

第3章 プログラミング

3-1 プログラム方法とツール

プログラミングは、Windows パソコンと接続ケーブルで行います。
開発用のソフトウェア(ツール)はインストーラ「Accel_Setup_jpn.msi」でセットアップします。

ハードウェア

- ・パソコン Windows XP 以降
- ・接続 USB-RS(ACCEL)、ケーブル DOS/V(ACCEL)、市販 USB シリアル変換器等

ソフトウェア

インストーラはネット上に公開しております。(無償でダウンロードできます)
標準インストールフォルダは x86 PC= "C:\Program Files\Accel", x64 PC="C:\Program Files (x86)\Accel"
で、サブフォルダーに取り説、デバイスドライバ等が配置されます。

Program Files (x86)		
Accel	← Accel フォルダ	開発ツールと関連ファイル
Documents	← Documents フォルダ	開発ツールの取り説等
usbcunet_x64	← usbcunet_x64 フォルダ	Windows 7,8 x64 用 usbcunet デバイスドライバ
usbcunet_x86_wlh	← usbcunet_x86_wlh フォルダ	Windows Vista,7,8 x86 用 usbcunet デバイスドライバ
usbcunet_x86_wxp	← usbcunet_x86_wxp フォルダ	Windows 2000,XP x86 用 usbcunet デバイスドライバ
USB-RS	← USB-RS フォルダ	MPC-2200,USB-RS,MPC-IP 用 USB シリアルデバイスドライバ

ACCEL フォルダに配置される主要ファイルです。



[FTMW2K.EXE] ターミナルソフト
MPC と接続して編集、デバッグ、ファイルの読込、保存
(資料等で FTM または FTMW と表記されている場合があります)



[MPCED2K.EXE] オフラインエディタ
MPC 専用のテキストエディタ。制御文、ラベル、コメント色分け



[MPC_Monitor.EXE] MPC モニタ
I/O・実行ステップ・変数等のモニタ、ファイル読込・保存



[SYSULD2K.EXE] システムローダー
MPC のファームウェアのアップデート



[F2KCHECK.EXE] プログラム簡易チェッカー
DO~LOOP などの制御構文チェック、ラベル重複チェック等



[ACTERM.EXE] 汎用ターミナルソフト
デバッグ、接続機器の動作確認



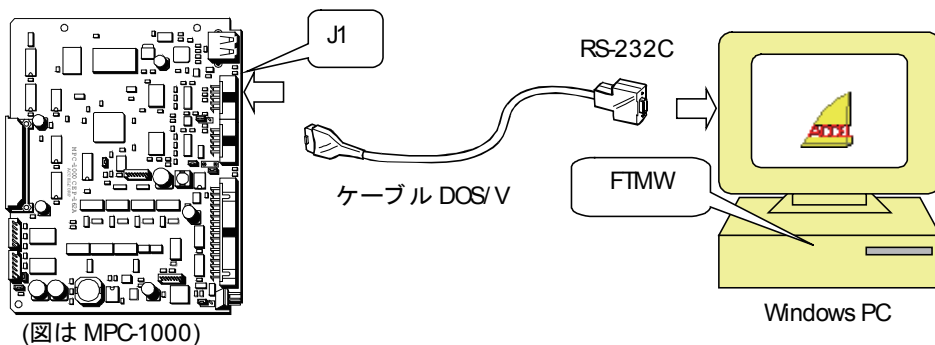
[LINEMON.EXE] 汎用 RS-232C ラインモニタ
デバッグ、接続機器の動作、通信内容モニタ



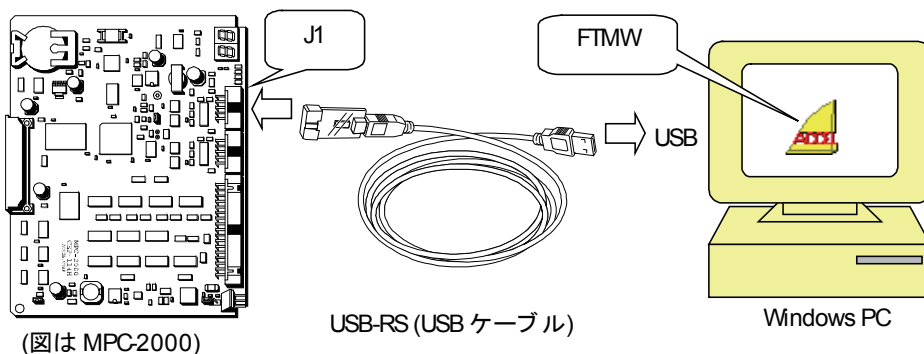
[CUMON.EXE] CUnet モニタ
USB-CUnet 動作確認、モニタ

3-2 MPC とパソコンの接続

- パソコンの標準 COM ポートを使う場合 (ケーブル DOS/V での接続)

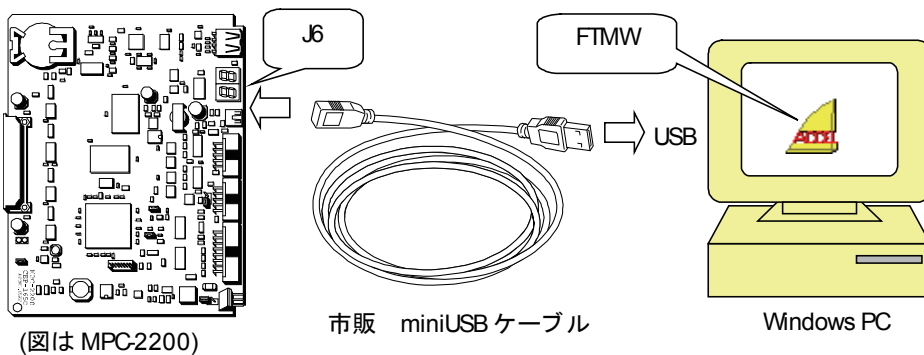


- パソコンの USB ポートを使う場合 (USB-RS での接続例)



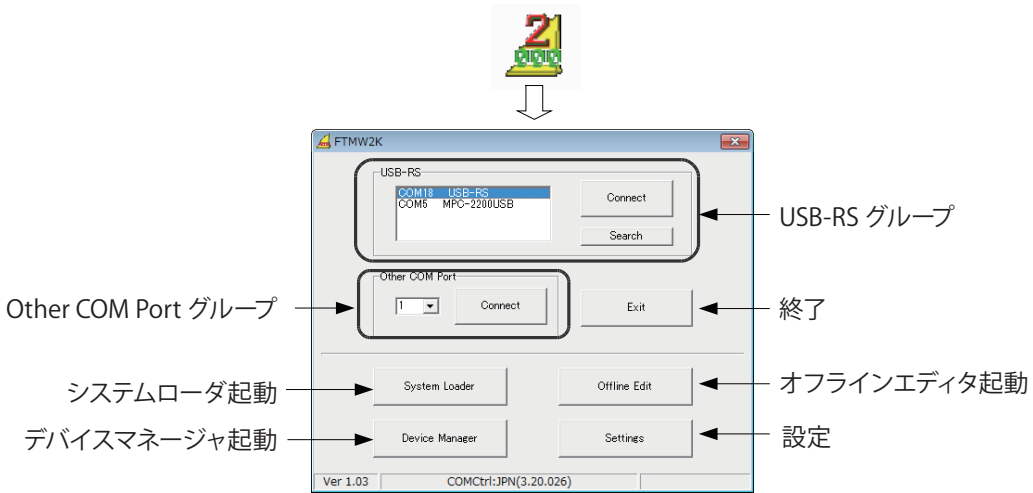
※ USB を使用するにはデバイスドライバのインストールが必要です。

- MPC-2200 の場合



3-3 FTMW 起動

① FTMW のショートカットアイコンをクリックしてください。スタート画面が表示されます。



■ USB-RS グループ

FTMW は起動時に USB-RS、MPC-2200(USB) を検出します。これらと接続する場合は接続先を選択してこちらの Connect を押します。

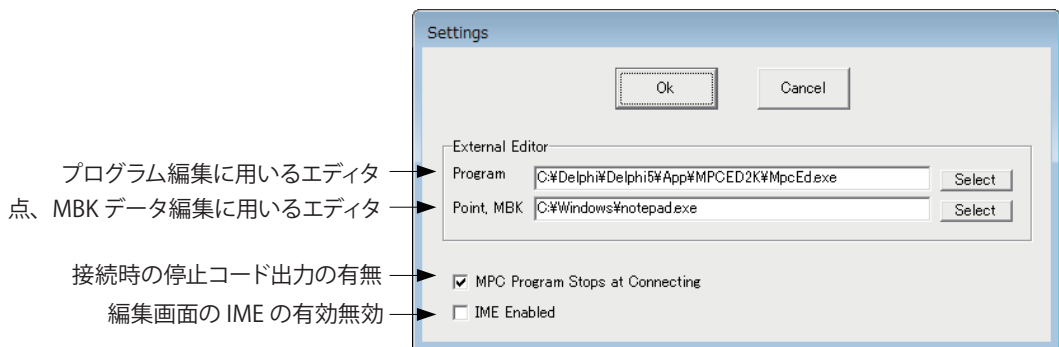
■ Other COM Port グループ

PC 内蔵の COM ポート、市販の USB シリアルコンバータで接続する場合は、COM 番号を設定してこちらの Connect を押します。

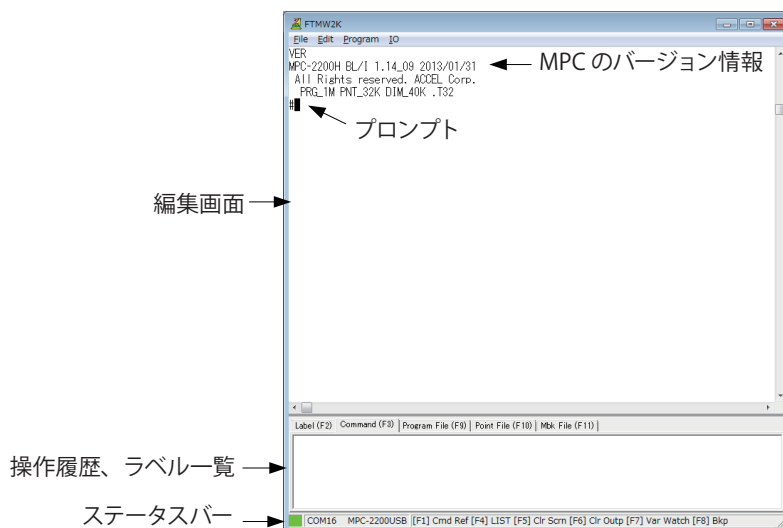
COM 番号はデバイスマネージャで調べてください。

■ 設定画面

プログラム編集用のテキストエディタ、MPC 接続時の停止コードの有無などを設定します。エディタ名はフルパスで指定します。



② MPC の電源を入れ Connect ボタンを押すと編集画面が開きます。



■ 編集画面

プログラムの編集、実行、デバッグ等、基本的な作業はこの画面で行います。

■ 操作履歴、ラベルー覧

実行したコマンドの履歴、読み・保存したファイルの履歴、プログラムのラベルー覧です。ラベルはソースファイルから読み込みます。

■ ステータスバー

入力インジケータ、接続ポート、操作キーガイドを表示します。

3-4 コマンド入力

プロンプトの後にコマンドを入力して Enter すると、その場で実行します。これをダイレクトコマンド実行と言います。殆どのコマンドはダイレクトに実行する事もプログラムに記述する事もできますが、メンテナンス・編集関係などでダイレクトコマンドでしか使えないものや制御文などでプログラムにしか書けない物もあります。

両方で使える	ダイレクトのみ	プログラムのみ
ON 0 OFF 0 PRINT A MOV L など	LIST MPCINIT ERASE RUN など	GOTO GOSUB IF ~ FOR ~ NEXT など

#ON 0<Enter> /* ダイレクト実行 10 ON 0<Enter> とすればプログラム
 #GOTO 100<Enter> /* ダイレクト実行しても何もおこらない
 #10 MPCINIT<Enter> /* このコマンドをプログラムするとプログラムが消える！

※本文中の <Enter> は、パソコンのキーボードの Enter キー押下を表します。

■ トレーニングキット (XY03) での実行例

```
#ON 0          /* フロントパネル緑 LED 点灯
#ON 1          /* 黄 LED 点灯
#PRINT SW(192) /* 緑 SW の状態確認
0             /* 0= オフ
#PR SW(192)   /* 緑 SW を押しながら実行 (PR は PRINT の短縮形)
1            /* 1= オン
#PR IN(24)    /* DSW を '3' にしてパラレル入力
48
#PRX IN(24)   /* PRX は HEX 表示 10(Dec)=30(Hex)
00000030
#SETIO       /* 出力一括 OFF
#
```

3-5 FTMW でのプログラム編集

FTMW で現在 MPC に入っているプログラムを編集する際、頻繁に行う操作を解説します。

LIST の表示

最も頻繁に使用するのが LIST コマンドです。

```
*書式
LIST [arg1 arg2]
arg1: 開始文番号或は開始ラベル
arg2: 表示行数
```

- LIST だけでも実行できます (下記①)。その場合は前回の続が表示されます。
- 第 1 パラメータに表示開始位置を文番号またはラベルで指定できます (下記②③)。
- 第 2 パラメータに表示する行数を指定できます (下記④)。以後この行数は保持されます。
- LIST 0 とすると最初から表示します (下記⑤)。

①

```
#LIST /* 開始位置、行数指定無し
10 GOSUB *READ_DSW
20 _RET_VAL D
30 PRINT D
40 END
50 *READ_DSW
#
```

②

```
#LIST 40 /* 開始位置 = 文番号指定
40 END
50 *READ_DSW
60 DSW_=IN(24)/16
70 RETURN DSW_
#
```

③

```
#LIST *READ_DSW /* 開始位置 = ラベル指定
50 *READ_DSW
60 DSW_=IN(24)/16
70 RETURN DSW_
#
```

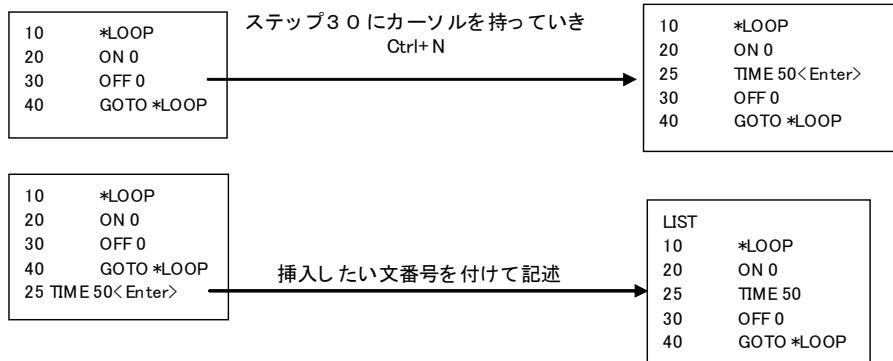
④

```
#LIST *READ_DSW 2 /* 開始位置 = ラベル指定、表示行数指
定
50 *READ_DSW
60 DSW_=IN(24)/16
#
```

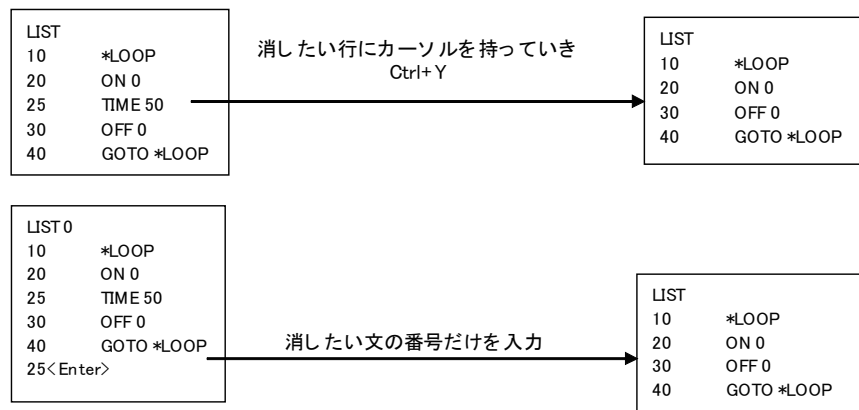
⑤

```
#LIST 0 5 /* 開始位置 = 先頭、表示行数指定
10 GOSUB *READ_DSW
20 _RET_VAL D
30 PRINT D
40 END
50 *READ_DSW
#
```

行の挿入



行の削除



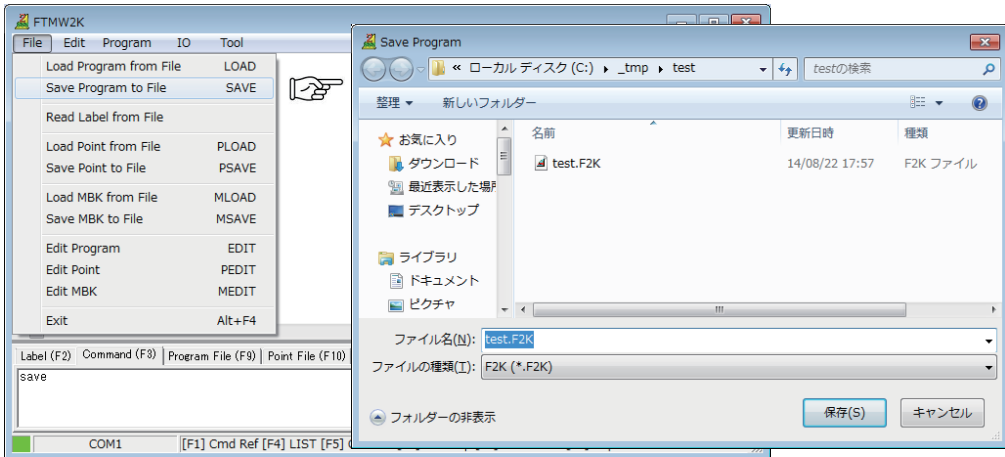
編集のための主なキー操作

F1	Web のコマンドリファレンス参照
F4	LIST コマンドを実行
F5	画面消去
Ctrl+C	選択範囲をクリップボードにコピー
Ctrl+V	クリップボードから貼り付け
Ctrl+N	1 行挿入
Ctrl+B	空行を挿入
Ctrl+Y	一行削除 (MPC のメモリからも消去)
Ctrl+U	キャレット位置から行頭を消去
Ctrl+K	キャレット位置から行末を消去
矢印キー	キャレット移動
Ctrl+ 矢印キー	画面スクロール
Shift+ 矢印キー	範囲選択
Insert	キャレット位置にスペース挿入
Delete	キャレット位置の文字を 1 文字消去
End	キャレットを行末に移動
Home	キャレットを行頭に移動

3-6 プログラムの保存・読み込み

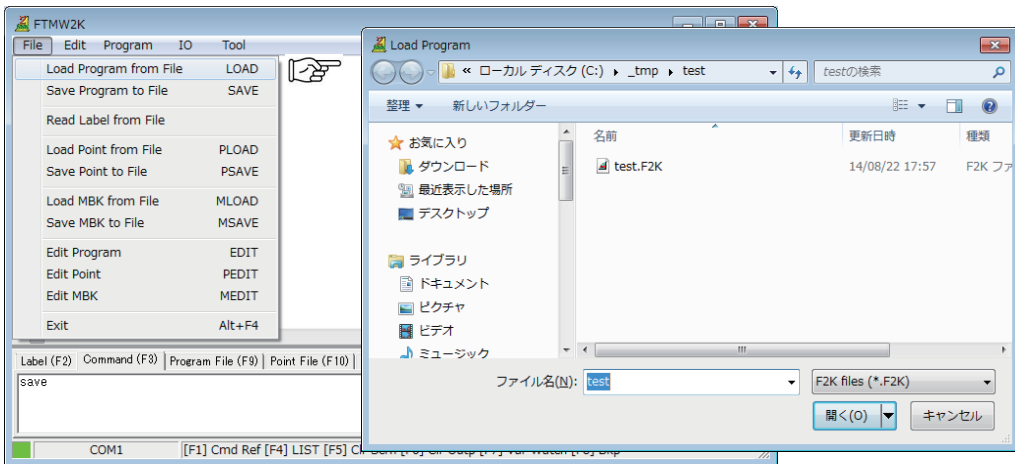
保存

メニュー File > Save Program to File でパソコンに保存します。拡張子 F2K で保存されます。保存したプログラムには文番号はありません。



読み込み

メニュー File > Load Program from File でパソコンからプログラムを読み込みます。文番号は初期状態で 10 間隔です。60000 ステップを越えると自動的に 5 で RENUM します。



オフライン作成

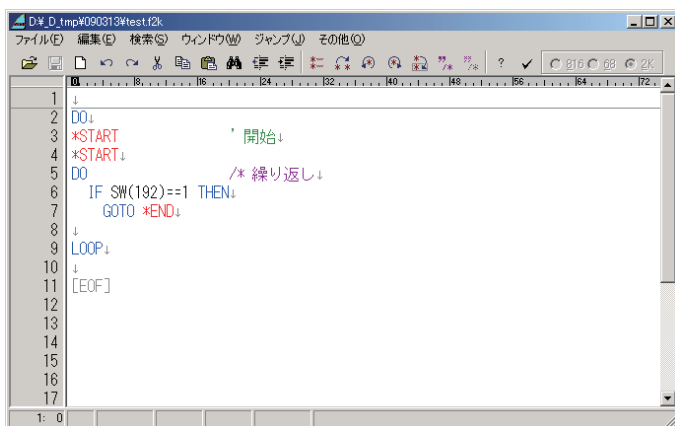
- FTMW は MPC 内のデータを直接操作するもので、接続していないと使うことはできませんが、プログラムはオフラインで作成することができます。
- エディタをご用意ください。MPCED は MPC 専用ですが、汎用のエディタやワープロでも使えます。その場合、プログラムはテキストデータ、拡張子を "F2K" として保存して下さい。
- プログラムのエラーは MPC に読み込み、実行するまではわかりません。

印刷

FTMW には印刷機能はありません。パソコンに保存したファイルを MPCED などのエディタやワープロソフトで印刷してください。

3-7 オフラインでプログラム作成・編集

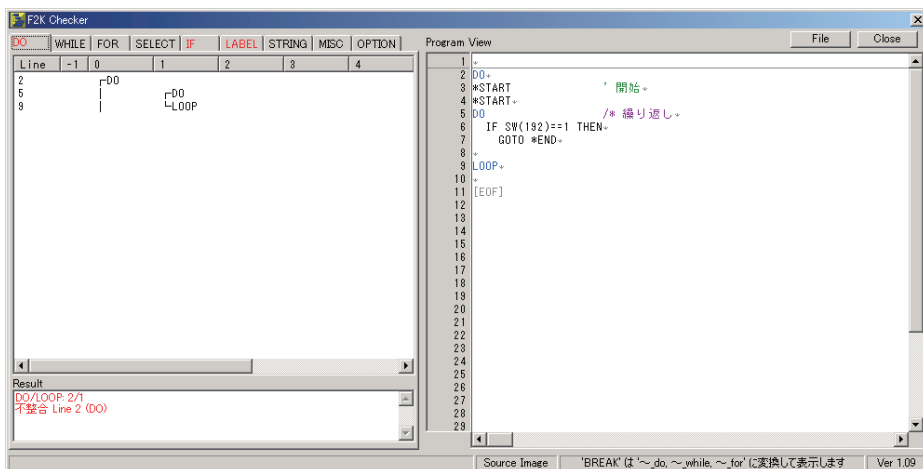
- FTMW は MPC と接続して使用するもので、パソコンに保存したプログラムの編集や、作成はできません。オフラインでのプログラミングはエディタを使用します。エディタは、一般的にテキストエディタと呼ばれるもので、市販品、フリーウェアや Windows のメモ帳でも作成できます。
- MPC 専用のエディタとして「MPCED」があります。制御文などの色分け、ラベルジャンプ、一括コメント/アンコメントなどの機能があります。
- インターネットに接続できる環境なら、(HELP を参照する要領で) コマンド上にカーソルを置き F1 キー押下で MPC-2000 サイトのコマンドリファレンスを直接参照できます。



```
1 ↓
2 DO↓
3 *START ' 開始↓
4 *START↓
5 DO /* 繰り返し↓
6 IF SW(192)==1 THEN↓
7 GOTO *END↓
8 ↓
9 LOOP↓
10 ↓
11 [EOF]
12
13
14
15
16
17
1: 0
```

- プログラムが大きくなると DO ~ LOOP や IF ~ END_IF などの対応が判り難くなりがちです。そんなときは簡易チェッカー「F2KCheck」を使ってみてください。MPCED のチェックボタンで起動します。

例えば上のプログラムは DO ~ LOOP、IF ~ END_IF が正しく対応していません。またラベルの重複があり、GOTO の行き先がありません。これをチェッカーでみると次のようになります。赤いタブが警告を表します。



Line	WHILE	FOR	SELECT	IF	LABEL	STRING	MISC	OPTION
2	-1	0	1	2	3	4		
5		-DO						
9			-DO					
			-LOOP					

```
1 ↓
2 DO↓
3 *START ' 開始↓
4 *START↓
5 DO /* 繰り返し↓
6 IF SW(192)==1 THEN↓
7 GOTO *END↓
8 ↓
9 LOOP↓
10 ↓
11 [EOF]
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

※これらは単純な数合わせです。プログラムの正常動作を保証するものではありません。

3-8 初期化

MPCの初期設定や実行時パラメータはフラッシュ ROM と S-RAM に記憶されます。開発中の試行錯誤や搬送中(特に基板単体で)の静電気印加などでパラメータが狂うと動作不良になります。次の時は初期化を励行して下さい。

- * 基板単体で搬送したとき。搬送中にダメージを受けることがあります。メンテナンス等で基板単体にプログラムを入れて搬送する場合は、静電気、バッテリーのショート・脱落、部品破損、結露などご注意ください。必ず帯電防止袋を使用して下さい。
- * デバッグ中、挙動不審になったとき。あれこれやっているうちにおかしくなったとき。プログラムは良いはずなのにうまく動かない。(バグの可能性も追求してください) etc
- * システムのアップデート後。

■ 初期化コマンド

MPCINIT SRAM の初期化 (英語モードになります)

ERASE フラッシュ ROM の消去

※ダイレクトコマンドでこの2つを実行して下さい。

JPN 日本語モードになります。

```
実行例: #MPCINIT
#ERASE
*
#JPN
#
```

■ 初期化時の注意

初期化するとプログラム、点データ、変数はクリアされます。必要に応じてパソコンに保存、記録を取って下さい。

3-9 I/O チェック

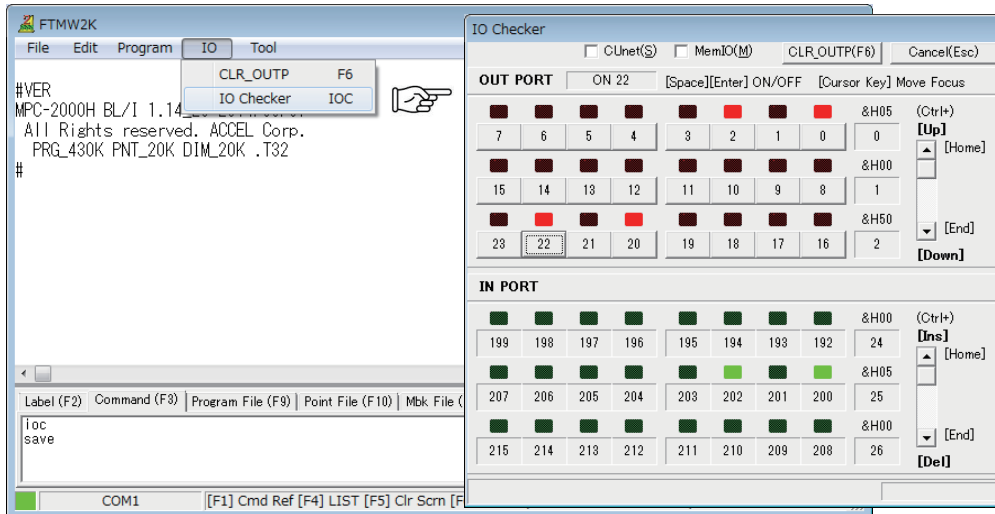
コマンドによるチェック

ダイレクトコマンドによる I/O チェックの例です。

```
#ON 0               /* フロントパネル緑 LED 点灯
#ON 1               /* 黄 LED 点灯
#PRINT SW(192)      /* 緑 SW の状態確認
0                   /* 0= オフ
#PR SW(192)         /* 緑 SW を押しながら実行 (PR は PRINT の省略形)
1                   /* 1= オン
#PR SW(195)         /* セレクタ SW 左側
0                   /*
#PR SW(195)         /* セレクタ SW 右側
1                   /*
#PR IN(24)          /* DSW を '3' にしてパラレル入力
56                  /*
#PRX IN(24)         /* PRX は HEX 表示 56(Dec)=38(Hex)
00000038            /*
#CLR_OUTP 15        /* 出力一括オフ
```

I/O チェッカによるチェック

まとめて見るなら [F8] I/O Checker で I/O チェッカを起動。
IOC<Enter> としても I/O チェッカが起動します。



3-10 言語の仕様

整数 BASIC

MPC-2000 の言語はマルチタスク整数 BASIC です。通常の BASIC にはマルチタスクはありませんが、MPC-2000 の BL/1(Basic Language 1) では、32 個までのプログラムを時分割処理により同時実行することができます。これにより、複雑な装置の動作に対応可能となっています。

変数は 4byte 整数を標準としています。

変数は、通常 BASIC と同様、定義することなく使うことができます。変数ラベルは 15 文字以内です。計測などに必要な浮動小数点演算は、FLOAT コマンドで実現しています。通常、整数のみである理由は、処理速度を鈍化させないことと、処理のあいまいさを避けるためです。

浮動小数点演算は、広い範囲の数値を扱うことができますが、プログラムによっては意図しない誤差が拡大し思わぬ誤作動を発生させることがあります。

マルチステートメント

BL/1 は 1 行ごとのコマンドや式から成立します。実行単位は 1 行ですが、以下のようにコマンドを ' (コロン) で区切ると 1 行の中に複数のコマンドを記述することができます。

ON 1:TIME 100:OFF 1:TIME 100

IF 文の記述も ' で区切れれば一行となります。

```
IF i%10==1 THEN
  USB_DEL FILE$
END_IF → IF i%10==1 THEN : USB_DEL FILE$ : END_IF
```

コメント

(シングルクォート)から始まる文字列はコメントとなり、実行されません。しかしインタプリタでは、コメントといえどもわずかに時間を消費します。

(シングルクォート)コメントは、実行されない部分になるべく使用してください。(プログラムの冒頭など)

マルチタスク

実際の装置制御では様々なアクチュエータが同時進行で制御されなければなりません。しかし一般的なソフトウェア言語ではマルチタスク対応に煩雑な手続きを必要としたり、動作しても処理速度が遅いのが一般的です。これは、一般にマルチタスクが情報処理の分野で開発されたこと、大規模システムで運用されることを前提としているためです。

これに対し、BL/1のマルチタスクは、装置制御に特化し、小規模なシステムで動作することを前提としています。このため、非常に簡単な手続きで使用することができるよう設計されています。

1) FORK

通常の BASIC 系インタプリタでは、プログラムは順次コマンドリストを実行するようにしています。その実行順序はただ1つです。例えば以下の例では、10～30の間のコマンドを延々と繰り返しています。

```
10 DO
20 ON 1:TIME 100:OFF 1:TIME 100
30 LOOP
```

しかし、MPC-2000では、FORK コマンドにより複数の実行経路を起動することができます。

FORK とは、RUN に相当するコマンドで、RUN *TASK をタスク1として実行するという意味です。

タスクは、1～31まで指定することができ、同時に31個のプログラムを実行することができます。

なお、最初に RUN によって実行されるプログラムはタスク0となります。

```
10 FORK 1 *TASK
20 DO
30 ON 1:TIME 100:OFF 1:TIME 100
40 LOOP
50 *TASK
60 DO
70 ON 2:TIME 100:OFF 2:TIME 100
80 LOOP
```

2) タスク管理

実行が開始されたタスクを途中で停止させたり、再開させたりすることができます。

QUIT n タスクを停止させるコマンドです。

PASUE n タスクを一時停止させるコマンドです。

CONT n 一時停止させたタスクを再開するコマンドです。

また、他のタスク状況を知りたく、自分のタスク番号を知りたいという場合は以下の方法があります。

TASK() 他のタスクが実行中であるか停止しているかを調べる。

TASKn 予約変数で常に自己のタスク番号を得ることができる。

3) セマフォ

実際のマルチタスクで最も難しいのが、1つのアクチュエータあるいは出力を複数のタスクから使用することです。例えば以下の例では、RS-232C CH1 に対して2つのタスクが文字列を出力しています。

```
10 FORK 1 *TASK1
20 FORK 2 *TASK2
30 END
40 *TASK1
50 DO
```

```

60   FOR i=&h0041 TO &h004A
70   PRINT# CHR$(i)
80   NEXT
90   PRINT# "¥r¥n"
100  TIME 500
110  LOOP
120  *TASK2
130  DO
140  FOR j=&h0030 TO &h0039
150  PRINT# CHR$(j)
160  NEXT
170  PRINT# "¥r¥n"
180  TIME 500
190  LOOP

```

結果は以下のようになり、2つのタスクからの出力が入り混じった状態となります。

```

【RS-232C 出力】
ABCDEF0123456IJ
789
ABCDEF0123456IJ
789

```

そこで、セマフォと呼ばれるタスク間のインターロックを追加します。以下プログラムで、WAIT ON(-1)と OFF -1 の部分です。

```

10   FORK 1 *TASK1
20   FORK 2 *TASK2
30   END
40   *TASK1
50   DO
55   WAIT ON(-1)
60   FOR i=&h0041 TO &h004A
70   PRINT# CHR$(i)
80   NEXT
90   PRINT# "¥r¥n"
95   OFF -1
100  TIME 500
110  LOOP
120  *TASK2
130  DO
135  WAIT ON(-1)
140  FOR j=&h0030 TO &h0039
150  PRINT# CHR$(j)
160  NEXT
170  PRINT# "¥r¥n"
175  OFF -1
180  TIME 500
190  LOOP
#

```

結果は以下のように整理された出力となります。

```

【RS-232C 出力】
0123456789
ABCDEF0123456IJ
0123456789
ABCDEF0123456IJ
0123456789

```

セマフォは、鉄道で使われた衝突防止の腕木式信号機のことです。タスクを複数の線路とすれば、セマフォ(腕木式信号機)によって、線路(タスク)の交差するところで列車の衝突がおこらないようにするものです。セマフォには、WAIT ON(-1)のように通常はメモリ I/O を使いますが、そのほか ON() 関数の対応する I/O エリアであれば、どの出力ポートでも構いません。

4) SWAP コマンド

プログラムを実行して CTRL_A で停止させると、以下のような表示が出力されることがあります。

```
*O! [20]
!は時間浪費タスクです。
```

タスク番号の後ろに!が時間を浪費していることを示しています。

マルチタスクは、人間の目からみれば、複数のプログラムが同時に走っているように見えますが、CPUとしては、時間を分割して順次タスクを実行しているのにすぎません。BL/1 ではラウンドロビン方式という単純な時分割マルチタスクとなっています。ひとつのタスクは3msecごとに切り替わっていきます。しかし、TIME, WAIT SW()などの条件待ちコマンドがあると、タスクを強制的に切り替えます。条件にあわない時は、これ以上、そのタスクを実行しても時間の無駄になるからです。

おなじ条件待ちでも、以下のようなプログラムでは時間の浪費が発生します。

従って、aが0の場合は、タスクの強制切り替えを発生させるべきです。

```
10 DO
20 IF a==1 THEN :BREAK :END_IF
30 LOOP
#run
```

```
*O! [20]
!は時間浪費タスクです。
#
```

これには、以下のように SWAP コマンドを追加します。

```
10 DO
20 IF a==1 THEN :BREAK :END_IF
25 SWAP
30 LOOP
#run
```

```
*O [25]
#
```

SWAP コマンドはタスクの強制切り替えを発生させるコマンドです。

装置制御は、条件がいくつか揃ったら、なんらかのアクションをおこすという処理の集まりです。従って、条件が揃わなければ、何もしないわけですので、この場合は SWAP を加筆して時間の浪費を抑制します。或いは、条件が揃うまでに、もっと時間が見込まれる場合や、もともと高速に反応する必要がなければ、SWAP の代わりに TIME 100 などのタイマーコマンドを使います。

TIME もタスク強制切り替えをして、その上に指定時間だけタスクをスリープさせます。そのタスクが使わない時間は、他のタスクで有効に使います。

デバック

1) BREAK_POINT

BL/1 では BREAK_POINT コマンドにより、8 個までの指定した文番号でプログラムを停止させることができます。(ラベル指定も可能です)

以下のようにプログラム番号を指定すると、指定行を表示します。

その後指定行の文番号は、反転表示されます。

ブレークポイントは、順々に文番号を指定します。指定した文番号を解除する場合は、同じ番号を入力します。どの文番号が登録されているかは、BKP コマンドを引数なしで実行します。

また、すべてのブレークポイントを解除するには、BKP 0 と入力します。

```
30 FORK 2 *bb
40 END
110 *bb
120 DO
```

```

130   FOR i_=8 TO 15
140   ON i_:TIME 50:OFF i_150   NEXT
160   LOOP
#bkp 110 140

```

```

110  *bb
140  ON i_:TIME 50:OFF i_
#bkp
BREAK_POINT 0=110
BREAK_POINT 1=140
#bkp 110

```

```

110  *bb
#bkp
BREAK_POINT 0=140
#

```

- ①実際にブレークポイントを指定して RUN させると、指定位置で実行が中断されます。そして、中断した行と、タスク番号が表示されます。n<ENTER> キーによって、次のブレークポイントまで実行再開します。このプログラムでは、文番号 30 を通るたびにブレークします。(実行前で)
- ②ステップ送り(1行ずつ継続的に実行)させる場合は t<ENTER> を押します。ステップ送りの解除は、<ENTER> キーを押します。
- ③ブレーク停止中に変数や関数の値を参照することができます。
'p' を押して、続けて変数名や関数名を入力します。
- ④ブレークポイントを追加することもできます。
'b' を押して文番号を入力すると、ブレークポイントを追加することができます。
- ⑤ブレーク中にそのブレークポイントを解除したい場合は、"u" を入力します。
- ⑥プログラム実行を停止する場合は 'e' を押します。

```

#list *aa
50  *aa
60  DO
70  FOR i_=0 TO 7
80  ON i_:TIME 200:OFF i_
90  NEXT
100 LOOP
#bkp 100

100 LOOP
#run *aa
50##
① 100 LOOP <00>
② #t
60 DO <00>
#t
70 FOR i_=0 TO 7 <00>
#t
80 ON i_:TIME 200:OFF i_ <00>
③ ?pi_
#PR i_-> 0
#
100 LOOP <00>
?b80
④ #BKP 80->

```

```

80    ON i_:TIME 200:OFF i_
#
80    ON i_:TIME 200:OFF i_ <00>
?p i_
#PR i_> 0
#
80    ON i_:TIME 200:OFF i_ <00>
#
80    ON i_:TIME 200:OFF i_ <00>
?p i_
#PR i_> 2
#
80    ON i_:TIME 200:OFF i_ <00>
⑤ ?u
#
100   LOOP    <00>
#
100   LOOP    <00>
⑥ ?e
##

```

2) FOR 文をブレークポイントにすると・・・

以下のようなプログラムで、20 をブレークポイントにした場合の注意です。

FOR 文の実行順序は、20->30->40->30->40->30->40->40 となり、FOR ループの中では、FOR 文は 1 回しか実行されません。

```

10    DO
20    FOR i=1 TO 3
30    PRINT i
40    NEXT
50    LOOP

```

これは、FOR 文が初期化式を含むためです。初期コンパイル時に、システムは NEXT 文に対して、FOR 文の TO 以後の場所を埋め込みます。NEXT 文はこの情報により、実行される毎に FOR 文のリミット値と STEP 値を評価し、ループする場合は、FOR 文の直後に制御を移します。このため、FOR 文自体は、ループ中では実行されません。

3) マルチタスクでの BREAK_POINT

以下のようなプログラムでは実行後にブレークポイントを設定できます。

設定されるとただちにブレークポイントは有効となります。つまり、実行中のプログラムでデバッグを開始することができます。あとの使い方はシングルタスクと同様です。

```

LIST
10    AAA=111: B=123
20    FORK 1 *aa
30    FORK 2 *bb
40    END
50    *aa
60    DO
70    FOR i_=0 TO 7
80    ON i_:TIME 200:OFF i_
90    NEXT
100   LOOP
110   *bb
120   DO
130   FOR i_=8 TO 15
140   ON i_:TIME 50:OFF i_
150   NEXT
160   LOOP
#run
#bkp 80
80    ON i_:TIME 200:OFF i_

```

```

##
80    ON i_ : TIME 200 : OFF i_  <01>
#
80    ON i_ : TIME 200 : OFF i_  <01>
#t
90    NEXT      <01>
#t
80    ON i_ : TIME 200 : OFF i_  <01>
#t
90    NEXT      <01>
?p i_
#PR i_-> 2

```

以下は、異なるタスクでブレークポイントを設定した場合です。ブレークしている最中に他のタスクのブレークが発生すると、待ち状態になります。このため、<ENTER> 実行を繰り返すと、タスク 1 とタスク 2 の交互のブレークの処理となります。

それぞれのブレークで、i_ の値を参照すると、タスクごとに異なった値となっています。

また、途中で 'u' を入力すると、80 番のブレークが解除され、以後はタスク 2 のみのブレークとなります。

```

#bkp 80 140

80    ON i_ : TIME 200 : OFF i_

140   ON i_ : TIME 50 : OFF i_
##
80    ON i_ : TIME 200 : OFF i_  <01>
#
140   ON i_ : TIME 50 : OFF i_  <02>
#
80    ON i_ : TIME 200 : OFF i_  <01>
#
140   ON i_ : TIME 50 : OFF i_  <02>
#
80    ON i_ : TIME 200 : OFF i_  <01>
?p i_
#PR i_-> 6
#
140   ON i_ : TIME 50 : OFF i_  <02>
?p i_
#PR i_-> 15
#
80    ON i_ : TIME 200 : OFF i_  <01>
?u
#
140   ON i_ : TIME 50 : OFF i_  <02>
#
140   ON i_ : TIME 50 : OFF i_  <02>
?

```

4) SLOW_RUN

装置を初めて動作させるには相当な慎重さが必要です。SLOW_RUN はこうした場合、プログラムの実行速度を遅くさせるものです。例えば以下のコマンドはタスク 10 を実行するのに、1 行ごとに 1000msec のタイマーをいれるものです。

```
SLOW_RUN 10 1000
```

引数はタスク番号と、1 行ごとの待ち時間です。最大 4000msec まで指定できます。

プログラムでは、WS0(),WS1() などのように、タイムアウト機能を持つものがあります。SLOW_RUN によってある特定のタスクの実行速度を遅くすると、タイムアウトによりデバッグに支障が生じます。こうした場合は、以下を実行します。

```
SLOW_RUN TMOUT
```


これによりタイムアウト時間は10倍となります。さらに余裕が必要な場合は、引数 10000 を追加すると、100 倍となります。これは、ダウンカウントタイマーが 100msec ごとに減算されるのを 10000msec つまり、10 秒ごとに減算するように設定するためです。

```
SLOW_RUN TMOUT 10000
```

グローバル変数とタスク・ローカル変数

BL/1 には、大きくわけて 2 種類の変数があります。グローバル変数とタスク・ローカル変数です。グローバル変数は、どのタスク、どこからでも使用できる変数です。通常の変数と考えてください。タスク・ローカル変数は、BL/1 独特の変数で、タスクごとに異なる値を持ちます。

例えば、以下のような例です。port_ のように末尾に '_' 記号を与えられた変数は、タスク・ローカル変数となります。以下の例で、サブルーチン、*ON_PORT は呼ばれたタスクごとに異なるポートをオンします。もし、このサブルーチンが複数のタスクから同時に使用された場合、port_ が通常の変数であると、複数のタスクが同時に同じ変数を使用してしまうために、処理が安定しなくなります。

port_ はタスクごとに独立した値を持つことができるため、こうした同一変数の共用による処理衝突がなくなります。

```
*ON_PORT
port_=X(TASKn)
ON port_
RETURN
```

しかし、タスク・ローカル変数はデバッグ時にどのような値をもっているかをモニタすることが困難になります。プログラムを停止させたあとに

```
print port_
```

を実行しても、実行させられたタスク、つまりタスク 0 の port_ の値しか参照できません。

これを解決するために、BL/1 では、

```
pra port_
```

のように pra コマンドを用意しています。pra コマンドは通常配列変数の要素を一括表示するのに用いますが、タスク・ローカル変数に対しては、タスクごとの値を表示します。

予約定数と予約変数

BL/1 にはあらかじめ用意されている予約定数と予約変数があります。予約定数とは、システム的に固定して使用する数値などが登録されています。

例としては、以下のようなものがあります。X_A は &H80000001 の値を持ち、RMVS コマンドでは、X 軸を指定します。このように予約定数は、コマンドの機能を効率よく記述できるように用意されています。

```
RMVS X_A 1000
```

予約定数は、相当数あり用途はコマンドにより個々様々です。こちらの情報については、Web 上のコマンドリファレンスで、【グループ】->【予約定数】条件絞込の上検索参照ください。

予約変数は、TASKn,SYSClk などのようにシステムが常時更新している変数です。

【予約変数】

グローバル変数	用途
SYSClk	1msec ごとに自動インクリメント CPU のクロック基準。
TASKn	自己タスク番号を返す変数。
SEC	1 秒ごとに自動インクリメント
PG_TASK0	タスク 0 に割り当てられている PG 番号を返す変数。 存在しない PG の場合は -1 となります。
MBK_ERR	MEWNET 通信のエラー回数

MBK_CMD	MEWNET 通信で処理できなかったコマンド。 prx MBK_CMD で 4142 であれば、AB という意味。
VER\$	バージョン文字列。バージョン番号は、MBK(8053) 参照。
CUM_PNT CUM_SRC CUM_NUM CUM_CNT CUM_ERR CUM_TASK	CUnet CU_POST で使用
FILE\$,FILE\$1,FILE\$2	USB メモリコマンド USB_WRITE で使用
CHK_SUM	プログラムのチェックサム ロード直後に同じプログラムかどうか照合
V_PGA,V_PGB	MPC-1000 のリターンバリュー 現在位置・バージョンなど
タスク変数	
timer_	タイムアウト処理 0.1 秒ごとにダウンカウトして 0 で停止します。
ptr_	文字列処理 文字列ポインタ
rse_	通信エラーステータス
err_	ON_ERROR 使用時のエラー情報

* 他にも予約変数がありますが、機能衝突が無ければ自由に使えます。

【予約定数】Vlist で " 定数リストとして表示されるもの "

データ型指定			
Lng	タッチパネル I/O	ロング型 (2 ワード) 指定	S_MBK,OUT
Wrd		ワード型指定	
Int		ワード型指定 (符号付)	MBK()
NIL		0 です。明示的に 0 を示す場合に使用。	
PG 割込設定			
CMP_PLS	MPG-2314	現在パルスカウンタと COMP+ と比較	INSET
CMP_CNT		エンコーダカウンタと COMP+ と比較	
C_MORE		カウンタ >=COMP+ で割り込み	INTA_ON/_OFF
C_LESS		カウンタ <COMP+ で割り込み	INTB_ON/_OFF
CUnet			
SA0 ~ SA15	MPC-Cunet	Cunet のステーションアドレスに対応した I/O 番号	ON/OFF SW()
SA0_B ~ SA15_B		Cunet のステーションアドレスに対応した I/O バンク番号	IN/OUT
通信			
EOL	シリアル通信	受信ターミネータの設定	INPUT#
CHR_C		受信文字数の指定	
CLR_BUF		受信バッファのクリア	
TMOUT		未受信タイムアウト指定	
LONG_PRG		プログラム番号のロング化	S_MBK
B7N,B7E,B7O B8E,B8O	タッチパネル	フレーム・パリティ指定	MEWNET
Ub,Lb	タッチパネル	上位、下位バイト指定	IN() *MBK I/O I17
CompoWay COMPOWAY	シリアル通信	OMRON コンポウェイ指定	PRINT#
RS485,RTS	シリアル通信	RTS 制御を行う RS-485 通信	PRINT#

その他			
_NEXT	制御文	RESUME のオプション	RESUME
OFF	SENSE_ON/_OFF	SENSE_ON/OFF のオプション	SENSE_ON/_OFF
AVOID	IO	コマンドの無効化	ON,OFF,OUT, PULSE_OUT
ON_USB	USB	USB イネーブルポート	MPC-1000
USB,USB0,USB1 USB2,COM		USB チャンネル指定	USB_WRITE, INPUT#
AD7890-10	AD	AD7890-10 換装時	SET_AD
AD0,AD1		AD ボード選択	
SET_SF	NC コマンド	GET_CODE のオプション	GET_CODE
ALLOW	未使用	将来抹消	
パルス発生			
SACL	MPG-2314 MPC-1200	S 字指定	ACCEL
VOID		無効引数	MOVS,MOVL
X_A		X 軸指定	PG 全般
Y_A		Y 軸指定	
Z_A		Z 軸指定	
U_A		U 軸指定	
ALL_A		全軸指定	
VOID_X		X 軸を除く =Y_A U_A Z_A	
VOID_Y		Y 軸を除く =X_A U_A Z_A	
VOID_Z		Z 軸を除く =X_A Y_A U_A	
VOID_U		U 軸を除く =X_A Y_A Z_A	
X_E		MPG-2314	
Y_E	Y 軸エラー指定		
Z_E	Z 軸エラー指定		
U_E	U 軸エラー指定		
ALL_E	全軸エラー指定		
POS_L	MPG-2314	正の大数	HOME
NEG_L	MPC-1200	負の大数	
XIN0 ~ XIN3	MPG2314	HPT 入力指定 IN1,IN2 ショート IN3 非接続	HPT()
XINP,XALM			
YIN0 ~ YIN3			
YINP,YALM			
UIN0 ~ UIN3			
UINP,UALM			
ZIN0 ~ ZIN3			
ZINP,ZALM			
STP_I	MPG-2314	即停止	STOP
STP_D	MPC-1200	減速停止	
INP_ON	MPG-2314	インポジション ON で有効	INSET
INP_OFF		インポジション OFF で有効	
INP_NO		インポジション無効	
ALM_ON		アラーム ON で有効	
ALM_OFF		アラーム OFF で有効	
ALM_NO		アラーム無効	
NO_PHASE		カウンタ入力	
PAHSE1		1 倍エンコーダ入力	

PHASE2	MPG-2314	2倍エンコーダ入力	INSET	
PHASE4		4倍エンコーダ入力		
UP_DWN		アップダウンカウンタ		
MD_2PLS		CW/CCWパルス出力		
MD_DPLS		方向指示パルス出力		
LMT_ON		X-LMT～Z-LMT ON 有効		
LMT_OFF		X-LMT～Z-LMT OFF 有効		
SLMT_ON		ソフトリミット有効		
SLMT_OFF		ソフトリミット無効		
IN0_ON		XIN0～ZIN0 ON で有効		
IN0_OFF		XIN0～ZIN0 OFF で有効		
IN1_ON		XIN1～ZIN1 ON で有効		
IN1_OFF		XIN1～ZIN1 OFF で有効		
IN2_ON		XIN2～ZIN2 ON で有効		
IN2_OFF		XIN2～ZIN2 OFF で有効		
IN3_ON		XIN3～ZIN3 ON で有効		
IN3_OFF		XIN3～ZIN3 OFF で有効		
CW		MPG-2314		円弧補間指定
CCW	円弧補間指定			
SLMTp	MPG-2314	ソフトリミット + エラー	LMT() or PGE()	
SLMTn		ソフトリミット - エラー		
LMTp		リミット + エラー		
LMTn		リミット - エラー		
EMG		EMG 入力エラー		
ALM		ALM 入力エラー		
IN0～IN3		停止原因		PGE()
CRL_ER		エラーリセット		
X_C	MPG-2314	カウンタ指定	STPS	
Y_C		カウンタ指定		
Z_C		カウンタ指定		
U_C		カウンタ指定		
VRING	MPG-2314	リングカウンタ設定	その他	
PR_CHK		動作事前チェック		
PGA,PGB	MPC-1000	PG アクティブ指定	MPC-1000 のみ	

* 予約定数は、読み出しのみ可で代入しようとするとエラーになります。

データ領域

任意に使用できる変数とは別に、予約された配列として、MBK(),X(),Y(),U(),Z()があります。MBK()はタッチパネル用の配列ですが、タッチパネルを使用しない場合は汎用のメモリアリアとして使用できます。

X()～Z()は、産業ロボットの用途に使用する点データです。点データとして使用しない場合は、MBK()と同様、配列変数として使用できます。また、DIM コマンドにより二次元迄の配列変数を定義、使用する事ができます。

配列要素	型と範囲	目的	変更方法
MBK(n)	ワード (2bytes) 0～8099	タッチパネルとの共有メモリ	S_MBK MBK(n)=m
X(n),Y(n),U(n),Z(n) PG コマンドでは P(n)として使用	ロング (4bytes) 点データ数は機種により異なります。ハードリファレンス参照	ロボット座標点データ あるいはデータエリア	SETP X(n)=m
DIM コマンドによる配列	合計で 20000 個まで	-	-

文字列変数

文字列変数は、末尾に \$ を与えられた変数です。文字列は、128 個まで使用することができ、ひとつの文字列のサイズは 255 バイトまでとなっています。

文字列演算もサポートされており、結合は '+' 演算子によって行うことができます。

```
#a$="12345"+chr$(&h41)+"bcdef"
#pr a$
12345Abcdef
```

文字列の検索編集は、BASIC スタンドの MID\$ を中心にした方法ではなく、ポインタであるタスクローカル変数 "ptr_" を用いた C 言語風な処理を可能としています。(参照: SERCH, SERCH\$, VAL, STRCPY 等)。文字列中の数値の取り出しについては、強力な VAL 関数を用意しています。上の文字列 a\$ の数値変換処理は以下のように記述できます。

```
#pr val(a$)
12345
#
```

なお、文字列変数は、通常の算術式ではポインタ (実際のアドレス) として扱われます。このため、以下のような操作で、文字列の切り出しを自由に行うことができます。

```
10 a$="1234567890abcdefgABCDEFGH"
30 SERCH a$ "a"
35 s=ptr_-1 : e=SERCH$("A") : c=e-s-1
40 ptr_=s
50 c$=PTR$(c)
60 PRINT c$
#run
abcdefg
#
```

算術式

BL/1 の算術式は、加算、減算に対して乗算、除算のみ優先となりますが、他は左から順に実行されます。そのほかの優先演算の場合は、() で閉じます。

```
a=1+2*3          a は 7 となります。2*3=>6 1+6=>7 という順序で実行されるためです。
a=(a1+a2+a3+a4)/4  a には、a1 から a4 の総和を四で割った値 (平均値) がはります。
c=sqr(a*b)       a と b の平方和の平方根となります。
```

なお、式 (条件式とも) の長さは 102 文字以内としており、相当長い式も記述できますが、() があまりに多いと内部メモリを浪費しスタック・オーバーフローとなる場合があります。演算は、簡潔に効率よく記述してください。

二項演算子			
+	加算	<<	左シフト (× 2n)
-	減算	>>	右シフト (/2n)
*	乗算	,	ワード合成
/	除算	;	上位バイト
%	剰余算	&	論理積
^	排他的論理積		論理和

条件式

条件式（論理式）と算術式の相違は、算術式の結果が整数になるのに対して論理式の結果は 1(真)0(偽)の値しかとらないことです。

```
a==5    a と 5 比較して 1 か 0 を得ます。等しければ 1(真) 等しくなければ、0(偽) となります。
```

論理式は以下のように IF 文などの引数となります。真であれば、THEN の直後から ELSE までもしくは、END_IF までを実行します。

```
IF a==5 THEN : ON 1 : ELSE : ON 2 : END_IF
```

```
a==5&(b==3)
```

この例では、a を 5 と比較し、b と 3 を比較した結果と AND をとります。従って、全体として真となるのは、a が 5、b が 3 という 2 つの条件を満たした時となります。同様に以下の場合には、OR です。

b==3 を () で閉じているのは、この比較演算子を優先させるためです。

```
a==5|(b==3)
```

この場合は、a が 5 であるか、b が 3 であれば真(1)となります。

こうした論理式が実際に真となるか、偽となるかは、

```
print a==5|(b==3)
```

で確認することができます。

この 1,0 の値をとるという論理ルールに従えば論理式を簡素化することができます。簡素化は、処理の高速につながります。

例えば、WAIT は引数である論理式が真(1)になるまで、待ち続けるコマンドで、条件待ちを以下のように記述できます。

```
WAIT (SW(0)==1)& (SW(2)==1)& (SW(4)==1)& (SW(7)==1)& (SW(-1)==1)
```

↓

```
WAIT SW(0)& SW(2)&SW(4) & SW(7) & SW(-1)
```

これは SW 関数が真の時に 1 の値を出力するために可能な簡素化です。

論理反転を必要とする場合は以下のように記述できます。

```
WAIT (SW(0)==1)& (SW(2)==1)& (SW(4)==0)& (SW(7)==0)& (SW(-1)==1)
```

↓

```
WAIT SW(0)& SW(2)&@SW(4) & @SW(7) & SW(-1)
```

@SW() 関数は SW() 論理反転です。

論理演算では、単純な論理比較だけでなく、以下のように文字列、配列などが混ざった論理式も記述可能です。この場合は、比較式を () で閉じて & や | で結合させます。

```
IF (a$=="123")&(b==100)&(mbk(5)>1000) THEN
```

なお IF 文では引数間の AND,OR も可能です。

```
IF a==1 AND b==2 THEN
```

一般に複数の要素を結合する複雑な論理演算では 1 つの式に組み立てた方が高速処理になりますが、判読しづらくなります。

意味ごとに式をまとめて、最終的に AND OR によって論理を結合したほうが、プログラムとして見通しが良くなります。

論理演算子 (結果が 1 か 0 になる)

==	一致	<	小さい
!={<>}	不一致	>=	同じか大きい
>	大きい	=<	同じか小さい

制御文

1) 繰り返し及び条件待ち

WHILE 条件式 ~ WEND
DO ~ LOOP、FOR ~ NEXT
BREAK
WAIT

条件付き繰り返し
繰り返し、順次処理
繰り返しから抜ける
条件式

■ 無限ループ

```
DO
  ON 0
  TIME 500
  OFF 0
  TIME 500
LOOP
```

■ 10 回繰り返し

```
FOR CNT=1 TO 10
  ON 0
  TIME 500
  OFF 0
  TIME 500
NEXT CNT
```

/* CNT を 1 つ加算して FOR に戻る

■ 繰り返しから BREAK で抜ける

```
CNT=0
DO
  CNT=CNT+1
  IF CNT>10 THEN : BREAK : END_IF
  ON 0
  TIME 500
  OFF 0
  TIME 500
LOOP
```

/* DO~LOOP から抜ける

■ 繰り返しから GOTO で抜ける

```
CNT=0
DO
  CNT=CNT+1
  IF CNT>10 THEN : GOTO *PASS : END_IF
  ON 0
  TIME 500
  OFF 0
  TIME 500
LOOP
*PASS
```

/* DO~LOOP から抜ける

※ FOR ~ NEXT から BREAK や GOTO で抜けることも出来ます。

■ 条件待ち

```
WAIT SW(192)==1
```

/* SW(192) が ON になるのを待つ

2) 条件分岐

IF ~ THEN ~ [ELSE] ~ END_IF
SELECT_CASE ~ END_SELECT

分岐
数値による分岐

■ IF 文。シンプルな条件判断向き。

```
IF SW(195)==1 THEN
  GOTO *MANU
ELSE
```

/* フロントパネルセクタ SW がオンならば

/* そうでなければ

```

GOTO *AUTO
END_IF
*MANU
OFF 0 : ON 1
PRINT "MANUAL MODE"
END
*AUTO
ON 0 : OFF 1
PRINT "AUTO MODE"
END

```

- SELECT_CASE 文。いくつもの条件があるとき。

```

OFF 0 : OFF 1 : OFF 2          /* LED 消灯
DSW=IN(24)/16                 /* フロントパネル DSW 読み込み
SELECT_CASE DSW               /* DSW の値を調べる
CASE 0                         /* DSW=0 なら
    ON 0 : OFF 1 : OFF 2
CASE 1                         /* DSW=1 なら
    OFF 0 : ON 1 : OFF 2
CASE 2                         /* DSW=2 なら
    OFF 0 : OFF 1 : ON 2
CASE_ELSE                     /* それ以外なら
    ON 3 : TIME 10 : OFF 3
END_SELECT

```

- SELECT_CASE の引数を VOID とすると、CASE 文独自の論理式を評価して実行します。複数の条件を調べなければならないとき、IF 文を羅列するより効率的です。

```

SELECT_CASE VOID
CASE SW(192)==1                /* 緑 SW ON -> 緑 LED ON
    ON 0
    OFF 1 2
CASE SW(193)==1                /* 黄 SW ON -> 黄 LED ON
    ON 1
    OFF 0 2
CASE SW(194)==1                /* 赤 SW ON -> 赤 LED ON
    ON 2
    OFF 0 1
CASE SW(195)==1                /* セレクタ SW 右側 -> LED OFF
    OFF 0 1 2
CASE_ELSE
END_SELECT

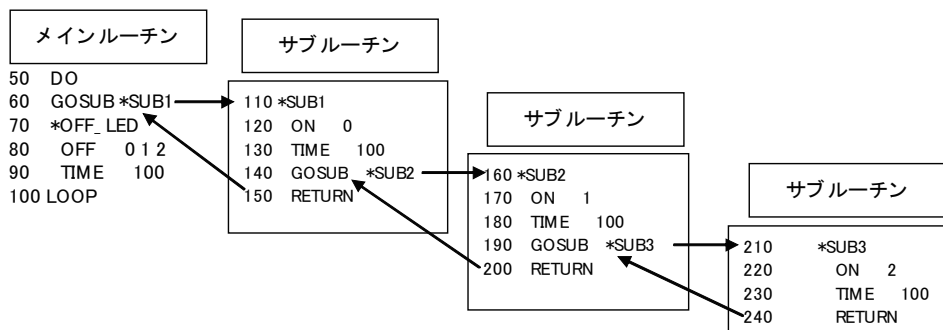
```

3) サブルーチン

GOSUB、RETURN
_VAR、_RET_VAL

サブルーチンへ飛ぶ、サブルーチンから戻る
サブルーチンへの引き数、サブルーチンからの戻り値

仕事単位でサブルーチン化して、それをメインルーチンから呼ぶようにするとプログラムが読み易くなります。GOSUB ばかりで RETURN が無いと "スタックが溢れました" エラーになります。



- GOSUB にはサブルーチンへの引き数を与えることができます。サブルーチン内では _VAR コマンドで値を取得します。
- RETURN の後ろにサブルーチンからの戻り値を与られます。戻り値の取得は _RET_VAL です。
- ローカル変数と組み合わせるとタスク間でのサブルーチンの共有が可能となります。

```

10    OFF 0 1 2
20    TIME 500
30    GOSUB *ON_LED 0 1 2      /* 引き数付きでサブルーチンコール
40    END
50    *ON_LED
60    _VAR A_ B_ C_          /* 引き数を受け取る ( _ 付きはローカル変数)
70    ON A_
80    TIME 500
90    ON B_
100   TIME 500
110   ON C_
120   RETURN

10    GOSUB *READ_DSW
20    _RET_VAL D             /* 戻り値を受け取る
30    PRINT D
40    END
50    *READ_DSW
60    DSW_=IN(24)/16        /* フロントパネル DSW 読み込み
70    RETURN DSW_          /* DSW_ は戻り値

```

ON_ERROR

BL/1 でプログラム実行中にエラーが発生すると、そこでプログラムの実行は停止します。通常、エラーは致命的であるために、そのエラーにもとづいてプログラムをエラーが発生しないように修正します。

プログラムの開発中にはそれで十分なのですが、実際に稼働を始めるとプログラムの停止は好ましくありません。ON_ERROR は実行中のエラーを捕らえて、エラー処理プログラム回避させることができます。ON_ERROR 処理では、エラーコードとエラー発生文番号をタスク変数 err_ に収納します。エラーコードは、err_>>24 で得られます。

なお、エラー処理プログラム中でさらにエラーが発生した場合は、エラーは表示されるのみエラー処理プログラムへのジャンプはありません。エラージャンプ禁止状態が解除されるのは、GOTO コマンド、RESUME コマンドが実行された場合のみです。エラーコードは巻末のエラーコード表を参照してください。

1) 回復可能な場合

USB メモリのような外部機器では、不良や劣化、あるいは信頼性の不足などで、ランタイムエラーが発生します。この場合は、RST_USB などの処置を実施して処理をリトライさせます。こうした場合に、プログラム制御をもとの位置に戻すコマンドとして RESUME があります。

```

RESUME          エラーの発生したコマンドに戻す。
RESUME _NEXT    エラーの発生したコマンド次の行に戻す。

```

2) 回復不能な場合

稼働中に I/O 番号の間違いや変数の設定間違いで発生するエラーは、後日プログラム修正する必要があります。この場合はエラー処理の中で、タッチパネルなどにエラー文字列と発生箇所を通知するという処理をします。

```

ON_ERROR *err
FILE$="TEST.TXT"
DO

```

```

USB_WRITE "TEST%n"
OUT 0 -10000
LOOP
*err
SELECT_CASE err_>>24
CASE 53
CASE 54
CASE 55
CASE 56
RST_USB :TIME 500 :INC usb_err :RESUME
CASE_ELSE
S_MBK err_>>24 100 : S_MBK err_&&H00FFFFFF 101 : S_MBK ERR$(err_) 102 40
ON PATRIGHT
END_SELECT
END
#run
#pr mbk(100)
9
#pr mbk(101)
50
#pr mbk$(102,40)
I/O 範囲を越えています
##

```

SELECT_CASE VOID の用法

SELECT_CASE には拡張された記述方法があります。
通常では以下のように変数値の分類制御になります。

```

DO
SELECT_CASE A
CASE 100 : FORK 1 *SHORI : WAIT SW(192)==0
CASE 101 : FORK 1 *SHOR2 : WAIT SW(193)==0
CASE 102 : FORK 1 *SHOR3 : WAIT SW(194)==0
CASE_ELSE
END_SELECT
LOOP

```

しかし、以下のように引数を VOID とすると、CASE 文が独立した論理評価を実行します。
並立する条件から排他的に処理をさせることができるため、複雑な IF_ELSE 構文を無くしてスッキリした記述が可能になります。

```

DO
SELECT_CASE VOID
CASE SW(192)==0 : FORK 1 *SHORI : WAIT SW(192)==0
CASE SW(193)==1 : FORK 1 *SHOR2 : WAIT SW(193)==0
CASE SW(194)==0 : FORK 1 *SHOR3 : WAIT SW(194)==0
CASE_ELSE
END_SELECT
LOOP

```