社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2006－ＡＲＣ－168（15）
2006／6／9

# ARMアーキテクチャ向け命令分解型スーパスカラ

中島　康彦†

† 奈良先端科学技術大学院大学情報科学研究科　〒630-0192　奈良県生駒市高山町 8916-5
E-mail: †nakashim@is.naist.jp

**あらまし**　組み込み用途プロセッサの業界標準である ARM アーキテクチャは，マルチプルロード／ストアなどの複合命令を備えた CISC 型命令セットを採用しており，そのままスーパスカラ化することが難しい．本研究では，ARM 命令セットを RISC 型内部命令セットに変換してスーパスカラ実行する際に生じる問題点と解決策を示す．また，パイプラインシミュレータによる性能評価の結果を報告する．
**キーワード**　ARM，命令分解，スーパスカラ

# A Superscalar Employing Instruction Decomposition for ARM Architecture

Yasuhiko NAKASHIMA†

† Nara Institute of Science and Technology　Takayama-cho 8916-5, Ikoma-shi, Nara, 630-0192 JAPAN
E-mail: †nakashim@is.naist.jp

**Abstract**　ARM architecture is one of de facto standard embedded processors and employs CISC-type instruction set that includes complex multiple load/store and so on. In general, it is difficult to execute CISC-type instructions in parallel with superscalar technique. This report shows the problems and the solutions for superscalar techniques that decompose ARM instructions into some RISC-type internal instructions. Finally, by evaluating the model with a pipeline simulator, the performance and the analysis are disclosed.
**Key words**　ARM, Instruction Decomposition, Superscalar

## 1. まえがき

近年，携帯端末機器や組み込み用途システムでは，ＯＳや制御など命令レベル並列度を期待できないプログラムと，マルチメディア処理など高い命令レベル並列度を期待できるプログラムとを同時に実行しながら，さらに，高いリアルタイム応答性能も要求される状況が一般化してきている．特に注目すべきは，同時に実行するプログラムが，必ずしも同一命令セットアーキテクチャに基づかない点である．主な理由は，ＯＳや制御にはソフトウェア資産が豊富な業界標準プロセッサが，また，マルチメディア処理には並列処理性能の高い低電力プロセッサが有利なためである．このような状況では，一般に，複数種類のプロセッサが並置されている．

ところで，命令セットアーキテクチャが異なるプログラムを実行する手法として，すでに，再コンパイル，静的命令変換（実行形式ファイルの命令レベル変換），インタプリタ実行（命令の逐次解釈実行），動的命令変換（JIT，OCT，OOCT）が知られている．しかし，これらはいずれもソフトウェアの助けを借りる方法であり，様々なオーバヘッドを伴うことから，複数の命令セット全てを実用的速度にて動作させる要求には応えられない．また，過去には，マイクロプログラム方式のシステムにおいて，マイクロプログラムの入れ換えによる高速エミュレーションが可能であったものの，最近のマイクロプロセッサでは一般的ではない．前述のように複数種類のプロセッサを並置するのは，性能維持のために専用ハードウェアによる実現が不可欠であること，また，従来のヘテロマルチプロセッサシステムの延長として実現が比較的容易なためである．最近では，異なる命令セットアーキテクチャのプロセッサを１チップに混載した商用マルチコア型プロセッサも登場しており，今後，複数の命令セットを同時に実行できるプロセッサが徐々に一般化していくと考えられる．ただし，複数種類のコアを単純並置する方式には，全体の回路規模が大きくなったり，コア間通信のために，ある程度複雑な調停機構が必要になるなどの欠点がある．

以上のような背景の下，本研究では，単一アーキテクチャを対象とする従来のマルチスレッド実行機構を拡張して，複数アーキテクチャを同時実行する枠組みを構築し，全体として面

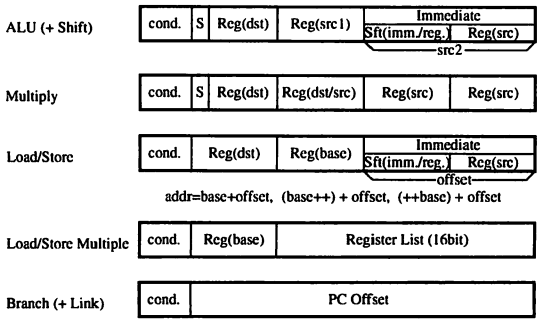


図 1　ARM 命令セットの特徴

Fig. 1　Characteristics of ARM instruction set.

積や電力遅延積を小さくできる高性能プロセッサアーキテクチャを模索する．具体的には，命令セットアーキテクチャ毎に専用の命令変換ステージを設け，各々，共通の内部命令に変換した後に，共通デコードステージ以降の SMT（Simultaneous Multithreading）機構において命令を混合して並列実行する構成を仮定する．ハードウェア共用につきものの，資源競合に伴う性能低下を抑えつつ，全体の回路規模を縮小することにより，総合性能としてはより高性能化できる可能性を探る．本報告では，組み込み用途システムにおいて業界標準となっている ARM アーキテクチャ [1] をとりあげ，まず，多少の浮動小数点演算が含まれる一般的な整数系ベンチマークプログラムに対し，浮動小数点演算器を装備しない構成でも十分な性能が得られるため，命令変換機構が簡単化できることを示す．次に，ARM 命令セットを RISC 型内部命令に分解して命令レベル並列実行機構に組み込む際に考慮すべき点と解決策を示す．さらに，パイプラインシミュレータ上において整数系ベンチマークプログラムを走行させて得た評価結果を示す．

## 2. ARM 命令セットの特徴

さほど命令レベル並列度を期待できない整数系プログラムの実行には，浮動小数点演算を省いた小さな命令セットがあればよい．ARM 命令セットでは，これまで版を重ねた過程において，浮動小数点演算命令がオプションに位置付けられており，演算ライブラリによる実行が一般的である．また最近の版では，浮動小数点演算命令が，一般的な RISC プロセッサと同等の FPA 体系から，並列演算が可能な VFP 体系 [2] に移行しており，高性能計算向けモデルのみが浮動小数点演算機構を備える状況が明確になっている．

本章では，32 ビットプロセッサの評価に多用される整数系ベンチマークプログラム（SPECint2K および MiBench）を用いて，多少の浮動小数点演算が含まれる場合でも，浮動小数点演算器を装備しない構成により，遜色のない性能が得られることを示す．比較対象は，代表的な商用 RISC である SPARC アーキテクチャのうち，32 ビットアーキテクチャの最終版である V8 [3] とした．整数乗除算および浮動小数点演算命令を生成するコンパイルオプション（gcc -msupersparc -O2）によりオブジェクトを生成した．一方 ARM については，整数除算や浮動小数点演算命令を生成せず，代わりに演算ライブラリを呼び出すコンパイルオプション（gcc -march=armv4 -msoft-float -O1）を用いた．最適化レベルが SPARC よりも低いのは，使用した gcc-4.0.0 では最適化レベルを O2 にすると，正常なオブジェクトが生成されないためである．

さて，上記コンパイルオプションおよび演算ライブラリにより使用される ARM 命令セットの概要を図 1 に示す．演算命令ごとの S ビットにより条件コードの更新／無変更を指定し，後続命令に実行条件を付加（プレディケート）することにより，分岐予測ミスペナルティが伴う条件分岐命令を削減できる．ALU 演算命令およびロード／ストア命令は，第 2 オペランド内にシ

表 1　SPARC と ARM の実行命令数比較

Table 1　Comparison between the number of executed instructions by SPARC and ARM.

(a) MiBench (small) の場合

| MiBench | SPARC (内 reg.window ミス分) | | ARM (固定小数点演算のみ) |
|---|---|---|---|
| basicmath | 58,765k | (7.8%) | ▲▲ 9.80 倍 |
| bitcount | 58,470k | (25.7%) | 0.78 倍 |
| qsort | 20,486k | (0.0%) | 0.87 倍 |
| susan | 23,686k | (0.0%) | 0.84 倍 |
| jpeg | 25,688k | (0.1%) | ▲ 1.12 倍 |
| typeset | 68,623k | (0.9%) | 0.93 倍 |
| dijkstra | 67,488k | (1.2%) | ▲ 1.07 倍 |
| patricia | 56,636k | (3.8%) | ▲ 1.65 倍 |
| ghostscript | 459,662k | (0.2%) | ▲ 1.01 倍 |
| stringsearch | 154k | (0.0%) | 0.83 倍 |
| blowfish | 31,040k | (0.0%) | 0.85 倍 |
| rijndael | 27,211k | (0.0%) | 0.85 倍 |
| sha | 13,316k | (0.0%) | 0.95 倍 |
| CRC32 | 26,009k | (0.0%) | 0.79 倍 |
| FFT | 26,491k | (3.7%) | ▲▲ 3.88 倍 |
| adpcm | 31,177k | (0.0%) | 0.85 倍 |
| | | | **幾何平均 1.19 倍** |
| | | | **basicmath と FFT を除く幾何平均 0.94 倍** |

(b) SPECint2K (train) の場合

| SPECint2K | SPARC (内 reg.window ミス分) | | ARM (固定小数点演算のみ) |
|---|---|---|---|
| 164.gzip | 49,476,829k | (0.0%) | 0.95 倍 |
| 175.vpr | 18,283,034k | (0.8%) | ▲▲ 2.23 倍 |
| 176.gcc | 4,991,055k | (2.1%) | 0.93 倍 |
| 181.mcf | 10,315,436k | (0.5%) | 0.95 倍 |
| 186.crafty | 34,800,552k | (0.7%) | 0.84 倍 |
| 197.parser | 12,847,182k | (7.0%) | 0.90 倍 |
| 253.perlbmk | 26,443,490k | (6.5%) | 0.83 倍 |
| 254.gap | 8,742,972k | (7.3%) | 0.80 倍 |
| 255.vortex | 18,593,963k | (11.7%) | 0.90 倍 |
| 256.bzip2 | 57,268,422k | (0.0%) | 0.95 倍 |
| 300.twolf | 17,635,619k | (0.0%) | ▲ 1.23 倍 |
| | | | **幾何平均 1.00 倍** |
| | | | **vpr を除く幾何平均 0.92 倍** |

表 2 RISC 型内部命令への分解
Table 2 Decomposition into internal RISC-type instructions.

| ARM 命令 | 内部命令 | 命令数 |
|---|---|---|
| ALU(シフト無し) | E | 1 |
| ALU(シフト無し) | SE | 2 |
| 乗算 (32bit*32bit+32bit→32bit) | [Mm]*4 | 8 |
| 乗算 (32bit*32bit+64bit→64bit) | [Mm]*8 m | 17 |
| 乗算 (符号付 32bit*32bit→64bit) | mm[Mm]*8 E m*3 | 22 |
| LD/ST(sft 無, 後で basereg 更新) | La | 2 |
| LD/ST(sft 無, 先に basereg 更新) | aL | 2 |
| LD/ST(sft 有, 後で basereg 更新) | LSa | 3 |
| LD/ST(sft 有, 先に basereg 更新) | SaL | 3 |
| マルチ LD/ST(N は対象 reg 数) | aa[aL]*N a | 2*N+3 |
| PC 相対分岐 | B | 1 |

フト操作を指定することにより，一般的な RISC 命令では 2 命令になるところを 1 命令に記述することができる．同様にロード／ストア命令は，同時にベースレジスタの更新が可能であり，シフト操作と併せて 3 命令になるところを 1 命令に記述できる．さらにマルチプルロード／ストア命令は，ベースレジスタにより指定した連続主記憶領域と複数レジスタとのデータ転送を 1 命令により記述でき，15 番レジスタに対するロード／ストアが PC への書き込み／読み出しに対応することと併せて，関数呼び出し／復帰時の命令数を大幅に削減できる．このように ARM 命令は，4 バイト固定長でありながら，複雑な機能を 1 命令に記述可能な CISC 型命令である．

表 1 に実行命令数の比を示す．なお，ARM では演算ライブラリにより実行され，SPARC では 1 命令により記述可能な命令に関し，SPARC が実態以上に有利にならないよう，SPARC の整数除算は 70，浮動小数点加減乗算は 4，単精度除算は 16，倍精度除算および平方根は 19 と実行レイテンシに相当する重み付けを行って計上している．また SPARC の実行命令数には，プログラムに明示的に含まれない，レジスタウィンドウ・オーバフロー／アンダフローに伴って実行される命令も含まれる (ウィンドウ数は 4 と仮定)．表中の▲は ARM の実行命令数が SPARC の 1.0 倍を超えていること，▲▲は 2.0 倍を超えていることを示す．MiBench および SPECint2K のいずれにおいても，実行命令数比率の幾何平均はほぼ等しくなり，浮動小数点演算器を装備しなくても大幅な性能低下は避けられると言える．また，浮動小数点演算を多用する basicmath，FFT，175.vpr を除くと，むしろ ARM のほうが実行命令数が少なく，複雑な命令を記述可能な ARM 命令セットの特徴が現れたと言える．

## 3. RISC 型内部命令への分解

以上のような ARM 命令列と，高い命令レベル並列度を期待できる命令列 (必ずしも ARM ではない) とを効率良く混合実行するためには，デコードステージ以降に共通内部命令を高速実行するスーパスカラ機構を配置し，主に並列度の高いプログラムに対応しながら，ARM 命令については，逐次，共通内部命令セットに変換して空き演算器に投入するよう命令スケジューリングを行う方式がよいと考えられる．この際，変換対象が浮動小数点演算を含まないことは，変換機構の簡単化に大きく貢献する．本章では，図 1 に示した ARM 命令を RISC 型命令に分解する際に考慮すべき点について述べる．まず，単一 RISC 型命令への対応付けが難しい機能と対策を以下に示す．

**第 2 オペランド内シフト機能 (ALU/Load/Store)：** ALU 演算やアドレス計算にシフト操作が組み込まれている ARM 命令セットは，シフタと ALU をカスケード接続するパイプライン構成との相性が良く，従来のインプリメントのほとんどがこのような構成である．しかし，粒度の小さい演算器を並列に配置し演算器の利用率向上を図る一般的なスーパスカラ機構には，演算レイテンシが増加し演算の組合せが制約を受けるカスケード接続は馴染まない．このため，共通内部命令セットではシフト操作を独立させ，シフトを含む ALU 演算やアドレス計算は 2 命令に分解して実行するのが妥当である．

**整数積和演算 (Multiply)：** ARM には，符号付き乗算結果の 64 ビットを既存の 64 ビットに累算する積和演算など，3 オペランド形式の RISC 命令よりも多くのソースオペランドを必要とする演算がある．3 オペランド形式かつ 1 サイクルにて実行可能な内部命令 (32*8 ビット乗算や符号反転など) に分解することにより，一般的なスーパスカラ機構の枠組に納めることができる．

**マルチプルロード／ストア機能 (Load/Store Multiple)：** 連続主記憶領域と複数レジスタとのデータ転送についても同様に，複数のロード／ストア命令とベースレジスタ更新命令に分解する．

**実行条件付き命令：** 実行条件付き命令は，それ自身は複雑な命令ではない．しかし，先行命令の実行結果を後続命令が待ち合わせるスーパスカラ機構において，先行命令となった実行条件付き命令がデスティネーションレジスタへの書き込みを抑止すると，後続命令の待ち合わせ先を別の先行命令に切り替えなければならず，複雑な機構が必要となる．このため，実行条件付き命令については，条件に関わらず依存関係が変化しない命令列に組み替える必要がある．具体的には，

(1) $if(condition)\ ADD\ R1, R2 \rightarrow R3$

を表現する ARM 命令は，作業用レジスタを介する，

(1) $ADD\ R1, R2 \rightarrow R_{tmp}$

(2) $if(condition)\ R_{tmp} \rightarrow R3\ else\ R3 \rightarrow R3$

の 2 命令に分解することにより，R3 を参照する後続命令が条件に依らず常に第 2 の選択命令に依存するようにできる．

以上をまとめると，ARM 命令をスーパスカラ実行するために必要な内部命令セットは次のようになる．

**シフト演算 (S)：** 左右論理シフト，右算術シフト，ローテート

**3 オペランド形式の ALU 演算 (E)：** 一般的な加減算および論理演算

**3 オペランド形式のアドレス計算 (a)：** ベースレジスタの更新にも使用する一般的な加減算

**積和演算用の基本乗算 (M)：** 8 通りの 32*8 ビット乗算 (4 通りのバイト位置と，演算結果の 64 ビットのうち前半／後半のいずれを取り出すかの組合せ)

**積和演算用の補助演算 (m)：** 絶対値，部分積加算，符号絶対

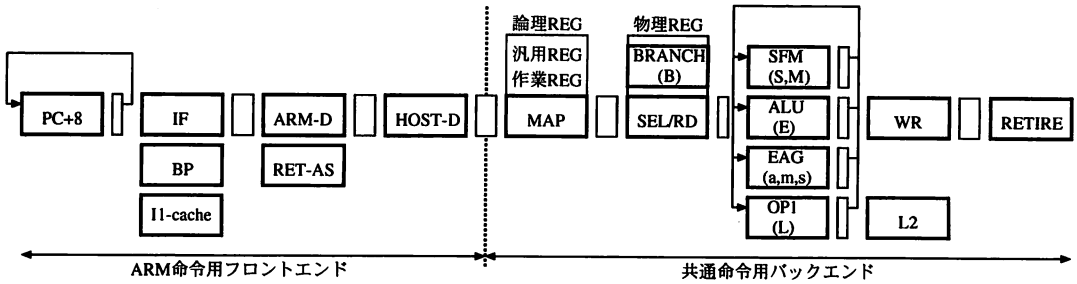


図 2　パイプライン構成
Fig. 2　Structure of pipeline.

表 3　パイプラインモデルのパラメタ
Table 3　Parameters of pipeline model.

| | |
|---|---|
| 分岐予測（gshare） | pht:2bit × 4K entry, PC と ghr 下位 6bit の xor により選択 |
| RET-AS | 8 entry |
| 命令ウィンドウ兼物理レジスタ | 32 entry |
| ストアバッファ | 8 entry |
| キャッシュライン | 64byte |
| I1 キャッシュ | 4way,16KB, ミス時 10cycle |
| D1 キャッシュ | 4way,16KB, ミス時 10cycle |
| L2 キャッシュ | 4way,2MB, ミス時 100cycle |

値，符号検査，符号反転
**ロード／ストア命令（L）：** 4 バイト長までのロード／ストア
**条件分岐命令（B）：** PC 相対の分岐命令
**選択命令（s）：** 条件コードに基づき 2 つのソースレジスタから 1 つを選択

括弧内の略号を用いて，各 ARM 命令がどのような命令列に分解されるかを表 2 に示す．例えば 32*32 ビットの符号付き乗算は，各ソースレジスタの絶対値を求める m，32*8 ビットの乗算を繰り返して 64 ビットの上位／下位各 32 ビットを求める M と m の組（8 組），ソースレジスタの符号ビットから演算結果の符号を求める E，結果の符号に応じて乗算結果の上位／下位の 2 の補数を求める m（2 命令），演算結果の 64 ビットから条件コード（Negative および Zero）を生成する m の合計 22 命令に分解される．また，各 ARM 命令の先頭において always 以外の実行条件が指定されている場合には，前述の方法により，汎用レジスタを更新する内部命令をさらに 2 命令に分解する．なお，言うまでもなく，分解後命令間の中間値の受け渡しには汎用レジスタを使用できない．以上に示した命令分解のために，16 本の汎用レジスタ以外に，作業用レジスタを 6 本設ける．

## 4.　評価モデルのパイプライン構成

図 2 に，一般的なスーパスカラ実行機構に，これまでに説明した命令分解機構を埋め込んだパイプライン構成を示す．命令フェッチ（IF）では，gshare 分岐予測機構（BP）と連携して，命令キャッシュ（I1-cache）の 8 バイト境界から連続 2 命令を命令バッファに取り出す．1 段目の命令デコード（ARM-D）では，前述の命令分解を行う．1 サイクルに可能な分解は，最大 2 個の ARM 命令から最大 4 個の内部命令とする．4 命令を超える分解は複数サイクルにより行う．関数呼び出しに用いる PC 相対分岐命令の場合は，関数呼び出しからの復帰先を予測するリターンアドレススタック（RET-AS）に次アドレスを格納し，15 番レジスタへの書き込み命令の場合は，復帰とみなしてリターンアドレススタックから取り出した予測復帰先アドレスを IF ステージに送る．ただし，ARM では復帰命令に実行条件を付けることができるため，リターンアドレススタックの制御が難しく，後述するようにヒット率向上が課題である．なお，予測の正誤は後述する RETIRE ステージにおいて判明し，もし誤りの場合はパイプラインをフラッシュする．2 段目の命令デコード（HOST-D）では，前述の実行条件付き命令の分解を行う．1 サイクルに可能な分解は，最大 4 個の内部命令から最大 4 個の内部命令とする．以上が ARM 命令専用のステージである．

続く MAP では，32 ビット幅の 3 オペランド形式に統一された内部命令について，汎用レジスタと作業用レジスタからなる論理レジスタから，命令ウィンドウを兼ねる物理レジスタへのマッピングを行う（最大 4 命令）．論理レジスタ数が 16 本と一般的な RISC プロセッサの半分であること，また，作業用レジスタ番号は命令分解のパターン毎に固定され競合しやすいことから，リネーミングの効果が高いと考えられる．SEL/RD では，演算器からのバイパスが利用可能かどうかも含めて，依存関係の待ち合わせおよび命令発行を行う（最大 4 命令）．条件コードの待ち合わせおよび条件分岐命令の実行も担当する．

さて，続く演算器については，分解後の内部命令がなるべく並列実行され，かつ，バイパスが必要以上に複雑にならないよう機能をグループ化すると効率的である．例えば表 2 に示した乗算では，M と m を同時実行できれば効果が高い．同様にロード／ストアを毎サイクル実行するためには，L，S，a の同時実行が必要である．一方，S と M の同時実行，および，a と m と s の同時実行が必要となる局面は出現頻度が低いと予想される．以上をまとめると，「S と M」，「E」，「a と m と s」，「L」の合計 4 グループを各機能ユニット「SFM：シフタおよび乗算器」，「ALU」，「EAG：アドレス計算と積和補助と選択」，「OP1：キャッシュおよびストアバッファ」に対応付けるのが効率的と言える．残る WRITE では，命令ウィンドウ兼物理レジスタへ

表 4　ARM 命令タイプの出現頻度 (%)

Table 4　Percentages of ARM instruction types. (%)

(a) MiBench (small) の場合

| | E | SE | [Mm]4 | [Mm]8 | mm.. | La | aL | LSa | SaL | aa.. | B |
|---|---|---|---|---|---|---|---|---|---|---|---|
| basic | 51.6 | 27.0 | 0.0 | 0.5 | 0.0 | 0.1 | 2.0 | 0.0 | 0.0 | 4.7 | 14.1 |
| bitco | 74.4 | 8.6 | 0.0 | 0.0 | 0.0 | 0.0 | 5.4 | 0.0 | 1.3 | 0.0 | 10.3 |
| qsort | 43.1 | 0.3 | 0.0 | 0.0 | 0.0 | 12.2 | 20.9 | 0.0 | 0.2 | 0.5 | 22.2 |
| susan | 41.7 | 3.9 | 16.4 | 0.1 | 0.0 | 16.3 | 10.3 | 0.0 | 0.0 | 0.3 | 11.1 |
| jpeg | 42.3 | 13.4 | 0.0 | 0.0 | 0.0 | 0.3 | 31.4 | 0.0 | 1.8 | 0.6 | 10.0 |
| types | 38.5 | 3.9 | 0.0 | 0.0 | 0.1 | 0.8 | 36.8 | 0.0 | 1.2 | 1.5 | 16.9 |
| dijks | 36.7 | 18.6 | 0.2 | 0.0 | 0.0 | 0.3 | 22.2 | 0.0 | 4.3 | 0.4 | 17.3 |
| patri | 48.2 | 24.5 | 0.1 | 0.4 | 0.0 | 1.0 | 7.5 | 0.0 | 0.4 | 2.0 | 15.8 |
| ghost | 47.0 | 6.2 | 0.0 | 0.3 | 0.9 | 3.1 | 20.0 | 0.0 | 1.1 | 1.1 | 20.0 |
| ispel | 42.6 | 5.8 | 0.0 | 0.0 | 0.0 | 2.7 | 28.7 | 0.0 | 0.0 | 2.7 | 17.6 |
| strin | 36.2 | 0.0 | 0.0 | 0.0 | 0.0 | 14.1 | 11.8 | 0.0 | 0.8 | 0.4 | 36.5 |
| blowf | 50.4 | 11.3 | 0.0 | 0.0 | 0.0 | 1.3 | 19.3 | 0.0 | 2.4 | 0.4 | 11.3 |
| rijnd | 49.6 | 18.8 | 0.0 | 0.0 | 0.0 | 0.0 | 24.2 | 0.0 | 4.7 | 0.3 | 2.1 |
| sha | 62.4 | 6.2 | 0.0 | 0.0 | 0.0 | 0.0 | 23.3 | 0.0 | 0.0 | 0.3 | 7.7 |
| CRC | 46.7 | 6.7 | 0.0 | 0.0 | 0.0 | 0.0 | 20.0 | 0.0 | 6.7 | 0.0 | 13.3 |
| FFT | 50.4 | 25.3 | 0.0 | 1.4 | 0.0 | 0.3 | 5.2 | 0.0 | 0.1 | 4.2 | 13.2 |
| adpc | 63.6 | 11.7 | 0.0 | 0.0 | 0.0 | 3.9 | 5.2 | 0.0 | 5.2 | 0.0 | 10.4 |

(b) SPECint2K (train) の場合

| | E | SE | [Mm]4 | [Mm]8 | mm.. | La | aL | LSa | SaL | aa.. | B |
|---|---|---|---|---|---|---|---|---|---|---|---|
| gzip | 42.4 | 6.6 | 0.0 | 0.0 | 0.0 | 0.7 | 32.4 | 0.0 | 0.9 | 1.5 | 15.4 |
| vpr | 47.4 | 23.5 | 0.0 | 0.8 | 0.0 | 0.0 | 13.0 | 0.0 | 0.7 | 2.3 | 12.3 |
| gcc | 42.6 | 4.0 | 0.0 | 0.0 | 0.0 | 7.8 | 22.6 | 0.0 | 2.0 | 1.8 | 19.0 |
| mcf | 42.4 | 4.6 | 0.0 | 0.0 | 0.0 | 0.0 | 29.2 | 0.0 | 1.9 | 0.6 | 21.2 |
| crafty | 43.0 | 10.7 | 0.2 | 0.0 | 0.1 | 0.1 | 24.5 | 0.0 | 2.3 | 9.5 | 9.7 |
| parser | 45.2 | 7.1 | 0.0 | 0.0 | 0.0 | 0.6 | 25.1 | 0.0 | 1.9 | 3.0 | 17.2 |
| perlb | 34.4 | 6.2 | 0.0 | 0.0 | 0.0 | 1.9 | 35.0 | 0.0 | 0.8 | 4.2 | 17.4 |
| gap | 36.3 | 7.0 | 0.1 | 0.0 | 0.0 | 2.5 | 34.8 | 0.0 | 2.2 | 2.6 | 14.6 |
| vortex | 41.2 | 2.3 | 0.0 | 0.0 | 0.0 | 1.2 | 33.9 | 0.0 | 1.7 | 5.0 | 14.7 |
| bzip2 | 42.2 | 9.3 | 0.0 | 0.0 | 0.0 | 0.2 | 33.5 | 0.0 | 0.8 | 0.8 | 13.2 |
| twolf | 44.0 | 21.3 | 0.1 | 0.3 | 0.1 | 0.0 | 21.3 | 0.0 | 0.5 | 1.2 | 11.3 |

の書き込みを行う（最大 4 命令）．RETIRE では，先行命令が全て完了した命令を命令ウィンドウ兼物理レジスタから外して論理レジスタを更新する（最大 4 命令）．

## 5. 性能評価

本章では，以上のようなパイプラインモデルにおいて，表 3 に示すパラメタを仮定し，SPECint2K および MiBench を走行させて得た結果を示す．これまでの仮定と対比させながら分析するために，以下の項目について測定を行った．

**ARM 命令の各出現頻度：** 表 2 に示した各命令パターンの出現頻度

**SFM,ALU,EAG,OP1 各動作率：** 全サイクルに対する，SFM，ALU，EAG，OP1 の動作率

**ARM 命令／内部命令の IPC：** 1 サイクルに実行（リタイア）した ARM 命令数および内部命令数の平均

**分岐予測ミス：** 実行（リタイア）した全内部命令のうち分岐予測ミスした PC 相対分岐命令の命令数比率，および，分岐予測ミス率

**RET-AS ミス：** 同様に RET-AS が HIT しなかった内部間接分岐命令の命令数比率，および，RET-AS ミス率

**I1 ミス：** ARM 命令フェッチ時の I1 キャッシュミス回数の命令数比率，および，ミス率

**D1 ミス：** 内部命令オペランドフェッチ時の D1 キャッシュミス回数の命令数比率，および，ミス率

**L2 ミス：** ARM 命令および内部命令オペランドフェッチ時の L2 キャッシュミス率

**Reifetch：** Gshare または RET-AS による分岐先予測アドレスが次命令でない場合に，命令フェッチをやり直した回数のサイクル数比．1 回あたり約 1 サイクルのペナルティが生じる．

**Flush：** 同様に分岐予測がミスした場合に，パイプラインをフラッシュした回数のサイクル数比．1 回あたり約 7 サイクルのペナルティが生じる．

表 5　IPC と演算器動作率 (%)

Table 5　IPC and operating ratio of units. (%)

(a) MiBench (small) の場合

| | ARM 命令 IPC | 内部命令 IPC | SFM 動作率 | ALU 動作率 | EAG 動作率 | OP1 動作率 |
|---|---|---|---|---|---|---|
| basicmath | 0.75 | 1.54 | 24.3 | 66.0 | 50.2 | 13.4 |
| bitcount | 0.90 | 1.24 | 9.2 | 77.1 | 25.5 | 6.4 |
| qsort | 0.64 | ★ 0.97 | 0.4 | 31.3 | 31.1 | 24.9 |
| susan | 0.81 | 2.05 | 58.1 | 37.7 | 81.9 | 23.7 |
| jpeg | 0.83 | 1.40 | 13.2 | 47.4 | 41.8 | 31.5 |
| typeset | 0.55 | ★ 0.96 | 3.7 | 24.8 | 34.6 | 27.7 |
| dijkstra | 0.80 | 1.37 | 19.2 | 45.3 | 36.7 | 23.3 |
| patricia | 0.74 | 1.38 | 22.1 | 56.4 | 40.5 | 12.3 |
| ghostscript | 0.76 | 1.39 | 13.4 | 43.8 | 48.0 | 24.1 |
| ispell | 0.65 | 1.20 | 3.8 | 32.2 | 43.3 | 30.9 |
| stringsearch | 0.80 | ★ 1.09 | 0.7 | 29.9 | 27.1 | 23.5 |
| blowfish | 0.74 | ★ 1.05 | 10.2 | 48.6 | 19.5 | 21.2 |
| rijndael | 1.13 | 1.80 | 26.6 | 77.7 | 37.6 | 35.8 |
| sha | 0.99 | 1.40 | 6.4 | 67.9 | 34.5 | 24.3 |
| CRC32 | 0.63 | ★ 0.88 | 8.4 | 37.7 | 16.8 | 21.0 |
| FFT | 0.78 | 1.66 | 29.8 | 63.4 | 55.2 | 14.9 |
| adpcm | 1.07 | 1.71 | 18.0 | 80.4 | 45.8 | 15.3 |
| 算術平均 | 0.80 | 1.36 | 15.7 | 78.9 | 39.4 | 22.0 |

(b) SPECint2K (train) の場合

| | ARM 命令 IPC | 内部命令 IPC | SFM 動作率 | ALU 動作率 | EAG 動作率 | OP1 動作率 |
|---|---|---|---|---|---|---|
| 164.gzip | 0.72 | 1.18 | 5.4 | 36.3 | 36.6 | 29.7 |
| 175.vpr | 0.71 | 1.39 | 22.6 | 53.3 | 44.6 | 17.1 |
| 176.gcc | 0.63 | ★ 1.08 | 4.6 | 30.1 | 36.8 | 28.0 |
| 181.mcf | 0.36 | ★ 0.56 | 2.6 | 17.4 | 16.7 | 12.5 |
| 186.crafty | 0.54 | 1.17 | 8.3 | 29.6 | 48.8 | 28.6 |
| 197.parser | 0.62 | 1.14 | 5.8 | 33.3 | 41.2 | 26.2 |
| 253.perlbmk | 0.52 | ★ 1.02 | 4.1 | 22.9 | 43.7 | 30.2 |
| 254.gap | 0.53 | ★ 1.00 | 5.5 | 25.7 | 41.3 | 28.7 |
| 255.vortex | 0.60 | 1.25 | 2.7 | 26.4 | 52.0 | 36.0 |
| 256.bzip2 | 0.84 | 1.34 | 8.5 | 43.3 | 39.8 | 31.5 |
| 300.twolf | 0.67 | 1.28 | 16.9 | 45.3 | 42.6 | 18.8 |
| 算術平均 | 0.61 | 1.13 | 7.9 | 33.1 | 40.4 | 26.1 |

表 4 に，リタイアした ARM 命令タイプの出現頻度を示す．各列は表 2 に示した各命令タイプに対応する．シフト無し ALU 命令（E）がほぼ半数を占め，シフト無しロード／ストア命令（La，aL），PC 相対分岐命令（B），シフト付き ALU 命令（SE）がこれに続く．乗算（M，m），および，ロード／ストア後にベースレジスタを更新するシフト付き命令（LSa）の頻度は低いことがわかる．特に乗算の出現頻度が低いことは，ビット幅の小さい乗算器の繰り返し利用でも性能に大きな影響を与えないことを示唆している．

表 5 に，ARM 命令および内部命令の IPC と，各演算器の動作率を示す．まず IPC の比から，1 個の ARM 命令が 1.8 個程度の内部命令に分解されていることがわかる．ARM 命令のフェッチ速度を 2 命令／サイクル，内部命令の実行を 2 倍の 4 命令／サイクルとする構成が妥当であったと言える．ALU と EAG が主に動作しており，SFM の動作率はさほど高くないものの，シフタや乗算器の機能を他の演算器に無理なく組み込むことは難しいため，前述した演算器のグループ化は妥当であったと言える．MiBench よりも SPECint2K の IPC が低いのは，後者の方が OP1 の比率が高いことから，キャッシュミスが一因ではないかと推測できる．なお表中の★は，IPC が 1.1 未満であることを示している．

さらにパイプラインの挙動を分析するために，表 6 にパイ

表 6 性能低下の分析 (%)
Table 6 Analysis of performance degradation. (%)

(a) MiBench (small) の場合

| | 分岐予測ミス回数（ミス率） | RET-AS ミス回数（ミス率） | I1 ミス回数（ミス率） | D1 ミス回数（ミス率） | L2 ミス率 | Reifetch | Flush |
|---|---|---|---|---|---|---|---|
| basicmath | 0.4 (6.5) | 0.5 (49.2) | 0.1 (0.3) | 0.0 (0.0) | 0.0 | 10.3 | 10.1 |
| bitcount | 0.2 (3.2) | 1.7 (100) | 0.0 (0.0) | 0.0 (0.0) | 88.8 | 13.5 | 16.7 |
| qsort | 0.5 (3.2) | ★ 1.7 (80.6) | 0.0 (0.0) | ★ 0.5 (3.7) | 14.8 | ★ 13.8 | ★ 14.8 |
| susan | 0.3 (6.1) | 0.0 (24.4) | 0.0 (0.0) | 0.0 (0.1) | 6.2 | 8.6 | 4.1 |
| jpeg | 0.4 (6.0) | 0.2 (83.7) | 0.0 (0.0) | 0.1 (0.4) | 13.2 | 8.1 | 5.0 |
| typeset | 0.6 (6.3) | 0.5 (47.6) | ★ 0.6 (1.4) | ★ 1.3 (6.2) | 4.1 | 8.2 | 7.7 |
| dijkstra | 0.2 (2.4) | 0.2 (44.6) | 0.0 (0.0) | 0.1 (0.5) | 1.0 | 10.7 | 4.5 |
| patricia | 0.5 (5.4) | 0.5 (52.9) | 0.5 (1.3) | 0.0 (1.0) | 0.7 | 10.4 | 9.1 |
| ghostscript | 0.3 (2.6) | 0.6 (64.2) | 0.1 (0.1) | 0.0 (0.4) | 5.0 | 12.9 | 8.8 |
| ispell | 0.5 (5.1) | 0.2 (28.5) | 0.0 (0.1) | 0.2 (1.6) | 25.2 | 10.4 | 6.2 |
| stringsearch | ★ 0.8 (2.9) | 0.3 (60.9) | 0.0 (0.1) | 0.0 (0.4) | 96.1 | ★ 19.9 | 8.5 |
| blowfish | 0.1 (1.8) | ★ 1.9 (71.8) | 0.0 (0.0) | 0.0 (0.0) | 98.9 | ★ 16.3 | ★ 15.2 |
| rijndael | 0.1 (4.0) | 0.2 (74.9) | 0.0 (0.0) | 0.0 (0.0) | 98.4 | 3.4 | 2.7 |
| sha | 0.2 (4.0) | 0.0 (0.7) | 0.0 (0.0) | 0.0 (0.0) | 97.1 | 7.8 | 2.1 |
| CRC32 | 0.0 (0.0) | ★ 0.7 (14.3) | 0.0 (0.0) | 0.0 (0.0) | 97.5 | ★ 22.1 | 4.2 |
| FFT | 0.3 (5.6) | 0.4 (38.9) | 0.0 (0.1) | 0.0 (0.3) | 1.2 | 8.8 | 8.8 |
| adpcm | 0.0 (0.0) | 0.0 (10.1) | 0.0 (0.0) | 0.0 (0.0) | 95.2 | 4.2 | 0.0 |

(b) SPECint2K (train) の場合

| | 分岐予測ミス回数（ミス率） | RET-AS ミス回数（ミス率） | I1 ミス回数（ミス率） | D1 ミス回数（ミス率） | L2 ミス率 | Reifetch | Flush |
|---|---|---|---|---|---|---|---|
| 164.gzip | 0.4 (3.9) | 0.2 (42.7) | 0.0 (0.0) | 1.1 (5.7) | 0.7 | 8.8 | 4.7 |
| 175.vpr | 0.6 (9.5) | 0.4 (37.3) | 0.2 (0.4) | 0.4 (4.7) | 0.0 | 8.1 | 10.0 |
| 176.gcc | 0.5 (4.8) | 0.4 (54.5) | ★ 0.9 (2.2) | 0.4 (2.8) | 1.2 | ★ 10.7 | 7.2 |
| 181.mcf | ★ 0.7 (5.2) | 0.1 (4.2) | 0.0 (0.0) | ★ 4.4 (25.7) | 12.6 | 7.3 | 3.1 |
| 186.crafty | 0.4 (8.8) | 0.3 (39.6) | 1.5 (5.1) | 1.1 (5.6) | 0.0 | 4.7 | 5.4 |
| 197.parser | 0.6 (6.4) | 0.5 (28.6) | 0.0 (0.0) | 0.7 (4.2) | 5.4 | 10.5 | 8.7 |
| 253.perlbmk | 0.3 (3.7) | ★ 1.4 (70.9) | ★ 1.1 (2.8) | 0.6 (3.0) | 0.0 | ★ 10.1 | ★ 12.5 |
| 254.gap | 0.3 (3.6) | ★ 1.8 (88.6) | 0.3 (0.7) | 0.5 (2.6) | 9.3 | ★ 9.9 | ★ 14.6 |
| 255.vortex | 0.1 (1.8) | 0.2 (15.2) | 0.8 (2.8) | 0.7 (4.0) | 0.6 | 5.6 | 2.6 |
| 256.bzip2 | 0.1 (1.6) | 0.0 (4.9) | 0.0 (0.0) | 0.3 (2.0) | 26.0 | 5.5 | 1.3 |
| 300.twolf | 0.5 (8.9) | 0.2 (33.5) | 0.2 (0.5) | 0.9 (8.6) | 0.0 | 6.3 | 6.3 |

プラインインタロックの要因および発生頻度を示す．表中の★は，表 5 に示した各★に対応しており，IPC の低下に関与したと考えられるパイプラインインタロックを強調している．SPECint2K では I1 および D1 のミス回数およびミス率がともに MiBench よりも高く，SPECint2K の IPC が低い主要因は，命令およびオペランドキャッシュミスにあると言える．一方，分岐予測に関わる機能では，gshare のミス率は低いのに対し，リターンアドレススタックによる復帰先予測のミス率は高く，パイプラインフラッシュの頻度を押し上げている．1 回あたり約 7 サイクルのペナルティを伴うパイプラインフラッシュは性能に大きな影響を与えることから，今後，リターンアドレススタックのヒット率向上が課題であると言える．

## 6. あとがき

本報告では，ARM 命令セットを RISC 型命令に分解してスーパスカラ実行する機構について述べた．特に，選択命令の導入により実行条件付き命令を効率良くスーパスカラ機構に組み込めること，1 個の ARM 命令が約 1.8 個の内部命令に変換されること．整数系ベンチマークプログラムでは乗算の出現頻度が低く，より幅の小さい乗算器の繰り返し利用で十分であることを示した．また，さらなる性能向上のためには，実行条件付き復帰命令に対応可能なリターンアドレススタックを考案し，ヒット率を向上させる努力が必要であることがわかった．

## 文　献


[1] ARM Architecture Reference Manual, ARM Limited, ARM DDI 0100E (2000).

[2] VFP11 Vector Floating-point Coprocessor Technical Reference Manual, ARM Limited, ARM DDI 0274E (2005)

[3] The SPARC Architecture Manual Version 8, Revision SAV080SI9308, SPARC International Inc. (1992).