プロセッサ仕様記述からの命令依存距離抽出

平岡 佑介† 石浦菜岐佐†† 今井 正治†††

† 関西学院大学 大学院理学研究科 〒669–1337 兵庫県三田市学園 2–1
†† 関西学院大学 理工学部 〒669–1337 兵庫県三田市学園 2–1
††† 大阪大学 大学院情報科学研究科 〒565–0871 大阪府吹田市山田丘 1–5
E-mail: †scbc9086@ksc.kwansei.ac.jp, ††ishiura@ksc.kwansei.ac.jp, †††imai@ist.osaka-u.ac.jp

あらまし 本稿では，特定用途向けプロセッサ設計システム ASIP-Meister のプロセッサ仕様記述から，フォワーディングを考慮した命令依存距離の抽出法を提案する．フォワーディングは，各命令のサイクル精度の動作記述においてフォワーディングユニットを用いて記述される．本研究では，命令とフォワーディングユニットの接続関係に着目して，正しく complete なフォワーディングを定義し，この条件を満たすフォワーディングに対して命令依存距離を定式化する．さらに，命令依存距離抽出に必要となる情報を read, write 単位でクラス化し，データ量を削減する．

キーワード リターゲッタブル・コンパイラ，命令依存距離，フォワーディング，スケジューリング，ASIP-Meister

Extraction of Instruction Latency from Cycle-True Processor Models

Yusuke HIRAOKA†, Nagisa ISHIURA††, and Masaharu IMAI†††

† Graduate School of Science, Kwansei Gakuin University, 2–1 Gakuen, Sanda, Hyogo, 669–1337, Japan
†† School of Science and Technology, Kwansei Gakuin University, 2–1 Gakuen, Sanda, Hyogo, 669–1337, Japan
††† Graduate School of Information Science and Technology, Osaka University, 1–5 Yamadaoka, Suita, Osaka, 565–0081, Japan
E-mail: †scbc9086@ksc.kwansei.ac.jp, ††ishiura@ksc.kwansei.ac.jp, †††imai@ist.osaka-u.ac.jp

Abstract In this parer, we present a method of extracting instruction latency considering *forwarding* from processor specification of "ASIP-Meister." The forwarding is assumed to be specified in cycle-accurate behavior description of each instruction using forwarding units. Based on the connectivities between the instructions and the forwarding units, we define the validity and the completeness of the forwarding, and formulate the instruction latency under the existence of valid and complete forwarding. We also show a method of reducing the data amount for recording instruction latency based on a classification of the register reads and writes of the instructions.

Key words retargetable compilers, instruction latency, forwarding, scheduling, ASIP-Meister

1. はじめに

ASIP（Application Specific Instruction set Processor）は，アプリケーションに特化した命令セットにより性能やコスト，消費電力を最適化できるため，種々のディジタル機器や組み込みシステムへの搭載が拡大している．ASIP を開発する際には，プロセッサの設計に加えて，それに伴うソフトウェアツール（コンパイラ，アセンブラ，リンカ等）の開発が必要となる．また，ハードウェアの一部を変更すると，これらのソフトウェアツールにも変更が必要になることがあり，一貫性の維持や開発時間，コストが問題になる．これを解決する一手法として，プロセッサの仕様記述から，プロセッサの HDL 記述およびソフトウェアツール一式を自動生成することにより，設計の効率化や一貫性を達成するシステムの研究・開発が行われている．

設計されたプロセッサにコンパイラをリターゲッティングするためには，プロセッサの命令セットやマイクロアーキテクチャに関する様々な情報が必要となる．その中の重要な情報の一つに命令依存距離がある．コンパイラがスケジューリングを行う際，データ依存関係の存在する 2 つの命令の実行間隔を一定以上隔てなければならないという制約が生じる．本研究では，データ依存のある命令を隔てるべき最小サイクル数を命令依存距離と定義する．誤りのない最適なスケジューリングを行うためには，全命令のペアに対して命令依存距離の情報が必要になる．また，この情報はサイクル精度の命令レベルシミュレー

ションを正確に行うためにも必要になる[1].

本稿では，ASIP-Meister [2], [3] のプロセッサ仕様記述から命令依存距離を自動抽出する方法を述べる．ASIP-Meister は，大阪大学で開発されている ASIP 開発環境であり，プロセッサ仕様記述からプロセッサの HDL とそのソフトウェアツールの自動生成が可能である．現在 RISC 型だけでなく VLIW 型プロセッサの設計が行えるように，合成系[2]，シミュレータ[3]，コンパイラ生成系の開発が進められている．

命令依存距離の情報は，他の様々なリターゲッタブルコンパイラでも必要になっている．VLIW アーキテクチャに対して優れた開発環境を持つ ISDL（Instruction Set Description Language）[4] では命令の全組み合わせに対して命令依存距離を人手で記述しなければならない．EPIC アーキテクチャに対するコンパイラのリターゲッティングが可能な Trimaran (注1) でも同様に人手による記述が必要になる．LISA [1] は本手法と同じくプロセッサ仕様記述から命令依存距離を自動的に抽出する．しかし，命令依存距離はレジスタの書き込みサイクルと読み出しサイクルからのみ計算されており，フォワーディング（データの先送りをしてデータハザードを回避する手法）の考慮はなされていない．フォワーディングは，余分なハードウェアコストと消費電力を要するが，高速化の効果も大きいため，設計の一オプションとして不可欠と考えられる．

本研究では，ASIP-Meister のプロセッサ仕様記述（各命令のサイクル精度の動作記述）から，フォワーディングまで考慮して命令依存距離を抽出する方法を提案する．まず，フォワーディングユニットとの接続関係から，正しいフォワーディング，complete なフォワーディングの定式化を行い，これに基づきフォワーディングまで考慮した命令依存距離の計算式を導出する．正しいフォワーディングの定式化はフォワーディング回路の設計誤りの検出にも利用することができる．また，レジスタ書き込み/読み出し情報に基づいて，命令のクラス化を行うことにより，命令依存距離情報の必要記憶量を削減する方法を示す．

以下 2 章では，ASIP-Meister のプロセッサ仕様記述例を紹介し，フォワーディングが無い場合の命令依存距離の定式化を行う．3 章では，フォワーディングユニットを用いたフォワーディングの動作記述について述べ，フォワーディングを考慮した命令依存距離の定式化を行う．4 章では，レジスタアクセスのクラス化に基づくデータ構造の効率化について述べ，5 章では，本研究で提案した手法に従った実装結果，6 章でまとめと今後の課題について述べる．

2. 命令の動作記述と命令依存距離

2.1 ASIP-Meister [2], [3] とそのプロセッサ仕様記述

現在大阪大学で開発中の ASIP-Meister は，特定用途向けプロセッサ開発環境であり，プロセッサの仕様記述からプロセッサの自動合成と同時に，ソフトウェアツールの自動生成を行う．現在，VLIW 型のプロセッサの設計が行えるよう拡張が行われている．プロセッサの仕様記述は次の構成要素から成る．

（1）宣言部

メモリ，レジスタ，演算器等のハードウェア資源や命令フォーマットの宣言を行う．

（2）動作定義部

各命令の動作を C-like な構文で記述する．この情報は命令セットシミュレータやコンパイラの命令パターン生成 [1] に用いられる．

（3）マイクロ動作定義部

各命令のサイクル精度の動作を記述する．この情報はプロセッサ合成系への入力となる．

本研究では，(3) のマイクロ動作記述から命令依存距離の抽出を行う．図 1 は，フォワーディングを考慮しない場合のマイクロ動作記述である．この記述は，ハードウェア資源グループ RG02ALU を用いて実行される ADD 演算の動作を表したものである．stage 2（6 ～ 9 行目）では，レジスタファイル GPR の rs0 番目と rs1 番目からデータを読み出している．stage 3（10 ～ 13 行目）では stage 2 で取得したオペランドに対して演算器 ALU1 で add 演算を行い，stage 4（14 ～ 16 行）で演算結果を GPR の rd 番目に書き込んでいる．

```
 1:  micro_operation ADD on RG02ALU
 2:  {
 3:     wire [31:0] source0;
 4:     wire [31:0] source1;
 5:     wire [31:0] result;
 6:     stage 2 {
 7:       source0 = GPR.read2(rs0);
 8:       source1 = GPR.read3(rs1);
 9:     };
10:     stage 3 {
11:       wire [3:0] flag;
12:       <result, flag> = ALU1.add(source0,source1);
13:     };
14:     stage 4 {
15:       null = GPR.write1(rd,result);
16:     };
17:  };
```

図 1 マイクロ動作記述（フォワーディングなし）

2.2 命令依存距離

プロセッサは複数の命令スロットを持つ VLIW 型プロセッサ(注2)で，各命令（演算）に対しその命令が発行されるスロットは一意に決まっているものとする．各命令はレジスタに対して複数の read と write を行う．各 read と write は，それぞれ何サイクル目でどのレジスタにアクセスするかの情報を持つ．以下にその表記を列挙する．

I: 命令の集合

$i \in I$ に対し，

- $i.slot$: 命令 i が発行されるスロット
- $i.read$: 命令 i の read の集合
- $i.write$: 命令 i の write の集合

(注1)：http://www.trimaran.com

(注2)：RISC 型プロセッサは特殊ケース（スロット数 1）として扱う

$r \in i.read$ に対し,

$r.reg$: r が read するレジスタ

$r.cycle$: r が レジスタを read するサイクル

$w \in i.write$ に対し,

$w.reg$: w が write するレジスタ

$w.cycle$: w がレジスタへ write するサイクル

データ依存には, RAW (Read After Write), WAR (Write After Read), WAW (Write After Write), の 3 種類存在し, それぞれに関する命令依存距離が定義される.

(1) RAW 依存

図 2 の (a) は命令 i の write w の結果を命令 j の read r が参照する状況を表す. 命令 j が r で正しい値を読み込むためには, j の実行開始を i の実行開始よりも, あるサイクル数以上遅らせる必要がある. このサイクル数 $d_{w,r}(i,j)$ が命令 i と j の w と r に関する命令依存距離であり, フォワーディングを考慮しない場合の $d_{w,r}(i,j)$ は, 次式で与えられる.

$$d_{w,r}(i,j) = w.cycle - r.cycle + 1.$$ (注3)

(2) WAR 依存

図 2 の (b) において, 命令 i と j がこの順にスケジューリングされているとする. i の read r は, j の write w によって REG が更新される前に実行しなければならない. このハザードを回避するための i と j の命令依存距離 $d_{r,w}(i,j)$ は

$$d_{r,w}(i,j) = r.cycle - w.cycle$$ (注4)

で与えられる.

(3) WAW 依存

図 2 の (c) において, 命令 i と j がこの順にスケジューリングされているとする. i の write w_1 は j の write w_2 によって REG が更新される前に実行されなければならない. このハザードを回避するための i と j の命令依存距離 $d_{w_1,w_2}(i,j)$ は

$$d_{w_1,w_2}(i,j) = w_1.cycle - w_2.cycle + 1$$

で与えられる.

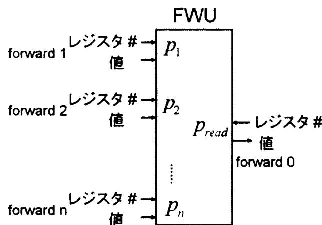


図 3 フォワーディングユニット

(注3) : ASIP-Meister は単相クロックを仮定しているため, w から r まで 1 サイクル必要となる. r と w が同一サイクルで行える場合は +1 は不要.

(注4) : 単相クロックをモデルでは, r と同じサイクルで w が行える. r と w が同一サイクルで行えない場合は +1 が必要

3. フォワーディングを考慮した命令依存距離

3.1 フォワーディングを行う命令の動作記述

フォワーディングはフォワーディングユニットを用いて実現することができる. 図 3 に本研究で考えるフォワーディングユニット (FWU) を示す. FWU は, read 側にポート p_{read} と, write 側に複数のポート $p_1, \cdots, p_n$ を持つ. write 側のポートには, フォワードされるレジスタ番号と値の組が入力される. この中に read 側から入力されたレジスタ番号に一致するものがあれば, そのデータが read 側に出力される. レジスタ番号の等しい write 側のポートが複数存在するときには, $p_1, p_2, \cdots, p_n$ の順に優先される.

図 4 はフォワーディングを考慮したマイクロ動作記述である. stage 2 (6 ~ 13 行目) でレジスタファイル GPR の rs0 番目, rs1 番目のデータをそれぞれ source0, source1 に読み込むが, この際にフォワーディングユニット FWU2, FWU3 からフォワーディングがあれば, そのデータを優先して読み込む. stage 3 (14 ~ 21 行目) で ALU1.add による演算結果 result を stage 4 (22 ~ 28 行目) で GPR の rd 番目に書き込むが, 演算結果が得られる stage 3 からレジスタ書き込みが行われる stage 4 の間, FWU0, …, FWU4 を通じて演算結果をフォワーディングする.

```
 1: micro_operation ADD on RG02ALU
 2: {
 3:   wire [31:0] source0;
 4:   wire [31:0] source1;
 5:   wire [31:0] result;
 6:   stage 2 {
 7:     wire [31:0] tmp0;
 8:     wire [31:0] tmp1;
 9:     tmp0 = GPR.read2(rs0);
10:     tmp1 = GPR.read3(rs1);
11:     source0 = FWU2.forward(rs0,tmp0);
12:     source1 = FWU3.forward(rs1,tmp1);
13:   };
14:   stage 3 {
15:     wire [3:0] flag;
16:     <result, flag> = ALU1.add(source0,source1);
17:     null = FWU0.forward2(rd,result);
18:     null = FWU1.forward2(rd,result);
19:     null = FWU2.forward2(rd,result);
20:     null = FWU3.forward2(rd,result);
21:   };
22:   stage 4 {
23:     null = FWU0.forward4(rd,result);
24:     null = FWU1.forward4(rd,result);
25:     null = FWU2.forward4(rd,result);
26:     null = FWU3.forward4(rd,result);
27:     null = GPR.write1(rd,result);
28:   };
29: };
```

図 4 マイクロ動作記述 (フォワーディングあり)

3.2 フォワーディングの定式化

本研究では, 前節で述べた FWU を用いてフォワーディングが記述されていると仮定する. すると, $i \in I$, $r \in i.read$,

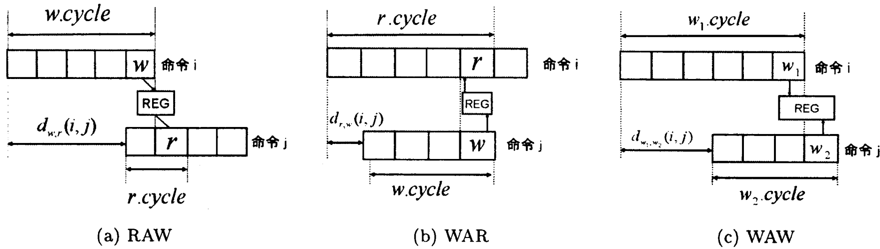


図 2　3 種類のデータ依存と命令依存距離

$w \in i.write$ なる r, w に対して，フォワーディングは次のように定式化できる．

$r.fwu$:　r が read する FWU

$r.fwcycle$:　r が FWU を read するサイクル

$w.fw$:　w のフォワーディングの集合

$f \in w.fw$ に対し，

$f.fwu$:　f がデータ転送する FWU

$f.port$:　f がデータ転送る FWU のポート

$f.cycle$:　f がデータ転送するサイクル

w に対し，最も早くフォワーディングが行われるサイクルを $w.fwcycle$ とする．また，FWU の write 側のポート p_m に対し，$F(p_m)$ を次のように定義する．

$$F(p_m) = \{(w,k) \mid w \in \bigcup_{i \in I} i.write \text{ s.t. } f \in w.fw \text{ s.t. } f.port = p_m,\ k = f.cycle\}.$$

例えば図 5 において，

$$F(p_3) = \{\ (w_1, 4), (w_2, 4)\ \}$$

であるが，これは p_3 へのフォワーディングは w_1 の 4 サイクル目と w_2 の 4 サイクル目から行われていることを表す．

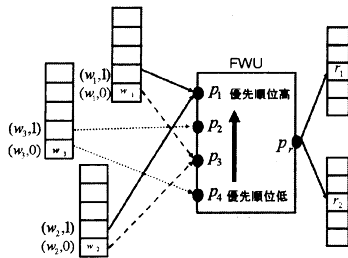


図 5　複数の命令からのフォワーディング

3.3　正しいフォワーディング

マイクロ動作記述では，自由度の高い設計が可能である反面，設計に誤りが生じる可能性がある．誤ったフォワーディングに対しては，矛盾の無い命令依存距離が定義できないため，まず「正しいフォワーディング」を定義する．これは，フォワーディングの設計誤りの検出にも利用することができる．正しいフォワーディングは（1）valid と（2）conflict free の 2 条件により定義される．

定義 1．フォワーディングが valid

任意の命令の write w_1, w_2 に対し次が成立するとき，このプロセッサのフォワーディングは valid であるという．

$(w_1, k) \in F(p_i)$ かつ $(w_2, l) \in F(p_j)$ かつ

$l - w_2.fwcycle > k - w_1.fwcycle$

ならば $i \geq j$.

これは，フォワーディングの優先順位が守られることを表す．フォワーディングの優先順位は，任意の write に対し最も早いサイクルで行われるフォワーディングが最も高く，そのサイクルから遅れるほど低くなる．図 6 の例では複数の write 命令 w_1, w_2 に対するフォワーディングを示している．例えば w_1 の p_3 へのフォワーディングは w_1 のフォワーディングの中で 2 番目の優先順位となる．このとき，p_3 よりも優先順位の高いポートに接続される write は，優先順位が 2 番目以上（2 または 1）でなければならない．この条件が満たされないと，データが更新されているにも関わらず，更新前の古いデータがフォワーディングされてしまう．

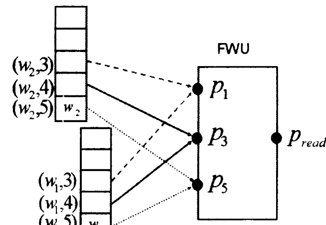


図 6　valid なフォワーディング

定義 2．フォワーディングが conflict free

任意のフォワーディングユニットの任意の書き込みポート p_m に対して次が成立するとき，このフォワーディングは conflict free であると言う。

$F(p_m) = \{(w_1, k_1), (w_2, k_2), \cdots, (w_t, k_t)\ \}$,

$w_r \in i_r.write$ $(r = 1, 2, \cdots, t)$ とするとき，

$i_1.slot = i_2.slot = \cdots = l_t.slot$ かつ

$k_1 = k_2 = \cdots = k_t$.

これは，同一サイクルで同一ポートへのフォワーディングが存在しないことを表す．例えば図 7 の例で FWU の p_1 ポートに対して，i_1 と i_2 からフォワーディングがあるが，i_1 と i_2 の発行されるスロットが同じで，フォワーディングを行うサイクル

が同じであれば，同時にデータが転送されることはない．

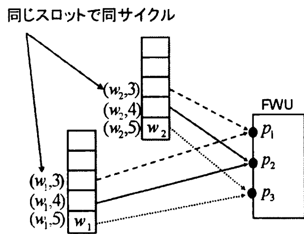
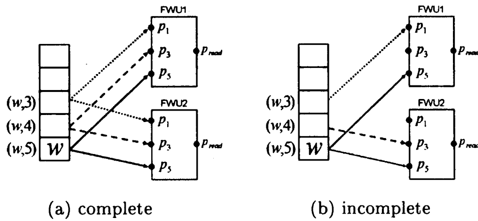


図 7 conflict free なフォワーディング

定義 3. 正しいフォワーディング

valid かつ conflict free なフォワーディングを「正しいフォワーディング」と言う．



(a) complete (b) incomplete

図 8 complete なフォワーディング

3.4 complete なフォワーディング

定義 4. フォワーディングが complete

任意の命令の write w に対し，次が成立するときこのプロセッサのフォワーディングは complete であるという．

$f \in w.fw, f.fwu = u$ ならば,

$w.fwcycle \leqq k \leqq w.cycle$ なる全ての k に対し,

$\exists f \in w.fw$ s.t. $f.cycle = k, f.fwu = u$.

例えば，図 8 (a) のフォワーディングは，全てのフォワーディングユニットに対し，最も早くフォワーディングされるサイクルが等しく，さらに毎サイクルフォワーディングが行われているので complete である．図 8 (b) のように FWU によって最も早くフォワーディングされるサイクルが異なったり，あるサイクルでフォワーディングを行わないものは incomplete なフォワーディングである．ハードウェアのコストの観点から，incomplete なフォワーディングの実装も提案されているが [5]，本研究は complete なフォワーディングのみを考える．

3.5 フォワーディングを考慮した命令依存距離

正しく complete なフォワーディングに対して，フォワーディングを考慮した命令依存距離を以下のように定義できる．

(1) RAW 依存

命令 i の write w と 命令 j の read r に対し，$d_{w,r}(i,j)$ は次式により与えられる．

$$d_{w,r}(i,j) = \begin{cases} w.fwcycle - r.fwcycle \\ \quad (\exists f \in w.fw[f.fwu = r.fwu] \text{ のとき}), \\ w.cycle - r.cycle + 1 \text{ (それ以外のとき)}. \end{cases}$$

(2) WAR 依存

命令 i の read r と 命令 j の write w に対し，$d_{r,w}(i,j)$ は次式により与えられる．

$$d_{r,w}(i,j) = \begin{cases} r.fwcycle - w.fwcycle + 1 \\ \quad (\exists f \in w.fw[f.fwu = r.fwu] \text{ のとき}), \\ r.cycle - w.cycle \text{ (それ以外のとき)}. \end{cases}$$

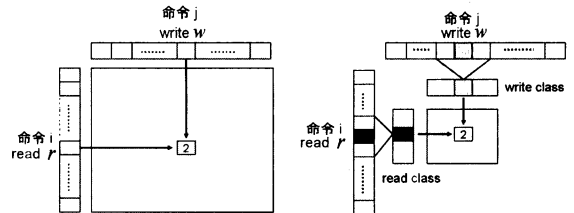
(3) WAW 依存

命令 i の write w_1 と 命令 j の write w_2 に対し，$d_{w_1,w_2}(i,j)$ は次式により与えられる．

$$d_{w_1,w_2}(i,j) = \begin{cases} w_1.fwcycle - w_2.fwcycle + 1, \\ (\exists f_1, f_2[f_1 \in w_1.fw, f_2 \in w_2.fw, f_1.fwu = f_2.fwu] \text{ かつ} \\ w_1.fwcycle - w_2.fwcycle > w_1.cycle - w_2.cycle \text{ のとき}), \\ w_1.cycle - w_2.cycle + 1 \text{ (それ以外のとき)}. \end{cases}$$

4. データ構造の効率化

命令依存距離の抽出結果は，図 9 (a) のように全命令の write と read に対する 2 次元の表となる．命令数の多いプロセッサでは，このデータ量は無視できない大きさとなるので，本研究では，命令依存距離算出の際に必要となる情報が共通する read, write をクラス化することで，データ量の削減を図る．即ち，図 9 (b) に示すように，2 次元の表は read と write のクラスに対してのみ記録する．命令 i の write w から命令 j の read r の命令依存距離は，w が属するクラスと r が属するクラスをそれぞれ求めた後に，表を参照することにより求める．



(a) クラス化を行わないデータ構造 (b) クラス化を行ったデータ構造

図 9 命令依存距離のデータ構造

(1) read のクラス化

2 つの read r_1, r_2 は次を満たすとき同じクラスに属する．

$r_1.reg = r_2.reg$,

$r_1.cycle = r_2.cycle$,

$r_1.fwu = r_2.fwu$,

$r_1.fwcycle = r_2.fwcycle$.

同じクラスに属する read から，同一の write に対する命令依存距離は等しくなる．

(2) write のクラス化

2 つの write w_1, w_2 は次を満たすとき同じクラスに属する．

$w_1.reg = w_2.reg$,

$w_1.cycle = w_2.cycle$,

$w_1.fwcycle = w_2.fwcycle.$

同じクラスに属する read から，同じクラスに属する wirte に対する命令依存距離は等しくなる．

5. 実験結果

5.1 実装と実験結果

以上の定義に基づき，ASIP-Meister のプロセッサ仕様記述から命令依存距離を抽出する処理系を作成した．実装は，Windows XP の cygwin 上に Perl 5.8.0 により行った．

RISC 型 DLX を VLIW 型に拡張したプロセッサの仕様記述に対し，RAW の命令依存距離の抽出を行った実験結果を表 1 に示す．このプロセッサは 2 スロット，102 命令を持ち，フォワーディング回路を持つ 4 ステージパイプラインにより実装され，仕様記述は 3952 行から成る．クラス化を行わない場合，read の数は 156，write の数は 98 となった．これに対し，クラス化した read の数と write の数は共に 5 となり，データ量を大幅に削減することができた．

表 1 データ量の比較

評価項目	クラス化なし	クラス化
read	156	5
write	98	5
組み合わせ	13,236	17

5.2 フォワーディングの検証

本処理系では命令依存距離の抽出に先立ち，フォワーディングが正しいかどうかの検証を行うので，この時点でフォワーディングに関する設計誤りの検出が行える．具体的には

- フォワーディングを行うフォワーディングユニットのポートの優先順位が誤っていないか，
- フォワーディングユニットの同一ポートに対して，同時に異なる命令からフォワーディングが行われていないか，

が検証できる．また，フォワーディングが complete かどうかの検査により，意図しないフォワーディングの「抜け」がないかどうかもチェックすることができる．

さらに read/write のクラス化の結果からもフォワーディングの設計ミスがないかどうかの検証が行える．例えば，図 10 は read/write のクラス化の結果を表示したものであるが，"SUB on RG01ALU" だけが他の ALU 命令と異なる write 情報を持つクラスに分類されてしまっているが，これが意図しないものであれば，フォワーディングに何らかの記述不足があったと判断することができる．

```
 1:  write class 1
 2:  fw_min_cycle:  3
 3:    register:  GPR
 4:  operations
 5:    ADD on RG01ALU.write2
 6:    ADDU on RG01ALU.write1
 7:    SUBU on RG01ALU.write2
                :
 8:             :
 9:  write class 2
10:  fw_min_cycle:  4
11:    register:  GPR
12:  operations
13:    SUB on RG01ALU.write2
```

図 10 設計を誤ったクラス化

6. まとめと今後の課題

本稿では，プロセッサ仕様記述から，フォワーディングを考慮した RAW，WAR，WAW 全ての命令依存距離の抽出法を提案し，さらにデータ構造の効率化について述べた．本手法を用いれば，与えられた動作記述に対して，フォワーディングの検証を行った上で命令依存距離を抽出できる．

本研究では，complete なフォワーディングを扱ったが，ハードウェアのコストと性能のトレードオフを追及する手段として incomplete なフォワーディングの実装が提案されている [5]．EXPRESSION [6] では，命令依存距離ではなくオペレーションテーブルの拡張によりこのようなフォワーディングの下でのスケジューリングを行う手法を提案しているが，RAW 依存だけを扱っており，WAR 依存や WAW 依存は考慮していない．incomplete なフォワーディングに関しては，その効果を見極めながら効率的なスケジューリングアルゴリズムも含めて，これを扱う方法を考えていく予定である．

謝　辞

本研究を進めるにあたり，有益な御助言，御指導を頂きました大阪大学 の武内良典助教授はじめ今井研究室の関係諸氏に感謝致します．最後に，本研究に際し，御討論頂き，また，種々の面でお世話になりました関西学院大学石浦研究室の関係諸氏に感謝致します．

文　献

[1] O. Wahlen, M. Hohenauer, G. Braun, R. Leupers, G. Ascheid, H. Meyr, and X. Nie: "Extraction of Efficient Instruction Schedulers from Cycle-True Processor Models," in *Proc. SCOPES 2003*, pp. 167–181, Sept. 2003.

[2] 小林悠記, 小林真輔, 坂主圭史, 武内良典, 今井正治: "コンフィギュラブル VLIW プロセッサの HDL 記述生成手法," 情報処理学会論文誌, vol. 45, no. 5, pp. 1311–1321, May 2004.

[3] K. Okuda, S. Kobayashi, Y. Takeuchi, and M. Imai: "A Simulator Generator Based on Configurable VLIW Model Considering Synthesizable HW Description and SW Tools Generation," in *Proc. SASIMI 2003*, pp. 152–159, Apr. 2003.

[4] G. Hadjiyiannis, S. Hanono, and S. Devadas: "ISDL: An Instruction Set Description Language for Retargetability," in *Proc. DAC*, pp. 299–302, June 1997.

[5] P. Ahuja, D. W. Clark, and A. Rogers: "The Performance Impact of Incomplete Bypassing in Processor Pipelines," in *Proc. MICRO-28*, pp. 36–45, Nov. 1995.

[6] A. Shrivastava, E. Earlie, N. Dutt, and A. Nicolau: "Operation tables for Scheduling in the Presence of Incomplete Bypassing," in *Proc. ISSS/CODES 2004*, pp. 194–199, Sept. 2004.