

FLATS FORTRAN 77 の使用感と処理系

稲田 信幸 (東大・理・情報科学科)

1. はじめに

数式処理指向計算機 FLATS のソフトウェアの一環として開発された FORTRAN コンパイラについて報告する。数式処理において真に意味のある計算を行なおうとすると、数値計算向きの計算機に実現されたシステムではスピードの期待は望めなく、且つ新しい数式処理のためのアイデアもそのため故に実現できない。現在 FLATS のソフトウェア・シミュレータが作動しており、FLATS ハードウェアはこれの完全な上位互換を考慮して作成される。現在ソフトで実行しているために違い部分は、これから作成される FLATS ハードが実現の時には、コンパイル時・実行時のスピードの向上を望めるし、FLATS ハードウェアにより依存した命令を使用する、あるいは FORTRAN 処理系が必要と思う命令セットを FLATS が実装した時には、更にスピードが向上する。

ソフトウェア工学の見地から、これからのソフトウェアにはポータビリティが要求されると思う。幸いにして FLATS ソフトウェア・シミュレータは完全に FORTRAN 語で書かれ、FORTRAN コンパイラの記述に使用した HLISP コンパイラもシミュレータ上で動作するので、FORTRAN コンパイラのポータビリティが保証されている。HLISP family の HLISP インタプリタ上でも容易に作動するだろうし、一般の Lisp 上でも容易に作動するはずである。但しコンパイラは FLATS のコードを出力するので、容易には目的コードの実行はできない。

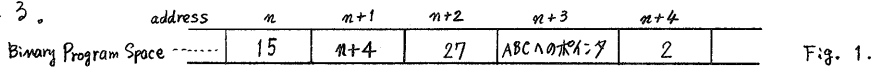
ソフトウェアの開発で第1に考えなければならないのは――まず正しく動作することを保証することである。スピード・アップ及びそのためのオプティマイズはそれから先のことである。

数式処理システムと FORTRAN 語は一見相反するかのように思われがちであるが、FLATS の目標は数式処理及び科学技術計算をまともなスピードで実現することである。これまでに FORTRAN で記述されたライブラリやアルゴリズムのプログラムが沢山蓄積されているのをそのまま使用する。計算機システムを手軽に使用するにも FORTRAN 語がよく使用されている。数式処理の最終結果を数値計算する。……などのために FORTRAN コンパイラはベシクなソフトウェアとして必要なものである。

2. FLATS ソフトウェア

ポータビリティを保証するために HLISP コンパイラは、中間言語 (仮想マシン≒FLATS) を driver ルーチンが解釈実行する形をとっている。詳しくは文献 [9] によってほしい。ここでは必要最小限の記述に留める。仮想マシンは HLISP が動作できるように 2 本のスタック (Value Stack & Control Stack) をもち、関数 (プログラムの手続き部分) の命令コードを格納するプログラム領域 (BPS), Lisp の cons で使用されるフリー・ストレージの L 領域、ハッシュされてシステム内ユニークな表現を可能とする H 領域といくつかのシステム・レジスタ群から構成されている。BPS は仮想化が行なわれ、一杯になるとすべての関数を追い出してしまう。BPS への関数の格納はローダと呼ばれる関数に

よ、て行なわれる。このローダは一種の簡易 one-pass アセンブラである。コード・リストの中の記号は命令を、負の整数はラベルを表わしているからで、ローダはこのラベルを絶対番地に変換している。例えば ((JUMP -3) (PLQ ABC) -3 RETURN) というコード・リストは BPS 上で 5 語占有し、図 1 のように格納される。



ここで 15, 27, 2 は各々 JUMP, PLQ, RETURN という FLATS 命令コードで、ラベルの -3 は絶対番地 n+4 である。アトム ABC は H 領域の中に存在し、それへのポインタが BPS の 1 語の中に納められる。

関数やサブルーチン類が CALL 命令で呼び出されると、frame pointer やスタックのポインタが変化する。例えば FORTRAN 文の CALL SUB (2, 10) は、((LQ 2) (PLQ 10) (CALLH SUB 2 0)) と翻訳され、これらのコードが実行されると V-stack は

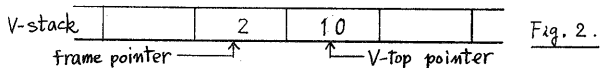
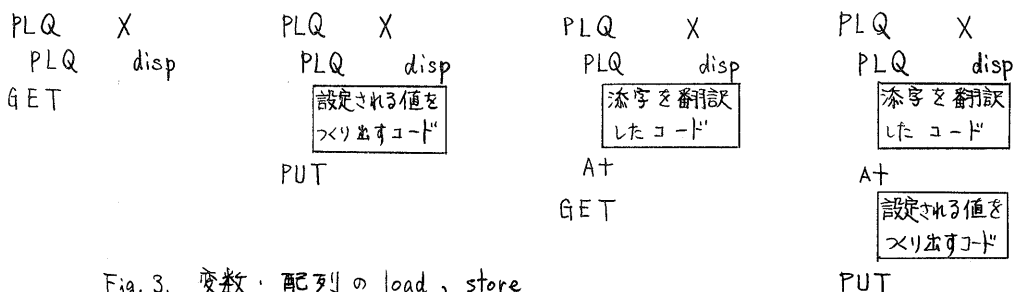


図 2 となっている。戻り番地は C-stack に積まれる。C-stack は特に意識しなくてもシステムが自動的に処理している。FORTRAN コンパイラでは局所的な変数、配列及びコモン変数、配列という静的割付け可能な領域は、HLISP という連想子 (associator) で表現している。将来配列を直接使用できる命令が FLATS に付加されたら、主記憶を直接参照する命令を使用するようにする。FORTRAN の関数・副プログラムとサブルーチン・副プログラムは同じようにコンパイルされて、システム内では副プログラムの名前によって動的なリンクが行なわれて、BPS 上に格納される。もちろん既に BPS に存在していれば、そのプログラムが使用される。RETURN 命令実行で呼び出し以前の状態に V-stack を戻し、C-stack の戻り番地に復帰する。変数や配列の値セルは、H 領域のどこに決定されるかは実行してみなければわからないし、データ領域が一定の連続した主記憶でもないのが初期設定をリンカーに行なわせるということとは現在のところできない。

将来 FLATS ではタイプに応じた演算が自動的に行なわれるが、HLISP の算術関数 A+, A-, A*, A/ は 2 倍長整数演算をサポートしている。それで整数型同士の演算はこれらの関数 (システム関数 = FLATS 命令) で行ない、他のデータ・タイプを含むものは関数プログラムで行なう。浮動小数点数も関数プログラムを使用して任意精度で行なわれる。

変数や配列要素からの値の取り出しには GET 命令、値の設定には PUT 命令を用いている。それ故に変数や配列要素のアドレス表現はデータ領域名 X とそのデータ領域の先頭からの相対位置 disp (offset) で示される。



3. FORTRAN 77 文法

ANSIが1966年にFORTRAN語を制定して以来、改訂のための討議が続けられて、今回実に10年振りに改訂案が1976年3月にACMのSIGPLAN Notices に発表された。その後も積極的に各国の学会に意見を求めており、改訂の中は大きくないにしろ、必要と思われる文法は取り入れられてきた。筆者は最新の情報を入手している訳ではないが、最近の国内・アメリカの研究誌・雑誌にも旧規格との比較、新規格の紹介、新規格への注文などについての解説記事が発表されている。

IBMによ、てFORTRAN語が生み出されて以来、計算機システムとともに数多くのFORTRANコンパイラが書かれ、それらコンパイラの言語仕様はANSI又はJISといった規格に準拠している場合が殆んどである。だが、最近の大型機等のコンパイラに見い出されるように拡張された文法が今回の規格で取り入れられたものが多いし、PL/I, PASCAL等の言語の影響、構造化プログラミングの影響を少なからず受けている。

FORTRAN語は計算機を意識して生み出された言語なので新規格になっても *data abstraction* は存在していないし、高級アセンブラの域を出ていない感が強い。旧規格で曖昧であった文字型は、新規格では新しくデータ・タイプの一種として扱うようになったのは評価できる。しかし、文字型データ領域と数値型データ領域とは共有できないと文法ではなっているが、実行時にチェックできる機能をもたない場合には完全に実現できない。

FORTRAN語の場合、静的にはデータ・タイプのチェックは可能であるが、サブルーチン等の壁を越えるとその先は何もチェックができなくなっている。しかし、これからのプログラム作成技術としてモジュラー構成をとる場合が多いので、実行時に何らかの処理をする必要が出てくるはずである。

FLATSでは数(整数型、実数型)は数学で表現される数を考えている。しかし、商用マシンがByte, Word 演算という昔からの流儀から抜け出ないために、実数型には単精度と倍精度が存在している。

今回の改訂の目的は、

- 1) ----- 翻訳処理系を容易に実現できること、
- 2) ----- 効率の良い目的コードを生成しうること、
- 3) ----- 旧規格を包含すること

となっている。1)ではSAVE文, *implied do* の処理など多少難しくなっている。

2)では、例えばDO文の制御変数及び式に整数型や実数型が許されているので、ループは場合によ、ては効率のよいコード生成は難しくな、てきている。3)では大筋のところ変更なしで働くらう。新規格への要望は、

- 1) ----- WHILE, REPEAT, CASE等の文が使えること、
- 2) ----- 複合文やブロック構造を有すること、
- 3) ----- 自由書式で入力でき、コメントも各所に挿入できること、
- 4) ----- ビット・ストリングが扱えること、
- 5) ----- COBOL, PL/Iにある構造体のデータ構造を扱えること、
- 6) ----- タイプを拡張できること、
- 7) ----- ポインタを扱えること、
- 8) ----- マクロの機能を有すること

などが挙げられる。

4. コンパイラ

FÖRTRAN コンパイラは完全にHLISPで記述され、4700枚程になっている。処理系は容易に拡張・変更ができる様に作成されている。現在までのところ入出力等いくつかの文は、まだコーディングされていない。また目的コード出力に際して特に最適化は行っていない。FLATSのすべての言語プロセッサの出力するコードを最適化するオブティマイザは将来FLATSマシンのために作成される。FLATSの命令が流動的な部分を含んでいるので、今回の版では純スタック命令のみを扱っている。

データ内部表現とシラブル・リーダー

言語の構文解析を行なう際の分解しうる最小の意味のある単位 (syllable or token) は英字名、定数及びデリミタに分類できる。英字名は何文字で構成されても構わない。定数には整定数、実定数、複素定数、論理定数及び文字定数がある。整定数、実定数とも任意桁許されるので、

164263371599802709531, .1E-50000, 634721621.41937.72825414E1615951002

などという数も扱える。文字定数の中にホレリス定数も許している。短整数はポインタそのものがタグと整数値 (絶対値が 10^3 より小さい数) を表わしているので直接命令のオペランドに埋め込まれる。

任意の倍長整数 (big num.) の場合には図4のようなリストがH領域上に作成される。A+やA*というFLATS命令を使用する限り、短整数、任意の倍長整数の扱いはシステムが自動的に管理する。

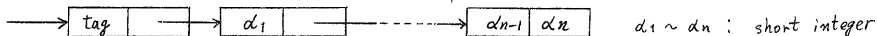


Fig. 4.

浮動小数実数は図5のようなリストがH領域上に作成される。m及びeは整数表現されており、任意桁許されている。後述のシラブル・リーダーでは将来のため単精度型と倍精度型とでは別々の情報を出力しているが、データ表現RESULTの中は同一である。

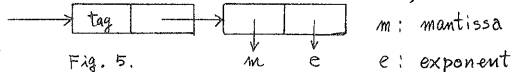


Fig. 5.

複素定数は図6のようなリストがH領域上に作成される。r及びiは実数定数へのポインタである。

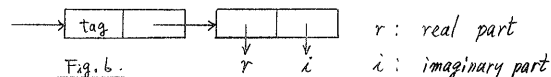


Fig. 6.

論理定数.TRUE.はアトムTへのポインタ、.FALSE.はアトムNILへのポインタで、システムではNIL以外はすべて真値扱いする。

文字定数は図7のように表現されている。変数の持つ値と文字定数を同一の内部表現にすると文字位置 (バイト・アドレス) でEQUIVALENCEできなくなってしまう。文字型変数CH, CHAを各々長さ6, 4とする時に



Fig. 7.

EQUIVALENCE (CH (3:6), CHA)

と部分的結合が行なわれると、文字変数と文字格納領域は図8のようになる。

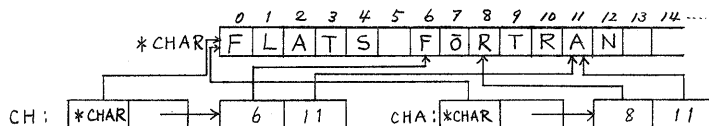


Fig. 8.

*CHARは文字格納領域名、CH, CHAは変数の値セルの内容を表わす。

デリミタは1~2文字の特殊文字より構成される。構文解析の都合上、E.Q.の類もデリミタに入れる。入力テキストより1つシラブルを取り出してくる関数はGETNEXTTOKENで表1のような結果を返す。

	kind of a syllable	returned value	value of variable RESULT
1	identifier	0 (zero)	pointer to a literal atom
2	integer number	& INTEGER	pointer to a numerical atom
3	real number	& REAL	
4	real num. with Dep.	& DOUBLE PRECISION	
5	complex number	& COMPLEX	
6	logical constant	& LOGICAL	atom T or NIL
7	character constant	& CHARACTER	pointer to a string atom
8	delimiter	internal code for delimiter character	NIL

Tab. 1.

表操作及びリスト

コンパイラは英字名やラベルに関する情報を表の形で登録しておき、参照される毎に表を調べて処理をしたり、誤りを指摘する。HLISPでは特別な表のために主記憶を使用できないし、HLISPで用意しているL領域とH領域以外使用できない。Lispでは表の代行をproperty-listで行なう。特にHLISPではproperty-listはハッシュ化されているので、1つのアトムに沢山の属性をつけても探し出す時間は要素の数とは独立になるので高速にアクセスできる。H領域の大きさはハッシュの時間と登録できる情報量を決定する。ラベルと英字名につけられる属性は次のような目的に使用されている。

- LABEL-----ラベルの値を記憶する。
- LABELUSE, LABELDEF-----ラベルの定義と参照において有効性のチェック。
- CLASS-----英字名の属類(変数, 配列, サブルーチン, 関数, 記号定数, 文関数, 主プログラム, ブロック・データ, 組込関数他)。
- TYPE-----英字名のデータ・タイプ(整数, 単精度実数他)。
- LENGTH-----文字型の時の長さの属性。引渡し文字長の場合は0。
- LOC-----あるデータ領域からの相対位置, 記号定数の場合は設定された値。
- DIM-----整合寸法の配列でない時, 上下限値のdotted pairがリストされる。
- ADJUSTABLE-----整合寸法の配列の時, 副プログラム入口で評価するための上下限のリスト。
- INCOMMON-----コモン並びの時, コモン・ブロック名が入る。
- INEQUIVALENCE-----EQUIVALENCEチェーンに使用。
- OFFSET-----EQUIVALENCE処理の時, 相対位置を格納。
- FORMAL-----仮引数並びの時, 副プログラム名が入る。
- ARGUMENT-----副プログラム, 組込み関数, 文関数の時, 引数の個数。
- BASE-----引数配列の時, データ領域名を格納しておくV-stackの相対位置。
- DISP-----引数配列の時, 相対位置を格納しておくV-stackの相対位置。
- STT-----引数が部分文字列の場合, 左側位置を格納しておくV-stackの相対位置。
- STP-----右側位置を格納しておくV-stackの相対位置。

IDENTLISTにはプログラム単位内で使用された英字名がリストされ, LABELLISTには使用されたラベルがリストされて登録されている。マップの出力, 宣言の処理や未定義ラベルの検出等のために使用される。

実引数の内部表現

FORTRANで扱かう実引数には単純変数、配列要素名、配列名、仮引数、式、組込み関数名、及び外部手続き名があり、サブルーチンの場合にはラベルもネをつけて書くことができる。プログラム・スペース内のアドレスを直接扱えないので、alternate returnについてはソフト的に解決して実引数には積まないことにしている。

- 1) 定数のみ又は式の場合は、V-stack 上に値を積む。
- 2) 変数名と配列名は、データ領域名と相対位置を hconsしたものをアドレス表現と称して V-stack に積む。
- 3) 配列要素名は、データ領域名と、相対位置に添字式の値を加えたものを h-consしてアドレス化し V-stack に積む。
- 4) 仮引数の場合には受け取った V-stack の内容をそのまま積む。
- 5) 組込み関数名はその名前に #FRT を連結して（実際のシステム内の関数名）、(FUNCTION 関数名) を実行したものを V-stack に積む。
- 6) 外部手続き名は即 (FUNCTION 関数名) を実行したものを V-stack に積む。
- 7) 文字型については更に左側文字位置と右側文字位置の情報をも引き渡す。

FORTRAN コンパイラが稼動する FLATS ソフトウェア上では、現在のところ間接ポインタが使えないので仮引数をアクセスする時には、値であるかアドレス表現であるか実行時にチェックしながら使用しなければならない。この点についてもハードウェア実現の晩には、解決されて高速化される。パラメータの引き渡しを合理的で効率のよいものとするハードウェアが要求される。

宣言の処理

one-pass 方式のコンパイラで且つ宣言文の出現の順序にも制限をつけないので、宣言文が出現した時点では必ずしも処理はすべて完了しない。PARAMETER 文のようにその文だけで完了するものもあれば、EQUIVALENCE 文のようにリスト表現に変換して保存しておくだけのものもある。実際の処理はすべての宣言文が出現し終った時に処理する。以下の 6 ステップで行なう。

- step 1. 第 2 回目呼出し以降ならば、SAVE 領域よりロードしてくる命令出力。
- step 2. 型や長さの属性のない変数、配列、関数名に暗黙の型と長さを割り当てる。仮引数ならば UNKNOWN, それ以外ならば変数という CLASS を割りつける。
- step 3. EQUIVALENCE チェーンの作成、リスト EQUIVLIST に全 EQUIVALENCE の情報が保存されている。
- step 4. COMMON 要素の相対位置決定。データ領域名はコモン・ブロック名である。EQUIVALENCE チェーンにある要素も全て相対位置が求まり、コモン・ブロックのサイズも決まる。
- step 5. 整合寸法以外の未処理の変数及び配列のデータ領域、相対位置が求まる。
- step 6. 整合寸法、仮引数配列の実行時評価の命令コード出力。

以上の中には文字型の変数の実行時の命令コードも出力される。宣言文の中で、IMPLICIT, PARAMETER, INTRINSIC, 及び EXTERNAL の各文はその文の処理ですべて処理が終わる。他の宣言文は構文のチェックを行ないながら属性を与えてゆき、上記の 6 つのステップであるデータ領域の先頭からの相対位置が求まる。

DO文の処理

DO文の構文は

DO S [,] i = e₁, e₂ [, e₃]

の形をしており、ラベルSの後に、(カンマ)が許される。制御変数iと式e_nは複素数型を除く算術型が許される。コンパイラが生成する目的コードは図9のようになる。式e_nと変数iの型が異なる時、変数iの型に型変換する命令が式e_nの翻訳の後ろに付加される。増分値e₃とループ・カウンタはV-stack上の作業領域上に確保される。

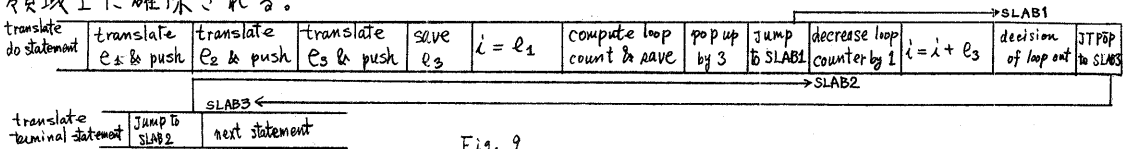


Fig. 9.

e_nが整数型の単純な式の時には明らかにループの回数がわかるので、最適化したコードを生成すべきであるが、式の中に関数呼出や配列参照がある場合には図9のようなコードを生成しなければならない。規格ではループの回数を次式

$$\max(\text{int}((m_2 - m_1 + m_3) / m_3), 0)$$

で求めなければならない。旧規格と明らかに異なるのは、ループ・カウンタの値が0の時は、DOの範囲を1回も実行しないことである。端末文処理に必要な情報はラベルS, SLAB2, SLAB3であるのでDOの入れ子処理、DOの範囲からくり返しに戻る命令作成のために図10のようなDO LIFOプッシュ・ダウン・リストを使用する。DO LIFO

このDO LIFOの形式は

拡張DO文のDO WHILE,

DO UNTIL型にも変更

なしで使用できる。ブロックIF文との入れ子関係をチェックするためにはCURRENT-IFとIF-LEVELの連想子にSを付加してチェックしている。

block IF文の処理

新FORTRANが構造化プログラミングを取り入れた唯一の証明となる文の集まりである。図11のように4つの文からなり、block-nに更にblock-IF文を含んでも構わないし、DOループを含んでもよい。blockの中で使用されたDO文の端末文も同一blockの中になければならない。ブロックの管理を行なうために、変数CURRENT-IFとIF-LEVELを導入する。これはJNIL命令のラベルを保持する。このblockの中でDO文が現れる時にはCURRENT-IFとIF-LEVELの連想子にSを付加してゆく。ELSE IF, ELSE, あるいはEND-IFが出現した時に連想子の値がアトムでなければ、閉じていないDOループがあるのでエラーとなる。block-nが実行し終った時にはEND-IFに行かなければならないので、END-STKにIF-LEVELに応じた出口のラベルがリストされている。

```
IF (logical expression) THEN
    block - 1
ELSE IF (logical expression) THEN
    block - 2
ELSE IF ----
    ...
ELSE
    block - n
END IF
```

Fig. 11.

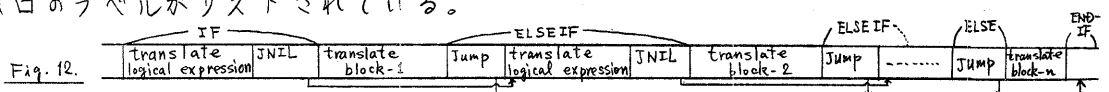


Fig. 12.

5. 使用してみたら

翻訳されたプログラムはプログラム単位名で直接実行させることができる。HLISPコンパイラ出力のコードと同じなので相互に呼出すこともできる。稼働中のHLISPコンパイラ版はL, H, BPSともにあまり大きくないシステムなので、1~3枚のカードを処理する毎にBPS領域からすべての関数を追い出している。翻訳時間、実行時間の両方ともメーカ提供のFORTRANシステムに比べて100倍以上時間がかかっている。それで大きなプログラムを翻訳するのは無理で付録のような例題だけを実行させている。時間が非常にかかるのはH領域が減少しハッシングに時間がかかるのと、配列というデータ構造を直接扱えないのと、中間語をインタプリートしているのと、実行時にすべてのデータ・タイプをチェックしているなどの理由による。コンパイラ及び目的プログラムの実行には、かなりの回数のハッシングが行なわれるので、ハッシング・ハードウェアが実装されただけでもかなりのスピード向上が図れるはずである。

PARAMETER文及びIMPLICIT文、実行文ではblock IF文がかなり有用な文であることが使用経験から得られた。旧規格のプログラムで文字型のものは全然動かなくなってしまう。本コンパイラの場合データ・タイプのチェックが実行時に行なわれるので中途半端なプログラムは動かなくなってしまう。FORTRAN 77の文法違反ではあるが、Recursiveなプログラムを書くことができる。これができ便利である。

文字型、大指数浮動小数実用の基本演算ルーチンは、HLISPインタプリタ上で動作することを確認している。文字型の方はcharacter spaceの表現法で実現方法を検討している。後者はA/の多倍長整数サポート待ちである。

6. おわりに

FORTRANコンパイラを作成中にHLISPのシステム・バグがいくつか発見された。それでコンパイラの原因なのかシステムの原因なのか迷路に入ってしまうので、システム作りに使用する道具には完璧さが要求されると思う。HLISPインタプリタ版は全国の大学・研究所でよく稼働しており殆んど虫のいないシステムと称えられる。HLISPコンパイラ版もよく使用されることが大切であると思う。同様のことがこのFORTRANコンパイラについても言えるのでシステムを完全なものにするためにも、とんとん使用されることが望まれる。このFORTRANコンパイラがFLATSハードウェア上で動作する日を夢みながら、現在更にデバッグ、未コーディングの文のコーディングを行っている次第です。FLATSソフトウェア上のスピードとハードウェア上のスピードの違いや効率の違いなどについては別の機会に発表したいと思っている。

謝 辞

本研究は筆者が東大・理・情報科学科・後藤英一教授のもとに情報処理関係内地研員として迎えられて行なわれたもので、有益なご助言、議論をして下さいました後藤教授に感謝いたします。HLISPシステムをはじめとして数々のご協力をして頂いた金田康正氏、舟島元章氏に感謝いたします。公私にわたりご援助、励まして下さいました戸嶋 照氏に感謝いたします。そして理研では筆者の研究を励まし、良き討論者となって頂きました相馬嵩氏、お沢正徳氏、佐々木建昭氏、及び井田哲雄氏に感謝いたします。

参考文献

- [1] Fortran 77 Full Language Working Document, X3J3/76.5, (76-11-30).
- [2] J. D. Woolley, A Comparison of the New Proposed Language (1976) to the Old Standard (1966), Vol. 12, No. 7, ACM SIGPLAN Notices, (July 1977).
- [3] A. H. J. Sale, The Classification of Fortran Statements, Vol. 14, No. 1, Computer Journal, (1976).
- [4] D. Gries, "Compiler Construction for Digital Computers", John Wiley & Sons, (1971).
- [5] F. Motoyoshi, "A Portable Lisp Compiler on a Hypothetical Lisp Machine", Technical Report, (76-05), Univ. of Tokyo, (1976).
- [6] E. Goto, "Monocopy and Associative Algorithms in an Extended Lisp", Technical Report, (75-03), Univ. of Tokyo, (1975).
- [7] L. P. Meissner, Fortran 77, Vol. 12, No. 1, ACM SIGPLAN Notices, (January 1977).
- [8] D. Salomon, A Design for Fortran to Facilitate Structured Programming, Vol. 12, No. 1, ACM SIGPLAN Notices, (Jan. 1977).
- [9] T. Terashima, A Note on an HLISP Compiler, 記号処理研究会報告 昭和51年度 研究報告集.

付 録 実 行 例

```

**      HLISP COMPILER VERSION (77/09/15)      **

1      INTEGER FUNCTION FACJAC(N)
2      IF (N.EQ.0) THEN
3          FACJAC=1
4      ELSE
5          FACJAC=N*FACJAC(N-1)
6      END IF
7      END

```

```
HLISP COMPILER VERSION (77/09/15) **
```

```

1      FUNCTION JACJAC(N)
2      JJ=N
3      MM=FACJAC(JJ)
4      N=1
5      DO 1 N=JJ-1, 1, -1
6      1  JJ=JJ*N
7      JACJAC=JJ
8      CALL OUTOUT(MM-JACJAC)
9      END

```

```

**      HLISP COMPILER VERSION (77/09/15)      **

```

• FORTRAN COMPILER END

```
*****
* INPUT TIME                               5 M SEC. *
* EVALUATING TIME                         16491 M SEC. *
* OUTPUT TIME                             1 M SEC. *
* NHO-CELL                               2752 / 6607 CELLS *
* GBC TIME                               3006 M SEC. *
*****
```

P O

P 8272237582511852109168640000000000000000000000000

```
... STOP STATEMENT HAS EXECUTED WITH AN ARGUMENT:  NIL
```

• NIL

```

*****
* INPUT TIME                               11 M SEC. *
* EVALUATING TIME                         520 M SEC. *
* OUTPUT TIME                             1 M SEC. *
* NHO-CELL                               1932 / 6607 CELLS
* G6C TIME                               182 M SEC. *
*****

```

```
HLISP COMPILER VERSION (77/09/15) **
```

```

C      SEEK PRIME NUMBERS
1      PROGRAM PRIME
2      INTEGER ONE,TWO,THREE
3      PARAMETER (ONE=1,TWO=2,THREE=3,L=300)
4      DIMENSION M(L)
5      M(ONE)=TWO
6      M(TWO)=THREE
7      J=THREE
8      N=THREE
9      20 N=N+TWO
10     K=TWO
11     KK=M(K)
12     IF ( N=N/KK*KK ) 2,20,2
13     2 IF ( KK*KK-N ) 3,4,4
14     3 K=K+ONE
15     GO TO 10
16     4 M(J)=N
17     J=J+ONE
18     IF ( J-L ) 1,20,100
19     100 DO 5 I=ONE,L
20     5 CALL OUTPUT( ( M(I) ) , 5)
21     END

```

```

** HLISP COMPILER VERSION (77/09/15) **

```

```

1      PROGRAM MAIN
2      DIMENSION IU(100)
3      EQUIVALENCE (IU(100),I1)
4      DO 4 I=1,100
5          4 IU(I) = I
6      CALL OUTOUT (JACJAC(I1))
7      END

```

```
*****
* INPUT TIME                      6 M SEC. *
* EVALUATING TIME                6481 M SEC. *
* OUTPUT TIME                    1 M SEC. *
* NHO-CELL                      1351 / 6607 CELLS
* GBC TIME                      0 M SEC. *
*****
```

	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67	71	73
179	73	79	83	89	97	101	103	107	109	113	127	131	137	139	149	151	157	163	167	177	181
283	181	191	193	197	199	211	223	227	229	233	239	241	251	257	263	269	271	277	281		
419	293	307	311	313	317	331	337	347	349	353	359	367	373	379	383	389	397	401	409		
547	419	421	431	433	439	443	449	457	461	463	467	479	487	491	499	503	509	521	523	541	
661	547	557	563	569	571	577	587	593	599	601	607	613	617	619	631	641	643	647	653	659	
811	661	673	677	683	691	701	709	719	727	733	739	743	751	757	761	769	773	787	797	809	
947	811	821	823	827	829	839	853	857	859	863	877	881	883	887	907	911	919	929	937	941	
1087	947	953	967	971	977	983	991	997	1009	1013	1017	1021	1031	1033	1039	1049	1051	1061	1063	1069	
1229	1087	1091	1093	1097	1103	1109	1117	1123	1129	1151	1153	1163	1171	1181	1187	1193	1201	1213	1217	1223	
1381	1229	1231	1237	1249	1259	1277	1279	1283	1289	1291	1297	1301	1303	1307	1319	1321	1327	1361	1367	1373	
1511	1381	1399	1409	1423	1427	1429	1433	1439	1447	1451	1453	1459	1471	1481	1483	1487	1489	1493	1499	1511	
1657	1511	1523	1531	1543	1549	1553	1559	1567	1571	1579	1583	1597	1601	1607	1609	1613	1619	1621	1627	1637	1657
1811	1657	1663	1667	1669	1693	1697	1699	1709	1721	1723	1733	1741	1747	1753	1759	1777	1783	1787	1789	1801	1811
1987	1811	1823	1837	1847	1861	1867	1871	1873	1877	1879	1889	1901	1907	1913	1931	1933	1949	1951	1973	1979	1987

```

1025 1031 1047 1051 1067 1071 1075 1077 1079 1083 1085 1087
... STOP STATEMENT HAS EXECUTED WITH AN ARGUMENT : "NIL"

```

• NIL

```

CLASS OF THE PROGRAM UNIT : MAIN
PROGRAM NAME : MAIN#FRT
THE NUMBER OF THE GENERATED OBJECT CODES : 563
USED VALUE STACKS : 7
USED STATIC AREAS : 72
USED CHARACTER AREAS : 0
THE NUMBER OF THE COMPILED STATEMENTS : 25
COMMON BLOCK NAME : SYSTEM
ITS ENTITIES LIST : (C1 C123 C3 C4 C45 )
THE SIZE OF THE COMMON BLOCK : 171
*10000:=8(DEF=6EXST,USED=6EXST) *1:=15(DEF=6EXST,USED=6EXST) *10001:=10(DEF=6EXST,USED=6EXST)
*2:=14(DEF=6EXST,USED=6EXST) *10002:=11(DEF=6EXST,USED=6EXST) *3:=12(DEF=6EXST,USED=6EXST)
*9:=13(DEF=6EXST,USED=6EXST) *10003:=16(DEF=6EXST,USED=6EXST) *10006:=17(DEF=6EXST,USED=6EXST)
*5:=20(DEF=6EXST,USED=6EXST) *10007:=18(DEF=6EXST,USED=6EXST) *10008:=19(DEF=6EXST,USED=6EXST)
*10010:=21(DEF=6EXST,USED=6EXST) *10011:=24(DEF=6EXST,USED=6EXST)
UHK CLASS:=6ARRAY TYPE:=6INTEGER LENGTH:=NIL LOCATION:=0 DIMENSION:=(1, 20) ) ADJUSTABLE:=NIL
INCOMMON:=NIL INEQUIVALENCE:=NIL OFFSET:=17 FORMAL:=NIL ARGUMENT:=NIL
TTY CLASS:=6ARRAY TYPE:=6INTEGER LENGTH:=NIL LOCATION:=20 DIMENSION:=(1, 20) (44, 45) ) ADJUSTABLE:=
* NPI INCOMMON:=NIL INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* NPI CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=17 DIMENSION:=NIL ADJUSTABLE:=NIL INCOMMON:=N
* IL INEQUIVALENCE:=UHK OFFSET:=0 FORMAL:=NIL ARGUMENT:=NIL
* KKLKL CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=60 DIMENSION:=NIL ADJUSTABLE:=NIL
INCOMMON:=NIL INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
K12345 CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=61 DIMENSION:=NIL ADJUSTABLE:=NIL
INCOMMON:=NIL INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
C123 CLASS:=6ARRAY TYPE:=6REAL LENGTH:=NIL LOCATION:=1 DIMENSION:=(1, 123) ) ADJUSTABLE:=NIL
INCOMMON:=SYSTEM INEQUIVALENCE:=IPH OFFSET:=119 FORMAL:=NIL ARGUMENT:=NIL
JYY CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=120 DIMENSION:=NIL ADJUSTABLE:=NIL
INCOMMON:=NIL INEQUIVALENCE:=C123 OFFSET:=0 FORMAL:=NIL ARGUMENT:=NIL
C1 CLASS:=6VARIABLE TYPE:=6REAL LENGTH:=NIL LOCATION:=0 DIMENSION:=NIL ADJUSTABLE:=NIL INCOMMON:=SYSTEM
* INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* C3 CLASS:=6VARIABLE TYPE:=6REAL LENGTH:=NIL LOCATION:=124 DIMENSION:=NIL ADJUSTABLE:=NIL INCOMMON:=SYST
* EM INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* C4 CLASS:=6VARIABLE TYPE:=6REAL LENGTH:=NIL LOCATION:=125 D
* EM INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* C5 CLASS:=6ARRAY TYPE:=6REAL LENGTH:=NIL LOCATION:=126 DIM
INCOMMON:=SYSTEM INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL
IPH CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=50
* IL INEQUIVALENCE:=JYY OFFSET:=70 FORMAL:=NIL ARGUMENT:=NIL
* K CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=62
* L INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* L CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=63
* IL INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* I CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=64
* L INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* I CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=65
* J INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* J CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=66
* IL INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* A CLASS:=6VARIABLE TYPE:=6REAL LENGTH:=NIL LOCATION:=67 D
* A INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* B CLASS:=6VARIABLE TYPE:=6REAL LENGTH:=NIL LOCATION:=68 D
* B INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* J CLASS:=6VARIABLE TYPE:=6INTEGER LENGTH:=NIL LOCATION:=69
* L INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
* C CLASS:=6VARIABLE TYPE:=6REAL LENGTH:=NIL LOCATION:=70 D
INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
SS CLASS:=6VARIABLE TYPE:=6REAL LENGTH:=NIL LOCATION:=71 D
INEQUIVALENCE:=NIL OFFSET:=NIL FORMAL:=NIL ARGUMENT:=NIL
OBJECT CODES:=(PROG#FRT MAIN#FRT 1 NIL NIL) (PLW NIL) (PUSH 6) -8 (LQ STATIC+MAIN#FRT) (PLW 62
) GET (PLW STATIC+MAIN#FRT) (PLW 63) GET EQ (JNIL -100002) (LQ 123) (PLW 0) (PLW 11) MINUS (STB
1) (PLW STATIC+MAIN#FRT) (PLW 64) (PLT 5) PUT (LT 2) (PLT 4) A- (PLT 2) A+ (PLT 2) A/ (STB 2
) (POP 3) (JUMP -100003) -100004 (LB 2) SUB1 (STB 2) (LQ STATIC+MAIN#FRT) (PLW 64) (PLT 2) (PLT
2) GET (PLB 1) A+ PUT -100003 (LB 2) (PLT 1) ZEROP (JTPOP -100005) MINUSP (JT -100005) -10
STATIC+MAIN#FRT) (PLW 65) GET (PLQ 1) A+ (PLQ STATIC+MAIN#FRT) (PLW 65) GET (PLQ 2) A/ (STB 3)
) (PLW 65) GET A* (PLQ STATIC+MAIN#FRT) (PLW 65) GET (PLQ 2) A/ (STB 3) (PLW 65) GET (PLQ 3)
) (PLW 65) (PLT 5) PUT (LT 2) (PLT 4) A- (PLT 2) A+ (PLT 2) A/ (STB 4) (POP 3) (JUMP -100006)
-100007 (LB 4) SUB1 (STB 4) (LQ STATIC+MAIN#FRT) (PLW 65) (PLT 2) (PLT 2) (PLW 5) A+ PUT
-100006 (LB 4) (PLT 1) ZEROP (JTPOP -100008) MINUSP (JT -100008) -11 (PLT 2) (PLW 5) (PLW 4)
) (STB 5) (PLQ STATIC+MAIN#FRT) (PLW 66) (PLT 5) PUT (LT 2) (PLT 4) A- (PLT 2) A+ (PLT 2) A/ (STB
6) (POP 3) (JUMP -100009) -100010 (LB 6) SUB1 (STB 6) (LQ STATIC+MAIN#FRT) (PLW 66) (PLT 2)
) (PLT 2) GET (PLB 5) A+ PUT -100009 (LB 6) (PLT 1) ZEROP (JTPOP -100011) MINUSP (JT -100011) -12
(PLQ STATIC+MAIN#FRT) (PLW 67) (PLQ STATIC+MAIN#FRT) (PLW 67) GET (PLQ STATIC+MAIN#FRT) (PLW 68
) GET (CALLH ADD#FRT 0) PUT (JUMP -100010) -100011 -13 (LQ STATIC+MAIN#FRT) (PLW 69) (PLT 5) PUT (LT 2) (PLT
1) A- (PLW 2358) (PLW 11) MINUS (STB 5) (PLQ STATIC+MAIN#FRT) (PLW 69) (PLT 5) PUT (LT 2) (PLT
4) A- (PLT 2) A+ (PLT 2) A/ (STB 6) (PLW 6) SUB1 (STB 6) (LQ STATIC+MAIN
* #FRT) (PLW 69) (PLT 2) (PLT 2) GET (PLB 5) A+ PUT -100012 (LB 6) (PLT 1) ZEROP (JTPOP -100014
) MINUSP (JT -100014) -14 (PLQ STATIC+MAIN#FRT) (PLW 70) (PLQ STATIC+MAIN#FRT) (PLW 67) GET (PLQ
STATIC+MAIN#FRT) (PLW 68) GET (CALLH ADD#FRT 0) (PLQ STATIC+MAIN#FRT) (PLW 70) GET (CALLH SUB#FRT
2 0) PUT (JUMP -100013) -100014 (JUMP -100007) -100008 -15 (LQ STATIC+MAIN#FRT) (PLW 71) (PLQ STATIC+MAIN
* #FRT) (PLW 71) GET (PLQ 1) (CALLH ADD#FRT 0) PUT (JUMP -100004) -100005 -16 (JUMP -100001) -100002
-17 (LQ 81) (PLQ 200) (STB 1) (PLQ STATIC+MAIN#FRT) (PLW 65) (PLT 5) PUT (LT 2) (PLT
4) A- (PLT 2) A+ (PLT 2) A/ (STB 2) (POP 3) (JUMP -100015) -100016 (LB 2) SUB1 (STB 2) (LQ STATIC+MAIN
* #FRT) (PLW 65) (PLT 2) (PLT 2) GET (PLB 5) A+ PUT -100015 (LB 2) (PLT 1) ZEROP (JTPOP -100017
) MINUSP (JT -100017) -18 (PLQ STATIC+MAIN#FRT) (PLW 65) (PLQ STATIC+MAIN#FRT) (PLW 65) GET (PLQ
1) A- PUT -19 (LQ STATIC+MAIN#FRT) (PLW 66) (PLW 5) PUT -20 (JUMP -100016) -100017 -21 -100001
(LQ STATIC+MAIN#FRT) (PLW 0) (PLQ STATIC+MAIN#FRT) (PLW 65) GET (PLQ 1) A- A+ (PLQ STATIC+MAIN#FRT
) (PLW 0) (PLQ STATIC+MAIN#FRT) (PLW 66) GET (PLQ 1) A- A+ GET (PLQ 1) A+ PUT (LQ STATIC+MAIN#FRT
) (PLW 20) (PLW 14) (PLW 1) A- (PLW 45) (PLW 44) A- (PLW 20) A* A+ A+ (PLQ STATIC+MAIN#FRT) (PLW
0) (PLQ STATIC+MAIN#FRT) (PLW 66) GET (PLQ 1) A- (PLW 45) GET A* (PLW 44) A- (PLW 20) A* A+ A+
(PLQ STATIC+MAIN#FRT) (PLW 20) (PLQ STATIC+MAIN#FRT) (PLW 65) GET (PLQ STATIC+MAIN#FRT) (PLW 66
) GET A/ (PLW 1) A- (PLW 45) (PLQ STATIC+MAIN#FRT) (PLW 65) GET A* (PLW 44) A- (PLW 20) A* A+
A+ GET A* PUT -24 (LQ NIL) (JUMP -100000) (LQ NIL) -100000 (CALLH STOP#FRT 1 0) RETURN (END 563
) )

```