第３章　プログラミング例

基本1（I/O定義）

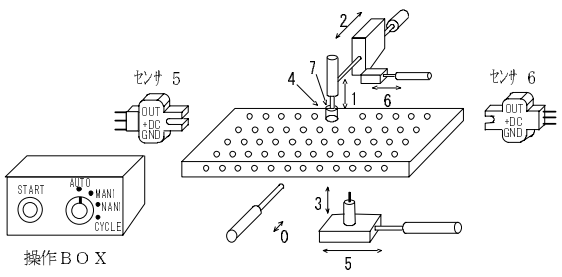
MPCの入出力はそれぞれ0番からスタートしています。MPC-816に付属している16点の入力と8点の出力は0～15の入力、0～7の出力にふりあてられています。MIFではこれが16～31（入力）8～15（出力）と順にふられています。MPCではI/Oをラベルで扱うことができます。これを定義するコマンドは#DEFS、#DEFOです。それぞれ入力ポート、出力ポートの定義に用います。もちろんラベルを使用しないでそのままポート番号で扱うこともできますが保守時にはラベル化されたI/Oの方が扱いやすくなります。次に、定義例を示します。最初に0から6までの入力を指定し、次に0から7までの出力ポートを定義しています。この定義は最初にしなくても、プログラム完成後追加しても有効です。

```
#DEFS START 0
#DEFS AUTO 1
#DEFS MAN1 2
#DEFS MAN2 1
#DEFS CYCLE 4
#DEFS MIGI 5
#DEFS HIDARI 6
'
#DEFO PIN 0
#DEFO HANDDWN 1
#DEFO HANDMAE 2
#DEFO PINUP 3
#DEFO AIR 4
#DEFO MAE 5
#DEFO STOP 6
#DEFO VACUM 7
```

Ｉ／Ｏマップ

	ポートNO.	名　称	
入力	0	ＳＴＡＲＴ　スイッチ	押してＯＮ
	1	ＭＯＤＥ　スイッチ　ＡＵＴＯ	
	2	〃　　ＭＡＮ１	
	3	〃　　ＭＡＮ２	
	4	〃　　ＣＹＣＬＥ	
	5	テーブル右	さえぎってＯＦＦ
	6	テーブル左	〃
	7		
出力	0	位置決めピン	ＯＮ　位置決め
	1	トランスポート上下	ＯＮ　下降
	2	トランスポート前後	ＯＮ　前進（Ｃ）
	3	案内板送りピン上下	ＯＮ　上昇
	4	トランスポート真空破壊	ＯＮ　真空破壊
	5	案内板送り前後	ＯＮ　前進
	6	度当り切替え	ＯＮ　Ｂ位置
	7	トランスポート真空	ＯＮ　真空

※前後の前（前進）とはシリンダのロッドが伸びた状態



サンプル装置

基本２　（自動・手動運転の切り替え）

MPCにはマルチタスクの機能があります。これは、複数のプログラムを同時に実行するものです。この機能を使用することにより動作の切り替えを簡単に実現することができます。もしマルチタスクを使用しないと自動と手動の動作切り替えはプログラムを複雑なものにしてしまいます。

```
 '*******      ----------
*LOOP                    |
 '*******                |
 WAIT SW(START)=1        |
 WAIT SW(START)=0        |
 QUIT 1                  |
 IF SW(AUTO)=1 THEN *A1  |   管理プログラム
 IF SW(MAN1)=1 THEN *M1  |
*A1                      |
 FORK 1,*AUTO            |
 GOTO *LOOP              |
*M1                      |
 FORK 1,*MANU            |
 GOTO *LOOP    ----------
 '*******      ----------
*AUTO                    |
 '*******                |
 FOR I=0 TO 7            |   自動運転プログラム
 ON I                    |   ' プログラム自体には意味無し
 TIME 5                  |
 OFF I                   |
 NEXT I                  |
 GOTO *AUTO    ----------
 '*******      ----------
*MANU                    |
 '*******                |
 FOR I=0 TO 7            |   手動運転プログラム
 WAIT SW(MAN2)=1         |   ' プログラム自体には意味無し
 PRINT# I                |
 NEXT I                  |
 GOTO *MANU    ----------
```

この例ではSTARTスイッチの操作をきっかけにして、手動と自動を切り替えています。STARTスイッチ後AUTOとMAN1のスイッチを読み取りこれにより必要に応じて異なるプログラムをFORKしています。このFORKコマンドがマルチタスクを起動させるものです。入力ポートAUTOがONの場合このコマンド実行後、管理プログラムと自動運転プログラムが同時に稼働します。その次に、AUTOをOFFとしてMAN1をONとしてSTARTスイッチを操作すると管理プログラム中のQUITコマンドによりタスクが停止されます。もちろん動作をいきなり停止すると危険なプログラムもありますのでタスク間インターロックによってタイミングを考慮します。通常はサイクル停止となるようにします。このように動作を決めるプログラムと動作を管理するプログラムを分離することによって見通しの良いプログラムとなります。また、管理プログラム中で非常停止スイッチを監視するようにすればいつでもプログラムを停止したり特別な処理をすることができます。次の例では非常停止によってプログラム停止、出力ポートクリア(SETIO)後にプログラムを停止しています。

```
 '*******
*LOOP
 '*******
 IF SW(START)=1 THEN *NORMAL
 IF SW(EMG)=1 THEN *EMG
*NORMAL
 WAIT SW(START)=0
 QUIT 1
 IF SW(AUTO)=1 THEN *A1
 IF SW(MAN1)=1 THEN *M1
*EMG
 QUIT 1
 SETIO
 END
*A1
 FORK 1,*AUTO
 GOTO *LOOP
*M1
 FORK 1,*MANU
 GOTO *LOOP
```

管理プログラム

また、マルチタスクは管理プログラムだけでなく装置の上に分散する幾つものユニットに個別のプログラムを対応させて使用します。

基本３　（タスク間インターロック）

先の例のように管理プログラムが一方的にタスクを停止すると不都合なことが多いものです。このため、動作切り替えにはサイクル停止が必要になります。こうした場合はタスク（プログラム）間で信号のやりとりをして動作を定めます。

```
 '*******
*LOOP
 '*******
 WAIT SW(START)=1
 WAIT SW(START)=0
 ON -1
 WAIT SW(-2)=1
 QUIT 1
 IF SW(AUTO)=1 THEN *A1
 IF SW(MAN1)=1 THEN *M1
*A1
 OFF -1,-2
 FORK 1,*MANU
 GOTO *LOOP
*M1
 ON -2
 FORK 1,*MANU
 GOTO *LOOP
 '********
*AUTO
 '*******
 FOR I=0 TO 7
 ON I
 TIME 5
 OFF I
 NEXT I
 IF SW(-1)=0 THEN *AUTO
 ON -2
 WAIT SW(-1)=0
 GOTO *AUTO
```

管理プログラム

自動運転プログラム
’プログラム自体には意味無し

ここでは、メモリI/Oの-1と-2を使用して*AUTOのプログラムの停止を制御しています。このように相手のプログラムを特定の箇所で停止させたり、状態を知らせたりするのにはメモリI/Oが有効です。ここでは直接番号でメモリI/Oを指定していますが#DEFS、#DEFOで定義することもできます。注意すべきことはメモリI/Oは入出力が同じ番号のため、2つ定義文が必要となります。

```
#DEFS CMND -1
#DEFS ACK  -2
#DEFO CMND -1
#DEFO ACK  -2
```

基本４　（サブルーチン）

以上でプログラムを記述する場合の大まかな段取りができました。あと必要な事は細かい動作をどのように記述すれば良いかということです。これにはサブルーチンという考え方を使って、プログラムをよく整理します。もしこの整理が無いとプログラムは秩序の無い冗長なものとなって保守が難しくなります。次の例ではP&Pの動作をさせるのに、動作をいくつかの単位に分けて全体を制御しています。P&P1で1の位置に移動しCHACKでワークを囲みます。P&P0は退避位置に移動するサブルーチンです。P&P2は2の位置に移動します。その後RELSはワークを離します。この一連の動作をサブルーチンを使用しないで記述すると繰り返しが多いこと、全体の見極めがつきにくいことなどからプログラムの保守性が著しく悪くなります。またサブルーチン単体の動作確認はRUNコマンドで行います。

```
>RUN *P&P1
NO STATE!
>RUN *P&P0
NO STATE!
>
```

NO STATE!というエラーはサブルーチンのRETURNコマンドを実行した時に現れるものです。戻るべき文番号が無いために発生します。サブルーチンで注意すべきことはRETURNで戻らないでGOTOで戻してはならないことです。(必ずRETURNで戻して下さい)こうすると覚えておいた戻り番地が消費されないためにメモリ上にどんどん蓄積され30回ほどでエラー停止を招きます。
これまでのことをまとめるとプログラムはタスク、メインプログラム、サブルーチンによってよく整理して記述すべきだということです。モード切り替えなど全体に関わることはマルチタスクを使用して別のプログラムを起動するような見通しの良い構成にします。

```
*LOOP
 GOSUB *P&P1
 GOSUB *CHACK
 GOSUB *P&P0
 GOSUB *P&P2
 GOSUB *RELS
 GOSUB *P&P0
 GOTO *LOOP
```

```
*P&P1
 OFF HANDDWN,STOP
 TIME 50
 ON HANDMAE
 TIME 100
 ON HANDDWN
 TIME 100
 RETURN
```

```
*P&P2
 OFF HANDDWN
 ON STOP
 TIME 50
 ON HANDMAE
 TIME 100
 ON HANDDWN
 TIME 100
 RETURN
```

```
*P&P0
 OFF HANDDWN
 TIME 80
 OFF HANDME
 TIME 50
 RETURN
```

```
*CHACK
 ON VACUM
 TIME 80
 RETURN
```

```
*RELS
 OFF VACUM
 ON AIR
 TIME 20
 OFF AIR
 RETURN
```

パルス発生1 （モードの選択）

P版でのパルス発生 (MPG-303 から出力)

MPC-816でのパルス発生は基本的にはMPG-303を使用します。またパルス発生の方法にはMODE5とMODE6があります。MODE5はステップモータの駆動に用い、MODE6はサーボモータに使用します。これは、多くのステップモータドライバが10μから20μsec以上のパルス幅を必要としていることに対して、サーボドライバは、1μsec度に設定されていることに対応しているためです。パルス発生のパルスレートはACCELコマンドで設定します。次の表はACCELコマンドで与えた数値に対してどのような周波数でパルスが出力されるかを示しています。
MPG でパルス発生させるためには最低次のサンプルプログラムの機能設定が必要です。ここでは4 相のステップモータで最大スピードを8kppsとしています。

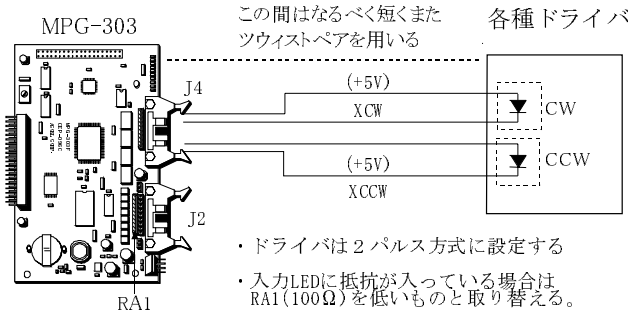
```
        PG 1
        MODE 5
        OVRUN 0
        D45 0
        ACCEL 8000
        SETPOS 0 0
*LOOP
        MOVE 10000 0
        TIME 100
        MOVE 0 0
        GOTO *LOOP
```

MPG-303ｶﾗﾊﾟﾙｽ発生				
REV 3.2(Pﾊﾞｰｼﾞｮﾝ)				
	MODE 5		MODE 6	
n	MOVE	JOG	MOVE	JOG
50000	--	--	56.3	14.6
45000	--	--	51.6	13.7
40000	--	--	44.2	12.0
35000	--	--	38.6	10.8
30000	29.0	13.7	32.5	9.3
25000	26.5	12.0	26.8	7.9
20000	21.1	8.9	21.3	6.4
15000	15.7	6.2	15.8	4.9
10000	10.4	3.9	10.3	3.3
8000	8.2	3.0	8.2	2.6
4000	4.1	1.4	4.1	1.3
2000	2.0	0.7	2.3	0.8
ﾊﾟﾙｽ幅	20 μ sec		4 μ sec	

(単位 kpps)

最初に設定すべきコマンドはPGです。MPG-303は、１つのシステムで３枚まで使用できます。このため、コマンドをどちらのMPGに対して与えるかを最初に選択する必要があります。PG 1でショートピンが一番目に設定されたMPGに対して有効となります。次にMODEコマンドでモードを選択し、オーバランの設定をクリアしておきます。OVRUNが設定されているとMPGのパルス動作が入力の状態によって停止します。D45は直線補間の方法を決めるコマンドで通常の直線補間では0とします。コマンドの詳細は各コマンドリファレンスを参照して下さい。コマンドSETPOSよりは実際の動作プログラムです。実際のプログラムではパルス発生に先立ち原点復帰をしますがこれは次の節にゆずります。コマンドSETPOSは現在位置に指定した値を与えるコマンドです。これで現在位置を0,0として原点復帰に代えます。そのあとは10000パルスの位置までステップモータを進め、タイマー1秒後にまた0の位置までもどります。このプログラムを実行するとACCELの処で時間待ちが発生します。これはACCELコマンドが加減速テーブルを内部計算で作成するために発生するロスタイムです。ACCEL 8000ではさほどの時間とはなりませんが50kppsぐらいになると無視できなくなります。この時間は加減速の距離が長ければ長いほど多くなります。なおプログラム中でパルス発生のスピードを変更するにはFEEDコマンドを使用します。ACCELで設定された範囲内で15段階でスピード変更できます(FEED 0～FEED 15)。前記の実験をする前にMPGとドライバの接続方法は次の様になります。ここではフォトカプラ入力のものを想定しています。TTLレベル入力や差動入力の場合はインターフェースが用意されていますので第2章「PIF-422」を参照して下さい。

尚、MPGを使用した場合の原点復帰の入力ポートはMPG-303上のJ2です。OVRUNの入力ポートはMIF-816上の入力ポート24～31となります。



P版でのパルス発生　(MIF-816　J5から出力)

PG -1とするとMIF-816　J5からパルスを出力します。小規模システム向きです。MPG-303のパルス発生に準拠したコマンド体系ですが、下記の相違点があります。

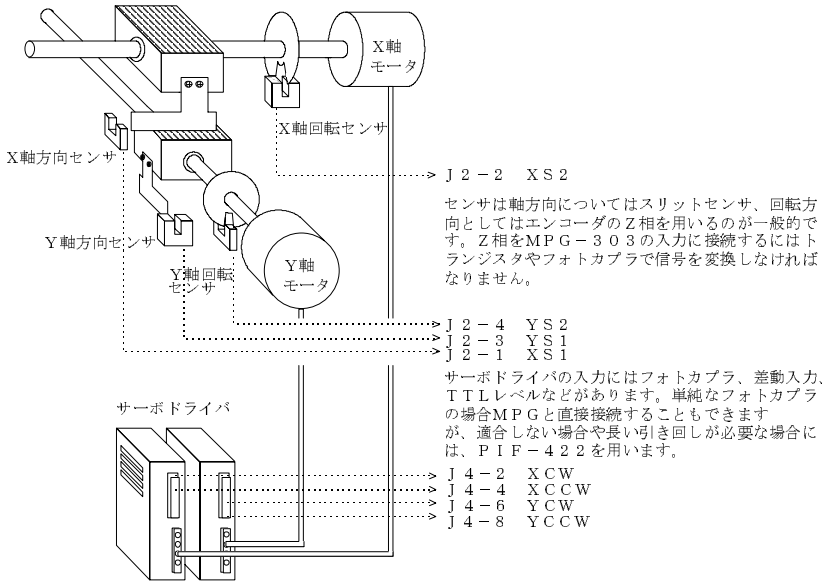
・PG　-1はCTRL＋Aでリセットされます。プログラム先頭に記述してください。
・どのタスクからでもパルス発生が可能ですが、その間はマルチタスクが停止します。
・原点センサーポートはMIF-816の入力SW(16)～(23)を用います。
・STOPコマンド、MODEコマンドは無効です
・途中停止はOVRUNだけです。OVRUNセンサーポートはSE(24)～(31)を用います。
・最高周波数は約60kppsです。
・「SHOM x,y,s」で　s（原点復帰スピード）はPPS単位で設定します。(1～200pps)
・「FEED n」　のnは0～63です。MAX60kでほぼ1kpps単位で指定できます。
・JOG，PULSEコマンドは使えません。

Z版でのパルス発生

従来MIF-816 J5からのパルス発生はZ版で行いましたが、前記のPG -1サポートによりZ版は保守版となりました。新規作成はP版で使用してください。

パルス発生2 （XYステージ・原点復帰・ティーチング・パレタイズ）

パルス発生を用いた応用の中で、比較的難しくなるのがXYステージの制御です。これはXYステージではロボット的な動作を要求されることが多くなるためです。その一番の応用例はパレタイズです。ここでは次の様な簡単な応用例にもとづいて解説します。



原点復帰について

原点復帰は通常 2 つのセンサによって実施します。1 つは実際のテーブルの移動方向に取り付けられたスリットセンサ、もう1つは回転軸に取り付けられたスリットセンサです。サーボモータの場合はZ相検出を用いるかサーボドライバ自体に備えられた原点検出機能を使用します。MPCで原点復帰する場合には次の2つのコマンドが重要です。

```
SHOM 2 8 10
HOME &HA 100 100
```

SHOMコマンドは原点復帰のパルス発生の方向とパルス発生のスピードを定めます。前記の様な設定例ではXY軸ともにCCW方向に原点復帰パルス出力します。逆にCWで原点復帰する場合は

```
SHOM 1 4 10
```

となります。SHOMの最後の引き数は原点復帰パルスのパルスレートで、ACCELで設定されたパルスレートとは独立して設定します。MODE5,6での原点復帰パルスレートは次の様に計算します。

$$f=8Mhz/(1841+208\times n)$$

例えば前記の様に10とすれば2040.3となり約2kHzとなります。またパルス幅はデューティ50％となっています。HOMEの最初の引き数は原点復帰入力の目的とするビットパタンを表しています。通常、遮光センサは常時ONでスリットの遮光でOFFとなります。またサーボのZ相はZ相検出でONとなります。次の表の通り、HOMEの最初の引き数は目的とするセンサパタン値（原点復帰後の最下段）を与えます。

Ｊ２	方向	センサ	原点復帰前	原点復帰後
1 XS1	Ｘ方向	遮光センサ	ON (1)	OFF (0)
2 XS2	Ｘ回転	Ｚ相	OFF (0)	ON (1)
3 YS1	Ｙ方向	遮光センサ	ON (1)	OFF (0)
4 YS2	Ｙ回転	Ｚ相	OFF (0)	ON (1)
-	-	-	&H5	&HA

尚、この値は関数ＨＰＴ(0)で直接読み取ることができます。関数ＨＰＴ()はＭＰＧ動作中には使用できません。HOMEの2番目と3番目の引き数は原点復帰に先立つ退避移動量です。この例の様にHOME&HA 100 100とすると原点復帰前にＣＷ方向に１００パルスずつ移動します。これは原点復帰時には一定の方向から原点復帰をしないと一定の精度を確保できなかったり、Z相検出（Ｃ相）が一回転ズレてしまうことがあるためです。必要の無い場合は0、0とします。又、Z相検出（Ｃ相）ではパルス幅や動作の不安定さから検出ミスすることがあります。これを防ぐのにSHRDコマンドで6～15までの値を与えます。電源投入後は4となっています。この場合の原点復帰のパルスレートは次の通りです。

f=8MHZ/(709+RX(n×52+283))

＊RはSHRDの設定値。nはSHOMの設定値

＊Z相はメーカによってＣ相と表記されることもある。

PG -1での原点復帰

PG -1では原点センサーをMIF-816の入力ポートに接続します。MPG-303 J2コネクタとの対応は次の通りです。

センサ	MPG-303 J2	MIF-816 J4
XS1	1番ピン	1番ピン(SW(16))
XS2	2番ピン	3番ピン(SW(17))
YS1	3番ピン	5番ピン(SW(18))
YS2	4番ピン	7番ピン(SW(19))
US1	5番ピン	9番ピン(SW(20))
US2	6番ピン	11番ピン(SW(21))
ZS1	7番ピン	13番ピン(SW(22))
ZS2	8番ピン	15番ピン(SW(23))

SHOMの原点復帰方向指定、HOMEのセンサーパターン指定はMPG-303と同様です。

ティーチング

点データにはあらかじめ設定してある場合と、装置が組上がってから設定するべきデータの2通りがあります。次に点データの扱い方について例示しますがその前に点データ（ベクトル）配列P(n)と成分配列X(n), Y(n), Z(N), U(n)の関係について解説します。MPCには予約配列としてAR(n)は純粋に演算用の配列ですが、他は点データとしての意味をもっています。点データはP(n)であらわされ、その各成分はX(n), Y(n), Z(n), u(n)で表されています。点配列P(n)は通常の演算で使用することができません。受け付けるコマンドはMOVE, JMPZ, JUMP, PRINTの点データに基づく移動コマンドと表示コマンドのみです。これに対してX(n)～U(n)は予約配列として単体で演算中で使用することができます。

①点データはコマンドSETP, STPZUで設定できる。

```
>SETP 1 100 200
>STPZU 1 300 400
```

②点データは各配列成分に演算で設定できる。

```
>X(1)=100
>Y(1)=200
>Z(1)=300
>U(1)=400
```

③点データはプログラム中で設定できる。

```
>10 SETP 1 100 200
20 STPZU 1 300 400
```

④点データはプログラム中で変更できる。

```
>10 X(1)=X(1)+D0
20 Y(1)=Y(1)+D1
```

点データはこのようにして様々な方法で設定・変更できます。またこの使用方法は様々です。次の3つの例はいずれも同等の意味を持ちます。

```
① MOVE P(1)

② MOVE X(1) Y(1)

③ A0=X(1)
   A1=Y(1)
   MOVE A0 A1
```

もう1つの点データの設定方法は、ティーチコマンドによるものです。ティーチコマンドはインチング動作(JOG動作)により機械体を目的の位置まで移動させ、その点を教示するものです。ティーチモードに移るにはT<ENTER>とします。単純な初期化が終わった時点では、次の様にデタラメな値が表示されますので所定の初期化を実施します。ティーチモードに移って最初に押すキーは0～3のいずれかです。このキーによってインチング量を選択します。実際に移動させるコマンドは、XでCW, xでCCWです。これはX軸の場合で、他の軸の場合はY, Z, Uキーです。目的の場所への移動が終了したら、Pを押します。すると改行して番号の入力を促してきます。ここで指定された番号のP(n)に現在位置が登録されます。次の例では点1と点3にデータを設定し、ティーチモード終了後PRコマンドで表示しています。尚、ティーチモードの終了はQです。

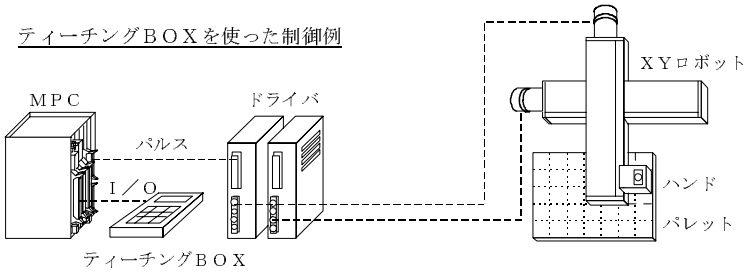
```
>MPC|N|T
PG 2#(X,Y,Z,U) 2048 28691 2048 28691 [XYZ,U] 400 400 (ここでT<ENTER>を実行しています)
>PG 1
>MODE 5
>ACCEL 15000
>SETPOS 0 0
>STPZU 0 0 0
PG 1#1(X,Y,Z,U) 0 0 0 0 [XYZ,U] 200 200  (ここでもT<ENTER>を実行しています)
>
PG 1#1(X,Y,Z,U) 400 0 0 0 [XYZ,U] 200 200
P1
PG 1#1(X,Y,Z,U) 800 0 0 0 [XYZ,U] 200 200
P3
PG 1#1(X,Y,Z,U) 800 0 0 0 [XYZ,U] 200 200
>PR P(1) P(3)
 400 0 0 0 800 0 0 0
```

主なティーチモードでの操作キャラクタは次の通りです。　　参照コマンド：T

X x	X軸のインチング移動（小文字でマイナス方向）
Y y	Y軸のインチング移動（小文字でマイナス方向）
Z z	Z軸のインチング移動（小文字でマイナス方向）
U u	U軸のインチング移動（小文字でマイナス方向）
0 1 2 3	インチング量の選択
P	点データの設定
Q	ティーチモードの終了

ティーチングプログラム

装置にティーチングタブレットを装備してティーチング機能を備えることは多くあります。I/Oに次の図の様にスイッチボックスを接続して XY ステージなどをならい教示させるものです。このティーチングにはJOG命令を使う方法とRMOVによって構成する場合の2通りがあります。



RMOVを使用する場合

RMOVは現在位置よりの相対移動です。スイッチ入力によって、RMOVを実行しインチング動作させます。点の教示ではSETPコマンドを使用します。次のプログラムはX軸1軸の場合の例です。インチング量を変更する場合はRMOVの引き数を変数として、変数をデジスイッチで設定するようにします。また、点番号の設定も同様な方法をとるものとします。

```
1000 IF SW(0)=1 GOSUB 1100
1010 IF SW(1)=1 GOSUB 1200
1020 IF SW(2)=1 GOSUB 1300
1030 GOTO 1000
1100 RMOV 10,0
1110 RETURN
1200 RMOV -10,0
1210 RETURN
1300 SETP 1,X(0),Y(0)
1310 RETURN
```

JOG命令を使用する場合

MPGでのJOG機能はJOGコマンドとXRANG～URANGまでの領域設定コマンドより成立しています。JOGコマンドはSTOPコマンドで停止となります。次にX1軸のみでJOGコマンドを使用した例を示します。

```
910 XRANG 10000,-10000
1000 IF SW(0)=1 GOSUB 1100
1010 IF SW(1)=1 GOSUB 1200
       |
1100 JOG 100,1
1110 WAIT SW(0)=0
1120 STOP 1
1125 TIME 5
1130 RETURN
1200 JOG 100,2
1210 WAIT SW(1)=0
1220 STOP 1
1225 TIME 5
1230 RETURN
```

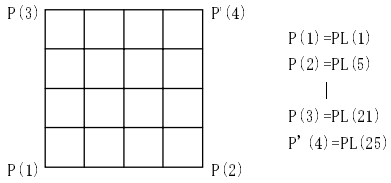
点の教示などは先の例と同様なので省略してあります。動作範囲の設定はXRANG(YRANG, URANG, ZRANG)で定めておきます。また、STOP後のTIME 5はMPG待ちです。

※これは標準入力を使用したティーチングのためのプログラム例です。
純正ティーチングペンダント「MPC-10KEY」はMPCのプログラムポートに接続してダイレクトコマンドで制御するため入出力やティーチングのためのプログラミングは必要ありません。

パレタイズについて

点データの中には、パレットのように大量の点を必要とするものがあります。これを前記の様に点データの教示によって実現しようとするとたいへんな労力となります。パレタイズコマンドはこのような規則的な点データを少ない基本的な点データから生成するものです。次のプログラム例では点1、2、3を基本点とする5*5のパレットの点を定義し、データは関数PL(n)で取り出しています。

```
100 PALET 1 2 3
110 MTRX 5 5
120 FOR I=1 TO 25
130 MOVE PL(I)
140 NEXT I
```



パレットはP版の場合同時に2つ設定することができます。コマンドはPALET1とMTRX1です。対応するパレット関数はPL1(n)です。さらに点データと同じようにパレット点はX、Yの各成分で得ることができるように関数PLX(), PLY()が用意されています。尚、パレット点PL(n)はX,Y,Z,Uの4成分を含んでいますがパレタイズされているのはX,Yの2次元のみでU,Z成分についてはPALETコマンドの最初の点の座標値にならいます。前記の例では、P(1)のZ,U値が各PL1(n)のU,Z成分となります。

実際のプログラム

ずいぶんと前置きが長くなりましたが、これで実際の応用の準備ができました。次にMPG-303でのモデルプログラムを例示します。自動、ティーチング、原点復帰を単純なモデルで表現しています。流れと基本的なコマンドをつかんで下さい。

```
#DEFS START 1
#DEFS HOMES 2          メイン操作スイッチ
#DEFS TEACH 4
'
#DEFO READY 1
#DEFO ERR 2
'
#DEFS SNS1 5   <------ インタロック用センサ
#DEFS XCW 6
#DEFS YCW 7            各軸のＪＯＧキー
#DEFS XCCW 8
#DEFS YCCW 9
#DEFS PENT 10  <------ 現在位置記憶スイッチ
#DEFS OP1 11   <------ パレット交換完了スイッチ
#DEFO PALET 5  <------ パレット交換表示
'************
'INIT
'************
PG 1                   ＭＰＧ－３０３初期化
MODE 5                 この例のようにＰＧ選択、モード設定、ＯＶＲＵＮ、
OVRUN 0                Ｄ４５等を初期してからＡＣＣＥＬを実行する
D45 0
ACCEL 15000
PALET 11,12,13         パレット宣言点１１，１２，１３を使って５×５のパレット構成
MTRX 5,5
'************
*MAIN
'************
IF SW(START)=1 GOSUB *START
IF SW(HOMES)=1 GOSUB *HOME     メインプログラム
IF SW(TEACH)=1 GOSUB *TEACH    自動、原点復帰、ティーチングの動作を分類する
GOTO *MAIN
'************
*START
'************
MOVE P(1)
ON PALET       <------ 出力パレットのＯＮはオペレータにパレット交換を促す
WAIT SW(OP1)=1 <------ ＯＰ１はオペレータスイッチで作業完了を意味する
OFF PALET
J=0
```

```
*LOOP
 WAIT SW(SYS1)=1
 MOVE PL(J)
 GOSUB *PLACE
 IF J=25 THEN *END
 GOTO *LOOP
*PICK
 'PICK MOTION
 RETURN
*PLACE
 'PLACE MOTION
 RETURN
*END
 RETURN
 '************
*HOME
 '************
 SHOM 2,8,10
 HOME &H000A,100,100
 RETURN
 '************
*TEACH
 '************
 IF SW(XCW)=1 THEN *XMV
 IF SW(YCW)=1 THEN *YMV
 IF SW(XCCW)=1 THEN *XMVN
 IF SW(YCCW)=1 THEN *YMVN
 IF SW(PENT)=1 THEN *PENT
 GOTO *TEACH
*XMV
 JOG 100,1
 WAIT SW(XCW)=0
 GOTO *MV_END
*XMVN
 JOG 100,2
 WAIT SW(XCCW)=0
 GOTO *MV_END
*YMV
 JOG 100,3
 WAIT SW(YCW)=0
 GOTO *MV_END
*YMVN
 JOG 100,4
 WAIT SW(YCCW)=0
*MV_END
 STOP 1
 TIME 10
 GOTO *TEACH
*PENT
 A=IN(2)^&H000F
 A1=IN(2)/16
 A1=A1*10
 A=A+A1
 SETP A,X(0),Y(0)
 RETURN
```

パレットとパルス発生の組み合わせ例
移動スピードを変更する場合は’ＦＥＥＤ’を使用します

チャッキング動作

プレース動作

原点復帰動作
サーボのＺ相を読み取る場合は他にＳＨＲＤを必要とします
（ＳＨＲＤ　６～ＳＨＲＤ　１５）

ティーチングスイッチによる分岐

各軸のＪＯＧコマンドと停止
停止はスイッチを離す（開放）ことによって行っています。
スピードを変更したい場合はＪＯＧの１００を変更します。

ＪＯＧコマンドはＳＴＯＰによって停止します
停止後０．１秒程のタイマを入れます

ティーチング点はデジスイッチで指定することを前提としています
入力１６～２３にＤＳＷが２ヶあると想定しています

パルス発生3

パルス発生の作業の中で最もやっかいなのがパルス発生中の途中停止です。パルス発生はI/O制御と異なり停止が必要となった時点でどのような状態であるか、あるいはどのように停止したいかによって処理が変わります。残念ながらMPC-816での途中停止は仕様追加によって構成されているためわかりにくいものであったり、コマンドが対称的にサポートされていなかったりします。この節では代表的なパルス停止の方法を紹介します。

パルス停止の注意事項

QUITの禁止

マルチタスクの使用方法に習熟してくると、ある作業の停止にはQUIT が便利だということがわかりますが、これは、パルス発生の場合にはあてはまりません。パルス発生中のタスクに対して不用意にQUITするとそれ以降MPGに対してのコマンドが無効になることがあります。

悪い例

```
*LOOP
 FORK 1 *PG
 TIME 100
 QUIT 1
 TIME 100
 GOTO *LOOP
*PG
 RMOV 1000 1000
 TIME 50
 RMOV -1000 -1000
 TIME 50
 GOTO *PG
```

良い例

```
 *LOOP
  FORK 1 *PG
  TIME 100
  STOP 4
  WAIT BSY(1)<>0
  QUIT 1
  STOP 5
  GOTO *LOOP
 *PG
  RMOV 1000 1000
  TIME 50
  RMOV -1000 -1000
  TIME 50
  GOTO *PG
```

STOP後の処理 （STOPさせても次の命令は有効）

MPGにSTOPをかけてもインタプリタの停止が止まるわけではありません。パルス発生中のコマンドが停止させられるだけです。このため、STOPによって完全に動作を中断させるには次の様な工夫が必要です。

悪い例

```
    FORK 1 *MPG
    WAIT SW(1)=1
    STOP 1
    END
*MPG
    RMOV 100 0
    GOTO *MPG
```

良い例

```
 FORK 1 *MPG
 WAIT SW(1)=1
 STOP 1
 END
*MPG
  RMOV 100 0
  WAIT BSY(0)=1
  GOTO *MPG
```

悪い例ではRMOV 100 0が1回停止させられてもすぐ次のコマンドが実行されて、再びRMOV 100,0が実行されます。良い例ではRMOVごとにWAIT BSY(0)=1のチェックをしており1度STOPで停止させられるとここで条件待ちとなります。

JOG, STOP後の現在位置読み取り

ティーチング用のプログラムを作る場合にJOGとSTOPを組み合わせますが、STOP実行後ただちに現在位置を読み取る関数を実行すると、STOPが無効になります。次に1軸のティーチングプログラム例を示します。STOP後に0.1秒のタイマを入れています。これはSTOPの実行とMPGの停止の間には時差があり、この間にMPGに対して指示を与えると誤動作してしまうことがあるためです。

```
*TEACH
 IF SW(1)=1 THEN *XCW
 IF SW(2)=1 THEN *XCCW
 IF SW(3)=1 THEN *END
 GOTO *TEACH
*XCW
```

```
 JOG 100,1
 WAIT SW(1)=0
 STOP 1
 TIME 10                    0.1秒タイマ
 X=X(0)
 GOTO TEACH
*XCCW
 JOG 100,2
 WAIT SW(2)=0
 STOP 1
 TIME 10                    0.1秒タイマ
 X=X(0)
 GOTO *TEACH
```

このことは通常のRMOVやMOVEでも同様です。次の例のようにSTOP停止をかけたあとでMPGの実際の停止をBSY()関数によって監視する必要があります。

悪い例

```
 PG 1
 FORK 1,*PG
 TIME 100
 STOP 1
 X1=X(0)
 END
*PG
 MOVE 10000,0
 END
```

良い例

```
 PG 1
 FORK 1,*PG
 TIME 100
 STOP 1
 WAIT BSY(1)<>0
 X1=X(0)
 END
*PG
 MOVE 10000,0
 END
```

複数のタスクからのコマンド

プログラムの設計時には各タスクの用途は整理されて、パルス発生用、I/O制御用と分類されていますが最終的な段階ではこの秩序が保ちきれなくなることがあります。例えば、前記の例にあったように他のタスクが停止コマンドを実行するために座標読み取りは他のタスクが実施します。又、複数のユニットがからみあった位置決めでは複数のタスクから同一のＭＰＧに対して指示を与えたくなることがあります。こうした場合には、交通整理をきちんとしなければなりません。セマフォを使う場合（複数のタスクが同じMPGにコマンドを出力）次の様にセマフォ関数を使えば複数のタスクがコマンドを出しても問題は発生しません。

①セマフォの場合

```
*TASK1
 WAIT RSV(-1)=0
  MOVE 1000,0
 OFF -1
 GOTO *TASK1
```

```
*TASK2
 WAIT RSV(-1)=0
  RMOV -1000,0
 OFF -1
 GOTO *TASK2
```

②パルス発生を１つのタスクにまとめた場合

```
*TASK1
  A=1
  WAIT A=0
  GOTO *TASK1
```

```
*TASK2
 B=1
 WAIT B=0
 GOTO *TASK2
```

```
*PG
  IF A=1 THEN *MV
  IF B=1 THEN *RV
  GOTO *PG
*MV
  MOVE 1000,0
  A=0
  GOTO *PG
*RV
  RMOV -1000,0
  B=0
  GOTO *PG
```

STOP コマンド

STOPコマンドは1～5までの引き数があり次の様に意味が異なります。

STOP 1	減速停止	STOP後減速しその後停止します。
STOP 2	急停止	STPO後ただちにパルス発生中止します。
STOP 3		次に続くパルス発生コマンドに対してのみ有効なI/O監視停止命令です。

```
STOP 3,10,1
MOVE 10000,0
```

この例ではポート10がONになったら次のMOVEコマンドで減速停止します。

STOP 4	MPGコマンド受付を一時停止します。
STOP 5	STOP4による一時停止を解除します。

パルス発生を停止するコマンドは、この他にOVRUNがあります。OVRUNは常時入力ポートを監視して条件が成立するとパルス発生を中止し、インタプリタの実行を停止します。このためエラー回復が必要な場合はSTOP 1,STOP 2を監視タスクから実行します。又、STOP 4,STOP 5はパルス発生コマンドそのものを停止することができるのでこれを組み合わせると非常停止は簡単に実現することができます。

```
      FORK 1 *PG
*MAIN
      IF SW(1)=1 THEN *EMG
      IF SW(2)=1 THEN *JOB1
         |  その他の処理
      GOTO *MAIN
*EMG
      STOP 4              ＭＰＧへのコマンド実行停止設定
      STOP 1              ＭＰＧのパルス発生中止
      WAIT BSY(1)<>0      停止確認
         |
         |  後処理
      QUIT 1              タスクの停止
      STOP 5              ＭＰＧ停止の解除
      GOTO *EMG
*PG
      MOVE 1000 0
      WAIT SW(3)=1
      MOVE 0 0
      WAIT SW(4)=1
      GOTO *PG
```

もしSTOP 4を先に実行しておかないと、STOP 1はその時実行されているパルス発生コマンドのみに有効ですからパルス発生中止後すぐに次のコマンドを実行してパルス発生してしまいます。もう一つの方法は、各移動コマンドにWAIT BSY(0)=1を組み込んでおきます。

```
   FORK 1 *PG
   TIME
   STOP 1
   WAIT BSY(1)<>0
   QUIT 1
   END
*PG
   RMOV 1000,0
   WAIT BSY(1)=1          ＳＴＯＰで停止すると次へは進まない
   GOTO *PG
```

ＰＧ　-1 でのパルス停止

PG -1ではSTOPコマンドが使用できません。パルス発生前にOVRUNコマンドでMIF-816の入力(SW(24)～(31))のパターンを指定します。次は指定例とサンプルプログラムです。

OVRUN指定例

```
OVRUN &HFF              ″SW(24)～(31)のどれかがONで停止
OVRUN &H01              ″SW(24)がONで停止。使わないポートは一般入力OK
OVRUN &HFFFF            ″SW(24)～(31)のどれかがOFFで停止
OVRUN &HFF01            ″SW(24)がOFFで停止。使わないポートは一般入力OK
```

サンプルプログラム

```
ACCEL 3000
FEED 0
FEDZ 0
OVRUN &HFF                      ″SW(24)～(31)のどれかがONで途中停止
MOVE 10000 10000                ″パルス発生
IF IN(3)◇0 THEN *OVRUNERR       ″停止原因確認。OVRUNならエラー
```

MPG-301の使用例

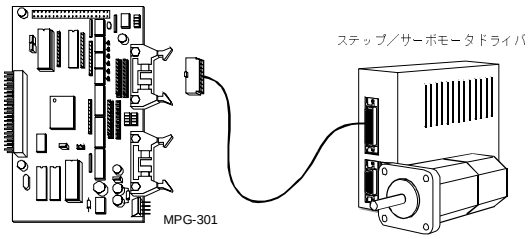
MPG-301は汎用のパルス発生IC:X3202を搭載しており、次のような用途に対応できます。

①50kppsを超える高速パルス発生

②パルス発生中での速度や加減即の変更、S字加減速などを必要とする用途

③エンコーダカウンタの入力

MPG-301には別途、詳細マニュアル（当社Webサイト、製品別マニュアル）が用意されていますので、使用例についてはこちらを参照してください。このマニュアルでは簡単な例を一つ紹介します。接続は、MPG-301のパルス出力をサーボモータあるいはステップモータのパルス入力に接続するだけです（MPG-301の出力は差動出力ドライバです）。プログラム例ではパルスを連続発生させながら、入力0から7の値にしたがって速度を変更し、IN8をONにすると停止するというものです。この例で見るようにMPG-301はリアルタイムで内部レジスタの参照・変更が可能です。



MPG-301 J4	ドライバ
1 (CW+)	Driver 入力 CW+
2 (CW-)	Driver 入力 CW-
3 (CCW+)	Driver 入力 CCW+
4 (CCW-)	Driver 入力 CCW-

※接続にはツウィストペアを推奨します。また、100ｋ以上の高速パルス発生には、高速信号伝達に配慮されたケーブルをご使用ください。

```
"SW(8) ONで停止
'"NO-IN7 DSWの読み取り
ST_REG 0,250                "周波数倍率250の時倍率は１／１
ST_REG 3,100                "起動周波数100pps
ST_REG 4,1000               "最高周波数1000pps
ST_REG 5,100                "加速ﾚｰﾄ100pps2
ST_REG 6,100                "減速ﾚｰﾄ
CMND &HA0                   "動作完了ﾌﾗｸﾞﾘｾｯﾄ
CMND &H06                   "連続駆動
IF SW(8)=1 GOTO *STOP
R4=IN(0)
R4=R4*20                    "読み取り値の20倍の値を速度とする。
ST_REG 4,R4                 "周波数変更
GOTO *LOOP
*STOP
CMND &H31                   "減速停止命令
WAIT REG(-1)=&H20           "動作完了待ち
WAIT SW(8)=0
GOTO *MAIN
```

MPC 間の通信

MPC間で通信をする場合は次の3つのコマンドで充分です。　参照：5章コマンドリファレンス
CNFG#, INPUT#, PRINT#

```
MPC-1                          MPC-2

10 CNFG# 2 0 2                 10 CNFG# 2 0 2
20 ON 1                        20 INPUT# A
30 PRINT# 1                    30 ON 1
40 INPUT# A                    40 TIME 100
50 IF A=0 THEN *ERROR          50 IF SW(1)=0 THEN *ACK
60 OFF 1                       60 OFF 1
70 GOTO 20                     70 PRINT# 1
1000 *ERROR                    80 GOTO 20
1010   ・                      1000 *ACK
       ・                      1010 PRINT# 0
```

この例ではMPC-1がホストとなり30で数値1を送出しています。これに対応するのはMPC-2のINPUT#でMPC-1から与えられたメッセージを受け取ると次のステップ進みます。MPC-1は、MPC-2に指示を送出すると40のINPUT#でMPC-2からの信号を待ちます。ここで注意するのは、返ってくる数値には0と1の2種類があり、それによってMPC-1の継続作業も変更されます。このように通信によるインタロックでは、情報が数値として扱えるためインタロックを高度なものとすることができます。又、通信を受け取る側でパルス発生器として使用する場合は次の様になります。

```
MPC-1                          MPC-2

       |                        5 *LOOP
100 PRINT# X1 Y1               10 INPUT# A B
120 INPUT# A                   20 RMOV A B
130 IF A<>7 THEN *ERR          30 PRINT# 7
       |                       40 GOTO *LOOP
```

パソコンと MPC の通信（CH1 を使用した場合）

```
パソコン(N88BASIC)             MPC側

10 OPEN "COM1:E71XN" AS #1      5 CNFG# 2 0 2
           |                   10 INPUT A B
           |                   20 RMOV A B 0
100 PRINT#1,A,B                30 PRINT# 7
110 INPUT#1,C                  40 GOTO 10
           |
           |
```

注意すべきことは、通信の手順をあわせることです。N88BASICとMPCの通信コマンドは良く似ているので簡単に通信が実現できます。

複雑なプロトコルに対応

TNYFSC では文字キャラクタの操作のために次の様なコマンドや関数がサポートされています。対パソコン、対MPC等の通信の場合はこうしたコマンドを使用する必要は無いのですが、固定の通信フォーマットを備えた機器（計測器・表示器　等）に対しては、MPC側でプロトコルを整合させる必要があります。

GET#()　PUT# PUTS#　GETN#　SKIP#　FIND#

ここでは、電圧計測器から次の様なフォーマットでデータが送られてくるものとします。

[SP]X1=230.13[SP][SP]Y1=156.17[SP][CR]

この場合、通常のINPUT#文では”X1”や”=”等の文字が邪魔になり、正常な数値の取り込みができません。こうした場合、次の様なプログラムを組めばデータを取り込むことができます。

```
100 SKIP# &H3D              &H3Dは '='
110 GETN# X0                230を変数X0にとりこむ
120 SKIP# &H2E              &H2Eは '.'
130 GETN# X1                13を変数X1にとりこむ
140 SKIP# &H3D
150 GETN# Y0                156を変数Y0にとりこむ
160 SKIP# &H2E
170 GETN# Y1                17を変数Y1にとりこむ
180 SKIP# &H0D              &H0Dは 'CR'
190 X0=X0*100
200 X0=X0+X1                X0には23013がセットされる
210 Y0=Y0*100
220 Y0=Y0+Y1                Y0には15617がセットされる
```

SKIP#は相当する文字コードを検索し、GETN#は数字・文字列のみを変数に代入します。この例はピリオドを挟んで1つのデータを2つの整数データとして取り込み後で再合成しています。前記はデータの読み込みでしたが、装置によってはRS-232Cでコマンドを与えるものもあります。デジボルに対して次の文字列を出力しなければならない時は例の様にします。

```
"CH1 DAT"[CR]
```

【例１】
```
100 'CH1 DAT
110 PUTS# STR(-1)
120 PUT# &H0D
```

【例２】
```
100 PUT# &H43,&H48,&H31
110 PUT# &H20,&H44,&H41
120 PUT# &H54,&H0D
```

このように文字を文字コードとして扱うことによってTNYFSC内部でデータを処理できるようにしています。通信プログラムで他に有効なコマンドとして覚えておきたいのは、関数RS(n)とTST#(0)です。RS(n)はコマンドで処理されていないが、バッファにたまっている受信キャラクタの数を知らせます。TST#(0)はCH1だけに限られますがデータを読み出さずに受け取っているキャラクタを分類します。INPUT#やGET#(0)はキャラクタを受け取っていなくてもキャラクタ待ちの状態となるため実行効率が悪くなります。RS(n) やTST#(0)で受信状態を確認の上プログラムを進めるとより効率の良いプログラムを記述できます。

パソコンと MPC の通信 （CH 0 を使用する場合）

ここでは、プログラム用CHをパソコンに接続して使用する方法について述べます。次の様な簡単な例に基づき専用プログラムをつくることにします。プログラムは、与えられたデータに基づいてポートをON/OFFします。

```
10 'SAMPLE
20 INPUT B
30 QUIT 1
40 OFF A
50 A=B
60 FORK 1,1000   ---------->
70 GOTO 20       これに対応する
1000 ON A        プログラムは次
1010 TIME 20     の様になります
1020 OFF A
1030 TIME 20
1040 GOTO 1000
```

```
10 'INITIAL
20 OPEN "COM1:E71XN" AS #1
30 PRINT#1,CHR$(&H1);
40 IF INPUT$(1,#1)=">" THEN #START
50 GOTO 40
60 '
70 *START
80 PRINT#1,"RUN";CHR$(&HD);
90 '
100 'MAIL LOOP
110 IF INPUT$(1,#1)="?" THEN *RSPNS ELSE 110
120 *RSPNS0
130 INPUT "PLEASE SET DATA PORT NUMBER ",A
140 IF A>B THEN *RESET
150 PRINT#1,A;CHR$(&HD);
160 GOTO 110
170 '
180 *RESET
190  INPUT "CAN I RESET FASIC !!(Y/N)",C$
200 IF C$="Y" THEN 30 ELSE *RSPNS
```

この例では、MPCのソフト・リセットが使用されています。01Hコードで”PRINT#1, CHR$(&H1);”によってMPCはソフト・リセットします。BASICのプログラム中では、”>”のレスポンスとINPUTの”?”をキー・キャラクタとして使用しています。エコー・バックは全て読み捨てとなります。こうしたやりとりで注意することは、MPCが割り込み禁止となっていないだろうか？ということです。割り込み禁止は、Z版でのパル

ス発生時におこり、この時MPCはただ1つのキャラクタのみ受付可能でそれ以上は送らないようにします。また、レスポンス（>）を待たずにキャラクタを送り続けるとプログラムの内容が破壊されることがあります。このように、CH 0に接続してMPCを制御することもできますがこれにはMPCのプロトコルついて詳細に知る必要があります。次に基本プロトコルについて解説します。MPCでは、次の様な基本プロトコルを持ちます。例外は、コマンド自身が通信出力を持つ場合、<CTRL>+<A>キーによるソフト・リセットの場合、さらにイニシャルのメッセージ出力等の場合です。例外コマンドは、TEACH, PRINT, INPUT等のデータ入出力コマンドに限られます。ここではまず、”ON 1”を例としてプロトコルを詳説します。

```
《TERMINAL》          《MPC》
 0                    0     [echo] echo: ｴｺｰﾊﾞｯｸ
 N                    N     [echo]   sp: ｽﾍﾟｰｽ
[sp]                 [sp]   [echo]   cr: ﾘﾀｰﾝ
 1                    1     [echo]   lf: ﾗｲﾝ･ﾌｨｰﾄを意味します。
[cr]                 [cr]   [echo]
                     命令実行
                     [cr]
                     [lf]
                      >
```

このように命令をエコーバックし、実行終了とともに［lf］>を出力します。プログラムの入力も全く同様です。例えば、一行入力　”10 ON 1”の場合は次の様になります。

```
《TERMINAL》          《MPC》
 1                    1     [echo]
 0                    0     [echo]
[sp]                 [sp]   [echo]
 0                    0     [echo]
 N                    N     [echo]
[sp]                 [sp]   [echo]
 1                    1     [echo]
[cr]                 [cr]   [echo]
                     命令実行
                     [cr]
                     [lf]
                     >
```

この資料でプロトコルの全てを前記の様に記述するのは困難でありますので、次の様な略記法を用います。エコー部は、( )で囲まれた部分です。レスポンスは、キャラクタとヘキサで表記します。ヘキサは、［&H_］と表現します。

```
☆★☆★☆★☆　特殊プロトコル一覧　☆★☆★☆★☆
(NEW[&HD])
     [&HD][&HA]>
(10 ON 1[&HD])
     [&HD][&HA]>
(20 ON 2[&HD])
     [&HD][&HA]>
(30 TIME 100[&HD])
     [&HD][&HA]>
(LIST[&HD])
     [&HD][&HA]10[&HA]ON 1
     [&HD][&HA]20[&H9]ON 2
     [&HD][&HA]30[&H9]TIME 100
     [&HD][&HA]>
(INPUT A B[&HD])
     [&HD][&HA]?(1234 567[&HD])  [&HD][&HA]>
(PRINT A B[&HD])
     [&HD][&HA] 1234 567         [&HD][&HA]>
(TEACH[&HD])
     [&HD](X,Y,Z,U) 0 0 210 0 [XYZ,U] 100 100              (*Q)
     [&HD][&HA]>
```

実際の詳細が不明の場合はFTMWディスクに含まれるACTERM を使用して下さい。ACTERM は無手順ターミナルソフトで受け取ったコントロールコードも表示できます。

デジタル社タッチパネルとの接続

MPC-816のRS-232Cユーザーチャンネル（CH1）とデジタル社「プログラマブル表示器 GP70シリーズ」のメモリリンク通信のサンプルです。メモリリンクではMPCがホストとなり、GPメモリの読込み／書込みでパネルをコントロールします。

例1)はGPの「「割り込み出力」を用いた例です。このプログラムを実行するには、GPの画面をデザインする際に、部品の出力を「割り込み出力」として設定します。「割り込み出力」とは「ワードSW」や「Tタグ」のワードアドレスをシステムデータエリア「13」に割り当てると、それを押したときに「定数」をRS-232C出力するというものです。各ワードSWやTタグに固有の「定数」を設定しておけば、押されたものの識別が出来ます。MPCはその出力をGET#で待っているだけです。ランプ等のMPCからの出力はGPの「W」コマンドで行っています。この方法の利点は、プログラムが簡単・常時ループしない・GPのボタンを押したときのレスポンスが良い、などがあります。しかし、出力可能な定数は1バイト（&HFFは不可）なので、これを超えるデータは得られません。MPCとリンクできる部品の種類と数も限られます。簡単なパネルデザイン向きです。

例2)はパネルからの読込みをGPの「R」コマンドで行っています。書込みは例1と同じGPの「W」コマンドです。この方法の利点はワードデータが扱えることです。GPの「数値表示器」や「設定値表示器」などの16ビットデータの読込みができます。欠点は、プログラムが複雑、常にループする、一度に多くのデータを入力するとレスポンスが低下します。レスポンスについては、ページ単位で入力範囲を制限すれば良いでしょう。(同一ページの部品のアドレスは１回のREADで読める範囲に設定する）このサンプルでは1ループで4ワード(64ビット）をREADしていますが、この程度ならレスポンスの低下はあまり感じられません。

GPの通信設定

通信プロトコル：アスキー互換
通信設定：伝送速度9600bps、データ長8bit、ストップビット1、パリティ無
通信方式：RS-232C

※メモリリンクプロトコル、GP操作、GP作画についてはデジタル社の各取扱説明書をご覧下さい。
※これはRS-232C通信を説明するためのサンプルプログラムです。実際にタッチパネルを接続するには「MBK-816」を用います。

例1)

```
'************
' GP-H70
'   SAMPLE
'------------
' USE T-TAG
' & SYS Adr13
'============
'GPH70S10
'   980818
'************
CNFG# 4,0,2
FORK 1,*RS_RCV          ″入力用のタスク
*LP1
A=600                   ″書込みアドレス
S=IN(0)                 ″書込みデータ
GOSUB *WRITE
WAIT IN(0)<>S
GOTO *LP1
*WRITE
PUT# &H1B,&H57          ″ESC W
S0=A^&HF000             ″ｱﾄﾞﾚｽ(hex表記)をｱｽｷｰｺｰﾄﾞで出力
S0=S0/&H1000
GOSUB *WRITE1
S0=A^&H0F00
S0=S0/&H0100
GOSUB *WRITE1
S0=A^&HF0
S0=S0/&H10
GOSUB *WRITE1
```

```
 S0=A^&H0F
 GOSUB *WRITE1
 S0=S^&HF000                    ″ﾃﾞｰﾀをｱｽｷｰｺｰﾄﾞに変換して出力
 S0=S0/&H1000
 GOSUB *WRITE1
 S0=S^&H0F00
 S0=S0/&H0100
 GOSUB *WRITE1
 S0=S^&HF0
 S0=S0/&H10
 GOSUB *WRITE1
 S0=S^&H0F
 GOSUB *WRITE1
 PUT# &H0D
 RETURN
*WRITE1
 IF S0>9 GOSUB *A_F
 ELSE_GOSUB *0_9
 PUT# S0
 RETURN
*A_F
 S0=&H37+S0
 RETURN
*0_9
 S0=&H30+S0
 RETURN
 '************
*RS_RCV
 R=GET#(0)                      ″入力待ち
 PRINT R
 OUT R,0
 GOTO *RS_RCV

例２）
'************
 '  GP-H70
 '    SAMPLE
 '------------
 'USE GP
 '  READ/WRITE
 '     COMMAND
 '============
 'GPH70S20
 '   980818
 '************
 S=0
 CNFG# 4,0,2
*LP1                            ″READ,WRITEとも同一タスク
 A=600                          ″書込みアドレス
 S=IN(0)                        ″書込みデータ
 GOSUB *WRITE
 '
 FOR A=500 TO 503               ″A=読込みアドレス
 GOSUB *READ
 NEXT A
 '
 GOTO *LP1
*WRITE
 PUT# &H1B,&H57
 S0=A^&HF000
 S0=S0/&H1000
 GOSUB *WRITE1
 S0=A^&H0F00
 S0=S0/&H0100
 GOSUB *WRITE1
 S0=A^&HF0
 S0=S0/&H10
 GOSUB *WRITE1
```

```
 S0=A^&H0F
 GOSUB *WRITE1
 S0=S^&HF000
 S0=S0/&H1000
 GOSUB *WRITE1
 S0=S^&H0F00
 S0=S0/&H0100
 GOSUB *WRITE1
 S0=S^&HF0
 S0=S0/&H10
 GOSUB *WRITE1
 S0=S^&H0F
 GOSUB *WRITE1
 PUT# &H0D
 RETURN
*WRITE1
 IF S0>9 GOSUB *A_F
 ELSE_GOSUB *0_9
 PUT# S0
 RETURN
*A_F
 S0=&H37+S0
 RETURN
*0_9
 S0=&H30+S0
 RETURN
 '************
*READ
 PUT# &H1B,&H52            "ESC R
 S0=A^&HF000               "ｱﾄﾞﾚｽ(hex表記)をｱｽｷｰｺｰﾄﾞで出力
 S0=S0/&H1000
 GOSUB *READ1
 S0=A^&H0F00
 S0=S0/&H0100
 GOSUB *READ1
 S0=A^&HF0
 S0=S0/&H10
 GOSUB *READ1
 S0=A^&H0F
 GOSUB *READ1
 PUT# &H30,&H30,&H30
 PUTS# 1                   "読込みデータ数＝１
 PUT# &H0D
 '
 SKIP# &H41                "Aまで読み飛ばし
 INPUT# D                  "CRまで入力
 0=A-500
 OUT D,0
 RETURN
*READ1
 IF S0>9 GOSUB *A_F
 ELSE_GOSUB *0_9
 PUT# S0
 RETURN
```

MBK-816 の使用方法

前節ではタッチパネルインターフェースをRS-232Cで実現しましたが、複雑な用途にはプログラムが煩雑になり保守しにくいものになります。本格的なタッチパネルの応用にはMBK-816が適しています。ここでは簡単にMBK-816の紹介をします。（詳しくは当社Webサイト、製品別マニュアルを参照ください。）

MBK-816を用いると次のようなメリットがあります。

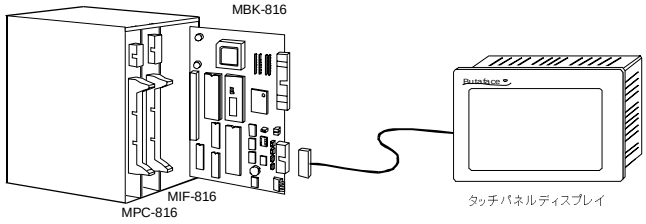
①RS-422接続となるため、信頼性が向上し、通信速度も速くなる。

②インターフェースコマンドがON/OFF/SW()/OUT/IN()などのI/Oコマンドになり扱いやすくなる

注意として次の事柄があります。

①MBK-816の消費電流が大きいので設計時に注意

②IF SW(5000)=1 THEN ~という記述が出来ない。A0=5000，IF SW(A0)=1 THEN ~とする。



出力トランジスタの破損

MPCのトラブルで最も多いのが出力トランジスタの破損です。これは通電したまま配線チェックをしたり、配線チェックしないままパワーオンした場合に発生しています。いずれにしても初歩的なミスです。十分に注意して配線は確かめて下さい。修理はトランジスタ交換となります。

バッテリバックアップ関係のトラブル

MPC-816のRAMは、リティウム電池でバックアップされています。これにより設定された変数、ポイントデータが無通電中も保持されます。しかし、バッテリバックアップはデリケートな保持方法のため相応の注意が必要となります。

組立時の注意

MPC-816は無通電中にもリティウム電池によりSRAMが活かされています。このためMPC-816を鉄やアルミの板に直接置いたりアルミ箔に包んだりすると電池放電やSRAMのデータが破壊されます。又、静電気の放電を受けることによってもデータが変わってしまいます。組立時には前記を考慮の上扱うことと、組立後はMPCINITの初期化を行うことが安全で確実です。

パソコンと接続した時

装置に組み込まれたMPCと外部のパソコンを接続するときにMPCのRS-232Cポートに電気的なショックが発生します。特に、200V系の装置ではこの度合が大きく甚だしい場合はインターフェースのICを壊してしまいます。これは、パソコンと装置のアースを共通にすることにより回避できますが、現場ではなかなか難しいことです。MPC-816では、RS-232Cインターフェース回路をフォトカプラで絶縁分離して外乱から内部回路を保護しています。

プログラム上の問題

MPC-816に登載されたTNYFSCは1989年以来フィールドで実用に供されているソフトです。このため、バグは相当減らされていますが、仕様が古いためにいくつか概念上の誤解をまねいたり用法が不明確であったりする場合があります。

FORKについて

FORKはマルチタスクを発生させるコマンドですが、ふだん普通のプログラム（パソコン等）ではお目にかかれません。このため、性質がつかみにくかったり、初歩的な用法の誤りを招くことがあります。例えば、次の例です。

```
5 *LOOP
10  IF SW(1)=0 THEN *NEXT
20  FORK 1 *SUB
30  *NEXT
40  GOTO *LOOP
50  *SUB
60  ON 0
70  TIME 10
80  OFF 0
90  END
```

前記の例ではSW(1)の値が1になった時にプログラム*SUBが動作することを期待しているのですが、これでは*SUBプログラムが動作しません。これはFORKが実行された時点でプログラム*SUBを実行し始めるという意味になるためです。この例では、SW(1)が1になっている限り何度もFORKが実行されその度に*SUBから再実行し始めます。つまり、プログラムが進行する前に再起動かけられてしまいます。正しくは次の様にします。

```
5 *LOOP
10 IF SW(1)=0 THEN *NEXT
12 FORK 1 *SUB
14 WAIT SW(1)=0
16 QUIT 1
30 *NEXT
40 GOTO *LOOP
```

反応が遅いと思われる場合

多くの入力で制御を分類する場合に、IF文とSW(n)関数を組み合わせて何行も記述すると反応が遅くなる場合があります。

```
5 *LOOP
10 IF SW(1)=1 THEN *WORK1
20 IF SW(2)=1 THEN *WORK2
30 IF SW(3)=1 THEN *WORK3
        (中略)
200 IF SW(20)=1 THEN *WORK20
210 GOTO *LOOP
```

このような場合は、IN(n)で入力関数を減らした方が有利です。

```
5 *LOOP
10 A=IN(0)~&HFE
20 IF A=2 THEN *WORK1
30 IF A=4 THEN *WORK2
40 IF A=8 THEN *WORK3
       |
50 IF A=128 THEN *WORK7
60 A=IN(1)
       |
```

HSW(n)を使用した場合の注意

先の節でとりあげたSW(n)はノイズ対策のための2度読み機能が組み込まれています。これに対して2度読みをしないのがHSW(n)等Hの付された関数です。このためこの関数を使えばプログラムがシングルタスクである限り高速反応するプログラムとなります。問題はこれをマルチタスクで使用すると逆効果となることです。

```
10 FORK 1 *PRG1
20 FORK 2 *PRG2
30 *LOOP
40 IF A=1 THEN *SUB1
50 IF B=1 THEN *SUB2
       |
100 GOTO *LOOP
200 *PRG1
210   A=HSW(1)
220   GOTO *PRG1
230 *PRG2
240   B=HSW(2)
250   GOTO *PRG2
```

このプログラムでは、*PRG1と*PRG2の中でHSW(1)とHSW(2)を使うことによって高速を期待していますが逆効果となります。結果としてはSW(1), SW(2)とした方が総合的には高速となります。これは、マルチタスクにおける時間資源の有効活用という難しい内容になります。とりあえずここでは、1つのタスクでHSW()等を使用する場合にのみ高速応答が期待できるということを御了解下さい。

エラー停止

TNYFSCはエラー発生と同時にインタプリタを停止して文番号とメッセージを表示しますがこの文番号はメインタスクの文番号となっています。これに対してエラーメッセージは発生したトラブルを表現していますのでマルチタスク下では読み方に注意が必要となります。

```
>LIST
10 FORK 1,*TASK2
20 FOR I=1 TO 10
30 ON I
40 TIME 5
50 OFF I
60 NEXTI
70 GOTO 20
80 *TASK2
90 PG 3
```

```
100 RMOV 1000,0
110 GOTO 100
>RUN
#40
!!Out of Range
>
```

前記の例ではあたかも#40で”Out ogf Range”となってしまったように見えますが、本当の問題は90のPG3で発生しています。マルチタスクでエラー停止した時はこの後にMONコマンドを使って停止文番号を参照します。

```
>MON
TASK1# 40 TASK2# 90
!!Out of Range
>
```

このようにMONコマンドで各タスクの停止番号を参照しどれがメッセージに対応するかを調べます。

瞬間停電について

MPC-816K,KFには瞬間停電検出機能があります。DC24Vが著しく低下した場合、MPCの赤いLEDが0.1秒程度の間隔で点滅します。MPC-816の赤いLEDの点滅は、プログラムのランタイムエラーでも発生します。この場合は一秒程度の間隔でゆっくりとした点滅です。瞬間停電の原因としては、ＡＣ　ラインの停電のほかに、DC24Vへの過負荷が考えられます。装置が間歇的に停止するような場合、赤いLEDの点滅を確認して下さい。

言語の制限

MPC-816のTNYFSCは簡易言語です。簡単な整数演算と制限された分岐制御文しかありません。TNYFSCではインタプリタの構造を単純にすることによって高速性を保っています。

変数の制限

MPC-816で使用できる変数はA～Zまでのアルファベットと数字を組み合わせたものに限られています。合計286個の変数です。

```
A A0 A1  ･･･  A8 A9
B B0 B1  ･･･  B8 B9
C C0 C1  ･･･  C8 C9
|  |  |
Y Y0 Y1  ･･･  Y8 Y9
Z Z0 Z1  ･･･  Z8 Z9
```

又演算は、3byte長整数のみで表現できる範囲は±8388608です。（Z版では±32767の2byte長整数です。）

配 列

配列要素として用意されているのは、AR()です。AR()はAR(0)～AR(31)の32個のみです。又、P版ではM(n)を使用することができます。AR()はSFTL、SFTRコマンドにてデータをローテーションすることができます。点データとしてX()、Y()、U()、Z()の4つが用意されており配列として使用することができます。要素は各1～300です。X(0)、Y(0)、U(0)、Z(0)は特別な意味を持ち配列として使用することができません。

コメント・文字列の限界

MPC-816でのコメント行はシングルクォートに続く12文字以内です。許される文字は英数及び‘ ^ ’を除く記号に限られます。ラベルは11文字以内です。ラベルの場合は＊（アスタリスク）を頭に付します。文字は英数及び‘ ^ ’と‘　’（スペース）を除く記号に限ります。コメント及びラベル行は文字列出力に使用できます。例えば、RS-232CのCH1に対して”ABC”という文字列を出力するサブルーチンを作る場合は次の様にします。

```
        GOSUB *ABC
            |
*ABC
        PRINT# STR(-1)
        RETURN
```

関数STR(-1)は1行前の文字列（ラベルもしくはコメント）を指定します。複雑なコメントを必要とする時は、ソースファイルにのみ可能ですが”（ダブルクォート）に続く文字がプログラムを読み込む時に切り捨てられるためコメントとして使用できます。但しこの方法は、ソースファイルのみに有効なのでMPC上で見ることはできませんし、パソコンへ上書き保存すると消えてしまいます。

制御文

MPC-816の制御文はFOR～NEXT、GOSUB、GOTO、IF～THEN、IF～GOSUBの５つです。GOSUB及びFOR～NEXTではスタックを用いた制御を行っているので、ネストの深さと制御文中からの飛び出しに注意して下さい。ありがちなトラブルは次の場合でいずれも禁止事項です。又、GOTOとGOSUBを混同して使用している場合も多くありますので注意して下さい。
制御用のスタックは各タスクとも48byteとなっています。これに対してGOSUBでは、4byte FOR～NEXTでは6byteスタックを消費します。このためGOSUBのみでは11、FOR～NEXTのみでは8までのネストが許されますが、混在する場合は次の式で表現されます。

48 ≧ (GOSUBの深さ)*4+(FOR/NEXTの深さ)*6

【例１】

```
10 ON A
20 OFF B
30 GOSUB *WORK1
40 WAIT SW(1)=1
50 OFF 1
100 *WORK1
110 ON 1,2,3
120 OFF 5,4
130 GOTO 40 <---ＲＥＴＵＲＮで戻らずＧＯＴＯで戻すとエラーとなる
```

【例２】

```
10 FOR A=1 TO 10
20 ON 4,5
30 IF SW(5)=1 THEN 70
40 TIME 10
50 OFF 4,5
60 NEXT A
70 ON 1,2 <----------
80 OFF 4,5
```

このようにＩＦ文などでＦＯＲ文からは飛び出せない。何度も飛び出すうちにエラーとなる。

【例１の修正】

```
10 ON A
20 OFF B
30 GOSUB *WORK 1
40 WAIT SW(1)=1
50 OFF 1
    |
100 *WORK 1
110 ON 1,2,3
120 OFF 5,4
130 RETURN
```

【例２の修正】

```
10 A=1
15 *LOOP
20 ON 4,5
30 IF SW(5)=1 THEN 70
40 TIME 10
50 OFF 4,5
60 A=A+1
65 IF A<11 THEN *LOOP
70 ON 1,2
80 OFF 4,5
```

又IF文では、ELSEを使用することができます。ELSEはIF文の直後に置きます。ELSEにはELSE_THENとELSE_GOSUBがあります。（P版のみZにはありません）

【例３】

```
IF SW(0)=1 THEN *SUB1
ELSE_THEN *SUB2
*SUB1       作業１
                :
*SUB2       作業２
```

【例４】

```
IF SW(0)=1 GOSUB *SUB1
ELSE_GOSUB *SUB2
*SUB1       作業１
            RETURN
*SUB2       作業２
            RETURN
```

記述の制限

一行に対する制限

残念ながらMPC-816では次のIF文が記述できません。

```
a. IF X(1)=100000 THEN 200
```

しかし、次のIF文は受付られます。

```
b. IF X(1)=10000 THEN 200
```

これは、TNYFSCの内部の問題なのですがプログラム文はメモリ上で16byte以内となっています。この規則はIF文において限界となっています。a．では100000が3byteとられるのに対してb．の10000は2byteです。b．では16byte以内に収まるのに対してa．では16byteを超えてしまいます。こうした場合インタプリタは!!Too longというメッセージを出力します。PRINT文では次の記述ができず、最後の70000が略されてしまいます。

```
10 PRINT SQR(50000),60000,70000
LIST
10 PRINT SQR(50000),60000
>
```

引き数に対する制限

各コマンドの引き数は3つまでとなっています。例えば、ON、OFFコマンドでは次の記述が可能です。

```
ON 1
ON 1,2
OFF 1,2,3
```

プログラム中に4つ目の入力をしても無視されます。又引き数は変数、定数もしくは関数です。式は記述できません。

```
ON A                    変数
OUT IN(0),0             関数
OUT &HAA,0              定数
```

関数の関数も処理することはできません。

```
>ON IN(IN(IN(0))
???                     エラーとなる
```

演算に対する制限

演算は2項演算のみです。また、式の長さは16文字以内です。次の様な記述をすると問題をおこします。

```
>10 A1=C(1000) ^ &HFF       (15文字)
>LIST
10 A1=C(1000) ^ &H00FF      カーソルアップリタンする (17文字)
>LIST
10 A1=C(1000) ^ &H000F      FFがFとなってしまう
>
```

これは、最初の入力の時15文字以内であったのがリストをとるとヘキサデータが4文字固定表現のため17文字となってしまいます。ここでカーソルを上に移動すると式が17文字となり最後の1文字が略され意味が変わってしまいます。これは、プログラムをセーブすると同様の問題となりますので注意して下さい。ヘキサ表現はできるだけ変数に置き換えておいた方が安全です。

```
H0=&H00FF
H1=&HFF00
A1=C(1000) ^ H0
A2=C(1000) ^ H1
```

演算は+, -, *, /, %, ^, |, xの8種です。+, -, *, /は電卓と同じく加減乗除を意味します。他は次の通りです。

%　余りを算出する（パーセント）
^　論理積（ハット）
|　論理和（パイプ）
x　排他的論理和（スモールエックス）

通信の制限

MPC-816では、CH1というユーザー用のRS-232Cを備えていますが文字列処理はあまり得意ではありません。また、9600bpsフルボーレートではプログラムの組み方によって問題を起こすことがあります。

文字列の扱い

INPUT#、PRINT#は文字列を扱うことができません。MPCとのやりとりは数値データに限った方が見通しが良くなります。ただ、例外的に次の様な応用があります。

STR()関数の使用

PRINT#、PUTS#ではSTR()関数を使用することができます。これにより、コメント文に書かれた文字列を出力することができます。

FIND#、SKIP#、GET#()

送られてくる通信データに文字列が含まれてくることはままありますが、MPCはこれを無視することができます。また、GET#()により1文字をアスキーコードとして扱うことができます。

通信時のトラブル

```
10 GETN# A
20 A1=GET#(0)
30 SKIP# &H000D
```

前記のプログラムは一見うまく動作しそうなのですが、システム的な問題により20のGET#(0)で正確に文字が読み取れないことがあります。これはGETN#が内部的に割り込み禁止を伴う演算をしているために発生します。つまり、文字列を受取りながらGETN#は処理を開始しますが、割り込み禁止とするためこの間文字を受け取らなくなってしまいます。これをさけるには、次のプログラムとするのが完全です。関数RS(1)によって文字列が貯まるのを待ち、バッファに貯まった文字列を処理するとこうした問題は発生しません。

```
5 WAIT RS(1)>10          文字がある程度貯まるのを待つ
10 GETN# A
20 A1=GET#(0)
30 SKIP# &H000D
```

この問題は、INPUT#でも同様に発生します。INPUT#を使用する場合は、CR、LFでターミネートされてからただちに次の文字列を受け取るプロトコルが少ないために問題となることはあまりありませんが、同様の問題をかかえていることに留意して下さい。