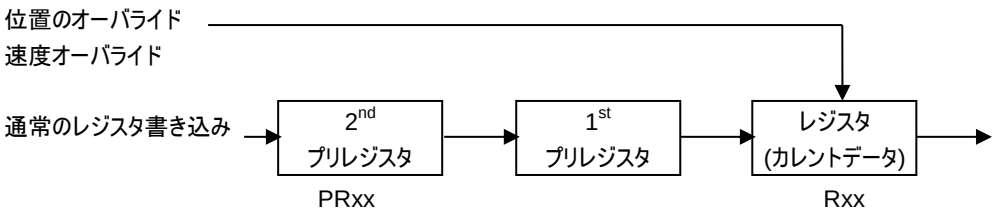
3.2 次動作連続実行

動作用プリレジスタ(2nd プリレジスタ，1st プリレジスタ)を切れ目なく使うことを「次動作連続実行」といいます．

3.2.1 動作用プリレジスタ

プリレジスタとは動作中に次の動作用データを予約するレジスタです．
RMV，RFL，RFH，RUR，RDR，RMG，RDP，RMD，RIP，RUS，RDS，RCI とスタートコマンドにはプリレジスタがあります．



停止中にプリレジスタに書き込まれたデータはシフトされカレントデータとして動作します．
動作中に書き込まれたデータは次の動作の予約となり，カレントデータの動作終了で自動的に動作が開始されます．
各レジスタはスタートコマンドの書き込みで確定します．
動作用プリレジスタのデータは動作用プリレジスタキャンセル，減速停止，即停止の各コマンドでキャンセルされます．
また，エラー停止でもキャンセルされます．

3.2.2 動作用プリレジスタ制御コマンド

No.	コマンド	略　称	CMD	機　　能	使用する関数
1	動作用プリレジスタキャンセル	PREGCAN	26h	動作プリレジの内容をキャンセル	wCmdW
2	動作用プリレジスタシフト	PREGSFT	2Bh	動作プリレジの内容をシフト	wCmdW

3.2.3 プリレジスタの状態変化

動作用プリレジスタへの書き込み，スタートコマンドの書き込み，動作完了により記憶状態，レジスタの内容がどのように変化するかPRMVレジスタを例にして説明します．

(1) 停止状態でPRMVに"1000"を書き込む

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	1000	未確定	00	0
(1stプリレジスタ)	1000	未確定		
RMV(レジスタ)	1000	未確定		

(2) スタートコマンド書き込みにより動作開始

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	1000	未確定	01	0
(1stプリレジスタ)	1000	未確定		
RMV(レジスタ)	1000	確定		

(3) 動作中に次動作データPRMV=-5000を書き込む(前回と同じ内容の時は省略できます)

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	*-5000*	未確定	01	0
(1stプリレジスタ)	*-5000*	未確定		
RMV(レジスタ)	1000	確定		

(4) 次動作用スタートコマンドを書き込む(1stが確定)

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	-5000	未確定	10	0
(1stプリレジスタ)	-5000	確定		
RMV(レジスタ)	1000	確定		

(5) 動作中に次々動作データPRMV=3000を書き込む(前回と同じ内容の時は省略できます)

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	*3000*	未確定	10	0
(1stプリレジスタ)	-5000	確定		
RMV(レジスタ)	1000	確定		

(6) 次々動作用スタートコマンドを書き込む(2ndが確定)　MSTS.b14=1 (プリレジスタ満杯状態)

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	3000	確定	11	1
(1stプリレジスタ)	-5000	確定		
RMV(レジスタ)	1000	確定		

(7) 最初の動作完了　MSTS.b14=0 (2ndが未確定), MSTS.b3=1, MSTS.b0=1

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	3000	未確定	10	0
(1stプリレジスタ)	*3000*	確定		
RMV(レジスタ)	*-5000*	確定		

(8) 次動作完了　MSTS.b14=0(1st,2ndが未確定), MSTS.b3=1, MSTS.b0=1

	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	3000	未確定	01	0
(1stプリレジスタ)	3000	未確定		
RMV(レジスタ)	*3000*	確定		

(9) 次々動作完了　MSTS.b14=0 (全レジスタが未確定), MSTS.b3=1, MSTS.b0=0

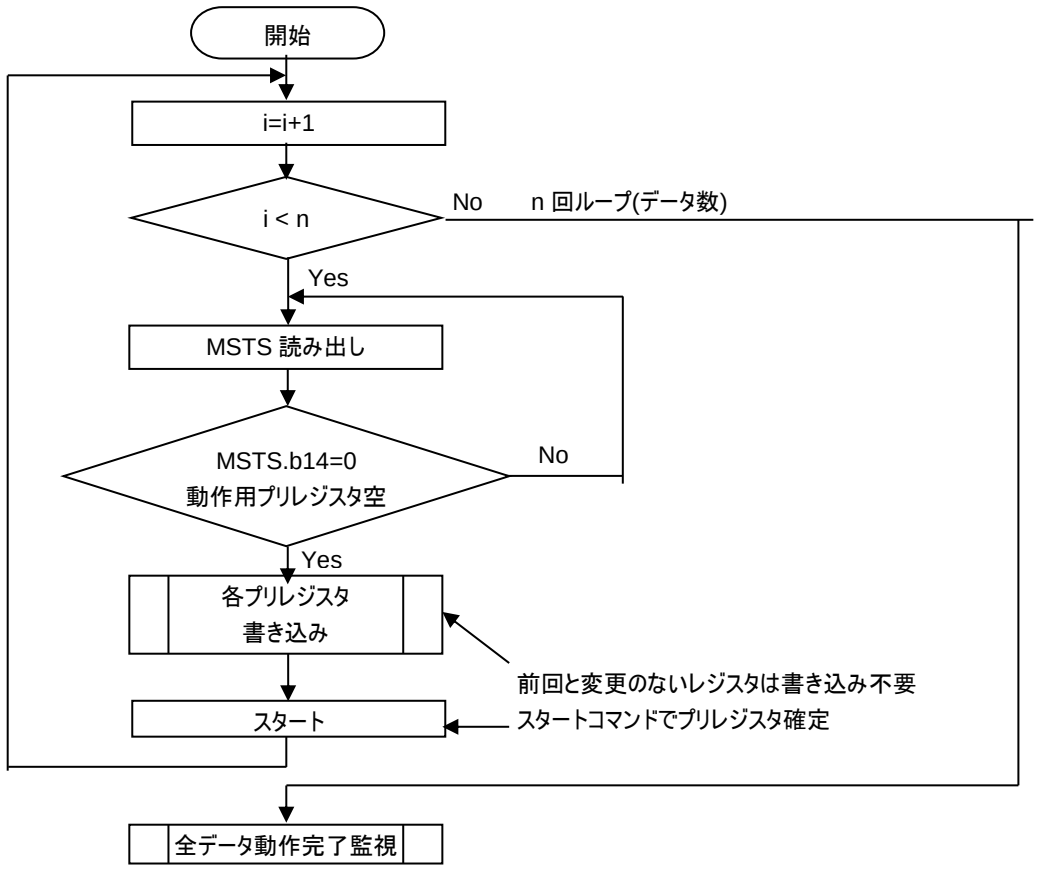
	内 容	記憶状態	RSTS.b21,20	MSTS.b14
PRMV(2ndプリレジスタ)	3000	未確定	00	0
(1stプリレジスタ)	3000	未確定		
RMV(レジスタ)	3000	未確定		

3.2.4 手順

MSTS をﾁｪｯｸしながら，以下のように運用します．

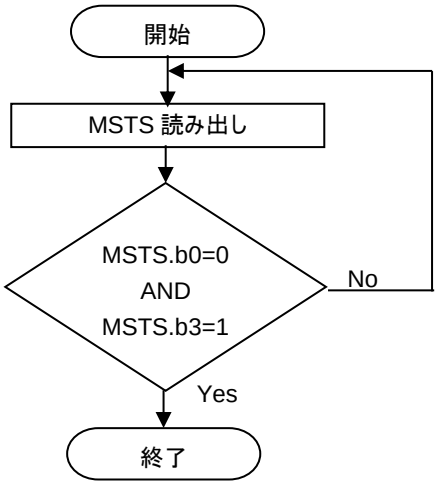
(1) プリレジスタへの書き込み

MSTS.b14=1(動作用プリレジスタフル)を検出したら，b14=0(動作用プリレジ空き)になるまで待ち，b14=0 になったら，次のデータをプリレジスタに書き，スタートコマンドを書きます．以上の処理を繰り返します．



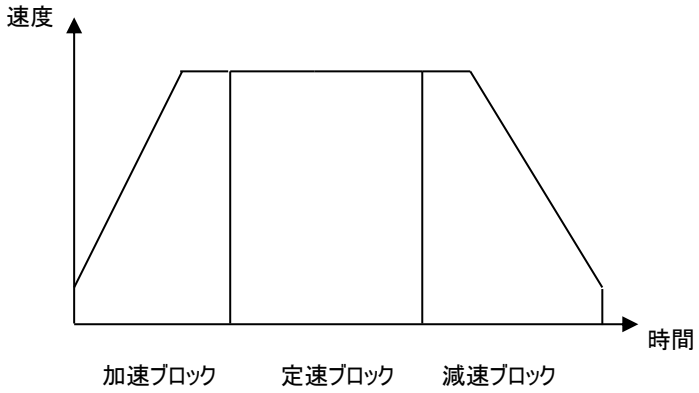
(2) 動作の完了監視

全てのデータの書込みが終了したら，全ての動作が完了するのを待ちます．
MSTS.b0=0(スタートコマンドが書込まれていない)，かつ MSTS.b3=1(停止中)の状態になるまで MSTS を監視します．



3.2.5　加速ブロック，定速ブロック，減速ブロック

次動作連続実行時に動作の初めに加速を追加する場合は「加速ブロック」，動作の最後に減速を追加する場合は「減速ブロック」を使用します．



(1) 加速ブロック作成方法

- 減速開始点手動計算(RMD.b13=1)
- 減速開始点(RDP=0)
- 加速スタート(53h)，AccStart

(2) 定速ブロック作成方法

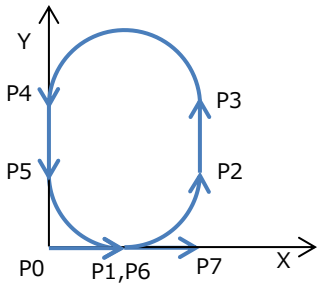
- FH 定速スタート(51h)，CnstStartFH

(3) 減速ブロック作成方法

- 減速開始点手動計算(RMD.b13=1)
- RFL,RFH,RDR,RMG から計算し減速開始点(RDP)を設定します．(計算方法は「2.2.5 速度と加速度」参照)
- FH 定速スタート後減速停止(52h)，CnstStartByDec

3.2.6　次動作連続実行例

下図の様な補間動作を，次動作連続実行で行います. (HPCI-CPD532, 534, 574N, 578N, 508 等の例)
速度等の設定は，ベース速度 500pps(RFL=500)，動作速度 5000pps(RFH=5000)，速度倍率 1 倍 (RMG=299)，加減速時間約 0.1sec(RUR=108, RDR=0)，直線加減速とします.



	位置(X,Y)
P0	(0,0)
P1	(1000,0)
P2	(2000,1000)
P3	(2000,2000)
P4	(0,2000)
P5	(0,1000)
P6	(1000,0)
P7	(2000,0)

	XYPRMD	XRMV	YRMV	XRIP	YRIP	XRDP	備考
P0→P1	0800A061h	1000	0	0	0	0	直線補間(加速ブロック)
P1→P2	0800A065h	1000	1000	0	1000	0	CCW 円弧補間(定速ブロック)
P2→P3	0800A061h	0	1000	0	0	0	直線補間(定速ブロック)
P3→P4	0800A065h	-2000	0	-1000	0	0	CCW 円弧補間(定速ブロック)
P4→P5	0800A061h	0	-1000	0	0	0	直線補間(定速ブロック)
P5→P6	0800A065h	1000	-1000	1000	0	0	CCW 円弧補間(定速ブロック)
P6→P7	0800A061h	1000	0	0	0	274(※)	直線補間(減速ブロック)

※.減速開始点の計算(RDR=0 の場合は RUR=RDR として計算します)

$$減速開始点[パルス]=\frac{(RFH^2-RFL^2)\times(RDR+1)}{(RMG+1)\times 32768}$$

[C 言語記述例]

```
DWORD h;                    //デバイスハンドル
// 連続実行データの構造体(X 軸：補間代表軸)
typedef struct {
    long            rmd;    //X,Y 動作モード
    long            xmv;    //X 移動量
    long            ymv;    //Y 移動量
    long            xcn;    //X 中心
    long            ycn;    //Y 中心
    long            rdp;    //X 減速開始点
    unsigned short  cmd;    //X スタートコマンド
} DATABLK;

// 設定データ
DATABLK dt_blk [7] = {
 //動作モード, X 移動量, Y 移動量, X 中心, Y 中心, 減速開始点,  コマンド
    {0x800A061,    1000,      0,      0,      0,          0, 0x353}    //dt_blk[0] 直線補間加速スタート
    {0x800A065,    1000,   1000,      0,   1000,          0, 0x351}    //dt_blk[1] CCW 円弧補間 FH 定速スタート
    {0x800A061,       0,   1000,      0,      0,          0, 0x351}    //dt_blk[2] 直線補間 FH 定速スタート
    {0x800A065,   -2000,      0,  -1000,      0,          0, 0x351}    //dt_blk[3] CCW 円弧補間 FH 定速スタート
    {0x800A061,       0,  -1000,      0,      0,          0, 0x351}    //dt_blk[4] 直線補間 FH 定速スタート
    {0x800A065,    1000,  -1000,   1000,      0,          0, 0x351}    //dt_blk[5] CCW 円弧補間 FH 定速スタート
    {0x800A061,    1000,      0,      0,      0,        274, 0x352}    //dt_blk[6] 直線補間加速スタート後減速停止
};
```

次ページに続く

```
short               ptr = 0;        //データポインタ
unsigned short      sts[2];         //メインステータス, サブステータス
unsigned long       ersts;          //エラーステータス,
unsigned long       evsts;          //イベントステータス

// 動作用プリレジスタフル(メインステータスの bit14)を監視しながらデータ設定
while (ptr<7) {                     // ptr が 7 未満の間ループ
    //動作用プリレジスタフル(メインステータスの bit14)を監視
    while (1) {
        cp530_rStsW(h, 0, sts);         //ステータス読込み
        if(0x10==(sts[0] & 0x10)) {             //エラー停止
        //エラー停止時の処理はアプリケーションに合わせて適宜処理してください.
        //以下は一例です.
            cp530_rReg (h, 0, 0xf2, &ersts);        //X:エラーステータス読込
            cp530_rReg (h, 1, 0xf2, &ersts);        //Y:エラーステータス読込
            return;                                 //処理キャンセル(終了)
        }
        if(0==(sts[0] & 0x4000)) {
                                    break;      //動作用プリレジスタフルでなければループを抜ける
        }
    }
    // データ書込み
    cp530_wReg (h, 0, 0x87, dt_blk[ptr].rmd);       //X:動作モード
    cp530_wReg (h, 1, 0x87, dt_blk [ptr].rmd);      //Y:
    cp530_wReg (h, 0, 0x80, dt_blk[ptr].xmv);       //X:移動量
    cp530_wReg (h, 1, 0x80, dt_blk [ptr].ymv);      //Y:
    cp530_wReg (h, 0, 0x88, dt_blk[ptr].xcn);       //X:中心位置
    cp530_wReg (h, 1, 0x88, dt_blk [ptr].ycn);      //Y:
    cp530_wReg (h, 0, 0x86, dt_blk[ptr].rdp);       //X:減速開始点
    cp530_wCmdW(h, 0, dt_blk[ptr].cmd);             //スタート(X,Y)
    ptr++;
}

//全書込みデータの動作完了を待つ
while (1) {
    cp530_rStsW(h, 0, sts);             //ステータス読込
    if(0x10==(sts[0] & 0x10)) {             //エラー停止
        //エラー停止時の処理はアプリケーションに合わせて適宜処理してください.
        //以下は一例です.
        cp530_rReg (h, 0, 0xf2, &ersts);        //X:エラーステータス読込
        cp530_rReg (h, 1, 0xf2, &ersts);        //Y:エラーステータス読込
        return;                                 //処理キャンセル(終了)
    }
    if(0x20==(sts[0] & 0x20)) {
        cp530_rReg (h, 0, 0xf3, &evsts);        //X:イベントステータス読込
        cp530_rReg (h, 1, 0xf3, &evsts);        //Y:イベントステータス読込
    }
    if(8==(sts[0] & 9)) {
                            break;      //メインステータス bit0=0 かつ bit3=1 で動作終了
    }
}
```