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 個までです。
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 ～ 7000* *MPC-2100 では 16000 ポイント 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。

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
^	排他的論理積	|	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。

@

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。
　* 注 1.0908 までは、@(&h101)-->&h100 となっていましたが 1.0909 より上位ビットはマスクされます。

@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

■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適当に設定します。
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 コマンドを用いてください。

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。
(1.11_58 2009/04/30 より MPG-2314,MPG-2541 とも、なお、S 字指定しても台形加減速度と加減速時間そのものは変わりません)
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。FLOAT コマンド中でのみ、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
IF AD(0)>1000 THEN
IF AD(1,7)>500 THEN

■機 能
MPC-AD12 のデータ取得

■解 説
【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)

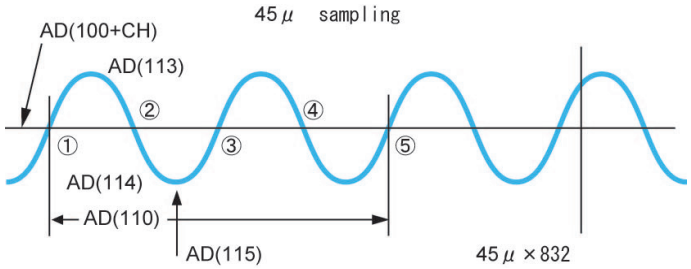
【自動連続データ取得】
MPC-AD12 は 1msec ごとにデータを取得しており、832 連続してその値を取得、参照できます。

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

【45 μ sec サンプリング】図参照
AD12 には、45 μ秒の高速サンプリング機能もあります。
AD(100+CH): 指定チャネルを 45usec 周期で 832 個データを取得。(約 37msec 間)
返される値は、最初の 400 個のデータの平均値。
データの取り出しは、AD_D(0,n) n:0 ～ 831
45 μサンプリングで 1msec ～ 20msec 程度の周期信号のサンプリングを行うと以下の結果も取得できます。



この機能は、AC 信号の分析用に組み込まれたものです。非周期・ノイズの多い信号の場合には適しません。
AD(110): 2 周期の数を返す。(平均値を境界値として① - ⑤の区間のサンプル数)
AD(112): 取得されたサンプルを AD(100+CH) の値で平均値、最大値、最小値および最小値の位置を算出。直接返す値は、平均値。
AD(113): 最大値
AD(114): 最小値
AD(115): 最小値の位置
AD(n,100+CH): n 個 45 μ秒サンプリングし、平均値を返す。(周期が判明したあとのデータ取得)

```
'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
'45usec sampling
 140    *get45
160     PRINT  " GET " AD(100) cyc=AD(110) a "AV= " AD(cyc,100) " Hz=" 2000000/(45*cyc)
190     dmy=AD(112)
200     PRINT  " MAX=" AD(113) " MIN=" AD(114)205      PRINT  "from MIN=>"
210     FOR  i=AD(115) TO AD(115)+10
```

```
220       PRINT  i AD_D(0,i)
230      NEXT
#run 140
140-
 GET  1059 357 8 AV=  874  Hz= 124 125
 MAX= 1835  MIN= 64
from MIN=>
166 64
167 111
168 192
169 269
170 341
171 410
172 476
173 539
174 600
175 657
176 712
#
```

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

■使い方
MOVS P(n) AD_P(X_A,1000)
MOVS P(n) AD_P(X_A,1000) AD_P(Z_A,-1000)

■機 能
移動点補正

■解 説
MOVS などの点データ引数 (座標値) 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。
点データそのものを修正しません。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 l angle

■使い方
AFFIN 2 1 3 i*10000

■機 能
点の回転演算

■解 説
指定点の X,Y 座標について点 n を m を中心として、angle 角度回転させ点 l の X,Y に代入します。角度は 10000 倍した値を与えます。

```
LIST
10      SETP   1 1000 2000
20      SETP   2 15142 16142
30      SETP   3 0 0
40      FOR   i=0 TO 180 STEP 15
50       AFFIN   2 1 3 i*10000
60       GETDG   1 3 B
70      A=(B+5000)/10000
80       FORMAT   " S    0"
90       PRINT   STR$(i) STR$(X(3)-X(1)) STR$(Y(3)-Y(1)) STR$(A)
100      NEXT
#run
     0  14142   14142     45
    15  10000   17320     60
    30   5176   19318     75
    45         0   20000     90
    60  - 5176   19318    105
    75  -10000   17320    120
    90  -14142   14142    135
   105  -17320   10000    150
   120  -19318    5176    165
   135  -20000          0    180
   150  -19318  - 5176    195
   165  -17320  -10000    210
   180  -14142  -14142    225
```

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-2314/2541

```
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'
```

APPEND

USB　　コマンド

■書 式
APPEND [USB] Str

■使い方
APPEND "data1.txt"
APPEND USB1 "data1.txt"

■機 能
USB データの書き込みオープン (追加書き込み)

■解 説

USB メモリに対してライト・オープンします。ファイルがあれば、追加書き込み。
なければ、新規作成し書き込みます。
書き込みは、PRINT# USB によって行います。書き込み終了時に、CLOSE を実行します。
実行中に CTRL_A が与えられた場合は、自動的に CLOSE 処理を実施します。

```
10      USB_DEL   "AA.TXT"
20      APPEND   "AA.TXT"
30      FOR   I=0 TO 10
40       PRINT#   USB "TEST=" STR$(I) "
"
50      NEXT
60      CLOSE
LIST
10      USB_DEL   "AA.TXT"
20      APPEND   "AA.TXT"
30      FOR   I=0 TO 10
40       PRINT#   USB "TEST=" STR$(I) "
"
50      NEXT
60      CLOSE
#run

A:>
#type "AA.TXT"
TEST=0
TEST=1
TEST=2
TEST=3
TEST=4
TEST=5
TEST=6
TEST=7
TEST=8
TEST=9
TEST=10

A:>
```

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 x r var [sf]

■使い方

atan 10000 1000 a
atan 100000 1000 a 173205

■機 能

ATAN 演算

■解 説

コプロセッサを用いて以下の浮動小数点演算を行います。
var = r × atan(x/sf)
注) sf を省略すると、sf を 10000 とします。

```
#atan 10000 1000 a
#pr a
 45000
#atan 100000  1000 a  173205
#pr a
 30000
#
```

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
コプロセッサを用いて以下の浮動小数点演算を行います。
var = r × atan2(y/x)
注) r を省略すると、sf を 10000 とします。

```
#atan2 100000 100000 a3
#pr a3
 450000
#atan2 100000 173205 a3 10000
#pr a3
 300000
#
```

AVOID

IO　　予約定数

■書 式
AVOID

■機 能
コマンド無効化

■解 説
コマンドの無効化。

```
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に、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、方向監視も兼ねています。

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と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。

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の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
- 実際にブレークポイントを指定して実際に 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。ENTER キィーによって、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。
- ステップ送り (一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、Enter キィーを押します。
- 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。
- 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。
- 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。
- プログラム実行を停止する場合は ‘e’ を押します。

```
 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)

■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。

CLOSE

USB　　コマンド

■書 式
CLOSE [USB]

■使い方
CLOSE
CLOSE USB1

■機 能
USB ポートを閉じる。

■解 説
APPEND,OPEN によって開かれていた USB ポートを閉じる。引数なしの場合、開かれているすべての USB ポートを閉じる。引数ありの場合 (CLOSE USB 等) は、指定ポートのみ閉じる。

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_OUTP

IO　　コマンド

■書 式
CLR_OUT arg

■使い方
CLR_OUTP 1|8
CLR_OUTP 15

■機 能
IO 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-
尚、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

```
INSET X_A CMP_CNT|PHASE1
```

see also INTA_ON

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 個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

```
INSET X_A CMP_PLS
```

see also INTA_ON

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です。
COMn = 1 MPC-2000/2100 USER ch1
COMn = 2 MPC-2100 USER ch2 (MPC2000 では欠番)
COMn = 3 MRS(DSW == 6) J4
COMn = 4 MRS(DSW == 6) J5 (RS422/485 兼用)
COMn = 5 MRS(DSW == 6) J6 (RS422/485 兼用)

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 通信が可能となる。
サンプルプログラムは CHINO 製 RS485 温度湿度計を接続した例。
1 分ごとに USB メモリにデータライトしている。
*USB メモリライトと RS485 は MRS-MCOM (20081107) にアップデート必要

```
CNFG# 5 RS485 "9600b7pes1NONE"
 FORMAT "data00.txt"
```

```
f=0
 APPEND  STR$(f)
flag=1
 GOTO *start
 DO
  WAIT  (&h00FF&TIME(0))==0
*start
  PRINT#  5 CHR$(5) "01" CHR$(2) "RPV01" CHR$(3) "
"
  INPUT#  5 a$
  PRINT  VAL(a$) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0)
  PRINT  o=VAL(10) VAL(0) h=VAL(10)
  FORMAT  ""
  PRINT#  20 HEX$(TIME(0)) " B " page " ondo=" o " hum=" h "
"
  PRINT#  5 CHR$(5) "02" CHR$(2) "RPV01" CHR$(3) "
"
  INPUT#  5 a$
  PRINT  VAL(a$) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0)
  PRINT  o=VAL(10) VAL(0) h=VAL(10)
  FORMAT  "T000"
 l$=STR$(o)
  FORMAT  "H000"
 l$=l$+STR$(h)
  PR_LCD  l$
  FORMAT  ""
  PRINT#  20 HEX$(TIME(0)) " A " page " ondo=" o " hum=" h "
"
  IF  &hFF00&TIME(0)==0 THEN
   FORMAT  "data00.txt"
   CLOSE  : f=f+1 : APPEND  STR$(f)
  END_IF
  WAIT  (&h00FF&TIME(0))!=0
 LOOP
```

==data00.TXT==

```
0173700 B 6 ondo=226 hum=366
00173700 A 6 ondo=226 hum=376
00173800 B 7 ondo=225 hum=368
00173800 A 7 ondo=226 hum=380
00173900 B 8 ondo=225 hum=365
00173900 A 8 ondo=226 hum=381
00174000 B 9 ondo=225 hum=369
00174000 A 9 ondo=226 hum=377
00174100 B 10 ondo=224 hum=353
00174100 A 10 ondo=225 hum=377
00174200 B 11 ondo=225 hum=357
00174200 A 11 ondo=224 hum=377
00174300 B 12 ondo=225 hum=356
00174300 A 12 ondo=224 hum=373
00174400 B 13 ondo=224 hum=358
00174400 A 13 ondo=224 hum=377
00174500 B 14 ondo=224 hum=358
00174500 A 14 ondo=224 hum=378
```

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 rcv$ に収納します。
COMPOWAY buff$ nod adr id rcv$

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_<>0 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

■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。

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
コプロセッサを用いて以下の浮動小数点演算を行います。
var = r × cos(deg/sf)
注) sf を省略すると、sf を 10000 とします。

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

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で、
J1-5,6 状態がオープンであれば、プログラムを再起動する (CTRL_A 0 状態 , 標準状態)
J1-5,6 の状態にかかわらず、プログラムの再起動はおこなわない (CTRL_A 1 状態 , 特殊状態)

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、ブロックこど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。

```
'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
MPC-2000 側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メールを読み取り、転送命令に従って、P(n),MBK(n) に格納します。
また、要求にもとづいて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、
そのほかは、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
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

メールの転送単位は、
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。
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、メールバッファより、データを取り出し Ary,Num の指定にしたがって書き込みます。
Cmd==1 の場合
CU_POST は、Ary,Num の指定にしたがってデータをメールバッファに書き込み、請求もとにデータを送り返します。なお、任意の引数を与えると、実行経過を表示します。

【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
```

```
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  -4 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
see also INTA_ON

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
　see also INTA_ON

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  *aho : END_IF
    PRX  DATE(0)
    WAIT  DATE(255)%16==1
```

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
FTMW のマージ機能に対応できるようになっています。

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
DIR USB1 1000

■機 能
USB メモリ (MRS-MCOM) 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$(nn+4,12) ファイル名 1
MBK$(nn+16,2) ファイル名 2

そして、以下の場所に取得データを保存します。

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

USB 番号を指定すると、DSW==6 以外の MRS-MCOM 上の USB メモリを参照できます。

DSW==6 -> USB (省略すると DSW=6 の MRS-MCOM にアクセスします。)
DSW==7 -> USB1
DSW==5 -> USB2

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,USB) などの関数によって DIR 命令実行後知ることができます。

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
1 秒カウンタを停止します。

DUMP

保守　　コマンド

■書 式
DUMP arg1
DUMP str_var

■使い方
DUMP &h200000

■機 能
メモリエリア (I/O エリア含む) を表示する。文字列をダンプ表示。

■解 説
&h600000 は、I/O エリア。以下は MPG-2541 のレジスタ状態を参照したところ。

```
#dump &h600100
00600100: 37 00 00 00 37 00 00 00|37 00 00 00 37 00 00 00
00600110: 41 41 41 41 41 41 41 41|41 41 41 41 41 41 41 41
00600120: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600130: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600140: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600150: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600160: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600170: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
```

引数に文字列を指定すると、英数字はそのまま、コントロールコードはヘキサで表示します。

```
#a$="12345"+chr$(13)+"abcde"
#pr a$
abcde5
#dump a$
12345[0D]abcde
#
```

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
　秒カウンタのカウント・イネーブル

■解 説
　指定した一秒カウンタをカウント・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$() はこの上位 1byte のコードにしたがってエラー文字列を返します。
したがって、手操作でエラー・メッセージを参照する場合は、
print err$(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
基本
FILL AHO(0) 0
AHO(0) ～すべて 0 に設定します。
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、すべてとなります。
FILL AHO(5) 10
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。

```
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

I/O　　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    SETIO
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*10000FLOAT 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)
'
```

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がありませんが、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する。
指定できるタスク番号は 1 ～ 31(ただし旧 MPC-2000 H8/300 版は 16 タスク)
起動されたタスクは END で終了するか、QUIT で強制終了する。すでに FORK しているタスクを FORK すると、2 重起動エラーとなる。QUIT してから再起動しなければならない。

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。

```
---MPC-XY03 example---
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(7) 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
```

FREEZE

編集　　コマンド

■書 式
FREEZE arg

■使い方

FREEZE 11
FREEZE 2007

■機 能

プログラムの部分固定 および固定された領域の 非表示化

■解 説

FREEZE コマンドは、プログラムトップより FREEZE_END までのプログラムを固定します。
右の例のように、110 番に FREEZE_END というコマンドがかかれています。このプログラムに freeze n（n は数値で解除のためのパスワードとなる) を実行すると、プログラム・トップから "FREEZE_END" までが固定されます。
尚、数値に図のように 1000 以上の値を与えると固定領域は非表示となります。
プログラム・セーブでも、非表示領域は保存されません固定されると NEW をしても、消えるのは 120-140 のみです。
LOAD をすると、120 番から LOAD され、固定領域は保持されます。
また、強制的に編集しようとしても無視されます。
固定の解除は、freeze n を再度実行します。n は固定した時と同じ値です。
異なった値を与えると、" すでに LOCK されています " と表示します。

注意 : FREEZE 固定を行ったプログラム・ロード、NEW では、変数領域や配列領域が初期化されません。このため、著しく異なったプログラムをいくつもロードしなおすと変数領域や配列領域が使用されないラベルで溢れていまいます。したがって、プログラム開発途上で FREEZE を実施した場合、ある程度落ち着いた時点で、一旦 FREEZE を解除し、再セーブ、再ロードをしてから、改めて FREEZE してください。

パスワードに 1000 以下の値を設定すると、表示のままプログラム固定されます。このため、すでに仕様の固まった部分のプログラム再ロードがされなくなるため、大きなプログラム開発では、ロード時間の短縮になります。
また、ケアレスミスによる不用意なプログラム変更を防ぐのにも役立ちます。

```
LIST
10    'INITIALIZE SYSTEM
20     DIM  aho(100)
30     FOR  i=0 TO 99
40    aho(i)=i
50     NEXT  i
60     FOR  i=100 TO 1000
70     SETP  i i i i i
80     NEXT  i
90    '
100    PRINT  "init"
110    FREEZE_END
120   'USER_PROGRAM_START
130    ON  1
140    JUMP  P(1)
#freeze 2007
Locked!!
110
#list
120    ON  1
140    JUMP  P(1)
#freeze 2001
Allready locked!!
#freeze 2007
Unlocked!!
```

```
#list 0
10    'INITIALIZE SYSTEM
20     DIM  aho(100)
30     FOR  i=0 TO 99
40    aho(i)=i
50     NEXT  i
60     FOR  i=100 TO 1000
70     SETP  iiiii
80     NEXT  i
90    '
100    PRINT  "init"
110    FREEZE_END
120    ON  1
140    JUMP  P(1)
#
```

FREEZE_END

編集　　コマンド

■書 式
FREEZE_END

■使い方
100 FREEZE_END

■機 能
FREEZE コマンドのためのエリア・マーク コマンドです。

■解 説
FREEZE_END コマンド自体は実行されてもなにもしません。

F_SYS

保守　　コマンド

■書 式
F_SYS [arg]

■使い方
F_SYS
F_SYS 0

■機 能
システムの切り替え

■解 説
MPC-2000 には、暫定的なシステムで稼動する Flash モードと CPU 内臓の ROM で動作する、H8rom モードがあります。
Flash モードではパワーオン時に RS-232c より 03 コードを出力して、システムローダの実行トリガとなります。
F_SYS 0 を実行すると、この 03 コード出力を抑止します。
F_SYS では H8rom モードで 03 コードは出力されます。

```
#f_sys 0
Into H8rom mode
*
Depress System_loader
#

#f_sys
Into H8rom mode
*
Enable System_loader
#
```

なお、Flash モードで実行されていたシステムを H8rom に戻す場合は、F_SYS を実行しますが、この場合プロンプトは戻りませんので、パワーオンリセットします。

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。
角度は 10000 倍された値で変数に返されます。

```
#setp 1 100 100
#setp 2 1100 1832
#getdg 1 2 a
#pr (a+500)/1000
 600
#
```

GETD_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

■機 能
AD データの連続取得

■解 説

内部 1msec タイマーを利用した AD のリアルタイムデータ取得です。
コマンドにより取得チャネル、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。
このコマンド自体は、設定のみですぐに抜け出てきますので、終了確認は、データの最後尾の確認をします。
以下のサンプルは、MPC-1000 でステップモータをまわしながら 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すま。

```
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_CODE

通信　　コマンド

■書 式

GET_CODE [val]or GET_CODE [ch str$]

■使い方

GET_CODE 1600
GET_CODE 1 a$

■機 能

G-CODE フォーマット入力

■解 説

RS-232C ポートより G-CODE を読み取ります。引数の文字列はダミーです。
GET_CODE は自動的に以下の予約変数にパラメータが格納されます。

Gc G コードの値 "G01"
Nm ラインナンバ "N1000"
Xn X 移動パルス数
Yn Y 移動パルス数
Zn Z 移動パルス数
An A 移動パルス数
In 円弧 X 中心 "I100.0"
Jn 円弧 Y 中心 "J150.0"
Mc マクロ値 "M01"
Rn 円弧半径 "R100"
Fd 送り速度 "F100.1"

その他の予約変数
Rmv G91,G90 にしたがって相対数値、絶対数値の GET_CODE 内部の演算フラグとなります。
Xcp ～ Acp 与えられた数値にしたがって算出されるリアル位置座標 例 1.23456mm -> 123456
Xmp ～ Amp 装置のパルス位置から算出された量子化位置
Xpp ～ App パルス位置

Xn ～ An は以下のように計算されます。

```
Xn = ( Xcp - Xmp ) * SF / 1000
Yn = ( Ycp - Ymp ) * SF / 1000
Zn = ( Zcp - Zmp ) * SF / 1000
An = ( Zcp - Zmp ) * SF / 1000

 Xpp += Xn  Ypp += Yn  Zpp += Zn  App += An

 Xmp = (Xpp * 1000)/SF
 Ymp = (Ypp * 1000)/SF
 Zmp = (Zpp * 1000)/SF
 Amp = (App * 1000)/SF
```

これらのパラメータを使用すると、サンプルプログラムのように NC マシンを構築する事ができます。
Xcp ～ Acp,Xpp ～ App は必要に応じて初期化が必要です。

サンプルプログラムの注意点
①送り速度を記述していません。
②連続・加減速移動の区別が曖昧です。改良が必要です。
③連続移動とするか、PtoP 移動とするかは、パルス距離、角度によって調整します。

GET_CODE 1600 *) スケールファクタの設定。1mm あたりの移動量を 100 単位で設定できます。(暫定)
GET_CODE 1 A$ *)1 は RS-CH です。

```
DIM=6999
 DIM  CODE(DIM+1) NM(DIM+1)
Rmv=0 :  count=0 :  o_ANGL=0 :  ANG=0
 PG  0 : PG  0 1 : PG  0 2
 CNFG#  1 "38400b8pns1NONE"
TAIL=1 :  TOP=1 :  count=0
 ACCEL  12000
 CLRPOS
 FORK  1 *CNC
 FORK  2 *COM
 END
*CNC
 FORMAT  "NM000000"
ACL=0
 DO
  WAIT  TAILTOP
  SELECT_CASE  CODE(TAIL)
   CASE  0
   IF  ACL==1 THEN  : EN_DACL  :  ACL=0 : END_IF
   WAIT  RR(ALL_A)==0 : RMVL  X(TAIL) Y(TAIL) 0 Z(TAIL)
   CASE  1
   RMVT  X_A|Y_A|Z_A X(TAIL) Y(TAIL) Z(TAIL)
   IF  ACL==0 THEN  : DS_DACL  :  ACL=1 : END_IF
   CASE  2
   RMVT  X_A|Y_A X(TAIL) Y(TAIL) CW U(TAIL) Z(TAIL)
   IF  ACL==0 THEN  : DS_DACL  :  ACL=1 : END_IF
   CASE  3
   RMVT  X_A|Y_A X(TAIL) Y(TAIL) CCW U(TAIL) Z(TAIL)
   IF  ACL==0 THEN  : DS_DACL  :  ACL=1 : END_IF
   CASE_ELSE
   END_SELECT
```

```
 TAIL=TAIL+1 : IF  TAIL>DIM THEN : TAIL=1 : END_IF
 count=count-1
 LC$=STR$(NM(TAIL)) : PR_LCD  LC$
  LOOP
*COM
 NNN=0
 Xcp=0 : Ycp=0 : Zcp=0
 Xpp=0 : Ypp=0 : Zpp=0
  GET_CODE  1600
  DO
   GET_CODE  1 A$
 F=0 : CD=0 : NNN=NNN+1
   IF  TOP==DIM THEN : WAIT  TAIL!=1
    ELSE
    WAIT  (TOP+1)!=TAIL
   END_IF
   IF  Gc!=VOID THEN : PRINT  "Gc=" Gc : END_IF
   IF  Nm!=VOID THEN : PRINT  "Nm=" Nm : END_IF
   IF  Xn!=VOID THEN : PRINT  "Xn=" Xn : F=X_A : Xn=0-Xn : ELSE  : Xn=0 : END_IF
   IF  Yn!=VOID THEN : PRINT  "Yn=" Yn : F=F|Y_A : ELSE  : Yn=0 : END_IF
   IF  Zn!=VOID THEN : PRINT  "Zn=" Zn : F=F|Z_A : ELSE  : Zn=0 : END_IF
   IF  Mc!=VOID THEN : PRINT  "Mc=" Mc : END_IF
   IF  Rn!=VOID THEN : PRINT  "Rn=" Rn : END_IF
   IF  Fd!=VOID THEN : PRINT  "Fd=" Fd : END_IF
   IF  In!=VOID THEN : PRINT  "In=" In : END_IF
   IF  Jn!=VOID THEN : PRINT  "Jn=" Jn : END_IF
   PRINT  "Count=" count "Rmv=" Rmv
   PRINT  "Pos P=" Xcp Ycp Zcp
   PRINT  "Pos M=" Xmp Ymp Zmp Xcp-Xmp Ycp-Ymp Zcp-Zmp
   SELECT_CASE  Gc
    CASE  VOID : CD=1
    CASE  91 : Rmv=1
    CASE  90 : Rmv=0
    CASE  0 : CD=0
    CASE  1 : CD=1
    CASE  2 : CD=3 :       'CW->CCW
    CASE  3 : CD=2 :       'CCW->CW
    CASE_ELSE
    END_SELECT
    IF  F!=0 THEN
 o_ANGL=ANG
     GOSUB  *ANGLE
 DIFF=(ANG-o_ANGL)%400 : PRINT  "DIFF=" DIFF
     IF  DIFF>10 AND CD
 CODE(TOP)=CD : IF  Nm==VOID : NM(TOP)=NNN : ELSE  : NM(TOP)=Nm : END_IF
'LINES
 X(TOP)=Xn : Y(TOP)=Yn : Z(TOP)=Zn
'CIRCLE
     IF  CD=>2 THEN : U(TOP)=XC-In : Z(TOP)=Jn-YC : END_IF
 TOP=TOP+1 : IF  TOP>DIM THEN : TOP=1 : END_IF  : count=count+1
    END_IF
    PRINT#  1 "

#" : TIME  100
   LOOP
*ANGLE
```

```
  IF  Xn==0 THEN : IF  Yn>0 THEN : ANG=100 : ELSE  : ANG=300 : END_IF  : RETURN  : END_IF
  IF  Yn==0 THEN : IF  Xn>0 THEN : ANG=0 : ELSE  : ANG=200 : END_IF  : RETURN  : END_IF
  IF  Xn>0 THEN
   IF  Yn>0 THEN : OFS=0 : ELSE  : OFS=-400 : END_IF
   ELSE
   IF  Yn>0 THEN : OFS=-200 : ELSE  : OFS=200 : END_IF
  END_IF
  GOSUB  *TAN
  RETURN
*TAN
 XA=ABS(Xn) :  YA=ABS(Yn)
  IF  XA=>YA THEN
 ANG=YA*50/XA+OFS
   ELSE
 ANG=XA*-50/YA+OFS+100
  END_IF
 ANG=ABS(ANG)
  RETURN
```

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

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

サブルーチンを呼び出す。RETURN と対で使用する。
ラベルの後ろに引数を与えると、サブルーチンに引数を引き渡すことができる。

■解 説

サブルーチンは、スタックを用いるので、GOSUB で呼ばれたプログラムは必ず RETURN でもどる必要がある。
GOTO などで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停止する。
MPC-2000 の GOSUB は、引数を渡すこともできる。サブルーチン側では、_VAR コマンドで取り出す。
_VAR コマンドの変数リストに、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
#
```

GOTO

制御文　　ステートメント

■書 式
GOTO *Label

■機 能
指定されたラベルに制御を移します。

■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
&HE1002
#
```

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。通常の IN() は読み取りを I/O エリアの入力ポートに指定すると 2 度読みベリファイするのに対して 1 度読みです。(メモリ I/O などの実入力でない領域の読み取りは、HIN,IN とも 1 度読みです。

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。
以下はタイムアウト 10 秒で原点復帰の例です。HPT(XS0),HPT(YS0),HPT(ZS0) は、各軸の S0 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。

```
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(XS0)==0 : RMVS X_A 20000 : END_IF
60 IF HPT(YS0)==0 : RMVS Y_A 20000 : END_IF
70 IF HPT(ZS0)==0 : RMVS Z_A -20000 : END_IF
80 WAIT RR(ALL_A)==0
85 TMOUT 10000
90 HOME -100000 -100000 0 100000
```

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。

```
---MPC-XY03 example---
*HOME1                  /* XY 軸原点復帰サブルーチン
 ACCEL X_A|Y_A 10000 100 100 /* 原点復帰スピード
 IF HPT(XIN0)0 THEN      /* X 軸 IN0 がオンなら退避移動
  RMVS X_A 10000
 END_IF
 IF HPT(YIN0)0 THEN      /* Y 軸 IN0 がオンなら退避移動
  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       /* 必要に応じてオフセット
 WAIT RR(ALL_A)==0
 STPS 0 0 VOID VOID       /* X,Y 軸ここを '0' にセット
 PRINT "XY HOME"
 TIME 100
 RETURN
```

HOME[MPG-2541]

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 0 -10000
HOME X_A -1000
HOME X_A|Y_A -1000

■機 能
原点復帰

■解 説
MPG-2541 の原点復帰は、IC の機能により決められています。
SD 信号がはいれば、低速になり、ORG 信号がアクティブになれば、停止します。
HOME コマンドによって与えるのは、原点復帰動作のための移動距離です。

HOME -10000 -10000 0 -10000

では、X,Y,Z 軸が CCW 方向に -10000 パルス移動します。
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。

この間に ORG がアクティブになれば、対応する軸がの座標値を 0 にして、すべての軸の動作が終了して、HOME コマンドが抜け出てきます。
抜け出たあとに、座標値が 0 になっていれば、ORG を検出したことになります。
SD,ORG の論理設定は、SHOM によって行います。

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 出力ポートの 4 ビットの一括設定です。
注意) MPG-2541 では、PG 動作中に H_ON/H_OFF/HOUT は使用できません。
ビット ON/OFF と PG コマンドが同一レジスタのため、動作状態を変更してしまうためです。

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 入力は、内部で IN1,IN2 とショートされています。このため、IN1 をオンとすると、"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。

【MPG-2541】

X_SD,ORG_X ～ Z_SD,Z_ORG をパラレルで読み取ります。
対応ビットが 1 であれば、アクティブになっているという意味です。

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 ボードの出力ポートオフ

■解 説
通常の出力ポート同様ビットオフします。
注意) MPG-2541 では、PG 動作中に H_ON/H_OFF/HOUT は使用できません。
ビット ON/OFF と PG コマンドが同一レジスタのため、動作状態を変更してしまうためです。

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 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。
注意) MPG-2541 では、PG 動作中に H_ON/H_OFF/HOUT は使用できません。
ビット ON/OFF と PG コマンドが同一レジスタのため、動作状態を変更してしまうためです。

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 の後ろへジャンプします。

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 8 ビット : パラレル取り込みです。

■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

```
---MPC-XY03 example---
DSW=IN(24)/16            /* DSW を読んで 4 ビット下にシフト
```

IN0_OFF

パルス発生　　予約定数

■書 式
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

■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

■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

■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

■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

SHOM X_A IN2_OFF
STOP X_A IN2_OFF

IN2_ON

パルス発生　　予約定数

■書 式
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

SHOM X_A IN2_ON
STOP X_A IN2_ON

IN3_OFF

パルス発生　　予約定数

■書 式
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

SHOM X_A IN3_OFF
STOP X_A IN3_OFF

IN3_ON

パルス発生　　予約定数

■書 式
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

SHOM X_A IN3_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
PG ボード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
MPG-2541
 X= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
 Y= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
 U= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
 Z= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
#
```

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。
OPEN コマンドでオープンされた USB メモリのファイルを 1 行読み取る。

■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。
なお、ポート番号 20 ～ 22 は、MRS-MCOM 上の USB メモリファイルに対応しています。
DSW==6 ->20 (省略すると DSW=6 の MRS-MCOM にアクセスします。)
DSW==7 -> 21
DSW==5 -> 22

--Serial Communication--

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

--USB Memory Access--
     OPEN  USB "AUTO.F2K"
      DO
       INPUT#  USB a$
       IF  LOF(USB)==0 THEN : BREAK  : END_IF
       PRINT  a$
      LOOP
     CLOSE  USB
```

INP_OFF

パルス発生　　予約定数

■書 式

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

■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。
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 が有効になります。
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、インポジションを有効にして 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 設定もクリア
```

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

■機 能
　ワード型指定 (符号付)

■解 説
　S_MBK,MBK(),IN,OUT のサイズを指定します。

```
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
JUMP ゲートモションの片ゲート移動です。Z 軸の下降を行いません。
JMPZ 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
JUMP はゲートモーションです。

■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。

```
---MPC-XY03 example---
/* パレット 1 の 2 番目の 500 パルス上へゲートモーション移動
   JUMP PL(1;PT) AD_P(Z_A,500)   WAIT RR(ALL_A)==0
/* メカチャック開
   OFF 14
```

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-2000/2100 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,LMTp,LMTn,SLMTp,SLMTn}
UbyteZbyteYbyteXbyte
もうひとつの参照法は、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、軸指定と参照ビットを決定しています。
【MPG-2541】
+X_LMT,-X_LMT ～ +Z_LMT,-Z_LMT をパラレル読み取りします。対応ビットが 1 になっていれば信号がアクティブです。

LMTn

パルス発生　予約定数

■書 式
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

■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

■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

■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

■機 能
ロング型 (2 ワード) 指定

■解 説
S_MBK,MBK(),IN,OUT のサイズを指定します。

```
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
```

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。

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

■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。

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

M_SW

I/O　　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 3 回、n ポートを読み取り 3 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 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。

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
タッチパネルデータをワード型で取り出す。
a=MBK(n)
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
a=MBK(n~Lng)
タッチパネルデータにワード形で代入する。
MBK(n)=a
MBK(n) を符号付整数で読み出す。たとえば値が &HFFF0 の場合 -16 となる。
b=MBK(n~Int)

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

■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

■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

■機 能
パルス出力方法設定

■解 説
対象ボード : MPG-2314
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

■機 能
パルス出力方法設定

■解 説
対象ボード : MPG-2314
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 0

■機 能
タッチパネル用 MEWNET プロトコル設定

■解 説
タスクを MEWNET(松下電工 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、使用シリアル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。(1 枚目の MRS-MCOM まで)

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-2100 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。

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

MKY(0) MKY40 の SCR の読み取り
MKY(1) MKY40 の BCR_SA の読み取り 上位 2bit は、Baud
MKY(2) MKY40 の BCR_OA の読み取り
MKY(3) MKY40 の MKY_ID の読み取り

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

MON
#mon
*0 [-1] *1 [980] *2 [1240] *3 [1320]
*4 [1360]
以下はタスク 0 を停止状態 (コマンド受付可能) にしてリアルタイムで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 で停止すると文番号が -1 となります。

#mon 1
mon 1
*1 RUNNING [850] *2 SLEEPING [1240] *3 SLEEPING [1320] *4 QUIT [1360]
*5 QUIT [-1]

尚、タスク中時間を浪費するタスクがあると以下のようにタスク番号の後ろに ! マークが表示されます。
#mon
*0 [-1] *1 [820] *2 [1240] *3 [1320]
*4 [1360] *5! [1880]
#

"MON 2" を実行すると、表示無しで、LOG データとしてのみ書き込みます。

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)

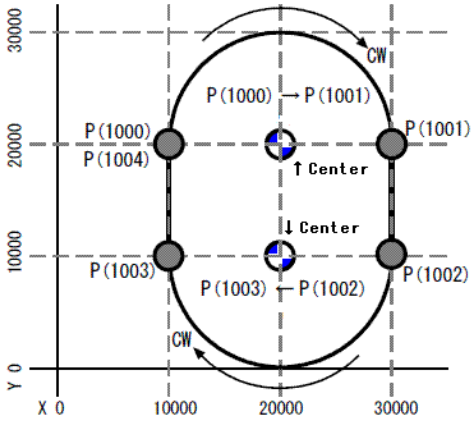
■機 能
座標値による連続補間移動

■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 にする。I/O エリアをすべて OFF. クリア状態とする。

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。
MPG 10 ～ 17 が、MPG-2541 となり 簡易位置決、補間なしです。
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。

NEG_L

パルス発生　　予約定数

■書 式
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
```

関連：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

■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 IO 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 エリアのビットオフです。

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 IO
ON 2000 // CUnet Area
ON 70000 // MBK IO 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 エリアのビットオンです。

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, 出力ポートに対しては、ON コマンドと同様、ポートを ON しますが、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に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+0.
 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、通常はプログラム実行が停止します。
ON_ERROR コマンドでは、エラー処理プログラムを定義することができ、プログラムの実行を継続させることができます。
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、MPC で発生するエラーは、致命的なものがほとんどで、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、接続デバイスの事情によりランタイム・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、プログラムを再継続することができます。
ON_ERROR から戻り処理には GOTO,RESUME があります。
GOTO の場合は、サブルーチンからのエラーかどうかの配慮が必要になります。
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 とし、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_&&HFFFFFFF --> 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

MPC-1000 の USB ポートのイネーブル・ディズエーブル

■解 説

MPC-1000 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。

OPEN

USB　　コマンド

■書 式
　OPEN [COM] str

■使い方
　OPEN A$
　OPEN USB1 "TXT.TXT"

■機 能
　USB ファイルをオープン

■解 説
　OPEN ファイル名により、以後 INPUT# USB A$
　でテキストデータを読み取ることができる。読み取るべき文字がなくなると LOF(USB) が 0 となる。
　また、INPUT# は空の文字列を返すようになります。USB# は、以下のようにアサインされています。

　DSW==6 ->USB（省略すると DSW=6 の MRS-MCOM にアクセスします。）
　DSW==7 -> USB1
　DSW==5 -> USB2

```
OPEN "TXT.TXT"
DO
IF LOF(USB)==0 THEN : END : END_IF
INPUT#  A$
PRINT A$
LOOP
```

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 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 mbk 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
PPALLET h P(i) P(j) P(k) [P(l)] m n * 0

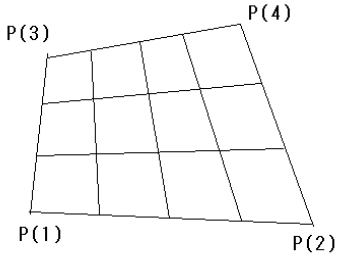
■使い方
PALLET 1 P(1) P(2) P(3) P(4) 4 3
PALLET 1 P(1) P(2) P(3) 4 3

■機 能
パレット決定

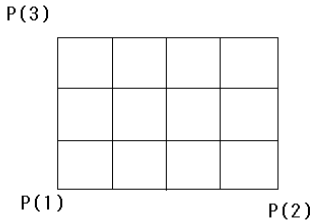
■解 説
PALLET 1 P(1) P(2) P(3) P(4) 4 3
点 p(1) ～ P(4) で生成される 4 × 3 のパレット、4 点指定の場合は、変形四角形も可。

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 1 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。




```
SETP   100 0 0 0 0
 SETP   101 20000 0 0 0
 SETP   102 20000 20000 20000 10000
 SETP   104 0 0 0 10000
 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
```



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 JUMP,JMPZ コマンド実行中であれば、再実行措置をとります。

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
MPG 10 ～ 17 が、MPG-2541 低価格 補間なし

■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 の PG 制御コマンド

■解 説
MPC-1000 の PG 機能制御コマンド。
引数に与えた文字によって PG が異なる動作する。PGA,PGB コマンドとも書式、機能は同一です。

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

C および 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、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
また以下のように軸指定と定数により入力指定を行えば個別にチェックすることができる)
PGE(X_A,LMTp)　LMTp をテスト
PGE(X_A,(IN1|LMTp))　LMTp もしくは、IN1 の双方をテスト

なお、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

■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 and MPG-2541 #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

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
倍率なし

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

PHASE2

パルス発生　　予約定数

■書 式
PHASE2

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
2 倍 (2 相パルス入力のみ)

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

PHASE4

パルス発生　　予約定数

■書 式
PHASE4

■機 能
カウンタ入力設定

■解 説
対象ボード : MPG-2314
4 倍 (2 相パルス入力のみ)

```
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あとで用います。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
```

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

dst に対してデータを転送。転送先では CU_POST コマンドが実行されていなければなりません。
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 ～ BIT3 に反映されます。
CUM_ERR

BIT7: MAIL SEND ERROR
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
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) ～ P(139) のデータが転送されます。

```
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) ～ P(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。

dst(相手アドレス) を負の数とすると、相手アドレスからのデータ転送要求となります。
POST -4 P(5)
では、SA4 に対して P(5) ～ P(19) の点データ転送を促します。
注)SA0 に対する要求は 0 のかわりに 64 を設定します。

```
10     CUNET  4 2 32
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  3 P(i)
150     IF  CUM_ERR!=0 THEN : PRINT  "X_ERR" CUM_ERR : END  : END_IF
160    NEXT
170    END
```

POS_L

パルス発生　　予約定数

■書 式
POS_L

■機 能
正の大数

■解 説
対象ボード : MPG-2314
正の大数
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
```

関連：HOME POS_L POS_L POS_L POS_L

PRA

保守　　コマンド

■書 式
PRA array(n)

■使い方
PRA AHO(10)
PRA FOOL(10,1)

■機 能
配列値の表示

■解 説
配列の中身をまとめて表示するツールです。
配列要素を 20 個ずつ表示します。2 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。プログラム中にい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#] [COMPOWAY] 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$(3)
PRINT# b$
```

【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する。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

■機 能
移動先チェック

■解 説
対象ボード : MPG-2314

```
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 LCD に表示

■解 説
与えられた文字列を LCD に 8 文字表示します。LCD で表示可能な文字は、0 ～ 9,A ～ Z と一部の記号です。小文字や複雑そうな文字は表示できません。

PR_LCD_DATE

時間管理 コマンド

■書 式
PR_LCD_DATE

■機 能
日付の LCD 表示

■解 説
搭載 RTC より日付データを取り出し LCD に表示します。サンプルプログラムでは、交互に日付と時間を LCD に表示します。

```
10          DO
20           PR_LCD_TIME
30           TIME  1000
40           PR_LCD_DATE
50           TIME  1000
60          LOOP
```

PR_LCD_TIME

時間管理 コマンド

■書 式
PR_LCD_TIME

■機 能
現在時間の LCD 表示

■解 説

搭載 RTC より時間データを取り出し LCD に表示します。
サンプルプログラムでは、交互に日付と時間を LCD に表示します。

```
10          DO
20           PR_LCD_TIME
30           TIME  1000
40           PR_LCD_DATE
50           TIME  1000
60          LOOP
#
```

PTR$

文字列　　関数

■書 式

PTR$(m)

■使い方

ptr_=a$
ptr_=ptr_+10
k$=PTR$(5)

■機 能

ポインタの位地から m 文字

■解 説

ポインタの位地から m 文字の文字列を生成します。文字列の切り出しが容易です。
ポインタ位置は ptr_ に反映されるため、この値を操作することによって文字列の切り出し位置を変更できます。
ptr_ は
ptr_=a$,SERCH コマンドで初期化されます。

```
10    a$="DATA X=AB0.4  Y=CD45 TEMP=DE55"
20     SERCH  a$ "X="
30    ptr_=ptr_-2
40    c$=PTR$(7)
50     PRINT  c$
#run

X=AB0.4
#

---MPC-XY03 example---
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 "(2)" TT$ "->" CL$       /* TT$: 元の文字列 CL$: 合成後の文字列

*RUN 結果
(2) 00123400 -> 12:34:00
```

ptr_

文字列　　予約変数

■書 式
ptr_

■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、inetrval は、0.1 秒単位設定。PULSE_OUT 0 10 ではポート 0 を一秒間隔で ON/OFF
PULSE_OUT VOID は、設定された PULSE_OUT をキャンセルする。
PULSE_OUT 32767 は 設定された PULSE_OUT の動作を同期させる。

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。

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。

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 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。
リングカウンタとは、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
```

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かならず 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。

【例】
```
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。
RMVS 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。

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。

RMVT

パルス発生　　コマンド

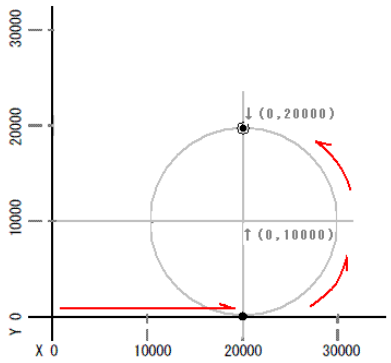
■書 式
RMVT axs arg1 arg2 [CCW|CW|0 cent1 cent2]

■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 の割り込みフラグ読み取りおよびフラグ解除

■解 説
MPG-2314 で割り込みを設定すると、割り込み発生レジスタ RR3 の内容にしたがって、INT 割り込みが発生します。割り込みの解除は、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-2000/2100 で対応しているのは、bit4,bit3 のみで他の割り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。
末尾がもっと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_

■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
#
```

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
　プログラムをコンパイルの後、フラッシュ ROM に保存して実行する。
　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する。

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

■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。
関連：IN/OUT。

SA0~SA63

CUnet 予約定数

■書 式
SA0~SA63

■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。
関連：ON/OFF/SW。

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、装置の稼動時間・稼働率測定などに用います。
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) となっています。
このため直接値を見ることはできません。
print SEC(0)/65536 --> 時間
print SEC(0)/256&255 --> 分
print SEC(0)&255 --> 秒
で参照できます。

このため、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やすくなります。
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 ごとにこと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

■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
引数を与えた場合 : 数値による分類分岐
VOID の場合 : 論理による分類

■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。
【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
 ptr_=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。

```
10     a$="DATA X=AB0.4  Y=CD45 TEMP=DE55"
20      SERCH   a$ "X="
40     b$=PTR$(5)
50     ptr_=SERCH$("Y=")
60     c$=PTR$(5)
70     ptr_=SERCH$("TEMP=")
80     d$=PTR$(4)
90      PRINT   b$ c$ d$
#run

AB0.4 CD45  DE55
#
```

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 2msec*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。

例えば、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<0 THEN : d=0 : END_IF
 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_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

なお、設定された時間は、DATE(0),TIME(0) で参照します。

```
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) .....

```
FOR  i=0 TO 9
  PRINT  i ary(i)
 NEXT
 PRINT  "SHOW SFTR"
 SFTR  ary(5)  1 1->2 〜 5 -> 0 とする。
 FOR  i=0 TO 9
  PRINT  i ary(i)
 NEXT
SHOW SFTR
0 5000
1 0
2 1000
3 2000
4 3000
5 4000
6 6000
7 7000
8 8000
9 9000
#
```

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 つずつあり IN0 と IN1 に区別されています。

例 : 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。

SHOM[MPG-2541]

パルス発生 コマンド

■書 式
SHOM pat

■使い方
SHOM &HFF

■機 能
MPG-2541 の SD,ORG の論理設定

■解 説
MPG-2541 の原点復帰は、IC の機能によって固定されています。
通常、SD をニアオリジン、ORG を Z 相あるいは原点センサに接続します。
デフォルト状態では、それぞれ ON(各入力接地) で機能有効となります。
SD は SLOW_DOWN の意味で、ACCEL で設定されて最低速度となります。
ORG は検出でパルス出力停止となります。SD は、開放されると再び最高速度となります。
論理を設定する場合は、SHOM で各対応ビットで 1 をします。
例 :
SHOME 3
X_SD,X_ORG のみショートでネガティブ、オープンで有効となります。
SHOM &HFF
全 SD,ORG が論理反転となり、オープンで有効となります。

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
コプロセッサを用いて以下の浮動小数点演算を行います。
var = r × sin(deg/sf)
注) sf を省略すると、sf を 10000 とします。

```
#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

■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

■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

■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

■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
指定タスクの実行を遅くする。あるいは、TMOUT ダウンカウンタを遅くする。

■解 説
SLOW_RUN では、指定タスクの実行速度を 1 行ごとにタイマー待ちをいれて遅くします。。
この場合、タイムアウトを回避するためにタイムアウトのダウンカウンタを遅くすることもできます。

1) 例 : タスク 1 の実行ステップごとに 1000msec 時間待ちする。
SLOW_RUN 1 1000

実行中に変更できます。最初は慎重にゆっくり実行し、しだいに早くし、最後は、
SLOW_RUN 1
というようにタスク番号のみを指定すれば、タイマー待ちはなくなります。

2) SLOW_RUN TMOUT n の場合は、タイムアウトダウンカウンタを遅くします。
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、n に 100 以上の値を設定すると、nmsec こどにダウンカウントします。
SLOW_RUN TMOUT 1000
この場合、1000msec ごとのダウンカウントになるため TMOUT が 1 秒に設定されていると 10 秒のタイムアウトとなります。n を省略した場合も 1000msec が指定されます。

*SLOW_RUN での設定は、パワーオンリセットで解除されます。
(プログラム中に記述すると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

■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。
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。

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

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、ＸＹＵＺ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

■機 能
停止方法選択

■解 説
対象ボード : MPG-2314/2541
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

■機 能
停止方法選択

■解 説
対象ボード : MPG-2314/2541
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
与えられた数値の文字列を生成します。

■解 説
文字列中に数値より数字文字列を生成します。デフォルト状態では以下のように生成します。
1000
-10000
正の値の場合スペースが付加され負の場合には、- が符号として与えられます。
生成様式は FOMAT コマンドで変更する事ができます。

STRCPY

文字列 コマンド

■書 式
STRCPY src$ dst$ [m n]

■使い方
STRCPY src$ dst$
STRCPY src$ dst$ 10 5

■機 能
文字列の複写

■解 説
文字列を複写。mはソース文字列の初期位置。n は複写文字数。
a$="012345abc"
strcpy a$ c$ 5 3
であれば、c$=> "abc" となる。

```
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
#
```

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
入力ポートを読み出します。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

■使い方
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 をい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。

SYSCLK

時間管理　　予約変数

■書 式
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
10 番に 1 を設定
S_MBK 1 10
20 番に 1000000 を設定。
S_MBK 1000000 20~Lng
結果 20 番に下位ワード、21 番に上位ワードがはいる。
S_MBK a$ 100 10 文字列 ("abc" も可) を 100 番地から 10 キャラ設定 (文字列引数は + を含まない単項文字列のみ)
S_MBK arg adr count のタイプはまとめて初期化などに使用
S_MBK n とすると内容表示。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、mbk(0)-mbk(2) への書き込みは行われません。
●プログラム番号の表示について
なお、MPC-2000/2100 では、7868 ～ 7899 にリアルタイムでプログラム番号を書き込んでいます。これによりタッチパネルでその番号を表示させることができます。
しかしながら、プログラム番号が 65535 を超える場合は、プログラム番号自体がロング変数になります。この場合は、S_MBK LONG_PRG を実行します。
これ以後、プログラム番号は、7836 より 2 ワードずつ使用してロング変数としてプログラム番号を書き込みます。

TAIL

編集 コマンド

■書 式
TAIL

■使い方
#TAIL
70
#

■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用します。

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
コプロセッサを用いて以下の浮動小数点演算を行います。
var = r × tan(deg/sf)
注) sf を省略すると、sf を 10000 とします。

```
#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(arg1)

■使い方
WAIT TASK(1)!=0

■機 能
タスクの状態を参照する。

■解 説
引数は、タスク番号で結果は以下のとおり。
255 : タスクは終了しているか、QUIT されている。
1: タスクはタイマーにより待機している。
0: タスクは実行中。
引数に -1 をいれると、自己タスク番号を返す。

TASKn

マルチタスク　　予約変数

■書 式
TASKn

■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ィ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  P3
```

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 のキィーで SET コマンド設定されたインチング量ほ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
P3
```

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)

■使い方

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

■機 能

時間データをヘキサ形式で取り出す。

■解 説

ヘキサ形式で時間を得ます。引数をいれると、引数と論理積をとって値を返します。
なお、RTC 設定は SET_RTC コマンドで実施します。

```
5          *aho
10           IF  TIME(0)<&H00182800 THEN  : GOTO  *aho : END_IF
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見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_

■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

■機 能

未受信タイムアウト設定

■解 説

未受信タイムアウトを設定します。

```
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
#
```

UIN0

パルス発生　　予約定数

■書 式

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

■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

■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(arg1)

■使い方
IF USB(USB)!=1 THEN : GOTO *NOUSB : END_IF
PR USB(1,USB)-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(USB) 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。
更に、MRS-COM そのものが無いと -2 を返します。
MRS-MCOM のバージョンが対応版でないと -1 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,USB)
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
"
  TIME  100
  DIR  1000
  IF  USB(1,USB)
  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 [USB#] Str

■使い方

USB_DEL "aaa.p2k"
UDL USB1 "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 [USB#] strg

■使い方

USB_LOAD "DEMO.F2K"
USB_LOAD USB2 "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 SAVE された、プログラムデータをロードします。
FTMW は、FTM コメントを含んだプログラムをロードすることができますが、USB_LOAD にはその機能がありません。
COM ポートは、
DSW==6 -> USB (省略すると DSW=6 の MRS-MCOM にアクセスします。)
DSW==7 -> USB1
DSW==5 -> USB2

USB_PLOAD {UPL}

USB コマンド

■書 式
USB_PLOAD [USB#] str

■使い方
USB_PLOAD "PL3.P2K"
USB_PLOAD USB1 "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_PASVE [USB#] P(n) cnt Str
USB_PASVE [USB#] 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。
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(remove) コマンドでファイルを抹消します。
RM "AUTO.P2K"
USB_PASVE P(1) 2000 "AUTO.P2K"
USB_PASVE 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。

USB_WRITE {UWR}

USB　　コマンド

■書 式
USB_WRITE [USB#] Strng

■使い方
USB_WRITE "123.456
"
USB_WRIYE COM1 STR$(n)

■機 能
USB メモリに追記書き込みする。(都度オープンクローズ)

■解 説
USB メモリにアペンドライトする。
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
"
70    A$=STR$(SUM)
80      FORMAT  "APND_CNT=0000
```

```
"
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

■機 能
U 軸指定

■解 説
対象ボード : MPG-2314/2541
see also X_A

U_C

パルス発生　　予約定数

■書 式
U_C

■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

■機 能
U 軸エラー指定

■解 説
対象ボード : MPG-2314
移動後の U 軸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+

see also X_E

VAL

浮動小数点　　関数

■書 式
VAL(str)
VAL(0)

■使い方

■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

```
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
#
```

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
VAL(a$) は、文字列中から数字列を取り出してその値を得ます。
文字列中に複数の数値が含まれている場合は、連続して VAL(0) を実行すると、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 arg 倍する。
X=123.4567
というような場合は、VAL(10000) で読み取れば 1234567 という整数値が得られる。

```
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
```

VER

保守　　コマンド

■書 式
VER

■使い方
```
VER#VER
MPC-2000 1.0903 released 2007/9/21
 H8 Multi_Task_Interpreter Flash
#
```

■機 能
バージョン表示

■解 説
VER コマンドは表示と同時にすべてのタスクを停止させます。

VER$

文字列　　予約変数

■書 式
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

■機 能

入力無効
設定解除

■解 説

対象ボード : MPG-2314/2541 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

■機 能

無効軸指定

■解 説
対象ボード : MPG-2314/2541
U 軸を除く軸を指定
X_A|Y_A|Z_A と同じ

VOID_X

パルス発生　　予約定数

■書 式
VOID_X

■機 能
無効軸指定

■解 説
対象ボード : MPG-2314/2541
X 軸を除く軸を指定
Y_A|U_A|Z_A と同じ

VOID_Y

パルス発生　　予約定数

■書 式
VOID_Y

■機 能
無効軸指定

■解 説
対象ボード : MPG-2314/2541
Y 軸を除く軸を指定
X_A|U_A|Z_A と同じ

VOID_Z

パルス発生　　予約定数

■書 式
VOID_Z

■機 能
無効軸指定

■解 説
対象ボード : MPG-2314/2541
Z 軸を除く軸を指定
X_A|Y_A|U_A と同じ

VRING

パルス発生　　予約定数

■書 式
VRING

■機 能
MCX-314 の可変リング設定

■解 説
対象ボード : MPG-2314

```
RANGE VRING|X_A 1000
```

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。

```
10    timer_=10
20     WAIT   SW(1)&SW(2) OR TIMEOUT(0)
30     IF   TIMEOUT(0) THEN  : GOTO  *TIME_OUT : END_IF
```

WHILE-WEND

制御文 ... ステートメント

■書 式
WHILE arg - WEND

■使い方
WHILE SW(0)==1
ON 0 : TIME 1 : OFF 0
WEND

■機 能
WHILE logic - WEND

■解 説
条件を定めて繰り返し実行する場合。

Wrd

タッチパネル　　予約定数

■書 式
Wrd

■機 能
ワード型指定

■解 説
S_MBK,MBK(),IN,OUT のサイズを指定します。

```
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
```

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。
このため、おおむね、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

■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)0 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

■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

```
PRINT HPT(XIN1)
```

X_A

パルス発生　　予約定数

■書 式
X_A

■機 能
X 軸指定

■解 説
対象ボード : MPG-2314/2541

```
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)0 THEN  /* Confirming reason for stop
  etc
```

X_C

パルス発生　　予約定数

■書 式
X_C

■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

■機 能
　X 軸エラー指定

■解 説
　対象ボード : MPG-2314
　移動後の X 軸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  MOVS X_A 10000
110  WAIT RR(X_A)=0
120  IF RR(X_E)0 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

■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

■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

Y_A

パルス発生　　予約定数

■書 式
Y_A

■機 能
Y 軸指定

■解 説
対象ボード : MPG-2314/2541
see also X_A

Y_C

パルス発生　　予約定数

■書 式
Y_C

■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

■機 能
Y 軸エラー指定

■解 説
対象ボード : MPG-2314
移動後の Y 軸のエラーの有無を調べます。
次のビットのどれかがセットされたことを表します。
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+

see also X_E

ZIN0

パルス発生　　予約定数

■書 式
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

■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

Z_A

パルス発生 予約定数

■書 式
Z_A

■機 能
Z 軸指定

■解 説
対象ボード : MPG-2314/2541
see also X_A

Z_C

パルス発生 予約定数

■書 式
Z_C

■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

■機 能
　Z 軸エラー指定

■解 説
　対象ボード : MPG-2314
　移動後の Z 軸のエラーの有無を調べます。
　次のビットのどれかがセットされたことを表します。
　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+

　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。