≫ 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23回　最適化の正しい活用②…周辺レジスタの読み書きに必須! 最適化の抑止

鹿取 祐二

リスト1　volatile型修飾子による最適化抑止①…アクセス・サイズを厳守させる
16ビット・サイズの変数操作をするプログラムの例．volatile型修飾子を使うことで，変数aは宣言時のワード・サイズ（16ビット）でアクセスされている

```
volatile unsigned short a;

void main(void)
{
    a &= 0x00FF;
}
```

（a）ソースコード

```
《オフセット》《コード》      《命令》
00000000 AF0000        movw    ax, !LOWW(_a)
00000003 F1            clrb    a
00000004 BF0000        movw    !LOWW(_a), ax
```

（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．ここでは最適化に関する基本的な内容を解説します．　（編集部）

前回（第22回，2023年5月号）で紹介したいろいろな最適化は，ほとんどのものがvolatile型修飾子で抑止できます．ここではvolatile型修飾子によって最適化が抑止された例を解説します．

38 最適化の抑止①… アクセス・サイズの厳守

●アクセス・サイズが変わると困るケース

前回の記事では，処理内容に応じて宣言時とは異なるサイズで変数にアクセスする可能性があることを紹介しました．その対象が単なる変数であれば，この最適化により問題が生じる可能性はほとんどありません[注1]．

しかし，その対象が周辺機能のレジスタの場合は，問題が発生する可能性が高くなります．周辺機能のレジスタには，第9回（2021年11月号）で紹介したアクセス・サイズの制約を持つものがあるからです．

● 対策…volatile型修飾子を使う

対策も第9回で紹介した通りです．volatile型修飾子を使うことでアクセス・サイズは宣言時の変数サイズで行われます．前回の例もvolatile型修飾子を使えば**リスト1**のような結果に変化します．

変数aのアクセス命令がmovw命令のワード・サイズになっています．変数aの内容は，いったんワード・サイズで汎用レジスタのaxに読み込み，上位バイトを格納した汎用レジスタのaをゼロ・クリアした後，再び変数aにワード・サイズで書き込みます．

このようにアクセス・サイズはvolatile型修飾子を利用することで最適化を抑止でき，変数宣言時のサイズで操作されることが保証されます．

39 最適化の抑止②…周辺機能レジスタへの書き込み/読み込み処理

● volatile型修飾子がなくても結果が変わる例

関数呼び出しなどで，対象の変数が変化する可能性がある場合は，volatile型修飾子がなくても結果が変化することがあります．

例えば，同一の大域変数に連続的に値を書き込む**リスト2**のような場合であれば，書き込み処理は削除されません．変数bは，処理の途中で呼び出したsub関数によって更新されているかもしれないので，書き込み処理を削除しなかったのです．

同様のことは無駄な読み込み処理の削除にも当てはまります．例えば，**リスト3**のような場合であれば，読み込み処理は削除されません．while文の繰り返し処理は，ラベル@1_1と@1_3の間です．繰り返し処理の中で変数cの値を読み込み，0であるかをbz命令で判断しています．こちらも同様に，while文の中で呼び出したsub関数によって変数cが書き換えられる可能性があるので，読み込み処理を削除しなかったのです．

注1：バス幅固定の空間に接続されたメモリでなければ問題は発生しない．