2

ERIM-197200-18-T

ARVD-87-085

Phase I Report

ADVANCED RESEARCH IN RECOGNITION OF HANDWRITTEN ADDRESS ZIP CODES

Charles J. Jacobus
Andrew M. Gillies
Brian T. Mitchell
Steven T. Smith

September 1986 through May 1987

Submitted to:

U.S. Postal Service
Office of Advanced Technology
Technology Resource Department
Washington, DC 20260-8121

Attn: Timothy Barnum

ERIM P.O. Box 8618
Ann Arbor, MI 48107-8618

Advanced Research in Recognition of Handwritten Address ZIP Codes

CHARLES J. JACOBUS
ANDREW M. GILLIES
BRIAN T. MITCHELL
STEVEN T. SMITH

Environmental Research Institute of Michigan
Image Processing Systems Division
P.O. Box 8618
Ann Arbor, Michigan 48107-8618

Contents

1 Introduction

1.1 Executive Summary

THIS report describes the Phase I activities in Advanced Research in Recognition of Handwritten Address ZIP Codes conducted for the **United States Postal Service** at the **Environmental Research Institute of Michigan** under contract 104230-86-H-0042. These activities include an in-depth review of the optical character recognition literature, the development of a handwritten address digitized image data base, the development of a hardware and software testbed for investigating the recognition of handwritten addresses, and the design of a prototype end-to-end ZIP Code recognition system. Beyond the scope originally intended for Phase I, ERIM has implemented that end-to-end system and has determined that it achieves ninety percent digit identification on limited test data. Featured within the overall activities is the concept that development of image algorithms is an incremental process. This concept is strongly reflected in the testbed architecture that has resulted from this work. This approach is unique in that it enables continued system refinement in a way that is both understandable and meaningful. A plan for such refinement of the prototype system is proposed for Phase II of this project.

1.2 Motivation

Presently, handwritten mail that is successfully read by Phase II OCR machines represents less than one half of one percent of all letter mail sorted in the United States. Approximately 15% of all U.S. letter mail is handwritten, of which only 4% was correctly identified by a Phase II OCR machine in a recent test [1]. It is desirable to extend the ability of optical character readers to include recognition of unconstrained handwritten characters so that this 15% can be sorted automatically. To minimize constraints on postal service users, techniques which impose restrictions upon handwritten addresses, such as preprinted guide boxes, will not be used. The present mail sorting system will therefore be enhanced by an automated system capable of recognizing a large percentage of unconstrained handwritten ZIP Codes.

1.3 Project Goals

The primary goal of this OCR project is to develop new techniques for the automatic location and recognition of unconstrained handwritten ZIP Codes in address blocks. A secondary goal is to develop techniques for increasing the ZIP Code recognition reliability with city and state information from the address block. To accomplish these goals, our research has been broken into two phases. In the first phase, the major goal has been to develop a prototype end-to-end system and a methodology for character recognition that facilitates incremental improvements. Tasks required to complete this system have been software development and hardware acquisition for a handwritten address recognition testbed and development of an image data base. The second phase will improve recognition rates achieved in the first phase through an iterative process of testing and refining the prototype algorithms developed during Phase I. The OCR methodology developed in the first phase will provide the framework for algorithmic improvements.

The work for Phase I was broken into the following tasks and completed as summarized here.

1.3.1 Literature Survey

Become familiar with performance requirements for postal OCRs and with previous handwritten address recognition system efforts through a literature survey and USPS library research.

An extensive literature search has been performed, and a survey article describing and categorizing previous techniques has been written [2]. This survey also relates ERIM's methods with past techniques. A trip was made to the USPS library in Washington, and to the Detroit post office. We would like to thank Leonard Tomlinson, Industrial Engineering Coordinator at Detroit's post office, who took the time to show us the sorting process for letters and flats. We observed both hand letter sorting and automatic OCR letter sorting processes. We were able to see first hand the types of handwritten addresses that are rejected or successfully read by current OCR machinery.

1.3.2 Image Data Base

Develop a handwritten address image data base for use in OCR system development.

Over 800 handwritten address images have been added to our data base. Sources of this data base include the EKTRON images originally supplied to us, but consist mostly of images digitized at the State University of New York SUNY. We expect to expand this OCR data base to over 2500 images during Phase II.

1.3.3 Prototype Recognition System

Develop prototype algorithms for image processing and character classification that are used in a prototype end-to-end system, and develop a methodology for improving these algorithms easily. The algorithms must automatically locate the five- or nine-digit ZIP Code block, and then recognize the characters within that block. They must also be extendable to use information outside the ZIP Code block, such as City/State information to reduce the possible number of ZIP Code candidates.

Prototype algorithms to locate the ZIP Code block and recognize its characters have been written and used to build a prototype OCR system. This system has been tested using the handwritten address data base. The USPS City/State/ZIP information data base has been read and will be used to rule out nonexistent ZIP Codes, and may also be used to enhance ZIP Code determination. These algorithms and the prototype recognition system will be described later.

1.4 Project Outcomes

Phase I activities have produced four major outcomes: the OCR literature review, the handwritten address digitized data base, the hardware and software testbed for investigating the recognition of handwritten addresses, and the prototype ZIP Code recognition system. The OCR literature review has provided insight into past character recognition techniques, which has played a major role in formulating an approach to this problem. The handwritten address digitized data base, now over 800 images, has facilitated the construction and initial testing of the end-to-end ZIP Code recognition system. The hardware and software testbed features an ERIM Cytocomputer and a Symbolics LISP Machine running several layers of software designed to minimize vision algorithm development efforts. The prototype ZIP Code recognition system consists of several phases: binary image generation, last line extraction, character segmentation, feature generation, feature segmentation, model matching, and ZIP Code assembly.

1.5 Future Directions

In Phase II, the testbed developed in Phase I will be used to refine the prototype ZIP Code recognition system. Performance data from tests on the image data base will be analyzed and used to focus subsequent research directions on approaches that indicate the greatest increase in system performance. Topics that will require further attention include extracting slanted address lines, segmenting touching digits, windowing digits with overlapping bounding boxes, refining and expanding digit models, adding last line data base information into ZIP Code assembly, and adding context information into ZIP Code hypotheses formulation.

Several improvements to the methods developed in Phase I can be made. Algorithms that perform well with textured backgrounds and analysis of broken digit characters are desired for improvements in binary image generation. Last line extraction performance can be enhanced through techniques that handle slanted address lines. Expansion of the current feature set, which consists solely on concavity features, will enhance character models. Finally, many improvements to ZIP Code assembly are planned. These include looking for more than standard five-digit ZIP Code sequences, analysis of last lines for missing ZIP Code information, and using additional contextual address block information.

2 Phase I Overview

2.1 Literature Review

OPTICAL character recognition is a relatively old field about which much has been written. To help us achieve an understanding of some of the fundamental issues associated with this field, a literature survey extending back to the earliest reported work in character recognition was performed early in Phase I, entitled *Methodologies of Optical Character Recognition*. Also, research at the USPS library was conducted. Abstracts dealing with optical and machine character recognition were collected by ERIM's Information Center through the Dialog information services, specifically through the Inspec and Compendex data bases. Papers included in the search were chosen based upon both a perceived contribution to the theoretical and practical aspects of OCR; and a perceived uniqueness. Additional important literature was gathered through the references of papers and articles that had already been collected. Judging from the computer data bases and references within papers, the final compilation of our literature search represents a near-complete set of significant OCR literature.

Upon completion of this search we wrote a literature review that describes and categorizes previous OCR methodologies and allows us to compare and contrast ERIM's new methods with past OCR research. This literature review segments OCR methodologies into three major steps: preprocessing techniques, feature extraction, and classification methods. Preprocessing techniques are distributed into the following: data representational conversion, thresholding, segmentation, normalization, skeletonization, and filtering. Feature extraction methods are grouped into template matching and transform methods, and topological and geometrical feature methods. Classification methods are assembled into statistical pattern recognition, syntactical pattern recognition, multilevel classification, contextual analysis, and n-gram error detection and correction. A section on ERIM methodologies is included that covers how our methods relate to previous work in light of the categories described in the review. Finally, an extensive annotated bibliography describing OCR literature referenced in the review has been prepared.

This literature review is published separately as an ERIM document, and accompanies this report. All literature collected for this review is available at ERIM through Information Services.

2.2 Address Data Base

We have compiled and are using a large image data base of address block information–at present we have over 800 images. The Computer Science Department of SUNY at Buffalo has been the major source of this data base. SUNY has digitized images from a large number of sources including Electrocom, Alcatel CGA-HBS, ERIM, the downtown Buffalo post office, the USPS main office, and themselves. ERIM supplied SUNY with approximately 700 envelopes to be digitized. These were collected from ERIM personnel at both our Ann Arbor and Washington offices, but the great majority were obtained from Community High School in Ann Arbor. Students were given envelopes and random listings of American publishers' and post-secondary institutions' addresses. They were requested to choose an address from their list and write it on an envelope as if they were actually sending the envelope through the mail. This effort was coordinated through Steve Eisenberg of Community High.

SUNY has designed a sampling procedure to obtain 2500 images of handwritten addresses. Two thousand of the samples are to be divided into ten groups of 200: the ten groups represent each of the ten ZIP Code zones denoted by the first numeral of the ZIP Code. Each of the 200 addresses of each group are to be distributed evenly over every state contained in the zone. The remaining 500 addresses are to be divided into 25 groups containing 20 samples apiece. Each group represents a major U.S. city, and of the 20 in each group, approximately half will be written in cursive and the remainder will be handprinted. The images will be chosen to provide variety in address composition. Addresses with both ZIP, ZIP+4, no ZIP, explicit and abbreviated city and state names will all be represented.

2.2.1 Digitization

We have written a general command file for converting SUNY's images to the format that we require in the Digital Command Language (DCL) of DEC VMS on the ERIM VAX. At first we tried using the DCL command file `READCOMP` given to us by SUNY and written by P. G. Mulgaonkar at SRI International, but image files produced by this operation could not be read at ERIM's computer site since a majority of our image processing software requires files with fixed-length records. To address this problem, we wrote a 'C' program (`fixed.c`) to convert image files to fixed-length record format. We also wrote our own command file that reads compressed image data from SUNY's tape, converts the file to a format required by the `compress.c` program, decompresses the file (`compress.c`), and then alters it using our `fixed.c` program. All source code for these programs and instructions for their use at ERIM are in Appendix C. We have completely automated the process of handling new images given to us in SUNY's image standard–all images are decompressed by running an overnight batch job.

2.2.2 Truthing

Tools for truthing the images have been developed during this project phase. These tools allow the program developer to label segmented characters with the correct information. This process creates a truth file for each individually segmented image. Additional tools have been developed to log the results of the recognition process. These tools take the results of the matcher and write them out to a log file to record the matching results. Other tools have been developed to relate the results of the matching process with the truth files. These tools allow the model development to work in a focused manner. For example, using these tools the model developer can build an image of all matching problems associated with the digit three so that he can focus on resolving these problems. Combined, all these tools create a powerful environment for building, testing, and debugging the digit models.

2.3 ZIP Code Testbed

A testbed for studying and evaluating various approaches to ZIP Code recognition was developed under Phase I of the Advanced Research in Recognition of Handwritten Address ZIP Code Project. This testbed consists of both hardware and software components. These components represent state-of-the-art technology in image processing/understanding and facilitate the use of a rapid prototyping methodology on this problem. Under this methodology, an end-to-end prototype system is first developed and then incrementally refined until satisfactory performance criteria are met. This methodology is well-known to work best on complex, difficult problems in which the solution must be interactively derived from knowledge acquired through experimentation and analysis. Because the ZIP-Code-recognition problem has these characteristics, this methodology is exceptionally well-suited for this problem. The testbed implements this methodology by providing an environment in which ideas can be implemented and tested rapidly. It enables the researcher to focus his attention on the true recognition issues, rather than the detailed implementation issues. We believe this is the only way that a problem this complex can be solved in an efficient manner.

2.3.1 Testbed Hardware

The primary hardware components of the ZIP Code testbed are an ERIM *Cyto-HSS* image processing machine and a *Symbolics 3650* LISP machine. An IP/TCP *Ethernet* link provides the communication between the two machines. The Cyto-HSS provides raw processing power for pixel-based neighborhood operations. It is able to perform high resolution 8-bit-per-pixel morphological vision operations at roughly 10 million neighborhood operations per second [3]. The Symbolics 3650 provides an environment for both numeric and symbolic computation.

2.3.1.1 Cyto-HSS Cytocomputer

The current fourth generation Cyto High-Speed System (Cyto-HSS) developed at ERIM incorporates cascaded neighborhood processing stages together with other significant processing, control, and storage units. Each neighborhood processing stage performs 10 million complete 3×3 neighborhood morphological operations per second in parallel on 8-bit image pixels. By installing 10 stages into the pipeline of stages, 100 million 3×3 neighborhood morphological operations per second are performed. The high-speed intelligent image memories can simultaneously supply and accept 8-bit image pixels at the rate of 10 million pixels per second. This is an effective pixel rate of 20 million 8-bit pixels per second per board. The Cyto-HSS processing is controlled from the ERIM-developed image processing language C4PL (Cytocomputer Portable Parallel Picture Processing Language). This software system is the result of several generations of evolution in ERIM proprietary interactive image analysis languages. C4PL is fully integrated with ERIM's Cyto-HSS Image Processing Systems.

2.3.1.2 Symbolics 3650

The Symbolics 3650 is a powerful computing environment. It features a hardware tagged memory architecture for run-time data-type-checking and generic operations, a stack-oriented architecture with large stack buffers, a powerful front-end microprocessor, hardware-assisted garbage collection for memory efficiency with low software overhead, and a sophisticated and supportive software engineering environment. The current configuration for this machine provides 8 MB of main memory, one 368 MB Winchester disk drive, two RS232C serial I/O ports, a built in Ethernet interface, and 15 expansion slots for additional options. The software environment features such extensions as *Flavors* object-oriented programming, networking, window management, graphics, multitasking, editors, and debuggers.

2.3.1.3 Networking and Communications

An initial communication capability between the Cytocomputer and the Symbolics was developed in the first phase of this project. An overview of this capability is shown in Figure 1. This implementation makes use of the existing C4PL environment and its ability to program the Cytocomputer. To do this, the VAX is introduced as an intermediate node between the Symbolics and the Cytocomputer, with all three machines connected via an ethernet. The VAX is used as a C4PL host. In this configuration, the Symbolics communicates with C4PL and as a consequence, C4PL interacts with the Cytocomputer. For example, if an OCR algorithm running on the Symbolics determines that more image analysis data is needed from the Cytocomputer to help make a good decision, it can send, via the ethernet, a

C4PL request to a C4PL process on the VAX. C4PL reads the request off of the ethernet and issues corresponding requests to the Cytocomputer.

2.3.2 Testbed Software

A major component of the testbed is software that was developed under this phase of the contracted effort. This software builds upon the native languages of the Cyto-HSS and the Symbolics LISP machine to provide an environment in which recognition algorithms can be developed in an efficient way. The testbed software is based on a library philosophy that enables ideas to be developed from existing algorithm modules. It thus minimizes the amount of new code that must be developed when new ideas are implemented. The testbed software is also based on an open architecture philosophy that permits continued extensions of the libraries of algorithm modules as the research develops. These extensions can be made at the image processing, segmentation, feature attribute, feature relation, and matching levels. Under this philosophy, as new fundamental approaches to ZIP Code recognition are identified, they can be incorporated into the testbed environment. This facilitates rapid prototyping by providing a powerful and flexible development environment in which research can be focused on recognition issues.

An overview of the software system developed under this project is shown in Figure 2. Processing within this system consists of three main phases: transforming the raw ZIP-Code image into state-labeled feature maps, composing the resulting feature maps into a composite symbolic feature map, and identifying ZIP Codes by matching feature-based digit models to the composite map. The interaction between these three phases is driven by a hierarchical matching strategy that is designed to minimize the amount of unnecessary processing while maintaining algorithm performance. The matching strategy itself is directed by feature-based digit models. These models facilitate rapid development of experimental vision systems.

In the first processing phase, the raw ZIP-Code image is transformed into state-labeled feature maps. These transformations are performed by low-level, image-processing algorithms operating on the Cytocomputer. These algorithms are directed at locating ZIP-Code features in the image. A typical feature might be a digit window, a concavity, an end point, or a junction

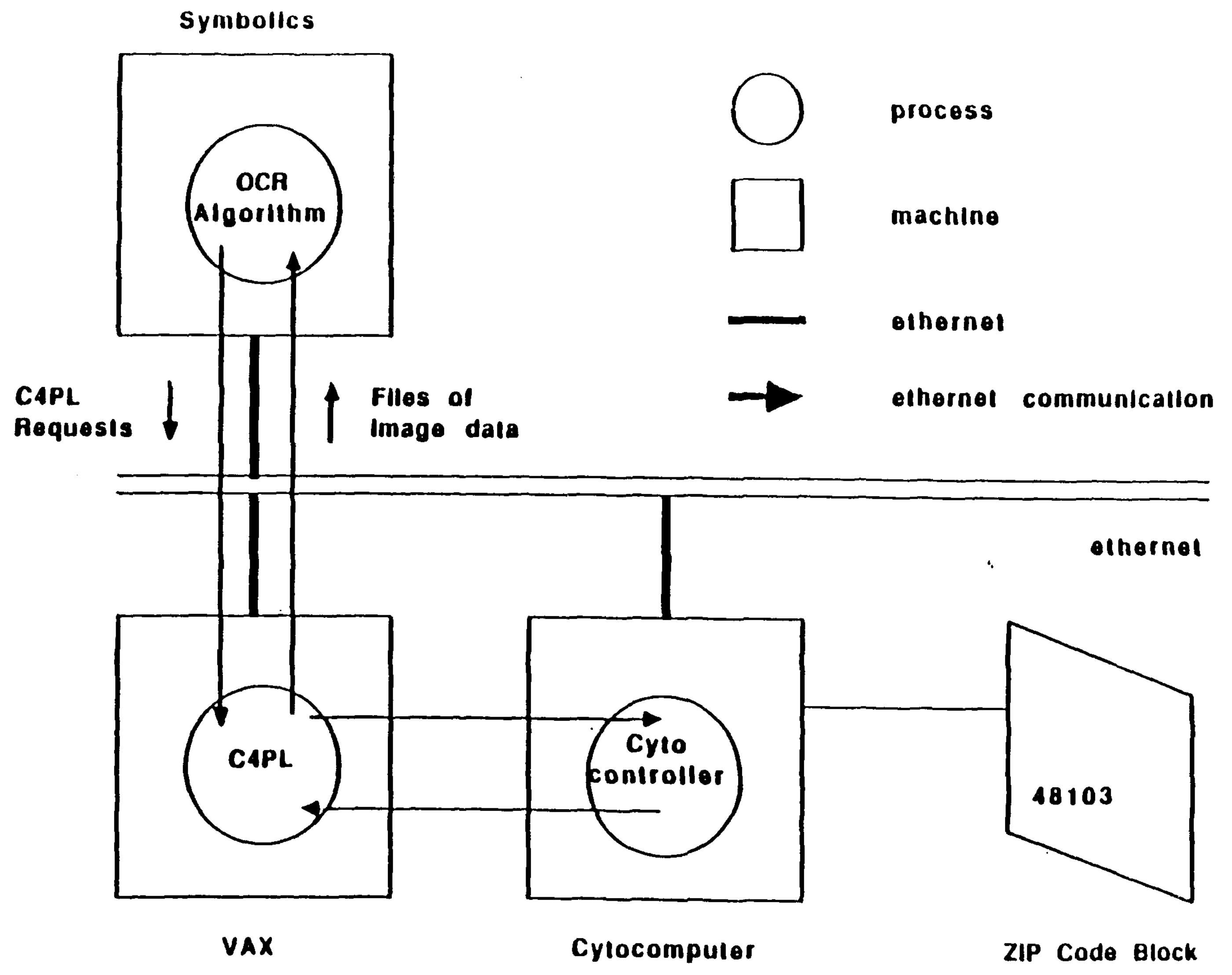


Figure 1. Networking and Communications Overview

point. The Cytocomputer is ideally suited to perform such operations.

In the second processing phase, the state-labeled feature maps are transformed into a composite feature map. This is accomplished by segmenting the individual state-labeled images and taking various measurements on the resulting regions. Since the feature maps are represented as state-labeled images, the process of segmentation is straightforward. Furthermore, the segmentation is performed only on an as needed basis. Thus, if simple initial tests determine that an important digit feature does not exist in certain portions of the image, then there is no need to segment additional feature maps in those areas. Likewise, if a feature does not possess a necessary digit attribute, e.g. size or shape, then a digit match is not possible and there is no need to compute additional feature attributes. Thus, the entire composite feature map is dynamically developed as needed by the matching process, and since only information that is required for the solution is calculated, the overall computational complexity of the resulting vision system is reduced.

In the third and final processing phase, digits are identified by matching prototypical feature-based digit models with portions of the composite symbolic feature map. The models are represented as ordered matching clauses that describe how features must exist in the image in order to be classified. These clauses identify the features that are required, the attributes that these features must have, and the relationships that must exist between the features. These clauses also dictate how attention will be focused in searching for the digits within the image and how those digits will be ultimately identified. The clauses represent an efficient, hierarchical matching strategy that eliminates nonmatches as quickly as possible by using simple tests. For example, if a test determines that a key digit feature does not exist in the image, then there is no reason to continue the matching process. In keeping with this idea, the overall testing strategy is hierarchically organized from least to most complex with the most complex tests being performed on only the most promising matches.

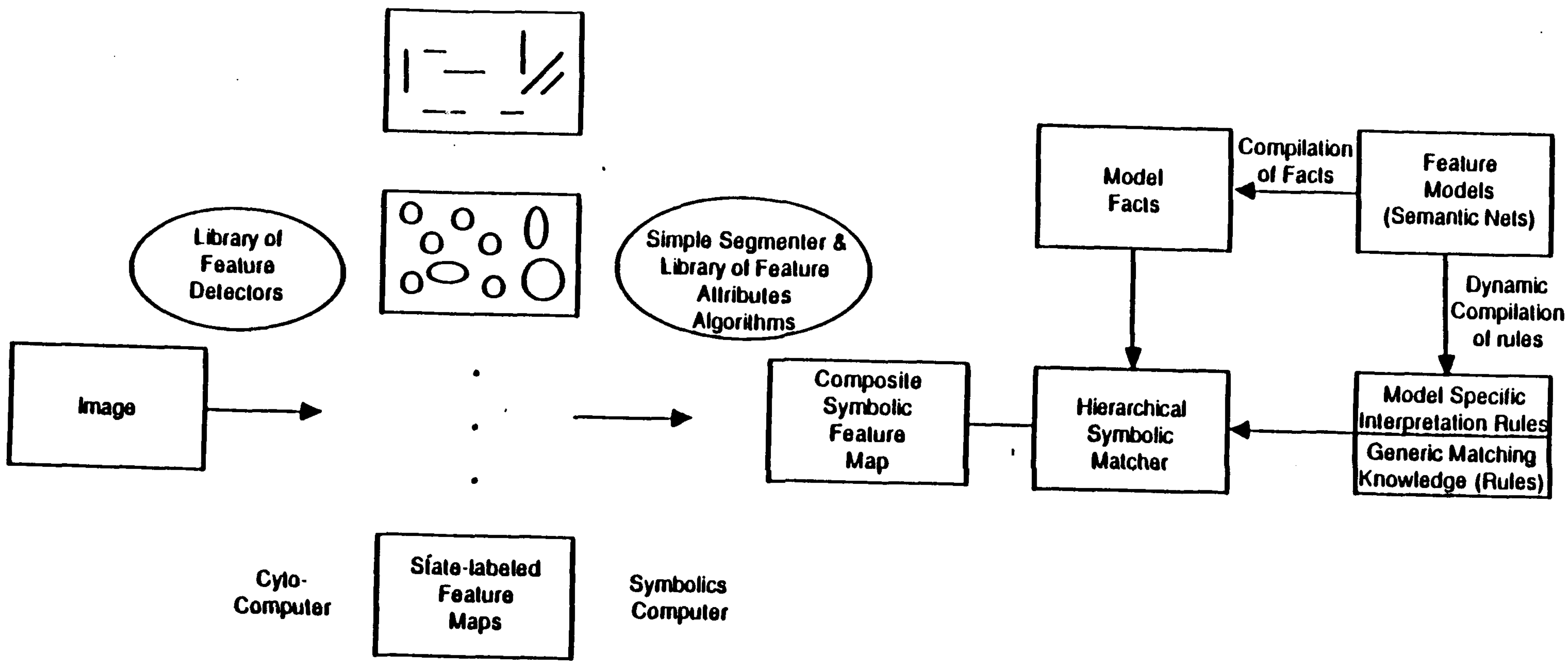


Image
Library of Feature Detectors
Cylo-Computer
Sĺate-labeled Feature Maps
Simple Segmenter & Library of Feature Attributes Algorithms
Symbolics Computer
Composite Symbolic Feature Map
Hierarchical Symbolic Matcher
Model Facts
Compilation of Facts
Feature Models (Semantic Nets)
Dynamic Compilation of rules
Model Specific Interpretation Rules
Generic Matching Knowledge (Rules)

Figure 2. TESTBED OVERVIEW

2.3.2.1 State-Labeled Feature Map Development

The testbed uses the Cytocomputer to process the input ZIP-Code image and produce the state labeled feature maps. A state-labeled feature map is an image in which pixels in a given state indicate the presence of a particular kind of feature in the original image. The Cytocomputer produces state-labeled feature maps by transforming the original image with a series of operators. The Cytocomputer has operators for filtering, thresholding, and skeletonizing images, as well as operators for finding and labeling regions in the image.

The Cytocomputer processing starts with grayscale filtering operations. One important class of Cytocomputer filters are those based on the opening and closing operations of mathematical morphology. These filters are tuned to specific spatial scales, and correct filtering depends on the spatial scale of the ZIP Codes located in the imagery. Small scale filters are used to smooth the image, removing variations which are too small to be of interest. Large scale filters are used to estimate slowly varying backgrounds, which may then be subtracted from the image to form a thresholdable image. The Cytocomputer can also perform matched filtering which will tend to emphasize digits which have shape characteristics which match the filter's shape characteristics.

The filtered image may be used as input for edge detection or may be thresholded to produce binary images. In the case of edge detection, the detected edge segments are sent directly to the composite feature map. The connected regions in binary images may also be passed directly to the composite feature map, or they may be subjected to further Cytocomputer processing.

In many cases skeletonization (also called thinning) may be used to reduce binary images to lines of single pixel thickness. The Cytocomputer can also mark endpoints and junctions in skeletonized images. A skeletonized and marked image provides a very useful summary of the overall structure of the shapes in the binary images.

Another kind of processing for binary images involves the use of operators from mathematical morphology. With these operators regions passing size and shape criteria may be kept while other regions are removed from the image. Region borders may also be smoothed. By noting exactly what kind of processing produced a given region, the composite feature map may know quite a lot about the regions it has without having to explicitly measure their properties.

The library of Cytocomputer operations provides the building blocks to perform a large number of image processing functions. The choice of which operators use, and how to combine them depends on the problem. The goal of this level of processing is to summarize the important information in the image in the form of state-labeled feature maps. The regions in these maps become the features on the composite feature map.

2.3.2.2 Composite Map Development

The second processing phase within the system is focused on the development of the composite symbolic feature map. This phase is critical to the overall system, since it provides the bridge between the pixel-based processing of low-level vision and the symbolic-based processing of high-level vision. The segmenter, which extracts features for the composite map from the state-labeled feature maps, must provide a rich set of techniques for identifying a wide range of features. The data structures that define the composite map must be flexible to allow long-term system enhancements. They must also be efficient to enable rapid information retrieval over a wide range of queries.

The representation of the composite map determines much of the overall system flexibility and performance. An overview of the map representation is presented in Figure 3. As illustrated in this figure, the map is an object with two primary components: a feature list and a relation hash table. Both facilitate access to information in the composite map. This access, however, is somewhat varied. The feature list is an indexed list of features identified within the image. This list provides quick and ready access to features located anywhere in the image. Typically, this form of map access is used to locate features within some region of interest in the image. The relation hash table is a simple hash table object. This table contains previously calculated relationships between features of the composite map. Typically, it is accessed anytime a relationship is requested to see if that relationship has already been determined. Together the feature list and the relation hash table facilitate efficient access to composite map information.

The primary component of the feature list is a feature. Within the testbed, a feature is represented as a simple object with a region and an attached property list. The region locates the feature boundaries within the original image. The property list holds the feature type and state produced by the segmenter. The type identifies the state-labeled image which produced the feature. The state identifies its intensity level within the image. Run-length encoding is used as the representation scheme by the testbed segmenter. A library of methods use this representation to calculate feature attributes. A scheme is used which intercepts the message defined by these methods. If the message has not been received before, it is allowed to continue, and the resulting attribute value is added to the feature property list. If a message has been received before, it is aborted, and the desired result is simply recalled from the property list. In this way, feature attributes are computed once when needed, and redundant calculations are eliminated.

Features are extracted from the state-labeled maps and placed into the composite map by the segmenter. Since this segmenter operates exclusively on state-labeled images, it is much simpler than corresponding gray-level segmenters. The segmenter simply identifies features as regions in the state-labeled image. For each identified region, the routine then creates a new feature object, attaches its region description, and adds it to the composite map.

The primary component of the relation hash table is a description of a relationship between two features and a relationship value. The description is stored as a Lisp form, and used as an index into the hash table. The relationship value represents the result of the relationship calculation. If the relationship has been calculated once, the hash table is used to retrieve the resultant value, and thus avoid unnecessary recomputation.

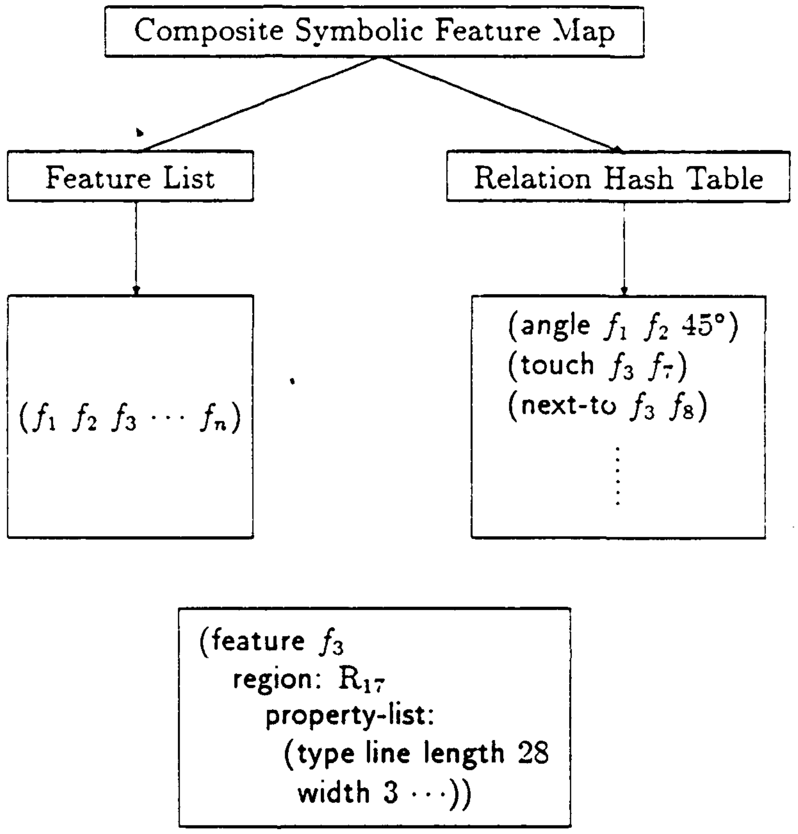


Figure 3. Composite Feature Map Representation

2.3.2.3 Model Development and Matching

The third processing phase within the system is focused on matching digit models to composite map facts. The models describe how selected features must exist in the image to be classified as a particular digit. The models are either primitive or complex. The primitive models correspond to features in the state-labeled feature map. The complex models consist of models, model attributes, and model relations. The attributes specialize the models and determine their size, shape, orientation, etc. The relations define the spatial qualities, e.g. above, next-to, or between, that exist between the models. This recursive model definition enables the development of complex models from simpler models. It also facilitates the development of model packages that can be developed and placed into libraries.

The models are developed in a model-matching language developed at ERIM. This language is comprised of clauses that can be organized into four different classes: bindings-operator, free-set-operator, side-effect-operator, and control-operator. The bindings-operator clauses either generate or prune the candidate matching bindings. The free-set-operator clauses manipulate the free-set from which the candidate bindings are selected. The side-effect-operator clauses generate a desired side-effect. The control-operator clauses determine the flow of control for clause evaluation. Together these clauses create a powerful model-matching language which is well integrated into the Symbolics LISP machine environment.

The model-matching language can be defined in BNF form as seen in Figure 4. This formal definition specifies the syntax that is used in writing models. The clauses shown in this definition are organized into the four classes outlined above in Figure 5. This organization is used below to describe the semantics associated with each clause.

Binding-Operator Clauses

The bindings-operator clauses define the way in which bindings are handled by the model-matching language. These clauses can be further broken down into two groups: those that generate bindings and those that prune bindings. Included within the clauses that generate bindings are the require, allow, and bind clauses. Included within the clauses that prune bindings are the test, ordered, pairwise, and and-nothing-else clauses.

Require

The require clause is used to bind a matched submodel to a variable. It generates all possible bindings for each entry on the current binding list. For each new binding, features are matched and removed from the free-set. If

```
⟨model⟩               ⟶ (defmodel (⟨arg-list⟩) (⟨clause-list⟩))

⟨arg-list⟩            ⟶ :color ⟨color⟩ | ⟨empty⟩
⟨color⟩               ⟶ :black | :yellow | :red | :green | ···

⟨clause-list⟩         ⟶ primitive | ⟨clause⟩ ⟨clause-list⟩ | ⟨empty⟩
⟨clause⟩              ⟶ ⟨require-clause⟩ | ⟨allow-clause⟩ | ⟨eval-clause⟩
                          | ⟨bind-clause⟩ | ⟨free-set-clause⟩ | ⟨ignore-clause⟩
                          | ⟨test-clause⟩ | ⟨ordered-clause⟩ | ⟨pairwise-clause⟩
                          | ⟨if-clause⟩ | ⟨forbid-clause⟩ | ⟨and-nothing-else-clause⟩

⟨require-clause⟩      ⟶ (require ⟨var⟩ ⟨model-or-list⟩)
⟨allow-clause⟩        ⟶ (allow ⟨var⟩ ⟨model-or-list⟩)
⟨eval-clause⟩         ⟶ (eval ⟨any-Lisp-form⟩)
⟨bind-clause⟩         ⟶ (bind ⟨var⟩ ⟨any-Lisp-form⟩)
⟨free-set-clause⟩     ⟶ (free-set ⟨any-Lisp-form⟩)
⟨ignore-clause⟩       ⟶ (ignore ⟨Lisp-single-arg-predicate⟩)
⟨test-clause⟩         ⟶ (test ⟨any-Lisp-form⟩)
⟨ordered-clause⟩      ⟶ (ordered ⟨Lisp-two-arg-predicate⟩ ⟨model-list⟩)
⟨pairwise-clause⟩     ⟶ (pairwise ⟨Lisp-two-arg-predicate⟩ ⟨model-list⟩)
⟨if-clause⟩           ⟶ (if ⟨any-Lisp-form⟩ ⟨clause⟩)
⟨forbid-clause⟩       ⟶ (forbid ⟨any-Lisp-form⟩)
⟨and-nothing-else⟩    ⟶ (and-nothing-else)
⟨var⟩                 ⟶ ⟨any-Lisp-symbol⟩
⟨model-list⟩          ⟶ ⟨model⟩ ⟨model-list⟩ | ⟨empty⟩
⟨model-or-list⟩       ⟶ ⟨model⟩ ⟨or⟩ ⟨model-or-list⟩ | ⟨empty⟩
⟨or⟩                  ⟶ or | ⟨empty⟩
```

Figure 4: BNF Definition of Matching Language

no match is possible for a particular binding, that binding is removed from the current binding list. Thus, this clause has both generating and pruning characteristics. It is classified as a generating clause, however, since it has overwhelming generating potential. In fact, it is recommended that this clause be used only when the binding list and free-set list have been pruned as much as possible to minimize the computational complexity associated with its usage.

Allow

The allow clause is identical to the require clause, except that It does not prune any current bindings which fail to produce a match. Furthermore, this clause carries forward all current bindings as they were before this clause was evaluated. Because of these two characteristics, this clause is exclusively a generating clause, and its generating potential exceeds that of the require clause. It thus should be used, as the require clause, only when the binding list and free-set list have been pruned as much as possible.

Bind

The bind clause is use to bind the results of a LISP form to a symbol. In essence, this operation defines a local variable for use in the matching process. Since, the LISP form can refer to variables that are bound to unique values for each binding in the current binding list, this operation adds a new binding to each binding on the current binding list. This operation is extremely useful when calculating local results that will be used over and over again.

Test

The test clause is used to make sure that bindings meet certain requirements. For each binding in the current binding list, this operation substitutes the values associated with the local variables into the LISP form. The resulting LISP form is then evaluated. If the evaluation is false, then the binding is removed from the current binding list. Otherwise, the binding passes the test, and remains on the list. This operation provides the mechanism for assuring that model-attribute and model-relationship requirements are satisfied.

Ordered

The ordered clause is an extended version of the test clause, in that it performs several tests. For each binding in the current binding list, this operation first creates local bindings for the symbols on the model list. If any of the symbols remain unbound, they are removed. Then the values of the remaining first and second symbols are applied to the two-argument predicate. If the result is false, the binding is removed from the current binding list. Otherwise, the values of the second and third remaining symbols are applied to the predicate. Again, if the result is false, the binding is removed. This process is then continued until it exhausts the ordered symbol values. Potentially, the length of the remaining symbol list minus one application of the predicate can result from this operation. If the results of all these applications is true, then the binding remains a match candidate of the current binding list.

Pairwise

The pairwise clause is an extension of the ordered clause, in that it performs additional tests. For each binding in the current binding list, this operation first creates local bindings for the symbols on the model list. If any of the symbols remain unbound, they are removed. Then the values of the remaining first and second symbols are applied to the two-argument predicate. If the result is false, the binding is removed from the current binding list. Otherwise, the values of the first and third remaining symbols are applied to the predicate. Again, if the result is false, the binding is removed. This process is then continued until it exhausts the pairwise symbol values. The number of pairwise combinations that exist in the remaining symbol list corresponds to the number of potential applications of the predicate can result from this operation. If the results of all these applications is true, then the binding remains a match candidate of the current binding list.

And-Nothing-Else

The and-nothing-else clause is used to test the free-set to make sure that everything is accounted. For each binding in the current binding list, this operation checks to see if the free-set list is empty. If it is not, then the binding is removed from the current binding list. If it is, then the binding remains a match candidate on the current binding list.

Free-Set-Operator Clauses

The free-set-operator clauses provide a mechanism for accessing the free-set. Three clauses fall into this class: and-nothing-else, free-set, and ignore. The and-nothing-else clause was described above as a binding-operator clause. It can also be thought of as a free-set operator because it bases its action on the state of the free-set. However, its overwhelming characteristic is to affect the current binding list, and because of this it is best thought of as a binding operator. The other two clauses are strictly free-set operators, since they have no effect on the binding list.

Free-Set

The free-set clause is used to update the free-set associated with each binding of the current binding list. For each binding, this operation substitutes the values associated with the local variables into the LISP form. The free-set associated with the binding is also substituted for the symbol: free-set. The resulting LISP form is then evaluated, and the resultant value becomes the new free-set associated with the binding. This operation can either add or subtract entries in the free-set. It is useful when significant changes in the free-set are required during the matching operation.

Ignore

The ignore clause is used to prune the free-set associated with each binding of the current binding list. This operation applies each element of the free-set associated with a particular binding to a specified single-argument predicate function. If the result is false, the element is removed from the free-set. If is true, it remains on the free-set. This clause is especially useful in pruning down large free-sets after some initial evaluation. This, in turn, results in significant computational savings.

Side-Effect-Operator Clauses

The side-effect-operator clause provides a mechanism for evaluating procedures that generate side effects, e.g. outputting graphical displays. Only one clause falls into this class, the eval clause. This single clause, however, is very powerful, since it enables the matcher to communicate with its environment.

Eval

The eval clause is used to generate side effects. For each binding, this operation substitutes the values associated with the local variables into the LISP form. The resulting LISP form is then evaluated. It has no affect on either the current binding list or the corresponding free-sets. This clause is especially useful in communicating with the LISP environment.

Control-Operator Clauses

The control-operator clauses provide a mechanism for directing the flow of control through the matching process. Two clauses fall into this class, forbid and if. These clauses allow the matcher to recognize when no match is possible and when certain operations should be performed.

Forbid

The forbid clause is used to abort the current match process. This operation applies each element of the free-set associated with a particular binding to a specified single-argument predicate function. If the result of any application is true, the matching operation is aborted immediately. This operation is especially useful for doing a scan of the free-set to see if something exists that would make a match impossible. When such an occurrence exists. there is simply no reason to expend any further effort in trying to coerce a match. it cannot be done.

If

The if clause is used to determine whether certain clauses should be used in the matching process. For each binding, this operation substitutes the values associated with the local variables into the LISP form. The resulting LISP form is then evaluated. If the resulting value is true, then the conditional clause is evoked on the current binding. This conditional clause then determines the final state of the binding. If the resulting value is false. the binding is unaffected. This clause is especially useful making sure that all the preconditions for the conditional clause are satisfied.

The digit models are developed in the model-matching language just described. The process begins by carefully analyzing the feature qualities associated with digits or digit components. This analysis results in an understanding of how features can be used to describe a particular numeric character. This description includes existence of particular features. measurable attributes for the resulting features and spatial relationships between those features. This description builds upon components of the state-labeled feature maps and the composite symbolic feature map created from the first two processing phases. The ordering of the model clauses also determine the efficiency of the resulting matching process.

2.4 Prototype ZIP Code System

A prototype ZIP Code recognition system was developed under Phase I of the Advanced Research in Recognition of Handwritten Address ZIP Codes. This system provides a first-cut implementation of an end-to-end system for recognizing ZIP Codes. It was developed on the testbed described above. This prototype system serves several purposes. It provides a tool for evolving specifications for a robust ZIP Code recognition system. It provides a framework for exploring the interaction between each of the system phases.

Clause	Binding Operator	Free-set Operator	Side-effect Operator	Control Operator
allow	X			
and-nothing-else	X	X		
bind	X			
eval			X	
forbid		X		X
free-set		X		
if				X
ignore		X		
ordered	X			
pairwise	X			
require	X			
test	X			

Figure 5. VISTA-MATCHER Clause Types

It also provides a mechanism for extending system performance through incremental testing, evaluating, and refining. This system thus becomes an integral part in the overall rapid prototyping and iterative refinement research methodology.

The prototype system goes through several processing phases. In the first phase, the initial, grey-level image is transformed into a binary image. Next, the last address line is extracted from the the binary image. The resultant last line image is then segmented into characters and/or symbols. Features are then generated for these characters. The resultant state-labeled feature map is then segmented to produce a composite symbolic feature map. Digit models are then matched to portions of this map. The resultant identified digits are then assembled into a five-digit ZIP Code. Each of these processing steps is described in detail below.

2.4.1 Binary Image Generation

The first processing phase in the prototype system transforms the initial, raw, grey-level image into a two-valued, binary image. A simple thresholding approach was implemented on the ERIM Cytocomputer to accomplish this task. Under this approach, limited testing of sample input images was performed to determine an appropriate threshold or cutoff value to use in separating the background of the address block from the stroke. This value was then used to implement the Cyto thresholding algorithm. Within this algorithm, morphological operations were used to sort pixels in the initial, input image into those belonging to the stroke and those belonging to the background. Those values that were below the selected threshold value were classified as belonging to the background by determining whether the pixel value was above or below the selected threshold. The input to this Cyto algorithm is an image from the SUNY Buffalo address block data set. The output is an image in which the character strokes of the address block has one value and the background has another value.

2.4.2 Last Line Extraction

Next, the prototype recognition system locates the last line in the binary image. As a first attempt at this portion of the overall recognition process, a histogramming approach was implemented on the ERIM Cytocomputer. Figure 6 shows a pictorial representation of the resulting algorithm. As seen in the figure, this approach begins by migrating the stroke pixels in the binary image in a leftward direction to create a histogram. The height of each entry in this histogram indicates the number of pixels located on each horizontal raster line of the binary image. Once the histogram is computed, it is sliced by removing a fixed number of pixels from each line. Any small gaps in the resulting sliced histogram are then removed to produce line locations within the image. The last such line is finally windowed and placed into a last line image.

2.4.3 Character Segmentation

Individual characters within the binary last line image are then identified. A histogramming approach similar to the one used in last line extraction was implemented on the ERIM Cytocomputer. Figure 7 shows a pictorial representation of the resulting algorithm. As seen in this figure, the approach begins by migrating the stroke pixels in the last line image in a downward direction to create the histogram. In this histogram, the height of each entry indicates the number of pixels in location on each vertical raster line of the last line image. As before, small gaps are removed from this histogram to produce estimated character locations within the last line. These estimates are then used to produce a last line image in which the individual characters are isolated and windowed.

2.4.4 Feature Generation

The segmented last line image is next processed to produce a state-labeled image. A concavity approach was implemented on the ERIM Cytocomputer for demonstration in the prototype system. Figure 8 shows a pictorial representation of the resulting features. As seen in this figure, six unique features

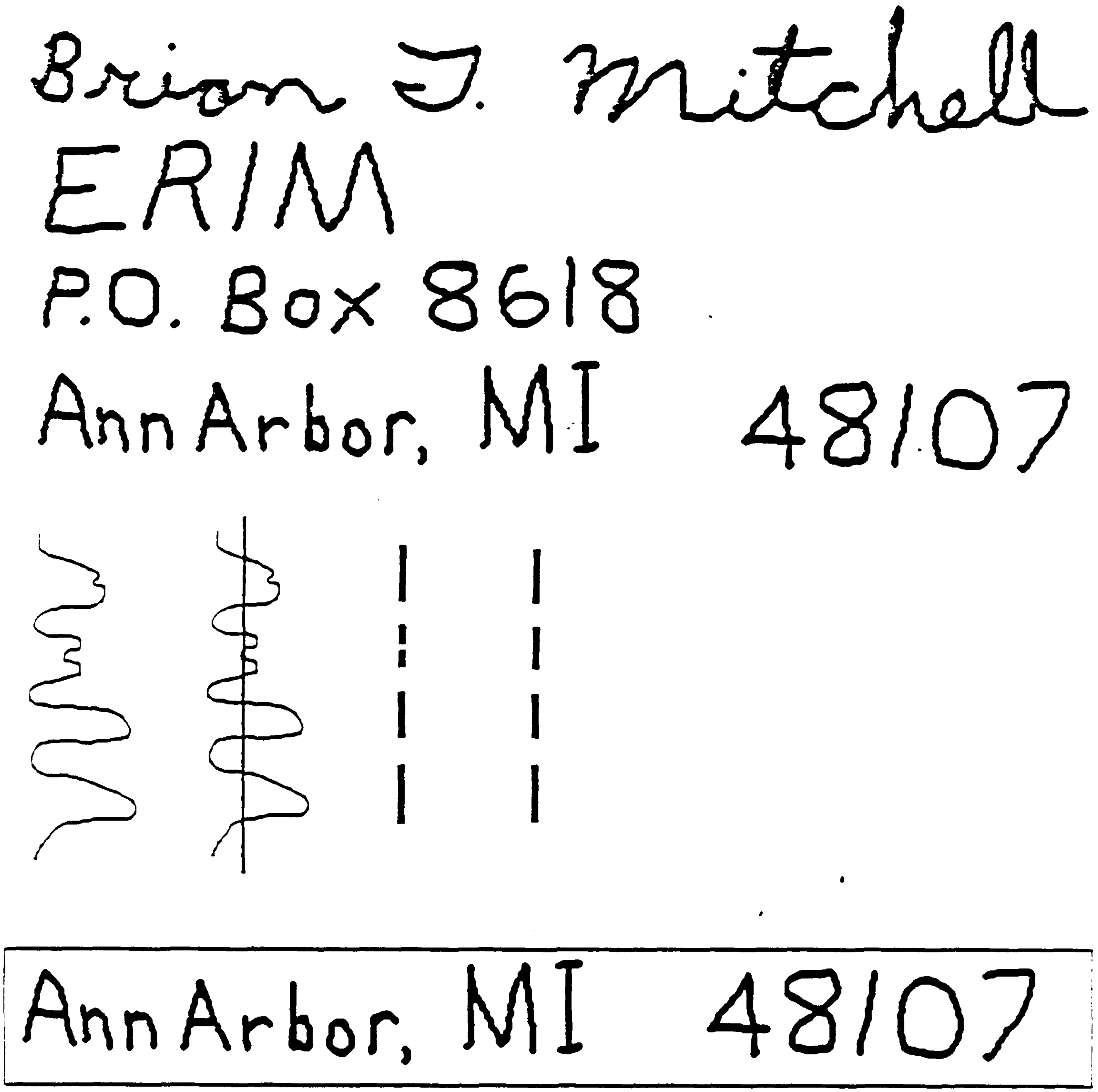


Figure 6. Last Line Detection

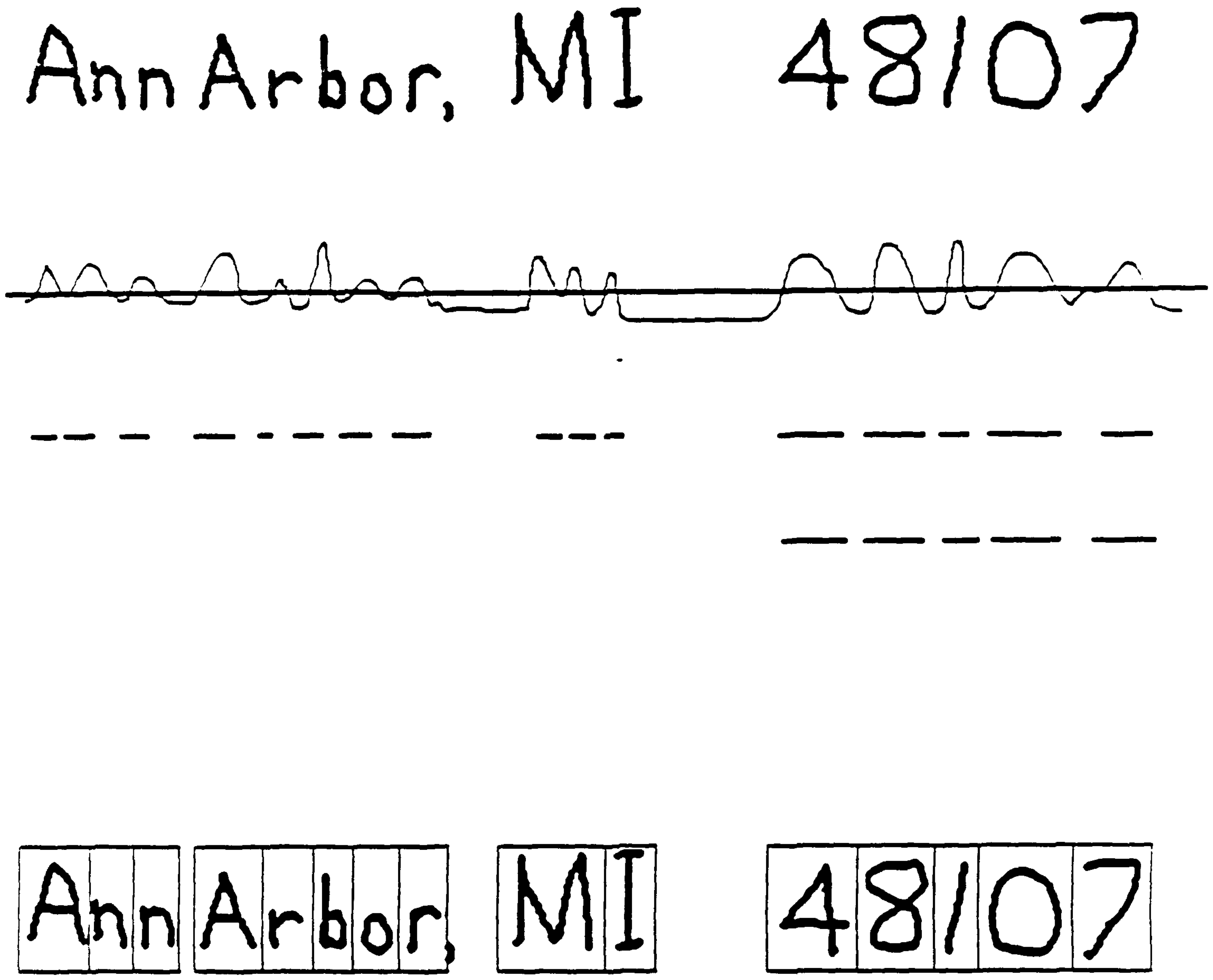


Figure 7. Character Segmentation

are developed in this process: north-cavities, south cavities, east-cavities, west-cavities, center-cavities, and holes. The north-cavities are defined as contiguous sets of pixels in the image that would not hit a stroke if moved vertically in a northward (upward) direction but would hit a stroke if moved in a vertically southward (downward), horizontally eastward (right), and horizontally westward (left) direction. The south-cavities, east-cavities, and west-cavities are similarly defined. The holes are defined as contiguous sets of pixels in the image that are completely encircled by stroke pixels. The center-cavities are defined as contiguous sets of pixels in the image that are not holes but from which any of the four movements would hit a stroke. These features are represented in the state-labeled image that results from this processing phase as tagged pixels in which each grey-level value corresponds to a particular feature. Figure 9 shows a pictorial representation of the features that would result for various representations of digit six.

2.4.5 Feature Segmentation

The features are then extracted from the state-labeled image and placed into a composite symbolic map. A segmenter was implemented on the Symbolics LISP machine in ZetaLISP to perform this task. An overview of the functionality of the segmenter is graphically represented in Figure 10. In this figure, a state-labeled image for a typical six is segmented into its various feature components: strokes and cavities. The segmentation process is straightforward. First, the state-labeled feature image is scanned from top to bottom and left to right to locate simply connected components of contiguous pixels with the same state or grey value. Next, the same state, simply connected components which touch are assembled into more complex components. Finally, the same-state, connected components are identified and placed into the composite map as objects with specified run-length regions.

2.4.6 Model Matching

The digits are next identified in the composite map by matching models to the extracted features. Two major efforts were performed to implement this capability in the prototype system: a model matcher was implemented and

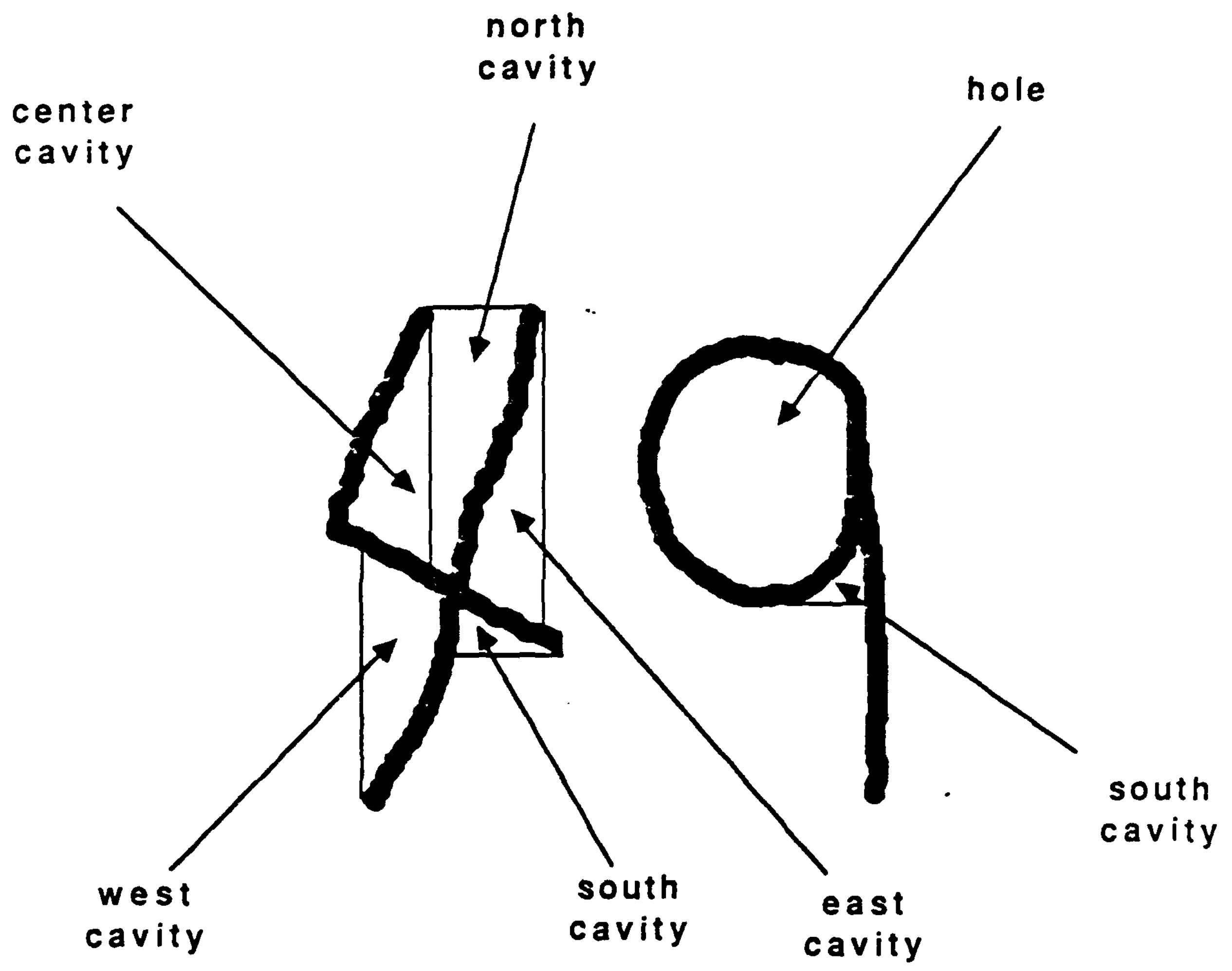


Figure 8. Cavity Features

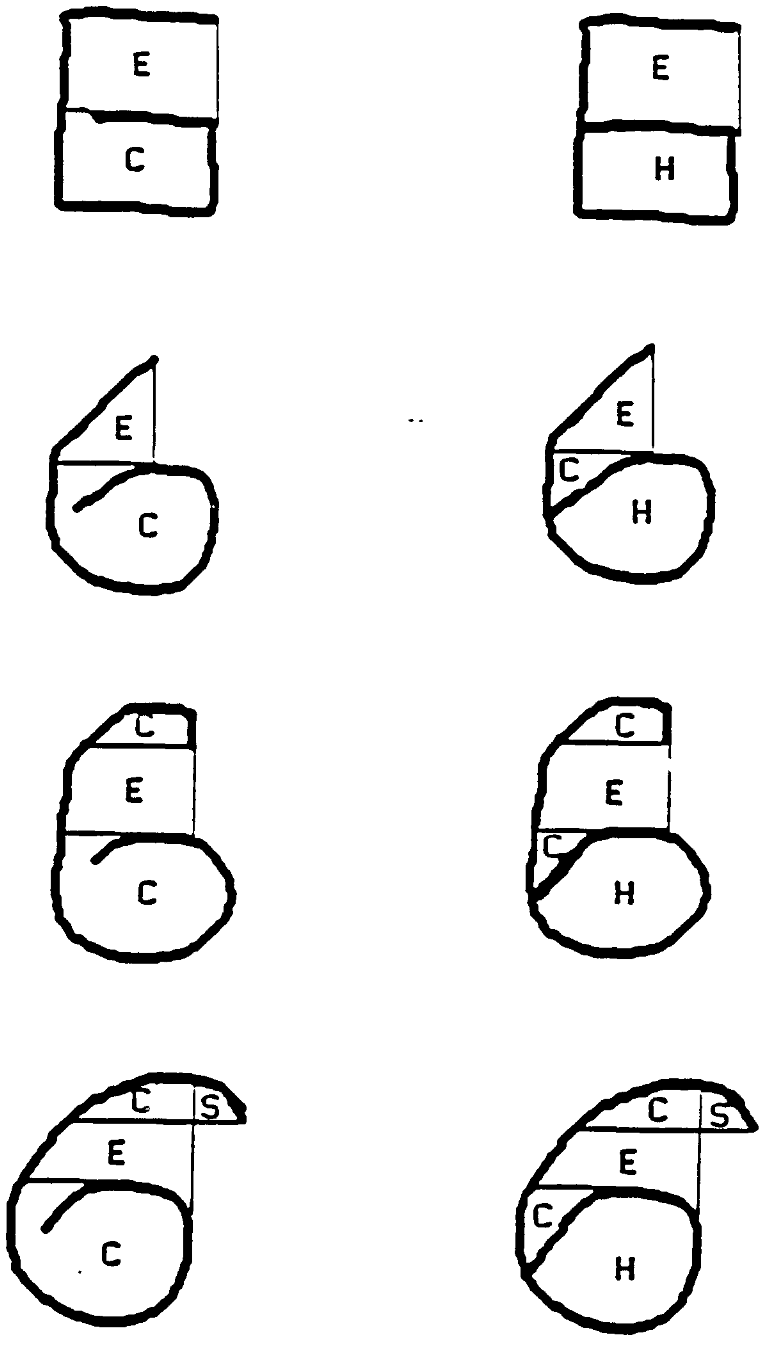


Figure 9. Typical Sixes

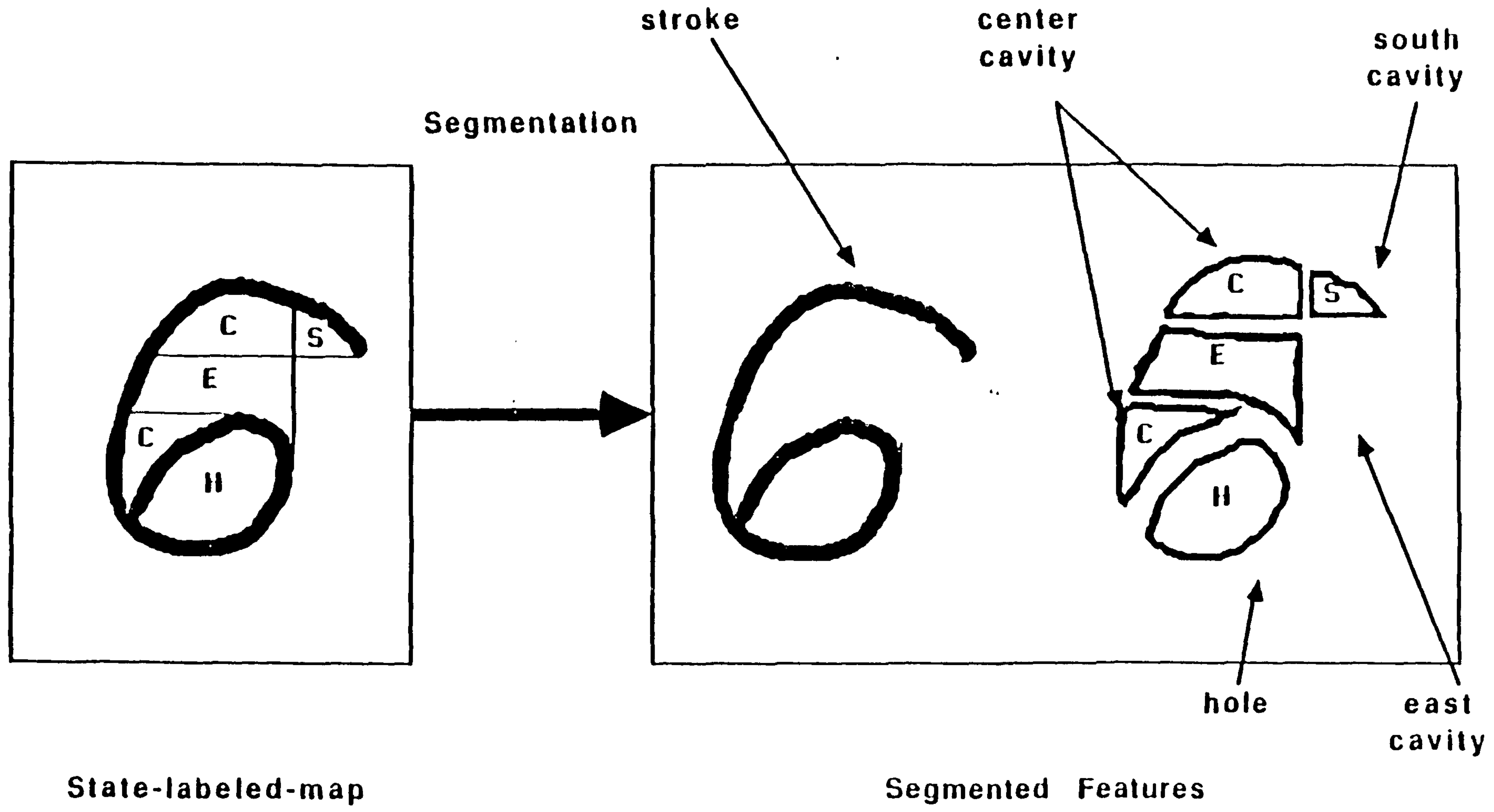


Figure 10. Segmentation

initial digit models were developed.

The model matcher was developed on the Symbolics LISP machine in ZetaLISP. It implements the testbed model matching language described above. As depicted in Figure 11 the match operates like a large sieve to filter out match candidates. An overview of the matching process for a typical six characterization is illustrated in Figure 12. In this figure matching is represented in a graphical form. Within the graph the undirected edges (no arrows) represent logical components. Thus, a six is made up of a top, a bottom, and in some instances an extra part. The solid directed edges (arrows) in the graph represent actual matching of the logical component to a feature in the composite map. Thus, the six bottom must match either a center cavity or a hole. Also seen in the figure are relations and tests that constrain the match. These components of the matching process test individual feature attributes and spatial relationships between features that must exist in a successful match. The figure also illustrates the hierarchical nature of the matcher which enables building complex models from simpler models. Thus, in the case of the six, a top submodel can be developed and used to build the six model.

The initial digit models were developed in the model matching language. Figure 13 shows an example of a fully developed six model. All the models developed in this phase were based on the cavity features described above. These features were used to build concavity models, complexes of touching cavities. The concavity and cavity models were then collectively used to develop digit models for each of the ten digits. Additional digit models were developed for the digits that had more that one characteristic representation. For example, several different models were developed for the digit two, including the loop-two model, the non-loop-two model, and the lazy-two model. Each represents a different morphology of the two concept, and as such is explicitly modeled. Throughout the development of the models, sample images were used to identify new digit models and to test matching performance.

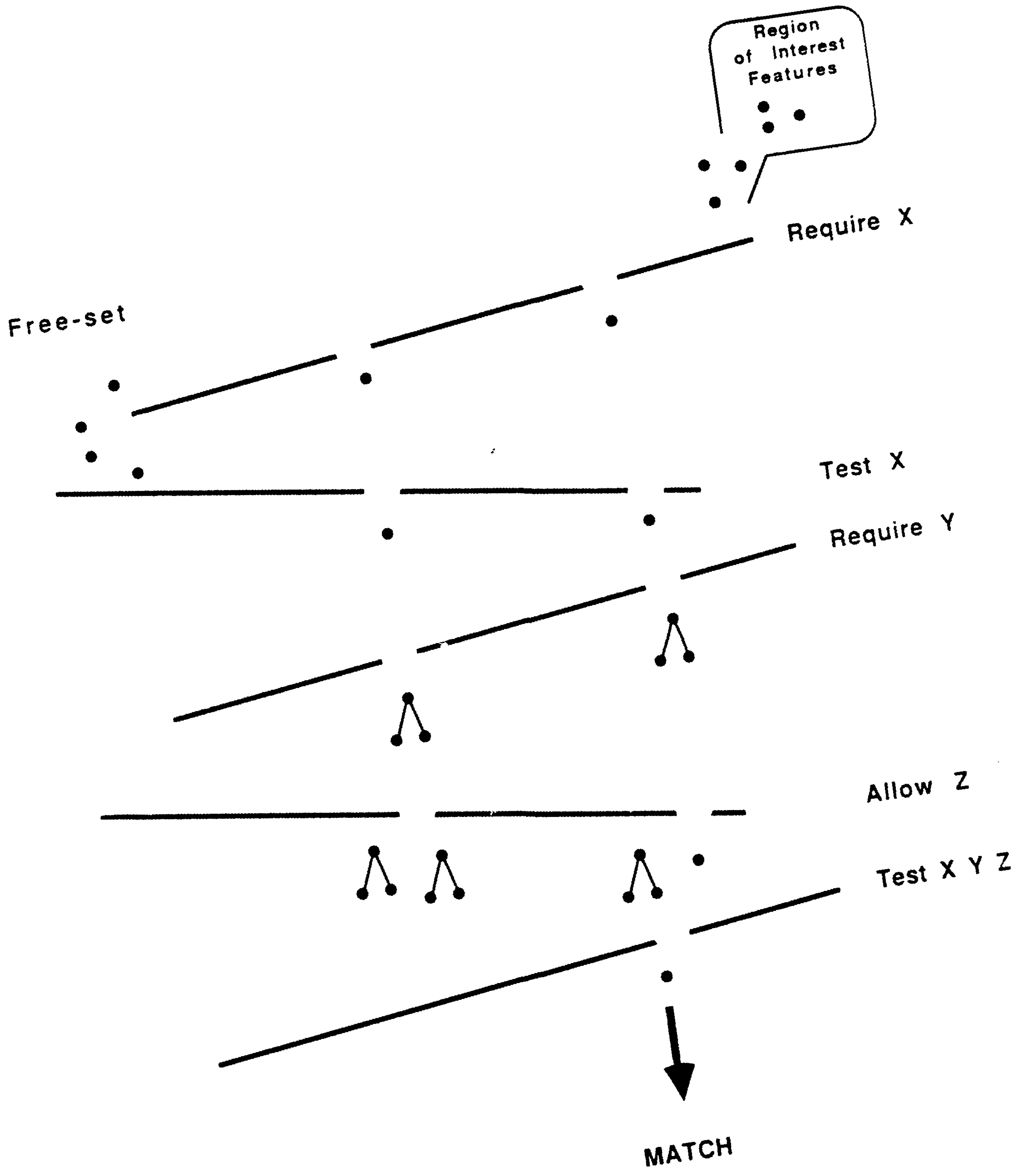


Figure 11. MATCHING PROCESS

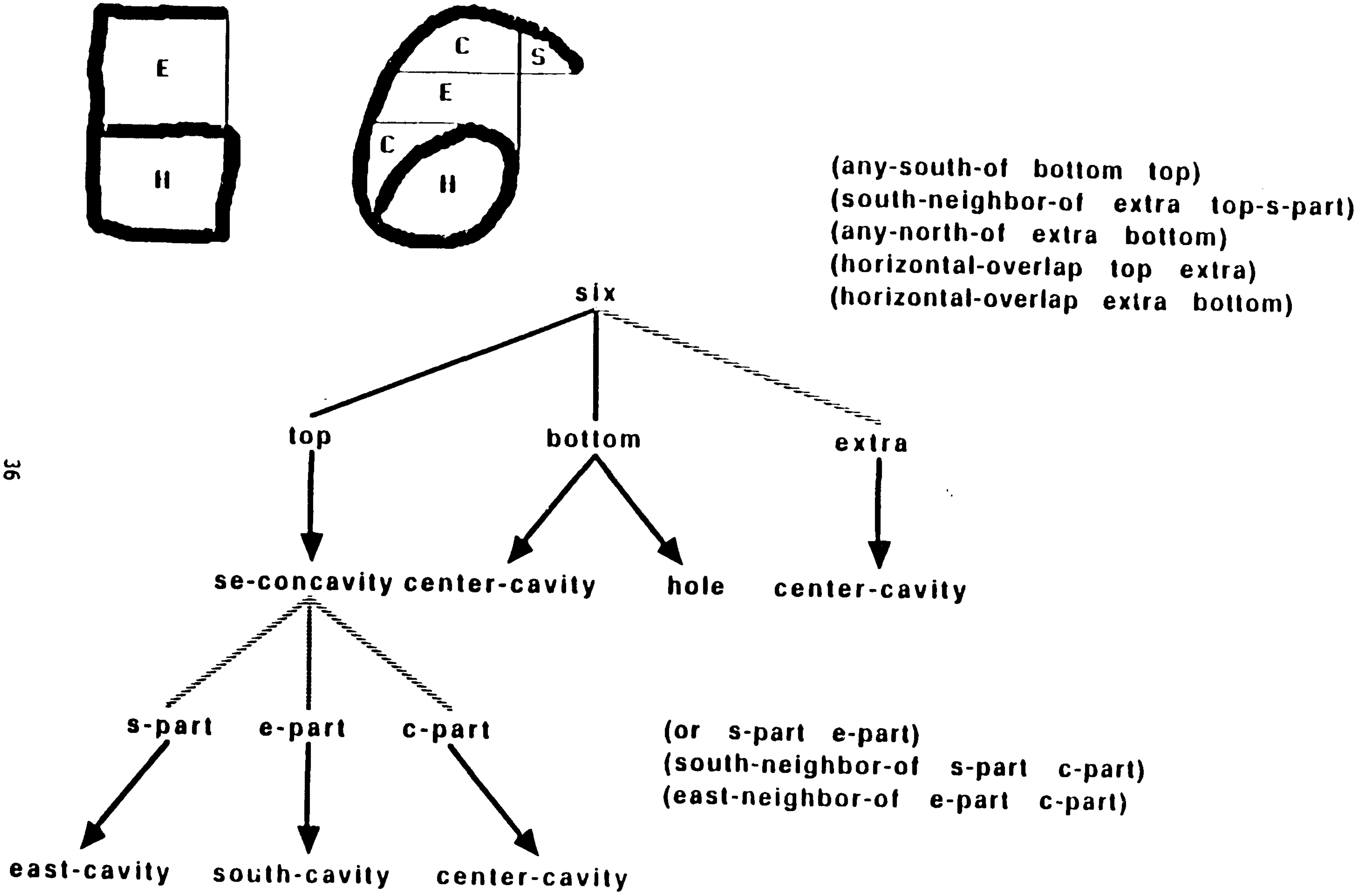

E
H
C
S
E
C
H
six
top
bottom
extra
se-concavity
center-cavity
hole
center-cavity
s-part
e-part
c-part
east-cavity
south-cavity
center-cavity
(any-south-of bottom top)
(south-neighbor-of extra top-s-part)
(any-north-of extra bottom)
(horizontal-overlap top extra)
(horizontal-overlap extra bottom)
(or s-part e-part)
(south-neighbor-of s-part c-part)
(east-neighbor-of e-part c-part)

Figure 12. Matching Digit Six

```
(defmodel six
      (:color :orange)
    (ignore small-ones)
    (require window digit-feature)
    (require character stroke)
    (allow character2 stroke)
    (forbid character2)
    (require top se-concavity)
    (require bottom center-cavity or hole)
    (test (any-south-of bottom top)
    (allow extra center-cavity)
    (bind top-s-part (sub-part top '>s-part))
    (if (and extra top-s-part)
          (test (south-neighbor-of extra top-s-part)))
    (ordered any-north-of extra bottom)
    (pairwise horizontal-overlap top extra bottom)
    (and-nothing-else))
```

```
(defmodel se-concavity
    ()
   (allow s-part east-cavity)
   (allow c-part center-cavity)
   (allow e-part south-cavity)
   (test (or s-part e-part))
   (ordered south-neighbor-of s-part c-part)
   (ordered east-neighbor-of e-part c-part))
```

Figure 13. Typical Digit Model

2.4.7 ZIP Code Assembly

Finally, the results of the matching process is used to determine the ZIP Code. Again, a simple approach was implemented. This approach is summarized in Figure 14. As seen in this figure, the digit matcher is run on all hypothesized digit regions. The results of this matching are then merged to assemble the ZIP Code. If five-digit regions possess the spatial qualities of a legal ZIP Code and if each of the five regions match a single digit model, then the five digits are assembled into a ZIP Code. If all five digits were not identified by the matcher, then the results of the matcher are reported. No further attempt is currently made, however, to assemble the ZIP Code from the partial matching data plus additional contextual information.

2.5 Discussion of Results

The Phase I activities in *Advanced Research in Recognition of Handwritten Address* ZIP *Codes* have been very fruitful. As discussed above, it has produced four major outcomes: the OCR literature review, the handwritten address digitized image data base, the hardware and software testbed for investigating the recognition of handwritten addresses, and the prototype ZIP Code recognition system. These four components create a solid foundation upon which our future research in this area can be conducted in an efficient and effective manner.

The OCR literature review has produced significant insight into past approaches to character recognition. This insight has already played a major role in formulating the overall approach to this problem. Several different approaches have been made to selected portions of the problem in the past. Many of these approaches were seemingly successful, but none were conducted in the context of an end-to-end system. Because of this, Phase I researchers decided to develop a testbed concept that would allow rapid development and continued refinement of a prototype ZIP Code recognition system. This concept thus allows continued integration of different approaches to various aspects of the recognition problem, including techniques which may be developed by other USPS contractors.

The handwritten address digitized image data base contains over 800

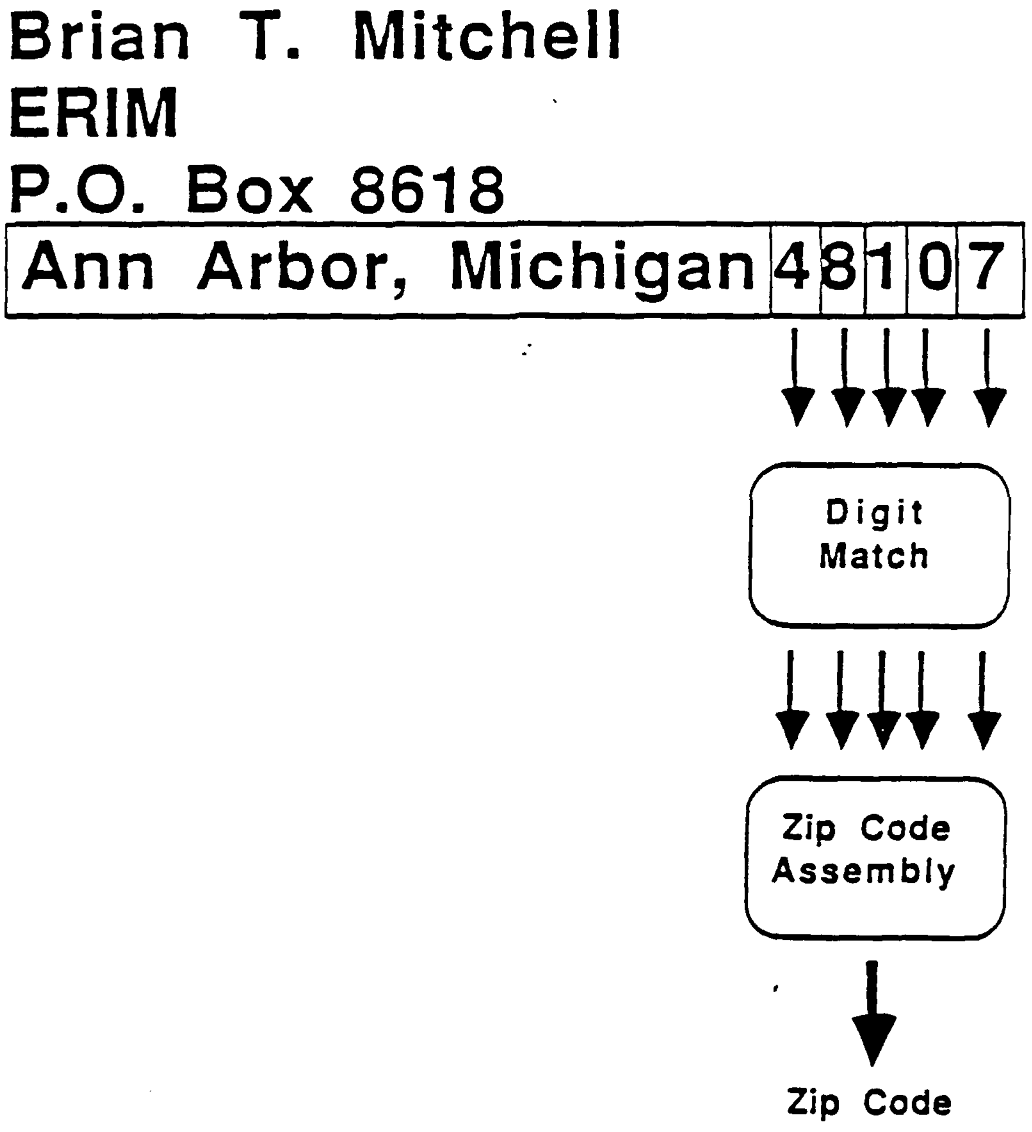


Figure 14. Zip Code Assembly

diverse address images and continues to grow. SUNY at Buffalo has provided a great majority of the digitized images. Their address sources include the USPS main office, local post offices, and and USPS OCR contractors. This data base has facilitated the construction and initial testing of the end-to-end ZIP Code reading system. Numeral models have been generated and refined using the ZIP Codes contained in this data base. Using the methodology developed for this project, the models can be quickly and easily evolved as the data base becomes larger. This supports a form of digit learning where the acquired knowledge is complete and precisely describable and rigorous.

A hardware and software testbed was designed and implemented. It consists of both hardware and software components. Featured within the testbed are an ERIM Cytocomputer and a Symbolics LISP machine. These specialized computers provide the computational pixel and symbolic throughput required to effectively develop and test recognition algorithms. Residing in these computers are several layers of software designed to minimize the effort required in developing vision algorithms. Together the hardware and software components of the testbed create a powerful development environment for exploring solutions to ZIP Code recognition.

An end-to-end prototype ZIP Code recognition system was developed. This prototype consists of several processing phases: binary image generation, last line extraction, character segmentation, feature generation, feature segmentation, model matching, and ZIP Code assembly. An initial solution to each processing phase was developed within this prototype system. Some of these solutions are recognized as simplistic. However, their implementation enables focused refinement of a complete recognition system. The software as of Phase I completion is included in Appendices A and B.

These four Phase I outcomes establish a solid foundation of Phase II research. The OCR literature review provides significant insight into steering the research effort in directions that promise the most success. The image data base provides the mechanism for testing and evaluating potential problem solutions. The address understanding testbed enables future work to be focused on problem solutions with minimal effort expended on implementation issues. The end-to-end prototype system establishes a baseline system in which each phase of the overall ZIP Code recognition process can be evaluated and further developed. Together these Phase I outcomes form the basis for accelerated Phase II research.

3 Phase II Overview

IN Phase II of the Advanced Research in Recognition of Handwritten Address ZIP Codes the performance capabilities of the prototype Phase I system will be extended. An overview of the methodology that will be used during this phase is seen in Figure 15. As seen in the figure, this methodology is iterative in nature. The OCR literature review, the handwritten address image data base, and the testbed will all play an integral part in this process. The literature review will serve to generate and evaluate new research ideas. Selected ideas will then be rapidly developed and integrated into the prototype system using the powerful tools within testbed environment. The performance of the extended system will then be evaluated on images from the handwritten image data base. The results of this evaluation will then be used to focus the next iteration of refinement.

3.1 Technical Approach

The testbed described above is used in Phase II to refine the prototype ZIP Code recognition system. In Phase I the hardware and software address recognition testbed was constructed. In Phase II this testbed will be used to explore intricacies and develop solutions. An iterative refinement methodology will be used in this process. Under this methodology, a set of test images will be processed through the current version of the prototype ZIP Code recognition system. This processing will produce performance data which can then be analyzed and used to focus subsequent research directions. By using this methodology, the Phase II research can be focused on areas that show most promise in upgrading the overall system performance. Thus, the research can be directed in such a way as to produce optimal results.

Several research directions are already known to be prime candidates for exploration in this phase of the research. Included in the list of topics that are known to require further attention are generating binary images,

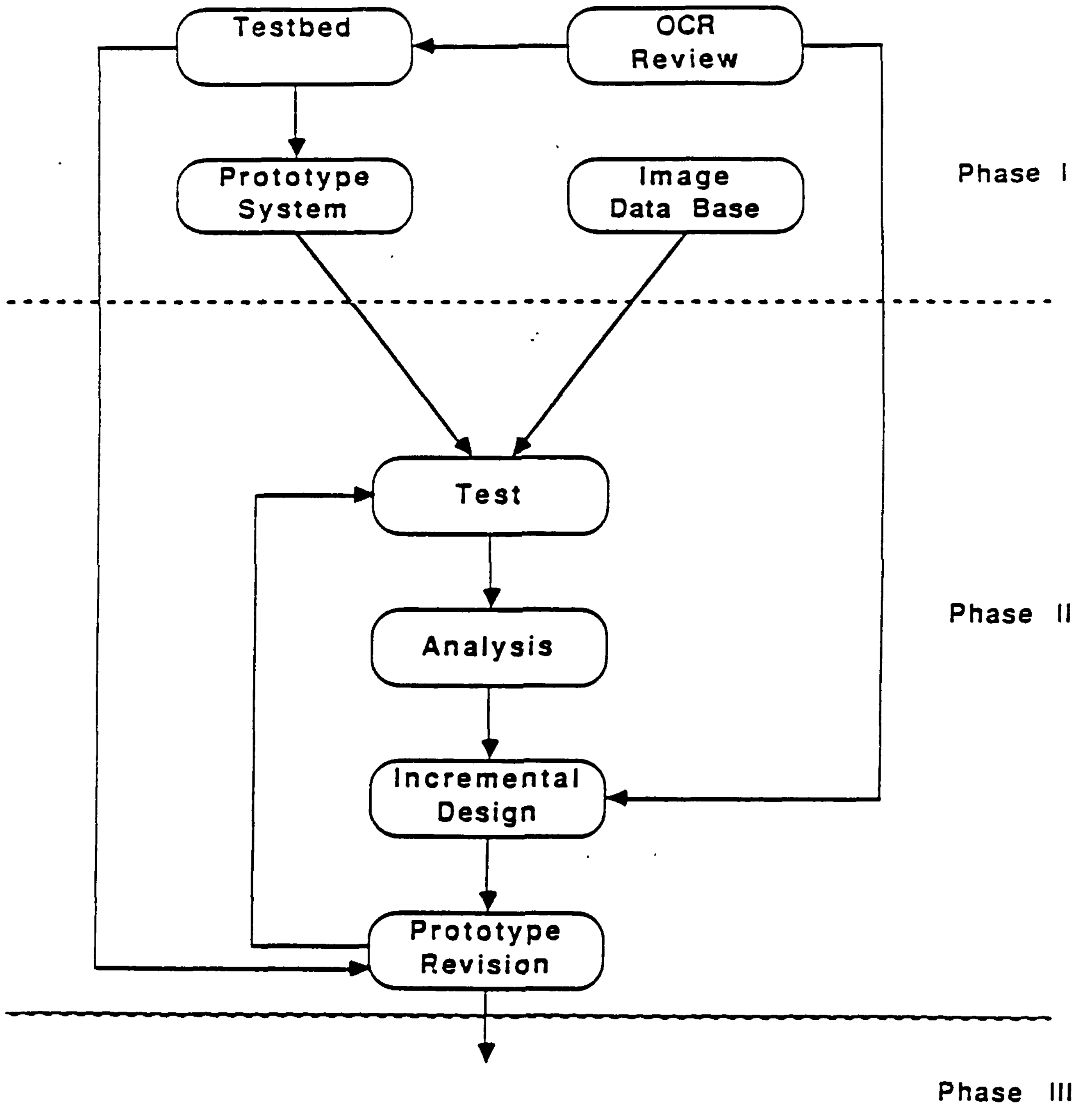


Figure 15. R & D Methodology

extracting slanted address lines, segmenting touching digits, windowing digits with overlapping bounding boxes, refining and expanding digit models, adding last line data base information into ZIP Code assembly, and adding context information into ZIP Code hypotheses formulation. Approaches are currently being developed to address these problems. Additional issues are also expected to arise throughout this research phase. Once identified, these issues will be analyzed and prioritized based the expected return on expended effort.

It is anticipated that each processing phase of the prototype system will be significantly enhanced during Phase II. Much of the anticipated work is outlined in the short subsections that follow. Although this work is described in some detail, it does not include all of the anticipated Phase II activities, since new activities are expected to arise as continued software development provides insight into advanced recognition system requirements. This is the nature of research and development of this kind.

3.1.1 Binary Image Generation

During Phase II this processing phase will be enhanced by developing new techniques to handle images with textured envelopes and broken characters.

The current prototype solution performs adequately on most images. It does not, however, perform well on images with textured backgrounds. Several techniques exist in the computer vision literature for identifying textured surfaces. These techniques will be explored and evaluated for performance on this problem. Several experiments will be conducted, and the best resulting technique will be integrated into the final system. One candidate technique that appears very promising is to identify and remove very short disconnected line segments from the image. An initial view of the test images suggests that this simple technique may eliminate much of this problem.

The current prototype solution also creates a significant number of broken characters. Some of these characters are broken because they are written with two disconnected strokes. The five with a flying top in a prime example. These problems must be modeled and identified by the matcher. Other characters, however, are broken because of the difference in intensity within the stroke. Initial review of these test data shows that looped two's are prime examples of this phenomenon. This is apparently caused by the fact that one must decelerate the writing instrument to go around the loop, thus creating a different intensity. This change in intensity is magnified when part of the stroke is eliminated during thresholding. This will be a problem for any simpleminded approach to creating the binary image. Further analysis of this problem is required during Phase II.

3.1.2 Last Line Extraction

During Phase II this processing phase will be enhanced by developing new techniques to handle slanted address lines. The current prototype histogramming technique fails on address blocks with slanted lines. Several possible solutions to this problem are technically feasible. One technique that offers much promise is model matching. Under this approach a ZIP block model will be developed to locate regions on the address block that seem to possess ZIP Code spatial qualities. This approach will be implemented and tested early on in Phase II.

3.1.3 Character Segmentation

During Phase II this processing phase will be enhanced by developing new techniques for isolating touching and intersecting ZIP Code characters. The current prototype solution performs well on segmenting characters that have no bounding box overlap. It also works well on segmenting touching characters that are simply connected by a single stroke. Unfortunately, there are numerous other cases in which this approach does not work well. Approaches to segmenting touching characters have been presented in the literature. These approaches appear to have some promise, although they are far from offering a comprehensive solution. Experiments on these approaches will be conducted, and application specific enhancements will be explored. From this work, one technique will be selected for integration into the final system.

3.1.4 Feature Generation

During Phase II feature generation will be enhanced. The current prototype solution incorporates only cavity features even though skeleton and endpoint features are computed by the Cytocomputer. Although concavity features appear to be very powerful, they are limited in what they can represent. At present, it appears that character recognition rates between 70 and 90 percent are possible with these features. Higher recognition rates, however, will require incorporation of the additional features which provide specific information about the character stroke. Because of the flexibility built into the testbed, this implementation will be fairly straightforward, thus facilitating online development through experimentation.

3.1.5 Feature Segmentation

There are no current plans to modify the prototype feature segmentation algorithm. However, it is possible that modifications may be necessary if data structures other than currently supported run-length regions become computationally desirable.

3.1.6 Model Matching

There will be significant modifications to the digit models during this project phase. The current prototype models perform as expected, and produce character recognition rates between 70 and 90 percent. They are however, extremely limited in some areas. Additional features will be developed to resolve these limitations. These features will then be incorporated into the digit models. Throughout Phase II, this aspect of the the overall system is expected to focus the overall research directions. During this process, the test images will play an important role in surfacing research issues and measuring performance.

3.1.7 ZIP Code Assembly

There will be several enhancements to ZIP Code assembly during Phase II of this project. The current prototype solution is simplistic in nature. It only looks for five-digit character sequences. It does not check the last line data base to see if a proposed ZIP Code exists. It does not use the last line data base to resolve missing information. It does not use additional contextual address block information to augment the digit matching process. These limitations will be addressed during this phase of the project. Solutions to the first two areas are fairly straightforward. and will be implemented in the obvious manner. The last area, however, will require significant thought and effort to identify techniques for reliably locating and identifying contextual sources of information. The methodology that will be employed here will be to study the content of the address block to identify areas of possible exploitation. Promising areas will then be analyzed from an image processing perspective to determine the reliability of required feature extraction. Those areas that look most promising will then be implemented and evaluated for performance on actual data.

3.2 Management Plan

It is expected that Phase II will be a continue the work of Phase I. The current research team will remain intact and will continue to work full time on this project.

4 Conclusions

TWO major topics in Advanced Research in Recognition of Handwritten Address ZIP Codes have been discussed in this report. The first discusses the work performed under Phase I of this effort in which four subjects were explored. These subjects are a review of the optical character recognition literature, the development of the address block image data base, the development of a testbed for developing computer vision solutions to address block understanding, and the development of a prototype ZIP Code recognition system. The second major topic describes the planned Phase II activities under this project. Under this topic a technical approach and management plan was outlined that proposed continued iterative refinement of the prototype system as the major Phase II activity. The research team feels that there is still much work that needs to be done on this problem. They also feel that the approach that has been developed will prove to be significant and will result in a positive research outcome.

5 References

1 USPS Request for Proposal, RFI-BL-004 BOA Task 8.

2 Smith, S.T., Schrader, M.E., Mitchell, B.T., Gillies, A.M., and Jacobus, C.J., "Methodologies of Optical Character Recognition," ERIM Technical Report ARVD 87-085.

3 Wallich, P., "Minis and Mainframes," IEEE Spectrum, January, 1985.

Appendix A

Digit Models

```
HSC000$DUA1:[MITCHEL.ZENO]IMI.LSP;1          21 MAY 1987 15:52          Page 1

;;; -*- Mode: LISP; Syntax: Zetalisp; Base: 10 -*-

(setq *release-string* "release 1")

(defun small-ones (x)
  (<= (area x) 2))

(defun huggers (x)        ;thin cavities which hug strokes
  (or (<= (height x) 1) (<= (width x) 1)))

(defun endpoint-features (x)
  (equal :endpoint (send x :get-attribute :type)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ZIP BLOCK MODELS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel raw-stroke () primitive)

(defmodel zip-code-block
          (:color :red)
  (require d1 raw-stroke)
  (test () (height d1) (* 0.5 (width d1))))
  (require d2 raw-stroke)
  (test () (height d2) (* 0.5 (width d2))))
  (ordered any-west-of d1 d2)
  (test (< (- (left d2) (right d1)) (min (height d1) (height d2))))
  (require d3 raw-stroke)
  (test () (height d3) (* 0.5 (width d3))))
  (ordered any-west-of d2 d3)
  (test (< (- (left d3) (right d2)) (min (height d2) (height d3))))
  (require d4 raw-stroke)
  (test () (height d4) (* 0.5 (width d4))))
  (ordered any-west-of d3 d4)
  (test (< (- (left d4) (right d3)) (min (height d3) (height d4))))
  (require d5 raw-stroke)
  (test () (height d5) (* 0.5 (width d5))))
  (ordered any-west-of d4 d5)
  (test (< (- (left d5) (right d4)) (min (height d4) (height d5)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DIGIT MODELS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defvar zero)
(defvar one)
(defvar two)
(defvar three)
(defvar four)
(defvar five)
(defvar six)
(defvar seven)
(defvar eight)
(defvar nine)
(defvar reject)

(defmodel digit-window   () primitive)
(defmodel stroke         () primitive)
(defmodel hole           () primitive)
(defmodel north-cavity   () primitive)
(defmodel south-cavity   () primitive)
(defmodel east-cavity    () primitive)
(defmodel west-cavity    () primitive)
```

```
HSC000$DUA1:[MITCHEL.ZENO]IMI.LSP;1          21 MAY 1987 15:52          Page 2

(defmodel center-cavity () primitive)

(defmodel feature
          ()
  (require region hole north-cavity south-cavity east-cavity west-cavity center

(defmodel 3flaws
          ()
  (allow flaw1 feature)
  (if flaw1 (test (<= (area (sub-part flaw1 '>region)) 1)))
  (allow flaw2 feature)
  (if flaw2 (test (<= (area (sub-part flaw2 '>region)) 1)))
  (allow flaw3 feature)
  (if flaw3 (test (<= (area (sub-part flaw3 '>region)) 1)))
  (pairwise neq flaw1 flaw2 flaw3))

(defmodel w-concavity
          ()
  (require main west-cavity)
  (allow t-part center-cavity)
  (ordered north-neighbor-of t-part main)
  (allow b-part center-cavity)
  (ordered south-neighbor-of b-part main)
  (allow tail north-cavity south-cavity)
  (if tail (test (> (* 3.0 (area main)) (area tail))))
  (if tail (test (or (and t-part (west-neighbor-of tail t-part))
                     (and b-part (west-neighbor-of tail b-part))))))

(defmodel e-concavity
          ()
  (require main east-cavity)
  (allow t-part center-cavity)
  (ordered north-neighbor-of t-part main)
  (allow b-part center-cavity)
  (ordered south-neighbor-of b-part main)
  (allow tail north-cavity south-cavity)
  (if tail (test (> (* 2.0 (area main)) (area tail))))
  (if tail (test (or (and t-part (east-neighbor-of tail t-part))
                     (and b-part (east-neighbor-of tail b-part))))))

(defmodel n-concavity
          ()
  (require main north-cavity)
  (allow l-part center-cavity)
  (ordered west-neighbor-of l-part main)
  (allow r-part center-cavity)
  (ordered east-neighbor-of r-part main)
  (allow tail east-cavity west-cavity)
  (if tail (test (> (* 4.0 (area main)) (area tail))))
  (if tail (test (or (and l-part (north-neighbor-of tail l-part))
                     (and r-part (north-neighbor-of tail r-part))))))

(defmodel s-concavity
          ()
  (require main south-cavity)
  (allow l-part center-cavity)
  (ordered west-neighbor-of l-part main)
  (allow r-part center-cavity)
  (ordered east-neighbor-of r-part main)
  (allow tail east-cavity west-cavity)
  (if tail (test (> (* 2.0 (area main)) (area tail))))
  (if tail (test (or (and l-part (south-neighbor-of tail l-part))
                     (and r-part (south-neighbor-of tail r-part))))))
```

```
(defmodel se-concavity
          ()
  (allow s-part east-cavity)
  (allow c-part center-cavity)
  (allow e-part south-cavity)
  (test (or s-part e-part))
  (if (and s-part e-part) (test c-part))
  (ordered south-neighbor-of s-part c-part)
  (ordered east-neighbor-of  e-part c-part)
;  (if s-part (eval (send s-part :highlight :blue)))
;  (if c-part (eval (send c-part :highlight :blue)))
;  (if e-part (eval (send e-part :highlight :blue)))
  )

(defmodel ne-concavity
          ()
  (allow n-part east-cavity)
  (allow c-part center-cavity)
  (allow e-part north-cavity)
  (test (or n-part e-part))
  (if (and n-part e-part) (test c-part))
  (ordered north-neighbor-of n-part c-part)
  (ordered east-neighbor-of  e-part c-part)
;  (if n-part (eval (send n-part :highlight :red)))
;  (if c-part (eval (send c-part :highlight :red)))
;  (if e-part (eval (send e-part :highlight :red)))
  )

(defmodel nw-concavity
          ()
  (allow n-part west-cavity)
  (allow c-part center-cavity)
  (allow w-part north-cavity)
  (test (or n-part w-part))
  (if (and n-part w-part) (test c-part))
  (ordered north-neighbor-of n-part c-part)
  (ordered west-neighbor-of  w-part c-part)
;  (if n-part (eval (send n-part :highlight :green)))
;  (if c-part (eval (send c-part :highlight :green)))
;  (if w-part (eval (send w-part :highlight :green)))
  )

(defmodel sw-concavity
          ()
  (allow s-part west-cavity)
  (allow c-part center-cavity)
  (allow w-part south-cavity)
  (test (or s-part w-part))
  (if (and s-part w-part) (test c-part))
  (ordered south-neighbor-of s-part c-part)
  (ordered west-neighbor-of  w-part c-part)
;  (if s-part (eval (send s-part :highlight :purple)))
;  (if c-part (eval (send c-part :highlight :purple)))
;  (if w-part (eval (send w-part :highlight :purple)))
  )

(defmodel complete-n-concavity
          ()
  (require middle north-cavity)
  (require l-part center-cavity)
  (ordered west-neighbor-of l-part middle)
  (require r-part center-cavity)
  (ordered east-neighbor-of r-part middle)
  (and-nothing-else))
```

```
;;; zero ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel zero
          (:color :pink :alias 0)
  (require it open-zero closed-zero)
  (bind window (sub-part it '>window))
  (eval (send window :highlight :pink)))

(defmodel closed-zero
          (:color :pink :alias 0)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark-green))
  (require character stroke)
  (require middle hole)
  (allow extra east-cavity or west-cavity)
  (if extra (test (< (area-ratio extra middle) 0.05)))
  (and-nothing-else)
  (eval (send window :highlight :pink)))

(defmodel open-zero
          (:color :pink :alias 0)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark-green))
  (require character stroke)
  (require middle complete-n-concavity)
  (allow extra east-cavity or west-cavity)
  (if extra (test (< (area-ratio extra middle) 0.05)))
  (and-nothing-else)
  (eval (send window :highlight :pink)))

;;; one ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel one
          (:color :yellow :alias 1)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark-green))
  (require character stroke)
  (test (> (height character) (* 1.0 (width character))))
  (allow slight-curve east-cavity west-cavity)
  (if slight-curve (test (< (width slight-curve) (* 0.1 (height character)))))
  (and-nothing-else)
  (eval (send window :highlight :yellow)))

;;; two ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel two
          (:color :red :alias 2)
  (require it non-loop-two or loop-two)
  (bind window (sub-part it '>window))
  (eval (send window :highlight :red)))

(defmodel non-loop-two
          (:color :red :alias 2)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark-green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require l-part w concavity)
  (require r-part e concavity)
  (bind left-area (float (area l part)))
  (bind right-area (float (area r-part)))
  (test (< (// (south-area-of l part r-part) left-area) 0.15))
  (test (< 0.5 (// left-area right-area) 10.0))
  (bind l-main (sub-part l-part '>main))
  (bind l-bot  (sub part l-part '>b-part))
  (if (and l-main l-bot) (test (< (area-ratio l-bot l-main) 1.0)))
  (allow b part south-cavity)
  (ordered generally-south-of b part l-part)
  (ordered generally south of b part r-part)
  (allow top-loop hole)
  (ordered any-south-of l-part top-loop)
  (if top-loop (test (< (area-ratio top-loop l-part) 1.0)))
  (ignore huggers)
  (and-nothing-else)
  (eval (send window :highlight :red)))

(defmodel loop-two
          (:color :red :alias 2)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require l-part w-concavity)
  (require loop hole)
  (allow r-part e-concavity)
  (bind left-area (float (area l-part)))
  (if r-part (bind right-area (float (area r-part))))
  (if r-part (test (< (// (south-area-of l part r-part) left-area) 0.15)))
  (if r-part (test (< 0.5 (// left-area right-area))))
  (bind l-main (sub-part l-part '>main))
  (bind l-bot  (sub-part l-part '>b part))
  (if (and l-main l-bot) (test (< (area-ratio l-bot l-main) 1.0)))
  (allow b-part south-cavity)
  (ordered any-north-of l-part loop)
  (ordered any-east-of r-part loop)
  (ordered generally-west-of loop b-part)
  (ordered generally-south-of b part l part)
  (ordered generally-south-of b part r part)
  (allow top-loop hole)
  (ignore huggers)
  (and-nothing-else)
  (ordered any-south-of l-part top-loop)
  (if top-loop (test (< (area ratio top-loop l-part) 1.0)))
  (if top-loop
      (test (> (// ( - (top loop) (bottom top loop)) (float (height character)))
  (eval (send window :highlight :red)))

;;; three ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel three
          (:color :green :alias 3)
  (require it single west three or double west three or west only three)
  (bind window (sub part it '>window))
  (eval (send window :highlight :green)))
```

```
(defmodel single west three
          (:color :green :alias 3)
  (ignore endpoint-features)
  (require window digit window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require r-part east-cavity)
  (require l-part w-concavity)
  (bind r-box  (bounding box r-part))
  (bind r-up   (list (left r-box) *-infinity* (right r box) (top r-box)))
  (bind r down (list (left r-box) (bottom r-box) (right r-box) *+infinity*))
  (test (and (> (area-within l-part r down) 0) (> (area within l-part r-up) 0)))
  (test (< (* 1.5 (height r-part)) (height l-part)))
  (allow extra center cavity)
  (if extra (test (vertical-within extra (sub-part l part '>main))))
  (ignore small-ones)
  (and-nothing-else)
  (eval (send window :highlight :green)))

(defmodel west-only-three
          (:color :green :alias 3)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require l-part w-concavity)
  (require extra center-cavity)
  (test (vertical-within extra (sub part l-part '>main)))
  (bind l-top-area (float (north-area of l part extra)))
  (bind l-bot area (float (south area of l part extra)))
  (test (and (> l-bot-area 0) (< 0.40 (// l-top-area l-bot-area) 2.0)))
  (test (< (height-ratio extra l-part) 0.20))
  (ignore small-ones)
  (and-nothing-else)
  (eval (send window :highlight :green)))

(defmodel double-west-three
          (:color :green :alias 3)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require r part east-cavity)
  (require t part w concavity)
  (ordered any north of t-part r part)
  (require b part w concavity)
  (ordered any south of b part r part)
  (ignore small ones)
  (and nothing else)
  (eval (send window :highlight :green)))

;;; four ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel four
          (:color :blue :alias 4)
  (ignore endpoint features)
  (require window digit window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require t-part open-four-top north-cavity)
  (test (> (* 5.0 (height t-part)) (height character)))
  (require b-part four-bottom)
  (ordered any-south-of b-part t-part)
  (allow side e-concavity)
  (allow extra south-cavity)
  (if extra (test (> (height-ratio character extra) 6.0)))
  (ordered any-north-of t-part extra)
  (ordered any-south-of b-part extra)
  (ignore small-ones)
  (and-nothing-else)
  (eval (send window :highlight :blue)))

(defmodel open-four-top
         (:color :blue :alias 4)
  (require main center-cavity)
  (allow nc-part north-cavity)
  (ordered east-neighbor-of nc-part main)
  (allow ec-part east-cavity)
  (ordered north-neighbor-of ec-part main)
  (test (or ec-part nc-part))
  (if ec-part (test (> (// (+ (area main) (area nc-part)) (float (area ec-part))

(defmodel four-bottom
         (:color :blue :alias 4)
  (require main west-cavity)
  (allow cc-part center-cavity)
  (ordered north-neighbor-of cc-part main)
  (allow sc-part south-cavity)
  (ordered west-neighbor-of sc-part cc-part)
  (if sc-part (test (> (height main) (height sc-part))))
  (if cc-part (test (> (height main) (height cc-part)))))

;;; five ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel five
         (:color :white :alias 5)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark-green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require l-part w-concavity)
  (require r-part five-top)
  (ordered any-north-of r-part l-part)
  (test (< 0.3 (area-ratio l-part r-part) 3.0))
  (ignore huggers)
  (ignore small-ones)
  (and-nothing-else)
  (eval (send window :highlight :white)))

(defmodel five-top
         ()
  (require main east-cavity)
  (allow t-part center-cavity)
  (ordered north-neighbor-of t-part main)
  (allow b-part center-cavity)
  (ordered south-neighbor-of b-part main)
  (if b-part (test (< (height-ratio b-part main) 0.25))))
```

```
;;; six ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel six
         (:color :orange :alias 6)
  (require it se-six or ne-six)
  (bind window (sub-part it '>window))
  (eval (send window :highlight :orange)))

(defmodel se-six
         (:color :orange :alias 6)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require t-part se-concavity)
  (require b-part center-cavity or hole)
  (test (any-south-of b-part t-part))
  (allow extra center-cavity)
  (bind top-s-part (sub-part t-part '>s-part))
  (if (and extra top-s-part)
      (test (south-neighbor-of extra top-s-part)))
  (ordered any-north-of extra b-part)
  (pairwise horizontal-overlap t-part extra b-part)
  (ignore huggers)
  (and-nothing-else)
  (eval (send window :highlight :orange)))

(defmodel ne-six
         (:color :orange :alias 6)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
  (require t-part ne-concavity)
  (require b-part center-cavity or hole)
  (test (any-north-of t-part b-part))
  (test (any-west-of t-part b-part))
  (allow extra center-cavity)
  (bind top-s-part (sub-part t-part '>s-part))
  (if (and extra top-s-part)
      (test (south-neighbor-of extra top-s-part)))
  (ordered any-north-of extra b-part)
  (pairwise horizontal-overlap t-part extra b-part)
  (ignore huggers)
  (and-nothing-else)
  (eval (send window :highlight :orange)))

;;; seven ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel seven
         (:color :purple :alias 7)
  (ignore small-ones)
  (ignore endpoint-features)
  (require window digit-window)
  (eval (send window :highlight :dark green))
  (require character stroke)
  (allow character2 stroke)
  (forbid character2)
```

```
 (require cave w7-concavity)
 (test (> (// (float (width character)) (height character)) 0.4))
 (test (> (height-ratio cave character) 0.25))
 (ignore huggers)
 (and-nothing-else)
 (eval (send window :highlight :purple)))

(defmodel w7-concavity
         ()
 (require main west-cavity)
 (allow t-part center-cavity)
 (ordered north-neighbor-of t-part main)
 (allow tail south-cavity)
 (ordered west-neighbor-of tail t-part))

;;; eight ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel eight
         (:color :brown :alias 8)
 (ignore small-ones)
 (ignore endpoint-features)
 (require window digit-window)
 (eval (send window :highlight :dark-green))
 (require character stroke)
 (allow character2 stroke)
 (forbid character2)
 (require b-part hole)
 (require t-part hole or ne-concavity or open-east-eight-top)
 (ordered strictly-south-of b-part t-part)
 (allow l-part west-cavity)
 (ordered any-south-of b-part l-part t-part)
 (allow r-part east-cavity)
 (ordered any-south-of b-part r-part t-part)
 (allow extra north-cavity)
 (ordered any-east-of extra t-part)
 (ordered any-north-of extra r-part)
 (and-nothing-else)
 (eval (send window :highlight :brown)))

(defmodel open-east-eight-top
         ()
 (require t-part center-cavity)
 (require m-part east-cavity)
 (require b-part center-cavity)
 (ordered north-neighbor-of t-part m-part b-part)
 (allow extra north-cavity)
 (ordered east-neighbor-of extra b-part))

;;; nine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel nine
         (:color :violet :alias 9)
 (ignore small-ones)
 (ignore endpoint-features)
 (require window digit-window)
 (eval (send window :highlight :dark-green))
 (require character stroke)
 (allow character2 stroke)
 (forbid character2)
 (require t-part hole)
 (require b-part sw-concavity)
 (test (any-north-of t-part b-part))
 (bind bot-w-part (sub-part b-part '>w-part))
 (if bot-w-part (test (< (south-area-of bot-w-part t-part) (* 0.2 (area t-part)
```

```
 (allow extra center-cavity)
 (bind bottom-n-part (sub-part b-part '>n-part))
 (if (and extra bottom-n-part)
     (test (north-neighbor-of extra bottom-n-part)))
 (ordered any-south-of extra b-part)
 (pairwise horizontal-overlap t-part extra b-part)
 (allow extra-east east-cavity)
 (if extra-east (test (< (width-ratio extra-east character) 0.3)))
 (ordered any-south-of extra-east t-part)
 (and-nothing-else)
 (eval (send window :highlight :violet)))
```

Appendix B

Feature Extraction

```
;** - xcaveline in - process from raw image to caves of last line
; in - input filename or number
procedure(in)
; global variable cave receives final result
gdeclare cave
; get the image and process to binary image
xgetbin in
; get the last line in a fixed image size
xgetfixedline
; seperate the characters on the last line
xseparate
; detect features in the separated characters
xprocseps
; get the result from global variable cave
copy cave
endprocedure
```

```
;** xgetbin im slice - get and slice an image -> bin (global)
; im - input filename or number
; slice - threshold
procedure (im,slice)
gray
setdef 15->slice  ; default threshold = 15
gdeclare bin      ; global variable to get binary image result
declare code      ; variable to hold stage programs
declare temp      ; temporary image storage

; get the raw image from disk
unsave im
; save it in temp
copy ->temp
gdeclare raw
copy ->raw
; estimate background level by closing raw by a cylendar of radius 20
loadcode 'closecyl20.noc' -> code
apply code
empty -> code
; subtract raw image from the background image
diffimages active temp  -> active
; threshold giving fg (strokes) state 5, and backgroung state 0
gdeclare diffimg
copy -> diffimg
slice 0 slice 0 5
color
; get rid of long horizontal things
spanv 0 5 2 110 40 ->,code
spanv 5 2 5 110 40 ->,code
cover 5 0 ->,code
cover 2 5 ->,code
apply code
; save binary image in global variable bin
copy ->bin
endprocedure
```

```
_HSC000$DUA1:[OCR.AMG.SYS]XGETFIXEDLINE.DEF;1  1-APR-1987 13:29 Page 1

;** xgetfixedline - get last line of address block in fixed image size
; assumes binary image with fg=5 bg=0
procedure(wheight,wwidth)
setdef 50->wheight        ; default image height = 50
setdef 400->wwidth        ; default image width = 400
declare ltop,lbot,lheight,extra,temp,left
gdeclare line,elw,eheight,ewidth,top,bot
; save image in temp
copy ->temp
; migrate pixels left to form accumulated width histogram
migleft
gdeclare lhist
copy -> lhist
; call lastline to find top and bottom of last line
xlastline 30->ltop,lbot
; compute fixed size window placement sa as not to go off image
lheight:=lbot-ltop
extra:=(wheight-lheight)/2
if (extra>0)
  ltop:=ltop-extra
endif
if (ltop>imglen-wheight)
  ltop:=imglen-wheight
endif
eheight:=wheight ; global variable eheight = window height (will be used
ewidth:=wwidth
top:=ltop
bot:=ltop+eheight
left:=imgwid-wwidth
; define the extended line window
window wheight wwidth top left -> elw
; get the binary image back
copy temp
; copy the last line window to global variable line
copy elw->line
; make the line image currently active
copy line
endprocedure
```

```
;** xlastline slop - find last line given "histogram" image
; slop - the amount to go from the left edge before cutting lines apart
procedure(slop)-> top,bot
; span in from left edge of "histogram" turning to state 2
spanv 0 5 2 10 slop
gdeclare lcut
copy ->lcut
; remove the leftmost part
cover 2 0
; span back toward right making fixed height boxes in state 2
spanv 5 0 2 100 slop+1
gdeclare lbox
copy ->lbox
; remove the rightmost parts of the histogram
cover 5 0
; merge lines which are very close together
spanv 2 0 2 2002 3
spanv 0 2 0 2002 3
gdeclare lmerge
copy ->lmerge
; eliminate leftover short lines
spanv 0 2 0 2002 3
spanv 2 0 2 2002 3
gdeclare lprune
copy->lprune
; make a very narrow window to speed up pixel scanning
declare lw
window imglen 1 1 1 -> lw
activate lw

; intro to loop to find top and bottom of each line in image
declare oldtop pix
iscan 2 2 1 1 -> top pix    ; this finds the top of the first line

; loop finding top and bottom of aech successive line
repeat
oldtop:=top
iscan 0 0 top pix -> bot pix
iscan 2 2 bot pix -> top pix
until (top=0)

; now we have top and bot values
top:=oldtop
; reactivate the binary image
activate scratch
endprocedure
```

```
;** - xseperate slop - seperate a line of text into single chars
; slop - height at which to cut histogram to seperate characters
procedure(slop)
setdef 4->slop   ; default slop value = 4
gdeclare eheight,seps
declare temp,code,trim,wind
trim:=10

; save the current image
copy -> temp
; migrate pixels down to from accumulated height "histogram"
migdown
gdeclare dhist
copy ->dhist
; span up from bottom amount slop turning to atate 2
spanv 0 5 2 2000 slop
gdeclare dcut
copy ->dcut
; remove bottom part of histogram
cover 2 0
; extend the remaining histogram to be full window in height
spanv 5 0 5 2002 eheight
gdeclare dbox
copy ->dbox
; mark windows narrow enough to merge in state 4
spanv 0 5 4 110 4
spanv 5 4 5 110 4
gdeclare dnarrow
copy ->dnarrow
; find narrow gaps with narrow windows on either side - state 2
spanv 4 0 3 10 5
spanv 4 3 2 100 5
; return wider gaps to state 0
cover 3 0
; turn narrow windows back to state 5
cover 4 5
; now turn all windows to state 2 (narrow gaps are already in state 2)
cover 5 2
gdeclare dmerge
copy ->dmerge
; make a non-zero background state to do skeletonizing
cover 0 1
; skeletonize background thus extending windows without merging them
skelrec8 1 2 3 off off
; retrun weindows to full height (lost in skeletonization)
spanv 1 2 1 2002 5
; return background to state 0
cover 1 0
gdeclare dthick
copy ->dthick
; add the original binary last line to windows image
addimages temp
gdeclare dchars
copy ->dchars
; now we have characters in windows but they extend beyond windows someti
; remove strokes which fall outside windows
cover 5 0
gdeclare wspill
```

```
copy ->wspill
; remove small strokes which touch borders of windows
spanr 0 7 1 trim
gdeclare wpoke
copy ->wpoke
spanr 7 1 7 (2*trim)
cover 1 2
gdeclare wtrim
copy ->wtrim
; trim all strokes to be al least 1 pixel away from the window's edge
spanr 0 7 2 1
cover 2 8
;remove windows which have no strokes in them
spanr 7 8 1 100
cover 8 0
cover 1 8
; copy the result to global variable seps
copy ->seps
endprocedure
```

```
;** - xprocseps - process seps image for ocr
; assume stroke=7 window=8 border=0
procedure(tl,np)
setdef 2 -> tl ; default trim-length 2
setdef 2 -> np ; default number of passes 2
gdeclare skel cave
; eliminate very small (single 2112) fg things
spanv 8 7 5 2112 1
spanr 7 5 7 30
cover 5 8
gdeclare wclean
copy ->wclean
; eliminate small bumps on fg objects
tranbx 7 7 8 7
; skeletonize the fg
skelrec4 7 3 1
skelrec8 7 3 10
cover 3 8
gdeclare wskel
copy ->wskel
; trim skeletons using the simplify function (assumes fg=2)
cover 7 2
markboth 2 3 4
gdeclare wmark
copy ->wmark
cover 3 2
cover 4 2
simplify tl np
; mark endpoints=3 and junctions=4
markboth 2 3 4
copy ->skel
; return fg to 7
cover 2 7
cover 3 7
cover 4 7
; thicken and 4-way skeletonize (this
tranbx 7 8 7 2100 1 off
tranbx 7 8 7 2010 1 off
skelrec4 7 8 2
gdeclare thickskel
copy ->thickskel
; detect concavities using the caves function
xcaves
; leave result in global variable cave
copy ->cave
endprocedure
```

```
_HSC000$DUA1:[OCR.AMG.WORK]MARKBOTH.DEF;5  20-JAN-1987 14:54   Page 1

;**- markboth fg ends juncs - mark endpoints and junctions in different s
procedure(fg,ends,juncs)
declare cache

tranbx fg fg ends 6 1 on # or
tranbx fg fg ends 3 1 on # or
tranbx fg fg ends 7 1 on # or
tranbx fg fg ends 1 1 on # or
tranbx fg fg ends 2 1 on #



copy ->cache
cover ends fg
markpoints fg juncs
pixelselect whereever (cache=ends) ends

endprocedure
```

```
;** - simplify - simplify 8-way connected skeletons
; assumes fg=2 bg=8
procedure(trim_length,npasses)
setdef 3 -> trim_length
setdef 4 -> npasses
declare fg bg endpoint jpoint temp i scode
fg:=2
bg:=8
endpoint:=3
jpoint:=4
temp:=5
  spanr endpoint fg temp trim_length ->,scode
  spanr fg temp fg (trim_length+2) ->,scode
  spanr temp endpoint temp 2 ->,scode
  cover temp bg ->,scode
  spanr jpoint endpoint bg 2 ->,scode
  cover endpoint fg ->,scode
  cover jpoint fg ->,scode
  skelrec8 fg bg 3 ->,scode
for i 1 npasses
  markboth fg endpoint jpoint
  apply scode
endfor
endprocedure
```

```
;** - xcaves - set up colors and call cavities to mark n s e w... cavitie
; assume stroke=7 window=8 border=0
; return stroke=240 window=0 border=10
; n=1 e=2 s=3 w=4 c=7 hole=6(orange)
procedure
declare code
cover 0 64 ->,code
cover 8 0 ->,code
cover 7 128 ->,code
apply code
cavities
empty ->code
cover 128 240
cover 5 7
cover 64 10
gdeclare wcave
copy ->wcave
spanv 0 7 orange 2112 30 ->,code
spanv 1 7 orange 2112 30 ->,code
spanv 2 7 orange 2112 30 ->,code
spanv 3 7 orange 2112 30 ->,code
spanv 4 7 orange 2112 30 ->,code
exch 7 orange ->,code
apply code
endprocedure
```

```
;** - cavities - mark concavities in numerals
; assume stroke=128 window=0 border=64
; n=1 e=2 s=3 w=4 c=5 (to view cover 5 130, cover 128 200)
procedure
gdeclare cavecode,cavecodeswitch
declare size
size:=100
setdef FALSE -> cavecodeswitch

if ( cavecodeswitch <> TRUE)  ; if cavecode does not already exist, make
cavecodeswitch := TRUE
spanv 128 0 1 2000 size ->,cavecode      ; span up in lowest bit plane
bitdisab 0
spanv 128 0 2 10 size ->,cavecode        ; span right in next plane
bitdisab 0,1
spanv 128 0 4 2 size ->,cavecode ; span down in next plane
bitdisab 0,1,2
spanv 128 0 8 100 size ->,cavecode       ; span left in final bitplane
bitmask 255

; using covers reduce the 16 state cavities to just 1,2,3,4,and 7
; for n,e,s,w and center cavities
cover 1 0 ->,cavecode
cover 2 0 ->,cavecode
cover 3 0 ->,cavecode
cover 4 0 ->,cavecode
cover 5 0 ->,cavecode
cover 6 0 ->,cavecode
cover 8 0 ->,cavecode
cover 9 0 ->,cavecode
cover 10 0 ->,cavecode
cover 12 0 ->,cavecode
cover 11 1 ->,cavecode
cover 7 2 ->,cavecode
cover 14 3 ->,cavecode
cover 13 4 ->,cavecode
cover 15 5 ->,cavecode

endif

apply cavecode 1 ; apply cavecode to mark cavities

endprocedure
```

```
_HSC000$DUA1:[OCR.AMG.TOOLS]HMIGRATE.DEF;3  27-MAY-1987 11:09  Page 1

;** hmigrate fore,passes,in - horizontal migration
; note: fore cannot be 0 (zero)
procedure (fore,passes,inimage ) -> outimage

setdef ACTIVE -> inimage
setret ACTIVE -> outimage
declare nlines,npixels
show_size inimage -> nlines,npixels
setdef 5 -> fore
setdef npixels -> passes

cover ~(fore) 0 inimage -> outimage
cover fore 2 inimage -> outimage

declare code
cover 0 1 ->,code
spanv 0 1 2 10 1 ->,code
cover 1 0 ->,code
bitor  1 0 ->,code
bitmask 1
max3d 000 200 000 1  ->,code
bitmask 255
bitor  1 2  ->,code
bitmask 4
max3d 000 002 000 1  ->,code
bitmask 255
cover 0 0 ->,code
cover 1 0 ->,code
cover 2 0 ->,code
cover 3 8 ->,code
cover 4 8 ->,code
cover 5 8 ->,code
cover 6 0 ->,code
cover 7 8 ->,code
cover 8 2 ->,code
apply code, passes, inimage -> outimage

cover 2 fore inimage -> outimage

endprocedure
```

```
;** vmigrate fore,passes,in - vertical migration
; note: fore cannot be 0 (zero)
procedure (fore,passes,inimage ) -> outimage

setdef ACTIVE -> inimage
setret ACTIVE -> outimage
declare nlines,npixels
show_size inimage -> nlines,npixels
setdef 5 -> fore
setdef nlines -> passes

cover ~(fore) 0 inimage -> outimage
cover fore 2 inimage -> outimage

declare code
cover 0 1 ->,code
spanv 0 1 2 2000 1 ->,code
cover 1 0 ->,code
bitor  1 0 ->,code
bitmask 1
max3d 000 000 020 1  ->,code
bitmask 255
bitor  1 2  ->,code
bitmask 4
max3d 020 000 000 1  ->,code
bitmask 255
cover 0 0 ->,code
cover 1 0 ->,code
cover 2 0 ->,code
cover 3 8 ->,code
cover 4 8 ->,code
cover 5 8 ->,code
cover 6 0 ->,code
cover 7 8 ->,code
cover 8 2 ->,code
apply code, passes, inimage -> outimage

cover 2 fore inimage -> outimage

endprocedure
```

```
_HSC000$DUA1:[OCR.AMG.TOOLS]MIGDOWN.DEF;1  18-MAR-1987 13:30   Page 1

;** migdown fore,passes,in - vertical migration using vcode
; note: fore cannot be 0 (zero)
procedure (fore,passes,inimage ) -> outimage

setdef ACTIVE -> inimage
setret ACTIVE -> outimage
declare nlines,npixels
show_size inimage -> nlines,npixels
setdef nlines -> passes
setdef 5->fore

cover ~(fore) 0 inimage -> outimage
cover fore 2 inimage -> outimage

gdeclare vcode
apply vcode, passes, inimage -> outimage

cover 2 fore inimage -> outimag

endprocedure
```

```
;** migleft fore,passes,in - horizontal migration using hcode
; note: fore cannot be 0 (zero)
procedure (fore,passes,inimage ) -> outimage

setdef ACTIVE -> inimage
setret ACTIVE -> outimage
declare nlines,npixels
show_size inimage -> nlines,npixels
setdef npixels -> passes
setdef 5->fore

cover ~(fore) 0 inimage -> outimage
cover fore 2 inimage -> outimage

gdeclare hcode
apply hcode, passes, inimage -> outimage

cover 2 fore inimage -> outimage

endprocedure
```

Appendix C

OCR Address Tape Processing

REPORT

OCR
SUNY Address Tape Processing

Francis Quek
7 April, 1987

OCR SUNY Address Tape Processing Report

F. Quek

April 7, 1987

This report describes the set of programmes and DCL command files which have been implemented to extract the OCR address label images from SUNY. Also included is information on how to make use of these programmes and command files.

The Data

The data we receive from SUNY is in a compressed (run-length encoded) format. Each image is preceded in the tape by a header file which describes the image. This header file is in ASCII and is formatted as follows :

```
image name
Number of Rows:       ####
Number of Columns:    ####
```

where

image name is a valid VMS file name

are integers specifying the number of rows and columns.

Data Extraction Procedure

The input tape has to be **MOUNT**ed using the *foreign* option and specifying the *blocksize* to be 16384. Data on the tape can then be read using the standard VMS **COPY** command. The mounting command is as follows:

```
$ MOUNT/foreign/bl=16384 MTAx
```

After reading the two files (header and image), the image must be first put into a fixed record format. This can be done using the VMS **CONVERT** facility. The necessary incantation is:

```
$ CONVERT/pad/fdl=FIXED.FDL Sourcefile Targetfile
```

FIXED.FDL is a VMS FDL declaration file the listing of which can be found in the appendix of this report.

The **COMPRESS.EXE** programme supplied by SUNY can now be run. This programme expects an input file name with a '.Z' extension. The output of the programme is a file of the same name with no extension. The programme can be run as follows:

```
$ COMPRESS -d filename
```

This will decompress the file *filename.Z* yielding a decompressed image file named *filename*..

The resulting image must be put in a fixed record format to be accessible from C4PL. The programme to do this is **FIXQ.EXE**. Besides putting the image into fixed record format, **FIXQ** also downsamples the image by a factor of 4 if the original image is larger than 512KByte in size (the limit of C4PL) and trims the image to make the dimensions even (the current implementation of C4PL has a bug which precludes operation with odd column size images). To execute this programme, type

```
$ FIXQ Sourcefile Targetfile rows columns
```

where *rows* and *columns* are integers specifying the number of rows and columns in the original image. This information is available in the header file described earlier.

Another version of **FIXQ** exists to facilitate the extraction of the images in unattended mode. **FIXQQ.EXE** takes as input the source image file and the header file. It reads the name of the target file, the number of rows and the number of columns form the header file. **FIXQQ** can be activated as follows:

```
$ FIXQQ imagefile headerfile
```

RDCOMP.COM is a VMS DCL file which permits the processing of the images in batch mode. It will read a tape image by image and process them, leaving the final image in the default directory. Several lines in **RDCOMP.COM** has to be altered for each run (to specify the physical tape drive on which the tape is mounted, the target directory of the images etc.). Instructions on the necessary changes are contained in the in code documentation of **RDCOMP.COM**

Required Files and Programmes

The required files and programmes are listed below for ease of reference.

RDCOMP.COM - The VMS DCL file which permits the processing of an entire tape in batch mode.

COMPRESS - The image compression/decompression programme provided by SUNY.

FIXQ - The programme which puts the image in a format readable by C4PL. downsampling and trimming the image as necessary.

FIXQQ - Similar to **FIXQ** except that it permits operation in unattended mode.

FIXED.FDL - The VMS File Description Language file to be used with the VMS **CONVERT** command.

The source code of the above are appended to this report.

```
HSC000$DUAL:[OCR COM]RDCOMP COM;53          6 APR 1987 15:59          Page 1

$!*************************************************************************
$!
$!      RDCOMP          Read an EKTRON/USPS compressed data tape.
$!
$!
$!              By P. Quek & S. Smith
$!                      ERIM, Ann Arbor, Michigan
$!
$!              Extensively modified from READCOMP (Prasanna G. Mulgaonkar
$!              SRI International, Menlo Park, California October 14, 1986)
$!
$!              Modifications :
$!
$!              P. Quek         3/18/87
$!              S. T. Smith     3/23/87 Added /pad to convert command.
$!
$!              This command procedure reads a set of compressed images in
$!              the EKTRON/USPS format and converts the format of the images
$!              into FIXED to conform to the input format required by the
$!              COMPRESS program. It then decompreases the images. The
$!              decompressed image is left on the current directory with
$!              the following file name: xxxxxx.IMG where xxxxxx is the
$!              image name contained in the header file preceding each input
$!              image in the tape.
$!              All images from a tape are processed.
$!
$!              Intermediate files created are:
$!
$!              data.tmp        raw data read from tape
$!              data.z          converted to fixed mode
$!              data.           decompressed file (out of COMPRESS)
$!              this.hdr        the header file which precedes each image.
$!
$!              These files are deleted after each xxxxx.img file is obtained.
$!
$!      CUSTOMIZATION :
$!              Several lines of this DCL command file needs to be altered
$!              to reflect the user's directory structure, physical tape
$!              device on which the tape is mounted and the number of
$!              files to skip on the tape.
$!              All lines which need to be altered for customization are
$!              marked with the comment 'QQQ Modify as necessary'
$!
$!
$!      REQUIRED FILES/PROGRAMMES
$!              FIXED.FDL       The FDL conversion declaration file to be
$!                              used with DCL CONVERT
$!              COMPRESS.EXE    The compression/decompression programme
$!              FIXQQ.EXE       The programme which puts the data into a
$!                              fixed length record format, downsamples the
$!                              image if it is too large, and ensures that all
$!                              images have even dimensions as required by C4PL.
$!
$!
$!      N B
$!      1       Another version of FIXQQ named FIXQ exists. FIXQ requires
$!              the user to specify the output file name, the number of
$!              rows and the number of columns and is more suitable for
$!              interactive terminal operation. FIXQQ makes an assumption
$!              of the format of the header file.
$!
$!      2.      The required file FIXED.FDL is created by the VAX file
$!              description language (FDL) utility. It specifies that the
$!              file is a "FIXED", "16384byte", NO carriage control file.
```

```
HSC000$DUAL:[OCR COM]RDCOMP COM;53          6 APR 1987 15:59          Page 2

$!              A sample FIXED.FDL file is:
$!
$!              IDENT "7 JUL 1986 17:49:39   VAX 11 FDL Editor"
$!
$!              RECORD
$!                      CARRIAGE CONTROL        none
$!                      FORMAT                  fixed
$!                      SIZE                    16384
$!
$!
$!      Note: This procedure does not rewind the tape to allow multiple
$!              image sets to be read from the same tape.
$!
$!*************************************************************************
$!
$       close/nolog f1                          ! was aborted with a ^Y
$       set def sd$1day:[ocr]                   ! QQQ Modify as necessary
$!
$!!!QQQ Modify as necessary.
$       compress :== $ UD$2:[OCR.PGM]compress   ! foreign command
$       fixed :== $ UD$2:[OCR.PGM]fixqq         ! Define foreign command
$       fixfdl :== UD$2:[OCR.PGM]fixed.fdl
$!
$       SrcTape :== MTA2:[]                     ! QQQ Modify as necessary
$       dbgcnt :== 0
$!
$       alloc 'SrcTape'
$       mount/for/block=16384/noassist 'SrcTape'
$!!!!!!
$!      set magtape/skip=file:272 'SrcTape'     ! QQQ Modify as necessary
$!!!!!!
$!
$loop:
$       copy 'SrcTape' this.hdr         ! Header directory
$       if  not  $status then goto donetape
$       copy 'SrcTape' data.tmp
$       convert/pad/fdl='fixfdl' data.tmp data.z
$       compress  d data
$       fixed data. this.hdr
$       delete data.tmp;*
$       delete data.z;*
$       delete data.;*
$       delete this.hdr;*
$       close f1
$!
$!      dbgcnt == dbgcnt+1
$!      if dbgcnt .gt. 10 then goto donetape
$!
$       goto loop
$!
$donetape:
$       write sys$output "Source Tape Complete"
$       close f1
$       dismount 'SrcTape'
$       dealloc 'SrcTape'
$       charges
$       exit
```

```
#define vax     1
#define vms     1
/*
 * Compress - data compression program
 */
#define min(a,b)        ((a>b) ? b : a)

/*
 * machine variants which require cc -Dmachine:  pdp11, z8000, pcxt
 */

/*
 * Set USERMEM to the maximum amount of physical user memory available
 * in bytes.  USERMEM is used to determine the maximum BITS that can be used
 * for compression.
 *
 * SACREDMEM is the amount of physical memory saved for others; compress
 * will hog the rest.
 */
#ifndef SACREDMEM
#define SACREDMEM       0
#endif

#ifndef USERMEM
# define USERMEM        450000  /* default user memory */
#endif

#ifdef interdata                /* (Perkin-Elmer) */
#define SIGNED_COMPARE_SLOW     /* signed compare is slower than unsigned */
#endif

#ifdef pdp11
# define BITS   12      /* max bits/code for 16-bit machine */
# define NO_UCHAR       /* also if "unsigned char" functions as signed char */
# undef USERMEM
#endif /* pdp11 */      /* don't forget to compile with -i */

#ifdef z8000
# define BITS   12
# undef vax             /* weird preprocessor */
# undef USERMEM
#endif /* z8000 */

#ifdef pcxt
# define BITS   12
# undef USERMEM
#endif /* pcxt */

#ifdef USERMEM
# if USERMEM >= (433484+SACREDMEM)
#  define PBITS 16
# else
#  if USERMEM >= (229600+SACREDMEM)
#   define PBITS        15
#  else
#   if USERMEM >= (127536+SACREDMEM)
#    define PBITS       14
#   else
#    if USERMEM >= (73464+SACREDMEM)
#     define PBITS      13
#    else
#     define PBITS      12
#    endif
#   endif
```

```
#  endif
# endif
# undef USERMEM
#endif /* USERMEM */

#ifdef PBITS            /* Preferred BITS for this memory size */
# ifndef BITS
#  define BITS PBITS
# endif BITS
#endif /* PBITS */

#if BITS == 16
# define HSIZE  69001           /* 95% occupancy */
#endif
#if BITS == 15
# define HSIZE  35023           /* 94% occupancy */
#endif
#if BITS == 14
# define HSIZE  18013           /* 91% occupancy */
#endif
#if BITS == 13
# define HSIZE  9001            /* 91% occupancy */
#endif
#if BITS <= 12
# define HSIZE  5003            /* 80% occupancy */
#endif

#ifdef M_XENIX                  /* Stupid compiler can't handle arrays with */
# if BITS == 16                 /* more than 65535 bytes - so we fake it */
#  define XENIX_16
# else
#  if BITS > 13                 /* Code only handles BITS = 12, 13, or 16 */
#   define BITS 13
#  endif
# endif
#endif

/*
 * a code_int must be able to hold 2**BITS values of type int, and also -1
 */
#if BITS > 15
typedef long int        code_int;
#else
typedef int             code_int;
#endif

#ifdef SIGNED_COMPARE_SLOW
typedef unsigned long int count_int;
typedef unsigned short int count_short;
#else
typedef long int          count_int;
#endif

#ifdef NO_UCHAR
 typedef char   char_type;
#else
 typedef        unsigned char   char_type;
#endif /* UCHAR */
char_type magic_header[] = { "\037\235" };      /* 1F 9D */

/* Defines for third byte of header */
#define BIT_MASK        0x1f
#define BLOCK_MASK      0x80
/* Masks 0x40 and 0x20 are free.  I think 0x20 should mean that there is
```

```
   a fourth header byte (for expansion)
*/
#define INIT_BITS 9                     /* initial number of bits/code */

/*
 * compress.c - File compression ala IEEE Computer, June 1984.
 *
 * Authors:     Spencer W. Thomas       (decvax!harpo!utah-cs!utah-gr!thomas)
 *              Jim McKie               (decvax!mcvax!jim)
 *              Steve Davies            (decvax!vax135!petsd!peora!srd)
 *              Ken Turkowski           (decvax!decwrl!turtlevax!ken)
 *              James A. Woods          (decvax!ihnp4!ames!jaw)
 *              Joe Orost               (decvax!vax135!petsd!joe)
 *              B. Gold / J. Potter     Ektron Applied Imaging
 *
 * $Header: compress.c,v 4.0 85/07/30 12:50:00 joe Release $
 * $Log:        compress.c,v $
 *
 * Revision 4.0a 86/05/07  12:20:00  B. Gold  &  J. Potter
 * Modified to run on VAX/VMS too.
 *
 * Revision 4.0  85/07/30  12:50:00  joe
 * Removed ferror() calls in output routine on every output except first.
 * Prepared for release to the world.
 *
 * Revision 3.6  85/07/04  01:22:21  joe
 * Remove much wasted storage by overlaying hash table with the tables
 * used by decompress: tab_suffix[1<<BITS], stack[8000].  Updated USERMEM
 * computations.  Fixed dump_tab() DEBUG routine.
 *
 * Revision 3.5  85/06/30  20:47:21  jaw
 * Change hash function to use exclusive-or.  Rip out hash cache.  These
 * speedups render the megamemory version defunct, for now.  Make decoder
 * stack global.  Parts of the RCS trunks 2.7, 2.6, and 2.1 no longer apply.
 *
 * Revision 3.4  85/06/27  12:00:00  ken
 * Get rid of all floating-point calculations by doing all compression ratio
 * calculations in fixed point.
 *
 * Revision 3.3  85/06/24  21:53:24  joe
 * Incorporate portability suggestion for M_XENIX.  Got rid of text on #else
 * and #endif lines.  Cleaned up #ifdefs for vax and interdata.
 *
 * Revision 3.2  85/06/06  21:53:24  jaw
 * Incorporate portability suggestions for Z8000, IBM PC/XT from mailing list.
 * Default to "quiet" output (no compression statistics).
 *
 * Revision 3.1  85/05/12  18:56:13  jaw
 * Integrate decompress() stack speedups (from early pointer mods by McKie).
 * Repair multi-file USERMEM gaffe.  Unify 'force' flags to mimic semantics
 * of SVR2 'pack'.  Streamline block-compress table clear logic.  Increase
 * output byte count by magic number size.
 *
 * Revision 3.0   84/11/27  11:50:00  petsd!joe
 * Set HSIZE depending on BITS.  Set BITS depending on USERMEM.  Unrolled
 * loops in clear routines.  Added "-C" flag for 2.0 compatibility.  Used
 * unsigned compares on Perkin-Elmer.  Fixed foreground check.
 *
 * Revision 2.7   84/11/16  19:35:39  ames!jaw
 * Cache common hash codes based on input statistics; this improves
 * performance for low-density raster images.  Pass on #ifdef bundle
 * from Turkowski.
 *
 * Revision 2.6   84/11/05  19:18:21  ames!jaw
```

```
 * Vary size of hash tables to reduce time for small files.
 * Tune PDP-11 hash function.
 *
 * Revision 2.5   84/10/30  20:15:14  ames!jaw
 * Junk chaining; replace with the simpler (and, on the VAX, faster)
 * double hashing, discussed within.  Make block compression standard.
 *
 * Revision 2.4   84/10/16  11:11:11  ames!jaw
 * Introduce adaptive reset for block compression, to boost the rate
 * another several percent.  (See mailing list notes.)
 *
 * Revision 2.3   84/09/22  22:00:00  petsd!joe
 * Implemented "-B" block compress.  Implemented REVERSE sorting of tab_next.
 * Bug fix for last bits.  Changed fwrite to putchar loop everywhere.
 *
 * Revision 2.2   84/09/18  14:12:21  ames!jaw
 * Fold in news changes, small machine typedef from thomas,
 * #ifdef interdata from joe.
 *
 * Revision 2.1   84/09/10  12:34:56  ames!jaw
 * Configured fast table lookup for 32-bit machines.
 * This cuts user time in half for b <= FBITS, and is useful for news batching
 * from VAX to PDP sites.  Also sped up decompress() [fwrite->putc] and
 * added signal catcher [plus beef in writeerr()] to delete effluvia.
 *
 * Revision 2.0   84/08/28  22:00:00  petsd!joe
 * Add check for foreground before prompting user.  Insert maxbits into
 * compressed file.  Force file being uncompressed to end with ".Z".
 * Added "-c" flag and "zcat".  Prepared for release.
 *
 * Revision 1.10  84/08/24  18:28:00  turtlevax!ken
 * Will only compress regular files (no directories), added a magic number
 * header (plus an undocumented -n flag to handle old files without headers),
 * added -f flag to force overwriting of possibly existing destination file,
 * otherwise the user is prompted for a response.  Will tack on a .Z to a
 * filename if it doesn't have one when decompressing.  Will only replace
 * file if it was compressed.
 *
 * Revision 1.9  84/08/16  17:28:00  turtlevax!ken
 * Removed scanargs(), getopt(), added .Z extension and unlimited number of
 * filenames to compress.  Flags may be clustered (-Ddvb12) or separated
 * (-D -d -v -b 12), or combination thereof.  Modes and other status is
 * copied with copystat().  -O bug for 4.2 seems to have disappeared with
 * 1.8.
 *
 * Revision 1.8  84/08/09  23:15:00  joe
 * Made it compatible with vax version, installed jim's fixes/enhancements
 *
 * Revision 1.6  84/08/01  22:08:00  joe
 * Sped up algorithm significantly by sorting the compress chain.
 *
 * Revision 1.5  84/07/13  13:11:00  srd
 * Added C version of vax asm routines.  Changed structure to arrays to
 * save much memory.  Do unsigned compares where possible (faster on
 * Perkin-Elmer)
 *
 * Revision 1.4  84/07/05  03:11:11  thomas
 * Clean up the code a little and lint it.  (Lint complains about all
 * the regs used in the asm, but I'm not going to "fix" this.)
 *
 * Revision 1.3  84/07/05  02:06:54  thomas
 * Minor fixes.
 *
 * Revision 1.2  84/07/05  00:27:27  thomas
 * Add variable bit length output
 *
 */
static char rcs_ident[] = "$Header: compress.c,v 4.0 85/07/30 12:50:00 joe Rele

#include <stdio.h>
#include <ctype.h>
#include <signal.h>
#include <types.h>
#include <stat.h>

#define ARGVAL() (*++(*argv) || (--argc && *++argv))

int n_bits;                          /* number of bits/code */
int maxbits = BITS;                  /* user settable max # bits/code */
code_int maxcode;                    /* maximum code, given n_bits */
code_int maxmaxcode = 1 << BITS;     /* should NEVER generate this code */
#ifdef COMPATIBLE               /* But wrong! */
# define MAXCODE(n_bits)        (1 << (n_bits) - 1)
#else
# define MAXCODE(n_bits)        ((1 << (n_bits)) - 1)
#endif /* COMPATIBLE */

#ifdef XENIX_16
count_int htab0[8192];
count_int htab1[8192];
count_int htab2[8192];
count_int htab3[8192];
count_int htab4[8192];
count_int htab5[8192];
count_int htab6[8192];
count_int htab7[8192];
count_int htab8[HSIZE-65536];
count_int * htab[9] = {
        htab0, htab1, htab2, htab3, htab4, htab5, htab6, htab7, htab8 };

#define htabof(i)       (htab[(i) >> 13][(i) & 0x1fff])
unsigned short code0tab[16384];
unsigned short code1tab[16384];
unsigned short code2tab[16384];
unsigned short code3tab[16384];
unsigned short code4tab[16384];
unsigned short * codetab[5] = {
        code0tab, code1tab, code2tab, code3tab, code4tab };

#define codetabof(i)    (codetab[(i) >> 14][(i) & 0x3fff])

#else   /* Normal machine */
count_int htab [HSIZE];
unsigned short codetab [HSIZE];
#define htabof(i)       htab[i]
#define codetabof(i)    codetab[i]
#endif  /* XENIX_16 */
code_int hsize = HSIZE;              /* for dynamic table sizing */
count_int fsize;

/*
 * To save much memory, we overlay the table used by compress() with those
 * used by decompress().  The tab_prefix table is the same size and type
 * as the codetab.  The tab_suffix table needs 2**BITS characters.  We
 * get this from the beginning of htab.  The output stack uses the rest
 * of htab, and contains characters.  There is plenty of room for any
 * possible stack (stack used to be 8000 characters).
 */
```

```
#define tab_prefixof(i) codetabof(i)
#ifdef XENIX_16
# define tab_suffixof(i)        ((char_type *)htab[(i)>>15])[(i) & 0x7fff]
# define de_stack               ((char_type *)(htab2))
#else   /* Normal machine */
# define tab_suffixof(i)        ((char_type *)(htab))[i]
# define de_stack               ((char_type *)&tab_suffixof(1<<BITS))
#endif  /* XENIX_16 */

code_int free_ent = 0;                  /* first unused entry */
int exit_stat = 0;

code_int getcode();

Usage() {
#ifdef DEBUG
fprintf(stderr,"Usage: compress [-dDVfc] [-b maxbits] [file ...]\n");
}
int debug = 0;
#else
fprintf(stderr,"Usage: compress [-dfvcV] [-b maxbits] [file ...]\n");
}
#endif /* DEBUG */
int nomagic = 0;        /* Use a 3-byte magic number header, unless old file */
int zcat_flg = 0;       /* Write output on stdout, suppress messages */
int quiet = 1;          /* don't tell me about compression */

/*
 * block compression parameters -- after all codes are used up,
 * and compression rate changes, start over.
 */
int block_compress = BLOCK_MASK;
int clear_flg = 0;
long int ratio = 0;
#define CHECK_GAP 10000 /* ratio check interval */
count_int checkpoint = CHECK_GAP;
/*
 * the next two codes should not be changed lightly, as they must not
 * lie within the contiguous general code space.
 */
#define FIRST   257     /* first free entry */
#define CLEAR   256     /* table clear output code */

int force = 0;
char ofname [100];
#ifdef DEBUG
int verbose = 0;
#endif /* DEBUG */
int (*bgnd_flag)();

int do_decomp = 0;

/*****************************************************************
 * TAG( main )
 *
 * Algorithm from "A Technique for High Performance Data Compression",
 * Terry A. Welch, IEEE Computer Vol 17, No 6 (June 1984), pp 8-19.
 *
 * Usage: compress [-dfvc] [-b bits] [file ...]
 * Inputs:
 *      -d:         If given, decompression is done instead.
 *
 *      -c:         Write output on stdout, don't remove original.
 *
 *      -b:         Parameter limits the max number of bits/code.
 *
 *      -f:         Forces output file to be generated, even if one already
 *                  exists, and even if no space is saved by compressing.
 *                  If -f is not used, the user will be prompted if stdin is
 *                  a tty, otherwise, the output file will not be overwritten.
 *
 *      -v:         Write compression statistics
 *
 *      file ...:   Files to be compressed.  If none specified, stdin
 *                  is used.
 * Outputs:
 *      file.Z:     Compressed form of file with same mode, owner, and utimes
 *      or stdout   (if stdin used as input)
 *
 * Assumptions:
 *      When filenames are given, replaces with the compressed version
 *      (.Z suffix) only if the file decreases in size.
 * Algorithm:
 *      Modified Lempel-Ziv method (LZW).  Basically finds common
 * substrings and replaces them with a variable size code.  This is
 * deterministic, and can be done on the fly.  Thus, the decompression
 * procedure needs no input table, but tracks the way the table was built.
 */

main( argc, argv )
register int argc; char **argv;
{
    int overwrite = 0;  /* Do not overwrite unless given -f flag */
    char tempname[100];
    char **filelist, **fileptr;
    char *cp, *rindex(), *malloc();
    struct stat statbuf;
    extern onintr(), oops();


    if ( (bgnd_flag = signal ( SIGINT, SIG_IGN )) != SIG_IGN ) {
        signal ( SIGINT, onintr );
        signal ( SIGSEGV, oops );
    }

#ifdef COMPATIBLE
    nomagic = 1;        /* Original didn't have a magic number */
#endif /* COMPATIBLE */

    filelist = fileptr = (char **)(malloc(argc * sizeof(*argv)));
    *filelist = NULL;

    if((cp = rindex(argv[0], '/')) != 0) {
        cp++;
    } else {
        cp = argv[0];
    }
    if(strcmp(cp, "uncompress") == 0) {
        do_decomp = 1;
    } else if(strcmp(cp, "zcat") == 0) {
        do_decomp = 1;
        zcat_flg = 1;
    }

#ifdef BSD4_2
    /* 4.2BSD dependent - take it out if not */
    setlinebuf( stderr );
```

```
#endif /* BSD4_2 */

    /* Argument Processing
     * All flags are optional.
     * D => debug
     * V => print Version; debug verbose
     * d => do_decomp
     * v => unquiet
     * f => force overwrite of output file
     * n => no header: useful to uncompress old files
     * b maxbits => maxbits.  If -b is specified, then maxbits MUST be
     *      given also.
     * c => cat all output to stdout
     * C => generate output compatible with compress 2.0.
     * if a string is left, must be an input filename.
     */
    for (argc--, argv++; argc > 0; argc--, argv++) {
        if (**argv == '-') {    /* A flag argument */
            while (*++(*argv)) {        /* Process all flags in this arg */
                switch (**argv) {
#ifdef DEBUG
                    case 'D':
                        debug = 1;
                        break;
                    case 'V':
                        verbose = 1;
                        version();
                        break;
#else
                    case 'V':
                        version();
                        break;
#endif /* DEBUG */
                    case 'v':
                        quiet = 0;
                        break;
                    case 'd':
                        do_decomp = 1;
                        break;
                    case 'f':
                    case 'F':
                        overwrite = 1;
                        force = 1;
                        break;
                    case 'n':
                        nomagic = 1;
                        break;
                    case 'C':
                        block_compress = 0;
                        break;
                    case 'b':
                        if (!ARGVAL()) {
                            fprintf(stderr, "Missing maxbits\n");
                            Usage();
#ifdef vms
                            exit(0);
#else
                            exit(1);
#endif
                        }
                        maxbits = atoi(*argv);
                        goto nextarg;
                    case 'c':
                        zcat_flg = 1;
                break;
            case 'q':
                quiet = 1;
                break;
            default:
                fprintf(stderr, "Unknown flag: '%c'; ", **argv);
                Usage();
#ifdef vms
                exit(0);
#else
                exit(1);
#endif
            }
        }
    }
    else {          /* Input file name */
        *fileptr++ = *argv; /* Build input file list */
        *fileptr = NULL;
        /* process nextarg; */
    }
    nextarg: continue;
  }

  if(maxbits < INIT_BITS) maxbits = INIT_BITS;
  if (maxbits > BITS) maxbits = BITS;
  maxmaxcode = 1 << maxbits;

  if (*filelist != NULL) {
    for (fileptr = filelist; *fileptr; fileptr++) {
        exit_stat = 0;
        if (do_decomp != 0) {                       /* DECOMPRESSION */
            /* Check for .Z suffix */
            if (strcmp(*fileptr + strlen(*fileptr) - 2, ".Z") != 0) {
                /* No .Z: tack one on */
                strcpy(tempname, *fileptr);
                strcat(tempname, ".Z");
                *fileptr = tempname;
            }
            /* Open input file */
            if ((freopen(*fileptr, "r", stdin)) == NULL) {
                    perror(*fileptr); continue;
            }
            /* Check the magic number */
            if (nomagic == 0) {
                if ((getchar() != (magic_header[0] & 0xFF))
                 || (getchar() != (magic_header[1] & 0xFF))) {
                    fprintf(stderr, "%s: not in compressed format\n",
                        *fileptr);
                continue;
                }
                maxbits = getchar();        /* set -b from file */
                block_compress = maxbits & BLOCK_MASK;
                maxbits &= BIT_MASK;
                maxmaxcode = 1 << maxbits;
                if(maxbits > BITS) {
                    fprintf(stderr,
                    "%s: compressed with %d bits, can only handle %d bits\n"
                    *fileptr, maxbits, BITS);
                    continue;
                }
            }
            /* Generate output filename */
            strcpy(ofname, *fileptr);
            ofname[strlen(*fileptr) - 2] = '\0';  /* Strip off .Z */
```

```
        } else {                                    /* COMPRESSION */
            if (strcmp(*fileptr + strlen(*fileptr) - 2, ".Z") == 0) {
                    fprintf(stderr, "%s: already has .Z suffix -- no change\
                        *fileptr);
                continue;
            }
            /* Open input file */
            if ((freopen(*fileptr, "r", stdin)) == NULL) {
                perror(*fileptr); continue;
            }
            stat ( *fileptr, &statbuf );
            fsize = (long) statbuf.st_size;
            /*
             * tune hash table size for small files -- ad hoc,
             * but the sizes match earlier #defines, which
             * serve as upper bounds on the number of output codes.
             */
            hsize = HSIZE;
            if ( fsize < (1 << 12) )
                hsize = min ( 5003, HSIZE );
            else if ( fsize < (1 << 13) )
                hsize = min ( 9001, HSIZE );
            else if ( fsize < (1 << 14) )
                hsize = min ( 18013, HSIZE );
            else if ( fsize < (1 << 15) )
                hsize = min ( 35023, HSIZE );
            else if ( fsize < 47000 )
                hsize = min ( 50021, HSIZE );

            /* Generate output filename */
            strcpy(ofname, *fileptr);
#ifndef BSD4_2              /* Short filenames */
            if ((cp=rindex(ofname,'/')) != NULL)    cp++;
            else                                    cp = ofname;
            if (strlen(cp) > 12) {
                fprintf(stderr,"%s: filename too long to tack on .Z\n",cp);
                continue;
            }
#endif  /* BSD4_2           Long filenames allowed */
            strcat(ofname, ".Z");
        }
        /* Check for overwrite of existing file */
        if (overwrite == 0 && zcat_flg == 0) {
            if (stat(ofname, &statbuf) == 0) {
                char response[2];
                response[0] = 'n';
                fprintf(stderr, "%s already exists;", ofname);
                if (foreground()) {
                    fprintf(stderr, " do you wish to overwrite %s (y or n)?
                    ofname);
                    fflush(stderr);
                    read(2, response, 2);
                    while (response[1] != '\n') {
                        if (read(2, response+1, 1) < 0) {   /* Ack! */
                            perror("stderr"); break;
                        }
                    }
                }
                if (response[0] != 'y') {
                    fprintf(stderr, "\tnot overwritten\n");
                    continue;
                }
            }
        }
```

```
            if(zcat_flg == 0) {          /* Open output file */
                if (freopen(ofname, "w", stdout) == NULL) {
                    perror(ofname);
                    continue;
                }
                if(!quiet)
                        fprintf(stderr, "%s: ", *fileptr);
            }

            /* Actually do the compression/decompression */
            if (do_decomp == 0) compress();
#ifndef DEBUG
            else                        decompress();
#else
            else if (debug == 0)        decompress();
            else                        printcodes();
            if (verbose)                dump_tab();
#endif /* DEBUG */
            if(zcat_flg == 0) {
                copystat(*fileptr, ofname);     /* Copy stats */
                if((exit_stat == 1) || (!quiet))
                        putc('\n', stderr);
            }
        }
    } else {            /* Standard input */
        if (do_decomp == 0) {
                compress();
#ifdef DEBUG
                if(verbose)             dump_tab();
#endif /* DEBUG */
                if(!quiet)
                        putc('\n', stderr);
        } else {
            /* Check the magic number */
            if (nomagic == 0) {
                if ((getchar()!=(magic_header[0] & 0xFF))
                 || (getchar()!=(magic_header[1] & 0xFF))) {
                    fprintf(stderr, "stdin: not in compressed format\n");
#ifdef vms
                    exit(0);
#else
                    exit(1);
#endif
                }
                maxbits = getchar();    /* set -b from file */
                block_compress = maxbits & BLOCK_MASK;
                maxbits &= BIT_MASK;
                maxmaxcode = 1 << maxbits;
                fsize = 100000;         /* assume stdin large for USERMEM */
                if(maxbits > BITS) {
                        fprintf(stderr,
                        "stdin: compressed with %d bits, can only handle %d bits
                        maxbits, BITS);
#ifdef vms
                        exit(0);
#else
                        exit(1);
#endif
                }
            }
#ifndef DEBUG
            decompress();
#else
            if (debug == 0)     decompress();
```

```
            else                printcodes();
            if (verbose)        dump_tab();
#endif /* DEBUG */
        }
    }
#ifdef vms
    exit((exit_stat==0) ? 1 : 0);       /* VMS uses 1 for success! */
#else
    exit(exit_stat);
#endif /* vms */
}

static int offset;
long int in_count = 1;                  /* length of input */
long int bytes_out;                     /* length of compressed output */
long int out_count = 0;                 /* # of codes output (for debugging) */

/*
 * compress stdin to stdout
 *
 * Algorithm:  use open addressing double hashing (no chaining) on the
 * prefix code / next character combination.  We do a variant of Knuth's
 * algorithm D (vol. 3, sec. 6.4) along with G. Knott's relatively prime
 * secondary probe.  Here, the modular division first probe is gives way
 * to a faster exclusive-or manipulation.  Also do block compression with
 * an adaptive reset, whereby the code table is cleared when the compression
 * ratio decreases, but after the table fills.  The variable-length output
 * codes are re-sized at this point, and a special CLEAR code is generated
 * for the decompressor.  Late addition:  construct the table according to
 * file size for noticeable speed improvement on small files.  Please direct
 * questions about this implementation to ames!jaw.
 */

compress() {
    register long fcode;
    register code_int i = 0;
    register int c;
    register code_int ent;
#ifdef XENIX_16
    register code_int disp;
#else   /* Normal machine */
    register int disp;
#endif
    register code_int hsize_reg;
    register int hshift;

#ifndef COMPATIBLE
    if (nomagic == 0) {
        putchar(magic_header[0]); putchar(magic_header[1]);
        putchar((char)(maxbits | block_compress));
        if(ferror(stdout))
                writeerr();
    }
#endif /* COMPATIBLE */

    offset = 0;
    bytes_out = 3;              /* includes 3-byte header mojo */
    out_count = 0;
    clear_flg = 0;
    ratio = 0;
    in_count = 1;
    checkpoint = CHECK_GAP;
    maxcode = MAXCODE(n_bits = INIT_BITS);
    free_ent = ((block_compress) ? FIRST : 256 );
    ent = getchar ();

    hshift = 0;
    for ( fcode = (long) hsize;  fcode < 65536L; fcode *= 2L )
        hshift++;
    hshift = 8 - hshift;              /* set hash code range bound */

    hsize_reg = hsize;
    cl_hash( (count_int) hsize_reg);          /* clear hash table */

#ifdef SIGNED_COMPARE_SLOW
    while ( (c = getchar()) != (unsigned) EOF ) {
#else
    while ( (c = getchar()) != EOF ) {
#endif
        in_count++;
        fcode = (long) (((long) c << maxbits) + ent);
        i = ((c << hshift) ^ ent);      /* xor hashing */

        if ( htabof (i) == fcode ) {
            ent = codetabof (i);
            continue;
        } else if ( (long)htabof (i) < 0 )      /* empty slot */
            goto nomatch;
        disp = hsize_reg - i;           /* secondary hash (after G. Knott) */
        if ( i == 0 )
            disp = 1;
probe:
        if ( (i -= disp) < 0 )
            i += hsize_reg;

        if ( htabof (i) == fcode ) {
            ent = codetabof (i);
            continue;
        }
        if ( (long)htabof (i) > 0 )
            goto probe;
nomatch:
        output ( (code_int) ent );
        out_count++;
        ent = c;
#ifdef SIGNED_COMPARE_SLOW
        if ( (unsigned) free_ent < (unsigned) maxmaxcode) {
#else
        if ( free_ent < maxmaxcode ) {
#endif
            codetabof (i) = free_ent++; /* code -> hashtable */
            htabof (i) = fcode;
        }
        else if ( (count_int)in_count >= checkpoint && block_compress )
            cl_block ();
    }
    /*
     * Put out the final code.
     */
    output( (code_int)ent );
    out_count++;
    output( (code_int) -1 );

    /*
     * Print out stats on stderr
     */
    if(zcat_flg == 0 && !quiet) {
```

```
#ifdef DEBUG
        fprintf( stderr,
                "%ld chars in, %ld codes (%ld bytes) out, compression factor: ",
                in_count, out_count, bytes_out );
        prratio( stderr, in_count, bytes_out );
        fprintf( stderr, "\n");
        fprintf( stderr, "\tCompression as in compact: " );
        prratio( stderr, in_count-bytes_out, in_count );
        fprintf( stderr, "\n");
        fprintf( stderr, "\tLargest code (of last block) was %d (%d bits)\n",
                free_ent - 1, n_bits );
#else /* !DEBUG */
        fprintf( stderr, "Compression: " );
        prratio( stderr, in_count-bytes_out, in_count );
#endif /* DEBUG */
    }
    if(bytes_out > in_count)      /* exit(2) if no savings */
        exit_stat = 2;
    return;
}

/*****************************************************************
 * TAG( output )
 *
 * Output the given code.
 * Inputs:
 *      code:   A n_bits-bit integer.  If == -1, then EOF.  This assumes
 *              that n_bits =< (long)wordsize - 1.
 * Outputs:
 *      Outputs code to the file.
 * Assumptions:
 *      Chars are 8 bits long.
 * Algorithm:
 *      Maintain a BITS character long buffer (so that 8 codes will
 * fit in it exactly).  Use the VAX insv instruction to insert each
 * code in turn.  When the buffer fills up empty it and start over.
 */

static char buf[BITS];

#ifndef vax
char_type lmask[9] = {0xff, 0xfe, 0xfc, 0xf8, 0xf0, 0xe0, 0xc0, 0x80, 0x00};
char_type rmask[9] = {0x00, 0x01, 0x03, 0x07, 0x0f, 0x1f, 0x3f, 0x7f, 0xff};
#endif /* vax */

output( code )
code_int  code;
{
#ifdef DEBUG
    static int col = 0;
#endif /* DEBUG */

    /*
     * On the VAX, it is important to have the register declarations
     * in exactly the order given, or the asm will break.
     */
    register int r_off = offset, bits= n_bits;
    register char * bp = buf;

#ifdef DEBUG
        if ( verbose )
            fprintf( stderr, "%5d%c", code,
                    (col+=6) >= 74 ? (col = 0, '\n') : ' ' );
#endif /* DEBUG */
```

```
    if ( code >= 0 ) {
#ifdef vax
        /* VAX DEPENDENT!! Implementation on other machines is below
         *
         * Translation: Insert BITS bits from the argument starting at
         * offset bits from the beginning of buf.
         */

#ifdef vms
        LIB$insv(&code,&r_off,&bits,bp);
#else
        0;      /* Work around for pcc -O bug with asm and if stmt */
        asm( "insv      4(ap),r11,r10,(r9)" );
#endif /* vms */

#else /* not a vax */
/*
 * byte/bit numbering on the VAX is simulated by the following code
 */
        /*
         * Get to the first byte.
         */
        bp += (r_off >> 3);
        r_off &= 7;
        /*
         * Since code is always >= 8 bits, only need to mask the first
         * hunk on the left.
         */
        *bp = (*bp & rmask[r_off]) | (code << r_off) & lmask[r_off];
        bp++;
        bits -= (8 - r_off);
        code >>= 8 - r_off;
        /* Get any 8 bit parts in the middle (<=1 for up to 16 bits). */
        if ( bits >= 8 ) {
            *bp++ = code;
            code >>= 8;
            bits -= 8;
        }
        /* Last bits. */
        if(bits)
            *bp = code;
#endif /* vax */
        offset += n_bits;
        if ( offset == (n_bits << 3) ) {
            bp = buf;
            bits = n_bits;
            bytes_out += bits;
            do
                putchar(*bp++);
            while(--bits);
            offset = 0;
        }

        /*
         * If the next entry is going to be too big for the code size,
         * then increase it, if possible.
         */
        if ( free_ent > maxcode || (clear_flg > 0))
        {
            /*
             * Write the whole buffer, because the input side won't
             * discover the size increase until after it has read it.
             */
            if ( offset > 0 ) {
```

```
                if( fwrite( buf, 1, n_bits, stdout ) != n_bits)
                        writeerr();
                bytes_out += n_bits;
            }
            offset = 0;

            if ( clear_flg ) {
                maxcode = MAXCODE (n_bits = INIT_BITS);
                clear_flg = 0;
            }
            else {
                n_bits++;
                if ( n_bits == maxbits )
                    maxcode = maxmaxcode;
                else
                    maxcode = MAXCODE(n_bits);
            }
#ifdef DEBUG
            if ( debug ) {
                fprintf( stderr, "\nChange to %d bits\n", n_bits );
                col = 0;
            }
#endif /* DEBUG */
        }
    } else {
        /*
         * At EOF, write the rest of the buffer.
         */
        if ( offset > 0 )
            fwrite( buf, 1, (offset + 7) / 8, stdout );
        bytes_out += (offset + 7) / 8;
        offset = 0;
        fflush( stdout );
#ifdef DEBUG
        if ( verbose )
            fprintf( stderr, "\n" );
#endif /* DEBUG */
        if( ferror( stdout ) )
                writeerr();
    }
}

/*
 * Decompress stdin to stdout.  This routine adapts to the codes in the
 * file building the "string" table on the fly; requiring no table to
 * be stored in the compressed file.  The tables used herein are shared
 * with those of the compress() routine.  See the definitions above.
 */

decompress() {
    register char_type *stackp;
    register int finchar;
    register code_int code, oldcode, incode;

    /*
     * As above, initialize the first 256 entries in the table.
     */
    maxcode = MAXCODE(n_bits = INIT_BITS);
    for ( code = 255; code >= 0; code-- ) {
        tab_prefixof(code) = 0;
        tab_suffixof(code) = (char_type)code;
    }
    free_ent = ((block_compress) ? FIRST : 256 );
    finchar = oldcode = getcode();
    if(oldcode == -1)   /* EOF already? */
        return;             /* Get out of here */
    putchar( (char)finchar );           /* first code must be 8 bits = char */
    if(ferror(stdout))          /* Crash if can't write */
        writeerr();
    stackp = de_stack;

    while ( (code = getcode()) > -1 ) {

        if ( (code == CLEAR) && block_compress ) {
            for ( code = 255; code >= 0; code-- )
                tab_prefixof(code) = 0;
            clear_flg = 1;
            free_ent = FIRST - 1;
            if ( (code = getcode ()) == -1 )    /* O, untimely death! */
                break;
        }
        incode = code;
        /*
         * Special case for KwKwK string.
         */
        if ( code >= free_ent ) {
            *stackp++ = finchar;
            code = oldcode;
        }

        /*
         * Generate output characters in reverse order
         */
#ifdef SIGNED_COMPARE_SLOW
        while ( ((unsigned long)code) >= ((unsigned long)256) ) {
#else
        while ( code >= 256 ) {
#endif
            *stackp++ = tab_suffixof(code);
            code = tab_prefixof(code);
        }
        *stackp++ = finchar = tab_suffixof(code);

        /*
         * And put them out in forward order
         */
        do
            putchar ( *--stackp );
        while ( stackp > de_stack );

        /*
         * Generate the new entry.
         */
        if ( (code=free_ent) < maxmaxcode ) {
            tab_prefixof(code) = (unsigned short)oldcode;
            tab_suffixof(code) = finchar;
            free_ent = code+1;
        }
        /*
         * Remember previous code
         */
        oldcode = incode;
    }
    fflush( stdout );
    if(ferror(stdout))
        writeerr();
}
```

```
/*****************************************************************
 * TAG( getcode )
 *
 * Read one code from the standard input.  If EOF, return -1.
 * Inputs:
 *      stdin
 * Outputs:
 *      code or -1 is returned.
 */

code_int
getcode() {
    /*
     * On the VAX, it is important to have the register declarations
     * in exactly the order given, or the asm will break.
     */
    register code_int code;
    static int offset = 0, size = 0;
    static char_type buf[BITS];
    register int r_off, bits;
    register char_type *bp = buf;

#ifdef vms
    int LIB$extzv ();
#endif

    if ( clear_flg > 0 || offset >= size || free_ent > maxcode ) {
        /*
         * If the next entry will be too big for the current code
         * size, then we must increase the size.  This implies reading
         * a new buffer full, too.
         */
        if ( free_ent > maxcode ) {
            n_bits++;
            if ( n_bits == maxbits )
                maxcode = maxmaxcode;   /* won't get any bigger now */
            else
                maxcode = MAXCODE(n_bits);
        }
        if ( clear_flg > 0) {
            maxcode = MAXCODE (n_bits = INIT_BITS);
            clear_flg = 0;
        }
        size = fread( buf, 1, n_bits, stdin );
        if ( size <= 0 )
            return -1;                  /* end of file */
        offset = 0;
        /* Round size down to integral number of codes */
        size = (size << 3) - (n_bits - 1);
    }
    r_off = offset;
    bits = n_bits;
#ifdef vax

#ifdef vms
    code = LIB$extzv(&r_off,&bits,bp);
#else
    asm( "extzv   r10,r9,(r8),r11" );
#endif /* vms */

#else /* not a vax */
        /*
         * Get to the first byte.
         */
        bp += (r_off >> 3);
        r_off &= 7;
        /* Get first part (low order bits) */
#ifdef NO_UCHAR
        code = ((*bp++ >> r_off) & rmask[8 - r_off]) & 0xff;
#else
        code = (*bp++ >> r_off);
#endif /* NO_UCHAR */
        bits -= (8 - r_off);
        r_off = 8 - r_off;              /* now, offset into code word */
        /* Get any 8 bit parts in the middle (<=1 for up to 16 bits)  */
        if ( bits >= 8 ) {
#ifdef NO_UCHAR
            code |= (*bp++ & 0xff) << r_off;
#else
            code |= *bp++ << r_off;
#endif /* NO_UCHAR */
            r_off += 8;
            bits -= 8;
        }
        /* high order bits. */
        code |= (*bp & rmask[bits]) << r_off;
#endif /* vax */
    offset += n_bits;

    return code;
}

char *
rindex(s, c)            /* For those who don't have it in libc.a */
register char *s, c;
{
        char *p;
        for (p = NULL; *s; s++)
            if (*s == c)
                p = s;
        return(p);
}

#ifdef DEBUG
printcodes()
{
    /*
     * Just print out codes from input file.  For debugging.
     */
    code_int code;
    int col = 0, bits;

    bits = n_bits = INIT_BITS;
    maxcode = MAXCODE(n_bits);
    free_ent = ((block_compress) ? FIRST : 256 );
    while ( ( code = getcode() ) >= 0 ) {
        if ( (code == CLEAR) && block_compress ) {
            free_ent = FIRST - 1;
            clear_flg = 1;
        }
        else if ( free_ent < maxmaxcode )
            free_ent++;
        if ( bits != n_bits ) {
            fprintf(stderr, "\nChange to %d bits\n", n_bits );
            bits = n_bits;
            col = 0;
        }
```

```
        fprintf(stderr, "%5d%c", code, (col+=6) >= 74 ? (col = 0, '\n') : ' ' );
    }
    putc( '\n', stderr );

#ifdef vms
    exit(1);
#else
    exit( 0 );
#endif
}

code_int sorttab[1<<BITS];      /* sorted pointers into htab */

dump_tab()      /* dump string table */
{
    register int i, first;
    register ent;
#define STACK_SIZE      15000
    int stack_top = STACK_SIZE;
    register c;

    if(do_decomp == 0) {        /* compressing */
        register int flag = 1;

        for(i=0; i<hsize; i++) {        /* build sort pointers */
                if((long)htabof(i) >= 0) {
                        sorttab[codetabof(i)] = i;
                }
        }
        first = block_compress ? FIRST : 256;
        for(i = first; i < free_ent; i++) {
                fprintf(stderr, "%5d: \"", i);
                de_stack[--stack_top] = '\n';
                de_stack[--stack_top] = '"';
                stack_top = in_stack((htabof(sorttab[i])>>maxbits)&0xff,
                                     stack_top);
                for(ent=htabof(sorttab[i]) & ((1<<maxbits) 1);
                    ent > 256;
                    ent=htabof(sorttab[ent]) & ((1<<maxbits)-1)) {
                        stack_top = in_stack(htabof(sorttab[ent]) >> maxbits,
                                                stack_top);
                }
                stack_top = in_stack(ent, stack_top);
                fwrite( &de_stack[stack_top], 1, STACK_SIZE stack_top, stderr);
                stack_top = STACK_SIZE;
        }
   } else if(!debug) {  /* decompressing */

       for ( i = 0; i < free_ent; i++ ) {
           ent = i;
           c = tab_suffixof(ent);
           if ( isascii(c) && isprint(c) )
               fprintf( stderr, "%5d: %5d/'%c'  \"",
                           ent, tab_prefixof(ent), c );
           else
               fprintf( stderr, "%5d: %5d/\\%03o \"",
                           ent, tab_prefixof(ent), c );
           de_stack[--stack_top] = '\n';
           de_stack[--stack_top] = '"';
           for ( ; ent != NULL;
                   ent = (ent >= FIRST ? tab_prefixof(ent) : NULL) ) {
               stack_top = in_stack(tab_suffixof(ent), stack_top);
           }
           fwrite( &de_stack[stack_top], 1, STACK_SIZE - stack_top, stderr );
```

```
            stack_top = STACK_SIZE;
        }
    }
}

int
in_stack(c, stack_top)
        register c, stack_top;
{
        if ( (isascii(c) && isprint(c) && c != '\\') || c == ' ' ) {
            de_stack[--stack_top] = c;
        } else {
            switch( c ) {
            case '\n': de_stack[--stack_top] = 'n'; break;
            case '\t': de_stack[--stack_top] = 't'; break;
            case '\b': de_stack[--stack_top] = 'b'; break;
            case '\f': de_stack[--stack_top] = 'f'; break;
            case '\r': de_stack[--stack_top] = 'r'; break;
            case '\\': de_stack[--stack_top] = '\\'; break;
            default:
                de_stack[--stack_top] = '0' + c % 8;
                de_stack[--stack_top] = '0' + (c / 8) % 8;
                de_stack[--stack_top] = '0' + c / 64;
                break;
            }
            de_stack[--stack_top] = '\\';
        }
        return stack_top;
}
#endif /* DEBUG */

writeerr()
{
    perror ( ofname );
#ifdef vms
    exit (0);
#else
    unlink ( ofname );
    exit ( 1 );
#endif /* vms */
}

copystat(ifname, ofname)
char *ifname, *ofname;
{
    struct stat statbuf;
    int mode;
    time_t timep[2];

    fclose(stdout);
    if (stat(ifname, &statbuf)) {                /* Get stat on input file */
        perror(ifname);
        return;
    }
    /* this is org line, changed to allow other file types, is this wise??? */
    if ((statbuf.st_mode & S_IFMT) != S_IFREG) {
        if(quiet)
                fprintf(stderr, "%s: ", ifname);
        fprintf(stderr, " -- not a regular file: unchanged");
        exit_stat = 1;
    } else if (statbuf.st_nlink > 1) {
        if(quiet)
                fprintf(stderr, "%s: ", ifname);
        fprintf(stderr, " -- has %d other links: unchanged",
```

```
                statbuf.st_nlink - 1);
        exit_stat = 1;
    } else if (exit_stat == 2 && (!force)) { /* No compression: remove file.Z */
        if(!quiet)
                fprintf(stderr, " -- file unchanged");
    } else {                        /* ***** Successful Compression ***** */
        exit_stat = 0;
        mode = statbuf.st_mode & 07777;
        if (chmod(ofname, mode))                /* Copy modes */
            perror(ofname);
        chown(ofname, statbuf.st_uid, statbuf.st_gid);  /* Copy ownership */
        timep[0] = statbuf.st_atime;
        timep[1] = statbuf.st_mtime;
#ifndef vms
        utime(ofname, timep);   /* Update last accessed and modified times */
        if (unlink(ifname))     /* Remove input file */
            perror(ifname);
#endif
        if(!quiet)
                fprintf(stderr, " -- replaced with %s", ofname);
        return;         /* Successful return */
    }

    /* Unsuccessful return -- one of the tests failed */
#ifndef vms
    if (unlink(ofname))
        perror(ofname);
#endif /* vms */
}
/*
 * This routine returns 1 if we are running in the foreground and stderr
 * is a tty.
 */
foreground()
{
        if(bgnd_flag) { /* background? */
                return(0);
        } else {                        /* foreground */
                if(isatty(2)) {         /* and stderr is a tty */
                        return(1);
                } else {
                        return(0);
                }
        }
}

onintr ( )
{
#ifdef vms
    exit ( 0 );
#else
    unlink ( ofname );
    exit ( 1 );
#endif  /* vms */
}

oops ( )        /* wild pointer -- assume bad input */
{
    if ( do_decomp == 1 )
        fprintf ( stderr, "uncompress: corrupt input\n" );
#ifdef vms
    exit ( 0 );
#else
    unlink ( ofname );
    exit ( 1 );
#endif  /* vms */
}

cl_block ()             /* table clear for block compress */
{
    register long int rat;

    checkpoint = in_count + CHECK_GAP;
#ifdef DEBUG
        if ( debug ) {
                fprintf ( stderr, "count: %ld, ratio: ", in_count );
                prratio ( stderr, in_count, bytes_out );
                fprintf ( stderr, "\n");
        }
#endif /* DEBUG */

    if(in_count > 0x007fffff) { /* shift will overflow */
        rat = bytes_out >> 8;
        if(rat == 0) {          /* Don't divide by zero */
            rat = 0x7fffffff;
        } else {
            rat = in_count / rat;
        }
    } else {
        rat = (in_count << 8) / bytes_out;      /* 8 fractional bits */
    }
    if ( rat > ratio ) {
        ratio = rat;
    } else {
        ratio = 0;
#ifdef DEBUG
        if(verbose)
                dump_tab();     /* dump string table */
#endif
        cl_hash ( (count_int) hsize );
        free_ent = FIRST;
        clear_flg = 1;
        output ( (code_int) CLEAR );
#ifdef DEBUG
        if(debug)
                fprintf ( stderr, "clear\n" );
#endif /* DEBUG */
    }
}

cl_hash(hsize)          /* reset code table */
        register count_int hsize;
{
#ifndef XENIX_16        /* Normal machine */
        register count_int *htab_p = htab+hsize;
#else
        register j;
        register long k = hsize;
        register count_int *htab_p;
#endif
        register long i;
        register long m1 = -1;

#ifdef XENIX_16
    for(j=0; j<=8 && k>=0; j++,k -= 8192) {
        i = 8192;
        if(k < 8192) {
                i = k;
```

```
        }
        htab_p = &(htab[j][i]);
        i -= 16;
        if(i > 0) {
#else
        i = hsize - 16;
#endif
        do {                            /* might use Sys V memset(3) here */
                *(htab_p-16) = m1;
                *(htab_p-15) = m1;
                *(htab_p-14) = m1;
                *(htab_p-13) = m1;
                *(htab_p-12) = m1;
                *(htab_p-11) = m1;
                *(htab_p-10) = m1;
                *(htab_p-9) = m1;
                *(htab_p-8) = m1;
                *(htab_p-7) = m1;
                *(htab_p-6) = m1;
                *(htab_p-5) = m1;
                *(htab_p-4) = m1;
                *(htab_p-3) = m1;
                *(htab_p-2) = m1;
                *(htab_p-1) = m1;
                htab_p -= 16;
        } while ((i -= 16) >= 0);
#ifdef XENIX_16
        }
    }
#endif
        for ( i += 16; i > 0; i-- )
                *--htab_p = m1;
}

prratio(stream, num, den)
FILE *stream;
long int num, den;
{
        register int q;                 /* Doesn't need to be long */

        if(num > 214748L) {             /* 2147483647/10000 */
                q = num / (den / 10000L);
        } else {
                q = 10000L * num / den;         /* Long calculations, though */
        }
        if (q < 0) {
                putc('-', stream);
                q = -q;
        }
        fprintf(stream, "%d.%02d%%", q / 100, q % 100);
}

version()
{
        fprintf(stderr, "%s\n", rcs_ident);
        fprintf(stderr, "Options: ");
#ifdef vax
        fprintf(stderr, "vax, ");
#endif
#ifdef NO_UCHAR
        fprintf(stderr, "NO_UCHAR, ");
#endif
#ifdef SIGNED_COMPARE_SLOW
        fprintf(stderr, "SIGNED_COMPARE_SLOW, ");
```

```
#endif
#ifdef XENIX_16
        fprintf(stderr, "XENIX_16, ");
#endif
#ifdef COMPATIBLE
        fprintf(stderr, "COMPATIBLE, ");
#endif
#ifdef DEBUG
        fprintf(stderr, "DEBUG, ");
#endif
#ifdef BSD4_2
        fprintf(stderr, "BSD4_2, ");
#endif
        fprintf(stderr, "BITS = %d\n", BITS);
}
```

```
HSC000$DUA1:[OCR.PGM]FIXQ.C;16              6 APR 1987 15:08              Page 1

/************************************************************************

Copyright 1986 Environmental Research Institute of Michigan
               Ann Arbor, Michigan


       fixq.c                                             ERIM ARV
       By : Francis Quek        Date : 5 April 1987

       Modified from FIXED.C by Steve Smith

       Last Modified by :
       Date :

Description :
This program will convert stream files to fixed length record files.
The record length will be number_of_columns.
If the original image is larger than 0.5 MByte in size, it is downsampled
by a factor of four (2*2). The downsampling is performed using simple
arithmetic averaging.
The resultant image is trimmed to yield an even dimensioned image (as
required by C4PL).

Input :
This programme takes as input 4 parameters :
       InputFileName, OutputFileName, Num_of_Rows, Num_of_Columns

Output :
A fixed-length record file of the specified name as described in the
programme description.

$ FIXQ oldfile newfile number_of_rows number_of_columns

N.B. There is another programme named FIXQQ.C which does the same thing,
       but, which reads OutputFileName, Num of Rows, Num of Columns
       from a header file (as provided by SUNY).

*************************************************************************/

#include        stdio
#include        file

#define         USAGE   "FIXQ SourceImage TargetImage SrcRows SrcCols.\n"

/* Obsolete: For use with old Ektron images of constant dimensions 2048 X 2048.
#define         FRDIM   2048
#define         BLKSIZ  512
#define         MAXREC  2048
*/
extern          BLKSIZ;

int     main (argc, argv)
int     argc ;                          /* Number of args in call */
char    *argv[] ;                       /* Character strings in call */
{

        FILE    *fopen(),
                *fp;

        int     row,
                column,
                bufsiz ,                        /* Size of buffer allocated for img */
                numrec ,                        /* Records in file (lines/img) */
                recsiz ,                        /* Size of record (pixels/line) */
```

```
HSC000$DUA1:[OCR.PGM]FIXQ.C;16              6 APR 1987 15:08              Page 2
                fd ,                        /* File descriptor for both I and O */
                siz ,
                TgtRow , TgtCol ,           /* Row & Col. pointers of target img */
                TgtRowSiz , TgtColSiz ,     /* Target img row & col. sizes */
                ColPad ;                    /* Spare bytes after making the
                                               dimensions of the image even */

static unsigned char image[2097152] ,       /* [4194304]; */

        if ( argc != 5 ) {
          fprintf ( stderr, USAGE ) ;
          exit(1) ;
        }

        /* Convert input image size to int */
        sscanf (argv[3], "%d", &row);
        sscanf (argv[4], "%d", &column);

        BLKSIZ = 512;
        recsiz = column ;
        bufsiz = row*column;

/* Read in image */
        fp = fopen ( argv[1], "r") ;
        siz = ( fread (image, 1, bufsiz, fp ));
        printf ( "%d bytes read\n", siz ) ;
        fclose (fp) ;

/* If the image is too large downsample it by 4 */
        if ( bufsiz >= 512 * 1024 ) {
          TgtRowSiz = (row / 4) * 2 ;           /* C4PL needs even dimension */
          TgtColSiz = (column / 4) * 2 ;        /* images */
          printf ("row = %5d, column = %5d, TgtRowSiz = %5d, TgtColSiz = %5d\n",
                  row, column, TgtRowSiz, TgtColSiz) ;
          ColPad = column - 2*TgtColSiz ;
          downsamp (image, row, column, TgtRowSiz, TgtColSiz, ColPad) ;
          row = TgtRowSiz ;
          column = TgtColSiz ;
          recsiz = TgtColSiz ;
          bufsiz = TgtRowSiz * TgtColSiz ;
        }
        else {          /* The image is not too large, we want to make the
                           dimensions even */
          if ( column != (column/2)*2 ) { /* The number of columns is odd */
            MakeEven (image, row, column, 1) ;
            column -= 1 ;
            recsiz = column ;
            row = (row/2) * 2 ;
            bufsiz = row * column ;
          }
          else {
            row = (row/2) * 2 ; /* Make sure the number of rows is even */
            bufsiz = row * column ,
          }
        }

/* Create fixed length record file (CREATE returns the fd) */
        if ( (fd = creat( (argv[2], recsiz, bufsiz) ) == -1 ) {
          fprintf ( stderr  "fixed, create failed on %s\n", argv[2] ),
          exit(1) ,
        }

/* Write fixed length record file */
```

```
        lseek (fd, 0, 0) ;
        if ( writef ( fd, image, recsiz, bufsiz) == -1 ) {
          fprintf ( stderr, "fixed, write failed on %s\n", argv[2] ) ;
          close (fd) ;
          exit(1) ;
        }
        close (fd) ;
}

/*--------------------------------------------------------------------
 *      downsamp        Downsamples an image by 4
 *--------------------------------------------------------------------
 */
int     downsamp (image, SrcRowSiz, SrcColSiz, TgtRowSiz, TgtColSiz, ColPad)
unsigned char   *image ;        /* the image */
        int     SrcRowSiz ;     /* Source image dimensions */
        int     SrcColSiz ;
        int     TgtRowSiz ;     /* Target image dimensions */
        int     TgtColSiz ;
        int     ColPad ;        /* Spare bytes after making the dimensions
                                   of the image even */
{
unsigned char   *SrcRow1, *SrcRow2 ;    /* Rows of the Source image */
unsigned char   *TgtRow ;
        int     sum ,           /* To take the average */
                row , col ;     /* To keep track of rows and cols */

        SrcRow1 = image ;
        SrcRow2 = image + SrcColSiz ;
        TgtRow = image ;
        for ( row=0 ; row < TgtRowSiz ; ++row ) {
          for (col=0 ; col < TgtColSiz ; ++col ) {
            sum = ((int) *SrcRow1++ + *SrcRow1++ + *SrcRow2++ + *SrcRow2++) ;
            *TgtRow++ = sum / 4 ;
          }
          SrcRow1 += ( SrcColSiz + ColPad ) ;
          SrcRow2 += ( SrcColSiz + ColPad ) ;
        }
}

/*--------------------------------------------------------------------
 *      MakeEven        Makes the image dimensions even
 *--------------------------------------------------------------------
 */
int     MakeEven (image, SrcRowSiz, SrcColSiz, ColPad)
unsigned char   *image ;        /* the image */
        int     SrcRowSiz ;     /* Source image dimensions */
        int     SrcColSiz ;
        int     ColPad ;        /* Spare bytes after making the dimensions
                                   of the image even */
{
unsigned char   *SrcRow ;       /* Rows of the Source image */
unsigned char   *TgtRow ;
        int     row , col ,     /* To keep track of rows and cols */
                TgtColSiz ,     /* The eventual dimension of the image */
                TgtRowSiz ;

        SrcRow = image ;
        TgtRow = image ;
        TgtRowSiz = (SrcRowSiz/2) * 2 ;
        TgtColSiz = SrcColSiz - ColPad ;
        for ( row=0 ; row < TgtRowSiz ; ++row ) {
          for (col=0 ; col < TgtColSiz ; ++col )
            *TgtRow++ = *SrcRow++ ;
```

```
          SrcRow += ColPad ;
        }
}

/*--------------------------------------------------------------------
 *      creatf          Creates file with fixed records
 *--------------------------------------------------------------------
 */
int     creatf (fname,recsiz,fsize)
                                        /* RC - file descriptor from creat call */
unsigned char   *fname ;                /* I - name of file to be created */
        int     recsiz ,                /* I - number of bytes per record */
                fsize ;                 /* I - number of bytes in file */
{
        int     fd ;                    /* File descriptor to image, returned */
unsigned char   mrs[20],                /* Maximum record size, call to creat */
                alq[20] ;               /* Allocation quantity, call to creat */

        sprintf ( mrs, "mrs=%d", recsiz ) ;
        sprintf ( alq, "alq=%d", (fsize+BLKSIZ-1)/BLKSIZ ) ;
        fd = creat ( fname, 0, "fop=cbt,tef", "rfm=fix", mrs, alq ) ;
        return(fd) ;
}
/*--------------------------------------------------------------------
 *      writef          Outputs buffer to file, fixed length records
 *--------------------------------------------------------------------
 */
int     writef ( fd, p_buf, recsiz, bytes )
                                        /* RC - TRUE, successful write */
                                        /*      FALSE, failed to write buffer */
        int     fd ;                    /* I - file descriptor */
unsigned char   *p_buf ;                /* I - pointer to buffer to be output */
        int     recsiz ;                /* I - size of records to use in write */
        int     bytes ;                 /* I - number of bytes to write out */
{
register int    rc ,                    /* Return code from write call, #bytes output */
                rsiz ,                  /* Register copy of recsiz */
                b_rem ;                 /* Bytes remaining to be written from buffer */
register unsigned char  *pb ,           /* Register pointer to buffer */
                *pb_org,*pb_rec;/* Used to copy last record if b_rem < recsiz */

        pb = p_buf ;                    /* Get ptr to buffer in register variable */
        b_rem = bytes ;                 /* Bytes remaining to be written, initialize */
        rsiz = recsiz ;                 /* Register copy of record size */

        while (b_rem >= rsiz) {
          rc = write (fd, pb, rsiz) ;
          if ( rc < 0 ) {
            fprintf(stderr,"writef, ERROR on write, %d bytes left\n", b_rem) ;
            return (FALSE) ;
          }
          b_rem -= rc ; pb += rc ;
        }
        if ( b_rem ) {                  /* Have to pad last record (odd size) */
          p_buf = NULL ;
          if ( !getmem(&p_buf,rsiz) ) {
            fprintf ( stderr,
                "writef, calloc error, %d bytes not output\n", b_rem ) ;
            return (FALSE) ;
          }
          pb_rec = p_buf ;              /* Pointer to new buffer space from getmem */
          pb_org = pb ;                 /* Pointer to start of remaining bytes */
          while ( b_rem ) {
            b_rem-- ; *pb_rec++ = *pb_org++ ;
        }
        rc = write (fd, p.buf, rsiz) ;
        cfree (p.buf) ;
        if ( rc < 0 ) {
          fprintf ( stderr,
              "writef, ERROR on last record, %d bytes left\n", b.rem ) ;
          return (FALSE) ;
        }
      }
      return (TRUE) ;
}
```

```
HSC000$DUA1:[OCR.PGM]FIXQQ.C;10          6 APR 1987 15:10          Page 1
/****************************************************************************

Copyright 1986 Environmental Research Institute of Michigan
               Ann Arbor, Michigan


       fixqq.c                                            ERIM ARV
       By : Francis Quek        Date : 5 April 1987

       Modified from FIXED.C by Steve Smith

       Last Modified by :
       Date :

Description :
This program will convert stream files to fixed length record files.
The record length will be number_of_columns.
If the original image is larger than 0.5 MByte in size, it is downsampled
by a factor of four (2*2). The downsampling is performed using simple
arithmetic averaging.
The resultant image is trimmed to yield an even dimensioned image (as
required by C4PL).

Input :
The programme takes as input, two file names. The first is the name of
the stream image file, and the second is the name of a header file of
the following format:
       [imagename]
       Number of Rows: ####
       Number of Columns: ####
  Where [imagename] is the name of the image without extension (nor '.'
                    following the name). The resultant image will bear
                    the name [imagename].IMG
This is the format in the header file provided by SUNY

$ FIXQQ oldfile newfile number_of_rows number_of_columns

N.B. There is another version of this programme which takes the output file
     name, the number of rows and the number of columns as explicit arguments.
     That programme is FIXQ.C
****************************************************************************/

#include        stdio
#include        file

#define         USAGE   "FIXQ SourceImage Headerfile\n"

/* Obsolete: For use with old Eltron images of constant dimensions 2048 X 2048.
#define         FRDIM   2048
#define         BLKSIZ  512
#define         MAXREC  2048
*/
extern          BLKSIZ;

int     main (argc, argv)
int     argc ;                       /* Number of args in call */
char    *argv[] ;                    /* Character strings in call */
{

        FILE    *fopen(),
                *fp,
                *hp ;                /* header file pointer */

        int     row,
```

```
HSC000$DUA1:[OCR.PGM]FIXQQ.C;10          6 APR 1987 15:10          Page 2
                column,
                bufsiz ,                    /* Size of buffer allocated for img */
                numrec ,                    /* #records in file (lines/img) */
                recsiz ,                    /* Size of record (pixels/line) */
                fd ,                        /* File descriptor for both I and O */
                siz ,
                TgtRow , TgtCol ,           /* Row & Col. pointers of target img */
                TgtRowSiz , TgtColSiz ,     /* Target img row & col. sizes */
                ColPad ;                    /* Spare bytes after making the
                                               dimensions of the image even */
        char    inline[255] ,
                outfilename[20] ;


static unsigned char image[2097152] ;   /* [4194304]; */

        if ( argc != 3 ) {
          fprintf ( stderr, USAGE ) ;
          exit(1) ;
        }

        /* The header file */
        hp = fopen(argv[2], "r") ;
        fgets (inline, 255, hp) ;
        sscanf (inline, "%s", outfilename) ;
        strcat(outfilename, ".img") ;           /* Name for the output image */
        fgets (inline, 255, hp) ;
        /* Obtain the image dimensions */
        sscanf (inline, "%*s %*s %*s %d", &row) ;
        fgets (inline, 255, hp) ;
        sscanf (inline, "%*s %*s %*s %d", &column) ;
        printf ("Output file: %s ; Rows = %d ; Cols = %d\n",
                  outfilename, row, column) ;
        BLKSIZ = 512;
        recsiz = column ;
        bufsiz = row*column;

/* Read in image */
        fp = fopen ( argv[1], "r") ;
        siz = ( fread (image, 1, bufsiz, fp ));
        printf ( "%d bytes read\n", siz ) ;
        fclose (fp) ;

/* If the image is too large downsample it by 4 */
        if ( bufsiz >= 512 * 1024 ) {
          TgtRowSiz = (row / 4) * 2 ;             /* C4PL needs even dimension */
          TgtColSiz = (column / 4) * 2 ,          /* images */
          printf ("row = %5d, column = %5d, TgtRowSiz = %5d, TgtColSiz = %5d\n",
                    row, column, TgtRowSiz, TgtColSiz) ;
          ColPad = column - 2*TgtColSiz ;
          downsamp (image, row, column, TgtRowSiz, TgtColSiz, ColPad) ;
          row = TgtRowSiz ;
          column = TgtColSiz ,
          recsiz = TgtColSiz ;
          bufsiz = TgtRowSiz * TgtColSiz ;
        }
        else {             /* The image is not too large, we want to make the
                              dimensions even */
          if ( column != (column/2)*2 ) { /* The number of columns is odd */
            MakeEven (image, row, column, 1) ;
            column -= 1 ,
            recsiz = column ,
            row = (row/2) * 2 .
            bufsiz = row * column ,
        }
        else {
          row = (row/2) * 2 ;     /* Make sure the number of rows is even */
          bufsiz = row * column ;
        }
      }

/* Create fixed length record file (CREATE returns the fd) */
        if ( (fd = creatf (outfilename, recsiz, bufsiz) ) == -1 ) {
          fprintf ( stderr, "fixed, create failed on %s\n", outfilename );
          exit(1) ;
        }

/* Write fixed length record file */
        lseek (fd, 0, 0) ;
        if ( writef ( fd, image, recsiz, bufsiz) == -1 ) {
          fprintf ( stderr, "fixed, write failed on %s\n", argv[2] ) ;
          close (fd) ;
          exit(1) ;
        }
        close (fd) ;
}
 /*-----------------------------------------------------------------------
  *     downsamp        Downsamples an image by 4
  *-----------------------------------------------------------------------
  */
 int    downsamp (image, SrcRowSiz, SrcColSiz, TgtRowSiz, TgtColSiz, ColPad)
unsigned char   *image ;          /* the image */
        int     SrcRowSiz ;       /* Source image dimensions */
        int     SrcColSiz ;
        int     TgtRowSiz ;       /* Target image dimensions */
        int     TgtColSiz ;
        int     ColPad ;          /* Spare bytes after making the dimensions
                                     of the image even */
{
unsigned char   *SrcRow1, *SrcRow2 ;    /* Rows of the Source image */
unsigned char   *TgtRow ;
        int     sum ,             /* To take the average */
                row , col ;       /* To keep track of rows and cols */

        SrcRow1 = image ;
        SrcRow2 = image + SrcColSiz ;
        TgtRow = image ;
        for ( row=0 ; row < TgtRowSiz ; ++row ) {
          for (col=0 , col < TgtColSiz ; ++col ) {
            sum = ((int) *SrcRow1++ + *SrcRow1++ + *SrcRow2++ + *SrcRow2++) ;
            *TgtRow++ = sum / 4 ;
          }
            SrcRow1 += ( SrcColSiz + ColPad ) ;
            SrcRow2 += ( SrcColSiz + ColPad ) ;
        }
}
 /*-----------------------------------------------------------------------
  *     MakeEven        Makes the image dimensions even
  *-----------------------------------------------------------------------
  */
 int    MakeEven (image, SrcRowSiz, SrcColSiz, ColPad)
unsigned char   *image ;          /* the image */
        int     SrcRowSiz ;       /* Source image dimensions */
        int     SrcColSiz ,
        int     ColPad ,          /* Spare bytes after making the dimensions
                                     of the image even */
{
```

```
unsigned char   *SrcRow ;         /* Rows of the Source image */
unsigned char   *TgtRow ;
        int     row , col ,       /* To keep track of rows and cols */
                TgtColSiz ,       /* The eventual dimension of the image */
                TgtRowSiz ;

        SrcRow = image ;
        TgtRow = image ;
        TgtRowSiz = (SrcRowSiz/2) * 2 ;
        TgtColSiz = SrcColSiz  ColPad ;
        for ( row=0 ; row < TgtRowSiz ; ++row ) {
          for (col=0 ; col < TgtColSiz , ++col )
            *TgtRow++ = *SrcRow++ ;
          SrcRow += ColPad ;
        }
}
 /*-----------------------------------------------------------------------
  *     creatf          Creates file with fixed records
  *-----------------------------------------------------------------------
  */
 int    creatf (fname,recsiz,fsize)
                                  /* RC   file descriptor from creat call */
unsigned char   *fname ;          /* I  - name of file to be created */
        int     recsiz ,          /* I  - number of bytes per record */
                fsize ;           /* I  - number of bytes in file */
 {
        int     fd ;              /* File descriptor to image, returned */
unsigned char   mrs[20],          /* Maximum record size, call to creat */
                alq[20] ;         /* Allocation quantity, call to creat */

        sprintf ( mrs, "mrs=%d", recsiz ) ;
        sprintf ( alq, "alq=%d", (fsize+BLKSIZ 1)/BLKSIZ ) ;
        fd = creat ( fname, 0, "fop=cbt,tef", "rfm=fix", mrs, alq ) ;
        return(fd) ;
}
 /*-----------------------------------------------------------------------
  *     writef          Outputs buffer to file, fixed length records
  *-----------------------------------------------------------------------
  */
 int    writef ( fd, p buf, recsiz, bytes )
                                  /* RC - TRUE, successful write */
                                  /*      FALSE, failed to write buffer */
        int     fd ;              /* I    file descriptor */
unsigned char   *p buf ;          /* I    pointer to buffer to be output */
        int     recsiz ;          /* I    size of records to use in write */
        int     bytes ;           /* I    number of bytes to write out */
 {
 register int   rc ,              /* Return code from write call, #bytes output */
                rsiz ,            /* Register copy of recsiz */
                b rem ;           /* Bytes remaining to be written from buffer */
register unsigned char  *pb ,     /* Register pointer to buffer */
                *pb org,*pb rec,/* Used to copy last record if b rem < recsiz */

        pb = p buf ;              /* Get ptr to buffer in register variable */
        b rem = bytes ,           /* Bytes remaining to be written, initialize */
        rsiz = recsiz ,           /* Register copy of record size */

        while (b rem >= rsiz) {
          rc = write (fd, pb, rsiz) ,
          if ( rc < 0 ) {
            fprintf(stderr,"writef, ERROR on write  %d bytes left\n"  b rem)
            return (FALSE) ,
          }
HSC000$DUA1:[OCR.PGM]FIXQQ.C;10          6 APR 1987 15:10          Page 5
        b_rem -= rc ; pb += rc ;
      }
      if ( b_rem ) {           /* Have to pad last record (odd size) */
        p_buf = NULL ;
        if ( !getmem(&p_buf,rsiz) ) {
          fprintf ( stderr,
              "writef, calloc error, %d bytes not output\n", b_rem ) ;
          return (FALSE) ;
        }
        pb_rec = p_buf ;       /* Pointer to new buffer space from getmem */
        pb_org = pb ;          /* Pointer to start of remaining bytes */
        while ( b_rem ) {
          b_rem-- ; *pb_rec++ = *pb_org++ ;
        }
        rc = write (fd, p_buf, rsiz) ;
        cfree (p_buf) ;
        if ( rc < 0 ) {
          fprintf ( stderr,
              "writef, ERROR on last record, %d bytes left\n", b_rem ) ;
          return (FALSE) ;
        }
      }
      return (TRUE) ;
}
```

```
HSC000$DUA1:[OCR.PGM]FIXED.FDL;1              18-MAR-1987 17:16          Page 1

IDENT    "17-MAR-1987 16:14:47   VAX-11 FDL Editor"

RECORD
        CARRIAGE_CONTROL            none
        FORMAT                      fixed
        SIZE                        16384
```

Appendix D

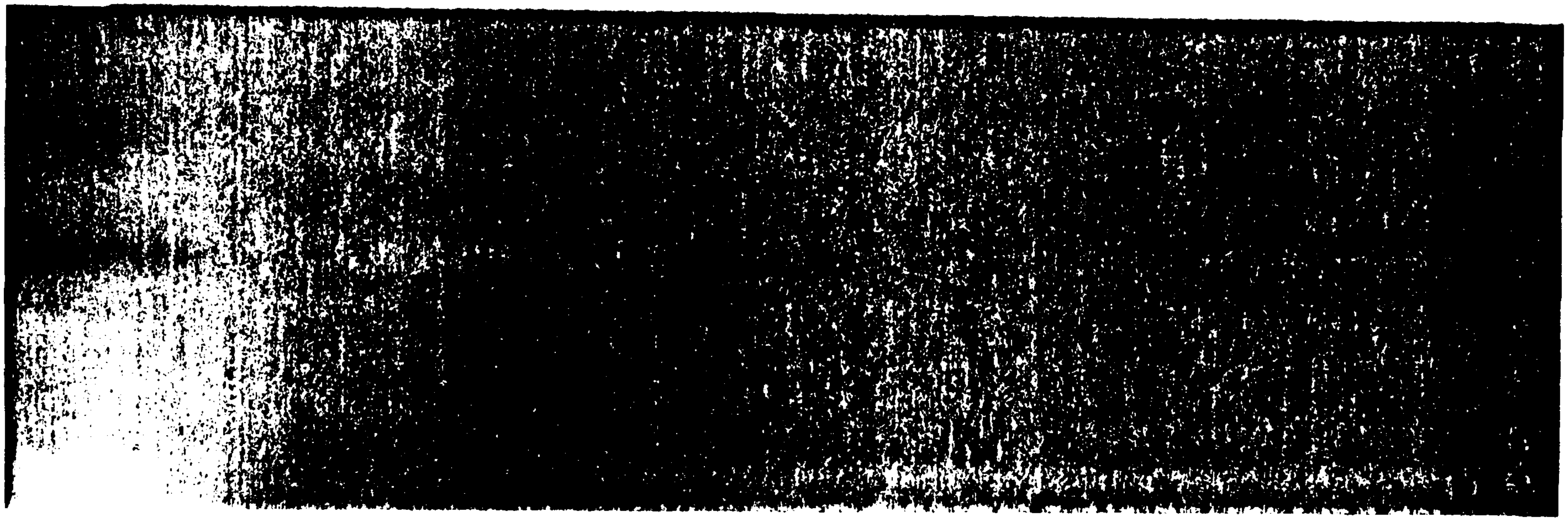
Processing Stages

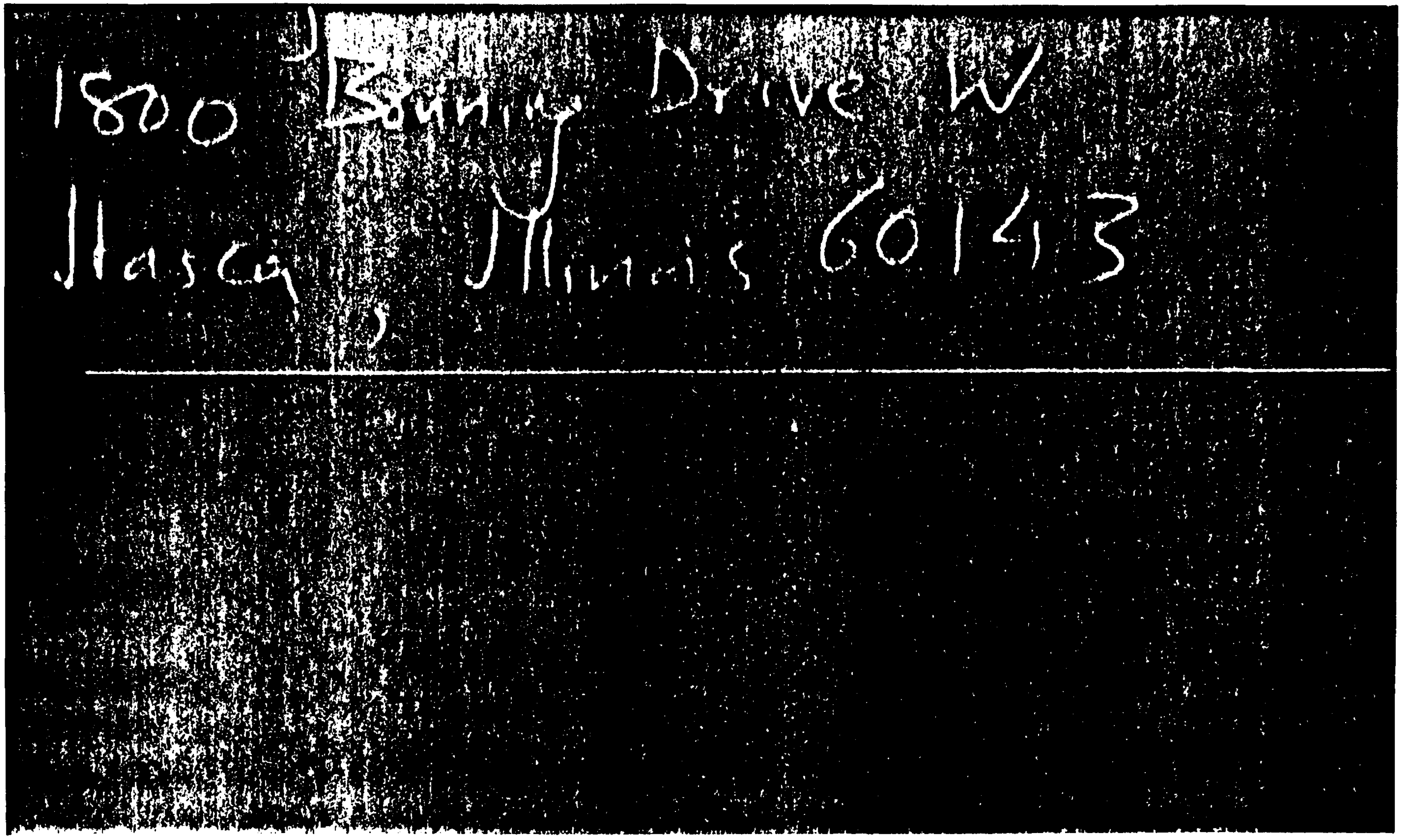
Processing Description

The first image shows the original grey-level image displayed in grey tones. To threshold this image, a threshold level is determined by filtering out the high-frequency address information with a morphological operation to give an approximation to the background, and subtracting this background from the original image. The resultant image possesses only foreground (address) information (the second image). The third image shows this thresholded image. To determine address lines, a line density image is created—all pixels in the thresholded image are shifted to the right, and the resultant peaks are separated to give line locations, as shown in the fourth image. The red bar at the right is used to separate peaks from eachother. Five lines are detected in this image: the name, title, and business lines (not shown), the *1800 Boeing Drive W* line, and the *Itasca, Illinois 60143* line. The fifth image shows the last address line. As a simplistic first guess, the ZIP Code is assumed to be on the last line. To quickly and easily separate characters, a the character density image is created—all pixels in the last line are shifted down, and the resultant peaks are separated to give character locations using a method similar to the line separating algorithm. The sixth and seventh images, continuations of the fifth, show further steps in the character segmentation algorithm. The result of the separation algorithm, along with concavity features of the lastline, are shown on the very bottom of the seventh image. The eighth image shows an enlargement of the separated ZIP Code block with colored concavity features.

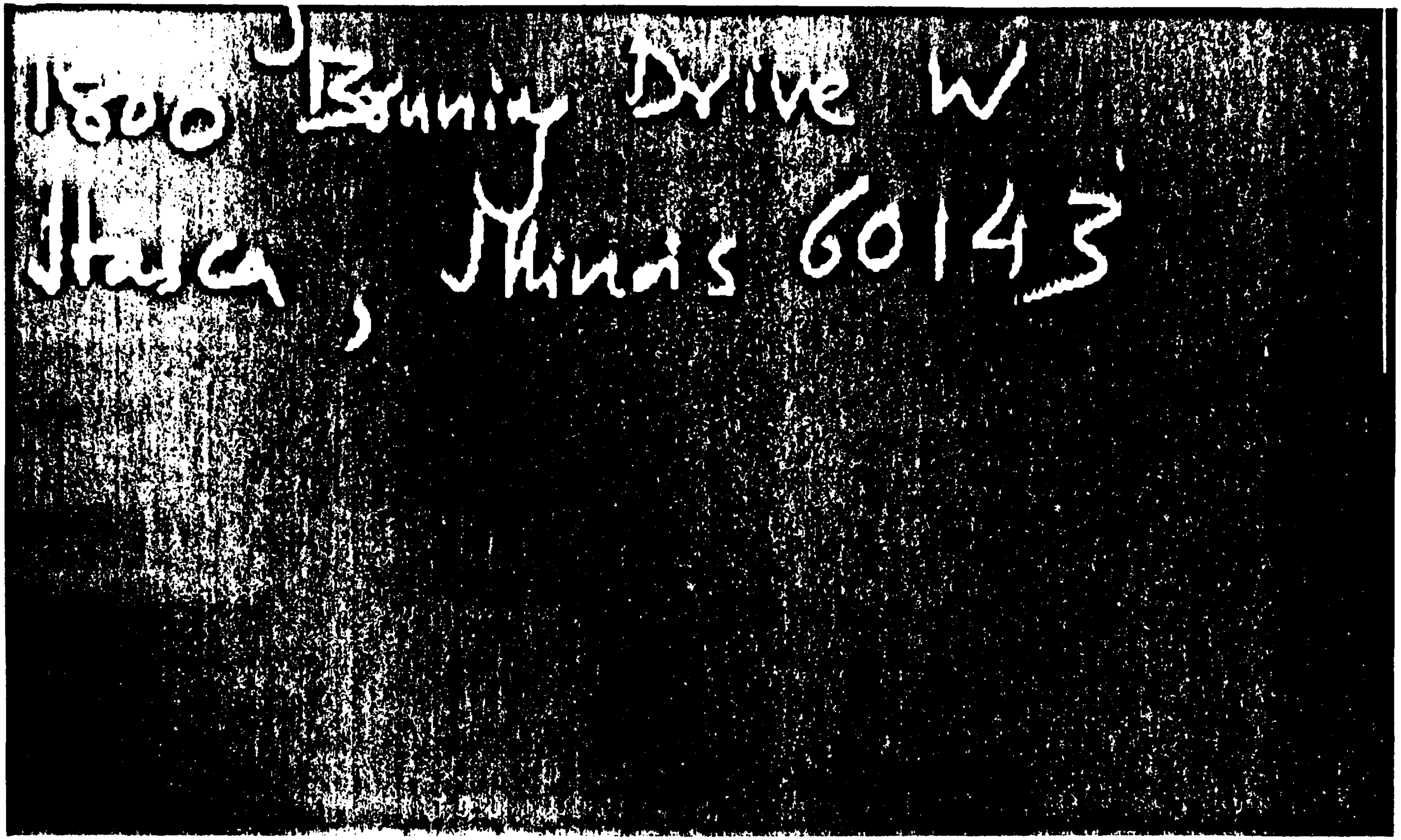
[illegible]

1820 Bonning Drive W
Itasca, Illinois 60143

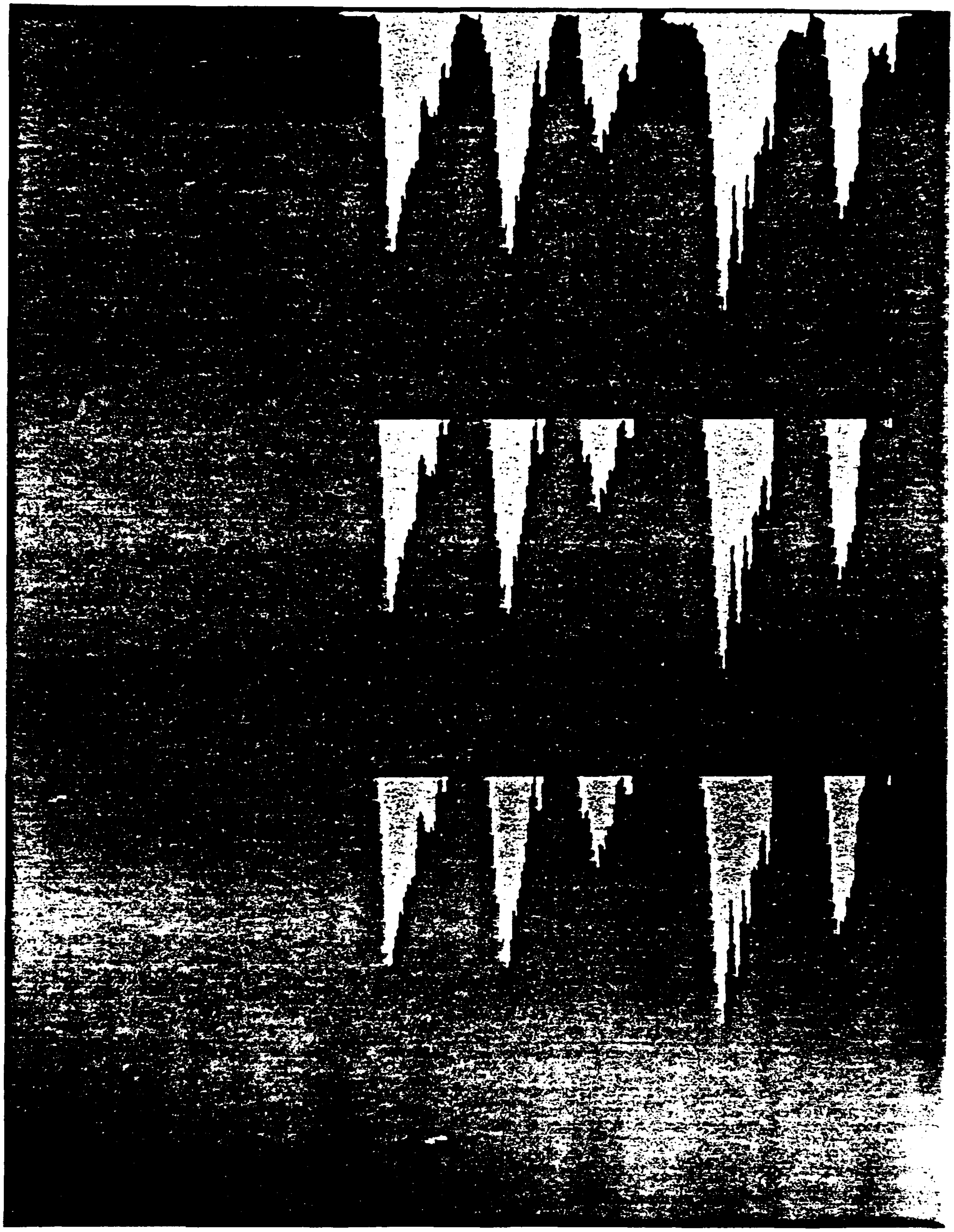




1800 Bunny Drive W
Itasca, Illinois 60143

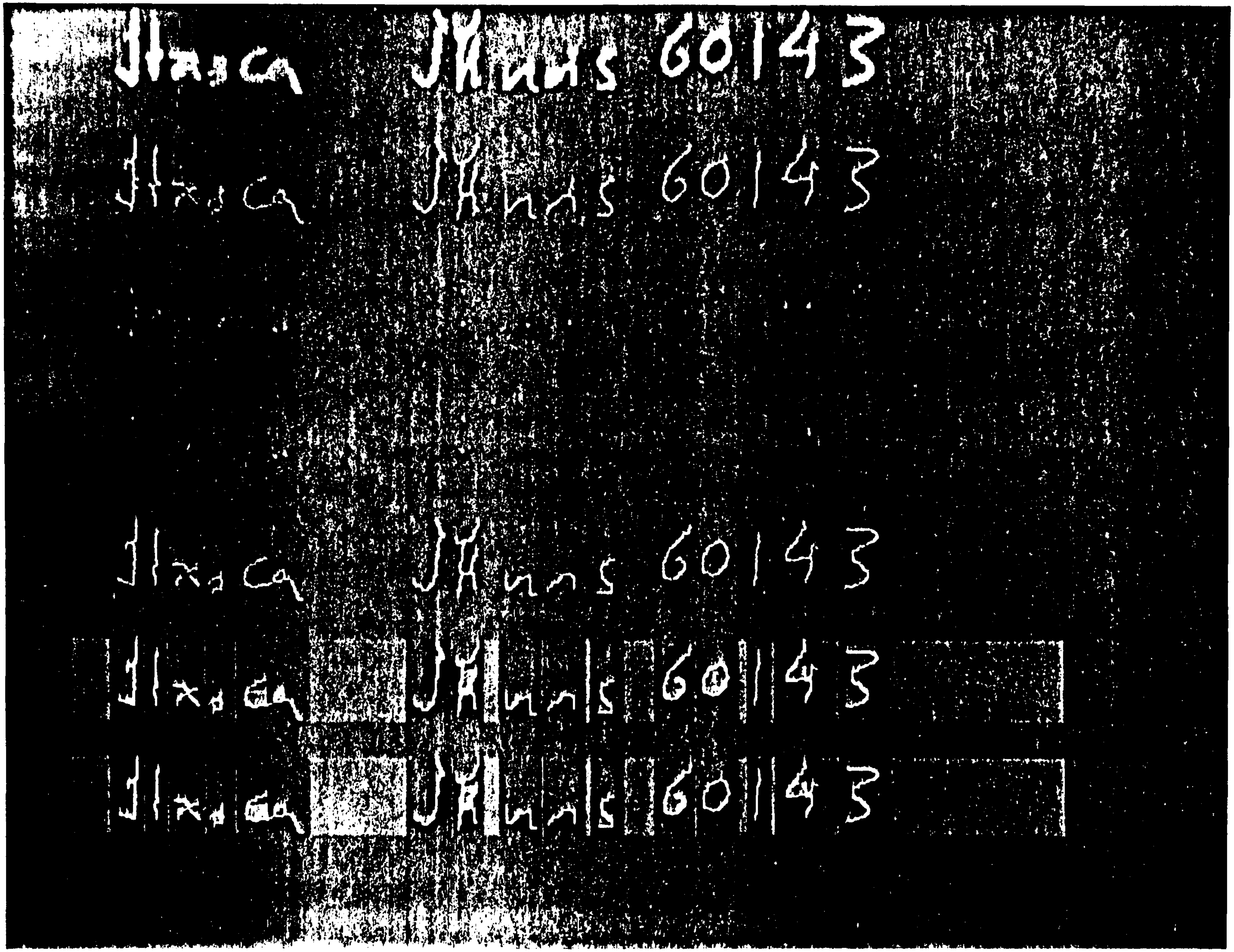


1800 Bunny Drive W
Itasca, Illinois 60143



Itasca Illinois 60143
Itasca Illinois 60143

Itasca Illinois 60143
Itasca Illinois 60143
Itasca Illinois 60143
Itasca Illinois 60143
Itasca Illinois 60143
Itasca Illinois 60143


JHxCa JHuns 60143
JHxCa JHuns 60143
JHxCa JHuns 60143
JHxCa JHuns 60143
JHxCa JHuns 60143