A COMPUTER-GENERATED TRANSLITERATION SYSTEM TO ASSIST THE TEACHING OF READING IN HEBREW TO NATIVE SPEAKERS OF ENGLISH

by

Curtis Rice
B.A., University of British Columbia, 1986

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ARTS (EDUCATION)
in the Faculty
of
Education

SIMON FRASER UNIVERSITY
August 1988

APPROVAL

Name: Curtis Rice

Degree: Master of Arts (Education)

Title of Project: A Computer-generated Transliteration System to Assist the Teaching of Reading in Hebrew to Native Speakers of English

Examining Committee:

Chairman: Jaap Tuinman

A. J. Dawson
Senior Supervisor

W. Rothen
Instructor

K. Toohey
External Examiner

Date Approved August 17, 1988

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

A Computer-generated Transliteration System to Assist the Teaching of Reading in Hebrew to Native Speakers of English

Author: _____
(signature)

Curtis RICE
(name)

August 17, 1988
(date)

1. Abstract

This paper examines the problems presented to speakers of languages using the Roman alphabet when learning another language using different alphabetic symbols. The case is taken of native speakers of English learning Hebrew, which presents particular problems in reading because Hebrew moves from right-to-left and either does not represent most vowels or uses a system of diacritic marks to represent them which is difficult to read. The hypothesis is advanced that the learning-to-read process for those literate in their first language will be faster and more efficient if the target language is presented together with transliteration into the Roman script in the early stages of reading. This allows for faster reading because the literate student can automatically decode the Roman text and project an understanding of pronunciation onto the Hebrew. It also allows for more extensive reading and hence the accumulation of a larger vocabulary which is key to advanced functioning in the target language. A computer-based system and tutorial has been designed to assist in making this possible. The system automatically generates transliteration into Roman of any Hebrew text while the tutorial facilitates the transfer of decoding understanding from the Roman to the Hebrew script. This system and tutorial are described in detail together with suggestions for their application.

TABLE OF CONTENTS

LIST OF FIGURES AND APPENDICES

LIST OF FIGURES AND APPENDICES (ctd)

LIST OF FIGURES AND APPENDICES (ctd)

1. The Problem

Acquiring a second language is a formidable task to anyone who has not grown up bilingual, regardless of age. An additional significant barrier, particularly to reading, has to be overcome by native speakers of English if the second language is represented by some other script than the Roman alphabet. This paper looks particularly at the problem presented to native speakers of English in learning Hebrew with its own distinctive script which, although alphabetic, is radically different from the Roman alphabet. The same problem applies to native speakers of other languages using the Roman script learning not only Hebrew but also other languages with a totally different alphabetic non-Roman script, e.g. Arabic or Persian. A different set of problems - not examined here - is presented in learning languages like Chinese and Japanese whose scripts are not alphabetic.

Although a high level of competence in a second language can be achieved with oral learning only, advanced competence comparable to that of a literate native speaker can be reached only by becoming literate oneself. Those already literate in their first language find themselves back at the prereading stage they experienced as young children. They have to learn to read all over again in an unfamiliar script.

As in reading their first language, no significant progress can be made until they achieve decoding automaticity: the ability to instantly convert print to sound. This paper

addresses the problem of achieving precomprehension automaticity in reading the non-Roman script. Once students can instantly identify words without belaboured decoding, their cognitive energy is freed to concentrate on the task of comprehension.

The specific problems presented by the Hebrew script are:

1. Reading is from right to left instead of left to right.
2. The Hebrew alphabetic characters bear no resemblance to the Roman characters and, additionally, have a right-to-left orientation which is opposite to the left-to-right orientation of the Roman characters.
3. Modern Hebrew is written with only two of some twelve vowel forms represented. You have to know a word to read it. The Hebrew alphabet does not code sound; rather, with its consonantal letters, it structures mnemonics.
4. A system for representing vowel sounds exists. It was developed by the Jewish scholars (Masoretes) in Tiberias around the seventh or eighth century AD, in order to preserve the sound of the sacred books of the Hebrew Bible. The Hebrew Bible is still printed with this Tiberian vowel system which also is used in modern Hebrew dictionaries and grammars. However, it is a system of diacritic marks applied to the consonants (see Section 5(b)) and bears no resemblance at all to the Roman system of representing vowels with independent letters. It is accordingly most difficult to read without extensive practice.
5. Modern Hebrew uses a cursive script for handwriting which

bears rather less resemblance to the printed characters than Roman lower case bears to Roman upper case. This presents an additional problem in reading handwriting: letters, notes, instructions etc. including, when learning Hebrew, written instructions from the teacher, both on the blackboard and on pieces of paper. Although the transliteration system described in this paper is applied to a printed Hebrew text, it could equally well be applied to a "handwritten" cursive script by developing a special font for this purpose.

2. Significance of the Problem

If a literate native speaker of a language using the Roman alphabet learns a second language also using the Roman alphabet, then that speaker starts off with the essential precondition for reading - decoding automaticity -- already met. He or she does not have to struggle with the basic task of decoding visual symbols to sound: from the very beginning the arrangements of letters convey the sounds of words to the brain. The student is free to apply all his cognitive energy to the essential tasks of recognizing and identifying words to facilitate comprehension (Durkin, 1970, p 109). However, when learning a language like Hebrew which uses an altogether different alphabet, the student is back in Grade 1, faced with the task of mastering basic decoding skills all over again. This imposes a significantly lengthy delay in mastering reading in the second language.

The question may be asked: does this really matter? After all nothing is preventing the oral learning of the language. In second language learning today the emphasis is usually laid on first learning the spoken language and coming to reading and writing later. There are two answers to this: 1) although reading may not be emphasised, there is the assumption when both first and second languages use the Roman alphabet that the student of the second language can decode, i.e. understand the shape and sound of what is written on the blackboard and printed in texts and handouts. In learning Hebrew, this

assumption cannot be made. 2) It is generally agreed that one of the essential keys to mastery of a second language is the development of as large a vocabulary as possible. Typically, native speakers of modern Western languages will have a use-and-recognition vocabulary of between thirty and sixty thousand words (Mario Pei, 1952). It is, of course, many years before the second language learner can even approach this level. His or her first objective in learning to read in English as a second language at the first elementary level of independence is to acquire a recognition vocabulary of about 3,600 words (Rivers and Temperley, 1978). From this point he can take off on the task of building and maintaining his own vocabulary. It is reasonable to assume a similar objective for Modern Hebrew. The student of Hebrew, however, is handicapped in achieving quickly this basic vocabulary level because decoding monopolizes the best part of his early reading efforts. It is true that he will achieve a survival level of vocabulary without reading. Reading, however, would enable him to achieve it quicker. Even more important, without extensive reading the student will not advance to the level necessary to cope with classes at university or technical school or to gain any sort of literary appreciation of the language. The difficulties of the script present the main barrier to reading.

A large research literature exists on various aspects of learning to read and improving reading performance. J. S. Chall (1967) in her landmark book Learning to Read:the Great Debate

brought together the findings that indicated a division between the tasks of decoding and comprehension with the recommendation of an emphasis on decoding rather than on meaning for beginning reading instruction. Chall (1983) followed up her previous work and advanced a Piagetian-type theory of reading stages. Stage 1, Initial Reading or Decoding she characterises as "grunting and groaning" and "barking at print". Stage 2, Confirmation, Fluency, Ungluing from Print is a key make-or-break stage: its outcome should be automaticity in decoding (see also Durkin, 1970) which marks the end of "learning to read" and the beginning of "reading to learn". The child who does not succeed in this stage remains a "problem reader". Chall comments with great relevance to our present topic:

> A tenable hypothesis would be that Stage 2 (leading to decoding automaticity) is the main failing point of most adult literacy campaigns. The literacy campaigns here and in Third World countries indicate that although most adults can get through stage 1 (initial decoding), they begin to falter at Stage 2. Reading a newspaper ... which requires at least Stage 3 reading (comprehension), will be difficult or impossible for most ... After the literacy classes complete their Stage 1 programs there are not enough readable materials available, material that is familiar in its use of language and content, for the new literates to gain the fluency of Stage 2. Nor is there usually a compelling need to keep on reading (pp 41,42).

Chall's description applies to many immigrants to Israel who, though literate in their first language, do not persist in achieving the same literacy in the new language. They fall short of decoding automaticity and remain in Stage 2.

From Stage 3 on, the emphasis moves from decoding to comprehension. Now the acquisition of vocabulary becomes of crucial importance. Sternberg, Powell and Kaye (1983) compare three methods for teaching vocabulary skills: rote learning, the keyword method and learning from context. They find that rote learning is the least effective method; that the keyword method has merit; but that learning from context produces the greatest depth of long term understanding. The student can learn from both oral and written contexts. An oral context, however, has the disadvantage of transitoriness - the student may or may not "get it". A reading context, on the other hand, is permanent, always available and can be repeated as often as necessary. Therefore reading is the royal road to the acquisition of vocabulary.

An additional problem associated with reading difficulties is the inability to efficiently use dictionaries and reference material in the second language. This also slows down the rate of progress which the student could otherwise achieve and contributes to his or her general sense of frustration in dealing with the written word.

All of these problems together have a significant psychological effect. Second language learners who are used to

reading and using written references with ease in their first language are faced first with frustration with their new, unaccustomed illiteracy, and then with surprise and discouragement upon finding just how long it takes to regain the literacy they have for so long taken for granted. As a result, many do not go beyond a very basic social proficiency in the second language and neither explore its literary depths nor achieve the level of competence necessary for continuing education with the second language as the language of instruction. Even more important than this, they become functional illiterates in the new society with all the disadvantages that this implies, particularly in the workplace where, more and more, literacy is a precondition for holding down a job, and even more so for advancing in it.

3. Research

A study that bears directly on the problems posed by a strange alphabetic script is reported by Lee Brooks and Amima Miller ("A Comparison of Explicit and Implicit Knowledge of an Alphabet" in Kolers, Wrolstad and Bouma, 1979). For this study two lists of twelve words were used, one of which is shown in Figure 1 (Figures are listed just before Appendices). The words on the right are non-alphabetic: there is no correspondence between the symbols and the sounds of the associated response (condition 1). The words on the left are alphabetic but must be decoded from right to left (condition 2). One of the groups of eight McMaster undergraduates tested was told that the words would be both alphabetic and non-alphabetic and were given training in the six-letter alphabet used, followed by a whole word training procedure in the actual words used. Immediately after the training they were subjected to 160 trials, each trial consisting of a presentation of six cards from one condition or the other with the request to identify all six words as rapidly as possible. Five trials were given in one condition followed by five in the other, alternating up to a total of eighty trials for each condition. After every ten group trials the list of words were presented in random sequence to the respondents individually. They were then transferred to a similar list of twelve new words and the whole training and testing procedure was repeated for another thirty group trials which were interspersed in the same way

with individual trials. All responses were timed. The results were surprising. Brooks and Miller report:

> For both kinds of timed trials and for both lists the words for which the underlying alphabet were known were identified significantly less rapidly than were the non-alphabetic words. This effect and its persistence are remarkable for several reasons. First is the strictly technical reason that the lists were carefully matched and exactly the same stimuli and responses were used across subjects in the two conditions. Second, the experiment was carried out within subjects, and since all of the subjects reported noticing that they performed faster on the non-alphabetic conditions, this might have prompted them to adopt a more efficient strategy for the alphabetic items. Third, the training procedure would have allowed the subjects to treat both sets of words identically ... Finally, the effect continues into the second list, at which point one might have expected either increasing facility with the alphabet or a shift in strategy to have markedly attenuated the disadvantage of the alphabetic items (p. 394).

Brooks and Miller go on to comment that although one source of the disadvantage for the explicit alphabet is the difficulty of applying unfamiliar symbol-to-letter or symbol-to-sound correspondences, there is more to it than this. All the improvement gained progressively for the alphabetic items in

the first set of lists was lost when the second list was presented. They cite another experiment using artificial letters described in Brooks (1977) which indicates that subjects organize their encoding around particular information in a word even though more distinctive information may be available. Quite striking differences between letters are not exploited by the explicit alphabet subjects until after considerable practice.

The key words here are "after considerable practice". Brooks (1977) used a glyphic alphabetic list (the letters arranged into words in the form of a unitary pattern) and a discrete alphabetic list presented both left to right and right to left (see Figure 2). He also used in one experiment a glyphic alphabetic list and and a glyphic non-alphabetic list.

Taking only six words, it took 180 practice trials before performance on the discrete alphabetic list became significantly faster than on the non-correspondence list. Performance on the glyphic alphabetic list became faster than on the discrete alphabetic list after about 60 trials but faster than on the glyphic non-correspondence list only after 200 trials. Interestingly, in the practice trials referred to, the explicit alphabet items were printed left to right, proving that a reverse order was not the cause of the difficulties experienced although, if used, it could have added to them.

The task of dealing with these trials with a six letter alphabet and a six word vocabulary can be compared with the

task of dealing with a Hebrew alphabet of 22 letters with almost no vowels, or with vowels organized in a different system than the consonants, and an unlimited vocabulary. Clearly, the achievement of automaticity in decoding a strange alphabet, even for a subject highly literate in his or her first language, is not a trivial task.

The difficulties experienced in the Brooks (1977) study are particularly significant because, as can be seen from Figure 2 (listed just before Appendices), the letters used for the alphabet were clear and bold and looked like "real" letters, unlike the squiggly shapes in Figure 1. Brooks and Miller comment: "Apparently a visual difference as robust as the one shown in Figure 2 was not exploited by the explicit alphabet subjects until after considerable practice" (p.397). This observation, combined with the greater success with the glyphic alphabet, indicates that some sort of word pattern recognition appears to be involved in successful decoding, that the patterns are not immediately obvious in a strange alphabet and that they take a considerable period of time to recognize.

A recent research study has directly addressed the question of representing Hebrew with the Roman alphabet: Romanization to Facilitate the Teaching of Modern Hebrew to Adult Native Speakers of English by E. P. Kellogg, Jr. (1983). This study evaluates five research projects - one unpublished thesis and four published papers - which address various issues concerning the Romanization of the Hebrew alphabet. Kellogg

makes the point in his introduction that during the 1800's in Europe the Roman alphabet was used to introduce students to Hebrew, that the Turkish dialect of Arabic was Romanized in the time of Mustafa Kemel Ataturk and that recent movements in this direction include the Romanization of Chinese by means of the Pin Yin system. He speaks of the advantages of Romanization in making Hebrew (and other languages) more accessible and more easily included in the curriculum, and the advantage of the left to right representation which is the norm in international communications (increasingly using the computer) and also the usual mode of representation of the language of science and mathematics. The conclusions reached from his review of the five studies are favourable towards Romanization and include the following specific observations of relevance to this paper:

1. More is involved than just learning 22 new letters. The learner has to look in different places for clues to the meaning of a letter. The recognition techniques that work in English do not work for reading Hebrew because the letters are formed with a right to left orientation. Roman letters, naturally, are formed with the opposite orientation.
2. The problem of spelling in Hebrew is complicated by the fact that some of the written letters have lost their uniqueness with the result there are two letter "t's", two letter "k's" and two letter "s's". Romanization eliminates this confusion.
3. Certain words experience vowel changes or deletions in the stem when making the transition from singular to plural. This

is not apparent in the Hebrew script but becomes transparent upon Romanization (e.g. Hebrew KTN,KTN'M; Roman "katan", "ktanim"). Kellogg has produced the Shalom Home Study Course in Conversational Hebrew (1983) which he describes as "a research and development product designed to demonstrate how Romanization can be employed to facilitate the acquisition of Modern Hebrew by the adult native speaker of English" (p. 31). It is not clear whether or not his intention is to permanently replace the traditional Hebrew script with the Roman alphabet for a second language learner. Such is not the proposal of this paper but rather to use Romanization as a bridge of easy entry to Hebrew and then to make the transfer to the traditional script as soon as the student has gained some facility with the language.

This strategy is, in fact, used by Thomas O. Lambdin in his Introduction to Biblical Hebrew (Harvard, 1971). Lambdin comments in his preface:

> The generous use of transliteration is meant to serve three purposes: to enable the student to perceive Hebrew as a language, and not an exercise in decipherment; to remove the customary initial obstacle, wherein the student was required to master innumerable pages of rather abstract phonological and orthographic details before learning even a sentence of the language; and to facilitate the memorization of the paradigms, where the essential features are, in my opinion, set in greater

relief than in the conventional script.

Some two thirds of the way through the book Lambdin weans the student from transliteration and uses only the traditional script.

Of the multi-lingual word processors researched, the one selected for the system described in Section 5 is Multi-Lingual Scholar, Version 3.0 (MLS) (1987). This supports 16 languages structured upon five alphabets: Roman, Hebrew, Cyrillic, Greek and Arabic. Other alphabets are available and the package includes a font generation utility for designing custom fonts. It is possible to move from one language to another within the same document so that a true multi-lingual text can be processed. Section 5 is written with Multi-lingual Scholar. A number of keyboard layouts are provided under software control and additional keyboard layouts can be designed by the user. A particularly valuable feature of this processor is its ability to handle Hebrew vowel pointing by overstrike characters so that they appear under or over the characters on screen and in print. MLS is configured for the IBM XT and requires 512K memory for the application described in Section 5. Another powerful package specially designed for academics is Nota Bene Version 3 (1987). Despite its many strengths, however, it displays the Hebrew vowel points as separate characters on screen although it treats them as overstrikes in printed output. This makes it unsuitable for designing a screen tutorial. Another candidate is Gutenberg (1985) which has

particularly strong text-formatting features but is only configured for the Apple II series of computers. A future possibility is a multi-lingual word processor under development for the Xerox personal computer work station called Star. This is discussed in a Scientific American article (July 1984). The project is an ambitious one, aiming at no less than an integrated system for encoding, typing and rendering all the world's graphic systems including those which are not alphabetically based. It will presumably be some time before a microcomputer version is available.

Different Hebrew transliteration conventions were researched. These are described in context in Section 5.

4. Objectives

Transliteration has been used for many years in teaching classical Hebrew to help students negotiate the initial problems of the unfamiliar Hebrew script (Lambdin, 1971; Weingreen, 1939). More recently, it has been successfully used in teaching modern Hebrew (Kellogg, 1983). It is widely used in publications from popular periodicals to scholarly works which quote Hebrew for non-Hebrew speakers. Transliteration is, therefore, of proven value. It is, however, troublesome and time-consuming to produce extensively, which has limited its use.

What follows is the documentation of a system, using a personal computer, that automatically generates transliteration from any given fully-pointed Hebrew text. This system makes possible a computer-assisted tutorial, also documented, for using transliteration to speed and facilitate the acquisition of reading competence in Hebrew.

The objectives in producing this system and tutorial are as follows:

1. prove that computer-generated transliteration, using a personal computer, is feasible;

2. provide the means of delivery of transliteration in any form desired;

3. reduce the non-Hebrew speaking student's long-lasting frustration and slowness in decoding Hebrew text, which becomes a barrier to persevering until reading competence is achieved;

4. encourage the transference of decoding automaticity from the transliterated to the Hebrew text and thus accelerate the acquisition of reading competence in Hebrew without

transliteration support;

5. indicate, by achieving success in a transliteration system and tutorial for Hebrew, that the same approach is feasible for learning to read in other non-Roman scripts.

Teaching classical Hebrew emphasises reading competence. Teaching modern Hebrew to non-Hebrew speakers rightly emphasises oral communicative competence. Once a survival level is achieved, the student tends to drop his or her studies. True communicative competence, however, depends upon substantially enlarging vocabulary which, in turn, depends upon extensive reading in a second language as in a first. Only with reading and an enlarged vocabulary will students move towards the communicative competence of the educated first language speaker. At the same time they will move towards an understanding of the literature of the language and towards the ability to undergo formal education with the second language the language of instruction. The aim of a transliteration system is to help achieve this goal by facilitating the transfer of the student's first language decoding automaticity to the second language and by encouraging him or her to persevere with reading until this takes place.

5. Description of Transliteration System

5(a). Overview

Purpose

The purpose of the system is to produce: 1) computer-generated transliteration of Hebrew texts for use in the initial teaching of both classical and modern Hebrew to native speakers of English; and 2) a computer-assisted reading tutorial on screen to help the student attain more quickly a mastery of reading Hebrew script. The reading tutorial can also be supplied, if required, as hard copy.

Process

The Hebrew source text is prepared, complete with vowel points, using the Multi-Lingual Scholar (MLS) word processor. It is transliterated into a Roman text file, using the MLS table-driven Configurable Text Interchange Utility (CTIU). This requires the design of tables of some complexity to produce transliteration that implements the pronunciation rules for Hebrew. The Hebrew and Roman text files are then merged by the author-designed computer program BITEX into a bitextual file with a line of Hebrew alternating with a line of Roman so that each Hebrew word has its Roman counterpart immediately beneath it. The Roman text is invariably longer than the Hebrew text. The Hebrew words are spaced out to allow for this. Eight additional files of the merged text are prepared, using a cloze procedure. In the first of these, every seventh Roman word is deleted; in the second an additional seventh word is deleted; in the third an additional seventh word again is deleted and so on until with the seventh file the Roman text has been erased. The eighth file erases the Hebrew vowel

points. The cloze procedure can be implemented with a cloze interval less or greater than seven. It is also possible to implement any combination of cloze stages, e.g. deleting every seventh word starting with word 1 and 4, or with word 1, 3 and 6 etc. A teacher may, therefore, use it as a flexible reading or writing test instrument.

Intention of tutorial

The bitextual file is produced to enable students to decode immediately the sound of the Hebrew word from the Roman transliterated word immediately below it and thus read the text without difficulty. The purpose of the series of cloze files is to enable students to repeat the reading while gradually transferring their understanding of the pronunciation directly to the Hebrew text. By the time they have completed the file series they should be reasonably comfortable with the Hebrew text without the transliteration assist. If not, they can repeat the reading exercise from the beginning or from any point in the middle as often as they desire.

System components

The system has three main components: alphabets, tables and a computer program. Four sets of alphabets and text interchange tables are provided to produce four alternative transliteration modes: classical, classical alternative, modern and simplified modern. The classical mode is that used in such classical grammars as Lambdin (1971) and Weingreen (1939). It relies on diacritic marks to distinguish different pronunciations of the same letter, whether consonant or vowel, and also to indicate the underlying Hebrew spelling. The classical alternative and modern modes use

phonetic spelling of consonants rather than diacritic marks but retain diacritic marks with the vowels. They also identify the underlying Hebrew spelling. The simplified modern mode does not distinguish different letters which have the same pronunciation and avoids diacritic marks. Therefore it does not identify Hebrew spelling. It is easiest to read but contains least information.

The computer program BITEX not only merges the Hebrew and Roman text files and produces the cloze files but also has a module for applying transliteration rules as required. The system relies primarily on the CTIU tables to generate transliteration. There is, however, one rule - short qametz in a final unaccented syllable - which has not been put into the tables because it takes up too many lines and causes the tables to exceed their capacity. This rule is applied by the computer program. If other exception conditions come to light, the computer program can be used to deal with them if they cannot be easily accomodated by the tables.

The sections which follow describe first the Hebrew alphabet and pronunciation rules; and then deal in turn with the three main components of the system: the alternative Roman alphabets, the alternative text interchange tables and the computer program.

5(b) Description of Transliteration System: The Hebrew Alphabet

The Hebrew alphabet consists of 22 consonants. It was derived in the tenth century B.C. from the Phoenician branch of the original alphabet which was invented in the Middle East sometime during the first half of the second millennium B.C. Originally, no vowels at all were indicated in writing. In the pre-Exilic period, before the Babylonian Exile in 587 B.C., the consonants י y, ו w, and ה h, were used at the end of a word to indicate final vowels. In the post-Exilic period, י and ו were used also to indicate vowels inside a word. Not until the 9th and 10th centuries A.D. was a complete system of vowel notation perfected by Jewish scholars known as the Masoretes (traditionalists), working in Tiberias. This system, using diacritic marks beneath and above the consonants, was superimposed on the earlier partial system.

Hebrew is written from right to left. Consonants and vowels are listed separately below. Alternative transliterations are discussed in Section 5(d).

1. The Consonants

Form			Name	Transliteration	Pronunciation
	א		'āleph	'	glottal stop or zero
בּ	ב		bêth	b b̲ bh v	[b] as in bait [v] as in have
גּ	ג		gîmel	g ḡ gh (g)	[g] as in go [g] as in go
דּ	ד		dāleth	d d̲ dh (d)	[d] as in door [ ] as in this
	ה		hē	h	[h] as in house
	ו		wāw (vāv)	w (v)	[w] as in wet or [v] as in vet
	ז		zāyin	z	[z] as in zone
	ח		ḥêth	ḥ	[x] like the ch in Scottish loch
	ט		ṭêth	ṭ	[t] as in time
	י		yôdh	y	[y] as in yes
כּ	כ	ך	kaph	k k̲ kh ch	[k] as in king [x] as for ḥêth
	ל		lāmedh	l	[l] as in line
	מ	ם	mêm	m	[m] as in moon
	נ	ן	nûn	n	[n] as in noon
	ס		sāmekh	s	[s] as in soon
	ע		'ayin	'	no Eng equiv; back of throat
פּ	פ	ף	pē	p p̄ ph f	[p] as in pay [f] as in face
	צ	ץ	tzādhê	ṣ tz	[ts] as in hits
	ק		qôph	q	[q] a 'k' at back of throat
	ר		rêsh	r	[r] as in rope
	שׂ		śîn	ś	[s] as in sing
	שׁ		shîn	š sh	[š] as in show
תּ	ת		tāw (tāv)	t t̲ th (t)	[t] as in time [ ] as in thin

Note 1: Three forms are shown in the first column of the table: the dagesh with a dot in the middle, the regular, and the sofit for the final letter of a word.

Note 2: Six letters have the dagesh form: בּ גּ דּ כּ פּ תּ (b g d k p t), known mnemonically as the "begadkepat" letters. Without the dagesh they are soft or spirant; with the dagesh they are hard. In modern Hebrew, three of these letters, גּ דּ תּ, are no longer pronounced softly. They take the hard pronunciation as shown in parenthesis in the transliteration column of the table. The dagesh can also be used with these and other letters to indicate doubling. When used with ה h, the dot is known as "mappiq" (bringing out) and indicates that the "h" is sharply audible. These points are discussed in Section 5(c).

Note 3: Five letters have a special form at the end of words: ן ך ף ם ץ (d m n p tz). They do not change their pronunciation.

Note 4: Pronunciation is the same today in classical and modern Hebrew except for the letters דּ גּ תּ, aspirated in classical but not in modern Hebrew; the letter וּ, which is "w" in classical but "v" in modern Hebrew; and the gutturals ע א whose pronunciation is retained only by some native speakers.

2. The Vowels

Sign	Name	Transliteration	Length	
בַ	pathaḥ	a as in had	short	
בֶ	s^{e}ghôl	e as in bed	short	
בֶי	s^{e}ghôl & yod	ê as in bed	short	
בֶה	s^{e}ghôl & hē	e as in bed	short	(final hē only)
בִ	short ḥîreq	i as in lid	short	
בֻ	qibbûtz	u as in bull	short	
בָ	qāmetz-ḥātûf	o as in top	short	
בָ	qāmetz	ā as in yard	long	
בָי	qāmetz & yôd	â as in yard	long	(rare with yôd)
בָה	qāmetz & hē	ā as in yard	long	(final hē only)
בֵ	tzērê	ē as in they	long	
בֵי	tzērê & yôd	ê as in they	long	
בֵה	tzērê & hē	ē as in they	long	(final hē only)
בִי	long ḥîreq	î as in machine	long	
בּ	shûreq	û as in flute	long	
בֹ	ḥōlem	ô as in note	long	
בֹ	ḥōlem	ō as in note	long	
בֹה	ḥōlem & hē	ō as in note	long	(final hē only, rare)
בְ	shewa, sheva	e as in haven	reduced	
בֲ	ḥaṭef-pathaḥ	ă as in abridge	reduced	
בֱ	ḥaṭef-s^{e}ghôl	ĕ as in haven	reduced	
בֳ	ḥaṭef-qāmetz	ŏ as in atom	reduced	

Note 1: Lambdin (1971, p XVII) points out that there is no consensus as to the length and quality of the Hebrew vowels in the Masoretic system. The diacritic marks used in the above

transcription are designed to reflect Hebrew spelling and are not always accurate length indicators.

Note 2: The distinction between long and short qametz (ā or o) is discussed in Section 5(c) on special pronunciation rules.

Note 3: Syllables are either open or closed. A syllable begins with a consonant and cannot begin with a vowel. An open syllable consists of a consonant and a vowel, e.g. בָּ; a closed syllable consists of a consonant and a vowel followed by another consonant, e.g. בָּר or בּוֹר.

Note 4: Words are stressed on the last syllable (most frequently) or on the penultimate syllable. Only penultimate stress is marked in this transliteration system, both in preparing source Hebrew text files, using an overstrike accent mark, e.g. מֶ֫לֶךְ, and in transliterated Roman files, using a raised period, e.g. me·lekh.

5(c). Description of Transliteration System: Pronunciation Rules

Pronunciation rules for Hebrew will be considered under the following headings:

1) dagesh lene, dagesh forte and mappiq

2) mobile and quiescent sheva

3) long and short qametz

4) effect of gutturals

5) redundancies.

6) metheg

1. Dagesh Lene and Dagesh Forte

Dagesh (דָּגֵשׁ, 'piercing') is a dot in the heart of a letter, as in the דּ of the Hebrew word just used. It may be either Dagesh Lene (weak) or Dagesh Forte (strong).

Dagesh lene is used with six letters only, which may have a hard or soft pronunciation. They are תּ, פּ, כּ, דּ, גּ ,בּ (b, g, d, k, p, t), known mnemonically as the "begadkepat" letters. Without the dot, these six letters are aspirated (bh, gh, dh, kh, ph, th). With the dot they become hard (b, g, d, k, p, t). In modern Hebrew, although the dagesh lene is preserved for all six letters in written usage, it applies only to פּ ,כּ ,בּ (b, k, p) in speech. Transliteration needs to reflect this. Therefore different font files are used for classical and modern Hebrew (needed also to distinguish vav (וֹ,w) in classical and vav (ו, v) in modern Hebrew).

The dagesh in a begadkepat letter is dagesh lene when that letter commences a syllable in the beginning or middle of a word, providing that there is no vowel before that letter. This means that the dagesh is always dagesh lene at the beginning of a word; and in the middle of a word is dagesh lene if it follows a quiescent

sheva. Quiescent sheva closes the syllable to which it belongs, thereby excluding the possibility of a vowel preceding the begadkepat letter. This rule is given effect by Roman table rules R1 and R2 (see Section 5(b)).

In all other circumstances, the dagesh in any letter, including the begadkepat letters, is dagesh forte, which has the effect of doubling the letter. The Hebrew Table treats all instances of dagesh as dagesh forte. The Roman Tables change dagesh forte to dagesh lene where required (Roman table rules R1 and R2).

The gutturals (א, ע, ה, ח) and resh (ר) do not take the dagesh. The dot in the middle of ה is not a dagesh. It is called "mappiq" (bringing out) and indicates that the "h" is sharply audible. This occurs infrequently and always at the end of a word.

2. Mobile and Quiescent Sheva

The two points under a letter (לְ) are called "sheva". Sheva is either not sounded (quiescent) or is sounded (mobile) like the phonetic character shwa. Mobile sheva is, therefore, a reduced vowel. It is represented by "ᵉ". Sheva is quiescent in the following cases:

(a) At the end of a word, whether there is one sheva or two, e.g. נֶפֶשׁ (nep̄ṭ), אַתְּ (att).

(b) When it is the first of two shevaim following each other in the middle or at the end of a word, e.g. כַּסְפֶּכֶם (kaspᵉk̲em), תִּסְפְּרוּ (tispᵉrû).

(c) After the short vowels (a, e, i, o, u) e.g. שֻׁלְחָן (šulḥān).

(d) After a long accented vowel, e.g. לֵכְנָה (lē·k̲nā).

The Hebrew table initially identifies quiescent sheva (rules H7 to H9) and represents it by an arbitrary ASCII code. This is because

the Roman tables need to identify quiescent sheva for the application of other rules: the conversion of the default dagesh forte from the Hebrew table to dagesh lene following quiescent sheva (rule R1); the correction rule R5 whereby a normally quiescent sheva becomes mobile when attached to the first of two similar letters, e.g. מִתְפַּלְלִים (mitpalelîm) (in this case, the sheva would normally be quiescent because it follows a short vowel); and, finally, identifying quiescent sheva at the end of a word where it may have been missed by the Hebrew table (rule R7).

The Hebrew table applies one rule for mobile sheva (rule H6). This is the case where the sheva under the first letter of a word is mobile while, if there is a second, it is quiescent. This rule is applied ahead of the rules for quiescent sheva. Otherwise Hebrew table rule H7, which makes the first sheva quiescent of two shevaim together, would incorrectly interpret the sheva under the first letter of a word as quiescent. Hebrew rule H6 is complemented by Roman rule R6, which makes the second sheva in such circumstances quiescent instead of mobile.

All other shevaim are mobile by default.

3. Long and Short qametz

The vowel sign qametz (ָא) is pronounced either as long a (ā) or short o (o). The qametz in a closed unaccented syllable is short. A closed syllable can be identified as one ending in a quiescent sheva or followed by a dagesh forte or followed by the binding hyphen, maqqef, or closed by a vowel-less letter at the end of a word. The qametz is also short if appearing before a "furtive" qametz (ָא), which is a sheva changed to half-sheva, half-qametz under the gutturals א ע ה ח to assist articulation; or if

appearing under a guttural which precedes a letter with sheva, the sheva under these circumstances being quiescent.

This pronunciation rule is partly implemented by Hebrew table rules H2 to H5 (short qametz before maqqef (hyphen) and before a letter with furtive qametz) and Roman table rule R3 (short qametz before dagesh forte). The rule R3 relies upon the Hebrew table to mark dagesh forte and upon the preceding two Roman table rules to convert dagesh forte to dagesh lene where required so that no illegal dagesh fortes are left.

One ambiguous case remains: the appearance of qametz before a letter with sheva. If the sheva is mobile (sounded) then the qametz is long as the syllable is open. If the sheva is quiescent then the qametz is short as the syllable is closed. Only a person with knowledge of the language can determine whether the sheva is mobile or quiescent and hence whether the qametz is long or short. Weingreen (1959, page 13) gives as an example the word אָכְלָה ('oḵlāh, 'food') where the sheva is quiescent and the qametz short and the word אָכְלָה ('āḵᵉlāh, 'she ate') where the sheva is mobile and the qametz long. The ambiguity is resolved by inserting a metheg (bridle) to indicate long qametz. The metheg is a signal to the reader to pause thus making the syllable open so that אָכְלָה will be read as 'āḵᵉlāh. The uses of metheg are discussed under point 6 below.

It is therefore important when preparing the Hebrew text to add metheg to qametz wherever the syllable with qametz is to be read as open. This is one of the uses of metheg in the Massoretic system. This makes it easy to apply Hebrew table rule 8 which automatically treats qametz as short before a letter with sheva and

makes the sheva quiescent. The insertion of metheg blocks this rule in which case the qametz is long by default.

All cases of qametz not identified as short by the table rules are long by default.

One case of short qametz is not identified by the tables: short qametz in the last syllable of a word closed by a letter without a vowel. This is because an excessive number of table entries is required which would create more table entries in total than MLS-CTIU supports at this time. Identifying a last letter is costly in terms of number of entries because of the number of end-of-word signals that must be tested: space, carriage return, period, comma, semi-colon, colon, question mark, exclamation mark and hyphen. There are 22 letters to test against these nine end-of-word signals which creates 198 table entries. The total table capacity is 600 entries. This one rule would use almost one third of the table capacity. This last syllable condition is therefore tested in the BITEX program which merges the Hebrew and Roman files into one bi-textual file. It should be noted that relatively few cases of short qametz in the final syllable are encountered because most Hebrew words carry their accent on the final syllable and short qametz does not appear in accented syllables.

4. Effect of Gutturals

The gutturals א, ע, ה, ח , being pronounced deep in the throat, have certain peculiarities. These have two effects on table rules:

(a) They do not take dagesh forte and are therefore not subject to doubling.

(b) When they are the terminal letter in circumstances where a

vowel normally would not follow, they attract an extra vowel, pathaḥ (אַ), which is pronounced before, not after, the guttural. Thus רוּחַ (wind) is transliterated rûaḥ, not rûḥa. This pronunciation rule is implemented by Roman table rule R4.

(c) Instead of simple mobile (vocal) sheva they take composite sheva, the so-called "furtive" vowels, furtive pathaḥ, furtive qametz and furtive seghol, חֲ חָ חֱ . A furtive qametz following a consonant with qametz has the effect of making that qametz short even though its syllable is not closed. This is reflected in Hebrew table rules H3 and H5.

(d) A point in the middle of ה h is called mappiq (bringing out), not dagesh, and indicates at the end of a word that the h is sharply audible, not silent. Although doubling is not indicated, this usage of hē, which is infrequent, is distinguished by doubling (hh), following the transliteration usage recommended by the Society for Biblical Studies.

5. Redundancies

In the earliest Hebrew inscriptions (10th cent B.C.) no vowels at all were indicated. The need for some vowel representation was, however, felt. In the pre-Exilic period (before the fall of Jerusalem in 587 B.C. and the Babylonian Exile) three consonants were used to indicate final vowels: ו w was used for û, י y was used for î, and ה h was used for any other final vowel. In the post-Exilic period י and ו were used additionally as vowel indicators inside a word with an extension of the values represented: י now being used for ê, ệ, or î and ו for û or ô. The letter ה h continued only at the end of a word to represent any other vowel. (Lambdin, 1971)

The use of וֹ was formalized as וֹ for û and וֹ for ô. These two vowels therefore became directly represented in the written script. The only problem for transliteration is to distinguish between וֹ as û and וֹ as וֹ with the dagesh point. This is done with MTS by having these two vavs represented by different ASCII codes and configured on different text entry keys. Care is required when preparing Hebrew text to use the right keys.

The task remains in transliteration to distinguish when ּ and ה are used as consonants and when as vowels and therefore redundant for transliteration purposes. Jewish scholars in Tiberias during the 9th and 10th centuries, called the Masoretes or Traditionalists, perfected the system of vowel notation now used. They superimposed their system on the crude system for vowel representation already described. This resulted in: 1) the use of ּ with the points for short i and e which become lengthened (î, ê) and with the points for long e and a (ê, â); and 2) in the use of final ה h with short e (e) and long e, a and o (ē, ā, ō). The transliteration for classical Hebrew, following Lambdin, is designed to not only indicate vowel length but also to reflect Hebrew spelling. Hebrew table rules H11 to H15 distinguish between redundant ּ, i.e. a vowel marker, ּ as a consonant, which it is when it has a vowel point under it and ּ as a semi-vowel at the end of a word when it needs representing in order to indicate a diphthongal effect. Hebrew table rule H16 eliminates redundant ה at the end of a word. Authorities differ in transliterating final-h. Lambdin retains it. Weingreen does not. This system follows Lambdin and does <u>not</u> apply rule H16 on the grounds that final -h is needed to indicate Hebrew spelling and, in the case of

the "eh" ending, "eh" is clearer than "e" alone which has the association in English of not being sounded but changing the value of the preceding vowel as in "rim", "rime" or "ton", "tone".

6. Metheg

The metheg (bridle) is a short vertical stroke placed under a consonant and to the left of the vowel sign if there is one. It serves a number of purposes including an indication that the reader must pause. In this way it acts as a check and causes a secondary stress. It is, in fact, sometimes used as a stress marker. It is not a reliable indicator of stress, however, because of other purposes that it serves. Therefore stress is indicated in this system, following Lambdin and Weingreen, by an independent primary stress marker above the letter (e.g. מֶ֫לֶךְ).

Other uses of the metheg are in marking anomalies, e.g. the retention of ā and ē two or more places before the main stress instead of their usual replacement by ᵉ, as in בֵּרַכְתַּ֫נִי (bērak̲ta·ni). More important from the viewpoint of transliteration is its use with qametz to indicate when either a long or short reading should be retained in any doubtful position. The use of metheg to give a long reading to qametz has been discussed under point 3 above, taking the examples of אָ֫כְלָה ('āk̲ᵉlāh, 'she ate') and אָ֫כְלָה ('ok̲lāh, 'food'). An example of metheg giving a short reading to qametz is given by Lambdin (1971, page XXVIII): אָ֫הֳלוֹ ('ohŏlô, his tent).

Because of variabilities in the use of metheg in different texts, its use as a marker for transliteration in this system is avoided except in Hebrew table rule H3 where it is retaining a short reading of qametz (see example at the end of the preceding

paragraph). The general rule (short qametz before a guttural carrying a furtive qametz) is covered by Hebrew table rule H5. This rule will not work, however, if a metheg is in the text because metheg is not included for checking in the rule. Hence the addition of rule H3. As qametz is read as long by default after checking for all rules for short usage, there is no need to use metheg as a check for long qametz in the table rules.

Metheg is not represented in the Roman text. It is converted to a null code by the tables. It is not necessary to insert it in the Hebrew text to get correct transliteration except in the case of indicating a long qametz before a letter with a mobile (sounded) sheva as discussed under point 3 above.

The following authorities have been consulted in summarising the above pronunciation rules for Hebrew: Lambdin (1971) and Weingreen (1939) for classical Hebrew; Livny and Kokhba (1973) and Talmage et al (1977) for modern Hebrew. The author is solely responsible for the table rules that implement these pronunciation rules in the BITEX computer-generated transliteration system.

5(d). Description of Transliteration System: Roman Alphabets & Keyboard layout

Transliteration can be achieved either through text interchange tables or through direct programming. This system relies, at present, primarily on text interchange tables run in conjunction with the MLS Configurable Text Interchange Utility (CTIU). However, regardless of the method used for the basic transliteration process, the ASCII codes that represent the transliterated characters must be given specific alphabetic expression. Here, choice must be exercised. This system offers four alternative Roman alphabets with additional choice between fixed and proportional spacing.

The four basic alphabets are: classical (C); classical alternative (CA); modern (M); and simplified modern (SM). These alphabets are shown in Appendix 2, in relation to the Hebrew alphabet and following the order of the Hebrew ASCII codes; and in Appendix 3, following the order of the Roman ASCII codes. The differences between these alphabets and their common keyboard layout will be discussed in this section.

1. Keyboard Layout

The four alphabets have a common keyboard layout. This is a modification of the default Hebrew map file used by MLS which is, as far as possible, a phonetic/mnemonic representation of the Hebrew alphabet rather than the standard Hebrew typewriter layout. Thus "a" = aleph, "b" = bet etc., while "w" = shin (ש) because it looks like shin even though the sound is different. The policy in designing the keyboard layout for the Roman alphabets is to configure on each key the Roman equivalent of the Hebrew

character assigned to that key on the MLS Hebrew keyboard. Lower case represents normal Hebrew letters. Upper case represents Hebrew letters with dagesh (point in middle of letter). The Alt keyboard represents Hebrew sofit (final) letters. The Control keyboard represents a few Hebrew characters that cannot be conveniently accomodated elsewhere (see Appendices 2 and 3).

Several keys are free on the Alt and Control keyboards after all MLS Hebrew characters have been assigned. These have been used for special purposes in the Roman key assignments. The Control keyboard is used for vowel assignments corresponding to the MLS Hebrew vowels, which are assigned as overstrikes on the function keys. The Alt keyboard is used for an overflow of vowels and also for configuring a standard version of characters that can change on the standard keyboard for the different alphabets, i.e. the "begadkepat" letters plus shin, sin, tzadi and vav. This is to provide a necessary constant required in the construction of the text interchange tables.

2. "Classical" Alphabet

The Roman alphabet for transliterating classical Hebrew follows the transcriptions used by Lambdin (1971) and by Weingreen (1939). The only difference between these two is that Lambdin provides additional transcriptions for qametz with yod, בָי â, and seghol with yod, בֶי ê. Our "classical" alphabet follows Lambdin in these particulars. Its characteristics are:

(a) The aspirated form of the "begadkepat" letters is distinguished by a line beneath or above the letter: b̲ ḡ d̲ k̲ p̄ t̲.

(b̲) Diacritic marks are used with the vowels to both indicate length and reflect the Hebrew spelling. For example, ê and ē both

represent the long form of the vowel "e" but ê indicates the spelling ֵי while ē indicates the spelling ֵ. These contrast with ę and e which both represent the short form of the vowel "e" with ę indicating the spelling ֶי and e indicating the spelling ֶ.

(c) The letters shin, sin and tzadi, שׁ שׂ צ, are represented by single characters with diacritic marks, š ś ṣ rather than the phonetic spellings sh, s, tz. Sin, שׂ ś, has the same sound as samekh, ס s, but is distinguished from it by the diacritic mark.

(d) Tet and taw (tav), ט ת ṭ t, have the same sound but are distinguished from each other by tet carrying a diacritic mark.

(e) Het and kaf, ח כ, have a similar guttural sound [x], like the "ch" in the Scottish "loch" but are distinguished in transcription: het ḥ and kaf k̲.

(f) The gutturals alef and ayin, א ע, are represented by the light breathing ' and the rough breathing ‘.

Note that this transliteration reflects Hebrew spellling.

3. "Classical Alternative" Alphabet

The "classical alternative" alphabet differs from the "classical" alphabet described above in two respects:

(a) The "begadkepat" letters are represented by the phonetic spelling form, bh gh dh kh ph th, rather than by single consonants with a diacritic mark

(b) Shin and tzadi are represented by the phonetic spellings sh tz, rather than by an "s" with diacritic mark. The form tz is chosen rather than ts because of the association of ts at the end of a word with the English plural form.

Note that this transliteration still reflects Hebrew spelling. It is the same as used by Cassuto (1983) except for shin and tzadi

which are transcribed in Cassuto as š and ṣ.

4. "Modern" Alphabet

The "modern" alphabet differs from the "classical alternative" in two respects:

(a) It represents the aspirated form of three only of the "begadkepat" letters, בּ כּ פּ (bh kh ph).

(b) It represents vav וֹ as v rather than w.

Note that this transliteration still reflects Hebrew spelling.

5. "Simplified Modern" Alphabet

The "simplified modern" alphabet differs from the "modern" alphabet in a number of repects. Its aim is to transcribe in the simplest way possible, avoiding diacritic marks and not attempting to indicate differences in Hebrew spelling.

(a) bet בּ is represented by v rather than bh.

(b) pē פּ is represented by f rather than ph.

(c) kaf כּ is represented by the more familiar ch as in Scottish "loch" rather than by kh.

(d) ḥet ח is retained as ḥ as in this case a useful distinction in spelling can be made with a familiar character.

(e) No distinction is made between sin, שׂ s and samekh, ס s.

(f) No distinction is made between tet, ט t and tav, ת t.

(g) The vowels are represented by a e i o u without diacritic marks for indicating length and spelling. The form ᵉ is, however, retained as it usefully represents the reduced vowel sheva with a familiar symbol.

This alphabet is the same as that used in Reif and Levinson (1965) except for ḥēt and tzadi which Reif and Levinson represent as x and c. Provided a student knows Hebrew pronunciation, this

alphabet is adequate for transliteration that accompanies the Hebrew text.

6. Fonts

Two types of font are used: screen fonts and print fonts:

Screen Fonts

R08TRA.S8, R08TRB.S8, R08TRM.S8, R08TRS.S8

Regular Print Fonts

R82TRAFC.PFT, R82TRBFC.PFT, R82TRMFC.PFT, R82TRSFC.PFT

R09TRAFC.PFT, R09TRBFC.PFT, R09TRMFC.PFT, R09TRSFC.PFT

R09TRAPC.PFT, R09TRBPC.PFT, R09TRMPC.PFT, R09TRSPC.PFT

Special Print Fonts

R09TRABC.PFT, R09TRAMC.PFT, R09TRASC.PFT

Legend

R = Roman

08 in screen font = 8 pixels wide

82 in print font = copied from 8 pixel screen font and resized by 2

09 in print font = point size

TR = transliteration font

A = classical alphabet

B (AB in special print font) = classical alternative alphabet

M (AM in special print font) = modern alphabet

S (AS in special print font) = simplified modern alphabet

F = fixed space P = proportional space

C = 24 pin printer

The screen fonts are fixed-spaced, eight pixels wide. The Roman screen fonts match the spacing of the MLS Hebrew screen font HB8.S8. In this way, perfect registration is assured when lining up Roman with Hebrew words on the screen.

The R82.... Roman print fonts have been copied from the Roman screen fonts for printing purposes and match the Hebrew printing fonts HB82.... which have been copied from the Hebrew screen font. This makes it possible to print the Hebrew/Roman text file with the Roman words lining up exactly with the Hebrew words. This is not possible with proportionally spaced printing. The screen font produces letters in print smaller than the same letters on screen. Therefore, the print fonts derived in this way have been resized to approximately the size of the 8 pixel screen fonts.

The R09 fixed space Roman print fonts, as also the H09 fixed space Hebrew fonts, are nine-point fonts derived from the corresponding proportionally spaced fonts. When used to print a bi-textual file they have the advantage of emphasising the Hebrew over the Roman in thickness of type while preserving exact line up of the two texts.

The R09.... proportionally spaced Roman print fonts, as also the MLS Hebrew print fonts, HB09ANPC (ancient or classical) and HB09MNPC (modern), are nine-point fonts. The Hebrew fonts are standard MLS fonts (modified only by the addition of an overstrike stress marker) where A or M indicates ancient or modern, N indicates "normal", P indicates proportionally spaced and C indicates 24 pin printer. When used to print Hebrew from a mixed Hebrew/Roman text, they have the advantage of emphasising the Hebrew over the Roman in thickness of type but the disadvantage of not registering exactly the Roman with the Hebrew words. This lack of registration does not, however, prevent effective reading of these printouts.

The R09TRAB, TRAM & TRAS fonts are special printing fonts

for use with the MLS configuration file used with the Roman "classical" table (RTRA.TBL). The MLS configuration file provides for a 40 col screen font, an 80 col screen font, a keyboard map file and six printing fonts, of which #1 is the default, for each of five alphabets. Four configuration files (TRA.CNF, TRB.CNF, TRM.CNF, TRS.CNF) have been prepared for use with the four Roman tables which produce the transliterated text. The default printing font is the one that matches the Roman transliteration table used: R09TRAPC for the "classical" table, R09TRBPC for the "classical alternative" table, R09TRMPC for the "modern" table and R09TRSPC for the "simplified modern" table.

In the case of the TRB, TRM & TRS configuration files, only the default printing font will accurately express the table results, as it works in conjunction with the table where certain single Hebrew characters have been converted into two Roman characters (the "begadkepat" letters with "h" added, shin (sh), and tzadi (tz)). However, the Roman "classical" table has only single character conversions from the Hebrew characters which are expressed through the R09TRAPC "classical" alphabet. Therefore for printing purposes only (not for screen where only one font is allowed), it is possible to design alphabets with the "classical alternative", "modern" and "simplified modern" two-character expressions (bh gh dh kh ph th sh tz ch) configured as single characters. This has been done with the R09TRABC, TRAMC & TRASC fonts. Therefore, when the Roman "classical" table (RTRA.TBL) is run, as well as the default printout with the R09TRAPC font, alternative printouts can be run in the "classical alternative", "modern" and "simplified modern" styles.

5(e) Description of Transliteration System: Text Preparation and Interchange Procedure and Table Rules

1. Text Preparation and Interchange Procedure

The main driver of this transliteration system is the MLS Configurable Text Interchange Utility (CTIU). This utility is designed to export MLS files to any other text file format; and to import any other text file format to MLS. The "other text file format" most commonly used is the standard IBM ASCII conventions described in the DOS 3.0 manual and used in this system. CTIU has two features which make it a practical transliteration device: 1) it will handle strings, i.e. it will convert any string of MTS codes into any string of ASCII codes; 2) it has an optional feature which "unbundles" the Hebrew vowels represented by overstrikes, treating them as separate characters. The following steps are necessary:

1. Prepare Hebrew source file by key entry to MLS.
   - use MLS mnemonic Hebrew keyboard map file.
   - avoid punctuation marks separated from words:
     e.g. word - word, ... word, word .....
   - distinguish between dagesh vav (Sh-v) and vowel vav (Reg-v).
   - insert stress marker when stress is not on the last syllable (note: this must be inserted from knowledge or in consultation with a native speaker).
   - insert metheg when required to signal long qametz before mobile sheva.
2. Convert MLS Hebrew file to Roman ASCII file, using CTIU

with: 1) the optional feature to treat overstrikes as separate characters (-v); and 2) tables designed to implement transliteration rules (accessed through the configuration file). The command format is:

CTIU (sourcefile.smp) (sourcefile.asc) (Heb config file.cnf) (-v)

3. Convert ASCII Roman file to MTS Roman file, again using tables designed to implement transliteration rules. The command format is:

CTIU sourcefile.asc sourcefile.mls Rom config file.cnf

4. Edit Roman output file (sourcefile.mls) to enter carriage return after CTIU language delimiter (/H/) and to check that it is parallel to the Hebrew input file (sourcefile.smp)

The pronunciation rules described in Section 5(c) are too complex to be implemented in their entirety in step 2. They are therefore implemented in two stages in steps 2 and 3. The table used in step 2 is called the Hebrew table and those used in step 3 are called the Roman tables. The rules applied by the two sets of tables are described below. It should be noted that even if all rules could be applied in one pass in step 2, step 3 is still necessary in order to create an MLS readable file. In such circumstances the table would be very simple, being confined to single character conversions from the one code to the other without the complexity of strings.

There is one Hebrew table and four Roman tables. One set of special pronunciation rules (described below) apply to all four Roman tables. The tables differ in their conversion of individual Hebrew consonants and vowels. Each of the four Roman alphabets, classical, classical alternative, modern and simplified modern (described in Section 5(d)), has its corresponding table of the same

name which converts Hebrew characters to Roman according to the particular scheme of each alphabet. The differences between the tables are therefore inaccordance with the differences between the alphabets: considerable between classical and simplified modern; small but significant between classical alternative and modern.

2. Text Interchange Table Rules

Hebrew Table

1. Special terminology - conversion of abbreviations, e.g. יי = adonai; and irregularly pronounced words, e.g. שִׁשְׁתַּיִם = shtayim

2. Short qametz before maqqef(־) preceded by a letter with sheva. The sheva is quiescent.

3. Short qametz before metheg when followed by letter with furtive qametz.

4. Short qametz before maqqef(־) preceded by a vowel-less letter other than aleph, he or 'ayin.

5. Short qametz before letter with furtive qametz.

6. Mobile sheva under 1st letter of word. When the sheva is first of two together, the second is quiescent (see Rom table rule 6)

Roman Tables

1. Dagesh lene following quiescent sheva

2. Dagesh lene at the beginning of a word.

3. Short qametz before dagesh forte.

4. Short "a" (pathaḥ) under a guttural at the end of a word pronounced before, not after, the guttural.

5. Mobile sheva under first of two similar letters even if otherwise it would be quiescent, e.g. following a short vowel.

6. Quiescent sheva when second of two together at beginning of a word. This rule complements Hebrew rule 6 which makes sheva mobile under the first letter of a word.

7. Quiescent sheva when the first of two in the middle or at the end of a word. If a dagesh letter follows the quiescent sheva, the dagesh is dagesh lene (see Roman table rule 1).

8. Quiescent sheva after a short vowel. This rule covers short vowels a e i o & u. Qametz before sheva is read as short 'o' unless accompanied by metheg.

9. Quiescent sheva after accent.

10. Redundant yod (y) before i.

11. Consonantal yod after a long a or e when the yod is followed by a vowel. This rule takes precedence over rules 13 - 15 below which provide for redundant yod after these vowels.

12. Consonantal yod after qametz at end of word.

13. Redundant yod after short i e and long a e with metheg.

14. Redundant yod after accented short i e and accented long a e.

15. Redundant yod after short i e and long a e.

7. Quiescent sheva at the end of a word.

16. Redundant hē (h) when the last letter without a vowel at end of word.

In addition to implementing these pronunciation rules, the tables convert individual Hebrew letters as required by the particular Roman alphabets (see Section (5d)). The Roman tables also convert to a null character the ASCII code (233), selected to represent quiescent sheva, to cover cases of quiescent sheva left over after application of the specific Roman table rules.

Note that Hebrew rule 16, although frequently applied in transliteration systems, is not applied in this system, following Lambdin (see also Section 5(c)5).

6(f). Computer Program Description

The computer program BITEX is written in Microsoft QuickBASIC 4.0. It has two modules: a front end, BITEXSCR, which allows the user to name the input and output files to be used and to set variables for a cloze procedure; and the main program, BITEX, which has three main functions:

1. Merging Hebrew and Roman texts into one bi-textual file
2. Refining transliterated Roman text
3. Applying a cloze procedure for a reading tutorial

These two modules and their functions will be discussed in context, following the general logic and flow illustrated in the block diagram attached as Appendix 6. The numbers of the blocks described below, starting with "Block 0. Front End", correspond to the numbers of the boxes in the block diagram. The section ends with a listing of error messages. The detailed discussion of the individual blocks explains the logic of the program and should be read in conjunction with the program.

Four factors create special programming problems:

1) The right to left orientation of Hebrew text compared with the right to left orientation of Roman. Both texts are oriented left to right in computer file form. The difference in orientation appears when printing to screen or hard copy. It is brought about by information included in the MLS file structure. The orientation is critical in the BITEX text file. The first word in a line of the Roman text file must be put last in the Roman line of the BITEX file in order to appear under the first Hebrew word in the Hebrew line of the BITEX file. The Roman words must therefore be moved into reverse order but still reading left to right.

2) The MLS file structure. Each character is represented by either two or four bytes. The first byte carries the standard ASCII code. The second byte, together with other information, carries a vowel marker "0" or "1" in the high order position and an alphabet code in the four low order positions. If set to "1", the vowel marker indicates that up to three overstrike vowel and/or accent marks follow in the next two bytes. Thus an MLS character will be two bytes long without overstrikes but four bytes long with overstrikes. This structure is significant in governing several features of the program:

(a) Counting bytes and characters for spacing the two texts relative to each other in the BITEX file: a Hebrew character may be two or four bytes long, a Roman character will normally be two bytes long.

(b) The need to use binary files as distinct from sequential or random-access files.

(c) The need to avoid misreading the second byte: because the alphabet number in the second byte is significant, two bytes at a time must be read in order to positively identify characters, including such key characters as space and carriage return. Byte 2 must not be read as a character although the combination of flags and alphabet code will give the equivalent of a valid ASCII code.

3) Defining a word: The simple definition of a word for purposes of the program is: "any sequence of characters ending in a space or a carriage return". A word will take with it any punctuation marks at its beginning or end (including parentheses) plus one space or carriage return at its end. A word may, of course, be one character (plus any punctuation mark plus space or carriage

return). For purposes of the cloze procedure, it is necessary to count words. A word count is taken after reading the first space at the end of the word. Difficulty arises in avoiding illegal word counts when there are leading spaces at the beginning of a line, more than one space between words, leading spaces before a carriage return or one or more carriage returns following a carriage return (i.e. blank lines of text). Any of these conditions may arise through normal paragraph indentation and line spacing between paragraphs, or careless preparation of the source Hebrew text, or the demands of the text itself: e.g. different spatial arrangements for songs or poetry.

There are some possible punctuation or graphic devices that may give rise to an illegal word count: for example, the punctuation dash (word - word); or a series of dots or other mark following a word and separated from it by a space (word .....); or a series of dots or other mark starting a line and separated from the first word by a space (..... word). These contingencies may be avoided by excluding them from the Hebrew text. If they should be included, they are likely to be so rare as not to cause a significant problem. They will be treated as words in the cloze procedure and so subject to blanking out. This does not prevent the cloze procedure effectively operating. They will also cause an inflation in the word count (probably insignificant) should this be used for some statistical purpose. If experience indicates that the problem is more significant than anticipated, then these contingencies can be eliminated by additional programming.

4) Spacing text in the BITEX file: because most of the Hebrew vowels are overstrikes, one Hebrew consonant and overstrike vowel

will appear as one character on screen and hard copy but will need two Roman characters to transliterate it. In addition, some Hebrew consonants are transliterated by two Roman consonants, as discussed in Section 5(d), Description of Transliteration System. Roman text, therefore, normally has a significantly larger number of characters than the corresponding Hebrew text. This means that the Hebrew words must be spaced out so that they appear directly above their corresponding Roman words.

Liberal print instructions have been included in the program in order to print out readings of ASCII codes and contents of variables for debugging purposes when running small test files. These instructions have been neutralised by the expedient of converting them to comments through a single quote mark inserted at the beginning of the affected lines. They can therefore be easily reactivated for test purposes on future occasions if required.

The following abbreviations are used in the discussion which follows:

hf - Hebrew File

rf - Roman File

bf - BITEX File

ha - Hebrew Array

ra - Roman Array

ba - BITEX Array

hwb - Hebrew Word Buffer

rwb - Roman Word Buffer

Block 0. Front End

This front end is a separate module, BITEXSCR, which is chained to the main program, BITEX. Its purpose is to enable the user to open two input files, a Hebrew and Roman text file, and a BITEX output file and to assign values to variables used in the cloze procedure. An Include File, BTXCOM.BI, includes all variables common to both BITEXSCR and BITEX. The input and output files are opened as binary files by the user as follows, instructed by a screen display and prompted by INPUT commands:

1) Hebrew input file: (drive: filename.heb)

2) Roman input file: (drive: filename.rom)

3) BITEX output file: (drive: filename.btx)

In addition, "LPT1" is opened for printer output.

A second screen display instructs the user, prompted by INPUT commands, in setting values to the following cloze variables:

cloze is the variable set to blank out Roman words in the BITEX file. It can be given any value. Seven is a characteristic value: every seventh word is deleted. For this to happen, it is necessary to keep track of crwc, the cumulative Roman word counter. Blanking out can start with any word from 1 to 7 according to choice of a cumcloze variable, which is described next.

cumcloze1, 2, 3, 4, 5, 6, 7 are alternative cumcloze choices which blank out every seventh word (or whatever value is selected for cloze) starting with the particular cumcloze digit. Thus, cumcloze1 = 1 will blank out words 1, 7, 15... Cumcloze4 = 4 will blank out words 4, 11, 18... If both cumcloze1 = 1 and cumcloze4 = 4, then words 1, 4, 7, 11, 15, 18... will be blanked out. If every cumcloze1 - 7 = 1, then all Roman words will be blanked out. This

feature is required for using the system as a tutorial for reading the non-Roman script.

clozev, if set to 1, will blank out the Hebrew vowel pointing. This is appropriate as a final stage in a reading tutorial in modern Hebrew as vowel pointing is not used in modern written Hebrew.

Block 1. Initialise variables and move header information to BITEX file (bf)

This block (a) declares and initialises variables, (b) moves header information and print format instructions from the MLS Hebrew file to the MLS BITEX file and (c), skips header and print format data and the CTIU language delimiter in the MLS Roman file, all to make ready for reading character data.

(a) Variables

Variables include arrays and single character variables. For convenience, variables opened in later blocks are listed and described here in addition to those (the majority) which are opened in this block. The arrays described below are initialised by the subroutine nullarray: which sets each cell to the null ASCII code instead of the Basic default of 1. This is done because the default 1 can be misread as an MLS byte two with the Hebrew alphabet code (also "1") when all other bits (flags) are set to 0.

ha = Hebrew Array, string, 200 bytes. This takes one line of text, ending with a carriage return, from hf, the Hebrew input file. The length is governed by the MLS file structure which can take up to four bytes per character. The Hebrew text file should be prepared with no more than 40 t0 45 Hebrew characters per line. If a line is blank, then its length will be one character, represented

by the carriage return.

ra = Roman Array, string, 200 bytes. This takes one line of text, ending with a carriage return, from rf, the Roman input file. Forty Hebrew characters can generate as many as sixty Roman characters in transliteration. The Roman characters are normally two bytes long.

hwb = Hebrew Word Buffer, string, 80 bytes. This takes one Hebrew word, ending with and including one space or carriage return, from ha, the Hebrew array. The purpose of hwb is to delineate one word, after count of word, bytes and characters, preparatory to moving it to ba(1), the first dimension of the BITEX two-dimensional array.

rwb = Roman Word Buffer, string, 80 bytes. This takes one word, ending with and including one space or carriage return, from ra, the Roman array. The purpose of rwb is to delineate one word, after count of word, bytes and characters, preparatory to moving it to ba(2), the second dimension of the BITEX two dimensional-array. While the Roman word is in rwb, further transliteration rules can be applied, if required.

ba = BITEX two-dimensional array, string, 200 bytes. This takes one line of text up to and including carriage return from hwb into ba1 and from rwb into ba(2). A spacing algorithm is applied while moving each Hebrew and Roman word into ba to ensure that they line up relative to each other. The purpose of ba is to assemble the Hebrew and Roman text, one line at a time, preparatory to writing it one line at a time to bf, the BITEX file.

b1, b2 = binary file read/write variables, string, 1 byte, for hf, Hebrew input file.

b3, b4 = binary file read/write variables, string, 1 byte, for rf, Roman input file.

hwc, rwc = Hebrew and Roman word counters. A word includes one space after the last character and is counted when that space is read.

chwc, crwc = cumulative Hebrew and Roman word counters. These counters are required for the cloze procedure.

hbc, rbc = Hebrew and Roman byte counter. These are needed for determining the length of words to move into the BITEX array from the Hebrew and Roman word buffers.

hcc, rcc = Hebrew and Roman character counters. These are needed for measuring the difference in screen or print length between a Hebrew word and its Roman counterpart for spacing purposes in the final BITEX screen or print output. One Hebrew character with a vowel overstrike (four bytes) will be represented by two Roman characters (two bytes each). In addition, transliteration sometimes results in two Roman characters to one Hebrew, e.g. שׁ = sh, ם = mm.

hap1, hap2, rap1, rap2 = Hebrew and Roman array pointers to keep track of the position reached in arrays until the end of line (carriage return) as characters are progressively moved to hwb and rwb, two bytes at a time.

hp1, hp2, rp1, rp2 = Hebrew and Roman BITEX array pointers to keep track of the position reached in the BITEX array as characters are progressively moved from hwb and rwb, two bytes at a time, until the end of line is reached (carriage return in ha and ra).

hcr, rcr = Hebrew and Roman carriage return markers to indicate whether or not a carriage return has been read.

heof, reof = Hebrew and Roman end-of-file markers to indicate whether or not the end of input files has been reached.

hwordfin, rwordfin = Hebrew and Roman end-of-word markers when moving words from ha, ra to hwb, rwb. When hwordfin, rwordfin = 1, the end of word space has been read which triggers a word count; when = 2, a second space has been read, which must not trigger a word count.

hspace, rspace keep track of the number of spaces read after a word or at the beginning of a line.

rread. When rread = 1, the first two bytes of the Roman input file have been read in block 1 (move header). This calls for exception action in block 2 (move hf to ha) to process these bytes before reading the next two bytes.

hpass, rpass = Hebrew and Roman passes (iterations). When hpass, rpass = 1 (set at the beginning of each new line cycle), the first character of the line has been read. If, in blocks 4 or 6 (move one word from ha, ra to hwb, rwb), rpass, hpass is set to 2, the first character of the line is a space or carriage return. This triggers appropriate action in blocks 4 & 6 (ha to hwb and ra to rwb) and blocks 7 & 9 (hwb to ba(1) and rwb to ba(2)).

hskipflag, rskipflag = Hebrew and Roman hpass, rpass flags. When hpass, rpass = 2 (first character of line is space or carriage return), appropriate action before moving the first character is taken in the hwb to ba(1) and rwb to ba(2) processes (blocks 7 & 9) and hskipflag, rskipflag are set to 1 to avoid duplicating this action when word 1 is encountered. If hpass, rpass have not been set to 2 then the first character of the line belongs to word 1 and appropriate action is taken before moving this character to ba.

twc = translit word counter, tracks words processed in block 6 (refine transliteration) to prevent processing non-words (e.g. leading spaces, blank line carriage returns).

(b) Header information and print format instructions

The first 400 bytes of an MLS file are reserved for header information required for running on MLS. This information - languages, link files etc. - is transferred from hf, the Hebrew input file to bf, the BITEX output file. In addition, the print format instructions on hf are moved to bf. Print format instructions must start with ";" (ASCII 59) and end with a carriage return (MLS carriage return = ASCII 128).

(c) Skip header, print format and CTIU delimiter in Roman file (rf)

The Roman input file rf has been prepared by the Configurable Text Interchange Utility (CTIU) belonging to the Multi-Lingual Scholar (MLS) package. Therefore header and print format data will be identical with what has already been transferred to the BITEX from the Hebrew file and can therefore be skipped in the Roman file. In addition, CTIU has transferred to the Roman file the language delimiter used in the CTIU configuration file. This delimiter indicates the language from which text has been derived - in our case, Hebrew. MLS allows the user to set his own convention in defining this delimiter. We have chosen /H/ and further specified that /H/ must be followed by a carriage return to mark the end of the delimiter. This delimiter is not required in the BITEX output file and is therefore also skipped in the Roman

file.

(Note that if Roman text also appears in the Hebrew input file, e.g. a quotation or student instructions, then /R/ will appear in the CTIU-generated Roman output file at the beginning of the Roman text. This will be followed by /H/ when transliteration from Hebrew resumes. These additional delimiters will be used by the BITEX program to differentiate between Hebrew transliteration and English text. It will convert English text to spaces (i.e. blank it out) as there is no point in having English comments on the Hebrew line repeated on the Roman line.)

A subroutine, Readloop:, is employed to read the print format instructions and CTIU language delimiter in the Roman file. If a carriage return marking the end of the instructions or delimiter is not encountered after a given number of bytes, the program stops with the error message: "Start of text in file #2 not found in Readloop."

When the program leaves this block, the first two character bytes of the Roman file are residing in b3 and b4. Because of this, the "read = 1" variable is acted upon in the move rf to ra process (box 3) to move these bytes to the Roman array.

Block 2. Move one line from Hebrew file (hf) to Hebrew array (ha)

This block reads one line of text from hf, the Hebrew file, and moves it to the Hebrew array. The end of line is indicated by a carriage return. In the case of a blank line, i.e. a carriage return without text, the carriage return is read and moved as one line.

To ensure that a line is not moved without a carriage return, an error check is run. If the array is filled without a carriage return, the program stops with the error message "Hebrew line too

long". The line may be legitimate Hebrew text but its length is such that in the BITEX screen or hard copy printout the Roman line will wrap around, interfering with the line-up of Hebrew and Roman text. The convention must be observed in preparing the Hebrew text file to limit each Hebrew line to 40 - 45 characters and end each line with a hard carriage return. This will prevent Roman text wrap around.

An end-of-file check is made in this block which, if positive, bypasses processing, goes to the next block and picks up an end-of-file trail leading to the end of program.

Block 3. Move one line from Roman file (rf) to Roman array (ra)

This block reads one line of text from the Roman file and moves it to the Roman array. The same carriage return conventions apply as in the case of the Hebrew file (block 2. above). The same error check is made so that if the Roman array is filled without a carriage return, the program stops with the error message: "Roman line too long".

Reference has been made to "rread" in the list of variables in block 1 above. The first instruction in block 3 is to check if rread = 1, which indicates that the first character of the file is being processed. Because the first character has already been read as the end result of skipping the Roman file header etc. described in block 1 above, the FOR...NEXT loop is modified to start with n = 3 (bytes 3 and 4 will constitute the second character). At the same time rread is set to "0" so that when the next line is read and the initialising block is bypassed, the FOR...NEXT loop is restored to start with n = 1.

Block 4. Move one word from Roman array (ra) to Roman word

buffer (rwb)

This block moves one word from the Roman array to the Roman word buffer, prior to moving it to the BITEX array, counts the number of the word in the line in the Roman word counter (rwc), counts the cumulative number of the word in the cumulative Roman word counter (crwc), counts the number of bytes in the word in the Roman byte counter (rbc) and the number of characters in the word in the Roman character counter (rcc).

The end of word is marked by the space or carriage return which immediately follows the last character (or only character, in the case of a one-character word). A "word" includes a numeral and any series of characters or one character only which is not a space or carriage return. The programming in this block is complicated by the need to provide for special action required later before the first character of a line is moved into the BITEX array; and also to provide for leading spaces, more than one space between words, and a blank line (carriage return only) in which cases byte and character counts are required but not word counts.

The strategy is 1) clear all possible exception conditions out of the way before the regular processing of a word; 2) process the word; 3) before exiting the block, read the next character and check whether it is an exception condition or the first character of the next word; 3) exit the block after setting the array pointers to re-read the next character; 4) repeat the cycle starting with 1) above. This strategy is largely governed by the need to avoid illegal word counts.

Because of the complexity of this block, a flow chart is shown in Appendix 7 to assist in the following discussion of conditions

checked, reasons for checking, and action taken.

(a) Roman carriage return marker (rcr) check. If rcr = 1 at this point, the beginning of a new word cycle, then it means, because the Roman line is being processed before the Hebrew line, that the program has gone through the Hebrew array to Hebrew word buffer process (block 6) without encountering a Hebrew carriage return and has returned on a new word cycle. As the Roman line invariably is longer than the Hebrew line, an error condition is indicated and the program stops with the error message: "Roman line ended before Hebrew".

(b) End-of-file check. EOF indicates that the last character of the Roman file is in process. The last Roman word will be the first word in the last Roman line entered in the BITEX(2) array. Therefore a space must be inserted in the Roman word buffer at this point to provide separation of this word from the one which follows it in the BITEX(2) array. The Roman end-of-file flag (reof) is set to "1" to activate later EOF instructions in the EOF trail. If rspace, the Roman space variable, is greater than 0, then the preceding word has already been counted. Therefore the space now in the Roman array pointers, rap1 and rap2, must be processed to rwb (rwbproc:) but the word count procedure (rwctr:) bypassed. If rspace is 0, then the word just read and processed has not been counted and the program must branch to rwctr: before proceeding to rwbproc:

(c) First pass check. The rpass variable is set to 1 at the beginning of the program and before returning on a new line cycle. Therefore if rpass = 1, the first character of a new line is being processed. If this character is a space or carriage return rather

than a regular character, then rpass is set to 2. In the routine space check or carriage return check (see (g), (h) below), if rpass = 2, then the space or space substituted for a carriage return will be processed (rwbproc:) but word count (rwctr:) will be bypassed. (Rpass = 2 also triggers action appropriate to the first character of a line in block 6, rwb to ba(2)). After the rpass check, rpass is neutralized to 0 so that it will be inoperative in subsequent passes.

(d) Current character second or more space after a word. If the Roman space counter, rspace, is greater than 0 and the present character is a space then, if a word has just been read, as indicated by rwordfin = 1, that word must be moved from the Roman word buffer to the BITEX(2) array before dealing with the current character. Accordingly, rspace is set back to 0, rwordfin is set to 2 and the program exits from this block to rwbend:. It goes on to read the corresponding Hebrew word and process both Roman and Hebrew words to the BITEX array. The Roman array pointers, rap1, rap2, are set back by one byte so that they will reread the present character as a first space in the next word cycle. At the check for first space (see (g) below), rwordfin = 2 will identify the character as the first of an unknown number of spaces following the regular space after a word. Whether there is one extra space after the word or several, the one or several will be read and processed, but word counting avoided, until the first character of a new word or a carriage return is encountered. Then the one or several spaces will be processed to the BITEX array before moving to the next word cycle.

(e) Current character carriage return after space at end of word. Normally, a carriage return comes immediately at the end of

a word. If rspace is greater than 0, a word count has already been taken. Therefore a second word count must be avoided. Also, as the last word of the line plus any following spaces will be moved to the beginning of the BITEX(2) array line, the carriage return must be substituted by a space. This is done, the Roman carriage return marker, rcr, is set to 1 and rspace and rwordfin are returned to 0. The program branches to rwbproc:, bypassing rwctr:, the word count routine. Provided it encounters a Hebrew carriage return in the ha to hwb block it will then exit the line cycle. If it does not encounter a Hebrew carriage return then it will return on the next word cycle and stop with an error message upon finding that rcr = 1 (see (a) above).

(f) Current character first character of a new word after second or more space following preceding word. If none of the above conditions exist and rspace is greater than 0, then the current character must be the first of a new word while the preceding word has already been counted. Therefore the program exits the block to rwbend: to move the contents of the Roman word buffer to the BITEX(2) array before dealing with the current character. Accordingly, rspace and rwordfin are set to 0 and the Roman array pointers, rap1 and rap2, are set back one byte so that they will re-read the current character as the first of the next word cycle.

This concludes the series of checks for when rspace is greater than 0. The remaining possibilities are a first space or carriage return immediately after a word, or the first character of a new word.

(g) Current character first space. There are three possibilities:

1) As discussed in (c) above, if rpass = 2, then this is not only a first space but also the first character of the line. Therefore it is processed (rwbproc:) but a wordcount bypassed. Subsequent spaces immediately following will be processed as discussed in (d) above until the first character of a new word or a carriage return is read.

2) As discussed in (d) above, if wordfin = 2, then it was set when the second space after a word was read and that space is now being read as the first space in a new cycle. Accordingly rwordfin is returned to 0, the space is processed (rwbproc:) and the word cycle continues. If more spaces are read they will be processed ((d) above) until the first character of a new word or a carriage return is read.

3) If neither of the above two conditions apply, then this is the first space after a word. Therefore the program branches to rwctr:, the word count routine, before processing at rwbproc:.

(h) Current character carriage return. This will be either a first read of a carriage return immediately after the end of a word, or a re-read of the carriage return first checked at (c) or (e) above. It is the signal for exiting the line cycle after processing. Therefore the Roman carriage return flag, rcr, is set to 1 (if coming from (e) above, it is already 1 and that setting is simply repeated here), rwordfin is set to 0 ready for a new line cycle, and a space is substituted for the carriage return in the Roman array pointers rap1 and rap2 (again, if coming from (e) above, where the reason for this space is explained, this is simply a repeat). There are now two possibilities:

1) If rpass = 2, then this carriage return is the first and only

character in the line. Therefore the program branches to rwbproc:, bypassing rwctr:, the word count routine. The space inserted instead of carriage return prevents an unwanted Roman carriage return being processed. A Hebrew carriage return is inserted in the BITEX(2) array for reasons explained in block 7(b) below.

2) If rpass is not 2, then this carriage return must be a regular carriage return immediately after a word. (If it is a carriage return after one or more spaces, then it has been picked up in (e) above.) Therefore a word count is required. The program branches to the word count routine, rwctr:, and from there to rwbproc: for processing and, provided a Hebrew carriage return is read in the ha to hwb block, exit from the line cycle.

(i) Regular processing. If the program has reached this point without a branch, then the current character belongs to a word. Therefore the program branches to rwbproc: for regular processing, avoiding the word count routine, rwctr:. Regular processing simply moves the character - whether space or regular character of a word - to the Roman word buffer, rwb, from the Roman array, ra, and updates byte and character counts (rbc, rcc).

After processing, both the Roman carriage return (rcr) and end-of-file (reof) flags are checked. If either is set to 1, then the program goes to rwbend: and, provided the same condition is encountered in the ha to hwb block, exits the line cycle.

(j) Check for vowel flag. The MLS file code structure has been discussed under point 2) at the beginning of this Section (5(f)). If the second byte of the current character has the overstrike vowel flag (in the high order position) set to 1, then special action is required. The second byte also contains other flags and an alphabet

code. In order to check the high order position only, the logical operator AND is used. The Basic AND combines two values, bit by bit, and produces a one only when both bits are one. Therefore if there is a one bit in the high order position of a byte and it is combined by AND with 128 (= binary 10000000), then the result will equal 128 as a one is left only in the high order position in both bytes.

If the current second byte is so identified as having the overstrike vowel flag set to 1, then the next two bytes contain up to three overstrike "vowels" (the term "vowel" includes any overstrike) which must be processed and counted for bytes but not counted for characters. The separate character count is used for calculating spacing requirements when lining up the Hebrew with the Roman word in the BITEX array. Therefore at this point an exception routine implements these requirements before the program returns to the regular word cycle.

(This vowel flag check is crucial in the next block (moving each word from Hebrew array (ha) to Hebrew word buffer (hwb)). In this block (ra to rwb) it will be unused for the time being. The Roman text is originally produced by CTIU. In pass 1 (MLS Hebrew to Roman ASCII) CTIU has the ability to unbundle the Hebrew overstrikes and produce regular ASCII codes for them as specified by the tables. It does not, however, have the ability in pass 2 (Roman ASCII to Roman MLS) to define any of those characters as overstrikes. It would, in fact, be an advantage to be able to specify certain characters as overstrikes. An example is the primary stress marker which is an overstrike in the Hebrew text but has to be a separate character in the Roman. A Roman character overstrike

could be accomplished only by programming back the vowel flag into Roman byte 2 where required. The present transliteration system works without this but it will be considered as a future enhancement.)

(k) Word count routine (rwctr:). This routine is implemented only upon encountering the first space, carriage return or EOF immediately after a word. Special action is required for the first Roman word in the line (rwc = 1). This word becomes the last word of the Roman line in the BITEX(2) array where a special Hebrew carriage return and hard space is inserted (for reasons explained in block 7(b) below). The Hebrew carriage return in the Roman line matches the Hebrew carriage return in the Hebrew line. The Hebrew hard space, however, is extra. When the program is in the word count routine, a space resides in the Roman array pointers (rap1, rap2). Therefore for word 1 the program branches out of the Roman word cycle to rwbend: without processing the space. This compensates for the Hebrew hard space specially inserted in the Roman line in the BITEX(2) array. Before the branch, however, the Roman character count (rcc) is advanced by one to take account of the Hebrew hard space in calculating the length of the word for spacing purposes.

(l) rwbend:. This address simply marks the end of the Roman word cycle to which the program has branched or to which it arrives after the completion of the FOR...NEXT loop. It may reach the end of the loop without reaching the end of a word or a non-word series of characters, although this is unlikely. For example, a text may have a dotted line drawn across it. This would not be construed as a word and a word count would not be taken as

there is not a space to trigger the count. However, the Roman word buffer may be filled. Therefore an error message is not produced if the Roman word cycle is terminated by the FOR...NEXT loop as this may happen legally on occasions. The program continues to the next block.

Block 5. Move one word from Hebrew array (ha) to Hebrew word buffer (hwb)

This block closely parallels block 4, Move one word from Roman aray (ra) to Roman word buffer (rwb). Therefore discussion will be restricted to only what varies compared with block 4.

(a) The Hebrew carriage return is processed in the normal way. It is not replaced with a space as in the case of the Roman carriage return. There is no need, therefore, to set a Hebrew carriage return flag at this point. The Hebrew carriage return remains in the Hebrew array pointers, hap1, hap2, and can be identified after processing.

(b) The Hebrew word count routine (hwctr:) is not complicated by the need to give special treatment to word 1 in the line. The Hebrew retains its natural right-to-left orientation in the BITEX(1) array with a regular Hebrew carriage return at the end of the line. The Roman text, in contrast, is put into reverse word order with the last word of the Roman array becoming the first word in the BITEX(2) array. The Roman carriage return must therefore be replaced with a space and the Roman line in the BITEX(2) array given a right-to-left orientation by enclosing it between two Hebrew hard spaces and giving it a Hebrew carriage return.

(c) The check for the vowel flag in byte 2 is of crucial importance in this block as almost every Hebrew word has overstrikes, which cause a greater number of characters in the transliterated Roman word than in the Hebrew word. Therefore the two vowel bytes following byte two are given a byte count but excluded from a character count.

(d) The vowel flag in byte 2 is checked also for another reason in this block. If clozev is set to 1 at the beginning of the program, the intent is signified of removing all Hebrew overstrike vowels from the Hebrew text as the last step in the cloze procedure. Therefore, if clozev = 1 and the vowel flag check is positive, then ASCII code 1 is moved into Byte 2. This eliminates all flags and represents the Hebrew alphabet code only. The Hebrew array pointers are advanced by two bytes to skip reading the vowel bytes 3 and 4. Therefore the hebrew overstrikes are removed from the text processed to the BITEX array. The program bypasses the vowel flag check for when clozev = 0 and returns to hwbloop: to read bytes 1 and 2 of a new character. If clozev = 0 and the vowel flag check is positive, then the processing described in (c) above is carried out.

It should be noted that the action taken when clozv = 1 removes also underlining and strikethroughs. These are infrequent in the Hebrew text. If, however, it is desired to accomodate them, then an additional series of logical AND operations must be devised for the underline flag and/or the strikethrough flag and an appropriate ASCII code moved into byte two that retains either or both of these flags and excludes the vowel flag.

(e) At hwbend: there is a critical error check. At this point a

Roman word and its corresponding Hebrew word should have been processed. The Hebrew word count (hwc) is compared with the Roman word count (rwc). If they are not the same, the program stops with the error message: "Difference between number Hebrew and Roman words processed: check text." This prevents the production of a BITEX file where the Roman words are out of synchronization with the Hebrew words. The remedy is to go back to the Hebrew and Roman source text files and find out why, the Roman text having been produced from the Hebrew text by CTIU, the two texts are not parallel. The reason may be print formatting commands inserted into either file after its production, or failure to put a carriage return after the CTIU language delimiter in the Roman file, or editing changes to either file after its production.

Block 6. Refine transliteration

The transliteration is done by MLS-CTIU in conjunction with the Hebrew and Roman tables which implement the pronunciation rules that have been discussed in Section 5(c). One part of a rule has deliberately been excluded from the tables because of space requirements. This is the case of a short qametz (o) in the final syllable of a word when this is unaccented. The tables are already close to capacity (600 entry pairs in each table). Because of the number of contingencies that define the end of a word - space, carriage return, hyphen, any punctuation mark - it takes 198 entry pairs to implement this very simple rule through the tables. Therefore it is implemented by programming in this block.

Because the regular Hebrew stress is on the last syllable, there will be relatively few cases of a qametz co-inciding with an

unaccented last syllable. The convention used in preparing the Hebrew text is to show the stress marker only when this is not on the last syllable. Therefore, all that is necessary here is to: 1) check the Roman word for a stress marker; and 2) if found, to check the last syllable for a qametz. If a qametz is found, then the default long "a" is changed to the required short "o".

Because the space after the first word in the Roman word counter (rwc) bypassed the byte count routine (see Block 4), it is necessary here to add the byte count that was missed so that word one will be consistent with all other words when the ensuing byte-controlled tests are applied.

System tests have not revealed any other pronunciation rule or feature not picked up by the tables. Hebrew is a very regular language. Therefore there is good reason to believe that nothing significant has been omitted from the tables. If, however, future experience indicates particular exceptions to the rules which cannot be conveniently picked up by the tables, then they can be dealt with by programming in this block.

Every language has its own collection of odd usages which do not conform to usual rules. They sometimes represent archaic remnants left over from a previous historical period of the language and sometimes simply human arbitrariness. Whatever the reason, they exist. Hebrew is no exception although it probably has fewer than most languages. These usages are easily accommodated by the tables as they represent usually only one entry: in effect, convert "abc" to "xyz". The important thing is to get these entries at the top of the tables so that they are taken first before applying other rules. As CTIU sorts by number of terms in an

entry, and as four is the maximum number of terms in the Hebrew table, then an exception usage requiring less than four terms may be a problem as then it will not be taken first. The program remains as a final point where such problems can be addressed.

A possible future alternative to the table approach to transliteration is direct programming. In this case, only a Hebrew text would be read. Each Hebrew word would be read into the Hebrew word buffer and then examined for application of all the pronunciation rules that have already been discussed. The transliteration would be moved into the Roman word buffer. Various checks would be applied. Then the Roman word would be written to a Roman output file and also to the BITEX array for reading, with the Hebrew, into the BITEX output file. This approach offers the advantage of a one-step approach to transliteration - one program pass rather than two CTIU table passes plus a program pass - which may also yield the advantage of reduced processing time.

Block 7. Move one word from Roman word buffer (rwb) to BITEX(2) array (ba(2)) and align

The purpose of this block is to move the word sitting in the Roman word buffer to the BITEX(2) array. In block 9 the Hebrew word will be moved to the BITEX(1) array from the Hebrew word buffer. After spacing adjustments and the cloze procedure, if required, are applied, the contents of BITEX(1) and (2) will be written to the BITEX output file. The task in the BITEX array, therefore, is to move each line of the two texts into the exact order required in the BITEX file: the exact text order (Hebrew

first), the exact word sequence (Roman words in reverse order), and the exact alignment (Roman words immediately beneath their corresponding Hebrew words). This requires the following steps for the Roman text:

(a) First character-of-line check. Two checks are required: 1) if rpass = 2, then the character is either a space or carriage return and is the first character of a new line; 2) if the Roman word counter (rwc) = 1, then the character is the first character of the word and the first character of the new line. The rpass = 2 check is applied first. If true, then the program branches to the special routine (ma2spec:) applied at the beginning of a new line and sets rskipflag to 1 to avoid a duplication of the routine if a word follows in the same line (as it will if the first one or more characters of the line are leading spaces). Then the rwc = 1 test is applied. If true, the branch to ma2spec: is followed only if rskipflag has not been set to 1 (i.e. rskipflag = 0).

(b) Special routine at beginning of new line (ba2spec:). The Roman text will be moved word by word from the Roman word buffer and placed in the BITEX(2) array in reverse order so that Roman word 1 will come directly under Hebrew word 1 when the BITEX file is printed out. For this to happen, The Bitex file must have a consistent right-to-left orientation, following the Hebrew. This is accomplished by enclosing the Roman line between two Hebrew hard spaces and substituting a Hebrew for a Roman carriage return. Because the Roman text is being moved in backwards, the last position in the line will be filled first. Before this happens the closing Hebrew hard space and Hebrew carriage return must be moved into the last positions of the BITEX(2) array. This

constitutes the special routine at the beginning of a new line (ma2spec:). The BITEX(2) array pointers, rp1 and rp2, are then set to move the first character of the Roman line (normally, the last letter of the first word) immediately before the Hebrew hard space. Note that in block 4, word one is moved from the Roman array to the Roman word buffer without the usual space at the end of the word because of the Hebrew hard space that ends up immediately after it in the BITEX(2) array through this special routine.

(c) Line-up of Roman word (rlineup:). This routine is the exception rather than the rule as normally, the Roman word being longer than the Hebrew, it is the Hebrew word that is aligned. However, if the number of characters in the Hebrew character counter (hcc) is greater than those in the Roman character counter (rcc), the requisite number of spaces to equal the difference are move into the BITEX(2) array before the Roman word is moved. This will ensure alignment of the two words, the Roman beneath the Hebrew, in printouts from the BITEX file.

(d) Regular processing (regloop:). Here, "regloop:" is in contrast to "clozeloop". Clozeloop: is discussed in the next block. Regular processing is accomplished in a FOR...NEXT loop where n = the number of bytes in the Roman byte counter (rbc), which is the number of bytes for the contents of the Roman word buffer. The Roman characters are moved from the last backwards, filling in the BITEX(2) array from the end of the line backwards so that the Roman words retain their left-to-right orientation for reading purposes but follow a right-to-left word order.

Block 8. Cloze procedure

The cloze procedure is inoperative when cloze = 0 and cumcloze1 to 7 = -1. The cloze procedure operates if cloze is given a value greater than 0 and one or more of the cumcloze variables have been given values greater than minus one. Let us assume that cloze is given a value of seven. At the beginning of the program cumcloze1 to 7 are deliberately arranged in the order 1 4 6 2 5 7 3 so that the Roman text is progressively blanked out to leave an even spacing of the words remaining. First, every seventh word starting with word 1 is blanked out; then every seventh word starting with words 1 and 4 and so on until only every seventh word starting with word 3 is left. Lastly, this series of words also is blanked out, leaving only the Hebrew text. If cumcloze is given a value of five, then only cumcloze variables 1 to 5 are defined, using the pattern 1 4 2 5 3. Cumcloze variables 6 and 7 are left with the default -1 which ensures that they do not operate.

If the cumulative Roman word counter (crwc) equals any of the cumcloze variables, then clozeloop: operates. The same routine as regloop: is performed except that 1) periods (ASCII 46) are moved to the BITEX(2) array instead of the regular characters of the word; and 2) two less bytes than at regloop: are moved so that the concluding space of the word remains and is not turned into a period. Note that any stress marks in the middle and punctuation marks at the end of the Roman word are converted to periods. Therefore the periods exactly equal the space the word occupies in the undeleted text. This helps the student visualize the missing word as he reads the Hebrew text until finally he is comfortable with the Hebrew text alone.

An adjustment is necessary before moving the periods in place of the regular characters of the word to be deleted. The cloze procedure loop is two bytes less than the regular procedure loop so that the space at the end of the word is not replaced by a period. Therefore that space must be moved into the BITEX(2) array as part of the cloze procedure. Again, word one is an exception because it does not carry a concluding space (a Hebrew hard space has already been moved into the position the regular space after word one would occupy). This means that word one needs a loop equal to all of its bytes, not two less. Therefore, for word one, a period instead of a space is moved into the BITEX(2) array to compensate for the two bytes less of the cloze loop.

The cloze procedure concludes with a new setting of the cumcloze that has just been acted upon. For example, if the cumulative Roman word counter (crwc) = 1 and cumcloze1 is activated, then word one will be blanked out and cumcloze1 will be reset to 8 (cumcloze + cloze). When word 8 is moved, it will automatically be processed by the cloze procedure. Finally, clozeflag is set to 1 to signal that the cloze procedure has just been applied and to thus enable bypassing of the regular routine which the cloze routine has just replaced.

The program moves to the next block from both the cloze routine, when applied, and from the regular routine.

Block 9. Move one word from Hebrew word buffer (hwb) to BITEX(1) array (ba(1)) and align

This block closely parallels block 7 which moves a word from the Roman word buffer to the BITEX(2) array and aligns.

Discussion will be restricted to where there are differences.

(a) First character-of-line check. There are no differences here except the substitution of hpass for rpass and hskipflag for rskipflag.

(b) Special routine at beginning of new line (balspec:). The Roman line in the BITEX(2) array is filled from the end backwards. The Hebrew line in the BITEX(1) array is filled from the beginning forwards. Therefore the special routine here deals with the first position of the BITEX(2) array whereas the Roman line special routine deals with the last positions of the BITEX(1) array. A Hebrew hardspace is moved into the first position of the BITEX(1) array (Hebrew line) to balance the Hebrew hardspace that will be moved to the beginning of the BITEX(2) array (Roman line) (see (e) below). The Roman line must have a Hebrew hardspace to enable MLS to give it right to left orientation. The Hebrew line could have a regular space. A hardspace is selected because it shows as a raised period on screen and matches the similarly visible hardspace at the beginning of the Roman line.

(c) Line-up of Hebrew word (hlineup:). This routine is the same as that described in point (c) of block 7 except that here it is the exception while there it is the rule. The Roman word will almost always be longer than the Hebrew because 1) each Roman vowel is a separate character while most Hebrew vowels are overstrikes and 2) several Hebrew consonants are transliterated by two Roman characters. The required number of spaces are inserted behind the Hebrew word so that the last character of the Hebrew word is directly above the first character of the Roman word. The

eye can then follow a circular movement, reading left-to-right along the Roman word and back right-to-left along the Hebrew word.

(d) Regular processing. Regular processing immediately follows as there is no "clozeloop" for the Hebrew text. Only Roman words are blanked out. The regular processing is straightforward as the Hebrew words are moved in regular sequence forwards, not backwards like the Roman.

(e) Hebrew carriage return and Hebrew end-of-file checks. The Hebrew carriage return flag was set at the end of block 9 at hwbend:. The Hebrew end-of-file marker was set at the beginning of block 6 at hwbloop:. If either of these flags equals one, then the program branches out of the word cycle to alignend:. A Hebrew hard space is moved into the first position of the Roman line in the BITEX(2) array to complete the enclosure of the Roman text between Hebrew hard spaces to give it right-to-left orientation. The program then moves on to block 10 which writes the BITEX(1) and (2) arrays to the BITEX output file. If neither the Hebrew carriage return nor the Hebrew end-of-file markers are set to one, then the program nulls both Roman and Hebrew word buffers and returns to rwb: in block 4 to start a new word cycle.

Block 10. Move BITEX array (ba) to BITEX output file (bf)

(a) Move BITEX(1) array. The Hebrew line in the BITEX(1) array is moved to the BITEX file. The end-of-line is marked by a carriage return which sends the program to the next step, moving the BITEX(2) array.

(b) Move BITEX(2) array (writeba2:). Before moving the Roman

line in the BITEX(2) array to the BITEX file, a check is made to ensure that the first character of the line is a Hebrew hardspace as this is critical for alignment of the Roman with the Hebrew line. If the Hebrew hardspace is found, then the Hebrew hardspace flag (hhsflag) is set to one to prevent the same check being made before moving the subsequent characters in the line. The first character of the Roman line in the BITEX(2) array (Hebrew hardspace) is not in the first position in the array because the Roman line is moved in word by word backwards and the array is larger by design than the Roman line. Therefore there will be an indeterminate number of null cells to check before the Hebrew hard space is encountered. If the Hebrew hardspace is not found, then the program stops with the error message: "Hebrew hardspace not found as first character of Roman line in ba(2)".

(b) Regular processing (ba2proc:). The Roman line is moved to the BITEX file. The end-of-line is marked by a Hebrew carriage return which sends the program to the next step (bawrend:).

(c) End of writing BITEX array to BITEX file (bawrend:). The program makes an end-of-file check of both Hebrew and Roman files. If both are in an end-of-file condition, then the program ends. If only one of the files is in an end-of-file condition, then the program stops with an error message: "EOF(1) and EOF(2) not coincident". Both files should end at the same time. If they do not, then something has been added to either one or the other and this should be checked. If the end-of-file condition has not been reached, then all pointers, counters and flags are reset, all arrays and buffers are nulled and the program returns to readhf: at block 2 to repeat the line cycle.

(d) Repeat run or END (closing:). A screen display tells the user that the program run is finished and lists the input and output files used. An INPUT prompt gives the user the choice of either ending the program or repeating another run with the same or different input files, a new output file and the same or different cloze settings.

Summary of Error Messages

1. Block 1, if a carriage return is not read after print format instructions and/or CTIU language delimiter in Roman input file:

"Start of text in file #2 not found in readloop:"

2. Block 1, if CTIU language delimiter not read in the Roman file:

"Language delimiter not found in file #2"

3. Block 2, if Hebrew array is filled without reading a carriage return in the Hebrew input file:

"Hebrew line too long"

4. Block 3, if Roman array is filled without reading a carriage return in the Roman input file:

"Roman line too long"

5. Block 4, if a Roman carriage return has been flagged but not a corresponding Hebrew carriage return so that a new line cycle is started:

"Roman line ended before Hebrew: rcr, hcr = x, x"

6. Block 6, if Hebrew word # is not equal to Roman word #:

"Difference between # of Hebrew and Roman words processed: check text"

7. Block 10, if upon moving the BITEX array to the BITEX output file, a Hebrew hard space is not found as the first character of the Roman line:

"Hebrew hardspace not found as first character in ba2"

8. Block 10, if Hebrew file has ended and Roman file has not:

"EOF(1) and EOF(2) not coincident"

9. Block 10, if Roman file has ended and Hebrew file has not:

"EOF(2) and EOF(1) not coincident"

Summary of Operating Messages

The following operating messages are printed as the program runs to give the user documentation of information inputs and to indicate the completion of key program phases:

1. Block 0, Front End: a listing of the binary input and output files opened for the program run.
2. Block 0, Front End: a listing of the cloze variables set.
3. Block 1: Confirmation of input and output files opened and passed to main program from front end.
4. Block 1: A message confirming successful reading of Hebrew input file:

"Print format instructions in Hebrew file are: (;xx...) (ASC code for carriage return)."

5. Block 1: A message confirming successful reading of Roman input file:

"First two bytes of Roman file in b3 & b4 = (ASC codes for bytes)."

6. Block 10: Message "Run completed."
7. Closing Routine: "END" when user replies "no" to the option of rerunning the program.

7. Applications and Evaluation

1. Applications

The transliteration system described has two main areas of application: 1) publishing; 2) the teaching of reading in Hebrew to native speakers of languages using the Roman script. A further ancillary area, with minor modifications to the system, is 3) the teaching of Hebrew writing to native speakers of languages using the Roman script.

1) Publishing

The main focus of this paper has been in the teaching of reading in Hebrew. A few words on publishing are, however, relevant as some need for a transliteration system in publishing is apparent at the present time.

As an example, one vendor specialising in CD-ROM applications, e.g. editions of major dictionaries on compact disc, is currently enquiring about a transliteration system for a project in hand for putting the Bible on compact disc both in original Hebrew and Greek texts and transliteration. Provided such a vendor is prepared to work within the MLS environment, this system would, after some refinements referred to in the course of the above discussion, be applicable. Particularly valuable would be the BITEX file with the transliterated text immediately beneath the Hebrew. An independent transliterated text could also be produced and printed side by side with the Hebrew.

Other publishers, wishing to quote Hebrew text with transliteration or in transliteration alone, e.g. in biblical commentaries or linguistic texts for English-speaking readers,

could find a use for such a system.

For such applications, the transliteration system should be integrated with the overall publishing system in use, whether this is in a micro computer desk-top publishing environment for smaller projects or a customized mini computer or main frame environment for larger ones.

2) Teaching reading

The application of the system for the teaching of reading in Hebrew has been the main focus of this paper. It is intended to be a tool for the teacher to use. The teacher decides which texts to use, varying the texts according to the ability and level of the individual student.

Ideally, each student has a computer terminal, either in the classroom or in a computer lab. The BITEX files are prepared ahead of time in readiness for the class. The students can proceed at their own pace with the teacher available for consultation. Once the students are used to the system and have acquired some experience with Hebrew reading they can work on reading assignments without the teacher being present. The assignments can then be "homework", to be done after class. If a student has an IBM compatible PC at home, then he or she can work at home.

The type of output the students view on the computer screen is illustrated in Appendix 9 (e) through (o). The students' task is to work through the series of cloze files until they can read the Hebrew text without the aid of transliteration. They have the freedom to return to easier levels for review. The teacher may, at a certain point,

restrict the files available so that the student is forced to work at the more difficult levels. The teacher may also use the cloze files for testing purposes.

It is evident from Appendix 9 that the system can be run on hard copy if computer terminals are not available. Students are simply handed the requisite sheets to work on. Computer terminals have the obvious advantage of eliminating the chore of copying, storing and handling large numbers of sheets by both teacher and students. The computer can also be used to advantage in timing both the reading and supervised tests. Therefore computer terminals are recommended.

3) Teaching writing

This paper has concentrated on the problem of reading in a non-Roman script. The system can also be modified by adding a cloze option for the Hebrew as well as the Roman text. It could then be used for working at the same time on writing in the non-Roman script with the attendant problems of spelling. The cloze procedure can then be used to block out either the Hebrew or the Roman word or both. The Roman word blocked out directs reading to the target script. The Hebrew word blocked out calls for writing in the target script, either with the Roman word present as a prompt or the Roman word absent with no prompt. Again, this could be done as a self-study assignment or as a supervised assignment with access limited only to certain files.

2. Evaluation

Evaluation of the system in the teaching of reading requires consideration of two issues: target student population

and hypothesis testing.

1) Target student population

The system is using the students' decoding automaticity in the Roman script as a bridge to reach the Hebrew script. Students must therefore be literate in the Roman script. This excludes automatically very young children who have not fully mastered reading and also excludes illiterate adults. In the case of illiterate adults, it is an interesting question whether they should acquire literacy in their first language before trying to acquire literacy in a second. Presumably the answer is yes, because they would have had long exposure to Roman script in their environment - advertisements, TV, picture captions etc. - and could therefore be expected more easily to "get the hang of it" in a familiar looking script than in a totally strange one. This may not necessarily hold true if they were moving permanently into a Hebrew speaking environment. In such a case, they would be starting grade 1 in speaking and reading in the new script. They would, however, need special attention to whatever problems have caused their functional illiteracy in their first language.

The target student population for this system is, therefore, literate adult and high school native speakers of English who wish to learn Modern or Classical Hebrew or both. The system could also be extended to selected intermediate and upper elementary students. The prerequisites for students are literacy in the Roman script, motivation to learn Hebrew and ability to work at a computer terminal. Typical motivations are: intentions to emigrate to Israel, desire to communicate

with Hebrew-speaking family and friends, wish to study the Hebrew Bible or other classical Hebrew literature, wish to read modern Hebrew publications and literature, and general linguistic interests.

2) Hypothesis testing

Several hypotheses require testing:

(a) that native English speakers who cannot read Hebrew acquire decoding automaticity in Hebrew faster with this transliteration system than working with the Hebrew script alone;

(b) that they learn to read Hebrew with comprehension faster and better with this system than working with the Hebrew script alone;

(c) that they learn to write faster and more accurately in Hebrew with this system than working with the Hebrew script alone (this is without explicitly using the system for writing by modifying the cloze feature to include optionally the Hebrew text, as discussed above, but rather to test the effect on writing of the system's particular attention to decoding skills in reading);

(d) that, after the formal course of instruction, they tend to continue to read in Hebrew more and to acquire a larger vocabulary after learning with this system than learning with the Hebrew script alone;

The following requirements are necessary to conduct this evaluation:

(a) An environment where Hebrew is taught and spoken, e.g. a Jewish high school with a strong Hebrew program, Hebrew classes for adults outside Israel or Hebrew classes for adult

immigrants inside Israel. Classes inside Israel would be favoured because of the higher level of motivation induced by the urgency to learn. Classes for teaching Biblical Hebrew could also be used for testing hypotheses (a), (b) and (c).

(b) A well structured course of instruction with well designed and clearly gradable tests in reading and writing.

(c) A teacher or teachers genuinely interested in the project and willing to co-operate in structuring their courses appropriately and administering test instruments.

(d) An experimental and control group formed by random selection, the size of the groups depending upon the number of students available. Fifteen per group would be satisfactory. Both groups should be taught by the same teacher in order to eliminate the teacher difference variable. If there were enough students, more than one experimental and control group could be formed with additional teachers participating.

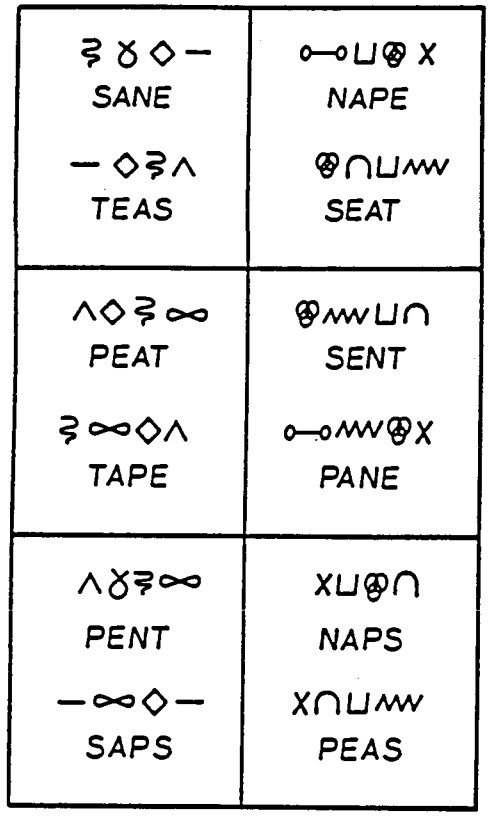
(e) Pre and post tests in the language skills of reading and writing, the pretest being given before the course of instruction begins and post tests after each agreed stage of evaluation. Standard instruments for Hebrew as a second language should be selected corresponding to those available in English as a second language, e.g. Wilkins Definitions of Levels (in Trim, 1978) and others.

Careful records should be maintained of all tests administered as the course of instruction progresses. A possible shedule is to test all hypotheses, except hypothesis (d), on a monthly basis for an academic year or on a weekly basis for an intensive three or five month course. An academic

year or at least five months is preferred as calendar time may be an important factor in consolidating learning. Hypothesis (d), the experimental group continuing reading and vocabulary expansion more than the control group after the class is over, can only be tested by a longitudinal study continuing for one or two years after the course of instruction ends.

This page is inserted to facilitate optional printing on both sides of the page, each figure or appendix starting with an odd number.

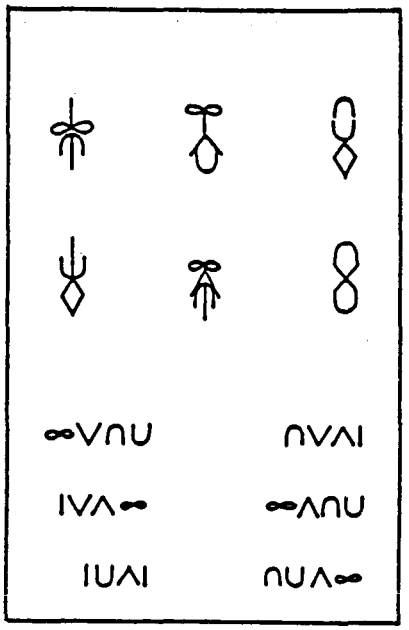
FIGURE 1



For the list of words on the right, there is no systematic correspondence between the symbols and the letters or sounds of the associated response. For the words in the left column there is an alphabetic relation if the symnols are decoded from right to left. (Reproduced from Lee Brooks and Amima Miller, Knowledge of an Alphabet, in Kolers, Wrolstad & Bouma, Processing of Visible Language, Volume 1, p. 392.)

This page is inserted to facilitate optional printing on both sides of the page, each figure or appendix starting with an odd number.

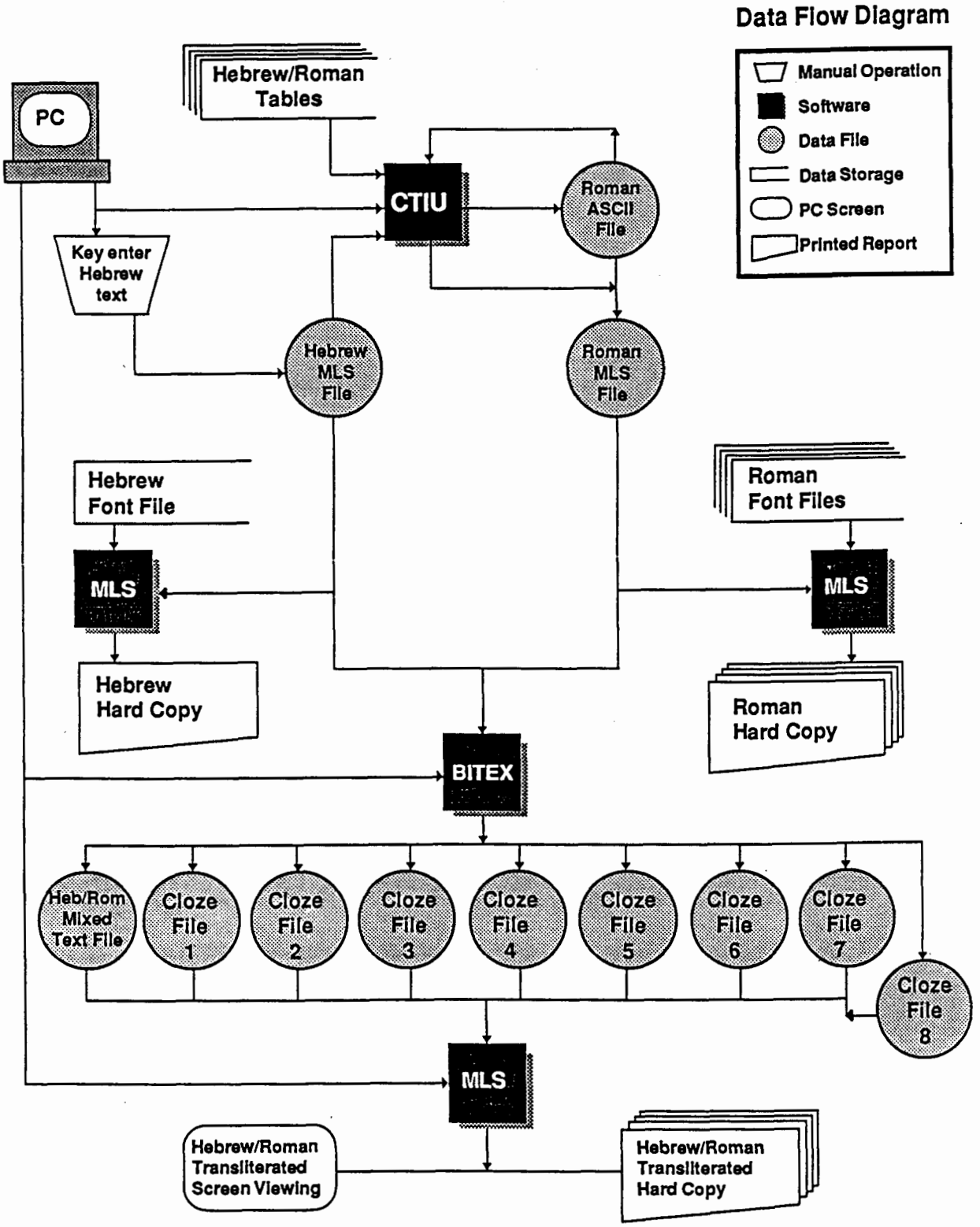
FIGURE 2



Two ways of arranging the artificial letters. Both the glyphic form, to be scanned from top to bottom, and the discrete form, to be scanned from left to right, are composed of the same artificial letters and consequently have the same potential for signalling phonological values. (Reproduced from Lee Brooks and Amima Miller, Knowlege of an Alphabet, in Kolers, Wrolstad & Bouma, Processing of Visible Language, Volume 1, p 396.)

This page is inserted to facilitate optional printing on both sides of the page, each appendix starting with an odd number

APPENDIX 1
Data Flow Diagram



This page is included to facilitate optional printing on both sides of the page, each appendix starting with an odd number.

APPENDIX 2

Table of Hebrew to Roman ASCII Codes and Alphabetic Representations

Abbreviations:
C - Roman "classical" alphabet
CA - Roman "classical alternative" alphabet
M - Roman "modern" alphabet
SM - Roman "simplified modern" alphabet

Hebrew Ascii	Key	Char	Roman Ascii	Key	C	CA	M	SM	Notes
001	Ctl-a	א	014	Ctl-n			y	y	alt've "y" in Mod Heb
003	Ctl-c	כ	003	Ctl-c	k	k	k	k	doubled by tables
004	Ctl-d	ת	022	Ctl-v			v	v	alt've "v" in Mod Heb
005	Ctl-e	ש	023	Ctl-w	š	š*	š*	š*	dbl š or *sh from tbls
066	Sh-b	ב	066	Sh-b	b	b	b	b	doubled by tables
067	Sh-c	כ	067	Sh-c	k	k	k	k	doubled by tables
068	Sh-d	ד	068	Sh-d	d	d	d	d	doubled by tables
069	Sh-e	שׂ	069	Sh-e	ś	ś	ś	s	doubled by tables
070	Sh-f	פ	070	Sh-f	p	p	p	p	doubled by tables
			080	Sh-p	p	p	p	p	Heb 070 on Sh-f, -p
071	Sh-g	ג	071	Sh-g	g	g	g	g	doubled by tables
072	Sh-h	ה	072	Sh-h	h	h	h	h	see note 2
073	Sh-i	ע	073	Sh-i	‘	‘	‘	‘	does not take dagesh
074	Sh-j	ט	074	Sh-j	ṭ	ṭ	ṭ	t	doubled by tables
076	Sh-l	ל	076	Sh-l	l	l	l	l	doubled by tables
077	Sh-m	מ	077	Sh-m	m	m	m	m	doubled by tables
078	Sh-n	נ	078	Sh-n	n	n	n	n	doubled by tables
081	Sh-q	ק	081	Sh-q	q	q	q	k	doubled by tables
082	Sh-r	ר	082	Sh-r	r	r	r	r	does not take dagesh
083	Sh-s	ס	083	Sh-s	s	s	s	s	doubled by tables
084	Sh-t	ת	084	Sh-t	t	t	t	t	doubled by tables
087	Sh-w	שׁ	087	Sh-w	š	š*	š*	š*	*sh from tables
088	Sh-x	צ	088	Sh-x	ṣ	ṣ*	ṣ*	ṣ*	dbl ṣ or *tz from tbls
089	Sh-y	י	089	Sh-y	y	y	y	y	doubled by tables
090	Sh-z	ז	089	Sh-z	z	z	z	z	doubled by tables

APPENDIX 2 (ctd)

Ascii	Key	Char	Ascii	Key	C	CA	M	SM	Notes
097	Reg-a	א	097	Reg-a	'	'	'	'	guttural
			065	Sh-a	'	'	'	'	Heb 097 on Reg-a, Sh-a
098	Reg-b	ב	098	Reg-b	b̲	b*	b*	v	*bh from tables
099	Reg-c	כ	099	Reg-c	k̲	k*	k*	c*	*kh or *ch from tables
100	Reg-d	ד	100	Reg-d	d̲	d*	d	d	*dh from tables
101	Reg-e	שׂ	101	Reg-e	ś	ś	ś	s	
102	Reg-f	פ	102	Reg-f	p̄	p*	p*	f	*ph from tables
			112	Reg-p	p̄	p*	p*	f	Heb 102 on Reg-f, -p
103	Reg-g	ג	103	Reg-g	ḡ	g*	g	g	*gh from tables
104	Reg-h	ה	104	Reg-h	h	h	h	h	guttural
105	Reg-i	ע	105	Reg-i	‘	‘	‘	‘	guttural
106	Reg-j	ט	106	Reg-j	ṭ	ṭ	ṭ	t	
107	Reg-k	ח	107	Reg-k	ḥ	ḥ	ḥ	ḥ	guttural
			075	Sh-k	ḥ	ḥ	ḥ	ḥ	Heb 107 on Reg-k, Sh-k
108	Reg-l	ל	108	Reg-l	l	l	l	l	
109	Reg-m	מ	109	Reg-m	m	m	m	m	
110	Reg-n	נ	110	Reg-n	n	n	n	n	
111	Reg-o	וֹ	111	Reg-o	ô	ô	ô	o	
			079	Sh-o	ô	ô	ô	o	Heb 111 on Reg-o, Sh-o
113	Reg-q	ק	113	Reg-q	q	q	q	k	
114	Reg-r	ר	114	Reg-r	r	r	r	r	
115	Reg-s	ס	115	Reg-s	s	s	s	s	
116	Reg-t	ת	116	Reg-t	t̲	t*	ʿt	t	*th from tables
117	Reg-u	וּ	117	Reg-u	û	û	û	u	
			085	Sh-u	û	û	û	u	Heb 117 on Reg-u, Sh-u
118	Reg-v	ו	118	Reg-v	w	w	v	v	
			086	Sh-v	w	w	v	v	Heb 118 on Reg-v, Sh-v
119	Reg-w	שׁ	119	Reg-w	š	š*	š*	š*	*sh from tables
120	Reg-x	צ	120	Reg-x	ṣ	ṣ*	ṣ*	ṣ*	*tz from tables
121	Reg-y	י	121	Reg-y	y	y	y	y	
122	Reg-z	ז	122	Reg-z	z	z	z	z	

APPENDIX 2 (ctd)

Ascii	Key	Char	Ascii	Key	C	CA	M	SM	Notes
138	Alt-p	ף	138	Alt-p	p	p	p	p	Heb sofit (last letter)
146	Alt-f	ף	146	Alt-f	p̄	p*	p*	f	Heb sofit *ph from tbls
158	Alt-x	ץ	158	Alt-x	ṣ	ṣ*	ṣ*	ṣ*	Heb sofit *tz from tbls
159	Alt-c	ך	159	Alt-c	k̲	k*	k*	c*	Heb sofit *kh/ch f tbls
162	Alt-n	ן	162	Alt-n	n	n	n	n	Heb sofit
163	Alt-m	ם	163	Alt-m	m	m	m	m	Heb sofit
172	F1	ִ	009	Ctl-i	i	i	i	i	Heb overstrike
173	F2	ַ	001	Ctl-a	a	a	a	a	Heb overstrike
174	F3	ָ	019	Ctl-s	ā	ā	ā	a	Heb overstrike
175	F4	ֵ	018	Ctl-r	ē	ē	ē	e	Heb overstrike
176	F5	ֶ	005	Ctl-e	e	e	e	e	Heb overstrike
177	F6	ֻ	021	Ctl-u	u	u	u	u	Heb overstrike
178	F7	ְ	017	Ctl-q	e	e	e	e	Heb overstrike
179	F8	ֲ	004	Ctl-d	ă	ă	ă	a	Heb overstrike
180	F9	ֳ	016	Ctl-p	ŏ	ŏ	ŏ	o	Heb overstrike
181	F10	ֱ	020	Ctl-t	ĕ	ĕ	ĕ	e	Heb overstrike
197	Sh-F1	ֹ	015	Ctl-o	ō	ō	ō	o	Heb o/strike med char
198	Sh-F2	ׄ	015	Ctl-o	ō	ō	ō	o	Heb o/strike nar char
199	Sh-F3	ך	019	Ctl-s	ā	ā	ā	a	Heb o/strike kaf sofit
200	Sh-F4	שׁ	015	Ctl-o	ō	ō	ō	o	Heb o/strike wide char
201	Sh-F5	ֽ							Hb o/str metheg nulled
202	Sh-F6	ֽ	126	Sh- ~	.	.	.	.	Hb o/str prim stress
203	Sh-F7	ך	017	Ctl-q	e	e	e	e	Heb o/strike kaf sofit
204	Sh-F8	ׇ	126	Sh- ~	.	.	.	.	Hb o/str prim stress
243	Alt -	־	045	Reg -	-	-	-	-	maqqef or hyphen

APPENDIX 2 (ctd)

Note 1: Three versions of ḥolem are provided (197, 198, 200) and two versions of the primary stress marker (202, 204) to accomodate varying widths of Hebrew letters.

Note 2: Hē with a dot (ה 072) is sharply audible at the end of a word. The dot is called "mappiq" (bringing out), not dagesh. It is distinguished from regular ה h by doubling (hh), following the transliteration usage of the Society of Biblical Literature.

APPENDIX 3

Table of Roman ASCII Codes and Alphabetic Representations

Abbreviations:
C - Roman "classical" alphabet
CA - Roman "classical alternative" alphabet
M - Roman "modern" alphabet
SM - Roman "simplified modern" alphabet

Ascii	Key	C	CA	M	SM	Notes
001	Ctl-a	a	a	a	a	Heb pathaḥ
003	Ctl-c	k	k	k	k	Heb dagesh kaph sofit: tbls dbl
004	Ctl-d	ă	ă	ă	a	Heb furtive (reduced) pathaḥ
005	Ctl-e	e	e	e	e	Heb s^{e}ghol
009	Ctl-i	i	i	i	i	Heb ḥireq (short)
014	Ctl-n			y	y	see note on Heb 001
015	Ctl-o	ō	ō	ō	o	Heb ḥolem
016	Ctl-p	ŏ	ŏ	ŏ	o	Heb furtive (reduced) qametz
017	Ctl-q	•	•	•	•	Heb shewa (sheva)
018	Ctl-r	ē	ē	ē	e	Heb tzere
019	Ctl-s	ā	ā	ā	a	Heb qametz
020	Ctl-t	ĕ	ĕ	ĕ	e	Heb furtive (reduced) s^{e}ghol
021	Ctl-u	u	u	u	u	Heb qibbutz
023	Ctl-w	š	š*	š *	š*	Heb dag shin: tbls dbl š or *sh
025	Ctl-y	î	î	î	i	Heb ḥireq + yodh (long, יִ)

APPENDIX 3 (ctd)

Ascii	Key	C	CA	M	SM	Notes
065	Sh-a	'	'	'	'	Hebrew 'aleph
066	Sh-b	b	b	b	b	Heb dagesh beth: tables double
067	Sh-c	k	k	k	k	Heb dagesh kaph: tables double
068	Sh-d	d	d	d	d	Heb dagesh daleth: tbls double
069	Sh-e	ś	ś	ś	s	Heb dagesh sin: tables double
070	Sh-f	p	p	p	p	Heb dagesh pe: tables double
071	Sh-g	g	g	g	g	Heb dagesh gimel: tables double
072	Sh-h	h	h	h	h	Heb mappiq he: tables double
073	Sh-i	‘	‘	‘	‘	Hebrew ‘ayin
074	Sh-j	ṭ	ṭ	ṭ	t	Heb dagesh ṭeth: tables double
075	Sh-k	ḥ	ḥ	ḥ	ḥ	Hebrew ḥeth
076	Sh-l	l	l	l	l	Heb dag lamedh: tables double
077	Sh-m	m	m	m	m	Heb dagesh mem: tables double
078	Sh-n	n	n	n	n	Heb dagesh nun: tables double
079	Sh-o	ô	ô	ô	o	Hebrew waw (vav) + ḥolem
080	Sh-p	p	p	p	p	Heb dagesh pe: tables double
081	Sh-q	q	q	q	q	Heb dagesh qoph: tables double
082	Sh-r	r	r	r	r	Hebrew resh
083	Sh-s	s	s	s	s	Heb dag samekh: tables double
084	Sh-t	t	t	t	t	Heb dag taw (tav): tbls double
085	Sh-u	û	û	û	u	Hebrew waw (vav) + shureq
086	Sh-v	w	w	v	v	Hebrew waw (vav)
087	Sh-w	š	š*	š*	š*	Hebrew shin: *sh from tbls
088	Sh-x	ṣ	ṣ	ṣ	s	Heb dag tzadhe: tables double
089	Sh-y	y	y	y	y	Heb dag yodh: tables double
090	Sh-z	z	z	z	z	Heb dagesh zayin: tbls double

APPENDIX 3 (ctd)

Ascii	Key	C	CA	M	SM	Notes
097	Reg-a	’	’	’	’	Hebrew ’aleph
098	Reg-b	b̲	b*	b*	v	Heb beth: *bh from tables
099	Reg-c	k̲	k*	k*	c*	Heb kaph: *kh or ch from tbls
100	Reg-d	d̲	d*	d	d	Heb daleth: *dh from tables
101	Reg-e	ś	ś	ś	s	Hebrew sin
102	Reg-f	p̄	p*	p*	f	Heb pe: *ph from tables
103	Reg-g	ḡ	g*	g	g	Heb gimel: *gh from tables
104	reg-h	h	h	h	h	Hebrew he
105	Reg-i	‘	‘	‘	‘	Hebrew ‘ayin
106	Reg-j	ṭ	ṭ	ṭ	t	Hebrew ṭeth
107	Reg-k	ḥ	ḥ	ḥ	ḥ	Hebrew ḥeth
108	Reg-l	l	l	l	l	Hebrew lamedh
109	Reg-m	m	m	m	m	Hebrew mem
110	Reg-n	n	n	n	n	Hebrew nun
111	Reg-o	ô	ô	ô	o	Hebrew waw (vav) + ḥolem
112	Reg-p	p̄	p*	p*	f	Hebrew pe: *ph from tables
113	Reg-q	q	q	q	k	Hebrew qoph
114	Reg-r	r	r	r	r	Hebrew resh
115	Reg-s	s	s	s	s	Hebrew samekh
116	Reg-t	t̲	t*	t	t	Heb taw (tav): *th from tbls
117	Reg-u	û	û	û	u	Heb waw (vav) + shureq
118	Reg-v	w	w	v	v	Hebrew waw (vav)
119	Reg-w	š	š*	š*	š*	Hebrew shin: *sh from tables
120	Reg-x	ṣ	ṣ*	ṣ*	ṣ*	Hebrew tzadhe: *tz from tbls
121	Reg-y	y	y	y	y	Hebrew yodh
122	Reg-z	z	z	z	z	Hebrew zayin

APPENDIX 3 (ctd)

Ascii	Key	C	CA	M	SM	Notes
131	Alt-e	ê	ê	ê	e	Hebrew tzere + yodh
132	Alt-r	ệ	ệ	ệ	e	Hebrew s°ghol + yodh
133	Alt-t	t	t	t	t	Constant t for table ref
137	Alt-o	o	o	o	o	Constant o for table ref
138	Alt-p	p	p	p	p	Heb dag pe sofit: tbls double
143	Alt-a	â	â	â	a	Hebrew qametz + yodh
144	Alt-s	f	f	f	f	Constant f for table ref
145	Alt-d	d	d	d	d	Constant d for table ref
146	Alt-f	p̄	p*	p*	f	Heb pe sofit: *ph from tables
147	Alt-g	g	g	g	g	Constant g for table ref
149	Alt-j	c	c	c	c	Constant c for table ref
150	Alt-k	k	k	k	k	Constant k for table ref
158	Alt-x	ṣ	ṣ*	ṣ*	ṣ*	Heb tzadhe sofit: *tz from tbls
159	Alt-c	k̲	k*	k*	c*	Heb kaph sofit: *kh/ch fr tbls
160	Alt-v	v	v	v	v	Constant v for table ref
161	Alt-b	b	b	b	b	Constant b for table ref
162	Alt-n	n	n	n	n	Hebrew nun sofit
163	Alt-m	m	m	m	m	Hebrew mem sofit

APPENDIX 4
Table to Convert MLS Hebrew to ASCII Roman Text Files

The first of each pair of expressions is ASCII, the second MLS. The Table is arranged in sorted order: from highest to lowest number of terms in the MLS expressions.

Wn\001\y\009\163	Rule 1 - Special terminology and
W\178\n\173\y\172\163	pronunciation exceptions.
Wt\001\y\009\163	- shenayim = shnayim
W\178\t\173\y\172\163	- shetayim = shtayim
Wn\018\045	- sheney- = shne-
W\178\n\175\y\243	- shetey- = shte-
Wt\018\045	
W\178\t\175\y\243	
\001\don\001\009	- yeya = adonai
\y\178\y\174	
\137\b\45	Rule 3 - Short qametz before maqqef
\174\b\178\243	(hyphen) which is preceded
\137\c\45	by a letter with sheva.
\174\c\178\243	The sheva is quiescent.
\137\d\45	
\174\d\178\243	
\137\e\45	
\174\e\178\243	
\137\f\45	
\174\f\178\243	
\137\g\45	
\174\g\178\243	
\137\i\45	
\174\i\178\243	
\137\j\45	
\174\j\178\243	
\137\k\45	
\174\k\178\243	
\137\l\45	
\174\l\178\243	
\137\m\45	
\174\m\178\243	
\137\n\45	
\174\n\178\243	
\137\q\45	
\174\q\178\243	
\137\r\45	
\174\r\178\243	
\137\s\45	
\174\s\178\243	
\137\t\45	
\174\t\178\243	
\137\v\45	
\174\v\178\243	

APPENDIX 4 (ctd)

```
\137\w\45
\174\w\178\243
\137\x\45
\174\x\178\243
\137\z\45
\174\z\178\243
\137\W\45
\174\W\178\243
\137\a\016
\174\201\a\180
\137\h\016
\174\201\h\180
\137\i\016
\174\201\i\180
\137\k\016
\174\201\k\180
\137\b\45
\174\b\243
\137\c\45
\174\c\243
\137\d\45
\174\d\243
\137\e\45
\174\e\243
\137\102\45
\174\102\243
\137\g\45
\174\g\243
\137\j\45
\174\j\243
\137\k\45
\174\k\243
\137\l\45
\174\l\243
\137\m\45
\174\m\243
\137\n\45
\174\n\243
\137\q\45
\174\q\243
\137\r\45
\174\r\243
\137\s\45
\174\s\243
\137\t\45
\174\t\243
\137\v\45
\174\v\243
\137\w\45
\174\w\243
```

Rule 3 - Short qametz with metheg before furtive qametz (see also rule 7).

Rule 4 - Short qametz before maqqef (hyphen).

APPENDIX 4 (ctd)

\137\x\45
\174\x\243
\137\y\45
\174\y\243
\137\z\45
\174\z\243
\137\W\45
\174\W\243
\137\a\016
\174\a\180
\137\h\016
\174\h\180
\137\i\016
\174\i\180
\137\k\016
\174\k\180

Rule 5 - Short qametz before furtive qametz.

\32\B\17
\32\B\178
\32\C\17
\32\C\178
\32\D\17
\32\D\178
\32\e\17
\32\e\178
\32\70\17
\32\70\178
\32\G\17
\32\G\178
\32\j\17
\32\j\178
\32\k\17
\32\k\178
\32\l\17
\32\l\178
\32\m\17
\32\m\178
\32\n\17
\32\n\178
\32\q\17
\32\q\178
\32\r\17
\32\r\178
\32\s\17
\32\s\178
\32\T\17
\32\T\178
\32\v\17
\32\v\178
\32\w\17
\32\w\178

Rule 6 - Mobile sheva under first letter of a word. When this is the first of two shevaim together, the second sheva is quiescent (see Roman Table Rule 8).

APPENDIX 4 (ctd)

\32\x\17
\32\x\178
\32\z\17
\32\z\178
\32\W\17
\32\W\178
\13\10\B\17
\128\B\178
\13\10\C\17
\128\C\178
\13\10\D\17
\128\D\178
\13\10\e\17
\128\e\178
\13\10\70\17
\128\70\178
\13\10\G\17
\128\G\178
\13\10\j\17
\128\j\178
\13\10\k\17
\128\k\128
\13\10\l\17
\128\l\178
\13\10\m\17
\128\m\178
\13\10\n\17
\128\n\178
\13\10\q\17
\128\q\178
\13\10\r\17
\128\r\178
\13\10\s\17
\128\s\178
\13\10\T\17
\128\T\178
\13\10\v\17
\128\v\178
\13\10\w\17
\128\w\178
\13\10\x\17
\128\x\178
\13\10\z\17
\128\z\178
\13\10\W\17
\128\W\178
\45\B\17
\243\B\178
\45\C\17
\243\C\178

APPENDIX 4 (ctd)

```
\45\D\17
\243\D\178
\45\e\17
\243\e\178
\45\70\17
\243\70\178
\45\G\17
\243\G\178
\45\j\17
\243\j\178
\45\k\17
\243\k\178
\45\l\17
\243\l\178
\45\m\17
\243\m\178
\45\n\17
\243\n\178
\45\q\17
\243\q\178
\45\r\17
\243\r\178
\45\s\17
\243\s\178
\45\T\17
\243\T\178
\45\v\17
\243\v\178
\45\w\17
\243\w\178
\45\x\17
\243\x\178
\45\z\17
\243\z\178
\45\W\17
\243\W\178
\233\b\17
\178\b\178
\233\c\17
\178\c\178
\233\d\17
\178\d\178
\233\e\17
\178\e\178
\233\102\17
\178\102\178
\233\g\17
\178\g\178
\233\i\17
\178\i\178
```

Rule 7 - First of two shevaim in the middle or at the end of a word is quiescent. If a dagesh letter follows the quiescent sheva, the dagesh is dagesh lene (see Roman Table Rules 2 & 3).

APPENDIX 4 (ctd)

```
\233\j\17
\178\j\178
\233\k\17
\178\k\178
\233\l\17
\178\l\178
\233\m\17
\178\m\178
\233\n\17
\178\n\178
\233\q\17
\178\q\178
\233\r\17
\178\r\178
\233\s\17
\178\s\178
\233\t\17
\178\t\178
\233\v\17
\178\v\178
\233\w\17
\178\w\178
\233\x\17
\178\x\178
\233\y\17
\178\y\178
\233\z\17
\178\z\178
\233\W\17
\178\W\178
\233\B\17
\178\B\178
\233\C\17
\178\C\178
\233\D\17
\178\D\178
\233\E\17
\178\E\178
\233\70\17
\178\70\178
\233\G\17
\178\G\178
\233\J\17
\178\J\178
\233\L\17
\178\L\178
\233\M\17
\178\M\178
\233\N\17
\178\N\178
```

APPENDIX 4 (ctd)

\233\Q\17
\178\Q\178
\233\S\17
\178\S\178
\233\T\17
\178\T\178
\233\V\17
\178\V\178
\233\X\17
\178\X\178
\233\Y\17
\178\Y\178
\233\Z\17
\178\Z\178
\233\005\17
\178\005\178
\009\b\233
\172\b\178
\001\b\233
\173\b\178
\137\b\233
\174\b\178
\005\b\233
\176\b\178
\021\b\233
\177\b\178
\009\c\233
\172\c\178
\001\c\233
\173\c\178
\137\c\233
\174\c\178
\005\c\233
\176\c\178
\021\c\233
\177\c\178
\009\d\233
\172\d\178
\001\d\233
\173\d\178
\137\d\233
\174\d\178
\005\d\233
\176\d\178
\021\d\233
\177\d\178
\009\e\233
\172\e\178
\001\e\233
\173\e\178

Rule 8 - Quiescent sheva after short vowel. This rule covers short vowels a,e,i,o,and u. Qametz before sheva is read as short "o" unless accompanied by metheg, in which case it is long "a" by default.

APPENDIX 4 (ctd)

\137\e\233
\174\e\178
\005\e\233
\176\e\178
\021\e\233
\177\e\178
\009\102\233
\172\102\178
\001\102\233
\173\102\178
\137\102\233
\174\102\178
\005\102\233
\176\102\178
\021\102\233
\177\102\178
\009\g\233
\172\g\178
\001\g\233
\173\g\178
\137\g\233
\174\g\178
\005\g\233
\176\g\178
\021\g\233
\177\g\178
\009\i\233
\172\i\178
\001\i\233
\173\i\178
\137\i\233
\174\i\178
\005\i\233
\176\i\178
\021\i\233
\177\i\178
\009\j\233
\172\j\178
\001\j\233
\173\j\178
\137\j\233
\174\j\178
\005\j\233
\176\j\178
\021\j\233
\177\j\178
\009\k\233
\172\k\178
\001\k\233
\173\k\178

APPENDIX 4 (ctd)

\137\k\233
\174\k\178
\005\k\233
\176\k\178
\021\k\233
\177\k\178
\009\l\233
\172\l\178
\001\l\233
\173\l\178
\137\l\233
\174\l\178
\005\l\233
\176\l\178
\021\l\233
\177\l\178
\009\m\233
\172\m\178
\001\m\233
\173\m\178
\137\m\233
\174\m\178
\005\m\233
\176\m\178
\021\m\233
\177\m\178
\009\n\233
\172\n\178
\001\n\233
\173\n\178
\137\n\233
\174\n\178
\005\n\233
\176\n\178
\021\n\233
\177\n\178
\009\q\233
\172\q\178
\001\q\233
\173\q\178
\137\q\233
\174\q\178
\005\q\233
\176\q\178
\021\q\233
\177\q\178
\009\r\233
\172\r\178
\001\r\233
\173\r\178

APPENDIX 4 (ctd)

\137\r\233
\174\r\178
\005\r\233
\176\r\178
\021\r\233
\177\r\178
\009\s\233
\172\s\178
\001\s\233
\173\s\178
\137\s\233
\174\s\178
\005\s\233
\176\s\178
\021\s\233
\177\s\178
\009\t\233
\172\t\178
\001\t\233
\173\t\178
\137\t\233
\174\t\178
\005\t\233
\176\t\178
\021\t\233
\177\t\178
\009\v\233
\172\v\178
\001\v\233
\173\v\178
\137\v\233
\174\v\178
\005\v\233
\176\v\178
\021\v\233
\177\v\178
\009\w\233
\172\w\178
\001\w\233
\173\w\178
\137\w\233
\174\w\178
\005\w\233
\176\w\178
\021\w\233
\177\w\178
\009\x\233
\172\x\178
\001\x\233
\173\x\178

APPENDIX 4 (ctd)

\137\x\233
\174\x\178
\005\x\233
\176\x\178
\021\x\233
\177\x\178
\009\y\233
\172\y\178
\001\y\233
\173\y\178
\137\y\233
\174\y\178
\005\y\233
\176\y\178
\021\y\233
\177\y\178
\009\z\233
\172\z\178
\001\z\233
\173\z\178
\137\z\233
\174\z\178
\005\z\233
\176\z\178
\021\z\233
\177\z\178
\009\W\233
\172\W\178
\001\W\233
\173\W\178
\137\W\233
\174\W\178
\005\W\233
\176\W\178
\021\W\233
\177\W\178
\126\b\233
\202\b\178
\126\c\233
\202\c\178
\126\d\233
\202\d\178
\126\e\233
\202\e\178
\126\102\233
\202\102\178
\126\g\233
\202\g\178
\126\i\233
\202\i\178

Rule 9 - Quiescent sheva after accent.

APPENDIX 4 (ctd)

```
\126\j\233
\202\j\178
\126\k\233
\202\k\178
\126\l\233
\202\l\178
\126\l\233
\204\l\178
\126\m\233
\202\m\178
\126\n\233
\202\n\178
\126\q\233
\202\q\178
\126\r\233
\202\r\178
\126\s\233
\202\s\178
\126\t\233
\202\t\178
\126\v\233
\202\v\178
\126\w\233
\202\w\178
\126\x\233
\202\x\178
\126\z\233
\202\z\178
\126\W\233
\202\W\178
\001\009
\173\y\172
\131\y\09
\175\y\172
\131\y\01
\175\y\173
\131\y\19
\175\y\174
\131\y\18
\175\y\175
\131\y\05
\175\y\176
\131\y\21
\175\y\177
\131\y\17
\175\y\178
\131\y\117
\175\y\117
\131\y\111
\175\y\111
```

Rule 10 - Mute yod before i (ayi --> ai).

Rule 11 - Consonantal yod after long a, e when the yod is followed by a vowel. This rule takes precedence over rules 14 and 15 below, which provide for redundant yod after these vowels.

APPENDIX 4 (ctd)

```
\143\y\09
\174\y\172
\143\y\01
\174\y\173
\143\y\19
\174\y\174
\143\y\18
\174\y\175
\143\y\05
\174\y\176
\143\y\21
\174\y\177
\143\y\17
\174\y\178
\143\y\117
\174\y\117
\143\y\111
\174\y\111
\143\y\32          Rule 12 - consonantal yod (y) after qametz
\174\y\32                    at end of word.
\143\y\.
\174\y\.
\143\y\,
\174\y\,
\143\y\;
\174\y\;
\143\y\:
\174\y\:
\143\y\?
\174\y\?
\143\y\45
\174\y\45
\143\y\13\10
\174\y\128
\25                Rule 13 - Redundant yod (y) after short i,e
\172\201\y                   and long a,e with metheg
\131
\175\201\y
\132
\176\201\y
\143
\174\201\y
\25\126            Rule 14 - Redundant yod (y) after accented
\25\202\y                    short i,e and long a,e.
\172\126
\131\202\y
\175\126
\132\202\y
\176\126
\143\202\y
```

APPENDIX 4 (ctd)

\25
\172\y
\131
\175\y
\132
\176\y
\143
\174\y
\32
\h\32
.
h.
,
h,
;
h;
:
h:
?
h?
\45
\h\45
\13\10
\h\128
\13\10
\128
\32
\32
y
\01
*
\02
\03
\03
\160
\04
WW
\05
*
\06
*
\07
*
\08
*
\10
*
\11
*
\12

Rule 15 - Redundant yod (y) after short i,e and long a,e.

Rule 16 - Redundant he (h) when the last letter, without a vowel, at end of word.
- This rule can be deleted when wishing to indicate Hebrew spelling. Some authorities show final he, others do not.

APPENDIX 4 (ctd)

```
*
\14
\22
\22
\23\23              Dagesh shin configured on Ctl-w (ASCII 23)
\23
*
\25
!
\33
"
\34
#
\35
$
\36
%
\37
&
\38
'
\39
(
\40
)
\41
*
\42
+
\43
,
\44
-
\45
.
\46
/
\47
0
\48
1
\49
2
\50
3
\51
4
\52
5
\53
```

APPENDIX 4 (ctd)

6
\54
7
\55
8
\56
9
\57
:
\58
;
\59
<
\60
=
\61
>
\62
?
\63
@
\64
BB
\66
CC
\67
DD
\68
EE
\69
FF
\70
GG
\71
H
\72
I
\73
JJ
\74
K
\75
LL
\76
MM
\77
NN
\78
QQ
\81

Upper case used for dagesh consonants. All dagesh letters converted to dagesh forte (double letters). The Roman tables identify when the six "begadkefat" letters (b,g,d,k, f,t) are hard (dagesh lene).

APPENDIX 4 (ctd)

R
\82
SS
\83
TT
\84
VV
\86
W
\87
XX
\88
YY
\89
ZZ
\90
[
\91
\92
\92
]
\93
^
\94
_
\95
`
\96
\97
\97
b
\98
c
\99
d
\100
e
\101
f
\102
g
\103
h
\104
i
\105
j
\106
k
\107

Dagesh shin configured on Ctl-w (ASCII 23).

APPENDIX 4 (ctd)

l
\108
m
\109
n
\110
o
\111
q
\113
r
\114
s
\115
t
\116
u
\117
v
\118
w
\119
x
\120
y
\121
z
\122
{
\123
¦
\124
}
\125
"
\126
*
\134
*
\136
\138
\138
\146
\146
*
\151
\158
\158
\159
\159

APPENDIX 4 (ctd)

\162
\162
\163
\163
\09
\172
\01
\173
\19
\174
\18
\175
\05
\176
\21
\177
\17
\178
\04
\179
\16
\180
\20
\181
\15
\197
\15
\198
\19
\199
\15
\200
\
\201
\126
\202
\126
\204
\17
\203
\45
\243

This page is inserted to facilitate optional printing on both sides of the page, each appendix starting with an odd number.

APPENDIX 5

Table to convert ASCII Roman to MLS Hebrew Text Files

HTRA is the table used for classical (ancient) Hebrew transliteration. HTRB is an alternative table for classical Hebrew that uses two letters rather than one with a diacritic mark to represent the "begadkepat" letters b,g,d,k,p and t and the letters tzadi and shin. HTRM is a table for modern Hebrew which aspirates only the letters b,k and p and in so doing simplifies "bh" to "v", "ph" to "f" and "kh" to "ch". The HTRB and HTRM variations are noted in this printout.

```
\b\233\BB
bB
\c\233\BB
cB
\d\233\BB
dB
\e\233\BB
eB
\f\233\BB
\102\B
\g\233\BB
gB
\i\233\BB
iB
\j\233\BB
jB
\k\233\BB
\107\B
\l\233\BB
lB
\m\233\BB
mb
\n\233\BB
nB
\q\233\BB
qB
\r\233\BB
rB
\s\233\BB
sB
\t\233\BB
tB
\v\233\BB
vB
\w\233\BB
wB
\x\233\BB
xB
```

Rule 1 - Dagesh lene following quiescent sheva.

APPENDIX 5 (ctd)

```
\z\233\BB
zB
\W\233\BB
WB
\b\233\CC
bC
\c\233\CC
cC
\d\233\CC
dC
\e\233\CC
eC
\f\233\CC
\102\C
\g\233\CC
gC
\i\233\CC
iC
\j\233\CC
jC
\k\233\CC
\107\C
\l\233\CC
lC
\m\233\CC
mC
\n\233\CC
nC
\q\233\CC
qC
\r\233\CC
rC
\s\233\CC
sC
\t\233\CC
tC
\v\233\CC
vC
\w\233\CC
wC
\x\233\CC
xC
\z\233\CC
zC
\W\233\CC
WC
\b\233\DD
bD
\c\233\DD
cD
```

APPENDIX 5 (ctd)

```
\d\233\DD
dD
\e\233\DD
eD
\f\233\DD
\102\D
\g\233\DD
gD
\i\233\DD
iD
\j\233\DD
jD
\k\233\DD
\107\D
\l\233\DD
lD
\m\233\DD
mD
\n\233\DD
nD
\q\233\DD
qD
\r\233\DD
rD
\s\233\DD
sD
\t\233\DD
tD
\v\233\DD
vD
\w\233\DD
wD
\x\233\DD
xD
\z\233\DD
zD
\W\233\DD
WD
\b\233\FF
\b\70
\c\233\FF
\c\70
\d\233\FF
\d\70
\e\233\FF
\e\70
\f\233\FF
\102\70
\g\233\FF
\g\70
```

APPENDIX 5 (ctd)

```
\i\233\FF
\i\70
\j\233\FF
\j\70
\k\233\FF
\107\70
\l\233\FF
\l\70
\m\233\FF
\m\70
\n\233\FF
\n\70
\q\233\FF
\q\70
\r\233\FF
\r\70
\s\233\FF
\s\70
\t\233\FF
\t\70
\v\233\FF
\v\70
\w\233\FF
\w\70
\x\233\FF
\x\70
\z\233\FF
\z\70
\W\233\FF
\W\70
\b\233\GG
bG
\c\233\GG
cG
\d\233\GG
dG
\e\233\GG
eG
\f\233\GG
\102\G
\g\233\GG
gG
\i\233\GG
iG
\j\233\GG
jG
\k\233\GG
\107\G
\l\233\GG
lG
```

APPENDIX 5 (ctd)

```
\m\233\GG
mG
\n\233\GG
nG
\q\233\GG
qG
\r\233\GG
rG
\s\233\GG
sG
\t\233\GG
tG
\v\233\GG
vG
\w\233\GG
wG
\x\233\GG
xG
\z\233\GG
zG
\W\233\GG
WG
\b\233\TT
bT
\c\233\TT
cT
\d\233\TT
dT
\e\233\TT
eT
\f\233\TT
\102\T
\g\233\TT
gT
\i\233\TT
iT
\j\233\TT
jT
\k\233\TT
\107\T
\l\233\TT
lT
m\233\TT
mT
\n\233\TT
nT
\q\233\TT
qT
\r\233\TT
rT
```

APPENDIX 5 (ctd)

```
\s\233\TT
sT
\t\233\TT
tT
\v\233\TT
vT
\w\233\TT
wT
\x\233\TT
xT
\z\233\TT
zT
\W\233\TT
WT
\32\BB                  Rule 2 - Dagesh lene at beginning of word.
\32\B
\32\CC
\32\C
\32\DD
\32\D
\32\FF
\32\70
\32\GG
\32\G
\32\TT
\32\T
\13\10\BB
\128\B
\13\10\CC
\128\C
\13\10\DD
\128\D
\13\10\FF
\128\70
\13\10\GG
\128\G
\13\10\TT
\128\T
\45\BB
\45\B
\45\CC
\45\C
\45\DD
\45\D
\45\FF
\45\70
\45\GG
\45\G
\45\TT
\45\T
```

APPENDIX 5 (ctd)

Rule 3 - Short qametz before dagesh forte.

```
\19\BB
\137\BB
\19\CC
\137\CC
\19\DD
\137\DD
\19\EE
\137\EE
\19\FF
\137\70\70
\19\GG
\137\GG
\19\JJ
\137\JJ
\19\LL
\137\LL
\19\MM
\137\MM
\19\NN
\137\NN
\19\QQ
\137\QQ
\19\SS
\137\SS
\19\TT
\137\TT
\19\XX
\137\XX
\19\YY
\137\YY
\19\ZZ
\137\ZZ
\19\003
\137\003
\19\138
\137\138
```

Rule 4 - Short "a" (pathah) under guttural at end of a word is pronounced before not after the guttural.

```
\k\001\32
\001\107\32
\k\001\.
\001\107\.
\k\001\,
\001\107\,
\k\001\;
\001\007\;
\k\001\:
\001\107\:
\k\001\?
\001\107\?
\k\001\45
\001\107\45
```

APPENDIX 5 (ctd)

```
\k\001\45\10
\001\107\128
\H\001\32
\001\H\32
\H\001\.
\001\H\.
\H\001\,
\001\H\,
\H\001\;
\001\H\;
\H\001\:
\001\H\:
\H\001\?
\001\H\?
\H\001\45
\001\H\45
\H\001\13\10
\001\H\128
\i\001\32
\001\i\32
\i\001\.
\001\i\.
\i\001\,
\001\i\,
\i\001\;
\001\i\;
\i\001\:
\001\i\:
\i\001\?
\001\i\?
\i\001\45
\001\i\45
\i\001\13\10
\001\i\128
\a\001\32
\001\a\32
\a\001\.
\001\a\.
\a\001\,
\001\a\,
\a\001\;
\001\a\;
\a\001\:
\001\a\:
\a\001\?
\001\a\?
\a\001\45
\001\a\45
\a\001\13\10
\001\a\128
```

APPENDIX 5 (ctd)

```
\b\233\b
\b\17\b
\c\233\c
\c\17\c
\d\233\d
\d\17\d
\e\233\e
\e\17\e
\f\233\102
\102\17\102
\g\233\g
\g\17\g
\j\233\j
\j\17\j
\k\233\107
\107\17\107
\l\233\l
\l\17\l
\m\233\m
\m\17\m
\n\233\n
\n\17\n
\q\233\q
\q\17\q
\r\233\r
\r\17\r
\s\233\s
\s\17\s
\t\233\t
\t\17\t
\v\233\v
\v\17\v
\w\233\w
\w\17\w
\x\233\x
\x\17\x
\y\233\y
\y\17\y
\z\233\z
\z\17\z
\W\233\W
\W\17\W
\17\b\17
\17\b
\17\c\17
\17\c
\17\d\17
\17\d
\17\e\17
\17\e
```

Rule 5 - The sheva is mobile in the first of two similar letters even if it would otherwise be quiescent, e.g. if following a short vowel.

Rule 6 - The second of two shevaim together at the beginning of a word is quiescent. This rule complements Hebrew Table Rule 6 which makes the sheva under the first letter of a word mobile.

APPENDIX 5 (ctd)

```
\17\102\17
\17\102
\17\g\17
\17\g
\17\i\17
\17\i
\17\j\17
\17\j
\17\k\17
\17\k
\17\l\17
\17\l
\17\m\17
\17\m
\17\n\17
\17\n
\17\q\17
\17\q
\17\r\17
\17\r
\17\s\17
\17\s
\17\t\17
\17\t
\17\v\17
\17\v
\17\w\17
\17\w
\17\x\17
\17\x
\17\z\17
\17\z
\17\W\17
\17\W
\17\13\10          Rule 7 - Quiescent sheva at end of word.
\128
\17\32
\32
\17\.
\.
\17\,
\,
\17\;
\;
\17\:
\:
\17\?
\?
\17\!
\!
```

APPENDIX 5 (ctd)

\17\45
\45
\13\10
\128
\32
\32
\233
\
\00
\00
\01
\01
\03
\03
\04
\04
\05
\05
\06
\06
\07
\07
\08
\08
\09
\09
\10
\10
\11
\11
\12
\12
\13
\13
\14
\14
\15
\15
\16
\16
\17
\17
\18
\18
\19
\19
\20
\20
\21
\21

APPENDIX 5 (ctd)

\22
\22
\23
\23
\25
\25
\26
\26
!
!
"
"
#
#
$
$
%
%
&
&
'
'
(
(
)
)
*
*
+
+
,
,
-
-
.
.
\47
\47
0
0
1
1
2
2
3
3
4
4
5
5

RTRS renders shin, dag. forte, as a single "sh". RTRA, B, & M process unchanged the double shin recd from the Hebrew table as either "ss" (RTRA) or "shsh" (RTRB, RTRM)

APPENDIX 5 (ctd)

6
6
7
7
8
8
9
9
:
:
;
;
<
<
=
=
>
>
?
?
@
@
A
A
B
B
C
C
D
D
E
E
F
\70
G
G
H
H
I
I
J
J
K
\107
L
L
M
M
N
N

APPENDIX 5 (ctd)

O
O
P
\70
Q
Q
R
R
S
S
T
T
U
U
V
V
W RTRB, M & S render shin as "sh"
W
X RTRB, M & S render tzadi as "tz"
X
Y
Y
Z
Z
[
[
\092
\092
]
]
^
^
_
`
a
a
b RTRB & M render b as "bh", RTRS as "v".
b
c (configured as k) RTRB & M render k as "kh", RTRS as "ch"
c
d RTRB renders d as "dh", RTRM & S as "d".
d
e
e
f (configured as p) RTRB & M render p as "ph", RTRS as "f".
\102
g RTRB renders g as "gh", RTRM & S as "g".
g

APPENDIX 5 (ctd)

h
h
i
i
j
j
k
\107
l
l
m
m
n
n
o
o
p RTRB & M render p as "ph", RTRS as "f".
\102
q
q
r
r
s
s
t RTRB renders t as "th", RTRM & S as "t".
t
u
u
v
v
w (configured as s) RTRB, M & S render shin as "sh".
w
x (configured as s) RTRB, M & S render tzadi as "tz".
x
y
y
z
z
{
{
¦
¦
}
}
~
~
\129
\129
\130
\130

APPENDIX 5 (ctd)

\131
\131
\132
\132
\133
\133
\134
\134
\135
\135
\136
\136
\137
\137
\138
\138
\143
\143
\144
\144
\145
\145
\146 (config as p) RTRB & M render pe sofit as "ph", RTRS
\146 as "f".
\147
\147
\148
\148
\149
\149
\150
\150
\151
\151
\158 (config as s) RTRB, M & S render tzadi sofit as "tz".
\158
\159 (config as k) RTRB & M render kaf sofit as "kh", RTRS
\159 as "ch".
\161
\161
\162
\162
\163
\163
\172
\172
\173
\173
\174
\174

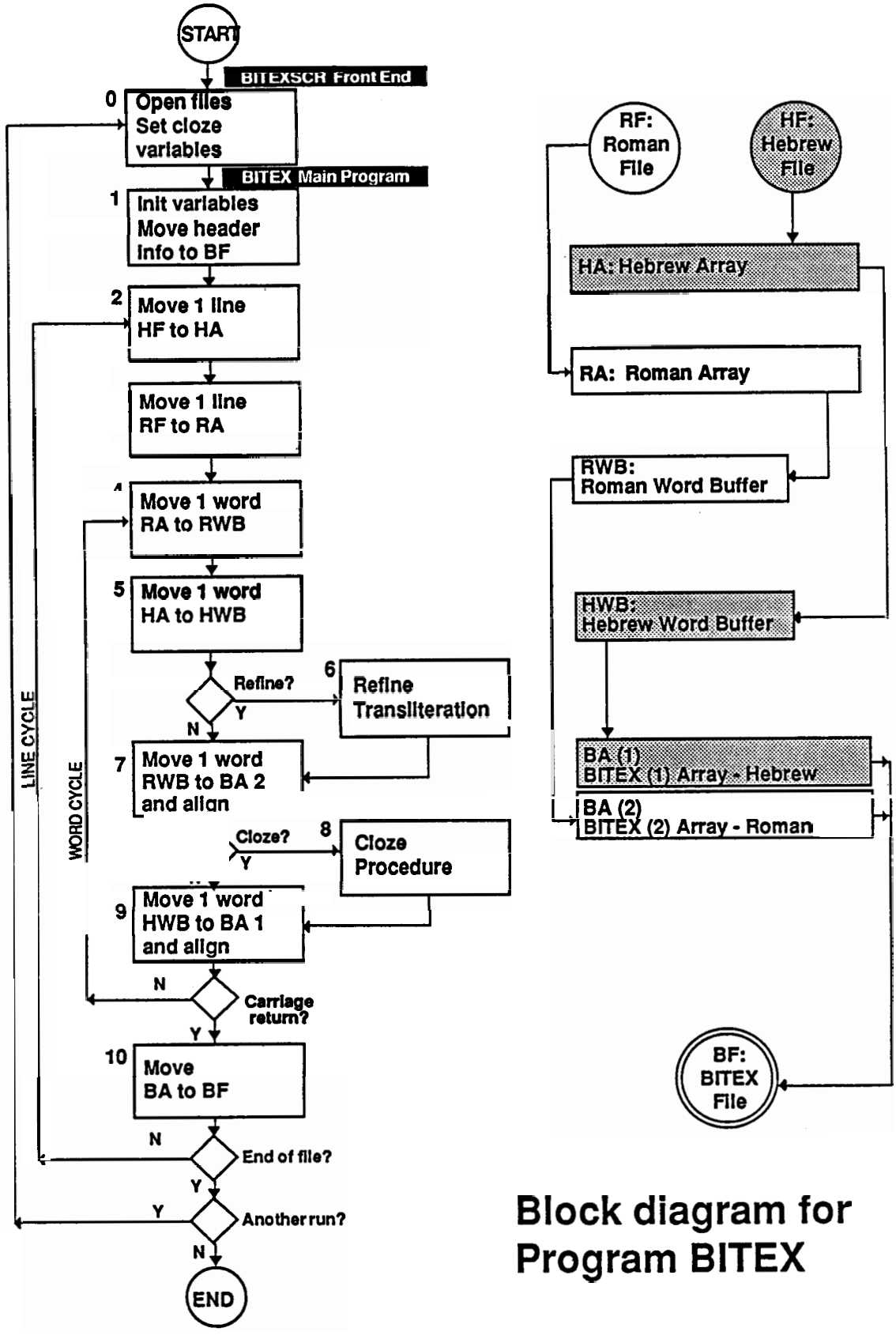
APPENDIX 5 (ctd)

\175
\175
\176
\176
\177
\177
\178
\178
\179
\179
\180
\180
\181
\181
\197
\197
\198
\198
\199
\199
\200
\200
\201
\201
\202
\202
\203
\203
\204
\204
\205
\205
\206
\206
\234
\234
\235
\235
\236
\236
\237
\237
\238
\238
\239
\239
\240
\240
\241
\241

APPENDIX 5 (ctd)

\242
\242
\243
\243

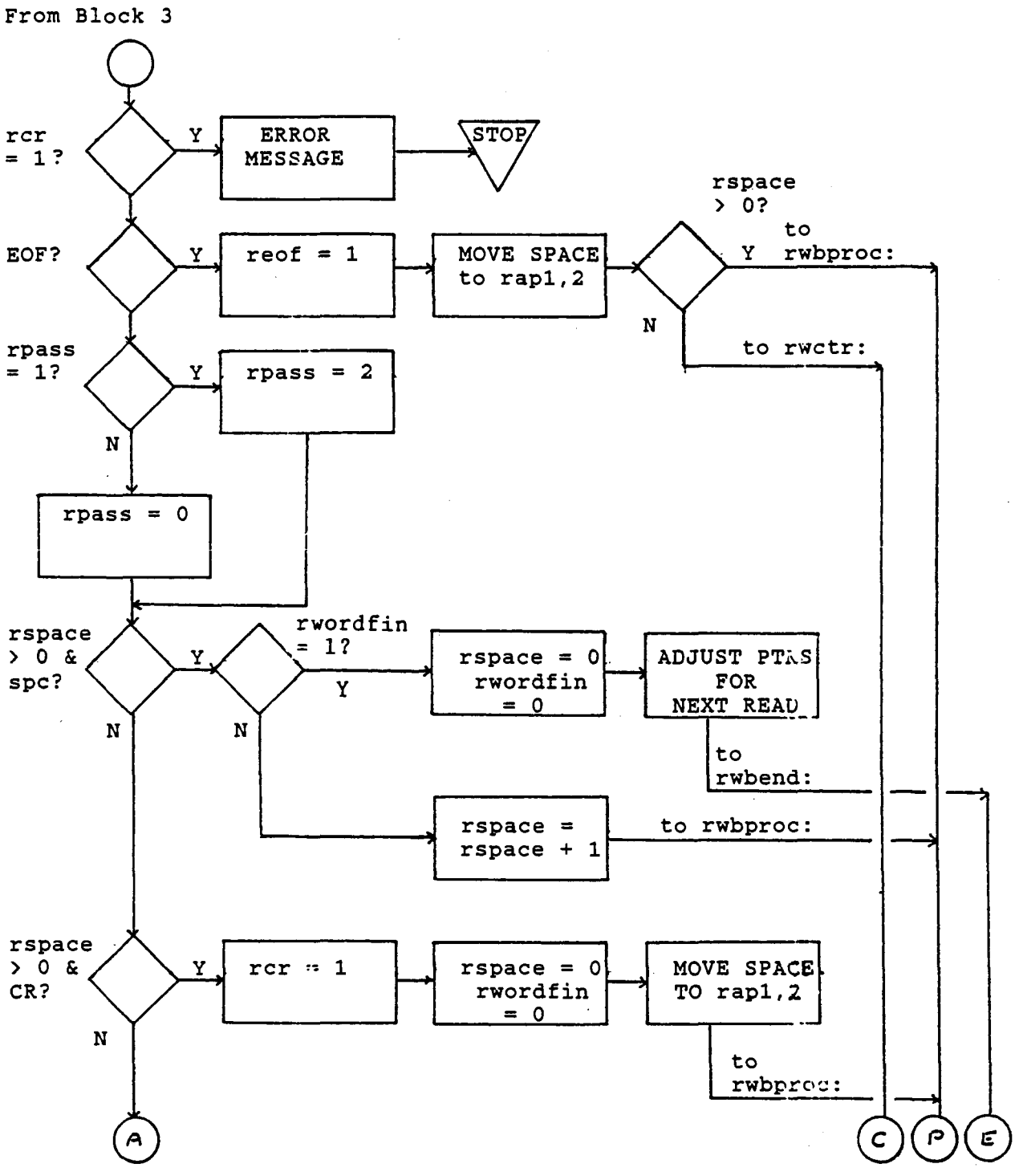
APPENDIX 6
Block Diagram for Program BITEX



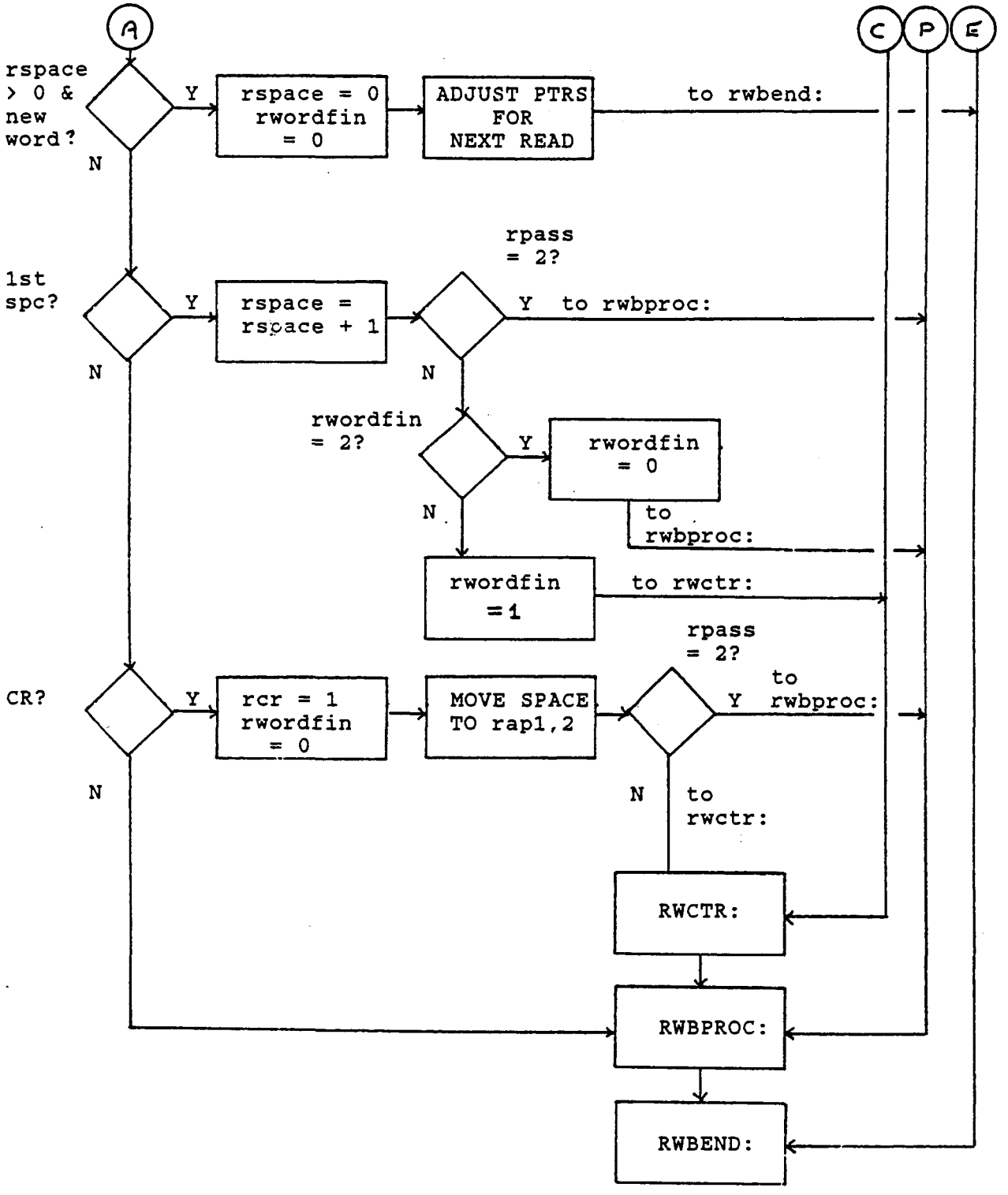
Block diagram for Program BITEX

This page is inserted to facilitate optional printing on both sides of the page, each appendix starting with an odd number.

APPENDIX 7
Flow Chart for Block 4 of Program BITEX



APPENDIX 7 (ctd)



APPENDIX 8
Program BITEX

```
80   'BLOCK 0. BITEXSCR Front End to Program BITEX
90   'Program BTXEXSCR.BAS is chained to BITEX.BAS and uses Include
95   'File BTXCOM.BI. It enables the user to open two input files,
100  'a Hebrew and Roman text file, and a BITEX output file and to
105  'assign values to variables used in the cloze procedure.
110  'Read in the contents of the BTXCOM.BI file:
112  '$INCLUDE: 'BTXCOM.BI'

204   ON ERROR GOTO errorhandler
208   IF filesw = 1 THEN GOTO filescreen    'set at end of BITEX
210   hebrew$ = "a:intext.heb"              'to repeat input files'
211   roman$ = "a:intext.rom"               'default names for
212   bitex$ = "a:outtext.btx"              'additional pass
214   OPEN "lpt1:" FOR OUTPUT AS #6

filescreen:
216   filesw = 0
220   CLS
222   LOCATE 1, 17:
223   PRINT "PROGRAM BITEX                Copyright, Curtis Rice, 1988"
230   LOCATE 3, 17
231   PRINT "This program makes an output file from two input files,"
234   LOCATE 4, 17
235   PRINT "a Hebrew and transliterated Roman text file. The output"
238   LOCATE 5, 17
239   PRINT "file displays alternating Hebrew and Roman lines, with"
240   LOCATE 6, 17
241   PRINT "each Roman word beneath its corresponding Hebrew word."
248   LOCATE 8, 17
249   PRINT "Enter input files to use and output file desired."
252   LOCATE 9, 17
253   PRINT "to make. Enter dsk drive & file name: eg A:GENESIS.HEB."
256   LOCATE 10, 17
257   PRINT "Be sure input files are on drive. <CR> enters file used"
260   LOCATE 11, 17
261   PRINT "on last pass if repeating run. BASIC 'redo from start'"
262   LOCATE 12, 17
263   PRINT "prompt means redo name you are on, not previous names."
274   LOCATE 14, 22
276   PRINT "1. Hebrew input file"
278   LOCATE 16, 22
280   PRINT "2. Roman  input file"
282   LOCATE 18, 22
284   PRINT "3. BITEX output file"

300   LOCATE 14, 43: INPUT ; filename$
304   IF filename$ = "" THEN filename$ = hebrew$
306   erreturn$ = hebrew$
```

APPENDIX 8 (ctd)

```
310  OPEN filename$ FOR INPUT AS #4 'to trap file not found error
320  CLOSE #4: OPEN filename$ FOR BINARY AS #1
321  hebrew$ = filename$              'hebrew$ identifies file in use
322  LOCATE 14, 60: PRINT "Opened "; filename$
325  PRINT #6, "#1 BINARY filename is "; filename$;

330  LOCATE 16, 43: INPUT ; filename$
332  IF filename$ = "" THEN filename$ = roman$
334  erreturn$ = roman$
335  OPEN filename$ FOR INPUT AS #5
345  CLOSE #5: OPEN filename$ FOR BINARY AS #2
346  roman$ = filename$
347  LOCATE 16, 60: PRINT "Opened "; filename$
350  PRINT #6, " #2 BINARY filename is "; filename$;

352  LOCATE 18, 43: INPUT ; filename$
353  IF filename$ = "" THEN filename$ = bitex$
355  LOCATE 21, 17: PRINT "                                              ";
356  PRINT "                                  ";
357  LOCATE 22, 17: PRINT "                                              ";
358  LOCATE 20, 50: PRINT "           (<CR> means Y)"
360  LOCATE 20, 12: PRINT "BITEX name "; filename$; " OK (Y/N)     "
361  LOCATE 20, 47: INPUT ; reply$
372  IF reply$ = "" THEN GOTO 380
374  IF reply$ = "y" OR reply$ = "Y" THEN GOTO 380
376  IF reply$ = "n" OR reply$ = "N" THEN GOTO 352
380  OPEN filename$ FOR BINARY AS #3
381  bitex$ = filename$
382  LOCATE 18, 60: PRINT "Opened "; filename$
384  LOCATE 25, 17: PRINT "       Press any key to continue       ";
385  DO
386  LOOP WHILE INKEY$ = ""
388  PRINT #6, " #3 binary filename is "; filename$
390  GOTO screencloze

errorhandler:
400 CONST filenotfound = 53
405 IF ERR = filenotfound THEN
415  LOCATE 20, 17
420  PRINT "File "; UCASE$(filename$); " not found. "
425  LOCATE 21, 17
426  PRINT "Enter drive, a:...b...or... unless file on drive in use"
440  LOCATE 22, 17: INPUT ; "Reenter filename: ", filename$
460  IF filename$ = "" THEN filename$ = erreturn$
470  RESUME
480 END IF
```

APPENDIX 8 (ctd)

```
screencloze:
502 CLS 0
504 LOCATE 1, 17:
505 PRINT "Program BITEX cloze procedure will delete as much Roman";
508 LOCATE 2, 17
509 PRINT "text as you wish. Set cloze to any n value from 0 - 7"
512 LOCATE 3, 17
513 PRINT "You can then set cumcloze variables to delete every nth"
516 LOCATE 4, 17
517 PRINT "word. If cloze = 7 & cumcloze = 1, words 1,8,15... are"
520 LOCATE 5, 17
521 PRINT "deleted. If in addition cumcloze4 = 4, words 1,4,8,11..."
524 LOCATE 6, 17
525 PRINT "are deleted. If all cumcloze variables = the #s in their"
528 LOCATE 7, 17
529 PRINT "names, all Roman text is deleted. If clozev = 1, all"
532 LOCATE 8, 17
533 PRINT "vowel pointing is deleted from the Hebrew text. "
548 LOCATE 10, 17
549 PRINT "Run BITEX as many times as desired to create a series of"
552 LOCATE 11, 17
553 PRINT "files to use as a reading tutorial. The BASIC 'redo from"
556 LOCATE 12, 17
557 PRINT "start' prompt means redo only the variable you are on."
560 LOCATE 14, 17
561 PRINT "Set cloze to 7      (Y/N):      (<CR> sets to 0)"
564 LOCATE 15, 17
565 PRINT "Set cumcloze1 to 1 (Y/N):      (<CR> or N unsets)"
568 LOCATE 16, 17
569 PRINT "Set cumcloze2 to 2 (Y/N):      (<CR> or N unsets)"
572 LOCATE 17, 17
573 PRINT "Set cumcloze3 to 3 (Y/N):      (<CR> or N unsets)"
576 LOCATE 18, 17
577 PRINT "Set cumcloze4 to 4 (Y/N):      (<CR> or N unsets)"
580 LOCATE 19, 17
581 PRINT "Set cumcloze5 to 5 (Y/N):      (<CR> or N unsets)"
584 LOCATE 20, 17
585 PRINT "Set cumcloze6 to 6 (Y/N):      (<CR> or N unsets)"
590 LOCATE 21, 17
591 PRINT "Set cumcloze7 to 7 (Y/N):      (<CR> or N unsets)"
594 LOCATE 22, 17
595 PRINT "Set clozev to 1    (Y/N):      (<CR> or N unsets)";

600 LOCATE 14, 43: INPUT ; cloze$
clozecheck:
610 IF cloze$ = "" THEN cloze = 0: GOTO cum1check
615 IF cloze$ = "y" THEN cloze = 7: GOTO cum1check
620 IF cloze$ = "Y" THEN cloze = 7: GOTO cum1check
630 IF cloze$ = "n" OR cloze$ = "N" THEN
635   LOCATE 14, 10
640   PRINT "cloze: enter value between 0 & 7 :    (<CR> sets to 0)"
```

APPENDIX 8 (ctd)

```
645   LOCATE 14, 45: INPUT ; cloze
650   IF cloze = 0 THEN GOTO cum1check
655   IF cloze < 0 OR cloze > 7 THEN
660    LOCATE 14, 10
665      PRINT "       cloze range 0 - 7, Reenter:     (<CR> sets to
0)"
670    LOCATE 14, 45: INPUT ; cloze
675    GOTO 650
680   END IF
682   GOTO cum1check
683 END IF
685 IF cloze$ <> "y" OR cloze$ <> "Y" OR cloze$ <> "n" OR  cloze$ <>
"N" THEN
690   LOCATE 14, 10
695    PRINT "cloze: enter Y(7) N(1-6) or <CR>(0):     (<CR> sets to
0)   "
696   LOCATE 14, 45: INPUT ; cloze$
697   GOTO clozecheck:
698 END IF
cum1check:
700 'PRINT #6, "cloze$ = "; cloze$; " cloze = "; cloze
704 LOCATE 14, 66: PRINT "set "; cloze
708 LOCATE 15, 43: INPUT ; cloze$
712 cn = 1: a% = 15: GOSUB cumroutine:
716 cumcloze1 = cumclozev
718 LOCATE 15, 66: PRINT "set "; cumcloze1
722 'PRINT #6, "cumcloze1 = "; cumcloze1;
724 LOCATE 16, 43: INPUT ; cloze$
726 cn = 2: a% = 16: GOSUB cumroutine
728 cumcloze2 = cumclozev
730 LOCATE 16, 66: PRINT "set "; cumcloze2
732 'PRINT #6, "cumcloze2 = "; cumcloze2;
736 LOCATE 17, 43: INPUT ; cloze$
740 cn = 3: a% = 17: GOSUB cumroutine
744 cumcloze3 = cumclozev
748 LOCATE 17, 66: PRINT "set "; cumcloze3
752 'PRINT #6, "cumcloze3 = "; cumcloze3;
756 LOCATE 18, 43: INPUT ; cloze$
760 cn = 4: a% = 18: GOSUB cumroutine
764 cumcloze4 = cumclozev
768 LOCATE 18, 66: PRINT "set "; cumcloze4
772 'PRINT #6, "cumcloze4 = "; cumcloze4;
776 LOCATE 19, 43: INPUT ; cloze$
780 cn = 5: a% = 19: GOSUB cumroutine
782 cumcloze5 = cumclozev
784 LOCATE 19, 66: PRINT "set "; cumcloze5
786 'PRINT #6, "cumcloze5 = "; cumcloze5;
788 LOCATE 20, 43: INPUT cloze$
790 cn = 6: a% = 20: GOSUB cumroutine
792 cumcloze6 = cumclozev
794 LOCATE 20, 66: PRINT "set "; cumcloze6
```

APPENDIX 8 (ctd)

```
796 'PRINT #6, "cumcloze6 = "; cumcloze6;
798 LOCATE 21, 43: INPUT cloze$
800 cn = 7: a% = 21: GOSUB cumroutine
802 cumcloze7 = cumclozev
804 LOCATE 21, 66: PRINT "set "; cumcloze7
806 'PRINT #6, "cumcloze7 = "; cumcloze7;

810 LOCATE 22, 43: INPUT cloze$
clozevrtne:
814 IF cloze$ = "" THEN clozev = 0: GOTO screendone
818 IF cloze$ = "y" OR cloze$ = "Y" THEN clozev = 1: GOTO screendone
822 IF cloze$ = "n" OR cloze$ = "N" THEN clozev = 0: GOTO screendone
826 IF cloze$ <> "y" OR cloze$ <> "Y" OR cloze$ <> "n" OR  cloze$ <>
"N" THEN
830   LOCATE 22, 15
834   PRINT "clozev: enter Y, N or <CR>:     (Y=1 N=0 <CR>=0)     "
838   LOCATE 22, 43: INPUT cloze$
842   GOTO clozevrtne
846 END IF

cumroutine:
900 IF cloze$ = "" THEN cumclozev = -1: GOTO exitcum
904  IF  cloze$  =  "y"  OR  cloze$  = "Y" THEN cumclozev = cn: GOTO
exitcum
908 IF cloze$ = "n" OR  cloze$  =  "N"  THEN  cumclozev  =  -1: GOTO
exitcum
920 IF  cloze$ <> "y" OR cloze$ <> "Y" OR cloze$ <> "n" OR cloze$ <>
"N" THEN
924   LOCATE a%, 10
928   PRINT "cumcloze";
930   LOCATE a%, 18
931   PRINT cn; " ="; cn; ":  Y, N or <CR>:     (<CR> or N unsets)"
932   LOCATE a%, 43: INPUT cloze$
936   GOTO cumroutine
940 END IF
exitcum:
944 RETURN

screendone:
950 LOCATE 22, 66: PRINT "set "; clozev
951 PRINT #6, "cloze settings: cloze: "; cloze; ", cumcloze: ";
952 PRINT #6, cumcloze1; " "; cumcloze2; " "; cumcloze3; " ";
953 PRINT #6, cumcloze4; " "; cumcloze5; " "; cumcloze6; " ";
962 PRINT #6, cumcloze7; " "; "clozev = "; clozev
990 CHAIN "BITEXTR2"
999 END
```

APPENDIX 8 (ctd)

```
1002 'Program BITEX.BAS is chained to BTXSCR.BAS
1004 'using Include file BTXCOM.BI
1006 'PROGRAM BITEX.BAS creates an output file from two input
1008 'files, a Hebrew text file, and a Roman text file which has
1010 'transliterated the Hebrew text. The output file displays
1012 'first a line of Hebrew and then a line of Roman with each
1014 'Roman word appearing beneath its corresponding Hebrew word.
1016 'A cloze procedure is included which progressively deletes
1018 'the Roman text until only the Hebrew is left. A block is
1022 'also included for direct transliteration of Hebrew to Roman
1024 'to supplement the MLS-CTIU tables or to act independently.

1026 'BLOCK 1. Initialise variables and move header information
1027 'to BITEX output file (bf).
1028 'Open files, declare variables and read first 400 bytes
1032 'from Hebrew File (hf,#1) to BITEX File (bf,#3)
1036 'Read in the contents of the BTXCOM.BI file
1040 ' $INCLUDE: 'BTXCOM.BI'

1100 PRINT #6, " Hebrew file = "; hebrew$      'to confirm
1150 PRINT #6, " Roman file = "; roman$       'opened files
1200 PRINT #6, " BITEX file = "; bitex$
1260 DIM ha(1 TO 200) AS STRING * 1           'Hebrew array
1270 DIM ra(1 TO 200) AS STRING * 1           'Roman array
1280 DIM ba(2, 200) AS STRING * 1             'BITEX array
1300 DIM rwb(1 TO 80) AS STRING * 1           'Roman word buffer
1320 DIM hwb(1 TO 80) AS STRING * 1           'Hebrew word buffer
1350 DIM b1 AS STRING * 1: DIM b2 AS STRING * 1 'Heb binary rd/wr
1370 DIM b3 AS STRING * 1: DIM b4 AS STRING * 1 'Rom binary rd/wr
1380 GOSUB nullarray:
1390 rwc = 0: hwc = 0:                        'Rom/Heb word counters
1392 crwc = 0: chwc = 0                       'cumulative word ctrs
1394 rap1 = -1: hap1 = -1                     'Rom/Heb array pointers
1395 rp1 = -1: hp1 = -1                       'BITEX array pointers
1396 rread = 1: rpass = 1: hpass = 1          'first pass flags
1399 heof = 0: reof = 0:                      'Heb EOF(1), Rom EOF(2)
1400 rwordfin = 0: hwordfin = 0               'end of proc'g wd flags
1401 rskipflag = 0: hskipflag = 0            '1st pass control flags
1410 twc = 1                                  'translit word ctr Block 6
1440 FOR n = 0 TO 399
1450 GET #1, , b1                             'read 400 bytes 1 by 1
1500   PUT #3, , b1
1600 NEXT n

1610 'Read 2-byte prt format instructions from hf to bf
1620 GET #1, , b1: GET #1, , b2          'read 2-bytes
1630 IF b1 = CHR$(59) THEN               'if 1st byte = ;
1640    PRINT #6, "Print format instructions in hf are:"
```

APPENDIX 8 (ctd)

```
1650    DO
1660      PUT #3, , b1: PUT #3, , b2
1670      PRINT #6, b1; b2;
1680      GET #1, , b1: GET #1, , b2
1690    LOOP UNTIL b1 = CHR$(128)          'continue until MLS CR
1700    PUT #3, , b1: PUT #3, , b2         'write CR and go to next
1730    PRINT #6, ASC(b1); ASC(b2)         'block to read Heb text
1750 END IF

1800 'Skip first 400 bytes, print format instructions & CTIU
1805 'language delimiter in rf in order to position read pointer
1810 'at start of text.
1820 GET #2, 401, b3: GET #2, , b4          'skip 1st 400 bytes
1830 IF b3 = CHR$(59) THEN                  'skip prt format instr
1840   GOSUB Readloop                       'starting with ";"
1850   GET #2, , b3: GET #2, , b4
1860 IF b3 = CHR$(47) THEN                  'skip lang delimiter
1870   GOSUB Readloop                       'starting with "/"
1880   GET #2, , b3: GET #2, , b4
1890 ELSE
1910   PRINT #6, "Language delimiter not found in File #2."
1920   STOP
1930 END IF
1940 IF b3 = CHR$(47) THEN                  'exits from 1870 above
1950   GOSUB Readloop
1960   GET #2, , b3: GET #2, , b4           'b3 & b4 have 1st char
1970 END IF                                 'or prog stops
1980 PRINT #6, "First two characters of rf in b3 & b4 =";
1985 PRINT #6, ASC(b3); ASC(b4)
1990 ra(1) = LEFT$(b3, 1)
1995 ra(2) = LEFT$(b4, 1)                   'puts 1st char into ra

readhf:
2100 'BLOCK 2. Move one line from Hebrew file to Hebrew array.
2150    FOR n = 1 TO 200 STEP 2
2170      n2 = n + 1
2172      IF EOF(1) THEN EXIT FOR
2200      GET #1, , b1: GET #1, , b2        'read 2-bytes
2300      ha(n) = LEFT$(b1, 1)              'byte 1 into ha 1...
2400      ha(n2) = LEFT$(b2, 1)             'byte 2 into ha 2...
2600      IF b1 = CHR$(128) AND b2 = CHR$(1) THEN
2620         EXIT FOR                       'exit at MLS CR
2630      ELSEIF EOF(1) THEN
2640         EXIT FOR
2650      END IF
2700    NEXT n
2750 'PRINT #6, "n = "; n; "n2 = "; n2
2760 'PRINT #6, "Ascii codes in ha up to above values of n are:"
2800    IF n >= 200 THEN
2820      PRINT #6, ""                      'error: need to check
2840      PRINT #6, "Hebrew line too long" 'text & shorten line
```

APPENDIX 8 (ctd)

```
2860       STOP
2900     END IF
2930 '    FOR n = 1 TO 200                    'print asc codes in ha
2931 '      IF ha(n) = "" THEN EXIT FOR
2932 '      PRINT #6, ASC(ha(n));
2934 '    NEXT n
2980 'PRINT #6, ""
2990 'PRINT #6, "End of Hebrew Array block"

readrf:
3000 'BLOCK 3. Move one line from Roman file to Roman array.
3010 IF rread = 1 THEN                       'for 1st read only b3
3014   a% = 3: rread = 0                     '& b4 moved to ra at
3016 ELSE a% = 1                             '1960 above.
3018 END IF
3100 FOR n = a% TO 200 STEP 2                'move char's into ra
3200    n2 = n + 1                           'up to & incl CR
3210    IF EOF(2) THEN EXIT FOR
3350    GET #2, , b3: GET #2, , b4
3400    ra(n) = LEFT$(b3, 1)
3450    ra(n2) = LEFT$(b4, 1)
3600    IF b3 = CHR$(128) AND b4 = CHR$(0) THEN
3610      EXIT FOR
3620    ELSEIF EOF(2) THEN
3624      EXIT FOR
3630    END IF
3700 NEXT n
3720 'PRINT #6, "n = "; n; "n2 = "; n2       'print final values
3750 IF n >= 200 THEN
3754    PRINT #6, "Roman line too long"
3760    STOP
3770 END IF
3780 'PRINT #6, "Ascii codes in ra for above values of n are:"
3800 'FOR n = 1 TO 200                       'print asc codes in ra
3820 '   IF ra(n) = "" THEN EXIT FOR
3840 '   PRINT #6, ASC(ra(n));
3860 'NEXT n
3920 'PRINT #6, "": PRINT #6, "End of Roman Array block"

rwb:
4000 'BLOCK 4. Move one word from Roman array to Roman word buffer.
4004 IF rcr = 1 THEN
4006   PRINT #6, "Rom ln ended before Heb: rcr, hcr = "; rcr; hcr
4008   STOP
4009 END IF
4010 rbc = 0: rcc = 0: rspace = 0:           'Roman byte, char &
4012 n = -1                                  'space ctrs set to 0
                                             'rpass is flag to mark 1st
rwbloop:
4014 n = n + 2: n2 = n + 1
4018 rap1 = rap1 + 2: rap2 = rap1 + 1
```

APPENDIX 8 (ctd)

```
4022 IF ra(rap1) = CHR$(0) AND ra(rap2) = CHR$(0) THEN
4026   reof = 1                                  'EOF flag - EOF needs
4030   ra(rap1) = CHR$(32)                       'space inserted as wd
4032   ra(rap2) = CHR$(0)                        'whose end it marks is
4034   IF rspace > 0 THEN                        'first not last in ma2
4038 '   PRINT #6, "rap 1,2 at spc/EOF(2), go rwbend: rspace =";
4039 '   PRINT #6, rspace
4042    GOTO rwbend:                             'keep byte ct consist
4046   ELSE                                      'for translit block
4050 '    PRINT #6, "rap1,2 at wd/EOF(2), goto rwctr: rspace =";
4052 '    PRINT #6, rspace
4054     GOTO rwctr:
4058   END IF
4062 END IF
4064 IF ra(rap1) = CHR$(32) AND rpass = 1 THEN rpass = 2
4065 ' First character space
4068 IF ra(rap1) = CHR$(128) AND rpass = 1 THEN rpass = 2
4069 ' First character carriage return (CR)
4072 IF rpass = 1 THEN rpass = 0
4076 IF rspace > 0 AND ra(rap1) = CHR$(32) THEN '2nd or more spc
4080   IF rwordfin = 1 THEN                      'real word processed
4084     rspace = 0: rwordfin = 2                'space after real word
4088     rap1 = rap1 - 2: rap2 = rap1 + 1 'adj for next read
4092     GOTO rwbend:                            'read Heb & proc to ma
4096   END IF
4100   rspace = rspace + 1                       'init space(s) & extra
4104   GOTO rwbproc:                             'space(s) before word
4108 ELSEIF  rspace > 0 AND ra(rap1) = CHR$(128) THEN
                                      'this is either CR or 1st
4112   rcr = 1                                   'Rom CRflag marks end
4116   ra(rap1) = CHR$(32)                       'line ending w spaces
4118   ra(rap2) = CHR$(0)                        'space substituted for
4120   rspace = 0: rwordfin = 0                  'CR for ma2
4124 '  PRINT #6, "rap1,2 on spc/CR, go rwbproc: rspace=";
4125 '  PRINT #6, rspace
4128   GOTO rwbproc:
4132 ELSEIF rspace > 0 THEN
4136 '  PRINT #6, "ra1,2 at new wd, go rwbend: rpass,rspace =";
4138 '  PRINT #6, rpass; rspace
4140   rspace = 0: rwordfin = 0          'ptrs on 1st char new wd
4144   rap1 = rap1 - 2: rap2 = rap1 + 1 'adj ptrs for next cycle
4148   GOTO rwbend:
4152 END IF
4156 IF ra(rap1) = CHR$(32) AND ra(rap2) = CHR$(0) THEN
4160   rspace = rspace + 1               'first rspace
4164   IF rpass = 2 THEN GOTO rwbproc:
4168   IF rwordfin = 2 THEN              'space after real word
4172     rwordfin = 0
4176     GOTO rwbproc:
4178   END IF                            'signals real wd procssd
4180   rwordfin = 1
```

APPENDIX 8 (ctd)

```
4182   GOTO rwctr:
4184 ELSEIF ra(rap1) = CHR$(128) AND ra(rap2) = CHR$(0) THEN
4186   rcr = 1: rwordfin = 0                'Rom CR flag
4188   ra(rap1) = CHR$(32): ra(rap2) = CHR$(0) 'init CR
4190   IF rpass = 2 THEN
4191 '    PRINT #6, "ra1,2 on init CR, goto rwbproc"
4192     GOTO rwbproc:
4193   ELSE
4194 '    PRINT #6, "ra1,2 on reg CR, goto rwctr, rspace =";
4195 '    PRINT #6, rspace
4196     GOTO rwctr:                        'CR marks end ra: adv
4197   END IF                               'rwc & replace w space
4198 END IF                                 'as this will be first
4199 GOTO rwbproc:                         'word in ma2
rwctr:
4200 rwc = rwc + 1: crwc = crwc + 1
4202 IF rwc = 1 THEN                       'allows for space not
4203   rcc = rcc + 1                       'proc to be replaced by
4204   GOTO rwbend                         'spec CR in ma2 as
4206 ELSE GOTO rwbproc:                    'this is last wd in ma2
4210 END IF
rwbproc:
4230 rwb(n) = ra(rap1): rwb(n2) = ra(rap2)
4240 rbc = rbc + 2                         '2 added to byte ctr
4250 rcc = rcc + 1                         '1 added to letter ctr
4260 IF rcr = 1 THEN
4264   GOTO rwbend:
4266 ELSEIF reof = 1 THEN
4267   GOTO rwbend:
4268 END IF
4300 v = ASC(ra(n2)) AND 128
4400    IF v = 128 THEN                    'if vowel bit "1" then
4420      n = n + 2: n2 = n + 1            'next 2 bytes read and
4424      rap1 = rap1 + 2                  'counted but character
4425      rap2 = rap1 + 1                  'count ignored
4430      rwb(n) = ra(rap1): rwb(n2) = ra(rap2)
4440      rbc = rbc + 2
4450      IF n < 79 THEN GOTO rwbloop:     'go to read next 2 bytes
4460    END IF
4600 IF n < 79 THEN GOTO rwbloop:          'when n=79, bytes 79/80
rwbend:                                    'have just been procssd
4610 'PRINT #6, "rpass:"; rpass; " rbc, rcc, rwc, crwc:";
4611 'PRINT #6, rbc; rcc; rwc; crwc
4612 'PRINT #6, "rcr, reof, rap1, n & contents of rwb:";
4613 'PRINT #6, rcr; reof; rap1, n
4620 'FOR n = 1 TO 80
4622 '   PRINT #6, ASC(rwb(n));
4624 'NEXT n
4626 'PRINT #6, "": PRINT #6, ""
```

APPENDIX 8 (ctd)

```
hwb:
5000 'BLOCK 5. Move one word from Hebrew array to Hebrew word
buffer.
5100 hbc = 0: hcc = 0: hspace = 0          'Heb byte, char &
5110 n = -1                                'space counters
hwbloop:
5120 n = n + 2: n2 = n + 1
5130 hap1 = hap1 + 2: hap2 = hap1 + 1      'hap1,2 governs ha
5140 IF ha(hap1) = CHR$(0) AND ha(hap2) = CHR$(0) THEN
5150   heof = 1                            'eof marker
5151   ha(hap1) = CHR$(128)                'CR needed to mark
5153   ha(hap2) = CHR$(1)                  'end of line
5152   IF hspace > 0 THEN
5160 '   PRINT #6, "hap1,2 on spc/EOF(1), go hwbend: hspace =";
5161 '   PRINT #6, hspace
5164    hwb(n) = ha(hap1):                 'to process CR without
5166    hwb(n2) = ha(hap2)                 'increasing byte count
5170    GOTO hwbend:                       'for byte consistency
5172   ELSE                                'in translit block
5173 '   PRINT #6, "hap1,2 on wd/EOF(1), go hwctr: hspace =";
5174 '   PRINT #6, hspace
5175    GOTO hwctr:
5180   END IF
5181 END IF
5182 IF ha(hap1) = CHR$(32) AND hpass = 1 THEN hpass = 2
5183      'First character space
5184 IF ha(hap1) = CHR$(128) AND hpass = 1 THEN hpass = 2
5185      'First character carriage return
5186 IF hpass = 1 THEN hpass = 0
5190 IF hspace > 0 AND ha(hap1) = CHR$(32) THEN
5191      'Second or more space
5192   IF hwordfin = 1 THEN
5194     hspace = 0: hwordfin = 2
5196     hap1 = hap1 - 2: hap2 = hap1 + 1
5198     GOTO hwbend:
5199   END IF
5200   hspace = hspace + 1                 'covers init & extra
5210   GOTO hwbproc:                       'space(s) before word
5220 ELSEIF hspace > 0 AND ha(hap1) = CHR$(128) THEN
5224 ' PRINT #6, "hap1,2 on CR after spcs; go hwbproc: hspace =";
5225 ' PRINT #6, hspace
5228   hspace = 0: hwordfin = 0            'CR follows spaces
5232   GOTO hwbproc:                       'after last word of line
5240 ELSEIF hspace > 0 THEN
5242 '  PRINT #6, "hap1,2 on new wd - to hwbend:hpass,hspace =";
5243 '  PRINT #6, hpass; hspace
5252   hspace = 0: hwordfin = 0            'must be start of new wd
5254   hap1 = hap1 - 2: hap2 = hap1 + 1    'adj ptrs for next cycle
5256   GOTO hwbend:
5258 END IF
```

APPENDIX 8 (ctd)

```
5260 IF ha(hap1) = CHR$(32) AND ha(hap2) = CHR$(1) THEN
5262   hspace = hspace + 1                'first space, space ct+1
5264   IF hpass = 2 THEN GOTO hwbproc:
5265   IF hwordfin = 2 THEN
5266     hwordfin = 0
6267     GOTO hwbproc:                    'avoid wd ct on init space
5268   END IF
5269   hwordfin = 1
5270   GOTO hwctr:                        '1st sp after wd goes w wd
5272 ELSEIF ha(hap1) = CHR$(128) AND ha(hap2) = CHR$(1) THEN
5273   hwordfin = 0
5274   IF hpass = 2 THEN
5275 '   PRINT #6, "ha1,2 on init CR, goto hwbproc"
5276     GOTO hwbproc:
5278   ELSE
5279 '    PRINT #6, "ha1,2 on reg CR, goto hwctr"
5280     GOTO hwctr:
5281   END IF                             'CR ends word & ha
5282 END IF
5283 GOTO hwbproc:
hwctr:
5284 hwc = hwc + 1: chwc = chwc + 1
hwbproc:
5330 hwb(n) = ha(hap1): hwb(n2) = ha(hap2)
5334 hbc = hbc + 2: hcc = hcc + 1
5343 IF ha(hap1) = CHR$(128) AND ha(hap2) = CHR$(1) THEN
5346   GOTO hwbend:
5347 END IF
5348 v = (ASC(ha(hap2)) AND 128)         'test for vowel bit "1" in
5350 IF v = 128 AND clozev = 1 THEN      'byte 2 (makes asc 128)
5351     hwb(n2) = CHR$(1)
5352     hap1 = hap1 + 2: hap2 = hap1 + 1
5354     IF n < 79 THEN GOTO hwbloop:
5355     GOTO hwbend:
5356 END IF
5358 IF v = 128 AND clozev = 0 THEN
5364   hap1 = hap1 + 2: hap2 = hap1 + 1     'process vowel byte
5368   n = n + 2: n2 = n + 1                'but dont adv char ctr
5370   hwb(n) = ha(hap1): hwb(n2) = ha(hap2)
5380   hbc = hbc + 2
5388   IF n < 79 THEN
5399    GOTO hwbloop:
5390   ELSE
5392    GOTO hwbend:
5394   END IF
5400 END IF
5404 IF n < 79 THEN GOTO hwbloop:
hwbend:
5450 IF ha(hap1) = CHR$(128) AND ha(hap2) = CHR$(1) THEN hcr = 1
5451    'CR flag
```

APPENDIX 8 (ctd)

```
5480 IF hwc <> rwc THEN
5482   PRINT #6, "": PRINT #6, "rwc ="; rwc; ";"; "hwc ="; hwc
5484   PRINT #6, "Diff between # Heb & Rom wds proc: check text"
5488 END IF
5489 ' PRINT #6, "hpass:"; hpass; "hbc, hcc, hwc, chwc:";
5490 ' PRINT #6, hbc; hcc; hwc; chwc
5491 ' PRINT #6, "hcr, heof, hap1, n & contents of hwb:";
5492 ' PRINT #6, hcr; heof; hap1; n
5493 ' FOR n = 1 TO 80
5494 '  PRINT #6, ASC(hwb(n));
5496 ' NEXT n
5498 ' PRINT #6, "": PRINT #6, ""
5499 IF hwc <> rwc THEN STOP

6000 'BLOCK 6. Refine transliteration.

6100 'Short qametz in final closed unaccented syllable.
6101 'Check for stress marker in the word. If present,
6102 'check for qametz in final syllable (unaccented).
6104 'If present, make qametz short.
6105 IF twc = rwc THEN                        'to avoid processing
6107   twc = twc + 1                         'non-words, e.g.leading
6109   GOTO transproc                        'spaces, blank line CRs
6110 ELSE GOTO rwbtoba2
transproc:
6112 IF rwc = 1 THEN                         'give consistent byte
6114   rbc = rbc + 2                         'count to word 1
6116 END IF
6120 FOR n = 1 TO (rbc - 7) STEP 2                  'rbc-7 last
6125   IF rwb(n) = CHR$(126) THEN GOTO checklast:   'possible pos
6130 NEXT n                                        'for secondary
6132 GOTO exitqam:                                 'stress marker
checklast:
6140   v = (ASC(hwb(hbc - 4)) AND 128) 'indic if last pos vowel
6145   IF v = 128 THEN GOTO exitqam:   'syllable must be open
6150   IF rwb(rbc - 3) = CHR$(104) THEN GOTO exitqam:  'final he
6155   IF rwb(rbc - 3) = CHR$(97) THEN GOTO exitqam:   'fin aleph
6160   IF rwb(rbc - 3) = CHR$(65) THEN GOTO exitqam:   'fin aleph
6170   IF rwb(rbc - 5) = CHR$(19) THEN        'penult char qametz
6174      rwb(rbc - 5) = CHR$(137)
6178      GOTO exitqam:
6180   END IF
exitqam:
6191 'PRINT #6, "n = "; N; "v = "; v; " rwb(rbc-5) = ";
6192 'PRINT #6, ASC(rwb(rbc - 5));
6193 'PRINT #6, " rwb(rbc-3) = "; ASC(rwb(rbc - 3))
6194 'PRINT #6, ""
6195 IF rwc = 1 THEN                          'to restore byte count
6197   rbc = rbc - 2                         'to word 1
6198 END IF
```

APPENDIX 8 (CTD)

```
rwbtoba2:
7000 'BLOCK 7. Move one word from Roman word buffer to BITEX2 array
7002 'and align.
7006 IF rpass = 2 THEN GOTO ba2spec:
7010 IF rwc = 1 AND rskipflag = 0 THEN
7011   GOTO ba2spec:
7012 ELSE GOTO rlineup:
7013 END IF
ba2spec:
7015 'Roman text must be enclosed between Heb hardspaces (or
7016 'other Heb characters) & end with Heb CR to align with Heb
7017 'text in bf
7020   ba(2, 199) = CHR$(128): ba(2, 200) = CHR$(1)
7030   ba(2, 197) = CHR$(252): ba(2, 198) = CHR$(1)
7040   rp1 = 197: rp2 = 198
7045   IF rpass = 2 THEN
7047     rpass = 0
7049     rskipflag = 1     'to avoid duplicating first pass
7050   END IF              'routine at 7010 above
rlineup:
7100 IF hcc > rcc THEN
7110   FOR n = 1 TO ((hcc - rcc) * 2) STEP 2 'move spaces to Rom
7120     rp1 = rp1 - 2: rp2 = rp1 + 1        'line - the exceptn
7130     ba(2, rp1) = CHR$(32)
7132     ba(2, rp2) = CHR$(0)                'rp = Rom ba2 ptr
7140   NEXT n
7144 END IF

8000 'BLOCK 8. Cloze procedure.
8715 FOR i = 1 TO 7
8152    IF crwc = cumcloze1 THEN GOTO clozeloop: 'initial cloze
8153    IF crwc = cumcloze2 THEN GOTO clozeloop: '& cumcloze
8154    IF crwc = cumcloze3 THEN GOTO clozeloop: 'set at start
8155    IF crwc = cumcloze4 THEN GOTO clozeloop: '& updated
8156    IF crwc = cumcloze5 THEN GOTO clozeloop: 'below 8171-7
8157    IF crwc = cumcloze6 THEN GOTO clozeloop:
8158    IF crwc = cumcloze7 THEN GOTO clozeloop:
8159    GOTO nexti
clozeloop:
8160      rp1 = rp1 - 2: rp2 = rp1 + 1
8161      IF rwc = 1 THEN
8162        ba(2, rp1) = CHR$(46)          'makes up for already
8163        ba(2, rp2) = CHR$(0)           'inserted Heb hd spc
8164      ELSE
8165        ba(2, rp1) = CHR$(32): ba(2, rp2) = CHR$(0)
8166      END IF
8167      FOR n = 0 TO (rbc - 3) STEP 2
8168         rp1 = rp1 - 2: rp2 = rp1 + 1
8169         ba(2, rp1) = CHR$(46): ba(2, rp2) = CHR$(0)
8170      NEXT n
```

APPENDIX 8 (ctd)

```
8171       IF crwc = cumcloze1 THEN cumcloze1 = cumcloze1 + cloze
8172       IF crwc = cumcloze2 THEN cumcloze2 = cumcloze2 + cloze
8173       IF crwc = cumcloze3 THEN cumcloze3 = cumcloze3 + cloze
8174       IF crwc = cumcloze4 THEN cumcloze4 = cumcloze4 + cloze
8175       IF crwc = cumcloze5 THEN cumcloze5 = cumcloze5 + cloze
8176       IF crwc = cumcloze6 THEN cumcloze6 = cumcloze6 + cloze
8177       IF crwc = cumcloze7 THEN cumcloze7 = cumcloze7 + cloze
8178       clozeflag = 1
nexti:
8180 NEXT i
8181 IF clozeflag = 1 THEN
8182   clozeflag = 0
8184   GOTO hwbtoba1:
8186 ELSE
8187   GOTO regloop:
8188 END IF
regloop:
7200 'Concluding BLOCK 7.
7230 FOR n = 0 TO (rbc - 1) STEP 2
7235   n2 = n + 1
7237   rp1 = rp1 - 2: rp2 = rp1 + 1
7240    ba(2, rp1) = rwb(rbc - n2)        'move Rom words rev seq

7245   ba(2, rp2) = rwb(rbc - n)
7250 NEXT n

hwbtoba1:
9500 'BLOCK 9. Move one word from Hebrew word buffer to BITEX1 array
9501 'and align.
9502 IF hpass = 2 THEN GOTO ba1spec:
9504 IF rwc = 1 AND hskipflag = 0 THEN
9506   GOTO ba1spec:
9508 ELSE GOTO hlineup:
9509 END IF
ba1spec:
9510 ba(1, 1) = CHR$(124)          'Heb delimiter to ctl rt margin
9511 ba(1, 2) = CHR$(1)
9512 hp1 = 1: hp2 = 2              'set ptrs for FOR loop
9515 IF hpass = 2 THEN
9516   hpass = 0
9517   hskipflag = 1
9518 END IF
hlineup:
9520 IF rcc > hcc THEN
9570   FOR n = 1 TO ((rcc - hcc) * 2) STEP 2 'move spaces to Heb
9580     hp1 = hp1 + 2: hp2 = hp1 + 1        'line - the rule
9590     ba(1, hp1) = CHR$(32): ba(1, hp2) = CHR$(1)
9600   NEXT n
9670 END IF
9680 IF heof = 1 THEN hbc = hbc + 2 'CR inserted without byte ct
```

APPENDIX 8 (ctd)

```
9700 FOR n = 1 TO hbc STEP 2  'when eof at hwbloop: to keep byte
9710   n2 = n + 1             'ct consist for translt blck. Byte
9720   hp1 = hp1 + 2: hp2 = hp1 + 1 'ct now increasd to proc CR
9730   ba(1, hp1) = hwb(hbc - (hbc - n))      'move Heb words in
9740   ba(1, hp2) = hwb(hbc - (hbc - n2))     'regular sequence
9750 NEXT n
9752 IF hcr = 1 THEN
9758   GOTO alignend:
9760 ELSEIF heof = 1 THEN                     'eof marker
9762   GOTO alignend:
9764 ELSE
9770   FOR n = 1 TO 80
9772     rwb(n) = CHR$(0)
9774     hwb(n) = CHR$(0)              'set word buffers to null
9776   NEXT n
9780   GOTO rwb:
9790 END IF
alignend:
9800 'PRINT #6, "": PRINT #6, "Final value of ptrs & counters:"
9810 'PRINT #6, "hap1,2 ="; hap1; hap2; "rap1,2 ="; rap1; rap2
9820 'PRINT #6, "hbc, rbc ="; hbc; rbc; "hcc, rcc ="; hcc; rcc;
9830 'PRINT #6, "hwc, chwc ="; hwc; chwc; "
9831 'PRINT #6, "rwc, crwc ="; rwc; crwc
9840 rp1 = rp1 - 2: rp2 = rp1 + 1        'move Heb delimiter as
9850 ba(2, rp1) = CHR$(124)              'first char in Rom line
9851 ba(2, rp2) = CHR$(1)
9860 'PRINT #6, "hp1, hp2 ="; hp1; hp2; "rp1, rp2 ="; rp1; rp2
9870 'PRINT #6, "Final contents of Heb ba1 are:"
9880 'FOR n = 1 TO 200
9890 ' PRINT #6, ASC(ba(1, n));
9900 'NEXT n
9910 'PRINT #6, ""
9920 'PRINT #6, "Final contents of Rom ba2 are:"
9930 'FOR n = 1 TO 200
9940 '  PRINT #6, ASC(ba(2, n));
9950 'NEXT n
9960 'PRINT #6, "": PRINT #6, "End of align block"

10000 'BLOCK 10. Move BITEX array to BITEX output file.
10100 FOR n = 1 TO 200 STEP 2            'write ba1 to CR to bf
10110   n2 = n + 1
10120   PUT #3, , ba(1, n): PUT #3, , ba(1, n2)
10140   IF ba(1, n) = CHR$(128) AND ba(1, n2) = CHR$(1) THEN
10142    GOTO writeba2:
10144   END IF
10150 NEXT n
10160 IF n2 >= 200 THEN
10161   PRINT #6, "ba1 exceeded length, n2 ="; n2:
10162   STOP
10163 END IF
```

APPENDIX 8 (ctd)

```
writeba2:
10180 'PRINT #6, "n, n2 for ba1 ="; n; n2
10190 hhsflag = 0                              'Heb hard space flag
10200 FOR n = 1 TO 200 STEP 2
10210   n2 = n + 1
10214   IF hhsflag = 1 THEN
10216     GOTO ba2proc:
10220   ELSEIF ba(2, n) = CHR$(0) AND ba(2, n2) = CHR$(0) THEN
10222     GOTO nextni:
10224   ELSEIF ba(2, n) = CHR$(124) AND ba(2, n2) = CHR$(1) THEN
10226     hhsflag = 1                  'denotes Heb delimiter found
10228     GOTO ba2proc:
10230   ELSE
10232     PRINT #6, "Heb delimtr not found as first char of ba2"
10236     STOP
10238   END IF
ba2proc:
10250   PUT #3, , ba(2, n)
10255   PUT #3, , ba(2, n2)                'write ba2 to CR to bf
10260   IF ba(2, n) = CHR$(128) AND ba(2, n2) = CHR$(1) THEN
10261     GOTO bawrend:
10262   END IF
nextni:
10270 NEXT n
bawrend:
10300 'PRINT #6, "": PRINT #6, "n,n2 ="; n; n2; ",";
10301 'PRINT #6, "hhsflag ="; hhsflag
10310 'PRINT #6, "End of write block"
10320 IF EOF(1) THEN
10330   IF EOF(2) THEN
10350     PRINT #6, "Run completed"
10352     GOTO closing
10404   ELSE
10405     PRINT #6, "EOF(1) and EOF(2) not coincident"
10406     STOP: GOTO closing
10407   END IF
10410 ELSEIF EOF(2) THEN
10411   PRINT #6, "EOF(2) and EOF(1) not coincident"
10412   STOP: GOTO closing
10414 END IF
10415 hwc = 0: rwc = 0: twc = 1                'reset counters, flags
10416 hpass = 1: rpass = 1                     '& pointers for next
10418 rap1 = -1: hap1 = -1: hp1 = -1          'line
10419 rskipflag = 0: hskipflag = 0
10420 rwordfin = 0: hwordfin = 0
10421 rcr = 0: hcr = 0
10422 GOSUB nullarray
10430 GOTO readhf:
```

APPENDIX 8 (ctd)

```
nullarray:
19100 FOR n = 1 TO 200
19110   ha(n) = CHR$(0)
19120   ra(n) = CHR$(0)
19130   ba(1, n) = CHR$(0)
19140 NEXT n
19160 FOR n = 1 TO 80
19170   rwb(n) = CHR$(0)
19180   hwb(n) = CHR$(0)
19190 NEXT n
19200 RETURN

Readloop:
20100 FOR n = 1 TO 50
20200  GET #2, , b3: GET #2, , b4
20300  IF b3 = CHR$(128) THEN EXIT FOR
20400  IF n = 50 THEN
20500   PRINT #6, "Start of file #2 text not found in Readloop"
20520   PRINT #6, "b3 asc ="; ASC(b3); "b4 asc ="; ASC(b4)
20540   STOP
20560  END IF
20600 NEXT n
20700 RETURN

closing:
29000 CLS
29100 LOCATE 2, 10: PRINT "Program run complete."
29110 LOCATE 4, 10: PRINT "Hebrew file = "; hebrew$
29120 LOCATE 5, 10: PRINT "Roman file = "; roman$
29130 LOCATE 6, 10: PRINT "Bitex file = "; bitex$
29140 LOCATE 8, 10
29141 PRINT "Do you want another run with the above input files"
29150 LOCATE 9, 10
29151 PRINT "but with a new BITEX output file and cloze settings?"
29160 LOCATE 11, 10
29161 PRINT "Answering Y returns you to the file creation and cloze"
29170 LOCATE 12, 10
29171 PRINT "screens for another run. At the file creation screen"
29180 LOCATE 13, 10
29181 PRINT "pressing <CR> returns the above files."
29184 LOCATE 15, 10
29185 PRINT "Answering N will end the program."
29190 LOCATE 17, 10: INPUT ; "Another program run (Y/N):", answ$
      SELECT CASE answ$
        CASE "y": filesw = 1: CHAIN "BTXSCR2"
        CASE "Y": filesw = 1: CHAIN "BTXSCR2"
        CASE "n": PRINT #6, "END": END
        CASE "N": PRINT #6, "END": END
        CASE ELSE: PRINT "Response out of range"
        END SELECT
```

APPENDIX 9(a)

Genesis Ch 1, vs 1 & 2: Hebrew text

in Hebrew 9-pt proportionally spaced ancient alphabet

בְּרֵאשִׁית

בְּרֵאשִׁית בָּרָא אֱלֹהִים
אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃
וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם
וְרוּחַ אֱלֹהִים מְרַחֶפֶת עַל־פְּנֵי הַמָּיִם׃
וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר
וַיְהִי־אוֹר׃
וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב
וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ׃
וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה
וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד׃

APPENDIX 9(b)

Genesis Ch1, vs 1 & 2: Transliteration
in Roman 9-pt proportionally spaced "classical" alphabet (top)
and "classical alternative" alphabet (bottom)

/h/
b^{e}rē'šîṯ

b^{e}rē'šîṯ bārā' 'ĕlōhîm
'eṯ hašāma·yim w^{e}'eṯ hā'ā·reṣ:
w^{e}hā'ā·reṣ hây^{e}ṯāh ṯōhû wāḇōhû w^{e}ḥō·šeḵ 'al-p^{e}nê ṯ^{e}hôm
w^{e}rû·aḥ 'ĕlōhîm m^{e}raḥe·p̄eṯ 'al-p^{e}nê hamâ·im:
wayō·'mer 'ĕlōhîm y^{e}hî 'ôr
wayhî-'ôr:
wayar' 'ĕlōhîm 'eṯ-hā'ôr kî-ṭôḇ
wayaḇḏēl 'ĕlōhîm bên hā'ôr ûḇên haḥō·šeḵ:
wayiqrā' 'ĕlōhîm lā'ôr yôm w^{e}laḥō·šeḵ qārā' lā·ylāh
wayhî-'e·reḇ wayhî-ḇō·qer yôm 'eḥāḏ:

/h/
b^{e}rē'shîth

b^{e}rē'shîth bārā' 'ĕlōhîm
'eth hashāma·yim w^{e}'eth hā'ā·retz:
w^{e}hā'ā·retz hây^{e}thāh thōhû wābhōhû w^{e}ḥō·shekh 'al-p^{e}nê th^{e}hôm
w^{e}rû·aḥ 'ĕlōhîm m^{e}raḥe·pheth 'al-p^{e}nê hamâ·im:
wayō·'mer 'ĕlōhîm y^{e}hî 'ôr
wayhî-'ôr:
wayar' 'ĕlōhîm 'eth-hā'ôr kî-ṭôbh
wayabhdhēl 'ĕlōhîm bên hā'ôr ûbhên haḥō·shekh:
wayiqrā' 'ĕlōhîm lā'ôr yôm w^{e}laḥō·shekh qārā' lā·ylāh
wayhî-'e·rebh wayhî-bhō·qer yôm 'eḥādh:

APPENDIX 9(c)

Genesis Ch1, vs 1 & 2: Transliteration
in Roman 9-pt proportionally spaced "modern" alphabet (top)
and "simplified modern" alphabet (bottom)

/h/
b^{e}rē'shît

b^{e}rē'shît bārā' 'ĕlōhîm
'et hashāma·yim v^{e}'et hā'ā·retz:
v^{e}hā'ā·retz hây^{e}tāh tōhû vābhōhû v^{e}ḥō·shekh 'al-p^{e}nê t^{e}hôm
v^{e}rû·aḥ 'ĕlōhîm m^{e}raḥe·phet 'al-p^{e}nê hamâ·im:
vayō·'mer 'ĕlōhîm y^{e}hî 'ôr
vayhî-'ôr:
vayar' 'ĕlōhîm 'et-hā'ôr kî-ṭôbh
vayabhdēl 'ĕlōhîm bên hā'ôr ûbhên haḥō·shekh:
vayiqrā' 'ĕlōhîm lā'ôr yôm v^{e}laḥō·shekh qārā' lā·ylāh
vayhî-'e·rebh vayhî-bhō·qer yôm 'eḥād:

/h/ '
b^{e}re'shit

b^{e}re'shit bara' 'elohim
'et hashama·yim v^{e}'et ha'a·retz:
v^{e}ha'a·retz hay^{e}tah tohu vavohu v^{e}ho·shech 'al-p^{e}ne t^{e}hom
v^{e}ru·aḥ 'elohim m^{e}raḥe·fet 'al-p^{e}ne hama·im:
vayo·'mer 'elohim y^{e}hi 'or
vayhi-'or:
vayar' 'elohim 'et-ha'or ki-tov
vayavdel 'elohim ben ha'or uven haho·shech:
vayikra' 'elohim la'or yom v^{e}laho·shech kara' la·ylah
vayhi-'e·rev vayhi-vo·ker yom 'eḥad:

APPENDIX 9(d)

Genesis Ch 1, vs 1 & 2: BITEX output
in Hebrew 9-pt fixed space ancient alphabet
and Roman 9-pt fixed space "classical" alphabet

```
                                                    בְּרֵאשִׁית
                                                    berē'šîṯ

                                                    בְּרֵאשִׁית בָּרָא אֱלֹהִים
                                                    'ĕlōhîm bārā' berē'šîṯ
                                                    אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ:
                                                    hā'ā·reṣ: we'eṯ hašāma·yim 'eṯ
                                    וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם
ṯehôm 'al-penê weḥō·šeḵ wāḇōhû ṯōhû hâyeṯāh wehā'ā·reṣ
                                    וְרוּחַ אֱלֹהִים מְרַחֶפֶת עַל־פְּנֵי הַמָּיִם:
                                    hamâ·im: 'al-penê meraḥe·peṯ 'ĕlōhîm werû·aḥ
                                                    וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר
                                                    'ôr yehî 'ĕlōhîm wayō·'mer
                                                    וַיְהִי־אוֹר:
                                                    wayhî-'ôr:
                                                    וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב
                                                    kî-ṭôḇ 'eṯ-hā'ôr 'ĕlōhîm wayar'
                                    וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ:
                                    haḥō·šeḵ: ûḇên hā'ôr bên 'ĕlōhîm wayaḇḏēl
                                    וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה
lā·ylāh qārā' welaḥō·šeḵ yôm lā'ôr 'ĕlōhîm wayiqrā'
                                    וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד:
                                    'eḥāḏ: yôm wayhî-ḇō·qer wayhî-'e·reḇ
```

APPENDIX 9(e)

Genesis Ch 1, vs 1 & 2: BITEX output
in Hebrew 9-pt fixed space ancient alphabet
and Roman 9-pt fixed space "classical alternative" alphabet

בְּרֵאשִׁית
berē'shîth

בְּרֵאשִׁית בָּרָא אֱלֹהִים
'ĕlōhîm bārā' berē'shîth

אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃
hā'ā·retz: we'eth hashāma·yim 'eth

וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם
thehôm 'al-penê weḥō·shekh wābhōhû thōhû hâyethāh wehā'ā·retz

וְרוּחַ אֱלֹהִים מְרַחֶפֶת עַל־פְּנֵי הַמָּיִם׃
hamâ·im: 'al-penê meraḥe·pheth 'ĕlōhîm werû·aḥ

וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר
'ôr yehî 'ĕlōhîm wayō·'mer

וַיְהִי־אוֹר׃
wayhî-'ôr:

וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב
kî-ṭôbh 'eth-hā'ôr 'ĕlōhîm wayar'

וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ׃
haḥō·shekh: ûbhên hā'ôr bên 'ĕlōhîm wayabhdhēl

וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה
lā·ylāh qārā' welaḥō·shekh yôm lā'ôr 'ĕlōhîm wayiqrā'

וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד׃
'eḥādh: yôm wayhî-bhō·qer wayhî-'e·rebh

APPENDIX 9(f)

Genesis Ch1, vs 1 & 2: BITEX output
in Hebrew 9-pt proportionally spaced ancient alphabet
and Roman 9-pt proportionally spaced "modern" alphabet

בְּרֵאשִׁית
bᵉrē'shît

בְּרֵאשִׁית בָּרָא אֱלֹהִים
'ĕlōhîm bārā' bᵉrē'shît
אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃
hā'ā·retz: vᵉ'et hashāma·yim 'et
וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם
tᵉhôm 'al-pᵉnê vᵉḥō·shekh vābhōhû tōhû hâyᵉtāh vᵉhā'ā·retz
וְרוּחַ אֱלֹהִים מְרַחֶפֶת עַל־פְּנֵי הַמָּיִם׃
hamâ·im: 'al-pᵉnê mᵉraḥ·phet 'ĕlōhîm vᵉrû·aḥ
וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר
'ôr yᵉhî 'ĕlōhîm vayō·'mer
וַיְהִי־אוֹר׃
vayhî-'ôr:
וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב
kî-ṭôbh 'et-hā'ôr 'ĕlōhîm vayar'
וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ׃
haḥō·shekh: ûbhên hā'ôr bên 'ĕlōhîm vayabhdēl
וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה
lā·ylāh qārā' vᵉlaḥō·shekh yôm lā'ôr 'ĕlōhîm vayiqrā'
וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד׃
'eḥād: yôm vayhî-bhō·qer vayhî-'ᵉ·rebh

APPENDIX 9(g)

Genesis Ch 1, vs 1 & 2: BITEX output
in Hebrew 9-pt proportionally spaced ancient alphabet
and Roman 9-pt proportionally spaced "simplified modern" alphabet

בְּרֵאשִׁית
b^{e}re'shit

בְּרֵאשִׁית בָּרָא אֱלֹהִים
'elohim bara' b^{e}re'shit
אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃
ha'a·retz: v^{e}'et hashama·yim 'et
וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם
t^{e}hom 'al-p^{e}ne v^{e}ḥo·shech vavohu tohu hay^{e}tah v^{e}ha'a·retz
וְרוּחַ אֱלֹהִים מְרַחֶפֶת עַל־פְּנֵי הַמָּיִם׃
hama·im: 'al-p^{e}ne m^{e}raḥe·fet 'elohim v^{e}ru·aḥ
וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר
'or y^{e}hi 'elohim vayo·'mer
וַיְהִי־אוֹר׃
vayhi-'or:
וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב
ki-tov 'et-ha'or 'elohim vayar'
וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ׃
haḥo·shech: uven ha'or ben 'elohim vayavdel
וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה
la·ylah kara' v^{e}laḥo·shech yom la'or 'elohim vayikra'
וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד׃
'eḥad: yom vayhi-vo·ker vayhi-'e·rev

APPENDIX 9(h)

Genesis Ch 1, vs 1 & 2: BITEX stage-one cloze output
in Hebrew "screen size" fixed spaced modern alphabet
and Roman modified "screen size" fixed spaced "simplified modern" alphabet

```
                                                                  |  בְּרֵאשִׁית
                                                                  |.........
                                                                  |
                                                                  |
                                                                  |
                                                  |  בְּרֵאשִׁית  בָּרָא  אֱלֹהִים
                                                  |'elohim bara' b^re'shit
                                          |  אֵת  הַשָּׁמַיִם  וְאֵת  הָאָרֶץ׃
                                          |..... v^e'et hashama·yim 'et
|  וְהָאָרֶץ  הָיְתָה  תֹהוּ  וָבֹהוּ  וְחֹשֶׁךְ  עַל־פְּנֵי  תְהוֹם
|..... `al-p^ne v^eḥo·shech vavohu tohu hay^tah v^eha'a·retz
                                  |  וְרוּחַ  אֱלֹהִים  מְרַחֶפֶת  עַל־פְּנֵי  הַמָּיִם׃
                                  |hama·im׃ `al-p^ne m^eraḥe·fet 'elohim v^eru·aḥ
                                                  |  וַיֹּאמֶר  אֱלֹהִים  יְהִי אוֹר
                                                  |'or y^hi ..... vayo·'mer
                                                                  |  וַיְהִי־אוֹר׃
                                                                  |vayhi-'or׃
                                                  |  וַיַּרְא  אֱלֹהִים  אֶת־הָאוֹר כִּי־טוֹב
                                                  |..... 'et-ha'or 'elohim vayar'
                                          |  וַיַּבְדֵּל  אֱלֹהִים בֵּין הָאוֹר וּבֵין  הַחֹשֶׁךְ׃
                                          |haḥo·shech׃ uven ha'or ben 'elohim vayavdel
|  וַיִּקְרָא  אֱלֹהִים  לָאוֹר יוֹם  וְלַחֹשֶׁךְ  קָרָא  לָיְלָה
|la·ylah kara' v^elaḥo·shech yom la'or 'elohim .....
                                  |  וַיְהִי־עֶרֶב  וַיְהִי־בֹקֶר יוֹם  אֶחָד׃
                                  |'eḥad׃ yom vayhi-vo·ker .....
```

APPENDIX 9(i)

Genesis Ch 1, vs 1 & 2: BITEX stage-two cloze output
in Hebrew "screen size" fixed spaced modern alphabet
and Roman modified "screen size" fixed spaced "simplified modern" alphabet

```
|    בְּרֵאשִׁית
|.........
|
|
|
|    בְּרֵאשִׁית   בָּרָא   אֱלֹהִים
|..... bara' b°re'shit
| אֵת      הַשָּׁמַיִם   וְאֵת      הָאָרֶץ׃
|..... v°'et hashama·yim 'et
|    וְהָאָרֶץ   הָיְתָה   תֹהוּ   וָבֹהוּ      וְחֹשֶׁךְ   עַל־פְּנֵי   תְהוֹם
|..... `al-p°ne v°ḥo·shech vavohu .... hay°tah v°ha'a·retz
|    וְרוּחַ   אֱלֹהִים      מְרַחֶפֶת   עַל־פְּנֵי   הַמָּיִם׃
|hama·im: `al-p°ne ..... 'elohim v°ru·aḥ
|    וַיֹּאמֶר   אֱלֹהִים   יְהִי אוֹר
|'or y°hi ..... vayo·'mer
|    וַיְהִי־אוֹר׃
|.........
|    וַיַּרְא   אֱלֹהִים   אֶת־הָאוֹר כִּי־טוֹב
|..... 'et-ha'or 'elohim vayar'
|    וַיַּבְדֵּל   אֱלֹהִים בֵּין הָאוֹר וּבֵין      הַחֹשֶׁךְ׃
|haḥo·shech: uven ha'or ... 'elohim vayavdel
|    וַיִּקְרָא   אֱלֹהִים   לָאוֹר יוֹם      וְלַחֹשֶׁךְ   קָרָא   לָיְלָה
|la·ylah kara' v°laḥo·shech ... la'or 'elohim .....
|    וַיְהִי־עֶרֶב   וַיְהִי־בֹקֶר יוֹם   אֶחָד׃
|..... yom vayhi-vo·ker .....
```

APPENDIX 9(j)

Genesis Ch 1, vs 1 & 2: BITEX stage-three cloze output
in Hebrew "screen size" fixed spaced modern alphabet
and Roman modified "screen size" fixed spaced "simplified modern" alphabet

```
                                                                  בְּרֵאשִׁית |
                                                                  .........|
                                                                           |
                                                                           |
                                                        בְּרֵאשִׁית  בָּרָא  אֱלֹהִים |
                                                        ..... bara' b°re'shit|
                                                 אֵת     הַשָּׁמַיִם  וְאֵת     הָאָרֶץ: |
                                                 ..... v°'et ..... 'et|
                                  וְהָאָרֶץ   הָיְתָה  תֹהוּ  וָבֹהוּ      וְחֹשֶׁךְ  עַל־פְּנֵי  תְהוֹם |
                                  ..... `al-p°ne ..... vavohu .... hay°tah v°ha'a·retz|
                                                 וְרוּחַ  אֱלֹהִים  מְרַחֶפֶת  עַל־פְּנֵי  הַמָּיִם: |
                                                 ..... `al-p°ne ..... 'elohim v°ru·aḥ|
                                                        וַיֹּאמֶר  אֱלֹהִים  יְהִי אוֹר |
                                                        'or y°hi ..... vayo·'mer|
                                                                 וַיְהִי־אוֹר: |
                                                                 .....|
                                                        וַיַּרְא  אֱלֹהִים  אֶת־הָאוֹר כִּי־טוֹב |
                                                        ..... 'et-ha'or ..... vayar'|
                                                 וַיַּבְדֵּל  אֱלֹהִים בֵּין הָאוֹר וּבֵין     הַחֹשֶׁךְ: |
                                                 haḥo·shech: .... ha'or ... 'elohim vayavdel|
                                  וַיִּקְרָא  אֱלֹהִים  לָאוֹר יוֹם        וְלַחֹשֶׁךְ  קָרָא  לָיְלָה |
                                  la·ylah ..... v°laḥo·shech ... la'or 'elohim .....|
                                                 וַיְהִי־עֶרֶב  וַיְהִי־בֹקֶר יוֹם  אֶחָד: |
                                                 ..... yom vayhi-vo·ker .....|
```

APPENDIX 9(k)

Genesis Ch 1, vs 1 & 2: BITEX stage-four cloze output in Hebrew "screen size" fixed spaced modern alphabet and Roman modified "screen size" fixed spaced "simplified modern" alphabet

בְּרֵאשִׁית |
.... |
|
|
בְּרֵאשִׁית בָּרָא אֱלֹהִים |
.... bara' .... |
אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃ |
.... v^{e}'et .... 'et |
וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם |
..... 'al-p^{e}ne .... vavohu .... hay^{e}tah .... |
וְרוּחַ אֱלֹהִים מְרַחֶפֶת עַל־פְּנֵי הַמָּיִם׃ |
.... 'al-p^{e}ne .... 'elohim .... |
וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר |
'or .... vayo·'mer |
וַיְהִי־אוֹר׃ |
.... |
וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב |
...... 'et-ha'or .... vayar' |
וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ׃ |
haḥo·shech: .... ha'or ... 'elohim .... |
וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה |
la·ylah ..... v^{e}laḥo·shech ... la'or .... |
וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד׃ |
...... yom .... |

APPENDIX 9(1)

Genesis Ch 1, vs 1 & 2: BITEX stage-five cloze output in Hebrew "screen size" fixed spaced modern alphabet and Roman modified "screen size" fixed spaced "simplified modern" alphabet

```
                                                                  בְּרֵאשִׁית |
                                                                  .........|
                                                                           |
                                                                           |
                                                                           |
                                                  בְּרֵאשִׁית   בָּרָא   אֱלֹהִים |
                                                  ......... bara' .........|
                                  אֵת        הַשָּׁמַיִם   וְאֵת      הָאָרֶץ׃ |
                                  ..... ..... ..... ve'et ..... ..... ... |
       וְהָאָרֶץ    הָיְתָה  תֹהוּ    וָבֹהוּ        וְחֹשֶׁךְ   עַל־פְּנֵי  תְהוֹם |
..... `al-pene ..... ..... ..... .... haxetah ..... ..... ..... ..... |
                  וְרוּחַ  אֱלֹהִים      מְרַחֶפֶת   עַל־פְּנֵי    הַמָּיִם׃ |
                  ..... ..... ..... ..... ..... 'elohim ..... ..... ..... |
                                                  וַיֹּאמֶר  אֱלֹהִים  יְהִי אוֹר |
                                                  'or .... ..... ..... vayo·'mer |
                                                                  וַיְהִי־אוֹר׃ |
                                                                  .........|
                                                  וַיַּרְא  אֱלֹהִים  אֶת־הָאוֹר כִּי־טוֹב |
                                                  ..... 'et-ha'or ..... ..... ..... |
                  וַיַּבְדֵּל  אֱלֹהִים בֵּין  הָאוֹר וּבֵין        הַחֹשֶׁךְ׃ |
                  ..... ..... ... 'elohim ..... ..... ..... haḥo·shech: |
       וַיִּקְרָא  אֱלֹהִים  לָאוֹר יוֹם        וְלַחֹשֶׁךְ  קָרָא   לָיְלָה |
..... ..... ..... ... la'or ..... ..... ..... ..... la·ylah |
                  וַיְהִי־עֶרֶב    וַיְהִי־בֹקֶר יוֹם  אֶחָד׃ |
                  ..... yom ..... ..... ..... ..... ..... ..... |
```

APPENDIX 9(m)

Genesis Ch 1, vs 1 & 2: BITEX stage-six cloze output
in Hebrew "screen size" fixed spaced modern alphabet
and Roman modified "screen size" fixed spaced "simplified modern" alphabet

בְּרֵאשִׁית

....

בְּרֵאשִׁית בָּרָא אֱלֹהִים

.... bara' ....

אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃

....

וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם

.... hay*tah ....

וְרוּחַ אֱלֹהִים מְבַחֶפֶת עַל־פְּנֵי הַמָּיִם׃

.... 'elohim ....

וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר

'or ....

וַיְהִי־אוֹר׃

....

וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב

....

וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ׃

.... 'elohim ....

וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה

.... la'or ....

וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד׃

...... yom ....

APPENDIX 9(n)

Genesis Ch 1, vs 1 & 2: BITEX stage-seven cloze output
in Hebrew "screen size" fixed spaced modern alphabet
and Roman modified "screen size" fixed spaced "simplified modern" alphabet

```
| בְּרֵאשִׁית
|..........
|
|
| בְּרֵאשִׁית בָּרָא אֱלֹהִים
|.......... ..... ........
| אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃
|... .......... ..... ..........
| וְהָאָרֶץ הָיְתָה תֹהוּ וָבֹהוּ וְחֹשֶׁךְ עַל־פְּנֵי תְהוֹם
|.......... ....... .... ...... .......... ......... .....
| וְרוּחַ אֱלֹהִים מְרַחֶפֶת עַל־פְּנֵי הַמָּיִם׃
|........ ....... .......... ....... ........
| וַיֹּאמֶר אֱלֹהִים יְהִי אוֹר
|......... ....... .... ...
| וַיְהִי־אוֹר׃
|..........
| וַיַּרְא אֱלֹהִים אֶת־הָאוֹר כִּי־טוֹב
|...... ....... .......... ......
| וַיַּבְדֵּל אֱלֹהִים בֵּין הָאוֹר וּבֵין הַחֹשֶׁךְ׃
|....... ........ .... ..... .... ...........
| וַיִּקְרָא אֱלֹהִים לָאוֹר יוֹם וְלַחֹשֶׁךְ קָרָא לָיְלָה
|......... ....... ..... ... ........... ..... .......
| וַיְהִי־עֶרֶב וַיְהִי־בֹקֶר יוֹם אֶחָד׃
|........... .......... ... ......
```

APPENDIX 9(o)

Genesis Ch 1, vs 1 & 2: BITEX stage-eight cloze output
in Hebrew "screen size" fixed spaced modern alphabet
and Roman modified "screen size" fixed spaced "simplified modern" alphabet

| בראשית

| ....

|

|

| בראשית ברא אלהים

| ....

| את השמים ואת הארץ:

| ... .

| והארץ היתה תהו ובהו וחשך על־פני תהום

| ....

| ורוח אלהים מרחפת על־פני המים:

| ....

| ויאמר אלהים יהי אור

| ....

| ויהי־אור:

| ....

| וירא אלהים את־האור כי־טוב

| ....

| ויבדל אלהים בין האור ובין החשך:

| ....

| ויקרא אלהים לאור יום ולחשך קרא לילה

| ....

| ויהי־ערב ויהי־בקר יום אחד:

| ....

This page is inserted to facilitate optional printing on both sides of the page, each figure, appendix or references starting with an odd number.

REFERENCES

Becker, Joseph D. (1984, July). Multilingual word processing. Scientific American, pp. 96-107.

Brooks, L. R. (1977). Visual pattern in fluent word identification. In A. S. Reber & D. Scarborough (Eds.), Toward a psychology of reading. Hillsdale, NJ: Erlbaum.

Brooks, L. & Miller, A. (1979). Knowledge of an alphabet. In P. A. Kolers, M. E. Wrolstad & H. Bouma (Eds.), Processing ofvisible language: Vol. 1. New York: Plenum Press.

Cassuto, U. A commentary on the book of Exodus. Jerusalem: Magnes Press, Hebrew University.

Chall, J. S. (1967). Learning to read: The great debate. New York: McGraw Hill.

Chall, J. S. (1983). Stages of Reading Development. New York: McGraw Hill.

Durkin, D. (1970). Teaching them to read. Boston: Allyn & Bacon.

Gutenberg (Computer program). Scarborough: Gutenberg Software.

Kellogg, E. P. Jr. (1983). Romanization to facilitate the teaching of modern Hebrew to adult native speakers of English. Walnut Creek, CA.: EHUD International Language Foundation. (ERIC Document Reproduction Service No. ED 235 689)

Kellogg, E. P. Jr. (1983). Shalom home study course in conversational Hebrew. Walnut Creek, CA.: EHUD International Language Foundation.

Lambdin, T. O. (1971). Introduction to biblical Hebrew. New York: Charles Scribner's Sons.

Livny Y. and Kokhba M. (1973). A Hebrew Grammar for Schools and Colleges. Jerusalem: Rubin Mass.

Pei, M. (1952). The story of English. New York: J.B.Lippincott.

Multi-lingual Scholar, Version 3.0 (computer program) (1987). Santa Monica: Gamma Productions.

Nota Bene Version 3 (computer program) (1987) New York: Dragonfly Software.

REFERENCES (CTD)

Reif, J. A. & Levinson, H. (1965). Hebrew basic course. Washington: Foreign Service Institute, Department of State.

Rivers, W. M. & Temperley, M. S. (1978). A practical guide to the teaching of English as a second or foreign language. New York: Oxford University Press.

Trim, J. L. M. (1978). Developing a unit credit scheme of adult language learning. Toronto: Pergamon Press.

Sternberg, R. J., Powell, J. S. & Kaye, D. B. (1983). Teaching vocabulary-building skills: a contextual approach. In A. C. Wilkinson (Ed). Classroom computers and cognitive science. New York: Academic Press.

Talmage, F., Rabin, C. & Garshowitz L. (1977). Study guide for sifron la-student. Toronto: University of Toronto Press.

Weingreen, J. (1939) A practical grammar for classical Hebrew. Oxford: Oxford University Press.