≫ 文法の曖昧さを理解して確実性と再利用性を高める

マイコンC言語 転ばぬ先のつえ

第24回　最適化の正しい活用③…変数に対する最適化の抑止

鹿取 祐二

リスト1　volatile型修飾子の有無で速度が変わる例

```
int total;

void main(void)
{
/* volatile */ int i, work=0;

    for( i=1 ; i<=100 ; i++ )
        work += i;
    total = work;
}
```

（a）ソースコード

```
オフセット  コード   ラベル  命令
00000000 306400       movw    ax, #0x0064
00000003 04FFFF @1_1  addw    ax, #0xFFFF
00000006 DF00         bnz     $@1_1
00000008 30BA13       movw    ax, #0x13BA
0000000B BF0000       movw    !LOWW(_total), ax
0000000F D7           ret
```

（b）コンパイル結果

第22回（2023年5月号）から，処理系が持つ最適化機構（以降，最適化と記す）について紹介しています．最適化はプログラムのブラッシュアップを行う反面，バグの原因にもなり得ます．ここでは過剰な最適化によるバグの発生を防ぐ方法について解説します．（編集部）

前回（第23回，2023年6月号）の解説で，最適化はvolatile型修飾子で抑止できることが理解できたと思います．また，その対象はほとんどの場合，周辺機能のレジスタであることも理解できたと思います．

それでは逆にvolatile型修飾子を変数に対して指定し，変数に対する最適化を抑止する必要はあるのでしょうか，それともないのでしょうか．今回は変数に対する最適化抑止の有無を紹介します．

なお，変数は大別すると関数内部に宣言された局所変数と関数外部に宣言された大域変数に分かれます．また，static記憶クラスの局所変数は大域変数と同じ性質を持つことになります．詳しくは次回紹介しますので今は単純な局所変数と大域変数で考えてください．ここでは先に局所変数に対する最適化抑止の有無から紹介します

表1　局所変数に対する最適化有無の性能

volatileの有無	RL78		RX	
	なし	あり	なし	あり
サイズ［バイト］	14	33	17	38
速度［クロック］	500	1613	401	1611

変数に対する最適化の抑止①…局所変数

● 局所変数にはvolatileは不要

結論から言えば，局所変数に対してvolatile型修飾子を指定する必要はありません．理由は簡単であり，局所変数はそれが宣言された関数内部でしか使用できないからです．実引数でアドレスを渡さない限り，他の関数からは操作できません．言い換えれば，変数が知らぬ間に変化したり，変数の変化によって影響を受けたりする他の関数は存在しないのです．あくまでも変数に対する操作は全て目的の関数内に記述されているため，極限まで最適化を施しても問題となることはありません．

● 局所変数にvolatileを指定すると性能が劣化する

逆に局所変数にvolatile型修飾子を指定してしまうと著しく性能が劣化する恐れがあります．局所変数はregister記憶クラスが指定可能であり，型や個数は処理系により左右されますが，高速にアクセス可能なCPU内部レジスタに割り付けることができます．また，近年の処理系はregister記憶クラスを指定しなくても，可能な限り局所変数をCPU内部レジスタに割り付けようとします．それは局所変数が頻繁に利用されるからです．もし，それをvolatile型修飾子で抑止してしまったら，割り当て場所としてはスタック領域が使われてしまいます．CPU内部レジスタとスタック領域（RAMのメモリ領域）では圧倒的な性能差がありますから，スタック領域では開発者の望む性能は得られません．

例えば，**リスト1**に示すような簡単な関数であって

第9回　派生型④…移植性は無いが可読性バツグン！ビット・フィールド（2021年11月号）
第10回　演算子①…「関数呼び出し」に使う小括弧（2021年12月号）
第11回　演算子②…［ ］と＊の意味，++aとa++の違い（2022年1月号）