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

最終回 第6回

64ビットArm CPU Cortex-A53シミュレータづくりに挑戦

中森 章

前回は，32ビットArmアーキテクチャのCortex-A7（Armv7-A）のシミュレータ作りを行って実際に動作させてみました．今回は，64ビットArmアーキテクチャのCortex-A53（Armv8-A）のシミュレータ作りに挑戦します．（編集部）

64ビットArmv8-Aアーキテクチャの特徴

まずは，Armv7-Aから進化したと思われる，Armv8-Aのアーキテクチャの特徴を以下に示します．

- 64ビットと32ビットの命令が存在できる（32ビットの命令はArmv7-Aと互換性はない）．
- 汎用レジスタは32本．うち1本はスタック・ポインタになったりゼロ・レジスタになったりする．
- 条件実行が一部の命令に制限された．
- PC相対命令が一部の命令に制限された．
- PCが現在実行中の命令を指している（Armv7-Aでは8番地先を指していた）．
- 複数個同時転送のロード/ストア命令が削除された（代わりに，2個同時が追加）．
- デフォルトでノーマル・メモリに対するロード/ストアが非アラインを許す．
- ロード/ストアでのベース・レジスタに対するオフセット（即値）はデータ・サイズでスケーリングされる（スケーリングしないロード/ストア命令も存在する）．
- 命令を汎用的にして，多くの機能を兼用させている．
- 即値との論理演算命令において，13ビットの情報から64ビットの即値を作れるように変態的（？）なエンコーディングをする．

● 特徴1…ゼロ・レジスタの導入

ここで，一番の特徴はゼロ・レジスタの導入でしょう．汎用レジスタはx0からx31（32ビットで使う場合は，w0からw31）の32本があります．ただし，x31とw31は便宜的なもので，正式には存在しません．この32番目のレジスタが命令で指定された場合は，ときと場合で変わるのですが，スタック・ポインタ（spまたはwsp）またはゼロ・レジスタ（xzrまたはwzr）として機能します．ゼロ・レジスタとは，値がゼロのレジスタです．リード時には0が読み出され，ライト時には値が書き捨てになります．

ゼロ・レジスタの使用例としては，比較命令やレジスタ間の移動命令が考えられます．例えば，Armv7-Aでいうところの比較命令，

```
cmp  rn, rm
```

は，Armv8-Aでは，

```
subs xzr, xn, xm
```

として減算命令を流用します．これは，ゼロ・レジスタへの書き捨ての例です．あるいは，Armv7-Aでのレジスタ間の移動，

```
mov rd, rn
```

は，Armv8-Aでは，

```
add xd, xn, xrz
```

として加算命令を流用します．これは，ゼロ・レジスタを0として使用する例です．

● 特徴2…命令の汎用化により命令数を削減しようとしている

ゼロ・レジスタを使用するのとは別に，Armv8-Aでは命令の汎用化が見られます．例えば，EXTR命令を流用すれば，右シフトや右ローテートを実現できますし，UBFM/SBFM命令を流用すれば，左シフトやゼロ拡張，符号拡張を実現できます．命令を汎用化することで命令数を減らす工夫をしてあります．

このように個々の命令を汎用化して命令数を削減しようとしているArmv8-Aのアーキテクチャですが，1点矛盾があります．それは，レジスタ間接分岐命令「BR」と関数からの復帰命令「RET」が別個になっていることです．アーキテクチャ・マニュアルを読む限り，2つは同一の機能を持った命令です．なぜなのでしょうか？ Armv7-Aでは「BX」命令に相当します．

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）