정신문화연구 제41권 제4호(통권 153호) 213-238쪽 研究論文

옛한글 코드의 변천사

양왕성
한글과컴퓨터 부사장, 정보통신공학 전공
yang@hancom.com

Ⅰ. 머리말: 옛한글 코드와 인문학

세종대왕의 한글 창제 이전에 우리말이 없었던 것은 아니다. 한글이 창제되기 이전에도 이미 한문을 통해 문자 생활을 영위했으며 향찰·이두·구결 등을 통해 우리말을 적기도 했다. 다만 한문 또, 한글 이전의 여러 우리말 표기 방식은 부자연스럽거나 불편한 것이어서 극소수의 사람만이 쓰고 읽을 수 있었다. 세종대왕의 한글창제는 이 모든 부자연스러움과 불편함을 한 번에 해결한 획기적인 일이었다.

코드는 정보세계의 문자이다. 코드가 없다면 어떤 정보도 입력 및 출력을 할 수 없다. 이를 통해 보면 한글 코드를 제정하는 것은 정보세계에 한글을 창제하는 것과 같은 종류의 일이다. 물론 한글 코드 체계는 우리말 그리고 한글의 기반 위에서 만들어진다. 또 한 개인이 아니라 많은 사람들의 협동 위에서 만들어지며, 전세계가 모여 만든 규약 위에서 만들어진다. 그러나 그것이 없다면 '말하고자 하는 바가 있어도 표현할 수 없게' 되기 때문에 감히 한글 창제와 비견하는 것이다.

한글이 세종대왕의 창제 이후 많은 변화가 있었던 것과 같이 한글 코드 또한 많은 변화가 있었다. 다행스럽게도 현대 한글 11,172자의 경우 비교적 초기에 안정화되고 국제문자코드(ISO/IEC 10646, 유니코드)에까지 반영이 되었다. 이 결과로 우리는 스마트폰을 비롯한 거의 모든 전산 기기에서 한글을 무리 없이 사용하고 있다. 반면 옛한글은 아직 현대 한글만큼 자유롭게 운영할 수 있는 환경이 아니다. 상대적으로 늦게 표준화가 진행되었을 뿐더러 이를 입력하거나 출력할 수 있는 제품들도 많지 않은 상황이다. 더구나 옛한글 코드는 비교적 짧은 기간 안에 압축적인 변화가 있었기 때문에 그 변화가 비록 긍정적인 방향으로 발전한 것이었다 하더라도 여러 문제를 야기하기도 했다. 옛한글을 컴퓨터를 이용해 입력하고 가공한다면 그것 역시 코드 없이는 불가능하며 코드 체계에 지대한 영향을 받는 일이기 때문이다. 따라서 옛한글 코드의 변화한 궤적을 일별하고 그 궤적을 정리하는 일은 옛한글을 다루는 모든 인문학 분야에 요긴한 일이다.

현재 옛한글 코드를 표준화 하려는 오랜 노력이 결실을 거두어 국제문자코드에 반영되어 있다. 이는 국제적인 표준이 확립되었다는 말이다. 그간의 현대 한글코드에 대한 논란은 논외로 하고, 초기부터 옛한글을

담기 위해 어떠한 시도들이 있었는지, 또 어떠한 발전이 있었는지 특히 흔글 워드프로세서의 코드체계에 유의하며 살펴보고자 한다.

II. 한글코드 변천사

옛한글코드는 한글코드의 확장이므로 한글코드의 변천사를 같이 볼 필요가 있다. 초기 미국에서 개발된 컴퓨터는 기본적으로 로마자 알파벳 기준의 입출력으로만 설계되어 있다. 여기에 한글을 인식하게 하려면 주어진 규격에 맞게 확장해야 했다. 주어진 규격을 벗어나면 시스템의 안정성을 해칠 수 있기 때문에, 규격의 준수는 중요한 제약조건이다.

각 코드 규격의 세부 장단점은 다분히 기술적인 내용이기에 이 글에서는 논외로 하고, 한글의 지원 범위와 옛한글 관련 부분을 위주로 살펴본다.

1. 7비트 한글코드

영어권은 기본 로마자 알파벳 기준으로 운영할 수 있기에 128자만으로도 충분하다. 영어권에서는 7비트 128자(2의 7승)가 정의된 "ASCII Code"를 사용하는데, 이는 제어문자, 알파벳 대문자, 알파벳 소문자, 숫자, 괄호 문자 및 기타 기호를 포함하고 있다.

이 7비트 시스템에서 한글을 표기하기 위한 방식은, 로마자 알파벳 코드 값이 어떤 모드(mode) 전환에 따라 영어가 표기되거나 또는 한글이 표기되도록 하는 방식이다. 즉 모드를 바꾸는 특별한 제어코드 SI(Shift-In)와 SO(Shift-out)가 그것인데, SO 제어코드 다음은 한글 자소로 해석하고, SI 제어코드 다음은 다시 로마자 알파벳으로 해석한다. 당시 전보를 발송했던 전신 시설이나 메인프레임 컴퓨터들이 이 방식을 이용했으며, 애플II PC의 한글 시스템(일명 "call 3327")도 이 방식이었다. ASCII 기호들과 한글자모 맵핑(mapping)은 각 구현 시스템마다 달랐고, 아래 예는 KS로 등록된 7비트 한글코드 맵핑을 나타낸다.

한글 글자마디 하나를 나타내기 위해 여러 바이트가 필요하므로 "N바이트 한글코드" 방식으로 분류되기도 한다. 조합되지 않으면 풀어쓰기 형태로 나열되고, 기본 운용에 있어 음절 구분 등의 문제가 있는 미숙한

코드체계였다. 당시(70년대 후반-80년대 초반)로는 한글이 입출력되는 것만으로도 대단한 일이었기에 옛한글까지 고려되지 않았다.

표1-N바이트 한글코드의 예

코드값	"English 〈SO〉 ^bDAzl 〈SI〉 English"
출력	"English 한글 English"

#Note: A hangul string must be between SO (0x0E) and SI (0x0F).[1)]

표2-ASCII코드와 7비트 한글 코드 Mapping(KS)

Hex	Char	한글
0x40	@	.
0x41	A	ㄱ
0x42	B	ㄲ
0x43	C	ㄳ
0x44	D	ㄴ
0x45	E	ㄵ
0x46	F	ㄶ
0x47	G	ㄷ
0x48	H	ㄸ
0x49	I	ㄹ
0x4A	J	ㄺ
0x4B	K	ㄻ
0x4C	L	ㄼ
0x4D	M	ㄽ
0x4E	N	ㄾ
0x4F	O	ㄿ
0x50	P	ㅀ
0x51	Q	ㅁ
0x52	R	ㅂ
0x53	S	ㅃ
0x54	T	ㅄ
0x55	U	ㅅ
0x56	V	ㅆ
0x57	W	ㅇ
0x58	X	ㅈ
0x59	Y	ㅉ
0x5A	Z	ㅊ
0x5B	[	ㅋ
0x5C	₩	ㅌ
0x5D	]	ㅍ
0x5E	^	ㅎ
0x5F	_	

Hex	Char	한글
0x60	‘	
0x61	a	
0x62	b	ㅏ
0x63	c	ㅐ
0x64	d	ㅑ
0x65	e	ㅒ
0x66	f	ㅓ
0x67	g	ㅔ
0x68	h	
0x69	i	
0x6A	j	ㅕ
0x6B	k	ㅖ
0x6C	l	ㅗ
0x6D	m	ㅘ
0x6E	n	ㅙ
0x6F	o	ㅚ
0x70	p	
0x71	q	
0x72	r	ㅛ
0x73	s	ㅜ
0x74	t	ㅝ
0x75	u	ㅞ
0x76	v	ㅟ
0x77	w	ㅠ
0x78	x	
0x79	y	
0x7A	z	-
0x7B	{	ㅢ
0x7C	|	|
0x7D	}	
0x7E	~	
0x7F		

1) SO/SI, 『KS X 1001:2004』, KS, Annex 4

2. DBCS 옛한글자모

환경이 IBM-XT/AT 호환 MS-DOS 환경으로 바뀌면서, 로마자 알파벳을 쓰는 환경도 8비트 256자로 확장되었다. "Extended ASCII Code"로 확장된 이 코드시스템은 로마자 알파벳 외 서구 유럽의 프랑스, 독일 등에서 필요한 알파벳(ç, ë, ô, î, Ä, ä, Ö, ö, Ü, ü, β)까지 추가되었고, 화폐 단위와 약간의 수학기호 및 장식 기호들이 추가되었다.

그러나 "Extended ASCII Code"처럼 8비트 영역을 확장한다고 해도 128자 밖에 확장이 안 되기 때문에, 몇 천 글자 이상을 포함하려면 다중바이트 문자 집합(MBCS: Multi Byte Character Sets)이란 특별한 방법으로 확장해야 한다. 한자를 포함해야 하는 한국과 중국, 일본, 대만은 일찍이 MBCS의 일종인 DBCS(Double Byte Character Sets) 방법으로 코드 영역을 확장하여 필요한 글자를 배열하였다. DBCS는 8비트 시스템에서 연달아 있는 두 개의 바이트(8비트 코드 두 개)를 합해서 16비트 코드로 처리하는 방법이다. DBCS도 ISO 2022 규격에 따르는가, 아닌가에 따라 확장 글자 수의 차이가 있다.

ISO 2022에 따라 확장하면 모두 9,216(96×96)자를 정의할 수 있는데, KS X 1001(구: KSC 5601-1987, 일명 "완성형 코드")이 이와 같은 규격으로 정의한 규격이다(한글 2,350자와 한자 4,888자 포함).

표3-KS X 1001(구 KS C 5601) 특수문자 영역 전각 한글자모

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
A4A0		ㄱ	ㄲ	ㄳ	ㄴ	ㄵ	ㄶ	ㄷ	ㄸ	ㄹ	ㄺ	ㄻ	ㄼ	ㄽ	ㄾ	ㄿ
A4B0	ㅀ	ㅁ	ㅂ	ㅃ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅉ	ㅊ	ㅋ	ㅌ	ㅍ	ㅎ	ㅏ
A4C0	ㅐ	ㅑ	ㅒ	ㅓ	ㅔ	ㅕ	ㅖ	ㅗ	ㅘ	ㅙ	ㅚ	ㅛ	ㅜ	ㅝ	ㅞ	ㅟ
A4D0	ㅠ	ㅡ	ㅢ	ㅣ	(채움)	ㄱ	ㄲ	ㄳ	ㄴ	ㄵ	ㄶ	ㄷ	ㄹ	ㄺ	ㄻ	ㄼ
A4E0	ㄽ	ㄾ	ㄿ	ㅀ	ㅁ	ㅂ	ㅄ	ㅅ	ㅆ	ㅇ	ㅈ	ㅊ	ㅋ	ㅌ	ㅍ	ㅎ
A4F0	ㅏ	ㅐ	ㅑ	ㅒ	ㅓ	ㅔ	ㅕ	ㅖ	ㅗ	ㅘ	ㅙ	ㅚ	ㅛ	ㅜ	ㅝ	ㅞ

표4-KS X 1001의 옛한글자소 수

	현대한글자모	옛한글자모	합
자음	30	34	64
모음	21	8	29
합	51	42	

완성형 코드는 영역 대부분이 모두 정의되어 옛한글 확장의 여지가 거의 없는 규격이다. 한글 음절 2,350자는 통계적으로 사용빈도가 높은 글자들이 선택되었고, 2,350자 외 빈도가 낮은 한글 음절은 표현할 수 없다는 치명적인 약점이 있는데, 이에 대해 제정 당시(1987)부터 많은 비판이 있었다. 특히 잘 알려진 문제로, 발음표기(예: 태권도[태꿘도]), 축약어(예: 아퐤/아뢰어), 방언(예: 똠방각하, 닝큼/냉큼), 외래어 표기(찦차/커피숖) 등 빈도는 낮지만 필요한 글자들을 처리할 수 없다는 점 등, 많은 전문가와 학자들로부터 비판을 받았다. (구체적인 비판 항목들은 여러 논문에 기록되어 있고, 이 글의 주제와 거리가 멀어 생략한다.)

단, 완성형 코드에도 특수 기호 문자 영역에 "한글 자모"로 정의된 94자(채움코드 1자 포함)가 있는데, 여기에 옛한글 자소 42자가 포함되어 있다. 이 영역의 한글자모는 독립자소로만 사용되고, 자소 간의 조합으로 글자를 이루는데 사용되지는 않는다. 따라서 옛한글 자소가 포함되어 있다는 것 외 그 이상의 활용도는 없다.(이 한글자모 96자는 이후 Unicode에 "Hangul Compatibility Jamo"로 포함되게 된다.)

3. 조합형 옛한글과 OCHP

DBCS에서 확장하는 또 다른 방법 하나는 ISO 2022 규격을 벗어나지만 모든 현대 한글을 나타낼 수 있는 조합형 한글코드가 가능한 방법이다. 1987년 완성형 코드(KSC 5601)가 정의되기 이전, 대부분의 PC 제조업체들이 조합형 한글코드를 사용하였는데, 코드 배열은 회사별로 차이가 있었다. 여러 가지 조합형 배열 중 '상용조합형'[2]이라 불리던 코드가 가장 일반적으로 사용되었다. 상용조합형 외는 특정 회사 제품에서만 구현된 경우가 대부분이다. 또 한글코드의 배열은 상용조합형과 같아도, 한자나 특수(도형) 기호들의 배열 차이가 있어 다른 코드로 취급하기도 한다. 1987년 KSC5601 이후 PC 제조업체들이 KSC5601 완성형을 따라가고, 대부분의 조합형은 사라진 반면, 상용조합형은 KSSM이란 규격이 만들어져 KS 대안으로 정리되고 있었다. 조합형과 완성형의 장단점

2) 상용조합형: 1987년 KSC 5601 제정 이전에 가장 일반적으로 사용되던 한글코드. 초기 IBM을 비롯한 삼보컴퓨터 등에서 쓰였고, 한글 외 기호들을 KS에 맞춘 KSSM코드로 개정한 뒤, 이후 KSC5601-1992에 "2바이트 조합형 부호계"로 KS에 등록되었다.

논쟁은 1992년 KSSM 조합형이 KS 규격에 포함됨으로써 표면적인 논쟁은 일단락된다.

조합형 중 옛한글 확장을 구현한 경우도 상용조합형이 거의 유일하다. 상용조합형은 옛한글 확장 관련하여 KSSM(1991년) 이전 TG코드[3]와 KSSM코드 두 가지로 구분할 수 있다.

조합형의 옛한글 확장은 한국어전산학회 요청에 삼보컴퓨터가 적극적

표5-TG코드 옛한글 확장 구조

상위	문자 영역 : TG (NKP)	전산학회 확장
80 : D3	상용 조합형 한글	+조합형확장 (중성2,종성1) : ·, ㅢ, ㆀ
D4	특수 문자(TG 212-)	
D5 D6 D7	(reserved)	
D8 D9 DA DB	사용자 정의 문자 영역 I (= 4 x 204)	+옛한글확장I : 816자
DC : ED	한자 3640자	
EE EF	사용자 정의 문자 영역 II (= 2 x 204)	+옛한글확장II : 408자
F0 F1		
F2 : FE	(reserved)	
FF	(reserved)	

표6-OCHP 옛한글 확장 구조(KSSM)

상위	문자 영역 : KSSM	OCHP 확장
80 : D3	상용 조합형 한글	+조합형확장 (중성2,종성2) : ·, ㅢ, ㆀ,ㅇ
D4	특수 문자(TG 212-)	
D5 D6 D7	(reserved)	
D8	사용자정의문자 I	
D9 DA DB DC DD DE	특수문자 986자 (KSC5601과 동일)	
DF	사용자정의문자 II	
E0 : F9	한자 4888자 (KSC5601과 동일)	
FA FB FC FD FE	(reserved) 1025자 영역	+옛한글확장 : 1025자
FF	(reserved)	

3) TG코드: TG는 삼보컴퓨터의 영문이름(TriGem)의 약자이다. "TG코드"는 일반적인 용어는 아니었지만, 상용조합형 KSSM 조합형코드와 대비하여, 그 이전에 쓰던 코드를 "TG코드"로 부른다.

으로 반영하여 구현되었고, 삼보컴퓨터는 전용 H/W(비디오카드, 프린터)를 제작하여 공급하기까지 하였다. 한국어전산학회는 초기 TG코드를 기반으로 "사용자 정의 영역"에 옛한글 확장[4]을 하였고, 1991년 KSSM 개발 이후 KSSM 기반으로 옛한글을 재정렬하여 OCHP 옛한글 코드시스템을 개발하였다.

상용조합형 한글코드는 DBCS 규격의 16비트 특성상 초중종은 각각 5비트까지만 허용되고, 각각 32개의 공간이 있다. 그러나 사용할 수 없는 배치 공간(사용제한)이 섞여 있어, 실제 사용할 수 있는 공간은 더 적다.

표7-상용조합형 자소 배열과 OCHP 자소 확장

	초성		중성		종성	
값	비트	배열	비트	배열	비트	배열
0	00000	-	00000	〈사용제한〉	00000	〈사용제한〉
1	00001	〈채움〉	00001	〈사용제한〉	00001	〈채움〉
2	00010	ㄱ	00010	〈채움〉	00010	ㄱ
3	00011	ㄲ	00011	ㅏ	00011	ㄲ
4	00100	ㄴ	00100	ㅐ	00100	ㄳ
5	00101	ㄷ	00101	ㅑ	00101	ㄴ
6	00110	ㄸ	00110	ㅒ	00110	ㄵ
7	00111	ㄹ	00111	ㅓ	00111	ㄶ
8	01000	ㅁ	01000	〈사용제한〉	01000	ㄷ
9	01001	ㅂ	01001	〈사용제한〉	01001	ㄹ
10	01010	ㅃ	01010	ㅔ	01010	ㄺ
11	01011	ㅅ	01011	ㅕ	01011	ㄻ
12	01100	ㅆ	01100	ㅖ	01100	ㄼ
13	01101	ㅇ	01101	ㅗ	01101	ㄽ
14	01110	ㅈ	01110	ㅘ	01110	ㄾ
15	01111	ㅉ	01111	ㅙ	01111	ㄿ
16	10000	ㅊ	10000	〈사용제한〉	10000	ㅀ
17	10001	ㅋ	10001	〈사용제한〉	10001	ㅁ
18	10010	ㅌ	10010	ㅚ	10010	+ ㅭ
19	10011	ㅍ	10011	ㅛ	10011	ㅂ
20	10100	ㅎ	10100	ㅜ	10100	ㅄ
21	10101	-	10101	ㅝ	10101	ㅅ
22	10110	-	10110	ㅞ	10110	ㅆ
23	10111	-	10111	ㅟ	10111	ㅇ
24	11000	-	11000	〈사용제한〉	11000	ㅈ
25	11001	-	11001	〈사용제한〉	11001	ㅊ
26	11010	-	11010	ㅠ	11010	ㅋ
27	11011	-	11011	ㅡ	11011	ㅌ
28	11100	-	11100	ㅢ	11100	ㅍ
29	11101	-	11101	ㅣ	11101	ㅎ
30	11110	-	11110	+ ㆍ	11110	+ ㆁ
31	11111	-	11111	+ ㅢ	11111	〈사용제한〉

+ : OCHP에서 확장된 자소.

4) 김홍규, 『한글 옛글자의 컴퓨터 처리 효율화 방안과 이에 대한 고시조 데이터베이스시스템 개발 연구』(한국어전산학회, 1992), 13-18쪽.

현대한글 자소를 배치하고 남은 공간을 보면, 중성이 2개, 종성이 2개이다. 초성 영역은 12개가 남는데, 한자나 도형문자 등을 확장하기 위한 영역이다. 중성과 종성은 각각 2개까지 확장 가능하다.

TG코드는 조합형 기반으로 중성 2개(ㆍ, ㅣ), 종성 1개(ㆆ)를 확장하였고, 옛한글 완성자 1,224자를 사용자정의영역에 확장하였다. TG코드 옛한글 확장은 마지막 단계에서 KSSM기반의 OCHP로 전환되었다.

OCHP는 조합형 기반으로 옛한글 중성 2개(ㆍ, ㅣ)와 종성 2개(ㆆ,ㅇ)씩 추가 배열하였고, 조합되지 않는 옛한글은 OCHP 완성자 형태로 1,025자(표8)를 배열하였다. 그 외 발음기호와 연구에 필요한 일부 한자 등을 포함하여 사용자정의영역II에 205자를 배치하였다. KSSM 시스템 하에서는 최대치 배열을 한 방법이라 할 수 있다.

그림1-삼보컴퓨터 옛한글 사용지침서(1991)

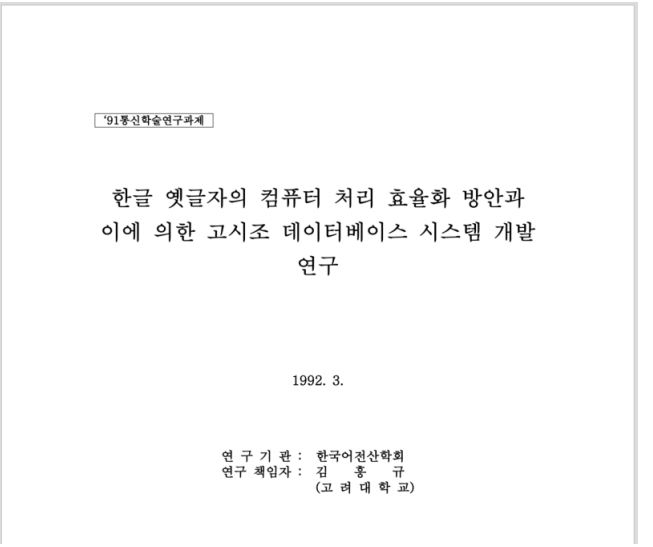
OCHP의 구체적 설계 과정과 사양은 「한글 옛글자의 컴퓨터 처리 효율화 방안과 이에 의한 고시조 데이터베이스 시스템 개발 연구」(김홍규: 1992)에 잘 나타나 있다.

OCHP를 구현한 한글비디오보드의 사용지침서에는 당시(1991) "알려진 옛한글의 95%까지 처리 가능하다"[5]고 안내하고 있다.

OCHP는 본격적인 옛한글 연구의 기틀이 되었다고 평가할 수 있다. 당시까지 이렇다 할 옛한글 코드 시스템이 없는 상황에서, OCHP의 등장은 옛한글 자료를 전산화하고, 분류·분석할 수 있는 기준 도구가 되었다는데 큰 의미가 있다. 특히 컴퓨터 운영체체(OS)에 조합형코드 체계(KSSM)를 확장 적용해서, 텍스트편집기와 DB프로그램 등 일반 응용프로그램들에서도 옛한글을 이용할 수 있게 된 것은 당시 옛한글 연구 상황에 비추어, 본격적인 연구를 할 수 있는 중요한 전환점이 되었다고 할 수 있다.

---

5) OCHP, 『한글 비디오보드(고어) 사용 지침서』(삼보컴퓨터, 1991), 2-3쪽.



'91통신학술연구과제

한글 옛글자의 컴퓨터 처리 효율화 방안과
이에 의한 고시조 데이터베이스 시스템 개발
연구

1992. 3.

연 구 기 관 : 한국어전산학회
연구 책임자 : 김 홍 규
(고 려 대 학 교)

그림2-OCHP 개발의 바탕이 된 논문(김흥규, 1992)

그러나 KSSM코드는 구조상 추가적인 옛한글 확장이 불가능하고, 당시까지 조사되었던 옛한글 외 옛한글이 새로 발견되어도 대응할 수 없다는 한계가 있었다.

코드상으로는 한글 조합에 이용되는 중성 코드에 아래아(ㆍ)의 확장이 큰 의의를 갖는다. 아래아(ㆍ)가 포함된 옛 글자들은 대부분 조합형으로 처리하고 나머지 옛한글을 완성자로 처리할 수 있어 코드의 효율을 크게 높인다. 모음인 아래아(ㆍ)가 포함된 옛한글의 개수가 절대적으로 많은 데다 근대까지 남아 있던 자소여서, 옛한글에 대한 표현 가능성이 알려진 글자들보다 훨씬 넓어졌다는 것을 의미한다.

4. HNC 1.x/2.x/3.x 조합형 (한글과컴퓨터 조합형)

옛한글 데이터 처리에 많이 쓰이는 아래아한글(ᄒᆞᆫ글, HWP)의 내부 데이터 문자 코드는 HNC코드라(한글과컴퓨터 코드 또는 한컴 코드)라 하는데, ᄒᆞᆫ글1.0 시절부터 완전한 16비트 시스템으로 설계되었다. 8비트 시스템과는 달리 16비트 시스템은 모든 글자를 16비트로 처리하는 것을 이르는데, 영문자 'A'나 숫자 '1' 또는 한글 '가' 등이 모두 16비트 크기의 글자 단위로 동일하게 취급된다. 완전한 16비트 코드 시스템이기에,

표8-OCHP 옛한글 완성자 확장 목록 1,025자(FA30-FEFE)

FA30	갃갌갏갻갌걳걺걻걿겏겋겿겴곬곯곲	FD30	봐봥뵈부북분불붏붐붓붖뷰브븍븐븓
FA40	괴균굶귷궋귀긌긊긆긋깃깅깘깛깚깡	FD40	블븛븜븟븨븨빅빔빕빗빙ᄫᆞ ᄫᆞᆫ ᄫᆞᆯ ᄫᆞᆹ ᄫᆞᆷ
FA50	ᄀᆞᆨ ᄀᆞᇫ 깄껋 ᄡᅡ ᄡᅡᆨ ᄡᅢ ᄡᅥ ᄡᅦ ᄡᅦᆯ ᄡᅧ ᄡᅧᆫ ᄡᅩ ᄡᅩᆽ ᄡᅬ ᄡᅮ	FD50	ᄫᆞᆺ ᄫᆡᆨ 삸삷 ᅀᅡ ᅀᅡᆷ ᅀᅡᆹ ᅀᅡᆺ ᅀᅢ ᅀᅢᆫ ᅀᅢᆺ ᅀᅢᇰ ᅀᅣ ᅀᅣᆨ ᅀᅣᇰ ᅀᅥ
FA60	ᄡᅮᆯ ᄡᅮᆷ ᄡᅰ ᄡᅱ ᄡᅳ ᄡᅳᆮ ᄡᅳᆯ ᄡᅳᆺ ᄡᅴ ᄡᅵᆺ ᄡᅵ ᄡᅵᆯ ᄡᆞ ᄡᆡ ᄢᅡ ᄢᅢ	FD60	ᅀᅥᆨ ᅀᅥᆯ ᅀᅥᆷ ᅀᅥᆹ ᅀᅥᆸ ᅀᅥᆺ ᅀᅥᇰ ᅀᅦ ᅀᅦᆺ ᅀᅧ ᅀᅧᆨ ᅀᅧᆫ ᅀᅧᆯ ᅀᅧᆷ ᅀᅧᇰ ᅀᅩ
FA70	ᄢᅢᆫ ᄢᅥ ᄢᅦ ᄢᅦᆯ ᄢᅧ ᄢᅧᆺ ᄢᅨᆺ ᄢᅮ ᄢᅮᆫ ᄢᅮᆯ ᄢᅮᆷ ᄢᅮᆺ ᄢᅰ ᄢᅱ ᄢᅳ	FD70	ᅀᅩᆫ ᅀᅩᆯ ᅀᅩᆷ ᅀᅩᆸ ᅀᅩᇰ ᅀᅭ ᅀᅭᆨ ᅀᅭᆺ ᅀᅭᇰ ᅀᅮ ᅀᅮᆫ ᅀᅮᆯ ᅀᅮᆷ ᅀᅮᆺ ᅀᅱ
FA80	ᄢᅳᆫ ᄢᅳᆯ ᄢᅳᆷ ᄢᅴ ᄢᅴᆫ ᄢᅴᆯ ᄢᅴᆷ ᄢᅴᆺ ᄢᅵ ᄢᅵᆯ ᄢᅵᆷ ᄢᆞ ᄢᆞᆫ ᄭᅡ ᄭᅡᆨ	FD80	ᅀᅲ ᅀᅲᆨ ᅀᅲᆫ ᅀᅲᆺ ᅀᅲᇰ ᅀᅳ ᅀᅳᆨ ᅀᅳᆫ ᅀᅳᆯ ᅀᅳᆱ ᅀᅳᆹ ᅀᅳᆷ ᅀᅳᆹ ᅀᅳᆺ ᅀᅴ
FA90	ᄭᅡᆫ ᄭᅡᆯ ᄭᅡᆷ ᄭᅡᆸ ᄭᅡᆹ ᄭᅡᇦ ᄭᅡᆺ ᄭᅡᇰ ᄭᅢ ᄭᅢᆫ ᄭᅢᆯ ᄭᅢᆺ ᄭᅥ ᄭᅥᆨ ᄭᅥᆪ ᄭᅥᆫ	FD90	ᅀᅴᆫ ᅀᅴᆺ ᅀᅵ ᅀᅵᆫ ᅀᅵᆯ ᅀᅵᆹ ᅀᅵᆷ ᅀᅵᆸ ᅀᅵᆺ ᅀᅵᇰ ᅀᆞ ᅀᆞᆨ ᅀᆞᆫ ᅀᆞᆯ ᅀᆞᆹ ᅀᆞᆷ
FAA0	ᄭᅥᆯ ᄭᅥᆲ ᄭᅥᇢ ᄭᅥᆷ ᄭᅥᆸ ᄭᅥᇦ ᄭᅥᆺ ᄭᅥᆪ ᄭᅦ ᄭᅦᆫ ᄭᅦᆯ ᄭᅧ ᄭᅧᆫ ᄭᅧᆯ ᄭᅧᆺ ᄭᅧᇰ	FDA0	ᅀᆞᆹ ᅀᆞᆸ ᅀᆞᇦ ᅀᆞᆺ ᅀᆡ ᅀᆡᆫ ᅀᆡᆯ ᅀᆡᆺ 앇 앓 앇 ᅌᅢᆺ 얎 ᅌᅣᇰ 얿 ᅌᅥᆺ
FAB0	ᄭᅩ ᄭᅩᆨ ᄭᅩᆫ ᄭᅩᆮ ᄭᅩᆯ ᄭᅩᆹ ᄭᅩᆷ ᄭᅩᆸ ᄭᅩᆺ ᄭᅩᇰ ᄭᅩᆽ ᄭᅩᆾ ᄭᅪ ᄭᅪᆨ ᄭᅪᇰ ᄭᅫ	FDB0	엣 옍 였 옓 엷 옋 옔 옰 옶 왒 외 욦 욠 욚 웺 위
FAC0	ᄭᅬ ᄭᅬᆯ ᄭᅭ ᄭᅭᆷ ᄭᅮ ᄭᅮᆨ ᄭᅮᆫ ᄭᅮᆮ ᄭᅮᆯ ᄭᅮᆷ ᄭᅮᆸ ᄭᅮᆺ ᄭᅮᇰ ᄭᅮᆽ ᄭᅯ ᄭᅯᇰ	FDC0	읁 읪 읓 읶 잃 잊 ᅇᅧ ᅇᅭ ᅇᅭᆫ ᅇᅭᆷ ᅇᅯ ᅇᅲ ᅇᅲᆫ ᅇᅲᆷ ᅇᅵ ᅇᅵᆫ
FAD0	ᄭᅰ ᄭᅰᆫ ᄭᅱ ᄭᅱᆫ ᄭᅲ ᄭᅲᆷ ᄭᅳ ᄭᅳᆫ ᄭᅳᇝ ᄭᅳᆮ ᄭᅳᆯ ᄭᅳᇡ ᄭᅳᆷ ᄭᅳᆺ ᄭᅳᇫ ᄭᅳᇰ	FDD0	ᅌᅡ ᅌᅢ ᅌᅥ ᅌᅥᆨ ᅌᅥᆸ ᅌᅥᆺ ᅌᅥᇰ ᅌᅦ ᅌᅦᆫ ᅌᅦᆺ ᅌᅧᇰ ᅌᅩ ᅌᅩᆯ ᅌᅭ ᅌᅮ ᅌᅮᆯ
FAE0	ᄭᅴ ᄭᅴᆫ ᄭᅴᆯ ᄭᅴᆺ ᄭᅵ ᄭᅵᆫ ᄭᅵᆯ ᄭᅵᆷ ᄭᅵᆸ ᄭᅵᆺ ᄭᆞ ᄭᆞᆨ ᄭᆞᆫ ᄭᆞᆯ ᄭᆞᆷ ᄭᆞᆺ	FDE0	ᅌᅳ ᅌᅳᆫ ᅌᅳᆯ ᅌᅴ ᅌᅵ ᅌᅵᆺ ᅌᅵᇰ ᅌᆞ ᅌᆞᆯ ᅙᅡᆫ ᅙᅳᆷ ᅙᅳᆸ 잓 잜 잸 쟛
FAF0	ᄭᆡ ᄭᆡᆫ ᄭᆡᆯ ᄭᆡᆺ 낛 낢 낤 낳 낛 낭 넚 넓 녁 녃 뇌	FDF0	젛 졋 졵 종 죄 죴 줗 쥐 쥔 즇 짒 징 ᄌᆞᆰ ᄌᆞᇢ ᄌᆞᇫ
FB30	눋 눐 눖 눖 눕 뉘 늘 늶 닚 닝 닰 닿 ᄂᆞ ᄂᆞᆫ ᄮᅡ 닸	FE30	ᄲᅡ ᄲᅡᆨ ᄲᅡᆫ ᄲᅡᆷ ᄲᅡᆺ ᄲᅡᇰ ᄲᅢ ᄲᅣ ᄲᅣᇰ ᄲᅥ ᄲᅥᆨ ᄲᅩ ᄲᅩᆨ ᄲᅩᆫ ᄲᅩᆷ ᄲᅩᆺ
FB40	닮 닶 닿 닷 닸 댯 덦 덩 돕 돆 돐 되 둘 둙 둥 뒤	FE40	ᄲᅩᇰ ᄲᅩᆽ ᄲᅩᆾ ᄲᅪᇰ ᄲᅬ ᄲᅬᆫ ᄲᅬᆯ ᄲᅭ ᄲᅮ ᄲᅮᆷ ᄲᅳ ᄲᅳᆨ ᄲᅳᆫ ᄲᅳᆮ ᄲᅳᆯ ᄲᅳᆷ
FB50	듫 듮 듷 듰 ᄃᆞᆯ ᄃᆞᆳ ᄃᆞᆲ ᄃᆞᇦ ᄃᆞᇫ ᄠᅡ ᄠᅡᆨ ᄠᅡᆯ ᄠᅡᆷ ᄠᅡᆸ ᄠᅢ ᄠᅥ	FE50	ᄲᅳᆺ ᄲᅳᆽ ᄲᅴ ᄲᅴᇰ ᄲᅵ ᄲᅵᆨ ᄲᅵᆫ ᄲᅵᆷ ᄲᅵᆺ ᄲᅵᇰ ᄲᆞ ᄲᆞᆨ ᄲᆞᆫ ᄲᆞᆯ ᄲᆞᆷ ᄲᆡ
FB60	ᄠᅥᆨ ᄠᅥᆫ ᄠᅥᆮ ᄠᅥᆯ ᄠᅥᆲ ᄠᅥᇢ ᄠᅥᆺ ᄠᅦ ᄠᅦᆺ ᄠᅧ ᄠᅨ ᄠᅩ ᄠᅩᆨ ᄠᅩᆷ ᄠᅬ ᄠᅬᆺ	FE60	ᄲᆡᆫ ᄴᅡ ᄴᅡᆨ ᄴᅡᆫ ᄴᅡᆯ ᄴᅡᆷ ᄴᅡᆸ ᄴᅡᆺ ᄴᅡᇰ ᄴᅢ ᄴᅢᆫ ᄴᅢᇰ ᄴᅣ ᄴᅣᆨ ᄴᅣᇰ ᄴᅥ
FB70	ᄠᅮ ᄠᅮᆨ ᄠᅮᆯ ᄠᅮᆲ ᄠᅮᇰ ᄠᅱ ᄠᅱᆯ ᄠᅲ ᄠᅳ ᄠᅳᆫ ᄠᅳᆮ ᄠᅳᆯ ᄠᅳᆹ ᄠᅳᇡ ᄠᅳᆷ	FE70	ᄴᅥᆨ ᄴᅦ ᄴᅧ ᄴᅧᆨ ᄴᅨ ᄴᅩ ᄴᅩᆨ ᄴᅩᆺ ᄴᅩᆾ ᄴᅬ ᄴᅬᆨ ᄴᅭ ᄴᅭᆨ ᄴᅮ ᄴᅮᆨ
FB80	ᄠᅳᆺ ᄠᅳᆽ ᄠᅴ ᄠᅵ ᄠᅵᆨ ᄠᅵᆫ ᄠᅵᆯ ᄠᅵᆷ ᄠᅵᆺ ᄠᆞ ᄠᆞᆫ ᄠᆞᆮ ᄠᆞᆯ ᄠᆞᆷ ᄠᆞᆺ	FE80	ᄴᅮᆯ ᄴᅱᆨ ᄴᅲ ᄴᅲᆨ ᄴᅳ ᄴᅳᆨ ᄴᅳᆫ ᄴᅳᆷ ᄴᅳᆺ ᄴᅴ ᄴᅵ ᄴᅵᆨ ᄴᅵᆫ ᄴᅵᆯ ᄴᅵᆷ
FB90	ᄠᆡ ᄣᅡ ᄣᅢ ᄣᅥ ᄣᅥᆲ ᄣᅥᆺ ᄣᅩ ᄣᅬ ᄣᅮ ᄣᅲ ᄣᅲᆷ ᄣᅳ ᄣᅴ ᄣᅵ ᄣᅵᆯ ᄣᆞ	FE90	ᄴᅵᆺ ᄴᅵᇰ ᄴᅵᇰ ᄴᆞ ᄴᆞᆨ ᄴᆞᆫ ᄴᆞᆯ ᄴᆞᆷ ᄴᆞᆸ ᄴᆞᆺ ᄴᆡ ᄴᆡᆨ ᄴᆡᇰ ᄶᅮᆸ 첳 ᄎᆈ
FBA0	ᄯᅡ ᄯᅡᆨ ᄯᅡᆫ ᄯᅡᆯ ᄯᅡᇡ ᄯᅡᆷ ᄯᅡᆸ ᄯᅡᆺ ᄯᅡᇰ ᄯᅡᇦ ᄯᅢ ᄯᅢᆨ ᄯᅢᆫ ᄯᅢᆯ ᄯᅢᇰ ᄯᅣ	FEA0	취 칠 ᄎᆞᆴ ᄎᆞᆹ ᄎᆡ ᄎᆞᆯ ᄎᆞᆸ 캃 ᄶᅥᆫ ᄶᅥᆯ ᄶᅩ ᄶᅩᆫ ᄶᅩᆺ ᄶᅩᇰ ᄶᅫ ᄶᅬ
FBB0	ᄯᅣᇰ ᄯᅥ ᄯᅥᆨ ᄯᅥᆫ ᄯᅥᆮ ᄯᅥᆯ ᄯᅥᆲ ᄯᅥᆺ ᄯᅥᇰ ᄯᅦ ᄯᅦᆫ ᄯᅧ ᄯᅨ ᄯᅩ ᄯᅩᆨ ᄯᅩᆯ	FEB0	ᄶᅬᇰ ᄶᅯ ᄶᅳ ᄶᅳᆫ ᄶᅴ ᄶᅵ 춰 텂 텋 텘 퇴 튀 틆 ᄩᅡ ᄩᅥ ᄩᅥᆨ
FBC0	ᄯᅩᆸ ᄯᅩᆺ ᄯᅩᇰ ᄯᅩᆹ ᄯᅬ ᄯᅮ ᄯᅮᆨ ᄯᅮᆫ ᄯᅮᆯ ᄯᅮᆲ ᄯᅮᆷ ᄯᅮᆺ ᄯᅮᇰ ᄯᅱ ᄯᅱᆯ ᄯᅱᆺ	FEC0	ᄩᅥᆯ ᄩᅥᆲ ᄩᅥᆺ ᄩᅧ ᄩᅩ ᄩᅩᆷ ᄩᅩᆸ ᄩᅮ ᄩᅮᆨ ᄩᅱ ᄩᅳ ᄩᅳᆫ ᄩᅳᆮ ᄩᅳᆺ ᄩᅴ ᄩᅴᆷ
FBD0	ᄯᅲ ᄯᅲᆷ ᄯᅲᇰ ᄯᅳ ᄯᅳᆨ ᄯᅳᆫ ᄯᅳᆮ ᄯᅳᆯ ᄯᅳᆲ ᄯᅳᆷ ᄯᅳᆺ ᄯᅳᇰ ᄯᅳᆽ ᄯᅴ ᄯᅴᆫ ᄯᅴᆺ	FED0	ᄩᅵ ᄩᅵᆯ ᄩᆞ ᄩᆞᆫ ᄩᆞᆯ ᄰᅡ ᄰᅡᆯ ᄰᅥ ᄰᅥᆯ ᄰᅦ ᄰᅧᆸ ᄰᅱ ᄰᅳ ᄰᅳᆯ ᄰᅳᆷ ᄰᅳᆺ
FBE0	ᄯᅴᇫ ᄯᅵ ᄯᅵᆨ ᄯᅵᆫ ᄯᅵᆮ ᄯᅵᆯ ᄯᅵᆺ ᄯᅵᇰ ᄯᆞ ᄯᆞᆨ ᄯᆞᆫ ᄯᆞᆯ ᄯᆞᆷ ᄯᆞᆺ ᄯᆞᇰ ᄯᆡ	FEE0	ᄰᅴ ᄰᆞ ᄰᆞᆷ 퍐 피 ᄑᆞᆹ ᄱᅧ ᄱᅧᆫ ᄱᅮᆫ ᄱᅴ ᄱᆞ 핛 햀 ᄒᆞᆬ ᄒᆞᆹ ᄒᆞᆬ
FBF0	ᄯᆡᆨ ᄯᆡᆫ ᄯᆡᆯ ᄯᆡᇰ 랎 랗 랷 랑 렇 렸 렿 롮 롷 룘 룟	FEF0	홠 회 훖 흫 힇 힚 ᄒᆞᆯ ᄒᆞᆴ ᄒᆞᇦ ᄻᅧ ᅘᅧ 혤 ᅘᅩ 횽 ᅘᅵ
FC30	뢰 륬 뤼 륬 릾 맇 ᄅᆞᆳ ᄅᆞᆲ ᄅᆞᆹ ᄅᆞᇦ ᄅᆞᇫ 맍 맏 뫖 뫼 묷		
FC40	묷 뭉 묷 믕 밓 ᄆᆞᆯ ᄆᆞᇫ 밓 밦 밟 밠 밨 벘 볐 봀 봒		
FC50	붒 붋 붔 븛 븟 븡 빇 빙 빐 ᄇᆞᆯ ᄇᆞᇢ ᄇᆞᆲ ᄇᆞᇫ ᄇᆡᆹ 빠 빡		
FC60	빤 빧 빨 빰 빳 빠ᇰ 빼 뺏 뺘 뺨 뻐 뻑 뻔 뻡 뻣 뻐ᇰ		
FC70	뻬 뻼 뼈 뼘 뼛 뼈ᇰ 뼤 뽀 뽁 뽄 뽐 뽑 뽓 뽃 뽀ᇰ		
FC80	뾰 뿌 뿐 뿔 뿜 뿣 쀠 쀼 쁘 쁙 쁜 쁠 쁴 삐 삔		
FC90	삘 삣 삐ᇰ ᄈᆞ ᄈᆞᆫ ᄈᆞᆯ ᄈᆞᆷ ᄈᆡ ᄈᆡᆨ ᄈᆡᆫ ᄈᆡᆯ ᄈᆡᆺ ᄈᆡᇰ ᄫᅡ ᄫᅡᆫ ᄫᅡᆮ		
FCA0	ᄫᅡᆯ ᄫᅡᆷ ᄫᅡᆺ ᄫᅡᇀ ᄫᅢ ᄫᅢᆺ ᄫᅥ ᄫᅥᆷ ᄫᅦ ᄫᅧ ᄫᅧᆨ ᄫᅧᆺ ᄫᅨ ᄫᅨᆺ ᄫᅩ ᄫᅩᆯ		
FCB0	ᄫᅩᇙ ᄫᅩᆷ ᄫᅪ ᄫᅮ ᄫᅮᆫ ᄫᅮᆯ ᄫᅮᆷ ᄫᅱ ᄫᅳ ᄫᅳᆫ ᄫᅳᆯ ᄫᅳᆹ ᄫᅳᆷ ᄫᅳᆺ ᄫᅴ ᄫᅵ		
FCC0	ᄫᅵᆫ ᄫᅵᆯ ᄫᆞ ᄫᆞᆫ ᄫᆞᆯ ᄫᆞᆰ ᄫᆞᆳ ᄫᆞᇙ ᄫᆡ ᄫᆡᆫ 샶 샋 샸 섫 섟 셟		
FCD0	셦 셢 셨 솏 솘 쇠 쇳 숛 숭 쉻 쉬 쉰 쉼 슶 싮 싦		
FCE0	ᄉᆞᇢ ᄉᆞᇜ ᄉᆞᇰ ᄉᆡᆹ ᄉᆡᆪ 싸 싼 쌀 쌉 쌋 싸ᇰ 싸ᇦ 쌔 쌰 쌰ᇰ		
FCF0	써 썩 썬 썰 썻 쎄 쎈 쎠 쎤 쏘 쏜 쏟 쏠 쏫 쏘ᇰ		

(*) 김홍규(1992), 앞의 논문, 27-29쪽.

모두 65,536자(2의 16승)를 배열할 수 있는데, 한글 약 32,000여자, 한자 16,000여자를 포함하여 사용자정의로 2,048자까지 배열할 수 있다.

HWP에서 사용하는 조합형도 상용조합형을 바탕으로 확장한 조합형이다. 16비트 조합형이라 자소당 5비트 제한은 있지만, 완전한 16비트 시스템이기 때문에 상용조합형처럼 〈사용제한〉 영역을 가지지는 않는다. 따라서 상용조합형보다 많은 자소를 배열할 수 있다.

ᄒᆞᆫ글1.0에서 구현했던 초기 조합형(HNC 1.x)과 ᄒᆞᆫ글2.0으로 판올림하면서 수정된 조합형(HNC 2.x)이 있다. 또 윈도우 버전인 ᄒᆞᆫ글3.0으로 판올림하면서, HNC 3.x로 바뀌었으나, 한글코드 부분은 2.x와 같아서, 한글코드 관련해서 HNC 3.x는 HNC 2.x로 통칭하기도 한다.

표9-HNC 1.x와 2.x/3.x 코드 비교-옛한글자소 수

	HNC 1.x 조합형 옛한글 확장	HNC 2.x/3.x 조합형 옛한글 확장
초성	+ 12자 ㅸ,ㅱ,ㅵ,ㅶ,ㅷ,ㅹ,ㅺ,ㅿ,ㆁ,ㅻ,ㅼ,ㆆ	+ 12자 ㅱ,ㅶ,ㅺ,ㅻ,ㅸ,ㅵ,ㅷ,ㅹ,ㅼ,ㅿ,ㆁ,ㆅ
중성	+ 8자 ㅑㅒ,ㅖ,ㅛ,ㅞ,ㅟ,ㆍ,ㅢ	+ 5자 ㅖ,ㅛ,ㅟ,ㆍ,ㅢ
종성	+ 4자 ㆅ,ㅁㄱ,ㆆ,ㆁ	+ 4자 ㆅ,ㅸ,ㅿ,ㆁ

HNC 1.x와 2.x에서 확장한 옛한글 자소 배열은 OCHP와 다르다. 중성의 아래아 두 글자(ㆍ,ㅢ) 외에는 모두 다른 배열을 가진다. 상용조합형에서는 하지 못했던 초성자음군 확장도 12자나 했고, 중성도 8자나 확장했다.

HNC 2.x에서는 오히려 1.x보다 중성 자소가 적게 배열되었다. 중성의 일부 코드(〈Ext〉)를 확장 플랙으로 이용하여, 옛한글 완성자를 추가할 수 있도록 하기 위함이다. 이 방법(기술적인 내용 생략)으로 HNC 2.x에서는 자소조합으로 지원하지 못하는 옛한글 1,865자를 완성자로 추가 지원한다. 확장된 옛한글 조합 자소는 실제 사용 용도나 중요도가 높은 자소를 우선으로 배치했지만, 32자 공간에 모두 배치하지 못하고, 다수의 옛한글 자소는 완성자로 지원하는 방법을 사용한 것이다. 대표적인 예로 정치음 치두음 자소가 포함된 옛한글은 완성자 방법으로만 지원한다.

표10-HNC 1.x 조합형 자소 배열

값	비트	초성	중성	종성
0	00000	+ ㅸ	-	+ ㆆ
1	00001	〈채움〉	-	〈채움〉
2	00010	ㄱ	〈채움〉	ㄱ
3	00011	ㄲ	ㅏ	ㄲ
4	00100	ㄴ	ㅐ	ㄳ
5	00101	ㄷ	ㅑ	ㄴ
6	00110	ㄸ	ㅒ	ㄵ
7	00111	ㄹ	ㅓ	ㄶ
8	01000	ㅁ	+ ㅛㅏ	ㄷ
9	01001	ㅂ	+ ㅛㅐ	ㄹ
10	01010	ㅃ	ㅔ	ㄺ
11	01011	ㅅ	ㅕ	ㄻ
12	01100	ㅆ	ㅖ	ㄼ
13	01101	ㅇ	ㅗ	ㄽ
14	01110	ㅈ	ㅘ	ㄾ
15	01111	ㅉ	ㅙ	ㄿ
16	10000	ㅊ	+ ㅟㅣ	ㅀ
17	10001	ㅋ	+ ㅟㅔ	ㅁ
18	10010	ㅌ	ㅚ	+ ㅁㄱ
19	10011	ㅍ	ㅛ	ㅂ
20	10100	ㅎ	ㅜ	ㅄ
21	10101	+ ㅱㅣ	ㅝ	ㅅ
22	10110	+ ㅱㅔ	ㅞ	ㅆ
23	10111	+ ㅱㅐ	ㅟ	ㅇ
24	11000	+ ㅱㅏ	+ ㅞㅣ	ㅈ
25	11001	+ ㅱㅑ	+ ㅟㅔ	ㅊ
26	11010	+ ㅱㅘ	ㅠ	ㅋ
27	11011	+ ㅿ	ㅡ	ㅌ
28	11100	+ ㆁ	ㅢ	ㅍ
29	11101	+ ㅱㅐ	ㅣ	ㅎ
30	11110	+ ㅱㅏ	+ ㆍ	+ ㅎ
31	11111	+ ㅎ	+ ㅢ	+ ㆁ

+: 조합형에서 확장된 자소

표11-HNC 2.x/3.x 조합형 자소 배열

값	비트	초성	중성	종성
0	00000	+ ㅱ	〈Ext〉	+ ㆆ
1	00001	〈채움〉	〈Ext〉	〈채움〉
2	00010	ㄱ	〈채움〉	ㄱ
3	00011	ㄲ	ㅏ	ㄲ
4	00100	ㄴ	ㅐ	ㄳ
5	00101	ㄷ	ㅑ	ㄴ
6	00110	ㄸ	ㅒ	ㄵ
7	00111	ㄹ	ㅓ	ㄶ
8	01000	ㅁ	〈Ext〉	ㄷ
9	01001	ㅂ	〈Ext〉	ㄹ
10	01010	ㅃ	ㅔ	ㄺ
11	01011	ㅅ	ㅕ	ㄻ
12	01100	ㅆ	ㅖ	ㄼ
13	01101	ㅇ	ㅗ	ㄽ
14	01110	ㅈ	ㅘ	ㄾ
15	01111	ㅉ	ㅙ	ㄿ
16	10000	ㅊ		ㅀ
17	10001	ㅋ	+ ㅟ	ㅁ
18	10010	ㅌ	ㅚ	+ ㅸ
19	10011	ㅍ	ㅛ	ㅂ
20	10100	ㅎ	ㅜ	ㅄ
21	10101	+ ㅲ	ㅝ	ㅅ
22	10110	+ ㅶ	ㅞ	ㅆ
23	10111	+ ㅵ	ㅟ	ㅇ
24	11000	+ ㅸ	+ ㅞㅣ	ㅈ
25	11001	+ ㅿ	+ ㅟㅔ	ㅊ
26	11010	+ ㅈㅅ	ㅠ	ㅋ
27	11011	+ ㅶ	ㅡ	ㅌ
28	11100	+ ㅉㅈ	ㅢ	ㅍ
29	11101	+ ㅿ	ㅣ	ㅎ
30	11110	+ ㆁ	+ ㆍ	+ ㅿ
31	11111	+ ㆅ	+ ㅢ	+ ㆁ

+: 조합형에서 확장된 자소

HNC 2.x/3.x 코드도 한글97(HWP)까지만 사용되고, 한글워디안(2000) 판올림 때부터 유니코드를 채택함으로서 HNC 코드는 더 이상 한글 제품에 반영되지 않는다.

한글의 첫 번째 코드시스템인 HNC 1.x 코드는 실제 사용된 기간이 길지 않고, 옛한글 배열의 실용성도 떨어졌다. 코드를 설계할 때, 옛한글 관련하여 참고할 데이터가 많지 않았던 때이기도 하다. 한글 2.0부터 탑재된 HNC 2.x/3.x 코드는 많은 옛한글 자소를 조합형으로 지원함과 동시에 옛한글 완성자로도 1,865자를 지원함으로써, 당시(1993년) 가장 많은 옛한글 표현할 수 있는 제품이 되었고, 옛한글을 실용적인 수준까지 끌어올렸다고 할 수 있다.

대부분 응용프로그램들 특히 문서편집기(Word Processor)들이 OS가 제공하는 2,350자 완성형 코드를 기반으로 제품을 만들 때도, 한글은 모든 한글을 표현하는 것을 중요한 가치로 삼아, 조합형 한글 기반으로 코드 시스템을 설계하였고, 덕분에 필요한 거의 대부분의 옛한글을 표현할 수 있었다. 또한 HNC코드는 HWP 내부 구현용으로 만들어졌으나, 외부데이터 처리를 위해 16비트 Text파일(.2b) 형태로 내보내기(Import/Export) 기능을 지원하였고, 실용적인 옛한글 DB구축과 입출력 도구가 되었다. 또 옛한글 관련 문헌들이나 논문을 정리하는 중요한 도구로써 2000년대 초반까지 옛한글에 관한 한 거의 유일한 해결책이었다.

그러나 조합형 자소 받침 배열이 OCHP와 다르게 확장된 부분은 아쉬운 부분으로 남는다. 중성의 아래아(ㆍ, ㅐ)는 같은 배열이지만, 받침은 1.x에서 배열되었던 2글자의 위치를 유지함으로서 결론적으로 OCHP 받침 배열과 다른 배열을 가지게 되었는데, 같은 배열을 가졌으면 좀 더 활용성이 높았을 것으로 생각된다. OCHP의 기반이 된 KSSM 배열이 KS 표준에 포함되었는데, 만약 OCHP의 중성 종성이 한글의 중성 종성 배열과 같았다면, 이도 KS 표준의 기회가 있었을 것 같다.

5. 유니코드 PUA 옛한글

1995년 유니코드 2.0의 기본평면(BMP[6])에 현대한글 글자마디 11,172

6) BMP(Basic Multilingual Plane): 유니코드는 65,536자를 처리하는 17개의 다국어 평면(Multilingual Plane)으로 구성되어 있는데, 그 중 코드포인트 U+0000부터 U+FFFF까지

자가 들어가는 역사적인 사건이 일어났다. 유니코드 1.0에서 배열되어 있던 한글 2,350자가 제외되고, 새로운 영역에 11,172자 현대한글이 모두 반영되었는데, 유니코드 역사상 이미 배치된 글자 블록이 제외되고 재배치되는 경우는, 이 경우 말고는 없다. 이로서 조합형/완성형 코드 논란[7]은 일단락이 되고, 이제 옛한글을 어떻게 처리할지가 남은 이슈가 되었다.

원래 유니코드 방안대로라면, 옛한글은 "초중종 열린 조합"으로 각각 3개씩 최대 9개까지를 모아서 한 글자로 처리해야 하지만, 어느 시스템도 실용적으로 구현하지 못했고, 이 방법으로 옛한글을 지원하는 시스템은 없었다.

대신 유니코드의 PUA(Private Use Area: 사용자 정의 영역)[8]를 이용하여 옛한글을 완성자로 배열하고, 이를 입출력하는 방법이 일반화되었다. PUA에는 초중종 자소 359자를 비롯하여 옛한글 글자마디 5,277자의 옛한글이 배열되어 있다. 일반적으로 PUA 옛한글은 유니코드 현대한글 글자마디와 같이 운용한다.

PUA 옛한글 폰트가 공급된 이후 많은 제품들(ᄒᆞᆫ글워디안(2000), ᄒᆞᆫ글2002-ᄒᆞᆫ글2007을 비롯하여 MS워드2000등)과 프로젝트들이 PUA방식의 데이터를 생성하였다. 최근(2018년 7월)까지도 PUA 방식의 옛한글 자료들이 많이 운용되고 있다.

그러나 유니코드 PUA는 임의로 정의해서 사용할 수 있는 영역으로 해당 시스템에 설치된 폰트나 응용프로그램 및 PC환경(Locale)에 따라 다른 내용으로 보일 수 있는 영역이다. 극단적으로 제2, 제3의 PUA 폰트가 설치되거나, 옛한글 PUA 폰트가 설치되지 않는다면, 해당 자료는 판독할 수 없는 상태가 될 수도 있다. 이 경우 웹서비스도 안정적이지 않아, 보는 기기에 따라 데이터가 올바르게 보이지 않을 수도 있다. 특히 PUA 옛한글은 완성형 특성을 갖고 있어서, 별도의 변환Table 참조를

를 처리할 수 있는 첫 번째 평면을 BMP라고 한다. 첫 번째 평면 글자들은 16비트로 처리할 수 있다.

7) 유니코드 글자마디 11,172자도 완성형이라는 지적이 있지만, 별도의 변환 Table 없이, 수학식으로 자소를 정확히 분리 및 조합할 수 있기에, 이는 완전히 조합형 특징을 갖는다고 할 수 있다.

8) 유니코드 PUA: Private Use Area. 원래 유니코드에는 확장 영역의 PUA(U+F0000-U+10FFFD)도 있으나, 여기서는 16비트로 처리할 수 있는 기본 BMP영역의 PUA (U+E000-U+F8FF)를 대상으로 한다.

표12-유니코드(5.2) 속 여러 종류의 한글 영역

	집합 이름	유니코드 내 영역 이름 코드 범위	설명
(1)	한글자모 한글자모확장 (한글자소: 초, 중, 종)	Hangul Jamo 1100~11FF Hangul Jamo Extended-A A960~A97C Hangul Jamo Extended-B D7B0~D7FB	"첫가끝" 조합 한글자소들 (옛한글 자소 포함) {초성+중성 또는 초성+중성+종성}. 초성 124자 중성 94자 종성 137자
(2)	한글 글자마다	Hangul Syllables AC00~D7A3	현대한글 11,172자
(3)	한글 호환용 자모 (자음, 모음)	Hangul Compatibility Jamo 3131~318E	주로 전각 처리(옛한글자모 포함.) 자음 64자, 모음 29자
(4)	반각 한글 부호	Halfwidth Hangul variants FFA1~FFDC	반각 한글 자소. (코드 호환 영역) 자음 30자, 모음 21자
(5)	사용자정의 옛한글 (PUA 옛한글)	*PUA (Private User Area) E000~F8FF	(비표준) 옛한글 글자마디: 5,277 초중종 자소: 359자 불완전 음절: 22 (()+중+종)

통해서만 자소 분리가 가능하기에 그리 효율이 좋은 방식은 아니라고 할 수 있다.

유니코드 첫가끝 방식의 옛한글 표준이 정착되면 자연스럽게 그 역할을 다했다고 할 수 있을 것이나, 아직까지 많은 데이터가 PUA로 구축되어 있어 한동안 같이 운용될 것으로 보인다. 가능한 한 이른 시일 안에 PUA자료들은 모두 첫가끝 표준형식으로 전환하는 게 바람직하다.

6. 유니코드 첫가끝 옛한글

유니코드에 옛한글 자소가 들어간 것은 유니코드 초기(2.0)부터 들어가 있었으나, 열린 조합 방식으로는 실용적인 시스템으로 구현하는 것이 힘들었다. 특히 3중자소를 자유롭게 결합하고 9개의 자소를 묶어서 글자 하나로 표시하는 것은 현실성이 없는 방식이었다. 따라서 복합자소(특히 3중자소)를 한 개의 자소처럼 운영할 수 있도록 하는 방안이 강구되었고, 이후 유니코드 5.2(2009.10)에서 옛한글 자소 117자가 추가됨으로써 실용적인 옛한글 시스템이 가능해졌다.

표13-유니코드 5.2의 한글자소 수

	한글자모	5.2에서 추가된 (옛)한글자모	자소 수
초성	90	+ 34	124자
중성	66	+ 28	94자
종성	82	+ 55	137자
초+중+종		+ 117	355자

이것은 전혀 별개의 사건으로 시작되었는데, 코드를 자소 단위로 풀고 조립하는 정규화(Normalization) 과정에서 옛한글과 현대한글이 혼용되는 문제가 제기되어 이에 대한 해결책으로 복합자소 코드의 필요성이 생겼고, 이에 따라 복합자소들을 코드로 추가하게 된다. 한글 조합시 '초+중' 또는 '초+중+종' 각 하나씩 최대 3개의 자소만으로 한글을 구성하는 방법을 "첫가끝 방식(첫소리+가운데소리+끝소리)"이라 부르는데, 첫가끝 방식으로 만들 수 있는 한글(현대한글 옛한글포함)은 160여 만자(=124×94×138, 유니코드 5.2 기준)가 가능하다.[9)]

유니코드의 추가된 옛한글 자소들은 고려대학교 비표준문자등록센터의 데이터를 바탕으로 117자가 선정[10)]되었다. 옛한글 자소들이 배치된 영역(Hangul Jamo Extended-A/B)이 몇 개로 나눠진 것은, 유니코드 BMP영역에 글자가 할당되지 않고 비어 있는 영역이 거의 없는 상황이어서 최선의 선택이었다고 본다. 만약 BMP영역에 배정받지 못하고, 16비트가 아닌 확장 영역 쪽에 배정이 되었다면, 자소 단위 처리에서 상당한 비효율이 발생했을 것이 자명하다. 다행히 모든 자소가 BMP영역에 있어서, 각 자소당 16비트로 처리 가능하다.

자소들의 순서가 바뀌어 있어 정렬이 쉽지 않다는 지적이 있다. 그러나 대부분 다른 언어에서도 코드 기준으로 정렬하는 순서가 사전 순서와 맞지 않는다. 영어도 코드 순서와 사전 순서가 다르다.

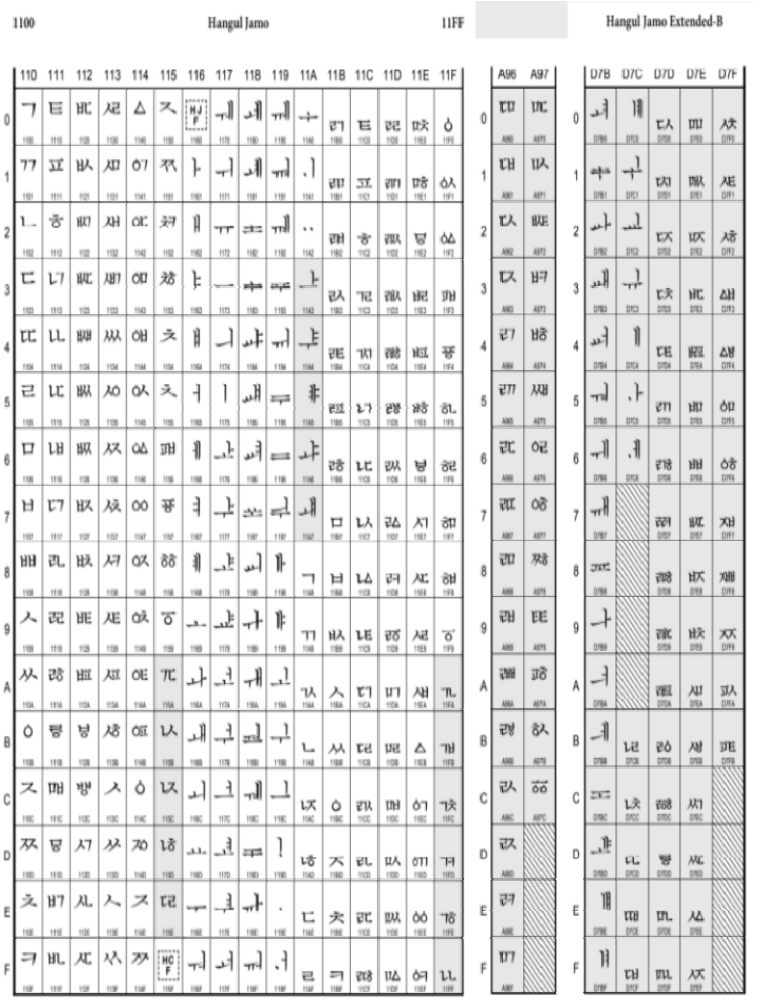
옛한글자소가 확장됨으로써, 쓸 만한 옛한글 시스템이 갖춰졌다는 것은 큰 의의를 갖는다. 실제 국제 표준의 지위를 가진 옛한글코드이기에, 그 데이터들의 영속성이 보장될 것이고, 옛한글 분석 연구 도구들도

9) 일반적으로 '첫가끝 방식'이라 하면 옛한글을 포함한 자소조합 방식을 이른다.

10) 정우봉, 「추가 옛한글 자소 선정과정」, 『21세기 세종계획 문자코드 표준화 연구』(국립국어원 2006), 9-12쪽.

효율적으로 개발 및 고도화될 것이다. 또 여러 기기들, 특히 모바일과 웹에서 옛한글을 안정적으로 지원하는 방향으로 발전할 것이다.

표14-유니코드 한글자소와 확장한글자소 배열



[_] 5.2에서 추가된 자소
그림 출처: www/unicode.org/charts/

Ⅲ. 옛한글코드 운용과 입출력

유니코드에서 옛한글을 사용하는 방식은 “첫가끝 옛한글” 방식과 “PUA 옛한글” 방식 두 가지가 있다.

표15-유니코드 내 옛한글 운용 방법

옛한글 운용 방법	표준
글자마디 11,172자 + 첫가끝 조합 옛한글	표준 규격
글자마디 11,172자 + PUA 옛한글	비표준 규격

두 방식 다 유니코드에서 옛한글을 운용하는 방법이고, 옛한글은 서로 다른 독립적인 영역에 존재해서 두 가지 종류의 데이터가 공존할 수 있다. 그러나 전혀 다른 두 방식의 데이터가 한 개의 문서에 공존한다는 것은 데이터의 일관성을 떨어뜨려 혼재하는 상황이 된다. 즉 글자 모양만 보고는 어느 방식의 옛한글인지 구분이 되지 않고, 검색 정렬 등 데이터 가공에 혼란이 발생하여 올바로 운용할 수 없거나 엄청난 비효율을 감수해야 한다.

KS X 1026-1은 유니코드의 한글 운용 방식에 대해 정의하고 있다. 요약하면, 다음과 같다.

“현대한글은 11,172자 글자마디를 사용하고, 그 외는 첫가끝 방식을 사용한다.”

즉 첫가끝 방식으로 현대한글 처리가 가능해도, 표준 규격은 아니라는 것이다. 또 옛한글은 무조건 첫가끝 방식으로 운용하라는 것이고, PUA 방식은 제외하라는 뜻이다. 옛한글 데이터에 대해 명확히 어느 방식인지 확인할 필요가 있고, 가능하면 표준 규격인 “첫가끝 옛한글” 방식으로 전환하는 것이 바람직하다.

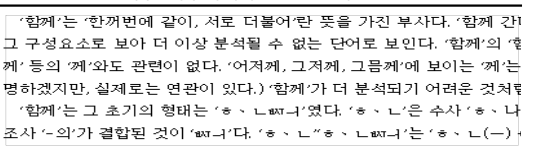
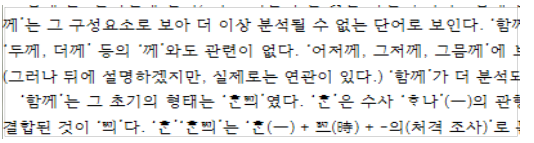
옛한글 관련 작업을 하려면, 우선 첫가끝 데이터가 올바로 나타나는지 확인해야 한다. 옛한글이 제대로 나타내려면 첫가끝 방식의 옛한글을 지원하는 폰트(글꼴)로 설정되어야 한다. 즉 “초중 두 개의 연속된 자소” 또는 “초중종 세 개의 연속된 자소”를 한 개의 글자로 표현할 수 있는 능력은 대부분 폰트의 능력[11]이고, 그러한 처리가 된 폰트로 설정되어야 한다. 이런 폰트들은 개발 비용이 많이 들기 때문에 흔치는 않다.

표16-첫가끝 옛한글을 지원하는 폰트(2018.7월 기준)

배포 업체	폰트 이름
한글과컴퓨터	함초롬바탕, 함초롬돋움
네이버	나눔바른고딕 옛한글
오픈 폰트	은글꼴,
구글, Adobe	본고딕, 본명조 Noto Sans CJK

웹 서비스에 HTML로 구현할 때, 첫가끝이 가능한 폰트를 정확히 지정하면 웹 브라우저에서도 조합된 옛한글을 볼 수 있다. 폰트가 제대로 지원하지 못하면, 첫가끝 자소들이 조합되지 않은 채 나눠져서 보일 것이다.

표17-폰트의 옛한글 지원 여부에 따라 HTML에서 출력 예

	웹브라우저에서 .html 출력 예
첫가끝 옛한글을 표시 못하는 글꼴	
첫가끝 옛한글을 표시할 수 있는 글꼴	

문서편집기에서 옛한글 첫가끝 방식(이하 첫가끝) 지원은 흔글은 2010, MS워드는 2002 확장팩을 통해서 지원한다. 옛한글 편집기로 주로 쓰이는 흔글의 경우, 각 버전에 따라 생성된 옛한글의 코드 방식이 다르다.

표18-흔글(HWP)버전에 따라 생성된 옛한글 코드

흔글 버전	생성된 옛한글코드
HWP 3.0, HWP 96, HWP 97	HNC 3.x 코드 옛한글 (==2.x)
HWP 2002~HWP 2007	유니코드 PUA 옛한글
HWP 2010~(HWP 2018)~	유니코드 첫가끝 옛한글

11) 한글폰트파일(.ttf)에 자소들의 조합 규칙과 조합 위치 정보를 담은 'GSUB 테이블'이 포함되어 있어야 옛한글을 예쁘게 조합하여 표현할 수 있다.

한글97 이전에 생성된 .hwp 파일의 경우 최근 한글버전(2010-2018)은 첫가끝으로 자동변환해서 읽는다. 그러나 2000년 이후 한글2007이전 버전으로 생성된 .hwp파일의 경우, 유니코드 PUA로 된 옛한글 글자마디 데이터는 변환 없이 그대로 읽어들인다[12]. "첫가끝 옛한글"로 변환하고자 하면, HWP 설치할 때에 같이 제공되는 옛한글 변환 도구를 이용해서 변환할 수 있다.

◇ HncPUAConverter.exe (HWP2010-)
:PUA 옛한글 코드(글자마디)를 찾아서 표준방식 첫가끝 옛한글로 변환(File 단위 변환)

표19-한글에서 옛한글 데이터 포함한 .hwp파일 열기

.hwp 파일 / 한글버전	HWP 97 이전 생성한 .hwp파일 (HNC 코드 3.x)	HWP 2007 이하 버전에서 생성한 .hwp 파일 (PUA 옛한글)	HWP 2010 이후 버전에서 생성한 .hwp파일 (첫가끝 옛한글)
HWP 3.0 ~ HWP 97	[HNC코드로 불러오기]	읽지 못함. (유니코드 처리 불가)	
HWP 2002 ~ HWP 2007	*자동변환 :* ▶▶ PUA 옛한글로.	**[PUA로 불러오기]**	[첫가끝 코드 유지] (옛한글 출력은 폰트 따라)
HWP 2010 ~ HWP 2018	*자동변환 :* ▶▶ 첫가끝 옛한글로.	**[PUA로 불러오기]** *(권고:"옛한글컨버터"로 변환)* ▶▶ *첫가끝*	**[첫가끝으로 불러오기]**

※ 참고: 한글 패치 (2018.6 이후)
- HWP 패치에서 "옛한글 변환기"를 Addon으로 메뉴에 등록 가능. 문서에 사용된 PUA옛한글을 찾아서 첫가끝으로 바꿀 수 있음. [PUA 옛한글 찾기], [첫가끝 옛한글로 변환]
- HWP 이외 타 응용프로그램에서 첫가끝 옛한글을 입력할 수 있도록, 옛한글 입력기인 HncIME 2018(윈도우)을 별도로 배포.

12) 한글의 PUA옛한글 변환 정책: "사용자의 의도 없이 임의로 변환하지 않는다."

그림3-훈글2018의 옛한글Addon 메뉴

IV. 표준코드의 의의와 전망

현재 옛한글코드는 국제표준 규격이라는 충분히 좋은 지위에 있다. 이는 비로소 옛한글에 대한 연구가 국제적으로 공유될 수 있음을 의미한다. 약간의 이슈가 있기는 하지만, 비로소 남북한과 중국 일본 학자들까지도 옛한글과 옛 문헌 연구를 공유하고 교류할 수 있는 상황이 되었음을 의미한다. 이전에 PUA 방식으로 작성되었을 때는, 국내용 코드 정의여서, 국제적으로 데이터를 공유하는 데 한계가 있었다.

첫가끝 방식의 의미는 옛한글의 범위를 이미 알려진 옛 문헌으로 한정하지는 않는다. 첫가끝 방식은 이론적으로 약 160만자 정도를 표현할 수 있다. PUA 방식은 알려진 글자들로만 처리하는 반면, 첫가끝 방식은 조합 가능한 글자들 모두가 처리 가능하다. 즉 아직 발견되지 않은 문헌에서 새로 나올 글자나, 또는 문헌상 표기는 없지만 필요한 글자 표기할 수 있다.

옛한글 용례에는 중세 외국어(중국, 몽골, 일본, 여진……) 통역 교재와 개화기 영어 교재 표기 등에도 많은 출처를 가지고 있다. 첫가끝 방식은 중세와 근대 문헌이 아닌 앞으로 연구될 제3세계 외국어(베트남, 태국 등 동남아 및 아랍어, 아프리카어 등) 표기에도 확장할 수 있는 가능성을 연 것이다. 예로 최근의 외국어 학습 교재에도 이러한 표기가 이용되는 경우가 종종 있다. 제3세계 언어를 표기할 때는 문헌에 출처가 없는 옛한글도 얼마든지 나올 수 있다. 즉 기존의 PUA 방식으로는 대응할 수 없는 경우도 첫가끝은 대응할 수 있는 것이다.

또 방언의 경우 기록되지 않은 경우가 많이 있다. 방언은 분명 우리말이

지만, 기록된 자료가 많지 않고, 또 지방 특유의 음가의 경우 현대 자모 표기보다 옛한글 자모로 더 근접하게 표기 할 수 있는 경우도 있을 것이다. 따라서 옛한글의 응용범위를 옛 문헌뿐 아니라 현대와 미래 자료까지 확장할 수 있다는 의미가 있다.

옛 문헌을 디지털화하는 데는 옛한글코드 외 여러 가지 요소가 필요하다. 한자는 물론이고, 방점[13]과 한글 이전의 이두 구결 향찰 등 여러 한국어 표기 등 여러 가지가 있다. 그러나 이두, 구결 등은 거의 표준화 논의가 진행되지 않고 있다. 반면 한자는 한중일 통합한자[14]로 계속 표준에 활발히 추가되고 있다.

옛 문헌을 깊이 있게 연구하려면 가능한 한 많은 정보를 디지털화해야 한다. 옛 문헌에는 문자(?) 외 기호들도 많이 있을 것이고, 이를 기록하는 방법도 필요하다. 일례로 최근 중국은 고전 악보에 표기하는 음계를 표준으로 등록했고, 갑골문의 문양도 등록했다. 우리나라의 옛 문헌들도 문자 외 기호들에 대한 연구와 표준화 노력이 필요하다. 이는 인문분야 옛문헌 전문가들의 연구와 도움이 꼭 필요한 부분이다.

Ⅴ. 맺음말

유니코드가 확장되고 옛한글 표준이 정해진지 약 10년이 지났다. 여러 가지 이유가 있겠지만 아직 옛한글 표준에 대해 잘 알려져 있지 않고, 또 각종 데이터베이스에서 일괄적으로 적용되고 있지도 않다. 아울러 그것이 적용된 제품 또한 많지 않은 실정이다.

이러한 표준방식과 비표준 방식의 공존은 매우 많은 혼란을 일으킨다. 오랜 세월 축적되어 온 데이터라면 더욱 문제가 클 것이다. 이런 혼재의 대표적인 문제가 검색과 정렬의 오류이다. 검색과 정렬이 되지 않는 데이터베이스의 쓸모를 찾기란 어렵다.

옛한글 코드의 표준이 제정된 이후 10년 동안 첫가끝 표준방식의 커다란 단점이 제기된 적은 거의 없다. 다만 비표준화로 인한 다양한

---

13) 방점(입성, 거성)은 유니코드 표준에 포함되어 있다.

14) CJKV 통합한자: 한국, 중국, 일본, 대만, 베트남까지 한자 기록을 가진 나라들이 IRG 위원회 합의를 통해서 유니코드에 한자를 추가하고 있다.

문제점들은 산발적으로 제기되어 왔다. 비표준 코드를 지원하는 각종 프로그램을 사용할 때 생기는 당연한 오류, 비표준 코드를 표준 코드로 전환할 때 생길 수 있는 데이터 오류 등이 그것이다. 그러나 이러한 문제는 극복 가능한 것이며 극복해야 할 것들일 뿐이다. 옛한글 표준이 하루 빨리 정착되어야만 전 세계의 많은 기계에서 자연스럽게 옛한글이 구현될 수 있을 것이다. 표준방식을 이용한 옛한글 데이터와 이에 대한 연구 성과물들이 많아지길 기대한다.

참 고 문 헌

1. 단행본

김병선,『컴퓨터 자판 옛자모 배열 연구』. 국립국어연구원, 1993.

김홍규,『한글 옛글자의 컴퓨터 처리 효율화 방안과 이에 대한 고시조 데이터베이스 시스템 개발 연구』. 한국어전산학회, 1992.

정우봉,『21세기 세종계획 문자코드 표준화 연구』. 국립국어원, 2006.

홍윤표,『한글 코드에 관한 연구』. 국립국어연구원, 1995.

2. 논문

이승재,『한국어 정보 자료의 구축과 활용 방안 연구』. 서강대학교 박사학위 논문, 2003.

3. 기타 자료

KSC 5601-1992 정보교환용 부호계, 2바이트 조합형 부호계, 1992

삼보컴퓨터,『한글 비디오 보드(고어) 사용설명서』. 1991

양왕성,「옛한글 코드와 표준코드」. 훈민정음학회(강연자료), 2018.

유니코드, www.unicode.org/charchart/, Unicode 11.0 Character Code Charts.

한글과컴퓨터,「흔글 사용자 메뉴얼」. 1993.

국 문 초 록

현재 우리가 쓰는 대부분의 IT 기기에서 한글은 매우 효율적인 입출력 구조를 가지고 있어, 크게 불편함을 느끼지 않는다. 반면 옛한글은 아직 실용적인 수준으로는 미흡하며, 많은 기기들이 옛한글 규격을 지원하기 위해서는 시간이 더 필요한 상황이다.

옛한글을 지원하기 위한 규격은 1980년대부터 있어 왔고, 몇 차례의 큰 변화를 겪으며 옛한글 코드도 국제문자코드에까지 반영되었다. 한글 코드들에서 옛한글을 지원하기 위한 어떤 노력들이 있었는지 살펴보고, 최근의 옛한글코드에 대한 표준 규격과 올바른 이용에 대해서도 살펴본다. 앞으로 많은 기기들이 옛한글코드를 지원하여, 옛한글들도 현대한글만큼이나 혼란 없이 자연스럽게 쓸 수 있는 환경이 되길 기대한다.

투고일 2018. 6. 20.
심사일 2018. 7. 15.
게재 확정일 2018. 8. 27.
주제어(keyword) 유니코드(Unicode), 옛한글(old Hangul), 현대한글(modern Hangul), 옛한글코드(old Hangul code), 현대한글코드(modern Hangul code), 조합형 한글 자모(conjoining Hangul Jamo characters), 한글 완성형 글자마디(precomposed modern Hangul syllables)

Abstracts

A Study on Changes in the Old Hangul Code

Yang, Wang-sung

Modern Hangul is very efficient to input and output on most IT devices. On the other hand, the input of old Hangul is not sufficient and it is not practical to input and output on the devices. More work is required to facilitate the use of old Hangul on many devices. The standard code of old Hangul has been on the system since the 1980s. A few rounds of changes have occurred to the standard and the changes are reflected in the International Character Set. My study examines the trajectory of changes in the old Hangul code as well as the standard code and its correct use. Many devices will need to support the old Hangul code in the future. I hope that old Hangul will be made as seemless as modern Hangul to use on the system.