

今回の内容

1.1 シラバス抜粋	1-1
1.2 プログラムが動くということ	1-2
1.3 CPU	1-3
1.4 メモリ	1-8
1.5 アドレス変換機構	1-9
1.6 付録: 16 進表記	1-10
1.7 付録: Big Endian と Little Endian	1-10

1.1 シラバス抜粋

以下は、4 月 1 日現在のシラバスですが、新型コロナウイルス感染症の影響で、成績評価方法を含めて変更される可能性がありますので注意してください。

講義概要	パソコンなどの計算機システムの構成とオペレーティングシステムの仕組みについて学びます。																
到達目標	計算機を構成する様々なハードウェアがオペレーティングシステムによってどのように管理され、アプリケーションプログラムから利用できるようになっているのかを理解し、その理解をプログラミングに応用できる。																
講義方法	配布した資料に沿って講義を行います。																
系統的履修	「情報処理システム I」(旧「情報処理の基礎」)の内容が前提となります。「プログラミング及び実習Ⅲ」(旧「プログラミング及び実習」)とともに履修してください。																
成績評価の方法	期末試験(100 点満点)と提出された演習課題等で評価します。期末試験が x 点、課題の得点率が y % のとき、総合的な成績は $x + (100 - x)y/500$ 点(端数切り捨て)となります。																
講義計画	<table border="0"> <tr> <td>(1) プログラムが動くということ</td> <td>(9) ファイル管理とファイルシステム</td> </tr> <tr> <td>(2) オペレーティングシステムの役割と構成</td> <td>(10) Unix 系 OS でのファイル入出力</td> </tr> <tr> <td>(3) カーネルとシステムコール</td> <td>(11) シェルと端末エミュレータ、 ウィンドウシステム</td> </tr> <tr> <td>(4) プログラムとプロセス管理</td> <td>(12) プログラムの起動と リダイレクション</td> </tr> <tr> <td>(5) メモリ管理</td> <td></td> </tr> <tr> <td>(6) 仮想記憶システム</td> <td>(13) プロセス間通信</td> </tr> <tr> <td>(7) ユーザプロセスへのメモリ割り当て</td> <td>(14) セキュリティ</td> </tr> <tr> <td>(8) ライブラリと動的リンク</td> <td>(15) まとめ</td> </tr> </table>	(1) プログラムが動くということ	(9) ファイル管理とファイルシステム	(2) オペレーティングシステムの役割と構成	(10) Unix 系 OS でのファイル入出力	(3) カーネルとシステムコール	(11) シェルと端末エミュレータ、 ウィンドウシステム	(4) プログラムとプロセス管理	(12) プログラムの起動と リダイレクション	(5) メモリ管理		(6) 仮想記憶システム	(13) プロセス間通信	(7) ユーザプロセスへのメモリ割り当て	(14) セキュリティ	(8) ライブラリと動的リンク	(15) まとめ
(1) プログラムが動くということ	(9) ファイル管理とファイルシステム																
(2) オペレーティングシステムの役割と構成	(10) Unix 系 OS でのファイル入出力																
(3) カーネルとシステムコール	(11) シェルと端末エミュレータ、 ウィンドウシステム																
(4) プログラムとプロセス管理	(12) プログラムの起動と リダイレクション																
(5) メモリ管理																	
(6) 仮想記憶システム	(13) プロセス間通信																
(7) ユーザプロセスへのメモリ割り当て	(14) セキュリティ																
(8) ライブラリと動的リンク	(15) まとめ																
テキスト	なし。配布資料は次の Web ページから入手できます。 http://www602.math.ryukoku.ac.jp/CSys2/																
参考文献	野口健一郎、他『オペレーティングシステム 改訂 2 版』(オーム社) 3,080 円 大久保英嗣『オペレーティングシステムの基礎』(サイエンス社) 1,760 円 Andrew S. Tanenbaum『モダンオペレーティングシステム』(ピアソン・エデュケーション・ジャパン) 7,980 円																

1.2 プログラムが動くということ

この科目の受講者なら、C 言語のプログラムを書いて、そのプログラムを実行してみたことがあるはずです。たとえば、次のような C プログラムを考えてみます。

```
myprog.c
1 #include <stdio.h>
2
3 int main()
4 {
5     int x, y;
6
7     printf("x = ");
8     scanf("%d", &x);
9     printf("y = ");
10    scanf("%d", &y);
11    printf("%d + %d = %d\n", x, y, x+y);
12    return 0;
13 }
```

このような C プログラム `myprog.c` を作成して、たとえば情報実習室の Linux 環境で次のようにコンパイルすると、機械語プログラム `myprog` ができます。

```
$ cc -o myprog myprog.c
```

また、こうして作成した機械語プログラム `myprog` は、次のように起動して、`myprog.c` に書いた通りに計算機を動作させることができるわけです。

```
$ ./myprog
x = 7
y = 25
7 + 25 = 32
```

ソースプログラムを書いて、コンパイルし、実行する — この一連の手順は、これまで何度も繰り返して来たことです。ここで紹介した流れの中で起きたことには、もう何の疑問を持たないかも知れません。しかし、よくよく考えてみると、まだよく分かっていない部分が残っていることに気づきます。

`myprog.c` をコンパイルすることで、C 言語で書かれたプログラムが機械語に翻訳され、`myprog` という名前の 1 つのファイルとして SSD やハードディスクなどの補助記憶装置に記憶されたはずですが、このプログラムを CPU で実行するためには、補助記憶装置上の機械語プログラムを主記憶装置 (メモリ) にコピーしないとイケないはずですが、この仕事を行ったのは一体誰なのでしょう。この仕事も CPU が行ったはずなのですが、その仕事はやはり機械語プログラムとしてどこかに存在しないとイケないはず。そのプログラムはどこから来たのでしょうか。

また、`myprog.c` で使われていた `scanf` や `printf` という関数も、結局は機械語プログラムのはずですが、これらのプログラムは一体どこにあって、いつどのように `myprog` とつながったのでしょうか。また、`printf` を呼び出すことで、ディスプレイ上のウィンドウには文字列が表示されたわけですが、この文字列はどのようにして表示されたのでしょうか。ディスプレイ装置自体にこの文

字列を表示する仕組みが備わっているとは思えませんので、やはり何か隠れた機械語プログラムが働いていることが想像できますが、このプログラムはどこから来て、いつどのように myprog とつながったのでしょうか。

また、私達が作成した機械語プログラムに限らず、PC やスマートフォンなどでたくさんのプログラムが同時に動作しているように見えます。PC やスマートフォンには限れた数の CPU しか搭載されていないはずなのに、その数を越えるようなプログラムが同時に動作できるのはなぜでしょうか？

この科目は、これらの謎に (少しでも) 答えることを目標としています。その謎を解く最大の鍵は、オペレーティングシステムと呼ばれるソフトウェアにあります。オペレーティングシステムは、(私達が作成するような) アプリケーションプログラムと PC やスマートフォンなどのハードウェアを上手につないでくれます。この科目では、このオペレーティングシステムと呼ばれるソフトウェアについて、主にアプリケーションプログラムを作成する際の視点で勉強していきます。

メモ

1.3 CPU

オペレーティングシステムを含めて、すべての (機械語) プログラムは、**CPU** (中央処理装置¹) によって実行されていきます。オペレーティングシステムの話に入る前に、この CPU とその周辺のハードウェアの仕組みについて解説します。

私たちが作成した C プログラムはコンパイラ²の働きによって、機械語プログラムに変換されますが、この機械語プログラムを読み取って、そこに書かれた指示を実行しているのが CPU と呼ば

¹Central Processing Unit

²Linux 環境の cc コマンドなど



図1: Intel Core i7 3960X (6つのコアが内蔵されている)

れる装置です。一般のパソコンで使用されている CPU は、図1のような部品の内部に格納された、ダイ (die) と呼ばれる 10mm 四方程度の半導体 (シリコン等) の小片上に、印刷技術を用いて形成された電子回路として実現されています。1つの CPU は百億個程度の素子 (トランジスタ³) で構成されています。

図1のような部品の中には、CPU として働くことのできる電子回路が1つだけ格納されていることもあれば、複数 (2～数十個程度) 格納されていることもあります。複数ある場合、その1つ1つを CPU と呼ぶ場合もあれば、それらを格納している (図1のような) 部品のことを CPU と呼び、その中に (複数) 用意された CPU として機能するものを、それぞれコア (core) と呼んで区別する場合があります。

メモ

CPU (コア) は、1台の計算機に1個から数十個搭載され、主記憶装置 (メモリ) に記憶されている機械語プログラムを読み取り、その指示に従って、四則演算などの計算や、各装置の制御、装置間でのデータの転送を行います。

機械語プログラムは、数 bit から数十 bit の大きさの機械語命令がたくさん並んだものです。どのようなビット列でどのような作業を行うかについての約束事が CPU 毎にあらかじめ決められており、CPU はこの約束事にしたがって、順に指示されたとおりの作業を行っていきます。CPU に用意されている機械語命令の集まりを命令セットと呼びます。

³3つの電気的な端子を持っており、その2つの間を流れる電流のオン・オフ (あるいは大小) を、もう1つの端子に与える電圧で切り替えることのできる素子の総称です。パソコンの CPU には、主に MOSFET (Metal-Oxide-Semiconductor Field Effect Transistor 金属酸化半導体型電界効果トランジスタ) というトランジスタが使われています。

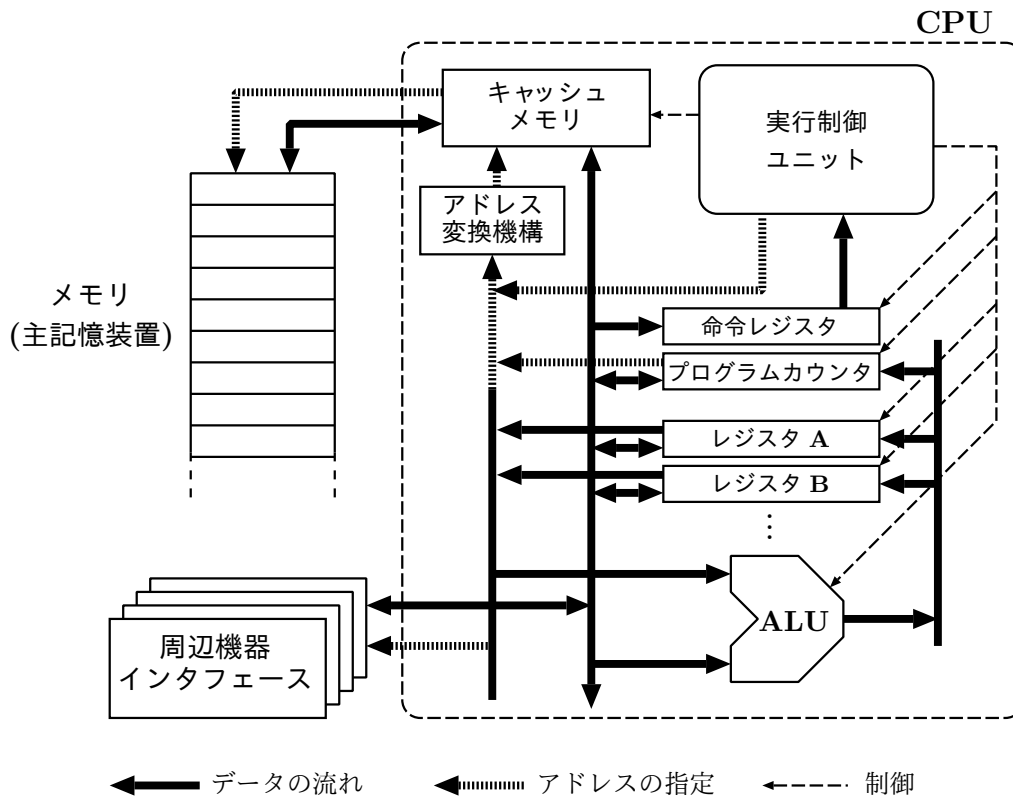


図 2: CPU の内部構成 (模式図)

CPU の内部構成

パソコン等で使用されている CPU は、おおよそ図 2 のような構成になっています。

キャッシュメモリ メモリに記憶されているデータの内、CPU が頻繁に読み書きする部分のコピーを記憶しておく装置です。主記憶装置のメモリは、通常 DRAM (ダイナミック RAM) で構成されますが、キャッシュメモリは、より高速な SRAM (スタティック RAM) で構成されます。キャッシュメモリの記憶容量は主記憶装置に比べると僅かなものです⁴が、レイテンシ (アクセスタイム)⁵がより小さく (高速に) なっています。パソコンで使用されている CPU のキャッシュメモリは、高速で容量の小さい 1 次キャッシュメモリと、それよりは若干低速で容量の大きい 2 次キャッシュメモリの 2 段構成、あるいは、さらに大容量の 3 次キャッシュまでを備えた 3 段構成となっている場合がほとんどです。

CPU がメモリ中のデータを必要とする場合、まず、このキャッシュメモリに読み込まれてから使用されます。また、メモリにデータを格納する場合、とりあえずキャッシュメモリに記憶して、その後メモリに格納されます。CPU が機械語命令を実行していく過程では、メモリ中のいろいろな場所 (アドレス) のデータにアクセスしますが、メモリ全体に全くでたらめな順番でアクセスすることは稀で、短い時間 (たとえば $10\mu\text{s}$) だけを見れば、アクセスするデータはいくつかのアドレス周辺に偏るのが普通です。たとえば、メモリ中には機械語プログラムが格納されており、これを CPU が読み取って、その指示を実行していきませんが、このとき CPU は (基本的には) 連続するア

⁴通常、主記憶は数百 MiB から数百 GiB、キャッシュメモリは数百 KiB から大きくても数十 MiB の大きさ (KiB = 1024 B、MiB = 1024² B、GiB = 1024³ B) です。

⁵対象となる情報 (の場所) を指定してから、その情報が実際に読み取られる (書き込まれる) までの時間的な遅れ

ドレスを順にアクセスしていくことになります。また、機械語プログラム以外の一般のデータに関しても、ひとまとまりのデータは連続するアドレスに置かれますから、このひとまとまりのデータを処理する場合にも、CPU がアクセスするアドレスは集中します。このため、頻繁にアクセスするアドレスのデータをキャッシュメモリにまとめてコピーしておき、直接メモリにアクセスする代わりにキャッシュメモリにアクセスし、キャッシュメモリの内容が変更された場合は、適宜、その内容をメモリに書き戻してやれば、全体としてデータの読み書きが高速化できます。

メモ

プログラムカウンタ CPU が次に実行する機械語命令を読み取るメモリアドレスを記憶します。パソコン等で使用される CPU は、通常、メモリアドレスを 32 bit あるいは 64 bit の符号なし整数で表現しますので、このような CPU の「プログラムカウンタ」は 32 bit あるいは 64 bit 長のデータを記憶します。通常は、機械語命令が 1 つ実行される度に、その機械語命令のビット列の長さに相当する分だけプログラムカウンタの値が増加して、次の機械語命令の場所を指すようになります。ただし、分岐命令が実行されると、その分岐命令によって指定されたアドレスがプログラムカウンタに書き込まれて、CPU はそのアドレスから機械語命令の実行を続けます。

メモ

命令レジスタ CPU が次に実行する機械語命令 (ビット列) を保持する働きを持ちます。「プログラムカウンタ」で指示されたアドレスのデータが、メモリから (キャッシュメモリを介して) この「命令レジスタ」に読み込まれ、「実行制御ユニット」によって機械語命令として解釈されて実行されます。レジスタという名前が付いていますが、CPU が機能するための仕組みの一部であって、通常、「命令レジスタ」の値を機械語プログラムから参照したり変更したりすることはできません。

メモ

実行制御ユニット「命令レジスタ」に読み込まれた機械語命令を解読して、そこに指示されている作業を実行します。ビット列として表現された機械語命令を解析して、CPUの行うべき仕事を決定する作業のことを、命令のデコード (decode) と呼びます。1つの機械語命令は、デコードされることによってマイクロ命令と呼ばれるいくつかの細かい手順に分割され、「実行制御ユニット」がCPUを構成している各部に順に指令を出すことで、そのマイクロ命令がCPUのクロック信号に同期して実行されていきます。1つの機械語命令は、数クロック⁶から数十クロック⁷の時間をかけて実行されますが、1つの機械語命令を分割してできた一連のマイクロ命令の実行がすべて終わらなくても、次の機械語命令のマイクロ命令の実行を始めることが可能な場合がほとんどなので、CPUの中では複数の機械語命令が時間的に少しずつずらされながら同時並行的に実行されていくことになります。このような機械語命令の並行処理はパイプライン処理と呼ばれます。



レジスタ 計算に使うためにメモリから読み込まれたデータや、複雑な計算の途中結果などを記憶します。通常、1つのCPUに、数個から数百個のレジスタが用意されています。一般的な用途に使用される汎用レジスタと呼ばれるレジスタもあれば、浮動小数点数値データ専用のレジスタや、アドレスの処理専用のレジスタなど、特定の用途専用のレジスタもあります。パソコンで使われるCPUの汎用レジスタやアドレス処理用のレジスタは、それぞれ32 bitあるいは64 bit長のデータを記憶することができるのが普通ですが、家電製品などに組み込まれる場合には、8 bit長など、もっと短いレジスタを持つCPUが使われます。

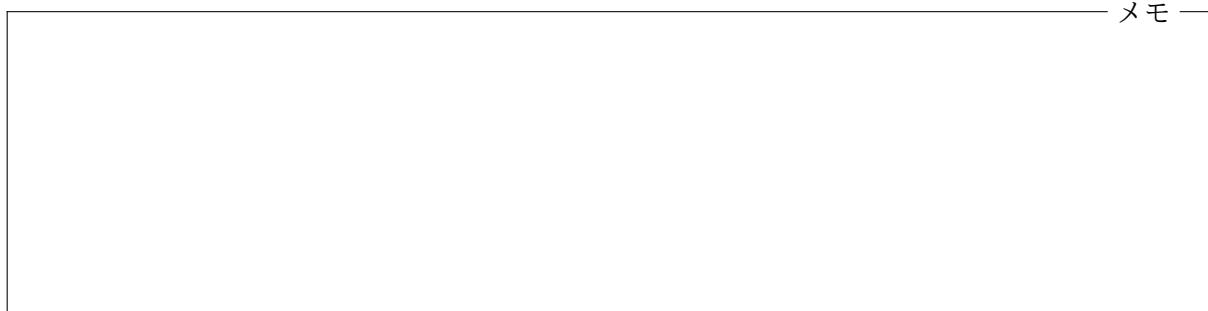
CPUは、ビット列として表現されたいろいろなデータを扱うことができますが、この中で、そのCPUが最も自然に扱うことのできる大きさのデータを、そのCPUのワード (word) と呼びます。通常、CPUの1ワードの大きさは、1つの汎用レジスタに格納できるデータの大きさとなっています。



⁶クロック信号の周期の長さ(時間)を1クロックと呼びます。たとえば2GHz (2×10^9 Hz)のクロック信号の場合、1クロックは0.5 ns (0.5×10^{-9} 秒)となります。

⁷ごく稀には、さら長い時間を必要とする命令も存在します。

ALU ALUは算術・論理ユニット(Arithmetical Logical Unit)の略称で、符号付き、あるいは符号なしの整数データの四則演算、ビット演算(ビット毎の論理演算やビットシフト、ローテート)、浮動小数点数値データの四則演算などの計算を行います。CPUによっては、浮動小数点数値データの平方根や指数関数、対数関数、三角関数の計算を行えるものもあります。



1.4 メモリ

CPUが実行していく機械語命令やCPUが処理するデータは、すべてビット列としてメモリ(主記憶装置)に記憶されます。メモリは、数個から十数個のメモリチップを装着したメモリモジュールと呼ばれる部品(図3⁸)数枚で構成されています。

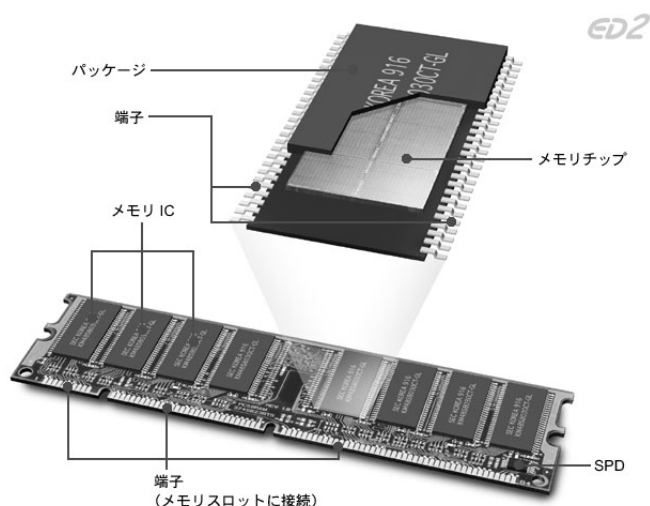


図3: メモリモジュール

アドレス	
0番地	11000101
1番地	01101001
2番地	00110000
3番地	10100110
4番地	10111110
5番地	01010001
6番地	11011110
7番地	01110111
8番地	01011101
⋮	⋮
	01000011

図4: メモリアドレス

これをCPUの側から見ると、図4のように1つの欄に1 byte (8 bit)のデータ⁹を記憶することのできる長い長い1列の表のように見えます。メモリの欄を区別するために、各欄にはアドレス(番地)と呼ばれる非負の整数が振られており、CPUは、このアドレスを指定してメモリ中のデータの読み書きを行います。アドレスは1つの欄(1 byte)ごとに振られるのが普通ですが、CPUの1ワードごとに振られることもあります。前者のアドレスの振り方をバイトアドレッシング(byte addressing)、後者をワードアドレッシング(word addressing)と呼びます。

⁸ED2と記された図は「情報機器と情報社会の仕組み素材集(<http://www.kayoo.org/mext/joho-kiki/>)」の一部を利用して頂いたものです。

⁹CPUによっては、1つの欄に格納できるデータの大きさが1ワードとなっている場合もあります。

機械語命令のレベルでは、CPU はメモリ中に記憶されたデータに、1 byte から十数 byte くらいの大さを単位としてアクセスすることになりますが、この時実際にアクセスされるのはキャッシュ中のデータです。キャッシュメモリとメモリの間では、より効率的にデータを転送するために、もっと大きなサイズ(たとえば 64 byte)を単位としてメモリの読み書きが行われます。しかし、キャッシュの働きで CPU の動作が高速化されるという点を除けば、このことが機械語プログラムの動作に影響を与えることはありません。CPU は、あくまで機械語命令で指定されたメモリアドレスに格納されたデータを直接読み書きしていると考えたときと同じ動作を行います。



1.5 アドレス変換機構

機械語命令の指示に従って CPU がメモリにアクセスするには、その機械語命令で指定されたアドレスが、そのままメモリモジュールに伝わるとは限りません。パソコンやスマートフォンなどに使われる CPU では、複数の(機械語)プログラムが同時に実行されますが、このとき使用されるメモリに競合が起きないようにするために、各プログラムが使用しているアドレスを、それぞれ異なるアドレスに変換してメモリモジュールに伝えるためのアドレス変換機構が内蔵されています。

各プログラムが使用する変換前のアドレスを論理アドレスと呼び、変換機構によって変換されてメモリモジュールに伝達される変換後のアドレスのことを物理アドレスと呼びます。ただし、どの論理アドレスもアドレス変換機構によって何らかの物理アドレスに変換されるとは限りません。どの物理アドレスにも対応していない論理アドレスもあります。そのような論理アドレスが使用された場合に、この科目で勉強するオペレーティングシステムの働きによって、臨機応変にアドレス変換機構の設定を変更し、何らかの物理アドレスを割り当てる場合があります¹⁰。このような仕組みがある場合には、変換前の論理アドレスは仮想アドレスと呼ばれます。この科目では、論理アドレスという用語は用いず、すべて仮想アドレスと呼ぶことにします。

¹⁰このような仕組みは仮想記憶システムと呼ばれます。

1.6 付録: 16進表記

デジタル計算機では、そこで扱う情報をすべてビット列 (0/1 の列) として表現しますので、機械語プログラムや計算機アーキテクチャの話をする際には数十桁の2進数が頻繁に登場することになります。このような2進表記は人間にとっては扱い難いので、下位の桁から4桁 (4 bit) ずつひとまとめにして、その4 bit のビットパターンを下の表の16種類の記号でそれぞれ表し、その記号を同じ順に並べて、全体を16進表記で表すことがよく行われます。

たとえば、0010 1110 というビットパターン (2進表記) を 2e と、1100 0101 0001 0110 1101 1010 1000 0111 というビットパターン (2進表記) を c516da87 と16進表記します。また、16進表記であること明示するために、それぞれ、0x2e や 0xc516da87 のように先頭に 0x (あるいは 0X) を付けたり、2eh や c516da87h のように末尾に h (あるいは H) を付ける慣習もあります。

ビットパターン	記号
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	a
1011	b
1100	c
1101	d
1110	e
1111	f

この表では、1010 から 1111 までのビットパターンを英小文字 a から f に対応させていますが、小文字の代りに大文字 A から F を使用することもあります。4桁の2進表記を10進表記した時、0から9までは、そのままの数字で、10から15までは、a から f までの英文字で表していることに注意してください。

1.7 付録: Big Endian と Little Endian

たとえば、CPU が 32 bit 長のデータを、メモリ中に (バイトアドレッシング) で書き込む場合には、4つの連続するアドレスにアクセスすることになります。この時、32 bit 長のデータは4つの 8 bit 長のデータとして、メモリに格納されることになりますが、これら4つのデータを、それぞれのアドレスに対応させて格納するかについては2つの方法が存在します。多バイト長のデー

データをメモリに格納する際に、上位バイトを下位アドレスに対応させる格納方法をビッグエンディアン (big endian) と呼び、下位バイトを下位アドレスに対応させる格納方法をリトルエンディアン (little endian) と呼びます。この2つのどちらの格納方法を採用するかは CPU ごとに異なります。この2つのいずれかを選択することのできる CPU もあります。

32 bit 長のデータ 0x12345678 を 0x00010000 番地に格納した例

アドレス	Big Endian	Little Endian
⋮	⋮	⋮
0x0000ffff		
0x00010000	0x12	0x78
0x00010001	0x34	0x56
0x00010002	0x56	0x34
0x00010003	0x78	0x12
0x00010004	⋮	⋮
⋮		

図 5: 多バイト長データのメモリへの格納法の違い