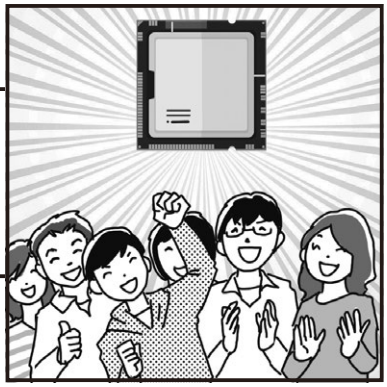
もう一度重要になる気がする
プロセッサ開発のセンス



第7回 腕の見せ所！
プロセッサの動的検証と静的検証

山下 源

今回はプロセッサ開発における機能検証について紹介します．

ハードウェアやソフトウェアに限らず，製品開発では検証に多くの工数をかけます．プロセッサ開発も例外ではありません．プロセッサの機能はプログラムの命令列を実行することです．もしプロセッサにバグがあってプログラムを正常に実行できないと，システム全体が正常に動作できません．検証の重要性は明らかで，決して不十分であってはならない開発フェーズです．一方で，製品開発には開発期間や予算といった制約があります．製品に求められる品質を検証で担保すると共に，開発コストやリリース日程を守って計画通りに開発を完了することもまた製品開発に求められます．

● プロセッサの機能検証…命令を正しく実行できるか確認

大変でない検証はありませんが，プロセッサの検証も大変なものになりがちです．その要因は数多くありますが，今回はデータ・パスの構造に起因する例を説明します．

次の命令はRISC-V(1)の整数加算命令で，レジスタt0，t1の加算結果をレジスタt2に格納します．

```
add t2,t0,t1
```

プロセッサがこの命令を正しく実行することを検証するためには，ソース・レジスタであるt0，t1に値を入力してこの命令を実行し，デスティネーション・レジスタt2に期待値が格納されることを確認する必要があります．

プロセッサの機能検証のポイント

● 公開検証プログラムだけでは不十分

RISC-Vでは命令の動作を確認する検証プログラムとして，GitHubで公開されているオープンソースのものや商用のテスト・スイートなど多数あり（riscv-tests(2)やRISC-V Architecture Test SIG(3)など），これらを活用することができます．しかし，多くのパイプライン構造のプロセッサではそのような検証だけでは不十分です．

その理由をプロセッサのデータ・パス構造の例で説明します．5段パイプライン・プロセッサのデータ・パスの例として，バイパスがない場合を**図1**に，バイパスがある場合を**図2**に示しました．次の①sub命令の結果を②add命令がレジスタt0経由で参照するケースを考えます．

```
①sub t0,t3,t4
②add t2,t0,t1
```

▶バイパスがない場合

図1のデータ・パス構造では，①sub命令の結果を一度レジスタ・ファイルのレジスタt0に書き込んだ後に，レジスタ・ファイルからレジスタt0の値を読み出して②add命令を実行する動作となります．この場合，レジスタt0のレジスタ・ファイルへの書き込みと読み出しで3クロック・サイクルのパイプライン・ストールが発生するため，②add命令はクロック・サイクル#9（先頭から9番目）で完了することになります．パイプライン・ストールの間，プロセッサは命令を実行することなく時間だけが過ぎていくため，性能の低いプロセッサとなります．

▶バイパスがある場合

一方，**図2**のデータ・パス構造では，①sub命令の演算結果をEXステージのALU（算術論理演算器）の出力から，②add命令のIDステージにバイパスすることができます．②add命令は①sub命令の結果を必ずしもレジスタ・ファイルから読み出す必要はありません．クロック・サイクル#3の①sub命令の演算結果を次のクロック・サイクル#4で②add命令が参照して演算できれば，パイプライン・ストールを発生させることなく②add命令をクロック・サイクル#6で完了することができ，より性能の高いプロセッサとすることができます．

● 内部構造を全て検証する必要がある

上記は一例であり，多段パイプライン・ステージを備えるプロセッサや，ALUを複数備えるスーパー・