APLインタプリタのファームウェア化とその効果について†

宮脇冨士夫†† 木下耕二†††
渡辺勝正††† 萩原宏†††

APLは会話型高級言語として，他の言語にみられない特徴をそなえているが，そのために，コンパイラ方式には不向きな面があって，インタプリタ方式で処理されるのが普通である．したがって，処理時間が大きくなることが問題である．この点に対する有力な解決手段として，インタプリタをファームウェア化する方法が注目されている．

本論文は，我々のシステムの中で，マイクロプログラム化した部分について，その機能の説明とそれぞれの処理時間の評価を示すこと，およびインタプリタのファームウェア化による総合的な効果を明らかにするために，ソフトウェアによる処理時間との比較を示すことを主体としている．また，議論の基礎として，実験に使用したハードウェアの性能，処理方式，およびシステムのアーキテクチャについて，あらかじめ述べている．

1. まえがき

APL会話型言語は，我が国では，まだあまり馴染まれていない言語であるが，欧米では応用分野も広く，会話型言語の一角に定着している言語である[1]．

我々もその有効性に着目し，APL会話型処理システムの実現にとりくんで来たが[2]，最近，APLを使用できるパーソナル・コンピュータが市販され[3]，手軽に利用できるようになりつつあるので，我が国においても，今後，利用者の増加がみられるであろうと思われる．

APLはFORTRAN等と同様，手続き言語であるが，演算要素の属性が動的に変化する特徴を持っており，コンパイラ方式で処理される言語とは異なっている．したがって，インタプリタ方式で処理するのが普通であるが，その場合，処理時間が大きくなることが問題である．現在，APLの処理に関する研究では，インタプリタの処理速度向上に重点がおかれており，これに対する1つの解決方法として注目されているのが，インタプリタのファームウェア化である．このことに関して，いくつかの報告があるが[4)~7)]，処理方式およびシステムの構造について，決定的な評価はまだなく，ハードウェア技術の進歩に支えられた高級言語処理の具体的なテーマとして，数多くの試みがなされなければならない段階である．我々のAPL会話型処理システムについては，先にインタプリタの解析について報告したが[8]，本論文では，それにもとづいて実際にファームウェア化した部分，マイクロプログラムの大きさ，ファームウェア化の効果，および典型的なAPL文における総合的な効果について報告する．また，その前提となるハードウェア，処理方式，およびシステムの構造についても述べる．本研究は，APL処理システムのファームウェア化の効果について目安を与えるものである．

2. ハードウェア

実験に使用した計算機はHITAC-10およびHITAC-8350（以下，H 10，H 8350とよぶ）である．我々はまずH 10の上にソフトウェアでシステムの機能を実現し，そのプログラムを分析することによってファームウェア化の要点を把握したうえで，H 8350の上にうつす方法をとった．後の章で，前論文[8]にのべたH 10のデータを引用するが，比較の基礎として，システム作成に主として使った単純な機械語命令の処理時間を**表1**に示す．これをみると，H 8350よりH 10の方が若干速くなっているが，間接命令を使用するとH 10の方が遅くなる．我々のプログラムにおける命令の使用頻度を（分岐：20%，シフト：10%，格納：20%，その他 50%）として計算すると，H 8350

† Effectiveness of Firmware in an APL Interpreter by FUJIO MIYAWAKI (Himeji Institute of Technology), KOUJI KINOSHITA, KATSUMASA WATANABE, and HIROSHI HAGIWARA (Department of Information Sience, Kyoto University).

†† 姫路工業大学電気工学科
††† 京都大学工学部情報工学科

表 1 基本命令の処理時間ならびに H-8350 の仕様
Table 1 Processing time and specification.

命令	H-10*	H-8350	
	16ビット	32ビット	
L (置数)	2.8 μs	2.9 μs	・水平型マイクロプログラム制御方式
ST (格納)	2.8	3.45	・RCM (Reloadable Control Memory) 1024ワード (60ビット／ワード)
A (加算)	2.8	2.9	・マシンサイクル 0.4 μs／ワード
S (減算)	2.8	2.9	・スクラッチパッドメモリ 128ワード (32ビット／ワード)
O (論理和)	2.8	2.9	
N (論理積)	2.8	2.9	
X (排他的論理和)	2.8	2.9	
SLL (左シフト)	2.8~9.8	2.7~15.1	
SRL (右シフト)	2.8~9.8	2.7~15.1	
B (分岐)	1.4	1.9~2.75	
BAL (連係分岐)	2.8	3.55	

* H 10 の場合，間接命令になれば，一様に 1.4 μs 増す．

の処理速度は H 10 の ±24% 内にあり，実際は間接命令が 30% 程度であるから，ソフトウェアによる処理速度に関して H 10 と H 8350 の差はほとんどない (10% 以下) と判断した．以下，特に断らないかぎり，H8350 のデータである．

H 8350 のマイクロプログラムに関する仕様についても表 1 に示したが，スクラッチパッドメモリ 128 ワードのうち，我々が自由に使用できるのは，汎用レジスタ (16 ワード) をのぞくと，32 ワードである．しかし，その使用方法は一様ではなく，比較的使い易いのは，そのうち 8 ワードである．我々が作成したマイクロプログラムは，RCM に納まりきらないので，メインメモリを補助記憶とし，RCM の一部分をオーバレイ領域として使って実験した．後に示す処理時間の評価では，ファームウェア化の効果を明らかにするため，オーバレイ処理に要する時間は無視した．また，H 8350 のアーキテクチャはマイクロプログラムのレベルでサブルーチンを使う機能が十分でないために，マイクロプログラムの階層構造が作りにくい．したがって，マイクロルーチンを共用するために，ひとまとまりのマイクロプログラムをあえて分割したり，一部分を既存の機械語で作ったりしなければならなかった．以下の文において，ソフトウェアというのは既存の機械語で作った部分を指している．

表 2 中間表現の各要素
Table 2 Elements of intermediate text.

機能部	略記	補助部	摘要
END		文の全長バイト数	普通文または関数頭文の先頭
ENDS			STOP 指定文の先頭
ENDT			TRACE 指定文の先頭
ENDTS			TRACE, STOP 指定文の先頭
END⍝			コメント文の先頭
ENDE			関数の末部
:;[ ]( ) ⍞⎕ . ∘ ← →		演算子表の先頭番地から該当記号のエントリ・ポイントにいたる変位バイト数	各演算子が固有の機能部をもつ
単項演算子	単		~ ⍋ ⍒
純2項演算子	純2		↑ ↓ T ⊥ ∈ \ ⊖
特種2項演算子	特		/ ⌿
純単2項演算子	純12		? ⍳ ρ ⌽ ⊖ ⍉ ,
合成可能2項演算子	合2		< ≤ = ≠ ≥ > ∧ ∨ ⍲ ⍱
合成可能単2項演算子	合12		+ − × ÷ * ⌈ ⌊ ! ⍟ ○
大域名前	大	*1	実行モードの文にあらわれる名前
大域定数	定	*2	実行モードの文にあらわれる定数
局部名前	局名	関数辞書部の該当エントリ・ポイントにいたる相対バイト数	定義関数内でのみ有効な名前
関数大域名前	関大		定義関数内で使われる大域名前
引数	引		*1：名前表先頭番地からエントリ・ポイントにいたる変位バイト数
ラベル	ラ		*2：大域定数領域の先頭番地からエントリ・ポイントにいたる変位バイト数
局部定数	局定		

3. 処理方式とシステムの構造

処理方式は原文を中間表現に変換し，それをインタプリタが解釈実行する方式である[8]．これは他の報告[4)~6)]にも共通している方式であるが，中間表現およびシステムの構造については，それぞれに工夫のみられるところである．我々の中間表現の例を図 1 に示す．すなわち，原文の各演算要素に 1 ワード (32 ビット) の中間表現が対応し，原文と全く同じ順序に並んでいる．先頭の END マークは制御情報である．各要

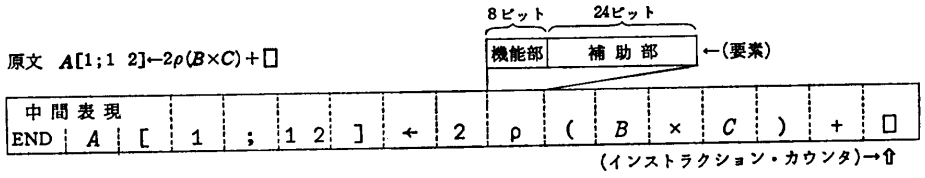

図 1 原文と中間表現の対応ならびに要素の構造
Fig. 1 Source statement, intermediate text and its element.

素は機能部と補助部からなっている．我々の選定した機能部の種類および補助部の意味について**表2**にまとめて示す．中間表現をこのように設定したのは，次の2項目を勘案した結果である．

（1） 原文の再現性

（2） 処理速度の向上

次にシステムの構造について，各領域，表，スタックの全体的構成を**図2**に示し，以下，各部分の要点をのべる．

中間表現領域： 実行モードにおける中間表現を格納する．処理の中で入力要求（□，[T]）に対する入力文の中間表現を格納することもあるので，スタック構造になっている．1文の実行が終るとポップアップする．

大域定数領域： 中間表現に含まれる大域定数に対する実データを1文ごとにまとめてスタックする．

演算子表： APL 文に現れる名前，定数以外の各記号に対するエントリからなり，各エントリは記号の内部コード，中間表現，および演算処理のあるものは処理ルーチンの入口を指すポインタからなっている．

名前表： 各エントリは，実データを指すポインタと名前の内部コードからなる．データ属性の動的変化による影響が中間表現におよばないようにするために重要な役割を果している．

データ領域： 大域名前，局部名前に対応する実データを格納する．ガーベッジ・コレクションが必要なため，バックポインタで名前表および関数スタックの局部名前と連絡している．

実行スタック： インタプリタから演算ルーチンに，処理に必要な情報を伝達するためのスタック．図2に示すように，6種類の要素と文脈の区切を示す END マークがスタックされる．

一時データ領域： 処理の途中結果をスタックする．

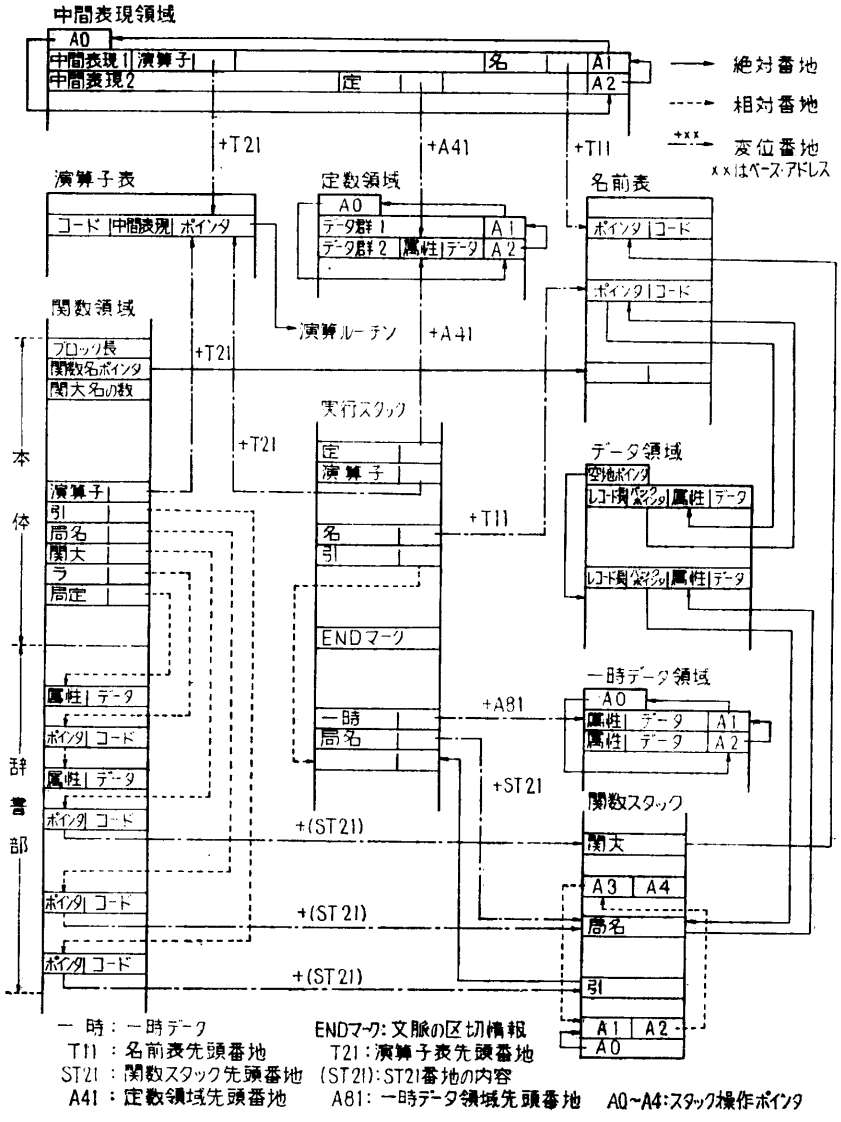

図2 スタック，表，領域の構造と関連

Fig. 2 Stacks, tables, areas and their organization.

関数領域： 定義関数に対する情報を格納する．1つのブロックは中間表現の本体と辞書部からなり，辞書部には名前の内部コードおよび局部定数データが含まれている．演算子以外はすべて辞書部を通して実データに連絡する．

関数スタック： 関数の回帰性を処理するために必要である．関数頭文にある引数と局部名前に対するエントリ，および関数大域名前と名前表を連絡するためのエントリがスタックされる．

4. ファームウェア化をおこなった部分

我々は先の報告[8]にもとづいて，使用頻度の高い機

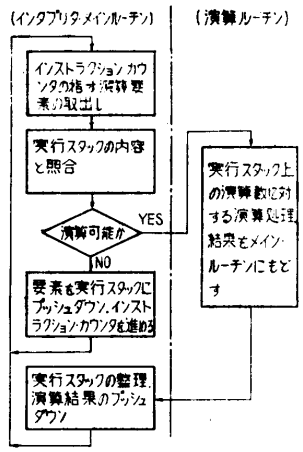

図 3 インタプリタの基本的な流れ
Fig. 3 Fundamental flow of the interpreter.

能単位とインタプリタの主要部分，および演算子の処理についてファームウェア化をおこなったが，この章では，インタプリタにおける効果に重点をおいてのべる．

インタプリタは中間表現をみながら，APL の演算順序（原則として右の演算子の優先順位が高い）に従って処理を進める．基本的な処理をフローチャートの形で示したものが**図 3** である．大きくは，インタプリタ・メインルーチンと演算ルーチンに分れているが，処理の要点は次の 3 つである．

（1） 取り出した要素と実行スタックの状態（インタプリタの内部状態）の組合せに対応する処理の決定（デコード処理）．

表 3 主要な内部状態と各状態において判別されるべき要素
Table 3 Main internal states and intermediate elements to be decoded.

状態	実行スタック上の要素	判別される演算要素
1	Ⓔ	]) □ ⍞ 名 関大 その他
2	(数) Ⓔ	単 名 関大 純2 合2 特 純1,2 合1,2 — その他
3	(特) (数) Ⓔ	合2 合1,2 その他
4	$\oplus_{1,2}$ (数) Ⓔ	名 関大 引 局名 局定 定 ラ □ ⍞) .] その他
5	$\oplus_{2}$ (数) Ⓔ	名 関大 □ ⍞] .) その他
6	— (数) Ⓔ	] □ ⍞ 名 関大 その他
7	(数) Ⓔ	END ENDS ENDT ENDTS : ; [(— その他
8	. $\oplus_{c}$ (数) Ⓔ	∘ その他
9	∘ . $\oplus_{c}$ (数) Ⓔ	名 関大 □ ⍞]) その他
10	$\oplus_{c}$ $\oplus_{c}$ (数) Ⓔ	名 関大 □ ⍞]) その他

Ⓔ：文脈の区切を示す END マーク　(数)：演算数　(特)：特種2項演算子　$\oplus_{1,2}$：単2項演算子（純$_{1,2}$，合$_{1,2}$）　$\oplus_{2}$：2項演算子（純$_{2}$，特，純$_{1,2}$，合$_{2}$，合$_{1,2}$）
$\oplus_{c}$：合成可能演算子（合，合$_{1,2}$）

（2） 演算子をスタックにつむ（プッシュダウン処理）．

（3） 演算処理．

デコード処理は，演算要素を取り出すごとに必要な処理であって，ソフトウェアによるインタプリタではメインルーチン処理時間の 10〜30% を占めていた部分である[8]．プッシュダウン処理は，演算要素の種類によって操作が異なるが，実行頻度が高く，時間を要する部分を含んでいるので高速化がのぞましい．演算処理については，個々の演算子のファームウェア化が対象となるが，ここでは共通性のある基本演算子の評価をのべる．

4.1 デコード処理

デコード処理とは，インタプリタの各状態において，取り出した演算要素の機能部を**表 3** に示すように判別し，状態と機能部の組合せに対応する処理に制御を渡すまでの処理である．我々はこの部分をファームウェア化するに当って 2 つの試みをした．1 つは，部分的なファームウェア化である．すなわち，デコード処理を分析して，処理に適切な機能単位をファームウェア化し，それを使ってプログラムする方法である．我々が選んだ機能単位は次の 2 つである．

（1） 取出した要素の機能部が特定コードであれば，指定の処理をして，指定の状態に移る機能 (BMP 2)．

（2） 取り出した要素の機能部が特定コードの集合に属するならば，指定の処理をして，指定の状態に移る機能 (GMP 2)．

もう 1 つの方法は，機能単位に分割することなく，デコード処理全体をまとめて，効率よくマイクロプログラムする方法である．これら 2 つの方法について比較すると**表 4** となる．

処理速度のみに着目すれば，(2)のファームウェアによる方法が最も速くなるのは当然のことである．中央値でみて，(2)のソフトウェアの約 9 倍，(1)のソフトウェアの約 26 倍となる．一方，(1)のファームウェアについてみれば，(1)のソフトウェアの約 7 倍，(2)のソフトウェアの約 2 倍となる．メモリの使用量に着目すると，RCM の 1 ワードを 8 バイトとして，(2)のファームウェアによる場合が 2636 バイトと最も大きい．また，(1)と(2)のファームウェアを比較すると，処理速度は 3.6 倍であるが，RCM の容量は約 4.5 倍必要である．マイ

表 4 デコード処理におけるソフトウェアとファームウェアの比較
Table 4 Decode process in software and firmware.

処理方法	(1)機能単位を用いる方法		(2)全体を効率よくプログラムする方法	
	ソフトウェア	ファームウェア	ソフトウェア	ファームウェア
最大処理時間	421.8 μs	58.8 μs	101.95 μs	11.6 μs
最小処理時間	61.9	8.8	54.8	6.8
中央値	241.85	33.8	78.38	9.2
メイン・メモリ	2242バイト	2044バイト	1284バイト	748バイト
RCM		52ワード		236ワード

クロプログラムの量は，その作成に要する労力の目安でもある．機能単位の融通性に着目すれば，BMP 2，GMP 2 はシステムの他の部分でも使用可能であるが，(2)のマイクロプログラムは他の部分では使用できない特殊な機能である．以上のことから判断すると，システムのファームウェア化に当って，適切な機能単位の選定が，いかに重要であるかがわかる．いずれの方法をとるかは，環境によって異なるであろうが，我々のシステムでは処理速度の向上に重きをおいて，(2)の方法を採用した．

4.2 プッシュダウン処理

実行スタックにプッシュダウンされる演算要素は，演算子と演算数（大域名前，関数大域名前，局部名前，引数，ラベル，大域定数，局部定数）があるが，関数大域名前はスタックにつまれると大域名前となり，また，ラベルと局部定数は一時データとなって，その実データは一時データ領域に移される．図 2 に示したように中間表現における要素と実行スタック上の要素とはデータの連絡方法および機能部が異なり，変更しなければならないものがあるので，プッシュダウン処理も一様ではない．各要素のプッシュダウン時間について，ソフトウェアとファームウェアの比較を表 5 に示す．この処理では他の機能単位を有効に使うために，部分的にソフトウェアによっている．演算子～大域定数については図 2 から明らかなように本質的な処理のちがいはない．処理時間の差は要素の種類を判定する順序のちがいが主である．ラベル，局部定数では実データを辞書部から一時データ領域に移すために処理時間が大きくなる．ソフトウェアとファームウェアの比較をみると，処理速度について約 10 倍の向上が得られている．この処理についても，前節と同様に，機能単位 (BMP 1, GMP 1) の部分のみをファームウェア化して試算してみると，わずか 43 ワードのマイクロプログラムで約 3 倍の速度向上が得られる．

4.3 演算処理

APL の演算子は基本演算子 (primitive function)，混合演算子 (mixed function)，および複合演算子 (composite function) に類別されているが[9]，ここでは基本演算子に重点をおいてのべる．

基本演算子の場合，演算数がスカラであれば結果はスカラとなり，配列であれば演算子が要素ごとに作用し，結果も配列となる．配列の加算について APL の意味を FORTRAN で示すと図 4 となる．したがって基本演算子の処理をフローチャートで示すと図 5 となる．コントロール部と演算子固有の処理に分離され，コントロール部のマイクロプログラムはすべての基本

APL	FORTRAN
$C \leftarrow A+B$	DO 10 I=1,M
	DO 10 J=1,N
	10 C(I,J)=A(I,J)+B(I,J)

図 4 配列加算の APL と FORTRAN による表現
Fig. 4 Addition in APL and FORTRAN.

表 5 プッシュダウン処理におけるソフトウェアとファームウェアの比較
Table 5 Pushdown process in software and firmware.

要素	ソフトウェア	ファームウェア（機能単位）	ファームウェア
演算子	210.6 μs	81.4 μs	(19.9)*1+15.4 μs
大域名前	370.1	98.2	(19.9)+17.8
引数	450	128.3	(19.9)+21.4
局部名前	506.2	134.7	(19.9)+22.2
関数大域名前	562.7	141.4	(19.9)+23.4
大域定数	658	138.6	(19.9)+19.8
ラベル	875.5	404.1	(48.6)+48.5
局部定数	1244.2+19.7n*2	723.3+19.7n	(47.5)+185.5+7.2n
使用する機能単位	BMP 1 GMP 1	LGTH CTOC*3	LGTH CTOC
プログラム	メイン・メモリ 742 バイト	マイクロプログラム 43 ワード	メイン・メモリ 244 バイト マイクロプログラム 157 ワード

*1: ( ) 内はソフトウェアによる部分 *2: 数値ベクトルの例 (n: 転送語数) *3: BMP 1 (要素の機能部が指定コードでなければ，指定番地にとぶ.) GMP 1 (要素の機能部が指定のコード集合に属さなければ，指定番地にとぶ) LGTH (実データの全長ワード数を得る) CTOC (データの転送).

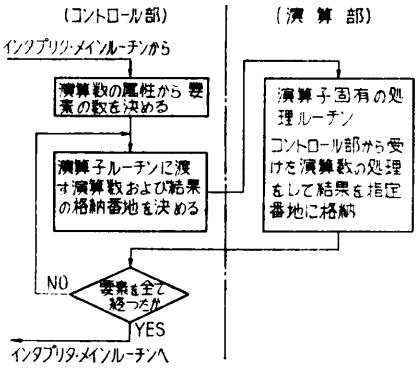

図 5 基本演算子処理の流れ
Fig. 5 Flow chart to process primitive functions.

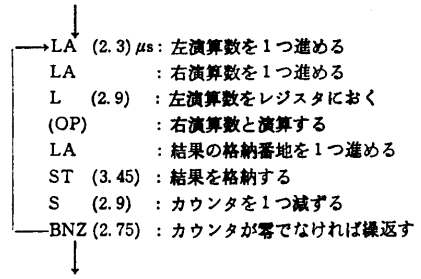

図 6 配列の要素対要素演算処理の機械命令
Fig. 6 Machine code to operate element by element in array.

演算子に共通に用いられる．複合演算子についても同様に考えられる．コントロール部についてファームウェア化をおこなった結果を**表 6** に示す．

基本二項演算子の配列の場合についてみると，1要素当りの演算数準備処理に要する時間は $272.7/n+5.4$ (μs) となる．APL では演算数の属性変化を許しているために，演算の準備として属性の判断をする時間が余分に必要である．これは，演算数の属性が固定している言語に比べて，処理速度の点で不利になるところであるが，今，属性が固定している言語の配列演算をハンド・コンパイルして H 8350 の機械語で部分的にオブジェクトを作ると，1つの例として**図 6** が考えられる．このルーチンで準備処理に要する時間は1要素当り 18.85 (μs) となる．したがって $n=20$ でほぼ同じ程度となり，n がそれ以上になるとマイクロプログラムによる我々のシステムの方が速くなる．

演算子固有の処理として，加減乗除の演算子について，マイクロプログラムの大きさと処理時間を**表 7** にあげる．処理されるデータによって処理時間に大きな差があるが，これはきめのこまかいマイクロプログラムの結果である．H 8350 の浮動小数点演算処理時間（±：13.4 μs，×：46.31，÷：67.67）に比較して，3〜4 倍であるが，これはデータの形式が本質的に異なるためである．我々のシステムにおける数値データは**図 7** に示すような BCD コードの浮動小数点形式である*．このようなデータ形式にもかかわらず，表に示す処理速度が得られたことは，ファームウェアによる効果として評価すべきものと考える．ちなみに，前に報告した[8] H 10 による加算では 6335 マシンサイクル，すなわち，8869 μs かかっているが，これに比

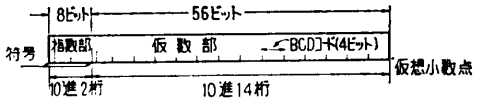

図 7 BCD 浮動小数点形式
Fig. 7 BCD floating number.

表 6 基本演算子，複合演算子のコントロール部の処理時間
Table 6 Time to control primitive and composite functions.

種 類	コントロール・メモリ	処 理 時 間
基本単項演算	86+(78)* ワード	スカラ：75.65+42+(OP) μs アレイ：75.65+143.2+(5.2+OP)×n
基本2項演算	298+(117)*	スカラ：84.3+90.2+(OP) アレイ：84.3+188.4+(5.4+OP)×n
リダクション(/)	161+(46)*	ベクトル：60.25+36.4+(17.2+OP)×(n−1)

アンダーライン：ソフトウェアによる部分，n：配列の要素数，OP：演算処理時間，( )*：共用部分．

表 7 ファームウェアによる基本演算子処理時間例
Table 7 Example of primitive function in firmware.

種 類	マイクロプログラム	処理時間	処 理 例
+ −	166 ワード	10.8〜137.2 μs	123.4+0.05678=123.45678 60 μs
×	176	8.4〜644	0.05678×123.4=7.006652 122
÷	194	5.6〜840.4	7.006652÷123.4=0.05678 185

* APL では実数型，整数型の区別がない．内部データ形式は一様であることがのぞましい．10進〜2進の変換誤差を許さない処理がある．これらの理由から，内部データとして，BCD コード浮動小数点型に統一した．

表 8 典型的 APL 文に対するインタプリタ・メインルーチン処理時間のソフトウェアとファームウェアの比較
Table 8 Time of interpreter main routine to process typical APL sentences in software and in firmware.

APL 文	ソフトウェア (H 10)	ファームウェアによるもの	効果
A	4086.6 μs	(332.95)+214.15 μs	7.5倍
⊕₁A	5653.2	(651.1)+410	5.3
B⊕₂A	8113	(773.4)+492.7	6.4
B⊕₁,₂⊕₁,₂A	10470.6	(1064)+732.33	5.8
(C⊕₁,₂B)⊕₂A	16609.6	(1519.8)+894.38	6.8

( ): ソフトウェアによる部分 A, B, C: 大域名前
⊕₁: 単項演算子 ⊕₂: 純2項演算子 ⊕₁,₂: 単2項演算子

較すると，148倍となる．もっとも，これはファームウェア化の効果だけではなく，アーキテクチャのちがいにもとづく要因が大きいことを考えて評価すべきである．すなわち，H 10 では BCD コード1桁の加減算サブルーチンを作り，それを繰返し使用して処理をおこなったのに対し，H 8350 のマイクロオーダには，BCD コード2桁の加減算が1マシンサイクルで処理できる機能があることによる．

5. 典型的な APL 文の処理時間

先の章では部分的な評価をおこなったが，他の機能単位の効果も含めて，基本的な APL 文についてインタプリタ・メインルーチンの処理時間を総合的に評価すると表8となる．

我々は先の論文において[8]，4倍以上の処理速度向上を期待したが，これらの結果からみて，約6倍の効果が得られた．しかもファームウェア化されなかった部分について，全体の処理時間に占める割合が 60% と目立ってきた．ファームウェア化された部分のみに着目すると，ソフトウェアに比較して 13 倍の向上をみたといえる．

6. ま と め

我々は，あらかじめソフトウェアによるシステムを実現し，それを分析することによってファームウェア化の要点をつかみ，重点的にマイクロプログラミングをおこなった結果，インタプリタの処理速度は約6倍の向上を得た．

部分的なファームウェア化において，2通りの方法を示し，適切な機能単位抽出の重要性を指摘した．

インタプリタの処理時間のうちファームウェア化をおこなわなかった部分を明らかにし，なお処理速度の向上をはかる余地のあることを示した．

我々のシステムは，現在，実行モードの処理が可能であり，演算子は28種類（単項演算子：＋－×÷○ ρ ι・2項演算子：＋－×÷＞≥＝≠≤＜⌈⌊ ρ ι・←[ ]；複合演算子：リダクション 外積）が実現されている．プログラム領域は約 45 キロバイト，マイクロプログラムは約 4.8 キロワードである．

APL 処理における今後の課題としては，演算子の特徴をとらえて処理のむだをはぶく方法[10]，演算子の並列性を本質的に処理する方法[11]の具体化があげられる．

最後に，富田真治博士をはじめ研究室の諸氏の御協力に謝意を表します．

参 考 文 献

1) APL Congress '73 North-Holland Publishing Company (1973).
2) 渡辺豊英，宮脇冨士夫，渡辺勝正，萩原 宏：ミニコンにおける APL 会話型処理システム，情報処理，Vol. 16, No. 9, pp. 781-788 (1975).
3) 小柳ゆき子：パーソナル・コンピュータ，情報処理，Vol. 18, No. 4, pp. 381-385 (1977).
4) Zaks, R., Steingart, D., and Moore, J.: A Firmware APL time-sharing system, AFIPS Conf. Proc., Vol. 38, 1971 SJCC, pp. 179-191.
5) Hassitt, A., Lageschulte, J. W., and Lyon, L. E.: Implementation of High Level Language Machine, Commun. ACM, Vol. 16, No. 4, pp. 199-212 (1973).
6) Edited by Yaohan Chu: High Level Language Computer Architecture, pp. 199-212, Academic Press (1975).
7) Amram, Y., B de Cosnac, Granger, J. L., Smoucovit, A.: An APL Interpreter for Minicomputer, A Microprogrammed APL machine, APL Congress '73 pp. 33-39.
8) 宮脇冨士夫，渡辺勝正，萩原 宏：APL 会話型処理システムのインタプリタの分析とファームウェア化の要点，情報処理，Vol. 19, No. 5, pp. 390-397 (1978).
9) Katzan, Jr., Harry: APL programming and Computer Techniques, pp. 307-309 Van Nostrand Reinhold Company (1970).
10) Abrams, Philip S.: An APL Machine, Stanford Linear Accelerator Center Report SLAC-114 (1970).
11) Thurber, Kenneth J. and Myrna, John W.: System Design of Cellular APL Computer, IEEE Trans. Computers. Vol. C-19, No. 4, pp. 291-303 (1970).

(昭和53年4月27日受付)
(昭和53年9月21日採録)