社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# リターゲッタブル・コンパイラのための命令パターン生成

岸本 充司† 石浦菜岐佐†† 益井 勇気†† 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: †scbc9044@ksc.kwansei.ac.jp, ††{ishiura,bux65238}@ksc.kwansei.ac.jp, † † †imai@ist.osaka-u.ac.jp

**あらまし** 本稿では，ASIP 設計システム ASIP Meister の各命令の C-like な動作記述からコンパイラのリターゲッティングに必要な命令パターンの生成を行う手法を提案する．さらに，プロセッサで実行可能な命令とコンパイラに必要な命令パターンの差を埋めるため，プロセッサ設計上の 1 命令からコンパイラに必要な複数の命令パターンを生成する手法を提案する．本研究のシステムでは，VLIW アーキテクチャやマルチサイクル演算にも対応している．

**キーワード** リターゲッタブル・コンパイラ，動作記述，命令パターン，コンフィギュラブルプロセッサ

# Instruction Pattern Generation for Retargetable Compiler

Atsushi KISHIMOTO†, Nagisa ISHIURA††, Yuuki MASUI††, and Masaharu IMAI†††

† Graduate School of Science, Kwansei Gakuin University 2–1 Gakuen, Sanda, Hyogo, 669–1337, Japan
†† 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, 565–0871, Japan
E-mail: †scbc9044@ksc.kwansei.ac.jp, ††{ishiura,bux65238}@ksc.kwansei.ac.jp, † † †imai@ist.osaka-u.ac.jp

**Abstract** This paper presents a method of generating instruction patterns for retargetable compilers from ASIP Meister's C-like behavioral description of a processor instruction set. We also propose a scheme of extracting multiple instraction patterns necessary for compilers from a sigle instruction description. Our system supports VLIW architectures as well as multicycle operations.

**Key words** retargetable compiler, behavioral description, instruction pattern, configurable processor

## 1. まえがき

近年，急速な普及を遂げているディジタル音響・映像器機には，汎用の信号処理プロセッサ（DSP） を搭載する場合もあるが，より高性能で低価格，低消費電力のシステムを実現するために，応用に特化した専用のプロセッサを新たに開発して用いる場合が増えてきた．具体的には Xtensa (注1)や MEP 等のカスタムコンフィギュラブルプロセッサや大阪大学で開発された ASIP Meister [?], [?] 設計システムなどが挙げられる．このようなカスタムプロセッサに対し，設計の都度コンパイラを開発するのは非現実的なため，C 言語などのプログラムとプロセッサのアーキテクチャ記述からそのプロセッサ用のコードを生成する「リターゲッタブル・コンパイラ」[?] の研究が重要になってきている．

リターゲッタブル・コンパイラのバックエンドは，一般に命令選択，バインディング，スケジューリングの 3 フェーズで構成される．命令選択フェーズではコンパイラフロントエンドで生成されたマシン独立な中間コードの命令をターゲットプロセッサが実行可能な命令に変換する．このためには，プロセッサが実行可能な全命令に関する情報を記述した「命令パターンテーブル」が必要になる．一般に GCC (注2) 等のリターゲッティングが可能なコンパイラでは，命令パターンテーブルをプロセッサ設計ごとに記述しなければならないが，これを人手で行うには膨大な作業が必要になる．Dortmund 大学で開発された RECORD [?], [?] リターゲッタブル・コンパイラでは，プロセッサアーキテクチャを MIMOLA HDL で記述し，そのハードウェア記述からプロセッサで実行可能な命令セットの抽出が可能である．各モジュールに可能な全ての入力を与え，そこか

(注1)：http://www.tensilica.com

(注2)：http://gcc.gnu.org/

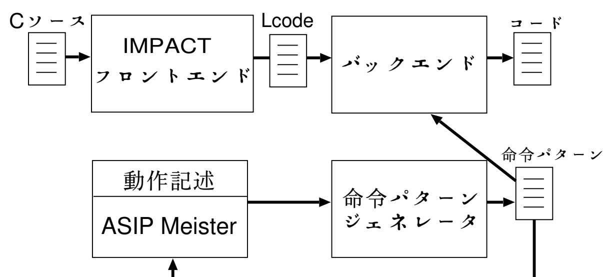



図 1　リターゲッタブル・コンパイラの構成

ら生成される RT の組み合わせから木パターンの命令セットを抽出する．しかしこのシステムでは，フロントエンドは DFL（Data-flow Language）という言語を採用しており C 言語等のプログラムを入力することはできない．また，パターンの生成にはデータパスに関する詳細な記述が必要であり，マルチサイクル演算に対応していないなどの問題点もある．

本研究では，プロセッサの命令の動作記述から命令パターンを抽出する手法を提案する．本研究で開発しているリターゲッタブル・コンパイラは，より一般性のある C フロントエンドを採用し，マルチサイクル演算にも対応している．さらに，プロセッサの 1 命令からコンパイラに必要な複数の命令パターンを抽出することが可能である．

本稿では，2 章で本研究におけるリターゲッタブル・コンパイラシステムの構成，およびコード生成手法と本研究の関わりについて述べる．3 章では，命令選択で必要となる命令パターンについて述べる．4 章では，命令パターン DAG の構築手法およびコンパイラに必要な複数命令パターンの生成手法について述べる．5 章では，本稿で提案する手法により命令パターンを生成した実験結果について述べる．6 章では，まとめと今後の課題について述べる．

## 2. リターゲッタブル・コンパイラの構成

本研究で開発を進めているコンパイラの構成を図 ?? に示す．コンパイラのフロントエンドは，イリノイ大学で開発された IMPACT (注3)である．IMPACT は K&R や ANSI-C で記述された C ソースコードに機械独立な最適化を行い，Lcode と呼ばれる機械独立な中間コードを生成する．プロセッサアーキテクチャには特定用途向けプロセッサ設計システム ASIP Meister [?], [?] の動作記述を用いる．この記述では C-like なシンタックスにより RISC 型や VLIW 型プロセッサの命令の動作を記述することができる．

VLIW プロセッサのコード生成は次の 3 フェーズで行う．

（1） 命令選択
（2） バインディング
（3） スケジューリング

命令選択では図 ?? のように，フロントエンドで生成された機械独立な中間コードを，ターゲットプロセッサが実行できる命令パターンで被覆する問題を解くことにより，機械依存のコードに変換する．中間コードは DFG（Data-flow Graph）であり，ターゲットプロセッサで実行可能な命令パターンを列挙したものがパターンテーブルである．バインディングでは命令選択が行われたコードに，実際に演算が実行される演算器や値の格納されるレジスタなどの物理資源の割り当てを行う．スケジューリングではデータの依存関係や演算器およびレジスタの資源制約を考慮して演算の並列化や実行順序の決定を行う．この 3 フェーズを経てターゲットプロセッサ用のコードが生成される．

(注3)：http://www.crhc.uiuc.edu/Impact/

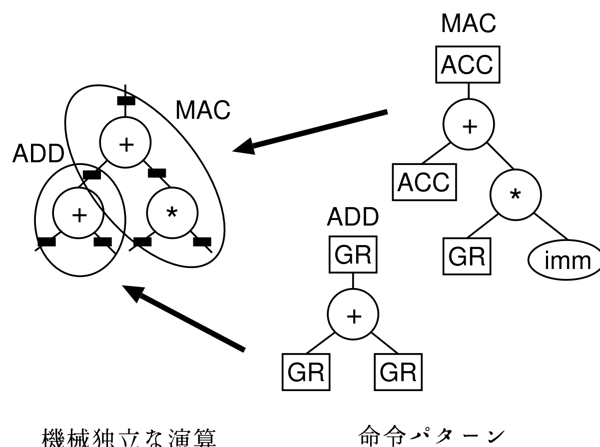



図 2　命令選択

同じ演算であっても，アドレッシングモードや条件付き実行の修飾が異なると，コンパイラにとっては異なる命令パターンとなる．また，コンパイラには加算命令の一方のオペランドを 0 にしてレジスタ間転送を行うというような命令パターンも必要である．そのため，命令パターン数が非常に多くなり手書きで記述すると作業が膨大になる．本研究の命令パターン生成手法は，命令の動作記述から命令パターンを自動抽出するとともに，コンパイラに必要な複数の命令パターンを生成することを狙っている．

## 3. 命令パターンとそのプロセッサ動作記述からの生成

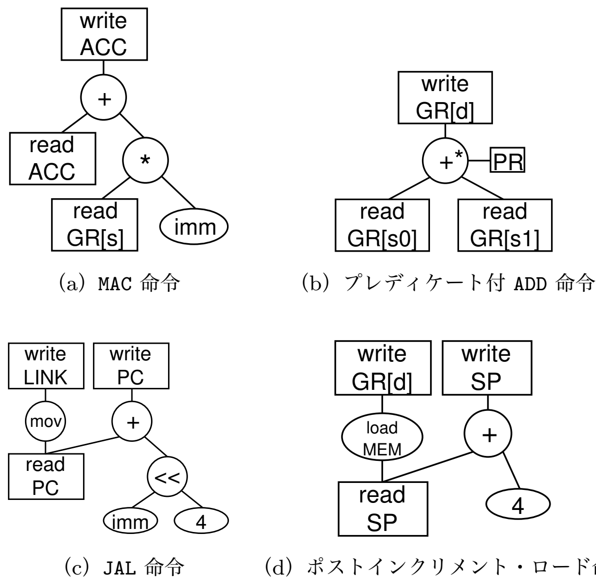
### 3.1 命令パターン

命令パターンは，木構造で表現されることが多いが，本研究ではより一般的な DAG（非巡回有向グラフ）を用いる．枝（下から上へ向かうとする）はデータの流れを表し，節点は以下のようなプリミティブな動作を表す．

- レジスタ (注4)の参照
- レジスタへの書き込み
- オペランドフィールドの参照
- 定数の参照
- 演算の実行（メモリアクセスを含む）

例えば，図 ?? (a) はプロセッサの `MAC` 命令を表現したものであり，`GR[s]` と即値 `imm` の積を求め，その値とアキュムレータ

(注4)：以降の例では混乱のない限り read, write やレジスタファイルのインデックスは省略する




(a) MAC 命令　(b) プレディケート付 ADD 命令

(c) JAL 命令　(d) ポストインクリメント・ロード命令

図 3　命令パターン例

ACC の値の和を ACC に書き込むという動作を表す．図 ??（b）はプレディケート付の ADD 命令の動作を表したもので，プレディケートレジスタ PR の値が真ならば GR[s0] と GR[s1] の値の加算結果を GR[d] に書き込み，PR の値が偽ならば何も行わないという命令の動作を表す．これらの命令は，1 命令が 1 つのレジスタ書き込みしか含まないため木で表現可能であるが，1 命令中に複数のレジスタ書き込みが含まれている場合には表現に DAG が必要になる．図 ??（c）の表す JAL 命令は LINK レジスタに PC レジスタの値を代入し，続く動作で imm を 4 ビット左シフトした値と PC レジスタの値の和を PC レジスタに書き込む．図 ??（d）は ポストインクリメント・ロードの動作を表したもので，SP レジスタの指すメモリの値を GPR[d] にロードし，続く動作で SP レジスタの値に 4 を加える．

### 3.2　プロセッサ動作記述からの命令パターン生成

ASIP Meister の動作記述では，各命令の動作を C-like な構文で記述する．図 ??（a）は MAC 命令の動作記述例である．s, imm は命令のオペランドフィールドを表す．動作記述は，構文解析後図 ??（b）のような解析木に変換される．ここから命令選択フェーズで必要となる図 ??（c）のような木構造の命令パターンを求めるのが命令パターン生成の処理である．図 ??（a）はプレディケート付命令の動作記述例である．この場合には，図 ??（c）のようなプレディケート付加算命令 (”+*”) の命令パターンを生成する必要がある．

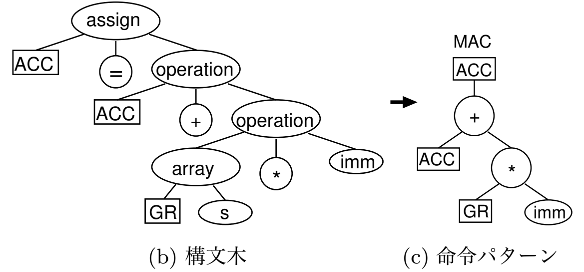
### 3.3　複数の命令パターン生成

プロセッサ設計上は 1 命令であっても, コンパイラではそれを複数の命令として扱わなければならない場合がある．

例えば，汎用レジスタ間のデータ転送は，独立した命令として設計されるのではなく，即値 0 やゼロレジスタを用いた加算命令で実現されることが多い．この場合, ADD 命令の記述から単純に加算の命令パターン 1 つの抽出を行っただけでは，コンパイラに必要なデータ転送命令のパターンが欠如することになってしまう．そこで, 図 ?? のように ADD 命令から通常の加算命

```
behavior MAC (s, imm) {
  ACC = ACC + GR[s] * imm;
}
```

(a) 動作記述例

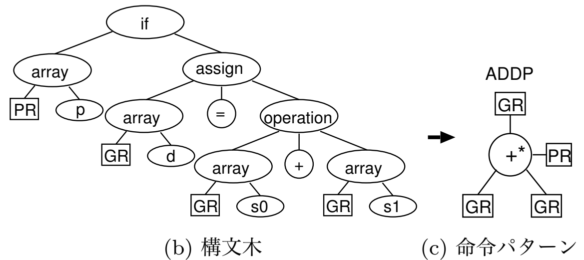



(b) 構文木　(c) 命令パターン

図 4　MAC の命令パターン生成

```
behavior ADDP (p, d, s0, s1) {
  if (PR[p]) {
    GR[d] = GR[s0] + GR[s1];
  }
}
```

(a) 動作記述例



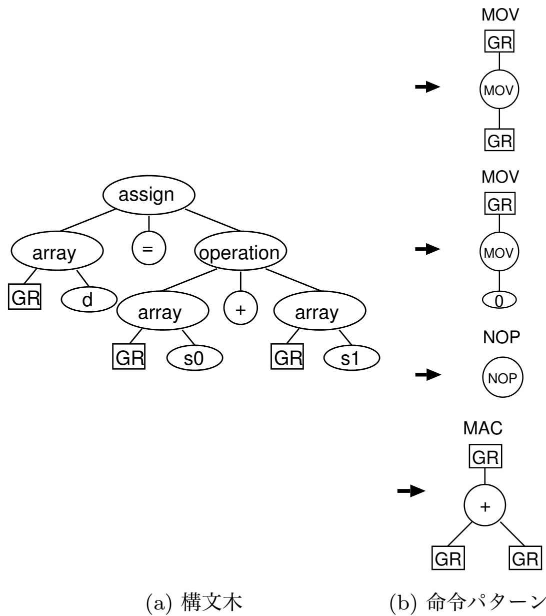

(b) 構文木　(c) 命令パターン

図 5　プレディケート付 ADD の命令パターン生成

令のパターンに加え，s0, s1 のどちらかを 0 にして (注5) データ転送を行う命令パターンを生成する必要がある．さらに，s0, s1 のどちらも 0 にして 0 を転送する命令や，d が ゼロレジスタを指す NOP 命令を生成する必要がある．

また，プレディケート付命令の場合は，プレディケート修飾を持つものと持たないものを独立した命令として設計するのではなく，図 ?? のようにプレディケート修飾なしが特殊ケースであるような 1 つの命令として設計することが多い．この場合には図 ?? を解析した結果の図 ?? の構文木において，図 ?? のように p を 0 と非 0 にして場合分けを行い，さらに d, s0, s1 についてもそれぞれ 0 と非 0 にすることで場合分けを行い，ADD や MOV に加えプレディケート付の ADD* や プレディケート付の MOV* に NOP と 2 つの ゼロを転送する命令を加えた，7 通りの命令パターンを抽出する必要がある．

(注5)：どのレジスタがゼロレジスタの機能を持つかはプロセッサ記述中に宣言されている




(a) 構文木　　(b) 命令パターン

図 6　ADD からの複数命令パターンの生成

```
behavior ADDIF (p, d, s0, s1) {
  if (p == 0) {
    GR[d] = GR[s0] + GR[s1];
  }
  else {
    if (PR[p]) {
      GR[d] = GR[s0] + GR[s1];
    }
  }
}
```

図 7　複数命令パターンを含む動作記述の例

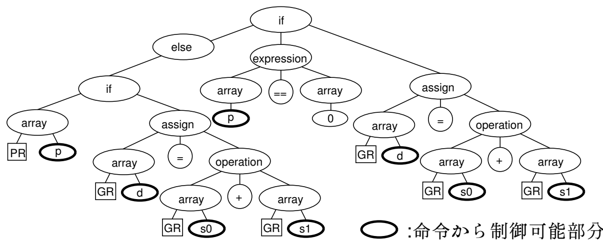
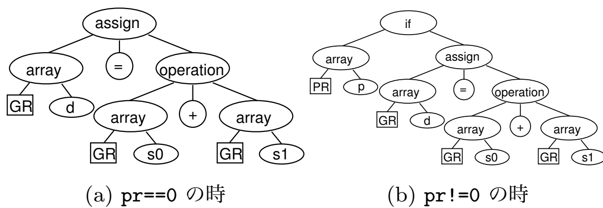



図 8　複数命令パターンを含む命令の構文木構造



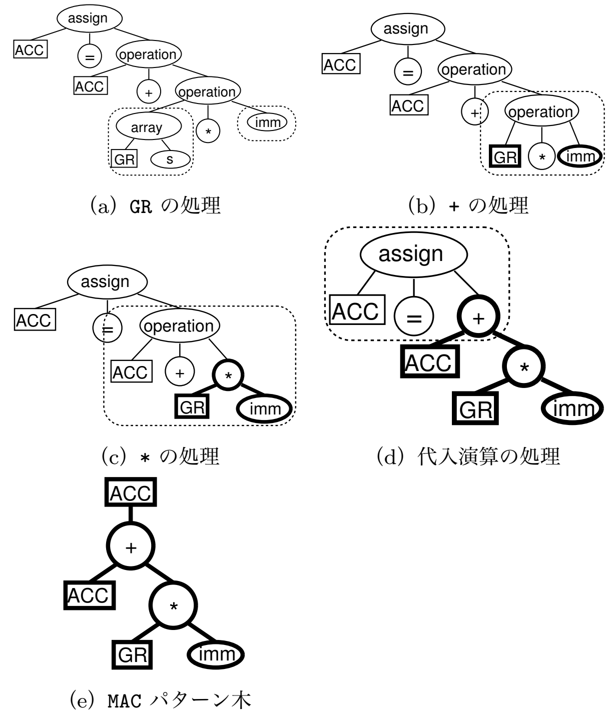

(a) pr==0 の時　　(b) pr!=0 の時

図 9　1 命令からの複数命令の生成

## 4.　命令パターン生成手法

### 4.1　命令パターン木

命令パターンの生成は，構文木を再帰的に走査し，ボトムアップに DAG を構築することにより行える．図 ?? に図 ??（b）の構文木から命令パターン（木）を抽出する例を示す．図 ??（a）の破線部分は GR[s] と imm の参照を行っているので，図 ??（b）の太線で示す GR と imm の 2 つのレジスタ参照節点に変換する．さらに図 ??（b）の破線部分は GR と imm の値の乗算を行っているので，図 ??（c）の太線部分のように GR と imm の子を持つ乗算演算の節点に変換する．同様に図 ??（c）の破線部分も図 ??（d）の太線部分のように変換する．図 ??（d）の破線部分は ACC に対する代入（書き込み）を行っている部分なので， ACC レジスタへの書き込み節点に変換し，最終的に図 ??（e）のような命令パターン木が得られる．




（a）GR の処理　　（b）+ の処理

（c）* の処理　　（d）代入演算の処理

（e）MAC パターン木

図 10　パターン木生成手法

### 4.2　プレディケート付命令パターン

図 ??（b）を例にプレディケート付命令パターン木生成法を示す．まず，then 節の処理を前項と同様の手法でパターン木に変換する．構文木中の if 文の then 節に代入文があり else 節がない構造からプレディケート付命令のパターンであることを認識し，生成されたパターン木に再帰的にプレディケートの付加を行えば図 ??（c）のようなプレディケート付命令パターン木の生成が行える．

### 4.3　命令パターン DAG

プロセッサの 1 命令が複数の文で表現される場合のパターン生成法を図 ?? の記述を例に示す．この記述は，図 ?? のように 2 つの代入文のリストからなる構文木に変換される．基本的には各代入文のパターン木を順に生成していくが，ハッシュ表を用いて同じ節点の参照を共有するようにする．この例では 1 つ目の代入に対して図 ??（a）のパターン木が生成され，2 つ目の代入に対しては図 ??（b）の太線のようなサブグラフが追加され，パターン DAG が得られる．

```
behavior JAL (imm) {
  LINK = PC;
  PC = PC + (imm << 2);
}
```

図 11　パターン DAG の動作記述の例

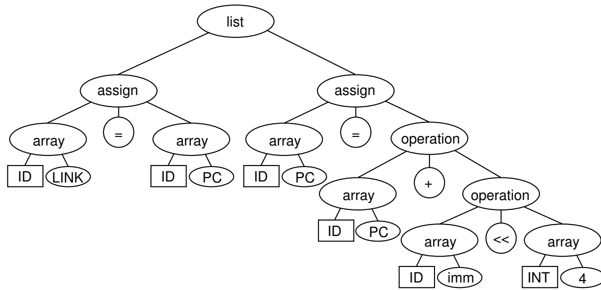
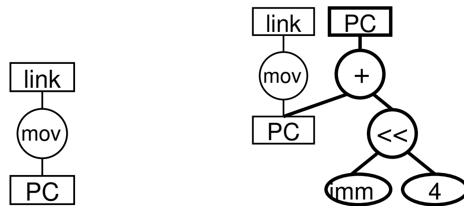



図 12　パターン DAG の構文木構造




(a) 1 つの動作のパターン木　(b) 2 つの動作の パターン DAG

図 13　パターン DAG の生成

```
behavior ADDI (d, s, imm) {
  GR[d] = GR[s] + imm;
}
```

図 14　動作記述例

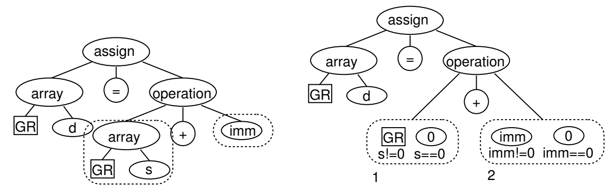
## 4.4　複数命令パターン

コンパイラに必要な複数の命令パターンの抽出は，

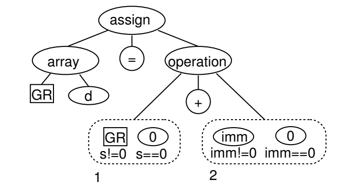
（1）　命令パターンを構築する過程で，制御可能部分に値を代入し場合分けを行う

（2）　重複するパターンや起こり得ないパターンを除去することにより行える．

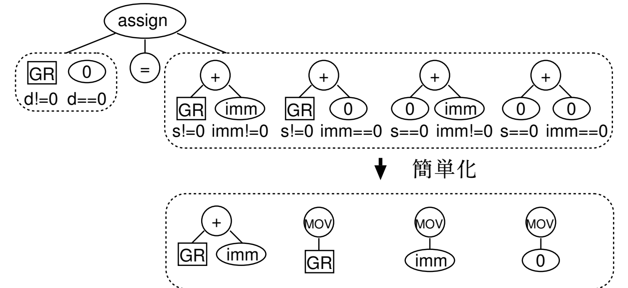
図 ?? の動作記述を例に考える．図 ?? (a) がその構文木構造である．ゼロレジスタが定義されている場合には，図 ?? (b) の右辺の破線部分 1 のように 制御可能部分 s を s!=0 と s==0 に場合分けすることにより，レジスタの値を参照する GR と定数 0 に場合分けすることができる．破線部分 2 も同様に imm!=0 と imm==0 に場合分けすることにより，即値の値を参照する imm と定数 0 に場合分けすることができる．次に (”+”) 節点への変換の際に，すべての組み合わせ ($2 \times 2 = 4$ 通り) のパターンを生成する．これら 4 つの加算をそれぞれ簡単化すると（図 ?? (c)），1 つの加算と 3 種類の転送命令が生成される．この時，書き込みレジスタ（代入文の左辺）も同様に場合分けが行えるので，図 ?? (c) のように d=!0 と d==0 の節点を生成する．左辺と右辺の全ての組み合わせから図 ?? (d) の 8 パターン（書き込み先がゼロレジスタの時は NOP に簡単化する）が生成される．重複するパターンを削除するとコンパイラに必要な 5 パターンの命令が生成できる．



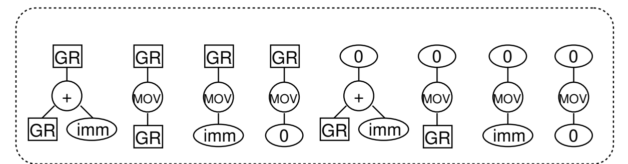

(a) ADDI の構文木

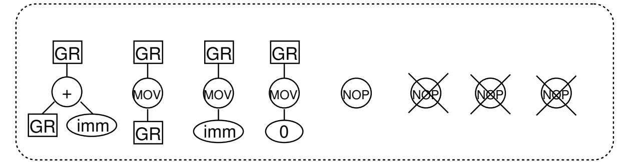



(b) GR と imm の場合分け




(c) 接点の組み合わせと簡単化

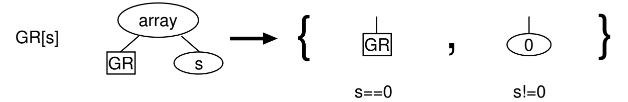


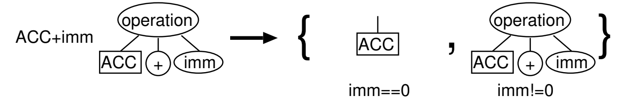

(d) 生成された命令パターン

図 15　複数命令パターンの生成手法

（1）　ゼロレジスタ




（2）　特定の演算




（3）　特定の演算

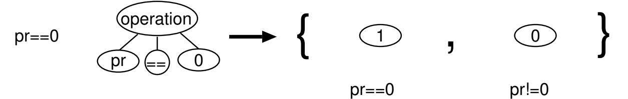



図 16　代入する値の選択

制御可能部分への代入による場合分けの例を図 ?? に示す．ゼロレジスタによる場合分け図 ?? (1) の他に即値などの特定の演算図 ?? (2) や特定の比較演算図 ?? (3) などが考えられる．

## 5. 実 験 結 果

以上の手法に基づく命令パターン抽出の処理系を実装し，実験を行った．処理系は Windows XP sp2 の Cygwin 上に Perl 5.8.5 で実装した．DLX のアーキテクチャ記述の 52 命令に対し，生成したテキスト形式の構文木を入力としてコンパイラに必要な全ての命令パターン DAG のテキスト形式を生成した．実行時間 0.012 秒（CPU Pentium 4 m 1.0 GHz，メモリ 768 MB）で表 **??** のような結果を得ることができた．1 命令の動作記述から値の代入により場合分けした節点の全ての組み合わせから 371 の命令パターンが生成できた．さらに，重複するパターンを削除した結果，最終的に 107 種類の命令パターンを生成することができた．

表 1 実 験 結 果

| サンプルプロセッサ | プロセッサの命令数 | 全命令パターン数 | 重複を削除した命令パターン数 |
|---|---|---|---|
| DLX | 52 | 371 | 107 |

## 6. む す び

本稿では，ASIP 設計システム ASIP Meister の各命令の C-like な動作記述からコンパイラのリターゲッティングに必要な命令パターンの生成を行う手法を提案した．さらに，プロセッサで実行可能な命令とコンパイラに必要なパターン命令の差を埋めるため，プロセッサ設計上の 1 命令からコンパイラに必要な複数の命令パターンを生成する手法を提案した．今後は, スケジューリング, バインディングなどのバックエンド部分の実装を行う予定である．課題として, コンディションコード命令や SIMD 命令への対応などが挙げられる．

## 謝 辞

本研究に際し, 御討論頂き, また, 種々の面々でお世話になりました大阪大学の武内良典先生をはじめ大阪大学今井研究室の関係諸氏, および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] K. Okuda, S. Kobayahi, Y. Takeuchi, and M. Imai: “A Simulator Generator Based on Configurable VLIW Model Considering Synthesizable HW Description and SW Tools Generation,” in *Proc.* Workshop on Synthesis and System Integration of Mixed information Technologies, pp. 152–159, Apr. 2003.

[2] 小林悠記, 小林真輔武, 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. 1311–1320, May 2004.

[3] R. Leupers and P. Marwedel: *Retargetable Compiler Technology for Embedded Systems*, Kluwer Academic, 2001.

[4] R. Leupers and P. Marwedel: “Instruction Selection for Embedded DSPs with Complex Instructions,” in *Proceedings x of the Conference on European Design Automation*, pp. 200–205, 1996.