Computer transcription of Stenotype code offers the possibility of producing English text from speech via the Stenotype keyboard in real time. Reported are experiments directed toward designing a time-shared Stenotype transcription system that makes use of earlier work in Stenotype dictionaries and language processing.

A content-addressing algorithm for direct-access storage, requiring a single access per retrieval, is presented. An existing experimental Stenotype dictionary program is used to implement on System/360 this algorithm and a dictionary-compaction technique. Transcription analysis indicates that the experimental design can reduce the average transcription error rate to six percent.

A structure for real-time Stenotype transcription

by J. W. Newitt and A. Odarchenko

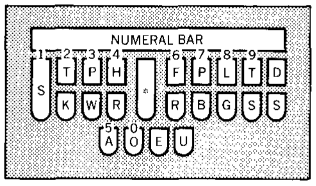
The Stenotype technique is the only presently known method for recording spoken language in hard copy at conversational speeds. This suggests the possibility of machine transcription of Stenotype code into English clear text at conversational speeds. Machine transcription could thus increase the productivity of typists, and thereby, provide a man-machine interface that is competitive with other forms of keyboard input.

In the late 1950's and early 1960's, work began in the automating of Stenotype transcription concurrently with research in automating language translation. In 1959, G. Salton[1] described his work at Harvard University that led to the first Stenotype transcription program. In the same year, E. Galli[2] began developing a Stenotype-to-English transcription program and dictionary for a special-purpose language processing computer.[3] By 1960 Galli had developed a workable system, and by 1964, he had extended the Stenotype rules, abbreviations, and practices to eliminate most of the ambiguities that existed in the language. The Stenotype dictionary-transcription program evolved to a point that table lookup translation principles could be applied. (Because the dictionary and transcription program being discussed in this paper form a single logical unit, it should be thought of as a *dictionary program.* We refer to it as a "dictionary" when emphasizing that attribute, and "dictionary program" when its dual nature seems appropriate.)

In 1965, a more comprehensive Stenotype transcription dictionary program that had several notable features was developed for a new language-processing computer[4] for further research in machine language translation. Fewer restrictions were placed on the input subject matter. Error rates were between two and ten percent of the words transcribed. Computer throughput was 60 words per second, theoretically sufficient to transcribe the output of 30 Stenotype reporters. This dictionary program, designed for a specialized computing system, was used successfully by a government agency[5] for one year, after which it was converted to operate with System/360 in a batch-processing mode.

The objective of the work discussed in this paper is to extend the file-addressing concepts of the language processor to a System/360 configuration using direct-access storage. The experimental work makes possible the production of English text from speech via the Stenotype keyboard in real time. The text can be produced as hard copy or possibly on an interactive display console for immediate on-line editing. Basic Stenotype concepts and machine processing of Stenotype code form the background for discussing a new System/360 file organization and retrieval procedure for the dictionary program. The new organization permits the content addressing of files of variable-length textual data and minimizes file accesses, processing time, and required storage. The new file organization was programmed and storage requirements are given. Results of experimental transcription of typical Stenotype notes are presented.

Figure 1 Layout of the Stenotype keyboard

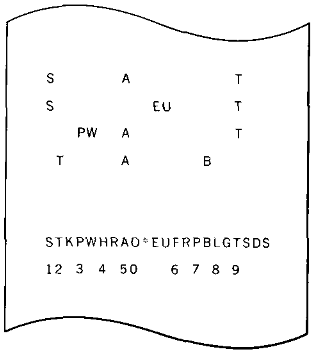


Basic concepts

Stenotype system

The Stenotype machine is a touch-driven printer about half the size of a typewriter on which any combination of its twenty-two keys can be struck simultaneously. Each key controls printing in a specific column of a paper tape. The Stenotype keyboard layout is shown in Figure 1 and a sample of the printing is shown in Figure 2. There is no horizontal movement of the platen or type. Vertical motion, however, is provided by a line feed that advances the paper when the struck keys are released. Each line of printing is called a *stroke*. Depressing the numeral bar, shown in Figure 1, causes the keys to shift so that numbers are substituted for letters on some keys.

Figure 2 Sample of Stenotype printing



The Stenotype language is phonetic, and each stroke represents a syllable. Many stroke patterns and some individual keys, however, are used to abbreviate common words, phrases, and affixes. A meaningful collection of strokes (representing a word, for example) is called a *form*. In Stenotype practice, there are many forms with several possible English translations, and there are no word boundaries.

Consonants are represented by unique combinations of keys actuated on both sides of the keyboard by the fingers. The thumbs operate the centrally positioned vowel keys. Thus, syllables, which most commonly occur as consonant-vowel-consonant combinations, can be printed by a single stroke of the hands.

Stenotype processing

Galli's original Stenotype dictionary program forms the basis for our research. This 80,000-entry, 3.4-million-byte dictionary requires at least one retrieval for every word translated. Each dictionary entry consists of two parts: an *argument*, which is the Stenotype input, and a *function*, which is its English equivalent. Arguments are variable in length and are ordered from low to high, i.e., A to Z, and from short to long word length, as are words in an ordinary dictionary.

The System/360 retrieval algorithm will be discussed in detail later in this paper, but first we introduce Galli's use of the language processor for Stenotype transcription. Although the language-processor algorithm is a multi-step, optimized procedure, it is equivalent to a serial comparison between the input and each successive argument beginning at the end (Z) of the dictionary and continuing until a match is made. Since the Stenotype input has no word boundaries, it must be treated as a string of symbols of undetermined length. By serially comparing this string with the dictionary entries in high-to-low order, the longest form that equals a dictionary argument is matched first. This is the *longest-match* principle. At least one match always exists, since an entry with a single-letter argument, called a *breakpoint*, is included in the dictionary for every possible input character.

A major problem for our System/360 dictionary-program implementation was the indeterminate number of entries that must be compared by the language processor in achieving a match. There was no known means for segmenting the dictionary program into fixed-length records and for providing indexing so that, for each input, only one record needed to be retrieved to assure a match.

To illustrate this problem, consider the following example, which contains an error and must match to a breakpoint entry:

TXATION WITHOUT REPRESENTATION

The procedure begins as one would search a conventional dictionary from the back, starting at a word beginning TY . . . . It would involve unnecessary searching to begin at a higher level than TY. Each subsequent argument is compared to TXATION until the breakpoint argument, T, is reached. This implies scanning almost every entry beginning with T. Thus, for a computing system using direct-access storage, many file accesses may be required to retrieve the record that contains the matching entry.

Analogous to the fetch-execute cycle of data processing systems, the language-processing computer, which executes the Stenotype dictionary program, has a fetch-compare cycle that operates (on a character-by-character basis) between the dictionary entry and the input. This cycle is controlled and interrupted by instructions in dictionary-program entries themselves.

One such instruction performs the operation of matching to whatever character is opposite it in the input. (Called "gamma," this instruction is denoted in the dictionary program by a comma.) We illustrate the fetch-compare cycle first without and then with the gamma instruction.

First, consider an input BAABG and the following set of stenotype dictionary entries:

Argument	*Function*
BAABG	BAKE
BAUBG	BALK
BA=BG	BACK

Comparisons start with the last argument (BA=BG) in which a mismatch occurs on the third character. (Machine Stenotype orthography is beyond the scope of this paper. However, argument-function identities are given where necessary.) The comparison then proceeds to the next lower argument (BAUBG) where a similar mismatch occurs. The search proceeds in this way until a match with the argument BAABG occurs. The associated function represents the correct transcription of the input. Thus, BAKE, BALK, and BACK are distinguished in the argument by the form of the A-vowel used.

Next, consider the work MAKE. Here, neither of the forms MAUK nor MACK are words. Also, some Stenotype reporters do not distinguish between long and short vowels when there is no ambiguity, and they write MAKE with a short A. This would be transcribed as MACK if the previous procedure were followed. One way to correctly transcribe MAKE is by storing a dictionary entry for each form of the A-vowel as follows:

Argument	*Function*
MAABG	MAKE
MAUBG	MAKE
MA=BG	MAKE

Such repetition is obviously wasteful of storage.

By using the gamma instruction (,), the same result can be accomplished more compactly because only the following single entry is required.

Argument	*Function*
MA, BG	MAKE

Thus, regardless of the vowel forms used by the stenotypist to write MAKE, it is transcribed correctly.

In our restructuring of the dictionary program for System/360 use, we devised a new search algorithm that avoids serial file accessing. Used with direct-access storage devices, the new dictionary organization-and-search algorithm has the following characteristics: equivalency with the longest-match algorithm, entry grouping by records of arbitrary size, and a single file access for retrieving any entry.

We devised such an algorithm, which we discuss first for an uncompacted dictionary in combination with an associated directory or index. Then the compaction and search procedures are presented. Results obtained when the compaction techniques are applied to the Stenotype dictionary-program are shown.

System/360 dictionary and search algorithm

the directory

The overall logic of our dictionary file organization and search algorithm involves a *directory* and dictionary *records*, as shown in Figure 3. The directory consists of elements A, B, C, ... with pointers 0, 1, 2, ... to System/360 records A, B, C, ... Each record contains many entries. Directory elements are based on the argument of the first entry in each dictionary record. In retrieval (dashed arrows), the directory is searched until a directory element equal to or just greater than the input is found. Then the associated dictionary record is retrieved, and each entry in that record is compared with the input until a match is found. This operation is effectively that of content addressing.

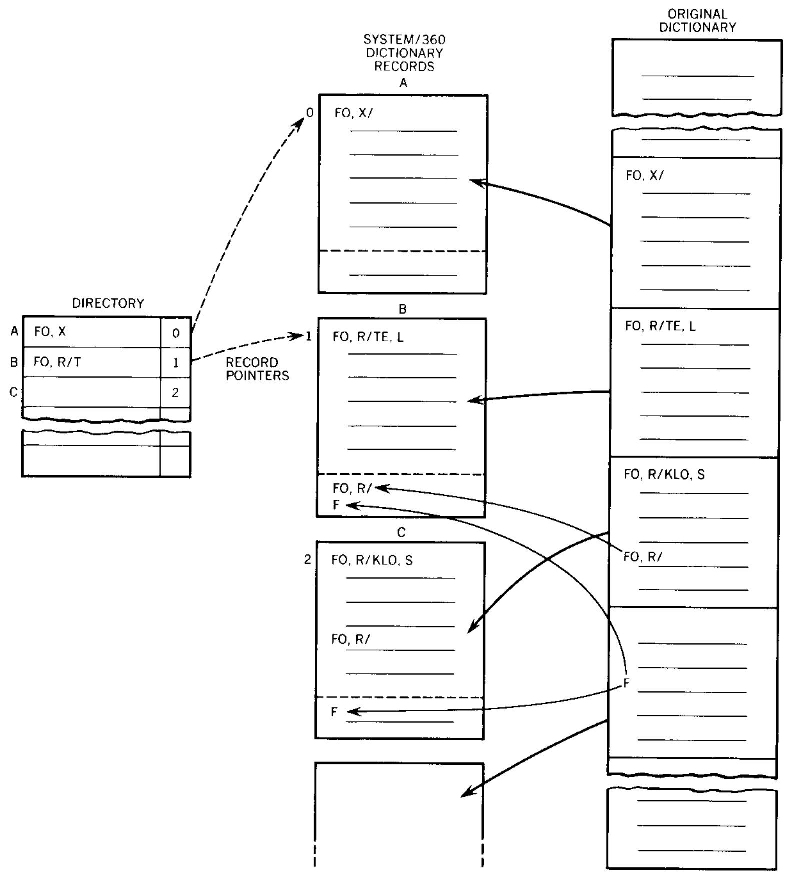
The directory elements are formed by taking as many bytes from the argument of the initial entry in the record as are required to distinguish it from the initial entries in adjacent records. Construction of the dictionary (solid arrows) begins by specifying the maximum allowable record length. Records are then filled with consecutive dictionary entries except for approximately one-hundred bytes at the end of each record that are reserved for duplicate entries that may have to be added later.

To illustrate the construction of a directory element, shown in Figure 3, consider the three successive dictionary records that have as first entries the following:

Argument	*Function*
FO,X/	FOX
FO,R/TE,L	FORTEL
FO,R/KLO,S	FORCLOSE

Four bytes are needed to distinguish FO,R/TE,L from FO,X/, and six bytes to distinguish it from FO,R/KLO,S. Six bytes are chosen, and thus FO,R/T becomes the directory element.

Figure 3 Construction and retrieval from the System/360 dictionary program



After the directory element has been determined, some entires may have to be added to the System/360 dictionary records. These are entries in subsequent dictionary records whose complete argument is an initial substring of the directory element. By repeating entries that are substrings, we assure that in retrieval any entry that matches the input is included in the record retrieved. These entries are called *short matches*, and characteristically there are one or two for each record.

For the directory element FO,R/T the initial substrings are the following:

FO,R/T
FO,R/
FO,R
FO
F

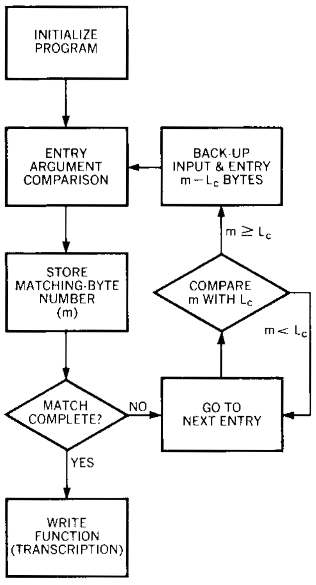
Of these, only FO,R/ (function FOR) and the breakpoint entry F (function F), added to record B of Figure 3, are short matches that have to be added to the record. In anticipation of these short matches, the extra bytes are allowed when the record is first constructed.

example of record retrieval

Consider the following hypothetical directory for a conventionally alphabetized dictionary:

Directory element	*Record number*
A	0
AB,CD	1
AB,CE	2
AB,E	3
AB,FGH	4
AB,FGJ	5
AB,G	6
AB,HAA	7
AB,HAB	8
AC	9
B	10
. . .	. . .

Figure 4 Transcription using the compacted dictionary program



INITIALIZE PROGRAM
ENTRY ARGUMENT COMPARISON
BACK-UP INPUT & ENTRY m − L_c BYTES
m ≥ L_c
STORE MATCHING-BYTE NUMBER (m)
COMPARE m WITH L_c
m < L_c
MATCH COMPLETE?
NO
GO TO NEXT ENTRY
YES
WRITE FUNCTION (TRANSCRIPTION)

In locating the record to be retrieved, the input is compared with each directory element starting with an element that is near the bottom of the list and is greater than the input (in the sense that A < B < ... < Z). Thus, comparisons for an input beginning with A start with the lowest ordered element beginning with B and proceed, element by element, until the directory element being compared is less than the input. At that point, the record whose directory element is just greater than the input is retrieved. On the other hand, if the directory equals the input, two records—the one equal and the next higher—are retrieved. Suppose the input is ABOUT. Then, for the above directory

AB,HAB < ABOUT < AC

is true, and record 9 with directory entry AC is retrieved. For an input ABSENT, the expression

AB,E = ABSENT < AB,FGH

is true, and records 4 and 3 are retrieved.

The directory search procedure, followed by a sequential search of the retrieved record entries, assures that the dictionary entry representing the longest match to the input is selected. Records are of arbitrary size, and their retrieval requires no more than one storage access.

dictionary compaction

A typical segment of Stenotype argument-function formats for language-processing computers are shown in Table 1. The search procedure for a compacted dictionary is shown in the flowchart in Figure 4. Entries in Table 1 are given in reverse alphabetical order; consequently, comparisons with an input proceed from top to bottom in arriving at the longest match. Redundancy in the original Stenotype dictionary is clearly illustrated by the redundancy in entry arguments. The format for System/360 processing eliminates redundancy and minimizes byte comparisons, as shown in the last two columns of the table.

The essential idea of dictionary compaction is to include in each entry the number of bytes that entry has in common with the preceding entry (L_c). When this is done, a comparison of identical bytes in successive entries need never be repeated. Instead, a single numeric comparison is substituted.

As an illustration, assume the following input:

TRAITT/$ITOOR/XYZ ...

In a serial search, comparisons would begin with the first entry in the second column, and the given input would match to the tenth entry, translating as TRAITOR.

Table 1 Comparison of Stenotype dictionary programs

	Stenotype entries for serial processing		*Entries for System/360 processing*	
Entry number	*Stenotype argument*	*English function*	*Compaction length L_c*	*Argument remainder*
1	TRA,TJ/KO,M/KA,L	TRAG-I-COM-I-CAL	0	TRA,TJ/KO,M/KA,L
2	TRA,TJ/KO,M/D,Q/	TRAG-I-COM-E-DY=	12	D,Q/
3	TRA,TJ/KO,M/$,MI,BG/	TRAG-I-COM-IC=	12	$,MI,BG/
4	TRA,TJ/KA,L	TRAG-I-CAL	8	A,L
5	TRA,TJ/D,Q	TRAG-E-D	7	D,Q
6	TRA,TT/$,TRU,S,	TRAI-TOR-OUS	5	T/$,TRU,S,
7	TRA,TT/$,TRI,S,/	TRAITRESS=	11	I,S,/
8	TRA,TT/$,TRE,S,/	TRAI-TRESS=	11	E,S,/
9	TRA,TT/$,TO,R/$,RU,S,	TRAI-TOR-OUS	10	O,R/$,RU,S,
10	TRA,TT/$,TO,R	TRAI-TOR'S4'	13	
11	TRA,TT/$,TE,R	TRAITOR'S4'	10	E,R
12	TRA,TJ/$,JI,BG	TRAG-IC'S4'	5	J/$,JI,BG
13	TRA=TJ/$,JIQ/D,Q	TRAG-E-D'S8'	3	=TJ/$,JIQ/D,Q
14	TRA,TJ/$,JE,/D,Q	TRAG-E-D'S8'	10	E,/D,Q
15	TRA,TJ/$,J,Q	TRAG-ED'S8'	10	,Q
16	TRA,TT	TRAIT'S4'	5	T
17	TRA,T,	TRAIT'S4'	5	,
18	TRAASS/RU,S/	TRAC-ER-IES=	3	ASS/RU,S,/
19	TRAASS/R,Q	TRAC-ER	8	,Q
20	TRAASS/$,SEE=SZ/	TRACES=	7	$,SSEE=SZI
21	TRAASS/$,S,E,R/$,R,Q	TRAC-ER	10	,E,R/$,R,Q
22	TRAASS/$,S,E,R	TRACER	14	
23	TRAASS/$,S,A,BL/	TRACEABLE=	10	,A,BL
24	TRA=SS	TRASS=	3	=SS/
25	TRAISS	TRAC	3	ISS
26	TRAASS	TRAC	3	ASS
27	TRA,SS	TRAC	3	,SS
28	TRA,RB/YE,S,/--T/	TRASH-I-EST=	4	RB/YE,S,/--T/
29	TRA,RB/YE,R/	TRASH-I-ER=	10	R/
30	TRA,RB/YE,F,TT/	TRASH-I-EST=	10	F,TT/

Table 2 Dictionary-program compaction

Original Stenotype dictionary program	3,411,000 bytes
Compacted System/360 dictionary program	2,580,000 bytes
Added by short matches	13,000 bytes
Resultant System/360 dictionary program	2,593,000 bytes

We now consider the compacted entries in the last two columns and use the compacted-dictionary search algorithm for System/360 shown in Figure 4. The number of bytes (m) in the first entry that correctly match the input is stored ($m = 5$). Since this is not a complete match, the search continues to the second entry where the number of matching bytes (m) is compared to L_c, which is 12. Since $L_c = 12$ is greater than $m = 5$, the procedure goes to the third entry. Not until the sixth entry is it necessary to make another character comparison. This is followed by five more comparisons as a result of which the matching-byte number is increased to 10, and a mismatch occurs. Entries 7 and 8 are bypassed because L_c is greater than m. Entry 9 increases m to 14 before mismatching. Then for the tenth entry, the input and entry pointers are backed up one character before comparison starts. Entry 10 immediately matches the input because no more bytes are found in the argument remainder.

Experimental results

To evaluate the practicality of on-line Stenotype transcription, we programmed our dictionary-file organization. In addition, we tested the dictionary program developed for the language processor using Stenotype notes made by practicing reporters. Dictionary-program compaction reduced the dictionary size by 24 percent, as shown in Table 2. A major concern was how many bytes would be added by the short-match entries. Based on the compacted dictionary, experimental directories were programmed for several record sizes. For a record size of 3,000 bytes, the directory contains 860 elements (4,260 bytes) or an average of 5 bytes per directory element. We found (as shown in Table 2) that, under these conditions, only 13,000 additional bytes had to be added to the dictionary for short-watch entries. Thus it is shown that the short matches contribute a relatively small addition to the dictionary-program (which was the key unknown factor in the System/360 implementation).

Transcription testing was done using the original dictionary program on the language processing computer with no modifications and with no special training of stenotype reporters. In this test, we transcribed and analyzed samples of Stenotype notes taken by three different reporters at public hearings. There were 27.3 percent word discrepancies between the machine-transcribed output and that produced by the Stenotype transcribers.

An analysis of the results of the transcription tests are summarized in the first data column of Table 3. It was found that most of the errors fall into categories that can be individually studied and improved. Typical of these improvements are dictionary addition and deletions and special training of Stenotype reporters. That is, there were no fundamental flaws in Galli's extended Stenotype language or in the procedures developed to translate it. Only 1.5 percent of the words in error could be attributed to inadequacies in the basic transcription algorithm.

We first discuss special reporter training to avoid the homograph errors given in Table 3. *Homographs* are a characteristic of the conventional Stenotype language wherein one Stenotype form stands for more than one English word. There are several causes of homographs: (1) homonyms, (2) ambiguous word boundaries, and (3) ambiguities in conventional Stenotype language. Stenotype reporters resolve these ambiguities by their context.

Consider first homographs caused by homonyms such as "steel" (transcribed as "steal") and "daze" (transcribed as "days"). Errors due to homonyms have remained most resistant to correction and account for most of the 1.5 percent residual homographic errors. To the extent that the dictionary program can resolve homonyms, it is done probabilistically, i.e., "days" is printed rather than "daze."

Typical of word-boundary homographs are those based on expressions such as "sell fish" (transcribed "selfish") and "agent sees" (transcribed "agencies"). The resolution of word-boundary ambiguities is accomplished to a large degree in the Stenotype dictionary program with full-phase arguments such as "to sell fish" and "a selfish." Referring to the third cause of homographs, consider the words "sink," "sing," and "singe" in which no distinction is made among the final consonants in the conventional Stenotype language. Stenotype extensions correct this problem by introducing new keyboard fingerings for —NK and —NG.

Another example of the third source of homographs is that both the left hand T and right hand T, as shown in Figure 1, are commonly used to abbreviate either "it" or "the." Called "paired abbrevia-

Table 3 Experimental Stenotype transcription word errors

Error classes	*Initial analysis (percentage)*	*Corrected analysis (percentage)*
Homograph	13	2
Missing dictionary entry	8	~0
Proper name	2	~0
Unedited words	4	4
Total	27	6

tions," such homographs are resolved in the extended Stenotype language by assigning one abbreviation to each hand—the right T for "the" and the left T for "it."

Language extensions, such as those discussed, may involve reporter retraining. For example, we found that many of the paired abbreviations that were resolved one way by extensions to the dictionary-program were often resolved in the opposite way by practicing reporters. Such discrepancies can be corrected either by personnel retraining or by tailoring dictionary entries to transcribe paired abbreviations as each reporter expects them to be.

The major portion of the homographic errors shown in Table 3 were found to be correctable by the procedures just discussed. Of the thirteen percent errors due to homographs, it is possible to approach seven percent correctability through retraining and four percent correctability through improvements in the dictionary-program. The residue of two percent uncorrectable homographs is due primarily to homonyms.

Errors caused by missing dictionary entries can be made to approach zero by appropriate additions. Several available methods enable a reporter to enter proper names so that they can be correctly transcribed.

There is a four percent error residue that the transcriber corrects by editing his tape prior to transcription. Providing a machine facility to correct such errors requires a system configuration that is beyond the scope of this experiment. Transcription requires a comparable post-transcription editing time to correct similar errors. In machine transcription, there is also an additional residue of errors requiring editing of a little over one percent, resulting from logical inadequacies, that have so far remained resistant to improvements in the basic algorithms.

Concluding remarks

We have experimentally demonstrated the technical feasibility of on-line Stenotype transcription for System/360 using a compacted dictionary program and a search algorithm for content addressing variable-length dictionary arguments. Methods of reducing word errors from twenty-seven percent to approximately six percent are discussed. Retraining in Stenotype extensions as well as modifications of the dictionary organization algorithm permit this reduction in the word-error rate.

CITED REFERENCES

1. G. Salton, "The automatic transcription of machine shorthand, *Proceedings of the Eastern Joint Computer Conference* **17,** No. 16, 148–159 (1959).

2. E. J. Galli, "The Stenowriter—a system for the lexical processing of stenotypy," *IRE Transactions on Electronic Computers* **EC-11,** No. 2, 187–189 (April 1962).
3. J. L. Craft, E. H. Goldman, and W. B. Strohm, "A table look-up machine for processing natural languages," *IBM Journal of Research and Development* **5,** No. 3, 192–203 (July 1961).
4. D. M. Bowers and M. B. Fisk, "The World's Fair machine translator," *Computer Design* **4,** No. 4, 16–29 (April 1965).
5. J. Hanlon, "CIA helps develop Stenotype reader," *Computer World* **2,** 46, 9 (November 13, 1968).