

■目次 (ほげが今回の範囲)

第1部 0と1だけでどうやって計算するの?	⇒	★3 負の数の表現と減算の仕組み
第2部 コンピュータの気持ち	⇒	★4 コンピュータの構成 (1)
第3部 情報をどのように表現するか		★5 CPUの仕組み
第4部 コンピュータシステム		★6 CPUと機械語

第4回の授業では、2進数で負の数を表現法する方法を学んだ。また、第3回で学んだ加算の場合と同様に、論理ゲートを組み合わせれば減算を行う論理回路を作れることも知った。今回は、「第1部」の締めくくりとして、まずは加算と減算を一つの論理回路で実現する方法を考える。次に、2進数と2の補数表現の復習を兼ねて、コンピュータによる演算で困ったことが起こる場合(オーバーフロー)について検討する。その後は、これまで学んだことをもとにしてコンピュータの仕組み、特にその中核であるCPUの仕組みを考える「第2部」に入っていく。

★3 負の数の表現と減算の仕組み (承前)

★3.4 複数桁の加減算ができる論理回路

第3回の授業で登場した複数桁の加算器と、第4回に登場した複数桁の減算器は、ほとんどの部分が共通の部品(全加算器)でできていた。少し回路を工夫すると、両者の機能を併せ持ち、加算と減算を切り替えて実行できる論理回路を作ることができる(下図右)。この回路は、 F が0ならば4ビットの2進数同士の和 $A_3A_2A_1A_0 + B_3B_2B_1B_0$ を出力し、 F が1ならば差 $A_3A_2A_1A_0 - B_3B_2B_1B_0$ を出力する。ただし、4ビットで2の補数表現を用いると仮定している。

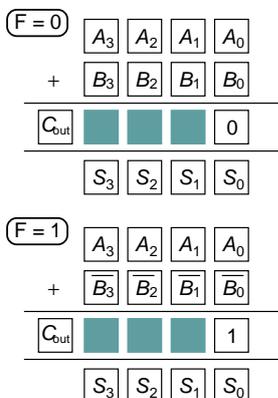
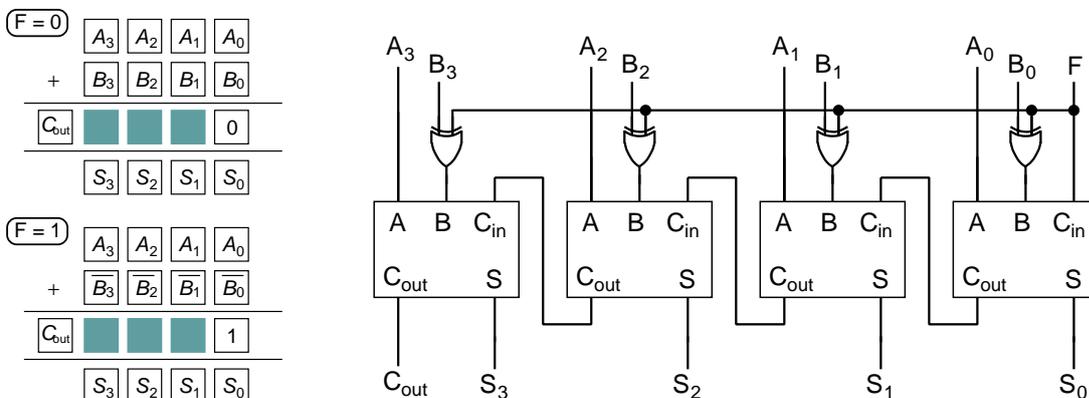
この回路の動作をもう少し詳しく説明すると、次のようになる。

- $F = 0$ のとき: B_i ($i = 0, 1, 2, 3$) がそれぞれ入力される XOR ゲートは、 B_i の値そのものを出力する(☆1)。また、最下位桁の全加算器の C_{in} への入力は0である。したがって、このときの回路は4ビットの加算器と同じ動作をする。
- $F = 1$ のとき: B_i がそれぞれ入力される XOR ゲートは、 B_i の値を反転したものを(\bar{B}_i)を出力する。また、最下位桁の全加算器の C_{in} への入力は1である。これらをあわせて、 $B_3B_2B_1B_0$ の2の補数をとった値と $A_3A_2A_1A_0$ の加算を実行することになる。つまり、このときの回路は4ビットの減算器と同じ動作をする。

☆1) F と B_i を入力として XOR の真理値表を書くと次のようになる。

F	B_i	出力
0	0	
0	1	
1	0	
1	1	

これを見ると、 $F = 0$ なら B_i の値がそのまま出力され、 $F = 1$ なら B_i を反転した値が出力されることになるのがわかる。



★ 3.5 オーバーフロー

★ 3.5.1 オーバーフローとは & 符号なしの 2 進数の場合

Q1. 4 ビット符号なし (☆2) の 2 進数で表せる最大の数はいくつか. 2 進数と 10 進数の両方で答えなさい.

Q2. 15 と 2 を 4 ビット符号なしの 2 進数で表しなさい. さらに, $15 + 2$ を 2 進数の筆算で計算しなさい.

Q2 の答えを眺めると, 15 も 2 も 4 ビットで表せる数なのに, 加算すると最上位桁からの桁上げが発生してしまい, 和を表すのに 5 ビット必要になってしまっている. このことは, 人間が筆算する場合にはなんともないが, コンピュータの気持ちになるとやっかいな問題となる. なぜなら, コンピュータの内部では普通, 整数を固定のビット長 (☆3) で表す前提で加減算の回路が作られているからである. 例えば★3.4 の加減算回路は入力が 4 ビットでなければならず, 5 ビット以上必要な数が出てきてしまうと, 正しい演算ができなくなってしまう (☆4).

上記の例のように, **固定の桁数で表した数同士の演算の結果が, その桁数で表せる範囲を外れてしまう**ことを, **オーバーフロー** (☆5) という. 4 ビット符号なしの 2 進数の場合, $15 + 2$ の演算はオーバーフローを起こす. 結果を 4 ビットの範囲で解釈すると 0001 ということになり, $15 + 2 = 1$ というおかしな結果となってしまう (☆6)

★ 3.5.2 符号ありの 2 進数の場合

2 の補数を用いて負の数も表現する場合にもオーバーフローは起こる. 復習を兼ねて, 加算の例で確かめてみよう.

Q3. 4 ビットで 2 の補数表現を用いて正負の数を 2 進表現するとして, 右の 10 進 2 進対応表を完成させなさい. 「:」の部分も省略せずに全て書くこと.

10 進	2 進	備考
:	:	最大の数
1		
0	0000	
-1		
:	:	最小の数

Q4. 次のそれぞれの数について, 2 の補数表現を用いて 4 ビットの 2 進数で表し, それらの和を求めなさい. さらに, 最上位桁からの桁上げを無視して結果を 4 ビットの数として解釈するといくつになるか, および, オーバーフローしているかどうか, を答えなさい.

(1) 4 と 3, (2) 4 と 4, (3) -4 と -4, (4) -4 と -5

☆2) 0 以上の整数のみを表すやり方で, ってください.

☆3) 現代のコンピュータでは, ビット長が 8 の倍数 (8, 16, 32, 64, ...) であることが多い. 理由はいづれまた.

☆4) 実は, C_{in} と C_{out} をうまく使って複数桁の加減算を何度も繰り返すようにすれば, そういう場合にも対処可能なのだが, 詳しい説明は省略する.

☆5) overflow

☆6) オーバーフローすると演算結果を信用できなくなってしまうので, コンピュータの演算回路はオーバーフローを検出できるようになっている. 例えば, ★3.4 の回路で 4 ビット符号なしの数の加算を行う場合, 最上位桁からの桁上げ C_{out} が 1 であればオーバーフローである. 符号ありの場合はもう少し複雑なので説明は省略するが, 同様に検出できる.

★ 3.5.3 C 言語プログラムとオーバーフロー

	overflow.c	実行結果
<pre> 1 #include <stdio.h> 2 3 int main(void) 4 { 5 char x; // char 型は 8bit 符号付き整数 6 unsigned char y; // unsigned char 型は 8bit 符号なし整数 7 8 x = 126; printf("x = %d\n", x); // char 型の変数に 9 x++; printf("x = %d\n", x); // 126 を代入して 10 x++; printf("x = %d\n", x); // 1 を加えていってみる 11 printf("\n"); 12 y = 126; printf("y = %d\n", y); // unsigned char 型の変数に 13 y++; printf("y = %d\n", y); // 126 を代入して 14 y++; printf("y = %d\n", y); // 1 を加えていってみる 15 printf("\n"); 16 y = 254; printf("y = %d\n", y); // unsigned char 型の変数に 17 y++; printf("y = %d\n", y); // 254 を代入して 18 y++; printf("y = %d\n", y); // 1 を加えていってみる 19 20 return 0; 21 }</pre>	<pre> x = 126 x = 127 x = -128 y = 126 y = 127 y = 128 y = 254 y = 255 y = 0</pre>	

	overflow2.c	実行結果
<pre> 1 #include <stdio.h> 2 3 int main(void) 4 { 5 char x; // char 型は 8bit 符号付き整数 6 int y; // int 型はコンピュータによって様々 7 // でも今時の普通は 32bit 8 9 // そのことを確認。「sizeof ほげ」は、ほげのビット長を 10 // 1 バイト (=8 ビット) 単位で教えてくれる 11 printf("char: %d, int: %d\n", 12 8*(int)sizeof x, 8*(int)sizeof y); 13 14 // printf の書式指定に %x と書くと 16 進数で出力 15 y = 9; 16 printf("y = %d %x\n", y, y); 17 y++; printf("y = %d %x\n", y, y); 18 y++; printf("y = %d %x\n", y, y); 19 printf("\n"); 20 y = 0x7ffe; // 16 進数で 7FFE になる数を y に代入 21 printf("y = %d %x\n", y, y); 22 y++; printf("y = %d %x\n", y, y); 23 y++; printf("y = %d %x\n", y, y); 24 printf("\n"); 25 y = 0x7fffffff; // 7F FF FF FE 26 // = 01111111 11111111 11111111 11111110 27 printf("y = %d %x\n", y, y); 28 y++; printf("y = %d %x\n", y, y); 29 y++; printf("y = %d %x\n", y, y); 30 31 return 0; 32 }</pre>	<p>(見やすくするために改行を適当に入れてます)</p> <pre> char: 8, int: 32 y = 9 9 y = 10 a y = 11 b y = 32766 7ffe y = 32767 7fff y = 32768 8000 y = 2147483646 7fffffff y = 2147483647 7fffffff y = -2147483648 80000000</pre>	

★ 4 コンピュータの構成

現代では、世の中の身の回りの様々なものに「コンピュータ」が内蔵されている。この授業の目標は、そのようなコンピュータたちが何をやっているのか、それらの動作の仕組みを学ぶことである。これから数回の授業（第2部）では特に、コンピュータの中核となっている「CPU」と呼ばれるものに焦点をあて、その仕組みの理解を目指す。

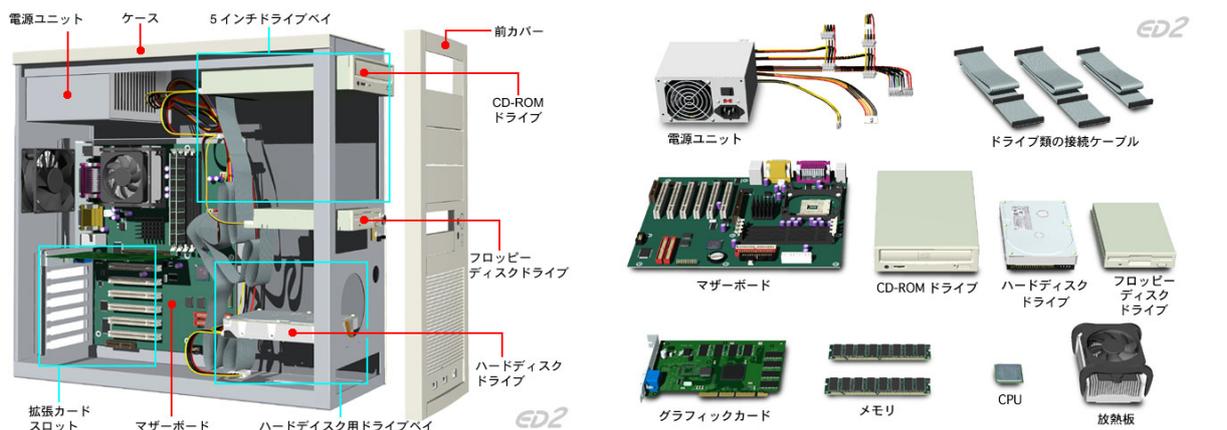
★ 4.1 コンピュータ共通の仕組み

★ 4.1.1 PC を分解してみよう

まずは、「コンピュータ」と聞いて真っ先に思い浮かびそうな、PC(☆7)の中身を覗いてみよう。右下図(☆8)は、典型的なPCがその周辺機器とともに設置されている場面を描いている(描かれてるのは一昔前のものだが、現在でも本質的にはそう変わらない)。PCに接続される周辺機器は、次のように大きく分類できる：

- **入力装置:** コンピュータに情報を入力する。キーボード、マウスなど
- **出力装置:** コンピュータからの出力を取り出す。ディスプレイ、プリンタなど
- **補助記憶装置:** コンピュータに入力／から出力される情報を保存しておく(あとで「主」記憶も出てきます)。ハードディスクドライブなど。

次の図は、PC本体の内部と、PCを分解して部品を取り出した様子を表している。



この図に描かれているPCの場合、ケースの内部にハードディスクドライブ等も内蔵されているが、これらは上述の周辺機器の一種である。他にも様々な機器があるが、コンピュータとしての機能の中核を担っているのは、CPU(☆9)とメモリ(主記憶装置とも言う(☆10))、およびそれらと様々な機器を接続する役割をするマザーボード(☆11)である。

☆ 7) Personal Computer, パーソナルコンピュータ

☆ 8) 「ED2」と記されている画像は、「情報機器と情報社会のしくみ素材集」のものを利用してもらってます：<http://www.sugilab.net/jk/joho-kiki/>

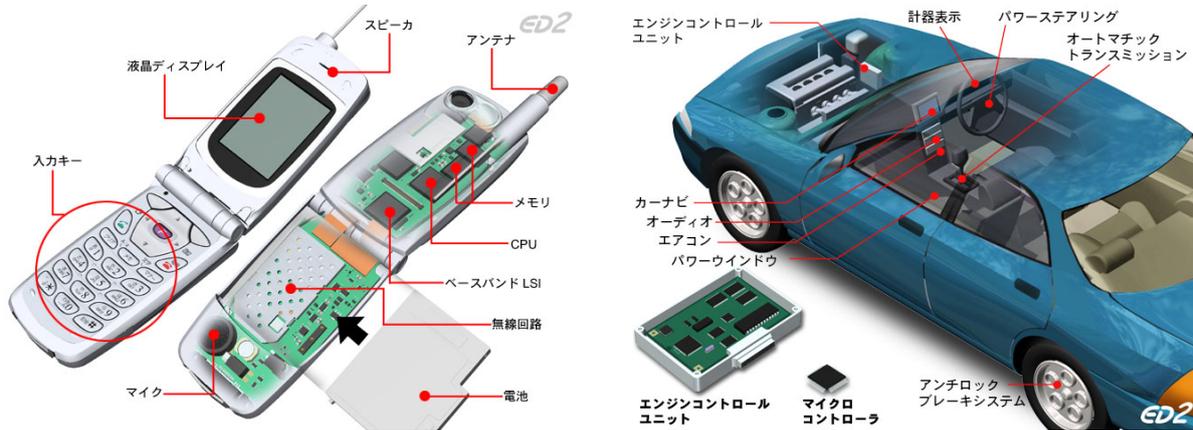
☆ 9) Central Processing Unit. 中央演算処理装置とも。単にプロセッサと呼ぶことも。

☆ 10) memory. メインメモリとも。

☆ 11) motherboard

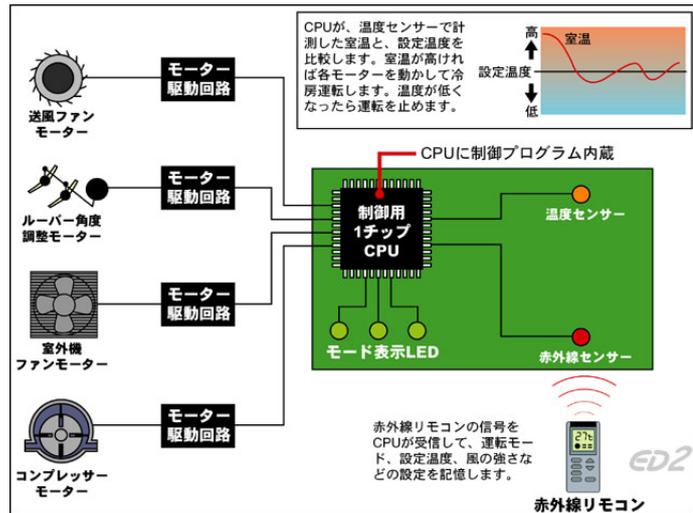
★ 4.1.2 こんなところにもコンピュータが

左下の図は、携帯電話内部の様子を示している。携帯電話やスマートフォンは、コンピュータとその周辺機器を、小さなケースの中にぎゅっと詰め込んだものといえる。図にも示されているように、やはり CPU やメモリが備わっている。



一方、右上は、自動車の例である。近年では、1 台の自動車の中には数十台ものコンピュータが搭載されている。その種類も多く、計器表示やパワーウィンドウの制御などを個別に行う単純なもの（後述の家電製品のものに近い）、カーナビのように PC やスマートフォンに近い機能をもったもの、エンジンのコントロールに特化した特殊なものなど、実に様々である。図には描かれていないが、やはりいずれも、CPU とメモリが様々な機器と接続した構成となっている。

右図は、エアコンを例に、家電製品に組み込まれたコンピュータの様子を示している。家電製品などの場合、CPU やメモリとその周辺の回路をひとまとめにした 1 つの電子部品でコンピュータができていることも多い。



★ 4.1.3 各種コンピュータに共通の仕組み

右図は、コンピュータの仕組みを、情報の流れに注目して描いたものである。

★★ (肝心の所。講義時に右図を使って解説します) ★★

注：I/O は Input/Output つまり入力/出力の略。インタフェース (interface) とは、inter (何かと何かの間), face (顔, 面) と分解できることからわかるように、2つ以上の何かに向き合っている間に入って、それらのなかだちをするもの。

