Technical Information		Ref No: ti2k-100803-1	Last Modify 131203
Title	MPC-684 と MPC-2000 シリーズのプログラム的相違点		

MPC-684 と MPC-2000 のプログラムには若干の相違点があります。MPC-2000 独自の記述方法もあります。本資料は MPC-684 から MPC-2000 へプログラムを移植する際の相違点をピックアップしました。(随時更新)

※MPC-2000 シリーズとは MPC-1000,N816,2000,2200 を指し、本文ではこれらをまとめて”MPC-2000”と表します。

ファイル拡張子の違い (F2K, P2K)

パソコンでの MPC-684 のプログラムの拡張子は "F68" ですが、MPC-2000 は "F2K" です。FTMW の読み込み(LOAD)・保存(SAVE)対象は"F2K"になります。外部エディタで作成・編集する場合は、テキストファイル形式で、拡張子を"F2K"としてください。
MPC-2000 の点データファイルの拡張子は"P2K"です。データの書式は MPC-684 と同じですが、要素が X=Y=U=Z=0 の場合は読み込み・保存されないので作業時間が短縮されます。

バージョンの確認 (VER)

VER コマンドで MPC-2000 のバージョンを表示します。FTMW とリンクする時も表示されます。

・MPC-2000 の場合
```
#VER
MPC-2000*(SH7030) BL/I 1.12_92 2012/02/20  /* MPC ハード(CPU),インタプリタバージョン,改版デート
 All Rights reserved. ACCEL Corp. .T32     /* コピーライト,タスク数
#
```

・MPC-1000 の場合
```
#VER
MPC-1000(SH7030) BL/I 1.12_03 2009/11/13
 All Rights reserved. ACCEL Corp. .T32
#
```

・MPC-1000 で USB ポートが ON になっている場合
```
#ON_USB                                  /* USB メモリポートをオンする
#VER
MPC-1000(SH7030) BL/I 1.12_03 2009/11/13
 All Rights reserved. ACCEL Corp. .T32
   +The USB Activated on TASK_29+        /* USB メモリポートがタスク 29 でアクティブ
#
```

定数、変数の確認 (VLIST)

VLIST コマンドで現在の定数、変数、配列を表示します。コマンドはスモールファルファベットで記述しても MPC が自動変換します。定数、変数はラージ/スモールを区別します。きっちり書いてください。
語尾に”_”(アンダーバー)を付けるとタスク変数(ローカル変数)になります。

定数を変更しようとするとエラーになります。

```
#IN2=12
        この変数は定数化されています:20
#
```

下記は MPC-2100(SH7030) BL/I 1.12_01 2009/11/09 のリストです。末尾の I: sol, I: j, L: i_, ary はプログラムの実行により追加された要素です。

```
10        CONST   sol 0
20        DIM   ary(11)
30       j=0
40        FOR   i_=0 TO 10
50       ary(i_)=j
60       j=j+1
70        NEXT
#RUN
#VLIST
■定数リスト
VOID            X_A             Y_A             U_A
```

Z_A	ALL_A	X_E	Y_E
U_E	Z_E	VOID_X	VOID_Y
VOID_Z	VOID_U	ALL_A	STP_I
STP_D	ALLOW	N_SDX	N_SDY
N_SDU	N_SDZ	P_SDX	P_SDY
P_SDU	P_SDZ	INP_ON	INP_OFF
INP_NO	ALM_ON	ALM_OFF	ALM_NO
PHASE1	PHASE2	PHASE4	UP_DWN
MD_2PLS	MD_DPLS	LMT_ON	LMT_OFF
SLMT_ON	SLMT_OFF	IN0_ON	IN0_OFF
IN1_ON	IN1_OFF	IN2_ON	IN2_OFF
IN3_ON	IN3_OFF	CW	CCW
XIN0	XIN1	XIN2	XIN3
XINP	XALM	YIN0	YIN1
YIN2	YIN3	YINP	YALM
UIN0	UIN1	UIN2	UIN3
UINP	UALM	ZIN0	ZIN1
ZIN2	ZIN3	ZINP	ZALM
SLMTp	SLMTn	LMTp	LMTn
ALM	EMG	IN0	IN1
IN2	IN3	CLR_ER	X_C
Y_C	Z_C	U_C	Lng
Wrd	Int	Ub	Lb
POS_L	NEG_L	NIL	SA0
SA1	SA2	SA3	SA4
SA5	SA6	SA7	SA8
SA9	SA10	SA11	SA12
SA13	SA14	SA15	SA0_B
SA1_B	SA2_B	SA3_B	SA4_B
SA5_B	SA6_B	SA7_B	SA8_B
SA9_B	SA10_B	SA11_B	SA12_B
SA13_B	SA14_B	SA15_B	AVOID
EOL	TMOUT	CHR_C	CLR_BUF
CMP_PLS	CMP_CNT	LONG_PRG	C_MORE
C_LESS	COM	SET_SF	RTS
RS485	USB	USB0	USB1
USB2	SACL	CompoWay	COMPOWAY
B7N	B7E	B7O	B8E
B8O	AD7890_10	AD0	AD1
_NEXT	OFF	PGA	PGB
ON_USB			

■変数リスト

L: timer_	L: ptr_	L: rse_	L: err_
R: PG_TASK0	R: SEC	R: timer	R: MBK_ERR
R: MBK_CMD	S: VER$	R: Nm	R: Gc
R: Xn	R: Yn	R: Zn	R: An
R: Fd	R: Kn	R: In	R: Mc
R: Rn	R: MVTn	R: Jn	R: Xcp
R: Ycp	R: Zcp	R: Acp	R: Rmv
R: Xmp	R: Ymp	R: Zmp	R: Amp
R: Xpp	R: Ypp	R: Zpp	R: App
R: Fax	R: Sfx	R: Sfy	R: Sfz
R: Sfa	R: CUM_PNT	R: CUM_SRC	R: CUM_NUM
R: CUM_CNT	R: CUM_ERR	R: CUM_TASK	S: FILE$
S: FILE1$	S: FILE2$	R: MF0	R: MF1
R: MF2	R: MF3	R: MF4	R: MF5
R: MF6	R: MF7	R: CHK_SUM	R: SYSCLK
R: TASKn	R: V_PGA	R: V_PGB	I: sol
I: j	L: i_		

■配列リスト

ary

#

タスクの再起動 (FORK、QUIT_FORK)

MPC-684 は既に FORK しているタスクを再 FORK してもエラーにはなりませんが、MPC-2000 はランタイムエラーになります。

```
10        DO
20         FORK   1 *TASK1
30         WAIT   SW(192)==0
40         WAIT   SW(192)==1
50        LOOP
60      *TASK1
70        DO
80         ON   0 : TIME   100 : OFF   0 : TIME   100
90        LOOP
#RUN


[20]    タスクの二重起動です:38      ← SW(192)をオン
#

10        DO
20         QUIT   1                  /* タスク停止を追加
30         FORK   1 *TASK1
40         WAIT   SW(192)==0
50         WAIT   SW(192)==1
60        LOOP
70      *TASK1
80        PRINT   "TASK1 START"
90        DO
100        ON   0 : TIME   100 : OFF   0 : TIME   100
110       LOOP

2010/04/05 コマンド追加
  書式  QUIT_FORK n *LABEL
  FORK と同じ機能ですが、相手タスクがすでに起動されていてもエラーを発生しません。
```

RS-232C 文字列、コントロールコードの出力 (PRINT#)

MPC-2000 の RS-232C 出力は PRINT#のみです。文字列は MPC-684 と同様に出力します。MPC-684 の PUT#で 1 文字づつ出力している場合、MPC-2000 には PUT#は無いので PRINT#で出力します。

```
10        CNFG#   2 "9600b7pes1"            /* CNFG#と CH 番号の間にスペース
20        PRINT#   2 "ON¥r"                 /* "ON<CR>"
30        PRINT#   2 CHR$(&H0000001B) "¥r"  /* &H1B"<CR>"
40      '
50       SND$=CHR$(&H0000001B)+"1"          /* 文字列変数代入
60        PRINT#   2 SND$                   /* &H1B"1"
70      '
80        PRINT#   2 CHR$(6)                /* &H06

NULL キャラクタ出力を出力するには  STR_LEN|1 を与えます。
PRINT# 1 STR_LEN|1 CHR$(0)                 /* CH1 から&H00 出力
```

RS-232C 1 文字受信 (INPUT#)

MPC-2000 の RS-232C 入力は INPUT#のみです。MPC-684 の INP$#で文字数を指定して入力している場合は INPUT#に文字数を設定して受信します。

```
10        CNFG#   2 "9600b7pes1"
```

```
20        INPUT#   2 CHR_C|1 A$              /* CH2 1文字受信→A$
30        IF   ASC(A$)==&H00000006 THEN  : PRINT   "ACK" : END_IF
40        END
#RUN

 ACK        ← 別のパソコンからCH2 に&H06を送信
#
```

RS-232C受信文字列から数値データを取り出す（INPUT#, VAL, GET_VAL）

1) MPC-2000のINPUT#はターミネータを指定できます(デフォルトはCR)。
次はLFをターミネータとして受信、VAL()で数値を切り出す例です。VAL()の戻り値取得は普通の変数で可。

```
10        CNFG#   2 "9600b7pes1"
20        INPUT#   2 EOL|10 TMOUT|5 C$      /* LFの前まで→C$、 TMOUT 5SECで結果はrse_
30        PRINT   C$                        /* 受信文字列
40       AC1=VAL(C$)                        /* 最初の数値を取得
50       AC2=VAL(0)                         /* 次の数値を取得
60        PRINT   AC1 AC2                   /* 取り出した数値
70        PRINT   LOF(2)                    /* バッファ残数
#RUN

 WT,-123456.7 g     ←別のPCから "WT,-123456.7 g<CR><LF>" と送信
 -123456 7
 0
#
```

2) INPUT#でバッファに10文字溜まったら受信、GET_VALで文字列から数値を取り出します。GET_VALは結果を配列変数に格納します。

```
10        DIM   AC(2)
20        CNFG#   2 "9600b7pes1"
30        INPUT#   2 CHR_C|10 C$        /* 受信文字数指定
40        PRINT   C$
50        GET_VAL   C$ AC(1)            /* 数値を取り出す
60       XO=ABS(AC(1))
70        PRINT   AC(1) XO
80        PRINT   LOF(2)
#RUN

 WT,-123456         ←別のPCから "WT,-123456.7 g<CR><LF>" と送信
 -123456 123456
 6
#
```

文字列から数値データを取り出す（VALの連続取得、倍率設定）

1)数字(+-含む)以外の文字をデリミタとして取り出します。次の場合、小数点で分かれます。

```
10       C$="WT,-12345.67 g"
20       AC1=VAL(C$)              /* 最初の数値
30       AC2=VAL(0)               /* 次の数値
40        PRINT   AC1 AC2
#RUN

 -12345 67
#
```

2)引数に10,100,1000などの数値を指定すると、小数点を含む数字列を指定倍して取り出します。次の場合は100倍します。

```
10        C$="WT,-12345.67 g"
20        ptr_=C$                 /* ポインタ移動
30        AC1=VAL(100)            /* C$内から少数点を含む数値を100倍して取り出す
40         PRINT   AC1
#RUN

 -1234567
#
```

文字列から数値データを取り出す（684と2000のGET_VALの違い）

MPC-2000とMPC-684のGET_VALは引数のあたえ方が異なります。

1)MPC-684は数字列中の小数点は無視して連続した数字列と扱います。MPC-2000は三番目の引数を省略すると、小数点もデリミタとされます。

```
10         DIM   AC(2)
20        C$="WT,-12345.67 g"
30         GET_VAL   C$ AC(1)
40         PRINT   AC(1)
#RUN

 -12345        ←MPC-684では -1234567 という結果になる
#
```

2)三番目の引数に10,100,1000などの数値を設定すると、小数点を含む数字列を指定倍数の数値として配列に代入します。

```
10         DIM   AC(2)
20        C$="WT,-12345.67 g"
30         GET_VAL   C$ AC(1) 100     /* 100倍
40         PRINT   AC(1)
#RUN

 -1234567
#
```

文字列の比較（IF, SELECT_CASE)

1) MPC-684と同様にIF～END_IFが使えます。

```
10         CNFG#   2 "9600b7pes1"
20       *LP
30         INPUT#   2 CLR_BUF
40         PRINT#   2 CHR$(&H0000001B) 0
50         INPUT#   2 EOL|13 CC$
60       '
70         IF   CC$=="RDY" THEN
80          PRINT   "RDY" : GOTO  *LP
90         END_IF
100        IF   CC$=="ERR" THEN
110         PRINT   "ERR" : GOTO  *LP
120        END_IF
130        IF   CC$=="ACK" THEN
140         PRINT   "ACK" : GOTO  *LP
150        END_IF
160        PRINT   "?"
170        GOTO  *LP
#RUN
```

```
 RDY      ← 別の PC から"RDY<CR>"
 ERR      ← 別の PC から"ERR<CR>"
 ACK      ← 別の PC から"ACK<CR>"
 ?        ← 別の PC から"AAA<CR>"
```

2) SELECT_CASE 文をつかうと条件分岐がスッキリします。

```
 10        CNFG#   2 "9600b7pes1"
 20      *LP
 30        INPUT#   2 CLR_BUF
 40        PRINT#   2 CHR$(&H0000001B) 0        /* &H1B"0"
 50        INPUT#   2 EOL|13 CC$                /* <CR>の前まで→CC$
 60      '
 70        SELECT_CASE   CC$
 80         CASE   "RDY" : PRINT   "RDY" : GOTO  *LP
 90         CASE   "ERR" : PRINT   "ERR" : GOTO  *LP
100         CASE   "ACK" : PRINT   "ACK" : GOTO  *LP
110         CASE_ELSE   : PRINT   "?" : GOTO  *LP
120       END_SELECT
#RUN

 RDY      ← 別の PC から"RDY<CR>"
 ERR      ← 別の PC から"ERR<CR>"
 ACK      ← 別の PC から"ACK<CR>"
 ?        ← 別の PC から"AAA<CR>"
```

文字列の切り出し、比較 (STRCPY, PTR$, SELECT_CASE)

1) MPC-684 と同様に STRCPY で切り出すこともがきます。

```
 10        CNFG#   2 "9600b7pes1"
 20      *LP
 30        INPUT#   2 CLR_BUF
 40        PRINT#   2 CHR$(&H0000001B) 0
 50        INPUT#   2 EOL|13 CC$
 60      '
 70        STRCPY   CC$ CC1$ 0 2
 80        SELECT_CASE   CC1$
 90         CASE   "CP" : PRINT   "CP" : GOTO  *LP
100         CASE   "UP" : PRINT   "UP" : GOTO  *LP
110         CASE   "DW" : PRINT   "DW" : GOTO  *LP
120         CASE_ELSE   : PRINT   "?" : GOTO  *LP
130       END_SELECT
#RUN

 CP
 UP
 DW
 ?
```

2) MPC-2000 はポインタ ptr_ で文字を切り出すことができます。
　下の例は STRCPY と大して変わりませんが、ポインタは文字列中から特定の文字を検索するときなどに便利です。

```
 10        CNFG#   2 "9600b7pes1"
 20      *LP
 30        INPUT#   2 CLR_BUF
 40        PRINT#   2 CHR$(&H0000001B) 0
 50        INPUT#   2 EOL|13 CC$
 60      '
```

```
70        ptr_=CC$                /* CC$先頭のポインタ
80        CC1$=PTR$(2)            /* ポインタ位置から2文字コピー
90         SELECT_CASE   CC1$
100         CASE   "CP" : PRINT   "CP" : GOTO  *LP
110         CASE   "UP" : PRINT   "UP" : GOTO  *LP
120         CASE   "DW" : PRINT   "DW" : GOTO  *LP
130         CASE_ELSE   : PRINT   "?" : GOTO  *LP
140        END_SELECT
#RUN

 CP
 UP
 DW
 ?
```

3) SELECT_CASE で長さの違う文字列の比較も可能です。

```
10         CNFG#   2 "9600b7pes1"
20       *LP
30         INPUT#   2 CLR_BUF
40         PRINT#   2 CHR$(&H0000001B) 0
50         INPUT#   2 EOL|13 CC$
60       '
70         SELECT_CASE   CC$
80          CASE   "I" : PRINT   "I" : GOTO  *LP
90          CASE   "D" : PRINT   "D" : GOTO  *LP
100         CASE   "C" : PRINT   "C" : GOTO  *LP
110         CASE   "CP" : PRINT   "CP" : GOTO  *LP
120         CASE   "UP" : PRINT   "UP" : GOTO  *LP
130         CASE   "DW" : PRINT   "DW" : GOTO  *LP
140         CASE   "RDY" : PRINT   "RDY" : GOTO  *LP
150         CASE   "ERR" : PRINT   "ERR" : GOTO  *LP
160         CASE   "ACK" : PRINT   "ACK" : GOTO  *LP
170         CASE_ELSE   : PRINT   "?" : GOTO  *LP
180        END_SELECT
#RUN

 C      ←別のPCから "C<CR>"
 I      ←別のPCから "I<CR>"
 D      ←別のPCから "D<CR>"
 RDY    ←別のPCから "RDY<CR>"
 ERR    ←別のPCから "ERR<CR>"
 ACK    ←別のPCから "ACK<CR>"
 CP     ←別のPCから "CP<CR>"
 UP     ←別のPCから "UP<CR>"
 DW     ←別のPCから "DW<CR>"
 ?      ←別のPCから 上記以外
```

入力の判別 (SELECT_CASE VOID)

複数の入力の条件分岐も SELECT_CASE VOID を用いると IF～の羅列がなくなりわかりやすくなります。実行効率も上がります。

```
10         CONST   LS_0 192
20         CONST   LS_1 193
30         CONST   LS_2 194
40         CONST   LS_3 195
50         CONST   LS_4 196
60         PRX   IN(24)
70         SELECT_CASE   VOID
80          CASE   SW(LS_1)==0 : GOTO  *CASE1
```

```
90          CASE   SW(LS_2)==1 : GOTO  *CASE2
100         CASE   SW(LS_3)==1 : GOTO  *CASE3
110         CASE   SW(LS_4)==1 : GOTO  *CASE4
120         CASE_ELSE   : GOTO  *CASE5
130       END_SELECT
140     *CASE1
150       PRINT   "1" : END
160     *CASE2
170       PRINT   "2" : END
180     *CASE3
190       PRINT   "3" : END
200     *CASE4
210       PRINT   "4" : END
220     *CASE5
230       PRINT   "5" : END

#RUN

00000002       ← LS_1 のみ ON
 5

#RUN

00000000       ← 全部 OFF
 1

#RUN

00000006       ←LS_1 と LS_2 が ON
 2

#RUN

0000000A       ←LS_1 と LS_3 が ON
 3

#RUN

00000012       ←LS_1 と LS_4 が ON
 4
```

繰り返し (DO～LOOP、WHILE～WEND)

MPC-2000 の DO～LOOP に条件判断 UNTIL、WHILE をつけることはできません。BREAK でループから抜けます 。WHILE～WEND は MPC-684 と同様です。

繰り返し文からの脱出 (BREAK)

MPC-684 の BREAK コマンドは単体で使うと IF 文からの脱出になります。BREAK LOOP、BREAK WEND、BREAK NEXT というように使うと DO～LOOP、WHILE～END、FOR～NEXT からの脱出となります。MPC-2000 は BREAK だけです 。BREAK で、その階層の DO～LOOP、WHILE～END、FOR～NEXT から抜けます。

・DO～LOOP

```
10        SYSCLK=0
20          DO
30            IF    SW(192)==1 THEN
40              BREAK
50            END_IF
60          LOOP
70          PRINT   SYSCLK
```

```
#RUN

 1044
#
```

・WHILE～WEND

```
10       SYSCLK=0
20        WHILE   SW(192)==0
30         IF   SYSCLK>5000 THEN
40          BREAK
50         END_IF
60        WEND
70        PRINT   SYSCLK
#RUN

 5001       ←SYSCLK>5000 で BREAK
#RUN

 1954       ←SW(192)をオン
#
```

・FOR～NEXT

```
10        FOR   I=0 TO 255
20         IF   I==128 THEN
30          BREAK
40         END_IF
50        NEXT
60        PRINT   I
#RUN

 128
#
```

タイムのアウト判定 (time_、TIMEOUT(0))

MPC-684 では 「 WAIT SW(192)==0 TMOUT 5000 : GOTO *PASS 」という記述でしたが、MPC-2000 では 予約ローカル変数 timer_ と TIMEOUT(0)関数 を使います。
MPC-2000 にも TMOUT コマンドもあります。TMOUT コマンドは WS0(),WS1(),HOME に有効です。これらのタイムアウトの精度は約±0.1sec です。

1) timer_ を使って

```
10       SYSCLK=0
20       timer_=50                          /* タイムアウト時間設定 50×0.1=5 Sec
30        WAIT   (SW(192)==1)|(timer_==0)   /* SW(192)がオンするかタイムアウトするまで
40        IF   timer_==0 THEN               /* もしもタイムアウトだったら
50         PRINT   "TIME OUT" SYSCLK
60         END
70        END_IF
80        PRINT   "GOOD JOB" SYSCLK
#RUN

 GOOD JOB 1505                              ← 時間内に SW(192)が押されたとき
RUN

 TIME OUT 4983                              ← タイムアウトしたとき
#
```

2) timer_ と TIMEOUT(0)を使って

```
10       SYSCLK=0
```

```
 20       timer_=50
 30        WAIT   SW(192)==1 OR TIMEOUT(0)
 40        IF   TIMEOUT(0)==1 THEN
 50         PRINT   "TIME OUT" SYSCLK
 60         END
 70        END_IF
 80        PRINT   "GOOD JOB" SYSCLK
#RUN

 GOOD JOB 1910
RUN

 TIME OUT 4966
#
```

3) SW(192)かつSW(193)がオン、またはタイムアウト

```
 10       SYSCLK=0
 20       timer_=50
 30        WAIT   ((SW(192)==1)&(SW(193)==1))|TIMEOUT(0)
 40        IF   timer_==0 THEN
 50         PRINT   "TIME OUT" SYSCLK
 60         END
 70        END_IF
 80        PRINT   "GOOD JOB" SYSCLK
RUN

 GOOD JOB 3358                                ← SW(192)==1 AND SW(193)==1
RUN

 TIME OUT 4901
#
```

ラッチ入力(SENSE_ON、SENSE_OFF、FLIP_FLOP)

MPC-684のSENSE_SW コマンドは

```
 40        SENSE_SW -1 5
 50        SENSE_SW -2 26
 60        FORK 1 *a
 70        FORK 2 *b
```

という具合に複数の設定が可能です。MPC-2000にはSENSE_SWは無く、SENSE_ON/SENSE_OFFがありますが1個しか設定できず、メモリI/Oはサポートしていません。MPC-2000の置き換えは FLIP_FLOP コマンドです。

FLIP_FLOP は1msec毎に入力ポートを読んで、出力・メモリI/OへOR出力します。入力、出力ともバンク(8ビット)単位です。入力が複数ある場合は同一のバンクへまとめて接続してください。動作はバックグランドで行います。

1) 論理反転指定無し。スイッチをオンすると対応するメモリI/Oのビットがセット(1)される。

```
 10        CLR_OUTP 15
 20        FLIP_FLOP   -1 IN(24)
 30        DO
 40         PRX   IN(24) IN(-1)
 50         TIME   500
 60        LOOP
#RUN

00000000 00000000     ← SW(194)=0,SW(193)=0,SW(192)=0
00000001 00000001     ← SW(194)=0,SW(193)=0,SW(192)=1
00000000 00000001
00000002 00000003     ← SW(194)=0,SW(193)=1,SW(192)=0
```

```
00000000 00000003
00000004 00000007      ← SW(194)=1,SW(193)=0,SW(192)=0
00000000 00000007
```

2) 下位3ビット論理反転。スイッチをオフすると対応するメモリI/Oのビットがセット(1)される。

```
10        CLR_OUTP 15
20        FLIP_FLOP   -1 IN(24) &H00000007      /* 第三パラメータにマスクパターン指定
30        DO
40         PRX   IN(24) IN(-1)
50         TIME   500
60        LOOP
RUN

00000007 00000000      ← SW(194)=1,SW(193)=1,SW(192)=1
00000006 00000001      ← SW(194)=1,SW(193)=1,SW(192)=0
00000007 00000001
00000005 00000003      ← SW(194)=1,SW(193)=0,SW(192)=1
00000007 00000003
00000003 00000007      ← SW(194)=0,SW(193)=1,SW(192)=1
00000007 00000007
```

3) 個々のタスクでメモリI/Oがセットされたら仕事→メモリI/Oをオフ=ラッチをリセットする。

```
10        CLR_OUTP 15
20        FLIP_FLOP   -1 IN(24)  /* Memory I/O <- IN 。バックグランドで実行
30        FORK   1 *JOB1
40        FORK   2 *JOB2
50        FORK   3 *JOB3
60        END
70      *JOB1
80        DO
90         WAIT   SW(-1)==1      /* Memory I/O オン待ち
100        PRINT   "1"           /* ┐お仕事
110        TIME   500            /* ┘
120        OFF   -1              /* Memory I/Oオフ = ラッチリセット
130       LOOP
140     *JOB2
150       DO
160        WAIT   SW(-2)==1
170        PRINT   "2"
180        TIME   500
190        OFF   -2
200       LOOP
210     *JOB3
220       DO
230        WAIT   SW(-3)==1
240        PRINT   "3"
250        TIME   500
260        OFF   -3
270       LOOP

#RUN

# 1      ← SW(192)オン
 3       ← SW(194)オン
 2       ← SW(193)オン
 1
 2
```

MPG-2314 原点復帰 (SHOM, HOME)

MPC-684 と MPG-314 では HOME コマンドに軸､速度､停止条件(入力)を設定しましたが､MPC-2000 と MPG-2314 では、速度=ACCEL、停止条件と Z 相検出方向=SHOM、移動=HOME の組み合わせで行います。

1) MPG-2314 原点復帰　X,Y 同時、CCW 方向にニアオリジンだけを検出する例

```
10        PG   0
20        ACCEL   ALL_A 1000 500 100
30        SHOM   X_A|Y_A IN0_ON      /* X,Y 軸 ニアオリジン検出設定
40        HOME   X_A|Y_A NEG_L       /* X,Y 軸 CCW 方向にニアオリジンを検出するまで移動
                                     /*  この時の停止は減速停止=ACCEL 設定に依存
50        WAIT   RR(ALL_A)==0
60        STPS   X_A|Y_A 0           /* X,Y 軸ここを'0'にセット
```

2) MPG-2314 原点復帰　X 軸、CCW 方向にニアオリジン検出後、CW 方向に Z 相を検出する例

```
10        PG   0
20        ACCEL   ALL_A 1000 500 100
30        SHOM   X_A IN0_ON|IN1_ON|CW   /* X 軸 ニアオリジン、オリジン検出設定
                                        /*  オリジン検出は CW 方向
40        HOME   X_A NEG_L              /* X 軸 CCW 方向にニアオリジン検出まで移動→減速停止
                                        /*  その後 CW に Z 相検出移動。この時の停止は即停止
50        WAIT   RR(ALL_A)==0
60        STPS   X_A 0                  /* X 軸ここを'0'にセット
```

この場合は次の A)B)の移動を連続して実行します。
A) CCW 方向へニアオリジン(XIN0)がオンするまで移動→XIN0 がオンすると減速停止
減速域は ACCEL の leng(上記の場合 500 パルス)

```
  XIN0(ニアオリジン)オン
   |← CCW  最高スピード=1000PPS、 加減速有り、加減速域=500
```

B) CW 方向へオリジン(XIN1=Z 相)がオンするまで移動→XIN1 がオンすると即停止
CW 方向の移動速度は ACCEL の lo_pps(上記なら 100PPS)となります。

```
                                  XIN1(オリジン)オン
    スピード=100PPS、加減速無し CW →|
```

注意
ACCEL コマンドの自起動(lo_pps)=0 は不可(HOME コマンドがパルス発生しない)。
SHOM X_A IN1_ON|CCW というように IN1 のみの設定は不可(HOME で停止しない)。

3) HOME コマンドに退避移動は含まれません。事前に原点センサを確認し必要に応じて退避させます。

```
IF HPT(XIN0)<>0 THEN          /* X 軸 IN0 がオンなら退避移動
  RMVS X_A 10000
END_IF
```

4) ニアオリジン(IN0)停止の場合、ACCEL コマンドの加減速設定が有効で、センサーがオンしてから減速領域分移動します。次は加減速無しの実行例です。

```
10        PG   0
20        ACCEL   X_A 1000 500 1000   /* ドライブ速度=初速度  ( leng == 0 は不可)
30        SHOM   X_A IN0_ON
40        HOME   X_A NEG_L
50        WAIT   RR(X_A)==0
60        STPS   X_A 0
```

5) オリジン(IN1)だけで原点復帰する例
ニアオリジン(IN0)は未接続、オリジン(IN1)がオンするまで CCW 方向に原点復帰。
この場合の移動速度は ACCEL の初速度(例では 5000pps)で即停止します。

```
10        PG   0
20        ACCEL   X_A 10000 500 5000      /* 初速度 5000pps
30        SHOM   X_A IN0_OFF|IN1_ON|CCW   /* IN1 がオンするまで CCW 移動
```

```
40        HOME   X_A NEG_L
50        WAIT   RR(X_A)==0
60        STPS   X_A 0
```

6) X,Y 同時、ニアオリジン(IN0)検出後オリジン(IN1:Z 相)を検出する例
X 軸は CW 方向に原点検出、Y 軸は CCW 方向に原点検出

```
PG 0
/* 退避移動とニアオリジン(IN0)検出まで最高速 4000pps、
/* ニアオリジンを検出してからオリジン検出まで(この場合は IN0 が ON→IN1 が ON)
/* は最低速度(この場合は 500pps)で動作します。
ACCEL X_A|Y_A 4000 1000 500       /* 最高速 4000pps、加減速域 1000 パルス、最低速度 500pps
FEED X_A|Y_A 100                  /* スピード 100%
RMVL -5000 5000                   /* 退避移動
WAIT RR(X_A|Y_A)==0               /* 退避移動終了待ち
SHOM X_A IN0_ON|IN1_ON|CW         /* X 軸 IN0 が ON して IN1 が ON するまで CW 方向
SHOM Y_A IN0_ON|IN1_ON|CCW        /* Y 軸 IN0 が ON して IN1 が ON するまで CCW 方向
HOME POS_L NEG_L 0 0              /* 原点復帰 X 軸 CW、Y 軸 CCW 方向に動作します
WAIT RR(X_A|Y_A)==0               /* HOME 終了後座標は X=0、Y=0 になります。
ACCEL X_A|Y_A 10000               /* スピード再設定
FEED X_A|Y_A 100                  /* スピード最高速の 100%
```

MPG-2541 原点復帰 (SHOM, HOME)

MPG-2541 の HOME コマンドは基本的に、XSD(ニアオリジン)と X_ORG(オリジン)を使う仕様になっています。原点センサが 1 つの場合は X_ORG に接続します。
下記のサンプルは X_ORG、XSD どちらもノーマリオープン(メカが叩いてオン)の LS を用いています。

1) MPG-2541 原点復帰　X 軸、CCW 方向に X_ORG だけを検出する例

```
10        PG 10
20        ACCEL    X_A 5000          /* 原点復帰の速度
30        FEED  X_A 100              /* 必ず FEED を設定
40        SHOM   0                   /* SHOM の設定
50        IF   HPT(2)==1 THEN        /* X_ORG が入っていたら
60         RMVS   X_A 5000           /* 退避移動
70         WAIT   RR(X_A)==0
80        END_IF
90        HOME   X_A -30000          /* 原点復帰 XORG オン:停止
100       IF   X(0)<>0 THEN          /* HOME が完了していれば 0 になる
110        PRINT   "HOME 未完了"
120       END_IF
```

2) MPG-2541 原点復帰　X 軸、CCW 方向に XSD→X_ORG を検出する例

```
10        PG   10
20        ACCEL    X_A 5000          /* 原点復帰の速度
30        FEED  X_A 100              /* 必ず FEED を設定
40        SHOM   0                   /* SHOM の設定
50        IF   HPT(1)==1 THEN        /* XSD が入っていたら
60         RMVS   X_A 5000           /* 退避移動
70         WAIT   RR(X_A)==0
80        END_IF
90        HOME   X_A -30000          /*原点復帰 XSD オン:減速→XORG オン:停止
100       IF   X(0)<>0 THEN          /* HOME が完了していれば 0 になる
110        PRINT   "HOME 未完了"
120       END_IF
```

※動作速度について
XSD 検出までは ACCEL で設定した最高速、検出後は同じく最低速の設定値(PPS)となります。ACCEL の最低速は省略すると最高速の 1/20 の速度になります。

動作スピードの設定 (FEED)

MPC-684 の FEED コマンドには幾つかの書式があります。MPG-314 の基本仕様は 0(最高速)～255 段階です 。MPC-2000 の FEED コマンドは 100(最高速)～0 です。pps を指定する場合 SPEED コマンドを使います。

タッチパネルの宣言 (MEWNET)

MPC-684 RS-232C でタッチパネルを使うときは PROTOCOL MEWNET と宣言しました。通信ポートも CH0 に限定されていました。MPC-2000 は MEWNET コマンドで通信ポート番号と通信速度(タッチパネル機種によってはモード)を宣言します。MEWNET は 32 - CH 番号 のタスクを占有します。これらのタスクはプログラムでは FORK できません。

```
MEWNET 38400 2              /* CH2 接続 占有タスクは 30
MEWNET 38400 5              /* MRS-MCOM CH5 接続 占有タスクは 27
MEWNET 9600 1 B70           /* CH1 接続 占有タスクは 31 Bit7 奇数パリティ(三菱小型タッチパネル)
```

廃止コマンド・関数

FAST

FAST コマンドは MPC-684 のタスク遷移効率を上げる(全体の実行効率を上げる)目的のコマンドですが 、MPC-2000 シリーズは格段に CPU の実行速度が速く、インタプリタも見直し、FAST は不要になりました。

?()

?() は HSW() の省略形です。MPC-684 は 1 行の記述可能な文字数が少なく、論理結合文を書くために省略形を用意しましたが、MPC-2000 シリーズは
・1 行は 256 文字 MAX(crlf 除けば 254 以下)
・コマンド・関数の引数の文字数は 100 文字 MAX(全て半角英数字)
となっており、省略形の必要がありません。

MBK(-1),MBK(-3),MBK(-4)

MBK(-1～-4)は 684 シリーズのタッチパネル通信ボード MBK-SH 対応の関数です。MPC-2000 にはありません。

LD_M,SV_M

配列～MBK エリア間のメモリコピー LD_M,SV_M に替わり、MPC-2000 は DIMCPY になりました。

PROTOCOL

タッチパネルプロトコル宣言は PROTOCOL に替わり、MPC-2000 は MEWNET コマンドになりました。

CALF

MPC-684 の不動少数点演算 CALF コマンドに替わり、MPC-2000 は FLOAT コマンドになりました。

ADR()

MPC-684 のアドレス取得関数 ADR()はありません。

SETIO

SETIO コマンドを実行してもエラーにはなりませんし、実出力はオフされますが、RS-232C Ch1 の動作が不安定になります。CLR_OUTP を使ってください。

INP$#0, INP$#1, PUT, PUT#0, PUT#2

INPUT#, PRINT#におきかえ。