第3章 エンディアン，アラインメント，ソフト・マクロのCPUコア，Linuxデバイス・ドライバ

ハードウェアを意識したプログラミングの基礎

デバイス・ドライバを作るためには，まずハードウェアをアクセスする手法を知らなければならない．エンディアンやアラインメントを意識したり，CPUのバージョンによる命令の違いなどを考慮する必要がある．さらに最近では，命令そのものを追加できるソフト・マクロのCPUコアなども登場している．そこで，ここではCPUとデバイス(メモリ)の間のエンディアンについて説明する．
(編集部)

荘司 靖

本稿では，筆者がこれまでに行ったLinuxを用いた開発の中で得た経験を元に，Linuxなどのデバイス・ドライバを開発・移植するときにハマりやすい点を紹介したいと思います．内容は大きく分けて，以下の四つになります．

- エンディアン
- I/Oアクセス
- ハードウェア，CPU，コンパイラ
- アラインメント

特に断りがなければ，ここではポインタ・サイズが32ビットのCPUを対象にします．具体的には，x86，ARM，MicroBlaze，PowerPCの四つのCPUを中心に進めていきますが，特にそれぞれのCPUについて知らなくても問題ないように説明しています．

1. ビッグかリトルか，それが問題だ

デバイス・ドライバをいろいろなアーキテクチャに移植するときに，まず問題になるところと言えばエンディアンです．エンディアン(またはバイト・オーダ)はメモリやネットワークにデータを配置する際の並び順です．

図1のようにバイト単位で読み込みを行う場合，レジスタに書かれる値は読み込んだメモリの値になります．しかし一度に複数バイトを読み込む場合は，**図2**のように2通り[注1]の方法があり，どちらかを選択しなければなりません．**図1**の左側の方法をリトル・エンディアン，右側の方法をビッグ・エンディアンと呼びます．

エンディアンは基本的に，CPUの設計段階でCPUの設計者が決定します．Intel社はx86を設計するときにリトル・エンディアンを選択しました．一方，Xilinx社はMicroBlazeを設計するときにビッグ・エンディアンを選択しました(**図3**)．

エンディアンの選択をユーザに任せているCPUもあります．ARMとPowerPCはバイ・エンディアンと呼ばれ，設定によってどちらのエンディアンにもなれるように設計されています．と言っても，実行中にポンポンとエンディ

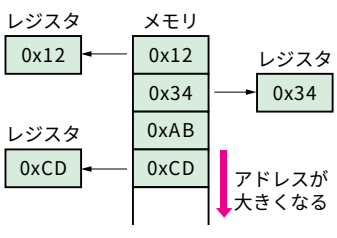

図1
バイト単位で読み込みを行う場合

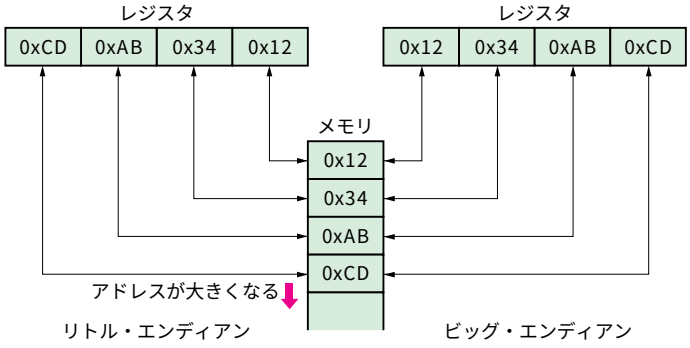

図2　リトル・エンディアンとビッグ・エンディアン

注1：PDP11で使われたミドル・エンディアンというのもあるが，最近では使われていないので説明を割愛する．

KEYWORD ―エンディアン，リトル・エンディアン，ビッグ・エンディアン，アラインメント，バイト・アクセス，ソフト・マクロのCPUコア，ARMv4命令，コンパイラ，構造体のパック

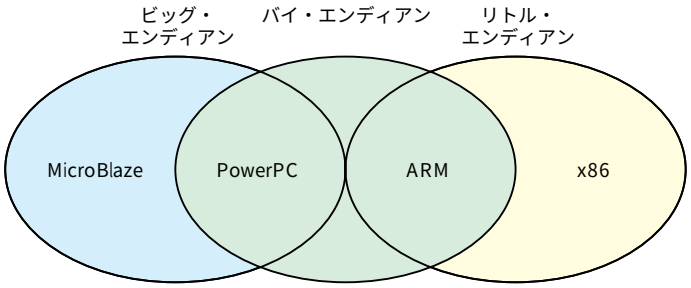

図3　CPUとエンディアンの関係

リスト1　どちらのエンディアンか

```
 1: #include <stdio.h>
 2: #include <linux/types.h>
 3:
 4: int main()
 5: {
 6:   char fifo[] = {0x12, 0x34, 0xAB, 0xCD};
 7:   __u32 val;
 8:
 9:   val = *(__u32 *)fifo;
10:
11:   printf("0x%X¥n", val);
12:   return 0;
13: }
```

アンを変更するわけではなく，起動時にどちらかに設定するのが一般的です．ARMではリトル・エンディアンが，PowerPCではビッグ・エンディアンが多く選択されているようです．

● 0x1234ABCDか0xCDAB3412か

それでは実際の例を示し，どんな時にエンディアンが問題になるかを見ていきましょう．

仮に，あるデバイスのFIFOから32ビットのデータを読み込まなければならないとします．このFIFOからデータを受け取るために，x86のデバイス・ドライバ・プログラマは以下のようなコードを書きました．

```
u32 val = *(u32 *)FIFO_ADDR;
```

どうやら，このデバイス・ドライバはx86上では正しく動いたようです．しかし，このコードをそのままビッグ・エンディアンに設定したPowerPCに移植したところ，思ったように動かなくなりました．

この問題を簡単にアプリケーション（C言語のソース・コード）で表したものが**リスト1**です．

2行目のヘッダは，コードの見た目をLinuxのデバイス・ドライバに似せるために使っています[注2]．デバイス・ドライバ内で使える「u32」という型は，ユーザ・ランドではPOSIXのネーム・スペースを侵害してしまいます．そこで「__」を先頭に付けてネーム・スペースを汚さないように変更したものが，アプリケーションから使えるようになっています．u32はUnsigned（符号なし）の32ビット変数という意味です．

6行目では，FIFOの代わりにスタック上にバッファをとっています．初期値は，アドレスの小さい方から0x12 0x34 0xAB 0xCDとなるようにしました．

9行目で，FIFOから値を読み出して，変数valに代入します．そして読み出した値をprintf()を使って出力しています．

本当のデバイス・ドライバであれば，受け取ったデータを処理したり，初期化関数内でデバイスIDを読み出して比較したり（この場合，FIFOではなくデバイスのレジスタだが）する場合に似たようなことを行います．さて，上記のコードをx86上でコンパイルして実行すると，

```
$ ./a.out
0xCDAB3412
```

と表示されます．これはx86がリトル・エンディアンだからです．それではPowerPCで実行するとどうなるのでしょうか？

```
$ ./a.out
0x1234ABCD
```

となりました．x86と違い，PowerPCはビッグ・エンディアンなので，メモリに並んだように値が読めます．混乱した人はもう一度，**図2**と見比べてみてください．

● マクロle32_to_cpu

さて，普通に読み込むと値が異なることは分かりました．しかし，このままではデバイス・ドライバを移植することがとても面倒です．Linuxではこの問題を解決するために，エンディアンによるバイト・オーダの違いを吸収するマクロを用意しています．マクロはエンディアンごとに，

```
include/linux/byteorder/little_endian.h
include/linux/byteorder/big_endian.h
```

で定義されています．リトル・エンディアンで32ビット書き込みを行う場合はcpu_to_le32()が，読み込みの場合はle32_to_cpu()が使えます．このほかにも，ビッグ・エンディアン用のbe32_to_cpu()とcpu_to_

注2：カーネルのヘッダ・ファイルをアプリケーションでインクルードするのはあまり良くない．自分の責任で使用すること．