

関西経済連合会  
組込みソフト産業推進会議

組込みソフトウェアエンジニア  
指導者養成講座  
テキストサンプル

南角 茂樹  
長濱 美保

# volatile修飾子に関する事例

並列に実行される処理(タスクまたは割り込み処理)が、同じ共有変数の異なるビットをセットする。

ほとんどの場合2つの並列処理が少なくとも1回ずつ走った後は両方のビットがセットされているが、どちらかのビットしかセットされていないケースも存在する。

それがどのような場合か考えよ。

なお、この現象はプロセッサや最適化のレベルによっては発生しない。

例えば右の例では、両方の並列処理が1回ずつ動いた後flagの値は0x3になるはずであるが、0x1または0x2になる場合がある。

```
volatile unsigned long flag = 0x0;
```

```
並列処置_1( )
```

```
{
```

```
...
```

```
flag |= 0x1; /* OR処理 */
```

```
...
```

```
}
```

```
並列処理_2( )
```

```
{
```

```
...
```

```
flag |= 0x2; /* OR処理 */
```

```
...
```

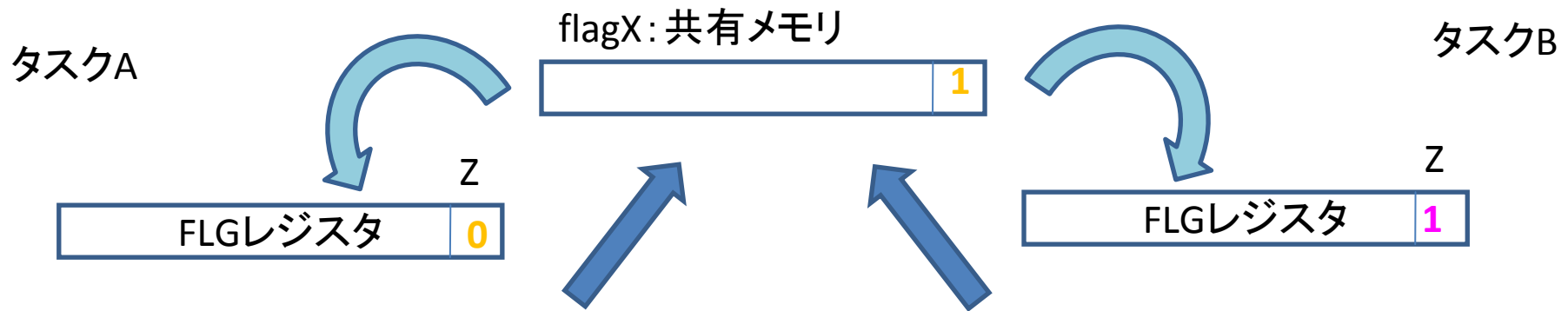
```
}
```

## 解説: コンパイラによる最適化の禁止 (volatile修飾子)

コンパイラはプログラム処理上意味のない命令については最適化を行い、不要な命令コードを生成しない。

しかしデータによっては、プログラム実行中に他の処理（タスクや割り込み、あるいはハードウェア）によって変化する変数もあり、このような変数にアクセスする命令が最適化されると困ることになる。

そのため、このようなデータは宣言時に「**volatile**」を記述することにより、コンパイラによる最適化を禁止しておくことが必要である。



**【 TAS命令 】**

^(flagX & 0x01) をタスクごとに持つ  
 Zフラグに反映して(return)  
 flgX = 0x01 する

```
loop: tas, &flagX
      jnz loop:
```

～ 以下 CS処理 ～

```
loop: tas, &flagX
      jnz loop:
```

～ 以下 CS処理 ～

flagXがnot 0 のときloop  
 だからタスクBはflagXは0だからloopしない  
 しかし、タスクAの方はflagXは1だからloopする

# オーバーフロー,アンダーフローに関する事例

あるプログラムの一部に右のようなコードがある。これは共有変数であるCntの値を1加算したり(Inc関数), 減算(Dec関数)したりするものである。

また, 1の加減算だけでなく任意の整数の加減算を行う関数(Add関数やSub関数)もある。

このプログラムを実行する中で, ある時点にくると, Add関数やSub関数は正しく動作しない場合があった。それはどのような場合かを考えよ。

なお, このケースでは各関数が同時に呼び出されることはないものとする。

※解説は本講座にて行います。

```
#include <limits.h>
static unsigned int Cnt = 0;
void Inc() {
    if (Cnt < UINT_MAX) Cnt++;
}
void Dec() {
    if (Cnt > 0) Cnt--;
}

void Add(unsigned int n) {
    if (Cnt + n < UINT_MAX) Cnt += n;
}
Sub(unsigned int n) {
    if (Cnt - n >= 0) Cnt -= n;
}
```