第 8 章 コマンドリファレンス

8-1 BL/1 文法

BL/1 は、Basic ライクなインタプリタです。基本的な操作、記述方法は Basic インタプリタに準拠しています。

プログラムの構成

プログラムは行単位で管理され、以下のような順序で記述します。
それぞれの構成要素と文字数などには制限がありますので注意してください。

470	**IF**	**HPT(XIN0)==0**	**THEN**	**:**	**RMVS**	**X_A 5000**	**:**	**END_IF**
文番号	コマンド	引数	予約語	コロン	コマンド	引数	コロン	コマンド

行	1 行に入力可能な文字数は、255bytes です。
コマンドライン	文番号、コマンド、引数から成りたつ最小限の実行単位です。
引数の数	引数はコマンドの仕様によって決定されますが、複数入れられるものでは最大 14 個迄。
引数の文字数	引数は、式、演算式、定数、変数、文字列、ラベルなどを与えます。 最大 100 文字です。
マルチステートメント	コロンを付加することによりコマンドラインを 1 行の中に複数記述することができます。
プログラムとコマンド	実行単位で、文番号を省略するとコマンドとして即実行されます。 文番号にしたがって、実行単位は、プログラム順序を与えられます。
コメント	'(シングルクォート) により ' クォート以降はコメントとして実行されません。
ラベル	*(アスタリスク) から始まる文字列は、ラベル文となり、実行文ではありませんが、GOTO GOSUB　FORK などの実行先頭を定義します。 ラベル文字数は、引数文字数制限まで可能です。

変数，定数

変数	4byte 長の整数で、15 文字以内です。A-z,0-9,$,_,@ が使用可能です。 変数は、2000 個まで使用することができます。 変数の初期値は、フラッシュ ROM に固定されます。 NEW 後、プログラムをロードして RUN すると、すべて 0 となりますが、途中で変更して RUN を繰り返すと、その時点での値が保持されます。 このため、プログラムではかならず変数の初期化を心がけてください。
タスク・ ローカル変数	末尾に _ の付加された変数で、タスクごとに独立した値を持ちます。 AB_ など、32 個まで定義できます。初期値は不定です。
文字列変数	末尾に $ の付加された変数で、文字列変数となります。A$ など。 文字列は 128 個まで使用することができ、文字列最大長は 255 まで。
配列変数	DIM コマンドによってラベルを与えて宣言できる配列変数です。 合計で 20000 個まで使用することができます。また、2 次元配列も使用可能です。 配列要素は、RUN 後 DIM コマンドが実行されるごとに確保されます。このため、DIM 宣言はプログラム中先頭にまとめて配置し、バックアップ変数などとして使用できるようにします。

予約配列	点データ	P(n)={X(n),Y(n),Z(n),U(n)}　n=1 〜最大 32000* * 機能により異なります。
	タッチパネル共有変数	MBK(m) m=0 〜 8099
	それぞれ部分的に文字列配列としても使用可	
定数	BL/1 があらかじめ持っている数値です。コマンドの入力オプションなどに使用します。	
文字列定数	“”(ダブルクォート) で囲まれた文字列です。通信や、文字列処理に使用できます。 注）文字列定数中、¥n(LF),¥r(CR),¥t(TAB) となります。また ¥ はバックスラッシュと同じ意味、同じ文字です。	

式・条件式

BL/1 では、式･条件式に区別はありません。条件式は、1 か 0 の論理値を持った関数や演算の集まりです。
A==B　結果は成立で 1、非成立で 0
SW(n)　結果は ON 状態で 1、オフ状態で 0
A=B+C　結果は、B と C の和を A に代入するため、整数値を持つ。

従って、通常の条件式で変数や整数値を持つ関数を混在させる場合は、結果が 1 もしくは、0 になるように、適切な演算式や関数を使用する必要があります。
　C*(A+B)>=1000　→ 演算ではあるが、比較演算子 >= によって結果は 1 もしくは 0 となる。

比較演算子は以下のとおりです。

>　左辺大	<　左辺小
>=　左辺大、同値含む	<=　左辺大、同値含む
==　一致	!=　不一致 (<> も使用可です)

なお、MPC では、比較演算子は、後述の算術演算子と区別はなく、演算結果が 0 か 1 しかないという二項演算として扱われています。
このため、比較演算子は算術演算子と混在して使用できますが、演算の優先順位はなく左から順に実行されることに注意が必要です。

1+2>3+4　: 結果は 4 となります。1+2>3 が先に実行され、その結果に 4 が足されます。
1+2>(3+4) : 結果は 0 となり、両辺の比較が最後に実行されます。

この例が示すように、左辺は自然に左から実行されますが、右辺の計算が先にされるためには、() が必要となります。通常は、左辺に複雑な計算式を置き、右辺は数値や変数のみしておくと、記述が簡単になります。
例えば、以下の記述例は、比較として同じ意味ですが、上の例の方が演算の優先順位を () で規定する必要がありません。

```
A*B+C*D>E
E<(A*B+C*D)
```

この仕様により、MPC では以下のような複雑な論理処理を一つの式で記述することができます。

```
((SW(0)==1)&(SW(1)==1)&(DAT>1000))|(SW(2)==1)
```

また、式の意味をより明示的にするには、OR AND といった論理接続詞を使用したほうがよい場合があります。

```
(SW(0)==1)&(SW(1)==1)&(DAT>1000) OR SW(2)==1
```

算術演算の順序は、左から順次実行されますが、加算・減算に対して乗算 , 除算のみ先に演算されます。
C+A*B の場合、A*B が実行されてから、C を足します。
ここで、和算優先とする場合は、(C+A)*B と記述して優先演算を明示します。

用意されている二項演算は、以下のとおりです。うち、","と";" は、ワード合成演算子です。また、"," は、( ) の中でのみ有効です。

```
#prx 1,2
00000001 00000002
#prx (1,2)
00010002
##prx 1;2
01000002
```

二項演算子			
+	加算	<<	左シフト （× 2n ）
-	減算	>>	右シフト （/2n ）
*	乗算	,	ワード合成
/	除算	;	上位バイト
%	剰余算	&	論理積
^	排他的論理和	|	論理和

文字列の演算

文字列演算では加算と比較 (一致) のみ許されます。

```
A$=C$+"1234"
IF A$==C$ THEN
```

その他の文字列処理では、ポインタの概念が導入されており、効率の良い文字列処理が可能となっています。SERCH,SUBST,VAL などの関数を参照してください。
なお、文字列配列として、点データエリアを文字列として使用する P$() があります。

ベクトル引数

XY ロボットコマンドでは、4 次元のベクトル量を扱うことが多くなります。4 次元の要素は XYZ の直交 3 次元の座標と姿勢軸に相当する U となります。このベクトル量の表現には P(n) を使用したものと、座標値を直接指定する 2 通りがあります。なお、座標値指定の場合は、

点表現	座標値表現
JUMP P(n)	JUMP VALx VALy VALu VALz
MOVS P(n)	MOVS VALx VALy VALu VALz
BACKLASH P(n)	BACKLASH VALx VALy VALu VALz

座標値表現の特殊ルール	例	意味
引数 VOID の軸は動作しない *	MOVS VOID 100 200 VOID	XZ は無動作
不足引数は VOID が与えられる	MOVS 1900 200 300	Z は無動作
軸指定 , 一個の値では同一値	MOVS X_A|Y_A 100	XY とも 100、他は無動作

さらに点表現では、以下のような指定が可能です。
ベクトル引数 ＝{ 軸指定 + P(n) + AD_P( ) }

```
MOVS VOID_U P(1)                引数は P(1)。ただし VOID_U により U 軸は無動作
MOVS P(1) AD_P(X_A,VAL)         引数として P(1)。X の値に VAL を加算
MOVS P(1) AD_P(P(100))          引数として P(1)。P(1) に P(100) を加算。
MOVS VOID_U P(1) AD_P(P(100))   引数として P(1)。P(1) に P(100) を加算。U 軸は動作無。
```

このベクトル引数が有効なコマンドは、STPS,BACKLASH,JUMP,JMPZ,MOVS,MOVL です。
RMVS, RMVL,RMVC, については、相対移動コマンドであるため、ベクトル引数は使用できません。

8-2 コマンドリファレンス

@

演算　　関数

■書 式
　@(arg)

■使い方
　IF @(A==1) THEN
　IF @((A!=1) &(B!=1)) THEN

■機 能
　論理反転

■解 説
　1->0 0->1 に変換する論理反転です。
　NOT() がロング型反転であるのに大して、@() は 0 か 1 しか返しません。

@SW

IO　　関数

■書 式
　@SW(arg)

■使い方
　IF (@SW(0)&SW(1))==1

■機 能
　入力ポートの反転読み取り

■解 説
　SW の値を論理反転して返す。

```
LIST
10              ON   -1 -3 -5
20              PRINT   @(SW(-1))|@(SW(-3))|@(SW(-5))
30              ON   -1 -3 -5
40              PRINT   @SW(-1)|@SW(-3)|@SW(-5)
50              PRINT   SW(-1)|@SW(-3)|@SW(-5)
#run

*
Compiling
————————
0
0
1
#
```

ABS

演算　　関数

■書 式
　ABS(arg)

■使い方
　A=ABS(-100)

■機 能
絶対値を得る

■解 説
引数を正の整数に変換して返す。
A=-123
A=ABS(A)
A は 123 となります。

ACCEL

パルス発生 コマンド

■書 式
ACCEL [axis] PPS [leng,lo_pps]

■使い方
ACCEL 4000
ACCEL 4000 1000 100
ACCEL Z_A 8000
ACCEL SACL 4000
ACCEL X_A|SACL 2000
ACCEL X_A|OUTSL 30000

■機 能
加速度設定

■解 説
PG の加速度を設定します。軸指定を省略すると、全軸に適用されます。
指定パラメータは、最大速度 (pps)、加速距離 (pulse), 自起動 (pps) です。
加速距離以下を省略するとデフォルト値を設定します。
このコマンドはパルス発生中には使用できません。パルス発生中は SPEED コマンドを用いて下さい。
また、自起動速度を低く設定すると、全体の動作が遅くなります。
ステップモータでも 100pps 程度、サーボモータでは 1kpss 程度を最小速度の目安として下さい。
(1pps を最小速度とすると、1 パルスの出力に 1 秒必要になります。もし、動作の最初と最後に 1pps が発生すると、それだけで二秒ずつ必要になります。)

・軸指定パラメータに定数 SACL を OR すると、S 字加減速となります。
例 : ACCEL X_A|SACL 80000

・軸指定パラメータに定数 OUTSL を OR すると、RANGE 設定値と現在位置が比較され、出力ポートに反映されます。(MPG-2314 CEP128D 以降)
RANGE X_A 10000 XXX の場合 X(0) が 9999 までは O0 が OFF,10000 以上になると ON
RANGE X_A -10000 XXX の場合 X(0) が -10001 までは O0 が OFF,-10000 以上になると ON
*O0 は、MPG-2314 J4-19 です。

なお、引数無しで実行すると、設定パラメータと、設定した ACCEL の文番号を表示します。文番号が 0 の場合はプログラムによっては設定されていないことを意味します。

```
#accel
X=> Max=3000 Length=150 Min=300 Feed=100 Set@20
Y=> Max=3000 Length=150 Min=300 Feed=100 Set@30
U=> Max=8000 Length=400 Min=800 Feed=100 Set@0
Z=> Max=3000 Length=150 Min=300 Feed=100 Set@40
#
```

ACOS,ATAN

浮動小数点　　関数

■書　式

ATAN(v)
ACOS(v)

■使い方

FP(0)=DEG(ACOS(1/SQR(2)))
FP(1)=DEG(ATAN(1))

■機　能

逆三角関数

■解　説

ラジアン引数出力の倍精度逆三角関数です。浮動小数点演算式中でのみ、意味を持ちます。

```
FP(0)=DEG(ACOS(1/SQR(2)))
FP(1)=DEG(ATAN(1))
```

AD

AD_DA　　関数

■書 式

AD(ch)
AD(fnc,ch)

■使い方

A=AD(0)
IF AD(1,7)>500 THEN

■機 能

MPC-AD12 のデータ取得

■解 説

MPC-1000/MPC-N816/MPC-1200 の AD 機能は、AD(20) ～を使用します。これについては、
単純な読み取り変換機能だけで、MPC-AD12 のようなマクロ機能はありません。

以下は、MPC-AD12 に対する説明です。

【1msec サンプリング】
関数 AD(ch) は AD コンバータの変換値を返します。ch 番号は、0 ～ 7 です。
このデータは、1msec ごとに更新されています。
MPC-AD12 をさらに一枚追加した場合は、ch 番号として 8 ～ 15 を指定します。
返される値は、AD7890-4(標準出荷状態) 搭載で 0 ～ 4095 1mV/1digit
AD7890-10 搭載の場合は、-2048 ～ 20471mV/1digit
です。
・平均値を得る方法
AD(1,ch) 平均値を返します。平均をとるデータ数の指定は SET_AD コマンドで指定します。
(デフォルトは 8 個ごとの平均値)

【自動連続データ取得】
MPC-AD12 は 1msec ごとにデータを取得しており、832 連続してその値を取得、参照できます。

AD(2,ch) スタート連続データ取得 1msec サンプリング
AD(4,ch) スタート連続データ取得 2msec サンプリング
AD(3,ch) 取得完了待ち

データの取り出しは、AD_D(0,n) n:0 ～ 831

【8CH 自動連続データ取得】
ch を 8 に指定すると全 ch 同時サンプリングします。この場合レートは 1msec 固定。
各 ch 104 個のデータを取得します。(0.1 秒)
AD(2,8) スタート連続データ取得 1msec サンプリング
AD(3,0) 取得完了待ち

データの取り出しは、AD_D(ch,n) ch:0 ～ 7 n:0 ～ 103

【その他の機能】
MPC-AD12 には、30 μ sec ～ 100 μ sec の高速サンプリングも用意されています。
これについては、ハードリファレンスを参照ください。

```
'1msec SAMPLING
 SYCSCLK=0
   FOR   i=0 TO 1440 STEP 2
    WAIT   i==SYSCLK
    X(i+1000)=AD(0)
   NEXT

'Bulk Sampling
SYSCLK=0
 dmy=AD(2,ch)
  PRINT   AD(3,ch) SYSCLK "1msec"
  PRINT   "dump"
  FOR   i=0 TO 800 STEP 50
   PRINT   i AD_D(0,i)
  NEXT

'8CH Bulk sampling
dmy=AD(2,8)
  PRINT   AD(3,0) SYSCLK
  FOR   i=0 TO 100 STEP 10
   PRINT   i AD_D(0,i) AD_D(1,i) AD_D(2,i) AD_D(3,i)
  NEXT
```

ADD_MBK

タッチパネル　　コマンド

■書 式
ADD_MBK add_value adrs

■使い方
add_mbk 1000 1

■機 能
MBK() 配列の直接加算

■解 説
配列 MBK() のデータを直接加算します。

```
#pr mbk(1)
1000
#add_mbk 1000 1
#pr mbk(1)
2000
#
```

ADD_STR

文字列　　コマンド

■書 式

ADD_STR Str [Str]

■使い方

ADD_STR "Win" a$
ADD_STR "7"

■機 能

文字列のアペンド

■解 説

ADD_STR は、指定文字列に文字列を追加します。最初は、追加先の文字列変数の指定と、初期値を与えます。

ADD_STR "Win" a$

この時点で、a$ には、Win がコピーされます。以後は追加文字列を指定するだけで、文字が追加されます。

ADD_STR "7"

この結果は、a$ が Win7 となります。ADD_STR は以下の記述によってヌル・コードも追加することができます。

ADD_STR chr$(0)

サンプルプログラムは、01,03,00,00,01,01 を出力する場合の記述です。

```
----------------------------------------
CNFG#  2 "38400b8pns1NONE"
     CH=2
      ADD_STR   CHR$(1) SEND$
      ADD_STR   CHR$(3)
      ADD_STR   CHR$(0)
      ADD_STR   CHR$(0)
      ADD_STR   CHR$(1)
      ADD_STR   CHR$(1)
      PRINT#   CH STR_LEN|6 SEND$
     END
```

AD_D

AD_DA　　関数

■書 式

AD_D(ch,n)

■使い方

a=AD_D(0,1)

■機 能

連続取り込みデータの読み出し

■解 説

連続サンプリングのデータの取り出し。単 ch の場合は、AD_D(0,n)　n: 0 ～ 831
全 ch の場合は、AD_D(ch,n)　　ch:0 ～ 7 n:0 ～ 103

AD_P

パルス発生　　関数

■書 式

AD_P(axs,n) n=+/-32767
AD_P(P(n))

■使い方

MOVS P(n) AD_P(X_A,1000)
MOVS P(n) AD_P(X_A,1000) AD_P(Z_A,-1000)
JUMP P(n) AD_P(P(m))

■機 能

移動点補正

■解 説

MOVSなどの点データ引数(座標値)に補正値を加える。指定点の上で停止させるのに使用します。
また画像処理で X,Y 点などを一時的に補正させるのに使用できます。
点データを指定すると、4 軸の座標値がそのまま加算されます。この演算では、点データそのものを修正しません。AD_P(axs,n) の場合の補正範囲は +/-32767 の範囲です。

```
MOVS P(5) AD_P(X_A,1000) =>MOVS X(5)+1000 Y(5) U(5) Z(5)
MOVS P(6) AD_P(X_A,1000) AD_P(Z_A,-1000) =>MOVS X(6)+1000 Y(6) U(6) Z(6)-1000
```

AFFIN

浮動小数点　　コマンド

■書 式

AFFIN　n m k deg

■使い方

AFFIN　2 1 3 i*10000

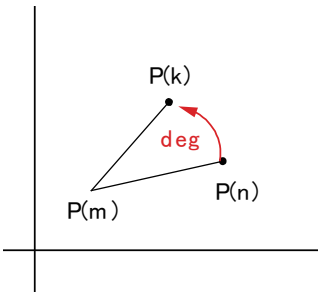
■機 能

点の回転演算

■解 説

P(n) を P(m) を中心に deg 度回転し結果を P(k) に代入します。
角度は 10000 倍した値を与えます。
サンプルでは、X 方向の水平線を 30 度 ccw 方向に回転しています。

```
#setp 1 10000 20000  0 0
#setp 2 1010000  20000   0 0
#affin 2 1 3  300000
#pr p(3)
 876025 520000 0 0
#
```



ALL_A

パルス発生　　予約定数

■書 式

ALL_A

■機 能

全軸指定

■解 説

対象ボード : MPG-2314/MPC-1200

```
ACCEL ALL_A 30000 1000 500              /* Acceleration/deceleration setting
FEED ALL_A 100                          /* Speed setting
INSET ALL_A MD_2PLS|ALM_OFF|LMT_OFF     /* In port set
STOP ALL_A STP_D                        /* Moving stop with deceleration
WAIT RR(ALL_A)==0                       /* Wait until moving complete
   etc
```

ALL_E

パルス発生　　予約定数

■書 式
ALL_E

■機 能
全軸エラー指定

■解 説
対象ボード : MPG-2314
移動後のエラーの有無を調べます。次のビットのどれかが立ったことを表します。
RR1 レジスタ (ドライブ終了ステータス) ENG,ALARM,LMT-,LMT+
RR2 レジスタ (エラー情報) EMG,ALARM,HLMT-,HLMT+,SLMT-,SLMT+

```
100  MOVL P(1)
110  WAIT RR(ALL_A)=0
120  IF RR(ALL_E) !=0 THEN        /* Confirming error status
130    PRINT "ERROR STOP"
140  ELSE
150    PRINT "NORMAL STOP"
160  END_IF
170  PRX RR(ALL_E)
```

ALM

パルス発生　　予約定数

■書 式
ALM

■機 能
エラービット指定

■解 説
対象ボード : MPG-2314
アラーム信号ビット

```
IF LMT(X_A,ALM) !=0 THEN          /* confirming reason for stop
```

ALM_OFF

パルス発生　　予約定数

■書 式
ALM_OFF

■機 能
アラーム設定

■解 説
対象ボード : MPG-2314
アラーム OFF で有効

```
INSET X_A ALM_OFF                 /* X-axis 'ALARM' enabled on signal 'OFF''
```

ALM_ON

パルス発生　　予約定数

■書 式
ALM_ON

■機 能
アラーム設定

■解 説
対象ボード : MPG-2314
アラーム ON で有効

```
INSET X_A ALM_ON                    /* X-axis 'ALARM' enabled on signal 'ON'
```

ASC

文字列　　関数

■書 式
ASC(str)
ASC(arg)

■使い方
ASC(a$)
ASC(4)

■機 能
文字列のアスキーコードを得る

■解 説
引数に文字列を与えると先頭の文字コードを返します。
0 ～ 4 の数値を与えると、ptr_ の位置から、与えられた数だけ文字コードを読み取っていきます。
このため、4 文字以内の文字列比較を簡単に行うことができます。

```
10      a$="123abcABC456"
20       PRINT   ASC(a$)
30       SERCH   a$ "abc"
40       FOR   i=0 TO 4
50        PRX   ASC(i)
60       NEXT   i
#run

49
00000041
00000041
00004241
00434241
34434241
#
```

ATAN

浮動小数点　　コマンド

■書 式
atan y r var [x]

■使い方
atan 10000 1000 a
atan 100000 1000 a 173205

■機 能
ATAN 演算

■解 説
ATAN 浮動小数点演算を行います。結果は度で、整数化されます。
整数化する時の倍率を r によって決定することができます。
var = r × atan(y/x)
注 1) x を省略すると、x を 10000 とします。
注 2) 結果 (度) の範囲は -90 ～ +90 となります。

例 1) atan 10000 1 a
この計算は二等辺直角三角形の ATAN 値で、結果は 45(度) です。
a=1 × atan(10000/10000)

例 2) atan 17321 1000 a 10000
この計算は 60 度の直角三角形の ATAN 値です。
a=1000 × atan(17321/10000)
60 度を 1000 倍するため、60000 という値になります。

```
-----------------------------------------
#atan 10000 1000 a
#pr  a
 45000
#atan 100000  1000 a  173205
#pr  a
 30000
#atan 17321 1000 a 10000
#pr  a
 60000
#
```

ATAN2

浮動小数点　　コマンド

■書 式
atan2 y x var [r]

■使い方
atan2 100000 100000 a3
atan2 100000 173205 a3 10000

■機 能
ATAN 演算

■解 説
ATAN 浮動小数点演算を行います。ATAN との違いは引数の順序のみです。
結果は度で、整数化されます。整数化する時の倍率を r によって決定することができます。
var = r × atan2(y/x)
注 1) r を省略すると、r を 10000 とします。
注 2) y>x の場合は、atan2(x/y) を算出しその補角により角度を算出します。このため x=0 でも正しい値を返すことができます。
注 3) 結果 (度) の範囲は -90 ～ +90 となります。

例 1) atan2 10000 10000 a
この計算は二等辺直角三角形の ATAN 値で、結果は 45(度) です。
r は省略されているので結果は 10000 倍されます。
a=10000 × atan(10000/10000)

例 2) atan2 173205081 100000000 a 100000
この計算は 60 度の直角三角形の ATAN 値です。

a=100000 × atan(173205081/100000000)
60 度を 100000 倍するため、6000000 という値になります。

```
#atan2 100000 100000 a
#pr  a
 450000
#atan2 100000 173205 a 10000
#pr  a
 300000
#
```

AUTO_RESET_1

IO　　予約変数

■書 式
AUTO_RESET_1

■使い方
AUTO_RESET_1=1000

■機 能
オートリセット

■解 説
参照　COUNTER_1

AUTO_RESET_2

IO　　予約変数

■書 式
AUTO_RESET_2

■使い方
AUTO_RESET_2=500

■機 能
オートリセット

■解 説
参照　COUNTER_1

AVOID

IO　　予約定数

■書 式
AVOID

■機 能
I/O コマンド無効化

■解 説
I/O コマンドの無効化。サンプルプログラムでは、sol1 を AVOID として、sol1 に対する ON コマンドを無効としています。

```
10      CONST   sol1 AVOID          /* not use
20      CONST   sol2 1
30      ON   sol1 sol2              /* sol1 disable, sol2 enable
```

BACKLASH

パルス発生　　コマンド

■書 式
BACKLASH Xb Yb Ub Zb

■使い方
BACKLASH 111 121 0 0

■機 能
バックラッシュ補正設定

■解 説
MPG-2314 のパルス出力に、バックラッシュ補正を与えるものです。
バックラッシュ補正は、単軸、直線補間のみに有効です。円弧補間には適用されません。

パワーオンリセット後は、0 となっており、電源切断後、都度設定が必要となります。
バックラッシュ補正は、パルス発生方向の変わり目でバクラッシュ設定されたパルスが加算されるものです。
使い方には注意が必要で、機械系のバックラッシュ状態を初期化しておく必要があります。
例えば、原点復帰後、CW 方向にバックラッシュ量以上にダミー移動させてバックラッシュ値を正の値に設定します。
パルス発生方向がバックラッシュ値と同じである限り、バックラッシュパルス加算は行われませんが、パルス発生が負方向になったとき、バックラッシュ値を負の値に変換し、パルス加算を行います。従ってバックラッシュ値は内部で、動作によって正負に反転させられており、方向監視もかねています。

なお、バックラッシュ補正は、万能ではありません。
機械系のバックラッシュ量は、移動速度、負荷、振動などの条件によって変動します。
機械系の特性をよく把握した上で使用してください。

BAT

保守　　コマンド

■書 式
BAT(arg)

■使い方
IF BAT(0)==1 THEN : PRINT "Battery error" : END_IF

■機 能
バッテリ エラー番号を得る

■解 説
前回の電源オフ時に正しく CPU が退避状態になったかどうかを示す関数です。0 で正常。
1 が返ってきたら、電源断時の CPU 異常、又はバックアップ電池が尽きている。のどちらかです。
バッテリ・エラーがあった場合、点データ、MBK データなどが破壊されている可能性があります。

BATTERY

保守　　予約変数

■書 式
BATTERY

■使い方
IF BATTERY != 0 THEN
MBK(20)=BATTERY

■機 能
電池状態

■解 説
電池を搭載している MPC-2000 で、電源オフ時に、電池電圧が低下すると、パワーオン直後に BATTERY OUT もしくは BATTERY LOW という表示が出ます。
BATTERY　OUT は電池完全消耗か電池自体が抜き取られたことを意味します。
BATTERY　LOW は電池電圧低下です。
BATTERY　OUT の場合は、予約変数 BATTERY が 1,BATTERY　LOW の場合は 2 となります。

BREAK

制御文　　ステートメント

■書 式
BREAK

■使い方
```
DO
IF SW(0)==1 THEN : BREAK : END_IF
LOOP
```

■機 能
FOR-NEXT,DO-LOOP,WHILE-WEND の繰り返し実行のキャンセル

■解 説
複数の条件でエンドレス実行をキャンセルさせる場合は、DO-LOOP で記述しておき、IF 文で BREAKを実行するほうが、明快な表現となる。BREAK文は、ループ中でどこにでも複数記述できる。

BREAK_POINT {BKP}

保守　　コマンド

■書 式
BKP [args]

■使い方
```
BKP 100
BKP 100 110
BKP *aa
BKP 0
```

■機 能
ブレークポイントの設定

■解 説
BREAK_POINT コマンドにより、8 個までの指定した文番号でプログラムを停止させることができます。(ラベル指定も可能です)
サンプルプログラムようにプログラム番号を指定すると、指定行を表示します。
その後指定行の文番号は、FTMW 上では反転表示されます。ブレークポイントは、順々に文番号を指定します。指定した文番号を解除する場合は、同じ番号を入力します。
どの文番号が登録されているかは、BKP コマンドを引数なしで実行します。
また、すべてのブレークポイントを解除するには、BKP 0 と入力します。

ブレークが発生したら

1) 実際にブレークポイントを指定し RUN させると、指定位置で実行が中断されます。
そして、中断した行と、タスク番号が表示されます。n キーによって、次のブレークポイントまで実行再開します。このプログラムでは、文番号 30 を通るたびに実行前にブレークします。
2) ステップ送り (1 行ずつ継続的に実行) させる場合は t を押します。 ステップ送りの解除は、n を押します。

3) ブレーク停止中に変数や関数の値を参照することができます。
‘p’ を押して、続けて変数名や関数名を入力します。
4) ブレークポイントを追加することもできます。
‘b’ を押して文番号を入力すると、ブレークポイントを追加することができます。
5) ブレーク中にそのブレークポイントを解除したい場合は、”u” を入力します。
6) プログラム実行を停止する場合は ‘e’ を押します。

なお、FTMW6.39s 以後、ブレークポイントはメニューで使用できます。

```
30       FORK   2 *bb
40       END
110     *bb
120      DO
130       FOR   i_=8 TO 15
140        ON   i_ : TIME   50 : OFF   i_
150       NEXT
160      LOOP
#bkp 110 140

110     *bb
140      ON   i_ : TIME   50 : OFF   i_
#bkp
BREAK_POINT 0=110
BREAK_POINT 1=140
#bkp 110

110     *bb
#bkp
BREAK_POINT 0=140
#
```

CANCEL_RETURN

制御文　　ステートメント

■書 式
CANCEL_RETURN

■使い方
CANCEL_RETURN : GOTO *AAAA

■機 能
RETURN スタックの破棄

■解 説
禁じ手です。
RETURN スタックを破棄します。サブルーチンから RETURN 文でもどらず、親ルーチンのラベルなどに直接飛ばす場合などに使用します。むやみに使うべきではありません。

```
FOR   i=1 TO 100
 s=0
  GOSUB   *aho
 NEXT
 PRINT   "normal" i j s
 END
*baka
 PRINT   i j s
 GOTO  *init
*aho
 FOR   j=0 TO 100
 s=s+j
  IF   j==50 THEN  : CANCEL_RETURN   : GOTO  *baka : END_IF
```

```
NEXT  j
RETURN
```

CCW

パルス発生　　予約定数

■書 式

CCW

■機 能

原点復帰サーチ方向指定
円弧補間指定

■解 説

SHOM では原点復帰の Z 相サーチ方向を指定します。MOVT では円弧補間の回転方向を指定します。

```
SHOM X_A|Y_A IN0_ON|CCW  /* set HOME condition. CCW movement until the sensor turns on
MOVT X_A|Y_A P(102) CCW          /* continuous interpolation. CCW revolution.
RMVC X_A CCW                     /* infinite pulse generation. CCW movement.
```

CHR$

文字列　　関数

■書 式

CHR$(arg)

■使い方

a$=CHR$(15)
print# 1 chr$(10)

■機 能

一文字生成

■解 説

a$="slkd" などで表現できない文字を生成します。
CHR$(1) ==> SOH 等です。

CHR_C

通信　　予約定数

■書 式

CHR_C

■機 能

受信文字数設定

■解 説

受信文字数を設定します。

```
10      CNFG#   1 "9600b8pns1NONE"
20      INPUT#   1 CHR_C|1 a$        /* receive 1 character
30      PRINT   a$
#RUN

 a                                   /* send 'a' from the terminal soft
```

CK_Z,CK_NZ

演算　　関数

■書 式
CK_Z(arg)
CK_NZ(arg)

■使い方
IF SW(1)&SW(2)|CK_Z(A) THEN : PRINT "OK" : END_IF

■機 能
ゼロテスト、ノンゼロテスト

■解 説
CK_Z(arg) 引数の値が 0 であれば 1,0 でなければ、0 を返します。
CK_NZ(arg) 引数の値が 0 であれば 0,0 でなければ、1 を返します。

CLRPOS

パルス発生　　コマンド

■書 式
CLRPOS [AXIS],[-1]

■使い方
CLRPOS
CLRPOS X_A
CLRPOS -1
CLRPOS X_A -1

■機 能
位置カウンタ、エンコーダカウンタのクリア

■解 説
引数がなければ現在位置をすべて 0 にします。
軸指定定数があれば、対象軸を 0 にします。-1 が与えられると、エンコーダ・カウンタを 0 にします。
CLRPOS X_A -1 の場合は、X 軸エンコーダ・カウンタをクリアします。

CLR_BUF

通信　　予約定数

■書 式
CLR_BUF

■使い方
INPUT# 1 CLR_BUF

■機 能
RS-232 入力バッファクリア

■解 説
INPUT# のオプションです。入力バッファをクリアします。
バッファのクリアには CNFG# を用いないでください。
INPUT# 参照

CLR_ER

パルス発生　　予約定数

■書 式
CLR_ER

■使い方
PGE(CLR_ER)
PGE(Z_A,CLR_ER|LMTp)

■機 能
エラーステータスクリア

■解 説
PGE() 参照

CLR_OUTP

IO　　コマンド

■書 式
CLR_OUT arg

■使い方
CLR_OUTP 1|8
CLR_OUTP 15

■機 能
I/O エリアの初期化

■解 説
CLR_OUTP [n]
n=1: 実出力ポート
2: CUNET
4: MBK
8: Memory IO
ビットパラメータのため必要な初期化エリアに対応するビットを ON して実行。
CLR_OUTP 15 はすべて初期化となる。

CMP_C

パルス発生　　関数

■書 式
CMP_C(axis)
CMP_C(port,axis)

■使い方
WAIT CMP_C(X_A)==2
A=CMP_C(16,X_C)

■機 能
カウンタと COMP+/- の比較結果を参照する。
カウンタと COMP+/- の比較結果が変化したら、指定ポートを ON する。

■解 説
MPG-2314 には、COMP+,COMP- レジスタがあり、カウンタと COMP レジスタの比較をリアルタイムで行うことができます。比較結果は、CMP_C 関数で参照します。

CMP_C() = [BIT0 <= CMP+ ,BIT1 <= CMP-]

CMP+
1: カウンタ値 >= COMP+ レジスタ
0: カウンタ値 < COMP+ レジスタ
CMP-
1: カウンタ値 < COMP- レジスタ
0: カウンタ値 >= COMP- レジスタ

又、比較カウンタ値には、現在位置カウンタ、エンコーダカウンタの何れかを選ぶ事ができます。INSET X_A CMP_PLS と設定すると、パルス位置と COMP レジスタの比較結果を CMP_C(X_A) で知る事ができます。INSET X_A CMP_CNT の場合は、エンコーダカウンタとの比較になります。

COMP レジスタは RANGE コマンドで設定する事ができます。
RANGE X_A COMP+ COMP-
なお、サンプルプログラムのように、CMP_C(port,X_A) 記述すると、比較フラグの変化を待って指定ボートを ON して抜け出します。
この場合、CMP+,CMP- のいずれのビットが変化しても変化検出となります。

```
40      ACCEL   30000
50      CLRPOS
60      INSET   CMP_PLS
65     P_DET=500
70      RANGE   X_A P_DET P_DET
80      RMVC   X_A 1
100     DO
110     A=CMP_C(16,X_A)
120      INC   P_DET 500
130      OFF   16
135      RANGE   X_A P_DET P_DET
140     LOOP
```

CMP_CNT

パルス発生　　予約定数

■書 式
CMP_CNT

■機 能
カウンタ比較

■解 説
対象ボード : MPG-2314
エンコーダカウンタと COMP+ と比較
INTA_ON も参照してください。

```
INSET X_A CMP_CNT|PHASE1
```

CMP_P

パルス発生　　関数

■書 式
CMP_P([axs,],v)

■使い方
CMP_P(n)
CMP_P(axs,n)

■機 能
現在位置と点データの比較

■解 説

現在位置と指定された点データを比較します。軸指定がなければ、XYZU の全軸の値を比較し、同じであれば 1、1 軸でも相違していれば、0 を返します。
軸指定を与えると、指定した軸のみ比較します。

```
10       PG   0
15       CLRPOS
20       ACCEL   8000
30       SETP   7000 10000 20000 30000 40000
40       MOVS   P(7000)
50       DO
60        IF   CMP_P(7000) THEN  : PRINT   "Arrived" : BREAK   : END_IF
70        TIME   100
80       LOOP
90       RMVS   Z_A 100
100       WAIT   RR(Z_A)==0
110       PRINT   CMP_P(7000)
120       PRINT   CMP_P(VOID_Z,7000)
#run
 Arrived
 0
 1
#
```

CMP_PLS

パルス発生　　予約定数

■書 式

CMP_PLS

■機 能

カウンタ比較

■解 説

対象ボード : MPG-2314
現在パルスカウンタと COMP+ と比較
INTA_ON も参照してください。

```
INSET X_A CMP_PLS
```

CNFG#

通信　　コマンド

■書 式

CNFG# COMn [RS485] "setting"

■使い方

CNFG# 1 "38400b8pns1NONE"
CNFG# 5 RS485 "38400b8pns1NONE"

■機 能

RS-232C ポートの初期化

■解 説

COMn は初期化する RS-CH 番号です。文字列はボーレート、キャラクタフォーマット指定です。
COMn = 1 MPC-USER ch1
COMn = 2 MPC-USER ch2
COMn = 3 MRS CH3
COMn = 4 MRS CH4

COMn = 5 MRS CH5
COMn = 6 MRS CH6
COMn = 7 MRS CH7
COMn = 8 MRS CH8

ボーレート 4800,9600,19200,38400 のいずれかを選択します。
b8 8 ビットキャラ
b7 7 ビットキャラ
pn パリティなし
pe 偶数パリティ
po 奇数パリティ
s1 1stop bit
s2 2 stop bit
NONE XON/XOFF 制御なし (XON/XOFF 制御未対応)
引数に RS485 を追加すると、RS422/485 兼用ポートで RS485 通信が可能になります。

プログラム実行中にバッファクリア等の目的で CNFG# を実行すると、字落ちの原因となります。CNFG# は初期設定で一回のみ使用として、バッファクリアには INPUT# と CLR_BUF 定数を併用してください。
参照 INPUT#、PRINT#

COMPOWAY

文字列 コマンド

■書 式
COMPOWAY n m l str1$ str2$
COMPOWAY str1$ v1 v2 v3 str2$

■使い方
COMPOWAY 1 2 0 cmnd$ buff$
COMPOWAY buff$ nod adr id rcv$

■機 能
OMRON COMPOWAY フォーマットの文字列生成および分解

■解 説
OMRON COMPOWAY は以下のようなコマンドフォーマットとなっています。
送信時 : ADR+SADR+ID+CMND 文字列
受信時 : ADR+SADR+END+RES 文字列
CMPOWAY コマンドは双方の文字列を効率よく生成、解析します。

生成 : 指定した、アドレス、サブアドレス、コマンド文字列 (cmnd$) を buff$ に収納します。
COMPOWAY 1 2 0 cmnd$ buff$

分解 : うけとった文字列 buff$ を変数 nod adr id にレスポンス文字列を res$ に収納します。
COMPOWAY buff$ nod adr id res$

nod,adr,id は COMPOWAY の定型フォーマットに含まれる数値データです。
res$ は、コマンドによって応答が異なりますので、適宜、接続機器の仕様に基づいて、読み出し判定を行います。なお、BCC エラーは input# COMPOWAY コマンド実行後 rse_ に反映されます。(0 で正常、4 は BCC エラーです)

```
*RS-485_SEND_READ
 _VAR data_len
 cmnd_txt$=mrc_src$+hensu_shu$+str_adr$+bit_ichi$+yoso_su$+setteichi$
 COMPOWAY node_no sub_adr sid cmnd_txt$ snd$
 PRINT# 5 COMPOWAY snd$
 INPUT# 5 COMPOWAY rcv$
```

```
  IF rse_!= THEN
 /* WHEN rse_ is 4 , BCC error happend , OTHER cases indicates RS-232c errors
    PRINT "communication error"
  END_IF
  COMPOWAY rcv$ node_no sub_adr end_code res$
  ptr_=res$+4
  res_code=HEX(PTR$(4))
  ptr_=res$+8
  res_data$=PTR$(data_len)
  RETURN
```

CONST

演算　　コマンド

■書 式
CONST var val

■使い方
CONST A_P 123

■機 能
変数の定数化

■解 説
変数を定数化して変更できないようにします。

CONT

マルチタスク　　コマンド

■書 式
CONT arg

■使い方
CONT 8
CONT VOID|1

■機 能
SLEEPING 中のタスクを再開

■解 説
PAUSE コマンドによって一時停止しているタスクを再開します。
PAUSE (STP_D,n) によって一時停止させられた、WARP,JUMP,JMPZ は、CONT によって、自動再実行されます。
ただ、状況によっては、不都合な場合もあります。この場合は、CONT VOID|n と実行します。
これにより一時停止された、WARP,JUMP,JMPZ の実行はキャンセルされます。

COS

浮動小数点　　コマンド

■書 式
cos deg r var [sf]

■使い方
cos 450000 100000 a
cos 4500000 100000 a 100000

■機 能
COS 演算

■解 説

浮動小数点 COS 演算を行います。
var = r × cos(deg/sf)
　注) sf を省略すると、sf を 10000 とします。
1) COS 600000 10000 a
#pr a
5000
#
cos(600000/10000) の演算で cos(60 度) の意味です。結果は 0.5 ですが、10000 × 0.5 のため、5000 となります。

```
-----------------------------------------
#cos 450000 100000 a
#pr  a
 70711
#
#cos 4500000 100000 a 100000
#pr  a
 70711
#
```

COUNTER_1

IO　　予約変数

■書 式

COUNTER_1

■使い方

COUNTER_1=0

■機 能

カウントアップ

■解 説

<u>カウント機能</u>
それぞれの入力ポートへの 1msec パルス幅以上に対して、カウントします。
*MPC-1200/MPC-1000/MPC-N816
SW(198) --> COUNTER1
SW(199) --> COUNTER2
*MPC-2000 では、
SW(206) --> COUNTER_1
SW(207) --> COUNTER_2
カウント開始は初期値を設定した時点となります。それまでの初期値は VOID でカウントは無効になっています。

<u>オートリセット機能</u>
予約変数 AUTO_RESET_1 に値を設定すると、その値になると COUNTER_1 は 0 クリアします。
予約変数 AUTO_RESET_2 に値を設定すると、その値になると COUNTER_2 は 0 クリアします。

<u>アップダウンカウント</u>
AUTO_RESET_1 を 0 にすると、COUNTER_1 がアップダウンになり、COUNTER_2 は無効となります。

COUNTER_2

IO　　予約変数

■書 式
　COUNTER_2

■使い方
　COUNTER_2=0

■機 能
　カウントアップ

■解 説
　参照　COUNTER_1

CP

パルス発生　　コマンド

■書 式
　CP

■機 能
　現在位置表示

■解 説
　MPG ボードの座標管理での現在位置を表示します。

CSW

IO　　関数

■書 式
　CSW(arg)

■使い方
　A=CSW(0)
　IF A==1 THEN : GOSUB *A :ELSE : GOSUB *B

■機 能
　指定入力ポートが変化するまで待ち、変化後の値を返す。

■解 説
　CSW(n) は関数自身が待ち (ポーリング) を含んでいます。
　入力状態が変化するまで 50 で待ち続けます。

```
10          FORK  1 *task1
20          END
30         *task1
40          DO
50         A=CSW(-1)
60           PRINT  A
70          LOOP
#run

#on -1
#off -1
#0
on -1
#1
off -1
#0
```

CTRL_A

保守　　コマンド

■書 式
　CTRL_A [val]

■使い方
　CTRL_A 1
　CTRL_A 0

■機 能
　CTRL_A 機能設定

■解 説
　プログラムポートで SOH(CTRL_A) を受信したあとのプログラム起動の可否を指定します。
　CTRL_A 0 ： J1-5,6 状態がオープンであれば、プログラムを再起動します。(標準状態)
　CTRL_A 1 : J1-5,6 の状態にかかわらず、プログラムの再起動は行いません。

CUNET

CUnet　　コマンド

■書 式
　CUNET arg1 arg2 arg3

■使い方
　CUNET 0 8 31
　CUNET 8 8 15

■機 能
　CUnet の初期化

■解 説
　CUNET sa own end
　sa は領域確保の開始ブロック番号 0 ～ 63
　own は領域ブロック数。 1 ～ 32
　end は全体で何ブロック共有するかという数 2 ～ 63 （ブロックは SA0 ～ SA63 と表現されます)
　CUnet は 8byte ごとのブロックが 64 個あります。(512byte)
　そのブロックの確保領域を定めて CUnet ボードを初期化します。
　初期化以後、CUnet のメモリエリアは 2000 番以上の IO アドレスとなります。

　例えば、CUNET 0 1 32
　とすると、SA0 のみを確保します。これによりその MPC では、
　OUT n SA0_B+m (m=0 ～ 7) でかきこめます。
　ON/OFF は、ON SA0+m (m=0 ～ 63) で ON/OFF できます。

　他のステーションでは IN/SW のみ有効で、同じ番号で読み取ります。
　SA0,SA0_B はそれぞれ予約定数で、ブロックごとに用意されています。
　CUNET 8 8 32 とすれば、ON/OFF は SA8 より 8*8*8=512 ビット制御します。
　OUT では、SA8_B より 8*8=64byte 制御できます。

　【通信レートの設定】
　CUnet は通常 12Mbps で通信を行いますが、必要に応じて、6M,3Mbps を選択することもできます。この場合は、引数 SA に &H80,&H40 を論理加算します。
　CUNET &H80|SA・・・・　　6Mbps に設定
　CUNET &H40|SA・・・・　　3Mbps に設定

```
'dsiplay io
 CUNET   0 8 31
```

```
 DO
  OUT   IN(SA8_B) 2
  OUT   IN(SA8_B+1) 3
 LOOP

'scan IO
 CUNET   8 8 31
 DO
  FOR   i=0 TO 15
   ON   SA8+i
   WAIT   SW(SA8+i)
   TIME   5
   OFF   SA8+i
   WAIT   SW(SA8+i)==0
  NEXT   i
  OUT   0 SA_B8 : OUT   0 SA_B8+1
 LOOP
```

CU_POST

CUnet　　コマンド

■書 式

CU_POST [n]|[VOID]

■使い方

CU_POST
CU_POST 28

■機 能

CUnet メールサーバ

■解 説

CUnet メールサーバー起動コマンドです。
自動的に送られてきた CUnet メールを読み取り、転送命令にしたがってデータを P(n),MBK(n) に格納します。また、要求にもとづいて (POST -n XXX コマンド) 自己データを転送します。

CU_POST コマンドは引数無しで実行すると、自動的に空きタスクを探しメールサーバを起動します。引き当てられたタスク番号は、グローバル変数 CUM_TASK に反映します。
また、引数を 1 ～ 31 の間で与えると、その番号のタスクでメールサーバを起動します。
引数に VOID を与えるか、指定タスク番号と VOID を OR して与えると、実行状況を表示します。
なお、メールサーバは CTRL_A によって停止します。

メールサーバの動作状態は以下のグローバル変数で監視することができます。
CUM_ERR(エラーに) ついては OR 更新、CUM_CNT(メールカウンタ) はインクリメント、その他は受信ごとに最新の値に更新されます。

CUM_TASK CU_POST サーバの使用タスク番号
CUM_SRC 受信メールの相手アドレス
CUM_PNT 受信メールの種類 1: P(n) 2: MBK(n)
CUM_NUM 受信メールの P(n) もしくは MBK(n) の n の値
CUM_CNT 受信メールごとにインクリメント
CUM_ERR エラー各 bit は以下のとおり。
BIT7: MAIL SEND ERROR
BIT6: 転送要求の応答が無い
BIT5: 通信停止
BIT4: 送信タイムアウト不正 (通常 0)
BIT3: 送信ブロック不正 (通常 0)
BIT2: 送信タイムアウト発生

BIT1: 送信相手不在
BIT0: 送信相手が受信待機となっていない

このコマンドを使用することにより、PC から MPC 側の点データや MBK データを書き換えたり、参照することができます。これについては、USB-CUnet の資料を参照ください。

【参考資料】
メールの転送単位は、以下のとおりです。
P(n) 15 個 Long 型 *4
MBK(n) 120 個 Word 型

メールのパケットは 256byte で以下のような構成で最初の 16byte をシステムエリアとして以下のように使い分けています。

256buffer= {Num(word)][Ary(byte)][Cmd(byte)][12byte reserved]P(n) 〜 P(n+14]}

Num は、P(n)、MBK(n)、IN(n) の最初の場所 n を示します。

Ary は P(n),Mbk(n),IN(n) の指定で、1 で P(n),2 で Mbk(n)、3 は IN(n) となります。
ただし、IN(n) の場合は 1byte ずつの扱いとなります。
33 を指定するとバルク転送となり、1 回の通信ですべての実 I/O 情報を得ることができます。
(これは USB-CUnet でのみ有効)
IN(n) では n に負の値を指定するとメモリ I/O 領域を参照します。

Cmd は 引渡しか請求かの区別で、1 で請求。2 で引渡しです。
*Ary に 33,Cmd に 2 を指定すると、I/O の一括設定となりますので運用には注意が必要となります。

buffer = {
00 64 01 02 00 00 00 00 00 00 00 00 00 00 00 00
00 00 03 E8 00 00 03 E8 00 00 03 E8 00 00 03 E8
00 00 27 10 00 00 27 10 00 00 27 10 00 00 03 E8
} 合計 256byte

Cmd==2 の場合
CU_POST は、送られてきたメールの Ary,Num の値にしたがって自分のエリアにデータを書き込みます。
Cmd==1 の場合
CU_POST は、送られてきたメールの Ary,Num の値にしたがって自分のエリアのデータを要求元に送り返します。

【サンプルプログラムについて】
SA2,SA4 をそれぞれの MPC-2000 システムにロードして実行します。
SA2 側で、*xf を実行すると、SA2 の点データが SA4 に複写されます。(5000 点で約 30 秒)
SA2 側で、*rcv を実行すると、SA4 の点データが SA2 に複写されます。。(5000 点で約 30 秒)
CU_POST の引数を無くせば実行表示はなくなります。

```
==SA2==
10 CUNET 2 2 32
20 TIME 5
60 CU_POST VOID|25
65 PRINT CUM_TASK
70 END
80 *xf
90 FOR i=1 TO 5000
100 SETP i i i i i
110 NEXT
120 FOR i=1 TO 5000 STEP 15
130 POST 4 P(i)
```

```
140 NEXT
150 END
160 *rcv
170 FOR i=1 TO 5000 STEP 15
180 POST -4 P(i)
190 PRINT i
200 NEXT
#
==SA4==
10 CUNET 4 2 32
20 TIME 5
60 CU_POST
8-31
65 PRINT CUM_TASK
70 END
80 *xf
90 FOR i=1 TO 5000
100 SETP i i i i i
110 NEXT
120 FOR i=1 TO 5000 STEP 15
130 CUM_ERR=0
140 POST 2 P(i)
150 IF CUM_ERR!=0 THEN : PRINT "X_ERR" : END : END_IF
160 NEXT
170 END
180 *rcv
190 FOR i=1 TO 5000 STEP 15
200 POST -2 P(i)
210 PRINT i
220 NEXT
#
```

CW

パルス発生　　予約定数

■書 式

CW

■機 能

原点復帰サーチ方向指定
円弧補間指定

■解 説

対象ボード : MPG-2314
SHOM では原点復帰の Z 相サーチ方向を指定します。
MOVT では円弧補間の回転方向を指定します。

```
SHOM X_A|Y_A IN0_ON|CW          /* set HOME condition. CW movement until the sensor turns
on
MOVT X_A|Y_A P(102) CW          /* continuous interpolation. CW revolution.
RMVC X_A CW                     /* infinite pulse generation. CW movement.
```

C_LESS

パルス発生　　予約定数

■書 式

C_LESS

■機 能

カウンタ比較

■解 説
対象ボード : MPG-2314
カウンタ＜ COMP+ で割り込み
INTA_ON, INTB_ON 参照

C_MORE

パルス発生　　予約定数

■書 式
C_MORE

■機 能
カウンタ比較

■解 説
対象ボード : MPG-2314
カウンタ >=COMP+ で割り込み
INTA_ON, INTB_ON 参照

DA

AD_DA　　コマンド

■書 式
DA val [ch]

■使い方
DA 1000 1
DA 2000

■機 能
DA 出力

■解 説
MPC-AD12 の DA 出力を設定します。0 ～ 4095 の値を指定します。標準設定で 1mV/1digit。
設定された値は 2mSEC 以内に DA 出力に反映されます。
MPC-AD12 の DA 出力は 4CH あり、0 ～ 3 までを指定できます。
さらに 1 枚 MPC-AD12 を追加した場合は、DA 出力 CH 番号として 4 ～ 7 が割り当てられます。
2 番目の引数で CH を指定しますが、省略すると CH0 となります。

DATE

時間管理　　関数

■書 式
DATE(0)
DATE(255)
DATE(VOID)

■使い方
```
IF   DATE(0)==&H20070731 THEN
  PRINT "HAPPY BIRTHDAY"
END_IF
```

■機 能
年月日取得

■解 説

ヘキサ形式で日付値を得ます。引数をいれると、引数と論理積をとって値を返します。
引数にVOIDを設定すると、10進で値を返します。尚、年月日設定は SET_RTC コマンドで実施します。

```
IF   DATE(0)==&H20070731   THEN
     GOTO  *Thisday
END_IF
     PRX   DATE(0)
```

DATE$

文字列　関数

■書 式

DATE$(n)

■使い方

a$=DATE$(1)+" "+TIME$(1)

■機 能

日付文字列取得

■解 説

日付文字列を得ます。
DATE$(0)-> 20090529
DATE$(1)-> 5/29/2009
DATE$(2)-> 5.29.2009
DATE$(3)-> 2009-05-29

```
a$=DATE$(1)+" "+TIME$(1)+": CNT="+STR$(i)
```

DEG

浮動小数点　関数

■書 式

DEG(v)

■使い方

FLOAT A=DEG(ATAN(SQR(2))

■機 能

度変換

■解 説

ラジアンを度に変換します。

```
#
 FLOAT A=DEG(ATAN(1))
#pr A
 45
#
```

DELETE

編集　コマンド

■書 式

DELETE arg1 [arg2]

■使い方
DELETE n
DELETE n m

■機 能
指定行の抹消

■解 説
行あるいは範囲を指定してプログラムを抹消します。

DIM

演算 コマンド

■書 式
DIM label(val)
DIM label(val1,val2)
DIM label1(val) label2(val) label3(val) ...

■使い方
DIM A(100)
DIM array(100,100)
DIM A(100) B(100) C(5)

■機 能
配列要素の宣言

■解 説
配列は 1 次元、 2 次元のいずれかで、合計 20000 データの範囲で自由に宣言できます。
ラベルは 1 5 文字以内で 64 個までです。一度、配列が 64 個を超えるとその後のラベル管理ができなくなりますので、プログラム修正後、再ロードしてください。

DIMCPY

演算 コマンド

■書 式
DIMCPY arg1 arg2 count

■使い方
DIMCPY 1000 U(3) 60
DIMCPY X(1) aho(10) 10
DIMCPY MBK(1) Y(4) 50
DIMCPY X(3) MBK(200) 60
DIMCPY X(3) MBK(200~Lng) 60

■機 能
MBK() X(),Y(),Z(),U() および定義した配列要素間でデータの転写。

■解 説
MBK() X(),Y(),Z(),U() および DIM 定義した配列要素間でデータの転写を実施。
DIMCPY MBK(1) MBK(100) 50 のように同一の要素でもエリアが重複していなければ転写可能です。なお、DIMCPY で MBK データは、ワード型としてのみ扱います。
MBK データをロング型とする場合は、~Lng を付加します。
ただし、DIMCPY MBK(1) MBK(100~Lng) 50 のような場合は、元も先もロング型扱いとします。

```
DIM   aho(300)
  DIM   baka(300)
  DIMCPY   1010 aho(3) 25
  FOR   i=1 TO 30
```

```
 PRINT  i aho(i)
 NEXT
 S_MBK  100 50 30
 DIMCPY  12345 MBK(52) 10
PR "MBK"
 FOR  i=50 TO 65
  PRINT  i MBK(i)
 NEXT
*test2
 FOR  i=1 TO 50
aho(i)=i*-1000
 NEXT  i
 DIMCPY  aho(1) baka(100) 30
PR "BAKA()"
 FOR  i=90 TO 135
  PRINT  i baka(i)
 NEXT
NEWP
 DIMCPY  baka(100) X(10) 20
 DIMCPY  baka(100) Y(11) 20
 DIMCPY  baka(100) Z(12) 20
 DIMCPY  baka(100) U(13) 20
PR "X()"
 FOR  i=5 TO 30
  PRINT  i P(i)
 NEXT
 DIMCPY  X(10) MBK(52) 20
PR "X->MBK"
 FOR  i=50 TO 65
  PRINT  i MBK(i)
 NEXT
 DIMCPY  MBK(52) baka(70) 20
PR "X->MBK"
 FOR  i=65 TO 95
  PRINT  i baka(i)
 NEXT
```

DIR

USB　　コマンド

■書 式

DIR [USBn] [n]

■使い方

DIR
DIR 100

■機 能

USB メモリのファイルリスト取得

■解 説

DIR とすると、USB メモリのディレクトリを表示します。
引数に番号を与えると、表示はしないで、以下のタイプのファイル名を MBK エリアに 8 文字 +.??? フォーマットでファイル名を複写します。(12 文字なので 6 データごと)

**.P??
**.C??
**.T??
**.F??

MBK$(n+4,12) ファイル名 1
MBK$(n+16,2) ファイル名 2

デフォルトでファイル名 50 個まで、MBK エリアに保存されます。
それ以上必要な場合、あるいはそれ以下にしたい場合は、三番目の引数で、MBK エリアの上限値を指定します。そして、以下の場所には取得データを保存します。

MBK(n)--> ファイル数
MBK(n+1)--> トータルファイル数
MBK(n+2)--> トータルディレクトリ数
MBK(n+3)-->USB 使用容量 (Mbytes) (ただし ルートディレクトリの分のみ)

注) USB メモリの残容量の直接取得は、実際の空きブロックのカウントアップ処理が必要となり、2G 以上の USB メモリでは相当時間必要です。8G の場合で 1 分程度となるため実用的ではありませんでした。
代替方法として、ルートにあるファイルの総バイト数が判明しますので、総容量から消費バイト数を算出します。総容量は、USB(1,0) などの関数によって DIR 命令実行後知ることができます。単位は Mbyte です。

DO-LOOP

制御文　　ステートメント

■書 式
DO
LOOP

■使い方
DO
ON 0 : TIME 1 : OFF 0
LOOP

■機 能
エンドレス繰り返し実行

■解 説
DO から LOOP の間をエンドレス繰り返し。繰り返しを停止させる場合は、BREAK 文を用いる。

```
DO
IF SW(0)==1 THEN : BREAK : END_IF
LOOP
```

DS_DACL

パルス発生　　コマンド

■書 式
DS_DACL [axs]

■使い方
DS_DACL
DS_DACL X_A

■機 能
減速無効設定

■解 説
自動減速無効とする。連続補間で使用。

DS_SEC

時間管理　　コマンド

■書 式
DS_SEC n

■使い方
DS_SEC 5

■機 能
1秒カウンタ停止

■解 説
n で指定した 1 秒カウンタ SEC(n) を、停止します。

EMG

パルス発生　　予約定数

■書 式
EMG

■機 能
エラービット指定

■解 説
対象ボード : MPG-2314
緊急停止信号 (EMGN) ビット

```
IF LMT(X_A,EMG)!=0 THEN              /* confirming reason for stop
```

END

制御文　　ステートメント

■書 式
END

■使い方
END

■機 能
実行終了

■解 説
プログラム終了。マルチタスク・プログラムの終了。
タスク0の場合は入力待ちプロンプトを出力。マルチタスク・プログラムの場合は、タスク停止となる。

ENG

保守　　コマンド

■書 式
ENG

■機 能
英語モードにする。

■解 説
MPCINIT 後は英語モードとなっている。エラー表示は英文となります。

EN_DACL

パルス発生　　コマンド

■書 式
EN_DACL [axs]

■使い方
EN_DACL
EN_DACL X_A

■機 能
減速有効設定

■解 説
自動減速有効とする。

EN_SEC

時間管理　　コマンド

■書 式
EN_SEC n

■使い方
EN_SEC 1

■機 能
1 秒カウンタのカウント・イネーブル

■解 説
n で指定した 1 秒カウンタ SEC(n)、をカウント・モードにします。

EOL

通信　　予約定数

■書 式
EOL

■機 能
受信ターミネータ設定

■解 説
受信ターミネータを設定します。

```
10      CNFG#   1 "9600b8pns1NONE"
20      INPUT#   1 EOL|10 a$            /* until receive LF(&HA=10)
30      PRINT   a$
RUN
 hello                                  /* send 'hello' from the terminal soft
```

ERASE

編集　　コマンド

■書 式
ERASE

■機 能
FLASH ROM の消去

■解 説
FLASH ROM の消去。システム入れ替え時は、MPCINIT 後 ERASE してください。

ERR$

保守 関数

■書 式
ERR$(n)

■使い方
pr ERR$(err_)

■機 能
エラーコードに対応するメッセージの出力

■解 説
ON_ERROR でエラー割り込みを設定すると、err_ 変数にエラーコードと対応する文番号が格納されます。上位 1byte がエラー・コード、下位 3byte が文番号です。
err$() はこの上位 1byte のコードにしたがってエラー文字列を返します。
従って、手操作でエラー・メッセージを参照する場合は、
print err$(1>>24)
というように値を 24bite 上位にシフトさせます。

FEED

パルス発生 コマンド

■書 式
FEED [axis] n
FEED fx fy fu fz

■使い方
FEED 10
FEED X_A 100

■機 能
速度設定

■解 説
ACCEL で設定された最高速、最低速をもとに 100 段階に速度を設定します。
数値は最高速の % を整数値で設定します。
FEED X_A 100 で、X 軸最高速
FEED X_A 0 で X 軸最低速度
途中の数値は、Fn=MIN + n*((MAX-MIN)/100)

軸パラメータがない場合は、X,Y,U,Z の順序で指定パラメータとします。
従って、FEED 100 は X 軸に対してのみ速度を指定したことになります。

FILL

演算 コマンド

■書 式
FILL array(N) Count [Val Inc]

■使い方
FILL aho(0) 0 0
FILL aho(10) 10 -110 2

```
FILL  X(6) 20 10000 100
FILL  MBK(100) 10 500 -2
FILL  MBK(200~Lng) 100000 10000
```

■機 能

配列要素にデータを連続設定する。点データ、MBK データにも有効

■解 説

配列要素の初期化コマンドです。
第 1 引数は初期化したい配列の先頭要素を記述します。
第 2 引数は、カウント数です。いくつ、初期化するかを指定します。
ただし、0 を指定すると、指定配列すべてとなります。

例えば、
FILL AHO(0) 0
とすると AHO(0) ～配列範囲内で内容をすべて 0 に設定します。

FILL AHO(5) 10
この場合は、AHO(5) ～ AHO(14) を 0 にするという意味になります。

第 3 引数をいれると、0 以外の数を設定できます。
FILL AHO(5) 10 100
AHO(5) ～ AHO(14) を 100 に設定するという意味になります。
さらに、第 4 引数をいれると設定値をオートインクリメントします。

FILL SYSDAT(1) 100 501~Lng 2

この例では、SYSDAT(1) ～ SYSDAT(100) に
501~Lng
503~Lng
と 2 を加えながら、設定していきます。負の数を設定すれば、減算しながら設定します。

配列要素は、バッテリバックアップされていますが、プログラムの変更などで、メモリ位置がかわります。このため、常に適切な初期化が必要となります。

配列要素に点データ P(n) を与えると、X,Y,Z,U のすべての要素を同じ値にします。
FILL P(1) 100 => P(1) ～ P(100) を 0,0,0,0 にします。

```
DIM   aho(100)
  FILL   aho(0) 0 0
  FILL   aho(10) 10 -110 2
  FOR   i=8 TO 30
   PRINT   i aho(i)
  NEXT
  FILL   X(6) 20 10000 100
  FILL   MBK(100) 10 500 -2
  FILL   MBK(200~Lng) 100000 10000
```

FLIP_FLOP

IO　　コマンド

■書 式

FLIP_FLOP o_port IN(port) [pat]

■使い方

FLIP_FLOP -1 IN(24)
FLIP_FLOP -1 IN(24) &H0F

■機 能
セットリセットプリップフロップ

■解 説
セットリセットタイプのフリップ・フロップです。8 ビットのバンク単位の I/O で設定できます。
実行内容としては、
o_port |= IN(n) xor pat
となります。pat が省かれていると pat は 0 になります。
このため、入力ポートがアクティブになると、対応する出力ビットがセットされ保持されます。
クリアするには、OFF bit_port します。セットに必要な時間は 1msec です。
ネガティプ論理が必要な場合は、pat の対応ビットを 1 にします。

```
10      CLR_OUTP 15
20      FLIP_FLOP   -1 IN(24)
30      DO
40       PRX   IN(24) IN(-1)
50       TIME   500
60      LOOP
#RUN

00000000 00000000    ← SW(194)=0,SW(193)=0,SW(192)=0
00000001 00000001    ← SW(194)=0,SW(193)=0,SW(192)=1
00000000 00000001
00000002 00000003    ← SW(194)=0,SW(193)=1,SW(192)=0
00000000 00000003
00000004 00000007    ← SW(194)=1,SW(193)=0,SW(192)=0
00000000 00000007
```

FLOAT

浮動小数点　　コマンド

■書 式
FLOAT equation1 equation2 ...

■使い方
FLOAT A=1/3*10000
FLOAT FP(1)=SIN(RAD(30))

■機 能
浮動小数点演算

■解 説
FLOAT コマンド中の演算は、倍精度浮動小数点演算となります。
FLOAT コマンド中整数変数への代入は、演算は倍精度でおこなわれ、代入時に整数化されます。
FLOAT コマンド FP(n) への代入は、演算を倍精度でおこない、倍精度として代入します。
FLOAT コマンド中、SIN,COS,TAN,ATAN,ACOS,SQR、RAD,DEG,VAL 等の関数は、倍精度関数として使用されます。

```
' Get Pie
  FLOAT   FP(6)=ACOS(SQR(3)/2)*6
  FLOAT   FP(6)=(FP(6)-3)*10
  PRINT   "PIE=3." FP(10000,6)
' Get Napier
 a=1
  FLOAT   FP(2)=1
  FOR   i=1 TO 100
 a=a*i
   FLOAT   FP(2)=FP(2)+1/a
  NEXT
  FLOAT   FP(2)=(FP(2)-2)*10
```

```
 PRINT   "Napier=2." FP(10000,2)
'
 PRINT   "Second order equation X*X+4*X+3"
a=1 :  b=4 :  c=3
 FLOAT   FP(0)=(SQR(b*b-(4*a*c))-b)/2/a
 FLOAT   FP(1)=(SQR(b*b-(4*a*c))*-1-b)/2/a
 PRINT   FP(10000,0) FP(10000,1)
'
```

FLP

パルス発生　　コマンド

■書 式

FLP

■使い方

FLP

■機 能

フラッシュ ROM 読込

■解 説

MPC-1000/MPC-N816 専用コマンド。
フラッシュ ROM から点データ P(100) ～ P(299) を読み込みます。
このエリアはパワーオンリセット時に自動的に読み込まれますが、FLP コマンドでも読み込みができます。フラッシュ ROM へ書き込むのは FSP コマンドです。

```
10      FOR   I=100 TO 299
20       SETP   I I I+1 I+2 I+3
30      NEXT   I
40      FSP
50      NEWP
60      PRINT   "P(100)=" P(100) "P(299)=" P(299)
70      FLP
80      PRINT   "P(100)=" P(100) "P(299)=" P(299)
#RUN

 P(100)= 0 0 0 0 P(299)= 0 0 0 0
 P(100)= 100 101 102 103 P(299)= 299 300 301 302
```

FOR-NEXT

制御文　　ステートメント

■書 式

FOR var=arg1 TO arg2 [STEP arg3]

■使い方

```
FOR i=0 TO 15 STEP 2
ON i : TIME 100 :OFF i
NEXT
```

■機 能

インクリメントあるいはデクリメント繰返処理

■解 説

決まった回数の繰り返し処理に使用する制御文です。NEXT には変数名をいれる必要はありませんが、変数名がはいっていると、FOR 文で指定された変数名と整合を確認します。

FORK

マルチタスク　　コマンド

■書 式

FORK n *LABEL

■使い方

```
FORK 1 *LABEL
END
*LABEL
DO
LOOP
```

■機 能

タスク起動

■解 説

マルチタスクで、*LABEL よりプログラムを実行します。指定できるタスク番号は 1 ～ 31 です。
起動されたタスクは END で終了するか、QUIT で強制終了できます。
すでに FORK しているタスクを FORK すると、二重起動エラーとなります。
この場合、QUIT してから再起動するか、QUIT_FORK を用います。
なお、タッチパネル通信や CU_POST コマンドはタスクを占有します。これらのタスクに干渉しないようにコマンドを用いてください。

FORMAT

文字列　　コマンド

■書 式

FORMAT Strng

■使い方

```
FORMAT   " DatB=[s00.000]"
FORMAT   "D=S00000"
```

■機 能

STR$() の展開様式を定義する。

■解 説

STR$() は、FORMAT コマンドで定義されていない限り、標準整数様式で文字列展開します。
STR$(1234) ->" 1234" STR$(-1234) ->"-1234"

FORMAT コマンドでは 15 文字の範囲で出力フォーマットを決定できます。

```
FORMAT   " DatA=[S 0.000]" --> DatA=[- 8.000]
FORMAT   " DatB=[s00.000]" --> DatB=[+02.000]
```

フォーマットで指定した文字列のスペースか 0 のところに右つめで数値がはいります。
S は符号で、ラージ S では、正の場合スペース、スモール s では、正の場合に + 符号が与えられます。S も s もつけないと、符号は付加されません。

```
FORMAT "0000 年 00 月 00 日 "          /* 文字列書式設定
DT$=HEX$(DATE(0))                      /* 年月日文字列取得
FORMAT "00 時 00 分 00 秒 "            /* 文字列書式設定
TM$=HEX$(TIME(0))                      /* 時分秒文字列取得
PR "(1)" DT$ TM$                       /* 表示

*RUN 結果
(1) 2007 年 11 月 07 日 12 時 34 分 00 秒
```

FP

浮動小数点　　予約変数

■書 式

FP(n)
FP(m,n)

■使い方

FP(0)=1000
STR(FP(100,1)

■機 能

浮動小数点変数配列

■解 説

FP(0) ～ FP(15) まであり、FLOAT コマンド中で倍精度浮動小数点変数として使用できます。
FLOAT FP(1)=1000
この場合、FP(1) には浮動小数点倍精度型として 1000 が保存されます。
また、VAL と組み合わせることによって、指数表現のデータを格納することができます。
a$="Mx+9.7042e+002 "
FP(2)=val(a$) とすれば、FP(2) は、9.704200E+02 としてデータが格納されます。
FP(n) を整数化して使用する場合は、FP(1,n) とします。
倍率が必要な場合には、1 のかわりに 1 ～ 10000 までの値をいれると、指定倍数ののち整数化されます。なお、内容の確認には、print 文が使用できます。

```
#a$="Mx+9.7042e+002 My+6.3210e+002"
#fp(2)=val(a$) fp(3)=val(0)
#pr fp(2) fp(3)
 9.704200E+02 6.321000E+02
#
```

FREE

編集　　コマンド

■書 式

FREE

■機 能

残容量の表示

■解 説

残容量をバイト数で表示します。

```
#free
176500
```

FSP

パルス発生　　コマンド

■書 式

FSP

■機 能

フラッシュ ROM 書き込み

■解 説

MPC-1000/MPC-N816 専用コマンド。
点データ P(100) ～ P(299) をフラッシュ ROM に書き込みます。

このエリアは Compiling 後のプログラムと共にフラッシュ ROM に書き込まれますが、プログラム中で強制的に書き込む場合は FSP コマンドを用います。

MPC-1000 はバッテリバックアップがありません。P(100) ～ P(299) は電源を切っても保持したいデータのエリア (ティーチングポイント、バックアップ変数 等) として使用します。
フラッシュ ROM から読み込むのは FLP コマンドです。

```
10      FOR   I=100 TO 299
20       SETP   I I I+1 I+2 I+3
30      NEXT   I
40      FSP
50      NEWP
60      PRINT   "P(100)=" P(100) "P(299)=" P(299)
70      FLP
80      PRINT   "P(100)=" P(100) "P(299)=" P(299)
#RUN

 P(100)= 0 0 0 0 P(299)= 0 0 0 0
 P(100)= 100 101 102 103 P(299)= 299 300 301 302
```

GETDG

浮動小数点　　コマンド

■書 式
GETDG　n m deg

■使い方
GETDG　1 3 deg

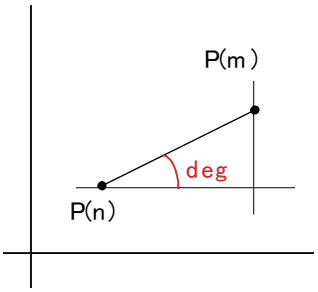
■機 能
角度計算

■解 説
ベクトル P(m)-P(n) の X 軸に対する角度を算出します。実際の計算は以下のように行われます。

deg = ATAN((Y(m)-Y(n))/(X(m)-X(n)))

角度 deg は 10000 倍された値で変数に返されます。サンプルプログラムでは、
X(2)-X(1)=> 17320508 Y(2)-Y(1)=1000000
従って、 ATAN(1000000/17320508) となり、30 度となりますが、
結果を 10000 倍するため、300000 が得られます。

```
#setp 1 10000 20000  0 0
#setp 2 17330508  10020000   0 0
#getdg 1 2 a
#pr a
 300000
```



GET_AD

AD_DA　　コマンド

■書 式
GET_AD　CH ARRAY() Cnt [ms]

■使い方
GET_AD　0 X(1090) 360 4
GET_AD　1 Z(1090) 100
GET_AD　0 X(100) (ch,100)

■機 能
AD データの連続取得

■解 説
CPU 側の 1msec タイマーを利用した MPC-AD12 からのリアルタイムデータ取得です。
唯一の 1msec タイマーで起動する機能のため、マルチタスク的な使い方はできません。

コマンドでは、取得チャネル、データの収納配列、獲得数、msec 単位での取得インターバルを指定します。インターバルを省略すると 1msec 間隔となります。
このコマンドは、データの取得完了を待たずに終了します。終了監視は、関数 GET_AD(0) によって行います。この関数が 1 を返している間は、データ取得中です。

サンプルは、MPC-1000 でステップモータを回しながら 4msec 間隔で、AD データを取得するものです。配列には点データ、MBK 配列、DIM 宣言配列を使用することができます。
この例では、データの更新確認を、X(1449) の値の更新によって行っています。

なお、データ配列に X() を指定し Cnt の上位ワードに 2 ～ 4 の値を設定すると、同時に複数チャンネルデータ取得します。

GET_AD 0 X(100) (3,100)
この場合、X(n) <- CH0,Y(n) <- CH1,Z(n) <- CH2 (n = 100 ～ 199) にデータ格納されます。

データ取得チャンネルも変更することができます。
GET_AD 2 X(100) (2,100)
この場合、X(n) <- CH2,Y(n) <- CH3 (n = 100 ～ 199) にデータ格納されます。
注) (3,100) は (3<<16)|100 と同じ値になります。

```
80      X(1449)=0
90       PGB   "P" 1800
95      SYSCLK=0
100      WAIT   SYSCLK=>360
110      GET_AD   0 X(1090) 360 4
120      WAIT   X(1449)!=0
125      PRINT   SYSCLK
130      WAIT   SW(RDY_PGB)==0
#
```

GET_VAL

文字列　　コマンド

■書 式
GET_VAL strg_val arry(n) [FPn]

■使い方
GET_VAL a$ a(1)
GET_VAL a$ a(1) 100

■機 能
文字列から数値を取り出し、配列に連続代入する。

■解 説
文字列に含まれる数値を配列に一括代入します。
引数は、文字列変数 (XX$) と配列変数 arry(n) に制限されています。(mbk,x(n) には代入できません。)
3 番目の引数を省略すると、小数点 (ドット) もデリミタとされます。
3 番目の引数に 10,100,1000 などの数値を設定すると、小数点を含む数字列を指定倍数の数値として配列に代入します。

```
10       DIM   a(10)
20      a$="1111.12   -2222.13   3333.1   4444.5 345m-9730"
```

```
25      PRINT   a$
30      FILL    a(1) 99 777
50      GET_VAL   a$ a(1)
60      PRA   a(1)
65      PRINT   "FP"
70      FILL    a(0) 99 777
80      GET_VAL   a$ a(1) 100
90      PRA   a(1)
#run

 1111.12   -2222.13   3333.1   4444.5 345m-9730
a(1)=1111
a(2)=12
a(3)=-2222
a(4)=13
a(5)=3333
a(6)=1
a(7)=4444
a(8)=5
a(9)=345
 FP
a(1)=111112
a(2)=-222213
a(3)=333310
a(4)=444450
a(5)=345
a(6)=-9730
a(7)=777
a(8)=777
a(9)=777
#
```

GOSUB,GOSUB_NE

制御文　　ステートメント

■書 式

GOSUB *Label [arg1,arg2..]

■使い方

GOSUB *Label
GOSUB *Label arg1 arg2 ..

■機 能

サブルーチン・コール

■解 説

サブルーチンは、スタック・メモリを用いているので、GOSUB で呼ばれたプログラムは必ず RETURN でもどる必要があります。
サブルーチン中から RETURN を用いず GOTO などで呼び出し元に戻すプログラムを作ってしまうと、スタック・オーバフローやアンダーフローというエラーになりプログラム実行が停止します。

BL/1 の GOSUB コマンドは、呼び出し先ラベルの後に、サブルーチンに引き渡す引数を追加することができます。この引数値は、サブルーチン側で、_VAR コマンドで取り出します。
_VAR の引数に、タスクローカル変数を用いるとサブルーチンは汎用的になります。

GOSUB の替わりに GOSUB_NE を用いると、呼び出し先ラベルが存在していなくてもコンパイル・エラーとならず、そのまま実行可能です。
飛び先ラベルの無い GOSUB_NE は、実行時には無視されます。

```
10 GOSUB *CAL 300 400
20 _VAR RES
30 PR RES
40 END
50 *CAL
60 _VAR V_ W_
70 RETURN SQR(SQ(W_)+SQ(V_))
RUN

*
Compiling
__________
500
#
```

GOTO

制御文　　ステートメント

■書 式

GOTO *Label

■使い方

```
IF A==1 THEN : GOTO *ERR : END_IF
GOTO *LOOP
```

■機 能

無条件分岐

■解 説

指定されたラベルに制御を移します。

HEX

文字列　　関数

■書 式

```
HEX(str)
HEX(arg)
```

■使 い 方

```
b$="ABC123 &H1234FJ  &HBCDEF1 "
  PRX   HEX(b$)
 SERCH   b$ "&H"
  PRX   HEX(5)
```

■機 能

ヘキサ文字列の読取。

■解 説

文字列中のヘキサ文字列を読み取る。
通常は SERCH などで場所を検索してから数値 (桁数) を指定して読み取る。
SERCH を使用したあとは、文字列を使用しないで数値をいれる。
a$="ABC123" のようにヘキサのみで成り立っている場合は、HEX(a$) でも読み取る。

```
LIST
10      b$="ABC123 &H1234FJ  &HBCDEF1 "
20       PRX   HEX(b$)
30       SERCH   b$ "ABC"
40       PRX   HEX(0)
```

```
50      SERCH   b$ "&H"
60      PRX   HEX(5)
#run

00ABC123
00000123
0001234F
#
```

HEX$

文字列　　関数

■書 式

HEX$(arg)

■使い方

a$=HEX$(100)
t$=HEX$(DATE(0))

■機 能

ヘキサデシマルで数値の文字列生成
t$=HEX$(DATE(0)) では年月日を得ることができます。

■解 説

FORMAT 指定がなければ、8 文字の HEX-DECIMAL 表現をします。
FORMAT 指定があればそれに従います。FORMAT の "S","s" は無効です。

```
10            FORMAT   ""
20            PRINT   HEX$(&H00ABCDEF)
40            FORMAT   "&H0000"
50            PRINT   HEX$(100000000)
#run
00ABCDEF
&HE100
#
```

HIN

IO　　関数

■書 式

HIN(arg)

■使い方

A=HIN(24)
A=HIN(24~Wrd)

■機 能

8 ビットパラレル入力 ~Lng,~Wrd 等の型指定をいれるとロング、ワードパラレルとして読み取れる。

■解 説

HIN() は、IN() と同じ機能関数ですが以下の相違があります。
IN() は IO エリアの入力ポートを 2 度読みしてチャタリングノイズを防いでいます。
これに対して HIN() は、1 回読みです。
メモリ I/O などの外部入力でない領域の読み取りは、HIN、IN とも 1 度読みです。

HOME[MPC-1200]

パルス発生　　コマンド

■書 式

HOME X Y U Z
HOME axs V

■使い方

HOME X_A NEG_L

■機 能

原点復帰コマンド

■解 説

MPC-1200 の入力ポート (SD,ORG) を使用した原点復帰コマンドです。SD 入力はスローダウン、ORG 入力は即停止です。
パワオン後は SD 無効です。SHOM で SD 有効化の場合、SD オン後 ORG が有効になります。

```
PG 17
ACCEL X_A|Y_A 20000 2000 500
 FEED  X_A|Y_A 100
 IF SW(200)==1 THEN  ' check ORGX
 RMVS X_A 5000
 END_IF
 IF SW(201)==1 THEN ' check ORGY
 RMVS Y_A 5000
 END_IF
 WAIT RR(X_A|Y_A)==0
 FEED X_A|Y_A 25
 SHOM &H50
 HOME X_A|Y_A NEG_L
```

HOME[MPG-2314]

パルス発生　　コマンド

■書 式

HOME X Y U Z
HOME axs V

■使い方

HOME 10000 10000 1000 1000
HOME NEG_L NEG_L NEG_L NEG_L
HOME X_A -1000
HOME X_A|Y_A -1000

■機 能

原点復帰コマンド

■解 説

SHOM で設定された停止条件を与えながら HOME シーケンスを実行します。
HOME コマンドにはタイムアウトが有効になっていますので、適切に設定します。
タイムアウトで停止した場合は、現在値をクリアしません。X(0) 等が 0 でなければ、エラー停止です。

HOME コマンドの引数は、ニアオリジンサーチのための移動量です。移動中にニアオリジンを検出すると減速停止します。
SHOM で IN1_ON/IN1_OFF が設定されていると、ニアオリジン検出後、ACCEL で設定した最低速度で Z 相サーチとなります。
このときの移動方向は、SHOM で与えた CW もしくは CCW で決定されます。デフォルトは、CCW となっています。

プログラム 1 はタイムアウト 10 秒で原点復帰の例です。HPT(XIN0),HPT(YIN0),HPT(ZIN0) は、各軸の IN0 のモニタです。
ニアオリジンの内側にいると、退避移動をしてからの原点復帰となります。

なお、原点復帰ニアオリジンの移動量を大きくとりたい場合は、POS_L,NEG_L を用いてください。これらは、正・負の 3byte 長の最大数です。
ニアオリジン検出移動量は、HOME X_A -1000 のように軸指定定数を使用することもできます。

```
—program1—
10          PG   1
20    ACCEL   40000
30    ACCEL   Z_A 20000
40   SHOM   X_A|Z_A|Y_A IN0_ON|IN1_ON|CW
50   IF   HPT(XIN0)==0 : RMVS   X_A 20000 : END_IF
60   IF   HPT(YIN0)==0 : RMVS   Y_A 20000 : END_IF
70    IF   HPT(ZIN0)==0 : RMVS   Z_A -20000 : END_IF
80    WAIT   RR(ALL_A)==0
85    TMOUT   10000
90    HOME   -100000 -100000 0 100000


—XY ROBOT example—
*HOME1
  ACCEL X_A|Y_A 10000 100 100

  IF HPT(XIN0) != THEN
    RMVS X_A 10000
  END_IF
  IF HPT(YIN0) != THEN
    RMVS Y_A 10000
  END_IF
  WAIT RR(ALL_A)==0
  TIME 100

  SHOM X_A|Y_A IN0_ON
  HOME -100000 -100000 0 0
  WAIT RR(ALL_A)==0

  TIME 100
  RMVL 2000 2000 0 0
  WAIT RR(ALL_A)==0
  STPS 0 0 VOID VOID
  PRINT "XY HOME"
  TIME 100
  RETURN
```

HOUT

パルス発生　　コマンド

■書 式
HOUT arg

■使い方
HOUT 1

■機 能
MPG ボードの出力ポートの制御

■解 説
MPG-2314 出力ポートの 4 ビットの一括設定です。

HPT

パルス発生　　関数

■書 式

HPT(arg1)

■使い方

```
prx HPT(0)
WAIT HPT(XIN0)==1
IF  HPT(XIN0)==0 : RMVS  X_A 20000 : END_IF
```

■機 能

PG ボードの原点復帰入力ポートを読み出す。

■解 説

【MPG-2314】
HPT(0) パラレルで IN0 ～ IN3,INPOS,ALM 入力を読み取る。
パラレル時は 8bit 単位各軸で 32bit パラ出力となります。

1) HPT(0)={U}{Z}{Y}{X}
{8bit}=ALM(bit7),INP(bit6),IN3(bit3)IN1(bit2,bit1)IN0(bit0)

IN1(bit2,bit1) は、MPG-2314 では、IN1 入力が、内部で IN1,IN2 とショートされていることを示しています。この為、IN1 をオンとすると IN2 もオンとなり、この場合 "hpt(0) -> 00000006" となります。

2) HPT(XIN0) 指定された、ポートを読み取る。
原点復帰入力 :XIN0,XIN1,XIN2,XIN3 ～ UIN0,UIN1,UIN2,UIN3
ALM 入力 : XALM ～ UALM
INPOS 入力 : XINP ～ UINP

HPT で読み取られる各指定ポートがオンであれば 1 となります。

HSW

IO　　関数

■書 式

HSW(arg)

■使い方

```
A=HSW(192)
IF HSW(192)&HSW(200)&HSW(208) THEN
```

■機 能

入力ポートのビット読み取り

■解 説

SW() は入力ポートに実入力ポートを指定すると 2 度読みをします。これに対して HSW() は 1 度読み取りのみです。

IF HSW(192)&HSW(200)&HSW(208) THEN
このように複数の SW の論理をとる場合、SW() では 2 度読みが発生し実行速度が遅くなってしまいます。上の例のように HSW() を使用すれば、高速で論理演算を行います。
出力ポート、メモリ I/O,MBK の I/O エリアのモニタでは、この区別はありません。

H_OFF

パルス発生　　コマンド

■書 式
　H_OFF arg

■使い方
　H_OFF 2

■機 能
　MPG-2314 の出力ポートオフ

■解 説
　通常の出力ポート同様ビットオフします。

H_ON

パルス発生　　コマンド

■書 式
　H_ON n

■使い方
　H_ON 1

■機 能
　MPG-2314 の出力ポートのオン

■解 説
　通常のポート同様ビットオンします。

IF-THEN-ELSE-END_IF

制御文　　ステートメント

■書 式
　IF arg THEN

■使い方
　IF SW(0)==1 THEN : ON 0 : END_IF
　IF SW(0)==1 THEN : ON 0 : ELSE : ON 1 : END_IF

■機 能
　条件分岐

■解 説
　arg が 0 でない場合 THEN の後ろを実行。0 の場合は、ELSE の後ろを実行するか、END_IF の後ろへジャンプ

IN

IO　　関数

■書 式
　IN(arg1)

■使い方
　IF IN(0)==&HAA THEN
　WAIT IN(1)==&H05
　A=IN(0~Lng)

■機 能
入力ポートのパラレル取り込み (8bits)

■解 説
MPC-2000 入力ポートは 24,25
最初の MIO-1616 の入力ポートは 26,27 となります。負の数を指定すると、メモリ I/O となります。
アドレス値に ~Lng,~Wrd,~Int を与えると、それぞれロング読み取り、整数 2byte 読み取り、符号付 2byte 読み取りとなります。タッチパネルの mbk エリアは、70000 以上を指定します。
ab は 00 ～ 99 となります。
IN(7ab00) byte 読み取り
IN(7ab00~Ub) hi-byte 読み取り
IN(7ab00~Wrd) ワード読み取り
IN(7ab00~Lng) ロング読み取り

```
DSW=IN(24)/16              /* GET DSW value and Shift down 4bits
```

IN0_OFF

パルス発生 予約定数

■書 式
IN0_OFF

■使い方
SHOM X_A IN0_OFF

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN0 ～ ZIN0 を OFF で有効にします。 see also IN0_ON

```
SHOM X_A IN0_OFF
STOP X_A IN0_OFF
```

IN0_ON

パルス発生 予約定数

■書 式
IN0_ON

■使い方
SHOM X_A|Y_A IN0_ON

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN0 ～ ZIN0 を ON で有効にします。

```
100 SHOM X_A|Y_A IN0_ON          /* set HOME condition.
110 HOME -100000 -100000 0 0
120 WAIT RR(ALL_A)==0
—
100 STOP X_A IN0_ON              /* set stop condition. if XIN0 turn on then stop.
110 MOVL 5000 0 0 0
120 WAIT RR(X_A)=0               /* wait for stop
130 IF HPT(XIN0)==1 THEN         /* confirming reason for stop
140   PRINT "IN0 stop"
```

```
150 ELSE
160   PRINT "normal stop"
170 END_IF
```

IN1_OFF

パルス発生　　予約定数

■書 式
IN1_OFF

■使い方
SHOM X_A IN1_OFF

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN1 ～ ZIN1 を OFF で有効にします。 see also IN0_ON

```
SHOM X_A IN1_OFF
STOP X_A IN1_OFF
```

IN1_ON

パルス発生　　予約定数

■書 式
IN1_ON

■使い方
SHOM X_A IN1_ON

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN1 ～ ZIN1 を ON で有効にします。 see also IN0_ON

```
SHOM X_A IN1_ON
STOP X_A IN1_ON
```

IN2_OFF

パルス発生　　予約定数

■書 式
IN2_OFF

■使い方
STOP X_A IN2_OFF

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN2 ～ ZIN2 を OFF で有効にします。 see also IN0_ON

```
STOP X_A IN2_OFF
```

IN2_ON

パルス発生　　予約定数

■書 式
IN2_ON

■使い方
STOP X_A IN2_ON

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN2 ～ ZIN2 を ON で有効にします。　see also IN0_ON

```
STOP X_A IN2_ON
```

IN3_OFF

パルス発生　　予約定数

■書 式
IN3_OFF

■使い方
STOP X_A IN3_OFF

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN3 ～ ZIN3 を OFF で有効にします。　see also IN0_ON

```
STOP X_A IN3_OFF
```

IN3_ON

パルス発生　　予約定数

■書 式
IN3_ON

■使い方
STOP X_A IN3_ON

■機 能
停止入力設定

■解 説
対象ボード : MPG-2314
XIN3 ～ ZIN3ON で有効にします。　see also IN0_ON

```
STOP X_A IN3_ON
```

INC

演算　　コマンド

■書 式
INC var [Val]

■使い方
INC A
INC A -10

■機 能
変数の増減 (マルチタスク)

■解 説
マルチタスクで共有の変数を用いて増減を行うと、リードとセットがバラバラのタイミングになる場合があり、タスク間で変数の増減を正確に行えない場合があります。
INC コマンドはリード & セットをタスク内で完結させるため、そうした不具合がありません。
引数がない場合は単純な +1 です。引数を追加すると、その値を変数に加えます。

INCHK

パルス発生 コマンド

■書 式
INCHK

■機 能
MPG-2314 の入力状態のモニタ

■解 説
INCHK と入力すると入力ポートの状態を表示します。'q' をタイプすると停止します。

```
inchk
 MPG-2314
 X=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
 Y=+LMT:off-LMT:off ALM:off INP:off IN0:on  IN1:off
 U=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
 Z=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
#
```

INPUT

通信 コマンド

■書 式
INPUT [CH] [EOL|x] [CHR_C|x] [TMOUT|x] a$

■使い方
INPUT a$

■機 能
文字列入力

■解 説
INPUT は、INPUT# で CH0(プログラムポート) に固定されたシリアル入力コマンドです。
使い方は、INPUT# と同様ですので、INPUT# を参照してください。

INPUT#

通信 コマンド

■書 式
INPUT# [CH] [EOL|x] [CHR_C|x] [TMOUT|x] a$
INPUT# [CH] CLR_BUF

■使い方
INPUT# a$

```
INPUT#   CH a$
INPUT#   5 EOL|10 c$
INPUT#   3 CHR_C|54 a$
INPUT#   3 TMOUT|10 a$
INPUT# 20 a$
INPUT# 2 CLR_BUF
INPUT# 5 COMPOWAY rcv$
```

■機 能

RS-232C ポートより文字列を取り込む。

■解 説

INPUT# は シリアルポートより文字列を取り込みます。CH 番号を省略すると、CH1 になります。
デフォルトでターミネータは CR となっていますが、EOL|xx オプションで変更できます。
xx にはアスキーコード。
文字カウント取り込みの場合は、CHR_C|xx オプションを用います。xx は 255 以内の数値。
カウントを指定するとターミネータは無視されます。
タイムアウトを必要とする場合は、TMOUT|xx オプションを用います。
xx には秒単位で制限時間をいれます。
TMOUT|10 の場合 10 秒以内でよみとれない場合は処理を打ち切ります。
タイムアウトしたかどうかは、rse_ 変数を参照します。rse_ が 1 の場合タイムアウトとなったことを意味します。
CLR_BUF を引数に与えた場合は、バッファの文字列をすべて読み捨てます。
なお、オプションパラメータとして COMPOWAY を与えると、OMRON COMPOWAY フォーマットでの受信になります。
TMOUT オプションも併用できます。チェックサムエラー発生の場合は、rse_ が 4 となります。
COMPOWAY フォーマットで受け取った文字列は、COMPOWAY コマンドで基本分解することができます。

数値変換は、VAL 関数 ,GET_VAL などの文字列処理コマンドで行います。

```
  INPUT#  a$
```

a=VAL(a$) : b=VAL(0) 関数 VAL を参照してください。

```
–Serial Communication–
CNFG#   3 "38400b8pns1NONE"
CNFG#   4 "38400b8pns1NONE"
CNFG#   5 "38400b8pns1NONE"
 a$="123456789012345678abcdefghijklmnopqrstuvwxyz$%&()01234"
'     GOTO  *RS422
  DO
   PRINT#   3 a$ "¥r"
   INPUT#   4 EOL|13 b$
   PRINT#   4 b$ "¥r"
   INPUT#   5 EOL|10 c$
   PRINT#   5 c$
   INPUT#   3 CHR_C|54 a$
   PRINT   a$
  LOOP
 *RS422
  DO
   PRINT#   4 a$ "¥r"
   INPUT#   5 b$
   PRINT   b$
   PRINT#   5 b$ "¥r"
   INPUT#   4 a$
  LOOP
```

INP_OFF

パルス発生　　予約定数

■書 式
　INP_OFF

■使い方
　INSET X_A INP_OFF

■機 能
　インポジション設定

■解 説
　対象ボード : MPG-2314
　インポション OFF で有効。INP_ON/INP_OFF はどちらかを設定すれば有効、しなければ無効です。

```
INSET X_A INP_OFF                /* X-axis 'INPOSITION' enabled on signal 'OFF'
```

INP_ON

パルス発生　　予約定数

■書 式
　INP_ON

■使い方
　INSET X_A INP_ON

■機 能
　インポジション設定

■解 説
　対象ボード : MPG-2314
　インポション ON で有効。INP_ON/INP_OFF はどちらかを設定すれば有効、しなければ無効です。

```
INSET X_A INP_ON                 /* X-axis 'INPOSITION' enabled on signal 'ON'
```

INSET

パルス発生　　コマンド

■書 式
　INSET [axs] Settings

■使い方
　INSET　PHASE4
　INSET ALL_A ALM_ON|INP_OFF

■機 能
　MPG-2314 用入力設定コマンド

■解 説
　入力ポートの機能を設定します。機能と予約定数との関係は以下のとおりです。

INPOS	= INP_ON,INP_OFF,INP_NO
ALARM	= ALM_ON,ALM_OFF,ALM_NO
LMT	= LMT_ON,LMT_OFF
ソフトリミット	= SLMT_ON,SLMT_OFF
エンコーダ	= UP_DWN,PAHSE1,PAHSE2,PAHSE4
PLS	= MD_2PLS,MD_DPLS(パルス発生モードの設定)

　最後に実行した INSET が有効になり、パラメータで与えられたもの以外の設定はリセットされます。

例)
INSET X_A ALM_ON|INP_ON
・X 軸に対して入力設定。アラームを有効にして ON 状態をアラームとする。また、INPOS を ON で有効とする。

INSET ALL_A ALM_ON|INP_OFF
・すべての軸に対して入力設定。アラームは ON で有効、INPOS は OFF で有効とする。

INSET PHASE4
・エンコーダ入力を四逓倍とする。

INSET ALL_A VOID
・全設定クリア。RANGE 設定 (ソフトリミット) もクリア

INSET【MPC-1200】

パルス発生　　コマンド

■書 式
INSET [axs] MD_DPLS

■使い方
INSET MD_DPLS
INSET Z_A MD_DPLS

■機 能
MPC-1200 用　パルス出力設定コマンド

■解 説
MPC-1200 のパルス発生は、CW,CCW パルス方式がデフォルト設定となっていますが、INSET コマンドで方向指示型に変更することができます。方向指示に変更したい軸を指定して、MD_DPLS を引数を与えます。
パワオンリセツトで CW,CCW 方式に戻ります。軸指定を省力すると全軸対象となります。

INSPEC

保守　　コマンド

■書 式
INSPEC

■機 能
セルフテスト

■解 説
現在は RAM の Write/Read テストのみです。

```
--------------------------------
#inspec
INSPECTION
1:Test Memory
PASSED
#
```

Int

タッチパネル　　予約定数

■書 式
Int

■使い方
IN(-1~Int)

■機 能
ワード型指定 (符号付)

■解 説
S_MBK,MBK(),IN,OUT の符号付 16bit 読み取りを指定します。

```
10      S_MBK   &H00008FFF 20~Wrd /* WORD write
20      PRINT   MBK(20~Wrd)       /* unsigned WORD read
30      PRINT   MBK(20~Int)       /* signed WORD read
40      OUT   -1 -1~Wrd           /* WORD write
50      PRINT   IN(-1~Wrd)        /* unsigned WORD read
60      PRINT   IN(-1~Int)        /* signed WORD read
RUN

 36863                            /* unsigned
-28673                            /* signed
 65535                            /* unsigned
-1                                /* signed
```

INTA_ON,INTB_ON

パルス発生　　コマンド

■書 式
INTA_ON portn (PG,axis)
INTB_ON portn (PG,axis)

■使い方
INTA_ON 16 (0,X_A)
INTB_OFF 17 (0,U_A)

■機 能
MPG-2314 の割り込によるポート ON もしくは OFF

■解 説
INTA_ON は、MPG-2314 のカウンタ比較検出 割り込みによりポートをオンします。割り込みを作動させるためには、以下のコマンド設定が必要です。

・比較カウンタの設定 INSET axis CMP_PLS　(もしくは CMP_CNT)
　CMP_PLS = パルス位置 , CMP_CNT = エンコーダ・カウンタ位置
・ 割り込みの有効化 STOP axis C_MORE (もしくは C_LESS)
　C_MORE Pls >= COMP+ , C_LESS PLs < COMP+
・比較値 COMP+ の設定
　　RANGE axis VAL1 dummy

以上の設定があり、カウンタ値が VAL1 を超えると (C_MORE の場合) 割り込みが発生し、INTA_ON で指定されたポートを ON します。(OFF の場合は INTA_OFF を用いる)
割り込みが発生したあとは、関数 RR3(axis) を読み出すことによって割り込みは解除されます。但し、解除する前に、RANGE 設定によって比較値 COMP+ を条件の外に変更しておく必要があります。

初期化

```
INSET axis CMP_CNT          /* 比較カウンタの選択
STOP  axis C_MORE           /* 割り込み比較条件設定
RANGE axis VAL1 dummy
a=RR3(axis)                 /* 割り込みをクリアしておく
```

実行順序としては、
RANGE axis VAL1 summy　　/* 条件を変えておく
a=RR3(axis)　　/* 割り込みクリア
SWAP
INTA_ON port　　/* 割り込みポート設定
WAIT SW(port)　　/* 割り込み発生の検出
となります。
*RR3 の axis は、X_A,Y_A 等の単一軸指定のみ意味を持ちます。

割り込みの解除は、INTA_ON VOID あるいは、INTA_OFF VOID を実行します。
なお、割り込みには、INTB_ON,INTB_OFF もあり、INTA_ と合わせて 2 つの PG の割り込みまで対応することができます。
サンプルプログラムは、移動 500 パルスごとに割り込みを発生させ、外部にタイミングトリガを出力します。
INTA_ON,_OFF,INTB_ON,_OFF は、ソフトウェアのタイミング待ちと異なり、正確に位置タイミングを出力することができます。

```
-----------------------------------------------------
INTA_ON   VOID
 PG   0
 PG   0 1
 PG   0 3
 ACCEL   5000
 CLRPOS
 CLRPOS   -1
 INSET   X_A CMP_PLS
DET_P=500
 STOP   X_A C_MORE
 RANGE   X_A DET_P 0
 PG   0
 FORK   1 *MPG
 FORK   3 *MPG2
 END
*MPG2
 DO
  INTA_ON   0 (0,X_A)
  WAIT   SW(0)
  TIME   1
  OFF   0
  INC   DET_P 500
  RANGE   X_A DET_P DET_P
 A_=RR3(X_A)
  TIME   5
 LOOP
*MPG
 RMVC   X_A 1
 END
```

JMPZ

パルス発生　　コマンド

■書 式
　JMPZ Pnt

■使い方
　JMPZ P(n)

■機 能
　Z 下降なし JUMP

■解 説

ゲートモション・コマンド JUMP の部分実行で、Z 軸の下降を行いません。
JMPZ は複数の動作が組み合わされた、複合コマンドです。このため、PAUSE、STOP、CONT を実行すると、思わぬところで横移動してしまうことがあります。
これを防ぐため JMPZ コマンドには PAUSE された場合の再実行がコマンドが組み込まれています。この機能を有効にするには、PAUSE (STP_D,n) によって、対象タスクを停止させます。
こうして停止したタスクは、CONT コマンドにより JMPZ コマンドが再実行されます。
CONT コマンド実行後、0.1 秒間は、再度 PAUSE (STP_D,n) を実行しないでください。

JPN

保守　　コマンド

■書 式

JPN

■機 能

日本語モードにする。

■解 説

日本語モードにする。エラー表示は日本語となる。MPCINIT 後は英語モードとなっている。

JUMP

パルス発生　　コマンド

■書 式

JUMP P(arg)
JUMP PL(pln;plm)
JUMP argx,argy,argu,argz

■使い方

JUMP P(1)
JUMP PL(0;5)
JUMP X Y U Z

■機 能

ゲートモーション

■解 説

JUMP P(n) 点 n へ、ゲートモーション移動
JUMP PL(n;m) パレット n の m 番目へ、ゲートモーション移動
JUMP X Y U Z 座標点へ、ゲートモーション移動
JUMP は複数の動作が組み合わせれた、複合コマンドです。このため、PAUSE、STOP、CONT を実行すると、思わぬところで横移動したり、下降してしまうことがあります。
これを防ぐため JUMP コマンドには PAUSE された場合の再実行がコマンドが組み込まれています。この機能を有効にするには、PAUSE (STP_D,n) によって、対象タスクを停止させます。
こうして停止したタスクは、CONT コマンドにより JUMP コマンドが再実行されます。
CONT コマンド実行後、0.1 秒間は、再度 PAUSE (STP_D,n) を実行しないでください。

サンプルプログラムは、
1 行目で、パレット 1 の 2 番目の位置の Z 位置が 500 パルス上へ、ゲートモーション移動
2 行目は、パルス出力完了待ち
3 行目は、メカチャックを OFF、つまりワークピースを離すという意味になります。

```
JUMP PL(1;PT) AD_P(Z_A,500)
WAIT RR(ALL_A)==0
OFF 14
```

LABELS

保守　　コマンド

■書 式

LABELS

■機 能

ラベル検査

■解 説

ラベルの二重定義を検査します。二重ラベルを発見すると以下のように表示します。

The two same labels
12810　*bb
13400　*bb

LATCH

IO　　コマンド

■書 式

LATCH i SW(n) IN(m) j
LATCH i @SW(n) IN(m) j

■使い方

LATCH　1 SW(208) IN(24) 2
LATCH VOID

■機 能

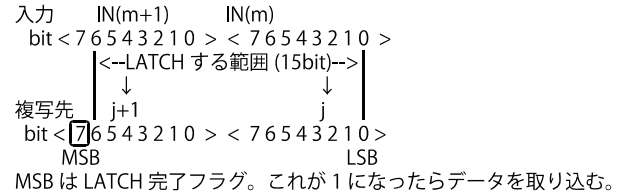
15bit ラッチ

■解 説

SW(n) をトリガとして、IN(m),IN(m+1) のパラレル値をバンク j,j+1 に複写します。
ただし j+1 バンクの MSB は、ラッチマークに使用されるので、有効データは 15bit です。
@SW(n) とすると論理反転入力になります。

LATCH i SW(n) IN(m) j

i　　: ラッチ番号 0~3
SW(n)　: トリガ入力。SW(n) で負論理、@SW(n) で正論理
IN(m)　: 入力信号
j　　: ラッチ (複写) 先ポート番号。正数は標準出力、負数はメモリ I/O



MSB は LATCH 完了フラグ。これが 1 になったらデータを取り込む。

トリガは 1msec 幅以上の ON 信号。
i は 0 ～ 3 まで指定可能なため、4 種類のラッチを設けることができきます。
LATCH VOID ですべてのラッチ設定を解除できます。また、パワーオンリセットでもラッチ機能は無効になります。
LATCH コマンドは引数無で現在のラッチ機能設定状態を確認することができます。

サンプルプログラムではそれぞれのトリガ信号に対して同じ入力バンクから、それぞれ異なるメモリ IO にデータを複写します。
ハンドシェークは、j+1 バンクのメモリ I/O の MSB で行っています。
なお、出力には使用していない出力ポートを指定して用いることもできます。

```
10      LATCH   0 SW(208) IN(24) -2
20      LATCH   1 SW(209) IN(24) -4
30      LATCH   2 SW(210) IN(24) -6
40      LATCH   3 SW(211) IN(24) -8
45      DO
50       IF   SW(-8)==1 THEN  : PRINT   "latch 0" : WAIT   SW(208)==0 : OFF   -8 : END_IF
60       IF   SW(-24)==1 THEN  : PRINT   "latch 1" : WAIT   SW(209)==0 : OFF   -24 : END_IF
70       IF   SW(-40)==1 THEN  : PRINT   "latch 2" : WAIT   SW(210)==0 : OFF   -40 : END_IF
80       IF   SW(-56)==1 THEN  : PRINT   "latch 3" : WAIT   SW(211)==0 : OFF   -56 : END_IF
90      LOOP
```

LEN

文字列　　関数

■書 式

LEN(string)

■使い方

print LEN(a$)
a=LEN(a$)

■機 能

文字列の文字数をカウントします。

■解 説

与えられた文字列の文字数を返します。

LIFE_TIME

時間管理　　コマンド

■書 式

LIFE_TIME [val]

■使い方

LIFE_TIME 100

■機 能

タイムスライス時間制御

■解 説

MPC-2000 のタイムスライスはデフォルトで 3msec ですが、アプリケーションによっては、この時間を調整したほうがよい場合があります。
LIFE_TIME コマンドをこの時間を 10 μ秒単位で、500 μ秒から 5msec の間で設定することができます。
LIFE_TIME 250 --> 2.5msec の意味です。
パワーオンリセットでデフォルトに戻りますので、変更が必要な場合はプログラムに記述します。
また、引数無しの場合、現在のタイムスライス時間を返します。

LIMZ

パルス発生　　コマンド

■書 式

LIMZ arg1 [arg2]

■使い方

LIMZ -5000
LIMZ -5000 100

■機 能
JUMP(ゲートモーション) の高速化

■解 説
JUMP は、Z 軸上昇後、XYU 軸を移動させます。
デフォルトでは、Z の値が 0 になるまで移動 (上昇) しますが、装置の速度が遅くなります。
LIMZ によって、この上昇天井を定めることができます。LIMZ -1000 の場合、-1000 の位置まで上昇します。
arg2 は、Z 軸の上昇が開始されて arg2 msec 経過したら、XYU 移動を開始させます。
これにより、ゲートモーションの出発点側の動きがアーチとなります。

LIST

編集 コマンド

■書 式
LIST arg1 [arg2]

■使い方
LIST 10 3
LIST *AHO
LIST

■機 能
プログラムリストを表示する。

■解 説
1 番目の引数は、表示指定文番号です。2 番目の引数は、表示行数です。
LIST 単独の場合は、先頭から表示します。無引数のまま LIST を実行すると続きを表示します。

LMT

パルス発生 関数

■書 式
LMT(n)

■使い方
```
IF LMT(X_A,LMTp)!=0 THEN
 RMVS X_A -10000
 END_IF
```

■機 能
エラー入力読み取り

■解 説
【MPG-2314】
LMT(0) とすると、XYZU すべての軸のエラー状態を参照できます。
byte= { EMG,ALM,LMTn,LMTp,SLMTn,SLMTp}
UbyteZbyteYbyteXbyte
もう 1 つの参照法は、LMT(X_A,ALM) のように軸とビットパラメータを与える方法です。
この場合は、軸指定 (X_A) と参照エラービット (ALM) を指定しています。

LMTn

パルス発生 予約定数

■書 式
LMTn

■使い方
LMT(X_A,LMTn)

■機 能
エラービット指定

■解 説
対象ボード : MPG-2314
ハードリミット - ビット

```
IF LMT(X_A,LMTn)!=0 THEN          /* confirming reason for stop
```

LMTp

パルス発生　　予約定数

■書 式
LMTp

■使い方
LMT(X_A,LMTp)

■機 能
エラービット指定

■解 説
対象ボード : MPG-2314
ハードリミット + ビット

```
IF LMT(X_A,LMTp)!=0 THEN          /* confirming reason for stop
```

LMT_OFF

パルス発生　　予約定数

■書 式
LMT_OFF

■使い方
INSET ALL_A LMT_OFF

■機 能
リミット入力設定

■解 説
対象ボード : MPG-2314
X-LMT ～ Z-LMT OFF で有効。リミット検出時は即停止。入力を無効にすることはできません。

```
INSET ALL_A LMT_OFF               /* 'LIMIT' enabled on signal 'OFF'
```

LMT_ON

パルス発生　　予約定数

■書 式
LMT_ON

■使い方
INSET ALL_A LMT_ON

■機 能
リミット入力設定

■解 説

対象ボード : MPG-2314
X-LMT ～ Z-LMT ON で有効。
リミット検出時は即停止。入力を無効にすることはできません。

```
INSET ALL_A LMT_ON                  /* 'LIMIT' enabled on signal 'ON'
```

Lng

タッチパネル　　予約定数

■書 式

Lng

■使い方

MBK(20~Lng)

■機 能

ロング型 (2 ワード) 指定

■解 説

S_MBK,MBK(),IN,OUT の値の 32bit ロング型での読み取り指定です。

```
10      S_MBK   &H12345678 20~Lng       /* LONG write MBK data area 20,21
20      PRX   MBK(20~Lng)               /* LONG read MBK data area 20,21
30      PRX   MBK(21) MBK(20)           /* WORD read
40      OUT   &H87654321 -1~Lng         /* LONG write memory I/O area -1~-4
50      PRX   IN(-1~Lng)                /* LONG read memory I/O area -1~-4
60      PRX   IN(-4) IN(-3) IN(-2) IN(-1)   /* BYTE read
RUN

12345678                                /* LONG read
00001234 00005678                       /* WORD read
87654321                                /* LONG read
00000087 00000065 00000043 00000021     /* BYTE read
```

LOCK

保守　　コマンド

■書 式

LOCK LockKey
LOCK -1 LockKey

■使い方

LOCK 1234567
LOCK -1 1234567

■機 能

プログラムの秘密化、変更禁止

■解 説

プログラムを秘密化します。
LOCK コマンドは RUN 前、RUN 後いずれにも使用できます。
RUN 前に LOCK した場合は、必ず RUN を実行します。
解除は、最初の引数を -1 として、設定時と同じ値をいれます。
LOCK が解除され、再度編集可能な状態となります。
そのまま再ロックする場合は、ERASE を実行してから LOCK を実行します。
継続編集を必要としない場合は、MPCINIT,ERASE でも、LOCK は解除されます。

LOF

通信　　関数

■書 式
LOF(ch)

■使い方
IF LOF(1)>10 THEN : input# 1 a$: END_IF

■機 能
バッファの文字列数を返す。

■解 説
各 RS-232C ポートのバッファにたまった文字数を返します。引数の CH は 0 ～ 11 でに対応。
また、LOF(20) の場合は USB メモリの残文字の有無で、1 であり、0 で EOF まで達したことを意味します。

PRX LOF(-1) とすると MRS-MCOM のバージョンを表示します。
-1 は CH3 ～ CH5、-2 は CH6 ～ CH8、-3 は CH9 ～ CH11 を指定します。
#PRX LOF(-1)
20090514
MRS-MCOM6 では、電源が投入されていないとこの値が 0 となります。

LOG

保守　　コマンド

■書 式
LOG [arg]

■使い方
LOG
LOG 0
LOG 1

■機 能
ログ表示

■解 説
LOG は実行中にプログラムポートに出力された文字の記録です。
LOG バッファ NEW, もしくは、LOG 0 でクリアされます。
停止時あるいは稼動中に LOG コマンドによってプログラムポートの出力を表示します。
20 行ごとに順々に表示しますので、LOG コマンドを繰り返します。

最初から表示する場合は、LOG 1 とします。LOG を初期化するには、LOG 0 を実行します。

LOG コマンドを実行すると LOG は停止します。再開させるためには LOG 3 を実行します。
稼動中の装置で LOG を実行をして様子をみたあとはかならず LOG 3 を実行して LOG を継続させます。

LONG_PRG

タッチパネル　　予約定数

■書 式
LONG_PRG

■使い方
S_MBK LONG_PRG

■機 能
プログラム番号のロング化

■解 説
タッチパネル用プログラム番号のロング化。
通常システムは、ワード領域 MBK(7868) ～ MBK(7899) に、実行しているプログラムの文番号をセットします。
プログラムが大きくなって番号が 65535 以上の値を持つ場合に。このコマンドでワード書き込みをします。この場合 MBK(7836) ～ MBK(7899) に、プログラム文番号をロング整数書き込みします。

```
10 MEWNET 38400 1                  /* RS-232C CH1 -> MBK-RS 38400bps
20 S_MBK LONG_PRG                  /* upper MBK(7836) -> long numeric
```

MBK

タッチパネル　　関数

■書 式
MBK(arg)

■使い方
a=MBK(n)
a=MBK(n~Lng)
MBK(n)=a
b=MBK(n~Int)

■機 能
タッチパネルデータを参照、設定する。

■解 説
MBK 配列は、タッチパネルと接続されると、メモリ共有される配列です。
MBK(n) は DTn に対応します。

a=MBK(n) → タッチパネルデータをワード型で取り出す。

b=MBK(n~Int) → タッチパネルデータを符号付ワード型で取り出す。 たとえば値が &HFFF0 の場合 -16 となる。
a=MBK(n~Lng) → タッチパネルデータをロング型で取り出す。Hi ワードが n+1 アドレス

MBK(n)= 式 → タッチパネルデータにワード形で代入する。

MBK(n~Lng)= 式 → タッチパネルデータにロング形で代入する。

MBK(n) には以下の予約領域があります。
1) 文番号
MBK(7868) ～ MBK(7899) 実行中のプログラムの文番号です。ワード型です。
文番号が 65535 を超える場合は、
S_MBK LONG_PRG
を実行します。以後 MBK(7836) ～ MBK(7899) にロング型で文番号が格納されます。

2) バージョン番号
MBK(8053) には、バージョン番号が格納されています。
ファームウェアバージョンが 1.12_60 であれば、
pr MBK(8053) -> 11260
となります。

3)MBK(7900) ～ MBK(7999) までは、タッチパネル側での R エリアとして扱われます。
R エリアのバンク 0 ～ 99 がこのエリアに対応します。

MBK$

タッチパネル　　関数

■書 式
MBK$(adr,val)

■使い方
A$=MBK$(100,6)

■機 能
MBK 配列を文字列として読み取る

■解 説
S_MBK a$ adr c と対になる関数です。MBK 配列上の文字列を読み出します。

MBK_CMD

タッチパネル　　予約変数

■書 式
MBK_CMD

■使い方
PRX MBK_CMD

■機 能
通信エラーキャラクタ

■解 説
MEWNET 通信で処理できなかったコマンド。
PRX MBK_CMD で 4142 であれば、AB という意味です。

MBK_ERR

タッチパネル　　予約変数

■書 式
MBK_ERR

■使い方
PR MBK_ERR

■機 能
通信エラーカウンタ

■解 説
MEWNET 通信のエラー回数を保持している変数です。

MD_2PLS

パルス発生　　予約定数

■書 式
MD_2PLS

■使い方
INSET ALL_A MD_2PLS

■機 能
パルス出力方法設定

■解 説
対象ボード : MPG-2314/MPC-1200
2 パルス方式 (CW/CCW) にする。

```
INSET ALL_A MD_2PLS              /* Set the pulse generator to '2 PULSE' mode
```

MD_DPLS

パルス発生　　予約定数

■書 式
MD_DPLS

■使い方
INSET ALL_A MD_DPLS

■機 能
パルス出力方法設定

■解 説
対象ボード : MPG-2314/MPC-1200
1 パルス方式 (方向指示) にする。

```
INSET ALL_A MD_DPLS              /* Set the pulse generator to 'DIR/PULSE' mode
```

MEWNET

タッチパネル　　コマンド

■書 式
MEWNET arg1 [COMn] [mode]
MEWNET [COMn]

■使い方
MEWNET 9600
MEWNET 9600
MEWNET 38400
MEWNET 38400 5
MEWNET 9600 1 B7O
MEWNET 38400 2 RS485
MEWNET 38400 1 COM

■機 能
タッチパネル用 MEWNET プロトコル設定

■解 説
タスクを MEWNET(Panasonic 電工 FP シリーズコンピュータリンク) 通信にわりあてて、MBK() 配列とタッチパネルをデータ共有させます。(WD,WC,RD,RC プロトコルによる共有)
どのタスクが割り当てらるかは、以下の規則に従って通信チャンネル番号で決定されます。

割り当てタスク = 32 - ch 番号
このため、最初のユーザチャンネル CH1 を MEWNET として使用すると、タスク 31 を通信タスクとして占有します。MRS-MCOM の最初の CH 番号は 3 です。この場合は、32 - 3 = 29 が、通信タスクに割り当てられます。

ボーレートは 9600,19200,38400 のうちから選択できます。
第 2 引数は RS のチャンネル番号で 1 ～ 5 を指定できます。(MPC-1000,N816 は 1,2 のみ)
第 3 引数は、通信フォーマット設定です。通常は省略し、8 ビットパリティ無し通信としますが、パリティやビット数の必要な場合に追加します。

B7O: 7bit 奇数パリティ
B7E: 7bit 偶数パリティ
B8O: 8bit 奇数パリティ
B8E: 8bit 偶数パリティ

なお、MEWNET コマンドは、通信プロトコルの初期化を含みますので、CNFG# コマンドとは併用しないでください。
例
MPC-2200 CH2 で接続 占有タスクは 30
MEWNET 38400 2
MRS-MCOM #1 CH5 で接続 占有タスクは 27 (RS-232,RS-422 とも、RS-485 は未対応です。)
MEWNET 38400 5
MPC-2000 CH1 で接続 占有タスクは 31 Bit7 奇数パリティ (三菱小型タッチパネル)
MEWNET 9600 1 B7O
B7E は Bit7 偶数パリティの場合

なお、MEWNET コマンドは一度実行されると、以後自動起動になります。このため、CH を変更する場合は、
MEWNET [COMn] を実行して登録を抹消してください。
*MEWNET 1 の場合、COM1 での MEWNET 停止。

MPCINIT ではすべての登録を初期化します。

--- デジタル社 GP2400 設定例 ---

初期設定 > I/O の設定 > 通信の設定
 伝送速度　　38400 (MEWNET コマンドと一致させます)
 データ長　　8
 ストップビット 1
 パリティビット 無
 制御方式　　X 制御
 通信方式　　RS232C

初期設定 > I/O の設定 > 通信監視時間の設定
 通信タイムアウト時間 (1-127) [10] 秒　→ [1] 秒 などに短くします。

引数の末尾に RS485 を追加すると、RS485 ポートでの MEWNET 接続が可能になります。
しかし、MEWNET には RS485 が規定されていないため、MEWNET プロトコルで、RS485 通信に対応しないタッチパネルもありますので注意してください。

なお、データの対応は以下のようになります。
DT0 ～ : MBK(0) ～
R0 ～ : ON/OFF/SW/IN/OUT 7YYXX ～ (DT7900 ～ DT7999 と重複します)
YY= バンク番号 (0 ～ 99) XX= ビット番号 (0 ～ 15)

DT エリアは通常の番号対応ですが、R エリアは 70000 以上の値で、下 2 桁がビット番号、中 2 桁がバンク番号となります。IN/OUT の場合は、XX のビット番号を 0 とします。

MEWNET タスクは "$" 文字で始まる、CR でターミネートされたコマンドを即時実行する機能があります。これを有効にするには、引数中に予約変数 "COM" を加えます。

例 : MEWNET 38400 1 COM

MKY

CUnet　関数

■書 式
MKY(val)

■使い方
A=MKY(0)
PRX MKY(1)

■機 能
CUnet IC MKY の制御レジスタの読み取り

■解 説
CUnet チップである MKY40 の各レジスタの値を読み取ることができます。

MKY(0) SCR
MKY(1) BCR_SA(上位 2bit は、Baud)
MKY(2) BCR_OA(上位 2bit は、LFS,CP)
MKY(3) CHIP_CD :"MKY4" を数値として返します。prx MKY(3) -> 4D4B5934
MKY(4) MES(Mail Error Status)
MKY(5) SSR(System Status Register)
MKY(6) MFR(Member Flag Register 0-31
MKY(7) MFR(Member Flag Register 32-63
MKY(8) MCR(Member Care Counter) 読み取りとクリア
MKY(9) LCR(Link Care Counter) 読み取りとクリア

MKY(1) の上位 2bit を除く値は、パワーオン直後、DSW1,DSW2 の値となります。
これにより、CUnet の DSW の値によって、スタートアドレスを設定することができます。
MKY(3) により、ボードの有無を確認できます。
MKY(6),MKY(7) により、ネットワーク上の MKY の有無 (電源 ON,OFF) を確認することができます。
MKY(8),MKY(9) の値が頻繁に増加する (読み取るたびに 0 でない 1 以上の値がはいっている) ようであれば、通信の品質が外部要因で劣化しています。

MON

保守　コマンド

■書 式
MON [arg]

■使い方
MON
MON 1
MON 2

■機 能
実行状態の確認 監視

■解 説
プログラム停止後、mon を実行すると以下のような表示されます。

#mon
*0_ [20] *1 [42] *2q [42] *3! [42]

#

0_ は、タスク 0 が END で終了したことを意味しています。

2q は、タスク 2 が QUIT コマンドで終了させられたことを意味しています。
3! は、タスク 3 が実行中で時間ロスがあることを示しています。

以下はタスク 0 を停止状態 (コマンド受付可能) にしてタスクの状態を監視した様子です。
他のタスクから QUIT されると、文番号が残って QUIT となり、END で停止すると 4_ のように
タスク番号に _ が付与されます。
SLP は、PAUSE されたか TIME コマンドによる停止中を意味します。

```
#mon 1
*1 RUN [42] *2 QUIT[42] *3 SLP [42] *4_ QUIT[120]
*5! RUN [140]
```

MON 2 を実行すると、表示はされず LOG データに書き込みます。
通常は、コマンドとして使用しないで、プログラム中に埋め込み、
プログラム実行中に特定の箇所で、全体の実行状態を確かめるために有効な機能です。

MOVL

パルス発生　　コマンド

■書 式

```
MOVL P(n) [option]
MOVL PL(n;m) [option]
MOVL arg1,arg2,arg3,arg4 [option]
```

■使い方

```
MOVL P(1)
MOVL P(1) AD_P(X_A,100)
MOVL X Y U VOID
MOVL PL(1;1)
MOVL P(3) VOID_U
```

■機 能

指定点あるいは、指定座標への直線補間移動。(座標管理による直線補間パルス発生)

■解 説

MOVL は座標管理された、補間パルス発生です。
引数には直接の座標値、点データ、パレット点などを与えることができます。
ただし、補間は 3 軸までしか対応できないため、4 軸移動になる場合はエラーになります。
option には AD_P,X_A|Y_A,VOID_U などの補正関数や軸指定定数を追加できます。
X_A|Y_A とすると指定軸の補間、VOID_U などでは指定軸の無視 (パルス発生しない) 等ができます。

MOVS

パルス発生　　コマンド

■書 式

```
MOVS [axis] n
MOVS arg1 [arg2,arg3,arg4]
```

■使い方

```
MOVS x y u z
MOVS X_A n
MOVS x VOID u z
```

■機 能

座標管理をしたパルス発生。

■解 説

補間を伴わない絶対位置パルス発生。MPC-2000 では座標管理をしています。
MOVS では、現在位置と指定された値の差をとって、差分量のパルスを発生します。
尚、短軸の場合は、軸指定定数で指定できます。また、引数に VOID を指定すると、その軸は動作しません。

MOVT

パルス発生　　コマンド

■書 式

MOVT axs Point [CCW|CW|0]

■使い方

MOVT　X_A|Y_A P(101)
MOVT　X_A|Y_A P(102) CCW
MOVT　X_A|Y_A P(i) M(i)

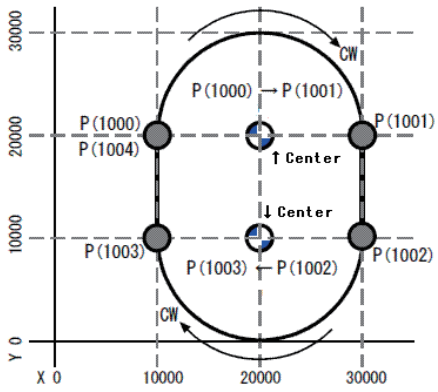
■機 能

座標値による連続補間移動

■解 説

点データによる連続補間です。絶対値でデータを入力しますが、実際には、起点 (ここでは P(100)) からの相対座標に変換されての移動となります。
第 3 パラメータに CCW,CW を入れると円弧補間となります。
第 3 パラメータが無いか、0 をすれれば、直線補間となります。

サンプルプログラムは、点データを使った汎用プログラムの例です。
P(1000) ～に座標データ　P(2000) ～に指令データをセットすると、さまざまな連続移動が可能になります。




```
PG   0
  ACCEL   8000
  CLRPOS
  GOSUB   *SET_POINT
 axis=X_A|Y_A
  MOVL   axis P(1000)
  WAIT   RR(axis)==0
  FEED   axis Y(2000)
  DS_DACL
  FOR   pnt=1001 TO X(2000)
   MOVT   axis P(pnt) X(1000+pnt)
  NEXT   pnt
  EN_DACL
  WAIT   RR(axis)==0
  END
*SET_POINT
  SETP   1000 10000 20000 0 0 : 'Start
  SETP   1001 30000 20000 20000 20000 : 'Cir target and Center
  SETP   1002 30000 10000 0 0
  SETP   1003 10000 10000 20000 10000
  SETP   1004 10000 20000 0 0
  SETP   2000 1004 50 0 0 :     'End of Point and FEED value
  SETP   2001 CW 0 0 0 :        'P(1000) to P(1001) CW
  SETP   2002 0 0 0 0 :         'P(1001) to P(1002) linear
  SETP   2003 CW 0 0 0 :        'P(1002) to P(1003) CW
  SETP   2004 0 0 0 0 :         'P(1003) to P(1004) linear
  RETURN
```

MPCINIT

編集　　コマンド

■書 式
MPCINIT

■機 能
MPC を初期状態にする。

■解 説
プログラムエリアを消去、変数・点データ・配列エリアを 0 にします。
I/O エリアをすべて OFF(クリア) 状態とします。

MPG

パルス発生　　コマンド

■書 式
MPG arg [taskn]
MPG

■使い方
MPG 1
MPG 1 4

■機 能
MPG ボードのアサイン

■解 説
どの MPG ボードを使用するか決定。タスクごとに指定できる。

taskn を指定しなければ、実行されたタスクで MPG を指定。
指定すると、指定タスクが、その MPG を使用する。指定結果は、MPG でリストできる。
同様のコマンドに PG があるが、こちらは、指定 PG の存在の有無について判定しない。

MPG コマンドでは存在しない PG 番号を指定すると、エラーが表示される。

なお、MPG 0 ～ 9 が、MPG-2314 となり直線・円弧補間まで対応。
MPC-1200 は MPG 17 です。

同機能コマンドとして PG がありますが、こちらは、存在しない PG 番号を指定しても、エラーとはなりません。

M_SW

IO　　関数

■書 式
M_SW([n,]n)

■使い方
M_SW(192)
M_SW(10,193)

■機 能
フィルタ付き SW 関数

■解 説
メカスイッチや反射センサなどチャタリング信号を発生しやすい入力に対して使用する SW() 関

数です。
M_SW(n) の場合は、1msec ごとに三回、n ポートを読み取り三回とも同じ値の場合の時のみポート値を返します。
M_SW(t,n) の場合は、t が読み取り回数指定となり、t 回 (tmsec) 同じ値の場合に、ポート値を返します。
従って、入力が 1msec 周期でパルス状に変化している場合には、M_SW() 関数はサスペンドとなります。指定できるポート番号 n はボード上の入出力 I/O のみで、メモリ I/O などには使用できません。

NEG_L

パルス発生　　予約定数

■書 式
NEG_L

■使い方
HOME NEG_L NEG_L NEG_L NEG_L

■機 能
負の大数

■解 説
負の大数
原点復帰ニアオリジンの移動量を大きくとりたい場合は、POS_L,NEG_L を用いてください。
これらは、正・負の 3byte 長の最大数です。
```
#prx POS_L
007FFFF0
#prx NEG_L
FF80000F
```

```
HOME NEG_L NEG_L NEG_L NEG_L
```

NEW

編集　　コマンド

■書 式
NEW

■機 能
プログラム消去

■解 説
プログラムを消去し予約変数以外の変数を抹消します。

NEWP

パルス発生　　コマンド

■書 式
NEWP

■機 能
点データ初期化

■解 説
点データをすべて 0 に初期化します。

NOT

演算　　関数

■書 式
NOT(arg)

■使い方
A=NOT(1)

■機 能
引数のビット反転

■解 説
ロング型でのビット NOT
#prx NOT(&Hf)
FFFFFFF0

NO_PHASE

パルス発生　　予約定数

■書 式
NO_PHASE

■使い方
INSET NO_PHASE

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
無効

```
INSET NO_PHASE                  /* Counter disable
```

OFF

IO　　コマンド

■書 式
OFF arg1 [arg2 arg3 arg4 ...]

■使い方
OFF 1 2 3 /* MIO-1616etc
OFF A A+1
OFF -1 /* Memory I/O エリア

■機 能
出力ポートの OFF

■解 説
出力ポートを OFF します。オープンコレクタ出力がフロート状態になります。
負の値の場合 (-1 ～) は、メモリ I/O エリアのビットオフです。
2000 以上の場合 (2000 ～) は、CUnet エリアのビットオフです。
70000 以上の場合 (7aabb) は、MBK I/O エリア (タッチパネル R エリア) のビットオフです。
aa がバンク番号 (0 ～ 99) で bb はビット番号で 0 ～ 15 の値になります。

ON

IO　　コマンド

■書 式

ON arg1 [arg2 arg3 arg4 ...]

■使い方

```
ON 1 2 3   /* MIO-1616 etc
ON A A+1
ON -1   /* Memory I/O
ON 2000 /* CUnet Area
ON  70000 /* MBK I/O area
```

■機 能

出力ポートの ON

■解 説

出力ポートを ON します。オープンコレクタ出力がシンク状態になります。
負の値の場合 (-1 ～) は、メモリ I/O エリアのビットオンです。
2000 以上の場合 (2000 ～) は、CUnet エリアのビットオンです。
70000 以上の場合 (7aabb) は、MBK I/O エリア (タッチパネル R エリア) のビットオンです。
aa がバンク番号 (0 ～ 99) で bb はビット番号で 0 ～ 15 の値になります。

ON

マルチタスク　　関数

■書 式

ON(n)

■使い方

```
WAIT  ON(-1)==0
 PRINT  "WATSHI HA " TASKn
OFF  -1
'
IF ON(-1)==0 THEN
 PRINT  "WATSHI HA " TASKn
OFF  -1
END_IF
```

■機 能

メモリ IO のリード & セット (セマフォ)

■解 説

ON(n) はメモリ I/O および出力ポートに対して、ON コマンドと同様、ポートを ON します。
関数値として、ON する直前の指定ポートの値を返します。
ON(n) で n をメモリ I/O に指定すると、ポート n がセマフォとなります。
n を通常の出力ポート番号としても同様に使用できます。

```
-------------------------------------------------------
OFF  -1
 FOR  i=1 TO 10
  FORK  i *test
 NEXT
 END
*test
 WAIT  ON(-1)==0
 PRINT  "WATSHI HA " TASKn
 OFF  -1
 TIME  SYSCLK%1000
 GOTO  *test
```

ON_ERROR

制御文　　コマンド

■書 式

ON_ERROR arg

■使い方

ON_ERROR *USB
ON_ERROR VOID

■機 能

エラー処理ジャンプ先を定義

■解 説

コマンド、関数などでエラーが発生した場合、通常、実行中のプログラムが停止します。
ON_ERROR コマンドは、プログラムを停止させず、エラー処理プログラムを指定して、プログラムの実行を継続させます。方法は以下のとおりです。
ON_ERROR *label で飛び先を規定します。規定を解除する場合は、ON_ERROR VOID を実行します。
ON_ERROR はどのようなエラーでもエラー処理に制御を移してしまうため、
エラー処理プログラムでは、適切にエラーコードによって処理を分類する必要があります。
　通常、実行時にで発生するエラーは、致命的なものが殆どで、リトライ措置は不可能です。
この場合は、エラー場所と内容を外部に知らせてデバグに役立てます。
　しかし、USB メモリをアクセスするような場合は、接続デバイスの状態によりランタイム・エラーが発生することがあります。
この場合は、RST_USB など適切な処理により、デバイスを正常化してプログラムを再継続することができます。

ON_ERROR によるエラー処理ルーチンから、通常処理プログラムへの戻りには GOTO,RESUME を使用します。GOTO の場合は、エラー発生場所がサブルーチン中の場合、戻し場所に注意してください。
RESUME の場合は発生箇所に戻すため、コマンドをリトライする場合は、RESUME,
リトライさせずに次の処理に移るには、RESUME _NEXT と記述します。

エラーコードは、タスク変数、err_ に反映されます。err_ の値に適合した処理を記述します。
err_ は上位 1byte がエラーコード、下位 3byte がプログラム番号となっています。

```
err_>>24    --> エラーコード
ERR$(err_)  --> エラーメッセージ
err_&&HFFFFFF --> プログラム番号
```

エラー番号はエラーメッセージの末尾に表示されます。以下は USB メモリ関連のエラーコードです。

```
この USB は使用中です。:53
USB メモリがありません。:54
MRS-MCOM がありません。:55
USB メモリが動作異常。:56
```

```
FORK  1 *case1
 TIME  500
 FORK  2 *case2
 END
*case1
 ON_ERROR  *err1
 DO
  S_MBK  1 9000
  PRINT  10
  PRINT  20
```

```
 LOOP
*err1
 PRINT  "case1=" TASKn  err_&&H00FFFFFF ERR$(err_)  err_>>24
 TIME  1000
 RESUME  _NEXT
 END
*case2
 ON_ERROR  *err2
 DO
  OUT  1 -10000
  PRINT  1
  PRINT  2
 LOOP
*err2
 PRINT  "case2=" TASKn  err_&&H00FFFFFF ERR$(err_)  err_>>24
 TIME  1000
 RESUME
 END
```

ON_USB,OFF_USB

USB　　コマンド

■書 式

ON_USB
OFF_USB

■機 能

USB ポートのイネーブル・ディズエーブル

■解 説

メイン CPU ボードの USB ポートは、タスク 29 で、USB ファイルアクセスシステムを起動することによって使用可能となります。ON_USB は必要な初期化とタスク 29 の起動を行います。逆に OFF_USB は、ポートを停止し、タスク 29 も開放します。

OUT

IO　　コマンド

■書 式

OUT val port
OUT val port1,port2..
OUT val port1 TO port2

■使い方

OUT &H55 2
OUT &HAA -1
OUT 0 1,2,5
OUT 0 -1 TO -10

■機 能

出力ポート、メモリ I/O を 8bit パラレル設定します。

■解 説

出力ポートを 1byte 設定するコマンドで、バンク指定となります。
MPC-2000 の I/O ポート 0 ～ 7 はバンク 0、8 ～ 15 は、バンク 1 となります。
1 枚目の MIO-1616 は、同様にバンク 2、3 が割り当てられます。バンク設定を負の値とするとメモリ I/O となります。

アドレス値に ~Lng、~Wrd、~Int を与えると、それぞれロング書き込み、整数 2byte 書き込みとなります。

書き込みには Wrd,Int の区別はありません。タッチパネルの I/O エリア (R エリア) は 70000 以上を指定します。
ab は 00 ～ 99 となります。
OUT data 7ab00 /* byte 書き込み
OUT data 7ab00~Ub /* Hi-byte 書き込み
OUT data 7ab00~Wrd /* ワード書き込み
OUT data 7ab00~Lng /* ロング書き込み

また、出力ポートを複数同じ値に設定する場合は、出力ポート番号を続けて記述します。
連続して同じ値とするときは、port1 TO port2 という記述をします。

P$

文字列　　関数

■書 式
P$(val)

■使い方
a$=P$(100)

■機 能
点データの文字列化

■解 説
点データエリアは、"SETP n strngs" コマンドによって、文字列配列のように使用することができます。P$() 文字列として格納されたデータを取り出す関数です。

```
FORMAT   "Test s    "
 FOR   i=1 TO 10
a$="setp"+STR$(i-5)
  SETP   i a$
 NEXT
 FOR   i=1 TO 10
  PRINT   P$(i)
 NEXT
```

PALLET

パルス発生　　コマンド

■書 式
PALLET h P(i) P(j) P(k) [P(l)] m n * 0<=h <= 63 m,n ~32767
PALLET h P(i) P(j) m

■使い方
PALLET 1 P(1) P(2) P(3) P(4) 4 3
PALLET 1 P(1) P(2) P(3) 4 3
PALLET 1 P(11) P(12) 3

■機 能
パレット定義

■解 説
PALLET 1 P(1) P(2) P(3) P(4) 4 3
点 p(1) ～ P(4) で生成される 4 × 3 のパレット、4 点指定の場合は、変形四角形も可。

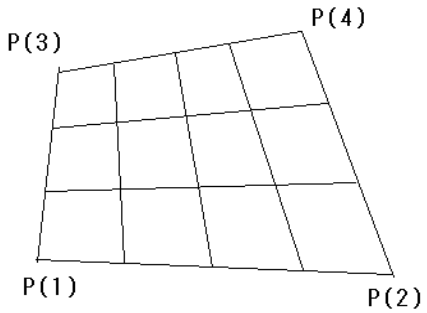
PALLET 1 P(1) P(2) P(3) 4 3
点 p(1) ～ P(3) で生成される 4 × 3 のパレット､3 指定の場合は､平行四辺形 (直方体) と扱われる。

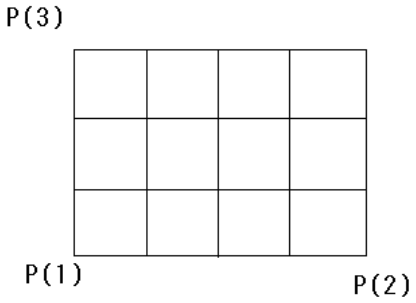
パレット上の点番は 1 〜となり、図の例では、P(1)-> 1,2,3,4,5 (p2)
という順になります。PL 関数の指定番号を正の数で与えると、6 は、p(1) 側の一個 p(3) よりとなります。指定番号を負の数にすれば、6 は P(2) のひとつ上となりジグザグ順序になります。

【一列パレットについて】
一列パレット P(1) --> P(2) の場合は以下のように記述します。12_48 以後
PALLET 1 P(1) P(2) m

12_48 以前では、以下のように記述してください。
PALLET 1 P(1) P(2) P(2) m 2
2 を指定する理由は、0 の除算を発生させないためです。

```
'4points teaching
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
SETP 3 0 20000 0 -5000
SETP 4 20000 20000 0 -5000
PALLET 1 P(1) P(2) P(3) P(4) 4 3
FOR   I_=1 TO 12
   JUMP   PL(1;I_)
   WAIT   RR(ALL_A)==0
NEXT
or
FOR   I_=-1 TO -12 STEP -1
   JUMP   PL(1;I_)
   WAIT   RR(ALL_A)==0
NEXT
------
'3points teaching
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
SETP 3 0 20000 0 -5000
PALLET 1 P(1) P(2) P(3) 4 3
FOR   I_=1 TO 12
   JUMP   PL(1;I_)
   WAIT   RR(ALL_A)==0
NEXT
------
'linear tray
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
PALLET 1 P(1) P(2) 4
FOR I_=1 TO 4
  JUMP PL(1;I_)
  WAIT RR(ALL_A)==0
NEXT
```





PAUSE

マルチタスク　　コマンド

■書 式
PAUSE arg
PAUSE (STP_D,taskn)

■使い方
PAUSE n

■機 能
タスクの一時停止
引数が、(STP_D,task) の場合 タスクの一時停止と停止コマンドの実行

■解 説

動作中のタスクを SLEEPING 状態 (無限タイマー停止) にします。CONT で再開します。
引数を (STP_D,task) のようにすると、対象タスクを停止するとともに、STOP STP_D を実行し、なおかつ、JUMP,JMPZ コマンドに対して、再実行フラグをたてます。
この場合、CONT コマンドにより再開されたタスクは WARP,JUMP,JMPZ コマンド実行中であれば、再実行措置をとります。ただし WARP のハンド操作機能は、PAUSE のかかった時点で有効無効が不明となるため、再開は要注意。

PEEK

文字列 関数

■書 式

PEEK(Str$+n)

■使い方

A=PEEK(b$+1)
B=PEEK(b$+LEN(b$)-1)

■機 能

文字列コードの取得

■解 説

PEEK は指定文字列の指定位置のコードを取得することができます。
通信用チェックサム計算などに役立ちます。

```
----------------------------------------
10      a$="123456789A"
20       PRX   PEEK(a$)
30       PRX   PEEK(a$+LEN(a$)-1)
#run

00000031
00000041
#
```

PG

パルス発生 コマンド

■書 式

PG arg1 [taskn]
PG

■使い方

PG 0
PG 1 2

■機 能

MPG ボードの指定
MPG 0 ～ 9 が、MPG-2314 高機能 円弧補間まで可
MPC-1200 は PG 17

■解 説

PG コマンドは MPG コマンドと同機能ですが、(MPG コマンド参照)MPG ボードの存在テストをしません。このために装備していない MPG を指定してもエラー表示しません。

PGA,PGB

パルス発生　　コマンド

■書 式

PGA str$ val

■使い方

PGA "G" 1000
PGB "V"
pr V_PGB

■機 能

MPC-1000,MPC-N816 の PG 制御コマンド

■解 説

MPC-1000,MPC-N816 には、PGA と PGB の 2 つの簡易 PG 機能があります。
それぞれの PG を制御するコマンドが PGA,PGB で、書式と機能は以下のとおりです。
ここでは、PGA の例ですが、PGB も同一の書式で使用出来ます。

PGA "G" pps ; PPS 指定パルス発生 (20 ～ 9000pps)
PGA "S" pps ; パルスレート設定 (20 ～ 9000pps)
PGA "W" duty ; PWM(40 ～ 970/1000)
PGA "P" pls ; パルス数指定パルス発生 (+/-8000000)
PGA "A" pps ; 加減速テーブル生成 (500 ～ 12000pps)
PGA "F" f ; 速度選択 (10 ～ 0)
PGA "R" pls ; 加減速パルス発生 相対 (+/-8000000)
PGA "M" pos ; 加減速パルス発生 座標 (+/-8000000)
PGA "H" pos ; 現在位置設定 (+/-8000000)
PGA "D" n ; パルス方式 (0: デフォルト 2PLS 1: 方向指示)
PGA "C" ; 現在位置取得
PGA "V" ; バージョン取得

"PGA C" および "PGA V" コマンド発行後のリターン値は、V_PGA に代入されます。
(PGB の場合は、V_PGB)
なお、パルス発生は、それぞれ、OFF PGA,OFF PGB で停止することができます。

PGE

パルス発生　　関数

■書 式

PGE(0)
PGE(axs,val)

■使い方

IF PGE(X_A,ALM) THEN : GOTO *EMG_X_A : END_IF
IF PGE(0) THEN : GOTO *EMG : END_IF
IF PGE(X_A,CLR_ER|ALM) THEN : GOTO *EMG_X_A : END_IF
IF PGE(CLR_ER) THEN : GOTO *EMG : END_IF

■機 能

MPG-2314 の停止原因の参照

■解 説

MPG-2314 は EMG,ALM,LMT,IN0 ～ IN1 の各入力によってパルス発生を停止させられる。
停止後、原因入力が解除されても、PGE() 関数で、停止原因を知ることができる。
引数にはふたとおりある。PGE(0) の場合、4 軸すべての停止原因フラグを参照できる。
PGE(0) = {Uaxs|Zaxs|Yaxs|Xaxs} で 4byte すべてに意味がある。
各 byte のビット構成は、{EMG,ALM,LMTn,LMTp|IN3,IN2,IN1,IN0} の 8bit です。

もうひとつの方法は、軸指定とビット指定定数によりエラーを個別にチェックする方法です。
PGE(X_A,LMTp) LMTp をテスト
PGE(X_A,(IN1|LMTp)) LMTp、IN1 の双方をテスト
ビット指定を 0 とすると、指定軸のエラー情報を byte で返します。

尚、引数に CLR_ER のみをセットすると、全ステータスの取得と同時にエラーステータスをクリアします。
また、軸指定の場合、ビット条件に CLR_ER を OR しておくと、該当軸のみリード & クリアとなります。

```
LIST
10      'XXXX=CLR_ER
20      'XXXX=(X_A,CLR_ER|IN0)
50        PG   1
60        ACCEL   4000
70        STOP   ALL_A IN0_ON
80        OFF   0 1
90        CLRPOS
100       MOVS   1000000 1000000 100000 100000
110       TIME   1000
120       ON   0 1
130       WAIT   RR(X_A)==0
140       PRX   PGE(0)
150       PRX   PGE(XXXX)
160       PRX   PGE(0)
#run

00000101
00000001
00000100
#prx XXXX
0001F100
#
```

PG_TASK0

パルス発生　　予約変数

■書 式
PG_TASK0

■使い方
print PG_TASK0

■機 能
PG 番号取得

■解 説
タスク 0 に割り当てれている PG 番号を返す変数。存在しない PG の場合は -1 となります。

```
/* USE MPG-2314 #0
10        PG   0
20        PRINT   PG_TASK0
30        PG   10
40        PRINT   PG_TASK0
50        PG   1
60        PRINT   PG_TASK0
#run

 0
 10
-1
```

PHASE1

パルス発生　　予約定数

■書 式
PHASE1

■使い方
INSET PHASE1

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
カウンターをエンコーダ入力モードとし、カウント倍率は無しとする。

```
INSET PHASE1                    /* multiplier: 1 time
```

PHASE2

パルス発生　　予約定数

■書 式
PHASE2

■使い方
INSET PHASE2

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
カウンターをエンコーダ入力モードとし、カウント倍率を 2 逓倍とする。

```
INSET PHASE2                    /* multiplier: twice
```

PHASE4

パルス発生　　予約定数

■書 式
PHASE4

■使い方
INSET PHASE4

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
カウンターをエンコーダ入力モードとし、カウント倍率を 4 逓倍とする。

```
INSET PHASE4                    /* multiplier: 4 times
```

PL

パルス発生　　関数

■書 式

PL(n;m)

■使い方

MOVS PL(1;10)
JUMP PL(2;100)

■機 能

パレット点を計算して MOVS 等の移動コマンドでその点データを引き渡す。

■解 説

PALLET 1 P(1) P(2) P(3) P(4) 4 3
JUMP PL(1;I)
パレットコマンドを実行した後で用います。パレットは 0 ～ 63 指定できます。
パレット番号と、パレット点の区切りは、";" であることに留意してください。
"," で区切るとパレットを正しく選択できません。
引数を負にすると ZIG_ZAG 順になります。

PLIST

パルス発生　　コマンド

■書 式

PLIST arg1

■使い方

PLIST
PLIST 10

■機 能

点データの表示

■解 説

点データの連続表示です。20 点ずつ表示してタイプイン待ちとなります。
'q' で終了。それ以外のキィーでは継続となります。

```
#plist
P(1)    X= 200  Y= 0  U= 0  Z= 0
P(2)    X= 0  Y= 0  U= 0  Z= 0
P(3)    X= 0  Y= 0  U= 0  Z= 0
P(4)    X= 0  Y= 0  U= 0  Z= 0
P(5)    X= 0  Y= 0  U= 0  Z= 0
P(6)    X= 0  Y= 0  U= 0  Z= 0
P(7)    X= 0  Y= 0  U= 0  Z= 0
P(8)    X= 0  Y= 0  U= 0  Z= 0
P(9)    X= 0  Y= 0  U= 0  Z= 0
```

POKE

文字列　　コマンド

■書 式

POKE arg1 arg2 .. (str$+n)

■使い方

POKE &H03 (a$+0)
POKE &H41 42 (a$+5)

■機 能
文字列データ変更

■解 説
文字列中のコードを指定コードに置き換えます。
指定コードは、NULL、そのほかのバイナリコードも入力できるため、CRC などのバイナリデータの設定が簡単にできます。
引数は、8 個まで設定することができ、最後の引数を文字列と文字位置の指定に用います。
文字列と文字位置指定は、(a$+n) というように () で閉じます。
この場合は、a$ の n 番目からという意味になります。

```
LIST
10      a$="1234567890"
20       POKE   &h0041 &h0042 &h0050 &h0051 (a$+3)
30       PRINT   a$
#run

 123ABPQ890
#
```

POST

CUnet コマンド

■書 式
POST dst ary

■使い方
POST 2 P(100)
POST 5 MBK(20)
POST -2 MBK(100)

■機 能
CUnet 経由でのデータの転送

■解 説
CU_POST が起動されている相手に対してデータを転送します。
POST コマンド一回の転送単位は 240byte です。
点データでは、15 point (240byte/16byte)
MBK データでは、120 個 (240byte/2byte)

【例】
POST 2 P(100)
SA=2 のステーションに対して、P(100) 〜 P(114) のデータが転送されます。

POST 3 MBK(20)
SA=3 のステーションに対して、MBK(20) 〜 MBK(139) のデータが転送されます。

また、dst を負の数にすると、データ転送を要求します。(注)SA0 に対する要求は 0 のかわりに 64 を設定します。)
この場合、自己側でも CU_POST が起動されていなければなりません。応答が帰ってくるまでこのコマンドは待ち状態になります。2 秒以内に応答が無いと CUM_ERR の BIT6 がセットされます。

【例】
POST -3 MBK(20)
SA= のステーションに対して MBK(20) 〜 MBK(139) のデータを要求し、自分の同じエリアに書き込みます。

このコマンドにより、CUnet を備えた MPC 間で、点データや MBK データのデータの共有を行うことができます。
しかし、応答速度は 0.1 秒～ 0.5 秒です。リアルタイム性はありませんので、高速共有は、CUnet のメモリ I/O で行ってください。

送信が正常に完了したかどうかは、CUM_ERR を参照します。
送信エラーとなると、BIT7 が 1 となり、詳細は、BIT0 ～ BIT6 に反映されます。

CUM_ERR
BIT7: MAIL SEND ERROR
BIT6: 転送要求の応答が無い
BIT5: 通信停止
BIT4: 送信タイムアウト不正 (通常 0)
BIT3: 送信ブロック不正 (通常 0)
BIT2: 送信タイムアウト発生
BIT1: 送信相手不在
BIT0: 送信相手が受信待機となっていない

POST 3 MBK(20)
IF CUM_ERR!=0 THEN : PRINT "X_ERR" CUM_ERR : END : END_IF
SA=3 のステーションに対して、MBK(20) ～ MBK(139) のデータが転送され、正常に送信終了したかどうか確認します。

"POST n" のように、データを指定しないと、相手側と自己の CU_POST が起動されているかどうかの確認できます。正常であれば、"Ok" と表示されます。

【サンプルプログラム】
MPC-A と MPC-B が CU-net で連結され、A 側にのみタッチパネルが接続されたシステムを想定しています。この時、MBK(1000) ～を MPC-A の操作画面、MBK(2000) ～を MPC-B の操作画面とします。

```
/*MPC-A
        CUNET 2 2 32
        MEWNET 38400 2
        CU_POST
        FORK 1 *SHARE_MBK
        END
*SHARE_MBK
        DO
        POST 4 MBK(2000) /*  MPC-B OUT AREA
        POST -4 MBK(2200) /* MPC-B IN AREA
        TIME 100
        LOOP

/*MPC-B
        CUNET 4 2 32
        CU_POST
        END
```

POS_L

パルス発生　　予約定数

■書 式
POS_L

■使い方
HOME POS_L POS_L POS_L POS_L

■機 能
正の大数

■解 説
対象ボード : MPG-2314
原点復帰ニアオリジンの移動量を大きくとりたい場合は、POS_L,NEG_L を用いてください。
これらは、正・負の 3byte 長の最大数です。
#prx POS_L
007FFFF0
#prx NEG_L
FF80000F

```
HOME POS_L POS_L POS_L POS_L
```

PRA

保守 コマンド

■書 式
PRA array(n)
PRA var_ [n]

■使い方
PRA J_
PRA J_ n
PRA AHO(10)
PRA FOOL(10,1)

■機 能
配列値の表示
タスクローカル変数の表示

■解 説
タスクローカル変数を表示します。第二引数で、タスク番号を指定できます。
指定されない場合は、すべてのタスクの変数価を表示します。

配列の中身をまとめて表示します。配列要素を 20 個ずつ表示します。
二次元配列にも対応します。

PRINT

保守 コマンド

■書 式
PRINT [val,str]

■使い方
PRINT "res=" a$ a cc bb$ a a a$ "123abc"

■機 能
数値 文字列の表示 デバッグ用

■解 説
変数や文字列を表示するコマンドです。
プログラム中に入れて状態監視に使用します。

```
10          a$="123"
15          bb$="koatae"
20          a=456 :  cc=1096
30           PRINT   "res=" a$ a cc bb$ a a a$ "123abc"
```

```
#run
res= 123 456 1096 koatae 456 456 123 123abc
#
```

PRINT#

通信　　コマンド

■書 式

PRINT# [COM#] [Options] arg1 arg2 ...

■使い方

PRINT# 1 a$ "123¥n"
PRINT# 5 COMPOWAY snd$
PRINT# 3 STR_LEN|32 a$

■機 能

通信ポートへ出力

■解 説

PRINT# はシリアルポートへの出力です。
最初に引数が数値である場合、その数値は RS-CH 番号指定となります。
出力引数として 文字列、文字列変数、変数などが使えます。
PRINT# "count=" i_ " " i_*i_

なお、PRINT# では引数の出力間にスペース挿入はありません。
また、引数の上で、+ による文字列結合はできませんが、
以下のように、引数を羅列するだけで、文字列結合して出力するのと同じことができます。
PRINT# CHR$(1) "DATA" CHR$(3)

よって以下と同じ結果になります。
b$=CHR$(1)+"DATA"+CHR$(3)
PRINT# b$

・固定長出力オプション STR_LEN
文字列出力は、通常 NULL ターミネーションされます。しかし、バイナリコードを含む
固定長文字列出力が必要となる場合があり、STR_LEN オプションはこうした場合に用います。
a$="1234567" : b$="abcdfge"
print# STR_LEN|4 a$ b$
この場合、出力されるのは、1234abcd となります。

・NULL コードを含む文字列の出力
NULL コード、つまりアスキーコード 0 は、通常文字列のターミネータと扱われており、
通常の方法では出力されません。

1) 簡単に 0 ～ 4 のコードを出力するには、文字列定数中に ¥0 ～ ¥4 を記述します。
PRINT# "ABC¥0DEF" --> ABC~00DEF ABC と DEF の間に 00 コードが出力されます。

2) チェックサムを出力する方法 1
例えば 16bit のチェック・サムを出力する場合は以下のようにします。
HI=CHK_SUM>>8
LO=CHK_SUM&255
PRINT# STR_LEN|2 CHR$(HI) CHR$(LO)

3) チェックサムを出力する方法 2
文字列の固定パケツトに直接バイナリコードを埋め込む方法です。

```
CMND$="CMNDEXE"
SUM=0
FOR i=0 TO LEN(CMND$)-1 : SUM=SUM+PEEK(CMND$+i) : NEXT
HI=SUM>>8
LO=SUM&255
POKE 0 HI LO (CMND$+7)
PRINT# STR_LEN|10 CMND$
```

*POKE コマンドは、直接メモリにデータを書き込みます。記述に誤りがあると、誤作動やプログラム破壊を引き起こします。
注意して使用してください。

【Options について】
COMPOWAY:
定数 COMPOWAY を与えると、OMRON COMPOWAY フォーマットで文字列を出力します。
転送する文字列はコマンド COMPOWAY であらかじめパケット化しておきます。

STR_LEN:
定数 STR_LEN に転送文字数を OR すると (例 :STR_LEN|32) 文字列出力は、
ヌルターミネータは無視され指定された転送文字数 出力します。
ヌルコードを含む転送に使用します。
ヌルを含む文字列の作成には、コマンド、ADD_STR を使用します。

```
-- Examples--
 PRINT# 1 "ABC¥r"          /* Xmit  "ABC[CR]" through CH1
 PRINT# 1 "ABC¥n"          /* Xmit  "ABC[LF]" through CH1
 PRINT# 1 "ABC¥r¥n"        /* Xmit  "ABC[CR][LF]" through CH1
 PRINT# 1 "ABC¥tDEF"       /* Xmit  "ABC[TAB]DEF" through CH1

 ¥r=[CR]=&H0D
 ¥n=[LF]=&H0A
 ¥t=[TAB]=&H09

-- An example of COMPOWAY--
  COMPOWAY node_no sub_adr sid cmnd_txt$ snd$
  PRINT# 5 COMPOWAY snd$
```

PRX

保守　　コマンド

■書 式
PRX val

■使い方
PRX A

■機 能
ヘキサ形式表示

■解 説
数値をヘキサ形式で表示します。デバッグ用コマンドです。プログラム中で文字列とし HEX 表現が必要な場合は HEX$() を用います。

```
A=100:B=1000:C=10000
prx A B C
00000064 000003E8 00002710
#
```

PR_CHK

パルス発生　　予約定数

■書 式
PR_CHK

■使い方
RANGE PR_CHK|X_A 10000 -10000

■機 能
移動先チェック

■解 説
対象ボード : MPG-2314
PR_CHK 指定をしておくと、リミット値を越えるかどうかが、事前に判断され、超える場合は動作以前にエラー停止となります。PR_CHK の無いソフトリミット指定は、リミットを越えた時点でスローダウン停止となるため、減速距離分のオーバーシュートが発生します。

```
RANGE PR_CHK|X_A 10000 -10000
RANGE PR_CHK|Y_A 11000 -10000
RANGE PR_CHK|Z_A 12000 -10000
```

PR_LCD

文字列　　コマンド

■書 式
PR_LCD string

■使い方
PR_LCD DD$
PR_LCD "ERR"

■機 能
文字列をに表示

■解 説
与えられた文字列を 7Seg に表示します。表示可能な文字は、0 ～ 9,A ～ Z と一部の記号です。
小文字や複雑な文字は表示できません。(7Seg 搭載機種のみ)

PTR$

文字列　　関数

■書 式
PTR$(m)

■使い方
ptr_=a$
ptr_=ptr_+10
k$=PTR$(5)

■機 能
ポインタの位置から m 文字

■解 説
ボインタの位置から m 文字の文字列を取り出します。
文字列の中から必要とする文字列の切り出しが容易にできます。
ポインタ位置は ptr_ に反映される為、この値を操作する事によって文字列の切り出し位置を調整できます。ptr_ は、ptr_=a$ や、SERCH コマンドで初期化されます。

例 1) 場所・文字数が予め判明している場合の、PTR$() の使用方法
例 2) 1 つの文字列になっている、 2 つの数値を、それぞれ独立した文字列に分解する方法です。
ポインタの位置の差を文字列の長さとして使用しているのに注意してください。

```
1)
FORMAT ""                          /* 文字列書式設定クリア
TT$=HEX$(TIME(0))                  /* 時分秒取得
ptr_=TT$                           /* 文字列の位置を取得
ptr_=ptr_+2                        /* ポインタ再設定
HH$=PTR$(2)                        /* ポインタの位置から 2 文字切り出し
ptr_=ptr_+2
MM$=PTR$(2)
ptr_=ptr_+2
SS$=PTR$(2)
CL$=HH$+":"+MM$+":"+SS$            /* 文字列連結
PR "(1)" TT$ "->" CL$              /* TT$: 元の文字列 CL$: 合成後の文字列

#RUN
(1) 00123400 -> 12:34:00
#

2)
 C41$="Mx+9.7042e+002 My+6.3210e+002"
' Serching the space position
 a_=C41$
 l_=LEN(C41$)
  SERCH   C41$ " "
 b_=ptr_
'b_ is the space position.
 a_=ptr_-a_
 ptr_=C41$
 C1$=PTR$(a_)
 ptr_=b_
 C2$=PTR$(l_-a_)
  PRINT   C1$
  PRINT   C2$
#RUN
 Mx+9.7042e+002
 My+6.3210e+002
#
```

ptr_

文字列　　予約変数

■書 式

ptr_

■使い方

ptr_=a$

■機 能

文字列ポインタ

■解 説

タスク変数。文字列内の位置を示します。

```
10      a$=HEX$(DATE(0))
20       PRINT   a$
30      ptr_=a$                    /* set the pointer position
40      y$=PTR$(4)                 /* copy 4 characters
50      ptr_=ptr_+4                /* re-set the pointer position
```

```
60      m$=PTR$(2)
70      ptr_=ptr_+2
80      d$=PTR$(2)
90       PRINT   y$ m$ d$
RUN

 20081117                          /* a$
 2008 11 17                        /* y$ m$ d$
```

PULSE_OUT

IO　　コマンド

■書 式

PULSE_OUT port# interval [count]

■使い方

PULSE_OUT 0 10 10
PULSE_OUT 0 10
PULSE_OUT VOID
PULSE_OUT 32767

■機 能

出力ポートの自動オンオフ

■解 説

出力ポートを自動でオンオフします。
count を指定すると指定回数 ON/OFF の後 OFF となります。
inetrval は 0.1 秒単位設定。
コマンドが実行されると内部カウンタが初期化され、指定時間後 (Interval/2)、最初に ON コマンドが実行されます。このため予め指定ポートを ON にしておくと、interval 後に OFF となります。

ON/OFF 中のポートを停止するには、interval を 0 にします。この時指定ポートは OFF にします。

PULSE_OUT 0 10 ではポート 0 を 1 秒間隔で ON/OFF。
PULSE_OUT 0 0 で ON/OFF を停止します。

PULSE_OUT VOID は、すべての設定された PULSE_OUT をキャンセルします。
PULSE_OUT 32767 は すべての設定された PULSE_OUT の動作を同期させます。

PWM

IO　　コマンド

■書 式

PWM portn k

■使い方

PWM 15 A

■機 能

PWM パルス発生

■解 説

指定された出力ポートを PWM オンオフします。
PWM 周期時間は 50msec です。与えられた k msec だけポートをオンします。
発熱体や、ペルチェ素子の電力制御に用います。

*PWM は Pulse Width Modulation の略式表現です。

QUIT

マルチタスク　　コマンド

■書 式
　QUIT arg1 arg2 arg3..

■使い方
　QUIT 1
　FOR I=1 TO 4 : QUIT I : NEXT

■機 能
　タスクの停止

■解 説
　FORK によって起動されたマルチタスク・プログラムを停止する。

QUIT_FORK

マルチタスク　　コマンド

■書 式
　QUIT_FORK n *LABEL

■使い方
　QUIT_FORK 1 *LABEL

■機 能
　タスク起動

■解 説
　FORK と同じ機能ですが、相手タスクが既に起動されていてもエラーを発生しません。

RAD

浮動小数点　　関数

■書 式
　RAD(v)

■使い方
　FP(0)=SIN(RAD(45))

■機 能
　ラジアン変換

■解 説
　角度を度からラジアンに変換します。Πを得るために、RAD(180) としても使用できます。

```
-----------------------------------------
#FP(0)=RAD(180)
#FP(1)=TAN(RAD(45))
#pr FP(0) FP(1)
 3.141593E+00 1.000000E+00
#
```

RANGE

パルス発生　　コマンド

■書 式
　RANGE axis pos_limit neg_lmit

■使い方

RANGE X_A 10000 -10000
RANGE X_A|Y_A 20000 0
RANGE X_A|PR_CHK 1000 -1000
RANGE VRING|X_A 1000

■機 能

可能動作領域の設定

■解 説

RANGE はそれぞれの軸に対して、ソフトリミットとなる限界値を設定します。
RANGE コマンドは内部のレジスタに値を設定するのみです。
この値によるソフトリミットを有効にするには、INSET コマンドの引数に SLMT_ON を追加します。
例 :INSET X_A XXXXX|SLMT_ON

axis 指定に定数 PR_CHK を OR すると PtoP 制御コマンド (RMVS,MOVL,JUMP など) に対して移動先座標チェックが行われます。移動先が指定範囲内にない場合は、エラー表示、停止します。
移動先チェックは、RMVC,RMVT 等のコマンドには無効です。SLMT_ON と併用してください。

axis 指定に定数 VRING を OR すると、内部の位置カウンタがリングカウンタになります。
リングウンタとは、回転軸の位置管理などに用います。
VRING を指定するとソフトリミットは無効です。

```
10 PG 1
20 RANGE X_A|Y_A 200000 -1000
30 RANGE Z_A|PR_CHK 1000 -1000
40 INSET X_A|Y_A LMT_ON|SLMT_ON
```

RCV

CUnet 関数

■書 式

RCV(arg)

■使い方

A=RCV(A$)
A=RCV(P(100))
A=RCV(DAT(10))

■機 能

メール受信

■解 説

RCV 関数は、XMT 関数と対で使用する、メール受信関数です。CU_POST,POST とは併用できません。
引数に P(n),X(n) ～ Z(n),MBK(n), 配列、文字列を指定することができ、受信した 256byte のデータを自動的に指定場所に格納します。
指定された時間以内 (デフォルトは 10 秒) にメールが受信されないと、-3 を返します。
タイムアウト時間の変更は、timer_ に指定時間 (0.1 秒単位) を設定してから、RCV() を実行します。
-2 は、CUM_ERR にエラー・コードがはいっている場合、-1 は引数の指定が間違っている場合です。正常に受信すると、受け取ったメールの送信元の番号を返します。

```
–MPC A side–
LIST
10        CUNET   0 4 31
20        DIM   a(100)
30        FILL   a(0) 0
40        TIME   100
50       CUM_ERR=0
```

```
60       a$="1234567890"
70        IF   XMT(8,a$)!=0 THEN  : END   : END_IF
80        IF   RCV(a(1))!=8 THEN  : END    : END_IF
90        PRINT   a(1) a(2) a(3) a(63) a(64)
#run

 10 20 30 630 640
#
–MPC B side–
LIST
10        CUNET   8 4 31
20        DIM   b(100)
30        TIME   100
40       CUM_ERR=0
50        IF   RCV(b$)!=0 THEN  : END   : END_IF
60        PRINT   b$
70        FOR   i=1 TO 64 :  b(i)=i*10 : NEXT
80        IF   XMT(0,b(1))!=0 THEN  : END   : END_IF
90        END
#run

 1234567890
#
```

RENUM

編集　　コマンド

■書 式

RENUM [n]

■使い方

RENUM
RENUM 5

■機 能

文番号の振り替え

■解 説

文番号を 10 ごとに振りなおします。数を指定するとその数で振りなおします。

RESUME

制御文　　コマンド

■書 式

RESUME [arg]

■使い方

RESUME
RESUME _NEXT

■機 能

エラー処理から戻る

■解 説

ON_ERROR からの戻り処理です。
RESUME は発生箇所に戻すため、コマンドをリトライする場合は、RESUME,
リトライさせずに次の処理に移るには、RESUME _NEXT と記述します。
ON_ERROR を参照してください。

RETURN

制御文　　ステートメント

■書 式

RETURN [arg1,arg2..]

■使い方

```
GOSUB *LABEL

*LABEL
 RETRUN
*LABEL
 RETURN aho
```

■機 能

サブルーチンから戻る。また、引数を GOSUB を実行した側に戻すことができる。

■解 説

サブルーチンから GOSUB 呼び出しをしたプログラムにもどる。
GOSUB で呼ばれたプログラムは必ず RETURN で戻らなければならない。
また、RETURN に引数を与えると、結果を親プログラムに戻すことができる。

```
10 GOSUB *CAL 300 400
20 _VAR RES
30 PR RES
40 END
50 *CAL
60 _VAR V_ W_
70 RETURN SQR(SQ(W_)+SQ(V_))
RUN
*
Compiling
_________
500
#
```

RMVC

パルス発生　　コマンド

■書 式

RMVC axis arg

■使い方

```
RMVC X_A CW
RMVC Y_A CCW
```

■機 能

量を指定しないパルス発生。CW,CCW は方向指定。+1,-1 でも可。

■解 説

arg で正の数を指定すると CW 方向、負の数を指定すると CCW 方向のパルス発生となります。

RMVL

パルス発生　　コマンド

■書 式

RMVL arg1 [arg2,arg3,arg4]

■使い方
RMVL x y 0 0
RMVL x y 0 z
RMVL 0 y u z

■機 能
直線補間でパルス発生する。

■解 説
3 軸までの直線補間でパルス発生をします。4 軸指定するとエラーになります。
補間での速度は、X>Y>Z>U の順序で有効軸の速度が使われます。
RMVL 0 y u z
であれば、yuz の直線補間となり、速度は y 軸の速度が使われます。

RMVL X_A|Y_A 10000 というような軸指定の記述はできません。

RMVS

パルス発生 コマンド

■書 式
RMVS [axis] n
RMVS X [Y,U,Z]

■使い方
RMVS X_A n
RMVS x y u z

■機 能
指定量のパルスを発生します。

■解 説
加減速つきの相対パルス発生命令です。正の値で CW 方向。負の値で CCW 方向です。

RMVS X_A|Y_A 10000 というような軸指定の記述はできません。

RMVT

パルス発生 コマンド

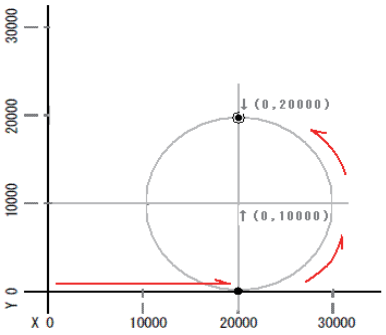
■書 式
RMVT axs arg1 arg2 [CCW|CW|0 cent1 cent2]

■使い方
RMVT X_A|Z_A 20000 0
RMVT X_A|Z_A 0 20000 CCW 0 10000

■機 能
連続補間移動

■解 説
相対座標による連続補間コマンドです。第 3 パラメータに CCW か CW を与えると円弧補間となります。この場合は、円弧中心を指定するパラメータが必要となります。
いずれの座標値もコマンドが実行される場所からの相対座標となります。
第 3 パラメータが無いか、0 とすれば、直線補間となります。
例では、X,Y の円弧補間を実施します。
EN_DACK,DS_DACL は、減速の有効・無効です。
図は RMVT X_A|Y_A 0 20000 CCW 0 10000 の
実行イメージです。

```
PG  0
  ACCEL  8000
  CLRPOS
  DS_DACL
  RMVT  X_A|Y_A 20000 0
  RMVT  X_A|Y_A 0 20000 CCW 0 10000
  RMVT  X_A|Y_A 0 -20000 CCW 0 -10000
  RMVT  X_A|Y_A 10000 0
  EN_DACL
```



RR

パルス発生　　関数

■書 式

RR(arg1)

■使い方

WAIT RR(X_A)==0
WAIT RR(ALL_A)==0
IF RR(X_E)!=0 THEN

■機 能

MPG の動作状態の監視

■解 説

RR(arg1)
は、ステータスと arg1 で AND をとった値を返します。(arg1 & ステータス)
ただし、arg1 が 0 の場合は AND をしないで、動作ステータスをそのままよみとって返します。
通常は、 WAIT RR(X_A|Y_A)==1
のように動作軸の停止を監視します。
MPG-2314 に対するステータス読み取りでは、上位 4bit が各軸のエラー状態となります。
X_E ～ U_E,ALL_E という予約定数が対応します。
* ステータスとは MCX-314As の RR0 レジスタ

RR3

パルス発生　　関数

■書 式

RR3(axis)

■使い方

A=RR3(X_A)

■機 能

MPG-2314 の割込みフラグ読み取りおよびフラグ解除

■解 説

MPG-2314 で割込みを設定すると、割り込み発生レジスタ RR3 の内容に従い、INT 割り込みが発生します。
割込みの解除は、割り込み条件の解除と、この RR3 レジスタの読み取りが必要になります。
RR3 レジスタは各軸に独立に用意されているため、その読み出しには軸指定が必要です。
このため、RR3(ALL_A) というような記述はできません。
必ず、RR3(X_A),RR3(U_A) というように、単 1 軸を指定して読み出します。
得られた値 (1byte) の意味は下記のとおりです。
bit7:D_END bit6:C_STA bit5:C_END bit4:P>=COMP+ bit3:P
MPC-2000 で対応しているのは、bit4,bit3 のみで他の割込みは発生しません。

RS

保守 コマンド

■書 式

RS ch

■使い方

RS 1

■機 能

受信バッファ表示

■解 説

コマンドを実行すると、以下のように 256byte 受信バッファを表示します。
末尾が最も最近受け取った文字となります。
">" が現在の未読文字位置をしめしています。INPUT ではここから読み始めます。

```
#rs 1

The '>' shows the 1st chr-byte
 5   -CR [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 >1   1   1   1   3   3   3   n   m   n   m   n   m   CR ~
#
```

rse_

通信 予約変数

■書 式

rse_

■使い方

pr rse_

■機 能

通信エラーステータス

■解 説

タスク変数
通信時のエラー内容を表します。

```
10      CNFG#   1 "9600b8pns1NONE"
20      INPUT#   1 TMOUT|5 a$          /* timeout 5 sec
35      IF   rse_==1 THEN              /* check timeout
36       PRINT   "timeout"
37       ELSE
38       PRINT   a$
40      END_IF
#RUN

 timeout                               /* fail
#RUN

 asdfg                                 /* success
#
```

RST_USB

USB　　コマンド

■書 式

RST_USB

■使い方

RST_USB : TIME 2000

■機 能

USB メモリプロセスの初期化

■解 説

USB 動作エラー発生の場合 (エラー 56,68) は、RST_USB コマンドで USB メモリと USB メモリプロセスを初期状態に戻し、再度、同じ処理を繰り返す。

```
ON_ERROR *ERROR_PROC
ON_USB
TIME 2000
FILE$=HEX$(DATE(0))+".CSV"
USB_DEL FILE$                  /* Delete a file
DO
  FORMAT "00:00:00"
  WRITE_STR$=HEX$(TIME(0))+"¥n"
  USB_WRITE WRITE_STR$         /* Write to the USB Memory
  TIME 5000
LOOP

*ERROR_PROC                    /* Error process
  err_code=err_>>24            /* Get an error code
  err_step=err_&&H00FFFFFF     /* Get an error step number
  PR "ERROR CODE" err_code "ERROR STEP" err_step
  WAIT SW(192)==1              /* Restart
  TIME 1000
  RST_USB                      /* Reset the USB memory
  TIME 2000
  RESUME
```

RUN

制御文　　ステートメント

■書 式

RUN arg1

■使い方

RUN
RUN *LABEL
RUN 900

■機 能

プログラム実行

■解 説

デバッグ時のプログラム実行です。
プログラムをコンパイルの後、フラッシュ ROM に保存して実行します。
プログラムがすでにコンパイル済みの場合はただちに実行します。
初期状態の MPC に既存のプログラムを読み込んで稼動する際にも、読込後 RUN を実行してください。
プログラミングケーブル未接続状態で電源を入れると自動実行になります。

SA

CUnet　関数

■書 式
SA(val)

■使い方
ON SA(5)+0

■機 能
CUnet の SA に対応した ON/OFF/SW 番号を得る

■解 説
CUnet のステーションアドレスと、ON/OFF 番号を関連づける関数。
SA5 の 最初の ON/OFF 番号は SA(5) となる。

SA0_B~SA63_B

CUnet　予約定数

■書 式
SA0_B~SA63_B

■使い方
IN(SA0_B)

■機 能
CUnet SA 番号

■解 説
CUnet のステーションアドレスに対応した I/O バンク番号。

SA0~SA63

CUnet　予約定数

■書 式
SA0~SA63

■使い方
ON SA0+5

■機 能
CUnet SA 番号

■解 説
CUnet のステーションアドレスに対応した I/O 番号。

SA_B

CUnet　関数

■書 式
SA_B(val)

■使い方
OUT &H55 SA_B(5)

■機 能
CUnet の IN/OUT バンク番号を得る。

■解 説

CUnet のステーションアドレスと、IN/OUT 命令のバンク番号を関連づける関数。
SA5 の最初のバンクは SA(5) となる。

SEC

時間管理　　コマンド

■書 式

SEC MBK(n)
SEC n h m s

■使い方

```
SEC   MBK(7000)
 SEC   7 17 20 2
 SEC   8 0
```

■機 能

1 秒カウンタの初期設定

■解 説

1 秒カウンタ SEC(0) ～ SEC(15) の設定を行います。
SEC 8 0
1 秒カウンタ SEC(8) をクリアします。

SEC 7 17 20 2
1 秒カウンタ SEC(7) を 17 時間 20 分 2 秒にセットします。

SEC MBK(7000)
カウンタ値の MBK への複写位置を決定し、複写をイネーブルします。

```
SEC   MBK(7000)
 SEC   5 10 58 40
 SEC   6 16 10 1
 SEC   7 17 20 2
 SEC   8 0
 FOR   i=5 TO 8
  EN_SEC   i
 NEXT   i
 FORK   11 *mon
 END
*mon
 DO
  TIME   1000
  FOR   i=5 TO 8
   SEC   i
   PRINT   MBK(7000+(i*3)) MBK(7001+(i*3)) MBK(7002+(i*3))
  NEXT   i
  PRINT   "next"
 LOOP
```

SEC

時間管理　　関数

■書 式

SEC(n)

■使い方

```
IF SEC(0)>SEC(1) THEN : print "TIME_OVER" : END_IF
```

■機 能
1 秒カウンタ

■解 説
1 秒カウンタは、SEC(0) ～ SEC(15) の 15 個用意されています。
SEC(n) は、パワオンリセット後カウントは停止しています。
EN_SEC n によってカウントが再開されます。カウンタの初期化は、SEC コマンドで行います。
SEC n 0 でクリア。SEC n 10 9 8 で 10 時間 9 分 8 秒です。
SEC(n) のデータは、時間 (2byte) 分 (byte) 秒 (byte) の 4byte 形式となっており、直接値を見ることはできません。参照するには以下の演算が必要です。

print SEC(0)/65536 --> 時間
print SEC(0)/256&255 --> 分
print SEC(0)&255 --> 秒

時間アラームとして用いるには、

SEC 10 11 12 15
IF SEC(9)>SEC(10) THEN

というように、使用していないカウンタに値をいれて比較する方法が見通しがよくなります。
また、SEC(n) の値は、MBK エリアの 3WORD にリアルタイムで複写させることができます。
これを有効にするには、SEC MBK(7800) 等実行し、なおかつ EN_SEC,DS_SEC のいずれかが、実行される必要があります。
EN_SEC,DS_SEC いずれも操作されていない SEC(n) は複写されません。
これにより、SEC(0) の複写エリアが決まり、以後 3WORD 毎に異なるカウンタの値が複写されます。

SEC(0) -> MBK(7800) MBK(7801) MBK(7802)
SEC(1) -> MBK(7803) MBK(7804) MBK(7803)

```
SEC   MBK(7000)
  SEC   5 10 58 40
  SEC   6 16 10 1
  SEC   7 17 20 2
  SEC   8 0
  FOR   i=5 TO 8
   EN_SEC   i
  NEXT   i
  FORK   11 *mon
  END
*mon
  DO
   TIME   1000
   FOR   i=5 TO 8
    SEC   i
    PRINT   MBK(7000+(i*3)) MBK(7001+(i*3)) MBK(7002+(i*3))
   NEXT   i
   PRINT   "next"
  LOOP
```

SEC

時間管理　　予約変数

■書 式
SEC

■使い方
pr SEC

■機 能
1 秒カウンタ

■解 説
1 秒ごとにアップカウントする変数です。

```
10      SEC=0
20       PRX   TIME(0)
30       WAIT   SEC>10
40       PRX   TIME(0)
#RUN

00022538
00022548
```

SELECT_CASE

制御文　　ステートメント

■書 式
SELECT_CASE arg

■使い方
```
SELECT_CASE IN(0)&&HF
 CASE 1 : GOSUB *A
 CASE 2 : GOSUB *B
 CASE_ELSE GOSUB *C
END_SELECT

SELECT_CASE VOID
 CASE SW(1) : GOSUB *A
 CASE SW(2) : GOSUB *B
 CASE_ELSE GOSUB *C
END_SELECT
```

■機 能
CASE 文中の数値による分類分岐
CASE 文中の論理式による分類分岐

■解 説
SELECT_CASE は、与えられた引数と CASE の引数を比較して、一致した CASE 文の後ろのみをを実行する排他的分類制御。【EXAM 1】

ただし、SELECT_CASE の引数を VOID とすると、CASE 文独自の論理式を評価して実行するようになる。CASE 文の評価は、上から順に行われる。
また、CASE 文中の論理式は AND,OR 等論理接続詞をもちいることができる。【EXAM 2】

CASE 1 : CASE 2 というように CASE2 文を連続して並べるとそれぞれの CASE 文の論理 OR となる。【EXAM 3】

```
【EXAM 1】
 SELECT_CASE  a
  CASE  1 :  PRINT  1
  PRINT  111
  CASE  2 :  PRINT  3
  PRINT  123
  CASE_ELSE :  PRINT  4
  PRINT  456
 END_SELECT
```

【EXAM 2】
```
 SELECT_CASE  VOID
  CASE  SW(192)==1 : PRINT  192 : WAIT  SW(192)==0
  CASE  SW(193)==1 : PRINT  193 : WAIT  SW(193)==0
  CASE  SW(194)==1 : PRINT  194 : WAIT  SW(194)==0
  CASE_ELSE
  END_SELECT
```

【EXAM 3】
```
 SELECT_CASE  A
  CASE  0
  CASE  1 : PRINT  1
  CASE  2 : PRINT  2
  CASE  5 : PRINT  5
  CASE_ELSE  : PRINT  3
 END_SELECT

 SELECT_CASE  VOID
  CASE  A==0
  CASE  A==1 : PRINT  1
  CASE  A==2 : PRINT  2
  CASE  A==5 : PRINT  5
  CASE_ELSE  : PRINT  3
 END_SELECT
```

SENSE_ON,SENSE_OFF

IO　　コマンド

■書 式
SENSE_ON port sw

■使い方
SENSE_ON 16 -1

■機 能
リアルタイム ON/OFF

■解 説
指定した入力が 1 になると、指定ポートをリアルタイム (1msec 以内) で ON します。(OFF の場合は SENSE_OFF)
SENSE_ON 16 192
SW(192) がオンすると、16 をオンします。
SENSE_OFF 16 192
SW(192) がオンすると、16 をオフします。一度反応すると、設定は解除されます。
また、強制解除は SENSE_ON VOID です。

SERCH

文字列　　コマンド

■書 式
SERCH src$ f$

■使い方
SERCH A$ "C="

■機 能
文字列の検索。

■解 説

文字列を検索し結果はポインタ ptr_ に反映される。
以下の例では、PTR$() と組み合わせて使用。

```
120     a$="adhjkashdjkas123_chuchu_tako_"
130      SERCH   a$ "123"
140     c$=PTR$(8)
150      PRINT   c$
#run

_chuchu_
#
```

SERCH$

文字列　　関数

■書 式

SERCH$(str)

■使い方

```
ptr_=d$
 ptr_=ptr_+20
 a=SERCH$("we") :  j$=PTR$(2)
```

■機 能

指定文字列を検索しポインタを検索後の位置に移動する。

■解 説

文字列を検索しポインタを検索後の位置に移動する。
SERCH$() には検索すべき文字列の指定がありません。このため、ptr_ をあらかじめ決定しておく必要があります。

```
ptr_=a$

a$="A=100 B=100"
ptr_=a$
ptr_=SERCH$("B=")-2
b$=PTR$(5)
```

b$ は、B=100 となる。

実際には文字列の最初の検索は、文字列を指定できる SERCH コマンドを使用し継続 SERCH に関数 SERCH$ を用います。
サンプルプログラムでは、ptr_、PTR$() と連携して文字列の切り出しを行っています。

```
10      a$="1234567890abcdefgABCDEFG"
30       SERCH   a$ "a"
35      s=ptr_-1 :  e=SERCH$("A") :  c=e-s-1
40      ptr_=s
50      c$=PTR$(c)
60       PRINT   c$
#run
 abcdefg
#
```

SET

パルス発生　　コマンド

■書 式

SET n x y u z

■使い方

SET 0 1 1 1 1
SET 1 5 5 5 1

■機 能

TEACH コマンドでのインチング量の設定

■解 説

TEACH コマンドでは xyuz キーでインチングすることができますが、その量を設定します。
SET で指定できるエリアは 4 つあり、0 ～ 3 を指定してそれぞれの値を設定します。
TEACH コマンドで相当する '0' ～ '3' のキーを押すと、設定されたインチング量が呼び出されます。

SETP

パルス発生 コマンド

■書 式

SETP n arg1 arg2 arg3 arg4
SETP n P(m)
SETP n PL(m;l)
SETP n strng

■使い方

SETP 1 100 100 20 3
SETP 2 X(0) Y(0) U(0) z(0)
SETP 13 P(3)
SETP 100 "abcdef"

■機 能

点データに値を設定する。n に 0 を指定すると現在位置。
n に -1 を指定するとエンコーダ・カウンタ、引数に文字列を使用すると、文字列収納。

■解 説

点データの編集コマンドです。引数に P(n) PL(m;n) を指定することができるので、点データのコピー生成にも使用できます。また、点データエリアには、文字列を収納することもできます。
文字列引数は単項のみです。(a$," ",str$() などで '+' 結合は不可)

```
FOR   i=1 TO 10
  SETP   i STR$(i-5)
 NEXT
 FOR   i=1 TO 10
  PRINT   P$(i)
 NEXT
 FOR   i=1 TO 10
a$=STR$(i-5)+" Volt"
  SETP   i a$
 NEXT
```

SET_AD

AD_DA コマンド

■書 式

SET_AD [args]

■使い方

SET_AD 10 10 10
SET_AD AD1 10 10 10
SET_AD AD7890_10 30 30 30 30
SET_AD AD1 AD7890_10

■機 能
AD 設定

■解 説
SET_AD コマンドは、AD コンバータのタイプや平均値サンプリングのサンプル個数を指定します。
AD(1,ch) での読み取りはデフォルトで 1msec*8 個での平均値となっていますが、
このサンプル個数を 2 ～ 127 の範囲で指定することができます。

たとえば、以下のコマンドラインでは、
SET_AD 10 10 10 10 20 20 20 30
ここでは
CH0 10,CH1 10,CH2 10,CH3 10 ,CH4 20,CH5 20, CH6 20,CH7 30
と平均値個数を設定しています。 2枚目の MPC-AD12 に対しての設定は以下のように行います。
SET_AD AD1 10 10 10 10 20 20 20 30

AD コンバータを AD7890-10 に交換した場合には以下を実行します。
SET_AD AD7890_10
これは、AD7890-10 が負の電圧に対して 2048 ～ 4095 の値を割り当てることに対する補正。
2枚目の MPC-AD12 を AD7890-10 に交換した場合は、以下のとおりです。
SET_AD AD1 AD7890_10
なお、AD1 AD7890_10 は予約定数で、-10 の値を持ちます。

```
FORK  1 *disp
 END
*disp
 dd=0
 M=400
  SET_AD   AD7890_10
  SET_AD   40
  FORMAT   "S000"
  DO
 t=AD(1,0) :  t2=AD(1,2)
 d=(M-t)*2 : IF   d>35 THEN  :  d=25 : END_IF
   IF   d
   PWM   0 d
 a$=STR$(t)
 b$=STR$(t2)
 c$=a$+b$
   IF   dd%10==0 THEN
    PR_LCD   c$
   END_IF
   INC   dd
   TIME   100
  LOOP
```

SET_MCX

パルス発生 コマンド

■書 式
SET_MCX axs Cmd WR6+WR7

■使い方
SET_MCX Z_A &h0006 400

■機 能
MCX314 コマンド直接設定

■解 説

SYNC コマンドで、動作トリガを規定し、そのトリガにより一定量パルス発生する場合は、
コマンドとパルス値を直接 MCX-314 に SET_MCX コマンドにより設定します。
サンプルでは、X 軸のカウントが 100 となったら、Z 軸を 50 パルス出力することにしています。
SET_MCX Z_A &h0006 50
は、Z 軸に対するコマンドで、移動量を指定するコマンド 06 と移動量 50 を指定しています。
コマンドの仕様については、MCX-314 のデータシートを参照しください。

```
ACCEL   Z_A|OUTSL 1000000 10000 1000000
 ACCEL   X_A|OUTSL 3000
 INSET   X_A CMP_CNT|PHASE2

 SYNC   X_A &H00004001 0
 SET_MCX   Z_A &h0006 50
 SYNC   Z_A 0 1
 CLRPOS Z_A
 RANGE   X_A 100 0
'
 WAIT   CMP_C(Z_A)!=0
 WAIT   RR(Z_A)==0
```

SET_RTC

時間管理　　コマンド

■書 式

set_rtc arg
set_rtc arg1 arg2 [arg3]

■使い方

SET_RTC &H20000119
SET_RTC &H00113000
SET_RTC 2007 12 19
SET_RTC 10 29 40

■機 能

RTC の時間を設定する。

■解 説

日付と時間を設定します。10 進形式では引数を 3 個入力します。
  SET_RTC 2007 12 19
  SET_RTC 10 29 40
ヘキサ形式では引数は一つで以下のフォーマットです。
#set_rtc &h20070731
 2007 年 07 月 31 日に設定
#set_rtc &h182200
 18 時 22 分 00 秒に設定

設定された時間は、DATE(0),TIME(0) で参照します。
なお、SET_RTC コマンドは、FREEZE コマンドを実行して、保護・秘密化された状態では、
実行できなくなります。実行できるは FREEZE で保護されたプログラム中の SET_RTC となります。
また、カレンダ IC は、バッテリ・ダウンを検出すると、2130/01/01 にプリセットされます。

```
SET_RTC   &H20000119
 SET_RTC   &H00113000
 PRX   DATE(0) TIME(0)
 SET_RTC   2007 12 19
 SET_RTC   10 29 40
 PRX   DATE(0) TIME(0)
```

```
S_MBK   DATE(0) 1000
S_MBK   TIME(0) 1003
PRINT   MBK(1002) MBK(1001) MBK(1000)
PRINT   MBK(1005) MBK(1004) MBK(1003)
```

SFTL

演算　　コマンド

■書 式

SFTL arry(val)
SFTR MBK(n) TO MBK(m)

■使い方

SFTL ary(5)
SFTL MBK(5) TO MBK(14)

■機 能

配列の左シフト

■解 説

SFTL arry(5)
arry(0) ～ arry(5) で左シフト
arry(5) -> arry(4) : arry(4) -> arry(3) .....

SFTL MBK(5) TO MBK(14)
MBK(5) ～ MBK(14) で左シフト
MBK(14) -> MBK(13) : MBK(13) -> MBK(12) .....

```
130             FOR   i=0 TO 9
140             ary(i)=i*1000
150             NEXT   i
160             FOR   i=0 TO 9
170              PRINT   i ary(i)
180             NEXT
190             PRINT   "SHOW SFTL"
200             SFTL   ary(5)   –> 5->4 4->3   0 -> 5
210             FOR   i=0 TO 9
220              PRINT   i ary(i)
230             NEXT

SHOW SFTL
0 1000
1 2000
2 3000
3 4000
4 5000
5 0
6 6000
7 7000
8 8000
9 9000
```

SFTR

演算　　コマンド

■書 式

SFTR arry(val)
SFTR MBK(n) TO MBK(m)

■使い方

SFTR arry(5)
SFTR MBK(5) TO MBK(14)

■機 能

配列もしくは、MBK データの右シフト

■解 説

SFTR arry(5)
arry(0) 〜 arry(5) で右シフト
arry(1) -> arry(2) : arry(2) -> arry(3) .....

SFTR MBK(5) TO MBK(14)
MBK(5) 〜 MBK(14) で右シフト
MBK(5) -> MBK(6) : MBK(6) -> MBK(7) .....

```
10       DIM   ary(5)
20       FOR   i=0 TO 4
30      ary(i)=i*1000
40       NEXT
50       FOR   i=0 TO 4
60        PRINT   i ary(i)
70       NEXT
80       SFTR   ary(3) :     'rotate ary(0)~ary(3)
90       PRINT   "SHOW SFTR"
100       FOR   i=0 TO 4
110        PRINT   i ary(i)
120       NEXT
RUN

 0 0
 1 1000
 2 2000
 3 3000
 4 4000
 SHOW SFTR
 0 3000
 1 0
 2 1000
 3 2000
 4 4000
```

SHOM[MPC-1200]

パルス発生　　コマンド

■書 式

SHOM pat

■使い方

SHOM &H50
SHOM &H55

■機 能

HOME コマンドで SD 入力の無効化

■解 説

原点復帰実行時でのニアオリジンのスローダウントリガ、SD 入力の有効、無効を設定します。
Bit の割り付けは以下のとおりです。
BIT0:SDX, BIT2: SDY,BIT4 :SDU,BIT6:SDZ
BIT に 1 を立てると無効。0 にすると有効です。

```
SHOM &H55 ' ALL SDinput ignored
 HOME ALL_A NEG_L
 WAIT RR(ALL_A)==0
 CLRPOS ALL_A
```

SHOM[MPG-2314]

パルス発生　　コマンド

■書 式

SHOM axis patn
SHOM patx paty patu patz

■使い方

SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF
SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF|CW
SHOM X_A|Z_A|Y_A IN0_ON
SHOM IN0_ON 0 0 0

■機 能

原点復帰の条件を決定する。

■解 説

MPG-2314 の原点復帰検出センサは各軸 2 二つずつあり IN0 と IN1 に区別されています。
例えば Y 軸の場合、MPG-2314 の J4 上では、YIN0,YIN1 と名づけられています。

IN0 はニアオリジン IN1 は Z 相を想定していますので必要に応じて設定します。
また、SHOM の設定は、HOME コマンドを実行しない限り有効ではありません。

SHOM X_A|Z_A|Y_A IN0_ON
この場合はニアオリジンのみの原点復帰を想定します。ニアオリジンをオン検出すると停止します。

SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF|CW
この場合は X,Y,Z 軸に対して動作を規定しています。
ニアオリジン停止後、Z 相サーチとなります。サーチ方向は CW 方向です。
ニアオリジンは ON 検出、Z 総は、OFF 検出です。CW/CCW を省略すると CCW 方向となります。

SIN

浮動小数点　　コマンド

■書 式

sin deg r var [sf]

■使い方

sin 450000 100000 a
sin 4500000 100000 a 100000

■機 能

sin 関数演算

■解 説

浮動小数点 SIN 演算を行います。
var = r × sin(deg/sf)
注) sf を省略すると、sf を 10000 とします。

以下は SIN コマンド実行例です。

```
#SIN 300000 10000 a
#pr a
5000
#
```

この意味は、sin(300000/10000) の演算で sin(30 度) の計算になります。
結果は 0.5 ですが、10000(sf) × 0.5 のため、5000 となります。

```
#sin 450000 100000 a
#pr a
 70711
#
sin 4500000 100000 a 100000
#pr a
 70711
```

SIN,COS,TAN

浮動小数点　　関数

■書 式
SIN(rad),COS(rad),TAN(rad)

■使い方
FP(0)=SIN(FP(0))
FP(1)=TAN(RAD(30))

■機 能
三角関数

■解 説
ラジアン引数の倍精度三角関数です。FLOAT コマンド中でのみ、意味を持ちます。

```
FLOAT FP(1)=SQR(SQ(SIN(RAD(i)))+SQ(COS(RAD(i))))
```

SLMTn

パルス発生　　予約定数

■書 式
SLMTn

■使い方
LMT(X_A,SLMTn)

■機 能
エラービット指定

■解 説
対象ボード : MPG-2314
ソフトリミット - ビット

```
IF LMT(X_A,SLMTn)!=0 THEN          /* confirming reason for stop
```

SLMTp

パルス発生　　予約定数

■書 式
SLMTp

■使い方
LMT(X_A,SLMTp)

■機 能
エラービット指定

■解 説
対象ボード : MPG-2314
ソフトリミット + ビット

```
IF LMT(X_A,SLMTp)!=0 THEN              /* confirming reason for stop
```

SLMT_OFF

パルス発生　　予約定数

■書 式
SLMT_OFF

■使い方
INSET X_A|Y_A SLMT_OFF

■機 能
ソフトリミット設定

■解 説
対象ボード : MPG-2314
ソフトリミットを無効にします。

```
INSET X_A|Y_A SLMT_OFF             /* 'SOFT LIMIT' disable
```

SLMT_ON

パルス発生　　予約定数

■書 式
SLMT_ON

■使い方
INSET X_A|Y_A SLMT_ON

■機 能
ソフトリミット設定

■解 説
対象ボード : MPG-2314　　ソフトリミットを有効にします。

```
10 PG 1
20 RANGE X_A|Y_A 200000 -1000      /* XY axes operative restriction set
30 INSET X_A|Y_A SLMT_ON           /* 'SOFT LIMIT' enabled
```

SLOW_RUN

保守　　コマンド

■書 式
SLOW_RUN taskn [timer]
SLOW_RUN TMOUT [n]

■使い方

SLOW_RUN 1 100
SLOW_RUN TMOUT 1000

■機 能

指定タスクの遅延実行
ダウンカウンタ時間設定

■解 説

SLOW_RUN は、引数により、以下の二通りの使い方があります。
例 1) SLOW_RUN 1 1000
この場合は、タスク 1 の実行ステップごとに 1000msec 時間待ちを指定することになります。
この値はプログラム実行中にも変更できます。
デバッグの最初はこのコマンドにより、慎重にゆっくりプログラムを実行させ、デバッグが進展するにしたがってより早く実行させるようにします。
プログラムの安全が確認できたら、"SLOW_RUN 1" を実行します。
このように、引数にタスク番号のみを指定すれば、タイマー待ちは解除されます。

例 2) SLOW_RUN TMOUT 1000
引数に予約定数 "TMOUT" を追加すると、タイムアウトダウンカウンタの設定となります。
通常、ダウンカウンタは通常 100m 秒ごとにダウンカウントしますが、この例のように 100 以上の値を設定すると、指定 msec 値ごとのダウンカウントとなります。
この例では、1000msec(1 秒) ごとにダウンカウントとなります。

なお、SLOW_RUN での設定は、パワーオンリセットで解除されますが、プログラム中に記述すると遅延要素が不用意に設定されてしまうのでコマンドとして使用してください。

SPEED

パルス発生 コマンド

■書 式

SPEED [axs] n

■使い方

SPEED n
SPEED X_A n

■機 能

パルス発生の pps 設定

■解 説

パルス発生を ACCEL で指定した最高速度 pps 以下で、n pps 指定できる。
FEED コマンドより速度を細かくドライブ速度を指定できる。ただし分解能は、(最高速度 /8192) pps となる。サンプルプログラムのように、パルス発生中の細かな速度変更にも有効。

```
40          ACCEL   40000 1000
50          RMVC   U_A 1
60          DO
70           FOR   i=1 TO 10
80            SPEED   U_A i*4000
90            TIME   100
100           NEXT
110           FOR   i=10 TO 1 STEP -1
120            SPEED   U_A i*4000
130            TIME   100
140           NEXT
150         LOOP
```

SQR

浮動小数点　　関数

■書 式

SQR(v)

■使い方

FP(3)=SQR(3)
A=SQR(3*3+4*4)

■機 能

平方根

■解 説

FLOAT コマンド中では、倍精度平方根取得関数です。整数演算中では、整数の開閉計算となります。

```
FP(0)=SQR(1+3+5+7)
```

STACKS

保守　　コマンド

■書 式

STACKS

■機 能

スタックエリアの消費状態を表示する。

■解 説

STACK FREE は未使用のスタックエリアのロング・ワード数。
POS は、現在のスタックポインタの位置です。ロング・ワードのカウント数で表示されます。0 となっているのは、まだ起動されていないタスクです。

```
#stacks
 TASK0 STACK FREE=156 STACK POS =38
 TASK1 STACK FREE=200 STACK POS =0
 TASK2 STACK FREE=200 STACK POS =0
 TASK3 STACK FREE=200 STACK POS =0
 TASK4 STACK FREE=200 STACK POS =0
 TASK5 STACK FREE=200 STACK POS =0
 TASK6 STACK FREE=200 STACK POS =0
 TASK7 STACK FREE=200 STACK POS =0
 TASK8 STACK FREE=200 STACK POS =0
 TASK9 STACK FREE=200 STACK POS =0
 TASK10 STACK FREE=200 STACK POS =0
 TASK11 STACK FREE=200 STACK POS =0
 TASK12 STACK FREE=200 STACK POS =0
 TASK13 STACK FREE=200 STACK POS =0
 TASK14 STACK FREE=200 STACK POS =0
 TASK15 STACK FREE=200 STACK POS =0
#
```

STOP

パルス発生　　コマンド

■書 式

STOP axis arg1

■使い方

STOP X_A STP_D
STOP ALL_A IN1_ON
STOP X_A|Y_A VOID

■機 能

停止命令発行もしくは停止モードの設定

■解 説

STOP X_A STP_D
このタイプのコマンドは、対象 MPG に対して減速停止、あるいは即停止命令を発行します。
STP_D は減速、STP_I は即停止です。プログラム例は、動作中に入力スイッチで停止させる方法の例です。

STOP ALL_A IN0_ON|IN1_OFF
このタイプのコマンドは、MPG-2314 の入力ポートの機能を決めます。
IN0_ON|IN1_OFF の場合は、IN0(S0) がオン、IN1(S1) がオフになると停止します。

停止条件は、コマンド実行後維持されますので、解除する場合は引数に VOID を与えます。
INn による停止は、ACCEL コマンドでドライブ速度 > 初速度の場合は、減速停止ですが、
ドライブ速度 == 初速度とすると即停止となります。

停止条件の解除は VOID を指定します。
STOP X_A VOID

```
EX1:
MOVL 10000 10000 0
 WHILE RR(ALL_A) : IF SW(192) THEN : STOP STP_D : END_IF : WEND

EX2:
STOP X_A IN0_OFF  /* setting the STOP condition
  RMVS X_A POS_L    /* Generating pulse
  WAIT RR(X_A)==0
  STOP X_A VOID     /* clear the STOP condition
  RMVS X_A 1000
```

STPS

パルス発生 コマンド

■書 式

STPS axis n
STPS argx [argy,argu,argz]

■使い方

STPS X_A 1000
STPS 100 200 300 400
STPS VOID 100 200
STPS X_A|Y_A 1000

■機 能

現在位置設定

■解 説

軸指定した場合は、相当軸に同じ値を設定します。
引数を並べた場合は、X Y U Z の順序で設定できます。
VOID が与えられているか、省略された引数の軸は設定されません。

STP_D

パルス発生　　予約定数

■書 式
STP_D

■使い方
STOP X_A STP_D

■機 能
停止方法選択

■解 説
対象ボード : MPG-2314/MPC-1200
減速停止

```
STOP X_A STP_D          /* X-axis Stop with deceleration
STOP ALL_A STP_D        /* All-axes Stop with deceleration
```

STP_I

パルス発生　　予約定数

■書 式
STP_I

■使い方
STOP X_A STP_I

■機 能
停止方法選択

■解 説
対象ボード : MPG-2314/MPC-1200
即停止

```
STOP X_A STP_I          /* X-axis Stop without deceleration
STOP ALL_A STP_I        /* All-axes Stop without deceleration
```

STR$

文字列　　関数

■書 式
STR$(arg)

■使い方
a$="data=" +str$(A)

■機 能
数値の文字列化

■解 説
数値を数字文字列に変換します。
例えば以下のように実行すると、A$ は、"DATA= 1000" という文字列となります。

A=1000
A$="DATA="+STR$(A)

標準状態で、正の値には先頭にスペースを付加、負の値には "-" を付加します。
変換の様式は、FORMAT コマンドで行います。

STRCPY

文字列　　コマンド

■書 式

STRCPY src$ dst$ [m n]

■使い方

STRCPY src$ dst$
STRCPY src$ dst$ 6 3

■機 能

文字列の複写

■解 説

文字列を dst$ に複写。mはソース文字列 (src$) の複写開始位置 (文字列先頭が 0)。n は複写文字数。m,n を指定しなければ、全文字複写となります。

a$="012345abc"
strcpy a$ c$ 6 3　　であれば、c$ は "abc" となります。

```
a$="111111111011aaaaaaa123baka_aabbanbQERaho_b11111229we48r9"
 PRINT  a$
 PRINT  LEN(a$)
 FOR  i=0 TO 5
  STRCPY  a$ b$ i 10
  PRINT  b$
 NEXT  i
 FOR  i=20 TO 25
  STRCPY  a$ b$ i
  PRINT  b$
 NEXT  i
#run
1111111110
1111111101
1111111011
111111011a
11111011aa
1111011aaa
23baka_aabbanbQERaho_b11111229we48r9
3baka_aabbanbQERaho_b11111229we48r9
baka_aabbanbQERaho_b11111229we48r9
aka_aabbanbQERaho_b11111229we48r9
ka_aabbanbQERaho_b11111229we48r9
a_aabbanbQERaho_b11111229we48r9
#
```

STR_LEN

通信　　予約定数

■書 式

STR_LEN

■使い方

PRINT# 3 STR_LEN|32 a$

■機 能

出力文字数指定

■解 説

PRINT# 参照

SUBST

文字列　　コマンド

■書 式

SUBST str

■使い方

```
b$="ABC123 &H1234FJ  &HBCDEF1 "
    SERCH   b$ "&H"
    ptr_=ptr_-2
   SUBST   " $"
```

■機 能

文字列を置き換える。

■解 説

SUBST は文字列ポインタの位置から与えられた文字列を上書きします。

```
70      b$="ABC123 &H1234FJ  &HBCDEF1 "
80      SERCH   b$ "&H"
120      ptr_=ptr_-2
130      SUBST   " $"
140     ptr_=SERCH$("&H")
150     ptr_=ptr_-2
160      SUBST   " $"
170      PRINT   b$
#ABC123  $1234FJ   $BCDEF1
#
```

SW

IO　　関数

■書 式

SW(arg)

■使い方

```
A=SW(192)   /* 入力ポートの読み出し
IF SW(A)==0 THEN : ON 5 : END_IF /* 入力による条件分岐
WAIT SW(192)==1 /* 条件待ち
```

■機 能

入力ポートの読み出し

■解 説

入力ポートが GND にショートされると 1 を返します。
フロート状態では 0 となります。

SWAP

マルチタスク　　コマンド

■書 式

SWAP

■機 能

実行中にプログラムを強制スワップ (実行タスク交替)

■解 説

長い処理時間を要するタスクを実行させていると、タスクのタイムスライス時間いっぱいまで時間を占有し他のタスクの実行が遅くなります。

こうした場合、人為的に SWAP を実行させることによって実行権を強制的に他のタスク移すことができます。

SYNC

パルス発生　　コマンド

■書 式

SYNC axs　WR6 WR7

■使い方

SYNC　X_A &H00004001 0
SYNC　Z_A 0 4

■機 能

MCX-314 レジスタ設定

■解 説

MPG-2314 に搭載されている MCX-314 には、ハード的にリアルタイム処理を行う機能があります。
・X 軸が一定パルスを超えたら、他の軸を起動する。
・入力信号がはいったら、その時点のカウンタ値をラッチする

このコマンドにより、このような機能をハード上の機構によりリアルタイムに実行することができます。
WR6,WR7 に実際どのような値をセットするかは、MC-314 のデータシートを参照してください。
サンプルプログラムでは、X 軸のカウント値 (100) により、Z 軸を起動し、その値が 50pulse になったところで、出力ポート (O3) をオンするというものです。

```
ACCEL   Z_A|OUTSL 1000000 10000 1000000
 ACCEL   X_A|OUTSL 3000
 INSET   X_A CMP_CNT|PHASE2

 SYNC   X_A &H00004001 0
 SYNC   Z_A 0 4
 CLRPOS Z_A
 RANGE   Z_A 50 0
 RANGE   X_A 100 0
'
 WAIT   CMP_C(Z_A)!=0
 STOP   Z_A STP_I
 WAIT   RR(Z_A)==0
```

SYSCLK

時間管理　　予約変数

■書 式

SYSCLK

■使い方

pr SYSCLK

■機 能

システムクロック

■解 説

パワーオン後約 1msec ごとにインクリメントする変数です (CPU のクロック基準)

```
10      SYSCLK=0            /* SYSCLK clear
20       TIME   100         /* delay 100msec
```

```
30      a=SYSCLK
40       PRINT   a                  /* display
RUN

 101
```

S_MBK

タッチパネル　　コマンド

■書 式

S_MBK arg1 arg2
S_MBK str adr c
S_MBK arg adr count

■使い方

S_MBK 1 10
S_MBK 2 11
S_MBK 1000000 20~Lng
S_MBK a$ 100 10
S_MBK 100 50 20
S_MBK 100~Lng
S_MBK DATE(0) 100

■機 能

タッチパネルデータ設定

■解 説

S_MBK はコマンドは、MBK 配列に値を設定します。

1) 10 番に 1 を設定　->　S_MBK 1 10

2) 20 番に 1000000 を設定。-> S_MBK 1000000 20~Lng
この場合 MBK(20) に下位ワード、MBK(21) に上位ワードが入る。

3) 文字列の代入 ->　S_MBK a$ 100 10 文字列 ("abc" も可) を 100 番地から 10 文字設定
文字列は文字列変数か文字列定数

4) 一括設定　-> S_MBK arg adr count
MBK(adr) ～　MBK(adr + count -1) に値 arg を一括代入します。

5) 表示 -> S_MBK n とすると MBK(n) ～内容表示。Lng 表示は、S_MBK n~Lng とする。

S_MBK DATE(0) n / S_MBK TIME(0) n の場合は、
日、秒　--> MBK(n)
月、分　--> MBK(n + 1)
年、時間 --> MBK(n + 2)
この値は 1 秒ごとに自動更新します。停止させる場合は、S_MBK DATE(0) 0 , S_MBK TIME(0) 0 と 0 を指定します。
この場合指定アドレスは 0 となっていますが、MBK(0) ～ MBK(2) への書き込みは行われません。

TAIL

編集　　コマンド

■書 式

TAIL

■機 能
最大文番号の表示

■解 説
最大文番号の表示。オンラインでプログラムを追加する時に使います。

TAN

浮動小数点　　コマンド

■書 式
tan deg r var［sf］

■使い方
tan 300000 100000 a
tan 3000000 100000 a 100000

■機 能
TAN 演算

■解 説
浮動小数点 TAN 演算を行います。
var = r × tan(deg/sf)
注) sf を省略すると、sf を 10000 とします。

例 1) TAN 450000 10000 a
#pr a
10000
#
tan(450000/10000) の演算で tan(45 度) の意味です。
結果は 1 ですが、10000 × 1 のため、10000 となります

```
#tan 300000 100000 a
#pr a
 57735
#
#tan 3000000 100000 a 100000
#pr a
 57735
#
```

TASK

マルチタスク　　関数

■書 式
TASK(n)
TASK(n|256)

■使い方
WAIT TASK(1)!=0
a=TASK(1|256)

■機 能
タスクの状態を参照する。

■解 説
n はタスク番号です。

TASK(n) の戻り値
0: タスクは実行中。
1: タスクはタイマーにより待機している。
255: タスクは終了しているか、QUIT されている。
引数に -1 をいれると、自己タスク番号を返す。

TASK(n|256) の戻り値
bit0: タスクはタイマーにより待機している。
bit1: タスクは PAUSE されている。
例)
0: タスクは実行中
1: TIME 実行中
2: PAUSE された
3: TIME のとき PAUSE された
255: タスクは終了しているか、QUIT されている。
(TASK 0 は END 終了すると 0 が返されます)

TASKn

マルチタスク　　予約変数

■書 式
TASKn

■使い方
IF TASKn==0 THEN

■機 能
自己タスク番号取得

■解 説
実行中のタスク番号を得ます。TASKn はグローバル変数ですが、タスク・モニタにより、タスクに実行権がひきわたされるたびに、TASKn にタスク番号を書き込んでいます。
従って、TASKn をあやまって、他の値に変更しても、タスクが切り替わるごとに正常化します。

```
10       FORK   10 *SUBTASK
20       PRINT   "main=" TASKn
30       END
40     *SUBTASK
50       TIME   500
60       PRINT   "sub=" TASKn
70       END
#run

 main= 0
# sub= 10
```

TEACH

パルス発生　　コマンド

■書 式
TEACH

■使い方
TEACH
T

■機 能
インチング操作による点データの教示

■解 説

TEACH コマンド実行前に、PG が選択され、ACCEL コマンドが実行されていなければなりません。
TEACH コマンドを実行すると、以下のように現在位置とインチング量が表示されます。
それぞれの軸は以下のキーでインチングします。
x,X
y,Y
u,U
z,Z

#t
PG=[1] X=1200 Y=0 U=0 Z=0 dx=200 dy=200 du=200 dz=200 P(30

P コマンドを押すと点番号入力待ちとなります。番号を入力すると指定された点データに現在地がセットされます。0 ～ 3 のキーで SET コマンド設定されたインチング量を選択することができます。

なお、対象 PG が MPG-2314 の場合、位置表示とあわせてエラー表示もします。
PG=[1] X=1200 Y=0! U=0 Z=0 dx=200 dy=200 du=200 dz=200
エラーとなった軸の数値の後ろに ! が表示されます。

TIME

時間管理　　関数

■書 式

TIME(0)
TIME(255)
TIME(VOID)

■使い方

```
IF  TIME(0) < &H00182800 THEN
 GOTO  *aho
END_IF
```

■機 能

時間データ取得

■解 説

ヘキサ形式で時間値を得ます。引数をいれると、引数と論理積をとって値を返します。
引数に VOID を設定すると、10 進で値を返します。
なお、時間設定は SET_RTC コマンドで実施します。

```
5           *aho
10            IF  TIME(0) < &H00182800 THEN
30            PRX  TIME(0)
40            WAIT  TIME(255)%16==0
```

TIME

時間管理　　コマンド

■書 式

TIME arg

■使い方

TIME 100

■機 能
指定 msec タスクを停止します。

■解 説
TIME はタイミングをとるコマンドであると同時に実行効率を向上させるコマンドです。
TIME で時間待ちをしている時間は、そのタスクは SLEEP となり、CPU の時間資源を多のタスクに割り当てることができます。

TIME$

文字列 関数

■書 式
TIME$(n)

■使い方
a$=DATE$(1)+" "+TIME$(1)

■機 能
時間文字列取得

■解 説
時間文字列を得ます。
TIME$(0)-> 00100957
TIME$(1)-> 10:09:57
TIME$(2)-> 10:09

```
a$=DATE$(1)+" "+TIME$(1)+": CNT="+STR$(i)
```

TIMEOUT

時間管理 関数

■書 式
TIMEOUT(n)

■使い方
WAIT SW(1)&SW(2) OR TIMEOUT(0)

■機 能
timer_ のタイムアウト判別

■解 説
n==0 の時、(timer_==0) と同じ意味。
WAIT SW(1)&SW(2) OR TIMEOUT(0) という記述により、意味が明確になる。
なお、TIMEOUT() 関数は他のタスクの timer_ も評価できる。
n : -1　タスク 0 の timer_ の 0 判別
n : 1 ～ 31 タスク n の timer_ の 0 判別

```
timer_=10
PRINT TIMEOUT(0)
WAIT SW(1)&SW(2) OR TIMEOUT(0)
PRINT TIMEOUT(0)
```

TIMER

時間管理 関数

■書 式
TIMER(arg)

■使い方
a=TIMER(3)
a=TIMER(VOID|3)
a=TIMER(1,1)

■機 能
timer_ を参照

■解 説
timer_ はタスク変数のため、他のタスクから値を参照したり設定することができません。
TIMER(n) は引数にタスク番号を指定すると、そのタスクの timer_ の値を得ることができます。
注) 単位は 0.1 秒です。

また、タスク番号に VOID を論理和すると、そのタスクの timer_ を 0 にできます。
TMOUT コマンドで管理されるタイムアウトは、この timer_ 変数を使っていますので、他ののタスクから強制タイムアウトさせるには、この関数を使って、timer_ を 0 にします。
TIMER(1,n) という形式で引数を与えると、タスク n の TMOUT 設定値を呼び出す事ができます。

timer_

時間管理　　予約変数

■書 式
timer_

■使い方
IF timer_==0 THEN

■機 能
ダウンカウンタ

■解 説
タスク変数
0.1 秒ごとにダウンカウトして 0 で停止します。

```
10      timer_=100                  /* set 10Sec -> 0.1Sec count down
20       PRX   TIME(0)              /* display current time
30       WAIT   timer_==0           /* wait 10sec
40       PRX   TIME(0)              /* display current time
#run
00014841
00014851

———

10       FORK   3 *JOB
20      SYSCLK=0                    /* SYSCLK init
30       TIME   100
40       WAIT   TIMER(3)==0         /* wait "timer_" of the TASK3 == 0
50       PRINT   SYSCLK "mSec"
60       END
70     *JOB                         /* TASK3
80      timer_=100                  /* set 10Sec -> 0.1Sec count down
90       DO   : SWAP   : LOOP
#RUN

 9995 mSec
```

TMOUT

時間管理　　コマンド

■書 式

TMOUT n [taskn]

■使い方

TMOUT 100
TMOUT 100 n
TMOUT VOID

■機 能

タイムアウト時間設定

■解 説

タイムアウト時間は msec 単位で設定します。設定できる最小時間は 10msec です。
(対象 :WS0(),WS1(),HOME)
デフォルトでは、13 日 (20000 秒) が設定されています。
2 番目の引数はタスク指定です。省略すると自己タスクに対する指定となります。
引数に VOID を指定すると、初期値の 20000 秒がセットされます。
なお、TMOUT で設定された値は、WS0,WS1,HOME の中で timer_ に設定されます。
このため、WS0,WS1,HOME の直前で、timer_ の値を操作しても無効です。
引数なしの TMOUT コマンドで現状の設定値を表示させることができます。

```
#TMOUT
TMOUTs
0 10000
1 10000
2 2000000
3 2000000
4 2000000
5 2000000
6 2000000
7 2000000
8 2000000
9 2000000
10 2000000
11 2000000
12 2000000
13 2000000
14 2000000
15 2000000
#
```

TMOUT

通信　　予約定数

■書 式

TMOUT

■使い方

INPUT# 1 TMOUT|5 a$

■機 能

受信待ちタイムアウト設定

■解 説

INPUT# コマンドの受信待ちには時間制限がありませんが、TMOUT オプションにより、制限時間を設ける事ができます。時間制限を越えた場合、rse_ 変数に TMOUT 発生の有無が反映されます。

```
10      CNFG#   1 "9600b8pns1NONE"
20      INPUT#   1 TMOUT|5 a$          /* timeout 5 sec
35      IF   rse_==1 THEN               /* check timeout
36       PRINT   "timeout"
37       ELSE
38       PRINT   a$
40      END_IF
#RUN

 timeout                                /* fail
#RUN

 asdfg                                  /* success
#
```

UIN0

パルス発生　　予約定数

■書 式
　UIN0

■使い方
　HPT(UIN0)

■機 能
　HPT 入力指定

■解 説
　対象ボード : MPG-2314
　HPT 入力ポートに UIN0 を指定します。
　関連 : HOME　　see also XIN0

UIN1

パルス発生　　予約定数

■書 式
　UIN1

■使い方
　HPT(UIN1)

■機 能
　HPT 入力指定

■解 説
　対象ボード : MPG-2314
　HPT 入力ポートに UIN1 を指定します。
　関連 : HOME　　see also XIN1

UP_DWN

パルス発生　　予約定数

■書 式
　UP_DWN

■使い方
　INSET UP_DWN

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
カウンタをアップダウンカウンタにする

```
INSET UP_DWN                    /* Set the counter to 'UP/DOWN' mode
```

USB

USB 関数

■書 式
USB(0)

■使い方
IF USB(0)!=1 THEN : GOTO *NOUSB : END_IF
PR USB(1,0)-MBK(1000+3)

■機 能
USB メモリの有無

■解 説
USB メモリの有無を得ることができます。
USB(0) 関数は、USB メモリが正しく装着されていると、1 を返しますが、無いと、0 を返します。
また、以下のように上位ワードに 1 をいれると USB メモリの総容量を返します。(Mbyte)
USB(1,0)
この値が有効になるのは、DIR コマンド直後か、電源オン時に USB メモリが装着されている場合です。

```
FILE$="TEST"
*RETRY
 DO
  IF   USB(USB)!=1 THEN  : GOTO  *NO_USB : END_IF
  USB_WRITE   "TEST_WRITING ¥r¥n"
  TIME   100
  DIR   1000
  IF   USB(1,USB) <100 THEN  : GOTO  *NO_SPACE : END_IF
  TIME   1000
 LOOP
*NO_USB
 WAIT   USB(USB)==1
 GOTO  *RETRY
*NO_SPACE
 PRINT   "CHANGE_USB_MEMORY"
```

USB_DEL {UDL}

USB コマンド

■書 式
USB_DEL Str

■使い方
USB_DEL "aaa.p2k"
UDL "aaa.f2k"

■機 能
USB メモリファイル抹消

■解 説
USB メモリ上の指定ファイルを抹消します。
ファイルが存在しない場合に実行してもエラーとはなりません。

USB_LOAD {UL}

USB コマンド

■書 式
USB_LOAD strg

■使い方
USB_LOAD "DEMO.F2K"
USB_LOAD "DEMO.F2K"

■機 能
USB メモリから、プログラムをロードする

■解 説
FTMW で USB メモリ上に SAVE された、プログラムデータをロードします。

USB_PLOAD {UPL}

USB コマンド

■書 式
USB_PLOAD str

■使い方
USB_PLOAD "PL3.P2K"
USB_PLOAD "PL3.P2K"

■機 能
点データをロード

■解 説
FTMW でセーブされた P2K,P68 タイプのデータを USB メモリからロードします。
上書きなので、新規データとする場合は、実行前に NEWP を実行します。
また、USB_PLOAD では、MBK データにも対応します。

```
= データサンプル =
SETP 1 -200 9280 0 -12680
SETP 2 30880 9280 0 -13280
SETP 3 400 29400 0 -13280
SETP 4 31080 29280 0 -13680
SETP 101 8000 0 0 0
SETP 102 8000 8000 8000 4000
SETP 103 0 8000 0 0
SETP 104 0 0 0 4000
s_mbk 100 1
s_mbk 200 2
s_mbk 300 3
s_mbk 400 4
s_mbk 500 5
s_mbk 600 6
s_mbk 700 7
s_mbk 800 8
s_mbk 900 9
s_mbk 1000 10
s_mbk 30 7868
```

USB_PSAVE {UPS}

USB　　コマンド

■書 式

USB_PSAVE P(n) cnt Str
USB_PSAVE MBK(n) cnt Str

■使い方

USB_PSAVE　P(1) 5000 "aa.p2k"
USB_PSAVE　MBK(10) 1000 "aa.p2k"

■機 能

点データ、MBK データの保存

■解 説

点データもしくは MBK データの n から cnt 個 USB に保存します。
点データ保存で座標要素がすべて 0 の場合は、保存されません。
保存は、APPEND 保存のため、ファイル既存の場合は、データ追加となります。
新規データとして保存する場合は、事前に USB_DEL コマンドでファイルを抹消します。
USB_DEL "AUTO.P2K"
USB_PSAVE P(1) 2000 "AUTO.P2K"
USB_PSAVE MBK(10) 1000 "AUTO.P2K"
この場合、AUTO.P2K には、点データと MBK データが保存されるため、リカバリデータとして使用することができます。

注意) 電源オンオフをまたいだ APPEND は USB メモリによってはできない場合があります。

USB_READ {URD}

USB　　コマンド

■書 式

USB_READ String

■使い方

USB_READ　a$
USB_READ　-1

■機 能

USB メモリファイル一行読み込み

■解 説

USB メモリファアルの一行読み出しです。ファイル名は FILE$ で指定します。
順々に実行することによって、一行ずつ読み取ることができます。
ファイルの終端になると、関数 EOF(0) が 1 を返すので、読み取りをそこで停止します。
無視して、URD を実行し続けると、またファイル先頭から読み始めます。
途中で読み出しを停止するには、USB_READ -1 を実行します。
この処理によりファイルはクローズされます。

```
FILE$="AUTO.P2K"
 DO
  USB_READ   a$ : PRINT   EOF(0) a$
  IF   EOF(0)==1 THEN  : END   : END_IF
 LOOP
```

USB_WRITE {UWR}

USB　　コマンド

■書 式

USB_WRITE Strng

■使い方

USB_WRITE "123.456"
USB_WRITE STR$(n)

■機 能

USB メモリにアペンドライトする。(都度オープンクローズ)

■解 説

USB メモリに追記書込みする。
ファイル名の指定は予約文字列 FILE$(USB1 -> FILE1$,USB2 ->FILE2$)
引数の文字列には書き込む文字列をセットする。
都度、ファイルオープン・クローズするため、途中での電源断が発生しても、最終書き込みされたデータは USB メモリ中に残る。

```
----------------------------------------------------
LIST
10        FOR   k=1 TO 100
20         FORMAT   "uwr_00.txt"
30       FILE$=STR$(k)
40       SEC=0
50         FOR   SUM=1 TO 100
60          FORMAT   "TEST_CNT=0000¥n"
70       A$=STR$(SUM)
80          FORMAT   "APND_CNT=0000¥n"
90       B$=STR$(SUM+1000)
100         USB_WRITE   A$+B$
110        NEXT
120        PRINT   k SEC
130       NEXT
#
```

U_A

パルス発生　　予約定数

■書 式

U_A

■使い方

RMVS U_A 1000

■機 能

U 軸指定

■解 説

対象ボード : MPG-2314/MPC-1200
RMVS 等の PG コマンドでの軸指定コマンドです。
see also X_A

U_C

パルス発生　　予約定数

■書 式

U_C

■使い方
stps U_C 1000

■機 能
カウンタ指定

■解 説
対象ボード : MPG-2314
U カウンタを指定します。
see also X_C

U_E

パルス発生　　予約定数

■書 式
U_E

■使い方
RR(U_E)

■機 能
U 軸エラー指定

■解 説
RR() 関数の引数として使用し、移動後の U 軸のエラーの有無を調べます。
0 でなければ、なんらかのエラー要因が発生していることを示しています。
エラーの詳細は、LMT 関数 ,PGE 関数で調査します。
対象ボード : MPG-2314
see also X_E

VAL

文字列　　関数

■書 式
VAL(str)
VAL(arg)

■使い方
```
a$="a=1000 b=-1000 c=100"
a=VAL(a$) : b=VAL(0) : c=VAL(0)
 a$="x=1000.123 y=-2120.1256 "
  SERCH   a$ "x="
  PRINT   VAL(1000)
```

■機 能
文字列から数字列を取り出しその値を得る。

■解 説
文字列中から数字文字列を探し出し、数値に変換します。
文字列中に複数の数字文字列が含まれている場合は、連続して VAL(0) を実行します。
このように、改めて文字列を指定せず arg を 0 とした場合は、順々に数字列を取り出して数値に変換します。arg に、10 ～ 100000 までの数値をいれると小数点の桁で指定倍します。
"X=123.4567"
というような場合は、VAL(10000) で読み取れば 1234567 という整数値が得ることができます。

```
10      a$="x=1000.123 y=-2120.1256 "
20       PRINT   VAL(a$)
```

```
30       SERCH   a$ "x="
40       PRINT   VAL(1000)
50      ptr_=SERCH$("y=")
60       PRINT   VAL(10000)
```

文字列最初から、小数点を含む場合

```
10 A$="123.22 B=456.12 C=789.34"
80 ptr_=A$
90 A=VAL(100)
100 B=VAL(100)
110 C=VAL(100)
120 PRINT A B C
```

VAL

浮動小数点　　関数

■書 式

VAL(str)
VAL(0)

■使い方

FP(0)=VAL(A$)

■機 能

浮動小数点値取得

■解 説

FLOAT コマンド中で、数字文字列を浮動小数点変数として取得。
内部では、小数点以上、以下、また E 指定の各数字列を倍精度整数として扱います。
このため、それぞれの数字列が倍精度整数 (9 桁以内) を超えるとエラーとなります。

```
A$="Mx+9.7042e+002 My+6.3210e+002"
#FP(0)=VAL(A$)
#FP(1)=VAL(0)
#pr fp(0) fp(1)
 9.704200E+02 6.321000E+02
#
```

VARS

保守　　コマンド

■書 式

VARS [arg]

■使い方

VARS
VARS VOID
VARS 0

■機 能

変数のリスト

■解 説

1) 引数無し
四文字以上、一文字のみ相違のある変数を表示します。紛らわしい変数を表示することによって変数による混乱を減らします。

2)VOID

RUN 後に実行します。"VARS VOID"
実行中に一度も値が代入されない変数を表示します。これは初期化が不完全であったり、誤って使われたラベルの発見に便利です。

3) 値　"VARS 0"
値を指定すると、その値を持つ変数を表示します。

VER

保守　　コマンド

■書 式
VER

■使い方
#VER

■機 能
バージョン表示

■解 説
MPC-1200 の実行例です。
バージョン番号、タイムスタンプ、プログラム容量、点データ数などを表示ます。

```
#VER
MPC-1200H BL/I 1.14_29 2014/08/08
All Rights reserved. ACCEL Corp.
PRG_430K PNT_20K DIM_20K .T32
```

VER$

文字列　　予約変数

■書 式
VER$

■使い方
pr VER$

■機 能
バージョンデータ取得

■解 説
バージョンデータが入っている文字列変数です。
バージョン番号は、MBK(8053) でも取得できます。

```
10      DIM   a(10)
20      FILL   a(0) 0
30     a$=VER$
40      PRINT   "MPC_Version" a$
50      GET_VAL   a$ a(0)
60      PRA   a(0)
70      PRINT   "MBK_8053=" MBK(8053)
#RUN

 MPC_Version 1.11_29 2009/01/22

a(0)=1
a(1)=11
a(2)=29
a(3)=2009
```

```
a(4)=1
a(5)=22
a(6)=-2147483648
a(7)=-2147483648
a(8)=-2147483648
a(9)=-2147483648
 MBK_8053= 11129
#
```

VOID

パルス発生　　予約定数

■書 式

VOID

■使い方

MOVL VOID 1000 2000 VOID

■機 能

入力無効
設定解除

■解 説

対象ボード : MPG-2314/MPC-1200 他
パルス発生コマンド、I/O コマンドの設定解除や SELECT_CASE 等に使われます。

```
INSET ALL_A VOID                /* INSET conditions clear
MOVL 1000 0 0 VOID              /* Z axis disable
STOP ALL_A VOID                 /* STOP conditions clear
INTA_ON VOID                    /* INTA_ON disable
INTA_OFF VOID                   /* INTA_OFF disable
INTB_ON VOID                    /* INTB_ON disable
INTB_OFF VOID                   /* INTB_OFF disable
—
SELECT_CASE VOID                /* SELECT_CASE condition
TMOUT VOID                      /* TMOUT disable
TIMER(VOID|3)                   /* TASK3 timer_=0
PULSE_OUT VOID                  /* PULSE_OUT disable
SENSE_ON VOID                   /* SENSE_ON disable
SENSE_OFF VOID                  /* SENSE_OFF disable
CU_POST VOID                    /* CU_POST monitor
```

VOID_U

パルス発生　　予約定数

■書 式

VOID_U

■使い方

movl P(1) VOID_U

■機 能

無効軸指定

■解 説

対象ボード : MPG-2314
U 軸を除く軸を指定
X_A|Y_A|Z_A と同じ

```
MOVL P(1) VOID_X                /* X axis doesn't move
```

```
MOVL P(2) VOID_Z          /* Z axis doesn't move
JUMP P(3) VOID_Z          /* It stops right above the P(3)
```

VOID_X

パルス発生　　予約定数

■書 式
VOID_X

■使い方
movl P(1) VOID_X

■機 能
無効軸指定

■解 説
対象ボード : MPG-2314
X 軸を除く軸を指定
Y_A|U_A|Z_A と同じ
see also VOID_U

VOID_Y

パルス発生　　予約定数

■書 式
VOID_Y

■使い方
movl P(1) VOID_Y

■機 能
無効軸指定

■解 説
対象ボード : MPG-2314
Y 軸を除く軸を指定
X_A|U_A|Z_A と同じ
see also VOID_U

VOID_Z

パルス発生　　予約定数

■書 式
VOID_Z

■使い方
movl P(1) VOID_Z

■機 能
無効軸指定

■解 説
対象ボード : MPG-2314
Z 軸を除く軸を指定
X_A|Y_A|U_A と同じ
see also VOID_U

VRING

パルス発生　　予約定数

■書 式
VRING

■使い方
RANGE VRING|X_A 999

■機 能
MCX-314 の可変リング設定

■解 説
MPG-2314 の現在位置カウンタを RING カウンタに指定します。
たとえば、サンプルのように指定した場合、0 ～ 999 までは増加し、999 を超えると 0 に戻ります。
ターレット機構などで、有用な機能です。

```
RANGE VRING|X_A 999
```

WAIT

制御文　　ステートメント

■書 式
WAIT logical_eqations

■使い方
WAIT SW(0)==1
WAIT SW(-2)

■機 能
条件待ち

■解 説
条件式が 1 になるのを待ちます。
WAIT SW(0)==0 SW(0) が 0 になるのを待つ
WAIT SW(-2) SW(-2) が 1 になるのを待つ
WAIT A==100 A==100 の論理式が 1 になるのを待つ。つまり、A が 100 になるのを待つ

また、条件式には AND,OR 等の接続詞を使用できます。タイムアウトが必要な場合には、
WAIT SW(1)&SW(2) OR TIMEOUT(0)
のように、OR TIMEOUT(0) 追加するとタイムアウト処理も可能になります。
WAIT に "UNTIL" を追加すると、リアルタイム対応となります
例えば、WAIT UNTIL HSW(192)==1
という記述では、このコマンドを実行したタスクは休止状態となり、変わって OS が条件式を実行します。
タスク切り替えのタイミングに、条件式が成立していれば、休止タスクを実行状態にもどし、そのタスクに実行権を渡します。
これにより、反応速度は、起動タスク数に依存せず、タイムスライスの時間（3msec）以内となります。
ただし、UNTIL 指定の WAIT 文は OS への負担が大きく、数多くのタスクが WAIT UNTIL で条件待ちをすると、かえって処理が遅くなる場合もあります。
WAIT UNTIL は緊急性の高い条件検出に対して使用してください。

この応答時間をより高速にするには、LIFE_TIME コマンドでタイムスライスの時間を短くします。

```
10      timer_=10
20       WAIT   SW(1)&SW(2) OR TIMEOUT(0)
30       IF   TIMEOUT(0) THEN  : GOTO  *TIME_OUT : END_IF
```

WARP

パルス発生　　コマンド

■書 式

WARP [AXS] [up_z] P(n) [dwn_z] [HAND p1 p2 logic] [WHEN logic]

■使い方

WARP　1000 P(100) 500 HAND 1 VOID X(0)>2000 WHEN SW(192)==1
WARP　X_A|Y_A 0 PL(1;I_) 1000 HAND OFF|15 OFF|2 HALF_P WHEN SW(193)==0
WARP　X_A|U_A 1000 P(100) 500

■機 能

高速ゲートモーション

■解 説

アーチ状のゲートモーションです。

AXS を指定する場合は、Z 以外の不要な軸を外して動作させることができます。
up_z を指定すると、パルス分上昇してから、XY 移動を開始します。
dwn_z を指定すると、下降において最下点より dwn_z パルス分直線下降を確保します。

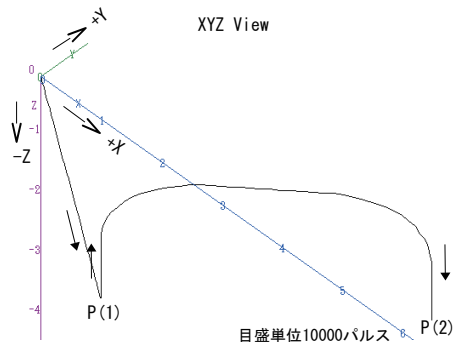
HAND 文を追加すると、条件が成立した場合に、ON/OFF 操作を行うことができます。
ON/OFF いずれかを無効にしたい場合は、引数に VOID を与えます。
また、条件に、定数 HALF_P を指定すると、XY 移動の中点で ON/OFF 操作を行います。
Z 下降が条件成立前に開始される場合は、Z 下降直前に ON/OFF 制御を実施します。
WHEN 文を追加すると、条件が成立した場合のみ下降します。

ARP動作例 (MPG-2314)

```
PG 0                              /* PGボードはMPG-2314
ACCEL ALL_A 50000 5000 500        /* 加減速
FEED ALL_A 100
SETP 1 10000 0 0 -30000           /* P(1) Z軸はマイナス方向
SETP 2 40000 25000 0 -30000       /* P(2) Z軸はマイナス方向
LIMZ -10000                       /* Z上昇制限
CLRPOS                            /* 現在位置クリア(原点復帰の代わり)
OFF 0 1

MOVL P(1)                         /* P(1)へ移動
WAIT RR(ALL_A)==0

WARP 10000 P(2) 10000             /* P(2)へWARP移動
```




```
---------------------------------------------
PG  0
ACCEL  10000
CLRPOS
ON  2
MOVS  P(1) : WAIT  RR(ALL_A)==0
WARP  100 P(5) HAND 1 2 X(0)>2000 WHEN SW(5)==1
```

WHILE-WEND

制御文　　ステートメント

■書 式

WHILE 論理式　~ WEND

■使い方

WHILE SW(0)==1
ON 0 : TIME 1 : OFF 0
WEND

■機 能

条件付無限ループ実行

■解 説

条件を定めて繰り返し実行する場合に使用します。
論理式の値が 1 である限り WHILE と WEND の間のプログラムを実行しつづけます。

Wrd

タッチパネル　　予約定数

■書 式

Wrd

■使い方

IN(-1~Wrd)

■機 能

ワード型指定

■解 説

S_MBK,MBK(),IN,OUT の読み取りを 16bit 符号なしに指定します。

```
10      S_MBK   &H00008FFF 20~Wrd /* WORD write
20      PRINT   MBK(20~Wrd)       /* unsigned WORD read
30      PRINT   MBK(20~Int)       /* signed WORD read
40      OUT   -1 -1~Wrd           /* WORD write
50      PRINT   IN(-1~Wrd)        /* unsigned WORD read
60      PRINT   IN(-1~Int)        /* signed WORD read
RUN
 36863                            /* unsigned
-28673                            /* signed
 65535                            /* unsigned
-1                                /* signed
```

WS0,WS1

IO　　関数

■書 式

WS0(arg1)

■使い方

IF WS0(0)==1 THEN : GOTO *TMOUT : END_IF

■機 能

タイムアウト付 I/O 待ち関数

■解 説

WS0(n) は SW(n) が 0 になるのを待ちますが、TMOUT で設定した待ち時間を越えると、値 1 を返します。時間内に 0 となったら、0 を返します。

WS1(n) は SW(n) が 1 になるのを待ちますが、TMOUT で設定した待ち時間を越えると、値 1 を返します。時間内に 1 となったら、0 を返します。

注)WS0,WS1 は timer_ を使用をします。このため、WS0,WS1 を実行する上位処理で timer_ を使用したタイムアップ監視を行っている場合は、WS0,WS1 内で timer_ の控えをとり、WS0,WS1 から抜け出るときに timer_ を戻します。
この為、概ね、矛盾なく動作しますが、WS0 から抜けるごとに 1 デジット (0.1 秒) 程度の誤差が生じます。

X Y Z U

パルス発生　　関数

■書 式

X(arg1)
Y(arg1)
U(arg1)
Z(arg1)

■使い方

MOVS X(1)+A VOID U(1)+B VOID
setp 1 x(0) y(0) u(0) z(0)

■機 能

現在地、および点データの座標を返す。

■解 説

arg1 が 0 の時、現在位置を返します。0 以外の数値では、指定された番号の点の座標値を返します。

XIN0

パルス発生　　予約定数

■書 式

XIN0

■使い方

HPT(XIN0)

■機 能

HPT 入力指定

■解 説

対象ボード : MPG-2314
HPT 入力ポートに XIN0 を指定します。
関連 : HOME

```
100  IF HPT(XIN0) != THEN              /* If IN0(near-org) is on
110    RMVS X_A 10000                  /* Moving to opposite direction to HOME
120  END_IF
130  WAIT RR(X_A)==0
```

XIN1

パルス発生　　予約定数

■書 式

XIN1

■使い方

HPT(XIN1)

■機 能

HPT 入力指定

■解 説

対象ボード : MPG-2314
HPT 入力ポートに XIN1 を指定します。
関連 : HOME

```
*HOME
  IF   HPT(XIN0)==1 THEN  : RMVS   X_A 5000 : END_IF
  IF   HPT(YIN0)==1 THEN  : RMVS   Y_A 5000 : END_IF
  IF   HPT(ZIN0)==1 THEN  : RMVS   Z_A -5000 : END_IF
WAIT   RR(ALL_A)==0
  SHOM   X_A|Z_A|Y_A IN0_ON
  HOME   -100000 -100000 0 100000
  WAIT   RR(ALL_A)==0
```

XMT

CUnet　　関数

■書 式

XMT(dst,arg)

■使い方

A=XMT(8,A$)
A=XMT(8,P(100))
A=XMT(16,DAT(10))

■機 能

メール送信

■解 説

XMT 関数は、RCV 関数と対で使用する、メール送信関数です。
CU_POST,POST とは併用できません。
引数に P(n),X(n) ～ Z(n),MBK(n), 配列、文字列を指定することができ、指定されたデータ 256byte を dst 宛に送信します。正常に送信が終了すると、 0 を返します。
XMT を実行する前に相手のステーションで RCV が実行されている必要があります。
0 以外の値がもどってきた場合は以下の原因による送信失敗です。

1:BIT0 送信先で RCV が実行されていない。
2:BIT1 送信先が無い
4:BIT2 メール送信通信不良
* サンプルプログラムは RCV() の項を参照してください。

X_A

パルス発生　　予約定数

■書 式

X_A

■使い方

RMVS X_A 1000

■機 能

X 軸指定

■解 説

対象ボード : MPG-2314/MPC-1200
RMVS 等の PG コマンドでの軸指定コマンドです。

```
ACCEL X_A 30000 1000 500              /* Acceleration/deceleration setting
FEED X_A 100                          /* Speed setting
INSET X_A MD_2PLS|ALM_OFF|LMT_OFF     /* In port set
SHOM X_A IN0_ON                       /* Setting Return to the Origin
MOVS X_A 1000                         /* Absolute coordinate movement
```

```
RMVS X_A 1000                                  /* Relative coordinate movement
STOP X_A STP_D                                 /* Moving stop with deceleration
WAIT RR(X_A)==0                                /* Wait until moving complete
IF LMT(X_A,LMTp)|LMT(X_A,LMTn) != THEN         /* Confirming reason for stop
   etc
```

X_C

パルス発生　　予約定数

■書 式

X_C

■使い方

stps X_C 1000

■機 能

カウンタ指定

■解 説

対象ボード : MPG-2314

X カウンタを指定します。

```
10      PG   0
20      STPS   X_C 1234              /* set the X counter
30      PRINT   X(-1)                /* display the X-counter value
#RUN

 1234
__
a=CMP_C(16,X_C)                      /* compare the COMP+ register to X counter
```

X_E

パルス発生　　予約定数

■書 式

X_E

■使い方

RR(X_E)

■機 能

X 軸エラー指定

■解 説

RR() 関数の引数として使用し、移動後の X 軸のエラーの有無を調べます。

0 でなければ、なんらかのエラー要因が発生していることを示しています。

エラーの詳細は、LMT 関数 ,PGE 関数で調査します。

対象ボード : MPG-2314

```
100  MOVS X_A 10000
110  WAIT RR(X_A)=0
120  IF RR(X_E) != THEN              /* Confirming error status
130    PRINT "ERROR STOP"
140  ELSE
150    PRINT "NORMAL STOP"
160  END_IF
170  PRX RR(X_E)
```

YIN0

パルス発生　　予約定数

■書 式
YIN0

■使い方
HPT(YIN0)

■機 能
HPT 入力指定

■解 説
対象ボード : MPG-2314
HPT 入力ポートに YIN0 を指定します。
関連 : HOME
see also XIN0

YIN1

パルス発生　　予約定数

■書 式
YIN1

■使い方
HPT(YIN1)

■機 能
HPT 入力指定

■解 説
対象ボード : MPG-2314
HPT 入力ポートに YIN1 を指定します。
関連 : HOME
see also XIN1

YPLS

パルス発生　　コマンド

■書 式
YPLS Var1 Var2 Var3 [Count]

■使い方
YPLS Port Rate Counter
YPLS Port Rate Counter 1000

■機 能
10pps ～ 5000pps の I/O パルス発生

■解 説
出力ポートによるパルス発生です。タスクとは無関係に、変数操作で、停止、速度の変更、
実行中のパルス数管理ができます。
Port,Rate,Counter は変数で直接数値を指定できません。変数に値を代入してからコマンド実行
します。

Port <= 1 or 2
MPC-2100L,2200 は 1--> 768 , 2--> 769
その他では、1--> 12 , 2--> 13

Rate <= 10 ～ 5000 (pps)
Counter は変数の指定のみです。 YPLS 起動後、発生したパルス数に応じて変化します。

サンプルプログラムでは、ポート 12 (MPC-1000 の場合) に 1kpps のパルスを出力します。
変数 C は、発生したパルス数の値となります。ポートを 2 とした場合は CCW 回転として負の値になります。
パルス発生数の上限値を決める場合は、四番目に絶対値で最大パルス数を指定します。

```
10      A=1
20      B=1000
30      C=0
40       YPLS   A B C
```

Y_A

パルス発生　　予約定数

■書 式
Y_A

■使い方
RMVS Y_A 1000

■機 能
Y 軸指定

■解 説
対象ボード : MPG-2314/MPC-1200
RMVS 等の PG コマンドでの軸指定コマンドです。
see also X_A

Y_C

パルス発生　　予約定数

■書 式
Y_C

■使い方
stps Y_C 1000

■機 能
カウンタ指定

■解 説
対象ボード : MPG-2314
Y カウンタを指定します。
see also X_C

Y_E

パルス発生　　予約定数

■書 式
Y_E

■使い方
RR(Y_E)

■機 能
Y 軸エラー指定

■解 説
RR() 関数の引数として使用し、移動後の Y 軸のエラーの有無を調べます。
0 でなければ、なんらかのエラー要因が発生していることを示しています。
エラーの詳細は、LMT 関数 ,PGE 関数で調査します。
対象ボード : MPG-2314
see also X_E

ZIN0

パルス発生 予約定数

■書 式
ZIN0

■使い方
HPT(ZIN0)

■機 能
HPT 入力指定

■解 説
対象ボード : MPG-2314
HPT 入力ポートに ZIN0 を指定します。
関連 : HOME
see also XIN0

ZIN1

パルス発生 予約定数

■書 式
ZIN1

■使い方
HPT(ZIN1)

■機 能
HPT 入力指定

■解 説
対象ボード : MPG-2314
HPT 入力ポートに ZIN1 を指定します。
関連 : HOME
see also XIN1

ZPLS

パルス発生 コマンド

■書 式
ZPLS Var1 Var2 Var3 [Count]

■使い方
ZPLS Port Rate Counter
ZPLS Port Rate Counter 1000

■機 能
10pps ～ 5000pps の I/O パルス発生

■解 説

出力ポートによるパルス発生です。タスクとは無関係に、変数操作で、停止、速度の変更、
実行中のパルス数管理ができます。
Port,Rate,Counter は変数で直接数値を指定できません。変数に値を代入してからコマンド実行
します。

Port <= 1 or 2
MPC-2100L,2200 は 1--> 770 , 2--> 771
その他では、1--> 14 , 2--> 15
Rate <= 10 ～ 5000 (pps)
Counter は変数の指定のみです。 ZPLS 起動後、発生したパルス数に応じて変化します。

サンプルプログラムでは、ポート 15 (MPC-1000 の場合) に 1kpps のパルスを出力します。
変数 C は、発生したパルス数の値ですが、A が 2 のため CCW 回転として負の値になります。
パルス発生数の上限値を決める場合は、四番目に絶対値で最大パルス数を指定します。
この例では 10000 パルス出力で停止し A の値を 0 にクリアします。

```
10      A=2
20      B=1000
30      C=0
40       ZPLS   A B C 10000
```

Z_A

パルス発生　　予約定数

■書 式

Z_A

■使い方

RMVS Z_A 1000

■機 能

Z 軸指定

■解 説

対象ボード : MPG-2314/MPC-1200
RMVS 等の PG コマンドでの軸指定コマンドです。
see also X_A

Z_C

パルス発生　　予約定数

■書 式

Z_C

■使い方

stps Z_C 1000

■機 能

カウンタ指定

■解 説

対象ボード : MPG-2314
Z カウンタを指定します。
see also X_C

Z_E

パルス発生　　予約定数

■書 式
　Z_E

■使い方
　RR(Z_E)

■機 能
　Z 軸エラー指定

■解 説
　RR() 関数の引数として使用し、移動後の Z 軸のエラーの有無を調べます。
　0 でなければ、なんらかのエラー要因が発生していることを示しています。
　エラーの詳細は、LMT 関数 ,PGE 関数で調査します。
　対象ボード : MPG-2314
　see also X_E

_VAR

演算　　コマンド

■書 式
　_VAR arg1 [arg2 ..]

■使い方
　*TASK
　VAR vala valb_

■機 能
　GOSUB もしくは RETURN で与えられた引数を取り出す。

■解 説
　GOSUB 文では引数を与えてサブルーチンを実行させることができます。
　_VAR はその引数を取り出して指定の変数に代入するコマンドです。
　_VAR は RETURN 文の引数も取り出すことができます。