第6章　コマンドリファレンス

ADVFSCの文法

MPC-684に搭載されているADVFSC（アドバンスト・フェージック）はBASICとよく似たインタプリタですが、マシン制御用として開発されたものでその文法・コマンドなどの仕様も一般的なBASICとは幾分異なっています。

使用できる文字

ADVFSCで使用できる文字は半角の英数字・記号・日本語です。コントロールコードは使用できません。ADVFSCで用いる演算子・変数名・数値としての数字は全て半角文字です。アルファベットは大文字、小文字のどちらでも使えますが、コマンド・関数は全て大文字に変換されます。日本語を使いコメントを書くと読みやすいプログラムになります。

プログラムモードとダイレクトモード

ターミナルソフトからコマンドを入力して直接MPCを動作させることができます。これをダイレクトモードといいます。コマンドの行頭に文番号をつけるとMPCはプログラムと解釈してその行をメモリーに格納します。これがプログラムモードです。ほとんどのコマンドはどちらでも使用できます。

```
#ON 0
#PRINT SW(192)
#A=10
```

直接コマンドを実行します。

```
#10 ON 0
20 PRINT SW(192)
30 A=10
```

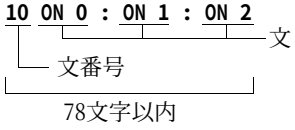
プログラムとしてメモリーに格納されます。

文番号

ADVFSCで使用できる文番号は1～65534までの整数です。プログラムは1つのタスクの中では条件分岐などが無い限り文番号の小さい順から実行されます。

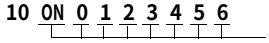
一行の文字数

一行に書くことのできる文字列は文番号やコメント文も含めて半角文字で78文字以内です。1行78文字を超えない範囲でコロン（:）で区切ることにより一行に幾つも文を書くことができます（これをマルチステートメントと言います）。



一文の文字列数

1つの文に書くことのできる文字列は8個以内です。
引き数が幾つも指定できるコマンドや、論理結合でも8個を超えることはできません。



コメント文

シングルクォート（'）をつけるとその後ろは実行されません。1行をそっくりコメントにしたり、実行文の後ろにコメントを書いたりすることができます。コメント文の文字数は半角で37文字以内です。日本語も使えます。コメント文の中にコロン（:）やカンマ（,）は使用できません。コロンはデリミタとして扱われそれ以降は別の文として解釈されます。カンマはスペースに置き換えられます。

```
10 ' この行はコメント文です。
20 ON 0 '出力0をオン
100 'コメント :ON 10
                └── コメントにならず実行されます。
```

ラベル

GOTO、GOSUBの飛び先、FORKの実行位置などは全てラベルで記述します。ラベルには次の制約があります。

- ラベルは先頭にアスタリスク（*）の付く文字列です。
- ラベルは行の先頭になくてはなりません。
- ラベルは半角37文字以内です。
- ラベルは重複はできません。

ラベルには半角英数字の他、日本語も使用できます。LISTコマンドやRUNなどの引き数にもラベルを使うことができます。VLISTコマンドで使用状況が見れます。

```
10 FORK 1 *task1
20 GOSUB *case1
1000 *case1
1010 FOR i=0 TO 255
       |
LIST *case1
```

コマンド

ADVFSCの基本はコマンドと引き数です。コマンドは、必ず大文字のアルファベットと数字になっています。小文字で入力しても自動的に大文字に変換されます。コマンドには必ず引き数が必要です。引き数の数はコマンドにより異なりますが最大で7個まで定数、変数、関数、代入式、式で与えることができます。IF～THEN～END_IF、DO～LOOP、WHILEなどの制御文が要求する引き数は条件式と呼ばれる特別な式です。

```
ON 0
ON a
ON A=B+C
PRINT SW(192)
PRINT A$
PRINT 123+&HFF
```

関　数

関数は引き数に対してある処理をして数値を与える働きをします。MPC の代表的な関数に SW(n) があります。SW(n)はnポートの状態を読み取ってON状態なら1、OFF状態なら0を与えます。関数の引き数には変数、配列変数、定数、式が使えます。

```
A=SW(0)
A=SW(B)
A=SW(B+C+D)
PRINT SW(0)
```

変数・配列変数

9文字以内の文字列で表現されます。変数の大きさは4byte整数で-2147483647～2147483647の範囲を表現することができます。変数は2000個まで定義することができます。特別な変数として配列変数があります。配列はDIMコマンドによって定義されます。確保できる配列は15配列までで、データの総和が10000個までです。DIM a(100)と定義するとa(0)～a(99)まで確保されます。変数名、配列変数名で使用できる文字は半角英数字、さらに最初の文字はアルファベットに限ります。

ローカル変数

語尾に‘！’マークを付けた変数はローカル変数として扱われます。ローカル変数とはタスク単位の変数です。同じ名前でもタスク毎に違うメモリエリアに割り当てられるので、1つのサブルーチンを複数のタスクで共有することができます。

```
A!=B!+C!                    /* ‘！’を付ければローカル変数
```

文字列変数

変数名の語尾がダラー（$）の変数は文字列変数です。文字列変数には文字列を格納することができます。文字列は80文字長で32個まで使用できます。

```
DO
  INPUT str$
  PRINT str$
LOOP
```

文字列変数名で使用できる文字は半角英数字、さらに最初の文字はアルファベットに限ります。

定　数

定数にはプログラムに記述された数値とCONSTコマンドで定数化された変数とがあります。数値としては自然数型、ヘキサ型（16進）、バイナリ型（2進）があり16という数は次の様な記述になります。

自然数型	16
ヘキサ型	&H10
バイナリ型	&B10000

定数化変数は定数をシンボルとして扱うもので、次の様に使います。

```
CONST sol 1
CONST sensor 192
```

このコマンドを実行するとその変数は定数とみなされて数値を変更しようとするとエラーメッセージが表わされます。プログラム中での再定義もできません。定数化変数名で使用できる文字は半角英数字、さらに最初の文字はアルファベットに限ります。

文字列定数

ダブルクォート（"）で囲まれた文字列は文字列定義です。

```
str$="ACCEL"
str$=str$+"CORP"
str$=""
```

変数・配列変数・文字列変数・定数・文字列定数の参照と初期化

変数・配列変数・文字列変数・定数・文字列定数は、VLISTコマンドにより使用状況を見ることができます。一度定義したものはNEWコマンドでクリアされます。プログラムのロード時にはディスクから読み込む前にNEWを実行します。

代入式

代入式の書式は、変数＝式　または　変数　または　定数　または　関数です。

```
A=&H0F|&HF0
B=C+D
E=IN(24)
```

ADVFSCでは代入式を引き数として扱うことができるため、独立した計算を減らすことができます。つまり計算をしながらコマンドを実行できるということです。例えば次のONコマンドのように、Aの値を1つずつ加算しながらポートをONすることができます。

```
  A=0
DO
  ON A=A+1
LOOP UNTIL A==10
```

式

式とは関数、定数、変数を演算子で結び付けたものをいいます。ADVFSCの演算子には次のものがあります。

演算子	演算内容	書式例
+	加算	A=B+C
-	減算	A=B-C
*	乗算	A=B*C
/	除算	A=B/C
%	剰余（モジュロ）	A=B%C
&	論理積（and）	A=B&C
¦	論理和（or）	A=B¦C
\	排他的論理和（xor）	A=B\C
;	シフト（※）	A=B;C
~	ワードロード（※）	A=B~C
,	ワードロード（※）	A=(B,C)

※印については特殊演算子の項を参照してください。

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A¦B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A\B
0	0	0
0	1	1
1	0	1
1	1	0

式には、括弧 () も許されており、通常の算術的なルールを守っています。

```
a=3000:b=4000:c=30:d=40
PRINT res=SQR(SQ(a)+SQ(b))-SQR(c*a+d*b)
```

条件式

条件式はIF文、DO～LOOP文、WHILE～WEND文で使用される条件を判断する式です。条件式はAND,OR,XORの論理結合式で結び論理演算することもできます。

演算子	書　式	内　容
==	A==B	AとBが等しい
<>	A<>B	AとBが等しくない
>=	A>=B	AがBより大きいかまたは等しい
>	A>B	AがBより大きい
<	A<B	AがBより小さい
<=	A<=B	AがBより小さいかまたは等しい

A,Bが式でも使用することができます。次は実際の使用例です。

```
① WHILE SW(i+192)==1 : WEND

② IF i>24 THEN
    WHILE SW(i+192)==0 : WEND
    ELSE
    TIME 100
  END_IF
```

```
③ IF IOR(&H1D0)<>tsk03 OR IOR (&H1D1)<>NOT(tsk03)
   GOTO *rr
   END_IF

④ t0=timer
   DO WHILE t0==timer
     SWAP
   LOOP
```

①i＋192で示されたポートがONとなっている間、無限ループになります。
②iの値によって条件分岐をしています。
③論理結合の例です。ORで結合されたどちらかの条件が成立すればGOTO *rrが実行されます。
④カレンダータイマーをポーリングして1秒待ちをしています。

ADVFSCでは数値演算の他に、文字列演算・文字列比較条件式なども扱うことができます。これらは文字列変数、文字列定数、あるいは文字列型関数と併せて使用します。

```
IF str$=="abc" THEN : GOTO *rr : END_IF
DO UNTIL A$==CHR$(&H0D)
  A$=INP$#0(1)
LOOP
```

パルス発生コマンドの停止条件として等しい場合（==）を与える時はプログラムを書く際に必ずイコール（=）を2つ並べて下さい。

```
RMOV 500 1000 2000 UNTIL SW(192)=1
                         ↓
RMOV 500 1000 2000 UNTIL SW(192)==1
```

文字列の演算

文字列の演算には加算と代入がサポートされています。また、文字列を作り出す関数もいくつかありますがこれらは文字列変数と同様に使用することができます。

```
A$="HEX=&h"+HEX$(16)
PRINT A$
HEX=&h10
```

条件式としての文字列比較も可能で次の演算ができます。

演算子	書　式	内　容
==	A$==B$	A$とB$がアスキーコードとして等しい
<>	A$<>B$	A$とB$がアスキーコードとして等しくない
>=	A$>=B$	A$がB$よりアスキーコードとして大きいかまたは等しい
>	A$>B$	A$がB$よりアスキーコードとして大きい
<	A$<B$	A$がB$よりアスキーコードとして小さい
<=	A$<=B$	A$がB$よりアスキーコードとして小さいかまたは等しい

論理結合式

演算式は次のような論理演算子を用いて複数の条件を組み合わせることができます。

```
DO
LOOP UNTIL A==1 AND B==1
```

AND	かつ
OR	または

パルス発生の停止条件では論理結合ができません。

```
MOVE 1000 1000 1000 UNTIL SW(192)==1 AND SW(-1)==1
                                     結合不可
```

予約変数

time$ date$
タッチパネル(GPシリーズ)のカレンダーの文字列です。time$が時間、data$が日付の文字列です。MBK-SH/RSでGPと接続した時のみ有効です。

```
PR time$
12:24:31
PR date$
92/11/17
```

timer　　この変数は時間を全て秒になおした数です。00:01:00の場合timerは60となります。
SYSCLK　パワーオンより5msec毎に+1される変数です。
TASKN　タスクナンバーを知る変数です。

特殊演算子

特殊なデータ操作を行う演算子です。

【 ; 】シフト　　A=B;C

Bを8ビット算術左シフトして、CのワードとORします。
用途)　MPG-314の論理演算
例)

```
#a=1;2
#prx a
 0102
#a=1;2;3;4
#prx a
 01020304
```

【 ~ 】ワードロード　　A=B~C

Cを上位ワード、Bを下位ワードに格納します。
用途)　メモリIO、MBKの型指定
例)

```
#a=1~2
#prx a
 00020001
```

【 , 】ワードロード　　A=(B,C)

Bを上位ワード、Cを下位ワードに格納します。関数や配列の二次元化に使用します。()の中のみ演算子として機能します。
用途)　MPG-314の論理演算、二次元配列
例)

```
#a=(1,2)
#prx a
 00010002
```

特殊な例ですが、次のようにも使用可能です。

```
10 a=1
20 b=2
30 c=3
40 d=4
50 prx e=((a;b),(c;d))
run
01020304
```

コマンドリファレンスの書式

ON ←(1)

I/O ←(2) **出力オン** ←(3)

■書　式
ON A1 [A2 A3 A4 A5 A6]　←(4)
　A1～A6:出力ポートナンバー

■解　説
・・・・・・・・・・・・　←(5)
関連:OFF,OUT　←(6)

(1) コマンド、関数の名前です。入力は大文字アルファベットで行います。小文字で入力しても自動的に大文字に変換されます(予約変数を除く)。
(2) コマンド、関数の分類です。
(3) コマンド、関数の機能です。
(4) 引き数の与え方を表します。[]は省略可能です。引き数が複数ある場合は1つ以上のスペースで区切ります。
(5) コマンド、関数についての内容や使用方法を解説しています。
(6) 関連するコマンド、関数です。

MPG-314のコマンドに関して

概　略

MPG-314は、制御用コンピュータMPC-684用のパルス発生ボードです。NOVA電子製MCX-314が搭載され4軸分の独立したPGと直線補間･円弧補間ドライバが一つシステムにまとめられたもので、多様かつ高度なパルス発生が可能です。従来のユーザにはMPG-68K/405ユーザが旧来のプログラムで最小限の修正でそのまま使用できるようになっています。
さらに、細かい制御が可能な専用コマンド(MPG314ネィティブコマンド)も用意されています。専用コマンドの中にはMCX-314に対するレジスタアクセスコマンドも含まれているため、MCX-314のすべての機能を引き出すことができます。MPG-314は同一システムに10枚まで使用することができます。ボードはPGコマンドで各タスクに割り当てて使用します。アドレスはボード前面のDSW1の値を10倍して&H400を加えた値です。DSW1=2の場合は&H420です。割り当て方法は次の通りです。

PG &H400　：ボードアドレス&H400(DSW1=0)のMPG-314を現在のタスクで使用します。
PG &h410 15：ボードアドレス&H410(DSW1=1)のMPG-314をタスク15で使用します。

免責事項

MPG-314を使用したプログラムは複雑かつ多様な制御に対応できますが、その多様性のため思わぬ誤作動を引き起こす可能性もあります。製作されたプログラムはすべて実行して動作を確認の上実用に供してください。コマンドの仕様書のみを根拠に、動作するであろうと机上で製作されたプログラムをそのまま適用しないでください。たとえば、エラー処理の回復プログラムはコーディングされただけで一回も実行されることなく実際の装置に組み込まれている場合があり大変危険です。また、コマンドには多様な引数入力についてバグ、弊害を発生させる可能性があります。このバグについてもユーザプログラムの試験実行によりバグ回避使用対応し、当方まで御指摘ください。当社ではバグの修正は責任を以って迅速に対応しますが、バグによる損害補償には応ずることができません。

MPG-68K互換コマンド

MPG-314は従来のソフトを変更なく使用できるようにするために、ほとんどのMPG-68Kコマンドに対応しています。このためMPG-68K(405)からMPG-314への移行は簡単です。
また、追加機能としてXYUが動作中でもZ軸の制御が独立してできること、またパルス発生中でもHPT()/X(0)～U(0)のボード上のIO関数、コマンドが使えるように改善されています。
注意事項として、エラーフラグが立つような入力を加えると、エラーメッセージを表示して制御インタプリタが停止します。このため、エラー回復操作などをソフトで対応する必要のある場合は、互換コマンドを使用しないか(専用コマンドで対応)、エラー入力を接続しないでください。

互換性の注意

- アドレスが&H400,410,420～＆H490となっています。PG &HE0などのMPG-68K/405アドレスでは使用できません。
- MPG-314のPULSEコマンドは405互換です。このため、パルスデューティの決定はできません。
- GO,RMコマンドは同時4軸スタートのパルス発生ですが、直線補間にはなりません。MPG-314では4軸直線補間がハードの制約から不可能となっています。また、GO,RMを使用する場合は4軸のACCEL設定が同一である必要があります。
- XPLS,WPLSは使用できません。しかし、ネィティブコマンドでは同等の機能を実現できます。
- HOME,HOMZコマンドはS1側に近傍センサ、S2にZ相(C相)入力を想定しています。また、近傍センサのみで原点復帰を完結する場合は、S2側のみ使用してください。これ以外の複雑な原点復帰はMPG-314専用コマンドのHOMEを参照してください。
- MPG-405にありMPG-68Kに無いコマンド、TR,SETXやACCELの405専用入力方式は使用できません。

互換性には注意しておりますが細部に仕様の相違やバグの可能性もあります。ユーザで動作を検証の上実用に供してください。

互換（従来）のパルスコマンドとの関係

次表はMPG-68K/405のパルス発生に関するコマンド・関数です。
これらはMPCのパルス制御にとって基礎となるもので、MPG-314でもこれらを使用することができます。
MPG-314専用コマンドもこれらの互換コマンドを踏まえた上で使用してください。例えば、単軸制御などMPG-314機能を使う場合には、専用コマンド･拡張ステートメントを使いますが、現在点の取得は表中のX～Zを用います。

conventional commands

ACCEL	･･加減速ﾃｰﾌﾞﾙの作成	PL1	･･ﾊﾟﾚｯﾄﾎﾟｲﾝﾄ
BSY	･･ﾊﾟﾙｽ発生状態入力	PL2	･･ﾊﾟﾚｯﾄﾎﾟｲﾝﾄ
CLRPOS	･･現在位置ｸﾘｱ	PL3	･･ﾊﾟﾚｯﾄﾎﾟｲﾝﾄ
CURPOS	･･現在位置表示	PL4	･･ﾊﾟﾚｯﾄﾎﾟｲﾝﾄ
FEDD	･･ｽﾋﾟｰﾄﾞ設定	PULSE	･･定速ﾊﾟﾙｽ発生
FEDH	･･ｽﾋﾟｰﾄﾞ設定	Q_PAUSE	･･ｸｲｯｸﾎﾟｰｽﾞ
FEDT	･･ｽﾋﾟｰﾄﾞ設定	RANGE	･･動作範囲の制限
FEDZ	･･ｽﾋﾟｰﾄﾞ設定	RM	･･4軸相対座標移動 (*1)
FEED	･･ｽﾋﾟｰﾄﾞ設定	RMOV	･･XYU軸相対座標移動
GO	･･4軸同時ﾊﾟﾙｽ発生 (*1)	RMVZ	･･Z軸相対座標移動
HOME	･･原点復帰	SET	･･ｲﾝﾁﾝｸﾞ量設定
HOMZ	･･原点復帰	SETP	･･点ﾃﾞｰﾀ設定
JMPZ	･･ｹﾞｰﾄﾓｰｼｮﾝ移動	SETPOS	･･現在位置変更
JUMP	･･ｹﾞｰﾄﾓｰｼｮﾝ移動	SHMZ	･･原点復帰ﾓｰﾄﾞ設定
LIMZ	･･ｹﾞｰﾄﾓｰｼｮﾝ規制	SHOM	･･原点復帰ﾓｰﾄﾞ設定
MOVE	･･XYU絶対座標移動	STOP	･･ﾊﾟﾙｽ発生停止
MOVZ	･･Z絶対座標移動	TEACH	･･ﾃｨｰﾁﾓｰﾄﾞ
P	･･点ﾃﾞｰﾀ	U	･･U軸点ﾃﾞｰﾀ
PALET1	･･ﾊﾟﾚｯﾄ宣言	X	･･X軸点ﾃﾞｰﾀ
PALET2	･･ﾊﾟﾚｯﾄ宣言	Y	･･Y軸点ﾃﾞｰﾀ
PALET3	･･ﾊﾟﾚｯﾄ宣言	Z	･･Z軸点ﾃﾞｰﾀ
PALET4	･･ﾊﾟﾚｯﾄ宣言	NEWP	･･点ﾃﾞｰﾀ初期化
PG	･･PG宣言	PLIST	･･点ﾃﾞｰﾀ表示
PGSEL	･･PGﾎﾞｰﾄﾞ選択	HOUT	･･ﾊﾟﾙｽﾎﾞｰﾄ汎用出力
		HPT	･･ﾊﾟﾙｽﾎﾞｰﾄﾞ原点ｾﾝｻﾎﾟｰﾄ入力

(*1)MPG-314ではDDA4軸制御がハード的に不可能なため疑似的に4軸同時パルス発生をしています。

主なコマンドの概要

詳細は各コマンド解説をご覧下さい。

PG

タスクとMPGを引当てます。タスクで実行された命令は引当てられたMPGに対して発行されます。
PGは全てのパルス発生コマンドに先立ち実行しなければなりません。
例えば、タスク0で "PG &HE4 2" または、タスク2の中で "PG &HE4" とすれば、タスク2で実行されるコマンドはボードアドレス&HE4に設定されたMPGに対して発行されます。
ボードアドレスはMPG-68K/405は &HE0 &HE4 -- 、MPG-314は &H400 &H410 -- となります。

ACCEL

加減速テーブル（最大スピード、加減速領域パルス数、立ち上がりパルス数）を作成します。
汎用コマンドでは台形駆動です。作成されたデータはMPG側で保持しており、MOVE・RMOV等の移動命令に適用されます。
MPG-314に対しては軸選択、S字加減速パラメータが拡張されています。

FEED

ACCELコマンドで設定した加減速テーブルが富士山だとすると、その何合目まで登るかを決めるのがFEEDです。
"FEED 0" で頂上まで登ります。FEEDはACCEL設定後に実行します。
MPG-314では軸指定と速度指定の分解能が拡張されています。

MOVE

XYU絶対座標移動です。最も一般的で使用頻度の高い移動コマンドです。
"MOVE 1000 2000 3000"とすれば座標 X=1000 Y=2000 U=3000 へ移動します。
パラメータには変数・定数・ポイントデータを与えることができます。
絶対座標移動の314専用コマンドとしては "MOVL、MOVS、MOVT"があります。

RMOV

XYU軸相対座標移動で、MOVEと同じく使用頻度の高いコマンドです。
パラメータで与えられたパルス数だけ現在位置から移動します。
パラメータには変数・定数を与えることができます。
相対座標移動の314専用コマンドとしては "RMVL、RMVS、RMVT"があります。

P(n),X(n),Y(n),U(n),Z(n)

点データを設定・参照する関数です。P(n)は4軸分、X(n)～Z(n)は各軸の位置を返します。
nには1～10000の点番号または0を設定します。
0は現在位置を表すもので、"nowXpos=X(0)"などと軸ごとに現在位置を変数に読み込むことができます。
現在位置の設定は"SETPOS"です。
ティーチングポイントは"PLIST(省略形PLS)"で一覧表示します。
また、X(n)～Z(n)は配列変数としても使えます(nは1～10000)。
MPG-314に対してはn=-1でエンコーダカウンタ値を返します。

TEACH（省略形T)

FTMWでティーチングするコマンドです。キーボードのX,Y,U,Zキーでジョグ移動し、Pキーで教示します。

```
   #T
    PG[0,400] X= -150 Y= 150 U= 0 Z= 0  dx= 50 dy= 50 du= 50 dz= 50
   点番号を指定して下さい   P100
```

ティーチモードはQキーで終了します。
MPGやモータドライバの動作確認にも利用できます。

HOUT

MPG-68K/405では汎用出力の制御。
MPG-314では汎用出力、ドライバSON信号の制御のほか、レジスタ制御に用いられます。

HPT(n)

MPG-68K/405では原点センサポートの読み込み、
MPG-314では原点センサ、インポジション、アラーム入力を一括で読込ます。

MPG-314 専用コマンド (314 ネィティブコマンド) 概略

MPG-314を用いて新しい装置を設計製作する場合は、ネイティィブコマンドを使用してください。任意の軸の組み合わせで二軸および三軸直線補間可能です。また、単軸パルス発生も軸ごとに独立して制御できます。ネィティィブコマンドと互換コマンドのもっとも大きな相違は、ネィティィブコマンドはコマンドの実行(MCX-314へのデータ設定)を終えるとただちにコマンドから抜け出すことです。このため、パルス発生中の条件停止や速度変更などをインタプリタ上で記述することができます。

```
10        PG &H410
20        ACCEL X_A 8000
30        STPS X_A 0
40        RMVS X_A 10000      /* MPG-314を設定するとただちにコマンドから出る
50        WAIT X(0)>5000      /*リアルタイムで現在位置の監視が可能
60        FEED X_A 200        /*途中で速度変更
70        TIME 1000
80        STOP X_A STP_D      /*途中停止
90        WAIT RR(X_A)==0     /*動作完了検出
100       CP
#run
 X= 7309 Y= 0 U= 0 Z= 0
#
```

制　約

しかし次のような使い方はハードの制約上できません。

- XY軸、ZU軸などのように二組の補間制御を同時に行うこと。同時でなければ可能です。
- 連続補間の途中で軸の組み合わせを変更すること。
- 絶対位置移動(MOVS,MOVLなど)は、軸が停止する前に実行できない。(現在位置が確定していないため)
- 移動コマンド（パルス発生）は、エラー発生時にパルス出力を停止しますが、インタプリタは実行継続となります。エラー発生時に実行をhaltするには、ERR_PAUSEコマンドを使います。

互換コマンドとネィティブコマンドの混在

MPG 互換モードコマンドと 314 ネィティブコマンド混在は可能です。MPG 互換モードコマンドから , ネィティブコマンドに移る場合はインタプリタの制御を開放しないので矛盾しませんが、ネィティブコマンドから互換モードに移る場合は条件待ちが必要です。

```
move 1000 1000 0        /* このコマンドはパルス発生終了まで抜けてこない。
movl 1000 0 0 1000      /* MOVL実行問題なし

movl 10000 10000 0 0    /* パルス発生終了を待たずに次ぎの行へ
wait rr(0)&(X_A|Y_A)=0  /* この条件待ちを外すと次のmove 0 0 0がただちに実行され
move 0 0 0                  て誤作動する。
```

ビット定数

MPC-684では、常用される機能のためコマンドを理解しやすいものにとりまとめていますが、細部の機能設定も可能なように、レジスタも直接アクセス可能としています。
また、レジスタ設定は多様なビット定数が必要となりますが、代表的なものを予約定数としてあらかじめ684に登録してあります。
次はレジスタ関連の関数・コマンドの使用例です。

- X軸のパルス発生終了を待つにはには　　WAIT RR(X_A)=0
- X軸の発生中のパルスを停止しさせるには　　STOP X_A STP_D
- Y軸の速度を変更するには　　FEED Y_A 128
- Z軸のパルス出力をパルス/方向信号方式にする　HOUT MD_DPLS (Z_A;NOP,2)

NOP	ノーオペレーション 軸選択のレジスタ設定に使用
VOID	入力無効定数
CLR_ERR (HOUT)	Wr0 コマンドエラー解除
STP_I,STP_D (STOP)	Wr0 停止コマンド
X_A,Y_A,U_A,Z_A,ALL_A	Wr0軸選択定数 ALL_AはACCEL等で全軸
CW,CCW	円弧補間方向指定　　連続補間に使用します
DS_DACL,EN_DACL	自動減速無効、有効　連続補間に使用します
IN0_ON,IN1_ON,IN2_ON, IN3_ON,IN0_OFF,IN1_OFF, IN2_OFF,IN3_OFF(STOP,HOME)	wr1の設定パラメータです。停止入力をON/OFF状態で有効にします。HOME X_A,STOP X_A のタイプのコマンドで使用します。
INP_ON,INP_OFF(INSET_314)	MCX-314/wr2 の設定パラメータです。インポジョンを有効にして ON/OFF を決定します。
ALM_ON,ALM_OFF(INSET_314)	MCX-314/wr2 の設定パラメータです。アラーム入力を有効にして ON/OFF を指定します。
LMT_ON,LMT_OFF(INSET_314)	MCX-314/wr2の設定パラメータです。リミット入力のON/OFF設定
SLMT_ON	MCX-314/wr2の設定パラメータです。RANGEコマンドによるソフトリミットの有効化定数
MD_2PLS	wr2の設定パラメータです。CW/CCWパルス発生指定
MD_DPLS	wr2の設定パラメータです。DIR,パルス発生指定

注:()中は使用コマンド

MCX-314 の重要なレジスタとアクセス方法（参考）

読み取り用

RR0: パルス発生とエラーの監視
パルス発生中の監視とエラーの監視用のレジスタです。
関数RR(0)で参照することができます。下位4ビットがビジー、次の4ビットがエラーです。
次のように使用します。エラーは次のRR1の上位8ビットの反映です。

- パルス発生の終了待ち　　WAIT RR(0)&X_A=0　もしくは WAIT RR(X_A)=0
- 他のタスクのRR0読み取り　RR(&H410)

RR1:軸毎のエラーステータス1
X,Y,U,Zそれぞれか持つステータスです。関数RR(軸定数,1)という形式で読み取ることができます。
このレジスタの上位8ビットはパルス発生がどのように終了したかを表します。
RR()関数で読み取りますが次のような使用方法となります。

- X軸リミットエラーの参照　LMT=&h3000&RR(X_A,1)

RR2:軸毎のエラーステータス2
X,Y,U,Zそれぞれが持つエラー入力の参照レジスタです。ALAMやLMT信号を直接参照するにはこのレジスタを参照します。

RR3: 軸毎のエラーステータス3
X,Y,U,Zそれぞれが持つカウンタの比較情報です。MCX-314には軸ごとに比較用レジスタと比較器があります。これにより特定の場所で正確な信号を発生させることも可能です。

設定用

WR0:コマンドレジスタ
軸選択とMCX-314に対するコマンドレジスタです。たとえば

```
HOUT X_A;STP_D
```

という記述では、X_A;STD_Dという演算によって&H0126という数値が生成されますがHOUTはこれをWR0への直接書き込みとして扱います。WR0は軸選択レジスタとしても使用されるため、次のWR1,WR2への書き込み時にもダミーライトが必要となります。

WR1:モードレジスタ1
主に IN0 ～ IN3 のドライブ停止検出信号の設定として使用します。WR1 への書き込みは次のようなフォーマットになります。

```
HOUT 2 (X_A;NOP,1)
```

2はWR2に書き込む値でIN0がLOWで停止という条件です。()のの中の二番目の引数きX_A;NOPでWR0に対して軸設定をします。,の後ろの番号がレジスタ番号です。レジスタ番号にボードアドレスを加えると任意のアドレスのMPG-314にアクセスすることができます。

WR2:モードレジスタ2
リミット検出やインポジション検出の設定を行うレジスタです。しかしながらこのレジスタにはパルス出力モードの設定ビットも含まれていますので注意して再設定してください。

```
HOUT INP_ON+MD_2PLS (X_A;NOP,2)
```

ここではINP_ON+MD_2PLSをWR2に設定する値としています。MD_2PLSはパルス出力モードをCW/CCW負論理方式(デフォルト)に設定するビット定数ですが、もしもこの値を加えておかないとパルス出力のモード設定フィルードがすべて0となり、パルス出力が正論理に変わってしまいます。(X_A;NOP,2)はWR1の場合と同様の入力でX軸選択でWR2を選んでいます。

WR3:モードレジスタ3
下位3ビットが加減速度のモード設定となります。初期値としてこのレジスタは4(自動減速、対称減速、S字加減速度有効)が設定されています。このほかにはOUT4～7の出力モード設定があり汎用出力として使用するか、位置信号の出力として使うかを決定できます。

浮動小数点演算(オプション コプロ演算)

MPC-684はオプションのコプロセッサを装着することにより浮動小数点演算(倍精度)が可能になります。しかしながらMPC-684のインタプリタ(ADVFSC)は整数形式しか取り扱うことができないため、浮動小数点演算のサポートは一般の演算から切り離された独立した形式となります。コプロ演算で必要なコマンドは次の四種類です。また、コプロという単一のIOでの計算のためマルチタスクには対応できません(複数のタスクからの同時実行はできません)。このためコマンドsetf,getfによりセマフォの獲得、開放を組み込んであります。

SETF fn1,fn2,fn3..	変数、定数内部をコプロに引き渡します。(セマフォ獲得)
GETF fn1,fn2,fn3..	コプロのデータをMPCの変数に引き渡します(セマフォ開放)
CALF	"式" コプロ演算
PRF fn1,fn2,fn3..	コプロ内部データの表示 fpの番号を指定します。

コプロ内部にはユーザで使用できる浮動小数点レジスタがFP0--FP4まであります。このFP0～FP4を変数として連続演算に使用します。SETFはデータをこのFPnにデータを引き渡すコマンドです。与えられた引数は順にFP0,FP1..に引き渡します。

```
SETF 0 3 4              /* FP0に0,FP1に3,FP2に4をセットします。
```

GETF は FPn からデータを取り出すコマンドで与えられた変数に順に FP0,FP1.. の値を取り出してセットします。この時数値は整数化され丸められます。注)

```
SETF 0 3 4 : GETF A B C : PR A B C
```

結果は0,3,4となります。

CALFは引数を文字列であらわされた計算式です。FPnおよび、作業用のメモリ変数M0-M9,N0-N9の間の演算を実行します。定数もそのまま記述することができます。括弧は一重のみ使用できます。、*/(乗除)は優先されませんので () で囲います。Fnの指定は大文字でも小文字でも有効です。

```
SETF 0 3 4 : CALF "F0=F1*F1+(F2*F2)" : GETF A : PR A
```

結果は25となります。

複数の演算を順次行う場合は、;で区切ります。

```
SETF 0 3 4 : CALF "F0=F1*F1+(F2*F2);F0=qF0" : GETF A : PR A
```

結果は5となります。

注)GETFの整数化丸めは、コプロが決定し10進数の四捨五入とは異なります。10進数の四捨五入によって整数化するには、GETF 45 A B等のように最初の引数に45を与えて下さい。

'q'は単項演算子です。単項演算子には次のようなものがあります。
単項演算子:右のレジスタに対してそれぞれの単項演算を行います。

記号	意味	表記	演算内容
s	sin	sF0	F0=sin(F0)
c	cos	cF0	F0=cos(F0)
t	tan	tF2	F2=tan(F2)
a	atan	aF3	F3=atan(F3)
q	√	qF0	F0=√F0
l	(int)	iF0	F0=int(F0)
p	π	pF6	F6=π
e	e	eF4	F4=e
z	zero	zF4	F4=0
b	n*n	bF3	F3=F3*F3
r	radian	rF4	F4=F4*π/180(fp5 使用)
d	degree	dF4	F4=F4*180/π (fp5 使用)

sin/cos などの三角関数はラジアンで扱われます。このため度での扱いが必要な場合は変換が必要になります。この時にr,dの単項演算子が必要になります。次の例ではSin(45°)を求めています。

```
SETF 0 45 : CALF "f0=rf1;f0=sf0" : PRF 0
# Fp0 7.0710678118654752e-001
```

定数の記述では17桁までの数値を扱うことができます。また少数点も記述できます。しかし、E-10という指数表現を追加することはできませんので、0.00000123などの場合は、M0=123/100000000とします。

```
calf "f0=1000.1234*10000" : getf A : pr A
```

結果は10001234です。

アキュムレータ

calfの演算は複雑な括弧や優先順位に対応できないために、演算を細かく区切る必要があります。しかし、アキュムレータという概念を用いると連続演算を記述することができます。アキュムレータは演算の途中結果がはいっているレジスタです。たとえばcalf "f0=1+2+3"の場合1+2+3は一旦アキュレータの上に保持されており、アキュムレータから結果をf0に転送します。単項演算子は単独で用いるとこのアキュムレータに作用します。アキュレータはprf 7で参照することができます。
たとえば次二つの演算は、まったく同じ意味で、ともに結果は5になります。

```
CALF "F0=q(b*3+b*4)
CALF "f0=b3+b4q"
```

下行の演算ではb3+b4と演算が完結した後に単独でqが現れています。このためqはアキュムレータの値の平方根をとります。
二次方程式2X*X+7*X+4=0の解は次のように求めることができます。F1=a,F2=b,F3=cとしています。

```
10 a=2 : b=7 : c=4
20 SETF 0 a b c
40 CALF "F0=-4*F1*F3+bF2q+(-1*F2)/2/F1"
50 PRF 0
#run
Fp0 -7.1922359359558485e-001
```

ここでは 二次方程式の一つの解F0=(-1*F2+root(F2*F2-4*F1*F3))/(2*F1)を実行しています。この式には二重括弧が含まれ CALF では計算できませんが、順序をおきなおすことにより括弧を最小限にして演算しています。注意としてbが2の場合この計算は正しい答えを返すことができません。これは平方根をとる中身が負の値となり解が虚数領域にはいるためです。平方根演算は対象が正の値であることを確認する必要があります。

計算事例

<<M変数を用いた例>>

```
40 CALF "M0=300;M1=400"
50 CALF "F3=bM0+bM1"
60 PRF 3
70 CALF "M2=b300+b400;F2=qM2+1"
80 PRF 2
```

次はsin/cosを求めるもので度－＞ラジアン変換を含みます。

```
90 'sin cos
100 SETF 0 0 30 60
110 CALF "F0=s(rF2)*1000;F1=c(rF3)*1000"
120 GETF sinv cosv
130 PRINT sinv cosv
```

次はatanでラジアン－＞度変換を含みます

```
140 ' atan
150 SETF 0 1
160 CALF "F0=d(aF1)*1000"
170 GETF atanv
180 PRINT atanv
```

次は点p(1)をp(2)基点でdeg回転する演算です。
ラジアン変換とsin/cosに括弧を用いないのはこの方が高速演算だからです。次の演算で2.3m秒です。

```
220 SETP 1 200 100
230 SETP 2 100 100
240 deg=45
250 SETF X(1) Y(1) X(2) Y(2) deg
260 CALF "M0=F0-F2;M1=F1-F3;M4=rF4c;M5=rF4s"
270 CALF "F0=M0*M4-(M1*M5)+F2"
280 CALF "F1=M0*M5+(M1*M4)+F3"
290 GETF xco yco
310 PRINT "x=" xco "y=" yco
```

組み込み関数について

MPC-684には次のような組み込み計算コマンドがあります。いずれもコプロを使用しています。

形　式	計算内容	実行例	結　果
SIN A1 A2 ADR(A3)	A3=sin(A1/10000)*A2	SIN 300000 1000 ADR(a)	a<-500
COS A1 A2 ADR(A3)	A3=cos(A1/10000)*A2	COS 600000 1000 ADR(b)	b<-500
TAN A1 A2 ADR(A3)	A3=tan(A1/10000)*A2	TAN 450000 1000 ADR(c) 1000	c<-100
ATAN A1 A2 ADR(A3)	A3=atan(A1/1000)*A2	ATAN 1000 100 ADR(d) 4500	d<-4500
ATAN2 A1 A2 ADR(A3)	A3=atan2(A1/A2)*10000	ATAN2 10000 10000 ADR(e)	e<-450000
GETDG n m ADR(A3)	A3= 成す角度(P(n)->P(m))*10000	SETP 100 0 0 0 0 SETP 101 1000 1000 0 0 GETDG 100 101 ADR(f)	f<-450000
AFFIN n m l deg	l=回転(nをm中心でdeg/10000°)	AFFIN 101 100 102 450000	p(102)<=0 1414 0 0

@

演算 **論理否定**

■書　式
@(n)
　n:変数、定数

■解　説
MPCのSW入力はビット値を返しますがNOT関数でこの論理否定をとると1は&HFFFFFFFEという値になってしまいます。@は1と0を反転する関数です。

```
pr @(1)
0
```

NOT 参照

?

I/O **ビットの読み込み(HSW代替)**

■書　式
?(n)
　n:変数、定数

■解　説
入力ポートの複雑な論理をとる場合、HSW(n)という記述は長すぎて式の長さが37文字以下という制限にすぐに抵触してしまいます。?は式を短く書くための代替関数です。
　　s1=@(?(-1)\?(-2)) は s1=&h1&NOT(HSW(-1)\HSW(-2)) と同じです

HSW 参照

ABS

演算 **絶対値**

■書　式
ABS(n)

■解　説
nの絶対値を返します。

```
#aho=ABS(-1000)
#PRINT aho
  1000
```

ACCEL

パルス（MPG-314専用） **加減速テーブルの作成**

■書　式
ACCEL [n] [s] max [long min]
 n:軸選択予約定数 X_A～Z_A もしくは論理和。省略時、全軸に対する加速度、速度設定
 s:S字加減速パラメータ -1～-100。省略時、台形制御
 max:最大スピード 1～4Mpps(円弧2M)
 long:加減速領域パルス数
 min:立ち上がりパルス数
引数省略時 設定パラメータ表示

■解　説
ACCELコマンドはこれまでmax、long、minパラメータのほかに軸選択、S字加減速の設定を追加することができます。引数を全て省略すると設定内容を表示します。ACCEL実行後はFEED=0になります。パルス発生中のACCEL変更はできません。動作不安定になります。

```
ACCEL U_A -50 10000        /* U軸、S字50%、MAX10Kpps
ACCEL X_A|U_A -1 10000     /* XとU軸を同時に設定
ACCEL ALL_A 5000           /* 全軸同じ設定
```

minは1ppsから指定可能ですが微小距離移動等で著しく遅くなる場合があるのでむやみに10pps以下に設定しないで下さい。
S字のパラメータを大きくし過ぎると加減速域が大きくなり、タクトタイム遅延の原因になります。装置の状況により異なりますが、大きくても20%～30%が妥当な範囲と思われます。
予約定数は必ず大文字。

PRSET ACCEL,FEED 参照

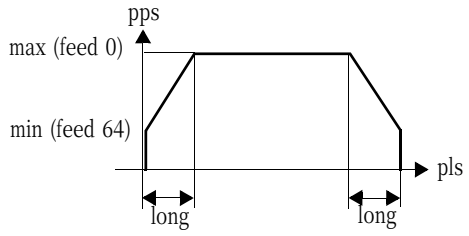
ACCEL

パルス（MPG-68K互換） **加減速テーブルの作成**

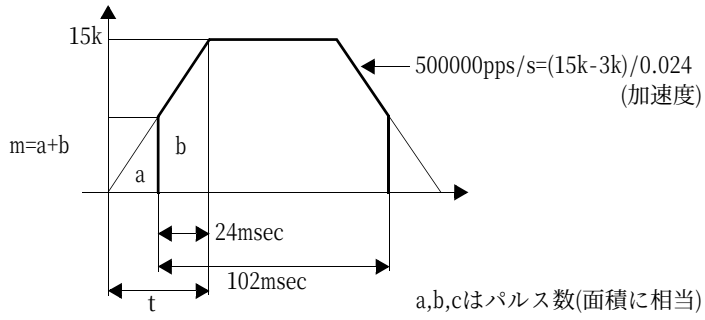
■書　式
ACCEL max [long min]
 max:最大スピード
 long:加減速領域パルス数
 min:立ち上がりパルス数
  100≦min≦max≦100000
  long≦10000
パラメーターを省略すると現在の設定値が表示されます。
MPC-683 Rev-3.53e (990702)longの10000の制約解除(405のみ)。MPG-68Kは制約有り。

■解　説
このパラメーターの与え方はMPG-68K,MPG-405のどちらにも使えます。ACCEL実行後FEED=0になります。

684のACCELでの加速距離・加速時間



次の図は3kppsでスタートして15kppsまで加速（24msec）全体は102msecの間に1250パルス移動するというものです。



例えば前記のような条件が与えられている場合にACCELコマンドには次のような計算によってパラメータを設定します。

ﾊﾟﾙｽ数mは m=t*15k/2 より 時間tは t=2m/15k

加速度は ｽﾋﾟｰﾄﾞ/時間 ですから 15k/t=15k/(2m/15k)=15k*15k/2m

加速度は500kですから 500k=15k*15k/2m

これよりtのあいだのﾊﾟﾙｽ数mは m=15k*15k/(2*500k)=225ﾊﾟﾙｽ

同様にﾊﾟﾙｽ数 a は a=3k*3k/(2*500k)=9ﾊﾟﾙｽ

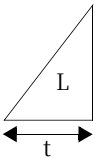
ﾊﾟﾗﾒｰﾀｰとして与えるﾊﾟﾙｽ数は b ですから b=m-a=216ﾊﾟﾙｽ

この結果よりACCELは(ACCEL 15000 216 3000)とします。このようにアクセルを設定した後で RMOV 1250 0 0とパルス発生すれば前の図のような加減速移動します。加速度と距離あるいは時間の関係は次のようになります。これは0から加速した距離Lを前提としています。等加速の場合移動距離が三角形の面積となるためこのような関係が成り立ちます。

加速度=(MAXSPD)*(MAXSPD)/(2*L)

加速時間=2*L/(MAXSPD)

移動距離=L=t*MAXSPD/2



ACCELとFEEDの計算について

ソフトウェアによる加減速テーブルの作成や中間のFEEDのテーブルには次の計算を実施しています。
ACCEL MAX L MIN を指定した場合のFEED nに対応する加速距離の計算

Ln=((128-k)*MIN+MAX*k)*L*k/(MIN+MAX)/4096 (k=64-n)

n= 0(k=64) Ln=L n=64(k=0) Ln=0

またpパルス目のタイマー値は次の式よりn(p)を求めている。

f(p)=sqrt(min^2+(max^2-min^2)*p/L)

t(p)=16000000/f(p)=22*n(p)+120

この計算の元になるのは加速度α=(MAX^2-MIN^2)/2Lであること、また距離とスピードの関係が次のようになることから得られます。

p=1/2αt^2+MIN*t

f(p)=α*t

tについて解いたpの関数をα*tに代入してf(p)を求めます。

ACCEL実行スピード実測値

ACCELコマンドの実行スピードは各パラメーターの数値により変化します。同じ最高スピードでも加速距離や立ち上がりパルス数で変わります。斜辺が滑らかな加減速にするほど演算に時間がかかります。

10 SYSCLK=0

```
20       ACCEL 50000
30         A=SYSCLK
40       ACCEL
50       PRINT A*5 "msecです"
60     '
70         SYSCLK=0
80       ACCEL 50000 10000 100
90         A=SYSCLK
100      ACCEL
110      PRINT A*5 "msecです"
120    '
130        SYSCLK=0
140      ACCEL 50000 1000 4000
150        A=SYSCLK
160      ACCEL
170      PRINT A*5 "msecです"
180    '
190        SYSCLK=0
200      ACCEL 100000
210        A=SYSCLK
220      ACCEL
230      PRINT A*5 "msecです"
240    '
250        SYSCLK=0
260      ACCEL 100000 10000 100
270        A=SYSCLK
280      ACCEL
290      PRINT A*5 "msecです"
300    '
310        SYSCLK=0
320      ACCEL 100000 1000 5000
330        A=SYSCLK
340      ACCEL
350      PRINT A*5 "msecです"
#RUN

最大スピード 50000  加速距離 2500  最小スピード 2500
805 msecです
最大スピード 50000  加速距離 10000  最小スピード 100
3155 msecです
最大スピード 50000  加速距離 1000  最小スピード 4000
335 msecです
最大スピード 100000  加速距離 5000  最小スピード 5000
1970 msecです
最大スピード 100000  加速距離 10000  最小スピード 100
4695 msecです
最大スピード 100000  加速距離 1000  最小スピード 5000
355 msecです
```

ADD_MBK

MBK-SH/RS **データインクリメント**

■書　式
ADD_MBK a dtadr
  a: 加算値
  dtadr: MBKデータアドレス

■解　説
dtadrの値にaを加算します。

```
#S_MBK 1 100
#PR MBK(100)
1
```

```
#ADD_MBK 1 100                /* DT100の値に1を足す
#PR MBK(100)
2

10 CONST counter1 100         /* カウンタの定義(ワード)
20 CONST counter2 102~Lng     /* カウンタの定義(ロング)
30 CONST counter3 104~Int     /* カウンタの定義(符号付ワード)
100 ADD_MBK 1 counter1        /* counte1 インクリメント
110 ADD_MBK 2 counter2        /* counter2 2加算
120 ADD_MBK -1 counter3       /* counter3 デクリメント
```

ADR

ユーザーコマンド **アドレス取得**

■書　式

usercom ADR(v) [A1 A2 A3]

usercom:ユーザーコマンド

v:変数・配列

A1 A2 A3:ユーザーコマンド引き数

■解　説

ADR()は変数・配列のアドレスを得る関数です。マシン語で作成したユーザーコマンドの結果をインタプリタに戻すのに使用します。次の例ではマシン語で作成したプログラムをMPC-684にダウンロードしてUSER COM0に”adrtst”として登録し、プログラムでは結果を変数dataに格納しています。

マシン語プログラム

```
      TEXT
adrtst:              次のプログラムの・・・
     movea.l D0,A0   ・・・D0に変数dataのアドレスが入る
     add.l  D2,D1    ・・・lがD1に、l*10がD2に。それを加算しD1へ
     move.l D1,(A0)  ・・・D1をD0の指すアドレス、つまり変数dataへ
     clr.l  D0
     rts
     end
----------------------------
#comset 0 "adrtst" &hbf100
#comset
       USERCOM0 -->ADRTST Entry Address->&H000BF100
10     FOR l=0 TO 5
20       adrtst ADR(data) l l*10    ・・・adrtstの結果をdataに格納
30       PRINT
40     NEXT l
RUN
0
11
22
33
44
55
```

ALT

I/O **ON/OFF反転**

■書　式

ALT A

A:出力ポートナンバー

■解　説
I/O操作コマンドでI/O反転です。ONされていたポートをOFFしOFFされていたポートをONします。

```
ALT 0 2 4
#ON 0
#OFF 2
#OFF 4
#ALT 0 2 4 <-- 0をOFF、2をON、4をON
```

1コマンドで設定できる出力は最大7つです。

AND

演算　　**論理結合式(論理積)**

■書　式
(式) AND (式)

■解　説
式と式を論理積結合します。

```
IF SW(A)==1 AND SW(B)==1 THEN
  (制御文)
END_IF

DO
  (制御文)
LOOP UNTIL A==1 AND B==1
```

パルス発生の停止条件では論理結合はできません（下線部）
MOVE -- UNTIL A==1 AND B==1

OR 参照

AR$

文字列　　**文字配列**

■書　式
AR$(n)
 n:配列数

■解　説
文字配列です。nはDIM_AR$のmで指定した値までとします（自動チェックしないので注意してください）。他の文字列と同じように扱うことができます。
【使用例】長さ35byteの文字列（Null含む）を4000個使用します。この場合、使用できる最大点番号は1250までになります。

```
10        DIM_AR$ 35 4000 :         '半角35文字 0～3999設定
20        FOR i=0 TO 3999
30            AR$(i)="ABC"+STR$(i*1000)
40        NEXT i
50        DIM_AR$  :                '確認表示
60        FOR i=0 TO 3999
70          IF (i<5) OR (i>3995) THEN
80            PRINT i AR$(i)
90          END_IF
100       NEXT i
#run
```

```
Length=35 Count =4000 P(MAX)=1250
0 ABC0
1 ABC1000
2 ABC2000
3 ABC3000
4 ABC4000
3996 ABC3996000
3997 ABC3997000
3998 ABC3998000
3999 ABC3999000
```

DIM_AR$ 参照

ASC

文字列 **文字からコードへ変換**

■書　式
ASC(s$)
　s$:文字列

■解　説
文字列の先頭文字のアスキーコードを返します。

```
#LIST
10      a$="ABC"
20    IF ASC(a$)==&H41 THEN
30      PRINT "aho"
40    END_IF
50      a$="baka"
60    PRX ASC(a$)
#run
aho
 0062 <---bのアスキーコード
#
```

ATAN

演算 **三角関数**

■書　式
ATAN A1 A2 adr(A3)

■解　説

```
A3=atan(A1/1000)*A2
```

結果は度

```
ATAN 1000 1000 ADR(A)
PR A
 45000
```

ATAN2,COS 参照

ATAN2

演算 **三角関数**

■書　式
ATAN2 A1 A2 adr(A3)

■解　説

```
A3=atan2(A1/A2)*10000
```

結果は度。

ATAN,COS 参照

BREAK

制御文 **制御フロー終了（IF文からBREAK)**

■書　式
BREAK

■解　説
IF文でのBREAK。一つ上位のIF文から直接抜け出ます。

```
IF --- THEN
  ---
ELSE
  IF b==1 THEN
    ---
    BREAK              /* ▽ﾍｼﾞｬﾝﾌﾟ
  END_IF
  ---
END_IF
(▽)
```

必ずEND_IFの直前に配置

BREAK

制御文 **制御フロー終了（繰り返し文からBREAK)**

■書　式
BREAK LOOP
BREAK WEND
BREAK NEXT

■解　説
DO-LOOP、WHILE-WEND、FOR-NEXTから抜け出ます。

```
FOR i=1 TO 1000
  j=0
  DO UNTIL j==10
    j=j+1
    IF i==500 THEN
      BREAK NEXT       /* ▽ﾍｼﾞｬﾝﾌﾟ
    END_IF
  LOOP
NEXT i
(▽)
```

引数 LOOP,NEXT,WEND は予約定数として定義されているので大文字で間違いのないように設定してください。

BSY

パルス（MPG-68K互換） **パルス発生状態入力**

■書　式
BSY (n)
 n:タスク番号
  -1,0≦n≦31

■解　説
BSY関数はMPGの動作状態を与えます。nの値が-1の場合は自己タスク、その他の値であればそれぞれのタスクのMPGの動作状態です。返される値は次の通りで、それぞれ次のような意味を持ちます。

BSY(n)==0	MPG動作中
BSY(n)==1	MPG正常停止
BSY(n)==256	MPG減速停止
BSY(n)==512	MPG即停止
BSY(n)==1024	DS_MPG

```
     PG &HE0 1                              'タスク1にPGボードを選択
     FORK 1 *task1
     DO
'
     WAIT BSY(1)==1                         'MPG動作待
'
       a=0
      DO UNTIL a==1 OR a==2
        INPUT"1か2のキーを押して下さい"a
      LOOP
      IF BSY (1)<>1 THEN
        STOP a 1
        SELECT_CASE BSY(1)                  'MPGの停止
           CASE 256 : PRINT "減速停止"
           CASE 512 : PRINT "即停止"
           CASE_ELSE  : PRINT "何かへん"
        END_SELECT
        ELSE
        PRINT "もう止まってるよ"
     END_IF
     LOOP
'
'    ============================
*task1
     ACCEL 5000
     FEED 0
     SETP 0 0 0 0 0                         '現在の点を0 0 0 0とする
     DO
      MOVE 50000 50000 50000
      TIME 2000
      MOVE 0 0 0
      TIME 2000
     LOOP
```

STOP 参照

CALF

コプロ演算 **演算**

■書　式
CALF "式"
　引数は文字列

■解　説
FPnおよび､作業用のメモリ変数M0-M9の間の演算を実行します｡定数もそのまま記述することができます｡括弧は一重のみ可｡*/(乗除)は優先されないので()で囲います｡Fnの指定は大文字でも小文字でも有効です｡

```
SETF 0 3 4
CALF "F0=F1*F1+(F2*F2)"
GETF A
PR A
  結果は25
```

複数の演算を順次行う場合は、;で区切る｡

```
SETF 0 3 4
CALF "F0=F1*F1+(F2*F2);F0=qF0"
GETF A
PR A
  結果は5
```

'q'は単項演算子｡

SETF, GETF, PRF 参照

CASE

制御文 **多値分岐**

■書　式
SELECT_CASE （省略形 SLC）
CASE
CASE_ELSE （省略形 CEL）
END_SELECT （省略形 ESL）

■解　説
CASE_ELSEも必ず記述｡

SELECT_CASE 参照

CASE_ELSE<CEL>

制御文 **多値分岐**

■書　式
SELECT_CASE （省略形 SLC）
CASE
CASE_ELSE （省略形 CEL）
END_SELECT （省略形 ESL）

■解　説
CASE_ELSEも必ず記述｡

SELECT_CASE 参照

CH_MOVS

パルス（MPG-314専用） **MOVS到達点途中変更**

■書　式
単軸指定
CH_MOVS n c
 n:軸指定予約定数 X_A～Z_A
 c:座標値

複数軸指定
CH_MOVS x y u x
 x y u x:各軸座標値。指定無=VOID予約定数。

■解　説
パルスが終了している場合には継ぎ足し移動します。MOVSと必ず対で使用します。RMVSのあとではまともに動作しません。
次記は、実験として5000パルスを超えたら行き先を変更しています。その時、既にそのポイントを超えていたり、減速域がない場合は急停止になります(①)。パルス量が伸びる場合は、減速停止になります②。

```
MOVS X_A 10000
WAIT X(0)>5000
'CH_MOVS X_A 5000  /* ①この場合、急停止になる
CH_MOVS X_A 15000  /* ②この場合、減速停止になる
WAIT RR(X_A)=0
CP
```

複数軸指定の場合は次のような記述です。

```
MOVS 10000 10000 VOID VOID
WAIT X(0)>5000
CH_MOVS 15000 13000 VOID VOID
WAIT RR(ALL_A)=0
CP
```

MOVSのS字減速中にCH_MOVSを実行しPAUSEをかけると、MPGがロック状態になり他のタスクからそのMPGへアクセス(RR()等)ができなくなります。

CHR$

文字列 **コードから文字へ変換**

■書　式
CHR$(n)
 n:1byte長数

■解　説
nをアスキーコードに見なした文字を返します。次の例ではAを表示します。

```
PRINT CHR$(&H41)
PRINT CHR$(65)
PRINT CHR$(&B01000001)
```

CHR$

MBK-SH/RS **MBKより文字列の取得**

■書　式
CHR$(dtadr,count)
 dtadr: 文字列先頭アドレス 0～7899
 count: 文字数

■解　説
MBKデータエリアの文字列を取得します。

```
#a$="漢字1234567890"
#S_MBK a$ 102 8        /* 文字列転送
#b$=CHR$(102,8)        /* アドレス102から8文字取得
#PR b$
漢字1234
```

S_MBK 参照

CLR_OUTP

I/O **領域別出力クリア**

■書　式
CLR_OUTP [n]
 n=1: MOP
  2: SLINK
  4: MBK
  8: Memory IO

■解　説
領域別に出力をクリアします。nを略すとMOP,SLINK,MBKIOエリア,MemoryIOすべてクリア。nにビット値を与えると指定された領域だけクリア。

```
CLR_OUTP 3 '---SLINKとMOPのみクリア
```

SETIO 参照
(add REV-3.82q)

CLRPOS<CLP>

パルス（MPG-68K互換） **現在位置クリア**

■書　式
CLRPOS（省略形 CLP）

■解　説
現在位置のクリアです。XYUZ各値が強制的に0とされます。

```
10   SHOM &H15 100                ’原点復帰動作設定
20   HOME &H15 100 100 100        ’メカ原点
30   MOVE 500 500 500             ’電気原点
40   CLRPOS                       ’現在点をクリアする。
50   DO
60     PRINT ”ここは”P(0)
70     MOVE 1000 1000 1000
80     PRINT ”ここは”P(0)
```

```
90      MOVE 0 0 0
100   LOOP
```

SETPOS,SETP 参照

CMN

MPC-LNK **共有変数参照**

■書　式
CLRPOS（省略形 CLP）
CMN(n [+ad])
 n:変数エリア 0～124
 +ad:リンクボードアドレス

■解　説
共有変数を参照する関数です。2バイト整数を参照する場合は注意が必用です。MPC-LNKのデータ管理はバイト単位で行われており、S_CMNによる値設定時には上位・下位が異なるタイミングで送信される場合があるためです。従ってインタロックや場合わけのような使用方法の場合は、0～255の範囲で使用します。また、CMN(n)/256として上位のみを使用することもできます。2バイトデータとして参照する時は、こうした255以内の数値を用いたフラグを確認後読み取ります。

```
WAIT CMN(12)=100
```

CMND

MPG-3202 **X3202コマンド実行**

■書　式
CMND code
 code:X3203アドレスと命令コードの和。
  &Hxx00+cmd

■解　説
MPG-3202のパルス発生IC「X3203」の命令を実行します。

```
CMND &H106              /* X3202#1のｺﾏﾝﾄﾞ6実行
```

詳細は「MPG-3202 製品別マニュアル」参照

REG,REG3,ST_REG 参照

CNFG#0

RS-232 **通信モード設定**

■書　式
CNFG#0 "[BAUD][WORD][PRTY][STOP][XCNT]"
 [BAUD] 19200,9600,4800,2400,1200,600
 [WORD] b8,b7　　（8ビット・7ビット）
 [PRTY] pe,po,pn　　（偶・奇・無）
 [STOP] s1,s2　　（1ビット・2ビット）
 [XCNT] XON,NONE　（有り・無し）
パラメーターを省略すると現在の設定値が表示されます。

■解　説

```
#cnfg#0
CNFG#0 "9600b8pns1NONE"
#cnfg#2
CNFG#2 "9600b8pns1XON"
#
10 CNFG#0 "9600b8pns1NONE"
20 CNFG#2 "9600b8pns1XON"
#
```

CNFG#n はRS-232ポートの通信条件の設定とバッファのクリアーを行います。PRINT#n の直後に CNFG#n すると送信途中でバッファの内容がクリアされてしまいます。次のプログラムの70行から *RXTASK に戻すと CNFG#n が LOOP されて不具合が生じます。

```
10      *RXTASK
20        CNFG#0 "2400b8pns1NONE"
25      *RXTASK1
30          A$="" :   B$=""
40        TIME 50
50        INPUT#0 B$
60        PRINT B$
70        IF B$=="TESTTEST" THEN : PRINT#0 "1234\n" : GOTO *RXTASK1 : END_IF
80        END
#
```

38400bpsが使えるのはMPC-684F以降。MRS-402を使う場合は必ずCNFG#nで初期化してください。

CNFG#2

RS-232　　**通信モード設定**

■書　式
CNFG#2 "[BAUD][WORD][PRTY][STOP][XCNT]"
 [BAUD] 19200,9600,4800,2400,1200,600
 [WORD] b8,b7（8ビット・7ビット）
 [PRTY] pe,po,pn（偶・奇・無）
 [STOP] s1,s2（1ビット・2ビット）
 [XCNT] XON,NONE（有り・無し）
パラメーターを省略すると現在の設定値が表示されます。

■解　説

```
CNFG#2 "9600b8pns1NONE"
```

38400bpsが使えるのはMPC-684F以降

CNFG#0 参照

CNT

デバッグ　　**実行継続**

■書　式
CNT [n]
 n:ブレークポイント

■解　説
ブレークポイントで停止したプログラムの再開。nは次のブレークポイントで略すとブレーク無しです。

RUN 参照

COMSET

ユーザーコマンド **コマンド名設定**

■書　式
COMSET n s$ [adrs]
 n:ユーザーコマンド番号
  0≦n≦9
 s$:コマンド名
 adrs:実行番地
パラメーターを省略すると現在の設定値が表示されます。

■解　説
USERCOM0～USERCOM9のコマンド名と対応するマシン語プログラムの配置位置を定めます。 adrsを省略すると、コマンド名だけが登録されマシン語プログラムは実行されません。また、COMSETすると登録されたコマンドを一覧表示します。

CONST

演算 **変数の定数化**

■書　式
CONST v n
 v:変数名(9文字以下)
 n:数値

■解　説
変数の定数化コマンド。おもにI/Oのシンボル化に使用します。定数化されるとデータを変更することはできません。変数に戻す時はNEW後プログラム再編集するかロードして下さい。

```
10   CONST SOL1 0
20   CONST SOL2 1
30   CONST SOL3 2
40   ON SOL1 SOL2 SOL3        ’変数不可の例です。
50     SOL1=1
#RUN
#50                           ・・・定数は変更できません
```

ON 0をON SOL1と記述できます。配列変数の定数化はできません。

```
10 DIM ofst(60)
20 CONST ofst(0) 200
30 CONST ofst(1) 200
RUN
#20                  ・・・引き数が適合しません
```

CONT

タスク操作 **タスク継続**

■書　式
CONT A
 A:タスク番号

■解　説
PAUSEされたタスクの再開。

```
        FORK 1 *task1
        DO
         WAIT SW(192)==1
         PAUSE 1
         WAIT SW(192)==0
         CONT 1
        LOOP
*task1
        DO
         ON 0
         TIME 100
         OFF 0
         TIME 100
        LOOP
```

FORK,PAUSE,QUIT 参照

COS

演算 **三角関数**

■書　式
COS A1 A2 adr(A3)

■解　説

ユーザコマンドを利用した、三角関数のサポート

三角関数SIN/COS/TAN/ATANをユーザコマンドとして使用することができます。この演算は68000用コンパイラのライブラリコールとなっているため、マルチタスクでは使用できません。各演算にはマルチタスクの停止が含まれていますので、注意して使用してください。引き数は整数しか扱うことができませんので、入力で10000倍の度、第二引き数を出力する数のスケールファクターとしています。

```
sin A1 A2 adr(A3)   A3=sin(A1/10000) *A2   引き数は度です
cos A1 A2 adr(A3)   A3=cos(A1/10000) *A2   引き数は度です
tan A1 A2 adr(A3)   A3=tan(A1/10000) *A2   引き数は度です
atan A1 A2 adr(A3)  A3=atan(A1/1000) *A2   結果は度です。
atan2 A1 A2 adr(A3) A3=atan2(A1/A2) *10000 結果は度です。
```

```
SIN 300000 1000 ADR(A)
PR A
 500
ATAN 1000 1000 ADR(A)
PR A
 45000
```

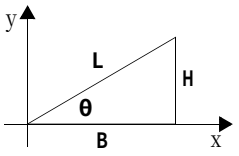
斜辺の長さと高さから角度を求めるには

```
B=sqr(L^2-H^2)
```

から

```
θ=arctan(H/sqr(L^2-H^2)
```

で得られます。MPC-684ではatan2で計算できます。



atan2 A1 A2 adr(A3)
A1:高さ
A2:斜辺の長さ
A3:戻り値が格納されている変数（単位:度）

ATAN2のプログラム例

```
10          H=10000
20       'L=14142 '45度
30       'L=11547 '60度
40          L=20000 '30度
50          X1=L*L
60          X2=H*H
70          X3=X1-X2
80          X4=SQR(X3)
90        ATAN2 H X4 ADR(A)
100       PRINT A
RUN
300007                ←10000倍されています。(単位:度)
```

ATAN2の実行スピード

```
10          H=10000
20       'L=14142 '45度
30       'L=11547 '60度
40          L=20000 '30度
50        SINGLE
60          OT=SYSCLK
70        FOR C=1 TO 1000
80            X1=L*L
90            X2=H*H
100           X3=X1-X2
110           X4=SQR(X3)
120       ATAN2 H X4 ADR(A)
130       NEXT C
140         NT=SYSCLK
150       MULTI
160       PRINT (NT-OT)*5 "msec"
170       PRINT A
RUN
5790 msec             ←結果=1000回で5.8秒
300007
```

CURPOS<CP>

パルス（MPG-68K互換） **現在位置表示**

■書　式
CURPOS　(省略形　CP)

■解　説
現在位置を表示します。現在位置の表示方法は次にもありますがCP最もシンプルです。

T　　(ティーチモードで現在位置を確認できる)
PR P(0)　(PRINT文で現在位置を表示する)
SETP 0　(SETPコマンドで0を指定する)

```
#SETP 0 100 200 300 400      現在位置の設定
#CP                          表示
X=100 Y=200 U=300 Z=400
```

date$

予約変数 **日付文字列取得**

■書　式
date$ (小文字で記述)

■解　説
MBK-SHまたはMBK-RSとデジタルGPが接続されているとGPの年月日が入ります。

```
#d$=date$
#pr d$
2001/10/23
```

DELETE<DEL>

編集 **プログラムの削除**

■書　式
DELETE n [m]
n,m:文番号

■解　説
指定番号プログラムの削除

```
DEL 20          20番削除
DEL 20 40       文番号20から文番号40を削除する
```

DIM

演算 **配列宣言**

■書　式
DIM A(n)
A:配列名
n:配列サイズ

■解　説
配列宣言。宣言可能な大きさはmモデル(出荷時)で10000データです。宣言された配列の総和がこれを越えると実行時にエラー表示されます。また、定義できる配列は15個までです。

```
DIM AHO(100)  BAKA(20)  AHO(0)～AHO(99)、BAKA(0)～BAKA(19)の配列を宣言しました。
```

DIM

演算 **配列宣言(二次元)**

■書　式
DIM A(m,n)
A:配列名
m,n:配列サイズ

■解　説
MPC-684では二次元配列宣言が可能です。

```
DIM a(2,3)
a(2,1)=100
PRINT a(2,1)
```

扱いは二次元でも中身は一次元です。

```
A(0,0)=A(0)
A(0,1)=A(1)
A(0,2)=A(2)
      |
```

DIM_AR$

文字列 **文字列配列宣言**

■書　式
DIM_AR$ n,m
 n,m:変数、定数

■解　説
文字列配列を定義します。長さnの文字列をm個P(13000)側から確保します。

```
DIM_AR$ 35 4000        /*半角35文字 0～3999設定
```

引数をいれないと現在の設定値と使用可能な点の最大指定値を表示します。

```
#DIM_AR$
Length=35 Count =4000 P(MAX)=4250
```

nはデリミタ(null)を含めた文字数。n=35なら入れることができる文字数は34文字です。

AR$ 参照

DIMCPY

文字列 **配列データのコピー**

■書　式
DIMCPY arrys(n) arryd(m) c
 arrys(n):コピー元配列、点データ。nはコピー先頭位置
 arryd(m): コピー先配列、点データ。mはコピー先頭位置
 c: コピー数 1～5000

DIMCPY var arryd(m) c
 var:変数、定数
 arryd(m): コピー先配列、点データ。mはコピー先頭位置
 c: コピー数 1～5000

■解　説
点データや配列データを一括複写します。方向はarrys(n)からarryd(m)へとなります。複写数の上限チェックは行き先配列に則してしておりませんので、不用意に大きな数をcに指定しないでください。配列や点データエリアを越えて複写するとデータが破壊されます。arryは点データを含みます。varを指定するとvarの値にしたがって同じ値で配列を埋めます。

```
DIMCPY A(1) X(10) 10     /* A(1)～A(10)の値をX(10)～X(19)にコピーする。
```

DIMCPY

MBK-SH/RS **データエリアコピー**

■書　式
DIMCPY MBK(n) MBK(m) c
MBK(n):コピー元先頭位置。0<n<8000
MBK(m):コピー先先頭位置。0<m<8000
c: コピー数。0<c<4000

■解　説
MBKデータエリアを一括複写します。方向はMBK(n)からMBK(m)へとなります。複写数の上限チェックは行き先配列に則してしておりませんので、不用意に大きな数をcに指定しないでください。

```
10        FOR i=1 TO 2000
20          S_MBK i i
30        NEXT i
40        DIMCPY MBK(1000) MBK(3000) 10
50        FOR i=0 TO 10
60          PRINT MBK(1000+i) MBK(3000+i)
70        NEXT i
80        DIMCPY MBK(1) MBK(3002) 5
90        FOR i=0 TO 10
100         PRINT MBK(1000+i) MBK(3000+i)
110       NEXT i
```

DO..LOOP

制御文 **繰り返し**

■書　式
DO [UNTIL/WHILE]
LOOP [UNTIL/WHILE]

■解　説
DOとLOOPの数は同じで対をなしていなければなりません。UNTIL、WHILEをDOの後ろに置くか、LOOPの後ろに置くかでDO～LOOPを実行する回数が違います。注意して下さい。DO～LOOP中からGOTOで外へ出るのは問題ありません。
DO～LOOP使用例

```
DO
  DO
    FOR I=0 TO 47
      ON I
      TIME 50
      OFF I
      TIME 50
      IF A==0 THEN : LOOP  : END_IF
  NEXT I
LOOP
```

LOOP,UNTIL,WHILE,BREAK 参照

DUMP

デバッグ **メモリ表示**

■書　式
DUMP [adr]
　adr:アドレス

■解　説
メモリの内容を表示します。

ELSE

制御文 **条件分岐**

■書　式
IF　条件式　THEN
　(制御文)
　ELSE
　(制御文)
END_IF

■解　説

```
IF SW(a)==1 AND SW(b)==1 THEN
  ON 0 : OFF 1
  ELSE
  OFF 0 : ON 1
END_IF
```

IF 参照

END

制御文 **プログラムの停止**

■書　式
END

■解　説
実行停止。実行中のタスクがENDにぶつかればそのタスクは自動的に停止されます。メインタスクではRUNを終了しコマンドモードとなります。ENDで停止されたプログラムはMONによる表示で「停止」と表示されます。

END_IF <EIF>

制御文 **条件分岐**

■書　式
IF　条件式　THEN
　(制御文)
　ELSE
　(制御文)
END_IF

■解　説

```
IF SW(a)==1 AND SW(b)==1 THEN
  ON 0 : OFF 1
  ELSE
  OFF 0 : ON 1
END_IF
```

IF 参照

END_SELECT <ESL>

制御文　　**多値分岐**

■書　式
SELECT_CASE（省略形 SLC）
CASE
CASE_ELSE（省略形 CEL）
END_SELECT（省略形 ESL）

■解　説
CASE_ELSEも必ず記述

SELECT_CASE 参照

ERASE

メンテナンス　　**フラッシュ ROMのプログラム消去**

■書　式
ERASE

■解　説
初期化コマンドです。フラッシュメモリ（FROM）内のプログラムをクリアします。ダイレクトコマンドのみの使用です。

```
#ERASE
*#
```

MPCINIT 参照

ERR_PAUSE

パルス（MPG-314専用）　　**エラー発生時のタスク制御**

■書　式
ERR_PAUSE n
 n:軸指定予約定数 複数監視する場合は和をとる

■解　説
エラーフラグを検出するとタスクはポーズします。ただしメインタスク(タスク0)はエラーメッセージを出力して停止します。エラーのクリアは HOUT X_A;CLR_ERR &h400 というように行います。

```
ERR_PAUSE X_A|Z_A      /*XとZのエラーを監視する。
```

FAST

制御文 **SWAP機能停止**

■書　式
FAST

■解　説
ADVFSCではマルチタクスの効率的な実行の為にIF文の条件外動作にSWAPを組み込んであります。これは、変数のチェックとIF文の組み合わせではタスクのスイッチの頻度が低下し、全体としての実行効率が下がる為です。FASTコマンドはこの機能を停止するものです。シングルタスクのみの応用の時に使用して下さい。

FCLK

システム **クロックスピード変更**

■書　式
FCLK n
　n:変数、定数

nの値	ｸﾛｯｸ(MHz)
-5	11.53 MPC-68K相当
-4	12.58
-3	13.63
-2	14.64
-1	15.73
0	16.78
1	17.82
2	18.87
3	19.92
4	20.97
5	22.02 デフォルト
6	23.06 CPU要冷却
7	24.12 CPU要冷却
684	MPC-684DIP版互換(3.84o以上)

■解　説
MPC-684のCPUクロックスピードを変更します。MPC-68Kからのプログラム移植で、実行速度に依存するタイミングがある、などの場合はクロックスピードを調整してください。電源OFFで初期状態に戻ります。

FCLK 684について
FCLKの引数に684を与えると動作速度がMPC-684(DIP)と同一になります。MPC-684(DIP)を使った古い装置に保守でMPC-684Fを入れる時の互換性の確保のためにあります。クロックは落ちませんが内部のRAMアクセス方法が変ります。

```
5 FCLK 684
```

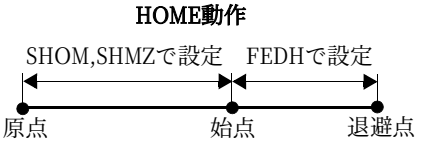
MPC-684Fは684(DIP)版に対して25%程度高速化されています。
MPC-684Fは68Kに比べて約2.2倍速です。MPC-684(DIP)版は68K比較で1.7倍速でした。
FCLK n とFCLK 684 は同時に使えます。FCLK 3：FCLK 684 とすればクロックが下がり、WAIT CYCLEは684DIPと互換になる。

FEDH

パルス（MPG-68K互換） **スピード設定**

■書　式
FEDH n
　n:スピード設定/変数・定数
　　0(最高速)≦n≦64
パラメーターを省略すると現在の設定値が表示されます。

■解　説
HOME,HOMZ コマンドでの原点復帰に先立つ原点退避移動のスピード設定です。引き数を省略すると現在の設定値が表示されます。



FEDT

パルス（MPG-68K互換） **スピード設定**

■書　式
FEDT n
　n:スピード設定/変数・定数
　　0(最高速)≦n≦64　(電源投入時 n=31)
パラメーターを省略すると現在の設定値が表示されます。

■解　説
ティーチモードでのインチング移動のスピード設定です。引き数を省略すると現在の設定値が表示されます。

FEDZ

パルス（MPG-68K互換） **スピード設定**

■書　式
FEDZ n
　n:スピード設定/変数・定数
　　0(最高速)≦n≦64
パラメーターを省略すると現在の設定値が表示されます。

■解　説
JUMPでの下降移動を除く全てのZ軸に対するスピード設定です。引き数を省略すると現在の設定値が表示されます。MPG-405はU軸に有効です。

FEED

パルス（MPG-314専用） **スピード設定**

■書　式
FEED n m
　n:軸指定予約定数 X_A～Z_A もしくは論理和
　m:速度指定 0～255

■解　説
ACCELで設定された最高スピードに対して256段階で速度を指定します。パルス発生中でも変更可能です。nを省略するとMPG-68KのFEED互換となりXYUに対する速度設定となります。ACCEL実行で初期値(0)になります。

```
FEED Z_A 128          /*Z軸を半分のスピードにします
FEED Z_A 0 X_A 32     /*連続設定もできます
FEED Z_A|X_A 0        /*論理和もできます(Z,X軸 FEED=0)
```

注意:S字を指定していると行き先の決まっているパルス発生では増速が無効になります。
予約定数は必ず大文字。

ACCEL 参照

FEED

パルス（MPG-314専用）　　スピード設定(微細設定)

■書　式
FEED n VOID|m
 n:軸指定予約定数 X_A～Z_A
 m:速度指定 1(最低)～8000(最大)

■解　説
速度指定に予約定数VOIDをorすると設定範囲は1～8000となり、MCX314の速度レジスタに直接書き込みます。ACCELの最高速度が8000以下の場合は指定値がそのままppsになります。ACCELの指定値が8000を超える場合は最高速に対して8000等分されたppsになります。

```
ACCEL 50000           /* 最高速8000超
FEED X_A VOID|4000    /*この場合25Kpps (50K/8K*4K)
```

予約定数は必ず大文字。

ACCEL 参照

FEED

パルス（MPG-68K互換）　　スピード設定

■書　式
FEED n
 n:スピード設定/変数・定数
  0(最高速)≦n≦64
パラメーターを省略すると現在の設定値が表示されます。

■解　説
MOVE,RMOV,JUMP,GO,RMの移動に対するスピード設定です。スピード設定はACCELコマンドによって作成された加減速テーブルに対するスピードテーブルを使用します。スピードテーブルとはACCELで指定されたスピード範囲について64分割し、それぞれに対応する加速距離を定義したものです。例えばACCEL　5000 1000とすれば、FEED 0では、1000の加速距離値が入っています。これはFEED 0が最大スピードを意味しており、先のACCELによればこれは1000パルスで最高のスピードに達するとしているからです。逆にFEED 64 ではその値は 1 で、加速テーブルを登らないということになります。この為、移動距離が少ない時にはFEEDをいくら設定しても目的のスピードに達しないことがあります。この例でもRMOV 1000 0　0としても最大スピードに達する前に減速に入ってしまうため期待通りの速度が得られないわけです。

FIND

デバッグ **文字列検索**

■書　式
FIND ["string"]
　string:検索する文字列

■解　説
プログラムのラベル重複、変数名などを検索します。引数無しでラベル重複、""で文字列を与えるとそれが含まれる行を表示します。変数名は大小文字を区別します。複数行発見した場合、16行で表示を停止、CRキーで継続になります。

```
FIND                              /* 引数無=ラベル多重定義の発見
FIND "ON"<ent>                    /* コマンドの検索
FIND "SW"<ent>                    /* 関数・配列の検索
FIND "aho"<ent>                   /* 変数の検索(文字列変数も含む)

list 0
10 *aho
20 TMOUT 10
30 IF a==1 THEN : TIME 10 : END_IF
40 GOTO *aho
50 *aho
60 WAIT SW(1)==1

#find
*aho[50][10]                      /* 二重定義されたラベルを表示
#find "SW"
60 WAIT SW(1)==1
#find "a"
30 IF a==1 THEN : TIME 10 : END_IF
#find "tmout"
20 TMOUT 10
```

FIX

デバッグ **フラッシュ ROMへ書き込み**

■書　式
FIX

■解　説
フラッシュ ROM（FROM）へプログラムを書込みます。RUNは書込み後プログラムを実行しますが、FIXは実行しません。ダイレクトコマンドのみの使用です。RAMモードでのFIXは出来ません。

```
#FIX
*-----+++++Ok
#
```

RUN 参照

FOR..NEXT

制御文 **繰り返し**

■書　式
FOR～TO～[STEP]
NEXT

■解　説
数指定繰り返し制御文。加算量はSTEPによって与えることができます。STEP以下が省略されれば、1ずつ増えることになります。また、FOR文のループからGOTO文で飛び出すことは許されています。

```
FOR j=0 TO 80 STEP 20
   FOR k=0 TO 100
      PRINT i j k
   NEXT k
NEXT j
```

FORK

タスク操作 **タスク実行**

■書　式
FORK n *ﾗﾍﾞﾙ
 n:タスク番号
  1≦n≦31

■解　説
他タスク実行コマンド。マルチタスクとはタイムシェアリングにより複数のプログラムを一定時間毎に順次実行し、同時に複数のプログラムが実行されているかのように見せかけるものです。ADVFSCにはこの機能が組み込まれており、同時実行できるプログラムの数は32です。機械制御ではI/Oのポーリング・タイマー待ちなど、それぞれのプログラムは比較的暇なことが多く、このマルチタスクが有効に動作します。ですから、演算のようにタイマーの要素が全くないプログラムをマルチタスクで実施しても何等メリットはありません。マルチタスクが効率よく動作し意味をなすのはそれぞれのタスクにTIME,WAIT,SW(n),IN(n)などのコマンドや関数が含まれている場合です。SW(n),IN(n)にはノイズ除去のために5msecのタイマーが組み込まれているためにTIMEがあるのと同じ意味を持ちます。

```
     FORK 1 *intval
     FORK 2 *p_port
*p_port
     DO
        FOR i01=0 TO 3
           P_ON i01
           TIME 100
           WAIT P_SW(i01)==1
        NEXT i01
     LOOP
*intval
     FORK 6 *b_out
     TIME 4000
     GOTO *intval
```

タスク32について
タスク32はリアルタイム(5～7m秒程度)で反応する特殊なタスクです。タスク32は常に優先的に実行されます。(task0->task32->task2->task32->task3--->)タスク32で実行できるのはSENSE_SW 文のみです。また、タスク32中のSENSE_SWの引数にSW(),IN()などタイマ要素を含む関数は使えません。HSW(),HIN() ?(),@()や変数のみ使用としてください。タスク32は必ず次のようにラベルとENDで挟んで途中はSENSE_SW文のみです。一行で250μ秒ですから記述は4行程度にしてください。急がない仕事にやたらに使うと、どんどん全体が遅くなります。用が済んだらquitしてください。実行中のタスクをFORKするとそのタスクは最初から実行されます。

```
FORK 32,*sense                  /* 起動

500 *sense                      /* 順番が回って来る度に*senseからENDまで実行
510 SENSE_SW ?(-1)&?(0) (OFF_P,0)
550 SENSE_SW ?(-1)&@(0) (ON_P,0)
555 SENSE_SW ?(-1)&?(-2) (X_A;STP_I,&H410)
556 SENSE_SW ?(-1)&?(-3) (Y_A;STP_I,&H410) (OFF_P,-3)  /*2回目は動作しないようにしている
560 END                         /* ENDまでSWAPしない
```

FORK 32は最初に実行してください。するとその後FORKしたタスクがすべて5m秒のライフタイムになります。これによりタスク32の反応速度を確保しています。
QUIT 32しても5m秒のライフタイム設定は解除されません。パワーオンリセットでデフォルトに戻ります。任意にライフタイムを戻すのにLIFE_TIMEコマンドは有効です。

QUIT,PAUSE,CONT 参照

FREE

編集 **残りメモリの表示**

■書　式
FREE

■解　説
メモリ残量表示。NEW後500kbyteです。ADVFSCでは1行の消費メモリが約20byteのため約25000ステップ記述可能です。

GET_VAL

文字列 **文字列からの数値自動取り出し**

■書　式
GET_VAL str arry(n)
 str:文字列
 arry(n):配列。nは取り込んだデータの複写先の先頭。

■解　説
文字列中にある数字を数値として自動的に配列に取り込みます。数字列中の小数点は無視して連続した数字列と扱います。
文字列からの取り出し

```
a$="b--99999.+321-456aho-874.1234e12b"
GET_VAL a$ aho(1)
FOR i=0 TO 8
 PRINT i : aho(i)
NEXT i
#run
0 0
1 -99999
2 321
3 -456
4 -8741234
5 12
6 0
7 0
8 0
```

バージョンデータの1行目から数字を取り込みます。

```
#VER
MPC-684 ADVFSC(r)eREV-3.81x
 BASIC like + multi tasking
 Created by ACCEL Co.'~2001

10        DIM a(2)
20        GET_VAL VER$ a(0)
30        FOR i=0 TO 1
40          PRINT i a(i)
50        NEXT i
#run
0 -684    /* MPC-684
1 -381    /* REV-3.81x
```

GETF

コプロ演算 **データ取り出し**

■書　式
GETF [45] fn1,fn2,fn3,..
 fn1,fn2,fn3,..:変数
[45]は、１０進数での四捨五入整数化の為のオプションです。

■解　説
コプロからデータを取り出します。与えられた変数に順番にコプロの不動小数点レジスタFP0,FP1..の値を取り出してセットします。

```
SETF 0 3 4
GETF A B C
PR A B C
  結果は0,3,4
```

SETF, CALF, PRF 参照

GO

パルス（MPG-68K互換） **4軸同時パルス発生**

■書　式
GO x y u z
GO P(n)
 xyuz:変数、定数
 n:点番号
このコマンドはMPG-405には使えません。

■解　説
GOコマンドは4軸同時のパルス発生です。これはXYロボットを構成した場合に斜め組み込みなどに有効です。GOコマンドは絶対位置移動です。これに対して相対4軸移動であるRMがあります。GOコマンドはMOVEに比べてスピードが遅くなるのは、ACCELコマンドが作成する加減速テーブルは3軸同時に見合ったものとなっている為です。

```
GO P(1)              点P(1)にダイレクトに移動します。
GO 50 100 200 400    絶対座標X=50,Y=100,U=200,Z=400にダイレクトに移動します。
```

RM,MOVE,RMOV 参照

GOSUB

制御文 **サブルーチンコール**

■書　式
GOSUB *ﾗﾍﾞﾙ

■解　説
サブルーチン実行（文番号は許されていません）

```
  GOSUB *p_port
  END
*p_port              /* ラベル
    DO
     FOR i01=0 TO3
        P_ON i01
        TIME 100
        WAIT P_SW(i01)<>1
        P_OFF i01
        TIME 100
        WAIT P_SW(i01)==1
     NEXT i01
    LOOP
    RETURN           /* RETURNでもどる
```

GOSUBのネストは各タスク約170です。

GOSUB

制御文 **サブルーチンコール(引数渡し)**

■書　式
GOSUB *ラベル [arg1 arg2 - -]
 arg:変数、定数

■解　説
argをサブルーチンの引数とします。ローカル変数と組合わせるとタスク間でのサブルーチンの共有が可能となります。受け側は _VAR var1,var2 - - です。

```
10  GOSUB *SUB 1 2 3  /*渡す
20  END
30  *SUB              /* ラベル!はローカル変数
40  _VAR A! B! C!     /*受け取る
50  PRINT A! B! C!
60  RETURN
#RUN
1 2 3
```

引数は10個までです。それ以上入力しても入りません。また、MPC-684に1行の文字数の制約等があり、長名の変数などは入らない場合がありますで注意してください。ローカル変数は26個までです。

```
10    DIM V(20)
20    FOR I=0 TO 20
30        V(I)=I+1
40    NEXT I
50    GOSUB *SUM V(0) V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8) V(9)
55    _RET_VAL RES
57    PRINT RES
60    END
100 *SUM
107   _VAR V0! V1! V2! V3! V4! V5! V6! V7! V8! V9!
110     J!=V0!+V1!+V2!+V3!+V4!+V5!+V6!+V7!+V8!+V9!
```

```
140   RETURN J!
#RUN
55
#
```

GOSUBのネストは各タスク約170です。

RETURN 参照

GOTO

制御文 **無条件分岐**

■書　式
GOTO *ﾗﾍﾞﾙ

■解　説
無条件分岐（文番号は許されていません）

```
*intval
   FORK 6 *b_out
   TIME 4000
   GOTO *intval
```

HEX$

文字列 **数値からヘキサ表現文字列へ変換**

■書　式
HEX$(n)

■解　説
数値をヘキサ表現文字列へ変換します。

```
PRINT HEX$(100)        64となります。
a$="aho"
b$=HEX$(ABC(a$))
PRINT b$
61                     ”a”のアスキーコードが文字列としてb$に入りました。
```

STR$,VAL 参照

HIN

I/O **パラレル入力**

■書　式
HIN(n)
 n:バンクナンバ
  HIN(n~Lng) 4byte読み込み
  HIN(n~Wrd) 2byte読み込み
  HIN(n~Int) 2byte符号付読み込み

■解　説
I/Oのパラレル入力（フィルタなし）。IN(n)と同様ですが、IN(n)のようなノイズフィルタ機能はありません。nはポートの指定、8bit単位で区切られてます。例えばポートの0～7まではバンク0となります。

```
PRINT HIN(0)&&H0F      0～3ビットを検査します。
```

IN 参照

HISTORY

メンテナンス **改版履歴表示**

■書　式
HISTORY

VER 参照

HOME

パルス（MPG-314専用） **定則低速原点復帰**

■書　式
HOME n rate cond [n1 rate1 cond]
 n:軸指定予約定数 X_A～Z_A
 rate:原点復帰速度 pps
 cond:入力指定 INx_ON～INx_OFFの論理和

■解　説
最初の引数に軸指定定数を設定すると1軸ごとの原点復帰コマンドになります。引数にパルスレートとIN0～IN3のどの入力の信号を使用するかを決定する値を設定できます。条件はIN0_ON～IN3_OFFが用意されており、和をとることにより複数の信号により停止させることができます。HOMEはACCELで設定したパラメータを破壊するため、PRSET_ACCELで復旧して下さい。また、座標値のクリア、プリセットは行わないのでSTPSで別途実施します。

```
HOME X_A 10 IN0_ON Y_A -10 IN1_ON    /* X軸CW10ppsIN0がオンになるまで
                                     /* Y軸CCW10ppsIN1がオンになるまで
```

原点復帰時に指定した停止条件は原点復帰後も有効となるためSTOP n VOIDで条件をクリアする。
予約定数は必ず大文字。

ACCEL,PRSET_ACCEL,STPS 参照

HOME

パルス（MPG-314専用） **高速加減速原点復帰**

■書　式
HOME n cond ±F [n1 cond1 ±F1]
 n:軸指定予約定数 X_A～Z_A
 cond:入力指定 INx_ON～INx_OFFの論理和
 F:feed 1～255　符号は回転方向 +:CW,-:CCW

■解　説
第1引数に軸、第2引数に停止条件を指定すると1軸毎の高速原点復帰コマンドになります。引数にパルスレートとIN0-IN3のどの入力の信号を使用するかを決定する値を設定できます。条件はIN0_ON～N3_OFFが用意されており論理和をとることにより複数の信号により停止させることができます。

```
HOME X_A IN0_ON -125 Z_A IN0_ON -200   /* X軸 IN0使用 FEED125 CCW
                                       /* Z軸 IN0使用 FEED200 CCW
```

原点復帰時に指定した停止条件は原点復帰後も有効となるためSTOP n VOIDで条件をクリアして下さい。
予約定数は必ず大文字。

ACCEL,PRSET_ACCEL,STPS 参照

HOME

パルス（MPG-68K互換） **原点復帰**

■書　式

HOME patn [x y u]

patn:原点センサーパターン

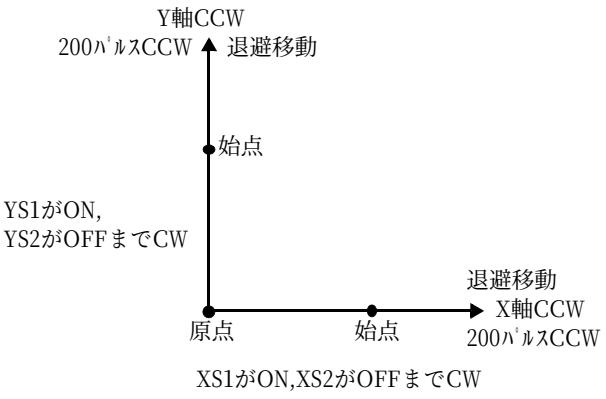
x y u:退避移動量（パルス）

■解　説

XYU同時3軸で原点復帰します。出力するパルスパタンとパルスレートはSHOMで定めた通りです。そしてpathで指定された入力条件となるまでパルス発生し続けます。

J4ｺﾈｸﾀｰﾋﾟﾝ番	8	7	6	5	4	3	2	1	
MSB	7	6	5	4	3	2	1	0	LSB
信号名	-	-	US2	US1	YS2	YS1	XS2	XS1	

例えば、HOME &H5とすればXS1が1、XS2が0、YS1が1、YS2が0となるまで原点復帰し続けます（1=ON、0=OFF）。もしここでUS1、US12のいずれかが1になっていると条件としてはUS1，US2がともに0ですからU軸の原点復帰も加わります。2番目以降の引き数は退避移動のパルス量です。初期状態では100パルスずつセットされているので必要に応じて設定します。1度設定されると2度目からは引き数がなくても設定され値だけ退避します。又、この退避移動のスピードはFEDHで指定します。HOMEコマンド終了時のポイントはX=0，Y=0，U=0になります。



例1)

```
SHOM &H5 5000          X軸CW、Y軸CW方向、5000pps
HOME &H5 200 200 0     退避量200パルス、XS1、YS1がONになるまで原点復帰動作
```

例2)

```
SHOM &B1010 500        <--Y軸:CCW,X軸CCW方向に原点復帰、ｽﾋﾟｰﾄﾞ:500pps
HOME &B0101            <--XS1とYS1がONになるまで原点復帰
```

【Q】原点復帰の前に原点センサーの入力を確認して、もし原点センサーがON状態ならOFFになるまで退避移動をしてから、再び原点復帰動作を行いたい。

【A】次のプログラムは原点復帰の前に HPT() 関数で原点センサーの状態を確認しています。もし原点センサーがすべてOFFでなければCCW方向に原点センサーがOFFするまで移動します。

```
10      IF HPT(0)<>0 THEN : GOSUB *TAIHI : END_IF
20      SHOM &B101 500          <--Y:CW,X:CW
30      HOME &B1010 100 100 0   <--YS2,XS2がONまでﾊﾟﾙｽ発生
40      END
50    *TAIHI
60      SHOM &B1010 500         <--Y:CCW,X:CCW
70      HOME 0 0 0 0            <--ｾﾝｻｰがALL OFFまでﾊﾟﾙｽ発生
80      TIME 500
90      RETURN
```

HOMEはSTOPで停止できます。

SHOM<ENTER>またはSHMZ<ENTER>で現在の設定状況が表示されます。

SHOM 参照

HOMZ

パルス（MPG-68K互換） **原点復帰**

■書　式
HOMZ patn [z]
　patn:原点センサパターン
　z:退避移動（パルス）

■解　説
Z軸の原点復帰をします。出力するパルスパタンとパルスレートはSHMZで定めた通りです。そしてpatnで指定された入力条件となるまでパルス発生し続けます。

J4ｺﾈｸﾀｰﾋﾟﾝ番							8	7	
MSB	7	6	5	4	3	2	1	0	LSB
信号名	-	-	-	-	-	-	ZS2	ZS1	

例えば、HOMZ &H1とすればZS1が1,2が0になるまで原点復帰し続けます。2番目以降の引き数は退避移動のパルス量です。初期状態では100パルスにセットされているので必要に応じて設定します。1度設定されると2度目からは引き数がなくても設定された値だけ退避します。又、この退避移動のスピードはFEDHで指定します。HOMZコマンド終了時のポイントはZ=0になります。

```
SHOM &H1 5000          Z軸CW方向、5000pps
HOME &H1 200           退避量200パルス、ZS1がONになるまで原点復帰動作
```

SHOM<ENTER>またはSHMZ<ENTER>で現在の設定状況が表示されます。

SHMZ 参照

HOUT

パルス（MPG-314専用） **ポート出力**

■書　式
HOUT p
　p:出力データ0～255

■解　説
HOUTを0～255の範囲の引数で使用するとOP1～4,X-SON～Z-SONのパラレル設定となります。下位がOP1～4の設定で従来のコマンドと互換性があります。

bit	7	6	5	4	3	2	1	0
port	Z-SON	U-SON	Y-SON	X-SON	OP4	OP3	OP2	OP1
J4pinNo	8	7	6	5	22	21	20	19

このポートをビット単位で操作するコマンドはありません。代用として
　　どこかのメモリIOバンクを専用のバッファにわりあてます。
　　そのバンクをON/OFFします。
　　そのバンクをHOUTでコピーします。

```
OUT 0 -1              /*メモリI/Oのバンク -1を初期化
*TASK1
 ON -1 : HOUT IN(-1)  /*メモリI/Oをビット操作してHOUT

*TASK2
 ON -2 : HOUT IN(-1)
```

という具合です。

HOUT

パルス（MPG-314専用） **レジスタ制御**

■書　式
HOUT n;cmnd
　n:軸選択予約定数 X_A～Z_A
　cmnd:コマンド予約定数

■解　説
MCX314のレジスタWR0を制御します。

```
HOUT X_A;STP_D     /* X軸に対して減速停止命令を実行します(STOP X_A STP_Dと同じ)
HOUT Y_A;CLR_ERR   /* Y軸のエラービットをクリアします
```

予約定数は必ず大文字。

HOUT

I/O **パルスボード汎用出力**

■書　式
HOUT patn

■解　説
MPG-68KのJ4コネクタには原点入力とあわせて4点の出力ポートがあります。HOUTはこの出力に対してパラレル出力します。尚このコマンドはMOVEやACCELと同じくPGコマンドひきあてられたタスク、もしくはPGSELによって選択されたMPGに対して有効です。MPG動作中には実行できません。
HOUTはMPC-684のJ4コネクタに対応していません。

```
PG &HE0 1              <--アドレス&HE0のMPGをタスク1に引き当て
FORK 1 *task1          <--タスク1の起動
(略)
*TASK1
     HOUT &H3          <--アドレス&HE0のMPGのJ4コネクタOP 1、OP 2をON
```

HPT

パルス（MPG-314専用） **入力ポート読み込み**

■書　式
HPT(0)

■解　説
MPG-314の入力を読み込みます。

```
#PRX HPT(0)
  00000000    /* 表示データ
   H5----0    /* 桁
```

返す値を8桁のヘキサで表現すると、次のように入力ポートの値がそれぞれの桁に反映されます。

H0: XS1,XS2(X-IN0,X-IN1),YS1,YS2(Y-IN0,Y-IN1)	(コネクタJ4)
H1: US1,US2(U-IN0,U-IN1),ZS1,ZS2(Z-IN0,Z-IN1)	(コネクタJ4)
H2: X-IN2～Z-IN2(差動入力)	(コネクタJ1)
H3: X-IN3～Z-IN3	(コネクタJ2)
H4: X-INPOS～Z-INPOS	(コネクタJ4)
H5: X-ALM～Z-ALM	(コネクタJ2)

例)X-INP(位置決め完了)待ち

```
WAIT HPT(0)&&H10000<>0
```

例)X-IN3オン待ち

```
WAIT HPT(0)&&H1000<>0
```

HPT

I/O **パルスボード原点ポート入力**

■書　式
HPT(n)
 n:入力ビット番号
  n=0:パラレル入力
  n=1～8:ビット入力

■解　説
HPT()はMPG-68Kの原点入力を読む関数で、原点センサーのI/Oチェックに利用できます。入力状態はTEACHモードのH操作でも表示されます。HOUT,HPT()はTEACHかPGSELで選択されたMPGに対して有効です。
PGSELでMPGを選んで

```
PRX HPT(0)
PRINT HPT(1)
HOUT &H3
```

MPC-684のJ4コネクター入出力はP_SW参照

J4ピン番号	コマンド・関数
1	HPT(1)
2	HPT(2)
3	HPT(3)
4	HPT(4)
5	HPT(5)
6	HPT(6)
7	HPT(7)
8	HPT(8)
	HPT(0)　入力
9	
10	HOUT patn　出力
11	
12	

HSW

I/O **ビットの読み込み**

■書　式
HSW(n)
 n:ポート番号

■解　説
I/Oのビット入力（フィルタなし）。nはポート番号の指定です。SW(n)と同様ですが、SW(n)のようなノイズフィルタ機能はありません。返す値は1がON状態、OFFで0となります。ポート番号に出力ポートを指定すると出力ポートの状態を得ることもできます。

```
ON 0                    0ﾎﾟｰﾄ ON
PRINT HSW(0)            0ﾎﾟｰﾄ の状態表示
 1
ON -1                   ﾒﾓﾘｰI/O ON
PRINT HSW(-1)           ﾒﾓﾘｰI/Oの状態表示
 1
```

SW 参照

IF

制御文 **条件分岐**

■書　式
IF　条件式　THEN
 (制御文)
 ELSE
 (制御文)
END_IF

■解　説
条件分岐文。条件式で、指定された式が成立すればIF～ELSE間が実行され非成立であれば、ELSE～END_IF間のみが実行されます。次の例ではiが10以下とそうでない場合に分類し10以下であればPRINT "i<=10" iを実行します。10を除く10以上であれば、ELSEより下を実行するわけですがこの中にもIF文がありiが13の時に限りPRINT "i=13" iを実行します。

```
FOR i=0 TO 20
  IF i<=10 THEN
    PRINT "i<=10" i
    ELSE
    IF i==13 THEN
      PRINT "i=13" i
    END_IF
    PRINT "i>10" i
  END_IF
NEXT i
```

注意

```
IF ---- THEN
ELSE IF ---- THEN     /* ESLEの後にIFは NG
     END_IF
END_IF
  ↓
IF ---- THEN
ELSE : IF ---- THEN   /* コロンを入れればok
      END_IF
END_IF
```

IN

I/O **パラレル入力**

■書　式
IN(n)
 n:バンクナンバー

I/Oサイズ指定予約定数を使って
IN(n~Lng)　4バイト入力
IN(n~Wrd)　2バイト入力
IN(n~Int)　2バイト符号付入力

■解　説
I/Oのパラレル入力。HIN(n)と同様ですが、2度読みのノイズフィルタ機能付きです。これは、チャタやノイズによる誤読みとりを防ぐためのものです。nはポートのバンク指定で、8bit単位で区切られています。例えばポートの0～7まではバンク0となります。I/Oサイズ指定予約定数を使って入力バイト数を指定できます。物理入力はSWAPを含む2度読みですが、メモリI/O,MBK入力は1度読みです。

```
#PR IN(3)             /* 通常I/Oのバンク3から1バイト入力
255                   /* 255=&HFF=全部オン状態
```

```
#OUT &H12 -1          /* メモリI/0のバンク1へ1バイト出力
#OUT &H34 -2          /*        〃       2へ1バイト出力
#OUT &H56 -3          /*        〃       3へ1バイト出力
#OUT &H78 -4          /*        〃       4へ1バイト出力
#PRX IN(-1)           /* メモリI/0のバンク1から1バイト入力
 0012
#PRX IN(-1~Lng)       /* メモリI/0のバンク1～4入力
 78563412
#PRX IN(-1~Wrd)       /* メモリI/0のバンク1～2入力
 3412

#OUT -1 -1~Int        /* メモリI/0のバンク1～2へ出力
#PR IN(-1~Wrd)        /* メモリI/0バンク1～2入力
65535
#PR IN(-1~Int)        /* メモリI/0バンク1～2符号付入力
-1
```

SW,HSW,OUT 参照

INCHK_314

パルス（MPG-314専用） **入力モニタ**

■書　式
INCHK_314

■解　説
INCHK_314はMPG-314の入力ポートをモニタするコマンドです。標準I/Oに対するIOコマンドと同等の機能を持ちます。

INP$#0

RS-232 **n文字読み込み**

■書　式
INP$#0(n)
　n:文字数

■解　説
CH0よりn文字取り出す。

```
a$=INP$#0(1)
a=ASC(INP$#0(1))
```

INPUT$ 参照

INP$#2

RS-232 **n文字読み込み**

■書　式
INP$#2(n)
 n:文字数

■解　説
CH2よりn文字取り出す。

INPUT$,INP$#0 参照

INPBLK#

RS-232 **バイナリ固定フォーマット入力**

■書　式
INPBLK#n v1 v2 v3
 n:ポート番号0,2
 v1,v2,v3:変数

■解　説
データのバイナリレシーブです。input#は文字列解析・計算が入りデータの取得に時間がかかりますがinpblk #はバイナリ固定フォーマットでデータ受け取ることにより高率アップします。
送信フォーマット
　　[STX B0 B1 B2 B3 B4 B5 B6 SUM]
　　STX = &H2
　　B0～B6 = ﾊﾞｲﾅﾘﾃﾞｰﾀ
　　SUM = ﾁｪｯｸｻﾑ (0-((B0+B1+B2+B3+B4+B5+B6) and &HFF))
受信したデータは変数v1～v3に数値として入力されます。チェックサムでエラーになるとv3の値が255になります。

　　(上位) (下位)
　v1 ← B0,B1,B2
　v2 ← B3,B4,B5
　v3 ← B6

パソコンから数値12345を送信する。10進12345は&H3039です。送り側データは次の通りです。

```
&H02 &H00 &H30 &H39 &H00 &H00 &H00 &H00 &H97
```

MPCの変数には次の様に入力されます。

```
prx v1
 3039                    ←10進12345
prx v2
 0000
prx v3
 0000                    ←通常は送信ﾃﾞｰﾀと一致。ﾁｪｯｸｻﾑｴﾗｰで&HFF(255)
```

つまり、3byte（±8388607）データを同時に2つ、チェックサム付きで受信可能です。

■書　式
INPUT [s$,v]　　CH1数値/文字列の入力
INPUT#0 [s$,v]　CH0数値/文字列の入力
INPUT#2 [s$,v]　CH2数値/文字列の入力

■解　説
INPUT 文は無手順系の通信コマンドでターミネータとしてスペース、タブ、CR を使用するコマンドです。CH1のINPUTコマンドはエコーバックがありますがCH0,CH2についてはエコーがありません。また、INPUT文も文字列を表示することができます。文字列定数・定数に対してはPRINTコマンドと同様出力として動作します。

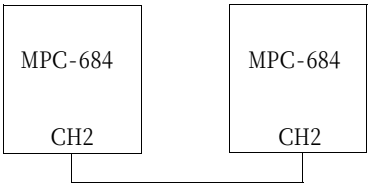
```
INPUT#2 "Please Set Any string ? " a$
PRINT#2 "echo--" a$+CHR$(13)+CHR$(10)

INPUT "Please Set Any string ? " a$
PRINT "echo--" a$
```

タスク0をEND終了させた場合INPUTは使えません。

INPUTのターミネーター

MPC-684 の RS-232 の INPUT コマンドはスペース、タブ、キャリッジリターンをターミネーターとして扱います。次のサンプルプログラムはMPC-684を2台を接続して作りました。



(1)　スペース/タブコードの入ったデータ

(1-1) 文字列の場合その1　スペースの入っている文字列
送信側

```
 LIST 0
5        CNFG#2 "9600b8pns1NONE"        CNFGコマンドでRS-232初期化
7        TIME 100
10       PRINT#2 "AB CDE\r"
```

受信側

```
 LIST 0
10       CNFG#2 "9600b8pns1NONE"
20         A$="" :   B$=""
25       TIME 50
30       INPUT#2 A$ B$
```

結果 スペースで区切られ2つの文字列になる

```
PRINT A$
AB
#PRINT B$
CDE
```

(1-2) 文字列の場合その2　スペースが入らない文字列
送信側

```
 LIST 0
5        CNFG#2 "9600b8pns1NONE"
7        TIME 100
10       PRINT#2 "ABCDE\r"
```

受信側

```
 LIST 0
10       CNFG#2 "9600b8pns1NONE"
20         A$="" :   B$=""
25       TIME 50
```

```
30        INPUT#2 A$
結果 とーぜん1つの文字列として扱われる
PRINT A$
ABCDE
#
```

(1-3) 定数・変数の場合その1　1つの定数
送信側

```
 LIST 0
5         CNFG#2 "9600b8pns1NONE"
7         TIME 100
10        PRINT#2 12345 "\r"
```

受信側

```
 LIST 0
10        CNFG#2 "9600b8pns1NONE"
20          A=0
25        TIME 50
30        INPUT#2 A
```

結果

```
PRINT A
12345
#
```

(1-4) 定数・変数の場合その2　データ間に半角スペースがある
送信側

```
 LIST 0
5         CNFG#2 "9600b8pns1NONE"
7         TIME 100
10        PRINT#2 123 45 "\r"
```

受信側

```
 LIST 0
10        CNFG#2 "9600b8pns1NONE"
20          A=0
25        TIME 50
30        INPUT#2 A
```

結果 1つの数値として入力（これは出力側の都合）

```
PRINT A
12345
#
```

(1-5) 定数・変数の場合その3　スペースコードが入っている
送信側

```
 LIST 0
5         CNFG#2 "9600b8pns1NONE"
7         TIME 100
10        PRINT#2 123 " " 45 "\r"
```

受信側

```
 LIST 0
10        CNFG#2 "9600b8pns1NONE"
20          A=0:B=0
25        TIME 50
30        INPUT#2 A B
```

結果 スペースで区切り2つの数値を入力

```
PRINT A
123
#PRINT B
45
#
```

```
RS 2                    RS-232バッファーを表示
```

RS受信

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 31 32 33 20 34 35 0D
```

RS送信

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
#
```

INPUTは文字列でも数値でもスペース(&H20)、TAB(&H09)、CR(&H0D)をターミネーターとして同じに扱います。スペースの入った文字列を1つの文字列変数に格納したい場合は次の様にINP$で受信するのが能率的です。

送信側

```
 LIST 0
5        CNFG#2 "9600b8pns1NONE"
7        TIME 100
10       PRINT#2 "ab cde\r"
```

受信側

```
 LIST 0
10       CNFG#2 "9600b8pns1NONE"
20         A$="" :   B$=""
25       TIME 50
30       DO UNTIL A$==CHR$(&HD) CRまで1文字づつ入力します。
35           A$=INP$#2(1)
40           B$=B$+A$            文字列の結合
50       LOOP
```

結果

```
PRINT B$              1つの文字列変数に格納されています
ab cde
```

(2) コントロールコードの場合は

例えばバックスペースコードが含まれているデータをINPUTで入力するとき数値として入力するか、文字列として入力するかで扱いが違います。

(2-1) バックスペースの入ったデータを数値として入力すると...

送信側

```
[1][ﾊﾞｯｸｽﾍﾟｰｽ][2][CR]  と送信
#PRINT#2 1 CHR$(&H8) 2 "\r"
```

受信側バッファの内容 (ﾊﾞｯﾌｧ先頭は"1"(&H31)になっている)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 31<08 32 0D
```

これを数値として読み込むと

```
#input#2 a
#print a              ...ﾊﾞｯｸｽﾍﾟｰｽをﾀｰﾐﾈｰﾀｰとして1だけをaに入力
1
```

受信側バッファの内容 (ﾊﾞｯﾌｧ先頭は"2"(&H32)になった)

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 31 08 32<0D
#input#2 b
#print b              ...CRをﾀｰﾐﾈｰﾀｰとして2をbに入力
2
```

つまり、数値として入力すると数字以外のコードはターミネーターとして扱われます。

(2-2) 文字列として入力すると・・・・
送信側

```
#print#2 1 chr$(&h8) 2 "\r"
```

受信側バッファ内容

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 31<08 32 0D
#input#2 a$           ...文字列変数a$に入力
#print a$
2                     ...2しか入らない
```

受信側バッファ内容

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 31 08 32 0D
```

つまり、文字列として入力するとﾊﾞｯｸｽﾍﾟｰｽとして解釈される。
送信側

```
#PRINT#2 "ABC" CHR$(&H8) "DEFG" "\r"
受信側
#INPUT#2 A$
#PRINT A$
ABDEFG                <--画面上"C"の上に"D"が重ね書きされた。
```

INPUTのLFコード（&HA）の扱いについて

INPUTはLFを普通の文字列と同様に扱い、RS-232のバッファから削除したりしません。
CR・LF(\n)でターミネートされてる文字列を入力するとCRまで読み込みLFはバッファに残されます。そして次に読み込む文字列の先頭はLFになります。先頭にLFがある文字列を文字列変数にINPUTで読み込むとその文字列変数の先頭にもLFが入りいますが、PRINTコマンドでその文字列を画面表示をするとLFは無視されるのでLFが無いように見えます（例1）。LFが先頭にある文字列を数値変数として読み込むとLFでキャンセルされて変数は常に0になってしまいます（例1）。（例3）では文字列変数に読み込みLFを削除して数値に変換しています。接続されている機器と送受信タイミングのインターロックされている場合はINPUTで入力した後、CNFGコマンドでバッファをクリアーしてしまうのが一番簡単です。
（例1）文字列変数に読み込む場合

```
CNFG#2 "9600b8pns1NONE"   <--CNFGで通信条件設定とﾊﾞｯﾌｧｸﾘｱｰ
#print#2 "123\n"          <--1回目送信
#rs 2
```

RS受信

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 31<32 33 0D 0A
```

RS送信

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 31 32 33 0D 0A
```

```
#input#2 str$         <--CRまで読み込む、LFはバッファに残す
#pr str$
123                   <--1回目は変数の内容、文字数OK、
#pr len(str$)
3
#print#2 "456\n"      <--2回目送信
#rs 2
```

RS受信

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 31 32 33 0D 0A<34 35 36 0D 0A
```
RS送信
```
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 31 32 33 0D 0A 34 35 36 0D 0A
#input#2 str$           <--前回のLFから読み込む
#pr str$
456                     <--文字列として読み込み表示するとまとも？
#pr len(str$)
4                       <--文字数はLFを含む数
```

(例2) 数値変数に読み込む場合

```
CNFG#2 "9600b8pns1NONE"
#print#2 "123\n"        <--1回目送信
#rs 2
```
RS受信
```
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 31<32 33 0D 0A
```
RS送信
```
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 31 32 33 0D 0A
#input#2 n              <--CRまで読み込む、LFはバッファに残す
#pr n
123                     <--1回目はOK
#print#2 "456\n"        <--2回目送信
#rs 2
```
RS受信
```
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 31 32 33 0D 0A<34 35 36 0D 0A
```
RS送信
```
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 31 32 33 0D 0A 34 35 36 0D 0A
#input#2 n              <--LFから読み込む
#pr n
0                       <--0???
```

(例3) 文字列変数に読み込みLFを削除して、数値に変換

```
10        CNFG#2 "9600b8pns1NONE"
20        FOR i=1 TO 3
30          PUT#2 &HA &HA
40          PRINT#2 "123\n"
50          TIME 20
60          INPUT#2 st$
70          GOSUB *lf_remove
80          PRINT st$ " " LEN(st$) VAL(s$)
90        NEXT i
100       END
110     *lf_remove
120         s$=""
130       FOR j=0 TO LEN(st$)-1
140         STRCPY st$ tmp$ j 1
150         IF tmp$<>CHR$(&HA) THEN :   s$=s$+tmp$ : END_IF
160       NEXT j
170       RETURN
```

```
#
RUN
123 5 123
123 6 123
123 6 123
```

前記実験はMPC-684のコネクター J1の8,9番ピンをショートして行いました。

INPUT$

RS-232 **n文字読み取り**

■書　式
INPUT$(n)　CH1よりn文字取り出す
INP$#0(n)　CH0よりn文字取り出す
INP$#2(n)　CH2よりn文字取り出す
　n:文字数

■解　説
INP$#0(), INP$#(), INPUT$()は各ポートより指定された文字だけ取り出す関数です。通常は次のように文字列変数へ複写して使用します。もし、バッファに指定された文字列が受け取られていなければ数が満たされるまでポーリングします。

```
A$=INP$#0(10)        CH0より10文字取り出してA$へコピー
A$=INP$#2(10)        CH2より10文字取り出してA$へコピー
A$=INPUT$(10)        ターミナルソフトからの入力です。
```

INPUT#0

RS-232 **データ入力**

■書　式
INPUT#0 [s$,v]

■解　説
CH0(RS-232Cユーザーポート)の 数値、文字列の入力。　ターミネータ：スペース、タブ、CR
数値変数に入力すれば数値に、文字列変数に入力すれば文字列になります。

```
INPUT#0 A            /*数値変数に入力
PRINT A              /*64(<x36><x34>)<cr>ト受信すると
64                   /*10進表現

INPUT#0 A$           /*文字列変数に入力
PRINT A$             /*64(<x36><x34>)<cr>ト受信すると
64
A=VAL("&H"+A$)       /*10進数値変換
PRINT A
100                  /*10進表現
PRX A
 0064                /*16進表現
```

INPUT 参照

INPUT#2

RS-232 **データ入力**

■書　式
INPUT#2 [s$,v]

■解　説
CH2(RS-232Cユーザーポート)の 数値、文字列の入力

INPUT 参照

INSET_314

パルス（MPG-314専用） **入力ポート機能設定**

■書　式
INSET_314 n cond
n:軸指定予約定数
cond:入力指定

■解　説
入力ポートの機能を設定します。最後に実行したINSET_314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。

INSET_314 X_A ALM_ON|INP_ON	**X軸に対して入力設定します。アラームを有効にしてON状態をアラームとします。また、インポジションを有効にしてONでインポズ成立とします。**
INSET_314 ALL_A ALM_ON|INP_OFF	**すべての軸に対して入力設定します。アラームはONで有効、INPOSはOFFで有効とします。**
INSET_314 ALL_A VOID	**全設定クリア。RANGE設定もクリア**

IO

I/O **I/Oモニタ**

■書　式
IO [n]
n:ポート番号

■解　説
I/O状態ビットマップ表現。I/Oの状態を画面に表示します。引き数で指定された値のポート番号のバンクから64点表示し、0.5秒毎リフレッシュされます。任意のキィーを押すと停止します。実際のI/OチェックはFTMWのIOC機能の方が便利です。

```
#IO 8<ent>          /* MIPの入力
#IO -1<ent>         /* メモリI/Oの入力
#IO 70000<ent>      /* GP(MBK)の接点エリア
#IO 2000<ent>       /* SLINKの入力
```

IOR

I/O **バス読み込み**

■書　式
IOR(adrs)
　adrsより読みだし（バイト読みだし）
　　0≦adrs≦&HFFFF
IOW n adrs
　adrsにnを書き込み（バイト書き込み）
　　0≦adrs≦&HFFFF

■解　説
MPC-684はPC-98のI/Oバスに準拠したバス構造となっています。この為、PC-98シリーズに用意されているI/Oボードをこのコマンドで直接読み書きする事が出来ます。

```
IOW tsk03 &H1D0
IOW NOT(tsk03) &H1D1
TIME 1
IF IOR(&H1D1)<>tsk03 OR IOR(&H1D1)<>NOT(tsk03)&&HFF THEN
GOTO *rr
END_IF
```

IOR(adrs)
I/Oバスアドレスadrsよりバイトアクセスします。但し、MPC-684はハード的な制約から奇数アドレスに対するバイト読みだしがワード呼び出しになってしまいます。（A0が常に0の為）

IOW n adrs
I/Oバスアドレスadrsに対してバイト書き込みします。但しMPC-684はハード的な制約から奇数アドレスに対するバイト書き込みがワード書き込みになってしまいます。（A0が常に0の為）

IOW

I/O **バス書き込み**

■書　式
IOW n adrs
　adrsにnをバイト書き込み
　　0≦adrs≦&HFFFF

IOR 参照

JMPZ

パルス（MPG-68K互換） **ゲートモーション移動**

■書　式
JMPZ P(n)
　n:点番号

■解　説
JMPZは調整用コマンドです。ゲートモーションを途中で終了してZ軸の下降移動を実施しません。

REV-3.85c JMPZ修正（031107）
MPG-314のJMPZコマンドがマニュアルの記述と異なっておりました。3.85cでバグフィクスとしてマニュアルに整合させました。しかし長い間放置されたバグであったため、既成事実として使用されている恐れもありましので、mpcinit後、確認コマンドを実行するまで、JMPZを動作しないようにしてあります。
"JMPZ -1"を一度実行すれば、警告メッセージが出なくなり、正常なJMPZを実行します。MPG-68K,MPG-405についてはこの手続きは発生しません。

```
#JMPZ P(1)              /*MPCINIT 直後はメッセージ表示
```

REV-385以前MPG314のJMPZはマニュアルの記載と異なっていました。修正しましたしので、了承・確認のため一度jmpz -1を実行してください。以後JMPZが使用可能となります。

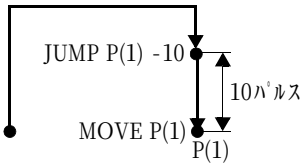
```
#JMPZ -1                /*解除。以後表示無し
```

JUMP

パルス（MPG-68K互換） **ゲートモーション移動**

■書　式
JUMP P(n) [a]
 n:点番号
 a:Z軸オフセット

■解　説
JUMPはゲートモーションです。第2引き数を指定すると値分Z軸のパルス発生に加算されます。JUMP P(1) -10は目的となるポイントP(1)より-10パルス上空で停止となります。



LD_M

MBK-SH **メモリー括コピー (MPC→MBK)**

■書　式
LD_M array mbktop count [opt]
 array:MPC点ﾃﾞｰﾀの先頭(XYUZ(1～13000))、配列変数の先頭
 mbktop:MBKﾒﾓﾘ先頭ｱﾄﾞﾚｽ。 20≦mbktop
 count:ｺﾋﾟｰ数
 opt:4=ﾛﾝｸﾞﾜｰﾄﾞで1ｱﾄﾞﾚｽおき、 Lng=ﾛﾝｸﾞﾜｰﾄﾞで連続

■解　説
配列からMBKのデータエリアにコピーします。パラメータはアドレスの使用範囲を超えねように設定して下さい。

```
LD_M X(9020) 20 683       /* データX(9020)からDT20へ683個ワードコピー
LD_M test(10) 500 10      /* 配列変数test(10)からDT500へ10個ワードコピー
LD_M X(9020) 20 342 4     /* 点データX(9020)からDT20へ342個ロングコピー
LD_M X(9000) 500 5 Lng    /* 点データX(9000)からDT500へ5個ロングコピー (連続)
LD_M test(0) 500 5 Lng    /* 配列変数test(0)からDT500へ5個ロングコピー (連続)
```

SV_M 参照

LEN

文字列 **文字数取得**

■書　式
LEN(s$)
 s$:文字列

■解　説
文字列s$の長さを与える。

```
PRINT LEN("ABC")
3
S$="ABC"
PRINT LEN(S$)
3
```

LET

演算 **式実行**

■書　式
[LET] 代入式

■解　説
LETは式を実行するコマンドです。LETは、入力してもLIST時には表示されません。又、式のみを記述しても内部で自動的に式として判断します。

数値演算
数値演算は、=()の記号、及び+,-,*,/,%,&,¦,\の演算子で記述する事が出来ます。()の中式は優先的に演算され*,/,%,&,¦,\は+,-より優先される演算となっています。

+	加算
-	減算
*	乗算
/	除算
%	余算
&	論理積（and）
¦	論理和（or）
\	排他的論理和（xor）

```
#LIST
10     dsw=(&HF0&IN(24))/16*10+&HF&IN(24)
20   PRINT "dsw=" dsw
110    a=3000 :   b=4000 :   c=30 :   d=40
120    res=SQR(SQ(a)+SQ(b))-SQR(c*a+d*b)
130  PRINT "res=" res
#run
dsw=55
res=4500
```

この例は、入力ポートに接続されたデジスイッチの内容を読み取る演算と一般の計算の例です。120の行に記述されたSQR及びSQはそれぞれ平方根と自乗（2乗）を意味しています。又、定数としての&HF0,&HFはヘキサ表現を意味しています。ビット表現の場合は&B□□とします。

文字列演算
ADVFSCでは文字列の演算もできます。許される演算子は、加算のみです。

```
#LIST
40     aho$="HEX->"+"&H"+HEX$(100*100+1)
50   PRINT aho$
80   INPUT ”何か入力して！” a$
90     b$=”あなたが入力したのは－＞”
100    c$=b$+a$
110 PRINT c$
#run
HEX->&H2711
何か入力して！ 123
あなたが入力したのは－＞123
```

文字列演算中でも40の行にある通り文字列を得る関数の中身である限り数値演算も可能です。HEX$()は与えられた数値のヘキサ表現の文字列を与えます。

LIFE_TIME

タスク操作 **タスク寿命を設定**

■書　式
LIFE_TIME taskn m
 taskn:ﾀｽｸ番号
 m:寿命値 1～20。2m秒単位

■解　説
MPC-684のタスクのライフタイムは約22m秒となっています。この時間を変更するコマンドとして、LIFE_TIMEがあります。FORK文のあとで実行します。次の例ではタスク1が2m秒でタイムアウトし、タスク0に実行権を引き渡します。

```
5 FORK 1 *aho
6 LIFE_TIME 1 1
10 DO
20 LOOP
100 *aho
110 DO
120 LOOP
```

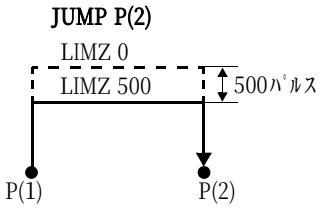
参考　ラウンドロビンのベースタイム 22msec　ディスパッチオーバーヘッド　0.2msec

LIMZ

パルス（MPG-68K互換） **ゲートモーション規制**

■書　式
LIMZ a
パラメーターを省略すると現在の設定値が表示されます。

■解　説
LIMZはJUMP,JMPZのZ軸の上限を指定します。例えば、初期状態で点P1=(1000,1000,100,2000)、点P2=(2000,3000,1000,3000)とし、点P1の位置から点P2の位置にJUMPもしくは、JMPZするとこのコマンドの最初の移動はMOVZ 0です。ここで、LIMZ 500と指定しておくと最初の移動がMOVZ 500となります。LIMZの値を適切に設定する事により、Z軸移動にかかる無駄な時間を減らす事が出来ます。



LIST

編集 **プログラム表示**

■書　式
LIST [n m]
 n:文番号またはラベル(省略すると前回の続き)
 m:表示行数（初期値）

■解　説
プログラムの表示

LIST	続きを表示
LIST 文番号	文番号より表示
LIST *ラベル	ラベルより表示
LIST *ラベル 4	ラベルより4行表示
LIST 0 0	全部表示

表示行数mは記憶されます。(Rev3.81a(2001/03)以降)

LNK

MPC-LNK **LNKアドレス取得**

■書　式
LNK(n [+ad])
 n:0
 +ad:リンクボードアドレス

■解　説
MPC-LNKのアドレスを得る関数です。

```
PR LNK(0)
```

LOF

RS-232 **バッファの文字数**

■書　式
LOF(n)
 n:チャンネル番号

■解　説
CHnのバッファ中のキャラクタの数の取得

```
(このプログラムはCH1をループバックさせていないと動作しません)
     CNFG#0 "9600b8pns1NONE"
     FORK 1 *aho
     DO
        PRINT LOF(0)   'バッファは255byte
        TIME 500       '255byteを越えると読み捨て
     LOOP
*aho
     PUT#0 &H41        '&H41=A
     DO
        TIME 100
        PUT#0 &H30     '&H30=0
     LOOP
```

LOG

デバッグ **プログラムポート出力記録**

■書　式
LOG [0]　(サポート EV-3.82以降)

■解　説
LOG<ent>とするとRS-232 CH1(プログラムポート)の出力記録を表示します。これにより自動実行時のランタイムエラーの事後確認ができます。記録メモリは1kバイトのリングバッファです。LOG 0<ent>とするとバッファをクリアします。
タスク0でのランタイムエラーのメッセージは記録できません。また、タスク0がEND終了していても記録できません。

LOOP

制御文 **繰り返し**

■書　式
DO [UNTIL/WHILE]
LOOP [UNTIL/WHILE]

DO.LOOP 参照

M_INP

MPC-LNK **アドレス&H1000のLNKからデータ取出**

■書　式
M_INP arg

■解　説
受信バッファからの読み込みです。arg の仕様はINPUTコマンドと同仕様です。バッファには250バイトのデータを格納されています。読みだす前にM_R n を実行します。

```
M_INP a$ data b$
```

M_INP1

MPC-LNK **アドレス&H2000のLNKからデータ取出**

■書　式
M_INP1 arg

■解　説
受信バッファからの読み込みです。arg の仕様はINPUTコマンドと同仕様です。バッファには250バイトのデータを格納されています。読みだす前にM_R n を実行します。

```
M_INP1 a$ data b$
```

M_PR

MPC-LNK **アドレス&H1000のLNKへデータ送出**

■書　式
M_PR arg

■解　説
送信バッファへの書き込みです。arg の仕様はPRINTコマンドと同仕様です。バッファには250バイトのデータを格納し一度に送出することができます。送出はM_X n で実行します。

```
M_PR "abc" 10000 "def"
```

M_PR1

MPC-LNK **アドレス&H2000のLNKへデータ送出**

■書　式
M_PR1 arg

■解　説
送信バッファへの書き込みです。arg の仕様はPRINTコマンドと同仕様です。バッファには250バイトのデータを格納し一度に送出することができます。送出はM_X n で実行します。

```
M_PR1 "abc" 10000 "def"
```

M_R

MPC-LNK **メール受信**

■書　式
M_R n [+ad]
 n:読み取るLNKアドレス 1~16
 +ad:リンクボードアドレス

■解　説
メールの受信です。読み取りするアドレスを設定します。

```
M_R 12
M_INP a$
```

M_RMVS

パルス（MPG-314専用） **非対称加減速移動(相対移動)**

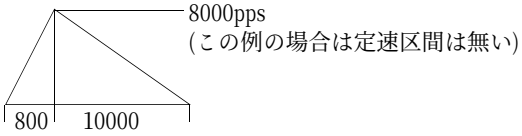
■書　式
M_RMVS n pls slow_length [adj]
 n: X_A～Z_Aのいずれか一つ
 pls: 移動距離
 slow_length: 減速距離
 adj: 減速開始位置微調整

■解　説
減速度ファクタを与え減速度を決定して非対称加減速度とします。内部演算を整合させるため直前に実行されたACCELのパラメータを使用します。加減速の比は加速距離と反比例関係にあることにより算出します。

減速度=L*加速度/減速距離。

adjは引きずりあるいは尻切れが発生した場合の微調整用です。引きずりは加速距離と減速距離の和が移動距離より大きいとかならず発生します。また、一度M_RMVSを実行するとパルス発生モードが変わってしまいその後のパルス発生に影響します。
このため用済みになったらM_RMVS X_A (パルス引数無し)を実行するかACCELを実行して通常モードに戻します。
例)800パルスで加速して8kppsになり残り10000パルスを減速する。

```
ACCEL 8000 800 10
M_RMVS X_A 10800 10000
WAIT RR(X_A)==0        /*停止待ち
M_RMVS X_A             /*ﾓｰﾄﾞｸﾘｱ
```



例)10000パルスで加速して8kppsになり残り800パルスを減速する。

```
ACCEL 8000 10000 10
M_RMVS X_A 10800 800
WAIT RR(X_A)==0        /*停止待ち
ACCEL 8000             /*ﾓｰﾄﾞｸﾘｱ
```

予約定数は必ず大文字。

M_X

MPC-LNK **メール送信**

■書　式
M_X n [+ad]
 n:送信先LNKアドレス 1～16
 +ad:リンクボードアドレス

M_X(n [+ad])
 n:送信先LNKアドレス 1～16
 +ad:リンクボードアドレス

■解　説

M_X n [+ad]
メールの送信です。送出したいアドレスを設定します。存在しないアドレスを設定するとこのコマンドでハングアップします。ハングアップしたら他のタスクからM_X 0 を実行すると解消されます。

```
M_PR "123":M_X 12
```

M_X(n [+ad])
メールの送信です。送出したいアドレスを設定します。存在しないアドレスを設定するとこの関数でポーリングしタイムアウトを検出します。一秒以内に転送できない場合は1を返します。0が返ってくれば送信が完了し相手は受信していることになります。

```
WAIT M_X(12)=0
```

MBK

MBK-SH/RS **データエリア読み出し**

■書　式
ワード(2バイト)読み出し
MBK(dtadr)
 dtadr:MBKレジスタアドレス

符号付ワード(2バイト)読み出し
MBK(dtadr~Int)
 dtadr: データアドレス
 Int: 予約定数

符号付ロング(4バイト)読み出し
MBK(dtadr~Lng)
 dtadr: データアドレス
 Lng: 予約定数

■解　説
MBK-SHのデータエリアからデータを読み出します

```
p=MBK(0)            /* DT0000(表示画面番号)読み出し
r=MBK(21~Int)       /* DT0021を符号付で読み出し
r=MBK(699~Lng)      /* DT0699(下位)とDT700(上位)をﾛﾝｸﾞ読み出し
```

S_MBK,SW,ON,IN,OUT 参照

MBK

MBK-SH/RS **バックアップ状態・機種確認**

■書　式
MBK(n)
 n=-1: バックアップ状態の確認
 n=-3: MBK機種確認
 n=-4: MBKバックアップデータの整合化

■解　説

バックアップ状態の確認
MBKのデータはMBK関数、S_MBKコマンドなどでのアクセス時にMPC-684のS-RAMにバックアップされ、電源投入時に復元されます。もしも、バックアップデータが破損した場は MBK にデータは復元されません。MBKデータが正常だったかどうかは、MBK(-1)関数で知ることができます。MBK(-1)が0であれば、データは正常でバックアップデータが復帰されています。もしデータが破損していると破損している最初のMBKデータの番号を返します。また、MBKのデータはクリア状態になっています。

MBK機種確認
MBKの種類を確認できます。返される値が799であればMBK-68、7899であればMBK-SH、-7899であればMBK-RSです。

MBKバックアップデータの整合化
引数に-4を指定すると全データエリアをダミーリードします。つまり、MBKの全データエリアをMPC-684のSRAMに再バックアップします。実行時間80mSec。頻繁に使うと全体のパフォーマンス低下の原因になりますから要注意。

```
#PR MBK(-1)
0                   /* バックアップ正常
#PR MBK(-3)
7899                /* MBK-SH
a=MBK(-4)           /* 全データエリアバックアップ。aはダミー変数
```

MEM

メンテナンス **メモリーテスト**

■書　式
MEM

■解　説
メモリーテストです。RAMの動作が疑わしい時に使用します。プログラムエリアのRAMをプログラムを破壊することなくテストします。このコマンドでプログラムが壊れたり、エラーが出力されればメモリ不良です。購入後1年以内であれば返品して下さい。MEM<ENTER>するとメモリテストスタートと表示されます。メモリテスト終了と表示されるまで待ちます。

MON

デバッグ **停止状態確認**

■書　式
MON [1]

■解　説
各タスクの状態モニタ。通常は<CTRL>＋<A>によって停止した状態をモニターするものですがサブタスクによって常時表示する事も可能です。引き数を与えると0.5秒毎に実行文番号を表示します。

```
#LIST
10   FORK 1 *task1
20   FORK 2 *task2
30   DO
40     FOR i=0 TO 7 : ON i TIME 100 : OFF i : NEXT i
50   LOOP
60 *task1
70   DO
80     FOR j=8 TO 15 : ON j : TIME 100 : OFF j : NEXT j
90   LOOP
100 *task2
110   DO
120     FOR k=16 TO 23 : ON k : TIME 100 : OFF k : NEXT k
130   LOOP
#run
      *0      [40]    <-- <CTRL>+<A>で停止
      *1      [80]       *2  [120]
#mon
      *0      [40]
      *1      [80]       *2  [120]
#
```

プログラム停止時とMON実行時のタスク番号後ろの「!」の意味。
タイムスライス内(20mSec)にSWAPが無いタスクを表します。これが沢山あるとタスク効率の低下になります。メモリーアクセス(メモリI/O,MBK入力,演算等)の連続、WAITで条件成立している時(次記)などがあります。

```
a=0
DO
  WAIT a==0 /*SWAPしない
LOOP
```

MOVE

パルス（MPG-68K互換） **XYU絶対座標移動**

■書　式
MOVE x y u
 x,y,u:絶対座標(変数、定数)
MOVE P(n)
 n:点番号
  1≦n≦13000

■解　説
3軸同時移動のパルス発生コマンドです。MOVEは絶対座標移動です。スピード上限及び加減速度はACCELコマンド既定されその範囲内でのスピード設定は FEED コマンドで指定できます。引き数としては 3 軸分(X,Y,U)をスカラ値で与えることも、点データP(n)で指定することもできます。点データP(n)を使用した場合はP(n)のX,Y,U成分だけが使用されZ成分は無視されます。

RMOV 参照

MOVL

パルス（MPG-314専用） **直線補間（絶対座標移動）**

■書　式
MOVL x y u z
　x y u z:各軸座標時
　パルス発生無=VOID

MOVL P(n)
 n:点番号

■解　説
直線補間パルス発生コマンドです。MOVL x y u zとしますが、現在位置との相対値を算出してパルス発生しますので、MOVSと同様現在位置が確定していることと、実際の直線補間軸が何軸になるか注意が必要です。例えば、現在位置 100 200 300 400 で MOVL 200 300 400 500とするとそれぞれ100パルスずつの出力が必要で、結果4軸直線補間となりエラーとなります。MOVL 200 300 400 400ではZ軸の発生量は0となり3軸直線補間で移動可能です。相対値を計算してからのパルス発生はRMVLと同じ扱いとなります。
パルス発生させない軸にはVOIDを設定します。

```
MOVL 100 200 300 VOID /* Z軸を除いた直線補間パルス発生
```

現在位置と比較してのパルス発生なので、実行前に現在位置が確定していなければなりません。ただし、直線補間を連続して実行する場合は発生終了待ちを省略できます。

```
40 RMVL 1000 2000 0 0
50 MOVL 100 200 VOID VOID
```

XYUを3軸直線補間した場合ACCELのパラメータ優先順位は X>Y>Uです。
Uの出力数が最も多くなったとしてもUが(DDAでいうところの)主軸にはならずXの移動開始・停止にUが同期して高速化してしまいます。XYサーボ、Uステップのような組み合わせでは3軸直線補間は困難です。その場合、UはMOVSで制御しますが、同時スタートでも終了はU軸非同期となります。

```
ACCEL X_A XYspeed
ACCEL U_A Uspeed          /* U軸はこの値に従う
MOVL Xpos Ypos VOID VOID
MOVS U_A Upos
WAIT RR(X_A|Y_A|U_A)==0   /* 終了確認
```

注意:「補間ドライブにおいても、ドライブする各軸のハードリミット、ソフトリミットは作動します。補間ドライブ中、いずれの軸のリミットがアクティブになっても、補間ドライブは停止します。」
(MCX314 取扱説明書より)
予約定数は必ず大文字 。

RMVS,RMVL,MOVS,ACCEL,PRSET_ACCEL,FEED 参照

MOVS

パルス（MPG-314専用） **軸独立パルス発生（絶対座標移動･直線補間無し）**

■書　式
単軸指定
MOVS n c
 n:軸指定予約定数 X_A～Z_A
 c:座標値

複数軸指定
MOVS x y u x
 x y u x:各軸座標値。指定無=VOID予約定数。

■解　説
パルス発生コマンドです。速度、加速度等はACCELで設定します。入力方法には単軸指定、複数軸指定の二通りあります。MOVSは直線補間はしません。複数軸指定でパルス発生をしない場合はVOIDをセットします。

```
MOVS X_A 1000                /* X軸1000位置
MOVS Y_A -1000               /* Y軸-1000位置
MOVS 1000 -2000 VOID VOID    /* X軸1000 Y軸-2000
```

絶対座標移動は現在位置と比較するため、直前のパルス発生が停止していなければなりません。

```
WAIT RR(Y_A)==0              /*Y軸停止待ち
MOVS Y_A 10000
```

RMVS,RMVL,MOVL,ACCEL,PRSET ACCEL,FEED 参照

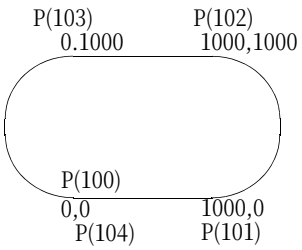
MOVT

パルス（MPG-314専用）　　**連続補間（絶対座標移動）**

■書　式
MOVT n point [<CW,CCW>]
n:軸指定予約定数。2軸の論理和
point:P(n)形式で点ﾃﾞｰﾀを指定
CWまたはCCW:円弧補間方向
注)指定する点列に重複する点、もしくは著しく近接する点を連続設定しないで下さい。ICの特性により異常動作する事があります。

■解　説
RMVTコマンドの座標位置指定連続補間です。RMVTでは全て相対位置指定となるため、実際のデータ入力が大変困難です。MOVTはあらかじめ座標値を点データに指定しておくことにより所定の軌跡制御を実行します。座標値は移動する順序に従って連続で登録します。これは、MOVTコマンドは指定された1つ前との点との差を計算し、その値をRMVTとして実行するためです。点データは通常X,Y,U,Zの値ですが、MOVTでは第一軸、第二軸、第一軸円弧中心、第二軸円弧中心の順に入力します。軸定数は必ず和で表記して下さい。

```
10 SETP 100 0 0 0 0
20 SETP 101 1000 0 0 0
30 SETP 102 1000 1000 1000 500
40 SETP 103 0 1000 0 0
50 SETP 104 0 0 0 500
60 PG &H410
70 ACCEL 2000
80 STPS 0 VOID 0 VOID
90 HOUT X_A;DS_DACL
100 MOVT X_A|U_A P(101)
110 MOVT X_A|U_A P(102) CCW
120 MOVT X_A|U_A P(103)
130 MOVT X_A|U_A P(104) CCW
140 MOVT X_A|U_A P(101)
150 HOUT X_A;EN_DACL
```



例えば、110の文では点102を指定しています。X_Aの移動量としてはX(102)-X(101),U_Aの移動量としてはY(102)-Y(101)が使用されます。CCWの円弧指定のためP(102)へ至る経路は円弧補間となり、その円弧中心はU(102)-X(101),Z(102)-Y(101)となります。ここではXU軸の円弧補間を含む軌跡制御ですが、座標のXYUZというラベルと適用される軸は無関係なことに留意して下さい。
次は同様の動作を実際のプログラムに応用したものです。点3999を図面上の基準点として4000～4005の各点を設定しています。このプログラムではD(0)～D(6)で動作モードを設定しておいてサブルーチン*UT1_dis1で実際の動作を行います。
最初の MOVS 1000 1000 0 0は待機点で任意の点です。次のMOVTで最初に点4000へ移動することになっているので点3999が図面上の基準点になります(MOVTは1つ前の点からの相対移動になるためです)。

点番号に相当する配列に0セットすると直線、1をセットするとCW円弧補間、2をセットするとCCW円弧補間、3で終端としています。このようにMOVTでは図面上の点を直接扱うことができます。
また、画像処理で補正する場合には、あらかじめAFFIN変換により各点を回転補正した上で作業点のXY座標を修正すれば良いことになります。
次のプログラムではMOVTとMOVTの間でいくつかのステートメントが実行されます。このため、点と点の間があまりに短いと、MOVTの連続実行が間に合わなくなる場合がありますので、注意してください。
その場合、位置制御そのものには矛盾が生じませんが、パルスが途切れるために機械系にストレスを与えます。
ここでは、HOUT X_A;DS_DACL,HOUT X_A;EN_DACLという特殊なコマンドが使われています。
それぞれの意味は、減速を無効、減速を有効にするというものです。
減速を無効にしておかないと通常のMOVS等と同様、最初のMOVTの終点で減速してしまうために以後のパルス発生が低速になってしまいます。しかし、終端では逆に減速を有効にしておかないと急停止になってしまいますので最後のMOVTのあとで減速を有効にします。MPG-314ではトレース移動を動的に決定できるために始点と終端をICに知らせることが必要です。

```
DIM D(100)
  SETP 3999 0 0 0 0
  SETP 4000 1000 0 0 0
  SETP 4001 2000 0 0 0
  SETP 4002 2000 1000 2000 500
  SETP 4003 1000 1000 0 0
  SETP 4004 0 1000 0
  SETP 4005 0 0 0 500
  SETP 4006 0 0 0 0
  SETP 4007 0 0 0 0
    D(0)=0
    D(1)=0
    D(2)=2
    D(3)=0
    D(4)=0
    D(5)=2
    D(6)=3
    D(7)=3
  PG &H410
  ACCEL 2000
  CLRPOS
  MOVS 1000 1000 0 0
  WAIT RR(ALL_A)==0
  TIME 1000
  CP
  PRINT "START"
  GOSUB *UT1_dis1
  WAIT RR(ALL_A)==0
  CP
  END
*UT1_dis1
  HOUT X_A;DS_DACL
  FOR Nans3=4000 TO 4999
    PRINT Nans3
      Turn=D(Nans3-4000)
    SELECT_CASE Turn
      CASE 0 : MOVT X_A|Y_A P(Nans3)
      CASE 1 : MOVT X_A|Y_A P(Nans3) CW
      CASE 2 : MOVT X_A|Y_A P(Nans3) CCW
      CASE_ELSE  : GOTO *UT1_dis2
    END_SELECT
  NEXT Nans3
*UT1_dis2
  HOUT X_A;EN_DACL
  RETURN
```

予約定数は必ず大文字。

RMVT,ACCEL,PRSET ACCEL,FEED 参照

MOVZ

パルス（MPG-68K互換） **Z絶対座標移動**

■書　式
MOVZ z
 z:絶対座標 (変数、定数)
MOVZ P(n)
 n:点番号
  1≦n≦13000

■解　説
パルス発生コマンドです。MOVZは絶対座標移動です。スピード上限及び加速度はACCELコマンドで既定されその範囲内でのスピード設定はFEDZコマンドで指定できます。引き数としてはZ軸分(z)をスカラ値で与えることも、点データP(n)で指定することもできます。点データP(n)を使用した場合はP(n)のZ成分だけが使用され他の成分は無視されます。

RMVZ 参照

MPC

システム **MPC-816互換**

■書　式
MPC [1]

■解　説
MPC-684ではプログラム用コネクタの5,6番ピンのショート/解放を検出し、パワーオンリセット時にプログラムモードか自動実行モードかを選択します。しかし、MPC-816ではこの検出を随時行っておりプログラム用コネクターを解放することで直ちにプログラムを実行します。この動作とMPC-684を整合させるのがこのコマンドです。MPC<ENTER>でMPC-816と同じ仕様になります。元に戻すにはMPC 1<ENTER>と入力します。タスク0をEND終了すると正常に動作しません。

MPCINIT

メンテナンス **SRAM初期化**

■書　式
MPCINIT [n]
 n無し 初期化
 n=4 無入力警告メッセージ（仕事しないなら‥）の禁止　＜サポート:REV2.21以上＞
新しいボードは使用前にMPCINITとERASEを実行してして下さい。

■解　説
n無し MPC-684のRAM上のプログラム、ポイントデータ、変数、パラメータエリアの初期化を行います。MPC-684システムの入れ替え後や全てのデータを初期化するときに実行します。

n=4 MPC-684はオンライン状態で一定期間入力が無いと警告メッセージを出力しますが、パソコンと接続したまま使用する場合には不都合が生じることがあります。この警告メッセージは"MPCINIT 4"と入力することで禁止されます。

ERASE 参照

MR

MPC-LNK **受信データの有無を確認**

■書　式
MR(n [+ad])
 n:LNKアドレス 1～16
 +ad:リンクボードアドレス
戻り値:nのLNKにデータが有れば1無ければ0

■解　説
受信データの有無の確認を行います。返される値は16ビットで各ビットがLSBから順に#1～#16を意味します。また、引き数に1から16の値を設定すると、相当するアドレスのバッファのみをテスト、1か0を返します。1であればバッファにデータが既に入っているということです。

```
WAIT MR(5)=1
M_R 5
```

MX

MPC-LNK **相手受信バッファデータの有無確認**

■書　式
MX(n [+ad])
 n:LNKアドレス 1～16
 +ad:リンクボードアドレス
戻り値:nのLNKにデータが有れば1無ければ0

■解　説
送出する前に相手の受信バッファをチェックする必用がありますが、この監視を MX(n) によって実施します。返される値は16ビットで各ビットがLSB から順に#1～#16を意味します。また、引き数に1から16の値を設定すると、相当するアドレスのバッファのみをテスト、1か0を返します。1であればバッファにデータが既に入っているということです。

```
WAIT MX(5)=0
M_X 5
```

NEW

編集 **プログラム初期化**

■書　式
NEW

■解　説
プログラムエリアの初期化を行います。点データはクリアされません。

NEWP

編集 **点データ初期化**

■書　式
NEWP

■解　説
点データエリアの初期化を行います。点データの全ての座標値を0にします。

NEXT

制御文 **繰り返し**

■書　式
FOR～TO～[STEP]
NEXT

FOR.NEXT 参照

NOT

演算 **補数**

■書　式
NOT(n)
　n:変数、定数

■解　説
nの補数を返します。

```
PRINT NOT(256)
-257
#PRX NOT(&H01)
  FFFFFFFE
#PRX NOT(&HFF)
  FFFFFF00
```

OFF

I/O **出力オフ**

■書　式
OFF A1 [A2 A3 A4 A5 A6]
　A1～A6:出力番号

ON,OUT 参照

ON

I/O **出力オン**

■書　式
ON A1 [A2 A3 A4 A5 A6]　　出力オン
OFF A1 [A2 A3 A4 A5 A6]　出力オフ
　A1～A6:出力ポートナンバー（変数も可）
　　-1～-8192はメモリーI/O

■解　説
出力ポートのビット単位の操作です。

```
10  CONST sol1 0
20  CONST sol2 1
30 '
35  DO
40    ON sol1 : OFF sol2
```

```
50    TIME 500
60    OFF sol1 : ON sol2
65    TIME 500
70  LOOP
#

10  DO
20    FOR i=0 TO 47
30      ON i : TIME 500
40      OFF i : TIME 500
50    NEXT i
60  LOOP
```

OUT,OFF 参照

OR

演算 **論理結合式(論理和)**

■書　式
(式) OR (式)

■解　説

```
IF SW(A)==1 OR SW(B)==1 THEN
  (制御文)
END_IF

DO
  (制御文)
LOOP UNTIL A==1 OR B==1
```

パルス発生の停止条件では論理結合はできません(次記の下線部)

```
MOVE -- UNTIL A==1 OR B==1
```

AND 参照

OUT

I/O **パラレル出力**

■書　式
OUT n m
 n:出力データ
  0≦n≦255
 m:出力ポートバンクナンバー

■解　説
出力ポートのバイト単位の操作です。nは0～255までのバイト値で、mは、ポートバンクの値です。例えばポートの0～7まではバンク0となります。

```
10   '  バンク0～5を順にオン・オフします
20  DO
30    FOR i=0 TO 5
40     OUT 255 i : TIME 100
50     OUT 0 i : TIME 100
60    NEXT i
70  LOOP
```

mが-1から-1024はメモリーI/Oです。

```
OUT &H0F -32
PRX IN(-32)
 000F
```

I/Oサイズ指定予約定数を使って出力バイト数を指定できます。

```
OUT n m~Lng             /* 4バイト出力
OUT n m~Wrd             /* 2バイト出力
OUT n m~Int             /* 2バイト符号付出力
```

次記はメモリI/Oのパラレル入出力の例です。

```
#OUT &H12345678 -1~Lng/*メモリI/0のバンク1～4へ出力
#PRX IN(-1~Lng)
 12345678

#OUT &H8000 -1~Wrd      /* メモリI/0のバンク1～2へ出力
#PR IN(-1~Wrd)          /* 2バイト入力
32768
#PR IN(-1~Int)          /* 2バイト符号付入力
-32768
```

通常のI/Oへも対応します。

```
#OUT &H1234 0~Wrd       /*バンク1(上位)と0(下位)に出力
#PRX IN(0~Wrd)          /*バンク1(上位)と0(下位)を入力
 1234
 ||||
 |||IN(0)の下位(0~3)
 ||IN(0)の上位(4~7)
 |IN(1)の下位(8~11)
 IN(1)の上位(12~15)
```

ON,OFF,IN 参照

P

パルス（MPG-68K互換） **点データ**

■書　式
P(n)
 n:点番号
  0≦n≦13000

■解　説
P(n)は特殊な予約配列で、点データを表します。点データは4つの座標値から成り立っています。従ってP(n)をMOVE, MOVZ, GOコマンドで使用するとそれぞれベクトル値として扱われ必要な座標成分が使用されます。又、printでは4つの成分を表示することとしています。この点データはX(n),Y(n),U(n),Z(n)という4つの予約配列として使用することが出来ます。これらは配列の為代入も可能です。点データにそのままデータを設定するにはSETPコマンドがあります。ティーチングの場合は番号を指定することによって直接現在位置を点データに記録することが出来ます。尚、nが0の時は現在位置を表します。各成分X(0),Y(0),Z(0),U(0)については現在位置の各成分の値を表します。現在位置はMPG毎に異なるため現在どのMPGにアサインされているかの注意が必要です。

```
#setp 10 100 200 300 400        /*P(10)を設定
#print p(10)                    /*P(10)の表示
 100 200 300 400
#stp 50 50 50 50                /*現在位置の設定
#print p(0)                     /*現在位置の表示
50 50 50 50
#clp                            /*現在位置のクリアー
#print p(0)                     /*現在位置の表示
```

```
0 0 0 0
#SETP 1 X(0) Y(0) U(0) Z(0)      /*現在位置をP(1)として登録
#MOVE P(10)                      /*P(10)にMOVE
#STP                             /*現在位置の表示
 X= 100 Y= 200 U= 300 Z= 0
#MOVE P(1)                       /*P(1)へMOVE
#STP
 X= 0 Y= 0 U= 0 Z= 0
```

P_HSW

I/O **MPC-684 J4コネクタの入出力**

■書　式
P_HSW(n)
 n:ポート番号 0≦n≦7

P_SW 参照

P_IN

I/O **MPC-684 J4コネクタの入出力**

■書　式
P_IN(0)

P_SW 参照

P_IN

I/O **MPC-684 J4コネクタの高速入力**

■書　式
P_IN(1,0)　パラレルラッチデータの読み出し。0ならストローブ未着。ストロー ブがあれば&H80以上の値になる。&h7fで7ビットのパラレルデータを得ることができる。一度読むと0に戻る。
P_IN(&H83,0)　ストローブ(P_SW(7))オンでラッチ指定。この時の返し値は-1。
P_IN(&H3,0)　ストローブ(P_SW(7))オフでラッチ指定。この時の返し値は-1。

■解　説
J4はDC24V入力のポートです。相手がTTL信号出力機器の場合、フォトカプラなどにより、オープンコレクタに変換して使用ください。
また、J4-15,16からJ5-1へのパタンを一箇所ハンダ面でカットし、RA5を 2.7K程度の抵抗に交換すれば、J4のIOを5V対応にすることも可能です。(ユーザ責任)

```
10 'p_sw(7) on strobe
20 PRINT P_IN(&H83,0)
30 'when p_sw off strb
40 ' PRINT P_IN(&H3,0)
50 *test_start
60 DO
70 aaa=P_IN(1,0)
80 IF aaa<>0 THEN : BREAK LOOP : END_IF
90 TIME 1000 : PRINT "Not Yet !!"
100 LOOP
110 PRX &H7F&aaa
```

P_LD

ファイルメモリ **フラッシュ ROMから点データを読み込む**

■書　式
P_LD [begin,end]
　begin:読込み開始点データ
　end:読込み終了点データ
　　1≦begin<end≦5000

■解　説
フラッシュ ROMに保存された点データを取り出して復旧します。引数が無いと全点データ、指定するとその範囲のみ復旧します。

```
#P_LD 100 110        ←点100から110のみを取り出してP(100)-P(110)に書き込みます。
```

P_SV 参照

P_OFF

I/O **MPC-684 J4コネクタの入出力**

■書　式
P_OFF A
　A:ポート番号
　　0≦A≦3

P_SW 参照

P_ON

I/O **MPC-684 J4コネクタの入出力**

■書　式
P_ON A
　A:ポート番号
　　0≦A≦3

P_SW 参照

P_OUT

I/O **MPC-684 J4コネクタの入出力**

■書　式
P_OUT n
　n:出力パターン
　　0≦n≦&HF

P_SW 参照

P_SV

ファイルメモリ **フラッシュ ROMに点データを保存**

■書　式
P_SV

■解　説
点データをフラッシュエリアに書き込みます。書き込み中（数秒）はマルチタスクが停止しますので、他のタスクの作業が休止中に実施してください。保存される点データは1～5000です。

```
#P_SV
**--          ←保存中表示
```

P_LD 参照

P_SW

I/O **MPC-684 J4コネクタの入出力**

■書　式
P_ON A
 A:ポート番号
  0≦A≦3
P_OFF A
 A:ポート番号
  0≦A≦3
P_OUT n
 n:出力パターン
  0≦n≦&HF
P_SW(n)
 n:ポート番号
  0≦n≦7
P_HSW(n)
 n:ポート番号
  0≦n≦7
P_IN(0)

■解　説

P_ON	ポート（J4）のビットオン
P_OFF	ポート（J4）のビットオフ
P_OUT	ポート（J4）のパラレルアウト
P_SW(n)	ポート（J4）のビット入力（5msecﾌｨﾙﾀｰ付き）
P_HSW(n)	ポート（J4）のビット入力（ﾌｨﾙﾀｰ機能無し）
P_IN(n)	ポート（J4）のパラレル入力

前記のコマンドはいずれもMPC-684に付属のI/OポートJ4に対するコマンドで、それぞれMIP/MOPに対するON/OFF/SW(n)などのコマンド・関数に対して同等の機能を持ちます。ポート番号とピンアサインの関係は次の通りです。

ピン	コマンド・関数	
1	P_SW(0)/P_HSW(0)	P_IN(0)　入力
2	P_SW(1)/P_HSW(1)	
3	P_SW(2)/P_HSW(2)	
4	P_SW(3)/P_HSW(3)	
5	P_SW(4)/P_HSW(4)	
6	P_SW(5)/P_HSW(5)	
7	P_SW(6)/P_HSW(6)	
8	P_SW(7)/P_HSW(7)	
9	P_ON 0/P_OFF 0	P_OUT n　出力
10	P_ON 1/P_OFF 1	
11	P_ON 2/P_OFF 2	
12	P_ON 3/P_OFF 3	

```
#P_ON 0                 'OP1をON
#PRX P_IN(0)            'パラレル入力上位バイトが出力、下位バイトが入力
 0100                   ・・・出力はOP1がON、入力は全部OFF
#PRINT P_SW(0)          'IN1のビット入力
0                       ・・・入力IN1はOFF
#P_OUT &H03             'パラレル出力
#PRX P_IN(0)
 0300                   ・・・出力はOP1,OP2がON入力は全部OFF
```

前記例のように、P_IN(0)に対しては、下位バイトが入力ポート、上位バイト(4bitのみ)が出力ポートのビットパタンとなります。尚、このコマンドはMPC-684に対して有効です。MPG-68KのJ4に対しては、HPT(入力) HOUT(出力)を使います。
MPG-68Kの原点入力のJ4コネクターの入出力はHOUT,HPT(n)でおこないます。

PALET1

パルス（MPG-68K互換）　　パレット宣言

■書　式
PALET1 P(i) P(j) P(k) m n
PALET2 P(i) P(j) P(k) m n
PALET3 P(i) P(j) P(k) m n
PALET4 P(i) P(j) P(k) m n
 i,j,k:ポイントナンバー
 m,n:マトリクス数

■解　説
パレット宣言です。P(i),P(j),P(k)によって構成された長方形（平行四辺形）をm,n分割したパレット（マトリクス）とします。パレットは1～4迄であり設定されたパレットの点は関数PL1(n)～PL(n)にて使用します。パレット計算は4次元で実施されますので傾斜のあるパレットも対応可能です。

```
10    PG &HE0 0
20    PALET1 P(1) P(2)P(3) 5 5          /*PALET1宣言
30    PALET2 P(11) P(12) P(13) 10 10    /*PALET2宣言
40      a=0 :   b=0
50   '
60    DO
70        a=a+1
80      IF a>25 THEN
90          a=1
100     END_IF
110     JUMP PL1(a)                     /*PALET1へJUMP
115  '
120       b=b+1
130     IF b>100 THEN
```

```
140         b=1
150      END_IF
160      JUMP PL2(b)                    /*PALET2へJUMP
170   LOOP
#
```

PALET2

パルス（MPG-68K互換） **パレット宣言**

■書　式
PALET2 P(i) P(j) P(k) m n
 i,j,k:ポイントナンバー
 m,n:マトリクス数

PALET1 参照

PALET3

パルス（MPG-68K互換） **パレット宣言**

■書　式
PALET3 P(i) P(j) P(k) m n
 i,j,k:ポイントナンバー
 m,n:マトリクス数

PALET1 参照

PALET4

パルス（MPG-68K互換） **パレット宣言**

■書　式
PALET4 P(i) P(j) P(k) m n
 i,j,k:ポイントナンバー

PALET1 参照

PAUSE

タスク操作 **タスク一時停止**

■書　式
PAUSE A
 A:タタスク番号　1～31

■解　説
タスクの一時停止。実行中のタスクを一時停止するものです。再開はCONT。

```
10    FORK 1  *task1
20    DO
30      WAIT SW(192)==1
40      PAUSE 1
50      WAIT TASK(1)==-3
55      PRINT "task1 pause"
```

```
60      WAIT SW(192)==0
70     CONT 1
75     WAIT TASK(1)==0
77     PRINT "task1 continue"
80    LOOP
90   '
100  *task1
110   DO
120     FOR i=0 TO 47
130      ON i :TIME 100 :OFF i : TIME 100
140      NEXT i
150   LOOP
```

FORK,QUIT,CONT 参照

PG

パルス（MPG-314専用） **PG宣言**

■書　式
PG adrs [task]
　adrs:MPG-314ボードアドレス　&H400,&H410...&H490
　task:タスク番号
パラメータ省略で現在の設定値表示

■解　説
MPG-314の使用宣言文です。adrsによって指定されたMPG-314をtaskで指定されたタスクに引き当てます。例えば PG &H400 1 とすればタスク1で実行されるMOVEやACCELはアドレス &H400のMPG-314に対してのコマンドとなります。この宣言が一度実行されるとRAM上に保存されリセットの度にインターフェースは初期化されます。PG<ent>とすると、現在の引き当て状況のリストを表示します。
尚、PGはティーチング中の切り替えコマンドではありません。この場合はPGSELを使用してください。

```
#PG &H400 0      /* タスク0で&H400のMPG-314を制御
#PG &H410 1      /* タスク1で&H410のMPG-314を制御
#PG &H410 2      /* 1枚のMPGを複数のタスクで制御することもできる(単軸制御など)
#PG              /* 引き当て確認
      指定されているPGアドレス-&H400

        TASK  0 for PG &h400    TASK  1 for PG &h410
        TASK  2 for PG &h410    TASK  3 for PG &h000
                              (以下略)
```

タスク番号を省略するとPGコマンドを実行したタスク自身への引き当てになります。

```
  FORK 1,*TASK1              /* TASK1起動
  PG &H400                   /* &H400のMPG-314をTASK1へ引き当て
  ACCEL X_A|U_A -1 10000     /* このACCELは&H400のMPG-314へ発行される
  (略)
  END
*TASK1
  PG &H410                   /* &H410のMPG-314をTASK1へ引き当て
  ACCEL X_A|U_A -1 10000     /* このACCELは&H410のMPG-314へ発行される
  (略)
```

1つのタスクの中でPG宣言を切り替えて使うこともできます。

```
PG &H400                     /* 次からのコマンドは&H400のMPG-314へ
ACCEL X_A|U_A -1 10000
FEED X_A|U_A 0
PG &H410                     /* 次からのコマンドは&H410のMPG-314へ
ACCEL X_A|U_A -1 10000
FEED X_A|U_A 0
```

PG

パルス（MPG-68K互換） **PG宣言**

■書　式
PG adrs [task]
　adrs:MPG-68K/405ボードアドレス　&HE0,&HE4,&HE8,&HEC...
　task:タスク番号
パラメータ省略で現在の設定値表示

■解　説
MPG-68K/405の使用宣言文です。adrsによって指定されたMPGをtaskで指定されたタスクに引き当てます。例えば PG &HE0 1 とすればタスク1で実行されるMOVEやACCELはアドレス &HE0のMPGに対してのコマンドとなります。
この宣言が一度実行されるとRAM上に保存されリセットの度にインターフェースは初期化されます。PG<enter>とすると、現在の引き当て状況のリストを表示します。
尚、PGはティーチング中の切り替えコマンドではありません。この場合はPGSELを使用してください。

```
#PG &HE0 0            /* タスク0で&HE0のMPGを制御
#PG &HE4 1            /* タスク1で&HE4のMPGを制御
#PG &HE8 2            /* タスク2で&HE8のMPGを制御
#PG                   /*引き当て確認
      指定されているPGアドレス-&H0E0
        TASK  0 for PG &h0E0    TASK  1 for PG &h0E4
        TASK  2 for PG &h0E8    TASK  3 for PG &h000
                          (以下略)
```

PGSEL

パルス（MPG-68K互換） **PGボード選択**

■書　式
PGSEL [n]
 n:タスク番号
　0≦n≦23

■解　説
PGコマンドではタスクとスレーブボードを結合しますが、それでは調節時に支障をきたします。調整時のコマンド実行、ティーチモードは全てタスク0によって実施されます。PGSELはこの問題を解決するためにタスク0に一時的に他のタスクのMPCを割り付けものです。PGSEL<ENTER>の後<TAB>, <+>, <->のキーによって希望のMPGを選択すると以降コマンドは選択されたMPGに対して有効となります。PGSELによる選択はRUN、リセットによりクリアされます。ティーチモードにもこの機能は含まれています。又、引き数を与えれば直接目的のMPGを選択します。

```
pgsel
 PG[1,E4]
#
      |  |
      |  |_____スレーブボードのアドレス
      |________タスクナンバー
               <TAB>, <+>キーでアップ
               <->キーでダウン
               <Q>で編集画面へ戻る
```

PL1

パルス（MPG-68K互換） **パレットポイント**

■書　式
PL1(n)
PL2(n)
PL3(n)
PL4(n)
n:パレタイズマトリックスナンバー。 n が負の値のときはZIGZAG動作となります。

■解　説
PALET1～4で設定されたパレットの点データを与えます。nの値は1から始まりx方向(P(i)→P(j)でm分割)に増加します。mに達するとx方向はリセットされ、y方向(P(i)→P(k)で分割)に1ピッチ増加します。計算式では次のようになります。(m-1),(n-1)が最後にあるのは精度確保の為除算を最後にしている為です。

```
dx=P(j)-P(i)
dx=P(k)-P(i)
PL(n)=((n-1)%m*dx/(m-1)+((n-1)/m)*dy/(n-1)+P(i)
```

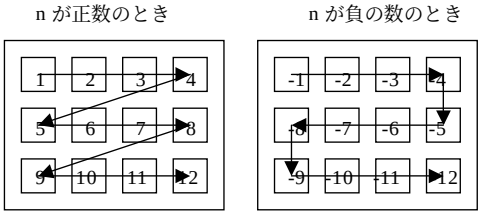
例えばn=1でPLはP(i)そのものとなります。nがmの場合は次のようにP(j)そのものとなります。

```
PL(n)=(m-1)*dx/(m-1)+P(i)
PL(n)=dx+P(i)=P(j)
```

尚、パレットポイントの座標成分を必要とする場合は次のようにパレットデータを点データに移してから使用して下さい。

```
SETP 3 PL1(n)
MOVE X(3) Y(3) 0
```

パレタイズマトリックスナンバーが負の値のときは次の図の動きになります。
列間の移動が小さくスピードアップになります。



※注意
パレタイズマトリックスナンバーがカウントオーバーしても、同じピッチで継続します。プログラムで進行状態を管理してください。

PALET1 参照

PL2

パルス（MPG-68K互換） **パレットポイント**

■書　式
PL2(n)
n:パレタイズマトリックスナンバー

PALET1,PL1 参照

PL3

パルス（MPG-68K互換） **パレットポイント**

■書　式
PL3(n)
n:パレタイズマトリックスナンバー

PALET1,PL1 参照

PL4

パルス（MPG-68K互換） **パレットポイント**

■書　式
PL4(n)
n:パレタイズマトリックスナンバー

PALET1,PL1 参照

PLIST<PLS>

編集 **点データ表示**

■書　式
PLIST　〔省略形 PLS〕

■解　説
P(n)の一括表示です。1画面単位で表示の継続中止を聞いてきますので、中止の場合はq/Qを入力して下さい。どれかのキーを押すと継続します。

PLSC

パルス（MPG-314専用） **一定速パルス発生**

■書　式
PLSC n m [pls] [n1,m1,pls1]
n:軸指定予約定数
m:パルスレート(pps)
pls:パルス量(省略時無限パルス発生)
デューティーは50%固定

■解　説
一定速パルス発生です。加減速はありません。引数の与え方で定量パルス発生にも、無限パルス発生にも使用でます。注意として指定パルスレート値によってMCX-314上のACCELパラメータが破壊されるのでPLSC実行後に通常のパルス発生コマンドを使用する場合は、PRSET_ACCELを実行します。移動量を規定しない無限パルス発生ではplsを省略します。この場合のCW/CCW方向規定はmの正負によって決定されます。移動量を規定する場合はplsの値の正負によって方向が決定されます。例では、X軸でパルス発生してXS2(IN1)入力で停止、その後PRSET_ACCELによってACCEL設定を回復し通常のRMVSを使用しています。PRSET_ACCELがないとRMVSでのパルス発生も加減速無しの一定速となります。

```
10 PG &H410
20 ACCEL 10000
40 STOP X_A IN1_ON
```

```
60 PLSC X_A -100
70 WAIT RR(X_A)==0
80 PRSET_ACCEL X_A
90 RMVS X_A 1000
```

ACCEL,PRSET_ACCEL 参照

PR

RS-232C **データ表示**

■書　式
PR　(PRINTの省略形)

PRINT 参照

PRF

コプロ演算 **内部データ表示**

■書　式
PRF fn1,fn2,fn3,..
 fn1,fn2,fn3,..:FPnの番号 7を与えるとアキュームレータ

■解　説
コプロの不動小数点レジスタFP0,FP1..のデータを表示します。

SETF, GETF, CALF 参照

PRINT<PR>

RS-232 **データ表示**

■書　式
PRINT (省略形　PR)
PRINT#0
PRINT#2

■解　説
　CH1　文字列・数値の表示
　CH0　文字列・数値の表示
　CH2　文字列・数値の表示
引き数間にタブ、スペースなどのデリミタはありません。また、PRINT#では改行コードすら出力されません。従って、外部との通信はPRINTコマンドのみで自由なフォーマットを構成することが出来ます。また次の文字列は特別な意味を持ちます。
　\n　ニューラインコード(CR-LF)
　\r　リターンコード
　\t　タブコード

```
PRINT "HEX$(ASC(xyz$))=" HEX$(ASC(xyz$))
PRINT CHR$(ASC("A")) ASC(CHR$(ASC("A")))
PRINT 1 "\t" 2 "\t" 3
```

次の場合はCH0より&H1B &H2A &H30 &H31 &H32と出力されます。

```
PUT#0 &H1B &H2A
PRINT#0 "012"
```

PRINT#で通信相手側がCR,LFコードを必要とする場合(N88BASIC INPUTｺﾏﾝﾄﾞなど)には次のように記述します。

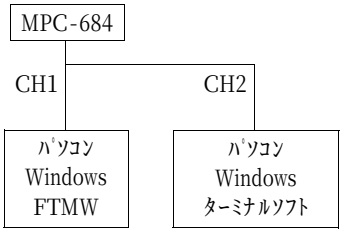
```
PRINT#0 ACCEL\n                          文字列“ACCEL”とCR･LFｺｰﾄﾞを出力
variable=123
PRINT#0 variable CHR$(13)                CH0へ数値123とCRｺｰﾄﾞを出力
PRINT#2 variable CHR$(13) CHR$(10)       CH2へ数値123とCR･LFｺｰﾄﾞを出力
```

PRINT#でのRS-232バッファの内容はRSコマンドで確認できます。

パラメーター間のスペースが無いと...

PRINT#2 a$ " ok\n" とするところを a$" ok\n" (a$と" ok\n"の間のｽﾍﾟｰｽが無い) としてしまうと、a$" ok\n"は数値が0の1つの変数として扱われてしまいます。MPCで①のプログラムを実行するとCH2に接続したターミナル側では②の様になります。

実験機器構成



①MPC側のプログラム

```
10        CNFG#2 "9600b8pns1NONE"
20        PRINT#2 "\n"
30        DO
40          INPUT#2 "a$=" a$
45          PRINT a$
50          PRINT#2 a$" ok\n"          <--送られてきた文字列を返したいのだが..
60        LOOP
```

②ターミナル表示画面

```
a$=0a$=0a$=0a$=0a$=0a$=0a$=0a$=0a$=0a$=        <--返ってくるのは0ばかり
```

③CH1でも同様です。

```
#PRINT a$              <--a$は1です。
1
#PRINT a$" ok\n"       <-- a$" ok\n"は一つの変数として扱われます。
0
#PRINT a$ok\n          <-- a$ok\nは一つの変数として扱われます。
0
#PRINT a$ ok\n         <--この場合はa$とok\nという2つの変数になります。
10
#PRINT a$ "ok\n"       <--これがベスト。
```

""の中のスペースは文字として扱います。
HEX形式の表示はPRXです。

PRX 参照

PRINT#0

RS-232 **データ出力**

■書　式
PRINT#0

PRINT 参照

PRINT#2

RS-232 **データ出力**

■書　式
PRINT#2

PRINT 参照

PROTOCOL

MBK-SH/RS **MBK-RS（デジタルGPダイレクトアクセス）**

■書　式
PROTOCOL MEWNET

■解　説
MPC-684のCH0で松下のMEWNETプロトコルに準拠した通信を行います。デジタル社GPシリーズとダイレクトアクセス方式で接続できます。"PROTOCOL　MEWNET"とプログラムに一行加筆すると有効になります。停止する場合はPROTOCOL MEWNETの行を削除の上"PROTOCOL 0" もしくはMPCINITを実行します。

この通信は9600bpsで、CPUにも負担がかかります。高速通信、高タクトタイムが必要な場合はMBKを使ってください。
MBK-RS実行時、タスク31は使用できません。
9600pbs,データ8,ストップ1,パリティ無し,XONXOFF無し
010921 3.81zでβ公開。

PRSET_ACCEL

パルス（MPG-314専用） **ACCELパラメータ復旧**

■書　式
PRSET_ACCEL n
　n:軸指定予約定数 X_A～Z_A

■解　説
ppsを指定する一部のコマンド(PLSC,HOME X_A)によって破壊されたACCELパラメータを復旧します。
PRSET_ACCELはIN0～3条件停止もクリアします。

```
PRSET_ACCEL X_A        /* X軸のパラメータ復旧
PRSET_ACCEL Z_A U_A    /* Z,U軸のパラメータ復旧
```

予約定数は必ず大文字。

ACCEL 参照

PRX

RS-232 **データHEX表示**

■書　式
PRX A
　A:整数値，変数

■解　説
PRINTと同様の目的のコマンドですが表示がヘキサ表現となります。I/Oの状態を直接見たい時に有効です。

```
#a=&HFF
#PRINT a              <--10進表示
 255
#PRX a                <--ヘキサ表示
 00FF
#PRX IN(24)
 0055
```

PRINT 参照

PULSE_OUT

I/O **出力ポート パルスON/OFF**

■書　式
PULSE_OUT port time count
  port: 出力ポート(port 0は使えません)
  time: 時間 (0.1秒単位、0は停止、省略不可)
  count: 回数 (省略すると無限)

■解　説
出力ポートのON/OFFをワンショットから回数指定までできる出力コマンドです。パトライトや照光SWの点滅などが簡単に行えます。

```
PULSE_OUT 10 1 1      /* ポート10を0.1秒ワンショットオン
PULSE_OUT 20 10       /* ポート20の点滅を1秒ON1秒OFFで繰返し
```

点滅の停止方法
PULSE_OUT 20 0 というように時間パラメータを0にして実行します。(add 031203)
コマンドサポート REV-3.84k以降。一度に制御できるポートは24ポートまでです。TASK 30を使用しています。count省略で無限回数となるのはRev-3.85c以降、それ以前は32767回。前記記停止機能はRev-3.85e以降。

PUT

RS-232 **データ表示**

■書　式
PUT [A1 A2 A3 A4 A5 A6]
PUT#0 [A1 A2 A3 A4 A5 A6]
PUT#2 [A1 A2 A3 A4 A5 A6]
  A1～A6:キャラクター

■解　説

PUT
コンソールへ1文字ずつ出力します。

```
#put &h41 &h42 &h43 &hd &ha
ABC
#
```

PUT#0,PUT#2
A1～A6の数値をアスキーコードとしてCH0もしくはCH2により出力します。特にPRINT#ではヌルコードが出力できないためにこのコマンドが有効です。例えば、NULL'A'と出力するには次の通りです。

```
PUT#0 0 &H41
```

PUT#0

RS-232 **データ出力**

■書　式
PUT#0 [A1 A2 A3 A4 A5 A6]
　A1～A6:ｷｬﾗｸﾀ

PUT 参照

PUT#2

RS-232 **データ出力**

■書　式
PUT#2 [A1 A2 A3 A4 A5 A6]
　A1～A6:ｷｬﾗｸﾀ

PUT 参照

Q_PAUSE

パルス（MPG-68K互換） **クイックポーズ**

■書　式
Q_PAUSE　　クイックポーズ設定
Q_PAUSE -1　クイックポーズ設定解除

■解　説
STOPによるパルス発生停止後にタスクのPAUSEを自動的に行います。コマンドQ_PAUSEを実行すると次のようにSTOPコマンドに機能が追加されます。これまでSTOPコマンドを実行すると発生中のパルスは停止させられましたがパルス発生に続くコマンドはそのまま実行されていました。これに対して、Q_PAUSEモードではSTOPコマンドの実行によりパルス発生を中止し、さらにそのタスクをポーズとします。状態の監視はBSY()関数とTASK()関数によって行います。再開はCONTコマンドのみで有効となります。途中で停止させられたパルス発生は強制的に再実行されます。この時RMOV, RMVZ, RM、RULSEの4つのコマンドは再実行されません。これは相対移動の場合目的地が明確で無いことによります。従ってパルス発生中のタスクの停止・再開が複雑なインタロックによることなく実現できます。Q_PAUSEモードは"Q_PAUSE -1"で解除されます。工場出荷時は"Q_PAUSE -1"の状態でこれまでのソフトがそのまま動作します。"Q_PAUSE"有効下ではSTOP 3も有効となります。これはパルス発生の終了時（1つのコマンド終了時）にただちにポーズとする機能です。STOP 1,2の途中停止無しのモードです。

- STOPコマンドが有効になるのはPG宣言されているタスクだけです。PG宣言されていないタスクをポーズするのはPAUSEコマンドです。
- CONTコマンドはEN_PGも併せて実行します。
- UNTILによる条件停止ではPAUSE状態になりません。

```
10      Q_PAUSE
20      PG &HE4 0
30      PG &HE4 1
40      FORK 1 *TASK1
50      DO
60        WAIT SW(192)==0
70        WAIT SW(192)==1
80        STOP 1 1              <--ﾀｽｸ1でMOVE中に減速停止をかけます
90        WAIT TASK(1)==-3      <--ﾀｽｸ1がPAUSE状態になるのを待ちます
100       PRINT X(0) Y(0)       <--停止した所の座標値を表示
110       WAIT SW(192)==0
120       WAIT SW(192)==1
130       CONT 1                <--PAUSE中のﾀｽｸ1を再開します
140       WAIT TASK(1)<>-3
150     LOOP
160     END
```

```
170    *TASK1
180      SETPOS 0 0 0 0
190      DO
200        MOVE 100000 100000 0 <--RMOV,RMVZ,RM,PULSEｺﾏﾝﾄﾞは無効
210        MOVE 0 0 0
220      LOOP
230      END
#RUN
40602 40602                        <--SW(192)がONして減速停止した所の座標値です。
84677 84677                           ﾀｽｸ1は停止した所でPAUSE状態になり、
84826 84826                           CONTで再び再開してﾊﾟﾙｽ発生を行います。
50639 50639                           到達座標はMOVEで与えられた座標です。
27944 27944
```

前記プログラムの文番号80のSTOP 1をSTOP 3にすると

```
80         STOP 3 1

#RUN
100000 100000  <--MOVEで与えられたﾊﾟﾙｽを出し終わってからﾀｽｸ1はPAUSE状態になります。
0 0
100000 100000
```

非常停止、サイクルストップSW、ポーズSW監視タスクから別タスクのパルス発生を停止する。

```
  TIME 100
  SETIO
  Q_PAUSE                  <--Q_PAUSEﾓｰﾄﾞ宣言
'  IN PORT
  CONST pause_sw 192       <--ﾘﾐｯﾄｽｲｯﾁ
  CONST emg_sw 193         <--非常停止ｽｲｯﾁ
  CONST cycle_sw 194       <--ｻｲｸﾙ停止ｽｲｯﾁ
'  OUT PORT
  CONST pause_led 0
  CONST emg_led 1
  CONST cycle_led 2
  FORK 1 *PULSE
'************************************
' 非常停止、PAUSE SW監視
'************************************
  PG &HE0
*MAIN
  DO
    IF SW(pause_sw)==1 THEN
      GOTO *PAUSE
    END_IF
    IF SW(emg_sw)==1 THEN
      GOTO *EMG
    END_IF
    IF SW(cycle_sw)==1 THEN
      GOTO *CYCLE
    END_IF
  LOOP
*PAUSE
  PRINT "PAUSE SW停止処理"
  STOP 1 1                 <--減速停止
  WAIT TASK(1)<0           <--ﾀｽｸが実行中でなくなるのを待つ
  GOSUB *TASK_STAT
  PRINT "現在点は" P(0)
  ON pause_led
  WAIT SW(pause_sw)==0
  OFF pause_led
  CONT 1
  WAIT TASK(1)>=0
  GOSUB *TASK_STAT
  GOTO *MAIN
*EMG
  PRINT "EMG SW処理"
```

```
  STOP 2 1                  <--急停止
  WAIT TASK(1)<0            --+ ﾀｽｸ1が実行を停止したら
  QUIT 1                    --+ QUITします。
  WAIT TASK(1)==-2
  GOSUB *TASK_STAT
  DO
    ON emg_led
    TIME 100
    OFF emg_led
    TIME 100
  LOOP
*CYCLE
  PRINT "CYCLE停止処理"
  STOP 3 1                  <--MOVEに与えられたﾊﾟﾙｽを出力してから停止
  WAIT TASK(1)<0
  GOSUB *TASK_STAT
  PRINT "現在点は" P(0)
  ON cycle_led
  WAIT SW(cycle_sw)==0
  OFF cycle_led
  CONT 1
  WAIT TASK(1)>=0
  GOSUB *TASK_STAT
  GOTO *MAIN
*TASK_STAT                  <--これはﾀｽｸ1の状態を表示するｻﾌﾞﾙｰﾁﾝです
  SELECT_CASE TASK(1)
    CASE -1 : PRINT "ﾀｽｸ未使用"
    CASE -2 : PRINT "ﾀｽｸQUIT状態"
    CASE -3 : PRINT "ﾀｽｸPAUSE状態"
    CASE_ELSE  : PRINT "ﾀｽｸ実行中"
  END_SELECT
  RETURN
'************************************
'パルス発生タスク
'************************************
*PULSE
  PG &HE0
  ACCEL 10000
  CLRPOS
  DO
    MOVE 100000 100000 100000
    TIME 100
    MOVE 0 0 0
    TIME 100
  LOOP
```

実行結果

前記プログラムをRUNして順番にSW(pause_sw)をON/OFF→SW(cycle_sw)をON/OFF→SW(emg_sw)をONしてみます。

```
#RUN
PAUSE SW停止処理                  <--SW(pause_sw)==1
ﾀｽｸPAUSE状態
現在点は63855 63855 63855 0       <--STOP 1 1は減速停止後ﾀｽｸ停止
ﾀｽｸ実行中                         <--SW(pause_sw)==0でﾀｽｸ再起動,ﾊﾟﾙｽ再出力
CYCLE停止処理                     <--SW(cycle_sw)==1
ﾀｽｸPAUSE状態
現在点は100000 100000 100000 0    <--STOP 3 1 はMOVE終了後にﾀｽｸ停止
ﾀｽｸ実行中                         <--SW(cycle_sw)==0
EMG SW処理                        <--SW(emg_sw)==1
ﾀｽｸQUIT状態
```

コマンドの後ろにコメントを記述しないで下さい。

```
Q_PAUSE 'ｸｲｯｸﾎﾟｰｽﾞﾓｰﾄﾞ            <--Q_PAUSE -1と同じになります。
```

STOP,BSY 参照

QUIT

タスク操作 **タスク停止**

■書　式
QUIT A
　A:タスク番号　1～31
QUIT

■解　説
　　QUIT AではAで指定されたタスクを停止します。
　　QUITのみの場合は全タスク（0を除く）の停止となります。
　　QUIT 1 2 3はタスク1,2,3を停止します。タスク0（メインタスク）は停止できません。
文字列処理やINPUT#,INP$#待ち中のタスクをQUITしないで下さい。

【Question】
a) もしSTOPコマンド及びBSY()を使用しないでパルス発生中のタスクをQUITした場合はどうなりますか？
b) パルス発生の停止を確認後、タスクをQUITしたとき、MPCの現在位置とパルスは一致していますか。
c) a,bの場合、再度FORKして、パルスを続行できますか？

【Answer】
a) パルス発生中のタスクに（STOPしないで）QUITをかけると、MPGに対する通信が折られてしまうため、次から使えなくなります。パワーオンリセットが必要となってしまいます。
b) 途中停止した場合でもパルス数と現在位置は一致します。（メカ的にズレがないとして）
c) タスクの停止とMPGの状態は、正規の手続き（通信中にQUITするなどは御法度です）さえ踏まれていれば無関係なので、もちろん停止、QUIT後の再FORKも大丈夫です。ただ、実際にパルス発生中であるかどうかの確認はBSY()だけでは難しく、問題を起こしやすいのでクイックポーズを使って下さい。例えば現在位置を取得するX(0)もMPGと通信をしますが、BSY()では時間が短すぎて検出しにくく、停止の確認後QUITという技がうまくできません。

Q_PAUSE,FORK,PAUSE,CONT 参照

_RET_VAL

制御文 **戻り値の受け取り**

■書　式
_RET_VAL var1 var2 - -
　var:変数

■解　説
RETURN戻り値を受け取ります。

RETURN 参照

RAM

デバッグ **RAMモード**

■書　式
RAM

■解　説
フラッシュ ROM書き込みを無効にします。プログラム実行時(RUN)のフラッシュ ROMへの書き込みをキャンセルします。編集後の実行が効率的になります。また、起動時のフラッシュ ROMからのプログラムの読

み込みと変数初期化もキャンセルされます。RAMモードではプログラムはプロテクトされません。デバッグが終了したらROMコマンドで元に戻しRUNまたはFIXでフラッシュ ROMへ書き込んで下さい。

```
#RAM                    ←RAMﾓｰﾄﾞ
**
#
#MPC-68K ADVFSC(r)s REV-2.52i
 BASIC like + multi tasking
 Created by ACCEL Co.'91~97
·····!! ROM化できません。ERASEしてください。      ←RAMﾓｰﾄﾞ起動時のﾒｯｾｰｼﾞ
#

#ROM                    ←ﾌﾗｯｼｭROMﾓｰﾄﾞ
**

#
```

ROM 参照

RANGE

パルス（MPG-314専用） **動作範囲の制限**

■書　式
RANGE n max min
 n:軸選択予約定数 X_A～Z_A
 max:動作範囲上限
 min:動作範囲下限

■解　説
MPG-68K互換RANEGコマンドは暗黙でSLMT_ON(ソフトリミット)を有効にしますが、このRANGE設定を使うためにはINSET_314コマンドで明示的にSLMT_ONを有効にします。
(このRANGEは入力条件の設定には関係しません)

```
RANGE X_A 1000 -1000
INSET_314 X_A SLMT_ON
```

停止は減速停止(加減速有りコマンド)=減速域分オーバーします。
予約定数は必ず大文字。

RANGE

パルス（MPG-68K互換） **動作範囲の制限**

■書　式
RANGE n
 n=0 領域設定を解除します。
 n=1 現在位置を最大領域として設定します。
 n=2 現在位置を最少領域として設定します。
 n=11 最大領域の値を現在位置に複写します。現在位置が失われるので注意
 n=12 最少領域の値を現在位置に複写します。現在位置が失われるので注意

■解　説
このRANGEコマンドはMPG-68K互換コマンドで、ソフトリミット(SLMT_ON)を自動的に有効にします。
次のプログラムの110行でSLMT_ONをorしているのは、INSET_314で入力条件を設定した時RANGE有効状態を保つためです。
(INSET_314はパラメータで与えられたもの以外はリセットします)

```
10 SETP 1 1000 500 400 200
20 SETP 2 -1000 -500 -400 -200
30 PG &H410
40 ACCEL 8000
50 SETPOS X(1) Y(1) U(1) Z(1)
60 RANGE 1
70 SETPOS X(2) Y(2) U(2) Z(2)
80 RANGE 2
90 CLRPOS
100 CP
110 INSET_314 ALL_A INP_OFF|SLMT_ON
```

また、RANGEによって停止したかどうかは、RR()関数とSLMP@(正方向ﾘﾐｯﾄ)SLMM@(負方向ﾘﾐｯﾄ)を組み合わせて検出することができます。

RR(X_A,SLMP@|SLMM@)
RR(Y_A,SLMP@|SLMM@)
RR(U_A,SLMP@|SLMM@)
RR(Z_A,SLMP@|SLMM@)

REG

MPG-3202 **X3202レジスタ読み込み**

■書　式

REG(reg)

reg: X3202のアドレス(xx)とレジスタ・カウンタセレクトコード(code)の和(&Hxx00+code)。
またはステータスレジスタアドレス(-1～-16)。

■解　説

MPG-3202のパルス発生IC「X3202」のレジスタを読込みます。regにステータスレジスタアドレスを指定すると動作状態を知ることができます。

```
PRINT REG(&H121)      /* X3202#1のﾚｼﾞｽﾀ&H21読込み
WAIT REG(-1)=&H20     /* X3202#1のｽﾃｰﾀｽﾚｼﾞｽﾀ読込み。この場合動作完了待ち
```

詳細は「MPG-3202 製品別マニュアル」参照。

CMND,REG3,ST_REG 参照

REG3

MPG-3202 **X3202レジスタ読み込み**

■書　式

REG3(reg)

reg: X3202のアドレス(xx)とレジスタ・カウンタセレクトコード(code)の和(&Hxx00+code)。
またはステータスレジスタアドレス(-1～-16)。

■解　説

MPG-3202のパルス発生IC「X3202」のレジスタを読込みます。3byte符号付きのレジスタの読込みに使用します。

```
PRINT REG3(&H121)     /*X3202#1のﾚｼﾞｽﾀ&H21読込み
```

詳細は「MPG-3202 製品別マニュアル」参照。

CMND,REG,ST_REG 参照

RENUM<RNM>

編集 **文番号ふりなおし**

■書　式
RENUM

■解　説
文番号のふりなおし。10毎にふりなおされます。

```
list 0
5     FOR i=0 TO 47
7       ON i : TIME 100
10      OFF i : TIME 100
15    NEXT i
#rnm
#list 0
10    FOR i=0 TO 47
20      ON i : TIME 100
30      OFF i : TIME 100
40    NEXT i
```

RETURN<RET>

制御文 **リターン**

■書　式
RETURN

■解　説
サブルーチンリターン。
GOSUBで呼ばれたサブルーチンはRETURNによってメインルーチンへ戻り次のステップが実行されます。

RETURN

制御文 **リターン 戻り値渡し**

■書　式
RETURN [arg1 arg2 - -]
 arg:変数、定数

■解　説
サブルーチンの結果を引き渡します。ローカル変数と組み合わせるとタスク間でのサブルーチンの共有が可能となります。受け取りは_RET_VALはで行います。

```
10 GOSUB *SUB
20 _RET_VAL A          /* 戻り値の受け取り
30 PRINT A
40 END
50 *SUB
60  C!=123             /* ﾗﾍﾞﾙ!はﾛｰｶﾙ変数
70 RETURN C!           /* C=戻り値
RUN
123
```

GOSUB 参照

RLS

タスク操作 **セマフォ開放**

■書　式
RLS(n)
　n:メモリI/O番号

■解　説
I/Oのビット評価及びビットオフ、セマフォ解放。

```
WAIT RSV(-256)==0      /*チェックアンドセット
PRINT "abc"
dummy=RLS(-256)        /*リセット
```

RSV 参照

RM

パルス（MPG-68K互換） **4軸相対座標移動**

■書　式
RM x y u z
　x,y,u,z:相対移動量 (変数, 定数)

■解　説
4軸同時の相対パルス発生です。x y u zによって指定された値だけパルス発生します。RMもGO同様RMOVに比べてスピードが遅くなっています。座標値は移動した分だけ加算されます。

GO 参照

RMOV

パルス（MPG-68K互換） **XYU軸相対座標移動**

■書　式
RMOV x y u
　x,y,u:相対移動量 (変数, 定数)

■解　説
3軸同時の相対パルス発生です。x y uで指定された値だけパルス発生します。座標値は移動した分だけ加算されます。

MOVE 参照

RMVC

パルス（MPG-314専用） **無限パルス発生**

■書　式
RMVC n m [n1,m1,n2,m2]
　n:軸指定予約定数
　m:方向指定 +1または-1

■解　説
移動量を指定しない無限パルス発生です。加減速はACCEL、速度はFEED指定に従います。nは軸指定、mは+1,-1で方向指定のみ。引数を続けて与えて連続で起動することができます。通常ニアオリジンなどへ高速復帰する場合などに用います。次は、X_IN0をニアオリジンと想定して加減速付き高速移動でパルス発生する例です。

```
10 PG &H410
20 ACCEL 4000
30 HOUT 2 (X_A;NOP,1) /*レジスタWR1 IN0-E 有効
40 RMVC X_A -1
45 WAIT RR(X_A)==0
50 HOUT 0 (X_A;NOP,1) /*レジスタWR1 IN0-E 無効
```

IN0-E = ドライブ停止入力信号。HOUT 2 (X_A;NOP,1) とはX_IN0入力がlowで停止という意味だが、STOP X_A IN0_OFFと記述するほうが判りやすい。
予約定数は必ず大文字。

ACCEL,PRSET_ACCEL,FEED 参照

RMVL

パルス（MPG-314専用）　　直線補間（相対座標移動）

■書　式
RMVL x y u z
　x y u z:各軸パルス (最大3軸まで)

■解　説
直線補間パルス発生コマンドです。RMVL x y u zとしますが、4軸直線補間はハード的に不可能なので0以外の値を設定できるのは3軸までです。直線補間の場合の速度加速度等はACCEL設定に従いますが各軸の値が異なる場合は"主軸"の値を適用します。主軸はX>Y>U>Zの優先順位で選択されます。例えばYUZの組み合わせではY軸が主軸となります。(一般的なDDA演算の主軸とは意味が異なります)

```
RMVL 1000 2000 0 0        /* X軸CW1000 Y軸CW2000パルス
RMVL 0 2000 2000 200      /* YUZ軸でパルス発生
RMVL VOID 2000 2000 VOID  /* VOID指定可能(Rev-3.85j以上)
```

予約定数は必ず大文字。

RMVS,MOVS,MOVL,ACCEL,PRSET_ACCEL,FEED 参照

RMVS

パルス（MPG-314専用）　　軸独立パルス発生（相対座標移動･直線補間無し）

■書　式
単軸指定
RMVS n pls
　n:軸指定予約定数 X_A～Z_A
　pls:パルス数

複数軸指定
RMVS x y u z
　x y u z:各軸パルス数

■解　説
パルス発生コマンドです。速度、加速度等はACCEL設定に従います。入力方法には単軸指定、複数軸指定の二通りあります。RMVSは直線補間はしません。

```
RMVS X_A 1000              /* X軸CW1000パルス
RMVS Y_A -1000             /* Y軸CCW1000パルス
RMVS 1000 -2000 0 0        /* X軸CW1000 Y軸CCW2000
RMVS 1000 2000 VOID VOID   /* VOID指定可能(Rev-3.85j以上)

10  PG &H400
20  ACCEL X_A 5000 500 100
30  FEED X_A 0
35  INSET_314 X_A ALM_ON  /*アラームonで有効
40  DO
50    RMVS X_A 10000      /*すぐ次の行へ
60    WAIT RR(X_A)==0     /*パルス終了待ち
70    TIME 1000
80    RMVS X_A -1000
90    WAIT RR(X_A)==0
100   TIME 100
110 LOOP
```

予約定数は必ず大文字。

RMVL,MOVS,MOVL,ACCEL,PRSET_ACCEL,FEED 参照

RMVT

パルス（MPG-314専用） **連続補間（相対座標移動）**

■書　式

直線補間

RMVT n pls1 pls2

n:軸指定予約定数。2軸の論理和

pls1,pls2:移動先の相対座標

注）pls1,pls2に0もしくは異常に小さな値を使用しないで下さい。ICの特性により異常動作する事があります。

円弧補間

RMVT n pls1 pls2 <CW,CCW> cnt1 cnt2

n:軸指定予約定数。2軸の論理和

pls1,pls2:移動先の相対座標

CW,CCW:円弧補間方向指定定数

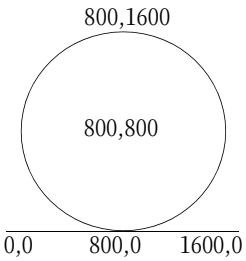
cnt1,cnt2:円中心

■解　説

連続移動補間コマンドです。軌跡制御に使用します。

入力フォーマットはRMVT n pls1 pls2もしくはRMVT n pls1 pls2 <CW,CCW> cnt1 cnt2です。nは軸定数で二つの軸定数の和を使用します。pls1,pls2は行き先の相対座標です。CWもしくはCCW定数に続いて円中心を相対位置で与えると目的位置まで円弧移動します。連続補間の途中で軸定数を変更することはできません。

```
10 PG &H400
15 CLRPOS
20 ACCEL 4000
25 HOUT X_A;DS_DACL
30   Aset=X_A|Y_A
40 RMVT Aset 800 0
50 RMVT Aset 0 1600 CCW 0 800
60 RMVT Aset 0 -1600 CCW 0 -800
70 RMVT Aset 800 0
80 HOUT X_A;EN_DACL
```



RMVTコマンドはパルス発生中に次のコマンドを設定することができるので、コマンドの切れ目でパルス出力が途絶えることがありません。25 のステートメントは通常のパルス発生で自動減速を有効としているの

を解除します。これを略すと最初のコマンドの到達時に減速し、残りのパルス発生が全て最低速となります。逆に80は減速機能を有効にして最終位置到達時に減速停止させるためのものです。
予約定数は必ず大文字。

MOVT,ACCEL,PRSET ACCEL,FEED 参照

RMVZ

パルス（MPG-68K互換） **Z軸相対座標移動**

■書　式
RMVZ z
　z:相対移動量 (変数, 定数)

■解　説
Z軸の相対パルス発生です。zで指定された値だけパルス発生します。座標値は移動した分だけ加算されます。

MOVZ 参照

ROM

デバッグ **フラッシュ ROMモード**

■書　式
ROM

■解　説
フラッシュ ROM書き込みを有効にします。無効にするにはRAMコマンドです。

RAM 参照

RR

パルス（MPG-314専用） **レジスタ読み取り**

■書　式
RR0を読み取る
RR(0)

パルス終了確認
RR(n)
　n:軸指定予約定数。指定軸がパルス終了すると0を返す

ボードを指定してRR0を読み取る
RR(adrs)
　adrs:MPG-314ボードアドレス

軸とレジスタを指定して読み取る(RR0以外の読み取り)
RR(n,a)
　n:軸指定予約定数
　a:レジスタ番号。オフセットアドレスを加えることによりMPGのアドレス指定が可能

■解　説
MCX314のレジスタを読み取ることで、パルス発生状態やエラー内容を知ることができます。

```
WAIT RR(X_A)=0          /* X軸パルス発生終了待ち
A=RR(&H410)&&HF0        /* アドレス&H410のMPG-314のｴﾗｰｽﾃｰﾀｽを読み取る。
A=RR(Y_A,1)             /* Y軸のRR1を読み取る。(終了ステータスの参照)
A=RR(Y_A,1+&H410        /* &H410のY軸のRR1を読み取る。(終了ステータスの参照)
```

ライトレジスタ(WRn)は読めません。
RR()関数に状態定数 (EMG@,ALM@,LMTP@,LMTM@,ALLl@,SLMP@,SLMM@) や停止条件定数 (EMG_,ALM_,LMTP_,LMTM_,IN3_,IN2_,IN1_,IN0_,ALLE_) をセットすることにより各入力状態の検出、またパルス発生が停止した原因を特定することができます。@の付いた定数グループは、状態検出です。たとえばALM入力は論理設定で、ON/OFFいずれかの入力でアラーム受付となりますが、この条件が成立している時にRR(X_A,ALM@)で検出することができます。これに対してALM_はパルス発生がアラーム入力によって停止したかどうかを判断するためのもので、RR(X_A,ALM_) が 0 であればアラーム入力で停止していないということになります。RR(X_A,ALM@¦LMTP@)のように状態検出は論理和をとって検出することができますが、状態入力グループと停止条件グループを混在させることはできません。

予約定数	状態監視グループ	EMG@	EMGがｲﾈｰﾌﾞﾙ検出
		ALM@	サーボアラーム出力ON検出
		LMTP@	ﾘﾐｯﾄ入力(+)検出
		LMTM@	ﾘﾐｯﾄ入力(-)検出
		SLMP@	ｿﾌﾄﾘﾐｯﾄ(+)検出
		SLMM@	ｿﾌﾄﾘﾐｯﾄ(-)検出
	停止条件グループ	EMG_	入力によって停止
		ALM_	アラーム入力によって停止
		LMTP_	ﾘﾐｯﾄ入力によって停止
		LMTM_	ﾘﾐｯﾄ入力によって停止
		IN3_	STOP条件のIN3によって停止
		IN2_	STOP条件のIN2によって停止
		IN1_	STOP条件のIN1によって停止
		IN0_	STOP条件のIN0によって停止

ﾘﾐｯﾄの検出

```
#INSET_314 X_A LMT_ON           /*リミットオンで検出
#PRX RR(X_A,LMTP@|LMTM@)        /*コネクタJ2 2,3番ﾋﾟﾝ入力
 000C                           /*両方on
```

表示されるビットとコネクタピン番号は整合しません。

RS

RS-232　　**バッファ表示**

■書　式
RS n
 n:チャンネル番号
  n=0または2

■解　説
対応するチャンネルナンバーのRS-232のリングバッファの内容を参照します。入出力とも表示されるので実際にどのようなキャラクタが通信されたのかをモニタすることができます。MPC-684のCH2, CH0のバッファは入出力とも256byteです。

RSE

RS-232 **RS-232エラー**

■書　式
RSE(n)
 n:チャンネルナンバー
  n=0または2

■解　説
RS-232受信時のエラーを返す関数です。外部機器と通信を行う時に通信状況のチェックが行えます。エラーが発生した場合の初期化はCNFGコマンドで行います。

戻り値　0　正常終了
　　1　パリティーエラー
　　2　オーバーラン
　　4　フレーミングエラー
　　8　ブレークコード検出

```
*LOOP
  CNFG#2 "9600b7pns1NONE"
*LOOP1
  INPUT#2 A$
  SELECLT_CASE RSE(2)
    CASE 0 : PRINT A$
    GOTO *LOOP1
    CASE 1 : PRINT "ﾊﾟﾘﾃｨｰｴﾗｰ"
    CASE 2 : PRINT "ｵｰﾊﾞｰﾗﾝ"
    CASE 4 : PRINT "ﾌﾚｰﾐﾝｸﾞｴﾗｰ"
    CASE 8 : PRINT "ﾌﾞﾚｰｸｺｰﾄﾞ"
    CASE_ELSE : PRINT "ｴ!? ﾅﾆ?"
  END_SELECT
  PRINT#2 "TRY AGAIN!\n"
GOTO *LOOP
```

MRS-402はサポート外。常に255です。

RSE

RS-232 **CH1キャラクタ入力**

■書　式
RSE(-1)

■解　説
CH1(プログラムポート)から1キャラクタ入力します。

```
getchar=RSE(1)        /* サポート REV-3.70c以降
```

RSV

タスク操作 **セマフォ獲得**

■書　式
RLS(n)　I/Oのビット評価及びビットオフ（セマフォ解放）
RSV(n)　I/Oのビット評価及びビットオン（セマフォ獲得）
 n:メモリーI/Oナンバー
  -8192≦n≦-1

■解　説
これらはマルチタスク下でのセマフォの概念に対応するものです。ADVFSCはセマフォとして、実在のI/Oもしくは内部のメモリ上のデータ（メモリI/O）を使用できます。メモリI/Oはnを負の値としたものです。セマフォとは、複数のタスクが1つの出力ポートなどを共通に使用する場合にその使用状況が混乱しないようにするためのものです。例えば、1つのRS-232ポートに対して2つのタスクがデータの出力をする場合キャラクタが交互にだされたりしてメッセージが意味をなさなくなることがあります。例えば次のプログラムを実行すると出力されるメッセージはかなり期待外れのものです。

```
                time0=timer
                time=timer
        FORK 1 *aho
        DO
                WAIT time<>timer
                PRINT "abcdefg" "ABCDEFG"
                        time=taimr
        LOOP
*aho
        DO
                WAIT time0<>timer
                PRINT "123456" "78901234"
                        time0=taimr
        LOOP
#run
123456789abcdefgABC01234
DEFGH
123456789abcdefgABC01234
DEFGH
  [25]  ・・・・・プログラムブレーク
#
```

これに対して次のプログラムは期待通りに動作します。これは、メモリI/O-1をセマフォとすることによって交通整理がなされたためです。なぜ、通常のON/OFF/SW(n)の組み合わせを使用しないかというと、このセマフォの獲得でデッドロックを防ぐには、テスト＆セットという一括動作が必要なためです。
RSV(n)はこのためのものです。
【RSV】

```
                time0=timer
                time=timer
        FORK 1 *aho
        DO
                WAIT time<>timer
                WAIT RSV(-1)==0
                PRINT "abcdefg" "ABCDEFGH"
                        dummy=RLS(-1)
                        time=taimr
        LOOP
*aho
        DO
                WAIT time0<>timer
                WAIT RSV(-1)==0
                PRINT "123456" "78901234"
                        dummy=RLS(-1)
                        time0=taimr
        LOOP
```

タスク間のセマフォの受け渡しについて

```
    TASK A
WAIT RSV(-1)==0
       |
       |
dummy=RLS(-1)
WAIT SW(-1)==1
WAIT RSV(-1)==0
```

```
  TASK B
    |
    |
    |
WAIT RSV(-1)==0
    |
dummy=RLS(-1)
```

TASK Aがセマフォを解放して、必ず次にTASK Bがセマフォを獲得する、というときはTASK AはSW()でメモリーI/Oを読みTASK Bの獲得を確認する。

```
WAIT RSV(-1)==0
      |
      |
dummy=RLS(-1)
SWAP
WAIT RSV(-1)==0
```

```
*LOOP
    |
    |
IF RSV(-1)<>0 THEN : GOTO *LOOP : END_IF
    |
dummy=RLS(-1)
GOTO *LOOP
```

TASK Aはほかのタスクのために1時的にセマフォを解放するが、他にセマフォを獲得するタスクが無ければ再びTASK Aがセマフォを獲得したい。そのためTASK Aがセマフォを解放した状態で必ずタスクを切り替えて他のタスクを実行するようにしたい、という場合はSWAPコマンドをいれます。SWAPのかわりにTIMEを使っても有効ですが、時間的に遅くなります。

RLS 参照

RUN

デバッグ **プログラム実行**

■書　式
RUN [n m]
  n:開始点(文番号,ラベル)
  m:ブレークポイント

■解　説
プログラムの実行。mはブレークポイントです。ブレークポイントよりの再実行にはCNTを使用します。

S_MBK

MBK-SH/RS **データエリア書き込み**

■書　式
ワード(2バイト)書き込み
S_MBK outdata dtadr
  outdata: データ 0～65535
  dtadr: データアドレス 0～7899

符号付きワード(2バイト)書き込み
S_MBK outdata dtadr~Int
  outdata: データ ±32767
  dtadr: データアドレス 0～7898
  Int: 予約定数

ロング(4バイト)書き込み
S_MBK outdata dtadr~Lng
  outdata: データ 0～4294967295
  dtadr: データアドレス 0～7898
  Lng: 予約定数

■解　説
MBK-SH/RSのデータエリアにデータを書き込みます。

```
S_MBK 2 8                 /* 2をDT0008(切り替え画面番号)に書き込む=2ﾍﾟｰｼﾞを表示
S_MBK 123456789 699~Lng   /* 123456789をDT0699(下位)とDT0700(上位)にﾛﾝｸﾞ書き込み
S_MBK 1234 20             /* 1234をDT0020にワード書き込み
S_MBK -30000 21~Int       /* -30000をDT0021に符号付きワード書き込み
```

MBK,SW,ON,IN,OUT 参照

S_MBK

MBK-SH/RS **指定エリア一括書き込み**

■書　式
S_MBK outdata dtadr count
　outdata: データ
　dtadr: 先頭アドレス
　count: 書き込み個数

■解　説
範囲を指定し、まとめてデータを書き込みます。

```
#S_MBK 5 10 20          /* アドレス10から20個に5を書き込みます
#S_MBK 10               /* 確認表示
 S_MBK 5 10
 S_MBK 5 11
 (中略)
 S_MBK 5 28
 S_MBK 5 29
 S_MBK 0 30

#S_MBK 1234567890 10~Lng 20    /* ｱﾄﾞﾚｽ10からﾛﾝｸﾞﾜｰﾄﾞで20個を1234567890にします。
#S_MBK 10~Lng                  /* 確認表示
 S_MBK 1234567890 10~Lng
 S_MBK 1234567890 12~Lng
 (中略)
 S_MBK 1234567890 46~Lng
 S_MBK 1234567890 48~Lng
 S_MBK 0 50~Lng
```

S_MBK

MBK-SH/RS **データリスト表示**

■書　式
S_MBK dtadr
　dtadr: 表示開始アドレス

■解　説
MBKデータをリスト表示します。

```
#S_MBK 100              /* アドレス100からワード表示
 S_MBK 1 100
 S_MBK 2 101
 S_MBK 3 102
 S_MBK 4 103
 (中略)
 S_MBK 0 120
 S_MBK 0 121
(スペースキーで続きを表示、その他のキーで終了)

#S_MBK 100~Lng          /* アドレス100からロングワード表示
 S_MBK 131073 100~Lng
 S_MBK 262147 102~Lng
 (中略)
 S_MBK 0 140~Lng
 S_MBK 0 142~Lng
(スペースキーで続きを表示、その他のキーで終了)
```

S_MBK

MBK-SH/RS **文字列転送**

■書　式
S_MBK a$ dtadr count
 a$: 文字列
 dtadr: データアドレス 0～7899
 count: 文字数

■解　説
文字列をcount数分MBKの指定アドレスにブロック転送します。count数が文字列より大きい時は0で埋めます。

```
a$="123567890ABC"
#S_MBK a$ 102 10      /* 10文字=5ワード使用
#PRX MBK(102)
 3231                 /* &H32='2'、&H31='1'
#PRX MBK(103)
 3533
#PRX MBK(104)
 3736
#PRX MBK(105)
 3938
#PRX MBK(106)
 4130                 /* &H41='A'、&H30='0'
#PRX MBK(107)
 0000
```

CHR$ 参照

S_CMN

MPC-LNK **共有変数変更**

■書　式
S_CMN m n [+ad]
 m: 設定値(変数,定数)±32767
 n: 変数エリア 0~124
 +ad: リンクボードアドレス

■解　説
共有変数を変更するコマンドです。#1ではすべての領域を変更できますが、#1以外ではS_RNGで確保された領域のみが変更できます。設定できる値は2バイト整数で、±32767の範囲です。

```
S_CMN 1000 12
```

S_LCL

MPC-LNK **ローカル変数変更**

■書　式
S_LCL m n
 m: 設定値(変数,定数)　±32767
 n: 変数エリア　0～120

■解　説
ローカル変数を変更するコマンドです。子LNKのみで使用できるコマンドです。設定できる値は2バイト整数で±32767の範囲です。親LNKはWINレジスタ(offset=$212)に読み出したい子LNKのアドレスを指定します。

```
S_LCL LNK(0) 120      /* ローカルエリア120(word)に自己アドレス番号を入れる
```

S_RNG

MPC-LNK **子LNKの共有変数エリア獲得宣言**

■書　式
S_RNG n [+ad] m
 n,m: 獲得エリア(絶対アドレス)
  0≦n≦m≦124
  n=-1 コマンド解除、 n=-2 連続I/Oモード
 +ad: リンクボードアドレス

■解　説
このコマンドはLNK#1では使用できません。#1以外で共有変数を使用する場合は、獲得宣言であるこの命令を実行します。例えば#2で S_RNG 5 10 とすると共有変数の5～10までは#1のS_CMNが有効となります。このコマンドを解除する場合は S_RNG -1 を実行します。 S_RNG設定は2バイト(ワード)単位。

```
S_RNG 1 5              /*bank10002~10011獲得

    bit      bank  S_RNG
10000~10007 10000 ┐ 0
10008~10015 10001 ┘
10016~10023 10002 ┐ 1
10024~10031 10003 ┘
10032~10039 10004 ┐ 2
10040~10047 10005 ┘
     :        :     :
```

連続I/Oモード
S_RNG -2 とすると、OUT/IN/ON/OFF/SWの10000番台は#1、11000番台は#2のLNKへアクセスするという制約が無くなり連続になります。使用できるMPC-LNKは1枚になります。パワーオンリセットでこのモードは解除されます。

バンク(OUT/IN)	10000～10249
ビット(ON/OFF/SW)	10000～11999

サポート：REV-384m以上

S_SCN

MPC-LNK **LNKサポートアドレス変更**

■書　式
S_SCN n [+ad]
 n: サポートアドレス　1～16
 +ad: リンクボードアドレス

■解　説
このコマンドはLNK#1のみ有効です。MPC-LNKがサポートするアドレス数は標準状態で#1～#5です。これを越えてネットを構成する場合はこのコマンドを実行します。サイクルタイム(MPC-LNKの共有変数の応答時間)はこの値に10mSecを乗じた値です。標準状態で50mSecです。

```
S_SCN 10
```

SELECT_CASE <SLC>

制御文 **多値分岐**

■書　式
SELECT_CASE vari
　CASE n : 式
　CASE_ELSE : 式
END_SELECT
　vari: 変数,文字列変数
　n: 比較する変数,文字列変数
必ずCASE_ELSEも記述してください

■解　説
選択制御文（多値分岐）。IF文などの条件文では、そうである・そうでないという、2つの可能性のみの場合分けていますが、実際の制御では1つの入力でいくつもの分岐が発生することがあります。例えば、IN(n)でデジスイッチの内容を読んでその内容に従っていくつかの違った動作をする場合です。例1ではデジスイッチの分岐、例2は文字列の場合分けです。この例の中にあるCASE文の後ろの:（コロン）はマルチステートメントのデリミタとしてのものです。
例1）

```
   a=IN(0)&&HF+IN(0)&&HF0/16*10
SELECT_CASE a
   CASE 1 :GOTO *work1
   CASE 2 :GOTO *work2
   CASE 10 :GOTO *work10
   CASE_ELSE :GOTO *abort
END_SELECT
```

例2）

```
   INPUT "a$=" a$
   SELECT_CASE  a$
      CASE "123" : PRINT        ”数字だよ”
      CASE "abc" : PRINT        ”小文字だよ”
      CASE "ABC" : PRINT        ”大文字だよ”
      CASE_ELSE  : PRINT        ”良くわかんない”
      SELECT_CASE a$
         CASE "aho" :PRINT      ”阿呆みたい”
         CASE "baka" :PRINT     ”馬鹿みたい”
         CASE_ELSE  :PRINT      ”さっぱりよねー”
      END_SELECT
   END_SELECT
```

複数条件の同一処理方法
式を記述せずにCASE文を並べます。CASE後ろの""コメント文不可。

```
10       SELECT_CASE a
20         CASE 0
30         CASE 1
40         CASE 3 : PRINT "(^_^)v " a
50         CASE_ELSE  : PRINT "(-_゜)? " a
60       END_SELECT
```

SENSE_SW

I/O **入力検出で出力操作**

■書　式
SENSE_SW inp outp
　inp: 検出入力ポート
　outp: 操作出力ポート

■解　説
検出入力がonすると操作出力をon/offします。

```
SENSE_SW ?(-1)&?(0) (OFF_P,0)      /*IF ?(-1)&?(0)<>0 then off 0 オンの場合はON_P
SENSE_SW ?(-1)&?(-2) (X_A;STP_I,&H410) /*IF ?(-1)&?(-2)<>0 THEN out (X_A;STP_I),&h410
                                       /*つまり314に対して停止命令
```

アクションは括弧でくくって表記しないと正しく動作しません。また、アクションは引数の許されるだけ記述できます。ON_P,OFF_Pは予約定数です。タスク32ではSENSE_SWしか使えません。

FORK 参照

SET

パルス（MPG-68K互換）　　インチング量設定

■書　式
SET n [x y u z]
 0≦n≦3

■解　説
このコマンドではティーチモードでのインチング量を既定します。ティーチモードではX～Zのキーインによって JOG移動しますがこの移動量は0～3のキーによって選択することが出来ます。SETコマンドのnはこの0～3に対応しています。そして、それぞれの移動量を定義することが出来ます。SETは次の例のように引き数がないと0～3のJOG量を表示します。又、番号のみ指定すればその対応するJOG量だけ表示します。

```
#SET 1 5 5 5 10
#SET
 dx= 5 dy= 5 du= 5 dz= 10
#SET
 dx= 10 dy= 10 du= 10 dz= 10
 dx= 5 dy= 5 du=5 dz= 10
 dx= 100 dy= 100 du=100 dz= 100
 dx= 500 dy= 500 du=500 dz= 500
#
```

パラメーターを省略すると現在の設定値が表示されます。

SETF

コプロ演算　　データ引渡し

■書　式
SETF fn1,fn2,fn3,..
 fn1,fn2,fn3,..:変数,定数

■解　説
変数、定数値をコプロに引き渡します。与えられた引数は順にコプロの不動小数点レジスタFP0,FP1..に引き渡されます。

```
SETF 0 3 4          /* FP0に0,FP1に3,FP2に4をセットします。
```

GETF, CALF, PRF 参照

SETIO

I/O **出力初期化**

■書　式
SETIO

■解　説
出力の一括OFFに使用します。

SETP

パルス（MPG-68K互換） **点データ設定**

■書　式
SETP [n x y u z]
 n: ポイントナンバー
 x,y,u,z: 座標値（パルス）
  0≦n≦13000
パラメーターを省略すると現在の設定値が表示されます。

■解　説
点データの設定です。通常はSETP 1 100 200 300 400というように点番号と設定すべき値を与えます。FTMWでサポートされている点データの保存はこのSETPコマンドの書式でファイルが作成されます。SETPは入力書式によって次の使い分けがあります。

```
#SETP 1 1000 1000 1000 2000
#SETP 1
 X= 1000 Y= 1000 U= 1000 Z= 2000
#SETP 100
 X= 0 Y= 0 U= 0 Z= 0
#SETP 0
 X= 12 Y= 12 U= 12 Z= 12
```

最初の例はこのコマンドの目的である点データの設定です。次に点番号だけ与えればその点番号のデータを表示します。さらに引き数が何も無ければ現在位置の変更となります。又、SETPでは点データのコピーもできます。

```
SETP 3 P(5)          ･･･P(5)のﾃﾞｰﾀをP(3)にｺﾋﾟｰ
SETP 10 PL2(5)       ･･･ﾊﾟﾚｯﾄの点PL2(5)をP(10)にｺﾋﾟｰ
```

SETPOS 参照

SETP

MPG-314 **円弧補間(MOVT)の点データ設定**

■書　式
SETP n f s fc sc
 n: ポイントナンバー
 f: 第一軸座標
 s: 第ニ軸座標
 fc: 第一軸円弧中心座標
 sc: 第ニ軸円弧中心座標

MOVT 参照

SETPOS

パルス（MPG-68K互換） **現在値変更**

■書　式
SETPOS x y u z
　x,y,u,z: 座標値（パルス）
パラメーターを省略すると現在の設定値が表示されます。

■解　説
現在位置の変更です。現在位置がx y u zで指定された値になります。引き数を与えないで実行すると現在位置が表示されます。

SETP 参照

SFTL

演算 **配列変数ローテート**

■書　式
SFTL ary(n)
SFTR ary(n)
　ary: 配列名

■解　説
DINで配列宣言された配列要素に対してデータのローテートを実施します。ary(n)のaryは配列名、nは数値でいくつまでのデータをローテートするかを指定します。尚、SFTL/SFTRはx (),y(), u(), z()には有効ではありません。例えば、SFTL aho(3)では次のようになります。

```
aho(0) <- aho(1)
aho(3) -> aho(2)
```

SFTL、SFTRは閉じたシフトを行います。

```
10        DIM aho(10)
20        FOR i=0 TO 10
30            aho(i)=i+10
40        NEXT i
45        SFTL aho(3)
50        FOR i=0 TO 10
60          PRINT "i=" i "aho=" aho(i)
70        NEXT i
#
RUN
i=0 aho=11
i=1 aho=12
i=2 aho=13
i=3 aho=10
i=4 aho=14
i=5 aho=15
i=6 aho=16
i=7 aho=17
i=8 aho=18
i=9 aho=19
i=10 aho=20
```

SFTR 参照

SFTR

演算 **配列変数ローテート**

■書　式
SFTR ary(n)
　ary: 配列名

SFTL 参照

SHMZ

パルス（MPG-68K互換） **原点復帰設定**

■書　式
SHMZ pat spd
　pat: 原点復帰出力パターン
　spd: 原点復帰スピード(pps)

■解　説

MSB	5	4	3	2	1	0	LSB
方向	-	-	-	-	ZCCW	ZCW	

```
SHMZ &H2 1000 /*Z-方向 1000pps
```

SHOM,HOMZ 参照

SHOM

パルス（MPG-68K互換） **原点復帰設定**

■書　式
SHOM pat spd
　pat: 原点復帰出力パターン
　spd: 原点復帰スピード(pps)
パラメーターを省略すると現在の設定値が表示されます。

■解　説
原点復帰の動作モード設定です。SHMZはZ軸原点復帰、SHOMはXYUの3軸原点復帰モード設定です。spdはppsで表現されるスピードとなります。patは原点復帰のパルス方向を定めます。patの各ビットが原点復帰時に出力されるパルスパターンとなります。patと原点復帰出力パルスの関係は次の表の通りです。

MSB	5	4	3	2	1	0	LSB
方向	UCCW	UCW	YCCW	YCW	XCCW	XCW	

例えばXYZそれぞれCCW方向にパルスを出力して原点復帰する場合はSHOM &H2A 1000とします。1000は1kppsを表します。SHOM,SHMZは引き数無しで実行すると現在設定された値を表示します。
X,Y,U軸がCCW方向に1000ppsで原点復帰。Z軸はCCW方向に500ppsで原点復帰。

```
SHOM &H2A 1000
SHMZ &H2 500
```

※注意 SHOMはZCW,ZCCWについても有効です。SHMZを設定してもその後にSHOMを実行するとSHOMの設定内容が有効になります。SHOMとSHMZを併用する場合はSHOM、SHMZの順に記述して下さい。

```
SHMZ &H2 1000          <--Z軸の設定を行っているが...
SHOM &H2A 1000         <--このSHOMでZ軸の設定がｸﾘｱｰされるのでこの順番はダメです。
```

HOME 参照

SIN

演算 **三角関数**

■書　式
SIN A1 A2 adr(A3)

■解　説

```
A3=sin(A1/10000)*A2
```

引き数は度。

COS 参照

SLOW

RS-232 **CH1キャラクタ送信間隔**

■書　式
SLOW n
パラメーターを省略すると現在の設定値が表示されます。

■解　説
CH1のキャラクタ送出スピードの低速化。パワーオンリセット時にn=0に設定されます。

SQ

演算 **自乗**

■書　式
SQ(n)
　n: 定数，変数

■解　説
nの自乗を返します。

```
#A=4
#B=SQ(A)
#PRINT B
16
```

MPCの演算は4バイト長(±2147483647)ですから、nは46340以下です。

SQR

演算 **平方根**

■書　式
SQR(n)
　n: 定数，変数

■解　説
nの平方根を返します。

```
PRINT SQR(SQ(100))     これは100となります。
```

小数点以下は切り捨てます。

ST192

RS-232C **CH1ボーレート変更**

■書　式
ST192

■解　説
実行後MPCの電源OFF/ONで変更される。9600と19200のトグル動作。FTMW側も設定変更。
MPC-684の場合はSP7,SP8で設定。

ST_REG

MPG-3202 **X3202レジシタ書き込み**

■書　式
ST_REG reg data
　reg: X3202のアドレス(xx)とレジスタ・カウンタセレクトコード(code)の和 (&Hxx00+code)
　data: 設定データ

■解　説
MPG-3202のパルス発生IC「X3202」のレジスタ設定を行います。

```
ST_REG &H101 1000      /* X3202 #1のﾚｼﾞｽﾀ1に1000をセット
```

CMND,REG,REG3 参照

STOP

パルス（MPG-314専用） **条件停止**

■書　式
STOP n cond
　n: 軸指定予約定数 X_A～Z_A
　cond: 入力指定
　　INx_ON～INx_OFFの論理和 または STP_I,STP_D

■解　説
nを論理和で指定すれば指定軸の同時停止が可能です。condにINxを指定した場合、STOP n VOIDで解除するまで有効となります。STP_I,STP_Dをセットすると発生中のパルスを停止します。

```
STOP Z_A IN0_ON|IN3_ON  /* Z軸IN0オンIN3オンになったら停止。
                        /* 一度設定すると STOP Z_A 0実行まで解除されない。

STOP X_A|Y_A STP_D      /* XY軸同時減速停止。
```

予約定数は必ず大文字。

STOP

パルス（MPG-68K互換） **パルス発生停止**

■書　式
STOP m n
 m: 停止パターン
 1＝減速停止(ACCELコマンドで設定された加減速テーブルに従い停止します。)
 2＝即停止
 3＝パルス発生終了時にタスクをPAUSEします
 (STOP 3はREV 1.2以上でQ_PAUSEモード指定時のみ有効です。)
 n: タスク番号
  0≦n≦31
どの場合も座標値は整合します。

■解　説
マルチタスクで使用します。パルス発生中のタスクに他のタスクから停止をかけるものです。STOPでは減速停止と即停止の選択が出来ます。STOP 1で減速停止、STOP 2で即停止です。停止確認はBSYで行います。

```
FORK 1 *aho
    MOVE 100000 10000 1000
    END
*aho
    IF SW(0)==1 THEN
       STOP 1 0
    END_IF
    IF BSY(0)<>0 THEN
       END
    END_IF
    GOTO *aho
```

この例では移動直前に停止タスクを起動しています。停止タスクはパルス動作の監視も併せて行っておりパルス停止後監視タスクも停止するようになっております。ここでSTOPの引き数は減速モードの停止とタスク番号となっています。
※MOVE、MOVZ、JUMP、GOにはQ_PAUSEモードでの停止がシンプルで合理的（お勧め）です→Q_PAUSE参照

次のプログラムではSTOPコマンドを発行してBSY()関数でパルス停止を確認、タスクをQUITしています。パルス発生中のタスクをいきなりQUITしてもパルスは止まりません。パルスの停止にはこの他にDS_PGコマンドがあります。

```
10       FORK 1 *TASK1
20       WAIT SW(192)==0 AND SW(193)==0
30       WAIT SW(192)==1 OR SW(193)==1
40       IF SW(192)==1 THEN : STOP 1 1 : END_IF  <--減速停止
50       IF SW(193)==1 THEN : STOP 2 1 : END_IF  <--即停止
60       WAIT BSY(1)<>0
70       PRINT "SW(192)=" SW(192) "SW(193)=" SW(193) "BSY=" BSY(1)
80       QUIT 1
90       END
100    *TASK1
110      PG &HE0
120      ACCEL 10000
130      FEED 0
140      DO
150        MOVE 0 0 0
160        WAIT BSY(-1)==1  <--正常停止でない時ﾌﾟﾛｸﾞﾗﾑは先へ進まない
170        MOVE 50000 0 0
180        WAIT BSY(-1)==1
190      LOOP
#RUN
SW(192)=1 SW(193)=0 BSY=256
#RUN
SW(192)=0 SW(193)=1 BSY=512
```

STOPコマンドによるパルス停止例

```
10       DO
20         FORK 1 *TASK1
30         WAIT SW(192)==0
40         WAIT SW(192)==1
50         PRINT "STOP"
60         STOP 1 1          <--STOPして
70         WAIT BSY(1)<>0    <--停止を確認して
80         PRINT BSY(1)
90         QUIT 1            <--QUITする
100        WAIT SW(193)==1
110      LOOP
120    *TASK1
130      DO
140        MOVE 100000 0 0
145        WAIT BSY(-1)==1   <--正常停止以外はここで停止する。この様な連続した
150        MOVE 0 0 0            LOOPの場合､もしここにWAIT BSY()がないとTASK 0か
155        WAIT BSY(-1)==1       らSTOPされてMOVEｺﾏﾝﾄﾞから抜けでても、すぐに次の
160        LOOP                  MOVEｺﾏﾝﾄﾞを実行してしまいﾊﾟﾙｽが止まらない。
```

STOPコマンドで停止してBSYで確認しない場合（正常に動作しない）

```
10       DO
20         FORK 1 *TASK1
30         WAIT SW(192)==0
40         WAIT SW(192)==1
50         PRINT "STOP"
60         STOP 1 1
90         QUIT 1            <--STOPのあといきなりQUITしているのでﾊﾟﾙｽが止まら
100        WAIT SW(193)==1       なくなる
110      LOOP
120    *TASK1
130      DO
140        MOVE 100000 0 0   <--MPGはﾊﾟﾙｽを出し切るまで止まらない
145        WAIT BSY(-1)==1       そしてMPGを管理していたﾀｽｸが止まってしまったの
150        MOVE 0 0 0            で8251ｴﾗｰとなる
155        WAIT BSY(-1)==1
160      LOOP
#RUN
STOP
```

タスク停止 *1 [140]MPG通信エラー8251側、STOPしないでQUITしてませんか？

↑ﾀｽｸ1を再FORKしてもMPGがｴﾗｰをおこしてしるので動かない。

DS_PGによるパルス発生の停止

```
10       DO
15         EN_PG 1
20         FORK 1 *TASK1
30         WAIT SW(192)==0
40         WAIT SW(192)==1
50         PRINT "STOP"
60         DS_PG 1           <--ﾊﾟﾙｽ発生禁止
70         WAIT BSY(1)<>0
80         PRINT BSY(1)
90         QUIT 1
100        WAIT SW(193)==1
110      LOOP
120    *TASK1
130      DO
140        MOVE 100000 0 0   <--MOVE実行中にDS_PGされるとこのﾀｽｸはMOVEｺﾏﾝﾄﾞ中で
145        TIME 1000             ﾊﾝｸﾞｱｯﾌﾟする。
150        MOVE 0 0 0            ﾊﾟﾙｽ発生以外のｺﾏﾝﾄﾞ実行中にDS_PGされるとこのﾀｽｸ
155        TIME 1000             は動くがﾊﾟﾙｽは出ない。
160      LOOP
```

```
#run
STOP
1024
STOP
1024
STOP
1024
```

次のプログラムはタスク1でパルス発生中にタスク1を再FORKしています。
このプログラムはRMOVですがこのほかMOVE、JUMP、GO、ACCELなどパルス発生関係コマンドでも同じです。

```
LIST 0
10        PG &HE0 1
20        DO
30          FORK 1 *TASK1
40          TIME 1000
50        LOOP
60      *TASK1
70        DO
80          RMOV 100000 100000 1000000
90        LOOP
#RUN
    タスク停止 *1 [80]MPG通信エラーです68301側
    タスク停止 *1 [80]MPG通信エラーです68301側
    *0   [40]
    *1   [80]
```

次のプログラムはパルス発生中のタスクをSTOPせずにQUITしています。
STOPせずにQUITするとパルスは最後まで出ます（この場合はX,Y,Uとも1000パルス）。しかしMPCとMPGの内部の通信が途切れてしまうので再びFORKしてパルスを発生しようとしてもエラーになります。

```
LIST 0
10        PG &HE0 1
20        DO
30          FORK 1 *TASK1
40          TIME 50
50          QUIT 1
60          TIME 2000
70        LOOP
80        END
90      *TASK1
100       DO
110         RMOV 1000 1000 1000
120       LOOP
#RUN
    タスク停止 *1 [110]MPG通信エラー8251側、STOPしないでQUITしてませんか？
    *0   [60]
    *1  停止
```

BSY,Q PAUSE 参照

STPS

パルス（MPG-314専用） **現在位置指定**

■書　式
STPS　n pos
　n: 軸指定予約定数 X_A～Z_A,ALL_A
　pos: 指定する位置

STPS x y u z
　x y u z: 各軸の指定する位置。指定無=VOID

■解　説
前者は選択した軸のみ現在値を設定します。後者はVOID定数以外の軸の現在位置を設定します。パルス発生中の軸に値を設定しようとすると停止待ちとなります。

```
STPS U_A 1000                 /* U軸の現在位置を1000にする
STPS VOID 1000 VOID 2000      /* YZ軸の現在位置を設定
STPS 0 0 0 0                  /* 全軸0設定
```

STPS

パルス（MPG-314専用） **カウンタ設定**

■書　式
STPS n c
　n: 軸選択予約定数 X_C～Z_C
　c: カウンタ初期値

■解　説
カウンタ値は X(-1)、CP -1などで読み取れます。カウンタを使用するにはMPG-314のOC5～OC8ソケットにTLP2630を実装します。

X 参照

STR$

文字列 **数値から文字列に変換**

■書　式
STR$(n)

■解　説
nの数字列を与えます。

```
#a=1000
#a$="BAKA"+str$(a)+"AHO"
#print a$
BAKA1000AHO
```

VAL,HEX$ 参照

STRCPY

文字列 **文字列の複写**

■書　式

STRCPY src$ dest$ [begin len]

  src$: コピー元文字列変数

  dest$: コピー先文字列変数

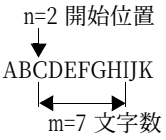
  begin: コピー開始位置(変数,定数)

  len: コピー文字数(変数,定数)

■解　説

src$のbegin文字目からlen文字をdest$にコピーします。beginは0文字目から数えます。

```
STRCPY A$ B$ n m       A$のn文字目からm文字をB$にコピー
```



n及びmはそれぞれ省略することが出来ます。nもmも省略された場合には、s1$よりs2$へすべての文字がコピーされます。nのみ与えたときはn文字目からすべてとなります。次の3つはすべて同じ働きをします。

```
B$=A$
STRCPY A$ B$
STRCPY A$ B$ 0
```

mを省略せずに指定するとコピーされる文字の数はmによって定められます。次の例では文字の長さを調べる関数、LEN()を使用して、いろいろなパターンで文字のコピーを実施しています。

例1)

```
10          a$="01234567890"
20        PRINT LEN(a$)
30        FOR i=0 TO LEN(a$)
40          STRCPY a$ b$ i
50          PRINT b$
60        NEXT i
RUN
11
01234567890
1234567890
234567890
34567890
4567890
567890
67890
7890
890
90
0
```

例2)

```
10          a$=time$            /*time$は予約変数です。
20        STRCPY a$ h$ 0 2
30        STRCPY a$ m$ 3 2
40        STRCPY a$ s$ 6 2
50        PRINT h$ "時" m$ "分" s$ "秒"
#
RUN
02時36分36秒
```

SV_M

MBK-SH/RS **メモリー括コピー (MBK→MPC)**

■書　式
SV_M array mbktop count [opt]
  array: MPC点データの先頭(XYUZ(1～13000))、配列変数の先頭
  mbktop: MBKメモリ先頭アドレス。19≦mbktop
  count: コピー数
  opt: 4=ロングワードで1アドレスおき、Lng=ロングワードで連続

■解　説
配列からMBKのデータエリアにコピーします。パラメータはアドレスの使用範囲を超えぬように設定して下さい。

```
SV_M X(9019) 19 683       /* DT19から点データX(9019)へ683個ワードコピー
SV_M test(10) 500 10      /* DT500から配列変数test(10)へ10個ワードコピー
SV_M X(9019) 19 342 4     /* DT19から点データX(9019)へ342個ロングワードコピー
SV_M X(9000) 500 5 Lng    /* DT500から点データX(9000)へ5個ロングコピー (連続)
SV_M test(0) 500 5 Lng    /* DT500から配列変数test(0)へ5個ロングコピー (連続)
```

LD_M 参照

SW

I/O **ビット入力**

■書　式
SW(n)
  n: ポートナンバー

■解　説
I/Oポートのビット入力。I/Oを2度読みし結果が一致するまで読み続けます。これは、チャタやノイズによる誤読み取りを防ぐためのものです。返す値は1がON状態、OFFで0となります。nに出力ポート番号を与え出力の状態を得ることもできます。物理入力はSWAPを含む2度読み、メモリI/O,MBK入力は1度読み。

IN,HSW 参照

SWAP

タスク操作 **実行権の放棄**

■書　式
SWAP

■解　説
タスクスワップ。他のタスクに実行権を譲ります。このコマンドはプログラム実行の高速化に必要です。ADVFSCのマルチタスクはタイムシェアリングによるものですが、これはプログラムの実行効率を著しく低下させます。マルチタスクでタスクを5個実行すれば、1つのタスクに割り当てられる時間は1/5となります。つまりプログラムの実行速度は1/5となってしまいます。このため、ADVFSCではタイマー待ちの状態では不要なタスクをスリープさせたり WAIT などのような条件待ちでは実行中のタスクを強制的に切り換えてプログラムの実行効率を高めています。つまり、プログラムの実行中にあまり速度を要求されない仕事やレベルの低い仕事をさせないようにしているわけです。

```
*WAIT
        IF A=0 THEN
           GOTO *WAIT
        END_IF
```

この例では、変数Aの状態が1となるのを待っています。こうした仕事をポーリングと言いますが、これに他の仕事と同じ時間を与えるのは効率上好ましくないことです。なぜならAを1にセットするのは他のタスクですから、条件不成立の場合にこれ以上このタスクを実行することは意味がありません。こうした場合にSWAPを用います。

```
*WAIT
        IF A=0 THEN
           SWAP
           GOTO *WAIT
        END_IF
```

これでポーリングして条件不成立ごとに、他のタスクに実行権が引き渡され、無駄なくAが1になることを待つことが出来ます。なお、これと等価のコマンドはWAITです。

```
        WAIT  A=1
```

WAITには条件不成立の場合のSWAPが組み込まれています。タイマー及びタイマーの組み込まれている関数などではSWAPと等価のことが起こっています。(MPC-68K システムタイマは 20msec)

SWP

演算 **上下位バイト交換**

■書　式
SWP(n)

■解　説
バイトスワップ。上位バイトと下位バイトを入れ換えます。684と8086では上位下位のアドレス順位が反対です。

```
#prx swp (&hff00)
 00FF
#prx swp (&h00ff)
 FF00
#
```

SYSCLK

タイマー **システムクロック**

■書　式
SYSCLK　(大文字で記述)

■解　説
パワーオンより5msec毎に+1される変数です。次の例はFOR～NEXT 10000回の所要時間を計測します。

```
5          SYSCLK=0
10       FOR I=1 TO 10000
30       NEXT I
40       PRINT SYSCLK
RUN
94                          ←94×5=470msec
```

※注意
SYSCLKはパワーオン後124日を経過しますと、負の値をとるようになりTMOUTの計算が正常にできなくなります。124日以上継続稼動する可能性がある場合、SYSCLKをクリアする操作をプログラム中に追加してください。

```
SYSCLK=0                クリア例
```

T

MPG-68K互換 **ティーチングモード**

■書　式
T

■解　説
TEACH （省略形 T）

TEACH 参照

TAIL

編集 **文番号の最大値**

■書　式
TAIL

■解　説
文番号の最大値を表示します。プログラムコーディング中の後ろにプログラムをつけ加える場合に使用します。

```
LIST 0
10   FOR i=0 TO 47
20     ON i : TIME 50 : OFF i
30   NEXT i
#tail
30
#
```

TAN

演算 **三角関数**

■書　式
TAN A1 A2 adr(A3)

■解　説

```
A3=tan(A1/10000)*A2
```

引き数は度。

COS 参照

TASK

デバッグ **タスク状態表示**

■書　式
TASK(n)
n: タスク番号
0≦n≦31

■解　説
nはタスク番号で、指定されたタスクの状態を知らせる関数です。この値が0であればそのタスクは実行中であることになります。0以外の正の数であればそのタスクは休止中であることを意味します。この数の意味はタイマー値です。1000であればあと1秒は休止していることになります。TASK(n)の値の意味は次の通りです。

戻り値	0	実行中

-1　未使用
-2　QUIT状態
-3　PAUSE状態

END 終了していると -1。TIME 実行中は残タイマ値。WAIT SW(0)==1 の場合は 0 または 5。

TASKN

タスク操作　　**タスク番号取得**

■書　式
TASKN　(大文字で記述)

■解　説
タスクナンバーを知る変数です。

```
10        FORK 13 *JOB
20        DO  : LOOP
30      *JOB
40        PRINT "I am " TASKN
50        END
RUN
I am 13
```

TEACH

パルス（MPG-68K互換）　　**ティーチングモード**

■書　式
TEACH　（省略形 T）

■解　説
TEACHコマンドはTで実行できます。ティーチモードでは点データの教示、及び調整時に必要なコマンドが1文字キーインで操作できるようになっています。

x	XCW方向にインチング移動
X	XCCW方向にインチング移動
y	YCW方向にインチング移動
Y	YCCW方向にインチング移動
u	UCW方向にインチング移動
U	UCCW方向にインチング移動
z	ZCW方向にインチング移動
Z	ZCCW方向にインチング移動
0,1,2,3	インチング量の選択0～3(SETコマンド参照)
P,p	点データの指定
L,l	LIMZ設定(JUMP,JMPZ参照)
O,o	出力ポートのON
F,f	出力ポートのOFF
S,s	I/Oの値表示
H,h	原点復帰入力ポートの表示(HPT(0)参照)
J,j	指定点へJUMP移動
A,a	指定点へJMPZ移動
R,r	RANGE設定(RANGEコマンド参照)
TAB	MPG切り替え(タスク番号インクリメント)
+	MPG切り替え(タスク番号インクリメント)
-	MPG切り替え(タスク番号デクリメント)
Q,q	ティーチモードの終了

THEN

制御文 **条件分岐**

■書　式
IF　条件式　THEN
 (制御文)
 ELSE
 (制御文)
END_IF

■解　説

```
IF SW(a)==1 AND SW(b)==1 THEN
  ON 0 : OFF 1
  ELSE
  OFF 0 : ON 1
END_IF
```

IF 参照

TIME

タイマー **時間待ち**

■書　式
TIME t
 t: 待ち時間 (msec)

■解　説
時間待ちです。単位はmsecで指定しますが実態は5msec単位です。TIME 3はTIME 0と同じ意味になります。

```
TIME 1000              1秒停止、この間タスクはスリープです。
```

time$

予約変数 **時刻文字列取得**

■書　式
time$　(小文字で記述)

■解　説
MBK-SHまたはMBK-RSとデジタルGPが接続されていると、GPの時刻が文字列として入ります。小文字で書いて下さい。(MPC-684 3.81x以上)
GP時計から秒の取得はできません。

```
#PRINT time$
11:31:00               /* 秒は常時"00"です。
```

timer

予約変数 **時間取得**

■書　式
timer　(小文字で記述)

■解　説
時間を全て秒になおした数です。00:01:00の場合timerは60となります。

```
#PR time$
00:01:08
#PR timer
68
```

time$,CLK 参照

TMON

メンテナンス **タスクモニタ**

■書　式
TMON

■解　説
各タスクのスタックとプログラムカウンタの値を表示します。使用されていないタスクではFFFFFFFFが表示されます。

TMOUT

タイマー **入力時間設定**

■書　式
TMOUT n
 n: タイマー値（sec）
　0=TMOUT無効

■解　説
TMOUTは関数WS0(),WS1()のタイムアウトの設定です。WS0,WS1は指定されたポートがONもしくはOFFになる迄ポーリングする条件待関数です。ポーリング最大時間をTMOUTによって定めることができ、返される値はポートの状態ではなく、時間内に条件が整ったかどうかです。時間内に条件が充たされれば0、そうでなければ1を返します。nの単位は秒です。TMOUTの時間は全タスク中最後に実行されたTMOUTが有効になります。全てのタスクに共通です。
[実験]　WS0(),WS1()実行中に裏のタスクでTMOUT値を変更したらどうなるか。

```
10    FORK 1 *TASK1
20      SYSCLK=0
30    TMOUT 5
40    PRINT "start " SYSCLK
50    IF WS1(192)==1 THEN : GOTO *TMOUT0 : END_IF
60    END
70  *TMOUT0
80    PRINT "task0 tmout " SYSCLK
90    END
100 *TASK1
110   TIME 2000
120   TMOUT 1         /* ※ここでTMOUT値を変える
130   PRINT "tmout change " SYSCLK
140   END
#RUN                  /* ※が TMOUT 1 (元より小)のとき
start 1
tmout change 402
task0 tmout 1004      /* 1004*5mSec  5Sec
#RUN                  /* ※が TMOUT 10 (元より大)のとき
start 1
tmout change 402
task0 tmout 1004      /* 1004*5mSec  5Sec
```

[結果]　裏タスクでTMOUTを変えても実行中のWS0(),WS1()には影響しません。(確認 REV-3.85c 040303)
WAITのタイムアウトは、WAIT 参照

TMOUT

タイマー **入力時間設定（WAIT文）**

■書　式
WAIT 条件式 TMOUT 時間：タイムアウト時の処理
　時間: msec(1000=1秒) max327670(327秒　5分)

■解　説
REV-3.84jからWAIT文にタイムアウト機能が付加されました。次の例のように条件式のあとにTMOUTと時間を指定してマルチステートメント部分にタイムアウト処理を書きます。TMOUTがなければWAIT文の通常仕様になります。

※注意
TMOUT時間は開始時のSYSCLK値に対する相対値です。TMOUT待機中に裏タスクでSYSCLKを操作すると正常に実行できません。

次のプログラムで、20行はTMOUT時のみマルチステートメントを実行します。

```
10     SYSCLK=0
20   WAIT SW(192)==1 TMOUT 5000: PRINT "A": GOTO *TMOUT
30   PRINT "B"
40   PRINT "IN TIME " SYSCLK
50   END
60 *TMOUT
70   PRINT "TMOUT " SYSCLK
#RUN
B
IN TIME 325           /* 5秒以内にonになった
#RUN
A                     /* 5秒以内にonにならなかった
TMOUT 1002
```

もしも処理文が無ければ、
例1)

```
10     SYSCLK=0
20   WAIT SW(192)==1 TMOUT 5000
30   PRINT "B"
40   PRINT "IN TIME " SYSCLK
50   END
60  *TMOUT
70   PRINT "TMOUT " SYSCLK
#RUN
B                     /* 5秒以内にonになったら次の行へ
IN TIME 294
#RUN
B                     /* onにならなくても5秒たてば次の行へ
IN TIME 1003
```

例2)

```
10    SYSCLK=0
20    A=0
30  WAIT SW(192)==1 TMOUT 5000 :   A=1
40  PRINT A SYSCLK
50  END
#RUN
0 316                 /* 5秒以内
#RUN
1 1001                /* タイムアウトの時A=1
```

REV-3.84q でRUNタイムエラーとなるようにしました。

```
10 WAIT SW(192)==1 TMOUT 1000   /* O(オー )とO(ゼロ)間違い
#run
#10                   ・・・TMOUT条件がまちがっています
```

TOFF

デバッグ **トレースモード**

■書　式
TOFF

■解　説
プログラム実行トレースモード解除。

TON 参照

TON

デバッグ **トレースモード**

■書　式
TON

■解　説
トレースモードでは実行中のプログラムの文番号を画面に表示します。タスク0のみ有効です。

```
10     FOR i=0 TO 47
20       ON i : TIME 50 : OFF i
30     NEXT i
#ton
#run
[10]
[20]
[20]
[20]
[30]
```

TOFF 参照

U

パルス（MPG-314専用） **カウンタ値読み込み**

■書　式
U(-1)

X,STPS 参照

U

MPG-314 **カウンタ値読込み&クリア**

■書　式
U(-2,A)

X,STPS 参照

U

MPG-314 **カウンタ値読込み&クリア（PG指定）**

■書　式
U(pgadrs,A)
　pgadrs: MPG-314ボードアドレス　(&H400～&H490)

X,STPS 参照

U

パルス（MPG-68K互換） **U軸点データ**

■書　式
U(n)
　n: ポイントナンバー

X 参照

UNTIL

制御文 **条件文**

■書　式
DO～LOOP UNTIL 条件式
MOVE x y u UNTIL 条件式
RMOV x y u UNTIL 条件式
PULSE axis acnt [t1 t2] UNTIL 条件式

■解　説
DO～LOOPやパルス発生コマンドの繰り返し条件、パルス停止条件を与えることができます。条件式には普通のI/O、メモリーI/O、変数が使えます。

UNTIL使用上の注意
DO～LOOPの場合は論理結合ができます。

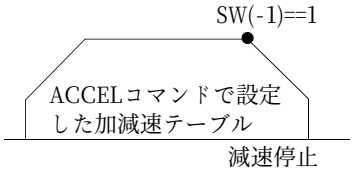
```
10   DO
20     ON 0
30     TIME 100
40     OFF 0
50     TIME 100
60     LOOP UNTIL A==1 AND B==1
70   END
```

パルス発生コマンドの停止条件では論理結合が出来ません。2つ以上の条件を結合したいときは別のタスクで結合します。また一致の条件を入力する場合は必ず”==”と入力して下さい。DO～LOOPの場合は”=”と入力してもMPCが自動的に”==”と変更しますがパルス発生コマンドの場合は行われません。

```
10       OFF -1
20       FORK 1 *TASK1
30       TIME 2000
40         IF SW(192)==1 AND SW(193)==1 THEN : ON -1 : END_IF
50       END
60     *TASK1
70       DO
80         CLRPOS
90         MOVE 10000 10000 10000 UNTIL SW(-1)==1    <--ｲｺｰﾙは必ず2つ入力
100        IF SW(-1)==1 THEN : GOTO *END : END_IF
110        MOVE 0 0 0 UNTIL SW(-1)==1
120        IF SW(-1)==1 THEN : GOTO *END : END_IF
130      LOOP
```

```
140     *END
150       PRINT "ｵﾜﾘ"
#
```

UNTILでのパルス停止は減速停止になります。



UNTIL使用例

```
10          B$="" :   A$=""        <--文字列初期化
20        DO UNTIL A$=="*"         <--"*"が来るとDO LOOPを終了
30            B$=B$+A$             <--文字列の合成
40            A$=INPUT$(1)         <--RS-232  CH1から1文字入力
50        LOOP
60        PRINT B$
#RUN
ABCD                               <-ﾀｰﾐﾅﾙから"ABCD*"と入力した後
#
```

_VAR

制御文 **引数からの受け取り**

■書　式
_VAR var1 var2 - -
　var: 変数

■解　説
GOSUBの引数を受け取ります。

GOSUB 参照

VAL

文字列 **数字文字列から数値**

■書　式
VAL(a$)
　a$: 数字文字列（ﾍｷｻ表記も可能）

■解　説
数字文字列a$を数値に変換します。

```
#a$="1000"
#a=VAL(a$)+2000
#print a
3000
```

ヘキサ表記文字列の変換例

```
#a$="12BC"
#a$="&H"+a$
#a=val(a$)
#pr a
4796
```

STR$,HEX$ 参照

VER

メンテナンス **改版データの表示**

■書　式
VER

■解　説
搭載されているROMの版数を表示します。ターミナルソフトのオープニングにも表示されます。

```
MPC-684f ADVFSC(r) REV-3.85n     /*REV-x.xxx が版数
 BASIC like + multi tasking
 Created by ACCEL Crp.~2004
```

HISTORY 参照

VER$

予約変数 **改版データの取得**

■書　式
VER$　(大文字で記述)

■解　説
バージョンデータが入っている予約変数です。

```
#a$=VER$
#pr a$
MPC-684f ADVFSC(r) REV-3.85n
 BASIC like + multi tasking
 Created by ACCEL Crp.~2004
```

GET_VAL 参照

VLIST

編集 **プログラムリファレンスの表示**

■書　式
VLIST

■解　説
プログラムで使用しているラベル、変数などの情報を表示します。

WAIT

制御文 **条件待ち**

■書　式
WAIT 条件式

■解　説
条件待ち。マルチタスク下での効率のよいポーリングを実行します。タスク処理の効率についてはSWAPを参照して下さい。

```
① WAIT P_SW(i01)<>1
② WAIT A==1
③ time=timer
   WAIT timer<>time
```

①の例はi01で指定された入力ポートが0になるのを待ち、②の例ではAが1になるのを待ちます。③の例ではRTCを使用した1秒タイマーとなっています。

WAIT

制御文 **タイムアウト付き条件待ち**

■書　式
WAIT 条件式 TMOUT 時間(msec)
サポート REV-3.84j以降

■解　説
WAIT文にタイムアウト機能が付加。例のように、条件式の後にTMOUTと時間を指定してマルチステートメント部分にタイマアウト処理を書きます。TMOUTがなければWAIT文の通常仕様になります。このTMOUTはWS0(),WS1()のTMOUT設定値には影響しません。

```
*aho
WAIT s==1 TMOUT 1000 : GOTO *err
ON 0
TIME 100
WAIT s==1 TMOUT 1000 : ON 128 : GOTO *err
OFF 0
TIME 100
GOTO *aho
*err
```

WARP

MPG-314 **ワープジャンプ**

■書　式
WARP [軸指定] [Upz] P(n) [Dwnz] [WHEN 論理式]
　軸指定: 予約定数
　Upz: パルス
　n: 点番号
　Dwnz: パルス

■解　説

軸指定
省略するとX_A|Y_A|U_Aとなっています。X_A,Y_A,X_A|Y_Aを指定してU軸を使わない場合やX,Y軸のいずれかを使うようにできます。WARPで使わない軸は他のタスクで単独軸として使用できます。

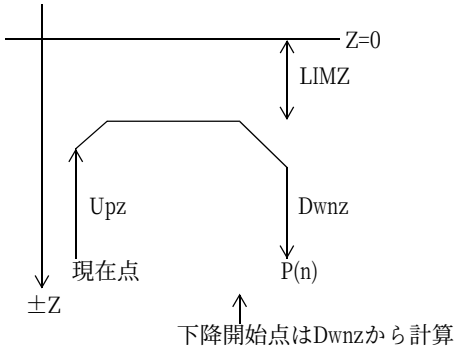
Upz
上昇時のZ軸垂直移動量。省略するとゲートモーションとなります。また、10以下の値は指定できません。

Dwnz
下降時のZ軸垂直移動量。省略するとゲートモーションとなります。また、10以下の値は指定できません。この値はリアルタイムでACCEL設定値から計算して制御されますので精度がよくありません。最初大き目に設定してください。また、最小速度を0とした近似計算のため、最高速度と最小速度の比が1/10の以下の場合は不正確になります。

WHEN 論理式
Z軸をおろし始めるところで条件式が成立していることを確認して下降にはいります。成立していないとZ下降をしません。WHEN文を省略すると無条件でワープジャンプとなります。

```
WARP X_A 1000 P(3) 1500 WHEN 1==HSW(-1)&HSW(-2)   /* X軸とZ軸のみの条件付ﾜｰﾌﾟ
WARP 1000 P(3) 1500                               /* XYU軸とZ軸の無条件ﾜｰﾌﾟ
WARP X_A|Y_A 1000 P(3) 1500 WHEN 1==HSW(-1)       /* X/Y軸とZ軸のみの条件付ﾜｰﾌﾟ
```

Z軸上昇量の制限はLIMZで設定します。



WEND

制御文 **条件ループ**

■書　式
WHILE 条件式～WEND

WHILE 参照

WHILE

制御文 **条件ループ**

■書　式
WHILE 条件式～WEND

■解　説
条件待ち制御文。WHILEはDO～LOOPのサブセットです。この場合は条件が成立している間という意味を特定しています。プログラムは読む人に意味を分かりやすく記述するのが大切ですが、このような意味のはっきりした制御文を使用することは、この目的のためです。

```
  a=1
  b=0
WHILE a<4
  DO UNTIL b>4
    PRINT a""b=b+1
  LOOP
  a=a+1
  b=0
WEND
```

WEND 参照

WIR

バスアクセス **ワード読み取り**

■書　式
WIR(adrs)
 0≦adrs≦&HFFFF（偶数のみ）

■解　説
アドレスadrsのワード読み込みです。ここでのアドレスは98バス側からみたアドレスであり、68000側と奇数と偶数が入れ替えられます。

WOW

バスアクセス **ワード書き込み**

■書　式
WOW n adrs
　0≦n≦&HFFFF
　0≦adrs≦&HFFFF（偶数のみ）

■解　説
アドレスadrsのワード書き込みです。WIRと同様98バス側からみたアドレスとなります。

WS0

I/O **タイムアウト付き入力**

■書　式
WS0(n)
WS1(n)
　n: 入力ポート番号

■解　説
WS0はnで指定されたポートの値が0(OFF)になるのを待ちます。WS1(n)は1(ON)になるのを待ちます。待ち時間はTMOUTで設定することができ、その時間を越えると1を返します。その時間内に条件が成立した場合は0を返します。

```
TMOUT 5
   n=192
IF WS1(n)==1 THEN      ’sw(n)が5秒以内にオンに
   GOTO *aho           ’ならない時はgoto *aho
END_IF
PRINT  ”スイッチ”n ”はオンです”
END
*aho
PRINT  ”タイムアウト”
```

また、WS1()、WS0()は条件のAND、ORもできます。しかしMPC-684では1行（命令）が8文字列以内とい制約がありますから、論理結合も3つまでです。次のプログラムは出力0<->入力192,1<->193,2<->194を接続して実行しました。
①この場合は192,193の両方がタイムアウトにならなければ*AHOには行きません。

```
10   *MAIN
20     TMOUT 1
30     IF WS1(192)==1 AND WS1(193)==1 THEN
40       GOTO *AHO
50     END_IF
60     PRINT "Ok"
70     END
80   *AHO
90     PRINT "Time out!!"
#OFF 0
#OFF 1
#RUN
Time out!!
#ON 0
#OFF 1
#RUN
Ok
#OFF 0
#ON 1
#RUN
Ok
#ON 0
```

```
#ON 1
#RUN
Ok
#
```

②どちらか、または両方がタイムアウトすると*AHOに行きます。

```
30      IF WS1(192)==1 OR WS1(193)==1 THEN
```

③1つ以上タイムアウトすれば*AHOに行きます。

```
30      IF WS1(192)==1 OR WS1(193)==1 OR WS1(194)==1 THEN
```

WS1

I/O **タイムアウト付き入力**

■書　式
WS1(n)
　n: 入力ポート番号

WS0参照

X

パルス（MPG-314専用） **カウンタ値読み込み**

■書　式
X(-1)

■解　説
MPG-314に対して-1を与えるとカウンタ値を返します。カウンタは4バイト長(±2147483647)です。

```
#PG &h400
#STPS X_C 2147483647
#PR X(-1)
2147483647
```

STPS 参照

X

MPG-314 **カウンタ値読み込み&クリア**

■書　式
X(-2,A)

■解　説
PGコマンドで引き当てられているMPG-314のエンコーダカウンタの値×A を返す。読み込み後カウンタをクリアする。JOGダイアルを使用したティーチング機能を実現する。

X

MPG-314 **カウンタ値読み込み&クリア（PG指定）**

■書　式
X(pgadrs,A)
　pgadrs: MPG-314ボードアドレス　(&H400～&H490)
サポート REV-3.83a 以降

■解　説
pgadrsで指定されたMPG-314のエンコーダカウンタの値×A を返す。読み込み後カウンタをクリアする。JOGダイアルを使用したティーチング機能を実現する。次は&H400のMPG-314に接続されたJOGダイアルの操作に従い、&H410のX軸を制御する。

```
PG &H410
DO
  RMVS X_A X(&H400,20)     /*JOGﾀﾞｲｱﾙの回転に従いX軸が動く
LOOP
```

X

パルス（MPG-68K互換）　　**X軸点データ**

■書　式
X(n)
Y(n)
U(n)
Z(n)
n: ポイントナンバー
1≦n≦13000

■解　説
予約配列です。点データP(n)の各座標成分は予約配列として表現されます。nが0の時は現在位置を表しますが、代入の場合は現在位置の変更とはなりません。ホストCPUのパルス発生モードの現在位置が変更されるだけで、MPG側は変更されません。点データとかかわりなく独立した配列として使用できます。又、NEWPでクリアされます。

```
#PRINT X(0)            ’現在位置のX成分表示
 3764
#PRINT Y(99)           ’P(99)のY成分表示
 100
#Y(99)=5000
#PRINT Y(99)
 5000
#
```

Y

パルス（MPG-314専用）　　**カウンタ値読み込み**

■書　式
Y(-1)

X,STPS 参照

Y

MPG-314　　**カウンタ値読み込み&クリア**

■書　式
Y(-2,A)

X,STPS 参照

Y

MPG-314 **カウンタ値読み込み&クリア（PG指定）**

■書　式
Y(pgadrs,A)
　pgadrs: MPG-314ボードアドレス　(&H400～&H490)

X,STPS 参照

Y

パルス（MPG-68K互換） **Y軸点データ**

■書　式
Y(n)
　n: ポイントナンバー

X 参照

Z

パルス（MPG-314専用） **カウンタ値読み込み**

■書　式
Z(-1)

X,STPS 参照

Z

MPG-314 **カウンタ値読み込み&クリア**

■書　式
Z(-2,A)

X,STPS 参照

Z

MPG-314 **カウンタ値読み込み&クリア（PG指定）**

■書　式
Z(pgadrs,A)
　pgadrs: MPG-314ボードアドレス　(&H400～&H490)

X,STPS 参照

Z

パルス（MPG-68K互換） **Z軸点データ**

■書　式
Z(n)
　n: ポイントナンバー

X 参照