HANDPRINTED CHARACTER RECOGNITION.

John W. Sammon, et al

Pattern Analysis and Recognition Corporation

Prepared for:

Rome Air Development Center

January 1973

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)*

Pattern Analysis and Recognition Corporation
128 East Dominick Street
Rome, New York 13440

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3 REPORT TITLE

HANDPRINTED CHARACTER RECOGNITION

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report

5 AUTHOR(S) *(First name, middle initial, last name)*

Dr. John W. Sammon
Dr. Jon H. Sanders
Robert J. McGrath
David B. Connell

6 REPORT DATE
January 1973

7a. TOTAL NO. OF PAGES
114

7b. NO. OF REFS
3

8a CONTRACT OR GRANT NO
F30602-71-C-0331

Job Order No. 55810214

9a. ORIGINATOR'S REPORT NUMBER(S)
PAR Report 72-29

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
RADC-TR-72-329

10 DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11 SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Rome Air Development Center (ISCP)
Griffiss Air Force Base, New York 13441

13 ABSTRACT

This report discusses a research effort whose object is the design of a handprint alpha-numeric reader capable of human recognition rates. The effort included:

1. Collection of a large data base of unconstrained handprinted alpha-numeric characters.

2. Editing the data to re-label mislabeled characters, remove noise, and delete totally illegible characters. A separate programming package (the Data Base Editing Package) was designed and implemented for this purpose.

3. The design of new features to improve the performance of the system.

4. The design of the recognition logic using an OLPARS-like program, the Alpha-Numeric Logic Package (ANLP), which used the expanded feature set.

5. An independent test of the logic using a set of 6127 characters not included in the design set.

6. Analysis of the results of the independent test to develop reject strategies to reduce substitution errors.

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Features						
Recognition						
Alpha-Numeric						
Handprint						

SAC--Griffiss AFB NY

UNCLASSIFIED
Security Classification

Ib

HANDPRINTED CHARACTER RECOGNITION

Dr. John W. Sammon
Dr. Jon H. Sanders
Robert J. McGrath
David B. Conwell

Pattern Analysis and Recognition Corporation

Approved for public release;
distribution unlimited.

IC

FOREWORD

This Final Report is submitted by Pattern Analysis and Recognition Corporation, 128 E. Dominick Street, Rome, NY, under contract F30602-71-C-0331, Job Order Number 55810214, for Rome Air Development Center, Griffiss Air Force Base, NY. Mr. John Palaimo, ISCP, was the RADC Project Engineer.

This report has been reviewed by the Information Office, OI, and is releasable to the National Technical Information Service (NTIS).

This technical report has been reviewed and is approved.

Approved: John Palaimo
JOHN PALAIMO
Project Engineer

Approved: William P Bethke
WILLIAM P. BETHKE
Chief, Information Sciences Division

FOR THE COMMANDER: Fred I Diamond

FRED I. DIAMOND
Chief, Plans Office

ABSTRACT

This report discusses a research effort whose object is the design of a handprint alpha-numeric reader capable of human recognition rates. The effort included:

1. Collection of a large data base of unconstrained handprinted alpha-numeric characters.

2. Editing the data to re-label mislabeled characters, remove noise, and delete totally illegible characters. A separate programming package (the Data Base Editing Package) was designed and implemented for this purpose.

3. The design of new features to improve the performance of the system.

4. The design of the recognition logic using an OLPARS-like program, the Alpha-Numeric Logic Package (ANLP), which used the expanded feature set.

5. An independent test of the logic using a set of 6127 characters not included in the design set.

6 Analysis of the results of the independent test to develop reject strategies to reduce substitution errors.

TECHNICAL EVALUATION

This effort represents a significant advance in the area of optical character recognition. The recognition logic developed was designed on a 33 thousand character data base, one of the largest if not the largest data base ever compiled for this purpose. The accuracy of the logic approaches human recognition capabilities. In fact, when the ability of the human to employ syntactical information to distinguish malformed character is negated, the performance of the recognition logic becomes equivalent with that of the human.

The character rejection rate which was found to be rather high, can be traced to specific problems with source data preparation. The necessity of exercising some controls on data preparation are very evident. The implementation of minimal printing constraints will reduce the substitution rate to 1%. This is quite significant in view of the fact that the error rate of unverified commercial keypunching is approximately 5%.

For all practical purposes the technique has been proven. The follow-on to this work should be the implementation and integration of hardware and software into a final system. It should be noted that there are many variables (i.e. degree of constraints, size of the data set, substitution and reject requirements, etc.) which influence the complexity and performance of a handprinted character reading system. For this reason, there exist the possibility for a family of readers, each tailored to a particular set of these variables. It follows that the best approach to a follow-on is not one directed at a general purpose reader but at readers customized for specific applications.

John Palaimo
JOHN PALAIMO
Project Engineer

TABLE OF CONTENTS

Table of Contents (Continued)

SECTION 1

INTRODUCTION AND SUMMARY

A researcher unfamiliar with the variation in unconstrained handprinting might easily underestimate the amount of sophistication needed in the design of an alpha-numeric reader. A great degree of complexity is necessary to approach human recognition rates in any automated system. The reason is that natural variations in shapes and/or breaks in a character cause a "continuum" of character shapes, starting with a character in one class and ending with a character in another. Figure 1-1 gives a few examples of this.

The variations shown in Figure 1-1 occur frequently enough in unconstrained handprinting to insure that substitution errors will occur. Our goal was to design and implement a system which would minimize substitution errors, i.e., to reduce the substitution rate down to the substitution one might expect from a human attempting to classify the characters.

To achieve this goal the following steps were taken:

1. Collection of a large data base of unconstrained handprinted alpha-numeric characters.

2. Editing the data to re-label mislabeled characters, remove noise and delete totally illegible characters. A separate programming package (The Data Base Editing Package) was designed and implemented for this purpose.

3. The design of new features to improve the performance of the system.

4. The design of the recognition logic using an OLPARS-like program, The Alpha-Numeric Logic Package (ANLP), which used the expanded feature set.

5. An independent test of the logic using a set of 6127 characters <u>not</u> included in the design set.

6. Analysis of the results of the independent test to develop reject strategies to reduce substitution errors.

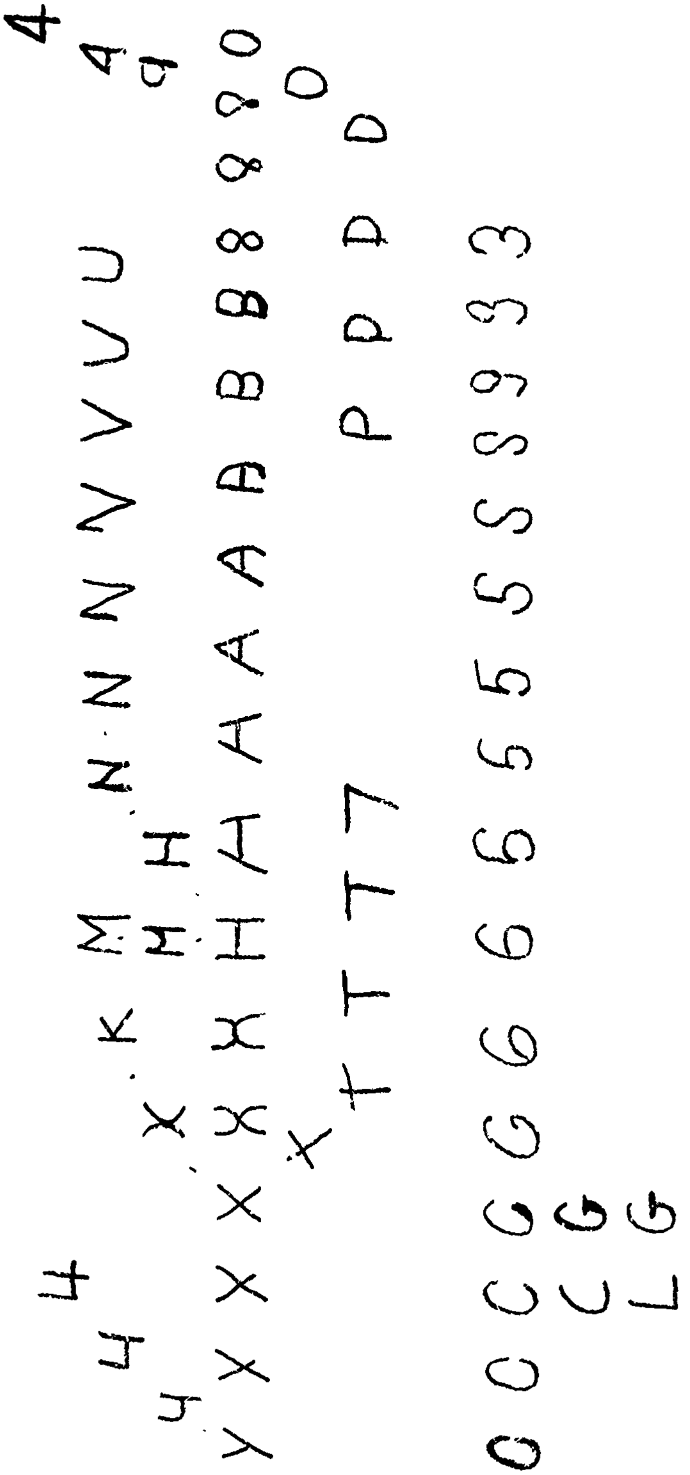


Figure 1-1

In summary, the logic, designed on an edited data base of 33,128 characters, was tested using four different reject strategies, D, C, B, and A. The test set consisted of 6127 unedited characters.

Using reject strategy D (which rejects a character only in the case of ties), a substitution rate of 18.67% and a reject rate of 2.01% was observed on the test set data. Using reject strategy C (which rejects a character if the maximum class receives 32 or less votes or if there is a tie), a substitution rate of 12.87% and a reject rate of 6.92% was observed on the test set. Using reject strategy B (which rejects a character if the maximum class receives 33 or less votes or if there is a tie), a substitution rate of 11.45% and a reject rate of 10.18% was observed on the test data. Using reject strategy A (which rejects a character if the maximum class receives 34 or less votes or if there is a tie), a substitution rate of 9.14% and a reject rate of 16.97% was observed on the test data. As pointed out before, confusion pairs accounted for all but 1% of the substitutions using reject strategy A.

A substitution rate of 1% with a reject rate of 16.97% on unconstrained alpha-numerics compares favorably with human performance and represents a significant advance in the field of OCR. To our knowledge there is no other alpha-numeric reader in existence which achieves these results on unconstrained data. We feel that the system with reject strategy A operates well enough to be of practical value as long as either; (1) certain unresolvable pairs are not permitted in the same fields, or (2) some constraints are placed on members of these pairs.

SECTION 2

THE STANDARD FEATURES

2.1. OVERVIEW

The alpha-numeric feature set consists of the up to 84 standard features developed under a previous contract plus the 8 special features described in Section 3.

The standard feature set consists of five measurements $M(C) = a_1, \ldots, a_5$ made on each of the convexities C of the left, right, top and bottom contours. Since each contour is forced to have 1, 3, or 5 convexities, this results in a maximum of 21* standard measurements for each contour and thus a maximum of 84 standard measurements from each character.

2.2. PRE-PROCESSING

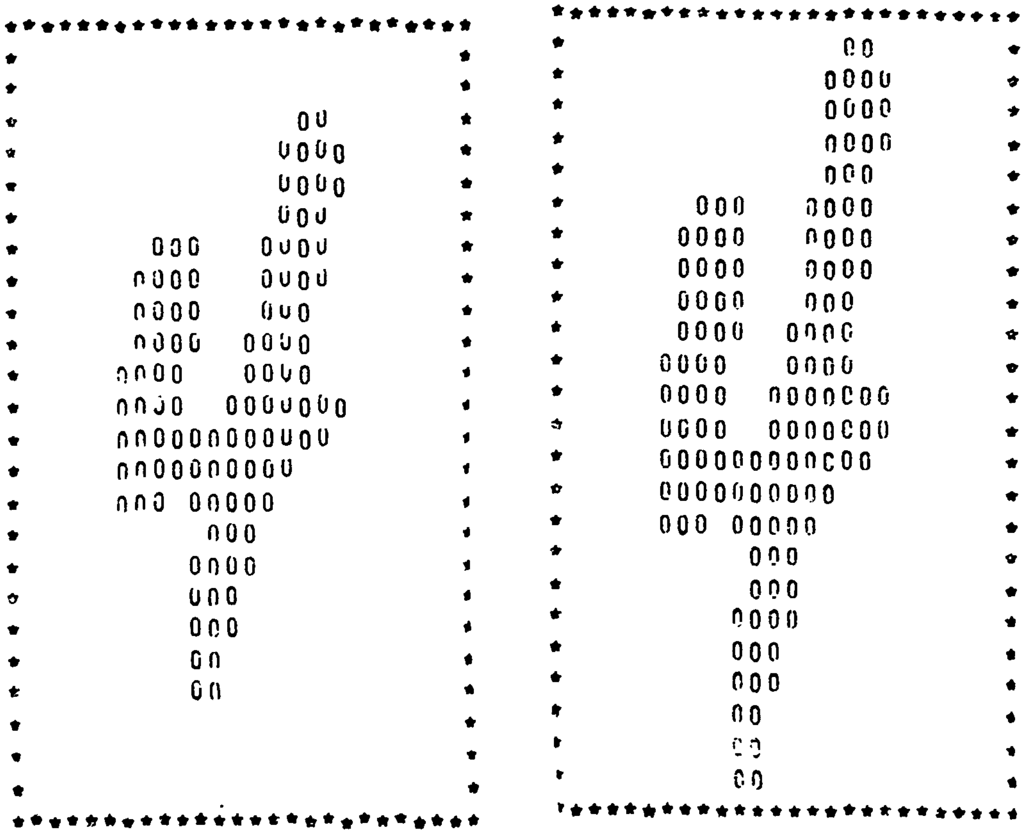
The first operation carried out was that of converting each character from its octal format to a 24 x 24 raster. (We define the character matrix $A = \langle a_{ij} \rangle$, j, i = 1, ..., 24 associated with a character by $a_{ij} = 1$ if a character mark is present in position i, j of the raster and $a_{ij} = 0$ if not; where it is assumed that the rows of the raster are numbered from top to bottom and the columns from left to right (unless otherwise stated)).

Next, a height normalization was performed. Each character was "stretched" in the vertical direction so that it extended from the bottom row (row 24) to the top row (row 1) of the raster (Figure 2-1). The height normalization operates in three steps. First, the character is moved to the bottom of the original raster. Next, the height of the character is computed and designated symbolically as H . Finally, the character is stretched by the expansion factor $M = 24/H$ as shown in Figure 2-2.

After height normalization, the left and right histogram vectors, $\underset{\sim}{L} = L_1, L_2, \ldots, L_{24}$ and $\underset{\sim}{R} = R_1, R_2, \ldots, R_{24}$ were calculated, where

* - Due to certain redundancies, 4 of the 25 measurements taken from five convexities are eliminated.

5



Before Normalization

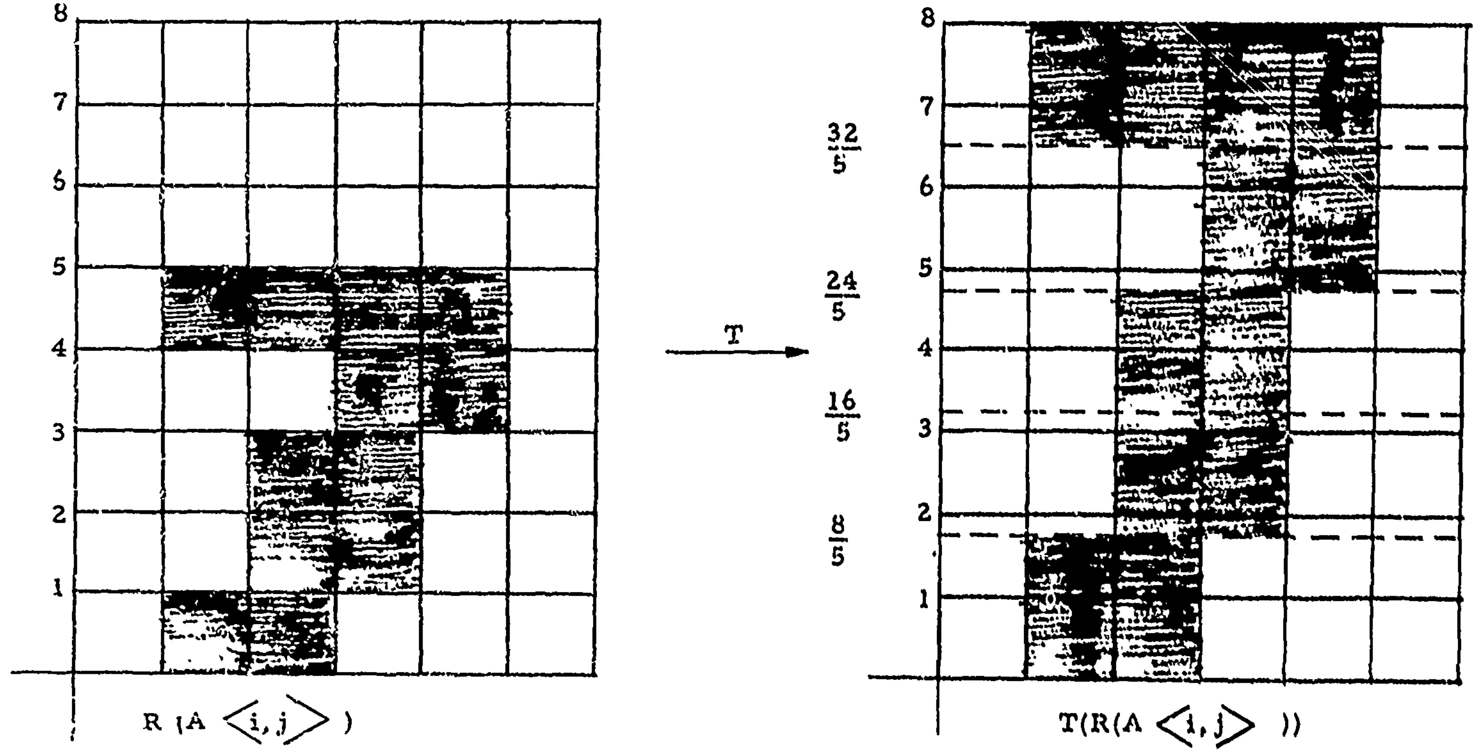
After Normalization

Figure 2-1

$$A = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 0 & 0 \end{matrix}$$

Figure 2-2

H = 5



$L_i(R_i)$ represents the distance from the left (right) margin to the character along the i^{th} row. The top and bottom histogram vectors $\underset{\sim}{T}$ and $\underset{\sim}{B}$ are calculated in a similar fashion after rotating the raster 90° counterclockwise. Figures 2-3, a-c illustrate the four histogram vectors.

The histogram vectors of a character define the contours which are used for character recognition. It seemed that the left and right histograms contained enough information for human recognition of the numeric characters in almost every case. It was felt that the left and right, as well as the upper and lower histograms would suffice for separation of the alphanumerics, except for a few confusion cases.

A straight forward pattern analysis (using the Non-Linear Mapping Algorithm of OLPARS) of the left and right histogram vectors was performed and some partial separation was achieved. For the most part, however, the results indicated that cluster analysis of the histogram vector (with Euclidean metric) was not the correct method of utilizing the "shape ' information inherent in the vector. Analysis of the situation led to the following explanation. The (left) histograms in Figures 2-4. a & b are easily recognized as belonging to the character 2. Intuitively, we feel that they have similar shapes in that they both have two "bumps" separated by a vertical section. Their vectors, however, are no closer to each other than they are to the left histogram in Figure 2-4. c., which belongs to a 1. This is because the "shape" quality of the histogram is of a statistical nature so that comparing the two vectors coordinate-by-coordinate (as we do in measuring Euclidean distance) does not measure closeness of "shape". For example, the vectors of Figure 2-4. a. and 2-4. b. are similar in shape because of the "statistical"* type of statement - "the coordinates decrease gradually, then a sharp increase occurs, then the coordinates are constant, followed by a slight decrease, followed by a sharp increase." This type of word description suggested the approximation of the histogram with line segments having quantized slopes and was the motivation behind the string representation of a wave.

Directed line segments having any of the five slopes were used, the five directions being 270°, denoted by V ; 0° denoted by +H; 180° denoted by -H; 225° denoted by -S; and 315° denoted by +S (Figure 2-5).

The length of the line segments approximating a given histogram was measured by the number of rows (for vertical and slanted segments) or number of columns (for horizontal segments) of the raster that the line

* - By statistical we mean a statement about a subset of the coordinates rather than individual coordinates.

8

```
**************************
*             00         *
*            0000        *
*            0000        *
*           0030         *
*           0000         *
*          00000         *
*         00000          *
*        000000          *
*       0000000          *
*      0000 000          *
*      000  000          *
*     0000 0000          *
*     0000 3000          *
*   00000  0000          *
*   0000   0000          *
*  00000 0000000000000   *
* 0000003300000000000000 *
* 000003000000           *
* 00000   0000           *
*         000            *
*         0000           *
*         0000           *
*         0000           *
*         000            *
**************************
```

Character which generated histograms

Figure 2-3. a.

9

```
HISTOGRAM LEFT VERTICAL   HISTOGRAM RIGHT VERTICAL

***************************************************
*00000000000000          *              0000000000*
*0000000000000           *               000000000*
*0000000000000           *               000000000*
*000000000000            *              0000000000*
*000000000000            *              0000000000*
*00000000000             *              0000000000*
*0000000000              *             00000000000*
*000000000               *             00000000000*
*00000000                *             00000000000*
*0000000                 *             00000000000*
*0000000                 *             00000000000*
*000000                  *             00000000000*
*000000                  *             00000000000*
*0000                    *             00000000000*
*0000                    *             00000000000*
*000                     *                    0000*
*00                      *                     000*
*00                      *            000000000000*
*00                      *            000000000000*
*0000000000              *           0000000000000*
*0000000000              *            000000000000*
*0000000000              *            000000000000*
*0000000000              *            000000000000*
*0000000000              *           0000000000000*
***************************************************
```

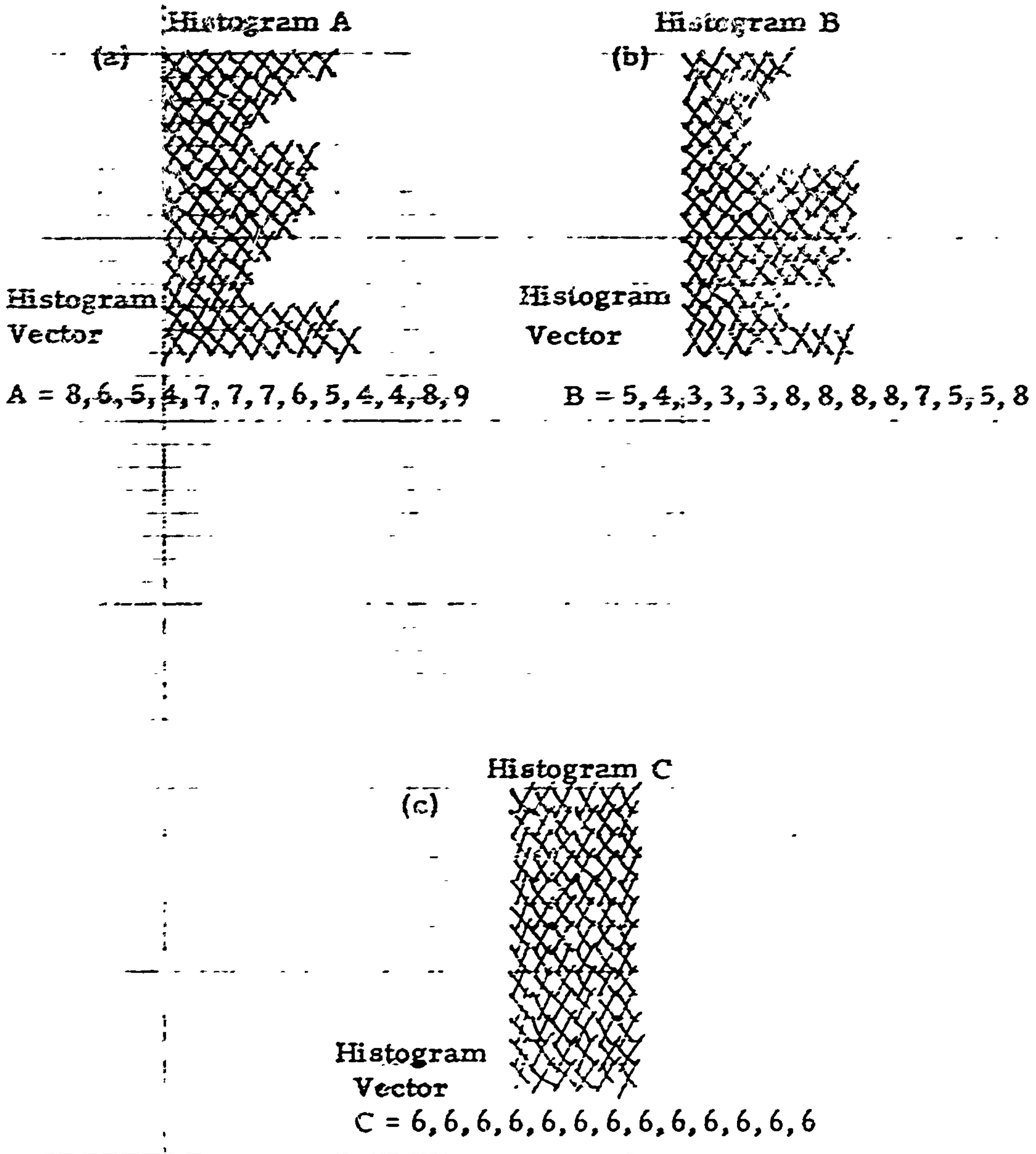
Figure 2-3. b.

20

```
 HIST, UPPER HORIZONTAL   HIST. LOWER HORIZONTAL

***************************************************
0000000000000   000000000*0                     00*
000000000000     00000000*0                     00*
000000000000     00000000*0                     00*
00000000000      00000000*0                     00*
00000000000      00000000*0                     00*
0000000000       00000000*0                     00*
000000000        00000000*0                     00*
00000000         00000000*0                     00*
0000000          00000000*0                     00*
000000           00000000*0                     00*
000000           00000000*0                     00*
00000            00000000*0                     00*
00000            00000000*0                     00*
000              00000000*0                     00*
000              00000000*0                     00*
00                    000*0                     00*
0                      00*0                     00*
0                      00*0            00000000000*
0                      00*0     000    00000000000*
0                      00*000000000    00000000000*
0                      00*000000000    00000000000*
0                      00*000000000    00000000000*
0                      00*000000000    00000000000*
0                      00*000000000   000000000000*
***************************************************
```

Figure 2-3. c.



A = 8, 6, 5, 4, 7, 7, 7, 6, 5, 4, 4, 8, 9

B = 5, 4, 3, 3, 3, 8, 8, 8, 8, 7, 5, 5, 8

C = 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6

$$|A - B| = \sqrt{59} \; ; \quad |A - C| = \sqrt{34} \; ; \quad |B - C| = \sqrt{55}$$

Figure 2-4

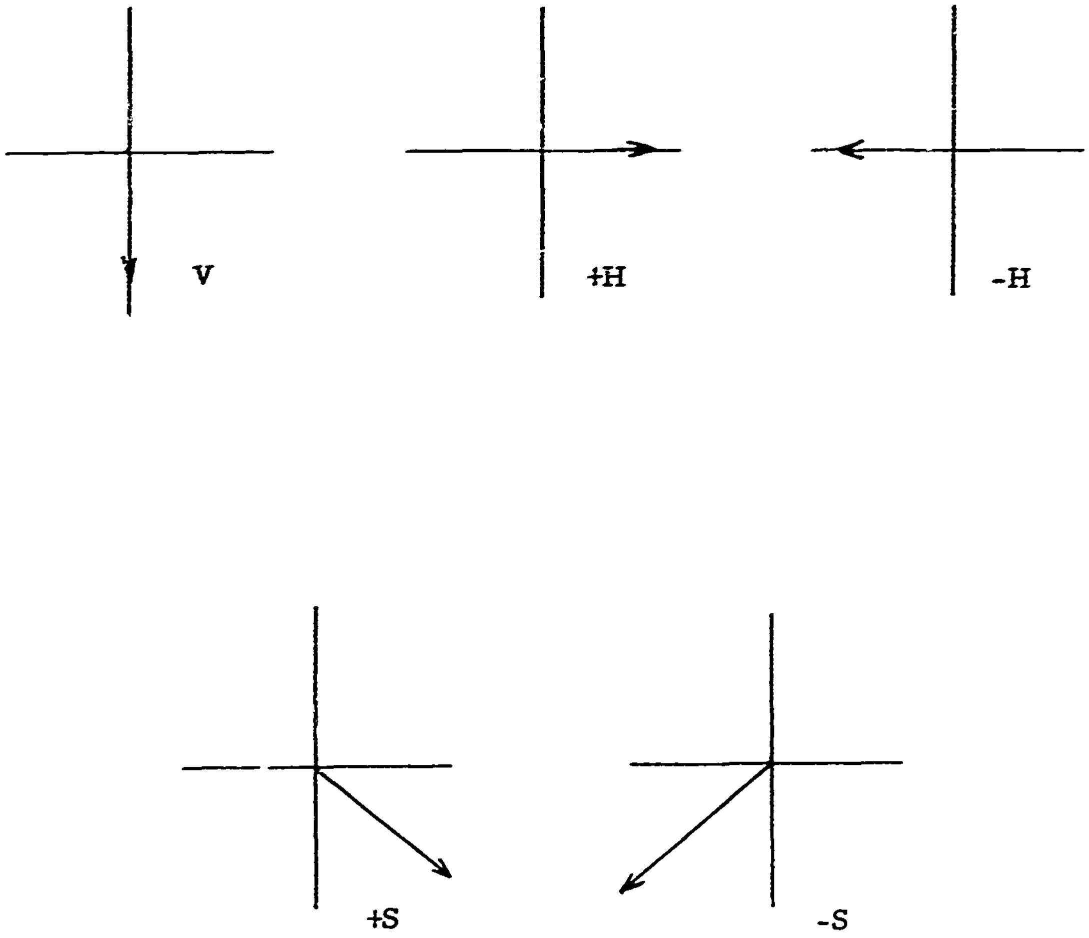


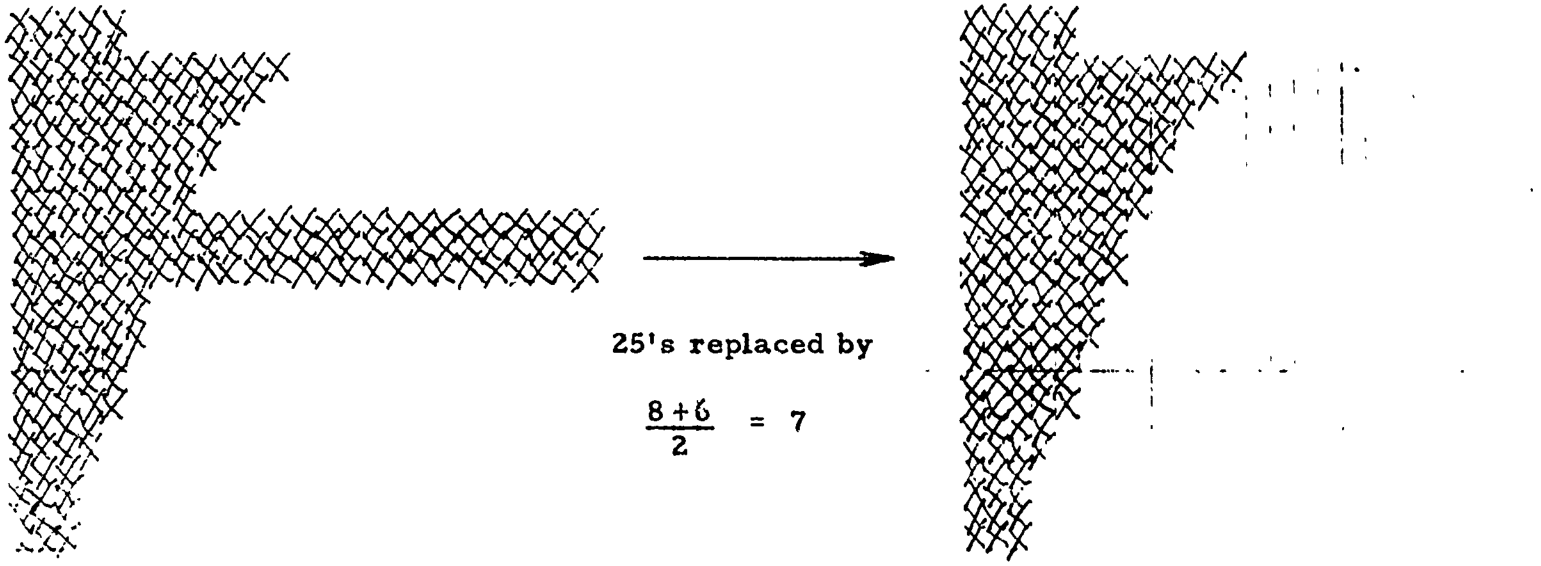
Figure 2 - 5

segment cut across. The ordering of the string associated with a given histogram was determined by the ordering of the corresponding line segments starting at the top of the raster for left and right histograms, starting at the right for top and bottom histograms.

Just prior to generating the string representation some editing is accomplished to fill in certain breaks in the character and also to smooth the histogram representation. No attempt has been made to eliminate salt and pepper noise since it was felt that this type of noise is highly dependent upon the actual scanner used and the quality of the paper being read. Salt and pepper noise elimination algorithms should be included during the prototype development stage. The editing algorithm used here was chiefly concerned with breaks along single stroke segments of the character. For example, if L_{i0} and L_{i1} are the first and last (i.e., $L_{i0-1} \neq 25$; $L_{i1+1} \neq 25$) histogram coordinates of value 25 in a consecutive string of coordinates equal to 25, then the histogram coordinates $L_{i0}, \ldots, L_{i1}$ are changed to the average value $\frac{L_{i0-1} + L_{i1+1}}{2}$. The effect of this editing is shown in Figure 2-6.

Upon completing the editing function, a difference string is generated from the edited histogram representation. The difference string is denoted A_i where $A_i = V_{i+1} - V_i$, $i = 1, 2, \ldots, 23$ and $\underset{\sim}{V} = V_1, V_2, \ldots, V_{24}$ represents an arbitrary histogram vector after editing. The A_i string is next used to "fit" the character contour with the straight line segments shown previously in Figure 2-5. This procedure is conducted in a straight-forward manner by first marking the position along the A_i string where sign changes occur. Between the marks are segments where the histogram elements are either increasing or decreasing monitonically, depending upon the sign of the A_i elements of the corresponding segment. The slope of each segment is computed and used to determine which straight line approximations of Figure 2-5 are appropriate. The criterion for fitting is given below.

Slope	String Symbol
$\lvert SL \rvert \leq \frac{1}{2}$	$V(\ell(s))$
$\frac{1}{2}\ \lvert SL \rvert < 4$	$\pm S(\ell(s))$
$4 \leq \lvert SL \rvert$	$\pm H(\lvert SL \rvert)$



25's replaced by

$$\frac{8+6}{2} = 7$$

Left Histogram of a Broken 7

(5, 5, 12, 11, 10, 9, 9, 8, 8, 25, 25, 25, 6, 6, 6,
5, 5, 5, 4, 4, 3, 3, 3, 3)

New Left Histogram

(5, 5, 12, 11, 10, 9, 9, 8, 8, 7, 7, 7, 6, 6, 6,
5, 5, 5, 4, 4, 3, 3, 3, 3)

Figure 2-6

The following definitions apply to the above table:

SL	=	slope of the segment
$\ell(s)$	=	the length of the segment
$V(\ell(s))$	=	a vertical segment of length $\ell(s)$
$S(\ell(s))$	=	a slant segment of length $\ell(s)$
$H(\lvert SL\rvert)$	=	a horizontal segment of length $\lvert SL\rvert$

The sign is determined in accord with the sign of the corresponding A_i string. If a +H(N) is adjacent to a -H(M), the symbol V(2) is inserted between. Upon completion of these steps a typical string for the character three (3) might be: (left string)

-S(4), +H(6), V(2), -H(5), V(2), +H(8), V(3), -S(2), -H(7), V(4)

This string representation will be used to determine the convexities, however, before proceeding to this step the first and last elements of the string are modified. The width of the character is measured at the top (designated T) and bottom (designated B). Next, a -H(T) is appended at the beginning of the left string (a +H(T) at the beginning of the right string) and a +H(B) is appended at the end of the left string (a -H(B) at the end of the right string. If adjacent horizontals occur, they are combined as follows:

$$H(\ell_1),\ H(\ell_2)\ =\!=\!=\ H(\ell_1+\ell_2).$$

By forcing each string to start and end with -H (+H) and end with a +H (-H) we insure that the string has an odd number of convexities.

The number of convexities for each string of each character is now calculated where a positive convexity is defined to be an increasing substring of maximal length, a negative convexity is defined to be a decreasing substring of maximal length where the symbols are ordered

+H(..) +S(..) V(..) -S(..) -H(..).

Example:

$$\mathrlap{\underbrace{}_{\text{positive convexity}}}+H(3),\ +S(1),\ \mathrlap{\overbrace{}^{\text{negative convexity}}}-H(5),\ V(6),\ \mathrlap{\underbrace{}_{\text{positive convexity}}}+S(4),\ \mathrlap{\overbrace{}^{\text{negative convexity}}}-S(4),\ \underbrace{V(5),\ +H(6)}_{\text{positive convexity}}$$

It is clear that every string (of length 2 or more) breaks down into an alternating sequence of positive and negative convexities.

For any character with more than 5 convexities in any of its strings the following will be iterated until there are only 5 convexities in the string. Find that segment in the string which has a minimal length and remove it. This is subject to the condition that we do not remove an initial H or a terminal H, nor do we remove any symbol which occurs between consecutive H's of opposite sign. After we remove a symbol we re-combine the symbols adjacent to it if they are identical, e.g., if +S(7), V(2), +S(6) is a subsequence of a string, and we remove V(2), we would then combine the two +S's to +S(13). After each removal we recalculate the number of convexities until that number drops to 5.

In order to apply standard pattern recognition techniques to the classification of the strings of the previous section, it is necessary to define a function M(s), which maps each string s into a vector space in such a way that strings with similar shapes are mapped into vectors which are close to each other and strings with dissimilar shapes are not.

The map M(s) will be defined on the positive convexities and then extended to arbitrary strings. For each of notation we replace the symbols +H, +S, V, -S, -H by A_1, A_2, A_3, A_4, A_5, * respectively. Next we add symbols of the form $A_j(0)$ to every positive convexity so that every positive convexity has the form $A_1(k_1)$, $A_2(k_2)$, $A_3(k_3)$, $A_4(k_4)$, $A_5(k_5)$. For example, $A_1(2)$, $A_3(4)$, $A_4(1)$ becomes $A_1(2)$, $A_2(0)$, $A_3(4)$, $A_4(1)$, $A_5(0)$; $A_1(3)$, $A_5(2)$ becomes $A_1(3)$, $A_2(0)$, $A_3(0)$, $A_4(0)$, $A_5(2)$, etc.

We define the <u>vector representation D(s)</u> of a positive convexity $s = A_1(k_1), A_2(k_2), A_3(k_3), A_4(k_4), A_5(k_5)$ to be the vector k_1, k_2, k_3, k_4, k_5. Then M(s) is defined by $M(s) = k_1, k_1 + k_2, k_2 + k_3 + k_4, k_4 + k_5, k_5$ where $D(s) = k_1, k_2, k_3, k_4, k_5$. The first and fifty measurements, k_1, k_5 are simply the lengths of the horizontal segments. The second, third and fourth measurements are the lengths of the top horizontal leg, a, vertical drop, b, and lower horizontal leg, c, respectively (figure 2-7). We define add symbols of the form $A_i(0)$ to it so that it has the form $A_5(k_1)$, $A_4(k_2)$, $A_3(k_3)$, $A_2(k_4)$, $A_1(k_5)$. For example, the negative convexity

* - These two sets of symbols will be used interchangeably.

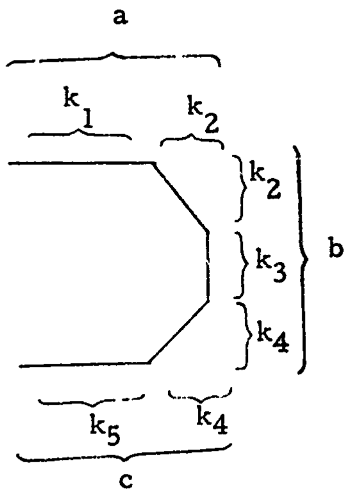


Figure 2-7

$A_4(1)$, $A_2(1)$, $A_1(3)$ becomes $A_5(0)$, $A_4(1)$, $A_3(0)$, $A_2(1)$, $A_1(3)$. We define the inversion mapping $I(s)$ on any negative convexity $s = A_5(k_1), A_4(k_2), A_3(k_3), A_2(k_4), A_1(k_5)$ by $I(s) = A_1(k_1), A_2(k_2), A_3(k_3), A_4(k_4), A_5(k_5)$. Clearly, this maps the negative convexities into the positive convexities. Figure 2-8 illustrates the effect of this mapping.

The mapping M is defined on a negative convexity s by:

$$M(s) = (-a, -b, c, -d, -e) \text{ where}$$

$$M(I(s)) = (a, b, c, d, e).$$

In order to define M on arbitrary strings*, we break the string down into an alternating sequence of positive and negative convexities. The following theorem shows that this is always possible.

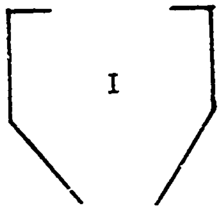
Theorem:

Any string s of length > 1 can be written in a unique way as an alternating sequence $a_1, a_2, \ldots, a_k$ of positive and negative convexities where the last element of a_i is the first element of a_{i+1}, $i = 1, \ldots, k-1$, i.e., if $s = s_1, \ldots, s_n$, $n > 1$, then there exists unique integers $1 < i_1 < i_2 < \ldots < i_k < n$ such that $a_0, a_1, \ldots, a_k$ is an alternating sequence of positive and negative convexities where $a_0 = s_1, \ldots, s_{i_1}$; $a_1 = s_{i_1}, \ldots, s_{i_2}$; ... ; $a_k = s_{i_k}, \ldots, s_n$.

Proof:

The proof is by induction on the length n of the string. For any string $s = s_1, s_2$ of length $n = 2$ we have either $s_1 < s_2$ or $s_1 > s_2$. Thus, any string of length 2 is a convexity. Assume the theorem is true

* - Strictly speaking the strings we deal with are not arbitrary since $A_i(k)$ followed by $A_j(k')$ implies $i \neq j$.



$-H(1),\ V(2),\ +S(1) \longrightarrow +H(1),\ V(2),\ -S(1)$

$A_5(1),\ A_3(2),\ A_2(1) \longrightarrow A_1(1),\ A_3(2),\ A_4(1)$

Figure 2-8

for $n = n_0$. Let $s_1, \ldots, s_{n_0+1}$ be any string of length $n_0 + 1$. Now by the induction hypothesis there are unique numbers $1 < i_1 < i_2 < \cdots < i_k < n_0$ such that $a_0, a_1, \ldots, a_k$ is an alternating sequence of positive and negative convexities where $a_0 = s_1, \ldots, s_{i_1}$; $a_1 = s_{i_1}, \ldots, s_{i_2}$; ... ; $a_k = s_{i_k}, \ldots, s_{n_0}$. Let us assume a_k is positive. If $s_{n_0+1} > s_{n_0}$ then, $a'_k = a_k, s_{n_0+1}$ is still a positive convexity, and $a_1, a_2, \ldots, a_{k-1}, a'_k$ is the required alternating sequence. Furthermore, this is the only alternating sequence possible since the numbers $i_1, \ldots, i_k$ are unique (by the induction hypothesis). If $s_{n_0+1} < s_n$, then $a_{k+1} = s_{n_0}, s_{n_0+1}$ is a negative convexity so $a_1, \ldots, a_k, a_{k+1}$ is the unique required alternating sequence of positive and negative convexities. A similar argument holds if a_k is assumed to be a negative convexity.

Using this theorem we extend the map M to arbitrary strings as follows: If $s = s_1, \ldots, s_n$ is an arbitrary string, then we define

$M(s) = (M(a_1), M(a_2), \ldots, M(a_k))$ where $a_1, \ldots, a_k$ is the alternating sequence of convexities associated with s as described in the above theorem. Since each string has 1, 3, or 5 convexities we thus have 5, 15, or 25 measurements for each string respectively.

SECTION 3

THE SPECIAL FEATURES

3.1. THE MIDUP MEASUREMENT

The first of the eight special measurements is designated MIDUP. As the name implies, this feature measures a characteristic related to the upward view of the character from a row somewhere around the middle of the character. The row used is row 16. The upward view of the character from row 16 is obtained by computing a "midline-up" histogram designated MHIST. The I^{th} element of MHIST, designated MHIST(I) is simply the row number of the first non-zero bit encountered when scanning the I^{th} column upward from (and including) the 16th row. In the case where no non-zero bit is found, the value of MHIST for that column is set equal to zero. The midline-up histogram for the character "two" of Figure 3-1 is listed in Table 3.1.

TABLE 3.1.

I	Midline-Up Histogram MHIST(I)	Topdown Histogram THIST(I)
1	0	24
2	0	24
3	0	24
4	0	24
5	0	24
6	0	24
7	8	4
8	16	4
9	16	3
10	16	1
11	16	1
12	14	1
13	14	3
14	12	3
15	9	6
16	0	21
17	0	21
18	0	21
19	12	12
20	0	24
21	0	24
22	0	24
23	0	24
24	0	24

Columns

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0									1	1	1												0
3	0								1	1	1	1	1	1										0
4	0						1	1	1	1	1	1	1	1										0
5	0						1	1	1	1	1	1	1	1										0
6	0						1	1	1				1	1	1									0
7	0						1	1						1	1									0
8	0						1	1						1	1									0
9	0												1	1	1									0
10	0											1	1	1										0
11	0											1	1	1										0
12	0										1	1	1	1					1					0
13	0									1	1	1	1											0
14	0									1	1	1	1											0
15	0																							0
16	0							1	1	1	1													0
17	0							1	1	1	1													0
18	0							1	1	1	1													0
19	0						1	1	1	1	1				1									0
20	0						1	1	1	1	1				1									0
21	0						1	1	1	1	1	1	1	1	1	1	1	1	1					0
22	0							1	1	1	1	1	1	1	1	1	1	1	1					0
23	0							1	1	1	1	1	1	1	1	1	1	1	1					0
24	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0

Rows

Figure 3-1

The midline-up histogram is used to determine the beginning column and ending column of the upper portion of the character, the two columns being designated BEGIN and END respectively. Next, the maximum histogram value in columns BEGIN through BEGIN+3 inclusive is found and designated MAX1. The maximum histogram value in columns END-6 through END inclusive is found and designated MAX2. Finally, the minimum histogram value in columns BEGIN+3 through END-4 inclusive is found and designated MIN. These three measurements are combined as follows to produce the value of the MIDUP feature.

$$\text{MIDUP} = \begin{cases} \text{MAX1} + \text{MAX2} - 2\cdot\text{MIN} & \text{END-BEGIN} > 7 \\ 0 & \text{Otherwise} \end{cases}$$

where

$$\text{MAX1} = \text{MAX}\{\text{MHIST(I)}\}\ ,\ \text{I} = \text{BEGIN}, \text{BEGIN}+1, \ldots, \text{BEGIN}+3$$

$$\text{MAX2} = \text{MAX}\{\text{MHIST(I)}\}\ ,\ \text{I} = \text{END-6}, \text{END-5}, \ldots, \text{END}$$

$$\text{MIN} = \text{MIN}\{\text{MHIST(I)}\}\ ,\ \text{I} = \text{BEGIN}+3, \ldots, \text{END-4}$$

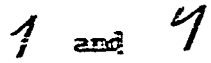
Referring to Table 3.1., it is seen that for the raster of Figure 3-1

BEGIN	=	7th Column (I)
END	=	19th Column (I)
MAX1	=	16
MAX2	=	14
MIN	=	9
MIDUP	=	$16+14-2\cdot 9=12$

3.2. MIDUP2

The second special feature is designated MIDUP2. Its value is determined by counting the number of rows between "middle" row 16 and the row containing the first non-zero bit along the J$^{\text{th}}$ column, where J = L_{16}-1, when scanning upward from (but not including) row 16. Stated differently, the column to be checked for a non-zero bit is determined by scanning the 16th row from the left until the first non-zero bit is found. By backing off one column, the column which will be scanned next is determined. This column is simply L_{16} - 1. Finally, the L_{16} - 1 column is scanned upward from row 16 until a non-zero bit is found. The row number containing this bit is subtracted from 16 to produce MIDUP2. Turning to the example shown in Figure 3-1, it is seen that $L_{16}-1 = 7$ and that the row containing the first non-zero bit is row 8. Thus MIDUP2 = 16 - 8 = 8.

The MIDUP and MIDUP2 features are useful in discriminating certain sevens from either fours, nines or A's. Consider for example, sevens such as:



The first seven will resemble a closed-top four or a skew A and the second will resemble a nine when viewing these characters from the left and right sides. The view from the bottom of the first 7 may be obscured by the slanted stem of the 7. However, the MIDUP and MIDUP2 measurements allow these sevens to be distinguished since the view up from the "middle" line of both fours, nines and A's will be blocked by a relatively low horizontal stroke which is not present in the case of a seven.

3.3. MOTOP

The third of the eight special measurements is designated MOTOP. Effectively, this feature measures the degree of openness at the top of a character and hence the name "open top measurement" symbolically referenced MOTOP. This feature is derived from viewing the character from the top row and is computed from the values of a "topdown" histogram designated THIST. The value of the I^{th} element of THIST is THIST(I) and is simply the row number of the first non-zero bit in the I^{th} column. The topdown histogram for the character "two" of Figure 3-1 is listed in Table 3.1. The THIST histogram is first used to determine the beginning column and the ending column of the character to be used for the MOTOP computation, the columns being designated BEGIN and END respectively. Next, the maximum histogram value in columns BEGIN+2 through END-2 inclusive is found and designated TMAX. The minimum histogram value in columns BEGIN through BEGIN+3 inclusive is determined next and designated TMIN1. Finally, the minimum histogram value in columns END-3 through END inclusive is found and designated TMIN2. These measurements are combined to produce the value of the MOTOP feature as shown below:

$$\text{MOTOP} = \begin{cases} 2\cdot \text{TMAX} - (\text{TMIN1} + \text{TMIN2}) & \text{END-BEGIN} > 8 \\ 0 & \text{Otherwise} \end{cases}$$

$$\text{TMAX} = \text{MAX}\left\{\text{THIST(I)}\right\}, \quad \text{I} = \text{BEGIN}+2, \text{BEGIN}+3, \ldots, \text{END}-2$$
$$\text{TMIN1} = \text{MIN}\left\{\text{THIST(I)}\right\}, \quad \text{I} = \text{BEGIN}, \text{BEGIN}-1, \ldots, \text{BEGIN}+3$$
$$\text{TMIN2} = \text{MIN}\left\{\text{THIST(I)}\right\}, \quad \text{I} = \text{END}-3, \text{END}-2, \ldots, \text{END}$$

Referring to Table 3.1., it is seen that for the raster of Figure 3-1.

BEGIN	=	7th Column (I)
END	=	19th Column (I)
TMAX	=	21
TMIN1	=	1
TMIN2	=	12

and therefore MOTOP = 2·21 - (1+12) = 29.

3.4. AVERAGE WIDTH MEASUREMENTS

Three additional special features are measured which pertain to the average width of the character. The first of these measures is the average width across a segment located near the bottom of the character and is designated BOTAVE. The second measure is the average width across a segment located near the middle of the character and is designated MIDAVE. The last measure is the average width over a large central region of the character and is designated OVRAVE. The width of the I^{th} row is given by RHIST(I) - LHIST(I) + 1, where RHIST and LHIST refer to the break-corrected histograms. Using this notation, the three average width features are given by:

$$\text{BOTAVE} = \frac{1}{6}\left[\sum_{I=16}^{21} (\text{RHIST(I)} - \text{LHIST(I)} + 1)\right]$$

$$\text{MIDAVE} = \frac{1}{6}\left[\sum_{I=10}^{15} (\text{RHIST(I)} - \text{LHIST(I)} + 1)\right]$$

$$\text{OVRAVE} = \frac{1}{16}\left[\sum_{I=5}^{20} (\text{RHIST(I)} - \text{LHIST(I)} + 1)\right]$$

Using the left and right histogram values listed in Table 3-2 corresponding to the "two" of Figure 3-2, the following values are computed:

$$\begin{aligned} \text{BOTAVE} &= \left[43/6\right] = 7 \\ \text{MIDAVE} &= \left[27/6\right] = 4 \\ \text{OVRAVE} &= \left[96/16\right] = 5 \end{aligned}$$

In each case, the lower integer value is used as the feature value.

3.5. TOPLIN and BOTLIN

The remaining two of the eight special features are related to the number of line segments which are crossed when scanning across a specified group of rows. For the purpose of this computation, a line segment is defined by the presence of one or more consecutive one bits which are bordered on the left and right by zeros when scanning a row of the character. For example, the following row contains two line segments:

0000110001110000

The first of these features, designated TOPLIN, is simply a count of the total number of line segments determined by scanning rows 5 through 9 inclusive. The second, designated BOTLIN, is a count of the total number of line segments for rows 16 through 20 inclusive. Following this procedure on the "two" of Figure 3-1, it is determined that:

TOPLIN = 8
BOTLIN = 7

It should be evident that the TOPLIN and BOTLIN features are highly related to the discrimination of eights, H's and X's from other characters. These are sometimes malformed in the sense that the shape information derived from the contours is unreliable. In these instances, the presence of two line segments at the top and the bottom, resulting in large TOPLIN and BOTLIN values, are very useful.

TABLE 3-2

I	Left Histogram LHIST(I)	Right Histogram RHIST(I)
1	10	12
2	10	12
3	9	14
4	[illegible]	14
5	5	14
6	7	15
7	7	15
8	7	15
9	13	15
10	12	14
11	12	14
12	11	19
13	10	13
14	10	13
15	25 (9 after break correction)	25 (12 after break correction)
16	8	11
17	8	11
18	8	11
19	7	15
20	7	15
21	7	19
22	8	19
23	8	19
24	8	19

										1	1	1	1	1	1	1	1	1	1	2	2	2	2	2
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0																							0
3	0																							0
4	0									1	1	1												0
5	0								1	1	1	1	1	1										0
6	0						1	1	1	1	1	1	1	1										0
7	0						1	1	1				1	1	1									0
8	0						1	1						1	1									0
9	0												1	1	1									0
10	0											1	1	1										0
11	0										1	1	1	1					1					0
12	0									1	1	1	1											0
13	0																							0
14	0							1	1	1	1													0
15	0							1	1	1	1													0
16	0						1	1	1	1	1				1									0
17	0						1	1	1	1	1	1	1	1	1	1	1	1	1					0
18	0							1	1	1	1	1	1	1	1	1	1	1	1					0
19	0							1	1	1	1	1	1	1	1	1	1	1	1					0
20	0																							0
21	0																							0
22	0																							0
23	0																							0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3 - 2

SECTION 4

RECOGNITION LOGIC

4.1. THE DECISION LOGIC

As explained in Section 2, measurements are taken from each of the four contours, left, right, top and bottom, of each character. These standard measurements in addition to the special measurements comprise the feature vector upon which the logic operates. * For each of the

$\frac{36 \times 35}{2}$ = 630 pairs of classes a partial decision is reached by operating

on a subset of the feature vector composed of the standard measurements taken from two of the four contours plus the 8 special measurements. The two contours used depend on the particular character class pair in question. For example, for the pair H/N, the top and bottom contour are used since these contours correspond to the two "looks" which give the best discrimination. That is, looking from the left and right these characters appear identical, but looking from the top and bottom they do not. The two contours used for each character pair are listed in Appendix B of (3).

The outcome of each pairwise decision can be a vote for one of the two classes or a no vote decision. Thus, if the character class pair were A/B the outcome would be a vote for A, a vote for B, or a vote for neither. The final decision is reached by adding all the votes for each character class and choosing the class which has the maximum number of votes. If two or more classes receive a maximum number of votes the character is rejected. **

Before we discuss the details of how each pairwise decision is made, a discussion of linear discriminant logic is in order.

Let us denote the feature vector $\underset{\sim}{X}$ of character by:

* - A detailed description of the program which implements the recognition logic is contained in Section 7.

** - Reject strategies based on the number of votes the maximum class receives are discussed in the following section.

$$\underset{\sim}{X} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_L \end{bmatrix} \updownarrow L \qquad (4.1.)$$

A linear discriminant is computed by taking the inner product of the discriminant vector $\underset{\sim}{d}$ with the character featrre vector $\underset{\sim}{X}$ where:

$$\underset{\sim}{d} = \begin{bmatrix} d_1 \\ d_2 \\ \cdot \\ \cdot \\ \cdot \\ d_L \end{bmatrix} \updownarrow L \qquad (4.2.)$$

The inner product generates a scalar Z (i. e., a number) which is used to make a decision between two classes. If the value of Z exceeds a threshold θ, the decision is made for one of the classes; otherwise the other class is decided. Specifically, the inner product is given by:

$$Z = \langle \underset{\sim}{d}, \underset{\sim}{x} \rangle = \underset{\sim}{d}^T \underset{\sim}{x} = \sum_{i=1}^{L} d_i x_i \qquad (4.3.)$$

As seen in equation (4. 3.) a linear discriminant is nothing more than a weighted linear combination of the features x_i. Our problem simply amounts to computing good weights (or equivalently a vector d) for discriminating the pair of classes in question. The optimal procedure used for both the numeric and alpha-numeric logic is based upon the Fisher Linear Discriminant.(1).

Programs PHASEONE and PHASONEA of the ANLP produce weights such that the difference between the mean value of Z (equation (4. 3.) for the two classes is maximized relative to the sum of scatter of Z for the two classes.* The mathematical derivation of the optimal linear discriminant is given in (2).

* - The scatter is a statistic closely related to the variance. The scatter is equal to (N-1) times the variance where N is the number of samples used to estimate the variance.

The following geometric interpretation of the decision procedure may be helpful. Consider the case where L = 2 ; that is, only two features x_1 and x_2 are used to discriminate the classes I and J . The sample data from the two classes are represented as feature vectors in the x_1, x_2 space as shown in Figure 4-1. The decision rule specified by a linear discriminant requires that the inner product between an unknown vector $\underset{\sim}{X}$ and the discriminant vector $\underset{\sim}{d}$ be computed. It is easily shown that an equivalent form of equation 4.3. is:

$$Z = \left|\underset{\sim}{d}\right| \left|\underset{\sim}{X}\right| \cos \alpha \qquad (4.4.)$$

where $\left|\underset{\sim}{d}\right|$ is the vector length of the discriminant vector,

$$\left|\underset{\sim}{d}\right| = \left(\sum_{i=1}^{2} d_i^{\,2} \right)^{\frac{1}{2}} \qquad (4.5.)$$

The discriminant vector is normalized such that $\left|\underset{\sim}{d}\right|$ = 1 and so $Z = \left|\underset{\sim}{X}\right| \cos \alpha$, where α is the angle between the vectors $\underset{\sim}{X}$ and $\underset{\sim}{d}$. Therefore, Z simply is the orthogonal projection of X onto the direction of d as shown in Figure 4-1. Since the decision rule is:

$$\begin{array}{lll} \text{Decide I} & \text{if} & Z \geq 0 \\ \text{Decide J} & \text{if} & Z < 0 \end{array} \qquad (4.6.)$$

the decision boundary is given by the locus of all points such that Z = 0. For this case, the decision boundary is simply a straight line perpendicular to the direction of $\underset{\sim}{d}$ at a distance of 0 from the origin along $\underset{\sim}{d}$. In general the decision boundary implemented by a linear discriminant is a linear hyperplane which divides the feature space into two regions, one associated with class I and the other with class J.

We now return to the detail of how each pairwise decision is made. The type of logic was determi?ed by the number of available samples from the two classes. The following rules were used:

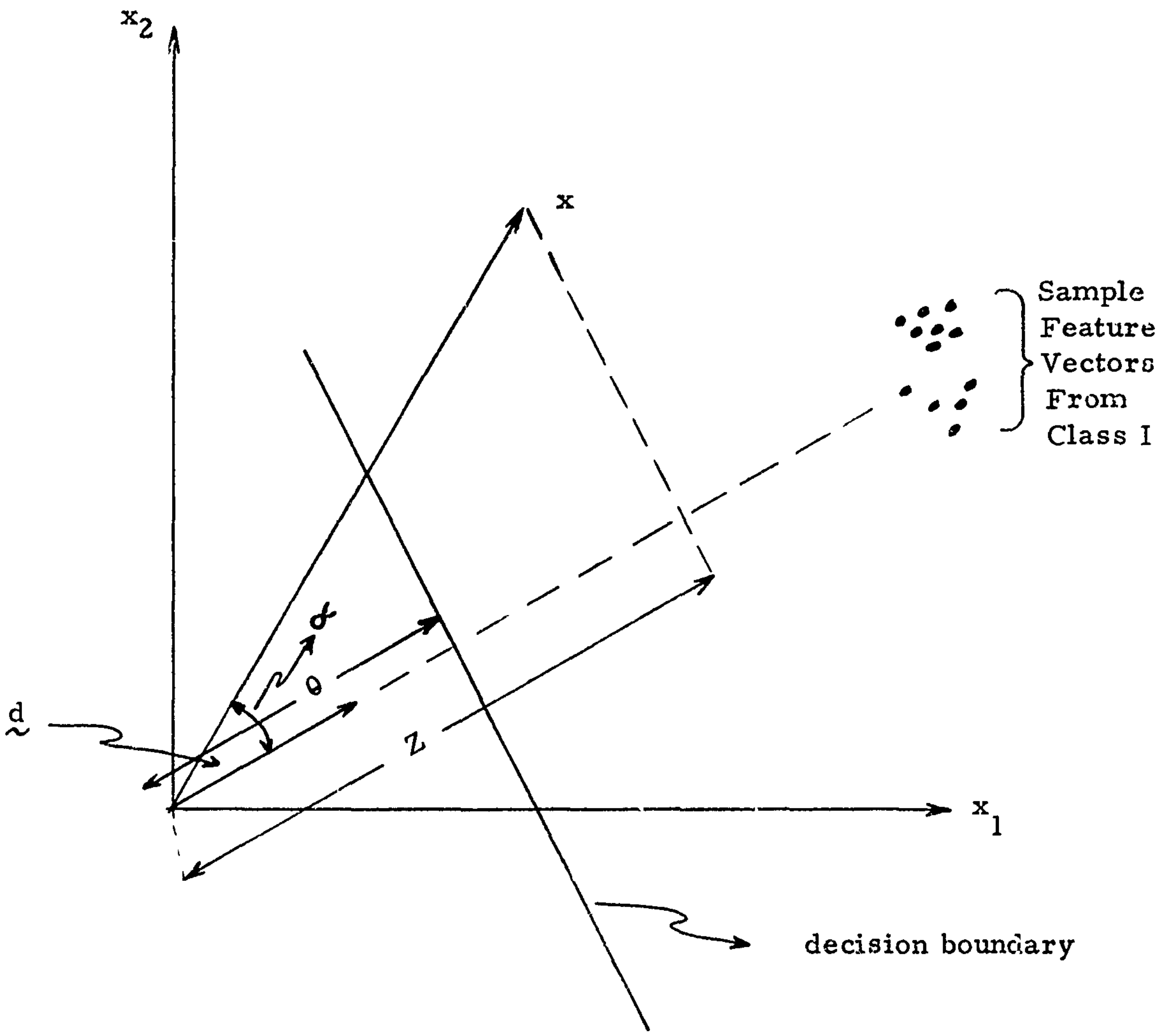


Figure 4-1

Linear Discriminant L = 2 Geometric Interpretation

$N_I \geq 2$	$N_J < 2$	Decide I
$N_J \geq 2$	$N_I < 2$	Decide J
$N_I < 2$	$N_J < 2$	No Vote
$N_I \geq 2$	$N_J \geq 2$	Optimal Linear Discriminant

where N_I equals the number of samples from class I. Types 1-3 were used in approximately 65% of the pairwise cases. The reason for this is due to the fact that every character has a preferred set of sort groups where it will normally be found, where the sort group is determined by the number of convexities a character has in each of two specified contours. For example, the numeral seven will be in the 3, 1 sort group when viewed from the left and right, respectively; i.e., it will have 3 convexities on the left and one on the right most of the time.

In contrast, an E never has just one convexity on the right and thus is never in the 3, 1 sort class when viewed from the left and right. Since the left and right looks are the ones used in the E/7 test, and since characters are grouped by sort class prior to entering the decision logic, the decision logic for E/7 in the 3, 1 sort class simply is "decide 7". Similarly, in the 1, 5 sort class (when viewing from left and right), since no sevens can occur there and E commonly occurs there, the logic is simply "decide E".

4.2. LOGIC OPTIMIZATION

The following is a description of the optimal strategy for sequencing through the logical computations pertinent to reject strategy A (see next section). Let K denote the number of classes (K = 36 for the alphanumeric problem) and assume that the rejection strategy requires that some class receive all of the possible votes (i.e., (K-1)), otherwise the character is rejected. Furthermore, let us assume that all of the logic for the K(K-1)/2 pairwise tests resides in core. Now, since the computer is a sequential device we must direct it to compute the pairwise tests in some prescribed order. We could compute all tests in an arbitrary order and simply tally up the votes for each class at the completion. Any class receiving (K-1) votes would be the decision class, otherwise a rejection is signaled. This strategy can easily be improved upon since it is possible to make a final classification or rejection without computing all K(K-1)/2 pairwise decisions. The optimal strategy which produces the fewest number of tests and therefore insures the fastest throughput would operate as follows; first, all of the pairwise tests, say I vs J (T_{IJ}) would be ordered in accord with the class probabilities, (i.e., rank order T_{IJ} such that $T_{IJ} > T_{RQ}$)

iff $\begin{cases} P_I > P_R & I \neq R \\ P_J > P_Q & I = R \end{cases}$. Suppose the ranking were as follows for a four class problem: $P_1 > P_2 > P_3 > P_4$; the pairwise tests would then be ordered 1/2, 1/3, 1/4, 2/3, 2/4, 3/4. The first test would be 1 vs 2. The next test will be determined on the basis of the outcome of the preceding test. For example, suppose the vote goes to class 1. In this case, class 2 cannot receive the maximum number of votes (i.e., (K-1) = 3) and thus the only classes in contention are 1, 3, and 4. Therefore, we would select the next highest probability pair not involving class 2. In this example, 1 vs 3 would be chosen. Now suppose that class 3 receives the vote, in which case neither 1 nor 2 can win. Repeating the above procedure, the next highest probability pair not involving classes 1 or 2 is the 3 vs 4 test. Suppose the outcome of this test is a vote for class 3. Now we know that classes 1, 2, and 4 cannot be winners, however, we do not know if class 3 is the winner without checking to see if class 3 receives the maximum number of votes. Thus, we would next perform the 2 vs 3 test. If 3 wins, then the final decision is 3, otherwise a rejection is issued. Notice in this case 4 tests were required. In general, the table below lists the number of tests required assuming $P_1 \geq P_2 \geq P_3 \geq \dots \geq P_K$.

True Class	# of Tests Required
1	K - 1
2	K - 1
3	K
4	K + 1
5	K + 2
.	.
.	.
.	.
K	2K - 3

Table 4 - 1

The expected number of tests is given by

$$\overline{T} = P_1 \left[K - 1 \right] + \sum_{I=2}^{K} P_I \left[K + 1 - 3 \right]$$

Notice, if all the classes were equally likely, then $P_I = 1/K$ and

$$\overline{T} = \frac{1}{K} \left\{ (K - 1) + (K - 1)(K - 3) + \sum_{I=2}^{K} I \right\}_{P_I = 1/K}$$

$$= \frac{1}{K} \left\{ (K - 2) + (K - 1)(K - 3) + \frac{K(K+1)}{2} \right\}_{P_I = 1/K}$$

It is interesting to compare $\overline{T} \Big|_{P_I = 1/K}$ to the total number of pairwise tests in order to appreciate the significance of the potential savings.

K	$\overline{T} \Big|_{P_I = 1/K}$	Total Tests = K(K-1)/2
2	1	1
3	2.33	3
10	12.6	45
36	50.5	630

Notice that for 36 classes the expected reduction in computation time might be in the order of 80%

The above strategy could be extended to the case where the entire pairwise logic cannot fit into core at the same time. In this case, we must consider the access time required to retrieve the pairwise logic from the mass storage device. The optimal strategy must not only consider the number of tests but also the time to retrieve the test from storage. The strategy actually used during the independent test of our alpha-numeric logic was of this nature and is described in Vote Assignment, Vote Tally, and Logic Select Procedures in Section 7 of this report.

SECTION 5

REJECT STRATEGY AND ERROR ANALYSIS

As described previously, the decision logic operates by a vote counting procedure. In order to investigate the utility of reject strategies based on the number of votes the winning class received, the results of the independent test were analyzed. The output of the TWO program of the ANLP which performed the independent test listed the maximum number of votes each character received and indicated which class(es) received that number of votes.

The reject strategy which resulted in the lowest number of rejects is to simply reject a character in the case of ties. Thus, if the class which received the maximum number was unique and matched the true identity class of the character, a correct decision resulted; if the class which received the maximum number of votes was unique and was not the true class identity, a substitution occurred; if more than one class received a maximal number of votes the character was rejected. This strategy, Strategy D, may be considered minimal since there was no way (based on vote counting) of making a logical decision between two or more classes all of which received a maximal number of votes, and thus the character must be designated a reject.

Since there were 36 classes in the alpha-numeric set, the most votes a character could receive for any one class was 35. For example, the class D could gain one vote if each of the 35 pairwise tests A/D, B/D, C/D, E/D, F/D, ..., Z/D, 0/D, ..., 9/D was decided in favor of D. By insisting that the class have at least 35 votes and rejecting in case of ties, we obtain reject strategy A. By replacing 35 by 34 and 33 respectively in the above sentence, we have the definitions of reject strategies B and C respectively.

The error and reject rates on the independent test using strategies A, B, C, and D are indicated in Figure 5-1. As we would expect, the errors decrease in the order D, C, B, A, and the rejects increase in the order D, C, B, A, since a character which is rejected using strategy D will surely be rejected using strategy C, a character rejected using strategy C will be rejected using strategy B, etc. That is, if the maximum character class is tied it certainly satisfies the criteria of receiving less than 33 votes or being tied; if the maximum character class receives less than 33 votes or is tied it certainly receives less than 34 votes or is tied, etc.

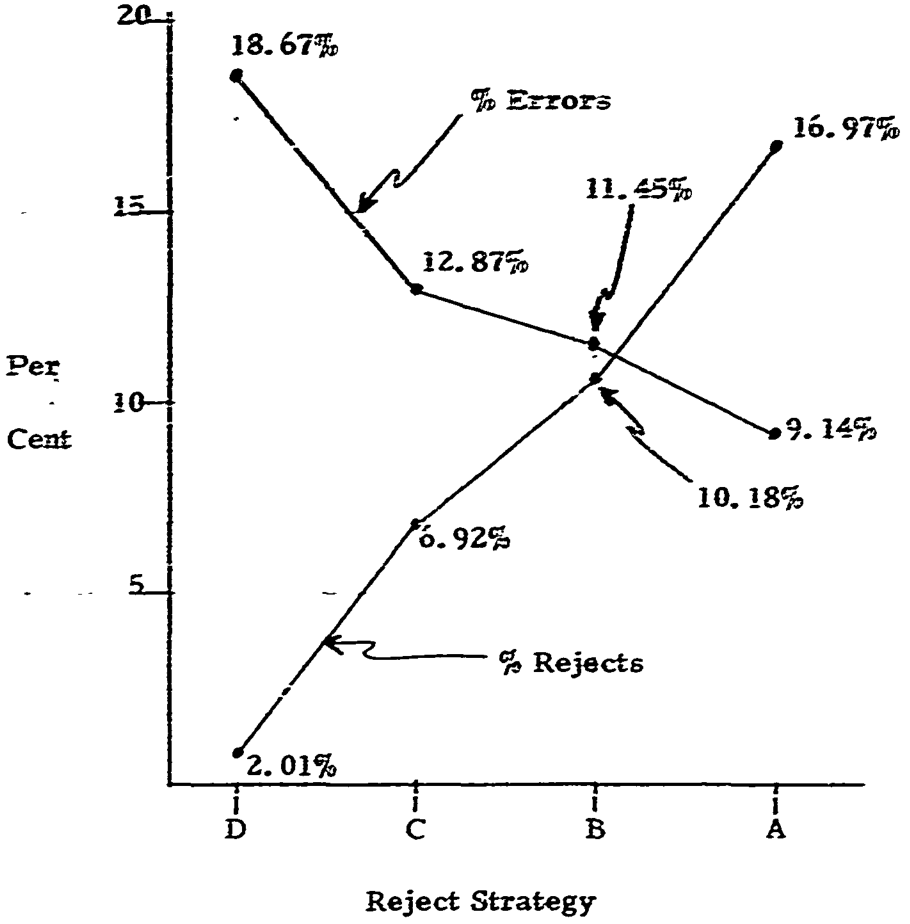


Figure 5-1

Error and Reject Rates Using the Various Reject Strategies

The minimum substitution rate of 9.14% using strategy A at first seemed higher than expected. Analysis of the substitution table (Figure 5-2) however, revealed that certain character pairs have inordinately high substitutions between them. These "confusion pairs" account for a large portion of the substitution. Figure 5-3 lists the most commonly confused character pairs and the number of characters in confusion using reject strategies A, B, and C, where the number of characters in confusion for pair x, y is defined to be the number of characters of true class x whose decision class was y plus the number of characters of true class y whose decision class was x.

As expected O's and 0's, S's and 5's, 2's and Z's, G's and 6's, I's and 1's, V's and U's, and B's and 8's head the list. Discounting just these confusion pairs reduces the substitution rate (using reject strategy A) from 9.14% to 3%. Discounting the rest of the confusion pairs in Figure 5-3 reduces the substitution rate to 1%. A review of Figure 1-1, Section 1, is all that is needed to appreciate why these pairs are in confusion. The characters contained in Figure 5-4 through Figure 5-6, taken from the test set data, further illustrate why certain character pairs are in confusion.

Decision Class

True Class Identity	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0						1						2		2			3								53		8									
1			1					1												1																
2											1		1																							1
3						1	1		1			1																	1							
4										3							1														2		1		6	
5									1											2									32							
6		1	2			5						1	2		1		18					2														
7			2	1						4																										
8							1	1				13			1	1												1								
9			1		3			4																												
A																		1				1						2						1		
B	1			1					4							1																				
C							9								3		3					1														
D	5										1						1								1		1				1					
E																																				
F									1																											
G							29						2																							
H					1																1												5			
J		31																		1										1						
J						1																							1	1						
K																																		8		
L		1	2				4								3																					
M					2													2																		
N																		6															3			

Figure 5-2 (Continued on next page) Table of Substitutions Using Reject Strategy A

	Decision Class																																		
True Class Identity	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
O	105								1					6			3											7								
P																2																				
Q	7									2																										
R			3								1										1															
S				1		24														2																
T								5		1																										
U																		1						1								5	1			
V							1																								19				1	
W																																				
X																					1														2	
Y					4																											10				
Z			47																																	

Figure 5-2 - Table of Substitutions Using Reject Strategy A

(Continued from Previous Page)

Confusion with Reject Strategy

Pair	A	B	C
O/0	158	161	162
S/5	56	59	59
Z/2	48	61	63
G/6	47	48	48
I/1	31	32	33
V/U	24	28	28
B/8	17	20	22
Q/0	15	15	15
V/Y	11	11	12
C/6	11	12	14
Y/4	10	10	10
K/X	9	16	17
7/9	8	8	10
O/D	7	9	11
Q/O	7	8	8
D/0	7	7	7
N/H	6	6	7
4/9	6	9	10
6/L	6	10	10
T/7	5	5	6
H/W	5	9	10
5/6	5	9	12

Figure 5-3

42

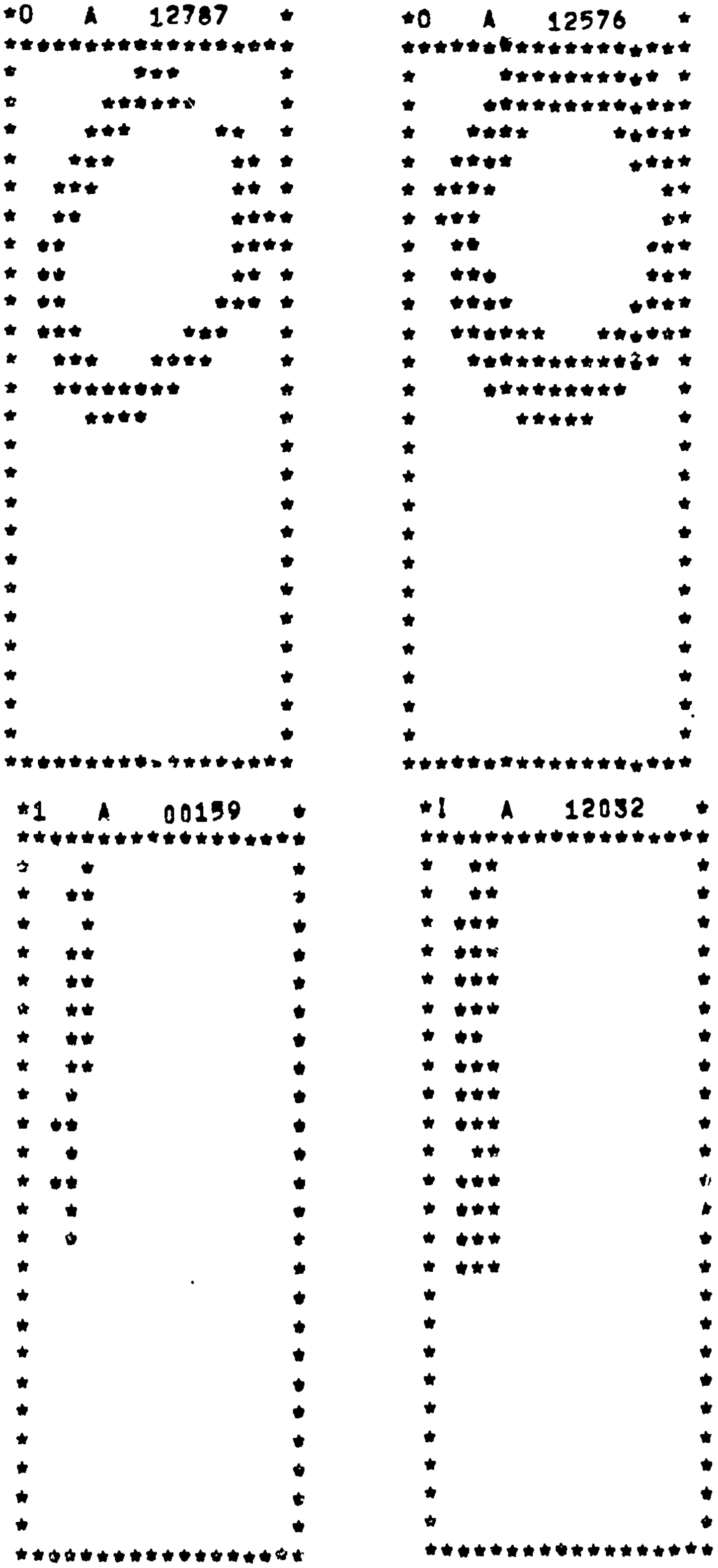


Figure 5 - 4

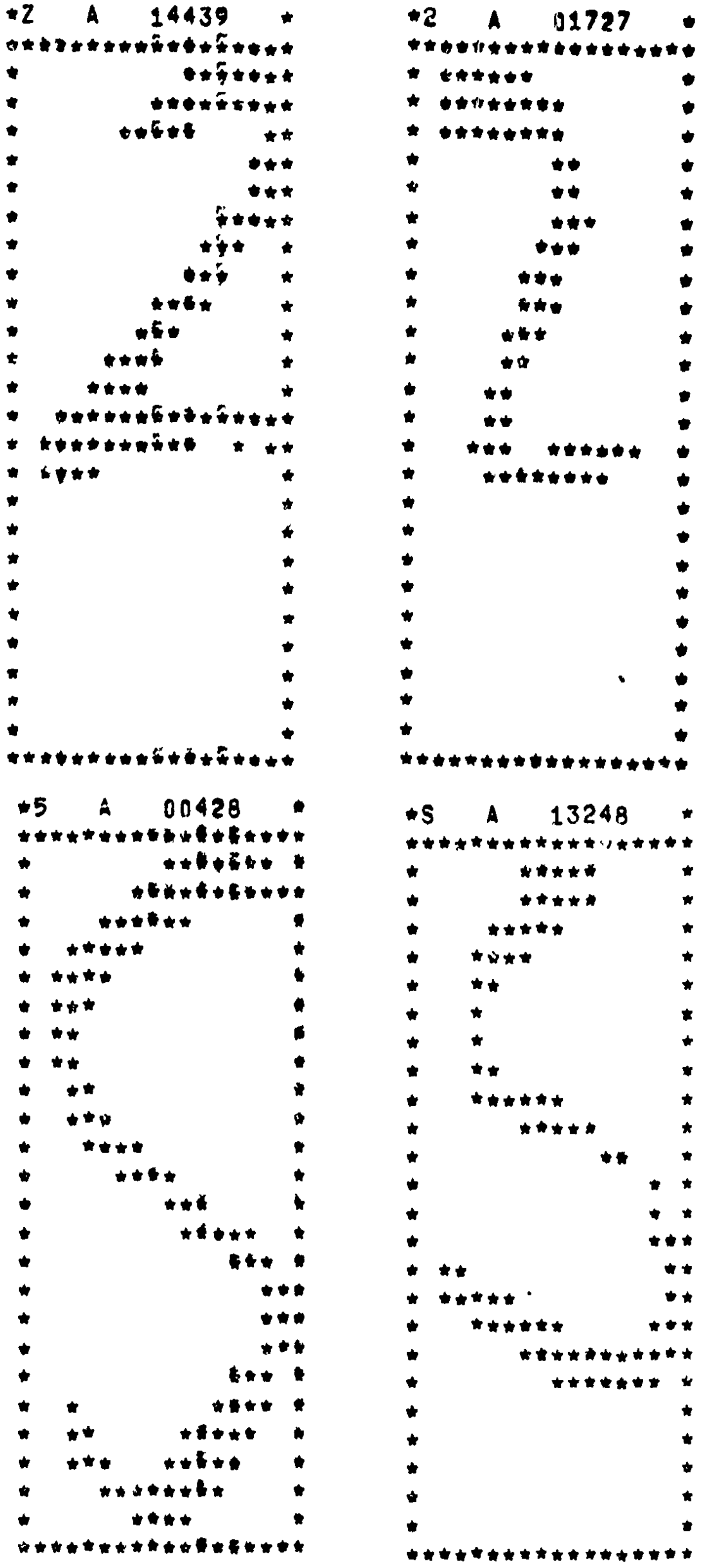


Figure 5 - 5

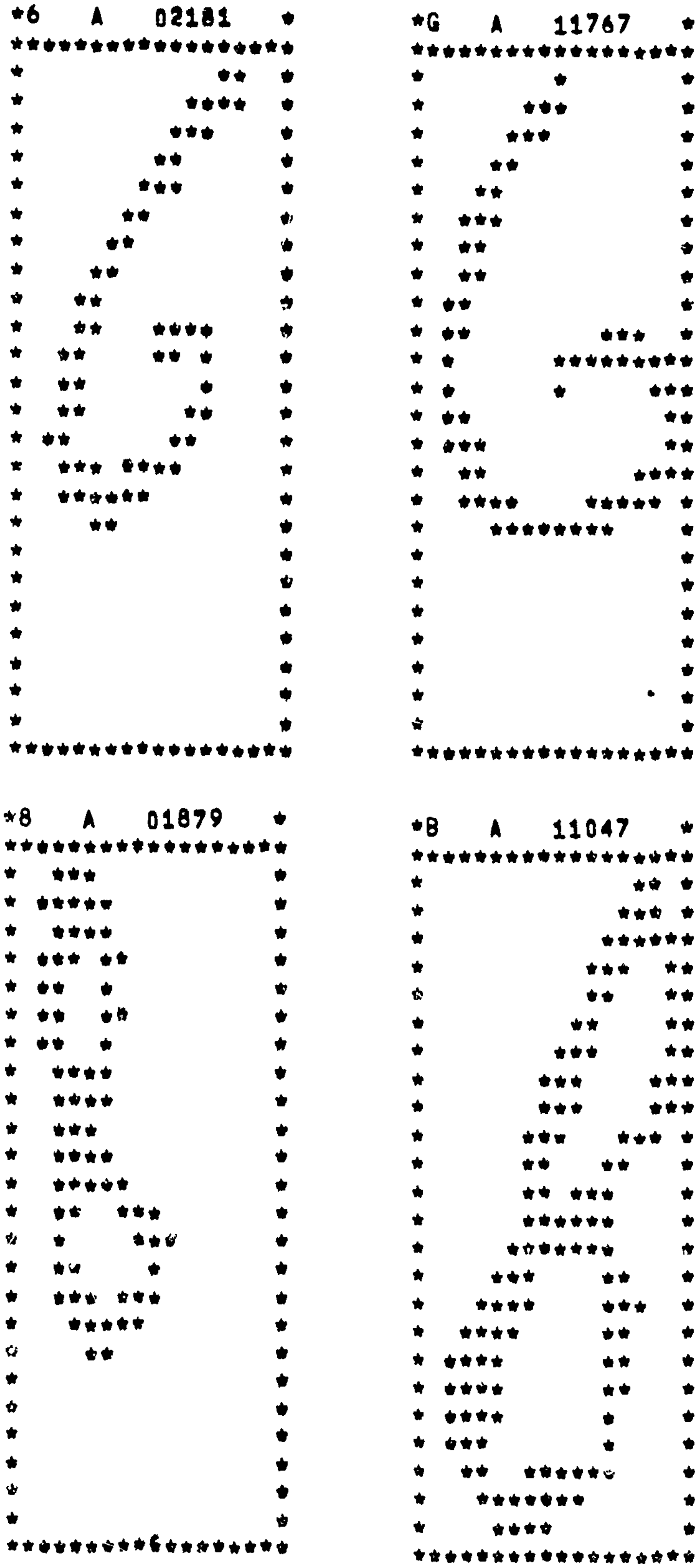


Figure
5 - 6

SECTION 6

DATA BASE EDITING PACKAGE

In an attempt to improve the design of the decision logic, PAR elected to develop a data base editing package. This package was used to prepare a data base suitable for logic design. In addition to correctly labeling mis-labeled characters, certain noise elimination operations were implemented. Specifically, rows or columns of noise caused by an intermittant "wrap around" effect of the camera were eliminated, as well as stray noise bits. In all cases the noise was separate from the character so that these eliminations did not alter the shape of the character. In addition, characters which were so malformed as to be humanly illegible were deleted. This was essential to good logic design since illegible characters as represented in the feature space have the same deleterious effect as noise on the design of optimal discriminant boundaries.

It should be noted that no editing was performed on the test data since any such editing would be counter to the objective of testing the system on "live" unconstrained characters.

The data base editing package was implemented on the CDC-1604 using the BR-85 as the inspection medium. To reduce the programming effort to a minimum, the CDC-1604 operating system and various existing routines developed by PAR for the manipulation of the BR-85 display were employed.

The input to the editing programs is a tape containing a digitized version of each character scanned by the Image Dissector and written by the PDP-8 at a density of 800 BPI. Since the CDC-1604 requires an input density of not more than 556 BPI, PAR has written a simple program which is operable on the Honeywell 645 to generate a tape with identical format and with an acceptable density. It is this converted tape that is used as input to the edit package.

The remainder of this section will discuss the edit function offered to the operator (Section 6. 1.), the computer set up and run deck construction to activate the package (Section 6. 2.) and program flow charts.

6. 1. EDIT FUNCTION

Each function made available to the operator may be executed by selecting and depressing the appropriate function key; 1 - 30 are used. The following describes each key and the associated function:

Key 1: Read a character from the input tape.

This operation will read the next character from the input tape, matrix size is 36 x 30, and display it on the BR-85. In addition, the character and author number are displayed over the character matrix.

Key 2: Not Used

Key 3: Backspace the input tape one character.

On occasion the operator finds the need or desire to examine the character just processed; this function allows the repositioning of the input tape to accommodate this condition.

Key 4: Not Used

Key 5: Write End-Of-File on output tapes on logical units 4, 5, and 6 (an example of the tape setup is shown on page 6-5).

At the completion of each editing session the operator will write EOF on each of the output tapes by selecting Key 5; and in addition, the selection of this key will cause the system to punch a card with the character ID and author number of the last character processed. This card provides a means of automatically positioning the input tape to last character processed in subsequent editing sessions.

Key 6: Not Used

Key 7: Delete a row of noise.

This function is used to delete all data points in the row selected by the light gun. Action: (a) Depress Key 7; (b) Select any data point in the row to be deleted using the light gun.

Key 8: Delete a column of noise.

To delete all data points in a given column depress Key 8 and select any data point in the desired column with the light gun.

Key 9: Delete one data point of noise.

Any given point of noise may be deleted by depressing Key 9 and selecting the point to be deleted with the light gun.

Key 10: Delete column one of the character matrix.

To delete column one, depress Key 10. Due to a programming error in the PDP-8 character matrix generation program, column one and two frequently contain erroneous data points. This special function was included to speed the editing phase (Key 15 may be used to delete column two).

Key 11: Change character ID.

The PDP-8 character matrix generation program assigns a character ID in accord with the position of the character on the input form. If the physical character displayed on the BR-85 disagrees with the assigned character ID the operator may change the ID by depressing Key 11 and depressing the correct ID on the BR-85 keyboard.

Key 12: Not Used

Key 13: Restore an erroneous delete.

After the completion of a delete operation, the user may restore the row, column, or point deleted by depressing Key 13. This operation will restore only the last deletion made.

Key 14: Set output author number (ID)

The number system employed by the PDP-8 program and that used by the FEATURE EXTRACTION and OLPARS programs are inconsistent. This function compensates for that inconsistency.

Key 15: Delete column two of the character matrix.

See description of Key 10.

Key 16: Not Used.

Key 17: Sort Edited Alpha/Numeric tape (physical unit 2).

This function is used to extract characters from the tape on logical unit 10 and write them on logical unit 11 if numeric, on logical unit 12 if the character ID is an alpha, and on logical unit 13 if it is a special symbol.

Key 18 & 19: Not Used

Key 20: Write an edited character on physical unit 16.

The characters written by this function are of a very distorted contour but are still recognizable. They are not included in the initial design of the decision logic but instead are included in a subsequent redesign of the decision logic. The function is activated by depressing Key 20.

Keys 21, 22, & 23: Backspace physical unit 4, 5, or 6.

Backspace one record (character) on the selected physical unit; it is activated by depressing Key 21, 22, or 23.

Key 24: Find a character on the input tape.

This function allows the operator to resume processing from the last character processed in the previous editing session. The function requires that the card punched when Key 5 (write EOF on units 4, 5, 6) was activated be in the card reader following the "EXECUTE" card. When Key 24 is depressed, the program will read the card in the reader and read the characters from the input tape (unit 2) until a match on character ID and author ID has been found.

Key 25: Not Used.

Key 26, 27, & 28: Skip to end-of-file on units 4, 5, or 6 (the output tapes).

These functions, when selected, will advance the selected tape to the last record previously written on the tape.

Key 29: Not Used.

Key 30: Write the character on physical 4 and if an alpha character on physical 5.

When the operator decides a character is acceptable for inclusion in the data base, he will depress Key 30. The resulting operation is to write the character on physical unit 4 regardless of nature (alpha/numeric) and if the character is alpha it will also be written onto physical unit 5.

Note: The two tape options, Key 20 and Key 30, will effect a write to the appropriate unit and will automatically read the next character on the input tape (physical 2).

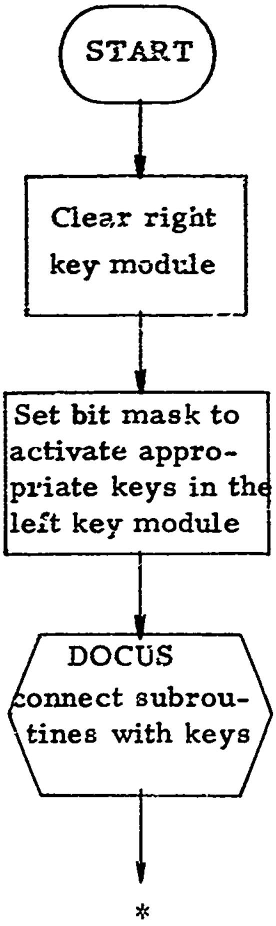
Tape Setup

Tape Unit		
Physical	Logical	Description
1		System master
2	10	Input character tape
3		Edit overlay tape (binary)
4	11	Alpha/Numeric output tape
5	12	Symbols output tape
6	13	Difficult character output tape.

1 Read a character from input tape	2 Not Used	3 Backspace input tape one char.	4 Not Used	5 Write EOF on output tape 4, 5, 6
6 Not Used	7 Delete a row of noise	8 Delete a col. of noise	9 Delete a point of noise	10 Delete Col. 1
11 Change Character ID	12 Not Used	13 Restore Last Deletion	14 Set output author number	15 Delete Col. 2
16 Not Used	17 Sort the full set tape (2)	18 Not Used	19 Not Used	20 Write character on 6
21 Backspace 4	22 Backspace 5	23 Backspace 6	24 Find a character on input tape	25 Not Used
26 Skip to EOF on 4	27 Skip to EOF on 5	28 Skip to EOF on 6	29 Not Used	30 Write character on 4 or 5

PROGRAM: TAG START

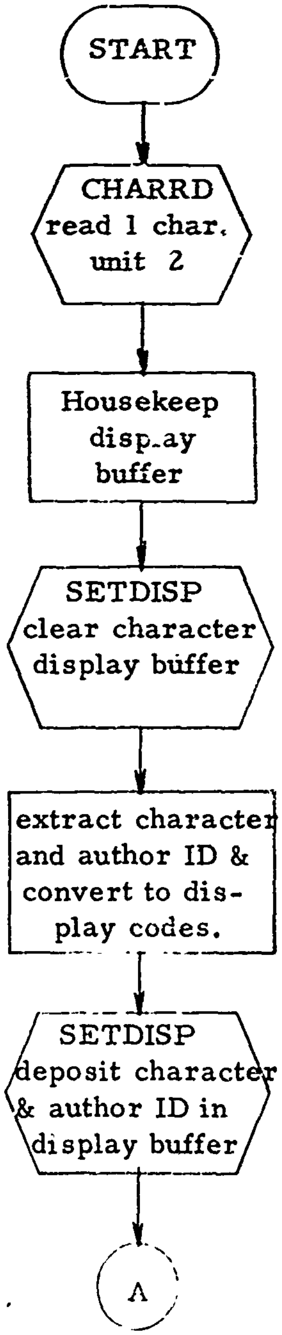
Purpose: This is the EDIT package program; it connects the subroutine entry point addresses with the appropriate keys on the BR-85.

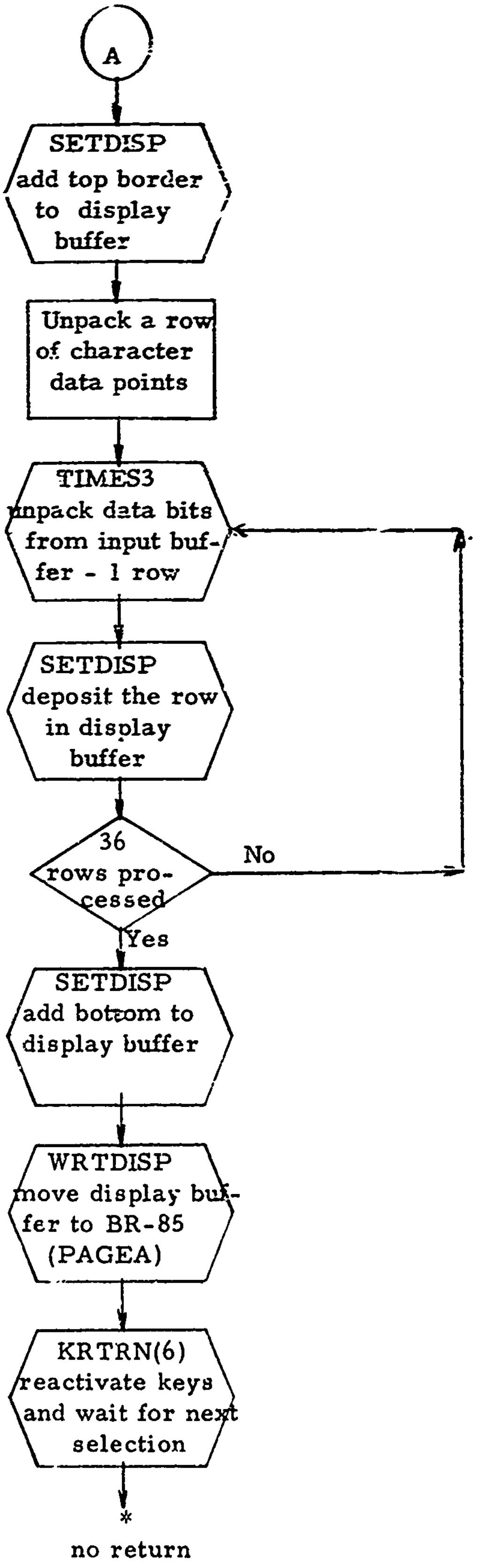

START
Clear right key module
Set bit mask to activate appropriate keys in the left key module
DOCUS connect subroutines with keys
*

* - There is no return from DOCUS; control is transferred to the first key program (Key #1 - "CONTROL").

SUBROUTINE: CONTROL

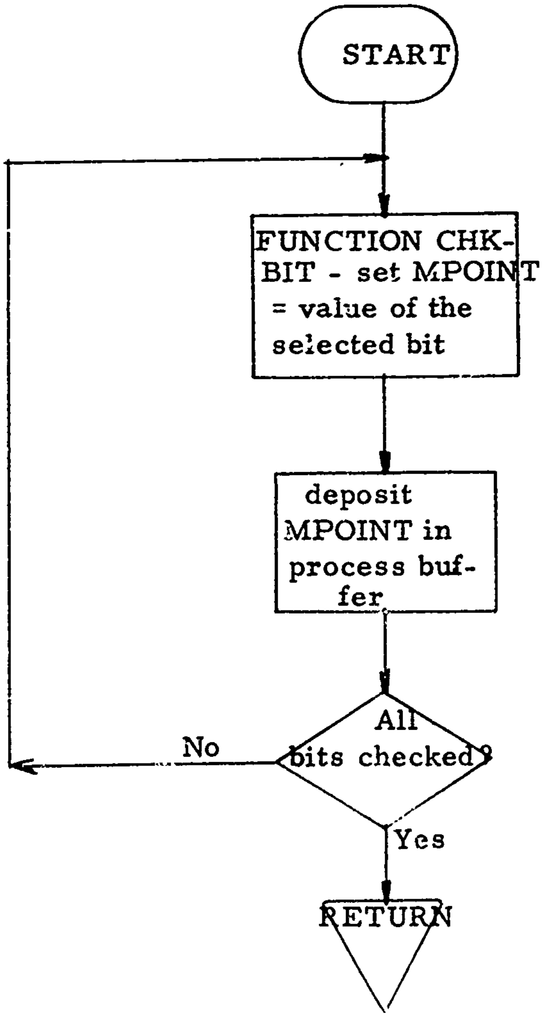
Purpose: CONTROL will read one character from the input tape (physical 2) and display the character matrix on the BR-85. In addition it extracts and displays the character and author ID.




A
SETDISP
add top border to display buffer
Unpack a row of character data points
TIMES3
unpack data bits from input buffer - 1 row
SETDISP
deposit the row in display buffer
36 rows processed
No
Yes
SETDISP
add bottom to display buffer
WRTDISP
move display buffer to BR-85 (PAGEA)
KRTRN(6)
reactivate keys and wait for next selection
*
no return

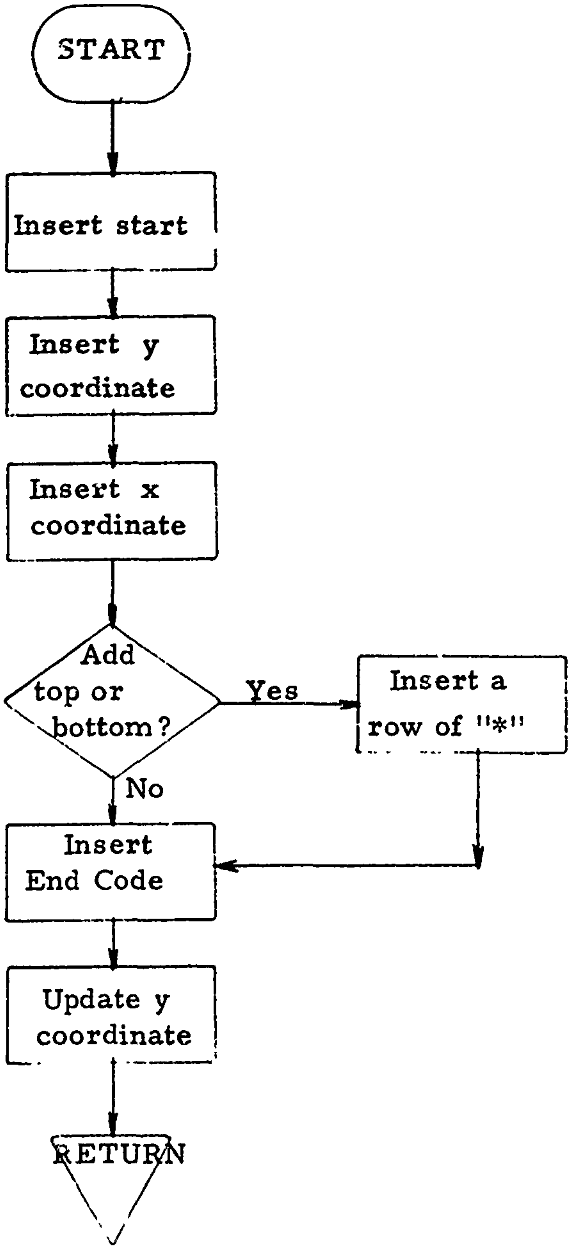
SUBROUTINE: TIMES3

Purpose: Extract character data bit from input buffer and move it to the process buffer (3 bits at a time)


START
FUNCTION CHK-BIT - set MPOINT = value of the selected bit
deposit MPOINT in process buffer
All bits checked?
No
Yes
RETURN

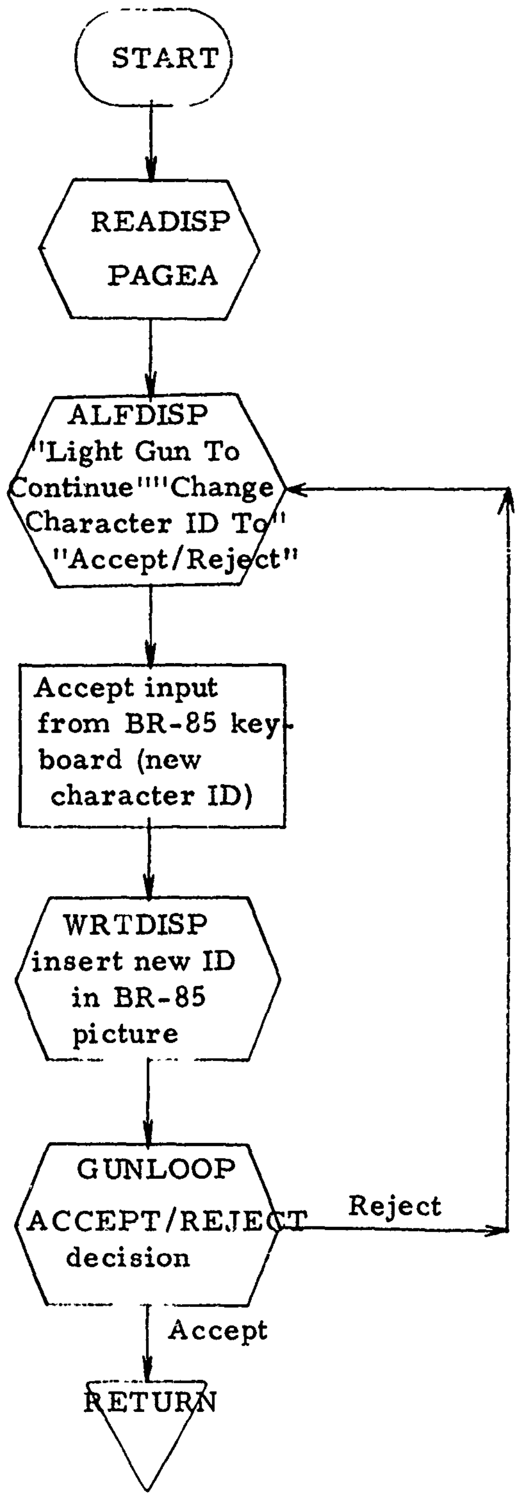
SUBROUTINE: SETDISP

Purpose: Insert BR-85 control and position character for one display line in the process buffer.



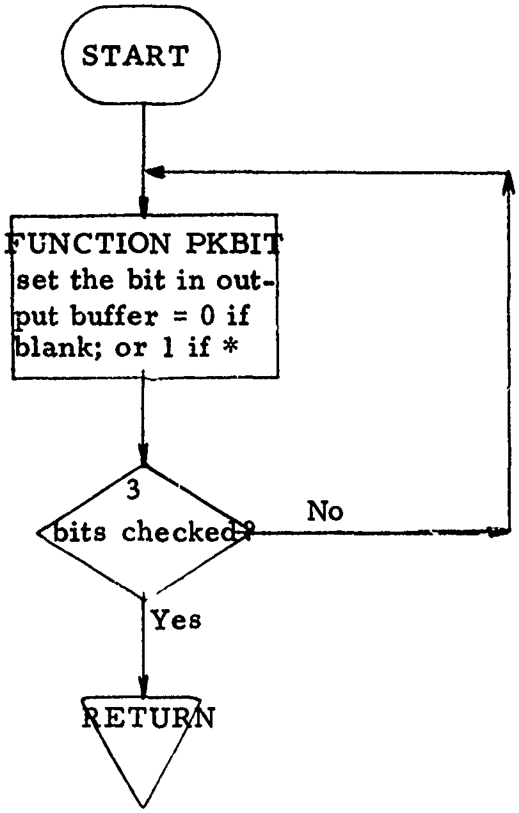
SUBROUTINE: CHARID

Purpose: Change the ID of the current BR-85 display.



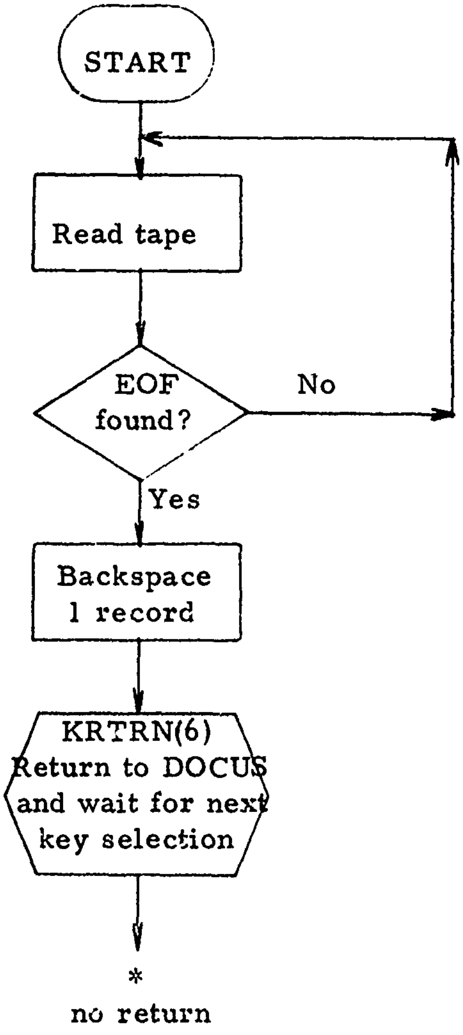
SUBROUTINE: SETHREF

Purpose: Used in construction of a character mat.ix. This matrix is generated from the BR-85 display buffer which contains all modifications made by the operator (3 bits at a time).


START
FUNCTION PKBIT set the bit in output buffer = 0 if blank; or 1 if *
3 bits checked?
No
Yes
RETURN

SUBROUTINE: SKIP11, SKIP12, SKIP13

Purpose: Advance the appropriate tape to the EOF and backspace over the EOF mark.

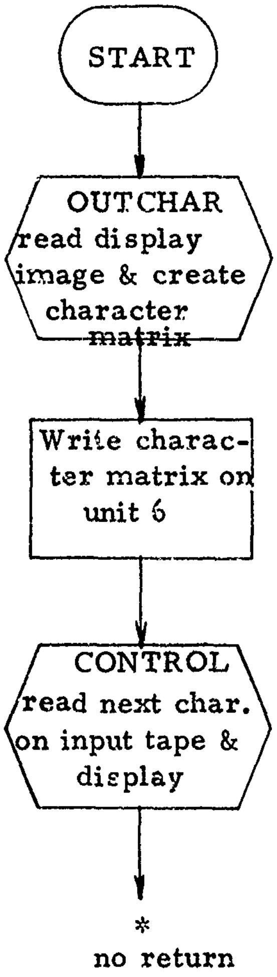


SUBROUTINE: MBOB

Purpose: This is a dummy routine and should never be entered. Its function is to return to DOCUS if an illegal function key is depressed and honored.

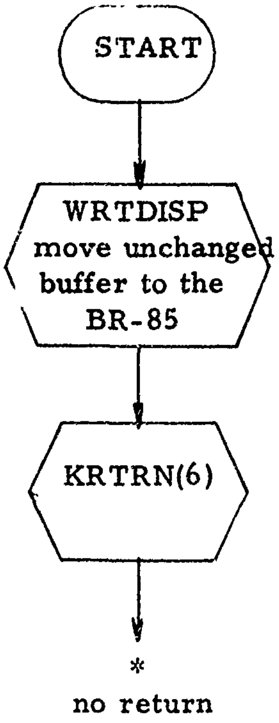
SUBROUTINE: HARD

Purpose: Reconstruct a character matrix from the BR-85 display image and write it on physical unit 6.


START
OUTCHAR
read display image & create character matrix
Write character matrix on unit 6
CONTROL
read next char. on input tape & display
*
no return

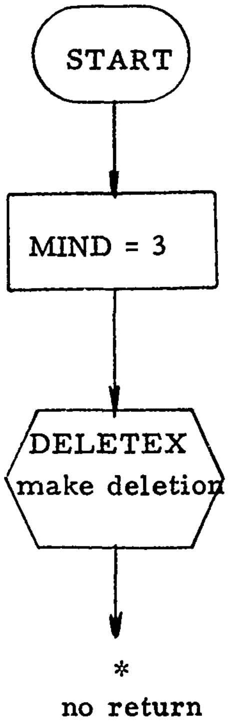
SUBROUTINE: RESTORE

Purpose: Restore the last deletion (row, column, point) made by the operator. When the operator selects any of the delete keys the display image is read into two buffers; one buffer is saved unchanged and the other is used to make the modification. The modified buffer is then transferred to the BR-85. When this program is activated it transfers the unchanged buffer to the BR-85.



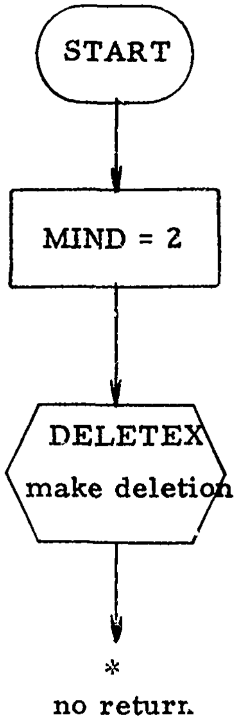
SUBROUTINE: DPOINT

Purpose: Set parameter MIND=3 to indicate a delete point operation is to be executed.



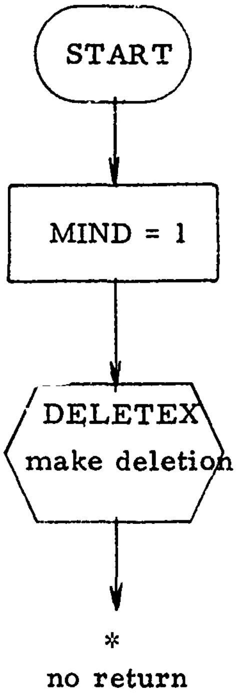
SUBROUTINE: DCOL

Purpose: Set parameter MIND=2 to indicate a delete column operation is to be executed.



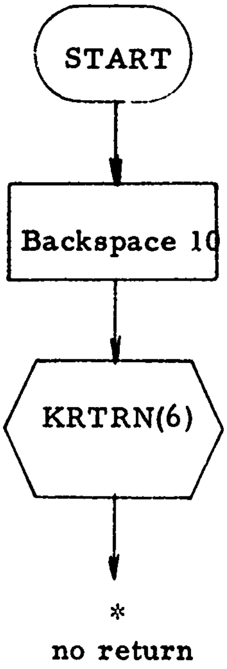
SUBROUTINE: DROW

Purpose: Set parameter MIND=1 to indicate a delete row operation is to be executed.



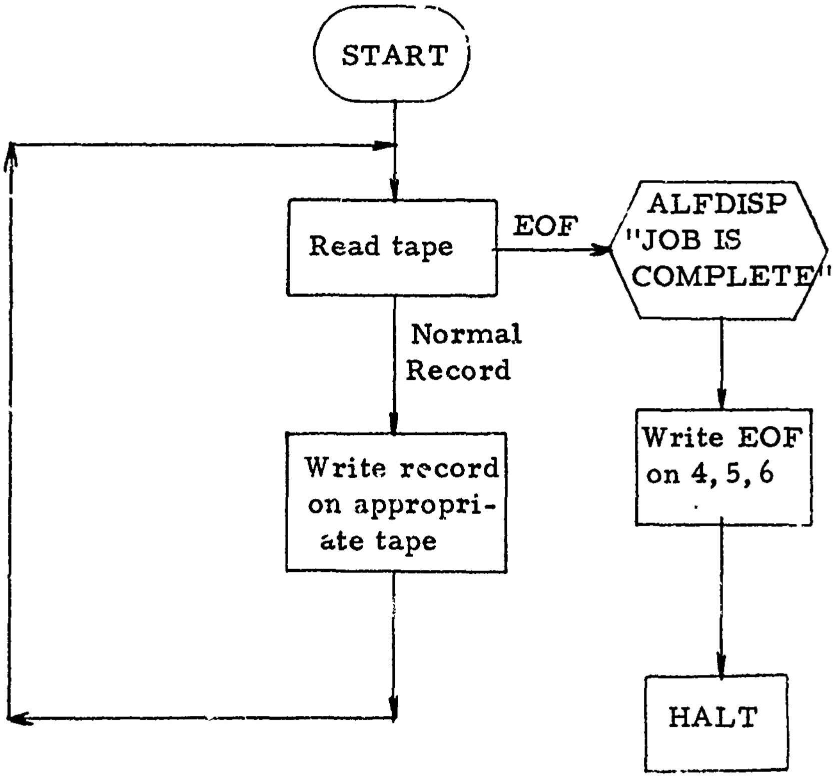
SUBROUTINE: TBACK

Purpose: Backspace input tape (physical 2) one record.



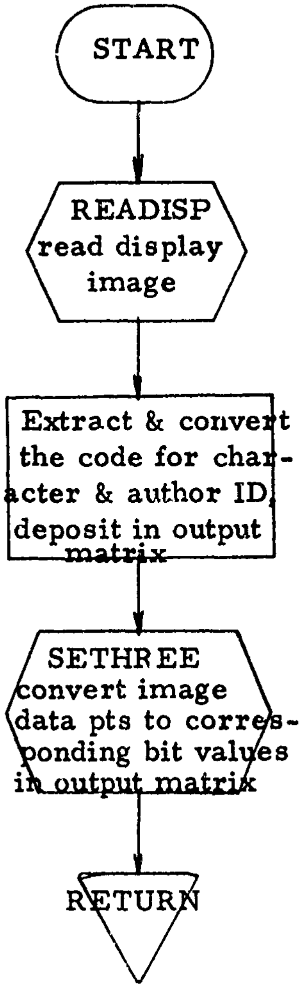
SUBROUTINE: NEWTAPE

Purpose: Sort the input tape (physical 2) and write the numerics on physical 4, the alpha characters C, X, T, X, and Z on physical 5, and special symbols on physical 6.



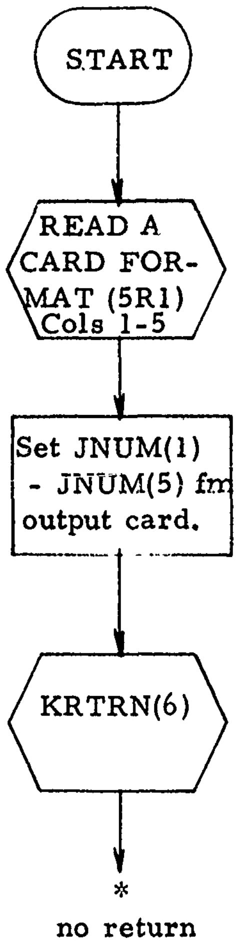
SUBROUTINE: OUTCHAR

Purpose: Retrieve the current BR-85 display image of the character in process and reconstruct the character matrix. This format is identical to the PDP-8 image dissector matrix, however it reflects any changes made by the operator.


START
READISP
read display image
Extract & convert the code for character & author ID, deposit in output matrix
SETHREE
convert image data pts to corresponding bit values in output matrix
RETURN

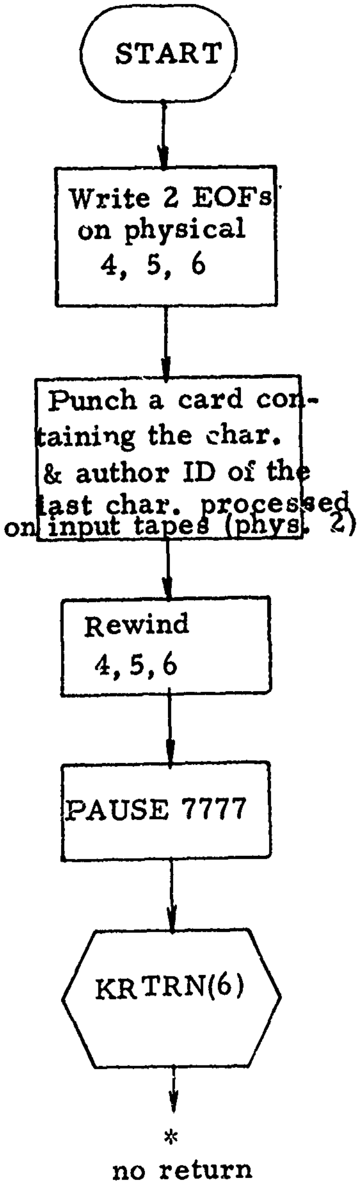
SUBROUTINE: SETA

Purpose: Set the author number to desired value. This operation permits the operator to set the author number to be assigned to the next output character written on either physical 4, 5, or 6.


START
READ A CARD FORMAT (5R1) Cols 1-5
Set JNUM(1) - JNUM(5) fm output card.
KRTRN(6)
*
no return

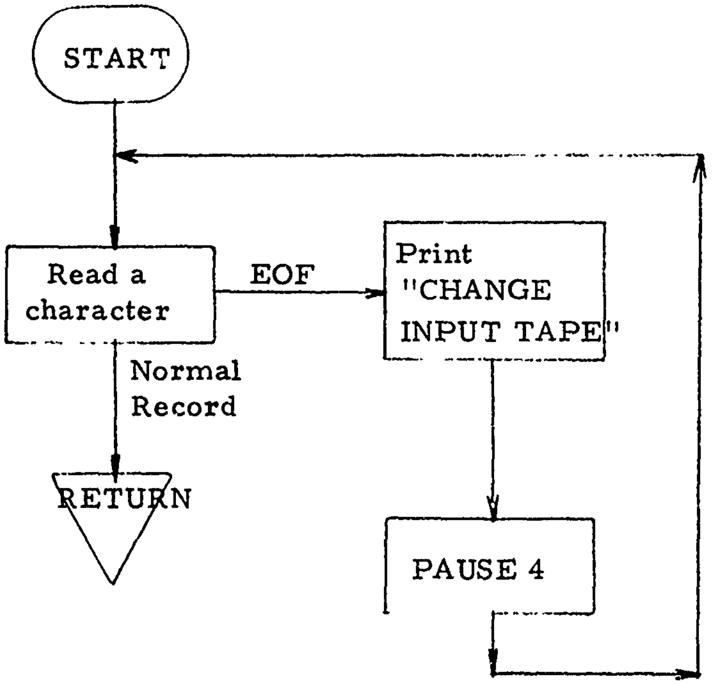
SUBROUTINE: WEOF

Purpose: Write EOF on the three output tapes (physical 4, 5, and 6).


START
Write 2 EOFs on physical 4, 5, 6
Punch a card containing the char. & author ID of the last char. processed on input tapes (phys. 2)
Rewind 4, 5, 6
PAUSE 7777
KRTRN(6)
*
no return

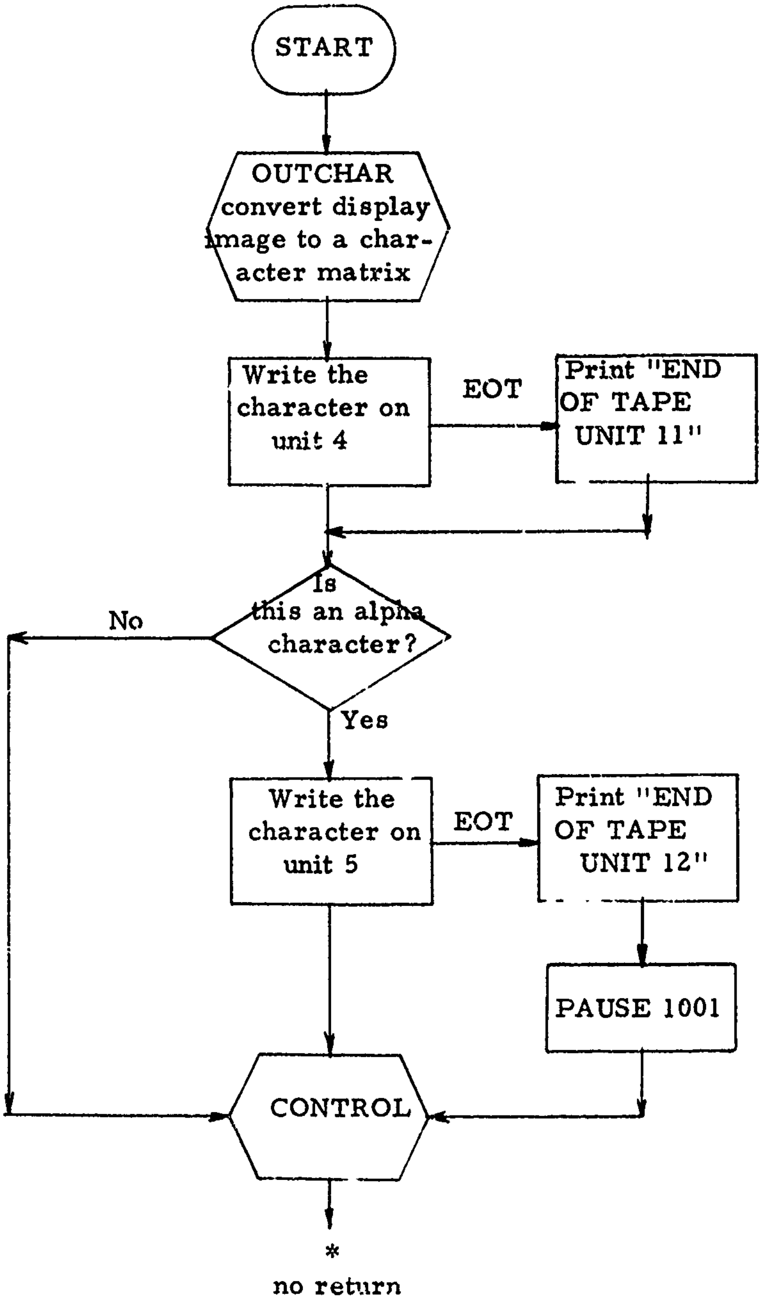
SUBROUTINE: CHARRD

Purpose: Read a character from the input tape (physical 2)



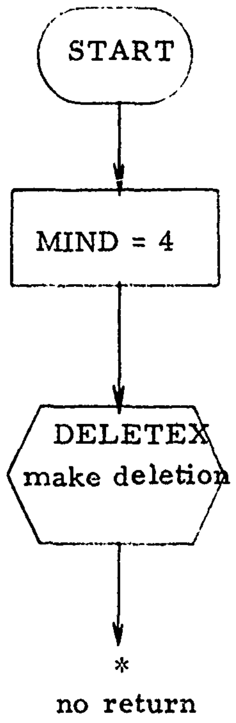
SUBROUTINE: CHARWRT

Purpose: Write the curr=nt BR-85 display image on the appropriate output tape(s).


START
OUTCHAR convert display image to a character matrix
Write the character on unit 4
EOT
Print "END OF TAPE UNIT 11"
Is this an alpha character?
No
Yes
Write the character on unit 5
EOT
Print "END OF TAPE UNIT 12"
PAUSE 1001
CONTROL
*
no return

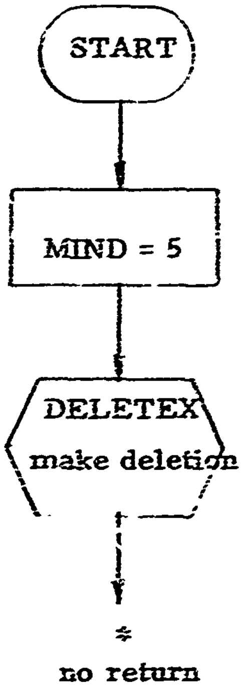
SUBROUTINE: DONE

Purpose: Set parameter MIND-4 to indicate that row one is to be deleted from the display image.



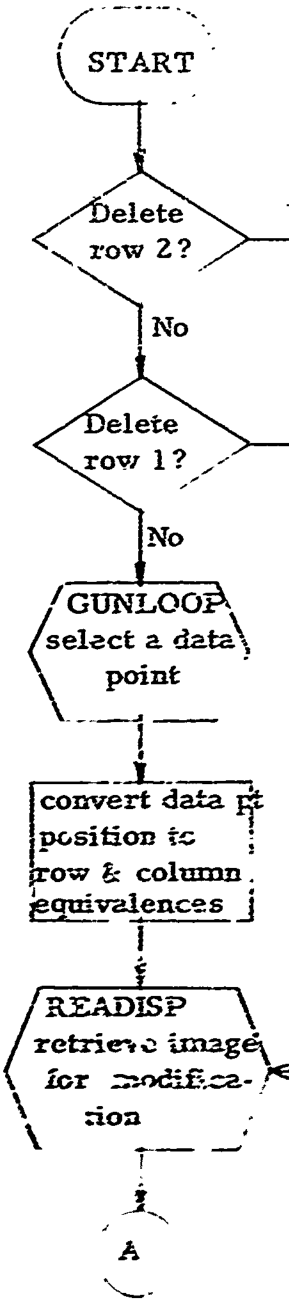
SUBROUTINE: DTWO

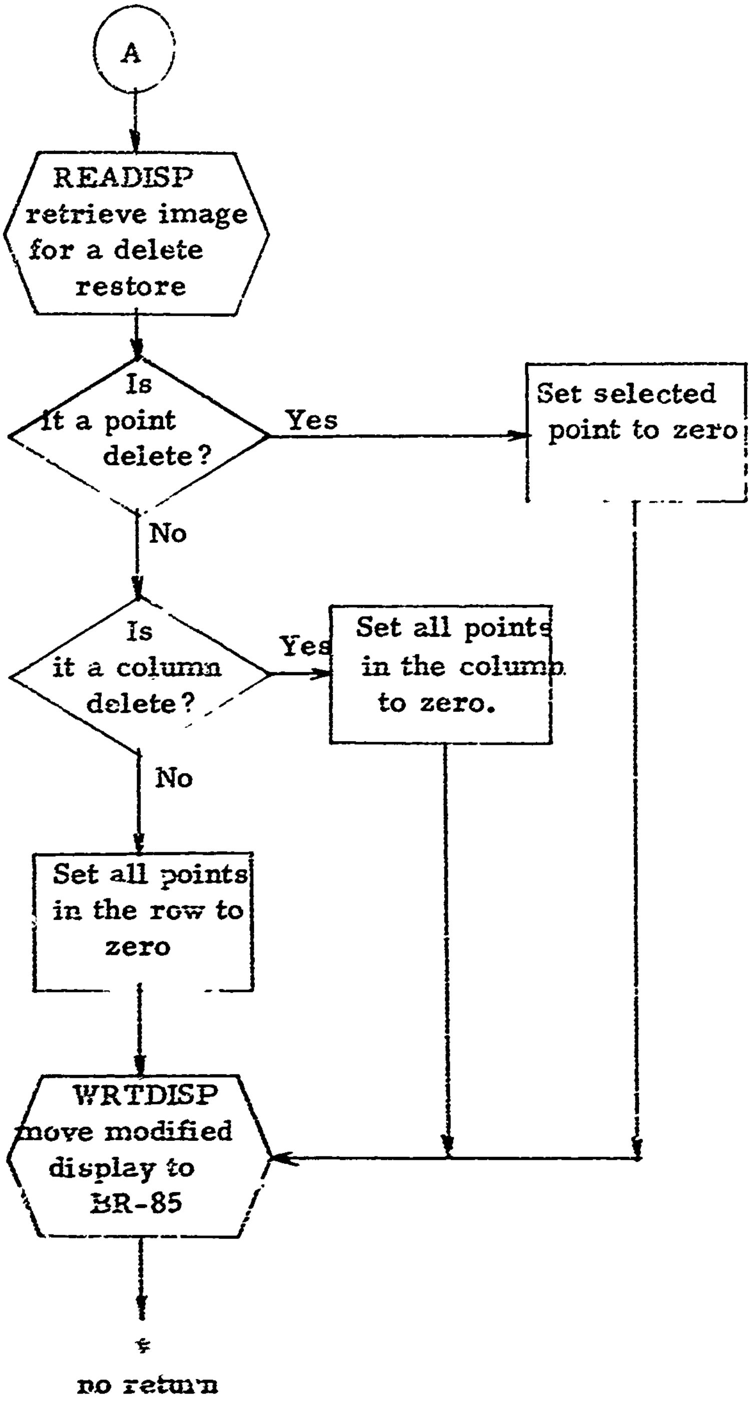
Purpose: Set parameter MIND=5 to indicate that row two is to be deleted.



SUBROUTINE: DELETEX

Purpose: This program will identify the delete operations selected by the operator, retrieve the current BR-85 display image, make the appropriate changes to that display and transfer the new display image to the BR-85.


START
Delete row 2?
Yes
No
Delete row 1?
Yes
No
GUNLOOP select a data point
convert data pt position to row & column equivalences
READISP retrieve image for modification
A


A
READISP
retrieve image
for a delete
restore
Is
it a point
delete?
Yes
Set selected
point to zero
No
Is
it a column
delete?
Yes
Set all points
in the column
to zero.
No
Set all points
in the row to
zero
WRTDISP
move modified
display to
BR-85
no return

The following routines were written for OLPARS and are documented in Final Report to Contract F30602-71-C-0367, RADC-TR-72-71, March 1972:

WRTDISP
KRTRN(6)
DOCUS
ALFDISP
READISP
GUNLOOP
FRAMER
CHKBIT
PKBIT

77

6.2. COMPUTER/RUN DECK SET UP

6.2.1. Density Conversion

The input tape to the Edit package is a 556 BPI tape with one character per physical record. This tape is generated on the 645 by using the original character generated on the PDP-8 at 800 BPI.

The following program will affect the 645 density conversion:

```
$       IDENT     /ISCP,NAME   ,OCR   ,00331,5581
$       OPTION FORTRAN
$       FORTRAN LSTOU
$       INCODE IBMF
        DIMENSION M(64)
        CALL FLGEOF(10,NE)
        CALL FXOPT(40,1,1,0)
10      READ (10)M
        IF (NE.EQ.1) GO TO 300
        WRITE (11)M
        GO TO 10
300     CALL FCLOSE(10)
        END FILE 11
        STOP
        END
$       EXECUTE DUMP
$       LIMITS  23,  11K
$       FFILE 10,NBUFFS/2,BUFSIZ/67,MLTFIL,FIXLNG/164,NSTDLB,NOSRLS
$       TAPE   10,AID,,NAME
$       FFILE 11,NBUFFS/2,BUFSIZ/67,MLTFIL,FIXLNG/64,NSTDLB,NOSRLS
$       TAPE  11,BID,,99999
$       ENDJOB
***EOF
```

Note: In the above program tape unit 10 is the input and unit 11 is the output tape. The "JOB" card must specify a density of 800 BPI for the input and 556 BPI for the output tape.

6.2.2. Edit Package

Tape Assignment

To exercise this package the following tape assignments are required:

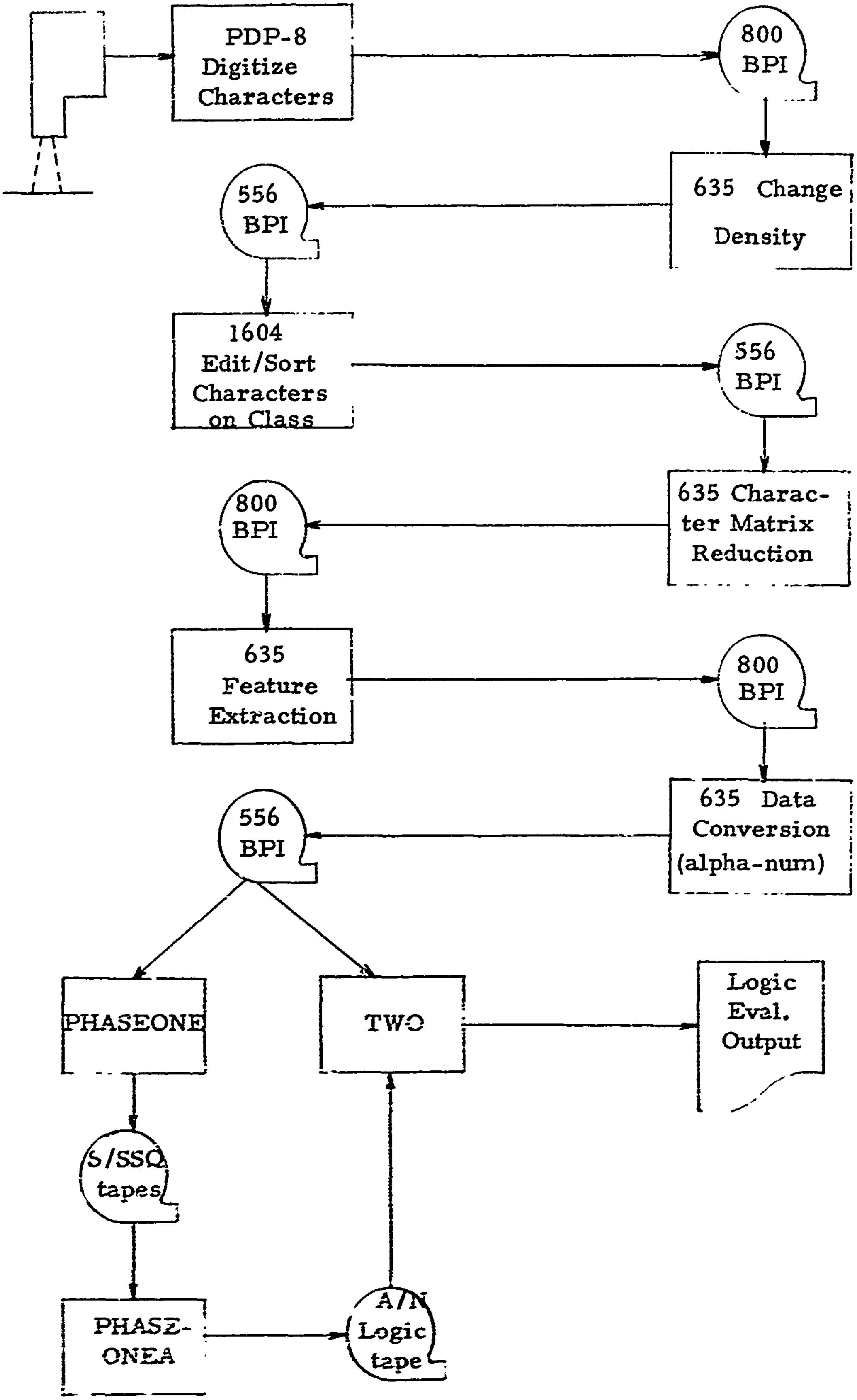
Tape Unit		
Physical	Logical	Description
1	1	System master with punch
2	10	Input character tape
3	3	Edit overlay tape
4	11	Alpha/numeric output tape
5	12	Symbol output tape
6	13	Difficult character output tape

Run Deck Setup

(* is a 7,9 punch in column 1)

```
*BEGIN JOB EDIT
*COOP, 116, EDIT, I/10/S/11/12/13/56, 60, 99999₀4, EDIT
*EXECUTE, , 56.
***
BLANK CARD (2 cards)
```

*** may be either a blank cord or it may contain the character and author ID generated by the operator in a previous editing run when he selected Key 5 - WRITE EOF ON OUTPUT TAPES to terminate processing.


PDP-8
Digitize
Characters
800
BPI
635 Change
Density
556
BPI
1604
Edit/Sort
Characters
on Class
556
BPI
635 Character Matrix
Reduction
800
BPI
635
Feature
Extraction
800
BPI
635 Data
Conversion
(alpha-num)
556
BPI
PHASEONE
TWO
Logic
Eval.
Output
S/SSQ
tapes
PHASE-
ONEA
A/N
Logic
tape

PDP-8 CHARACTER DIGITIZE - This program (provided by RADC) controls the IMAGE DISSECTOR (ITT) and converts each hand-printed character to a matrix of 36 rows by 30 columns.

A character ID, determined by the character's position on the input form, and an author number is assigned, added to the digitized matrix and written on the output tape.

Figure -1 is a sample form used in the collection of data.

635 TAPE DENSITY CHANGE - Since the PDP-8 has a fixed density of 800 BPI and the CDC-1604 requires a density of either 256 or 556 BPI, this program is required. The only function performed is a density change, matrix size and organization are not affected.

CDC-1604 EDIT/SORT - The EDIT program is provided to compensate for human and hardware error and thereby possibly save a character that otherwise would have been deleted from the data base. Some typical problems that may be resolved are:

a. Mislabeled characters - an "A" character written on the form in a box where a "B" should have been.
b. Camera noise indicating a non-existent data point.
c. An incomplete erasure with a cha [illegible] erwrite.
d. Character overwrite without an [illegible]
e. A character written so lightly that [illegible] atrix is incomplete.

The 1604 SORT program processes a tape with a random sequence of characters and produces an output tape with all characters of a given class (class A, B, 1, 2, etc.) grouped in consecutive records.

635 MATRIX REDUCTION - This routine is necessitated by the conflict in the matrix size (36 x 30) of a character digitized by the PDP-8 and the expected matrix size (24 x 16) of the FEATURE EXTRACTION program. The various algorithms in the FEATURE EXTRACTION subsystem were designed using a matrix of 24 x 16; any variation in this matrix size would render these algorithms useless.

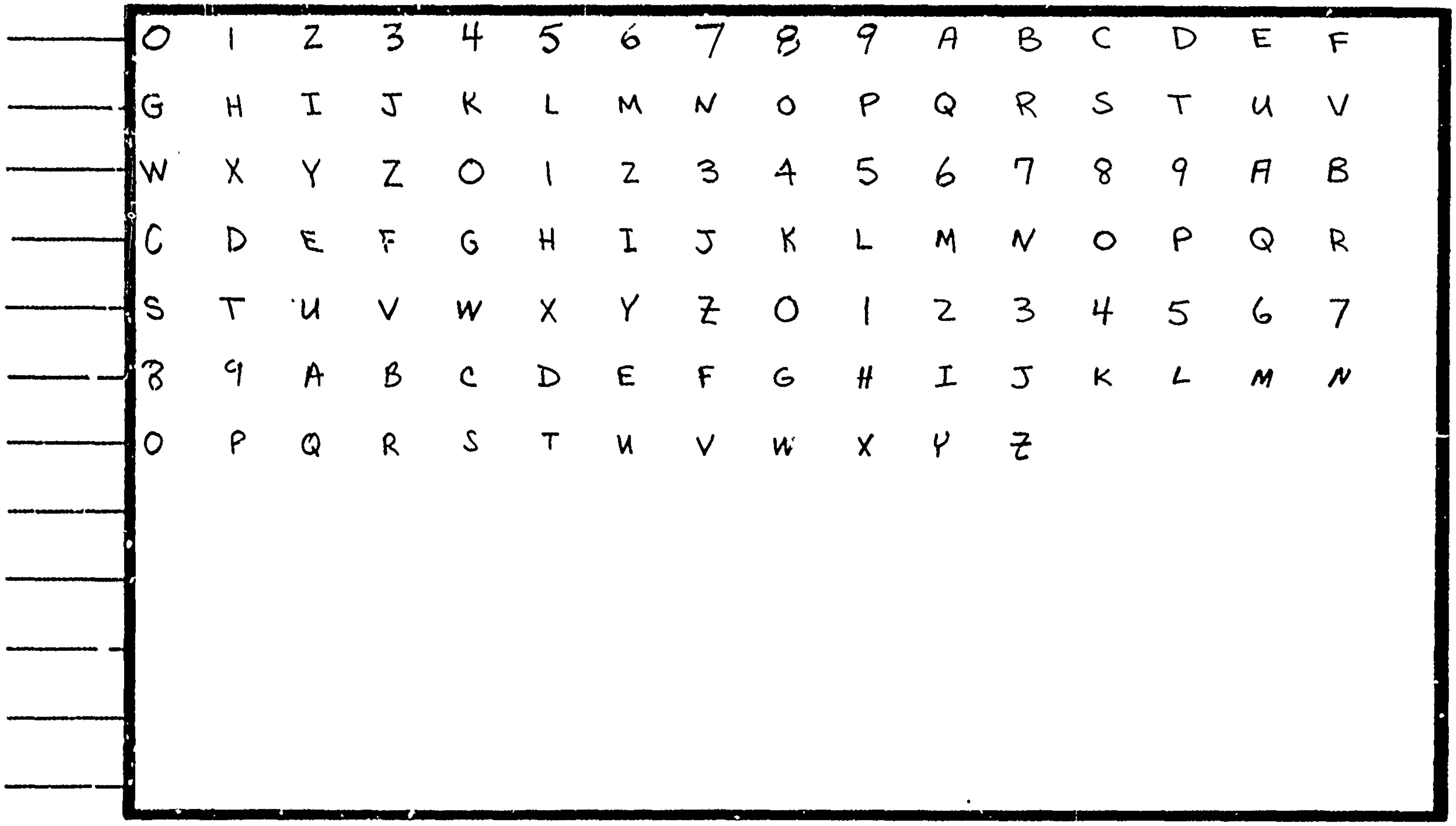
635 FEATURE EXTRACTION - These algorithms convert a digitized character matrix into an OLPARS vector that, when in correct format, may be used for either character recognition design or character evaluation.

635 DATA CONVERSION - The input tape to the Edit package is a 556 BPI tape with one character per physical record. This tape is generated on the 645 by using the original character generated on the PDP-8 at 800 BPI.

PHASEONE The PHASEONE(P1) package computes an array of sums and a matrix of squared sums for each character for which logic is desired. The input to P1 consists of a series of feature vector tapes generated on the Honeywell 635 computer as described previously. The output from P1 is up to three (3) sum/sum square (S/SSQ) tapes which are used as input to the PHASONEA(P1A) package.

PHASONEA The PHASONEA (P1A) package computes the logic for all pairs of classes in the design character set. Logic is designed for one look pair (TR, TB, TL, RB, RL, or BL) for each pair of classes (36 classes results in 630 class pairs) for each of the 9 sort classes. Therefore, the decision logic for 36 classes contains 5670 (9 x 630) decision points. The input to P1A consists of the set of up to 3 S/SSQ output tapes from P1 and a deck of cards consisting of one card for each class pair with the pair looks to be used for that class pair. The output is a logic tape containing one record for each class pair and a printout of the logic. P1A may be started at any designated class pair and terminated after any desired class pair. A set of utility programs for merging a series of logic tapes generated by P1A is provided in the utility package (a complete logic tape for 36 classes will consist of 630 records).

TWO The TWO package evaluates the vectors on the Standard Input Tape (SIT) against logic generated by the P1A package.



0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	A	B
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z	0	1	2	3	4	5	6	7
8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N
O	P	Q	R	S	T	U	V	W	X	Y	Z				

Figure 6. 1 - Sample Form used for collection of data.

SECTION 7

ALPHA NUMERIC LOGIC PACKAGE (ANLP)

The ANLP performs the generation of discrimination logic, preliminary evaluation, and independent testing of handprinted alpha-numeric characters. Three major program packages have been developed to perform these tasks along with a small set of utility programs for tape editing. Each set of programs will process up to 48 separate characters; however, for the current contract only the 10 numerics and 26 alpha characters have been processed. Each program will be described in the following format: (1) an introduction to the purpose and general operation of the program; (2) a description of the input cards and tapes, output cards and tapes, and general run time instructions; (3) a listing of possible program pauses and corrective measures (if possible); and (4) a functional flow diagram of the program package.

To generate and test logic via the ANLP package, then, requires the user to operate the following program packages in sequence: PHASEONE, PHASONEA, TWO; where PHASEONE computes a measurement sum array and scatter matrix for each possible look pair from the Standard Input Tape (SIT), PHASONEA utilizes PHASEONE output and the pair look deck to generate a final logic tape, and TWO evaluates the logic against vectors on any SIT. Two basic inputs, then, are required by ANLP: The SIT and the Pair Look Deck, Page A-8, A-9, A-13, A-14, A-15/A-16.

7.1. PHASEONE

The PHASEONE(P1) package computes an array of sums and a matrix of squared sums for each character for which logic is desired. The input to P1 consists of a series of feature vector tapes generated on the Honeywell 635 computer as described previously. The output from P1 is up to three (3) sum/sum square (S/SSQ) tapes which are used as input to the PHASONEA(P1A) package.

OLPARS Alpha-Numeric Data Tape Format

The data tape for the ANLP package (P1 and TWO) is represented symbolically as follows:

B	2113
R	2113
T	2113
L	2113
A	2613
C	A1
S	15
CB	I1
CR	I1
CT	I1
CL	I1

B: Bottom view measurements, 21 numbers, 3 characters/number
R: Right view measurements, 21 numbers, 3 characters/number
T: Top view measurements, 21 numbers, 3 characters/number
L: Left view measurements, 21 numbers, 3 characters/number
A: Additional measurements, 26 numbers, 3 characters/number
C: Character Symbol
S: Sequence number
CB: Number of bottom convexities
CR: Number of right convexities
CT: Number of top convexities
CL: Number of left convexities

The input tape from the 635 will be a BCD tape. Each vector on the tape will consist of three BCD records, each record consisting of 120 decimal characters. Each vector is broken into three parts: (1) standard measurements, (2) additional measurements, and (3) identification data.

(1) Standard Measurements

The standard measurements occupy the first 252 decimal characters of each vector. Each measurement occupies three decimal characters and includes a sign of two decimal digits. The total of 84 standard measurements are divided into four equal subgroups of 21 measurements each. The four subgroups represent the four "looks" used to measure the character; namely, bottom, right side, top and left side, in that order. Each subgroup is further divided into five convexities. The first convexity pertains to the first five measurements of the subgroup, the second through fifth convexities each pertain to four measurements, making a total of 21 measurements in all. The subgroups always contain 21 measurements even when only three or one convexities are actually supplied. If one convexity is supplied, it will occupy the first five measurements of the subgroup, with the remaining 16 measurements set to zero. If three convexities are supplied, they will occupy the first 13 measurements of the subgroup, with the remaining 8 measurements set to zero. Each of the four subgroups will always contain at least one convexity.

(2) Additional Measurements

Following the 252 decimal characters which contain the standard measurements are 78 decimal characters which contain room for 26 additional measurements. Each measurement occupies three decimal characters and includes a sign and two decimal digits. Eight additional measurements were calculated from the character to further help recognition. The number of additional measurements to be used for each character is supplied by the pair-look parameter deck. The eight additional features supplied for each character occupy the first eight additional measurement positions for this vector with the remaining 13 measurements set to zero.

(3) Identification Data

Following the 330 decimal characters which contain the standard and additional measurements, are ten decimal characters which contain identification data pertaining to this vector. The first character of the identification data contains the actual character in question. The next 5 characters contain the author ID, as four digits and a sign. The remaining four characters contain four decimal digits which pertain to the four looks described above. Each digit contains either a 1, 3, or 5, depending upon the number of convexities contained in the corresponding subgroup within the standard measurement portion of this vector. The first digit corresponds to the first subgroup, the second digit to the second subgroup, etc. The remaining 20 decimal characters of the vector are not used.

Pair Look Deck Format

The pair-look parameter deck contains pertinent information concerning all possible pairs for a given 635 BCD input tape. Needed for each possible pair is such information as which looks are to be used when evaluating the pair, as well as the number of additional measurements to be used for this pair (see (2) above). To facilitate easy updating of this deck, each pair will be represented by a separate card. Each card will have the following format:

Columns	
1	Class 1 character
2	Class 2 character
3-7	Blank
8	Look 1 (1 = bottom, 2 = right side, 3 = top, 4 = left side)
9	Blank
10	Look 2 (same as Look 1)
11-16	Blank
17-18	Number of additional measurements to be used for this pair, right adjust.

For each character, fifty-four (54) records (6 "look" pairs of 9 sort classes each) are written on the S/SSQ tapes in the following sequence:

Look Pair #				Sort Class #	Record #'s
	Looks TR*	Sort Class	5,5	1	
			5,3	2	
			5,1	3	
			3,5	4	
1			3,3	5	1-9
			3,1	6	
			1,5	7	
			1,3	8	
			1,1	9	
2	Looks TB				10-18
3	Looks TL				19-27
4	Looks RB				28-36
5	Looks RL				37-45
6	Looks BL				46-54

* - T indicates top view, R - right side, B - bottom, L - left side

Thus, each vector input to P1 contributes to six records, one for each look pair. Each S/SSQ tape contains the data for up to sixteen (16) alpha-numeric characters. In addition, a header record is written on each S/SSQ tape containing the number of classes to be input and their symbols in sequence. Under this contract, S/SSQ #1 has characters 0 - F (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F); SSQ #2 has G - V (G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V) and SSQ #3 contains W - Z (W, X, Y, Z). The P1 package may operate in the tape initiate or tape modify mode. In the initiate mode, the 54 records for each character are written in the order of character input. The tape modify mode reads the previous S/SSQ matrix for the character onto the data drum of the 1604B, adds the information from the new vectors to the various matrices, and outputs the modified records. In both modes, a restart capability allows the user to commence operation with any character. Sense switch settings allow the user control over system wrap-up and data printouts. In either mode, vectors may be eliminated from consideration by use of a discard vector tape generated by program MAKE (See Utilities Section).

Run Time Instructions for PHASEONE

(1) Compile and Execute the P1 package.

(2) The program will halt at Pause 111 (111_8 in the A register on the 1604 console) to allow for mounting of tapes.

(a) Mount a scratch tape on physical tape drive 3 to be used as the output S/SSQ tape.

(b) Mount the appropriate current S/SSQ tape on physical tape drive 4 (the appropriate S/SSQ tape is that tape containing the character with which the current run will begin input).

(c) Mount the appropriate input tape on physical tape drive 5.

(d) Mount the Discard Vector tape (if appropriate) on physical tape drive 6.

(3) A number of sense switch (SS) settings are available which allow the user to control program wrap-up and printouts. To set a sense switch during program operation the user may hit the carriage return, then type "f, n/1", where n is the sense switch number to be set. The procedure for setting and clearing sense switches is further discussed in the COOP Users' Guide, Page 5-1. For the P1 package, setting SS1 will cause the program to complete its output at the completion of 'he current data tape;

SS2 will have the same effect at the completion of the current data character input; SS3 will suppress printout of the feature values of each input vector; and SS4 will suppress printout of the entire vector, heading as well as feature values.

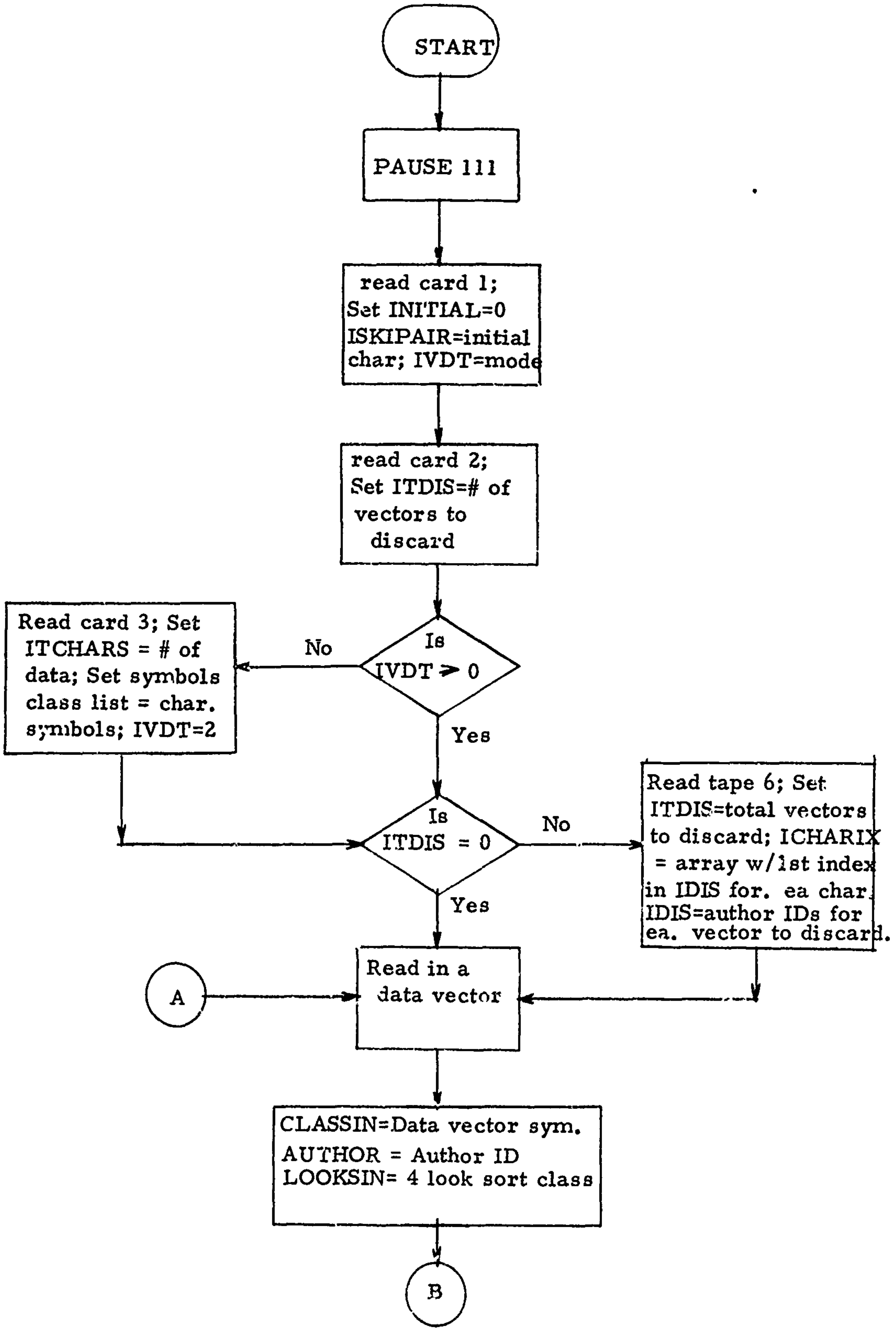
(4) Two data cards are required for the operation of P1:

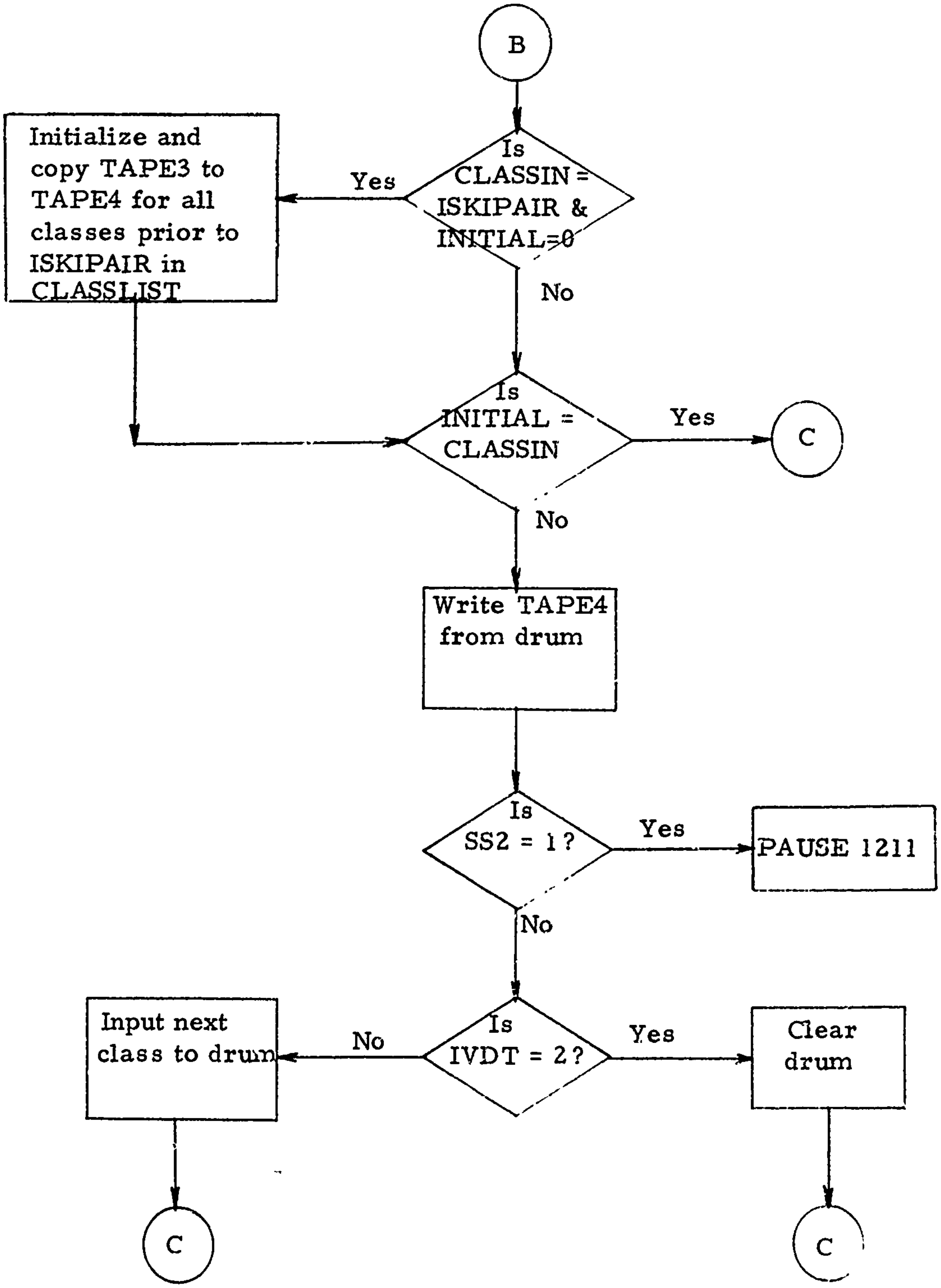
(a) A card with column 1 punched with the initial character to be processed under the current run. P1 will search the input tape for this character and will copy the current S/SSQ tape onto the new S/SSQ tape those characters which fall before the initial character on the appropriate S/SSQ tape. Column 2 of this card contains the program mode indicator. For Mode = 0, P1 assumes no previous data output and expects a third input card to follow card 2. This mode is to be used only when initializing a set of S/SSQ tapes. When Mode = 1, an update procedure will be performed. S/SSQ tape records will be expected on the current tape, and will be added to the vectors input from the current data input tape. For Modes 2, S/SSQ records will be created entirely from the input data set and the current S/SSQ tape will only be utilized to copy those S/SSQ's which fall prior to the initial character to be processed on the current input operation.

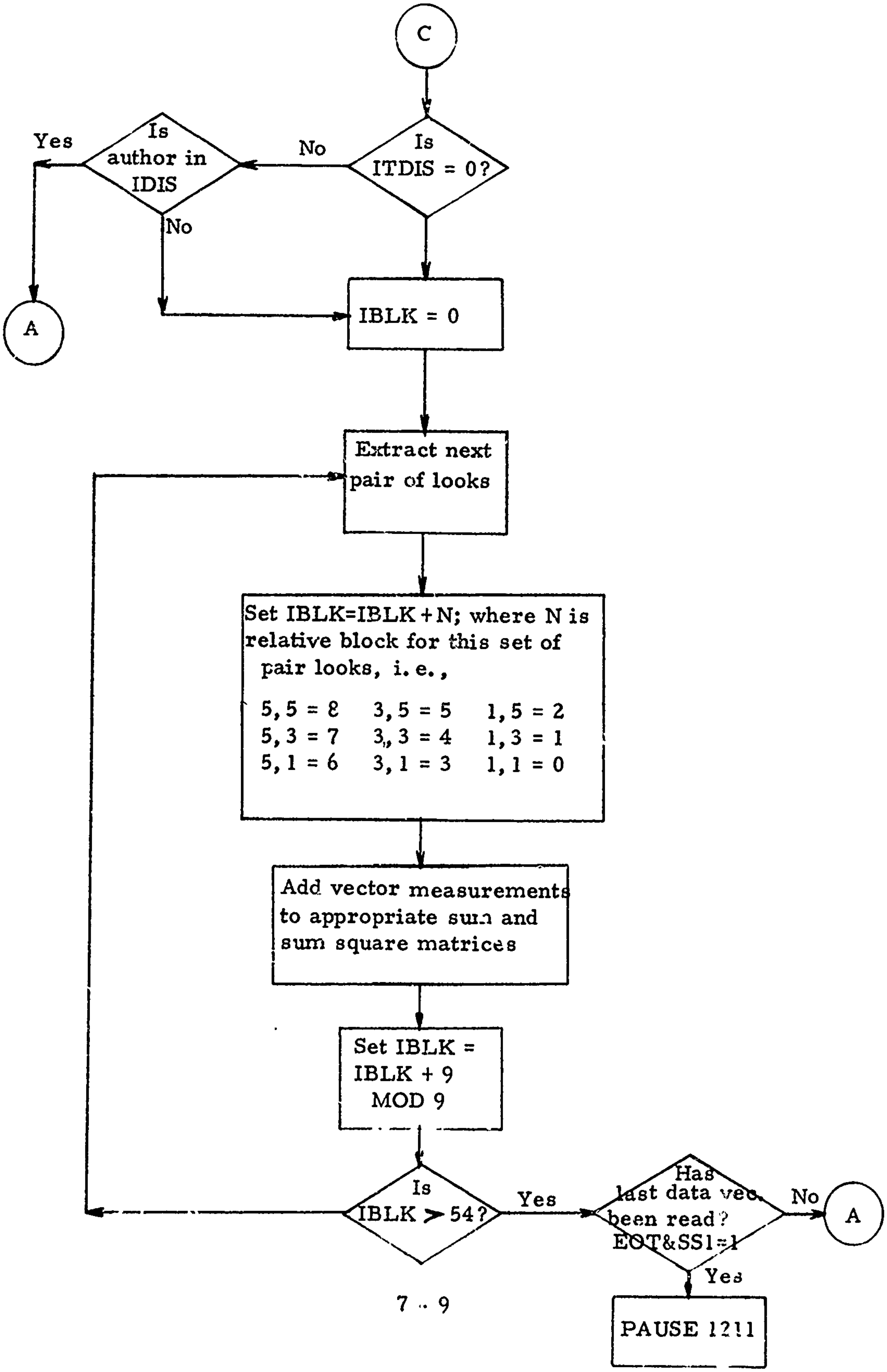
(b) The second input card signals the existence of a Vector Discard tape on physical tape drive 6 via a non-zero character in columns 1-5.

(c) The third input card (necessary only if column 2 of card 1 = 0) contains the number of data characters to be input in the entire design (Columns 1-2) and up to 48 class symbols for the data set (columns 3-50). The character punched in columns 30-50 must be listed in the order of data input.

Pause Number	Event	Continuation Action
22	End of Tape on Tape 4	Restart
23	Parity on Tape 4	Restart
32	End of Tape on Tape 3	Restart
33	Parity on Tape 3	Restart
77	Parity error on data input tape	Continue; vector will be skipped
600	Program Entrance	Mount tapes & continue
610	Character in col. 1 of input card 1 not found in class list	Restart; correct appropriate card
704	Character on input tape	


START
PAUSE 111
read card 1;
Set INITIAL=0
ISKIPAIR=initial
char; IVDT=mode
read card 2;
Set ITDIS=# of
vectors to
discard
Is
IVDT > 0
No
Read card 3; Set
ITCHARS = # of
data; Set symbols
class list = char.
symbols; IVDT=2
Yes
Is
ITDIS = 0
No
Read tape 6; Set
ITDIS=total vectors
to discard; ICHARIX
= array w/1st index
in IDIS for. ea char.
IDIS=author IDs for
ea. vector to discard.
Yes
A
Read in a
data vector
CLASSIN=Data vector sym.
AUTHOR = Author ID
LOOKSIN= 4 look sort class
B


B
Is CLASSIN = ISKIPAIR & INITIAL=0
Yes
Initialize and copy TAPE3 to TAPE4 for all classes prior to ISKIPAIR in CLASSLIST
No
Is INITIAL = CLASSIN
Yes
C
No
Write TAPE4 from drum
Is SS2 = 1 ?
Yes
PAUSE 1211
No
Is IVDT = 2 ?
No
Input next class to drum
Yes
Clear drum
C
C


C
Is ITDIS = 0?
No
Is author in IDIS
Yes
No
A
IBLK = 0
Extract next pair of looks
Set IBLK=IBLK+N; where N is relative block for this set of pair looks, i. e.,
5, 5 = 8   3, 5 = 5   1, 5 = 2
5, 3 = 7   3, 3 = 4   1, 3 = 1
5, 1 = 6   3, 1 = 3   1, 1 = 0
Add vector measurements to appropriate sum and sum square matrices
Set IBLK = IBLK + 9 MOD 9
Is IBLK > 54?
Yes
Has last data vec. been read? EOT&SS1=1
No
A
Yes
PAUSE 1211

704	End of Data Input Tape	a. Mount another tape and continue b. Set sense switch 1 and continue; P1 will complete output
1211	Final Output Complete	None

7.2. PHASONEA

The PHASONEA (P1A) package computes the logic for all pairs of classes in the design character set. Logic is designed for one look pair (TR, TB, TL, RB, RL, or BL) for each pair of classes (36 classes results in 630 class pairs) for each of the 9 sort classes. Therefore, the decision logic for 36 classes contains 5670 (9 x 630) decision points. The input to P1A consists of the set of up to 3 S/SSQ output tapes from P1 and a deck of cards consisting of one card for each class pair with the pair looks to be used for that class pair. The output is a logic tape containing one record for each class pair and a printout of the logic. P1A may be started at any designated class pair and terminated after any desired class pair. A set of utility programs for merging a series of logic tapes generated by P1A is provided in the utility package (a complete logic tape for 36 classes will consist of 630 records). The logic records are ordered in the following manner for n classes:

Logic Tape Record Number	Class Pair
$1 \longrightarrow n-1$	$1.2 \longrightarrow 1.n$
$n \longrightarrow 2n-3$	$2.3 \longrightarrow 2.n$
$2n-2 \longrightarrow 3n-5$	$3.4 \longrightarrow 3.n$
$\vdots$	
$(n-2)(n-1)/2 + (n-2)$	
$\longrightarrow (n-2(n-1)/2 + (n-1)$	$(n-2)(n-1) \longrightarrow (n-2)(n)$
$n.(n-1)/2$	$(n-1)(n)$

Therefore, the logic records produced for the 36 alpha-numeric characters begin with the character 0 vs. character 1 logic, then 0/2, through 0/Z, followed by 1/2 through 2/Z, etc. The individual logic records are formatted as follows:

Word

1 Number of words within the physical record

2 5, 5 sort class logic type code
1 = Decide class 1 - There were 2 or more class one vectors for this pair and sort class, but 1 or less class two vectors.
2 = Decide class 2 - There were 2 or more class two vectors for this pair and sort class, but 1 or less class one vectors.
3 = No Vote - There were 1 or less vectors in both classes for this pair and sort class
NDIM = number of weights for a Fisher Record of 1 discriminant line.
N(NDIM) - where N is the number of discriminant lines. (If this field is negative it implies that the Fisher record has been reversed; i. e., the B/A logic was constructed by reversing the A/B logic).

3 5, 3 sort class logic type code (see (5, 5))

.
.
.

10 1, 1 sort class logic type code (see (5, 5))

11 - last Fisher and discriminant logic for all sort classes for this pair. A Fisher record will consist of NDIM weights followed by a threshold. One discriminant line will consist of NDIM weights followed by a threshold. For two or more discriminant lines for a sort class, there will be N(NDIM) weights followed by N thresholds, where N is the number of discriminant lines.

Card Input Format

There are two input cards containing parameters to this program. These cards will be read at the beginning of each run. The format of these cards is as follows:

Card One		
Columns	Format	Description
1 - 4	I4	The total number of PROGRAM ONE output tapes (maximum 10) (IPO)
5 - 8	I4	Total number of special OLPARS discriminant tapes (maximum 10)(IOD)
11 - 12	I4	Total number of characters (currently 36)
13 - 16	I4	Number of sort classes per pair (currently 9)

Run Time Instructions for PHASONEA

(1) Compile and execute the P1A package.

(2) The program will halt at Pause 1111 (1111_8 in the A register on the 1604 console) to allow the mounting of tapes.

(a) Mount the S/SSQ Tape #1 on physical tape 3

(b) Mount S/SSQ #2 on physical tape 5

(c) Mount S/SSQ #3 on physical tape 4

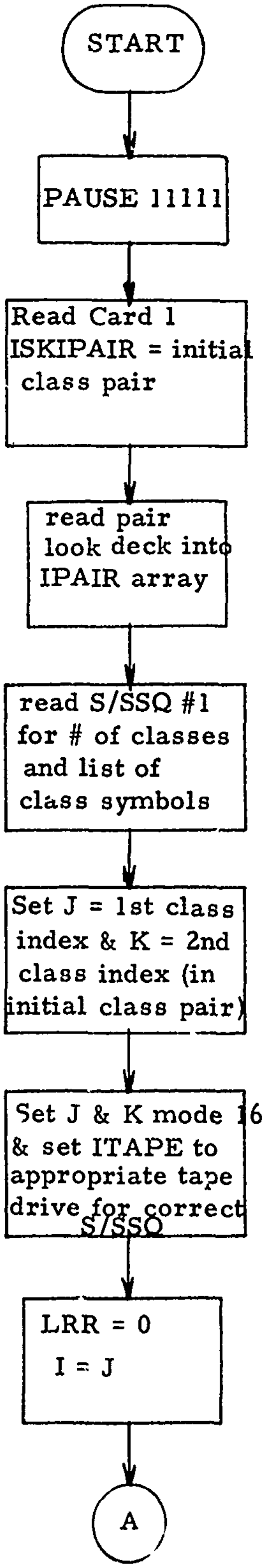
(d) Mount a scratch tape on tape drive 6 to be used as the output logic tape.

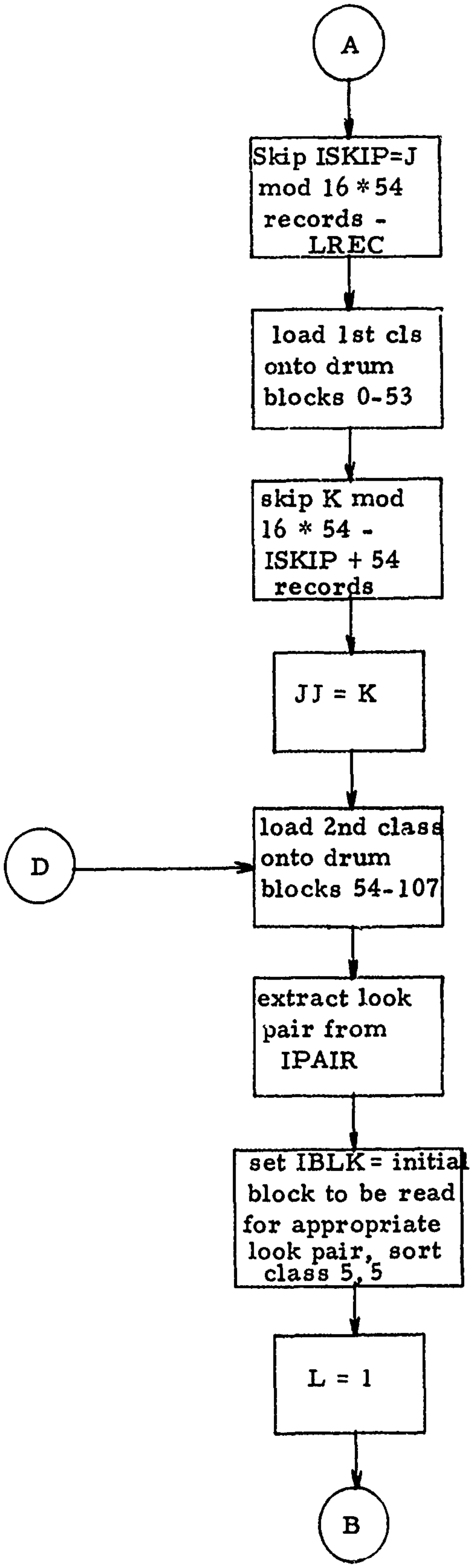
(3) Two sense switch settings are available to allow user control over program wrap-up and printout. Setting SS1 will suppress the output of the logic record printout. Setting SS2 will cause P1A to complete processing following the next logic record output.

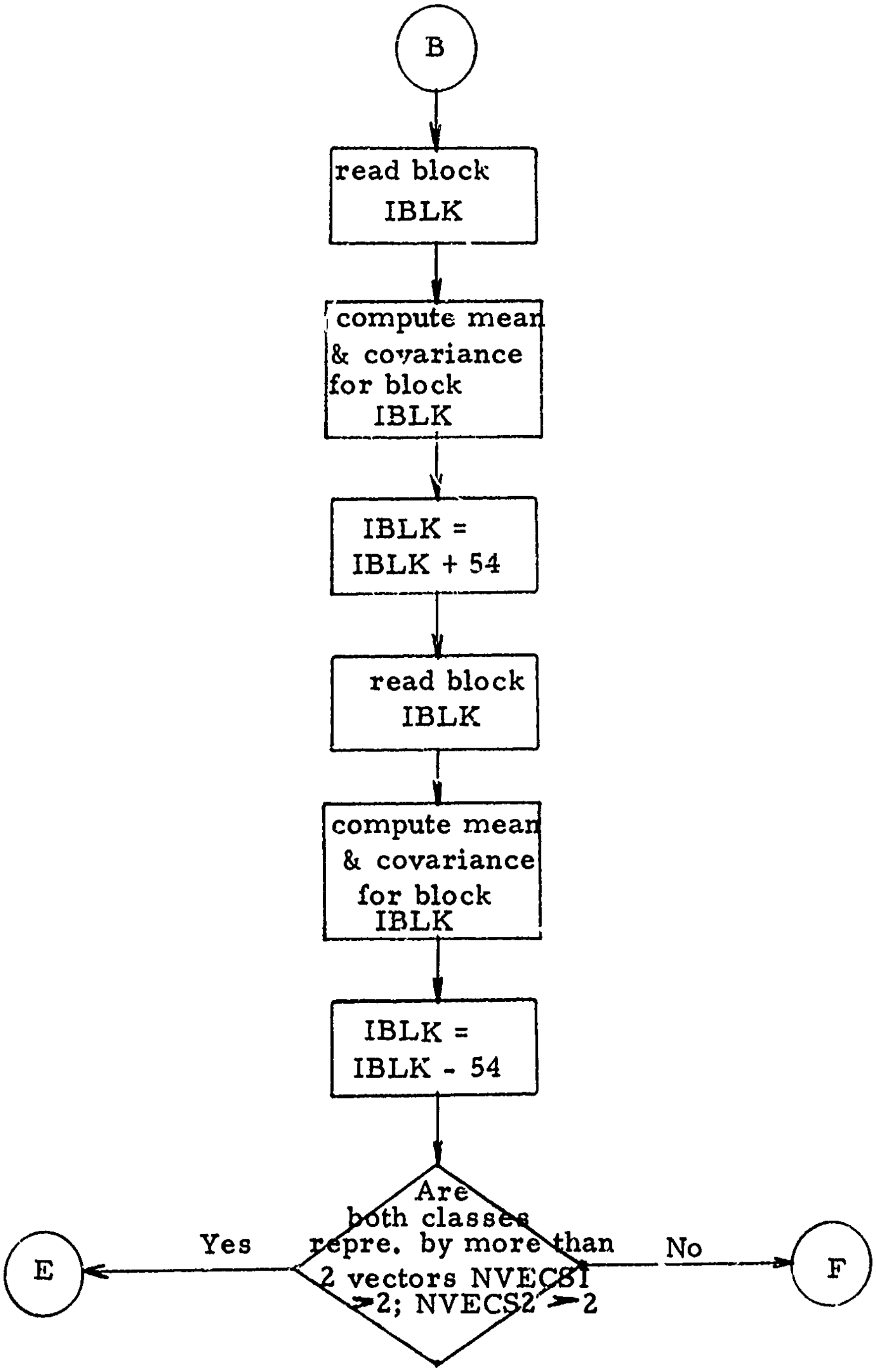
(4) One data card is required in addition to the pair look deck (which follows the data card) for the operation of P1A.

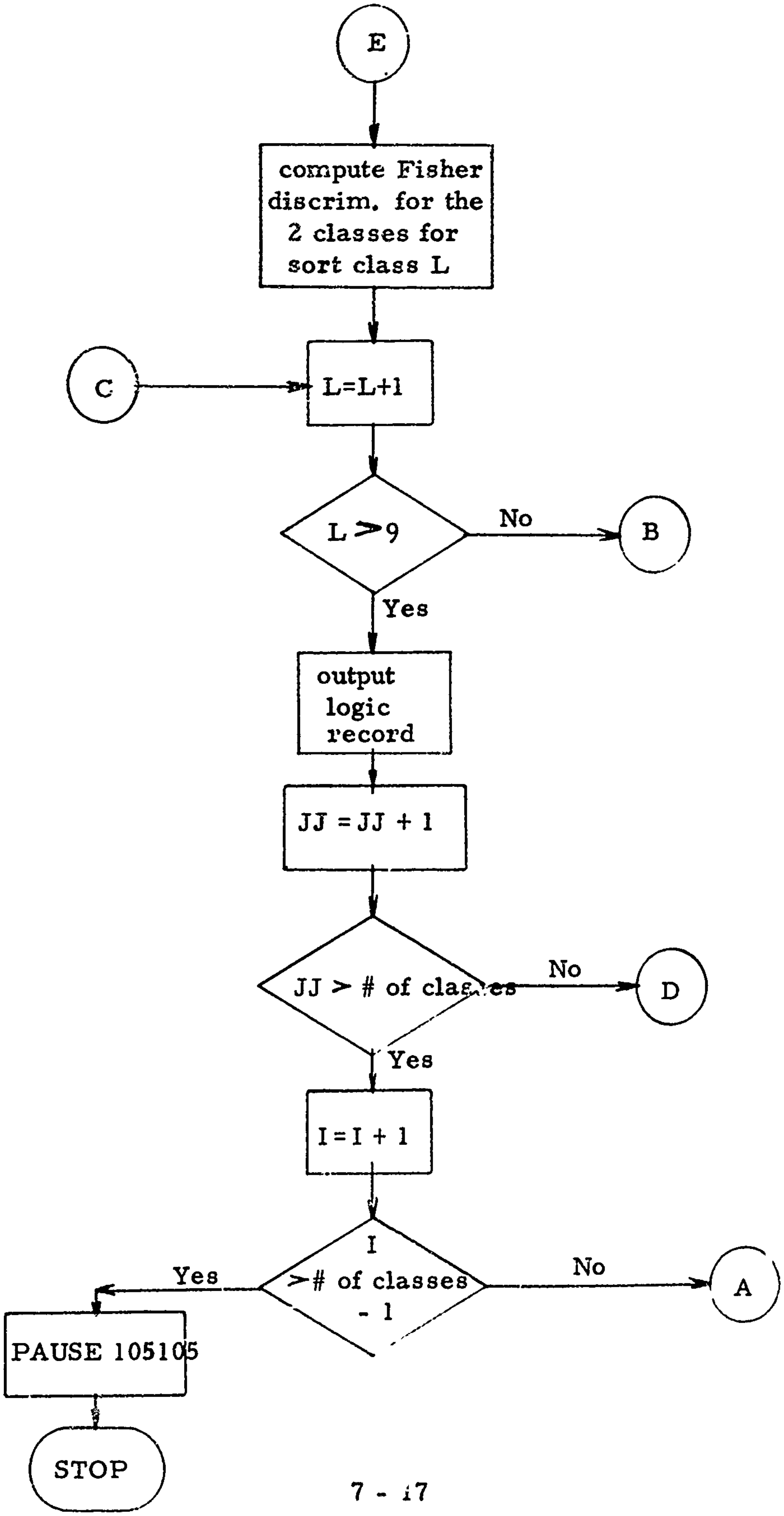
(a) The start card contains the initial pair of classes for which logic is to be designed on the current run. The higher indexed class for the alpha-numerics, in the array 0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z, is punched in column 1 and the lower indexed class in column 2.

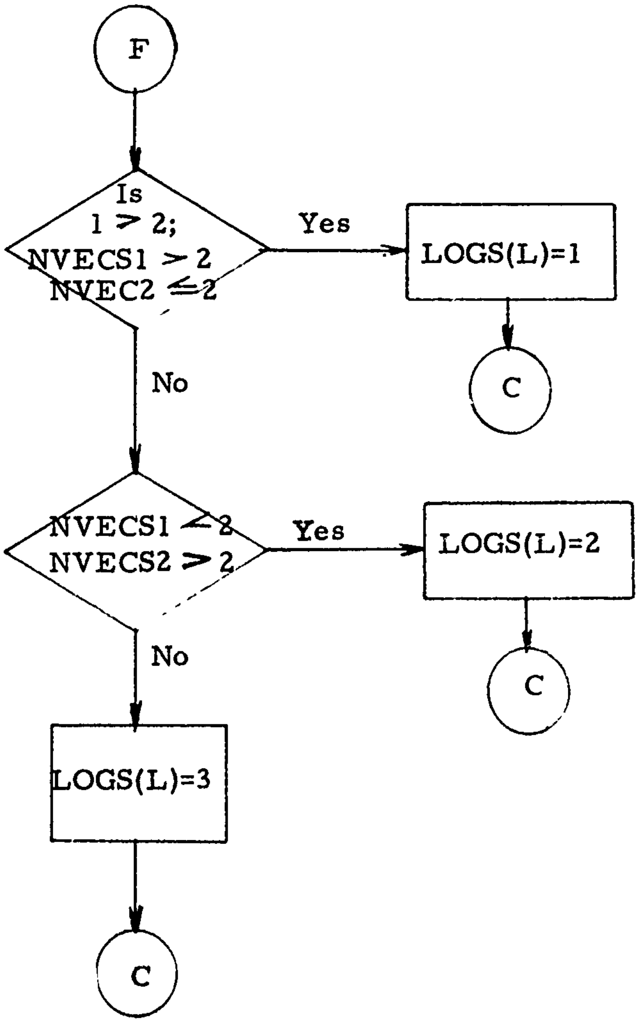
Pause Number	Event	Continuation Action
104	End of tape on S/SSQ tape during record skipping	Hardware malfunction
105	Parity on S/SSQ tape during record skipping	Continue
114	End of tape on S/SSQ tape during read of first class	Hardware malfunction
115	Parity on S/SSQ tape during read of first class	Continue or restart
214	End of tape on S/SSQ tape during read	Hardware malfunction
215	Parity on S/SSQ tape during read	Continue or restart
504	End of tape on S/SSQ tape during record skipping	Hardware malfunction
505	Parity on S/SSQ tape during record skipping	Continue
11111	Program Entrance	Mount tapes and continue
105105	Program Completion	None


START
PAUSE 11111
Read Card 1
ISKIPAIR = initial
class pair
read pair
look deck into
IPAIR array
read S/SSQ #1
for # of classes
and list of
class symbols
Set J = 1st class
index & K = 2nd
class index (in
initial class pair)
Set J & K mode 16
& set ITAPE to
appropriate tape
drive for correct
S/SSQ
LRR = 0
I = J
A


A
Skip ISKIP=J mod 16 * 54 records - LREC
load 1st cls onto drum blocks 0-53
skip K mod 16 * 54 - ISKIP + 54 records
JJ = K
D
load 2nd class onto drum blocks 54-107
extract look pair from IPAIR
set IBLK = initial block to be read for appropriate look pair, sort class 5, 5
L = 1
B


B
read block IBLK
compute mean & covariance for block IBLK
IBLK = IBLK + 54
read block IBLK
compute mean & covariance for block IBLK
IBLK = IBLK - 54
Are both classes repre. by more than 2 vectors NVECS1 >2; NVECS2 >2
Yes
E
No
F


E
compute Fisher discrim. for the 2 classes for sort class L
C
L=L+1
L > 9
No
B
Yes
output logic record
JJ = JJ + 1
JJ > # of classes
No
D
Yes
I = I + 1
I > # of classes - 1
No
A
Yes
PAUSE 105105
STOP


F
Is
1 > 2;
NVECS1 > 2
NVEC2 ≤ 2
Yes
LOGS(L)=1
C
No
NVECS1 ≤ 2
NVECS2 > 2
Yes
LOGS(L)=2
C
No
LOGS(L)=3
C

7.3. TWO

The TWO package evaluates the vectors on a Standard Input Tape (SIT) against logic generated by the P1A package.

The inputs to the TWO logic evaluation consist of the following:

- a complete block of logic for each designation class
- the table of pairwise "looks" which indicates which measurements are to be used in evaluating a vector under all possible pairwise class evaluations.

The output of PROGRAM TWO logic evaluation consists of the following:

- for each logic error made (that is, a pairwise evaluation when one class of the pair is the true class designation) the following is printed under headings on the right side of the printer paper: the Vector Index, the sort class Record Type, the Number of logical errors accumulated for the current vector index value, and the Class to which the vote is assigned.
- for each vector evaluation error made (that is, a final classification made to an incorrect class) the following is printed under heading on the left side of the printer paper: the Vector Index, the correct class evaluation, the votes tallied for and against the correct class, the final (incorrect) class evaluation, and the votes tallied for and against the incorrect class.
- for each evaluation tie made (that is, the logic evaluation resulted in a tie between the correct class and one or more others) the following is printed under headings on the left side of the paper: the Vector Index, the true class symbol, the votes tallied for and against the correct class, and the other classes involved in the tie.
- for each complete class of vectors evaluated a recap of the logic results is printed which contains the following: the class symbol designation, the total number of vectors evaluated under that designation, the votes required for a perfect evaluation (that is, no logic errors made for an individual vector), and the number of correctly classified vectors which accumulated each vote count.

The Vote Assignment Procedure

The vote assignment procedure is described below:

- o The vector is first evaluated using its "true" class logic against all other class possibilities. After a logic run is completed the vote tally, and, if necessary, the logic select procedures are operated.

- o For each pairwise evaluation which has not been accomplished previously, a pair index is computed (total pairs - (number of classes - lower class index) . (number of classes - lower class index + 1) / 2 + higher class index - lower class index.

- o The appropriate pair "looks" are selected for the computed pair and the sort class values for those looks are extracted from the last word of the vector. The number of measurements to be used is computed at this time ((sort class 1 value + sort class @ value) * 4 + 2 + extra measurements). The logic record type is determined.

- o If a Fisher or Discriminant evaluation is called for, the number of lines is computed (number of lines = |record type| / number of measurements to be evaluated), and the location of the "weights" and the "thresholds" are found within the logic record. The proper measurements are then extracted from the vector and the standard evaluation is performed, a "win" vote and a "loss" vote is assigned, and the evaluation continues (after a logic error printout if appropriate)

- o If an arbitrary vote record type is assigned the "win" and "loss" votes are assigned (a "no decision" record causes "loss" votes to be assigned to both pair classes) and the logic evaluation continues.

The Vote Tally procedure is described below:

- o The Vote Tally procedure operates at the conclusion of each logic run (that is, the in-core logic has been evaluated for each pairwise combination for the current vector).

- A "temporary winner" is found by selecting a class which has the most "win" votes and has completed its vote tabulation (the sum of "win" votes and "loss" votes for that class equals the number of classes minus one).

- A "final winner" is determined when the "temporary winner" is found to have the fewest "loss" votes among all the classes including those for whom vote tabulation is incomplete. When a "final winner" has been determined it is checked against the true class, the appropriate printout generated (if necessary) and the next vector called for evaluation.

- If there is any class which has fewer "loss" votes than the temporary winner, the Logic Select procedure is operated.

- If there are one or more classes which have the same number of "loss" votes as the temporary winner, then each class in that category is checked to determine if the vote tabulation is complete for that class. If there are any of these classes which fail this check, the Logic Select procedure is placed into operation, otherwise the tie vote printout is generated and the next vector called for evaluation.

The Logic Select procedure is described below:

- The Logic Select procedure describes which class logic is to be evaluated when a "final winner" cannot be determined at the end of a logic run.

- The class logic selected is determined by finding the class with the fewest losses which has not had a complete vote tabulation. If there are more than one of these, the one with the highest number of "win" votes is selected.

Run Time Instructions

(1) Compile and execute the TWO package.

(2) The program will come to Pause 111 (111_8 in the A register on the 1604 console).

(a) Mount the SIT on physical tape 5

(b) Mount the design logic tape on physical tape drive 3.

(3) Four sense switch settings are available to allow user control over program wrap-up and printout. Setting SS1 will cause TWO to output final class statistics and complete operations immediately. SS3 set will cause the system to generate a primary audit trail for each logic evaluation event. If the user sets SS4, each incorrectly evaluated vector will be printed; and SS6 will cause the generation of a detailed audit trail for each input vector.

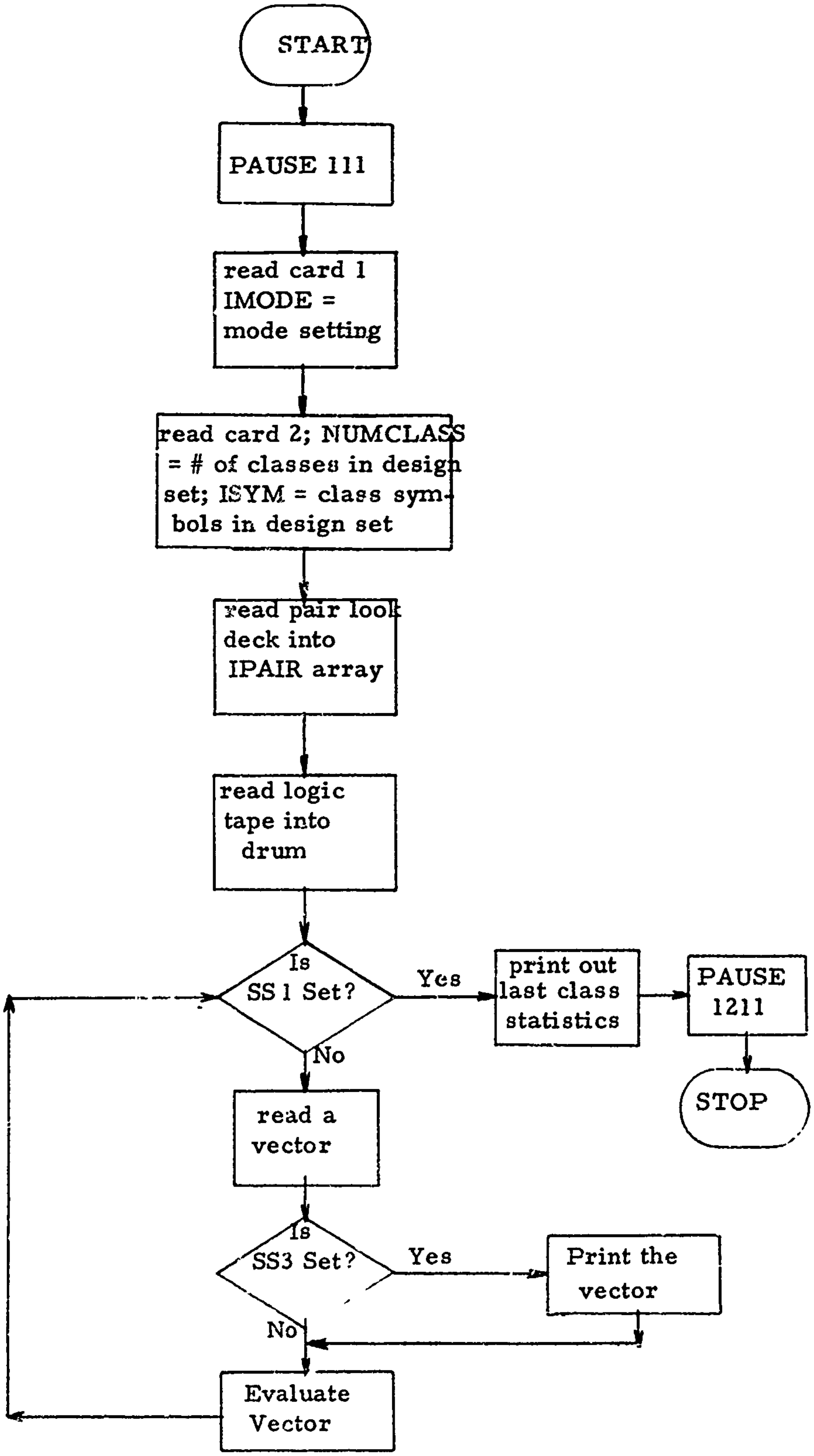
(4) Two data cards are required for the execution of TWO:

(a) Card 1 contains an integer value in column 1 representing the operating mode of TWO. When the mode is zero (0), all vectors on the input tape will be evaluated. Currently, a mode value of one (1) evaluation of a single class will be limited to 101 consecutive vectors, after which the SIT vectors will be skipped until a new class vector is input. A mode value of two (2) will cause a printout in the format "a/b nnnnn" for each input character where a is the final character evaluation, b is the symbol attached to the input character, and nnnnn is the author identification value for that character. *

(b) Card 2 for program TWO is the same as card 3 for P1; that is, the number of data characters in the logic design in columns 1-2 and the class symbols for the data set in columns 3-50 listed in the order of original data input.

Pause Number	Event	Continuation Action
1	End of file on logic input tape	Hardware malfunction
2	Parity error on logic input tape	Continue or restart
77	Parity error on data input tape	Continue
111	Program Entrance	Mount tapes and continue
710	End of file on data input tape	a. Set sense switch and continue b. Mount another input tape and continue
1211	Program Completion	None

* - Not implemented


START
PAUSE 111
read card 1
IMODE =
mode setting
read card 2; NUMCLASS
= # of classes in design
set; ISYM = class sym-
bols in design set
read pair look
deck into
IPAIR array
read logic
tape into
drum
Is
SS 1 Set?
Yes
print out
last class
statistics
PAUSE
1211
STOP
No
read a
vector
Is
SS3 Set?
Yes
Print the
vector
No
Evaluate
Vector

7.4. UTILITY PROGRAMS

7.4.1. MAKE

Program MAKE creates a discard vector tape on physical tape 6 for input into Pl from a punched card deck. The input deck consists of card 3 from Pl plus a card for each vector to be discarded with the symbol for the vector in column 1 and the author identification value in columns 2-6. The cards must be grouped by class and ordered (by class) in the same manner as the design data input. The output tape receives the total number of discarded vectors, a table of indices into the first vector to be discarded from a given class, and a list of up to 3000 author identification values to be discarded.

7.4.2. S/SSQ TAPE MAKE UTILITY and LOGIC TAPE COMBINE UTILITY

These utilities are a set of tape handling routines which copy and/or skip non-formatted records from one tape to another. Subroutine SKIP(I, J) skips I records on logical tape J (where J = 1 for physical tape 2; 10 for physical tape 5; or 20 for physical tape 6). Subroutine COPI(I, J) copies I records from tape J to a tape mounted on physical tape drive 6. Subroutine CLEAR(I, J) writes 54 null records for each class symbol in the design set from class index I to class index J. Subroutine WRTHEAD writes an S/SSQ header record on tape 6. Subroutine COPY(I, J) copies 54 records for each of I classes from logical tape J.

SECTION 8

CONCLUSIONS AND RECOMMENDATIONS

The importance of careful data collection and editing to the generation of good logic should not be underestimated. Due to mislabeling, noise, and drop-outs caused by ink-camera incompatibility, many characters were unsuitable for inclusion in the design set data base. The system for correcting these faults (the Data Base Editing Package) by either re-labeling, noise elimination, or deletion, was of great importance in arriving at a usable data base.

The alpha-numeric recognition system with reject strategy A achieved recognition rates comparable to humans. Most substitutions occurred when the substituted character looked like a character in the decision class due to breaks in the characters or to similarity of the shape of the character with the decision class. Some errors, however, were due to insufficient sample size in the design class in the particular sort class to which the character belonged. We recommend strengthening the logic by increasing the number of samples of a given character class in those sort classes where it is needed.

The existence of confusion pairs such as Z and 2, S and 5, 0 and O, makes the totally unconstrained alpha-numeric character set an impossible set upon which to achieve practical recognition rates. We recommend the adoption of some constraints, e.g., slashing zeros and Z's, and insisting that the upper right horizontal bar on the 5 not be curved down in confusion with an S - so that the confusion between certain character pairs is removed. The ANSI set of guidelines for handprinting may be used as a fairly rigid set of constraints or a less stringent set of constraints which attempts only to differentiate characters in the confusion pairs may be designed.

The amount of constraint depends upon the specific application chosen. If a controlled group of people generates the handprinting, more constraints can be placed on the printing since training the people to follow the constraints would be possible. If the input to the recognition system comes from an uncontrolled environment, such as the general public, constraints must be kept to a minimum.

The amount of constraint will dictate the parameters of the machine. If mild constraints are chosen, then constraints must be effectively "built in" to the recognition logic, resulting in a "tight" machine which rejects a character unless it satisfies the built-in constraints. If rigid constraints are chosen and followed, then the machine can be of the "open" variety, where more variability in character shapes are permitted.

The technique has been tested and proven to provide a capability comparable to human recognition. As in any development program, it is difficult to proceed beyond a certain point without a specific application. Furthermore, refinement of the technique will only be accomplished via a specific problem application.

We recommend that a specific application be chosen and appropriate constraints be adopted so that a solution tailored to the character set with the particular constraints may be achieved.

REFERENCES

1. S.S. Wilks, Mathematical Statistics, New York, Wiley, 1962, pp 573-581.

2. Sammon, J.W., "An Optimal Discriminant Plane", IEEE Transactions on Computers, Sept. 1970.

3. Sammon, J.W., Sanders, J.H., Dimeo, M.P., et.al., "Hand-Printed Character Recognition Techniques," Final Report to Contract F30602-69-C-0374, RADC-TR-70-206, October 1970. AD876 875