```
  1100          1110
+ 1345        + 1245
-------       -------
 12345         12345
```

とかね。決め方は自由だけど、普通はセグメントの値は変えないので、表せる物理アドレスが違うんだ。オフセットは１６ビットだから、その値は「０００００～ＦＦＦＦ」までだ。だから、セグメントが「１０００」の時は「１００００～１ＦＦＦＦ」まで、セグメントが「１１００」の時は「１１０００～２０ＦＦＦ」となる。すなわち、物理アドレスの範囲は違うけど、何れにしても表せる領域は６４Ｋバイトなんだ。オフセットの値だけで表せる範囲を「セグメント」とも呼んでいる。

セグメント（Segment） とは「分節、分割する」とかいう意味で、オフセット（Offset）は「変位、ずれ」という意味だよ。」

予想通り、Ａ君は優越感に浸っていた。Ｂ君は「なるほど」なんて言い、「あいつ知らん間に勉強しとるな」なんて思っていた。

１１．「ＣＰＵの内部も少しかじっとこう」

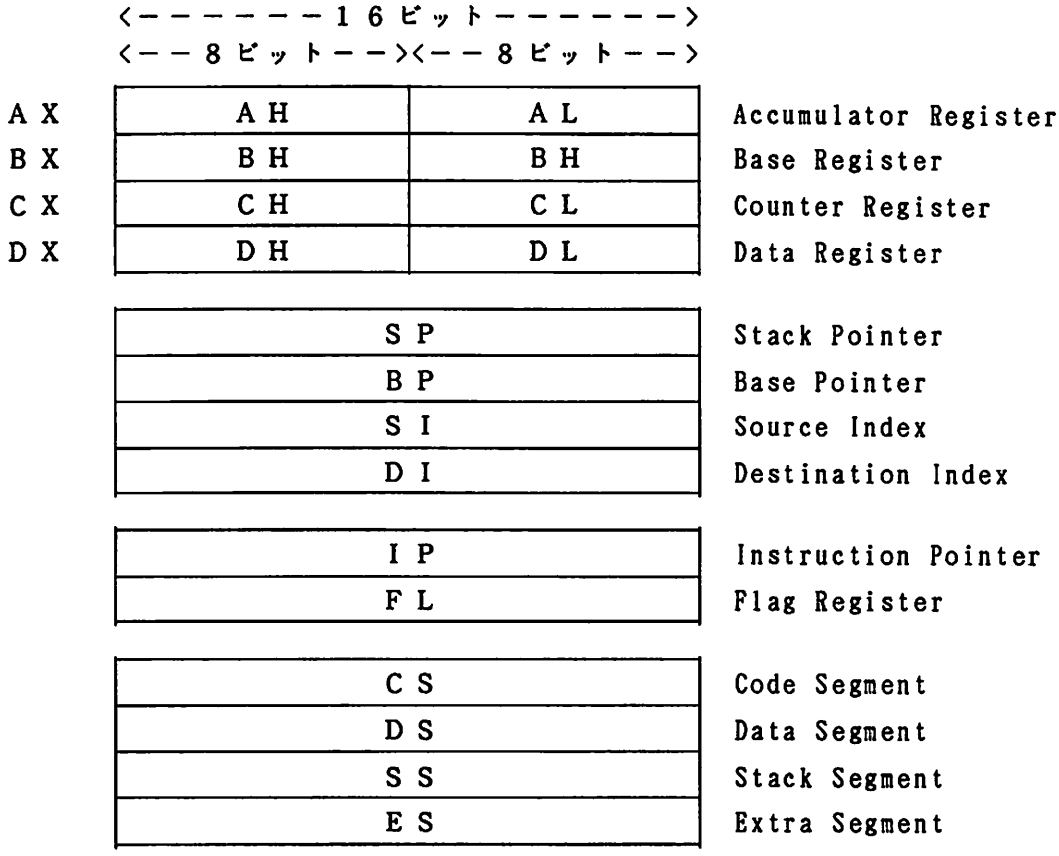
ＣＰＵとは、前にも述べたようにCentral Processing Unit （中央処置装置）のことである。一般に、コンピュータと言うのはＣＰＵとこれを効率よく動作させるために周辺に接続されている各種のＬＳＩから構成されたものをいう。ＣＰＵはその中核となるＬＳＩであり、ユーザーが作成したプログラムに従って色々な処理を行う。色々な処理と言って、一番最初に思い浮かぶのは計算、すなわち加算、減算などの四則演算である。しかし、実際はプログラム中に現れない様々な動作を行っている。例えば、ＣＰＵが演算を行うには、どの様な演算を行うのかを表すプログラムをメモリから読み込まなければならない。この時、メモリに対してアドレスを決め、バスにその信号を乗せる。また、データバスからデータを出し入れしたりする。さらに、キーボードからデータを取り込んだり、ディスプレイに表示するためにデータを送ったりしている。コンピュータの外部とデータをやり取りするものを「入出力装置」という。

このように、ＣＰＵの構成は大きく演算部分とバスを制御する部分に分けられる。前者は実行ユニット（ＥＵ：Execution Unit）と呼ばれ各種演算処理を行う。後者はバス・インターフェイスユニット（ＢＩＵ：Bus Interface Unit）と呼ばれ、アドレス計算や命令、データの転送処理を行う。ＣＰＵでのすべての演算は、ＥＵ部にある算術演算ユニット（ＡＬＵ：Arithmetic & Logic Unit ）で行われ、その演算結果がどういう状態にあるのか（例えば、結果が０になったなど）示す「フラグ」と呼ばれるレジスタがある。ＣＰＵの動作は、まず命令の読み込み、次にその命令の解読を行い演算などの処理を行い、必要ならば次にデータの読み書きを行う。命令がどの様にして取り込まれるのかということはそんなに重要ではないが、８０８６では、処理の合間を見計らって次の命令を読み込むという「プリ・フェッチ」動作を行っている。実際にプログラムを作成する場合には、ＣＰＵの内部構成はあまり知らなくてもそれほど問題にならないので、「そういうことか」くらいでいいでしょう。

１２．「知らないと話にならないレジスタ」

前の所で「レジスタ」という言葉が出てきたが、「レジスタ」とは本来「登録簿、記録、置数器」などの意味であり、これは一種のメモリと考えていよいが、それぞれ独自の役割をもっている。「レジスタ」はＣＰＵの内部にあるので、同じ演算を

行うにしても、通常のメモリを使用した場合に比べて、高速にその処理を行うことができる。８０８６の「レジスタ」の構成は次のようになっている。



すべてのレジスタは１６ビットで構成されており、ＡＸからＤＩまでは「汎用レジスタ」とも呼ばれる。

ＡＸ、ＢＸ、ＣＸ、ＤＸのレジスタは各種演算やデータ転送に使用する。８ビットでの演算もできるようにそれぞれのレジスタは８ビット単位でも使用できる。この中でＡＸ（アキュムレータ）はその中心となるレジスタであり、命令実行時間も他のレジスタに比べて速い。その他、ＢＸはアドレス間接指定に、ＣＸは繰り返し命令に、ＤＸは一部の演算命令などの補助として用いられる。

ＳＰ、ＢＰ、ＳＩ、ＤＩレジスタは主にアドレスを指定するために用いる。その中で、ＳＰはサブルーチンや割り込みルーチンあるいはデータの退避、復帰命令などでＣＰＵ（実際には使用するプログラム）が管理するので、ユーザーは使用しない。ＢＰは特殊な事（サブルーチンの引き数など）を行わない限りユーザーは通常の汎用レジスタとして使用しても差し支えない。

ＩＰは命令のアドレスを指定するレジスタであり、ユーザーは使用しない。

ＦＬは他のレジスタとは異なり、それぞれのビットが意味を持ち、演算の結果に応じて変化するビットや、ＣＰＵを制御するためのビットを有する。

ＣＳからＥＳはそれぞれセグメントを表す。ＣＳはＣＰＵが実行する命令が格納されているメモリのセグメントである。ＤＳはデータ転送において、そのデータを格納するメモリのセグメントである。また、ＳＳは上述のＳＰに関係するセグメントであり普通ユーザーは使用しない。また、ＥＳはデータ転送命令に用いられる。この中で、ＣＳ、ＳＳはほとんど気にしなくてよいが、ＤＳとＥＳは実際にプログラムを作成する場合に重要であるので、覚えておこう。

１３．「ＭＳ－ＤＯＳを覚えよう」

これからみんなが作成する色々なプログラムを実際にコンピュータで実行させるには、まず、そのプログラムの実行を管理するためにオペレーティングシステムプログラムでコンピュータを動作させる必要がある。そのオペレーティングシステムの一つに「ＤＯＳ（Disk Operating System ）」がある。「ＤＯＳ」はコンピュータの機種が異なっても同じようにプログラムを実行できるように規格化されたオペレーティングシステムであり、現在、ほとんどがこの「ＤＯＳ」のシステムを使用している。これから使用するものは「ＭＳ－ＤＯＳ（Microsoft-DOS ）」である。「ＤＯＳ」のシステムはフロッピーディスクあるいはコンピュータに内蔵されたハードディスクに入っている。ディスクの管理はすべて「ＤＯＳ」上で行われる。ユーザーは「コマンド」と呼ばれる命令をキーボードから入力することにより、様々な実行を行うことができる。従って、「コマンド」にどういうものがあるかを知ってないとどうしようもない。「コマンド」は実際に入力してみてどうなるかを確かめるのが一番よいが、ディスクの内容を壊す場合があるので注意すること。

それでは、コンピュータを実際に動かしてみよう。卒論でみんなが使用するコンピュータ（ＰＣ９８０１）にはすべて「ハードディスク」があるので、電源を入れればハードディスクからＤＯＳシステムを読み込んで立ち上がる。ディスプレイに色々なメッセージが表示されて、最後に

Ａ：¥＞

が表示されて入力待ち状態となる。この表示を「プロンプト」と言い、現在どこのドライブにカレントがあるかを示している。ここで、「ドライブ」とはディスク番号のことで、通常、システムを立ち上げたディスクのところが「ドライブＡ」となる。いまは、ハードディスクから立ち上げているので、ハードディスクが「ドライブＡ」、コンピュータ本体の上のフロッピディスク（１）が「ドライブＢ」、その下のディスク（２）が「ドライブＣ」となっている。さらに、本体には「ＲＡＭディスク」というものを組み込んでいるので、本体のメモリ上に仮想のディスクがあり、これが「ドライブＤ」となっている。「ＲＡＭディスク」はもう１つフロッピディスクがあると考えればよいが、電源を切るとその内容は消えてしまうので注意しないといけない。また、「カレント」とは一応、「現在、プロンプトを表示しているドライブ」と考えておいてよい。すなわち

Ａ：¥＞

の状態では「カレントドライブはＡ」と言う。まず、最初に覚えるコマンドは

ＤＩＲ

でしょう。これは「Directory 」のことで元来「住所録、人名簿」などの意味である。このコマンドはディスクの内容を表示するものである。実際に入力してみればすぐわかる。そのほか色々なコマンドがあるが、コマンドには「内部コマンド」と「外部コマンド」がある。「内部コマンド」は「ＣＯＭＭＡＮＤ．ＣＯＭ」というＤＯＳが管理するファイルの中にある。通常みんなが使用するコマンドは「内部コマンド」で十分である。そのコマンド内容については「ＤＯＳ」に関するマニュアルを参照のこと。また「外部コマンド」はハードディスクの「ＤＯＳＶＥＲ３３」のディレクトリに入っているが、これについてもマニュアルを参照して勉強してもよいが、あまり使うことがないので、興味があればの話です。

ディスクの中には色々なプログラムが入っている。プログラム名は

ファイル名．拡張子

で表示される（実際の表示では「．」はない）。ファイル名は８桁の英数字（頭は必ず英字）で拡張子は３文字である。ファイル名は自分で好きなように決めてよい

が、拡張子は通常、どういうプログラムを作るかで決まっている。この中で、拡張子が「ＥＸＥ」あるいは「ＣＯＭ」のプログラムがそのまま実行可能なファイルである。その他、「ＢＡＴ」や「ＳＹＳ」あるいは「＜ｄｉｒ＞」というのがあるがこれらはまた、先の方で説明しよう。最終的に、みんなが作成するのは実行可能ファイルすなわち「ＥＸＥファイル」である。どうやって作るのか、これから順次説明していくが、まだまだ先は長いんです。

１４．「とにかく何かプログラムを作ってみよう」

「とにかく何かプログラムを作ってみよう」といっても、そう簡単にはいかない。まず、フロッピーディスクを１枚用意し、そのディスクが「ＭＳ－ＤＯＳ」で使えるように「フォーマット」しなければならない。つぎに、卒論で使うＰＣ９８０１のハードディスクで立ち上がるシステムに適用できるようにする。つぎに、実際にプログラムを作成するには、何等かの「エディタ（編集プログラム）」を起動するので、その方法と使い方を覚えないといけない。さらに、実行可能ファイルにするには「コンパイル」および「リンク」という操作をしなければならない。このように、たったプログラムを作るだけなのに、たくさんの事を覚えないといけない。１度に覚えるのは無理だから少しずつ慣れて覚えることにして、ここでは、とにかくこれらの１連の操作を順次、行ってみよう。

最終的に作成するプログラムは「機械語」であり、このプログラムを「Ｃ言語」プログラムのサブルーチンとして実行させる。したがって、「Ｃプログラム」はその基本的なプログラミング技法を覚えればよい。ここでは、よく「Ｃ」に関する本にも載っているように、

「ディスプレイに（ＰＯＷＥＲ　ＥＬＥＣＴＲＯＮＩＣＳ）を表示させる」

というプログラムを作ってみよう。

まず、パソコンを立ち上げる。プロンプト表示後に新しいディスクをドライブＢに入れる。ディスクをフォーマットするために、ＤＯＳの「FORMAT」コマンドを次のように入力する（フォーマット済みのディスクの場合には必要なし）。（アンダーライン部を入力する。大文字でも小文字でもよい）。

```
A:¥>DOSVER33¥FORMAT B: /S
```

ＦＯＲＭＡＴプログラムが起動されるので、メッセージに従って入力し、フォーマットを終了する。ここで、

```
A:¥>DIR B:
```

を入力して、ドライブＢのファイルに「COMMAND.COM 」があることを確認しよう。

次に、Ｃ言語の「ソースプログラム」を作成するためにエディタを立ち上げる。ここで使用するエディタは「MIFES 」というものである。ここでは、「MIFES 」を立ち上げるための「EDIT」という「バッチファイル」から起動するので、次のように入力する。

```
A:¥>EDIT
```

そうすると、水色の「MIFES 」初期画面が現れるので、そのままカーソルのある位置からソースファイル名を入力する。ファイル名をここでは「TEST」とするので、次のように入力する。

```
B:TEST.C
```

ドライブＢにファイルを作るので「Ｂ：」を付ける。また、「Ｃ」のソースファイルの拡張子は「Ｃ」と決まっているので、「．Ｃ」を付ける。そうすると、編集画面となる。ドライブＢに編集しようとするファイルがなければ「新しいテキストです」というメッセージを表示する。ドライブＢに既にファイルがあれば、その内容が表示される。次のようなプログラムを入力する。（但し、Ｃ言語では大文字と小

文字を区別して入力のこと）。

```
main()
{
printf("POWER ELECTRONICS");
}
```

以上を入力し終わったら「MIFES 」を終了する。終了は「Ｆ・１」キーを押すとポップアップメニューが現れるので、カーソルキーで「Save & Quit 」を選択してリターンキーを押す。「MIFES 」の起動と終了は基本的にこれだけである。ただし、次の事に注意すること。 MIFESによって間違って関係ないファイルを開いたり、ＥＸＥなどを見ようとして（実際ＥＸＥファイルは見れない、オープンすると訳のわからない内容が現れる）開いた場合には、必ず「Quit（強制編集終了）」で終わること。そうしないと、開いたファイルを壊すことになる（これは少し MIFESの使い方を覚えた卒論生がよくやることである）。また、ＳＡＶＥしなくてよいファイルもできるだけ「Quit」で終了するように習慣をつけることが望ましい。さて、ここで、作成した「TEST.C」のプログラムがディスクにあるかどうかを見るために

A:¥>DIR B:

を入力し、確認してみよう。

さて、これで「Ｃ」のソースプログラムができたが、このままでは実行できなので、「コンパイル」と「リンク」という操作をしなければならない。その準備として「Ｃ」の「コンパイル」と「リンク」を行うプログラムを「ＲＡＭディスク」へコピーする。（使用しているシステムでは全てのこれらの作業を「ＲＡＭディスク」で行うようにしている。これは、コンパイル等の時間短縮のためである。）そこで次のようなコマンドを入力する。

A:¥>CCOPY

メッセージが色々出てくるが、あまり気にしなくてよい。これで必要なプログラムをすべて「ＲＡＭディスク」へコピーする。なお、このコマンドはこのパソコンで「Ｃ」を使用するために作成した「バッチファイル」である。また、この「ＣＣＯＰＹ」コマンドはパソコンの電源を入れてから「１度だけ」実行すればよい（コンパイルするたびに毎回実行しなくてよい）。これでコンパイルの準備ができたので、さっそくやってみよう。コンパイル等の作業はすべて「ＲＡＭディスク」すなわち「ドライブＤ」で行うので、「カレントドライブ」を「Ｄ」とする。そのために、次のように入力する。

A:¥>D:

これで「カレントドライブ」は「Ｄ」となり、プロンプトは「Ｄ：¥＞」となる。「Ｃコンパイラ」の実行ファイル名は「ＭＳＣ．ＥＸＥ」なので

D:¥>MSC

を入力すると、「ＭＳＣ」の起動メッセージが表示されたあと、次のメッセージが表示されるので、ファイル名を入力する。

Source filename [.C]:B:TEST

ソースファイルが保存してあるドライブは「Ｂ」なので「Ｂ：」を付ける。「Ｃ」のソースファイルの拡張子は「．Ｃ」と決っているので、省略してもよい。さらに次のメッセージに対して入力を行う。

Object filename [TEST.OBJ]:B:

コンパイルで作成するファイルは「オブジェクトファイル」と呼ばれるものである。その拡張子は「．ＯＢＪ」となる。ここでは、その「オブジェクトファイル」をドライブＢに作成するので「Ｂ：」を入力する。さらに、次のメッセージが表示されるが、これらについては何も入力せず、そのままリターンキーを押す。

Source listing [NUL.LST]:
Object listing [NUL.COD]:
これで、コンパイル作業が始まる。
ソースプログラムに「エラー」がある場合には、それに対応した「エラーメッセージ」が表示されるので、プログラムを修正してコンパイルをやり直す。「エラー」がなければ、カレントドライブ「Ｄ」のプロンプトが表示される。ここで、ドライブＢの内容をみて、「TEST.OBJ」があることを確認してみよう。
最後に「リンク」という作業を行うために次のように入力する。
D:¥>LINK
「ＬＩＮＫ」の起動メッセージが表示されたあと、次のメッセージが表示されるので、ファイル名を入力する。
Object Modules [.OBJ]:B:TEST
オブジェクトファイルが保存してあるドライブは「Ｂ」なので「Ｂ：」を付ける。オブジェクトファイルの拡張子は「．ＯＢＪ」と決っているので、省略してもよい。さらに次のメッセージに対して入力を行う。
Run File [TEST.EXE]:B:
実行可能ファイルをドライブ「Ｂ」に作成するので「Ｂ：」を入力する。さらに、次のメッセージが表示されるが、これらについては何も入力せず、そのままリターンキーを押す。
List File [NUL.MAP]:
Libraries [.LIB]:
これで、リンク作業が始まる。エラーがあればエラーメッセージが表示される。エラーがなげればカレントドライブＤのプロンプトが表示される。これで、実行可能ファイル「TEST.EXE」が作成されたので、ドライブＢにそのファイルがあるかどうかを確認する。さて、作成したファイルを実行してみよう。カレントドライブを「Ｂ」にするために
D:¥>B:
を入力し、次のように作成したファイル名を入力する。
B:¥>TEST
どうですか？。うまく表示してればＯＫです。
このように、たった何等かの文字を表示させるための実行可能ファイルを作成するためにたくさんの手順を行わなければならない。こうした手順をいつも行っていたのでは面倒なので、実際はこれらの手順を１度に行うような「バッチファイル」というものを作成し、それを利用している。これについては、ある程度コンピュータに慣れてから利用するということにしましょう。

１５．「コンパイルって何ですか」

コンピュータで何等かの計算あるいはこれから行うような制御する場合には、プログラムを作らなければならない。コンピュータすなわちＣＰＵが解読できるプログラムは２進数で表された「機械語（マシン語）」である。しかし、実際、人間が直接２進数の「機械語」でプログラムすることはほとんどない。そのために、人間に分かりやすい言語でプログラムを作成し、それを２進数に変換（翻訳）するということを行っている。このように人間がわかりやすいプログラムからＣＰＵが理解できる２進数の機械語に翻訳することを「コンパイラ」といい、この場合、１度に全部のプログラム翻訳を行う。これらの言語にはこれから覚えないといけない「アセンブラ言語」や「Ｃ言語」、科学計算用言語「ＦＯＲＴＲＡＮ」、さらに「ＤＳＰのアセンブラ言語」がある。これに対して、パソコンでよく知られているものに

「ＢＡＳＩＣ言語」がある。この「ＢＡＳＩＣ」はプログラムを１行づつ機械語に翻訳して実行するシステムであり、これを「インタープリタ」という。

「コンパイラ（Compiler）」とは「編集者」の意味で、「インタープリタ（Interpreter ）」とは「解釈、解説者、通訳」などの意味である。いずれにしても、最終的に機械語に翻訳されるわけであるが、実際に同じ様な内容のプログラムを実行させた場合には、その実行速度（演算を行う時間）が異なる。一般に、人間にとって分かりやすい言語ほど実行速度は遅くなると考えてよい。特に制御用のプログラムを組むためにはこの実行時間が重要になり、できるだけ実行時間の速いプログラムを作成する必要がある。そのためには「アセンブラ言語」を使用しなければならない。「アセンブラ言語」は「機械語」に直接対応した言語であり、ＣＰＵそのものの動作を表した言語である。したがって、コンピュータのハードウエアに直接関係するので、ハードウエアの構造自体を知ってないとどうしようもない。「ＢＡＳＩＣ」や「ＦＯＲＴＲＡＮ」などの高級言語になるほどハードウエアの知識は必要でなくなってくる。言い換えれば、このような言語は制御用としては向いていないということである。「Ｃ言語」は両者の中間的な言語であるが、これでも制御用としては実行時間が遅すぎる。

では、これから使用する「Ｃ言語」と「アセンブラ言語」について、話を進めよう。これらのソースプログラムは直接ＣＰＵが理解できるものではないので、コンパイラという翻訳をしなければならない。「Ｃ言語」だけについては、前節で説明したので、だいたいわかったとして、その中で、「リンク」というのがあるので、それについて説明しよう。例えば、膨大な大きさのソースプログラムをコンパイルする時、いちいちコンパイルしていたのでは時間がかかる。それでソースプログラムをいくつかに分割（通常はサブルーチン毎に分割）し、それぞれをコンパイルしで、あとで一緒にくっつけようというわけである。そうすれば、一部を変更した場合でも、そのソースプログラムだけをコンパイルすればよいことになる。この「一緒にくっつける」ということを行うものが「リンカー（Linker）」である。「くっつける」ということ「リンク（Link）」という。「Link」とは「結合する」の意味である。前述したように、Ｃ言語のソースプログラムは「ＭＳＣ」というコンパイラプログラムによって「オブジェクトファイル」というものを作成する。「オブジェクトファイル」はまだ、ＣＰＵが直接解読できるものではなく、そのあとに行う「ＬＩＮＫ」に対する色々な情報を含んだファイルである。「ＬＩＮＫ」によって、別のオブジェクトファイルと結合するための情報なども含んでいる。先の、説明では「コンパイル」とは「機械語に翻訳すること」と言ったが、通常は、この「オブジェクトファイル」を作成することを言っている。「ＬＩＮＫ」に対するファイルは必ず「オブジェクトファイル」である。

では、「アセンブラ言語」の場合にはどうだろう。特に「Ｃ言語」と変わるところはない。ところが「アセンブラ」と「Ｃ」では、言語が違うので、同じ「コンパイラ」を使うわけには行かない。このため、「アセンブラ言語」のコンパイラとして「ＭＡＳＭ」というものが用意されている。すなわち、「Ｃ言語」の「ＭＳＣ」に対して、「アセンブラ言語」では「ＭＡＳＭ」を用いるということが違うだけである。前述したように、実際にプログラミングする場合には、「アセンブラプログラム」を「Ｃ言語プログラム」のサブルーチン（「サブルーチン」とは一種の関数と思えばよい）として組むので、アセンブラプログラムはＣ言語に対応させて作成しなければならない。対応と言っても、形式的なプログラムの書き方に合わせればよい（ただし、複雑な事をしようと思えば、その形式的な事も理解していないといけないが）。ＣファイルとＡＳＭ（アセンブラ）ファイルを１つの実行可能ファイルとして作成する場合には、Ｃファイルは「ＭＳＣ」で、ＡＳＭファイルは「ＭＡＳＭ」でそれぞれコンパイルして、それぞれのオブジェクトファイルを作成し、その２つのオブジェクトファイルを「ＬＩＮＫ」で結合するということになる。

１６．「とにかくアセンブラも作ってみよう」

ここでは、「Ｃ」と「アセンブラ」のプログラムを作成し、コンパイルとリンクを行ってみよう。プログラムはアセンブラによって、０から９までの数字をディスプレイに表示させるものである。前と同様に「MIFES 」で次のようなプログラムを作成する。

「Ｃ」ソースプログラム（ファイル名は NUM.Cとする）

```
main()
{
nume();
}
```

「アセンブラ」ソースプログラム（ファイル名は NUMA.ASMとする）

```
_TEXT   segment byte public 'CODE'
        assume  cs:_TEXT
        public  _nume
_nume   proc near
        MOV   AX,0A000H
        MOV   ES,AX
        MOV   BX,20
        MOV   CX,10
        MOV   AL,30H
LOP:    MOV   ES:[BX],AL
        INC   AL
        ADD   BX,00A0H
        LOOP LOP
        RET
_nume   endp
_TEXT   ENDS
        END
```

作成し終わったらコンパイルをするために「カレントドライブ」を「Ｄ」とする。Ｃプログラムは前と同じようにコンパイルだけを行い、その「オブジェクトファイル」をドライブＢのディスクに作成する。

「アセンブラコンパイラ」の実行ファイル名は「ＭＡＳＭ」なので

D:¥>MASM

を入力すると、「ＭＡＳＭ」の起動メッセージが表示されたあと、次のメッセージが表示されるので、ファイル名を入力する。

Source filename [.ASM]:B:NUMA

ソースファイルが保存してあるドライブは「Ｂ」なので「Ｂ：」を付ける。「アセンブラ」のソースファイルの拡張子は「．ＡＳＭ」と決っているので、省略してもよい。さらに次のメッセージに対して入力を行う。

Object filename [NUMA.OBJ]:B:

「オブジェクトファイル」をドライブＢに作成するので「Ｂ：」を入力する。さらに、次のメッセージが表示されるが、これらについては何も入力せず、そのままリターンキーを押す。

Source listing [NUL.LST]:

```
    Cross reference [NUL.CRF]:
```

これで、コンパイル作業が始まる。

ソースプログラムに「エラー」がある場合には、それに対応した「エラーメッセージ」が表示されるので、プログラムを修正してコンパイルをやり直す。「エラー」がなければ、次のメッセージが表示され、カレントドライブ「Ｄ」のプロンプトが表示される。

```
    ***** Bytes free

    Warning Severe
    Errors  Errors
    0       0
```

ここで、ドライブＢの内容をみて、「NUMA.OBJ」があることを確認してみよう。これで「Ｃ」のオブジェクトファイル「NUM.OBJ 」と「アセンブラ」のオブジェクトファイル「NUMA.OBJ」が作成されたので、この２つのファイルを「ＬＩＮＫ」で結合して実行可能ファイルを作成する。前と同じように

```
    D:¥>LINK
```

を入力し、「リンカー」を起動する。

```
    Object Modules [.OBJ]:B:NUM+B:NUMA
```

オブジェクトファイルが保存してあるドライブは「Ｂ」なので「Ｂ：」を付け、２つのファイルを結合するので、「＋」でつないで入力する。さらに次のメッセージに対して入力を行う。

```
    Run File [NUM.EXE]:B:
```

さらに、メッセージが表示されるが、これらについては何も入力せず、そのままリターンキーを押す。

これで、ドライブＢに実行可能ファイル「NUM.EXE 」が作成されたので、実行してみよう。どうですか？。０から１までの数字が縦に表示されたと思います。見ずらい時は「ＤＯＳ」の内部コマンド「ＣＬＲ」で画面をクリアして実行させるとよいでしょう。基本的に「Ｃ」と「アセンブラ」のコンパイラはこのように行う。実際はこのような手順を組み込んだバッチファイルを用いているが、ある程度慣れるまではこのやり方で行うのがいいでしょう。卒論のプログラムではほとんど「Ｃ」と「アセンブラ」の２つのプログラムが必要となるので、普通、「アセンブラ」のファイル名は「Ｃ」のファイル名に「Ａ」のアルファベットを付け足した名前にしています。

興味のある人は、このアセンブラプログラムを解読してみてはいかが？

１７．「単純明解なアセンブラ命令」

いままで「アセンブラ」という言葉を使ってきたが、これはなにか？。もうだいぶわかってると思うが、これは機械語そのものに対応した言語であり、「アセンブリ言語」と呼ばれる。ＣＰＵが解読する命令は２進数で表された数字であるが、その数字でプログラムを組むことは非常に面倒であり、また、後でみる場合にも何がなんだかわからない。そこで、人間にとってわかりやすく、しかも機械語に１対１に対応している言語が考えられ、それが「アセンブリ言語」である。例えば、ＡＸレジスタに１０を代入する命令は、機械語では

```
    Ｂ８　１０
```

である。これをアセンブラで書くと

```
    ＭＯＶ　ＡＸ，１０
```

となる。この「ＭＯＶ」は「ＭＯＶＥ（移動する）」の略である。全ての命令はこのように英語の一部を用いてその処理を表す。この「命令の処理内容」を表す部分を「ニーモニック」と呼び、その後に続く「場所や値」を表す部分を「オペランド」と呼んでいる。「ＡＸ」を第１オペランド、「１０」を第２オペランドという。通常、アセンブラの命令では第２オペランドから第１オペランドへ数値が移動する。例えば、「ＡＸ」と「ＢＸ」の内容の加算を求める場合

　　ＡＤＤ　ＡＸ，ＢＸ

と

　　ＡＤＤ　ＢＸ，ＡＸ

の書き方がある。上の場合、加算結果はＡＸに保存され（ＢＸの内容は不変）、下の場合には、ＢＸ（ＡＸの内容は不変）に保存される。

　このように、アセンブラの命令は単純であり、そんなに難しくないが、単純なだけに何か複雑な処理をさせようとすると、そのプログラムテクニックが重要であり、全体の処理（演算）時間に影響を与えることになる。

１８．「アドレスはどこ行った」

　Ａ君は先ほどの「０から９までをディスプレイに表示させる」プログラムを作成して実行させていた。ところが、プログラムを作成していて、妙な事に気が付いた。Ｂ君ではわからないと確信していたので、Ｃ君を呼び出し質問したが、Ｃ君でもわからなかった。Ｃ君はそれに関してプロフェッショナルであるＤ君を呼び出した。じゃじゃーん。Ｄ君の登場である。さっそくＡ君は質問した。

Ａ君：「さっき、このプログラムを作ったんやけど、アドレスとかどうなっちょうと？　せっかく物理アドレスとか論理アドレスとかを勉強したとに」

Ｄ君：「それがマクロアセンブラのすごいとこたい」

Ｃ君：「マクロアセンブラ？　なんやそれ」

Ｄ君：「マクロアセンブラっちいうのは、アドレスなんかほとんど考えんでいいったい。ＢＡＳＩＣみたいに「ラベル」とか「変数」とかが使えて、ただ単にアセンブリ言語で書いていけばいいんだよ」

Ａ君：「そしたら、アドレスとか、メモリ構成なんか覚えなくてもいいやんか」

Ｄ君：「でも、ＤＳＰを使ったり、さっきのディスプレイに表示させる場合にはアドレスの事がわかってないと、話にならんったい。今の所はアドレスは考えなくていいばい」

Ａ君は「ふーん」、Ｃ君は「あ、そう」なんて言ったけど、二人ともどうもピンとこなかった。Ｄ君はさらに話を進めた。

Ｄ君：「コンパイルとリンクをして、実行可能ファイルを作成したでしょ。そのファイルはディスクに記憶されてるやろ。そして、それを実行するときには、そのファイル名を入力すれば実行できるやろ。このような操作はすべてＤＯＳのシステムがしようわけたい。ＤＯＳは入力されたコマンドに従って、ディスクからそのファイルを読みだし、メモリにロード（書き込み）するわけ。そして、ロードされたメモリから実行されるわけたい。その時、どこのメモリにロードされるかは、その時のメモリの使用状況をＤＯＳシステムが調べて、決めるんや。だから、同じプログラムでも、いつも同じメモリにロード（配置）されるわけやないんたい。ということは、いつもセグメントが変わっているとも考えられるやろ。だから、マクロアセンブラでプログラムを作成する時には、アドレス（セグメント）なんかはユーザーが決められないということさ。だけんがくさ、プログラムにアドレスなんか書かないんですよ。ユーザーが使用できる物理アドレスは９ＦＦＦＦＨまで、しかも、下位番地の方はＤＯＳのシステムがあるので、その範囲にロードされるんだ。」