Vol. 22 No. 4　　情報処理学会論文誌　　July 1981

スケルトンの作成を容易化した多機種向きコード生成部を持つコンパイラ†

国立　勉††　杉山和弘††　米田實男††

多機種計算機向きの目的プログラムを生成するコンパイラを作成する方法については，二，三の提案が発表されている．しかし，これらの提案においては，コンパイラのコード生成部において必要な，対象計算機のハードウェア記述をレジスタ類と命令セットの定義程度のみに限定しているため，この方法によって生成される目的プログラムは性能の点で実用的なものとはいえない．

この問題に対処するため，ここで提案するシステムでは通常の個別作成コンパイラの作成方法と同様，中間言語命令に対応する機械語命令列（スケルトン）の定義を行うこととする一方，スケルトン記述とそのデバッグが容易となるよう，またレジスタ割当て・管理ルーチンの定義が不要となるようスケルトン記述言語を設計した．

本システムにより，実用的コンパイラ（アセンブラ記述に比べて1.5倍のメモリ量）の作成は約1,800枚のカードで行うことができ，個々にコンパイラを作成する場合と比べて，機種依存のコード生成部作成の生産性を2倍とすることができた．

1. まえがき

コンパイラを数多く作成する要求は次の2点にある．

1）1機種の計算機上に複数の高級言語のコンパイラを実現すること．

2）多機種の計算機向のコンパイラ（コンパイラの走行機種は限定する*）を実現すること．入力言語仕様は特定のものに限定してよい．

前者の場合，その要求は古くからあり，構文解析部の汎用化によりコンパイラ作成の効率化を図る研究が多い[1),2)]．後者の場合，コンパイラ作成の効率化の研究は前者ほど多くはないが，以下の場合重要である．とりわけ①においては，LSI技術の発展に伴い，マイクロプロセッサ（μP）の機種数が増加しているため，多機種に適用できるコンパイラを効率よく早期に作成することが重要である．

① マイクロプロセッサ用クロス・コンパイラ

② コンパイラ等の未整備な専用計算機用クロス・コンパイラ

③ 計算機のアーキテクチャ設計時に実験的に必要となるクロス・コンパイラ

```
MCON: PROC OPTIONS(MAIN('Y',1100H)) ;————①
  DCL (T,R,A) BINP(8),F BIT(8) ;
   DO FOREVER ;    /* INFINITE LOOP */
  DO I=1 TO 250 ;
   CALL TIMER$INT ;
   CALL SUB1 ;
   CALL INPUT(A,230) ;
   IF A>R THEN CALL OUTPUT('FF'H,231) ;————②
        ELSE CALL OUTPUT('00'H,231) ;
  END ;
   END ;          /* INFINITE LOOP */
  TIMER$INT: PROC ;
   L1: CALL INPUT(F,232) ;
       IF F^='01'H THEN GOTO L1 ;
       CALL OUTPUT('00'H,233) ;
   END TIMER$INT ;
  SUB1: PROC ;
   IF T<200 THEN R=T ;
            ELSE R=1000-T*4 ;
   END SUB1 ;
  END MCON ;
```

(注) ① の MAIN 内パラメータについては表2の MAIN 中間言語命令を参照．
② の 'FF' H は16進表現ビット列を示す．

図1 PMP-C 言語によるプログラム例

Fig. 1 Sample program written in PMP-C language.

これらの場合に特に必要とされる言語はシステム開発用言語であり，μP においては PL/I に類似した PL/M などの言語が一般化している．言語仕様を1種に限定すると構文・意味解析部を固定化でき，言語に依存する目的プログラムの最適化を小規模に実現できる．さらに対象計算機に依存する部分（主にコード生成部）のみを機種ごとに作成することでクロス・コンパイラが作成でき，効率的である．

ここでは以上の考えのもとにコンパイラ生成システム CGEN を開発した．

作成されるコンパイラは PL/I 類似の言語仕様

† A Compiler with Multi-Target Code Generator for Facilitated Writing of Skeleton by TSUTOMU KUNITACHI, KAZUHIRO SUGIYAMA and TOMIO YONEDA (Data Communication Development Division, Yokosuka Electrical Communication Laboratory, N.T.T.).

†† 日本電信電話公社横須賀電気通信研究所データ通信研究部

* コンパイラの走行計算機以外の計算機の機械語プログラム（目的プログラム）を出力するコンパイラをクロス・コンパイラと呼ぶ．

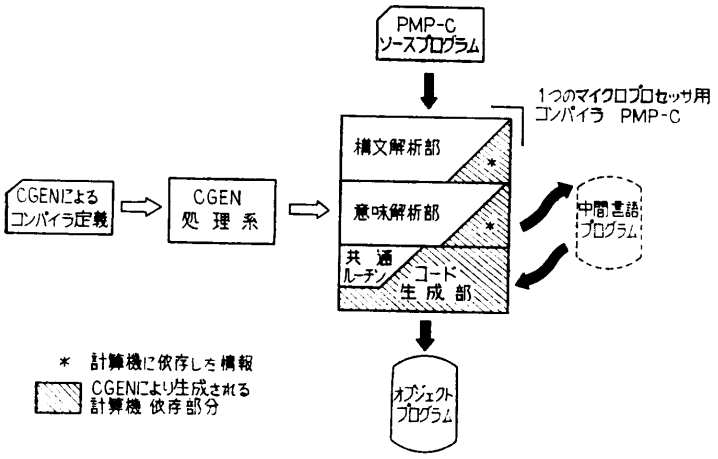


図 2 PMP-C と CGEN のシステム構成
Fig. 2 Organization of the PMP-C and CGEN system.

(PMP-C) を持つ．ソースプログラム例を **図 1** に，システムの構成を **図 2** に示す．

多機種の計算機向のコンパイラを効率的に作成する提案は二，三ある[3)~5)]．しかしこれらの提案においては，コード生成部に必要な，対象計算機のハードウェア情報をレジスタ類と命令セットの定義程度のみに限定しているため，生成される目的プログラムの性能の点で実用的なものとはいえない．この問題に対処するため，本システムでは個別作成コンパイラと同様，中間言語命令に対応する機械語命令列（スケルトン）の定義をコンパイラ作成者（以下では機種依存部のみ作成する者についてこう呼ぶ）に負担させ，その一方で，コンパイラ作成の容易化を図るため，スケルトンの記述とそのデバッグが容易となるようにし，かつレジスタ割当て・管理ルーチンを汎用化し定義不要とした．

本システムの特徴は次の 2 点に集約できる．

（1） コンパイラ作成の容易さ……5 機種に適用した結果，コンパイラ作成が約 1,800 枚のカードで行え，1 機種の個別作成（以前の経験に基づく）に比べると，機種依存情報の記述量・工数が1/2となった．

（2） 実用的レベルの目的プログラム性能……3 本総計 20 k ステップ程度のソースプログラムのコンパイル結果から，目的プログラムの所要メモリ量はアセンブラ言語記述に比し，1.5 倍～2.0 倍（PL/M 等と同等）に抑えられていることが判明した．

本システムは主に μP を対象に検討されたが，基本技術は汎用計算機にも適用できる．対象とした機種のアーキテクチャの条件を **表 1** に示すが，これは市販の 8 ビット，16 ビットの有力な μP を大部分対象としている．

本論文では **2** 章でコンパイラの多機種適用の技術を，**3** 章で中間言語仕様について述べ，**4** 章で実現したコンパイラのコード生成部の構成を，**5** 章で CGEN 言語仕様を，**6** 章で適用例と評価を述べる．

2. 多機種適用の技術

高級言語を多機種に適用する技術の 1 つとして，コンパイラの移植技術[6),7)]がある．これは移植の対象機種上でコンパイラを走行させることを目的としており，対象機種の既存ソフトウェア（言語処理系，ユーティリティ）を駆使して行われる．

一方，本研究は既存ソフトウェアの少ない開発直後の機種向のコンパイラを一つの汎用計算機（ここでは DIPS）上に効率よく早期に作成（主にコード生成部の作成が主体）することを目的としている．

本研究と同じ目的を持ちながら，方式の異なるものが数例[3)~5)]あるが，これらの方式では計算機のハードウェア記述だけをもとに，中間言語命令を機械語命令列に展開していく．この方式はスケルトン定義不要で，かつ複雑なレジスタ割当て・管理ルーチンも定義不要である利点をもつが，コード生成部による機械語命令の意味の完全な認識が必要なため，以下の問題をもつ．すなわち，

（1） μP 用高級言語では割込み，入出力など μP のハードウェア制御を直接記述する必要があり，コンパイラでそれら機械語命令の意味を十分認識することは困難なため，μP 用コンパイラの実現性に問題がある．

（2） μP にはレジスタ構成や命令機能の整備され

表 1 システムの適用可能アーキテクチャ
Table 1 Applicable architechture of the PMP-C and CGEN system.

1	8 ビットごとにアドレスの付与されるバイト・マシン
2	データのレジスタ間接アドレシング可能なマシン
3	レジスタ間接分岐可能なマシン
4	ベース・レジスタとオフセット（オフセットはメモリの一部分のみ指定）による分岐のみ可能なマシン（DIPS, IBM 370 など）でないこと．
5	ページ・アドレシング（固定ページまたはプログラム・カウンタ近傍ページ）のマシン（Nova など）でないこと．

表 2 代表的な中間言語命令
Table 2 Typical examples of intermediate language instruction.

	中間語命令	オペランド(注)	意味
算術演算	BIN ♯ ADD 1	♯OPR 1, ♯OPR 2, ♯OFL	♯OPR 1+♯OPR 2→TS (一時結果) ♯OFL はオーバ・フロー時のとび先ラベル
	MADD 2	♯OPR 1, ♯OPR 2, ♯OFL	メモリ直接加算. ♯OPR 1+♯OPR 2→♯OPR 1 (♯OPR 1 はメモリ・オペランド ソース・プログラム例 A=A+5)
転送	LOAD 1	♯OPR 1, ♯OPR 2	2進, ビット列データのロード
	PTR ♯ STORE	♯OPR 1, ♯OPR 2	ポインタ・データのストア
実行制御	CHAR♯IF	♯LBL, ♯OPR 1, ♯OPR 2, ♯REL, ♯OPR 3	♯OPR 1 と ♯OPR 2 の文字列比較 (関係を ♯REL で, 文字列長を ♯OPR 3 で表わす) による分岐 (分岐先 ♯LBL)
	GOTO	♯LBL	♯LBL へ分岐
	MAIN	♯ENT, ♯PN, ♯PRM 1,…,♯PRM$_i$	メイン手続きのプロローグ (♯ENT は手続名, ♯PN はパラメータ数, ♯PRM$_i$ は機種特有のパラメータで割込みや電源 ON 時の環境に対応したコンパイル処理を指定する.)

(注) 各オペランドはオペランド・テーブルへのポインタか即値である.

ていないものがあり，機種依存の目的プログラムの最適化が困難である.
などである.

以上のことから，本研究では，通常の方式と同様コンパイラ作成者が機械語命令の意味を考慮してスケルトンを定義するものとした．このようにすると，前述の方式よりコンパイラ作成に多くの労力を必要とすることになるが，それでも通常の方式よりコンパイラ作成を容易化できるよう，次の点に留意した.

① スケルトン定義の対象となる中間言語命令を低レベル化し (3章)，スケルトン記述をデシジョン・テーブル記述とする (5章) ことにより，記述を容易化した.

② レジスタ割当て・管理ルーチンを汎用化し，それらルーチンを定義不要とした (4章).

③ スケルトンに指定された機械語命令のレジスタ・オペランド使用法のコンパイル時検証により，スケルトンのデバッグを容易化した (4章).

3. 中間言語仕様

ここでは，コンパイラ作成者に示される中間言語仕様について述べる.

中間言語仕様はソース言語仕様 (PMP-C 言語) が大部分機種共通* であることから大部分機種共通である.

中間言語のレベルとしてはいくつかの案が考えられるが，ここではスケルトン記述が容易となるよう，できるだけ低レベルなものとした．すなわち，コンパイラ共通部で翻訳処理を十分行い，オペランド属性別の中間言語命令——たとえば加算命令を2進，10進，単精度／倍精度で区別——とした．また，データの型，精度により区別される仮想レジスタが無限に存在すると仮定して生成される3つ組形式[8] の命令とした．中間言語命令は 69 個あるがそのうち代表的なものを表 2 に示す.

* 機種依存仕様は PROC 文 (手続き宣言文) のパラメータ，機種依存組込みサブルーチン等に局所化されている.

4. コード生成部の構成

ここでは実現したコンパイラのコード生成部の構成，とくにレジスタ割当て・管理ルーチンがどのように汎用化(機種共通化)されているかについて述べる.

4.1 コード生成部の概要

図3にコード生成部の構成を示す．主制御ルーチンは逐次中間言語命令を入力しては，それに対応するスケルトン処理ルーチンを呼び出す．スケルトン処理ルーチンは CGEN 言語記述のスケルトン定義部より生成される．データ接近ルーチンは中間言語命令のメモリ上オペランドのアドレシングを，対象機種のメモリ・アドレシングで行えるようにするもので，いくつかの命令——たとえば，アドレス修飾部をアドレス・レジスタにロードしたり，2つのアドレス修飾部 (ソースプログラム上のポインタ修飾に対応) を1つのアドレス・レジスタにロードするため加算したりする命令——の出力を伴う．このルーチンはスケルトン処理ルーチンのサブルーチンとなり，CGEN 言語記述の

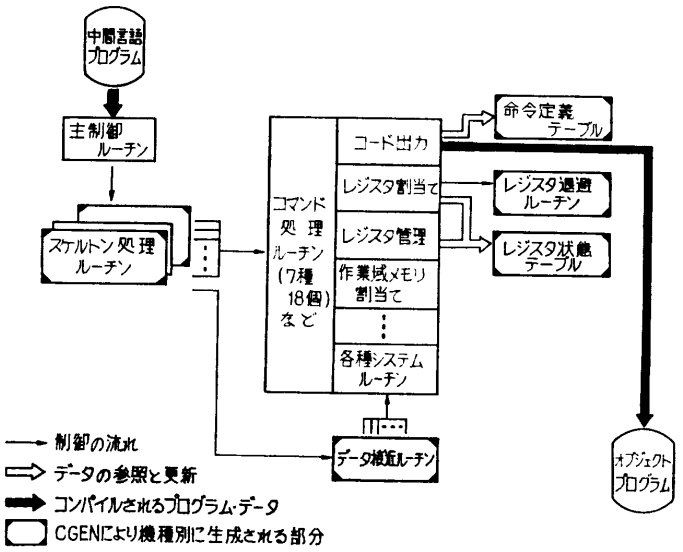


図 3 コード生成部の構成

Fig. 3 Organization of code generator.

データ接近定義部より生成される．同様にレジスタ退避ルーチン（**4.2.1** 参照）も CGEN 言語記述の対応部分より生成される．コマンド処理ルーチンなどは機種に依存しない共通ルーチンであり，このうちレジスタ割当て・管理ルーチンについて以下で説明する．

4.2 レジスタ割当てと管理

レジスタの割当ては2項演算の両オペランドがメモリ上のとき，一方をレジスタにロードする時などに必要となる．

レジスタの割当ては，機械語命令の意味をシステムで認識しないのと同様の理由（**2** 章参照）から，コンパイラ作成者が必要な契機で必要なレジスタ（またはレジスタ・クラスの中の1つ）の割り当て要求コマンドを記述して行うこととした．

レジスタ割り当て要求コマンドに対応するレジスタ割当てのシステム・ルーチン（コンパイラ作成者は定義不要）は，要求されたレジスタがどのような使用状態となっているかを調べ，各状態に応じた割当てを行う．割当て後はそのレジスタの状態を変更するために，レジスタ管理のシステム・ルーチンを呼び出す．

4.2.1 割当て方式

割当てを制御するレジスタの状態は次の通りである．

ⅰ）空き状態
ⅱ）アベイラブル状態
ⅲ）テンポラリ状態
ⅳ）マスク状態

ⅰ）はレジスタ上に有意なデータが格納されていない状態である．ⅱ），ⅲ）は有意なデータと関係（同値関係と呼ぶ）づけられている状態で，ⅱ）はメモリ上データ（入力ソースプログラム中の変数など）と同じデータをもつレジスタの状態であり，データを参照するときメモリのかわりにレジスタを代用できる．ⅲ）は演算結果を格納する状態で，メモリ上には同じデータがない状態である．ⅱ），ⅲ）の関係はいずれも，現在処理（コード生成）中の中間言語命令以前に処理したどれかの中間言語命令の処理で成立するものとした．中間言語命令は共通であるのでⅱ），ⅲ）の関係は自動的に設定される．ⅳ）の状態のレジスタは，（a）展開処理中の中間言語命令のオペランド自身，（b）展開処理中の中間言語命令のオペランドのアドレス修飾部，（c）レジスタ割り当て要求コマンドで割り当てたレジスタ，のいずれかであり，レジスタ割当ての要求に対し，特別な場合を除いて割当てが抑制される．

レジスタ割り当ての優先順位の基本はⅰ），ⅱ），ⅲ）の順であるが，次の2点が例外である．

（1）レジスタ退避ルーチンでの割り当て要求時——レジスタ退避定義は，ⅲ）の状態のレジスタを退避して割り当てる時に呼び出されるので，ここでの割り当て要求にⅲ）のレジスタを対象とすることはできない．

（2） スケルトン中で1回のみ接近（参照または更新）する中間言語命令のメモリ・オペランドに対するデータ接近処理（データ接近ルーチン）での割り当て要求時——要求されるアドレシング・レジスタはメモリ上データの接近に1度だけ使用されるレジスタであり，それがデータ・レジスタとして（μP の場合，レジスタがデータ用にもアドレシング用にも使われることが多い）コンパイル処理中のスケルトンですでに使用中（ⅳ）の状態）であっても，有意なデータが格納されていない場合* には割当て，レジスタの有効利用を図る．データ接近ルーチン中で割当てを行い，アドレス・データをロードしても，1回のメモリ・アクセスで用済みとなるので，スケルトン定義部側からは割当てが行われることを意識する必要がない．割り当ての優先順位はⅰ），ⅳ），ⅱ），ⅲ）である．

* 参照可能回数（**4.2.2** 参照）が0のとき

以上がレジスタの割り当て方式であるが，実際にはさらにレジスタ退避ルーチン，データ接近ルーチンで本来必要となるレジスタ(前者は退避先レジスタ，後者はアドレシング・レジスタ)以外の作業用レジスタ(前者で退避域のメモリ・アドレシング・レジスタ，後者でアドレス加算用レジスタなど) も必要となり，少し複雑化したがここでは省略する．

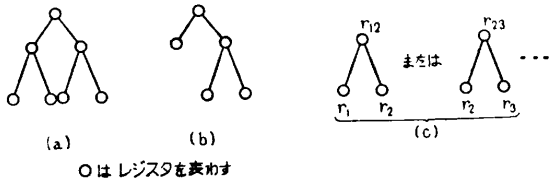


図 4 種々のレジスタ構成

Fig. 4 Various types of register organization.

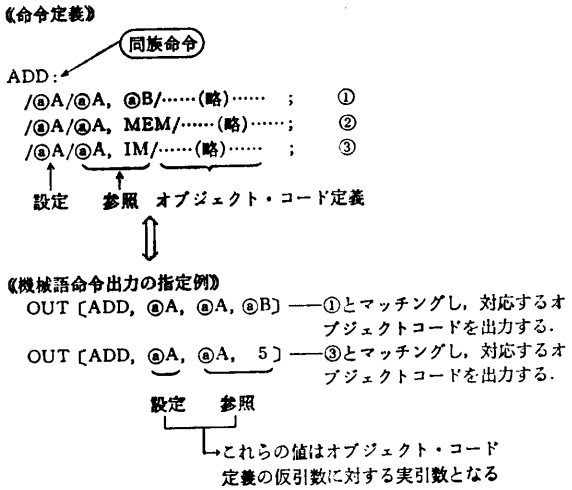


図 5 命令定義と機械語命令出力

Fig. 5 Definition and outputs of instruction.

また **図 4** に示すような種々のレジスタ構成（ペア構造）においては，その部分要素のみ割り当てるとき，全体を割り当てるときの各々について，現在の使用状態が，分割状態か統合状態かの把握，分割状態のとき，一方と他方の使用状態に差異はないかの把握，割当て後の分割／統合状態への移行などの処理も必要となった．

4.2.2 参照可能回数

上記レジスタ割当てを実現し，またレジスタの使用法の正当性を確認するため，参照可能回数 (Ref. no.) の管理を行っている．

図 5 に示すように，機械語命令出力用コマンドにより機械語命令を出力するごとに，レジスタに新規データが設定された場合は Ref. no. を 1 とし，レジスタを参照した場合は Ref. no. を 1 減じる* ようにしている．

Ref. no. は自動更新されるのが基本であるが，複合命令などでは参照／設定の関係を論理的に矛盾なく定義し難いものもあるので，コンパイラ作成者が管理し更新するコマンドも設けた．

Ref. no. のコンパイル時の追跡により，たとえばデータを設定（ロード）しておきながら，1 度も参照せず再び新規データを設定したりするような目的プログラムの——すなわち，スケルトン定義の——誤りを検出できる．

```
/*-------------------------------------------------*/
$REGISTER
/*-------------------------------------------------*/
/*------- REGISTER CLASS DEFINITIONS --------------*/
R-WR1: @C,@E;                                    ①
R-WR2: @A,@C,@E;
R-WR3: @B,@D;
R-WR4: @A,@B,@C,@D,@E,@H,@L;
R-WR5: @A;
R-PR1: @BC,@DE;
R-PR2: @BC,@DE,@HL;
R-PR3: @DE,@HL;
R-IX: @IX,@IY;
/*-------- REGISTER ORGANIZATIONS AND CODES -------*/
@A:111B; @B:000B; @C:001B; @D:010B; @E:011B;
@H:100B; @L:101B;                                  ②
@BC[@B,@C]:00B; @DE[@D,@E]:01B; @HL[@H,@L]:10B;
@IX:01B; @IY:11B;
```

(注) ① はレジスタ・クラス (R-WR 1) の要素 (レジスタ) ⓐC，ⓐEの定義，② はレジスタのバイナリ・コードの定義およびペア・レジスタの構成 (例：ⓐBC は ⓐB と ⓐC のペア) の定義である．

図 6 レジスタ定義部の例 (Z-80)

Fig. 6 Example of register definition part (Z-80).

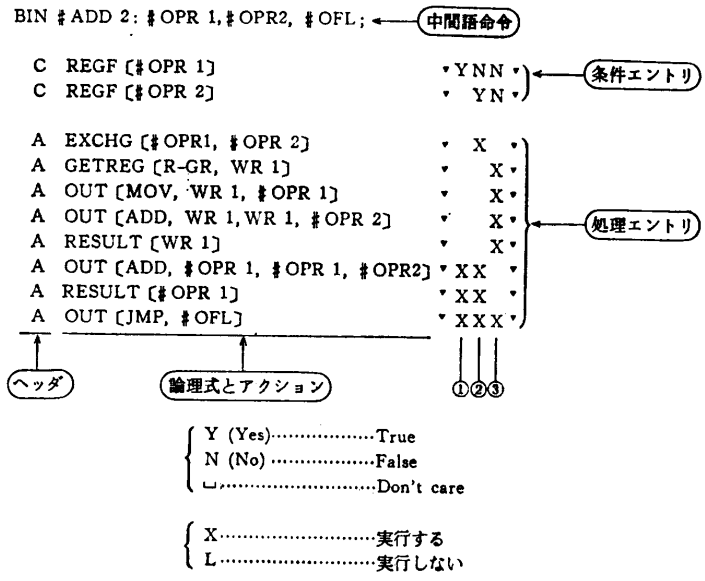
5. CGEN 言語仕様

機種依存部定義言語 CGEN[9] の主な定義部について以下に述べる．

レジスタ定義部では，レジスタの構成（レジスタ・クラスとペア構成）およびコードを定義する．Z-80 の定義の一部を **図 6** に示す．これにより，レジスタ状態テーブル（図 3 参照）が作成される．

命令定義部では，機械語命令をオペランドの如何にかかわらず演算動作の同じものをまとめて定義する．図 5 の①，②，

* 0 としないのは複数回参照するために Ref. no ≥ 2 となっていることがあるためである．



(注) ①は #OPR 1 がレジスタ上のデータの場合で，ADD 命令を出力し，オーバ・フロー時の処理用命令を出力することを示す．RESULT は本中間語命令の実行結果が #OPR 1 (レジスタ) 上に残ることを示す．
②は #OPR 1 がレジスタ上のデータでなく，#OPR 2 がレジスタ上のデータの場合で，#OPR 1 と #OPR 2 の番号つけかえにより，①と同じ処理を行うことを示す．
③は #OPR 1, #OPR 2 のデータが ①, ② のいずれでもない場合で，レジスタを一つ割当て，そこへ #OPR 1 をムーブする命令を出力し，ADD 命令を出力することを示す．

図 7 スケルトン定義の例
Fig. 7 Example of skeleton definition part.

③は ADD 命令のオペランドの組み合せを示す．これらは機械語命令出力用コマンド (OUT) の引数と比較され，一致するもののビット・パタンが生成される．このように定義することにより，Ref. no. の自動更新が可能となる．

スケルトン定義部では，中間言語命令のオペランド状態（レジスタ，リテラル，メモリ等）により出力機械語命令列を定義する．この場合分け処理はデシジョン・テーブルによる記述が適当である．デシジョン・テーブルの採用により，場合分けに漏れが少なくなり，また重複記述も少なくなるため記述量を削減できた．

図 7 にスケルトン定義の一例を示す．

データ接近定義部では中間言語命令のメモリ・オペランドをアクセスできるようにする処理を定義する．この処理においても機械語命令が出力されることがあり，スケルトン定義部と同様なコマンドを使用できる．

レジスタ退避定義部ではレジスタ割り当てルーチンから呼ばれるレジスタ退避処理を定義する．たとえば，アキュムレータを直ちにメモリへ退避せず別の作業用レジスタに退避して最適化を図る処理も記述できる．

6. 適用例と評価

6.1 記述性と生産性

表 3 に CGEN 言語で定義した各機種向コンパイラの記述量を示す．

本記述は大部分がコード生成部における機種依存部である（図 2 参照）．

以前の経験では，CGEN 言語を考えず，数機種について，汎用言語（手続きの記述に使用）とアセンブラ・マクロ（スケルトン等のテーブル記述に使用）を使用する，通常の方式を採った．以前の経験に比べ，本方式では機種共通部について，汎用化による記述量の増加分が約 6.7 k ステップ（汎用言語のステップ数）であり，機種依存部について，1 機種の記述量節減分が約 2 k ステップであった．また，作成工数もほぼ記述量と比例関係にあった．つまり，4 機種以上作成する場合は本方式が有利となる．

この理由としてはスケルトン定義部をデシジョン・テーブル記述とする等，記述の簡単化を図ったこと，およびコンパイラ走行時に，目的プログラム中のレジスタの使用法の検証（参照可能回数による）や各 CGEN コマンドの使用法の検証を行い，コンパイラ記述に関する誤り検出を行ったことなどが考えられ，CGEN によるコンパイラ作成の記述性と生産性が優れていると考える．

なお，コンパイラのデバッグには，PMP-C 共通ソース・プログラムをテスト・プログラムとし，マイク

表 3 コンパイラ定義のカード枚数（コメントを除く）
Table 3 Numbers of cards for compiler definitions in CGEN (Excluding Comments).

マイクロプロセッサ	カード（千枚）
Z-80	1.9
8085	2.0
8086	1.7
μCOM 1600	1.5
DIPS CCP*	1.1

* Commucation Control Processor

* 増加分には汎用レジスタ管理ルーチンの記述量のほかに，CGEN によるコンパイラ記述に関する誤り検出機能も含まれている．

ロプロセッサ・シミュレータ[10]で目的プログラムを走行確認できたことが有効となったことを付け加えておく.

6.2 目的プログラムの効率

同一の目的のプログラムをPMP-Cコンパイラとアセンブラで書く場合のプログラムの所要メモリ量を3本 20 kステップ（コンパイラ記述）程度のプログラムで比べると 1.5～2.0 倍であった.

コンパイラの目的プログラムのメモリ量を削減するには以下のような方策が考えられる.

（i） フロー解析により，変数・共通式に対して広域レジスタ割付をすること，またはコンパイラ言語仕様上，変数をレジスタに固定的に割り付けて宣言できること.

（ii） コンパイラ言語仕様上，サブルーチンの引数の授受方法を指定できること.

（iii） コンパイラ言語仕様上，機械語記述ブロックの記述を可能とすること.

（iv） レジスタの同値関係更新を中間言語命令単位で行わず，1機械語命令の出力単位で行うこと.

いずれも処理方式上の大きな変更を要するが，(i)，(ii)は機種共通部の改造が中心であり問題ない．これに対し，(iii)では機械語記述ブロック内で記述されたレジスタの使用法まで管理する必要があるため，機種依存部を局所化し CGEN で定義する今回の方針では採用できない．また現方式が機械語命令の意味をシステムで認識しない前提である限り，(iv)を実現するには機械語命令の出力のたびに，どのような同値関係更新を行うかコンパイラ作成者が指定しなければならず，大きな記述負担となる．この点はコンパイラ生産性の向上と最適化とのトレード・オフであり，現方式の限界とも言える.

なお，通信制御処理などビット列および文字列処理の多いものに対し，ビット処理用の組込みサブルーチンを CGEN 言語で定義したり，文字列転送をインライン展開および実行時ルーチン呼出しの両方で実現すること* などの最適化を図ることは現方式で容易に対処できる.

7. あとがき

コンパイラの機種依存部（とくにコード生成部）を局所化して，これを専用言語 CGEN により定義し，各種 μP のコンパイラを能率よく作成する方式について述べた．この方式ではコンパイラのオブジェクト・コードの効率低下も少ない.

今後の課題としては以下のものが考えられる.

（i） 5章に述べた，採用可能な最適化の実現

（ii） 適用対象機種の拡大，また OS を持つ機種へ適用するための OS インタフェースの実現

（iii） コンパイラ言語仕様の拡張

（iv） CGEN 言語仕様の改良，とくにデータ接近定義部等のデシジョン・テーブル化

（v） 構文解析部の改良および，コード生成部でのレジスタ使用法などの動的検証を，指定により中止しコンパイル時間の短縮を図ること.

最後に，本論文の執筆に懇切な指導を頂いた横須賀電気通信研究所データ通信研究部飯村統括役および本システムの実用化に有力な助言を頂いた元応用プログラム研究室藤田調査役に深甚なる謝意を表します．また，本システムの製造を担当された日本電気（株）の関係各位に感謝致します.

参考文献

1) 井上：コンパイラ・コンパイラ，産業図書 (1970).
2) Mckeeman, W. M. : A Compiler Generator, Prentice-Hall, Inc., Englewood Cliffs, N. J. (1970).
3) Miller, P. L. : Automatic Creation of a Code Generator from a Machine Description, Project MAC report, No. TR-85 (1971).
4) Newcomer, J. M. : Machine-Independent Generation of Optimal Local Code, Ph. D. thesis, Carnegie-Mellon University (1975).
5) Fraser, C. W. : A Knowledge-Based Code Generator Generator, SIGPLAN Notices (1977).
6) Richards, M.: The Portability of the BCPL Compiler, Software- Practice and Experience, Vol. 1, No. 2, pp. 135-146 (1971).
7) Wirth, N.: Pascal and Portability, Pascal Newsletter No. 2, SIGPLAN Notices, Vol. 9, No. 11, p. 17 (1974).
8) Gries, D.: Compiler Construction for Digital Computers, John Wiley and Sons, Inc., New York Chap. 17 (1971).
9) 国立他：多機種マイクロプロセッサ向コンパイラのコード生成部の汎用化，信学技報，EC 79-2, pp. 11-22 (1979).
10) 神谷他：プログラム・シミュレータ用ハードウェア記述言語，情報処理学会論文誌，Vol. 21, No. 3, pp. 175-182 (1980).

（昭和55年5月22日受付）
（昭和55年12月18日採録）

* 現在はインラインか実行時ルーチン呼出しかの一方のみが排他的に使用できる.