コンパイラ生成のための ASIP 必要命令セット判定手法

由井 暁大　　劉 載勲　　武内 良典　　今井 正治

大阪大学 大学院情報科学研究科 情報システム工学専攻

概 要

組込みシステムの分野において利用される ASIP(Application Specific Instruction-Set Processor) は開発を行う際に設計・評価を繰り返す必要があるため，開発期間の短期化が課題とされている．その解決法の一つとして，ASIP の応用開発環境の一つであるコンパイラを ADL(Architecture Description Language) から自動生成することで開発期間の短期化を進めることが提案されている．本研究では，コンパイラ生成に必要な命令機能に対し命名を行い，プロセッサ命令とのマッチングを行うことでコンパイラ生成に使用する命令を選択し，コンパイラの生成を可能とした．

An Instruction-set Test Method for Compiler Generation of Application Specific Instruction-set Processors

Akihiro YOSHII，Jaehoon YU, Yoshinori TAKEUCHI, and Masaharu IMAI

Department of Information Systems Engineering
Graduate School of Information Science and Technology，Osaka University

Abstract

Application Specific Instruction-Set Processors (ASIPs) are often used in embedded system field. For the development of ASIPs, it is necessary to repeat design and validation ASIPs, and it is a time consuming process. In these days, in order to shorten development time, one of solutions is to generate ASIPs with compiler from a single architecture description language (ADL), but compiler generation has not still been an easy task. This paper proposes a method to name functions necessary for compiler generation of ASIPs. Proposed method tests whether compiler can be generated from declared instruction set in ADL by matching between defined functions and designed processor instructions.

1　はじめに

現代の社会において生活していく上で自動車や家電，携帯電話などの組込みシステムは欠かせないものとなっており，それらの組込みシステムは年々高性能化，複雑化している．組込みシステムの開発には，主に汎用プロセッサと ASIC(Application Specific Integrated Circuit) が用いられている．汎用プロセッサはプログラムを実行可能なため柔軟性は高いが，処理内容に対する消費電力が高く，性能が低い．また，ASIC は特定用途を想定した集積回路のため処理内容に対する消費電力が抑えられて性能も高いが，柔軟性が低い．そこで，これらの二つの利点をあわせ持つ，特定用途に特化したプロセッサである ASIP(Application Specific Instruction-Set Processor) が注目されている．ASIP は汎用プロセッサのようにプログラムを実行することが可能なため柔軟性が高く，専用演算器を搭載することで単位面積当たりの消費電力や性能を向上させるものである．しかし，ASIP の開発は設計・評価を繰り返すアーキテクチャ探索が必要なため，開発期間や開発コストが大きな問題となっている．

そこで近年，開発期間を短縮するために ADL を利用した ASIP 開発が提案されている．プロセッサの開発には，HDL 記述やアセンブラやリンカ，シ

ミュレータ，コンパイラ等の ASIP の応用利用のための開発環境が必要となるが，ADL を用いることでそれらを一つの記述より生成することを目指している．ASIP 開発向けの ADL には LISA [1]，nML [2]，ASIP Meister で利用されている ADL [3] 等がある．いずれもプロセッサ命令を記述する際に，命令の意味と動作を共に記述する必要があるため冗長であり，記述量が多くなる．また，命令の意味と動作の整合性がとれていない場合，生成された HDL と応用開発環境が正しく動作しないという問題がある．そこで新たに，プロセッサ命令を意味記述のみで定義することで冗長性を抑えた ADL が村田らにより提案されている [4]．

村田らによって提案された ADL では，ASIP 向け統合開発環境である ASIP Meister [3] を利用して ASIP の開発を行う．しかし，村田らの ADL はプロセッサ命令記述が ASIP Meister の HDL 生成とは異なるため，その ADL 向けのコンパイラの生成手法が提案されている [5]．村田らの ADL からコンパイラを自動で生成する際，ADL に記述された命令セットがコンパイラ生成の際の制約条件を満たさない場合があるため，その場合は複数の命令を組み合わせて利用することで制約条件を満たし，コンパイラの生成を行う．しかし，ADL に記述された命令セットがコンパイラ生成に必要な機能をすべて満たしているかを命令の意味記述から判断することは困難である．そのため，コンパイラ生成ができない場合があった．また，代替命令の記述方法が定義されていないという問題がある．

そこで，本研究では，プロセッサの命令がコンパイラ生成に必要とされるどの機能に相当するかを ADL に記述し，その記述からコンパイラ生成に必要な機能をすべて満たしていることを確認する手法を提案する．また，村田らの ADL に代替命令定義部を追加することで，代替命令を定義できるようにした．

以下，2 節では ASIP Meister [3] について説明し，3 節で ASIP Meister の入力である ADL からコンパイラを生成する手法および，提案手法について述べる．次の 4 節で評価実験について述べ，最後に 5 節でまとめを述べる．

2 ASIP 開発環境

ASIP の開発に利用される ADL には様々なものが提案されている [6]．以下に代表的な ADL を紹介する．

LISA [1] は抽象度の高い記述での ASIP 設計が可能な ADL である．そのため，ASIP の設計においてとても高い柔軟性を持ち，プロセッサの HDL 記述生成，アセンブラ，命令セットシミュレータ，リンカ，コンパイラの生成が可能である．しかし，プロセッサの命令を記述する際に命令の意味を記述するビヘイビア記述と，命令の動作を記述するマイクロ動作記述を記述しなければならない．このため，記述が冗長になる上に，二つの記述内で整合性がとれない可能性があり，工数が長くなりやすいという問題がある．また，それぞれの命令に対してパイプラインごとの動作を定義する必要があるため，パイプライン段数を変更すると命令記述をすべて書き換える必要がある．

nML [2] は，HDL 記述生成，コンパイラ，命令セットシミュレータ等の生成が可能な ADL である．しかし，nML のプロセッサ命令定義は抽象度が低く，パイプラインステージごとに使用する．演算器等を指定する必要があり，抽象度が低いため，仕様変更が難しくなっているという問題がある．

2.1 ASIP Meister

ASIP Meister [3] は ASIP 開発を行うための統合開発環境である．設計者は，ASIP Meister 独自の ADL を用いて設計したいプロセッサの記述を作成し，それを ASIP Meister に入力することで，論理合成可能なプロセッサ HDL 記述，およびサイクル精度のシミュレータ，アセンブラ，デバッガ，コンパイラなどの応用プログラム開発環境を生成することができる．ADL には，プロセッサのパイプライン段数，命令語長などのアーキテクチャ・パラメータ，使用する演算器などのリソース，レジスタやメモリなどのストレージ，インターフェースといった設計したいプロセッサの詳細な構造を記述できる．また，プロセッサの命令セットの定義には，命令タイプ，ビットフィールドの使い方，アドレッシングモードの定義等を行う．プロセッサの命令定義には，以下の 2 種類の記述を使用している．

ビヘイビア記述 ビヘイビア記述では命令が実行された際に，引数や引数に対して行う操作といった命令の意味的な動作を定義する．ビヘイビア記述は，シミュレータやコンパイラなど，応用プログラム開発環境の生成に用いられる．

マイクロ動作記述 マイクロ動作記述では，命令が実行されたときにプロセッサがパイプライン・

ステージごとにどのような演算器を使用して，どのような動作を行うかを定義する．マイクロ動作記述は，プロセッサの HDL 記述の生成に用いられる．

上で述べた通り，ASIP Meister では命令の定義に，ビヘイビアとマイクロ動作の 2 種類の記述を採用している．しかし，これには以下の 2 つの問題がある．同じ命令の動作を表すのに，2 つの記述を用意するのは冗長である．マイクロ動作記述は ADL の命令記述において 50%以上の割合を占めており [7]，記述量を増加させている．また，2 つの記述の間に矛盾がある場合，設計したプロセッサ上で応用プログラム開発環境が動作しない．そのため，設計者が常に整合性を意識しなければならず，設計時間の増加につながる．これらの問題を解決するために，命令の定義を変更した新たな ADL が村田らによって提案されている．

2.2 村田らの ADL による ASIP 開発

村田らの ADL による ASIP 開発フローを図 1 に示す．村田らの ADL は，ASIP Meister を基として設計されており，ADL 記述を ASIP Meister の入力形式である PDB 形式に変換し，ASIP Meister に入力して HDL 記述の生成を行う．しかし，プロセッサ命令の記述方法が ASIP Meister の HDL 生成部と異なるため，同じ記述を用いて ASIP Meister からコンパイラを生成する場合は制約が大きい．そのためコンパイラを生成する際には，CoSy コンパイラ開発システム [8] を利用する．ADL を CoSy の入力形式である CGD に変換して入力することでコンパイラの生成を行う．これらにより，ASIP の設計及び ASIP の応用利用のための開発環境を生成するものである．

3 ADL からのコンパイラ生成と命令選択

3.1 CoSy コンパイラ開発システム

CoSy コンパイラ開発システム [8] は，プロセッサの仕様記述を基にターゲット・プロセッサに最適化されたコンパイラを生成できるシステムである．CoSy によるコンパイラ生成に必要な情報は 3 つある．

アーキテクチャ情報 命令ビット幅，C データ型のサイズや使用可能なメモリタイプなどのプロセッサの基本的な情報を記述する．

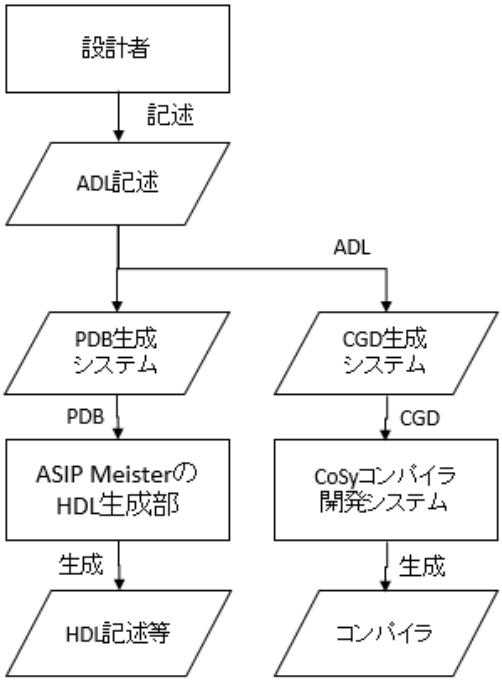

図 1: ADL からの ASIP 開発フロー

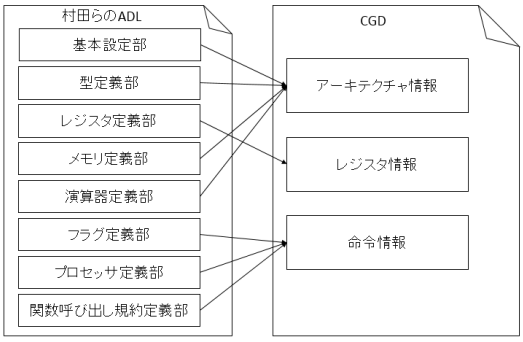

図 2: 村田らの ADL と CGD の情報の構成

レジスタ情報 レジスタファイル数や使用可能なレジスタの属性などの情報を記述する．

命令情報 コンパイラ生成の際に使用する CoSy 独自の中間表現へマッチングするためのプロセッサ命令の情報を記述する．

村田らの ADL から CGD を生成する際の情報の構成を図 2 に示す．CGD のアーキテクチャ情報は，村田らの ADL の基本設定部，型定義部，メモリ定義部，演算器定義部に記述されており，そこから必要な情報を抽出し，CGD に変換して生成を行う．また，レジスタ情報は，村田らの ADL のレジスタ定義部に記述されているため，同様に CGD への変換が可能となる．命令情報は，プロセッサ命令定義部，フラグ定義部，関数呼び出し規約定義部で定義されており，CGD への変換を行う．

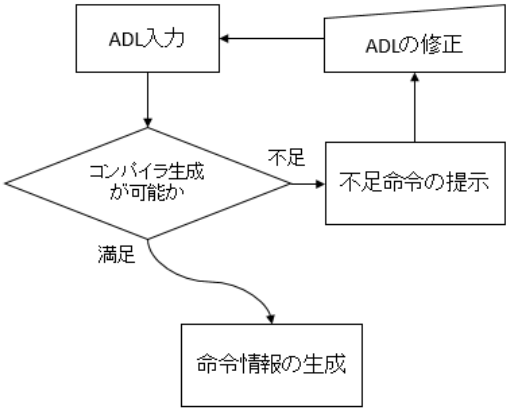

図 3: 命令情報の生成

アーキテクチャ情報とレジスタ情報は，ADL の持つ情報に対して一意に決まる．しかし，命令情報については，ADL に記述された命令セットがコンパイラを生成するのに必要な命令機能をすべて満たすかどうかの判断がつかないこともあり，生成は容易ではない．そこで，ADL からのコンパイラ生成において，次節で述べるように CoSy によるコンパイラ生成に必要な命令機能が決定される．

3.2 コンパイラ生成に必要な命令機能

CoSy によるコンパイラ生成に必要な命令機能は，算術論理演算機能，比較機能，論理機能，シフト演算機能の 4 つに分類される．算術論理演算機能は，算術演算，論理演算，シフト演算等の数値計算に関する命令機能である．比較機能は，条件分岐等に利用する，二値の比較を行う命令機能である．分岐機能は，無条件分岐，条件分岐に関する命令機能である．ロード・ストア機能は，レジスタ・メモリ間での値の移動を行う命令機能である．32bit RISC プロセッサの場合では，計 45 個の命令機能が必要とされる．

ADL からの命令生成フローを図 3 に示す．設計者が用意した ADL より，命令セットがコンパイラ生成に必要な機能をすべて満たすかの判定を行う．命令セットがコンパイラ生成に必要な命令を全て満たしていない場合，不足している命令を設計者に提示し，設計者が ADL の修正を行う．コンパイラ生成に必要な命令を全て満たしていた場合，ADL の情報を基に，命令情報の生成を行う．しかし，命令のビヘイビア記述を読み取り，コンパイラ生成に必要な機能が全て含まれているかを確認することは難しい．また，ADL を記述する設計者はプロセッサ命令についてよく理解していることが想定されるため，プロセッサに用意されている命令がコンパイラ生成に必要とされるどの機能に相当するかを判断することは容易と考えられる．

そこで本研究では，村田らの ADL のプロセッサ命令の記述を変更し，プロセッサに用意されている命令とコンパイラ生成に必要な機能の対応を記述することで，コンパイラ生成に必要な機能を全て満たしていることの確認を行う手法を提案する．

3.3 提案手法

先行研究 [5] におけるプロセッサ命令の記述に必要な情報は 3 つである．命令がどのような意味を持つのかを定義するビヘイビア記述，命令によって操作されるフラグとその条件の定義を行うフラグ操作記述，アセンブラコードを出力する際の命令のフォーマットの定義を行う命令フォーマット記述である．以上のプロセッサ命令記述に，コンパイラ生成に必要な命令機能の定義において決定した名前を記述する命令機能対応情報を追加する．

また，命令が不足していた場合の代替命令の記述方法が定義されていなかったため，新たに図 4 のような代替命令の記述を定義した．代替命令とは，コンパイラ生成に必要な機能が不足している場合に，プロセッサ命令の組み合わせで不足している機能を補完する命令である．代替命令記述部では，プロセッサ命令の組み合わせと，コンパイラ生成に必要な命令機能の定義において決定した名前を記述する．図 4 の例はレジスタの値を他のレジスタに移動する機能”MOV”がコンパイラ生成に必要な際の例である．”substitute”記述の後の”ADDI(rd, rs, 0)”はプロセッサ命令だけでは不足している機能を代替する命令コードである．この場合は，レジスタ間の値の移動を，即値加算命令で代替するものである．”equivalent”記述の後の”MOV”は，コンパイラ生成に必要な命令機能の定義において決定した名前を記述するものである．この場合は，レジスタ間移動の機能を補完していることを示している．

以上の 2 箇所の修正により，プロセッサの命令とコンパイラ生成に必要な命令機能との対応関係を得ることができる．また，ADL に記述された命令セットがコンパイラ生成に必要な機能を満たすかの判定を以下をに示す．

$$L = C \cap \bar{F} \tag{1}$$

ここで，C はコンパイラ生成に必要な機能の集合，

```
MOV{
  substitute
    ADDI(rd, rs, 0);
  equivalent
    "MOV"
}
```

図 4: 代替命令記述の例

F は ADL に記述した命令の集合，L は ADL に不足している機能の集合とする．式 (1) は，コンパイラ生成に必要な機能の集合 C から，ADL に記述した命令の集合 F の補集合との積をとることで，ADL に記述された命令セットに不足している機能の集合 L を求められることを表している．L が空集合であれば，命令は不足していないことを示している．ここで求められた，ADL に記述された命令セットに不足している機能の集合 L を設計者に提示することで，設計者は作成するべき代替命令を知ることができる．

4 評価実験

4.1 実験

コンパイラ生成に必要な機能の名前を ADL に記述することで，命令セットがコンパイラ生成に必要な機能をすべて満たすかの判定が可能であることを確認する．本実験では，32 ビット RISC プロセッサである Brownie STD 32 [9], DLX [10] および MIPS R2000 [11] と，独自命令セットを持つプロセッサを対象命令セットとし，コンパイラ生成に不足している機能を提示されているかの確認を行った．表 1 にそれぞれの命令セットに対して不足している機能を記載する．

Brownie STD 32 プロセッサの命令セットについて，システムを利用して判定を行ったときの判定結果を図 5 に示す．それぞれの命令セットに対して，追加しなければならない代替命令を過不足なく提示できていることが確認できた．

判定結果を基に，各命令セットに対して不足している機能の代替命令を記述を追加した．このとき，既存プロセッサの命令セットについてはすべての不足している機能の代替命令を作成できたが，独自の命令セットでは，ADDI, SUBI, ANDI, LTU, ELTU, GTU, EGTU, LHI, LWI 機能の計 9 機能についての代替命令が作成できなかった．代替命令を追加した後に，再び判定を行った．判定結果をまとめたものを表 2 に示す．

表 1: それぞれの命令セットに対して不足している機能

プロセッサ名	不足している機能
Brownie STD 32	NOT，MOV，NEG，ABS，LHI，LWI，ELT，ELTU，GT, GTU, EGT，EGTU
DLX	NOT, MOV, NEG, DIV, DIVU, MOD, MODU, EXBW, EXHW, ABS, LHI, LWI, LTU, ELTU, GTU, EGTU
MIPS R2000	DIV, DIVU, MOD, MODU, XOR, NOT, NEG, ABS, SUBI, ELT, ELTU, GT, GTU, EGT, EGTU, NEQ, EXHW, EXBW, LHI, LWI, MOV, JPRL
独自の命令セット	NOT, XOR, NEG, ABS, ADDI, SUBI, ANDI, LTU, ELT, ELTU, GT, GTU, EGT, EGTU, NEQ, LH, LW, SH, SW, LHI, LWI, MOV

```
lack of required instructions
lack: EGT instruction
lack: EGTU instruction
lack: ELT instruction
lack: NOT instruction
lack: ELTU instruction
lack: LWI instruction
lack: MOV instruction
lack: ABS instruction
lack: GTU instruction
lack: NEG instruction
lack: GT instruction
lack: LHI instruction
```

図 5: Brownie STD 32 の標準の命令セットでの不足している機能のメッセージ

判定結果より，既存プロセッサの命令セットについては不足している機能をすべて定義することができた．また，独自の命令セットについては上記の 9 つの機能についてのみ不足しているということが確認できた．

4.2 考察

以上の実験結果より，提案手法を用いることで，コンパイラ生成に必要な機能に対応した命令の情報を得られることが可能となった．これにより，コンパイラ生成のための CGD ファイルに記述する情報を取得できたこととなり，コンパイラの生成が可能となる．また，命令セットによっては，コンパイラ生成に必要な機能をすべて満たすことができない場合があることが分かった．コンパイラ生成に必要な機能をすべて満たすことができない場合，コンパイラの生成ができないため，プロセッサの設計者はプ

表 2: 各命令セットに対して不足していると提示された機能

命令セット	不足している機能
Brownie STD 32	なし
DLX	なし
MIPS R2000	なし
独自の命令セット	ADDI, SUBI, ANDI, LTU, ELTU, GTU, EGTU, LHI, LWI

ロセッサの設計を修正してコンパイラ生成に必要な機能を満足できる命令セットに設計する必要がある．

5 まとめ

本研究では，ADL からの ASIP 開発環境の生成における，プロセッサの命令セットがコンパイラ生成に必要な機能をすべて満たすことの判定が難しいという問題に着目した．その問題に対して，プロセッサ設計者にとってコンパイラ生成に必要な機能とプロセッサ命令の対応関係は容易に判断できるという点に注目し，ADL の記述を追加することで命令の対応を定義し，定義を基にプロセッサの命令セットがコンパイラ生成に必要な機能をすべて満たすことの判定を行う手法を提案した．評価では，Brownie STD 32, DLX および MIPS R2000 の 3 つのプロセッサの命令セットと独自命令セットを持つプロセッサから，コンパイラ生成に必要な機能の判定を行った．その結果，命令セットがコンパイラ生成に必要な機能をすべて満たすことができる場合とそうでない場合があることが確認できた．

今後の課題として，現在は 32 ビット RISC プロセッサのコンパイラ生成に必要な機能しか定義できていないため，他のアーキテクチャ向けのコンパイラ生成に必要な機能の定義を行う必要がある．

参考文献

[1] Schliebusch, O., Chattopadhyay, A., Leupers, R., Ascheid, G., Meyr, H., Steinert, M., Braun, G. and Nohl, A.: RTL processor synthesis for architecture exploration and implementation, *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, Vol. 3, IEEE, pp. 156–160 (2004).

[2] Johan, Van, P., Dirk, L., Werner, G. and Gert, G.: *Processor description languages*, chapter 4, Morgan Kaufmann (2011).

[3] 今井正治，武内良典:コンフィギュラブル・プロセッサ生成技術の現状と今後の展望，第 17 回回路とシステム 軽井沢ワークショップ 論文集，pp. 541–548 (2004).

[4] 村田謙介，稗田拓路，坂主圭史，武内良典，今井正治:HDL 記述と応用プログラム開発環境の生成が可能なプロセッサのためのアーキテクチャ記述言語の提案，DA シンポジウム 2011 論文集，pp. 171–176 (2011).

[5] 兵藤佑亮，村田謙介，稗田拓路，坂主圭史，武内良典，今井正治:ASIP 統合開発環境のための ADL からのコンパイラ生成手法，DA シンポジウム 2011 論文集，pp. 177–182 (2011).

[6] Johan, Van, P., Dirk, L., Werner, G. and Gert, G.: *Processor description languages*, Morgan Kaufmann (2011).

[7] Shiro, T., Abe, M., Sakanushi, K., Takeuchi, Y. and Imai, M.: A processor generation method from instruction behavior description based on specification of pipeline stages and functional units, *2007. ASP-DAC'07. Asia and South Pacific Design Automation Conference*, IEEE, pp. 286–291 (2007).

[8] ACE Associated Compiler Experts: CoSy Compiler Development System. `http://www.ace.nl/compiler/cosy.html`.

[9] 岩戸宏文，稗田拓路，田中浩明， 佐藤淳，坂主圭史，武内良典，今井正治:ASIP 短期開発のための高い拡張性を有するベースプロセッサの提案，情報処理学会研究報告. SLDM， Vol. 2007, No. 114, pp. 133–138 (2007).

[10] Hennessy, J. L. and Patterson, D. A.: *Computer architecture: a quantitative approach*, Elsevier (2012).

[11] Patterson, D. A. and Hennessy, J. L.: *Computer organization and design: the hardware/software interface*, Newnes (2013).