2006－ARC－170 (11)
2006／11／30

OROCHI評価用集中命令ウィンドウ型スーパスカラの設計

小島 知也† 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: †{tomoya-k,nakashim}@is.naist.jp

あらまし 異種命令セットを同時実行するプロセッサ“OROCHI”の定量的比較評価のため，集中命令ウィンドウ型の命令分解型 ARM スーパスカラの回路設計および評価を行った．特に，ARM 命令を RISC 型命令に変換する回路の規模や性能について分析しマルチプルロード／ストアモジュールの高速化が重要であることを述べる．

キーワード ARM，命令分解，スーパスカラ，集中命令ウィンドウ

Design of a Centralized Instruction Window Superscalar for Evaluation of OROCHI

Tomoya KOJIMA† and Yasuhiko NAKASHIMA†

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

Abstract Processor named OROCHI executes heterogeneous instruction sets at a time. In this report, for quantitative evaluation of OROCHI, we designed and evaluated an instruction decomposition ARM superscalar that has a centralized instruction window. Especially, we show the area and the performance of the circuit that decomposes ARM instructions into RISC-type instructions, and conclude it is important to speed up the module that handles multiple load/store instructions.

Key words ARM, Instruction Decomposition, Superscalar, Concentrated Instruction Window

1. まえがき

近年，携帯端末機器や組み込みシステムでは，命令レベル並列度を期待できない OS などの制御プログラムと，高い命令レベル並列度を期待できるマルチメディア処理プログラムを同時に実行する環境が一般化しつつある．さらに，このような状況下でリアルタイム応答性も求められている．特に注目すべきは，マルチメディア処理プログラムと OS などの制御プログラムが，同一の命令セットアーキテクチャに基づかない点である．この理由は，OS など制御用プログラムにはソフトウエア資産が豊富な業界標準であるプロセッサが使用され，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となるためである．このように，マルチメディア処理と OS などの制御用プログラムを同時に実行する環境では，一般に複数種類のプロセッサが並置される．

ところで，命令セットが異なるプログラムを実行する方法には，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の変換），インタプリタ実行（命令の逐次解釈実行），動的命令変換（Jit : Just in time compiler や OCT : Object code translation）が知られている．しかし，これらの方法はいずれもソフトウエアの助けを借りるため，様々なオーバー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理由は，性能維持のため専用ハードウエアによる実現が不可欠であり，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 [1] 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に並置する方式では，全体の回路規模が大きくなったり，場合や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．以上のような背景から，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を拡張して複数アーキテクチャを同時実行するプロセッサである“OROCHI”の性能評価を最終目的とする．

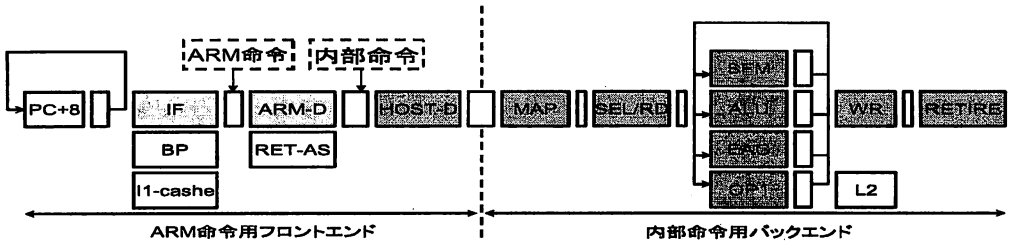

図 1　パイプライン構成

2. OROCHI の概要

OROCHI は，マルチメディア処理用の並列処理性能の高いバックエンド部分の空き演算能力を利用して OS など並列性の低いプログラムを混合実行することを狙うプロセッサである．バックエンドの構成は VLIW でも，スーパースカラでもよい．最終的には VLIW とスーパースカラの比較を行うことを目的としており，本研究では，まず，スーパースカラを仮定している．図 1 に示すように，OROCHI は，ARM 命令用フロントエンドと，内部命令用バックエンドから構成されている．命令フェッチステージ (IF) では，gshare 分岐予測機構 (BP) を用いて命令キャッシュ (I1-cashe) から連続 2 命令を ARM 命令バッファに取り出す．1 段目の命令デコーダ (ARM-D) では，ARM 命令を RISC 型内部命令に変換する．本デコーダの設計と分析が本稿の主要部分であり，詳細については後述する．2 段目の命令デコーダ (HOST-D) では実行条件付命令を条件に関わらず依存関係が変化しない 2 つの命令列に分解する．以上が ARM 命令用のパイプラインステージである．

次に，内部命令用バックエンドについて説明する．MAP では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．SEL/RD では，演算器からのバイパスが利用可能かどうか調べるとともに，依存関係の待ち合わせと命令発行を行う．演算器については，分解後の内部命令がなるべく並列に実行されるように，「SFM：シフト演算と積和演算用の補助演算」，「ALU」，「EAG：アドレス計算と積和補助演算と選択」，「OP1：キャッシュおよびストアバッファ」に対応付ける．WR では命令ウィンドウ兼物理レジスタへの書き込みを行う．最後に 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．

3. 命令分解機構の概要

3.1 ARM 命令セットの概要

図 2 に示すように，ARM 命令 [2] は 4 バイト固定長であり，複雑な機能を 1 つの CISC 型命令として記述できる．ALU 演算命令およびロード／ストア命令は，第 2 オペランド内にシフト演算を指定することで，一般的な RISC 命令では 2 命令になるところを 1 命令に記述できる．さらに，ロード／ストア命令はベースレジスタの更新が同時に可能であり，シフト操作，ベースレジスタ更新，ロード／ストア命令と計 3 命令になるところを 1 命令に記述できる．また，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る．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を利用して，関数呼び出し／復帰時の命令数を大幅に削減できる．また，演算命令ごとの S ビットによって条件コード更新／有無を設定し，後続命令に実行条件を付加（プレディケート）することで条件分岐命令に伴う分岐予測ミスペナルティを削減できる．

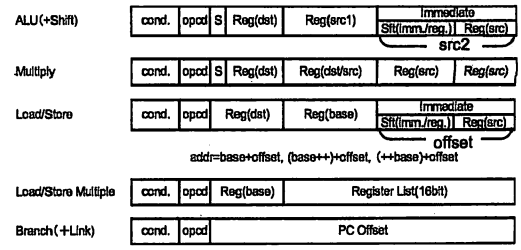

図 2　ARM 命令ビットフィールド

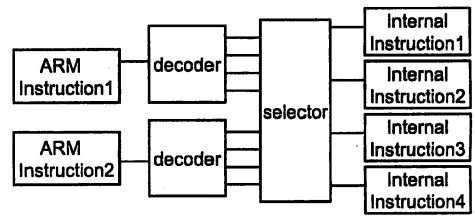

図 3　デコーダ構成 (ARM-D)

3.2 命令分解について

文献 [3] での解析に基づき，1 段目のデコーダ (ARM-D) では，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（HOST 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．例えば，乗算命令は最大 22 命令に分解する．主な内部命令を以下に示す．

シフト演算：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：一般的な加減算と論理演算.
3オペランド形式のアドレス計算：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：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み合わせ）.
積和演算用の補助演算：絶対値，部分積加算，符号絶対値，符号検査，符号反転.
ロード／ストア命令：4バイト長までのロード／ストア.
既存の命令分解機構に，Intel社のNetburst機構[4] [5]があるが，本研究の分解機構は演算器をより単純化しており，ベースレジスタ更新をロード／ストアから分離し，さらに乗算も32*8ビットであるなど，Netburstよりも粒度が細かい．図3に，1段目のデコーダ（ARM-D）構成を示す．2個のARM命令を各々のデコーダで分解し，入力された2個のARM命令の種類に応じて最大4個の内部命令に変換する．ところで，この構成では一般的なパイプラインに対して，命令分解ステージが増える．命令変換の回路規模が大きくなると，複数のパイプライン段数に分割する必要が生じ，性能低下が生じる．デコーダの構成を検討し，1ステージにおさまる構成を考えることが本稿の目的である．さて，命令デコーダが生成する値には，命令の種類，オペランド値，分岐アドレスなどがあり，各出力に対応する各モジュールによって処理する規模が異なる．

4. 命令分解機構の設計

4.1 モジュールについて

図4にデコーダ（ARM-D）の詳細を示す．入力されたARM命令を内部命令の各々のビットフィールドに対して変換値を出力する．モジュールとはこの内部命令の各ビットフィールドを各出力を対応付ける機構である．なお，マルチプルロード／ストア命令の各種変換値を生成するモジュールが他のモジュールに前置される．

4.2 モジュールの機能

次に，各々のモジュールの機能を示す．
validフラグ（v）：分解された命令が分岐予測ミスなどによって途中終了する際に，命令を中断することが可能かどうかを示す．分解された命令の中には論理レジスタを更新するものが存在するためである．
分岐予測フラグ（bpred）：IF段で得られた分岐予測フラグを，入力された命令が分岐命令の際に出力する．
分岐フラグ（br）：分岐命令であるかを出力する．
プログラムカウンタ（pc）：IF段で得られたプログラムカウンタを出力する．
分岐アドレス（target）：分岐命令の場合は，プログラムカウンタと入力値から分岐先アドレスを計算し出力する．PCと共通である15番レジスタを変更する命令の場合はリターンアドレスを出力する．
分岐予測（履歴）（ghr）：IF段で得られた分岐予測に用いるグローバル履歴6ビットを出力する
条件コード（cond）：ARM命令の上位4ビットの条件コードを出力する

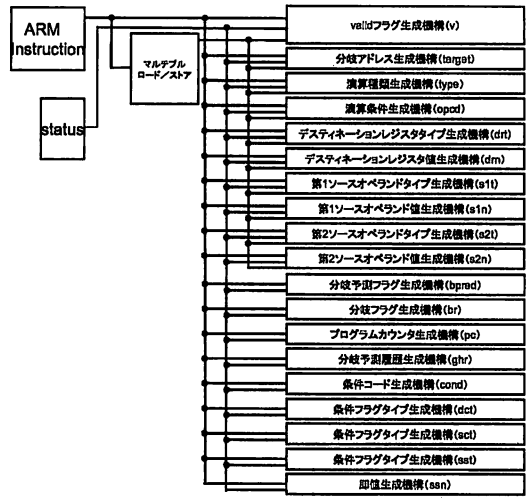

図4 デコーダ詳細（ARM-D）

演算種類（type）：入力された命令が，NOP，シフト演算，乗算，ALU演算，補助演算，ロード，ストア，分岐のいずれであるかを出力する．バックエンドにおいて，どの演算器を使用するかの判定に用いる．
演算条件（opcd）：演算器で具体的にどの処理を行うかを出力する．シフト演算の場合は左右論理シフト，右算術シフト，ローテート．ALU演算ではAND，EOR，SUBなどである．
条件コードタイプ（dct）：命令分解を行うことでARM命令が複数の命令に分割される．分解された中間命令が出力した条件コードは中間値となり最終的な条件コードではない．例えば，ALU+シフト演算で1命令であるARM命令を分解した場合，ALU命令とシフト命令に分解される．この際，シフト演算の結果得られた条件コードは中間値であり，最後にALU演算を行って条件コードが決定する．このように，条件コードを最終的に更新するかどうを出力する．
デスティネーションオペランドタイプ（drt）：デスティネーションの値が中間値か最終的な値であるかを出力する．
デスティネーションオペランドレジスタ番号（drn）：デスティネーションオペランドのレジスタ番号を出力する．
第1ソースオペランドタイプ（s1t）：第1ソースオペランドに中間命令の結果，プログラムカウンタ，自身のオペランド値のいずれを使用するかを出力する．
第1ソースオペランド値（s1n）：第1ソースの値を出力する．
第2ソースオペランドタイプ（s2t）：第2ソースに中間命令の結果，即値，自身のオペランド値のいずれを使用するかを出力する．
第2ソースオペランド値（s2n）：第2ソースの値を出力する．
条件コードタイプ（sst）：命令実行の際に，中間命令の出力した条件コードを参照するかを出力する．

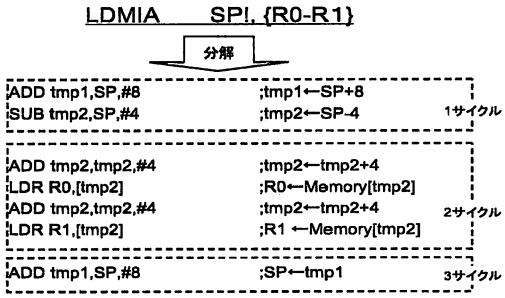

図 5 マルチプルロード／ストア命令分解例

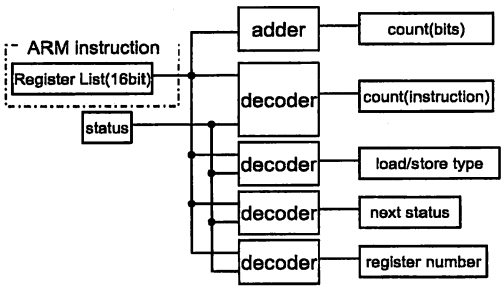

図 6 マルチプルロード／ストアモジュール

即値（ssn）：即値をローテートした最上位ビットを出力する．
条件コードタイプ（sct）：条件コードを参照するかを出力する．
次に，主なモジュールについて説明する．

4.2.1 マルチプルロード

図5に示すマルチプルロード命令 LDMIA は，ベースレジスタである SP が指す主記憶のアドレスから 0，1 番レジスタに値を転送し，最後にベースレジスタを更新する．分解後の 1 サイクル目では更新後のベースレジスタ値と，「ベースレジスター 4」を計算する．「ベースレジスター 4」を行うのは，その後のアドレス計算命令が常に+4 を行うことにより，他の命令と共通化できるためである．2 サイクル目ではアドレス計算とロード命令に分解する．そして，3 サイクル目では 1 サイクル目で計算した更新後のベースレジスタの値を用いてベースレジスタの値を更新する．このような分解を行う機構を図 6 に示す．現在のサイクルを保持する status，および ARM 命令の Register List(16bit) ビットフィールドを用いて，マルチプルロード／ストア命令が必要とする値を各モジュールに出力する．まず，1 サイクル目について説明する．先ほどのベースレジスタ更新の計算に使用した値である 8 を生成するために，個数を count(bits) により出力する．具体的には 0，1 番レジスタを示す Register List の 0，1 番ビットが 1 になっており，これらのビットを足すと 2 となり，全レジスタへの転送が終わった時のアドレスは SP+4*2 となる．Next status は次のサイクル 2 を出力する．また，分解される命令の個数 2 を出力する．2 サイクル目では転送先のレジスタ番号として転送先レジスタ番号 0 を出力し，

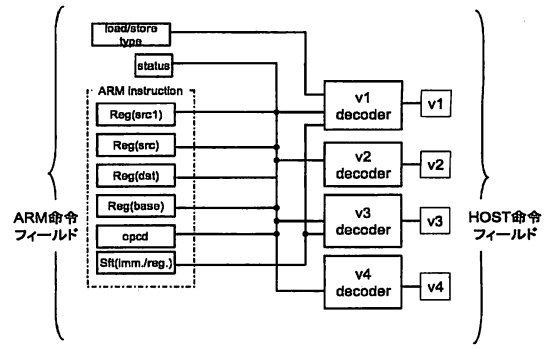

図 7 valid フラグ生成機構

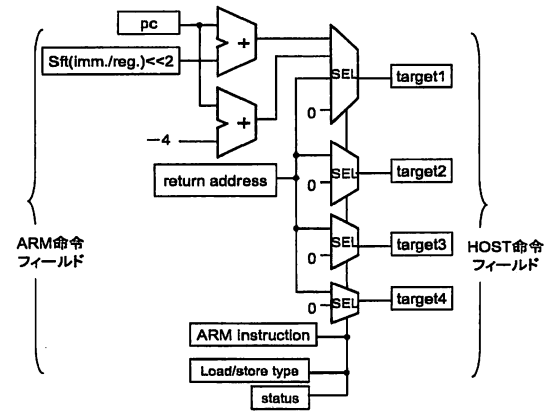

図 8 分岐アドレス生成機構

分解される命令の個数 4 を出力する．なお，レジスタ番号 1 は第 2 ソースオペランド値生成機構において加算することで生成する．最後の 3 サイクル目では分解される命令の個数 1 を出力する．このマルチプルロード／ストアは図 5 のように 1 サイクル目で補助演算，2 サイクル目以降に RISC 型のロード／ストア命令に分解する．そして最後に補助演算を行うため 3 種類があり，この種類を他の機構に伝える．

4.2.2 valid フラグ生成機構

次に，図 7 に示す valid ビット生成機構（v）について説明する．valid ビット生成機構は図 2 に示すビットフィールドに分けられた ARM 命令の値と，現在，分解の何サイクル目であるかを保持する status，そして先ほど述べたマルチプルロード／ストアの種類に基づいてデコードを行う．この結果として，命令が途中終了可能であるかを表す 2 ビットの変換値を出力する．v1 から v4 の 4 個の出力値は図 3 の decoder の 4 個の出力に対応する．

4.2.3 分岐アドレス生成機構

次に，分岐アドレス生成機構（図 8）について説明する．ARM 命令の Sft（imm./reg.）部分とプログラムカウンタ 32 ビット

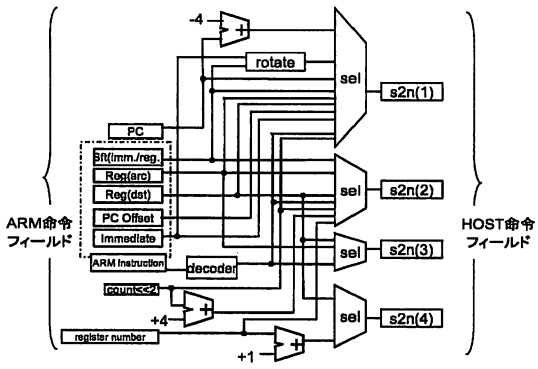

図 9　第 2 ソースオペランド値生成機構（s2n）

から分岐先アドレスを計算し 32 ビットの値を出力する演算器と，PC − 4 を計算する演算器から構成され，valid フラグ生成機構と同様に入力された ARM 命令の各ビットフィールドに基づき出力される値を選択する．このモジュールは他のモジュールに比べ入力されるビット数が多く，さらに，32 ビット加算を行う必要がある．なお，図 8 では，ARM 命令の詳細なビットフィールドについては省略している．

4.2.4　第 2 ソースオペランド値生成機構

最後に，第 2 ソースオペランド値生成機構（s2n）について説明する（図 9）．Valid フラグ生成機構や分岐アドレス生成機構と同様に，ARM 命令の各ビットフィールドに基づいて出力する．このモジュールは，入力された命令をデコードして値を出力する部分と ARM 命令の各ビットフィールドの値をそのまま出力する部分からなる．さらに入力された値を加算して出力する部分，ローテートして出力する部分がある．具体的には，前段の IF ステージから伝播されたプログラムカウンタの値 32 ビットと定数値である − 4 を加算し出力する．また，入力された 8 ビットの Immediate を 32 ビット幅で Sft(imm./reg.)*2 だけローテートし出力する．この他に，count<<2 は，マルチプルロード／ストアモジュールで計算され出力されたレジスタの個数をもとにマルチプルロード／ストア命令で用いるベースレジスタの更新値を生成する．先ほどのマルチプルロード／ストア命令の例において，8 を作る際に 4*2 を行った操作が該当する．なお，この count<<2 の値に対して 4 を加算する理由は，マルチプルロード／ストアには様々なアドレシングモードがあり，転送を開始する主記憶のアドレスが「ベースレジスター（レジスタの合計数*4+4)」となるためである．さらに，register number はマルチプルロードで使用するレジスタの値であり，連続するレジスタに対してロード／ストアを行う場合は 1 を加算し，もう 1 つのレジスタ番号を作り出す．先ほどの例では 1 番レジスタをこの加算操作を行って生成する．このように，valid フラグ生成機構に比べ出力されるパターンが複雑となる．

5.　評　　価

5.1　評価方法

論理合成ツールに Synopsys 社の 2003．03 版 Design Compiler を用い，CMOS セルライブラリに日立 0.18 μ m を使用して，モジュールの遅延時間，面積の評価を行った．Design Compiler は次のように設定した [6]．まず，各モジュールの遅延時間を知るため，組み合わせ回路に信号が入力される際の遅延と，組み合わせ回路から信号が出力される際の遅延設定を 0ns にした．このために，以下を指定した．

set_output_delay 0 -clock CLK all_outputs()
set_input_delay 0 -clock CLK all_inputs()

そして，速度方向に合成するため，以下を指定した．

compile -incremental
ungroup -all -flatten

さらに，遅延時間制約を付加するために遅延制約 2ns の際は

create_clock -p 2-w {0 1} -name CLK

として遅延制約を付加した．

同様の方法により，遅延制約 2ns，1ns，0.5ns の 3 パターンを用いた．なお，本遅延制約は Design Compiler に対して，努力を求める条件であるため，合成結果が必ずしも制約を満たすとは限らない．

5.2　測定結果

図 10 に，各モジュールの遅延時間，図 11 に，遅延制約 0.5ns 時の分岐アドレス生成機構（v）の面積を基準にした相対面積を示す．遅延制約 0.5ns の条件下では，分岐アドレス生成機構（v）は 32 ビットのキャリールックアヘッド加算器から構成される．遅延制約 2ns にて合成を行った結果，分岐アドレス生成機構 (target) に大きな遅延が生じている．この原因は，リップルキャリー加算器が合成されたためである．遅延制約を 1ns にすることで先ほど問題になっていた分岐アドレス生成機構もキャリールックアヘッド型になり 1ns に収まっている．遅延制約 0.5ns では多くのモジュールが遅延制約を満たせなくなっており，その中で遅延の大きいものは分岐アドレス生成機構 (target)，第 2 ソースオペランド値生成機構（s2n)，マルチプルロード／ストアモジュールである．面積に関しては遅延制約 0.5ns の際に第 2 ソースオペランド値生成機構がやや 1 を上回るものの，他は小さい値となっている．ところで，図 4 に示したように，一部のモジュールはマルチプルロード／ストアモジュールの出力結果を使用している．ARM 命令をデコーダに入力してから，分解されるまでの全体の遅延時間を知るために，このマルチプルロード／ストアモジュールの遅延時間を各モジュールに加えた結果を図 12 に示す．なお，この際のマルチプルロード／ストアモジュールの遅延時間は 0.5ns 制約時の値を使用した．分岐アドレス生成機構 (target)，第 2 ソースオペランド値生成機構（s2n) の遅延が他に比べると大きいことが分かる．

5.3　考　　察

分岐アドレス生成機構（target）の遅延時間が大きくなるのは，図 8 に示した加算器の存在に起因する．また，第 2 ソースオペランド値生成機構（s2n）についても，ローテートシフタ

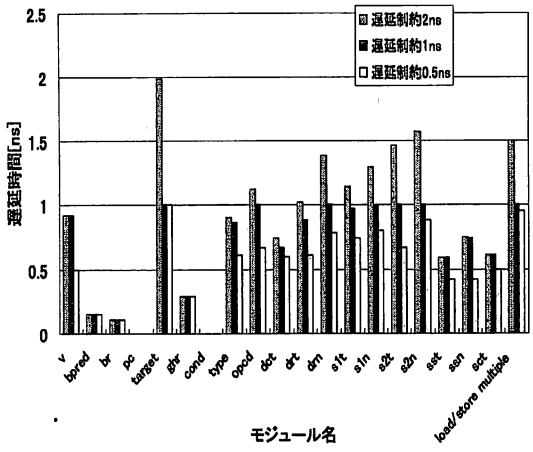

図 10　各モジュール遅延時間

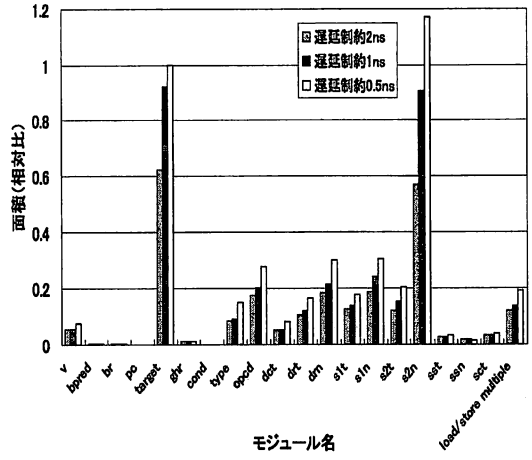

図 11　各モジュールが占める面積：遅延制約 0.5ns 時の分岐アドレス生成機構 (target) 基準での相対比

や，加算器の存在により遅延時間が大きくなっている．図 12 においてマルチプルロード／ストアと分岐アドレス生成機構の遅延時間の合計が大きくなっているが，マルチプルロード／ストアが出力した値を用いるのは分岐アドレス生成機構のセレクタ部分であるので，実際の遅延時間はより少なくなると思われる．

6. おわりに

本稿では，ARM 命令を RISC 型内部命令に変換するデコーダの構成について述べた．各モジュールの遅延時間を調べた結果，マルチプルロード／ストアモジュール，分岐アドレス生成機構，第 2 ソースオペランド値生成機構の遅延が大きいことが分かった．回路規模については，32 ビットキャリールックアヘッド加算器が 2 つで，その他は大幅に小さいことが分かった．これらのことから，デコーダ全体の遅延時間を減らすためには，まずマルチプルロード／ストアモジュールについて見直し，その後，単体のモジュールとして遅延の大きい分岐アドレス生成機構，第 2 ソースオペランド値生成機構の回路構成を考えていく必要がある．

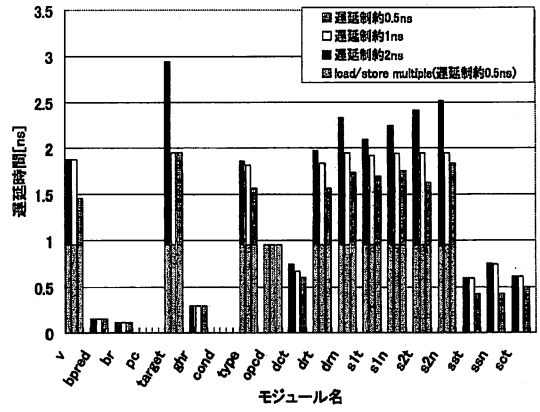

図 12　各モジュール遅延時間および 0.5ns 制約時のマルチプルロード／ストア遅延時間の合計時間

謝辞 本研究は，半導体理工学研究センターとの共同研究によるものである．加えて，東京大学 VDEC を通じて，Synopsys 社，Cadence 社，日立製作所の協力で行われたものである．

文　献

[1] B. Flachs et al., " A streaming processing unit for a CELL processor, " ISSCC Digest of Technical Papers,pp.134-135, Feb 2005.

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

[3] 中島 康彦，" ARM アーキテクチャ向け命令分解型スーパスカラ，" 情処研報 Vol.2006,No.62,2006-ARC-168,pp.77-82

[4] L.Gwennap, " Intel's P6 Uses Decoupled Superscalar Design, " Microprocessor Report,Vol.9,No.2,Feb.16,1995,pp.9-15.

[5] David B.Papworth, " Tuning the Pentium Pro microarchitecture, " IEEE Micro,April 1996,pp.8-15

[6] STARC, " STARC, " RTL 設計スタイルガイド Verilog HDL 版第 2 版，" STARC,2006