教科書には載っていない！

現場で役立つ プログラミングのちょい技

第10回　データ型にまつわる移植性を高めるためのちょい技

邑中 雅樹

CPUと外部メモリの間のバス幅に注目して，バス·アクセスを効率良く行えば処理時間を減らすことができます．しかし，バス幅を指定したデータ型が思わぬバグを引き起こすことがあります．

今回は，適切なデータ型で実行時間を短くしつつ移植性も良くするデータ型の使い方を説明します．さらに，その周辺に潜む落とし穴について，mrubyで実際に起こったバグを例に解説します．（編集部）

前号（2012年9月号 pp.100-104）では，プリプロセッサ・マクロを利用して配列に割り当てるメモリ容量を削減するちょい技を紹介しました．前号を見逃したり，忘れてしまったりした読者のために，ソース・コードを**リスト1**に再掲します．配列の要素数が0から15までのときは4ビットで値を表現できるため，char型（8ビット）の中に二つの要素を入れるマクロが有効に作用しました．

具体的な使い方は“i = table[n]”の代わりに“i = table_get(n)”とし，“table[n] = i”の代わりに“table_put(n, i)”とします．結果として，配列の“table[]”の専有するメモリ・サイズはおおむね半分になります．思い出していただけたでしょうか．

今回は，前回のメモリ容量を削減する方法に加えて，さらに実行時間を短くして，移植性も良くする最適化について考えてみます．

外部メモリ・アクセスとバス幅の関係

外部メモリへアクセスを減らせば，アクセス時間を節約できる

どのCPUアーキテクチャでも，おおむねCPUのレジスタへのアクセス時間が最速です．そして，キャッシュがあればキャッシュ，続いて内部バスに接続されたメモリ，さらに外部バスで接続されたメモリの順でアクセス時間がかかります．つまり，外部バスに接続されているメモリ・アクセスを減らせば実行時間が節約できます．

外部メモリへのアクセスは，データ・バス幅に合わせると効率が良くなる

外部バスに接続されたメモリの場合は，メモリのデータ・バス幅が1回のアクセスで取得できる最大のバイト数になります．ほとんどのCPUアーキテクチャでは，外部バスに接続されたメモリに対して1バイトを読み込むのも，データ・バスの最大幅（複数バイト）を読み込むのも，かかる時間は同じです．

具体的な例として，**リスト2**の場合を考えてみましょう．access_4bytesは，unsigned int *…で4バイトを取得しています．一方，access_1byteではchar *（1バイト）でアクセスしています．ARM用のgccでコンパイルした結果（一部抜粋）は，**リスト3**のようになりました．

リスト3のldr，ldrbはメモリからレジスタへの読み込み命令で，str，strbはレジスタからメモリへの書き込み命令です．共にバス・アクセス時間がかかります．つまり，これらの命令は少ないほうがバス・アクセスの時間は短く，結果として実行時間が短くなります．access_4bytesとaccess_1byteのどちらが早いのかは，これらの命令を数えれば一目瞭然です．

実際のアプリケーションでは，コンパイラの最適化などで，今回の例のようなあからさまな違いが出るとは限りません．しかしながら，データ・バスの幅を意識することで，コンパイラにはできない最適化を施せる傾向はあります．

リスト1　再掲載：配列の各要素に二つの数値を入れるマクロ

```
unsigned char table[MAP_TABLE_SIZE / 2];

#define table_get(index) do { ((table[index / 2] >> (4 * index % 2)) & 0x0fU) } while(0);
#define table_put(index, value) do { (table[index / 2] = table[index / 2] & ~(0x0fU << (4 * index % 2) | ((value & 0x0fU) << (4 *
index % 2)) } while(0);
```