システムソフトウェアと 89－1
オペレーティング・システム
シ ス テ ム 評 価 2－1
（２００２．２．１４）

目的コードサイズ縮小のための短形式命令サブセットの最適構成

平尾智也 [1)]　　中西恒夫 [1)]　　福田晃 [2)]

[1)] 奈良先端科学技術大学院大学 情報科学研究科
[2)] 九州大学大学院システム情報科学研究院 情報工学部門

概要

アセンブリコード内に頻繁に現れる命令のいくつかをビット長の短い命令サブセットとして定義することにより，オブジェクトコードのサイズを縮小する．しかし，どの命令が頻出するかはアプリケーションによって異なるため，コードサイズを最小化する短形式命令サブセットの構成はアプリケーション毎に異なる．本稿では対象となるアプリケーションのオブジェクトコードを最小化する命令語ビット長の決定と命令サブセットの構成方法について述べる．
予備評価によれば提案手法によりオブジェクトコードを圧縮した結果，コードサイズは非圧縮時のサイズの 40%以下に縮小された．

Constructing a Short Format Instruction Subset for Code Size Reduction

Tomoya Hirao[1)]　　Tsuneo Nakanishi[1)]　　Akira Fukuda[2)]

[1)] Graduate School of Information Science
Nara Institute of Science and Technology
[2)] Graduate School of Information Science and Electrical Engineering
Kyushu University

Abstract

The object code size can be reduced by defining frequently appeared instructions as a narrower bit width instruction, or a short format instruction subset. However, each application has a different content of the short format instruction subset which minimize the object code size. This paper describes how to decide the bit width of instructions and construct a short format instruction subset to reduce the object code size. In our preliminary evaluation of the presented method the object code size is reduced to less than 40% of its original code size.

1　背景

オフチップメモリは容量や価格の面での制約は緩やかであるが，外部バスの駆動を必要とするため消費電力の面からは不利である．さらに部品点数の増加による信頼性の低下も無視できない．これらの理由からソフトウェアをオンチップメモリに搭載することが望まれているが，オンチップメモリはチップ上に占める面積が大きく，大量に確保することは困難である．一方で組込みシステムの複雑化の一途をたどっており，ソフトウェア規模は大きくなる一方である．

ソフトウェアをオンチップメモリに収めることによりワンチップ化，省電力化を図ることができる．そのためにオブジェクトコードのサイズを実行可能な形式で縮小することは重要である．

コードサイズを縮小するために，CISC プロセッサでは多くのアプリケーションで多用される命令にビッ

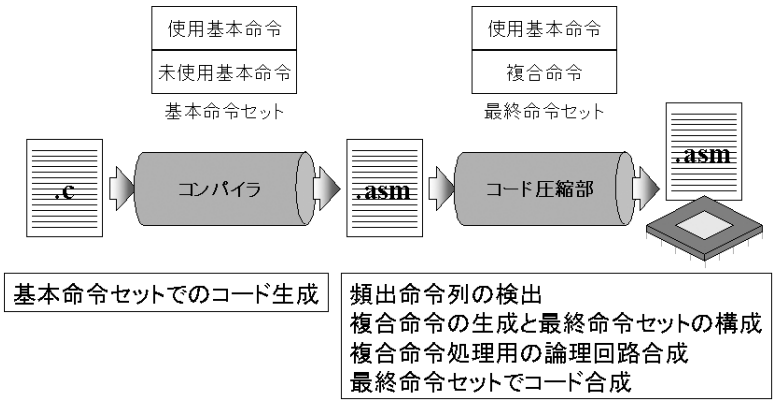

図 1: プロセッサおよび目的コードの生成

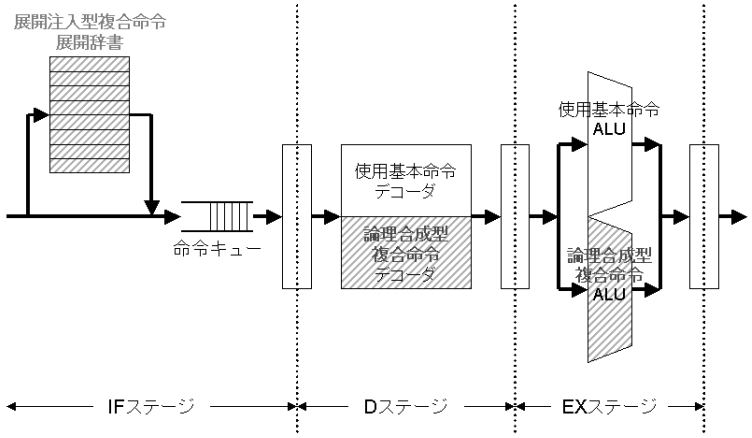

図 2: 生成されるプロセッサの構成

ト長の短い命令語を割り当てている．また，RISC プロセッサは一般に命令のビット長が一定であるが，ARM では多くのアプリケーションで多用される命令に対し短い命令語を割り当てた Thumb 命令セットを利用することによりコードサイズの削減している．

しかし，アセンブリコード内での命令の出現頻度はアプリケーションによって異なる．単一のアプリケーションのみが動作する組込み用プロセッサにおいては，対象となるアプリケーションのアセンブリコードを詳細に解析し，どの命令に対して短い命令語を割り当てるか，すなわちどの命令を短形式命令サブセットに組込むかを決定することにより，さらにオブジェクトコードサイズを縮小することが可能である．

我々はこれまでにコードサイズの縮小を目的としたプロセッサ／オブジェクトコードコジェネレータを実装している [4]．図 1 はこのコジェネレータを用いたプロセッサおよびその上で動作するオブジェクトコード生成のフローである．コジェネレータは，入力されたプログラムのオブジェクトコードを最小化する命令セットをもつプロセッサの VHDL ソースと，そのプロセッサ上で動作するオブジェクトコードを生成する．具体的には，入力されたプログラム内に頻繁に出現する命令列をより短い複合命令とし

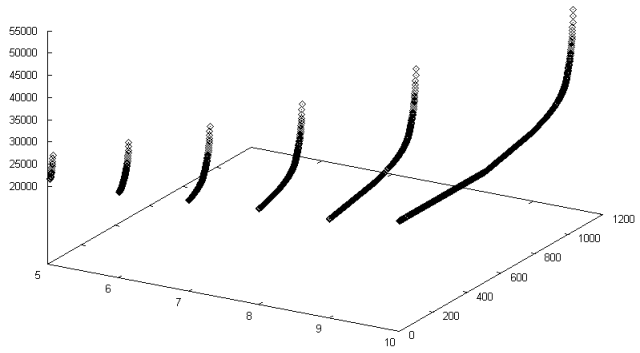

図 3: コードサイズと各パラメータとの関係

て定義することでコードサイズを縮小する．複合命令は通常のプロセッサでは処理できないため，生成されるプロセッサには複合命令処理用の専用回路が付加される．コジェネレータで生成されるプロセッサの構成を図 2 に示す．図中，灰色の部分がコジェネレータによりアプリケーション毎に構成が変更される部分である．複合命令の展開辞書，複合命令のデコーダ，および ALU がアプリケーション毎に構成が変更される．

上述のコジェネレータは，オブジェクトコードを垂直方向（流れ方向）に圧縮しているが，水平方向（命令後のビット列）には依然として冗長性が残っている．水平方向の冗長性排除のためには頻出命令に対して短い命令語の割り当てをおこなう．本稿では，オブジェクトコードのサイズを最小化するために，命令語のビット長と短形式命令サブセットを最適に構成する方法について述べる．

本稿第 2 節では実行可能形式へのコードサイズ縮小に関する関連研究を紹介する．第 3 節で提案手法の詳細を述べる．第 4 節では圧縮の対象となるアプリケーションのアセンブリコードからオブジェクトコードならびにプロセッサ生成までの流れを示す．第 5 節で提案手法の有効性を評価し，最後の第 6 節でまとめを述べる．

2 関連研究

オブジェクトコードのサイズを実行可能な形式で縮小する方法は大きく三つに分類される．

ひとつ目はアセンブリコード内で頻出する命令列をより短い命令列に置き換える方法である．頻出命令列を関数化する方法や，新たな命令として定義する方法がある [1]．頻出命令列を関数化すると関数呼び出しのオーバヘッドがかかる．新たな命令として定義する場合は命令を解釈するデコーダを変更する必要がある．

ふたつ目の方法はオブジェクトコードを可変長符号を用いて圧縮する方法である．この場合，プロセッサによるフェッチに先立って符号語を展開し実行する．Wolfe と Chanin[2] はハフマン符号，Lekatsas[3] らは算術符号によりコード圧縮を行う手法を提案している．いずれの手法においても主記憶とキャッシュの間，もしくはキャッシュとプロセッサの間にデコードエンジンを搭載しており，プロセッサの構造自体は変更していない．

三つ目の方法は，頻繁に用いる命令に対して短いビット長の命令語を割り当てる方法である．ARM の Thumb 命令セットがこれにあたる．本稿でもこのアイデアに基づく手法を述べる．

3　提案手法

提案手法は，アセンブリ命令への命令語の割り当てをすべて変更する．したがって，同じアセンブリ言語の同じ命令であっても，アプリケーション毎に別々の命令語が割り当てられることになる．このことはアプリケーション毎に専用設計のプロセッサが必要であることを意味している．

しかし，本研究ではプロセッサの VHDL ソースを縮小オブジェクトコードと同時に機械的に生成するため，プロセッサを専用設計することは大きな問題ではない．

3.1　命令語

提案する手法では命令セットは単位長命令と，その半分のビット幅の短形式命令の 2 種類からなる．単位長命令，短形式命令の命令語の構造を図 4 に示す．短形式命令のビット長は，単位長命令のビット長の半分とする．高い圧縮率を得るためにはハフマン符号や算術符号を用いたほうが有効である．しかしこれらの符号化方式は符号長が 1 ビット単位で可変であるため，そのまま命令語として用いるとハードウェアの複雑化，大規模化を招そのため提案手法では 2 種類の固定長の命令語を用いる．

単位長命令の前半部は単位長命令であることを示すプレフィックスであり，後半部のビットパターンにより命令を識別する．一方，短形式命令は 1 コードワードからなり，そのビットパターンにより命令を識別する．

一般的にオブジェクトコードのサイズ S は次の式で表される．ただし，ターゲットとなるアセンブリコード内に出現する命令が m 種類であり，頻繁に出現する命令から順に $1, 2, \cdots, m$ と番号付けされているものとする．

$$S = \sum_{i=1}^{m} s_i f_i \tag{1}$$

ここで s_i は命令 i のビット数，f_i は命令 i の出現回数である．

短形式命令のビット幅を w，識別子の種類を d 種類とすると，次の式で表される（図 5）．

$$s_i = \begin{cases} w & : \quad 1 \leq i \leq 2^w - d \\ 2w & : \quad 2^w - d + 1 \leq i \leq m \end{cases} \tag{2}$$

よって，式 (1) は次のように書き換えることができる．

$$S = w \sum_{i=1}^{2^w - d} f_i + 2w \sum_{i=2^w - d + 1}^{m} f_i \tag{3}$$

m と f_i はアセンブリコードの走査により得ることができる既知数である．この手法によりコードサイズを最小化することは，式 (3) において最小の S を与える w, d を求める問題に帰着する．

この問題を解くため，パラメータである w, d の性質について詳しく考察する．

3.2　コードサイズと w, d の関係

式 (3) は d について単調増加である．そのため最小の S を与える d は，w の値に応じて一意に決定される．

図 3 はオブジェクトコードサイズ S と短形式命令のビット幅 w，単位長命令であることを示す識別子の数 d の関係である．図中，X 軸に w を，Y 軸に d を，Z 軸にオブジェクトコードのサイズをとっている．

w, d, m の間には次の関係がある．

$$2^w - d + d \times 2^w \geq m \tag{4}$$

式 (4) の左辺は w, d によって定まる識別可能な命令の数を意味する．この数がアプリケーション内に出現する命令の種類 m を上回る必要がある．

この関係より，

$$d \geq \frac{m - 2^w}{2^w - 1} \tag{5}$$

を得る．ここで式 (3) は d について単調増加であり，d は可能な限り小さい値であることが望ましいので，

d は w の関数

$$d = \left\lceil \frac{m - 2^w}{2^w - 1} \right\rceil \tag{6}$$

である．

結果，コードサイズ S は w の関数であり，最小の S を与える w は数学的に容易に求めることができる．

さらに，w がとりうる値は式 (7) で表される範囲にとどまる．

$$\frac{\log_2 m}{2} \geq w \geq \log_2 m \tag{7}$$

アセンブリコード内に出現する命令をすべて短形式命令で実現した場合が w の下界であり，すべてを単位長命令で実現した場合が w の上界である．

式 (3) は非線形計画問題であり線形計画法では解くことができないが，式 (7) で与えられるすべての w に対して S を求め最小値を与える w を採用することによりオブジェクトコードを最小化する命令セットを構築することができる．

この方法で解を求める際の計算コストはたかだか $O(m \log_2 m)$ である．

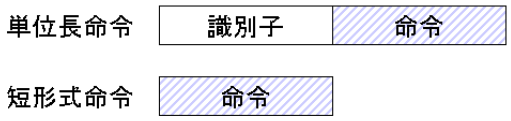

図 4: 命令語の構造

4 コードおよびプロセッサ生成までの流れ

提案手法を用いてオブジェクトコードとプロセッサを生成するまでのフローは次のとおりである．

1. アセンブリコードを解析し，何種類の命令が使われたか，各命令が何回使われたかを計測する．

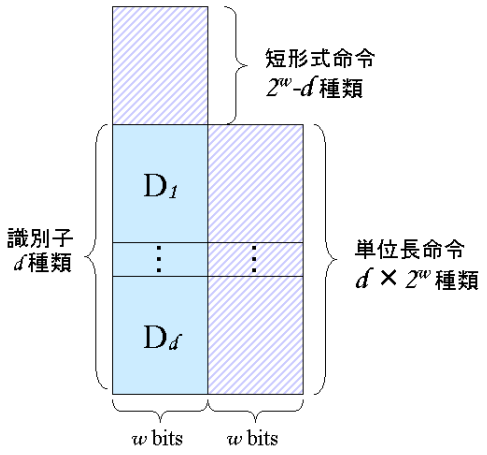

図 5: 命令語のビット幅，識別子の数と識別可能な命令の種類との関係

2. 計測結果に基づき，第 3 節の手法で命令語のビット長と識別子の数を決定する．

3. アセンブリコードに出現するの命令のうち，出現頻度の高い命令から順に $2^w - d$ 種類を短形式命令，残りを単位長命令とする命令セットを構築する．

4. 構築した命令セットに従いプロセッサのデコードステージに変更を加える．もしくは命令語を通常の命令のビットパターンに展開する辞書をプロセッサに搭載する．同時に，オブジェクトコードの生成を行う．

提案手法ではアプリケーションによって命令語のビット長が変化するので，プロセッサのアーキテクチャによってはデコードステージ以外に命令を転送するバスの幅の変更が必要になる場合がある．

5 予備評価

第 3 節で提案した手法に基づき ADPCM, FFT および quick sort の 3 種類のプログラムのオブジェクトコードサイズの縮小を行った．これら 3 種類のプ

表 1: 各アプリケーションにおける出現命令数と圧縮率

	出現命令数 m	短形式命令のビット長 w	単位長命令プレフィックスの数 d	圧縮率
ADPCM	518	6	8	34.2%
FFT	192	4	16	37.9%
quick sort	93	7	0	38.4%

ログラムに出現する命令の種類 m と，提案手法により決定された短形式命令のビット長 w，単位長命令のプレフィックスの数 d，および達成された圧縮率は表 1 のとおりであった．さらに，コードサイズ縮小の結果を図 6 に示す．縦軸はオブジェクトコードのサイズである．ただし外部参照のライブラリやデータセグメントはコードサイズに含まない．また，同じ種類の命令であっても，定数の指定などオペランド値が異なるものは別の命令としてカウントした．

評価の結果，圧縮率が最も低かった quick sort で非圧縮時のサイズの 38.4%，最も圧縮された ADPCM では 34.2%であった．命令語長 w にはアセンブリコード内の命令を識別するのに必要なビット数しか必要ないことと，オペランド値が異なるものは別命令とみなしたため，オペランド値を保持する必要がなくなったことがコードサイズ削減の大きな要因である．また，ADPCM と FFT では，すべての命令の命令語長を一律 $\log_2 m$ ビットとし，短形式命令を用いない場合と比較すると，短形式を用いた場合のほうがより高い圧縮率を示した（図 6）．

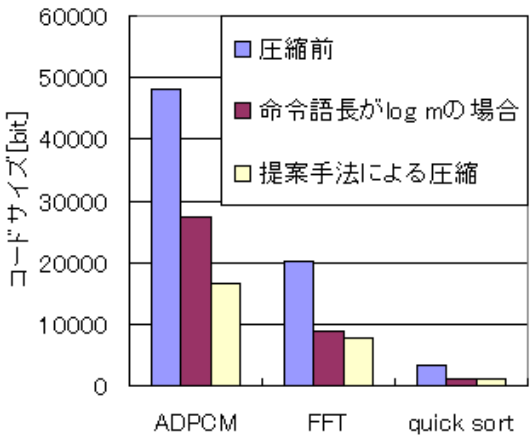

図 6: コードサイズ縮小効果

6 まとめ

本稿では命令語のビット長と短形式命令サブセットの最適構成によりオブジェクトコードのサイズを縮小する手法を提案した．予備評価によるとこの手法を用いることでオブジェクトコードのサイズを 40%程度にまで圧縮できることがわかった．

今回の評価では，単純に同じ命令であってもオペランド値が異なるものは別の命令語を割り当てるものとしたが，オペランド値の取り扱いについては今後さらに検討する必要がある．また，提案手法により構成した命令セットをデコーダで直接解釈するのか辞書を用いて通常の命令語に展開するのかといったプロセッサの実装についても検討する必要がある．

今後は，本稿で示した短形式命令サブセット構成手法を第 1 節で述べた開発中のコード／プロセッサコジェネレータに組み込み，垂直方向圧縮，水平方向圧縮の併用について検討する．さらに，現在，本研究と並行して CMOS 回路の消費電力を削減することを目的とした省電力指向符号化アルゴリズムの研究 [5] を行っており，今後この研究と連携し，省スペース・省電力な組込みシステムの開発手法について研究を進める予定である．

謝辞

本研究はマツダ財団・第 16 回研究助成の助成（課題名:組込み機器向け省資源プロセッサ／目的コードコジェネレータの実装）を受けています.

参考文献

[1] Charles Lefurgy, Peter Bird, I-Cheng Chen, and Trevor Mudge, “Improving Code Density Using Compression Techniques,” *Proc. of the 30th Annual Int. Symp. on Microarchitecture*, pp. 194–203, December 1997.

[2] Andrew Wolfe and Alex Channin, “Executing Compressed Programs on an Embedded RISC Architecture,” *Proc. of 25th Annual Int. Symp. on Microarchitecture*, pp. 81–91, December 1992.

[3] Haris Lekatsas, Jörg Henkel, and Wayne Wolf, “Code Compression for Low Power Embedded System Design,” *Proc. of ACM/IEEE Design Automation Conf.* , pp. 294–299, June 2000.

[4] 平尾 智也, 中野 猛, 中西 恒夫, 福田 晃, 「コードサイズ縮小を目的としたプロセッサ／目的コードコジェネレータの実装」, 第 3 回組込みシステム技術に関するサマーワークショップ (SWEST3) 予稿集, pp.140-145, 2001 年 7 月.

[5] 中西 恒夫, 福田 晃, 「省電力指向符号化アルゴリズムとその予備評価」, 情処研報, 発表予定