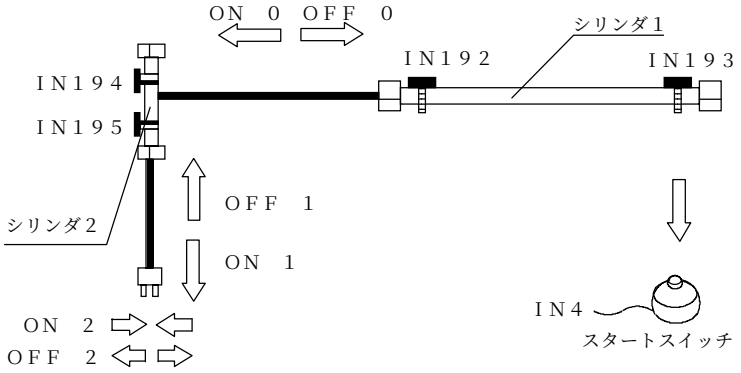
第２章　インタプリタ言語でのプログラミング

２．１　コントローラのいきさつについて

現在のコンピュータの姿が確立したのは１９７０年頃のことです。またソフトウェア技術についても同時期に現在の基礎はできあがっていました。しかし当時のコンピュータはまだ実用化の初期段階でとても高価でしたし、デリケートでもあったため現場の機械の制御をしたり、その大きさから装置に埋め込んで使用するなどということは不可能でした。コンピュータは主に政府大企業の中の膨大なデータを処理するためだけに用いられていました。それでも１９７０年ころのアメリカの自動車産業は最盛期を迎えており巨大な工場と多くの制御を必要とする装置がありました。当時の装置はリレー回路による論理制御と、カムによる制御がほとんどでした。機械設計者はカラクリ仕掛をつくるように綿密に装置とカムを設計し機械自体で動くものをつくりました。ただこの方法では装置に柔軟性をもたせることが難しく調整完了までに熟練工と時間を必要としていました。またリレー回路による制御では、プログラム変更は配線変更を意味し、ちょっとしたシーケンス変更も多くの作業を必要としました。アメリカの自動車工場のなかでこうしたリレー回路制御をなんとかもう少し保守性が高く、作業のしやすいものができないかということで現在のシーケンサのルーツになるものが考案されたそうです。そしてこの１９７０年はマイクロプロセッサ黎明期でもありました。この時代は大変多くのことが同時進行していたわけです。やがて１９８０年になるとマイクロプロセッサは量産されるようになり次第に機器制御に用いられるようになってきました。そして、その中でシーケンサと呼ばれる専用のコントローラが市販されるようになってきました。シーケンサはリレー回路をマイコンで模するもので、リレー回路の欠点を補いコンパクトかつ保守性の高いものです。当初のものは制御点数も少なくプログラムの入力方法も機械語に近い状態のものでしたが、最近では１０００点近い規模とリレー回路をそのまま書き込める回路設計のしやすいものになってきました。シーケンサはリレー回路を模するという初期の目的に忠実に発展してきました。現在でもシーケンサのプログラミング言語の主流はラダー言語と呼ばれるリレー回路を記述するものとなっています。しかし制御の実態は単純なＩ／Ｏ制御からＮＣ制御へと主流が移ってきています。安価になったステップモータやサーボモータが専用機に組み込まれそれとともに通信も必要になってきています。制御に対する考え方（言語）はそのままですが対象は大きく変わってきています。これに対してマイクロプロセッサ技術はハード・ソフトとともに長足の進歩をとげ現在では、卓上型のパソコンですら１９７０年の先端コンピュータの性能をしのぎソフトウエアも昔とは比較にならない強力なマンマシンインターフェースを備えています。シーケンサがリレー回路の域を脱出しないのに対して対照的です。私どものＭＰＣ－６８４はＮＣ制御、通信を目的に企画されています。シーケンサのラダー言語では数値・通信を素直に扱うことができません。そのことが、装置の柔軟性・機能を損ねているのではないかと考えられます。ＡＤＶＦＳＣはＢＡＳＩＣライクなインタプリタです。しかし、通常のＢＡＳＩＣとは大きくことなり、Ｉ／Ｏ制御やパルス発生をもともとの機能として含んでおり制御用インタプリタとして開発されています。当初ＢＡＳＩＣインタプリタはコンピュータのビギナのために考案された言語ですがその簡便性と実用性のため現在でも広く使用されている言語です。デバッグや昨今流行の構造的プログラミングという観点ではやや難がありますが一人でコーディンクできる程度のアプリケーションにはその起動性は魅力的です。こうした環境を制御の分野に実現しようとしたのがＭＰＣ－６８４です。

２．２　インタプリタ言語でのプログラミング

ラダー言語とインタプリタ言語（ＭＰＣ６８４ではＡＤＶＦＳＣ）ではプログラミングの考え方がまったく異なっています。(この章はラダー言語をご存知の方のためです。ＭＰＣのみ検討されている方は読み飛ばしてください)ラダー言語の基本は電子回路で物事を考えます。たとえばこのセンサとあのセンサがＯＮしたらこのソレノイドを動作するというようにです。これに対してＡＤＶＦＳＣでは装置を時間の流れの中でとらえます。一番に何々して二番目に何々してというふうにシーケンスを字義どおり考えます。制御の分野の人の中にはときどき機械の動作を考えた瞬間にリレー回路が頭の中に浮かぶ人も多いと思います。たまにある誤解はこんな方がＡＤＶＦＳＣでこうしたリレー回路を表現することです。残念ながらＡＤＶＦＳＣはリレー回路を表現することは得意ではありません。リレー回路の観点にたてばＡＤＶＦＳＣは良い制御言語とは言えません。ここでは、単純なＰ＆Ｐの制御例としてとりあげてみます。構成は次のようなものとします。

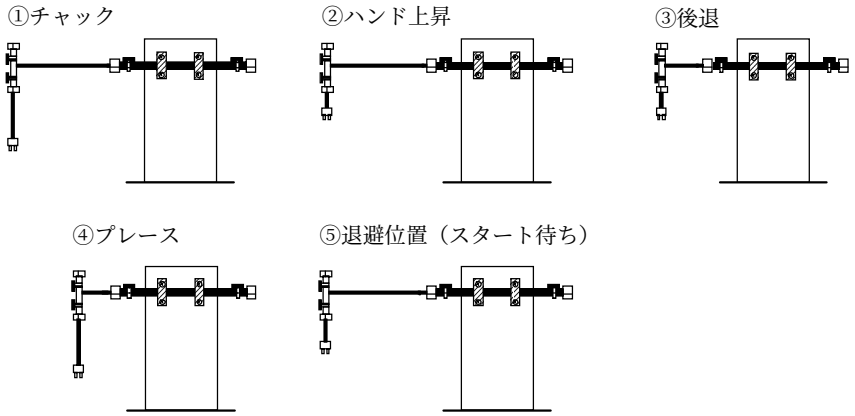


Ｉ／Ｏ割り付表

	出力		入力
0	シリンダ１	１９２	シリンダ１　前進
1	シリンダ２	１９３	シリンダ１　後退
2	チャック	１９４	シリンダ２　上端
3		１９５	シリンダ２　下端
4		１９６	スタートスイッチ

出力はポート０がＯＮで前進しポート１がＯＮで下降とします。センサはそれぞれのシリンダ終端を検出するものとします。

作業としては次の図のようなフローとします。パワーオン後の退避状態（ＯＦＦ０、ＯＦＦ１の状態）から前進下降しワークをつかみ、その後上昇し、後退し、さらに下降してワークを置きます。（プレース）そして上昇しさらに前進の上その状態でタイミング待ちとなります。

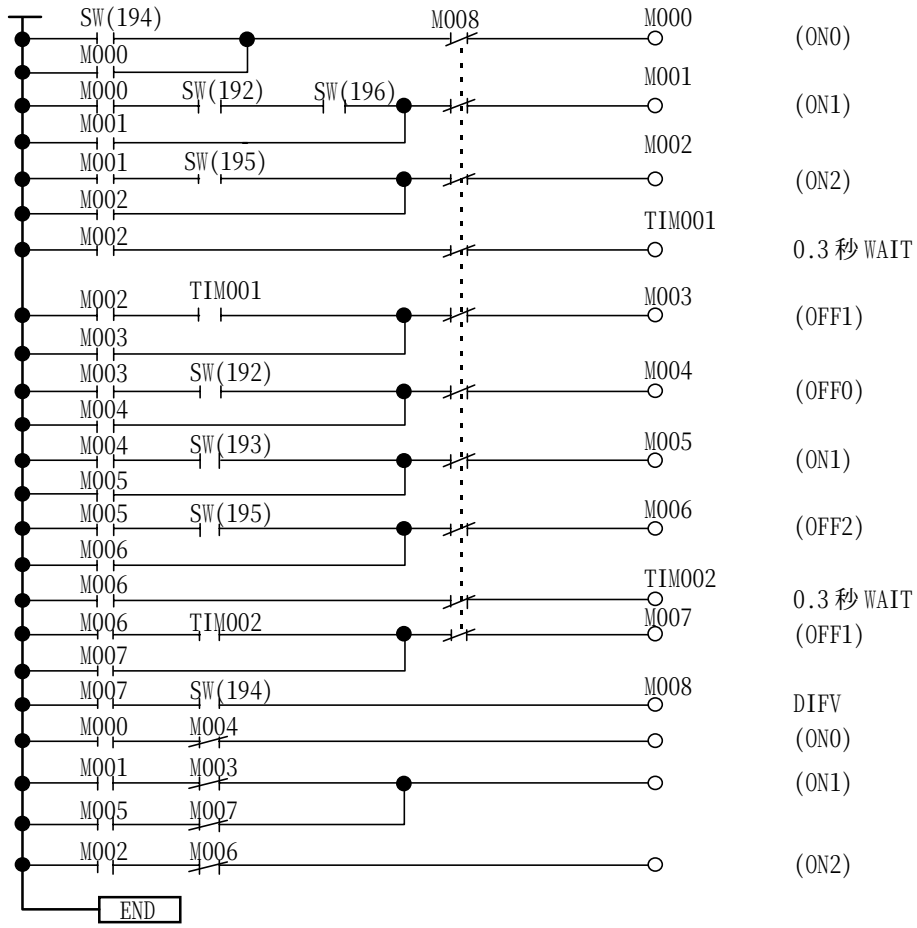


前記の動作をＡＤＶＦＳＣで表現すると次のようになります。
Ｐ＆Ｐサンプルプログラム

```
DO
  ON 0                          'ハンド前進
  WAIT SW(192)==1               '前進検出
  WAIT SW(196)==1               'スタート待ち
  ON 1                          'ハンド下降
  WAIT SW(195)==1               '下降検出
  ON 2                          'チャック
  TIME 300                      '０．３秒待ち
  OFF 1                         'ハンド上昇
  WAIT SW(194)==1               '上昇検出
  OFF 0                         'ハンド後退
  WAIT SW(193)==1               '後退検出
  ON1                           'ハンド下降
  WAIT SW(195)==1               '下降検出
  OFF 2                         'チャック解放
```

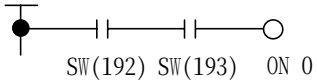
```
  TIME 300                    '0．３秒待ち
  OFF 1                       'ハンド上昇
  WAIT SW(194)==1             '上昇検出
LOOP
```

このようにＡＤＶＦＳＣでは、動作を時間を追って順に記述しますがシーケンサでは、全く異なった考え方でプログラムします。シーケンサのラダー図で同等の動作を表現すると次の図のようになります。



シーケンサでは、装置の動作を回路で実現するものとしてプログラムします。これは、２．１で述べたように自動機がリレー回路によって制御されており、シーケンサはこれを”ソフト化”するために開発されたものであることを意味しています。両者のプログラム方法の違いは根本的なもので、目的は同じ結果でも異質なものと言えます。
ここでは例として、Ｐ＆ＰをとりあげたためにＡＤＶＦＳＣの例が明解でラダー図が冗長なものとなっていますが、対象となる機器を替えると状況が変わることがあります。工場の現場で使用されている自動機の殆どは単純なプレイバックマシンですが、これらは簡単な論理から成立しています。こうした場合はラダー図の方がむしろシンプルです。例えば、コンベア制御の殆どが数個のセンサの論理から出力応答を得ます。
一定の大きさのワークが来たらストッパーをＯＮするというロジックは次の図のようになります。これに対してインタプリタでは次のようにやや難しそうな雰囲気を持つプログラムになります。

```
100  IF SW(192)==1 AND SW(193)==1 THEN
110  ON 0
120  END_IF
```



ラダー図とインタプリタのプログラミングを比較すると一般に次のようなことが言えます。
ラダー言語でのプログラムの特徴

①動作を回路で表現するために論理表現は素直であるが、Ｐ＆Ｐのように時間順序を持つ制御のプログラムは自己保持リレーのようなラダー独特の考え方を必要として冗長になりやすい。

②プログラムの基本概念に数値操作が含まれていないために数を扱う制御（パルス発生等）を直接扱うことができない。

③リレー回路によってプログラムを表現するので、回路そのもののデバッグはプロコンの発達に伴い非常に容易となっている。只プロコンは専用機であることが多く高価であることを免れない。

④リレー回路であることから装置の仕様変更がプログラムの全体に波及することが多い。このことは保守性の困難さにつながってくる。規模の大きなプログラムでは自分の書いたものですら読み返せない。

ＡＤＶＦＳＣの特徴

①動作をインタプリタで表現するため論理よりも物語のようにプログラムを記述するスタイルとなる。Ｐ＆Ｐや軸制御のように時間順序を持つ制御は素直に記述できる。

②プログラムの基本概念に数値制御が含まれており、パルス発生等のステップモータやサーボモータの制御を直接素直に扱うことができる。

③デバッグの方法が､リレー回路のように具体的でなく変数や実行文番号のモニターを介して間接的に行う｡このためプログラミングに対する基礎的な教養が身に付いていないと迷子になる。

④装置の仕様変更に対して柔軟性が高く部分の変更は全体に及ばない。又、プログラムの整理の仕方にもよるが一般に読み直し易い。これは、稼働後の変更に際して部分の変更が全体に及ばないことによる。

これらは、どちらが良いとか悪いのではなく性質の違いを表しています。制御すべき装置の動作がリレー回路で記述しやすいものであれば、ラダーは有効でＡＤＶＦＳＣは記述可能ということにとどまります。もし、ステップモータやサーボモータが無く多品種少量という目的が無ければラダー図で全ての装置を効率よく記述できます。只、最近の装置は多軸制御をベースとした多品種少量の傾向が強まっており、ＡＤＶＦＳＣが有用となる機会が増えているものと考えられます。

２．３　プログラムの基本テクニック

ＡＤＶＦＳＣではプログラムを効率よく記述するための方法が用意されています。

①Ｉ／Ｏのシンボル化

Ｉ／Ｏを番号のみで表現すると意味不明の場合があります。ＡＤＶＦＳＣではＣＯＮＳＴコマンドで定数を定義することによって入出力をラベルで表現することができます。

```
'     Ｐ＆Ｐサンプルプログラム
CONST SOL0 0                    'シリンダ１
CONST SOL1 1                    'シリンダ２
CONST HAND 2                    'チャック
CONST CL1F  192                 '前進端センサ
CONST CL1B  193                 '後進端センサ
CONST CL2U  194                 '上端センサ
CONST CL2D  195                 '下端センサ
CONST START 196                 'スタートスイッチ
DO
  ON SOL0                       'ハンド前進
  WAIT SW(CL1F)==1              '前進検出
  WAIT SW(START)==1             'スタート待ち
  ON SOL1                       'ハンド下降
  WAIT SW(CL2D)==1              '下降検出
  ON HAND                       'チャック
  TIME 300                      '０．３秒待ち
  OFF SOL1                      'ハンド上昇
  WAIT SW(CL2U)==1              '上昇検出
  OFF SOL0                      'ハンド後退
  WAIT SW(CL1B)==1              '後退検出
  ON SOL1                       'ハンド下降
  WAIT SW(CL2D)==1              '下降検出
  OFF HAND                      'チャック解放
  TIME 300                      '０．３秒待ち
  OFF SOL1                      'ハンド上昇
  WAIT SW(CL2U)==1              '上昇検出
LOOP
```

②マルチタスク

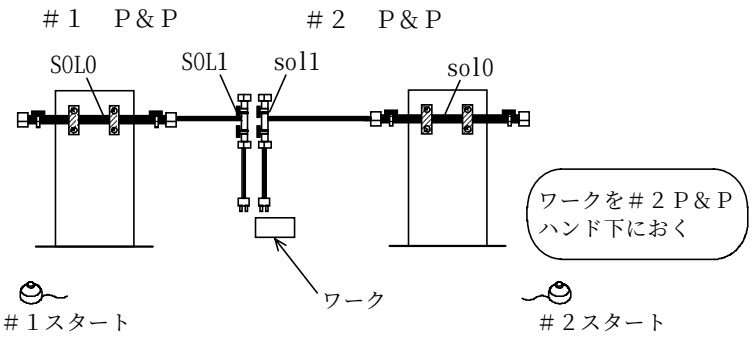
インタプリタではプログラムが時間順序で書かれるためにひとつの動作しか記述できません。しかし、実際の装置ではいくつもの要素が同時に進行するために複数のプログラムを動作させる必要があります。ＡＤＶＦＳＣではＦＯＲＫコマンドにより複数のプログラムを動作させることができます。例としては、先のＰ＆Ｐを２個同時にサポートするものを想定して次のサンプルプログラムを作りました。ここでは互いのＰ＆Ｐ＃１と＃２は全く別の装置として独立して動作します。尚、ＡＤＶＦＳＣのマルチタスクは２４迄です。

```
'     Ｐ＆Ｐサンプルプログラム＃１
 CONST SOL0 0                    'シリンダ１
 CONST SOL1 1                    'シリンダ２
 CONST HAND 2                    'チャック
 CONST CL1F 192                  '前進端センサ
 CONST CL1B 193                  '後進端センサ

 CONST CL2U  194                 '上端センサ
 CONST CL2D  195                 '下端センサ
 CONST START 196                 'スタートスイッチ
'
 CONST sol0  4                   'シリンダ１
 CONST sol1  5                   'シリンダ２
 CONST hand  6                   'チャック
 CONST cl1F  197                 '前進端センサ
 CONST cl1B  198                 '後進端センサ
 CONST cl2U  199                 '上端センサ
 CONST cl2D  200                 '下端センサ
 CONST start 201                 'スタートスイッチ
'
 FORK 1 *P&P#2
*P&P#1
 DO
   ON SOL0                       'ハンド前進
   WAIT SW(CL1F)==1              '前進検出
   WAIT SW(START)==1             'スタート待ち
   ON SOL1                       'ハンド下降
   WAIT SW(CL2D)==1              '下降検出
   ON HAND                       'チャック
   TIME 300                      '０．３秒待ち
   OFF SOL1                      'ハンド上昇
   WAIT SW(CL2U)==1              '上昇検出
   OFF SOL0                      'ハンド後退
   WAIT SW(CL1B)==1              '後退検出
   ON SOL1                       'ハンド下降
   WAIT SW(CL2D)==1              '下降検出
   OFF HAND                      'チャック解放
   TIME 300                      '０．３秒待ち
   OFF SOL1                      'ハンド上昇
   WAIT SW(CL2U)==1              '上昇検出
 LOOP
'     Ｐ＆Ｐサンプルプログラム＃２
*P&P#2
 DO
   ON sol0                       'ハンド前進
   WAIT SW(cl1F)==1              '前進検出
   WAIT SW(start)==1             'スタート待ち
   ON sol1                       'ハンド下降
   WAIT SW(cl2D)==1              '下降検出
   ON hand                       'チャック
   TIME 300                      '０．３秒待ち
   OFF sol1                      'ハンド上昇
   WAIT SW(cl2U)==1              '上昇検出
   OFF sol0                      'ハンド後退
   WAIT SW(cl1B)==1              '後退検出
   ON sol1                       'ハンド下降
   WAIT SW(cl2D)==1              '下降検出
   OFF hand                      'チャック解放
   TIME 300                      '０．３秒待ち
   OFF sol1                      'ハンド上昇
   WAIT SW(cl2U)==1              '上昇検出
 LOOP
'
```

③タスク間インタロック（セマフォ関数　ＲＳＶ（）、ＲＬＳ（））

複数のタスクによってサポートされた機器が一つの装置の上で動作する時には、干渉もしくは同期という問題があります。例えば、先の例でのＰ＆Ｐが次の図のように配置されており早くボタンを押されたＰ＆Ｐがワークをとるものとすると、インタロックが必要となります。このためにＡＤＶＦＳＣは、メモリＩ／Ｏとセマフォ関数が用意されています。セマフォとは、マルチタスクで広く用いられているインタロック用の函数で矛盾なくインタロックを実現します。ＲＳＶ（）、ＲＬＳ（）についてはコマンドリファレンスを参照して下さい。




```
'     Ｐ＆Ｐサンプルプログラム＃１
 CONST SOL0  0                  'シリンダ１
 CONST SOL1  1                  'シリンダ２
 CONST HAND  2                  'チャック
 CONST CL1F  192                '前進端センサ
 CONST CL1B  193                '後進端センサ
 CONST CL2U  194                '上端センサ
 CONST CL2D  195                '下端センサ
 CONST START 196                'スタートスイッチ
'
 CONST sol0  4                  'シリンダ１
 CONST sol1  5                  'シリンダ２
 CONST hand  6                  'チャック
 CONST cl1F  197                '前進端センサ
 CONST cl1B  198                '後進端センサ
 CONST cl2U  199                '上端センサ
 CONST cl2D  200                '下端センサ
 CONST start 201                'スタートスイッチ
'
 CONST memo1 -1                 'セマフォ用メモリＩ／Ｏ
 FORK 1 *P&P#2
*P&P#1
 DO
    WAIT SW(START)==1           'スタート待ち
    WAIT RSV(memo1)==0          'セマフォ獲得
    ON SOL0                     'ハンド前進
    WAIT SW(CL1F)==1            '前進検出
    ON SOL1                     'ハンド下降
    WAIT SW(CL2D)==1            '下降検出
    ON HAND                     'チャック
    TIME 300                    '０．３秒待ち
    OFF SOL1                    'ハンド上昇
    WAIT SW(CL2U)==1            '上昇検出
    OFF SOL0                    'ハンド後退
    WAIT SW(CL1B)==1            '後退検出
    ON SOL1                     'ハンド下降
    WAIT SW(CL2D)==1            '下降検出
    OFF HAND                    'チャック解放
    TIME 300                    '０．３秒待ち
    OFF SOL1                    'ハンド上昇
    WAIT SW(CL2U)==1            '上昇検出
      a=RLS(memo1)              'セマフォ解放
 LOOP
'
'     Ｐ＆Ｐサンプルプログラム＃２
*P&P#2
 DO
    WAIT SW(start)==1           'スタート待ち
    WAIT RSV(memo1)==0          'セマフォ獲得
    ON sol0                     'ハンド前進
    WAIT SW(cl1F)==1            '前進検出
    ON sol1                     'ハンド下降
    WAIT SW(cl2D)==1            '下降検出
    ON hand                     'チャック
    TIME 300                    '０．３秒待ち
    OFF sol1                    'ハンド上昇
    WAIT SW(cl2U)==1            '上昇検出
    OFF sol0                    'ハンド後退
    WAIT SW(cl1B)==1            '後退検出
    ON sol1                     'ハンド下降
    WAIT SW(cl2D)==1            '下降検出
    OFF hand                    'チャック解放
    TIME 300                    '０．３秒待ち
    OFF sol1                    'ハンド上昇
```

```
    WAIT SW(cl2U)==1              '上昇検出
      a=RLS(memo1)                'セマフォ解放
  LOOP
```

④非常停止・モード切り換え

装置制御の基本に一時停止やモード切り換えがあります。インタプリタでは順次制御の途中で入力を検出して動作を停止したり、変更することは本来難しいものです。しかし、マルチタスクを使用するとこれは比較的容易にできます。次に一時停止とモード切り換えの例を示します。モード切り換えはポート１９７のスイッチ一時停止は１９８としています。又、モード切り換えがオートからマニュアルに切り換わるところでは再スタートスイッチ１９９を通ってポートをクリアすることとしています。こうした後始末に相当するプログラムは装置によって大きく異なります。この例で重要なことは動作のプログラムをタスク＃１に移して主タスクでは制御スイッチの操作を監視し、動作タスク＃１をＦＯＲＫ、ＱＵＩＴ、ＣＯＮＴで制御することにあります。又、マニュアルモード＊ＭＡＮＵ～ＧＯＴＯ＊ＭＡＩＮまでのプログラムは入力ポートをパラレルで読み取って出力ポートにそのままパラレル出力しています。これはマニュアルモードでスイッチにより各出力を単動させる場合に有効なプログラムです。

```
'     Ｐ＆Ｐサンプルプログラム＃１
 CONST SOL0 0                     'シリンダ１
 CONST SOL1 1                     'シリンダ２
 CONST HAND 2                     'チャック
 CONST CL1F 192                   '前進端センサ
 CONST CL1B 193                   '後進端センサ
 CONST CL2U 194                   '上端センサ
 CONST CL2D 195                   '下端センサ
 CONST START 196                  'スタートスイッチ
 CONST MODE 197                   'モード・スイッチ
 CONST PAUSE 198                  '一時停止スイッチ
 CONST RSTART 199                 '再スタートスイッチ
'
*MAIN
 IF SW(MODE)==1 THEN              'オートマニュアル識別
   GOTO *AUTO
   ELSE
   GOTO *MANU
 END_IF
*AUTO
 FORK 1 *P&P#1                    'オート時制御
 DO WHILE SW(MODE)==1
   IF SW(PAUSE)==1 THEN
     PAUSE 1
     WAIT SW(PAUSE)==0
     CONT 1
   END_IF
   TIME 100
 LOOP
 QUIT 1
 WAIT SW(RSTART)==1
 OFF SOL0 SOL1 HAND
 GOTO *MAIN
*MANU
 DO WHILE SW(MODE)==0             'マニュアル時制御
     port=IN(25)
   TIME 100
   OUT port 0                     'パラレル入力パラレル出力
 LOOP
 GOTO *MAIN
*P&P#1
 DO
   WAIT SW(START)==1              'スタート待ち
   ON SOL0                        'ハンド前進
   WAIT SW(CL1F)==1               '前進検出
   ON SOL1                        'ハンド下降
   WAIT SW(CL2D)==1               '下降検出
   ON HAND                        'チャック
   TIME 300                       '０．３秒待ち
   OFF SOL1                       'ハンド上昇
   WAIT SW(CL2U)==1               '上昇検出
   OFF SOL0                       'ハンド後退
   WAIT SW(CL1B)==1               '後退検出
   ON SOL1                        'ハンド下降
   WAIT SW(CL2D)==1               '下降検出
   OFF HAND                       'チャック解放
   TIME 300                       '０．３秒待ち
   OFF SOL1                       'ハンド上昇
```

```
    WAIT SW(CL2U)==1                ’上昇検出
  LOOP
```

⑤制御文

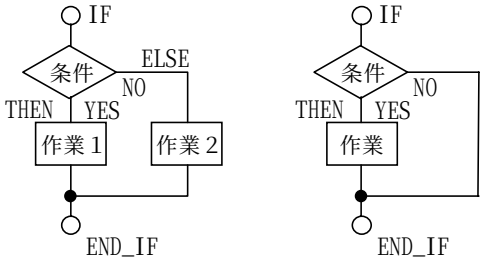
さて、ここまでの例題でも様々な制御文が例示されてきましたが、ＡＤＶＦＳＣでは次の制御文が用意されています。

・ＩＦ～ＴＨＥＮ～ＥＬＳＥ～ＥＮＤ＿ＩＦ
・ＩＦ～ＴＨＥＮ～ＥＮＤ＿ＩＦ
・ＤＯ～ＬＯＯＰ
・ＦＯＲ～ＮＥＸＴ
・ＳＥＬＥＣＴ～ＣＡＳＥ～ＣＡＳＥ＿ＥＬＳＥ～ＳＥＬＥＣＴ＿ＥＮＤ
・ＧＯＴＯ～
・ＧＯＳＵＢ～ＲＥＴＵＲＮ
・ＷＨＩＬＥ～ＷＥＮＤ
・ＷＡＩＴ～ＷＳ０（）、ＷＳ１（）

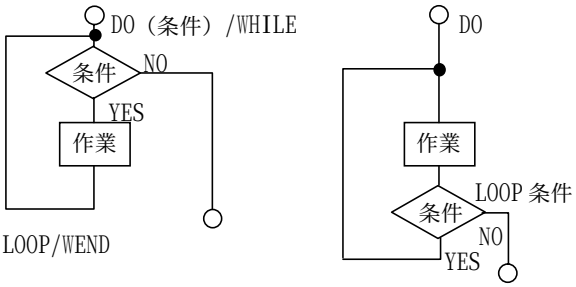
これらはいずれも求める条件判断や繰り返し作業にマッチした使い方をされるべきです。ここではそれぞれの制御文の意味を概説しておきます。詳細はコマンドリファレンスを参照して下さい。

・ＩＦ～ＴＨＥＮ～ＥＬＳＥ～ＥＮＤ＿ＩＦ
　ひとつの条件から異なる動作を必要とする場合に有効です。

・ＩＦ～ＴＨＥＮ～ＥＮＤ＿ＩＦ
　ある条件が成立した時にある動作する場合に用います。



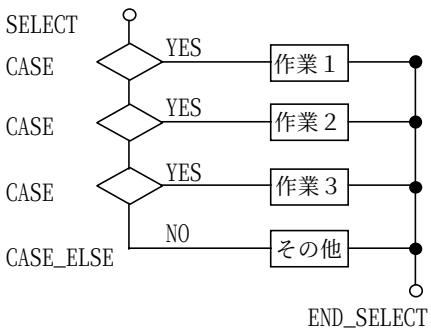
・ＤＯ～ＬＯＯＰ、ＷＨＩＬＥ～ＷＥＮＤ
　繰り返し動作で条件を付加することができます。ＤＯもしくはＬＯＯＰのあとにＷＨＩＬＥあるいはＵＮＴＩＬという条件を与えることができます。ＷＨＩＬＥは条件が成立している間ＵＮＴＩＬは条件成立するまでという意味です。



・ＦＯＲ～ＮＥＸＴ
　数の定まった繰り返し作業です。１００回などという場合は次のように記述します。又、ＳＴＥＰを与えることによって数の増加を１だけでなくいろいろ数（負まで含めて）を与えることができます。

```
FOR    I=1 TO 100
          作業
NEXT I
```

・ＳＥＬＥＣＴ～ＣＡＳＥ～ＣＡＳＥ＿ＥＬＳＥ～ＥＮＤ＿ＳＥＬＥＣＴ
　ひとつの入力によっていくつもの場合分けを必要とする場合に有効です。ＤＳＷ（デジスイッチ）によって得た数にしたがって異なるプログラムを動作させる等が例となります。



・ＧＯＴＯ～　、ＧＯＳＵＢ～ＲＥＴＵＲＮ

ＧＯＴＯ～は単純にラベルで示された場所へ制御を移すことです。ＧＯＳＵＢはサブルーチンコールです。ＧＯＳＵＢで呼ばれたモジュールはＲＥＴＵＲＮ文でＧＯＳＵＢの下に戻ります。

・ＷＡＩＴ～、ＷＳ０（）、ＷＳ１（）

ＷＡＩＴは最も単純な条件待ちです。条件が成立するまでＷＡＩＴ文から抜け出せません。これを関数化したものがＷＳ０（）、ＷＳ１（）です。これにはタイムアウトが付加できます。

⑥ＢＡＳＩＣと比較して

ＡＤＶＦＳＣは、ＢＡＳＩＣと良く似た言語です。これによりプログラムを親しみ易いものにしてユーザーの負担を軽くしています。只、装置制御を対象とした組み込み用ボードコンピュータという制限からその仕様は様々な点で簡略化されています。例えば、文字列変数は３２個までに限定していますし、配列宣言も１５までで一次元に限っています。関数は引き数を１つに限定しているため本来２つの引き数を必要とする関数もＡＤＶＦＳＣに限った特別なラベルとなっています。（ＩＮＰ＄＃０（）、ＩＮＰ＄＃２（））もちろんＡＤＶＦＳＣは装置制御を広くサポートするように設計されていますので、Ｉ／O制御パルス発生等のアプリケーションには柔軟に効率良く対応します。ＡＤＶＦＳＣ／ＭＰＣ－６８４が有用となるアプリケーションは主に次のようなものです。

①パルス発生で数値制御を含むもの
②通信や演算を含む制御
③専用のプロコンを用いず汎用パソコンでプログラミングを必要とする場合
④Ａ／Ｄ、Ｄ／Ａ等の汎用パソコンの周辺を用いる応用

又、ＢＡＳＩＣと大きく異なっているのは次のような事柄です。

①扱える変数が４ byte 長整数に限られていること
②配列は一次元のみで１５ラベルまで、又総数１００００に限られていること
③文字列変数は３２個までで長さが８０文字以内
④一部関数が引き数１ヶの為変形されていることＩＮＰ＄＃０（）、ＩＮＰ＄＃２（）等