第7章 コンピュータの仕組み

Ⅰ. 計算の実現機構

コンピュータの扱う情報は、プログラムとデータに大別される。

プログラム ： コンピュータの計算処理手順にかかわる情報

データ　　： コンピュータが処理する対象の情報

現在コンピュータは普通メインメモリ上にプログラムとデータを両方保持し、プログラムに従って計算を進める。このような機構を**プログラム内蔵方式**といい、プログラム内蔵方式のコンピュータを**フォンノイマン(*von Neumann*)型コンピュータ**と呼ぶ。

1. コンピュータの基本構成

コンピュータの中核を成すのは**主記憶装置（メインメモリ）**と**中央処理装置(*CPU: Central Processing Unit*)**である。

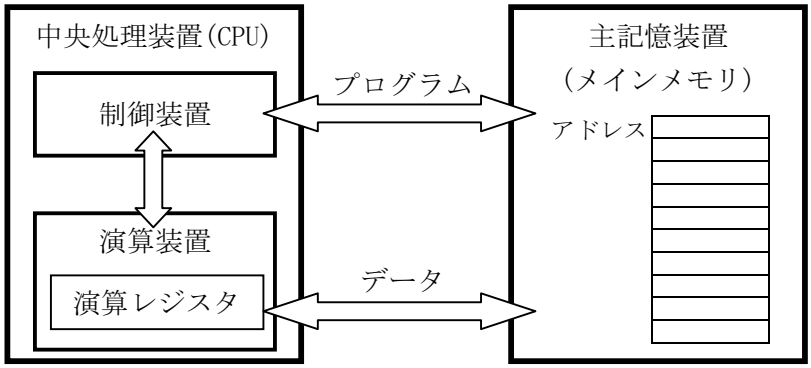
CPU の中には、**制御装置**と**演算装置**がある。これら両方を 1 つの半導体集積回路に詰め込んだものを**マイクロプロセッサ**乃至 ***MPU*(*Micro Processing Unit*)**という。

制御装置 ： メインメモリ上のプログラムに従って演算装置を動かしたり、メインメモリへデータを読み書きしたりする。

演算装置 ： **演算レジスタ**（一時的にデータを読み込んで保持する装置）のデータを対象として演算を行う。

初期のコンピュータには演算レジスタ 1 つしかなく、これは**アキュムレータ(*AC: accumulator*)**と呼ばれていた。メインメモリ上の情報の選択には、記憶域に割り当てられた**アドレス(*address*)**が用いられる。

※ 教科書 p.154 には *accumelater* と書かれていますが、恐らく誤植です。



2. 機械語レベルのプログラム例

機械語（人間にとって読み易い高級プログラム言語ではなく、機械の処理を直接的に表した言語）レベルのプログラムは、コンピュータに対する**命令(*instruction*)**の並びとして記述される。個々の CPU で利用できる命令群のことを**命令集合(*instruction set*)**と呼ぶ。次ページの表にその例を示す。

種類	内容	意味
データ転送命令	load A	アドレス A のデータを演算レジスタに読み込む
	store A	演算レジスタのデータをアドレス A に書き込む
演算命令	add A	アドレス A のデータを演算レジスタの値に加える
	subtract A	アドレス A のデータを演算レジスタの値から引く
分岐命令	jump A	アドレス A にプログラムの実行を飛ばす
	jumpzero A	演算レジスタの値が 0 ならばアドレス A に飛ばす
その他	write	演算レジスタのデータを出力する
	halt	プログラムの実行を停止する

命令には、**データ転送命令**(読み書きの命令)・**演算命令**(四則演算や論理演算等の命令)・**分岐命令**(条件分岐等の命令)などがある。命令は**命令コード(*operation code*)**と**オペランド(*operand*)**の組合せからなる。例えば「load A」ならば「load」が命令コード、「A」がオペランドである。

細かいプログラムの例はここでは省略する(ED21 の練習でやった筈)。

機械語レベルのプログラムは、高級言語(高水準言語)のプログラムと比べて非常に解り難い。それでも、load や jump といった英単語で書かれているからまだマシなのであり(厳密にはこれをアセンブリ言語という)、本当に機械が扱うのは更にこれを二進数で表したもの(バイナリプログラム)である。

II. 論理演算と組合せ回路

2 進符号表現の演算を実現する回路には、**組合せ回路**と**順序回路**とがある。組合せ回路は、記憶を持たない回路である。すなわち、内部状態を持たず、n ビットの入力から即座かつ一意に m ビットの出力を返す関数(**論理関数**)を実現する回路である。

(「課題対策プリント」では論理関数の値は 1 ビットと書いたが、別に多ビットでも構わない)

この節の内容については、「課題対策プリント No. 1 (論理演算と組み合わせ回路演習)」を参照。

完備性

「課題対策プリント」では**完備性**について書かなかったので、それについて触れておく。

一般に、「入力が n ビットで出力が 1 ビット」という関数(下の式)は、2^{2^n} 種類ある。

$$f(x_1, x_2, \dots, x_n) : \{0, 1\}^n \to \{0, 1\}$$

n ビットの入力の組 $(x_1, x_2, \dots, x_n)$ は 2^n 通りあり、それぞれについて 0 か 1 を出力するから、全部で 2^{2^n} 通りである。この 2^{2^n} 種類の関数全てを表せる演算の組合せを**完備な組合せ**という。

完備な組合せとして最も有名なものが {AND, OR, NOT} の組合せである。つまり、AND, OR, NOT の 3 つの演算だけで 2^{2^n} 種類の全ての関数を表せるのである。

(本当に完備なのか気になる人は、教科書 p. 159 の証明を見て欲しい)

「課題対策プリント」に書いたが、NAND という演算を用いると、AND も OR も NOT も表せる。ということは、2^{2^n} 種類の全ての関数は NAND だけで表せるということである。つまり NAND は単独で完備な演算である。これは、NOR についても同様である。

(NOR だけで AND も OR も NOT も表せることについては、気になる人は示してみましょう)

III. 第 7 章 あとがき

たぶん今月作っているシケプリよりも先月の「課題対策プリント」の方が解り易いと思います。それは、先月はまだ元気と情熱があったからです。遣っ付け仕事のシケプリですみません。