A BOOLEAN MATRIX PROCESSOR
WITH APPLICATIONS TO HANDWRITING RECOGNITION

by
LESTER DONALD EARNEST
S.B., CALIFORNIA INSTITUTE OF TECHNOLOGY
(1953)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF
SCIENCE
AT THE
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
MAY, 1960

Signature of Author

Department of Electrical Engineering
May 22, 1960

Certified by
Thesis Supervisor

Accepted by
Chairman, Departmental Committee
on Graduate Students
A BOOLEAN MATRIX PROCESSOR

WITH APPLICATIONS TO HANDWRITING RECOGNITION

by

LESTER DONALD EARNEST

Submitted to the Department of Electrical Engineering on 22 May 1960
in partial fulfillment of the requirements for the degree of Master
of Science.

ABSTRACT

A computer that processes rectangular arrays of binary elements,
rather than numbers, has been simulated using an existing machine. Such
a computer greatly facilitates pattern recognition work. The experi-
mental system has been used to investigate certain approaches to the
recognition of cursive writing, including:

a) a scheme for locating the upper and lower bounds of the
"small" letters ("a", "c", "e", etc.)

b) a data reduction technique that facilitates the detection
of salient features (e.g., long upward strokes, or tops of "t"s).

c) an algorithm for the extraction of individual "strokes"
from the written pattern, yielding a list of end point locations and
orientations.

Thesis Supervisor: Jack B. Dennis
Title: Assistant Professor of Electrical Engineering
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>I</th>
<th>INTRODUCTION ........................................</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1 History .......................................</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1.1.1 Pattern Recognition .......................</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>1.1.2 Character Recognition .....................</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>1.1.3 Script Recognition .........................</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1.2 Summary of Author's Work ....................</td>
<td>9</td>
</tr>
<tr>
<td>II</td>
<td>PROCESSOR ............................................</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2.1 Configuration ..................................</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.2 Input/Output ..................................</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.3 Word Operations ................................</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2.4 Matrix Operations ................................</td>
<td>15</td>
</tr>
<tr>
<td>III</td>
<td>APPLICATIONS ........................................</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.1 Envelope Detection ............................</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3.2 Feature Extraction ............................</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.3 Stroke Extraction .............................</td>
<td>27</td>
</tr>
<tr>
<td>IV</td>
<td>CONCLUSIONS ........................................</td>
<td>34</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT ...................................</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>BIBLIOGRAPHY .......................................</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>APPENDIX - MAT I ....................................</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A.1 Word Operations ................................</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>A.2 Matrix Operations ................................</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>A.3 Input/Output ...................................</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>A.4 Configuration ..................................</td>
<td>51</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Console Display</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Connectivity Example</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Envelope Detection</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Pattern Density</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Feature Extraction</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Stroke Extraction</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Stroke Printout</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Stroke Extraction Flow Diagram</td>
<td>31</td>
</tr>
<tr>
<td>A.1</td>
<td>And/Or Operations</td>
<td>43</td>
</tr>
</tbody>
</table>

### LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>TX-0 Instruction Set</td>
<td>40</td>
</tr>
<tr>
<td>A.2</td>
<td>Matrix Instructions</td>
<td>45</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

The proven utility of electronic computing devices in processing numerical data and in manipulating symbols has stimulated interest in the possibility of applying similar devices and techniques to other fields, such as pattern recognition. The problem of recognizing printed characters, being of great practical interest, has been approached by a number of investigators, with varying success and generality. The recognition of cursive writing has received comparatively little attention. This is understandable, since such an undertaking has less direct utility and is clearly more difficult -- the connectedness of the letters and freedom of style presenting a challenge not found in printed characters. Recognition by comparison with templates (such as can be used with mechanically printed matter) is inapplicable. Aside from any direct utility that a script recognition system might have, the study of this problem offers an opportunity to develop new skills that, hopefully, may contribute to the evolution of general pattern recognition principles.

It is well known that a black and white monocular visual field can be represented by a suitable array of elements, each of which is entirely white or entirely black (as in a newspaper photograph). If a given field is represented by \( n \) elements, a pattern in this field may be defined as a Boolean function of the \( n \) variables. Each of the various configurations that are to be identified as the same pattern is represented by a term in the disjunctive normal form of the corresponding Boolean function. In order that a system may be made to recognize \( m \) distinct patterns, it is necessary (in effect) that the corresponding \( m \) functions be found and implemented. For the values of \( m \) and \( n \) that are of practical interest,
this is not a simple task.

In the experimental work reported below, for example, arrays of 8192 elements were used. With such arrays, the available number of Boolean functions is

\[
\begin{align*}
&8192 \quad 2465 \\
&2 \quad 10 \\
&2 \quad \approx 10
\end{align*}
\]

which gives considerable freedom of choice.

It may be argued, of course, that pattern recognition is not a problem in Boolean algebra. Considering the usual complexity of such problems, in fact, neither this nor any other branch of mathematics appears to offer a suitably concise and meaningful way of expressing what needs to be done, let alone how to do it. The lack of a convenient notation for pattern recognition work is certainly related to the lack of underlying theory (and, perhaps, vice-versa). There is no shortage of ideas on the subject -- psychologists, engineers, physiologists, and mathematical biologists have attacked the problem from various directions but unifying concepts are lacking. Since interest in extra-biological pattern recognition is relatively new, there is some hope for improvement.

1.1 HISTORY

1.1.1 PATTERN RECOGNITION

A number of devices for recognizing patterns have been investigated, in some cases as adjuncts to studies of neural nets. For example, Culbertson\(^1\) and others discuss ways in which certain hypothetical neurons might be connected to recognize patterns in a visual field. Clark and Farley\(^2\), Uttley\(^3\), Rosenblatt\(^4\), Mattson\(^5\), and Bledsoe
and Browning\textsuperscript{6} study various systems in which randomly connected elements "learn" collectively. Taylor\textsuperscript{7} argues for analog computational techniques, while Selfridge\textsuperscript{8}, Dineen\textsuperscript{9}, and Kirsch and his colleagues\textsuperscript{10} consider the use of digital computers. While there has been much imaginative and interesting work in this field, there is little agreement among the various investigators on either devices or techniques (or even terminology).

A number of the above investigators attempt to circumvent the problem of determining how a given pattern may be recognized by having the pattern recognition device do this job as well. In some cases, so-called "self-organizing" systems are used, in which certain connections between components are chosen randomly, unwanted connections being either suppressed or modified during a "learning" phase. There seems to be a feeling among a number of investigators that the ability to recognize patterns and the ability to learn are intimately related. At the same time, the use of self-organizing systems for existing computing jobs would appear to be grossly inefficient. (It is appalling to consider teaching such a machine discipline--let alone long division). While the goal of shifting the labor to the machine is commendable, the optimum balance between "built-in" and "self"-organization is not clear. It may be that the answer is strongly dependent on the chosen task and degree of specialization.

1.1.2 CHARACTER RECOGNITION

The recognition of characters has long been of interest, both as the classical pattern recognition problem and as a field with substantial commercial potential. A report by Stevens\textsuperscript{11} surveys much of the earlier work.
In situations in which it is possible to control the type face being used, fairly simple mechanizations suffice\textsuperscript{12-17}. Such systems tend to have difficulty when faced with misorientation, moderate distortion, or even (in most cases) slight changes in scale. Diamond\textsuperscript{18} has shown that a remarkably simple device can recognize handwritten numerals if the writer is constrained to position them around fixed pairs of dots.

Digital techniques having somewhat wider applicability are under study by several groups. Unger\textsuperscript{19,20} has done some interesting and important work with a simulated "spatial computer" similar to the one of the present study. He has developed a series of tests for sorting hand-lettered alphanumeric characters using, in part, the sequence of pattern edge orientations. A Russian group at the Institute for Precise Mechanics and Computing Techniques\textsuperscript{21} is experimenting with a photo-electric scanner that yields the edge sequence as its output.

A group at the University of Manchester\textsuperscript{22} has developed a system that breaks the pattern into components, then assembles these into units with enumerable characteristics, which are then compared with descriptions of meaningful characters. Bomba\textsuperscript{23} examines the occurrences of certain small patterns within the large pattern area. Sherman\textsuperscript{24} is developing a system based on the connectivity between pattern junctions and end points. Doyle\textsuperscript{25} uses a set of tests to formulate a numerical score for each of the alternatives, selecting the character with highest score.

The most accurate characterization of the present state of the art seems to be "catch-as-catch-can". There is little similarity between the various techniques, yet all have been reported as fairly successful for some set of inputs.
1.1.3 SCRIPT RECOGNITION

The recognition of cursive writing has received relatively little attention in the literature. The Manchester group has suggested that their system is potentially capable of script recognition, but do not indicate just how this would be done. Bledsoe and Browning were able to recognize some isolated script characters with their system, but the approach used does not appear to offer any feasible extension to the recognition of connected letter sequences.

Harmon has shown that a remarkably simple device can differentiate between a small number of words, provided the writer is suitably constrained. His device uses only twelve relays to decide which of ten words ("zero" through "nine") have been written with a stylus on an array of conductors. The logic senses the presence of lines in certain key regions and counts axis crossings. While this approach is well suited to the chosen task, an attempt to expand the vocabulary to, say, 100 words would require a seemingly prohibitive increase in complexity. Even so, this device appears to represent the most versatile script recognition system currently in operation.

Eden and Halle have made an initial classification of writing strokes, their goal being a simple characterization of latinized script. They suggest that the recognition problem can best be approached through analysis-by-syntheses. This feedback technique has been investigated by Halle and Stevens in connection with the analysis of speech, and has met with success in its initial application there.

The application of analysis-by-syntheses to script recognition calls for an initial "guess" to be made at what has been written. The various alternatives are then synthesized and compared with the sample under
examination, adjustments being made to obtain a "best fit" (neither the fitting criterion nor the exact feedback mechanism having been determined as yet). Ideally, the synthesis would take into consideration style peculiarities observed in preceding samples from the same writer.

1.2 SUMMARY OF AUTHOR'S WORK

The present work takes a step toward a system of the kind just outlined. The first concern is with the device to be used. Digital computers have the requisite flexibility, but the internal organization of existing machines is ill-suited to the processing of two dimensional arrays of binary elements. A large number of instructions are required and, consequently, programs run slowly.

A more appropriate configuration has been selected and, for purposes of experimentation, has been simulated using an existing computer. The simulated machine works directly with two-dimensional arrays of binary elements (as well as computer "words"). Using this machine, a method is given for locating the upper and lower bounds of the "small" letters (letters that do not have long vertical strokes), so as to establish the position and scale.

A fast scanning technique for identifying certain salient features of written words is described. Prominent vertical and horizontal line segments are extracted and tabulated. This operation appears to offer a feasible approach to the identification of some words and a "guess" at others (perhaps to serve as the first input to the synthesis operation). Finally, an algorithm for extracting individual "strokes" from a line pattern is described, the output being the location (x,y) and direction of both ends of each stroke.
II PROCESSOR

A computer that operates on either linear arrays of binary elements (i.e. computer "words") or rectangular arrays (Boolean matrices) has proven to be a convenient tool for script recognition work. The configuration outlined here (called MAT I), in effect, includes the capabilities of a machine that was advanced by Unger\(^1\) for the character recognition problem. Certain other operations are also included. The matrix operations were chosen with a view toward pattern recognition problems, and so are not particularly well suited to certain other problems involving Boolean matrices (e.g. graph theory).

The goal has been to develop a set of operations that would be efficient for a machine built for pattern recognition work. Since such a machine could execute "two-dimensional" commands in parallel, this kind of operation is emphasized. The use of operations that are inherently sequential over small subsets is avoided, even in cases where their use might be relatively efficient for existing machines.

For the purpose of experimentation, the matrix processor has been simulated using the TX-O computer*. An outline of the system is given here, with more detailed information included in the appendix.

\* TX-O is an experimental transistorized computer, originally designed and built at the M.I.T. Lincoln Laboratory and currently operating at the M.I.T. Research Laboratory of Electronics.
2.1 CONFIGURATION

Two kinds of memory are used: matrix storage and word storage, the computer program residing in the latter. Single address logic is used for operations on both computer words and on matrices. For the latter, an Operation Matrix (abbreviated OM) plays a role analogous to that of the accumulator in arithmetic operations.

Since the machine being used has but one control element, special instructions are used to transfer control back and forth between the matrix mode (which uses an interpretive routine) and the word mode (in which the instructions are executed directly by the hardware). In a real system of this kind, it probably would be profitable to separate these functions, permitting parallel operation.

The matrix dimensions in the experimental system are variable. Matrices of 64 x 128 elements have been found convenient for representing individual handwritten words (provided they are not too long). The storage allocation in current use provides for eight matrices (including the OM) plus 4096 computer words.

2.2 INPUT/OUTPUT

Thus far, on-line experimentation with the computer has been used principally. For this purpose, the contents of the OM are displayed on a 12 inch cathode ray tube, as shown in figure 2.1. The experimenter can select certain subroutines by pointing a light pen* at the appropriate numeral on the scope. For example, if the pen is pointed at the

* A photo-electric device whose binary output can be sensed by the computer.
FIG. 2.1
CONSOLE DISPLAY

SAMPLE WRITTEN WITH LIGHT PEN

THICKENED

CUSTERED

0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
lower "0", the OM is cleared (zeros everywhere) and a pen-tracking subroutine is entered, permitting the experimenter to write (deposit ones) in the OM.

As an input for handwriting, the scope and light pen provide an unrealistically clean sample (provided the writer has a steady hand). Lines are uniformly one to two elements wide, the only anomaly being an occasioned "ball point skip", when the pen is moved very fast. An attempt has been made to develop processing techniques that are compatible with both varying line width and occasional "noise". Some tests with artificially thickened samples are encouraging. Samples reproduced from, say, pencil-and-paper writing are expected to be far "noisier" than any that can be simulated readily at present, however. More realistic tests must await the completion of a planned photo-electric page scanner.

2.3 WORD OPERATIONS

The present instruction list of the TX-0 computer (which is given in the appendix) differs noticeably from that of most digital computers in just two ways:

a) there are only six addressable instructions,

b) the seventh is a microprogramming* instruction, which helps make up for the shortage.

*The binary digits in the address portion have individual meaning, for the most part, so that various transfers and logical operations between internal registers can be commanded, as well as an input or output operation. In many cases, several sub-commands can be executed in a single instruction.
An available assembly program permits a group of instructions to be defined as a macro-instruction with variable arguments, if desired. Macro-instructions have been defined for depositing a one at a specified point in the OM, for locating pattern extremes (the coordinates of the "one" nearest the specified OM boundary, with an appropriate convention in case of a tie), and for changing to the interpretive (matrix) mode.

2.4 MATRIX OPERATIONS

The matrix operations, which are executed by an interpretive program, are in many ways similar to certain logical operations that are sometimes provided in arithmetic machines. For example, it is possible to form the logical product or sum ("and" or "or") of each element of the OM with the corresponding element of some other matrix, leaving the result in the OM.

More generally, with a single instruction it is possible to combine any combination of the two corresponding elements just mentioned and the eight surrounding elements in the addressed matrix. For example, if an "or" is performed on the pattern of figure 2.1a with all the surrounding elements participating, any ones spread one position in all directions, yielding the thickened pattern of figure 2.1b. These operations may also be combined with logical inversion (complementing) of the addressed matrix, if desired.

An interesting operation that is readily performed in terms of the matrix "and"s and "or"s is the location of all boundaries between regions contain ones and those containing zeros (this is sometimes
called the "spatial derivative" or "Custer"* operation). The result of operating on figure 2.1b is shown in 2.1c. This function is accomplished by two matrix instructions.

Another useful instruction is "expand", which permits the pattern in the OM to expand indefinitely in certain specified directions, provided the corresponding elements of the addressed matrix contain ones. This facilitates the tracing of continuous patterns in cases where connectivity is of interest. For example, figure 2.2a shows a pattern containing several distinct lines. If a set of starting points, such as those of figure 2.2b, are placed in the OM and an appropriate three-instruction subroutine is entered, all points that are continuously connected to the starting point are extracted, as shown in figure 2.2c.

Instructions are provided for storing the contents of the OM in any given matrix and for sensing coincidence between the OM and any given matrix. Unconditional transfers permit alteration of the instruction sequence within the interpretive mode or in combination with a shift to machine language. Other instructions facilitate the masking out (setting to zero) of unwanted portions of the OM and provide a count of the number of ones in each row or each column, as desired.

While an instruction set of the kind outlined above takes a while to get used to, it is the experience of the author that the programming of matrix operations is not appreciably more complicated than programming for conventional computers.

*So named by the Bureau of Standards Group10 in honor of the late General (a one surrounded by ones is wiped out).
FIG. 2.2
CONNECTIVITY EXAMPLE
III APPLICATIONS

The ultimate goal of the present undertaking, as indicated in the introduction, is a script recognition system using some technique closely akin to analysis-by-synthesis. As presently envisioned, a photo-electric scanner together with paper handling equipment will provide the input, under control of the processor, and the following functions will be performed.

a) An initial processing program will extract written words from the input field one at a time (or, rather, sets of pattern elements whose connectivity or close spacing makes them appear to be words). Some initial elimination of isolated points and smoothing of the pattern will be performed, if needed by later processing.

b) An envelope detection scheme will attempt to locate the upper and lower bounds of the small letters, so as to determine the vertical position and scale.

c) By examining certain salient features of the written word (e.g., the presence of "tall" letters in certain locations), an analysis program will determine, up to some level of ambiguity, which word is under examination. A dictionary* will be used in this connection to reduce the number of alternatives.

d) In cases where ambiguities exist, resolution will be attempted by synthesizing the alternatives and comparing with the pattern, adjusting the synthesis parameters for "best fit". Consideration of context may also prove useful.

*Clearly, this requires a great deal of fixed, rapid access storage.
e) The output of this system would provide the input to some other process. The written matter might, for example, represent a problem to be solved. Alternatively, if the output were printed, the system could function as a partial replacement for secretaries (this is not advocated).

The initial processing program has received little attention thus far, since it does not appear to offer any fundamentally difficult problems (this view may, of course, prove naive). An envelope detection scheme that works well on a large class of words is reported below. A data reduction technique, also reported below, facilitates the detection of salient script features and offers a possible approach to making the initial "guesses" as to what has been written. As a first step toward the analysis-by-synthesis, a study of methods for characterizing writing strokes has been undertaken. A process which extracts individual "strokes" from a written word is described.
3.1 ENVELOPE DETECTION

In order to determine the vertical location and scale of a word in the input field, a program has been written to find upper and lower bounds on the central region. The basic approach used was to scan the pattern density in the vertical direction (i.e., examine the number of ones in each row as a function of the y-coordinate). This approach is not completely general, in that the word must be approximately horizontally oriented. Even so, it has worked well on nearly all words that have been tested.

A typical display of the bounds obtained from this operation is shown in figure 3.1a. As shown in figure 3.1b, even the subtle density variations of the word "little" are often properly sensed. A word which yields consistent failure, however, is "syzygy". As might be expected, the strong descending strokes are uniformly misidentified as the lower bound of the small letters.

The density functions corresponding to the samples of figure 3.1 are shown in figure 3.2, together with the selected bounds (i.e., all ones in the original patterns are pushed to the far left). Several methods for examining these density functions were studied. The use of the mean and standard deviation to establish the location and width of the envelope was considered, but was rejected on the grounds that the deviation gives too much weight to the extreme points (e.g. the tops of "t"s). A system that examined the cumulative distribution and selected certain percentiles was tested and found to be only moderately successful (still trouble with "t"s). The following method is the one used to obtain the accompanying figures and is the best found thus far.
FIG. 3.1

ENVELOPE DETECTION
FIG. 3.2

PATTERN DENSITY

"ENVELOPE"

"LITTLE"

"SYZYGY"
1) The maximum value of the density function is found. The sum of this value and the values of the rows immediately above and below establishes a reference value.

2) Beginning at the bottom of the matrix, successive rows are examined until a density is found which exceeds a specified proportion (currently $1/8$) of the reference value. The lower bound on the small letters is taken to be two units below this.

3) Skipping upward a few units (five to be exact), successive rows are examined until a density is found which falls below a specified proportion (currently $3/16$) of the reference value. The upper bound is taken to be two units above this.

It can be seen from figure 3.2b that a fairly subtle change in the density function for "little" could cause the sensing for the upper bound to miss the small notch and select the tops of the tall letters instead. In the case of "syzygy", on the other hand, the density function offers hope that a modified scanning logic may yield consistently good results. There are, in fact, at least four possible ways around these difficulties:

a) it might be possible to improve the scanning procedure, eliminating the problem at its source,

b) the positions of the bounds obtained for a given word might be compared with those of the words on either side, permitting discontinuities to be detected and corrected,

c) the recognition techniques might be adjusted to take account of possible anomalies of this sort, or

d) we could simply refuse to recognize such obviously poorly designed words.
3.2 **FEATURE EXTRACTION**

Once the envelope of the small letters is determined, a number of features become accessible. The width of the envelope provides a measure which can be compared with the length of the word to indicate the approximate number of letters. The presence or absence of strokes protruding outside the envelope in certain positions characterize the word to some extent.

In order to investigate how much information might be obtained in this way, a data reduction program has been written that facilitates the detection of prominent features. Basically, this process looks for predominantly horizontal or vertical line segments in various regions.* Figure 3.3a shows a two word group that was processed. The first step is to find all points in the pattern where there are three adjacent ones in a row, as shown in figure 3.3b (this is accomplished by a single matrix instruction). Similarly, points where there are three adjacent ones in a column are found, as in figure 3.3c.

The number of points in various regions of these reduced patterns are then compared with threshold values to yield the quantized printout of figure 3.3d. Specifically, the following steps are performed.

1) The envelope detection scheme is employed, with the original pattern, yielding the amplitude (envelope width; labeled "a" in the printout).

2) The overall length of the word is found and expressed in amplitude units ("n" in the printout).

3) The total number of ones in the pattern is divided by "n" to yield "d", the mean pattern density (number of ones per unit length).

---

*Bomba\textsuperscript{23} has used related techniques on the character recognition problem.
FIG. 3.3
FEATURE EXTRACTION
4) The envelope lines are used to divide the pattern into three regions--top, central, and bottom--which are examined separately. For some purposes, the central region is split at the center into two subregions. In each region of a given reduced pattern, a "moving window" is used to record the occurrences of line segments as a function of the x coordinate. Specifically, in each area of width a/4 (in the x direction), the total number of ones is compared with specified proportions of the mean pattern density, d, to determine which of the four levels (blank), 1, 2, or 3 is printed out. (The proportions of "d" currently being used for thresholds are 1/16, 3/16, and 5/16 for horizontal line segments and 3/16, 1/2, and 3/4 for vertical line segments). This process is repeated at intervals of a/8 in x, so that each succeeding sample overlaps half the area of its predecessor.

Reading from top to bottom in figure 3.3d, the six channels represent:

1) horizontal lines in the top region,
2) vertical lines in the top region,
3) horizontal lines in the upper central region,
4) vertical lines in the central region,
5) horizontal lines in the lower central region,
6) vertical lines in the bottom region.

Since there were no descending letters in the sample, the bottom row recorded all blanks. It can be seen that the "t" shows up prominently in the printout. Although the pattern of the printout resembles the original pattern, it is not a reproduction, since the various numbers represent the number of line segments with a given orientation and are not necessarily printed in a location corresponding to their appearance in the original pattern.
Even though the six lines of information represent an order of magnitude or two less data than the original pattern (based on bit counts), much of the pertinent information is retained. It would appear to be a fairly straightforward task (though not a simple one) to pick out salient features from this kind of data, so as to provide initial "guesses" for the synthesis procedure. It should, in fact, be possible to uniquely identify a number of words in this way (e.g., "a", "an", "the").

Even though the thresholds that were used in quantization process were proportional to the mean pattern density, the process, as described, would not be expected to be entirely independent of line thickness. A very thick vertical line, in effect, contains many short horizontal line segments. A way around this difficulty would be to first apply the "Custer" operation (figure 2.1). This would remove the interior points but leave the boundaries, which retain the "direction" information.

Several more elaborate printouts were examined. The inclusion of diagonal line segments with positive and negative slopes adds some useful information, but does not appear to provide much improvement.

3.3 STROKE EXTRACTION

An algorithm has been developed for extracting "strokes" one at a time from a written sample, as in figure 3.4. The process begins at the leftmost point of the pattern and produces connected line segments as long as new points are found. When a dead end is reached, a new sequence is begun at the leftmost point of the reduced pattern (from which all extracted segments have been deleted). As shown in the figure, succeeding strokes are permitted to pass through portions of the pattern that have already been extracted in order to reach new points
SAMPLE

FIRST STROKE

SECOND STROKE

REDUCED PATTERN

FIG. 3.4

STROKE EXTRACTION
ORIGIN = 19, 11 AMPLITUDE = 26

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Θ</th>
<th>X</th>
<th>Y</th>
<th>Θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.00</td>
<td>0.00</td>
<td>0</td>
<td>0.70</td>
<td>2.00</td>
<td>270</td>
</tr>
<tr>
<td>0.70</td>
<td>2.00</td>
<td>270</td>
<td>0.80</td>
<td>0.10</td>
<td>112</td>
</tr>
<tr>
<td>0.80</td>
<td>0.10</td>
<td>0</td>
<td>1.30</td>
<td>0.80</td>
<td>270</td>
</tr>
<tr>
<td>1.30</td>
<td>0.80</td>
<td>270</td>
<td>1.30</td>
<td>0.10</td>
<td>90</td>
</tr>
<tr>
<td>1.30</td>
<td>0.10</td>
<td>0</td>
<td>2.20</td>
<td>0.80</td>
<td>225</td>
</tr>
<tr>
<td>2.20</td>
<td>0.80</td>
<td>270</td>
<td>2.10</td>
<td>0.10</td>
<td>68</td>
</tr>
<tr>
<td>2.10</td>
<td>0.10</td>
<td>90</td>
<td>2.60</td>
<td>0.80</td>
<td>180</td>
</tr>
<tr>
<td>2.60</td>
<td>0.80</td>
<td>270</td>
<td>2.80</td>
<td>0.10</td>
<td>90</td>
</tr>
<tr>
<td>2.80</td>
<td>0.10</td>
<td>0</td>
<td>3.00</td>
<td>0.20</td>
<td>180</td>
</tr>
<tr>
<td>0.20</td>
<td>1.51</td>
<td>0</td>
<td>1.60</td>
<td>1.80</td>
<td>180</td>
</tr>
<tr>
<td>1.60</td>
<td>1.80</td>
<td>180</td>
<td>1.30</td>
<td>1.60</td>
<td>0</td>
</tr>
<tr>
<td>1.30</td>
<td>1.10</td>
<td>0</td>
<td>1.50</td>
<td>1.10</td>
<td>180</td>
</tr>
</tbody>
</table>

a) COMPUTER PRINTOUT

b) RECONSTRUCTION FROM PRINTOUT

FIG. 3.5

STROKE PRINTOUT
The output of the process, shown in figure 3.5a, is a list of strokes, giving the x, y coordinates and the direction of the line, \( \theta \), at the two end points. The origin of the coordinate system is taken to be the intersection of the lower envelope bound and a vertical line passing through the leftmost point of the original pattern. The x and y coordinates are scaled in amplitude units, so that a point within the envelope has a y-coordinate between 0 and 1. The angles are measured counterclockwise from the positive x-axis to the nearest multiple of 22.5°. A reconstruction from the stroke printout is shown in figure 3.5b.

The decomposition is not always the same as would be produced by a human, if asked to extract strokes. For example, the "short cut" across the bottom of an e is almost invariably extracted as a single stroke, leaving the loop to be picked up in a later sequence.

Even so, most of the information is retained in a rather concise form. Certain short cuts can be avoided by permitting less variation in direction for individual strokes. Unfortunately, this has the effect of breaking the pattern into smaller pieces, giving too much detail in most cases.

The method used is illustrated in the flow chart of figure 3.6. Beginning with the leftmost point of the reduced pattern (which initially coincides with the original), all points are found that can be reached by moving through the original pattern in some combination of three directions 45° apart. For example, considering "east" to coincide with the positive x-axis, the first expansion is made to the east, northeast,
BEGIN

REDUCED PATTERN = ORIGINAL PATTERN

ARE THERE ANY POINTS IN THE REDUCED PATTERN?

YES

STARTING POINT = LEFTMOST POINT IN REDUCED PATTERN

NO

FINISHED

EXPAND ABOUT THE STARTING POINT IN EACH OF 8 DIRECTIONS. THE DIRECTION WHICH YIELDS THE MOST NEW POINTS* BECOMES THE CURRENT FORWARD DIRECTION.

IS THE CURRENT STARTING POINT THE END POINT OF A PRECEDING STROKE?

YES

EXPAND IN FORWARD DIRECTION TO FIND END POINT. EXPAND BACKWARD FROM THIS POINT.

NO

EXPAND IN FORWARD DIRECTION TO FIND END POINT. EXPAND BACKWARD FROM THIS POINT TO FIND A NEW STARTING POINT. EXPAND FORWARD AGAIN.

"AND" THE LAST TWO EXPANSIONS TOGETHER TO FORM THE STROKE. DELETE ALL NEW POINTS* AND ISOLATED POINTS FROM THE REDUCED PATTERN.

WERE A SIGNIFICANT NUMBER OF NEW POINTS* FOUND?

YES

RECORD x, y COORDINATES AND STROKE ORIENTATION AT BOTH ENDS.

NO

END POINT BECOMES NEW STARTING POINT

*POINTS WHICH ARE COMMON TO THE MOST RECENT EXPANSION AND THE REDUCED PATTERN

FIG. 3.6
STROKE EXTRACTION
and southeast. Each of the eight possible direction combinations is examined and the central direction of the set yielding the most new points (points in the expansion that have not yet been deleted from the reduced pattern) is selected as the current forward direction.

The end of the stroke is found next. If the forward direction coincides with one of the cardinal directions (north, south, east, or west), the new point which is furthest in this direction is selected. If one of the diagonal expansions was used, the predominant cardinal direction is used. This direction is sensed by finding the smallest rectangle that will contain the selected expansion and comparing its height with its width.

If the stroke being extracted represents a continuation from a preceding stroke (i.e., the starting point was not selected as the leftmost point of the reduced pattern), the new end point is made the origin of a backward expansion. The resulting pattern is logically multiplied (anded) with the pattern from the forward expansion to yield a new stroke. This combination of forward and backward expansions tends to eliminate branches. For example, the forward expansion from "s" to "e", below, might include a branch to "b". The backward expansion might also include a branch, such as "a". The combination of the two yields the desired path. Branches which rejoin without violating the direction restrictions are retained, as shown.

If the current starting point represents the leftmost point of the reduced pattern, an attempt is made to lengthen the stroke by expanding
backward from the end point and taking the extreme as a new starting point. Processing is the same otherwise.

All new points that have been found are deleted from the reduced pattern. Following this, any isolated points (ones surrounded by zeros) are deleted from the reduced pattern. This tends to reduce the "false alarm rate", saving processing time. Next, it is sensed whether a significant number of new points were found (5 is the current threshold). If not, the leftmost point of the reduced pattern is selected as a new starting point and the process is repeated.

If a significant number of new points were found, the coordinates and stroke direction at the starting and ending points are recorded and the extraction process is re-entered using the end point as a new starting point.

The stroke direction is found by masking out all points outside a small square (currently 12 units on a side), centered at the given point, and sensing the predominant line segment direction by a process similar to the one used for feature extraction in the preceding subsection. For example, if the forward direction was northeast (45°), the numbers of occurrences of the following point patterns are found.

```
0°  22.5°  45°  67.5°  90°
```

The pattern which occurs most frequently determines the direction. If none of these patterns occurs, the direction is recorded as indefinite (this theoretical possibility has not been observed in any of the samples tested).
IV CONCLUSIONS

A Boolean matrix processor of the kind studied appears to offer a powerful and versatile tool for pattern recognition work. Judging from the performance of existing systems, the execution of any very complex recognition work on a production basis requires some such new approach. Perhaps as important as the increase in speed that a parallel matrix processor offers is the convenience of performing logical operations over entire patterns. In effect, the matrix instructions provide a more powerful notation for conceiving and describing processing functions. This makes these methods attractive even for conventional computers.

The handwriting envelope detection scheme that has been developed offers hope, despite imperfections, that just such a simple approach may reliably perform this function. It should be possible to reduce sensitivity to strong descending strokes by one of the methods indicated. More general approaches should be sought to cope with gross misorientation or envelope curvature.

The feature extraction system offers a fast (in terms of parallel matrix operations) method of scanning a written pattern for salient features. It should be possible to uniquely identify many words in this way, particularly if the vocabulary is restricted.

The stroke extraction technique offers an approach to a more precise characterization of the written pattern. For convenience of interpretation, the stroke data have been expressed in the coordinate system established by the envelope. Since the fundamental extraction technique is independent of the coordinate system, however, it could, in principle, be used to decompose any line drawing.
A major step yet to be taken is to face reality, in the form of writing samples from unsuspecting subjects. The use of the light pen as an input mechanism, while very convenient for on-line experimentation, has not exposed the system to many potential problems—things such as smudges, variable line width, and breaks in the pattern. Hopefully, this step will be taken soon.
ACKNOWLEDGEMENT

The author has benefited greatly from association with his Thesis Supervisor, Professor Jack B. Dennis, who has been a consistent source of help, inspiration, and ideas. A debt of gratitude is also owed Professor Morris Halle, not only for his part in original work on the problem, but for thought provoking discussions and sustaining encouragement.
BIBLIOGRAPHY


17. Evey, R. J., "Use of a Computer To Design Character Recognition Logic", *Proc. EJCC* (December 1959), PP. 205-211.


APPENDIX

MAT I

MAT I, the experimental matrix processor used in the study above, has been simulated on the TX-O computer. Operations on Boolean matrices are performed by an interpretive program, while operations on computer words are executed directly by the hardware. The matrix operations use single address logic and are stored one to a word (18 bits). A simulated program counter operates in the conventional manner. An Operation Matrix (abbreviated OM) plays a role analogous to that of the accumulator in arithmetic operations. In preparing programs for this processor, an available binary symbol tape permits the use of the interpretive instruction language and facilitates other coding functions (see below).

A.1 WORD OPERATIONS

Operations on computer words are performed by the TX-O hardware commands. This machine has 8192 words of core memory with a 6 microsecond cycle time. The interval between successive executions of most instructions is 12 microseconds. Current input/output equipment includes a high speed paper tape reader, flexowriter, scope, light pen, and analog-digital converter.*

*The addition of a magnetic tape unit and a photo-electric page scanner is planned. Also, the instruction set is to be quadrupled.
There are two principal internal registers: an Accumulator (AC) and a "Live Register" (LR). The current instruction set is given in table A.1. The "operate" instruction permits a large number of combinations of sub-commands, such as:

a) an input or output operation
b) clear the left, right, or both halves of the AC.
c) complement the AC,
d) cycle or shift the AC one position,
e) combine the AC and LR using one of the operations "add", "and", "or", or "exclusive or" (arithmetic sum, logical product, logical sum, or distinguish).

It may be noted that to "clear and add" some storage register requires two instructions, in general. A more complete description of the machine has been published.*

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Octal Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store Accumulator</td>
<td>sto</td>
<td>0</td>
<td>Store the contents of AC at the given address.</td>
</tr>
<tr>
<td>Store Live Register</td>
<td>slr</td>
<td>100000</td>
<td>Store the contents of LR at the given address.</td>
</tr>
<tr>
<td>Add</td>
<td>add</td>
<td>200000</td>
<td>Add the addressed word to the AC.</td>
</tr>
<tr>
<td>Load Live Register</td>
<td>llr</td>
<td>300000</td>
<td>Load the LR with the addressed word.</td>
</tr>
<tr>
<td>Transfer on negative</td>
<td>trn</td>
<td>400000</td>
<td>If the AC is positive, execute the next instruction; if negative execute the addressed instruction next.</td>
</tr>
<tr>
<td>Transfer</td>
<td>tra</td>
<td>500000</td>
<td>Execute the addressed instruction next.</td>
</tr>
<tr>
<td>Operate</td>
<td>opr</td>
<td>600000</td>
<td>Perform the input, output, or internal operations specified by the bits in the address portion.</td>
</tr>
</tbody>
</table>

In addition to the instruction set provided in the conversion program, a number of often used operate commands and macro instructions have been defined.* In addition to these, the following macro-instructions are provided.

**plot** - Deposits a one in the OM at the location specified by the contents of the AC (x in bits 0-8, y in 9-17), returning control to the following instruction. Matrix coordinates are used, not scope coordinates.

**north** - Finds the northernmost one in the OM and returns to the instruction following with the coordinates in the AC. In case of a tie, the eastern point wins.

**south** - Finds the southernmost point in the same way, ties going to the west.

**east** - Finds the easternmost point, ties going to the north.

**west** - Finds the westernmost point, ties going to the south.

**matrix** - Transfers control to the interpretive program, causing the next instruction to be considered a matrix operation.

All these macro-instructions require two machine language instructions.

The following working storage registers may be addressed:

**tt** - Beginning of the temporary storage registers used by the various subroutines (1008 are provided).

**bb** - Beginning of the buffer storage registers in which the results of the "xno" and "yno" matrix instructions (below) are placed.

### A.2 MATRIX OPERATIONS

The matrix operations, which are executed by an interpretive program, fall into several classes that differ in format. The "A" class instructions have the following bit configuration.

```
i    a   a   a   a    m   m
```
The i's specify the instruction, while the m's specify a matrix address. The OM has address 0 (so that if the coder omits the address, the OM is implied). The a's represent a set of sub-commands, called "activity bits".

The activity bits form a mask that determines which bits in the vicinity of a given element are included in the operation. Figure A.1 depicts the mapping for the logical operations ("and" and "or"). Up to 10 matrix elements (9 in the addressed matrix and 1 in the OM) are combined to form each output element. As shown, each output element may be a function of the corresponding elements in the OM (m) and the addressed matrix (c). In addition, the elements to the north, south, east, and west (n, s, e, and w) in the addressed matrix can be active, as well as the diagonal elements (nw, ne, sw, and se). Beginning with the leftmost activity bit (bit 3 of the instruction word), these bits control

\[
\text{nw n ne w c e sw s se m.}
\]

The coder may use these symbols in any combination (within a given instruction, the order is not significant). In figure A.1, "A" denotes a set of gates that control activity. Complementing, if any, is done in the box labeled "-". The "and" (\(\land\)) or "or" (\(\lor\)) operation is performed in the box so labeled.

In all operations, the matrices are, in effect, surrounded with zeros. Thus, if the OM is shifted upward, a row of zeros will appear at the bottom. If the operation involves a complement, the border elements become ones. If no activity is specified (activity bits = 0), the identity matrix for the given operation results. Thus, either of the "and" instructions used by itself will place ones throughout the OM; while either of the "or" instructions clear the OM.
INSTRUCTION

ACTIVITY

OPERATION MATRIX (OM)

ADDRESSED MATRIX

FIG. A.1

AND/OR OPERATIONS
Another instruction which warrants special attention is "Expand" (also an A class instruction), which may be used to sense connectivity. The activity bits specify in which direction(s) any ones in the OM may spread (the "c" and "m" bits have no meaning for this instruction). The pattern may spread only in those regions in which the corresponding elements of the addressed matrix contain ones (see figure 2.2). Expansion does not occur in all directions at once. It occurs first in any combination of the directions e, ne, n, or nw which are active, followed by active combinations of the set w, sw, s, se. In order to get the effect of simultaneous expansion in combinations from these two sets, then, it is necessary to iterate (see below).

The M class instructions use a matrix address, but no activity bits. The format is:

```
000 000 iii iii Omm mmm.
```

The W class instructions use a word address in the following format:

```
000 iii www www www www.
```

It may be noted that since only 12 bits are provided for the address, these instructions cannot refer to the upper half of memory. The 0 class instructions have no address and use the format

```
000 000 000 000 iii iii
```

The various matrix instructions are listed in Table A.2. The symbols "A", "M", and "W" are used to denote, respectively, some combination of activity bits, a matrix address, and a word address. It may be noted that certain octal codes are not used, permitting some expansion of the instruction set. If any undefined codes are encountered by the interpretive program, the computer will halt with the illegal instruction in the live register and the location from which it came in the accumulator. If it is desired to ignore such an instruction and proceed, the RESTART button may be depressed.
TABLE A.2

**MATRIX INSTRUCTIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Octal Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complement, Or</td>
<td>cor A M</td>
<td>500000</td>
<td>Form the logical sum of all active bits, using the complement (logical inverse) of the addressed matrix (M), leaving the result in the OM.</td>
</tr>
<tr>
<td>Or</td>
<td>or A M</td>
<td>400000</td>
<td>Form the logical sum, in similar fashion, without complementing.</td>
</tr>
<tr>
<td>Complement, And</td>
<td>can A M</td>
<td>300000</td>
<td>Form the logical product with the complement of the addressed matrix.</td>
</tr>
<tr>
<td>And</td>
<td>and A M</td>
<td>200000</td>
<td>Form the logical product without complementing.</td>
</tr>
<tr>
<td>Expand</td>
<td>exp A M</td>
<td>100000</td>
<td>Expand the pattern in the OM in the direction(s) specified by A in those regions where matrix M contains ones.</td>
</tr>
<tr>
<td>Mask</td>
<td>msk W</td>
<td>70000</td>
<td>The four registers W through W + 3 specify the coordinates of a rectangle ((x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}})), within which the OM is not to be changed, all other elements being set to zero. If the rectangle lies entirely outside the OM or does not exist (e.g., (x_{\text{min}} &gt; x_{\text{max}})), the entire OM is cleared.</td>
</tr>
<tr>
<td>Name</td>
<td>Symbol</td>
<td>Octal Code</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Bound</td>
<td>bnd W</td>
<td>60000</td>
<td>Sense whether there are any ones in the OM. If not, execute the next instruction. If so, deposit in word locations W through W + 3 the coordinates of the smallest rectangle that includes all ones in the OM, and skip the next instruction.</td>
</tr>
<tr>
<td>Transfer to TX-O</td>
<td>txo W</td>
<td>20000</td>
<td>Transfer control to the machine language instruction in location W (leaving the interpretive mode).</td>
</tr>
<tr>
<td>Transfer</td>
<td>tfr W</td>
<td>10000</td>
<td>Execute the instruction in location W next (still in the interpretive mode).</td>
</tr>
<tr>
<td>Compare</td>
<td>cmp M</td>
<td>200</td>
<td>Compare the OM with M. If they are identical, skip the next instruction. If they differ in any element, store the OM contents in M and execute the next instruction.</td>
</tr>
<tr>
<td>Store Matrix</td>
<td>stm M</td>
<td>100</td>
<td>Store the contents of the OM in M.</td>
</tr>
<tr>
<td>Y-Count</td>
<td>yno</td>
<td>2</td>
<td>Count the number of ones in each row of the OM and deposit the results in the buffer registers in order of increasing y-coordinate.</td>
</tr>
<tr>
<td>X-Count</td>
<td>xno</td>
<td>1</td>
<td>Count the number of ones in each column of the OM and deposit the results in the buffer registers in order of increasing x.</td>
</tr>
</tbody>
</table>
For convenience in coding, several instruction/activity bit combinations have been assigned symbols of their own, as follows.

\[
\begin{align*}
\text{all} &= 777008 \quad (\text{all activity bits of the addressed matrix}) \\
\text{lom} &= \text{and c} \quad (\text{load the OM with the addressed matrix}) \\
\text{lcm} &= \text{can c} \quad (\text{load the OM with the complement of the addressed matrix}) \\
\text{ano} &= \text{and c m} \quad (\text{"and" one-to-one}) \\
\text{oro} &= \text{or c m} \quad (\text{"or" one-to-one}).
\end{align*}
\]

Certain simple applications of the matrix instructions are now considered. Suppose that it is desired to locate all points where there are three ones in a row horizontally (as in figure 3.3b). If matrix \( M \) contains the pattern in question, this is accomplished by the instruction

\[
\text{and w c e M}.
\]

Consider next the problem of locating the boundary between regions containing all zeros and those containing all ones (i.e., the Custer operation of figure 2.1). It can be seen that two steps will suffice:

\[
\text{cor all M} \\
\text{ano M}.
\]

Corresponding to each element of \( M \) that has a zero in some adjacent position, the first instruction places a one in the OM. The second instruction then masks out those points that were not ones to start with.

Suppose, now, that it is desired to find all lines that are connected to some given set of starting points, as in figure 2.2. If the basic pattern is in matrix \( M \), the starting points are in the OM, and \( P \)
is an available matrix, the following loop may be used.

\begin{verbatim}
stm P 
exp all M 
cmp P 
tfr.-2
\end{verbatim}

The first instruction is used only once, to place the starting points in P. The "expand" instruction permits the starting points to spread in all directions, under control of M. The "compare" instruction senses whether there has been any change in the pattern. If so, the transfer is executed and the loop is repeated. If the patterns in the OM and P coincide, however, a jump is made to the instruction following the transfer, with the desired pattern in the OM (and P).

A.3 INPUT/OUTPUT

Several subroutines are provided to facilitate on-line communication between the experimenter and the machine. The MONITOR routine (entered at location 20) will cycle indefinitely, displaying on the scope the contents of the OM (the display raster is interleaved modulo 4 in the y direction). MONITOR also displays the numbers 0 through 7 at the top and bottom of the scope, permitting the experimenter to select certain functions by light pen. When the light pen is pointed at one of these characters, it brightens noticeably. As soon as the pen is removed, control is transferred to the corresponding subroutine:

Lower 0 - Clears the OM and permits the experimenter to write into it. Initially, random points are displayed on the scope until one is seen by the pen (this happens almost the instant the pen
is pointed at the screen). Thereafter, a small cross is displayed, marking the location at which the pen is pointed. At the location in the OM corresponding to the center of the pen, ones are deposited continually. When the pen is removed, MONITOR regains control.

1 - Permits writing without clearing (e.g., for dotting "i"s and crossing "t"s).

2 - Causes the contents of the toggle switch accumulator (TAC) to be executed as a matrix instruction. If TAC contains something other than a transfer instruction, MONITOR automatically resumes control.

The functions selected by the other numbers depend on the experiment. These functions are determined by a transfer table located in "mot" (lower "0") through "mot + 17" (upper "7"). Thus, if the programmer wishes to transfer control to location x whenever the light pen is pointed at the lower "5", he uses the symbols

\[
\text{mot + 5} | \text{tra x}.
\]

Numbers which are not "hooked up" leave MONITOR in control.

In the current set-up, the following functions may be selected:

Lower 3 - Performs the "Custer" operation on the OM (figure 2.1).

4 - Complements the OM (logical inverse).

5 - Provides an inverted display, without the rows of numbers, for use with the Land camera. The number of bars (rows) that are to be displayed before the computer halts is selected by TAC. Depressing RESTART repeats the display.
If, at any time, TAC is made negative, MONITOR regains control immediately.

6 - Finds the envelope of the small letters and deposits corresponding lines in the OM (figure 3.1).

7 - Selects the feature extraction printout (figure 3.3)

Upper

0 - Places the contents of matrix 1 in the OM.

1 - Places the contents of the OM in matrix 1.

2 - Selects a connectivity subrouting (figure 2.2). This subroutine is entered with the starting points in the OM and the mask in some other matrix (other than 2). TAC should contain an "Expand" instruction addressing the mask and specifying the directions in which the pattern is permitted to expand. The result is stored both in the OM and matrix 2.

3 - Selects a subroutine which fills-in single-bit gaps in the OM, so as to overcome the problem of "ball-point skip".

4 - Selects the subroutine which generates a display of pattern density as a function of y-coordinate (figure 3.2).

5 - Punches out on paper tape up to 6 matrices, beginning with the OM, in a format suitable for later read-in. TAC specifies the last matrix to be punched (if TAC lies outside the range 0-6, control returns immediately to MONITOR). The output is compacted by punching out only those words which contain non-zero elements. A subroutine is placed at the beginning of the tape to clear the matrix storage.

6 - Initiates the stroke extraction process (figures 3.4 and 3.5). If the Toggle Buffer Register (TBR) is negative, the
strokes are extracted and displayed, but printing is suppressed. If TAC is negative, control is returned to MONITOR after each successful stroke extraction. If TAC is positive, the extraction of successive strokes continues automatically, with each stroke being displayed the number of times specified by TAC (if TAC is made negative during the display cycle, control immediately reverts to MONITOR).

7  - Causes the next stroke to be extracted. Whenever the reduced pattern becomes all zeros, control reverts to MONITOR.

A.4  PROGRAM PARAMETERS

For any given matrix, the origin is taken to be the lower left-hand corner, with the positive x-axis horizontal to the right. Matrix elements are stored 16 to a word (bits 0 and 17 being zero). All elements in a single word have the same y-coordinate, successive words being one unit apart in y (except for the top of a column, where a jump is made to the bottom of the next). The user of this program can specify the following parameters.

$x_{md}$ - Matrix x-dimension $x 2^{-4}$. Any of the values 1, 2, 4, 8, 16, or 32 may be used (giving a matrix up to 512 elements in width).

$y_{md}$ - Matrix y-dimension. Any of the values 16, 32, 64, 128, 256, or 512 may be used.

$x_{sc}$ - x display scale $x 2^{9}$. (Number of scope units corresponding to one matrix element). Must be some (non-negative) power of 2.

$y_{sc}$ - y display scale (scope units). Must be some (non-negative) power of 2.
bom - **Beginning of matrix storage** (address of the lower left corner of the OM).

The number of matrices that can be stored between "bom" and the top of core memory is the integral part of

\[
\frac{20,000_{8}}{xmd \times ymd} - bom
\]

The following octal values are currently being used.

- \(xmd = 10\)
- \(ymd = 100\)
- \(xsc = 4000\)
- \(ysc = 4\)
- \(bom = 10000\)

With this set of values, 8 matrices (including the OM) can be stored, so that a matrix address greater than 7 should not be used.

The contents of the following registers can also be varied.

**dor** - Register containing the **display origin** for the DEPOSIT subroutine (which deposits ones in the OM location corresponding to the light pen position). The number is given in scope coordinates with bits 0 and 9 complemented. The current value (200\(_{8}\)) places the center of the OM at the center of the scope.

**dio** - Register containing the **display origin** for the DISPLAY subroutine (which displays the contents of the OM). The number stored represents the scope coordinates plus

\[
\frac{1}{2} (xsc + ysc) - ysc (llr + bom)
\]

The current value (742600\(_{8}\)) corresponds to that of "dor".
These values, together with the given display scales, provide a centered display with a 4 x 4 square (in scope units) allocated to each matrix element.