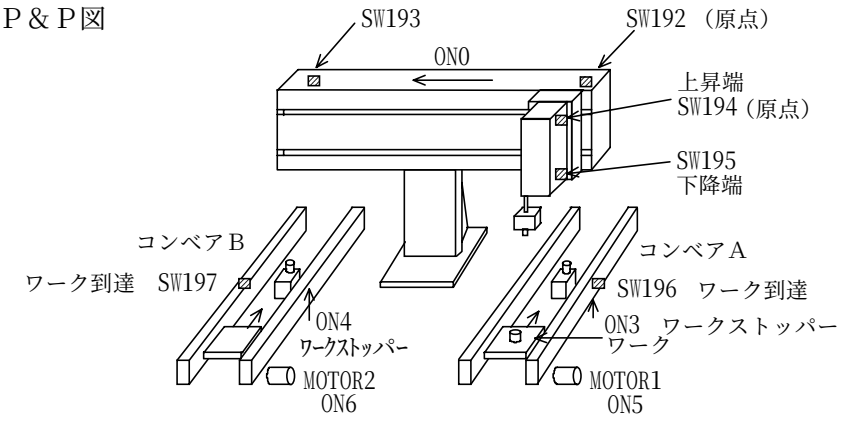
第３章　プログラム例

３．１　Ｉ／Ｏ制御・マルチタスク

１）ＭＰＣの構成と装置概要

図のようなＰ＆Ｐのプログラムを考えます。このＰ＆ＰはコンベアーＡでパレットに乗って搬送されているワークをストッパーで止め､エアーシリンダーで構成されたＰ＆Ｐでコンベアー Ｂのパレットへ乗せかえるというものです｡入力はフォトマイクロセンサーまたはリードスイッチ、出力モーターはＳＳＲを介してＯＮ／ＯＦＦを行いそれ以外はソレノイドバルブを直接駆動するという設定です。この例でのＭＰＣのハード構成はＭＰＣ－６８４、ＭＩＰ－０４８、ＭＯＰ－０４８、ＲＡＣＫ－Ｎ６を各１台ずつ、アドレス設定は出荷時のまま使用します。

Ｐ＆Ｐ図



Ｉ／Ｏマップ

入力（ＭＩＰ－０４８）　出荷時の設定ではポート番号は１９２から始まります。

ポート	接続機器	名称	機能
１９２	シリンダーセンサー	ｋｏｔａｎ	Ｐ＆Ｐ後退端
１９３	シリンダーセンサー	ｚｅｎｔａｎ	Ｐ＆Ｐ前進端
１９４	シリンダーセンサー	ｕｅ	Ｐ＆Ｐ上昇端
１９５	シリンダーセンサー	ｓｈｉｔａ	Ｐ＆Ｐ下降端
１９６	フォトマイクロセンサー	ｗｏｒｋ１	コンベアーＡワーク到達
１９７	フォトマイクロセンサー	ｗｏｒｋ２	コンベアーＢワーク到達

出力（ＭＯＰ－０４８）　出荷時の設定ではポート番号は０から始まります。

ポート	接続機器	名称	機能
０	ソレノイドバルブ	ｚｅｎｓｈｉｎ	Ｐ＆Ｐ前進
１	ソレノイドバルブ	ｋａｋｏｕ	Ｐ＆Ｐ下降
２	ソレノイドバルブ	ｃｈａｃｋ	メカチャック閉
３	ソレノイドバルブ	ｓｔｏｐｐｅｒ１	コンベアーＡストッパー
４	ソレノイドバルブ	ｓｔｏｐｐｅｒ２	コンベアーＢストッパー
５	ＳＳＲ	ｍｏｔｏｒ１	コンベアーＡモーター
６	ＳＳＲ	ｍｏｔｏｒ２	コンベアーＢモーター

メモリーＩ／Ｏ　メモリーＩ／Ｏは－１から－２５６です。

ポート	接続機能	名称	機能
－１	メモリーＩ／Ｏ	ａ＿ｒｅａｄｙ	コンベアーＡＲＥＡＤＹ
－２	メモリーＩ／Ｏ	ｂ＿ｒｅａｄｙ	コンベアーＢＲＥＡＤＹ

この装置の動作は単純でＡからＢへの移し替えの仕事は１つのシーケンスとしてプログラム出来るかも知れません｡しかし、実際の多くの装置はとても複雑で、タクトタイムなどを考えると適当なユニット単位に分割してそれぞれが独立して動作するように制御するのが合理的です。例えばこの装置の場合コンベアーＡの部分、Ｐ＆Ｐの部分、コンベアーＢの部分という３つブロックに分けてプログラムします。このブロックをタスクと言い、ＭＰＣ－６８４では複数のタスクを同時に実行するのでマルチタスクと言います。しかしそれぞれのタスクが勝手に動作していたのでは装置としての目的を達成することは出来ません。そこでタスク同士の連絡を行う必要がありこれをインターロックと言います｡インターロックは､メモリーＩ／Ｏもしくは変数で行います｡メモリーＩ／Ｏとは通常装置間のインターロッ

クを物理的なＩ／Ｏで行うようにタスク間のインターロックを行うための内部的なＩ／Ｏです。メモリーＩ／Ｏは物理的なＩ／Ｏと同様に１つのビットで１（ＯＮ）または０（ＯＦＦ）の２つの状態を表します。そしてＭＰＣ－６８４ではＯＮ／ＯＦＦ／ＳＷなどのコマンドに引き数を負の整数として与える事によりメモリーＩ／Ｏとして、物理的なＩ／Ｏと同様に扱う事ができます。１つのメモリーＩ／ＯではＯＮ／ＯＦＦのどちらしか無いのに対し、変数を用いると複数の情報を伝えることができます。ＭＰＣ－６８４の変数は４byte整数ですから、１つの変数で約±２０億の状態を表すことができます。

２）Ｉ／Ｏのシンボル化

プログラム中の入出力ポートのパラメーターとしてＩ／Ｏのポートナンバーを整数でそのまま記述する事ができますが、シンボルとして扱うと見た目にも分かりやすいプログラムになります。次のようにＣＯＮＳＴコマンドで定数をシンボル化します。

```
CONST  sol1 0
CONST  sw1  192
```

これでＯＮ　０とＯＮ　ｓｏｌ１、ＷＡＩＴ　ＳＷ（１９２）＝０はＷＡＩＴ　ＳＷ（ｓｗ１）＝０と同義になります。（Ｐ＆Ｐ図）の装置のＩ／Ｏ定義

```
10 'I/O定義
20 '入力
30 CONST kotan 192
40 CONST zentan 193
50 CONST ue 194
60 CONST shita 195
70 CONST work1 196
80 CONST work2 197
90 '出力
100 CONST zenshin 0
110 CONST kakou 1
120 CONST chack 2
130 CONST stopper1 3
140 CONST stopper2 4
150 CONST motor1 5
160 CONST motor2 6
170 'メモリーＩ／Ｏ
180 CONST a_ready -1
190 CONST b_ready -2
200 'Ｉ／Ｏ初期化
210 SETIO
```

３）タスクの生成

それぞれのユニットを１～２３のどのタスクで実行するかを宣言します。０はメインのタスクで、プログラムが実行されると最初にこのタスクが動きます。この例ではＰ＆Ｐはタスク０で、コンベアーＡはタスク１、コンベアーＢはタスク２で動かします。

```
220 'タスク実行
230 FORK 1 *conv_a              'タスク１でコンベアーＡを実行
240 FORK 2 *conv_b              'タスク２でコンベアーＢを実行
```

４）Ｐ＆Ｐのタスク

Ｐ＆ＰはコンベアーＡからのＲＥＡＤＹを待ちワークをピックアップしコンベアーＢ側に搬送します。コンベアーＢの準備が出来たところでワークを置きます。このタスクはメインタスク（タスク０）で実行します。

```
1000 *PANDP
1010 '
1020 DO
1030 WAIT SW(a_ready)==1        'コンベアＡＲＥＡＤＹ待ち
1040 '
1050 ON kakou                   'ワークピックアップ
1060 WAIT SW(shita)==1
1070 ON chack
1080 TIME 500
1090 OFF kakou
1100 WAIT SW(ue)==1
1110 '
```

```
1120 OFF a_ready                ’ピックアップ完了－＞コンベアＡ
1130 '
1140 ON zenshin                 ’Ｐ＆Ｐ前進
1150 WAIT SW(zentan)==1
1160 '
1170 WAIT SW(b_ready)==1        ’コンベアーＢ準備完了待ち
1180 '
1190 ON kakou                   ’ワークプレース
1200 WAIT SW(shita)==1
1210 OFF chack
1220 TIME 300
1230 OFF kakou
1240 WAIT SW(ue)==1
1250 '
1260 OFF b_ready                ’ワークプレース完了－＞コンベアー
1270 '
1280 OFF zenshin                ’Ｐ＆Ｐ後退
1290 WAIT SW(zentan)==1
1300 '
1310 LOOP
```

５）コンベアーＡのタスク

コンベアーＡはストッパーを上げｍｏｔｏｒ１を駆動してパレットが到達するのを待ちます。パレットが到達するとＰ＆ＰにＲＥＡＤＹを出し、Ｐ＆Ｐはワークを持ち上げるとＲＥＡＤＹをＯＦＦします。

```
2000 *conv_a
2010 ON motor1
2020 DO
2030 ON stopper1
2040 WAIT SW(work)==1
2050 TIME 500
2060 OFF motor1
2070 '
2080 ON a_ready                 ’ワークが来たョ
2090 WAIT SW(a_ready)==0        ’Ｐ＆Ｐ動作完了待ち
2100 '
2110 OFF stopper1
2120 ON motor1
2130 WAIT SW(work1)==0
2140 TIME 500
2150 LOOP
```

６）コンベアーＢのタスク

コンベアーＢはモーターを回しストッパーを上げ空パレットが到達するのを待ち、Ｐ＆ＰへＲＥＡＤＹを出しＰ＆Ｐはワークを置くとＲＥＡＤＹをＯＦＦします。

```
3000 *conv_b
3010 ON motor2
3020 DO
3030 ON stopper2
3040 WAIT SW(work2)==1
3050 TIME 500
3060 OFF motor2
3070 '
3080 ON b_ready                 ’パレット準備完了
3090 WAIT SW(b_ready)==1        ’ワークを置いたョ
3100 '
3110 ON motor2
3120 OFF stopper2
3130 WAIT SW(work2)==0
3140 TIME 500
3150 LOOP
```

７）装置全体のプログラム

前記プログラムを入力してＲＥＮＵＭすると次のＬＩＳＴになります。

```
10   'Ｉ／Ｏ定義
20   '入力
30    CONST kotan 192
40    CONST zentan 193
50    CONST ue 194
60    CONST shita 195
```

```
70    CONST work1 196
80    CONST work2 197
90   '出力
100   CONST zenshin 0
110   CONST kakou 1
120   CONST chack 2
130   CONST stopper1 3
140   CONST stopper2 4
150   CONST motor1 5
160   CONST motor2 6
170  'メモリーＩ／Ｏ
180   CONST a_ready -1
190   CONST b_ready -2
200  'Ｉ／Ｏ初期化
210   SETIO
220  'タスク実行
230   FORK 1 *conv_a              'タスク１でコンベアーＡを実行
240   FORK 2 *conv_b              'タスク２でコンベアーＢを実行
250  *PANDP
260  '
270   DO
280     WAIT SW(a_ready)==1       'コンベアＡＲＥＡＤＹ待ち
290  '
300     ON kakou                  'ワークピックアップ
310     WAIT SW(shita)==1
320      ON chack
330      TIME 500
340      OFF kakou
350      WAIT SW(ue)==1
360  '
370      OFF a_ready               'ピックアップ完了－＞コンベアＡ
380  '
390      ON zenshin                'Ｐ＆Ｐ前進
400      WAIT SW(zentan)==1
410  '
420   WAIT SW(b_ready)==1         'コンベアーＢ準備完了待ち
430  '
440   ON kakou                    'ワークプレース
450   WAIT SW(shita)==1
460   OFF chack
470   TIME 300
480   OFF kakou
490   WAIT sw(ue)==1
500  '
510   OFF b_ready                 'ワークプレース完了－＞コンベアーＢ
520  '
530   OFF zenshin                 'Ｐ＆Ｐ後退
540   WAIT SW(zentan)==1
550  '
560   LOOP
570  *conv_a
580   ON motor1
590   DO
600   ON stopper1
610   WAIT SW(work1)==1
620   TIME 500
630   OFF motor1
640  '
650   ON a_ready                  'ワークが来たョ
660   WAIT SW(a_ready)==0         'Ｐ＆Ｐ動作完了待ち
670  '
680   OFF stopper1
690   ON motor1
700   WAIT SW(work1)==0
710   TIME 500
720   LOOP
730  *conv_b
740   ON motor2
750   DO
760     ON stopper2
770     WAIT SW(work2)==1
780     TIME 500
790     OFF motor2
800  '
810   ON b_ready                  'パレット準備完了
820   WAIT SW(b_ready)==0         'ワークを置いたョ
830  '
840   ON motor2
850   OFF stopper2
```

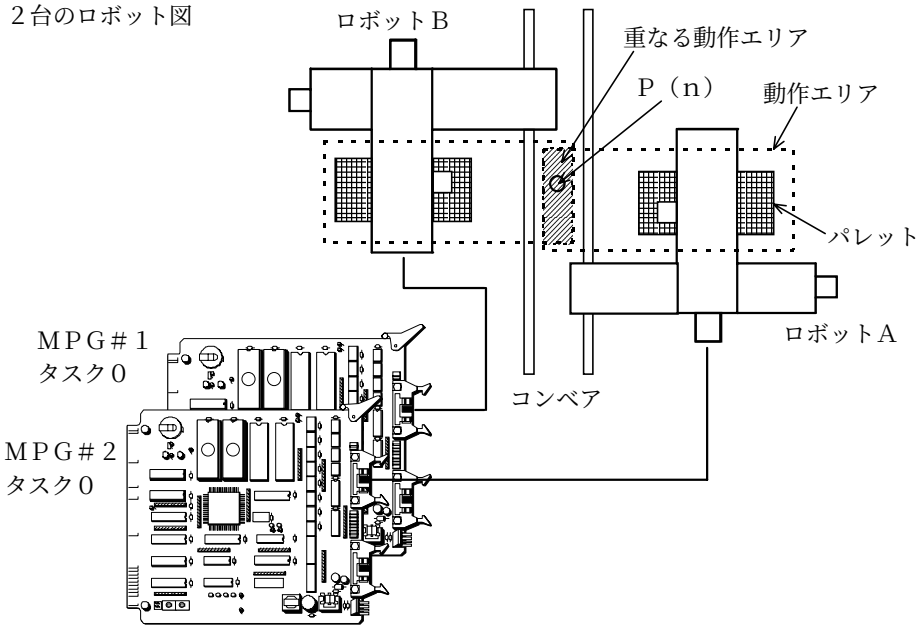
```
860     WAIT SW(work2)==0
870     TIME 500
880     LOOP
```

３．２　タスクの制御

１）タスクの交通整理（セマフォ）

タスク間のインターロックが必要なケースとして先の例でのタスクの同期と、もう１つ、一度に１つのタスクしか使用できない共通資源（クリティカルソース）に対し排他処理を行う場合があります。例えば、１つのＲＳ－２３２Ｃポートやリエントラント出来ないサブルーチンを複数のタスクで使用したり、重なる動作エリアを持つ２台のロボット（図）の干渉防止などがあります。本来、前者のような一人の聞き手に対して一度に何人もの人が同時に話しかけるようなプログラムは避けるべきです。聖徳太子のように器用に聞き分けることはできないのです。１つのデバイスは１つのタスクで管理することが原則ですが、それができないときは複数のタスクが同時にアクセスしないようにしなければなりません。後者はタスクのアクセスする対象は異なりますが、ポイントや空間が共通資源として考えられます。このようなタスクの交通整理をセマフォと言い、それを合理的に確実に行う方法として、セマフォ関数（ＲＳＶ（ｎ），ＲＬＳ（ｎ））があります。

２台のロボット図



＊コンベアー上のワークを２台のロボットのどちらかでパレットに搬送する。
片方がＰ（ｎ）へアクセス中、他方は絶対にそのエリアに進入してはならない。

① のプログラムはコマンドリファレンスのＲＳＶ（ｎ）の項のプログラムです。このプログラムは２つのタスクが勝手に走ってＲＳ－２３２ＣのＣＨ１を同時にアクセスしてしまい出力が混乱しています。もしもこれが２台のロボットなら衝突して大きな事故になってしまいます。

② はメモリーＩ／ＯとＯＮ／ＯＦＦ／ＳＷ（ｎ）を組み合わせてテスト＆セットを行うことによりタスクの衝突を防ごうとしています。しかしこの方法ではメモリーＩ／Ｏのテストとセットが別々のコマンドによって行われているため時間差が生じてタイミングによってはやはり混乱が生じる恐れがあります。

③ はメモリーＩ／Ｏをセマフォ関数を使ってテスト＆セットしています。ＲＳＶ（－１）でメモリーＩ／Ｏの－１の状態を確認し０であればそれに１をセットする事によりセマフォの獲得をしてＰＲＩＮＴ文を実行し、ＲＬＳ（－１）でセマフォを解放します。この様にＲＳＶは②）のＳＷとＯＮによるテスト＆セットを１つ関数で行う事により確実な排他処理を実現できます。

④ はＲＳ－２３２Ｃへ出力する専用のタスクを設けたものです。ここでもメモリーＩ／Ｏを使ってタスクの情報交換をしています。

①２つのタスクが勝手に動作（交通整理が行われていない）

```
                                        'もしもロボットなら
10      time0=timer                        ↓
```

```
20      time=timer
30    FORK 1 *aho
40    DO
50      WAIT time<>timer
60      PRINT "abcdefg" "ABCDEFGH"          'ロボットAがP(n)へ動作
70        time=timer
80    LOOP
90 *aho
100   DO
110     WAIT time0<>timer
120     PRINT "123456" "7890123"            'ロボットBがP(n)へ動作
130       time0=timer
140   LOOP
#run
123456789abcdefgABC0123                     'ガーン衝突！エライコッチャ
DEFGH
abcdefgAB1234567890CDEFGH
123
12345678901abcdefgA23
BCDEDGH
```

（ｔｉｍｅｒは予約変数です）

②ＯＮ／ＯＦＦ／ＳＷ（ｎ）による交通整理（悪い例・交通整理できません）

```
10      time0=timer
20      time=timer
30    OFF -1
40    FORK 1 *aho
50    DO
60      WAIT time<>timer
70      WAIT SW(-1)==0
80      ON -1
90      PRINT "abcdefg" "ABCDEFGH"
100     OFF -1
110       time=timer
120   LOOP
130 *aho
140   DO
150     WAIT time0<>timer
160     WAIT SW(-1)==0
170     ON -1
180     PRINT "123456" "7890123"
190     OFF -1
200       time0=timer
210    LOOP
#run
abcdefgABCDEFGH                             'たまたまうまくいった。デモ．．．
1234567890123
abcdefgABCDEFGH
1234567890123
```

③セマフォ関数を用いた交通整理（良い例）

```
10      time0=timer
20      time=timer
30    FORK 1 *aho
40    DO
50      WAIT time<>timer
60      WAIT RSV(-1)==0                     'セマフォ確認
70      PRINT "abcdefg" "ABCDEFGH"          'ロボットA動作
80        dummy=RLS(-1)                     'セマフォ解除
90        time=timer
100   LOOP
110 *aho
120   DO
130     WAIT time0<>timer
140     WAIT RSV(-1)==0                     'セマフォ確認
150     PRINT "123456" "7890123"            'ロボットB動作
160       dummy=RLS(-1)                     'セマフォ解除
170       time0=timer
180   LOOP
#run
abcdefgABCDEFGH                             'ロボットA動作
1234567890123                               'ロボットB動作
abcdefgABCDEFGH                              メデタシメデタシ
1234567890123
```

④ＲＳ－２３２Ｃへの出力を１本化（良い例）

```
10      time0=timer
20      time=timer
30    FORK 1 *aho
40    FORK 2 *baka
50    DO
60      IF SW(-1)==1 THEN : GOSUB *case1 : OFF -1 : END_IF
70      IF SW(-2)==1 THEN : GOSUB *case2 : OFF -2 : END IF
80    LOOP
90  *case1
100   PRINT "abcdefg" "ACDEFGH"
110   RETURN
120 *case2
130   PRINT "123456" "78901234"
140   RETURN
150 *aho
160   DO
170     WAIT time<>timer
180     ON -1
190     WAIT SW(-1)==0
200       time=timer
210   LOOP
220 *baka
230   DO
240     WAIT time0<>timer
250     ON -2
260     WAIT SW(-2)==0
270       time0=timer
280   LOOP
#run
12345678901234
abcdefgABCDEFGH
12345678901234
abcdefgABCDEFGH
```

２）タスクのＦＯＲＫ、ＱＵＩＴ、ＰＡＵＳＥ

インタプリタはタスク０を親のタスクとして実行し、ＦＯＲＫコマンドで子タスクを実行します。子タスクには優先順位や連続性はありません。昇順、降順、中飛びは関係ありません。また、子タスク同士のＦＯＲＫ、ＰＡＵＳＥ、ＱＵＩＴもできます。たとえば、自動モード／単動モードの切り替えを行うのに常に自動と単動の２つのタスクを走らせておく必要はありません。自動、単動それぞれルーチンを用意しておき自動モードでは単動のルーチンをＱＵＩＴし、自動のルーチンをＦＯＲＫします。単動モードではその逆をします。このように、１つのタスクでもＦＯＲＫするプログラムを差し替えることにより複数の機能を実現する事ができます。

```
*MAIN
  DO
    WAIT SW(start)==0
    WAIT SW(start)==1           'スタートスイッチ入力待ち
    IF SW(mode)==1 THEN
      FORK 1 *AUTO              'ｍｏｄｅスイッチがＯＮならばＡＵＴＯモード
      ELSE
      FORK 1 *MANUAL
    END_IF
    WAIT SW(stop)==1            'ストップスイッチ入力待ち
    QUIT 1                      'タスク１をＱＵＩＴ
    WAIT TASK(1)==-2
  LOOP
'**********************
*AUTO                          '自動運転のプログラム
   DO
     JUMP P(1)
     ON 0
     JUMP P(2)
     OFF 0
   LOOP
'**********************
*MANUAL                        '単動運転のプログラム
   DO
     JUMP P(1)
     ON 0
     WAIT SW(tando)==0
     WAIT SW(tando)==1
   LOOP
```

３．３　パルス発生について

ＭＰＣ－６８４シリーズにはの３種類のＰＧボードがあります。最高パルスレート、軸数、その他の機能から使用目的に合ったボードを選択して下さい。

１）各ボードの概要

☆ＭＰＧ－６８Ｋ

最高パルスレート	１００kpps
軸数	４
補間	３軸直線補間、４軸直線補間
制御コマンド	MOVE,RMOV,HOME等のロボットコマンド

汎用的なＰＧボードです。座標値はＭＰＧ－６８Ｋで管理されており、ティーチングポイント、変数、定数による絶対・相対座標移動、パレタイズができます。
ロボット、Ｐ＆Ｐ、スピンドルモータ等 一般的な軸制御

☆ＭＰＧ－４０５

最高パルスレート	４００kpps
軸数	４
補間	２軸直線補間、円弧補間
制御コマンド	MOVE,RMOV,HOME等のロボットコマンド、円弧補間コマンド(SETX,TR)

ＭＰＧ－６８Ｋの上位機種です。高速性や円弧補間を必要とする装置に適します。
ロボット、巻き線機、塗布機 等

☆ＭＰＧ－３２０２

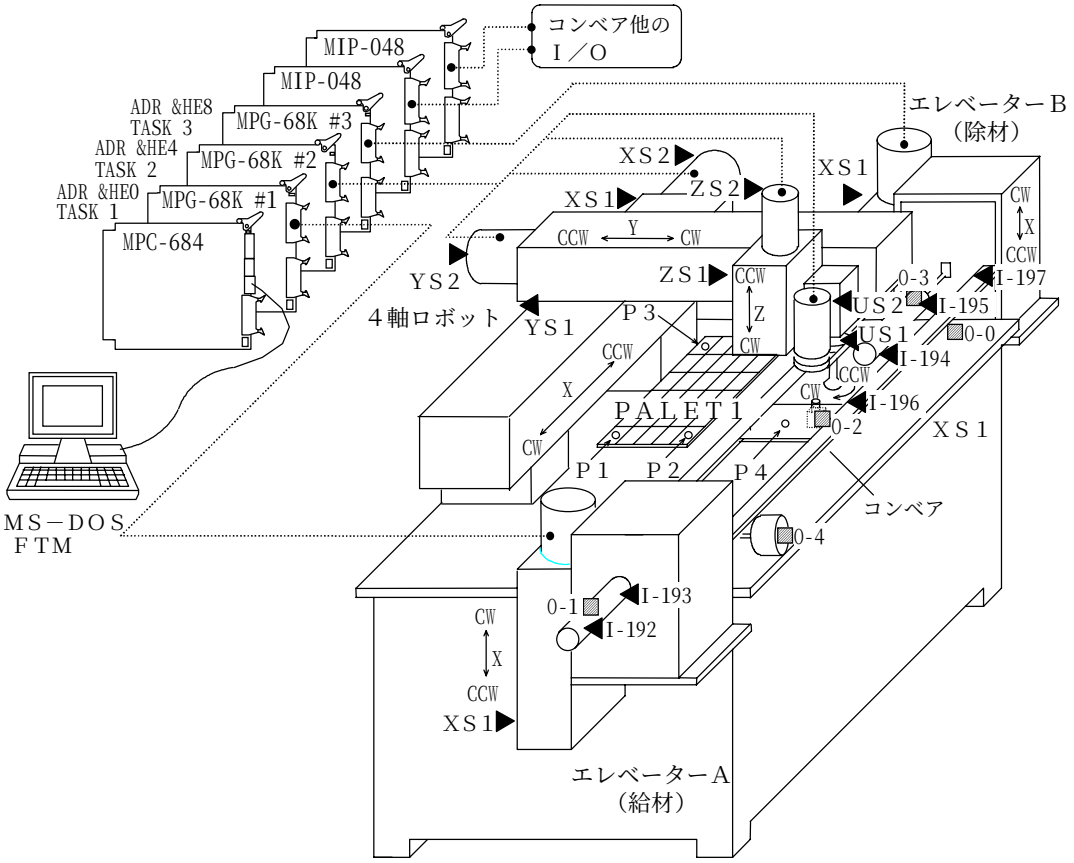
最高パルスレート	１mpps
軸数	非同期２軸
駆動	インデックス、Ｓ字加減速、定速、パルスレート途中変更
その他の機能	エンコーダーカウント、独自の原点復帰、ドライバーＩ／Ｆ 等
制御コマンド	専用コマンド（ＳＴ＿ＲＥＧ、ＲＥＧ、ＲＥＧ３、ＣＭＮＤ）

１枚のボード上に全く同等の２組のパルスポートが有り、独立して動きます。
非同期の多軸、高速、高精度、レート途中変更などの複雑なパルス発生を必要とするものに適します。
ロータリーエンコーダーのカウンタボードとしても利用できます。
（ＭＰＧ－３２０２については第７章ハードリファレンス、製品別マニュアルをご覧下さい。）

次にＭＰＧ－６８Ｋ（またはＭＰＧ－４０５）を使った４軸ロボットシステムを考えます。

２）ＭＰＣの構成と装置の概要

ＭＰＣ－６８４の本領が発揮されるのがパルス発生です。最大４軸の制御が可能なパルスボードをタスク毎に割り当てることにより最高２３枚まで増設できるので（実際はラックが最高で１８スロットなのでボード数は最高１７枚までです。）コンパクトな多軸ＮＣ制御マシーンを容易に実現することができます。また、パルス制御はＩ／Ｏの制御も含めて全て１つのプログラム中に記述できるのでソフト面でも開発・保守性の優れたシステムを構成できます。次の図はステップモーター駆動のエレベーターＡからコンベアーに供給された基板にサーボモーター駆動の４軸ロボットでＰＡＬＥＴ１からＩＣを実装しエレベーターＢに除材するという簡易的なマウンターを想定したものです。ここではこのシステムでのＭＰＣ－６８４の制御プログラムをパルス発生を中心に考えます。ボード構成はホストのＭＰＣ－６８４が１枚、パルス発生ボードとしてＭＰＧ－６８Ｋを４軸ロボットに１枚、各エレベーターに１枚ずつの計３枚、コンベアーなどのセンサーやソレノイドバルブに、入出力ボードをそれぞれ１枚ずつ使用します。これで入力４８点、出力４８点を持つ最大１２軸のパルス制御可能なコントローラーができました。



簡易マウンター・Ｉ／Ｏ　ＭＡＰ

◇入力（ＭＩＰ－０４８　Ｊ３　コネクター）

ＰＯＲＴ	接続機器	機　能
１９２	シリンダーセンサー	基板排出シリンダー後退端
１９３	シリンダーセンサー	基板排出シリンダー前進端
１９４	シリンダーセンサー	基板収納シリンダー後退端
１９５	シリンダーセンサー	基板収納シリンダー前進端
１９６	フォトセンサー	実装位置基板到達検出
１９７	フォトセンサー	収納基板到達検出

◇出力（ＭＯＰ－０４８　Ｊ３　コネクター）

ＰＯＲＴ	接続機器	機　能
0	ソレノイドバルブ	ワーク吸着真空
1	ソレノイドバルブ	基板排出シリンダー
2	ソレノイドバルブ	基板ストッパー
3	ソレノイドバルブ	基板収納シリンダー
4	リレー	コンベアーモーター回転
5		

４軸ロボット原点センサー入力
（ＭＰＧ－６８Ｋ＃１　Ｊ４コネクター）

ＰＯＲＴ	接続機器	機　能
ＸＳ１	フォトセンサー	Ｘ軸原点近傍
ＸＳ２	サーボドライバー	ドライバーＺ相　Ｘ軸原点
ＹＳ１	フォトセンサー	Ｙ軸原点近傍
ＹＳ２	サーボドライバー	ドライバーＺ相　Ｙ軸原点
ＵＳ１	フォトセンサー	Ｕ軸原点近傍
ＵＳ２	サーボドライバー	ドライバーＺ相　Ｕ軸原点
ＺＳ１	フォトセンサー	Ｚ軸原点近傍
ＺＳ２	サーボドライバー	ドライバーＺ相　Ｚ軸原点

除材・給材エレベーター原点センサー入力
（ＭＰＧ－６８Ｋ＃２，＃３　Ｊ４コネクター）

ＰＯＲＴ	接続機器	機　能
ＸＳ１	フォトセンサー	エレベーター原位置
ＸＳ２		
ＹＳ１		
ＵＳ１		
ＵＳ２		
ＺＳ１		
ＺＳ２		

３）各ボードの設定

パルス発生ボードやＩ／Ｏボードなど同種のボードを複数使用するときはアドレスが重ならないようにＤＳＷやショートピンの設定を変更します。ここではホストのＭＰＣ－６８４とスレーブのＭＰＧ－６８Ｋ＃１及びＭＩＰ－０４８、ＭＯＰ－０４８は出荷時の設定のまま使用します。ＭＰＧ－６８Ｋの＃２と＃３はＤＳＷのアドレス設定を変更します。パルスボードのタスクへの引き当ては次の図のアドレスを宣言することにより実施されます。Ｉ／Ｏボードはアドレスの設定によりポート番号が決定されます。

ボード	アドレス
ＭＰＣ－６８４	
ＭＰＧ－６８Ｋ　＃１	&ＨＥ０　出荷時設定
ＭＰＧ－６８Ｋ　＃２	&ＨＥ４　アドレス変更
ＭＰＧ－６８Ｋ　＃３	&ＨＥ８　アドレス変更
ＭＩＰ－０４８　＃１	&Ｈ２０　出荷時設定
ＭＯＰ－０４８　＃１	&Ｈ００　出荷時設定

４）基本的なパルス発生

最も簡単なパルス発生としてエレベーターＡでの１軸（Ｘ軸）のパルス発生について考えます。

①タスクへのパルスボードの引き当て

パルスボードはタスク別に管理され、ＰＧコマンドで特定のタスクに引き当てられるとそのタスクでのパルス発生コマンドを実行します。

```
pg &he0 0          'アドレス＆ｈｅ０のパルスボードをタスク０で使用する
pg &he0 1          'アドレス＆ｈｅ０のパルスボードをタスク１で使用する
pg &he0            'タスクを省略するとｐｇコマンドを実行したタスクの支配を受ける
```

パルスボードの設定状況はｐｇとだけ入力すると一覧表示されます。プログラムでは次のようになります。

```
10 pg &he0 1                    '＆ｈｅ０はタスク１で使用
20 FORK 1 *task1                'タスク１を起動
  |
1000 *task1
1010 ACCEL 10000 1000 100       'タスク１でのパルス命令は＆ｈｅ０のボードに対して有効
  |
```

次のプログラムは前記と同じです。

```
10 FORK 1 *task1                'タスク１を起動
  |
1000 *task1
1010 pg &he0                    'タスク１の中でＰＧ宣言
1020 ACCEL 10000 1000 100
```

②パラメーターの設定

ＰＧコマンドもパルス関係のパラメーターの１つですがその他にスピード､原点復帰に関するものなどがあります｡これらは必ずしも全て設定しなければならないというものではありません。デフォルトの設定値のままで良い場合もありますし、使用する軸によっては必要のないものもあります。また、コマンドラインからのダイレクト入力での設定も行えますが､ボードの初期化やＲＯＭ化､保守等のことを考慮してプログラムの実行や環境を設定するパラメーターは必ずプログラム中に記述しておきます。

スピードに関するもの

ＡＣＣＥＬ	加減速テーブルの作成
ＦＥＤＤ	ＪＵＭＰ移動のスピード
ＦＥＤＨ	ＨＯＭＥ，ＨＯＭＺの待避移動のスピード
ＦＥＤＴ	ＴＥＡＣＨでのスピード
ＦＥＤＺ	ＪＵＭＰのＺ軸のスピード（下降移動を除く）
ＦＥＥＤ	ＭＯＶＥ，ＲＭＯＶ，ＧＯ，ＲＭ移動のスピード

原点復帰に関するもの

ＳＨＯＭ	ＸＹＵの原点復帰モードの設定
ＳＨＭＺ	Ｚの原点復帰モードの設定

③原点復帰

原点復帰の方向はＣＣＷですが初期状態ですでにセンサーがオンとなる位置にいる可能性があるので原点復帰に先立ち適当なＣＷ方向の退避移動をします。原点センサーはコネクターＪ４の１番ピンに接続されたＸＳ１で、このセンサーがオン状態になるまで原点復帰動作を行います。

```
PG &he0 0                       '＆Ｈｅ０のボードはタスク０に引き当て
ACCEL 100000 10000 1000         '加減速テーブル作成
FEDH 30                         '退避移動のスピード設定
SHOM &h02 1000                  'Ｘ軸ＣＣＷ方向へ１kppsで
HOME &h01 500                   'ＸＳ１がＯＮまで原点復帰、退避量は５００パルス
'
```

原点復帰が終了するとその点の座標はＸ＝０、Ｙ＝０、Ｕ＝０となります。これがいわゆる機械的原点です。

④ティーチングによるポイントの表示
エレベーターは機械的原点に復帰後、作業を開始する電気的原点へと移動します。この点の設定はティーチングで行います。ティーチングにより機械的原点の位置には関係なく正確な作業開始位置を割り出すことができます。
ティーチングモードに移るのはＴまたはＴＥＡＣＨと入力します。

```
#T
 PG[0,E0] X= 0 Y= 0 U= 0 Z= 0 dx= 500 dy= 500 du= 500 dz= 500
```

ＰＧの後ろの［］内は現在ティーチングの対象となっているパルスボードのタスク番号とアドレスを表します。これはＰＧコマンドにより設定されたもので、この例ではタスク０にターゲットのボードが引き当てられているためこの状態でティーチングを行うことができます（ターミナルにはタスク０が引き当てられている）が、タスク０以外に引き当てられているボードは＜ＴＡＢ＞、＜＋＞、＜－＞で切り替えます（後述)。続くＸ，Ｙ，Ｕ，Ｚの数値は現在点の座標値（パルス）です。ｄｘ，ｄｙ，ｄｕ，ｄｚはＪＯＧ送りのインチング量（パルス）で、＜０＞～＜３＞のキーで切り替えます。また、この移動量はＳＥＴコマンドで設定変更ができます。

```
#SET                                    '現在の設定状況の表示(デフォルト)
 dx= 10 dy= 10 du= 10 dz= 10            '＜０＞キーの設定量
 dx= 50 dy= 50 du= 50 dz= 50            '＜１＞　〃
 dx= 100 dy= 100 du= 100 dz= 100        '＜２＞　〃
 dx= 500 dy= 500 du= 500 dz= 500        '＜３＞　〃
#SET 0 2 2 2 2                          '＜０＞キーの設定を変更
#SET                                    '表示
 dx= 2 dy= 2 du= 2 dz= 2
 dx= 50 dy= 50 du= 50 dz= 50
 dx= 100 dy= 100 du= 100 dz= 100
 dx= 500 dy= 500 du= 500 dz= 500
#
```

ＪＯＧ送りはアルファベットキーです。例えばｘでＸ軸ＣＷ方向、ＸでＸ軸ＣＣＷ方向に動きます。(コマンドリファレンスＴＥＡＣＨ参照）最初は大きく、除々に細かくインチングして正確に位置を合わせ、＜Ｐ＞を押した後、ポイントナンバーを入力して＜ＥＮＴＥＲ＞を押します。ティーチモードからもどるには＜Ｑ＞を押します。

```
#T
 PG[0,E0] X= 312 Y= 0 U= 0 Z= 0 dx= 2 dy= 2 du= 2 dz= 2      '＜Ｐ＞を押す
  点番号を押して下さい  P1
 PG[0,E0] X= 312 Y= 0 U= 0 Z= 0 dx= 2 dy= 2 du= 2 dz= 2      'ここがＰ(１)
#
```

⑤定ピッチ送りのプログラム
エレベーターはＰ（１）から一定量のピッチを繰り返します。パルス発生のコマンドには絶対座標移動と相対座標移動が用意されており、目的によって使い分けることが出来ます。次の３つのプログラムは同じ動作になります。

```
①
MOVE P(1)
TIME 500
FOR i=1 TO 9
  MOVE X(0)+500 0 0              '現在点基準の絶対座標移動
  TIME 500
NEXT i
-------------------------
②
MOVE P(1)
TIME 500
FOR i=1 TO 9
  MOVE X(1)+i*500 0 0            'P(１)基準の絶対座標移動
  TIME 500
NEXT i
-------------------------
③
MOVE P(1)
TIME 500
FOR i=1 TO 9
  RMOV 500 0 0                   '相対座標移動
  TIME 500
NEXT i
```

⑥パルスの監視・停止

ソフトで非常停止を行うにはスイッチを監視する専用のタスクを設けて他のタスクをＱＵＩＴするのが合理的ですが、パルス発生中のタスクがある場合は、パルス出力を停止した後でＱＵＩＴをしなくてはいけません。次のプログラムは、タスク０で非常停止スイッチを監視してパルス出力をしているタスク１を停止する例です。非常停止スイッチが押されるとＤＳ＿ＰＧコマンドでパルス発生を禁止、ＢＳＹ（）関数でパルス停止を確認してタスク１をＱＵＩＴします。ＥＮ＿ＰＧコマンドは禁止したパルス発生を再開します。

```
            PG &HE0 1
            CONST em_stop 192
'
            DO
              FORK 1 *task1                 'タスク１起動
              WAIT SW(em_stop)==1           '非常停止スイッチ監視
              DS_PG 1                       'タスク１パルス発生禁止
              WAIT BSY(1)<>0                'パルス停止確認
              TIME 10
              QUIT 1                        'タスク１停止
              WAIT TASK(1)==-2
              SETIO                         'Ｉ／Ｏ初期化
              WAIT SW(em_stop)==0
              EN_PG 1                       'タスク１パルス発生再開
            LOOP
'
*task1
            DO
                MOVE 1000 0 0
                TIME 500
                MOVE 0 0 0
                TIME 500
            LOOP
```

☆ＲＥＶ１．２より、パルス発生の停止とタスクのポーズを一括で行うＱ＿ＰＡＵＳＥモードがサポートされました。詳しくは、７）Ｑ＿ＰＡＵＳＥモードでのパルス発生の停止、第６章コマンドリファレンス”Ｑ＿ＰＡＵＳＥ”をご覧下さい。

５）４軸ロボットのプログラム例

まず、作業開始に先立ち原点復帰を行います。次にＰＡＬＥＴ命令を使ってパレットの給材点へ移動し、バキュームパットで吸着したワークをコンベアー上の基板に搬送して実装します。ＭＰＣ－６８４のパルスコマンドは直行座標系の３，４軸ロボットを前提としており、例えばＪＵＭＰ命令１つで姿勢制御を含むゲートモーションが実現できます。

①原点復帰の動作

このロボットでは各軸の原点センサーは次のようにＭＰＧ＃２のＪ４コネクターに接続されています。原点復帰方向はすべてＣＣW方向で、原点復帰順序はまずＺ軸を原点近傍に移動、次にＸＹＵ軸を原点近傍に３軸同時に移動、そしてＺ軸の原点復帰を行い最後にＸＹＵ軸を原点復帰させます。原点近傍までは早くそこから原点まではゆっくり動作させます。

原点の座標値はＸ＝０，Ｙ＝０，Ｕ＝０，Ｚ＝０となります。

コネクター	原点センサー
Ｊ４－１	ＸＳ１　Ｘ軸原点近傍
Ｊ４－２	ＸＳ２　Ｘ軸原点
Ｊ４－３	ＹＳ１　Ｙ軸原点近傍
Ｊ４－４	ＹＳ２　Ｙ軸原点
Ｊ４－５	ＵＳ１　Ｕ軸原点近傍
Ｊ４－６	ＵＳ２　Ｕ軸原点
Ｊ４－７	ＺＳ１　Ｚ軸原点近傍
Ｊ４－８	ＺＳ２　Ｚ軸原点

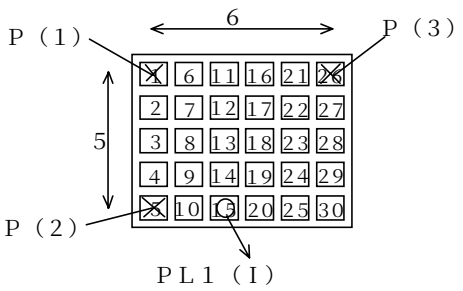
```
SHOM &H2A 5000      'Ｚ方向、スピード設定
SHMZ &H2  5000      'ＸＹＵ方向、スピード設定
HOMZ &H1 200        'Ｚ原点近傍まで動作
HOME &H15 200       'ＸＹＵ原点近傍まで動作
'
SHOM &H2A 100       'Ｚ方向、スピード設定
SHMZ &H2  100       'ＸＹＵ方向、スピード設定
HOMZ &H3 0          'Ｚ原点復帰
HOME &H3f 0         'ＸＹＵ原点復帰
```

②パレタイズ

ロボットの作業点はティーチングで設定しますが、パレットからの除給材の場合、マトリックスを１つ１つティーチングしていたのでは大変な労力になります。この様なパレタイズにはＰＡＬＥＴ命令を使うことにより効率の良いプログラムを作ることができます。ＭＰＣ－６８４には４つのＰＡＬＥＴ命令が用意されていてパレット間の搬送作業

などが容易に行えます。パレタイズにはまず次の様にパレット宣言をします。このパレットは図のように縦横５×６のマトリックスで３隅をＰ（１），Ｐ（２），Ｐ（３）としてティーチングを行うものとします。このパレット宣言により３箇所のティーチングポイントからパレットのマトリックスの３０点の座標が計算されＰＬｍ（ｎ）関数により与えられます。パレットはＰＡＬＥＴ１からＰＡＬＥＴ４までの４つが宣言できます。次のプログラムはパレットとコンベアー間をゲートモーションでワークを搬送する無限ループです。

P(1) P(2) P(3) 5 6




```
PALET1 P(1) P(2) P(3) 5 6
DO
  FOR I=1 TO 30
    JUMP PL1(I)
    ON 0
    TIME 500
    JUMP P(4)
    OFF 0
    TIME 500
  NEXT I
LOOP
```

６）装置全体のプログラム

ここまでユニット別に考えていたプログラムをタスクのインターロックをおりこみながら一本化して装置全体のプログラムとして仕上げます。

```
'****************************
'    非常停止スイッチ監視
'****************************
'
  WAIT SW(216)==1                    'ＷＡＩＴ　ＳＴＡＲＴ
  DO
    FORK 1 *elevator1
    FORK 2 *elevator2
    FORK 3 *robot
    FORK 4 *conveyer
    WAIT SW(217)==1                  '非常停止スイッチ監視
    DS_PG 1 2 3                      'パルス発生禁止
    TIME 10
    WAIT BSY(1)<>0                   'パルス停止確認
    WAIT BSY(2)<>0
    WAIT BSY(3)<>0
    QUIT 1 2 3 4                     'タスク停止
    SETIO                            'Ｉ／Ｏ初期化
    WAIT SW(217)==0
    EN_PG 1 2 3                      'パルス発生再開
  LOOP
'****************************
'    給材エレベーター
'****************************
*elevator1
'          原点復帰
  PG &HE0                            'ＰＧボード引き当て
  ACCEL 100000 10000 1000            '加減速テーブル作成
  FEDH 30                            '待避スピード設定
  SHOM &H2 1000                      '原点復帰条件設定
  HOME &H1 500                       '原点復帰
'
  DO
    MOVE P(11)                       'Ｐ（１１）はティーチングした点
    FOR i=1 TO 9
      ON -1                          'コンベアーへＲＥＡＤＹ
      WAIT SW(-1)==0                 '基板押し出し完了待ち
      RMOV 500 0 0                   '５００パルス相対移動
    NEXT i
  LOOP
'****************************
'    除材エレベーター
```

```
'**************************
*elevator2
'         原点復帰
  PG &HE4                                 'ＰＧボード引き当て
  ACCEL 100000 10000 1000                 '加減速テーブル作成
  FEDH 30                                 '待避スピード設定
  SHOM &H2 1000                           '原点復帰条件
  HOME &H1 500                            '原点復帰
'
  DO
    MOVE P(21)                            'Ｐ（２１）はティーチングした点
    FOR j=1 TO 9
      ON -2                               'コンベアーへＲＥＡＤＹ
      WAIT SW(-2)==0                      '基板収納完了待ち
      RMOV 500 0 0                        '５００パルス相対移動
    NEXT j
  LOOP
'**************************
'   ロボット
'**************************
*robot
  PG &HE8                                 'ＰＧボード引き当て
  ACCEL 10000 1000 100                    '加減速テーブル作成
'
'        ＮＥＡＲ　ＯＲＩＧＩＮ　へ
  SHMZ &H2 5000                           'Ｚ方向、スピード設定
  SHOM &HAA 5000                          'ＸＹＵ方向、スピード設定
  HOMZ &H1 200                            'Ｚ原点近傍まで動作
  HOME &H15 200                           'ＸＹＵ原点近傍まで動作
'
'        ＯＲＩＧＩＮへ
  SHMZ &H2 100                            'Ｚ方向、スピード設定
  SHOM &HAA 100                           'ＸＹＵ方向、スピード設定
  HOMZ &H3 0                              'Ｚ原点復帰
  HOME &H3F 0                             'ＸＹＵ原点復帰
'
  PALET1 P(1) P(2) P(3) P(4) 5 6          'パレット宣言
  FEED 0                                  'スピード設定
  FEDD 5                                  'Ｚ軸スピード設定
'
  DO
    FOR k=1 TO 30
      JUMP PL1(k)                         'パレットへ移動
      ON 0                                'ワーク吸着
      TIME 500
      IF SW(-3)==1 THEN                   'コンベアーがＲＥＡＤＹな
        JUMP P(4)                         'ら直接Ｐ（４）
        ELSE                              'ＲＥＡＤＹでなければＰ（４）
        JUMP P(4) -10                     '上で待機
        WAIT SW(-3)==1                    'ＲＥＡＤＹ待ち
        GO P(4)                           'Ｐ（４）へ移動
      END_IF
      OFF 0                               'ワーク実装
      TIME 500
      MOVZ 0                              'Ｚ軸上昇
      OFF -3                              'コンベアーへ実装完了
    NEXT k
  LOOP
'**************************
'   除材エレベーター
'**************************
*conveyer
  DO
    WAIT SW(-1)==1                        '給材エレベーターＲＥＡＤＹ待ち
    ON 4                                  'モーター回転
    ON 1                                  '基板押し出し
    WAIT SW(193)==1
    OFF 1
    WAIT SW(192)==1
    OFF -1                                '基板押し出し完了
    ON 2                                  'ストッパー突出
    WAIT SW(196)==1                       '基板検出
    TIME 500
    OFF 4                                 'モーター停止
    ON -3                                 'ロボットへＲＥＡＤＹ
    WAIT SW(-3)==0                        'ロボット作業完了待ち
    ON 4
    OFF 2
    TIME 500
```

```
    WAIT SW(197)==1                    'コンベアー末端基板検出
    WAIT SW(-2)==1                     '除材エレベーターＲＥＡＤＹ待ち
    ON 3                               '基板収納
    WAIT SW(195)==1
    OFF 3
    WAIT SW(194)==1
    OFF -2                             '基板収納
  LOOP
```

７）Ｑ＿ＰＡＵＳＥモードでのパルス発生の停止

非常停止スイッチ、ポーズスイッチ、サイクル停止スイッチ等を常時監視する場合には監視専用のタスクを設け、動作実行タスクをコントロールするのが合理的です。動作実行タスクの制御がＩ／Ｏだけならばプログラム的にはＰＡＵＳＥやＱＵＩＴはいつでもできます（機械のタイミング等を考慮する必要はあります）。しかし、パルス発生を含むタスクをＰＡＵＳＥ・ＱＵＩＴするのにはパルス発生の有無を確認する必要があります。そしてパルス発生中であればそれを停止させてからＰＡＵＳＥ・ＱＵＩＴを行います。従来これらの処理はＳＴＯＰ、ＤＳ＿ＰＧやＢＳＹ（）関数を組み合わせて実施しましたが、ＲＥＶ１．２以降ではＱ＿ＰＡＵＳＥモードに設定する事により、ＳＴＯＰコマンドだけでパルスの停止やその確認、そしてタスクのＰＡＵＳＥまで一括で処理をします。次のプログラムはＱ＿ＰＡＵＳＥモード下でスイッチの監視とパルス発生タスクのＰＡＵＳＥ・ＱＵＩＴを行っています。

```
        TIME 100
        SETIO
        Q_PAUSE                        'Q＿ＰＡＵＳＥモード宣言
'       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します。
        QUIT 1
        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に与えられたパ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ブルーチンです
        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

前記のプログラムを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，パルス再出力
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
```

３．４　ＲＳ－２３２Ｃのプログラム

１）ＭＰＣ同士での通信例

最近のシステムはますますインテリジェント化され画像処理装置、表示装置、パソコン、ＬＡＮなどの外部機器と通信で結ばれ多くのデータの授受が行われています。ＭＰＣ－６８４にはＲＳ－２３２Ｃポートが３チャンネル装備されておりそのうち２チャンネルをユーザーが使用できます。ＰＲＩＮＴ文やＩＮＰＵＴ文による文字列単位の送受信、ＩＮＰ$・ＰＵＴ#によるキャラクター単位の送受信、さらにＬＥＮ・ＳＴＲ・ＳＴＲＣＰＹ・ＶＡＬなどの関数・コマンドによる柔軟な文字列処理で複雑な通信フォーマットにも対応できます。次のプログラムはＭＰＣ－６８４２台のＣＨ０同士を接続し、ＭＰＣ－Ａから送信されるデータをＭＰＣ－Ｂで解析しています。

ＭＰＣ－Ａ（送信側）

送信フォーマット　［ＳＸ］［変数名］［＊］［データ］［＊］［チェックサム］［ＥＸ］［ＥＴ］

概要　最初にＥＱコードを送信しＡＫコードの応答を待つ（受信側のＲＥＡＤＹ確認）。その後フォーマットに従い変数名・データ・チェックサムを送信する

```
'  CNFG#0 "9600b8pns1XON"
```

```
PRINT#0 CHR$(&H5)                    'ＥＱコード送信
WAIT INP$#0(1)==CHR$(&H6)            'ＡＫコード受信待ち
  ABCDE=12356
  FGHIJ=98765
  A$="FGHIJ"                         'Ａ＄に文字列ＦＧＨＩＪを入れる
  B$=STR$(FGHIJ)                     'Ｂ＄にＦＧＨＩＪの値を文字列として格納
PRINT#0 CHR$(&H2) A$ "*" B$ "*"      'ＳＸコードに続きデータを出力
  CS=0
FOR I=0 TO LEN(A$)                   'Ａ＄のチェックサムを計算
  STRCPY A$ C$ I 1
   CS=CS+ASC(C$)
NEXT I
FOR I=0 TO LEN(B$)                   'Ｂ＄のチェックサムを計算
  STRCPY B$ C$ I 1
    CS=CS+ASC(C$)
NEXT I
  CS=CS+ASC("*")
  CS$=HEX$(CS&HFF)                   'チェックサムの値を文字列としてＣ＄に格納
PRINT#0 CS$ CHR$(&H03)               'Ｃ＄とＥＸコードを出力
PRINT#0 CHR$(&H4)                    'ＥＴコードを出力
```

ＭＰＣ－Ｂ（受信側）

受信フォーマット　［ＳＸ］［変数名］［＊］［データ］［＊］［チェックサム］［ＥＸ］

概要　ＥＱコードを受信したらＡＫコードを返してＲＥＡＤＹ状態を知らせ受信を開始する。ＥＸコード受信によりチェックサムデータを比較、ＥＴコードで終了する。

```
  CNFG#0 "9600b8pns1XON"
'
  WAIT INP$#0(1)==CHR$(&H5)                  'ＥＱコード待ち
  PRINT#0 CHR$(&H6)                          'ＡＫコード出力
    A$=""
*LOOP
  WAIT LOF(0)<>0
    A$=INP$#0(1)
  SELECT_CASE A$                             'ＳＸコードなら受信ＥＴコードなら終了
        CASE CHR$(&H2) : GOTO *CASE1
        CASE CHR$(&H4) : GOTO *CASE2
  END_SELECT
  GOTO *LOOP
*CASE1
    C=0
    A$=""
    B$=""
    C$=""
  DO UNTIL A$=="*"                           '変数名を受信
     B$=B$+A$
     A$=INP$#0(1)
     C=C+ASC(A$)                             'チェックサムデータ加算
  LOOP
  PRINT B$
    A$=""
  DO UNTIL A$=="*"                           'データを受信
     C$=C$+A$
     C=C+ASC(A$)
     A$=INP$#0(1)
  LOOP
  PRINT C$
    C=C&&HFF
    D$=HEX$(C)                               'チェックサムデータを文字列に変換
    A$=INP$#0(2)                             'チェックサムデータを受信
  PRINT D$ A$
  WAIT INP$#0(1)==CHR$(&H03)                 'ＥＸコード待ち
  IF D$<>A$ THEN
    PRINT "チェックサムエラー"
    PRINT#0 CHR$(&H15)                       'チェックサムエラーならＮＫコード出力
    GOTO *LOOP
  END_IF
  SELECT_CASE B$                             '変数にデータを格納
    CASE "ABCDE"
      ABCDE=VAL(C$)
    CASE "FGHIJ"
      FGHIJ=VAL(C$)
    CASE_ELSE
    PRINT "カイドクデキマセン"
  END_SELECT
  GOTO *LOOP
```

```
*CASE2
  PRINT "ABCDE=" ABCDE
  PRINT "FGHIJ=" FGHIJ
```

２）ＲＳ－２３２Ｃのエラー処理

ＲＳ－２３２Ｃで通信を行う場合にはデータの信頼性が重要になります。どんなに高度な周辺機器と接続されてもデータがデタラメでは装置として正常に稼働しません。そこで通信が正常に行われているかを常時確認する必要があるわけですが、高速で複雑な通信内容を管理するのは骨のおれることです。前記のサンプルプログラムでのチェックサムも通信内容の確認の１つで、１キャラクター受信のたびにそのチェックサムを計算してチェックサムデータと照合してデータが正常かどうかを確認しています。このようにプログラムによるチェックのほか、ＭＰＣ－６８４ではＲＳＥ（ｎ）関数によりＲＳ－２３２Ｃの受信状態を監視してソフト上で再送信要求などのエラー処理を行うことができます。

ＲＳＥ（ｎ）関数

ｎ＝チャンネルナンバー　０　または　２

返される値／内容／エラー原因として考えられるもの

０	正常終了	
１	パリティーエラー	ノイズの混入
２	オーバーラン	通信速度が速すぎる（ボーレート不一致）
４	フレーミングエラー	ケーブル不良・初期化異常
８	ブレークコード検出	異常データ

次のプログラムはＭＰＣ－６８４同士を接続し簡単な文字列を送受信したものですが、ＣＮＦＧでの初期化に誤りがありエラーとして検出されました。エラーが発生した場合はＣＮＦＧコマンドで初期化し直します。

送信側

```
  CNFG#2 "9600b8pns1NONE"                    'ＲＳ－２３２Ｃ　ＣＨ２　初期化
   TIME 50
  PRINT#2 "ABC" CHR$(&HD)
  END
```

受信側

```
*LOOP
  CNFG#2 "9600b7pns1NONE"                    'データビット長が違う
*LOOP1
  INPUT#2 A$
  SELECT_CASE RSE(2)                         'ＲＳ－２３２Ｃ　CH2チェック
    CASE 0 : PRINT A$ : GOTO *LOOP1
    CASE 1 : PRINT "ﾊﾟﾘﾃｨｰｴﾗｰ" : GOTO *LOOP     'エラーが起きたら
    CASE 2 : PRINT "ｵｰﾊﾞｰﾗﾝ" : GOTO *LOOP       'ＣＮＦＧで初期化
    CASE 4 : PRINT "ﾌﾘｰﾐﾝｸﾞｴﾗｰ" : GOTO *LOOP
    CASE 8 : PRINT "ﾌﾞﾚｰｸｺｰﾄﾞ 検出"   : GOTO *LOOP
    CASE_ELSE  : PRINT "???": GOTO *LOOP
  END_SELECT
  END

  RUN
  ﾌﾚｰﾐﾝｸﾞｴﾗｰ
```

ＲＳＥ（ｎ）関数では受信時のチェックはできますが、送信時のチェックはできません。また、実際にはＲＳＥ（ｎ）が頻繁に活躍するのでは困りものです。ＲＳ－２３２Ｃのエラーに限らずトラブルは元から絶たなければ根本的な解決にはなりません。特に、別電源のパソコンや画像処理装置などの外部機器との接続にはフレームグランドの共通化による浮遊電圧の防止や配線の経路・長さ・結線の方法、そして通信フォーマットといったハード・ソフト両面での仕様を考慮して、場合によってはアイソレーターを使用するなどの対策が必要になることもあります。

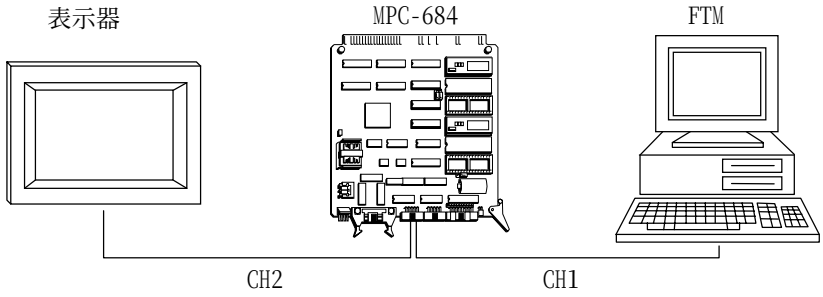
３）タッチパネル（デジタル社製）の使用例

グラフィカルユーザーＩ／Ｆとしてタッチパネルが普及しています。ＭＰＣ－６８４シリーズではＭＢＫ－６８でＭＥＷ　ＮＥＴ－ＦＰプロトコルをサポートし、通信を意識することなくＩ／Ｏ機器と同レベルでＧＰを制御することができます。次は、ＲＳ－２３２Ｃ通信例としてのＧＰ制御です。

（ＭＢＫ－６８については第７章ハードリファレンス、製品別マニュアルをご覧下さい。）

☆使用機器
ＭＰＣ－６８４
パソコン：ＤＯＳ／Ｖ
ターミナルソフト：ＦＴＭＷ

表示器
表示器：デジタル社　ＧＰシリーズ
ＧＰ作画支援ソフト：ＧＰ－ＰＲＯⅢ



【機器接続図】

☆ＲＳ－２３２Ｃの結線

３線式クロスで結線しました。
☆ＲＳ－２３２Ｃの初期化
９６００ｂｐｓ、８ビット、パリティー無し、ストップビット１、ＸＯＮ／ＯＦＦ
ＭＰＣの初期化は　CNFG#2 "9600b8pns1XON"　です。表示器側の設定を確認して下さい。表示器とやりとりするデータはすべてＡＳＣＩＩ文字列です。たとえば０を送るには’０’（&Ｈ３０）になります。

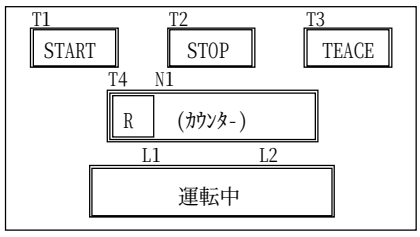
☆ＭＰＣのＲＳ－２３２Ｃ送受信バッファの確認の方法
プログラムのデバッグ中にＲＳ－２３２Ｃから送信したり受信したりしたデータを確認したい時があります。それにはＲＳコマンドを使用します。バッファの内容をＡＳＣＩＩコードで表示します。

```
#PRINT#2 CHR$(&H1B) "R00100001\r"
#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 1B<41 30 30 30 31 0D    <EC>A0001<CR>
ＲＳ送信                                        "<"記号は残バッファ先頭
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 1B 52 30 30 31 30 30 30 30 31 0D    <EC>R00100001<CR>
#
```

この送受信バッファは前記ＣＮＦＧ＃２コマンドでクリアーされます。

☆表示器の画面
スイッチの入力、メッセージ表示、テンキーの使用、画面切り替えを想定して２つの画面をつくりました。画面の編集はデジタル社の専用ソフトで作成しました。

【画面１】
「ＳＴＡＲＴ」、「ＳＴＯＰ」、「ＴＥＡＣＨ」のスイッチと、生産個数を表示するためのカウンターとリセットスイッチを設けてあります。その下には「運転中」と「停止」の

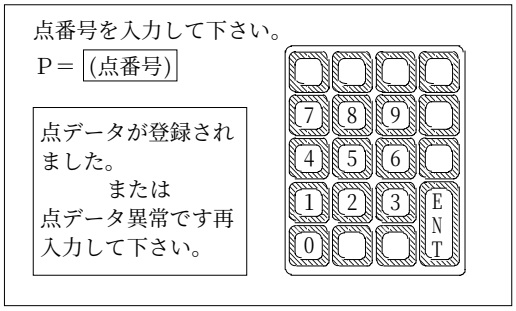


RはカウンターのＲＥＳＥＴ　ＳＷ

【画面１の構成】

文字を表示するようにします。

【画面２】
点番号の入力のためのテンキーと正常な入力か否かを表示するメッセージを表示します。



テンキーで入力された数字は(点番号)に表示されます(3桁)。

【画面２の構成】

☆動作
①「ＳＴＡＲＴ」を押すとＭＰＣは自動運転を行います。「運転中」の表示をして、カウンターの値を加算していきます。
②「ＳＴＯＰ」を押すと停止します。
③「ＴＥＡＣＨ」を押すと表示が画面２に切り替わります。テンキーで点番号の設定を行います。
④「テンキー」テンキーの数字を押すと左側に入力した数字を表示し、「ＥＮＴ」キーで確定してＭＰＣがそれを現在点とします。もしその値が１～２００であれば「点データが登録されました」、それ以外なら「点データ異常です。再入力してください。」を表示します。正常な入力ならば画面１に戻ります。異常な入力ならば画面２のまま再入力になります。

【画面１】
表示器システムエリア

ＳＷ名称	タグネーム	アドレス
「ｽﾀｰﾄ」	T1	001600
「STOP」	T2	001601
「TEACH」	T3	001602
(ｶｳﾝﾀｰﾘｾｯﾄ)	T4	001901

表示文字	タグネーム	アドレス
「運転中」	L1	001700
「停止」	L2	001701

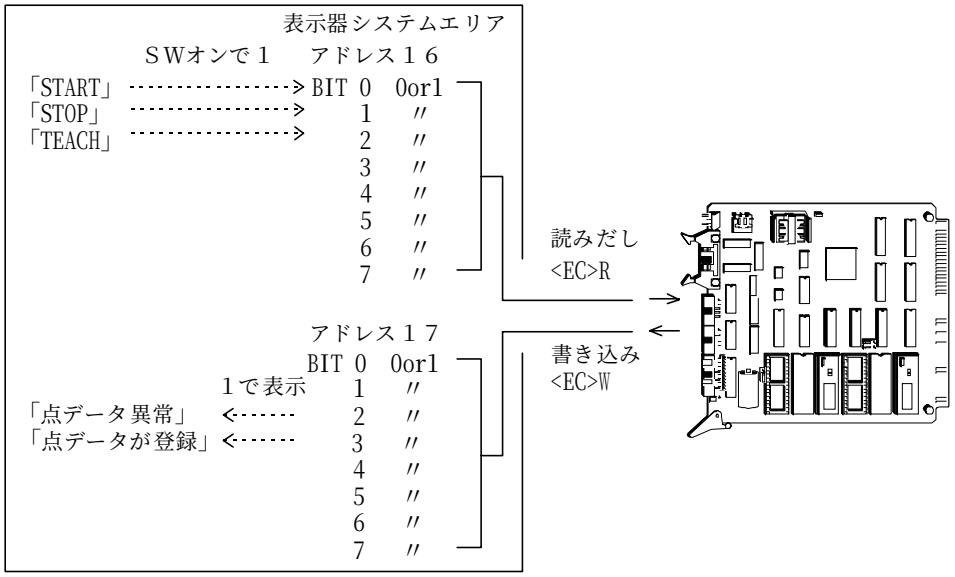
☆表示器システムエリアのアドレス設定
スイッチの確認、メッセージ表示、ページ切り替えは全て表示器のシステムエリアを介して行います。スイッチが押されたらそれに対応したシステムエリアのビットが０から１となる様にします。ＭＰＣは通信でそれを読みだしてスイッチの状態を知ることができます。メッセージも割り当てられたシステムエリアのビットが０から１になると画面に表示される様にしておきます。ＭＰＣは通信でそのビットを１にセットするのです。
「START」　を押すとアドレス１６の第０ビットが１となります。
「STOP」　を押すとアドレス１６の第１ビットが１となります。
「TEACH」　を押すとアドレス１６の第２ビットが１となります。
(ｶｳﾝﾀｰﾘｾｯﾄ)　を押すとアドレス１９の第２ビットが１となります。

【画面２】
表示器システムエリア

ＳＷ名称	タグネーム	アドレス	
（テンキー）		001800	ライブラリーを使用。
「ＥＮＴ」	T1	001900	テンキーのＥＮＴキー

表示文字	タグネーム	アドレス	
（点番号）	K1	0020	テンキーを押すとその値が表示される
「点データ異常・・」	L1	001702	
「点データが登録・・」	L2	001703	

これらのＳＷは押されている間だけ１になります。
アドレス１７の第０ビットを１にすると「運転中」が表示されます。
アドレス１７の第１ビットを１にすると「停止」が表示されます。



【読み込み・書き込みイメージ】

テンキーから入力されたデータはＥＮＴキーで確定されます。「ＥＮＴ」を押すとアドレス１９の第０ビットが１となります。また、そのときのデータがアドレス 0020 に入ります。
アドレス１７の第２ビットを１にすると「点データが登録‥」が表示されます。
アドレス１７の第３ビットを１にすると「点データ異常‥」が表示されます。
☆画面１のＳＷの状態を知るには。
リードコマンドでシステムエリアからの読みだしをします。リードコマンドを送信すると対応するアドレスのデータが返ってきます。読みだしのフォーマットは次の通りです。

```
<EC>Raaaadddd<CR>
```

＜ＥＣ＞はＥＳＣコード（&Ｈ１Ｂ)、Ｒは”Ｒ”(&Ｈ５２)、ａａａａはアドレス、ｄｄｄｄは読みだすデータの数です。　ＭＰＣでは

```
PRINT#2 CHR$(&H1B) "R00100001\r"
```

を送ります。ここで注意しなければならないのがアドレスは１６進数で表すことです。アドレス１６（ＤＥＣ）ならば１０（ＨＥＸ）です。返ってくるデータのフォーマットは

```
<EC>Adddd<CR>
```

です。＜ＥＣ＞はＥＳＣコード（&Ｈ１Ｂ)、Ａは”Ａ”(&Ｈ４１)、ｄｄｄｄはデータです。ＭＰＣでは

```
INPUT#2 A$
```

で読み込みます。ＩＮＰＵＴ文は＜ＣＲ＞まで一括で読み込みます。もし「ＳＴＡＲＴ」ＳＷが押されていれば　＜ＥＣ＞Ａ０００１＜ＣＲ＞　と返ってきます。「ＳＴＯＰ」が押されていれば＜ＥＣ＞Ａ０００２＜ＣＲ＞,「ＴＥＡＣＨ」ならば　＜ＥＣ＞Ａ０００４＜ＣＲ＞です。何もおされていなければ＜ＥＣ＞Ａ０００００＜ＣＲ＞です。ここで注意しなければならないことは先頭にＥＳＣコードが入っているためこのままでは変数に変換することが出来ません。ＰＲＩＮＴ　Ａ＄と実行しても表示されません。そこで次の様にして変数に変換します。

```
INPUT#2 A$                    'データをＡ＄にいれます。
STRCPY A$ B$ 2                'Ａ＄の３文字以降をＢ＄にコピーします。
DATA=VAL(B$)                  '文字列Ｂ＄を数字に変換します。
```

例えばこの変換で「ＳＴＡＲＴ」が押された場合、変数ＤＡＴＡは１になります。

☆画面１の文字表示をするには。

ライトコマンドでシステムエリアへの書き込みをします。書き込みのフォーマットは次の通りです。

```
<EC>Waaaadddd<CR>
```

＜ＥＣ＞はＥＳＣコード、Wは”Ｗ”（&Ｈ５７）、ａａａａはアドレス、ｄｄｄｄはデータです。「運転中」はアドレス１７の第０ビットを１にすれば表示されますから、

```
PRINT#2 CHR$(&H1B) "W00110001\r"
```

とします。ここでもアドレス指定は１６進数です。（ＤＥＣ）１７は（ＨＥＸ）１１です。「停止」を表示するにも同様に

```
PRINT#2 CHR$(&H1B) "W00110002\r"
```

です。

☆カウントの表示
ここでは実際に何かを数えるという訳ではありませんが､一定時間毎に加算していく様にしています｡このカウンターは表示器が加算してくれるのではなく、ＭＰＣから送られてくるデータを表示するだけです。

```
    I1=0
  FOR I=0 TO 9999              '0～９９９９までを表示します。
    TIME 50
      TMP$=STR$(I1)
      COUNT$=""
      STRCNT=4-LEN(TMP$)       '4byteのデータにするため"0 "を連結します。
    DO WHILE STRCNT<>0         'カウント値が"１２ "なら"００１２ "にします。
        COUNT$=COUNT$+"0"
        STRCNT=STRCNT-1
    LOOP
      COUNT$=COUNT$+TMP$
    PRINT COUNT$
     COUNT$=CHR$(&H1B)+"W0019"+COUNT$+"\r"    'ついでにＥＳＣコードも＜ＣＲ＞も
    PRINT#2 COUNT$                            連結
      I1=I1+1
    GOSUB *COUNTER_RESET       'カウンターリセットボタン入力確認サブルーチン
    GOSUB *READ                '「停止」ＳＷ確認サブルーチン
    IF DATA==2 THEN : RETURN  : END_IF
  NEXT I
  RETURN
*COUNTER_RESET
  PRINT#2 CHR$(&H1B) "R00130001\r"  'リセットボタンの入力確認
  INPUT#2 A$
  STRCPY A$ B$ 2
  IF VAL(B$)<>2 THEN : RETURN  : END_IF
    I1=0                       'リセットが押されたらカウント変数を０クリアー
  RETURN
```

☆画面の切り替え
画面の切り替えはアドレス１５にページ番号を書き込みます。

```
PRINT#2 CHR$(&H1B) "W000F0002\r"      '２ページ(画面２)へ切り替え
PRINT#2 CHR$(&H1B) "W000F0001\r"      '１ページ(画面１)へ切り替え
```

☆画面２の操作
テンキーを使うにはテンキーが設定されているアドレス１８の第０ビットに１をセットしてイネーブル状態にします。

```
PRINT#2 CHR$(&H1B) "W00120001\r"
```

テンキーで入力したデータはアドレス２０に入っています。ＭＰＣは「ＥＮＴ」キーの押されるのを待ってこの２０のデータを読み込みます。

```
(「ＥＮＴ」キーの入力待ち)
DO
```

```
  PRINT#2 CHR$(&H1B) "R00130001\r"     'アドレス１９の読み込み
  INPUT#2 A$
  STRCPY A$ B$ 2
LOOP UNTIL VAL(B$)<>0
点データ入力
PRINT#2 CHR$(&H1B) "R00140001\r"      'アドレス２０の読み込み
INPUT#2 A$
STRCPY A$ B$ 2
POINT=VAL(B$)                         '変数ＰＯＩＮＴにテンキーのデータがはいります。
```

☆画面２の文字表示

前記POINTの値が１～２００ならば「点データが登録されました。」を表示します。それ以外の数値なら「点データ異常です。再入力して下さい。」を表示します。

```
PRINT#2 CHR$(&H1B) "W00110008\r"      '登録
PRINT#2 CHR$(&H1B) "W00110004\r"      '異常
```

☆プログラム全体

前記プログラムをまとめたものです。(*TASK1、*TASK2はﾀﾞﾐｰです）このプログラムはスイッチの監視のために常にタスク０が通信文でＬＯＯＰしています。実際の装置ではもっと複雑な制御が要求されるので、この方法では不具合なことがあるかもしれません。表示器についているＩ／Ｏの機能を利用して、スイッチが押された時だけ通信する様に工夫すれば良いと思いますが...。

```
  FORK 1 *TASK1
  FORK 2 *TASK2
  CNFG#2 "9600b8pns1XON"
  PRINT#2 CHR$(&H1B) "W000F0001\r"
  GOSUB *CLS1
  DO
    DO
      GOSUB *READ
    LOOP UNTIL DATA<>0
    SELECT_CASE DATA
      CASE 1 : GOSUB *START
      CASE 4 : GOSUB *TEACH
    END_SELECT
    DO
      GOSUB *READ
    LOOP UNTIL DATA==0
  LOOP
*READ
  PRINT#2 CHR$(&H1B) "R00100001\r"
  INPUT#2 A$
  STRCPY A$ B$ 2
    DATA=VAL(B$)
  RETURN
*START
  GOSUB *CLS1
  PRINT#2 CHR$(&H1B) "W00110001\r"
  GOSUB *COUNT
  RETURN
*TEACH
  PRINT#2 CHR$(&H1B) "W000F0002\r"
  GOSUB *TENKEY_READ
  TIME 1000
  PRINT#2 CHR$(&H1B) "W000F0001\r"
  RETURN
*CLS1
  PRINT#2 CHR$(&H1B) "W00110000\r"
  RETURN
*COUNT
    I1=0
  FOR I=0 TO 9999
    TIME 50
      TMP$=STR$(I1)
      COUNT$=""
      STRCNT=4-LEN(TMP$)
    DO WHILE STRCNT<>0
       COUNT$=COUNT$+"0"
       STRCNT=STRCNT-1
    LOOP
      COUNT$=COUNT$+TMP$
```

```
    PRINT COUNT$
      COUNT$=CHR$(&H1B)+"W0019"+COUNT$+"\r"
    PRINT#2 COUNT$
      I1=I1+1
    GOSUB *COUNTER_RESET
    GOSUB *READ
    IF DATA==2 THEN : RETURN  : END_IF
  NEXT I
  RETURN
'
*COUNTER_RESET
  PRINT#2 CHR$(&H1B) "R00130001\r"
  INPUT#2 A$
  STRCPY A$ B$ 2
  IF VAL(B$)<>2 THEN : RETURN  : END_IF
    I1=0
  RETURN
*TENKEY_READ
  PRINT#2 CHR$(&H1B) "W00120001\r" 'TENKEY ENB
  DO
    PRINT#2 CHR$(&H1B) "R00130001\r" 'ENT KEY READ
    INPUT#2 A$
    STRCPY A$ B$ 2
  LOOP UNTIL VAL(B$)<>0
  PRINT#2 CHR$(&H1B) "R00140001\r" 'TEN KEY DATA READ
  INPUT#2 A$
  PRINT A$
  STRCPY A$ B$ 2
    POINT=VAL(B$)
  IF POINT<=0 OR POINT>200 THEN
    PRINT#2 CHR$(&H1B) "W00110004\r"
    TIME 500
    PRINT#2 CHR$(&H1B) "W00110000\r"
    GOTO *TENKEY_READ
    ELSE
    PRINT#2 CHR$(&H1B) "W00110008\r"
    TIME 500
    PRINT#2 CHR$(&H1B) "W00110000\r"
  END_IF
  RETURN
*TASK1
  DO
    FOR J=0 TO 23
      ON J
      TIME 50
      OFF J
      TIME 50
    NEXT J
  LOOP
*TASK2
  DO
    FOR K=23 TO 47
      ON K
      TIME 50
      OFF K
      TIME 50
    NEXT K
  LOOP
```