JAPANESE APL LANGUAGE SYSTEM ON IBM MULTISTATION 5550

M. Udo, Y. Akimoto, S. Kaneko, T. Sanuki (*)
and M. Alfonseca (**)

(*) Tokyo Scientific Center,
Science Institute, IBM Japan, Ltd.
5-19, Samban-cho, Chiyoda-ku, Tokyo, Japan
(**) IBM Madrid Scientific Center
P. Castellana, 4. Madrid-1, Spain

ABSTRACT

This paper exemplifies the national language support of a programming language by describing the implementation of APL with a language facility for processing Japanese characters: that is, a character set that has "wide" characters that occupy space of more than one byte. By national language support, in this paper, we mean that users are enabled to use their own national language in the data, as legal character strings of the programming language. We describe the implementation of APL that provides Japanese language support in the above sense on a personal computer called the IBM Multistation 5550. We also discuss the design principle adopted to include a new two-byte character set in APL data objects coexisting with a conventional one-byte APL character set.

1. INTRODUCTION

It is desirable, or now becoming to be requisite, that users be able to use their own national languages in programming languages: for example, in data, identifiers, and messages [1]. But to accommodate national languages other than occidental languages, such as Arabic, Chinese, Japanese and so on, programming languages should remove the present 256-character limitation that confines legitimate characters to traditional alphanumerics.

The Japanese characters consist of several thousands of ideographic characters (Kanji), adapted from Chinese, as well as two Japanese phonetic alphabets, Kana (Hira-kana and Kata-kana). Each of the Kana character sets consists of 46 different symbols, each expressing a simple sound of the Japanese syllabary, with the addition of some diacritical marks and subscripted symbols. In the Japanese writing system we use a mixture of Kanji and Kana, where Kanji is used as the root words and Kana as inflections for most purposes, as well as Arabic numbers and Roman alphabets.

Because of this complexity, we have had no reasonable way to type our own language, especially Kanji characters. In recent years, the use of Kanji in computers has greatly increased because of a breakthrough in Kanji typing methods: that is, a phonetic or Kana-to-Kanji conversion [2]. It permits the user to key a phonetic spelling from a standard typewriter keyboard using Kana, for which the computer supplies the corresponding Kanji character by looking up the Kanji dictionary stored in a diskette.

The IBM Multistation 5550 is a personal computer equipped with the Kana-to-Kanji conversion method for Kanji input in the keyboard BIOS, as well as the Kanji output functions to the screen and printer [3].

We implemented the APL system on the 5550, based on the IBM PC APL developed by the IBM Madrid Scientific Center [4], aiming at the Japanese national language support, which had been strongly required by mainframe APL users because of their need for business applications.

Our basic idea is that a Japanese character should be dealt with as a single character scalar in APL, even if it needs two bytes for encoding in memory. To implement this, we had to extend the PC APL interpreter so that a new data type for the Japanese characters could be incorporated in the system, which we now call "Japanese APL". At the same time, to take advantage of the valuable Kanji I/O functions of the 5550 DOS/BIOS, we developed the supervisor and auxiliary processors for Japanese APL.

Japanese APL on IBM Multistation 5550 was announced by IBM Japan and has been available to customers since December 1984.

In the remaining sections of this paper, we first describe the coding scheme of the Japanese characters and the design principle to include them in APL. Then we describe an implementation of the Japanese APL on 5550, with emphasis on the issues we solved based on the design principles in each component of the system: the APL supervisor, interpreter, and auxiliary processors. Finally I/O operations with the Japanese characters are discussed in terms of the difficulty encountered when they are mixed with alphanumeric characters.

2. CODING SCHEME OF THE JAPANESE CHARACTERS

A two-byte coding scheme, in which each letter would be identified by two successive bytes, would yield 256 by 256, or 65,536, possible codes. We use the following notation to show a two-byte code:

|D1D2|

where D1 and D2 are the first and the second byte values, respectively, in hexadecimal representation of the two-byte code.

There exist two representative coding schemes for the Japanese characters:

- IBM Kanji Code : an extension of EBCDIC
- Shift JIS Code : an extension of ASCII

The former is used on mainframes while the latter is used on personal computers; IBM 5550 is architecturally based on the latter, where the range of D1 is X'81'-X'9F' and X'E0'-X'FC', and the range of D2 is X'40'-X'FC' except X'7F'. Therefore the valid two-byte codes are shown in Fig. 1 as the points in rectangles. There are 11,280 possible two-byte codes that can be handled by this coding scheme, in addition to the remaining 196 one-byte codes.

The characteristic of this scheme is that the range of D1 is reserved for the first byte of two-byte codes in a one-byte code table as shown in Fig. 2. Then to encode a mixed string of one-byte and two-byte codes, no special control character is needed, such as Shift-In or Shift-Out on the mainframe.

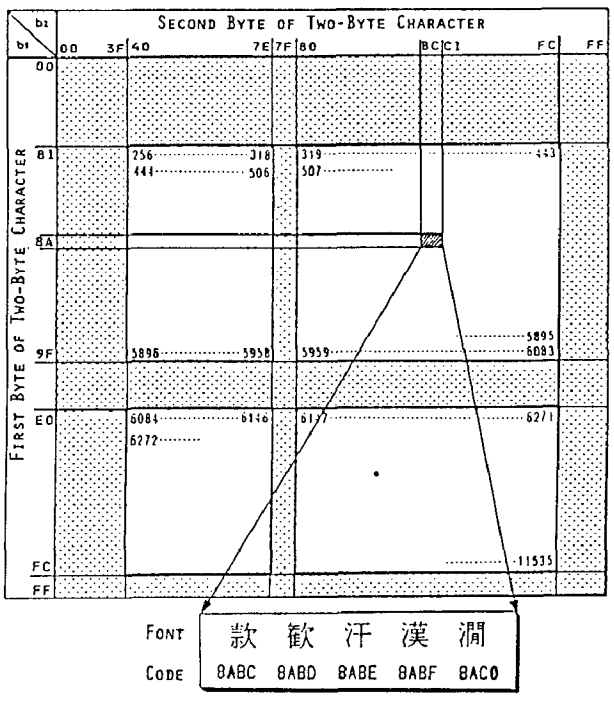

Figure 1. Valid Two-Byte Codes of IBM 5550

For example, '日本語APL' ('ni-hon-go-APL') is encoded on 5550 as:

X'93FA967B8CEA41504C'

that can be easily interpreted by software as:

```
|93FA|967B|8CEA|41|50|4C|
 ——— ——— ——— — — —
  日   本   語  A  P  L
(ni) (hon) (go)
```

because X'93', X'96', and X'8C' are valid first bytes of Kanji, but X'41', X'50', and X'4C' are not. The Japanese APL uses this coding scheme as the external representation of characters: i.e. keyboard input codes, records in DOS files, output string to the screen and printers.

3. DESIGN PRINCIPLES OF JAPANESE APL

To introduce the Japanese characters into APL, we adopted the following three principles;

1. A Japanese character, even if it is encoded by two bytes, should be a character scalar, not a two-element character vector.

2. Japanese characters should be supported as elements of a literal constant: i.e. a character vector notation represented by a list of characters enclosed with quotes as follows:

```
A← '日本語'  (Japanese characters only)
B←'日本語APL' (a mixture of Japanese and APL)
```

3. Any APL primitive function defined on character arrays should be applicable to arrays that include Japanese characters as well; e.g.

```
      'A'='日本語APL'
0 0 0 1 0 0
      '本'='日本語APL'
0 1 0 0 0 0
      ρ'日本語APL'
6
```

To maintain the standard of the APL Language, we did not introduce any new primitive function or system function to manipulate Japanese characters specifically.

The most significant design choice was, according to the first principle, whether a Japanese character should be handled as:

- a two-element character vector, consisting of two successive one-byte characters

or

- a character scalar, even though it has to be coded by two bytes.

If we had selected the former, the APL interpreter need not have been modified; because 1) the character set remains as collection of all-possible 256 one-byte codes and 2) users identify the Japanese characters in a sequence of one-byte codes. In this case, Japanese characters are encoded by a "byte-wise representation" in memory, e.g.

93	FA	96	7B	8C	EA	41	50	4C
——	——	——	—	—	—	—	—	—
日		本		語		A	P	L

Bit 4 - 7

Bit 0 - 3	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p				ｰ	ﾀ	ﾐ		
1	SOH	DC1	!	1	A	Q	a	q			｡	ｱ	ﾁ	ﾑ		
2	STX	DC2	"	2	B	R	b	r			｢	ｲ	ﾂ	ﾒ		
3	ETX	DC3	#	3	C	S	c	s			｣	ｳ	ﾃ	ﾓ		
4	EOT	DC4	$	4	D	T	d	t			、	ｴ	ﾄ	ﾔ		
5	ENQ	NAK	%	5	E	U	e	u			・	ｵ	ﾅ	ﾕ		
6	ACK	SYN	&	6	F	V	f	v			ヲ	ｶ	ﾆ	ﾖ		
7	BEL	ETB	'	7	G	W	g	w			ｧ	ｷ	ﾇ	ﾗ		
8	BS	CAN	(	8	H	X	h	x			ｨ	ｸ	ﾈ	ﾘ		
9	HT	EM	)	9	I	Y	i	y			ｩ	ｹ	ﾉ	ﾙ		
A	LF	SUB	*	:	J	Z	j	z			ｪ	ｺ	ﾊ	ﾚ		
B	VT	ESC	+	;	K	[	k	{			ｫ	ｻ	ﾋ	ﾛ		
C	FF	FS	,	<	L	¥	l	|			ｬ	ｼ	ﾌ	ﾜ		
D	CR	GS	-	=	M	]	m	}			ｭ	ｽ	ﾍ	ﾝ		
E	SO	RS	.	>	N	^	n	~			ｮ	ｾ	ﾎ	ﾞ		
F	SI	US	/	?	O	_	o				ｯ	ｿ	ﾏ	°		

Mesh pattern area represents the first byte of two-byte code character.

Figure 2. One-Byte Codes of IBM 5550

But we selected the latter. We extended the APL interpreter by adding a new data type for Japanese characters to the current four internal data types: boolean, integer, floating point, and one-byte character. The extended interpreter identifies Japanese characters as legitimate elements of an extended character set consisting of 256 APL characters and 11,280 Japanese characters. In this case, Japanese characters are encoded by a "word-wise representation" in memory: e.g.

93FA	967B	8CEA	4100	5000	4C00
日	本	語	A	P	L

The rationales for our selection are as follows:

1. To preserve the applicability of primitive functions, and hence idioms.

   The difference of the data shape between scalar and vector severely affects the applicability of primitive functions. If we handle a Japanese character as a two-element vector,

   ```
   '日本語APL' ≠ '語'
   ```

   can not be applied because of a LENGTH ERROR. But if a Japanese character is a character scalar, the above statement becomes valid and the following idiom that deletes the specified character Y in the character vector X can be applied without caring whether either X or Y includes Japanese characters or not, as follows:

   ```
         X ← '日本語APL'
         Y ← '語'
         (X≠Y)/X
   日本APL
   ```

2. To avoid the disruption of the pairing of two successive codes of a Japanese character.
   The sequence of codes in a pair of the first and the second byte of a Japanese character is not commutative.

   |8C|EA| defines 語('go')

   but

   |EA|8C| defines 鼬('yu')

   The APL language has so many primitive functions to rearrange the structure of data: Reshape, Ravel, Reverse, Rotate, Catenate, Transpose, Take, Drop, Indexing, and so on. The sequence of codes representing Japanese characters has to be exposed to these functions. If we handle a Japanese character as a two-element character vector, the pairing of codes is easily broken apart.

   ```
   ⍎ '日本語APL' ⟷⍎|93|FA|96|7B|8C|EA|41|50|4C|
                 ⟷ |4C|50|41|EA|8C|7B|96|FA|93|
                   ─  ─  ─  ──  ─  ──  ─
                   L  P  A  鼬  {  愈
   ```

   while if we use an extended character set,

   ```
   ⍎ '日本語APL' ⟷⍎|93FA|967B|8CEA|4100|5000|4C00|
                 ⟷ |4C00|5000|4100|8CEA|967B|93FA|
                   ──── ──── ──── ──── ──── ────
                   L    P    A    語   本   日
   ```

3. To avoid the code conflict in the range of the second byte of a Japanese character.

   As shown in Fig. 1 and Fig. 2, the valid D2 range (X'40'-X'FC' except X'7F') overlaps most of the one-byte APL characters, even with the range of D1. This conflict causes the following problems if a Japanese character is handled as just a two-byte sequence of one-byte codes.

   ```
   '日本語APL' ∈ '{'
       ⟷ |93|FA|96|7B|8C|EA|41|50|4C| ∈ |7B|
       ⟷  0  0  0  1  0  0  0  0  0

   5 ↓ '日本語APL'
       ⟷ 5 ↓ |93|FA|96|7B|8C|EA|41|50|4C|
       ⟷ |EA|41|50|4C|
           ──── ─  ─
           鷲   L  P
   ```

To resolve these problems in the scheme that a Japanese character is a two-element vector, users have to check each byte in the target string to find whether it is the second byte of a Japanese character or a one-byte APL character. This would overburden APL programs. On the other hand, in the scheme that a Japanese character is a scalar, users need not care about the codes of any characters.

```
'日本語APL' ∈ '{'
      ⟷ |93FA|967B|8CEA|4100|5000|4C00| ∈ |7B|
      ⟷ 0    0    0    0    0    0

5 ↓ '日本語APL'
      ⟷ 5 ↓ |93FA|967B|8CEA|4100|5000|4C00|
      ⟷ |4C|
          —
          L
```

4. IMPLEMENTATION

To realize the Japanese language support, following the design principles described above, we implemented the APL system the IBM Multistation 5550, based on the IBM PC APL.

The modules of the IBM PC APL are well organized in three different parts: 1) APL supervisor, 2) APL interpreter, and 3) Auxiliary processors. Based on this, we developed the Japanese APL on 5550 as follows.

4.1 APL Supervisor

The APL supervisor is the machine-dependent interface between the APL interpreter and the DOS operating system that manages connection, initialization, and disconnection. The main functions are 1) I/O from the keyboard, display, and printer, and 2) file management. To enable I/O of Japanese characters, we developed a new APL supervisor in Macro Assembler using 5550 DOS/BIOS interrupts. When developing the modules, we had to solve the following issues:

1. the APL character code assignment and fonts

The 5550 BIOS for screen-video I/O and printer I/O uses the code table defined in Fig. 1 and Fig. 2 to display or print a character. As for one-byte codes in Fig. 2, there is no room to add APL characters in the table. We decided to replace the one-byte Kata-kana characters located at X'A1' through X'DF' by the APL characters, as shown in Fig. 3, considering that there also exists the code conflict between APL and one-byte Kata-kana characters even on the mainframes.

On 5550, the data of character fonts are not stored in the ROM but, fortunately, they are loaded into an area of the read/write memory at system initialization time. Once loaded, therefore, user programs can define other character fonts even in text mode. The APL supervisor replaces the fonts of one-byte Kata-kana characters by APL fonts when APL is invoked by DOS and restores them when it exits to DOS.

An APL system must provide the atomic vector (⎕AV) containing all 256 possible characters, ordered by ascending value of their binary representation in the system. We arranged the characters in the atomic vector as shown in Fig. 4, maintaining compatibility with IBM PC APL. The supervisor translates a code between the outside and inside of the APL system, as shown in the following example;

	A	←	'	日	本	語	A	P	L	'
outside	41	DE	27	93 FA	96 7B	8C EA	41	50	4C	27
inside	3E	01	88	B3 DA	B6 FB	AB CA	3E	4D	49	88

based on the tables in Fig. 3 and 4.

2. the screen management

Japanese characters are physically twice as "wide" on the screen as the alphanumeric characters, for they need a 24-by 24- and 24-by 12-pixel font respectively. Therefore the 5550

screen can display 25 lines of 40 Japanese characters, or 25 lines of 80 alphanumerics. But since the 25th (bottom) line is reserved by the BIOS, user programs are allowed to use only 24 lines.

It might not be a coincidence that there exists a one-to-one correspondence between the number of bytes of a string in a text buffer and that of columns on the screen to display it. In other words, a two-byte character in the memory is a two-column character on the screen, and a one-byte character is a one-column character, of course.

9 bytes in memory 93 FA 96 7B 8C EA 41 50 4C

9 columns on screen

日	本	語	A	P	L

Just as a mixture of two- and one-byte characters in memory, so a mixture of two- and one-column characters on screen causes the screen management complex: 1) never to insert a character into the middle of a two-column character, 2) never to delete the second column (the right half of a Japanese character) only, or to delete both columns if the first column (the left half of a Japanese character) is deleted, 3) the movement of a cursor must be controlled not to be located on the second column of a Japanese character, and so on. The APL supervisor takes care of all the above. It also supports printing the line just displayed on the screen as a session-log, i.e. echo printing. The combination of the Ctrl- and PrtSc-keys controls the echo printing mode.

4.2 APL interpreter

The Japanese APL interpreter enhances the PC APL interpreter version 1.0 to include:

1. A new data type: two-byte characters, according to the Japanese character support proposed in

Bit 4 - 7

Bit 0 - 3	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p				⊬	⌶	⊤		
1	SOH	DC1	!	1	A	Q	a	q			→	⍞	⍺	⍫		
2	STX	DC2	"	2	B	R	b	r			⍐	∈	⊂	⍀		
3	ETX	DC3	#	3	C	S	c	s			⍒	≤	⍵	⊣		
4	EOT	DC4	$	4	D	T	d	t			⍋	⍕	⌈	⊢		
5	ENQ	NAK	%	5	E	U	e	u			≡	≥	↓	≠		
6	ACK	SYN	&	6	F	V	f	v			⍓	~	⍳	∨		
7	BEL	ETB	'	7	G	W	g	w			⍎	∇	¨	○		
8	BS	CAN	(	8	H	X	h	x			⍂	∆	⍈	⍁		
9	HT	EM	)	9	I	Y	i	y			⍏	⍌	⍇			
A	LF	SUB	*	:	J	Z	j	z			⍟	⊥	⋄			
B	VT	ESC	+	;	K	[	k	{			⍱	⊃	∪	≠		
C	FF	FS	,	<	L	¥	l	|			⍝	⌊	⍎	∧		
D	CR	GS	-	=	M	]	m	}			⍤	⍴	÷	↑		
E	SO	RS	.	>	N	^	n	¯			⍲	⋆	×	←		
F	SI	US	/	?	O	_	o				⍙	∩	⍟	⍋		

Mesh pattern area represents the first byte of two-byte code character.

Figure 3. APL I/O Interface Codes

Bit 4 - 7

Bit 0 - 3	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	⋆	⍋		C	S	h	x	.	≡					SYN	`
1	←	×	⍒	⊂	D	T	i	y	CR	≤					SOH	@
2	→	!	⍱	⊃	E	U	j	z	LF	⍞					STX	"
3	↑	|	⍎	⍟	F	V	k	⍙	BS	⍒					ETX	#
4	↓	⌈	⍝	⍺	G	W	l	_	SP	⍈					EOT	$
5	∈	⌊	⍈	⍵	H	X	m	¯	TAB	⍇					ENQ	%
6	⍳	=	⍫		I	Y	n	0	⍞	⋄					ACK	&
7	⍴	≠	⍫		J	Z	o	1	⍞	⊣					BEL	DC4
8	,	>	⌶	;	K	⍙	p	2	'	⊢					DLE	NAK
9	?	≥	⊥	]	L	a	q	3	⍵						DC1	ETB
A	~	<	⊤	[	M	b	r	4	∇						DC3	DC2
B	○	≤	≠	)	N	c	s	5	⍱	US					VT	{
C	+	⍲	⊬	(	O	d	t	6	¨	*	∪				FF	|
D	-	⍱	/	:	P	e	u	7		¥				CAN	SO	}
E	÷	∧	\	A	Q	f	v	8	RS	^				EM	SI	DEL
F	⍟	∨	∩	B	R	g	w	9	⌶					SUB	FS	ESC

Mesh pattern area represents the first byte of two-byte code character.

Figure 4. APL Internal Codes

the previous chapter (Design principles)

2. some other improvements and extensions, such as the maximum number of elements in APL objects of about 64K.

There exist several implementation methods for the internal representation of texts consisting of a mixture of two- and one-byte characters, such as the Japanese characters and alphanumerics [5]. One of the typical methods, mostly used on mainframes, is to separate two different character sets by the Shift-Out (X'0E') and Shift-In (X'0F') codes as follows:

0E 4562 4566 487E 0F C1 D7 D3

SO	日	本	語	SI	A	P	L

where characters are coded using the IBM Kanji Code. It seems difficult for APL to adopt this method as the internal representation, for APL treats data with any shape. What would happen if a user makes a 3-by-2 matrix from the above text vector?

On most personal computers, as the Shift JIS Code is used, the byte-wise representation shown below is efficient enough to store a mixed string in memory:

B3	DA	B6	FB	AB	CA	3E	4D	49

日 本 語 A P L

because a shift code is, so to speak, embedded in the first byte of each character. But this method is also difficult for APL to adopt, because the Japanese APL wants to process both two- and one-byte characters in a uniform way: i.e., each may be a scalar.

We adopted the word-wise representation to store a character data object that includes Japanese characters in a workspace. In this form, a byte-character is normalized to a word-character (two bytes) by appending a byte value X'00' as follows;

B3DA	B6FB	ABCA	3E00	4D00	4900

日 本 語 A P L

Consequently, the Japanese APL interpreter has two types of characters, just like two types of numbers, i.e. integer and floating point, in a conventional APL system. Figure 5 shows the internal data format for an APL data object, where the data type of a word-character has been added to support the Japanese character set.

All the primitive functions and system functions applicable to character objects were enhanced so as to be applicable to word-character objects as well. These new word-characters are also accepted as an element of in-line comments in defined functions, but not allowed in the names of variables, defined functions, and workspaces. The enhancement of the APL interpreter was first done to the IAPL interpreter written in IL (Intermediate Language) [4], and second, it was compiled into 8086 Macro Assembler codes.

We have to mention the output format of a mixed-character array. When a character array con-

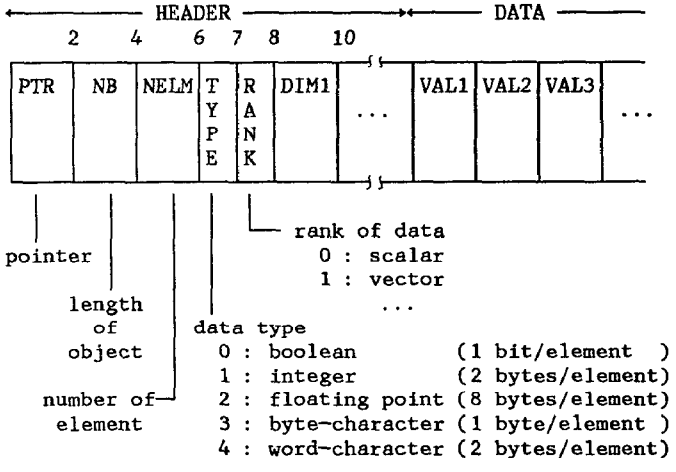

Figure 5. APL Data Format

sists of Japanese characters (i.e. two-column characters) and alphanumerics (i.e. one-column characters), the APL interpreter formats it as follows and pass it to the supervisor, which physically displays it on the screen.

```
      2 3⍴'日本語APL'
```

日	本	語
A	P	L

```
      ,2 3⍴'日本語APL'
```

日	本	語	A	P	L

If all the elements in a column are representable as one-column (one-byte) characters, the output editor displays them as a single column; otherwise, the whole column is displayed double-width, so as to maintain column alignment of APL objects.

5. I/O OPERATIONS

For users to perform peripheral I/O operations from an APL workspace, the 5550 Japanese APL provides the following auxiliary processors (AP);

- AP80 : Printer control,
- AP205 : Full-screen display management,
- AP210 : DOS file management,
- AP232 : Asynchronous communications,
- AP555 : Kanji data conversion,
- AP100 : BIOS/DOS interrupt handling, and
- AP440 : Music generator.

The first five auxiliary processors have been extensively enhanced to support Kanji data so that users can easily develop their own Kanji application programs by using a full-screen, printer, DOS files, etc.

We describe here, not the functions performed by these APs, but the issues which arose from the handling of two-byte or two-column characters. A common problem to the I/O for printer, screen, file, and asynchronous communication was how to measure the number of columns or the number of bytes necessary to output the following mixed string A;

```
A ⟷
```

B3DA	B6FB	ABCA	3E00	4D00	4900

日 本 語 A P L

A simple solution is to write a defined function: e.g.;

```
      ∇LENGTH[0]∇
[0]   Z←LENGTH R
[1]   Z←(⍴R)++/256<⎕AV⍳R ⍝ R: Character Vector
```

that returns the value 9 when applied to A. It means that the number of elements of A is six but A includes three two-byte or two-column characters, so that three additional bytes or columns are needed to print out, to display, or to write the whole string A. But we needed a more efficient solution from the point of view of performance.

There is a convenient correspondence between the number of elements (⍴) and the number of bytes on columns for alphanumerics. This implicit rule is very efficient for programmers who have to do physical I/O operations using some APs. As we mentioned in the previous chapter, that correspondence could be maintained even in a mixed string A if it had been expressed by the byte-wise representation;

```
A ⟷
```

B3	DA	B6	FB	AB	CA	3E	4D	49

日 本 語 A P L

Since this is not the case, we decided to implement an auxiliary processor, AP555, that performs character data type conversion from word-wise representation (or word-character) to byte-wise representation (or byte-character), and vice versa as shown in Fig. 6. It also provides some other functions, such as:

1. Providing the data type (type 3 or type 4) of the character vector and the length in bytes if it is type 4.

2. Mapping between word-characters and the corresponding sequence numbers defined by the Shift JIS Code table.

Now a programmer who wants to write an 80-byte long record can use a popular idiom;

```
80 ↑ A
```

after converting A from type 4 to type 3.

6. CONCLUDING REMARKS

After observing the user experience of one year in programming the Japanese APL on 5550 since the first shipment to customers in December 1984, we believe that our APL implementation supporting the Japanese characters is well accepted, and users are going to enlarge their Kanji applications on 5550. Furthermore, we announced the Japanese APL Version 2.0 last September, which runs on a variety of Japanese personal computers such as 5550, 5540, 5560, and JX.

APL2, on the other hand, has implemented the concept of extended characters for DBCS (Double-Byte Character Set) support, such as Kanji, in the mainframe environment [6]. Its aim is supporting multi-national languages simultaneously, while the Japanese APL supports only Japanese.

Type = 4 : word-wise representation

PTR	NB	NELM	TY	RA	DIM1	日	本	語	A	P	L
	22	6	4	1	6	B3DA	B6FB	ABCA	3E00	4D00	4900

Type = 3 : byte-wise representation

PTR	NB	NELM	TY	RA	DIM1	日		本		語		A	P	L
	20	9	3	1	9	B3	DA	B6	FB	AB	CA	3E	4D	49

Figure 6. Two Character Data Formats

We hope the approach we adopted for the national language support in 5550 APL will be applicable to other languages that also need "wide" characters on the 5550, such as the Chinese and Korean languages, and in the near future, we will be able to realize the concurrent multi-national language support on personal computers.

REFERENCES

[1] SigAPL Workshop on Character Sets and Keyboards (Asilomar, 1985-2-11 / 14), APL Quote Quad, Vol. 15 No. 3, March 1985

[2] J. D. Becker, Multilingual Word Processing, Scientific American, Vol. 251 No. 1, July 1984, pp. 96-107

[3] R. Willis, Big Blue Goes Japanese, BYTE, Vol. 8 No. 11, November 1983, pp. 144-163

[4] M. L. Tavera, M. Alfonseca, and J. Rojas, An APL system for the IBM Personal Computer, IBM Systems Journal, Vol. 24 No. 1, 1985, pp. 61-70

[5] K. Ushijima, T. Kurosaka, and K. Yoshida, SNOBOL4 with Japanese Text Processing Facility - Experience in Implementing a Programming Language for Handling the Large Character Set -, Proceedings of ICTP '83, October 1983, pp. 235-240

[6] APL2 Programming: Language Reference, SH20-9227, IBM Corporation, 1984