北海道大学工学部研究報告
第73号（昭和49年）

Bulletin of the Faculty of Engineering,
Hokkaido University, No. 73 (1974)

FORTRANによるコンパイラの記述に関する研究

山田茂樹* 須戸 満** 高城洋明**
栃内香次** 永田邦一**

（昭和49年6月29日受理）

Studies of writing a FORTRAN compiler using FORTRAN language

Shigeki YAMADA Mitsuru SUDO Hiroaki TAKAJO
Koji TOCHINAI Kuniichi NAGATA

(Received June 29, 1974)

Abstract

An experiment with the intent of writing a FORTRAN compiler using FORTRAN language along with the assembly language is reported.

Although it is known to be desirable to use high level languages, such as FORTRAN, in compiler writing for good programming productivity, it has been pointed out that the compilers written in such a manner are generally inefficient.

In this experiment, an attempt has been made to utilize the FORTRAN as much as possible, wherever it is proper, and the assembly language was used only where a high efficiency is required.

As result of our experiment, a good productivity was obtained, and some approaches to develop high level languages for writing system programs such as compilers were also proposed.

1. 序　　言

コンパイラなどのシステム・プログラムを作成する場合，プログラムの効率がよいことが要求されるとともに，高度でかつ非常に幅の広いプログラミング技法が必要なので，一般にはハードウェアの特徴を生かしやすいアセンブリ言語で記述される場合が多い。しかし，アセンブリ言語を用いると，システム作成に要する工数が莫大になり，しかもシステムの規模が大きくなるにつれて，プログラマ1人当たりのプログラム作成量は減少してゆく傾向にある。したがって，アセンブリ言語を使用してシステム・プログラムを作成するという従来行われてきたソフトウェア開発手法には限界がある。これを解決する手段の1つとして，より密度の高いシステム記述用プログラム言語の開発が急がれているが，いまだ決定的なものはあらわれていない。

本研究は科学技術計算用言語であるFORTRANをシステム記述言語として用い，システム・プログラムのうち，とくにコンパイラを対象にして作成を試みたものである。ここで用いた方法は，特に高い能率が要求される部分や，FORTRANで記述できない部分のみをアセンブリ言語で記述するという方法により，上述の限界を乗り越える一つの試みを行ったもので，今後システ

* 日本電信電話公社
** 電子工学科 電子機器講座

ム記述専用の言語を開発するために必要ないくつかの結果と問題点が得られたので，ここに報告する。

システム・プログラムのうち，特にコンパイラに関しては，その作成の簡単化の方法がいくつか考えられているが，それらをまとめてみると以下に示す4種に大別できる。

(1) 簡単化に十分留意した上でアセンブリ言語で書く。

(2) コンパイラ・コンパイラなどにより自動作成する[7),14)]。

(3) 以前に作られた古い計算機のコンパイラを，新しい計算機用に自動変換して作り出す[5)]。

(4) コンパイラまたはシステム・プログラムを記述する言語（システム記述用言語）を使って作成する[6),8)~12),20~22)]。

(1)は従来から行われていた方法の改良にすぎず，前述のような欠点がある。(2)のコンパイラ・コンパイラとは，入力として作成したいコンパイラに関する情報を与えると，出力として希望のコンパイラを作り出すプログラムのことをいうが，現在のところコンパイラのごく一部しか自動作成できない。(3)の方法に関しては，いくつかの実験例[5)]が報告されているが，実用のレベルには達していない。したがって現時点では(4)の方法が工学的に重要であり，本研究ではこれを基本にしてコンパイラ作成簡単化の実験を行うことにした。

2. FORTRAN によるコンパイラの作成

アセンブリ言語でシステムを記述すると膨大なステップ数になり，完成までに多大なマンパワーを必要とする。それにもかかわらず，現在までアセンブリ言語が使用されてきたのはつぎのような理由にもとづく。

(1) 細部にわたって能率の良いプログラムを作ることができる。

(2) SUPERVISOR CALL の命令，PERIPHERAL 命令など，ハードウェアと関連の深い命令を自由に使うことができる。

(3) システムを記述するのに適した高級言語として一般に普及しているものがない。

一方，高級言語の持つ長所として

(1) プログラムが書きやすく，かつ高密度なので，生産性が大幅に上がる。

(2) 不注意なミスが入りにくく，デバッグが容易である。

(3) プログラムが読みやすいので，作成分担が容易であり，作成者以外の者がデバッグを行うことが可能である。

(4) 保守が容易である。あとで改良や拡張を加えたい場合などもプログラムの判読が容易なため比較的簡単に行える。

などがあげられる。しかし，これを用いてコンパイラを作成する場合の最大の短所として，書かれたプログラムの能率が悪いことが指摘されている。われわれはこれを解決すするために，システムを主として高級言語によって記述し，機械に依存する部分や，高い能率が要求される部分のみをアセンブリ言語で記述するという方法を採用した。記述用言語としてどのような言語を採用すべきかは重要な問題点であるが，つぎのような理由により，FORTRAN を採用した。

(1) 新しいシステム記述用言語を開発するには，一般に膨大なマンパワーを必要とする[17)]。

(2) 北大大型計算機センターで使用できる言語には FORTRAN, PL/I, ALGOL, COBOL があるが，コンパイルの速さ，デバッグ機能の豊富さなどを考慮すると FORTRAN がもっとも有利である。

また，アセンブリ言語は，使用計算機の関係から FACOM 230-60 の FASP を使用している。

作成を試みたコンパイラは JIS 3000 レベル程度の FORTRAN コンパイラであり，FORTRAN ソース・プログラムを入力として，FASP 言語のプログラムを出力するものである。なお，リンケージ・エディタは，コンパイラやアセンブラなどにより，独立に作り出された相対形式オブジェクト・モジュール間の連絡をとって，実行可能な1つのロード・モジュールを作成するプログラムである。システム全体のブロック図を Fig. 1 に示す。

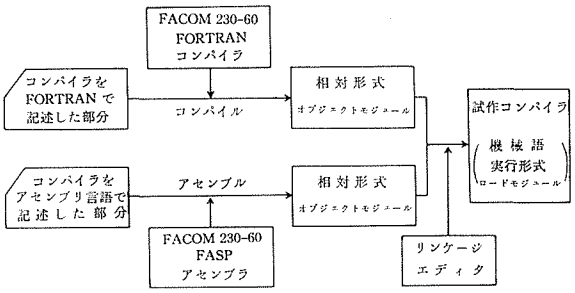


Fig. 1. システムの構成

上述のような方法でコンパイラを作成する場合，ある部分を FORTRAN で記述するか，FASP で記述するかを決定しなければならないが，その具体的規準はつぎに示すとおりである。

(1) メイン・ルーチンは FORTRAN で記述する。

高級言語で記述する利点の1つは，プログラムの見やすさ，読みやすさであるから，なるべく FORTRAN で記述し，記述が困難な部分があらわれたらその都度，FASP で記述したサブルーチンをコールする形式にしている。

(2) ビット処理，バイト処理，ポインタ処理は FASP で記述する。

FACOM 230-60 FORTRAN は，ワード (1 ワード=36 ビット) 単位の処理が基本となっており，ワード中の特定ビットを操作したり，変数の値で示されるアドレスの内容を操作する (間接アドレス指定をする) ことは困難である。そこで，これらはすべて FASP で記述する。

(3) 表の探索，登録，操作は FASP で記述する。

各種の表の探索を行う部分は，FORTRAN でも記述可能であるが，探索して該当する要素を発見した場合，そのアドレスを特定の場所 (ポインタ) へストアするなどの付加的な作業が必要である。さらに表の探索は，コンパイルスピードを決定する重要なファクタになるので，FASP で能率の良いプログラムを作成することにしている。また，表の登録，操作も大部分 FASP で記述している。

つぎに問題となるのは FORTRAN プログラムと FASP プログラム間のインタフェイスであり，前者から後者，または後者から前者への情報の受け渡しが必要である。FACOM 230-60 の場合，これを行うのに，

(1) 引数で受け渡しをする。

(2) FORTRAN, FASP 両方からアクセスできる領域 (COMMON 領域) に情報を入れて受け渡しをする。

の2通りが考えられるが，(2) の方法が命令実行時間は短かく，FASP プログラムも判読しやすく，受け渡し情報の数の修正，変更も比較的簡単になるなど，(1) よりすぐれているので，この方法で処理することとし，FORTRAN, FORTRAN 間，FASP, FASP 間インタフェイスも，同様に (2) の方式を採用している。

3. フェーズの分割

コンパイラは，いくつかの仕事単位から構成されており，その各々をフェーズと呼ぶ。試作コンパイラは3つのフェーズに分割し，Fig. 2 のようにオーバレィを行うことにした。

3.1 フェーズ 1

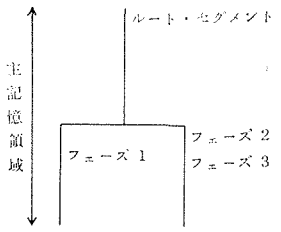


Fig. 2. オーバレイ構造

このフェーズでは FORTRAN ソース・プログラムを文単位で読み，それを作っている文字列を走査して，文を構成する要素を識別し，分解する。これを単語解析という。さらに，これらの要素の組み合わせを調べて，それがどの FORTRAN 文に属するかを判定し，合わせてそこに含まれる文法的な誤りをチェックする。最後に，それぞれの文に固有の処理を行って，つぎのフェーズに渡せる形に情報をまとめる。ここで作り出す情報の大部分は，表の形で記憶される。主要な表を以下に示すが，ここで表に入れるべき一纏まりの情報をエントリ (*Entry*) と呼ぶことにする。

名前表 (*Name Table*): ソース・プログラムにあらわれる各種の名前 (変数名，配列名，関数名など) と，それに付随する情報 (変数の型，関数が，文関数か組込み関数かの区別など) を登録する。エントリは3ワードから構成されている。

文番号表 (*Statement Number Table*): ソース・プログラム中で使用される文番号や，コンパイラが処理の必要上作り出した文番号，およびそれらの付随情報を登録する。エントリは2ワードから構成されている。

次元表 (*Dimension Table*): 配列宣言子で宣言された配列名と各次元の大きさを登録しており，エントリは可変長で2ワードまたは3ワードである。

定数表 (*Constant Table*): ソース・プログラム中で使用された整数型および実数型の定数を2進表示で登録しておく。定数表の内容はそのままの形で，オブジェクト・プログラムのデータ部として出力される。

FORMAT ストリング表 (FORMAT *String Table*): FORMAT 文の最初の左カッコから最後の右カッコまでを記録しておき，オブジェクト・プログラムにそのままの形で出力する。

手続き表 (*Procedure Table*): オブジェクト・プログラムの手続き部分を作り出すために必要な情報を記録する表である。フェーズ3で，この表の中の各エントリを先頭から処理してゆくことにより，オブジェクト・プログラムの手続き部分が作り出される。手続き表のエントリは次の2種よりなる。

(1) ヘッダ語 (*Header Word*): 文の種類や処理手順を識別する語で，1ワードから構成されている。

(2) トレイラ語 (*Trailar Word*): 処理に必要な詳細情報を記憶しておく部分で，通常は数ワードからなる。

EQUIVALENCE 表 (*Equivalence Table*): EQUIVALENCE 文で宣言された情報を保存して，番地の割当ての際に使用する。

算術式表 (*Arithmetic Expression Table*): ソース・ステートメントのうち，算術式の部分をほぼそのままの形でこの表に保存する。ただし，以下のフェーズで演算子間の優先順位にもとづく構文解析を行うのに都合がよいように，ソースステートメントにあらわれる演算子 (+, *, /, −, ** など) は数値に変換され，強い演算子ほど大きな数値が与えられている。

添字表: 配列要素を用いるときに使用される添字式を記憶しておく表で，2種類の表が用意される。

前述のようにフェーズ1ではこれらの表を探索し，必要な要素を登録する処理がほとんどを占める。この処理概要の流れ図を Fig. 3 に示す。

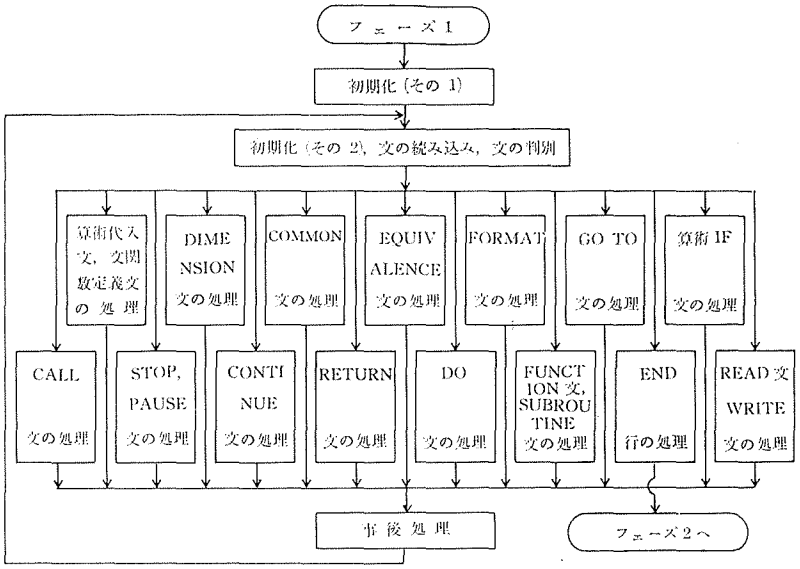


Fig. 3. フェーズ1の処理

3.2 フェーズ 2

このフェーズでは，中間言語の生成を行う。すなわち，フェーズ1で作り出された各種の表をもとにして，フェーズ3でオブジェクト・プログラムを作り出すのに適した形に変換する。具体的には，Fig. 4に示すつぎのような処理を行う。

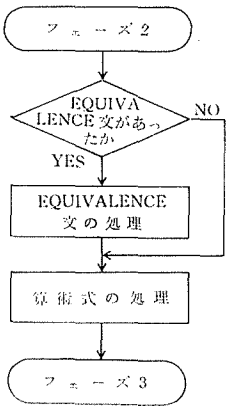


Fig. 4. フェーズ2の処理

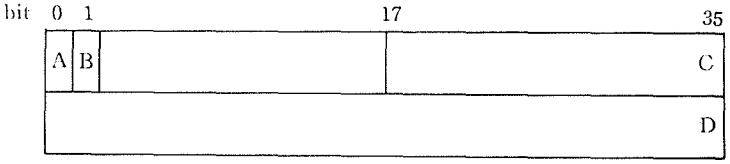
3.2.1 EQUIVALENCE 処理

ここでは，フェーズ1で作成されたEQUIVALENCE表を参照しながら，変数および配列要素に対して番地を割付ける。その処理の概要はつぎの通りである。

(1) EQUIVALENCE表から1ワードのエントリを取り出す。このとき，エントリの第2ワード目には，変数の場合は0が，配列要素の場合は，配列の先頭番地からの相対番地が入っている。

(2) 変数の場合はその変数に，配列要素の場合は配列の先頭(以下ベースと呼ぶ)に一時的に番地0を割付ける。この値をもとに同じグループ内の他の要素のベース番地を計算する。

(3) この一時的な番地と名前表のエントリ番地をEQTMP表(作業用の表)に格納する。こ



A： 共通ブロック内の要素のとき1，それ以外のとき0
B： 配列要素のとき1，それ以外のとき0
C： 名前表のエントリのアドレス
D： 一時に割付けられたアドレスの値

Fig. 5. EQTMP表のエントリの構成

の表は1つのグループについての番地割付けが完了すると空にされる。この表のエントリの形式をFig. 5に示す。

(4) 処理し終ったエントリをEQUIVALECE表から削除する。

以上で1つのグループの処理は終了するが，別々のグループに同じ要素がある場合，たとえば，

EQUIVALENCE (D, G), (G, H)

のような場合は，同じ要素をもつ別々のグループは本来一つのグループとみなさなければならない。すなわちこの文は，

EQUEIVALENCE (D, G, H)

としたのと同じである。そのため，いま処理したグループの要素のどれかが別のグループにあるかどうかを調べるルーチンが用意されている。

(5) つぎのグループから1つエントリを取り出し，EQTMP表に同じものがあるか調べる。

(6) もし同じものがあれば，EQTMP表に入っている値を使って，その要素の番地を計算する。この値をもとにして，同じグループの他の要素の番地を計算する。EQTMP表にまだ登録されていない要素があれば，登録する。

(7) (5)～(6)の処理を，新しくEQTMP表にエントリが追加されなくなるまで続ける。

(8) 以上の処理の終了後，EQTMP表をもとに一時的に割付けられていた番地の調整を行い，最終的な番地を決める。

(9) (1)～(8)までの処理をEQUIVALENCE表のエントリがすべてなくなるまで続ける。

3.2.2 算術式の処理ルーチン

フェーズ2では，フェーズ1で作られた算術式表を処理して，Fig. 6に示すようなエントリを持つ算術命令表を作り出す。このエントリは，手続き表のエントリの場合と同じように1ワードのヘッダ語と0，1，または2ワードのトレイラ語から構成されている。

手続き表のエントリはフェーズ1ではFig. 7に示すように算術式表の対応するアドレスを示すポインタであるが，フェーズ2ではFig. 8に示すように，対応する算術命令表を示すポインタに置き換えられる。さらに算術命令表にステートメント分のエントリを作成したのち，手続き表への戻り先のアドレスを記憶しておく。

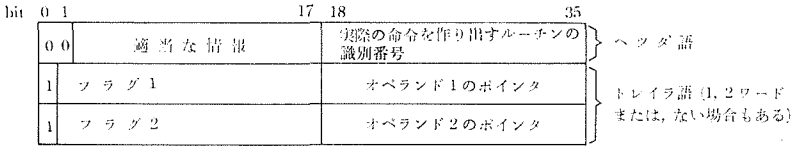


Fig. 6. 算術命令表のエントリの構成

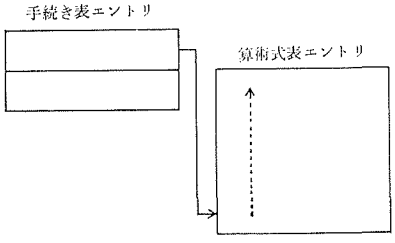


Fig. 7. フェーズ1で作製された算術式表エントリ

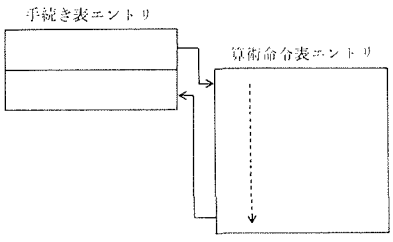


Fig. 8. フェーズ2で作製された算術命令表エントリ

3.3 フェーズ 3

このフェーズでは，手続き表や算術命令表などにしたがって，オブジェクト・コードが出力される。なお，オブジェクト・コードは，FACOM 230-60 のアセンブリ言語である。

フェーズにおける処理の概要を流れ図で示すと Fig. 9 のようになる。FORMAT ストリングや定数の出力は定数出力ルーチンを使用し，それぞれ FORMAT ストリング表，定数表の内容をそのままの形でオブジェクト・プログラムとして出力する。配列用領域や変数用領域を出力する際には，名前表を参照しながら，必要な領域を確保し，番地を割付けてゆく。

さらに，文関数定義文，算術 IF 文，算術代入文および CALL 文の中の算術式で使用した一時記憶場所に対しても領域確保の命令が出力される。

ついで，手続き表の各エントリのヘッダ語に示された識別番号を見て，対応する処理ルーチンへ飛び，トレイラ語の内容にしたがってオブジェクト・コードを出力する。

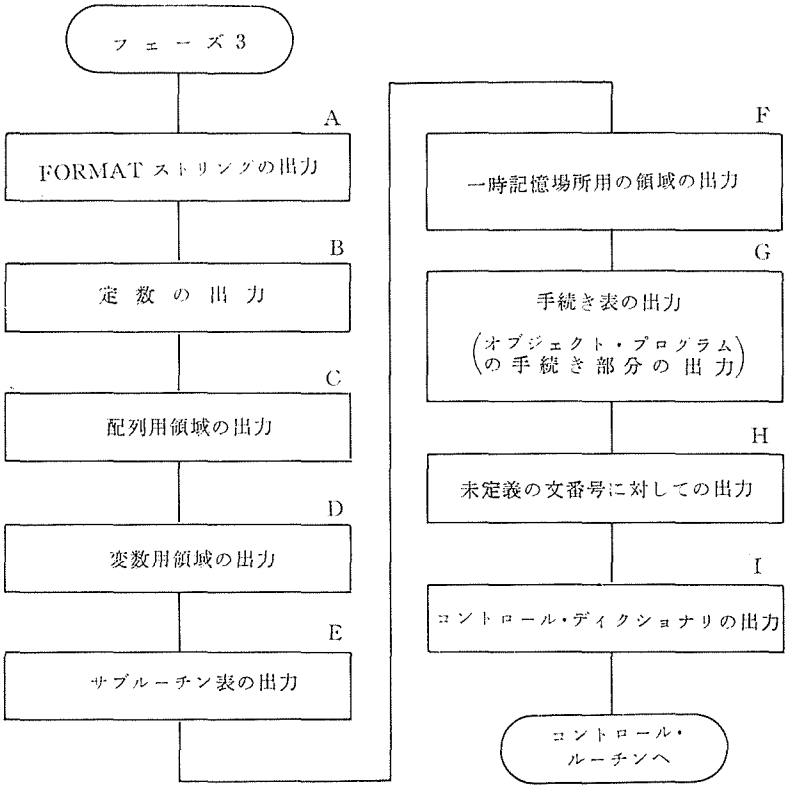


Fig. 9. フェーズ 3 の処理

4. 試作コンパイラの性能

本章では，作成したコンパイラについて，FORTRAN 部分，FASP 部分それぞれのステップ数の詳細と，FORTRAN を使用したことによりどの程度生産性が上昇したか，あるいは，実際にコーディングを行った経験から，システム記述言語にどのような機能が必要とされるかを考察する。そして主記憶占有量，コンパイルスピードなどの面から，試作コンパイラの性能評価を試みる。

4.1 プログラム・ステップの詳細

FORTRAN 記述部のステップ数と FASP 記述部のステップ数を比較するために，各フェーズごとのステップ数を表1，表2に示す。なお，COMMON 領域は FORTRAN, FASP 両方からアクセスできる領域であるので，どちらにも属さない領域としてこの部分のステップ数を除いた数字も示してある。なお，表1で主記憶占有量のうち(C), (D)とは，それぞれ北大大型計算機センターの FORTRAN C コンパイラ，D コンパイラを用いてコンパイルしたときの主記憶占有量を示す。

表 1 FORTRAN 記述部のステップ数

フェーズ	COMMON 領域を含むステップ数	COMMON 領域を含まないステップ数	主記憶占有量（ワード）	
			C	D
フェーズ1～3共通	1,547	1,525	9,276	7,850
フェーズ 1	945	899	7,274	5,946
フェーズ 2	342	316	2,264	1,928
フェーズ 3	310	296	1,826	1,440
FORTRAN 合計	3,144	3,036	20,640	17,164

表 2 FASP 記述部のステップ数

フェーズ	COMMON 領域を含むステップ数	COMMON 領域を含まないステップ数	主記憶占有量（ワード）
フェーズ1～3共通	456	283	272
フェーズ 1	820	665	638
フェーズ 2	951	847	876
フェーズ 3	838	717	756
FASP 合計	3,065	2,512	2,542

4.2 ステップ数に関する考察

FORTRAN で書かれた部分と FASP で書かれた部分とのステップ数を表3に示す。FASP で書かれた部分のうち 18% 程度が，COMMON 領域の記述のために使用されている。

表4は，最終的に機械語に翻訳されたときの，FORTRAN, FASP 両部分の大きさを示す。

表 3 FORTRAN と FASP のステップ数

	FORTRAN ステップ数	FASP ステップ数
ソース・プログラムステップ数	3,144	3,065
COMMON 領域記述部を除いたステップ数	3,036	2,512

表 4 機械語コンパイラ・プログラムの主記憶占有量

	FORTRAN 記述部	FASP 記述部	COMMON 領域	システム・ライブラリ領域	合計（総主記憶占有量）	オーバレイ構造にしたときの主記憶占有量
Cコンパイラ翻訳による主記憶占有量（ワード）	20,640	2,542	8,072	10,962	42,216	34,218
Dコンパイラ翻訳による主記憶占有量（ワード）	17,164	2,542	8,072	10,962	38,740	31,950

表 5 コンパイラ出力の場合とハンドコーディングの場合のオブジェクト・コードの比較

	サブルーチン名	GETCHR	GETNAM	EXPTMP	PACV	平均
I	Cコンパイラ翻訳による主記憶占有量（ワード）	168	174	286	266	
II	Dコンパイラ翻訳による主記憶占有量（ワード）	144	142	230	232	
III	FASPで記述したときの主記憶占有量（ワード）	84	83	120	126	
	比率 III/I (%)	50.0	47.7	42.0	47.4	46.8
	比率 III/II (%)	58.0	58.5	52.2	54.3	55.8

両者を比較するとソース・ステップ数はほぼ同程度であるのに，主記憶占有量は FORTRAN 部分が圧倒的に多い。そこで作成工程がどの程度短縮できたかを考察するために，このコンパイラを構成している 2, 3 のプログラムについて FORTRAN, FASP の両方の言語を用いて記述し，それぞれの場合について生成された機械語の大きさを比較したものを表 5 に示す。ここで FASP で記述した場合には，FORTRAN で記述した場合に比べて平均で 55.8% 程度の主記憶占有量で済むことがわかる。故に，FORTRAN 記述部によるコンパイラ主記憶占有量 17164 ワードは，本来 FASP で記述すれば $17164\times0.558=9578$ ワード程度に減少すると仮定してよいであろう。一方，FORTRAN プログラムは 3036 ステップを要しているので，FORTRAN 1 ステップで $9578/3036=3.15$ ワードのオブジェクト・コードを作り出すと考えられる。FASP 1 ステップは，ほぼ 1 ワードのオブジェクト・コードを作り出すので，FORTRAN を使用すると，生産性が 3.15 倍に上がることになり，高級言語によるシステムプログラム作成の利点が実証される。しかし，これと同時に主記憶占有量は $1/0.558=1.79$ 倍になり，主記憶有効利用の点では問題がある。

以上は FORTRAN-D コンパイラを使用した場合であるが，C コンパイラオブジェクト・コードに関しても，同様の議論を進めると，生産性が 3.18 倍に上がり，主記憶占有量が 2.14 倍になっていることがわかる。

4.3 システム記述言語に必要な機能

試作コンパイラの FORTRAN 記述部分を調べてみると，FORTRAN をその本来の目的とは異なるシステム記述用に使用しているために，表 6 に示すように，通常の数値計算プログラム等とは相当異なるステートメント構成比になっている。この結果や，実際にコーディングを行った経験から，システム記述用言語に必要な特有の機能として以下の各項が考えられる。

(1) システム記述用言語では，算術演算子に優先順位は必要がない。表 6 で A＝B または A＝B±C という簡単な形が代入文全体の 90% 程度を占めており，A＝B+C*D のような形はほとんど出現しないので，数式は左から順次処理するものとし，必要ならばカッコを用いるという規則で十分である。

(2) 浮動小数点演算，べき乗演算はほとんど必要としない。

(3) 配列宣言をした場合，その各々の配列要素が別々の属性をもってもよいようにできると便利である。

(4) 2 進数，8 進数，16 進数に対する演算を記述できること。

表 6　メイン・プログラムのステートメント構成比　　カッコ内は %

	ステートメント数		フェーズ 1	フェーズ 2	フェーズ 3	合計
2	CALL 文		284 (43.0)	23 (41.8)	255 (36.3)	562 (38.9)
	算術代入文	A = B の形	71 (10.7)	13 (23.6)	220 (31.3)	304 (21.0)
		A = B+C の形	11 (1.7)	2 (2.6)	57 (8.1)	70 (4.8)
		A = B=…= C	8 (1.2)	3 (5.4)	23 (3.3)	34 (2.4)
		A = B−C の形	11 (1.7)	3 (5.4)	3 (0.4)	17 (1.2)
		それ以外	0 (0)	2 (3.6)	0 (0)	2 (0.2)
3	IF 文 (算術 IF / 論理 IF)		151 (22.9)	5 (22.9)	35 (5.0)	191 (13.2)
4	GO TO 文		117 (17.7)	4 (7.3)	69 (9.8)	190 (13.1)
5	関数呼出し		0 (0)	0 (0)	21 (3.0)	21 (1.5)
6	入出力文		1 (0.2)	0 (0)	6 (0.9)	7 (0.5)
7	FORMAT 文		1 (0.2)	0 (0)	6 (0.9)	7 (0.5)
8	DO 文		3 (0.5)	0 (0)	2 (0.3)	5 (0.3)
9	宣言文		19			19 (1.3)
10	DATA 文		7			7 (0.5)
11	その他		2 (0.3)	0 (0)	7 (1.0)	9 (0.6)
	合計 (9, 10 を除く)		660	55	704	1445 (100)

(5) 領域のダイナミックな割付け，解放が行えること。FORTRAN で確保する領域は，すべてスタティックであるが，ダイナミックに領域の割付け，解放が行えると，主記憶が有効に利用できることになる。

(6) 変数をポインタとして使用できること。たとえば，INDEX 1 の内容を IUA へストアする場合（これを [INDEX 1]→IUA と書くことにする）IUA=INDEX 1 と記述できるが，INDEX 1 によって示されるアドレスの内容を IUA へストアしたい場合がある。([[INDEX 1]]→IUA) これは間接アドレス指定を行うことになる。また，逆に IUA の内容を INDEX 1 で示されるアドレスへストアしたい場合 ([IUA]→[INDEX 1]) もあり，これらの機能がぜひ必要である。

(7) ビットやフィールドを記述できること。ある語中の特定のビットを ON, OFF したり，あるビット幅のフィールドを演算対象として，たとえば各種のテストを行う等のステートメントが必要である。また，このように抜き出したフィールドに名前をつける機能も必要であろう。

(8) アドレスの取り扱いができること。たとえば，変数 INDEX 3 に，NAMTAB という場所のアドレスを記憶させておき，Fig. 10 のように NAMTAB のアドレスから NAMPTR だけオフセットしたアドレスを指定できると索表処理などに便利である。

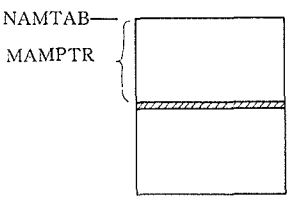


Fig. 10.

(9) スタックや表の処理が簡単に行えること。たとえば，スタックの *pop up* や *push down* などの操作を 1 ステートメントで記述できると便利である。

(10) 強力なデバッグルーチンが備わっていること。

4.4 デバッギング作業経過

完成まで150回程度のテストランを繰返したが，誤りの発生数を原因別に集計すると表7のとおりである。この表によるとFORTRAN部のエラーとFASP部のエラーの種類や傾向が類似している点は興味深い。FORTRAN記述部もFASP記述部もコーディングステップ数は300強で同程度であるから，ステップ当たりエラー発生率はほぼ同じとみてよいであろう。したがって，ステップ当たりの生産性のより高い高級言語のほうがアセンブリ言語よりエラーの発生頻度が相対的に減少することがある程度裏づけられた。

表 7 エラーの原因別分類 (単位はエラーの回数)

FORTRAN 記述部で起きたエラー		FASP 記述部で起きたエラー		その他のエラー
アルゴリズムミス	68 (49.0)	アルゴリズムミス	52 (41.6)	
単純コーディングミス	38 (27.3)	単純コーディングミス	34 (27.2)	
他のサブルーチンとのインタフェイス関係のエラー	22 (15.8)	他のサブルーチンとのインタフェイス関係のエラー	16 (12.8)	
初期値セット忘れ	11 (7.9)	初期値セット忘れ	3 (2.4)	
		ビット，バイト，アドレス処理ミス	18 (14.4)	
		レジスタの回復忘れ	2 (1.6)	
合計 (カッコ内は %)	139	合計 (カッコ内は %)	125	12
ステップ当たりのエラー	0.0442	ステップ当たりのエラー	0.0408	

4.5 試作コンパイラのコンパイル例

試作コンパイラによって作成されたオブジェクト・プログラムをFACOM 230-60 FORTRAN-CおよびDコンパイラで作成したオブジェクト・プログラムと比較したのが，表8である。試作コンパイラは，JIS 3000 水準 FORTRAN 用コンパイラであるのに対し，FACOM 230-60

表 8 コンパイルされたオブジェクトプログラムの大きさの比較

テストプログラム		試作コンパイル	FORTRAN Cコンパイラ	FORTRAN Dコンパイラ	ソースステートメント
No. 1	オブジェクトプログラムの大きさ	59ワード	62ワード	58ワード	10
	コンパイル時間 (m sec)	822	182	209	
	コンパイル時間比	4.52	1	1.15	
No. 2	オブジェクトプログラムの大きさ	114ワード	126ワード	124ワード	22
	コンパイル時間 (m sec)	1405	289	358	
	コンパイル時間比	4.87	1	1.24	
No. 3	オブジェクトプログラムの大きさ	907ワード	924ワード	854ワード	21
	コンパイル時間 (m sec)	2167	342	560	
	コンパイル時間比	6.34	1	1.64	
	実行時間 (m sec)	769	769	669	

FORTRAN-C および D コンパイラは JIS 7000 水準を越える大きなコンパイラであるため，一概に比較はできないが，配列などを使用しない単純なプログラム（テスト・プログラム No. 1, No. 2）に対しては，試作コンパイラは C コンパイラと同程度のオブジェクト・コードを生成できる。しかし，テスト・プログラム No. 3 のように配列を使用したプログラムでは，D コンパイラに比較して，試作コンパイラが生成したオブジェクト・コードはかなり冗長である。

また，試作コンパイラのコンパイルスピードは，平均 C コンパイラの 5.24 分の 1，D コンパイラの 3.91 分の 1 程度とたいへん遅くなっているが，この原因はつぎのように考えられる。

(1) コンパイラの手続き部のうち，約 82% を FORTRAN で記述しているので，できあがったコンパイラのオブジェクト・コードの能率が悪い。

(2) FORTRAN, FASP のプログラム間の連絡に時間を要する。

(3) 表処理に高度な手法を使用していない。

(1) を解決するには，FORTRAN コンパイラがさらに徹底したオブジェクト・コードの最適化を行うことが必要である。

(2) を解決するには，FORTRAN プログラムの中に，アセンブリ言語を直接記述できるようにしたり，先に提案した機能を言語仕様に組み込むことが必要である。

(3) の手法を行うには，試作コンパイラのアルゴリズムの一部を変更するだけでよい。

5. 結　　言

本研究では，コンパイラを FORTRAN 主体に記述することにより，つぎのような利点を実証することができた。

(1) FORTRAN によってコンパイラを作成すると，従来から行われてきたアセンブリ言語による方法に比較して生産性が 3.15～3.18 倍程度上昇する。

(2) デバッグ過程で，ステップ当たりのエラー発生率は FORTRAN でもアセンブリ言語でもほぼ同程度であることが確認された。そのためステップ当たりの生産性の高い高級言語のほうが，アセンブリ言語よりエラー発生頻度が相対的に減少する。

一方，問題点としては，

(1) 主記憶占有量は 1.8～2.14 倍になる。

(2) FORTRAN, FASP 間のリンクは，すべてクローズド・サブルーチン形式になっているため，コンパイルスピード低下の一因になっている。

などがあるが，(1) の問題に関しては，今後プログラムの生産性に重点が置かれるようになり，ハードウェアのコストパーフォマンスの上昇も手伝って，主記憶占有量は，現在ほど重視されなくなると思われるが，無視することのできない問題である。(2) の問題を解決するには，FORTRAN のような高級言語の中に，アセンブリ言語を直接記述できるような機能を持たせたり，あるいは **4. 3** で提案した機能を言語仕様にもつシステム記述言語を開発することにより，コンパイルスピードを向上させることができる。

以上のように FORTRAN をシステム記述用として使用してみた結果について批判，提案を行い，コンパイラ記述上参考となるデータを得ることができた。これらの提案は，FORTRAN に対する機能の追加という範囲にとどまっているが，実際に作成したプログラムによる定量的な判断であるから，今まで直感的にそうではないかと考えられていたことが，本研究で，ある程度実証できたと考えられる。コンパイラ以外のシステム記述には，もっと別な機能も必要と思われるので，今後さらにこれらのデータを積みあげることによって，システム記述言語の標準化や管

理方法が確立されていくことが重要である。

謝辞 本研究に際し，北大大型計算機センターの各位より有益な御協力をいただいたことを記して感謝の意を表する次第である。

参考文献

1) Weissmann, C.: 最近のソフトウェア技術の展望と将来，情報処理，Vol. 14, No. 10, pp. 739–745 (1973).
2) 美間敬之: コンピュータ大衆化のために，情報処理，Vol. 14, No. 12, pp. 915 (1973).
3) 小田一博: コンパイラ入門，日本評論社 (1971).
4) 田中 一，栃内香次，宮本衛市: コンパイラ，森北出版 (1971).
5) コンパイラ自動作成シンポジウム報告集，情報処理学会プログラミングシンポジウム委員会 (1969).
6) 井上謙蔵: システム記述言語 SWL の素顔，日経エレクトロニクス，Vol. 4, No. 26, pp. 50–56 (1971).
7) 井上謙蔵: コンパイラ・コンパイラ，産業図書 (1970).
8) 浜田，中田，霜田，小林: PL/I によるシステムの開発，情報処理学会プログラミングシンポジウム報告集 (1970).
9) 井上謙蔵，藤崎 湛: ハードウェアに独立なシステム・ライティング・ランゲージについて，情報処理学会プログラミングシンポジウム報告集 (1970).
10) 渡辺，高野，伊谷，長岡: ハードウェアディペンデントなシステム・ライティング・ランゲージについて——一つの試作，情報処理学会プログラミングシンポジウム報告集 (1970).
11) 萩原 宏，渡辺勝正: Compiler 記述言語 COL，情報処理，Vol. 9, No. 4 (1968).
12) 萩原 宏，渡辺勝正: COL で書かれた Compiler について，情報処理，Vol. 10, No. 6 (1969).
13) ESDL 特集号，電総研彙報，Vol. 34, No. 5–6 (1970).
14) 山田，栃内，仲丸: 構文解析プログラムの自動作成について，電気四学会北海道支部大会論文集 (1972).
15) 山田，栃内，永田，仲丸: コンパイラ作成簡単化の一方法，電気四学会北海道支部大会論文集 (1973).
16) Hopgood, F. R. A.: 首藤 勝，関本彰次訳，コンパイラの技法，サイエンス社 (1973).
17) 戸田 巌・他: PL/I コンパイラ構成法の研究，通研実用化報告，Vol. 18, No. 1 (1969).
18) Corbato, F. J.: PL/I as a tool for system programming Datamation, pp. 68–76 (May 1969).
19) Floyd, R. W.: Syntactic Analysis and Operator precedence, JACM, Vol. 10, No. 7, pp. 316–333 (1963).
20) Richards, M.: BCPL, A tool for compiler writing and system programming, Proc. of SJCC, Vol. 34, pp. 557–565 (1969).
21) Lang, C. A.: SAL-Systems Assembly Languages, Proc. of SJCC, Vol. 34, pp. 543–555 (1969).
22) Wirth, N.: PL 360, A Programming Language for the 360 computers, JACM, Vol. 15, No. 1, pp. 37–74 (1968).
23) Wolf, W. A. Russel, D. B., Habermann, A. N.: BLISS, A Language for System Programming, CACM, Vol. 14, No. 12, pp. 780–790 (1971).
24) Lowry, E. S., Medlock, C. M.: Object code optimization, CACM, Vol. 12, No. 1, p. 13 (1969).
25) Mauer, W. P.: An improved hash code for scatterstorage, CACM, Vol. 11, No. 1, p. 35 (1968).