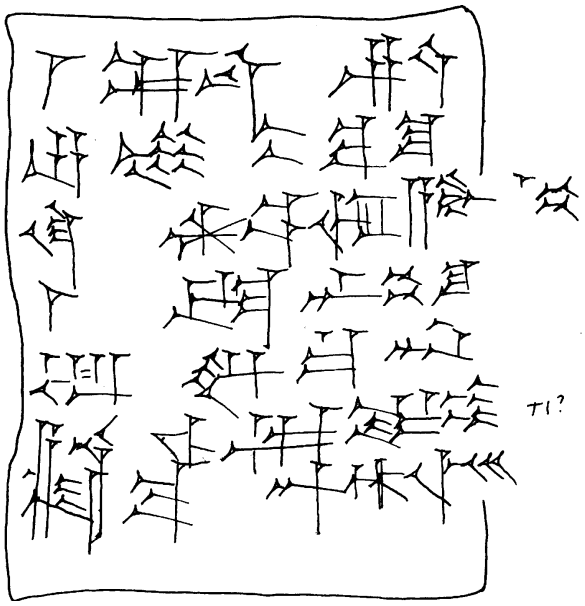
The Computer Representation of Cuneiform:

Towards the Development of a Character Code

1. Traditional Approaches to the Study of Cuneiform

The information recorded in cuneiform documents can be studied either directly, through first hand examination of originals, or indirectly, by means of second-hand *representations*. At the present time, such representations follow one of two approaches: (1) pictorial, *i.e.* photographs and hand copies, or (2) interpretative, *i.e.* transliterations (Ger. 'Umschrift') and transcriptions (Ger. 'zusammenhängende Umschrift'):

Pictorial **Interpretive**



1. 1 m a - n a k ù - b a b b a r
2. k ù n a m - t a b - b a - š è
3. k i ^{d}*šamaš* ù *a-bu–wa-qar*
4. ^{n}*ip-qu–*^{d}*šala*
5. ù *im-gur–*^{d}*šamaš*
6. *a-na ši-ib-qá-at-*[*ti*]
7. š u - b a - a n - t i - m e - e š

Figure 1

Each of these approaches conveys one aspect of the information to the detriment of the other: the first provides a relatively precise reflection of the form of an original while leaving the matter of interpretation to the reader, whereas the second reports content, but implies form only to a limited degree (and in the case of transcription not at all):

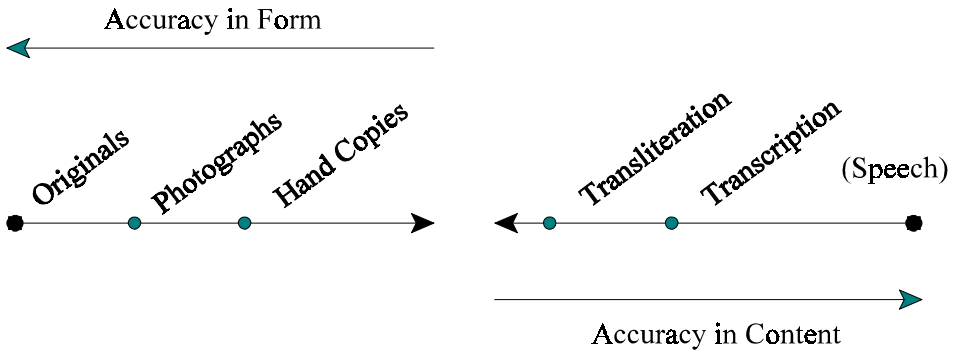


Figure 2

As can be seen from Figure 2, there is presently no second-hand means of representation in widespread use which addresses both form *and* content. However, the technology presently exists to design and implement a representation of information which would bridge this gap, specifying at once the *essential form* of cuneiform characters as they appear in a given document (thus taking account of geographic, temporal, and genre considerations), and their *identity* as elements of the writing system. Furthermore, by means of a suitable input mechanism, character form and character identity could be recorded alongside character *interpretation* with little more effort than is presently expended in typing traditional transliteration. These three aspects of cuneiform information are illustrated for clarity in Figure 3:

1.	One character (**AN**) in two *forms*:		(early)		(late)
2.	Two separate character *identities* sharing one form:		(**PA**)		(**BANMIN**)
3.	One character and form with two different *interpretations*:		(**UD** = /babbar/)		(**UD** = /tam/)

Figure 3

Note that in general, the traditional system of transliteration is able to communicate the information on line 3 of the figure; what is needed is a mechanism for communicating the information on lines 1 and 2.

2. Character Code Basics

The key to the solution is the development of an internationally recognized character code for cuneiform. Briefly, a character code is a standard which assigns to each core symbol of a given writing system a unique identifier in a one-to-one relationship. In the context of computer encoding, these identifiers are numbers, and characters are represented within computer systems by these numbers (their so-called *internal* representation). With the help of some additional information (style and so on), characters are converted for display on computer screens and hard copy in the manner familiar to us, or in any other manner desired (so-called *external* representation). The following figure illustrates this relationship by means of the ASCII code which underlies much of modern computer science:

Some Western European Characters and Their ASCII Internal Equivalents

0	1	2	3	A	B	C	D	a	b	c	d
48	49	50	51	65	66	67	68	97	98	99	100

Figure 4

3. Fundamental Issues in the Development of a Cuneiform Character Code

The first step in the development of a character code is to identify all the core symbols in the system, that is, all its characters. On the assumption that the code should serve the needs of the entire cuneiform community, irrespective of period and area of interest, the compilation of characters must be done *comprehensively* and *inclusively* across time periods, geographic areas, genres, and so on. This in itself is fairly straightforward: however, there are a number of non-trivial issues which arise from such an approach which must be addressed. A few of these will be discussed briefly further on by way of example.

Once all characters—to the extent they are known—have been identified, they can be

assigned representative numbers as discussed above according to some agreed-upon scheme. In character code development, this is generally done in such a fashion as to accommodate a notion of 'natural sort order', concerning which more will be said below. For the moment, notice in particular the relationship between the order of ASCII characters and the values assigned as their internal representation as per Figure 4.

The second step, without which the character code would be of limited value, is the encoding of *essential form* as an additional dimension to the system. For each character identified and encoded in the first step, one must compile a register of the *significant* variants to be recognized by the system, to each of which a secondary or *variant* code will be assigned, again according to some practical overarching scheme. The results might in part look something like this:

Character & Variant Code	Character Identified	Form Identified	General Comments
[121, 3]	**AN**		Early 'plenary' form
[121, 4]	**AN**		Later 'standard' form
[299, 1]	**2**		Common 'standard' form
[300, 1]	**A**		Common 'standard' form
[300, 2]	**A**		Cursive form of the previous

Figure 5

Note that the first two items are recognized and encoded as identical characters (here assigned an arbitrary internal representation of 121), but different significant variants; similarly for the last two items (assigned an internal representation of 300). Secondly, note that the middle item (the

character for '**2′**'), even though it shares an identical external form with the last item (the cursive variant of '**A**'), is encoded as an entirely separate character (with an internal representation of 299), because in terms of historical development, it is a separate character.

It is critical, at this juncture, to make one matter *absolutely clear*. To say that provision will be made for encoding *essential forms* of characters is *not at all* the same as saying that it will be possible to encode *all possible occurring shapes* of characters individually, for at least two reasons: (1) computer codes are inherently finite, that is, they can accommodate only a finite number of possibilities, and infinite variation in detail cannot be represented by finite means; (2) it is the function of a character code to identify *significant* differences inherent in a script (*i.e. underlying forms*), and not to describe accurately all specific instances of characters (*i.e. surface representations*). Thus, the two forms of **AN** and the two forms of **A** in Figure 5 must be treated and encoded as distinct, since they differ fundamentally on a structural level in terms of wedge count and wedge arrangement. On the other hand, the specific angles at which the diagonal wedges of an early **AN** are placed is irrelevant to the identification of the character or its variant form. Similarly, the *presence* or *absence* of hash marks (*gunû*) is significant in determining character identity, whereas the specific *number* of hash marks probably is not.

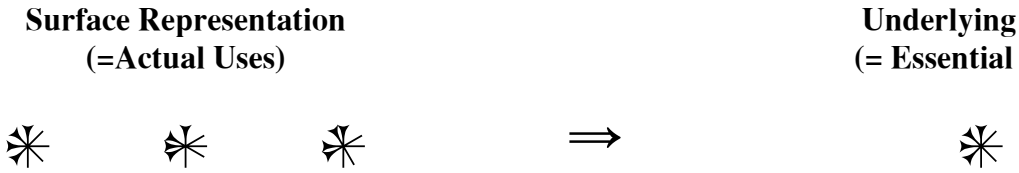


Figure 6

4. Implementation Considerations

Thus far, this presentation has focussed upon describing the ideal design structure of a character

code as it emerges naturally from a comprehensive analysis of the cuneiform script over the course of its lifespan. The obvious and hence best encoding—on the theoretical level, at least—is to represent each significant variant of a given cuneiform character internally as a pair of codes of the form [<character no.>, <variant no.>]. The next logical question is 'How is this theoretical structure best translated into practical reality?'

It is of course clear that a competent programming team can, given the time and resources, design a system as complex and suited to the particular need as one may like. The problem is that such time and resources probably do not exist, and even if they did, the result would be a proprietary system with limited portability, if not in terms of computer architectures and operating systems, then at least in terms of external applications able to process the encoded text. In other words, this option should be considered as a last resort only.

A compromise option exists in the form of a standard one-dimensional character code with variants identified by mark-up tags. This may work, so long as one appreciates that variants *cannot* be neatly parcelled up according to external criteria such as time period, geography, genre, or even scribal tradition, since multiple variants of characters are frequently employed within a single text; the tagging mechanism must take this into account. However, this solution is, from a systems design point of view, at best inelegant since it masks the inherent structure of the data, which in turn would likely translate into an awkward system in practice. It is preferable to restrict external tagging of any kind to purposes fundamentally external to the data, that is, cosmetic concerns of a non-structural nature such as choice of hollow-head vs. full-head forms, wedge styles, and so on.

Fortunately, a third option is on the horizon. Unicode, a recognized international body

whose goal is to encode all writing systems of the world, is presently considering the adoption of a character and variant code system similar or identical to the one presented above, on the basis of past experience with other character sets such as Han characters, which are used for writing Chinese. A Unicode-compliant encoding would ensure across-the-board portability not only in terms of computer architectures (Apple, IBM, and so on) and operating systems (Windows, Mac, Linux), but also at the level of application programmes and the Internet, since many application tools are already Unicode-compatible or are expected to be so in the near future.

The benefits of such wide portability are obvious: one would be able to encode, display, and manipulate cuneiform data directly from one's favourite word processor and transmit them using standard protocols—without having to resort to graphics, which are space intensive— as well as use all the tools already at our disposal for our own native tongues, such as searching and replacing, spell-checking, sorting, dictionaries, and so on.

5. Sample Non-trivial Issues Arising from a Comprehensive Encoding

As alluded to earlier, the encoding into one system of a script with such a long and varied development as cuneiform gives rise to certain non-trivial issues which must be addressed. A few will be outlined here by way of example, though any discussion in detail must necessarily be relegated to another time and place.

a. Mergers

Certain characters, originally distinct, merged over time (*cf.* **BAR**, **BÁN**, and **MAŠ**, which all eventually converged in the form of original **MAŠ**, and ceased to be regarded as independent characters; see Figure 7). With respect to the encoding of texts written *beyond* the historical merger point, there are two ways to proceed, each with advantages and consequent trade-offs.

(1) One may continue to encode separate characters but allow for sharing one external shape through the variant mechanism. Under this scenario, comparisons of encoded text across the merger point will successfully identify comparable usage (*e.g.* early **BAR** would match later **BAR**, early **BÁN** would match later **BÁN**, and so on). On the other hand, these characters would *never* match each other, not even beyond the merger point (*e.g.* searches for **BAR** would not find **BÁN**, and so on).

(2) One can disable the characters which no longer have an independent existence by means of the input mechanism, and force merged characters to be encoded in the same way (*e.g.* all three characters would share a **MAŠ** encoding). Under this scenario, the merged characters would be identical in encoded text and hence match each other (*e.g.* searches for any of **BAR**, **BÁN**, or **MAŠ** would all yield the same results). However, when making comparisons with texts whose composition *precedes* the merger point, merged characters would *all* match the one original character with which they were ultimately identified, regardless of usage (*e.g.* all later characters would match early **MAŠ**, even though they might actually represent usages formerly represented by **BAR** or **BÁN**).

b. Splits

In some cases, characters whose variants were all originally used for the same purpose came over time to be used for quite separate purposes. For example: in Neo-Assyrian Royal Inscriptions, one version of the character **TA** was used for the syllable /ta/, whereas a surviving Middle Assyrian variant of **TA** (usually transliterated as **TA***) was used as a logogramme for /issu/ (see Figure 7). The situation here is almost—though not quite—a reversal of the previous one, with similar implications for encoding.

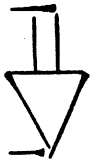
Mergers:

(BAR) 

(BÁN)  ⇒ 

(MAŠ) 

Splits:

 ⇒ (TA) 

(TA*) 

Note: with the exception of TA*, all characters are taken from Labat's *Manuel*, 5th edition.

Figure 7

c. Sort Order

Default sort order in a computer system is defined in terms of the internal representations assigned to characters, in that characters are sorted according to the numerical sequence so defined (recall Figure 4). In the case of small character sets such as those used for Western European languages, the sequence is assigned so as to reflect the essentially arbitrary conventions already established historically, which are memorized by language speakers. In the case of large character sets, such as Han/Chinese, it is preferable to establish sort order according to some general organizing principle such as character shape in order that the memory not be overburdened when it comes to looking up characters in lists and dictionaries; internal

representations are then assigned accordingly.

The traditional practice with cuneiform has been to follow the second approach, and to order signs according to wedge-count and wedge-arrangement. This has worked well so long as one restricted one's self to a single script tradition (*e.g.* Neo-Assyrian, Old Babylonian), though it has made the compilation of common ground resources awkward.

There are at least two obvious difficulties with continuing this tradition, without at least a few modifications:

- a sort order based upon such an organizing principle will differ depending upon the script tradition followed, and only *one* default sort order can be defined; sort orders for all other script traditions would have to be accommodated secondarily through look-up index tables and the like;

- it is doubtful whether any script tradition exists in which *all* characters across the system are attested at least once, meaning that some auxiliary sorting principle would have to be defined to accommodate characters not attested within the chosen script tradition.

For this reason, the question of assigning a default order to cuneiform characters must be considered carefully and objectively, in order to maximize the benefit to the field as a whole. It may be of value to point out here that secondary look-up tables are *already* provided for within Unicode, so that if the Unicode route is followed, it will be possible to establish special sort orders by script tradition where desired, obviating the need to use the default order for this purpose. This frees up the default order for any other useful purpose upon which the cuneiform community can agree.

6. Linking the Character Code to Character Interpretation

It was suggested above that character identity and essential form could be recorded alongside character interpretation with little more effort than presently required to type standard transliteration. The procedure would be a simple one; the following is intended to serve as an

illustration of how this might be accomplished.

First of all, before entering text, certain control parameters could be set, according to which expected variant frequency tables would be loaded and matched to certain standard keying sequences, on a character by character basis within the context of a given script tradition. In most cases, standard transliteration inherently specifies character identity, and in the case of mergers and splits and so on, ambiguity would be resolved by the control parameters already specified. Thus, entering a transliteration through a specially designed input mechanism could easily provide both character identity and character interpretation.

A suitably located scroll window could display character variants in order of expected frequency, with the highest frequency choice preselected, which would be the variant to be encoded if no further action is taken. Continuing on with the next transliterated character would cause the chosen character and variant combination to be encoded and registered in one file, and the corresponding transliteration in another. In the event that a less frequent variant is required, it would be a simple matter to use the arrow keys to scroll through the display window until the appropriate variant is selected before proceeding.

The advantages of such a data entry system are quite clear. Among other things:

- text and transliteration editions can be produced quickly and efficiently;
- because the scroll window provides a visual equivalent of the transliteration, errors such as the inadvertent specification of the wrong subscript index can be minimized;
- texts and transliterations are guaranteed to be in complete agreement with each other;
- the input mechanism can be designed to be heuristic, meaning that it could learn from experience by counting variants and re-evaluate expected variant frequencies for a given corpus. This means that accurate statistics concerning the corpus would always be automatically available; also, upon conclusion of corpus entry, the accumulated data could be used to generate automatic character lists and so on; and finally,

- given some reasonable preliminary work in setting up frequency tables for given corpora or script traditions, it would be a simple matter to feed existing transliterations through a translator to produce preliminary versions of texts in the new format. It would then only be necessary to scan each text to verify that the appropriate variants have been encoded, and to correct these where necessary.

7. Conclusion

The foregoing analysis and proposal is the result of several years' consideration of the problem of representing cuneiform text on modern computer systems. In its present form, the proposal is a joint effort between myself, a former Analyst Programmer and currently a doctoral student in Assyriology at the University of Toronto, and Lloyd Anderson of Ecological Linguistics, a professional linguist specializing in character codes who is also a member of Unicode. Together, we have launched the Computer Representation of Cuneiform Project, whose goal is to perfect this proposal, and, with the support of our respective communities, make a substantial contribution to its realization.

Rosemere, Québec

July 2000

We welcome the participation of scholars with an interest in the encoding of cuneiform. Interested parties are invited to visit our website at

www.imprimus.ca/~jenniedavis/crc.htm

where additional technical details about the proposal are available as well as a comprehensive description of the project and the project team. We can also be reached by email at

kfeuerherm@hotmail.com (Karljürgen Feuerherm) and

ecoling@aol.com (Lloyd Anderson).

Unicode has also kindly made available to us a discussion group which can be subscribed to by submitting a request to

cuneiform-request@unicode.org

Details are available at the Unicode homepage, www.unicode.com.