ソフトウェア工学　28−2
（１９８３．２．９）

マルチ・ターゲット・Ｃコンパイラの作成

引地　信之　　今井　正治
（豊橋技術科学大学）

1．はじめに

ミニ／マイクロコンピュータのソフトウェア開発の手法は，クロスコンパイラを用いるクロス処理，及びレジデントコンパイラを用いるセルフ処理の二つに分類できる。ソフトウェアの蓄積及び一元的管理という観点から，ホストコンピュータ上で開発を行うクロス処理の方が適している。

クロスコンパイラを作成する場合，二つの方法が考えられる。一つは，移植性の良いコンパイラを利用する方法であり，もう一つは，マルチターゲットコンパイラを使用する方法である。移植性の良いコンパイラの例として，ＰＣＣ[1]がある。これは，目的計算機に固有な部分を表にまとめて，移植性を良くしているが一方，この表は，コンパイラ内部に依存しているので変更が容易でないという欠点がある。後者の例としてＰＱＣＣ[2]があり，コード生成部生成系（ＣＧＧ：Code generator generator）を使用して，サポート言語や目的計算機の差異を吸収しようとするものである。その特徴を次に示す。①目的計算機の形式的仕様を非手続き的な言語で与え，出力として，コード生成部で使用するテンプレートを生成すること，②発見的手法を用い，問題向きの簡単な定理証明システムに似たアルゴリズムを使用していること，③定理，実現データベースを使用して，効率化をはかっていること，④ＣＧＧ自体は，サポート言語に依存しないこと，等があげられる。しかし，ＣＧＧが複雑で規模が大きくなるという欠点がある。

本報告では，マルチターゲットＣコンパイラの構成の一手法について述べる。このシステムでは，ＣＧＧを使用し，新しい目的計算機に対するクロスコンパイラの作成を容易にしようとするものである。このＣＧＧは，ＰＱＣＣをモデルとしているが，サポート言語をＣ言語に限定し，定理や実現のデータベースを使用せずに，ＣＧＧの簡単化をはかっている。そのシステム全体の構成，ＣＧＧの基本的な考え方を報告する。

2．マルチターゲットコンパイラの構成

マルチターゲットコンパイラ（ＭＴＣ：Multi-target Compiler）の構成上の問題点として，コンパイラの目的計算機に対する依存性がある。この依存性を分類すると，まず，目的計算機のアーキテクチャに依存する部分として，①記憶領域の割当て（例えば，目的計算機がワードアドレスマシンかあるいはバイトアドレスマシンかにより異なる），②式のコード生成，③制御文のコード生成，等である。さらに，コンパイラやオブジェクトコードが使用される環境に依存する部分として，①ファイル形式，②アセンブラ，リンカー，等に分類できる。

これらの目的計算機に依存する部分のほとんどを，コード生成部で吸収することが可能である。したがって，コード生成部を，目的計算機ごとに用意することにより，ＭＴＣが構成できる。さらに，図１のように，コード生成部のなかで，目的計算機に依存する部分を表の形式でまとめ，ＣＧＧでその表を生成するようなシステムとすると，新たな目的計算機をＭＴＣに追加することが容易になる。また，字句解析部，構文解析部，意味解析部のなかで，目的計算機に依存する部

分をフラグで切替えることにする。表1に，本システムで使用しているサブセットC言語[3]のフラグを示す。

表1　サブセットC言語のフラグとその機能

フラグ	機　能
−Bn	1ワードがnバイトのバイトアドレスマシンであることを示す。
−Wn	1ワードがnバイトのワードアドレスマシンであることを示す。
−U	スタックの成長方向がアドレスの小さい方から大きい方へ。
−L	スタックの成長方向がアドレスの大きい方から小さい方へ。

MTCシステムを設計する上で，以下のような目標を設定した。(1) 移植性：CGGを含めたシステム全体の移植性も考慮する。(2) オブジェクトコードの効率：目的計算機上で，レジデントコンパイラが作成できる程度の効率を目標とする。(3) 目的計算機記述の独立性：使用するコンパイラの内部形式やサポート言語に依存しないような目的計算機の記述。(4) 目的計算機記述の最小化：目的計算機の記述が小さくまとまるような形式とする。

(1)の目標のために，CGGを含めたMTCシステム全体をC言語で記述するという方針を採用した。さらに，目的計算機として，変更先のホスト計算機を選ぶことにより，ホスト計算機の変更が容易になる。

(2)の目標のために，コード生成部を表駆動形式とし，その表をCGGが生成する方式とする。コード生成部の入力は，図1に示すように中間形であるので，その中間形の個々のオペレーションに対する生成コード（アセンブラのリースプログラム）をテンプレートとして，CGGが出力する。このCGGの基本的なアルゴリズムを次章に示す。

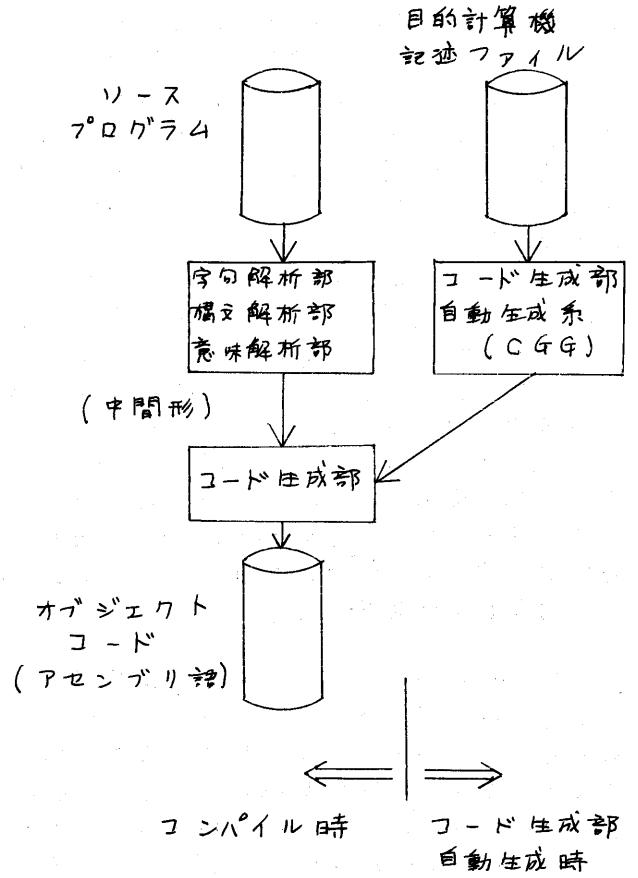

図1　マルチターゲットコンパイラシステム

(3)の目標に対して，目的計算機の意味を記述している目的計算機の記述ファイ

ル（ＭＤ：Machine Description）を使用することによって対応している。このＭＤで記述すべき項目は，図２のようになると考えられる。サポート言語に依存する項目は，スタックポインタやフレームポインタ等の特別な“レジスタ割当て”のみである。Ｃ言語をサポート言語に選んだ場合，register宣言に対して割当て可能なレジスタについて，ＭＤで記述しなければならない。さらに，これらのレジスタの退避，回復を，関数の入口，出口で行なわなければならない。このプロローグコードおよびエピローグコードが，オブジェクトプログラムの実行速度の大きなウェートを占める。しかし，ＣＧＧで，これらの効率良いコードを自動生成することは，非常に難しいので，自動生成系を使用せずに，別の形で，プロローグコード，エピローグコードを与えるようにする。

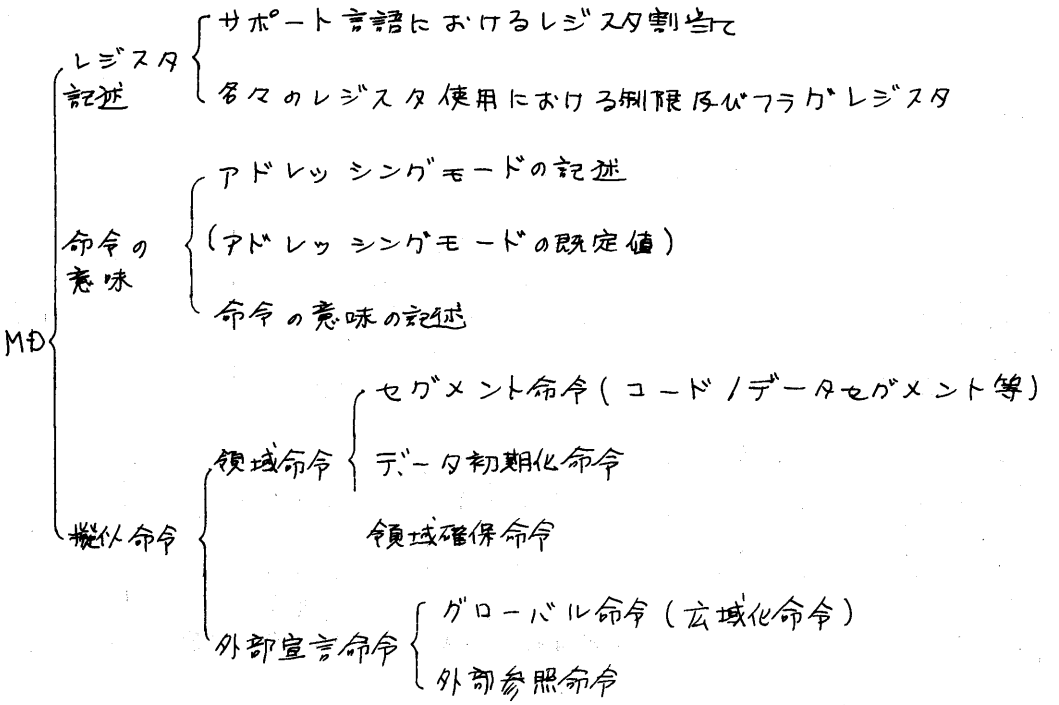

図２　目的計算機記述ファイルで記述すべき項目

ＭＤの規模の最小化という(4)の目標のため，図２に示すように，アドレッシングモードの記述の項目を入れた。アドレッシングモードの既定値における各々の命令の意味を記述し，さらに，アドレッシングモードの記述とその既定値を与える。したがって，アドレッシングモードが多い目的計算機に対してもそれほどＭＤの規模が大きくならない。

3. コード生成部自動生成系

ＣＧＧの基本的な考え方を示すために，まず，ＣＧＧの入出力関係について述べ，そのアルゴリズムを示す。さらに，そのアルゴリズムを使用した例を与える。

3.1 CGGの入出力

CGGが，どのようなテンプレートを生成しなければならないかを図3に示す。図3(a)のようなソースプログラムが与えられ，かつ，オブジェクトコードとして(e)のようなコード生成が要求されている場合を考える。さらに(b)のような中間形が，コード生成部の入力として与えられる場合のテンプレートが，(c)の形式だとする。ここでASSIGN, MINUSは，それぞれ代入，及び減算の演算子であり，(c)の[]は，仮引数を示している。この場合，(d)のような，オブジェクトコード生成のためのデータを，CGGで生成しなければならない。したがって，CGGの入出力は，次のようになる。（図4参照）

入力(i)：テンプレート（図3(c)参照），CGGで，ゴールとして使用する。
入力(ii)：目的計算機のそれぞれの命令の意味，MD（図4参照）。
出力　：入力のテンプレートの意味に一致する命令のデータ（図3(d)参照）。

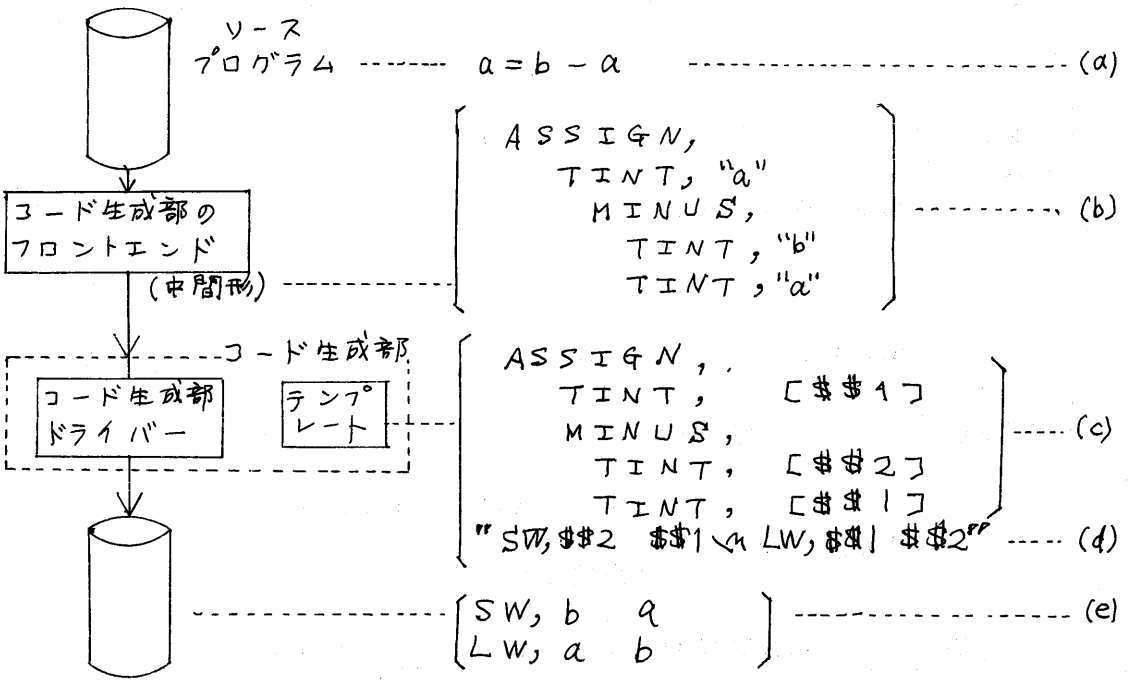

図3　コード生成例

3.2 CGGのアルゴリズム

CGGのアルゴリズムは，以下のように記述される。

step 1　各々の中間形のゴールに対して，後に示す手続きSearchを用いて，意味的に一致する命令系列を探索する。

step 2　探索に成功した命令系列が複数得られた場合，ある評価基準により，最も良い命令系列を，コード生成部へ与える。

手続きSearch

step 1　ゴールと直接一致するMDがあれば，この命令を結合して出力し，終了する。

step 2　直接一致するものがない場合，次の手順で，サブゴールを設定する。

(i)　ゴールが分解可能ならば分解し，それぞれをサブゴールとする

(ii)　ゴールが分解できない場合，ゴールのオペレーションの中心となるものと一致する命令を捜す。得られた命令を使用して，ゴールと同一の意味を持つように，前後で行うべき操作を決定する。これらの操作をサブゴールとする。

step 3　各々のサブゴールに対応するオブジェクトコードの副作用を解析し，どのサブゴールから，探索を行うべきかを決定する。

step 4　各々のサブゴールに対する解を求め，それらの解を順に結合し，出力する。手続き終了。

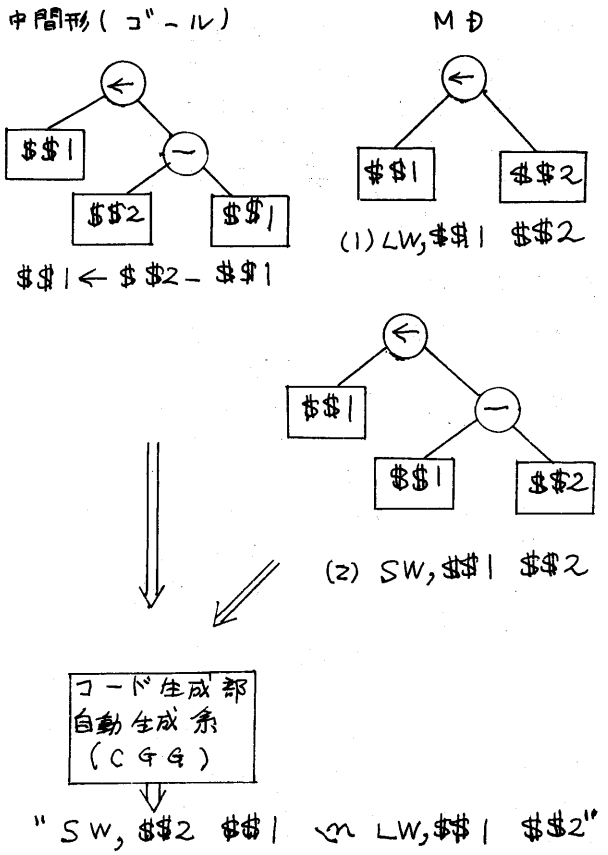

図4　CGGの入出力関係

3.3　自動生成の例

図4のような入力が与えられた場合，上述のアルゴリズムを使用した，コード生成部の自動生成の例を，以下に示す。（図5参照）

(1)　中間形（ゴール）と直接一致する命令をさがす。見つからなかったのでゴールが分解できるかどうか，調べる。

(2)　図5(a)，(b)のように分解できるので，それぞれをサブゴールに設定し，探索を続ける

(3)　(a)，(b)の副作用の有無を調べ，どちらのサブゴールから調べるかを決定する。この場合，(b)で決定した値を(a)で使用するので，(b)のサブゴールを最初に探索する。

(b)と直接一致する意味を持つ命令を調べる。図4の"SW"命令と一致するので，結合して，(b)が(b')になり，その解を出力する。

(4)　(b')の結合により，(a)も結合され，(a)から(a')になる。

(5) サブゴール(a')に対する探索を行い，直接一致する命令(d)が得られたので出力する。ここで，全てのサブゴールに対する解が得られたので終了する。

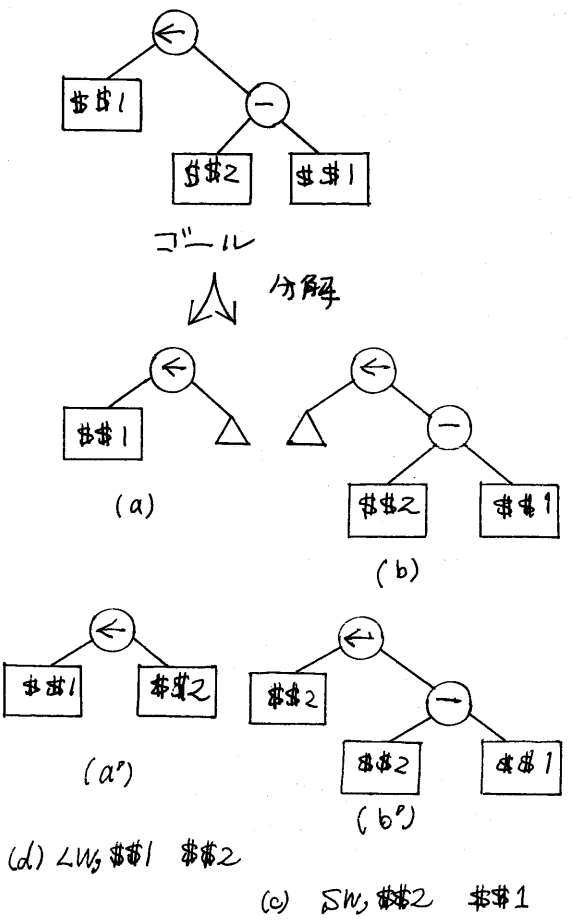

図5 自動生成の例

4. おわりに

現在，CGGのそれぞれのアルゴリズムのチェックのためにPrologを用いながら，CGG全体をC言語で作成中である。サブセットC言語のコンパイラは，完成しており，コメントを含めて，全体で三千行程度である。目的計算機として，PDP 11/60 及び，Melcom Cosmo 700/800を想定している。ホスト計算機も両方の計算機を考えている。

MTCシステムに，クロスアセンブラ及びアセンブラ生成系[4]を追加し，最終的なMTCシステムが完成する。また，本報で示したMDを使用し，C言語の最適化生成系により，最適化部をコンパイラに追加することによって，オブジェクトコードの効率化がはかれる。

末筆ながら，日頃ご指導いただく本学本多波雄教授，名古屋大学福村晃夫教授，同吉田雄二助教授に深謝いたします。

〔参考文献〕

[1] S.C.Johnson: A Tour Through the Portable C Compiler, Unix 7th Edition Vol.2, documentation (Jan. 1979)

[2] R.G.G. CATEL: Automatic Derivation of Code Generation from Machine Descriptions, ACM Transactions of Programming Languages and System, Vol.2, No.2, pp. 173-190 (Apr. 1980)

[3] Ron Cain: A Small C Compiler for the 8080's, 日日J, Vol.5, Issue 5, pp. 5-19 (May 1980)

[4] 国立他：超言語記述によるマイクロプロセッサ用相互クロス・アセンブラ，情報処理論文，Vol. 19, No. 7, pp. 606-613 (Jul. 1978)