文字列全体を返します。

**注意**

**< 桁数 > はバイト数として扱われます。したがって、漢字コードは、2 バイトと数えてください。**

**関連項目**

LEN、MID$、LEFT$

**用　例**

```
DIM li1 As Integer
DEFSTR ls1, ls2
ls1 = RIGHT$( "abcdefg", 3 )      ' 文字列 "abcdefg" の右から 3 文字の文字列 "efg" を ls1
                                  ' に代入します。
ls2 = RIGHT$( ls1, li1 )          'ls1 の右から li1 の値の文字数分の文字列を ls2 に代
                                  ' 入します。
```

# STRPOS（関数）【SLIM 準拠】

**機　能**

文字列の位置を得ます。

**書　式**

STRPOS(< 文字列 1>,< 文字列 2>)

**説　明**

< 文字列 1> 内での < 文字列 2> の位置を得ます。

順序は左から 1、2… と数えます。

見つからない場合は 0 を返します。

**用　例**

```
DIM li1 As Integer
li1 = STRPOS("abcdefg", "bc")      '"abcdefg" 内での "bc" の位置 (2) を li1 に代入します。
```

# STR$（関数）【SLIM 準拠】

**機　能**

数値から文字列に変換します。

**書　式**

STR$(< 数式 >)

**説　明**

< 数式 > に指定した値を文字列に変換します。

**補足説明**

数値を文字列に変換すると、戻り値の先頭に符号を表示するためのスペースが常に確保されます。数値が正の場合は、Str 関数の戻り値の先頭にスペースが挿入されます。このスペースはプラス記号を意味します。Str 関数は、ピリオド (.) だけを有効な小数点記号として認識します。

**関連項目**

BIN$、CHR$、HEX$、VAL

**用　例**

```
DIM li1 As Integer
DEFSTR ls1, ls2
ls1 = STR$(20)              '20 を文字列に変換し ls1 に代入します。
ls2 = STR$(li1)             'li1 を文字列に変換し ls2 に代入します。
```

# VAL（関数）【SLIM 準拠】

**機　能**

文字列から数値へ変換します。

**書　式**

VAL(< 文字列 >)

**説　明**

< 文字列 > に指定した文字列を数値に変換します。

< 文字列 > の最初の文字が +、-、&、または数値でなければ、VAL の値は 0 になります。

**補足説明**

文字列中に数字以外の文字が見つかると、Val 関数は読み込みを中止します。円記号 (\) やカンマ (,) など、通常は数値の一部とみなされる記号や文字も、Val 関数は数値として解釈しません。ただし、Val 関数は基数を示すプリフィックス &H (16 進数 )、&B (2 進数 ) は認識します。引数の文字列中に含まれるスペース、タブ、ラインフィードは無視されます。

**関連項目**

BIN$、CHR$、HEX$、STR$

**用　例**

```
DEFINT li1, li2, li3
li1 = VAL("&B100")          '"&B100" を数値 (10 進数で 4) に変換し li1 に代入します。
li2 = VAL("&H20")           '"&H20" を数値 (10 進数で 32) に変換し li2 に代入します。
li3 = VAL("-30")            '"-30" を数値 (10 進数で -30) に変換し li3 に代入します。
```

# 第 16 章
# 定数

この章では、PAC に用意された定数について説明します。

# 第 16 章　定数

## 16.1　組み込み定数

### OFF（組み込み定数）

**機　能**

OFF(0) の値を与えます。

**書　式**

OFF

**説　明**

式に対して、OFF(0) の値を与えます。

**関連項目**

ON

**用　例**

```
IF I0 = TRUE THEN              ' ブール値の真 (1) の値を与えます。
  I1 = ON                      ' 整数型変数に ON(1) の値を代入します。
ELSEIF I0 = FALSE THEN         ' ブール値の偽 (0) の値を与えます。
  I1 = OFF                     ' 整数型変数に OFF(0) の値を代入します。
ELSE
  D1 = PI                      ' 実数型変数に π を代入します。
ENDIF
```

# ON（組み込み定数）

**機　能**

ON(1) の値を与えます。

**書　式**

ON

**説　明**

式に対して、ON(1) の値を与えます。

**関連項目**

OFF

**用　例**

```
IF I0 = TRUE THEN              ' ブール値の真 (1) の値を与えます。
  I1 = ON                      ' 整数型変数に ON(1) の値を代入します。
ELSEIF I0 = FALSE THEN         ' ブール値の偽 (0) の値を与えます。
  I1 = OFF                     ' 整数型変数に OFF(0) の値を代入します。
ELSE
  D1 = PI                      ' 実数型変数に π を代入します。
ENDIF
```

# PI（組み込み定数）

**機　能**

π の値を与えます。

**書　式**

PI

**説　明**

倍精度型で π の値を返します。

**用　例**

```
IF I0 = TRUE THEN              ' ブール値の真 (1) の値を与えます。
  I1 = ON                      ' 整数型変数に ON(1) の値を代入します。
ELSEIF I0 = FALSE THEN         ' ブール値の偽 (0) の値を与えます。
  I1 = OFF                     ' 整数型変数に OFF(0) の値を代入します。
ELSE
  D1 = PI                      ' 実数型変数に π を代入します。
ENDIF
```

# FALSE（組み込み定数）

**機　能**

ブール値の偽 (0) の値を与えます。

**書　式**

FALSE

**説　明**

式に対して、ブール値の偽 (0) の値を与えます。

**関連項目**

TRUE

**用　例**

```
IF I0 = TRUE THEN              ' ブール値の真 (1) の値を与えます。
  I1 = ON                      ' 整数型変数に ON(1) の値を代入します。
ELSEIF I0 = FALSE THEN         ' ブール値の偽 (0) の値を与えます。
  I1 = OFF                     ' 整数型変数に OFF(0) の値を代入します。
ELSE
  D1 = PI                      ' 実数型変数に π を代入します。
ENDIF
```

# TRUE（組み込み定数）

**機　能**

ブール値の真 (1) の値を与えます。

**書　式**

TRUE

**説　明**

式に対して、ブール値の真 (1) の値を与えます。

**関連項目**

FALSE

**用　例**

```
IF I0 = TRUE THEN              ' ブール値の真 (1) の値を与えます。
  I1 = ON                      ' 整数型変数に ON(1) の値を代入します。
ELSEIF I0 = FALSE THEN         ' ブール値の偽 (0) の値を与えます。
  I1 = OFF                     ' 整数型変数に OFF(0) の値を代入します。
ELSE
  D1 = PI                      ' 実数型変数に π を代入します。
ENDIF
```

# 第 17 章
# 時刻／日付制御

この章では、日付や時刻、経過時間を知るためのコマンドと、時刻による割り込み制御のためのコマンドについて説明します。

# 第 17 章　時刻／日付制御

## 17.1　時刻／日付

### DATE$（システム変数）【SLIM 準拠】

**機　能**

現在の日付を得ます。

**書　式**

DATE$

**説　明**

現在の日付が "yyyy/mm/dd"（年 / 月 / 日）の形で格納されています。

**関連項目**

TIME$

**用　例**

```
DIM ls1 As String
ls1 = DATE$                    ’現在の日付を ls1 に代入します。
```

# TIME$（システム変数）【SLIM 準拠】

**機　能**

現在の時刻を得ます。

**書　式**

TIME$

**説　明**

現在の時刻が "hh:mm:ss"（時：分：秒）の形で格納されています。
時刻表示は 24 時制です。

**関連項目**

DATE$

**用　例**

```
DIM ls1 As String
ls1 = TIME$                    ’現在の時刻を ls1 に代入します。
```

# TIMER（システム変数）【SLIM 準拠】

**機　能**

経過時間を得ます。

**書　式**

TIMER

**説　明**

コントローラの電源を入れた時点を基準 (0) としてミリ秒で計った経過時間を得ます。

**注意**

**経過時間が 1431655750 ミリ秒を超える毎に、0 にリセットされます。**

**用　例**

```
DEFINT li1, li2, li3
li1 = TIMER                    ' 基準時間からの経過時間を li1 に代入します。
li2 = TIMER + li3              ' 基準時間からの経過時間に li3 を足した値を li2 に代
                               ' 入します。
```

# 第 18 章
# エラー制御

エラーが発生した場合に、エラーの内容を調べたり、復帰するための手順をプログラムすることができます。この章では、エラーに対処するためのコマンドを説明します。

# 第 18 章　エラー制御

## 18.1　エラー情報

## ERRMSG$（関数）

**機　能**

エラーメッセージを与えます。

**書　式**

ERRMSG$(< 数式 >)

**説　明**

< 数式 > で指定されるエラー番号に対応したエラーメッセージを与えます。

**用　例**

```
DIM ls1 As String
ls1 = ERRMSG$(&H600C)          ' エラー番号 &H600C のエラーメッセージを ls1
                               ' に代入します。
```

## SETERR（ステートメント）【Ver.1.98 以降】

**機　能**

ユーザ定義エラーを I 型変数領域にセットします。

**書　式**

SETERR　<. エラーコード >

**説　明**

ユーザ定義エラーをエラー格納機能で宣言された I 型変数領域にセットし、ポインタを加算します。

**関連項目**

CLRERR、GETERRLVL、GETERR

**注意事項**

エラー格納機能が設定されている必要があります。またエラー格納機能で宣言された I 型変数領域は命令、TP 操作によって変更可能ですのでご注意ください。

**用　例**

```
PROGRAM  PRO1
------------
SETERR  100                    ' リングバッファにエラーコード "100" を書き込む。
------------
END
```

# GETERR（関数）【Ver.1.98 以降】

**機　能**

エラー格納機能で宣言されたバッファからエラーコードを取得します。

**書　式**

GETERR(< 数式 >)

**説　明**

エラー格納機能で宣言されたバッファからエラーコードを取得します。最新のエラーを取得する場合は引数に "0" をセットします。

**関連項目**

CLRERR、GETERRLVL、SETERR

**注意事項**

エラー格納機能が設定されている必要があります。またエラー格納機能で宣言された I 型変数領域は命令、TP 操作によって変更可能ですのでご注意ください。

またこのコマンドは、エラー格納機能が設定されていない場合、通常のエラーログからエラーコードを取り出します。

**用　例**

```
PROGRAM  PRO1
------------
I1 = GETERR(0)                  ’最新のエラーをリングバッファから取り出す。
------------
END
```

# CLRERR（ステートメント）【Ver.1.98 以降】

**機　能**

エラーをクリアします。

**書　式**

CLRERR

**説　明**

現在発生しているエラーをクリアします。

**関連項目**

ERR、SETERR、GETERR

**注意事項**

特権タスクからのみ実行可能です。

**用　例**

```
PROGRAM  TSR1
------------
I1 = SYSSTATE                    ’システム状態取得
IF  (I1  AND  &H0082)  THEN      ’異常、または警告発生ならば
CLRERR                           ’エラークリア
END IF
------------
END
```

# GETERRLVL（関数）【Ver.1.98 以降】

**機　能**

エラーコードのレベルを与えます。

**書　式**

GETERRLVL(< 数式 >)

**説　明**

< 数式 > で指定されるエラーコードに対応したエラーレベルを与えます。

**関連項目**

CLRERR、SETERR、GETERR

**注意事項**

該当するエラーがない場合は "-1" を返します。

**用　例**

```
PROGRAM  PRO1
------------
I1 = GETERRLVL(&H6001)        ’エラーコード "6001" のエラーレベルを得る。
------------
END
```

# 第 19 章
# システム情報

この章では、ロボットのシステム情報を得るためのコマンドについて説明します。

# 第 19 章　システム情報

## 19.1　システム

## GETENV（関数）

**機　能**

システムの環境設定値を取得します。

**書　式**

GETENV(< テーブル番号 >,< 要素番号 >)

**説　明**

< テーブル番号 > と < 要素番号 > で指定されたシステムの環境設定値を取得します。環境設定値の項目は、付録 3「環境設定値」を参照してください

**関連項目**

VER$、環境設定値（付録 3）

**用　例**

```
DIM li1 As Integer
li1 = GETENV(9, 1)              '9, 1 番の環境設定値の内容を li1 に代入します。
```

# LETENV（ステートメント）

**機　能**

システムの環境設定値を設定します。

**書　式**

LETENV < テーブル番号 >,< 要素番号 >,< 設定値 >

**説　明**

< テーブル番号 > と < 要素番号 > で指定された、システムの環境設定値を < 設定値 > に設定します。
環境設定値の項目は、付録 3「環境設定値」を参照してください。
設定不可能な要素番号や設定値を指定すると実行時エラーが発生します。

**補足説明**

設定可能か不可能かを見分けるもっと簡単な方法は、WINCAPS Ⅲの [ パラメータ編集 ] 画面でプロパティの欄が＊印になっていない項目が設定可能な要素です。ただし、パスワードのレベルは <programmer> にしてログインしてください。

**注意**

一般に環境設定値を変更した場合は、コントローラを起動し直す必要があります。

**関連項目**

VER$、GETENV

**用　例**

```
LETENV 6, 22, 5600          '6, 22 番の環境設定値に 5600 を設定します。
```

# VER$（関数）

**機　能**

各モジュールのバージョンを取得します。

**書　式**

VER$(< 数式 >)

**説　明**

< 数式 > の値に対応するモジュールのバージョンが文字列で格納されます。< 数式 > の値と対応するモジュールについては、付録 7 のバージョン対応表を参照してください。

**関連項目**

GETENV、バージョン対応表（付録 7）

**用　例**

```
DEFSTR ls1, ls2, ls3
DIM li1 As Integer
ls1 = VER$(1)                ' メインボードのバージョンを ls1 に代入します。
ls2 = VER$(2)                'DIO ボードのバージョンを ls2 に代入します。
ls3 = VER$(li1)              'li1 の値に対応するバージョン情報を ls3 に代入します。
```

この例では、ls1, ls2 には次のような形式で文字列として格納されています。（日付はバージョンのできた日付です。）

| | | |
|---|---|---|
| ls1 | 1.000 | 07/30/1998 |
| ls2 | V1.00 | 8/03/1998 |

# GETLANGUAGE（関数）【RC7_Ver.2.2 以降】

**機　能**

現在の言語設定を取得します。

**書　式**

GETLANGUAGE

**説　明**

現在の言語設定を取得します。

0：日本語　1：英語　2：ドイツ語　3：韓国語　4：中国語

を表します。

**用　例**

```
defint li
li=GETLANGUAGE
```

## 19.2　ログ

# STARTLOG（ステートメント）

**機　能**

サーボ制御ログの記録を開始します。

**書　式**

STARTLOG

**説　明**

プログラムでサーボ制御ログの記録を開始するのに、STARTLOG コマンドを使います。

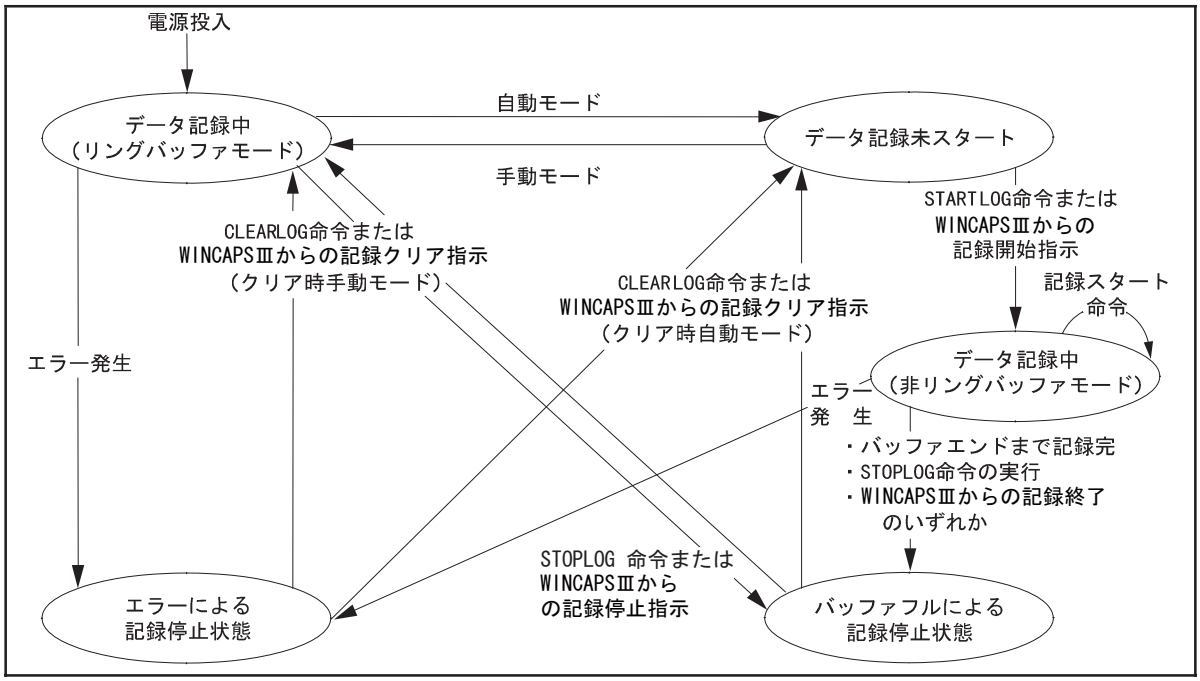
確実にサーボ制御ログを記録するために、STARTLOG コマンドを実行する前に、あらかじめ CLEARLOG コマンドを実行するか、WINCAPS Ⅲでサーボログクリア操作を行なってください。すでに、サーボ制御ログの記録を開始しているときには、STARTLOG コマンドを実行しても、記録できません。

手動モードとティーチチェックモードのときには、自動的にサーボ制御ログの記録を開始します。

自動モードでは、プログラムの中で記録開始指示があるか、または WINCAPS Ⅲでの記録開始操作によって、サーボ制御ログの記録を開始します。

サーボ制御ログの記録を停止するのは、エラーが発生したときと、バッファがいっぱいになった（10 秒間）ときです。手動モード時は、記録停止の指示がされるまで最新の 10 秒間分のログを保持します。

**ログ動作の状態遷移図**




**関連項目**

CLEARLOG、STOPLOG

**用　例**

```
STARTLOG                    ’サーボ制御ログの記録を開始します。
```

# CLEARLOG（ステートメント）

**機　能**

サーボ制御ログの記録を初期化します。

**書　式**

CLEARLOG

**説　明**

プログラムの中で、サーボ制御ログの初期化を指示するために、CLEARLOG コマンドを使用します。

サーボログ記録がバッファがいっぱいになるか、あるいはエラーの発生で停止した場合、記録を再開する前に、ログの内容をクリア（初期化）する必要があります。P19-4「ログ動作の状態遷移図」を参照してください。

確実にサーボ制御ログを記録するために、STARTLOG コマンドを実行する前に、CLEARLOG コマンドを実行してください。記録をすでに開始した状態では、STARTLOG コマンドを実行しても、記録できません。

**関連項目**

STARTLOG、STOPLOG

**用　例**

```
CLEARLOG                    ’サーボ制御ログの記録を初期化します。
```

# STOPLOG（ステートメント）

**機　能**

サーボ制御ログの記録を停止します。

**書　式**

STOPLOG

**説　明**

プログラムの中で、サーボ制御ログの停止を指示するために、STOPLOG コマンドを使用します。

自動モードでは、記録開始後バッファがいっぱいになると、自動的にログ記録を停止しますので、STOPLOG コマンドをプログラム内で実行する必要は、通常はありません。

**関連項目**

STARTLOG、CLEARLOG

**用　例**

```
STOPLOG                         ’サーボ制御ログの記録を停止します。
```

# STARTTRACELOG（ステートメント）【Ver.2.7 以降】

**機　能**

トレースログの記録を開始します。

**書　式**

STARTTRACELOG

**説　明**

プログラム内でトレースログの記録の開始を指定します。
トレースログは WINCAPS Ⅲでログデータを受信して閲覧します。
トレースログ（マルチ）とシングルトレースログを同時に記録します。（Ver.2.7 以降）

**関連項目**

STOPTRACELOG、CLEARTRACELOG

**用　例**

```
CLEARTRACELOG                ’トレースログデータをクリアします。
STARTTRACELOG                ’トレースログの記録を開始します。
・・・・
動作
・・・・
STOPTRACELOG                 ’トレースログの記録を終了します。
```

# STOPTRACELOG（ステートメント）【Ver.2.7 以降】

**機　能**

トレースログの記録を終了します。

**書　式**

STOPTRACELOG

**説　明**

プログラム内でトレースログの記録の終了を指定します。
トレースログは WINCAPS Ⅲでログデータを受信して閲覧します。
トレースログ（マルチ）とシングルトレースログを同時に記録停止します。（Ver.2.7 以降）

**関連項目**

STARTTRACELOG、CLEARTRACELOG

**用　例**

```
CLEARTRACELOG                ’トレースログデータをクリアします。
STARTTRACELOG                ’トレースログの記録を開始します。
・・・・
動作
・・・・
STOPTRACELOG                 ’トレースログの記録を終了します。
```

# CLEARTRACELOG（ステートメント）【Ver.2.7 以降】

**機　能**

トレースログデータをクリアします。

**書　式**

CLEARTRACELOG

**説　明**

プログラム内でトレースログデータをクリアします。
トレースログは WINCAPS Ⅲでログデータを受信して閲覧します。
トレースログ（マルチ）とシングルトレースログのデータを同時にクリアします。（Ver.2.7 以降）

**関連項目**

STARTTRACELOG、STOPTRACELOG

**用　例**

```
CLEARTRACELOG                ’トレースログデータをクリアします。
STARTTRACELOG                ’トレースログの記録を開始します。
・・・・
動作
・・・・
STOPTRACELOG                 ’トレースログの記録を終了します。
```

# STARTVARLOG（ステートメント）【Ver.2.7 以降】

**機　能**

変数ログの記録を開始します。

**書　式**

STARTVARLOG

**説　明**

プログラム内で変数ログの記録の開始を指定します。

変数ログは WINCAPS Ⅲでログデータを受信して閲覧します。

ログを記録する変数はウオッチに登録し、WINCAPS Ⅲで「ログ」欄にチェックを入れた変数のみです。（Ver.2.7 以降）

**関連項目**

STOPVARLOG、CLEARVARLOG

**用　例**

```
CLEARVARLOG                    ’変数ログデータをクリアします。
STARTVARLOG                    ’変数ログの記録を開始します。
・・・・
動作
・・・・
STOPVARLOG                     ’変数ログの記録を終了します。
```

# STOPVARLOG（ステートメント）【Ver.2.7 以降】

**機　能**

変数ログの記録を終了します。

**書　式**

STOPVARLOG

**説　明**

プログラム内で変数ログの記録の終了を指定します。

変数ログは WINCAPS Ⅲでログデータを受信して閲覧します。

ログを記録する変数はウオッチに登録し、WINCAPS Ⅲで「ログ」欄にチェックを入れた変数のみです。（Ver.2.7 以降）

**関連項目**

STARTVARLOG、CLEARVARLOG

**用　例**

```
CLEARVARLOG                    ’変数ログデータをクリアします。
STARTVARLOG                    ’変数ログの記録を開始します。
・・・・
動作
・・・・
STOPVARLOG                     ’変数ログの記録を終了します。
```

# CLEARVARLOG（ステートメント）【Ver.2.7 以降】

**機　能**

変数ログデータをクリアします。

**書　式**

CLEARVARLOG

**説　明**

プログラム内で変数ログデータをクリアします。

変数ログは WINCAPS Ⅲでログデータを受信して閲覧します。

ログを記録する変数はウオッチに登録し、WINCAPS Ⅲで「ログ」欄にチェックを入れた変数のみです。（Ver.2.7 以降）

**関連項目**

STARTVARLOG、STOPVARLOG

**用　例**

```
CLEARVARLOG                    ’変数ログデータをクリアします。
STARTVARLOG                    ’変数ログの記録を開始します。
・・・・
動作
・・・・
STOPVARLOG                     ’変数ログの記録を終了します。
```

## 19.3　動作モード

# CHGEXTMODE（ステートメント）【Ver.1.98 以降】

**機　能**

外部自動モード切替を行ないます。

**書　式**

CHGEXTMODE

**説　明**

内部自動モードから外部自動モードへの切替を行ないます。

**関連項目**

INIT、CHGINTMODE

**注意事項**

手動、ティーチチェックモードからの切替はできません。特権タスクからの実行のみ有効で、しかもユーザタスクが停止している状態で実行可能です。

**用　例**

```
PROGRAM  TSR1
------------
CHGEXTMODE                    ’外部自動モード切替
------------
CHGINTMODE                    ’内部自動モード切替
------------
END
```

# CHGINTMODE（ステートメント）【Ver.1.98 以降】

**機　能**

内部自動モード切替を行ないます。

**書　式**

CHGINTMODE

**説　明**

外部自動モードから内部自動モードへの切替を行ないます。

**関連項目**

INIT、CHGEXTMODE

**注意事項**

手動、ティーチチェックモードからの切替はできません。特権タスクからの実行のみ有効で、しかもユーザタスクが停止している状態で実行可能です。

**用　例**

```
PROGRAM  TSR1
------------
CHGEXTMODE                    ’外部自動モード切替
------------
CHGINTMODE                    ’内部自動モード切替
------------
END
```

# CUROPTMODE（ステートメント）【Ver.1.98 以降】

**機　能**

動作モードを取得します。

**書　式**

CUROPTMODE

**説　明**

現在の動作モード（手動、ティーチチェック、内部自動、外部自動）の情報を取得します。

1：手動
2：ティーチチェック
3：内部自動
4：外部自動

**用　例**

```
PROGRAM  PRO1
------------
I1 = CUROPTMODE
------------
END
```

# SYSSTATE（ステートメント）【Ver.1.98 以降】

**機　能**

コントローラのステータスを取得します。

**書　式**

SYSSTATE

**説　明**

コントローラステータスを取得します。I/O 割付の設定により有効なデータは変化します。取得可能なデータを下記に示します。

| Bit | RC5 コントローラの場合 | RC7 コントローラの場合<br>（Mini I/O 使用時） |
|---|---|---|
| 0 | ロボット運転中 | ← |
| 1 | ロボット異常 | ← |
| 2 | サーボ ON 中 | ← |
| 3 | ロボット初期化完了 (I/O 標準モード選択時 )　/<br>ロボット電源入り完了 (I/O 互換モード選択時 ) | ロボット初期化完了 |
| 4 | 自動モード | ← |
| 5 | 外部モード | ← |
| 6 | バッテリ切れ警告 | ← |
| 7 | ロボット警告 | 予約 |
| 8 | コンティニュースタート許可 | ← |
| 9 | SS モード | ← |
| 10 | ロボット停止 | ← |
| 11 | 自動運転イネーブル | ← |
| 12 ～ 15 | 予約 | ← |
| 16 | プログラムスタートリセット<br>（I/O 互換モード選択時） | 予約 |
| 17 | CAL 完了（I/O 互換モード選択時） | 予約 |
| 18 | ティーチング中（I/O 互換モード選択時） | 予約 |
| 19 | 1 サイクル完了（I/O 互換モード選択時） | 予約 |
| 20 ～ 23 | 予約 | 予約 |
| 24 | コマンド処理完了（I/O 標準モード選択時） | コマンド処理完了 |
| 25 ～ 31 | 予約 | ← |

**注意**

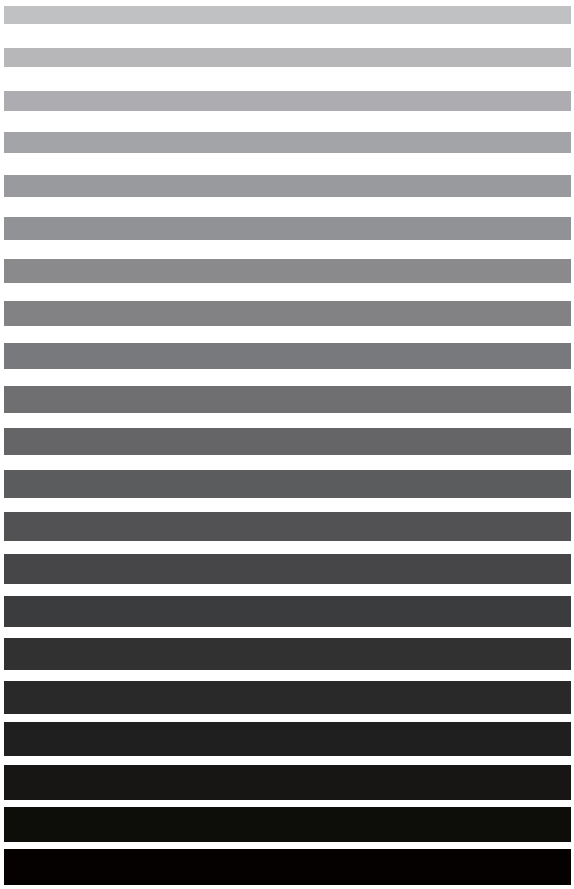
RC7 コントローラ (Mini I/O 使用時 ) についての注意

「運転準備完了」のステータスを取得するには、Bit 2 の「サーボ ON 中」と Bit 5 の「外部モード」の論理積 (AND) を状態として使用してください。

用　例

```
PROGRAM  TSR1
------------
I1 = SYSSTATE                          'システム状態取得
IF  (I1  AND  &h0082)  THEN            '異常、または警告発生ならば
CLRERR                                 'エラークリア
END  IF
------------
END
```

# 第 20 章
# プリプロセッサ



WINCAPS Ⅲでプログラムを作成すると、この章で説明するプリプロセッサコマンドを利用できます。
WINCAPS Ⅲは、コンパイルする直前に、プリプロセッサコマンドでの定義に従って、指定の文字列やファイルを自動的に置き換えてから、コンパイルを行ないます。
プリプロセッサコマンドを利用すると、ライブラリにあるプログラムを利用したり、プログラムを読みやすく記述したりすることができます。

# 第 20 章　プリプロセッサ

## 20.1　記号定数・マクロ定義

### #define（プリプロセッサステートメント）

**機　能**

プログラム中の指定した定数またはマクロ名を、指定文字列で置き換えます。

**書　式**

#define < 記号定数 > < 文字列 >

または

#define < マクロ名 ( 引数 )> < 引数を含む文字列 >

**説　明**

プログラム中の < 記号定数 > または < マクロ名 > を指定文字列で置き換えます。マクロ名の場合には、引数も含めて置き換えられます。

プログラム中の < 記号定数 > または、< マクロ名 > がダブルクォーテーション " " で囲まれた文字列は置き換えられません。

#define 文は 1 行に書かなければなりません。

< 記号定数 > と < 文字列 > の間には、必ず 1 個以上のスペース文字を置かなければなりません。

マクロ名と引数を囲む ( ) との間に、スペース文字を置いてはいけません。

< 記号定数 > と < マクロ名 > は再定義できますが、いったん #undef で無効化する必要があり、後から定義された方が有効になります。

< 記号定数 > と < マクロ名 > は、64 文字以内でなければなりません。

マクロ名は 1 つのプログラムで最大 2048 個まで使えます。また、マクロ関数の引数は数に制限がありません。

**関連項目**

#undef

**用　例**

```
#DEFINE NAME "Denso Corporation"
                              'NAME という記号定数に "Denso Corporation" を割り
                              ' 当てます。
#DEFINE mAREA(radius) PI * POW(radius, 2)
                              'mAREA(radius) をマクロ関数として宣言します。
S1 = NAME                     'S1 に "Denso Corporation" が代入されます。
D1 = mAREA(10)                'D1 に PI*POW(10,2) の計算値が代入されます。
```

# #undef（プリプロセッサステートメント）

**機　能**

#define で定義されている記号定数またはマクロの定義を無効にします。

**書　式**

#undef {< 記号定数 >|< マクロ名 >}

**説　明**

#define で定義されている記号定数またはマクロの定義を、この #undef 以後のプログラムで無効にします。引数の付いたマクロは、マクロ名だけ指定します。

**関連項目**

#define

**用　例**

```
#UNDEF NAME                    '#DEFINE 文で記号定義もしくはマクロ定義された NAME
                               ' を無効にします。
```

# #error（プリプロセッサステートメント）

**機　能**

#error コマンドを実行すると、強制的にコンパイルエラーとします。

**書　式**

#error [< メッセージ >]

**説　明**

プログラムをテストするために、故意にエラーを発生させる場合などに使用します。

#error コマンドを使っている行をコンパイルすると、エラーとなります。

< メッセージ > で定義されている内容を、エラーメッセージとして表示します。

**用　例**

```
#error "Error Occur"          ' コンパイルすると "Error Occur" メッセージが表示さ
                              ' れます。
```

# 20.2　ファイル取り込み

## #include（プリプロセッサステートメント）

**機　能**

プリプロセッサプログラムを取り込みます。

**書　式**

#include "[ パス ] ファイル名 "

#include <[ パス ] ファイル名 >

**説　明**

#include 文を置いた場所に、プリプロセッサプログラムファイルを取り込みます。ファイルのパスを省略すると " " の場合は、カレントディレクトリ、システムディレクトリの順で、ファイルを探します。< > の場合は、システムディレクトリだけからファイルを探します。パスをフルパスで指定すると、そのディレクトリだけを探します。

#include 文で指定したファイルの中に、さらに #include 文を含むことができ、8 回までのネストが可能です。

指定可能なファイルの拡張子は H と PAC があります。

**注意事項**

**フォルダ機能 [RC7_Ver.2.2 以降 ] 使用時の注意**

**注意**

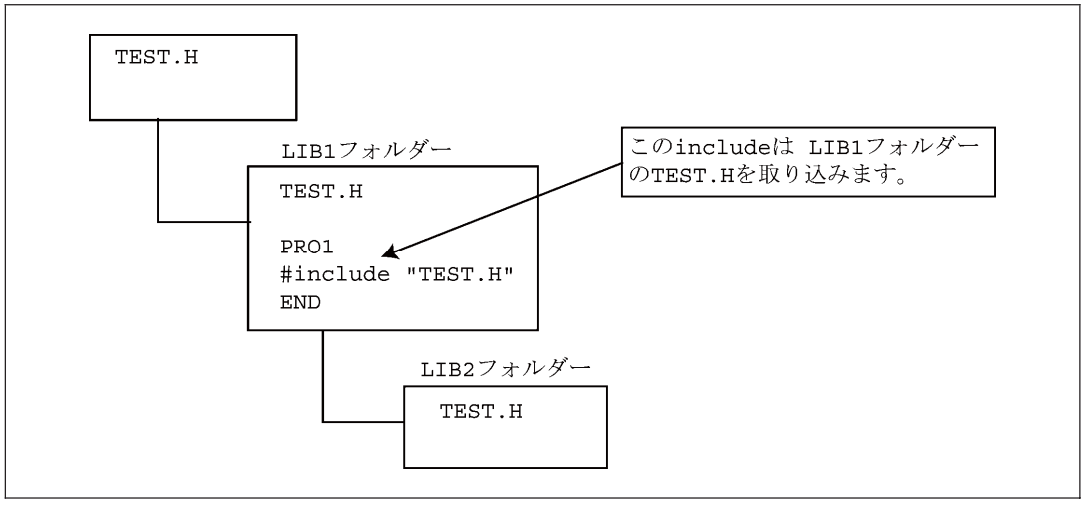
**フォルダ間の区切りを表す '¥' の表示は、表示系によって ' ＼ ' で表示されます。**
**例）「ペンダントは ' ＼ ' 表示」、「ミニペンダントは '¥' 表示」**
**これは表示上だけの違いであり、内部的には同じものとして扱われます。**

(1) #include < ファイル名 >

システムフォルダを探します。

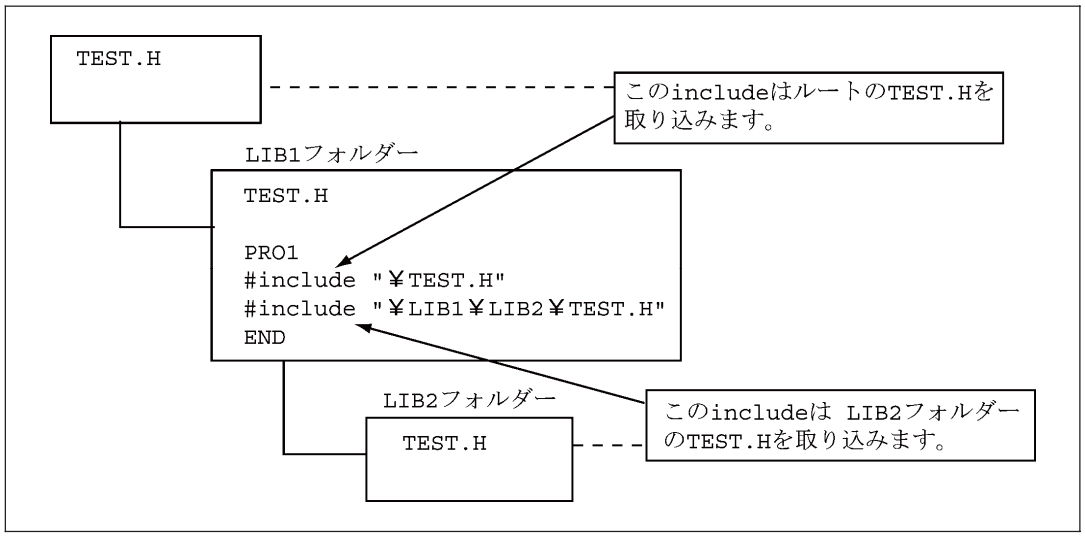
(2) #include " ファイル名 "

include 文の記載してある PAC の存在するフォルダ内を探します。



(3) #include"¥( フォルダパス )¥ ファイル名 "

ルートからのフォルダパス先のフォルダ内を探します。



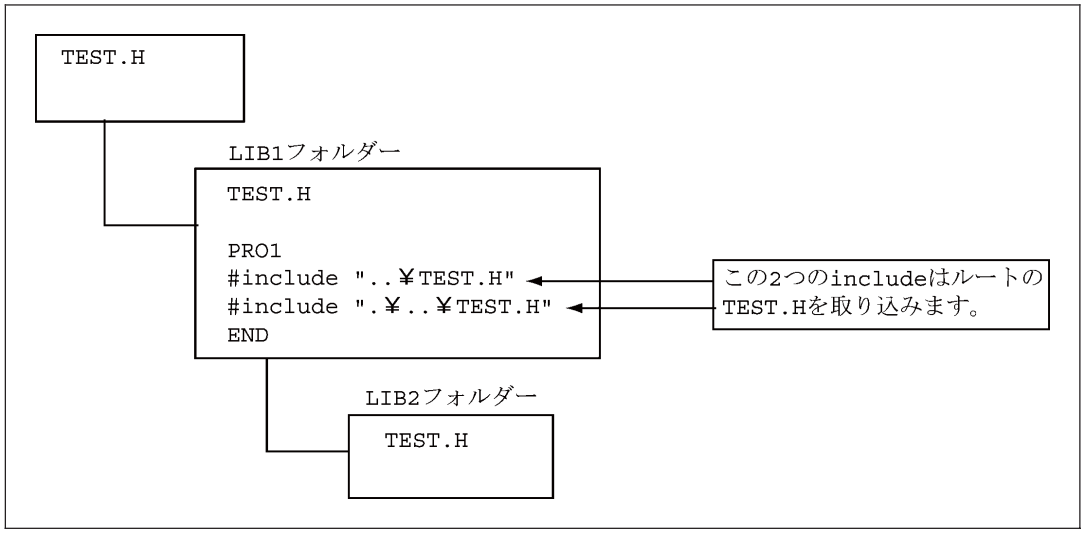

(4) #include"..¥ ファイル名 " または　#include".¥ ファイル名 "

..¥:　1 つ上のフォルダ内を探します。

.¥ :　include 文の記載してある PAC のフォルダ内を探します。

例）#include ".¥..¥TEST.H"

1 つ上の TEST.H を探します。




用　例

```
#include "samp1.h"          'samp1.h ファイルをこの行に展開します。
```

## 20.3　最適化

# #pragma optimize（プリプロセッサステートメント）

**機　能**

プログラムごとに行う最適化を指定します。

**書　式**

#pragma optimize ("< オプションリスト >", {on/off})

**説　明**

プログラムごとに行う最適化を指定します。PROGRAM 宣言文または有効なステートメント行より前に記述します。

< オプションリスト > には、次の表のパラメータを任意の個数指定します。

| パラメータ | 最適化の種類 |
|---|---|
| A | 配列検査コードの削除。 |
| C | サイクルタイム算出コードの削除。 |

**補足説明**

最適化オプションを有効に活用すると、非動作命令の実行速度を 10% ～ 20% 向上させることができます。

**注意事項**

プログラムにコマンドで記述すると、プロジェクトパラメータの設定値よりも優先されます。

**用　例**

```
#pragma optimize( "ac", ON )'配列検査コードおよびサイクルタイム算出
                            'コードの削除を有効にします。
```

# 第 21 章
# 視覚制御 ( ロボットコントローラ : オプション )

この章では、μVision ボードおよび視覚装置 μVision-21 で使用可能なコマンドのすべてがまとめてあります。

# 第 21 章　視覚制御（ロボットコントローラ : オプション）

## 21.1　視覚命令を使うにあたっての注意点（視覚ボード）

- ロボットコントローラにおいて視覚命令を使用する際は、μVision ボード（オプション）が必要です。
- 視覚命令は視覚処理権取得命令（TAKEVIS）を先立って実行する必要があります。μVision-21 では必要ありません。

（例） TAKEVIS
CAMIN 1
VISPLNOUT 0
:

- 命令実行時にエラーとなった場合、カメラ入力命令以外の命令では次に視覚命令を実行した時点でエラーメッセージが表示されます。

（例） WINDMAKE R,1,100,10,0,2
VISPRJ 1,100,100,1,128　　←この時点で " ウインドウ形状異常 " エラー
WINDDISP 1　　←次の命令実行でエラーメッセージを表示します。
:

- WINCAPS Ⅲの視覚マネージャから視覚ボードに対し通信を行っている際に、視覚処理権取得命令（TAKEVIS）を実行した場合はエラーになります。

## 21.2　従来の μVision-15 との互換性

従来の μVision-15 の機能は外部出力（パラレル I/O、RS232C）関連以外の機能は継承しています。ただし、命令の書式などに変更がありますので、従来のプログラムを参考に編集し直す場合、下記の対応表をご参照ください。

| | μVision-15 | μVision ボード | 説　明 | 掲載ページ |
|---|---|---|---|---|
| 画像入出力 | VIN | CAMIN | カメラ入力 | 21-3 |
| | AOUT | VISCAMOUT | カメラ画像の表示 | 21-7 |
| | VOUT | VISPLNOUT | 処理画面の表示 | 21-8 |
| | | VISOVERLAY | オーバレイ設定 | 21-9 |
| ウィンドウ設定 | WNDALN | WINDMAKE | ウィンドウの設定 | 21-13 |
| | WNDCIR | | | |
| | WNDDPT | | | |
| | WNDELP | | | |
| | WNDRCT | | | |
| | WNDSCT | | | |
| | HWIND | | | |
| | CLRWND | WINDCLR | ウィンドウ設定データの消去 | 21-18 |
| | WNDCPY | WINDCOPY | ウィンドウ設定データのコピー | 21-19 |
| | WDISP | WINDDISP | ウィンドウの表示 | 21-22 |

| | μVision-15 | μVision ボード | 説　明 | 掲載ページ |
|---|---|---|---|---|
| 描 画 | SCREEN | VISSCREEN | 描画条件の設定 | 21-23 |
| | CLS | VISCLS | 画面の消去 | 21-26 |
| | PUTP | VISPUTP | 点の描画 | 21-28 |
| | LINE | VISLINE | 直線（長さ、角度指定）の描画 | 21-29 |
| | LNDPT | VISPTP | 直線（2 点指定）の描画 | 21-30 |
| | RECT | VISRECT | 矩形の描画 | 21-31 |
| | CIRCLE | VISCIRCLE | 円の描画 | 21-33 |
| | ELIPSE | VISELLIPSE | 楕円の描画 | 21-34 |
| | SECT | VISSECT | 扇の描画 | 21-36 |
| | PAINT | VISRECT | 矩形の描画 | 21-31 |
| | | VISCIRCLE | 円の描画 | 21-33 |
| | | VISELLIPSE | 楕円の描画 | 21-34 |
| | CROSS | VISCROSS | 十字マークの描画 | 21-37 |
| | CRSALN | | | |
| | LOC | VISLOC | 文字表示位置指定 | 21-38 |
| | PRINT | VISDEFCHAR | 描画文字の設定 | 21-40 |
| | | VISPRINT | 文字の描画 | 21-42 |
| 画像処理 | GETP | VISGETP | 指定座標の輝度取得 | 21-44 |
| | HIST | VISHIST | ヒストグラム実行 | 21-45 |
| | HCLR | | | |
| | | VISREFHIST | ヒストグラム結果の取得 | 21-46 |
| | LEVEL | VISLEVEL | 2 値化レベル計算 | 21-47 |
| | BINA | VISBINA | 2 値化処理 | 21-49 |
| | RBINA | VISBINAR | 2 値化表示 | 21-51 |
| | AREA | VISMEASURE | 特徴 ( 面積、重心、主軸角等 ) 計算 | 21-57 |
| | BIGT | | | |
| | CENTR | | | |
| | MOMENT | | | |
| | STRT | | | |
| | PROJ | VISPROJ | 投影処理 | 21-60 |
| | EDGE | VISEDGE | エッジ計測 | 21-62 |
| サーチ機能 | CORN | SHCORNER | コーナーのサーチ | 21-91 |
| | CIRC | SHCIRCLE | 円のサーチ | 21-95 |

# 21.3　画像入出力

# CAMIN（ステートメント）

**機　能**

カメラからの映像を画像メモリ（処理画面）に格納します。

**書　式**

CAMIN＜カメラ番号＞［,＜格納メモリ番号＞[,＜テーブル番号＞]］

**説　明**

＜カメラ番号＞

　カメラ番号を指定します。(1 または 2)

＜格納メモリ番号＞

　格納メモリ（処理画面）の番号を指定します。(0 ～ 3）省略時は 0 を指定します。

＜テーブル番号＞

　格納する際のルックアップテーブル番号を指定します。(0 ～ 15）省略時は 0 を指定します。

**注意**

- **カメラが接続されていない、または故障で正しく入力できない場合、エラーになります。**
- **格納の際テーブル番号が 0 でない場合、テーブルの変更を行ないますので、画面が乱れることがありますが故障ではありません。**
- **実行後、テーブル番号は自動的に 0 番に戻ります。**
- **本命令はロボットコントローラでは μVision ボード（オプション）が必要です。**

**関連項目**

CAMMODE、CAMLEVEL、VISDEFTABLE

**用　例**

```
CAMMODE 1,0,0              ' カメラ 1 番の映像を格納メモリ 0 番にテーブル 0 番
                           ' ( カメラの映像と同じ輝度 ) で変換し格納します。
DELAY 2000                 '2 秒停止
CAMIN 1                    'CAMIN 1,0,0 と同じ結果が得られます。
DELAY 2000                 '2 秒停止
I1 = 1                     '
I2 = 0                     '
I3 = 3                     '
CAMIN I1,I2,I3             ' カメラ 1 番の映像を格納メモリ 0 番にテーブル 3 番
                           ' ( 反転 ) で変換し格納します。
VISPLNOUT 0                ' 格納メモリ 0 番の画像をモニタに出力 ( 静止画像 ) し
                           ' ます。
```

# CAMMODE（ステートメント）

**機　能**

カメラ映像を格納する際の機能を設定します。

**書　式**

CAMMODE＜カメラ番号＞,＜機能＞,＜格納方法＞

**説　明**

＜カメラ番号＞

カメラ番号を指定します。(1 または 2)

＜機能＞

カメラの機能を指定します。(0 または 1)

0：ノーマル（カメラの設定が通常の場合）

1：リセット機能（カメラの動作をリセットしてから、カメラ映像を格納します。）

0：フレーム取り込み

＜格納方法＞

格納方法を指定します。(0 または 1)

カメラの映像を 1 フレーム分格納します。垂直方向の分解能が最大になります。

1：フィールド取り込み

フィールドシャッターカメラのシャッター機能を使う場合設定します。

フィールド間の遅れ（1/60 秒）がなく、ブレのない映像が取り込めます。

ただし、垂直方向の分解能は 1/2 になります。

**注意**

- 本命令を設定しない場合、初期設定値に基づきます。
- 本命令は初期設定を変更しません。電源再立ち上げの場合設定は失われています。
- カメラの接続を確認します。カメラが正常でないと判断した場合、ステータス VISSTATUS(0) の値は -1、正常な場合 0 を返します。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

CAMIN、VISSTATUS

用　例

```
CAMMODE 1,0,0                  ' カメラ 1 番の機能をノーマル、取り込み方法をフレ
                               'ームに設定します。
I1 = VISSTATUS(0)              ' 正常の場合 I1 = 0
IF I1 = 0 THEN
 CAMIN 1                       ' カメラ 1 番の映像を格納メモリ 0 番にテーブル 0 番 ( カ
                               ' メラの映像と ' 同じ輝度 ) で変換し格納します。
 VISPLNOUT 0                   ' 格納メモリ 0 番の画像をモニタに出力 ( 静止画像 ) し
                               ' ます。
 VISLOC 10,10                  ' 表示位置設定
 VISPRINT " 取り込み正常 "     ' 画面に文字を表示します。
ELSE
 VISLOC 10,10                  ' 表示位置設定
 VISPRINT " カメラ異常 "       ' 画面に文字を表示します。
END IF
```

# CAMLEVEL（ステートメント）

**機　能**

カメラ映像の入力レベルを設定します。

**書　式**

CAMLEVEL＜カメラ番号＞,＜下限レベル＞,＜上限レベル＞

**説　明**

＜カメラ番号＞

　カメラ番号を指定します。(1 または 2)

＜下限レベル＞

　カメラ映像の取り込みレベルの下限値を設定します。(0 ～ 93)

＜上限レベル＞

　カメラ映像の取り込みレベルの上限値を設定します。(7 ～ 100)

**注意**

- 本命令は最大入力範囲の下限を 0%、上限を 100% とし入力範囲の設定を指定します。
- 設定した下限、上限レベルの範囲を 256 階調にします。
- 設定は 0 ≦ 下限値 < 上限値 ≦ 100 かつ 上限 － 下限 ≧ 7 である必要があります。
- 暗い映像や明るい映像、部分的に輝度の分解能を上げる場合に調整するとより詳細な映像情報を得ることができます。
- 本命令を設定しない場合、初期設定値に基づきます。
- 本命令は初期設定を変更しません。電源再立ち上げの場合設定は失われています。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 本命令は次のカメラ入力（CAMIN）実行時に有効となります。

**関連項目**

CAMIN、VISCAMOUT

**用　例**

```
CAMLEVEL 1,0,100          ' カメラ 1 番の入力レベル範囲を最大にします。
VISCAMOUT 1               ' カメラからの映像 ( 動画像 ) をモニタに表示します。
CAMIN 1                   ' カメラ 1 番の映像を格納メモリ 0 番にテーブル 0 番
                          ' ( カメラの映像と ' 同じ輝度 ) で変換し格納します。
DELAY 2000                '2 秒停止
CAMLEVEL 1,30,80          ' カメラ 1 番の入力レベル下限を 30%、上限を 80% に
                          ' 設定します。
VISCAMOUT 1               ' カメラからの映像 ( 動画像 ) をモニタに表示します。
CAMIN 1
```

# VISCAMOUT（ステートメント）

**機　能**

カメラからの映像をモニタに表示します。

**書　式**

VISCAMOUT＜カメラ番号＞［,＜テーブル番号＞］

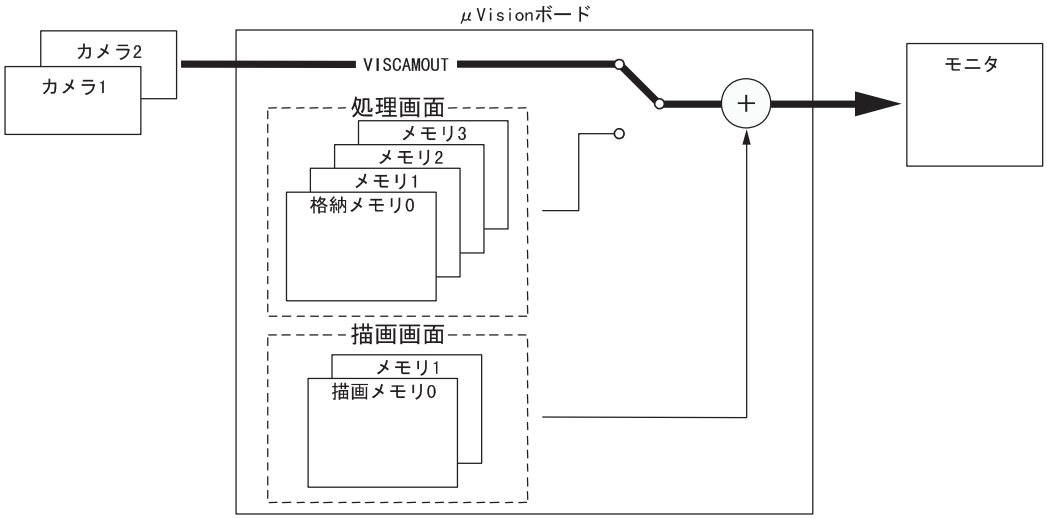
**説　明**

＜カメラ番号＞

　カメラ番号を指定します。(1 または 2)

＜テーブル番号＞

　表示する際のルックアップテーブル番号を指定します。(0 ～ 15) 省略時は 1 を指定します。




**注意**

本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISDEFTABLE、VISPLNOUT、VISOVERLAY、CAMMODE

**用　例**

```
VISCAMOUT 1,1          ' カメラ 1 番からの映像 ( 動画像 ) をモニタにテーブル
                       '1 番 (0 ～ 175 約 70% 輝度圧縮 ) で変換し表示します。
VISCAMOUT 1            'VISCAMOUT 1,1 と同じ結果が得られます。
```

# VISPLNOUT（ステートメント）

**機　能**

格納メモリの画像をモニタに表示します。

**書　式**

VISPLNOUT＜格納メモリ番号＞［,＜テーブル番号＞］

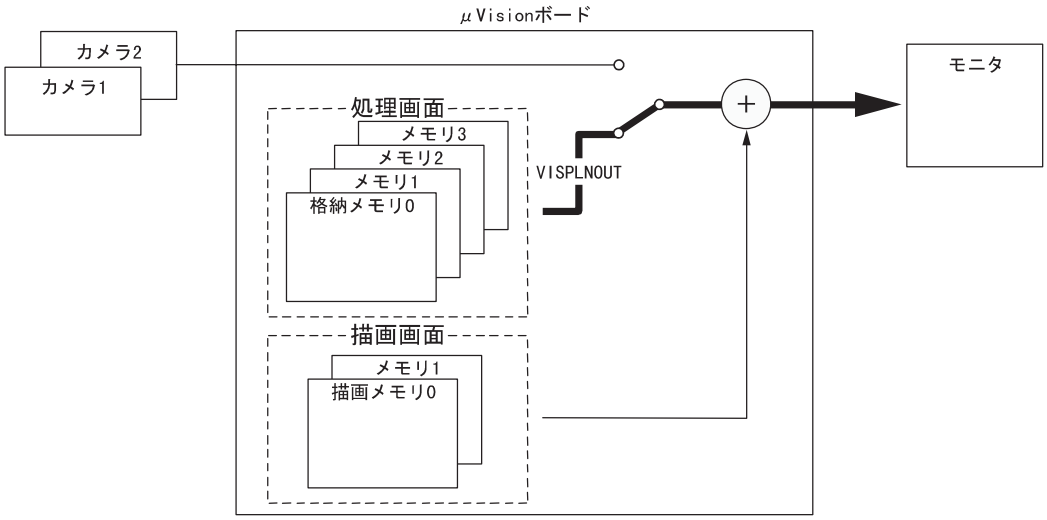
**説　明**

＜格納メモリ番号＞

格納メモリ番号（処理画面番号）を指定します。(0 ～ 3)

＜テーブル番号＞

表示する際のルックアップテーブル番号を指定します。(0 ～ 15) 省略時は 1 を指定します。




**関連項目**

VISDEFTABLE、VISCAMOUT、VISOVERLAY

**用　例**

```
VISPLNOUT 0,1          ' 格納メモリ 0 番の画像 ( 停止画像 ) をモニタにテーブ
                       ' ル 1 番 (0 ～ 175 約 70% 輝度圧縮 ) で変換し表示します。
VISPLNOUT 0            'VISPLNOUT 0,1 と同じ結果が得られます。
```

# VISOVERLAY（ステートメント）

**機　能**

描画画面の情報をモニタに表示します。

**書　式**

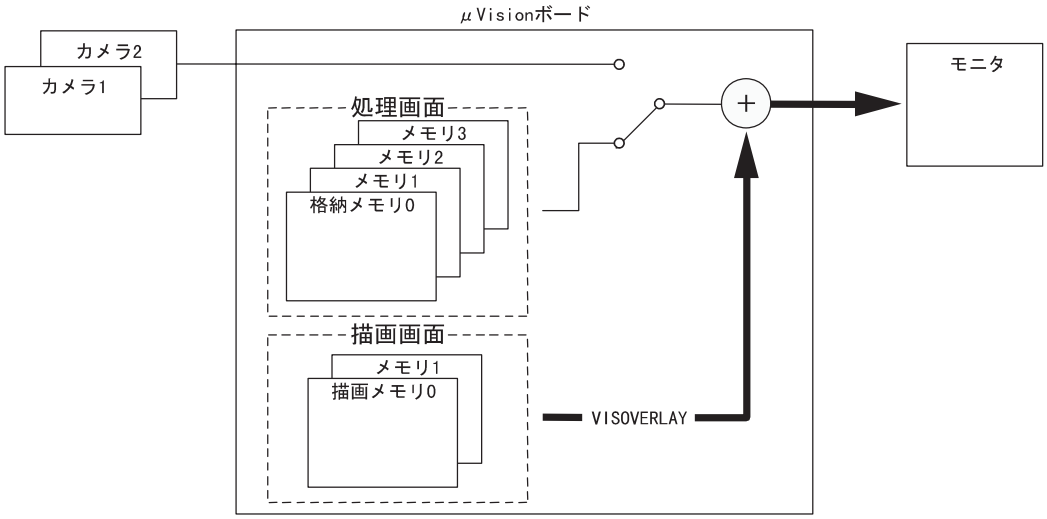
VISOVERLAY＜番号＞

**説　明**

＜番号＞

描画画面の表示を設定します。(0 ～ 3)

0：描画画面を表示しません。
1：描画画面 0 番を表示します。
2：描画画面 1 番を表示します。
3：描画画面 0、1 の両方を同時に表示します。




**関連項目**

VISCAMOUT、VISPLNOUT

**用　例**

```
VISOVERLAY 3                    ' 描画先画面の設定
VISSCREEN 1,0                   '
VISCLS 0                        '
VISLOC 10,10                    ' 表示位置設定
VISPRINT " 描画画面 0 に描画 "  ' 画面に文字を描画します。
VISSCREEN 1,1                   ' 描画先画面の設定
VISLOC 10,11                    ' 表示位置設定
VISPRINT " 描画画面 1 に描画 "  ' 画面に文字を描画します。
VISOVERLAY 0                    ' 描画画面表示を中止します。
DELAY 5000                      '5 秒停止
VISOVERLAY 1                    ' 描画画面 0 番をモニタに表示します。
DELAY 5000                      '5 秒停止
VISOVERLAY 2                    ' 描画画面 1 番をモニタに表示します。
DELAY 5000                      '5 秒停止
VISOVERLAY 3                    ' 描画画面 0、1 番をモニタに表示します。
```

# VISDEFTABLE（ステートメント）

**機　能**

カメラ映像の取り込み、画像出力の際のルックアップテーブルデータを設定します。

**書　式**

VISDEFTABLE＜テーブル番号＞,＜入力値＞,＜出力値＞

**説　明**

＜テーブル番号＞

ルックアップテーブル番号を指定します。(5 ～ 15)

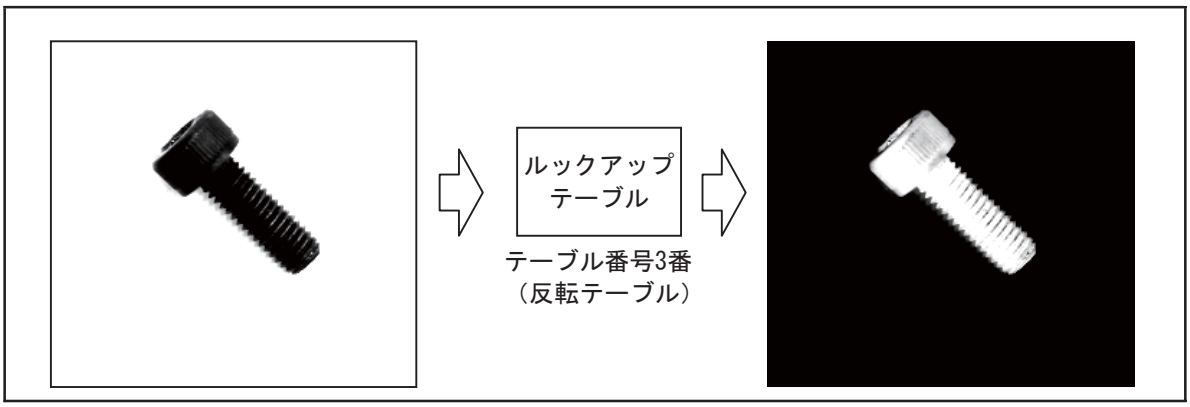
＜入力値＞

テーブルの入力値を指定します。(0 ～ 255)

＜出力値＞

テーブルの出力値を指定します。(0 ～ 255)

ルックアップテーブルを用いて変換した例




**注意**

- 本命令を設定しない場合、初期設定値に基づきます。
- 本命令は 2 ～ 3( 秒 ) の実行時間を要します。
- テーブル番号 0 ～ 4 はあらかじめ有用と考えられるテーブルが設定されています。本命令では変更はできません。
  テーブル 0 番：ノーマル（0 ～ 255）
  テーブル 1 番：70% 輝度圧縮 （0 ～ 175）
  テーブル 2 番：γ 補正
  テーブル 3 番：反転
  テーブル 4 番：70% 輝度圧縮反転
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 本命令実行時に電源を OFF しないでください。次の電源 ON 時に正常終了しなかったと判断し、視覚に関する情報を初期化します。

**関連項目**

CAMIN、VISCAMOUT、VISPLNOUT、VISREFTABLE

用　例

```
VISSCREEN 1,0,1                ’
VISCLS 0                       ’
VISCAMOUT 1,1                  ’カメラ 1 番からの映像 ( 動画像 ) をモニタにテーブル 1
                               ’番 (0 ～ 175 約 75% 輝度圧縮 ) で変換し表示します。
DELAY 5000                     ’5 秒停止
VISDEFCHAR 4,4,2               ’
FOR I1 = 0 TO 255              ’
 VISLOC 10,10                  ’
 VISPRINT I1                   ’
 VISDEFTABLE 5,I1,255-I1       ’テーブル 5 番の設定
NEXT I1                        ’
VISCAMOUT 1,5                  ’カメラ 1 番からの映像 ( 動画像 ) を反転しモニタに表
                               ’示します。
```

# VISREFTABLE（関数）

**機　能**

ルックアップテーブルのデータを参照します。

**書　式**

VISREFTABLE(< テーブル番号 >, < 入力値 >)

**説　明**

< テーブル番号 >

ルックアップテーブル番号を指定します。（0 ～ 15）

< 入力値 >

テーブルの入力値を指定します。（0 ～ 255）

**関連項目**

VISDEFTABLE

**用　例**

```
VISSCREEN 1,0,1                ’
VISCLS 0                       ’
FOR I1 = 0 TO 255              ’
  I2 = VISREFTABLE(1,I1)       ’テーブル 1 番のデータを取得します。
  VISLOC 10,10                 ’表示位置の設定
  VISDEFCHAR 1,1,2             ’表示文字の設定
  VISPRINT "データ ";I1;"=";I2’表示
NEXT I1                        ’
```

## 21.4　ウィンドウ設定

# WINDMAKE（ステートメント）

**機　能**

画像処理する範囲を指定します。

**書　式**

WINDMAKE＜ウィンドウ形状＞,＜ウィンドウ番号＞,＜パラメータ 1>,＜パラメータ 2>,…

直線ウィンドウ（2 点指定）:

WINDMAKE P,＜ウィンドウ番号＞,＜始点 X 座標＞,＜始点 Y 座標＞,＜終点 X 座標＞,＜終点 Y 座標＞

直線ウィンドウ（長さ、角度指定）:

WINDMAKE L,＜ウィンドウ番号＞,＜長さ＞［,＜角度＞］

円ウィンドウ:

WINDMAKE C,＜ウィンドウ番号＞,＜半径＞

楕円ウィンドウ:

WINDMAKE E,＜ウィンドウ番号＞,＜幅＞,＜高さ＞

扇ウィンドウ:

WINDMAKE S,＜ウィンドウ番号＞,＜外径＞,＜内径＞,＜開始角度＞,＜終了角度＞,＜刻み角＞[,＜モード＞]

矩形ウィンドウ:

WINDMAKE R,＜ウィンドウ番号＞,＜幅＞,＜高さ＞［,＜角度＞［,＜モード＞]]

**説　明**

＜ウィンドウ形状＞

ウィンドウの形状を指定します。

P：直線ウィンドウ Point to Point （2 点指定）
L：直線ウィンドウ Line （長さ、角度指定）
C：円ウィンドウ Circle
E：楕円ウィンドウ Ellipse
S：扇ウィンドウ Section
R：矩形ウィンドウ Rectangle

＜ウィンドウ番号＞

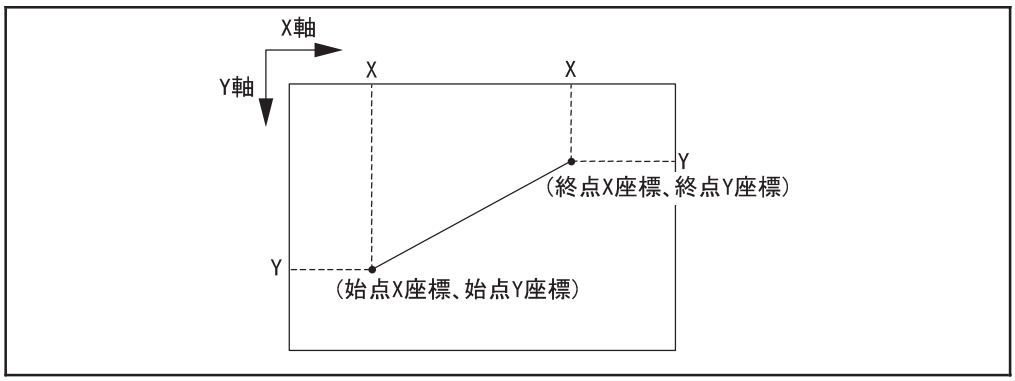
ウィンドウの番号を指定します。（0 ～ 511）

＜パラメータ＞

各ウィンドウ形状を設定する場合の値を指定します。

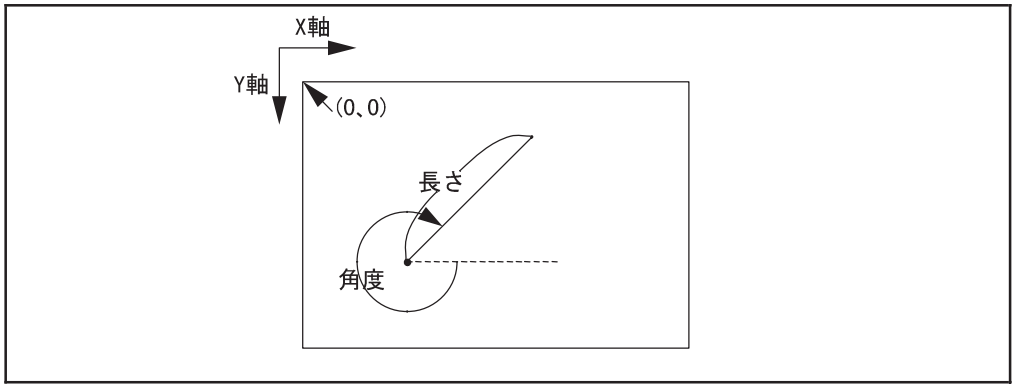
直線ウィンドウ（2 点指定）：

＜始点 X 座標＞直線の始点 X 座標を指定します。（0 ～ 511）
＜始点 Y 座標＞直線の始点 Y 座標を指定します。（0 ～ 479）
＜終点 X 座標＞直線の終点 X 座標を指定します。（0 ～ 511）
＜終点 Y 座標＞直線の終点 Y 座標を指定します。（0 ～ 479）



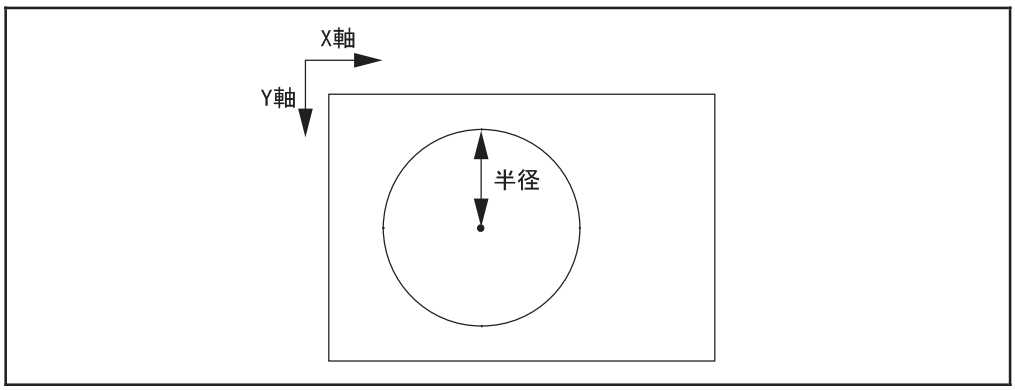

直線ウィンドウ（長さ、角度指定）：

＜長さ＞直線の長さを指定します。（1 ～ 512）
＜角度＞直線の角度を指定します。省略時は 0° が設定されます。




円ウィンドウ：

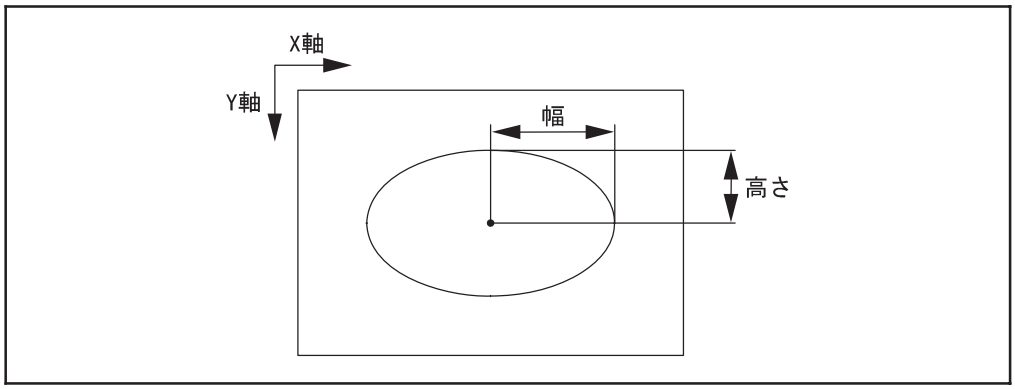
＜半径＞円の半径を指定します。（1 ～ 240）



楕円ウィンドウ：

＜幅＞楕円の幅を指定します。（1 ～ 256）

＜高さ＞楕円の高さを指定します。（1 ～ 240）




扇ウィンドウ：

＜外径＞扇の外径を指定します。（外形＞内径）（1 ～ 9999）

＜内径＞扇の内径を指定します。（外形＞内径）（1 ～ 9999）

＜開始角度＞扇の開始角度を指定します。　（-720 ～ 720）

＜終了角度＞扇の終了角度を指定します。　（-720 ～ 720）

＜刻み角＞処理する際の分解能を指定します。（0.1 ～ 360）

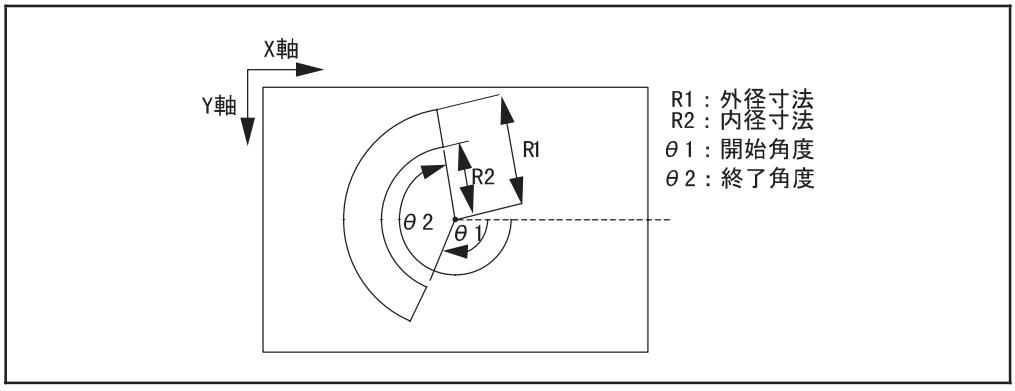
＜モード＞描画する際のモードを指定します。（0 ～ 2）

0：VISPROJ、VISEDGE の走査方向を子午線方向に指定します。

1：VISPROJ、VISEDGE の走査方向を円周方向に指定します。

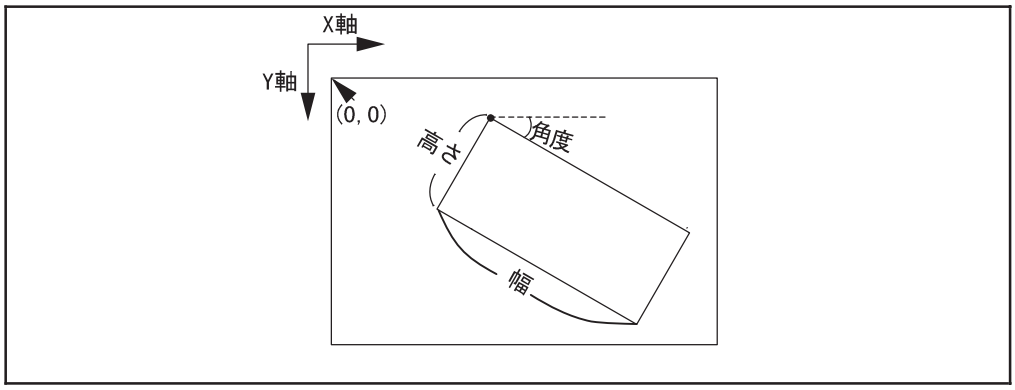
2：VISPROJ、VISEDGE の走査方向を指定しません。

省略時は 2 を指定します。



矩形ウィンドウ：

＜幅＞矩形の幅を指定します。（1 ～ 512）
＜高さ＞矩形の高さを指定します。（1 ～ 480）
＜角度＞描画する直線の角度を指定します。省略時は、0° が設定されます。（-720 ～ 720）
＜モード＞描画する際のモードを指定します。（0 ～ 2）
0：VISPROJ、VISEDGE の走査方向を幅方向に指定します。
1：VISPROJ、VISEDGE の走査方向を高さ方向に指定します。
2：VISPROJ、VISEDGE の走査方向を指定しません。
省略時は 0 を指定します。




**注意**

- 本命令は初期設定を変更しません。電源再立ち上げの場合、初期設定値が復活します。
- 恒久的に変更する必要がある場合、WINCAPS Ⅲにより編集してください。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISHIST、VISBINA、VISFILTER、VISMASK、VISMEASURE、VISPROJ、VISEDGE、VISREADQR、BLOB、SHMODEL、SHCORNER、SHCIRCLE

用　例

```
VISSCREEN 1,0,1                    ’
VISCLS 0                           ’
VISCAMOUT 1                        ’
CAMIN 1                            ’
VISPLNOUT 0                        ’
WINDMAKE P,1,50,100,100,150        ’直線ウィンドウ (2 点指定 ) を作成します。
WINDMAKE L,2,100,45                ’直線ウィンドウ ( 長さ、角度指定 ) を作成します。
WINDMAKE C,3,50                    ’円ウィンドウを作成します。
WINDMAKE E,4,50,100                ’楕円ウィンドウを作成します。
WINDMAKE S,5,100,80,90,300,1,2     ’扇ウィンドウを作成します。
WINDMAKE R,6,100,50,45,2           ’矩形ウィンドウを作成します。
VISMEASURE 1,100,100,1,1,128       ’
WINDDISP 1                         ’
VISMEASURE 2,150,150,1,1,128       ’
WINDDISP 2                         ’
VISMEASURE 3,200,200,1,1,128       ’
WINDDISP 3                         ’
VISMEASURE 4,250,250,1,1,128       ’
WINDDISP 4                         ’
VISMEASURE 5,300,300,1,1,128       ’
WINDDISP 5                         ’
VISMEASURE 6,350,350,1,1,128       ’
WINDDISP 6                         ’
```

# WINDCLR（ステートメント）

**機　能**

設定したウィンドウ情報を消去します。

**書　式**

WINDCLR < ウィンドウ番号 >

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。（0 ～ 511）

**注意**

- 本命令は初期設定を変更しません。電源再立ち上げの場合初期設定値が復活します。
- 恒久的に消去する必要がある場合、WINCAPS Ⅲにより消去してください。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE

**用　例**

```
VISSCREEN 1,0,1                    '
VISCLS 0                           '
VISCAMOUT 1                        '
WINDMAKE R,1,50,100,0,2            '
FOR I1 = 0 TO 7                    '
  I2 = WINDREF(1,I1)               ' ウィンドウ 1 番のデータを取得します。
  VISLOC 0,I1                      ' 表示位置の設定
  VISPRINT " データ ";I1;"=";I2    ' 表示
NEXT I1                            '
WINDCLR 1                          '
VISLOC 0,9                         '
VISPRINT " ウィンドウ情報消去後 "  '
FOR I1 = 0 TO 7                    '
  I2 = WINDREF(1,I1)               ' ウィンドウ 1 番のデータを取得します。
  VISLOC 0,10+I1                   ' 表示位置の設定
  VISPRINT " データ ";I1;"=";I2    ' 表示
NEXT I1                            '
```

# WINDCOPY（ステートメント）

**機　能**

ウィンドウのデータをコピーします。

**書　式**

WINDCOPY＜コピー元ウィンドウ番号＞,＜コピー先ウィンドウ番号＞

**説　明**

＜コピー元ウィンドウ番号＞

コピーするウィンドウの番号を指定します。（0 ～ 511）

＜コピー先ウィンドウ番号＞

コピー先ウィンドウの番号を指定します。（0 ～ 511）

**注意**

- 本命令は初期設定を変更しません。電源再立ち上げの場合コピーした設定は失われています。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE

用　例

```
VISSCREEN 1,0,1                          ’
VISCLS 0                                 ’
VISCAMOUT 1                              ’
WINDCLR 2                                ’
WINDMAKE R,1,50,100,0,2                  ’
VISLOC 0,0                               ’
VISPRINT " コピー元ウィンドウ情報 "          ’
                                         ’
FOR I1 = 0 TO 7                          ’
  I2 = WINDREF(1,I1)                     ’ ウィンドウ 1 番のデータを取得します。
  VISLOC 0,1+I1                          ’ 表示位置の設定
  VISPRINT " データ ";I1;"=";I2          ’ 表示
NEXT I1                                  ’
VISLOC 0,9                               ’
VISPRINT " ウィンドウ情報コピー前 "          ’
                                         ’
FOR I1 = 0 TO 7                          ’
  I2 = WINDREF(2,I1)                     ’ ウィンドウ 1 番のデータを取得します。
  VISLOC 0,10+I1                         ’ 表示位置の設定
  VISPRINT " データ ";I1;"=";I2          ’ 表示
NEXT I1                                  ’
VISLOC 0,18                              ’
VISPRINT " ウィンドウ情報コピー後 "          ’
                                         ’
FOR I1 = 0 TO 7                          ’
  I2 = WINDREF(2,I1)                     ’ ウィンドウ 1 番のデータを取得します。
  VISLOC 0,19+I1                         ’ 表示位置の設定
  VISPRINT " データ ";I1;"=";I2          ’ 表示
NEXT I1                                  ’
```

# WINDREF（関数）

**機　能**

ウィンドウの情報を得ます。

**書　式**

WINDREF(< ウィンドウ番号 >, < 項目 >)

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。(0 ～ 511)

< 項目 >

得たい情報の番号を指定します。(0 ～ 9)

項目番号 0：ウィンドウ設定の有無　　戻り値 有り = 0 なし = -1

項目番号 1：ウィンドウ形状　　戻り値 ( 下表参照 )

項目番号 2：ウィンドウ原点の X 座標

項目番号 3：ウィンドウ原点の Y 座標

項目番号 4 ～ 9：ウィンドウの各設定データ　　戻り値 各設定データ ( 下表参照 )

| | 項　目 | | | | | | |
|---|---|---|---|---|---|---|---|
| ウィンドウ形状 | 1 | 4 | 5 | 6 | 7 | 8 | 9 |
| 直線 (2 点指定 ) | 0 | 始点 X 座標 | 始点 Y 座標 | 終点 X 座標 | 終点 Y 座標 | -1 | -1 |
| 直線 ( 長さ、角度指定 ) | 1 | 長さ | 角度 | -1 | -1 | -1 | -1 |
| 円 | 2 | 半径 | -1 | -1 | -1 | -1 | -1 |
| 楕円 | 3 | 幅 | 高さ | -1 | -1 | -1 | -1 |
| 扇 | 4 | 外径 | 内径 | 開始角度 | 終了角度 | 刻み角 | モード |
| 矩形 | 5 | 幅 | 高さ | 角度 | モード | -1 | -1 |

**注意**

- **設定データがない場合、戻り値は -1 になります。**
- **得られるデータは現状の設定データであり、初期設定データではありません。**
- **本命令はロボットコントローラでは μVision ボード（オプション）が必要です。**

**関連項目**

WINDMAKE

**用　例**

```
VISSCREEN 1,0,1                  '
VISCLS 0                         '
VISCAMOUT 1                      '
WINDMAKE R,1,50,100,0,2          '
FOR I1 = 0 TO 7                  '
  I2 = WINDREF(1,I1)             ' ウィンドウ 1 番のデータを取得します。
  VISLOC 0,I1                    ' 表示位置の設定
  VISPRINT " データ ";I1;"=";I2  ' 表示
NEXT I1                          '
```

# WINDDISP（ステートメント）

**機　能**

設定したウィンドウを描画します。

**書　式**

WINDDISP＜ウィンドウ番号＞

**説　明**

＜ウィンドウ番号＞

ウィンドウの番号を指定します。（0 ～ 511）

**注意**

- ウィンドウ情報には、ウィンドウの処理原点位置がありますが、このデータは各画像処理命令により決定されます。したがって、処理命令を実行した後に本命令を実行すれば、正しい位置に表示されます。
- 新規にウィンドウを設定した直後は、ウィンドウ原点は X = 0 、Y = 0 に設定されています。
- 描画する画面は VISSCREEN で設定された画面になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISSCREEN

**用　例**

```
VISOVERLAY 1                ’描画専門画面 0 番をモニタに表示します。
VISSCREEN 1,0,1             ’描画画面 0 番に即時描画します。
WINDMAKE R,1,200,200,0,2    ’矩形のウィンドウ作成
WINDDISP 1                  ’ウィンドウの表示 ( 原点 X=0、Y=0)
VISHIST 1,100,100           ’ヒストグラム処理実行
WINDDISP 1                  ’ウィンドウの表示 ( 原点 X=100、Y=100)
```

## 21.5　描画

# VISSCREEN（ステートメント）

**機　能**

描画する画面を指定します。

**書　式**

VISSCREEN＜描画対象＞,＜画面番号＞[,＜描画モード＞]

**説　明**

＜描画対象＞

格納メモリ、オーバレイメモリ（画面）のどちらかを指定します。（0 または 1）

0：格納メモリ（処理画面）

1：オーバレイメモリ（描画専用画面）

＜画面番号＞

各画面での画面番号

格納メモリ（処理画面）：0 ～ 3 の 4 画面

オーバレイメモリ（描画専用画面）：0、1 の 2 画面

＜描画モード＞

描画する際の表示方法を指定します。（0 または 1）

0：即時表示 OFF

1：即時表示 ON

省略時は 1 を指定します。

**注意**

- 電源投入直後の初期設定は VISSCREEN 1，0，1 に設定されています。
- 即時表示　ON　では描画命令実行時、描画結果はモニタ表示されますが、OFF　では　VISPLNOUT、VISOVERLAY のいづれか描画対象に応じた命令を実行することで画面が更新されます。
- 即時描画 ON では描画命令実行時約 0.02 秒の更新時間が含まれますので、頻繁に描画する場合その累積が大きくなります。即時描画を OFF し、まとめて表示することで処理時間を短縮することができます。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISBRIGHT、VISCLS、VISPUTP、VISLINE、VISPTP、VISRECT、VISCIRCLE、VISELLIPSE、VISSECT、VISCROSS、VISLOC、VISDEFCHAR、VISPRINT

用　例

```
VISSCREEN 1,0,1                ' 描画画面 0 番に即時描画します。
VISCLS 0                       '
FOR I1 = 100 TO 200 STEP 2     '
  VISRECT I1,I1,200,200        ' 幅 200 高さ 200 の矩形を描画します。
NEXT I1                        '
VISCLS 0                       ' 画面のクリア
VISSCREEN 1,0,0                ' 描画専用画面 0 番に描画します。
FOR I1 = 100 TO 200 STEP 2     '
  VISRECT I1,I1,200,200        ' 幅 200 高さ 200 の矩形を描画します。
NEXT I1                        '
VISOVERLAY 1                   ' 描画専用画面 0 番をモニタに表示 ( 更新 ) します。
```

# VISBRIGHT（ステートメント）

**機　能**

描画する際の輝度値を指定します。

**書　式**

VISBRIGHT < 輝度値 >

**説　明**

< 輝度値 >

描画する際の輝度値を指定します。（0 ～ 255）

**注意**

- 描画対象が格納メモリ（処理画面）の場合は、指定輝度で描画しますが、オーバレイメモリ（描画専用メモリ）の場合下記の表示輝度になります。
  輝度値 128 ～ 255 ：255（白）
  輝度値 1 ～ 127 ：1（黒）
  輝度値 0：スルー（透明）
- 電源投入時、初期設定は VISBRIGHT 255 に設定されています。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISSCREEN、VISCLS、VISPUTP、VISLINE、VISPTP、VISRECT、VISCIRCLE、VISELLIPSE、VISSECT、VISCROSS、VISLOC、VISDEFCHAR、VISPRINT

**用　例**

```
VISPLNOUT 0                     ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                 ' 処理画面 0 番に即時描画します。
VISCLS 0                        '
FOR I1 = 10 TO 200 STEP 5       '
  VISBRIGHT I1                  '
  VISRECT 100+I1,100,200,200    ' 幅 200 高さ 200 の矩形を描画します。
NEXT I1                         '
DELAY 500                       '0.5 秒停止
VISCLS 0                        ' 画面のクリア
VISSCREEN 1,0,1                 ' 描画専用画面 0 番に即時描画します。
FOR I1 = 10 TO 200 STEP 5       '
  VISBRIGHT I1                  '
  VISRECT 100+I1,100,200,200    ' 幅 200 高さ 200 の矩形を描画します。
NEXT I1                         '
```

# VISCLS（ステートメント）

**機　能**

モードで指定した画面を指定輝度で塗りつぶし（クリア）します。

**書　式**

VISCLS [< モード >[,< 輝度値 >]]

**説　明**

< モード >

(0 ～ 3) 省略時は 2 を指定します。

0：VISSCREEN 命令で設定された画面を指定輝度で塗りつぶします。
1：処理画面すべてを指定輝度で塗りつぶします。
2：描画画面すべてを指定輝度で塗りつぶします。
3：すべての画面を指定輝度で塗りつぶします。

< 輝度値 >

塗りつぶし（クリア）する際の輝度値を指定します。(0 ～ 255) 省略時は 0 を指定します。

**注意**

- 描画対象が格納メモリ（処理画面）の場合は、指定輝度でクリアしますが、オーバレイメモリ（描画専用メモリ）の場合下記の表示輝度になります。
  輝度値 128 ～ 255 ：255（白）
  輝度値 1 ～ 127 ：1 （黒）
  輝度値 0：スルー（透明）
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISSCREEN

用　例

```
VISSCREEN 1,0,1                 ’
VISCLS 0                        ’
VISSCREEN 1,1,1                 ’
VISCLS 0                        ’
VISPLNOUT 0                     ’格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                 ’処理画面 0 番に即時描画します。
FOR I1 = 10 TO 200 STEP 5       ’
  VISCLS 0,I1                   ’画面を I1 の指定輝度でクリアします。
NEXT I1                         ’
DELAY 500                       ’0.5 秒停止
VISCLS 0                        ’画面を輝度 0 でクリアします。
VISRECT 200,200,200,200         ’幅 200 高さ 200 の矩形を描画します。
DELAY 500                       ’0.5 秒停止
VISCLS 0                        ’描画画面の矩形をクリアします。
VISCAMOUT 1                     ’
VISSCREEN 1,0,1                 ’描画専用画面 0 番に即時描画します。
VISRECT 100,100,200,200         ’幅 200 高さ 200 の矩形を描画します。
DELAY 500                       ’0.5 秒停止
VISCLS 0                        ’描画画面の矩形をクリアします。
VISCLS 0,127                    ’描画画面をクリア ( 黒 ) します。
DELAY 500                       ’0.5 秒停止
VISCLS 0,255                    ’描画画面をクリア ( 白 ) します。
DELAY 500                       ’0.5 秒停止
VISCLS 0                        ’描画画面をスルー ( 透明 ) にします。
```

# VISPUTP（ステートメント）

**機　能**

画面に点を描画します。

**書　式**

VISPUTP <X 座標 >, <Y 座標 >

**説　明**

<X 座標 >

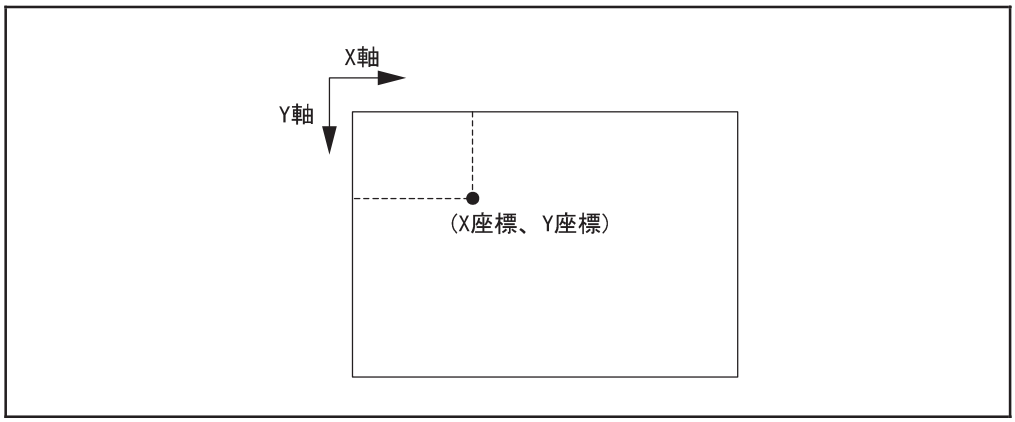
点を描画する X 座標を指定します。

<Y 座標 >

点を描画する Y 座標を指定します。

**注意**

- X、Y 座標の数値のチェックはしていません。描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

**用　例**

```
VISPLNOUT 0                    ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                ' 処理画面 0 番に即時描画します。
VISCLS 0                       ' 画面のクリア
VISBRIGHT 255                  ' 描画する輝度値の設定
FOR I1 = 10 TO 200 STEP 5      '
  VISPUTP 100+I1,100           '(100+I1,100) 座標に点を描画します。
NEXT I1                        '
```

# VISLINE（ステートメント）

**機　能**

画面に直線を描画します。

**書　式**

VISLINE<X 座標 >，<Y 座標 >，< 長さ > [，< 角度 >]

**説　明**

<X 座標 >

直線を描画する X 座標を指定します。

<Y 座標 >

直線を描画する Y 座標を指定します。

< 長さ >

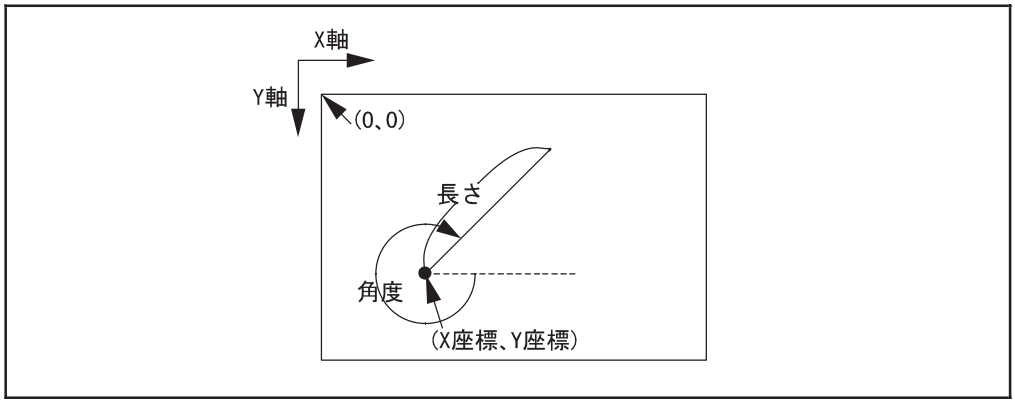
描画する直線の長さを指定します。

< 角度 >

描画する直線の角度を指定します。省略時は、0° が設定されます。

**注意**

- X、Y 座標、長さ、角度の数値のチェックはしていません。描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

**用　例**

```
VISPLNOUT 0                    ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                ' 処理画面 0 番に即時描画します。
VISCLS 0                       ' 画面のクリア
VISBRIGHT 255                  ' 描画する輝度値の設定
FOR I1 = 0 TO 360 STEP 5       '
  VISLINE 256,256,100,I1       ' 長さ 100 の直線を描画します。
NEXT I1                        '
```

# VISPTP（ステートメント）

**機　能**

画面に 2 点間を結ぶ直線を描画します。

**書　式**

VISPTP < 始点 X 座標 >, < 始点 Y 座標 >, < 終点 X 座標 >, < 終点 Y 座標 >

**説　明**

< 始点 X 座標 >

直線を描画する始点 X 座標を指定します。

< 始点 Y 座標 >

直線を描画する始点 Y 座標を指定します。

< 終点 X 座標 >

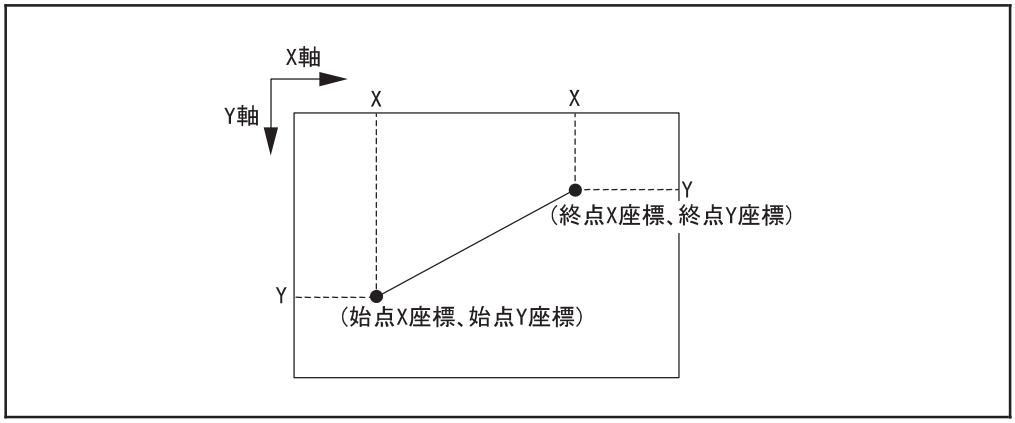
直線を描画する終点 X 座標を指定します。

< 終点 Y 座標 >

直線を描画する終点 Y 座標を指定します。

**注意**

- X、Y 座標の数値のチェックはしていません。描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

**用　例**

```
VISPLNOUT 0                  ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1              ' 処理画面 0 番に即時描画します。
VISCLS 0                     ' 画面のクリア
VISBRIGHT 255                ' 描画する輝度値の設定
FOR I1 = 200 TO 300          '
  VISPTP 100,100,200,I1      '2 点間を結ぶ直線を描画します。
NEXT I1                      '
```

# VISRECT（ステートメント）

**機　能**

画面に矩形を描画します。

**書　式**

VISRECT <X 座標 >, <Y 座標 >, < 幅 >, < 高さ >[, < モード >[, < 角度 >] ]

**説　明**

<X 座標 >

矩形を描画する X 座標を指定します。

<Y 座標 >

矩形を描画する Y 座標を指定します。

< 幅 >

描画する矩形の幅を指定します。

< 高さ >

描画する矩形の高さを指定します。

< モード >

描画する際のモードを指定します。

0：矩形のアウトラインのみ描画します。

1：すべて塗りつぶした矩形を描画します。

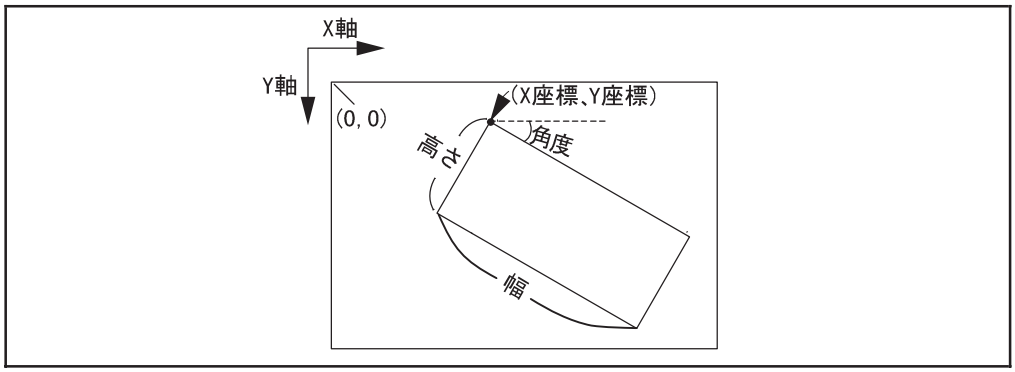
省略時は 0 を指定します。

< 角度 >

描画する直線の角度を指定します。省略時は、0° が設定されます。

**注意**

- X､Y 座標､長さ､角度の数値のチェックはしていません｡描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

用　例

```
VISPLNOUT 0                              ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                          ' 処理画面 0 番に即時描画します。
VISCLS 0                                 ' 画面のクリア
VISBRIGHT 255                            ' 描画する輝度値の設定
FOR I1 = 0 TO 360 STEP 5                 '
  VISRECT 256,256,100,100,0,I1           ' 高さ 100 幅 100 の矩形を描画します。
NEXT I1                                  '
VISRECT 200,200,50,50,215                ' 高さ 50 幅 50 の塗りつぶした矩形を描画します。
```

# VISCIRCLE（ステートメント）

**機　能**

画面に円を描画します。

**書　式**

VISCIRCLE <X 座標 >, <Y 座標 >, < 半径 > [, < モード >]

**説　明**

<X 座標 >

円を描画する X 座標を指定します。

<Y 座標 >

円を描画する Y 座標を指定します。

< 半径 >

描画する円の半径を指定します。

< モード >

描画する際のモードを指定します。

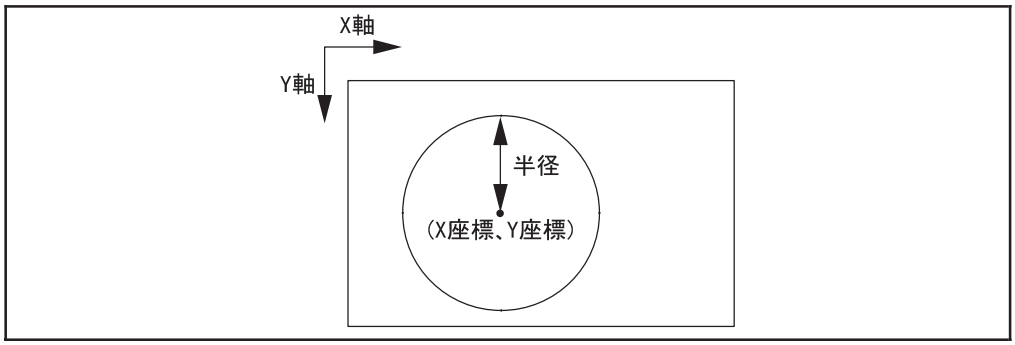
0：円のアウトラインのみ描画します。

1：すべて塗りつぶした円を描画します。

省略時は 0 を指定します。

**注意**

- X、Y 座標、半径の数値のチェックはしていません。描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

**用　例**

```
VISPLNOUT 0                 ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1             ' 処理画面 0 番に即時描画します。
VISCLS 0                    ' 画面のクリア
VISBRIGHT 255               ' 描画する輝度値の設定
VISCIRCLE 255,255,100       ' 半径 100 の円を描画します。
VISCIRCLE 200,200,50,1      ' 半径 50 の塗りつぶした円を描画します。
```

# VISELLIPSE（ステートメント）

**機　能**

画面に楕円を描画します。

**書　式**

VISELLIPSE <X 座標 >，<Y 座標 >，< 幅 >，< 高さ > [，< モード >]

**説　明**

<X 座標 >

　楕円を描画する X 座標を指定します。

<Y 座標 >

　楕円を描画する Y 座標を指定します。

< 幅 >

　描画する楕円の幅を指定します。

< 高さ >

　描画する楕円の高さを指定します。

< モード >

　描画する際のモードを指定します。

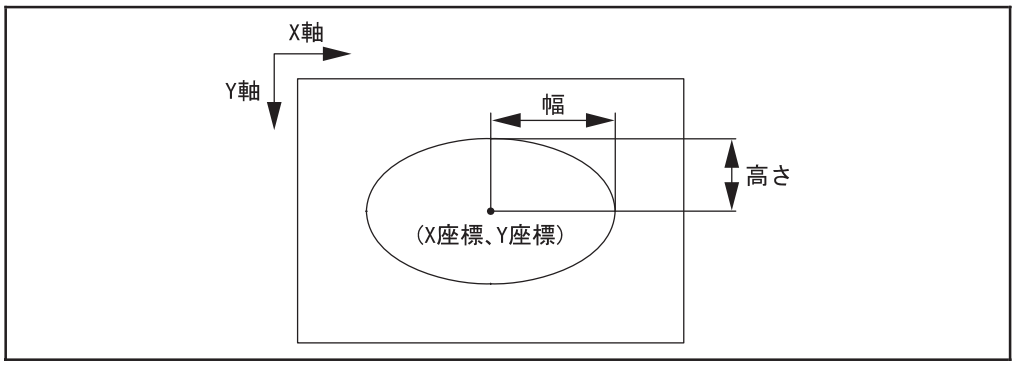
　　0：楕円のアウトラインのみ描画します。

　　1：すべて塗りつぶした楕円を描画します。

　省略時は 0 を指定します。

**注意**

- X、Y 座標、幅、高さの数値のチェックはしていません。描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

用　例

```
VISPLNOUT 0                    ’格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                ’処理画面 0 番に即時描画します。
VISCLS 0                       ’画面のクリア
VISBRIGHT 255                  ’描画する輝度値の設定
VISELLIPSE 255,255,100,50      ’幅 100 高さ 50 の楕円を描画します。
VISELLIPSE 255,255,50,100,1    ’幅 50 高さ 100 の塗りつぶした楕円を描画します。
```

# VISSECT（ステートメント）

**機　能**

画面に扇を描画します。

**書　式**

VISSECT <X 座標 >, <Y 座標 >, < 外径 >, < 内径 >, < 開始角度 >, < 終了角度 >

**説　明**

<X 座標 >

扇を描画する X 座標を指定します。

<Y 座標 >

扇を描画する Y 座標を指定します。

< 外径 >

描画する扇の外径を指定します。

< 内径 >

描画する扇の内径を指定します。

< 開始角度 >

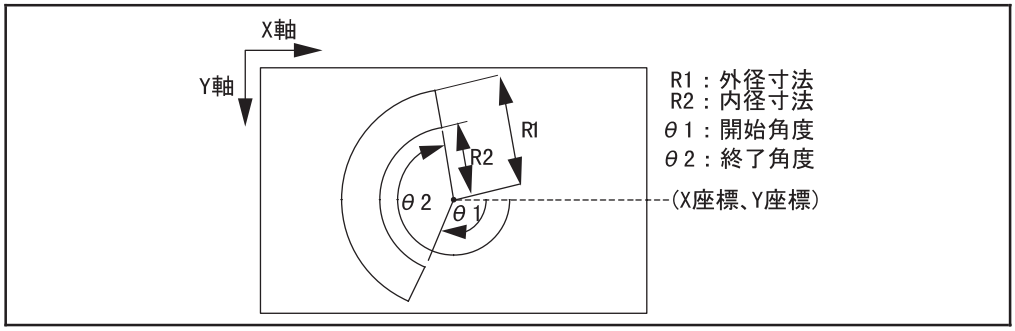
描画する扇の開始角度を指定します。

< 終了角度 >

描画する扇の終了角度を指定します。

**注意**

- 指定する数値のチェックはしていません。描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

**用　例**

```
VISPLNOUT 0                          ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                      ' 処理画面 0 番に即時描画します。
VISCLS 0                             ' 画面のクリア
VISBRIGHT 255                        ' 描画する輝度値の設定
VISSECT 255,255,100,50,100,260       ' 扇を描画します。
```

# VISCROSS（ステートメント）

**機　能**

画面に十字マークを描画します。

**書　式**

VISCROSS <X 座標 >, <Y 座標 > [, < 軸長 1>[, < 軸長 2> [, < 角度 >] ] ]

**説　明**

<X 座標 >

十字マークを描画する X 座標を指定します。

<Y 座標 >

十字マークを描画する Y 座標を指定します。

< 軸長 1>

描画する十字マークの軸長を指定します。省略の場合 10 を指定します。

< 軸長 2>

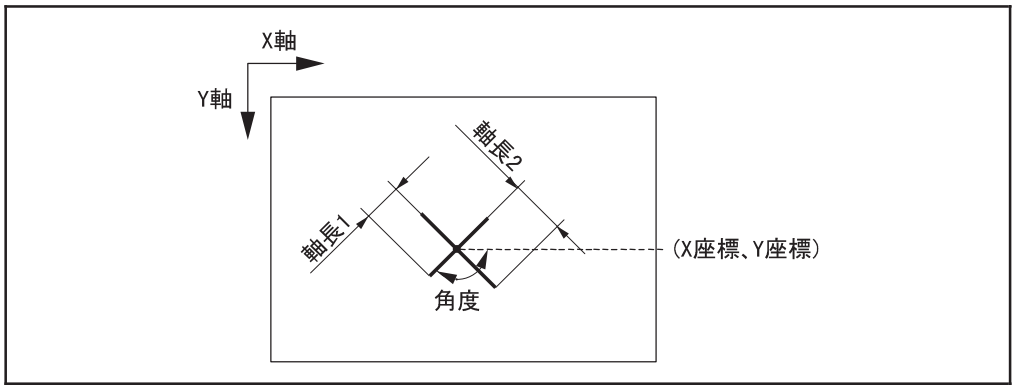
描画する十字マークの軸長を指定します。省略の場合 10 を指定します。

< 角 度 >

描画する十字マークの角度を指定します。省略の場合 0° を指定します。

**注意**

- 指定する数値のチェックはしていません。描画の範囲を超えてもエラーにはなりません。
- 対象画面は VISSCREEN で設定された画面になります。
- 描画の輝度値は VISBRIGHT で指定した輝度になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISSCREEN、VISBRIGHT

**用　例**

```
VISPLNOUT 0                  ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1              ' 処理画面 0 番に即時描画します。
VISCLS 0                     ' 画面のクリア
VISBRIGHT 255                ' 描画する輝度値の設定
VISCROSS 255,255,10,20,45    ' 十字マークを描画します。
```

# VISLOC（ステートメント）

**機　能**

文字の表示位置を指定します。

**書　式**

VISLOC <X 位置 >, <Y 位置 > [, < モード >]

**説　明**

<X 位置 >

文字を描画する X 位置を指定します。

<Y 位置 >

文字を描画する Y 位置を指定します。

< モード >

X、Y 位置データの座標モードを指定します。省略した場合、0 を指定します。

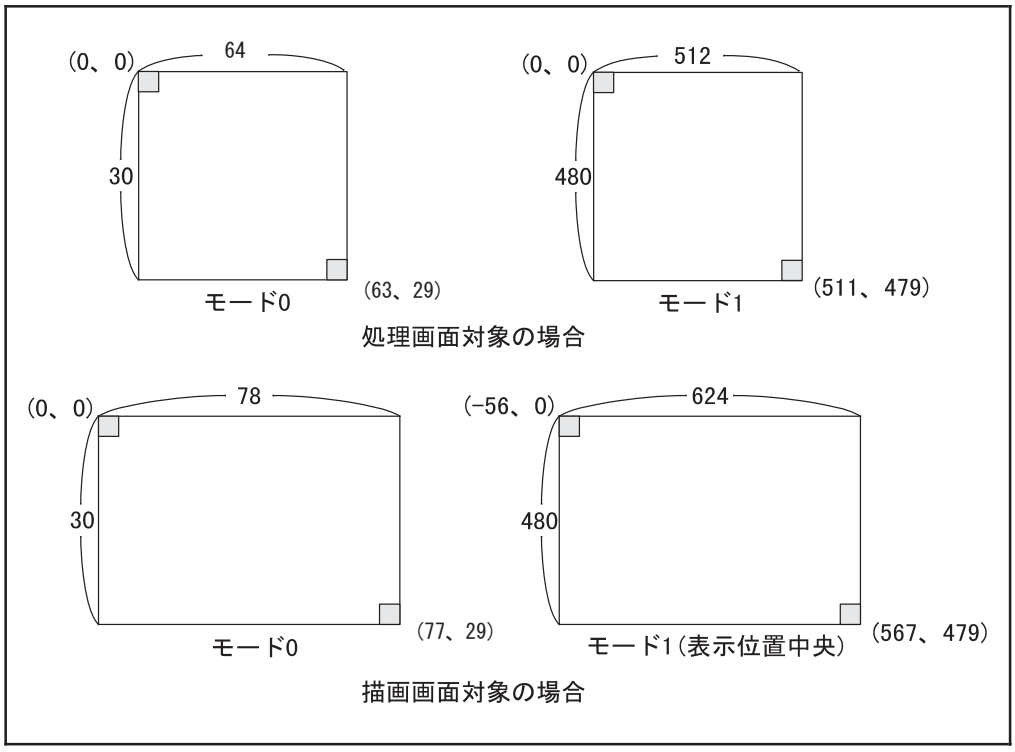
0：桁、行で指定します。

1：XY 座標系で指定します。

指定する数値のチェックはしていません。描画の範囲を超えてもエラーになりません。

**注意**

- 電源投入時、初期設定は VISLOC 0，0，0 を設定しています。
- モードを 0 にしていた場合、文字を描画した後の最終位置を記憶しています。
- 文字の表示が右端に達した場合自動的に改行します。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。



関連項目

VISPRINT

用　例

```
VISPLNOUT 0                          ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                      ' 処理画面 0 番に即時描画します。
VISCLS 0                             ' 画面のクリア
VISBRIGHT 255                        ' 描画する輝度値の設定
VISLOC 10,10                         ' 文字を表示する位置を指定します。
VISPRINT "テストモード 0"             '
VISLOC 10,10,1                       ' 文字を表示する位置を指定します。
VISPRINT "テストモード 1"             '
VISPRINT "テストモード 1 の続き "     '
VISPRINT "******** 改行 ********"
```

# VISDEFCHAR（ステートメント）

**機　能**

文字のサイズと表示方法を指定します。

**書　式**

VISDEFCHAR＜横サイズ＞，＜縦サイズ＞，＜表示方法＞

**説　明**

＜横サイズ＞

文字の横サイズを指定します。（1 ～ 4）

＜縦サイズ＞

文字の縦サイズを指定します。（1 ～ 4）

＜表示方法＞

文字の表示方法を指定します。（0 ～ 3）

0：白文字
1：黒文字
2：白抜き文字
3：黒抜き文字

**注意**

- 電源投入時、初期設定は VISDEFCHAR 1，1，0 を指定しています。
- 文字サイズは縦横 16 x 16 ドットを基本にしています。（VISDEFCHAR 1、1、＊）




VISDEFCHAR 1,1,1

VISDEFCHAR 2,2,1

- 白、黒文字は背景をそのままにして文字のみ表示します。
- 白抜き文字は背景を黒にし、文字を白く、黒抜き文字は背景を白にし、文字を黒く表示しますので背景によって文字が見にくい場合でも文字を見やすくします。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISPRINT

用　例

```
VISPLNOUT 0                 ’格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1             ’処理画面 0 番に即時描画します。
VISCLS 128                  ’画面のクリア
VISLOC 10,10                ’表示位置の設定
VISDEFCHAR 1,1,0            ’文字サイズ、表示方法を指定します。
VISPRINT "サイズ 1"         ’
VISLOC 10,11                ’表示位置の設定
VISDEFCHAR 2,2,1            ’文字サイズ、表示方法を指定します。
VISPRINT "サイズ 2"         ’
VISLOC 10,13                ’表示位置の設定
VISDEFCHAR 2,2,2            ’文字サイズ、表示方法を指定します。
VISPRINT "白抜き文字"       ’
VISDEFCHAR 2,2,3            ’文字サイズ、表示方法を指定します。
VISPRINT "黒抜き文字"       ’
```

# VISPRINT（ステートメント）

**機　能**

画面に文字・数字を表示します。

**書　式**

VISPRINT < メッセージ > [< セパレータ >< メッセージ > …]

**説　明**

< メッセージ >

"" で指定した文字、変数、定数などを指定します。

< セパレータ >

カンマ（,）、セミコロン（ ; ）を使用します。
カンマ：メッセージ間をスペースで空けます。
セミコロン：メッセージ間を空けません。

< メッセージ > の最後にカンマやセミコロンを指定すると改行が起こらず、その行で次の VISPRINT による表示を続行します。

**注意**

**実数変数を表示する場合は、小数点以下第四位までを画面に表示します。**

**関連項目**

VISSCREEN、VISBRIGHT、VISLOC、VISDEFCHAR

**用　例**

```
VISPLNOUT 0                              ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
VISSCREEN 0,0,1                          ' 処理画面 0 番に即時描画します。
VISCLS 0                                 ' 画面のクリア
VISLOC 10,10                             ' 表示位置の設定
VISDEFCHAR 1,1,0                         ' 文字サイズ、表示方法を指定します。
I1= 10
F1 = 0.999
VISPRINT " 整数変数 ";I1," 実数変数 F1";F1
```

## 21.6　画像処理

# VISWORKPLN（ステートメント）

**機　能**

処理対象の格納メモリ（処理画面）を指定します。

**書　式**

VISWORKPLN＜格納メモリ番号＞

**説　明**

＜格納メモリ番号＞

処理対象の格納メモリ番号を指定します。（0 ～ 3）省略時 0 を指定します。

**注意**

- **電源投入時、初期設定は VISWORKPLN 0 が設定されています。**
- **本命令はロボットコントローラでは μVision ボード（オプション）が必要です。**

**関連項目**

VISGETP、VISHIST、VISBINA、VISMEASURE、VISPROJ、VISEDGE、VISREADQR、BLOB、SHDEFMODEL、SHMODEL、SHCORNER、SHCIRCLE

**用　例**

```
VISWORKPLN 0                    ’処理対象を格納メモリ 0 番に指定します。
```

# VISGETP（関数）

**機　能**

格納メモリ（処理画面）から指定座標の輝度を取得します。

**書　式**

VISGETP（<X 座標 >, <Y 座標 >）

**説　明**

<X 座標 >

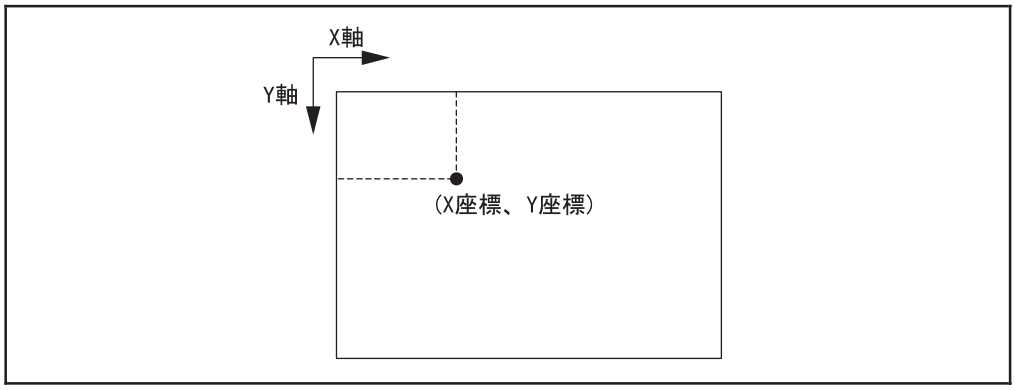
X 座標を指定します。（0 ～ 511）

<Y 座標 >

Y 座標を指定します。（0 ～ 479）

**注意**

- 処理対象は VISWORKPLN で指定した画面になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。




**関連項目**

VISWORKPLN

**用　例**

```
VISPLNOUT 0                    ' 格納メモリ 0 番 ( 処理画面 ) を表示します。
FOR I1 = 10 TO 200 STEP 5      '
  I2 = VISGETP(100+I1,100)     '(100+I1,100) 座標の輝度値を I2 に代入します。
  VISLOC 10,10                 ' 表示位置の設定
  VISDEFCHAR 1,1,3             ' 表示文字サイズ、表示方法を指定します。
  VISPRINT " 輝度値 =";I2      '
NEXT I1
```

# VISHIST（ステートメント）

**機　能**

画面のヒストグラム（輝度分布）を得ます。

**書　式**

VISHIST < ウィンドウ番号 >, <X 座標 >, <Y 座標 >

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。(0 ～ 511)

<X 座標 >

X 座標を指定します。(0 ～ 511)

<Y 座標 >

Y 座標を指定します。(0 ～ 479)

**注意**

- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウの場合、エラーになります。
- 処理対象は VISWORKPLN で指定した画面になります。
- 処理結果は VISREFHIST 命令で読み出すことが可能です。
- 処理結果はメモリに記憶されています。再度本命令を実行するまで保持されています。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISWORKPLN、VISREFHIST

**用　例**

```
WINDMAKE R,1,100,100,0,2     ' ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                      ' カメラ映像を格納メモリに取得します。
VISWORKPLN 0                 ' 処理の対象画面を格納メモリ 0 番に設定します。
VISHIST 1,100,100            ' 座標 (100,100) を原点にウィンドウ 1 番でヒストグラ
                             ' ムを実行します。
FOR I1 = 0 TO 255            '
  I2 = VISREFHIST(I1)        ' 輝度値 I1 の分布データを I2 に代入します。
  VISLOC 10,10               ' 表示位置の設定
  VISDEFCHAR 1,1,3           ' 表示文字サイズ、表示方法を指定します。
  VISPRINT " 分布データ =";I2  '
NEXT I1
```

# VISREFHIST（関数）

**機　能**

ヒストグラム結果を読み出します。

**書　式**

VISREFHIST（< 輝度値 >）

**説　明**

< 輝度値 >

読み出すヒストグラムの輝度値を指定します。（0 ～ 255）

**関連項目**

VISHIST

**用　例**

```
WINDMAKE R,1,100,100,0,2     ' ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                      ' カメラ映像を格納メモリに取得します。
VISWORKPLN 0                 ' 処理の対象画面を格納メモリ 0 番に設定します。
VISHIST 1,100,100            ' 座標 (100,100) を原点にウィンドウ 1 番でヒストグラ
                             ' ムを実行します。
FOR I1 = 0 TO 255            '
  I2 = VISREFHIST(I1)        ' 輝度値 I1 の分布データを I2 に代入します。
  VISLOC 10,10               ' 表示位置の設定
  VISDEFCHAR 1,1,3           ' 表示文字サイズ、表示方法を指定します。
  VISPRINT " 分布データ =";I2  '
NEXT I1
```

# VISLEVEL（関数）

**機　能**

ヒストグラム結果に基づき 2 値化レベルを求めます。

**書　式**

VISLEVEL（< モード >[, < 基準面積 >[, < 処理対象 >] ]）

**説　明**

< モード >

2 値化レベルを求める方法を指定します。(0 ～ 2)

0：モード法
1：判別分析法
2：P タイル法

内容については P5-4「5.1.1.5 ?2 値化」をご覧ください。

< 基準面積 >

P タイル法による 2 値化レベルを求める際の面積値を指定します。(1 ～ 245760)省略時は、1 を指定します。

< 処理対象 >

P タイル法による 2 値化レベルを求める際の対象（白、黒）を指定します。(0 または 1) 省略時は、0 を指定します。

0：黒（輝度値 0 から積算）
1：白（輝度値 255 から積算）

**注意**

- **モード法、判別分析法では < 基準面積 >< 処理対象 > は使用しません。**
- **本機能を使用する場合は、VISHIST をあらかじめ実行する必要があります。**
- **指定データに誤り、または処理結果が異常である場合、戻り値は -1 になります。**
- **本命令はロボットコントローラでは μVision ボード（オプション）が必要です。**

**関連項目**

VISHIST

用　例

```
VISCLS 0                          ’
WINDMAKE R,1,100,100,0,2          ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                           ’カメラ映像を格納メモリに取得します。
VISWORKPLN 0                      ’処理の対象画面を格納メモリ 0 番に設定します。
VISHIST 1,100,100                 ’座標 (100,100) を原点にウィンドウ 1 番でヒストグラ
                                  ’ムを実行します。
WINDDISP 1                        ’
I2 = VISLEVEL(0)                  ’モード法により 2 値化レベルを求めます。
VISLOC 10,10                      ’
IF I2 = -1 THEN VISPRINT "エラー" ELSE VISPRINT "2 値化レベル =";I2
I2 = VISLEVEL(1)                  ’判別分析法により 2 値化レベルを求めます。
VISLOC 10,11                      ’
IF I2 = -1 THEN VISPRINT "エラー" ELSE VISPRINT "2 値化レベル =";I2
I2 = VISLEVEL(2,100,0)            ’P タイル法により 2 値化レベルを求めます。
VISLOC 10,12                      ’
IF I2 = -1 THEN VISPRINT "エラー" ELSE VISPRINT "2 値化レベル =";I2
```

# VISBINA（ステートメント）

**機　能**

画面を 2 値化処理します。

**書　式**

VISBINA < ウィンドウ番号 >, <X 座標 >, <Y 座標 >, <2 値下限 >[, <2 値上限 >]

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標 >

X 座標を指定します。（0 ～ 511）

<Y 座標 >

Y 座標を指定します。（0 ～ 479）

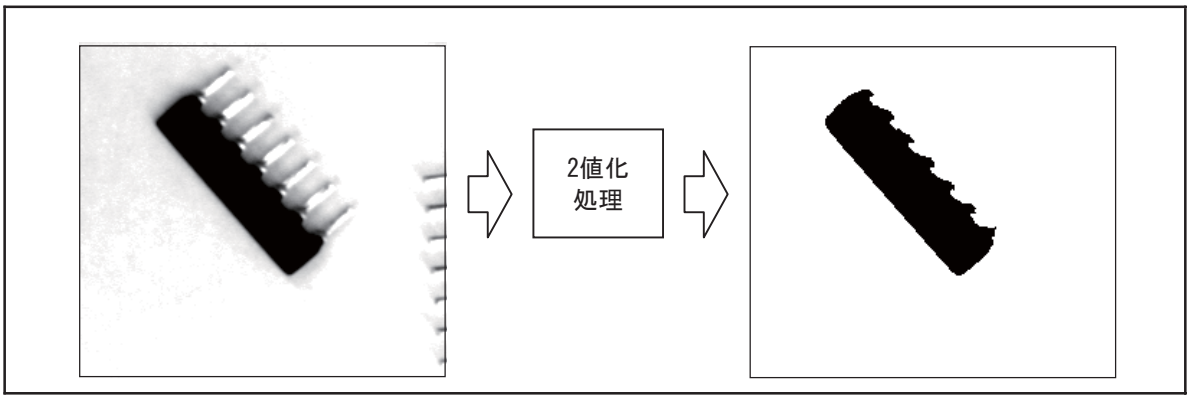
<2 値下限 >

2 値化する際の下限レベルを指定します。（0 ～ 254 下限 < 上限）

<2 値上限 >

2 値化する際の上限レベルを指定します。（1 ～ 255 下限 < 上限）
省略時 255 を指定します。

**2 値化の例**




**注意**

- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他の　　ウィンドウの場合、エラーになります。
- 処理対象は VISWORKPLN で指定した画面になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISWORKPLN

用　例

```
VISSCREEN 1,0,1                ’描画画面 0 番に即時描画します。
WINDMAKE R,1,100,100,0,2       ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                        ’カメラ映像を格納メモリに取得します。
VISWORKPLN 0                   ’処理の対象画面を格納メモリ 0 番に設定します。
VISPLNOUT 0                    ’格納メモリ 0 番をモニタに表示します。
VISBINA 1,100,100,128,255      ’ウィンドウ内を 2 値化します。
WINDDISP 1                     ’ウィンドウを描画します。
```

# VISBINAR（ステートメント）

**機　能**

画面を 2 値化表示します。

**書　式**

VISBINAR < モード > [，<2 値下限 > [，2 値上限]]

**説　明**

< モード >

2 値化表示するモードを指定します。(0 または 1)
0：2 値化表示を中止し、元の表示に戻します。
1：2 値化表示を実行します。

<2 値下限 >

2 値化する際の下限レベルを指定します。(0 ～ 254 下限 < 上限)

<2 値上限 >

2 値化する際の上限レベルを指定します。(1 ～ 255 下限 < 上限)
省略時 255 を指定します。

**注意**

- 現在モニタに表示している画面全体を 2 値化表示します。
- 格納メモリ（処理画面）が表示選択されていてもメモリの内容は書き換わりません。
- モードを 0 にしない限り、2 値化表示を継続します。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISBINA

**用　例**

```
VISBINAR 1,128,255          '2 値化表示を開始します。
VISCAMOUT 1                 ' カメラ 1 番をモニタに表示します。
DELAY 5000                  '5 秒停止
VISPLNOUT 0                 ' 格納メモリ 0 番をモニタに表示します。
DELAY 5000                  '5 秒停止
VISBINAR 0                  ' 元の表示に戻します。
```

# VISFILTER（ステートメント）

**機　能**

画像にフィルタ処理を行ないます。

**書　式**

VISFILTER＜ウィンドウ番号＞，<X 座標＞，<Y 座標＞，＜処理画面＞，＜格納画面＞［，＜モード＞］

**説　明**

＜ウィンドウ番号＞

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標＞

X 座標を指定します。（0 ～ 511）

<Y 座標＞

Y 座標を指定します。（0 ～ 479）

＜処理画面＞

格納メモリ番号を指定します。（0 ～ 3）

＜格納画面＞

フィルタ処理結果を格納する格納メモリ番号を指定します。(0 ～ 3)

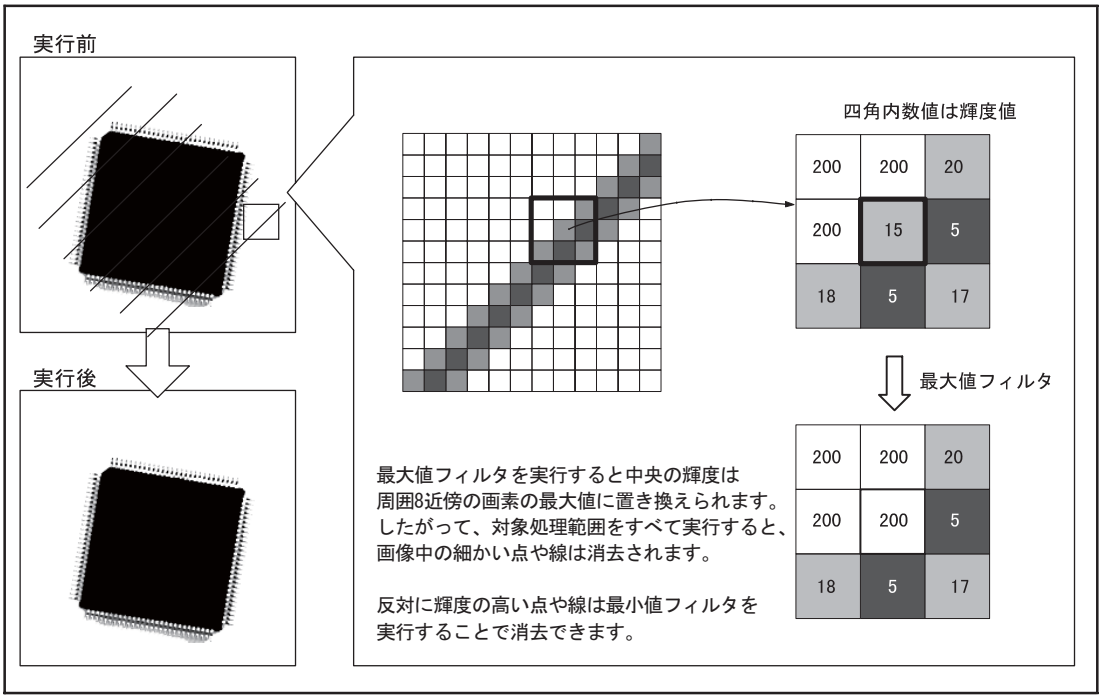
＜モード＞

フィルタ処理（3 x 3 空間フィルタ）の種類を指定します。

0：最小値フィルタ

1：最大値フィルタ

省略時は 0 を指定します。



**注意**

- 処理画面と格納画面が同じ番号の場合エラーになります。
- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウの場合、エラーになります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE

**用　例**

```
VISSCREEN 1,0,1                 ’描画画面 0 番に即時描画します。
WINDMAKE R,1,100,100,0,2        ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                         ’カメラ映像を格納メモリに取得します。
VISPLNOUT 1                     ’格納メモリ 1 番をモニタに表示します。
VISFILTER 1,100,100,0,1,0       ’
VISFILTER 1,100,100,1,0,0       ’
VISFILTER 1,100,100,0,1,0       ’
VISFILTER 1,100,100,1,0,0       ’
VISFILTER 1,100,100,0,1,0       ’
WINDDISP 1                      ’ウィンドウを描画します。
```

# VISMASK（ステートメント）

**機　能**

画像間演算をします。

**書　式**

VISMASK＜ウィンドウ番号>, <X座標>, <Y座標>, <画面1>, <画面2>, <モード>[, <2値下限>[, <2値上限>]]

**説　明**

＜ウィンドウ番号＞

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標＞

X 座標を指定します。（0 ～ 511）

<Y 座標＞

Y 座標を指定します。（0 ～ 479）

＜画面 1>

演算する格納メモリ番号を指定します。（0 ～ 3）

＜画面 2>

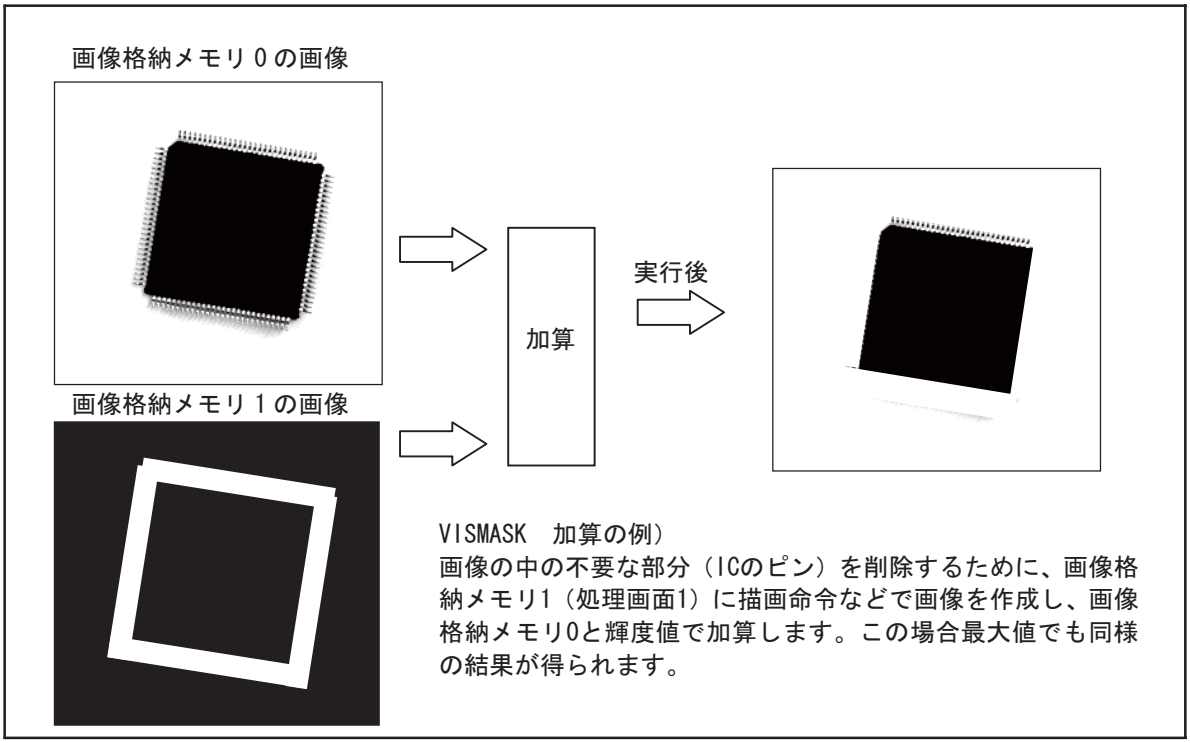
演算する格納メモリ番号を指定します。処理結果の格納先になります。（0 ～ 3）

＜モード＞

画像間演算の種類を指定します。（0 ～ 10）

0：2 値 AND
1：2 値 OR
2：2 値 XOR
3：AND（輝度値を 2 進数化し各ビットごとに AND）
4：OR（輝度値を 2 進数化し各ビットごとに OR）
5：XOR（輝度値を 2 進数化し各ビットごとに XOR）
6：加算（輝度値が 255 以上の場合 255 に固定）
7：減算（輝度値が 0 以下の場合 0 に固定）
8：最大値（輝度値の大きい方を選択）
9：最小値（輝度値の小さい方を選択）

10：絶対値（輝度値の差の絶対値）




**注意**

- 処理画面と格納画面が同じ番号の場合エラーになります。
- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウの場合、エラーになります。
- ２値化の演算では、画面１も２値化処理されます。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISBINA

**用　例**

```
VISSCREEN 0,0,1                 ’格納メモリ 0 番に即時描画します。
WINDMAKE R,1,512,480,0,2        ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                         ’カメラ映像を格納メモリに取得します。
VISCOPY 0,1                     ’格納メモリ 0 番を 1 番にコピーします。
VISBRIGHT 255                   ’描画輝度を 255 に設定します。
VISRECT 100,100,100,100,1       ’画面に塗りつぶした矩形を描画します。
VISPLNOUT 1                     ’格納メモリ 1 番をモニタに表示します。
VISMASK 1,0,0,0,1,10            ’2 画面間の差の絶対値を演算します。
WINDDISP 1                      ’ウィンドウを描画します。
```

# VISCOPY（ステートメント）

**機　能**

画面をコピーします。

**書　式**

VISCOPY < コピー元画面 >, < コピー先画面 >

**説　明**

< コピー元画面 >

コピーする際のコピー元の格納メモリ番号を指定します。(0 ～ 3)

< コピー先画面 >

コピーする際のコピー先の格納メモリ番号を指定します。(0 ～ 3)

**注意**

- コピー元と先の番号が同じ場合エラーになります。
- コピー先の番号を画面に表示していた場合 VISSCREEN の描画モードの指定に従います。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISSCREEN

**用　例**

```
VISSCREEN 1,0,1              ' 描画画面 0 番に即時描画します。
CAMIN 1                      ' カメラ映像を格納メモリに取得します。
VISCOPY 0,1                  ' 格納メモリ 0 番の画像を 1 番にコピーします。
VISPLNOUT 1                  ' 格納メモリ 1 番をモニタに表示します。
```

# VISMEASURE（ステートメント）

**機　能**

ウィンドウ内の特徴（面積、重心、主軸角）を計測します。

**書　式**

VISMEASURE＜ウィンドウ番号＞,<X 座標＞,<Y 座標＞,＜処理対象＞,＜モード＞,<2 値下限＞[,<2 値上限＞]

**説　明**

＜ウィンドウ番号＞

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標＞

X 座標を指定します。（0 ～ 511）（Ver. 1.6 以降：-16384 ～ 16383）

<Y 座標＞

Y 座標を指定します。（0 ～ 479）（Ver. 1.6 以降：-16384 ～ 16383）

**注意**

**画面外を指定した場合、「ウィンドウ異常」になります。**

＜処理対象＞

計測する対象を指定します。（0 または 1）

0：黒（輝度値 ＜ 2 値化下限，2 値化上限 ＜ 輝度値 ）

1：白（2 値化下限 ≦ 輝度値 ≦ 2 値化上限）

＜モード＞

計測する特徴を指定します。

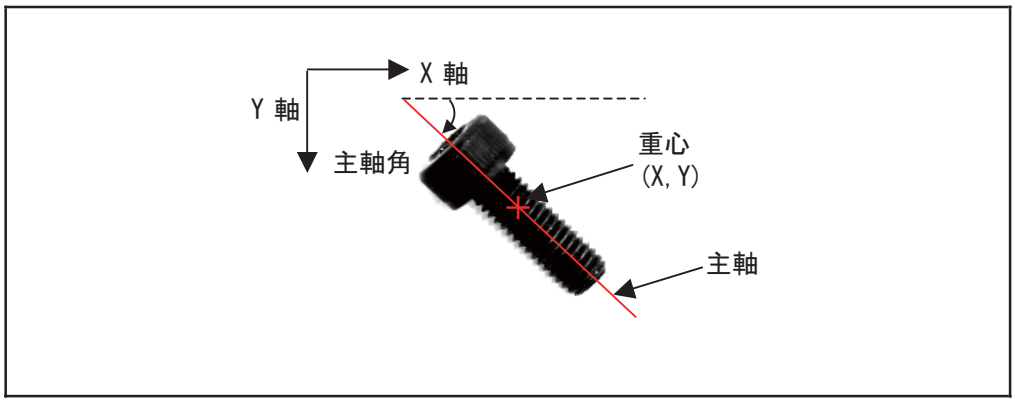
0：累積輝度、面積

1：累積輝度、面積、重心（1 次モーメント）

2：累積輝度、面積、重心（1 次モーメント）、主軸角（2 次モーメント）

**注意**

- **計測する特徴が多くなるほど処理時間が増加します。**
- **本命令を実行時に処理画面 3 番をワークエリアとして使用するため処理画面 3 番のデータは保証されません。また、処理画面 3 番（VISWORKPLN 3）は処理できません。**
- **本命令はロボットコントローラでは μVision ボード（オプション）が必要です。**



<2 値下限 >

2 値化する際の下限レベルを指定します。(0 ～ 254 下限 < 上限)

<2 値上限 >

2 値化する際の上限レベルを指定します。(1 ～ 255 下限 < 上限)
省略時 255 を指定します。

**注意**

- 処理範囲をウィンドウで指定します。(扇ウィンドウのみ主軸角は計測できません)

| | | モード | | |
|---|---|---|---|---|
| | ○：可 ×：不可 | 0 | 1 | 2 |
| 直線（2 点指示） | Windmake P | ○ | ○ | ○ |
| 直線（長さ、角度） | Windmake L | ○ | ○ | ○ |
| 円 | Windmake C | ○ | ○ | ○ |
| 楕円 | Windmake E | ○ | ○ | ○ |
| 扇 | Windmake S | ○ | ○ | × |
| 四角 | Windmake R | ○ | ○ | ○ |

- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS（n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | unknown |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 |
| 0 | 面積 |
| 1 | 重心 X 座標値 |
| 2 | 重心 Y 座標値 |
| 3 | 主軸角角度 |
| 4 | 累積輝度 |
| 5 | 1 次モーメント X |
| 6 | 1 次モーメント Y |
| 7 | 2 次モーメント X |
| 8 | 2 次モーメント Y |
| 9 | 2 次モーメント XY |

- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS

用　例

```
VISSCREEN 0,0,1                          ’格納メモリ 0 番に即時描画します。
WINDMAKE R,1,512,480,0,2                 ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                                  ’カメラ映像を格納メモリ 0 番に指定します。
VISWORKPLN 0                             ’対象を格納メモリ 0 番に指定します。
VISPLNOUT 0                              ’
VISMEASURE 1,0,0,1,2,128                 ’累積輝度、面積、重心、主軸角を求めます。
IF VISSTATUS(0) = 0.0 THEN               ’
 FOR I1 = 0 TO 9                         ’
  VISLOC 0,I1                            ’
  VISPRINT "結果";I1;"=";VISGETNUM(0,I1)
 NEXT I1                                 ’
END IF                                   ’
```

# VISPROJ（ステートメント）

**機　能**

ウィンドウ内の投影データを計測します。

**書　式**

VISPROJ < ウィンドウ番号 >，<X 座標 >，<Y 座標 >，< 処理対象 >，<2 値下限 > ［，<2 値上限 >］

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標 >

X 座標を指定します。（0 ～ 511）（Ver. 1.6 以降：-16384 ～ 16383）

<Y 座標 >

Y 座標を指定します。（0 ～ 479）（Ver. 1.6 以降：-16384 ～ 16383）

**注意**

**画面外を指定した場合、「ウィンドウ異常」になります。**

< 処理対象 >

計測する対象を指定します。（0 または 1）

0：黒（輝度値 ＜ 2 値化下限，2 値化上限 ＜ 輝度値 ）

1：白（2 値化下限 ≦ 輝度値 ≦ 2 値化上限）

<2 値下限 >

2 値化する際の下限レベルを指定します。（0 ～ 254 下限 < 上限）

<2 値上限 >

2 値化する際の上限レベルを指定します。（1 ～ 255 下限 < 上限）

省略時 255 を指定します。

**注意**

- **処理範囲はウィンドウで指定します。**
  **〇：可 ×：不可**

| | | |
|---|---|---|
| 直線（2 点指示） | Windmake P | ○ |
| 直線（長さ、角度） | Windmake L | ○ |
| 円 | Windmake C | × |
| 楕円 | Windmake E | × |
| 扇 | Windmake S | ○ |
| 四角 | Windmake R | ○ |

- **指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。**

- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | 投影データ個数 |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 ～ 511 |
| 0 | 面積 |
| 1 | unknown |
| 2 | unknown |
| 3 | unknown |
| 4 | 輝度積分 |
| 5 | unknown |
| 6 | unknown |
| 7 | unknown |
| 8 | unknown |
| 9 | unknown |

- 本命令を実行時に処理画面 3 番をワークエリアとして使用するため処理画面 3 番のデータは保証されません。また、処理画面 3 番（VISWORKPLN 3）は処理できません。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS

**用　例**

```
VISSCREEN 0,0,1                ’格納メモリ 0 番に即時描画します。
WINDMAKE R,1,100,20,0,1        ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                        ’カメラ映像を格納メモリに取得します。
VISWORKPLN 0                   ’対象を格納メモリ 0 番に指定します。
VISPROJ 1,100,100,1,128        ’投影データを計測します。
IF VISSTATUS(0) = 0.0 THEN     ’
  FOR I1 = 0 TO 19             ’
    VISPRINT VISGETNUM(I1,0)   ’
  NEXT I1                      ’
END IF                         ’
```

# VISEDGE（ステートメント）

**機　能**

ウィンドウ内のエッジを計測します。

**書　式**

VISEDGE< ウィンドウ番号 >, <X 座標 >, <Y 座標 >, < ステップ >, < 処理対象 >,
< レベル >[, < モード > [, <2 値下限 > [, <2 値上限 >]] ]

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標 >

X 座標を指定します。（0 ～ 511）（Ver. 1.6 以降：-16384 ～ 16383）

<Y 座標 >

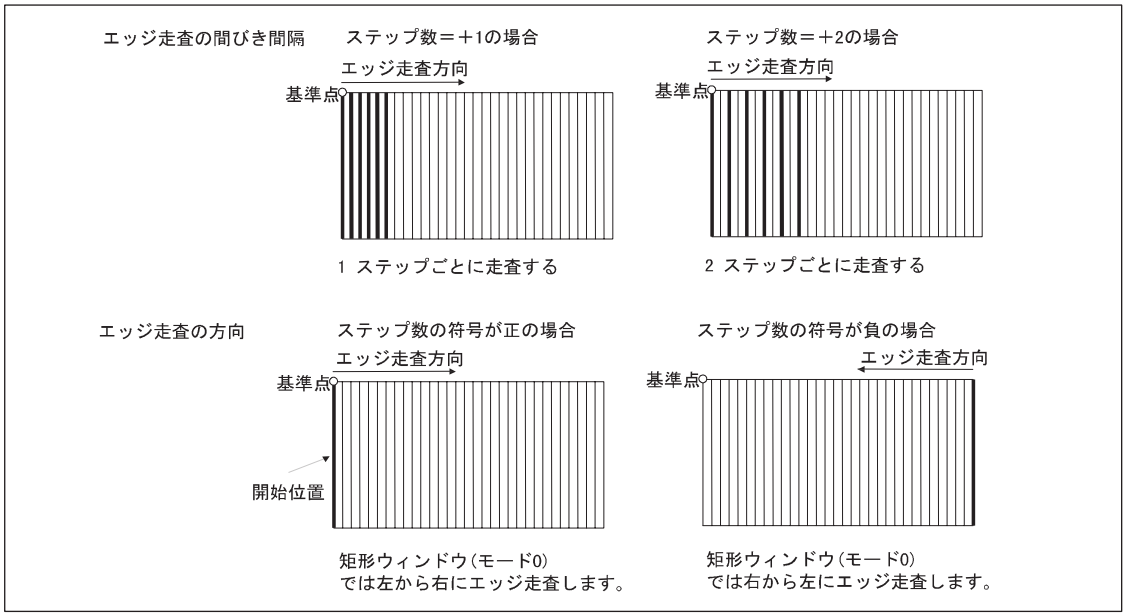
Y 座標を指定します。（0 ～ 479）（Ver. 1.6 以降：-16384 ～ 16383）

**注意**

**画面外を指定した場合、「ウィンドウ異常」になります。**

< ステップ >

エッジの走査方向と間引き間隔を指定します。（-511 ～ 511）



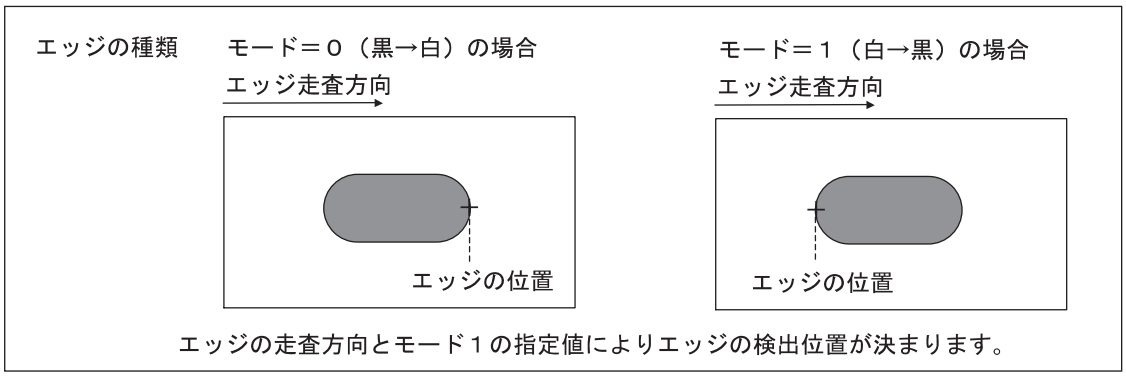
＜処理対象＞

計測する対象を指定します。(0 ～ 2)

0：黒から白への変化点（エッジ）

1：白から黒への変化点

2：変化点（0、1 の両エッジ）




＜レベル＞

エッジを検出するレベルを指定します。(0 ～ 512)

＜モード＞

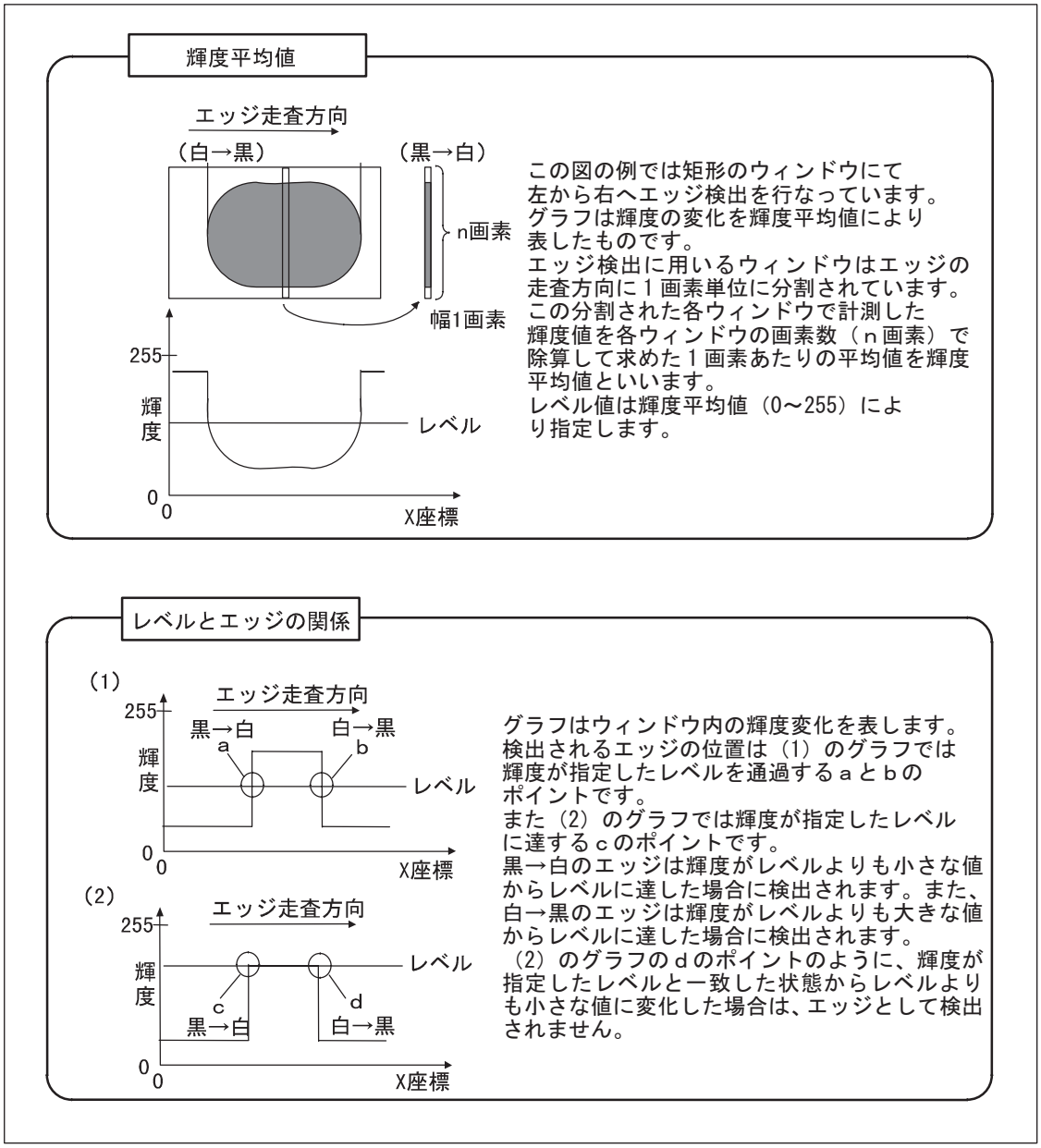
エッジの検出方法を指定します。省略時は 0 が指定されます。

0：平均輝度の絶対値

1：平均輝度の差分値

2：面積の絶対値

3：面積の差分値




<2 値下限>

2 値化する際の下限レベルを指定します。(0 ～ 254 下限 < 上限)
省略時は 0 が指定されます。

<2 値上限>

2 値化する際の上限レベルを指定します。(1 ～ 255 下限 < 上限)
省略時 255 を指定します。

**注意**

- 処理範囲をウィンドウで指定します。
  ○：可 ×：不可

| | | |
|---|---|---|
| 直線（2 点指示） | Windmake P | ○ |
| 直線（長さ、角度） | Windmake L | ○ |
| 円 | Windmake C | × |
| 楕円 | Windmake E | × |
| 扇 | Windmake S | ○ |
| 四角 | Windmake R | ○ |

- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | エッジ検出数 |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 ～ 511 |
| 0 | Unknown |
| 1 | X 座標値 |
| 2 | Y 座標値 |
| 3 | 角度（※） |
| 4 | Unknown |
| 5 | Unknown |
| 6 | Unknown |
| 7 | Unknown |
| 8 | Unknown |
| 9 | Unknown |

※処理範囲を扇（モード =0）ウィンドウで指定した場合にのみ計測結果（角度）を得ることができます。

- 処理画面は VISWORKPLN で確定した画面になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- ステップ数を大きくした場合、検出時間は短縮されますが、検出分解能は低くなります。
- 扇ウインドウで 360° 以上を設定した場合、360° 以上の部分の解析を行ないます。

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS

用　例

```
VISSCREEN 1,0,1                ’描画画面 0 番に即時描画します。
VISPLNOUT 0                    ’
VISCLS 0                       ’
WINDMAKE R,1,300,20,0,0        ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 2                        ’カメラ映像を格納メモリに取得します。
VISWORKPLN 0                   ’対象を格納メモリ 0 番に指定します。
VISPLNOUT 0                    ’
VISEDGE 1,100,100,1,0,128      ’エッジを計測します。
WINDDISP 1                     ’
I1 = VISSTATUS(0)              ’
IF I1 = 0 THEN                 ’
 FOR I1 = 0 TO VISSTATUS(1)-1
    VISCROSS VISPOSX(I1),VISPOSY(I1)
 NEXT I1                       ’
 I1 = VISSTATUS(1)             ’
 IF I1 = 0 THEN                ’
  VISLOC 10,10                 ’
  VISPRINT "エッジは見つかりません。"
 END IF                        ’
ELSEIF I1 <> 0 THEN            ’
 VISLOC 10,10                  ’
 VISPRINT "計測できません。"   ’
END IF                         ’
```

## 21.7　コード認識

# VISREADQR（ステートメント）

**機　能**

QR コードを読み取ります。

**書　式**

VISREADQR < ウィンドウ番号 >, <X 座標 >, <Y 座標 >, < モード > [, <2 値下限 > [, <2 値上限 >] ]

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標 >

X 座標を指定します。（0 ～ 511）

<Y 座標 >

Y 座標を指定します。（0 ～ 479）

< モード >

2 値化の方法を指定します。

0：自動 2 値化（判別分析法で処理範囲内を 2 値化します。）

1：指定値で 2 値化（命令中の 2 値化レベルに基づき 2 値化します。）

<2 値下限 >

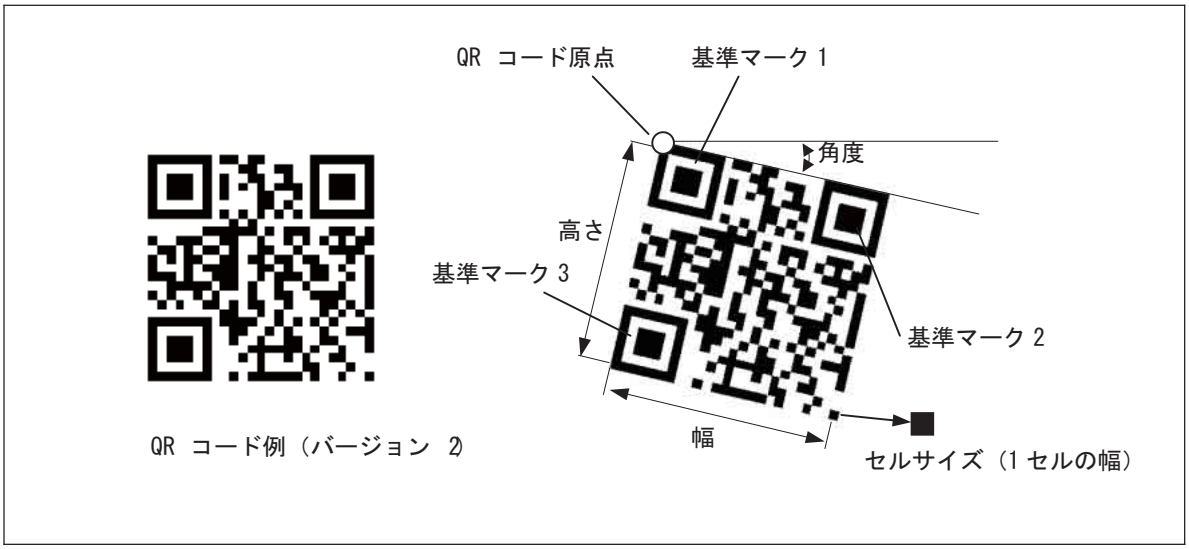
2 値化する際の下限レベルを指定します。（0 ～ 254 下限 < 上限）

省略時 0 を指定します。

<2 値上限 >

2 値化する際の上限レベルを指定します。（1 ～ 255 下限 < 上限）

省略時 255 を指定します。



**注意**

- 処理範囲をウィンドウで指定します。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウまたは、指定したウィンドウの位置が画面からはみ出した場合エラーになります。
- 処理対象は VISWORKPLN で指定した画面になります。
- コードの内容はボード内のメモリに格納され、VISGETSTR で取得することが可能です。
- コードにエラーなどがあり、読み取れなかった場合、VISSTATUS（0）の結果が -1 になります。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | 文字数（バイト単位） |
| 2 | 実行時間 |

| | VISGETNUM（a、b） | | | |
|---|---|---|---|---|
| b | a = 0 | a = 1 | a = 2 | a = 3 |
| 0 | 文字数（バイト単位） | unknown | unknown | unknown |
| 1 | X 座標値 | 基準マーク 1 X 座標値 | 基準マーク 2 X 座標値 | 基準マーク 3 X 座標値 |
| 2 | Y 座標値 | 基準マーク 1 Y 座標値 | 基準マーク 2 Y 座標値 | 基準マーク 3 Y 座標値 |
| 3 | 角度 | unknown | unknown | unknown |
| 4 | 幅 | unknown | unknown | unknown |
| 5 | 高さ | unknown | unknown | unknown |
| 6 | バージョン | unknown | unknown | unknown |
| 7 | セルサイズ | unknown | unknown | unknown |
| 8 | モデル識別 ［V1.4 以降］ | unknown | unknown | unknown |
| 9 | unknown | unknown | unknown | unknown |

- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS、VISGETSTR

用　例

```
VISSCREEN 1,0,1                 '
VISCLS                          '
WINDMAKE R,1,512,480,0,2        'ウィンドウ1番を矩形ウィンドウに設定します。
CAMIN 1                         'カメラ映像を格納メモリに取得します。
VISPLNOUT 0                     '
VISREADQR 1,0,0,0               'QRコードの読み取り
I1 = VISSTATUS(0)               '
VISPRINT I1,VISSTATUS(1)        '
IF I1 = 0 THEN                  '
  VISLOC 10,10                  '
  VISPRINT VISGETSTR(1,VISSTATUS(1))
END IF                          '
VISCAMOUT 1                     '
```

## 21.8　ラベリング

# BLOB（ステートメント）

**機　能**

ラベリングを実行します。

**書　式**

BLOB < ウィンドウ番号 >，<X 座標 >，<Y 座標 >，< 処理対象 >，<2 値下限 >[，<2 値上限 > [，< 連結 > [，< 面積下限 > [，< ソート >]]］]

**説　明**

< ウィンドウ番号 >

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標 >

X 座標を指定します。（0 ～ 511）

<Y 座標 >

Y 座標を指定します。（0 ～ 479）

< 処理対象 >

ラベリングで求める対象を指定します。（0 または 1）

0：黒（輝度値 < 2 値化下限 , 2 値化上限 < 輝度値 ）

1：白（2 値化下限 ≦ 輝度値 ≦ 2 値化上限）

<2 値下限 >

2 値化する際の下限レベルを指定します。（0 ～ 254 下限 < 上限）

<2 値上限 >

2 値化する際の上限レベルを指定します。（1 ～ 255 下限 < 上限）

省略時 255 を指定します。

< 連結 >

連結の条件を指定します。（0 または 1）

0：連結 4 近傍（左右、上下の隣り合う画素の状態をチェック）

1：連結 8 近傍（左右、上下、に斜めも含めた隣り合う画素の状態をチェック）

< 面積下限 >

ラベリングの際無視する面積値の下限を指定します。（0 ～ 245760）

< ソート >

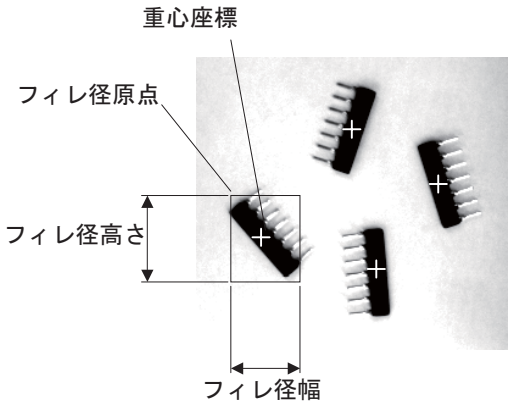
ラベリングで求めた番号の並び替えを指定します。（0 ～ 2）

0：求められた順序

1：面積値降順

2：面積値昇順

ラベリングの例




ラベリングを実行することで複数の物体の数、位置などを求めることができます。
左の例では、ラベリング数は4になります。
また、BLOBMEASURE　でより個別に物体の傾き（方向：主軸角）と周囲長を求めることができます。

**注意**

- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウの場合、エラーになります。
- 処理対象は VISWORKPLN で指定した画面になります。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | ラベル数 |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 ～ max511 |
| 0 | 面積 |
| 1 | 重心 X 座標値 |
| 2 | 重心 Y 座標値 |
| 3 | unknown |
| 4 | フィレ径原点 Y 座標 |
| 5 | フィレ径原点 X 座標 |
| 6 | フィレ径幅 |
| 7 | フィレ径高さ |
| 8 | unknown |
| 9 | unknown |

- 指定時間内にラベリングが終了しない場合、エラーとなります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS

用　例

```
VISSCREEN 1,0,1                ’描画画面 0 番に即時描画します。
VISCLS 0                       ’
WINDMAKE R,1,512,480,0,2       ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                        ’カメラ映像を格納メモリ 0 番に指定します。
VISPLNOUT 0                    ’
VISWORKPLN 0                   ’対象を格納メモリ 0 番に指定します。
BLOB 1,0,0,0,128               ’ラベリングを実行します。
I1 = VISSTATUS(0)              ’
IF I1 = 0 THEN                 ’
  I2 = VISSTATUS(1)            ’
  VISDEFCHAR 1,1,2             ’
  VISLOC 10,10                 ’
  VISPRINT I1,I2               ’
  IF I2 <> 0 THEN              ’
    FOR I1 = 0 TO I2 -1        ’
      VISLOC 10,11             ’
      VISPRINT VISGETNUM(I1,1),VISGETNUM(I1,2)
      VISCROSS VISGETNUM(I1,1),VISGETNUM(I1,2)
    NEXT I1                    ’
  END IF                       ’
END IF                         ’
VISCAMOUT 1                    ’
```

# BLOBMEASURE（ステートメント）

**機　能**

対象ラベル番号の特徴計測を行ないます。

**書　式**

BLOBMEASURE＜ラベル番号＞,＜特徴＞

**説　明**

＜ラベル番号＞

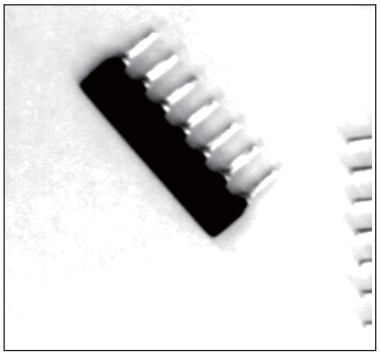
ラベリングで求めたラベル番号を指定します。(0 ～ 511)

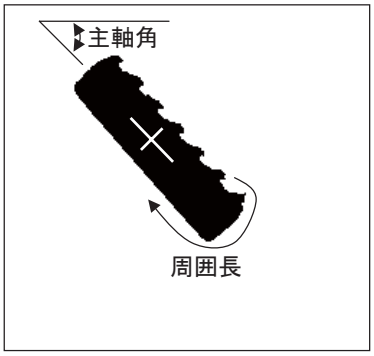
＜特徴＞

求めたい特徴を指定します。(0 または 1)

0：主軸角度

1：周囲長






2値化は元画像を変えることなく内部で自動的に行なっています。
あらかじめ2値化処理する必要はありません。

**注意**

- 本関数を実行する前に、BLOB であらかじめラベリングする必要があります。
- 周囲長を求める場合は、ラベリングを実行した際の元画像を残しておく必要があります。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | unknown |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 ～ max511 |
| 0 | 面積 |
| 1 | 重心 X 座標値 |
| 2 | 重心 Y 座標値 |
| 3 | 主軸角角度 |
| 4 | フィレ径原点 X 座標 |
| 5 | フィレ径原点 Y 座標 |
| 6 | フィレ径幅 |
| 7 | フィレ径高さ |
| 8 | 周囲長 |
| 9 | unknown |

- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

BLOB

**用　例**

```
VISSCREEN 1,0,1                    ' 描画画面 0 番に即時描画します。
WINDMAKE R,1,512,480,0,2           ' ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                            ' カメラ映像を格納メモリに取得します。
BLOB 1,0,0,0,128                   ' ラベリングを実行します。
IF VISSTATUS(0)=0.0 THEN           '
  IF VISSTATUS(1)<>0.0 THEN        '
    FOR I1 = 0 TO VISSTATUS(1)-1   '
      BLOBMEASURE I1,0             'I1 ラベル番号の主軸角度を求めます。
      VISCROSS VISGETNUM(I1,1),VISGETNUM(I1,2),10,20,VISGETNUM(I1,3)
    NEXT I1                        '
  END IF                           '
END IF                             '
```

# BLOBLABEL（ステートメント）

**機　能**

指定座標のラベル番号を取得します。

**書　式**

BLOBLABEL（<X 座標 >，<Y 座標 >）

**説　明**

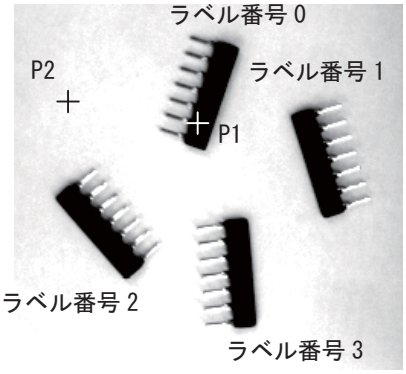
<X 座標 >

X 座標を指定します。（0 ～ 511）

<Y 座標 >

Y 座標を指定します。（0 ～ 479）

**ラベル番号の取得例**




BLOBLABEL　は指定座標が各ラベルの物体に内包されているかを確認します。
ラベリング数4、ラベル番号が左図に示すように計測された画像に対し、BLOBLABEL　でP1の座標を指定した場合、戻り値はラベル番号の0になります。
P2を指定した場合は該当する物体がないので、戻り値は-1になります。

**注意**

- 本関数を実行する前に、BLOB であらかじめラベリングする必要があります。
- 指定した座標に存在するラベル番号を取得します。
- ラベル番号が存在しない場合 -1 を返します。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

BLOB

用　例

```
VISSCREEN 1,0,1                ’描画画面 0 番に即時描画します。
WINDMAKE R,1,512,480,0,2       ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                        ’カメラ映像を格納メモリに取得します。
BLOB 1,0,0,0,128               ’ラベリングを実行します。
IF VISSTATUS(0)=0.0 THEN       ’
  IF VISSTATUS(1)<>0.0 THEN    ’
    IF BLOBLABEL(100,100)<>-1 THEN
      VISLOC 100,100,1         ’
      VISPRINT "ラベル番号 =";BLOBLABEL(100,100)
    END IF                     ’
  END IF                       ’
END IF                         ’
```

# BLOBCOPY（ステートメント）

**機　能**

対象ラベル番号をコピーします。

**書　式**

BLOBCOPY < ラベル番号 >, < コピー先画面 >, <X 座標 >, <Y 座標 >

**説　明**

< ラベル番号 >

ラベリングで求めたラベル番号を指定します。(0 ～ 511)

< コピー先画面 >

コピー先の格納メモリ番号を指定します。(0 ～ 3)

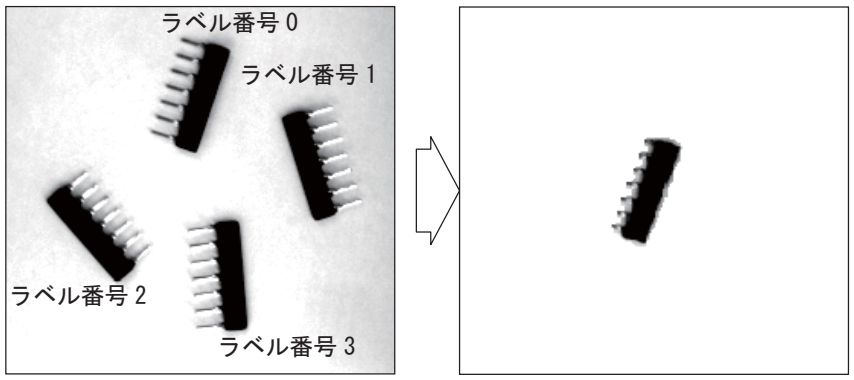
<X 座標 >

コピー先の原点 X 座標を指定します。(0 ～ 511)

<Y 座標 >

コピー先の原点 Y 座標を指定します。(0 ～ 479)

BLOBCOPY の例




指定したラベル番号の物体の画像をコピーします。
左の例ではラベル番号0を処理画面1にコピーした結果です。

**注意**

- 本関数を実行する前に、BLOB であらかじめラベリングする必要があります。
- コピーする際は元画面（ラベリング対象画面）を参照しますので、画面を変更しないでください。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

BLOB

用　例

```
VISSCREEN 0,1,1                 ’格納メモリ 1 番に即時描画します。
VISCLS 128                      ’画面をクリアします。
VISSCREEN 1,0,1                 ’描画画面 0 番に即時描画します。
VISCLS 0                        ’
WINDMAKE R,1,512,480,0,2        ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                         ’カメラ映像を格納メモリに取得します。
BLOB 1,0,0,0,128                ’ラベリングを実行します。
I1 = VISSTATUS(0)               ’
IF I1 = 0 THEN                  ’
  I1 = VISSTATUS(1)             ’
  IF I1<>0 THEN                 ’
    BLOBCOPY 0,1,100,100        ’格納メモリ 1 番にラベル 0 番をコピーします。
  END IF                        ’
END IF                          ’
VISPLNOUT 0                     ’
DELAY 2000                      ’
VISPLNOUT 1                     ’
DELAY 2000                      ’
VISCAMOUT 1                     ’
```

## 21.9　サーチ機能

# SHDEFMODEL（ステートメント）

**機　能**

サーチモデルの登録をします。

**書　式**

SHDEFMODEL＜モデル番号＞，<X 座標＞，<Y 座標＞，＜幅＞，＜高さ＞，＜オフセット X>,
＜オフセット Y>[,＜オフセット角度＞]

**説　明**

＜モデル番号＞

登録するモデルの番号を指定します。（0 ～ 99）

<X 座標＞

登録モデルの原点 X 座標を指定します。（16 ～ 485）

<Y 座標＞

登録モデルの原点 Y 座標を指定します。（16 ～ 463）

＜幅＞

登録モデルの幅を指定します。（10 ～ 256）

＜高さ＞

登録モデルの高さを指定します。（10 ～ 255）

＜オフセット X>

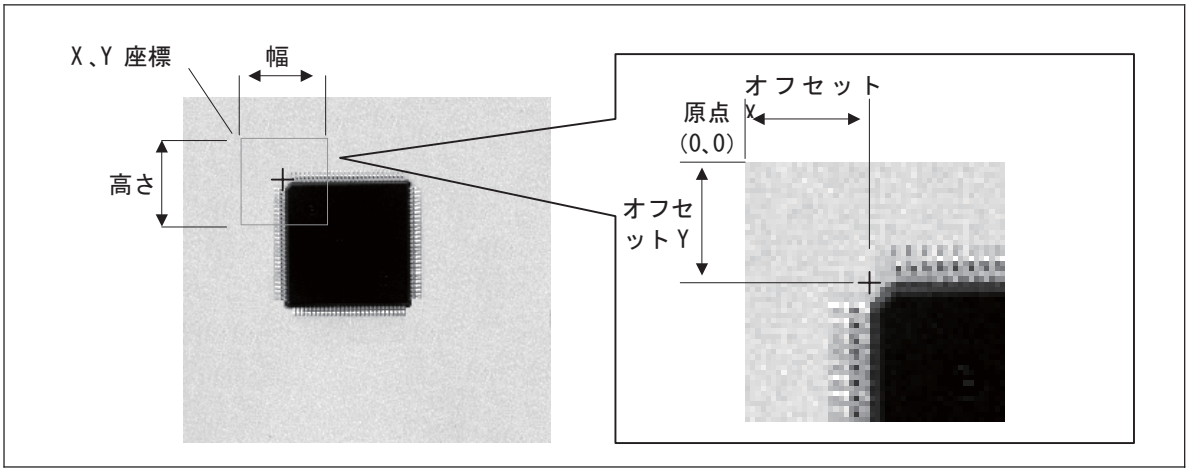
原点からのオフセット X を指定します。（-511 ～ +511）

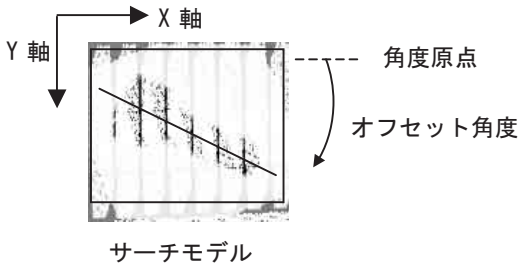
＜オフセット Y>

原点からのオフセット Y を指定します。（-511 ～ +511）

＜オフセット角度＞

角度原点からのオフセット角度を指定します。（-360 ～ 360）［V1.4 以降］






[オフセット角度]には角度原点からのオフセット値を指定します。図の様に、サーチモデルに登録する対象が角度原点に対してオフセットをもつ場合に[オフセット角度]を指定すると[SHMODEL]の計測結果[角度]にオフセット値を反映することができます。

**注意**

- 登録モデルは画面端から内側 16 画素以上離れていなければ登録できません。
- 基準座標はモデルサーチ時に検出する点をどこにするかを指定するものです。
- 登録モデルはある程度の輝度分布と特徴が必要です。あまり輝度分布が平坦すぎるもの、逆に細かい変化が多すぎる場合登録できません。
- すでに登録されている番号を指定した場合上書きします。
- 登録の可否は命令実行後、VISSTATUS により知ることができます。
- 本命令を実行時に処理画面 3 番をワークエリアとして使用するため処理画面 3 番のデータは保証されません。また、処理画面 3 番（VISWORKPLN 3）は処理できません。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果<br>0 = 正常<br>-1 = 指定範囲エラー<br>-2 = スペースフル<br>-3 = 均一モデル<br>-4 = 複雑<br>1,2 = サーチ時間大<br>3,4 = 角度特定不可［V1.4 以降］ |
| 1 | 未使用 0 |
| 2 | 実行時間 |

- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 本命令実行時に電源を OFF しないでください。次の電源 ON 時に正常に終了しなかったと判断し、視覚に関する情報を初期化します。

**関連項目**

SHREFMODEL、SHDISPMODEL、SHCLRMODEL、SHMODEL

用　例

```
VISSCREEN 1,0,1                          ’描画画面 0 番に即時描画します。
VISCLS 0                                 ’
VISRECT 100,100,100,100                  ’
CAMIN 1                                  ’カメラ映像を格納メモリに取得します。
SHDEFMODEL 1,100,100,100,100,50,50       ’モデルを登録します。
I1 = VISSTATUS(0)                        ’
VISLOC 10,10                             ’
IF I1 = 0 THEN VISPRINT "登録完了" ELSE VISPRINT "登録できません。"
VISCAMOUT 1                              ’
```

# SHREFMODEL（ステートメント）

**機　能**

登録モデルデータを参照します。

**書　式**

SHREFMODEL（< モデル番号 >, < 項目 >）

**説　明**

< モデル番号 >

参照モデル番号を指定します。（0 ～ 99）

< 項目 >

参照モデルのデータ種類を指定します。（0 ～ 8）

0：ステータス（モデルの登録有無 有 = 0 無 = -1）
1：登録モデルの幅
2：登録モデルの高さ
3：登録モデルのオフセット X
4：登録モデルのオフセット Y
5：登録モデルのファイルサイズ
6：登録可能ファイルサイズ（残り容量）
7：登録モデルのオフセット角度 ［V1.4 以降］
8：互換モード（1: 角度計測無し用、2: 角度計測有り用、3: 角度計測無し + 有り用）［V1.4 以降］

**関連項目**

SHDEFMODEL

**用　例**

```
VISSCREEN 1,0,1                          '
VISCLS 0                                 '
IF SHREFMODEL(1,0) = 0 THEN              ' 登録の有無を確認します。
 VISLOC 10,1                             '
 VISPRINT " 幅 =";SHREFMODEL(1,1)        ' 幅を表示します。
 VISLOC 10,2                             '
 VISPRINT " 高さ =";SHREFMODEL(1,2)      ' 高さを表示します。
 VISLOC 10,3                             '
 VISPRINT "X=";SHREFMODEL(1,3)           ' 基準 X 座標を表示します。
 VISLOC 10,4                             '
 VISPRINT "Y=";SHREFMODEL(1,4)           ' 基準 Y 座標を表示します。
 VISLOC 10,5                             '
 VISPRINT " サイズ =";SHREFMODEL(1,5)    ' ファイルサイズを表示します。
 VISLOC 10,6                             '
 VISPRINT " 残り =";SHREFMODEL(1,6)      ' 残り容量を表示します。
END IF                                   '
```

# SHCOPYMODEL（ステートメント）

**機　能**

登録モデルをコピーします。

**書　式**

SHCOPYMODEL＜コピー元モデル番号＞,＜コピー先モデル番号＞

**説　明**

＜コピー元モデル番号＞

コピーするモデルの番号を指定します。(0 ～ 99)

＜コピー先モデル番号＞

新しいモデル番号を指定します。(0 ～ 99)

**注意**

- コピー元と先が同じ番号の場合エラーになります。
- コピー元のファイルがない場合、また残り容量が足りない場合、コピーしません。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | unknown |
| 2 | unknown |

- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 本命令実行時に電源を OFF しないでください。次の電源 ON 時に正常に終了しなかったと判断し、視覚に関する情報を初期化します。

**関連項目**

SHDEFMODEL、SHREFMODEL

**用　例**

```
VISSCREEN 1,0,1                ’
VISCLS 0                       ’
SHCOPYMODEL 2,0                ’モデル 2 番の情報を 0 番にコピーします。
VISLOC 10,10                   ’
I1 = VISSTATUS(0)              ’
IF I1 = 0 THEN VISPRINT "コピー完了" ELSE VISPRINT "コピー失敗"
```

# SHCLRMODEL（ステートメント）

**機　能**

登録モデルを消去します。

**書　式**

SHCLRMODEL＜モデル番号＞

**説　明**

＜モデル番号＞

消去するモデル番号を指定します。（0 ～ 99）

**注意**

- 指定したモデル番号が存在しないときは何もしません。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 本命令実行時に電源を OFF しないでください。次の電源 ON 時に正常に終了しなかったと判断し、視覚に関する情報を初期化します。

**関連項目**

SHDEFMODEL、SHREFMODEL

**用　例**

```
VISSCREEN 1,0,1                ’
VISCLS 0                       ’
SHCLRMODEL 1                   ’登録モデル 1 番を消去します。
I1 = SHREFMODEL(1,0)           ’
VISLOC 10,10                   ’
IF I1 = -1 THEN VISPRINT " 消去完了 " ELSE VISPRINT " 消去失敗 "
```

# SHDISPMODEL（ステートメント）

**機　能**

登録モデルを画面に表示します。

**書　式**

SHDISPMODEL < モデル番号 >, < 表示先画面 >, <X 座標 >, <Y 座標 >

**説　明**

< モデル番号 >

表示するモデルの番号を指定します。(0 ～ 99)

< 表示先画面 >

表示する際の格納メモリ番号を指定します。(0 ～ 3)

<X 座標 >

登録モデルの原点 X 座標を指定します。(0 ～ 511)

<Y 座標 >

登録モデルの原点 Y 座標を指定します。(0 ～ 479)

**注意**

- 指定したモデル番号が存在しないときは何もしません。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

SHDEFMODEL

**用　例**

```
VISSCREEN 0,0,1              ' 格納メモリ 0 番に即時描画します。
VISPLNOUT 0                  ' 格納メモリ 0 番をモニタに表示します。
SHDISPMODEL 1,0,100,100      ' モデル 1 番を画面に表示します。
```

# SHMODEL（ステートメント）

**機　能**

モデルをサーチします。

**書　式**

SHMODEL＜ウィンドウ番号＞，<X 座標＞，<Y 座標＞，＜モデル番号＞，＜一致度＞
[，＜モード＞ [，＜個数＞[，＜開始角度＞ [，＜終了角度＞]]]]

**説　明**

＜ウィンドウ番号＞

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標＞

X 座標を指定します。（0 ～ 511）

<Y 座標＞

Y 座標を指定します。（0 ～ 479）

＜モデル番号＞

サーチするモデル番号を指定します。（0 ～ 99）

＜一致度＞

サーチするモデルの一致度を指定します。（0 ～ 100）

＜モード＞

サーチする際のモードを指定します。

0：ピクセル単位のサーチ

1：サブピクセル単位のサーチ

省略時は 0 を指定します。

＜個数＞

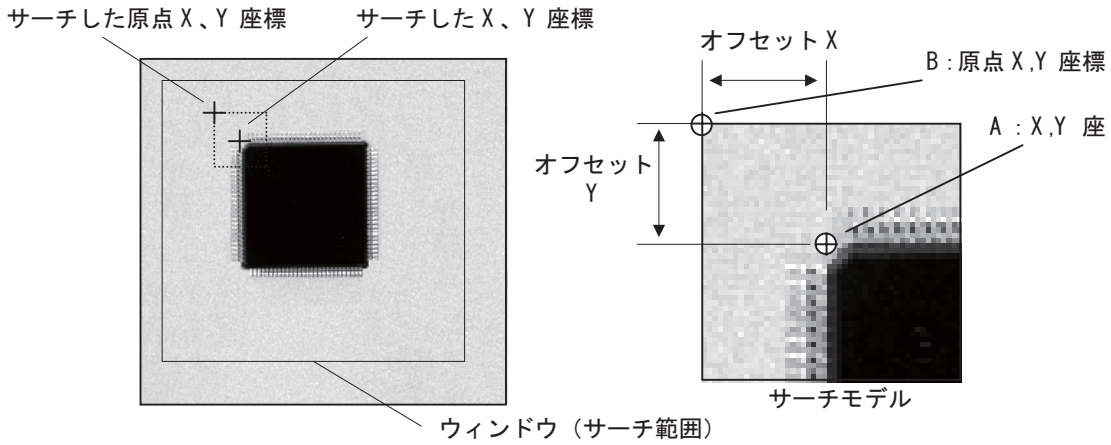
一度にサーチする個数を指定します。（1 ～ 49）省略時は 1 を指定します。

＜開始角度＞

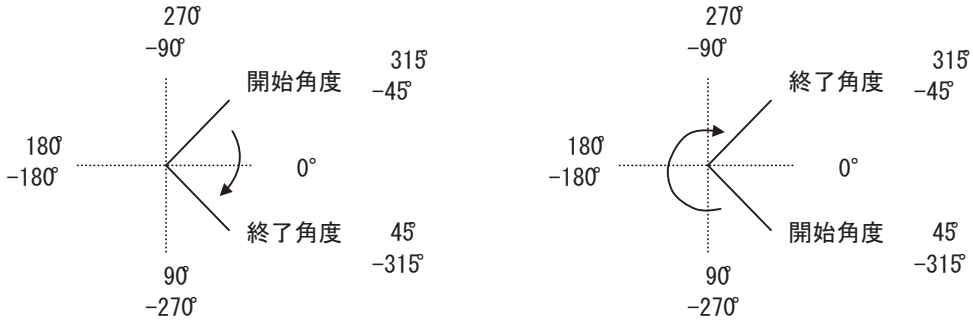
サーチする対象の回転範囲（開始角度）を指定します。（-360 ～ 360）　　省略時は 0 を指定します。
［V1.4 以降］

＜終了角度＞

サーチする対象の回転範囲（終了角度）を指定します。（-360 ～ 360）　　省略時は 0 を指定します。
［V1.4 以降］

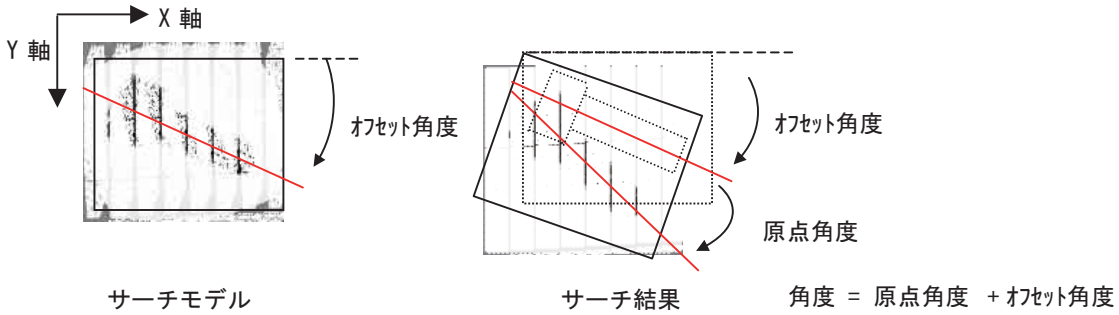



計測結果はサーチモデルのAの位置を［X,Y座標値］に格納します。また、サーチモデルの原点Bの位置を［原点X,Y座標値］に格納します。




［開始角度］から［終了角度］の範囲に原点角度のある対象を角度計測します。左図で示すように(1),(2)どちらの条件でも角度計測の範囲は同じになります。

(1) ［開始角度］=-45°，［終了角度］=45°　(2) ［開始角度］=-45°，［終了角度］=-315°




計測結果［原点角度］にはサーチモデルに登録した画像とのズレ角度を格納します。また、計測結果［角度］にはサーチモデルに登録した［ｵﾌｾｯﾄ角度］を［原点角度］に加算した結果を格納します。

**注意**

- サーチするモデルを SHDEFMODEL で登録する必要があります。
- 個数を 2 以上に設定した場合、設定個数サーチできなければ検出できた結果だけを格納します。
- サーチには時間がかかる場合がありますが、制限時間内にサーチが終了しない場合タイムアウトになります。設定はパソコンで行ないます。（初期値 2 秒）
- 対象画面の状態によっては、一致度を高めに設定するとサーチできない場合があります。そのような場合、一致度を下げるなどの調整をしてください。
- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウの場合、エラーになります。
- 処理対象は VISWORKPLN で指定した画面になります。
- 本命令を実行時に処理画面 3 番をワークエリアとして使用するため処理画面 3 番のデータは保証されません。処理画面 3 番（VISWORKPLN 3）は処理できません。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果<br>0 = 正常<br>-1 = モデル未登録<br>-2 = 失敗<br>-3 = タイムアウト |
| 1 | 個数 |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 ～ 49 |
| 0 | 個数 |
| 1 | X 座標値 |
| 2 | Y 座標値 |
| 3 | 一致度 |
| 4 | 角度［V1.4］ |
| 5 | 原点 X 座標値［V1.4］ |
| 6 | 原点 Y 座標値［V1.4］ |
| 7 | 原点角度［V1.4］ |
| 8 | Unknown |
| 9 | Unknown |

- 本命令は μVision ボード（オプション）が必要です。
- サーチモデルサイズとウィンドウサイズが同じ場合、その位置での一致度のみ計算します。
- [ 開始角度 ]=0，[ 終了角度 ]=0 に設定すると角度計測を行いません。
- [ 開始角度 ] と [ 終了角度 ] を 0 以外の同じ値に設定すると、[ 開始角度 ]=0，[ 終了角度 ]=360 と同じ角度計測を行います

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS、SHDEFMODEL

用　例

```
VISSCREEN 1,0,1                ’描画画面 0 番に即時描画します。
VISCLS 0                       ’
VISPLNOUT 0                    ’
WINDMAKE R,1,512,480,0,2       ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                        ’カメラ映像を格納メモリに取得します。
VISWORKPLN 0                   ’対象を格納メモリ 0 番に指定します。
SHMODEL 1,0,0,1,80             ’モデル 1 番と一致度 80% の部分を画面からサーチしま
                               ’す。
I1 = VISSTATUS(0)              ’
VISLOC 10,10                   ’
VISDEFCHAR 1,1,3               ’
VISPRINT I1                    ’
IF I1 = 0 THEN                 ’
  VISPRINT VISGETNUM(0,1),VISGETNUM(0,2)
  VISCROSS VISPOSX(0),VISPOSY(0)
END IF                         ’
```

# SHDEFCORNER（ステートメント）

**機　能**

コーナーサーチの条件を設定します。

**書　式**

SHDEFCORNER < 距離 >, < 間隔 >, < 幅 >, < 高さ >

**説　明**

< 距離 >

コーナーサーチのパッドのコーナーからの距離を指定します。(1 ～ 10)

< 間隔 >

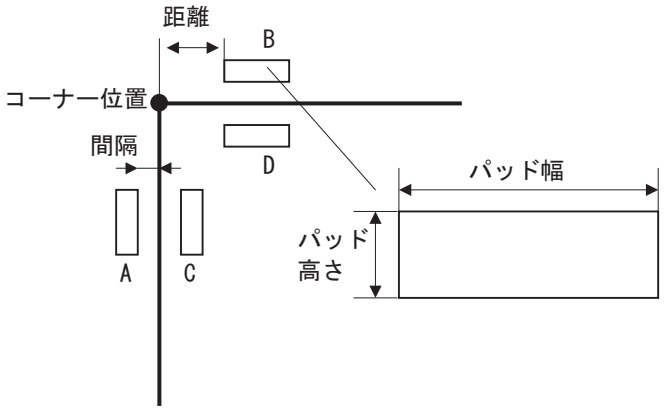
コーナーサーチのパッドの間隔を指定します。(1 ～ 10)

< 幅 >

コーナーサーチのパッドの幅を指定します。(1 ～ 10)

< 高さ >

コーナーサーチのパッドの高さを指定します。(1 ～ 10)




コーナーの検出方法は左図に示す4つのパッドの平均輝度で求めます。
つまり、SHCORNER　のモードが1の場合、A+B の平均輝度がC ＋D の平均輝度よりSHCORNER　のレベル差以上大きい場合コーナーとして検出します。
SHDEFCORNER　はそのパッドの位置と大きさを変更することができます。

**注意**

- 本命令は一時設定であり、初期値の恒久的変更は伴いません。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 間隔を大きくすれば、検出の際外乱に強くなりますが、検出精度は低くなります。
- 工場出荷時の初期値は、距離 =4、間隔 =1、幅 =3、高さ =3 に設定されています。

**関連項目**

SHCORNER

**用　例**

```
SHDEFCORNER 4,1,3,3          '
```

# SHCORNER（ステートメント）

**機　能**

コーナーをサーチします。

**書　式**

SHCORNER ＜ウィンドウ番号＞，<X 座標＞，<Y 座標＞，＜レベル差＞，＜モード＞

**説　明**

＜ウィンドウ番号＞

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標＞

X 座標を指定します。（0 ～ 511）

<Y 座標＞

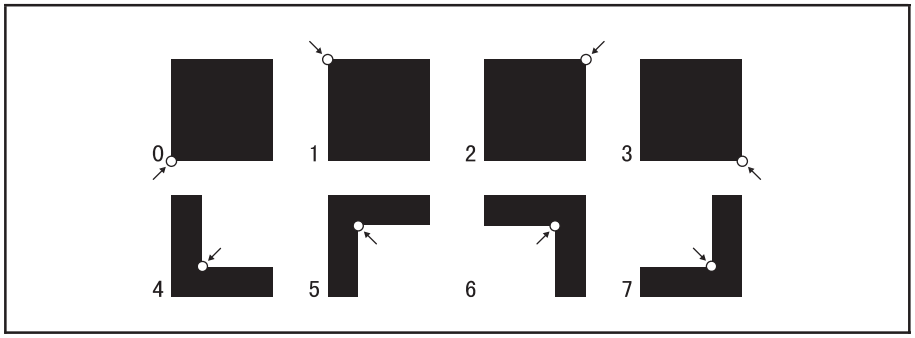
Y 座標を指定します。（0 ～ 479）

＜レベル差＞

コーナーサーチの検出レベルを指定します。（0 ～ 255）

＜モード＞

コーナーサーチの対象を指定します。（0 ～ 7）



**注意**

- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウの場合、エラーになります。
- 処理対象は VISWORKPLN で指定した画面になります。
- 本命令を実行時に処理画面 3 番をワークエリアとして使用するため処理画面 3 番のデータは保証されません。また、処理画面 3 番（VISWORKPLN 3）は処理できません。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | 個数 |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 ～ 511 |
| 0 | 個数 |
| 1 | X 座標値 |
| 2 | Y 座標値 |
| 3 | Unknown |
| 4 | レベル |
| 5 | Unknown |
| 6 | Unknown |
| 7 | Unknown |
| 8 | Unknown |
| 9 | Unknown |

- 検出中にタイムアウトが起こった場合、それまでに検出したコーナー数となります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS、SHDEFCORNER

用　例

```
VISSCREEN 1,0,1                ’描画画面 0 番に即時描画します。
VISCLS 0                       ’
WINDMAKE R,1,512,480,0,2       ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                        ’カメラ映像を格納メモリに取得します。
VISWORKPLN 0                   ’対象を格納メモリ 0 番に指定します。
SHCORNER 1,0,0,180,0           ’黒の左下コーナーをサーチします。
I1 = VISSTATUS(0)              ’
IF I1 = 0 THEN                 ’
  I2 = VISSTATUS(1)            ’
  IF I2 <> 0 THEN              ’
    FOR I1 = 0 TO I2-1         ’
      VISCROSS VISGETNUM(I1,1),VISGETNUM(I1,2)
    NEXT I1                    ’
  END IF                       ’
END IF                         ’
```

# SHDEFCIRCLE（ステートメント）

**機　能**

円サーチの条件を設定します。

**書　式**

SHDEFCIRCLE＜間隔＞，＜幅＞

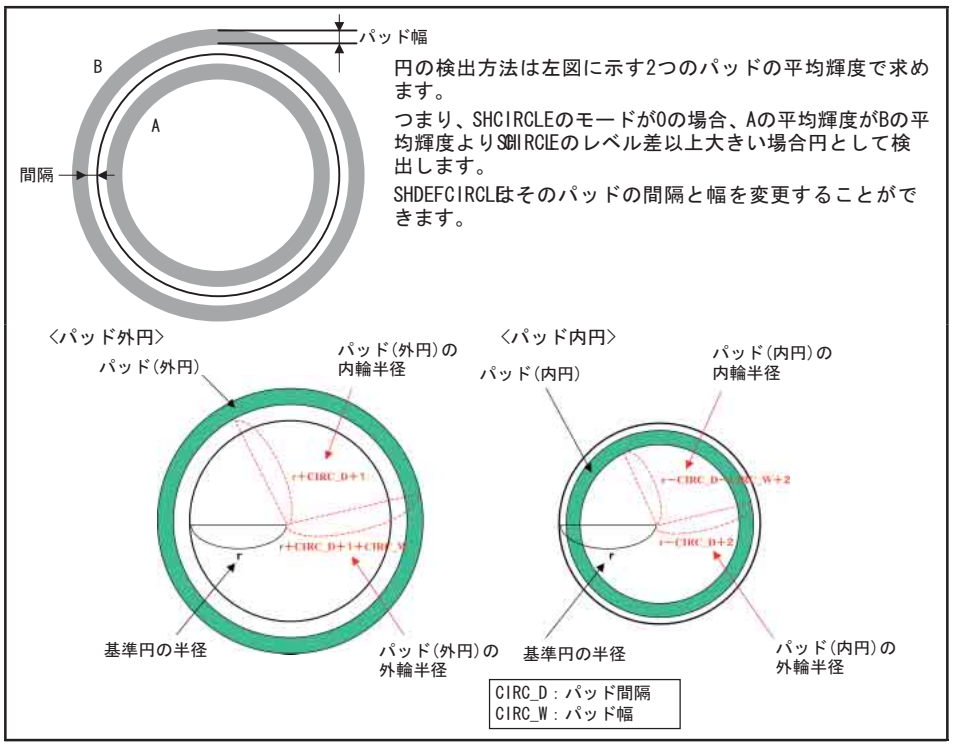
**説　明**

＜間隔＞

円サーチのパッドの間隔を指定します。（1 ～ 10）

＜幅＞

円サーチのパッドの幅を指定します。（1 ～ 10）




**注意**

- 本命令は一時設定であり、初期値の恒久的変更は伴いません。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 間隔を大きくすれば、検出の際外乱に強くなりますが、検出精度は低くなります。
- 工場出荷時の初期値は、間隔 =1、幅 =3 に設定されています。

**関連項目**

SHCIRCLE

**用　例**

```
SHDEFCIRCLE 1,3
```

# SHCIRCLE（ステートメント）

**機　能**

円をサーチします。

**書　式**

SHCIRCLE ＜ウィンドウ番号＞，<X 座標＞，<Y 座標＞，＜半径＞，＜レベル差＞，＜モード＞

**説　明**

＜ウィンドウ番号＞

ウィンドウの番号を指定します。（0 ～ 511）

<X 座標＞

X 座標を指定します。（0 ～ 511）

<Y 座標＞

Y 座標を指定します。（0 ～ 479）

＜半径＞

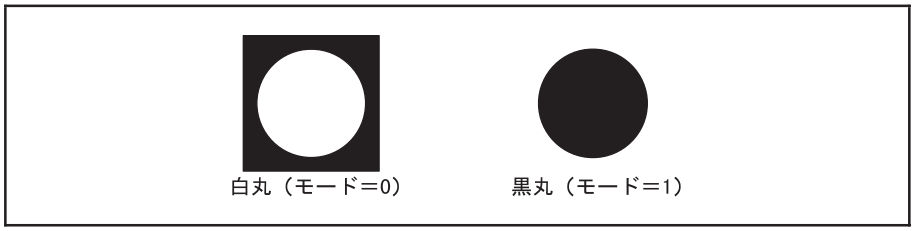
サーチする円の半径を指定します。（3 ～ 240）

＜レベル差＞

円サーチの検出レベルを指定します。（0 ～ 255）

＜モード＞

円サーチの対象を指定します。（0 または 1）




**注意**

- 処理範囲をウィンドウで指定します。
- 指定したウィンドウの位置が画面をはみ出す場合、実行結果はエラーになります。
- 指定できるウィンドウの形状は矩形の角度 0° のみです。その他のウィンドウの場合、エラーになります。
- 処理対象は VISWORKPLN で指定した画面になります。
- 本命令を実行時に処理画面 3 番をワークエリアとして使用するため処理画面 3 番のデータは保証されません。また、処理画面 3 番（VISWORKPLN 3）は処理できません。
- 以下は処理結果取得関数で得ることができるデータです。

| | VISSTATUS (n) |
|---|---|
| n | 項目 |
| 0 | 実行結果　0= 正常　-1= 異常 |
| 1 | 個数 |
| 2 | 実行時間 |

| | VISGETNUM（a、b） |
|---|---|
| b | a = 0 ～ 511 |
| 0 | 個数 |
| 1 | X 座標値 |
| 2 | Y 座標値 |
| 3 | Unknown |
| 4 | レベル |
| 5 | Unknown |
| 6 | Unknown |
| 7 | Unknown |
| 8 | Unknown |
| 9 | Unknown |

- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。
- 指定したウィンドウのサーチ範囲が大きくかつ半径が小さい場合、処理時間が増大しタイムアウトエラーになる場合があります。

**関連項目**

WINDMAKE、VISWORKPLN、VISGETNUM、VISSTATUS、SHDEFCIRCLE

**用　例**

```
VISSCREEN 1,0,1              ’描画画面 0 番に即時描画します。
VISCLS 0
WINDMAKE R,1,512,480,0,2     ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                      ’カメラ映像を格納メモリに取得します。
VISWORKPLN 0                 ’対象を格納メモリ 0 番に指定します。
SHCIRCLE 1,0,0,30,128,1      ’黒の円をサーチします。
I1 = VISSTATUS(0)            ’
VISLOC 10,10                 ’
VISPRINT I1                  ’
IF I1 = 0 THEN               ’
  I2 = VISSTATUS(1)          ’
  IF I2 <> 0 THEN            ’
    FOR I1 = 0 TO I2-1       ’
      VISCROSS VISGETNUM(I1,1),VISGETNUM(I1,2)
    NEXT I1                  ’
  END IF                     ’
END IF                       ’
```

## 21.10 結果取得

# VISGETNUM（関数）

**機　能**

画像処理結果を格納メモリから取得します。

**書　式**

VISGETNUM（＜パラメータ 1>, ＜パラメータ 2>）

**説　明**

＜パラメータ 1>

取得する処理結果の番号を指定します。（0 ～ 511）

＜パラメータ 2>

取得する処理結果の種類を指定します。（0 ～ 9）
得られる値は単精度実数型定数（F 型）です。

**注意**

- 処理結果は事前に行なった画像処理命令によって格納されている結果の内容が異なります。画像処理の命令を参照してください。
- 結果は次の画像処理命令が実行されるまで保持されていますので何度でも取得できます。
- 事前の画像処理命令では意図していない格納先を指定した場合の値は未知数になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISMEASURE、VISPROJ、VISEDGE、VISREADQR、BLOB、BLOBMEASURE、SHMODEL、SHCORNER、SHCIRCLE、VISPOSX、VISPOSY

**用　例**

```
VISSCREEN 1,0,1              ’描画画面 0 番に即時描画します。
VISCLS 0                     ’
WINDMAKE R,1,512,480,0,2     ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                      ’カメラ映像を格納メモリに取得します。
VISPLNOUT 0                  ’
VISWORKPLN 0                 ’対象を格納メモリ 0 番に指定します。
BLOB 1,0,0,0,128             ’ラベリングを実行します。
I1 = VISSTATUS(0)            ’
IF I1 = 0 THEN               ’
  I2 = VISSTATUS(1)          ’
  IF I2 <> 0 THEN            ’
    FOR I1 = 0 TO 15         ’
      VISDEFCHAR 1,1,0       ’
      VISLOC 10,10+I1        ’
      VISPRINT "X=";VISGETNUM(I1,1),VISGETNUM(I1,2)
    NEXT I1                  ’
  END IF                     ’
END IF                       ’
```

# VISGETSTR（関数）

**機　能**

コード認識結果を取得します。

**書　式**

VISGETSTR（< 先頭文字番号 >, < 文字数 >）

**説　明**

< 先頭文字番号 >

取得する文字の先頭番号を指定します。（1 ～ 611）

< 文字数 >

取得する文字数を指定します。（1 ～ 240）

**注意**

- 取得する文字の長さが未知である場合、各命令の処理結果を参照してください。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISREADQR、VISGETNUM

**用　例**

```
VISSCREEN 1,0,1                ’
VISCLS 0                       ’
WINDMAKE R,1,512,480,0,2       ’ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                        ’カメラ映像を格納メモリに取得します。
VISPLNOUT 0                    ’
VISREADQR 1,0,0,0              ’QR コードの読み取り
I1 = VISSTATUS(0)              ’
VISPRINT I1,VISSTATUS(1)       ’
IF I1 = 0 THEN                 ’
 VISLOC 10,10                  ’
 VISPRINT VISGETSTR(1,VISSTATUS(1))
END IF                         ’
```

# VISPOSX（関数）

**機　能**

画像処理結果（X 座標）を格納メモリから取得します。

**書　式**

VISPOSX (< パラメータ >)

**説　明**

< パラメータ >

取得する処理結果（X 座標）の番号を指定します。（0 ～ 511）
得られる値は単精度実数型定数（F 型）です。

**注意**

- VISGETNUM のパラメータ 1 を 1 に指定した場合と同じです。
  VISGETNUM（n，1）= VISPOSX（n）
- 処理結果は事前に行なった画像処理命令によって格納されている結果の内容が異なります。画像処理の命令を参照してください。
- 結果は次の画像処理命令が実行されるまで保持されていますので何度でも取得できます。
- 事前の画像処理命令では意図していない格納先を指定した場合の値は未知数になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISMEASURE、VISPROJ、VISEDGE、VISREADQR、BLOB、BLOBMEASURE、SHMODEL、SHCORNER、SHCIRCLE、VISGETNUM、VISPOSY

**用　例**

```
VISSCREEN 1,0,1              ' 描画画面 0 番に即時描画します。
VISCLS 0                     '
WINDMAKE R,1,512,480,0,2     ' ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                      ' カメラ映像を格納メモリに取得します。
VISPLNOUT 0                  '
VISWORKPLN 0                 ' 対象を格納メモリ 0 番に指定します。
BLOB 1,0,0,0,128             ' ラベリングを実行します。
I1 = VISSTATUS(0)            '
IF I1 = 0 THEN               '
  I2 = VISSTATUS(1)          '
  IF I2 <> 0 THEN            '
    FOR I1 = 0 TO 15         '
      VISDEFCHAR 1,1,0       '
      VISLOC 10,10+I1        '
      VISPRINT "X=";VISPOSX(I1),"Y=";VISPOSY(I1)
      VISCROSS VISPOSX(I1),VISPOSY(I1)
    NEXT I1                  '
  END IF                     '
END IF                       '
VISCAMOUT 1                  '
```

# VISPOSY（関数）

**機　能**

画像処理結果（Y 座標）を格納メモリから取得します。

**書　式**

VISPOSY (< パラメータ >)

**説　明**

< パラメータ >

取得する処理結果（Y 座標）の番号を指定します。（0 ～ 511）
得られる値は単精度実数型定数（F 型）です。

**注意**

- VISGETNUM のパラメータ 1 を 2 に指定した場合と同じです。
  VISGETNUM（n，2）= VISPOSY（n）
- 処理結果は事前に行なった画像処理命令によって格納されている結果の内容が異なります。画像処理の命令を参照してください。
- 結果は次の画像処理命令が実行されるまで保持されていますので何度でも取得できます。
- 事前の画像処理命令では意図していない格納先を指定した場合の値は未知数になります。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISMEASURE、VISPROJ、VISEDGE、VISREADQR、BLOB、BLOBMEASURE、SHMODEL、SHCORNER、SHCIRCLE、VISGETNUM、VISPOSX

**用　例**

```
VISSCREEN 1,0,1              ' 描画画面 0 番に即時描画します。
VISCLS 0                     '
WINDMAKE R,1,512,480,0,2     ' ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                      ' カメラ映像を格納メモリに取得します。
VISPLNOUT 0                  '
VISWORKPLN 0                 ' 対象を格納メモリ 0 番に指定します。
BLOB 1,0,0,0,128             ' ラベリングを実行します。
I1 = VISSTATUS(0)            '
IF I1 = 0 THEN               '
  I2 = VISSTATUS(1)          '
  IF I2 <> 0 THEN            '
    FOR I1 = 0 TO 15         '
      VISDEFCHAR 1,1,0       '
      VISLOC 10,10+I1        '
      VISPRINT "X=";VISPOSX(I1),"Y=";VISPOSY(I1)
      VISCROSS VISPOSX(I1),VISPOSY(I1)
    NEXT I1                  '
  END IF                     '
END IF                       '
VISCAMOUT 1                  '
```

# VISSTATUS（関数）

**機　能**

各命令の処理結果をモニタします。

**書　式**

VISSTATUS(<パラメータ>)

**説　明**

<パラメータ>

取得するデータを指定します。(0 ～ 2)

0：実行結果状態

1：補助データ

2：処理時間

得られる値は単精度実数型定数（F 型）です。

**注意**

- 取得した結果は直前に実行した命令により異なります。各命令の説明を参照してください。
- 処理時間はすべての命令が対象になります。直前に実行した命令の処理時間を得られます。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

VISMEASURE、VISPROJ、VISEDGE、VISREADQR、BLOB、BLOBMEASURE、SHMODEL、SHCORNER、SHCIRCLE

**用　例**

```
VISSCREEN 1,0,1              '描画画面 0 番に即時描画します。
VISCLS 0                     '
WINDMAKE R,1,512,480,0,2     'ウィンドウ 1 番を矩形ウィンドウに設定します。
CAMIN 1                      'カメラ映像を格納メモリに取得します。
VISPLNOUT 0                  '
VISWORKPLN 0                 '対象を格納メモリ 0 番に指定します。
BLOB 1,0,0,0,128             'ラベリングを実行します。
I1 = VISSTATUS(0)            '
I2 = VISSTATUS(1)            '
F1 = VISSTATUS(2)            '
VISLOC 10,9                  '
VISPRINT "実行結果 =";I1;" ラベリング数 =";I2," 実行時間 ( 秒 )=";F1
IF I1 = 0 THEN               '
  IF I2 <> 0 THEN            '
    FOR I1 = 0 TO I2 -1      '
      VISCROSS VISGETNUM(I1,1),VISGETNUM(I1,2)
    NEXT I1                  '
  END IF                     '
END IF                       '
VISCAMOUT 1                  '
```

# VISREFCAL（関数）

**機　能**

CAL（視覚－ロボット座標変換）データを取得します。

**書　式**

VISREFCAL（<セット番号>, <データ番号>）

**説　明**

<セット番号>

何番の CAL データ群を使用するかを指定します。（0 ～ 31）

<データ番号>

CAL データ番号を指定します。

［Ver. 1.5 以前：0 ～ 11］

［Ver. 1.6 以前：0 ～ 38］

| データ番号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 内容 | NX | OX | AX | RX | NY | OY | AY | RY | NZ | OZ | AZ | RZ |

| データ番号 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|
| 内容 | Vx1 | Vy1 | Vz1 | Vx2 | Vy2 | Vz2 | Vx2 | Vy2 | Vz2 |

| データ番号 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 内容 | Rx1 | Ry1 | Rz1 | Rrx1 | Rry1 | Rrz1 | Rx2 | Ry2 | Rz2 | Rrx2 | Rry2 | Rrz2 |

| データ番号 | 33 | 34 | 35 | 36 | 37 | 38 |
|---|---|---|---|---|---|---|
| 内容 | Rx3 | Ry3 | Rz3 | Rrx3 | Rry3 | Rrz3 |

0-11：変換テーブルデータ
12-20：視覚座標データ
21-38：ロボット座標データ

**注意**

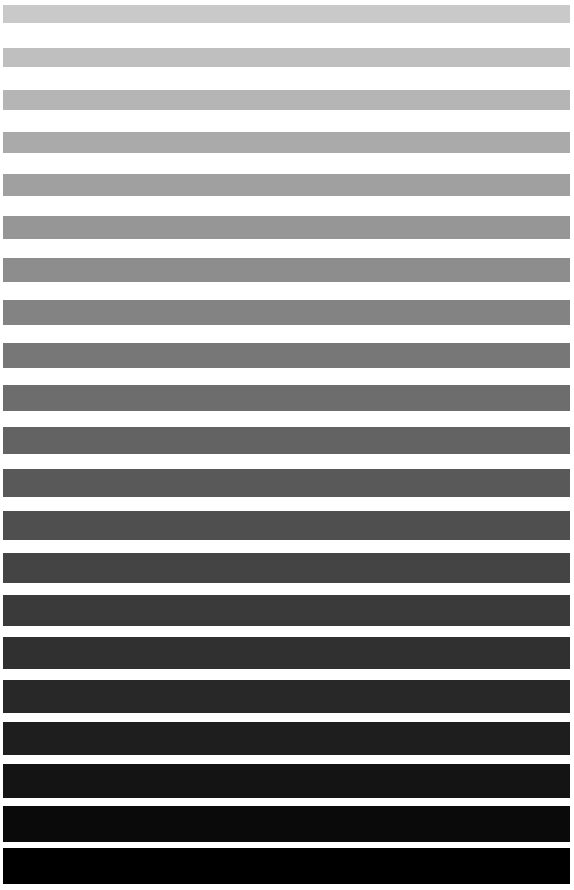
- CAL データ群は 32 セット記憶することができます。
- CAL データの設定はパソコンから行ないます。
- 本命令はロボットコントローラでは μVision ボード（オプション）が必要です。

**関連項目**

**用　例**

```
VISSCREEN 1,0,1                ’格納メモリ 0 番に即時描画します。
VISCLS 0                       ’
FOR I1 = 0 TO 11               ’
  VISLOC 10,10+I1              ’
  VISPRINT "Data";I1;"=";VISREFCAL(0,I1)
NEXT I1                        ’
```

# 第 22章 付録



# 第 22 章　付録

## 22.1　文字コード表

表付録1-1　文字コード表

| | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 | +A | +B | +C | +D | +E | +F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | |
| 20 | | ! | ” | # | $ | ～ | & | ’ | ( | ) | * | + | , | − | . | / |
| 30 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 40 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 50 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | ¥ | ] | ^ | _ |
| 60 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 70 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ～ | ・ |
| 80 | | | | | | | | | | | | | | | | |
| 90 | | | | | | | | | | | | | | | | |
| A0 | ・ | 。 | 「 | 」 | 、 | ・ | ヲ | ァ | ィ | ゥ | ェ | ォ | ャ | ュ | ョ | ッ |
| B0 | − | ア | イ | ウ | エ | オ | カ | キ | ク | ケ | コ | サ | シ | ス | セ | ソ |
| C0 | タ | チ | ツ | テ | ト | ナ | ニ | ヌ | ネ | ノ | ハ | ヒ | フ | ヘ | ホ | マ |
| D0 | ミ | ム | メ | モ | ヤ | ユ | ヨ | ラ | リ | ル | レ | ロ | ワ | ン | ゛ | ° |
| E0 | | | | | | | | | | | | | | | | |
| F0 | | | | | | | | | | | | | | | | |

# 22.2　ロボット形態

(1) 16種類の形態

6軸ロボットは、ツール先端の1つの位置と姿勢（X、Y、Z、RX、RY、RZ）に対して、以下に説明するように、腕・ひじ・手首・第6軸・第4軸において違った形態をとることができます。

図付録2-1～2-5に腕・ひじ・手首・第6軸・第4軸に対する、それぞれの形態の違いを説明します。

これらの形態を組み合わせると、1つの位置と姿勢に対して、32種類の形態をとることがわかります。この形態の組み合わせを、表付録2-1に示します。また、図付録2-6にVSシリーズロボットの、腕、ひじ、手首形態の8種類の組み合わせ例を示します。

表付録2-1　ロボット形態

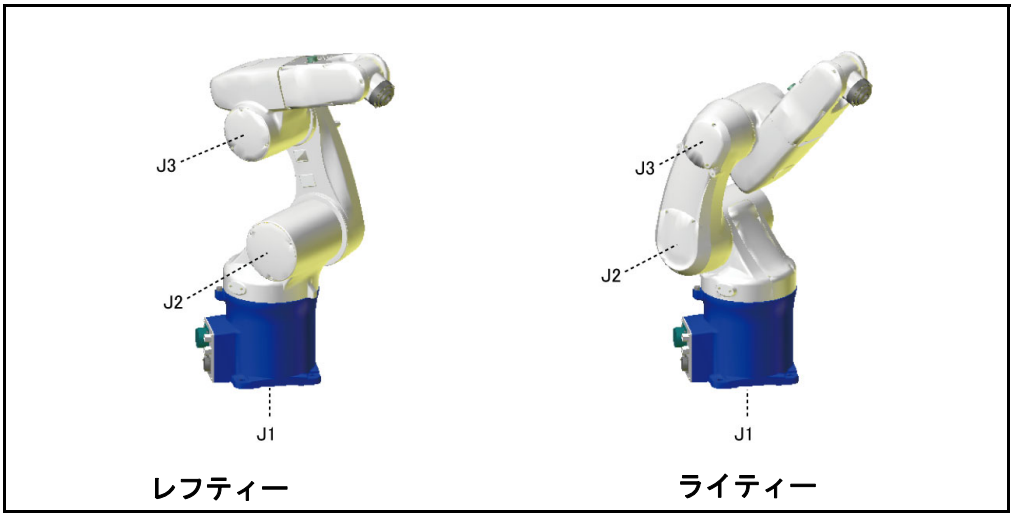
| 値 | 第4軸形態 | 第6軸形態 | 手首形態 | ひじ形態 | 腕形態 |
|---|---|---|---|---|---|
| 0 | SINGLE 4 | SINGLE | FLIP | ABOVE | RIGHTY |
| 1 | SINGLE 4 | SINGLE | FLIP | ABOVE | LEFTY |
| 2 | SINGLE 4 | SINGLE | FLIP | BELOW | RIGHTY |
| 3 | SINGLE 4 | SINGLE | FLIP | BELOW | LEFTY |
| 4 | SINGLE 4 | SINGLE | NONFLIP | ABOVE | RIGHTY |
| 5 | SINGLE 4 | SINGLE | NONFLIP | ABOVE | LEFTY |
| 6 | SINGLE 4 | SINGLE | NONFLIP | BELOW | RIGHTY |
| 7 | SINGLE 4 | SINGLE | NONFLIP | BELOW | LEFTY |
| 8 | SINGLE 4 | DOUBLE | FLIP | ABOVE | RIGHTY |
| 9 | SINGLE 4 | DOUBLE | FLIP | ABOVE | LEFTY |
| 10 | SINGLE 4 | DOUBLE | FLIP | BELOW | RIGHTY |
| 11 | SINGLE 4 | DOUBLE | FLIP | BELOW | LEFTY |
| 12 | SINGLE 4 | DOUBLE | NONFLIP | ABOVE | RIGHTY |
| 13 | SINGLE 4 | DOUBLE | NONFLIP | ABOVE | LEFTY |
| 14 | SINGLE 4 | DOUBLE | NONFLIP | BELOW | RIGHTY |
| 15 | SINGLE 4 | DOUBLE | NONFLIP | BELOW | LEFTY |
| 16 | DOUBLE 4 | SINGLE | FLIP | ABOVE | RIGHTY |
| 17 | DOUBLE 4 | SINGLE | FLIP | ABOVE | LEFTY |
| 18 | DOUBLE 4 | SINGLE | FLIP | BELOW | RIGHTY |
| 19 | DOUBLE 4 | SINGLE | FLIP | BELOW | LEFTY |
| 20 | DOUBLE 4 | SINGLE | NONFLIP | ABOVE | RIGHTY |
| 21 | DOUBLE 4 | SINGLE | NONFLIP | ABOVE | LEFTY |
| 22 | DOUBLE 4 | SINGLE | NONFLIP | BELOW | RIGHTY |
| 23 | DOUBLE 4 | SINGLE | NONFLIP | BELOW | LEFTY |
| 24 | DOUBLE 4 | DOUBLE | FLIP | ABOVE | RIGHTY |
| 25 | DOUBLE 4 | DOUBLE | FLIP | ABOVE | LEFTY |
| 26 | DOUBLE 4 | DOUBLE | FLIP | BELOW | RIGHTY |
| 27 | DOUBLE 4 | DOUBLE | FLIP | BELOW | LEFTY |
| 28 | DOUBLE 4 | DOUBLE | NONFLIP | ABOVE | RIGHTY |
| 29 | DOUBLE 4 | DOUBLE | NONFLIP | ABOVE | LEFTY |
| 30 | DOUBLE 4 | DOUBLE | NONFLIP | BELOW | RIGHTY |
| 31 | DOUBLE 4 | DOUBLE | NONFLIP | BELOW | LEFTY |

①腕形態

腕は、第1軸、第2軸、第3軸の値で形態が決まります。

腕形態は、左腕系レフティー（LEFTY）、右腕系ライティー（RIGHTY）の2種類をとることができます。

（J1～J6は各軸を示します。）



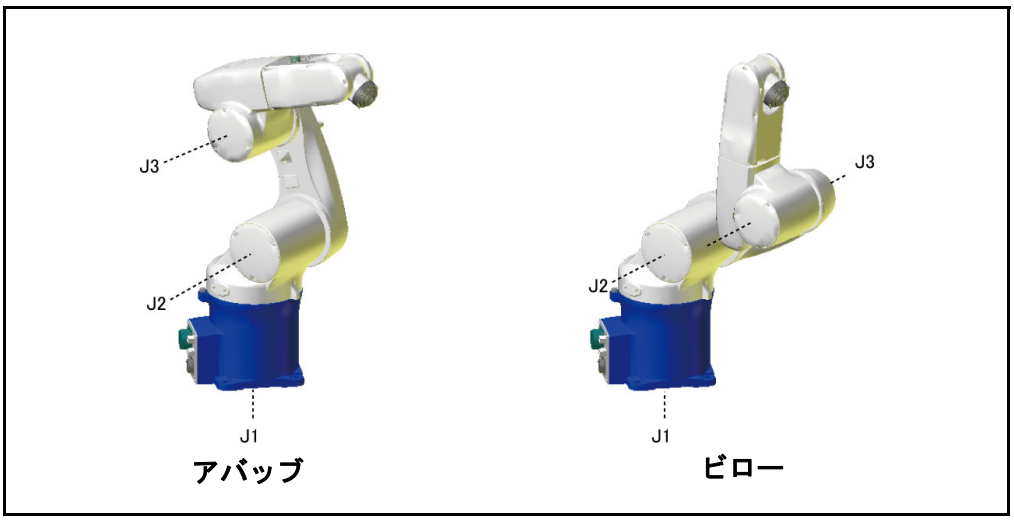

図付録2-1　腕形態

②ひじ形態

ひじは、第2軸、第3軸の値で形態が決まります。

ひじの形態はアバッブ（ABOVE）とビロー（BELOW）の2種類をとることができます。

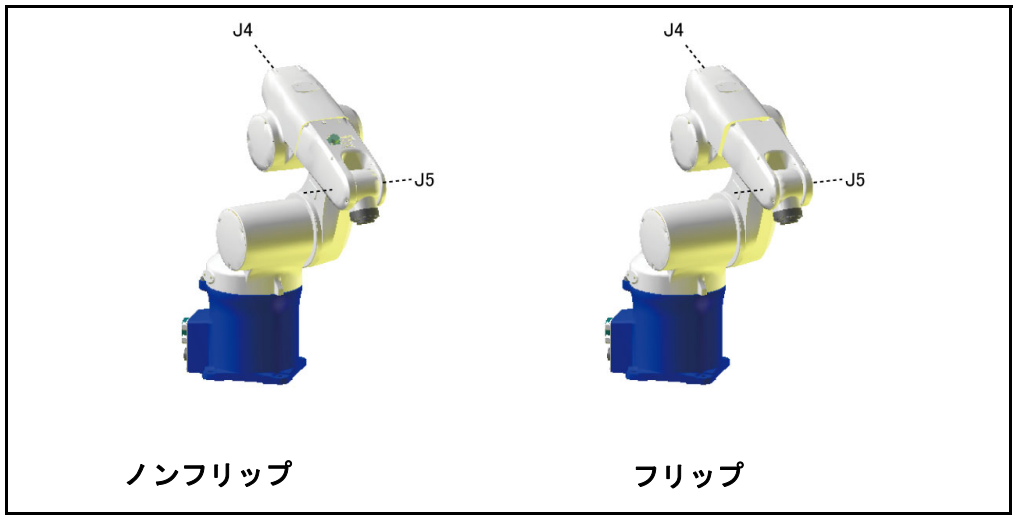
（J1～J6は各軸を示します。）




図付録2-2　ひじ形態

**③手首形態**

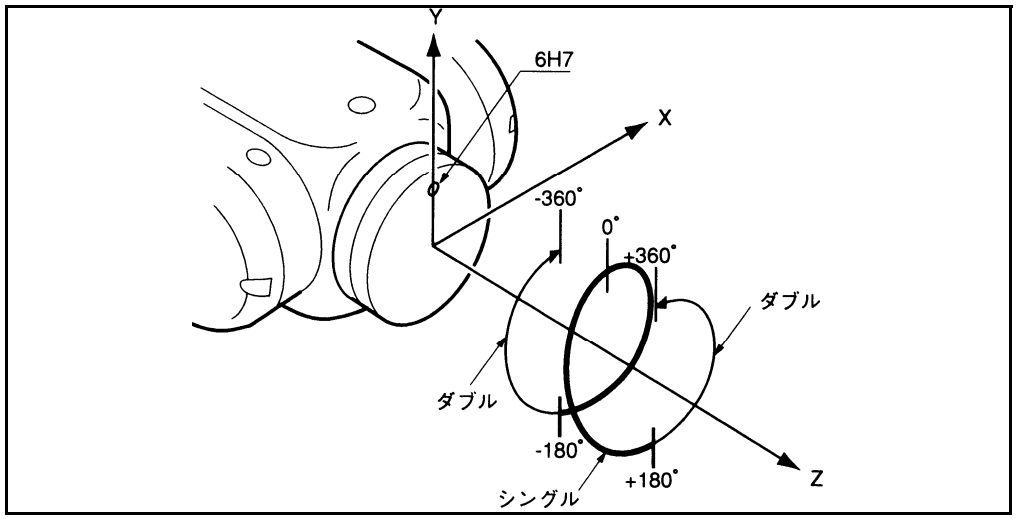
手首は、第4軸、第5軸の値で形態が決まります。
手首の形態はフリップ（FLIP）とノンフリップ（NONFLIP）の2種類をとることができます。
ノンフリップは、フリップの形態から手首部の姿勢を変えずに第4軸を180度回転させた形態です。
（J1～J6は各軸を示します。）




図付録2-3　手首形態

**④第6軸形態**

第6軸形態は、第6軸の値で形態が決まります。
第6軸の形態はシングル（SINGLE）とダブル（DOUBLE）の2種類をとることができます。
第6軸の回転角 $\theta_6$ が（$-180 < \theta_6 \leqq 180$）の場合がシングルで、（$180 < \theta_6 \leqq 360$ または $-360 < \theta_6 \leqq -180$）の場合がダブルです。
$\theta_6 = 180°$ と $\theta_6 = 181°$ では形態が変わります。位置データを数値変更する場合、形態の設定にご注意ください。（$\theta_6 = 181°$ に修正する予定で、形態が修正されないと $\theta_6 = -179°$ になります。）




図付録2-4　第6軸形態

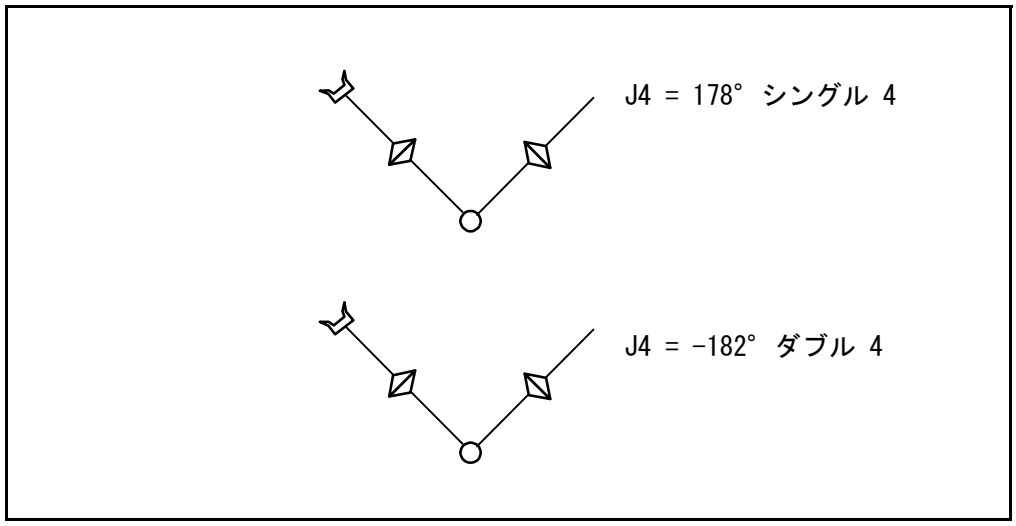
**⑤第4軸形態**

第4軸形態は、第4軸の値で形態が決まります。

第4軸の形態はシングル4（SINGLE 4）とダブル4（DOUBLE）の2種類をとることができます。

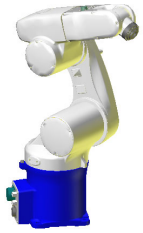
第4軸の回転角$\theta_4$が（-180＜$\theta_4$≦180）の場合がシングル4で、（180＜$\theta_4$≦185または-185＜$\theta_4$≦-180）の場合がダブル4です。

$\theta_4$＝180°と$\theta_4$＝181°では形態が変わります。位置データを数値変更する場合、形態の設定にご注意ください。（$\theta_4$＝181°に修正する予定で、形態が修正されないと$\theta_4$＝-179°になります。）

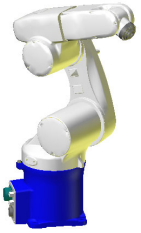



図付録2-5　第4軸形態

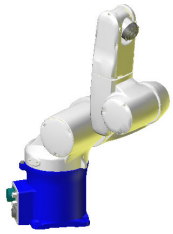
形態の種類1
「レフティー・アバッブ・ノンフリップ」



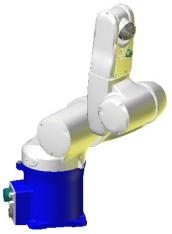
形態の種類2
「レフティー・アバッブ・フリップ」



形態の種類3
「レフティー・ビロー・ノンフリップ」



形態の種類4
「レフティー・ビロー・フリップ」



形態の種類5
「ライティー・アバッブ・ノンフリップ」



形態の種類6
「ライティー・アバッブ・フリップ」



形態の種類7
「ライティー・ビロー・ノンフリップ」

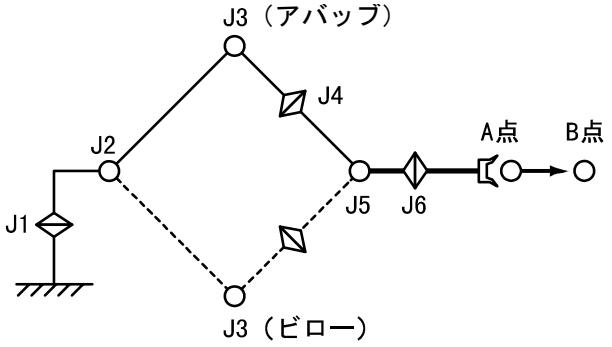


形態の種類8
「ライティー・ビロー・フリップ」



図付録2-6　腕、ひじ、手首形態の組み合わせ例

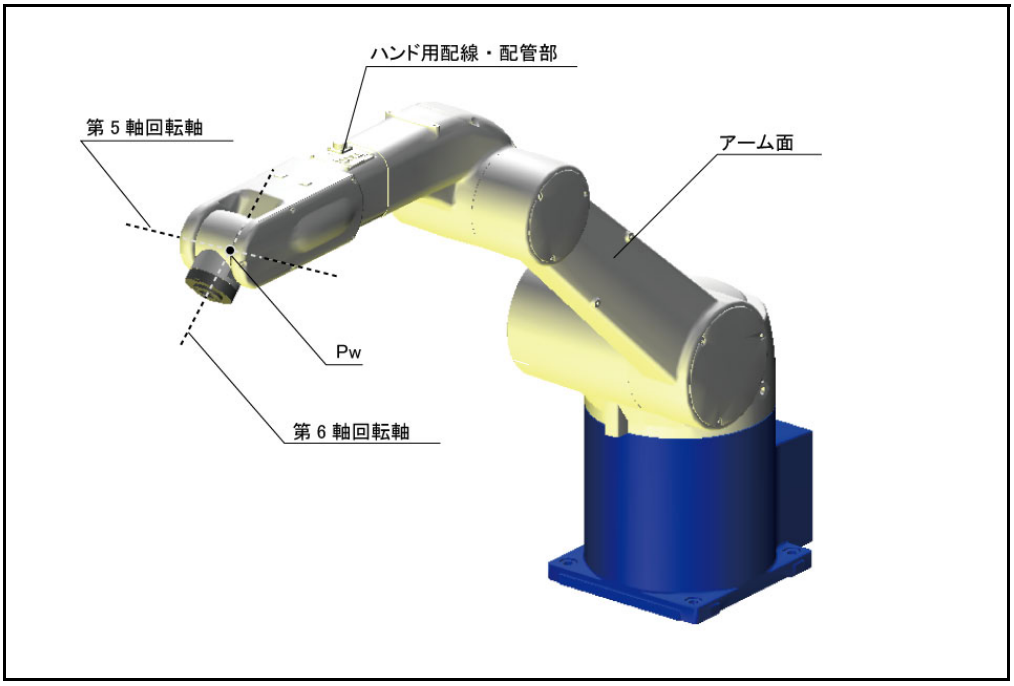
⚠注意：CP動作をするコマンドを実行する場合、出発点での位置と姿勢が同じでも、腕・ひじ・手首の形態が異なれば、各軸は異なった動作をして目標位置へ移動します。したがって、出発点での形態が異なる場合でのCP動作による移動は、各軸の設備等への干渉がないことを事前に確認してから、実施してください。
ただし、ツール先端で見た移動経路は、形態が変わっても同一です。




注意：ロボットの構造上、任意の1つの位置と姿勢に対して、すべて16種類の形態がとれるわけではありません。
位置と姿勢によっては、例えば、レフティー・アバッブ・ノンフリップの形態しかとれない場合もあります。
（ただし、通常の動作範囲内では、レフティー・アバッブ・ノンフリップ、またはレフティー・アバッブ・フリップの2種類の形態しかとれない場合がほとんどです。また、4軸形態はシングル4の場合がほとんどです。）

### (2) 形態の境界

腕・ひじ・手首・第6軸の各種の形態の境界条件について説明します。
腕・ひじ・手首の境界の判定は、第5軸の回転軸と第6軸の回転軸が交わる点「Pw」の位置を使って行ないます。（図付録2-7参照）



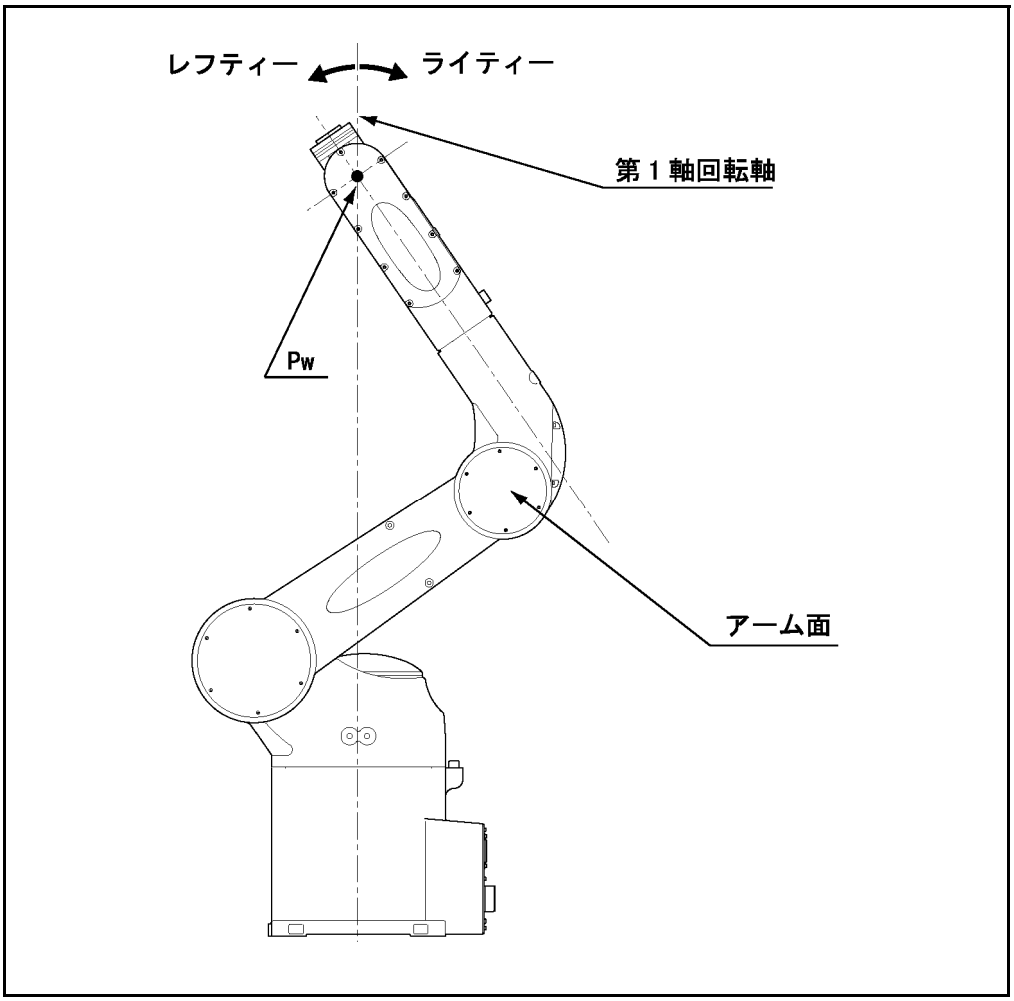

図付録2-7　「Pw」の位置

各種形態の境界位置を特異点と呼びます。
MOVE、APPROACH、DEPARTのような、CP動作をするコマンド（p.3-11「3.3　補間制御」）は、特異点の近傍を通ることはできません。
もし、軌道上に特異点近傍がある場合、ERROR6080番台（速度オーバ）または、ERROR6070番台（ソフトリミットオーバ）を発生し、停止することがあります。

①レフティー・ライティー

アーム面に対して垂直方向から見た「Pw」の位置が、第1軸の回転軸より左にある場合がレフティーで、右にある場合がライティーです。

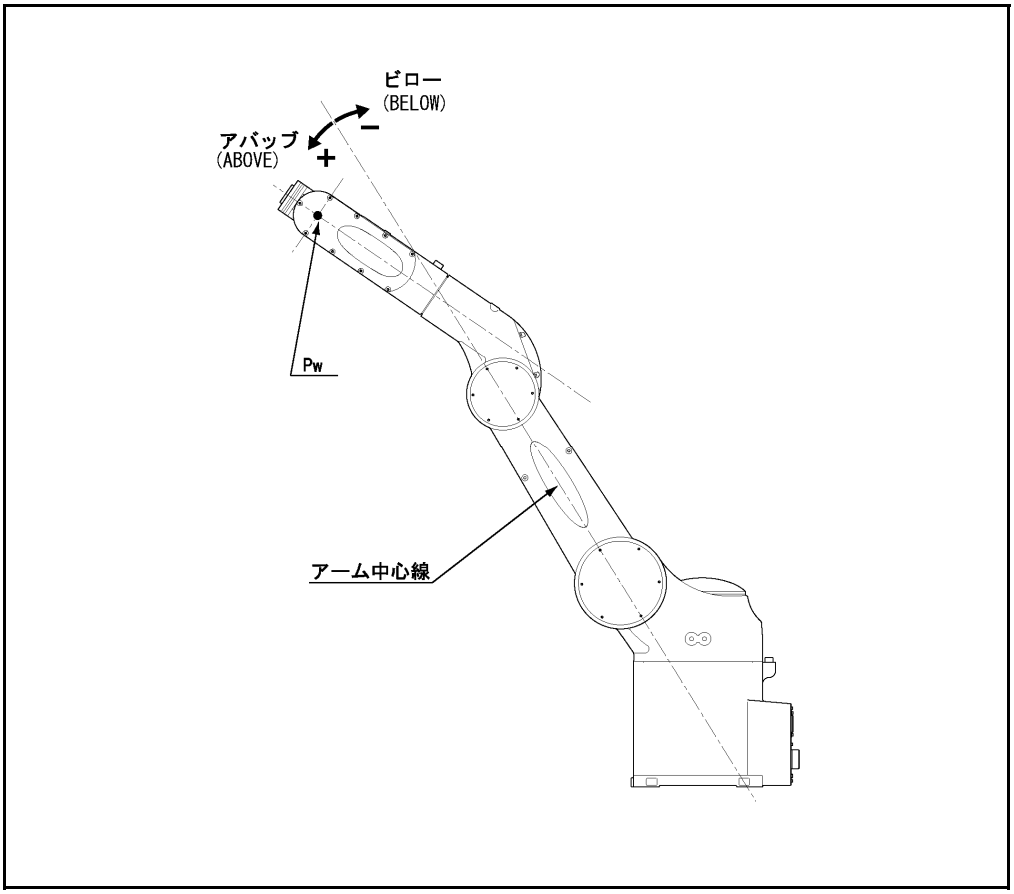
注意：「Pw」の位置が、第1軸の回転軸の上にある場合、2つの形態の境界上にあることとなり、この位置を特異点といいます。



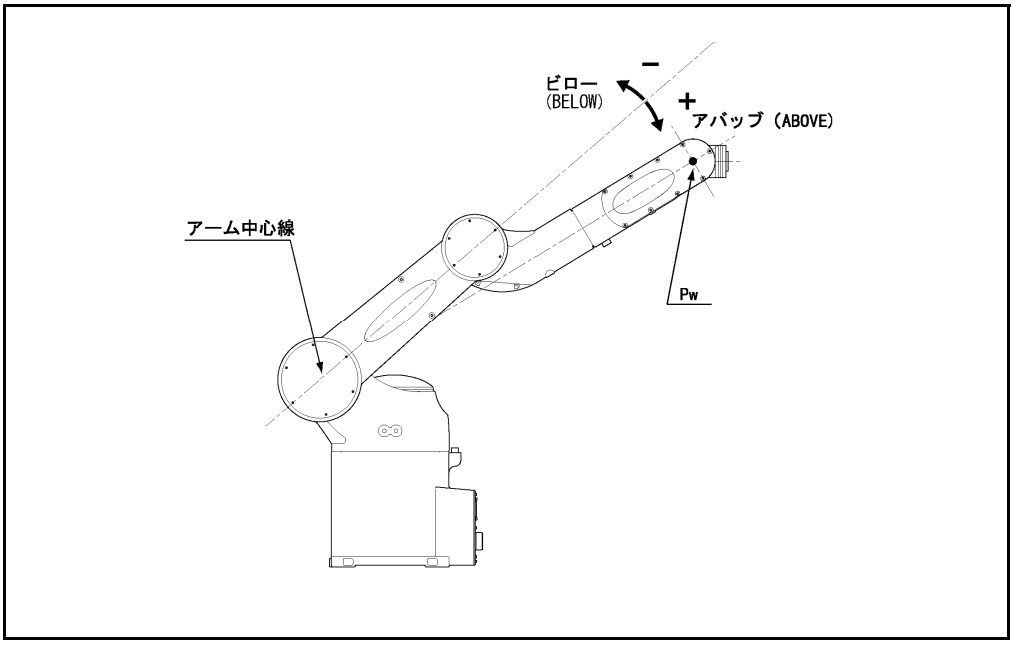

図付録2-8　レフティー・ライティーの境界

②アバッブ・ビロー

アーム中心線に対して、「Pw」の位置が＋側にある場合がアバッブで、－側にある場合がビローです。



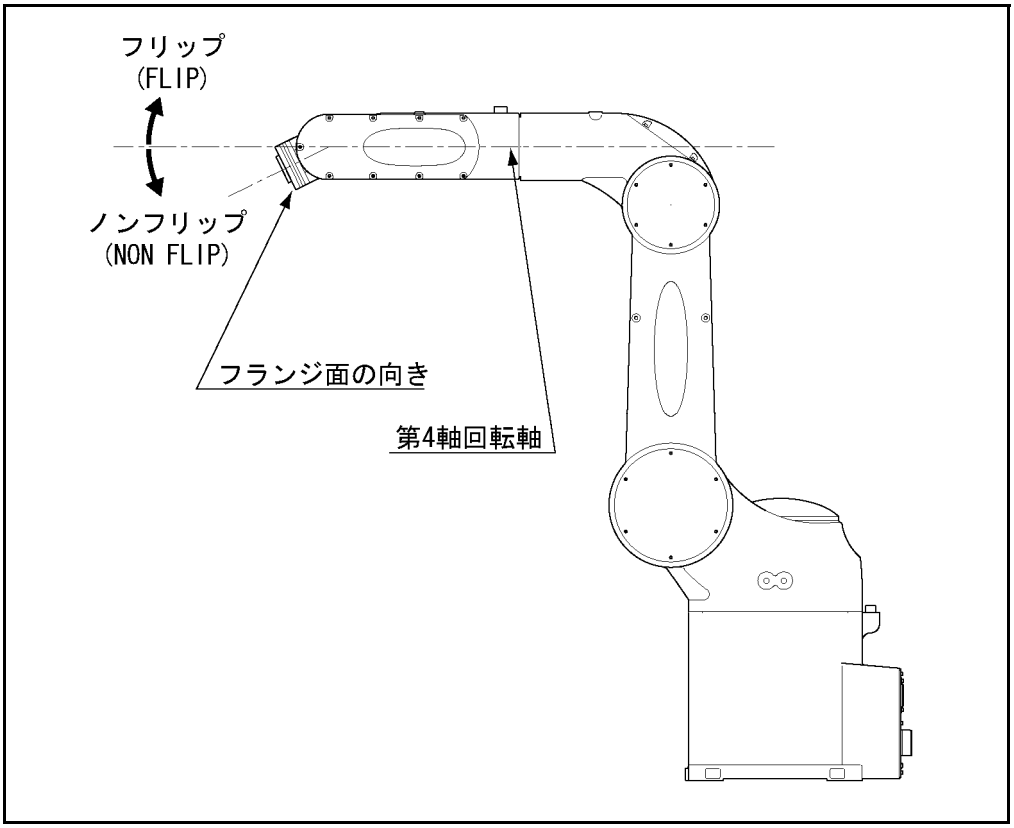

図付録2-9　アバッブ・ビローの境界（レフティーの場合）



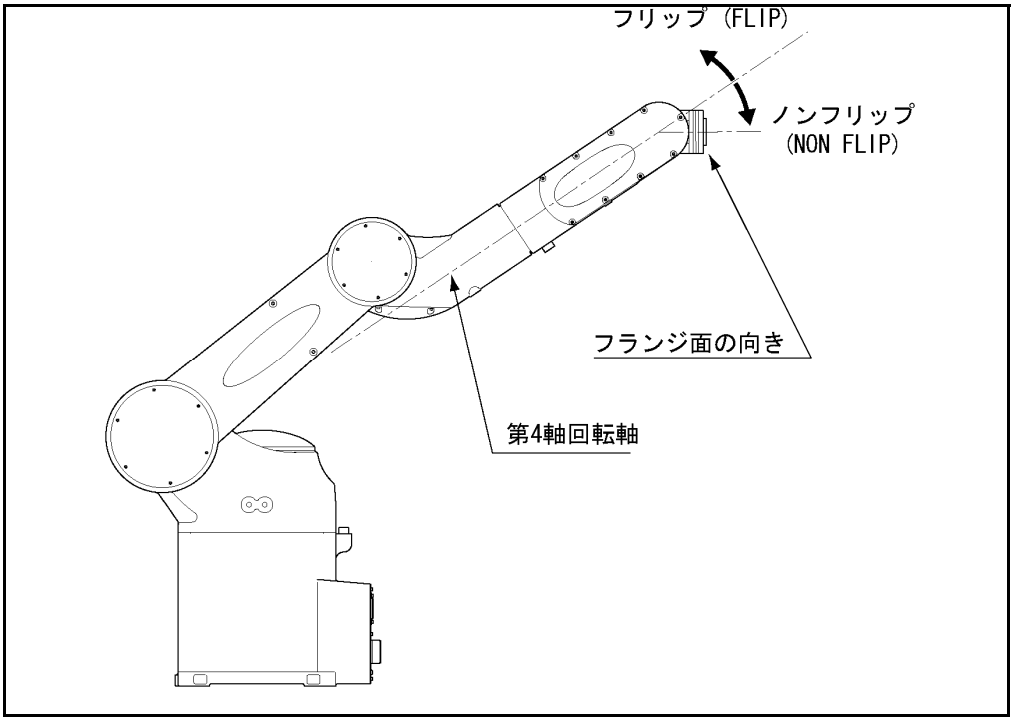

図付録2-10　アバッブ・ビローの境界（ライティーの場合）

③フリップ・ノンフリップ

第4軸の回転軸を基準にして、フランジ面の向きが上側にある場合がフリップで、下側にある場合がノンフリップです。（図付録2-11と2-12は下側にあります）



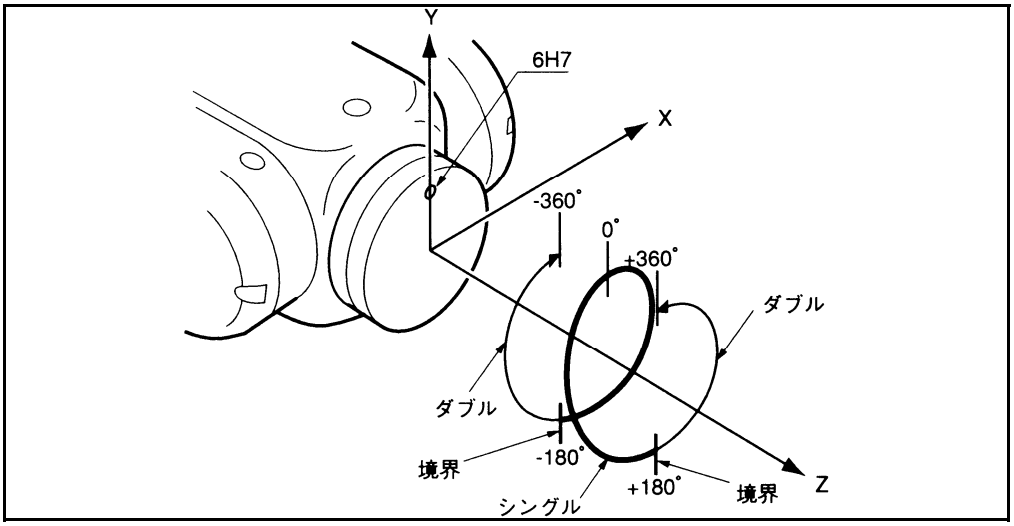

図付録2-11　フリップ・ノンフリップの境界（レフティーの場合）




図付録2-12　フリップ・ノンフリップの境界（ライティーの場合）

④シングル・ダブル

第6軸の回転角 $\theta_6$が（$-180 < \theta_6 \leqq 180$）の場合がシングルで、（$180 < \theta_6 \leqq 360$ または$-360 < \theta_6 \leqq -180$）の場合がダブルです。




図付録2-13　シングル・ダブルの境界

## 22.3 環境設定値

| テーブル番号 | マクロ名 | 内　　容 |
|---|---|---|
| 1 | cnfSYS | システムパラメータテーブル |
| 2 | cnfARM | 軌道生成パラメータテーブル |
| 3 | cnfVIS | 視覚パラメータテーブル |
| 4 | cnfPAC | PAC パラメータテーブル |
| 5 | cnfSRV | サーボパラメータテーブル |
| 6 | cnfSPD | 使用条件パラメータテーブル |
| 7 | cnfITP | インタプリタパラメータテーブル |
| 8 | cnfDIO | DIO パラメータテーブル |
| 9 | cnfCOM | 通信パラメータテーブル |

備考①：マクロ名は<pacman.h>に定義されています。GETENV、LETENV命令でマクロ名を使う場合は、次のように、そのファイルをインクルードしてください。

#INCLUDE <pacman.h>

②：任意のテーブルの要素番号のマクロ名は、WINCAPSⅢで作成することができます。

たとえば、軌道生成パラメータテーブルのマクロ定義ファイルを作成した場合は、次のようにそのファイルをインクルードしてください。

#INCLUDE "arm_cnf.h"

# 22.4　使用条件パラメータ

ティーチングペンダントの使用条件ウィンドウ（ 操作経路 ： [F2 アーム]－[F6 補助機能.]－[F7 使用条件] ）、およびWINCAPSⅢの使用条件タブ（ 操作経路 ：[プロジェクト(P)]-[パラメータ (T)...]-[使用条件]タブ）に表示されるパラメータ一覧です。

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 7 | 最適可搬質量設定モード | 0 | 0 | 0：OFF<br>1：PTP 動作のみ<br>2：CP 動作のみ<br>3：CP, PTP とも有効<br>（プログラミングマニュアル「4.6 最適可搬質量設定機能」参照） | aspChange（）にて変更可 |
| 8 | 床置き/天吊り設定 | 0 | 電源 OFF 時の値 | 0：床置き<br>1：天吊り<br>2：壁掛 | 6 軸ロボットのみ設定必要 |
| 9 | 先端負荷質量（g） | 機種により異なる | 電源 OFF 時の値 | ツールとワークを合わせた質量 | aspACLD にて変更可 |
| 10 | 先端負荷重心位置 X（mm） | 0 | 電源 OFF 時の値 | ツールとワークを合わせた負荷の重心位置の X 成分（プログラミングマニュアル「4.6 最適可搬質量設定機能」参照） | aspACLD にて変更可 |
| 11 | 先端負荷重心位置 Y（mm） | 80 | 電源 OFF 時の値 | ツールとワークを合わせた負荷の重心位置の Y 成分（プログラミングマニュアル「4.6 最適可搬質量設定機能」参照） | |
| 12 | 先端負荷重心位置 Z（mm）<br><4 軸ﾛﾎﾞｯﾄ Ver.1.9 以降><br>負荷イナーシャ（$kgcm^2$） | 100 | 電源 OFF 時の値 | ツールとワークを合わせた負荷の重心位置の Z 成分（プログラミングマニュアル「4.6 最適可搬質量設定機能」参照） | |
| 13~20 | 停止時許容パルス幅（J1～J8） | 20 | 20 | 動作命令の@E 指定時の該当軸（1 軸～8 軸）収束範囲（パルス数） | mvSetPulseWidth（）にて変更可 |
| 21 | 動作終了タイムアウト（ms） | 5600 | 5600 | 動作命令の@E 指定時、指定時間以内に指定パルス以内に収束しない場合、チェック命令タイムオーバエラーとなる。 | mvSetTimeOut（）にて変更可 |
| 22 | 制御ログ記録モード | 1 | 電源 OFF 時の値 | 制御ログの記録数。<br>設定値：1～24<br>記録数：1250 個×（設定値） | プログラム、変数を多く使用している場合は、記録個数を多くすると電源 ON 時エラーになる事があります。エラー発生時は個数を減らしてください。 |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 23 | 制御ログ記録間隔 | 8 | 電源 OFF 時の値 | 制御ログの記録間隔。<br>設定値：8、16、24、32 ms | ８の倍数以外の数値に設定された場合、８の倍数に修正されます。 |
| 24 | 重力補償有効／無効設定（6 軸ロボット用） | 0 | 電源 OFF 時の値 | 0：重力補償機能を無効。<br>1：重力補償機能を有効。 | SetGravity、ResetGravity にて変更可 |
| 25 | 電流制限リセット設定 | 0 | 電源 OFF 時の値 | a)最下位 bit0 の時：モータ電源投入時に電流制限がリセットされます。<br>b) 2bit 目が 0 の時：モータ電源投入時にサーボロック解除がリセットされます。(4 軸ロボット専用)<br>c) 3bit 目が 0 の時：モータ電源投入時に PWM スイッチング解除がリセットされます。(4 軸ロボット専用) | 初期設定値は変更しないでください。 |
| 26 | サーボロック設定<br>(4 軸ロボット用) | 0 | 電源 OFF 時の値 | 1：サーボロックを解除。 | OffSrvLock、OnSrvLock にて変更可 |
| 27 | 制御方法<br>(HM/HS-D ロボット用) | 0 | 電源 OFF 時の値 | 1：制御方法を P 制御。<br>(H*-D 設置・保守ガイド、「2.7 制御方法の変更機能」参照) | 「加速度変更モード」が 0 であることを確認してください。 |
| 28 | 高可搬ゲイン設定<br>(HM/HS-D ロボット用) | 0 | 電源 OFF 時の値 | 1： 制御ゲインを高可搬ゲインに変更。<br>(H*-D 設置・保守ガイド、「2.8 高可搬ゲインの設定」参照) | 「加速度変更モード」が 0、先端負荷質量(g)が 10000 であることを確認してください。 |
| 29 | 加速度変更モード | 0 または 1 | 電源 OFF 時の値 | 0：ゲイン変更機能を有効。<br>1：ゲイン変更機能を無効。 | 初期値は、4 軸ロボットが 0、6 軸ロボットが 1 です。**初期設定値は変更しないでください。** |
| 34 | モータ電源保持機能 | 1 | 電源 OFF 時の値 | 自動イネーブル SW 切替時のモータ電源の状態を設定する。<br>0： 自動イネーブル SW 切替時にモータ電源 ON ならばモータ電源 OFF にする。<br>1： 自動イネーブル SW を切り替えてもモータ電源の状態は変わらない。 | |
| 35 | サイクロイド動作設定 | 0 | 電源 OFF 時の値 | 0：サイクロイド動作を無効。<br>1：サイクロイド動作を有効。 | Setcycloid、Resetcycloid にて変更可 |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 53～60 | ゲイン減少割合<br>(J1～J8) | ロボット固有値 | 電源 OFF 時の値 | 該当軸（1～8 軸）のゲイン減少割合を設定。 | 「加速度変更モード」0、「制御方法」0、「高可搬ゲイン設定」0 の場合、有効です。<br>**初期設定値は変更しないでください。** |
| 61～68 | 高可搬 10kg 設定ゲイン減少割合（J1～J8）<br>(HM/HN ロボット用) | 0 | 電源 OFF 時の値 | 高可搬ゲイン設定時の該当軸（1～8 軸）のゲイン減少割合を設定。 | 「加速度変更モード」0、「制御方法」0、「高可搬ゲイン設定」1 の場合、有効です。<br>**初期設定値は変更しないでください。** |
| 69 | 新旧メカ選択<br>(4 軸ロボット用) | 0 | 電源 OFF 時の値 | 0：新式メカ（D シリーズ）の設定値。<br>1：旧式メカ（C シリーズ）の設定値。 | コントローラを単体購入し、HM/HS/HC-C シリーズに接続する場合、1 に設定してください。 |
| 70 | ﾊﾟｽ再起動時の動作設定 | 0 | 電源 OFF 時の値 | パス動作中に停止処理が行われた場合の再起動時の動作目標位置の設定<br>0：パス動作後の目標位置への動作。（デフォルト）<br>1：パス動作前の目標位置への動作。 | |
| 71 | ﾊﾟｽ動作完了範囲 | 5 | 電源 OFF 時の値 | 再起動時にパス動作前の目標位置へ動作しないための条件。<br>目標位置からの距離で設定します。 | 指令値レベルでの目標位置からの距離であり、ロボットの手先位置との距離ではありません。<br>初期設定値は変更しないでください。 |
| 78 | ﾀﾞﾝﾊﾟ設定割合<br>(X 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の X 方向のダンパ割合 | SetDampRate、ResetDampRate にて設定可。<br>ペンダントによる変更はできません。<br>（Ver. 1.4 以降） |
| 79 | ﾀﾞﾝﾊﾟ設定割合<br>(Y 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Y 方向のダンパ割合 | |
| 80 | ﾀﾞﾝﾊﾟ設定割合<br>(Z 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Z 方向のダンパ割合 | |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 81 | ﾀﾞﾝﾊﾟ設定割合<br>(X 周り)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の X 周りのダンパ割合 | SetDampRate、ResetDampRate にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 82 | ﾀﾞﾝﾊﾟ設定割合<br>(Y 周り)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Y 周りのダンパ割合 | |
| 83 | ﾀﾞﾝﾊﾟ設定割合<br>(Z 周り)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Z 周りのダンパ割合 | |
| 84 | ｺﾝﾌﾟﾗｲｱﾝｽ制御ﾓｰﾄﾞ選択 (6 軸ロボット用) | 1 | 1 | 最下位 bit が 0 の時:コンプライアンス速度制御モードとなります。<br>2bit 目が 1 の時：力制限有効設定時の重力補償補正処理をなくします。 | SetCompVMode、ResetCompVMode、SetCompControl、SetCompFControl にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 86 | 防振ﾓｰﾄﾞ選択<br>(6 軸ロボット用) | 0 | 電源 OFF 時の値 | 1：残留振動低減モード | SetVibControl、ResetVibControl にて設定可 |
| 87 | 力制限機能有効／無効設定<br>(6 軸ロボット用) | 0 | 0 | 1：力制限実行中 | SetCompControl、SetCompFControl、ResetCompControl にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 88 | 力制限座標系選択<br>(6 軸ロボット用) | 0 | 0 | 力制限座標系設定値<br>0：ベース座標<br>1：ツール座標<br>2：ワーク座標<br>となります。 | SetFrcCoord にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 89 | 力制限割合 (+X 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の+X 方向力制限割合 | SetFrcCoord にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 90 | 力制限割合 (+Y 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の+Y 方向力制限割合 | |
| 91 | 力制限割合 (+Z 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の+Z 方向力制限割合 | |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 92 | 力制限割合（+X 周り）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の+X 周り力制限割合 | SetFrcCoord にて設定可。<br>ペンダントによる変更はできません。<br>（Ver. 1.4 以降） |
| 93 | 力制限割合（+Y 周り）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の+Y 周り力制限割合 | |
| 94 | 力制限割合（+Z 周り）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の+Z 周り力制限割合 | |
| 95 | 力制限割合（-X 方向）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の-X 方向力制限割合 | SetFrcCoord にて設定可。<br>ペンダントによる変更はできません。<br>（Ver. 1.4 以降） |
| 96 | 力制限割合（-Y 方向）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の-Y 方向力制限割合 | |
| 97 | 力制限割合（-Z 方向）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の-Z 方向力制限割合 | |
| 98 | 力制限割合（-X 周り）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の-X 周り力制限割合 | SetFrcCoord にて設定可。<br>ペンダントによる変更はできません。<br>（Ver. 1.4 以降） |
| 99 | 力制限割合（-Y 周り）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の-Y 周り力制限割合 | |
| 100 | 力制限割合（-Z 周り）<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の-Z 周り力制限割合 | |
| 101 | ｺﾝﾌﾟﾗｲｱﾝｽ設定割合<br>(X 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の X 方向柔らかさ割合 | SetCompRate にて設定可。<br>ペンダントによる変更はできません。<br>（Ver. 1.4 以降） |
| 102 | ｺﾝﾌﾟﾗｲｱﾝｽ設定割合<br>(Y 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Y 方向柔らかさ割合 | |
| 103 | ｺﾝﾌﾟﾗｲｱﾝｽ設定割合<br>(Z 方向)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Z 方向柔らかさ割合 | |
| 104 | ｺﾝﾌﾟﾗｲｱﾝｽ設定割合<br>(X 周り)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の X 周り柔らかさ割合 | SetCompRate にて設定可。<br>ペンダントによる変更はできません。<br>（Ver. 1.4 以降） |
| 105 | ｺﾝﾌﾟﾗｲｱﾝｽ設定割合<br>(Y 周り)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Y 周り柔らかさ割合 | |
| 106 | ｺﾝﾌﾟﾗｲｱﾝｽ設定割合<br>(Z 周り)<br>(6 軸ロボット用) | 10000 | 10000 | 力制限時の Z 周り柔らかさ割合 | |
| 107 | ｺﾝﾌﾟﾗｲｱﾝｽ偏差許容値<br>(X 方向)<br>(6 軸ロボット用) | 100 | 100 | 力制限時の X 方向偏差許容値 | SetCompEralw にて設定可。<br>ペンダントによる変更はできません。<br>（Ver. 1.4 以降） |
| 108 | ｺﾝﾌﾟﾗｲｱﾝｽ偏差許容値<br>(Y 方向)<br>(6 軸ロボット用) | 100 | 100 | 力制限時の Y 方向偏差許容値 | |
| 109 | ｺﾝﾌﾟﾗｲｱﾝｽ偏差許容値<br>(Z 方向)<br>(6 軸ロボット用) | 100 | 100 | 力制限時の Z 方向偏差許容値 | |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 110 | ｺﾝﾌﾟﾗｲｱﾝｽ偏差許容値(X 周り)(6 軸ロボット用) | 300 | 300 | 力制限時の X 周り偏差許容値 | SetCompEralw にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 111 | ｺﾝﾌﾟﾗｲｱﾝｽ偏差許容値(Y 周り)(6 軸ロボット用) | 300 | 300 | 力制限時の Y 周り偏差許容値 | |
| 112 | ｺﾝﾌﾟﾗｲｱﾝｽ偏差許容値(Z 周り)(6 軸ロボット用) | 300 | 300 | 力制限時の Z 周り偏差許容値 | |
| 113 | 力ｵﾌｾｯﾄ設定 (X 方向)(6 軸ロボット用) | 0 | 0 | 力制限時の X 方向オフセット力 | SetFrcAssist にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 114 | 力ｵﾌｾｯﾄ設定 (Y 方向)(6 軸ロボット用) | 0 | 0 | 力制限時の Y 方向オフセット力 | |
| 115 | 力ｵﾌｾｯﾄ設定 (Z 方向)(6 軸ロボット用) | 0 | 0 | 力制限時の Z 方向オフセット力 | |
| 116 | 力ｵﾌｾｯﾄ設定 (X 周り)(6 軸ロボット用) | 0 | 0 | 力制限時の X 周りオフセットモーメント | SetFrcAssist にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |
| 117 | 力ｵﾌｾｯﾄ設定 (Y 周り)(6 軸ロボット用) | 0 | 0 | 力制限時の Y 周りオフセットモーメント | |
| 118 | 力ｵﾌｾｯﾄ設定 (Z 周り)(6 軸ロボット用) | 0 | 0 | 力制限時の Z 周りオフセットモーメント | |
| 120 | 最適可搬質量初期化設定 | 0 | 電源 OFF 時の値 | 0：可搬質量設定モードは電源 ON 時 0 に設定されます。(デフォルト)<br>1：可搬質量設定モードが電源 OFF 時に初期化されなくなります。電源 OFF 時の値が保持されます。 | (Ver. 1.4 以降) |
| 121～128 | ｺﾝﾌﾟﾗｲｱﾝｽ軸ﾄﾙｸ制限値(J1～J8)(6 軸ロボット用) | 0 | 0 | 力制限時の該当軸 (1～8 軸) の電流制限値。 | SetCompJLimit、ResetCompJLimit にて設定可。<br>ペンダントによる変更はできません。<br>(Ver. 1.4 以降) |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 196 | J4ブレーキロック選択<br>(VM-6083D,VM-60B1D,<br>VS-E 用) | 0 | 0 | ブレーキ解除中に J4 ソフトリミットオーバが発生した場合の選択<br>0：J4 ブレーキがロック<br>1：ブレーキがロックしない | (Ver.1.7 以降) |
| 197 | CP 動作時の速度設定<br>（注１） | 0 | 電源 OFF 時の値 | 0: 従来と同じ速度設定<br>1：TCP 速度（TooL 先端 CP 速度）が一定になる速度設定 | (Ver.1.8 以降) |
| | 注１：従来の設定では手先の回転動作を含んだCP動作を行った場合、TCP速度（TooL先端CP速度）は回転動作量に応じて減速されていました。そのため、指定した速度にならないとか速度が一定にならないという現象が起こっていました。<br>このパラメータを１に設定することによりTCP速度を一定に保つことが可能になりました。ただし、動作条件に応じて手先の回転速度が増減させており、回転速度が規定された限界値を超えるような動作の場合は警告を表示するとともにTCP速度が指定された速度に対して低減されます。 | | | | |
| 198 | TOOL・WORK 復帰条件<br>（注２） | 0 | 電源 OFF 時の値 | 0: 従来と同じ動作<br>（復帰しない）<br>1：電源投入時 TOOL・WORK の座標系宣言、座標系変更復帰 | (Ver.1.8 以降) |
| | 注２：従来、プログラム中において実行したTOOL・WORKの座標系宣言および変更したTOOL・WORK座標系は、コントローラの電源を切ってしまうと保持されませんでした。従って同じ操作を行なう場合、次回の電源投入後に電源を切る前と同様の操作をする必要がありました。このパラメータを１に設定することによりTOOL・WORKの座標系宣言および変更したTOOL・WORK座標系を電源投入時に前回の状態に復帰することが可能になりました。 | | | | |
| 199 | 簡単教示の円弧動作許容値 | 100 | 100 | 簡単教示時の円弧動作の位置ずれ許容値 | 通常は変更しないでください。<br>(Ver.1.8 以降) |
| 200 | アーチ動作実行フラグ | 0 | 電源 OFF 時の値 | ArchMove を実行する場合のアーチ形状の設定 | SetArchParam で設定可<br>(Ver1.9 以降) |
| 201 | アーチ動作上昇距離 | 0 | 電源 OFF 時の値 | ArchMove 時の上昇動作中に横方向動作を開始する位置 | SetArchParam で設定可<br>(Ver1.9 以降) |
| 202 | アーチ動作下降距離 | 0 | 電源 OFF 時の値 | ArchMove 時の下降動作中に横方向動作を開始する位置 | SetArchParam で設定可<br>(Ver1.9 以降) |
| 233 | 常時ベルト切断検出設定 | 0 | 電源 OFF 時の値 | 特殊ロボット専用のパラメータ | (Ver1.9 以降) |
| 234 | 電流制限時の ZT 干渉補正 | 0 | 電源 OFF 時の値 | ZT 軸にギア干渉がある場合に設定 | 通常は変更しないでください。<br>(Ver1.9 以降) |
| 235 | 予約パラメータ | 0 | 電源 OFF 時の値 | 将来の機能拡張のために予約されたパラメータ | (Ver1.9 以降) |
| 237 | 基準異常クリア許可 | 0 | 電源 OFF 時の値 | 基準位置異常を TP からクリア許可できる設定 | 通常は変更しないでください。<br>(Ver1.95 以降) |
| 238 | 動作速度制限設定 | 0 | 電源 OFF 時の値 | 特殊ロボット専用のパラメータ | 通常は変更しないでください。<br>(Ver1.95 以降) |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 239 | トラッキングモード | 0 | 電源 OFF 時の値 | 別途ﾄﾗｯｷﾝｸﾞ設定画面で設定を行ないます。（現在未対応） | この画面で変更は行なわないでください。（Ver1.95 以降） |
| 240 | 認識時のｴﾝｺｰﾀﾞ 1 基準位置 | 0 | | | |
| 241 | 動作時のｴﾝｺｰﾀﾞ 1 基準位置 | 0 | | | |
| 242 | 認識時のｴﾝｺｰﾀﾞ 2 基準位置 | 0 | | | |
| 243 | 動作時のｴﾝｺｰﾀﾞ 2 基準位置 | 0 | | | |
| 244 | ｴﾝｺｰﾀﾞ 1 現在値 | 0 | | | |
| 245 | ｴﾝｺｰﾀﾞ 2 現在値 | 0 | | | |
| 246 | ｴﾝｺｰﾀﾞ 1CALDTA | | | | |
| 247 | ｴﾝｺｰﾀﾞ 2CALDAT | 0 | | | |
| 248 | ﾜｰｸ位置検出精度 1 | 5 | | | |
| 249 | ﾄﾗｯｷﾝｸﾞ範囲上限値 1 | 20000 | | | |
| 250 | ﾄﾗｯｷﾝｸﾞ範囲下限値 1 | -20000 | | | |
| 251 | ﾄﾗｯｷﾝｸﾞ範囲上限値 2 | 20000 | | | |
| 252 | ﾄﾗｯｷﾝｸﾞ範囲下限値 2 | -20000 | | | |
| 253 | ﾄﾗｯｷﾝｸﾞ開始範囲 1(+側) | 20000 | | | |
| 254 | ﾄﾗｯｷﾝｸﾞ開始範囲 1（-側） | -20000 | | | |
| 255 | ﾄﾗｯｷﾝｸﾞ開始範囲 2 (+側) | 20000 | | | |
| 256 | ﾄﾗｯｷﾝｸﾞ開始範囲 2（-側） | -20000 | | | |
| 257 | 割り込み使用設定 | 0 | | | |
| 258 | ｴﾝｺｰﾀﾞ 1 割り込み回数 | 0 | | | |
| 259 | ｴﾝｺｰﾀﾞ 2 割り込み回数 | 0 | | | |
| 260 | ｴﾝｺｰﾀﾞ 1 割り込み設定 | 0 | | | |
| 261 | ｴﾝｺｰﾀﾞ 2 割り込み設定 | 0 | | | |
| 262 | ｴﾝｺｰﾀﾞ 1 割り込みﾃﾞｰﾀ更新 | 0 | | | |
| 263 | ｴﾝｺｰﾀﾞ 2 割り込みﾃﾞｰﾀ更新 | 0 | | | |
| 264 | ｴﾝｺｰﾀﾞ 1CALDAT(Z) | 0 | | | |
| 265 | ｴﾝｺｰﾀﾞ 2CALDAT(Z) | 0 | | | |
| 266 | ﾜｰｸ検出位置精度 2 | 5 | | | |
| 267 | ｴﾝｺｰﾀﾞ加減速度ﾁｪｯｸ | 0 | | | |
| 268 | ｴﾝｺｰﾀﾞ 1 割り込みﾃﾞｰﾀ設定 | 0 | | | |
| 269 | ｴﾝｺｰﾀﾞ 2 割り込みﾃﾞｰﾀ設定 | 0 | | | |
| 270 | 基準異常検出設定 | 0 | | | |
| 271 | CP 直線高軌跡制御 | 0 | | | |
| 272 | ｴﾝｺｰﾀﾞ速度加速度異常検出 | 1 | | | |
| 273 | ﾄﾗｯｷﾝｸﾞ対象 | 0 | | | |
| 274 | ｲﾝﾃﾞｯｸｽ 1 中心座標(X) | 100000 | | | |
| 275 | ｲﾝﾃﾞｯｸｽ 1 中心座標(Y) | 100000 | | | |
| 276 | ｲﾝﾃﾞｯｸｽ 1 中心座標(Z) | 100000 | | | |
| 277 | ｲﾝﾃﾞｯｸｽ 1 中心半径 | 100000 | | | |
| 278 | ｲﾝﾃﾞｯｸｽ 2 中心座標(X) | 100000 | | | |
| 279 | ｲﾝﾃﾞｯｸｽ 2 中心座標(Y) | 100000 | | | |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 280 | ｲﾝﾃﾞｯｸｽ 2 中心座標(Z) | 100000 | 電源 OFF 時の値 | 別途ﾄﾗｯｷﾝｸﾞ設定画面で設定を行ないます。<br>（現在未対応） | この画面で変更は行なわないでください。<br>（Ver1.95 以降） |
| 281 | ｲﾝﾃﾞｯｸｽ 2 中心半径 | 100000 | | | |
| 282 | ｲﾝﾃﾞｯｸｽ 1 半径方向ﾄﾗｯｷﾝｸﾞ上限値 | 1000 | | | |
| 283 | ｲﾝﾃﾞｯｸｽ 1 半径方向ﾄﾗｯｷﾝｸﾞ下限値 | 0 | | | |
| 284 | ｲﾝﾃﾞｯｸｽ 2 半径方向ﾄﾗｯｷﾝｸﾞ上限値 | 1000 | | | |
| 285 | ｲﾝﾃﾞｯｸｽ 2 半径方向ﾄﾗｯｷﾝｸﾞ下限値 | 0 | | | |
| 286 | ｲﾝﾃﾞｯｸｽ 1 基準座標(X) | 0 | | | |
| 287 | ｲﾝﾃﾞｯｸｽ 1 基準座標(Y) | 0 | | | |
| 288 | ｲﾝﾃﾞｯｸｽ 1 基準座標(Z) | 0 | | | |
| 289 | ｲﾝﾃﾞｯｸｽ 2 基準座標(X) | 0 | | | |
| 290 | ｲﾝﾃﾞｯｸｽ 2 基準座標(Y) | 0 | | | |
| 291 | ｲﾝﾃﾞｯｸｽ 2 基準座標(Z) | 0 | | | |
| 292 | ｲﾝﾃﾞｯｸｽ 1 姿勢追従 | 0 | | | |
| 293 | ｲﾝﾃﾞｯｸｽ 2 姿勢追従 | 0 | | | |
| 294 | ﾜｰｸ重複確認範囲 1 | 200 | | | |
| 295 | ﾜｰｸ重複確認範囲 2 | 200 | | | |
| 296 | モータコマンド設定 | 0 | 電源 OFF 時の値 | モータコマンドの異常検出設定<br>0:ロボット動作中に MOTOR OFF コマンド実行で異常検出<br>1:ロボット動作中に MOTOR OFF コマンド実行で異常未検出 | （Ver1.98 以降） |
| 297 | ｻｰﾎﾞﾃﾞｰﾀ取得番号 | 0 | 電源 OFF 時の値 | 特殊ロボット専用のパラメータ | 通常は変更しないでください。<br>（Ver1.98 以降） |
| 307 | 特異点回避機能有効設定 | 0 | 電源 OFF 時の値 | 特異点回避機能の有効/無効の設定<br>0：無効<br>1：有効<br>(プログラミングマニュアル Ⅰ「特異点回避機能」参照) | （Ver.2.61 以降）<br>SetSingularAvoid ()にて変更可能 |

| 番号 | 表示 | 出荷時値 | 電源投入時 | 内容 | 備考 |
|---|---|---|---|---|---|
| 394 | たわみ補正機能有効設定 | 0 | 電源 OFF 時の値 | 重力によるロボットアームのたわみを動作位置、姿勢、負荷状況に応じ自動的に補正します。<br>0：無効<br>1：有効 | (Ver. 3.0 以降) |
| | 注：ポイント教示後にたわみ補正を有効にした場合、教示ポイントが変わります。ポイント教示前に有効にしてください。また、先端負荷重量や先端負荷重心位置によっても教示ポイントが変わります。ポイント教示は先端負荷重量や先端負荷重心位置を正しく合わせて行ってください。 | | | | |
| 398 | 特異点回避拡張設定 | ロボットタイプによって異なる。<br>VS-050,060,068,087<br>：1<br>それ以外<br>：0 | 電源 OFF 時の値 | パス動作中に特異点回避を行います。ただし、この機能を有効にするには、特異点回避機能有効設定を有効（1）にしておく必要があります。<br>0：無効<br>1：有効 | (Ver.3.2 以降) |
| 404 | ティーチチェック時の軌道補正 | 1 | 電源 OFF 時の値 | ティーチチェック時のロボットの軌道を自動運転時の軌跡にあわせます。<br>0：パス動作での軌道が自動運転時とティーチチェック時で異なる場合があります。<br>1：ティーチチェック時のパス動作は自動運転時と同じ軌跡で動作します。 | (Ver.2.8 以降) |

# 22.5　予約語一覧

これらの予約語は、変数名やラベル名として使用することはできません。

| | |
|---|---|
| A | ABOVE ABS ACCEL ACOS ADDCOMLOG ADDERRLOG ALL AND APPROACH AREA AREAPOS AREASIZE ARRIVE AS ASC ASIN ATN ATN2 AVEC |
| B | B BELOW BIN BIT BLOB BLOBCOPY BLOBLABEL BLOBMEASURE BREAK BUZZER BYTE |
| C | C CALCWORKPOS CALL CAMASPECT CAMIN CAMLEVEL CAMMODE CAMTIN CAMTOFF CAMTON CASE CHANGE_BCAP CHANGE_PCAP CHANGECOORD CHANGETOOL CHANGEWORK CHGEXTMODE CHGINTMODE CHR CLEARLOG CLOSECOMLOG CLOSEERRLOG CLRERR CODE COM_DISCOM COM_ENCOM COM_STATE CONTINUERUN CONT COS CREATESEM CURACC CURDEC CUREJNT CUREXJ CUREXTACC CUREXTDEC CUREXTSPD CURFIG CURJACC CURJDEC CURJNT CURJSPD CUROPTMODE CURPOS CURSPD CURTOOL CURTRACKPOS CURTRACKPOSEX CURTRACKSPD CURTRN CURWORK CYCLE |
| D | D DATE DECEL DEF DEFDBL DEFEND DEFINT DEFIO DEFJNT DEFPOS DEFSNG DEFSTR DEFTRN DEFVEC DEG DEGRAD DELAY DELETESEM DEPART DESTEXJ DESTJNT DESTPOS DESTTRN DETECT DIM DISP_PAGE DIST DO DOUBLE DRAW DRIVE DRIVEA |
| E | E EJ ELSE ELSEIF EMGSTOP END ENDIF EQJ EQP EQS EQT EQU EQV ERL ERR ERRMSG ERROR EX EXA EXECAL EXIT EXP EXTSPEED |
| F | F FALSE FIG FIGAPRL FIGAPRP FLIP FLOAT FLUSH FLUSHSEM FOR FORMAT FREE |
| G | GETCOMLOG GETENV GETERR GETERRLOG GETERRLVL GETJNTDATA GETSRVDATA GETSRVSTATE GIVEARM GIVEARMNOWAIT GIVESEM GIVEVIS GO GOHOME GOSUB GOTO GRASP |
| H | H HALT HAND HEX HMOVE HOLD HOME |
| I | I IF IN INIT INPUT INPUTB INPUTB$ INT INTEGER INTERRUPT IO IOBLOCK IS |
| J | J J2P J2T JOINT JSPEED JACCEL JDECEL |
| K | KEEP KILL KILLALL |
| L | L LEFT LEFTY LEN LET LETA LETENV LETF LETJ LETJ1 LETJ2 LETJ3 LETJ4 LETJ5 LETJ6 LETO LETP LETR LETRX LETRY LETRZ LETT LETX LETY LETZ LINEINPUT LINPUTB LOG LOG10 LOOP LPRINTB |
| M | MAGNITUDE MAX MID MIN MIRROR MOD MOTOR MOVE MPS |
| N | NEJ NEP NEQ NES NET NEV NEXT NONFLIP NORMTRN NOT |
| O | OFF ON OR ORD OUT OVEC |
| P | P P2J P2T PI POSCLR POSE POSITION POSRX POSRY POSRZ POST POSX POSY POSZ POW PRINT PRINTB PRINTDBG PRINTLBL PRINTMSG PRIORITY PROGRAM PTP PVEC |
| R | R RAD RADDEG RELEASE REM REPEAT RESET RESETAREA RESETVALVE RESULT RESUME RETURN RIGHT RIGHTY RND ROB ROBOT ROTATE ROTATEH RUN RVEC |

| | |
|---|---|
| S | S SEC SELECT SENDKEY SET SETAREA SETVALVE SGN SHADDGROUP SHCIRCLE SHCLRGROUP SHCLRMODEL SHCOPYMODEL SHCORNER SHDEFCIRCLE SHDEFCORNER SHDEFMODEL SHDISPMODEL SHDROPGROUP SHGROUP SHLOADMODEL SHMODEL SHREFGROUP SHREFMODEL SHSAVEMODEL SIN SINGLE SPEED SPRINTF SQR SQRT ST_ASPACLD ST_ASPCHANGE ST_OFFSRVLOCK ST_ONSRVLOCK ST_RESETCOMPCONTROL ST_RESETCOMPERALW ST_RESETCOMPJLIMIT ST_RESETCOMPRATE ST_RESETCOMPVMODE ST_RESETCURLMT ST_RESETDAMPRATE ST_RESETERALW ST_RESETFRCASSIST ST_RESETFRCLIMIT ST_RESETGRAVITY ST_RESETGRVOFFSET ST_RESETZBALANCE ST_SETCOMPCONTROL ST_SETCOMPERALW ST_SETCOMPFCONTROL ST_SETCOMPJLIMIT ST_SETCOMPRATE ST_SETCOMPVMODE ST_SETCURLMT ST_SETDAMPRATE ST_SETERALW ST_SETFRCASSIST ST_SETFRCCOORD ST_SETFRCLIMIT ST_SETGRAVITY ST_SETGRVOFFSET ST_SETZBALANCE STARTDATA STARTLOG STATUS STEP STOP STOPEND STOPLOG STARTTRI STR STRING STRPOS SUB SUSPEND SUSPENDALL SYSSTATE |
| T | T T2J T2P TACCEL TAKEARM TAKESEM TAKEVIS TAN TDECEL THEN TIME TIMER TINV TLSPEED TO TOOL TOOLPOS TRACKDATAGET TRACKDATAINFO TRACKDATAINITIALIZE TRACKDATANUM TRACKDATASET TRADIUS TRANS TRAVELD TRAVELG TRNS TRUE TS TSMOVE TSPEED TSPIN TSRSTATE TTURN |
| U | UNTIL |
| V | V VAL VECTOR VER VISBINA VISBINAR VISBRIGHT VISCAL VISCAMOUT VISCIRCLE VISCLS VISCOPY VISCROSS VISDEFCHAR VISDEFTABLE VISEDGE VISELLIPSE VISFILTER VISGETNUM VISGETP VISGETSTR VISHIST VISLEVEL VISLINE VISLOC VISMASK VISMEASURE VISOVERLAY VISPLNOUT VISPOSX VISPOSY VISPRINT VISPROJ VISPTP VISPUTP VISREADBAR VISREADQR VISRECT VISREFCAL VISREFHIST VISREFTABLE VISSCREEN VISSECT VISSTATUS VISWORKPLN |
| W | WAIT WAITTRACKMOVE WAITTRACKMOVEEX WDIN WDOUT WEND WHILE WINDCLR WINDCOPY WINDDISP WINDMAKE WINDREF WORD WORK WORKPOS WRITE |
| X | X XOR XY XYH |
| Y | YZ YZH |
| Z | ZX ZXH |

# 22.6　旧言語コマンド対応表(VS)

## ■ 動作コマンド対応表

| 旧言語コマンド | PAC コマンド | MOVE マクロ |
|---|---|---|
| MVE J*n* | MOVE P, J*n* | ―― |
| MVP J*n* | MOVE P, @P J*n* | ―― |
| MVSE P*n* | MOVE L, T*n* | ―― |
| MVSP P*n* | MOVE L, @P T*n* | ―― |
| DRVE (*j1j2j3j4j5j6*) | DRIVE (1*j1*),... | *Jn* = *Jc* + (*j1j2j3j4j5j6*)<br>MOVE P, *Jn* |
| DRVE J*n* | DRIVE (1,JOINT(1,J*n*)),... | *jj* = *Jc* + (JOINT(1,J*n*)),...,JOINT(6,J*n*)<br>MOVE P, @P J*n* |
| DRVP (*j1j2j3j4j5j6*) | DRIVE @P (1*j1*),... | *Jn* = *Jc* + (*j1j2j3j4j5j6*)<br>MOVE P, @P J*n* |
| DRVP J*n* | DRIVE @P (1,JOINT(1*j1*)),... | *jj* = *Jc* + (JOINT(1,J*n*)),...,JOINT(6,J*n*)<br>MOVE P, @P *jj* |
| DRWE (*x,y,z*) | DRAW L, (*x,y,z*) | MOVE L, *Pc* + (*x,y,z*) |
| DRWE Pn | DRAW L, PVEC(T*n*) | MOVE L, *Pc* + (POSX(P*n*),POSY(P*n*),<br>POSZ(Pn)) |
| DRWP (*x,y,z*) | DRAW L, @P (*x,y,z*) | MOVE L, @P *Pc* + (*x,y,z*) |
| DRWP P*n* | DRAW L, @P PVEC(T*n*) | MOVE L, @P *Pc* + (POSX(P*n*),<br>POSY(P*n*),POSZ(P*n*)) |
| DEPE *n* | DEPART L,*n* | MOVE L, *Pc* + (0,0,−*n*)H |
| DEPP *n* | DEPART L, @P *n* | MOVE L, @P *Pc* + (0,0,−*n*)H |
| APRE *n* | APPROACH L, *Px*, *n* | MOVE L,*Px* + (0,0,−*n*)H |
| APRP *n* | APPROACH L, @P *Px*, *n* | MOVE L,@P *Px* + (0,0,−*n*)H |
| APRJE *n* | APPROACH P, *Px*, *n* | MOVE P,*Px* + (0,0,−*n*)H |
| APRJP *n* | APPROACH P, @P *Px*, *n* | MOVE P,@P *Px* + (0,0,−*n*)H |
| ROTE *n* | ROTATEH *n* | MOVE L, *Pc* + (*0,0,0,0,0,n*)H |
| ROTP *n* | ROTATEH @P *n* | MOVE L, @P *Pc* + (*0,0,0,0,0,n*)H |
| MVRE P*n2*,P*n3* | MOVE C, T*n2*, T*n3* | ―― |
| MVPR P*n2*,P*n3* | MOVE C, T*n2*, @P T*n3* | ―― |

*(!)* *Pc*/*Jc* は現在位置、*Px*/*Jx* は次の目標位置。
*(!)* DRVE は、非ゼロの軸だけ DRIVE 文の引数に並べればよい。
*(!)* *jj* は一時的なローカル変数。

## ■ 速度指定コマンド対応表

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| ISP *n* | SPEED *n* | |
| ACC *n* | ACCEL *n,n* | |
| AACC *n* | ACCEL *n* | |
| RACC *n* | DECEL *n* | |

## ■ ジャンプコマンド対応表

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| JI *m*-*n* | IF IO[m]=1 THEN *label*n | |
| JZ *m*-*n* | IF IO[m]=0 THEN *label*n | |
| JMP *n* | GOTO *label*n | |
| CMP *l s m* | IF *l s m* THEN *label*n | *s* は比較記号 |

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| LABL *n* | *label*n.* | |
| LABL *n* | *label*n.* | |
| IPCLR *n* | CALL pltResetAll(*n*) | |
| INTRPT | INTERRUPT ON | 次のコマンドの後で INTERRUPT OFF する。 |
| REM | REM\|' | |
| ON *n*<br>ON *n-m* | SET IO[*n*]<br>SET IO[*n* TO *m*] | |
| OFF *n*<br>OFF *n-m* | RESET IO[*n*]<br>RESET IO[*n* TO *m*] | |
| ONT *n-m* TIME=*t* | SET IO[*n* TO *m*], t*10 | 単位は msec. (注)PAC の場合は次の処理に進まない。 |
| VON *n*<br>VON *n-m* | SET IO[*n*]<br>SET IO[*n* TO *m*] | |
| VOFF *n*<br>VOFF *n-m* | RESET IO[*n*]<br>RESET IO[*n* TO *m*] | |
| ON PLT1END | SET IO[*n*] | DIO 割付表参照 |
| OFF PLT1END | RESET IO[*n*] | DIO 割付表参照 |
| ON PLTEND | SET IO[*n*] | DIO 割付表参照 |
| OFF PLTEND | RESET IO[*n*] | DIO 割付表参照 |
| INB *L m-n* | CALL ndInb(*L,m,n*) | ライブラリコール |
| ONB *L m-n* | CALL ndOnb(*L,m,n*) | ライブラリコール |
| ONB l*x m-n* | CALL ndOnbl(*x,m,n*) | ライブラリコール |

## ■ 停止コマンド対応表

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| END | END | |
| STOP | HOLD | |
| STOPEND | STOPEND | |
| TIM *n* | DELAY *n** 10 | |

## ■ SETI コマンド対応表

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| 変数 6 軸 P*n* | T[*n*] | |
| 間接参照 I*n*.P | P[*n*] | |
| $ | CURPOS\|* | |
| ¥ | CURJNT\|* | |
| ISP | CURSPD | |
| AACC | CURACC | |
| RACC | CURDEC | |
| SETI l*x*=N_*n* | CALL pltGetN(*n,x*) | ライブラリコール |
| SETI l*x*=M_*n* | CALL pltGetM(*n,x*) | ライブラリコール |
| SETI l*x*=K_*n* | CALL pltGetK(*n,x*) | ライブラリコール |
| SETI l*x*=N1_*n* | CALL pltGetN1(*n,x*) | ライブラリコール |
| SETI l*x*=M1_*n* | CALL pltGetM1(*n,x*) | ライブラリコール |
| SETI l*x*=K1_*n* | CALL pltGetK1(*n,x*) | ライブラリコール |

## ■ 演算コマンド対応表

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| + | + | 加 |
| − | − | 減 |
| * | * | 乗 |
| / | / | 除 |
| % | MOD | 剰余 |
| . | * | 内積(ベクトル用) |
| x | x | 外積(ベクトル用) |
| ABS(*n*) | ABS(*n*) | |
| SIN(*n*) | SIN(*n*) | |
| COS(*n*) | COS(*n*) | |
| TAN(*n*) | TAN(*n*) | |
| ATAN(*n*) | ATAN(*n*) | |
| ATN2(*y,x*) | ATN2(*y,x*) | |
| SQRT(*n*) | SQRT(*n*) | |
| FWRD(J*n*) | J2P(J[*n*]) | |
| REV2(P*n*) | P2J(P[*n*]) | |
| TRNS(P*m*,J*k*) | P*m* + (*x,y,z,* $\alpha$, $\beta$, $\gamma$)H | 次ページの「座標系移動関数について」を参照 |
| TINV(P*n*) | TINV(T*n*) | |
| DATE | CALL ndDate | ライブラリ仕様検討中 |
| TIME(0) | CALL ndTime | ライブラリ仕様検討中 |
| TIME(1) | TIMER | |

## ■ 通信コマンド対応表

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| VIS *n* | CALL ndVis(*n*) | ライブラリコール |
| JF *n-m* | CALL ndJf(*n,k*)<br>IF I[*k*] = 0 THEN *label | ライブラリコール |
| VSET *n* | CALL ndVset() | ライブラリコール |
| VDT | CALL ndVdt | ライブラリコール |
| VPUT $<br>VPUT P*n* | CALL ndVput() | ライブラリコール |
| VRST | CALL ndVrst | ライブラリコール |

## ■ 定義済み命令対応表

| 旧言語コマンド | PAC コマンド | 備　　考 |
|---|---|---|
| SUB *n* | CALL SUB*n* | SUB*n* はプログラム名 |
| PALT *n* | CALL pltMove(*n*) | ライブラリコール |
| TOOL *n* | CHANGETOOL *n* | |

## 座標系移動関数について

PAC コマンドでは、旧言語コマンドの座標系移動関数（TRNS コマンド)はありませんが、ポジション演算（「7.9.7　ポジション演算」参照）により、座標系を移動できます。
座標系は、P 型変数を用いる方法と T 型変数を用いる方法があり、それぞれワーク座標基準の移動とツール座標基準の移動が可能です。
J 型変数に対し座標系移動する場合は、J2P、J2T で P 型または T 型変数に変換してください。

(1) P 型変数を用いる方法

① ワーク座標系を基準にして相対移動量を指定する場合

Pn = Pm + (X,Y,Z,Rx,Ry,Rz)

基準位置 Pm から、位置は、ワーク座標系の X 軸、Y 軸、Z 軸方向へそれぞれ、X(mm)、Y(mm)、Z(mm)移動し、姿勢は、ワーク座標系の X 軸周り、Y 軸周り、Z 軸周りにそれぞれ Rx（度）、Ry（度）、Rz（度）回転した位置 Pn を求めます。ロボット形態は、Pm の形態となります。

② ツール座標系を基準にして相対移動量の指定する場合

Pn = Pm + (X,Y,Z,Rx,Ry,Rz) H

基準位置 Pm から、位置は、ツール座標系の X 軸（ノーマルベクトル）、Y 軸（オリエントベクトル）、Z 軸（アプローチベクトル）方向へそれぞれ、X(mm)、Y(mm)、Z(mm)移動し、姿勢は、ツール座標系の X 軸（ノーマルベクトル）周り、Y 軸（オリエントベクトル）周り、Z 軸（アプローチベクトル）周りにそれぞれ Rx（度）、Ry（度）、Rz（度）回転した位置 Pn を求めます。ロボット形態は、Pm の形態となります。

(2) T 型変数を用いる方法

① ワーク座標系を基準にして相対移動量を指定する場合

```
DEFTRN LT1,LT2
DEFPOS LP1
LP1=(0,0,0,Rx,Ry,Rz)
LT1=P2T(LP1)
LT2=Tm
LETP LT2=(0,0,0)
Tn=LT1*LT2
LETP Tn=PVEC(Tm)+(X,Y,Z)
LETF Tn=FIG(Tm)
```

基準位置 Tm から、位置は、ワーク座標系の X 軸、Y 軸、Z 軸方向へそれぞれ、X(mm)、Y(mm)、Z(mm)移動し、姿勢は、ワーク座標系の X 軸周り、Y 軸周り、Z 軸周りにそれぞれ Rx（度）、Ry（度）、Rz（度）回転した位置 Tn を求めます。ロボット形態は、Tm の形態となります。

② ツール座標系を基準にして相対移動量の指定する場合

```
DEFTRN LT1
DEFPOS LP1
LP1=(X,Y,Z,Rx,Ry,Rz)
LT1=P2T(LP1)
Tn=Tm*LT1
LETF Tn=FIG(Tm)
```

基準位置 Tm から、位置は、ツール座標系の X 軸（ノーマルベクトル）、Y 軸（オリエントベクトル）、Z 軸（アプローチベクトル）方向へそれぞれ、X(mm)、Y(mm)、Z(mm)移動し、姿勢は、ツール座標系の X 軸（ノーマルベクトル）周り、Y 軸（オリエントベクトル）周り、Z 軸（アプローチベクトル）周りにそれぞれ Rx（度）、Ry（度）、Rz（度）回転した位置 Tn を求めます。ロボット形態は、Tm の形態となります。

（注意）TRNS コマンドは、ツール座標系を基準にした座標系移動命令ですが、姿勢の回転方法が、P 型変数、T 型変数によるツール座標系移動命令と以下の点で異なります。

| | 座標系の姿勢回転方法 |
|---|---|
| Js=(X,Y,Z,Rx,Ry,Rz)<br>Pn=TRNS (Pm,Js) | 基準位置 Pm の姿勢から、ツール座標系の X 軸（ノーマルベクトル）周りに Rx（度）回転し、回転したツール座標系の Y 軸周りに Ry（度）回転し、さらに回転したツール座標系の Z 軸周りに Rz（度）回転する。 |
| Pn=Pm+(X,Y,Z,Rx,Ry,Rz)H | 基準位置 Pm の姿勢から、ツール座標系の X 軸（ノーマルベクトル）周りに Rx（度）回転し、回転前のツール座標系の Y 軸周りに Ry（度）回転し、回転前のツール座標系の Z 軸周りに Rz（度）回転する。 |

旧言語の TRNS コマンドと同等な姿勢回転は、以下のように指定してください。

(1) P 型変数を用いる場合

```
Pn=Pm+(X,Y,Z,0,0,0)H
Pn=Pn+(0,0,0,Rx,0,0)H
Pn=Pn+(0,0,0,0,Ry,0)H
Pn=Pn+(0,0,0,0,0,Rz)H
```

(2) T 型変数を用いる場合

```
DEFTRN LT
DEFPOS LP
LP=(X,Y,Z,0,0,0)
LT=P2T(LP)
Tn=Tm*LT
LP=(0,0,0,Rx,0,0)
LT=P2T(LP)
Tn=Tn*LT
LP=(0,0,0,0,Ry,0)
LT=P2T(LP)
Tn=Tn*LT
LP=(0,0,0,0,0,Rz)
LT=P2T(LP)
Tn=Tn*LT
LETF Tn=FIG(Tm)
```

# 22.7 バージョン対応表

VER$関数で使われる数式の値とモジュールの対応を下表に示します。
VER$関数については、p.19-3「19.1 システム、VER$（関数）」を参照してください。

**バージョン対応表**

| 数式の値 | モジュール名称 |
|---|---|
| 0 | ROM |
| 1 | メインボード |
| 2 | DIO ボード |
| 3 | 視覚ボード |
| 4 | デバイスネットボード |
| 5 | 教示ペンダント |
| 6 | 操作パネル |
| 7 | PAC 言語処理モジュール |
| 8 | 中間コード処理モジュール |
| 9 | 軌道処理モジュール |
| 10 | サーボ処理モジュール |
| 11 | 通信処理モジュール |
| 12 | ペンダント処理モジュール |
| 13 | 視覚処理モジュール |
| 14 | 視覚通信処理モジュール |
| 15 | デバイスネット処理モジュール |
| 16 | PAC 言語仕様 |
| 17 | 中間コード仕様 |
| 18 | サーボ通信仕様 |
| 19 | 視覚通信仕様 |
| 20 | ROBOTalk 通信仕様 |
| 21 | ペンダント通信仕様 |
| 22 | デバイスネット通信仕様 |

# 22.8　設定パラメータ表

## ■ Pac Manager - プログラム

| 項目名 | マクロ名 | 説明 |
|---|---|---|
| I 型変数の数 | PC_NO_INT | 番号付き長整数型変数の数 |
| F 型変数の数 | PC_NO_SNG | 番号付き単精度実数型変数の数 |
| D 型変数の数 | PC_NO_DBL | 番号付き倍精度実数型変数の数 |
| V 型変数の数 | PC_NO_VEC | 番号付きベクトル型変数の数 |
| P 型変数の数 | PC_NO_POS | 番号付きポジション型変数の数 |
| J 型変数の数 | PC_NO_JNT | 番号付きジョイント型変数の数 |
| T 型変数の数 | PC_NO_TRN | 番号付き同次変換型変数の数 |
| S 型変数の数 | PC_NO_STR | 番号付き文字列型変数の数 |
| 近似比較精度 (*10 ^ 6) | PC_EPSILON | 近似比較演算子(=.)の許容偏差量(×1000000) |
| トレースコードの削除 | PC_NO_TRACE | コンパイルコードからデバッグコードを削除する<br>(0:削除しない,　2:すべて削除) |
| サイクルタイム算出コードの削除 | PC_NO_CYC_TIME | コンパイルコードからサイクルタイム算出コードを削除する<br>(0:削除しない, 1:一部削除, 2:すべて削除) |
| 配列範囲チェックコードの削除 | PC_NO_CHK_INDEX | コンパイルコードから配列範囲チェックコードを削除する<br>(0:削除しない, 1:削除する) |
| エラー割り込み処理コードの削除 | PC_NO_ERR_TRAP | コンパイルコードからエラー割り込み処理コードを削除する<br>(0:削除しない, 1:削除する) |

## ■ Dio Manager - ハードウェア

| 項目名 | マクロ名 | 説明 |
|---|---|---|
| I/O割付<br>■I/O ハードウエア選択<br><br>■I/O の割付モードの設定 | IO_ASSIGN | ■I/O ハードウエア選択<br>(パラレル、DeviceNet ｽﾚｰﾌﾞ、DeviceNet マスタ、PROFIBUS-DP スレーブ)<br>■I/O の割付モードの設定<br>(互換モード, 標準モード) |
| パリティビット(0:無効, 1:有効) | IO_PARITY | プログラム起動時のプログラム No.選択パリティの設定<br>パリティビット(0:無効, 1:有効) |
| HandIO.割り込み設定<br>(0:無効, 1:有効) | IO_HD_ENABLED | Hand I/O の割り込み設定(0:無効, 1:有効) |
| DeviceNet.入力スロット数 | IO_DN_IN_SLOT | DeviceNet の入力スロット数 |
| DeviceNet.出力スロット数 | IO_DN_OUT_SLOT | DeviceNet の出力スロット数 |
| DeviceNet.準備異常検出<br>(0:無効, 1:有効) | IO_DN_EXERRCHK | DeviceNet の準備異常検出(0:無効, 1:有効) |
| 専用 I/O 出力モード<br>(0:無効,1:有効) | IO_DEDICTDIO | 専用 I/O 出力モード (0:無効,1:有効) |
| M-Dnet 異常表示(0:毎回, 1:初回) | IO_MDN_ERRDISP | M-Dnet 異常表示(0:毎回, 1:初回) |
| PROFIBUS ﾉｰﾄﾞｱﾄﾞﾚｽ(1～125) | IO_SPRFI_ADRS | PROFIBUS ﾉｰﾄﾞｱﾄﾞﾚｽ(1～125) |
| PROFIBUS 入力設定<br>(0:8/1:12/2,16/3:20/4:32) | IO_SPRFI_INTYPE | PROFIBUS 入力設定<br>0 : 8byte Output con<br>1 : 12byte Output con<br>2 : 16byte Output con<br>3 : 20byte Output con<br>4 : 32byte Output con |
| PROFIBUS 出力設定<br>(0:8/1:12/2,16/3:20/4:32) | IO_SPRFI_OUTTYPE | PROFIBUS 出力設定<br>0 : 8byte Input con<br>1 : 12byte Input con<br>2 : 16byte Input con<br>3 : 20byte Input con<br>4 : 32byte Input con |

## ■ Arm Manager - 軌道生成

| 項目名 | マクロ名 | 説明 |
|---|---|---|
| 正方向ソフトリミット(J1,deg*10＾3) | AM_JPRM_PLIM1 | 1 軸の正方向ソフトリミット(×1000, 単位：度) |
| 正方向ソフトリミット(J2,deg*10＾3) | AM_JPRM_PLIM2 | 2 軸の正方向ソフトリミット(×1000, 単位：度) |
| 正方向ソフトリミット(J3,deg*10＾3) | AM_JPRM_PLIM3 | 3 軸の正方向ソフトリミット(×1000, 単位：度) |
| 正方向ソフトリミット(J4,deg*10＾3) | AM_JPRM_PLIM4 | 4 軸の正方向ソフトリミット(×1000, 単位：度) |
| 正方向ソフトリミット(J5,deg*10＾3) | AM_JPRM_PLIM5 | 5 軸の正方向ソフトリミット(×1000, 単位：度) |
| 正方向ソフトリミット(J6,deg*10＾3) | AM_JPRM_PLIM6 | 6 軸の正方向ソフトリミット(×1000, 単位：度) |
| 正方向ソフトリミット(J7,deg*10＾3) | AM_JPRM_PLIM7 | 7 軸の正方向ソフトリミット(×1000, 単位：度) |
| 正方向ソフトリミット(J8,deg*10＾3) | AM_JPRM_PLIM8 | 8 軸の正方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J1,deg*10＾3) | AM_JPRM_NLIM1 | 1 軸の負方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J2,deg*10＾3) | AM_JPRM_NLIM2 | 2 軸の負方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J3,deg*10＾3) | AM_JPRM_NLIM3 | 3 軸の負方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J4,deg*10＾3) | AM_JPRM_NLIM4 | 4 軸の負方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J5,deg*10＾3) | AM_JPRM_NLIM5 | 5 軸の負方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J6,deg*10＾3) | AM_JPRM_NLIM6 | 6 軸の負方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J7,deg*10＾3) | AM_JPRM_NLIM7 | 7 軸の負方向ソフトリミット(×1000, 単位：度) |
| 負方向ソフトリミット(J8,deg*10＾3) | AM_JPRM_NLIM8 | 8 軸の負方向ソフトリミット(×1000, 単位：度) |
| RANG(J1,deg*10＾5) | AM_JPRM_RANG1 | 1 軸の RANG 値(×100000, 単位：度) |

## ■ Arm Manager - 使用条件

| 項目名 | マクロ名 | 説明 |
|---|---|---|
| 最適可搬質量設定モード | CND_ASPACC | 最適可搬質量設定モード<br>(0:通常, 1:PTP のみ有効, 2:CP のみ有効, 3:PTP/CP ともに有効) |
| 床置き、天吊り、壁掛け設定 | CND_SPACE | 床置き、天吊りまたは壁掛けの設定<br>(0:床置き, 1:天吊り, 2：壁掛け) |
| 先端負荷質量(g) | CND_LOAD | ロボット先端の負荷質量(単位:g) |
| 負荷重心位置 X6(mm) | CND_LDPOSGX | ロボット先端の負荷重心位置の X 成分(単位:mm) |
| 負荷重心位置 Y6(mm) | CND_LDPOSGY | ロボット先端の負荷重心位置の Y 成分(単位:mm) |
| 負荷重心位置 Z6(mm) | CND_LDPOSGZ | ロボット先端の負荷重心位置の Z 成分(単位:mm) |
| 停止時許容パルス幅 (J1) | CND_PLSWDTH1 | 1 軸の停止時許容パルス幅(単位:pulse) |
| 停止時許容パルス幅 (J2) | CND_PLSWDTH2 | 2 軸の停止時許容パルス幅(単位:pulse) |
| 停止時許容パルス幅 (J3) | CND_PLSWDTH3 | 3 軸の停止時許容パルス幅(単位:pulse) |
| 停止時許容パルス幅 (J4) | CND_PLSWDTH4 | 4 軸の停止時許容パルス幅(単位:pulse) |
| 停止時許容パルス幅 (J5) | CND_PLSWDTH5 | 5 軸の停止時許容パルス幅(単位:pulse) |
| 停止時許容パルス幅 (J6) | CND_PLSWDTH6 | 6 軸の停止時許容パルス幅(単位:pulse) |
| 停止時許容パルス幅 (J7) | CND_PLSWDTH7 | 7 軸の停止時許容パルス幅(単位:pulse) |
| 停止時許容パルス幅 (J8) | CND_PLSWDTH8 | 8 軸の停止時許容パルス幅(単位:pulse) |
| 動作終了タイムアウト(msec) | CND_MVTIMOUT | @E 指定時の動作終了タイムアウト時間(単位:msec) |

## ■ Vision Manager - 一般設定

| 項目名 | マクロ名 | 説明 |
|---|---|---|
| ｶﾒﾗ 1－ｼｬｯﾀ方式 | CA_SHUT1 | ｶﾒﾗ 1 のｼｬｯﾀ方式(0:ﾌｨｰﾙﾄﾞ、 1:ﾌﾚｰﾑ) |
| ｶﾒﾗ 1－入力下限レベル | CA_LEVEL_L1 | ｶﾒﾗ 1 の入力レベル下限 |
| ｶﾒﾗ 1－入力上限レベル | CA_LEVEL_H1 | ｶﾒﾗ 1 の入力レベル上限 |
| ｶﾒﾗ 1－ｶﾒﾗ使用可否 | CA_CONNECT1 | ｶﾒﾗ 1 の使用可否 (0:使用可、 1:未接続、 2：ｵﾌﾟｼｮﾝ 3：なし) |
| ｶﾒﾗ 2－ｼｬｯﾀ方式 | CA_SHUT2 | ｶﾒﾗ 2 のｼｬｯﾀ方式(0:ﾌｨｰﾙﾄﾞ、 1:ﾌﾚｰﾑ) |
| ｶﾒﾗ 2－入力下限レベル | CA_LEVEL_L2 | ｶﾒﾗ 2 の入力レベル下限 |
| ｶﾒﾗ 2－入力上限レベル | CA_LEVEL_H2 | ｶﾒﾗ 2 の入力レベル上限 |
| ｶﾒﾗ 2－ｶﾒﾗ使用可否 | CA_CONNECT2 | ｶﾒﾗ 2 の使用可否 (0:使用可、 1:未接続、 2：ｵﾌﾟｼｮﾝ 3：なし) |
| ｶﾒﾗ同期方式 | CA_SYNC | ｶﾒﾗの同期方式(0:ｶﾒﾗ内部同期、 1:ｶﾒﾗ外部同期) |
| ShCorn－距離 | SH_CORN_DIST1 | ShCorner 命令の計測条件 (ﾊﾟｯﾄﾞとｺｰﾅｰの距離) 1～10 |
| ShCorn－間隔 | SH_CORN_DIST2 | ShCorner 命令の計測条件 (ﾊﾟｯﾄﾞとﾊﾟｯﾄﾞの間隔) 1～10 |
| ShCorn－幅 | SH_CORN_WIDTH | ShCorner 命令の計測条件 (ﾊﾟｯﾄﾞの幅) 1～10 |
| ShCorn－高さ | SH_CORN_HEIGHT | ShCorner 命令の計測条件 (ﾊﾟｯﾄﾞの高さ) 1～10 |
| ShCirc－間隔 | SH_CIRC_DIST | ShCircle 命令の計測条件 (ﾊﾟｯﾄﾞとﾊﾟｯﾄﾞの間隔) 1～10 |
| ShCirc－幅 | SH_CIRC_WIDTH | ShCircle 命令の計測条件 (ﾊﾟｯﾄの幅) 1～10 |
| ｻｰﾁﾀｲﾑｱｳﾄ時間(ms) | SH_TIMEOUT | ｻｰﾁ計測のﾀｲﾑｱｳﾄ時間 |
| 視覚ﾓﾆﾀ表示位置 | VS_DISP_LOC | 処理画面を視覚ﾓﾆﾀへ表示する位置(0:中央、 1:左、 2:右) |

## ■ Log Manager - 通信仕様

| 項目名 | マクロ名 | 説明 |
|---|---|---|
| タイムアウト時間(msec) | COM_RTK_TOUT | 通信時のタイムアウト時間(単位:msec) |
| タイムアウト発生時のリトライ回数 | COM_RTK_TRETRY | 通信タイムアウト発生時のリトライ回数 |
| NAK 発生時のリトライ回数 | COM_RTK_RETRY | NAK(パケット異常)発生時のリトライ回数 |
| RS232C(2).通信権 | COM_RS2_ACCESS | コントローラの RS232C チャンネル 2 の通信権の設定 (0:使用不可, 1:読込のみ可, 2:読込/書込可能) |
| RS232C(2).ボーレート | COM_RS2_SPEED | コントローラの RS232C チャンネル 2 のボーレートの設定 (単位：bps) |
| RS232C(2).パリティ (0:偶数, 1:なし, 2:奇数) | COM_RS2_PARITY | コントローラの RS232C チャンネル 2 のパリティの設定 (0:偶数, 1:なし, 2:奇数) |
| RS232C(2).データ長 | COM_RS2_DATALEN | コントローラの RS232C チャンネル 2 のデータ長の設定 (単位：bit) |
| RS232C(2).ストップビット (0:1bit, 1:1.5bit, 2:2bit) | COM_RS2_STOPBIT | コントローラの RS232C チャンネル 2 のストップビットの設定(0:1bit, 1:1.5bit, 2:2bit) |
| RS232C(2).デリミタ (0:CR, 1:CR+LF) | COM_RS2_DELIMITER | コントローラの RS232C チャンネル 2 のデリミタの設定 (0:CR, 1:CR+LF) |
| RS232C(3).通信権 | COM_RS3_ACCESS | コントローラの RS232C チャンネル 3 の通信権の設定 (0:使用不可, 1:読込のみ可, 2:読込/書込可能) |
| RS232C(3).ボーレート | COM_RS3_SPEED | コントローラの RS232C チャンネル 3 のボーレートの設定 (単位:bps) |
| RS232C(3).パリティ (0:偶数, 1:なし, 2:奇数) | COM_RS3_PARITY | コントローラの RS232C チャンネル 3 のパリティの設定 (0:偶数, 1:なし, 2:奇数) |
| RS232C(3).データ長 | COM_RS3_DATALEN | コントローラの RS232C チャンネル 3 のデータ長の設定 (単位：bit) |
| RS232C(3).ストップビット (0:1bit, 1:1.5bit, 2:2bit) | COM_RS3_STOPBIT | コントローラの RS232C チャンネル 3 のストップビットの設定(0:1bit, 1:1.5bit, 2:2bit) |
| RS232C(3).デリミタ (0:CR, 1:CR+LF) | COM_RS3_DELIMITER | コントローラの RS232C チャンネル 3 のデリミタの設定 (0:CR, 1:CR+LF) |

| 項目名 | マクロ名 | 説明 |
|---|---|---|
| RS232C(4).通信権 | COM_RS4_ACCESS | コントローラの RS232C チャンネル 4 の通信権の設定<br>(0:使用不可, 1:読込のみ可, 2:読込/書込可能) |
| RS232C(4).ボーレート | COM_RS4_SPEED | コントローラの RS232C チャンネル 4 のボーレートの設定<br>(単位：bps) |
| RS232C(4).パリティ<br>(0:偶数, 1:なし, 2:奇数) | COM_RS4_PARITY | コントローラの RS232C チャンネル 4 のパリティの設定<br>(0:偶数, 1:なし, 2:奇数) |
| RS232C(4).データ長 | COM_RS4_DATALEN | コントローラの RS232C チャンネル 4 のデータ長の設定<br>(単位：bit) |
| RS232C(4).ストップビット<br>(0:1bit, 1:1.5bit, 2:2bit) | COM_RS4_STOPBIT | コントローラの RS232C チャンネル 4 のストップビットの設定(0:1bit, 1:1.5bit, 2:2bit) |
| RS232C(4).デリミタ<br>(0:CR, 1:CR+LF) | COM_RS4_DELIMITER | コントローラの RS232C チャンネル 4 のデリミタの設定<br>(0:CR, 1:CR+LF) |
| Ethernet.通信権 | COM_ET_ACCESS | コントローラの Ethernet の通信権の設定<br>(0:使用不可, 1:読込のみ可, 2:読込/書込可能) |
| Ethernet.IP アドレス | COM_ET_IP_ADDR | コントローラの Ethernet の IP アドレスの設定 |
| Ethernet.サブネットマスク | COM_ET_SUBNET_MASK | コントローラの Ethernet のサブネットマスクの設定 |
| Ethernet.接続先 1.ローカルポート | COM_ET_LPORT1 | コントローラの Ethernet の接続先 1.ローカルポートの設定 |
| Ethernet.接続先 2.ローカルポート | COM_ET_LPORT2 | コントローラの Ethernet の接続先 2.ローカルポートの設定 |
| Ethernet.接続先 3.ローカルポート | COM_ET_LPORT3 | コントローラの Ethernet の接続先 3.ローカルポートの設定 |
| Ethernet.接続先 4.ローカルポート | COM_ET_LPORT4 | コントローラの Ethernet の接続先 4.ローカルポートの設定 |
| Ethernet.接続先 5.ローカルポート | COM_ET_LPORT5 | コントローラの Ethernet の接続先 5.ローカルポートの設定 |

# デンソーロボット

---

プログラミングマニュアルⅠ 基礎知識とコマンド

初　版　2009年　4月
第3版　2011年　1月
第4版　2011年　4月

株式会社デンソーウェーブ　4N**C

---

● この取扱説明書の一部または全部を無断で複製・転載することはお断りします。

● この説明書の内容は将来予告なしに変更することがあります。

● 本書の内容については、万全を期して作成いたしましたが、万一ご不審の点や誤り、記載もれなど、お気づきの点がありましたら、ご連絡ください。

● 運用した結果の影響については、上項にかかわらず責任を負いかねますのでご了承ください。