＝１３００Ｄ
＝０００００１０１０００１０１００Ｂ
という結果がＡＸレジスタに求まる。そこで６４で割るためにＡＸを６ビット算術右シフトを行うと
ＡＸ＝０００００００００００１０１００Ｂ
となりこれは「２０」である。すなわち、シフトを行う前のＡＸの値は
ＡＸ＝０００００１０１００．０１０１００Ｂ
のように「６ビットの位置に小数点がある」と考えられる。このような表し方が「固定小数点演算」の基本である。小数点の右１ビット目は「２の（－１）乗すなわち０．５、右２ビット目は２の（－２）乗すなわち０．２５」を表している。そうするとＡＸの値は「20.3125 」を表している。この値は「100*13/64=20.3125 」に当然等しくなる。
さらに、除数の値を大きくして（除数を大きくすると演算誤差が小さくなる）「２５６（２の８乗）」とすると、乗数は256/5=51.2となるので「５１」として、乗算を行うと、
ＡＸ＝１００＊５１
＝５１００Ｄ
＝０００１００１１１１０１１００Ｂ
となる。この時、小数点の位置は
ＡＸ＝０００１００１１．１１１０１１００Ｂ
である。ここで、８ビット算術右シフトを行うが、これはＡＨレジスタの値そのものである。すなわち、シフトを行わなくても答えが得られていることになる。ただし、この場合は答えが「１９」となっている。実際はＡＸは「19.921875 」を表しているが、これは前に説明したようにビット操作で得られる値は「ＩＮＴ」化されるためである。従って、正確な値に近い整数値を得るには「四捨五入」しなければならない。「四捨五入」を行うには「固定小数点」データに「０．５」を加えればよい（どうしてかは各自考えてみよう）ので、このＡＸに
００００００００．１０００００００Ｂ
を加算すればよい。そうすると答えは
０００１０１００．０１１０１１００Ｂ
となり、整数部は「２０」が得られる。ＤＳＰでは取り扱っているデータの小数部桁が大きいので、四捨五入は行っていないが、より正確な結果を得るにはこの操作が必要である。
このように除数を８ビットでは「２５６」、１６ビットでは「６５５３６」に選べば、それぞれＡＨ、ＤＸに答えが得られる。これは　例えば「１」を掛ける場合に「２５６」を掛けて上位を取るということであり、単に「１」を「２５６」に対応させて考えているだけである。扱う値が正負表現のデータでも同じである。
このような方法は通常「１以上の値で割る、すなわち１以下の値を掛ける」場合には問題ない。しかし、時によってはあるデータを「１．３」倍したいということがあるが、この時には扱う数値の大きさを考慮していないとオーバーフローする場合があるので、注意しないといけない。「１．３」倍の時には被乗数８ビットでは、乗数が「３３３」となり、８ビットでは表せないので１６ビットに拡張して同様に計算すればよいが、被乗数１６ビットでは「８５１９７」となり計算不可能となる。この時には次のように行う。「１．３」倍を
１．３＝１＋０．３
とし、小数部を上述した方法で計算し、被乗数を加える（１倍は被乗数そのものである）。すなわち、
ＡＸ＊１．３＝ＡＸ＊１＋ＡＸ＊０．３
ただし、小数部の演算結果は上位（この場合はＤＸ）であるので、「ＡＸ＊１」の

部分は上位に加算する必要がある。これをプログラミングすると次のようである。
（ＡＸの値の１．５倍を求める。答えはＤＸ）

```
MOV  BX,AX        ;ＡＸの内容をＢＸに保存
MOV  CX,19661     ;乗数(0.3*65536)
MUL  CX
ADD  AX,8000H     ;小数部に0.5加算（四捨五入のため）
ADC  DX,BX        ;上位にAX*1.0を加算
```

（四捨五入を行わない場合は ADDは不要、ADCはADDとなる）

扱う数値が正の数だけであれば全ての計算はこれでよいが、正負表現の時には不都合が起こる。それは、小数部が「０．５を越える（０．５の時には１ビット右シフトでよい）」時である。この場合には１６ビットデータでは乗数が「３２７６７」より大きくなる。正負表現では「３２７６７」より大きい値はなく、これを越えると負の数となる。例えば「１．８」の場合には乗数は「５２４２９」となり、これは同じ計算に対しては「－０．２」に対応する。そうすると実際に行う計算は「１－０．２」となってしまう。そこで、正負表現時は

１．８＝２．０－０．２

として計算を行う必要がある。「０．２」は「１３１０７（=65536-52429）」であるから、小数部演算に対して乗数に「１３１０７」を用いると整数部から減算、乗数に「５２４２９」を用いると整数部に加算となる。ここでの演算方法は、既知の値（定数）で除算する場合であり、未知数（値のわからない数、すなわち変数）で除算するには「IDIV (DIV)」を実行せざるを得ない。

ここで説明した演算の方法は、じっくり考えて理解しよう。

## ３４．「よく使う条件ジャンプ命令」

プログラム流れを変える条件分岐（ジャンプ）命令を以下に示す。
なお、大小関係は

ＣＭＰ　Ａ，Ｂ　　（Ａはreg/mem、Ｂはreg/mem/imm、値は不変）

を実行した場合である。

| 分岐条件 | 符号無し数値の場合 | 符号付き数値の場合 |
|---|---|---|
| Ａ＞Ｂ | ＪＡ　／ＪＮＢＥ | ＪＧ　／ＪＮＬＥ |
| Ａ≧Ｂ | ＪＡＥ／ＪＮＢ | ＪＧＥ／ＪＮＬ |
| Ａ＝Ｂ | ＪＥ　／ＪＺ | ＪＥ　／ＪＺ |
| Ａ≦Ｂ | ＪＢＥ／ＪＮＡ | ＪＬＥ／ＪＮＧ |
| Ａ＜Ｂ | ＪＢ　／ＪＮＡＥ | ＪＬ　／ＪＮＧＥ |
| Ａ≠Ｂ | ＪＮＥ／ＪＮＺ | ＪＮＥ／ＪＮＺ |

A: Above　　B: Below
G: Greater　L: Less
E: Equal　　N: Not　　Z:Zero

例：JAE: Jump if Above or Equal
　　JNG: Jump if Not Greater

## ３５．「割り込みプログラムの中心はＤＳＰだ」

今までの話の中ででで「ＤＳＰ」という言葉が何度か出てきたが、これはみんなが卒業研究のためのプログラム作成において中核となるもので、１つの「マイクロコンピュータ」である。電力変換器の直接の制御はＰＣ９８０１コンピュータ（Ｖ３０ＣＰＵ）で行うが、そのための種々の演算はすべてこの「ＤＳＰ」によって行う。「ＤＳＰ」は通常の１６ビットＣＰＵ（８０８６など）に比べて、命令実行時間が非常に速く、例えば、ＣＰＵでの乗算命令が数μｓかかるのに対して、「ＤＳＰ」では同じ演算を１００ｎｓで行う。特に、卒論で作成するプログラムはできるだけ割り込み処理時間を短縮させる必要があるので、「ＤＳＰ」の利用は不可欠である。また、使用している「ＤＳＰ」は浮動小数点演算が可能であるが、実際のプログラムではこの演算は使用していない。これから、現在使用している「ＤＳＰ」について色々説明しよう。

「ＤＳＰ」とは「Digital Signal Processing Microprocessor（ディジタル信号処理プロセッサ）」のことである。

その前にＤＳＰの略歴を書いておこう。１９８０年代初期に最初のＤＳＰが登場し、それは

Ｉｎｔｅｌ　２９２０、　続いて　ＮＥＣ　μＰＤ７７２０

である。その後、

Ｔｅｘａｓ　Ｉｎｓｔｒｕｍｅｎｔｓ　ＴＭＳ３２０１０

が１９８２年に発表された。その後、第２世代ＤＳＰ「ＴＭＳ３２０ｓ」すなわち、

ＴＭＳ３２０２０（１９８５）とＴＭＳ３２０Ｃ２５（１９８６）

が発表された。また、

ＮＥＣ　μＰＤ７７２３０

も開発され、さらに新しいＤＳＰも出現している。

「ＤＳＰ」はディジタルフィルタや高速フーリエ変換などを行うのに適しているが、今ではその高速演算機能から様々な制御用として利用されている。

卒論で使用しているものは「ＴＭＳ３２０Ｃ２５」であるので、以下は、これについて述べることにする。

## ３６．「ＤＳＰの構成を知ろう」

「ＴＭＳ３２０Ｃ２５」はそれぞれ独立した「プログラムメモリ」と「データメモリ」をもち、それらのデータ転送は特殊な命令だけによって行われる。このＤＳＰは５４４ワード（１６ビット）のデータＲＡＭ（on-chip data RAM）と４Ｋワードのプログラム ＲＯＭ（on-chip maskable program ROM）をもっている。データＲＡＭ５４４ワードの内、２５６ワードの領域はプログラムあるいはデータメモリとしてアドレスできる。また、プログラムメモリとデータメモリは６４Ｋワードのアドレス空間がある。

次に示す図はＴＭＳ３２０Ｃ２５のレジスタ等の簡単な構成図である。

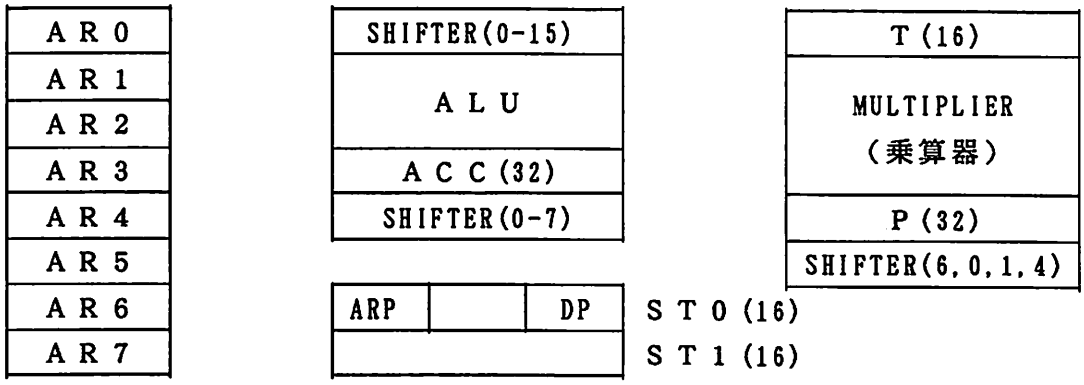
ＡＲ０－ＡＲ７：１６ビット補助レジスタ(Auxiliary Register)

ＳＴ０，ＳＴ１：ステータスレジスタ(Status Register)
（プロセッサのステータスと制御ビットをもつ）

ＡＲＰ：補助レジスタポインタ(Auxiliary Register Pointer)
（データメモリ参照に使用するＡＲを選択）

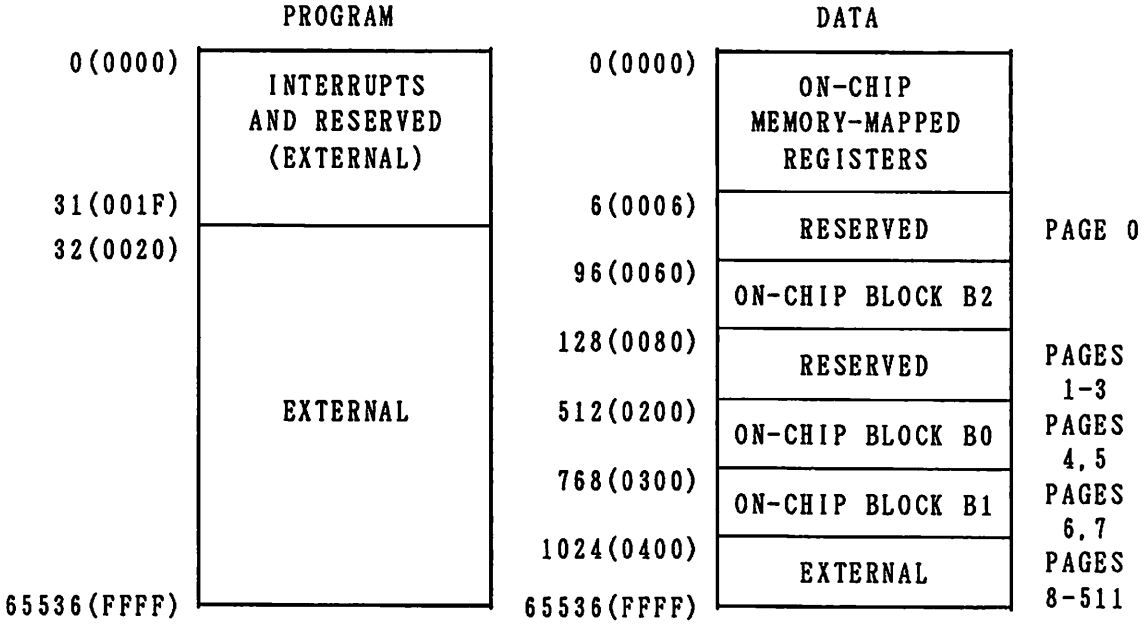
ＤＰ：データメモリページポインタ（Data Memory Page Pointer)
（アクセスできるデータメモリページを選択）

補助レジスタ(16)




T：１６ビットテンポラリレジスタ(Temporary Register)
（乗算器の入力およびスケーリングシフタのシフトコード）
P：３２ビット乗算結果レジスタ(Product Register)
ＡＣＣ：３２ビットアキュムレータ(Accumulator)

「ＡＣＣ」はＡＬＵの出力（ＣＰＵでのＡＣＣに対応）であり、上位１６ビット（ＡＣＣＨ）あるいは下位１６ビット（ＡＣＣＬ）だけの演算も可能である。
下図は使用するメモリ構成である（ただし、モードはマイクロプロセッサモードであり、「CNFD」命令実行後）。

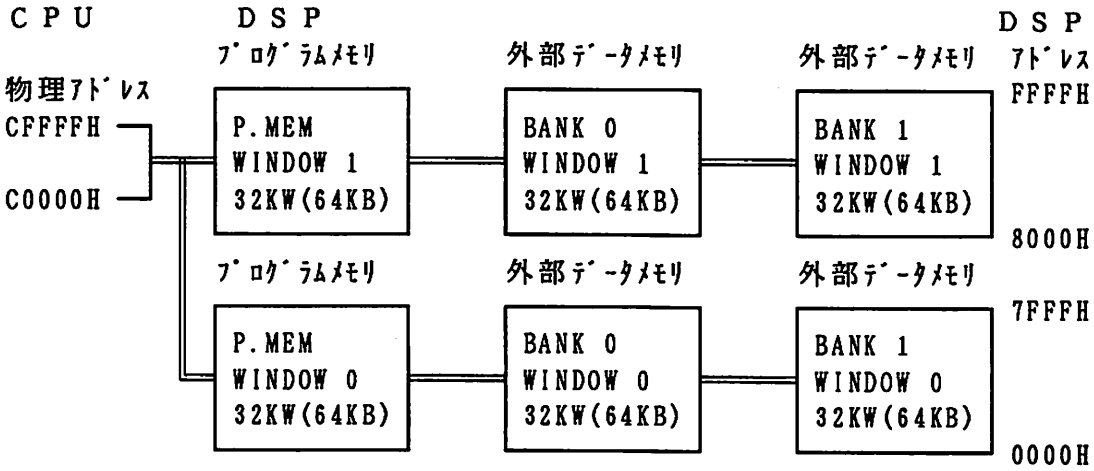



「PROGRAM 」は「プログラムメモリ」である。０から３１番地までは割り込みベクタ等に予約されているので、実際のメインプログラムは３２番地からである。
「DATA」はＤＳＰの「内部データメモリ」である。使用する内部データはページ４から７である。通常使用しているのは「ページ４（１２８ワード）」である。

## ３７．「同じメモリを使うなんてどうなっているの」

実験で使用している「ＤＳＰ」は「ＤＳＰボード」として作製された「ＡＥＣのＤＳ－Ｃ２５－Ｈ０２」というものであり、ＰＣ９８０１の拡張スロットに挿入して利用する。すなわち、ＰＣ９８０１の中で２つのコンピュータを同時に動作させることになる。

このボードではＤＳＰのプログラムメモリはそのままＰＣ９８０１コンピュータ（以下、単にＣＰＵという）のメインメモリの一部（メモリ位置は前のメモリ配置を参照）として構成され、さらに同じアドレス上に２バンク（バンクとは１かたまりのメモリと考えてよい）の外部データメモリ（ＤＳＰの内部メモリではない）が存在するような構成となっている。これらの構成を図に示すと以下のようである。




ＣＰＵから見ると物理アドレスC0000-CFFFF（セグメントC000H はスイッチで設定）に６４ＫＢの６つのメモリが並列にあると考えて、いずれかのメモリを選択して、データの書き込み、読みだしを行う。通常、外部データメモリは使用してなく、プログラムメモリの「ウインドＯ」だけを選択している。「ウインドＯ」におけるＣＰＵとＤＳＰとのアドレス関係は次のようである（ほかのメモリも同様）。

| ＣＰＵ物理アドレス | ＤＳＰアドレス |
|---|---|
| ＣＦＦＦＦ（８ビット） | ７ＦＦＦ（１６ビット） |
| ＣＦＦＦＥ（８ビット） | |
| ＣＦＦＦＤ | ７ＦＦＥ |
| ＣＦＦＦＣ | |
| | |
| Ｃ０００５ | ０００２ |
| Ｃ０００４ | |
| Ｃ０００３ | ０００１ |
| Ｃ０００２ | |
| Ｃ０００１ | ００００ |
| Ｃ０００Ｏ | |

ＤＳＰでは全ての命令およびデータは１６ビット単位で構成されるので、１アドレスは１ワードである。したがって、ＣＰＵの２アドレスがＤＳＰの１アドレスに対応する。例えば、ＤＳＰの「３番地」にＣＰＵからデータを書き込む場合には、１６ビット転送命令を用いて、

```
MOV  ES:[BX],AX   (ES=C000H, BX=0006, AX=転送データ)
```

のようにする。この対応はＣＰＵとのデータ転送において非常に重要であるので、覚えておくこと。
ＤＳＰはこのプログラムメモリに書かれた命令に従って実行していく。

## ３８．「ＤＳＰのプログラムはどこだ」

ＤＳＰのプログラムメモリはＣＰＵとの共有メモリとなっていることは前に述べた通りである。ＰＣ９８０１の中に２つのＣＰＵがあるということになるが、ＣＰＵから見てＤＳＰは一種の外部Ｉ／Ｏと考えておいてよい。ＤＳＰのプログラムはＣＰＵの物理アドレスC0000Hから始まる。ＤＳＰが実行している時はこのアドレス領域C0000H-CFFFFH のメモリはＣＰＵのバスから切り離され、ＤＳＰが停止している時だけ、このアドレス領域のメモリがＣＰＵのバスにつなっがていると考えてよい。したがって、ＤＳＰのプログラム領域にＣＰＵからデータを転送するときには必ず、ＤＳＰを停止状態にしなければならない。ＤＳＰのプログラムはＭＡＳＭプログラム等と同様にエディタを使って作成し、ディスクに保存する。作成したＤＳＰのプログラムは実行時にＤＳＰのプログラムメモリ上にロードする。

ＤＳＰはＣＰＵの外部Ｉ／Ｏと考えられ、ＤＳＰのコントロールはすべてＣＰＵからのコントロールデータによって行うことができる。以下、このコントロールについて説明する。使用しているＤＳＰボードはコンピュータの拡張スロットに幾つでも挿入でき（現在は１つだけである）、それぞれのボードにボード番号が付けられる。そのために、まず、目的のＤＳＰボードを選択しなければならない。現在の１つのＤＳＰボードはボード番号を

「００」

としているので、このデータを「ボード・セレクト・レジスタ（ＢＳＲ）」に出力する。「ＢＳＲ」のＩ／Ｏポートアドレスは

「０２Ｄ０Ｈ」

としている。次に、ＤＳＰをスタートしたり停止したり、あるいはメモリ選択などを行うための「コントロールレジスタ（ＣＲ）」が用意されている。このレジスタのＩ／Ｏアドレスは

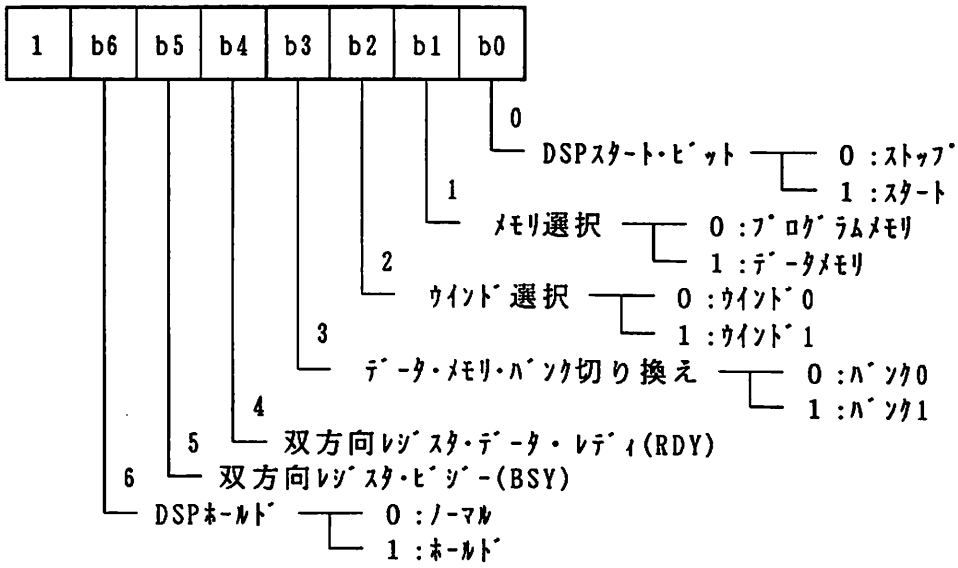
「０２Ｄ２Ｈ」

としており、データの構成は次のようになっている（次ページ参照）。各ビットを操作することによって制御する。

これらのコントロールはＭＡＳＭプログラムを実行する前に、すべてＣプログラムによって行っている。では、実際に行っている「ＣＲ」のデータを調べてみようまず、ＤＳＰボード番号「０」を選択するために

```
outp(0x02d0,0);                MOV  DX,02D0H
                               XOR  AL,AL
                               OUT  DX,AL
```

を実行する（命令はマニュアル参照、参考のために右側に対応したＭＡＳＭ命令を書いておく）。次に、ＤＳＰのプログラムをディスクからロードするが、これはＣＰＵによって行うので、その前に、ＤＳＰのメモリを選択を行う。コントロールデータに従って変更するが、「ＣＲ」の内容はある値に初期設定されているので、目的ビットだけの操作を行う。




コントロールデータは次のように初期設定する。

| | |
|---|---|
| ＤＳＰストップ | b0=0 |
| プログラムメモリ | b1=0 |
| ウインドウ0 | b2=0 |
| データバンク1 | b3=1　（使用しないので、どちらでもよい） |
| ＤＳＰホールド | b6=1 |

なお、ここでは、「ＲＤＹ」と「ＢＳＹ」は共に「０」としておく。したがって、コントロールデータは

１１００１０００Ｂ

となる。このデータを「ＣＲ」アドレス「０２Ｄ２Ｈ」へ出力すればよい。プログラムは

```
outp(0x02d2,0xC8);
```

```
MOV  DX,02D2H
MOV  AL,0C8H
OUT  DX,AL
```

となる。ビット「b0とb6」は

ＤＳＰスタート　b0=1,b6=0
ＤＳＰストップ　b0=0,b6=1

として操作する。「ホールド」とはＤＳＰのバスを「ハイインピーダンス」にするということである（つながってないと考えればよい）。「メモリ(b1)」、「ウインド(b2)」および「バンク(b3)」は以後変更しないので、これらのビットは変えないようにビット処理を行う。「ＲＤＹ」と「ＢＳＹ」については次で説明する。

さて、これでＤＳＰプログラムのローディンクの準備ができたので、ディスクからＤＳＰメモリへプログラムを転送する。ＤＳＰプログラムのロードは

ＬＤ２５．ＥＸＥ　（元はLD32.EXE）

によって行うが、これをＣプログラムでの「子プロセス」として次のようにプログラムして実行する。

```
spawnlp(P_WAIT,"d:ld25","d:ld25","######","-m","c000",NULL);
```

「######」はＤＳＰプログラムのファイル名であり、例えば、ドライブＢの子ディレクトリDFILE 内のDSP.OJ（ＤＳＰロードプログラムの拡張子は「．ＯＪ」と決まっている）であれば

```
B:¥¥DFILE¥¥DSP.OJ
```

のように指定する。なお、Ｃプログラムでは「¥」記号は「エスケープキャラクタ」として扱われているので、ディレクトリ句切り記号「¥」に解釈させるには「¥¥」のように重ねて記入する。また、この関数を使用する時には、プログラムの最初でインクルードファイル「<process.h>」を指定しないといけない。
以上で、ＤＳＰのプログラムロードが完了する。ＤＳＰのプログラムは別のプログラムを再ロードするか、あるいは、電源を切るまでメモリ上に残っている。というのは、このメモリ領域はＣＰＵのメモリの一部であり、現システムではＤＯＳが直接管理および使用していない領域だからである。

## ３９．「ハンドシェイクが必要だ」

Ａ君たちもみんな色々勉強してきて、ＤＳＰのこともやり始めた。久々に彼らの話しを聞いてみよう。

Ａ君：「ＣＰＵのことはだいたいわかったし、ＤＳＰのメモリ構成なんかもだいたいわかったけどくさ、ＤＳＰで何するんかいな」

Ｂ君：「何か、複雑な計算させるったい」

Ｃ君：「ＤＳＰはね、これから制御するための全ての計算をするったい。ＣＰＵだけでもできるけど、ＤＳＰでやると演算時間が短くて済むから、高速処理ができるというわけたい」

Ｂ君：「でもさ、実際に制御を行うのは、ＣＰＵの方やろ。インバータとかコンバータなんかの電圧や電流データを取り込むのはＣＰＵやき、ＤＳＰはどうやってそんなデータを取り込むわけ？。そして、ＤＳＰで計算した結果のデータはどうやってＣＰＵから出力するんや？」

Ｄ君：「だから、ＣＰＵからＤＳＰにデータを送ったり、ＤＳＰからＣＰＵにデータを送ったりしないといけないんだ。すなわち、ＣＰＵとＤＳＰとの間でデータ転送しなくちゃいけんとたい。そのために、僕たちが使っているＤＳＰボードには「デュアルポートレジスタ（ＤＰＲ）」という、１６ビットデータ転送用のＩ／Ｏポートがあるんよ。このポートレジスタにはＣＰＵからもＤＳＰからも書き込みと読みだしができるので、データのやり取りができるということたい。ＣＰＵのＤＰＲのＩ／Ｏポートアドレスは

０２Ｄ４Ｈ

となっているので、

```
MOV  DX,02D4H
MOV  AX,(データ)
OUT  DX,AX
```

と、ＣＰＵの方で実行すれば、ＤＰＲへデータが出力される。そこで、ＤＳＰの方は、ＤＰＲのポートアドレスが「ＰＡ１」と決まっているから、

```
IN   VAR1,PA1
```

を実行すれば、ＣＰＵから送られたデータをＤＰＲから取り込んで、VAR1の変数（アドレス）に保存できるということたい」

Ｂ君：「なるほどね」

Ａ君：「なんで、ＤＳＰの場合はＤＰＲのポートアドレスが「ＰＡ１」なの」

Ｃ君：「本当は、ＤＰＲのアドレスは「０１」に割り当てられているんだけど、そのアドレスを「ＰＡ１」という名前に定義しているだけさ」

Ａ君：「あ、そう。そしたら、ＤＳＰからＣＰＵにデータをおくる場合は？」

Ｄ君：「だいたいわかるやんか、ＤＳＰの方は、ＤＰＲにデータを出力するから、

```
OUT  VAR1,PA1
```

を実行するやろ。そして、ＣＰＵの方は、

```
        MOV   DX,02D4H
        IN    AX,DX
```

を実行したら、ＡＸレジスタにデータを取り込めるっちいうことたい」

ここで、Ｃ君がいきなり質問をぶちかました。

Ｃ君：「ちょっと、いいかいな」

Ｄ君はびっくりした。Ｃ君から質問がくるなんて、予期していなかった。

Ｄ君：「えっ、なに？」

Ｃ君：「あのね、ＣＰＵとＤＳＰの間でデータを送れるというのは、わかったけど、ＤＳＰとＣＰＵは全く別に動きよるんやろ」

Ｄ君：「そうくさ」

Ｃ君：「そしたら、例えば、ＣＰＵがＤＰＲにデータを出力したとするやろ。ＣＰＵがデータを出力した時にちょうどＤＳＰがＤＰＲからそのデータを取り込むことなんて、できんっちゃない？。反対にＤＳＰからＣＰＵにデータを送る場合も。要するに、ＤＳＰはいつＣＰＵからＤＰＲにデータが送られてくるかわからないんじゃないの」

わかる質問だったので、ほっとして、

Ｄ君：「そうなんです」

なんて、丁寧な言葉になってしまった。

Ｄ君：「そこで、「ハンドシェイク」ということをしなくちゃ、いけんったい」

Ａ君：「「バンドシェイク」？、なんやそれ。「マックシェイク」なら食ったことあるぜ」

Ｂ君：「俺はやっぱり、バニラが好きやね」

Ｃ君：「しょうがないね、ハンドシェイク（Handshake）は握手の意味たい。」

Ｄ君：「そう、データをやり取りする場合には、必ずこのバンドシェイクを行わないといけないんだな。データが送られてきたとか、データを送ったとかいう合図を行うことだよ」

Ｄ君はさらに説明を続けた。

「ＤＳＰボードにはＤＰＲにデータを送ったり、ＤＰＲからデータを取り込んだりした時に、それに応じてビットが動くレジスタをもっちょったい。まあ、フラグみたいなものだね。それが、ＣＰＵに対しては前に出てきた、コントロールレジスタ（ＣＲ）の「ビット４（ＲＤＹ）」と「ビット５（ＢＳＹ）」たい。これらのビットは次のように動くんだ。

ビット４　０：ＤＳＰからデータが送られてないとき
　　（ＣＰＵがデータを取り込んだ後）
　　１：ＤＳＰからＤＰＲにデータが送られたとき

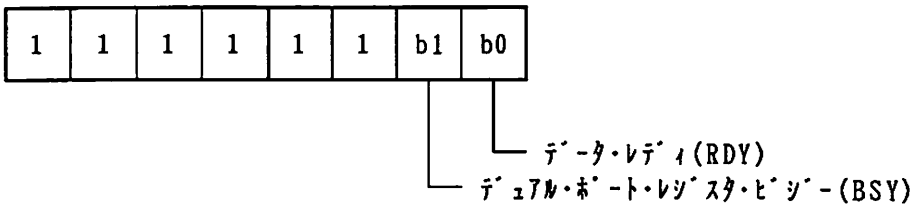
ビット５　０：ＣＰＵがＤＰＲに送ったデータをＤＳＰが取り込んだ後
　　１：ＣＰＵがＤＰＲに送ったデータをＤＳＰが取り込んでないとき

だから、ＤＳＰからＤＰＲにデータが送られてきたかどうかは、「ビット４(b4)」を調べていればわかるんだ。だけん、ＣＰＵがＤＰＲからデータを取り込む時には次のようにプログラムを組んでないといけないよ。

```
        MOV   DX,02D2H
DWAIT:  IN    AL,DX
        AND   AL,10H
        JZ    DWAIT
        MOV   DX,02D4H
        IN    AX,DX           (AX:DPRのデータ)
```

内容は自分で解読してね。

同じようにＤＳＰ側にもこのような「コントロールレジスタ」があるったい。その構成は次のようだ。




ビット０　０：ＣＰＵからデータが送られてないとき
（ＤＳＰがデータを取り込んだ後）
１：ＣＰＵからＤＰＲにデータが送られたとき
ビット１　０：ＤＳＰがＤＰＲに送ったデータをＣＰＵが取り込んだ後
１：ＤＳＰがＤＰＲに送ったデータをＣＰＵが取り込んでないとき
ＤＳＰがＣＰＵからデータを取り込む時には、「b0」を調べて、次のようにプログラムする必要があるんだ。

```
        LACK  1
WAIT:   IN    BUFF,PA0      (BUFFはCRﾃﾞｰﾀ保存用変数)
        AND   BUFF
        BZ    WAIT
        IN    VAR1,PA1      (VAR1:DPRのデータ)
```

こんな感じかな」
Ｃ君：「なるほどね、なかなか難しいね」
Ａ君：「俺、ようわからんやったき、もう一度勉強しよう」
Ｂ君：「やったね」（意味不明である）
Ｄ君は「でも、このＤＰＲのデータ転送では、一度に転送できるのは１６ビットデータなんだ」なんてことをいって「マックシェイク」を買いにいった（さっきのＡ君の言葉で食べたくなったのかな）。

それでは、ここで実際のプログラムについて付け加えておこう。Ｄ君の説明の通りにハンドシェイクを行ってテータ転送を行っているが、ＣＰＵの「ＣＲのビット５」およびＤＳＰの「ＣＲのビット１」は使用していない。これらのビットは相手がデータを取り込んだかどうかを確認するビットであり、このビットは続けて幾つものデータを送る場合に必要となる。現在、ＤＰＲは１つのデータを送るために使用するか、あるいは単にＤＳＰの演算が終了したというＤＳＰからの合図に使用している（ＤＳＰが演算を終了したかどうかは、ＣＰＵ側ではわからないので、ＤＲＰにダミーのデータを送ってＣＲのビットを動かすことで行っている）。実際には、複数のデータをＤＰＲを介さずにＣＰＵとＤＳＰの間で転送しているが、この転送方法は次で説明しよう。

## ４０．「データ転送は慎重に」

前に説明したようにＣＰＵとＤＳＰと間でのデータ転送はＤＰＲによって行うことができるが、複数のデータを転送する時には、１つのデータを転送する毎にハンドシェイクを行わないといけないので、そのためのプログラムが必要になり、結果として命令のステップ数が増え、実行速度の低下となる。ここでは、通常行っている「共有メモリによるデータ転送」について説明する。
ＤＳＰのプログラムメモリ（アドレス0000H-7FFFH、8000H-FFFFH）はＣＰＵの物