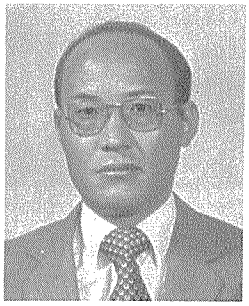


吳 吉 祿
韓國電子技術研究所
컴퓨터연구부장 / 工博

한글 정보처리 표준화 연구

"한글 데이터의 입출력 뿐만아니라 한글 정보의 문서화, 한글 통신 등을 위하여 근본적으로 한글을 처리 한글 표준화를 소개 등의 정보처리를 소개한다."

1. 서 론

우리나라에 컴퓨터가 도입된 이래 한글 처리에 대한 요구가 증가하여 한글데이터의 입출력 뿐만 아니라 한글 정보의 문서화, 한글통신 등을 위하여 좀더 근본적으로 한글을 처리할 수 있는 방법이 요구되었다. 그러나 한글은 다른 문자에서 찾아보기 드문 모아쓰기라는 독특한 표기 형태를 취하고 있어 영문자를 기초로 개발된 컴퓨터로 한글 데이터를 처리하기 위해서는 특별한 배려가 필요하다. 이에 따라 한글 입출력 장치를 비롯하여 한글 처리를 위한 연구가 계속되어 왔다. 그러나 어떤 표준이나 제약없이 제각기 한글 입출력 장치나 한글 코드 등이 만들어져 지금까지도 한글 입출력 방식과 한글 코드 등이 서로 달라 한글처리를 하는 데 많은 어려움이 뒤따르고 있다. 이런 혼란이 계속되어 오던 중 정부에서 한글 표준안을 마련하였으나 그 일부만을 통일하여 한글처리를 위한 시스템을 구성하는데 있어 그 혼란은 예나 마찬가지인 상황이다. 여기서는 한글 문제를 처음 대하는 분들을 위하여, 한글 표준화를 소개하고, 그것에 대한 우리의 의견을 제안하고 우리는 어떻게 한글처리를 수행하였으며 그 문제점은 무엇인가에 대하여 쓰고자 한다.

2. 한글 정보처리용 건반배열

컴퓨터를 위한 한글 정보처리용 건반배열의 표준화를 위하여 한국 과학기술원에서 수집한 자료[8]를 보면 지금까지 나와있는 한글 정보처리용 건반은 34개의 모델, 10개 유형으로 분류되었다. 이중 7개 유형(30개 모델)의 배열이 텔렉스의 표준건반 배열을 기본으로 하여 부분적인 변형을 이룬 것이었다. 배열된 자모는 모두 26자모(기본 24자+ㅐ, ㅔ)이상이었으며, 33개자모(복자음 5개 ㄲ, ㄸ, ㅃ, ㅆ, ㅉ 및 ㅒ, ㅖ추가)를 택한 것이 3개 유형, 24개 모델로 가장

많았다. 이를 토대로 한글 자모별 출현빈도를 감안하고 복자음의 배열 위치의 통일, 복합 자모음 입력시 한 손가락 연타가 일어나지 않도록 하는 점 등이 고려되어 한글 정보 처리용 건반 배열의 표준안[3]이 결정되었다. 이를 좀 더 구체적으로 살펴보면 다음과 같다.

1) 건반 수용문자

a. 한글 자모

-한글 자모 중 기본적으로 수용하는 문자는 다음의 26자로 한다.

ㄱㄴㄷㄹㅁㅂㅅㅇㅈㅊㅋㅌㅍㅎㅏㅑㅓㅕㅗㅛ ㅜㅠㅡㅣㅐㅔ

-그 밖의 한글 자모에 대한 수용여부는 확정하지 않는다.

b. 영문

-영문 대문자는 다음의 26자를 수용한다.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

-영문 소문자를 수용할 경우에는 다음의 26자로 한다.

a b c d e f g h i j k l m n o p q r s t u v x y z

c. 숫자 및 기호

-숫자는 다음의 10자를 수용한다.

1 2 3 4 5 6 7 8 9 0

-기호는 다음의 32자를 수용하는 것을 원칙으로 한다.

(sp) ! " # $ % & ' ( ) * + , - . / : ; 〈 = 〉 ? @ 〔 / 〕 ^ - ′ { | } ~

d. 기능문자

-수용하는 기능문자의 종류는 확정하지 않는다.

2) 건반배열

a. 한글자모

-한글 기본 자모 24자(자음 14자, 모음 10자) 및 복모음 2자(ㅐ, ㅒ)는 그림 2-1의 배열에 따른다.

-한글 자모 26자(기본 자모 24 및 ㅐ, ㅒ) 의 배열은 삭제하거나 조정할 수 없다.

-5자의 쌍자음(ㄲ, ㄸ, ㅃ, ㅆ, ㅉ)및 2자의 복모음(ㅐ, ㅒ)의 위치는 그림 2-1에서 괄호한 부분에 위치하는 것을 권장사항으로 한다.

b. 영문, 숫자 및 기호

-그림 2-2의 배열에 따른다.

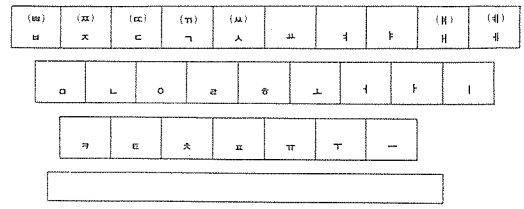


그림 2-1 한글 자모 부분의 배열

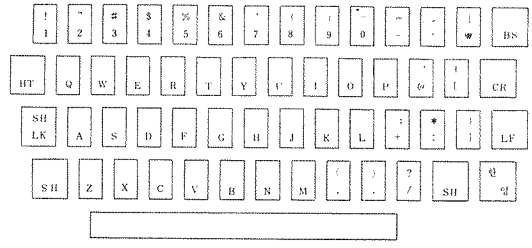


그림 2-2 영문 숫자 기호 부분의 배열

-영문자의 배열은 삭제하거나 조절할 수 없다.

-숫자의 배열은 조정될 수 있다.

-기호의 수용여부 및 배열은 조정될 수 있다.

c. 기능 문자

-기능 문자의 배열 위치는 위의 그림 2-2 에서 표시한 부분을 권장사항으로 한다는 것 외에는 위치를 확정하지 않는다.

여기서 우리가 개발한 workstation의 건반 배열을 소개하여 보면, 영문은 vt-100* 을 기준으로 하고 한글은 앞에서 설명한 한글 건반 표준안을 기준으로 하여 그림 2-3과 같이 정의하였으며 그 고려사항은 다음과 같다.

-기본 자모 24자

-복자음 5자(ㄲ, ㄸ, ㅃ, ㅆ, ㅉ)

-복모음 4자(ㅐ, ㅒ, ㅖ, ㅖ)

-단위부호(₩)

-한글, 영문 구분 표시

-그외 특수 문자

한글 기본 자모 33자는 한국 표준 자판 배열의 권장 사항인 복자음 및 복모음을 포함하여정의되었다.

3. 한글 정보교환용 코드계

이제까지 사용되어 오던 한글 코드들로는 7비

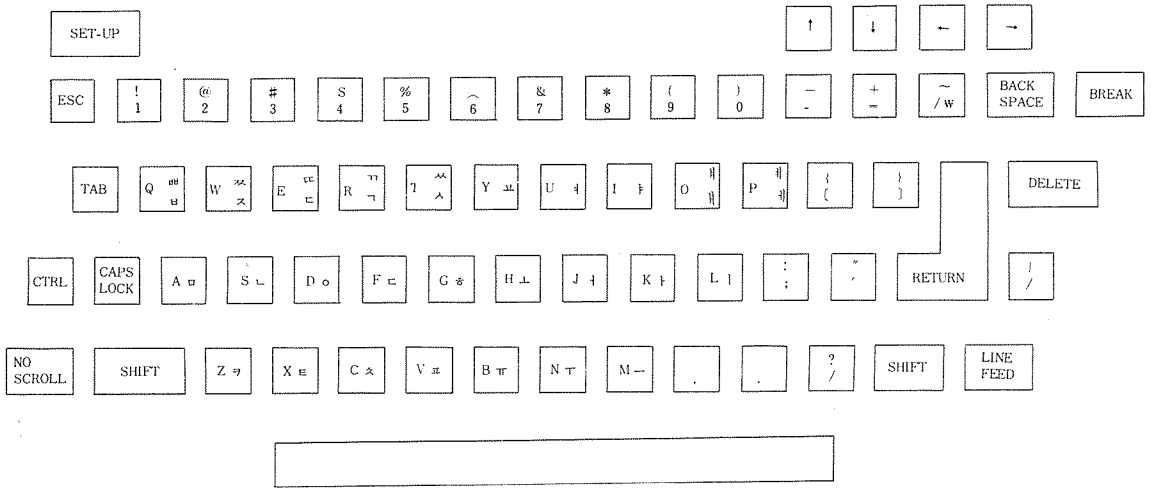


그림 2-3 개발된 Workstation의 건반 배열

트 코드, 8비트 코드, EBCDIC 코드 그리고 2바이트 코드 등이 있었다. 이를 기초로하여 한글 정보교환용 코드계의 표준화가 정해졌다. 그것의 기본 코드로는 가장 보편적으로 사용되는 7비트계를 설정하였으며 이상적인 코드계로 평가되는 보조코드계를 병용하는 것을 골자로 하는 표준화가 결정되었다. 이에 대해 좀더 구체적으로 살펴보면 다음과 같다.

1) 한글 자모는 다음의 51종(Fil을 사용할 경우 52종)으로 한다.

a. 자음 30종

ㄱ ㄲ ㄳ ㄴ ㄵ ㄶ ㄷ ㄸ ㄹ ㄺ ㄻ ㄼ ㄽ ㄾ ㄿ ㅀ ㅁ ㅂ ㅃ ㅄ ㅅ ㅆ ㅇ ㅈ ㅉ ㅊ ㅋ ㅌ ㅍ ㅎ

b. 모음 21종

ㅏ ㅐ ㅑ ㅒ ㅓ ㅔ ㅖ ㅗ ㅘ ㅙ ㅚ ㅛ ㅜ ㅝ ㅞ ㅟ ㅠ ㅡ ㅢ ㅣ

c. Fil(한글 자모 단위 빈자리 표시)

2) 기본 코드계

a. 정보 교환의 기본이 되는 코드계는 7비트로 구성한다.

b. 도형문자 코드는 로마 문자용과 한글 자모용을 두고, 단독적으로 사용하거나 병용할 수 있다.

c. 그에 따른 기능 문자코드 및 로마 문자용 코드는 표 3-1에 나타나 있다.

d. 그에 따른 한글 자모용 코드는 표 3-2에 나타나 있다.

3) 보조 코드계

a. 8비트 코드

- 8비트 코드는 기본 코드계에 제8비트의 0 혹은 1을 추가한 것으로 부록 A-1에 나타나 있다.
- 한글 자모용 코드 부분은 7비트 한글 자모음 코드에 제 8비트로서 2진수 1을 추가한 것으로 한다.

b. EBCDIC 코드

- EBCDIC 코드는 8비트로서 기능코드 및 로마문자용 도형코드, 한글 자모형 코드를 표현하며 부록 A-2에 따른다.

c. 16비트 코드

표 3-1 7단위 기능문자 및 로마문자용 코드

하\상	0	1	2	3	4	5	6	7
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	del

표 3-2 7단위 한글 자모용 코드

하/상	0	1	2	3	4	5	6	7
0						ㅀ		
1					ㄱ	ㅁ		
2					ㄲ	ㅂ	ㅏ	ㅗ
3					ㄳ	ㅃ	ㅐ	ㅘ
4					ㄴ	ㅄ	ㅑ	ㅙ
5					ㄵ	ㅅ	ㅒ	ㅚ
6					ㄶ	ㅆ	ㅓ	ㅛ
7					ㄷ	ㅇ	ㅔ	ㅜ
8					ㄸ	ㅈ		
9					ㄹ	ㅉ	ㅕ	ㅝ
a					ㄺ	ㅊ	ㅖ	ㅞ
b					ㄻ	ㅋ	ㅗ	ㅟ
c					ㄼ	ㅌ	ㅘ	ㅠ
d					ㄽ	ㅍ	ㅙ	ㅡ
e					ㄾ	ㅎ	ㅚ	ㅢ
f					ㄿ		ㅝ	ㅣ

-16비트 코드는 연속된 16비트의 코드로서, 그림 3-1에 나타난 것과 같이 모아 쓴 한글 문자를 표현한다.

15	14 10	9 5	4 0
1	초 성	중 성	종 성

그림 3-1 16비트 코드의 구성형식

-제 16번째 비트는 2진수 1로 한다.
-다섯 비트씩 구분하여 각각 초성, 중성, 종성 부분으로 한다.
-각 부분의 비트 구성은 표 3-3과 같이 7비트 한글 자모용 코드표와 같은 비트 구성을 가지게 한다.

d. 도형 문자코드(정보 교환용 한자 부호계)
-코드의 단위는 2바이트로 한다
-각 바이트는 앞에서 설명한 7비트 코드, 8 비트 코드로 한다.
-도형 문자의 종류는 특수문자, 숫자, 한글 자모음, 로마자, 그리이스 문자, 일본 문자, 한글 및 한자로 한다.
-코드의 범위는 표 3-4에 나타나 있다.
-각 도형문자의 구체적인 형태는 참고문헌7에 나타나 있다.
-선택된 한글과 한자의 수가 적은 것 같고 일어를 구태여 넣을 필요가 없는 것 같다.

표 3-3 16비트 코드에서의 부분적인 비트 구성에 해당 하는 한글 자모

b 4	b 3	b 2	b 1	b 0	초 성	중 성	종 성
0	0	0	0	0	ㄱ		없을때
0	0	0	0	0	ㄲ		ㄱ
0	0	0	1	0		ㅏ	ㄲ
0	0	0	1	1	ㄴ	ㅐ	ㄳ
0	0	1	0	0		ㅑ	ㄴ
0	0	1	0	1		ㅒ	ㄵ
0	0	1	1	0	ㄷ	ㅓ	ㄶ
0	0	1	1	1	ㄸ	ㅔ	ㄷ
0	1	0	0	0	ㄹ		
0	1	0	0	1			ㄹ
0	1	0	1	0		ㅕ	ㄺ
0	1	0	1	1		ㅖ	ㄻ
0	1	1	0	0		ㅗ	ㄼ
0	1	1	0	1		ㅘ	ㄽ
0	1	1	1	0		ㅙ	ㄾ
0	1	1	1	1		ㅚ	ㄿ
1	0	0	0	0		ㅝ	ㅀ
1	0	0	0	1	ㅁ		ㅁ
1	0	0	1	0	ㅂ	ㅞ	ㅂ
1	0	0	1	1	ㅃ	ㅟ	
1	0	1	0	0		ㅠ	ㅄ
1	0	1	0	1	ㅅ	ㅡ	ㅅ
1	0	1	1	0	ㅆ	ㅢ	ㅆ
1	0	1	1	1	ㅇ	ㅣ	ㅇ
1	0	0	0	1	ㅈ		ㅈ
1	1	0	0	1	ㅉ		
1	1	0	1	0	ㅊ		ㅊ
1	1	0	1	1	ㅋ		ㅋ
1	1	1	0	0	ㅌ		ㅌ
1	1	1	0	1	ㅍ		ㅍ
1	1	1	1	0	ㅎ		ㅎ
1	1	1	1	1			

표 3-4 도형문자 코드에서의 첫번째 코드의 범위

41	특수 문자(75자)
42	영문 7비트 코드(94자)
43	한글 7비트 코드(51자)
44	그리이스 문자, 로마 숫자(58자)
45	일본어(히라까나) (83자)
46	일본어(가다까나) (86자)
47-49	자유 영역
50-58	한자(1692자)
59-7F	자유 영역
80-FF	한글(1316자)

4. 한글 코드의 실행

먼저 앞에서 설명한 한글 코드를 실행한 한글 CRT, 프린터 등 입출력 장치들에 대하여 분석하여 보면 다음과 같다.

1) 7비트 코드의 사용

현재까지 개발된 한글 입출력 장치들은 한글을 내보내는 방법이 서로 달라 각 한글 입출력 장치끼리의 정보 교환이 불가능한 상태이다. 다음은 한글을 내보내는 방법 중 어떠한 것들이 서

로 같지 않은가를 설명한 것이다.

a. 7비트 코드를 사용할 때는 앞의 표 3－1, 표 3－2에서 알 수 있듯이 한글과 영어의 코드가 서로 겹치고 있다. 이를 구별하기 위하여 한글 구분코드가 정해져 있는 데 이 구분 코드가 한글 입출력 장치마다 달라서 큰 혼란을 빚고 있다. 또 이 구분 표시는 한글 Word Processing System, 한글DBMS, 한글 프로그래밍 언어 등 한글에 관련된 모든 응용 프로그램에서 쓰지 않는 코드가 정해져야 한다. 그러나 7비트 코드에서는 이러한 코드를 구하기 어려우므로「Escape Sequence」코드처럼 어떤 정해진 여러 코드들의 순서에 의해 그 구분 코드를 정하면 좋을 것 같다.

b. 7비트 코드 체계에서는 한글입력의 경우 ㅣ ^ 등 여러개의 특수 코드가 한글코드와 겹치기 때문에 그러한 코드의 사용이 불가능한 경우가 있다. 그래서 어떤 한글 입출력 장치에서는 한글입력의 경우에 ㅣ ^ 등의 특수 코드를 만들 수 없는 경우가 있다. 이것의 해결 방법으로 모든 한글 입출력 장치에서 다음의 순서에 의해 특수 코드를 처리해주면 된다. 즉 한글 입력의 상태일때 ㅣ ' 등이 입력되면, 영문입력을 위한 구분 코드를 출력시키고, ㅣ ^ 등의 코드를 출력시킨 다음 한글입력을 위한 구분 코드를 출력시키면 된다.

c. 건반에서 입력된 코드들이 어떻게 컴퓨터로 보내지느냐 하는 문제이다. 만약 ㄱㅏㄱㅏㄱ이라는 자소들이 건반상에서 입력 되었다면 ㄱㅏ@ㄱㅏㄱ이라고 주 컴퓨터로 보내는 방법이 있다. 즉 한 글자를 만들기 위하여 종성이 없는 글자는 Fil코드(@)로 종성을 채워주는 방법이다. 또 한 방법은 ㄱㅏㄱㅏㄱ이라고 건반에서 입력시킨 그대로 주 컴퓨터로 보내는 방법이다. 이 중에서 첫번째 방법은 3바이트씩 끊어서 주 컴퓨터로 보내기 때문에 프로그램하기는 쉬울지 모르나, 종성이 없는 글자마다 Fil 코드를 넣어야 하기 때문에 낭비되는 메모리가 많고 입출력 장치에서 부담이 많기 때문에 두번째 방법을 채택하는 것이 좋을 듯 하다.

d. 한글 한 글자를 어떻게 모아 쓰느냐 하는 방법이다. ㅗㅏㄱ과 같이 입력이 될 경우에는 한 글자가 형성이 되지 않는다. 이런 경우 ㅗㅏㄱ을 각각 내보내는 경우가 있고 '곽'처럼 글자가 되는 데 까지 모아쓰는 경우가 있다. 이럴 경우 다음과 같은 문서를 치기에는 전자의 경우는 불가능하다. 즉,「'곽'의 글자에서 초성 'ㄱ'을 뺀 '곽'은……」이라는 문장에서 글자가 되지 않는 '곽'은 내보낼 수 없다. 그러므로 입력으로 들어오는 한글은 글자가 되는 데 까지 모아쓰기를 하는 것이 좋은 방법인 것 같다.

결국 이제까지 설명한 한글 영문 구분 코드, 코드 출력 방법 등은 한글입출력 장치가 만들어질 때 통일된 조건을 가지고 있어야 하며, 이는 가장 시급히 해결되어야 할 문제이다. 한글 입출력 장치의 통일은 그 속에서 수행되고 있는 한글 Automata를 통일하면 일단 위에서 언급한모든 문제들이 해결될 것이다. 그래서 지금 우리는 여기서 우리의 경험을 바탕으로, 한글 입출력 장치에서 어떻게 한글 Automata가 통일이 되면 좋은가 그 상태도를 그림 4－1로 나타내어 한글 입출력 장치에서의 표준 Automata로 제안하고자 한다.

2) 8비트 코드 사용

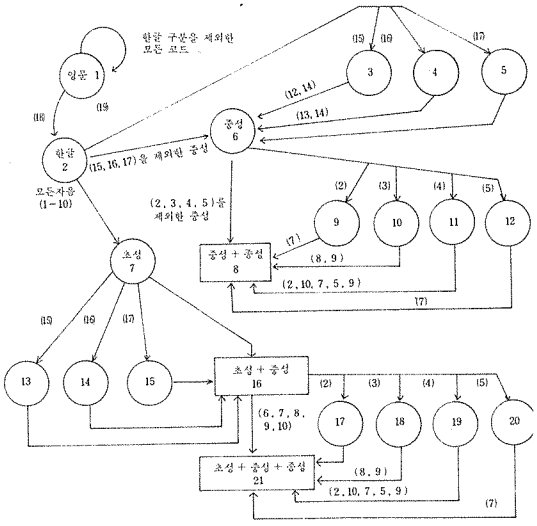
a. 8비트 코드를 사용할 때는 부록 A－1 에서 보듯이 한글 코드를 나타낼 때는 8번째 비트를 1로 한다. 그러므로 한글과 영문을 구별하는 구분 코드가 필요없이 코드 자체만으로 한글과 영문을 구별할 수 있는 장점이 있다.

b. 현존하는 거의 모든 시스템이 7비트 입출력 장치를 쓰기 때문에 8비트 코드 사용이 주 컴퓨터에 맞지 않는 경우가 있다. 특히 UNIX 시스템에서는 8번째 비트를 Parity 비트로 사용하기 때문에 8비트 코드 사용이 쉽지 않다.

3) EBCDIC 코드 사용

EBCDID 코드는 IBM 등에서 주로 사용되는 코드이므로 본 Project의 UNIX 시스템과는 크게 관계가 없는 코드이므로 긴 설명을 피한다.

4) 2바이트 코드



1) ㄸㅃㅉ 6) ㄲㄷㅆㅇㅈ 11) ㅑ ㅒㅕㅖㅛㅠ 16) ㅜ
2) ㄱ 7) ㅅ 12) ㅏ, ㅐ 17) ㅡ
3) ㄴ 8) ㅈ 13) ㅓ, ㅔ 18) 한글구분
4) ㄹ 9) ㅎ 14) ㅣ 19) 영문구분
5) ㅂ 10) ㅁㅍㅌ 15) ㅗ 20) @

그림 4-1 제안된 한글 입출력 장치를 위한 Automata 상태도

a. 한글 WPS 등 한글 소프트웨어 입장에서 보면 한글 영문 구분 코드가 필요하지 않고 한글 순서배열이 편리한 2바이트 코드가 가장 좋은 코드라고 할 수 있다.

b. 2바이트 코드를 내보내는 한글 입출력 장치를 쓸 경우 7비트 영문의 컴파일러나 DBMS 등 응용 프로그램이 수행되지 않기 때문에 모든 소프트웨어에 적용하기 힘든 코드이다.

5. 한글 Automata 디자인

앞 장에서는 이제까지 나와있는 한글 코드에 대하여 알아 보았다. 그러면 여기서 이러한 상황에서 우리가 개발한 한글 입출력 시스템에 대하여 설명하고자 한다. 먼저 우리는 영문으로 된 모든 소프트웨어들을 수행시킬 수 있게 하기 위하여 Ascii 7비트 입출력 장치를 사용했으며, 한글 WPS개발, 한글 Font Handling 등을 위해서 내부코드로는 2바이트 코드를 사용하였다. 그 내용을 그림으로 나타내면 다음 그림 5-1과 같다.

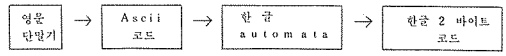


그림 5-1 한글 처리를 위한 코드

여기서 한글 2바이트 코드라 함은 그림 3-1, 표 3-3에 해당하는 코드를 뜻하며 그 특성을 다음과 같이 설명할 수 있다.

a. 초성, 중성, 종성은 각각 5비트로 구성되어 있다.

b. 모든 조합의 한글 표현이 가능하다.

c. 한글 코드 자체가 「가나다」 순으로 설정되어 있다.

d. 영문, 숫자 및 특수문자의 경우에는 첫번째는 바이트는 Hexa코드 '42'를 사용

e. 여백은 첫번째 바이트에 Hexa코드 '40'을 사용하였다.

f. 2바이트 코드의 범위를 살펴보면 표 5-1과 같다.

표 5-1 2바이트 코드 범위 그림

첫번째 바이트	두번째 바이트	내 용
40	40	여 백
41	41-FE	사용 하지 않음
42	41-FE	영문/숫자/특수문자
43-83	41-FE	사용 하지 않음
84-D3	41-FE	한 글
D4-FE	41-FE	사용 하지 않음

풀어쓴 한글, 영문, 숫자, 특수문자의 조합으로 된 입력이 들어올 때 한글 상태로 가는 구분 코드가 입력되면 한글 음소들을 모아서 한글 한 글자씩 만들어 낸다. 이러한 기능은 Finite Automata를 이용하였으며, 이는 Chomsky의 4가지 형의 문법 중 제일 취급이 간단한 Right Linear Grammar로 표시할 수 있다. 한글 모아쓰기를 위하여 한글 모아쓰기 Deterministic F Finite Automata를 구성하고 이것을 이론적으로 표시하면 다음과 같다.

1) 한글모아쓰기 Automata M

$M = (Q, \Sigma, \delta, q_0, F)$

a. 상태 Q = {Qi | i = 0, 11}
b. 입력 ∑= {건반 상의의 영문 대소문자, 숫자, 특수문자}
c. 처음 상태 qo = Qo
d. 마지막 상태 F = {Qi | i = 1,5,6,7,8}
e. Q × ∑에서 Q로 가는 함수 δ로서 이것의 상태도를 나타내면 그림 5-2와 같이 된다.

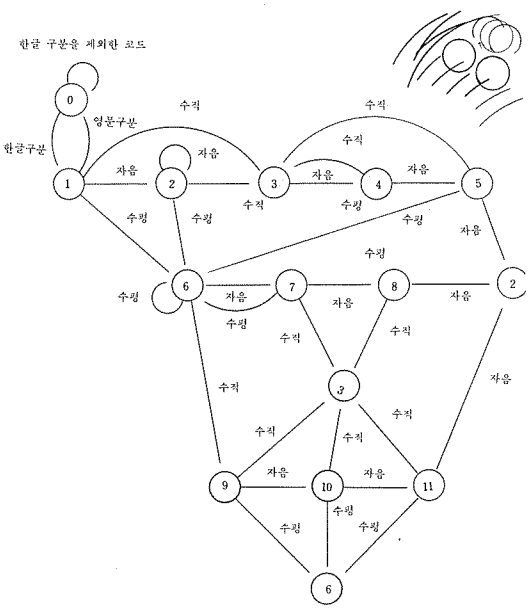


그림 5-2 한글 모아쓰기를 위한 상태도

2) 이 상태도에 대한 상태 테이블과 그 때 행하여져야 할 동작에 대한 테이블이 다음 표 5-2에 나타나 있다.

이 한글 Automata는 Bit Map 그래픽 시스템에서 이용되기 때문에 한글 조합이 가능한데 까지 조합하였다.

그리고 영문이나 특수 문자는 상태 0에서 처리되며 이때는 영문 2바이트로 처리하게 된다. 상태 0에서 한글 구분 코드가 입력되면 한글 상태가 되고 한글 상태에서 영문 구분 코드를 만나면 영문 상태로 가게 된다. 상태 1에서는 한글 내에서 사용하는 특수 문자를 처리하거나, 자음수평모음이나 수직모음에 의하여 상태 2, 3, 6으로 가게 되어 그 다음부터 2바이트 코드가 생성되어지며 상태 테이블에서 분류하였듯이 그 상태에서 일어나는 동작을 정의하면 다음과 같다.

0 : 아무런 동작이 필요없다.

1 : 영문자나 특수문자로서 입력된 글자를 바로 2바이트 코드로 변환한다.

2 : 한글 입력 상태로 들어간다.

3 : 영문 입력상태로 들어간다.

4 : 입력된 문자를 버퍼에 넣는다.

5 : 버퍼에 있는 내용을 2바이트 코드로 변환하고 영문 입력 상태로 돌아간다.

6 : 특수문자가 들어 왔으므로 버퍼의 내용을

표 5-2 상태도를 위한 테이블

\	자 음		수직 모음		수평 모음		한글 모드		영문 모드		그 외	
0	0	1	0	1	0	1	1	2	0	3	0	1
1	2	4	3	4	6	4	1	2	0	3	1	1
2	2	7	3	4	6	4	1	2	0	5	1	6
3	4	4	3	7	3	7	1	2	0	5	1	6
4	5	4	3	8	6	8	1	2	0	5	1	6
5	2	7	3	8	6	8	1	2	0	5	1	6
6	7	4	9	4	6	7	1	2	0	5	1	6
7	8	4	3	8	6	8	1	2	0	5	1	6
8	2	7	3	8	6	8	1	2	0	5	1	6
9	10	4	3	7	6	7	1	2	0	5	1	6
10	11	4	3	8	6	8	1	2	0	5	1	6
11	2	7	3	8	6	8	1	2	0	5	1	6

행하여져야 할일
다음 상태

한글 2바이트 코드로 만들고 입력된 글자도 역시 2바이트 코드로 만든다.

7 : 입력된 문자에 의하여 버퍼에 있는 문자들로서 한글 한글자를 구성할 수 있으므로 버퍼에 있는 내용을 2바이트 코드로 만들고 입력된 문자는 다음 글자를 위해 버퍼에 저장한다.

8 : 입력된 문자에 의하여 버퍼에 있는 문자 중 마지막 한 문자를 제외하고 한글 한글자를구성할 수 있으므로 그 문자를 제외한 버퍼에 있는 내용을 2바이트 코드로 만들고 입력된 다음 글자를 위해 버퍼에 저장한다.

한글은 연속된 몇 개의 문자가 모여 한 글자의 한글을 만들게 되므로, 먼저 들어온 글자를 저장하기 위한 버퍼가 필요하게 된다. 버퍼가 가진 문자의 수에 따라 그 상태를 살펴보면 다음 그림 5-3과 같다.

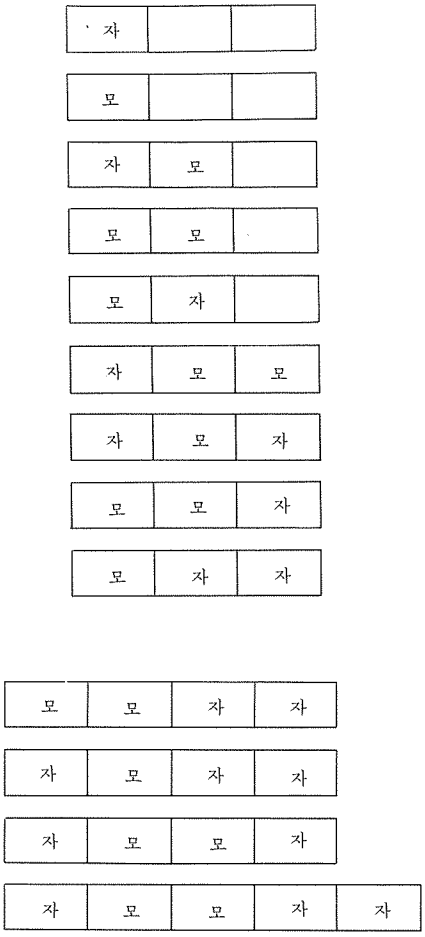


그림 5-3 버퍼의 상태

6. 한글 Font Image

한글의 Font Image를 가지는 방법에는 크게 두가지 방법으로 분류할 수 있다. 그 첫째 방법은 한글의 자모구성에 의해 한글을 몇 가지 형태로 분류하여 각 형태의 구성 자소마다 Image를 만들어 이 Image들의 조합에 의해 한 글자를 만들어내는 방법이며, 두번째 방법은 완성된 한글 한글자마다 Image를 만들어 글자를 나타내게 하는 방법이다. 첫 번째 방법에서 Font를 지정하는 방식은 코드 자체 내에 초성, 중성, 종성의 코드를 5비트씩 가지는 2바이트 코드 체계가 유리하다. 그리고 이 방법은 모두 가능한 한글을 다 포함할 수 있다는 장점이 있는 반면 코드 범위가 너무 넓어 사용하지 않는 주소가 많아 코드를 그대로 Font Image의 주소로 사용하기가 쉽지 않고, 여러 Font Image를 모아야 한 글자가 된다는 단점이 있다. 그런 반면 두번째 방법은 글자의 주소가 연속적으로 되어 있어 직접적으로 Font Image를 지정할 수 있는 장점이 있는 반면, 한 글자에 대한 코드를 정의하기가 힘이 들 뿐 아니라, 새로운 Font Image를 삽입하거나 삭제할 때 다른 Font Image를 지정하는 주소가 영향을 받는다는 단점이 있다. 뿐만 아니라 Font Image를 위한 기억공간도 다음에서 보듯이 첫번째 방법보다 더 많이 든다.

한 Font Image의 Dot크기가 24×24이라고 가정할 때 소요되는 기억공간은 3×1(8 비트)×24=72바이트가 된다. 즉 한 Font Image를 위하여 72바이트가 소요된다.

1) 첫 번째 방법

한글 문자는 초성+중성+(종성)으로 구성되어 있으며 이 때 허용되는 자소는 총 67개로 서 초성 19개, 중성 21개, 종성 27개이다. 한글의 모양에 따라 글자의 형태를 20가지(표 6-1)로 나눌 때, 필요한 Font Image의 갯수는 초성과 그 형태의 중성음 합한 것이 많아야 32개이하 (형태 0일 경우 자음+ㅏㅑㅓㅕㅣ=24)이며, 종성 또한 32개 이하이다. 그리고 20개의 형태 중 받침이 필요한 형태는 10가지이므로 필요한 Font Image의 갯수는 32(초성+그 형태의 중성)×20+32(종성)×10=960개 이하가 된다. 여기

서 한글 한 Font Image를 표시하기 위하여 72 바이트가 필요하므로 960×72=70K 바이트가 필요하게 된다.

2) 두 번째 방법

한글 문자의 가능한 총 조합의 갯수는 다음과 같으며 그 전체의 글자는 참고문헌 7에 나타나 있다.

19(초성)×21(중성)×28(종성+None)=10,172

이 중에서 약 7,000개 정도 만이 올바른 조합이 되고 약 2,000개 정도면 99.99% 이상의 사용률을 감당하고 있는 것으로 나와 있다. 그러므로 총 필요한 바이트 수는 2,000×72=144K 바이트 정도 사용된다. 정도

두 가지 방법 모두가 장단점이 있으나 본 Project에서는 조합 가능한 모든 글자를 나타내기 위하여 첫번째 방법을 사용하였다. 한글의 모양은 그 글자가 포함하고 있는 모음에 의해 결정되며 이 모음에 의한 구성글자를 세분하면 할수록 더욱 더 세련된 글자의 모양을 얻어 낼 수 있다. 본 Project에서는 모음의 형태를 다음과같이 20가지 형태로 분류하였다.

표 6-1 모음에 의한 한글의 형태 분류

수직모음	0	ㅏ ㅑ ㅓ ㅕ ㅣ
	1	ㅐ ㅒ ㅔ ㅖ
복합모음	2	ㅘ ㅝ
	3	ㅙ
	4	ㅚ
	5	ㅟ ㅞ
	6	ㅙ
수평모음	7	ㅗ ㅛ
	8	ㅡ
	9	ㅜ ㅠ

앞의 표 6-1에서 나타난 모음의 형태 10가지는 받침이 있는가 없는가에 따라 나눌 수 있으므로 총 형태는 10×2=20가지가 된다.

7. Font editor

Bit map 그래픽스 시스템에서 한글이나 영문, 특수문자를 나타내기 위해서는 각 글자마다 font image가 필요하게 된다. 이러한 Font image는 Bit map 그래픽스 시스템에서는 필수적인 것이며, Font image의 크기나 모양, 종류에 따라서 많은 image화일 들이 존재하게 된다. 우리가 I 개발한 이 Font editor는 사용자가 CRT 단말기를 이용하여 Image 화일을 만들고, 지우고,고치기 쉽게 개발되었다. 또한, 사용자가 만든 Image를 실제 그래픽화면에 나타나는 형태를 보고 고칠 수 있도록 하기 위하여 CP/M-86 시스템의 그래픽 기능이나 Tektronix 4027a 그래픽 터미널을 사용하였다.

Font editor의 구조는 그림 7-1 에서 보듯이 명령어 처리부분은 사용자가 단말기에서 입력하는 모든 명령을 받아서 필요한 조치를 취하게 된다.

이러한 명령들은 화면에 나타나 있는 Image들을 지우거나 삽입하는 Image 제어명령, Image를 화일로 저장하거나 저장된 Image 화일을 Load하거나 지우는 화일의 제어 명령, Image를 그래픽 단말기를 통해서 화면으로 볼 수 있는 명령, 편집을 끝내는 명령 등으로 구분되어 질 수 있다. 그림 7-2는 Font editor 에서의 데이터 흐름을 나타낸 것이다.

그리고 이 Font editor를 사용하는 방법은 참고문헌 7에 잘 설명되어 있다.

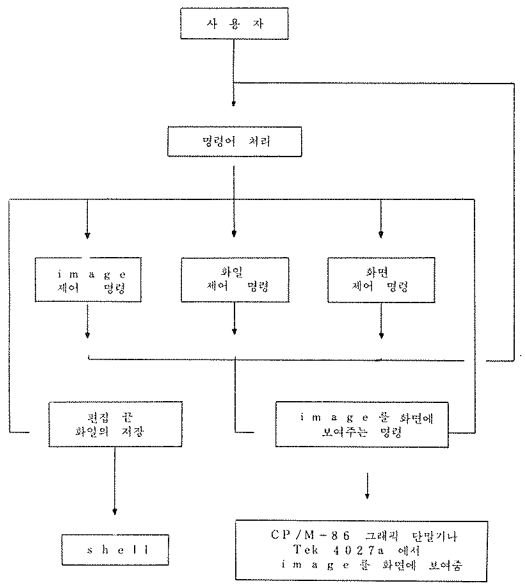


그림 7-1 Font editor 의 구조

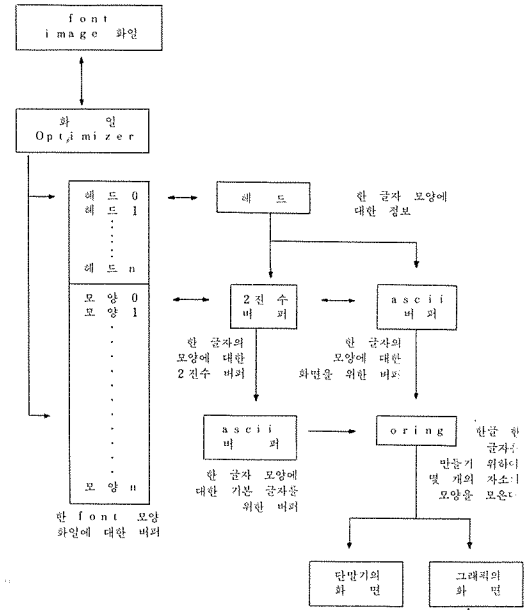


그림 7-2 font editor 의 데이타구성과 흐름도

8. Font 화일의 구조 화일의 구조

일반적으로 한글의 모양은 그 글자에 포함하어 있는 모음에 의해 결정된다. 이렇게 해서 분리된 모양들은 그 형태별로 모이게 되는게 본 Font Editor 에서는 이를 하나의 화일로 구성하였다. 그림 8-1은 이러한 형태로 분류한 화일들의 구조와 실제 각 글자의 모임인 Font Image들과 이 Image들의 정보에 관한 헤드 화일들의 관계를 나타낸 것이다. 되

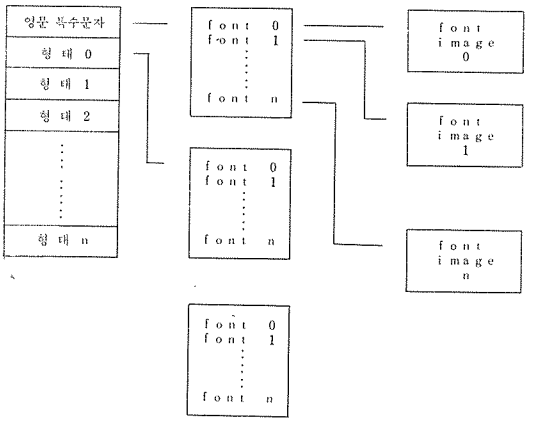


그림 8-1 font 화일의 구성도

사용자가 단말기에서 한글을 입력하게 되면 한글 Automata에 의해서 2바이트 코드가 만들어지며, 이 코드에 의해서 한글의 형태가 결정된다. 이러한 형태에 의해서 한 글자의 정보를 찾아내고 이 정보를 기초로 하여 Bit Map 그래픽스 시스템에서 글자를 보여주고, 지우고, 이동하고, 수직배열, 수평배열 등의 기능을 수행하게 된다. 그림 8-2는 이러한 한 글자에 대한 정보와 실제 Image의 관계를 나타낸 것이고, 그림 8-3은 이러한 한 글자에 대한 정보의 요소를 그림으로 나타낸 것이다.

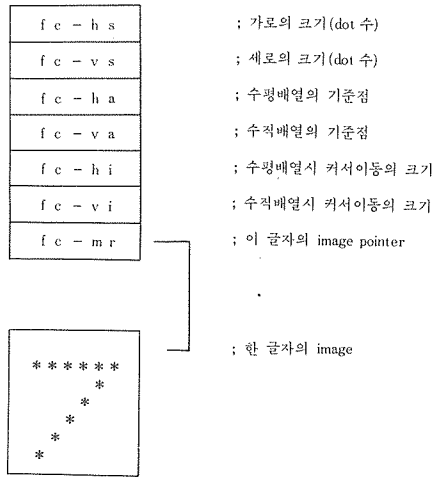


그림 8-2 글자 한자에 대한 image의 구성도

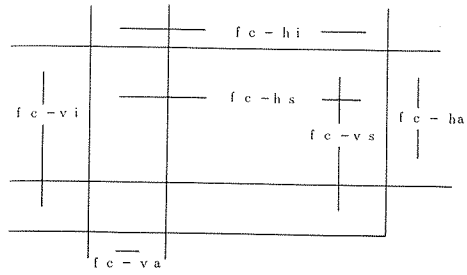


그림 8-3 한 글자의 정보 요소

그림 8-3에서 보듯이 글자의 크기는 Fc-hs * Fc-vs에 의해 결정됨으로, 사용자 마음대로 글자의 크기를 조정하여 만들 수 있으며 Fc-ha와 Fc-va에 의해 일정한 수평, 수직 배열을 할 수 있다. 또 Fc-hi와 Fc-vi에 의해 수평, 수직 배열시 글자의 크기가 다르더라도 쉽게 글자를 화면에 나타낼 수 있다. Font 화일에서 각 Ima-

ge들에 대한 Pointer나 크기는 다음과 같이 얻어진다.

```
rasterptr= (((char *) & ((fcp) -> fc-
                mr)) + ((fcp) -> fc- mr))
rastersize= (((fcp) - fc- vs) * (((fcp) - fc
                - hs) + 15) >> 4))
```

여기에서, Pointer는 어떤 글자를 구성하는 Mini-raster에 대한 Pointer이며 Size는 Mini-raster의 갯수를 말한다. 각 Mini-raster는 시스템과 Image Font의 특성을 생각하여 16비트 Words로 구성하였다.

9. 결론

이제까지 살펴본 바에 의하면 한글 정보 처리에 있어서, 처리해야 할 데이터가 한글이기 때문에 생기는 기술적인 문제들은 거의 해결할 수 있는 것들이라 할 수 있겠다. 그럼에도 불구하고 아직도 한글을 위한 응용 프로그램을 작성하는데 많은 혼란이 야기되는 것은 이제까지 나와 있는 한글을 처리하는 기본적인 방법들이 서로 통일이 되어 있지 않기 때문이다. 그러므로 빠른 시일내로 한글 정보처리를 위한 코드 및 건반에 대한 표준안 뿐만 아니라 한글 모아쓰기를 위한 Automata 등 한글 입출력 장치에서 표준화 되어야 할 것들에 대한 더욱 구체적인 표준안이 마련되어야 할 것이다. 뿐만 아니라 한글 WPS, 한글 Bit Map 처리방식, Network 등에 대한 표준안도 마련되어야 하며, 한글 Font Image에 대한 것도 표준안이 마련되어야 할 것이다.

그것이 구체적인 실현 방법으로는 한글 입출력 장치에 쓰이는 모아쓰기 기능이나 Image를 Chip이나 Board로 만들어 표준화로 유도하는 방법이 좋을 듯 하다. 앞으로의 한글 문제에 있어 더 연구 되어야 할 부분은 한글 프로그래밍 언어 개발 및 이의 표준화 작업 등이다.

참고문헌

1. 김 길창, 「한글 및 한자 문서 처리 식스템」 한국과학기술원
2. 이 기식, 한 선영, 「마이크로 컴퓨터를 이용한 한글 Word Processor에 관한 연구」
3. 「정보처리용 건반 배열」, KSC 5715, 한국 공업 규격 1982. 6. 17
4. 「정보 교환용 한자 부호계」, KSC 5619, 한국 공업 규격 1982. 10. 8
5. 「정보 교환용 부호의 확장법」, KSC 5620, 한국 공업 규격 1982. 9. 29
6. 「정보 교환용 부호」, KSC 5601, 한국 공업 규격 1982. 10. 8
7. 「사무 자동화 개발에 관한 연구」 최종 보고서, 한국전자기술연구소 1984
8. 「컴퓨터 표준화에 관한 연구」, 한국과학기술원, 1982
9. 「우리말 표준화 보고서 1, 2」, 한국과학기술정보센터, 1980
10. 「한글 한자 찾기 조사 일람표」, 한국과학기술정보센터
11. 「낱말 찾기 조사 일람표」, 한국과학기술정보센터
12. 이 수연, "An automatic generation system of bilevel document image by dynamic composition method", Kyoto Univ. 1983

부록 A-1. 8비트 로마문자 및 한글자모용 코드

				b_8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b_7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b_5	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1	열 행	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0 0	NUL	TC_7 (DLE)	SP	0	@	P	、	p	˙				(FIL)	ㅀ		
0	0	0	1	1	TC_1 (SOH)	DC_1	!	1	A	Q	a	q					ㄱ	ㅁ		
0	0	1	0	2	TC_2 (STX)	DC_2	〃	2	B	R	b	r					ㄲ	ㅂ	ㅏ	ㅛ
0	0	1	1	3	TC_3 (ETX)	DC_3	#	3	C	S	c	s					ㄳ	ㅃ	ㅐ	ㅜ
0	1	0	0	4	TC_4 (EOT)	DC_4	$	4	D	T	d	t					ㄴ	ㅄ	ㅑ	ㅝ
0	1	0	1	5	TC_5 (ENQ)	TC_8 (NAK)	%	5	E	U	e	u					ㄵ	ㅅ	ㅒ	ㅞ
0	1	1	0	6	TC_6 (ACK)	TC_9 (SYN)	&	6	F	V	f	v					ㄶ	ㅆ	ㅓ	ㅟ
0	1	1	1	7	BEL	TC_{10} (ETB)	'	7	G	W	g	w					ㄷ	ㅇ	ㅔ	ㅠ
1	0	0	0	8	FE_0 (BS)	CAN	(	8	H	X	h	x					ㄸ	ㅈ		
1	0	0	1	9	FE_1 (HT)	EM	)	9	I	Y	i	y					ㄹ	ㅉ		
1	0	1	0	10	FE_2 (LF)	SUB	*	:	J	Z	j	z					ㄺ	ㅊ	ㅕ	ㅡ
1	0	1	1	11	FE_3 (VT)	ESC	+	;	K	[	k	{					ㄻ	ㅋ	ㅖ	ㅢ
1	1	0	0	12	FE_4 (FF)	IS_4 (FS)	,	<	L	₩	l	|					ㄼ	ㅌ	ㅗ	ㅣ
1	1	0	1	13	FE_5 (CR)	IS_3 (GS)	-	=	M	]	m	}					ㄽ	ㅍ	ㅘ	
1	1	1	0	14	SO	IS_2 (RS)	.	>	N	^	n	~					ㄾ	ㅎ	ㅙ	
1	1	1	1	15	SI	IS_1 (US)	/	?	O	-	o	DEL					ㄿ		ㅚ	

비 고 : 1. 00/15 "SI"와 00/11 "SO"는 연속되는 부호군이 각각 본 부호계임을 지정하거나, 본 부호계가 아님을 지정하는 문자로 사용한다.
12/00 "FIL"은 한글자모 단위 빈자리 표시를 위한 부호로서 필요시 사용할 수 있다.

부록 A -2. EBCIC 8비트 로마문자 및 한글자모용 코드

				b_8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				b_7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1	열 행	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	NUL	DLE	DS		SP	&	-						{	}		0
0	0	0	1	1	SOH	DC1	SOS			₩	/		a	j	~		A	J		1
0	0	1	0	2	STX	DC2	FS	SYN	FIL	ㄸ	ㅀ	ㅈ	b	k	s		B	K	S	2
0	0	1	1	3	ETX	TM			ㄱ	ㄹ	ㅁ	ㅉ	c	l	t		C	L	T	3
0	1	0	0	4	PF	RES	BYP	PN	ㄲ	ㄺ	ㅂ	ㅊ	d	m	u		D	M	U	4
0	1	0	1	5	HT	NL	LF	RS	ㄳ	ㄻ	ㅃ	ㅋ	e	n	v		E	N	V	5
0	1	1	0	6	LC	BS	ETB	US	ㄴ	ㄼ	ㅄ	ㅌ	f	o	w		F	O	W	6
0	1	1	1	7	DEL	IL	ESC	EOT	ㄵ	ㄽ	ㅅ	ㅍ	g	p	x		G	P	X	7
1	0	0	0	8		CAN			ㄶ	ㄾ	ㅆ	ㅎ	h	q	y		H	Q	Y	8
1	0	0	1	9		EM			ㄷ	ㄿ	ㅇ	、	i	r	z		I	R	Z	9
1	0	1	0	10	SMM	CC	SM		₡	./	|	:	ㅏ	ㅕ	ㅛ					LVM
1	0	1	1	11	VT	CU1	CU2	CU3	·	$	,	#	ㅐ	ㅖ	ㅜ					
1	1	0	0	12	FF	IFS		DC_4	<	*	%	@	ㅑ	ㅗ	ㅝ					
1	1	0	1	13	CR	IGS	ENQ	NAK	(	)	-	'	ㅒ	ㅘ	ㅞ					
1	1	1	0	14	SO	IRS	ACK		+	;	>	=	ㅓ	ㅙ	ㅟ					
1	1	1	1	15	SI	IUS	BEL	SUB	|	ㄱ	?	„	ㅔ	ㅚ	ㅠ					EO

비 고 : 2. 기본 부호계에서 정의되지 않은 기능문자 부호는 정보 교환 당사자의 합의하에서만 사용할 수 있다.
4/2 "FIL"은 한글자모 단위 빈자리 표시를 위한 부호로서 필요시 사용할 수 있다.