第６章　コマンドリファレンス

６．１　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。

１）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。

２）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)    直接コマンドを実行します。
#A=10

#10 ON 0
20 PRINT SW(192)  プログラムとしてメモリーに格納されます。
30 A=10
```

３）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。

４）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８０文字以内です。一行８０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)。

```
10 ON 0 : ON 1 : ON 2
          └──────┴──────┴── 文
└── 文番号
└──────────────────┘
     80 文字以内
```

５）一文の文字列数

一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。

```
10 ON 0 1 2 3 4 5 6
   └─┴─┴─┴─┴─┴─┴─┴── 一文の文字列は 8 個以内
```

６）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。一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。
```

７）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。

①ラベルは先頭にアスタリスク（＊）の付く文字列です。

②ラベルは行の先頭になくてはなりません。

③ラベルは半角３７文字以内です。
④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 *ﾀｽｸ1
20 GOSUB *case1
1000 *case1
1010 FOR i=0 TO 255
       |
LIST *case1
```

８）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
```

９）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)
```

１０）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２０００個までです。ＤＩＭａ（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。

１１）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。

１２）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。

１３）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$+"CORP"
str$=""
```

１４）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。

１５）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
```

１６）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

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)
```

１７）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。
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。

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

```
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
```

１８）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

１９）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 ―――――┘
```

２０）予約変数

time$ date$

カレンダーＩＣサポートの文字列です。ｔｉｍｅ＄が時間、ｄａｔａ＄が日付の文字列です。

```
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。

６．２　コマンドリファレンス

コマンドリファレンスは次のような書式で書かれています。

```
【ON】  ←①        機能：Ｉ／Ｏ  ←②      種別：コマンド  ←③
                   サポート：ボード名／ＲＥＶ番号  ←④

書式  ：  ＯＮ  Ａ１［Ａ２  Ａ３  Ａ４  Ａ５  Ａ６］  ←⑤
          Ａ１～Ａ６：出力ポートナンバー

解説  ：  ・・・・・・・・・・・・・・・・・・・・・・  ←⑥
          関連：ＯＦＦ，ＯＵＴ  ←⑦
```

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）。また、△マークの付いたコマンドはＭＰＣ－６８４でサポート停止予定のものです。

② コマンド・関数の機能分類です。

③ コマンドまたは関数の種別です。

④ サポートボード名称，サポートＲＥＶ番号です。同じ名前のコマンドでもこのボード専用の使用方法がある場合や、追加サポートコマンドにこのマークがついています。例えば、ＡＣＣＥＬコマンドはＭＰＧ－６８Ｋ，ＭＰＧ－４０５に共通に使用できますが、ＭＰＧ－４０５専用の使用方法についてはサポート：ＭＰＧ－４０５と記してあります。

⑤ 引き数の与え方を表します。[] は省略可能です。引き数が複数有る場合は１つ以上のスペースで区切ります。書式欄の引き数の意味はおおよそ次の通りです。

Ａ	引き数並び　Ａ１～Ａ６
ｎ	引き数（変数、定数、式、代入式を含む）
ｓ＄	文字列・文字変数
ｖ	変数
［ｓ＄，ｖ］	文字列・文字変数；変数
［ｓ＄，ｎ］	文字列・文字変数；引き数
条件式	Ａ＞１　ＯＲ　Ａ＝＝１　、　ＡＨＯ＜＞１　ＡＮＤ　ＢＡＫＡ＝＝１　など

⑥ コマンド・関数について内容や使用方法を解説しています。

⑦ 関連するコマンドや関数です。

ここの解説以外にもプログラムのヒントになることがあるかも知れません。ぜひ一読を。

【＊ラベル】　　機能：制御文

■書　式　　*str

str＝文字列

■解　説　　ラベルとは最初に＊(アスタリスク)のつく文字列です。

```
*aho
    goto *aho
    gosub *aho
```

ＧＯＴＯやＧＯＳＵＢなどの移動先や、ＦＯＲＫの実行位置の指定はすべてラベルを使います。(半角３７文字以内)

【ＡＢＳ】　　機能：演算　　種別：関数

■書　式　　ABS(n)

■解　説　　ｎの絶対値を返します。

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

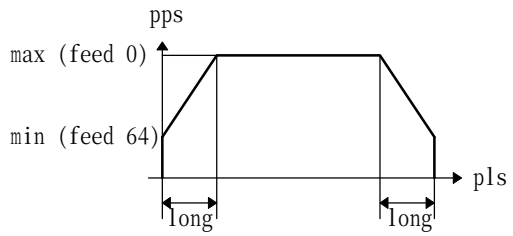
【ＡＣＣＥＬ】　　機能：パルス　　種別：コマンド

■書　式　　ACCEL max [long min]

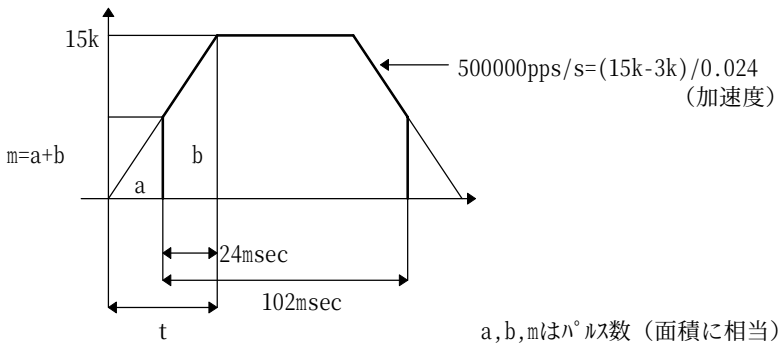
max：最大スピード
long：加減速領域パルス数
min：立ち上がりパルス数
100 ≦ min ≦ max ≦ 100000
long ≦ 10000

■解　説　　※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。
※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。

☆６８４のＡＣＣＥＬでの加速距離・加速時間



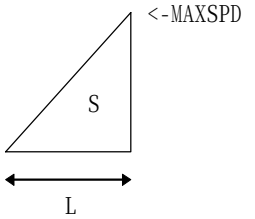
次の図は３ｋｐｐｓでスタートして１５kppsまで加速（２４msec）全体は１０２msecの間に１２５０パルス移動するというものです。



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。

パﾙｽ数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 =S=L*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=1/2 α t^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
```

【ＡＣＣＥＬ】　機能：パルス　種別：コマンド
サポート：ＭＰＧ－４０５

■書　式　ACCEL 0 &H□△

□１～４までの数字で乗数を表します。０は１とみなします。
△０～３までの数字で軸を指定します。
０（または１）＝Ｘ＋Ｙ軸、２＝Ｕ軸、３＝Ｚ軸

■解　説　ＭＰＧ－４０５では最大４００kppsのパルス発生が可能です。また、Ｘ＋Ｙ軸とＵ軸、Ｚ軸に異なるパルスレートを設定できます。例えば、ＸＹ軸にサーボモーター、Ｕ軸にはステップモーターを使い同時に移動する、というような装置の場合に、ＭＰＧ－６８Ｋでは加減速をどちらかにあわせなければならず都合の悪いことがおきましたが、ＭＰＧ－４０５では各々に適正な加減速を与えることができます。(ただしＸ＋Ｙ軸は直線補間となりますが、Ｕ軸は補間移動になりません)ＭＰＧ－６８Ｋに対するＡＣＣＥＬコマンドは１００kpps・全軸共通の加減速としてあたえますが、ＭＰＧ－４０５に対するＡＣＣＥＬは軸別に次の様になります。

```
#PG &HE0           <--PG 設定は MPC-68K(MPG)と同じです。
#ACCEL 0 &H40      ┐ X+Y 軸に対して 4 倍に設定。
#ACCEL 100000      ┘ つまり最高速 400Kpps となります。
#ACCEL 0 &H02      ┐ U 軸に対して 1 倍に設定。
#ACCEL 50000       ┘ 50Kpps となります。
#ACCEL 0 &H03      ┐ Z 軸に対して 1 倍に設定。
#ACCEL 25000       ┘ 25Kpps となります。

#ACCEL 0 &H00      ┐ X+Y 軸の ACCEL 設定値表示
#ACCEL             ┘
最大スピード 400000   加速距離 20000   最小スピード 20000

#ACCEL 0 &H02      ┐ U 軸の ACCEL 設定値表示
#ACCEL             ┘
最大スピード 50000   加速距離 2500   最小スピード 2500
```

```
#ACCEL 0 &H03
#ACCEL                    ┐Z軸のACCEL設定値表示
                          ┘
最大スピード 25000    加速距離 1250   最小スピード 1250
```

【ＡＤＲ】　　機能：ユーザー定義　　種別：関数

■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ＵＳＥＲＣＯＭ０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        ・・・ＩがＤ１に、Ｉ＊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】　　機能：Ｉ／Ｏ　　種別：コマンド

■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。

【ＡＲ$】　　機能：文字列　　種別：関数

■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
```

【ＡＳＣ】　　機能：文字列　　種別：関数

■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】　　機能：演算　　種別：三角関数

COS参照

【ＡＴＡＮ２】　　機能：演算　　種別：三角関数

COS参照

【ＢＳＹ】　　機能：パルス　　種別：関数

■書　式　BSY(n)

n：タスク番号

n=-1,0≦n≦23

■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
```

【ＣＡＳＥ】　機能：制御文　種別：コマンド

SELECT_CASE 参照

【ＣＡＳＥ＿ＥＬＳＥ】＜ＣＥＬ＞

機能：制御文　種別：コマンド

SELECT_CASE 参照

【ＣＨＲ＄】　機能：文字列　種別：関数

■書　式　CHR$(n)

n：1byte 長数

■解　説　ｎをアスキーコードを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)
```

【ＣＬＲＰＯＳ】＜ＣＬＰ＞　機能：パルス　種別：コマンド

■書　式　CLRPOS

■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
```

【ＣＭＮＤ】　　機能：ＭＰＧ－３２０２　　種別：コマンド

■書　式　CMND code

code：Ｘ３２０３アドレスと命令コードの和。

■解　説　ＭＰＧ－３２０２のパルス発生ＩＣ「Ｘ３２０２」の命令を実行します。

ＭＰＧ－３２０２関係コマンド・関数：CMND,REG,REG3,ST_REG

参照：「ＭＰＧ－３２０２ 製品別マニュアル」 ･･･ｗｅｂ参照または営業係へ請求下さい。

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

【ＣＮＦＧ＃０】　機能：ＲＳ－２３２Ｃ　　種別：コマンド

■書　式　CNFG#0

ＣＨ０（Ｊ２）の通信モード設定です。

CNFG#2

ＣＨ２（Ｊ１）の通信モード設定です。

各パラメータは次の文字列から選択し次のフォーマ文字列定数として与えます。

CNFG#n "[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－２３２Ｃ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
#
```

【ＣＮＦＧ＃２】　機能：ＲＳ－２３２Ｃ　種別：コマンド

CNFG#0 参照

【ＣＮＴ】　機能：デバッグ　種別：コマンド

■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。

【ＣＯＭＳＥＴ】　機能：宣言　種別：コマンド

■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】　機能：演算　種別：コマンド

■書　式　CONST v n

v：変数名

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語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。

【ＣＯＮＴ】　機能：タスク操作　種別：コマンド

■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
```

【ＣＯＳ】　　機能：演算　　種別：三角関数

☆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
```

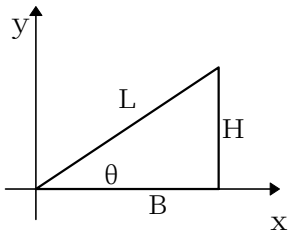
斜辺の長さと高さから角度を求めるには

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＞　　機能：パルス　　種別：コマンド

■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】△　　機能：カレンダー　　種別：コマンド

■書　式　　DATE　　日付の表示
DATE YYMMDD W　　日付の設定

YY：西暦
MM：月
DD：日
W ：曜日(0：日～6：土)

■解　説　　引き数が無ければ日付の表示。設定の場合は引き数を次のように入力します。日付の設定９２年１月１日月曜は次のようにします。

```
DATE 920101 1
```

Wは曜日で、日曜を０とし土曜を６としています。このコマンドはＩＣ１４（ＴＣ８５２１）が実装されていないと使えません。

【ｄａｔｅ$】△　機能：カレンダー　　種別：予約変数

■書　式　　date$

■解　説　　カレンダＩＣサポートの文字列です。

```
#DATE 990510 1                   ←カレンダ設定
#MPC-68K ADVFSC(r)m REV-2.68a    ←パワーオンリセット
 BASIC like + multi tasking
 Created by ACCEL Co.'91~99
#DT$=date$                       ←日付を文字列変数に取得
#PR DT$                          ←表示
99/05/10
```

time$,DATE参照

【ＤＥＬＥＴＥ】 ＜ＤＥＬ＞　機能：編集　　種別：コマンド

■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】　　機能：演算　　種別：コマンド

■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＿ＡＲ$】 機能：文字列　　種別：コマンド

■書　式　DIM_AR$ n m
　　n：文字長
　　m：確保数

■解　説　文字列配列を定義します。長さｎの文字列をm個Ｐ(１００００)側から確保します。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。

AR$参照

【ＤＩＲ】△　　機能：ファイル　　種別：コマンド

■書　式　DIR

■解　説　点データ用メモリファイルのディレクトリ参照です。拡張メモリが実装されている場合に有効です。

【DO.. ＬＯＯＰ】機能：制御文

■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
```

【ＤＵＭＰ】　　機能：デバッグ　　種別：コマンド

■書　式　　DUMP [adr]
　　adr：アドレス

■解　説　　メモリの内容を表示します。

【ＥＬＳＥ】　　機能：制御文　　種別：コマンド

IF参照

【ＥＮＤ】　　機能：制御文　　種別：コマンド

■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＞　　機能：制御文　種別：コマンド

IF参照

【ＥＮＤ＿ＳＥＬＥＣＴ】＜ＥＳＬ＞

機能：制御文　　種別：コマンド

SELECT_CASE参照

【ＥＲＡ】　　機能：ファイル　　種別：コマンド

■書　式　　ERA n
　　n：メモリーバンクナンバー
　　0≦n≦31

■解　説　　メモリファイルの削除です。指定された番号のメモリファイルを削除します。拡張メモリが実装されている場合に有効です。

【ＥＲＡＳＥ】　　機能：メンテナンス　　種別：コマンド

■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

【ＦＡＳＴ】　機能：制御文　種別：コマンド

■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。

【ＦＥＤＤ】　機能：パルス　種別：コマンド

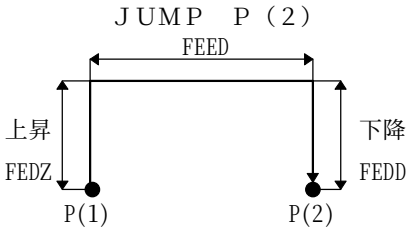
■書　式　FEDD n

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コマンドのゲートモーションの下降時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。ＭＰＧ－４０５はU軸に有効です。



【ＦＥＤＨ】　機能：パルス　種別：コマンド

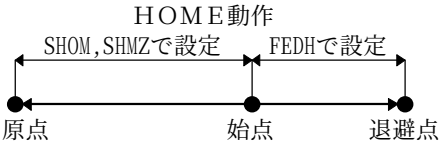
■書　式　FEDH n

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】　機能：パルス　種別：コマンド

■書　式　FEDT 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　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】　機能：パルス　種別：コマンド

■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】　　機能：パルス　　種別：コマンド

■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】　　機能：編集　　種別：コマンド

■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。

```
10      'FIND TEST
20        CONST PORT0 24
30      *LABEL
40        FOR count=0 TO 23
50          ON count
60          TIME 100
70          OFF count
80        NEXT count
90      *LABEL
#FIND                          ←ラベル重複を調べます
*LABEL[90][30]
#FIND "PORT0"                  ←PORT0という文字を含む行を探します
20        CONST PORT0 24       ←１行発見
#FIND "Port0"                  ←大小文字は区別されます
#                              ←みつかりませんでした
#FIND "count"                  ←countという変数を探します
40        FOR count=0 TO 23
50        ON count
70        OFF count
80        NEXT count
#FIND "FOR"                    ←コマンドの場合、探せないこともあります
40        FOR count=0 TO 23    ←FORは見つかりましたが、
#FIND "TO"                     ←TOは探せません。
#
```

【ＦＩＸ】　　機能：メンテナンス　　種別：コマンド

■書　式　FIX

■解　説　フラッシュメモリ(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。

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

RUN 参照

【ＦＯＲ.. ＮＥＸＴ】　　機能：制御文　　種別：コマンド

■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】　　機能：タスク操作　　種別：コマンド

■書　式　FORK n *ﾗﾍﾞﾙ
n：タスク番号
1≦n≦23

■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２４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
```

【ＦＲＥＥ】　　機能：編集　　種別：コマンド

■書　式　　FREE

■解　説　　メモリ残量表示。出荷時のmモデルではＮＥＷ後２３０kbyteです。ＡＤＶＦＳＣでは１行の消費メモリが約２２byteのため約１００００ステップ記述可能です。

【ＧＯ】　　機能：パルス　　種別：コマンド

■書　式　　GO ｘ ｙ ｕ ｚ
GO P(n)
※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。
```

【ＧＯＳＵＢ】　　機能：制御文　　種別：コマンド

■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
    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
```

【ＧＯＴＯ】　　機能：制御文　　種別：コマンド

■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＄】　　機能：文字列　　種別：関数

■書　式　　HEX$(n)

■解　説　　数値のヘキサ表現文字列への変換

```
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。
```

【ＨＩＮ】　　機能：Ｉ／Ｏ　　種別：関数

■書　式　　HIN(n)

■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。
```

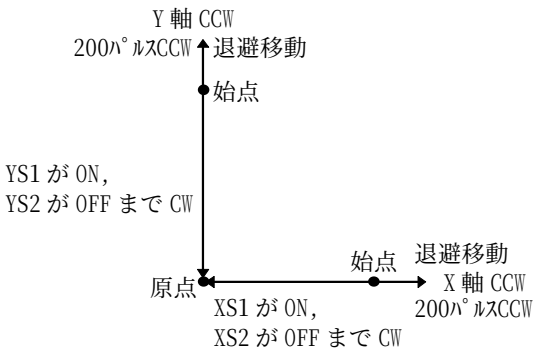
【HOME】　　機能：パルス　　種別：コマンド

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

■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、ＵＳ２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。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
```

【ＨＯＭＺ】　　機能：パルス　　種別：コマンド

■書　式　　HOMZ patn [z]
　　z：退避移動（パルス）
＊ＳＨＯＭ＜ＥＮＴＥＲ＞またはＳＨＭＺ＜ＥＮＴＥＲ＞で現在の設定状況が表示されます。

■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	
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０Ｎになるまで原点復帰動作
```

【ＨＯＵＴ】　　機能：Ｉ／Ｏ　　種別：コマンド

■書　式　　HOUT patn

■解　説　　ＭＰＣ－６８４、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。

```
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ＯＰ１、ＯＰ２をＯＮ
```

【ＨＰＴ】　　機能：Ｉ／Ｏ　　種別：関数

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

【ＨＳＷ】　機能：Ｉ／Ｏ　種別：関数

■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ポート　ＯＮ
PRINT HSW(0)            ０ポートの状態表示
 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
```

【ＩＦ】　機能：制御文　種別：コマンド

■書　式　IF　条件式　THEN
ELSE
END_IF<EIF>

■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
```

【ＩＮ】　機能：Ｉ／Ｏ　種別：関数

■書　式　IN(n)
n：バンクナンバー

■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。

【ＩＮＰ＄＃０】　機能：ＲＳ　ＣＨ０　　種別：関数

INPUT$参照

【ＩＮＰ＄＃２】　機能：ＲＳ　ＣＨ２　　種別：関数

INPUT$参照

【ＩＮＰＢＬＫ#】機能：ＲＳ－２３２Ｃ　　種別：コマンド

■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。

【ＩＮＰＵＴ】　機能：ＲＳ　ＣＨ１　　種別：関数

■書　式　　INPUT [s$,v]

INPUT#0 [s$,v]

INPUT#2 [s$,v]

■解　説　　ＣＨ１数値／文字列の入力コマンド

ＣＨ０数値／文字列の入力コマンド

ＣＨ２数値／文字列の入力コマンド

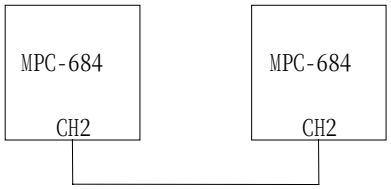
ＩＮＰＵＴ文は無手順系の通信コマンドでターミネータとしてスペース、タブ、ＣＲを使用するコマンドです。ＣＨ１のＩＮＰＵＴコマンドはエコーバックがありますがＣＨ０、ＣＨ２についてはエコーがありません。また、ＩＮＰＵＴ文も文字列を表示することができます。文字列定数・定数に対してはＰＲＩＮＴコマンドと同様出力として動作します。

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

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

☆ＩＮＰＵＴのターミネーター

ＭＰＣ－６８４のＲＳ－２３２Ｃ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－６８４２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-232C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－２３２Ｃ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 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－２３２Ｃ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２)。(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 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 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 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 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＄】　機能：ＲＳ　ＣＨ１　　種別：関数

■書　式　INP$#0(n)
INP$#2(n)
INPUT$(n)

■解　説　INP$#0：ＲＳ－２３２Ｃ　ＣＨ０よりｎ文字取り出す
INP$#2：ＲＳ－２３２Ｃ　ＣＨ２よりｎ文字取り出す
INPUT$：ＲＳ－２３２Ｃ　ＣＨ１よりｎ文字取り出す

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】機能：ＲＳ　ＣＨ０　　種別：関数

INPUT 参照

【ＩＮＰＵＴ＃２】機能：ＲＳ　ＣＨ２　　種別：関数

INPUT 参照

【ＩＮＴ１】△　　機能：割り込み　　種別：コマンド

■書　式　　INT1 n *ﾗﾍﾞﾙ
INT2 n *ﾗﾍﾞﾙ

■解　説　　ＩＮＴｎはＭＩＰ－０４８に付属のＩＮＴ１，ＩＮＴ２のトリガによって起動されるタスクを定めます。例えば”ＩＮＩ１　２３　＊ＡＨＯ”とすれば、＊ＡＨＯというラベルよりプログラムをＩＮＴ１の割り込み信号をスタート信号として、タスク２３として実行開始します。尚、タスクの起動がかかると割り込みは自動的に禁止され次にこのコマンドが実行されるまでＩＮＴ信号は無視されます。ＩＮＴ１，ＩＮＴ２の使用にはＭＩＰ－０４８のＤＩＰ２－ＩＲ１，ＩＲ２をＯＮにしておきます。

```
'
SETIO
'
DO
  INT1 23 *aho
  FOR i=0 TO 23
    ON i : TIME 100 : OFF i : TIME100
  NEXT i
LOOP
'*****************
*aho
PRINT "INT1割り込み処理"
FOR j=24 TO 47
  ON j : TIME 10 : OFF i : TIME 10
NEXT j
END
```

【ＩＮＴ２】△　　機能：割り込み　　種別：コマンド

INT1 参照

【ＩＯ】　　機能：Ｉ／Ｏ　　種別：コマンド

■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。

【ＩＯＲ】　　機能：Ｉ／Ｏ　　種別：関数

■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】　機能：Ｉ／Ｏ　種別：コマンド

IOR参照

【ＪＭＰＺ】　機能：パルス　種別：コマンド

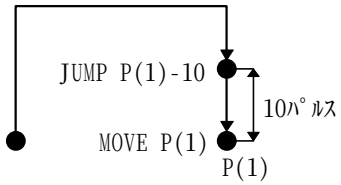
■書　式　JMPZ P(n)
n：ポイントナンバー
1≦n≦10000（mモデル）

■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。ＪＭＰＺはファンクションキィーでサポートされている他、ティーチモードの 'A 'によっても実施されます。

【ＪＵＭＰ】　機能：パルス　種別：コマンド

■書　式　JUMP P(n) [a]
n：ポイントナンバー
1≦n≦10000（mモデル）

■解　説　ＪＵＭＰはゲートモーションです。ＪＵＭＰはファンクションキィーでサポートされている他、ティーチモードの 'J 'によっても実施されます。又、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。



【ＬＥＮ】　機能：演算　種別：コマンド

■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】　　機能：演算　　種別：コマンド

■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。（）の中の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。

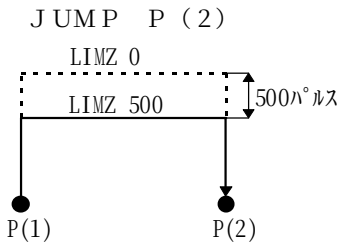
```
#LIST
40      aho$="HEX->"+"&H"+HEX$(100*100+1)
50    PRINT aho$
80    INPUT "何か入力して！" a$
90      b$="あなたが入力したのは－＞"
100     c$=b$+a$
110 PRINT c$
#run
HEX->&H2711
何か入力して！123
あなたが入力したのは－＞123
```

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。

【ＬＩＭＺ】　　機能：パルス　　種別：コマンド

■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、ＪＵＰＺ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】　機能：編集　種別：コマンド

■書　式　LIST [n m]
　　n：表示開始文字番号（省略すると前回の続き）
LIST [*ﾗﾍﾞﾙ m]
　　m：表示行数（省略すると２０）

■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

【ＬＯＦ】　機能：ＲＳ－２３２Ｃ　種別：関数

■書　式　LOF(n)
　　n：ＲＳ－２３２Ｃポートチャンネルナンバー

■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ＣＨ０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
```

【ＬＯＯＰ】　機能：制御文　種別：コマンド

DO.LOOP 参照

【ＭＢＫ】　機能：ＭＢＫ－６８　種別：関数

■書　式　MBK(mbkadr[+opt])
　　mbkadr：ＭＢＫレジスタアドレス
　　opt：オプション
　　　省略＝ワード（２ byte）読込み
　　　&HD000＝符号付ワード読込み
　　　&HB000＝バイト読込み

&HxE000=ビットセット＆クリア待ち xはビット番号0～7
&HxC000=ビット評価 xはビット番号0～7

■解　説　ＭＢＫ－６８のレジスタへアクセスする関数です。レジスタの読込みやＭＢＫコマンドの実行を行います。

ＭＢＫ－６８関係コマンド・関数：MBK,S_MBK,SW,ON,IN,OUT
参照：「ＭＢＫ－６８ 製品別マニュアル」 ･･･ｗｅｂ参照または営業係へ請求下さい。

【ＭＥＭ】　機能：メンテナンス　種別：コマンド

■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】　機能：デバッグ　種別：コマンド

■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]
#
```

【ＭＯＶＥ】　機能：パルス　種別：コマンド

■書　式　MOVE x y u
x,y,u：絶対座標（パルス）
MOVE P(n)
n：ポイントデータ
1≦n≦10000（mモデル）

■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。

【ＭＯＶＺ】　　機能：パルス　　種別：コマンド

■書　式　MOVZ z
　　z：絶対座標（パルス）
MOVZ P(n)
　　n：ポイントデータ
　　1≦n≦10000（mモデル）

■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。

【ＭＰＣ】　　機能：メンテナンス　　種別：コマンド

■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。

【ＭＰＣＩＮＩＴ】　機能：メンテナンス　　種別：コマンド

■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。

【ＭＵＬＴＩ】　機能：タスク操作　　種別：コマンド

■書　式　MULTI

■解　説　マルチタスクモードになります。ＡＤＶＦＳＣではＭＰＣ－６８４のパワーオン後はマルチタスク・モードとなっておりＦＯＲＫコマンドで指定されたプログラムを並列処理する事が出来ます。しかし、特定のアプリケーションではこのマルチタスクを必要としないこともあるため、コマンドＳＩＮＧＬＥによってマルチタスクを停止することもできます。このコマンドは、ＳＩＮＧＬＥコマンドによって停止されたマルチタスクを再スタートするものです。

【ＮＥＷ】　　機能：編集　　種別：コマンド

■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】　　機能：パルス　　種別：コマンド

■書　式　　NEWP

■解　説　　点データの初期化です。点データの全ての座標値を０にします。

【ＮＥＸＴ】　　機能：制御文　　種別：コマンド

FOR.NEXT 参照

【ＮＯＴ】　　機能：演算　　種別：関数

■書　式　　NOT(n)

■解　説　　ｎの補数を返します。

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

【ＯＦＦ】　　機能：Ｉ／Ｏ　　種別：コマンド

ON 参照

【ＯＮ】　　機能：Ｉ／Ｏ　　種別：コマンド

■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～-256 はメモリーＩ／Ｏ

■解　説　　出力に定義されたポート(ＭＯＰ－０４８)に対して有効なコマンドです。尚、７６８～７７５は、ＭＰＳ－３２４のリレー出力に対応します。

```
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】　　機能：Ｉ／Ｏ　　種別：コマンド

■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　Ｉ／Ｏのパラレルアウト。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－３２はメモリーＩ／Ｏです。

```
out &h0f -32
prx in(-32)
  000F
```

【Ｐ】　　機能：パルス　　種別：関数

■書　式　P(n)

0≦n≦10000

■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】　　機能：Ｉ／Ｏ　　種別：関数

P_SW 参照

【P＿ＩN】　　機能：Ｉ／Ｏ　　種別：関数

P_SW参照

【P＿ＬD】　　機能：パルス　　種別：コマンド

■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

■解　説　フラッシュ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】　　機能：Ｉ／Ｏ　　種別：コマンド

P_SW参照

【P＿ON】　　機能：Ｉ／Ｏ　　種別：コマンド

P_SW参照

【P＿ＯＵＴ】　　機能：Ｉ／Ｏ　　種別：コマンド

P_SW参照

【P＿ＳV】　　機能：パルス　　種別：コマンド

■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。

```
#P_SV
**--                 ←保存中表示
```

P_LD参照

【P＿ＳW】　　機能：Ｉ／Ｏ　　種別：関数

■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。

ﾋﾟﾝ	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	

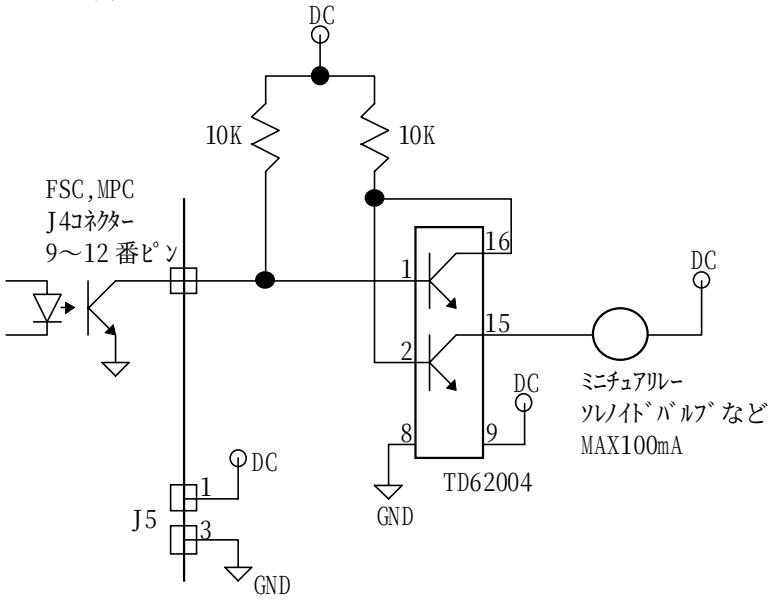
```
#P_ON 0                    'ＯＰ１をＯＮ
#PRX P_IN(0)               'パラレル入力上位バイトが出力、下位バイトが入力
 0100                      ・・・出力はＯＰ１がＯＮ、入力は全部ＯＦＦ
#PRINT P_SW(0)             'ＩＮ１のビット入力
0                          ・・・入力ＩＮ１はＯＦＦ
#P_OUT &H03                'パラレル出力
#PRX P_IN(0)
 0300                      ・・・出力はＯＰ１，ＯＰ２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

☆ＭＰＧ－６８ＫＪ４コネクターの出力でリレー等を制御するには
この出力はフォトカプラーオープンコレクターです。シンク電流は数mA程度ですから、リレー・ソレノイドバルブなどのアクチェーターは直接制御できません。外部回路でＩ／Ｆします。

出力Ｉ／Ｆ例



【ＰＡＬＥＴ１】　機能：パルス　　種別：コマンド

■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)～ＰＬ４(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】　機能：パルス　　種別：コマンド

PALET1 参照

【ＰＡＬＥＴ３】　機能：パルス　　種別：コマンド

PALET1 参照

【ＰＡＬＥＴ４】　機能：パルス　種別：コマンド

PALET1 参照

【ＰＡＵＳＥ】　機能：タスク操作　種別：コマンド

■書 式　PAUSE A

A：タスクナンバー　1～23

■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。

```
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
```

【ＰＥＥＫ】　機能：メモリーアクセス　種別：コマンド

■書 式　PEEK(adrs)

&HC0000 ≦ adrs ≦ &HFFFFF

■解　説　ユーザメモリの値をワード読みだしします。ａｄｒｓが奇数指定されても偶数に切り捨てされます。

【ＰＧ】　機能：パルス　種別：コマンド

■書　式　PG adrs [task]

0 ≦ adrs ≦ &HFFFF

0 ≦ task ≦ 23

adrs：スレーブボードアドレス

task：タスクナンバー

＊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　スレーブボードの使用宣言文です。ａｄｒｓによって指定されたスレーブボードをｔａｓｋで指定されたタスクに結合します。例えば、ＰＧ　&ＨＥ０　１と実行すればタスク１で実行されるＭＯＶＥやＡＣＣＥＬはアドレス&ＨＥ０のＭＰＧ－６８Ｋ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。また、ＰＧ＜ＥＮＴＥＲ＞としますと現在設定されているスレーブアドレスと、タスク番号が一覧できます。尚、ＰＧはティーチング中の切り換えコマンドではありません。この場合はＰＧＳＥＬを使用して下さい。

```
#pg &he0 0
#pg &he4 1
#pg &he8 21
#pg &hec 22
#pg
```

指定されているＰＧアドレス－&ＨＥ０

```
TASK  0 for PG &hE0  TASK  1 for PG &hE4
TASK  2 for PG &h00  TASK  3 for PG &h00
TASK  4 for PG &h00  TASK  5 for PG &h00
TASK  6 for PG &h00  TASK  7 for PG &h00
TASK  8 for PG &h00  TASK  9 for PG &h00
TASK 10 for PG &h00  TASK 11 for PG &h00
TASK 12 for PG &h00  TASK 13 for PG &h00
TASK 14 for PG &h00  TASK 15 for PG &h00
TASK 16 for PG &h00  TASK 17 for PG &h00
TASK 18 for PG &h00  TASK 19 for PG &h00
TASK 20 for PG &h00  TASK 21 for PG &hE8
TASK 22 for PG &hEC  TASK 23 for PG &h00
#
```

☆ＰＧ宣言とタスクの関係

```
PG &HE0 1                          <--adrs &HE0 のボードとﾀｽｸ1を引き当て
FORK 1 *TASK1
(中略)
*TASK1
 ACCEL 50000 10000 5000            <--adrs &HE0のPGﾎﾞｰﾄﾞに有効
```

前記のようにタスク０で　PG &HE0 1　と宣言してもタスク０が＆ＨＥ０に引き当てられたわけではありません。ですからつぎの①のようにＰＧコマンドのすぐ後にパルスコマンドを書いてもそれはタスク０に引き当てられているＰＧに対して命令されることになります。ＰＧとタスクのＦＯＲＫ、パルスコマンドは②か③のようになります。④はＰＧ宣言をしたあとにタスク０のボードを一時的に切り替えてＡＣＣＥＬコマンドを実行しています。

①

```
10        PG &HE0 1
20        ACCEL 50000 10000 50000          <-これは&HE0に有効ではない
30        FORK 1 *TASK1
40        END
50      *TASK1
60        DO
70          TIME 500
80          MOVE 5000 5000 5000
90          TIME 500
100         MOVE 0 0 0
110       LOOP
```

②

```
10        PG &HE0 1
20        FORK 1 *TASK1
30        END
40      *TASK1
50        ACCEL 50000 10000 5000           <-ﾀｽｸの中で実行
60        DO
70          TIME 500
80          MOVE 5000 5000 5000
90          TIME 500
100         MOVE 0 0 0
110       LOOP
```

③

```
10        FORK 1 *TASK1
20        END
30      *TASK1
40        PG &HE0                          <-PG宣言もﾀｽｸの中で
50        ACCEL 50000 10000 5000
60        DO
70          TIME 500
80          MOVE 5000 5000 5000
90          TIME 500
100         MOVE 0 0 0
110       LOOP
```

④

```
10        PG &HE0 1
20        PG &HE4 2
30        PG &HE0
```

```
40        ACCEL 50000 10000 5000
50        PG &HE4
60        ACCEL 30000 5000 5000
70        FORK 1 *TASK1
80        FORK 2 *TASK2
90        END
100     *TASK1
130       DO
140         TIME 500
150         MOVE 5000 5000 5000
160         TIME 500
170         MOVE 0 0 0
180       LOOP
190     *TASK2
200       DO
210         TIME 300
220         RMOV 10000 10000 10000
230         TIME 300
240         RMOV -10000 -10000 -10000
250       LOOP
```

【ＰＧＳＥＬ】　　機能：パルス　　種別：コマンド

■書　式　PGSEL [n]

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を割り付けるものです。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】　　機能：パルス　　種別：コマンド

■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

■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
```

【ＰＬ２】　機能：パルス　種別：コマンド

PL1参照

【ＰＬ３】　機能：パルス　種別：コマンド

PL1参照

【ＰＬ４】　機能：パルス　種別：コマンド

PL1参照

【ＰＬＩＳＴ】＜ＰＬＳ＞　機能：編集　種別：コマンド

■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。

【ＰＯＫＥ】　機能：メモリーアクセス　種別：コマンド

■書　式　POKE n adrs

0≦n≦HFFFF

&HC0000≦adrs≦&HFFFFF

■解　説　ユーザメモリに対してワード単位でデータを書き込みます。奇数アドレスを指定しても切り捨てにより偶数アドレスとして扱われます。

【ＰＲＩＮＴ】＜ＰＲ＞機能：ＲＳ－２３２Ｃ　種別：コマンド

■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。したがって、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"
```

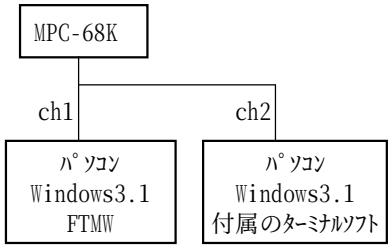
ＰＲＩＮＴ＃で通信相手側がＣＲ、ＬＦコードを必要とする場合（N88BASIC INPUTｺﾏﾝﾄﾞなど）には次のように記述します。

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

ＰＲＩＮＴ＃でのＲＳ－２３２Ｃ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$=0a$=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。

【ＰＲＩＮＴ＃０】 機能：ＲＳ－２３２Ｃ　　種別：コマンド

PRINT 参照

【ＰＲＩＮＴ＃２】 機能：ＲＳ－２３２Ｃ　　種別：コマンド

PRINT 参照

【ＰＲＸ】　　機能：ＲＳ－２３２Ｃ　　種別：コマンド

■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。

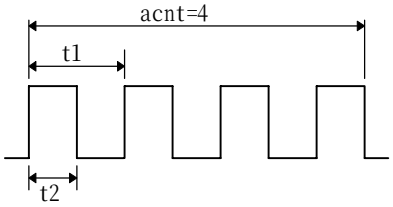
```
#a=&HFF
#PRINT a                    <--１０進表示
 255
#PRX a                      <--ヘキサ表示
 00FF
#PRX IN(24)
 0055
```

【ＰＵＬＳＥ】　機能：パルス　種別：コマンド

■書　式　PULSE axis acnt [t1 t2]

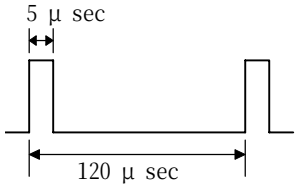
axis：軸 1～8
acnt：パルス数
t1：パルス間隔　（t1=約ｎ×1.2 μ sec）
t2：パルス幅　3～65535　（t2=n×0.25 μ sec）

■解　説　定速パルス発生です。ｔ１で指定された間隔でａｃｎｔパルスａｘｉｓで指定された軸にパルスを出力します。



1:x-cw　2:x-ccw　3:y-cw　4:y-ccw　5:u-cw
6:u-ccw 7:z-cw　8:z-ccw

☆ｔ１，ｔ２省略時の値
パラメーターｔ１，ｔ２を省略すると、
t1=100
t2=20
の値が入ります。これは省略時に強制的に設定される値であり、前のパラメーターが残るものではありません。



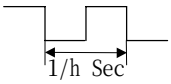
☆お詫び
ｔ２の値は１～６５５３５とありますが、１または２の設定にバグがあり正常に動作しません。ｔ２は３以上に設定して下さい。（'94/03/07）

【ＰＵＬＳＥ】　機能：パルス　種別：コマンド
サポート：ＭＰＧ－４０５

■書　式　PULSE axis acnt h

axis：軸 1～8
1:XCW 2:XCCW 3:YCW 4:YCCW 5:UCW 6:UCCW 7:ZCW 8:ZCCW
acnt：パルス数
h：パルス周波数

■解　説　hで指定された周波数でａｃｎｔパルスを、ａｘｉｓで指定された軸に出力します。第３パラメータには周波数を設定します。デューティーは５０%固定です。



【ＰＵＴ】　機能：ＲＳ－２３２Ｃ　種別：コマンド

■書　式　PUT [A1 A2 A3 A5 A6]
PUT#0 [A1 A2 A3 A5 A6]
PUT#2 [A1 A2 A3 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】　機能：ＲＳ－２３２Ｃ　種別：コマンド

PUT 参照

【ＰＵＴ＃２】　機能：ＲＳ－２３２Ｃ　種別：コマンド

PUT 参照

【Ｑ＿ＰＡＵＳＥ】 機能：パルス　種別：コマンド
サポート：REV1.2 以降

■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
注意）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。

■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、ＰＵＬＳＥ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
```

【ＱＵＩＴ】　　機能：タスク操作　　種別：コマンド

■書　式　QUIT A

QUIT

A：タスクナンバー　1～23

■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 １２３・・・・・タスク１，２，３を停止

タスク０（メインタスク）は停止できません。

【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 参照

【ＲＡＭ】　機能：編集　種別：コマンド

■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モード
**
#
#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モード起動時のメッセージ
#

#ROM                                                    ←フラッシュＲＯＭモード
**

#
```

【ＲＡＮＧＥ】　機能：パルス　種別：コマンド
サポート：REV2.21 以上(MPC,MPG 共に)

■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　現在位置を領域設定データとして保存します。通常ｔｅａｃｈのインチング動作により動作最大位置まで移動し、その位置でＲＡＮＧＥコマンドを実行します。ＲＡＮＧＥはｔｅａｃｈ組み込まれており、“Ｒ”によって実行することができます。

```
PG[0,0E0] X= 20000 Y= 0 U= 0 Z= 0  dx= 500 dy= 500 du= 500 dz= 500 /*“Ｒ”ｷｰを押す
･････!!領域設定 [1:最大,2:最少]1
PG[0,0E0] X= 10000 Y= 0 U= 0 Z= 0  dx= 500 dy= 500 du= 500 dz= 500 /*“Ｒ”ｷｰを押す
･････!!領域設定 [1:最大,2:最少]2
```

ｔｅａｃｈを使用しない場合はＳＥＴＰＯＳによって現在位置を強制的に変更しＲＡＮＧＥコマンドを実行します。

```
PG &HE0
 SETPOS 10000 20000 0 0
 RANGE 1
 SETPOS -10000 -20000 0 0
 RANGE 2
CLP
```

領域の最大値・最小値が０の場合領域チェックはしません。ＭＰＣＩＮＩＴにより初期化されたあとはこの値が０となっており領域チェックはありません。この制限値はパルス発生を担当するＭＰＧによって管理されており領域を越えようとするとエラーメッセージを出力します。またはＰＧ　０とすればＭＰＣでも制限を設けることができますがこの場合エラーメッセージは出力されません。
（ＭＰＧ－４０５はサポートしていません）

☆影響を受けるコマンド
次の各コマンドとそのコマンドより構築されるコマンドに対して有効です。例えば原点復帰前の待避移動も相対移動のためこの制限をうけます。また、ＪＵＭＰ、ＪＭＰＺも次のコマンドより合成されるためこの制限をうけます。TEACH コマンドによりインチング動作もこの制限をうけます。

```
MOVE,RMOV,MOVZ,RMVZ,GO,RM,JUMP,JMPZ
```

☆ＲＡＮＧＥは絶対座標での動作範囲を設定します。上限と下限をどちらも正（または負）に設定しても有効です。

```
#SETPOS -10000 0 0 0
#RANGE 1
#SETPOS -20000 0 0 0
#RANGE 2
#CLP
#PR P(0)
0 0 0 0
#MOVE -1000 0 0
 ＰＧが動作領域を越えようとしました          <--ﾌﾟﾛｸﾞﾗﾑ中ではMPCの赤LEDが点滅します
#MOVE -15000 0 0
#MOVE 0 0 0
 ＰＧが動作領域を越えようとしました
#
```

☆ティーチングしたポイントデータでＲＡＮＧＥを設定するには..

```
#T
 PG[0,0E0] X= 1150 Y= 0 U= 0 Z= 0  dx= 50 dy= 50 du= 50 dz= 50     /*“Ｐ”ｷｰを押す
点番号を指定して下さい  P181                                      /* P181とﾃｨｰﾁﾝｸﾞ
 PG[0,0E0] X= -400 Y= 0 U= 0 Z= 0  dx= 50 dy= 50 du= 50 dz= 50     /*“Ｐ”ｷｰを押す
点番号を指定して下さい  P191                                      /* P191とﾃｨｰﾁﾝｸﾞ

#SETPOS X(181) 0 0 0
#RANGE 1                    /* X(181)を上限に設定
#SETPOS X(191) 0 0 0
#RANGE 2                    /* X(191)を下限に設定
#RANGE 11                   /*確認のため現在位置を上限位置と置き換え表示してみます。
#PR P(0)
1150 0 0 0
#RANGE 12                   /*確認のため現在位置を下限位置と置き換え表示してみます。
#PR P(0)
-400 0 0 0
#
```

1995/04/12,05/22

【ＲＤ】△　　機能：バスアクセス　　種別：コマンド

■書　式　RD(n)

0≦ｎ≦&HFFFF

■解　説　ｎで指定されたスレーブボードから１キャラ読み取ります。ｎが０の時はＬＩＮＫ或いはＰＧコマンドで指定された自分のタスク番号のアドレスとなります。ｎを０以外の値とすればｎで指定されたアドレ

スを直接アクセスします。

【ＲＥＧ】　　機能：ＭＰＧ－３２０２　　種別：関数

■書　式　　REG(reg)

reg：Ｘ３２０２のアドレスとレジスタ・カウンタセレクトコードの和。
またはステータスレジスタアドレス。

■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。

ＭＰＧ－３２０２関係コマンド・関数：CMND,REG,REG3,ST_REG
参照：「ＭＰＧ－３２０２ 製品別マニュアル」 ･･･ｗｅｂ参照または営業係へ請求下さい。

```
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
```

【ＲＥＧ３】　　機能：ＭＰＧ－３２０２　　種別：関数

■書　式　　REG3(reg)

reg：Ｘ３２０２のアドレスとレジスタ・カウンタセレクトコードの和。
またはステータスレジスタアドレス。

■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。

ＭＰＧ－３２０２関係コマンド・関数：CMND,REG,REG3,ST_REG
参照：「ＭＰＧ－３２０２ 製品別マニュアル」 ･･･ｗｅｂ参照または営業係へ請求下さい。

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

【ＲＥＮＵＭ】 ＜ＲＮＭ＞　機能：編集　　種別：コマンド

■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＞ 機能：制御文

■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。

【ＲＬＳ】　　機能：タスク操作（セマフォ）種別：関数

RSV参照

【RM】　　機能：パルス　　種別：コマンド

■書　式　RM ｘ ｙ ｕ ｚ

x,y,u,z：相対移動量（パルス）

■解　説　４軸同時の相対パルス発生です。ｘｙｕｚ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。

【RMOV】　　機能：パルス　　種別：コマンド

■書　式　RMOV ｘ ｙ ｕ

x,y,u：相対移動量（パルス）

■解　説　３軸同時の相対パルス発生です。ｘｙｕ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。

【RMVZ】　　機能：パルス　　種別：コマンド

■書　式　RMVZ ｚ

ｚ：相対移動量（パルス）

■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。

【ROM】　　機能：編集　　種別：コマンド

■書　式　ROM

■解　説　フラッシュROM書き込みを有効にします。

RAM参照

【ＲＳ】　　機能：ＲＳ－２３２Ｃ　　種別：コマンド

■書　式　RS n

n：チャンネルナンバー

n=0または2

■解　説　対応するチャンネルナンバーのＲＳ－２３２Ｃ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】　　機能：ＲＳ－２３２Ｃエラー　種別：関数

■書　式　RSE(n)

n：チャンネルナンバー

n=0または2

■解　説　　ＲＳ－２３２Ｃ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
```

【ＲＳＶ】　　機能：タスク操作（セマフォ）種別：関数

■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
-256 ≦ 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－２３２Ｃ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。

☆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)
```

【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
```

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。

【ＲＵＮ】　　機能：デバッグ　　種別：関数

■書　式　　RUN [n m]

■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】　　機能：ＭＢＫ－６８　　種別：関数

■書　式　　S_MBK
S_MBK "string"
S_MBK outdata mbkadr[+opt]
　　string：文字列
　　outdata：書込みデータ
　　mbkadr：ＭＢＫレジスタアドレス
　　opt：オプション
　　　省略＝ワード（２byte）書込み
　　　&HB000＝バイト書込み
　　　&HA000＝ビットセット　outdataはビット番号０～７

&HC000=ビットリセット outdataはビット番号0～7

■解　説　ＭＢＫ－６８のレジスタへアクセスするコマンドです。ＧＰへ出力する文字列、四角形などのデータ設定を行います。

ＭＢＫ－６８関係コマンド・関数：MBK,S_MBK,SW,ON,IN,OUT

参照：「ＭＢＫ－６８　製品別マニュアル」 ･･･ｗｅｂ参照または営業係へ請求下さい。

【ＳＥＬＥＣＴ＿ＣＡＳＥ】 ＜ＳＬＣ＞

機能：制御文　　種別：コマンド

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

【ＳＥＮＳＥ＿ＳＷ】 機能：Ｉ／Ｏ　　種別：コマンド

■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します。感度は５msec固定です。これはシスクロックの固定タイマーのなかで検出しています。指定できるセンスは８個までです。パワーオン後、実行された順に登録されます。

```
SENSE_SW 192 -5
SENSE_SW 193 -6
DO
IF SW(-5)==1 THEN
ON 0
```

```
OFF -5
TIME 100
OFF 0
END_IF
IF SW(-6)==1 THEN
ON 1
OFF -6
TIME 100
OFF 1
END_IF
LOOP
```

【ＳＥＴ】　　機能：パルス　　種別：コマンド

■書　式　SET n [x y u z]
　　0≦n≦3
＊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　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
#
```

【ＳＥＴＩＯ】　　機能：Ｉ／Ｏ　　種別：コマンド

■書　式　SETIO [n m]

■解　説　出力の一括ＯＦＦに使用します。

【ＳＥＴＰ】　　機能：パルス　　種別：コマンド

■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≦10000（ｍモデル）
＊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)のデータをＰ(３)にコピー
SETP 10 PL2(5)       ･･･パレットの点ＰＬ２(５)をＰ(１０)にコピー
```

【ＳＥＴＰＯＳ】　機能：カウンター　　種別：コマンド

■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ｘｙｕｚ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。

【ＳＥＴＸ】　機能：パルス　　種別：コマンド

■書　式　SETX n X Y F+256*J

n：点番号　０～200

X：Ｘ軸座標

Y：Ｙ軸座標

F：スピード　０～64

J：0=直線、1=円弧

■解　説　ＭＰＧ－４０５の連続補間はＸＹ２軸に限定されており、連続移動中はＵ・Ｚ軸は動作しません。連続移動は２つのコマンドでサポートされています。点データを設定するＳＥＴＸコマンドと連続移動を指定するＴＲコマンドです。これにより、ティーチングによって教示しり数値によって与えた点を直線・円弧を交えて連続移動します。連続移動中は各点のスタート時に１０㎳secずつＭＰＧ－４０５のＯＰ１にパルスを発生します。この信号により移動途中のＩ／Ｏ制御を実施します。また、各点の移動には個別にスピードを設定することができます。ＳＥＴＸコマンドは点データをＭＰＧ－４０５に転送します。転送された点データはＭＰＧ－４０５に保存されます。円弧補間の場合はＪを１としますが、これは始点に設定します。円弧補間には始点・途中点・終点の３つの点が必要となりますが、始点に続く点を途中点、さらに次の点を終点と判断します。

TR参照

【ＳＦＴＬ】　機能：演算　　種別：コマンド

■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】　機能：演算　種別：コマンド

SFTL 参照

【ＳＨＭＺ】　機能：パルス　種別：コマンド

SHOM 参照

【ＳＨＯＭ】　機能：パルス　種別：コマンド

■書　式　SHMZ pat spd
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	7	6	5	4	3	2	1	0	LSB
SHOM			UCCW	UCW	YCCW	YCW	XCCW	XCW	
SHMZ	-	-	-	-	-	-	ZCCW	ZCW	

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１０００ pps で原点復帰。Ｚ軸はＣＣW方向に５００ pps 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。
```

【ＳＩＮ】　機能：演算　種別：三角関数

COS参照

【ＳＩＮＧＬＥ】　機能：タスク操作　種別：コマンド

■書　式　SINGLE

■解　説　マルチタスクの停止。マルチタスクモードを停止し、この命令以降はタイマー命令あるいは、タイマーコマンドを含むコマンド・関数が無い限りこのコマンドが実行されたタスクのみが実行されます。

【ＳＬＯＷ】　機能：ＲＳ－２３２Ｃ　種別：コマンド

■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】　機能：演算　種別：関数

■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
```

【ＳＱＲ】　機能：演算　種別：関数

■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。

【ＳＴ】△　機能：バスアクセス　種別：コマンド

■書　式　ST(n)
0≦n≦&HFFFF

■解　説　ｎで指定されたスレーブボードのステータスを読み取ります。ｎが０の時はＬＩＮＫ或いはＰＧコマンド指定された自分タスク番号のアドレスとなります。読み出すべきキャラクタがあれば、１となります。ｎを０以外の値とすればｎで指定されたアドレスを直接アクセスします。

```
#PRINT ST(&HE0)
1
#PRINT RD(&HE0)
65
#
```

【ＳＴ＿ＲＥＧ】　機能：ＭＰＧ－３２０２　　種別：コマンド

■書　式　ST_REG reg data

reg：Ｘ３２０２のアドレスとレジスタ・カウンタセレクトコードの和

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。

ＭＰＧ－３２０２関係コマンド・関数：CMND,REG,REG3,ST_REG

参照：「ＭＰＧ－３２０２ 製品別マニュアル」 ･･･ｗｅｂ参照または営業係へ請求下さい。

【ＳＴＯＰ】　　機能：パルス　　種別：コマンド

■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：タスクナンバー

0≦n≦23

＊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。

```
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 LOOP
150         MOVE 0 0 0              の場合、もしここに WAIT BSY()がないと TASK 0 から
155         WAIT BSY(-1)==1         STOP されて MOVEｺﾏﾝﾄﾞから抜けでても、すぐに次の MOVE
160         LOOP                    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
100         WAIT SW(193)==1         ﾊﾟﾙｽが止まらなくなる
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
150         MOVE 0 0 0              しまったので 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
145         TIME 1000             ｺﾏﾝﾄﾞ中で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
155         TIME 1000             このﾀｽｸ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
```

【ＳＴＲ＄】　機能：文字列　種別：関数

■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
```

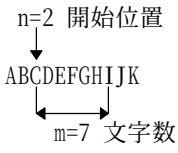
【ＳＴＲＣＰＹ】　機能：文字列　種別：コマンド

■書　式　STRCPY s1$ s2$ [n m]

■解　説　文字列のコピー(制御機能付き)

```
STRCPY A$ B$ n m          Ａ＄のｎ文字目からm文字をＢ＄にコピー
```

ｎは０文字目から数える。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を省略せずにし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
```

【ＳＷ】　　機能：Ｉ／Ｏ　　種別：コマンド

■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。ポート番号に出力ポートの状態を得ることもできます。

【ＳＷＡＰ】　　機能：タスク操作　　種別：コマンド

■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ュ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。

【ＳＷＰ】　機能：演算　種別：関数

■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】　機能：タイマー　種別：予約変数

■書　式　SYSCLK

■解　説　パワーオンより５msec毎に＋１される変数です。
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
```

【ＴＡＩＬ】　機能：編集　種別：コマンド

■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】　機能：演算　種別：三角関数

COS参照

【ＴＡＳＫ】　　機能：デバック　　種別：コマンド

■書　式　TASK(n)

n：タスクナンバー

0≦n≦23

■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

【ＴＡＳＫＮ】　　機能：タスク操作　　種別：予約変数

■書　式　TASKN

■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】　　機能：パルス　　種別：コマンド

■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

【ＴＥＳＴ】　　機能：メンテナンス　　種別：コマンド

■書　式　　TEST

■解　説　　６ヶのヘキサデータを表示します。表示される内容は次の順になっています。

```
TEST ①　②　③　④　⑤　⑥
```

① ＵＳＥＲＣＯＭでテスト用のＴＥＳＴモジュール
② ６８４で使用のＤＡＴＡエリアの最終アドレス
③ ６８４で使用のプログラムエリアの先頭アドレス
④ ６８４で使用のスタックエリアの先頭アドレス
⑤ ６８４で使用のＢＳＳエリアの最終アドレス
⑥ 現在のシステムスタックの最終アドレス

【ＴＩＭＥ】　　機能：タイマー　　種別：コマンド

■書　式　　TIME

■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＄】△　機能：カレンダー　　種別：予約変数

■書　式　　time$

■解　説　　カレンダＩＣサポートの文字列です。

```
#CLK 154800                  ←時間設定
#TM$=time$                   ←現在時間を文字列変数に取得
#PR TM$                      ←表示
15:48:18
#
```

date$,CLK参照

【ｔｉｍｅｒ】△　機能：カレンダー　　種別：予約変数

■書　式　　timer

■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】　　機能：メンテナンス　　種別：コマンド

■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】　　機能：タイマー　　種別：コマンド

■書　式　　TMOUT n

n：タイマー値（sec）

■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。

【ＴＯＦＦ】　　機能：デバック　　種別：コマンド

TON参照

【ＴＯＮ】　　機能：デバック　　種別：コマンド

■書　式　　TON　トレースモード
TOFF　トレースモード解除

■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]
```

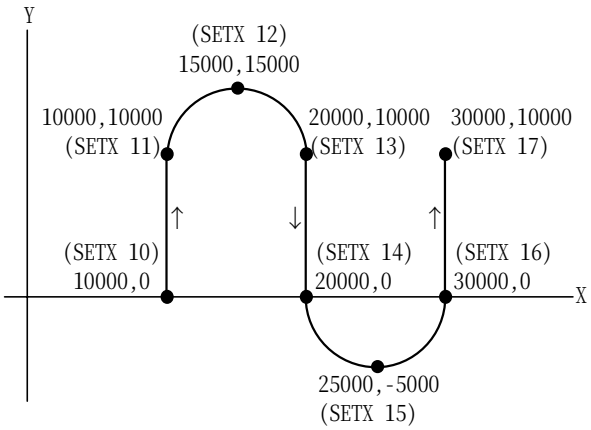
【ＴＲ】　　機能：パルス　　種別：コマンド

■書　式　　TR n m

n､m：ＭＰＧ－４０５の点データ番号
1≦n<m≦200

■解　説　　ＳＥＴＸコマンドで作成された点ｎ～mをたどって連続移動します。ＴＲコマンドは移動開始位置がＳＥＴＸで指定したｎの点に一致していないと動作しません。また、ｍ－１よりmの移動では減速停止となりますが、移動距離が不足していると減速が不完全となります。

```
SETX 10 10000 0 64              <--ｽﾀｰﾄ時はFEED 64
SETX 11 10000 10000 256         <--円弧始点
SETX 12 15000 15000 0           <--円弧途中点
SETX 13 20000 10000 0           <--円弧終点
SETX 14 20000 0 256
SETX 15 25000 -5000 0
SETX 16 30000 0 0
SETX 17 30000 10000 30
MOVE 10000 0 0                  <--ｽﾀｰﾄ位置まで移動
TR 10 17                        <--10～17まで連続移動
```



【95-02-09.95-08-07追加】(対象ＭＰＧ－４０５)
トレースコマンドにおいて動作位置を変更する機能を追加しました｡どのような位置に移動しても相対的に同じトレースを実行します。

■書　式　　TR -1 n
　　n=1～5　データ番号

■解　説　　保存された動作を再現します。TR -2 n コマンドによって保存された動作を再実行します。この時、現在位置テストは行われません。

■書　式　　TR -2 n
　　n=1～5　データ番号

■解　説　　直前のトレースを保存します。トレースコマンド実行後このコマンドを実行すると移動パターンをＭＰＧのメモリに保存します。

■書　式　　TR -3 1

■解　説　　通常のトレースコマンドの動作を禁止しします。その後TR動作を実行することによりメカを動作させずに移動パターンを演算することができます。ただし TR -1ではこの動作禁止は無効です。かならず動作します。

■書　式　　TR -4 1

■解　説　　動作禁止の解除

例１）

```
MOVE P(1)          <-- ﾃｨｰﾁﾝｸﾞした点1に移動
TR 1 5             <-- SETXで作成された点1～5を移動
TR -2 1            <-- 前行の移動をﾃﾞｰﾀ1に保存
MOVE P(6)          <-- ﾃｨｰﾁﾝｸﾞした点6に移動
TR 6 10            <-- SETXで作成された点6～10を移動
TR -2 2            <-- 前行の移動をﾃﾞｰﾀ2に保存
MOVE P(100)        <-- ﾃｨｰﾁﾝｸﾞした点100に移動
TR -1 1            <-- 点100の位置からﾃﾞｰﾀ1の移動を再現
MOVE P(101)        <-- ﾃｨｰﾁﾝｸﾞした点101に移動
TR -1 2            <-- 点101の位置からﾃﾞｰﾀ2の移動を再現
```

例２）

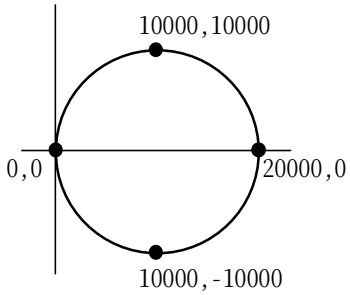
```
PG &HE0
ACCEL 20000
CLRPOS
SETX 5 10000 10000 256
SETX 6 20000 20000 0
SETX 7 20000 0 256
SETX 8 10000 10000 0
```

```
SETX 9 20000 20000 0
TR -3 1                          <-- トレースコマンドの動作禁止
TR 5 9                           <-- トレース計算実行
TR -2 3                          <-- データの保存
END
TR -1 3                          <-- 保存データの実行
TIME 1000
TR -1 3                          <-- 保存データの実行
```

ＴＲ　－２で作られたトレースデータはＭＰＧ－４０５のＳＲＡＭに保存されます。このＳＲＡＭはバッテリバックアップされているのでＭＰＣの電源を切っても消えませんが、これを保証するものではありません。必ずプログラムでＴＲ　－３を実行してデータを初期化して下さい。

☆連続円弧補間の例

```
  PG &HE0
  SETP 100 0 0
  SETP 101 10000 10000
  SETP 102 20000 0
  SETP 103 10000 -10000
'
    F1=15
'連続円弧
 SETX 1 X(100) Y(100) 256+F1
  FOR I=2 TO 18 STEP 4
    SETX I X(101) Y(101) 0
    SETX I+1 X(102) Y(102) 256
    SETX I+2 X(103) Y(103) 0
    SETX I+3 X(100) Y(100) 256
  NEXT I
'最後の動作
  SETX I X(101) Y(101) 0
  SETX I+1 X(102) Y(102) 0
'
  MOVE P(100)
  TR 1 I+2
```



そして、TR -1 n, TR -2 nを使って連続移動する様にします。

【U】　機能：パルス（演算）　種別：配列

X参照

【ＵＮＴＩＬ】　機能：制御文　種別：コマンド

■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。

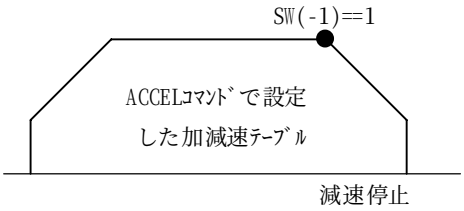
☆ＵＮＴＩＬ使用上の注意
ＤＯ～ＬＯＯＰの場合は論理結合ができます。

```
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-232C  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
#
```

【ＵＳＥＲＣＯＭ１～９】　機能：ユーザー定義　種別：コマンド

■書　式　　USERCOM0～9

■解　説　　ユーザーコマンドユーザーコマンドは初期状態(ＭＰＣＩＮＩＴ実行後)にはＵＳＥＲＣＯＭ０～ＵＳＥＲＣＯＭ９というコマンド名が与えられていますが、これは、ＣＯＭＳＥＴコマンドによって変更できます。このことにより、ユーザーコマンドはユーザーの使いやすい名前に変えて使用することが出来ます。ＣＯＭＳＥＴはコマンド名と実行アドレスを与えます。例えば、ＵＳＥＲＣＯＭ３にＧＯＯＤというコマンド名を与え実行番地を＄Ｃ００００とするには次のようにします。

```
COMSET 3 "GOOD"&HC0000
```

このように、ユーザーで用意したマシン後プログラムをＡＤＶＦＳＣに組み込みコマンドとして使用することが出来ます。尚、コマンドの引き数を得るにはＡＤＶＦＳＣに用意されたインターフェースライブラリを使用します。

【ＶＡＬ】　　機能：文字列　　種別：関数

■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
```

【ＶＥＲ】　　機能：メンテナンス　　種別：コマンド

■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-68K ADVFSC(r)m REV-2.68g    ････ｱﾄﾞﾊﾞﾝｽﾄﾞﾌｪｰｼﾞｯｸ版数
 BASIC like + multi tasking     ････ﾏﾙﾁﾀｽｸﾍﾞｰｼｯｸﾗｲｸｲﾝﾀﾌﾟﾘﾀ
 Created by ACCEL Co.'91~99     ････(株)アクセル製
```

【ＶＬＩＳＴ】　　機能：編集　　種別：コマンド

■書　式　VLIST

■解　説　プログラムリファレンスリストの表示です。

【ＷＡＩＴ】　　機能：制御文　　種別：コマンド

■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。

【ＷＥＮＤ】　　機能：制御文　　種別：コマンド

WHILE参照

【WHILE】　機能：制御文　種別：コマンド

■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
```

【WIR】　機能：バスアクセス　種別：関数

■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】　機能：バスアクセス　種別：コマンド

■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】　機能：I／O　種別：関数

■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】　　機能：Ｉ／Ｏ　　種別：関数

WS0参照

【Ｘ】　　機能：パルス（演算）　　種別：配列

■書　式　X(n)
Y(n)
U(n)
Z(n)
n:ポイントナンバー
１≦ｎ≦10000（ｍモデル）

■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】　　機能：パルス（演算）　　種別：配列

X参照

【Z】　　機能：パルス（演算）　　種別：配列

X参照