内部タイマー８２５３のベースアドレス（最下位アドレス）は７１Ｈとなっており、これがカウンタ＃０へデータを送るアドレスである。カウンタ＃１は７３Ｈ、＃２は７５Ｈであり、各カウンタのモードを決めるためのデータを送るアドレスは７７Ｈとなっている。内部割り込み用のカウンタは「＃０」である。カウンタを動作させる前にまず、動作モードを決めなければならない。そのデータは８ビットで構成され表に示した形式に従って作成する。そのデータは次のようにする。

```
COUNTER SELECT       :COUNTER #0
READ/WRITE MODES     :LSB,MSB SEQUENCIAL R/W
OPERATIONAL MODES    :MODE 0
COUNT MODES          :BINARY
```

そうすると、８ビットデータは

００１１０００００Ｂ＝３０Ｈ

となる。これを「ＯＵＴ」命令によって「７７Ｈ」番地へ送ればカウンタ＃０が指定通りにセットされる。ここで、「R/W MODES 」の設定は、１６ビットのカウントデータを上位および下位８ビットに分け、まず下位８ビットを書き込んで、次に上位８ビットを書き込むということである。これは８２５３のデータバスが８ビットとなっているためである。従って、カウンタ＃０にデータを送るプログラムは

```
OUT  71H,AL
MOV  AL,AH
OUT  71H,AL      (AX:COUNT DATA)
```

となる。カウンタは２番目の「ＯＵＴ」命令によってデータを受け取った時点からカウントを開始し、指定カウントが終わると割り込み信号を発生するということになる。

ここで、カウンタのモード０という動作を簡単に説明する。カウンタの出力信号は動作していない時では、「１」となっている（ただし、モード設定が終了した時点で「０」となっている）。カウンタはデータを受け取った時点からカウントを開始するが、カウント中は出力が「０」となり、カウント終了すると「１」となる。従って、カウント値が大きいほど「０」の期間が長くなる。

このＰＩＴは、これからみんなが作成するプログラムの中心となるものである。すなわち、ある幅のパルスを作成するために使用しているものであり、最終的にＰＷＭパターンを作る。プログラムする最終目的の一つはこのカウンタのカウント値を求めることである。従って、このＰＩＴの動作を十分理解しておかなければならない。このような、パルスを作成するＰＩＴは外部Ｉ／Ｏとして、パソコンの拡張スロットに組み込んである。内部ＰＩＴは８ビットデコードであるのに対し、パルス発生用外部ＰＩＴは１６ビットデコードとしているので、Ｉ／ＯアドレスはＩ６ビットであるが、モード設定やデータ書き込みなどのプログラミングの基本は変わらない。現在、４個の外部ＰＩＴ（計１２個のカウンタ）があり、それぞれのベースアドレスは 70D0H, 70D1H, 80D0H, 80D1H としている。

２５．「バッチファイルは便利がいいよ」

さて、話をＤＯＳに移そう。Ａ君はかなり興味が涌いてきて自分で色々ＣやＭＡＳＭのプログラムを作成し、実行していたが、どうも面倒くさい。それで、Ｄ君に質問した。そこにはＢ君、Ｃ君もいた。３人はいつも一緒にいたのだ。

Ａ君：「プログラミングが結構おもしろいんで、色々作ってみようんやけど、コンパイルとかリンクする作業がどうも面倒くさいっちゃ。なんかいい方法はないとかいな？」

Ｂ君：「すげー。おまえの口からそんな言葉がでるとは思わんかった」（あいつ、

何か知らん間に勉強しとるな、なんて思った）
Ｄ君：「そういうためにバッチファイルっちいうのがあるったい」
Ｃ君：「そうそう」（Ｃ君は知っていた）
Ａ君：「バッチファイル？　なんか、だいぶ前に聞いたことがあるぜ」
Ｄ君：「コンパイルとかリンクとかの１連の操作を書いたファイルたい。これを知ってたら便利いいよ。そしたらバッチファイルを作ってみるけん、コンパイルとリンクの手順を思いだしながら、よーーと聞きよきーよ」
と言ってＤ君は説明を始めた。
「まず、バッチファイルを作るために、 MIFESを開くやろ。バッチファイルの名前は何にしようかな。何でもいいけど、「ＣＯＬＩ」にしよう。ドライブＢに作ることにして、 MIFESで
　B:COLI.BAT
のファイルをオープンする。バッチファイルの拡張子は「ＢＡＴ」と決まっとうけんね。そして、次のように書くったい。

```
D:
MSC B:%1,B:;
MASM B:%2,B:;
LINK B:%1+B:%2;
A:
```

まず、カレントドライブをＤとするのに、「D:」としたやろ。だけん、それを１行目に書く。次に「Ｃコンパイラ」を起動するのに、「ＭＳＣ」と入力したやろ。だけん、それを２行目に書くけど、この時、ファイルが何かとか色々聞いてきたことを、そのまま続けて「,（カンマ）」で区切って書いとくわけたい。すなわち、「Ｂ：ファイル名」とオブジェクトファイルの保存箇所「Ｂ：」、後はソースリストとオブジェクトリストやけど、これは何も入力しないで、リターンキーだけ押したので、省略するということで、最後は「；」をつける。これで、ＣコンパイルをやってオブジェクトファイルをドライブＢに作ってくれる。ただ、どのファイルでもいいように、バッチファイル名を入力するときに、一緒に入力したファイルをコンパイルできるように、「引き数」にしておくわけたい。これが「％１」なのだ。すなわち、バッチファイル入力時の第１パラメータとなる。次に、「ＭＡＳＭコンパイラ」も同じように考えると３行目のようになるやろ。この時、一応、バッチファイル入力時の第２パラメータを作っとこうね。それが「％２」たい。そして、「リンカ」では２つのファイルを入力したので、４行目のように％１と％２を使って書いておくわけ。そして、最後に、カレントドライブを「Ａ」に戻す、こういうことになるわけたい。これを見てわかるように、実際に入力することを、ただ、そのまま書き並べていったというだけなんよ。これを保存して、このバッチファイルを実行させるには、例えば、まえに作った加算プログラム（「Ｃ」の「SUM.C 」と「ＭＡＳＭ」の「SUMA.ASM」）の場合には
　A:¥>B:COLI SUM SUMA
のように入力するだけでいい。そうすると、バッチファイルの中では「%1」の所がそのまま「SUM 」、「%2」の所が「SUMA」に置き変わって実行されるというわけたい。どう？　簡単やろ」
Ａ君：「そうね。簡単やね。これやったら楽じゃん」
Ｂ君もわかったのでうれしかった。
そこで、Ｃ君が付け加えて言った。
「でも、僕たちが普通作るプログラムの名前は「Ｃ」のファイル名に「アルファベットＡ」をくっつけたものを「ＭＡＳＭ」のファイル名としているので、バッチフ

ァイルは

```
D:
MSC B:%1,B:;
MASM B:%1A,B:;
LINK B:%1+B:%1A;
A:
```

とすれば、
A:¥>B:COLI SUM
となるったい」
A君：「そうか。入力は、これの方がまだ簡単やん」
D君：「そうだね。だから、僕たちはこんな風に名前を付けてるわけですよ」
A君もB君も十分納得したので気分がよかった。

バッチファイルはこんなものかというのかわかれば簡単である。しかし、上のバッチファイルはやっかいな点が1つある。それは、このバッチファイルは常に「C」も「MASM」のソースプログラムをコンパイルを行うということである。実際、「MASM」だけのプログラムを変更した場合には、「C」のオブジェクトファイルはドライブBに保存されているので、「MASM」だけをコンパイルしてリンクすればよい（別に、いつも両方コンパイルしても問題はないが、時間の無駄である）。また、「C」だけの単体のプログラムだけの「EXE」ファイルを作成する（前のTEST.Cなど）場合には利用できない。このような場合でも利用できるバッチファイルを作成しているので、これからはそれを利用することにしよう。バッチファイル名は「VX」であり、第1パラメータはファイル名、第2パラメータに何を行うかの引き数を入力する。第2パラメータの意味は

CC	C単体コンパイル、リンク
AC（またはCA）	CとMASM共にコンパイル、リンク
C	Cのみコンパイルして、リンク
A	MASMのみコンパイルして、リンク
D2	DSP（C25）コンパイル、リンク
AD2（またはDA2）	MASMのみコンパイルおよび DSP（C25）コンパイル、リンク

である。例えば、前に作成した加算プログラムのMASMだけをコンパイルするには次のように入力する。
A:¥>VX SUM A
ただし、このバッチファイルを利用するには、「C」および「MASM」ソースファイルを決められた「ディレクトリ」に保存しておく必要がある。これについては次で説明しよう。

26．「ファイルはきちんと整理しよう」

これからみんなが作成するプログラムには、既に作成した「C」、「MASM」があり、コンパイルによって作られる「オブジェクトファイル(*.OBJ)」および「EXEファイル」もある。今の所、これらのファイルはすべてドライブBのディスクに保存されているであろう。さらに、「DSPソースファイル」あるいは「BASICファイル」も作成することになり、こうなると、ドライブBのディスクには様々なファイルが保存される。このようなファイルの保存状態では、いざ、目的の

ファイルを探したり、どのようなファイルがあるかを調べたりする時には、わかりにくくなる。こういった場合を考えて、普通、ファイルを「ディレクトリに分類する」ことを行う。すなわち、「Cのファイル」とか「ＭＡＳＭのファイル」とかに分類して、それぞれに名前（ディレクトリ名）を付けてファイルを整理するというわけである。前に示した「ＶＸバッチファイル」はすべて、ドライブＢのディレクトリに保存されたファイルを対象にして処理を行うようにしている。

それでは、実際にファイルの分類を行おう。このテキストに従って勉強していれば、ドライブＢのディスクには次のファイルが作成されている。（DIR B:を入力して調べてみよう）

```
COMMAND.COM
TEST.C, TEST.OBJ, TEST.EXE
NUM.C, NUM.OBJ, NUMA.ASM, NUMA.OBJ, NUM.EXE
SUM.C, SUM.OBJ, SUMA.ASM, SUMA.OBJ, SUM.EXE
```

とりあえず、ソースファイルだけが必要なので、他のファイルを除去するために、カレントドライブを「Ｂ」として、次のコマンドを入力する。

```
B:¥>DEL *.OBJ
```

これは内部コマンド「DELETE（削除）」であり、その次に入力されたファイルを削除する。ファイル名は通常「ファイル名．拡張子」であるが、上のように入力すると、「拡張子が（．ＯＢＪ）のファイルをすべて削除」ということになる。この記号「＊」を「ワイルドキャラクタ」といい、「＊」を含めた記入方法を「ワイルドカード」という。「ワイルドキャラクタ」には他に「？」がある（これについては興味があれば他のマニュアルで調べること）。だたし、このような「ワイルドカード」を使用する場合には、関係したすべてのファイルに対して処理されるので、注意が必要である。

これで、オブジェクトファイルがすべて削除される。さらに、

```
B:¥>DEL *.EXE
```

を入力して「ＥＸＥファイル」を削除する。ドライブＢのディスクにはCOMMAD.COMファイルとソースファイルだけが残っていることを確認しよう。

次に、ファイルを分類する「ディレクトリ」を作成するために、

```
B:¥>MD AFILE
```

を入力する。これは内部コマンド「MAKE DIRECTORY」である。ＭＡＳＭファイルはすべてディレクトリ名「AFILE 」の中に保存する（ディレクトリの削除は内部コマンド「RD (REMOVE DIRECTRY)」で行う）。これで、名前「AFILE 」のディレクトリが作成されるので、ドライブＢのディスクの中に「AFILE 〈DIR〉」が存在することを確認しよう。ディレクトリは通常のファイルと異なり、「〈DIR〉 」で表示される通常の「DIR 」コマンドで見れるドライブの内容を保存している所を「親ディレクトリ」と言い、「〈DIR〉 」を「子ディレクトリ」と言う。

これでＭＡＳＭ用ディレクトリを作成したので、その中にＭＡＳＭソースファイルをコピーするために、次のように入力する。

```
B:¥>COPY *.ASM AFILE¥*.*
```

これは、内部コマンド「COPY FILE1 FILE2」であり、「FILE1 」のファイル（拡張子まで書く）を「FILE2 」のファイル名でコピーする。上の記入法は「ワイルドカード」による書き方である。子ディレクトリを指定するときには、ディレクトリ名とファイル名の間に「句切り記号¥」を入れる。上のコマンドにより、親ディレクトリにある拡張子「．ＡＳＭ」のファイルはすべて子ディレクトリ「AFILE 」の中に同じ名前で保存される。これは、次のように入力しても同じである。

```
B:¥>COPY B:¥*.ASM B:¥AFILE¥*.*
```

すなわち、カレントドライブが「Ｂ」となっている時には、その「親ディレクトリ名Ｂ：¥」は省略可能である。また、例えば、１つのファイルだけを同じファイル

名でコピーする時には
　B:¥>COPY FILE1.ASM AFILE¥
と書け、コピー先のファイル名は省略できる（ワイルドカード使用時は省略できない）。ここで、ディレクトリ「AFILE 」の内容を見るために次のように「DIR 」コマンドを入力してみよう。
　B:¥>DIR AFILE
ソースファイルがあることを確認する。このなかで、「. <DIR> 」と「.. <DIR>」はディレクトリを管理するためのファイルである（あまり、気にしなくてよい）。子ディレクトリ「AFILE 」の中にきちんとＭＡＳＭファイルがコピーされていれば、親ディレクトリのＭＡＳＭファイルは削除しておこう。
　次に、同様に子ディレクトリ「CFILE 」を作成し、その中にＣのソースファイルをコピーし、親ディレクトリのファイルを削除する。これで、ドライブＢの中には次のファイルが存在する。
　COMMAND.COM
　AFILE <DIR>, CFILE <DIR>
さらに、これから他のプログラムも作成するので、次の子ディレクトリも作成しておこう。
　BFILE, DFILE, COBJE, BINARY
ここで、これらの子ディレクトリの「BFILE 」はＢＡＳＩＣソースプログラム用、「DFILE 」はＤＳＰソースプログラムおよびオブジェクトファイル用、「COBJE 」はＣとＭＡＳＭのオブジェクトファイル用、「BINARY」はＢＡＳＩＣで作成する色々なバイナリーデータ用である。
　これで、バッチファイル「ＶＸ」を利用するための準備ができたので、試しにやってみよう。

　ここで、もう一度エディタ「MIFES 」（バッチファイル名「EDIT」）について説明する。だいぶん前の所で、ソースプログラムを作成するために「MIFES 」を起動したが、今度は、それぞれのソースプログラムが子ディレクトリの中にあるので、少しファイルオープンの方法が違ってくる。バッチファイルでは「MIFES 」起動時にオプションとして子ディレクトリ名だけを表示するようにしているので、起動メニューでは、ドライブＢの子ディレクトリ名が表示される。この中から開きたい子ディレクトリを
　「カーソル移動キー」で選択してリターンキーを押すと
そのディレクトリ名が表示されるので、もう一度「リターンキー」を押せば、その子ディレクトリの内容が表示される。そこで、また、「カーソル移動キー」で開きたいファイルを選択して、リターンキーを押せば、そのファイル名が表示されるので、もう一度「リターンキー」を押せば、そのファイルがオープンされる。すなわち、ファイル名をキーボードから入力しなくても、選択してリターンキーを押すだけで、必要なファイルがオープンできる。終了時は前と同じである。

２７．「ＢＡＳＩＣの基本操作について」

　ＢＡＳＩＣ言語については、知ってる人もかなり多いと思う。言語自体は簡単であり、ハードの知識をほとんど必要としないので、ここでは、簡単に説明する。卒論で使用するＢＡＳＩＣはアセンブラあるいはこの先で勉強するＤＳＰプログラムにおけるデータテーブル（簡単に言えば機械語によって何等かの計算をするためのデータ）を作成するために利用している。命令については「ＢＡＳＩＣリファレンスマニュアル」を参照して勉強してもらうとして、ここでは、「ＢＡＳＩＣ」のた

ち上げとファイルの「SAVE」、「LOAD」について述べる。
ここで使用する「ＢＡＳＩＣ」は「Ｎ８８－日本語ＢＡＳＩＣ（８６）Ｖｅｒ．６．０（ＭＳ－ＤＯＳ版）」である。すなわち、ＭＳ－ＤＯＳ上で動作する「ＢＡＳＩＣ」である。ＤＯＳからこのＢＡＳＩＣを起動するには、そのためのバッチファイルを作成しているので、それを利用する。カレントドライブをＡとして

A:¥>N88

を入力すれば、Ｎ８８ＢＡＳＩＣが起動されて、通常のＢＡＳＩＣと同じテキスト画面となる。あとは、プログラミングすればよい。ただし、ＢＡＳＩＣの場合は行番号が必要である（こんなことは知ってると思うけど）。
作成したＢＡＳＩＣプログラムはドライブＢの子ディレクトリ「BFILE 」に保存するので、「SAVE」する場合は次のように指定する（例として、ファイル名FNAMEの場合）

SAVE "B:¥BFILE¥FNAME.BAS",A

ＭＳ－ＤＯＳ版Ｎ８８ＢＡＳＩＣの場合、ファイル名の拡張子は「ＢＡＳ」と決まっている。また、「Ａオプション」はアスキー形式セーブ（「Ａオプション」指定を省略した場合はバイナリ形式セーブとなる）であり、このセーブ形式のファイルは「MIFES 」のエディタでも編集できる（ただし、エディタでの編集時には、行番号に注意が必要である）。また、ファイルを「LOAD」する場合も同様に

LOAD "B:¥BFILE¥FNAME.BAS" または　LOAD "B:BFILE¥FNAME

と指定する。そのほか、ディスク内容を見る場合には

FILES "B:¥BFILE

のように指定する。
Ｎ８８ＢＡＳＩＣからＤＯＳシステムへ戻るには

SYSTEM

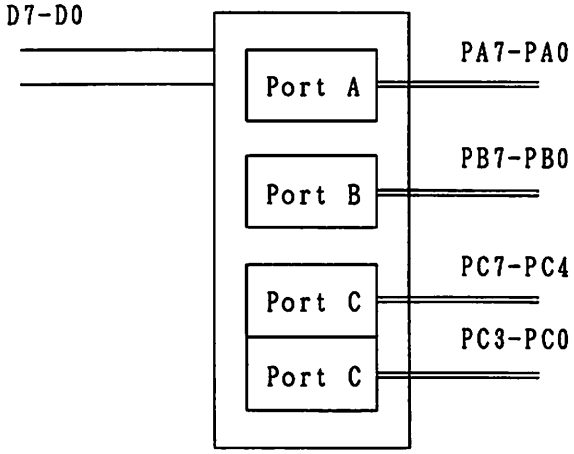
を入力すればよい。

２８．「外部入出力Ｉ／Ｏ（８２５５）」

コンピュータ外部とＴＴＬレベルでのデータのやり取りを行うものである。コンピュータ内部のデータは当然ディジタル（８ビットあるいは１６ビットＴＴＬレベル）であるが、コンピュータ外部のＴＴＬで構成した回路に対してデータを出力したり、その回路からデータを取り込んだりするためのものであり、通常「ＰＰＩ(Programable Peripheral Interface)」と呼ばれるＬＳＩである。。８２５５は８ビットコンピュータシステムのために設計された汎用プログラマブルＩ／Ｏである。８２５５はポートＡ、ポートＢ、ポートＣのそれぞれ８ビットＩ／Ｏ（全部で２４ビット）をもち、各ポートは独立にプログラムできる。また、ポートＣはビット単位でプログラム可能である。次の図は８２５５の基本構成図である。
実際に使用しているＰＰＩは「コンテックＰＩＯ－４８Ｗ（９８）」のモジュールであり、これには８２５５が２個実装されており、１６ビット単位での入出力ができる。前に説明した８２５３ＰＩＴと同様に、このＰＰＩも３つの動作モードがあるので、使用する前にモード設定を行う。まず、各ポートのモードを決めるが、ポートＡとＢは８ビット、ポートＣは上位と下位４ビットに分けて行う（ポートＣは制御信号の出力あるいはステータス情報の入力として用いるために分けてある）図に従うと、例えば、すべてモード０（基本入出力モードであり、通常使用しているのはこの動作モードである）として、ポートＡとポートＣ（下位）を出力、ポートＢとポートＣ（上位）を入力に設定するときのコントロールデータは

１０００１０１０Ｂ＝８ＡＨ

となる。このデータをコントロールデータの番地へ書き込む。



Ｉ／Ｏアドレス
（１６ビットデコード）
BASE ADDRESS = 20D0H

20D0H	PA(L) DATA R/W
20D0H+1	PA(H) DATA R/W
20D0H+2	PB(L) DATA R/W
20D0H+3	PB(H) DATA R/W
20D0H+4	PC(L) DATA R/W
20D0H+5	PC(H) DATA R/W
20D0H+6	8255(L) CONTROL W
20D0H+7	8255(H) CONTROL W

(L:Lower, U:Upper)

CONTROL DATA FORAMT

MODE SELECT

1	D6	D5	D4	D3	D2	D1	D0

D6, D5	PA, PC(U) MODE SEL.	00 01 1X	MODE 0 MODE 1 MODE 2
D4	PA I/O	0 1	OUTPUT INPUT
D3	PC(U) I/O	0 1	OUTPUT INPUT
D2	PB, PC(L) MODE SEL.	0 1	MODE 0 MODE 1
D1	PB I/O	0 1	OUTPUT INPUT
D0	PC(L) I/O	0 1	OUTPUT INPUT

BIT CONTROL (PORT C)

0	X	X	X	D3	D2	D1	D0

D3, D2, D1: BIT SELSCT
D0: BIT SET/RESET

PORT	D3	D2	D1
PC0	0	0	0
PC1	0	0	1
PC2	0	1	0
PC3	0	1	1
PC4	1	0	0
PC5	1	0	1
PC6	1	1	0
PC7	1	1	1

SET=1
RESET=0

すなわち、前述したように８２５５が２つ（それぞれ上位と下位と呼ぶ）あるので、例えば下位の８２５５をセッティングするには、

```
OUT  DX, AL     (DX=20D6H, AL=8AH)
```

となる。下位の８２５５も同じモードで設定するには同じデータを 20D7H番地へ書き込めばよい。次に、ポートＡ（下位８２５５）からデータを出力するには、

```
OUT  DX, AL     (DX=20D0H)
```

を実行すれば、ＡＬのデータが出力される。また、ポートＢ（下位８２５５）から

データを取り込むには

```
IN   AL,DX     (DX=20D2H)
```

を実行すれば、ＡＬにデータが入るということである。

ここで、８２５５が２個使用される理由は、１度に１６ビットデータの入出力を行うためである。例えば、ＡＸのデータを出力するには、１個の８２５５ではポートＡとポートＢに２回に分けて８ビットづつ出力しなければならないが、２個の８２５５を使用して、しかも連続したＩ／Ｏアドレスに設定してあれば、

```
OUT  DX,AX     (DX=20D0H)
```

を実行することによって、ＡＸの下位データは下位８２５５のポートＡから、ＡＸの上位データは上位８２５５のポートＡから出力され、しかも同時に出力できるわけである。入力の場合も同様である。

ポートＣはビット単位の出力（ビット単位の入力はできない）が可能である。例えば、下位８２５５のＰＣ３を「High (1)」とするには

```
OUT  DX,AL     (DX=20D6H, AL=00000111B)
```

「Low(0)」とするには

```
AL=00000110B
```

のデータを出力する。なお、ポートＣのビット出力の場合のＩ／Ｏアドレスはコントロールアドレスである。

ベースアドレスは実際にプログラミングする時に確認する必要がある。

２９．「アナログデータを取り込もう（Ａ／Ｄコンバータ）」

みんなが最終的に行う制御はコンバータやインバータなどであり、それらの電圧や電流をコンピュータに取り込んで、その情報をもとに制御する。例えば、電流が一定になるように制御するには、実際に流れている電流が幾らであるかを調べなければならない。この時、電圧や電流は連続的なデータ、すなわちアナログ量であり、しかも、これらの量は正の値と負の値をもち、さらに、電圧の場合には200Vとか300Vのように高い電圧である。電流の場合には、検出用抵抗を回路に接続してその抵抗の電圧降下を調べる（例えば、測定抵抗が0.05Ωの時、その両端の電圧が0.5Vであれば、流れている電流が 10Aとわかる。大きな値の抵抗を入れると回路の動作特性が変わるので、できるだけ小さいものを接続する）。このような、アナログ量のデータを２進数で表した値に変換するものが「Ａ／Ｄ(Analog/Digital)コンバータ」である。実験で使用しているものは「コンテックＡＤ１２－１６Ａ（９８）」のモジュールであり、これには１６個のチャンネル（CH0-CH15）のＡ／Ｄコンバータが入っている。このコンバータのアナログ入力は最大±１０Ｖであり、分解能は１２ビットである。シングルエンドバイポーラ（２極性）入力としており、変換速度は約１２μｓである。

電力変換回路とＡ／Ｄコンバータのアナログ入力は絶縁する必要があるので、実際の検出はＬＥＭモジュールで行っている。電圧検出は400Vの時にアナログ量 10V（-400Vの時は-10V ）であり、電流検出は 20Aの時にアナログ量 10V（-20Aの時は-10V）となるように構成している。

それでは、実際のプログラミングについて簡単に説明しよう。Ａ／ＤコンバータのＩ／Ｏアドレスを

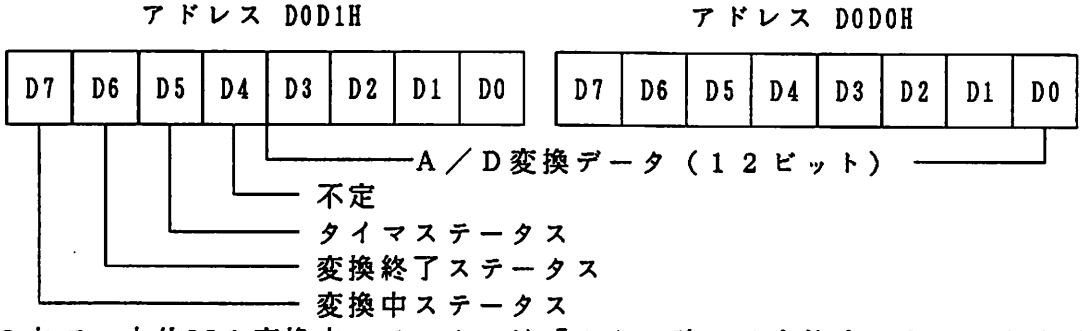
```
ＤＯＤＯＨ
```

とすると（実際は D0D1Hも占有するが、このアドレスデータはＡ／Ｄ用内蔵タイマーであり、使用していない）、次のような手順となる。まず、Ａ／Ｄ変換をスタートさせるために次のフォーマットに従ったデータを出力する。

D7	D6	D5	D4	D3	D2	D1	D0

D7: タイマー用ゲート（タイマーは使用しないので、D7='0'）
D6=0 : D5=X
D4: 変換スタート時は１（０の時はチャンネル切り替えのみを行う）
D3-D0: チャンネル選択（0000時はCH0, 1111の時はCH15など）

例えば、チャンネル１のＡ／Ｄ変換をスタートさせるには
　　OUT　DX,AL　　(DX=D0D0H, AL=00010001H)
を実行すればよい。次に、変換したデータを取り込むが、Ａ／Ｄ変換には前述したように変換するための時間（変換時間が短いＡ／Ｄほど高価である）が必要であり、すぐにデータを取り込んでも意味がない。そのために、変換が終了したかどうかの信号（ステータス信号）があり、これをチェックしてデータを取り込む。取り込む場合のデータフォーマットは次のようである。



この中で、上位D7の変換中ステータスは「１」の時には変換中であることを示す。従って、このビットが「０」であることを確認して１６ビットデータを読み込む。よって、
　　IN　AX,DX　　(DX=D0D0H)
を実行し、ＡＸのＭＳＢが「０」かどうかを調べて、「０」であればもう一度上の命令を実行して、データを取り込む。この時、フォーマットにあるように、データは１６ビットの下位１２ビットである。このデータとアナログ量の関係は次のようになっている（１２ビットによる正負表現）。

ｱﾅﾛｸﾞ電圧	入力データ D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
+10-1LSB 9.99512V	0	1	1	1	1	1	1	1	1	1	1	1	7FFH
0 V	0	0	0	0	0	0	0	0	0	0	0	0	000H
-10V	1	0	0	0	0	0	0	0	0	0	0	0	800H

取り込んだ１６ビットの内、上位４ビットは実際のデータとは関係ない。よって、取り込んだ１６ビットのデータは、上位４ビットを例えば「０」としても、１６ビットでは正負表現されないことに注意しないといけない（１６ビットの正負表現では D15の値が正であるか負であるかを決める）。したがって、この１２ビット正負データを１６ビット正負データに変換しなければならないが、どうしたらいいかは各自考えてみよう。１６ビットデータが得られれば、あとはそのデータどのように

処理するかということになる。その時には、実際の電圧あるいは電流の値との対応を十分に考慮しておく必要がある。

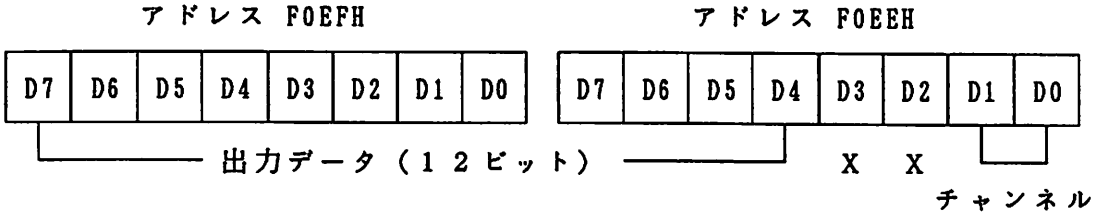
３０．「アナログ出力は簡単だ（Ｄ／Ａコンバータ）」

前のＡ／Ｄコンバータの逆処理を行う、すなわちコンピュータで計算したディジタルデータをアナログデータとして出力するものが「Ｄ／Ａコンバータ」である。実験で使用しているものは「コンテックＤＡ１２－４（９８）」であり、４つのチャンネル出力をもっている。分解能は１２ビットであり、出力形式はバイポーラ±１０Ｖである。Ｄ／Ａコンバータの場合には、ただ単に決められたフォーマットのデータを出力すれば、それに対応したアナログデータが得られるので、プログラミングは簡単である。

Ｄ／ＡコンバータのＩ／Ｏアドレスは

ＯＦＯＥＥＨ

としており、つぎのOFOEFH番地も占有している。すなわち、１６ビット１命令で実行できる。出力データのフォーマットは次のようになっている。



出力データ形式は前のＡ／Ｄコンバータの場合と同様に、１２ビット正負表現データである。下位のD3とD2の値は関係ない。また、下位D1とD0でチャンネルを選択する。例えば、チャンネル１にデータ C00Hを出力、すなわち、

```
OUT  DX,AX     (DX=0F0EEH, AX=0C001H)
```

を実行すると、Ｄ／ＡコンバータのＣＨ１の出力は－５Ｖとなる。

普通、Ｄ／Ａコンバータはコンピュータで計算した色々な値がどうなっているかを確認するために用いている。

３１．「ＭＡＳＭプログラミング」

それでは、前半の最後としてＭＡＳＭプログラムについていくつか説明しようと思うが、やっぱり、この辺で、Ａ君達に登場してもらって、質問形式のディスカッションを行ってもらおう。

Ａ君：「じゃあ、始めるけど、やっぱり一番ようわかっとうＤ君が答える人やね。あとの３人は質問する人。いいかいな？」
Ｂ君：「よか」
Ｃ君：「意義なし」
Ｄ君：「まあ、いいけど、僕もわからんやったらどうするや？」
Ｂ君：「そんなこと、あるわけないやんか」
Ａ君：「そしたら、卒論生のみんなも、もうそれぞれのテーマに関係したプログラムリストをもらっとうと思うき、始めるけど、テーマによっては少しプログラムが違うから、だいたい全体に共通した所を重点的に質問しましょう。それでは、誰か質問して」