[*Mechanical Translation*, Vol.6, November 1961]

A High-Speed Large-Capacity Dictionary System

by Sydney M. Lamb and William H. Jacobsen, Jr.,* University of California, Berkeley

A system of dictionary organization is described which makes it possible for a computer with 32,000 words of core storage to accommodate a vocabulary of hundreds of thousands of words, with a look-up speed of over a hundred words per second. The central part of the look-up process involves using the first few letters of each word as addresses, one after another.

Introductory

This paper describes a method of adapting dictionaries for use by a computer in such a way that comprehensiveness of vocabulary coverage can be maximized while look-up time is minimized. Although the programming of the system has not yet been completed, it is estimated at the time of writing that it will allow for a dictionary of 20,000 entries or more, with a total look-up time of about 8 milliseconds (.008 seconds) per word, when used on an IBM 704 computer with 32,000 words of core storage. With a proper system of segmentation, a dictionary of 20,000 entries can handle several hundred thousand different words, thus providing ample coverage for a single fairly broad field of science. Although the system has been designed specifically for purposes of machine translation of Russian, it is applicable to other areas of linguistic data processing in which dictionaries are needed.

Preliminary Definitions

An entity for which there is (or should be) a dictionary entry is a lexical item or *lex.* A text is made up of a sequence of lexes, for each of which we hope to find a dictionary entry, if we are translating or analyzing it. It is also made up of a sequence of words, but if any segmentation of words is incorporated in the system, many of the words will consist of more than one lex. (In the system used at the University of California, there are about two lexes per word, on the average.) A *word,* on the graphemic level, is a sequence of graphemes which can occur between spaces; any specific occurrence of a word is a *word token.* A lex (in the present discussion) has its existence on the graphemic level, and corresponds to a *lexeme* on the morphemic level. Any specific occurrence of a lex is a *lex token.* The relationship between lex and lexeme is like that between morph and morpheme: that is, a lexeme may have more than one graphemic representation, or lex (since one or more of its constituent morphemes has more than one morph). Such alternate representations may be called *allolexes* of the lexeme. Just as a (graphemic) word may comprise more than one lex, so a lex may comprise more than one morph.

A dictionary entry may be thought of as consisting of two parts, the *heading* and the *exposition.*[1] The heading is an instance (or coded representation) of the lex itself, and serves to identify the entry. The remainder of the entry, the exposition, is the information which is provided concerning that lex. If the dictionary is part of an automatic translation system, the exposition might contain the following three parts (not necessarily separated): (1) the syntactic-semantic code, signifying distributional and semantic properties about which information may be needed in dealing with other lexemes occurring in the environment of the one in question; (2) (highly compressed) instructions for selecting the appropriate target representation for any given environment; and (3) the target representations. In an efficient automatic dictionary system the target representations might be kept together on tape, to be brought into core storage as a body when needed, after the look-up and translation proper have been completed. In this case, the expositions would be split up, the target representations being separated from the rest; in their place would be put the addresses where the representations would be located after the "target-language tape" has been run into core storage. Then we have what may be called *abbreviated expositions.*[2]

Where a lexeme has allolexes, there must be a heading for each allolex, but (except for that part of the syntactic code which defines their complementary distribution) they all have the same exposition. The exposition, then (aside from the qualification mentioned parenthetically above), is oriented to the morphemic level, while the headings are graphemic in character. In a dictionary, the exposition for such a lexeme could be repeated under each allolex, or all but one of the allolexes could have *cross-referential expositions,* which

* This work was supported by the National Science Foundation. This manuscript was received over one year ago. The authors recently submitted a number of revision describing improvements in the dictionary systems; the editor regrets that these revisions were received too late to be included in the present article.

[1] These terms correspond (more or less) to *argument* and *function* in William S. Cooper, The Storage Problem, *Mechanical Translation* 5.74-83 (1958). (Although the authors favor the principle of priority in nomenclature, they felt it necessary to introduce these new items since Cooper's *argument* applies ambiguously to both *heading* and *vestigand,* which can be quite different.) The authors have profited not only from this article, but also from several informal discussions with Mr. Cooper.

[2] The next stage of refinement would be to split many of the target forms into parts which recur in other target forms. This would require using more than one address in the abbreviated exposition for many forms, but it enables simplification of many translation instructions, as well as a considerable shortening of the target language tape.

would refer to the full exposition given for that one allolex.

A word being looked up in the dictionary, or ready to be looked up, may be called the *vestigand* (based on the gerundive of Lat. vestigare, 'to track, trace out; to search after, seek out; to inquire into, investigate;' hence "that which is to be traced out, searched after, investigated"). A vestigand will coincide with some heading only in the special case in which it is not segmented; otherwise it will contain two or more headings. The look-up process involves segmentation as well as location of headings. (An alternative approach has been used[3] in which "suffixes" are separated before the look-up process begins. Such a practice is rejected here, since it (1) often leads to false segmentation; (2) requires the use of arbitrary, non-structural segmentation principles; (3) involves setting up more stem allolexes, hence more dictionary entries, than would otherwise be necessary.) Every word token in a text is a vestigand at the time it is being looked up.

Simple System

Let us begin considering the dictionary problem in terms of the simplest type of organization, in which the machine dictionary is set up very much like an ordinary printed dictionary, except that stems themselves are used as headings, rather than combinations of the stems with standard suffixes such as nominative singular or infinitive. In the simple system, then, there is a list of entries, each one containing its heading, followed by the exposition. For this type of dictionary, the look-up process would involve matching the vestigand or part of it with one of the headings, after which the exposition next to this heading would be placed where needed for further reference after the process of look-up has been completed. (We assume for any type of machine dictionary system that the look-up process is handled for all of the words in the text or some portion thereof before the next stage of translation begins. This sequence of operations has apparently been universally recognized as essential for translation by computers, because of the limitation in the size of rapid-access memories.)

In this type of organization the dictionary is set up much like dictionaries that are used by human beings, except for the obvious adaptations needed for storage in the machine—such as the use of binary coding, etc.

The reason we have a problem is that in any of the available computers there is insufficient space to provide for the whole dictionary within the rapid access memory. The usual solution has been the "Batch Method,"[4] in which each "batch" of words (i.e. all the word tokens in a portion of text) is alphabetized before the look-up proper begins. The dictionary is stored on magnetic tape and is organized by alphabetic order of the headings like familiar paper dictionaries. In the look-up process, it is brought into core storage a portion at a time, and all the words in the alphabetized batch are looked up in one pass of the tape. As headings are matched, the adjoining expositions are stored in some specified location still in alphabetic order of the corresponding headings. Having obtained the expositions for all the lexical items in the batch, the machine must re-sort them back into text order. Thus there are two areas of nonproductive data processing: sorting the vestigands into alphabetic order and sorting the expositions back into text order. If this excess baggage could be done away with, a great saving of time would result.

Segregating the Headings

We have already noted that it might be efficient to divide each exposition into its target representations and an abbreviated exposition, keeping all the target representations together in one body until needed. The amount of time that can be reclaimed by such separation depends to a great extent on various features of the translation system itself. In any case, a much more significant saving of time will result if an additional separation is effected. We may detach the headings from the (abbreviated or full) expositions and then combine the headings into one body and the expositions into another.

This principle has already been implemented in a look-up system designed at The RAND Corporation.[5] The body of expositions, which we may call the *exposition list* for short, is kept on magnetic tape until after the essential part of the look-up process has been completed. The economy involved in terms of space saving for the look-up itself is obvious. The location of a given dictionary entry requires that only the heading part of the entry be known. And for dictionaries of up

[3] See, for example, A. G. Oettinger, W. Foust, V. Giuliano, K. Magassy, and L. Matejka, *Linguistic and Machine Methods for Compiling and Updating the Harvard Automatic Dictionary,* Preprints of Papers for the International Conference on Scientific Information, Area 5, 137-159 (1958), especially p. 141.

[4] Previously described (with the term *batch)* by Victor H. Yngve, The Technical Feasibility of Translating Languages by Machine, *Electrical Engineering* 75.994-999 (1956), p. 996. This method has been used by MT groups at Georgetown University, Ramo-Wooldridge, and Harvard. See A. F. R. Brown, *Manual for a "Simulated Linguistic Computer"—A System for Direct Coding of Machine Translation,* Georgetown University, Occasional Papers on Machine Translation No. 1, Washington, D. C., 1959, p. 36; *Experimental Machine Translation of Russian to English,* Ramo-Wooldridge Division of Thompson Ramo Wooldridge Inc., Project Progress Report M20-SU13, Los Angeles, 1958, p. 26; and P. E. Jones, Jr., *The Continuous Dictionary Run,* Mathematical Linguistics and Automatic Translation, Report No. NFS-2, Sec. I, Harvard Computation Laboratory, 1959, pp. 11-18.

[5] But not yet programmed. The system was designed by Hugh Kelly and Theodore Ziehe of the RAND programming staff, but programming of it has been suspended during a test of the feasibility of an alternative system which makes use of a RAMAC. Information concerning the system was obtained from personal communication. Another system designed by Mr. Ziehe which also uses the principle of segregating headings is described by him in Glossary Look-up Made Easy, (to appear in the proceedings of the National Symposium on Machine Translation). The idea of heading segregation has been mentioned previously in print by William S. Cooper, op. cit. (footnote 1), p. 75.

to several thousand entries, it is possible to store all the headings within a 32,000 word rapid access memory. At the end of the look-up process, the machine has for each lex token an address for the exposition to which it corresponds.

The RAND system uses the familiar approach to the process of locating headings, in that the headings are listed in core storage and it is necessary to find the right one by matching. The headings are grouped into eighteen different sections, depending on their length (from one to eighteen letters) and the technique of successive binary divisions[6] is used within the appropriate length-group. If no match is found, one or more final letters (comprising a possible suffix) are chopped off and the process is repeated.

Now, for a dictionary of adequate size, the exposition list itself is much too large to be in core storage at one time; it may comprise about 70 to 90 per cent of the volume of the dictionary as a whole. On the other hand, a 32,000 word memory can contain the expositions for all the different lexes occurring in any one text, provided it is of reasonable length. The RAND group has found that while a typical issue of a journal contains around 30,000 word tokens, there are never more than 3,000 different lexical items represented in it, where by lexical items we mean the type of units used in the RAND system. (The degree of segmentation for that system is less than that which has been worked out at the University of California. Therefore, the corresponding figure would be less than 3,000 for the "Berkeley system".) Abbreviated expositions for all those 3,000 dictionary entries can be contained in core storage at one time, since an estimate of eight machine words as the average size of an abbreviated exposition is liberal.[7] This means that an added feature is necessary which will make it possible to bring into core storage for the stage of translation proper only those two or three thousand expositions which are actually needed. There are several ways of making this possible, one of which is included in the RAND system. A somewhat different process is incorporated in the present scheme. We may call it the *intermediate stage.*

As each of the headings is located, what will be found is neither the exposition itself nor the address where the exposition will be stored. It is, instead, what we may call the *intermediate address.* After the headings for the text have been identified, in place of the original text there will be arranged in text sequence a series of these intermediate addresses, one for each lex token. Then about twenty thousand words of core storage are to be filled from tape (in one file) with what we may call the *intermediate list.* Each machine word of the intermediate list represents a particular dictionary entry, and each intermediate address is the address of the corresponding word in the intermediate list. The use of the intermediate list is explained below (see *The Intermediate Stage).*

When the intermediate stage has been completed, only the expositions which are actually needed are left in core storage, and they are all immediately addressable during the stage of translation proper.

Addressing the Dictionary Entry

If core storage were large enough that we could use the shape of the heading itself as an address, only the intermediate address would have to be stored, and the heading, rather than occupying storage space, would be the address of the location where the intermediate address would be stored. Obviously, there is no core storage large enough for this method. Moreover, if there were, its use for this purpose would be a colossal extravagance. Let us consider, however, some of the more realistic aspects of this general idea. Suppose we were to take just the first two letters of the vestigand and use them as an address. If the standard 6-bit code is used, the table needed would require 4,096 (2^{12}) machine words. Even to use this device with no further refinements gives a rather efficient system for as much as it covers. That is, if we get the desired location narrowed down according to the first two letters, we have already come very close to it, and we have spent practically no time at all.

We have saved some space as well. Suppose the dictionary has 18,000 entries. Then the space required to store the first two letters of each heading would be the equivalent of 6,000 machine words. But our table occupies only 4,096 words, so we have a space saving of almost 2,000 words.

Below there are introduced a series of refinements which make it possible to efficiently use a portion of each vestigand as an address. Space limitations require that the process be somewhat indirect; nevertheless, the system provides extremely rapid entry into a dictionary.

First of all we shall see that it is necessary to conduct the addressing letter by letter. This principle is, in fact, the key to the system.[8] It enables us to take advantage of the fact that letters tend to occur in certain combinations, and that many "theoretically possible" combina-

[6] Described, for example, by J. P. Cleave, A Type of Program for Mechanical Translation, *Mechanical Translation* 4.54-58 (1957).

[7] For the purposes of this paper, a machine word consists of 36 bits. An abbreviated exposition of more than average complexity might have a syntactic-semantic code of three machine words, three machine words of compressed instructions, and two machine words of target-form addresses.

[8] An application of this principle similar to the one given here is described by Rene De La Briandais, File Searching Using Variable Length Keys, *Proceedings of the Western Joint Computer Conference,* 1959, 295-298. His system differs from ours primarily in that each successive table is scanned entry-by-entry to determine whether the next letter is entered, instead of being directly addressed with the next letter as an index. The use of this alternative type of letter table as an intermediate step between the directly addressable letter tables and the truncate lists would be a device for conserving space so as to allow more headings to be accommodated, at a cost of greater look-up time and more red tape. For this purpose, the tables should probably be arranged so that the entries in a given table occupy successive machine words, rather than being in the overlapping arrangement described in this reference, which is more applicable to situations in which the tables are to be repeatedly formed anew.

tions of letters do not occur in natural written languages. It also permits a simple and direct approach to segmentation. In conducting the look-up operation, we may use the first letter of the vestigand as an address in the "first-letter table," a table of sixty-four words (if the standard 6-bit code is used). At this address is given the final address of the table to consult for the next letter.[9] At the location corresponding to each possible first character, there is an address which gives the location of the table for the second. The second letter of the vestigand may now be placed in an index register, and the proper address for the third-letter table may be obtained by addressing. The essential part of the look-up routine is as follows, in the language of the Share Assembly Program:

```
LDQ   VSTGD   Load vestigand into MQ.
PXD           Clear accumulator.
LGL   6       Put first letter into accumulator,
PAX   ,1          then into index register 1.
CLA   FIRST, 1 Get address from first-letter table.
STA   *+4
PXD           Clear accumulator.
LGL   6       Put second letter into accumulator,
PAX   ,1          then into index register 1.
CLA   **,1    Get address from second-letter table.
STA   *+4
PXD
etc.
```

For the first letter we need sixty-four positions in the table, and for the second we would need sixty-four tables of sixty-four words each—*if* the language of the text used sixty-four characters any of which could occur initially. But in fact we do not need this many. For the second letter we need only as many tables as there occur first letters. If we are using an IBM 704 computer, only forty-eight characters are readily available (the letters of the alphabet, ten digits, and various other symbols); if we limit ourselves to these, then for the second character we will need forty-eight tables of sixty-four entries each, occupying 3,072 machine words. (We must allow space for an entire block of sixty-four entries per table in order to provide insurance against the possibility of an error.)

Instead of visualizing a set of forty-eight second-letter tables and a first-letter table, one may prefer to think of an array containing forty-eight rows and sixty-four columns, in which the row we get represents the first letter of the word and the position we get on it represents the second letter.

When we get to the third letter, the economy of letter-by-letter addressing becomes more striking, as we need only a few hundred tables, very much less than the 4,096 (64 × 64) which would be necessary for direct addressing taking the first three letters together. The number of tables needed for the third character (if this technique is used for the third character of all vestigands) is equal to the number of occurring combinations of first two characters which can be followed by a third character; i.e., there must be a third-letter table for each such combination. If all possible combinations of the forty-eight available characters occurred as first and second letter, the number of tables needed would be 2,304 (48 × 48), but of course most of these combinations do not occur. Further details are given below under the heading *Space Needs for Letter Tables.* It is of course necessary to conserve further space. The need becomes particularly clear when we realize that there are over 3,000 occurring combinations of first three letters. There are two possible approaches to reducing space needs, both of which must be used: (1) the adoption of refinements to cut down the size of the tables; (2) the elimination of the use of these tables in certain situations.

[9] We refer here and in what follows specifically to the IBM 704, in which index registers are subtractive.

Code Conversion

A possibility for reducing the size of the tables is suggested by the fact that there are for most languages thirty-two letters or less in the alphabet, whereas there are sixty-four entries needed in a table if the usual 6-bit code is used. (The number of entries in a table of this type is, of course, 2^n, where n is the number of bits in the code.) Thirty-two characters can be handled by a 5-bit code, and the number of entries needed in a table would be only thirty-two. Thus we can cut the space requirements by half.

Naturally, some provision must be made for non-alphabetic characters. There are various ways of managing this. In a language with twenty-six letters like English, there are six extra spaces within the thirty-two for the most common non-alphabetic symbols such as blank, comma, period, semi-colon, etc. The Russian alphabet seems less tractable at first glance, since it has thirty-two letters. Two of these letters are, however, in complementary distribution so that only one symbol is needed for both of them, and an efficient coding system would use only one symbol for both, no matter what type of dictionary organization is used. These letters are the soft sign, which occurs only after consonants, and the short "i", which occurs only after vowels. In the transliteration system used at the University of California, both of these letters are represented by *j*. The transliterated alphabet, then, has only thirty-one letters. The thirty-second position in the table may be used for "nothing" (i.e., end of a heading); its contents will be the intermediate address for a heading ending at that point, rather than the address of another table.

Space in tables need not be provided for the non-alphabetic characters, since they require special treatment. For example, in the case of arabic numbers, we will not want to look up the whole number since we will not want to have all the arabic numbers in the dictionary. Instead, whenever an arabic number comes up, it will be handled character by character, and no translation will be necessary since each character will be the same in the target language as in the input text.

Thus the computer will not proceed in the same way for the special symbols as for the letters, and it will not need letter tables for them.

The machine can convert from the standard BCD to a code in which the thirty-*one* letters have a zero in the first bit and all the other characters (punctuation marks, numerals, etc.) have a one in the first bit. We still have a 6-bit code, but it can be used to give an effective 5-bit code. Suppose we place the first bit in the sign bit position of the machine word; the other five bits can be placed in the low-order positions of the same machine word. This makes it easy to check whether the next character is alphabetic or not, and after the checking, we have in effect a 5-bit code, making possible a table of only thirty-two entries.

The conversion from the BCD code to the one under consideration can be done efficiently at the same time as the look-up process. It is not worth while to convert for the first letter as only thirty-two machine words would be saved, and this is an insignificant amount. With this refinement, the look-up process would be as follows:

LOOK-UP ROUTINE WITH CODE CONVERSION

```
LDQ   VSTGD     Load vestigand into MQ.
PXD             Clear accumulator.
LGL   6         Put first letter into accumulator,
PAX   ,1            then into index register 1.
CLA   FIRST,1   Get address from first-letter table.
STA   *+7
PXD             Clear accumulator.
LGL   6         Put second letter into accumulator,
PAX   ,1            then into index register 1.
CLA   CONV,1    Code conversion.
TMI   NOLTR     Test for non-alphabetic symbol.
PAX   ,1        Place 5-bit code in index register 1.
CLA   **,1      Get address from second-letter table.
STA   *+7
PXD
etc.
```

Two Table Entries per Machine Word

Let us at first consider this refinement independently of the previous one. It is another means of cutting down the length of the tables from sixty-four to thirty-two words. Since the table entries consist only of addresses, we may economize by placing two of them in each machine word. One can be put in the address position, the other in the decrement. Half-words cannot be addressed if we are using the 704, so some speedy contrivance must be used whereby one of the bits in the letter code will indicate which half of the word is needed in each case. In shifting a next letter from the MQ register into the accumulator, we can shift only five places instead of six. Then the sixth bit will be in the MQ sign position, and the instruction TQP (transfer on MQ plus) can be used to indicate whether the address or decrement is needed. Again it is unnecessary to use this device before the second letter.

LOOK-UP ROUTINE WITH TWO ENTRIES PER WORD

```
LDQ   VSTGD     Load vestigand into MQ.
PXD             Clear accumulator.
LGL   6         Put first letter into accumulator,
PAX   ,1            then into index register 1.
CLA   FIRST,1   End of table for second letter[10]
PAX   ,2            placed into index register 2.
PXD             Clear accumulator.
LGL   5         First five bits of second letter
PAX   ,1            placed into index register 1.
CLA   ,3        Double indexing.
TQP   * + 3     If MQ minus,
PDX   ,2            use the decrement;
TRA   *+2           if MQ plus,
PAX   ,2            use the address.
LGL   1         Remove sixth bit of second letter
                        from MQ.
PXD             Clear accumulator.
LGL   5         First five bits of third letter
PAX   ,1            into index register 1.
CLA   ,3        Double indexing.
TQP   * + 3     If MQ minus,
PDX   ,2            use the decrement;
TRA   *+2           etc.
etc.
```

The next step is to combine the two refinements, with the result that only sixteen cells are needed for each table. Then the tables have thirty-two entries, two of them in each word, and accommodate only the letters. Special treatment, as indicated above, is given to the non-alphabetic characters; that is, if the transfer on minus is taken, then in most cases it will mean that the end of the word has been reached. On the other hand, the first time this device is used, say at the second letter the minus sign might also reflect a word composed of special symbols, such as an arabic numeral.

In combining the two refinements we have a problem that can be stated as follows: How can we convert the code and still leave one bit in the MQ sign position? One possibility is to take note of the sign of the MQ before converting, using a sense light or other device. To make this feasible, it is necessary that of the BCD representations of the thirty-one letters, sixteen have a one in the low order bit and the other fifteen a zero, or vice versa. Another possibility is to convert from the bits other than the low order bit, leaving the latter in the MQ to be tested after the conversion. This would impose the additional restriction that the pairs of BCD

[10] This and the following look-up routines differ from the preceding ones in that the address for the next table is placed in an index register and used for double indexing with the code for the next letter, instead of being stored further along in the sequence of instructions. This alternative procedure is more convenient for purposes of segmentation. The double indexing in this routine makes it necessary that the addresses of the ends of the tables for the second and subsequent letters be integral multiples of 32 (i.e., the five low-order bits must all be zero), since in double indexing on the 704 the "logical *or*" of the contents of the index registers constitutes the amount by which an address is modified.

symbols that differ only in the low order bit both represent either alphabetic characters or non-alphabetic characters. Since the coding of Russian in BCD requires the choice of certain more or less arbitrary symbols to represent some of the Russian letters anyway, these requirements can be met by a wise choice of these characters. A routine for look-up based on this latter device would treat the second letter as follows:

```
PXD             Clear accumulator.
LGL  5          First five bits of second letter
PAX  ,1             placed into index register 1.
CLA  CONV,1     Code conversion.
TMI  NOLTR      Test for non-alphabetic symbol.
PAX  ,1         4-bit code placed in index register 1.
CLA  ,3         (Table address is in index register
                    2.)
TQP  *+3        If MQ minus,
PDX  ,2             use the decrement;
TRA  *+2            if MQ plus,
PAX  ,2             use the address.
LGL  1          Remove sixth bit of second letter
                    from MQ.
PXD             Clear accumulator for third letter.
etc.
```

An alternative possibility is to test the low order bit of the accumulator instead of the MQ sign bit, after which the accumulator may be shifted one position to the right. This will give a look-up routine that is just one cycle per letter slower,[11] on the average, due to the extra transfer that must be taken half the time after the LBT (low order bit test). This alternative has the advantage of making no demands on the choice of BCD characters; the low order bit is tested after code conversion.

LOOK-UP ROUTINE WITH SIXTEEN WORDS PER TABLE, USING LBT
(Beginning with second letter)

```
LGL  6          Put second letter into accumulator,
PAX  ,1             then into index register 1.
CLA  CONV,1     Code conversion.
TMI  NOLTR      Test for non-alphabetic symbol.
LBT             If low order bit 1, use the decrement;
TRA  * +6           if low order bit O, use the address.
ARS  1          Now a 4-bit code,
PAX  ,1             which is placed into index register
                    1.
CLA  ,3         End of table for third letter[12]
PDX  ,2             placed into index register 2.
TRA  *+5
ARS  1          Now a 4-bit code,
PAX  ,1             which is placed into index register
                    1.
CLA  ,3         End of table for third letter[12]
PAX  ,2             placed into index register 2.
PXD             Clear accumulator for third letter.
LGL  6          Put third letter into accumulator.
etc.
```

[11] A cycle on the 704 is 12 microseconds (i.e., 12 millionths of a second).

Neither of the above alternatives exploits to the fullest extent the possibilities offered by the machine. Instead of testing the low-order bit, after which we must transfer half the time and shift the accumulator every time, we may design the conversion table so that the bit to be tested will be in the high order bit of the address field, leaving the other four bits in the low order positions of the address field, in the same position they would occupy after the accumulator right shift of the preceding routine (see fig. 1). Then the operation TXL (transfer on index low or equal) can be used to distinguish the two table entries of the machine word. The use of this device is possible even though the bit position being tested is also used (in the other index register) in addressing the next table because (1) in multiple indexing on the 704, the logical *or* of the contents of the index registers determines the extent of address modification; and (2) the letter tables will occupy less than half of core storage (see *Allocation of Memory Space* below). Consequently, the high order bit of the index register defining the address of the next letter table can always be 1, and the presence or absence of 1 in the other index register will have no effect upon the address determination for the following CLA operation. (See fig. 2.)

LOOK-UP ROUTINE WITH SIXTEEN WORDS PER TABLE, USING TXL

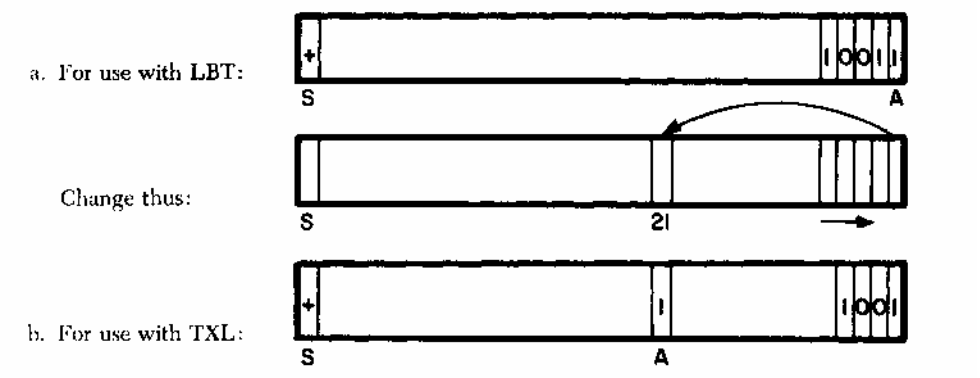
```
LDQ  VSTGD      Load vestigand into MQ.
PXD             Clear accumulator.
LGL  6          Put first letter into accumulator,
PAX  ,1             then into index register 1.
CLA  FIRST,1    End of table for second letter
PAX  ,2             placed into index register 2.
PXD             Clear accumulator.
LGL  6          Put second letter into accumula-
                    tor.
PAX  ,1             then into index register 1.
CLA  CONV,1     Code conversion.
TMI  NOLTR      Test for non-alphabetic symbol.
PAX  ,1         Get correct pair
CLA  ,3             of table entries.
TXL  *+ 3,1,8192[13]  If test bit is 1,
PDX  ,2             use the decrement;
TRA  *+2            if text bit is O,
PAX  ,2             use the address.
PXD             Clear accumulator for third
                    letter.
LGL  6          Put third letter into accumu-
                    lator,
PAX  ,1             then into index register 1.
CLA  CONV,1
etc.
```

It should be noted that the location of the end of the vestigand does not have to be known by the machine,

[12] The double indexing in this routine makes it necessary that the addresses of the ends of the tables for the second and subsequent letters be multiples of 16 (cf. footnote 10).

[13] This decrement consists of a 1 in the second position of the decrement field, all the rest zeros. The decrement could be any number from 32 to 16383.

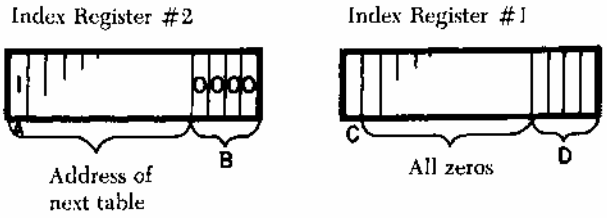
FIGURE 1. CODES FOR THE LETTER "T" (AFTER CODE CONVERSION).



S: + indicates alphabetic character
— non-alphabetic

A: 1 or 0 determines choice of address or decrement.

FIGURE 2. USE OF INDEX REGISTERS IN LETTER-BY-LETTER ADDRESSING.



A: Always 1

B: All zeros

D: Determines proper pair of table entries.

C: Determines choice of one of the pair.

because as soon as the following space or punctuation mark comes up for consideration, the transfer on minus is taken after code conversion. After passage to the truncate lists (see below) it will remain unnecessary to know where the word ends, and no checking to distinguish space from alphabetic characters is needed. (Thus it is possible to have combinations of words, such as НЕСМОТРЯ НА for Russian, entered in the dictionary as units, where desired, provided that the (first) space in such combinations will not be encountered while the process is still in the stage of letter-by-letter addressing.)

Space Needs for Letter Tables

As indicated above, our space needs for the letter tables can be calculated with reference only to the alphabetic characters, of which there are thirty-one (contrasting ones) in Russian. For the first letter we need one table, and we need one second-letter table for each possible first letter. Three of the Russian letters (Й/Ь Ъ Ы) do lot occur initially, so twenty-eight second-letter tables are needed. Thus, at sixteen machine words per second-letter table, roughly 500 words of memory space will be occupied by the tables for the first and second letters, Note that the equivalent of over 6,000 machine words would be needed to store the first two letters of 20,000 leadings.

If the letter tables are used for the third letter of all vestigands, the number of tables needed is equal to the Lumber of possible combinations of first two letters, minus those combinations for which no third letter is possible. An estimate of this number may be obtained by tabulating the possibilities for all of the words in some appropriate dictionary. Table I shows all the possibilities for the first two letters occurring in Callaham's *Chemical and Technical Dictionary.*[14] Every square occupied by a number represents an occurring combination of first two letters. The number in each such square indicates how many possible third characters may follow, including period (in the case of abbreviations) and blank (or other punctuation, counted as one). An asterisk in a square betokens a prefix, implying that there may be additional possible third letters. The numbers in the column labeled *T* indicate how many second letters can follow each first letter. (In making the tabulation, capitalization of letters was ignored.) The table discloses 507 occurring combinations of first two letters out of a "theoretically possible" 961), twenty of which do not occur with any following third letter. This result would indicate a need for 487 third-letter tables. With regard to this figure it should be noted that the Callaham dictionary is somewhat larger than the one envisaged in this paper, since the former, by a rough

[14] Ludmilla Ignatiev Callaham, *Russian-English Technical and Chemical Dictionary* (New York and London, 1947). Tables I and IV through X were prepared by Janet V. Kemp and Alfred B. Hudson.

TABLE I. COMBINATIONS OF FIRST TWO AND FIRST THREE LETTERS (BASED ON CALLAHAM)

FIRST LETTER \ SECOND LETTER	а	б	в	г	д	е	ж	з	и	й/ъ	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	э	ю	я	Т
а	1	13	9	10	15		2	7	3	7	13	+24	11⁻	+18	1	9	+23	12	+12⁻	8	7	4	2	1	2				1	3		26
б	23				2	18			22*	2		8			+20		7			20								+7	2	6	1	13
в	+21	4	+7	5	7	2	1	13	16	1	6	4	7⁻	6	+24	8	8	14	+11⁻	7		2	3	2	3		2	+24	1	1	4	29
г	+24	+1	5	1⁻	1	17		1	+16			+10⁻	1	+7	16	1	9	+3		+19	1		+1						2	4		20
д	+18	+1	5	+1		25*	+6	1	24*	1	2	+3⁻	+2⁻	3	+27*		8⁻			17								6	3	13	2	20
е			+7	3	+6	+1	+5	1		+1	+1	4	2	2		1	6	2				1	2				1				+1		18
ж	10	1	1	1	1	+12	1		12					3	2	6		3			5	1	1							1	1		17
з	+28*		5		4	11			8			2	2	2	7		4			4								2		2	1	14
и	+2	3	2	7	8	6	1	+26*		1	5	+8	+10	16	8	7	+9	10	5	1		+4			1					3	1	23
й/ъ																																0
к	24		+7	+2	1	13			20	1	1	+11	+2	6	+27	+3	9	6	2	17								1	3	4		20
л	18	1		1		24	2		+23	6	+1	1	+1	1⁻	22			+1		17								3	1	14	9	18
м	+24		+2	+7		19			20		8	+5	+1	3	+24		1	3		16	+1	2		1	2			+7	3	3	8	21
н	+28*			1		+28*			+23	+2					+17		1		1	12								3	2	3		12
о	1	24	+11	8	10		3	+10⁻	1	3	13⁻	5	+15	+10	4	13	19	17	+28	1	7	5	2	6	6	2						25
п	23		1			19		+1	+23	2		7		1	+29*		9⁻	+4	3	19	1	1	1	1				6	2	2	4	21
р	21	1	3		1	25	2		18						20			1	4	13								10	2	2	4	15
с	19	+8	7⁻	+8	6	16	3	2	22		+9	+9	10⁻	+9	+28	8	8⁻	+7	13⁻	22	5	8	3	3	2		1	9	4	3	4	29
т	+24	4				21			18	2	2	3	1		+21	1	7	1		23				1⁻		3		5	4	8	4	19
у	4	7	6	5	9	4	+5	10	4	2	8	7	11	11	2	9	13	+16	11			5	1	4	5	1			2	1	2	27
ф	18					12		16		2		7		1⁻	12	7			3⁻	13								1	1	2		13
х	14		2			13			10			6	1	1	12		6		1	5												11
ц	4		2			17			17		+1		1	1	5			1		4								8	1	1		13
ч	16					16			12	2		1			3		1		2	13									3			10
ш	16		5			13			13		6	5	4	3	10	7	4		9	7		2										14
щ	2					9			+5											4												4
ъ																																0
ы																																0
э		4	+13	4	7		1	2		+16	13	8	10	12	6	7	11	5	9	1	4	2		1	3							21
ю		1	1	+3			2	1	1		3			6		1	+5	1	1		1											13
я		3	7	3	+4	2		3⁻	1	1	+3	4	6	6		2	12	6	2	1		2		3	1	3						21
																																507

\+ one of the possible third characters is a blank
— one of the possible third characters is a period
* a prefix of two letters
T number of possible second letters

estimate, contains some 33,000 entries. To be sure, many of these entries would not be represented as distinct entries in the planned system because of segmentation, but the effect of the segmentation is partially offset by the fact that many of the Callaham entries cover multiple lexemes. At any rate one may say that the Callaham dictionary probably accommodates a few thousand more lexemes than the twenty thousand to which the present discussion applies.

A tabulation based on a much smaller vocabulary[15] is shown in Table II. In this case the vocabulary consists of "words (not including proper names, formulas, mathematical symbols, and reference symbols) which appeared in a Russian physics text of 73,364 running words."[16] There appear to be about one-tenth as many lexemes represented in this listing as in the Callaham dictionary. Again, occupied squares indicate occurring combinations of first two letters; no count was made of the number of third-letter possibilities. There are 266 two-letter combinations, slightly more than half as many as in the larger vocabulary. As it represents the most frequently occurring lexemes (for physics), and thus to a large extent the most important two-letter combinations, Table II tends to provide a somewhat clearer picture of the patterns involved.

Table I also provides an estimate, albeit somewhat high, of the number of combinations of first three letters which can be expected. The number of such combinations occurring in Callaham (equal to the total of all the numbers in the squares less one for each + or –) is 3,440. A number of these, of course, cannot be followed by any fourth letter, since they constitute lexes and are not included in larger lexes. Allowing for this factor and for the larger size of the Callaham dictionary, we are still left with perhaps well over 2,000 as the number of fourth-letter tables which would be needed if the tables were to be used for the fourth letter of all vestigands. At sixteen words per table, this would amount to over 32,000 words, obviously too much space to allow. Aside from the fact that the limits of the capacity of core storage would be exceeded, it would be a highly inefficient utilization of space since the great preponderance of the table entries would be empty (reflecting lack of occurrence of the letter sequences involved).

There are devices available which could cut down the size of the tables to eight words each or even to less than that, at the expense of an appreciable amount of look-up time. However, any kind of letter-by-letter addressing or searching is necessarily inefficient after a certain point, just as searching through a list of headings for a match is inefficient up to that point. In other words, the letter-by-letter addressing should be continued until the possibilities for the desired heading have been narrowed down to a very few, at which point it becomes more efficient to consider up to several following letters at the same time.

Table III, which is based on Table I, shows the high proportion of combinations of first two letters for which there are only a very limited number of possibilities for the third letter. For about a quarter of the two-letter combinations, there is at most one possibility for the third letter. For well over a third of them there are only two possibilities or less, and for over half of them there are less than five possibilities.

TABLE III. NUMBER OF COMBINATIONS OF FIRST TWO LETTERS FOR WHICH THERE ARE VERY FEW POSSIBILITIES FOR THE THIRD LETTER. (FOR RUSSIAN, FIRST LETTER *a* THROUGH *O*. DATA FROM TABLE I.)

First Letter	Total Number of Different Possible 2nd Letters	Number of Different Possible 2nd Letters for which the Number of Possible Third Letters is only:				
		1 or 0	2 or less	3 or less	4 or less	5 or less
а	26	4	7	9	10	10
б	13	1	4	4	4	4
в	29	4	7	9	12	13
г	20	8	10	10	11	12
д	20	5	7	10	10	12
е	18	8	12	13	15	16
ж	17	9	10	12	12	13
э	14	1	6	6	9	10
и	23	6	7	10	10	12
й/ь	0	0	0	0	0	0
к	20	5	8	9	10	10
л	18	8	9	10	10	10
м	21	5	7	11	12	12
н	12	4	5	7	7	7
о	25	3	5	7	8	10
	276	71	104	127	140	151
% of 276:		26%	38%	46%	51%	55%

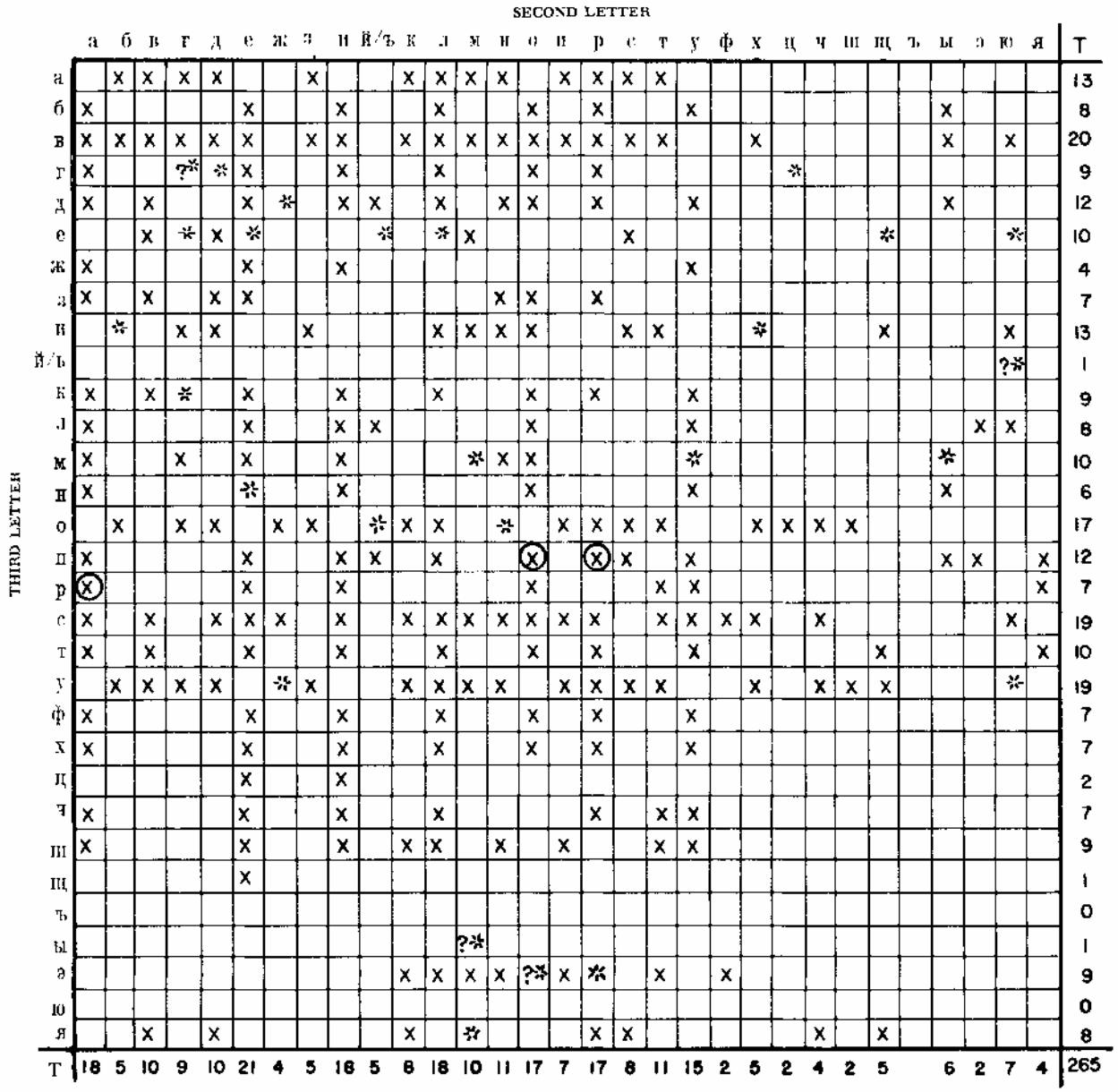
As is to be expected, the corresponding proportions are even higher for limited fourth letter possibilities after combinations of first three letters, as can be seen from Tables IV through X, which show the numbers of possibilities for second, third, and fourth letters after certain initial letters.

It will conserve time as well as space if the system is designed so that in the look-up process, beginning with the third letter, a test is made to determine whether to continue to another letter table or to proceed to the next stage of the process, in which one of the few headings remaining as possibilities can be selected. This test can be made possible by the use of the sign bit of each word of the letter tables, a minus indicating that the time has come to go on to the next stage. This minus sign would have to apply to both table entries of the machine word concerned, so it would not be placed in the word if for

[15] A. Koutsoudas and A. Halpin, Russian Physics Vocabulary, with Frequency Count: Left-to-Right Alphabetization, *Research in Machine Translation:* II, Vol. 1, Willow Run Laboratories, 1958.

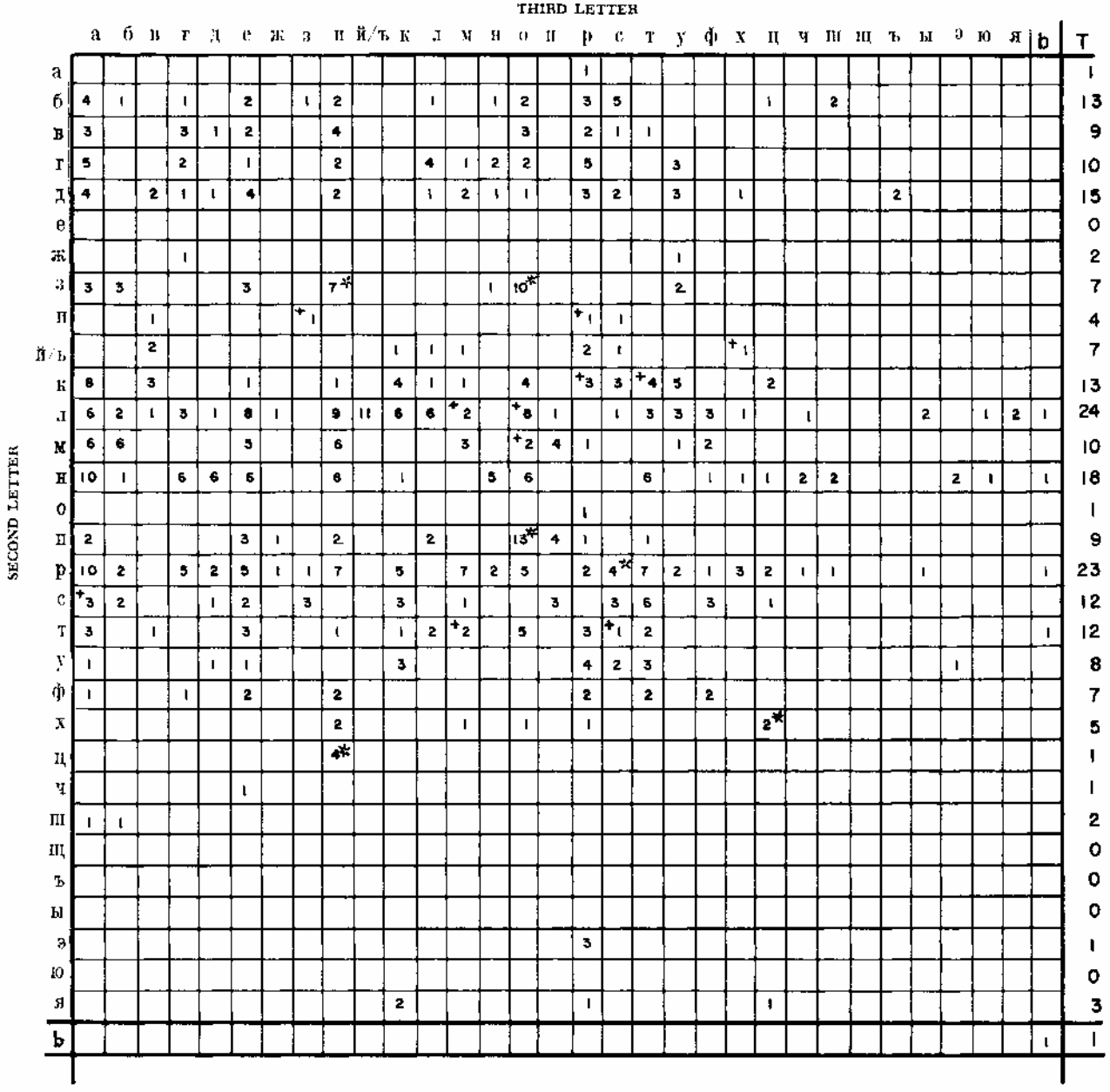
[16] Op. cit., p. 1.

TABLE II. COMBINATIONS OF FIRST TWO LETTERS (BASED ON KOUTSOUDAS AND HALPIN)



° only in words of 3 lettters or less
X extremely frequent
T number of possible second letters

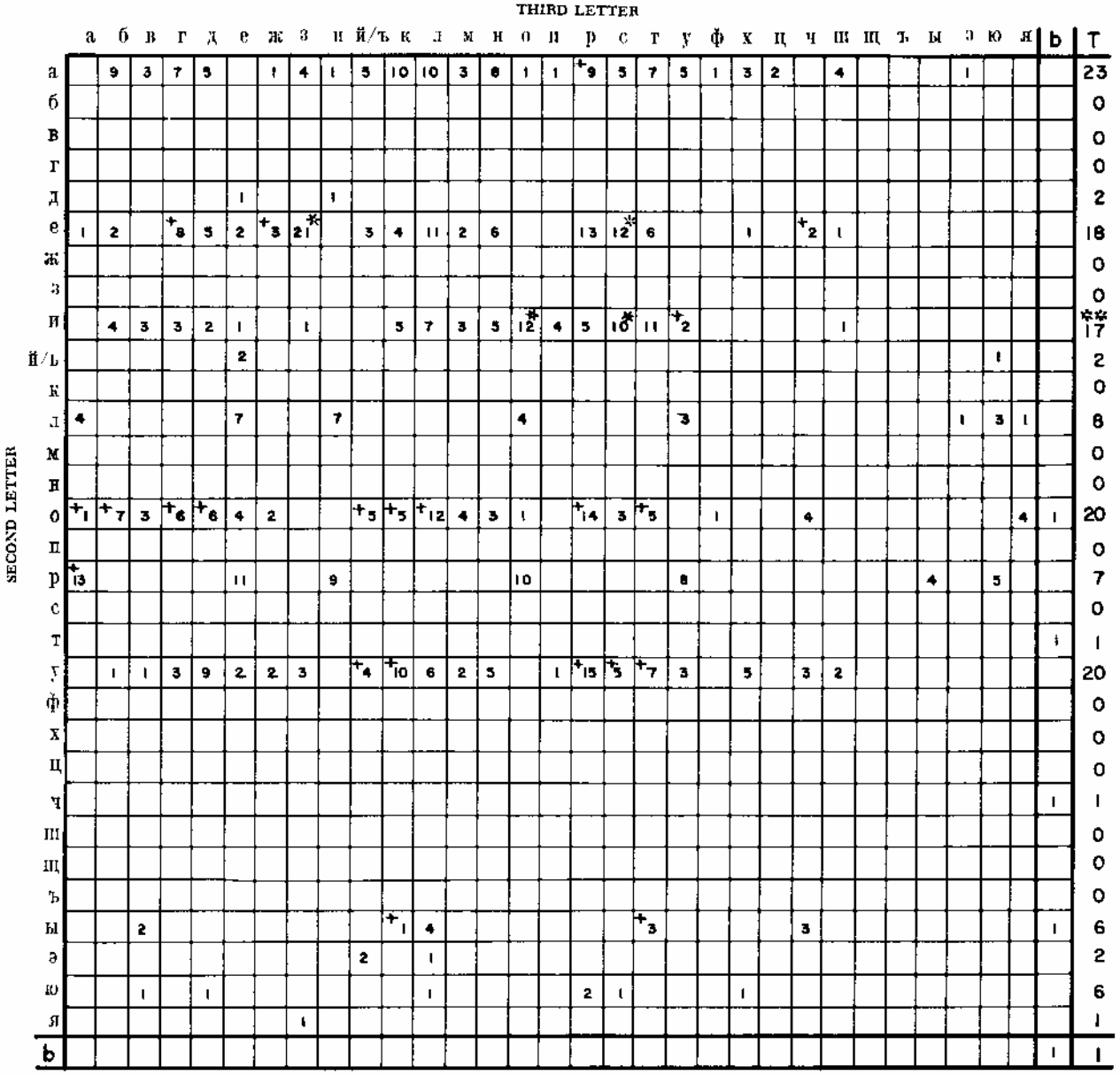
TABLE IV. COMBINATIONS OF FIRST THREE AND FIRST FOUR LETTERS (BASED ON CALLAHAM)
FIRST LETTER A



+ one of the possible fourth characters is a blank
– one of the possible fourth characters is a period
* a prefix of three letters
b position blank
T number of possible third letters

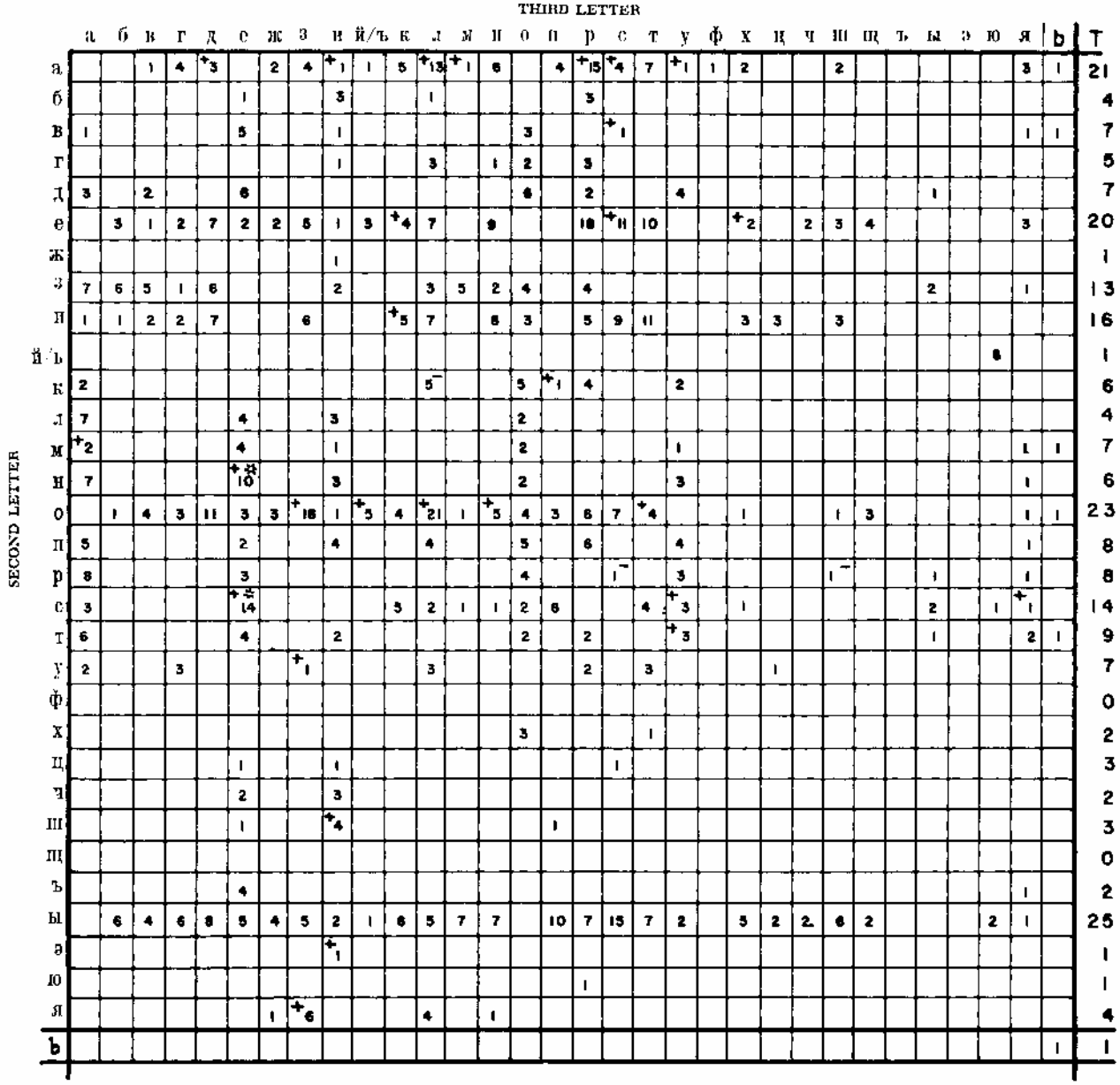
TABLE V. COMBINATIONS OF FIRST THREE AND FIRST FOUR LETTERS (BASED ON CALLAHAM)

FIRST LETTER ъ



+ one of the possible fourth characters is a blank
— one of the possible fourth characters is a period
* a prefix of three letters
b position blank
T number of possible third letters
** a prefix of two letters not included in T

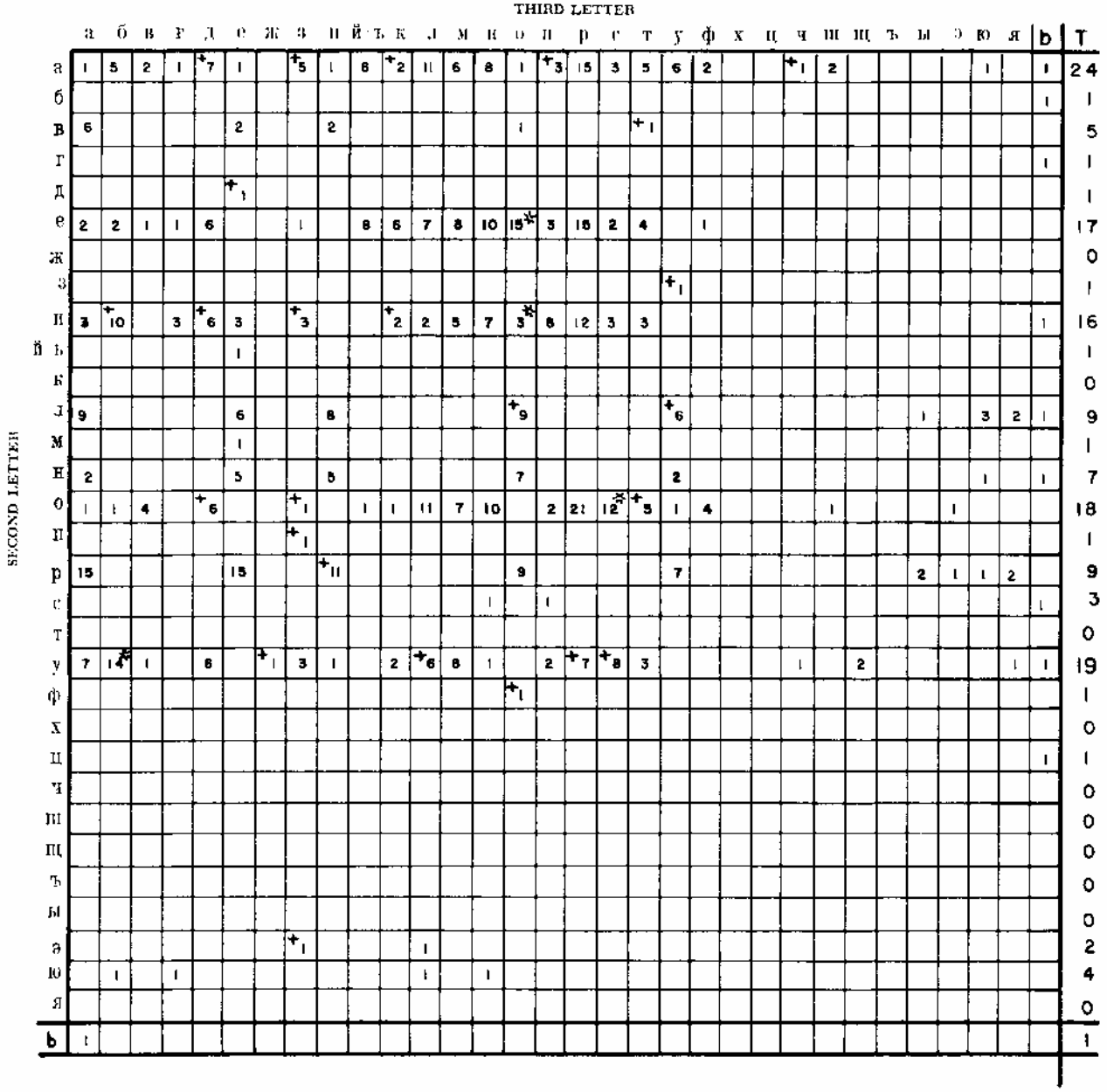
TABLE VI. COMBINATIONS OF FIRST THREE AND FIRST FOUR LETTERS (BASED ON CALLAHAM)
FIRST LETTER В



\+ one of the possible fourth characters is a blank
— one of the possible fourth characters is a period
* a prefix of three letters
b position blank
T number of possible third letters

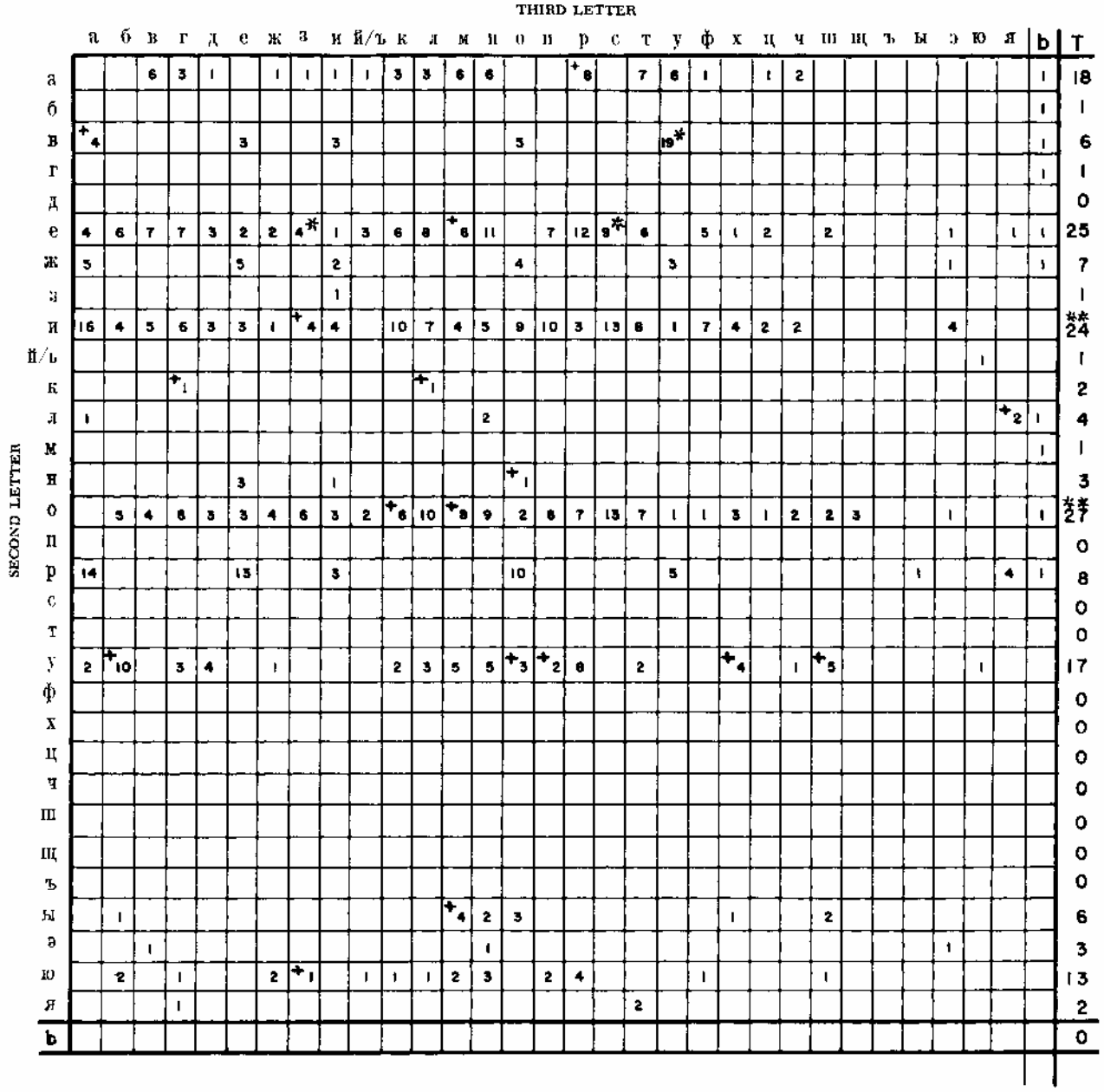
TABLE VII. COMBINATIONS OF FIRST THREE AND FIRST FOUR LETTERS (BASED ON CALLAHAM)

FIRST LETTER Г



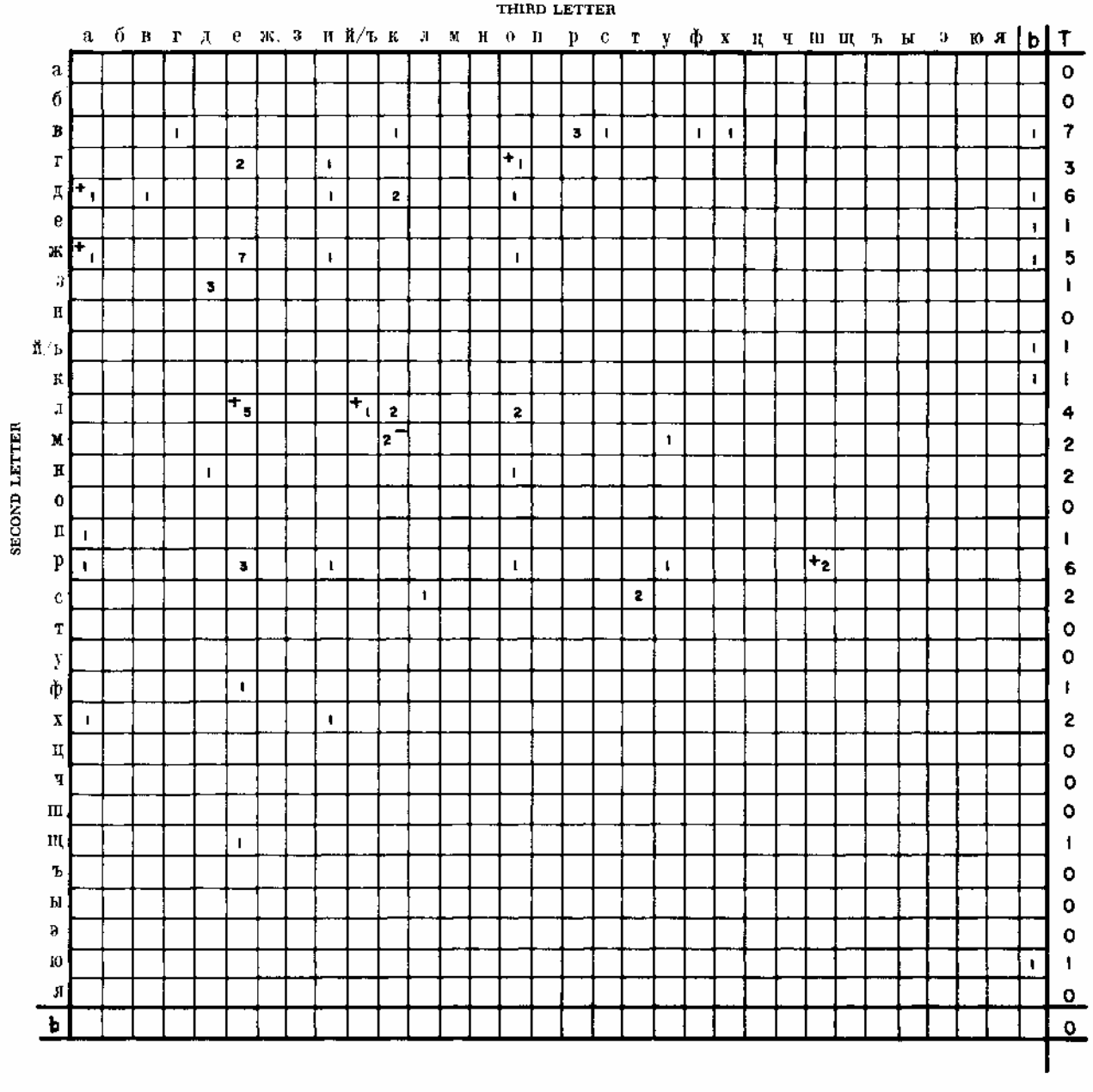
\+ one of the possible fourth characters is a blank
— one of the possible fourth characters is a period
* a prefix of three letters
b position blank
T number of possible third letters

TABLE VIII. COMBINATIONS OF FIRST THREE AND FIRST FOUR LETTERS (BASED ON CALLAHAM)
FIRST LETTER Д



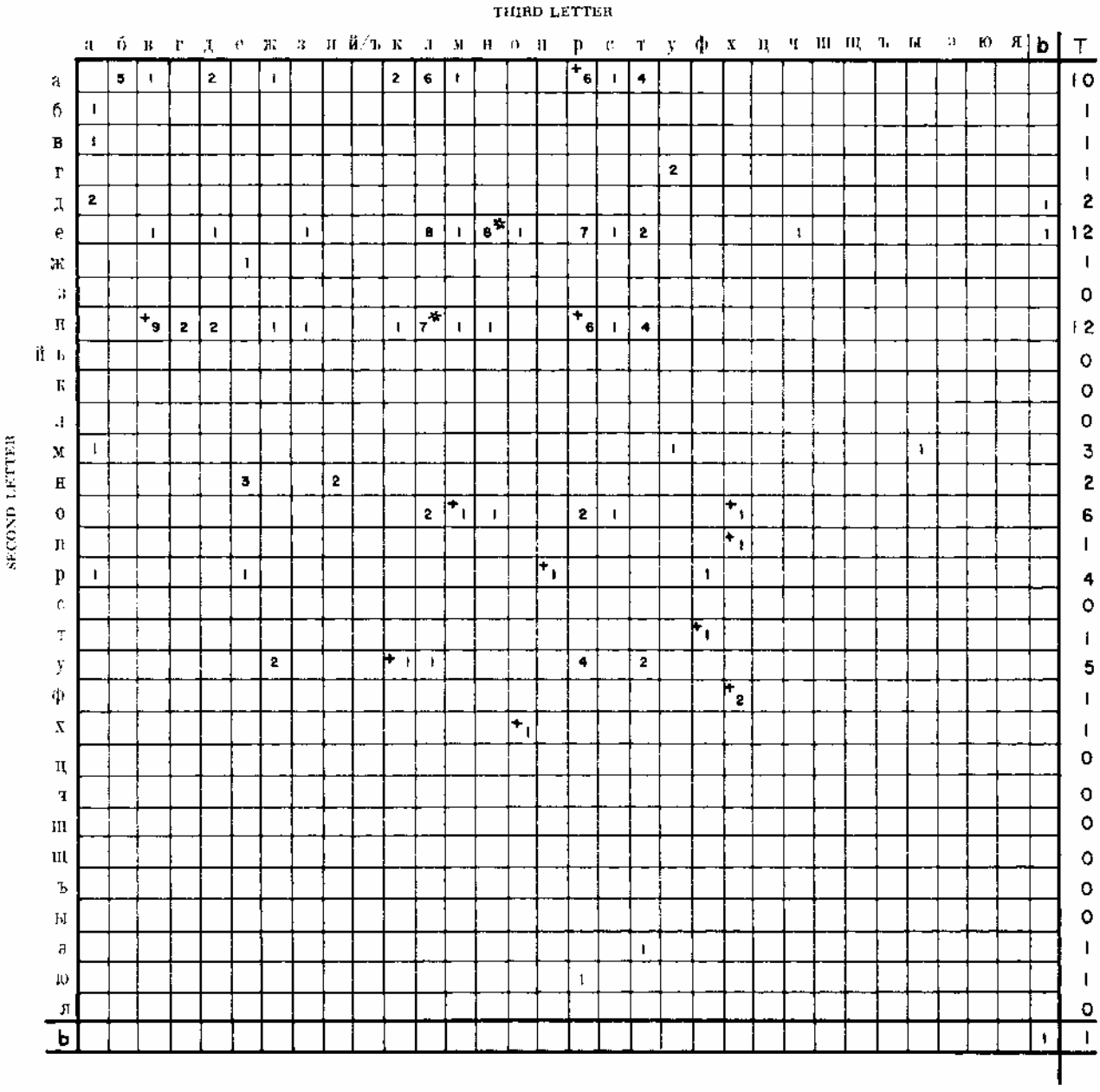
+ one of the possible fourth characters is a blank
— one of the possible fourth characters is a period
* a prefix of three letters
b position blank
T number of possible third letters
** a prefix of two letters not included in T

TABLE IX. COMBINATIONS OF FIRST THREE AND FIRST FOUR LETTERS (BASED ON CALLAHAM)
FIRST LETTER E



\+ one of the possible fourth characters is a blank
— one of the possible fourth characters is a period
* a prefix of three letters
b position blank
T number of possible third letters

TABLE X. COMBINATIONS OF FIRST THREE AND FIRST FOUR LETTERS (BASED ON GALLAHAM)
FIRST LETTER Ж



+ one of the possible fourth characters is a blank
— one of the possible fourth characters is a period
b position blank
T number of possible third letters

one of the pair it were desirable to go on to another letter table. The incidence of conflicts ought to be quite low, as may be determined from a study of Tables IV through X, In most cases where one of the pair is ready for the next stage, the other entry will be empty. Addresses would be placed in the address and decrement fields as before, but now they would be addresses for lists of "truncated headings" (see below).

The incorporation of this feature into the look-up routine requires only a slight additional modification.

LOOK-UP ROUTINE WITH PROVISION FOR PASSING TO NEXT STAGE AFTER POSSIBILITIES HAVE BEEN NARROWED DOWN

```
LDQ   VSTGD       Load vestigand into MQ.
PXD               Clear accumulator.
LGL   6           Put first letter into accumulator.
PAX   , 1             then into index register 1.
CLA   FIRST, 1    End of table for second letter
PAX   , 2             placed into index register 2.
PXD               Clear accumulator.
LGL   6           Put second letter into accumulator,
PAX   , 1             then into index register 1.
CLA   CONV, 1     Code conversion.
TMI   NOLTR       Test for non-alphabetic symbol.
PAX   , 1         Get correct pair
CLA   , 3             of table entries.
TXL   *+4,1,8192  If test bit is 0, transfer ahead.
PDX   ,2          Address for table or list into index.
TMI   TLIST       Transfer to next stage if minus.
TRA   * + 3       Otherwise continue to third letter.
PAX   ,2          Address tor table or list into index.
TMI   TLIST       Transfer to next stage if minus.
PXD               Clear accumulator for third letter.
etc.
```

The Truncate Lists

After the stage of letter-by-letter addressing has been completed (i.e., when the transfer on minus to TLIST is taken), we have what may be called the *truncated vestigand,* all or part of which will have to be matched with some truncated heading, or *truncate.* It is estimated that a typical system will have some three or four thousand *truncate lists* containing on the average five or six truncates each. In each of these lists the truncates will be portions of headings all of which begin with the same first three or four letters or so. The truncates of each list can be listed in order of length, from longest to shortest, and in reverse alphabetical (i.e., numerical) order wherever two or more have the same length. The look-up routine at this stage involves simply going through the truncate list from the beginning to get a match, either with the entire truncated vestigand or the first few letters thereof. In the latter case the remainder is to be looked up in the suffix tables.

It is necessary to mark a boundary between adjacent truncate lists, and this can be done by placing a minus in the sign bit of the first machine word of each. Five bits are needed for the segmentation-checking code, whose use is explained below (see *Segmentation).* For the sake of programming efficiency, these five bits are best placed in bit positions 31-35 (i.e., at the right end of the machine word). This leaves positions 1-30 for the truncate itself, thus providing for five BCD characters. For those truncates which are longer than five characters, the following cell (or two) may be used and, because of programming details which need not be discussed here, confusion (on the part of the look-up routine) between such supplementary cells and ordinary (or initial) truncate cells is best avoided by the placement of six ones in bit positions 1 through 6 of each supplementary machine word. Thus there is room for only four characters in each supplementary word. Nevertheless, if an effective segmentation system is used, the number of headings for which a second supplementary word is needed is very small. (For the dictionary being compiled at the University of California, a preliminary survey indicates that two supplementary words will be required by less than one per cent of the truncates.) For truncates of fewer than five characters, the right-hand portion of the available space should be filled, leaving vacant space at the left.

If, upon comparison, a truncated vestigand is found to be numerically smaller than a given truncate (except the first one in the list, which has a minus sign), comparison can immediately be made with the following truncate. If, on the other hand, it is numerically larger, it is immediately obvious, as it were, that it cannot be matched with any truncate in the list and must therefore be shortened by at least one letter before further comparison can hope to be fruitful. The need for shortening truncated vestigands under such circumstances can be reduced if it is known beforehand how long the longest truncate in a list is. Such information can be provided as a part of the entry in the letter table which sends the routine to the truncate lists. There are two bits available (1-2 for the left-half entry, 19-20 for the right one) for this purpose, providing for four length categories: (1) less than three letters, (2) three letters, (3) four letters, (4) five or more letters. The truncate-length categories are denoted by L in Fig. 4 (below), which illustrates a typical letter table.

It is not necessary to provide space for the storage of intermediate addresses of headings located during this stage, since their intermediate addresses can be identical with those of the cells where their truncates (or final portions thereof) are stored.

Segmentation

Much of the power of the system described here resides in the simple means it provides for segmenting words into ideal units for purposes of translation. This ability to segment effectively not only promotes efficiency in the translation routines; it also enables the automatic translator to deal with most neologisms and, by the same token, allows it to accommodate a vocabulary of hundreds of thousands of graphemic words, even though there are only twenty thousand dictionary entries.

Operational segmentation of words by the machine program can be effective, in the sense that it can follow the same principles of segmentation that would be used in a structural description,[17] if the program is so constructed that it takes the longest heading contained in the vestigand (beginning at the left) as the first lex, the longest heading contained in the *remainder* (if any) as the next, etc.; provided that the resulting tentative segmentation yields lexes whose co-occurrence in the order found is allowable. The proviso makes it necessary that *segmentation codes* for all headings be present at the time of look-up. The codes can be used to test the compatibility of provisional segments, and such testing must include a check to determine whether the final (or only) provisional segment can occur without a following lex. The first lex of a polylexemic vestigand will be either a base or a prefix. If it is the former, then the suffix tables will be used in containing the look-up process; if it is a prefix, the main part of the look-up system will be used. However, if the initial segment is a base and no provisional segmentation checks out using the suffix tables for the remainder, the word could be a compound and the remainder can be looked up in the main part of the system. For Russian, this roundabout treatment of compounds can be avoided for the most part by including known and/or frequently-occurring compounds in the dictionary as unit lexes. The roundabout procedure would then be used only for infrequent and/or neologistic compounds. On the other hand, for languages in which compounding is a highly productive process, like German, such treatment is undesirable since it would make the dictionary too bulky. Overall efficiency might be maximized for German by including suffixes in the main part of the dictionary, so that all remainders could be looked up in the same manner.

For those headings which end in the truncate lists (rather than the letter tables) the requirement that the longest contained heading be chosen is built into the system as an automatic feature, since the truncates are listed in reverse order of length (i.e., from longest to shortest). If segmentation checking fails to yield a satisfactory result, consideration can pass immediately to the following truncate in the list.

On the other hand, many headings are short enough to come to an end while the look-up process is still in the letter tables. Of these, some are included within longer headings while others are not. The latter arc automatically provided for in the program by a feature to be described below. In the case of the former, the look-up routine will need to know, as it were, whether one of the longer headings is also contained in the vestigand, so it will have to continue the look-up process, usually by going to a truncate list. If it does not find a longer contained heading, however, it will want to return to the shorter one. Provision for such return can be furnished by keeping track of the segmentation code and intermediate address of each such heading as the look-up routine proceeds. For each vestigand then (except those not composed of letters), we will want to make a short *segmentation checking list.* Two pairs of alternatives confront us with regard to what should be stored in this list. On the one hand, we must decide whether to store there the contents of the words that contain the segmentation checking codes for the various lexes or merely the addresses of these words. (Each of these words is the last of the sixteen words of its letter table. See Fig. 4.) On the other hand, when passing to successive letter tables, we have to choose between storing the final word of each table whether or not it contains a segmentation checking code, and testing each such word as we come to it, storing only those that actually contain this information (i.e., those which represent ends of headings). It will often be the case that a cut after one of the longer headings contained in a vestigand will give the correct segmentation and thus render unnecessary the testing of the remaining possible shorter headings. The best policy, therefore, would seem to be to postpone as many as possible of the operations connected with testing a given segmentation until such time as the need for that test arises, in the expectation that these operations will often not have to be performed at all. Following this policy will lead us to store the addresses of the table-final words without testing their contents, as this is the procedure that will take the fewest operations.[18] With reference to the routine given above we need add only an SXD (store index in decrement) operation to store the contents of index register 2 when the (2's complements of the) addresses for the third-, fourth-, and fifth-letter tables are there, since the address of the cell containing the segmentation code and intermediate address for a heading ending at the nth letter is the same as that of the (n + l)th-letter table for the sequence in question. We will not bother storing the second-letter table address since it is very easy to obtain this again (by repeating the first three operations of the look-up routine) in those infrequent instances when it is necessary to do so.

The case of headings which come to an end while the process is in the letter tables and which are not included within longer headings is easier to deal with. If the vestigand comes to an end at the same point, it coincides with the heading and the look-up process is completed. On the other hand, if the vestigand is longer, a cut has to be made, and the design of the look-up system is such that this situation can be dealt with automatically, as it were, i.e. without the need of inserting extra checking operations into the routine which would slow down

[17] Except with regard to the degree of segmentation. While the ultimate constituents on the morphemic level for a structural description are the morphemes, segmentation for a translation system should stop short of this point. It is not efficient to segment, with regard to a given construction, if the target representations of the constitutes cannot economically be treated as combinations of the representations of the constituents. In addition, segmentation of individual forms should generally be avoided whenever the cut would necessitate the setting up of allolexes that would otherwise be unnecessary.

[18] Thus in interpreting Fig. 4 (below), one should bear in mind that the machine will go through the same operations in cases 1 and 3, which differ only in the storing or not storing of a segmentation code; whether or not a code is stored will depend only on whether or not a code has been entered in the word that is stored.

FIG. 3. STATUS OF TABLES AND ACTION TAKEN UNDER EACH POSSIBLE SITUATION.

nth letter			$(n+1)$th letter	Situation	
I. is not: II. is:	A. is: B. is not:	1. is: 2. is not:	a. is: b. is not:		
last alphabetic character in word	in a sequence entered in dictionary	final letter of a heading	in a sequence entered in dictionary		
$(n+1)$th-letter is:	nth-letter table has:	$(n+1)$th-letter table has:	$(n+1)$th-letter table has:	Action taken	
I. alphabetic	A. address for $(n+1)$th-letter table	1. intermed. address and segmentation code	a. address for $(n+2)$th-letter table or list	Store segmentation code; go to table or list for $(n+2)$th letter	1
			b. no address for $(n+2)$th-letter table or list	Test remainder and check segmentation	2
		2. no intermed. address or segmentation code	a. address for $(n+2)$th-letter table or list	Go to table or list for $(n+2)$th letter	3
			b. no address for $(n+2)$th-letter table or list	Try shorter sequence (s)	4
	B. no address for $(n+1)$th-letter table			Try shorter sequence (s)	5
II. non-alphabetic	A. address for $(n+1)$th-letter table	1. intermed. address and segmentation code		Check whether sequence occurs without suffix	6
		2. no intermed. address or segmentation code		Try shorter sequence (s)	7
	B. no address for $(n+1)$th-letter table			Try shorter sequence (s)	8

the look-up process for all vestigands. For whenever such a situation is encountered, the entry in the letter table which corresponds to the next letter will be empty, reflecting lack of occurrence of the encountered sequence among the headings in the dictionary. An empty table entry, of course, will not be blank but will contain the (2's complement of the) address for the "no-such-heading" table. Then the machine, not yet aware that this situation is present, will proceed as usual. In the no-such-heading table, all sign bits will be minus, so that the transition as if to a truncate list will be launched into, at which time a test to reveal the nature of the situation can be made. Thus we need to test for this condition only once per vestigand (at the time the transition is made), rather than once per letter. On the other hand, if the partner of an empty table entry (i.e. the other member of the pair) represents a situation which is ready for the transition to a truncate list, then the transfer will be taken right away and it will not be necessary to go to the no-such-heading table.

Figure 3 shows the ways in which different combinations of conditions found in the machine lead to different subsequent actions as the *n*th letter is being looked up. At the head of the four left-hand columns are entered four alternative possibilities having to do with the letter sequence that ends with the nth letter and its relationship to the letter sequences that are entered in the dictionary. Within each column are found descriptions of the actual conditions in the text or letter tables that correspond to one or the other of the alternatives given at its head, each alternative and its corresponding condition being labeled by the same number or letter. In the column at the right are found the actions that are to be taken when the combination of conditions given in each horizontal row is encountered. A blank space in the table indicates that the particular question is irrelevant for determining the course of action, under the combination of conditions defined to its left.

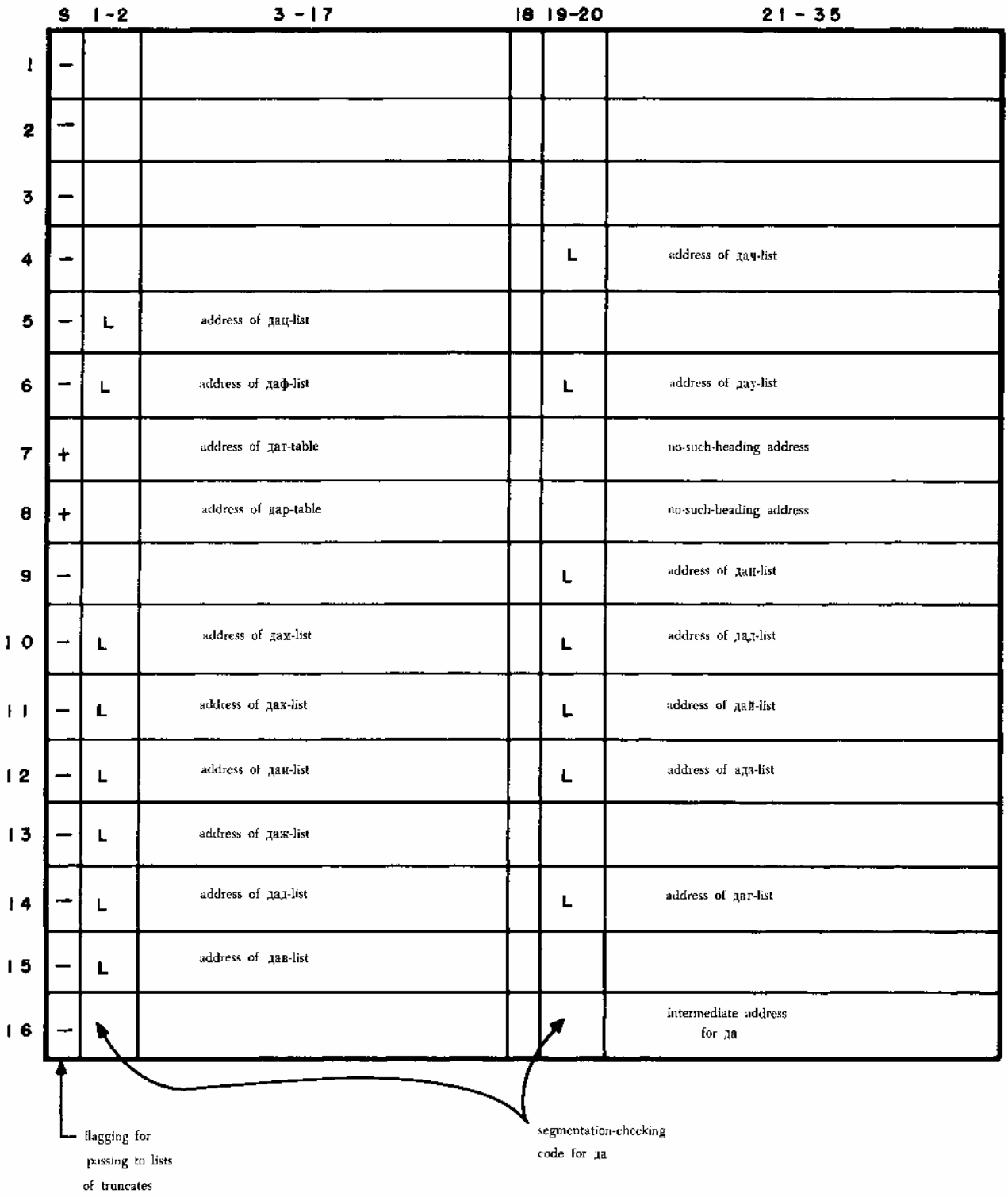
The descriptions of these actions which are entered in the chart must be amplified as follows. Testing the remainder involves determining whether or not it consists of a suffix or combination of suffixes entered in the dictionary. If a positive result is obtained, segmentation checking will reveal whether or not the segmentation codes of the stem and suffix(es) are compatible. When a test of the remainder and segmentation check, either for the longest sequence found to be entered in the dictionary or for shorter ones, is indicated as the action to be taken, and a correct segmentation is obtained, the machine will then store the intermediate addresses of the lexes involved. After every remainder test or segmentation check which yields negative results, there will be a looping back, not shown on the chart, to determine whether there are shorter sequences contained in the vestigand which are entered in the dictionary, and, if so, to test the resulting remainders. If there is no shorter sequence in such a case, the machine will provide a transliteration of the word, together with a mark indicating that it was not found in the dictionary. After the machine has either stored the intermediate addresses of the lexes or transliterated the vestigand, it will begin the look-up of the next word token.

Let us examine more closely the way in which the determination of the alternative situations is based on conditions found in the text and letter tables. In the first place, whether or not the *n*th letter is the last alphabetic character in the vestigand is shown by whether the (*n* + l)th character in the text is alphabetic or non-alphabetic. If it is a non-alphabetic character, it will be further tested to ascertain whether it is a punctuation mark, indicating the end of the text word, or some other character that should be transliterated, but in either case it will signal the end of the progression to successive look-up tables.

The relationship of the sequence being looked up to sequences entered in the dictionary is shown by entries or the lack of entries in the proper places in the look-up tables. If the sequence ending with the *n*th letter is part of a sequence which is entered in the dictionary, the (2's complement of the) address of the (last word of the) *(n* + l)th-letter table will be found at the place in the *n*th-letter table which corresponds to the *n*th letter; whereas if it is not part of an entered sequence, the no-such-heading address or zero will be found at that place. If there is a heading entered in the dictionary which is coterminous with the sequence ending with the *n*th letter, the intermediate address and segmentation-checking code for the lex formed by this sequence will be found in the last word of the (*n* + l)th-letter table. The absence from the dictionary of a sequence ending at this point will be shown by the absence of this address and code. Next, whether or not there is a longer sequence in the dictionary continuing with the (*n* + l)th letter is shown by whether or not there is entered at the proper place in this (*n* + l)th-letter table the address of a table or list for that sequence.

And lastly, the occurrence of any shorter entered sequences in the sequence through which we have passed is shown by their class codes which will have been stored in the segmentation checking list.

These relationships may be made more vivid by the consideration of some actual examples. Let us look first at Fig. 4, which shows the third-letter table for sequences beginning with да based on the letter combinations given in Table VIII. The sixteen numbered rows represent sixteen consecutive words of core storage. In the address portion of the sixteenth word is found the intermediate address for the lex да, and in the last two bits of the prefix and the three bits of the tag of this word is the five-bit segmentation code for this lex. For every occurring three-letter sequence beginning with да, the address of the corresponding fourth-letter table or list is entered in the address or decrement portion of a table word. The sign bit contains a minus in those words that do not have entered in them any address of a fourth-letter table. For this example it was decided that the change-over from tables to lists would be made when the number of different fourth



L : truncate length category

FIGURE 4. THIRD-LETTER TABLE FOR да

letters following the sequence in question was six or less,[19] which is found to be the case for all the sequences in the table except дар and дат. Therefore all the table words contain a minus except for the two words that contain the addresses for these two sequences. In the address portions of these two words is entered the address of the no-such-heading table. Such a no-such-heading address, or a blank address or decrement portion in a table word that has a minus in its sign bit, then, correlates with the non-occurrence of a given three-letter sequence. In the last two bits of either the prefix or tag which immediately precedes each list address is entered a number indicating the length category of the longest truncate in that list. Each of these is indicated by an L in the figure.

Note that the address in the decrement portion of a final word in a table must be one of a table rather than a truncate list, because the space taken up in this word by the segmentation-checking code precludes entering a truncate length category for a list. Therefore a minus in the sign bit of such a word can only indicate the absence of an address in its decrement portion. Wastage of space due to addressing certain sequences to letter tables for this reason when other criteria would indicate that their look-up should be continued in truncate lists can be minimized by assigning to this place in the letter tables a Russian letter which turns up infrequently in the first few letters of Russian words, such as the "hard sign."

Suppose, now, that we are looking up the sequence дар. The proper place in the second-letter table for д will have yielded the address of our sample table. The intermediate address and segmentation-checking code in the sixteenth word of this table show that the dictionary contains a complete lex having the form да. The address of the fourth-letter table for дар which is entered in the decrement portion of the eighth word of this table indicates that the sequence дар is found in the dictionary. So the machine will store the segmentation code for да and pass to the table for the next letter. This will be an instance of type 1 of Fig. 3.

On the other hand, if the sequence дас is being looked up, the machine will encounter the no-such-heading address in the address portion of the seventh word of our sample table. After passing to the no-such-heading table, the minus in the sign bit of a word of this table will signal the machine to make a test and discover that no such sequence is entered in the dictionary. The machine will therefore proceed to check whether the remainder of the word (beginning with с) is a possible suffix or combination of suffixes, and if so, whether the lex да may occur with this remainder. If the segmentation is not found to be correct, the word will be transliterated, since there is no shorter lex in this sequence which is entered in the dictionary. This is a situation of the second type.

[19] In actual practice, however, the point at which this change-over is made will depend on the number of different lexes beginning with a given sequence, rather than on the number of different letters immediately following that sequence.

If, again, the sequence that is being looked up is да# (that is да followed by a space), the space in the text will show that there are no more letters in the word, so the machine will use the segmentation-checking code in the sixteenth word of our sample table to check whether the lex да may occur without a suffix. This will be found to be possible, so the machine will store the intermediate address for fla and pass to the next word. This will fall under type 6 on our chart.

One more example, this time of a letter sequence that will not be looked up by means of our sample table, should suffice to clarify these sequences of operations. If the sequence encountered in a text is вем, there will be found a third-letter table for ве in the dictionary. This table, however, will not contain an intermediate address or segmentation-checking code in its last word, since there is no lex having the form ве, nor will it contain an address to a fourth-letter table or list for вем, since such a sequence will not be entered. After ascertaining the absence of these entries in the ве-table, the machine will find the code for the lex в. A check will be made of whether this lex may occur with the remainder. This will not be possible, so the word will be transliterated. This is an instance of our type 4.

It may be helpful to have examples of three-letter sequences that will fall into the various categories of our chart as the second letter is being looked up. For each category two such sequences are given, one beginning with в, and one with д. Since в is a one-letter lex, while д is not, the sequences beginning with the former letter will contain a shorter sequence whose segmentation may be checked if this becomes necessary, while those beginning with the latter letter will not. The differences between the categories should be clear if it is remembered that во and да are two-letter lexes, while the other two-letter sequences (ве, вф, дф, дн) are not. (See also tables VI and VIII). The examples, then, are: 1, вот, дар; 2, воф, дас; 3, век, дне; 4, вем, днс; 5, вфа, дфа; 6, во#, да#; 7, ве#, дн#; 8, вф#, дф#.

Segmentation Checking

The fact that a provisional segmentation yields a stem present in the dictionary and a suffix also present does not necessarily mean that the vestigand has been correctly segmented. It is necessary to check whether the provisional suffix can occur with the potential stem from which it has been separated, since the provisional segmentation could be a false one for either of two reasons: (1) the vestigand or one of its constituent lexes is absent from the dictionary and it happens to lend itself to a spurious segmentation; (2) the real base is shorter than the one provisionally selected.

As an example of the first situation, suppose that the form ранет 'rennet (a type of apple)' has not found its way into our dictionary, but turns up as a vestigand. Without segmentation checking, it would be identified as consisting of the verb stem ран 'to injure' (also a noun stem meaning 'wound') plus the third person singular

suffix -ет. Segmentation checking can identify such a segmentation as spurious, since ран belongs to that class of verb stems for which the third sg. suffix has the allomorph -ит rather than -ет. With segmentation checking, then, the machine may be made aware, as it were, of the fact that ранет is absent from the dictionary and it can print out a transliteration, together with a mark indicating the absence from the dictionary of the form. The reason for the need to check the segmentation during the look-up process when this type of situation occurs is that it is desirable for the machine to dispense with the Russian graphemic forms after they have been looked up; but any transliteration of forms absent from the dictionary must be done before they are discarded.

The second type of situation, in which the real base is shorter than the one provisionally selected, may be illustrated by the form позволят. The longest contained heading would be позволя 'to permit/allow (imperfective)', leaving as the provisional suffix -т, an allomorph of the past passive participial suffix. Segmentation checking will reveal that these two lexes cannot occur with each other. Thus the next longest contained heading, позвол, 'to permit/allow (perfective)' will be tried, and since the suffix, -ят 'third person plural non-past', will be shown to be compatible with the stem, this segmentation will be selected as the correct one.

The checking can be accomplished by means of a table in core storage which can be thought of as a matrix in which the rows represent suffix classes (most of which will contain a single member), the columns base classes distinguished on the basis of occurrence with the suffixes. Each of the elements of the matrix will consist of a single bit with the value zero or one depending upon whether or not the combination represented is allowable.

According to the design of the system described above, 5 bits are allowed in each machine word of the truncate lists and in each final word of the letter tables for the segmentation code. One of the 32 possible combinations (decimal 31) is needed for long truncates as an indication that the truncate continues in the following word (see above, under *The Truncate Lists)*. Thus we are allowed 31 different segmentation codes, probably enough for all practical purposes, even though this amount is clearly insufficient to reflect all the details of an exhaustive classification.[20] Using this number of segmentation codes, each row of the matrix can be stored in a single cell. The presence of one or zero in the appropriate position for a given instance can be determined by a transfer on zero following an ANA ("logical and" to accumulator) operation, using one of the 31 masks corresponding to the 31 segmentation codes. (Each of the masks must have a one in one of 31 bit positions, all the rest zeros.)

Quasi-Prefixes

The system as described so far can handle dictionary look-up with great efficiency for a dictionary of about 20,000 entries, and can probably handle up to 21 or 22 thousand entries without much difficulty. If it should ever be desirable to have more entries than this in the dictionary, an appreciable amount of memory space to provide for additional ones, with only a small increase in look-up time can be made available by a device which involves "quasi-segmentation" of "quasi-prefixes." By the term *quasi-prefix* is meant a sequence of graphemes occurring initially in words which in at least some of its occurrences represents a frequently occurring prefix in the source language as analyzed in isolation but not necessarily as analyzed for an MT system.[21] Examples for Russian as source language, where the target language is non-Slavic, would be по-, паз-, при-. If analyzing Russian itself (apart from other languages), a linguist would set up prefixes having these graphemic representations, but they would not be regarded as prefixes in a Russian-to-English MT system since in most of their occurrences the English representations of the composite forms in which they occur could not economically be treated as composites of English representations of the constituents. Moreover, a quasi-prefix is considered to be present in *all* words beginning in the appropriate letter sequence, even for those in which the prefix (of the source language as analyzed in isolation) is not present, e.g. Russian полно 'full' (for по-). Thus a quasi-prefix is not a separate lex and has no separate dictionary entry (except in special cases in which separate treatment is really desirable) but *part* of a lex, whereas a *real* (MT) prefix is a separate lex, and is to be segmented only where it actually occurs (but not where a homographic letter sequence occurs).

The way quasi-segmentation works is quite simple: whenever the machine encounters a quasi-prefix as it is addressing the letter-tables, it goes back to the first-letter table for the next letter. Thus the letter following the quasi-prefix is treated as if it were an initial letter, so that the same letter-tables actually serve multiple situations. A considerable amount of duplication of letter-tables is thus avoided. Naturally, the machine must keep in mind as it were which quasi-prefix was encountered, if any, as it will need this information later on in order to choose the correct intermediate address.

From the foregoing it will be clear that it is desirable to set up quasi-prefixes for those cases in which the range of possibilities for following letter sequences approximates that of beginning letter sequences, and only for those cases. This principle dictates, among other

[20] Minor classes which, if included, would bring the number to more than 31, can be dealt with in one or more of three ways. For a given deviant base we can either (1) refrain from segmenting and enter the composite form as a heading; or (2) assign the deviant base to a class of slightly wider distribution, after ascertaining that no false segmentation is likely to result. But if for some use of the system these two devices prove to be inadequate and it becomes desirable to use more than 31 segmentation codes, (3) additional codes can be assigned to deviant bases and they can be placed in supplementary machine words in the truncate lists. Code 31 would then mean either that the truncate continues in the following cell or that the base is of a deviant type whose code is given in the following cell.

[21] Cf. the "pseudo-prefixes" of Ida Rhodes, described by her in A New Approach to the Syntactic Analysis of Russian, elsewhere in this issue.

things, that only those prefixes (of the source language as analyzed in isolation) which have the widest occurrence in the lexicon be set up as quasi-prefixes.

The effect of quasi-segmentation is that words containing quasi-prefixes are looked up not with regard to their beginnings, which are not very distinctive, but with regard to what follows the relatively non-distinctive portion. The process of narrowing down is thus rendered much more effective.

Provision for quasi-segmentation in the letter-by-letter addressing system can be made by means of information placed in the appropriate entries of the letter tables involved, in lieu of an address for a next letter table or a truncate list. In the case of no-, for example, the o-entry in the second-letter table for п would have the address for the "quasi-prefix table" plus a particular number, less than 16, assigned to по-. The quasi-prefix table would function like the no-such-heading table (see above), providing a means for identifying the nature of the situation without the need of checking operations at every step. Like all letter table addresses, the quasi-prefix address would be a multiple of 16.[22] Thus the significant part of the quasi-prefix address would occupy bit positions other than the four low-order ones, while the number identifying the individual quasi-prefix involved would occupy these four low-order positions. (It is assumed that there would be no more than 16 quasi-prefixes.)

When it has been determined by the machine (in the manner described above) that either the no-such-heading table or the quasi-prefix table has been entered, the index register will still have the information defining the nature of the situation. A TXH (transfer on index high) or TXL (transfer on index low or equal) can serve to distinguish between the no-such-heading and the quasi-prefix situations and, if it is the latter, subtraction of the quasi-prefix address from the contents of the index register will leave the number indicating which quasi-prefix is involved.

Allocation of Memory Space

An estimate of the way in which memory space might be allocated within core storage for the look-up process is summarized in Table XI. The amount of space assigned to each of the parts of the system is estimated to the nearest hundred machine words. The number of letter tables in each category is estimated to the nearest ten. (The figure 30 is given for first- and second-letter tables combined; there are 28 second-letter tables needed for Russian, but the first-letter table has 64 words instead of 16, so 32 would be the precise figure to us as a multiple of 16 in calculating space needs for the tables.)

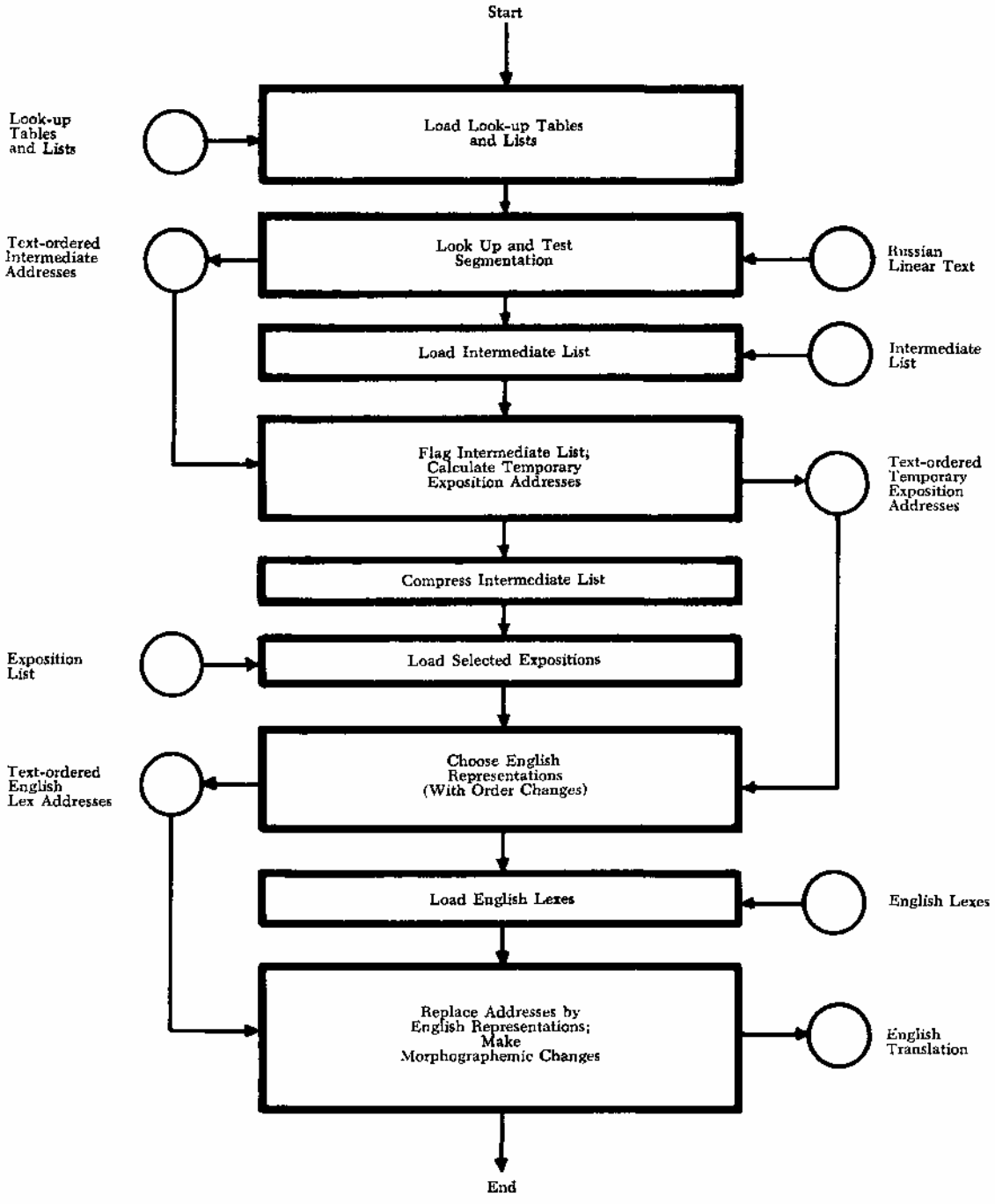
The estimate has been made for a dictionary of 20,000 entries, with the headings having lengths as calculated for the dictionary being compiled at the University of California. It is assumed for the estimate that quasi-segmentation is not used in the system. A dictionary of more than 20,000 entries could be accommodated by cutting down on the number of third-, fourth-, and fifth-letter tables (thus increasing the average length of the truncate lists) and/or by resorting to quasi-segmentation. Either of these measures would slow down the look-up process to a slight extent. By the same token, with a dictionary of fewer than 20,000 entries, more letter tables, especially for the third letter, could be added, thus reducing the average size of the truncate lists and increasing by a slight amount the speed of the look-up process. The extent to which speed could be increased by adding more letter tables is quite limited, however, since a minor difference in the average length of truncate lists (for example a reduction from five to three or four truncates) makes very little difference in the amount of time required to select desired truncates. Therefore, for a much smaller dictionary a more profitable avenue to explore with a view to increasing overall speed would lead to including equipment for the intermediate stage in core storage at the same time as the look-up proper.

TABLE XI: ALLOCATION OF SPACE IN RAPID-ACCESS MEMORY FOR THE LOOK-UP PROCESS, ESTIMATED FOR RUSSIAN DICTIONARY OF 20,000 ENTRIES.

	Number of tables	Number of machine words	
Letter Tables (for bases and prefixes)			
1st and 2nd Letters	30		
3rd Letter	180		
4th Letter	220		
5th Letter	20		
Total	450 times 16		7,200
Truncate Lists			
First machine word for each truncate:		18,600	
First supplementary machine word:		3,600	
Additional supplementary word:		200	
			22,400
			29,600
Suffix Tables (for about 160 suffixes)			400
Provision for Non-Alphabetic Characters			100
Segmentation-Checking Table			100
Program			1,400
Input/Output Buffer			1,200
Total (32,768)			32,800

[22] Cf. footnote 12. (It would he possible to let the quasi-prefix address coincide with the no-such-heading address. In this case, 15 quasi-prefixes could be provided for and the 16th possibility would indicate the true no-such-heading situation.)

FIGURE 5. THE TRANSLATION PROCESS.



The Intermediate Stage

After the look-up and segmentation checking stages, we will have recorded on tape a list of addresses, one for each lex token of the original text, arranged in the same order as that in which the lexes occur in the text. Since the segmentation system used at the University of California gives about twice as many lex tokens as word tokens in a text, an initial text of 30,000 words would now be represented by a list of about 60,000 addresses.

It may be helpful in following the explanations in this section to refer to Fig. 5, which is a block diagram of the translation process, showing the points at which information is transferred between tapes and core storage. Our discussion so far has been concerned with the operations in the second block, and the operations pertaining to the intermediate stage are those in the next four blocks. The remaining operations have to do with translation proper rather than dictionary utilization, and are not considered in detail in this paper.

Our problem now is to bring into core storage, from the tape containing the actual expositions, the entries corresponding to the lexes which occur in the text being processed, and to replace each text-ordered address by the address to the location that the corresponding exposition will occupy in core storage. This is done by means of an *intermediate list.* The addresses which we have on tape are addresses to this list, and each one may therefore be called an *intermediate address.* Note that the initial stages of look-up give us neither the expositions themselves nor any part thereof (except for the information used in segmentation checking), nor do they yield the addresses where the expositions are stored at that point.

The format of the intermediate list, which is stored on tape in one file, is as follows. It consists of a series of about 20,000 machine words, one for each lex that is to be recognized in the look-up process; each intermediate address is therefore the address of the corresponding word in the intermediate list, and this word may be called the *intermediate word* for its heading. The arrangement of an intermediate word is shown in Fig. 6.

In the form in which it is stored on tape, each intermediate word contains four pieces of information, which may be grouped into three categories.

(1) The sign bit is used for flagging, by storing a minus there, the first time each lex occurs in the text being translated.

(2) The remaining pieces of information refer to the actual exposition represented by the intermediate word. The expositions are stored in sequence on another tape and together constitute the *exposition list.* The next two pieces of information serve to address the individual entry in the exposition list.

(a) The tag (bits 18-20) contains the *list portion number.* Since the exposition list is too long to be held in core storage at one time, it must be divided into portions which will be read in separately for the purpose of selecting out the entries needed for a particular text. The list portion number indicates the portion of the exposition list in which the related exposition is located. The three bits allocated for this number allow, of course, for the exposition list to be divided into eight portions.

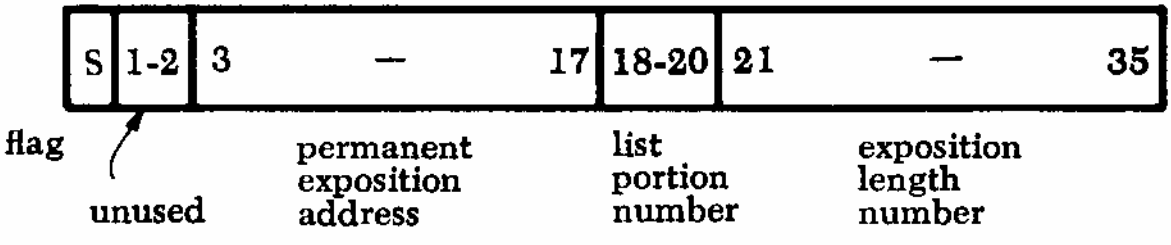
(b) The decrement (bits 3-17) contains the *permanent exposition address,* which is the address to the place that the related exposition will occupy after the portion of the exposition list containing that exposition has been read into core storage. For ease in programming, this address will be the 2's complement of the address in core storage of the first word of the exposition.

(3) The remaining piece of information refers to the length of the corresponding exposition. The address portion (bits 21-35) contains the *exposition length number,* which is equal to the number of machine words in the exposition. Each exposition may be of any length; no space is wasted by requiring standard lengths of expositions.

During the intermediate stage the exposition length number will be replaced by the *temporary exposition address* for occurring lexes. This is the address at which the exposition is to be placed in core storage for the stage of translation proper. Like the permanent exposition address, it will be the 2's complement of the address of the first word of the exposition.

The intermediate list is used in the following manner. It is first read into core storage so that it is located in a block of consecutive words. Then the tape of text-ordered intermediate addresses is read in, a portion at a time, and operations are performed to give the following results: a. The intermediate word corresponding to

FIGURE 6. AN INTERMEDIATE WORD.



each different lex in the text is flagged with a minus in its sign bit. b. The temporary exposition address is calculated for each different lex in the text. c. This address is stored in the address portion of each corresponding intermediate word, replacing the exposition length number. d. Each text-ordered intermediate address is replaced by the temporary exposition address for the lex, and these are put on tape for use during the next stage of translation.

Let us look more closely at the actual sequence of operations involved. Each text-ordered intermediate address is used in succession to place the designated intermediate word in the accumulator. Then the sign bit is tested. If it is found to have been minus, this means that the corresponding lex has previously occurred in the text, and that its temporary exposition address has already been calculated and is now in the address portion of the accumulator. So this address is merely stored in place of the intermediate address, after which the machine passes to the next intermediate address in text order. On the other hand, if the tested sign is found to have been plus, this indicates that the lex is occurring for the first time in the text, and slightly more complicated series of steps is necessary. The last calculated temporary exposition address is taken from storage and added to the exposition length number in the address portion to give the temporary exposition address for this lex, after which this new form of the intermediate word with its sign made minus is stored back in the intermediate list. The new temporary exposition address is also stored in place of the intermediate address and in a temporary storage location in place of the previous temporary exposition address. After this the next intermediate word is placed in the accumulator and the same procedure is followed for it. We give at the top of the next column a routine designed to accomplish these operations.

The routine as shown gives the substance of the manipulations performed, but simple modifications which need not be given in detail here will let it be incorporated into the program in a maximally efficacious way. In the first place, it will be seen that the loop is short enough to be able to fit within a copy loop. On the other hand, the need for saving time dictates that two intermediate addresses be placed in one machine word on tape, so as to halve the amount of time needed to write and read them. Since the loop shown cannot fit twice in a single copy loop, a slightly more complicated routine must be constructed, so that the first half or more of a record of intermediate addresses may be processed during the read-in of the entire record, the remainder during the read-out of the resulting temporary exposition addresses. Thus all the internal operations shown in the above routine can be performed while the tapes are running at full speed.

Complications arise from the fact that transliterated and non-alphabetic material will be present in the text in addition to the intermediate addresses. Words that do not contain two intermediate addresses can be flagged with a minus in their sign bit, with a code in the other two positions of the prefix to indicate whether there is an intermediate address in the address portion, or in the decrement portion, or in neither. The words that contain no intermediate address can be processed completely during one copy loop, so that one does not know in advance at what point after the read-in phase the processing will be completed. This can be handled by going through three phases: a read-in with processing until the end-of-record gap is reached; a read-out with processing until the number of words processed equals the number of words that were read in; and a read-out without processing until the number of words read out equals the number of words that were read in. It is also necessary to have a cut-off device that will stop the process if the space available for expositions in core storage becomes completely allocated before the end of the text. This is done by comparing each newly calculated temporary exposition address with the highest temporary exposition address that can be accommodated. It seems advisable not to stop the process immediately when the space is completely filled, but to let it continue until a temporary exposition address that is too high has been calculated, in hopes that the record currently being processed can be finished before a new lex turns up, since the lex tokens towards the end of a text will preponderantly represent lexes that have already occurred in that text. In addition, a test can be

ROUTINE TO FLAG INTERMEDIATE LIST AND CALCULATE AND STORE TEMPORARY EXPOSITION ADDRESSES

```
        LXA   TXLTH ,2    Initialize text length test.
NEXT    CLA   TEXT ,2     Get intermediate address
        PAX   ,1          from text, and place into
                          index register 1.[23]
        CLS   ,1          Get intermediate word and
                          change sign.
        TPL   FLAGD       Test for flagging.
        ADD   ADDR        Add previous exposition ad-
                          dress to length.
        STO   ,1          Store addressed, flagged in-
                          termediate word.
        STA   ADDR        Store exposition address.
FLAGD   STA   TEXT ,2     Store exposition address in
                          text.
        TIX   NEXT ,2,1   Go to next text word.
Locations used by routine:
TXLTH   DEC   (Number of words in text portion.)
ADDR    MZE   Temporary storage for temporary ex-
              position address.[24]
```

[23] The use of the index register (for following CLS and STO instructions) requires either that the intermediate list run backwards or that the 2's complement of the location of each intermediate word be used as the intermediate address. This is a purely technical consideration which is ignored in the body of the paper for ease of presentation.

[24] This word must have a minus in its sign bit since its contents are to be added to intermediate words flagged with a minus sign. If the intermediate list is to run backwards from the high-numbered end of core storage, the remaining contents of this word in its initialized form should be zero. Otherwise they should be the 2's complement of the address of the location in core storage immediately following the intermediate list.

made before starting each new record to ascertain whether the current temporary exposition address is so high that it would probably run over during that record, to avoid the time waste involved in partially processing a record that cannot be completed.

After all the intermediate addresses have been replaced by temporary exposition addresses, it is necessary to make a *compressed intermediate list.* To do this we use a simple routine to run through the intermediate list, pick out the flagged intermediate words, and recopy them into a shorter list. This compressed intermediate list will have a length of as many machine words as there are different lexes in the text, some 2,000 to 3,000. The purpose of making it is to clear space in core storage for the expositions which are to be brought in. It is probably desirable to sort the words in the compressed intermediate list at the same time, to facilitate bringing in the expositions. A sort according to list portion numbers would be enough, although sorting them by list portion numbers and permanent exposition addresses taken together would place the intermediate words in the same order as their corresponding entries in the exposition list.

After this we are ready to bring the necessary expositions into core storage from the exposition list. The routine given below will do this. It moves word-by-word through the compressed intermediate list, taking the temporary exposition address and the permanent exposition address from each word and placing them in index registers. The end of each exposition in the exposition list is marked by a word containing only zeroes. Thus the program takes the successive words of each exposition from the locations indicated by the permanent exposition address, places them in the accumulator, and tests for this word of zeroes before storing them in the locations indicated by the temporary exposition address. This process terminates when the word of zeroes is encountered. The word of zeroes is not itself transferred, and it is not counted as a word of the exposition for the purpose of determining the exposition length number.

Thus only the expositions of those dictionary entries which are actually needed are kept in core storage, and they will all remain there, immediately addressable, during the stage of translation proper. The routine shown here assumes that the compressed intermediate list has been sorted according to the portions of the exposition list and that index register 4 contains a number showing the number of words in the compressed intermediate list that correspond to the current portion.

Some time can be saved by determining a minimum exposition length and transferring that many words of each exposition before beginning the testing for the word of zeroes.

In the intermediate list there must be one entry for each heading (or lex) that is provided for in the dictionary, but the exposition list contains only one entry for each exposition (or lexeme). Thus for those expositions that correspond to more than one heading (i.e., those lexemes that have allolexes), more than one entry in the intermediate list will contain an address to the same entry in the exposition list. The intermediate list, then, is seen to be, among other things, a device for passing from the graphemic level to the morphemic (or lexemic) one. The present system, or any system which stores expositions separately from their headings so that the headings representing the allolexes of a lexeme may all refer to the same exposition, entered only once, will be seen to differ from an ordinary dictionary by having neither cross-referential nor duplicate expositions. It therefore does not choose the heading corresponding to any one of the allolexes as a primary one that would be more directly related to the exposition for the lexeme in question. This type of system is more in accord with modern linguistic theory, which gives equal status to all allomorphs of a morpheme.

There are, however, two noteworthy cases in which not all the allolexes of a lexeme can have the same exposition. The first arises when two or more lexemes have in common a homographic allolex, but at least one of the lexemes has in addition an allolex not shared by the other lexeme(s). The homographic allolexes must have one exposition, different from that pertaining to the other allolex or allolexes. This shared exposition will contain rules for unraveling the homography—rules that would be otiose in the unshared expositions. An example of this situation is furnished by the Russian morphemes бред, ‘delirium’ and бред/бреж ‘to be delirious’. Instead of one exposition for each morpheme, there will have to be one exposition for бред ‘delirium/to be delirious’ and another for бреж ‘to be delirious’.

The second case turns up when the identification of one or another allolex of a lexeme will serve to resolve the homography between two lexes which can occur contiguous to this lexeme. The different allolexes of this lexeme must then have separate expositions. A Russian example is the stem хозяин/хозяев ‘master, owner’. Either allolex may occur with a suffixal lex -a, but the former allolex of this stem will show that the suffix

ROUTINE TO BRING EXPOSITIONS INTO CORE STORAGE

```
NEXT   CLA  COMPR,4   Place intermediate word into accumulator.
       PAX  ,2        Temporary exposition address into index register 2.
       PDX  ,1        Permanent exposition address into index register 1.
CAL    CAL  ,1        Place word of exposition into accumulator.
       TZE  *+4       Test for word of zeroes.
       SLAV ,2        Store at temporary exposition address.
       TXI  *+ 1,1,—1 Go to next word
       TXI  CAL,2,—1  of exposition.
       TIX  NEXT,4,1  Go to next intermediate word.
```

represents the genitive singular morpheme, while the latter will show that it stands for nominative plural.

Note that if it should be possible and desirable, whether because of a smaller dictionary or by use of a machine with a larger rapid-access memory, to store the intermediate addresses of all headings during the stage of look-up proper, one could make the transition to the lexemic level at an earlier point by addressing the headings for all allolexes of a lexeme to the same word in the intermediate list. This would save space at later stages of the process in two ways: (1) The intermediate list would be shorter in that it would contain, like the exposition list, one entry per lexeme, instead of one entry per allolex as things now stand; (2) less space would be taken up by the expositions brought into core storage for use during the actual translation procedure and by the condensed intermediate list used for addressing them. In the system as presently constituted an exposition will be repeated for every different allolex of the same lexeme that occurs in the text being translated, whereas this modification would entail that an exposition will be brought in only once for each occurring lexeme. There would be corresponding time savings due primarily to the avoidance of having to deal with extra allolexes of occurring lexemes: calculating addresses for storing the expositions and going through the necessary steps to locate them and bring them into core storage.

With the segmentation system used for Russian at the University of California, it is estimated that there will be around 17,000 different lexemes represented in a dictionary with 20,000 headings. Therefore, the exposition list will contain around 17,000 expositions. The set-up of the intermediate stage that has just been described will accommodate this many expositions only if their average length is not more than about 5.6 machine words. In order to accommodate this many expositions with a longer average length it would be necessary to divide the exposition list into more than eight portions and to find space in an intermediate word for a list portion number of four or five bits. This can easily be done by shifting the permanent exposition address one or two bits to the left, thus making available one or two additional bits in the low order portion of the decrement field. This would make necessary only one additional operation, a right shift before placing the permanent exposition address in an index register.

If the expositions in the exposition list are arranged in the same order as their corresponding words in the intermediate list, so that there is one exposition for every heading, a modification in the manner of bringing in the needed expositions is possible. Instead of reading in the whole exposition list in portions and then moving the needed expositions to the temporary exposition locations, the machine can read in only the needed expositions, copying them directly to the temporary exposition locations. There is ample time during the copy loop for modifying the addresses at which the words of the needed expositions are to be stored. During the operation of forming the compressed intermediate list, in preparation for this read-in, the machine would total up the lengths of the expositions pertaining to the unneeded intermediate words between successive flagged intermediate words and store these totals in the unused portions of the flagged words. If this modification is used, it is not necessary to store a list portion number or a permanent exposition address in an intermediate word, so that the information pertaining to two intermediate words can be stored in one word of tape. On the other hand, one loses the flexibility and convenience of being able to order the exposition list independently of the intermediate list. Moreover, the time gained by using this modification is largely offset by the time needed for reading in the duplicate expositions made necessary by it, so that it would be advantageous only if there are additional reasons for having one exposition per heading.

A considerably different alternative approach would be one in which the expositions themselves would be arranged in text sequence and stored on tape, each exposition being repeated as many times as its corresponding lexeme occurs in the text. This approach could be carried out without the use of the intermediate stage, if the look-up proper were designed to yield permanent exposition addresses or some substitute therefor instead of intermediate addresses. There would then be less complication in parts of the program, but it would require much more time for its execution, since it would necessitate running the exposition list through core storage several times, once for each portion of the text (in the form of a series of exposition addresses or serial numbers of expositions or the like) brought into core storage, and would also entail more movement of the text tape during the stage of translation proper.

Timing

An estimate of the amount of time required for the look-up process, including input of the original text and the intermediate stage, is given in Table XII in terms of milliseconds per word, estimated to the nearest quarter of a millisecond. (A quarter of a millisecond is about the time required for 10 operations of the usual type.)

Similar calculations made for other IBM computers result in figures of slightly less than five milliseconds per word on the 709 and 1.2 milliseconds per word on the 7090. That is, the speed of the system ranges from 125 words per second on the 704 to 833 words per second on the 7090.

The estimates for the stage of look-up proper and segmentation have been made in terms of an average word. Many words will take longer than the average one, but it is believed that this excess expenditure of time will be at least compensated for by the many words which will require less time than average. For most words, the letter-by-letter addressing will be conducted for the first three letters; for some this process will continue for the fourth or even the fifth letter, but for

TABLE XII: ESTIMATE OF TOTAL LOOK-UP TIME PER WORK FOR RUSSIAN TEXT OF 30,000 WORDS, USING 704 COMPUTER.

		Milliseconds
Input		2
Look-up Proper and Segmentation		
Letter-by-letter addressing	¾	
Truncate selection	¾	
Suffix location	¼	
Complications in segmentation	¼	
Segmentation checking	¼	
Misc. "red tape" operations	¼	
		2½
Read-Out of Intermediate Addresses		½
Intermediate Stage		
Loading intermediate list	¼	
Run-through of text	¾	
Compressing intermediate list	¼	
Loading selected expositions	1¾	
		3
Total		8

others it will stop after the second or even after the first (in the case of one-letter words). The average vestigand has one suffix, and most suffix tokens have one letter. Where more than one suffix is involved or where a suffix is longer than average, the look-up time is slightly longer than that shown in the table; but on the other hand there are many vestigands which are unit lexes, so that segmentation time is zero. For some vestigands it will be necessary to consult supplementary truncate words, with a resulting loss of time; but many others (including the most frequently occurring words) are so short that it is not necessary to consult the truncate lists for them at all. The item in the table called "Complications in segmentation" is one which most vestigands will not be concerned with at all; it is put in the table to provide for those situations in which the first provisional segmentation does not check out and/or it becomes necessary to consult the segmentation checking list. In the latter circumstance, especially when the former is also present, a relatively considerable amount of time might be expended, perhaps as much as a millisecond or more. These situations, however, will occur for only a small proportion of vestigands; the quarter millisecond given in the table represents the time taken up by the occurrence of these situations apportioned among all vestigands. It is doubtless too high a figure.

Since a major portion of the time used in the intermediate stage is taken up by the movement of the tape containing the exposition list, the length of expositions makes a great deal of difference in the amount of time required. If the average length of expositions is seven or eight machine words, the total time necessary for the operations that constitute the intermediate stage will be between 85 and 95 seconds, which amounts to about three milliseconds per word token. As six seconds is a liberal estimate for the operations which cannot be performed while tape is moving, it is seen that the great preponderance of the time is taken up by tape movements. Four of these are necessary. First the 20,000 words of the intermediate list must be brought into core storage. Next the tape of text-ordered intermediate addresses must be read into core storage, and a replacement list of text-ordered temporary exposition addresses must be written out onto tape. The addresses of these two text-ordered listings can be stored two per word of tape, so that copying the 60,000 addresses of each will require moving through 30,000 words of tape. Finally, the permanent exposition list must be read into core storage so that the needed expositions can be selected. If it contains 17,000 expositions with an average length of eight words, this will mean a tape movement of 136,000 words. The total amount of tape to be moved will then be 216,000 words, which, at a rate of 2500 words per second, will take 86.4 seconds. Another half-second must be added on to allow for putting the tapes in motion for each individual portion that is copied. It will be seen that almost two-thirds of the tape time is devoted to the running in of the permanent exposition list. If we make a more moderate estimate of five words for the average exposition length, the total tape movement is reduced to 165,000 words, which would consume 66 seconds. The time for the whole intermediate stage would then be less than 2.5 ms. per word token or 1.25 ms. per lex token.

Let us contrast the time necessary for carrying out the alternative approach of storing all needed expositions on tape in text sequence without using an intermediate stage. This would likewise entail running in the 30,000 words of (exposition) addresses, but would mean putting out the expositions on 300,000 words of tape (at five words per exposition). Since the permanent exposition list must be run through for each portion of the address tape brought into core storage, and since those expositions that are selected out must be kept in core storage until the completion of each pass of this tape, a large number of these passes will be necessary. If we assume that 25,000 words of core storage will be available to hold the selected expositions, then twelve will be the lowest possible number of passes. This will amount to a tape movement of 1,020,000 words, giving a total of 1,350,000 words. The time consumed by tape runs, allowing for tape starts, will then be about 543 seconds (9.05 minutes). This comes to 18.1 ms. per word token—over seven times as long as for the intermediate stage. These calculations indicate that even for the purpose of listing expositions on tape in text order a great saving in time is to be obtained by the use of the intermediate stage.

Organizing and Updating

The organization and coding of the dictionary for use by the program described here would be too tedious a job to be performed by hand, but it is relatively easy to

construct a program which will enable the computer to do it. Such a program may be called the *dictionary adapter*. It will take as its input the dictionary in the form in which it is worked on by linguists, namely with the headings attached to the expositions and the entries arranged in alphabetical order of the headings. Although the adapter has not yet been written, it is estimated that its running time on the 704 will not amount to more than ten or fifteen minutes at the most. Therefore, the easiest way to make modifications will be to wait until several changes and additions have been accumulated and then reorganize the entire dictionary.

The operations of the adapter fall into three stages. First it will separate the headings from the expositions and put the latter on tape to create the exposition list, at the same time forming the intermediate word for each exposition and associating this with its heading. In the second stage it will work with the headings to form the letter tables and truncate lists, thereby determining the intermediate address for each intermediate word. Finally, it will place the intermediate words at the locations indicated by their intermediate addresses to make the intermediate list.

Received February 23, 1960
Revised August 19, 1960