# ARM 系 CPU の性能評価
## －Application Binary Interface の及ぼす影響－

川村　暁*，小山　智大**，佐々木慶文*

# Performance Evaluation of the ARM CPU
## —Effects of Application Binary Interface—

Satoshi KAWAMURA*, Tomohiro OYAMA** and Yoshifumi SASAKI*

*Dept. of Information Technology and Electronics, Faculty of Science and Engineering, Ishinomaki Senshu University
1 Shinmito Minamisakai Ishinomaki-shi MIYAGI. 986-8580 JAPAN

Abstract

Embedded CPUs, manufactured by ARM, are one of the leading CPU in the field of smart device. It is well known that the performance of ARM CPUs depends on the application binary interface (ABI) such as OABI and EABI. However, actual difference in performance to these ABIs was not clarified. Thus, we have evaluated the performances of the OABI and the EABI systems by measuring operation time of some fundamental arithmetic and sort operations. As a result, we have confirmed that their performances depend on not only ABIs but also quality of compilers, which generate ABI specific execution codes.

## １．はじめに

半導体技術の高度化（微細化）に伴い CPU の能力も向上している。組込み向け CPU においては、投入できるリソース（トランジスタ数）が多くなるに従って、1）CPU 自身の能力・機能の向上とコア数の増大、2）CPU コアに周辺回路を載せる余裕の増大・搭載される機能の増大・高度化、3）合目的性のため多種多様な能力・機能を持った多くの種類の CPU が存在する、という状況にある[(1)～(5)]。

ここで、組込み向け CPU は、単一・限られたアーキテクチャで覆われた市場とはなっておらず、複数の CPU アーキテクチャが存在しているため、公称値から性能を評価することが難しい[(1)～(7)]。また、CPU アーキテクチャが異なるだけではなく、能力・機能が合目的性であることに起因して CPU チップに実装される各種機能も様々であり、同一の CPU 系列であっても構成が大きく異なる製品が無数にあることは、開発者が目的（製品・応用）に即した製品を選択する際の障壁となっている。換言すれば、a）様々な CPU アーキテクチャが存在し、b）CPU 自身の拡張機能も様々であり、c）周辺回路も千差万別である多様な CPU 製品から、目的に対して最適な CPU 製品を選択しなければならないということである[(1),(2),(5)]。

CPU の性能を表す最も単純な指標には、動作クロック周波数、実際に演算を行わせた上でのランク付けを行う SPEC（Standard Performance Evaluation Corporation. SPECint, SPECfp が有名）などがある[(8)]。これらは基本的に、ノイマン型 CPU が命令を逐次実行することに立脚している。しかしながら、異なる CPU アーキテクチャでは、そもそも命令実行の様態が異なる（パイプライン段数とハザードの影響、レジスタ数、IPC 等を指す）上に、SIMD のような命令の逐次実行とは考え方のことなる専用の命令・機能を搭載している場合もあるため、クロック周波数や SPECint, SPECfp だけでは十全な評価が行えないことは明らかであろう。

このような状況のため、我々は、実際に CPU

*石巻専修大学理工学部情報電子工学科
**平成 23 年度石巻専修大学理工学部情報電子工学科卒業。現東日本旅客鉄道株式会社。

が用いられている状況に即した組込み環境での評価を行うため、いくつかの考え方について検討を加えている[6],[7]。実システムの性能を測るには、単純な演算の速度だけではなく実応用を指向したベンチマークも必要であることを示している[6],[7]。

本稿では、上記の流れを踏まえ、ARM 系 CPU におけるアプリケーションとシステム間のインタフェース（Application Binary Interface: ABI[9]）の違いが、各種演算・プログラムの実行速度にどのような影響を与えるかについて検討を加えた。ARM 系 CPU は、次世代 Windows においてもサポートされることからもわかる通り、比較的組込み向けでシェアが高い CPU であること、ARM 系 CPU の特異性（ARM 系 CPU は、命令セットを定める英国 ARM 社からライセンス（IP）を得、製造・販売する会社が命令セットレベルで互換性のある CPU を市場に投入している）のため、同一の命令セットであっても数多くの CPU があること、ARM11 からは新規な動作モードが追加されており、単一 CPU でも異なった処理速度の様態を示す場合があるためである。即ち，同一の世代・同一の IP を採用した CPU であっても、設計・製造したメーカにより CPU の性能にばらつきがある。

ARM CPU 環境において代表的な 2 種類の ABI、即ち OABI（Old ABI / Legacy ABI）と EABI（Embedded ABI）環境下での性能評価を Atmark Techno 社製組込み CPU ボード Armadillo-500FX で検証を行った結果、ABI が異なることによる影響は、どのような処理内容のプログラムであるかによって大きく異なること、コンパイラが生成するマシン語の質（生成されたマシン語命令の種類）にもよることが示されたので報告する。

## 2．ABI（Application Binary Interface）

ABI とは、アプリケーションプログラムと OS 間にあって，オブジェクトレベルでの互換性を確保するために規定された低レベルインタフェースである[9]～[11]。ABI は、CPU ファミリー毎・用途毎に存在し、Intel 系 CPU の Intel Binary Compatibility Standard（iBCS）、MIPS 系 CPU の MIPS-ABI（日本版 MIPS-ABI という側面を持つ OCMP）、PowerPC 向けの PowerPC ABI などがある。特に、組込み用途向けの ABI としては、ARM EABI（Embedded ABI）、MIPS EABI が知られている。ARM CPU における ABI はマシン語を生成するに際し、CPU と関連する各種資源（レジスタ、スタックの使い方、システムコールの方法、拡張機能への対応）をどのように利用するかを規定する。これにより、世代の異なる ARM CPU において、後方互換性と高速化・多機能化に資する諸機能を、矛盾なく取り扱えるようにしている。

ARM CPU 環境における ABI は、大きく 3 種類に分けられる[9]～[11]。

(1) The bare platform family （例：bare metal).
(2) The DLL-based family （例：Palm OS, Symbian OS).
(3) The SVr4-based family（例：Linux, free BSD).

本稿では、(3)に該当する Old ABI（OABI. Debian/GNU Linux では arm と表記）と Embedded ABI（EABI. Debian/GNU Linux では armel と表記）を取り上げている。特に OABI については、Debian のサイト（Debian ports -> arm）において、今後は推奨されない旨（deprecated）の記載もある[12]。

OABI は、特定の用途を指向しない ABI である。特に浮動小数点演算に関しても、通常の実行形式と同様、CPU 側に FPU が搭載されていない場合の処理は処理系・ライブラリに依存する。これに対して EABI では、組込み系 CPU においては FPU が搭載されていないことが多いこと、浮動小数点演算などの特殊な命令を実行するたびに例外が発生し処理性能が低下することを考慮し、無駄な例外発生やモード切替などが発生しないように考えられている。すなわち EABI では、割り込みが発生しパイプラインがハザードした上にキャッシュミスの可能性を生じる可能性を減らすことを企図している。これを実現するため、OABI と EABI では、一部の命令の呼び出しに専用の命令を用いている。また、ARM CPU においても新しい世代の CPU ほど追加の機能（Thumb 拡張命令や VFP 等を指す）があるた

め、EABI 環境では、これらを用いて高速化することも考えられている。

## 3．異なる ABI 下での計算機システムの性能評価

組込みシステムは、実製品でのシステム制御を担うなど合目的で設計・製造されるため、汎用的な PC の環境とは大きく異なった開発環境・運用環境で用いられる。このため、計算機システムの性能評価を行う場合も、組込みシステムの様相に即した手法で行う必要がある。そこで本稿では、組込みシステムの評価や学習を行う目的で販売されている組込み CPU ボードを用いて評価実験を行った。

表 1 に、評価実験に用いた計算機環境を示す。評価ボード（Atmark Techno 社製 Armadillo-500FX）を 2 台用意し、それぞれ異なる ABI 環境が動作するように設定した。ここで、構築した環境の差異（Linux OS の kernel revision は同一でも distribution の version が異なり、distribution で用意される gcc compiler が異なっている）は、これまでに述べてきたとおり、評価ボード側で用意された開発環境・実行環境であることは明記する。すなわち、なるべく実際の使用環境に近い形での評価を行うため、あえて提供されたままの環境を用いて実験を行った。

システムの性能を測るための基本的な演算として、ソート各種（bubble sort, insertion sort, oddeven sort, quick sort, selection sort, shaker sort, shell sort）、整数演算（32 bit int 型の加減乗除算および剰余）、浮動小数点演算（32 bit float 型の加減乗除算）を用いて実験を行った。処理時間の計測には、gettimeofday 関数を用いた。計測に用いた手法を表 2 に示す。

表 1　実験環境。OABI と EABI 環境を用いて実験を行った。

| | OABI | EABI |
|---|---|---|
| 機種 | Atmark Techno Armadillo-500FX | |
| CPU | Freescale i.MX31 (ARM1136JF-S) | |
| Memory | 128MB LPDDR SDRAM | |
| L1 Cache | Data 16KB　Instruction 16KB | |
| L2 Cache | 128KB | |
| OS | Linux :2.6.26-at14<br>Debian 4.0 | Linux :2.6.26-at14<br>Debian 5.0.4 |
| Compiler | arm-linux-gnu-gcc-4.1.2 | arm-linux-gnueabi-gcc-4.3.2 |

## 4．計算機実験の結果と考察

図 1 から図 3 に、計算機実験の結果を示す。グラフは、EABI の実行速度（秒）を OABI の実行速度（秒）で除して示した（EABI が OABI より高速な場合は 1.0 未満の値になり、EABI の方が遅い場合は 1.0 以上の値となる）。

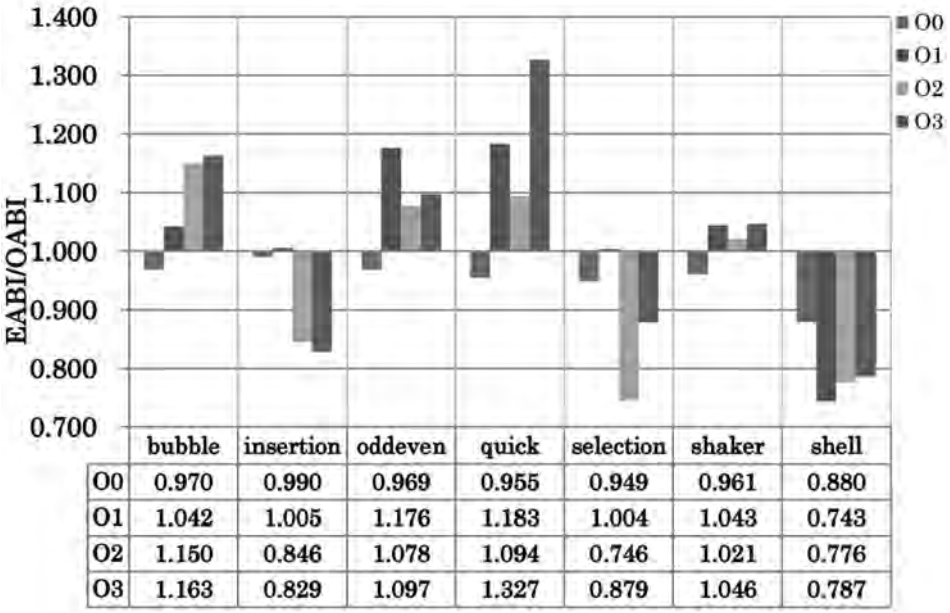
結果に差が出たものについては、アセンブラコードによる解析を行い、ABI による差であるか検証を行った。特に、処理構造や命令数などが同等であるか、 EABI 専用命令の使用はあるかについて検討を加えた。

### 4．1　ソート

図 1 のソートの実験結果では、shell sort 以外のソート演算については、ソースコードレベルで浮動小数点演算は存在しない。実験結果より、最適化なし（O0）の場合は、shell sort 以外は EABI システムの実行時間が 3〜5 %程度高速でほとんど差は無かった。但し、すべてのアルゴリズムで EABI が高速になっている。これに対し最適化あり（O1〜O3）の場合は、EABI が高速なもの、OABI が高速なものに結果が分かれた。

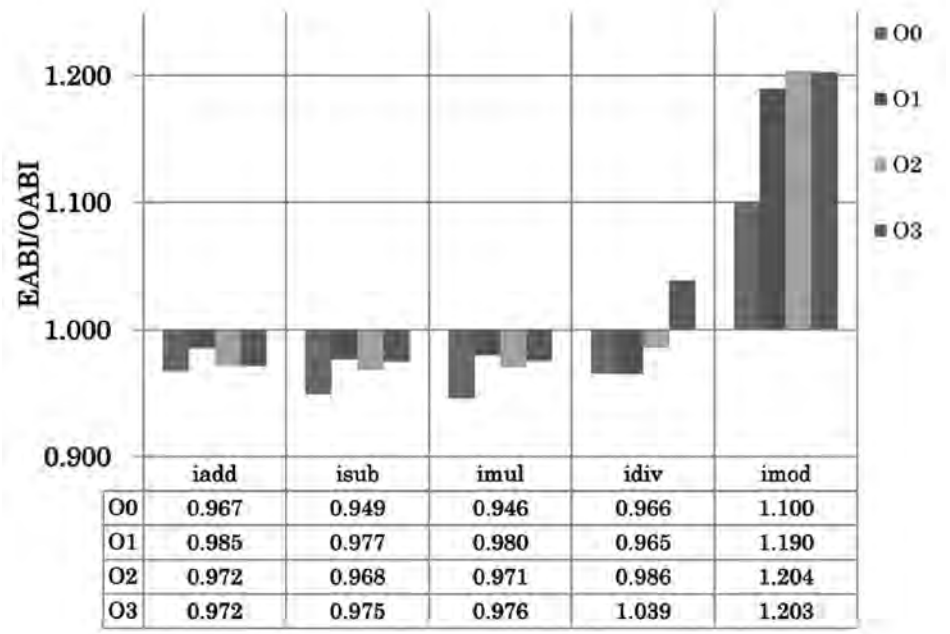
表 2　評価実験の詳細

| 評価項目 | | 詳細 |
|---|---|---|
| ソート | ソートアルゴリズム | bubble, insertion, oddeven, quick, selection, shaker, shell |
| | データサイズ | 1kbytes〜64kbytesは1kbytes刻み，68kbytes〜128kbytesは4kbytes刻み |
| 整数演算 | 演算の種類 | 加減乗除算および剰余 |
| | 演算階数 | 100000回から1000000回，100000回刻み |
| | 試行回数 | 10回数 |
| | データサイズ | 32 bit int型整数 |
| 浮動小数点演算 | 演算の種類 | 加減乗除算 |
| | 演算階数 | 100000回から1000000回，100000回刻み |
| | 試行回数 | 10回数 |
| | データサイズ | 32 bit float型整数 |



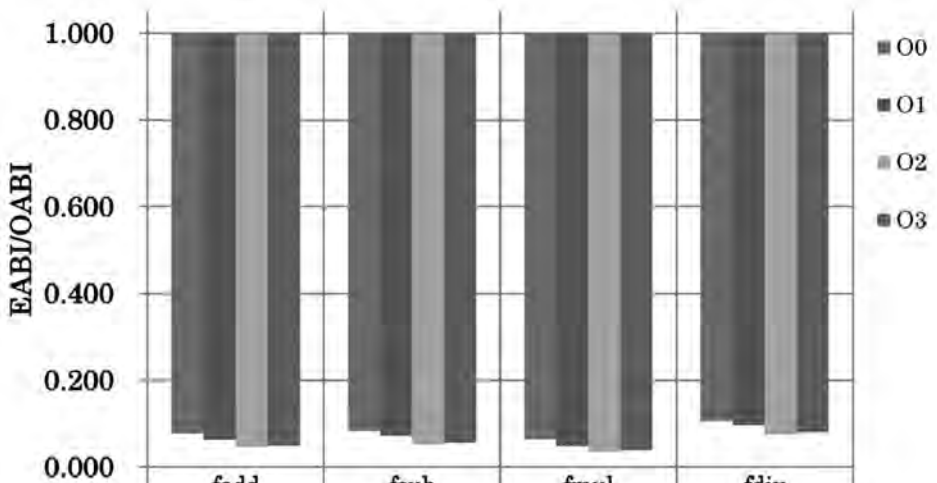

| | bubble | insertion | oddeven | quick | selection | shaker | shell |
|---|---|---|---|---|---|---|---|
| O0 | 0.970 | 0.990 | 0.969 | 0.955 | 0.949 | 0.961 | 0.880 |
| O1 | 1.042 | 1.005 | 1.176 | 1.183 | 1.004 | 1.043 | 0.743 |
| O2 | 1.150 | 0.846 | 1.078 | 1.094 | 0.746 | 1.021 | 0.776 |
| O3 | 1.163 | 0.829 | 1.097 | 1.327 | 0.879 | 1.046 | 0.787 |

図 1　ソートの結果。EABI の処理時間を OABI の処理時間で除している（EABI / OABI）。




| | iadd | isub | imul | idiv | imod |
|---|---|---|---|---|---|
| O0 | 0.967 | 0.949 | 0.946 | 0.966 | 1.100 |
| O1 | 0.985 | 0.977 | 0.980 | 0.965 | 1.190 |
| O2 | 0.972 | 0.968 | 0.971 | 0.986 | 1.204 |
| O3 | 0.972 | 0.975 | 0.976 | 1.039 | 1.203 |

図 2　整数演算の結果。EABI の処理時間を OABI の処理時間で除している（EABI / OABI）。




| | fadd | fsub | fmul | fdiv |
|---|---|---|---|---|
| O0 | 0.077 | 0.084 | 0.065 | 0.107 |
| O1 | 0.064 | 0.072 | 0.049 | 0.097 |
| O2 | 0.046 | 0.053 | 0.035 | 0.075 |
| O3 | 0.050 | 0.056 | 0.039 | 0.080 |

図 3　浮動小数点演算の結果。EABI の処理時間を OABI の処理時間で除している（EABI / OABI）。

表 3　ソート結果のまとめ。EABI と OABI の処理時間の比率により、グループ分けを行った。

| | ソートアルゴリズム | ソースコードレベルでの浮動小数点演算 | 備考 |
|---|---|---|---|
| グループ A | bubble,oddeven,quick | 無 | 最適化をかけた場合、OABI システムの実行時間が 10%程度の速くなる |
| グループ B | insertion,selection | 無 | 最適化をかけた場合、EABI システムの演算時間が、10%から 15%程度速くなる |
| グループ C | shell | 有 | 最適化に関わらず EABI の高速化の度合いが大きい |
| グループ D | shaker | 無 | 有意な差は無 |

ソート時間の計測結果に基づき、表 3 のようにグループ化できる。それぞれについて生成されたソースコードレベルでの考察を行った。

はじめに、グループ A について考える。

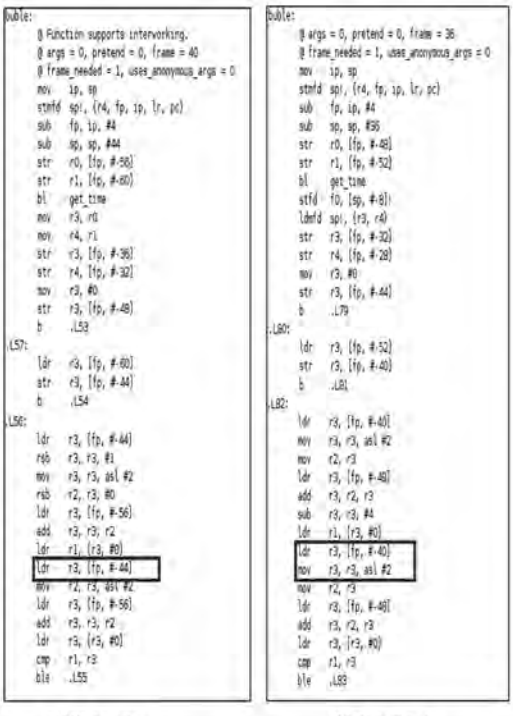
最適化を適用しなかった場合（O0）の bubble sort のアセンブラコードを図 4 に示す。基本的に処理構造は同じであることと、比較を行う部分の構造が異なっている（要素の並べ替えが必要か比較を行うためにメモリアドレスを計算するが、同一の作業であるにも関わらず、命令数が 1 異なる）。高々1 命令であるが、bubble sort の計算量（$n^2$）だけ実行されてしまうことを考慮すると数%程度の時間差となったと示唆される。

グループ A は最適化をした場合、OABI の方が高速である。この原因を考えるために、生成されたアセンブラコードに基づき考察を行った。図 5 に bubble sort（O2 最適化）のアセンブラコードを示す。EABI システムのコードには EABI の専用命令は使用されていないこと、命令数や命令の種類が多少異なっていることが分かる。ここで、比較に用いているコンパイラのバージョンが異なっていることを考慮し、EABI のコンパイラを gcc-4.1.2 に強制的にダウングレードして実験を行った。図 6 に、生成されたアセンブラコードと実験結果を示す。結果から明らかな通り、2 つのシステムにおいて演算時間の有意な差は見られ

なくなり、アセンブラコードもほぼ同等のコードとなった。すなわち、グループ A のアルゴリズムについては、ABI による差ではなくコンパイラバージョンによる差が生じていることが確認された。なお、組込みシステム（開発）において、推奨環境以外を用いるのは推奨されない（自己責任）ことは強調する。

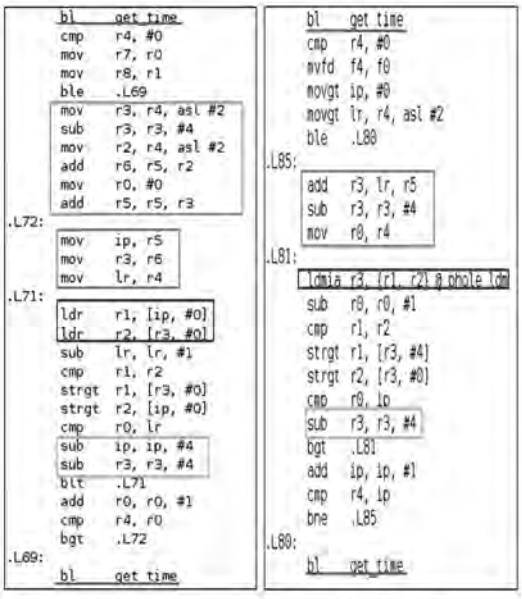
次に、グループ B について考察する。グループ A の解析と同様、アセンブラコードに基づき考察した結果、特に最適化ありの場合において、コンパイラバージョンによる影響が大きいことが示された（コンパイラバージョンを同一にすると殆ど同等のコード、実行時間となった）。

グループ C については、浮動小数点演算が含まれるため EABI 専用命令が用いられているが、アルゴリズム中における実行回数はあまり多くないため、実行時間の差の大きさと符合しない。これもコンパイラバージョンの差が大きな影響を与えている可能性が示唆された。



EABI システム　　OABI システム

図 4　グループ A・Bubble ソート、O0 のアセンブラコード。一部アセンブラが異なっている。



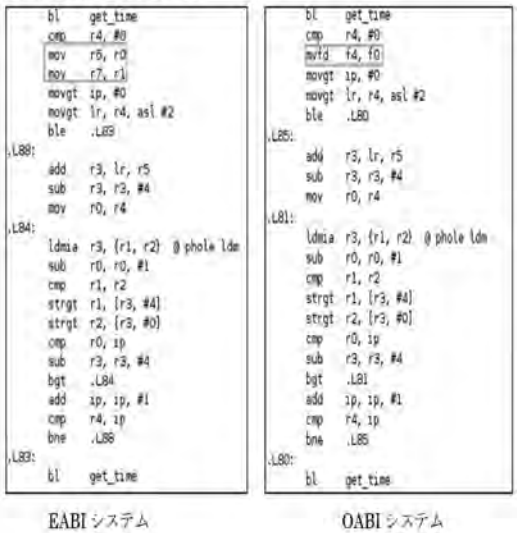
EABI システム　　OABI システム

図 5　グループ A・Bubble ソート、O2 のアセンブラコード。一部アセンブラが異なっている。

(a) gcc 4.1.2を用いた場合の、EABIとOABIの処理速度の比較

| Date size[KB] | gcc_v4.1.2EABI [s] | gcc_v4.1.2OABI [s] | EABI/OABI |
|---|---|---|---|
| 128 | 15.740 | 15.779 | 0.998 |

(b) 生成されたアセンブラコード



EABI システム　　OABI システム

図 6　グループ A・Bubble ソート、O2、gcc 4.1.2 でコンパイルした場合の比較．実行速度、アセンブラコード共に殆ど同一となった。

グループ D については、最適化の有無に関わらず有意な演算時間の差は見られなかった。アセンブラコードでの比較においても、生成されたコードに大きな差は見られなかった。

ソートの結果をまとめると、プログラムの実行速度の差は浮動小数点演算など EABI 専用命令が必要かどうかに依存する事が示された。ただし、コンパイラのバージョンによっても生成されるコードも異なることも付記する。

### 4．2　整数演算

図 2 の整数演算の結果では、加減乗除算に関しては、殆どの場合で 3〜5 %程度 EABI システムが速いという結果が得られた。最適化なし（O0）の場合が EABI と OABI の差が大きい（EABI が高速）という結果となった。最適化あり（O3）の除算および剰余に関しては、OABI が高速という結果となった。ARM 製 CPU はハードウェアに除算を計算するハードウェアを持っていないため、除算と剰余に関しては、EABI は専用命令、OABI は汎用のライブラリ関数で演算が行われている。最適化あり（O3）の除算および剰余については、OABI システムの方が 10 %程度速くなり、若干大きな差が見られた。この点に関しては、詳細な検討が必要である。

4．1 節と同様に、生成されたアセンブラコードで比較した結果、整数演算のうち加減乗算では生成されるコードに有意な差は見られなかった。これに対し除算では、OABI では通常のライブラリ関数の呼び出しが、EABI では EABI 専用命令が使用されていた。ARM CPU においては、除算器は搭載されていないため、除算は ABI に関わらずソフトウェアにより演算を行っているためと考えられる。実行ファイルを逆アセンブルし用いられている命令を比較したが、大きな差は無く、ほぼ同等であることを確認している。

剰余に関しては、最適化に関わらず、OABI システムの実行時間が 10〜20 %程度高速化されている。アセンブラコードを比較したところ、命令数や処理構造には大きな違いは見られなかった。但し、実際に剰余演算を行っている部分についてみると、EABI システムでは EABI 専用命令、OABI システムでは剰余を求めるライブラリ関数が用いられていた。実行ファイルを逆アセンブルし、用いられている命令を確認したところ、EABI システムでは、さらに EABI 除算命令を呼び出していることが確認された。一方、OABI システムでは、直接、剰余を求めるライブラリ関数を呼び出していることが確認された。これらの関数について調査を行ったところ、OABI で用いられている関数の方が、効率が良いことが明らかとなった。従って、剰余に関する差は、内部で呼び出されている関数に依存した物であることが確認された。

### 4．3　浮動小数点演算

図 3 の浮動小数点演算に関しては、EABI システムの方が高速であるという結果が得られた。具体的には演算時間が OABI システムの 10 %未満（10 倍以上高速）という結果になった。また、最適化レベルを上げると、演算時間の差が大きくなるという傾向が見られた。この知見は、プログラムの作成に際して重要であると考えられる。汎用のソフトウェア割り込みをも用いた浮動小数点演算処理による割り込み処理が、非常に重い処理となっていることが分かる。

前節までと同様に、アセンブラコードによる比較を行った結果、EABI では EABI 専用命令が用いられていること、OABI は汎用命令が用いられていることが明らかとなった。その他の部分については、命令数などに違い（演算前のアドレス計算）があるが、ステート数から判断して同等であると考えられ、また、処理構造も同等であったことから、浮動小数点演算に関しては、EABI 専用命令の効果が大きく、EABI システムは浮動小数点演算を多用する場合に極めて有用であるということが明らかとなった。

## 5．まとめ

本研究では、ARM 製 CPU における ABI の違いが演算性能に与える影響について実験的に評価を行った。代表的な ABI である、EABI と OABI の演算性能の比較を行った結果、浮動小数点演算を多用する場合は EABI が非常に有用であることが明らかになった。また、コンパイラバージョンや最適化が演算性能に与える影響も少なくないことが、実験を行った過程で明らかになったことは、極めて重要な成果であると考えら

れる。

今後の課題としては、他の ARM 製 CPU で同様の実験を行い、同様な結果になるか明らかにすることや、コンパイラのバージョンや最適化が演算性能に与える影響についても明らかにすることが重要である。

## 文献

(1) Heath S(2002)Embedded systems design, Second Edition, Newnes

(2) Peter M (2003) Embedded System Design, Springer

(3) Rob T, Tim W(2012), Fast and Effective Embedded Systems Design: Applying the ARM mbed, Newnes

(4) ARM Port (web site),
http://www.debian.org/ports/arm/

(5) Andrew N. Sloss ら著，アーム訳 (2007) ARM 組み込みソフトウェア入門―記述例で学ぶ組み込み機器設計のためのシステム開発 (Design Wave Advance シリーズ)、CQ 出版

(6) 川村、佐々木、佐藤、丸岡 (2010) 組込み向け CPU の性能評価試み―最適化の影響―、石巻専修大学紀要第 21 号、pp.19-25

(7) 川村、佐々木、佐藤、丸岡 (2011) 代表的な数値計算を用いた組込み向け CPU の性能評価、石巻専修大学紀要第 22 号、pp.19-25

(8) Standard Performance Evaluation Corporation, SPEC CPU 2006 (web site),
http://www.spec.org/cpu2006/

(9) Arm ltd. (2009) Application Binary Interface for the ARM Architecture v2.08 (PDF),
infocenter.arm.com/help/topic/com.arm.doc.ihi0036b/IHI 0036B_bsabi.pdf

(10) ARM ltd. (2002, 2003) ARM1136JF-S/ARM1136J-S(Rev:r0p2) テクニカルリ ファレンスマニュアル、ARM ltd

(11) ARM ltd. (2001, 2002) ARM926EJ-S/(Rev:0 S(Rev:0) テクニカルリファレンスマニュアル、ARM ltd

(12) Debian ARM Port (web site),
http://www.debian.org/ports/arm/