[From: *IBM Technical Newsletter* No.9, January 1955]

RESEARCH IN LANGUAGE TRANSLATION ON THE IBM TYPE 701

Peter Sheridan
IBM Scientific Computing Service

Introduction

January 7, 1954 marked the successful culmination of a joint project of the Institute of Language and Linguistics of the Georgetown University School of Foreign Service and IBM: one language (Russian) was successfully translated into another (English) by means of a high-speed electronic digital computer.

For over the past decade a small group of men in various institutions throughout the country had been engaged in research on the practicability of using high-speed electronic computing or data processing machines for language translation. Finally about two years ago, a staff member of the Institute of Languages and Linguistics of Georgetown University devised a systematic method for converting the meaning of words in one language into words of another language without the need for pre-editing or post-editing text. * Though Russian into English translation was used as the model in the first practical experiment, the Georgetown language conversion method can be applied almost universally.

Briefly, in this system a rule-tag is attached to a normal sentence word, determining its position and exact meaning in a sentence. Six rule-tags in all were chosen, specifically because they have a broader effect on language translation than all the other rules that were studied. These rule-tags govern the transposition of words where that is required to make sense, choice of meanings where a word has more than one interpretation, omission of words that are not required for correct translation, and insertion of words that are required to make sense. Though it is estimated that at least one hundred rule-tags may be required to translate adequately scientific and technical literature in general, the six selected rule-tags remain basic to translation.

Staff members of the Georgetown Language Institute proposed to IBM that their practical solution to the language translation problem be adapted to the 701 computer. Then under the sponsorship of the IBM program for endowed research in computation, ' work was undertaken at IBM World Headquarters on the enormously detailed problem of programming the Georgetown linguistic solution into a meaningful program in the binary language of the 701 computer.

* L. E. Dostert (Institute of Language and Linguistics, Georgetown University School of Foreign Service), "An Experiment in Mechanical Translation: Aspects of the General Problem."

In the first language translation demonstration in January 1954, a sample dictionary consisting of 250 Russian words and their English equivalents, together with the rule-tags which govern word choice and position, was stored on the surface of a magnetic drum on the 701. The detailed instructions were placed in electrostatic storage. Two hundred Russian sentences were then fed into the computer and printed out in correct English translation at the rate of one full sentence every six or seven seconds.

I. Fundamentals of Language Translation Program

The systematic conversion of a given "source" language text into a desired "target" language text involves two essentially distinct though logically interrelated programs, which may be referred to as "lexical" and "operational" syntax. "Lexical" syntax refers to discrimination between and recognition of the word elements of the source language. "Operational" syntax refers to their subsequent manipulation in terms of the grammatic-syntactic requirements of the target language. The term "word" refers to any of the meaningful "strings," or series, of alphabet symbols occurring as source entries in an authoritative intralingual dictionary. Alphabet symbols include the actual letters, punctuation marks, diereses and diacritical marks. The term "sentence" connotes any meaningful string of words formed in accordance with the grammatical rules of the language itself.

The Russian-English dictionary utilized in the 701 conversion program comprises two types of source language words, that is, "complete" and "subdivided." "Sub-divided" words are those which occur in the dictionary as a separate root and ending. Each of the two portions of a subdivided word is referred to as a "partial" word. The root is the left partial, and the ending is the right partial. An example of a subdivided word is

polyityichyeskyix,

which occurs under "p" as the left-partial word

polyityichyesk-

and under "y" as the right-partial word

-yix.

It should be noted that the term "subdivided" is applied to a source language word only with respect to the program's internal operation, that is, only after the word has been read from punched card input and is being processed as datum within the machine's electrostatic memory. The machine recognizes the type of word only when reference is made to the magnetic drum-stored dictionary.

A "complete" word is one which, as the name indicates, does not occur as a separate stem and ending. Such a word, for example, is

yizgotovlyayetsya,

which occurs precisely in the form spelled out in the "y" region of the stored dictionary.

Each line of the dictionary includes the following material in the order mentioned:

Russian source language entry (complete or partial), successive alternate English target equivalents, and a set of three syntactical codes. The English target language equivalents have been restricted to two in number for this particular experiment. The three syntactical codes, which will be referred to as "diacritics," have the basic functions of choice and linear arrangement of target language material.

When a Russian language word is looked up as a complete or subdivided sentence item, its alternate English language equivalents are stored in a specially assigned region of electrostatic storage and its associated diacritics are stored in another region. The machine is programmed to remember which set of English language equivalents in the former storage region is associated with any given set of diacritics in the latter.

Upon completion of this lexical syntax subprogram, the machine proceeds to execute its operational syntax program of choice, rearrangement, insertion and deletion. (This will be explained later in some detail.)

Three kinds of diacritics are associated with each Russian dictionary entry: two are stored in the dictionary and the third is generated by the program. They include:

a) A "program-initiating" diacritic, or "PID," which initiates the execution of one of the six rules of operational syntax incorporated in the stored program.

b) A "choice-determining" diacritic, or "CDD," further subdivided into a three-digit "CDD_1" and two-digit "CDD_2" code. (The need for two choice-determining diacritics is chiefly due to both forward and backward reference to the same item of source language text.)

c) An "address diacritic," containing the initial full-word addresses ADD_1, ADD_2 in electrostatic storage of both English language equivalents associated with the PID and both CDD's. This diacritic is essentially a "bookkeeping" feature.

II. Working Assumptions of the Program

The Machine

A total of 33 distinct operations may be performed on the EDPM Type 701 computer. These operations include addition, subtraction, multiplication, division, shifting, logical transfers, etc.

The primary unit of information in the 701 is defined as a "full word" which consists of 35 bits (binary digits) and a sign bit, or 36 bits in all. Each full word may be further split up into two "half words" which consist of 17 bits and a sign, or 18 bits in all.

. Following is a brief description of the characteristics and organization of those units of the 701 specifically involved in the execution of the language translation program.

1. Electrostatic Storage

Electrostatic (high-speed) storage of the 701 consists of a bank of cathode ray tubes. One electrostatic storage unit can accommodate 2048 full words or 4096 half words. The 2048 full-word locations (cells) in electrostatic storage are identified by the negative even integers from -0000 to -4094, and the 4096 half-word locations by the positive

integers from +0000 to +4095. The relationship between full and half-word addresses is as follows: if "-2n" refers to a full-word location, then "+2n" refers to the left half word and "+2n+l" to the right half word into which the full-word location may be split.

Average access time to full and half-word information (consisting of machine instructions or data) is 12 microseconds.

Although two electrostatic storage units are available, one unit only is used in this pilot language translation program.

2. Magnetic Drums

The magnetic drum unit in the 701 consists of four addressable drums, each drum accommodating 2048 full words of information. The full words on the drums are addressable by a system similar to that used in electrostatic storage, except that there is no provision for recognizing half words.

Because access to individual words on a drum is relatively slow in comparison with electrostatic storage access, it is more efficient to use drums to store large blocks of information. Average access time to the first word in a block, or "unit record," of information is 40 milliseconds. After the first word of such a block has been located, the remaining words are read or written at the rate of 800 full words per second.

3. Card Reader

Cards may be punched (up to 72 columns) in any symbol desirable to the programmer--usually alphabetic, decimal, or binary. Cards are read at the rate of 150 per minute. The card punches in alphabetic or decimal numerical code are converted simultaneously to the computer binary code at full card reading speed by providing a suitable "utility" program.

4. Printer

The 716 alphabetical printing unit of the 701 prints up to 72 alphabetic, numeric, and special characters (in any desired format) at the rate of 150 lines per minute.

The Program

A seven-bit code is used to represent the alphabetic, numeric and special characters in the translation program. Letters A-Z are denoted by the seven-bit equivalents of $(11\text{-}36)_{10}$, namely, 0001011-0100100; numerals 0-9 by 0000000-0001001; hyphen (-) by $0001010=(10)_{10}$; blank by $0100101=(37)_{10}$; and asterisk (*) by $0100110=(38)_{10}$. Thus, a full-word location in storage may contain up to five alphabetic, numeric or special characters in bit positions 1-7, 8-14, 15-21, 22-28 and 29-35.

Character	Decimal Equivalent	Seven-Bit Code
0	0	0000000
⋮	⋮	⋮
9	9	0001001
hyphen (-)	10	0001010
A	11	0001011
⋮	⋮	⋮
Z	36	0100100
blank	37	0100101
asterisk (*)	38	0100110

Although a six-bit code would be adequate to represent any of the 40 alphanumeric or special characters used in this program, seven bits have been allotted for the following reasons:

a) The sign bit in a full word is reserved for various different tests in dictionary search and operational syntax routines. Thus a maximum number of five characters can be stored in a full-word location, using either a six-bit or a seven-bit code.

b) It was decided that it would be useful to add a leading zero bit to each six-bit character code. The presence of a 1 in this leading character bit position signals the end of execution of an English language sentence print-out, causing the machine to re-initiate the card reading and conversion routine for the next sentence to be translated.

c) A seven-bit character code permits uniform spacing in a full-word location. This is extremely desirable because of programming considerations.

Detailed statements of the working assumptions of the program follow.

1. Dictionary Storage

The Russian-English dictionary is stored on magnetic drums. Each line makes up a drum unit record and has the following format:

a) Russian and English language word strings occupy integral numbers of consecutive drum locations.

b) Spacing between Russian, English and diacritic strings is accomplished by means of "null strings." A null string is defined as a full-word location containing zeros.

c) PID, CDD_1 and CDD_2 occupy two consecutive drum storage full-word locations. The three-digit PID is contained in bit positions 15-35 of the first storage location; positions 1-14 are occupied by zero bits. CDD_1 also a three-digit diacritic, occupies bit positions 1-21 of the second storage location; CDD_2 , a two-digit diacritic, occupies the remaining bit positions, 22-35.

d) Each line is terminated by means of two consecutive null strings. Each line begins at the first full-word location following termination of the preceding line.

e) Any unused (non-significant) character positions in a dictionary-stored Russian word entry are occupied by seven-bit coded zeros.

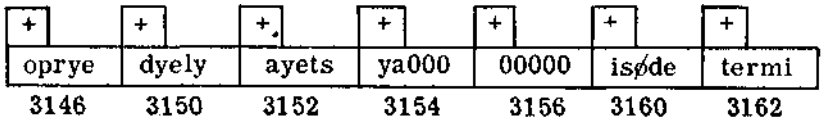
f) Similarly, any unused character positions in a dictionary-stored English language equivalent are occupied by seven-bit coded blanks, subsequently changed to zeros. (The Type 716 control panel is wired for zero suppression.)

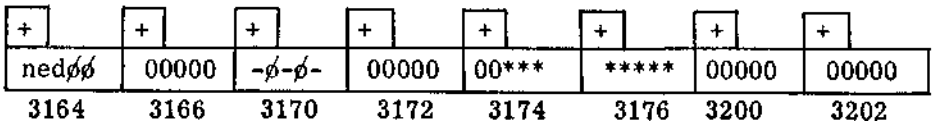
g) The algebraic sign (+ or -) of each line is distributive, i.e., it belongs to each and every full-word location defining the contents of that line. It is determined in the following way: the sign "+" is associated with a complete or left-partial (root) word; the sign " – " with a right-partial (ending) word. This distinction of signs is useful at various stages of both the lexical and operational syntax subprograms.

Example 1. The line containing the Russian complete word entry

opryedyelyayetsya

appears as follows (each rectangle denotes a drum full-word location and the numerals below each word its octal drum address):



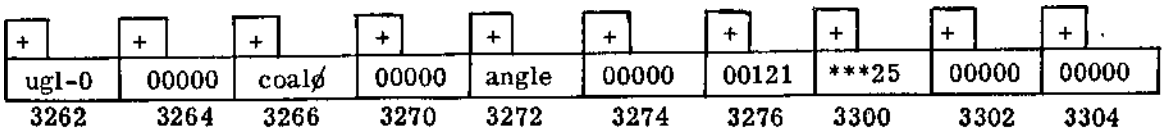


As stated before, each character is represented by a seven-bit code. The sign ø is used here to denote a coded blank. If only one significant English equivalent is available (as in this case), it occupies the first of the ordered English equivalent locations, Eng_1 . Eng_2 is denoted by " -ø-ø-."

Example 2. The line containing the Russian left-partial (root) entry

ugl-

appears:



Example 3. The line containing the right-partial (ending) entry

-a

-	-	-	-	-	-	-	-	-	-
a0000	00000	of000	00000	-ø-ø-	00000	00131	22225	00000	00000
0002	0004	0006	0010	0012	0014	0016	0020	0022	0024

As may be easily verified, the binary-coded representation for the unit record in Example 2 is the following:
appears:

Drum Storage Address	First Character	Second Character	Third Character	Fourth Character	Fifth Character
3262	0011111	0010001	0010110	0001010	0000000
3264	0000000	0000000	0000000	0000000	0000000
3266	0001101	0011001	0001011	0010110	0100101
3270	0000000	0000000	0000000	0000000	0000000
3272	0001011	0011000	0010001	0010110	0001111
3274	0000000	0000000	0000000	0000000	0000000
3276	0000000	0000000	0000001	0000010	0000001
3300	0100110	0100110	0100110	0000010	0000101
3302	0000000	0000000	0000000	0000000	0000000
3304	0000000	0000000	0000000	0000000	0000000

2. Sentence Read-In

Each Russian sentence word read into electrostatic storage is separated from its successor by a single null string, and two successive null strings terminate a sentence. Again (see Address 3262 above), unused character positions in the last full-word location containing a given sentence word are occupied by seven-bit coded zeros. Therefore spacing between successive sentence words varies between 35 and 63 zero bits (respectively 5 and 9 unused character positions), depending upon the length of the Russian word.

3. Russian Word Look-Up

To facilitate Russian word look-up in the drum-stored dictionary, the first character of each input sentence word is used as the argument of a "thumb index" table of 26 full words. Each word in the table contains the drum address and drum location of the initial full-word location of the first Russian dictionary entry in that particular alphabetic region. Thus, if "X" denotes an alphabetic character $11 \leq X \leq 36$, the following mapping is given:

$$X \longrightarrow (D_x, L_x),$$

where D_x= address of one of the four addressable drums in the magnetic drum unit

(0128, 0129, 0130 or 0131) and L_x= drum location (=2t, where $0000 \leq t \leq 2047$).

Since the binary numeric equivalents of A-Z are consecutive, reference to the "thumb index" is virtually immediate. If "X" denotes the decimal equivalent of the initial character of a Russian sentence word, then

$$-[T+(2X-20)], \text{ where } T > 0 \text{ and even}$$

denotes the full-word location in electrostatic storage containing D_x and A_x, in the left and right half-word address positions respectively. (The address part of a half word in storage is contained in the rightmost 12 bits.)

4. Data Storage

a) Each Russian sentence read into the machine is simultaneously converted from standard IBM punched-card code to the corresponding string of seven-bit equivalents and then stored in an electrostatic storage region beginning at - (S+2), where S > 0 and even. (See paragraph 2.)

b) In the course of each look-up subroutine iteration, each Russian dictionary entry to be examined is copied from its drum location(s) into an erasable-storage "comparison" region of electrostatic storage beginning at - (C+2), where C > 0 and even.

c) Immediately following the completion of a sentence-word look-up, the pair of English language equivalent strings from the dictionary, Eng_1 and Eng_2, and associated PID, CDD_1 and CDD_2 strings are stored in electrostatic storage regions as follows:

1) Eng_1 and Eng_2 strings are stored in a region beginning at - (E+2), where E > 0 and even.

2) Diacritic strings are stored in successive blocks of three full-word locations. The j-th (j=l,2,3) word of the i-th diacritic block has address

$$-[D+6(i-l)+2j], \text{ where } D > 0 \text{ and even.}$$

Location

$$-[D+6(i-l)+2]$$

contains PID^1 in bit positions 15-35.

Location

$$-[D+6(i-l)+4]$$

contains CDD_1^i in bit positions 1-21, and CDD_2^i in bit positions 22-35.

Finally, half-word location

$$+[D+6(i-l)+6]$$

contains the initial full-word address, ADD_1^i, of Eng_1^i in the English-equivalents region. And half-word location

$$+[D+6(i-l)+7]$$

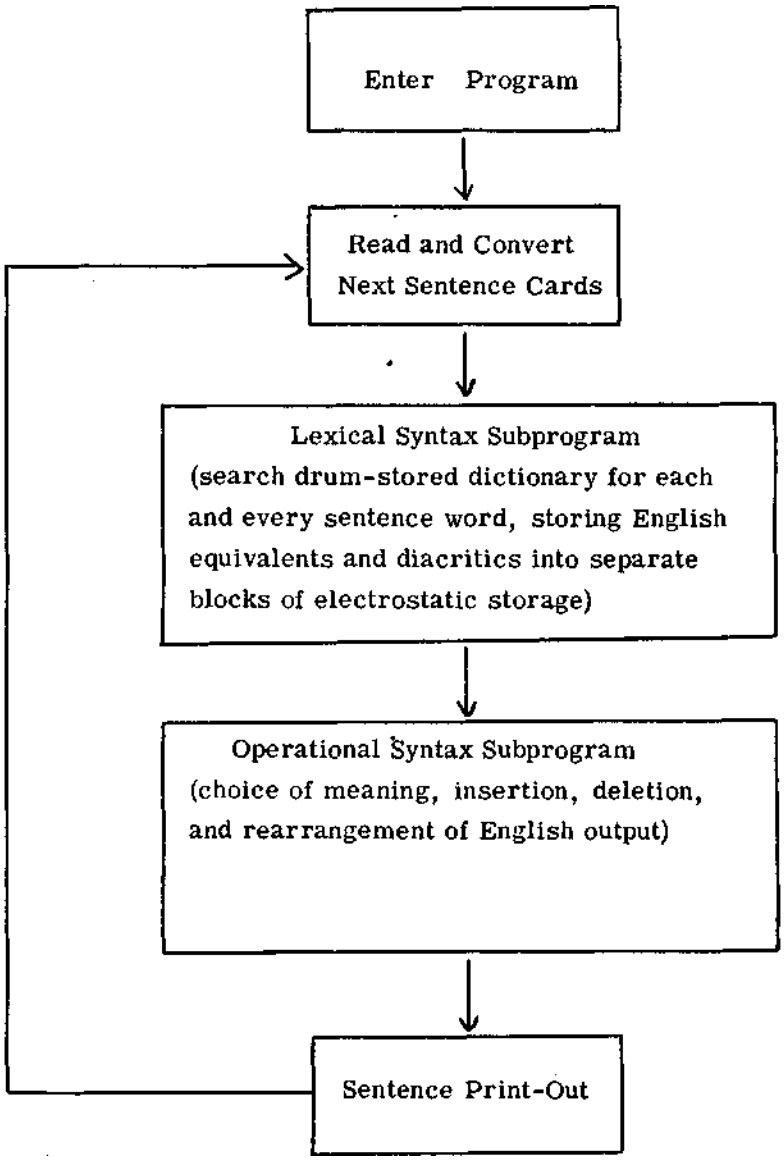
contains the initial full-word address, ADD_2^i, of Eng_2^i.

Of course, $ADD_1^i = -(E+2)$.

d) An erasable region of electrostatic storage beginning at location - (R+2) is specially assigned for temporary storage of right-partial (ending) words during the course of each ending look-up subroutine.

e) The erasable final-print-out region of electrostatic storage begins at -(P+2), where P > 0 and even.

The basic operations of the machine translation program are indicated in the flow chart in Figure 1.



Basic Operations of the Machine Translation Program

Figure 1

III. Lexical Syntax Subprogram

Following is a brief description of the basic logical functions and operations of the initial lexical syntax subprogram of the machine translation program. The lexical syntax subprogram is completed for each and every sentence word in the source language before the operational syntax program controlling choice, rearrangement, insertion, and deletion is initiated.

(1) Let $X_1X_2...X_n$, where $n \geq 1$

denote a source word, consisting of a string of meaningfully juxtaposed characters. As mentioned before, an initial quick table, look-up search notifies the machine of the drum address D_{x_1} and location A_{x_1} of the first source entry in the X_i region of the magnetic-drum-stored dictionary. Since source and target words occupy integral numbers of full-word locations in drum and electrostatic storage, and each full-word location contains up to five significant alphabetic characters,the above character string (1) may be briefly denoted by

(2) $X_1^iX_2^iX_3^iX_4^iX_5^i$,

where $i \geq 1$, X_j^i = A, B, C, ...or Z, and X_j^i = 0000000 only if $j \geq 2$.

If $Y_1^i\ Y_2^i\ Y_3^i\ Y_4^i\ Y_5^i$ denotes a dictionary-stored source entry, where $X_1^i = Y_1^i$, the machine is then programmed to run a character-by-character comparison test on both input and dictionary strings.

Character extraction for purposes of comparison is accomplished by means of the EXTRACT operation:

(3) - EXTR [ ]

The EXTRACT operation is a process of logical multiplication. The contents of the accumulator register are matched with the contents of the electrostatic storage location specified by the address part of the EXTRACT instruction. For each bit position containing a 1 in both the accumulator and in storage, a 1 is placed in the bit position in storage. The remaining bit positions in storage are filled with zeros.

In order to extract a character from a five-character string in storage location -U, the entire contents of -U are placed in the accumulator. The appropriate extractor, or "mask," is then placed in location -V. The "mask" has 1's in only those bit positions of location -U that are being compared at the moment. Thus if the second character in location -U is to be extracted, the mask

0000000	1111111	0000000	0000000	0000000

is stored in location -V. The sequence

(4) -RADD U
-EXTR V

then stores the desired information in location -V.

A set of five character extraction masks are stored permanently in locations

(5) -(M+2k), where k=l,2,3,4,5

The masks may be denoted by

(6) $Z_1^k\ Z_2^k\ Z_3^k\ Z_4^k\ Z_5^k$, where $Z_L^k = \begin{cases} 1111111 & \text{if } k = L \\ 0000000 & \text{if } k \neq L \end{cases}$

If for some i, j

(7) $X_h^h = Y_x^h$ for all $h < i$ and $k=1,2,3,4,5$, or for $h=i$ and each $k < j$

but $X_j^i = 0000000$,

look-up of a complete or right-partial (ending) sentence word has been achieved, since dictionary entries are stored in ascending numerical order and every sentence word occurs either complete or subdivided in the dictionary.

If, on the other hand, (7) holds, but

(8) $X_j^i - Y_j^i \neq 0000000$,

then either

a) Y_j^i and X_j^i disagree alphabetically

or

b) Y_j^i is a coded hyphen (0001010).

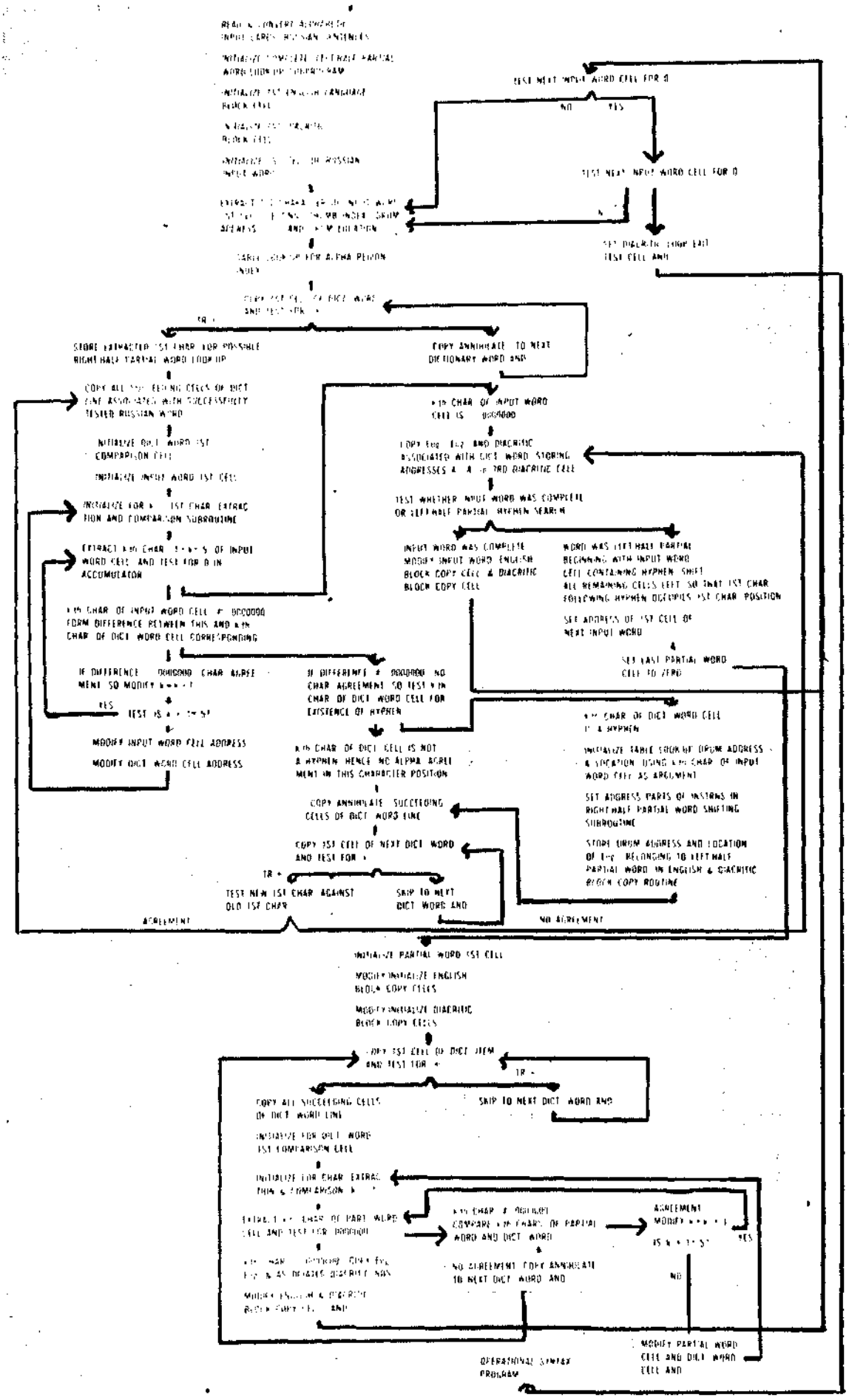
If case a) holds, the machine is programmed to skip to the next dictionary source entry. If case b) holds, the machine is programmed to remember the initial drum locations of Eng_1, Eng2 and diacritic strings associated with the possible root encountered, and continue searching the rest of the X_1^i region for a possible longer root. Since the last root encountered is the longest, it is necessary to retain only the last occurring English-equivalent and diacritic references.

Having looked up the longest word root, the machine is programmed to search for the ending (right-partial) entry whose first character is X_j^i. Before the ending look-up subroutine begins, however, the ending character string appearing in the original subdivided word must be repositioned so that the j-th character becomes the first, the j+l-st the second, etc. This is accomplished merely by shifting the contents of the full-word location(s) containing the ending character string 7(j-l) bit positions to the left. The ending character string is then stored in the region of storage set aside for the temporary storage of right-partial words during each ending look-up subroutine:

-(R+2), -(R+4),...

As soon as each source sentence complete and subdivided word has been looked up and its associated English equivalents and diacritics read from the drum into electrostatic storage, a small subroutine writes a special exit-transfer test value (a 1 in the leading character bit position) in the first full-word location following the last diacritic block in the diacritics region of storage. This "terminal" diacritic value is later recognized by the machine as signalling the end of the operational syntax subprogram.

A flow chart of the lexical syntax subprogram is shown in Figure 2. It should be noted that a_I and α_Γ in Figure 2 are equivalent to D_x and L_x, resepectively, in the text.



Lexical Syntax Subprogram

Figure 2

IV. Operational Syntax Subprogram

Rules of Operational Syntax

Following are the six rules of operational syntax programmed into the machine. These rules are in virtually the original language of presentation of the problem.

Rule 1

If PID associated with a Russian dictionary entry is " ***," then adopt English equivalent I of alternative English language equivalents, retaining order of appearance of output with respect to previous sentence words.

Rule 2

If PID is "110," is CDD_2 associated with the first preceding complete sentence word encountered "21"? If so, reverse order of appearance of words in output (i.e., word carrying "21" should follow that carrying "110"); otherwise, retain order.
In both cases, adopt English equivalent I associated with "110."

Rule 3

If PID is "121," is CDD_1 of the first following complete, subdivided or partial (root or ending) sentence word encountered "221" or "222"? If it is "221," adopt English equivalent I of word carrying "121"; if it is "222," adopt English equivalent II.
In both cases, retain order of appearance of words in output.

Rule 4

If PID is "131," is CDD_2 of first preceding complete sentence word or either portion (root or ending) of first preceding subdivided sentence word encountered equal to "23"? If so, adopt English equivalent II of word carrying "131," and retain order of appearance of words in output; if not, adopt English equivalent I and reverse order of appearance of words in output.

Rule 5

If PID is "141," is CDD_1 of first preceding complete sentence word or either portion (root or ending) of first preceding subdivided sentence word encountered equal to "241" or "242"? If it is "241," adopt English equivalent I of word carrying "141"; if it is "242," adopt English equivalent II.

Rule 6

If PID is "151," is CDD_1 of first following complete sentence word, or either portion (root or ending) of first following subdivided sentence word encountered equal to "25"? If so, adopt English equivalent II of word carrying "151"; if not, adopt English equivalent I.

In both cases, retain order of appearance of words in output.

Example

The following example illustrates how the rules of operational syntax are applied in the computer language translation program.

Source Sentence: Vyelyichyina ugla opryedyelyayetsya otnoshyenyiyem dlyini dugi k radyiusu.

Analysis:

Russian Item	English Equivalents Eng_1	Eng_2	PID	CDD_1	CDD_2
Vyelyichyina	Magnitude	-------	***	***	**
ugl -	coal	angle	121	***	25
- a	of	--------	131	222	25
opryedyelyayetsya	is determined	-------	***	***	**
otnoshyenyi -	relation	the relation	151	***	**
- yem	by	-----...	131	***	**
dlyin -	length	-------	***	***	**
- i	of	.....	131	***	25
dug -	arc	-------	***	***	**
- i	of	.....	131	***	25
k	to	for	121	***	23
radyius -	radius	-------	***	221	**
- u	to	.....	131	***	**

Target Sentence: Magnitude of angle is determined by the relation of length of arc to radius.

Programming Assumptions

An intensive analysis of the Georgetown linguists' statement of the rules of operational syntax was made in conjunction with a detailed study of the structure of diacritic strings corresponding to a limited (ten) but syntactically representative Russian sentence sample. The following programming assumptions were subsequently evolved and confirmed for a larger sample:

P1) For each $i \geq 1$, $PID^i = ***, 110, 121, 131, 141$ or 151.

P2) If $PID^i = 110$, $CDD_2^{i-j} \neq 21$ $(j \geq 2)$

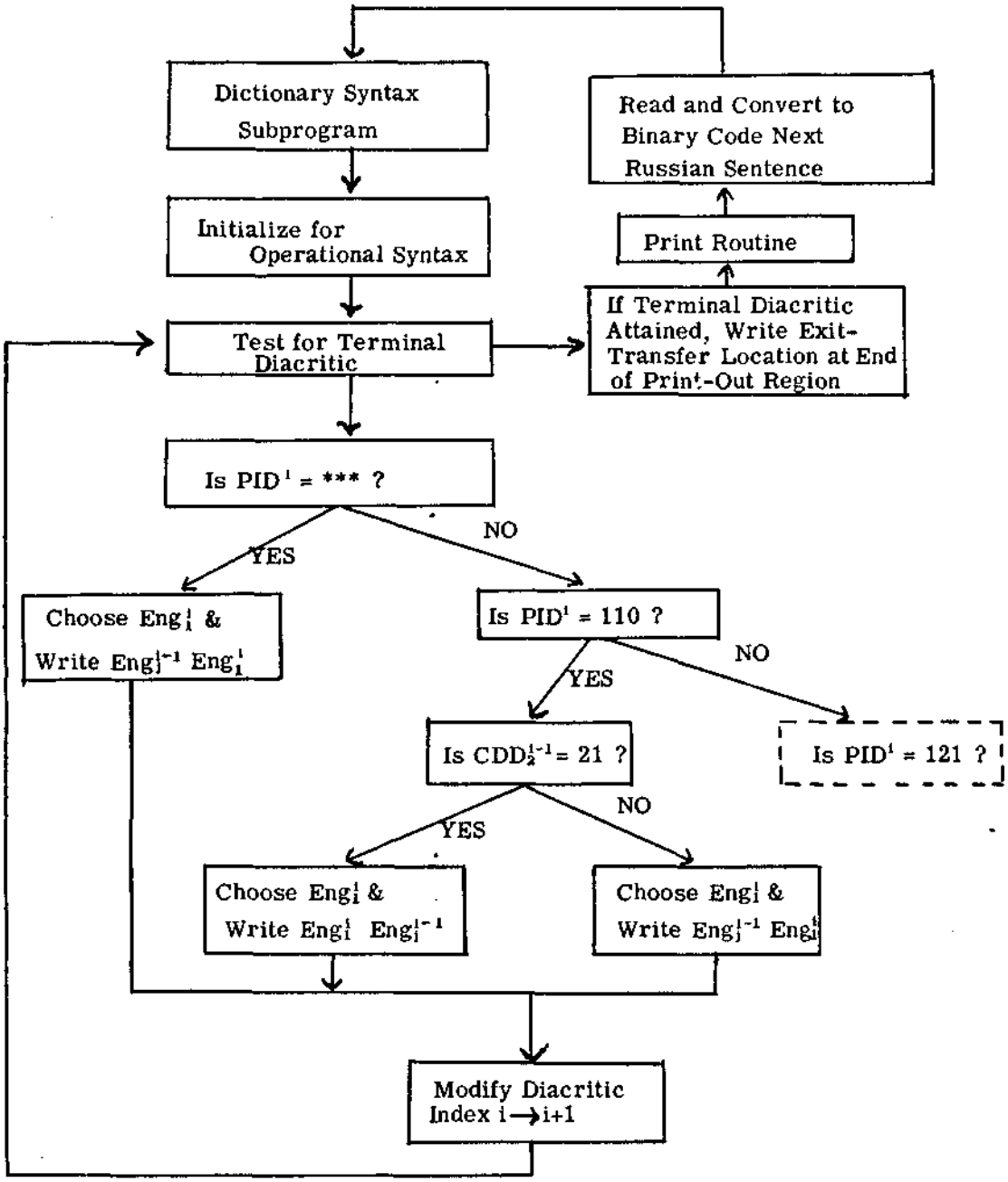
P3) If $PID^i = 121$, then for some $j \geq 1$, $CDD_1^{i+j} = 221$ or 222.

P4) If $PID^i = 131$ but < 0, then $CDD_2^{i-1} \neq 23$.

P5) If $PID^i = 141$, then for some $j \geq 1$, $CDD_1^{i-j} = 241$ or 242.

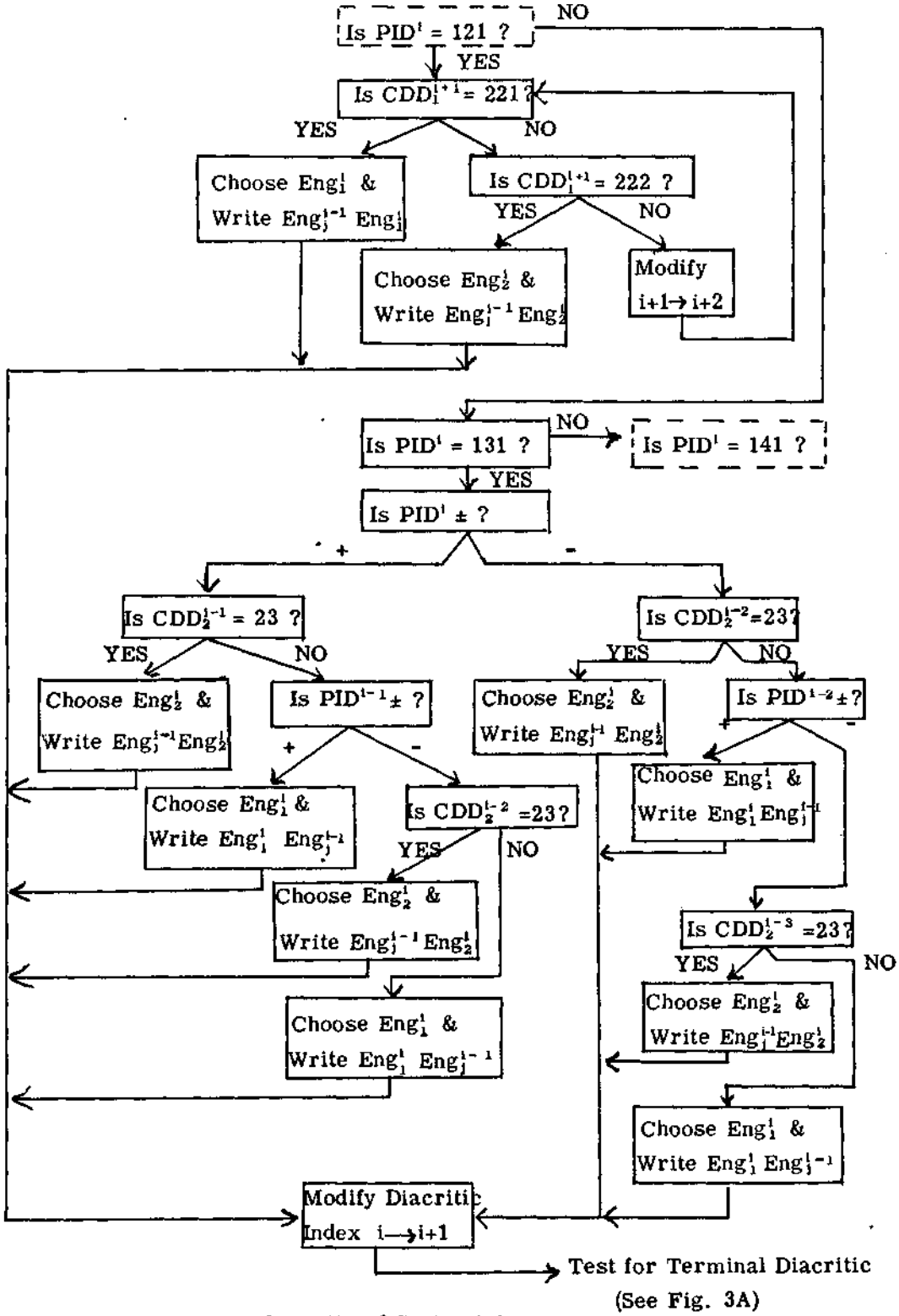
P6) If $PID^i = 151$, then $CDD_2^{i+1} \neq 25$.

These assumptions together with the rules stated before led to the operational syntax subprogram flow charts shown in Figures 3A, 3B, and 3C. These charts represent completely the phase of programming related to language structure.



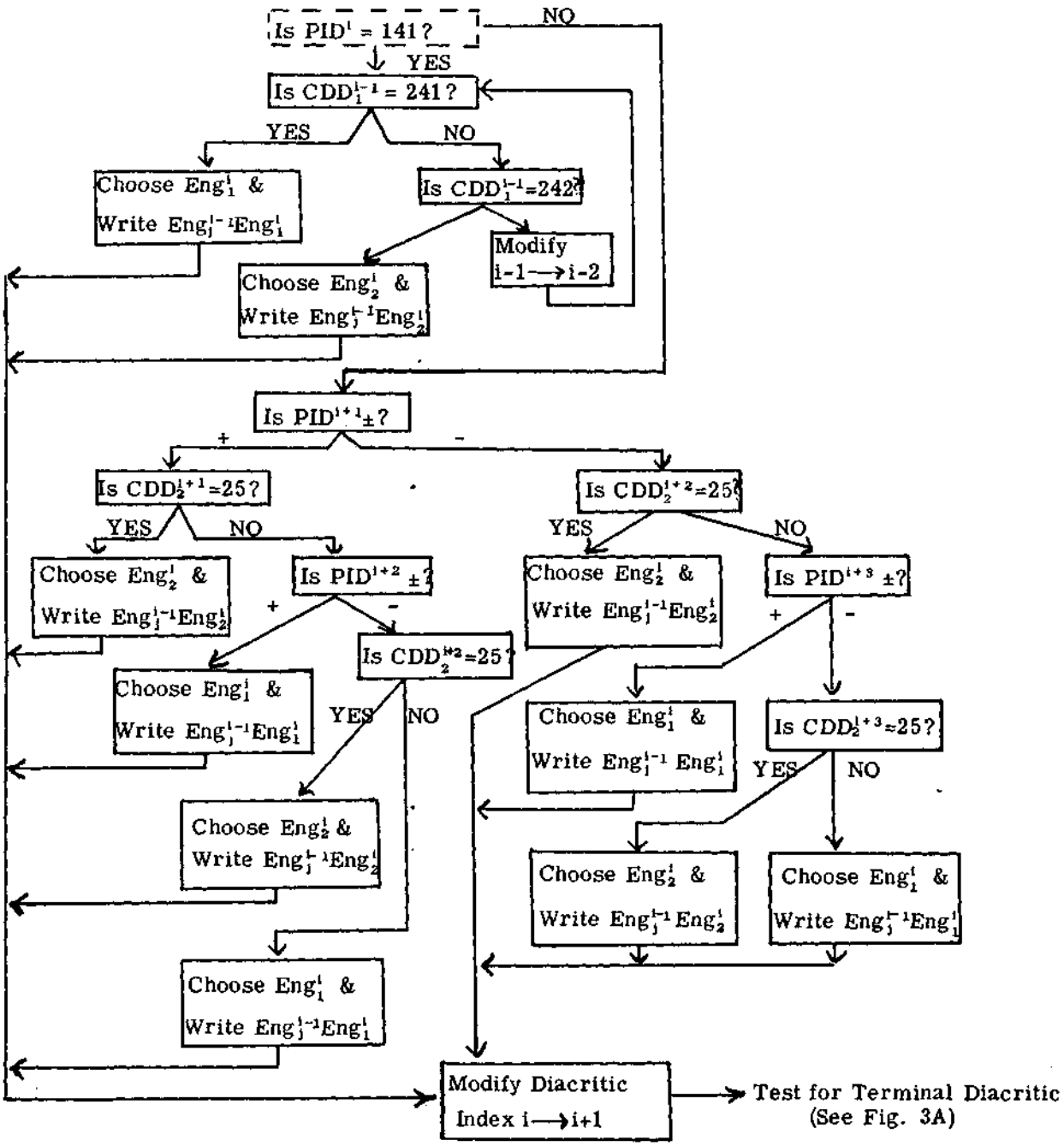
Operational Syntax Subprogram

Figure 3A



Operational Syntax Subprogram

Figure 3B



Operational Syntax Subprogram

Figure 3C

Except for some rather intricate bookkeeping routines, actual instruction-coding of the reformulated rules was straightforward.

Upon termination of the operational syntax subprogram a special exit-transfer location is written at the end of the print-out region in storage. A special exit-transfer test value contained in this location causes the machine to reinitiate the card reading and conversion routine for the next sentence to be translated.

V. Utility Programs

Reading Program

Sentences are keypunched by the 026 Printing Punch in the standard IBM punched-card code in columns 10-44 and 46-80. The printed interpretation is placed at the top of each card. Columns 9 and 45 contain the algebraic signs of the full-word values contained in the left and right-half card rows.

In the standard IBM punched-card code, each alphabetic character is represented in a single column by a double-punch consisting of a "zone" punch in either the zero, 11 or 12 row, and a numeric punch in one of the remaining rows. Figure 4 shows how the numerals 0-9 and the letters A-Z are represented on a standard IBM punch card.

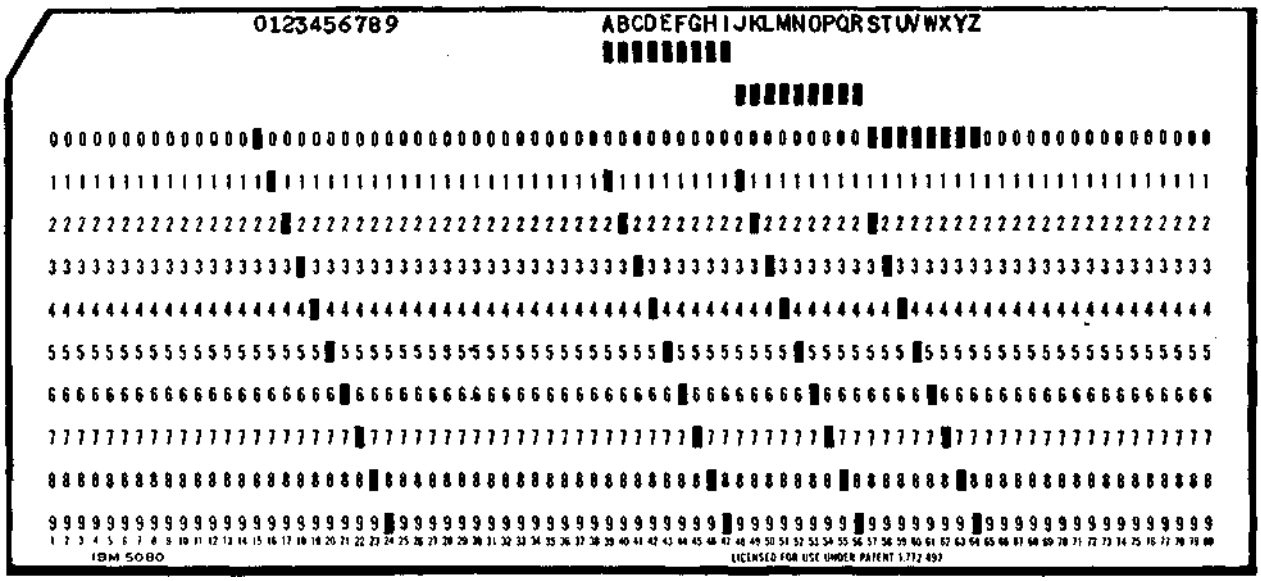


Figure 4

It may be observed in Figure 4 that the decimal equivalents (11-36) of the alphabet letters may be obtained by adding a 10 to the numeric punch value when the zone punch is 12, a 19 when the zone punch is 11, and a 27 when the zone punch is 0.

Conversion of each alphabetically punched sentence card occurs as follows. The machine is programmed to scan across the card and convert successive 5-column fields, storing the results in successive full-word storage locations.

In a 5-column card field there are twelve 5-bit arrays - one array in each card row. Each 5-bit array falls in the range of values 00000-11111. Therefore each 5-bit array may be used as the argument of a full-word table look-up, defined by the simple transformation

(1) $b_1b_2b_3b_4b_5 \longrightarrow 000000b_1\,000000b_2\,000000b_3\,000000b_4\,000000b_5$

When a sentence card is read, each 5-bit array in a 5-column card field is transformed into a 35-bit array as defined by (1). Associated with each 5-column card field is a double-punch blank-column (DPBC) register, a numeric conversion register, and three "zone" registers.

The DPBC register stores the sum of the functional values (1) corresponding to the arrays in rows 9-1. If a card field has been punched alphabetically in each column (zone punch and a numeric punch), after card reading of rows 9-1 the contents of the DPBC register will appear as follows:

0000001	0000001	0000001	0000001	0000001

The numeric conversion register stores the 35-bit numeric conversion of a five-column field. This is obtained by summing successive partial sums of the functional values (1) corresponding to the arrays in rows 9-1. Thus if a 5-column card field contains the numeric punches 58638, the contents of the numeric conversion counter will appear as follows:

0000101	0001000	0000110	0000011	0001000

The "zone" registers are reserved for storing the functional values (1) corresponding to the 0-row, 11-row, and 12-row arrays.

Since the 0-row is punched for the numeral zero as well as for the alphabetic characters S-Z, provision must be made for the machine to discriminate between these two cases. This is done by using the contents of the "0-row" register as an extraction mask with respect to the contents of the DPBC register.

The 0-row extracted value of the DPBC register is multiplied by 27. The contents of the "11-row" register are multiplied by 19 and the contents of the "12-row" register are multiplied by 10. All three products are then added to the contents of the numeric conversion counter to obtain the final conversion for the 5-column field.

The above outlined basic conversion cycle is repeated for each sentence card. A blank card is used to separate sentences. When the machine encounters the blank card, it transfers control to the lexical syntax subprogram.

Each binary card image is reconverted to a decimal card image which is checked row for row against the original "built-up" decimal card image. Sentence loading is

thereby checked during each card-reading cycle.

Printing Program

The 716 alphabetical printer unit is programmed to print out 14 full-word locations of translation per line without the "echo-checking" feature, which is unnecessary in this program. Part of the printing routine, that involving the construction of each decimal card image to be printed, is also utilized in the reconversion-checking subroutine of the reading program.

ACKNOWLEDGMENT

The author wishes to express his deepest gratitude to Professor Leon Dostert and Dr. Paul Garvin of the Georgetown Institute of Languages and Linguistics for their splendid cooperation in connection with every detail of analysis and programming relating to this joint project. Grateful acknowledgment is also made to Mr. Tom Steel for his very valuable assistance in connection with the input-output conversion routine.