Parallel Algorithms
for
Handwritten Character Recognition

Melad Y. Ghabrial

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

May 1990

ISBN 0-315-97702-7

ABSTRACT

Parallel Algorithms
for
Handwritten Character Recognition

Melad Y. Ghabrial

A set of parallel algorithms is presented for (1) extracting shape features from horizontally, vertically and diagonally scanned handwritten characters, (2) sorting these features and (3) classifying the scanned characters. A special parallel architecture to implement these algorithms is designed. The reliability of the algorithms/architecture is established through simulation and experiments, and the scheme is shown to be tolerant to writer style variation, rotation and distortion. The architecture is cost effective and amenable to VLSI implementation.

Acknowledgments

Thanks to God. Thanks to my father and mother for their belief and support. Thanks to my wife and son for their patience and bearings during the good and bad times we went through in a new life, country and weather. I would also like to thank Dr. H. F. Li, my supervisor and professor at Concordia for his confidence in me, his invaluable assistance, ideas, review and patience during the whole period of my thesis. I would also like to thank Dr. R. Jayakumar, my professor at Concordia for his excellent efforts and the time he spent reviewing my thesis till it became what it is today.

Table of Contents

Table of Contents (cont.)

List of Figures

List of Figures (cont.)

CHAPTER I

Parallel Algorithms for Handwritten Character Recognition

1.1 Introduction

Pattern recognition algorithms are usually designed for a sequential computer which processes information in a serial manner. Although most of the well known pattern recognition algorithms can be transformed into parallel ones, as mentioned in [5,13], until now none of the above mentioned references has solved the complete problem starting at feature extraction and ending at classification. This is due to the fact that most of the designs suggested in the literature deal with a subtopic in pattern recognition such as correlation, digital transforms, thinning, or binarization. Because of the variety of applications, each with special characteristics and constraints, it is hard to apply a general architecture to most of these applications. On the other hand, if we categorize those applications into specific domains as handprinted character recognition, handwritten character recognition and printed character recognition we might reach a useful and effective architecture for each category.

The purpose of this thesis is to design and simulate an architecture for Handwritten Character Recognition (HCR), specifically handwritten numerals, using cellular or systolic arrays. This architecture has to be easily amenable to implementation in current VLSI technologies. The real aim is to build an integrated, reliable, cost effective, and very fast system that will extract specific features from the input patterns and classify them into their respective classes.

1.2 Work organization

Chapter I includes a discussion on different algorithms used previously in HCR problem, and a comparison between those algorithms leading to the selection of the best suited ones that:

i) are tolerant to pattern distortion, rotation, translation and variation of writer style, and

ii) can be easily implemented using systolic arrays.

Chapter II, (HCR Applied Method), presents the overall problem abstraction, and an explanation of the theory and algorithms used for features extraction. Chapter III, (Features Extraction Algorithms), is on the design of the first phase in the system, namely features extraction and storing phase. Chapter IV, (Features Gathering and Sorting), presents the design of the second phase: chain gathering and sorting stages.

Chapter V, (Classification Phase), concentrates on the design of the classification phase, describes the simulation experiment conducted and analyzes the results extracted from the 1200 test samples. Chapter VI, (System Performance), contains the system performance evaluation, conclusions and future work to be explored. Appendix A contains a list of references. Appendix B contains some samples from the 1200 digit and some samples of the rotated digits. Appendix C contains the tables constructed during the simulation experiment which are used during the classification phase.

1.3 Comparison between different HCR methods

This thesis does not claim to introduce any new approach for HCR. Instead it emphasizes on parallelization and amelioration of some previous attempts, and on the design of an architecture suitable for VLSI implementation. In this section we include a concise review of the different techniques used in HCR in order to appreciate the choice of an approach that fulfill the following criteria:

i) Tolerant to pattern distortion and deformation.

ii) Suitable and practical for VLSI implementation.

Various HCR algorithms available in the literature can be divided into two categories based on the type of features extracted from the patterns. These two categories are:

i) Global and statistical features.

ii) Geometrical and structural features.

1.3.1 Global feature extraction algorithms

This class of algorithms involve (i) matching an input pattern with stored templates using correlation [9], or (ii) calculating statistical parameters such as the density of black pixels within certain regions [7] and the number of times a certain directed vector crosses from a white region to a black one [4,17].

Although these techniques have abundant parallelism and can be easily implemented on VLSI chips, they suffer from their intolerance to rotation, translation, and high sensitivity to distortion and style variation which are inherent in handwritten characters. So this approach does not usually work well in HCR.

1.3.2 Geometrical and structural algorithms

The second approach is based on the extraction of features that describe the geometry or topology of the character pattern. We can identify three different methods.

i) The first method uses Fourier descriptors [18,27] and is complex. Although the features extracted are tolerant to rotation, they are very sensitive to distortion and style variation. It suffers from a serious drawback as its results are affected by tiny details that may be attributed to noise, binarization process, or writer style. These tiny details reflected in the transform space often lead to confusion, rejection, or misclassification.

ii) The second method is based on contour tracing, or thinning and skeletonization [13,31,32,33]. It is very sensitive to rotation. Often thinning algorithms introduce other problems such as broken lines or loss of features.

iii) The third method [1,2] is based on detecting the edges surrounding a pattern, and by defining a set of relations between coincident edges (that is, edges meeting at a common end point). These relations allow further shape characterization. Higher order features can be constructed by the association of two or more consecutive simple ones. This method was found to be less complicated and more promising as it is less sensitive to noise, and appears to accommodate style variation rather naturally. Previous attempts using this method [1,2] had some drawbacks in extracting the higher order features and in classification. Some improvements are suggested in this thesis in order to remedy these drawbacks and to achieve tolerance to translation and rotation (-15° to $+15^{\circ}$).

A comparative study on different algorithms for HCR [16] reported that geometrical and topological features appeared to be superior to global and statistical features because of their low sensitivity to distortion, rotation, and translation. The problem confronting the former approach is the apparently large amount of computation and storage space required. This problem can be solved by parallelizing these algorithms and directly implementing them in hardware, which form the main scope of this thesis. Our conclusion agrees exactly with the study in [16]. It is not necessary, and even undesirable, to extract exact shape features which may lead to inflexible classification. We need to extract only relevant features required for classification. The effectiveness of any such algorithm has to be judged by its tolerance to style variation and distortion.

Chapter II

HCR Applied Method

Presented in this chapter is a complete discussion of the method applied for our HCR parallel system. We will define all the parameters and shape features that are of interest to us. Included also are the enhancements we applied to the method of [1] to be able to fulfil the two criteria mentioned earlier.

2.1 Extracting Shape Features

Given a binarized and segmented pattern enclosed within a frame of size MxN, where M is the number of rows and N is the number of columns, we wish to extract the shape features surrounding the black regions in the character pattern. Some definitions are first introduced.

2.1.1 Edges

An edge is a transition from a white region to a black region or vice-versa. Two different edges are encountered in scanning across a body region which is the region containing only black pixels. Figure 2.1 shows the different body regions in a sample pattern and the edges surrounding them. Figure 2.2 shows the start, end, split and merge points of body regions. From figure 2.2 we can observe the following:

- The start of an edge, called the head, is encountered when a body region starts or splits.

- The end of an edge, called the tail, is encountered when a body region ends or when two body regions merge together.

- Each new body region encountered generates the start of two edges of different types, e^1 for the left edge and e^2 for the right one.

- Each split in a body region generates the start of two other edge types, e^3 for the left edge and e^4 for the right one.

Figure 2.3 illustrates these different edge types that can appear in tracing the boundaries of any pattern. Thus we have the following characteristics:

- An edge of type e^1 corresponds to a left transition from a white region into a body region.

- An edge of type e^2 corresponds to a right transition from a white region into a body region.

- An edge of type e^3 corresponds to a left transition from a split body region into a white region.

- An edge of type e^4 corresponds to a right transition from a split body region into a white region.

- The head of an e^3 and e^4 edge occurs at the point where the body region split into two parts.

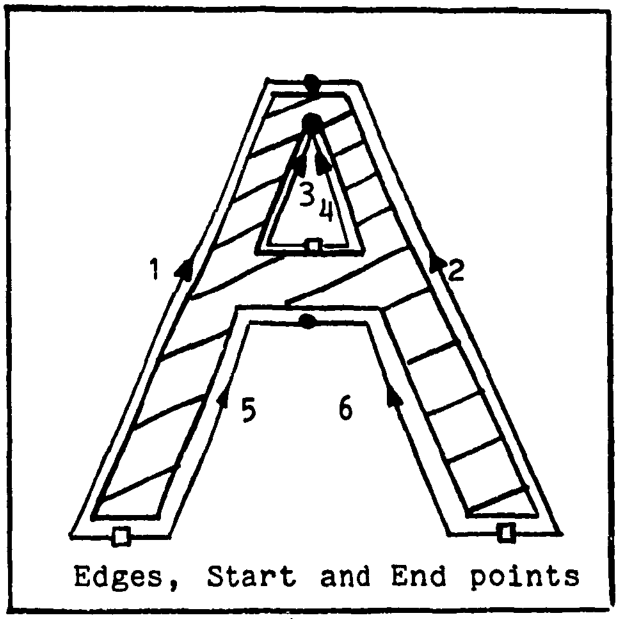


Figure 2.1

- In the figures shown on this page,
  circles represent start points
  squares represent end points

Figure 2.2

Body Regions B.R.

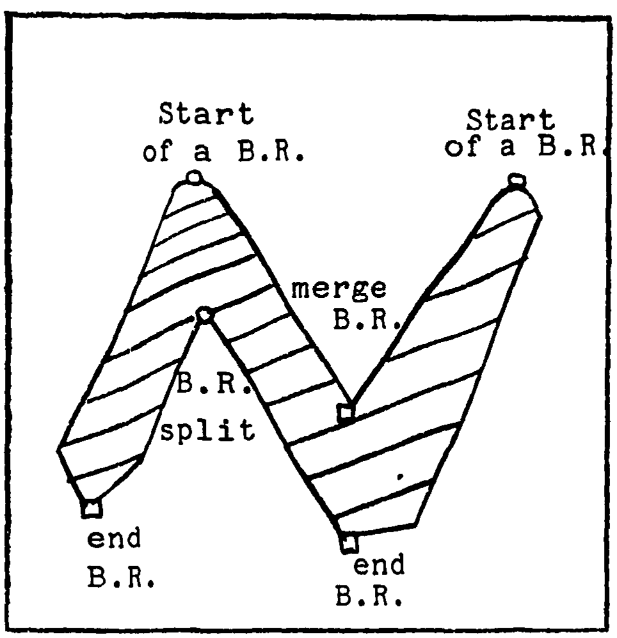
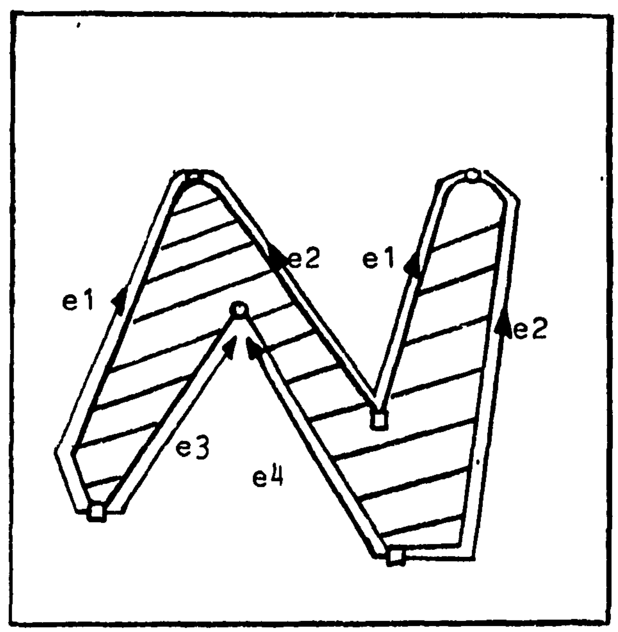


Figure 2.3

Start and End Points



2.1.2 Edge relations

An edge relation can be established between two edges which meet at their heads or at their tails. We will give the symbol (x) for the relation generated when two edges meet at their heads and the symbol (-) for that when two edges meet at their tails.

Table 2.1 reveals all possible relations that can be defined among the four edge types e^1, e^2, e^3, and e^4. In any pattern, only ten different relations, namely R_1 to R_{10} can be found. Two higher order relations are also defined. These two are the concatenation of R_1 and R_2 written as R_{12} and that of R_7 and R_8 written as R_{11}. Figure 2.4 shows example patterns for these twelve relations.

left edges \ right edges	e^1	e^2	e^3	e^4
e^1		x,-	-	
e^2	-			-
e^3	-			x,-
e^4		-	-	

Table 2.1 Allowable Relations

2.1.3 Simple shapes

The 12 relations shown in figure 2.4 construct all the primitive or simple shapes that will be used later. These shapes can be classified as follows:

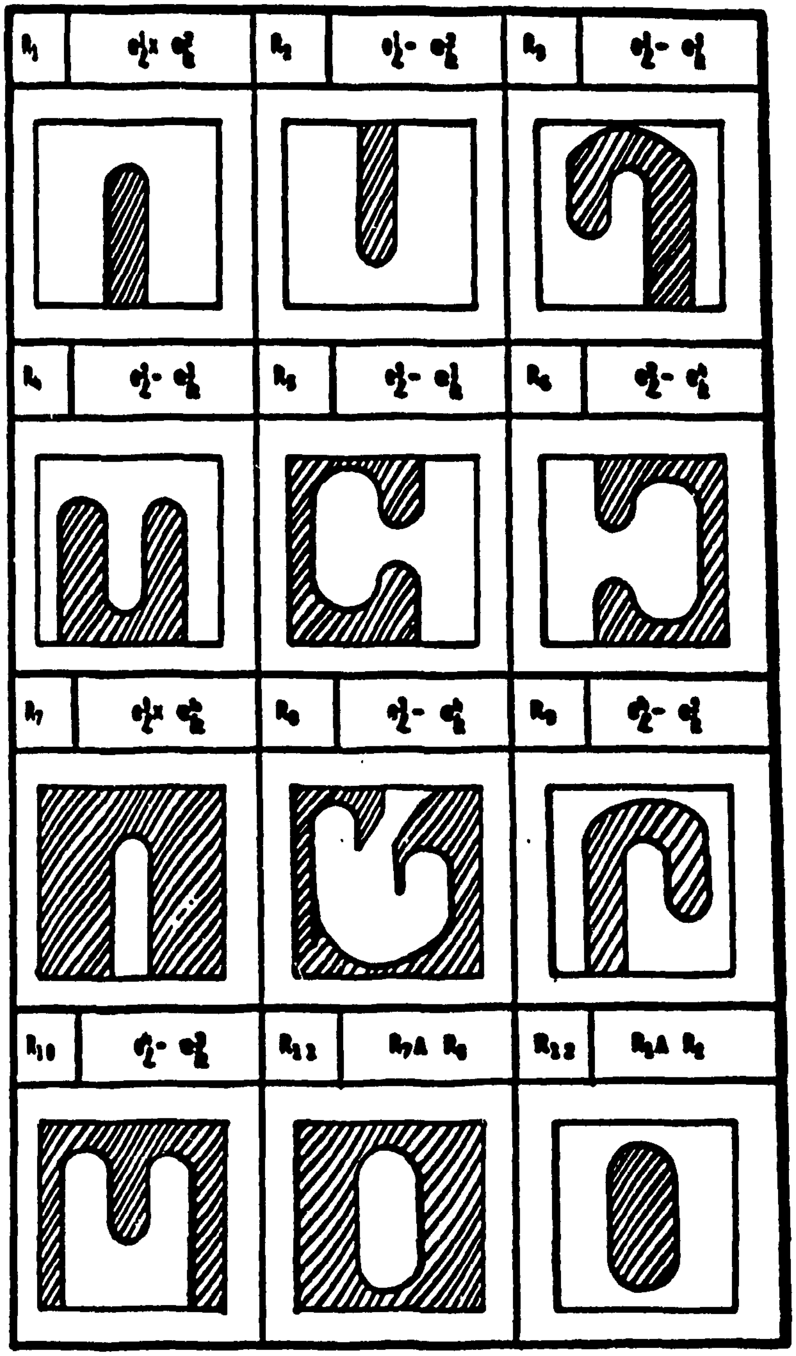


Figure 2.4
Twelve Edge Relations by [1]

i) **Beginning of a body region:** Relation $R_1 = e^1 x e^2$ implies the start of a body region.

ii) **End of a body region:** Each of relations R_2, R_3, R_9, and R_{10} implies the end of a body region.

iii) **Simple cavity (type 1):** Each of relations R_4, R_5, R_6 and R_8 implies the existence of a cavity open from the top (cavity-1). This cavity is formed by the merging of two body regions. We can classify cavities of type 1 according to other information, such as the relative rank (position) of the heads of the edges. Specifically if the head of the left edge is higher (lower) than the head of the right edge then rightward (leftward) open cavity is formed. Figure 2.5 explains this point, where i and j represents the row numbers at which the left and right edges, respectively, start.

iv) **Simple cavity (type 2):** Relation R_7 implies the existence of a cavity open from the bottom, (cavity-2). It is created by the splitting of a body region.

v) **Simple hole :** Relation R_{11} implies the existence of a hole. This hole is formed by the splitting of a body region into two regions and then those two regions merge together to enclose a hole.

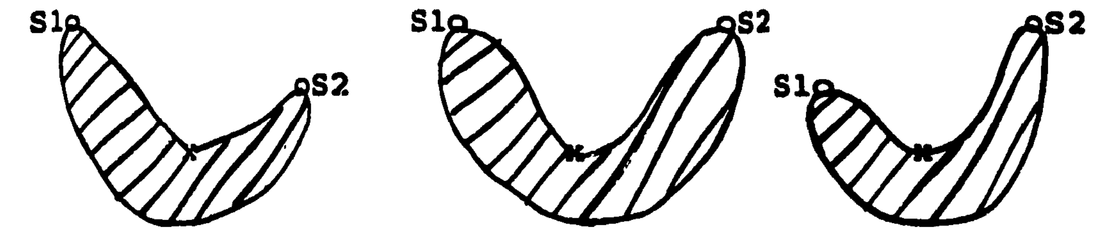
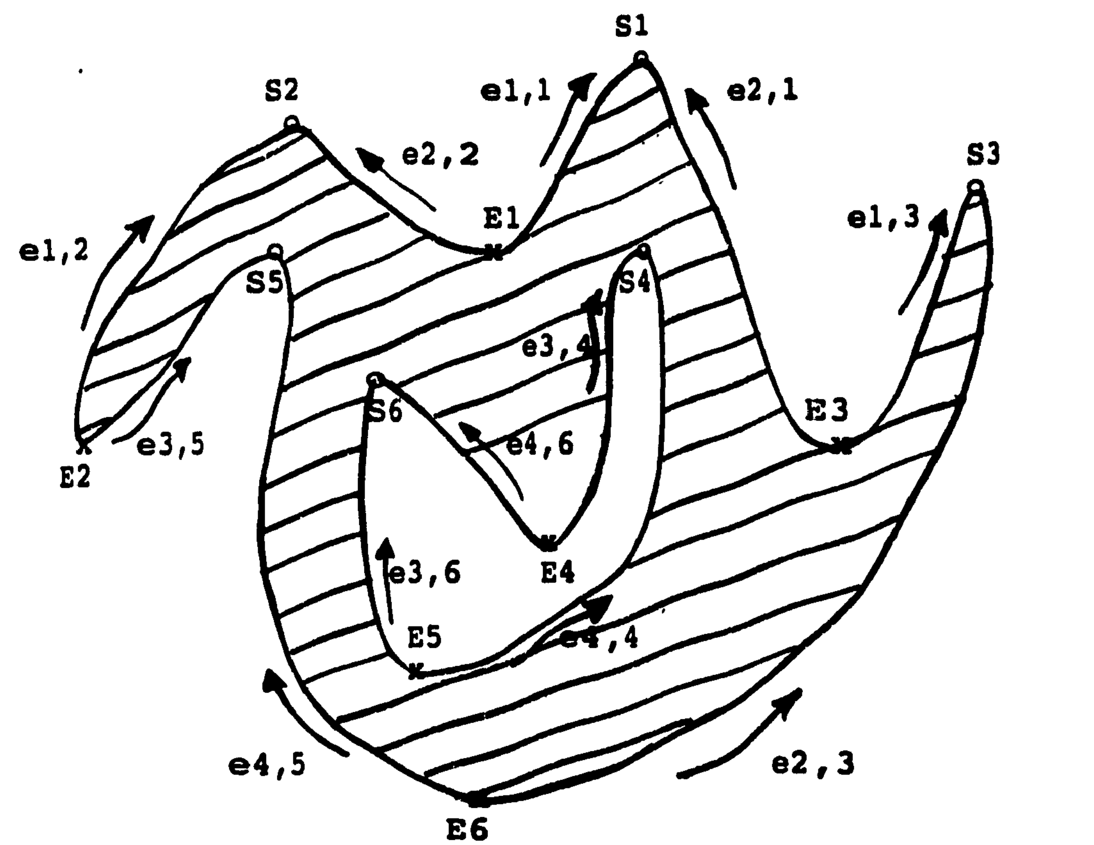


Figure 2.5
Edge Ranking



e(k,l) : where k is edge type and l is edge rank
S(l) : start point with rank l
E(l) : end point with rank l
Chain 1: S1,E3,S3,E6,S5,E2,S2,E1
Chain 2: S4,E5,S6,E4

Figure 2.6 chains

2.1.4 Chains

Each pattern might contain one or more inner chains which trace the holes. Also each character (numeral) contains only one outer chain along its outer contour. In other words, we can define a chain as sequence of edges that form a closed loop on the inner or outer contour of a pattern. The number of chains for any specific pattern has to be equal to the number of its inner and outer contours. The edges in a chain are ordered in clockwise direction starting with the highest edge on that chain. Figure 2.6 Shows a pattern containing two chains. The list of edges forming each chain is also shown. Depending on the context of usage, the chain can be equivalently represented by the sequence of start/end points traversed or the corresponding relations as depicted in figure 2.6.

2.1.5 Complex/Simple holes

In figure 2.6 there exists a hole inside the pattern. This hole is not a simple one as defined by an R_{11} relation. This type of hole is actually constructed from the concatenation of more than two edges. We can detect any hole, complex or simple, by simply looking at the first relation in an ordered chain. This first relation will correspond to the two edges emerging from the highest start point in the whole chain. If this relation is $R_7 = e^3 \times e^4$ then a hole exists. On the other hand if this relation is an $R_1 = e^1 \times e^2$ then the chain does not correspond to a hole. The proof for the above statement is included in [1]. Referring to figure 2.6, we find two chains available C_1 and C_2. Notice that chain C_1 does not correspond to a hole because the first relation is R_1 not R_7. Chain C_2 on the other hand constructs a hole.

2.2 Modifications to Ahmed and Suen's method

In their method, Ahmed and Suen [1,2] started the feature extraction algorithm by scanning the digitized character matrix from top to bottom and left to right. Upon discovering a start point, their sequential algorithm followed the character contour to reach the end point. Then the relations are extracted by comparing the edge types and the coordinates of the start/end points. These relations are then combined into circular chains and different features are extracted from each chain as holes and cavities of different types. They also extracted the coordinates of each feature by calculating the upper-right corner and bottom left-corner of the rectangle that best fits/enclose this feature. After the edges, relations, features and the coordinates of these features, are extracted, Ahmed and Suen start the classification phase by searching through a very large database, comparing each extracted (feature, coordinates) tuple with the records stored in the database. If a match is found the search continues with the rest of the extracted features. If no match is discovered, then the pattern is rejected.

This sequential processing requires a significant amount of processing time and space. The implemented system is also sensitive to rotation and translation, because exact location matching is used. On the other hand some useful information extracted has not been used to advantage. For example, during classification, each extracted feature (relation) is treated individually without paying attention to the sequence ordering. The sequence information actually is very useful as it will be shown later.

2.2.1 The novelty of the work

Three modifications to the work of [1,2] have been introduced in this thesis. First, we will not associate a location attribute to each feature point. Second, an entire chain is treated as a complex feature. As mentioned in section 2.1.5 an ordered chain is formed by the sequence of edges and their relations surrounding a shape and sorted in a clockwise direction. Third, smoothing is applied to each chain. The smoothing removes relations involving edges shorter than some threshold. This leads to better recognition results as the system can accommodate variations in writer style and more noise created in preprocessing. These modifications have resulted in better system performance as illustrated by the simulation reported in detail in chapter V.

2.3 Applied method summary

The method we have developed can be summarized as follows. First the edges of a given character pattern are extracted together with their types, relations and ranks. From the extracted edges and their joint relations, the chains can be constructed in clockwise direction. Each chain is further smoothed to accommodate variation of writer style and digitization noise resulting in a complex feature. The above steps are repeated on three versions of the same character pattern. The first version corresponds to scanning the pattern horizontally, named H-scan. The second corresponds to scanning the pattern vertically, named V-scan. The third corresponds to scanning at -135^{o}, named D-scan. All the three versions are processed in parallel. Each resulting complex chain will be compared with a pre-stored database. In each record of the database is also stored a list of all the character classes to which the associated complex chain may belong. A successful search of the database returns the list of classes for the given chain. From the lists of classes obtained from the three scans, the intersection of the three lists is obtained. A successful (unique) class is identified when only one member exists in the intersection. Else the pattern is either rejected or additional features will be needed.

CHAPTER III

Feature Extraction Algorithms

This chapter will present the parallel algorithms for extracting all the edges surrounding a given pattern, start and end points for each edge and edge relations. A problem abstraction follows this introduction. The algorithms for edge extraction and storing stages are then discussed.

12	11	10	09	08	07	06	05	04	03	02	01	
				X	X	X	X	X				1
		X	X	X	X	X	X	X	X			2
		X	X	X			X	X	X	X		3
		X	X					X	X	X		4
		X	X					X	X			5
			X	X	X	X	X	X				6
		X	X						X			7
	X	X							X	X		8
	X	X							X	X		9
		X						X	X			10
			X	X	X	X	X	X				11

X represents one black pixel
Figure 3.1
Matrix of a Digitized Pattern

3.1 The HCR Algorithm

Starting with an input character of size MxN as illustrated in figure 3.1 above, the steps that should be followed in extracting shape features are:

a) Extract all edges that surround the character's inner and outer contours. Each edge is identified by its start and end point coordinates. An end point occurs whenever two edges meet at their tails, whereas at a start point two edges emanate (one tracing left and the other tracing right).

b) Build the chains of these feature points by ordering them in a clockwise sequence, starting each chain with the start point that has the highest rank (row number in case of horizontal scan). Some smoothing is applied to merge short edges in order to remove small features and ripples.

c) Process each chain containing a hole to derive the smallest rectangle that encloses the hole. The chain is then replaced with a hole feature and a rectangle enclosing it.

d) Merge all the remaining chains and hole features obtained in (c) together to form one complex chain, called feature string. This string when matched against the records stored in a classification database identifies a set of classes that may contain the feature string.

e) Repeat steps (a-d) on three separate and rotated scans of the character image as discussed in chapter II. These three scans are processed in parallel, each returning a set of classes to which the image may correspond.

f) Take the intersection of the three output sets to produce the final classification result. If the intersection contains more than one choice, a rejection is necessary.

The above steps will be executed repeatedly for a continuous stream of input characters using a suitable systolic architecture. Each stage discussed above will be constructed of a finite number of Processing Elements called PEs. Each PE has a constant number of storage registers and all PEs in each stage execute the same instruction each cycle. In each stage the PEs will be connected linearly. A PE will be able to communicate only with its predecessor and successor in the linear array. The reason this architecture is called systolic is due to the fact that data flow into and out of the column of PEs in a rhythmic way. During each cycle a new data/pixel enters each PE, get processed and the result is delivered at the end of the cycle. This scenario is repeated for as long as there is input data flow. During each clock cycle the PE process an input data and delivers an output data the same as a heart pumping blood in and out during each beat.

If the system is formed of a number of stages, as will be shown later, then each stage has to finish processing a character image before the following image enters it. Since one complete column of a picture matrix enters the system each cycle, the maximum number of cycles available for any stage to process a character is equal to the number of columns in the character. This criteria must be met in the system design for achieving systolic processing.

The overall system design is divided into three phases and each phase is further subdivided. The architecture and algorithms of the systolic array used in each phase will be explained in the following sequence:

1- Edge extraction and ranking phase (chapter III).

2- Chain gathering and sorting phase (chapter IV).

3- Chain smoothing, merging, and classification phase (chapter V).

3.2 Edge Extraction and Storing Phase

During this phase all edges, their start points and end points of a skewed pattern are extracted and stored inside two columns of PEs. Each PE is responsible for processing one row of the pattern, one pixel each cycle. The idea is that a PE detects the start point of an edge by detecting a 0 to 1* or a 1 to 0** transition and then sends a message to the PE beneath it to tracing this edge. The PE that receives the message forwards it further down when it detects the same type of transition and this process repeats until an end point is encountered. Each feature point generated is passed from the first column to the next column of PE's, called the unskewing and storing stage, to be accumulated for the whole pattern. The pattern is skewed in order that an edge starting at an angle bounded by $+45^{\circ}$ or -45° will be detected as one single edge during the extraction.

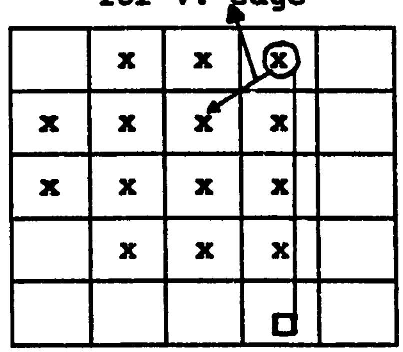
* (that is, white pixel to black pixel)
** (that is, black pixel to white pixel)

3.2.1 Pattern skewing

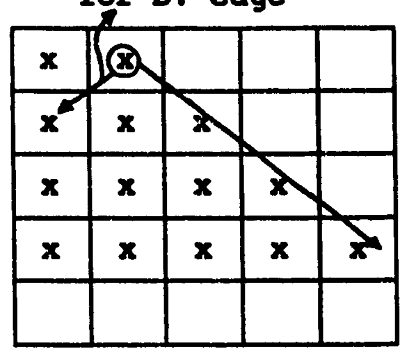
From the above description it is evident that an edge is traced by vertical messages passing between the PEs from top to bottom. Since there must be at least one cycle delay between the time a message is generated at PE_x and the time it is consumed in PE_{x+1}, there is a chance that the transition for the same edge has occurred in PE_{x+1} before the message from PE_x reaches it. This problem can occur in case of a vertical or -45^o slanted edge, as exemplified in Figures 3.2 and 3.3 respectively.

In order to alleviate this problem, we chose to skew the input pattern by delaying the input to each row by two cycles relative to the row above it, as illustrated in Figure 3.4. This skewing enables our first stage to identify horizontal, vertical, and diagonal edges correctly and continuously without breaking them into segments.

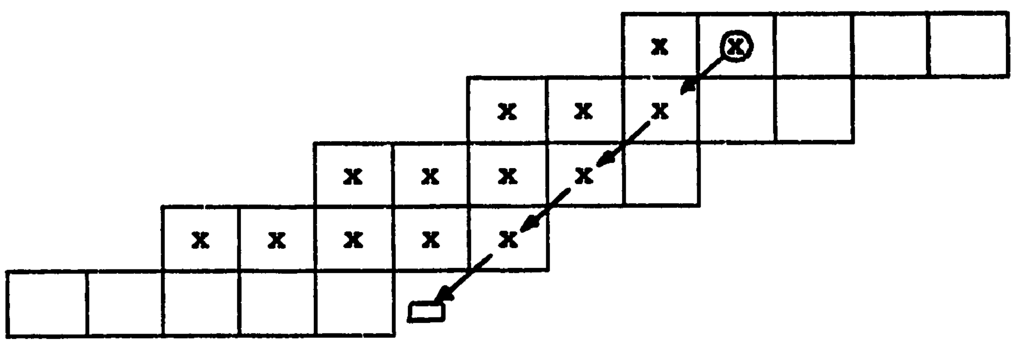
The start points and end points extracted in the first column emerge in a skewed manner as well. To restore the proper alignment, an unskewing network is needed after the first stage. This is the second stage in phase 1. This network reverses the effect of the skewing delay elements as will be shown later. After the extracted points are aligned by the second stage, they will then be stored in the second column of PEs, and this is the third and final stage in phase 1.



vertical edge missed
without skewing
Figure 3.2
Vertical Edge Example



diagonal edge missed
without skewing
Figure 3.3
Diagonal Edge Example



Diagonal edge in figure 3.3 after it is skewed (each row delayed by 2 cycles from its predecessor).

The arrows show that the messages generated are sent below without missing. The diagonal and vertical edges are followed correctly when the pattern is skewed.

Figure 3.4
Patter Skewing

3.3 Edge Extraction and Ranking Algorithm [First stage]

During this stage the start and end points of a pattern will be extracted and passed to the next stage. Each PE has a register S (see figure 3.5) in which it stores the type of transition it is expecting to detect. Initially this register is set to detect a 0 to 1 transition, i.e. a white pixel followed by a black one. Upon detecting the expected transition, a PE generates a start point, tags it with its coordinates, reverses its S register state, generates a left edge message and stores it in one of its registers, generates a right edge message and passes it to the PE below it. After receiving a message from its top neighbor, a PE is alerted to anticipate another transition. If the transition occurs, then the PE sends a new message to the next PE which will continue to trace the edge. On the other hand if a PE is tracing two different edges and a transition occurs then it will pass the message corresponding to the transition to the PE below it, and will reverse its own state (maintained in register S) to continue tracing the other edge. The only case during which a PE may generate an end point and terminate the trace is when a PE has received two edge messages, but no transition is detected during that cycle. This is due to the fact that any two meeting edges either

i) end the body region (black pixels region) that they surround, as in the case of R_2, R_3, R_9 and R_{10} relations in figure 2.4 or

ii) start the merge of two body regions into one single region, as in the case of R_4, R_5, R_6, and R_8 in figure 2.4. In case (i) above we should only have white pixels adjacent to the point where those two edges meet and in case (ii) we should only have black pixels adjacent to the meeting point. In both cases no transition from a 0 to 1 or vice-versa occurs. In this case the PE generates an end point and tags it with its current column and row numbers.

A reset signal, end of frame signal, is included after the last column of each pattern to separate consecutive characters, and to reset each PE to its initial state.

3.3.1 PE Architecture

Figure 3.5 shows the registers, contents and signals, of each PE in the edge extraction stage. Each PE contains two registers ROW# and COL#, the first register stores permanently the *row number* of the pattern which the PE processes, and the second register is incremented each cycle to keep track of the column coordinates of the pixel entering the PE in that cycle. Whenever a PE discovers a transition it reverses the S register's state. If the state is 0-1 it reverses it to 1-0 and vice-versa.

The other registers used within a PE are :

(i) E1 register: it is used to store the message passed from the top PE or the left edge message generated within the same PE when a start point is detected.

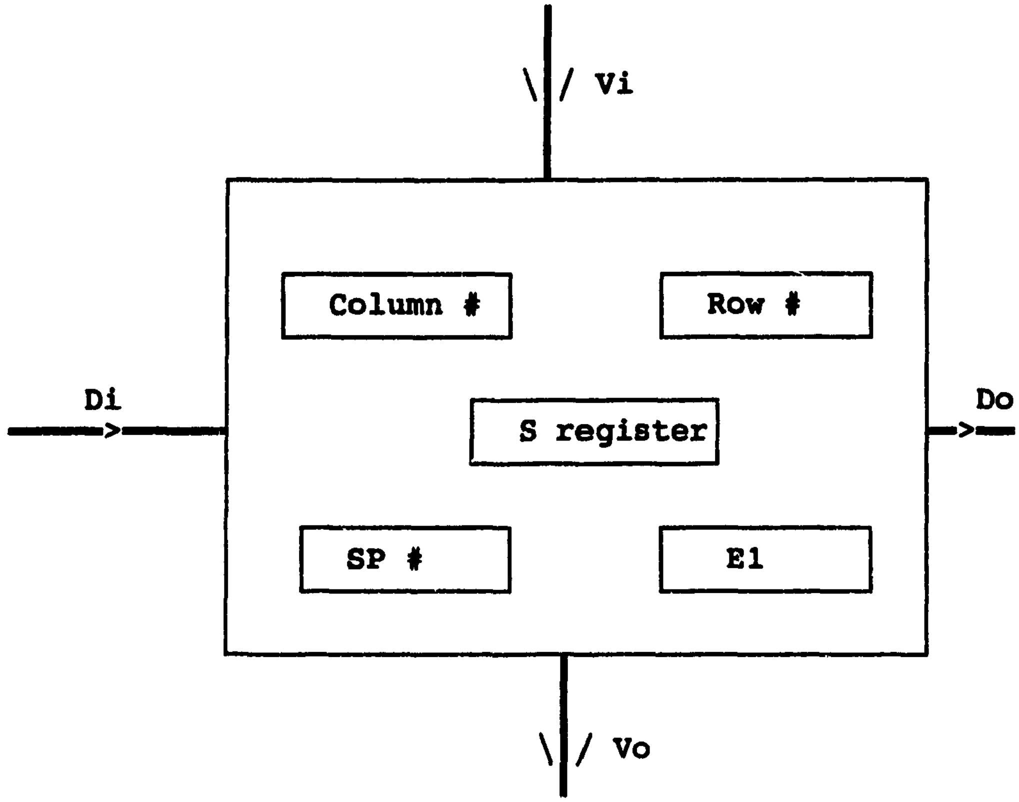


Figure 3.5
PE Architecture of Edge Extractor

S/start point OR E/end point	Left edge data				Right edge data				end point coordinates	
	S	Row	Col	edge type	S	Row	Col	edge type	row #	col #

The number of the start point within it's row

1,2,3 or 4

will have a value in case of end points only

Figure 3.6
Format of Output Data Signal

(ii) SP# register: it is initialized to zero by the reset signal and is incremented by one when the PE detects a start point.

The signals entering and leaving a PE are:

(i) V_i, V_o are two vertical signals. They carry edge continuation messages passed between PEs.

(ii) D_i is a horizontal input data; it carries the input pixel or the reset signal R.

(iii) D_o is a horizontal output data; it carries the generated start and end points to the next stage. The format of D_o is shown in figure 3.6.

3.3.2 Pseudo code for extracting the edge features

Initially all the PE's in the first stage are set to detect a 0 ---> 1 transition. The vertical (message) input to PE_i is set to null.

```
for 2M+N cycles do
     for all PE_i, 1 ≤ i ≤ M+1, pardo
          if the anticipated transition occurs
          then
          if PE_i is receiving a message from PE_i-1
               then
```

```
if PEi contains a message in E register
then
                send message in E below to PEi+1;
                store new message in E register;
        else % no stored messages in E register%
                % this transition corresponds to an
                  edge continuation %
                send this message below to PEi+1;
else % PEi is not receiving a message%
        if PEi contains a message in E register
        then
                send message in E below to PEi+1
        else %this transition corresponds to
                 a start point %
                generate the coordinates of this
                start point;
                generate two edges (of type e1 and
                e2 if this is a 0--->1 transition
                or of type e3 and e4 for 1-->0
                transition);
                store the left edge message (e1 or
                e3) in E register;
                send right edge message below to
                PEi+1;
                output the point to next stage;
```

```
else    % no transition takes place in PEi%
        if PEi receives a message
        then
            if PEi contains a message
            then % this case corresponds to an
                end point where two edges(messages)
                are meeting at a PE while  there is
                        no transition occurring %
                generate the end point coordinates;
                delete the messages received;
                output point to next stage;
            else
                store the message in E register;
% end of algorithm %
```

3.3.3 Theorems of the first stage algorithm

Theorem 1: A PE will not trace more than two edges at any given cycle.

Idea of the Proof: To prove theorem 1 we consider the worst case when input pixels alternate between 0 and 1 consecutively in a row and prove that even in the worst case no more than two edges will be traced within any PE as demonstrated below in PE_x.

CYCLE NUMBER

4	3	2	1	0	
d	d	0	0	0	PE_{x-1}
0	1	0	1	R	PE_x
d	d	R	d	d	PE_{x+1}

d means don't care pixel

R is the reset signal

During cycle 1 PE_x detects a start point and sends a message to PE_{x+1} to continue tracing the right edge. PE_x also stores the left edge in its E1 register, and changes its S register to 1-0 state. During cycle 2 a 1-0 transition occurs in PEx as anticipated by its S register. Because there is a valid edge data in E1, PE_x will pass it to PE_{x+1} and will then clear the E1 register and reverse its S register back to 0-1. During the same cycle PE_{x+1} receives the right edge message sent from PE_x in cycle 1 and stores it in its E1 register as no transition is detected. During cycle 3 PE_x detects another new transition while no edge is stored in its registers. It then generates a new start point with two edges, sends the right edge down, stores the left edge in E1 and reverses its S register to 1-0. During the same cycle PE_{x+1} receives the left edge message of the first start point from PE_x. It now contains two edge messages.

If a transition occurs in it during this cycle, i.e. the don't care pixel is a 1, then it will pass the edge stored in E1 to a lower PE, take the new edge and store it in E1 and reverse its S register to 1-0. In this case only one edge is remaining in PE_{x+1}. The second case happens if no transition occurs, i.e. the don't care pixel in the above illustration is a 0, then the two edges lead to an end point and both of the registers will be cleared, leaving no edge in PE_{x+1}.

At cycle 4 and 5 the same process repeats proving that no more than two edges will be contained in any PE.

Corollary 1: No more than one start point or one end point will be generated by any PE during any cycle.

Corollary 2: Signals V_i and V_o will never carry more than one edge message during any cycle.

Corollary 3: The total number of start points extracted by the column of PEs from a single pattern has to be equal to the total number of end points extracted from the same pattern.

Corollary 4: An end point can only be detected after the two start points of the meeting edges have been detected. Also the PE detecting it has to be lower to, or at the same level as the PE that detected the lower start point.

3.3.4 Edge detection examples

Figures 3.7 and 3.8 shows two examples of this algorithm. They depict the space-time diagram of the messages passing inside and between PEs. The pattern has been skewed before it is fed to the column of PEs. Also one column of R (reset signals) is included at the end of each pattern. The diagonal arrows appearing on the space-time diagram correspond to the V_i and V_o messages passed between PEs.

Referring to point 1 in Figure 3.7 we notice that PE_8 sees two edge messages in that cycle while a transition from 0 to 1 occurs. Accordingly the horizontal message is passed to the PE below and the message it just received from it's upper PE takes the place of the former. On the other hand, point 2 in Figure 3.7 indicates an end point because no transition occurs while PE_6 sees two edge messages in that cycle. Point 3 in the same figure shows a transition from 0 to 1 in the absence of an edge message. This generates a start point with e^1 as left edge and e^2 as right edge. The right edge is passed to the PE below. Point 4 is similar to point 1 where PE_{10} sees two edge messages during that cycle while a transition from 0 to 1 occurs. The PE passes the horizontal message to PE_{11} below and new message replace the former.

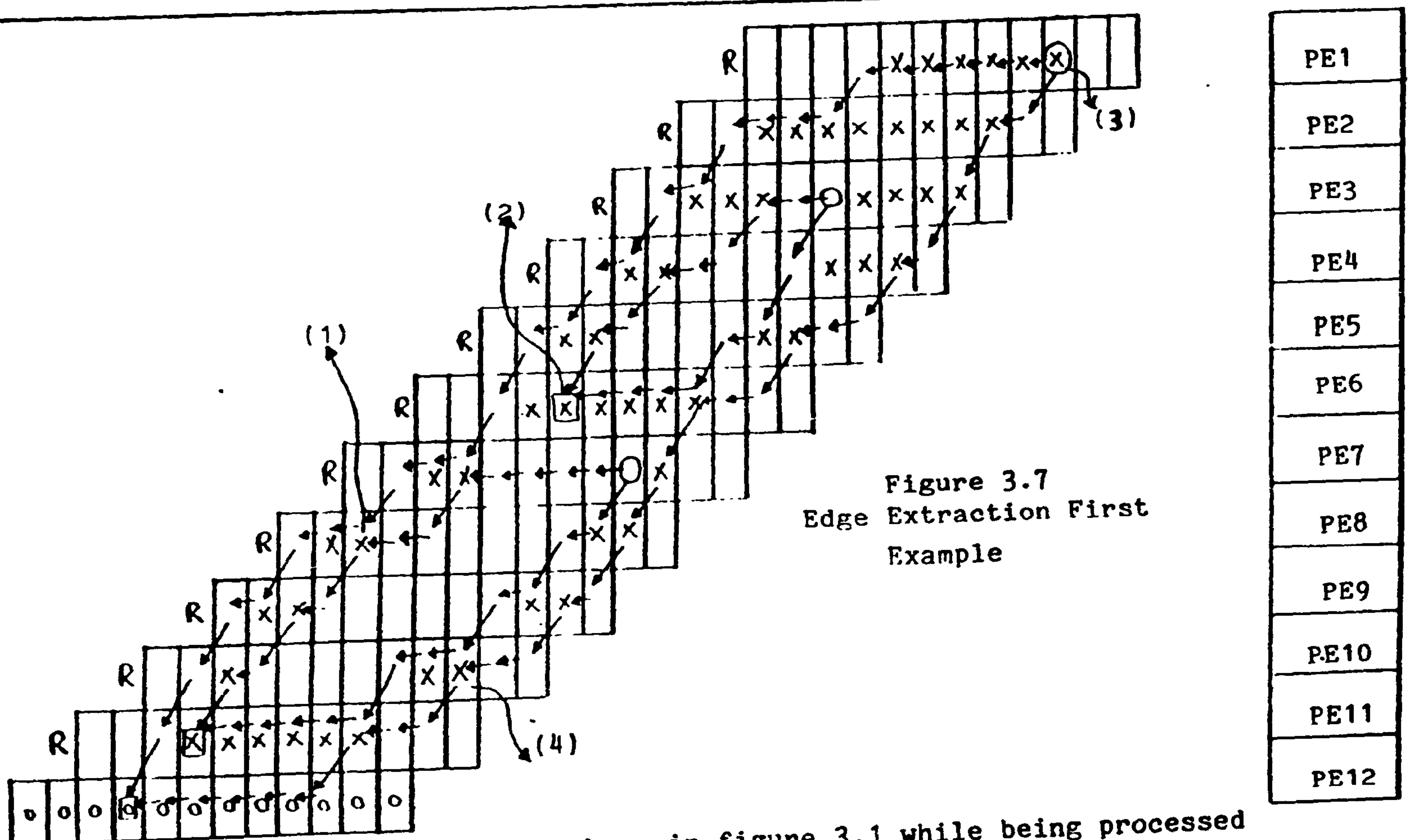


Figure 3.7
Edge Extraction First
Example

Time space diagram for pattern shown in figure 3.1 while being processed in the edge extraction stage.

Diagonal arrows show the messages passed between PEs while data is pumped

Horizontal arrows show edges that are stored in the PE.

Figure 3.8
Edge Extraction
Second Example

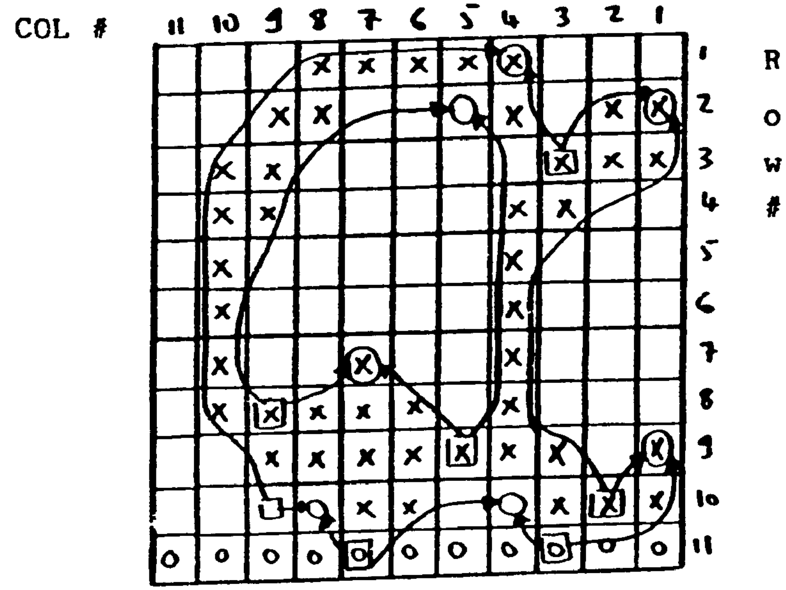


Figure 3.8.a

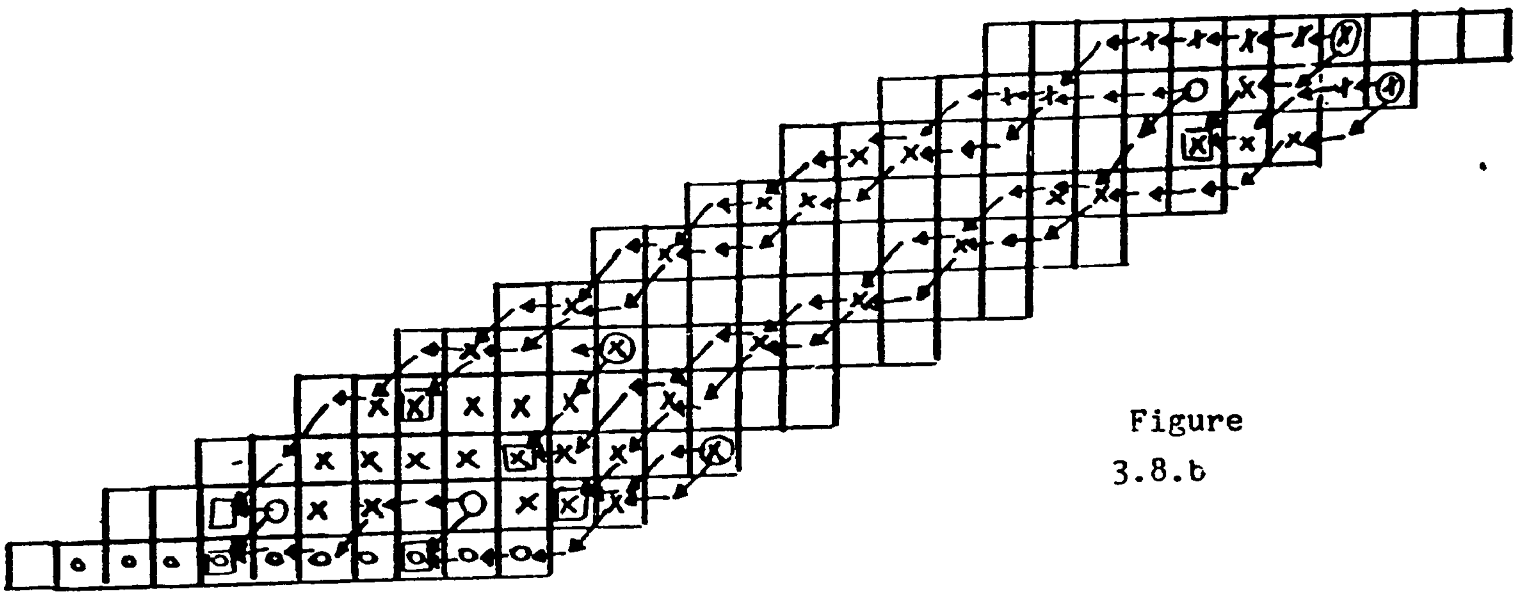


Figure
3.8.b

Time space diagram for the above pattern in figure 3.8.a

3.3.5 First stage performance

We will position all our characters so that they fit in a standard matrix of MxN by simply including empty rows or columns. Then all our calculations for performance will be done over the standard MxN matrix.

For any input pattern of M rows and N columns,

(i) Flush time, which is the time needed between the arrival of the first pixel of the first row and that of the first pixel of last row to the column of PE, is 2M - 2. This time is actually the result of the two cycles skewing we introduced earlier.

(ii) The delay between the first pixel to enter this stage and the last processed point to leave the stage equals the flush-time + N = 2M + N - 2.

(iii) In every N + 1 cycles the PE column in stage 1 extracts all the start and end points in one pattern's matrix.

(iv) The number of PEs used in this stage is M. Those PEs are only linearly connected (i.e. each PE is connected to the one above it and the one below it) which minimize the communications network complexity.

3.4 The unskeweing and storing stage [second stage]

During this stage the extracted points from stage 1 will be unskewed to realign them. A delay element network is used to perform this task. After the unskewing is performed the start and end points extracted will be collected in a second column of PEs as discussed earlier. The storing stage eliminates message collision and interference between consecutive patterns. It also helps in keeping the data bandwidth between stages constant, as it will be shown in the following chapters.

The following assumption will be made: "**No more than three start points and three end points can occur within any row of a pattern**". This assumption was found to hold during the simulation experiment on the 1200 scanned digits from 0-9 and enclosed in a [54x54] matrix. The maximum number of start and end points in a single row had to be determined in order to design a fixed architecture for the PEs used hereafter. Although this assumption suits regularly sized handwritten numerals, it may not be valid for completely unconstrained handwritten characters. In such cases minimal changes to the storing and tagging stages will be required to increase registers and clock rates, but the algorithms used will still hold.

3.4.1 Storing stage Procedure

Upon receiving a start or an end point, the PE will store it in one of the empty registers. Each PE contains three registers to store the start points and three other registers to store the end points. Whenever the column of storage PEs receives an R (reset signal) it then pumps its stored data out to the next phase, namely, the chain gathering phase. This means that the complete set of start and end points extracted from any pattern will be pumped out from this stage only at the end of every N cycles. Figure 3.9 shows the points extracted from the pattern shown in figure 3.8.a. It also shows how the points are stored during this stage.

3.4.2 The PE architecture of the storing stage

Figure 3.10 shows the contents of each PE in this stage. The registers S1, S2, and S3 store three start points. The registers E1, E2, and E3 store three end points. The input signal D_i delivers the start and end points extracted from stage 1. Its format is the same as D_o format in Figure 3.6. The output signals DS_o and DE_o deliver one start point and one end point, respectively, to the next stage. They are activated only after receiving an end of frame, i.e. a reset signal, in D_i. Starting at the cycle during which a reset signal is received, and during the following two cycles, each PE will pump the contents of its S and E registers to the next stage.

PE1	[S/1,1,4,1/1,1,4,2] └─>number of the start point in row
PE2	[S/1,2,1,1/1,2,1,2] [S/2,2,5,3/2,2,5,4]
PE3	[E/1,1,4,2/1,2,1,1/3,3] left right └─>end point edge edge coordinates
PE4	
PE5	
PE6	
PE7	[S/1,7,7,1/1,7,7,2]
PE8	[E/1,7,7,2/2,2,5,3/8,9] └> number of the start point /7,7,2/ within the row where it's discovered
PE9	[S/1,9,1,1/1,9,1,2] [E/1,7,7,2/2,2,5,4/9,5]
PE10	[E/1,2,1,2/1,2,1,1/10,2] [S/1,10,4,3/1,10,4,4] [S/2,10,8,3/2,10,8,4] [E/2,10,8,3/1,1,4,1/10,9]
PE11	[E/1,10,4,4,1/1,9,1,2/11,3] [E/2,10,8,4/1,10,4,3/11,7]

Figure 3.9
Data Sent to Storage Stage

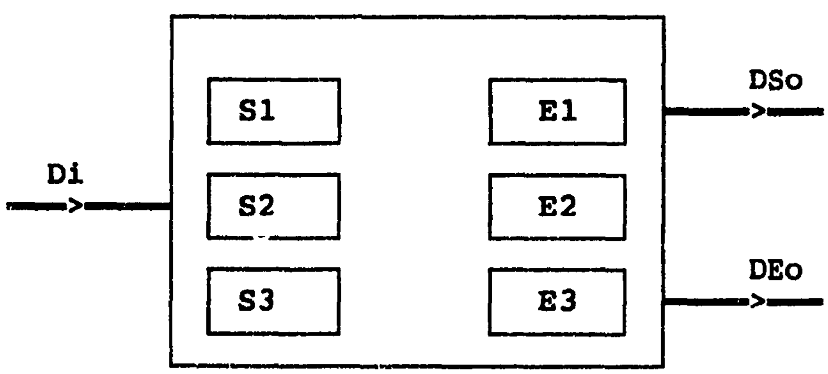


Figure 3.10
PE Architecture of Storage Stage

CHAPTER IV

Feature Gathering and Sorting

During this phase, phase II, the data extracted in phase I will be processed to order all the start and end points on each continuous contour of the pattern into a chain. Elements in a chain are sorted in the clockwise order. This is followed by smoothing of the list (by deleting relatively small features; three pixels or less) and merging all the chains into one complex chain, which will be explained in chapter V.

Figure 4.1 shows a processed pattern at the end of phase I. As could be seen, the pattern has two chains. What we want to achieve at the end of phase II is to collect and order the points of each chain. For the example pattern shown in Figure 4.1, at the end of phase II the following lists should be produced.

Chain 1 (C1): (S1, E3, S4, E6, S5, E2, S2, E1).
Chain 2 (C2): (S3, E5, S6, E4).

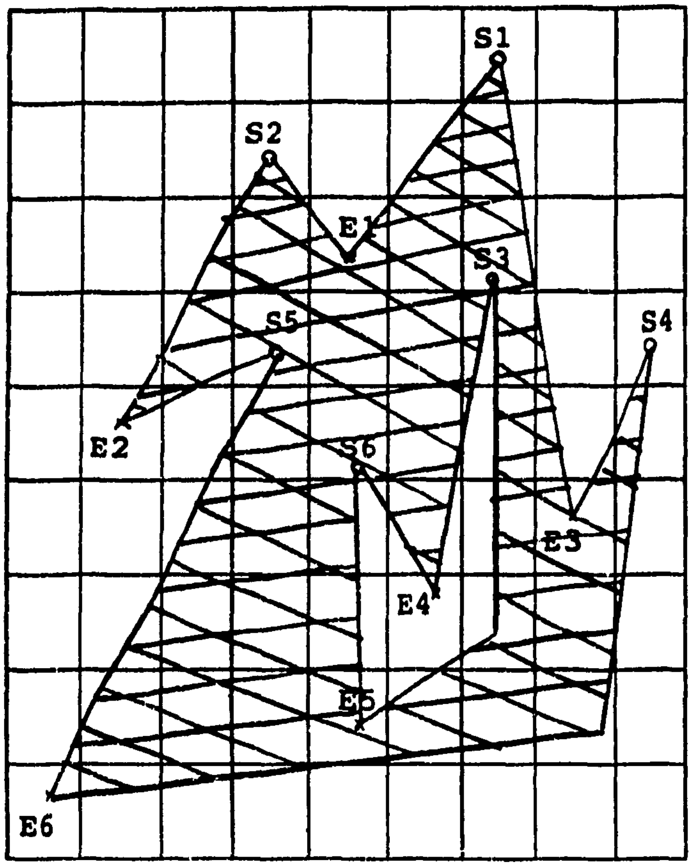
Note that S1 has to be the first point in chain C1 because it's the point with the highest (row) rank. Similarly S3 is the first point on C2. Also notice the clockwise order in which the two chains are written.

In order to accomplish the above, the design of phase II is divided into the following three stages:

(i) Tagging stage.

(ii) Rerouting stage.

(iii) Sorting stage.



PE1	S1	1,1,3,e	1,1,3,e	
PE2	S2	1,2,6,e	1,2,6,e	
PE3	S3 E1	1,3,3,e 1,1,3,e	1,3,3,e 1,2,6,e	3,5
PE4	S4 S5	1,4,1,e 2,4,6,e	1,4,1,e 2,4,6,e	
PE5	S6 E2	1,5,5,e 1,2,6,e	1,5,5,e 2,4,6,e	5,8
PE6	E3	1,1,3,e	1,4,1,e	6,2
PE7	E4	1,5,5,e	1,3,3,e	7,4
PE8	E5	1,5,5,e	1,3,3,e	8,5
PE9	E6	2,4,6,e	1,4,1,e	9,9

Figure 4.1 Data collected in storage stage after processing shown pattern in 1st phase (S# is a Start point , E# is an End point)

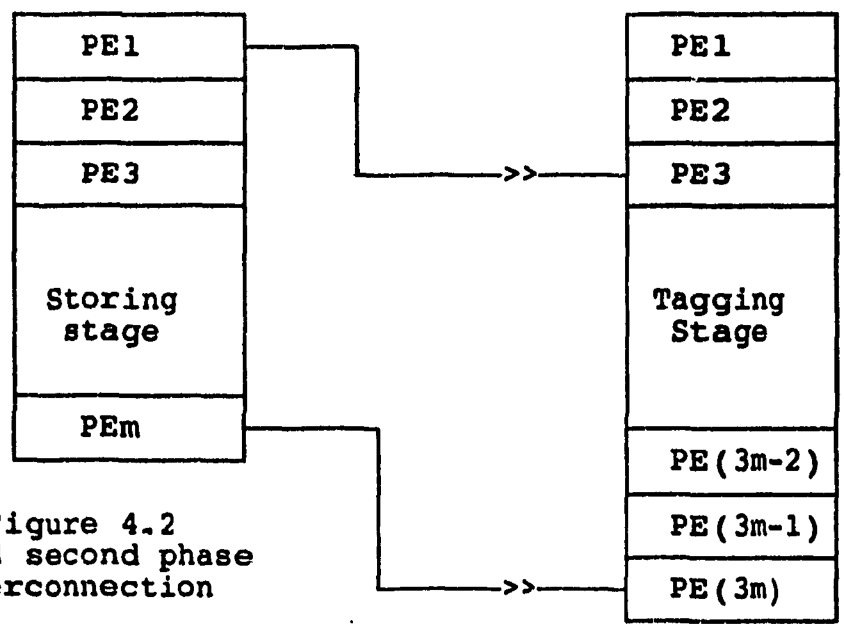


Figure 4.2
First and second phase interconnection

4.1 Tagging stage

During this stage each point on chain C_i, $1 \leq i \leq n$, where n is the number of chains in the pattern, will be tagged with the row number of a higher point on the same chain C_i. Only the highest point on C_i will remain untagged. From the format of signal D_o, shown in figure 3.6, we notice that the data contained in each end point E_j carries the following information:

(i) The coordinates of the start point connected to the left edge of E_j.

(ii) The coordinates of the start point connected to the right edge of E_j.

Where a right edge is defined as an edge whose slope is between 0^o and 45^o at the end point. On the other hand a left edge has a slope between -45^o and 0^o at the end point.

So by simply comparing the coordinates of these two points, we can detect the higher start point of an end point. Knowing that, the PE containing that end point sends a message/tag to the lower start point so that the latter knows which PE it may go to. This message/tag contains the address of the higher start point. The same tag also applies to the end point. At the end of this stage each start and end point will be tagged with the address of a higher start point and only the highest start point will remain untagged.

4.1.1 Tagging algorithm

The number of PEs used in this stage is 3M. The triple size is due to the fact that each PE in the storage stage in phase I can hold up to three start points and three end points. Figure 4.2 shows the interconnection between the storage stage and the tagging stage. Only every third PE in the tagging stage is connected to a PE in the storage stage. The first set of start and end points is passed from each PE in the storage upon reset to PE_x in the tagging stage, PE_x sends them to PE_{x-2} above through PE_{x-1}. During the following cycle the second set of start and end points will be picked by PE_x and sent to PE_{x-1}. During the third cycle the third set of start and end points are picked by PE_x so that each PE becomes responsible for storing the data and tags corresponding to at most one start point and one end point. This approach reduces the complexity of each PE, and the data bandwidth requirement between adjacent ones, as well as it improves the expandability of the architecture when more than 3 start and 3 end points may be encountered in each row of the pattern matrix. (This case might happen if we use higher scanning resolutions.)

The clock-rate during this stage has to be 3 times that of the previous stages because the PE column triples in size while data rippling is done systolically between adjacent PEs.

Upon receiving an end point, PE_x compares the coordinates of its two start points, and identifies the PE number (row number) of the highest start point. The address tag is obtained using the following formula:

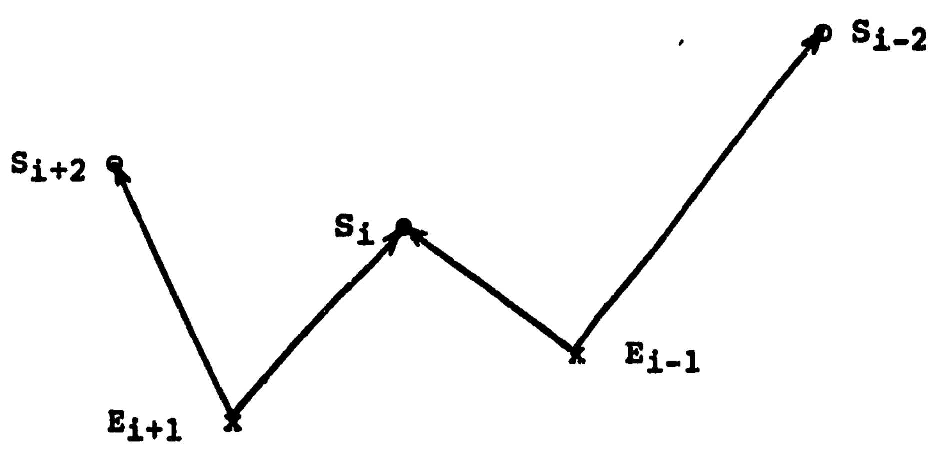
Formula 1: This formula is used to calculate the address at which a start and end point will be stored as mentioned earlier and it is also used to calculate the addresses of start points to which the tags are sent.

Address tag = ((row# - 1) * 3) + S#

where: row# is the row number of the start or end point

S# is the number of the start point within the row as shown in figure 3.6.

It then sends the tag message to the lower start point. It also tags itself with the same PE number. A start point in any PE may receive two tags from two different end points. This case can only happen if the start point is the lower for both end points it is connected to, as shown below.



Point S_i is the lower start point for end points E_{i+1} and E_{i-1}. So it will receive tagging messages from both end points. Eventually (regardless of the arrival ordering of these two tags), point S_i compares the two tags and store the larger one in its tag register. In the above example this is the address tag pointing to the PE containing S_{i-2}. It then sends a new tag message to S_{i+2}. This tag message also points to the PE containing point S_{i-2}. It is thus apparent that the tag register of a PE containing a start point may be readjusted each time a new tag is received.

4.1.2 PE architecture of the Tagging stage

Figure 4.3 shows the registers and signals of each PE in this stage. Following is a description of each register:

(i) PE# register contains the row number of the PE within the column. Numbering ranges from 1 for higher PE to 3M for lower one.

(ii) S register is used to store the data of one start point.

(iii) E register is used to store the data of one end point.

(iv) STAG register is used to store the address tag to which the start point stored in S will be rerouted.

(v) ETAG similar to STAG but is used for the end point.

The signals entering and leaving each PE include:

(i) DS_i, and DE_i carry start and end point data respectively. This is the data passed from the storage stage during the three cycles following the reset.

(ii) S_i, S_o, E_i, and E_o carry the start and end points to their corresponding storage address.

(iii) TO_i, and TO_o carry the address-tag to a specific start point.

(iv) DS_o, and DE_o forward the S register data with its STAG and the E register data with its ETAG to the next stage, the rerouting stage at the end of this tagging process (flushed by reset).

The format of some of the above signals is shown in figure 4.4. Each signal is formed from two parts as follows:

1. The PE#, or the address, to which this signal is being sent.

2. The message or the value to be forwarded to this PE#. This message can be a rerouted address tag as in the case of TO_i, a start point data, or an end point data as in the case of S_i and E_i respectively.

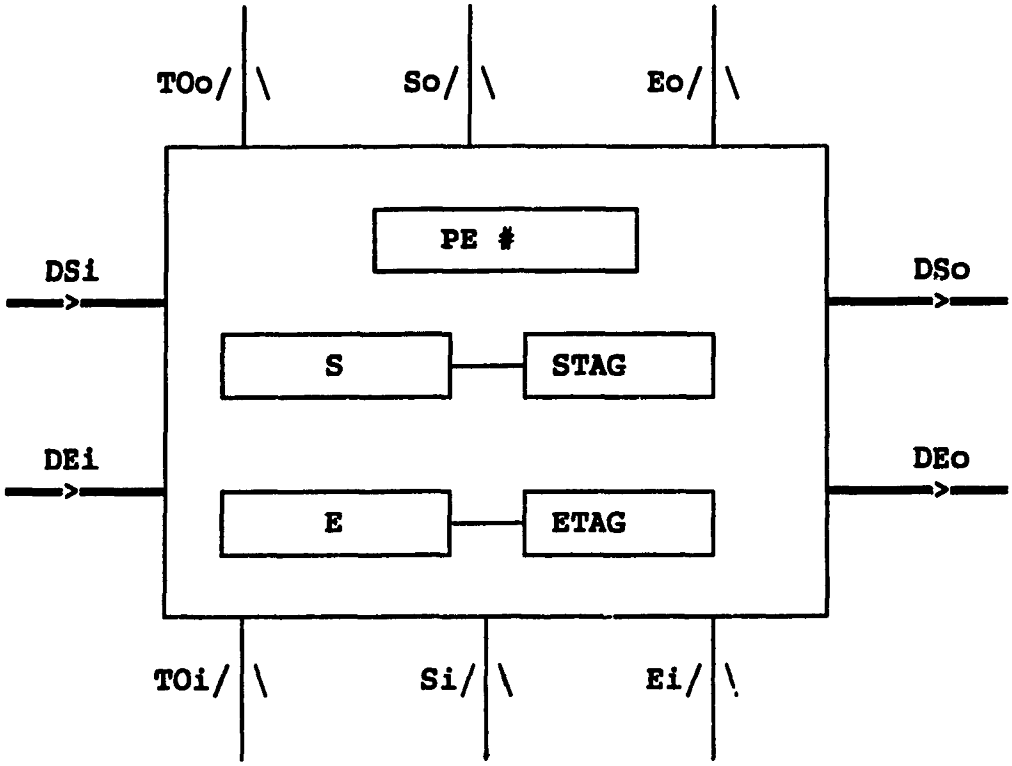


Figure 4.3
PE Architecture of the Tagging Stage

Destination	Contents
TO-PE#	Start point data

Format of Si and So

Destination	Contents	
TO-PE#	End point data	Etag

Format of Ei and Eo

Destination	Contents
TO-PE#	STAG

Format of TOi and TOo

Figure 4.4
Data Format of Tagging Stage

4.1.3 Pseudo code of the tagging stage algorithm

After receiving a reset signal all registers inside a PE are cleared.

```
n = 0;
for the first 3 cycles after the reset signal do
     increment n by 1;
     for all PE_i  1 ≤ i ≤ 3(M+1) pardo
          if DS_i carries a start point
          then
               generate message (PE_{i-3+n} <---- DS_i);
               send this message upward to PE_{i-1} through
               S_o signal;
          if ES_i carries an end point
          then
               generate message (PE_{i-3+n} <---- DE_i);
               send this message upward to PE_{i-1} through
               E_o signal;
               tag this end point with min{j,k} (where
               j and k are the numbers of the two
               PEs that contain the two start points
               whose edges are meeting at this end point)
               %this numbers are included within the end
               point data when it was generated in the first
               stage %
```

```
        generate message (PE_min{j,k} <---- PE_max{j,k})
        and send it to PE_i-1 through TO_0 signal;

    if S_i carries message (PE_x <----- SD) (where SD
        stands for Start point Data)
    then
        if x = i
        then %the start point reached its storing PE%
            store SD in S register;
        else
            send S_i to PE_i-1;
    if E_i carries message (PE_x <----- ED) (where ED
        stands for End point Data)
    then
        if x = i
        then % the end point reached its storing PE %
            store ED in E register;
            store its tag in ETAG;
        else
            send E_i to PE_i-1;

if n > 3 then n = n + 1; %where n is current cycle #%
```

```
for all the cycles where 1 ≤ n ≤ 3(M+1) do
     for PE_i, 1 ≤ i ≤ 3(M+1) pardo
          if TO_i carries message (PE_k <---- PE_j)
          then
               if i = j
               then %this message corresponds to this PE %
                    if STAG is empty
                    then
                         store k in STAG;
                    else
                         let l be the value in STAG;
                         generate the message
                         (PE_min{l,k} <---- PE_max{l,k});
                         send this message to  PE_i-1 through
                         TO_o;
                         store min{l,k} in STAG;
               else %message does not correspond to this
                     PE %
                    send TO_i to PE_i-1 through TO_o;
% end of algorithm %
```

4.1.4 Theorems of the Tagging algorithm

Theorem 2: All the signals generated by any end point would be directed vertically, from bottom to top.

Proof: From corollary 4 in section 3.2.3 which stated that "an end point has to be lower than or at the same row as its lowest start point", it follows that all the address tag signals that any end point may generate have to be sent either to the same PE or to a higher PE. This means that the signals would be directed from bottom to top.

Q.E.D.

Another point has to be taken care of during this stage. A new reset signal should not reach this stage until all the vertical tag migrations have reached their final destinations. In order to eliminate this overlapping problem we have to adjust the width of the input pattern to be no smaller than its height. In the worst case, this can be guaranteed by adding empty columns before or after the pattern till M, number of rows in the pattern, become less than N, number of columns in the pattern. This leads to the following:

Implied Rule 1 : No overlapping will occur between two consecutive input pattern frames if and only if M, the number of rows, is less than N, the number of columns, in each pattern entering the system.

Validity of Rule 1 : Assume a reset signal is received at cycle t_0. On the following cycle, cycle t_1, all the points inside the PEs of the storing stage will be passed to the PEs of the tagging stage. During the same cycle all the vertical signals would be generated. The longest path a vertical signal can take will be from the bottom PE_{3M} to the top PE_1.The last signal will reach PE_1 at cycle t_{1+3M-1}, i.e. t_{3M}. Since the clock is three times faster during this stage than in previous stages, and since a reset signal is included after each N columns to separate consecutive patterns, the next reset signal will be entering the tagging stage during cycle t_{3N}. In order to eliminate the overlapping, t_{3N} has to be greater than t_{3M}, or N has to be greater than M (N > M).

4.1.5 Example for the tagging algorithm

Figure 4.1 shows a pattern after it has gone through phase I. Figure 4.5 shows the space time diagram of the tagging stage while processing the results of phase I for the example.

At cycle 1.1 a message 7 <-- S3 is generated in PE_9. This means that start point number 3 is to be rerouted to PE number 7. The number 7 is calculated using formula 1. Since the row# of S3 is 3 and its SP# is 1 then :

address tag = ((3 - 1) * 3) + SP# = 7.

Also a 1 <-- 4 message is generated. This message is caused by end point E1. When the two edges of E1 were compared it was discovered that the left edge connected to point S2 is lower than the right edge connected to point S1. The reroute tags are then calculated using formula 1 and the 1 <-- 4 message that will tell PE number 4 to send its start point data to PE number 1 is generated.

At cycle 2.1 (after three cycles from last reset) all the start points arrive at their destinations and are stored in their respective S registers.

At cycle 6.2, we notice that the message 10 <-- 11 reaches PE_{11} and detects that the STAG register already contains a value (4), that is less than 10. In this case a new message 4 <-- 10 is generated telling PE number 10 to reroute to PE number 4. This message appears on the figure at cycle 6.3 in PE number 10. When this message reaches PE_{10}, it is converted into a new one containing 1 <-- 4 for similar reasons. At cycle 9.3 all the tag registers will contain their rerouting addresses.

PE#	Cycle 1.1		Cycle 2.1		Cycle 6.2		Cycle 6.3		Cycle 7.1		Cycle 8.3		Cycle 9.3	
1			S_1 --		S1 --		S_1 --		S_1 --		S_1 --		S_1 --	
2														
3		1 <-S_1												
4			S_2 --		S_2 1		S_2 1		S_2 1		S_2 1	1<-4	S_2 1	
5														
6		4 <-S_2		1<-4										
7			S_3 --		S_3 --		S_3 --		S_3 --		S_3 --		S_3 --	
8														
9		7 <-S_3 1<-4								1<-4				
10			S_4 --		S4 1		S_4 1	4<-10	S_4 1		S_4 1		S_4 1	
11			S_5 --		S_5 4	10<-11	S_5 4		S_5 4		S_5 4		S_5 4	
12		10<-S_4 11<-S_5		4<-11										
13			S_6 --		S_6 7		S_6 7		S_6 7		S_6 7		S_6 7	
. .														
27		10<-11												

* each cycle is split into 3 sub-cycles to accomodate the systolic data passed. Also the end points are not shown for simplicity.

Figure 4.5
Time Space Diagram of Tagging Stage

4.2 Rerouting stage

After tagging each start and end point with the address of a higher start point, as explained in section 4.1, we now want to collect the chains. Each PE containing a feature point now can forward it to the proper destination according to its accompanying tag. When the feature point arrives at its destination, it is either collected or rerouted again to a higher PE. The latter situation arises when the receiving PE is also tagged to forward such receipt to a higher PE. So the start and end points keep on migrating upward until they finally reach a destination which is not tagged. The later should correspond to the highest start point in the chain, according to theorems 1 and 2. In this stage also N has to be greater than M and the clock is three times faster than that in phase I.

4.2.1 PE architecture of the reroute stage

Figure 4.6 shows the contents of each PE in this stage.

(i) PE# register contains the row number of the PE in the column.

(ii) TO register stores the address-tag to which all points, whose destination is this PE, should be rerouted.

The signals entering and leaving each PE are:

(i) S_i, S_o, E_i, and E_o carry the start and end points with their destination tags.

(ii) DS_i, and DE_i carry information passed from the previous stage for rerouting

(iii) DS_o, and DE_o carry the start and end points to the next stage after rerouting for sorting.

4.2.2 Pseudo code of the rerouting stage

```
for all the n cycles where 1 ≤ n ≤ 3(M+1) do
     for PE_i, 1 ≤ i ≤ 3(M+1) pardo
          if DS_i carries data
          then
               if this data has no tag
               then % this PE is a chain collector%
                    % this means that all the start and end
                    points lying on the same chain as the
                    currently entering this PE will be
                    rerouted here for collection %
                    collect point and send it to the sorter;
               else % received point (data) is tagged %
                    store this tag in TO register;
                    send point upward through S_o;
          if ES_i carries data
          then % received end point (data) HAS TO BE tagged%
                    send point upward through E_o;
          if S_i carries message (PE_x <----- SD) (where SD
               stands for Start point Data)
          then
               if x = i
               then % the start point reached its tagged PE%
```

```
                if TO register is empty
                then %this PE is a collector %
                     collect point and send to sorter;
                else (let l be the value in TO register)
                     generate message  PE_l <---- SD;
                     send point upward through S_o;
      if E_i carries message (PE_x <------ ED) (where ED
           stands for End point Data)
      then
           if x = i
           then % the end point reached its tagged PE%
                if TO register is empty
                then %this PE is a collector %
                     collect point and send to sorter;
                else (let l be the value in TO register)
                     generate message  PE_l <---- ED;
                     send point upward through E_o;

% end of algorithm %
```

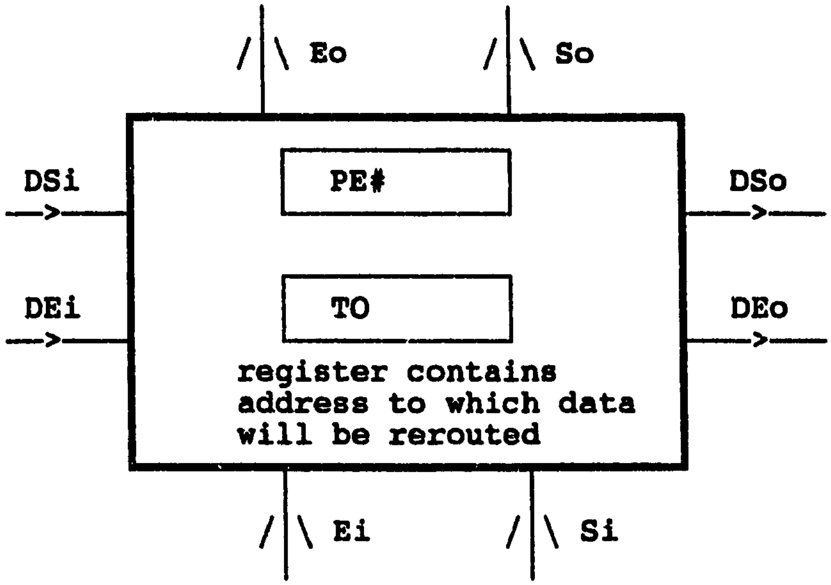


Figure 4.6
PE Architecture of Rerouting Stage

4.2.3 An example for the rerouting stage

Figure 4.7 shows the space time diagram of the rerouting algorithm for the same pattern shown in Figure 4.5. At cycle 1.1 start point S5 will start its trip from PE_{11} with a destination toward PE number 4, according to its tag register. At cycle 3.2 this point reaches PE number 4 and detects that the tag register of PE number 4 is not empty, and contains a reroute tag to PE number 1. So start point S_5 will continue its trip up until it reaches PE number 1 at cycle 6.1 where it will be collected.

PE # \ cycle #	1.1		1.2		3.2		4.1		6.1		9.1, 9.2, 9.3	
1	S_1 —	S_1	—	S_1	—	S_1, S_2 E_1	—	$S_1 S_2$ $E_1 S_4$	—	$S_1 S_2 E_1$ $S_4 S_5 E_2$ E_3	—	$S_1 S_2 E_1 S_4$ $S_5 E_2 E_3 E_6$
2								$1 \leftarrow S_5$				
3				$1 \leftarrow S_2$		$1 \leftarrow S_4$						
4	S_2 1	$1 \leftarrow S_2$	1		1	$1 \leftarrow S_5$	1	$1 \leftarrow E_2$	1		1	
5												
6				$1 \leftarrow E_1$		$4 \leftarrow E_2$						
7	S_3 —	S_3 $1 \leftarrow E_1$	—	S_3	—	S_3, S_6	—	$S_3 S_6$ $1 \leftarrow E_3$	—	$S_3 S_6 E_4$ E_5	—	$S_3 S_6 E_4 E_5$
8												
9				$1 \leftarrow S_4$		$1 \leftarrow E_3$						
10	S_4 1	$1 \leftarrow S_4$	1	$4 \leftarrow S_5$	1		1	$7 \leftarrow E_4$	1	$1 \leftarrow E_6$	1	
11	S_5 4	$4 \leftarrow S_5$	4		4		4		4		4	
12				$7 \leftarrow S_6$ $4 \leftarrow E_2$		$7 \leftarrow E_4$						
13	S_6 7	$7 \leftarrow S_6$ $4 \leftarrow E_2$	7		7		7	$7 \leftarrow E_5$	7		7	
14												
15				$1 \leftarrow E_3$		$7 \leftarrow E_5$						
16		$1 \leftarrow E_3$						$10 \leftarrow E_6$				
17												
18				$7 \leftarrow E_4$		$10 \leftarrow E_6$						
19		$7 \leftarrow E_4$										
20												
21				$7 \leftarrow E_5$								
22		$7 \leftarrow E_5$										
23												
24				$10 \leftarrow E_6$								
25		$10 \leftarrow E_6$										
26												
27												

PE1 Collected all the start and end points on the outer chain of pattern shown in Figure 4.1

PE7 Collected all the points on the inner chain of pattern shown in Figure 4.1

Figure 4.7 Time-Space diagram for rerouting stage while processing the pattern shown in figure 4.1. This stage is prformed after processing the pattern in the tagging stage.

4.3 The sorting stage

So far, we have detected all the start and end points that lie on the inside and outside contours of the pattern, and collected all the points that lie on the same chain through a single PE, which receives the highest point of that chain. The latter has never been rerouted.

The third stage, namely sorting stage, of the second phase will order these points in a clockwise direction for subsequent classification.

4.3.1 PE architecture of the sorting stage

Figure 4.8 shows the contents of each PE in this stage. The registers used are:

(i) S register: stores a start point data.

(ii) E register: stores an end point data.

The signals entering and leaving a PE are:

(i) V_i, and V_o: carry the data stored in E and S registers.

(ii) Swap signal: exchanges the contents of PE_i with PE_{i+1}.

(iii) E_i, and E_o: carry the migrating end points.

(iv) D_o: generated at the end of the stage carries the data stored in E and S registers and passes it to the last phase, namely the classification phase.

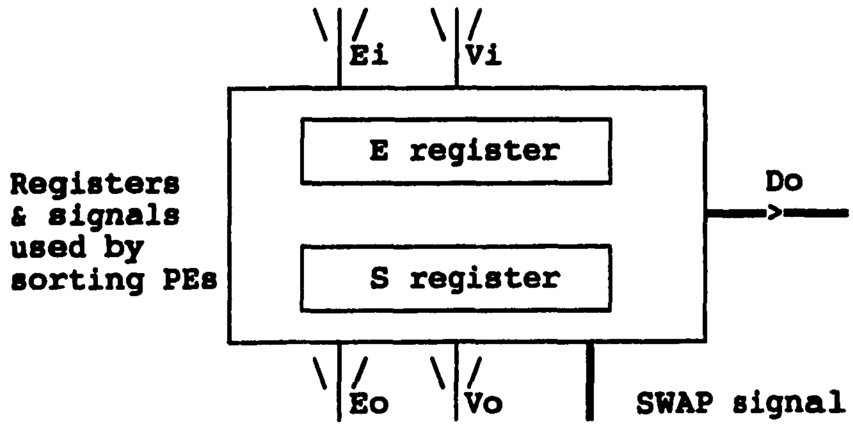


Figure 4.8
PE Architecture of the Sorting Stage

4.3.2 Sorting algorithm

As mentioned earlier, the first point that enters this stage has to be the highest start point. The first PE will receive this point and store it in its S register. This point is never migrated from that PE. Each chain in the input pattern will be processed in a separate column. The first PE acts as a dispatcher and upon receiving a start point it forwards it to its lower PE (except for the first start point in the chain). The latter will store this point and forward any old points, i.e. any start or end point that was already stored in it, further down. The first PE also checks each end point to see if its right edge is connected to the start point it holds. If so, the first PE will tag this point with (E) which means that this end point should reside in the last PE at the bottom of the sorting column.

Any end point entering the sorting stage will be guaranteed to find its two start points below it as will be shown and prooved latter. An end point keeps on migrating till it reaches the PE containing the start point connected to its left edge. While migrating and searching for its left start point, the end point might pass through its right start point. In such a case a data swapping takes place between the PE containing the right start point and the one below.

Some start point of the chain may be connected to the left edge of two end points, as illustrated in figure 4.9. Start point S5 is connected to the left edge of end points E6 and E4. According to our algorithm and since each end point entering the sorting stage is always searching for the start point that is connected to its left edge, both end points E6 and E4 will be searching for the PE containing start point S5. Since any PE in this stage is allowed to store only one start point and one end point, then the PE containing S5 has to make a decision on which end point should be stored with S5 and which end point should be passed below. To solve this conflict the PE uses the following rule:

Implied Rule 2 :

"The end point that is kept in case of conflict is the one that is connected to the LEFT EDGE of the start point, and the other end point is passed below to the following PE."

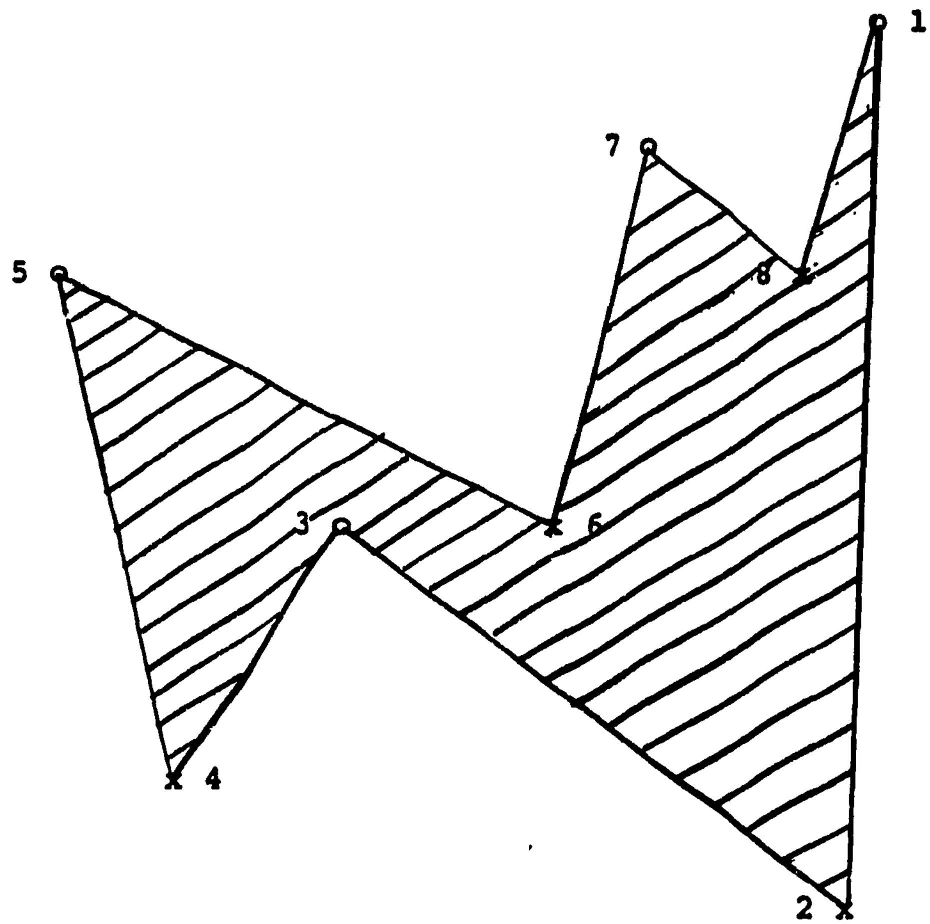


Figure 4.9

Example showing an end point hooked to the left edge of it's two start points

4.3.3 Sorting Stage Theorems

Since corollary 4 in section 3.2.3 guarantees that any end point will be detected in the edge detection stage if and only if its two start points (i.e. the two start points connected to its left and right edges) would have been already detected, we can deduct the following:

Theorem 3: "Any end point entering the sorting stage is guaranteed to find its two start points below it."

Proof:

Assume end point Ei was detected in the first phase inside PEi. Also assume the two start points connected to Ei are Sj detected in PEj and Sk detected in PEk respectively. From corollary 4 we can deduct the following:

PEj and PEk has to be higher than PEi, i.e. $i > j$ and $i > k$.

Between the first phase and the second phase Ei will be passed to PE(3i) in the tagging stage, Sj will be passed to PE(3j) and Sk will be passed to PE(3k). During the rerouting stage all points migrate from bottom to top, and we can conclude that end point Ei will be collected and passed to the sorting stage after its two start points Sj and Sk are both collected and passed to the sorting stage.

Q.E.D.

4.3.4 Pseudo code of the sorting algorithm

Initially all the PE's are reset at the end of each L consecutive cycles, where L=3N (N is the number of columns/pattern).

```
for PE1 do
    for L cycles do
    if Di carries start point
        then
            if S register is empty
            then
                store Di in S register
            else
                pass point below
        else if  Di carries an end point
            then
                if end-point  is connected  to left edge
                   of start point in S register
                then
                    tag end point with "E"
                    pass the point below;
for all PEi 2 ≤ i ≤ L pardo
    for L cycles do
    during the first half of each cycle do
    if Di carries start point
  then
```

```
    pass data in S register to PE_{i+1}
    store Di in S register
else
    if Di carries an end point
    then if end point has an "E" tag
         then
           if S register contains data
           then
              pass end point below
           else % S register is empty %
              store end point in E register
         else % end point has no "E" tag %
              if end point is searching for start
                 point stored in S register
              then if your E register is full
                   then % conflict rule 2 %
                        store the end point that is
                        connected to left edge of
                        start point stored in S
                        register
                        pass the second end point to
                        lower PE
                   else % no end points stored in E
                                register %
                        store Di in E register
```

```
                    else % end point not searching for start
                              point in S register %
                         if start point in S register is
                            connected   to   right  edge  of
                            incoming end point
                         then
                              generate a swap signal
                              pass end point to PE below
                         else
                              pass end point to PE below;

          during the second half of the cycle do
               if  a swap signal was generated in PEi
               then if  end point  that generated  this signal is
                         looking for start point stored in PEi+1
                    then
                         ignore this swap signal
                    else
                         swap S and E registers of PEi with
                              S and E registers of PEi+1;

% end of sorting algorithm %
```

4.3.5 An example for the sorting algorithm

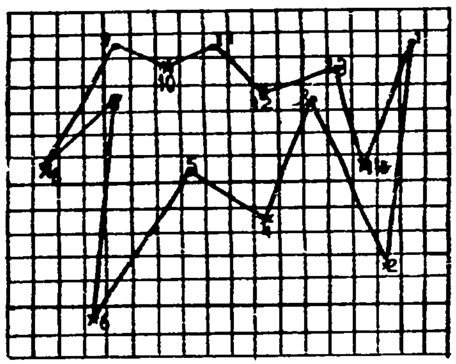
Figure 4.10.a shows a sample pattern and the points extracted from it at the end of the rerouting stage. Figure 4.10.b shows the space/time diagram of the sorting algorithm while processing this pattern.

At cycle 5 PE_1 receives the end point 10. It retrieves the left start point data from this end point and tags the end point with this start point data generating the message (10 --> 9), which means that end point 10 is to be sent below to be stored with start point 9.

At cycle 6 this message reaches PE_3 and the end point 10 is stored with start point 9.

At cycle 8 while the message 12 --> 11 is passing through PE_3 it detects the right start point 13 of end point 12. In this case a swap command is generated causing the contents of PE_3 to be swapped with the contents of PE_4 during cycle 8'.

At cycle 13 the message 8 --> 9 reaches PE_5 and detects that start point 9 has another end point (10) stored with it. According to our algorithm and using the conflict rule discussed earlier, end point 10 will be forwarded to be stored with its second start point (point 11) and end point 8 is stored with start point 9 (since 8 is the end point connected to the left edge of start point 9).



• Start point

x End point

Figure 4.10.a

x ---> y stands for: end point x will imigrate in the column of PEs searching for start point y. Reaching its destination point x will be stored with y.

SW) stands for: A swap command causes PEx and PE(x+1) to swap their data together.

↘ stands for: Edge data is passed to the PE to be stored. If the PE receiving this data already contains another edge data then the old data is passed further down while new arrived data is stored in its place.

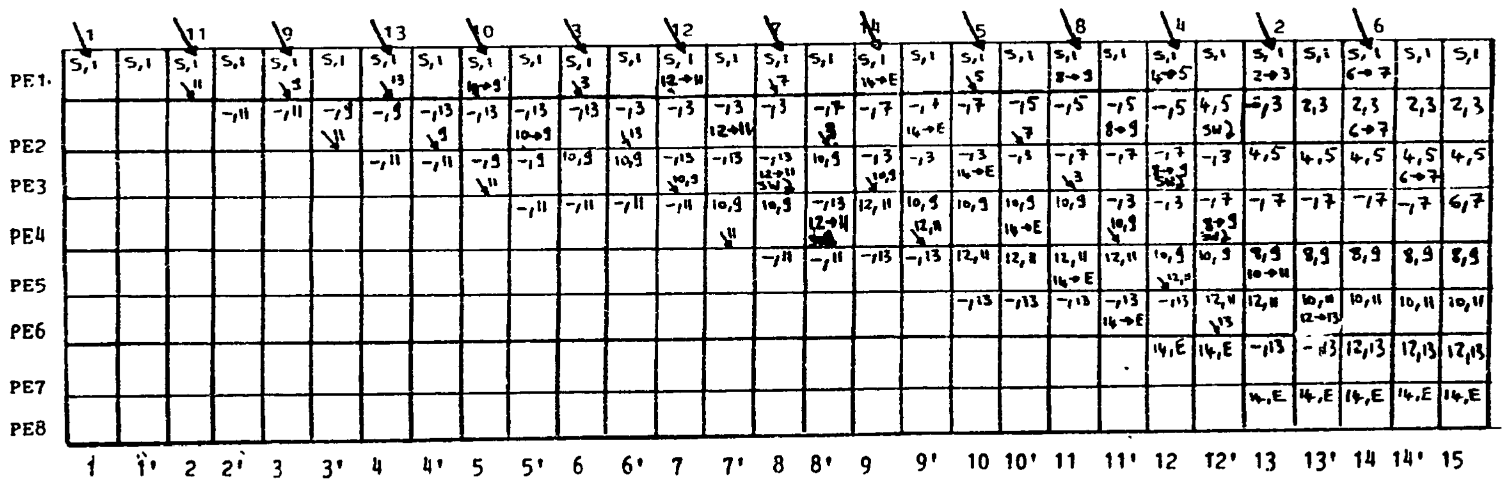


Figure 4.10.b Sorting Stage Example

CHAPTER V

Classification Phase

In this chapter we will discuss the simulation experiment of the HCR system described so far. We will analyze the threshold value selected to smooth the chains, the classification algorithm employed, the database used for classification, and the simulation results.

The simulation was performed on a sequential machine and we simulated all the parallel algorithms described in Chapters III and IV. We conducted the simulation experiment on 1200 samples, 120 sample/numeric digit. The resolution we used for digitizing each sample was 54 rows and 54 columns. Some of the samples have been rotated by 10, 15 , -10 and -15 degrees before being scanned to test the system's sensitivity for rotation. Appendix B shows some of these samples. No preprocessing, noise elimination, or filling has been done on the samples. We only assume that the characters are properly segmented and no overlapping between two patterns was allowed.

The objectives of the simulation are:

(1) Test system behavior on samples of hand written numerals.

(2) Decide on the best threshold values to be used for smoothing the chain of feature points.

(3) Retrieve the chains resulting from each directional scan, (H-scan, V-scan, D-scan), on the 1200 samples. This helped in determining the number of necessary scans needed for correctly classifying the patterns.

(4) Fine-tune the classifier.

The samples were first processed through H-scan and V-scan. We first trained the system on a subset of the 1200 sample set. We gathered all the classes that were generated. Then we started to augment the training subset gradually to see if there will be new classes added. Starting from 800 samples and above we reached a steady state in the number of resulting classes. After processing the whole 1200 sample set there were 31 resulting horizontal classes (table C-1 in appendix C) and 40 resulting vertical classes (table C-2 appendix C).

We also found that the best classification resulted while using a threshold of 3 pixels for smoothing. This means that any edge having a vertical height less than 3 pixels is merged with its predecessor edge in the chain. We had some samples of digits 1 and 7 and some samples of digits 4 and 9 that were confused together, i.e. their H-scan and V-scan resulted in similar compound chains. These confused samples are shown in the following pages (figure 5.1).

In order to eliminate the above confusion we found it necessary to process the samples through a D-scan following which all the confused samples were properly distinguished.

The system performed well with rotated characters (between -15 to 15 degrees), and was quite immune to translation and writer style variation.

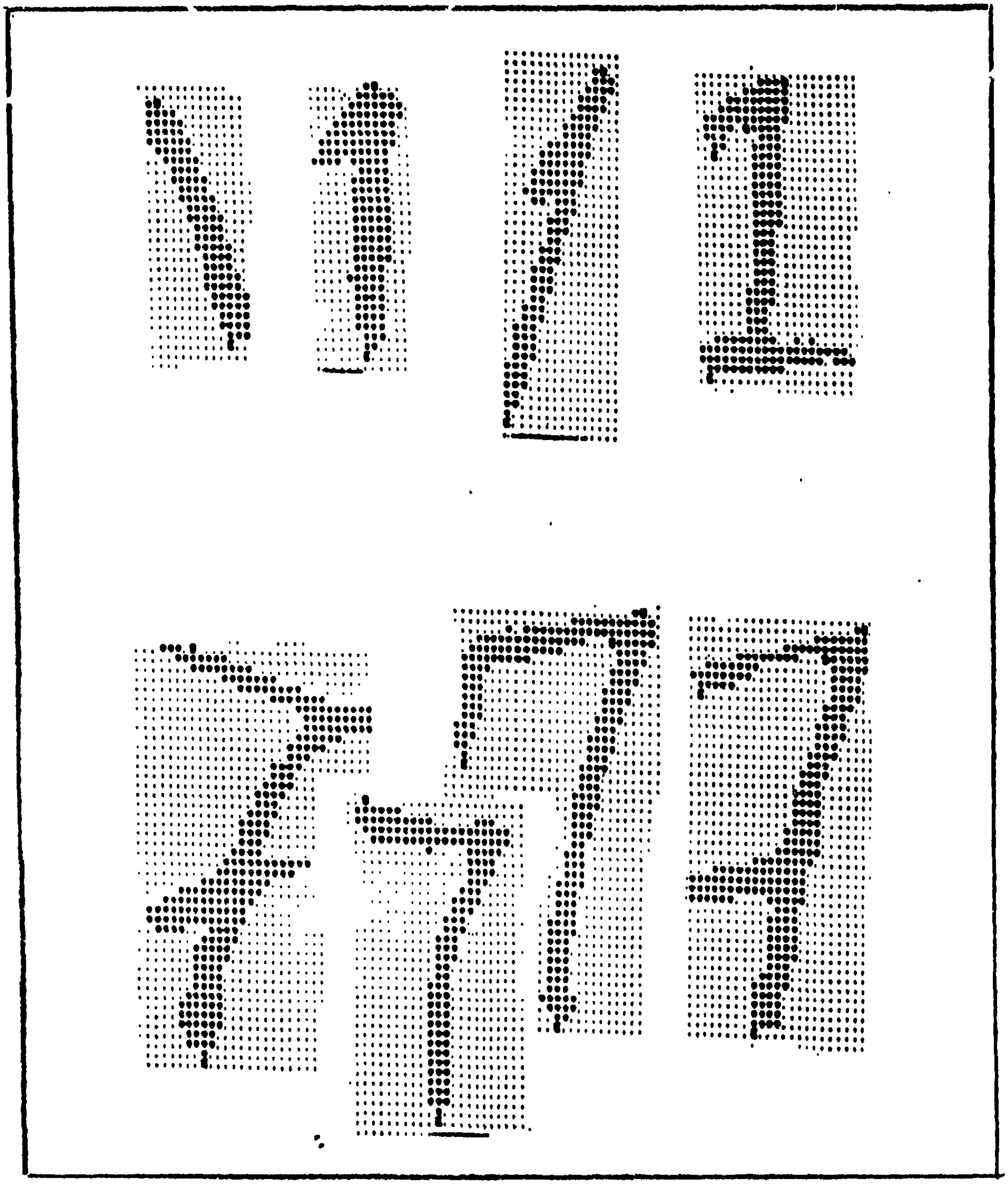


Figure 5.1 Samples of the confused set of numerals

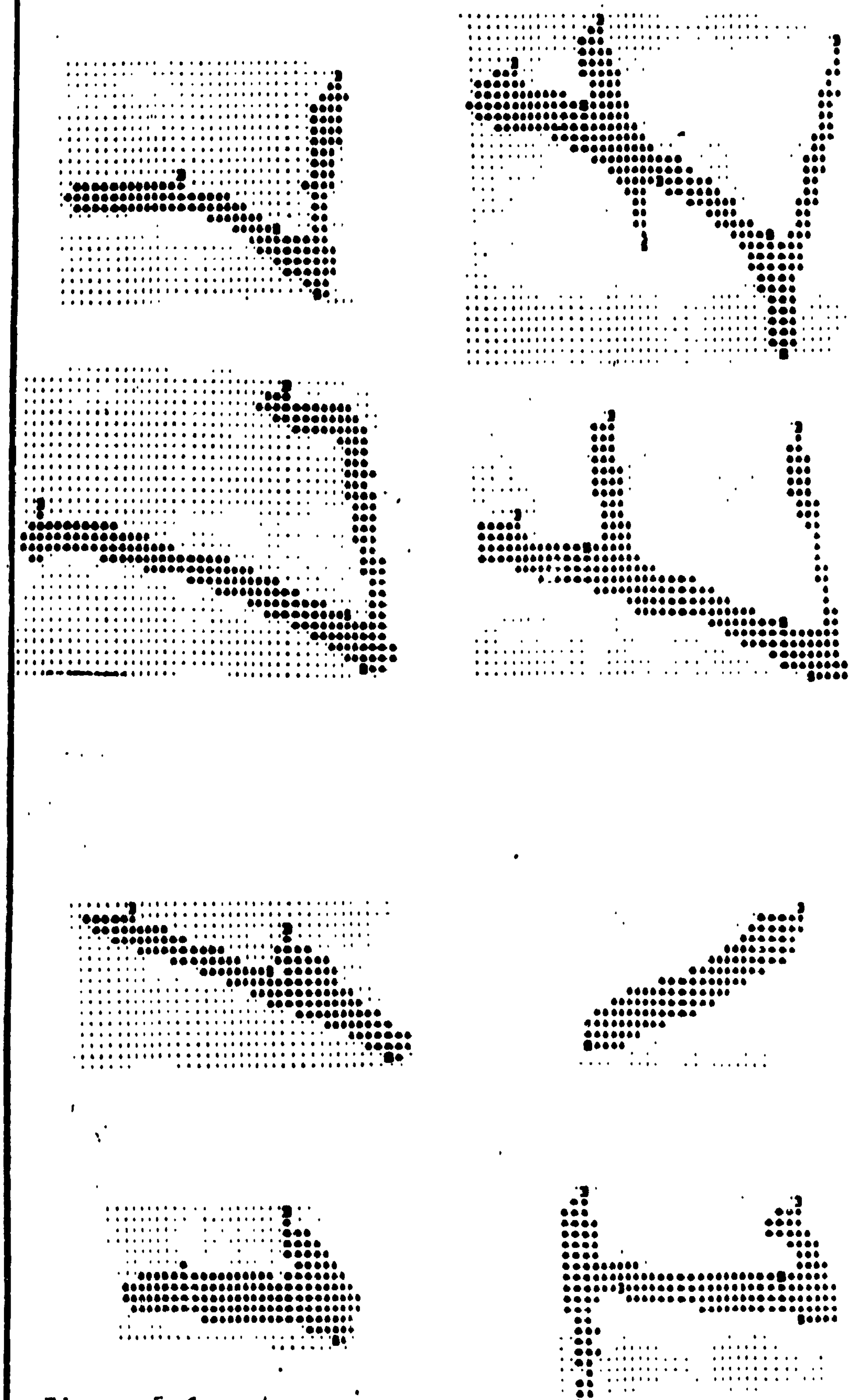


Figure 5.1 cont.

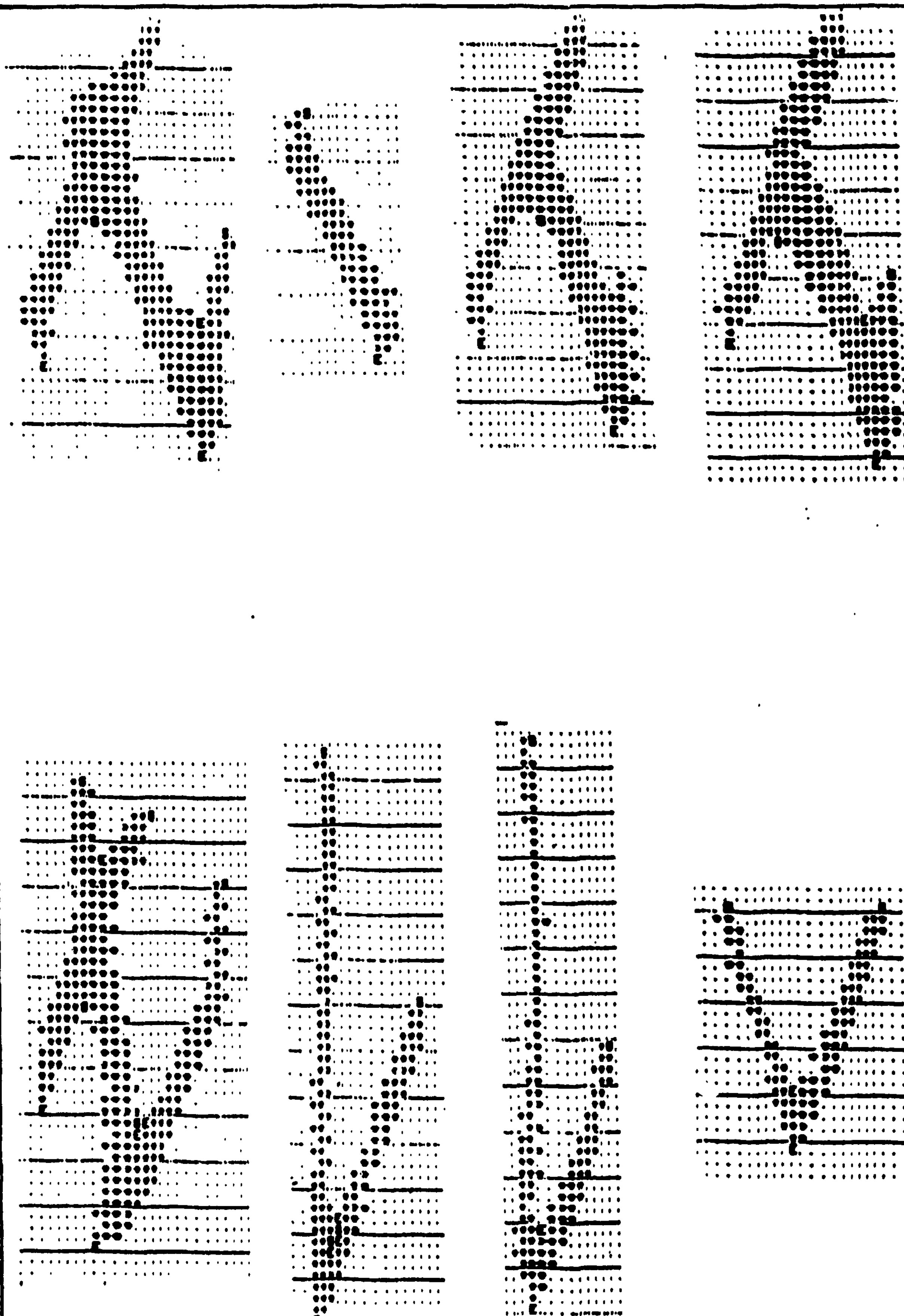


Figure 5.1 cont.

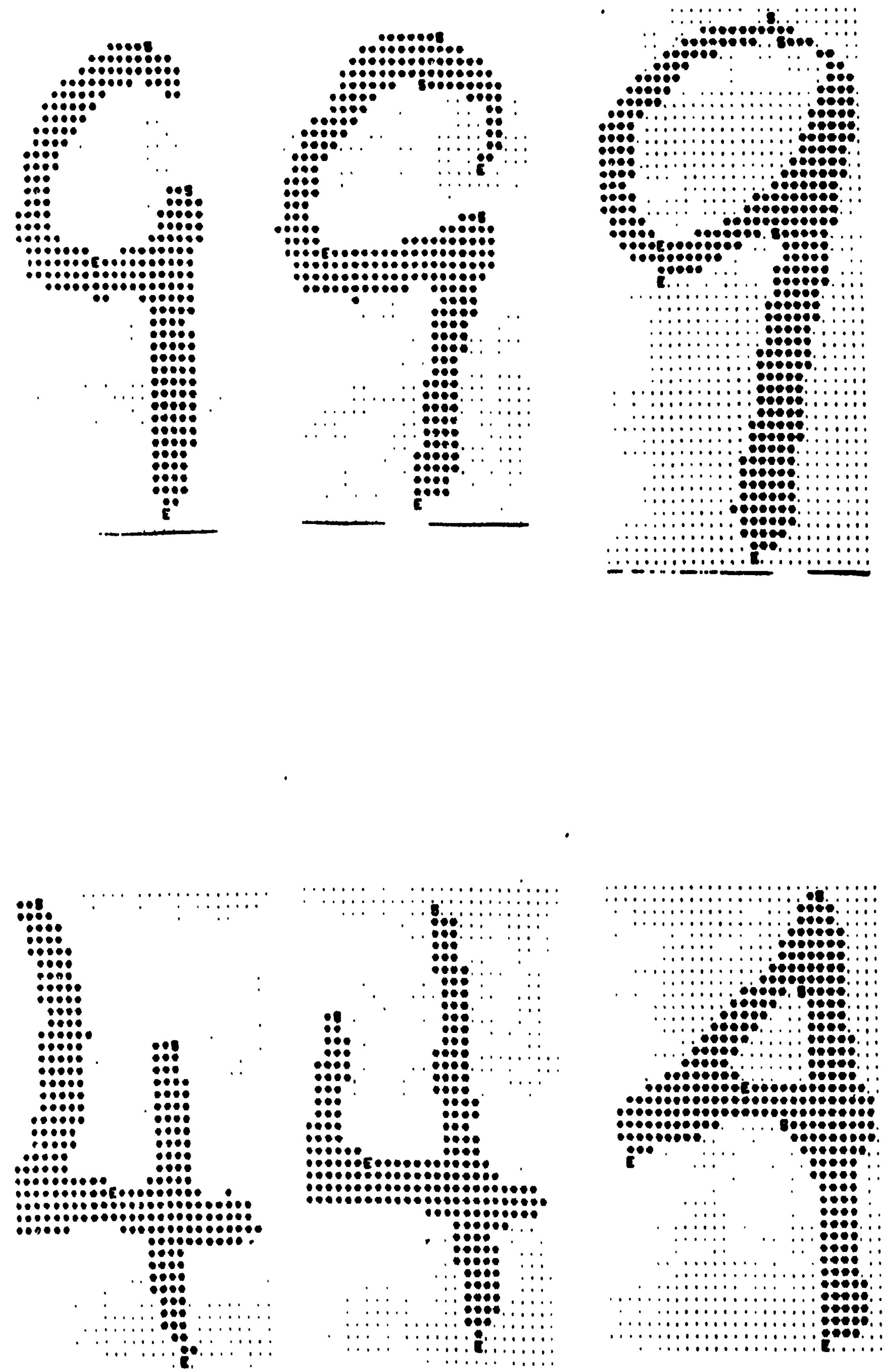


Figure 5.1 cont.

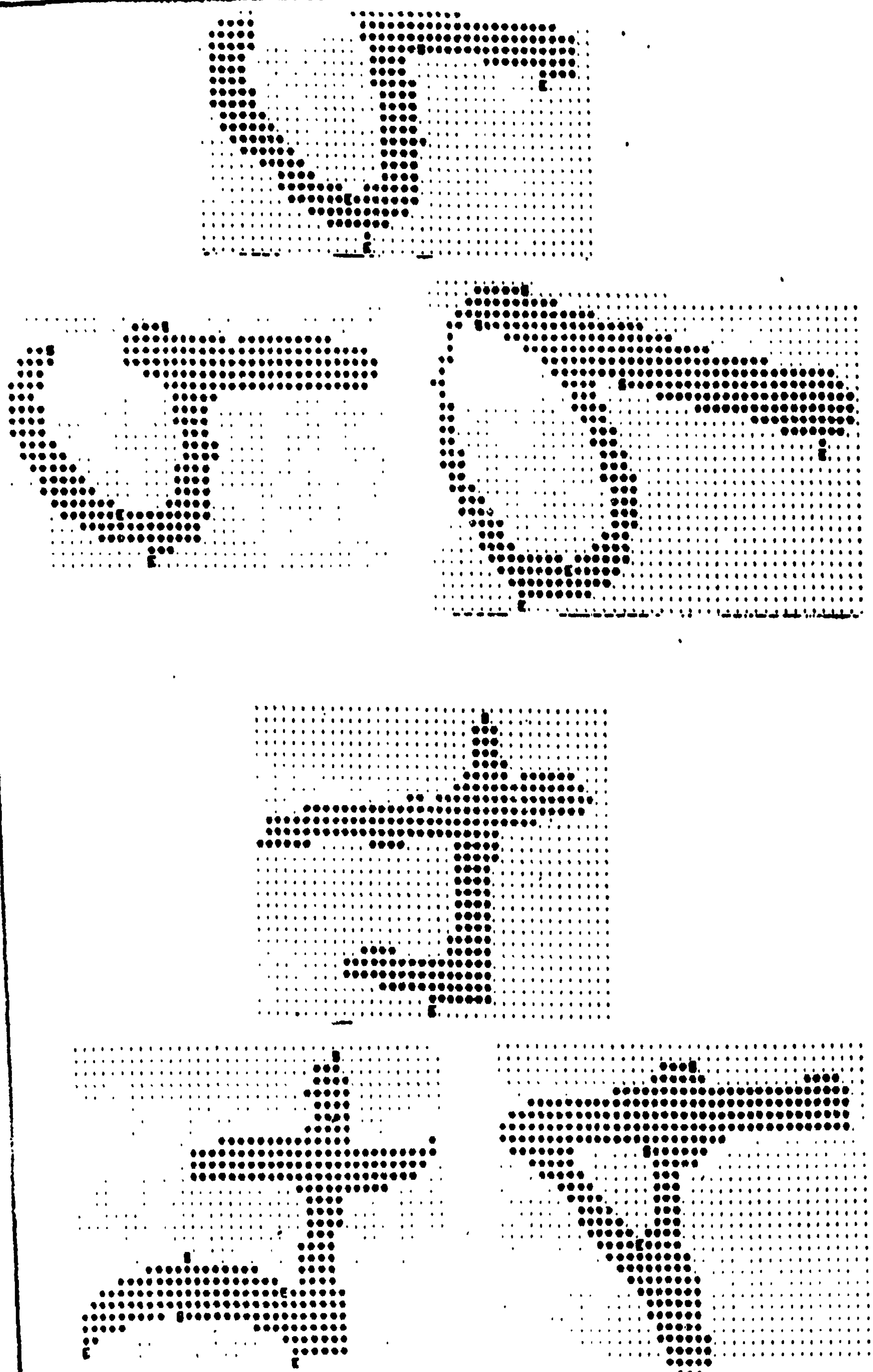


Figure 5.1 cont.

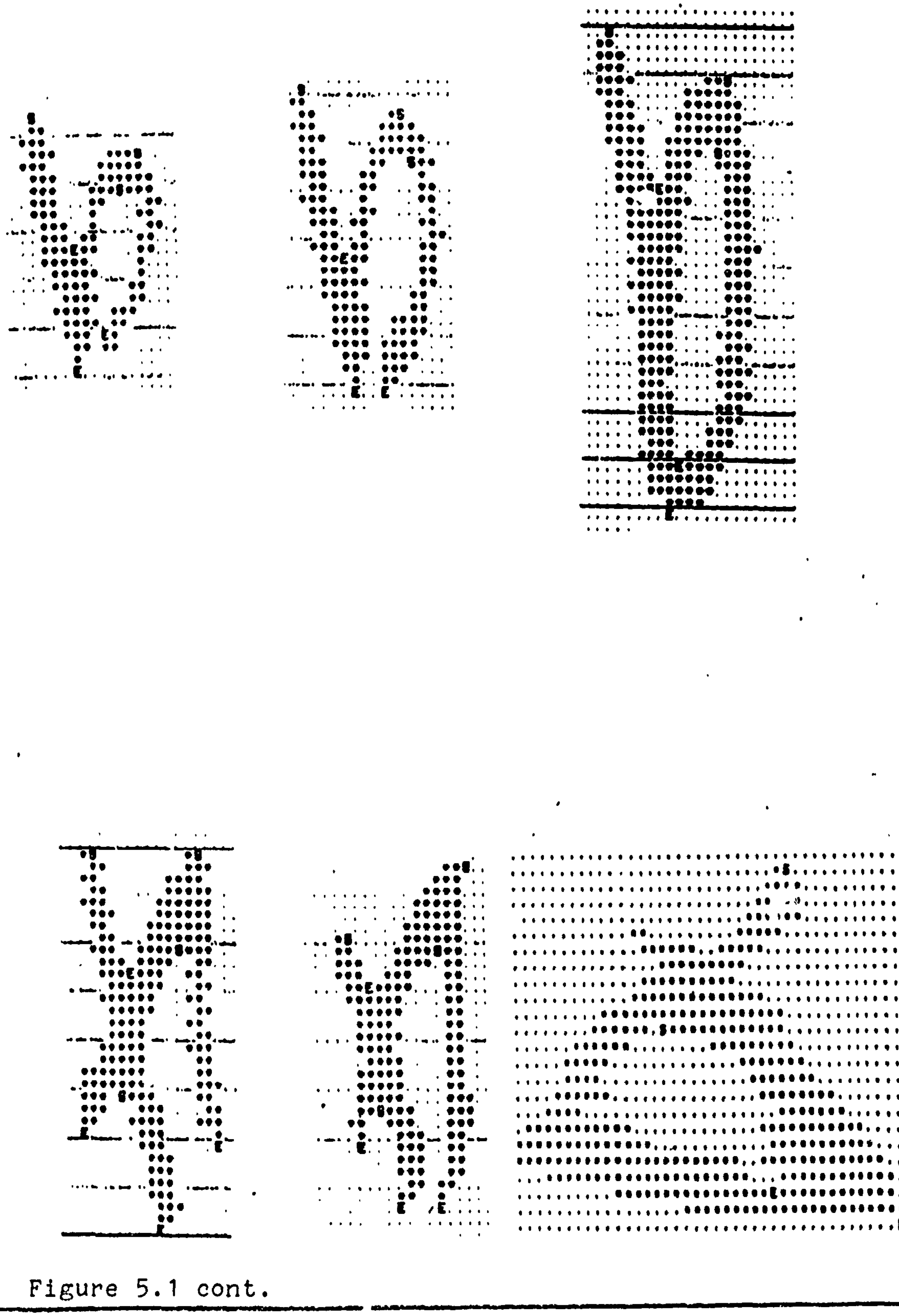


Figure 5.1 cont.

5.1 Smoothing and merging stage

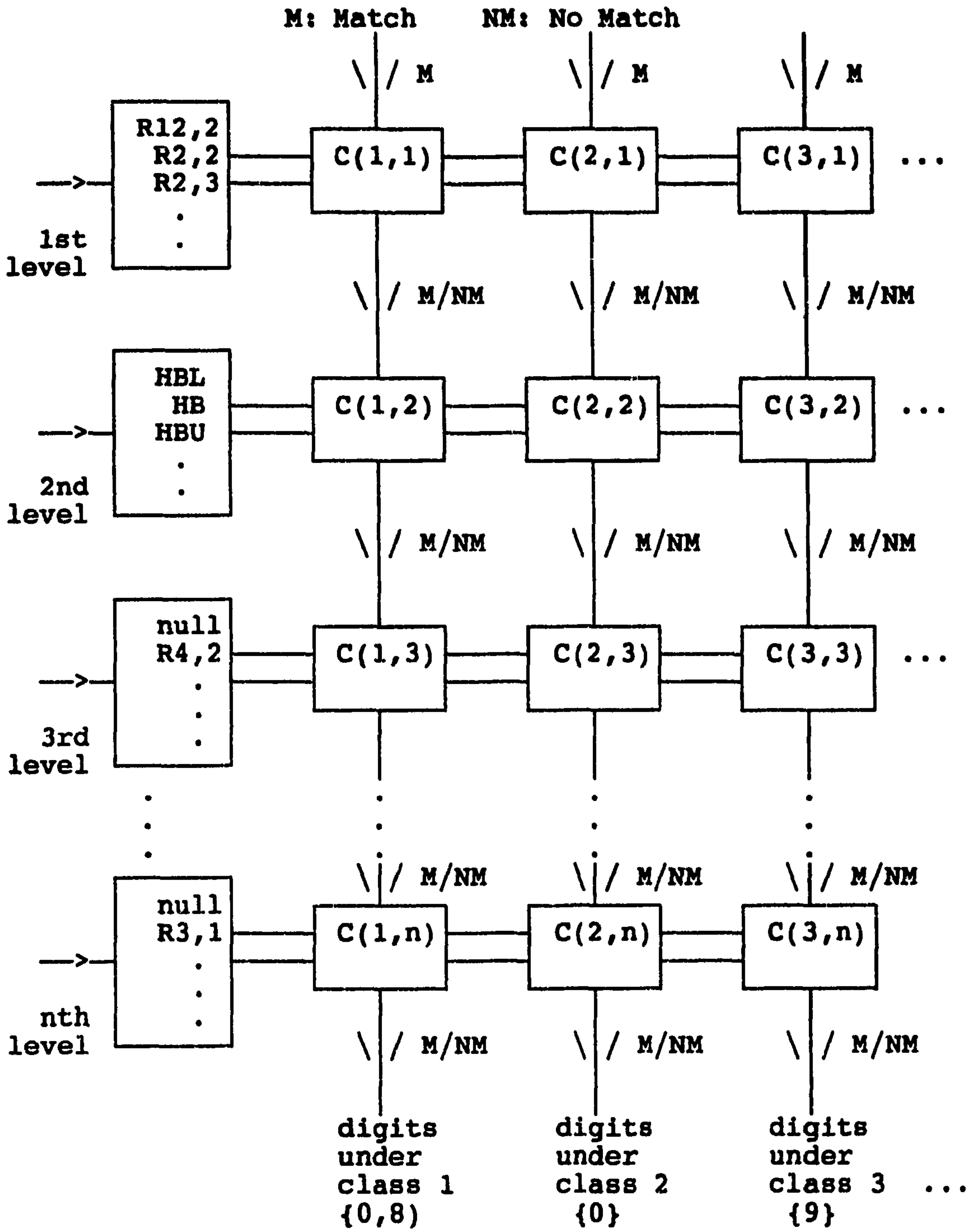
The smoothing is done within a column of PEs in which each PE will delete an edge, if the edge is less than the threshold. The gap generated by the deletion can be eliminated by a shift left instruction. This is the same technique used in database machines when deleting an entity [3,23,25,26,30]. In smoothing, we can also detect if a chain encloses a hole or not. This is simply done by examining the first start point. If it has an R_7 relation then a hole is detected. The minimum size of the rectangle that encloses this hole is then obtained by detecting the left upper-most and right lower-most feature points in the chain. The chain that forms the hole is then replaced with a hole-feature enclosed by the detected rectangle. All the remaining chains are then merged with the extracted hole-features to form a compound list of chain features that is passed to the classification stage. The tables in appendix C are the final compound lists of chain features extracted from the 1200 samples during the simulation for the H-scan and V-scan.

5.2 The classification stage

The compound chain features of the training set in each character class in the H-scan is used to program the PLA used for H-scan classification. The same thing is done for the V-scan and the D-scan. Let us assume that class C_i of the H-scan consists of the following compound chain: R_1, R_4, R_1, R_2 and HBL, where HBL stands for (Big Hole on the Left).

We say feature R_1 in the compound chain is in level 1, feature R_4 in level 2, feature R_1 in level 3 and so on. The PLA is set up such that vertical lines represent different classes and horizonal rows correspond to levels. The horizontal lines emerging from the first level represent all the possible relations that may be extracted as the first feature in any processed pattern. If a certain class contains this feature in level one, then the corresponding horizontal line will be connected to the vertical line representing this class by an AND-gate. In our example the vertical line representing class C_i will be connected to the horizontal line emerging from the first level and representing feature R_1 through a gate. The output of this gate will be the new C_i's vertical line entering level 2. The horizontal line emerging from level 2 and representing our second feature R_4 will then be connected to this new vertical line. The output of this gate will be the new C_i vertical line for level 3 and so on.

Similar programming is done for the rest of the classes. Figure 5.2 shows a portion of the PLA used for H-scan classification.



Each square acts as a logic gate between the vertical signal entering it and one of the horizontal signals. If those two signals are high then a Match signal is generated, otherwise a No Match signal is generated

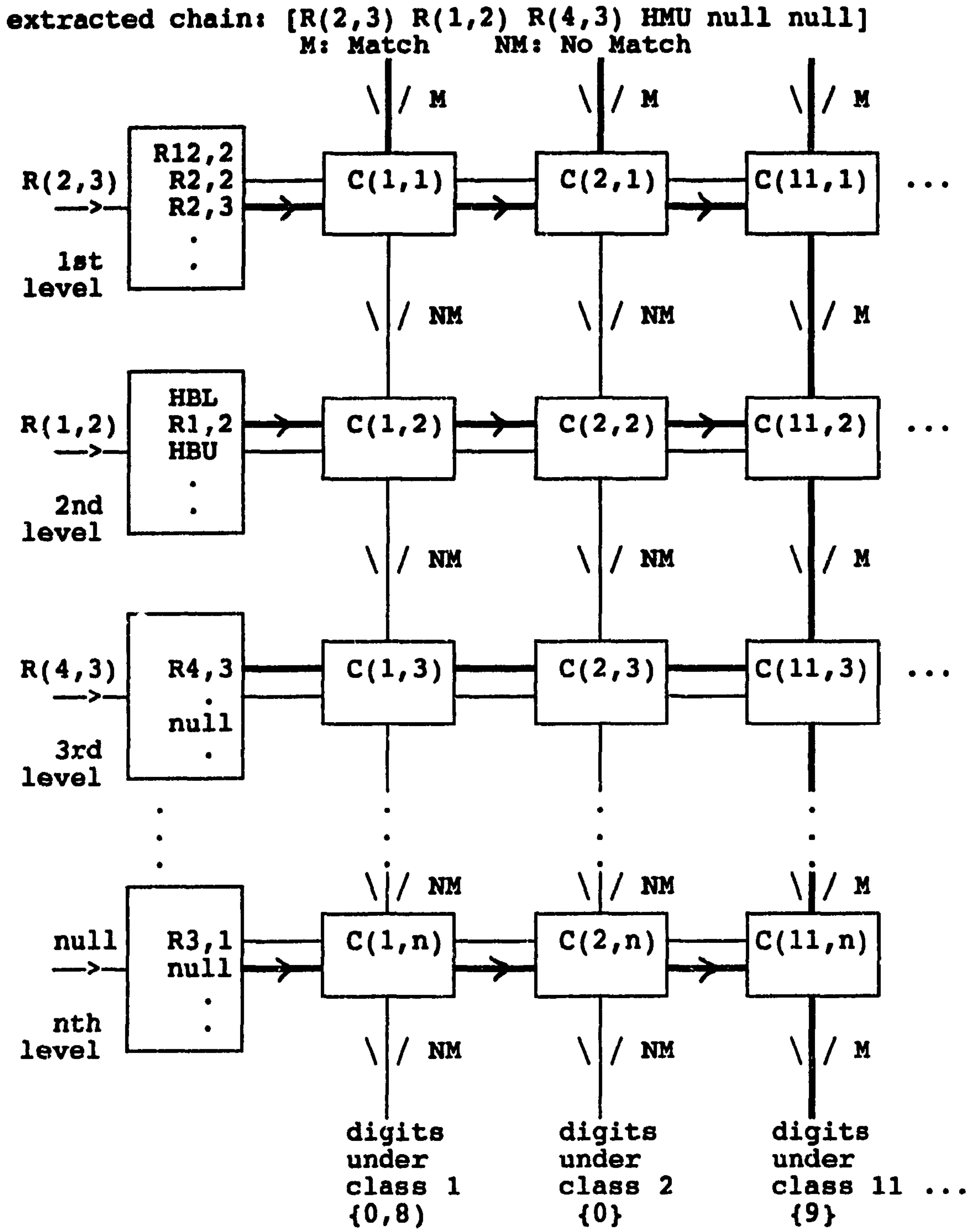
Figure 5.2
Horizontal Classifier Architecture

Now let us see how the classification is done. Upon extracting the compound chain from the input pattern we will pass the features in the chain to the corresponding levels of the PLA. The first feature will enter level 1, the second enters level 2 and so on. The vertical lines representing all the classes will be initially set to 1 (Match). The feature entering each level will set the corresponding horizontal line emerging from that level to 1. The gate whose two inputs are set to 1 will correspondingly set its output to 1. At the last level only one vertical line (class) will be set to 1. Each class points to a set of character numerals (digits) to which the processed character may belong.

The above sequence is performed in parallel for the H-scan, V-scan and the D-scan and the set of characters of each directional scan is extracted. The intersection of these three sets will give the probable numeral(s). A unique result is detected if and only if the above intersection yields one numeral as its result. Table C.1 included in appendix C shows the compound chain of each class for the H-scan. This table is used to program the H-scan PLA classifier. The set of all possible numerals falling in each class is also shown in the table.

5.2.1 An example for the classification algorithm

Figure 5.3 shows the compound chain extracted from an H-scan on a pattern and the set of numerals to which the pattern may be classified. As shown the gate between C_{11} and $R_{2,3}$ in level 1 is set. Then the gate between C_{11} and $R_{1,2}$ in level 2 is set followed by C_{11} and $R_{4,3}$ in level 3, C_{11} and HMU in level 4, C_{11} and - (the empty feature) in level 5 and finally followed by C_{11} and - in level 6. The setting of C_{11} at level six leads to the extraction of the set that contains only one numeral, {9}, as the only candidate to which the pattern will be classified. Notice that only the vertical line representing C_{11} will be set to 1.



Double lines shows the vertical/horizontal set signals. The result is that gate C(11,n) is set and a Match signals is generated leading to the input character to be uniquely classified as digit number 9.

Figure 5.3
Classification Example

CHAPTER VI

System Performance

6.1 The overall system performance

Figure 6.1 shows the complete HCR system with all its phases and different stages. From the 1200 samples processed during the simulation we found that no more than 4 chains/sample occurs. So only 4 columns for sorting these chains and 4 other columns for smoothing/merging are shown in Figure 6.1.

To calculate the maximum delay time between the first arrival of a pattern matrix and the final output, the following remarks are applicable: Number of cycles needed in edge extraction and unskewing stages is (2M+N), as determined in chapter III. Then N cycles are needed for each succeeding stage. Thus the total system delay time is (2M + N) + 6(N) = 2M + 7N.

The total system initialization time (flush time: the time between the arrival of the first pixel at phase I and the arrival of the resultant processed data at the last stage) is equal to 2M + 6N. After the system is initialized, i.e. after 2M + 6N cycles, one pattern will be classified every N cycles.

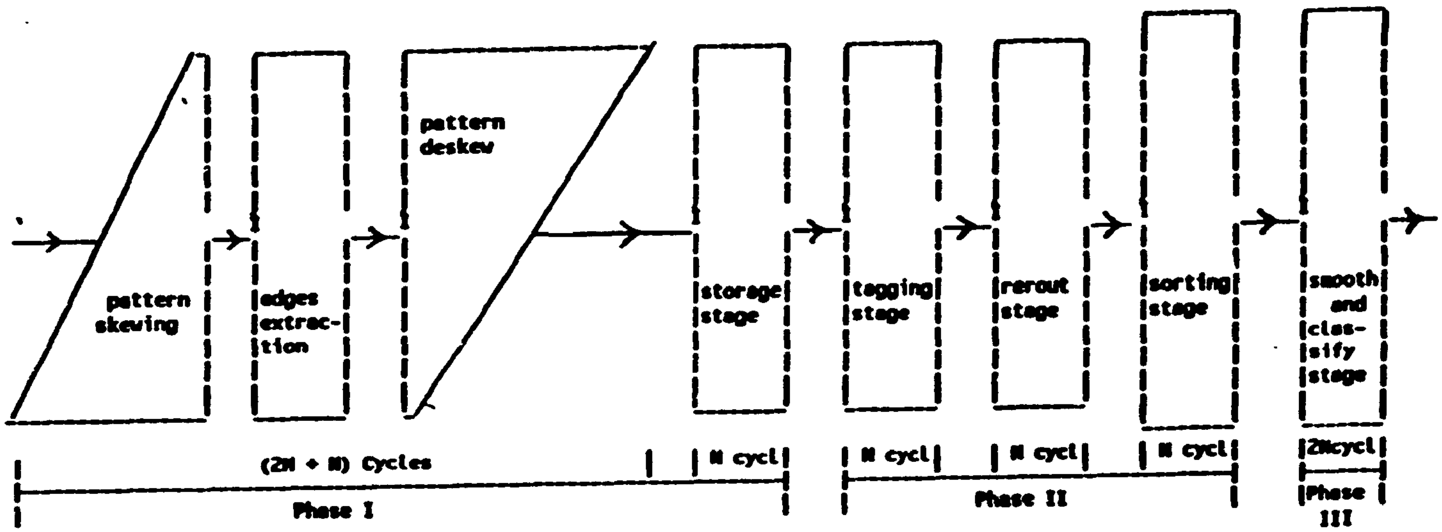
Each of the edge extraction stage and the storing stage uses M(PEs). Each of the tagging stage and the rerouting stage uses 3M(PEs).

The number of PEs needed for each of the sorting stage and the smoothing stage is at most 3M(PEs). Thus the maximum number of PEs used by the system for scan is 14M. The maximum number of PEs used for the three directional scan is 42M. Since all the PEs in each stage will be processing a constant number of pixels each cycle, the area of PEs used in each stage will be constant independent of M and N. The PLAs used in the classification are very simple, cost effective and the length of horizontal and vertical lines used will be constant. This is ideal for VLSI implementation.

6.2 Further work

The system has performed well in our simulation experiment on 1200 numeric samples. The next step worth examining is to expand the system to recognize alphanumerics. We expect no major modification or limitation in the current system in order for it to be adaptable for alphanumeric recognition. However, we may need to expand the number of directional scans in order to accommodate more (basically similar) alphanumeric patterns. This expansion should improve recognition as the geometrical or shape similarity between different classes will diminish with multiple scans.

Another enhancement to our system can be considered by including the curvatures (concave, convex, or straight) of the edges during extraction in the first stage. This enhancement could reduce the number of directional scans needed.



Delay Time= Total number of cycles elapsed since first pixel of a pattern enters the system untill the hole pattern is classified = [2M + 7N]

After [2M + 6N] (Flush Time) a complete pattern will be classified each N cycles

Where M: number of rows in pattern
N: number of columns in the pattern

Figure 6.1
Complete System Diagram

6.3 Conclusion

This thesis has presented parallel algorithms to extract shape features of handwritten numerical patterns, to sort these features, and to classify these patterns. The recognition of these patterns is based on the approach proposed by Ahmed and Suen [1,2] with some modifications. The start and end points of edges in a pattern and the relations between a pair of edges meeting at a start point or at an end point are first extracted by the parallel algorithm of chapter III. The chains formed by these edges are then extracted and sorted by the parallel algorithms of chapter IV. The simulation and classification are presented in chapter V. An experiment was conducted on 1200 samples, 120 for each of the 10 numerals. It has been observed that using the features extracted during H-scan, V-scan and D-scan all the handwritten numerals are correctly classified. Moreover, these numerals can be correctly recognized if they are rotated by an angle between -15 and +15 degrees.

The system can be expanded to recognize handwritten alphanumerics by increasing the number of directional scans used. Finally, all the parallel algorithms given can easily be implemented on VLSI chips using current technology.

APPENDIX A

REFERENCES

[1] P. Ahmed, "Computer Recognition of Totally Unconstrained Handwritten Zip Codes", Ph.D. Thesis, Dept of Computer Science, Concordia University, Montreal, Canada, July 1986.

[2] P. Ahmed, and C.Y. Suen, "Edge Classification and Extraction of Shape Features", 7th Intl. Conf. on Pattern Recognition, 1984, pp. 593-596.

[3] M.J. Atallah, "A Generalized Dictionary Machine For VLSI", Trans. on Computers, Vol. C-34, Feb 85, pp. 151-155.

[4] T.W. Calvert, "Nonorthogonal Projections for Feature Extraction in Pattern Recognition", IEEE Trans. on Computers, Vol. C-19, 1970, pp. 447-452.

[5] Heng-Da cheng and King-Sun Fu, "VLSI Architectures for Dynamic Time-Wrap Recognition of Handwritten Symbols", IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-34, NO 3, pp. 603-613.

[6] Heng-Da Cheng, W. Lin, and King-Sun Fu, "Space-Time Domain Expansion Approach to VLSI and its Application to Hierarchical Scene Matching", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No 3, May 85, pp. 306-319.

[7] B.V. Dassarathy and K.P.B. Kumar, "Chitra: Cognitive Handprinted Input-Trained System for Recognition of Alphanumeric Characters", Intl. J. Computer Inf. Sci, Vol. 7, 1978, pp. 253-282.

[8] Ronald Davis, and D. Thomas, "Systolic Array Chip Matches The Pace of High Speed Processing", Elec. Design, Oct 31, 1984, pp. 207-218.

[9] B. Duerr, W. Haettich, H. Tropf, and G. Winkler, "A Combination of Statistical and Syntactical Pattern Recognition Applied to Classification of Unconstrained Handwritten Numerals", Pattern Recognition, Vol. 12, pp. 189-199.

[10] M.J. Duff, and S. Levialdi, "Languages And Architectures for Image Processing", London; Toronto:Academic Press, 1981.

[11] Michael P. Ekstrom, "Digital Image Processing Techniques", New York:Academic Press, 1984.

[12] King-Sun Fu, "Special Computer Architectures For Pattern Processing", Boca Raton, Fla:CRC Press, c 1982.

[13] H. Genchi, K. Mori, S. Watanabe, and S. Katsuragi, "Recognition of Handwritten Numerical Characters for Automatic Letter Sorting", Proc. IEEE, Vol. 56, No 8, Aug 68, pp. 1292-1301.

[14] Wyndham Hanaway, and G. Shea, "Handling Real-Time Images Comes Naturally to Systolic Array Chip", Elec. Design, Nov 15, 1984, pp.289-300.

[15] Kai Hwang, and F.A. Briggs, "Computer Architecture and Parallel Processing", McGraw Hill, 1987.

[16] C.C. Kwan, L. Pang, and C.Y. Suen, "A Comparative Study of Some Recognition Algorithms in Character Recognition", Proc. Intl. Conf. on Cybernetics, 1979, pp. 530-535.

[17] S.K. Kwon, and D.C. Lai, "Recognition Experiments with Handprinted Numerals", Pattern recognition and Artificial Intelligence, 1976, pp. 74-83.

[18] M.T.Y. Lai, and C.Y. Suen, "Automatic Recognition of Characters by Fourier Descriptors and Boundary Line Encodings", Pattern recognition, Vol. 14, Nos 1-6, 1981, pp. 383-393.

[19] H.F. Li, R. Jayakumar, D. Pao, and M.Y. Ghabrial, "Systolic Architectures for Structural Feature Extraction Using Curve Tracing and Hough Transform" to appear in Computer Vision and Shape Recognition, A. Krzyzak, T. Kasvand, and C.Y. Suen (eds.), World Scientific Publishing Co., Singapore, 1989, pp. 153-182.

[20] H.F. Li, R. Jayakumar, and M.Y. Ghabrial, "Parallel Algorithms for Recognizing Handwritten Characters Using Shape Features", Pattern Recognition Journal, Vol. 22, No. 6, 1989, pp. 641-652.

[21] H.F. Li, M.Y. Ghabrial, and R. Jayakumar, "Parallel Algorithms for Extracting Shape Features of Handwritten Characters:, Proc. Vision Interface '88 conf., Edmonton, June 1988, pp. 80-85.

[22] Lionel, and A.K. Jain, "A VLSI Architecture for Pattern Clustering", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No 1, Jan 85, pp. 80-89.

[23] T.A. Ottman, "A Dictionary Machine For VLSI", Trans. on Computers, Vol. C-31, Sep 82, pp. 892-897.

[24] W. Porod, and D.K. Ferry, "Pattern Recognition in Highly-Integrated Circuits", Pattern Recognition, Vol. 18, No 2, 1985, pp. 179-189.

[25] A.M. Scwartz, "Dictionary Machines on Cube-Class networks", Trans. on Computers, Vol. C-36, Jan 87, pp. 100-105.

[26] H., Scmeck, "Dictionary Machines for different models of VLSI", Trans. on Computers, Vol. C-34, May 85, pp. 472-475.

[27] M. Shridhar, and A. Badreldin, "High Accuracy Character Recognition Algorithms Using Fourier and Topological Descriptors", Pattern Recognition, Vol. 17, No 5, 1984, pp. 515-524.

[28] H.J. Siegel, "A Model of SIMD Machines and a Comparison of Various Interconnection Networks", IEEE Trans. on Computers, Vol. C-28, No 12, Dec 1979, pp. 907-917.

[29] W.W. Smith, and P. Sullivan, "Systolic Array Chip Recognizes Visual Pattern Quicker than a Wink", Elec. Design, Nov 29, 1984, pp. 257-266.

[30] A.K. Somani, "An efficient Unsorted VLSI Dictionary Machine", Trans. on Computers, Vol. C-34, Sep 85, pp. 841-851.

[31] C.Y. Suen, "Distinctive Features in Automatic Recognition of Handprinted Characters", Signal Processing, Vol. 4, 1982, pp. 193-207.

[32] C.Y. Suen, M. Berthod, and Shunji Mori, "Automatic Recognition of Handprinted Characters, The State of the Art", Proc. IEEE, Vol. 68, No 4, Apr 80, pp. 469-485.

[33] G.T. Toussaint, and Robert W. Donaldson, "Algorithms for Recognizing Contour Traced Handprinted Characters", IEEE Trans. on Computers, June 70, pp. 541-546.

[34] S. Wakabayashi, T. Kikuno, N. Yoshida, and T. Fujii, "A Programmable Pattern Matching Machine for High Speed Recognition of Regular Sets", Trans. of IEEE of Japan, Vol. E67, No 7, July 84, pp. 363-370.

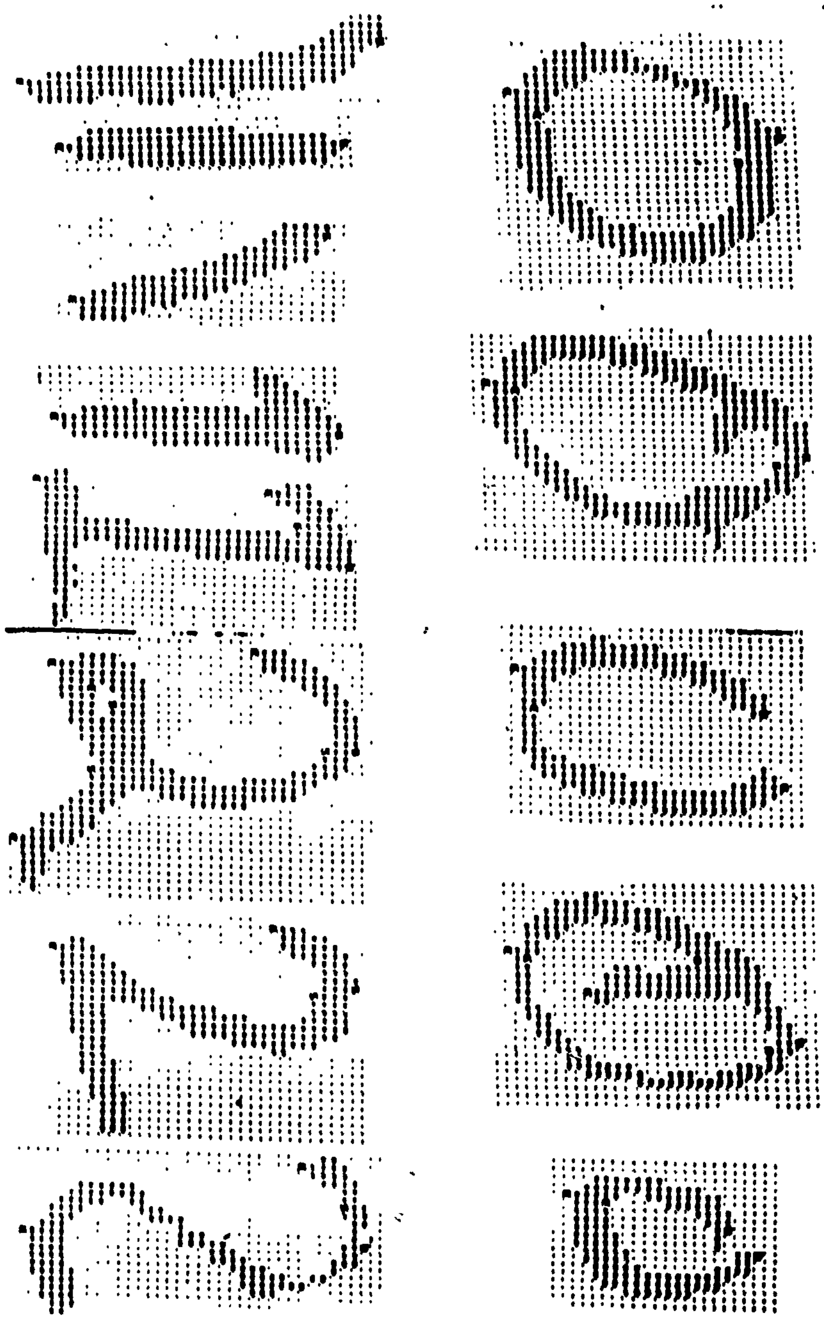
[35] A.H. Watt, and R.L. Burle, "Recognition of Handprinted Numerals Reduced to Graph-Representation Form", Pattern Recognition I, Session 1, pp. 322-332.

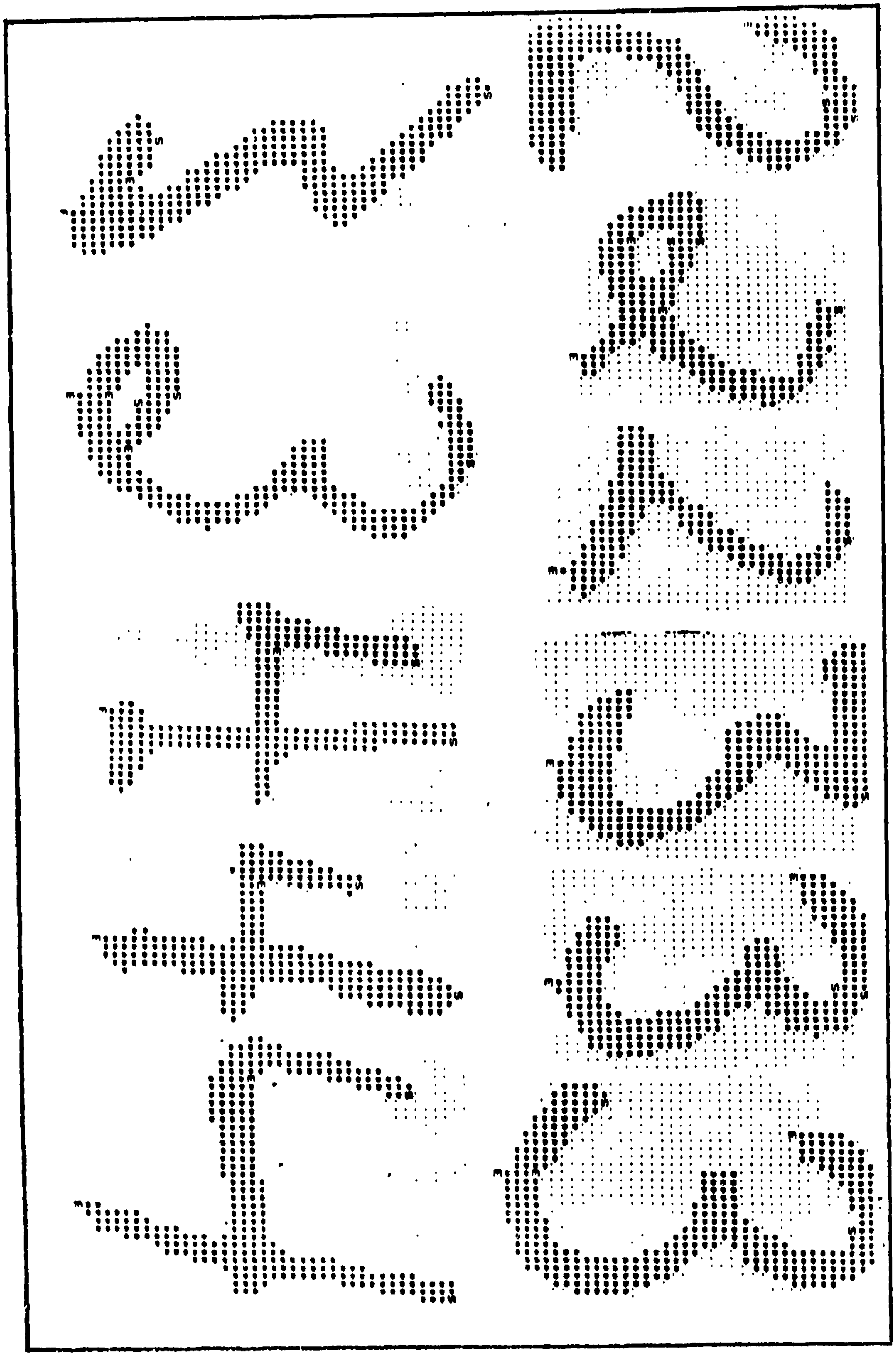
[36] S. Yalamanchili, and J.K. Aggarwal, "A System Organization for Parallel Image Processing", Pattern Recognition, Vol. 18, No 1, 1985, pp. 17-29.

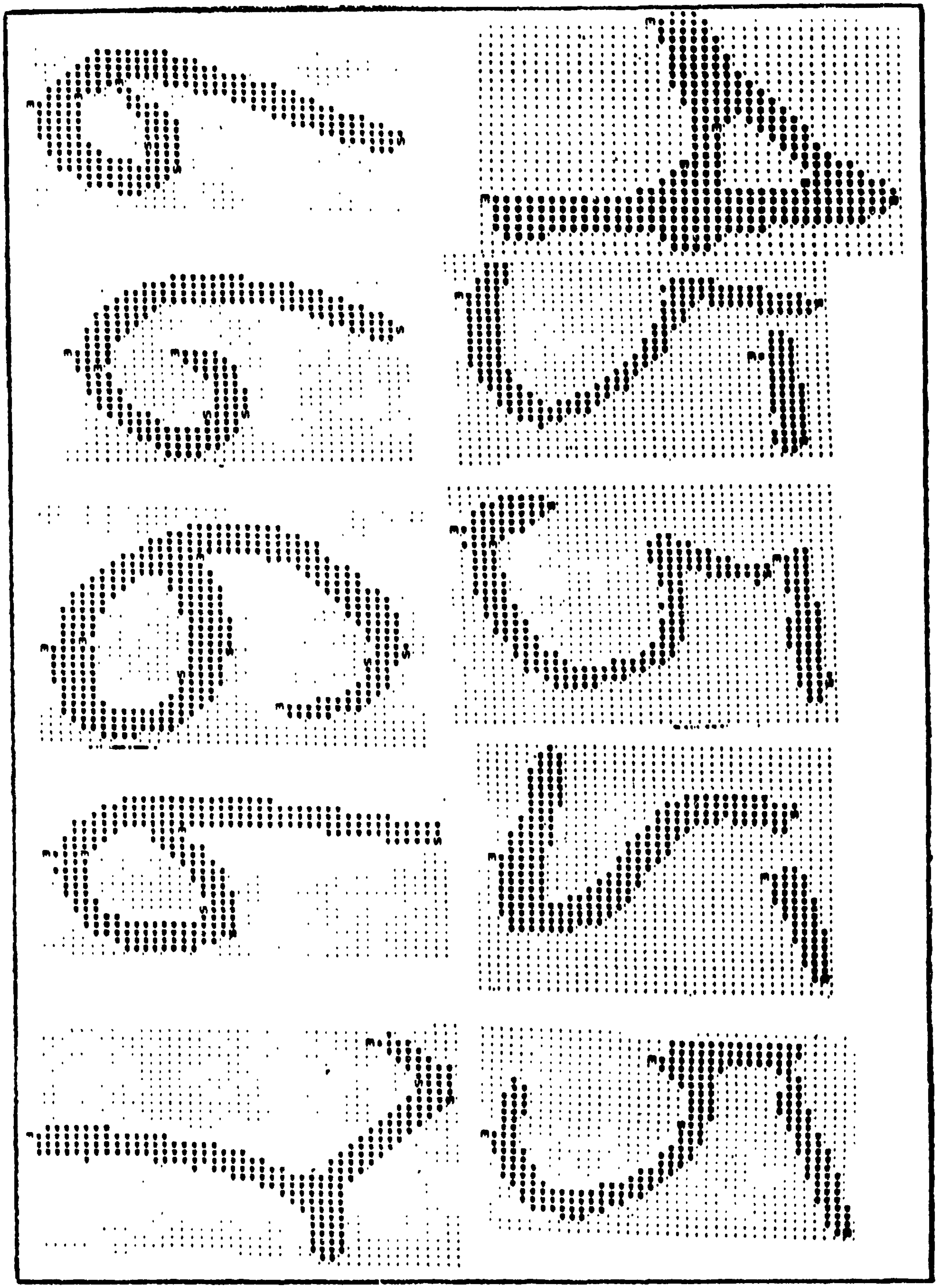
Appendix B

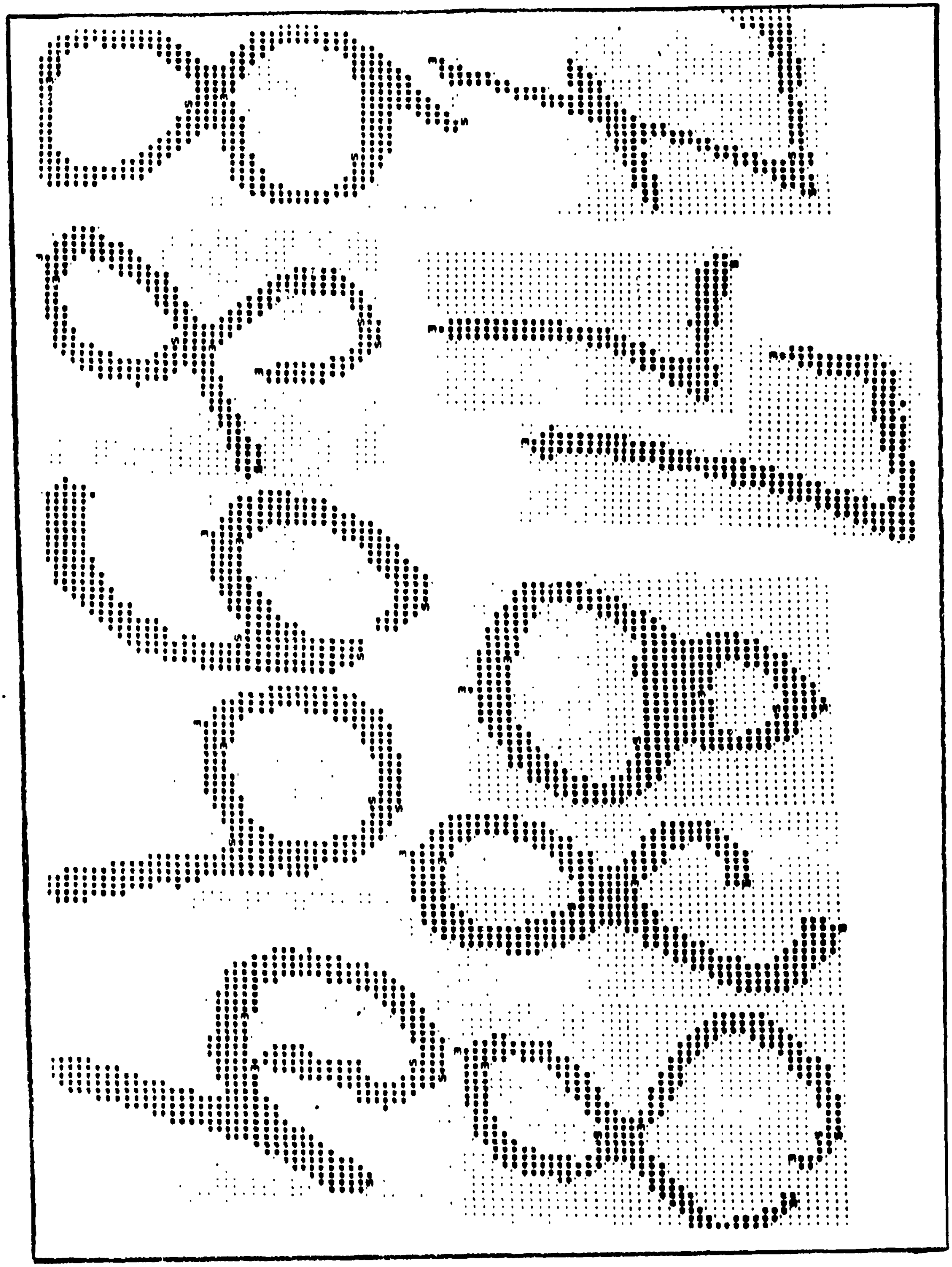
Samples of Digitized Numerals

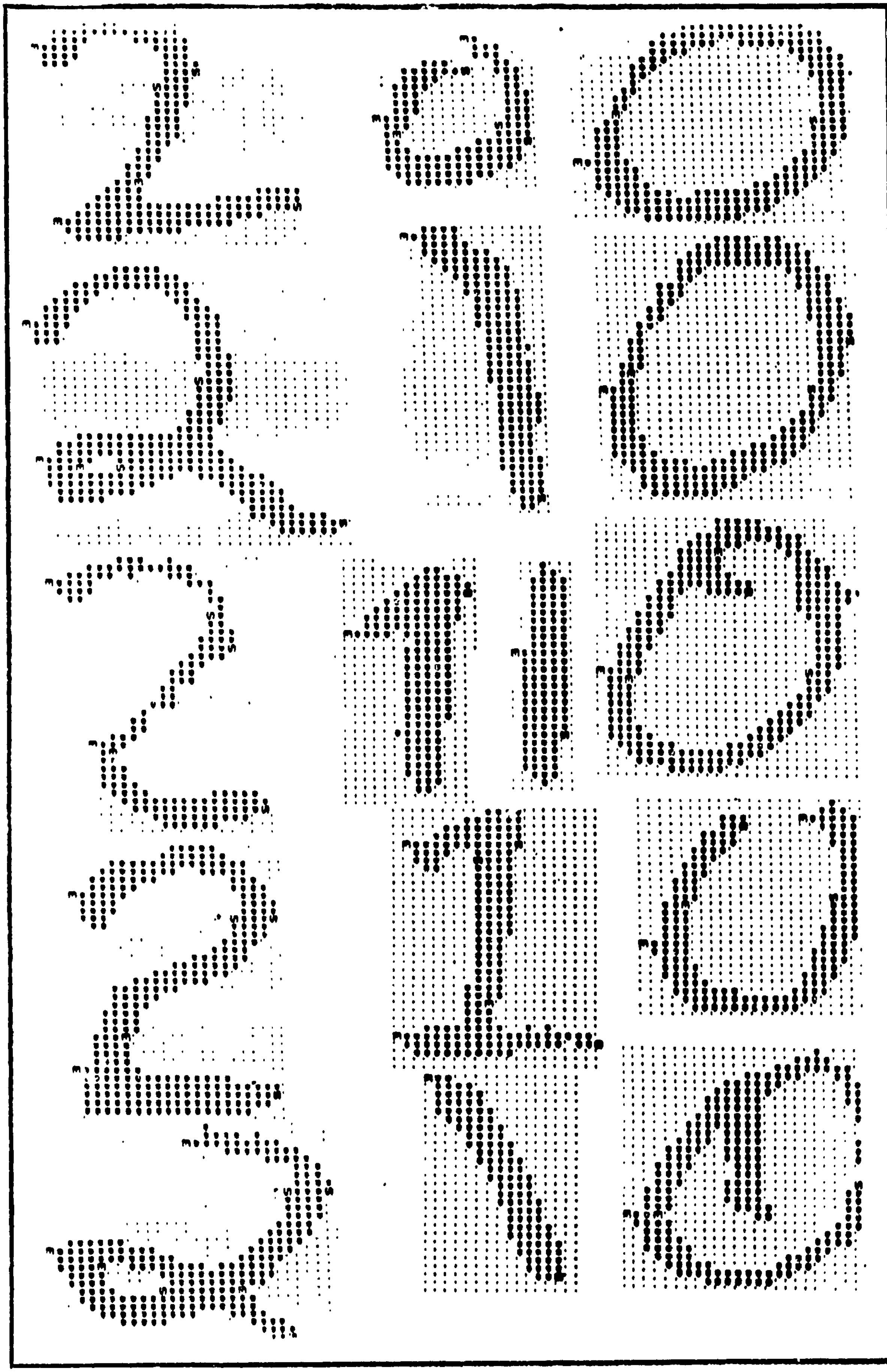
The following figures show samples of the digitized numerals used during the learning and test phases.

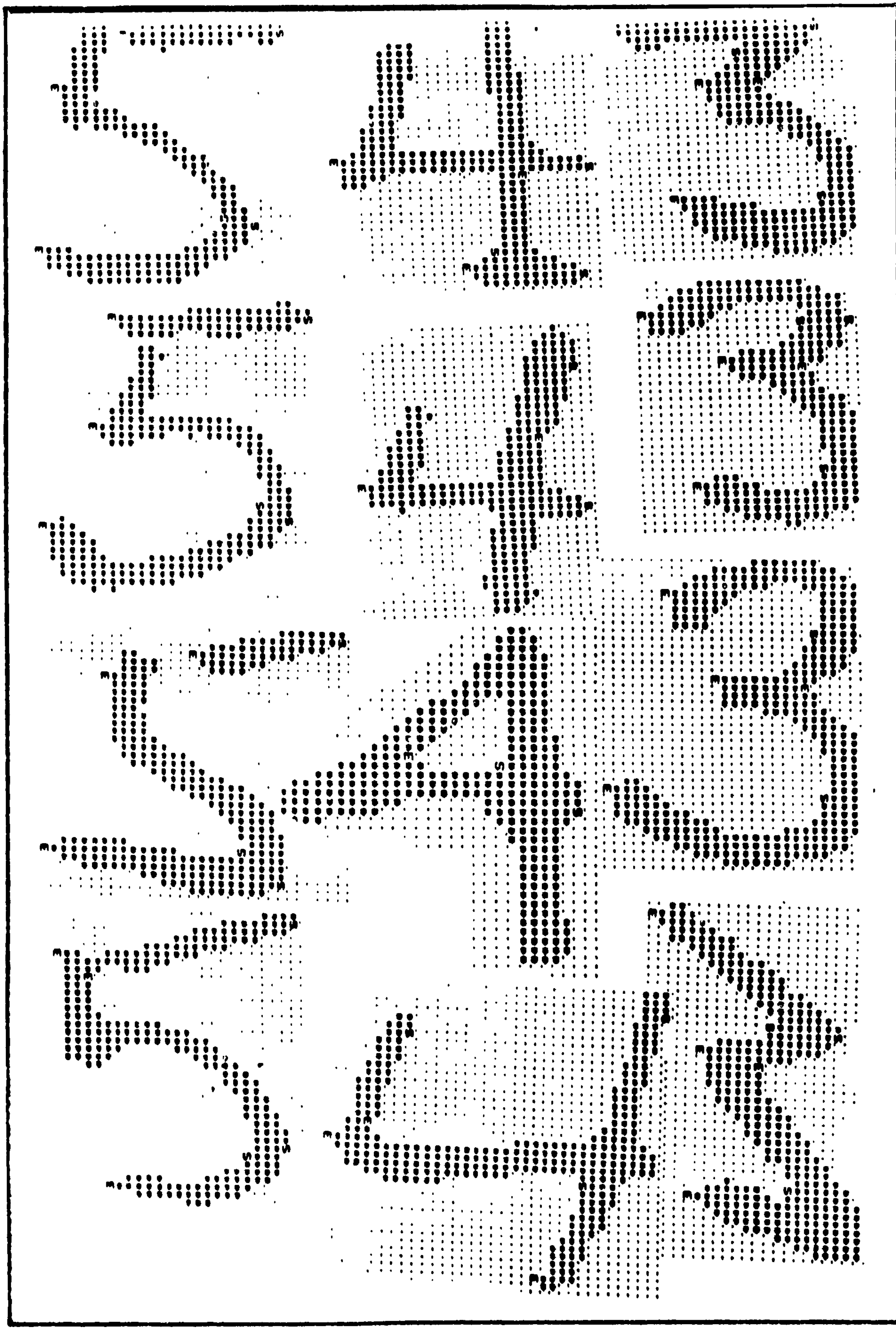


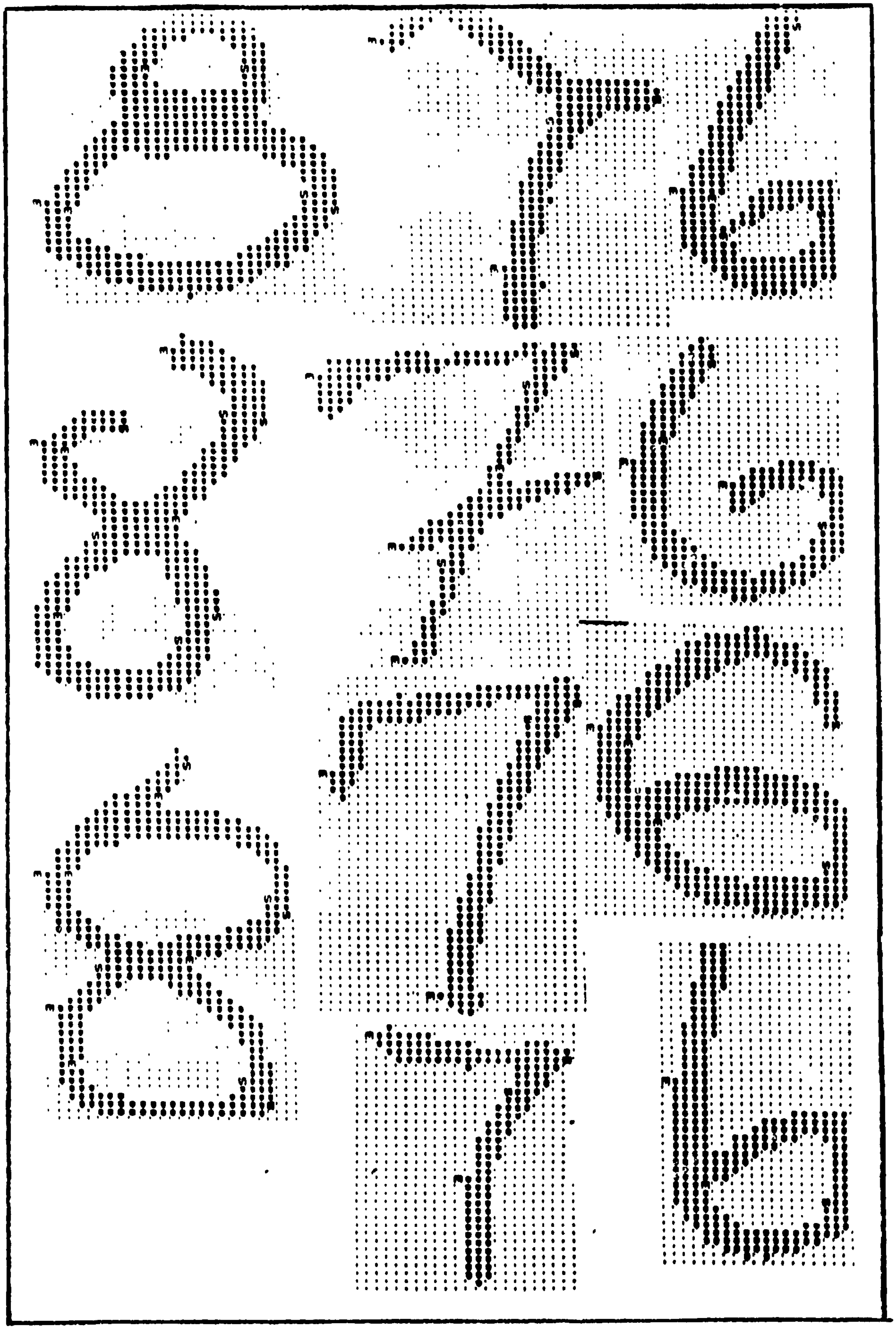


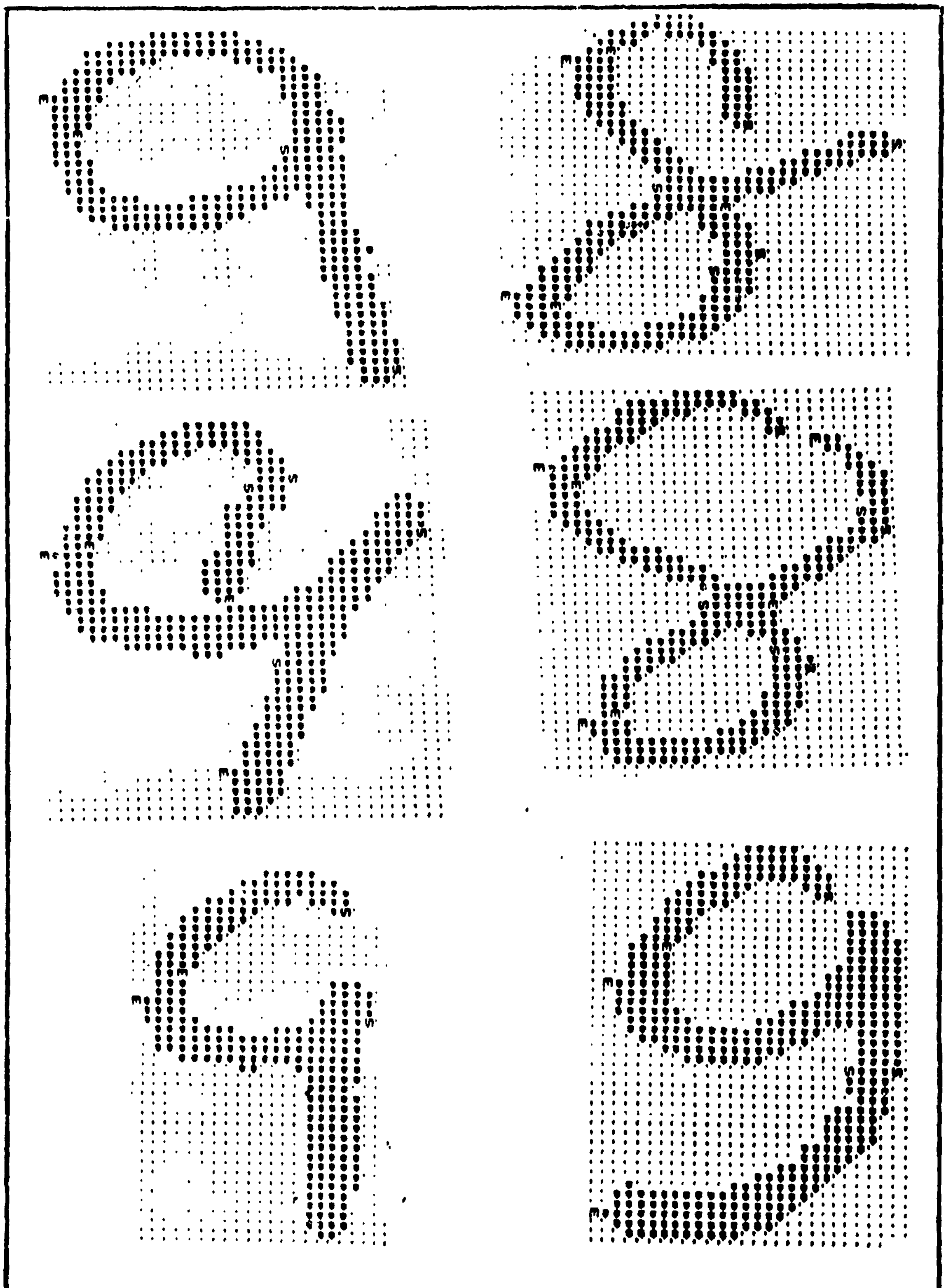


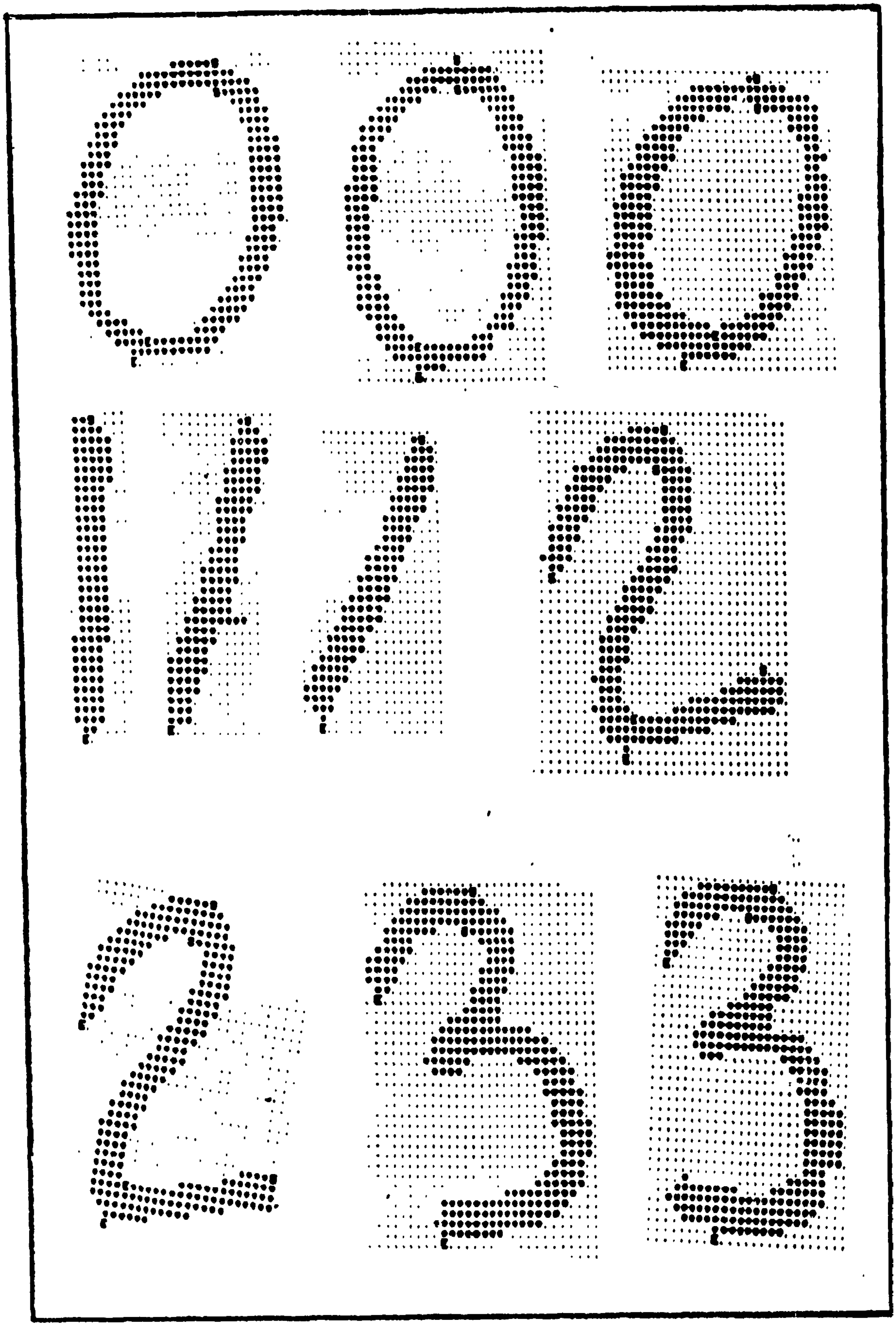


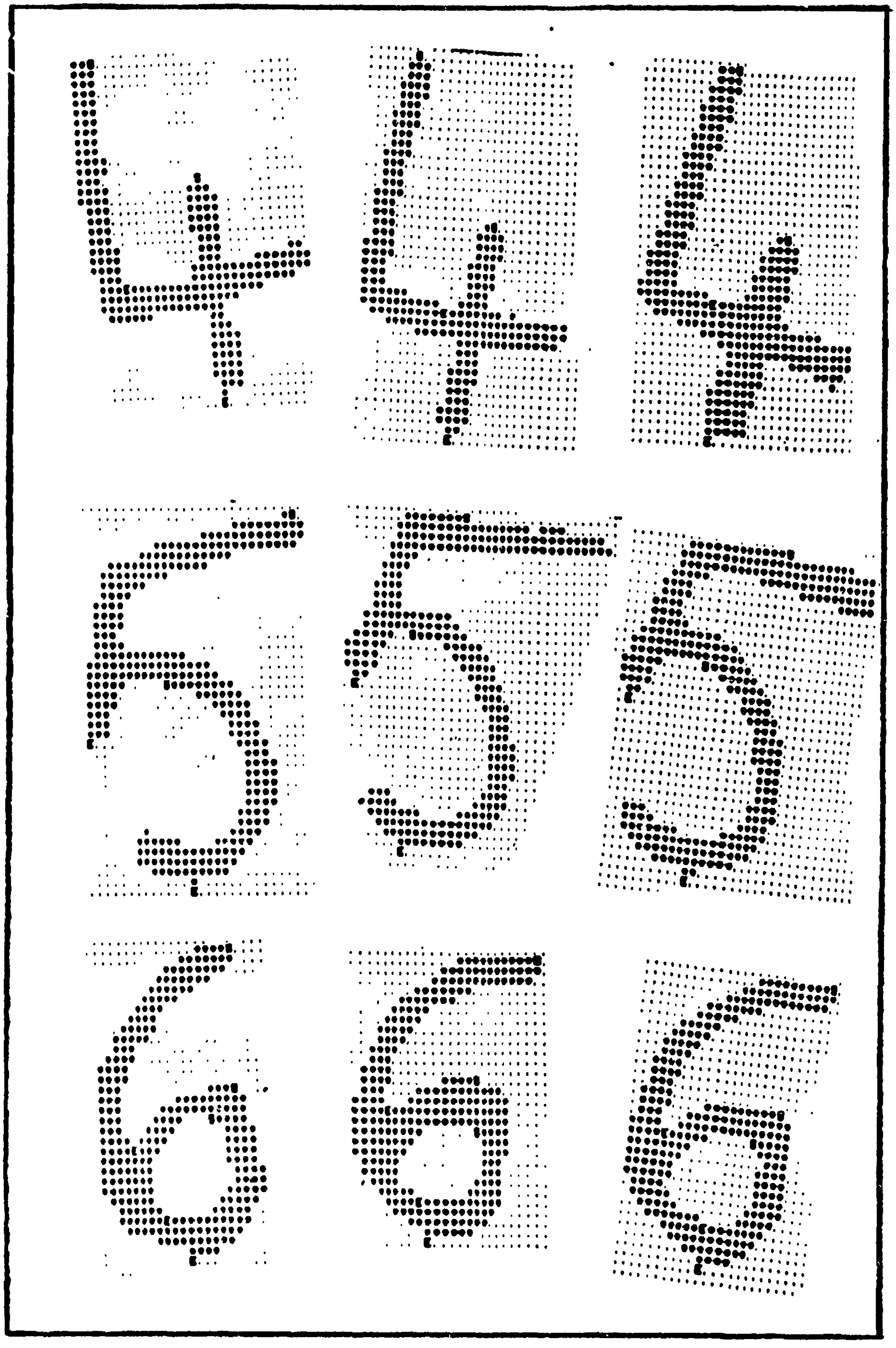


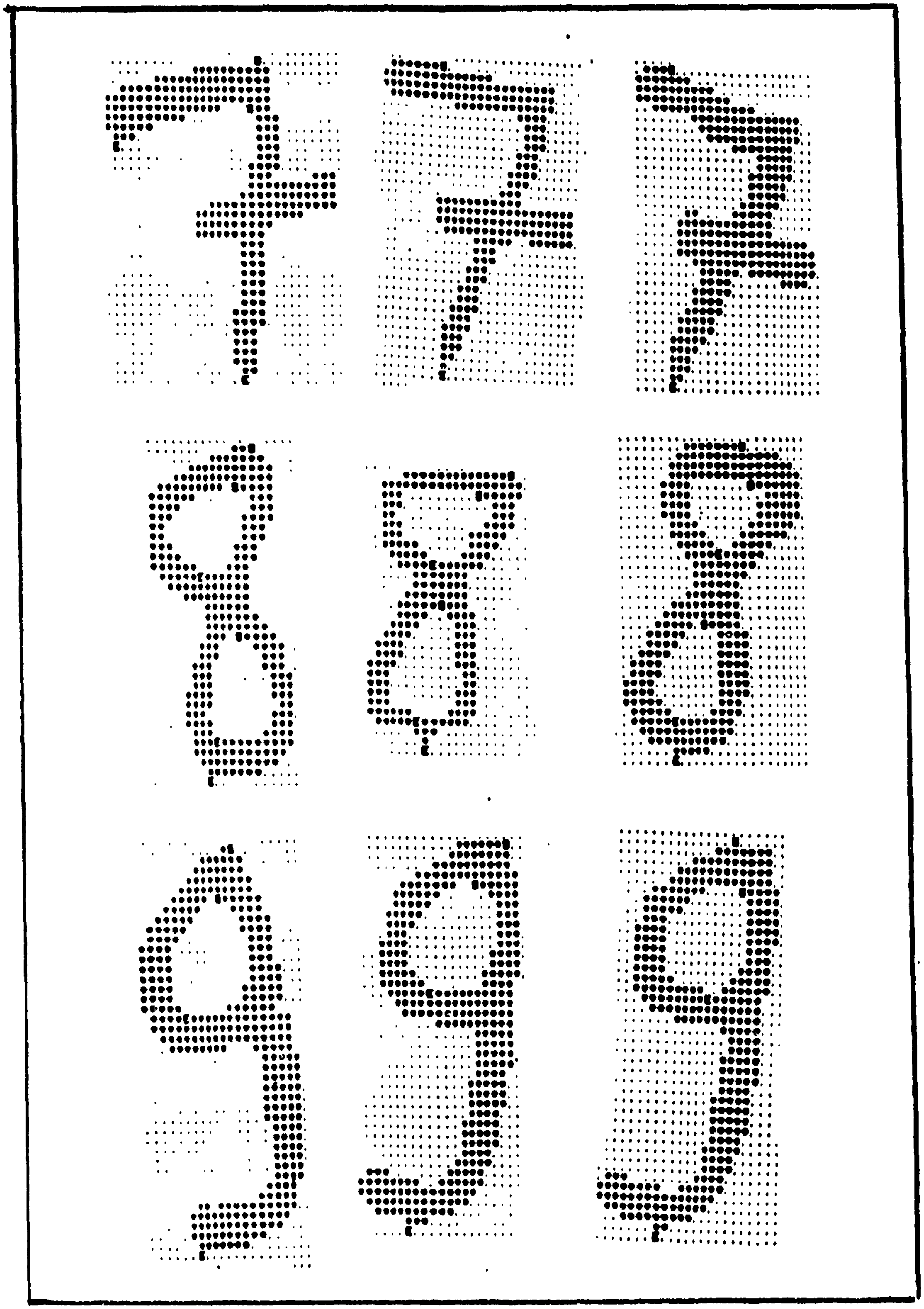












Appendix C

Tables of H-scan and V-scan Classes

Table 1: H-scan Classes

C #	Compound Chain List (features)	Numerals in Class
1	$R_{12,2}$;HBL;HBR	0,8
2	$R_{12,2}$;HB	0
3	$R_{12,2}$;HBU	9
4	$R_{12,2}$;HML;HBR	0
5	$R_{12,2}$;HMU	4,9
6	$R_{12,2}$	1,2,3,5,7
7	$R_{12,2}$;HSD	2
8	$R_{2,2}$;$R_{1,2}$;$R_{4,2}$;HBL;HBR	0
9	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;HBL;HBR	0
10	$R_{2,2}$;$R_{1,2}$;$R_{4,2}$	0,4
11	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;HMU	9
12	$R_{1,2}$;$R_{2,3}$;$R_{1,2}$;$R_{4,3}$	3,4,5
13	$R_{2,3}$;$R_{1,2}$;$R_{6,3}$;$R_{7,2}$;$R_{10,1}$;$R_{7,2}$;$R_{3,1}$	3
14	$R_{12,2}$;HBU;HBD	8
15	$R_{12,3}$;HBU;HMD	8
16	$R_{12,2}$;HMU;HMD	8
17	$R_{2,3}$;$R_{1,2}$;$R_{6,3}$;$R_{7,2}$;$R_{3,1}$	0,3,5
18	$R_{4,1}$;$R_{1,2}$;$R_{2,1}$;HBD	6
19	$R_{4,1}$;$R_{1,2}$;$R_{2,1}$;HMD	6,8
20	$R_{4,1}$;$R_{1,2}$;$R_{2,1}$	6,4,9
21	$R_{4,1}$;$R_{1,2}$;$R_{9,3}$;$R_{7,2}$;$R_{3,1}$	5

Where HMU represents Hole of Medium size in Upper half
HBR represents Hole of Big size in Right half

Table 1 (continued)

C #	Compound Chain List (features)	Numerals in Class
22	$R_{4,1}$;$R_{1,2}$;$R_{2,1}$;HMU;HMD	8
23	$R_{4,1}$;$R_{1,2}$;$R_{2,1}$;HMD	6
24	$R_{4,1}$;$R_{7,2}$;$R_{3,1}$;$R_{1,2}$;$R_{2,1}$	6
25	$R_{9,3}$;$R_{7,2}$;$R_{10,1}$;$R_{7,2}$;$R_{3,1}$;HMD	2
26	$R_{9,3}$;$R_{7,2}$;$R_{10,1}$;$R_{7,2}$;$R_{3,1}$	2
27	$R_{9,3}$;$R_{7,2}$;$R_{3,1}$;HMU	9
28	$R_{9,3}$;$R_{7,2}$;$R_{3,1}$;HMU;HMD	8
29	$R_{9,3}$;$R_{7,2}$;$R_{3,1}$	0,1,3,5,7

Where HMU represents Hole of Medium size in Upper half

HBR represents Hole of Big size in Right half

Table 2: V-scan Classes

C #	Compound Chain List (features)	Numerals in Class
1	$R_{12,2}$;HBR;HBL	8
2	$R_{12,2}$;HB	0
3	$R_{12,2}$;HML	4,9
4	$R_{12,2}$;HMR	6
5	$R_{12,2}$;$R_{1,2}$;$R_{9,3}$;$R_{7,2}$;$R_{3,3}$;$R_{1,2}$;$R_{4,3}$	5
6	$R_{12,2}$	1,7
7	$R_{12,2}$;HBL	9
8	$R_{12,2}$;HMR;HML	8
9	$R_{2,2}$;$R_{1,2}$;$R_{4,2}$;HMR	6
10	$R_{2,2}$;$R_{1,2}$;$R_{4,2}$	0
11	$R_{2,2}$;$R_{1,2}$;$R_{4,3}$;HBU;HBD	0
12	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;HBR	6
13	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;HBL	9
14	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;HMR	6
15	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;HSR	6
16	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;HMU	6
17	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$	4,9
18	$R_{2,3}$;$R_{1,2}$;$R_{4,3}$;$R_{1,2}$;$R_{4,3}$	4
19	$R_{2,3}$;$R_{1,2}$;$R_{6,1}$;$R_{7,2}$;$R_{3,1}$	6
20	$R_{2,3}$;$R_{1,2}$;$R_{6,3}$;$R_{7,2}$;$R_{3,1}$	0,6
21	$R_{4,1}$;$R_{1,2}$;$R_{2,1}$;HMR;HML	8
22	$R_{4,2}$;$R_{1,2}$;$R_{9,1}$;$R_{7,2}$;$R_{3,1}$;HMU;HMD	0

Where HMU represents Hole of Medium size in Upper half

HBR represents Hole of Big size in Right half

Table 2: (continued)

C #	Compound Chain List (features)	Numerals in Class
23	$R_{4,1};R_{1,2};R_{2,1}$;HMR	6
24	$R_{4,1};R_{1,2};R_{2,1}$	0,9
25	$R_{4,1};R_{1,2};R_{9,3};R_{7,2};R_{3,1}$	2
26	$R_{4,1};R_{1,2};R_{9,3};R_{7,2};R_{3,1}$;HSR	2
27	$R_{4,1};R_{1,2};R_{9,3};R_{7,2};R_{10,1};R_{7,2};R_{3,1}$	3
28	$R_{4,3};R_{1,2};R_{9,3};R_{7,2};R_{3,1}$	5
29	$R_{4,3};R_{1,2};R_{9,3};R_{7,2};R_{2,1}$;HBR;HBL	8
30	$R_{4,3};R_{1,2};R_{9,2};R_{7,2};R_{3,1}$	5
31	$R_{9,3};R_{7,2};R_{10,3};R_{7,2};R_{3,1};R_{1,2};R_{4,2}$	3
32	$R_{9,3};R_{7,2};R_{3,1}$;HML	9
33	$R_{9,3};R_{7,2};R_{3,1}$;HMR	8
34	$R_{9,3};R_{7,2};R_{3,1}$;HBL	9
35	$R_{9,3};R_{7,2};R_{3,1}$;HMR	2
36	$R_{9,3};R_{7,2};R_{3,1}$	1,7
37	$R_{9,3};R_{7,2};R_{3,3};R_{1,2};R_{4,1}$	4,9
38	$R_{9,3};R_{7,2};R_{3,3}$	7
39	$R_{9,3};R_{7,2};R_{3,1};R_{1,2};R_{4,2}$	2
40	$R_{9,3};R_{7,2};R_{3,1};R_{1,2};R_{4,3}$	1,7

Where HMU represents Hole of Medium size in Upper half

HBR represents Hole of Big size in Right half