【組込みシステム編】第１回 パイプラインストール

こんにちは、南角です。
前回に続き、「組込みシステム編」ということで 6 回ほどの予定で記事を掲載することになりました。引き続きよろしくお願いします。

最初はまず肩慣らしに、復習から始めましょう。
今までお話ししてきたように

- 組み込みシステムは現実世界の変化に一定時間以内に対応しなければならない
- 現実世界の変化は同時多発的に発生する
- 変化の種類には重要度がある、言い換えるとはすぐに対応しなければならない変化も、後回しにしてよい変化もある

組み込みシステム側も並行処理をできるように構成しておけば、上記の実現を効率よく行うことが出来ます。
並行処理を行うための実行単位としては割り込みハンドラ（あるいは割り込みサービスルーチン）とタスクがあります。
それぞれの特徴は前回の連載第 5 回“並行処理とソフトウェアによる排他制御”に書きました。
そして並行処理があるところには、排他制御が必要です。
例えば，、並行処理を行う 2 つのタスクが初期値 0 の flag のビット 0 とビット 1 を OR するのに、2 つの処理が実行された後でもその値が 0x03 にならない場合があるという話も上記に書きましたが、ある意味排他制御が必要なもっとも粒度が小さな例と言えます。

この問題はロード・ストア アーキテクチャの RISC 系 CPU で発生しやすいのも特徴です。ロード・ストア アーキテクチャとは、時間のかかるメモリとの直接演算はやめて、メモリモジュールとの通信はメモリの内容を CPU のレジスタに持ってくるだけのロード、レジスタの内容をメモリにコピーするだけのストアに限っているアーキテクチャです。
これはパイプラインストールを防ぐ（減らす）ためです。
RISC 系においてはパイプラインが重要であり（現在では CISC 系においても当たり前の技術ですが・・・）そのためには大胆な言い方をすると個々の命令がなるべくなるべく、同じ時間で実行できることが重要であり，、時間のかかるメモリアクセスに関しては一番単純な内部レジスタとメモリ間のロードとストアだけに限定して、演算はレジスタにある変数にのみに限定しているわけです。
そのためメモリにおかれた変数との OR なども CISC 系の CPU であれば 1 命令で実行できたものが、RISC 系では

- 変数の値をレジスタに持ってくる（ロード）
- レジスタ上に持ってきた変数と OR 演算を行う
- 算を行った結果のレジスタの値を変数に書き戻す（ストア）

と複数命令となるため、メモリ上の変数に対する演算は、その変数に対して複数の並行処理がアクセスする場合は排他制御が必要になることが原因でした。

上記とパイプラインの関係でさらにもう一つ問題があります。
ロード命令でメモリ上の変数をレジスタに持ってきても、ロード命令のすぐ次の命令でそのレジストを対象として演算を行おうとしても、パイプラインで命令の実行を分割している関係でレジスタに値を代入するステップとレジスタに演算するステップの構成によっては順番が逆になる場合もあります。
例えば古典的で典型的な 5 段のパイプランの命令実行のステップは以下のようになります。
つまり 5 個の命令が同時に実行されるわけです。

- IF : メモリ（命令キャッシュ）から命令を読みだす（フェッチする）
- ID : 命令をデコードしながら（制御部で制御信号を生成しながら）、レジスタを読み出す
- EX : 命令操作の実行または（データや分岐先の）アドレスの生成を行う
- MEM : データ･メモリ中のオペランドにアクセスする
- WB : 結果をレジスタに書き込む

レジスタからの読み出しより書き込みが後のステップでなされていることがわかります。
以下の疑似アセンブラの命令はそのまま実行すると OR 命令で使用する reg1 はまだ X の値がコピーされる前の古い値ということです。

LD X、reg1
OR reg1、0x01

これを防止するために以下のような処理のどれかを実施します。

- LD 命令と OR 命令の間でパイプラインの実行をハードウェア的に調整する
- NOP 命令を挿入するか
- 前の方におかれていて位置を移動しても、その処理内では問題ない命令をこの間に移動する

一つ目のやりかたは（機構）パイプラインインターロックと呼びますが、CPU ハードウェアが複雑になります。
二つ目、三つ目はソフトウェア、特にコンパイラに任せることが出来ますが、二つ目の方法は命令数、実行時間の増加を招くためパイプラインインターロック機能のない CPU の場合、コンパイラはまず 3 つ目の方法を適用しようとします。
3 つ目の方法は命令のリ・オーダーまたは命令の再配置と呼ばれます。

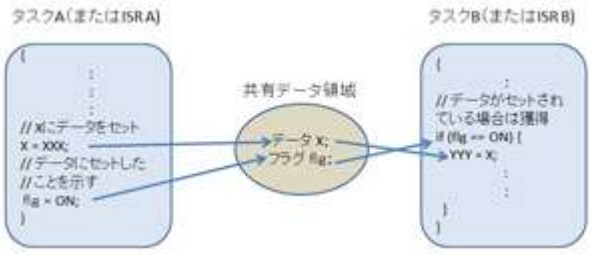
この場合の問題点はコンパイラ（ここでは C または C++コンパイラに話を絞ります）が並列処理を認識していないことです。つまり自分が操作しようとしているある変数を別の人が書き換えるかもしれないとは全く認識していません。

ところで、パイプラインで似たような問題は命令の実行順番が変化する場合にも起こります。
パイプラインはバケツリレーと一緒で命令が順番に実行される前提で作られています。もちろんある命令を実行する時に次に実行される確率が最も高いのはその命令の次の命令です。しかしジャンプ命令、関数(サブルーチン)呼び出し命令などはそうではありません。
その命令の次に実行されるのは飛び先の命令です。

JMP ラベル A
ラベル B： MOV．．．

上のような命令であっても飛び先のアドレスの計算が終わるのはパイプラインの EX ステップですし、そもそも持ってきた命令が何かわかるのはパイプラインの ID ステップです。
この時はすでに次の命令が 1 つ目のステップに読み込まれています。
しかしこの命令は実行してはまずい命令です。
この命令は実行できないわけで、パイプラインハザード（分岐ハザード）と呼びます。
ちなみに前に述べたデータの方もパイプラインハザード（データハザード）と呼びます。
ちなみに何らかのハザードの発生によりパイプラインがスムーズに流れないことをパイプラインストールと呼びます。ストールとは失速の意味でエンストとはエンジンストールの略で、エンジンストップの略ではありません。

この分岐ハザードの対策もデータハザードと同じ方法が使えます。
というところまでを前提知識として、具体的な話に移ります。



図ではタスク同士または ISR(Interrupt Service Routine)同士の例ですが、ISR とタスクでも構いません。
ひとつのタスクが共有データ X に値をセットした後、値をセットしたことを示すためにフラグを ON します。
そのタスクとは独立して動く別タスクでは、まず最初にフラグを調べて ON の場合は共有データ X の値を取り込みます。
この処理も当初 CISC で実行していた時は正しく動作していたのですが、RISC に移植すると動作がおかしくなる場合が生じるようになりました。
毎回ではないのですが、タスク B が flg が ON なのでデータを取り込むのですが、そのデータにはまだタスク A がデータをセットしていない場合が生まれてしまいました。

この原因を考えてみてください。解説は次回に回します。ヒントは命令のリ・オーダーとフラグを ON する命令がタスクの最後（コンパイラからは関数の最後に見える）にあったことです。
それではまた次回に。

組込みシステム編 第 1 回おわり