論　文

APL 会話型処理システムにおけるインタプリタの分析とファームウェア化の要点*

宮脇 冨士夫** 渡辺 勝正*** 萩原 宏***

Abstract

We intend to implement an APL interactive processing system in firmware. One of the most important points for it is to evaluate the processing time of each part of the system. In this paper, we make an analysis of the interpretation part which has been implemented in software. Firstly, we show the state-diagram and each state-transition time. Secondly, according to the diagram, we integrated the processing time of the fundamental APL sentences. In addition, the outlines of the functional elements of the system are described. And their frequency, used in the processing of some examples, is also shown.

1. まえがき

APL 会話型言語は簡潔性，論理性，機能の多様性を備えたユニークな言語である[1),2),3)]．我が国における APL の評価はまだ十分ではないが，外国では APL に関する国際的な大会[4)]が開催されるなど会話型言語の一角に定着している．

APL 会話型言語の特徴を生かすための一番大きな問題は，いかにして処理速度をあげるかということである．我々が APL 会話型処理システムのファームウェア化にとりくむ理由もここにある．APL のファームウェア化に関する研究としては外国に先駆的なもの[5),6)]がある．また我が国においても部分的な研究がある[7),8),9)]．しかしまだ決定的な評価はなく，よりよい処理システムを求めて多くの例をつみ上げねばならない段階である．

処理システムをファームウェア化する場合の要点としては，先ず第一にシステムの構造を洗練し，適切な機能単位を抽出すること，第二にシステムの各部分の処理時間および機能単位の使用頻度に関するデータを把握することである．

我々のシステムの構造[10)]は大きく分けると，APL のソース文を中間表現に編集するコンパイラの部分と，その中間表現を解釈実行するインタプリタの部分からなっている．この論文では後者について，処理速度の分析とファームウェア化の要点を述べることに重点をおいているが，先ずインタプリタの処理方式と状態遷移図について説明する．これはソフトウェアによるシステムを作成して十分性を確認したものである．次に各状態遷移に要する時間を検討した結果にもとづいて，基本的な APL 文の処理時間を分析する．第三にシステム作成の中から抽出された機能単位について機能の説明とプログラムの大きさおよび使用頻度の測定結果を示す．そして最後にファームウェア化の効果が大きいと判断される点について総括する．

2. インタプリタの処理方式と状態遷移図

インタプリタに制御が渡った段階では，APL のソース文は中間表現に編集されて **Fig. 1** に示すように中間表現領域に格納されている．

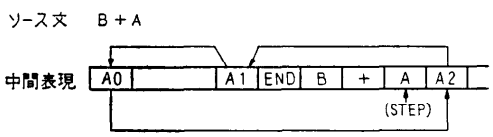

Fig. 1 Intermediate expression.

* Analysis of the Interpreter of the APL interactive processing system and the points to implement in firmware by Fujio MIYAWAKI (Himeji Institute of Technology), Katsumasa WATANABE, and Hiroshi HAGIWARA (Department of Information Science, 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

インタプリタは有限オートマトンである．その状態遷移図を **Fig. 2** に示すが，機能を大きく分けると，①一入力文の処理をする主部分 (Fig. 2-a)，②関数の処理部分 (Fig.2-b)，③添字の処理部分 (Fig.2-c)，④演算式の処理部分 (Fig. 2-e) がある．②と④は対等である．すなわち，演算式の中に関数名があると，スタックに END マークをつんで④から②に移り，関数の中で演算式を処理する時は②から④に移りネストを構成する．③と④の関係も同じである．END マークの処理部分 (Fig. 2-d) はネストをもとにもどす時に補助的な役割をはたす．

インタプリタの初期状態は (SE0) である．(SE0) —84→ (SE1) の処理でインストラクション・カウンタ (STEP) は処理すべき最初の要素を指すように設定される．Fig. 1 は (SE1) における (STEP) の位置を示している．状態遷移図の各枝につけられた値は状態遷移に要する処理時間の必要単位数を示している．前記の 84 は (SE0) から (SE1) に移る間の所要時間が 84 単位であることを意味している．これはプログラムのステップを追って計算したものである．我々のソフトウェア・システムに使っている計算機 (HITAC-10) では 1 単位当り 1.4 μs である．したがって (SE0) から (SE1) に移る時間は 117.6 μs となる．所要時間の計算は処理が複雑になると評価が難しくなり，正確に一つの数値として把握することが困難であるが，処理の分岐点においては各枝に等しく分岐するとし，具体的な状況が必要な場合は次のように想定して数値を出した．(1) データは 3×3 の配列，(2) データの有効桁は 5，(3) 名前の長さは 4 文字，(4) 名前表の 5 番目に的中名がある，(5) 実行スタックのネスティング・レベルは 1．

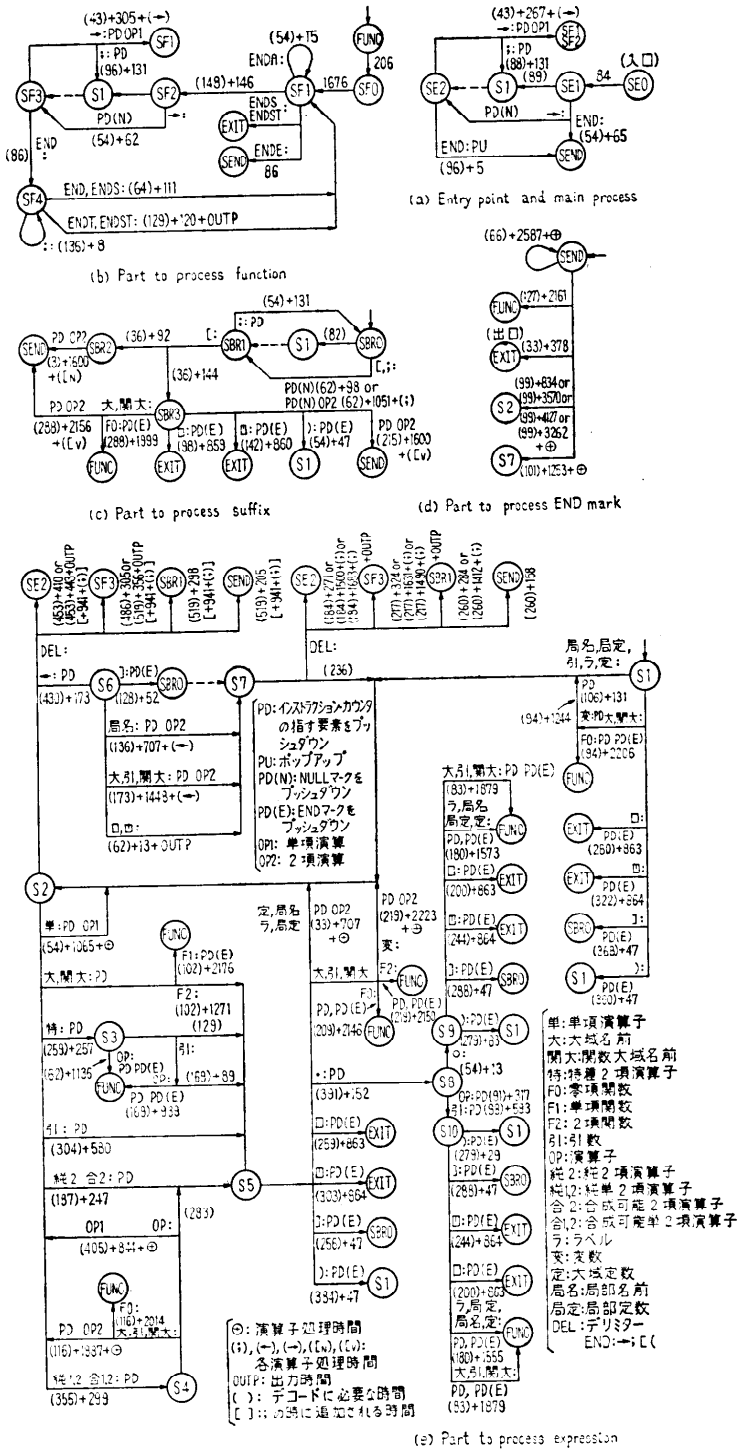

Fig. 2 Interpreter state-diagram.

インタプリタの状態遷移を説明するために，先ず演算数と演算子の組合せだけの単純な演算式の例を述べる．ついで，( ) の処理，[ ] に囲まれた添字の処理，定義関数の処理，および合成演算の処理について要点を説明する．

単純な演算式 ～A＝B←C＋－D/E の状態遷移

この式に対する中間表現と各状態における (STEP) の位置および状態遷移を抜出したものが **Fig. 3** である．(SE0)[1]→(SE1)[2] は初めに述べたように演算式の右端の要素に (STEP) を設定する処理である．(S1)[3]～(S2)[17] が演算式の処理をする主な状態遷移である．それは APL の演算規則(右の演算子から順に処理する) に従った，スタック操作と演算の単純な繰返しである．スタックの使い方については 3 章で例をあげて説明する．(SE2)[18]→(EXIT)[20] は後処理である．(S2)[17]→(SE2)[18] の遷移選択は実行スタックのボトムにある END マークの情報[10]によって決定される．

( ) の処理

(STEP) が「)」を指すと，3 章の例が示すように，インタプリタは「)」に対応する END マークを実行スタックにプッシュダウンして式のネスティングのレベルを一段上げる．状態は (S1) に移り ( ) 内の演算式を処理する．( ) 内の処理が終って (STEP) が「(」を指す時 (状態は (S2) もしくは (S7))，実行スタックの様子は **Fig. 4**(次頁参照) に示すようになっている．インタプリタは先にプッシュダウンした END マークの情報にしたがってレベルをもとにもどし，保留されていた処理を実行する．再び状態は (S2) に移って中間表現の解釈実行をつづける．

添字の処理

(STEP)が「]」を指すと，「]」に対応する END マークをプッシュダウンしてレベルを一段上げる．状態は (SBR0) (Fig. 2-c) に移り [ ] 内の演算式の処理をする．(STEP) が「[」を指す時 (状態は (SBR1))，実行スタックのトップを指すポインタ (STAK) は **Fig. 5** (次頁参照)の (STAK)[1] の位置を指している．ここで保留されている演算が (←) であれば添字に対応する配列要素の番地を計算する 2 項演算子「[N」をプッシュダウンし，状態は (SBR2) に移る．その他の演算であれば添字に対応する要素の値を計算する 2 項演算子「[V」をプッシュダウンし，状態は (SBR3) に移る．この時スタックは Fig. 5 の (STAK)[2] の指す状態になっている．その後，処理が進んで添字演算子 ([N, [V) に対する

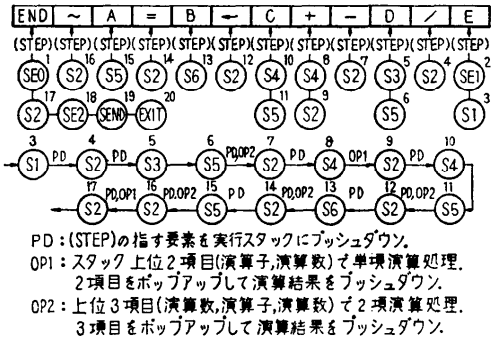

Fig. 3 Instruction counter and state-transition to interprete an APL sentence.

左演算数が確定すると，添字演算がなされ，END マークの情報にしたがってレベルをもとにもどし，保留されていた処理をする．状態は再び (S2) に移って中間表現の解釈実行をつづける．

定義関数の処理

2 項関数について述べる．2 項演算処理を要求する状態 (S5) では，(STAK) は **Fig. 6** (次頁参照) の (STAK)[1] の位置を指している．インタプリタはスタック上の 2 項演算子の位置に関数名があることを判断して 2 項関数に対応する END マークをプッシュダウンし，レベルを一段上げて状態 (FUNC) (Fig. 2-b)に移る．関数の処理が完了すると，先につんだ END マークの情報によって後処理をし，レベルをもとにもどす．状態は再び (S2) にもどって保留されていた中間表現の解釈実行をつづける．

合成演算子 (リダクション，外積，内積) の処理

我々のシステムでは定義関数の形式を拡張して合成演算の処理をしている．したがって処理過程は前述の関数の場合と同じである．ただ実行スタックの状態が若干異なるので **Fig. 7**(次頁参照) に示しておく．

3. 基本的な APL 文の処理時間の分析

次に基本的な APL 文 8 種について処理を分析し，その処理時間を計算するが，細部の説明については次の一例にとどめる．

解析例：$(C\oplus_{1,2}B)\oplus_2 A$

これは ( ) の処理を含む例である．先ず単 2 項演算子 $\oplus_{1,2}$ による 2 項演算 $C\oplus_{1,2}B$ を実行し，その結果を左演算数とし A を右演算数として 2 項演算 $\oplus_2$ を実行する．そして結果を出力する．

この例について状態の遷移と所要時間およびアルゴリズムの概略を次に述べるが，各状態における(STEP)

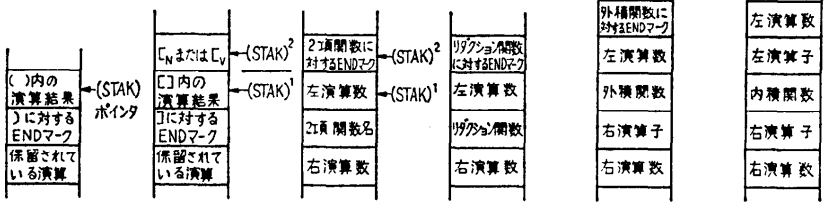

Fig. 4 Stack to process parenthesized expression.

Fig. 5 Stack to process suffix.

Fig. 6 Stack to process binary function.

Fig. 7 Stack to process compound operator (reduction, outer product, inner product).

の位置と実行スタックの様子については **Fig. 8** に示している.

SE0 —84→ SE1 (1) (STEP) が中間表現の右端の要素を指すように設定.

SE1 —(89)→ S1 (2) 要素A が大域名前であることを認識.

S1 —(94)+1244→ S2 (3) (STEP) の指す要素を実行スタックにプッシュダウン.

S2 —(187)+247→ S5 (4) $\oplus_2$ をスタックにプッシュダウン.

S5 —(384)+47→ S1 (5) 「)」に対するENDマークをプッシュダウン. このENDマークには2項演算指示情報(T2)がついている.

S1 —(94)+1244→ S2 (6) B をプッシュダウン.

S2 —(355)+299→ S4 (7) $\oplus_{1,2}$ をプッシュダウン.

S4 —(116)+1887+($\oplus_{1,2}$ の2項処理)→ S2 (8) C をプッシュダウン. スタックの上位3要素で $\oplus_{1,2}$ の2項演算処理. 上位3要素をポップアップして, 演算結果をプッシュダウン.

S2 —(513)+205→ SEND (9) スタックをポップアップ.

SEND —(99)+3262+($\oplus_2$ の処理)→ S2 (10) ( )内の演算結果をENDマークの代りにスタックにつみ, 2項演算指示にしたがって, スタックの上位3要素で2項演算処理. スタックの上位3要素をポップアップして, 演算結果をプッシュダウン.

S2 —(453)+443+OUTP→ SE2 (11) スタックのトップにある要素を出力.

SE2 —(96)+5→ SEND (12) スタックをポップアップ.

SEND —(33)+378→ EXIT (13) スタックのENDマークの処理. 次のAPL文のよみこみにもどる.

この文について, 処理時間を総計すると, インタプリタのメイン・ルーチンの働く時間 11864, 単2項演算子 $\oplus_{1,2}$ の2項演算処理時間, 2項演算子 $\oplus_2$ の処理時間, および出力に要する時間 (OUTP) を合せたものである. またメイン・ルーチンの中で中間表現の

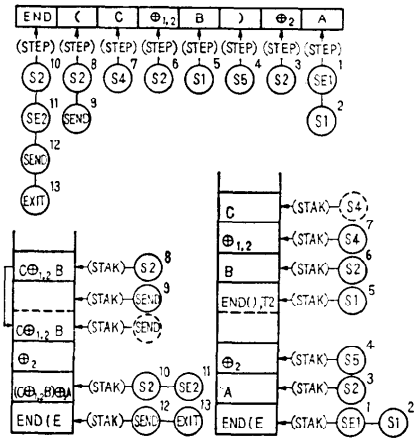

Fig. 8 Instruction counter and stack of each state.

要素を判断する, いわゆるデコードに要する時間に注目すると 2519 費しており, メイン・ルーチンに対する割合は 0.212 である. なお, 演算子の処理時間については単2項演算子の例として「+」の2項演算 (加算) を分析すると $1249+6335\times n$(n: 配列の要素数), また2項演算子の例として「∨」(論理和) を解析すると $1249+395\times n$ である.

同様の方法によって APL 文の基本的なものについて分析した結果を表にすると **Table 1**(次頁参照)となる. これから判断すると, 演算子の処理時間の割合が大きくなるであろうということが先ず第一にあげられる. これは APL の演算子が配列に対する演算を許していることから当然考えられることであるが, 一方, 要素の数が少ない時はインタプリタのメイン・ルーチンの処理時間の方が演算子の処理時間を上まわることも見逃がせないところである. その他, メイン・ルーチンの中でデコード処理に費される時間が 1〜3 割におよぶことが注目される. また前章の状態遷移図を検討してみると, デコード以外の部分で比較的処理時間

Table 1 Fundamental APL sentences and processing times.

基本文	*1	*2	演算子ルーチン処理時間*3
A	2919	765(0.262)	
$\oplus_1$A	4038	819(0.203)	～ (否定) 513+178×n
B$\oplus_2$A	5795	1171(0.202)	∨ (論理和) 1249+395×n
B$\oplus_{12}{}^L\oplus_{12}{}^R$A	7479	1996(0.267)	$-_1$ (符号反転) 513+327×n
(C$\oplus_{12}$B)$\oplus_2$A	11864	2519(0.212)	$+_2$ (加算) 1249+6335×n
C$\oplus_2$B[A]	10666	2563(0.240)	[ν (添字処理) 758+1655×n
B←A	4702	1099(0.234)	← (代入) 1362+15×n
B FUNC A	*4 10887	1185(0.109)	

*1: インタプリタ・メイン・ルーチンの処理時間（マシンサイクルを1とする）.
*2: メイン・ルーチンのうちで要素をデコードするのに要した時間（割合）.
*3: 演算数はBCDコード浮動小数点型(10桁). n は配列要素の数.
*4: 定義関数 FUNC の処理時間は入っていない.

の大きい部分（500 以上）はスタックのプッシュダウンとデリミタの処理である.

我々が現在実現しているシステムはファームウェア化を前提とした設計ではあるが APL 会話型処理システムの機能をソフトウェアでほぼ完全に実現した一つの例ともいえる. したがってこの分析による処理時間はファームウェア化の効果をみる上で一つの基準となるものである.

試みに A+－B の処理時間について考察してみると, Table 1 から処理時間の総計は $(9241+6662\times n)\times 1.4\mu s = 12937+9326\times n(\mu s)$ となる. 実際にファームウェア化した場合の処理時間は今後の研究によって明らかにしていかねばならないが, 一つの見通しとして, 後の **Table 2**(次頁参照)にあげた機能単位 IFA1 について所要単位数をしらべてみると, HITAC-10 では 24, HITAC-8350 のマイクロプログラムでは 23 とほぼ同じである. したがって HITAC-8350 の1単位は 0.4(μs) であるから少なくとも4倍以上の処理速度の向上を期待している. 文献[4]によれば6倍の効果が得られたと報告されているが, 妥当な値ではないかと思われる. これで推定すると B+－A の計算時間は $2156+1554\times n(\mu s)$ となる.

4. システムの主なエレメントの機能, 大きさおよび使用頻度

我々のソフトウェア・システムでは全体で比較的よく使われるであろうと思われるエレメントにカウンタを設けて, それぞれの使用頻度を測定できるようにしている. その測定例を Table 2 にあげる. 各エレメントについて使用頻度の高いものから順に並べ, それぞれのプログラムの大きさ, 機能の概略, およびエレメントの親子関係がつけ加えてある. 使用頻度は処理された問題によって異なるものであるから一概にはいえないが, この例は Katzan の著書[2]から, APL の機能を説明するために用いられている例をほとんどすべて実行してみたものである. この結果からみると, 分岐に関係するエレメントの使用頻度が高いことが目立つ. Fig. 2 の各状態における演算要素の判断と分岐には, BMP1, BMP2, GMP1, GMP2, IFA1 が用いられている. また, 使用頻度とプログラムの大きさの積をとってみると PDWO, LGTH, SYCH, ADD1, OPE2 等が比較的大きい数値を示す. SYCH はインタプリタでは使用されないが, PDWO はスタック操作の主役であり, LGTH, ADD1, OPE2 は演算子の処理でルーチンの中で主要な役割を果している.

5. あ と が き

以上, システムのインタプリタの部分について, 基本的な APL 文の処理時間の分析とエレメントの使用頻度の測定から, ファームウェア化の効果が大きいと期待される点を把握することができた. まず第一に各演算子の処理ルーチン, つづいてインタプリタのスタック操作およびデリミタの処理部分, そして分岐に関係するエレメントの処理速度を上げることが重要である. また, 我々のシステムでは合成演算の処理は関数を改造した形で定義して処理しているので, 合成演算の速度が非常に遅くなっている. したがって合成演算のファームウェア化も重要である. なお, この報告では述べなかったが, ソース文を中間表現に編集するコンパイルの部分については, その所要時間は全体の処理時間に比べて十分に小さく, 現状のソフーウェアでも満足できる程度であることをつけ加えておく.

最後に, 本研究を進めるに当り富田真治博士をはじめとして研究室の方々に御援助をいただいたことをここに記して謝意を表します.

Table 2 Functional elements, their frequency used, memory space required and descendants.

エレメント名と引数	語数	頻度 $\times 10^3$	機能	使用エレメント
IFA1 (A1, A2, A3)	17	157	(A1)≠A2 ならば A3 にとぶ	
BMA1 (A1, A2, A3, A4)	23	112	A2 をマスクとして (A1)≠A3 ならば A4 にとぶ	
SHL (A1, A2, A3)	20	103	A2～A3 番地のデータ全体を A1 ビット左にシフト	
BMP1 (A1, A2, A3)	21	85	A1 をマスクとして (((REGP)))≠A2 ならば A3 にとぶ	
CRS (A1, A2, A3)	31	71	A2～A3 番地のデータ全体を A1 ビット右に回転シフト	
IFA2 (A1, A2, A3, A4)	21	59	(A1)=A2 ならば A3 の処理をして A4 にとぶ	
IFP2 (A1, A2, A3)	19	57	(((REGP)))=A1 ならば A2 の処理をして A3 にとぶ	
BMP2 (A1, A2, A3, A4)	26	53	A1 をマスクとして (((REGP)))=A2 ならば A3 の処理をして A4 にとぶ	
SHR (A1, A2, A3)	29	53	A2～A3 番地のデータ全体を A1 ビット右にシフト	
GMP2 (A1, A2, A3, A4, A5)	36	43	A1 をマスクとして (((REGP)))=(A2～A3) のいずれかが成立てば，A4 の処理をして A5 にとぶ	
BGP2 (A1, A2, A3, A4)	29	37	(((REGP)))=(A1～A2) のいずれかが成立てば A3 の処理をして A4 にとぶ	
SETP (A1)	5	35	(REGP)←A1 (((REGP))) による分岐のためのセット命令	
CTOC (A1, A2, A3)	19	26	A1 語の連続したデータを先頭番地 A2 から先頭番地 A3 に転送	
SUBT (A1, A2, A3)	36	25	BCD コードの (A1)4−(A2)4−(A3)4 を実行し，結果を (A1)4 に格納. 上位より借があれば (A3)←1 なければ (A3)←0	
IF2 (A1, A2, A3)	16	15	(A1)>(A2) であれば A3 にとぶ	
BGP1 (A1, A2, A3)	24	14	(((REGP)))≠(A1～A2) ならば A3 にとぶ	
IFP1 (A1, A2)	14	13	(((REGP)))≠A1 ならば A2 にとぶ	
GMP1 (A1, A2, A3, A4)	31	12	A1 をマスクとして (((REGP)))≠(A2～A3) ならば A4 にとぶ	
IDL2 (A1, A2)	14	12	(A1)←(((A2)))	
PDWO	237	12	インストラクション・カウンタ (STEP) の指す要素をスタックにプッシュダウン	IF2, GMP2, BMP2, BMA1, LUKN, IFA1, IFA2, CTOC, BMP1, LGTH
CAN (A1)	19	11	スタック上の ((A1)) が一時変数ならばキャンセル	BMA1, IDL2
ADD (A1, A2, A3)	36	11	BCD コードの (A1)4+(A2)4+(A3)4 を実行し，結果を (A1)4 に格納. 桁上りがあれば (A3)←1 なければ (A3)←0	
GMA1 (A1, A2, A3, A4, A5)	34	10	A2 をマスクとして (A1)≠(A3～A4) ならば A5 にとぶ	
BI (A1)	24	10	(A1)8 の指数 BCD コードを 2 進数に変換して (A1) に格納	
ATR (A1)	57	10	スタック上 (A1) 番地にあるエントリの属性番地を (REG1) にもどす	SETP, BMP1
LGTH (A1)	208	7	属性番地が (A1) にあるデータの全長を (A1) にもどす	IFA1, BMA1
OP2	83	4	スタック上位 3 要素で 2 項演算処理をして，上位 3 要素をポップアップし，結果をプッシュダウン	BMA1, IFA1, CAN, LUK1, POW2
BCDE (A1)	21	4	(A1) の 2 進数を指数 BCD コードに変換して (A1) に格納	
STOR (A1, A2, A3)	17	3	A1 を (A2)～(A3) 番地に格納	
BGA1 (A1, A2, A3, A4)	27	2	(A1)≠(A2～A3) ならば (A4) にとぶ	
BI1 (A1)	105	2	A1 番地を先頭とする 3 語の数値データを 2 進コードに変換して (REG1) にもどす. 但し正の整数でない時は (REG1)←$FFFF_{16}$	CTOC, BI, SHR, SHL
ADD1	335	2	(REG3～REG1) と (REG6～REG4) にあるそれぞれ 3 語の数値データの加算をして結果を (REG3)～(REG1) に格納	CTOC, BMA1, BI, SHL, IFA1 SHR, ADD, CRS, BCDE, SUBT, HOSU
LFW1 (A1, A2, A3)	19	2	A3 番地から高い番地に向かって走査し，A2 と異なる内容に最初に出合う番地を (A1) に格納	
IF1 (A1, A2, A3)	16	2	(A1)≥(A2) ならば A3 にとぶ	
PDW2 (A1, A2)	18	1	スタックに A1, A2 の 2 語データをプッシュダウン	
NAM	8	1	ソース文の名前の部分をよみとばす	BGP1

エレメント名と引数	語数	頻度 ×10³	機 能	使用エレメント
ATRB (A1)	132	1	(STEP) の指す要素の属性を (A1) にもどす	BMP1, IDL2, LUKN, IFA1, BMA1
LBK1 (A1, A2, A3)	20	1	A3 番地から低い番地に向かって走査し，A2 と異なる内容に最初に出合う番地を (A1) に格納	
OPE2 (A1)	476	1	A1 をエントリ・ポイントとする基本演算子ルーチンを使って2項演算をする	ATR, IFA1, GMA1
LUKN (A1)	65	1	A1 番地を先頭にパックされた名前と名前表を照合して，そのエントリ・ポイントを (REG1) にもどす．なければ (REG1)←$FFFF_{16}$	
SYCH	725	1	ソースの演算式に関する文法チェック	IFP2, BGP2, SYCH, IFA1, IFP1, SUCH, BGP1, NAM
OP1	74	1	スタック上位2要素で2項演算処理をして，上位2要素をポップアップし，結果をプッシュダウン	BMA1, IFA1, CAN, LUK1, PDW2
BMA2 (A1, A2, A3, A4, A5)	28	1	A2 をマスクとして (A1)=A3 ならば A4 の処理をして A5 にとぶ	
IDS2 (A1, A2)	14	1	(((A1)))←(A2)	
ADRN (A1, A2)	57	1	A1 を属性番地とするデータに関して，A2 番地に格納された添字に対応する数値データの先頭番地を (REG1) にもどす	
HOSU (A1, A2)	31	1	BCD コードの 9−(A1)4+(A2)4 を実行し，結果を (A1)4 に格納．桁上りがあれば (A2)←1，なければ (A2)←0	
PAKN (A1)	54	1	(A1) 番地に始まる内部コードの名前をパック	SETP, BGP1
MULT	262	1	(REG9〜REG7) と (REG6〜REG4) の数値データの積を (REG3〜REG1) に格納	BMA1, BI, SHL, SHR, CRS, BCDE
PDW3 (A1)	13	1	スタックに A1 をプッシュダウン	
ADBC (A1, A2, A3)	44	1	BCD コードで (A1)+(A2)+(A3) を実行して，結果を (A1) に格納．桁上りがあれば (A3)←1，なければ (A3)←0	
BCD2 (A1)	26		(A1) を正の2進数として数値データに変換して (REG3〜REG1) に格納	SHR
BGA2(A1, A2, A3, A4, A5)	32		(A1)=(A2〜A3) のいずれかが成立てば A4 の処理をして A5 にとぶ	
INCR	39		添字を一つ進める	
SROP	15		(REG1) の演算子シンボルを中間表現に変換して (REG1) にもどす	
ICBN	12		文番号を一つ進める	ADBC
BI2 (A1)	31		A1 番地を先頭とする数値データを2進数に変換して (REG1) にもどす 整数でなければ (REG1)←$7FFF_{16}$	CTOC, BI1, IFA1
LUK1	46		(STEP) を関数領域または中間表現領域の先頭番地からの変位として (REG1) にもどす	SETP, BMP1, GMP1
ADSE	162		関数定義において一つの APL 文を作業領域に追加もしくは挿入	LBK1, CTOC, GRB3
FUCH	108		関数文の文法チェック	IFP2, IFP1, BGP1, SYCH, NAM, IFA1
LUK2	42		スタックの END マークの継続情報から継続すべき要素の位置を (REG1) にもどす	SETP, BMP1, GMP1
PDW1	29		スタックに NULL データをプッシュダウン	
ADRC (A1, A2)	60		A1 を属性番地とするデータに関して，A2 番地に格納された添字に対応する文字データの番地を (REG1) にもどす	
CFST	64		関数スタックのポップ・アップ	IFA1, BMA2, STOR
DIV	222		BCD コードの数値データ (REG6〜REG5) を (REG9〜REG7) で割って，結果を (REG3〜REG1) に格納	BMA1, BI, SHL, SUBT, CRS, SHR, BCDE
SUCH	61		APL ソース文の添字に関する文法チェック	IFP2, SYCH, IFA1, IFP1
OPE1 (A1)	104		A1 番地をエントリ・ポイントとする基本演算子の単項演算処理	ATR, IFA1, BMA1
COM1	543		コマンドの文法チェック	LFW1, IFP2, BGP1, BGP1, BGP2, NAM, STOR
GRB3	62		作業領域のガーベッジ・コレクション	CTOC
BCD (A1)	28		(A1) を正の2進数として BCD コードに変換し (A1) にもどす	
RESE	362		関数の中間表現をソース文の形に再現して作業領域に格納	SETP, ADBC, BMP1, IFA1, GMP1, BMA1, RNUM, BGA1

エレメント名と引数	語数	頻度 ×10³	機能	使用エレメント
FHCH	146		関数頭文の文法チェック	SETP, IFP2, BGP1, NAM, BGP2
IDL3 (A1, A2)	16		(A1)←((((A2))))	
RNUM (A1)	183		(A1) 番地を先頭とする数値データをソースの形式に再現	CTOC, SHL, BI, SHR, IFA1, BCD
ADRA (A1, A2)	55		A1 を属性番地とするデータに関して，A2 番地に格納された添字に対応する番地データの格納されている番地を (REG1) にもどす	
GRB1	49		データ領域のガーベッジ・コレクション	CTOC
GMA2 (A1, A2, A3, A4, A5, A6)	39		A2 をマスクとして (A1)=(A3～A4) のいずれかが成立てば A5 の処理をしてA6 にとぶ	

注意：◦ 頻度 10^3 以下については 4 捨 5 入　◦ 名前は値を代表する　◦ (××) は××番地の内容　◦ (A～B) は A 番地から B 番地にいたる各番地の内容　◦ (A1)4 は (A1) の右14ビット

参考文献

1) 長田純一，内山 昭：APL SV (新しいコンピュータ言語)，丸善 (1975).
2) Harry Katzan, Jr.: APL Programming and Computer Jechniques, Van Nostraud Reinhold Co. (1970).
3) K.E. アイバーソン著，和田 弘訳：アルゴリズムによる初等関数，共立出版 (1969).
4) Y. Amram, B de Cosnac, J. L. Granger, A. Smoucovit: An APL Interpreter for Mini-Computer, A Microprogrammed APL Machine. APL Congress 73 pp. 33～39, North-Holland Publishing Co. (1973).
5) R. Zaks, D. Steingart, and J. Moore: A firmware APL time-sharing system, Proc. AFIPS SJCC, Vol. 38, pp. 179～190 (1971).
6) Hassit A., J.W. Lageschulte, and L.E. Lyon: Implementation of a High Level Language Machine, Commun. ACM, Vol. 16, No. 4, pp. 199～212 (April 1973).
7) 元岡 達，川北 茂：ファームウェア化 APL 計算機，第 16 回情報処理学会大会予稿集，pp. 109～110 (1975).
8) 坂間保雄：APL 演算子のファームウェア化，第 17 回情報処理学会大会予稿集，pp. 297～298 (1976).
9) 森本陽二郎，中村 明，大筆 豊：APL インタプリタにおける演算処理，第 17 回情報処理学会大会予稿集，pp. 547～548 (1976).
10) 宮脇富士夫，渡辺勝正，萩原 宏：ファームウェアによる APL 会話型処理システムについて (処理方式およびシステムの構造)，情報処理学会第 16 回大会予稿集. pp. 111～112 (1975).

(昭和 52 年 5 月 10 日受付)
(昭和 52 年 8 月 19 日再受付)