[機械語序論　１回目]

この講義の目的は、機械語すなわちアセンブリ言語のプログラミングを通じて、コンピュータについての基礎的な仕組みを理解することです。アセンブリ言語をすなわち機械語を直接プログラミングする機会はそれほど多くはありませんが、コンピュータがどのように動作しているかについて理解することは情報科学、コンピュータ科学を学ぶ上でもっとも基本的な事柄の一つです。

４ビットマイコン（プロセッサ）を理解しよう

添付した資料にある４ビットのプロセッサについて、理解してみましょう。重要なポイントは以下の点です。

① コンピュータは、０と１の２進数で動作しています。０と１とは、電圧が高い、低い、あるいは、メモリでは電気がたまっている、たまっていない、といった２つの物理的な状態で表現されています。

② 現在のコンピュータのもっとも基本的な要素は、**メモリ**と**プロセッサ**です。メモリはプログラムやデータを格納する場所です。プロセッサはそのメモリからプログラムやデータを読み出して、プログラムを実行しています。プログラムとデータをメモリにいれて、プロセッサがメモリから読み出して実行する方式を、**ストアードプログラム方式**といい、現在のコンピュータの基本となる方式です。

③ プロセッサは、演算を行う**演算部**、プログラムの読み出して必要な信号を開け閉めする**制御部**（資料では指令部）からなっています。命令やデータのとおり道（実際は電線です）を「**バス**」と言います。データのとおり道は**データバス**、メモリにどのデータを読み出すかを伝えるバスを**アドレスバス**といいます。メモリの横にある「番人」（セレクタという）に接続されているのがアドレスバスで、操作するメモリを指定しています。

④ 電子回路は０と１の信号を入力して、ＯＲ／ＡＮＤ／ＮＯＴなどの論理操作を行い、０と１の信号を出力するもの。これで、２進数の加算や減算などもできます。

⑤ 三角でしめされているのがゲート。信号の流れを制御する。このゲートを制御する信号は、命令解読部（**デコード**）で作られる。現在の命令（０と１の組み合わせ）から作られます。

⑥ プロセッサの中にも一時的にデータを格納するメモリ（のようなもの）がある。**レジスタ**と呼ばれる。そのいくつかはプログラムからは見えない（例えば、現在の命令を保持している命令レジスタや読み出すメモリの番地を保持しているアドレスレジスタなど）。

⑦ 実行するプログラムの番地を保持しているレジスタを**プログラムカウンタ**と呼びます。

⑧ 演算の一時的な結果を保持するレジスタを**アキュムレータ**と呼ぶことがあります。実際のプロセッサではこのようなレジスタがいくつもあります。

⑨ メモリ上にあるプログラムのそれぞれの命令は、動作とその対象からなっています。動作を指定するのが、**命令コード（オプコード）**、対象を**オペランド**と呼びます。実際のマシンではオペランドのない命令もあります。

⑩ クロック信号が入力されると、順序制御部から①から⑥までの信号が順番に送られる。これが、コンピュータの速度を決定しています。この単純なプロセッサでは、⑤以外は同じパターンでゲートを開け閉めしますが、⑤だけは現在の命令から、デコード部で生成された信号でゲートを開け閉めします。

資料にしたがって、ゲートの開け閉めとデータの流れを追ってみてください。このプロセッサでは基本的に 4 ビットの命令コードはこれに続く 4 ビットのオペランドを持っています。メモリ上の同じような 2 進数の並びが時には命令コード、時にはオペランドとして読み出されます。「どのような意味でその 2 進数が読み出されているか」によって、データの流れる道筋が変わっていることに注意してください。「0100 1001」という機械語命令（前半 4 ビットが命令コード、後半 4 ビットがオペランド）は「1001 番地のメモリ内容を読み出し足される数とする」という命令です。

この 2 つの 4 ビット数値のメモリからの読み出しと、1001 番地の内容の読み出しの都合 3 回のメモリ読み出しが行われますが、読み出された内容がプロセッサのどの部分に格納されるかがそれぞれ違います。こういったデータの流れの制御を行うことがプロセッサ内で最も重要な動作であり、各ゲートの操作によってこれらが正しく制御されます。要は、クロック信号によって、決められた順にゲートを開け閉めして、命令をとりだし、必要なときに命令に対応したゲートを開けるという動作を高速に実行することにより、コンピュータが動作しているわけです。

アセンブリ言語とは

さて、マシン語をそのものでは０１のパターン、つまり数字ですので、これをつかってプログラミングするのは人間にとって非常に面倒な作業になります。そのために考えられたのが、アセンブリ言語です。基本的に、マシン語に１対１に対応するように記号を使って表記したのがアセンブリ言語です。機械語にはどのような操作をするのかを示すオプコードと何についてその操作を行うかのオペランドがあるのは説明しました。オプコードを表す記号を**ニーモニック**といいます。

例えば、メモリから読み出す操作をロードといいます。また、格納する操作をストアといいます。加算はＡＤＤなので、上の例では例えば、

```
load 9
addi 1
store 11
```

というように表記したのが、アセンブリ言語です。このマシンでは、簡単なのでこんなものですが、実際のマシンではオペランドはレジスタだったり、メモリだったり、値そのもの（即値、イミディエトという）します。そのための記法があり、マシンごとに決められています。

アセンブリ言語で記述されたプログラムは、アセンブラによって、機械語に翻訳（変換）されます。アセンブリ言語は基本的には機械語と１対１なので、アセンブリ言語で書かれたプログラムは機械語で書かれたプログラムとは同じものとして考えることができます。

Ｃ言語などプログラミング言語で書かれたプログラムは、コンパイラによってアセンブリ言語に翻訳され、さらに機械語になり、プロセッサで実行されます。ためしに、どのようなアセンブリ言語になっているかを確認してみてください。Ｃコンパイラでは、－Ｓ　オプションをつけてコンパイルすると、.sというファイルができるはずです。どんなコードができているかをみてください。たとえば、Ｃのプログラムを t.s とすると、

```
%  cc –S t.c
```

でコンパイルすると、t.s というファイルにアセンブリプログラムが出力されているはずです。

x86 プロセッサについて

この講義では、インテルの x86 ファミリーと呼ばれるプロセッサを題材に進めていきます。このプロセッサは学類の計算機で使われているプロセッサであり、普通のＰＣに使われているプロセッサでもあります。あまり、初めて学ぶプロセッサとして適当とはいえませんが、もっともみじかなプロセッサということで、このプロセッサを使うことにしました。

x86 プロセッサはいろいろな種類がありましたが、この講義では 80586 移行のいわゆる Petinum プロセッサファミリを対象とします。プログラミング上の主な特徴を挙げておきます。

① ＣＩＳＣ(Complex instruction set computer)であるが､内部的に簡単な命令に分解して実行する機能を持ち、非常に高速化されている。
② 論理的なアドレス空間は３２ビット
③ レジスタは、ＰＣ（プログラムカウンタ）のほか、３２ビットの汎用レジスタとして、eax, ebx, ecx, edx, esi, edi, esp, ebp の８個のレジスタがある。このうち、esp は、スタックポインタ、ebp はベースレジスタと名づけられ、ソフトウエア的につかい方が決まっている。
④ メモリアクセスする場合には、豊富なメモリアクセスモードが使える。
⑤ 浮動小数点レジスタは８個で、スタック状に使う。

このプロセッサは歴史的な経緯を引きずっており、非常に複雑な命令セットになっていますが、この講義では、必要な部分のみを使ってプログラミングすることにします。詳しく知りたい方は、Intel から出されている「Pentium ファミリー　デベロッパーズマニュアル（下）アーキテクチャとプログラミングマニュアル」などを参照してください。

次回は、アセンブリ言語のプログラミング環境について解説します。アセンブリ言語のプログラムの実行の確認やデバックのために、gnu debugger 、gdb を使いますので、その使い方についても解説します。