ハードの理解で差がつく時代

シミュレータの自作からはじめるCPU動作メカニズム

第4回 Cortex-M0シミュレータ作りに挑戦

中森 章

今回は参考用に，これまでのRISC-Vよりさらに本格的な，Cortex-M0（Armv6-M）のシミュレータ作りに挑戦します．（編集部）

RISC-VとCortex-M0の命令セット・アーキテクチャの違い

Cortex-M0のソフトウェア・シミュレータ（ISS：Instruction Set Simulator）を作る前に，RISC-VとCortex-M0の命令セット・アーキテクチャ（ISA：Instruction Set Architecture，以下ISA）の違いを押さえておく必要があります．

(1) RISC-Vの特徴

- 除算命令がある
- 汎用レジスタが32本

(2) Cortex-M0の特徴

- 16ビット/32ビット可変長命令がある
- 汎用レジスタが16本
- 条件フラグがある
- 多重転送（PUSH/POP/LDM/STM）などの1命令で複数の機能を持つ命令
- 命令実行の結果，条件フラグが変化する
- PC（プログラム・カウンタ）が，実行中の命令より4番地先を指している
- 汎用レジスタのr15がPCとなっており，r15にライトすることで分岐する
- PC相対ロード

まあ，こんなところです．RISC-Vに比べてCortex-M0は機能が豊富なだけあって，実際に作る前からRISC-VのISSの小変更ではCortex-M0のISSを作ることが困難と予想されます．それでも，Cortex-M0のISSに挑戦するのは，RISC-Vがいかに単純なのかを読者の皆さんに知ってもらいたいからです．

今回挑戦するCortex-M0シミュレータ

今回製作するCortex-M0シミュレータは，大きく分けて①命令を解釈する「命令デコード」，②命令を取り込む「命令フェッチ」，③命令を実行する「命令実行」の3つから構成されます．

● ①命令デコード

「命令デコード」部を作るためには，RISC-Vの場合と同じく，命令エンコードを知っていなければなりません．Cortex-M0（Armv6-M）の命令エンコードについては，Armv6-Mのアーキテクチャ・リファレンス・マニュアル（ARM：Architecture Reference Manual）(1)に，すばりそのものの章が存在します．第A5章「Thumb命令セットのエンコーディング」がそれです（マニュアルのバージョンによっては章番号がずれているかもしれない）．Armv6-MやArmv7-MのISAはThumb（親指：腕であるArmに対して小規模という由来）と呼ばれます．

Cortex-M0というかThumbの命令エンコードの抜粋を**図1**～**図7**に示します．これを見ると，Thumb命令は基本が16ビット長で，命令のビット15～10でおおまかな命令の分類が決定されています（**図1**）．その分類が決まったら，16ビット長の命令を再度デコードすることで具体的な命令が決まります．

例えば，シフトや加減算の分類（**図2**）では，今度は命令のビット13～9をデコードすることで個々の命令に行きつきます．あるいは，ロード/ストア命令の分類（**図5**）では，ビット11～9の再デコードで個々の命令に行きつきます．このように，Cortex-M0では2段のデコードが必要になります．

Cortex-M0の「命令デコード」部が**リスト1**です．上述したように素直に2段階でデコードを行っています．OPやOP2といったデコード結果の情報は，RISC-Vの場合と違って共通化が難しいものが多く，RISC-Vの3倍の数になっています．OPやOP2の多さは「命令実行」部の複雑さに直結します．

● ②命令フェッチ

Cortex-M0の命令フェッチは少し複雑です．それは16ビットの命令長と32ビットの命令長が混在しているからです．最初の16ビットの命令コードのビット15～10をデコードすることで命令のビット長が決ま

第1回 CPUの基本動作メカニズム入門（2018年9月号）
第2回 シンプルRISC-Vシミュレータを自作する（2018年10月号）
第3回 RISC-Vシミュレータを動かしてみる（2018年11月号）